UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

eOSI: Um modelo para Desenvolvimento de Sistemas Embarcados Tolerantes a Falhas

Antônio Higor Freire de Morais

Orientador: Prof. Dr. Gláucio Bezerra Brandão

Co-Orientadora. Profª. Dra. Ana Maria Guimarães Guerreiro

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Natal/RN, 17 de Julho de 2009.

Divisão de Serviços Técnicos

Catalogação da Publicação na Fonte. UFRN / Biblioteca Central Zila Mamede

Morais, Antônio Higor Freire de.
eosi: um modelo para desenvolvimento de sistemas embarcados tolerantes a falhas / Antônio Higor Freire de Morais. – Natal, RN, 2009.
107 f.

Orientador: Gláucio Bezerra Brandão.
Co-orientador: Ana Maria Guimarães Guerreiro.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e Computação.

1. Tolerância a falha – Dissertação. 2. Sistemas embarcados – Dissertação. 3. FPGA – Dissertação. I. Brandão, Gláucio Bezerra. II. Guerreiro, Ana Maria Guimarães. III. Universidade Federal do Rio Grande do Norte. IV. Título.

RN/UF/BCZM CDU 681.5.09(043.3)

Antônio Higor Freire de Morais

Projeto de Dissertação de Mestrado aprovada em _____ de _____ de 2009 pela banca examinadora composta pelos seguintes membros:

_____

Prof. Dr. Gláucio Bezerra Brandão (Orientador).....DCA/UFRN

_____

Profa. Dra. Ana Maria Guimarães Guerreiro (Co-Orientador).....DCA/UFRN

_____

Prof. Dr. Ricardo Alexandro de M. Valentim (Avaliador Externo).....DATINF/IFRN

“A glória de Deus é ocultar as coisas,
a glória dos reis é pesquisá-las.” (Pr 25, 2)

Dedico este trabalho à minha namorada
Por ser uma pessoa com um coração de ouro;
Pelo seu carinho e pelo seu amor.
Por isto, este trabalho é para você minha Princesa.

Agradecimentos

Agradeço ao meu Bom Deus por não ter permitido que eu vacilasse em momento algum e por ter acalmado meu coração nos momentos de aflição.

À minha mãe, Francisca, e à minha bisavó, mãe Ana, meu muito obrigado por tudo que fizeram em minha vida. Obrigado pela formação do meu caráter e pelo amor incondicional.

Agradeço a minha querida namorada, Josi, que mesmo distante serviu-me como uma fonte de energia para que eu chegasse até este último degrau.

Ao meu amigo e orientador, Gláucio, só tenho que agradecer por todas as sábias perguntas que me foram feitas, fundamentais para o desenvolvimento desta Dissertação. Muito obrigado também pela ajuda dispensada, minha amiga e co-orientadora Ana Maria.

Meu caro amigo e parceiro Ricardo, todos os nossos debates foram igualmente importantes para que eu chegasse ao final dessa jornada. Muito obrigado amigo.

Um agradecimento especial a toda a turma do Laboratório de Automação Hospitalar e Bio-engenharia – LAHB. Obrigado ao meu camarada Diego (Diegão o homem do café, na ausência de Ricardo), Heitor (sempre com uma rapadura), Hélio (HB2 e seus comentários e suposições inesperadas). Obrigado a todos vocês pelos momentos de descontração e estudos.

Obrigado aos meus caros colegas da UFRN e aos servidores que nos dão o apoio necessário para possamos desenvolver nossas pesquisas.

Resumo

A evolução das tecnologias em semicondutores possibilita que dispositivos sejam desenvolvidos cada vez mais com uma maior capacidade de processamento. Neste sentido, estes componentes passam a ter sua utilização ampliada para um maior campo de atuação. Ambientes da indústria petroleira, mineração, automotivos e hospitalares são exemplos de setores que estão utilizando tais dispositivos com maior frequência em seus processos. As atividades que são desenvolvidas por estas indústrias estão diretamente envolvidas com a segurança ambiental e a saúde daqueles que nela trabalham. Desta forma, torna-se mister a utilização de sistemas que sejam dotados de características de segurança extra que possam conferir a estes sistemas maior confiabilidade, segurança e disponibilidade. O modelo de referencia eOSI que será apresentado por esta Dissertação tem por objetivo permitir que estes sistemas sejam desenvolvidos sob uma nova perspectiva que facilite a escolha das estratégias de tolerância a falha a serem empregadas na aplicação. Como forma de validar a utilização deste modelo será apresentada uma arquitetura de suporte que foi desenvolvida em FPGA com base neste modelo.

