UNIVERSIDADE FEDERAL DE MINAS GERAIS – UFMG
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

FERNANDO ESQUÍRIO TORRES

DESENVOLVIMENTO DE UM SISTEMA DE EMULAÇÃO DE *SINGLE EVENT UPSETS* EM DISPOSITIVOS COTS BASEADO NA METODOLOGIA *CODE EMULATING UPSETS*

Belo Horizonte
Fevereiro 2013

FERNANDO ESQUÍRIO TORRES

DESENVOLVIMENTO DE UM SISTEMA DE EMULAÇÃO DE *SINGLE EVENT UPSETS* EM DISPOSITIVOS COTS BASEADO NA METODOLOGIA *CODE EMULATING UPSETS*

Dissertação submetida ao Curso de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Sistemas de Computação e Telecomunicações.

Linha de Pesquisa: Microeletrônica e Microssistemas (MeMSs)

Orientador: Prof. Dr. Ricardo de Oliveira Duarte

Belo Horizonte
Escola de Engenharia da UFMG
Fevereiro 2013

Desenvolvimento de um Sistema de Emulação de *Single Event Upsets* em Dispositivos COTS Baseado na Metodologia *Code Emulating Upsets*

Fernando Esquírio Torres

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Belo Horizonte, fevereiro de 2013.

_____

Prof. Dr. Ricardo de Oliveira Duarte (Orientador)

DELT – UFMG

_____

Prof. Dr. Frank Sill Torres

DELT – UFMG

_____

Prof. Dr. Júlio Cezar David de Melo

DELT – UFMG

"O cientista que consome horas a fio, durante anos seguidos, no silêncio dos laboratórios de cálculo e de pesquisa, debruçado sobre microscópios ou manipulando computadores, serve, de fato, à humanidade e contribui para a solução dos seus problemas". (Pequena Enciclopédia de Moral e Civismo Ministério da Educação - 1967)

“Aos meus pais José Braz Torres e Myriam Esquírio Torres, que são grandes exemplos de honestidade e perseverança. Vocês sempre estiveram ao meu lado nessa caminhada, demonstrando muito afeto e carinho. Obrigado por vocês serem os melhores amigos que eu, como filho, poderia ter.”

AGRADECIMENTOS

A Deus, um ser supremo, de inteligência infinita, criador do universo que penetra o espaço ilimitado; por me guiar até este momento com saúde e perseverança.

Ao professor Dr. Ricardo de Oliveira Duarte, do Departamento de Engenharia Eletrônica, da Escola de Engenharia da UFMG, empenhado em um grande esforço em desenvolver, ao máximo, as minhas potencialidades, possibilitando-me a execução deste trabalho. Um educador e orientador exemplar que me fez crescer, tanto profissional como pessoalmente, de quem recebi todo incentivo necessário, nos momentos de decisão. A quem agradeço muito pela convivência durante todo esse período.

Ao professor Dr. Julio Cesar de Melo, chefe do laboratório L.S.I., do Departamento de Engenharia Eletrônica, Escola de Engenharia da UFMG, pelo empenho e confiança em disponibilizar o Laboratório e Equipamentos necessários nas horas em que deles necessitei, contribuindo assim, para o sucesso deste trabalho.

A minha família, principalmente, aos meus pais pelo amor incondicional e dedicação aos filhos, pelo grande apoio a tudo que fiz até hoje, pelo exemplo de seres humanos e pela herança do saber que nos legaram desde criança.

Aos meus amigos pessoais, pelo carinho, atenção, incentivo e apoio, cada um a seu modo, pela torcida por esta vitória.

Ao Thalles e ao Wagno, companheiros do laboratório L.S.I., pela ajuda no desenvolvimento da ferramenta apresentada nesse trabalho e na prototipagem das placas de circuito impresso.

Ao programa CAPES-REUNI e ao programa Uniespaço da AEB, entidades financiadoras.

À Universidade Federal de Minas Gerais (UFMG), pela oportunidade de ser mais útil a mim mesmo, a Deus, à família e à sociedade, através dos conhecimentos adquiridos nessa universidade.

Por fim, a todas as pessoas que, direta ou indiretamente, contribuíram para esta conquista desse objetivo. Deixo aqui meu muito obrigado, e que de alguma forma eu possa retribuir futuramente.

RESUMO

Esse trabalho consiste no desenvolvimento de um método e uma ferramenta, que possibilita a emulação do comportamento de falhas do tipo *Single Event Upset* em aplicações gravadas em microcontroladores COTS, destinados a funcionar em satélites científicos projetados para operar em baixa órbita terrestre. O método apresentado nesse trabalho tem como finalidade validar sistemas de computação desenvolvidos com técnicas de tolerância a falhas, os quais foram construídos a partir desses dispositivos. A ferramenta desenvolvida nesse trabalho, denominada PORTHES, baseia-se na reprodução da metodologia conhecida como *Code Emulating Upsets*, que permite emular o comportamento de falhas, por meio de um trecho de código inserido em uma Rotina de Tratamento de Interrupção dentro do *firmware* da aplicação gravada no microcontrolador, que proporciona simular *upsets* no conjunto dispositivo-aplicação testado, reproduzindo os efeitos da radiação ionizante em um ambiente espacial de baixa órbita terrestre.

O sistema PORTHES foi desenvolvido para ser uma ferramenta portátil e foi construído com equipamentos de baixo custo. Além disso, o sistema não requer a construção de uma placa de *hardware* específica para validar um microcontrolador a SEUs. O sistema é controlado por uma interface gráfica, executando em um computador, permitindo o controle do processo experimental, a geração de falhas, a emulação do comportamento de falhas e a análise dos dados coletados. O PORTHES serve para investigar o comportamento de aplicações gravadas em microcontroladores COTS na presença de falhas e, também, para ser empregado na validação de sistemas desenvolvidos com esses dispositivos e técnicas de tolerância a falhas, sem a necessidade de submeter o conjunto a um processo de exposição à radiação ionizante.

As sessões de experimentais indicaram que o sistema PORTHES pode ser utilizado como uma ferramenta para a emulação de falhas do tipo *Single Event Upset* em aplicações gravadas em microcontroladores COTS, sendo capaz de realizar ensaios como se o dispositivo-aplicação estivesse exposto à radiação ionizante do ambiente espacial de baixa órbita terrestre.

Palavras-chave: Metodologia de código para emulação de *upsets*, *single event upset*, métodos de validação de microcontroladores COTS, taxa de *soft error*, falhas transientes.

ABSTRACT

This work presents a method and a tool that enable the emulation of the behavior of Single Event Upset faults in applications running on microcontrollers COTS installed in scientific satellites, which are designed to operate in low Earth's orbit. The method presented in this work aims to validate computational systems developed with techniques for fault tolerance, which are built from these devices. The tool developed in this work, called PORTHES, is based on a fault emulating methodology known as Code Emulating Upsets, that allows to emulate the behavior of faults through a piece of code inserted as an Interrupt Service Routine into the firmware of the application running in the microcontroller, which allows the simulation of upsets in the device-application under test, reproducing the effects of ionizing radiation of low Earth orbit in space environment.

The PORTHES system was developed to be a portable tool, and was constructed with low cost equipment. Moreover, the system doesn't need to build a hardware-specific board to validate microcontrollers to SEUs. The system is controlled by a graphical user interface that is running on a computer. The graphical user interface allows to configure the variables and to control the actions used in the experimental process, the fault generation, the emulation of faults behavior and the data analysis. The PORTHES is used to investigate the operation of applications running on COTS microcontrollers in the presence of faults and also it is useful to be employed to validate systems developed with these devices and fault tolerance techniques, without need to submit the system to a process of ionizing radiation exposure. The experimental sessions indicated that the system PORTHES can be used as a tool for emulation of Single Event Upsets faults in applications running on microcontrollers COTS and the system may be able to execute tests as if the device-application was exposed to ionizing radiation of low Earth orbit in space environment.

Keywords: Code Emulating Upsets methodology, single event upset, methods validation of COTS microcontrollers, soft error rate, transient faults.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA ABREVIATURA

ARIS	–	*Architectures for Robust and complex Integrated Systems*
Bps	–	*bits per second*
CERN	–	*European Organization for Nuclear Research*
CEU	–	*Code Emulating Upset*
CI	–	Circuito Integrado
CMOS	–	*Complementary Metal-Oxide-Semiconductor*
COTS	–	*Commodity-Off-The-Shelf*
DAQ	–	*Data Acquisition*
DD	–	*Displacement Damage* ou Danos por Deslocamento
DFT	–	*Discrete Fourier Transformer* ou Transformada Discreta de Fourier
DS	–	Disponibilidade do sistema
DUT	–	*Device Under Test* ou Dispositivo em Teste
HDL	–	*Hardware Description Language* ou Linguagem de Descrição de *Hardware*
ID	–	Número de identificação
IG	–	Interface com elementos gráficos
IRPS	–	*International Reliability Physics Symposium*
LHC	–	*Large Hadron Collider*
MCU	–	Microcontrolador
MCUp	–	*Multiple-Cell Upset*
MMI	–	*Memory Mapped Interface*
NIEL	–	*Non-ionizing Energy Loss*
PC	–	*Program Counter* ou Contador de Programa
PORTHES	–	*Portable Testbed for Harsh Environment of Single Event Upset*
SAA	–	*South Atlantic Anomaly* ou Anomalia do Atlântico Sul
SDATF	–	Sistema de Determinação de Atitude com Tolerância a Falhas
SE	–	*Soft Error*
SEE	–	*Single Event Effect*
SER	–	*Soft Error Rate*
SET	–	*Single Event Transient*
SEU	–	*Single Event Upset*
SFR	–	*Special Function Register* ou Registrador de Função Especial
THESIC	–	*Testbed for Harsh Environment Studies on Integrated Circuits*
TID	–	*Total Ionizing Dose* ou Dose de Ionização Total
TIMA	–	*Techniques of Informatics and Microelectronics for integrated systems Architecture*
TSR	–	*Trap Service Routine*

SUMÁRIO

Capítulo 1 - INTRODUÇÃO

Sistemas computacionais são constituídos por circuitos integrados COTS (*Commercial-Off-The-Shelf*) ou dispositivos semicondutores. Esses sistemas embarcados em satélites estão sujeitos a apresentarem falhas no seu funcionamento, provocadas pela colisão com partículas ionizadas [1, p. 28]. Tais partículas podem ser originalmente de duas fontes, íons pesados presentes no ambiente espacial, ou partículas alfa produzidas por isótopos radioativos, como também criadas por prótons ou nêutrons de alta energia, que interagem com o silício e demais impurezas presentes nos componentes eletrônicos CMOS (*Complementary Metal-Oxide-Semiconductor*) [1, p. 28]. Esse fenômeno, conhecido como *Single Event Effects* (SEEs), ocorre frequentemente em quase todos os sistemas computacionais compostos, exclusivamente, por circuitos integrados COTS que compõem satélites de baixa órbita, quando passam pelo cinturão de Van Allen, mais especificamente pelas zonas das auroras do norte e do sul (Boreal e Austral) e da anomalia do atlântico sul [2]. Os SEEs são causados pelo choque das partículas ionizadas com regiões sensíveis do material semicondutor dos dispositivos integrados [3]. Dodd e Massengill, em [3] e Kastensmidt, em [4] explicam que essas regiões são vizinhanças das junções dos drenos dos transistores reversamente polarizados. Se uma partícula radioativa de alta energia atravessa a junção *pn* de um transistor CMOS no estado *off* (Figura 1), um curto circuito momentâneo pode ser criado entre o substrato e o terminal de dreno. O excesso de carga causado pela trilha ionizante é reunido na camada de depleção e produz elétrons que podem acabar sendo armazenados nos poços dos elementos de memória [1, pp. 3-4]. Entretanto, se esse curto não acontece, a quantidade de carga armazenada, resultante da penetração dessa partícula no material, pode ser suficiente para exceder certo valor crítico (Q_{crit}) de carga admissível, criando, assim, um pequeno pulso de corrente transiente será produzido por um período de tempo δ, como mostrado na Figura 2, até que o efeito da carga desapareça, seja por recombinação das cargas ou através de um caminho livre de V_{dd} ou *ground*. Nesse último caso, a lógica do dispositivo continuará a funcionar normalmente assim como fora projetado sem causar erros no sistema. Entretanto, se tal recombinação de cargas não desaparecer dentro de um período de tempo, no qual certo dado está sendo computado ou armazenado por qualquer elemento de memória, poderá produzir um SEE.

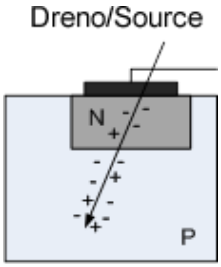


Figura 1. Trilha de ionização.

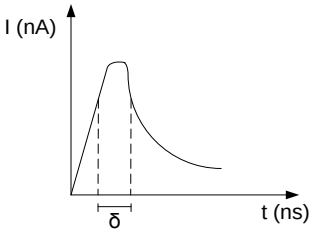


Figura 2. Pulso de corrente transiente.

Os SEEs podem ocorrer em um circuito combinacional ou em uma célula de memória. O primeiro é classificado como *Single Event Transient* (SET) [1, p. 77] e o segundo como *Single Event Upsets* (SEU) [5]. Caso o SEE se propagar como um erro para as saídas observadas do circuito integrado acontecerá então um *Soft Error* (SE) [5]. SEs são eventos nos quais dados são corrompidos sem danos permanentes no dispositivo, portanto são considerados erros temporários [1, p. 2]. Esses erros temporários podem resultar em perdas de dados, detectáveis ou não, mau funcionamento do dispositivo e até defeitos no sistema [1, p. 1]. De acordo com Ziegler e Lanford [6], a taxa de falhas em dispositivos eletrônicos CMOS, operando em altitudes de aeronaves espaciais, é cerca de 100 vezes maior do que ao nível do mar. O termo SEU é frequentemente usado na literatura como sinônimo de SEEs [1, p. 2]. Devido às dificuldades em detectar os SETs e o fato deles poderem se transformar em SEUs, segundo as estimativas probabilísticas relatadas em [7] e [8], será usado, daqui por diante, com o propósito de simplificação de nomenclaturas, o termo SEU para representar qualquer ocorrência de SEEs. Assim, um *Single Event Upset* (SEU) é entendido como uma mudança de estado em um elemento de memória resultante da recombinação de cargas em regiões sensíveis dos transistores CMOS, perceptíveis ou não ao nível da aplicação como um erro. Essa mudança é causada por íons ou radiação eletromagnética que colidiram com essas regiões de transistores em elementos de memória de circuitos integrados (CIs), tais como microprocessadores, microcontroladores ou qualquer outro dispositivo semicondutor [5]. Como um *bit* de memória está associado a um nível de tensão, essa mudança de estado provoca alteração de um dado em uma célula de memória.

1.1. Objetivo

O objetivo deste projeto é desenvolver um método ou ferramenta para emular os efeitos de SEUs em microcontroladores COTS programados com *firmware* voltados à operação em baixa órbita terrestre e com isso validar um sistema tolerante a falhas com o método ou a ferramenta desenvolvida de emulação do comportamento de falhas em circuitos baseados em microcontroladores COTS.

1.2. Justificativa e Relevância

Especificamente em relação à área de tolerância a falhas, sabe-se que ela é multidisciplinar e a capacidade de integração a outras áreas é evidente, pois através de técnicas específicas possibilita agregar atributos de confiabilidade, disponibilidade e segurança a qualquer tipo de sistema. A validação de sistemas desenvolvidos com tais técnicas, entretanto, é extremamente dependente da aplicação e do ambiente no qual se encontrará em funcionamento.

Mediante o desenvolvimento tecnológico, a relevância deste projeto baseia-se na contribuição prática de um conhecimento específico, essencial ao sucesso na validação de sistemas de computação com técnicas de tolerância a falhas para aplicações espaciais.

A relevância deste projeto face ao desenvolvimento científico fundamenta-se na possibilidade de apresentar resultados significativos com soluções de baixo custo e rápidas, ainda não divulgadas cientificamente, para validar sistemas sujeitos aos efeitos das radiações cósmicas do ambiente espacial. Os resultados desse projeto possuem potencial econômico e aplicação social para o desenvolvimento do programa espacial brasileiro.

1.3. Estrutura do Trabalho

Esse trabalho está organizado em seis capítulos. A contextualização do problema das falhas provocadas por efeito da radiação em sistemas microprocessados foi realizada no capitulo 1. A revisão teórica é apresentada no capítulo 2, através de um levantando de fatos históricos, os quais comprovam a existência do problema, da apresentação de conceitos relevantes ao desenvolvimento do trabalho, da demonstração das principais técnicas de validação existentes e do detalhamento de um trabalho de referência que motivou a pesquisa. O capítulo 3 apresenta o desenvolvimento do trabalho proposto. O sistema de determinação

de atitude com tolerância a falhas é descrito no capítulo 4. Os testes e resultados são discorridos no capítulo 5, onde são expostos detalhes dos experimentos realizados e são discutidos e analisados os resultados obtidos. Finalmente, no Capítulo 6 são expostas as conclusões do trabalho e as propostas para trabalhos futuros.

Capítulo 2 - REFERENCIAL TEÓRICO

2.1. Histórico de *Soft Errors*

Soft Errors induzidos por radiação e seus efeitos no funcionamento em circuitos eletrônicos vem se tornando um dos problemas mais desafiadores, que impactam na confiabilidade dos sistemas eletrônicos modernos. Diversos estudos têm sido publicados nas últimas décadas (a partir da década de 60), expondo fatos reais ao longo da história [9, pp. vii-viii]. O primeiro relato, evidenciando o efeito radioativo, que ocasionou um mau funcionamento na operação de um circuito eletrônico, ocorreu em 1962. Na época, o recém-lançado satélite de comunicações *Telstar* apresentou falhas quando operava em alta altitude [9, pp. vii-viii].

Em 1975, Binder et al. reportaram "anomalias", que ocorreram em circuitos eletrônicos de um satélite durante um período operacional de 17 anos [1, pp. 3-7]. De acordo com Binder et al., as mudanças de estado nos circuitos flip-flop eram a comprovação dessas "anomalias". Essas mudanças de estado foram provocadas por acúmulo crítico de cargas armazenadas em capacitores das junções de transistores usados na fabricação dos circuitos integrados empregados nos sistemas eletrônicos do satélite. Esse acúmulo de cargas era produzido por trilhas de ionização, geradas pelas partículas de raios cósmicos de alta energia, que colidiam com regiões sensíveis do material semicondutor dos dispositivos eletrônicos do satélite [1, pp. 3-7].

May e Woods da Intel, em 1978, publicaram um relatório expondo a presença de *soft errors* em *chips* de memória fabricados pela empresa [10, pp. 2-3]. O estudo de May e Woods concluiu que os *soft errors* foram causados por partículas alfa, emitidas pelo decaimento dos elementos radioativos urânio e tório, os quais contaminaram o material do encapsulamento no processo de fabricação dos *chips* de memória [10, pp. 2-3]. O relatório publicado no *International Reliability Physics Symposium* (IRPS) definiu que "*soft errors*" são efeitos aleatórios causados por radiação, não recorrentes e aconteciam nos elementos de memória [1, pp. 3-7]. Os autores apresentaram o conceito de carga crítica (Q_{crit}), que é o valor da carga máxima nos capacitores das junções dreno substrato dos transistores necessário para evitar que um *soft error* aconteça no circuito [10, pp. 2-3]. Quanto maior a Q_{crit}, menor a ocorrência de *soft errors*. Entretanto para aumentar a carga crítica máxima para reduzir o efeito de *soft errors* demandaria construir transistores mais lentos. O avanço da tecnologia de fabricação de

chips permite a construção de transistores cada vez menores, mais rápidos e com tensão de alimentação menor, diminuindo, assim, a quantidade máxima de carga crítica, aumentando a preocupação com a ocorrência de *soft errors* em circuitos integrados.

Ziegler e Lanford, em 1979, presumiram a possibilidade de raios cósmicos induzirem o surgimento de *soft errors*, tanto no nível terrestre quanto em aeronaves viajando a certas altitudes, e que a quantidade de erros aumentaria à medida que a altitude aumentasse [10, pp. 2-3]. De acordo com a teoria elaborada por Ziegler e Lanford, raios cósmicos podem interagir com o material do *chip*, causando a fragmentação do núcleo de silício. Esses fragmentos poderiam induzir rajadas de cargas elétricas, resultando em *soft errors* [1, pp. 3-7]. Porém a previsão de Ziegler e Lanford foi tratada com ceticismo, devido à dificuldade, na época, de isolar os erros provocados por raios cósmicos [10, pp. 2-7]. Em 1984, a IBM validou essa hipótese, reportando erros devidos a raios cósmicos em dados coletados, a partir de registros de reparação de um computador da empresa [10, pp. 2-7].

Em 1995, Baumann et al. da *Texas Instruments* apresentaram um estudo no qual observaram uma nova fonte causadora de *soft errors*, os isótopos do Boro (^{10}B – 19.1% e ^{11}B – 80,1% de abundância). Esses isótopos são altamente instáveis quando expostos a nêutrons, provocando a emissão de partículas alfa. Essa descoberta levou a remoção desse elemento químico dos processos de fabricação (o BPSG, *Boro Pospho Silicate Glass*, era comumente usado na dopagem do tipo *p* das camadas intermediárias dos dispositivos semicondutores) [10, pp.2-3].

Não há dúvida de que a dificuldade em rastrear um *soft error* provocado por um raio cósmico ou por partículas ionizadas, impossibilita encontrar dados históricos com descrições sobre esse tipo de erro em sistemas comerciais [10, pp. 2-3]. Dos poucos relatos existentes até agora, pode-se citar o fenômeno observado nos servidores da Sun Microsystems, em 2000, quando *soft errors* ocorreram nos *chips* de memória SRAM do sistema, expondo a ineficiência do esquema de proteção desenvolvido para o banco de dados [10, pp. 2-3]. Em 2004, a Cypress *Semiconductor* relatou o surgimento de problemas em sistemas computacionais, provocados por *soft errors* [10, pp. 2-3]. Em 2005, a Hewlett-Packard reportou a queda de um sistema servidor 2048-CPU em *Los Alamos National Laboratory*, localizado à cerca de 7000 pés (um pouco mais de 2000 metros) acima do nível do mar, causada por frequentes colisões de raios cósmicos com a matriz de memória cache [10, pp. 2-3].

Sendo assim, os efeitos da radiação, seja natural ou produzida em laboratório, tornaram-se objetos de estudo na comunidade científica, agências espaciais e órgãos militares, a partir do momento, em que diversas pesquisas foram publicadas, expondo os problemas relacionados a esses efeitos.

2.2. Falhas Temporárias em CIs Fabricados com Tecnologia CMOS

Assim, diante do exposto, é constatado que a integridade do funcionamento dos CIs torna-se cada vez mais crítica, porque à medida que a tecnologia usada na fabricação de CIs avança, permite a construção de transistores cada vez menores e mais rápidos, onde o problema da confiabilidade face aos efeitos das partículas radioativas se torna mais crítico. Dessa forma, um crescente interesse na confiabilidade de *chips* produzidos com as tecnologias de fabricação de CIs atuais, surge como uma questão desafiadora para pesquisadores e projetistas de circuito integrados [1, pp. 1-2].

O efeito causado por um *soft error* induzido por radiação depende fortemente também da aplicação do sistema e impacta na confiabilidade de CIs fabricados com tecnologia CMOS. Por exemplo, *soft errors* geralmente não são um problema decisivo em aplicações que não demandam altos índices de confiabilidade e disponibilidade, por exemplo, computadores pessoais e telefones celulares. Porém *soft errors* são críticos para aplicações que contêm grande quantidade de memória, ou que possuem rigorosos critérios de confiabilidade, como um banco de armazenamento de dados com grande quantidade de memória ou um sistema desenvolvido para operar em satélites artificiais [1, pp. 7-10].

Até 1978, a radiação era considerada um problema na confiabilidade de dispositivos eletrônicos apenas para as aplicações espaciais, mas não para sistemas computacionais em funcionamento nas regiões da superfície terrestre em altitudes do nível do mar [1, pp. 1-2]. Sob as condições espaciais, não somente *soft errors* acontecem, mas o efeito radioativo sobre sistemas eletrônicos provoca a degradação do dispositivo, principalmente se a dose ionizante total (do inglês, *Total Ionizing Dose*, TID, Figura 3) for elevada [1, pp. 1-2]. A TID é o efeito cumulativo de exposição prolongada à radiação ionizante, que provoca a degradação do material do circuito integrado, causando a degradação de propriedades elétricas desses circuitos, como por exemplo, aumento no consumo de energia ou até perda do *chip* por efeito conhecido como Burnout [11]. Essa exposição prolongada acarreta alterações no dispositivo, que são causadas pelas cargas elétricas acumuladas induzidas por radiação, prejudicando o

funcionamento correto do circuito integrado [12]. Esse mau funcionamento pode ser irreversível e danificar permanentemente o dispositivo (*hard errors*), porém isso vai depender da quantidade de carga acumulada no circuito e do tempo de exposição do dispositivo à radiação [13].

Por sua vez, a radiação também pode provocar outro efeito conhecido como danos por deslocamento (do inglês, *Displacement Damage*, DD), que são danos ocasionados pela perda de energia de forma não ionizante (NIEL – *Non-ionizing Energy Loss*) (Figura 3), provocada por deslocamentos atômicos na estrutura cristalina do material, como o silício no caso de semicondutores [14]. Os distúrbios provocados pelos danos por deslocamento não são aleatórios e são provocados pela radiação eletromagnética emitida especialmente pelas explosões solares [15].

A respeito dos dois fenômenos causados por radiação citados, TID e DD, não serão tratados nesta dissertação. O interesse desse trabalho está nos efeitos estocásticos conhecidos como *Single Event Effects* (Figura 3), mais especificamente *Single Event Upsets* (*SEUs*). Os *SEEs* são falhas no funcionamento de sistemas construídos com dispositivos eletrônicos COTS, que ocorrem devido ao impacto de partículas ionizadas com o silício e a combinação com as demais impurezas presentes nos componentes eletrônicos CMOS, ionizando o material semicondutor densamente e podendo provocar um pulso de corrente transiente por um curto intervalo de tempo [16]. CIs estão cada vez mais sensíveis a SEEs em razão da redução das dimensões entre transistores e do crescente aumento da densidade desses componentes em um circuito integrado [16].

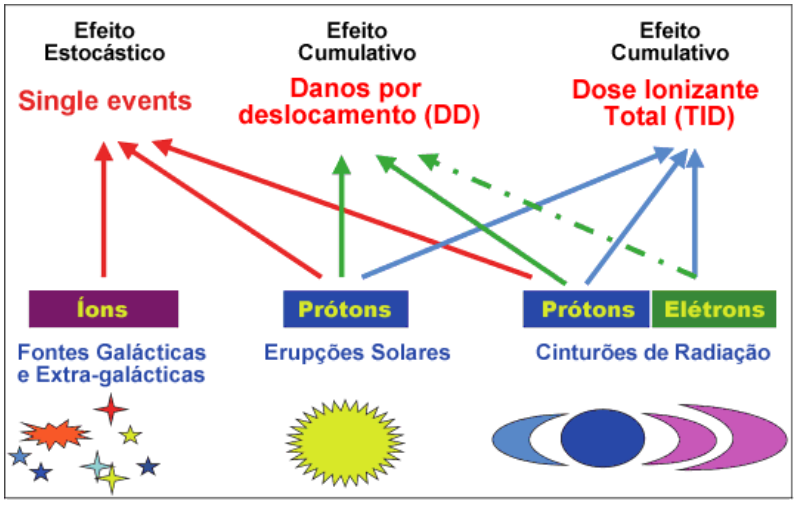


Figura 3. Relação entre as fontes de radiação espacial e a classificação dos efeitos. Figura adaptada de [17].

A Figura 3 relaciona os três efeitos citados com as fontes de radiação espacial. Como é possível observar, os SEEs podem ser provocados por diversas fontes de radiação, inclusive radiações cósmicas galácticas e extragalácticas provenientes do espaço.

2.1.1. Radiações Cósmicas

A radiação cósmica é a emissão e a propagação de energia no espaço ou através de um meio material, podendo ser por ondas ou partículas subatômicas [18]. Diferentes tipos de radiação fazem parte do ambiente espacial e podem ser classificadas em duas categorias: ionizante e não ionizante [19]. Radiações ionizantes (raios cósmicos, raios-X e radiações provenientes de matérias radioativos) produzem a emissão de elétrons, quando interagem com determinado material. Radiações não ionizantes (luz ultravioleta, ondas de rádio e micro-ondas) são incapazes de ionizar um material, com o qual elas interagem [19].

A radiação espacial interfere, com menos frequência, nos equipamentos eletrônicos projetados para funcionar na superfície terrestre, porque a Terra é protegida pela atmosfera, que é uma camada semipermeável, a qual permite a passagem de luz e calor, bloqueando raios ultravioletas [20]. Porém circuitos integrados fabricados para operar em altas altitudes sofrem com a ocorrência de efeitos indesejados em consequência da maior exposição à radiação, pois quanto mais alta a altitude em relação ao nível do mar, mais intensos são os efeitos à radiação ionizante [21].

O interesse deste trabalho concentra-se nas principais fontes de radiação ionizante provenientes do espaço, pois circuitos integrados podem apresentar *SEEs*, quando são expostos a essas fontes de radiação. A radiação espacial é constituída de partículas subatômicas (prótons, elétrons, nêutrons entre outras), que podem ser originadas de íons pesados, presentes no ambiente espacial ou de partículas alfa emitidas de isótopos radioativos. Essas partículas viajam no espaço em altíssimas velocidades e os mais rápidos podem viajar a velocidades bem próximas da velocidade da luz [22], podendo atravessar facilmente a matéria e provocar diversos efeitos sobre ela [19]. A radiação espacial pode ser proveniente de três fontes: galáctica ou raios cósmicos, solar, e cinturões de radiação (Figura 4).

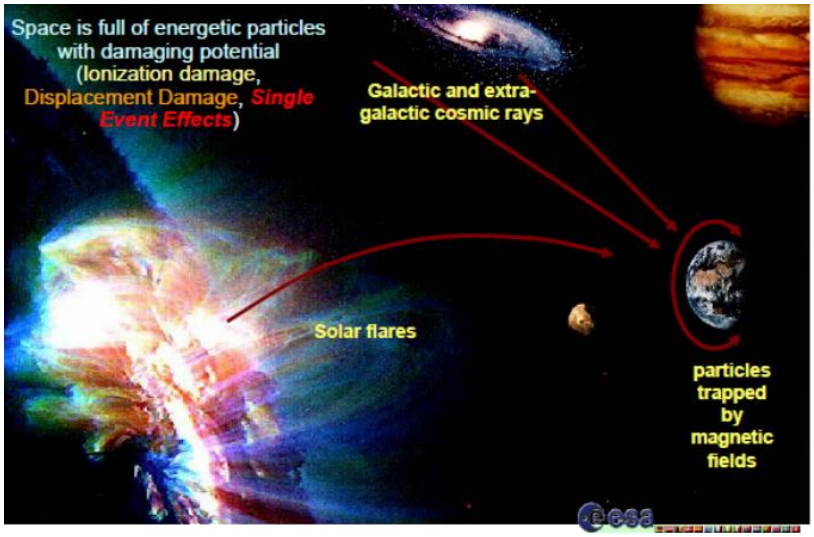


Figura 4. Fontes de radiação cósmica. Figura extraída de [23].

A radiação cósmica é composta de partículas energéticas, chamadas de raios cósmicos, pertencentes a regiões entre galáxias ou emitidas pela atividade solar [24] e foi descoberta em 1912, durante uma série de voos de balão realizados por Victor Hess [22]. O termo, raios cósmicos, não tem uma definição clara científica, sendo usado desde o início do século vinte para indicar partículas energéticas desconhecidas, que interfiram com estudos de materiais radioativos [25]. Os raios cósmicos viajam no espaço em velocidades altíssimas e com grande quantidade de energia, que, quando colidem com os átomos na atmosfera da Terra, provocam uma chuva em cascata de partículas na direção da superfície terrestre, como mostrado na Figura 5 [25]. Acredita-se que eles são produzidos e acelerados por erupções solares, supernovas e explosões de núcleos galácticos [25]. As partículas radioativas dos raios cósmicos são milhões de vezes mais energéticas do que partículas radioativas produzidos nos avançados aceleradores de partículas terrestres, como o *Large Hadron Collider* (LHC) do *European Organization for Nuclear Research* (CERN), em Genebra [22].

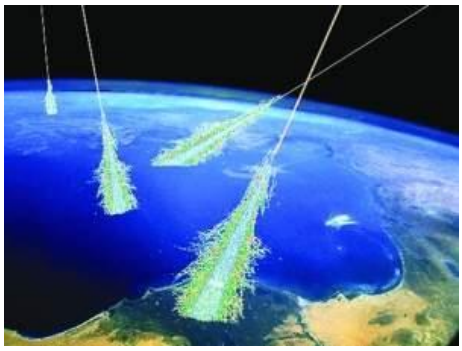


Figura 5. Chuva em cascata das partículas radioativas. Figura extraída de [26]. Imagem: Simon Swordy/University of Chicago, NASA.

Ziegler e Lanford elaboraram uma visão esquemática (Figura 6) dos raios cósmicos de alta energia interagindo com partículas na atmosfera, causando uma reação nuclear em cascata [1, p. 5]. De acordo com Ziegler e Lanford, apenas a sexta geração de partículas alcançaria o nível do mar, consistindo apenas de prótons, nêutrons, elétrons e partículas transientes, tais como múons e píons, além disso, com uma frequência muito menor do que em altas altitudes, aproximadamente uma partícula por centímetro quadrado a cada segundo [1, p. 5].

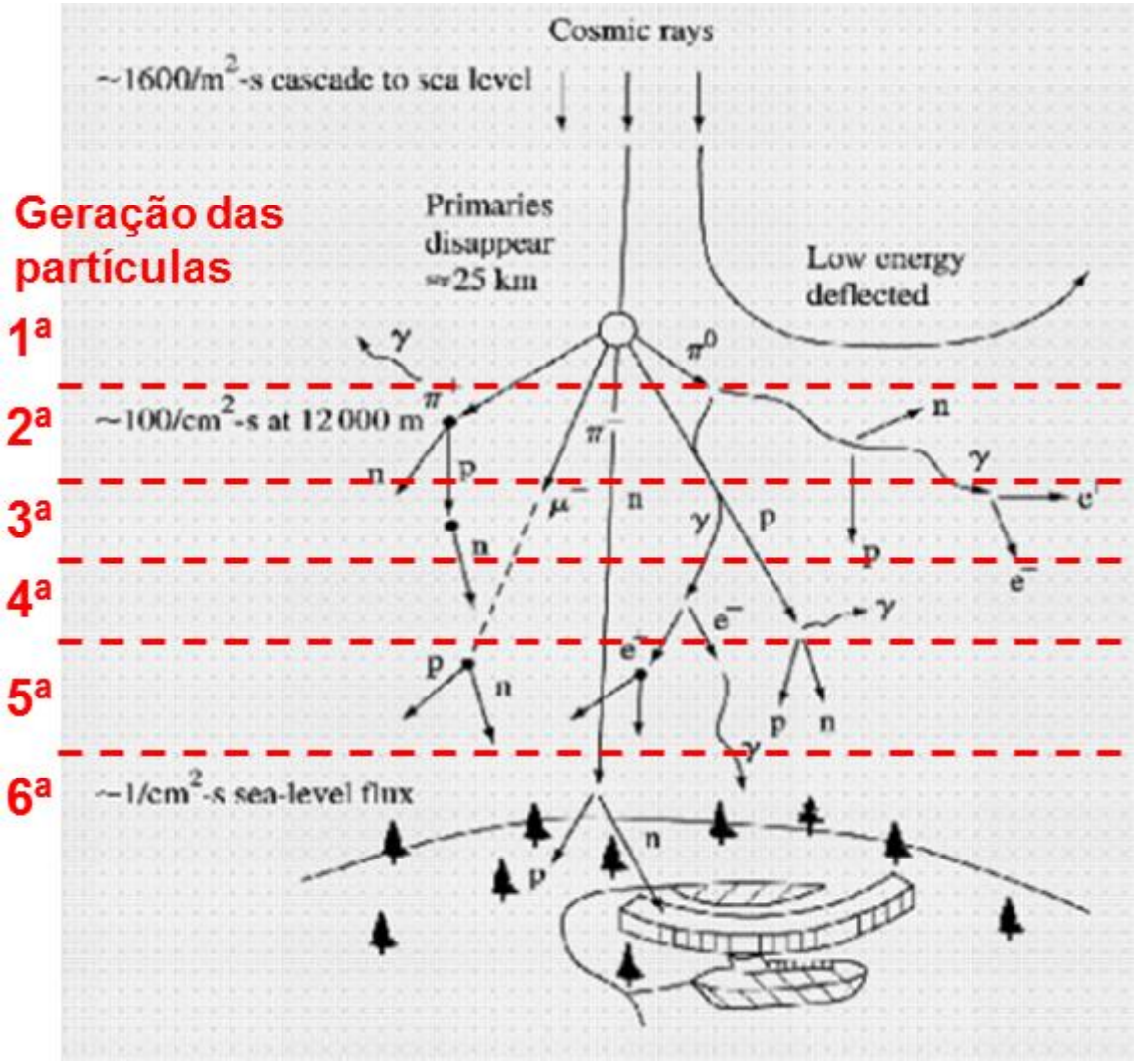


Figura 6. Visão esquemática dos raios cómicos provocando a cascata de partículas. Figura extraída de [1, p. 5].