Palavras chaves: Tolerância a falha, Sistemas embarcados, FPGA.

Abstract

The semiconductor technologies evolutions leads devices to be developed with higher processing capability. Thus, those components have been used widely in more fields. Many industrial environment such as: oils, mines, automotives and hospitals are frequently using those devices on theirs process. Those industries activities are direct related to environment and health safe. So, it is quite important that those systems have extra safe features yield more reliability, safe and availability. The reference model eOSI that will be presented by this work is aimed to allow the development of systems under a new view perspective which can improve and make simpler the choice of strategies for fault tolerant. As a way to validate the model an architecture FPGA-based was developed.

Key words: Fault Tolerant, Embedded Systems, FPGA.

Sumário

Lista de Figuras

Lista de Equações

Lista de Quadros

Lista de Gráficos

Lista de Acrônimos

ANSI/ISA	*American National Standards Institute/Instrument Society of America*
ASIC	*Application-Specific Integrated Circuit*
CI	Circuito Integrado
CISC	*Complex Instruction Set Computer*
CLB	*Configurable Logic Block*
CPLD	*Complex Programmable Logic Device*
CRC	*Cyclic Redundancy Check*
DSE	*Design Space Eploration*
ECC	*Error Detection Code*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
EDA	*Eletronic Design Automation*
ESL	*Eletronic System Level*
FPGA	*Field-Programmable Gate Array*
FPLA	*Field-Programmable Logic Array*
ILP	*Integer Linear Programing*
IP	*Intellectual Property*
IPI	*Intellectual Property Interface*
ISO	*International Standards Organization*
KNP	*Khan Process Network*
LUT	*Lookup-Table*
MSRS	*Motorola Semiconductor Reuse Standards*
MPSoC	*Micro-Processor System-on-Chip*
NASA	*National Aeronautics and Space Administration*
NMR	*N-Modular Redundancy*
NoC	*Network-on-Chip*

OSI	*Open Systems Interconection*
PAL	*Programmable Array Logic*
PCI	*Peripheral Component Interconnect*
PLD	*Programmable Logic Device*
PROM	*Programmable Read-Only Memory*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
RF	Rádio Frequência
RTL	*Register Transfer Level*
SoC	*System-on-Chip*
TCP	*Transmission Control Protocol*
TMR	*Triple-Modular Redundancy*
ULA	Unidade Lógica Aritmética
VHDL	*Very High Speed Integrated Circuits Hardware Description Language*
VMM	*Virtual Machine Monitor*
XML	*eXtensible Markup Language*

Sessão 1

1. Introdução

O cenário da microeletrônica mundial atualmente passa por uma revolução no desenvolvimento de sistemas embarcados, desde sua forma de concepção física à complexidade com que são projetados. A complexidade e heterogeneidade dos sistemas embarcados que são desenvolvidos vem crescendo rapidamente ao longo da última década [Vermeulen et. al. 2008]. Estes tipos de sistemas microprocessados estão, cada vez mais, fazendo parte do nosso dia-a-dia, sejam eles em computadores pessoais sejam em sistemas automotivos. Neste sentido, faz-se necessário a criação de níveis de abstração para o desenvolvimento de *System-on-Chip* (SoC), de tal forma que a equipe de desenvolvedores possa ser gerenciada e analisada, durante o desenvolvimento do projeto, para que a quantidade de erros (erros de projeto) seja a menor possível [Lahtnen, 2006]. As pessoas que integram a equipe de projetistas de um sistema embarcado são especializadas no desenvolvimento de *software* e *hardware*, cada qual atendendo às necessidades particulares do projeto.

Esta necessidade de criação organizada de projetos fez com que alguns padrões fossem pensados e elaborados com este fim. Atualmente, um conceito que trata das questões referentes às abstrações de projeto é chamado de *Electronic System Level* (ESL). Os conceitos que são abordados pelo ESL estão destinados ao desenvolvimento de cinco metodologias de projetos que podem ser encontradas em [Lahtinen, 2006]. Tais conceitos referem-se a questões de desenvolvimento de *hardware* e *software* para sistemas embarcados de forma concorrente. Algumas das causas que impulsionaram a criação destas metodologias de projetos estão no fato de que o tempo gasto com *re-design* e o risco de erros de projetos tendem a ser menor quando se desenvolve um projeto de forma estruturada.