Além da radiação cósmica, a atividade solar também colabora com a emissão de partículas radioativas no espaço. A radiação solar é a energia emitida pelo sol no espaço, podendo atingir a superfície terrestre, porém uma parte dessa radiação é absorvida pelos dois meios de proteção terrestre, o campo magnético e a atmosfera [22]. O sol possui uma atividade pontuada por eventos comuns e de natureza excepcional e possui um ciclo de onze

anos, sendo sete anos de alta atividade e quatro anos de baixa atividade [20]. A radiação emanada pelo sol normalmente pode ser produzida por erupções na sua extensão (Figura 7) e através de ventos solares (Figura 8). As erupções solares são explosões, que acontecem na superfície de sol, as quais emitem íons pesados em quantidade menor do que o fluxo de raios cósmicos, que viajam pelo sistema solar [13]. Os ventos solares são elétrons excitados, devido à alta temperatura da coroa solar, que ganham energia e conseguem escapar do campo gravitacional do sol, sendo ejetados para o espaço [13].

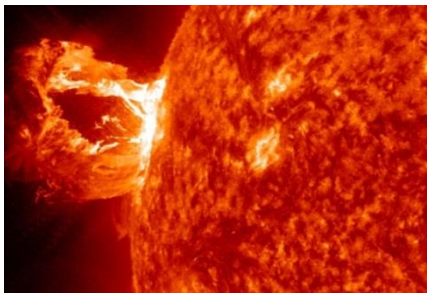


Figura 7. Erupção Solar. Figura extraída de [27].

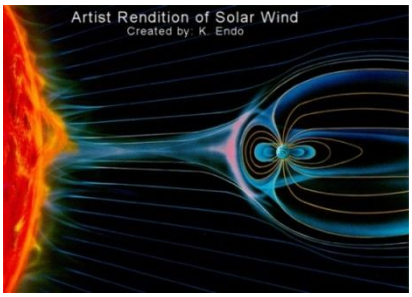


Figura 8. Vento Solar. Figura extraída de [28].

Por fim, os cinturões radioativos são regiões do espaço, descobertas nas mais altas camadas da atmosfera, mais precisamente na magnetosfera (Figura 9), repletas de partículas radioativas de energia considerável [24]. Essas regiões são formadas pela interação de partículas originadas de erupções e ventos solares e também dos raios cósmicos com o campo magnético da Terra [30], formando dois anéis, um mais interno e outro mais externo, conhecidos como cinturões de Van Allen (Figura 10). O cinturão mais interno estende-se na faixa entre cem e cinco mil quilômetros de altitude, sendo a intensidade máxima de radiação ocorrendo em média aos três mil quilômetros, esse anel mais próximo da Terra está carregado de prótons e o mais externo situa-se entre treze mil e sessenta mil quilômetros de altitude, contendo elétrons de até 7 MeV (Figura 10) [24].

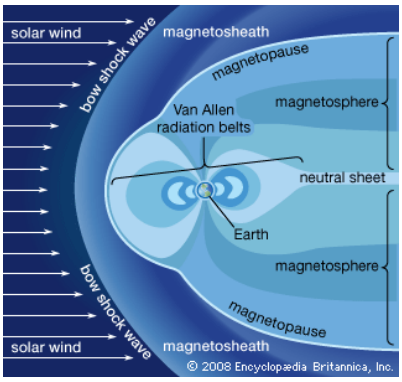


Figura 9. Esquema da magnetosfera terrestre e os cinturões de Van Allen. Figura extraída de [29].

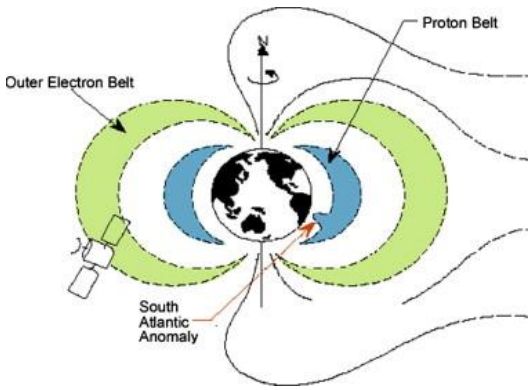


Figura 10. Cinturões de Van Allen e Anomalia do Atlântico Sul. Figura extraída de [31].

A inclinação do eixo de rotação terrestre em relação ao eixo do campo magnético terrestre influencia na distribuição do fluxo de partículas radioativas, criando uma espécie de região de depressão, que produz efeitos indesejados nos equipamentos eletrônicos de espaçonaves e satélites, que sobrevoam o sul do Brasil e o oceano atlântico [13]. Essa região é conhecida por Anomalia do Atlântico Sul (do inglês, *South Atlantic Anomaly*, SAA, Figura 10). Essa anomalia é conhecida como uma região *Vile Vortices*, onde ocorrências de falhas em dispositivos eletrônicos são frequentes [32]. A radiação proveniente da SAA é famosa por causar avarias geradas em naves espaciais [32] e, igualmente, a SAA é responsável pela maior parte de radiação incidente nos satélites de baixa órbita [24].

A Figura 11 e a Figura 12 mostram a influência do campo magnético terrestre nas distribuições de prótons e elétrons, respectivamente, na altitude de quinhentos quilômetros, mostrando a concentração dessas partículas na Anomalia do Atlântico Sul. Na Figura 11, é possível observar que o fluxo de prótons, desprezível fora da SAA, é muito significativo sobre a região do oceano atlântico sul, sendo que no centro (região vermelho da Figura 11), próximo à região sul do Brasil, os valores computados são até 100 vezes maiores do que na região de periferia da SAA (região azul da Figura 11) [13]. Já na Figura 12, o fluxo de elétrons é crítico na SAA e nas regiões onde o cinturão mais externo atinge altitudes mais baixas, próximo dos polos terrestres, nessa última região a concentração de partículas pode chegar a 100 vezes maior do que em outras regiões, como por exemplo, sobre o sul do Brasil [33].

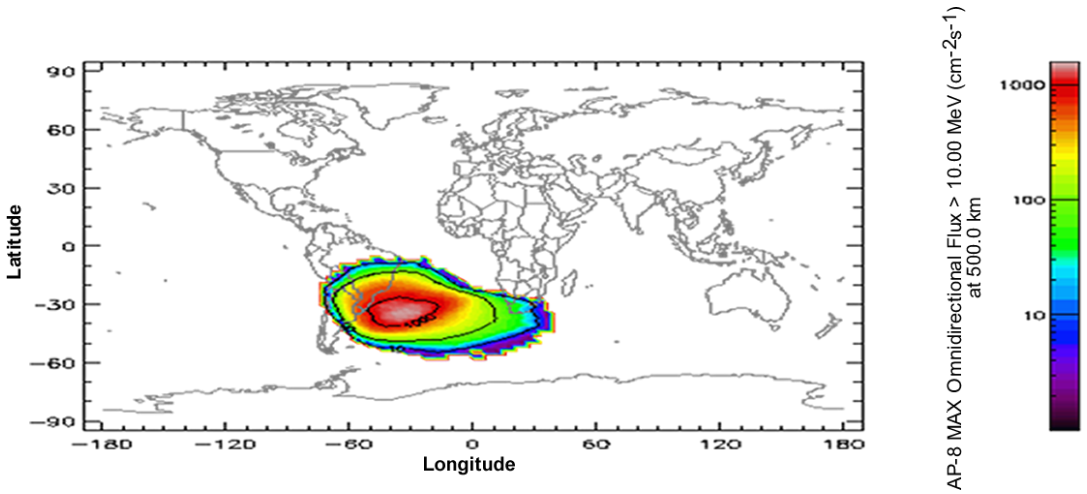


Figura 11. Fluxo de prótons utilizando o modelo AP-8 para uma altitude de 500 km na região da SAA. Figura extraída de [33].

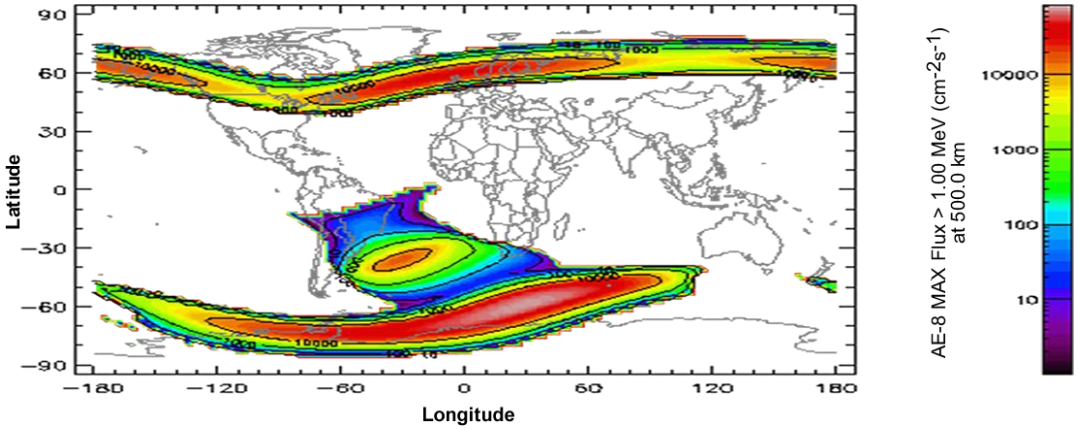


Figura 12. Fluxo de elétrons utilizando o modelo AP-8 para uma altitude de 500 km na região da SAA e nas regiões próximas dos polos terrestres. Figura extraída de [33]

2.1.2. Efeito de Partículas de Alta Energia em Circuitos Integrados CMOS

Partículas ionizadas de alta energia provocam em circuitos integrados fabricados com tecnologia CMOS efeitos indesejados no seu funcionamento. Esses efeitos são causados pelo choque dessas partículas com regiões sensíveis do material semicondutor dos dispositivos eletrônicos COTS [3]. Um dos pontos sensíveis do material semicondutor nos circuitos integrados são as vizinhanças das junções dos drenos dos transistores polarizadas reversamente. Se uma partícula radioativa de alta energia energizada atravessa a junção *pn* de um transistor CMOS em estado *off*, um curto circuito momentâneo pode ser criado entre o substrato e o terminal de dreno (Figura 13). Entretanto, se esse curto não acontece, a quantidade de carga armazenada resultante da penetração dessa partícula no material poderá ser suficiente para produzir um pequeno pulso de corrente transiente, que poderá persistir por um pequeno intervalo de tempo δ até que o efeito dessa carga desapareça, seja por recombinação com outras cargas ou através de um caminho livre para V_{dd} ou V_{ss} no circuito integrado. Nesse último caso, a lógica do dispositivo continuará funcionar de forma normal assim como fora projetado sem causar erros no sistema. Todavia, se a recombinação de cargas não desaparecer dentro de um período de tempo no qual certo dado estiver sendo computado ou armazenado por um elemento de memória, o efeito dessa partícula radioativa produzirá um fenômeno conhecido como *Single Event Effects* (SEEs), que se comporta no universo da informação como uma inversão de estado lógico booleano (*bit-flip*) [13].

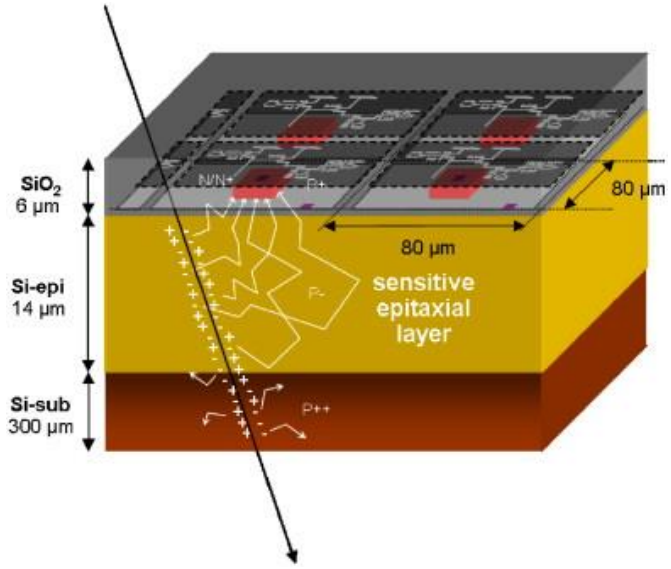


Figura 13. Elétrons produzidos pela interação de partículas radioativas com o silício. Figura extraída de [34].

Caso um SEE, produzido pelo acúmulo de carga armazenada resultante da passagem de uma partícula radioativa no material, se propagar para uma das saídas monitoradas do circuito integrado, será chamado de *soft error*. *Soft errors* podem ser classificados nas seguintes categorias [1, pp. 1-2]:

- *Single-bit upset* (SBU): é o evento que acarreta uma mudança de estado de um *bit*, causada pela colisão de uma partícula ionizada em uma célula de memória ou *latch* (Figura 14).
- *Multiple-cell upset* (MCUp): é o evento que causa uma mudança de *bits* em duas ou mais células de memória ou *latch* (Figura 15).
- *Multiple-bit upset* (MBU): é o evento que gera um *upset* em dois ou mais *bits* na mesma palavra (Figura 16).
- *Single-event transient* (SET): é o evento que produz uma falha de tensão no circuito combinacional, a qual se torna um erro de *bit*. Se esse *bit* for capturado por uma célula de memória o SET se torna um SEU.

Como mencionado anteriormente, o termo SEU é usado com frequência na literatura como sinônimo para designar todas as classificações dos efeitos SEs [1, p. 2].

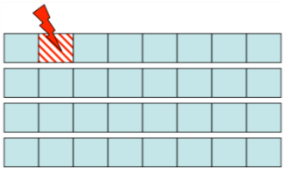


Figura 14. *Single-bit upset*. Figura adaptada de [11, pp. 9-10].

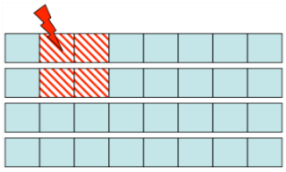


Figura 15. *Multiple-cell upset*. Figura adaptada de [11, pp. 9-10].

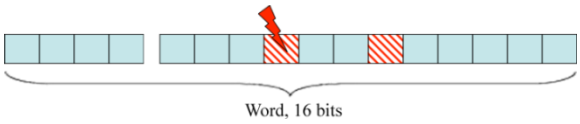


Figura 16. *Multiple-bit upset.*

2.1.3. Falha

A inversão de estado lógico booleano (*bit-flip*), provocada por um efeito singular (SEE), é qualificada como uma falha. De acordo com Avizienis et al. em [35], uma falha é um evento, que provoca um desvio no funcionamento correto de um sistema, que pode acontecer em algum instante, durante o seu período de vida. As falhas em estruturas de *hardware* ou módulos de *software*, as quais aparecem na atividade de um sistema, podem resultar defeitos por uso, imperfeições de fabricação ou interações com o ambiente externo [10, pp. 6-7]. Por exemplo, imperfeições na fabricação do *chip* de silício, *bugs* de *software* ou *bit-flips* causados por partículas ionizadas. Normalmente, conforme sua característica, as falhas podem ser classificadas como: transientes ou permanentes. Falhas transientes são falhas que aparecem durante um determinado momento e desaparecem. SEEs causados por colisões de partículas alfa ou nêutrons de alta energia são exemplos de falhas transientes. Falhas permanentes são aquelas, que se conservam por um período de tempo indefinido, até que uma ação corretiva, como a substituição do sistema, seja tomada. Por exemplo, destruição do óxido, que conduz a um funcionamento incorreto de um transistor no *chip* de silício [10, pp. 6-7].

Na Figura 17 (regiões 1 e 2), é possível visualizar a latência de propagação de uma falha em um microcontrolador, desde a colisão da partícula de alta energia no circuito CMOS, região 1 da Figura 17, até a propagação do pulso de corrente transiente (região 2 da Figura 17) capaz de provocar um *bit-flip* (efeito da falha, SEE, região 3 da Figura 17) em um elemento de memória. [11]

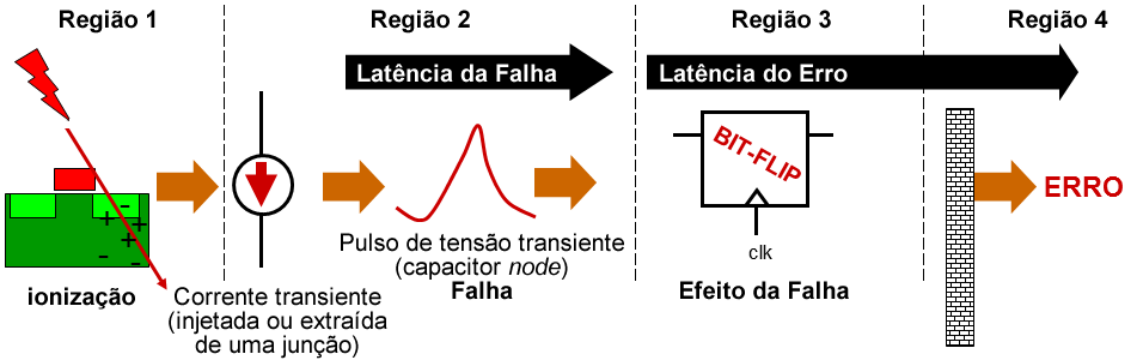


Figura 17. Latência de uma falha e do erro. Adaptada de [11].

2.1.4. Erro

O *bit-flip*, provocado pelas cargas provenientes da interação da partícula radioativa com o material que compõe o dispositivo, pode se propagar para as saídas observadas do circuito integrado, resultando em um erro (latência do erro, região 4 da Figura 17) [11]. Um erro é uma manifestação de uma falha, que se propaga até as saídas do sistema (Figura 18-b), no qual pode ser detectável e perceptível [35]. Nem todas as falhas causam, necessariamente, um erro, mas, com certeza, um erro é gerado por uma falha (Figura 18) [10, pp. 7-9]. Como as falhas, os erros podem ser classificados como permanentes ou transientes. As falhas permanentes causam erros permanentes ou *hard errors* e as falhas transientes provocam erros transientes, também chamados de *soft errors* [10, pp. 7-9].

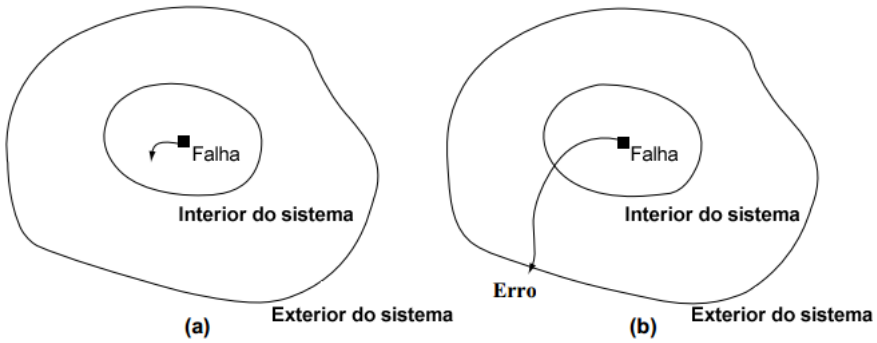


Figura 18. (a) A falha ocorreu no interior do sistema e não se propagou. (b) A falha ocorreu no interior do sistema, propagou-se para fora, gerando um erro. Adaptada de [10, p. 9].

2.3. Métodos de Validação de Sistemas Tolerantes a Falhas Baseados em Circuitos Integrados Sujeitos a SEUs

Para validar o dispositivo microcontrolados e sistemas tolerantes a falhas deve-se reproduzir o ambiente espacial, a que o dispositivo será exposto, da maneira mais fidedigna possível para simular *Single Event Upsets* (SEUs também chamados de "*upsets*" ou "*bit flips*"). Encontra-se na literatura científica a validação de microcontroladores COTS e

sistemas tolerantes a falhas sendo realizada por quatro métodos distintos, a saber: métodos baseados na exposição do circuito a um fluxo de partículas radioativas com auxílio de acelerador de partículas; métodos baseados na exposição do circuito a feixe de laser pulsado; métodos baseados em injeção de falhas por *hardware* e métodos baseados em injeção de falhas por *software* [36].

Os métodos de validação de sistemas computacionais e circuitos integrados baseados na exposição do circuito a um fluxo de partículas radioativas são de custo altíssimo e dispendioso. Esses métodos demandam câmaras a vácuo, aceleradores de partículas, experiência e conhecimento dos pesquisadores, técnicos qualificados e laboratórios adequados para manipular e armazenar o material radioativo [10, pp. 62-66]. O método de exposição do circuito a um fluxo de partículas radioativas permite estimar experimentalmente de forma mais realística a vulnerabilidade de um circuito integrado a SEUs, porque o dispositivo é exposto a um fluxo contínuo de radiação, seja através de lâminas de elementos radioativos, que emitem partículas alfas, fixadas bem próximas do *chip*, ou atacados intensamente por um feixe de nêutrons ou prótons [10, pp. 62-66]. Sendo assim, esse método é o que mais se aproxima do fenômeno real ocorrido no espaço, por isso é particularmente útil para calcular com precisão a *Soft Error Rate* (SER) do dispositivo, medida em erros por unidade de tempo. Contudo, esse método aplicado a sistemas baseados em dispositivos microprocessados (ex.: microcontroladores) só é válido, se o *software* programado no dispositivo estiver executando durante a exposição ao fluxo de partículas radioativas e de preferência que o programa seja o da aplicação final para a qual ele será utilizado. Além disso, a cada ensaio radioativo, um novo protótipo deve ser produzido para a realização do teste, pois o protótipo deve ser descartado e isolado adequadamente após os testes, porque partes metálicas do dispositivo testado podem tornar-se radioativas, devido à exposição do dispositivo ao material radioativo [10, pp. 62-66].

Métodos de validação de circuitos integrados baseados na exposição a um feixe de laser pulsado são técnicas promissoras para testar e descobrir regiões vulneráveis de circuitos integrados, emulando determinados efeitos radioativos com a principal vantagem sendo a precisão no tempo e no local da injeção de falhas [36]. O efeito de ionização do material semicondutor submetido à radiação é simulado por um efeito fotoelétrico produzido pelo feixe de laser pulsado de alta frequência e potência, que é aplicado ao elemento de memória do circuito integrado [37]. A precisão desse método permite obter um efeito localizado que a

falha provoca em um determinado transistor ou região específica do dispositivo. As desvantagens dessa técnica se traduzem na lentidão do processo de emulação de SEUs e na demanda equipamento de alta precisão e custo elevado, além de um preparo específico do dispositivo em teste. A fim de que o laser possa atingir as camadas semicondutoras, o *chip* deverá ser submetido a um processo de raspagem ou a um processo químico de remoção da cobertura do encapsulamento (*etching* ou *thinning*), para deixar o circuito livre à exposição do feixe de laser pulsado [37].

As técnicas baseadas em *hardware*, quando aplicadas a microcontroladores, são rápidas, demandam o uso e a configuração de recursos para depuração e *tracing* e necessitam de um *hardware* adicional para injetar as falhas no dispositivo alvo. Essas técnicas demandam ainda a ativação de interrupções externas ou o acesso direto à memória. Tais técnicas são divididas em duas categorias: injeção de falhas por *hardware* com contato e sem contato. A primeira categoria, o *hardware* adicional, em contato físico direto com o dispositivo alvo, produz mudanças de tensão e/ou corrente no alvo. A segunda, uma fonte externa produz a injeção de falhas, tal como uma fonte de radiação ou interferência eletromagnética, causando o surgimento de correntes dentro do *chip* testado [38]. Esses métodos têm-se demonstrado bem adequados para a emulação de SEU [38].

Por último, métodos de validação de circuitos integrados baseados em injeção de falhas por *software* têm despertado o interesse de pesquisadores como uma ferramenta atrativa que não necessita de gastos dispendiosos com *hardware* [38]. Essa técnica é dividida em duas categorias: injeção em tempo de compilação e injeção em tempo de execução. Essas técnicas, quando aplicadas em microcontroladores, demandam simuladores e/ou emuladores com o conjunto de instruções do dispositivo em teste, ou ainda, um modelo do circuito em teste descrito em alguma *Hardware Description Language* (HDL). Possuem maior flexibilidade, porém demandam muito tempo de processamento de CPU e a frequência de execução da aplicação simulada, nem sempre é a mesma quando a aplicação é executada no dispositivo microprocessado. São baseadas em modelos já existentes e a qualidade da resposta depende da precisão do modelo usado, também é limitada à validação de circuitos integrados COTS de baixa complexidade [38].

O Quadro 1 mostra um resumo comparativo entre esses quatro diferentes métodos desenvolvidos em laboratório, baseados em características como: custo, flexibilidade, demanda, tempo de experimento e limitações/vantagens.

Quadro 1. Comparação entre os diferentes métodos de validação de circuitos integrados COTS e sistemas tolerantes a falhas.

Baseadas em *Software*	Baseadas em *Hardware*	Baseadas a Laser Pulsado	Baseadas em Testes de radiação
Custo menor	Custo intermediário	Custo elevado	Custo mais alto de todos
Maior flexibilidade	Menor flexibilidade	Demanda pessoal, laboratório e instrumentação específicos para os ensaios de injeção de falhas e medida de erros.	Demanda experiência dos pesquisadores, instrumentação e laboratório específicos para ensaios envolvendo materiais radioativos.
Baseada em modelos já existentes. Qualidade da resposta depende da precisão do modelo.	Demanda desenvolvimento de *hardware* específico.	Demanda remoção de parte do encapsulamento para a exposição do circuito integrado COTS ao fluxo de laser.	Controle complicado sobre o tempo de injeção e recuperação dos dados.
Demanda muito tempo da CPU.	Muito rápido	Processo demorado e tem a necessidade de instrumentação específica.	Processo lento. O protótipo deve ser descartado ao final do ensaio.
Limitado a validação de microprocessadores ou microcontroladores COTS de baixa complexidade.	Aplicável à maioria dos microprocessadores ou microcontroladores COTS usados em sistemas de computação.	Permite injeção de falhas em locais precisos dos circuitos integrados COTS, com isso permite-se obter o efeito localizado de provocar uma falha em um determinado transistor ou região específica do dispositivo.	Maior proximidade com o fenômeno real ocorrido no espaço. Essa metodologia de teste é particularmente útil para levantar com precisão a *Soft Error Rate* (SER).

Fonte: Mukherjee [10, pp. 62-66], M.-C., Fouillat et al. [37], Hsueh et al. [38].

Essas técnicas, aqui apresentadas, são relevantes para aprofundar o estudo sobre os efeitos transientes provocados por SEUs em dispositivos eletrônicos COTS, pois através desse estudo é possível investigar soluções para mitigar esses efeitos causados por *upsets*, em circuitos integrados. Dessa forma, o uso dessas técnicas em laboratório é importante para a validação de sistemas computacionais tolerantes a falhas construídos a partir de circuito integrados, usados em missões espaciais [10, pp. 1-2].

2.4. Descrição de Materiais e Métodos da Referência Base usada nessa Dissertação

Métodos de validação de circuitos integrados baseados em injeção de falhas por *hardware* e *software* vêm sendo desenvolvidos desde 2000 pelo grupo ARIS (*Architectures for Robust and complex Integrated Systems*) do laboratório TIMA (*Techniques of Informatics and Microelectronics for integrated systems Architecture*) localizado em Grenoble – França [39]. O grupo possui como uma das atividades de pesquisa o estudo dos diferentes tipos de interferências e efeitos parasitários, que afetam o comportamento de circuitos integrados em ambientes sujeitos a partículas de radiação de alta energia [39]. Para isso, o grupo desenvolveu uma metodologia com a necessidade de um *hardware* específico e dedicado, projetado para a validação de circuitos integrados. Esse *hardware* é usado para quantificar com comprovação estatística a SER em aplicações finais que serão executadas em diversos dispositivos, como por exemplo, microprocessadores, microcontroladores, memórias, DSPs e FPGAs voltados para missões espaciais [39][40]. O objetivo dessa metodologia é obter informações sobre a sensibilidade à radiação de dispositivos programados com *software* usado em missões espaciais, através de experimentos em laboratório sujeitos à incidência de partículas radioativas de alta energia, permitindo prever a SER para aplicações executadas nesses dispositivos ou sistemas [41].

Para concretizar a metodologia de validação de circuitos integrados sujeitos à radiação, o grupo da França desenvolveu e projetou uma plataforma dedicada para validar circuitos integrados sujeitos a SEUs, chamada THESIC *tester* (do inglês, *Testbed for Harsh Environment Studies on Integrated Circuits*). O THESIC, Figura 19, é um sistema projetado para realizar testes em laboratório de exposição à radiação em dispositivos com arquiteturas digitais e emular os efeitos de SEUs nos mesmos. O THESIC permite realizar testes *on-line* com dispositivo, que foram submetidos à radiação (do inglês, *Device Under Test*, DUT), reduzindo os custos dos testes e o tempo de experimento se comparados aos experimentos realizados exclusivamente com aceleradores de partículas [41]. O THESIC é composto principalmente por dois blocos: a placa mãe, mostrada na parte inferior esquerda da Figura 19, desenvolvida para fornecer dados para/da interface com o usuário e controlar as operações no dispositivo a ser testado; e a placa filha, mostrada na parte inferior direita da Figura 19, desenvolvida e projetada adequadamente para cada dispositivo (DUT) a ser qualificado, microprocessadores, microcontroladores, memórias, DSPs ou FPGAs [41]. A placa mãe, como já citado, é responsável pelo controle das operações relacionadas ao DUT, a saber: ligar

e desligar o DUT, controle de consumo de corrente, iniciar e parar os ciclos de testes, provocar interrupções para injeção de falhas no DUT, receber, pré-processar e transmitir os dados para/da interface do usuário no computador através da comunicação serial [41]. Uma memória compartilhada (do inglês *Memory Mapped Interface*, MMI) é usada para realizar a comunicação assíncrona entre as duas placas, durante um teste típico do THESIC. A placa filha sinaliza através de um sinal de interrupção, quando a área da MMI está preenchida com dados a serem transferidos para a placa mãe [41]. O computador, mostrado na parte superior esquerda da Figura 19, faz a interface com o usuário e permite o monitoramento *on-line* do teste executado [41].

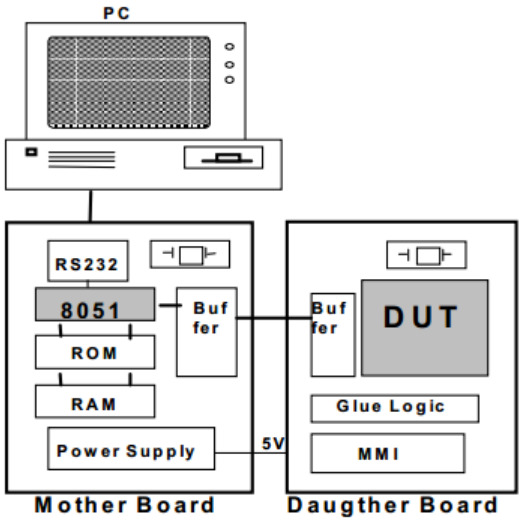


Figura 19. Diagrama esquemático do THESIC. Fonte: [41].

A metodologia desenvolvida no laboratório TIMA permite a injeção e a emulação de SEUs aleatoriamente no tempo e no espaço, simultaneamente a execução de um programa no dispositivo microprocessado COTS testado [42]. O alvo dos *upsets* injetados são *bits* de informações armazenadas em um conjunto de elementos de memória, como registradores e endereços da memória interna do dispositivo a ser qualificado [42]. A injeção é realizada através do mecanismo de interrupção do dispositivo microprocessado COTS [41]. O elemento fundamental dessa técnica é a inserção de um trecho de código armazenado a partir de um endereço de memória de programa (mais especificamente, na rotina de tratamento de interrupção), o qual será responsável por provocar a inversão de *bits* selecionados como alvo [41]. Um sorteio no trecho de código inserido permite a escolha aleatória do local (alvo) da injeção da falha [40]. Velazco et al. chamam essa metodologia de *Code Emulating Upset* (CEU), bem como o trecho de código inserido no programa para emular os *upsets* é chamado

de CEU *code* e os *bits* selecionados como alvos de CEU *targets* [43]. Como conclusão de sua análise, Faure et al., em [44], provaram por meio dos resultados de seus experimentos de exposição à radiação de um processador baseado na arquitetura SPARC V8 [45], executando um programa de leitura a diversos tipos de memória (registradores de propósito geral, cache de instrução e cache de dados), que o tempo entre chegadas de duas partículas é exponencialmente distribuído e caracterizado pela distribuição de Poisson, com parâmetro lambda igual a 2,3 partículas por segundo. Por isso o processo de injeção das interrupções (taxa de injeção de SEUs), emulado a parte e independente de um acelerador de partículas, deverá obedecer a essa distribuição, permitindo assim, que a emulação de falhas no DUT seja totalmente aleatória no tempo [44]. O sorteio aleatório, que gera os números aleatórios exponencialmente distribuídos, é computacionalmente eficiente usando o método de transformação inversa [44], desde que a função de distribuição inversa cumulativa da distribuição exponencial tenha a forma matemática, (1), como comprovado em [44] por Faure et al.. Dado uma variável U uniformemente distribuída entre 0 e 1, a função de probabilidade (Q) é exponencialmente distribuída com parâmetro λ:

$$Q = \frac{-\ln(U)}{\lambda} \tag{1}$$

Velazco et al. destacam em diversos artigos que a técnica CEU pode ser alcançada com a mínima “intrusividade” no sistema [41] e [42], essa afirmação é questionável, pois o conceito de intromissão é subjetivo. Se essa intromissão for considerada do ponto de vista da aplicação, essa técnica para injeção de falhas demanda a inserção de um trecho de código na memória de programa da aplicação final com o objetivo de modificar elementos de memória do dispositivo, caracterizando uma forma clara de intromissão no sistema testado. Por outro lado, acredita-se que Velazco et al usou o termo “mínima intromissão” com o sentido de o método causar mínimo impacto na operação do circuito, pois o tempo necessário para injetar um *upset* é relativamente curto e na maioria das vezes pouco prejudicial ao desempenho da aplicação [43].

Velazco et al. enfatizam, também, o baixo custo da metodologia, comparada com experimentos realizados exclusivamente com materiais radioativos em laboratório, tais como os realizados com aceleradores de partículas. Velazco também destaca a possibilidade da automatização do processo, a facilidade de realizar experimentos com o DUT e analisar os resultados *on-line* e a disponibilidade de um modelo de validação de um dispositivo

microprocessado ou microcontrolado COTS flexível, além da possibilidade de se testar diversos tipos de dispositivos com várias versões de *software* de maneira rápida [41]. Entretanto, Velazco et al. apontam, em [41], duas limitações dessa metodologia: a primeira limitação, "*as interrupções são sempre levadas em consideração em um instante fixo e pré-determinado, assim os efeitos dos upsets que ocorrerem durante a execução da instrução não poderão ser simulados",* e, a segunda limitação, *"nem todos os possíveis alvos sensíveis poderão ser alcançados*".

Entende-se que Velazco et al. descartaram essas limitações por dois motivos: a primeira, o desempenho de processadores modernos permite que eles executem milhões de instruções por segundo, sendo assim o tempo de execução de uma instrução é muito rápido - cerca de algumas dezenas de nanossegundos ou menos - em relação ao tempo médio de ocorrência de *upsets*, aproximadamente 2,3 *upsets* por segundo por centímetro quadrado [44], deste modo a possibilidade dos dois eventos acontecerem no mesmo instante pode ser considerada pouco provável; a segunda, o percentual acessível da área sensível é relativamente pequeno, em razão do enorme espaço de memória interna disponível dentro dos circuitos integrados COTS. Velazco et al. ainda reforçam que a metodologia produz resultados próximos e/ou superiores à situação real vivida por sistemas baseados em microprocessadores ou microcontroladores COTS susceptíveis à SEUs em ambientes espaciais [41].

Velazco et al. descrevem em detalhes [40][41][42][43] os procedimentos necessários para utilizar essa metodologia de injeção de falhas baseada em *Code Emulating Upset* e os cálculos necessários para obter a SER na validação de um dispositivo microprocessado COTS. Em síntese, a metodologia de forma organizada e completa é composta dos seguintes passos:

Passo 1: Obtenção da *cross-section* estática.

O processo de obtenção da *cross-section* estática é realizado com *chips* de memória (DUT) produzido com uma tecnologia de fabricação CMOS. Nesse processo, o THESIC preenche completamente o espaço de memória do DUT com um padrão de dados conhecido, combinações de 0's e 1's alternados.

O DUT (*chips* de memória) gravado com um padrão de bits conhecido é submetido a um feixe de radiação com material radioativo para determinar a sua *cross-section* estática. A

cross-section estática, representada por σ_{SEU}, é calculada pela divisão do número de erros contabilizados pela fluência, número de partículas incidentes por cm^2 do material radioativo usado no experimento. A medida é calculada em (2):

$$\sigma_{SEU} = \frac{\#erros}{\frac{número\ de\ partículas}{cm^2}} \tag{2}$$

De acordo com Velazco et al. em [40] e [43], a *cross-section* estática extraída nesse processo de exposição à radiação serve para todos os dispositivos com a mesma tecnologia de fabricação CMOS do DUT exposto. Durante a sessão de testes radioativos, o THESIC é responsável por monitorar os *bits* da memória com o objetivo de levantar a frequência de SEUs que ocorrem durante o experimento. Desse modo, o número de erros é contabilizado sempre que houver discrepância em ao menos um *bit* entre os dados lidos/recebidos e os valores esperados.

Os erros contabilizados pelo THESIC podem ser classificados em três categorias [40][43]:

- *Tolerated Errors*: são injeções de *bit flips* em elementos de memória nos quais o seu conteúdo não é relevante para a resposta da execução do programa, por exemplo, o *upset* é injetado em um registrador e logo em seguida esse registrador é atualizado, sobrescrevendo a inversão de *bit* injetada. *Tolerated Errors* são, portanto, injeções de *upsets* que não provocam nenhum efeito nas saídas do programa.
- *Result Errors*: são injeções de *bit flips* em elementos de memória que provocam alterações detectáveis nas saídas do programa, divergindo, ao menos em um *bit*, o resultado recebido pela CPU principal em relação ao padrão esperado.
- *Sequence Loss*: são casos que provocaram travamentos no sistema, suficientes para não receber nenhuma resposta do DUT. Nos casos de erros críticos de perda de sequência, um *watchdog* programável deve estar disponível no THESIC para reiniciar o DUT e retornar ao seu funcionamento normal.

Passo 2: Cálculo da *cross-section* predita da aplicação final.

A *cross-section* predita de uma aplicação em execução no dispositivo microprocessado ou microcontrolado, representada por $\sigma_{SEU(predita)}$, é o percentual de memória ocupado pela aplicação vezes a *cross-section* estática mensurada pelo THESIC com a memória no processo de exposição à radiação, calculada pela equação (3):

$$\sigma_{SEU(predita)} = percentual * \sigma_{SEU} \quad (3)$$

Velazco et al. afirmam que a *cross-section* predita de uma aplicação gravada em microprocessador ou microcontrolador, que foi fabricado com a mesma tecnologia de fabricação CMOS da memória exposta à radiação no passo 1, é diretamente proporcional ao percentual de memória ocupado pela aplicação [40][43].

Passo 3: Sessões de injeção de falhas com o THESIC.

Esse terceiro passo é realizado em um ambiente controlado que não demanda experimentos do DUT com partículas de materiais radioativos. Se o DUT é uma memória, o THESIC preenche todo o espaço dessa memória com um determinado padrão de dados conhecido (0's e 1's alternados) e sem a presença de falhas, a placa mãe do THESIC dispara sinais de interrupção (*upsets*) aleatoriamente no tempo e no local, podendo alterar uma ou mais posições de memória, invertendo o nível lógico do(s) dado(s) armazenados nessa(s) posição(ões) de memória. Após isso, os dados são descarregados na MMI e comparados com os valores conhecidos, para assim classifica-los segundo uma das categorias de erros mencionadas antes [41]. Quando o DUT é um dispositivo microprocessado, um programa para inicialização e testes é executado e a placa mãe do THESIC dispara sinais de interrupção (*upsets*) randomicamente no tempo e nos pinos de entrada do DUT [41]. Após isso, o DUT descarrega os dados das saídas da execução do programa na MMI e indica ao THESIC, por meio de interrupção, quando os dados estão disponíveis para a leitura [40]. O THESIC as compara com os valores esperados e classifica os resultados segundo uma das três categorias de erros mencionadas anteriormente no passo 1 [41].

Desse modo, pode-se identificar a medida probabilística que expressa o percentual da quantidade de *upsets*, os quais resultaram em SEUs observáveis na saída do DUT, representada por τ_{CEU}. Esse valor percentual de erros é calculado pela divisão do número de erros detectados pelo número de CEUs injetados, mostrado em (4):

$$\tau_{CEU} = \frac{\#erros}{\#CEUs\ injetados} \tag{4}$$

Passo 4: Definição da *cross-section* semi-experimental da aplicação final.

A *cross-section* semi-experimental do programa particular em execução no dispositivo microprocessado ou microcontrolado, representada por $\sigma_{SEU\ (semi-experimental)}$, é calculada pelo produto da *cross-section* predita obtida no passo 2, $\sigma_{SEU(predita)}$, vezes o valor percentual de erros obtido no experimento simulado, τ_{CEU}, que por sua vez é calculada no passo 3. As *cross-sections* semi-experimental e predita são dependentes da aplicação gravada no dispositivo em teste e da quantidade real de elementos de memória que a aplicação no DUT demanda. A *cross-section* semi-experimental é obtida pela equação (5):

$$\sigma_{SEU\ (semi-experimental)} = \sigma_{SEU(predita)} * \tau_{CEU} \tag{5}$$

Passo 5: *Soft Error Rate* estimada (semi-experimental).

A *Soft Error Rate* estimada é uma medida de erros por unidade de tempo, representada pela sigla SER, e pode ser estimada em função do tempo ou em situação de voo. A SER semi-experimental é calculada pelo produto da *cross-section* semi-experimental vezes o fluxo de radiação, que é uma medida obtida para cada dispositivo microprocessado COTS submetido pelo menos uma vez a um feixe de material radioativo e a unidade é expressa em números de erros observáveis por segundo. A SER é calculada em (6):

$$SER = \sigma_{SEU(semi-experimental)} * Fluxo \qquad \left(\frac{erros\ observáveis}{unidade\ de\ tempo}\right) \tag{6}$$

Velazco et al. demonstram que os valores obtidos experimentalmente do τ_{CEU} em sua metodologia baseada em injeção de falhas com o *hardware* THESIC, se aproximam dos valores obtidos pelo dispositivo microprocessado COTS com a aplicação final em funcionamento como se estivesse sujeito à exposição por radiação em um acelerador de partículas, com um intervalo de confiança maior ou igual a 95% [44]. Sendo assim, essa metodologia é uma importante contribuição para a previsão da SER, com precisão e

embasamento estatístico, de um dispositivo microprocessado COTS executando um programa de aplicação final usado em experimentos de exposição à radiação [41]. Segundo Velazco et al., uma vez obtida a *cross-section* de um dispositivo microprocessado COTS expondo-o a testes de radiação, pode-se usar a metodologia para estimar com precisão a *cross-section* de outras aplicações sem a necessidade de recorrer novamente à exposição do dispositivo microprocessado COTS a radiação [44].

Capítulo 3 - PROPOSTA DO PROJETO

Esse trabalho consiste no desenvolvimento de uma ferramenta, que reproduz a metodologia baseada em *Code Emulating Upsets* implementada e apresentada por Velazco et al. em [41], [42], [43] e [44]. Essa ferramenta pode ser usada para a validação de sistemas computacionais com tolerância a falhas, construídos com microcontroladores COTS, destinados a satélites científicos projetados para operar em baixa órbita. É importante relatar algumas diferenças dessa abordagem em relação à apresentada por Velazco et al., são elas: utiliza dispositivos de fácil aquisição/compra e de baixo custo e não possui a necessidade de construir um *hardware* específico para cada validação do dispositivo microprocessado COTS. É necessário realizar uma montagem do DUT com a arquitetura mínima para o seu funcionamento, usando componentes como: cristal oscilador para *clock*, fonte de alimentação, resistores e capacitores. Os resultados relatados por Velazco et al. em [40], [41] e [44] são usados como base de comparação com os obtidos nos experimentos práticos de validação da ferramenta desenvolvida nesse trabalho, a princípio, sem a necessidade de expor o dispositivo microprocessado COTS à radiação. No capítulo de resultados, uma análise detalhada dessa premissa será apresentada.

A Figura 20 mostra o ambiente de injeção de falhas desenvolvido nesse trabalho. Esse ambiente é dividido em três módulos: uma interface com elementos gráficos (IG), na parte superior esquerda destacada em verde na Figura 20; a placa de aquisição, na parte superior direita destacada em azul na Figura 20; e o gerador de *clock*, abaixo da placa de aquisição destacada em preto na Figura 20, além do dispositivo a ser qualificado (dispositivo alvo), na parte inferior destacado em vermelho na Figura 20. A IG pode ser executada em um computador ou em um dispositivo que permite interação com o usuário, como por exemplo, um microcontrolador com display LCD e botões para controle.

O sistema de injeção de falhas faz todo o controle e monitoramento dos experimentos, como por exemplo, controlar o início e o fim dos mesmos. O *software* do sistema de injeção de falhas exerce as funções de controlador, gerador de falhas, monitor, analisador de dados e *watchdog*, e o *hardware* exerce as funções de injetor de falhas, transmissor e receptor de dados entre o computador com o *software* e o dispositivo alvo e coletor de dados. Essa reprodução da metodologia baseada em CEU de Velazco, proposta no trabalho, pode ser aplicada em qualquer dispositivo microprocessado COTS (dispositivo alvo), que possa

executar um determinado programa e permita a configuração seu mecanismo de interrupção para injetar *bit flips* através do seu conjunto de instruções na sua área sensível.

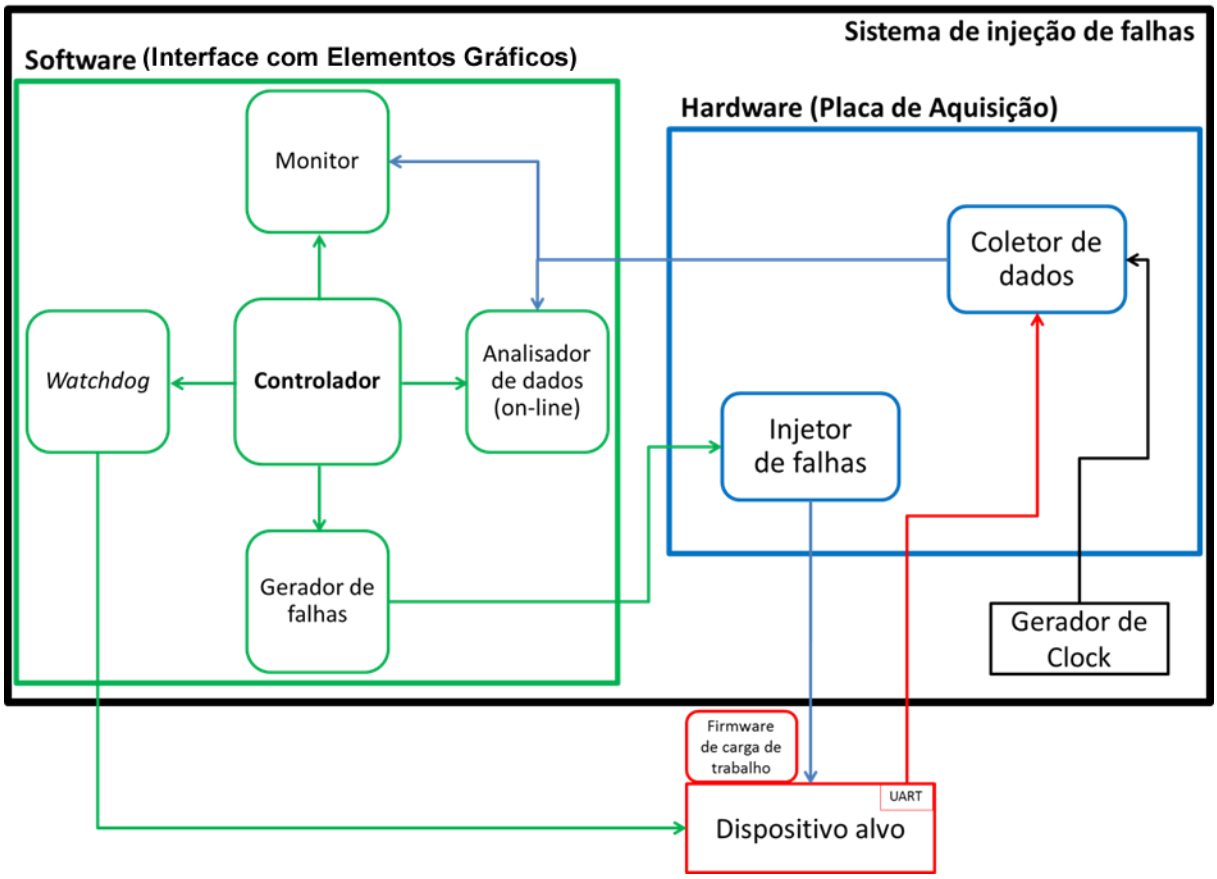


Figura 20. Diagrama de blocos do ambiente de injeção de falhas. Adaptado [38].

Ferramentas desenvolvidas para testes de injeção falhas são usadas por engenheiros para verificar o funcionamento de componentes ou de sistemas construídos a partir de microprocessadores ou microcontroladores COTS, a fim de investigar seu comportamento na presença de falhas e também são empregados na validação de sistemas desenvolvidos com técnicas de tolerância a falhas [38]. As técnicas de tolerância a falhas, quando corretamente implementadas, permitem que os sistemas computacionais continuem a operar adequadamente mesmo após o aparecimento de falhas em alguns de seus componentes [46, pp. 265-266].

3.1. Generalização dos Materiais e Métodos Necessários

Nesta seção, serão descritos os materiais usados para implementar o gerador e o injetor de falhas e o sistema de aquisição e análise dos dados, bem como os procedimentos necessários para a o uso correto do sistema desenvolvido.

3.1.1. Materiais

O desenvolvimento de interfaces e o uso de algumas ferramentas e dispositivos para aquisição e análise dos dados com características e recursos específicos são necessários para aplicar a reprodução da metodologia baseada em CEU descrita nesse trabalho. Nesta seção, serão relatados os detalhes de *software* e *hardware*, que são necessários ao desenvolvimento do injetor de falhas. O *software* e o *hardware* são denominados materiais do projeto.

Os principais materiais para o desenvolvimento do projeto são:

- *Software*: Um *software* de projeto de sistemas;
- *Hardware*: Um dispositivo eletrônico de aquisição de dados e um gerador de frequências e um computador.

O *software* de projeto de sistemas é necessário para o desenvolvimento da interface com elementos gráficos do sistema de emulação do comportamento de falhas. Ele deve conter recursos que permitam a implementação de tarefas de tratamento de dados como, por exemplo, a aquisição e a análise de informações coletadas de um dispositivo externo (DUT), e também permitir a transmissão e recepção de um conjunto de dados entre a IG e o DUT através de portas de comunicação, como por exemplo, UART, USB, entre outras. Esses atributos são essenciais no processo de detecção e injeção de falhas proposto na reprodução da metodologia baseada em CEU desse trabalho. A IG deve permitir o controle total de todo o processo experimental, além disso, deverá ser responsável pela geração e pelo monitoramento das falhas injetadas, detecção e computação dos erros através da comparação dos dados coletados com o padrão esperado, entre outras funções de gerenciamento.

O dispositivo eletrônico de aquisição de dados é outro componente essencial para o desenvolvimento do sistema proposto, pois ele realiza a intercomunicação da IG com o DUT, executando funções como o condicionamento de sinais externos para a interface gráfica e transmissão ou recepção de comandos e informações de comunicação entre as duas partes. O condicionamento de sinais tem a tarefa de adaptar os sinais de entrada para que eles possam

ser interpretados pela IG, enquanto a transmissão de sinais tem a tarefa de controle e sincronismo da aquisição de dados. Esse dispositivo de aquisição de dados deve ter recursos para aquisição dos sinais externos, para envio e recepção de diversos sinais de controle e sincronismo como: *master clear* em casos de ocorrência de erros por travamento no DUT, injeção de falhas no DUT e contadores/temporizadores, para contagem de eventos ou geração de pulsos/sinais digitais. Além disso, deve permitir a comunicação com dispositivos externos, através da interface de comunicação como UART, USB, *etc.*, para realizar a troca de informações entre o DUT e a IG.

Um gerador de frequência é necessário para fornecer a taxa de amostragem da aquisição de dados no sistema de emulação do comportamento de falhas. De acordo com [47, p. 292], a frequência mínima de amostragem ou frequência de Nyquist (do inglês, *Sampling Rate*, f_s) deve ser maior que o dobro da frequência máxima do sinal (f_m), ou seja, $f_s > 2 * f_m$. O gerador de frequência escolhido para o desenvolvimento da reprodução da metodologia baseada em CEU deve apresentar características importantes como precisão e estabilidade na geração do sinal de *clock*, fornecendo a frequência necessária para uma adequada aquisição dos dados. Quaisquer geradores de frequência disponíveis podem ser usados, desde que eles garantam uma frequência correta e estável para a aquisição dos dados durante o processo experimental.

O computador é usado para controlar o processo de injeção, gerenciamento e análise dos dados, essas tarefas são realizadas pelo *software* (IG). Os outros materiais como: fonte de alimentação e osciloscópio podem ou não ser utilizados, porém não necessitam de descrição tão detalhada, porque são ferramentas genéricas para auxiliar a montagem dos experimentos, não necessitando de configuração específica.

3.1.2. Métodos

Para validar um dispositivo COTS microcontrolado ou microprocessado, usando a ferramenta de reprodução da metodologia baseada em CEU, é importante que a aplicação final esteja pronta e sendo executada pelo DUT no momento da realização das sessões experimentais. Isso se torna essencial para preparar o CEU *code* com elementos de memória do DUT (com o programa), que produzam erros perceptíveis ao sistema. Isso se torna imprescindível a fim de se calcular a SERs e se alcançar taxas reais de SEUs perceptíveis ao

longo do tempo como se o DUT estivesse sendo submetido a uma sessão experimental exposto à radiação.

O método adaptado da metodologia baseada em CEU desenvolvido nesse trabalho é composto por três etapas: a primeira etapa denomina-se mapeamento da área sensível, a qual consiste primeiramente a identificação dos elementos de memória e em seguida, a determinação dos efeitos causados por um *bit-flip* em cada alvo selecionado. A segunda etapa é a realização do experimento de emulação do comportamento de falhas em dispositivos microprocessados COTS, que abrange a construção do trecho de código CEU *code* e posteriormente a execução do experimento de injeção de falhas. Finalmente, a terceira e última etapa é composta pela definição da *cross-section* e da SER (*Soft Error Rate*) do programa em execução, que envolve o cálculo da *cross-section* e logo após o cálculo da SER (*Soft Error Rate*) da aplicação final. O detalhamento das etapas é descrita a seguir:

Etapa 1: Mapeamento da área sensível.

a) Identificação dos elementos de memória.

Essa tarefa consiste em identificar elementos de memória (por exemplo, registradores e as regiões da memória interna) usados pela aplicação gravada no DUT, os quais caracterizam possíveis alvos para o experimento de emulação do comportamento de falhas em dispositivos microprocessados COTS. Esses registradores são basicamente todos os registradores usados pelo programa da aplicação final, os registradores de controle e configuração do dispositivo, além do *Program Counter* (PC), do registrador de instrução e dos registradores da pilha (*Stack*). É importante ressaltar que o processo de definição da área sensível requer o conhecimento da arquitetura do DUT e da aplicação desenvolvida gravada no mesmo, impossibilitando assim uma completa automatização do processo de definição da área atacada nos experimentos de injeção de falhas. Esse mapeamento deve ser realizado sempre que se quiser qualificar uma nova aplicação desenvolvida ou um novo dispositivo COTS.

b) Determinação dos efeitos causados pelos alvos.

Continuando a etapa de mapeamento da área sensível, há a necessidade de se determinar os efeitos causados na aplicação quando os alvos identificados na tarefa de identificação de elementos de memória são atacados no processo de injeção de falhas, ou seja,

quando se altera um ou mais *bits* do seu conteúdo. Para isso, um teste específico deve ser executado, denominado teste de avaliação. O teste de avaliação consiste em diagnosticar os efeitos causados pela emulação do comportamento de falhas, por meio de interrupções programadas no *firmware* do dispositivo, em circuitos integrados COTS em um único alvo (CEU *target*, [43]). O mecanismo de interrupção externa do dispositivo em teste deverá ser configurado corretamente para permitir a emulação da falha desejada. E, desse modo, definir quais elementos de memória produzem erros detectáveis nas saídas do sistema, quando ocorre alguma mudança, em ao menos um *bit* desses elementos, causada pela emulação do comportamento de falhas no DUT.

No teste de avaliação, as falhas devem ser injetadas em curtos intervalos de tempo, cerca de algumas dezenas de milissegundos enquanto isso os resultados das saídas do circuito são observados para computar possíveis erros. O intervalo de tempo muito curto (poucas dezenas de milissegundos) é para garantir que o *bit-flip* permaneça no alvo, evitando que ele seja sobreposto com alguma outra informação, consequentemente, eliminando o *upset* inserido nele, e assim impossibilitando a visualização do tipo de erro real provocado pela falha, o que não seria interessante nos testes de avaliação usados para mapear a área sensível do DUT. Desse modo, será possível identificar quais os alvos realmente propagam as falhas nas saídas do dispositivo DUT, definindo desta maneira os CEU *targets*. Ao fim dessa tarefa, é possível classificar os elementos de memória que causam algum tipo de erro perceptível ao sistema como um dos três tipos de situações esperadas, segundo a classificação de erros apresentada por Velazco como descrito na seção 2.4 do capítulo 2, ou seja, erros toleráveis ou ausência de erros (*Tolerated Errors*); *Result Errors* ou dados recebidos diferentes do padrão esperado; e erros por perda de sequência ou erros que geram travamento do sistema (*Sequence Loss*) [40][43].

A execução dos testes de avaliação tem como objetivo identificar os elementos de memória que produziram cem por cento um tipo de erro, pois esses alvos serão utilizados na construção do CEU *code*, realizado na etapa seguinte.

Etapa 2: Realização do experimento de emulação do comportamento de falhas em dispositivos microprocessados COTS.

a) Construção do trecho de código CEU *code*.

Uma vez conhecidos a área de memória sensível do circuito e os efeitos causados pelo ataque a cada uma dessas regiões, procede-se à realização do experimento de emulação do comportamento de falhas em dispositivos microprocessados COTS. Primeiramente, o trecho de código, CEU *code*, é montado em uma rotina de tratamento de interrupção de eventos externos na aplicação final gravada no DUT com todos os alvos que serão usados nos testes. É preciso ressaltar aqui a necessidade de se configurar o mecanismo de interrupção externa do dispositivo em teste para a execução do CEU *code*, quando uma falha é injetada da mesma forma como foi feito na etapa 1.

A construção do CEU *code* (quantidade e tipo dos alvos) depende dos resultados de um experimento que se pretende realizar ou reproduzir com a ferramenta proposta no trabalho. Por exemplo, pode-se reproduzir uma sessão de testes de acordo com os dados obtidos por ensaios experimentais de injeção de falhas realizados em outros trabalhos semelhantes, como os experimentos apresentados por Velazco et al. em [44], ou por meio dos resultados de uma observação da aquisição de dados, quando o DUT é exposto à radiação ou, ainda, com os resultados alcançados por telemetria, quando se observa o comportamento do DUT programado com a aplicação em um ambiente sujeito à ação de partículas de alta energia e/ou partículas radiativas, como em voos de balões estratosféricos ou em voos de satélites artificiais de baixa órbita terrestre. Todos os alvos devem ser organizados em uma estrutura de dados do tipo "*switch case*" dentro da rotina de interrupção da aplicação. Como já mencionado, esse trecho de código é o chamado CEU *code*. Não há necessidade de embaralhar a ordem dos alvos dentro dessa estrutura de controle de fluxo, pois um sorteio aleatório configurado na própria rotina de interrupção, programada dentro do dispositivo, garante a escolha aleatória do alvo a ser atingido. Esse ataque aleatório pode alterar um ou mais *bits* de cada vez, isso vai depender do tipo de evento que se quer simular: SEU ou MCUp (do inglês, *Multiple-Cell Upset*) [1, pp. 1-2].

b) Execução do experimento de emulação de falhas.

Após a configuração do CEU *code* na rotina de interrupção dentro da aplicação final e a mesma deve estar gravada no dispositivo microprocessado COTS, o DUT é submetido ao

sistema de emulação de falhas com a ferramenta proposta. Inicialmente nesta etapa, precisa-se realizar a configuração do gerador de falhas da interface gráfica desenvolvida, responsável por criar um vetor temporal com o instante de tempo, no qual as falhas (interrupções) serão injetadas no DUT.

Primeiramente, configuram-se os parâmetros do gerador de falhas para que o sistema produza um vetor intermediário, denominado de vetor de falhas (vetor de *n* posições com os números aleatórios), com a finalidade de gerar o número de interrupções aleatórias emulando, por exemplo, o comportamento de ocorrências de partículas radioativas em um ambiente especial em baixa órbita [44]. Tais parâmetros são: *lambda*, número de amostras e o tipo de distribuição de probabilidade (Figura 21). O tipo de distribuição é a função de distribuição de probabilidade de *Poisson*, de acordo com Faure et al. em [44] o tempo entre chegadas de duas partículas radioativas é exponencialmente distribuído e caracterizado por essa distribuição. *Lambda* é o número médio esperado de ocorrências de eventos, ou seja, o número de falhas médias a serem injetadas, de acordo com Faure et al. em [44] a taxa média de chegada de partículas radioativas, ou parâmetro *lambda*, é igual a 2,3 partículas por segundo. E o número de amostras (*n*) indica o número de valores aleatórios que serão gerados, ou seja, o total de índices/posições do vetor de falhas. Baseado na configuração dos valores dos parâmetros de entrada mostrados na Figura 21, o gerador cria um vetor de falhas com *n* posições preenchidas, contendo os números aleatórios sorteados, ou seja, nesse momento o sistema gera o vetor com a frequência de falhas a serem injetadas (Figura 22).

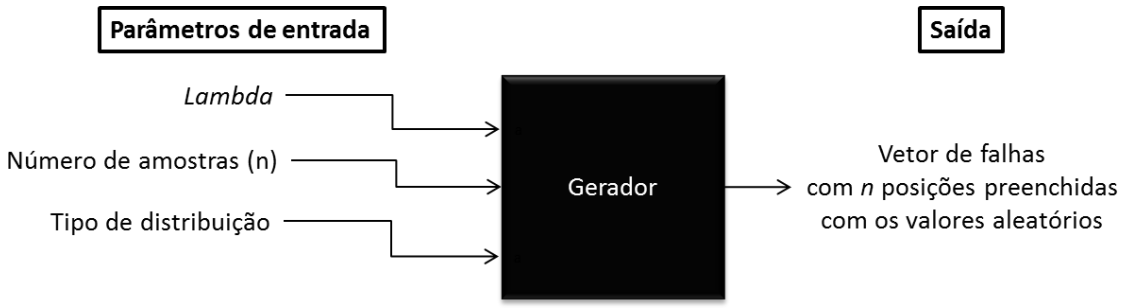


Figura 21. Parâmetros de entrada do Gerador de valores aleatórios e vetor de falhas.

Logo após o sorteio aleatório que gerou o vetor de falhas, o gerador de falhas deve criar o vetor temporal com os instantes de tempo da injeção de falhas, baseado no vetor aleatório criado anteriormente (Figura 22). Como citado e apresentado por Faure et al. em [44], a taxa média entre chegadas de duas partículas radioativas é 2,3 partículas por segundo.

Então, cada valor aleatório gerado anteriormente e armazenado no vetor de falhas indica o número de *upsets* (a frequência das falhas) que o sistema deve injetar por segundo.

Para criar o vetor temporal, o sistema divide o tempo de um segundo (intervalo de tempo na Figura 22) por cada valor de frequência armazenado nos índices do vetor de falhas, obtidos primeiro desta maneira garante-se que o número de falhas sorteado será injetado no intervalo de tempo programado. O resultado dessa divisão é somado ao valor acumulado e armazenado no vetor temporal, executando essas operações até o último índice do vetor de falhas (Figura 22).

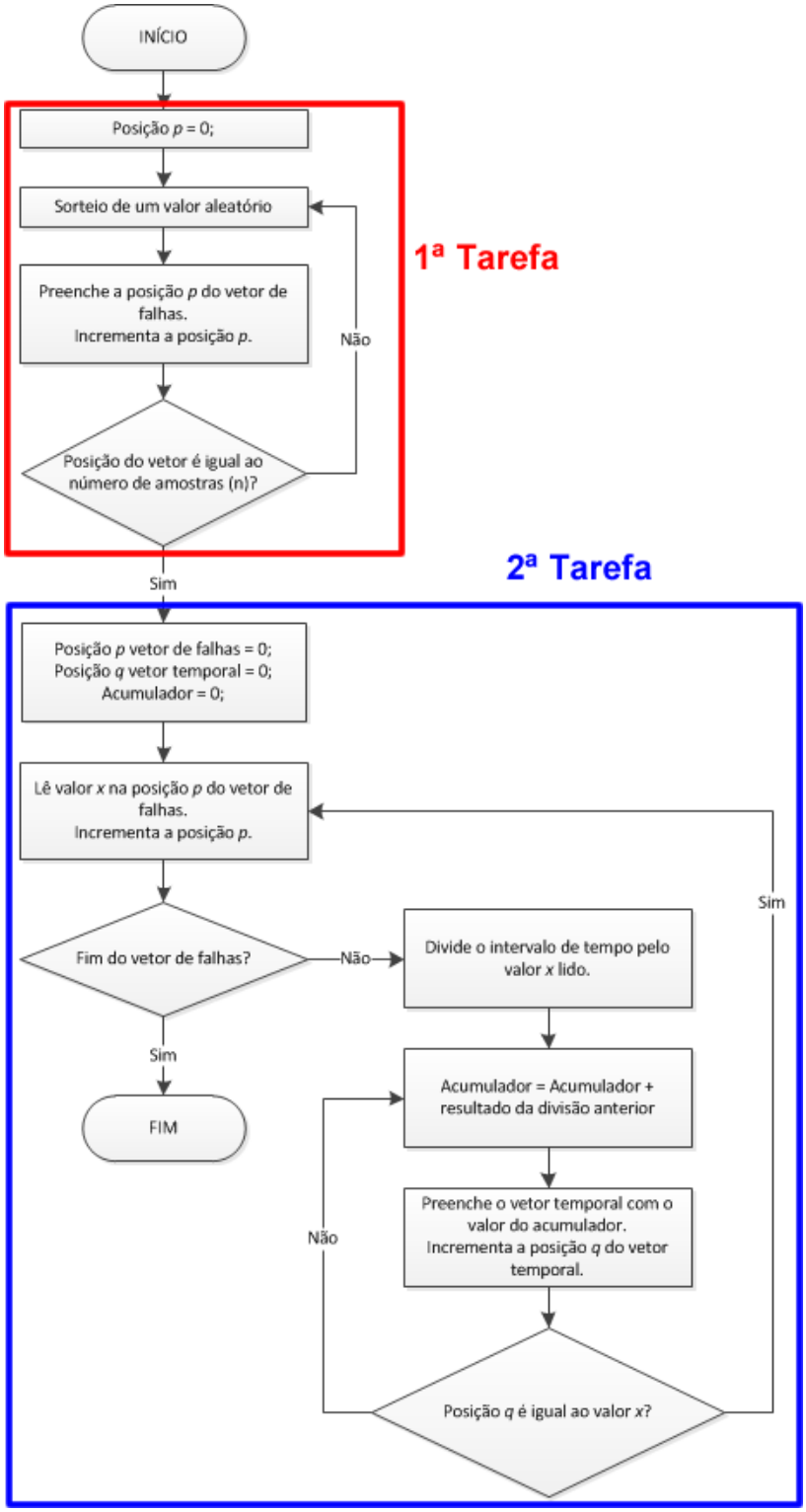


Figura 22. Fluxograma da criação do vetor de falhas e do vetor temporal.

Observando a Figura 22, que demonstra o fluxograma da criação do vetor de falhas e do vetor temporal, é possível identificar a sequência de tarefas executadas pela IG na criação desses vetores. A primeira tarefa: o sistema realiza o sorteio de um valor aleatório e armazena-o na posição *p* (inicia-se na posição *p* = 0) do vetor de falhas, após o

armazenamento o sistema incrementa a posição *p*. O sistema repete isso até o número de posições preenchidas for igual ao número de amostras configuradas pelo usuário no gerador. A segunda tarefa: o sistema realiza a leitura do valor *x* no vetor de falhas na posição *p* (inicia-se na posição $p = 0$). Com o valor lido *x*, o sistema realiza a divisão do intervalo de tempo por esse valor lido *x* (Esse intervalo de tempo pode ser configurado com qualquer valor de tempo desejado para injeção das falhas). O resultado da divisão realizada anteriormente é somado ao acumulador (valor que inicia com zero) e guardado nele mesmo. O valor do acumulador é armazenado na posição *q* do vetor temporal (inicia-se na posição $q = 0$) e a posição *q* é incrementada. Por último, realiza-se a comparação da posição *q* preenchida no vetor temporal com o valor *x* lido no vetor de falhas. Se a posição *q* for menor do que o valor lido x, o sistema continua o preenchimento do vetor temporal a partir do processo de atualização do acumulador. Caso contrário, o sistema retorna ao processo de ler um novo valor x no vetor de falhas, reiniciando a segunda tarefa.

A geração do vetor de falhas aleatórias e a construção do vetor temporal com os instantes de ocorrência das falhas são realizadas antes do início do experimento de emulação do comportamento de falhas. O motivo da realização desse passo antes do início da emulação de SEUs no DUT se deve, para não dividir o processamento da CPU principal entre a tarefa de geração de falhas e as tarefas de injeção de falhas e aquisição de dados.

Após a transformação do vetor aleatório de falhas para o vetor temporal realizada pelo sistema, o usuário deve configurar os parâmetros da aquisição de dados. São eles: tempo máximo de espera para a aquisição dos dados, padrão de saída esperado como resposta da aplicação gravada no dispositivo e canais de transferência de dados e comandos, como os canais para a injeção de falhas, para a coleta de dados e para o sinal de *clock* da frequência de amostragem. O tempo máximo de espera para a aquisição de dados é o tempo limite que a interface gráfica aguarda para a chegada de dados enviados pelo DUT. O tempo limite depende do sistema operacional (onde a IG está sendo executada), do dispositivo de aquisição usado para recebimento dos dados e da aplicação gravada no DUT, pois o intervalo mínimo de transmissão de dados é limitado por todos esses fatores. Esse tempo é importante para detectar algum travamento do sistema ou demora no envio da informação e para identificar os erros por perda de sequência (*Sequence Loss*). O *watchdog*, implementado na interface gráfica, é responsável por monitorar o tempo limite esperado para aquisição de dados. De acordo com Velazco et al. em [43], esse *watchdog* programável deve estar disponível no

sistema de injeção de falhas para reiniciar o DUT e retornar ao seu funcionamento normal, portanto ele é responsável por disparar o sinal de *Reset* para reiniciar o dispositivo em teste, sempre que erros de travamento aparecerem durante as sessões experimentais com o DUT. O padrão de saída esperado é a resposta do dispositivo em teste sem a influência do processo de injeção de falhas, ou seja, o valor correto dos dados que a aplicação gravada no DUT deverá enviar ao sistema de aquisição de dados. E os canais de I/O (Entrada e Saída) são os meios de comunicação, por onde o dispositivo de aquisição de dados irá transmitir/receber dados da/pela interface gráfica ao/do DUT.

Após a configuração correta desses parâmetros, as sessões dos testes podem ser iniciadas e assim computar a quantidade de erros detectados durante cada sessão. Ao final do experimento, os resultados da aquisição dos dados são armazenados em um arquivo, salvo automaticamente pela IG, com os valores absolutos dos erros observados, juntamente com a quantidade de falhas injetadas durante os testes realizados.