A forma de comunicação entre *software-software*, *software-hardware* e *hardware-hardware*, bem como o mecanismo de troca de informações entre os

elementos de um sistema embarcado é um tema abordado por [Gervini et. al. 2003]. Estes são fatores de grande importância quando se trata de projetos de SoC, pois estes parâmetros estão intimamente associados: à potência consumida pelo sistema; à área que será ocupada pelo SoC na pastilha de silício; e à velocidade de processamento e desempenho alcançado pelo SoC. Ao se projetar um sistema embarcado, o projetista deve estar atento aos compromissos e especificações do projeto. Normalmente, a melhoria de um requisito afeta as características do outro. Por exemplo, um ganho em desempenho de processamento pode levar ao consumo de uma maior área ocupada pelo processador e/ou aumento de consumo de energia.

A linha que separa cada um destes elementos deve ser observada com atenção para que o requisito de uma dada característica do sistema não comprometa o escopo do projeto. Apesar da importância destes *tradeoffs*[1], este trabalho não está preocupado com a minimização destes fatores no tocante aos projetos de sistemas embarcados. Embora sejam considerações que devam ser observadas quando da elaboração de projetos deste tipo, outras questões relacionadas às metodologias de projeto, segurança de sistemas, integridade e disponibilidade necessitam de um estudo mais aprofundado por parte da comunidade acadêmica.

Os ataques a sistemas embarcados é um tema abordado por Diguet em [Diguet et. al. 2007]. Os autores tratam de questões referentes à integridade de dados e autenticidade de entidades, os quais podem ser verificados através do monitoramento de comportamentos anormais de comunicação [Diguet et. al. 2007]. Temas relacionados à garantia da entrega de dados e fatores intrínsecos de segurança de informação são pontos que podem ser explorados de maneira mais apurada nestes projetos. Não é escopo deste trabalho discutir de forma aprofundada métodos de desenvolvimento de sistemas embarcados visando um menor consumo de área e potência ou melhora de desempenho.

O modelo de referência *Open Systems Interconection* (OSI) surgiu em função da necessidade de se ter um modelo que formalizasse o mecanismo de comunicação entre os computadores. Tal metodologia foi necessária para que os

[1] Termo originário do inglês e utilizado para situações em que há conflito de escolha. Neste caso, a escolha de um determinado componente que melhor dada característica resultará no prejuízo de outra

fabricantes pudessem desenvolver produtos que fossem capazes de trocar informações independentemente de *drivers*. Este modelo foi formalmente desenvolvido pela *International Standards Organization* (ISO) e também é conhecido como modelo ISO/OSI.

A ANSI/ISA-95, ou ISA-95, é um padrão que trata da forma de criação de interfaces entre sistemas de controle e sistemas corporativos. Este padrão visa estabelecer uma forma eficiente para que estas aplicações possam trocar informações de uma forma segura, robusta, com um melhor custo-benefício, de tal forma que a integridade entre os dois sistemas seja preservada [ANSI/ISA–95.00.01, 2000]. Assim como o modelo OSI da ISO, a ISA-95 preocupa-se com a forma de comunicação entre sistemas e, devido a esta preocupação, ela sugere um modelo mais específico para os sistemas de controle e automação.

1.1 Objetivo

O objetivo desta Dissertação é apresentar uma metodologia para desenvolvimento de sistemas embarcados que tem como base um modelo de representação em camadas. O foco desta metodologia de desenvolvimento concentra esforços sobre questões voltadas à disponibilidade de sistemas/dispositivos em redes, utilizando como base conceitos de tolerância à falhas e sistemas distribuídos, empregando técnicas de redundância em *hardware*.

Esta forma de desenvolvimento segmentado irá permitir que o projeto seja desenvolvido com um maior desacoplamento entre as partes e, conseqüentemente, uma maior agilidade para desenvolvimento de projetos. Esta segmentação facilitará a prática de gerenciamento, devido ao fato do projeto estar sendo pensando em camadas logicamente separadas. Isto fará com que o projetista tenha uma visão abstrata da representação do sistema embarcado, aumentando as chances de detecções de falhas de projeto em um tempo menor.