Etapa 3: Cálculo da *cross-section* semi-experimental e da SER da aplicação final.

Uma vez terminado o experimento de emulação do comportamento de falhas em dispositivos microprocessados COTS e computados os valores absolutos de cada tipo de erro, pode-se passar para a terceira etapa. Através da Interface Gráfica, realizam-se os cálculos dos percentuais de cada tipo de erros computados, da *cross-section* semi-experimental e da SER da aplicação final (semi-experimental) e da quantidade de falhas injetadas, durante as sessões experimentais para os valores dos parâmetros configurados na etapa 2. As equações (4), (5) e (6), disponíveis na seção 2.4 do capítulo 2, possibilitam a execução desses cálculos, assim, definindo os valores percentuais para cada tipo de erro, como também os valores da *cross-section* e da SER da aplicação final gravada no DUT durante os experimentos de injeção de falhas.

3.2. Sistema PORTHES

O sistema PORTHES (*Portable Testbed for Harsh Environment of Single event upset*) é uma ferramenta desenvolvida no Laboratório de Sistemas Inteligentes (LSI), na Universidade Federal de Minas Gerais (UFMG), usando o método descrito na seção 3 do capítulo 3, que permite reproduzir a metodologia CEU de Velazco et al. em [41]. O PORTHES é um sistema de emulação de SEUs que reúne técnicas baseadas em *software* e *hardware*.

O *software* usado para o desenvolvimento da interface gráfica do sistema PORTHES, foi o LabVIEW *System Design Software*, do fabricante *National Instruments*, licenciado para versão profissional. O LabVIEW é um ambiente de desenvolvimento para engenheiros e cientistas com ampla compatibilidade e integração com o *hardware*, permitindo a solução de diversos tipos de problemas e garantindo a confiabilidade para criar, desenvolver e implantar sistemas de medição e controle [48]. A IG, que durante os testes foi executada em um *notebook* com plataforma *Windows*, foi desenvolvida para permitir o controle total do processo experimental por parte de um usuário. Ela desempenha as funções de controlador, gerador de falhas, monitor, analisador de dados, *watchdog*; também é responsável por disparar aleatoriamente o sinal de injeção de falhas para o dispositivo eletrônico de aquisição de dados.

O dispositivo eletrônico de aquisição de dados, usado para realizar a intercomunicação da IG com o DUT, foi uma placa de aquisição da *National Instruments* (do inglês, *Data AcQuisition*, DAQ), modelo NI USB-6212. A DAQ (Figura 23) realiza o papel de injetor de falhas e coletor de dados em conjunto com a IG, sendo o *hardware* de contato com o dispositivo em teste (Figura 20, região destacada em azul). A DAQ coleta o sinal proveniente do DUT e repassa para o computador, onde a IG está executando.

Figura 23. Placa DAQ usado no PORTHES. Extraída de [49].

O gerador de *clock* foi necessário ao projeto para fornecer a frequência de amostragem do sinal adquirido, podendo ser qualquer equipamento, como um gerador de função ou um dispositivo microprocessado, ex.: microcontroladores. É necessário, que o equipamento escolhido forneça um sinal preciso e estável da frequência necessária. No sistema PORTHES, o gerador de *clock* escolhido para o PORTHES foi o microcontrolador, pois esse dispositivo atendeu às necessidades do sistema no fornecimento da frequência de amostragem. Não se procurou investigar a razão do problema ocorrido com o gerador de função.

O sistema PORTHES tem três modos de configuração para injeção de *bit-flips*: Manual, Automático 1 e Automático 2. No primeiro modo, o PORTHES está configurado para injetar *bit-flips* manualmente no DUT e no momento escolhido pelo usuário que conduz o processo experimental. A injeção manual de um *bit-flip* por vez é realizada por três botões independentes, um para cada MCU, disponíveis na IG. Portanto, o condutor do experimento pode introduzir um ou mais *bit-flips* em instantes de tempo distintos, sem controlar o intervalo de tempo entre um e outro evento. Nos modos Automático 1 e Automático 2, a injeção de *bit-flips* é realizada automaticamente pelo sistema PORTHES, de acordo com os parâmetros, descritos na seção 3.1.2, configurados na IG do sistema. A diferença entre esses dois modos está na quantidade de DUTs usados durante o experimento: o modo Automático 1 permite a injeção de *bit-flips* em somente um DUT e o Automático 2 realiza a injeção em três DUTs. No modo Automático 2, a falha é injetada em um DUT por vez, sendo que antes da injeção do *bit-flip* é realizado um sorteio aleatório para escolher em qual dos três microcontroladores o *upset* será injetado.

Outra configuração disponível no sistema PORTHES é a possibilidade de habilitar ou desabilitar o *watchdog* disponível na IG. Esse *watchdog* é responsável por emitir o sinal de *master clear* sempre que for detectado travamento do sistema. Porém em testes com sistemas tolerantes a falhas o *watchdog* da IG deve permanecer desabilitado, pois esses sistemas são desenvolvidos para manter seu funcionamento contínuo na presença de falhas, com o mínimo de intervenção da CPU principal (papel feito pelo PORTHES).

Além das características que foram mencionadas, o sistema PORTHES possui ainda dois modos de configuração para o recebimento dos dados: modo recebe-dado e modo recebe-dado-id. No primeiro modo, o sistema é configurado para receber apenas os dados enviados pelo microcontrolador COTS testado. Já no segundo modo, o PORTHES recebe no mesmo conjunto os dados da aplicação para serem verificados pelo PORTHES e um número de

identificação (ID) do dispositivo responsável pela transmissão dos dados. Nesse último modo de configuração da recepção de dados do PORTHES, a recepção dos dados é configurada para separar a ID dos dados, só então realizar os cálculos necessários para computar erros do sistema em teste.

Capítulo 4 - SISTEMA DE DETERMINAÇÃO DE ATITUDE COM TOLERÂNCIA A FALHAS

O funcionamento normal de um satélite artificial depende do seu correto controle e determinação da atitude [50]. Quando um satélite artificial é colocado em órbita para executar diversas funções, como coletar dados de câmeras ou sensores e transmiti-los para a Terra, o controle de atitude se faz necessário [51][52]. A atitude representa o movimento rotacional do corpo do satélite em torno do seu centro de massa, ou seja, é a orientação do satélite no espaço e está relacionada com o seu correto apontamento no espaço [53]. Para o cálculo da atitude utilizam-se basicamente sensores e microcontroladores (MCUs) embarcados no satélite [50]. Os MCUs são sistemas computacionais altamente integrados em um único circuito (CPU, memória de dados e programa, conversores AD, interfaces seriais, etc.) e atendem aos requisitos da aplicação para a determinação da atitude para satélites de baixa órbita terrestre, além de oferecer um grande potencial na redução do custo de projeto do satélite [51]. Porém microcontroladores comerciais são suscetíveis a SEUs ou *bit-flips* transientes quando expostos à radiação espacial [51]. Algumas técnicas de detecção de erros e recursos de recuperação do sistema são combinadas com técnicas de tolerância a falhas, com o objetivo de mitigar os efeitos provocados por SEUs nesses circuitos integrados utilizados em aplicações espaciais [51].

O objetivo de um sistema de determinação de atitude é definir a orientação do satélite em relação a um referencial inercial ou algum objeto de interesse específico, por exemplo, a Terra [53]. Para calcular a atitude, um ou mais vetores de referência se fazem necessários, isto é, vetores com direções conhecidas em relação ao satélite. O campo magnético da Terra, geralmente, é usado como vetores de referência para satélites artificiais [53]. Sensores de campo magnético, magnetômetro, acoplados no corpo do satélite são responsáveis por medir a orientação atual do mesmo e, sendo possível computar a atitude do satélite, através de um modelo matemático de determinação de atitude [50]. Desse modo, o satélite se orienta para manter-se em sua órbita [50]. Por conseguinte, o controle de atitude se faz necessário para monitorar a orientação do satélite em relação a referencial inercial, sendo um problema clássico, amplamente estudado na área de sistemas de controle [52].

O Sistema de Determinação de Atitude Tolerante a Falhas (SDATF), destinado à utilização em satélites artificiais de baixa órbita, tem como finalidade fornecer a atitude para

uma correta orientação do satélite em sua órbita, a partir da medição do campo magnético de um determinado ponto da órbita [50]. O SDATF tem como ideia básica usar recursos de detecção de erros fornecidos pelo dispositivo selecionado, combinado com um arranjo redundante de três microcontroladores programados com técnicas de *software* de tolerância a falhas e, desta maneira, calcular a atitude para fornecê-la à CPU principal do satélite [51].

4.1. Descrição do SDATF

A Figura 24 apresenta a arquitetura do sistema de determinação de atitude, que é composto de três módulos idênticos organizados em um esquema redundante, com o objetivo de proporcionar uma região de contenção de erros, para prevenir o sistema de controle da atitude de possíveis erros que podem ser provocados por SEUs [51]. Cada módulo é formado por um magnetômetro de três eixos, um *Position Sensitive Detector* (PSD) e um microcontrolador (Figura 24) [51]. A troca de dados interna no sistema entre os microcontroladores é realizada pelo periférico de comunicação serial UART2, enquanto a UART1 provê à saída de dados do sistema para um dispositivo externo, como por exemplo, a CPU principal do satélite (Figura 24).

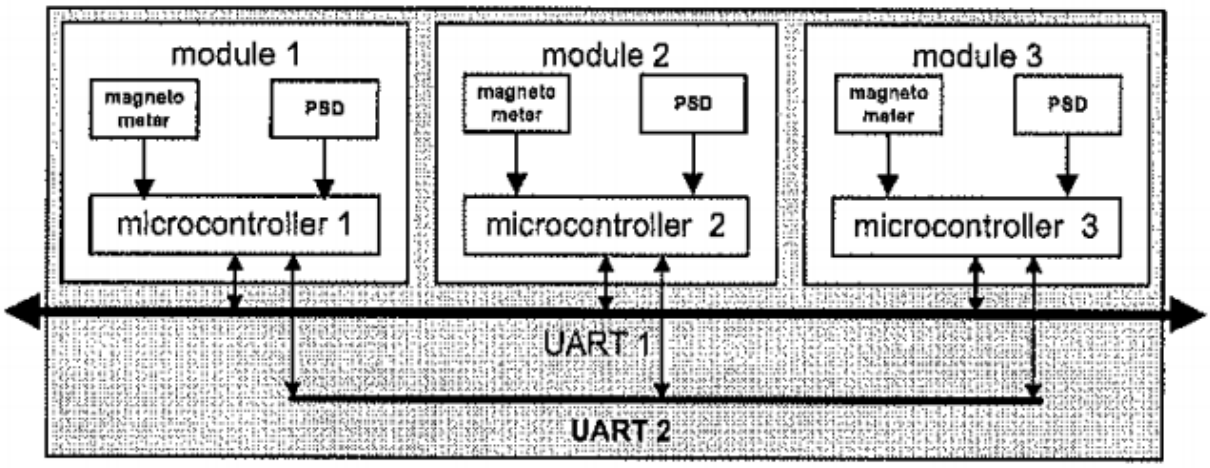


Figura 24. Sistema de determinação de atitude tolerante a falhas. Figura extraída de [51].

A arquitetura do sistema de determinação de atitude com tolerância a falhas, mostrada na Figura 24, é um subsistema independente, que necessita ser iniciado por uma CPU principal do satélite, através de um sinal de interrupção externa, para então, o sistema começar a trabalhar normalmente [51].

Os três microcontroladores usados no sistema de determinação de atitude (Figura 24) são programados com o mesmo *firmware*, assim qualquer um dos MCUs pode desempenhar uma das três funções esperadas, a saber: *SLEEPER*, *SAMPLER*, *MASTER* [51]. No momento em que o SDATF entra em funcionamento, os três microcontroladores ficam em um estado denominado *SLEEPER*. Nesse estado os MCUs do SDATF ficam em baixo consumo de energia esperando por um comando externo para começar a calcular e fornecer a atitude. Uma vez que o SDATF está funcionando com os três microcontroladores na função de *SLEEPER*, no início de sua configuração, um sinal externo de interrupção para ativar um dos MCUs é necessário para inicializar o sistema. Esse sinal de interrupção externa pode ser emitido por um dispositivo microprocessado do satélite, como exemplo, a CPU principal do mesmo. Após essa intervenção externa, o dispositivo acordado assume uma nova função: a função de *MASTER* [51]. O microcontrolador acordado, agora na função de *MASTER*, tem a funcionalidade de estabelecer a comunicação com um dos outros dois MCUs, despertando um de seus dois vizinhos, que estão no modo *SLEEPER*. No caso do *MASTER* falhar nessa tarefa, a CPU principal do satélite entende que ele não está funcionando corretamente por ausência de sinal de *acknowledgement* [51]. Consequentemente a CPU principal deve reiniciar o microcontrolador *MASTER* com falha, retornando-o ao modo *SLEEPER* e, logo após, iniciar novamente o processo de despertar outro microcontrolador para assumir a função de *MASTER* (essa é uma das poucas vezes que a CPU principal deve intervir no sistema). Um módulo despertado com sucesso pelo *MASTER* assume a função de *SAMPLER* e os dois estabelecem a comunicação trocando dados entre si [51]. As tarefas de cada função desempenhada pelos microcontroladores no SDATF são detalhadas a seguir:

- Modo *MASTER*: O MCU, que desempenha a função de *MASTER*, deve executar as seguintes tarefas: calcular a atitude uma única vez, no momento em que ele é acordado pela CPU principal, e armazenar essa atitude como última atitude válida, tentar despertar um de seus vizinhos, que estão no modo *SLEEPER*¸ para que ele se torne um *SAMPLER*. Após despertar um *SAMPLER* com sucesso e a comunicação entre eles estabelecida, o *MASTER* que executa as seguintes tarefas: transmitir temporizadamente a atitude para a CPU principal, enviar periodicamente, a cada 220ms, um inteiro sequencial, apelidado de *heartbeat*, para o *SAMPLER* através da comunicação UART2, aguardar periodicamente, até no máximo 280ms, o recebimento da atitude calculada pelo *SAMPLER* por meio da comunicação UART2 e validar a atitude

recebida, comparando-a com a última atitude válida armazenada. Em casos de discrepâncias da atitude recebida, o *MASTER* envia para CPU principal a última atitude armazenada por ele e descarta a atitude errada recebida. O *MASTER* admite que o *SAMPLER* envie a atitude errada apenas uma única vez, se o *SAMPLER* enviar mais de uma atitude errada ou não transmitir nenhum dado, o *MASTER* detecta o mau funcionamento do *SAMPLER*, reiniciando-o e desperta o outro MCU no modo *SLEEPER* para assumir como novo *SAMPLER*. Caso haja sucesso na validação, a atitude recebida é armazenada como última atitude válida e transmitida para a CPU principal do satélite e o temporizador de espera da atitude é reiniciado. O envio da atitude para a CPU principal está programado dentro da função principal do *firmware* e é condicionado para acontecer somente após quatro envios de *heartbeats* do *MASTER* para o *SAMPLER*, em um tempo máximo de um segundo [51].

- Modo *SAMPLER*: O MCU, que desempenha a função de *SAMPLER*, deve executar as seguintes tarefas: amostrar os sensores para calcular a atitude essa tarefa é executada de forma contínua na função principal do programa. Após ser despertado com sucesso pelo *MASTER* e a comunicação entre eles for estabelecida, o *SAMPLER* executa a seguintes tarefas: enviar a atitude calculada periodicamente, a cada 240ms, para o *MASTER* por meio da UART2, aguarda periodicamente, até no máximo 250ms, o recebimento do sinal de *heartbeat* enviado pelo *MASTER* através da comunicação UART2. O *SAMPLER* verifica a validade do *heartbeat*, logo que ele é recebido, em caso de encontrar discrepâncias ou na ausência do recebimento do *heartbeat*, o *SAMPLER* reinicia os dois MCUs e assume a função de *MASTER*. Caso ocorra sucesso na validação, o *heartbeat* recebido do *MASTER* é armazenado como *heartbeat* atual para ser comparado com o próximo que será enviado pelo *MASTER* e o temporizador de espera do *heartbeat* é reiniciado [51].
- Modo *SLEEPER*: O MCU, que desempenha a função de *SLEEPER*, deve executar as seguintes tarefas: esperar pelo sinal externo para acordar e assumir alguma das outras duas funções disponíveis. Esse sinal externo pode vir da CPU principal, por meio da interrupção externa INT0, nesse caso o microcontrolador acordado assume a função de *MASTER*, ou esse sinal pode ser emitido pelo microcontrolador na função de *MASTER*, através da

interrupção externa INT1 ou INT2, nesse caso o dispositivo acordado assume a função de *SAMPLER* [51].

A abordagem tolerante a falhas apresentada na Figura 24 é parcialmente baseada nas seguintes técnicas: redundância modular tripla (TMR), *cold standby spare* e controle de vivência mútua (*mutual aliveness control*), por meio de envios de *heartbeats* em uma direção e chegadas de *quaternion* (atitude) na direção inversa através da interface de comunicação serial UART2 dos microcontroladores [51].

4.2. Temporização da comunicação no SDATF

A comunicação no SDATF é temporizada e organizada para garantir o funcionamento normal do sistema, sem que ocorra alguma colisão entre os dados transmitidos, que pode interferir na comunicação entre *MASTER* e *SAMPLER* ou na comunicação entre *MASTER* e CPU principal e na estabilidade do sistema. A Figura 25 mostra a temporização dos envios do *heartbeat* entre *MASTER* e *SAMPLER* e do *quaternion* entre *SAMPLER* e *MASTER* e da atitude entre *MASTER* e CPU principal. A linha de tempo iniciada no ponto zero caracterizado como a fase de inicialização entre a comunicação do *MASTER* e *SAMPLER*, as setas em vermelho representam o envio periódico (220ms, o tempo de envio é monitorado pelo temporizador 1, TIMER1, de *MASTER*) do *heartbeat* no sentido *MASTER* para *SAMPLER*. O temporizador 2, TIMER2 (240ms), no *SAMPLER*, é responsável por monitorar o tempo de envio do *heartbeat* pelo *MASTER*. As setas em azul figuram o início das transmissões periódicas (240ms, o tempo de envio é monitorado pelo temporizador 2, TIMER2, de *SAMPLER*) do *quaternion* no sentido *SAMPLER* para *MASTER*. As áreas sombreadas em verde-cinza representam o tempo gasto para o envio do *quaternion*. O temporizador 2, TIMER2 (280ms), no *MASTER*, é responsável por monitorar o tempo de envio do *quaternion* pelo *SAMPLER*. Ainda na Figura 25, as setas menores em preto correspondem aos envios periódicos da atitude o sentido *MASTER* para a CPU principal. Na Figura 25, é possível observar o envio da atitude para a CPU principal condicionado para acontecer somente após quatro envios de *heartbeats* do *MASTER* para o *SAMPLER*, em um tempo máximo de um segundo. E por fim, o temporizador 3, TIMER3, no *MASTER* e no *SAMPLER* controlado o sincronismo do quadro a cada um segundo.

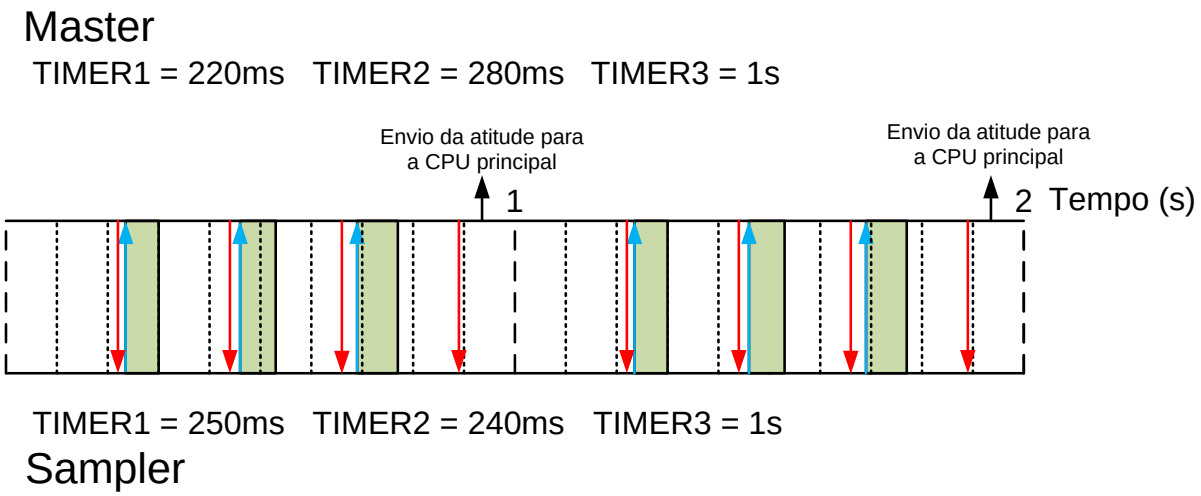
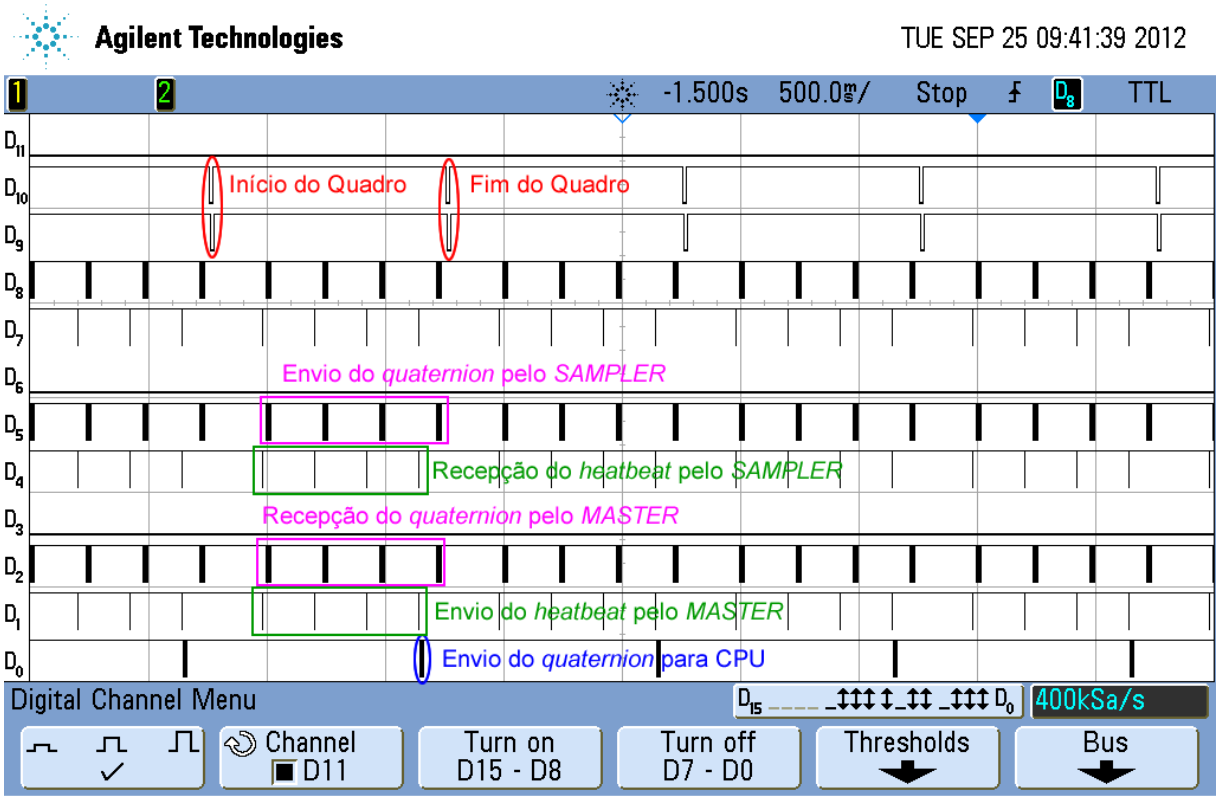


Figura 25. Quadro de temporização do SDATF.

A Figura 26 apresenta o quadro de temporização real obtido durante o funcionamento normal do sistema de determinação de atitude tolerante a falhas em um Analisador Lógico. Em destaque na Figura 26 estão: o início e fim do quadro de sincronismo (apresentados na cor vermelha), o envio e recepção do *heartbeat* (apresentados na cor verde), o envio e recepção do *quaternion* (apresentados na cor roxa) e envio a atitude para a CPU (apresentados na cor azul). Explicando mais detalhadamente os canais digitais mostrados na Figura 26, tem-se: os canais digitais D_0, D_1, D_2 e D_9, são pertencentes ao microcontrolador na função de *MASTER* e esses canais representam, respectivamente, a atitude enviada para a CPU principal (UART1), envio do *heartbeat* para o *SAMPLER*, o recebimento do *quaternion* calculado pelo *SAMPLER* e quadro de sincronismo temporizado em 1s. Os canais digitais D_3, D_4, D_5 e D_{10}, são pertencentes ao microcontrolador na função de *SAMPLER* e esses canais representam, respectivamente, o não envio da atitude para a CPU principal (UART1 desligada no modo *SAMPLER*), recebimento do *heartbeat* enviado pelo *MASTER*, o envio do *quaternion* calculado para o *MASTER* e quadro de sincronismo temporizado em 1s. Por fim, os canais digitais D_6, D_7, D_8 e D_{11}, são pertencentes ao microcontrolador *SLEEPER* e esses canais representam, respectivamente, o não envio da atitude para a CPU principal (UART1 desligada no modo *SLEEPER*), comunicação do *heartbeat* entre *MASTER* e *SAMPLER*, comunicação do *quaternion* calculado entre *SAMPLER* e *MASTER* e o microcontrolador no modo repouso (ausência do quadro de 1s). Os canais para transmissão e recepção de dados via UART2 são interligados nos MCUs, mas isso é indiferente para o microcontrolador na função de *SLEEPER*, não atrapalhando seu funcionamento normal, pois o *SLEEPER* não está configurado para transmitir ou receber algum dado.




```
Legenda:
D0  - UART1 PIC1          | D6  - UART1 PIC3
D1  - HeartBeat PIC1      | D7  - HeartBeat PIC3
D2  - Quaternion PIC1     | D8  - Quaternion PIC3
D3  - UART1 PIC2          | D9  - Quadro de 1s PIC1
D4  - HeartBeat PIC2      | D10 - Quadro de 1s PIC2
D5  - Quaternion PIC2     | D11 - Quadro de 1s PIC3
```

Figura 26. Quadro de temporização em tempo real do SDATF.

A Figura 27 representa o processo de inicialização com sucesso do sistema tolerante a falhas, realizado pelo CPU principal do satélite, no instante em que todos os microcontroladores estão no modo *SLEEPER*. A CPU envia um sinal de interrupção externa, por exemplo, através do INT0 no SDATF (sinal de Acorda *MASTER* apresentado na cor roxa, Figura 27) a um dos microcontroladores, como por exemplo, o MCU1 mostrado na Figura 27. O microcontrolador que recebeu o sinal para acordar tenta acordar o *SAMPLER* da vez (sinal de Acorda o *SAMPLER* da vez apresentado na cor alaranjada, Figura 27). Uma vez que o *SAMPLER* é despertado corretamente, ele envia um sinal para o MCU *MASTER* (sinal de Estou vivo apresentado na cor verde, Figura 27). Logo depois disso, o *MASTER* envia o sinal de *acknowledgement* para a CPU principal (sinal de *Acknowledgement* apresentado na cor azul, Figura 27) [51].

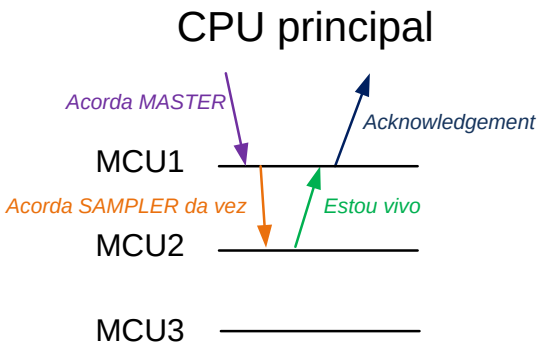


Figura 27. Processo de inicialização do sistema tolerante a falhas.

4.3. Casos de erros no SDATF

A partir do momento em que o sistema tolerante a falhas inicia seu funcionamento normal, o quadro temporizado mostrado na Figura 25 e na Figura 26 é formado, realizando a comunicação entre os microcontroladores *MASTER* e *SAMPLER*, sem a presença de falhas. Porém falhas transientes, provenientes de partículas radioativas presentes no ambiente espacial, podem produzir efeitos indesejados em elementos de memória dos circuitos integrados que formam o SDATF, alterando seu funcionamento normal e provocando mudanças nas funções de cada módulo do sistema. Os casos de erros que podem ser provocados por SEUs no SDATF são descritos a seguir:

- **Casos possíveis de erros no modo *MASTER*:**

Caso 1: *MASTER* não responde a CPU principal. O microcontrolador que foi acordado com *MASTER* pela CPU principal do satélite não enviou o sinal de *acknowledgement*. Em casos de sucesso, o MCU, que recebeu o sinal da CPU principal do satélite, deve se tornar *MASTER* e emitir o sinal para seu vizinho assumir a função de *SAMPLER*, deixando o terceiro MCU no modo *SLEEPER*. Somente após receber o Estou vivo do microcontrolador despertado como *SAMPLER*, o MCU *MASTER* pode enviar o *acknowledgement* para o computador principal. Um SEU pode provocar um *upset* no MCU *MASTER* durante esse processo, impedindo-o de completar a comunicação com a CPU principal do satélite. Nesse caso, um *watchdog* configurado no computador principal intervém reiniciando todos os microcontroladores. E, após o reinício, o computador

principal tenta despertar um novo *MASTER*, iniciando mais uma vez o processo mostrado na Figura 27 [51].

Caso 2: *MASTER* parou de funcionar. Suponha que a comunicação entre os microcontroladores *MASTER* e *SAMPLER* está funcionamento normalmente, conforme a Figura 26, onde cada MCU está desempenhando uma das funções esperadas (*MASTER*, *SAMPLER* ou *SLEEPER*). Em determinado instante, um SEU eventual provoca uma parada no funcionamento do *MASTER*. Nesse caso, o MCU *SAMPLER* irá detectar o funcionamento indevido do MCU *MASTER*, pois o último está programado para enviar *heartbeats* periodicamente para o primeiro, por meio do barramento da comunicação UART2. Se o MCU *SAMPLER* não receber o *heartbeat* do MCU *MASTER* ou se a diferença entre o *heartbeat* recebido e o último *heartbeat* recebido for maior do que uma unidade, o MCU *SAMPLER* reinicia os outros dois MCUs, inclusive o *MASTER* atual, e assume o novo papel de *MASTER*. Logo em seguida, fornece atitude para a CPU principal e tenta despertar o microcontrolador que está no modo *SLEEPER*, para se tornar *SAMPLER* [51].

Caso 3: Os detectores do *MASTER* param de funcionar. O mecanismo de detecção é baseado nos recursos do próprio microcontrolador: o *watchdog* que verifica seu próprio funcionamento, temporizadores programados para sinalizar os tempos de espera para receber a atitude calculada pelo *SAMPLER* e interrupções específicas para sinalizar um problema ocorrido no *clock* ou uma queda abrupta na alimentação do dispositivo. Um desses recursos pode falhar, mas os casos de recuperação são programados no *SAMPLER* para lidar com essas situações [51].

Caso 4: O *MASTER* envia dados inesperados ou inválidos. Esse erro pode surgir de uma falha causada por um SEU. Nesse caso, um *bit-flip* ocorre na memória do *MASTER* ou no caminho de dados, que foi provocado por uma partícula radioativa, pode induzir o MCU *MASTER* a produzir valores de dados inesperados ou inválidos. Todavia, o *MASTER* é programado com técnicas de *Recovery Blocks* [50][51], que força os dados a passarem por um teste de validação, antes de fornecer a atitude correta para a saída do sistema. O valor armazenado no MCU *MASTER* será testado e, em caso de sucesso na validação, será usado como última atitude válida. Mesmo se uma nova falha ocorrer no mesmo MCU *MASTER*,

provocando erros no envio do *heartbeat*, o MCU *SAMPLER* detecta esse comportamento e assume a função de *MASTER* [51].

Caso 5: O *MASTER* recebe dados de atitude errada fornecidos supostamente por *SAMPLER* com falha. Nesse caso, o dado recebido não passa no teste de validação do algoritmo implementado no *MASTER*, sendo o valor de atitude armazenado no *MASTER* considerado correto, devido às interações que aconteceram previamente, e enviado para a saída do sistema. O *MASTER* tolera esse tipo de erro apenas uma vez, em outros casos, ele reinicia do *SAMPLER* com falha e desperta um novo *SAMPLER* [51].

Caso 6: O *MASTER* entra em *loop* infinito devido a um defeito geral no seu sistema. Esse defeito pode ser provocado por um SEU no registrador contador de programa e no ponteiro da pilha (*Stack*). Também nesse caso o *SAMPLER* pode identificar o comportamento inadequado do *MASTER* por inatividade e tomar o controle da situação tornando-se o novo *MASTER* [51].

Caso 7: O *MASTER* perde seu modo de configuração durante a fase de inicialização. Essa situação pode ocorrer no mecanismo de decisão do *software* programado no microcontrolador devido a uma memória de programa corrompida por um SEU. Nessa situação, o computador principal deve intervir, reiniciando o *MASTER* com falha e fornecendo uma reconfiguração, colocando ordem no funcionamento do sistema [51].

- **Casos possíveis de falhas no modo *SAMPLER*:**

Caso 1: O *SAMPLER* não responde ao comando de acordar do *MASTER*. Isso pode ser originado por uma falha nos registradores responsáveis pela comunicação ou um atraso do *SAMPLER*. Nesse caso, o *MASTER* fornece a atitude para a saída do sistema, reinicia o *SAMPLER* e acorda o *SLEEPER*, que se torna automaticamente o novo *SAMPLER* do conjunto de tolerância a falhas [51].

Caso 2: O *SAMPLER* sofre um atraso no seu funcionamento ou simplesmente para de funcionar, retardando o envio da atitude calculada para o *MASTER*. Ao detectar a inatividade do *SAMPLER*, o *MASTER* reinicia o antigo parceiro e inicia o processo para despertar um novo *SAMPLER* [51].

Caso 3: Os detectores do *SAMPLER* param de funcionar. Nesse caso, a falha pode provocar um problema grave no sistema e parar de fornecer a atitude correta. O *MASTER* detecta a inatividade do *SAMPLER*, uma vez que ele espera receber uma informação periodicamente, através do barramento da UART2. O *SAMPLER* antigo é reiniciado e o *MASTER* acorda o *SLEEPER*, que se torna automaticamente o novo *SAMPLER* do conjunto de tolerância a falhas [51].

Caso 4: O *SAMPLER* produz valores de dados errados ou inválidos. Nesse caso a falha é detectada pelo *MASTER* no teste de validação da atitude recebida. O *SAMPLER* antigo é reiniciado e o *MASTER* acorda o *SLEEPER*, que se torna automaticamente o novo *SAMPLER* do conjunto de tolerância a falhas [51].

Caso 5: O *SAMPLER* entra em *loop* infinito devido a uma falha geral do seu sistema. Nesse caso, o *MASTER* pode identificar o comportamento inesperado do *SAMPLER* por sua inatividade e tomar o controle da situação [51].

Caso 6: O *SAMPLER* perde o seu modo de configuração. Isso pode ocorrer no mecanismo de decisão do *software* programado no MCU, devido a uma memória de programa corrompida. Nessa situação a CPU principal deve intervir, reiniciando e colocando ordem no funcionamento do sistema [51].

- **Casos possíveis de falhas no modo *SLEEPER*:**

Caso 1: O *SLEEPER* não responde o comando do *MASTER*. O *MASTER* detecta o comportamento inadequado, por meio do barramento da comunicação UART2 e reinicia o *SLEEPER* [51].

Caso 2: O *SLEEPER* para de funcionar. O *MASTER* também detecta essa operação inadequada e reinicia o microcontrolador [51].

Caso 3: O *SLEEPER* entra em *loop* infinito devido a uma falha geral do sistema. Essa situação é também identificada pelo *MASTER* que segue o mesmo procedimento descrito nos casos 1 e 2 [51].

Caso 4: O *SLEEPER* perde seu modo de configuração. Essa situação é também identificada pelo *MASTER*, que segue o mesmo procedimento descrito nos casos 1 e 2 [51].

Nesse capítulo realizou-se a descrição do SDATF que foi submetido às sessões experimentais com o sistema PORTHES, com a finalidade de atestar a reação do seu comportamento diante da emulação dos SEUs. No capítulo 5, estão descritos os testes realizados e resultados obtidos usando a ferramenta desenvolvida nesse trabalho para validar a solução de tolerância a falhas implementada no SDATF.

Capítulo 5 - DESCRIÇÃO DOS TESTES DE VALIDAÇÃO DO SISTEMA PORTHES, RESULTADOS E DISCUSSÕES.

5.1. Testes Realizados

Os testes realizados, usando o sistema PORTHES baseado na metodologia CEU de Velazco et al. em [41], foram executados com o microcontrolador, PIC24FJ64GA002 do fabricante Microchip, gravado com o *firmware* de determinação de atitude para satélites artificiais de baixa órbita [48]. Essas sessões de testes tiveram como objetivo a validação do sistema PORTHES (*PORtable Testbed for Harsh Environment of Single event upset*) desenvolvido para gerar e injetar falhas, do tipo *Single Event Upset*, além de detectar e analisar os erros causados no DUT que foi submetido aos testes.

O sistema de determinação de atitude em satélites é necessário para controlar a orientação do satélite no espaço, quando ele é colocado em órbita para executar diversas funções, como coleta e transmissão de dados para a Terra [52]. Um sistema de determinação de atitude tem com função definir a orientação do satélite em relação a um referencial inercial ou algum objeto de interesse específico, por exemplo, a Terra [53]. O sistema, que foi utilizado na primeira e na segunda sessão de testes, fornece a atitude na forma de *quaternion* (um vetor de quatro termos representados em ponto flutuante de precisão simples), a cada 300ms, via comunicação serial UART como saída para um sistema de controle de atitude. O *firmware* de determinação de atitude gravado no PIC24F foi desenvolvido na linguagem de programação C, com alguns trechos de código em linguagem *Assembly*, quando foi necessário realizar o remapeamento dos periféricos do microcontrolador. A IDE usada para programar o *firmware* foi o *software* MPLAB versão 8.30 instalado com um compilador C30 versão 3.31 do fabricante MICROCHIP.

As primeiras sessões de testes, no *firmware* de determinação de atitude, foram realizadas para identificar os elementos de memória sensíveis a SEUs, denominada de mapeamento da área sensível (seção 3.1.2, capítulo 3). O objetivo dessa bateria de testes é submeter todos os elementos de memória do referido microcontrolador COTS (registradores de função especial, espaço utilizado da memória interna, contador de programa, registrador de instruções, pilha, *etc*.), que são usados pelo *firmware* de determinação de atitude para satélites artificiais de baixa órbita, a sessões de emulação do comportamento de falhas e identificar quais desses elementos seriam sensíveis a SEU. Esse teste é chamado de teste de avaliação.

A identificação dos elementos de memória sensíveis a SEUs é realizada configurando um trecho de código na rotina de tratamento de interrupção por mudança de nível do DUT com um único alvo, apenas o elemento de memória a ser submetido aos testes, emulando um *bit-flip*, que é o comportamento de uma falha injetada no dispositivo. Em seguida, *upsets* são emulados, por meio de uma mudança de nível lógico no pino sensível a interrupção por mudança de nível lógico habilitada no DUT, enquanto as saídas do sistema de determinação do DUT que produzem a atitude a cada trezentos milissegundos devem ser monitoradas para detectar possíveis erros observados. Desse modo, pode-se determinar qual(is) o(s) elemento(s) de memória produzem erros e de qual tipo, definindo assim os CEU *targets*. Consequentemente, é possível configurar apropriadamente o trecho de código CEU *code* na rotina de tratamento de interrupção da aplicação gravada no DUT e submetê-lo às sessões de testes para determinar a *cross-section* semi-experimental e a SER semi-experimental a ser comparada. A configuração do CEU *code* é realizada escolhendo os alvos que produziram cem por cento um tipo de erro. Não são usados alvos, que causaram dois ou mais tipos de erros, ou os que causaram erros observáveis inferiores a cem por cento, por se tratarem de alvos que poderiam influenciar para uma SER muito menor do que a SER esperada no processo de emulação. Daí a importância da investigação dos efeitos gerados por cada elemento de memória. A quantidade de alvos no CEU *code* vai depender do tipo de experimento que se quer realizar ou reproduzir e também da precisão da SER que se quer alcançar no experimento.

Após a identificação dos elementos de memória sensíveis a SEUs e a constatação do tipo de erro(s) produzido(s) quando esses elementos são submetidos a inversões de *bits*, ambas concluídas na primeira sessão de testes, foi possível configurar o CEU *code* e realizar a segunda sessão experimental. A segunda bateria de testes foi realizada com o CEU *code*, configurado juntamente com a aplicação de determinação de atitude gravada na memória do MCU a ser qualificado. Teve como objetivo confrontar os resultados experimentais apresentados em [43] por Velazco et al. com os resultados obtidos usando a ferramenta projetada nesse trabalho, sem a necessidade de submeter inicialmente o DUT a um processo de exposição à radiação para a obtenção da *cross-section* do programa, e assim validar o método proposto nesse trabalho. Com os dados dessa sessão de testes foi possível calcular os percentuais de cada tipo de erro, o percentual da quantidade de *upsets* que resultaram em SEUs (τ_{CEU}), a *cross-section* do semi-experimental ($\sigma_{SEU\ (semi-experimental)}$) e determinar a *Soft Error Rate* (SER) do dispositivo microcontrolado.

Como meta final dessa etapa, executaram-se as sessões de testes de emulação de falhas propostas com a ferramenta, que reproduz a metodologia baseada em CEU de Velazco et al. [41], desenvolvida nesse trabalho, para a validação da solução do sistema de determinação de atitude para satélites artificiais de baixa órbita com tolerância a falhas, que foi construído a partir do mesmo microcontrolador COTS usado na primeira e na segunda sessões de testes.

5.2. Resultados dos testes

A validação da reprodução da metodologia CEU de Velazco et al. [41], desenvolvida nesse trabalho, foi comprovada através dos testes descritos na seção 5, nos quais o DUT foi submetido às sessões de emulação do comportamento de falhas em circuitos COTS, baseadas em inversões de *bits* (*bit-flips*), introduzidas no microcontrolador por chamada de interrupção por mudança de nível lógico. Os resultados dessas sessões de testes, descritos a seguir, foram realizados com o MCU PIC24FJ64GA002. Para simplificar a nomenclatura será usado, daqui por diante, apenas PIC24F para designar o microcontrolador usado nos testes. Os alvos CEU para esse microcontrolador foram os 806 *bytes* de memória de dados ocupados pelo *firmware* de determinação de atitude (Figura 28), sendo o espaço total da memória de dados igual a 8 B*ytes* (Figura 29, destacada em amarelo e verde). Através desses 806 *bytes* é possível calcular o valor percentual de memória utilizada pelo *firmware* de determinação de atitude, por meio da equação (7):

$$percentual = \frac{bytes\ de\ memória\ utilizados\ pelo\ firmware}{bytes\ de\ memória\ total\ do\ DUT} \tag{7}$$

$$percentual = \frac{806}{8192} \approx 0{,}0984 \approx 9{,}84\%$$

Nesse espaço de memória de dados ocupado pelo *firmware* incluí as variáveis locais e globais e os 45 registradores de função especial (do inglês SFRs, *Special Function Registers*) de um total de 221 registradores de 16 *bits* cada, que ocupam os endereços 0000h até 07FEh da memória de dados (Figura 29, destacada em amarelo).

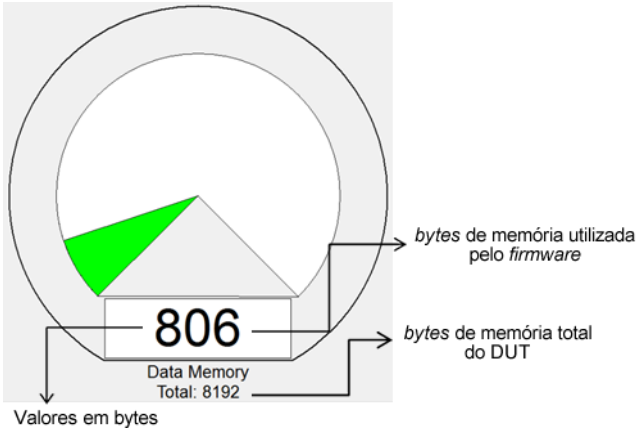


Figura 28. Área ocupada da memória de dados pelo *firmware* de determinação de atitude.

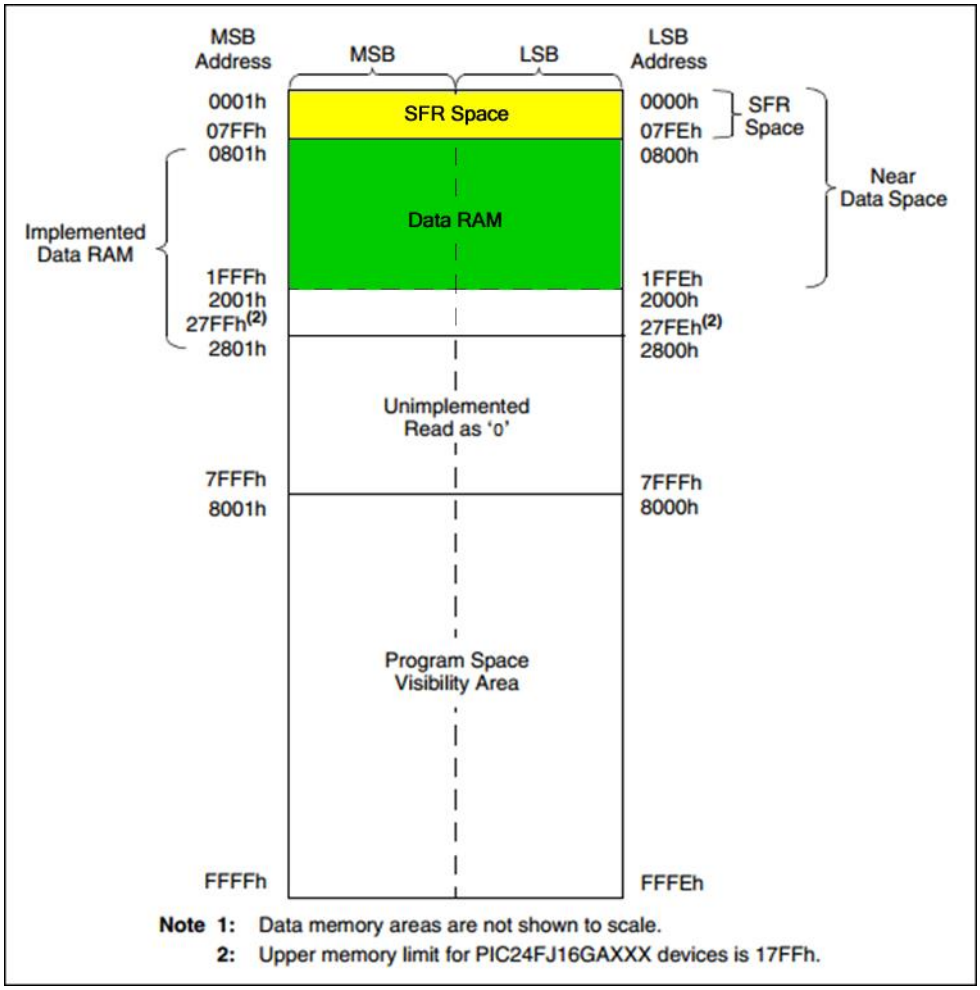


Figura 29. Organização da memória de dados para dispositivos da família do PIC24F. Figura extraída de [54].

Os resultados dos testes serão apresentados em três partes: a primeira parte mostra o mapeamento da área sensível do DUT em teste, gravado com o *firmware* de determinação de atitude. A segunda parte apresenta o cálculo da *cross-section* predita e a semi-experimental e

da SER semi-experimental do DUT, tomando como base a *cross-section* estática obtida no experimento apresentado por Velazco et al. em [43]. Na terceira parte, são expostos os resultados obtidos ao submeter o sistema de determinação de atitude com tolerância a falhas (SDATF) à reprodução da metodologia CEU descrita no trabalho e a comparação dos mesmos com o experimento de Velazco diante de procedimentos semelhantes.

5.2.1. Mapeamento da Área Sensível do DUT (Testes de avaliação)

As primeiras sessões experimentais realizadas no DUT foram os testes de avaliação, que auxiliaram no mapeamento da área sensível a SEUs do dispositivo a ser testado. Os testes de avaliação são importantes, pois através deles é possível diagnosticar os efeitos causados pela emulação do comportamento de falhas, por meio de interrupções programadas no *firmware* do dispositivo, em circuitos COTS em um determinado alvo (CEU *targets*, [43]). E, desse modo, definir quais elementos de memória produzem erros detectáveis nas saídas do sistema, quando ocorre alguma mudança, em ao menos um *bit* desses elementos, causada pela emulação do comportamento de falhas no DUT.

O objetivo dos testes de avaliação, dentro do método proposto no trabalho, que reproduz metodologia baseada em CEU, é identificar os elementos de memória que provocam cem por cento erros do tipo *Result Errors* ou perda de sequência. Isso porque esses elementos de memória, necessariamente, serão escolhidos como alvos (CEU *targets*) para configurarem a montagem do CEU *code* no experimento, descrito na próxima seção de emulação do comportamento de falhas usando o sistema PORTHES.

Os 45 SFRs e os 806 *bytes* de espaço da memória de dados usados pelo *firmware* de determinação de atitude foram submetidos às sessões de emulação de falhas dos testes de avaliação. Dessa forma, foi possível observar o efeito produzido pela emulação do comportamento de falhas no alvo escolhido e computar o tipo de erro propagado na saída do sistema testado. Nesse modelo de teste, o número de injeções de *upsets* pode ser superior ao número de dados recebidos (erros computados), pois número de falhas injetadas nesse experimento é em média 2,3 falhas por trinta milissegundos (cerca de 80 vezes por segundo) obedecendo à distribuição de Poisson. Entretanto, a taxa de observação das saídas do mesmo experimento, característica da aplicação do sistema de determinação de atitude é em média 3,3 vezes por segundo, daí justificando-se a quantidade de SEUs injetados se erros ser muito maior que o número de erros observados. Através desse procedimento, é possível identificar e

classificar o tipo de erro produzido por cada elemento de memória caracterizado como alvo nos testes de avaliação.

O *firmware* desenvolvido de determinação de atitude para satélites artificiais de baixa órbita utiliza dois periféricos do MCU PIC24F. São eles: um periférico de comunicação serial UART (o MCU citado possui dois periféricos UART, o *firmware* de cálculo de atitude usa apenas a UART1) e um temporizador (o MCU citado possui cinco contadores/temporizadores, o *firmware* usa apenas o TIMER1). O primeiro periférico é responsável pela transmissão da atitude calculada e o último é responsável por temporizar a janela de envio da atitude para a CPU principal a cada trezentos milissegundos para o sistema PORTHES, o computador onde o PORTHES esta em funcionamento faz esse papel.

5.2.1.1. Testes com os registradores do periférico de comunicação UART1

Os testes de avaliação foram iniciados pelos registradores usados pelo periférico de comunicação serial UART, mais especificamente a UART1. Esses registradores são responsáveis pela integridade dos dados enviados para a CPU principal e pela correta configuração do periférico de comunicação serial. Os SFRs ligados diretamente à comunicação serial UART são: U1BRG, U1RXREG, U1TXREG, U1MODE, U1STA, IEC0, RPOR2, IFS0, IPC2 e IPC3.

O registrador 16-*bit* U1BRG controla o período de um temporizador de 16 *bits*, indicando a taxa de transmissão da comunicação serial UART (*UART Baud Rate register*) [54]. O valor decimal do registrador U1BRG configurado para funcionar no *firmware* de cálculo de atitude é 25, de acordo com a equação disponível no *datasheet* do MCU, esse valor define a taxa de transmissão para 9600 bps (do inglês, bps, *bits per second*) [54]. Os testes de avalição realizados com esse registrador modificaram a taxa de transmissão para valores mostrados na Tabela 1.

Tabela 1. Resultados dos testes de avaliação no registrador U1BRG.

***Upset no* Registrador U1BRG* - Valores em decimal (Taxa de transmissão)**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
			TE	**RE**	**SL**	**TE**	**RE**	**SL**
U1BRG = 0 (500000 bps)	11288	480	0	480	0	0	100%	0
U1BRG = 7 (28800 bps)	11304	480	0	480	0	0	100%	0
U1BRG = 12 (19200 bps)	11252	480	0	480	0	0	100%	0
U1BRG = 51 (4800 bps)	11282	480	0	480	0	0	100%	0
U1BRG = 65535 (8 bps)	11095	393	0	197	196	0	49,9%	50,1%
U1BRG ^= mascara	10943	430	27	267	136	6,3%	62,1%	31,6%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados obtidos nos experimentos com o registrador U1BRG mostram que a mudança na taxa de transmissão provoca alteração no padrão de dados enviados para a CPU principal, resultando erros do tipo *Result Errors*. Isso ocorre porque as taxas de transmissão e recepção entre os dispositivos devem ser iguais, para não ocorrer perda de sincronismo no processo de transmissão/recepção de dados [54, pp. 17-18]. A exceção foi quando alteramos para o valor 65535 e usamos a máscara, que apresentaram *Tolerated Errors* e *Sequence Loss*.

O registrador 16-*bit* U1TXREG é um *buffer* com a finalidade de armazenar os dados transmitidos pelo periférico de comunicação serial UART [54]. O conteúdo de cada um desse registrador foi alterado com valores fixos e em outras situações de teste com modificações em posições fixas (*bits* específicos) do registrador, aplicando-se lógica *XOR* com uma máscara pré-programada para alterar cada um dos 16 *bits* do registrador que se quisesse modificar. Os resultados são apresentados na Tabela 2.

Tabela 2. Resultados dos testes de avaliação no registrador U1TXREG.

Upset no* Registrador U1RXREG e U1TXREG	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
			TE	**RE**	**SL**	**TE**	**RE**	**SL**
U1TXREG = 0x0000	7547	1080	21	1059	0	1.9%	98.1%	0
U1TXREG = 0xFFFF	7784	1075	28	1047	0	2.6%	97.4%	0
U1TXREG ^= mascara	7649	1082	27	1055	0	2.5%	97.5%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

A alteração dos dados no registrador U1TXREG, responsável pela transmissão dos dados para a CPU principal, gerou na maioria erros do tipo *Result Errors* (RE) (por volta de 98%), pois ao modificar esse registrador, dados diferentes do padrão esperado são enviados.

Os poucos 2% podem ser explicados, porque em determinado momento a função responsável por enviar a atitude calculada para a CPU principal sobrepôs o efeito causado pelo *upset* e o valor da atitude foi enviado sem a presença de erros.

O registrador 16-*bit* U1MODE é responsável pelo modo de configuração da comunicação UART, como por exemplo: habilitar/desabilitar o periférico de comunicação UART, selecionar a paridade, entre outras configurações [54]. Os testes de avaliação no U1MODE foram realizados alterando a informação completa e *bit*-a-*bit* do registrador. O objetivo do segundo experimento foi verificar quais *bits* eram responsáveis pelos *Result Errors* no registrador citado. A consequência dos testes realizados com o registrador U1MODE completo e com os *bits* de função específica desse registrador é mostrada na Tabela 3.

Tabela 3. Resultados dos testes de avaliação no registrador U1MODE.

Upset no* Registrador U1MODE		**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
					TE	**RE**	**SL**	**TE**	**RE**	**SL**
Registrador completo										
U1MODE = 0x0000			11550	480	0	480	0	0	100%	0
U1MODE = 0xFFFF			11282	480	0	480	0	0	100%	0
U1MODE ^= mascara			11109	480	0	480	0	0	100%	0
Registrador *bit*-a-*bit*										
Nº	**Nome**	**Operação realizada**	**CEU injetados**	**Total**	**TE**	**RE**	**SL**	**TE**	**RE**	**SL**
15	UARTEN	UARTEN = 0	11311	482	0	482	0	0	100%	0
		UARTEN ^= mascara	11376	481	0	481	0	0	100%	0
12	IREN	IREN = 1	11512	480	0	480	0	0	100%	0
		IREN ^= mascara	8017	1040	3	1037	0	0.3%	99.7%	0
3	BRGH	BRGH = 1	11357	480	0	480	0	0	100%	0
		BRGH ^= mascara	11093	480	23	457	0	4.8%	95.2%	0
1-2	PDSEL	PDSEL = 01	11280	480	0	480	0	0	100%	0
		PDSEL = 10	11482	480	0	480	0	0	100%	0
		PDSEL = 11	11414	480	0	480	0	0	100%	0
0	STSEL	STSEL = 1	11415	480	0	480	0	0	100%	0
		STSEL ^= mascara	11341	480	230	250	0	47.9%	52.1%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Ao alterar os *bits* do registrador U1MODE, os resultados observados mostram erros do tipo *Result Errors*, porque a mudança na informação desses *bits* altera os dados enviados para a CPU principal, modificando-os do padrão esperado, como pode ser visto na Tabela 3, com os *bits*: UARTEN<15> (Desabilita/Habilita o periférico de comunicação serial UART),

IREN<12> (Desabilita/Habilita a associação de dados infravermelhos), BRGH<3> (Desabilita/Habilita a alta taxa de transmissão), PDSEL<2:1> (Altera a paridade dos dados enviados) e STSEL<0> (Altera o número de um *Stop bit* enviados nos dados). Esses *bits* do registrador U1MODE são responsáveis pela integridade dos dados enviados para a CPU principal [54]. Assim, nos casos mostrados na Tabela 3, os *upsets* injetados nesses alvos provocaram erros do tipo *Result Error*, porque o ataque a esses *bits* causaram uma divergência entre o valor enviado e o padrão esperado pelo sistema PORTHES.

Outros *bits* do registrador U1MODE (U1MODE <9:4>, U1MODE<11> e U1MODE<13>) não são mostrados na Tabela 3, porque os testes realizados com eles resultaram em cem por cento de erros toleráveis (*Tolerated Errors*). E como o interesse é nos *upsets* que propagam as falhas injetadas e geram erros perceptíveis na saída do dispositivo, não houve necessidade de abordar esses resultados. Já os *bits* U1MODE<10> e U1MODE<14> não possuem função específica no PIC24F, por isso não se justifica a realização de testes nesses *bits* [54].

A partir dos resultados mostrados na Tabela 3 com os testes de avaliação no registrador U1MODE pode-se concluir os seguintes pontos. Os erros do tipo *Result Errors* são mais predominantes se comparados aos erros do tipo toleráveis (*Tolerated Errors*). Isso indica a grande sensibilidade desse registrador a uma grande diversidade de posições (inversões de *bits*) que possa ocorrer caso ocorra uma falha por SEU. Dessa análise pode-se concluir com um percentual próximo a cem por cento que a falha provocada através de uma inversão de *bit* desse registrador, irá emular um *Result Error* que contribuirá para a contabilização da SER do emulador. Por fim pode-se concluir que esse registrador é um elemento de memória muito sensível a SEU para a aplicação/DUT em questão.

Situações semelhantes de predominância de *Result errors* sobre erros toleráveis foram observadas em testes realizados com outros registradores do microcontrolador PIC24F gravado com o sistema de determinação de atitude.

Um comportamento particular pode ser observado nos resultados da Tabela 3 e diz respeito ao *bit* STSEL (*STOP Selection Bit*) do registrador U1MODE, responsável por configurar o número de *stop bits* da transmissão serial assíncrona. Quando a inversão de *bit* afeta esse *bit* do registrador U1MODE pôde-se observar que em 47,9% das injeções provocadas foram contabilizadas como *Tolerated Errors*, isto é, não interferiram no valor da

atitude recebida e comparada com o padrão de atitude programada na IG do PORTHES. De certa forma esse comportamento seria previsível, pois o número de *stop bits* pode pouco interferir no processo de comunicação se houver um espaçamento temporal suficientemente grande entre a recepção de um caractere e outro subsequente.

O registrador 16-*bit* U1STA é responsável pelo controle e exibição do *status* do periférico de comunicação serial UART, como por exemplo, selecionar o modo de transmissão por interrupção, inserir um *bit* de parada na transmissão dos dados, entre outras funções de controle e *status* [54]. O registrador U1STA foi submetido aos mesmos testes realizados com o registrador U1MODE. Pela Tabela 4, são apresentados os resultados obtidos nos testes de avaliação com o registrador U1STA completo e com os *bits* de função específica desse registrador.

Tabela 4. Resultados dos testes de avaliação no registrador U1STA.

Upset no* Registrador U1STA	**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
				TE	**RE**	**SL**	**TE**	**RE**	**SL**
Registrador completo									
U1STA = 0x0000		11366	482	65	417	0	13.5%	86.5%	0
U1STA = 0xFFFF		11388	480	0	480	0	0	100%	0
U1STA ^= mascara		11310	480	0	478	2	0	99.6%	0.4%
Registrador *bit*-a-*bit*									
Nº	**Nome**	**Operação realizada**	**CEU injetados**	**Total**	**TE**	**RE**	**SL**	**TE**	**RE**	**SL**
14	UTXINV	UTXINV = 1	11459	480	0	480	0	0	100%	0
		UTXINV ^= mascara	8111	1005	8	997	0	0.8%	99.2%	0
11	UTXBRK	UTXBRK = 1	11556	480	75	405	0	15.6%	84.4%	0
		UTXBRK ^= mascara	11668	480	59	421	0	12.3%	87.7%	0
10	UTXEN	UTXEN = 0	11317	480	86	394	0	17.9%	82.1%	0
		UTXEN ^= mascara	7898	999	8	991	0	0.8%	99.2%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados apresentados na Tabela 4 mostram que a alteração dos *bits* do registrador U1STA provoca discrepâncias nos dados recebidos, computados e percebidos como *Result Errors* em certas situações. Esse registrador possui um *bit* muito sensível que modifica diretamente a informação enviada para a CPU principal, trata-se do *bit* de configuração UTXINV<14>, que é responsável pela inversão de polaridade do estado *Idle* da linha de dados. Entretanto o *bit* UTXBRK<11> que em combinação com UTXEN, participa

da habilitação de um modo de transmissão particular constituído de um *start bit*, seguido por doze caracteres ‘0’ e um *stop bit* provoca entre 84,4% e 87,7% de *Result Errors* quando afetado. Pode-se dizer que esse percentual não é cem por cento, pois há momentos em que o sinal UTXEN é desabilitado dentro da aplicação de determinação de atitude e nesses momentos tal comportamento singular dessa forma de transmissão não afeta a transferência de dados. No caso de desabilitar a transmissão, através do *bit* de configuração UTXEN<10>, observou-se o mesmo efeito verificado no *bit* de controle UARTEN<15> do registrador U1MODE [54].

Os *bits* do registrador U1STA (U1STA<9:0>, U1STA<13> e U1STA<15>) não são mostrados na Tabela 4, pois os testes de avaliação realizados nesses *bits* resultaram cem por cento em erros toleráveis (*Tolerated Errors*) e o *bit* U1STA<12> não possui função específica no PIC24F, logo não foi testado.

Os registradores 16-*bit* IECx (do inglês, *Interrupt Enable Control*) do MCU PIC24F possuem os *bits* de habilitação das interrupções dos periféricos. Estes *bits* de controle são usados para habilitar ou desabilitar interrupções individualmente dos periféricos e de sinais externos [54]. Em simulações de *upsets* com os bits desse registrador, especial atenção foi observada quando se realizou testes com o *bit* IEC0<11>, responsável por habilitar e desabilitar a interrupção da recepção do módulo de comunicação UART1 do MCU, ele é chamado de U1RXIE [54] no PIC24F. Os resultados dos testes sobre esse registrador são mostrados na Tabela 5.

Tabela 5. Resultados dos testes de avaliação no bit de configuração IEC<11>.

Upset no* Registrador IEC0		**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
					TE	**RE**	**SL**	**TE**	**RE**	**SL**
Nº	**Nome**	**Registrador *bit*-a-*bit***								
11	U1RXIE	U1RXIE = 1	7647	1077	0	1077	0	0	100%	0
		U1RXIE ^= mascara	7501	1080	0	1080	0	0	100%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Apesar do periférico de comunicação serial UART no *firmware* de determinação de atitude não estar configurado para receber dados e não usar nenhum registrador para essa funcionalidade, nos testes de avaliação, detectou-se um comportamento pouco previsível na emulação de *upsets* no *bit* que habilita a interrupção do módulo receptor na comunicação

serial UART do MCU PIC24F. A documentação oficial disponibilizada pelo fabricante (Manuais de referência e *datasheets* do produto) não possui informação suficiente para se explicar tal comportamento e a simulação, através do MPLAB SIM [56], apontou evidências, que poderiam explicar esse comportamento. Pela experiência em arquitetura de microcontroladores e na sua programação é possível admitir que esse comportamento tenha sido provocado por uma alteração no fluxo de execução do *firmware* para algum endereço da tabela de interrupções, pois se presume que ao habilitar a interrupção do módulo receptor na comunicação serial UART, sem configurar todos os *bits* dessa interrupção na sequência recomendada pelo *datasheet* do fabricante [54] tal efeito possa interferir no fluxo de execução da aplicação em curso. Portanto o fluxo do programa pode se perder nos endereços da tabela de interrupções à procura da rotina de tratamento de interrupção (do inglês, *Interrupt Request Service*, ISR) referente à recepção da UART, atrasando o cálculo da atitude. Entretanto, para comprovar essa afirmação deveria ser realizada uma análise mais aprofundada dessa situação com ajuda de equipamentos de depuração de sinais a fim de se visualizar o problema.

Esse comportamento descrito no parágrafo anterior comprova a afirmativa da impossibilidade da completa automação do processo de mapeamento da área sensível nos experimentos de emulação do comportamento de falhas sem uma análise detalhada de cada elemento de memória sensível a SEU no conjunto DUT-aplicação em teste, pois como o código não utiliza o módulo de recepção da UART1 esperava-se que *bit-flips* nesses registradores não afetassem o funcionamento do sistema.

O MCU PIC24F possui uma particularidade inerente de remapeamento de periféricos, permitindo a liberdade de associar pinos de E/S a múltiplos periféricos e flexibilizando assim a construção do *hardware* [54]. Os registradores 16-*bit* RPORx (*Peripheral Pin Select Output Register*) são usados para o controle do mapeamento das saídas dos periféricos. Para melhor entendimento, o RPOR2 usado no *firmware* em teste é responsável pelo mapeamento do pino de transmissão (U1TX) da comunicação serial UART, mais especificamente os *bits* 8 ao 12 (RP5R) [54]. Esse registrador foi submetido aos mesmos testes de avaliação realizados com os dois registradores apresentados anteriormente nessa seção. A Tabela 6 mostra os resultados obtidos nos testes de avaliação com o registrador RPOR2 e com os *bits* de função específica desse registrador.

Tabela 6. Resultados dos testes de avaliação no registrador RPOR2.

Upset no* Registrador RPOR2	**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
				TE	**RE**	**SL**	**TE**	**RE**	**SL**
Registrador completo									
RPOR2 = 0x0000		12229	364	0	4	360	0	1.1%	99.9%
RPOR2 = 0xFFFF		12121	361	0	1	360	0	0.3%	99.7%
RPOR2 ^= mascara		8055	1025	7	1018	0	0.7%	99.3%	0
Registrador *bit*-a-*bit*									
Nº	**Nome**	**Operação realizada**	**CEU injetados**	**Total**	**TE**	**RE**	**SL**	**TE**	**RE**	**SL**
8-12	RP5R	RP5R = 0x00	12349	363	0	3	360	0	0.8%	99.2%
		RP5R = 0xFF	12215	361	0	1	360	0	0.3%	99.7%
		RP5R ^= mascara	7903	1022	7	0	1015	0.7%	0	99.3%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados mostram erros de perda de sequência (*Sequence Loss*) ao alterar a informação no registrador RPOR2 com um valor pré-fixado (0x0000 no primeiro teste e 0xFFFF no segundo teste) ou com um valor aleatório (linha com o caso RPOR2 = 0xFFFF da Tabela 6). A justificativa está na modificação do mapeamento do pino transmissor na comunicação serial UART, isso impede o recebimento de algum dado pela CPU principal daí a razão do sistema PORTHES identificar o erro como uma perda de sequência (*Sequence Loss*). Como não há nenhuma reposta do MCU, a interface gráfica computa erros de perda de sequência (*Sequence Loss*); desse modo o *watchdog* do PORTHES atua no DUT provocando um sinal de *reset*, para que o MCU retome seu funcionamento normal.

O registrador 16-*bit* IFSx (*Interrupt Flag Status Register*) possui todas as *flags* de sinalização das interrupções, as quais têm o seu *bit* de *status* que é setado pelo respectivo periférico ou sinal externo [54]. O registrador IFS0 contém a *flag* de sinalização da interrupção do transmissor e receptor da comunicação serial UART1, porém, no *firmware* de determinação de atitude, a comunicação serial UART não trabalha por tratamento de interrupções. Logo, o registrador foi colocado em teste e as mudanças nos seus *bits* resultaram em cem por cento de erros toleráveis (*Tolerated Errors*), não influenciando em nada sobre a aplicação em funcionamento no DUT.

O registrador 16-*bit* IPCx (*Interrupt Priority Control Register*) é usado para definir o nível de prioridade de cada fonte de interrupção, que pode ser associada em até oito níveis de prioridade [54]. O registrador IPC2 contem os *bits* de prioridade da recepção (IPC2<14:12>) e

o IPC3 contém os *bits* de prioridade da transmissão (IPC3<2:0>). Esses registradores foram colocados em teste e as mudanças nesses *bits* resultaram também em cem por cento de erros toleráveis (*Tolerated Errors*), por esse motivo os seus resultados não são mostrados.

5.2.1.2. Testes com os registradores do periférico temporizador 1 (TIMER1)

Finalizados os testes de avaliação com os alvos ligados à comunicação serial UART, passou-se para as sessões experimentais nos registradores utilizados pelo periférico temporizador (especificamente o TIMER1 usado no *firmware* gravado no MCU), pois esses registradores são responsáveis pelo envio temporizado dos dados para a CPU principal. Os SFRs ligados diretamente ao temporizador: T1CON, PR1, TMR1, IEC0, IFS0 e IPC0.

O registrador 16-*bit* T1CON é responsável pelo controle do temporizador 1 (TIMER1), como por exemplo: ligar e desligar o temporizador, configurar o fator de *prescaler* na contagem dos ciclos de *clock* [54]. Os testes de avaliação no T1CON foram realizados alterando a informação completa e *bit*-a-*bit* do registrador. A consequência dos testes realizados com o registrador T1CON completo e com os *bits* de função específica desse registrador é mostrada na Tabela 7.

Tabela 7. Resultados dos testes de avaliação no registrador T1CON.

Upset no* Registrador T1CON		**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
					TE	**RE**	**SL**	**TE**	**RE**	**SL**
Registrador completo										
T1CON = 0x0000			11953	360	0	0	360	0	0	100%
T1CON = 0xFFFF			11920	360	0	0	360	0	0	100%
T1CON ^= mascara			12037	360	0	0	360	0	0	100%
Registrador *bit*-a-*bit*										
Nº	**Nome**	**Operação realizada**	**CEU injetados**	**Total**	**TE**	**RE**	**SL**	**TE**	**RE**	**SL**
15	TON	TON = 0	11929	360	0	0	360	0	0	100%
		TON ^= mascara	11967	360	0	0	360	0	0	100%
6	TGATE	TGATE = 1	11993	360	0	0	360	0	0	100%
		TGATE ^= mascara	11046	384	187	0	197	48.7%	0	51.3%
4-5	TCKPS	TCKPS = 00	7390	1077	33	1044	0	3.1%	96.9%	0
		TCKPS = 01	7652	1079	29	1050	0	2.7%	97.3%	0
		TCKPS = 10	8728	845	838	7	0	99.2%	0.8%	0
1	TCS	TCS = 1	11921	360	0	0	360	0	0	100%
		TCS ^= mascara	11309	360	0	0	360	0	0	100%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os efeitos dos testes de avaliação no registrador T1CON mostram erros do tipo *Result Errors* e perda de sequência (*Sequence Loss*), porque a mudança em uma ou mais informações desse registrador provoca ausência de resposta do dispositivo, não enviando dentro do limite de tempo esperado os dados da atitude do DUT para o PORTHES. Testes com o *bit* TON<15>, que é responsável por habilitar e desabilitar o temporizador de tempo, mostraram que uma vez desabilitado o *timer*, a atitude calculada não é enviada para a CPU principal, resultando em cem por cento de erros por perda de sequência. O *bit* TGATE<6> é responsável por habilitar o modo *Timer 1 Gated Time Accumulation Enable* do MCU. Esse *bit* configura o temporizador com a funcionalidade de medir a largura de um pulso, sem a necessidade de periférico específico [56, pp. 91-98]. Ao alternar sua configuração com uma máscara, ou seja, uma nova funcionalidade do temporizador, a atitude calculada ora era enviada para a CPU principal, ora não. Um percentual de 96,9 % e 97,3% de *Result Errors* foi observado quando os *bits* TCKPS<5:4> foram modificados do padrão normal de funcionamento "10" para respectivamente "00" e "01". Esses bits são responsáveis pelo *prescaler* do temporizador, fazendo com que o periférico conte mais rápido, enviando a atitude incompleta e assim provocando *Result Errors*. Os testes com o *bit* TCS<1> apresentaram cem por cento de erros por perda de sequência, pois esse *bit* é responsável por selecionar a fonte de *clock* do temporizador, interna ou externa [54]. Quando esse bit é desconfigurado, o periférico cessa o envio da atitude para a CPU principal. Os *bits* <3> e <12:7> não possuem funções específicas que afetassem o sistema de determinação de atitude, portanto não foram mostrados na Tabela 7.