1.2 Objetivos específicos

Os trabalhos científicos são fundamentados em teorias e pesquisas que comprovem a veracidade do que está sendo apresentado. A fundamentação teórica é o suporte que é utilizado para apresentar o objeto de estudo. Como forma de consolidar este trabalho é apresentado um estudo de caso que demonstra um exemplo de arquitetura que emprega os conceitos do modelo de referência *Embedded Open Systems Interconnection* [Morais et. al. 2009]. Este modelo servirá para estabelecer uma divisão hierárquica de níveis de responsabilidades que possam determinar funcionalidades específicas de um sistema embarcado, de tal forma que seja possível um desenvolvimento organizado e estruturado deste sistema. Além disso, este modelo permite uma visão abstrata de alto nível com relação ao sistema que será implementado o que facilita a compreensão deste para a equipe de desenvolvedores. Desta forma, destacam-se como objetivos específicos deste trabalho:

- Estudo das principais propostas no meio acadêmico para metodologias de sistemas embarcados;
- Elaboração de um modelo em camadas para desenvolvimento de sistemas embarcados baseado em redundância em *hardware* e tolerância à falhas;
- Elaboração de uma arquitetura de suporte baseada no modelo de referência eOSI voltada para problemas de disponibilidades;
- Implementações de técnicas de redundância em *hardware* em uma *Field-Programmable Gate Arrays* (FPGA) para aumento da confiança e disponibilidade do sistema e validação da Camada de Redundância do modelo eOSI;

1.3 Organização do trabalho

A estrutura organizacional desta Dissertação está definida da seguinte forma: no Sessão 1 é feita uma breve contextualização do trabalho e apresentação da proposta; no Sessão 2 serão apresentados conceitos referentes à redundância em *hardware*, sistemas tolerantes a falhas, sistemas distribuídos e demais assuntos associados a esta proposta. A análise destas técnicas de tolerância a falhas será necessária para a implementações que serão demonstradas na Sessão 4; a Sessão 3 é destinado ao estudos de trabalhos correlatos e estado da arte; na Sessão 4 são apresentadas as camadas que devem ser utilizadas como referência para os projetos de sistemas embarcados tolerantes a falhas. Como parte da consolidação deste trabalho foi implementado um sistema embarcado baseando-se no modelo de referência eOSI. Tal sistema é parte complementar a Tese de Doutorado desenvolvida por Valentim em [Valentim, 2008] que apresenta um protocolo multiciclo, baseado em IGMP Snooping, para automação hospitalar. A Sessão 5 apresenta o resultado da implementação da arquitetura de suporte tolerante a falha que é apresentada na Sessão 4. Na Sessão 6 são feitas as considerações finais do trabalho de dissertação de mestrado e sugestões a cerca de trabalhos futuros.

Sessão 2

2. Fundamentação Teórica

Nesta Sessão são abordadas as pesquisas e tecnologias em torno do objeto de estudo desta Dissertação, bem como os conceitos básicos que serviram de ferramenta para elaboração da mesma.

2.1. Sistemas Embarcados e Dispositivos Reconfiguráveis

O século 20 foi marcado como um período de grandes revoluções para a ciência e tecnologia no âmbito do processamento de informações. Em 1945, John Von Neumann propôs um modelo de arquitetura para computadores que vem sendo usado até os dias atuais. Sua arquitetura é constituída por três características principais: a existência de um programa; uma unidade de processamento com funcionalidades particionadas; e um ciclo de busca-decodificação-execução [Yi & Lilja, 2006].

O programa é unidade responsável pela descrição das ações que serão tomadas pelo sistema de acordo com os valores de entrada que são passados (interações com um usuário ou valores de sinais digitais, por exemplo). Este programa pode estar gravado diretamente no *hardware* ou armazenado em uma memória, comumente chamada de memória de programa. Para o caso em que o programa está armazenado em memória faz-se necessário que o processador, a cada passo que é executado, carregue as instruções armazenadas. Destas duas abordagens podemos concluir que: a opção pelo armazenamento do programa em memória diminui o desempenho que pode ser alcançado pelo sistema, quando comparado ao que suas instruções fixas em *hardware* podem alcançar. Porém, o ganho é desempenho implica na perda de flexibilidade e uma vez fabricados não podem ter seu programa alterado. O mesmo não acontece com os processadores que tem o programa armazenado em memória. Estes processadores podem ser

utilizados para aplicações diversas e podem ter a função do programa modificada facilmente por meio de uma nova programação.