Ao submeter o registrador T1CON aos testes de avaliação, destaca-se que grande parte os *Result Errors* foram de perda de sequência, apesar da maioria dos *bits* produzirem outros tipos de erros (*Result Errors* e toleráveis), quando eles foram testados separadamente. Isso demostra que os erros do tipo perda de sequência são mais predominantes se comparados com os outros dois tipos de erros (*Result errors* e toleráveis).

O registrador 16-bit PR1 carrega o período associado ao temporizador. Ele armazena o valor de referência usado pelo comparador para confrontar com o valor do TMR1 e gerar a interrupção quando os conteúdos dos dois registradores são iguais [56, pp. 91-98]. Os testes de avaliação realizados apresentam a modificação do conteúdo desse registrador para diversos valores conforme mostrado na Tabela 8.

Tabela 8. Resultados dos testes de avaliação no registrador PR1.

Upset no Registrador PR1*	Total de *upsets* injetados**	Total de Erros**	Valores absolutos de erros**			Porcentagem de erros**		
			TE	RE	SL	TE	RE	SL
PR1 = 0x0000	12107	360	0	0	360	0	0	100%
PR1 = 0x0250	7617	1054	532	522	0	50.5%	49.5%	0
PR1 = 0x0500	8977	836	774	62	0	92.6%	7.4%	0
PR1 = 0x1000	9634	360	360	0	0	100%	0	0
PR1 = 0x1500	10032	360	360	0	0	100%	0	0
PR1 = 0x1750	11533	360	4	0	356	1.1%	0	99.9%
PR1 = 0x2000	11998	360	0	0	360	0	0	100%
PR1 ^= mascara	10974	380	181	0	199	47.6%	0	52.3%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados dos testes de avaliação da Tabela 8 apresentam os efeitos causados pela alteração dos valores no registrador PR1. O valor configurado como período do temporizador 1 no *firmware* de determinação de atitude é, em hexadecimal, 0x124F, que corresponde ao tempo de 300ms. Valores muito acima do configurado (como por exemplo, 0x1750, 382ms, e 0x2000, 524ms) produzem erros do tipo perda de sequência, porque o temporizador ultrapassa o tempo máximo (300ms) de envio da atitude para a CPU principal. Observa-se também o mesmo tipo de erro computado quando se configura o registrador PR1 com o valor mínimo (0x0000), pois isso significa que a interrupção do temporizador acontece no momento de contagem máxima atingida pelo recurso TMR1 [57, pp. 33-34]. O tempo gasto para o TMR1 atingir a contagem máxima é maior do que o tempo esperado pela CPU principal para receber a atitude, resultando em erros de perda de sequência. Quando configuramos o registrador PR1 com valores próximos ao valor normal esperado pelo firmware de cálculo da atitude (valores inferiores: 0x0250 (38ms), 0x0500 (76ms) e 0x1000 (152ms), ou valores superiores: 0x1500, 344ms), o tipo de erros computados são toleráveis e *Result Errors*, atingindo cem por cento de erros do tipo toleráveis, em alguns casos, para valores próximos ao período configurado. A última linha da Tabela 8 (PR1 ^= mascara) apresenta os resultados obtidos usando uma máscara aleatória construída com a lógica *XOR* para alterar os todos os *bits* do registrador PR1. Observou-se 47,6% de erros do tipo toleráveis e 52,4% de perda de sequência. O primeiro tipo de erro é devido a alguns momentos nos quais a configuração do período está correta para enviar a atitude calculada. O segundo pode ser explicado, porque quando se aplica a máscara *XOR* (*bit flips*) no registrador, o valor do período é alterado para uma quantidade maior do que a configurada, ultrapassando o tempo limite esperado para envio da atitude calculada.

O registrador 16-*bit* TMR1 é um contador associado ao temporizador 1, que é incrementado a cada contagem de ciclo de *clock* de acordo com a configuração do temporizador [54]. Os testes de avaliação realizados com esse contador atribuindo-se diversos valores e alternando-o com o auxilio de uma máscara de *bits* montada com uma lógica *XOR* são mostrados na Tabela 9.

Tabela 9. Resultados dos testes de avaliação no registrador TMR1.

Upset no **Registrador TMR1***	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
			TE	**RE**	**SL**	**TE**	**RE**	**SL**
TMR1 = 0x0050	11825	360	0	0	360	0	0	100%
TMR1 = 0x0100	12017	360	0	0	360	0	0	100%
TMR1 = 0x0500	11797	360	0	0	360	0	0	100%
TMR1 = 0x124F	7565	1080	458	622	0	42.4%	57.6%	0
TMR1 = 0x1500	11869	360	0	0	360	0	0	100%
TMR1 = 0x2000	11926	360	0	0	360	0	0	100%
TMR1 ^= mascara	11830	360	0	0	360	0	0	100%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados dos testes de avaliação mostram erros do tipo perda de sequência, quando o conteúdo do registrador TMR1 foi alterado para valores abaixo ou acima do valor do período programado no temporizador 1 do *firmware* de cálculo da atitude (período programado, PR1 = 0x124F, 300ms). Esse mesmo comportamento também foi observado quando uma máscara de inversão de *bits* aleatória construída com lógica *XOR* foi aplicada nesse registrador. Isso acontece, porque ao atribuir um valor fixo ao registrador TMR1, impede-o de realizar a tarefa de contagem de tempo durante o experimento, assim esse registrador nunca alcança o valor do período previsto pelo *firmware* de cálculo da atitude, postergando a ocorrência da interrupção do temporizador. E por consequência, a não ocorrência da interrupção do temporizador, provoca o estouro do tempo limite de espera da atitude calculada pelo sistema PORTHES [54]. O caso em que o registrador TMR1 foi igual ao período configurado no temporizador (TMR1 = 0x124F), os dados foram transmitidos, pois a interrupção por estouro do limite de contagem do recurso foi sinalizada. Porém, em alguns instantes não houve tempo suficiente para terminar o cálculo da atitude, assim computando erros dos tipos *Result Errors* e *Tolerated Errors* como mostrado na Tabela 9.

O registrador 16-*bit* IEC0 possui um *bit* para habilitar e desabilitar individualmente a interrupção do temporizador 1. O *bit* IEC0<3> é o responsável por essa funcionalidade, sendo esse *bit* denominado de T1IE [54]. Dois tipos de testes de avaliação foram realizados, o

primeiro zerando o *bit* de habilitação da interrupção e outro usando uma lógica *XOR* para alterar a informação contida no *bit* citado. Os resultados obtidos nos testes de avaliação com o *bit* T1IE<3> de função específica do registrador IEC0 podem ser visualizados na Tabela 10:

Tabela 10. Resultados dos testes de avaliação no bit de configuração IEC<3>.

Upset no* Registrador IEC0		**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
					TE	**RE**	**SL**	**TE**	**RE**	**SL**
Nº	**Nome**	**Registrador *bit*-a-*bit***								
3	T1IE	T1IE = 0	12181	360	0	0	360	0	0	100%
		T1IE ^= mascara	10706	360	360	0	0	100%	0	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Pela avaliação dos resultados dos testes no *bit* de configuração da interrupção do temporizador 1 percebe-se que ao desabilitar a interrupção (*upset* injetado T1IE = 0), observou-se cem por cento de erros por perda de sequência. Isso ocorreu porque a interrupção do temporizador foi suspensa e, como o sistema PORTHES não recebe nenhum dado, ele computa essa ocorrência como erro de travamento. Entretanto, a mudança através da lógica *XOR* produziu somente erros toleráveis, pois a injeção de *upsets* alterna o estado do *bit* de configuração, ora a interrupção está habilitada, ora ela está desabilitada. Todavia, o temporizador continua funcionando, incrementando o contador (TMR1), comparando com o valor configurado no período (PR1) e sinalizando a *flag* de interrupção se esses valores são iguais. E em um determinado instante, quando a interrupção foi habilitada e a *flag* de ocorrência da interrupção ficou sinalizada, o tratamento da interrupção acontece, logicamente, enviando a atitude calculada para o PORTHES dentro do período de tempo esperado, justificando o resultado de cem por cento de *Tolerated Errors*.

O registrador 16-*bit* IFS0 (*Interrupt Flag Status Register*) possui a *flag* responsável pela sinalização ao interromper o temporizador 1 (TIMER1). O *bit* T1IF<3> armazena o *status* da ocorrência ou não da interrupção desse temporizador [54]. Os testes de avaliação, que foram realizados alterando a informação desse *bit*, são mostrados na Tabela 11.

Tabela 11. Resultados dos testes de avaliação no bit de sinalização IFS0<3>.

Upset no Registrador IFS0*		Operação realizada	Total de *upsets* injetados**	Total de Erros**	Valores absolutos de erros**			Porcentagem de erros**		
					TE	RE	SL	TE	RE	SL
Nº	Nome	Registrador *bit*-a-*bit*								
3	T1IF	T1IF = 1	7708	1081	438	643	0	40.5%	59.5%	0
		T1IF ^= mascara	7493	1080	451	629	0	41.8%	58.2%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

A mudança na *flag* de sinalização do temporizador 1 produz erros dos tipos *Tolerated* e *Result Errors*, conforme são visualizados na Tabela 11. No momento que a *flag* da interrupção é sinalizada, o dispositivo executa a rotina de tratamento de interrupção do temporizador 1, que é responsável por enviar a atitude para CPU principal, mesmo se o cálculo da atitude estiver terminado ou não. Deste modo, enviando atitudes incompletas (*Result Errors*) ou atitudes corretas (*Tolerated Errors*).

O registrador 16-*bit* IPC0 (*Interrupt Flag Status Register*) possui um *bit* de configuração para definir a prioridade da interrupção do temporizador 1 (TIMER1). Os *bits* T1IP<14:12> têm essa função de definir a prioridade para o atendimento da interrupção do temporizador [54]. Na realização dos testes, o registrador foi alterado zerando os seus *bits* e usando uma lógica *XOR* para modificar a informação contida nos *bits* citados. É possível visualizar os resultados na Tabela 12.

Tabela 12. Resultados dos testes de avaliação nos bits de configuração de prioridade IPC0<12-14>

Upset no Registrador IPC0*		Operação realizada	Total de *upsets* injetados**	Total de Erros**	Valores absolutos de erros**			Porcentagem de erros**		
					TE	RE	SL	TE	RE	SL
Nº	Nome	Registrador *bit*-a-*bit*								
12-14	T1IP	T1IP = 0	11947	360	0	0	360	0	0	100%
		T1IP ^= mascara	11530	360	480	0	0	100%	0	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Constata-se, por meio dos resultados dos testes de avaliação apresentados na Tabela 12, que a ação de zerar os *bits* de configuração de prioridade da interrupção do temporizador 1, produziu cem por cento erros por perda de sequência. Ao se definir a prioridade mínima (zero) a interrupção não é tratada em nenhum momento, impossibilitando o envio da atitude para o PORTHES [54], observado e computado como erro de travamento do sistema. A

alteração através da lógica *XOR* produziu erros toleráveis, porque ao alterar a configuração da prioridade maior que a mínima (zero) e a máxima (sete) usada na configuração normal do *firmware*, a interrupção foi atendida imediatamente por não haver outro tipo de interrupção configurada no firmware de cálculo de atitude. Daí a razão dos 100% de *Tolerated Errors*, fazendo com que a atitude fosse enviada ao PORTHES sempre dentro do limite de tempo esperado.

5.2.1.3. Testes com os outros registradores do *firmware* de determinação de atitude

Além dos registradores usados pelos periféricos da comunicação UART e do temporizador (totalizando 15 registradores), o *firmware* de determinação de atitude utiliza outros 30 registradores de função especial, são eles: 15 registradores de trabalho (do inglês, *work registers*, WREG), CLKDIV, PC, SPLIM, CORCON, CRCCON, OSCCON, RCON, SR, DISICNT, PORTA, PORTB, LATA, LATB, TRISA e TRISB.

Os registradores de trabalho WREG podem armazenar dados, endereços e endereços de registradores de *offset*. Por exemplo, o décimo quinto registrador de trabalho (W14) é usado como ponteiro de *frame* definido pelo usuário da pilha (*Stack*), tem a função de alocar na memória o endereço das variáveis locais (mudança de contexto). O décimo sexto registrador de trabalho (W15) é usado como um ponteiro da pilha (*Stack*), armazenando o endereço de retorno nas chamadas de interrupção [54]. Por razões de implementação de projeto da arquitetura do DUT, o registrador W15 não pode ser acessado pelo conjunto de instruções do microcontrolador [54]. A Tabela 13 expõe os resultados obtidos nos testes de avaliação com os registradores de trabalho que apresentaram algum tipo de erro detectável na saída do sistema (*Result Errors e Sequence Loss*).

Tabela 13. Resultados dos testes de avaliação nos registradores de trabalho.

Upset nos WREGs*	Total de *upsets* injetados**	Total de Erros**	Valores absolutos de erros**			Porcentagem de erros**		
			TE	RE	SL	TE	RE	SL
W3	7674	1038	19	1016	3	1.8%	97.9%	0.3%
W4	11331	480	447	33	0	93.1%	6.9%	0
W5	11446	480	476	4	0	99.2%	0.8%	0
W6	11373	480	336	144	0	70%	30%	0
W7	11274	480	346	134	0	72.1%	27.9%	0
W8	11292	480	331	149	0	68.9%	31.1%	0
W9	10238	649	240	408	1	36.9%	62.9%	0.2%
W10	11465	480	445	35	0	92.7%	7.3%	0
W11	10961	500	239	254	7	47.8%	50.8%	1.4%
W12	11367	480	439	41	0	91.5%	8.5%	0
W14	7574	1078	0	1078	0	0	100%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os registradores de trabalho W0, W1, W2 e W13 apresentaram cem por cento de erros toleráveis e não foram exibidos na Tabela 13. Os testes de avaliação nos registradores de trabalho (W3, ... , W12, W14) apresentam variações nos tipos de erros gerados na injeção de *upsets*, pois esses registradores são usados para armazenar dados temporários de operações realizadas pelo MCU. Por exemplo, os registradores de trabalho podem armazenar o resultado de uma operação realizada pela unidade lógica aritmética (ULA) [54]. Sendo assim, essas alterações nos registradores de trabalho afetam valores relacionados ao cálculo da atitude provocando *Result Errors*. Um caso especial foi notado no registrador W14, *frame* da pilha (STACK), que gerou cem por cento de *Result Errors*. Isso porque, esse registrador é usando para armazenar os endereços dos dados da mudança de contexto nas chamadas de função ou de interrupção [54], portanto zerando ou alterando a informação desse registrador o contador de programa perde a referência do endereço dos dados armazenados, interferindo no cálculo da atitude.

O registrador 16-*bit* CLKDIV (do inglês, *Clock Divider register*) controla características de funcionamento da CPU do microcontrolador, por exemplo, o modo *Doze*, que é um modo de configuração que coloca a CPU em modo reduzido de consumo de energia, enquanto o dispositivo continua executando o código [54]. Nesse modo de operação do microcontrolador a frequência de *clock* da CPU é reduzida, enquanto os periféricos continuam funcionando com a frequência de *clock* normal [56, pp. 48-50]. A sincronização entre os dois sinais de *clock* é mantida, permitindo que os periféricos acessem os SFRs enquanto a CPU

executa o código com velocidade menor. Essa redução na velocidade de processamento da CPU pode introduzir erros de comunicação ou até interrompê-la, quanto uma aplicação necessita de manter uma comunicação síncrona e ininterrupta com outro dispositivo [54].

Os *bits* DOZEN<11> e DOZE<14:12> são responsáveis pela configuração da frequência de *clock*, como padrão, eles são configurados com o valor zero indicando a frequência máxima de configuração do *clock* [54]. Os testes de avaliação realizados no CLKDIV foram realizados, ora atribuindo valores fixos (0x0000 e 0xFFFF) no registrador e ora aplicando uma máscara com a lógica *XOR* para alterar os *bits* desse mesmo registrador. Testes suplementares *bit*-a-*bit* foram realizados fixando o DOZEN<11> com o valor um (habilitado) e variando os valores dos *bits* DOZE<14:12> alterando a configuração do *clock* da CPU do microcontrolador. Os resultados podem ser visualizados na Tabela 14:

Tabela 14. Resultados dos testes de avaliação no registrador CLKDIV.

Upset no* Registrador CLKDIV	**Operação realizada**	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
				TE	**RE**	**SL**	**TE**	**RE**	**SL**
Registrador completo									
CLKDIV = 0x0000		11537	480	480	0	0	100%	0	0
CLKDIV = 0xFFFF		11546	480	480	0	0	100%	0	0
CLKDIV ^= mascara		11563	480	235	245	0	49.9%	51.1%	0
Registrador *bit*-a-*bit*									
Nº	**Nome**	**Operação realizada**	**CEU injetados**	**Total**	**TE**	**RE**	**SL**	**TE**	**RE**	**SL**
11	DOZEN	DOZEN = 1	Habilita especificar a velocidade de *clock* através dos *bits* DOZE<14:12>							
12-14	DOZE	DOZE = 000	11423	480	480	0	0	100%	0	0
		DOZE = 001	9501	480	282	198	0	58,8%	41,2%	0
		DOZE = 010	11476	480	0	480	0	0	100%	0
		DOZE = 011	11433	480	0	480	0	0	100%	0
		DOZE = 100	11531	480	0	480	0	0	100%	0
		DOZE = 101	11607	480	0	480	0	0	100%	0
		DOZE = 110	11462	480	0	480	0	0	100%	0
		DOZE = 111	8598	480	0	480	0	0	100%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Ao diminuir a frequência de *clock* da CPU do MCU e o periférico de envio da atitude (TIMER1) continua funcionando normalmente e os erros do tipo *Result Errors* são gerados, como mostram os resultados dos testes de avaliação apresentados na Tabela 14. Esse comportamento se explica que devido à redução da velocidade de processamento da CPU do

microcontrolador a atitude não foi calculada a tempo de enviá-la para o PORTHES, sendo assim os dados foram enviados com o cálculo incompleto.

O registrador contador de programa (do inglês, *Program Counter*, PC) é um registrador da unidade central de processamento usado no controle da sequência de execução das instruções do microcontrolador [59]. O PC no PIC24F possui 23 *bits* e armazena o endereço da próxima instrução a ser executada. O PC do PIC24F é capaz de endereçar 4M instruções no espaço da memória de programa [59]. O registrador PC não pode ser acessado diretamente pelo conjunto de instrução [54]. Então, os testes de avaliação para emular a manipulação do registrador PC foram realizados usando o auxílio de instruções de desvios incondicionais colocadas dentro da rotina de tratamento de interrupção, causando um desvio forçado de execução do programa para uma região de memória desejada. As instruções usadas foram: GOTO, CALL, RCALL, RETURN e RETFIE, essas instruções podem ser utilizadas para mudar a direção do fluxo de execução do programa [59, pp. 88-95]. A Tabela 15 apresenta os resultados obtidos nos testes de avaliação com as instruções de emulação de SEUs no *Program Counter*.

Tabela 15. Resultados dos testes de avaliação com as instruções de manipulação do Program Counter.

Upset no Program Counter*	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
			TE	**RE**	**SL**	**TE**	**RE**	**SL**
asm ("GOTO 0x000000")	11394	365	0	5	360	0	1.4%	98.6%
asm ("GOTO 0x000002")	7831	1070	0	1070	0	0	100%	0
asm ("GOTO 0x000004")	7453	1081	0	1081	0	0	100%	0
asm ("GOTO 0x000100")	7736	1077	0	1077	0	0	100%	0
asm ("GOTO 0x000200")	10581	626	0	379	247	0	60.5%	39.5%
asm ("GOTO 0x001800")	7708	1076	0	1076	0	0	100%	0
asm ("GOTO 0x008000")	7784	1074	0	1074	0	0	100%	0
asm ("GOTO 0x00F800")	7552	1078	0	1078	0	0	100%	0
asm ("CALL 0x000000")	11385	365	0	5	360	0	1.4%	98.6%
asm ("CALL 0x000002")	7691	1069	0	1069	0	0	100%	0
asm ("CALL 0x000004")	7859	1071	0	1071	0	0	100%	0
asm ("CALL 0x000100")	7674	1073	0	1073	0	0	100%	0
asm ("CALL 0x000200")	10375	480	480	0	0	100%	0	0
asm ("CALL 0x001800")	7708	1069	0	1069	0	0	100%	0
asm ("CALL 0x008000")	7727	1083	0	1083	0	0	100%	0
asm ("CALL 0x00F800")	7647	1072	0	1072	0	0	100%	0
asm ("RCALL 0x000000")	11631	363	0	3	360	0	0.8%	99.2%
asm ("RCALL 0x000002")	7349	1080	0	1080	0	0	100%	0
asm ("RCALL 0x000004")	7709	1068	0	1068	0	0	100%	0
asm ("RCALL 0x000100")	7510	1077	0	1077	0	0	100%	0
asm ("RCALL 0x000200")	9298	791	2	667	122	0.3%	84.3%	15.4%
asm ("RCALL 0x001800")	7708	1077	0	1077	0	0	100%	0
asm ("RCALL 0x008000")	7859	1085	0	1085	0	0	100%	0
asm ("RCALL 0x00F800")	7647	1079	0	1079	0	0	100%	0
asm ("RETURN")	7473	1080	0	1080	0	0	100%	0
asm ("RETFIE")	7598	1077	0	1077	0	0	100%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os endereços de desvio foram escolhidos por serem áreas importantes na organização da memória de programa do microcontrolador PIC24F (Figura 30), tendo como limite da memória de programa até 4M de instruções (espaço até o endereço 7FFFFEh). Esses endereços são divididos em espaço do usuário e espaço de configuração [56, pp. 35-36]:

Espaço do usuário:

- Endereço 000000h até 000003h: vetor de *Reset*;
- Endereço 000004h até 0000FEh: tabela de interrupções;
- Endereço 000100h até 000103h: reservado;
- Endereço 000104h até 0001FEh: tabela alternativa de interrupções;

- Endereço 000200h até o final (endereço limite 7FFFFEh): memória de *flash*.

Espaço de configuração:

- Endereço F80000h até F8000Eh: registradores de configuração (durante a gravação do microcontrolador);
- Endereço FF0000h até FFFFFEh: identificação do microcontrolador.

A mudança no contador de programa, através das instruções de desvio do fluxo de execução do *firmware*, gerou *Result Errors*, exibidos na Tabela 15 na maior parte dos casos testados. Uma vez desviado continuamente o fluxo de programa, o tempo necessário para o cálculo da atitude é afetado. Sendo assim a atitude foi enviada com valores incorretos comparados com o padrão esperado. Duas exceções podem ser notadas na Tabela 15, desvios para o endereço 0x00000 e para o endereço 0x00200 que produziram ambos, erros por perda de sequência. No primeiro caso (endereço 0x00000), a simulação da situação de *Reset* (Figura 30) do MCU é realizada, forçando o contador de programa a ficar na posição zero e travando o sistema [54]. No segundo caso (endereço 0x00200), os erros de travamento do sistema devem-se ao desvio forçado do fluxo de execução do *firmware* para o início da memória de programa (Figura 30), provocando consequentemente a reconfiguração dos periféricos no microcontroladores e impedindo o envio da atitude para o PORTHES.

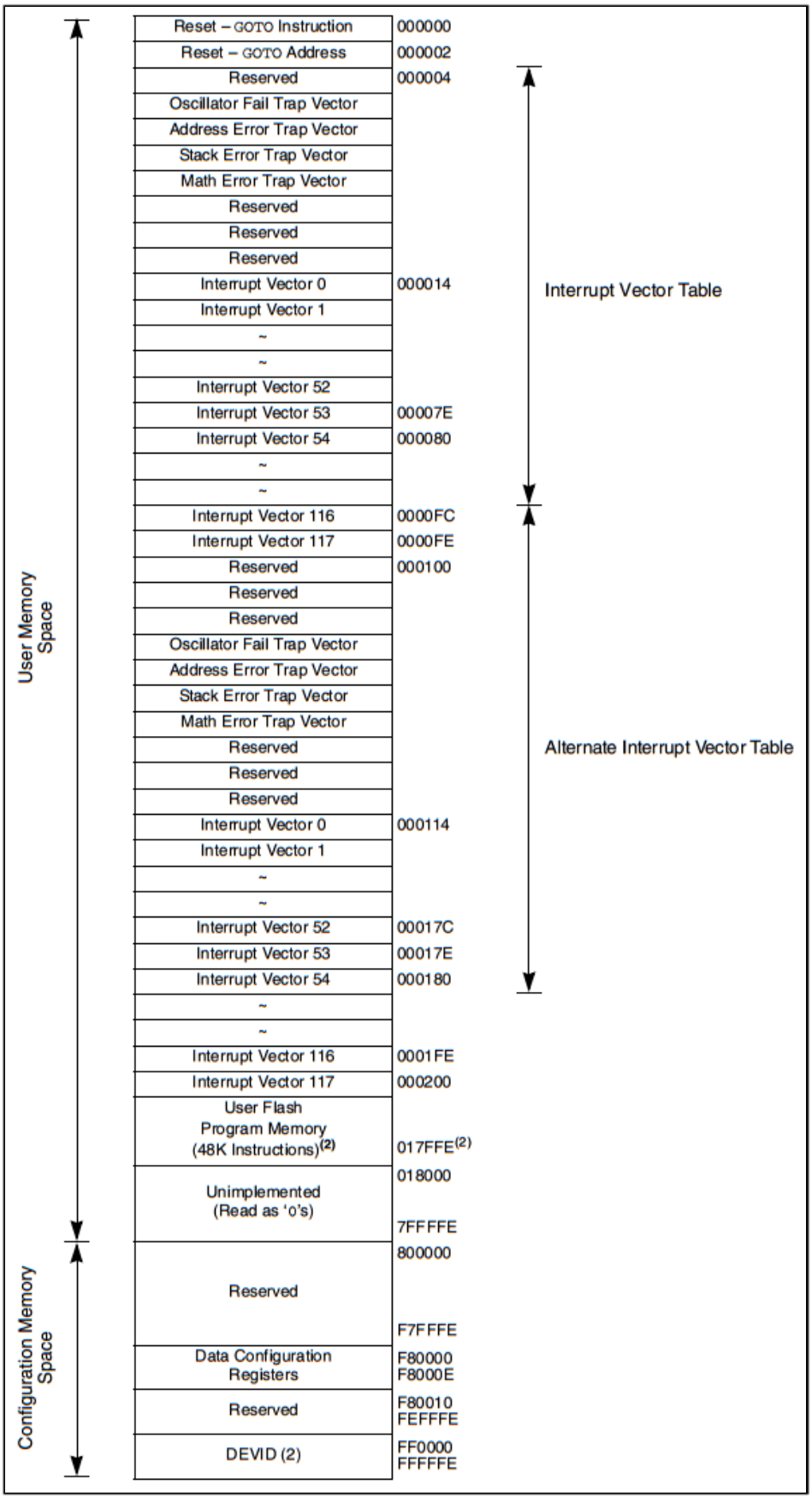


Figura 30. Mapeamento da Memória de Programa, família PIC24F. Figura extraída de [54].

O registrador de 16 bits SPLIM (*Stack Pointer Limit Value register*) está associado à pilha (*Stack Pointer*) definindo um limite máximo para o endereço dessa pilha [54]. O conteúdo do registrador SPLIM é sempre comparado com o registrador W15 (ponteiro da pilha) [56, pp. 40-42]. Se os conteúdos são iguais uma operação de desvio é realizada, caso contrário acontece uma *Trap* [56, pp. 40-42]. A *Trap* é uma forma especial de interrupção para o tratamento de erros, iniciada em caso de ocorrência de uma exceção, e são usadas para tratar condições especiais de erros no microcontrolador, por exemplo, uma falha no oscilador, endereçamento incorreto (acesso a endereço ímpar), *underflow* da pilha ou divisão por zero [60, p.57]. Quando uma *trap* acontece, uma rotina de serviço *trap* (do inglês, *Trap Service Routine*, TSR) é executada. A TSR funciona para as *traps*, assim como a ISR para interrupções no microcontrolador, necessitando zerar a *flag* da *trap* ao final do tratamento do erro, evitando assim a reentrada na TSR [54]. As *traps* são consideradas interrupções que não podem ser mascaradas, por intermédio dos *bits* de habilitação de interrupções (registradores IECx) [61, cap. 3], e, além do mais, elas não obedecem ao sistema de priorização do MCU, pois têm atendimento prioritário [56, pp. 40-42]. A Tabela 16 mostra os resultados obtidos dos testes de avaliação com o registrador SPLIM:

Tabela 16. Resultados dos testes de avaliação com o registrador SPLIM.

Upset na Stack*	**Total de *upsets* injetados****	**Total de Erros****	**Valores absolutos de erros****			**Porcentagem de erros****		
			TE	**RE**	**SL**	**TE**	**RE**	**SL**
SPLIM	7466	1078	0	1078	0	0	100%	0

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).
** Valores médios calculados por três observações experimentais.
Legenda: TE = *Tolerated Errors*, RE = *Result Errors* e SL = *Sequence Loss*.

Os resultados dos testes de avaliação apresentados na Tabela 16 mostram que a mudança de valores do registrador SPLIM causa o envio de atitudes erradas, gerando cem por cento de *Result Errors*. Isso porque, a ação de zerar o conteúdo do registrador SPLIM provoca a sinalização da *Trap* de erro da pilha, portanto o microcontrolador interrompe todas as tarefas que estão sendo executas para atender essa sinalização [56, pp. 40-42], atrasando o cálculo da atitude. Logo, no momento em que acontece a interrupção do temporizador, a atitude não está completamente calculada devido aos desvios provocados pelo tratamento da TSR.

Os registradores CORCON (CPU *Control register*), CRCCON (*Cyclic Redundancy Check Control register*), OSCCON (*Oscillator Control register*), RCON (*Reset Control register*), por motivos de segurança, não possuem acesso direto, em tempo de execução, através do conjunto de instruções do microcontrolador, portanto não puderam ser

reconfigurados para emulação de SEUs. Esse fato não influência no mapeamento da área sensível, pois era esperado que nem todos os possíveis alvos permitem acesso através do conjunto de instruções do microcontrolador. Como eles representam uma pequena porção da de toda a memória acessível não trazem consequências para o estudo realizado nesse trabalho.

Os registradores SR (*Status register*), DISICNT (*Disable Interrupts Counter register*), PORTA, PORTB, LATA, LATB, TRISA e TRISB foram submetidos aos testes de avaliação e apresentaram cem por cento de erros toleráveis (*Tolerated Errors*).

O programa de determinação de atitude ocupa 806 bytes da memória de dados do PIC24F (incluindo os SFRs usados). Desses 806 bytes (6448 bits), somente 359 bits produziram algum erro observável na saída do sistema, sendo 307 bits dos SFRs e 52 bits das variáveis globais e locais. Isso representa um percentual de 5,57% de área sensível.

As variáveis que ocupam a memória de dados, em sua maioria, são locais e seu conteúdo é atualizado constantemente durante o cálculo da atitude, minimizando, portanto os efeitos de *upsets* injetados nelas.

Devido ao comportamento encontrado no *bit* U1RXIE<11> do registrador IEC0, testes de avaliação complementares foram realizados com os outros 176 registradores de função especial do PIC24F, porém nenhum outro registrador apresentou comportamento incomum, originando resultados de cem por cento de erros toleráveis.

5.2.2. Resultados da Aplicação do Sistema PORTHES no microcontrolador COTS gravado com o sistema de determinação de atitude

Concluído o mapeamento da área sensível a SEUs e identificados os efeitos causados na saída do sistema para cada um dos alvos pertencentes à região de sensibilidade, realizou-se a segunda sessão de testes. A segunda sessão de testes tem por objetivo reproduzir o experimento de Velazco et al. em [43] para atestar se a validade da afirmação de que a *cross-section* experimental do DUT em funcionamento com uma certa aplicação, esta relacionada diretamente com a porção de memória ocupada pela aplicação gravada no DUT em relação ao total de memória disponível no mesmo e com a *cross-section* estática do DUT sob um processo de irradiação contendo toda sua memória preenchida com um determinado padrão binário conhecido. A partir da determinação da *cross-section* experimental do DUT em funcionamento com o programa obtém-se a SER. Velazco cita em [43] que realizou um

experimento com o microcontrolador 80C51, executando um *firmware* de multiplicação de matrizes 6x6. Esse experimento foi operado completamente pelo sistema THESIC, que segundo o autor, automatiza todo o processo de levantamento da *cross-section* predita e experimental do MCU 8051 em funcionamento com a aplicação de multiplicação de matrizes 6x6 e consequentemente o cálculo da SER do mesmo. A Tabela 17 mostra os resultados encontrados por Velazco et al. no experimento apresentado em [43] em valores absolutos e percentuais de falhas injetadas, total de erros observados segundo a mesma classificação de erros usada nesse trabalho.

Tabela 17. Resultados apresentados por Velazco et al. em [43] com o microcontrolador 80C51.

	Falhas injetadas*	**Total de erros***	**Erros toleráveis ou ausência de erros***	***Result Errors** *	**Erros por perda de sequência***
Valores absolutos	12245	12245	6117	5784	344
Percentuais	-	100%	49,96%	47,24%	2,8%

* Dados repostados por Velazco et al. no artigo "*Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection*" [43].

A determinação da *cross-section* experimental do DUT em funcionamento com o programa de cálculo da atitude no PIC24F e o cálculo da SER, usando a reprodução da metodologia baseada em CEU de Velazco et al, é o objetivo dessa parte do trabalho. O DUT PIC24F possui características próprias de arquitetura, projeto lógico e processo de fabricação do circuito integrado diferentes do DUT 80C51 usado por Velazco. A quantidade e o tipo de memória disponível em cada DUT, assim como o tipo da aplicação, os pontos de observabilidade das saídas do DUT, a forma de avaliar o seu correto funcionamento na ausência de falhas e a quantidade de memória de programa e memória de dados que cada aplicação ocupa é significantemente diferente de um experimento de teste para o outro. Apesar dessas diferenças e características particulares de cada conjunto DUT-aplicação, ainda assim certas comparações de resultados desses experimentos podem ser realizadas para efeito de elucidar diferenças entre os procedimentos de emulação de SEU com o PORTHES e com o THESIC. E também a validação se o PORTHES emula SEU de maneira similar ao THESIC desenvolvido por Velazco.

O *firmware* gravado no PIC24F usado nesses experimentos calcula a atitude e a envia na forma de quatro números no formato da notação IEEE-754 ponto flutuante de precisão simples ao PORTHES via comunicação serial assíncrona a cada 300ms. O sistema PORTHES por sua vez é configurado para injetar em média 2,3 falhas por segundo obedecendo à

distribuição de Poisson, configuração esta usada por Velazco no experimento com o 80C51 executando o cálculo de multiplicação de matrizes 6 x 6. Portanto o sistema PORTHES só poderá avaliar o efeito de um ou mais SEUs que possam eventualmente ocorrer dentro do *firmware* de cálculo da atitude a cada 300ms, pois é o único ponto externo observável ao DUT para avaliar se houve ou não um erro a partir do SEU emulado. E evidentemente que os pontos de observabilidade, assim como, a frequência de observabilidade são particularidades inerentes à aplicação gravada no DUT.

Para preparo dessa etapa do experimento, os alvos no CEU *code* foram organizados dentro da estrutura condicional "*switch case*" na rotina de tratamento de interrupção do *firmware* de determinação de atitude gravado no DUT. Conforme os percentuais arredondados apresentados por Velazco et al. reproduzidos para esse trabalho na Tabela 17: 50% dos casos na estrutura do "*switch case*" no CEU *code* foram alvos que produzissem erros toleráveis ou ausência de erros (*Tolerated Errors*); 47% os alvos que provocassem *Result Errors* e 3% a alvos que gerassem erros de perda de sequência ou travamento do sistema (*Sequence Loss*). Os valores arredondados foram necessários para evitar que a estrutura "*switch case*" ficasse muito extensa e interferisse significativamente no tempo de execução da rotina de tratamento de interrupção, assim como na quantidade total de memória de programa do sistema em teste. Os alvos foram distribuídos sem ordenação específica dentro dessa estrutura de controle de fluxo, pois um sorteio aleatório, configurado na própria rotina de tratamento da interrupção programada dentro do dispositivo, garantiu a escolha aleatória do alvo a ser atingido. O sorteio aleatório era realizado da seguinte maneira: o valor atual do registrador TMR1 era lido e esse valor lido era usado para acessar um vetor de números aleatórios com 100, 200 ou 300 posições (dependendo do tamanho do CEU *code*), sempre que a rotina de tratamento de interrupção era iniciada. Os alvos do CEU *code* foram selecionados com elementos de memória do DUT testados na etapa I do método proposto nesse trabalho (apresentada na forma de tabelas na seção 4), que produziram cem por cento um tipo de erro perceptível ao sistema. Mantendo a proporção de 50% de *Tolerated Errors*, 47% de *Result Errors* e 3 % de *Sequence Loss* foram preparados três conjuntos de CEU *codes*. O primeiro com cem alvos, o segundo com duzentos alvos e o terceiro com trezentos alvos. A razão da preparação de três experimentos diferentes de CEU *code* com variação nas quantidades absolutas de CEU *targets* se justifica para analisarmos a influência do crescimento em números absolutos de CEU *targets* dentro de CEU *code* no tempo necessário para processar corretamente um erro e na qualidade numérica do resultado gerado.

Posteriormente à organização do CEU *code* foi possível submeter o DUT PIC24F com o sistema de determinação de atitude à reprodução da metodologia CEU de Velazco et al. em [41]. Os resultados encontrados submetendo o DUT às sessões de testes com CEU *codes* contendo cem, duzentos e trezentos CEU *targets* estão apresentados na Tabela 18:

Tabela 18. Resultados dos experimentos com o sistema PORTHES para o MCU PIC24F.

	Número de casos no "*switch case*" do CEU *code***	***Bit-flips* injetados**	**Total**	***Tolerated errors***	***Result errors***	***Sequence Loss***
Valores absolutos*	100	11864	19149	9564	8282	1303
Percentuais*		-	100%	49,95%	43,25%	6,80%
Valores absolutos*	200	11714	18464	9228	8026	1210
Percentuais*		-	100%	49,98%	43,47%	6,55%
Valores absolutos*	300	12020	18734	9600	1150	7984
Percentuais*		-	100%	51,24%	42,62%	6,14%

* Valores médios calculados por duas observações experimentais.
** Cada bateria de sessão experimental teve a mesma duração de tempo (aproximadamente 1,5h).

Tabela 19. Tempo de processamento da ISR para diversos tamanhos de CEU *code*.

	Número de casos no "*switch case*" do CEU *code*	**Tempo de execução**
Tratamento da ISR:	100, 200 e 300	25,50us
Fluxo do programa sem executar ISR:		10,4905ms
Fluxo do programa executando ISR:		10,5155ms

A primeira constatação que se pode tirar do experimento de variação do número de CEU *targets* dentro do CEU *code* é que o aumento de CEU *targets* dentro de um CEU *code* pouco contribui na influência da precisão dos resultados e o tempo de processamento da ISR não foi percentualmente significante, a ponto de interferir no processamento normal da aplicação (Tabela 19). A segunda constatação é que o experimento realizado com o PIC24F – *firmware* de determinação de atitude, embora tenha sido realizado com a mesma taxa de inserção de falhas usada por Velazco em outro conjunto DUT–aplicação não apresenta a mesma proporção de *Result Errors* e *Sequence Loss* como resultado apresentado no trabalho de Velazco (Tabela 17). Apesar da aparente evidência dessa constatação ela é importante para justificar o porquê dessa observação e a nulidade da comparação desses experimentos. O fato de serem aplicações diferentes com características e frequências de observabilidade próprias, por si só, já justificaria a grande diferença de um resultado para outro. Um segundo fato, com menor influência sobre essa diferença percebida, é que a própria natureza estocástica do problema produz experimentos com valores percentuais distintos. Evidentemente que pelas razões expostas não há como comparar os resultados da Tabela 17 com a Tabela 18. Entretanto o que se deseja e se faz sentido comparar é se a *cross-section* predita, calculada a

partir da *cross-section* estática e da porção de memória ocupada pelo sistema de determinação de atitude no PIC24F é significantemente próxima a *cross-section* experimental obtida pelo processo de emulação no PORTHES, de acordo com a afirmação apresentada do experimento de Velazco no referido trabalho citado na Tabela 17.

Uma observação importante a se destacar nos resultados da Tabela 18 é que o número de falhas injetadas nesse experimento é em média 2,3 falhas por segundo obedecendo à distribuição de Poisson, mas a taxa de observação das saídas do mesmo experimento, característica da aplicação do sistema de determinação de atitude, é em média 3,3 vezes por segundo, daí justificando-se a quantidade de erros observados (coluna Total da Tabela 18) ser maior que o número de SEUs injetados (coluna *Bit-flips* injetados da Tabela 18). De certa forma o leitor pode interpretar essa situação como uma controvérsia. Isto é, como certo número de falhas injetado em um sistema poderia produzir um número superior de erros observáveis no mesmo sistema, já que o erro é definido como um sintoma observado a partir de uma falha ocorrida? Isso na verdade não ocorre, porque a coluna Total inclui o tipo *Tolerated Errors* da mesma forma que Velazco. E *Tolerated Errors*, na verdade são valores de saídas do PIC24F observadas pelo PORTHES que coincidem completamente com os valores esperados programado na IG do PORTHES. Portanto somente *Result Errors* e *Sequence Loss* deveriam ser contabilizados para a totalização do número de erros. A soma dos *Result errors* com o número de *Sequence Loss* é que utilizada para o cálculo da obtenção da *cross-section* experimental da aplicação executando no PIC24F.

A diferença na organização da arquitetura do 80C51 e do PIC24F também pode influenciar no percentual de casos de elementos de memória sensíveis a SEU, se o percentual de elementos de memória do 80C51 e do PIC24F acessível pelo conjunto de instruções de cada arquitetura for muito maior em um do que outro e ainda se essa quantidade for significativa em relação ao total de elementos de memória de cada dispositivo em questão. Mas o fator que certamente afeta de forma mais significativa e que poderia comprometer a validade do experimento apresentado na Tabela 18 e, sobretudo sua comparação com os resultados de Velazco é a diferença entre as tecnologias usadas na fabricação dos transistores que compõem os circuitos combinacionais e sequenciais desses DUTs, que datam aproximadamente 10 anos, uma da outra. Essa diferença afeta para mais a medida de SER que é também dependente da *cross-section* estática do dispositivo submetido a um processo de radiação ionizante. Não se pode afirmar de quanto seria essa diferença, pois teria que

submeter o PIC24F executando o sistema de determinação de atitude ao mesmo processo de radiação ionizante com o mesmo isótopo radioativo e o mesmo fluxo de radiação a fim de se obter a *cross-section* estática real para o PIC24F. Entretanto, o PORTHES permite simular situações extremas que representem emulações de SEUs com uma frequência de injeção com valores de até uma ordem de grandeza superior ao simulado no presente experimento para efeito de validação da qualidade de emulação de SEU.

A essência do resultado da metodologia baseada em CEU desenvolvida por Velazco é a possibilidade de se emular a SER de um dispositivo microprocessado COTS gravado com um programa, como se esse estivesse sendo submetido a um processo de radiação ionizante, sem a necessidade de submeter o DUT várias vezes a um mesmo processo de exposição à radiação ionizante. Velazco et al. apresentam diversos resultados, por exemplo, em [40] e [44], que justificam o uso da metodologia baseada em CEU para predizer e calcular experimentalmente a *cross-section* de diversos conjuntos de dispositivos-aplicações, evitando expô-los mais de uma vez ao fluxo de partículas radioativas. Evidentemente que essa metodologia se apoia na viabilidade de conhecimento e acesso ao código gravado no microcontrolador DUT, assim como na possibilidade de intervir no processo de execução da aplicação durante a emulação através de uma chamada de interrupção a um evento externo, quando o DUT em questão for um dispositivo microprocessado.

A determinação da *cross-section* predita do DUT programado com o *firmware* de determinação de atitude é calculada pela equação (8). Através do percentual dos *bits*, que foi calculado na equação (7), usados pelo *firmware* de determinação de atitude em relação ao total de bits dos elementos de memória do PIC24F e do valor da *cross-section* estática usada como base para configuração da taxa de injeção de falhas no DUT pelo PORTHES ($\sigma_{SEU} = 1{,}73 * 10^{-2}\ cm^{-2}$) (mesma usada por Faure et al. em [44], no experimento radioativo com o isótopo radioativo do cloro), é possível realizar o cálculo da *cross-section* teórica predita da aplicação de determinação da atitude no DUT em teste.

$$\begin{aligned} \sigma_{SEU(Predita)} &= Percentual * \sigma_{SEU} \\ \sigma_{SEU(Predita)} &= 0{,}098 * 1{,}73 * 10^{-2}\ cm^{-2} \\ \sigma_{SEU(Predita)} &= 0{,}169 * 10^{-2}\ cm^{-2} \\ \sigma_{SEU(Predita)} &= 1{,}69 * 10^{-3}\ cm^{-2} \end{aligned} \tag{8}$$

Em seguida, usando a equação (4) e os valores do experimento com 300 CEU *targets* dentro do CEU *code*, apresentados na Tabela 18, calcula-se por meio da equação (9) o valor percentual de erros observados nas saídas do programa gravado no DUT PIC24F após o experimento realizado no PORTHES:

$$\tau_{CEU} = \frac{\#erros}{\#CEUs\ injetados} = \frac{9134}{12020} = 0{,}760 \tag{9}$$

Para a aplicação de determinação de atitude gravada no MCU PIC24F, o valor do percentual de erros observados calculado a partir da equação (9) foi $\tau_{CEU} = 0{,}760$. Com o valor de τ_{CEU} computado, calcula-se através da equação (5) a *cross-section* experimental do programa particular em execução no dispositivo microprocessado COTS. A equação (10) mostra o cálculo da *cross-section* experimental do programa, segundo Velazco.

$$\begin{aligned} \sigma_{SEU\ (semi-experimental)} &= \sigma_{SEU(predita)} * \tau_{CEU} \\ \sigma_{SEU\ (semi-experimental)} &= 1{,}69 * 10^{-3} * 0{,}760 \\ \sigma_{SEU\ (semi-experimental)} &= 1{,}28 * 10^{-3} cm^{-2} \end{aligned} \tag{10}$$

Nesse caso, a $\sigma_{SEU\ (semi-experimental)}$ é igual a $1{,}28 * 10^{-3} cm^{-2}$, mostrando que o sistema PORTHES emula SEUs de forma significativamente próxima do sistema THESIC de Velazco, comprovando assim as observações e resultados de Velazco obtido com o THESIC. A Tabela 20 mostra as *cross-sections* predita e semi-experimental do PORTHES e do THESIC.

Tabela 20. Comparação entre as cross-sections Predita e Calculada.

	Cross-section Predita (cm^{-2})	*Cross-section* Semi-experimental (cm^{-2})
PORTHES	$1{,}69 * 10^{-3}$	$1{,}28 * 10^{-3}$
THESIC*	$1{,}66 * 10^{-3}$	$1{,}55 * 10^{-3}$

* Dados repostados por Velazco et al. no artigo "*Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection*" [43].

Calculada a *cross-section* semi-experimental do *firmware* de determinação de atitude, procede-se, por fim, o calculo da *Soft Error Rate* (SER) esperada da aplicação gravada no dispositivo em teste. Através da equação (6) e o valor do fluxo de partículas ($\Phi \cong 1{,}10\ 10^4\ p.cm^2.s^{-1}$) de partículas radioativas usado para o levantamento da *cross-section* no trabalho apresentado por Velazco et al. em [44], é possível se computar a *Soft Error Rate* (SER) esperada da aplicação gravada no dispositivo em testes. Assim, a *Soft Error Rate*

calculada para o *firmware* de determinação de atitude resulta em 14,08 erros observáveis/segundo.

O cálculo da *cross-section* predita de forma teórica e a *cross-section* obtida via procedimento experimental no PORTHES com o conjunto PIC24F executando o cálculo da atitude foram obtidos utilizando a *cross-section* estática mensurada por Velazco et al. com um dispositivo fabricado com uma tecnologia de fabricação de circuitos integrados CMOS cerca de uma década anterior à tecnologia usada na fabricação do PIC24F. Apesar de haver essa diferença, foi observado em artigos publicados por Velazco et al., que os valores obtidos do levantamento da *static cross-section* de outros dispositivos microprocessados fabricados em diferentes anos a mais e a menos do referido trabalho, quando submetidos aos mais diferentes tipos de isótopos radioativos e sob os mais diversos fluxos de partículas seguiam valores absolutos de mesma ordem de grandeza da *static cross-section* que foi usada para calcular os parâmetros de injeção do emulador e por consequência validar o sistema de emulação a SEU no PORTHES. Mesmo ainda, se essa afirmação estiver incorrendo em um falso pressuposto, suponha que ao expor o DUT utilizado nesse trabalho a um fluxo de partículas radioativas e a *static cross-section* obtida tivesse uma ordem de grandeza acima dos valores apresentados por Velazco, está comprovado que ainda assim por simulação realizada com valores fictícios maiores que 2,3 em uma ordem de grandeza, que a ferramenta desenvolvida PORTHES conseguiria emular o comportamento de SEUs no sistema de determinação de atitude no PIC24F.

5.2.2.1. Teste de emulação de falhas do sistema PORTHES

Um último teste se faz necessário para atestar a qualidade de emulação de falhas do sistema PORTHES. O sistema PORTHES deve simular a taxa de injeção de falhas com certa frequência, no DUT com uma aplicação em funcionamento, obedecendo a uma distribuição de Poisson com certo λ (representativo da taxa de ocorrência de falhas do tipo SEU por unidade de tempo). A verossimilhança da distribuição estatística mais adequada deve foi comprovada através de experimento e testes estatísticos apropriados realizados sob os seus resultados obtidos a partir do experimento. A fim de verificar a qualidade do gerador de números aleatórios, o PORTHES foi programado com uma taxa de injeção de falhas média com $\lambda = 2,3$. A ocorrência de eventos (falhas) foi medida na saída do injetor do PORTHES durante um tempo total de experimento de 2 minutos (120 segundos). A Figura 31 comprova

que as taxa de SEUs emulados durante um experimento de dois minutos, obedecem à distribuição de Poisson.

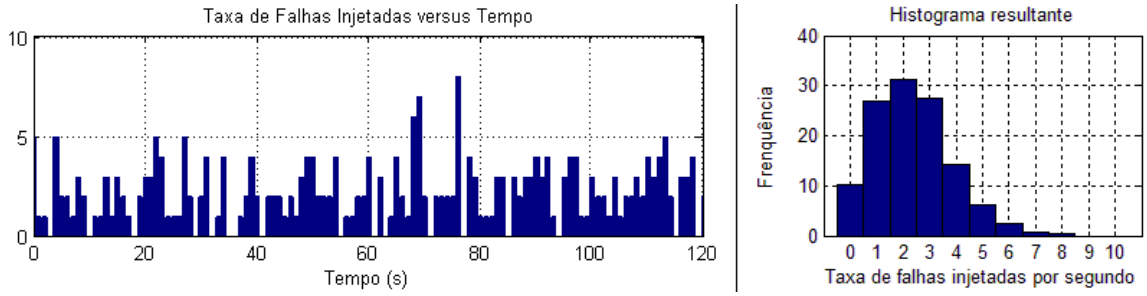


Figura 31. Resultado das sessões de testes de injeção de falhas com a metodologia proposta no trabalho.

Uma análise mais detalhada na base de dados obtida durante esse experimento, usando a ferramenta computacional estatística MATLAB [63], permite extrair os parâmetros da distribuição de probabilidade encontrados no padrão dos dados podem ser calculados, esse padrão é apresentado na Figura 31. Nessa análise foi utilizada a função *poissfit*, que permite estimar parâmetros da distribuição de Poisson e computar o número médio de ocorrências das falhas emuladas, ou seja, o parâmetro *lambda* dessa distribuição de probabilidade com um nível de confiabilidade de 95% [63]. Na função *poissfit* a média do conjunto de dados (amostra) é a estimativa de máxima verossimilhança do parâmetro *lambda* [63]. Portanto o valor da taxa média de *upsets* emulados com o sistema PORTHES calculada com a função *poissfit* é igual a 2,3196 ± 0,1160.

Essa última análise mostra que o sistema PORTHES possui um controle na configuração dos parâmetros de injeção das falhas no DUT representando a distribuição de Poisson com um nível de confiabilidade superior a 95%, comprovando que a ferramenta desenvolvida atinge uma qualidade na emulação de SEUs sob esse aspecto.

5.2.3. Resultados dos testes de validação do Sistema de Determinação de Atitude com Tolerância a Falhas (SDATF) a SEUs no PORTHES.

A última sessão de experimentos com o PORTHES consistiu na injeção de *upsets* no Sistema de Determinação de Atitude com Tolerância a Falhas (SDATF) desenvolvido para operar em satélites artificiais de baixa órbita, teve como objetivo atestar a reação do seu comportamento diante da emulação dos SEUs descrito no capítulo 3.

O SDATF tem uma característica particular de enviar para a CPU principal, junto com o *quaternion*, um número de identificação (ID). O ID serve para identificar o microcontrolador, que está na função de *MASTER*, responsável por transmitir os dados da atitude para fora do sistema. Essa identificação é necessária para computar a disponibilidade do sistema e a quantidade de trocas que ocorreram na função de *MASTER* do sistema de determinação de atitude com tolerância a falhas. Os microcontroladores do SDATF são configurados com três IDs representados por um número inteiro sem sinal: um, dois e três. Esses IDs são referentes aos microcontroladores MCU1, MCU2 e MCU3, respectivamente. No caso do sistema travar gerando erros de perda de sequência, o ID zero é computado para indicar que nenhum microcontrolador está respondendo. Além disso, é possível que uma falha no sistema envie valores de IDs com um ou mais *bits* alterados.

A disponibilidade do sistema é calculada pela divisão dos IDs corretos recebidos pelo número total de IDs esperados durante o tempo do experimento, mostrado na equação (11):

$$DS = \frac{número\ de\ IDs\ corretos}{número\ de\ IDS\ esperado} \tag{11}$$

Para essa última sessão de experimentos com o PORTHES, três cenários experimentais foram elaborados para testar o SDATF, descritos a seguir:

5.2.3.1. Primeiro cenário experimental com o SDATF:

O primeiro cenário foi configurado da seguinte maneira: na primeira versão do *firmware* de determinação de atitude com tolerância a falhas gravado nos três microcontroladores (*MASTER*, *SAMPLER* e *SLEEPER*) foi inserido um trecho de código com um determinado alvo na rotina de interrupção de eventos externos. Os alvos forma escolhidos entres os 52 registradores usados pela primeira versão do programa de determinação de atitude com tolerância a falhas, são eles: T1CON, TMR1, PR1, T2CON, TMR2, PR2, T3CON, TMR3, PR3, IFS0, IFS1, IEC0, IEC1, IPC0, IPC1, IPC2, U2MODE, U2STA, U2BRG, U2TXREG, U2RXREG, INT0, INT1, INT2, CLKDIV, WREG0, WREG1, WREG2, WREG3, WREG4, WREG5, WREG6, WREG7, WREG8, WREG9, WREG10, WREG11, WREG12, WREG13, WREG14, WREG15, RPOR4, RPINR0, RPINR1, RPINR19, TRISA, TRISB, PORTA, PORTB, LATA, LATB, SPLIM e PC. A cada inicio da bateria de testes um registrador era configurado com alvo na rotina de tratamento de

interrupção e submetido aos testes de injeção de *bit-flips*. Nesse primeiro cenário, o sistema PORTHES foi configurado no modo Manual, conforme descrito no capítulo 4, permitindo o usuário escolher o microcontrolador e o momento de injeção do *bit-flip*.

Mesmo que esse primeiro cenário não estivesse simulando a taxa de chegada de partículas radioativas ionizantes, ele teve como objetivo detectar algum problema ou erro não previsto, que podia ter passado despercebido na programação do SDATF.

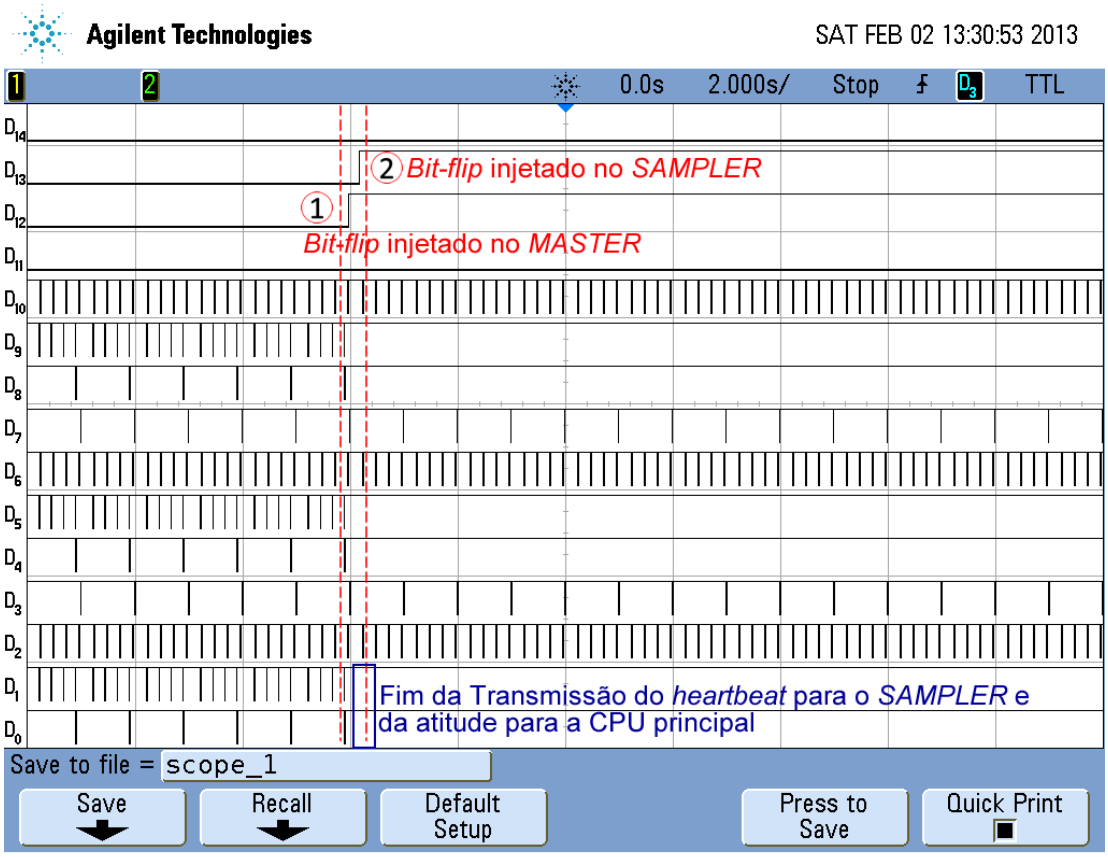
Para essa primeira bateria de testes realizados no primeiro cenário, a maioria dos registradores do SDATF não apresentou falhas críticas de funcionamento do sistema, mantendo a execução da determinação e envio da atitude para fora do sistema, como esperado. Com exceção dos registradores T1CON, T2CON e T3CON, que apresentaram travamento do sistema, quando *bit-flips* eram inseridos nos *bits* desses registradores, parando o envio da atitude para a CPU principal em situações particulares.

O primeiro registrador a apresentar um problema crítico foi o T1CON, que é responsável pela configuração e controle do temporizador 1 [54]. Ele possui funções distintas em cada modo do SDATF: na função de *MASTER*, o temporizador 1 é responsável pelo envio periódico do *heartbeat* para o *SAMPLER* e incrementar o contador de *heartbeats* enviados. Na função de *SAMPLER*, o temporizador 1 é responsável por esperar um tempo limite para a chegada do *heartbeat* enviado pelo *MASTER*, no caso de extrapolar o tempo limite o *SAMPLER* tornar-se *MASTER*. Se os *bits* do registrador T1CON do *MASTER* sofrerem uma mudança de estado, provocada por um *bit-flip* injetado, a configuração do envio do *heartbeat* do *MASTER* para o *SAMPLER* é alterada. O *SAMPLER* detecta esse comportamento impróprio, reiniciando o MCU com falha e se tornando o novo *MASTER*. Todavia, se dois *bit-flips* sequenciais atingirem primeiro o registrador T1CON do *MASTER* e depois o registrador T1CON do *SAMPLER* (Figura 32), e, ainda, forem rápidos o suficiente para desabilitar os temporizadores antes de disparar a interrupção, os microcontroladores continuam a comunicação, mantendo as funções originais, como se nenhum problema tivesse ocorrido. O motivo desse comportamento inadequado é o fato do temporizador 1 do *SAMPLER* ter sido desativado, assim o *SAMPLER* não percebe que o *MASTER* deixou de enviar *heartbeats*. No lado do *MASTER*, a atitude para a CPU principal é enviada apenas após quatro *heartbeats* serem transmitidos para o *SAMPLER* (Figura 25) e condicionada pelo contador de *heartbeats*. Portanto o *MASTER* cessa a transmissão da atitude para a CPU principal, porque o temporizador 1 não incrementa mais o contador de *heartbeats* enviados, mantendo a condição

sempre falsa. Esse mesmo comportamento também foi observado quando os *bit-flips* sequenciais eram inseridos na ordem inversa, o primeiro *bit-flip* é inserido no registrador T1CON do *SAMPLER*, e depois no T1CON do *MASTER* (Figura 33).

A Figura 32, obtida com o auxílio de um Analisador Lógico, apresenta a situação de dois *bit-flips* sequenciais no registrador T1CON, o primeiro ocorrendo no MCU *MASTER* e o segundo no MCU *SAMPLER*. Esse segundo *bit-flip* deve ser rápido o suficiente para acontecer antes do temporizador 1 do *SAMPLER*, impedindo a sinalização da interrupção, caso contrário o MCU *SAMPLER* detecta o funcionamento incorreto do MCU *MASTER* e assume o controle da situação. O número (1) e o número (2), representados na Figura 32, indicam o momento, no qual os *bit-flips* são injetados, primeiro no *MASTER* e depois no *SAMPLER*. É possível observar que instantes após o *bit-flip* inserido no *MASTER*, ele encerra o envio do *heartbeat* para o *SAMPLER* e também da atitude para a CPU principal. O segundo *bit-flip* inserido no *SAMPLER* foi rápido o suficiente para desabilitar o temporizador antes de disparar a interrupção. Sendo assim, os microcontroladores continuaram desempenhar as suas respectivas funções, e o sistema apresentou travamento por falta de envio da atitude para a CPU principal a cada segundo.

A Figura 33, obtida em um Analisador Lógico, apresenta a ocorrência de dois *bit-flips* sequenciais no registrador T1CON, o primeiro ocorrendo no MCU *SAMPLER* e o segundo no MCU *MASTER*. O primeiro *bit-flip*, número (1) na Figura 33, desabilitou o temporizador do *SAMPLER* programado para verificar o envio do *heartbeat* pelo *MASTER*. O segundo *bit-flip*, número (2) na Figura 33, desabilitou o envio do *heartbeat* do *MASTER* para o *SAMPLER*. A comunicação continuou normalmente, pois o *SAMPLER* não percebeu o mau funcionamento no *MASTER*. Portanto, os microcontroladores continuaram a desempenhar as suas respectivas funções, mesmo com o sistema cessando o envio da atitude para a CPU principal.




```
Legenda:
D0  - UART1 PIC1         | D8  - UART1 PIC3
D1  - HeartBeat PIC1     | D9  - HeartBeat PIC3
D2  - Quaternion PIC1    | D10 - Quaternion PIC3
D3  - Quadro de 1s PIC1  | D11 - Quadro de 1s PIC3
D4  - UART1 PIC2         | D12 - Bit-flip no PIC1
D5  - HeartBeat PIC2     | D13 - Bit-flip no PIC2
D6  - Quaternion PIC2    | D14 - Bit-flip no PIC3
D7  - Quadro de 1s PIC2  |
```

Figura 32. Cenário de *bit-flip* no registrador T1CON, primeiro no *MASTER* e depois no *SAMPLER*.

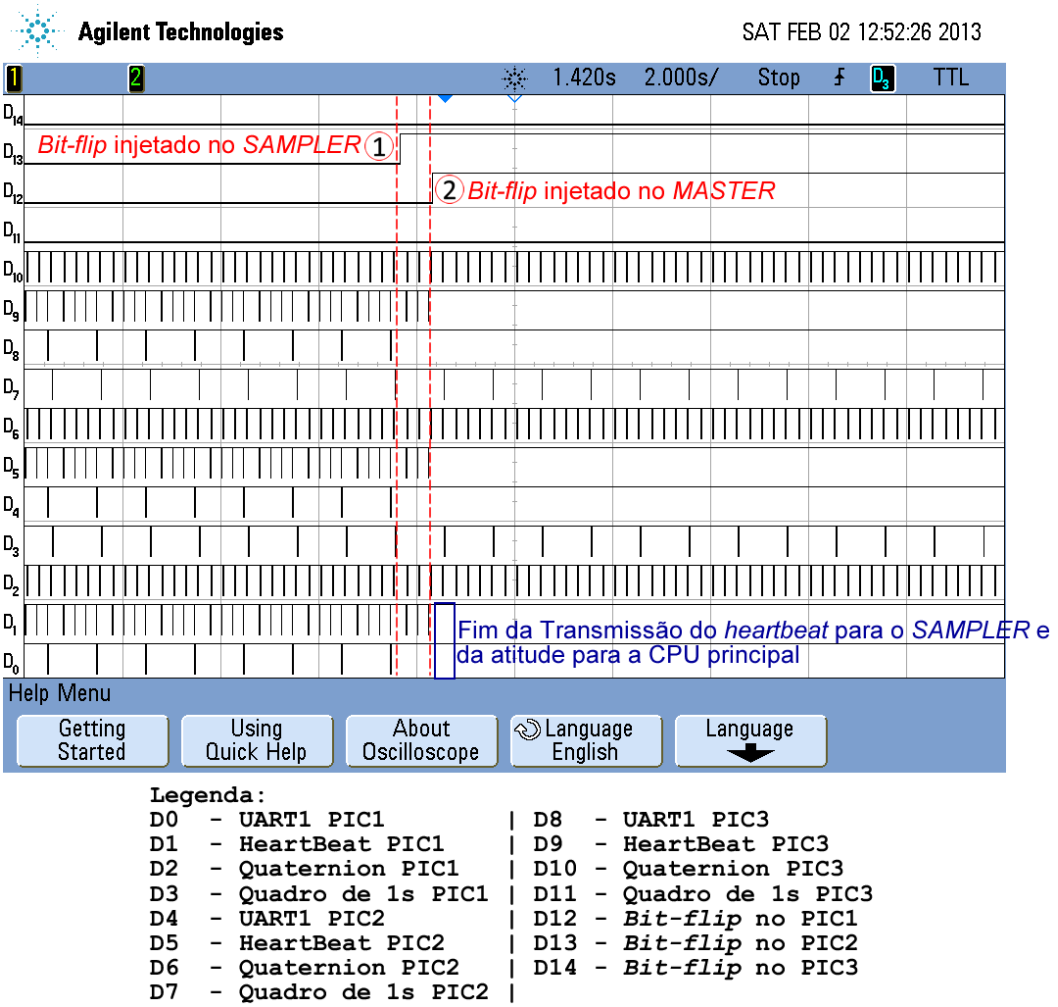
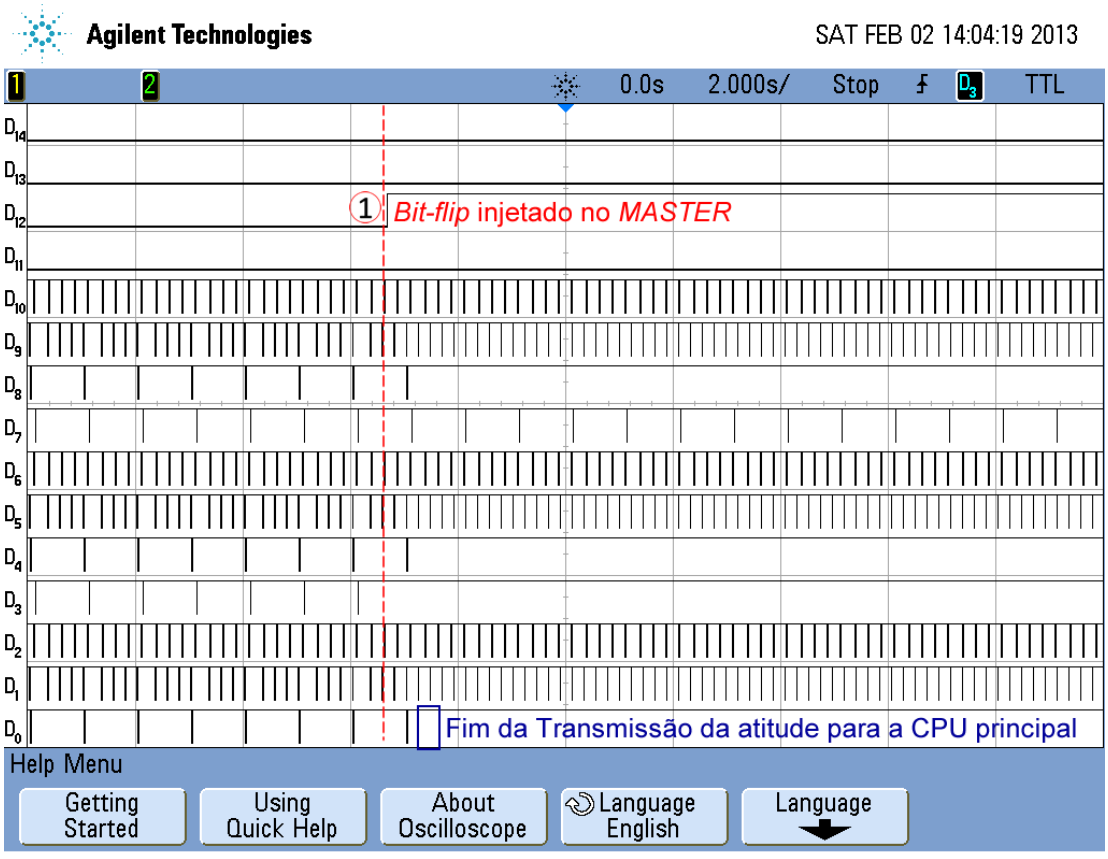


Figura 33. Cenário de *bit-flip* no registrador T1CON, primeiro no *SAMPLER* e depois no *MASTER*.

Esse comportamento não previsto do SDATF foi solucionado inserindo a reconfiguração do temporizador 1 na rotina de sincronismo dos microcontroladores. Essa rotina é controlada pelo temporizador 3 do MCU *MASTER* e do MCU *SAMPLER* e é responsável pelo sincronismo do quadro de temporização do funcionamento normal do SDATF (Figura 25). A ideia para essa solução foi inspirada em um *Application Note* do fabricante MICROCHIP, que descreve uma biblioteca de rotinas para detectar a ocorrência de falhas em um microcontrolador, realizando a reconfiguração do MCU se necessária [64].

O segundo registrador T2CON é responsável pela configuração e controle do temporizador 2 [54]. Ele possui funções distintas em cada modo do SDATF: no modo *SAMPLER*, o temporizador 2 é responsável pelo envio periódico do *quaternion* calculado para o *MASTER* e incrementar o contador de *quaternions* enviados (no SDATF a atitude calculada é representada em forma de *quaternion* [51]). No modo *MASTER*, o temporizador 2 é

responsável por monitorar o tempo limite para o recebimento do *quaternion* enviado pelo *SAMPLER*. No caso de exceder o tempo limite de espera pelo *quaternion*, o *MASTER* detecta esse funcionamento incorreto do *SAMPLER*, reinicia-o e acorda o *SLEEPER* para tornar-se seu novo parceiro *SAMPLER* no calcula da atitude. Um *bit-flip* inserido no registrador T2CON do MCU *SAMPLER*, cessa o envio do *quaternion* calculado e o *MASTER* detecta esse funcionamento inadequado, reinicia o *SAMPLER* e acorda o *SLEEPER* para ser o novo *SAMPLER*. Porém o SDATF apresentou problemas quando um *bit-flip* foi inserido no registrador T2CON no MCU *MASTER* (Figura 34, obtida em um Analisador Lógico). Não foi possível identificar a(s) causa(s) desse comportamento inadequado.