O segundo ponto, que define a arquitetura física da máquina de Von Neumann, é certamente o mais importante do modelo. A Figura 2.1 ilustra um exemplo da arquitetura de Von Neumann. Ela é composta por uma Unidade Lógica Aritmética (ULA) responsável pelo processamento das informações; uma memória que pode ser utilizada tanto para armazenamento de dados quanto para o armazenamento do programa; dispositivos de entrada e saída, responsáveis pela aquisição e envio de sinais, que realizam a interface com o usuário e o ambiente externo; e uma unidade de controle que gerencia as operações de processamento da ULA, bem como o armazenamento das informações na memória e as atividades dos dispositivos de entrada e saída.

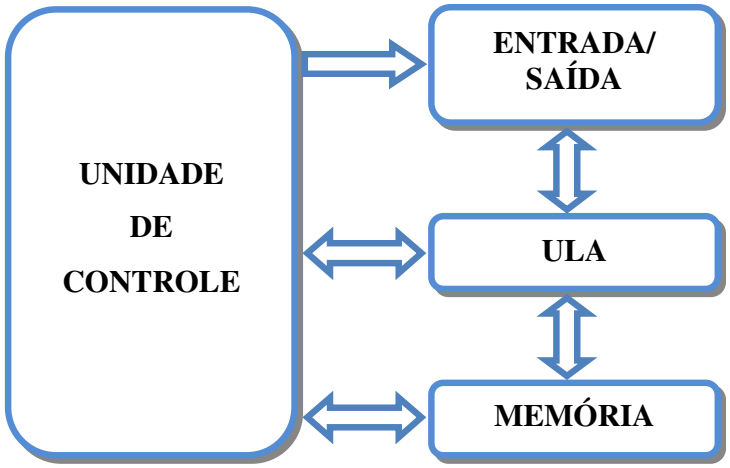

Figura 2.1: Arquitetura de John Von Neumann [Stallings, 2002]

O último ponto é destacado pela forma de operação do processador. Na verdade trata-se dos processadores que possuem as instruções do programa armazenadas em memória, as quais para serem executadas devem ser buscadas e decodificadas. Por isto, esta forma de execução das instruções recebe o nome de ciclo de busca-decodificação-execução, pois para cada instrução armazenada este mesmo processo deve ser realizado. Dependendo do tipo de programa, este ciclo pode ser otimizado por meio da implementação de um *pipeline*, mas para isso faz-se necessário que o programa não possua muitas instruções de desvio.

Hoje em dia, os sistemas embarcados que são desenvolvidos não são tão diferentes dos que eram desenvolvidos na década de 80. Nesta época, os sistemas embarcados eram comumente tidos como um conjunto de periféricos, os quais juntos eram possíveis de serem programados para execução de uma tarefa. Basicamente, estes sistemas eram compostos por um micro-controlador, uma memória e portas de entrada e saída analógica e digital. Por se tratarem de um sistema imutável em termos de *hardware*, o desempenho que era alcançado por estes era dependente do compilador que convertia o código, normalmente escrito em *assembly* para a linguagem de máquina que seria interpretada pela ULA. Não obstante, o algoritmo utilizado para implementar o programa era outro fator que influenciava sobre o desempenho que o sistema teria na execução de suas tarefas, posto que quanto menos eficiente o algoritmo fosse em termos de linhas de código maior seria a quantidade de instruções de máquinas executadas pela ULA.

Esta realidade na qual se encontravam os projetistas de sistemas proporcionou o surgimento das duas maiores vertentes de arquitetura de processadores que imperaram durante o final do século passado. A sessão 2.1.1 descreve as características e razões para o surgimento de tais arquiteturas.

2.1.1. Arquitetura RISC x Arquitetura CISC

Atualmente, o potencial de desempenho e a capacidade de armazenamento dos sistemas embarcados são bastante superiores quando comparados aos sistemas que eram desenvolvidos na década de 70. No início desta mesma década, vários fatores conspiravam para a criação de um sistema que atendesse as limitações que eram impostas. Estes fatores eram principalmente o custo com armazenamento de código e o valor da hora de trabalho para os desenvolvedores de *software*. Antes do advento das memórias RAMs (*Random Access Memory*), as memórias magnéticas existentes estavam a quem em termos de velocidade de acesso em comparação aos processadores. A memória era um ponto de extrema relevância, pois não só impactava diretamente sobre a velocidade do sistema, como também, o uso indiscriminado da mesma tornava o sistema mais caro. Isto nos leva

a conclusão que, para se ter um sistema a um custo reduzido, faz-se necessário um programa com código “enxuto”.