Legenda:

D0 - UART1 PIC1	D8 - UART1 PIC3
D1 - HeartBeat PIC1	D9 - HeartBeat PIC3
D2 - Quaternion PIC1	D10 - Quaternion PIC3
D3 - Quadro de 1s PIC1	D11 - Quadro de 1s PIC3
D4 - UART1 PIC2	D12 - *Bit-flip* no PIC1
D5 - HeartBeat PIC2	D13 - *Bit-flip* no PIC2
D6 - Quaternion PIC2	D14 - *Bit-flip* no PIC3
D7 - Quadro de 1s PIC2	

Figura 34, Cenário de um *bit-flip* no registrador T2CON do MASTER,

Mesmo sem saber a causa do problema, a mesma proposta de solução utilizada com T1CON foi realizada para o T2CON. Deste modo, o problema encontrado no T2CON foi

solucionado inserindo a reconfiguração do temporizador 2 na rotina de sincronismo dos microcontroladores. Com já foi citado, essa rotina é controlada pelo temporizador 3 do MCU *MASTER* e do MCU *SAMPLER* e é responsável pelo sincronismo do quadro de temporização do funcionamento normal do SDATF (Figura 25).

E por fim, o terceiro registrador a apresentar problemas nesse cenário foi o T3CON, que é responsável pela configuração e controle do temporizador 3 [54]. Ele possui a funcionalidade de iniciar e manter o quadro de sincronismo a cada um segundo no MCU *MASTER* e no MCU *SAMPLER*, além de configurar algumas *flags* de sinalização utilizadas pela função principal do *firmware* de determinação de atitude com tolerância a falhas. Essas *flags* controlam o fluxo contínuo da execução do programa, determinando locais de parada na função principal do *firmware*. Esses locais são necessários no programa para esperar um determinado evento acontecer, como por exemplo, a execução da configuração do quadro de sincronismo. Se um SEU provocar uma falha no T3CON, a interrupção do TIMER3 não será tratada, ocorre o travamento do fluxo de execução do *firmware* dentro da função principal (*main*) e não enviando a atitude para o PORTHES (Figura 35). Nesse caso, somente o fluxo da função principal permanece travado, os outros periféricos do MCU (TIMER1, TIMER2 e UART2) continuam funcionando normalmente, não afetando a comunicação realizada entre *MASTER* e *SAMPLER*.

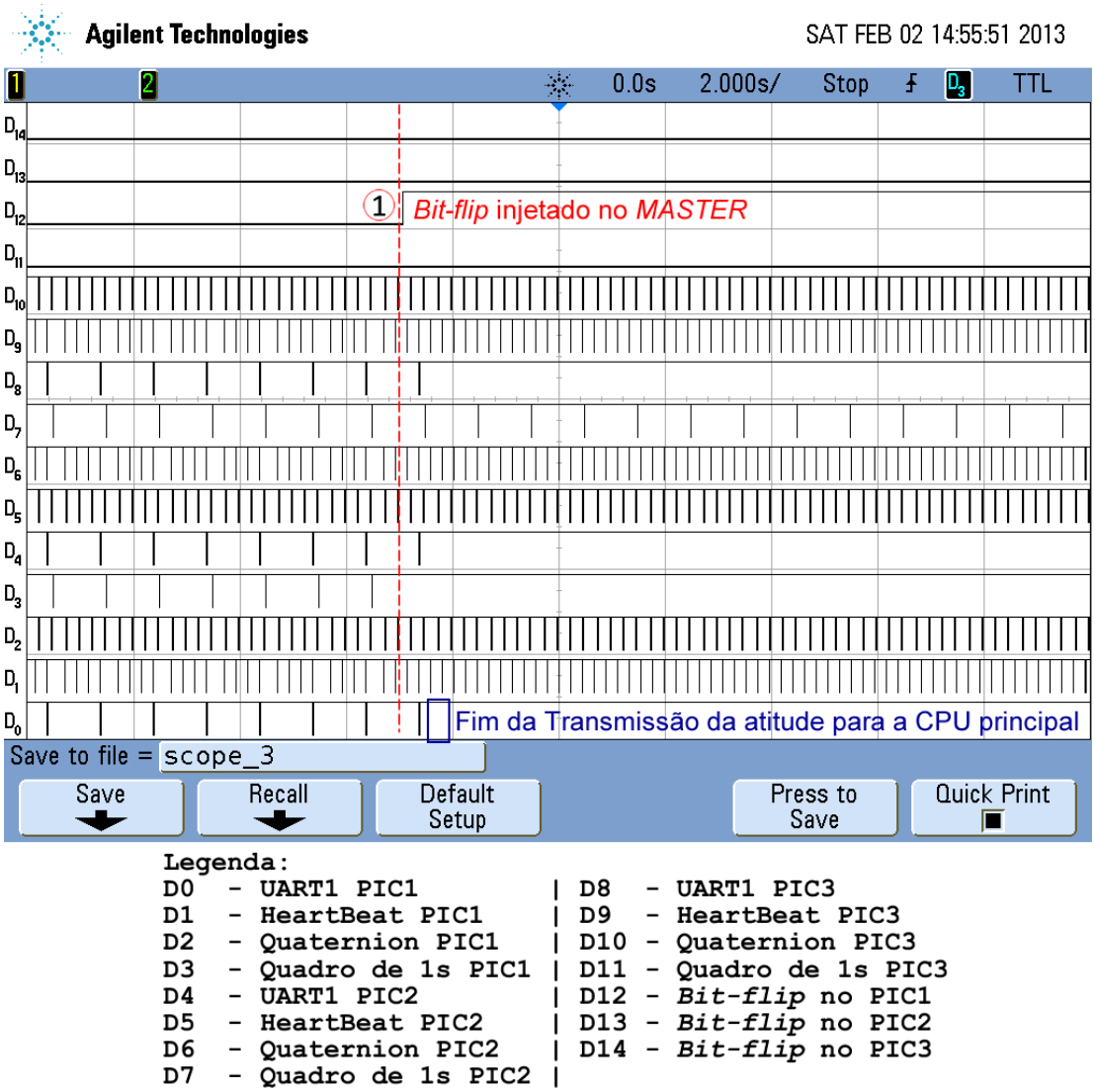


Figura 35. Cenário de um *bit-flip* no registrador T3CON do MASTER.

Na Figura 35, obtida em um Analisador Lógico, é possível observar que o sistema cessa o envio da atitude para a CPU principal, logo após a injeção do *bit-flip* no registrador T3CON do MCU *MASTER*, representada pelo número (1). Também é possível visualizar que a comunicação entre *MASTER* e *SAMPLER* permanece ininterrupta. Portanto, os microcontroladores continuaram a desempenhar suas respectivas funções, com exceção do envio do *quaternion* do sistema para o PORTHES (Figura 35).

A solução proposta anteriormente usada para solucionar os problemas dos registradores T1CON e T2CON, não pôde ser aplicada para esse registrador, pois não justifica a interrupção do TIMER3 reconfigurar ele mesmo. Portanto a solução partiu da ideia de redundância utilizada em *softwares* computacionais, para prevenir falhas no sistema. O problema foi solucionado, por meio da configuração de um novo temporizador (TIMER5)

com as características do temporizador 3. Com uma diferença entre eles, um realiza a reconfiguração do outro, ou seja, o temporizador 3 executa a reconfiguração do temporizador 5 e vice-versa.

Esse primeiro cenário de testes foi importante para identificar e diagnosticar problemas de funcionamento no SDATF que proporcionaram os ajustes na solução tolerante a falhas, descritos nessa seção. Os problemas detectados foram nos periféricos responsáveis pelo sincronismo do quadro da comunicação entre *MASTER* e *SAMPLER* e da comunicação entre *MASTER* e o PORTHES. Como foi descrito, esses problemas foram críticos, pois provocaram o travamento no sistema, parando o envio de dados para o PORTHES e afetando a medida de disponibilidade do SDATF. Desta forma, o sistema PORTHES permitiu a observação desse comportamento inadequado do SDATF, expondo possíveis defeitos no projeto e na implementação da primeira versão do *firmware* de tolerância a falhas, mostrando a necessidade de modificações e adaptações no *firmware* desenvolvido. Uma nova versão do SDATF com todas as modificações descritas foi usada nos experimentos realizados no segundo e terceiro cenários.

5.2.3.2. Segundo cenário experimental com o SDATF:

Nesse segundo cenário o mapeamento dos elementos de memória, sensíveis do sistema tolerante a falhas foi realizado.

Assim esse cenário foi montado para que ele emulasse a taxa de chegadas de partículas radioativas como descrito na seção 2.4. O segundo cenário foi preparado de forma semelhante ao primeiro cenário, um alvo de cada vez era configurado na rotina de tratamento de interrupção de eventos externos, no *firmware* de determinação de atitude com tolerância a falhas. A diferença desse novo cenário para o anterior estava na configuração do modo de injeção de *bit-flips* do sistema PORTHES. Nessa bateria de testes foi usado o modo Automático 2, que injeta os *bit-flips* automaticamente pela interface gráfica do PORTHES em três DUTs, que são selecionados um a cada por injeção de *upset* de forma aleatória, obedecendo à distribuição de Poisson com parâmetro *lambda* de 2,3 *upsets* por segundo. Consequentemente, foi possível identificar o comportamento do sistema e os efeitos causados pela inserção de *upsets* nesse alvo específico. A Tabela 21 apresenta os resultados obtidos nos testes do SDATF nesse segundo cenário configurado.

Tabela 21. Resultados dos testes com SDATF, segundo cenário.

SFR*	*Bit-flips* injetados*	Efeito causado pelo *upset***			Disponibilidade do sistema
		TE	RE	SL	
T1CON	304	87,59%	0%	12,41%	91.09%
TMR1	303	89,51%	1,23%	9,26%	90,08%
PR1	346	86.52%	1,12%	12,36%	85,60%
T2CON	296	99,32%	0%	0,68%	100%
TMR2	291	100%	0%	0%	100%
PR2	318	98,80%	0%	1,20%	99,20%
T3CON	327	100%	0%	0%	100%
TMR3	292	100%	0%	0%	100%
PR3	326	100%	0%	0%	100%
T5CON	273	100%	0%	0%	100%
TMR5	278	100%	0%	0%	100%
PR5	267	100%	0%	0%	100%
IFS0	397	62,36%	28,65%	8,99%	89,6%
IFS1	437	70,98%	24,11%	4,91%	93,60%
IEC0	358	2,36%	1,57%	96,07%	3,20% (P)
IEC1	278	100%	0%	0%	100%
IPC0	279	100%	0%	0%	100%
IPC1	271	100%	0%	0%	100%
IPC2	292	100%	0%	0%	100%
U2MODE	337	93,67%	5,88%	0%	100%
U2STA	330	94,84%	5,16%	0%	100%
U2BRG	334	70,22%	29,78%	0%	100%
U2TXREG	347	23.30%	75.99%	0,71 %	99,20%
U2RXREG	265	100%	0%	0%	100%
INT0	278	100%	0%	0%	100%
INT1	297	100%	0%	0%	100%
INT2	333	100%	0%	0%	100%
CLKDIV	284	48,41%	51,58%	0%	100%
WREG0	278	100%	0%	0%	100%
WREG1	287	99,21%	0,79%	0%	100%
WREG2	294	43,68%	56.32%	0%	100%
WREG3	278	100%	0%	0%	100%
WREG4	323	100%	0%	0%	100%
WREG5	337	57,14%	0,79%	42,07%	58,40% (P)
WREG6	298	99,24%	0.76%	0%	100%
WREG7	304	100%	0%	0%	100%
WREG8	274	100%	0%	0%	100%
WREG9	313	100%	0%	0%	100%
WREG10	339	100%	0%	0%	100%
WREG11	297	100%	0%	0%	100%
WREG12	289	100%	0%	0%	100%
WREG13	290	100%	0%	0%	100%
WREG14	394	5,38%	1,54%	93.08%	3,20% (P)
WREG15	312	100%	0%	0%	100%
RPOR4	303	100%	0%	0%	100%
RPINR0	267	100%	0%	0%	100%
RPINR1	269	100%	0%	0%	100%
RPINR19	287	100%	0%	0%	100%
TRISA	276	100%	0%	0%	100%
TRISB	355	7,74%	38,09%	54,17%	10,40% (P)
PORTA	300	100%	0%	0%	100%
PORTB	262	36,05%	63,95%	0%	99,2%
LATA	463	3,21%	28,84%	67,95%	3,27% (P)
LATB	308	97,67%	2,33%	0%	100%
SPLIM	383	45,16%	7,09%	47,74%	36,80% (P)
PC - asm ("GOTO 0x000000")	349	89,58%	9,90%	0,52%	90,0%
PC - asm ("GOTO 0x000002")	351	36.97%	62,56%	0,47%	93,6%

* Cada bateria de sessão experimental teve a mesma duração de tempo (2 min).

** Valores médios calculados por três observações experimentais.

Legenda: TE = *Tolerated Errors*, RE = *Result Errors*, SL = *Sequence Loss* e (P) = *travamento do sistema*.

O segundo cenário de testes mostrou que as modificações realizadas no primeiro cenário foram suficientemente satisfatórias para eliminar a maior parte dos problemas encontrados no comportamento de falhas observadas nos registradores T2CON e T3CON. Na Tabela 21, observa-se que tais registradores apresentaram cem por cento de ausência de erros nesse cenário de testes. O mesmo resultado não foi observado com a solução proposta para o registrador T1CON, como é possível visualizar na Tabela 21. A grande maioria dos outros registradores usados pelo SDATF também apresentaram cem por cento ausências de erros nesse cenário de testes.

Pode-se observar que, na Tabela 21, alguns registradores indicados com o símbolo (P) provocaram travamento do sistema mesmo após encerrar o processo de injeção de *bit-flips* do PORTHES. Entre esses registradores estão: o registrador U2TXREG, utilizado pelo periférico de comunicação UART2, que atua na transmissão de dados entre *MASTER* e *SAMPLER*, o WREG14, usado como ponteiro de *frame* definido pelo usuário da pilha (*Stack*), tem a função de alocar na memória o endereço das variáveis locais (mudança de contexto) e o LATA, que possui os pinos para reiniciar os outros dois MCUs, entre outros. Esses problemas observados no segundo cenário de testes indica a necessidade de um estudo mais aprofundando de melhorias no SDATF para solucionar os problemas apresentados.

A Tabela 21 também mostra casos de registradores que mesmo apresentando erros de perda de sequência, o sistema tolerante a falhas foi capaz de retomar o seu funcionamento sem intervenção externa (Figura 36). Um desses casos é o registrador IFS0, que apresentou erros por perda de sequência, e mesmo após o travamento do sistema o SDATF recuperou-se e retomou a comunicação com o PORTHES, conforme mostrado na Figura 36. Através da amplificação da imagem do canal digital D_0 responsável pelo envio da atitude para o PORTHES, é possível visualizar o instante do experimento, que o sistema demorou 1,57s para enviar a próxima atitude, caracterizando o travamento do sistema. Mesmo assim, após a latência de tempo citada, o sistema retomou a comunicação com o PORTHES. Isso foi observado em diversos outros casos similares apresentados na Tabela 21.

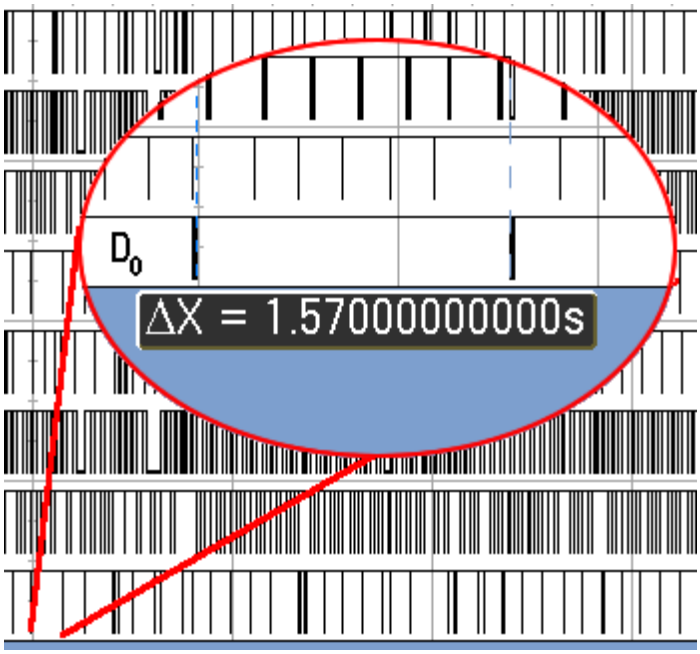


Figura 36. Teste do registrador IFS0, segundo cenário.

5.2.3.3. Terceiro cenário experimental com o SDATF:

Um terceiro cenário para testar o SDATF foi configurado, inserindo um trecho de código CEU *code* na rotina de tratamento de interrupção com os alvos mapeados na área sensível. Esses alvos foram distribuídos dentro de uma estrutura "*switch case*" no *firmware* de MCU, com os mesmo percentuais da sessão de testes realizada na seção 5.2.2 com cem casos do PIC24F. Foram configurados da seguinte maneira: 50% dos casos na estrutura do "*switch case*" no CEU *code* configurado foram alvos que produzem erros toleráveis ou ausência de erros (*Tolerated Errors*); 47% representam os alvos que provocam na maioria *Result Errors* e 3% referem-se a alvos que geram erros de perda de sequência ou travamento do sistema (*Sequence Loss*). O modo de injeção de *bit-flips* escolhido no PORTHES foi o Automático 2, emulando SEUs segundo a distribuição de *Poisson* com a taxa de 2,3 *upsets* por segundo. A Tabela 22 mostra os resultados obtidos com o sistema de determinação de atitude com tolerância a falhas no terceiro cenário descrito.

Tabela 22. Resultados do teste com SDATF, terceiro cenário.

	Falhas* injetadas	**Total de erros**	**Erros toleráveis ou ausência de erros**	***Result Errors***	**Erros por perda de sequência**	**Disponibilidade do sistema**
Valores absolutos*	13590	5723	5491	223	9	---
Percentuais*	-	100%	95.95%	3,89%	0,16%	99.73%

* Valores médios calculados por três observações experimentais.

Por meio desse terceiro cenário, foi possível calcular a disponibilidade do sistema de determinação de atitude com tolerância a falhas sob o efeito da e SEUs a uma taxa de 2,3 *upsets* por segundo.

As baterias de testes do segundo e do terceiro cenários permitiram observar problemas no SDATF, mostrando a necessidade de um estudo mais aprofundado sobre as razões da ocorrência de comportamentos não previsíveis e consequentemente realizar modificações no projeto do sistema tolerante a falhas.

Capítulo 6 - CONCLUSÃO

O valor da cross-section mensurada em experimentos com o PORTHES e por consequência a SER calculada a partir desta, dependem não só das condições ambientais e da tecnologia de fabricação usada na produção do DUT, mas principalmente da aplicação que está em execução no DUT, não só quanto à quantidade de memória usada pela aplicação, mas principalmente da frequência com que as saídas do sistema DUT+aplicação estão sendo monitoradas.

O PORTHES se mostrou como uma ferramenta configurável capaz de emular SEU em microcontroladores programados com aplicações que possam ser interrompidas por algum sinal externo e ter suas saídas monitoradas e esperadas com um determinado padrão conhecido e aceitável como correto.

O tamanho do CEU *code* e o tempo de processamento da execução da rotina de tratamento de interrupção do CEU *code*, podem interferir na emulação de SEU em um conjunto DUT+aplicação em teste, se o tempo de processamento dessa rotina de tratamento de interrupção for percentualmente significante a modo de interferir no processamento normal da aplicação.

Um sistema de emulação a SEU baseado em hardware e/ou software, tal como o PORTHES ou o THESIC, não pode ser completamente automatizado sem o conhecimento da arquitetura, organização e tecnologia de fabricação empregada no projeto e construção do DUT.

Esse trabalho também comprova que na atualidade, qualquer outro sistema de emulação a SEU baseado em hardware e/ou software não pode ser completamente automatizado sem o total conhecimento e sem o acesso ao firmware gravado no microcontrolador.

A SER é dependente dos elementos de memória utilizados pelo firmware gravado no DUT, os quais provocam erros observáveis em pelo menos uma das saídas do DUT+aplicação, aferidos pelo sistema de emulação PORTHES.

Conseguiu-se estabelecer um método semi-automatizado para injeção de falhas SEU em microcontroladores executando aplicações, usando-se de ferramentas de aquisição de dados de baixo custo.

Conseguiu-se provar que SEU podem ser emulados em laboratório com ferramentas de baixo custo.

Conseguiu-se avaliar experimentalmente a disponibilidade de um sistema de determinação de atitude desenvolvido com técnicas de tolerância a falhas.

A principal contribuição desse trabalho é o desenvolvimento da ferramenta PORTHES e a elaboração de um método adaptado de emulação de SEU conhecido *Code Emulating Upsets* em conjuntos de DUTs microcontrolados + aplicações.

Futuros ensaios com outros conjuntos de DUT + aplicações se fazem necessários a fim de se atestar as limitações e versatilidades do PORTHES.

Futuras melhorias no sistema PORTHES necessitam ser implementadas, como por exemplo, a incorporação de funcionalidades estatísticas para que o sistema possa realizar todas as análises dos dados.

Alterações futuras no sistema de determinação de atitude com tolerância a falhas serão necessárias para que esse sistema atinja índices de confiabilidade exigidos para os sistemas embarcados em satélites artificiais de baixa órbita terrestre.

Por fim, o PORTHES e o método usado para emular SEUs em conjuntos de DUT+aplicações mostraram-se úteis na validação do SDATF descrito nesse trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. Nicolaidis, *Soft Errors in Modern Electronic Systems*. New York: Spring-Verlag, 2010.

[2] "USGS – U.S. Geological Survey – Science for a changing world," http://landsat.usgs.gov/science_an_seu.php, April 4, 2012.

[3] P. E. Dodd and L.W. Massengill, "Basic Mechanism and Modeling of Single-Event Upset in Digital Microelectronics," *IEEE* Transaction on Nuclear Science, vol. 50, issue: 3, page(s): 583-602, June 2003.

[4] F. L. Kastensmidt, "SEE Mitigation Strategies for Digital Circuit Design Applicable to ASICs and FPGAs", Part II of the Short Course presented at the 2007 Nuclear and Space Radiation Effects Conference, Honolulu, HI, July, 23, 2007.

[5] R. Baumann and E. Smith, "Neutron-induced boron fission as a major source of soft errors in deep submicron SRAM devices", in Proceedings of *IEEE* International Reliability Physics Symposium. 38th Annual 2000 *IEEE* International, 2000, pp. 152-157, 2000.

[6] J. F. Ziegler and W. A. Lanford, "Effect of cosmic rays on computer memories". Science, Volume 206, Issue 4420, pp. 776-788.

[7] M. Omana, G. Papasso, D. Rossi, and C. Metra, "A Model for Transient Fault Propagation in Combinatorial Logic," in Proc. of 9th *IEEE* Intl. On-Line Testing Symposium, pp. 111-115, Kos Island, Greece, June 2003.

[8] A. Nieuwland, S. Jasarevic, and G. Jerin, "Combinational Logic Soft Error Analysis and Protection," in Proc. of IEEE International On-line Testing Symposium, 12th *IEEE* International, IOLTS 2006.

[9] R. Velazco, P. Fouillat and R. Reis (Ed.). *Radiation effects on embedded systems*. Dordrecht: Springer, 2007.

[10] S. Mukherjee, *Architecture Design for Soft Errors*. Burlington: Elsevier, Inc. Feb 2008.

[11] F. L. Kastensmidt, R. R., L. Carro, *Fault-Tolerance Techniques for SRAM-Based FPGAs (Frontiers in Electronic Testing)*, Springer, 2006.

[12] J. R. Schwank, M. R. Shaneyfelt, D. M. Fleetwood, J. A. Felix, P. E. Dodd, P. Paillet, V. Ferlet-Cavrois, "Radiation Effects in MOS oxides", IEEE Transactions on Nuclear Science, Snowmass Village, v. 55, n. 4, pp. 1833-1853, Aug. 2008.

[13] T. R. Balen, Efeitos da Radiação em Dispositivos Analógicos Programáveis (FPAAs) e Técnicas de Proteção. Porto Alegre: UFRGS, 2010. 206 p. Tese (Doutorado) – Programa de Pós-Graduação em Engenharia Elétrica, Escola de Engenharia, Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2010.

[14] J. R. Srour, C. J. Marshall and P. W. Marshall, "Review of displacement damage effects in silicon devices", IEEE Transactions on Nuclear Science, [S.1.], v.50, n.3, pp. 653-670. June 2003.

[15] R. Hatcher, R.D. Schrimpf, D.M. Fleetwood, S.T. Pantelides, "Quantum Mechanical Description of Displacement Damage Formation", IEEE Transactions on Nuclear Science, v. 54, n. 6, pp.1906-1912. December, 2007.

[16] F. Wang, V. D. Agrawal, "Single event upset: an embedded tutorial", in Proceedings of IEEE Internacional Conference on VLSI Design, 21., 2008, [S. 1.]:[s. n.]. p. 429-434, 2008.

[17] R. Ecoffet, "In-flight Anomalies on Electronic Devices". Chapter in: R. Velazco, P. Fouillat and R. Reis (Ed.), *Radiation effects on embedded systems*. Dordrecht: Springer, 2007. p. 31-68.

[18] A. B. de H. Ferreira, Novo Dicionário da Língua Portuguesa. 2. ed. Rio de Janeiro: Nova Fronteira, 1986. 1838 p.

[19] W. F. Temporal, G. da F. Oliveira, R. L. C. de Campos, M. S. Galizia. Radiação Cósmica e Voo. Rio de Janeiro: RMAB, Jan./Dez. 2005. Disponível em: <http://www.dirsa.aer.mil.br/revistas/2005/02_05.pdf>. Acesso em: 10 de janeiro de 2013.

[20] J. C. Boudenot, "Radiation Space Environment". Chapter in: R. Velazco, P. Fouillat and R. Reis (Ed.), *Radiation effects on embedded systems*. Dordrecht: Springer, 2007. p. 1-9.

[21] T. J. O'Gorman, "The Effect of Cosmic Rays on the Soft Errors of a DRAM at Ground Level", IEEE Transactions on Electron Devices, v. 41, n. 4, pp. 533-557, Apr. 1994.

[22] ESA – European Space Agency. Documento on-line, disponível em: http://esamultimedia.esa.int/HSO/Radiation_v1_pt.pdf. Acesso em: 09 de janeiro de 2013.

[23] ESA – European Space Agency. Disponível em: http://spaceinimages.esa.int/Images. Acesso em: 10 de janeiro de 2013.

[24] E. Stassinopoulos, J. Raymond, “The Space Radiation Environment for Electronics”. Proceedings of the IEEE, v. 76, n. 11, p. 1423-1442, Nov. 1988.

[25] J. F. Ziegler, “Terrestrial cosmic rays”. IBM Journal of Research and Development, v. 40, n. 1, p. 19-39, Jan. 1996.

[26] NASA – National Aeronautics and Space Administration. Disponível em: http://apod.nasa.gov/apod/ap060814.html. Acesso em: 11 de janeiro de 2013.

[27] NASA – National Aeronautics and Space Administration. Disponível em: http://www.nasa.gov/mission_pages/sunearth/news/News041612-M1.7flare.html. Acesso em: 10 de janeiro de 2013.

[28] NOAA National Geophysical Data Center (NGDC). Disponível em: http://www.ngdc.noaa.gov/ngdcinfo/onlineaccess.html. Acesso em: 11 de janeiro de 2013.

[29] Encyclopedia Britannica – Britannica Academic Edition. Disponível em: http://www.britannica.com/EBchecked/media/60532/The-Van-Allen-radiation-belts-contained-within-Earths-magnetosphere. Acesso em: 11 de janeiro de 2013.

[30] NASA – National Aeronautics and Space Administration. Disponível em: http://radbelts.gsfc.nasa.gov/outreach/RadMovies.html. Acesso em: 11 de janeiro de 2013.

[31] C.R.A. Augusto, C.E. Navia, K.H. Tsui, H. Shigueoka, P. Miranda, R. Ticona, A. Velarde and O. Saavedra, “Simultaneous observation at sea level and at 5200 m.a.s.l. of high energy particles in the South Atlantic Anomaly”, Astroparticle Physics: ScienceDirect.com, v. 34, n. 1, p. 40-47. Disponível em: http://www.sciencedirect.com/science/article/pii/S0927650510000800. Acesso em: 11 de janeiro de 2013.

[32] Paranormal – Encyclopedia.com. Artigo: The South Atlantic Anomaly. Disponível em: http://www.paranormal-encyclopedia.com/s/south-atlantic-anomaly. Acesso em: 11 de janeiro de 2013.

[33] SPENVIS. European Espace Agency: space environment information system. Disponível em: <http://www.spenvis.oma.be/help/background/traprad/traprad.html>. Acesso em: 11 Jan. 2013.

[34] S. Higuereta, D. Husson, M. Trocmé, A. Nourreddine, T.D. Lê, N. Michielsen, “Measurement of 222Rn at the Bq m−3 level with the AlphaRad chip”, Science Direct, Radioation Measirements, vol. 43, issues 2-6, p. 1059-1062, Feb-June 2008.

[35] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing”, IEEE Transactions on Dependable and Secure Computing, v. 1, n. 1, p. 11-33, Jan-March 2004.

[36] F.R. Palomo, J.M. Mogollon, J. Napoles, H. Guzman-Miranda, A.P. Vega-Leal, M.A. Aguirre, P. Moreno, C. Mendez, and J.R. Vazquez de Aldana, “Pulsed Laser SEU Cross-Section measurement using coincidence detectors,” Radiation and Its Effects on Components and Systems (RADECS), pp. 147-151, Sept. 2008.

[37] P. Fouillat, V. Pouget, D. McMorrow, F. Darracq, S. Buchner, and D. Lewis, “Fundamentals of the Pulsed Laser Technique for Single-Event Upset Testing”. Chapter in: R. Velazco, P. Fouillat and R. Reis (Ed.), *Radiation effects on embedded systems*. Dordrecht: Springer Netherlands – 2007, page(s): 121-144.

[38] M.-C. Hsueh, T.K. Tsai and R.K. Iyer, "Fault Injection Techniques and Tools," Computer, vol. 30, no. 4, pp. 75-82, Apr. 1997.

[39] Michael Nicolaidis and Raoul Velazco, “Architectures for Robust and complex Integrated Systems (ARIS)” – TIMA Annual Report 2008, http://tima.imag.fr/research/files/gr-08/aris.pdf, April 18, 2012.

[40] S. Rezgui, R. Velazco, R. Ecoffet, S. Rodriguez, and J. R. Mingo, “Estimating Error Rates In Processor-Based Architectures”, IEEE Transactions On Nuclear Science, vol. 48, no. 5, page(s): 1680-1687, Oct 2001.

[41] R. Velazco, S. Rezgui, E. Reguer, “THESIC: Una plataforma flexible para la validación funcional de circuitos integrados”, VII Workshop IBERCHIP, Montevideo Uruguay, March 21-23, 2001.

[42] R. Velazco and S. Rezgui, “Transient Bitflip Injection in Microprocessor Embedded Applications,” Procs. 6th Int. On-Line Testing Workshop (IOLTW). Palmas de Mallorca Spain, pp. 80-84, July 2000.

[43] R. Velazco, S. Rezgui, and R. Ecoffet, “Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection”, IEEE Transactions On Nuclear Science, vol. 47, no. 6, page(s): 2405-2411, Dec 2000.

[44] F. Faure, R. Velazco, and P. Peronnard, “Single Event Upset like Fault Injection: a Comprehensive Framework,” IEEE Nuclear and Space Radiation Effect Conference (NSREC 2005), vol. 52, issue: 6, pp. 2205-2209, July 12-15, 2005.

[45] The SPARC *Architecture Manual Version* 8, SPARC Int. Inc., Menlo Park, 1992. Disponível em: http://www.sparc.com/standards/V8.pdf. Acesso em: 20 de janeiro de 2013.

[46] D. K. Pradhan, *Fault-Tolerant Computing: Theory and Techniques*, vol. I, Prentice Hall, New Jersey, 1996.

[47] S. Haykin, and B. V. Veen, *Sinais e Sistemas*. Porto Alegre: Bookman, 2001.

[48] National Instruments. Disponível em: http://www.ni.com. Acesso em: 10 de janeiro de 2013.

[49] NI USB-6212. 16-Bit, 400 kS/s M Series MIO DAQ, Bus-Powered - National Instruments. Disponível em: http://sine.ni.com/nips/cds/view/p/lang/pt/nid/207096. Acesso em: 10 de janeiro de 2013.

[50] M. T. Cabral, "Desenvolvimento de um Protótipo de um Sistema de Determinação de Atitude de Satélites Artificiais com Tolerância a Falhas". Monografia de Graduação, Colegiado do Curso de Engenharia de Controle e Automação, Escola de Minas, Universidade Federal de Ouro Preto, Ouro Preto, Minas Gerais, 2010.

[51] R. O. Duarte, F. E. Torres, T. H. Gomes. L. S. Martins-Filho, H. K. Kuga, "An Attitude Determination System Implementation to Low Orbit Small Satellite with Fault-Tolerant Techniques", 8th IAA Symposium on Small Satellites for Earth Observation, 4-8 April 2011, Berlin, Germany.

[52] A. C. de Souza e L. S. Martins Filho, "Controle da Atitude de Satélites Artificiais". Documento online, disponível em: http://ic.ufabc.edu.br/II_SIC_UFABC/resumos/paper_5_202.pdf. Acesso em: 17 de janeiro de 2013

[53] V. T. L. Rampinelli, "Controle de Atitude de Satélites Artificiais". Monografia de Graduação, Colegiado do Curso de Engenharia de Controle e Automação, Escola de Minas, Universidade Federal de Ouro Preto, Ouro Preto, Minas Gerais, maio 2006.

[54] Microchip, "PIC24FJ64GA004 Family Data Sheet", PIC24FJ64GA002 datasheet, rev. DS39881B. Disponível em: http://ww1.microchip.com/downloads/en/devicedoc/39881b.pdf. Acesso em: 02 de janeiro de 2013.

[55] J. Axelson, *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*, 2ª Ed., Lakeview Research LLC, Madison, 2007.

[56] Microchip, “MPLAB® IDE User’s Guide with MPLAB Editor and MPLAB SIM Simulator”, datasheet, rev. DS51519C. Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_User_Guide_51519c.pdf . Acesso em: 02 de janeiro de 2013.

[57] D. R. de Sousa and D. J. de Souza, *Desbravando o PIC24: Conheça os Microcontroladores de 16 bits*, 1ª Ed., Editora Erica, São Paulo, 2008.

[58] H. S. Sandhu, *Making PIC Microcontroller Instruments and Controllers*, 1ª Ed., The McGraw-Hill Companies, New York, 2009.

[59] Microchip, “dsPIC30F Programmer’s Reference Manual”, datasheet, rev. DS70030F. Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/70030f.pdf. Acesso: 13 de janeiro de 2013.

[60] F. Pereira, *Microcontroladores PIC: Técnicas Avançadas*, 1ª ED., Editora Erica, São Paulo, 2002.

[61] L. Di Jasio, “Programming 16-bit Microcontrollers in C – Learning to Fly the PIC24”, 1ª Ed., Elsevier, Burlington, 2007.

[62] Z. Milivojević e D. Šaponjić, “Programming dsPIC MCU in C”. E-Book, disponível em: http://www.mikroe.com/products/view/266/programming-dspic-mcu-in-c. Acesso em: 17 de janeiro de 2013.

[63] MATLAB GUI, The Math Works Inc.. Disponível em: http://www.mathworks.com/discovery/matlab-gui.html. Acesso em: 13 de janeiro de 2013.

[64] AN1229 – Application Note da Microchip, “Class B Safety Software Library for PIC® MCUs and dsPIC® DSCs”, manual, rev. DS01229C. Disponível em: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en537355. Acesso: 03 de fevereiro de 2013.