Aliado a estas questões, o custo com desenvolvimento de *software* tornava-se cada vez mais elevado, pois para se ter um programa compacto e eficiente era necessário que este fosse desenvolvido em linguagem de montagem (*assembly*). Esta linguagem era (e continua sendo) pouco amigável e inteligível para os desenvolvedores de códigos para sistemas embarcados e poucos eram os que verdadeiramente dominavam este estilo de programação. Desta forma, por se tratar de uma tarefa dispendiosa e que exigia a contratação de profissionais especializados, os bons códigos de programas para sistemas embarcados tornaram-se caros. A fim de contornar estes custos e desenvolver um sistema que tivesse um preço cada vez mais competitivo, e baseado na premissa estabelecida por Gordon E. Moore[2] em 1965, os projetistas pensaram numa forma de transferir a complexidade da programação para o *hardware*, haja vista, que este se tornava cada vez mais barato.

A filosofia de projetos CISC (*Complex Instruction Set Computer* – Computador com Conjunto Complexo de Instruções) tem como premissa a resolução destes problemas, ou seja, fornecer uma plataforma capaz de oferecer uma linguagem *assembly* que fosse mais amigável aos programadores. Isto implica principalmente em transferir a complexidade de uma série de funções para o *hardware*. Dentre as principais razões e ganhos que fomentaram o desenvolvimento de sistemas baseado em CISC destacam-se [Ditzel & Patterson, 1998], [Stallings, 2002]:

- Redução na complexidade de desenvolvimento de compiladores para linguagens de máquina;
- Redução no custo com desenvolvimento *softwares*;
- Redução do tamanho do código dos programas;

[2] A Lei de Moore serviu durante muito tempo como meta para os fabricantes de circuitos integrados (CIs) atenderem ao que fora estabelecido, fazendo com que estes investissem em pesquisas promovendo um grande avanço tecnológico na área de semicondutores no final do século XX.

- Redução da complexidade para desenvolvimento de *softwares*;
- Aumento da quantidade de instruções capazes de realizar operações mais complexas.

No início da década de 80, os projetistas de sistemas embarcados estavam divididos em dois grupos: os que defendiam a arquitetura RISC (*Reduced Instruction Set Computer* – Computador com Conjunto Reduzido de Instruções) e os que defendiam a arquitetura CISC. A arquitetura RISC teve seu primeiro processador desenvolvido pela IBM em 1975 e contradizia a forma de desenvolvimento de processadores daquele período. As arquiteturas CISC, por transferirem grande parte da complexidade do sistema para o *hardware*, acabavam por produzir um sistema demasiadamente grande, em termos de utilização de transistores, para os padrões da época. Esta grande necessidade de transistores fazia com que alguns dos processadores CISC não estivessem inseridos em um único CI (Circuito Integrado), aumentando a possibilidade de problemas no sistema. A despeito do que era tido como melhor prática para desenvolvimento das plataformas embarcadas, vários estudos estavam sendo feitos com o intuito de encontrar uma metodologia ótima para o desenvolvimento destes sistemas. Pesquisas apontavam que 25% das instruções contidas em uma plataforma CISC eram utilizadas em aproximadamente 95% do tempo de execução do processador. Além disso, uma quantidade de 30 instruções eram responsáveis por cerca de 99% das instruções que eram executadas pelo processador [Jamil, 1995].

Em 1980, Patterson e Ditzel propuseram a criação de um processador RISC com base nos estudos feitos sobre a utilização das instruções por parte dos processadores. Eles chegaram à conclusão que o desenvolvimento de uma arquitetura baseada nas instruções que eram mais utilizadas pelos processadores maximizaria a eficiência destes. As idéias que davam suporte ao desenvolvimento das arquiteturas RISC eram as seguintes [Patterson & Ditzel, 1980], [El-Aawar, 2006]:

- Pequeno conjunto de instruções;

- Instruções de tamanho único;
- Pequena quantidade de modos de endereçamento;
- A maior parte das instruções é executada em um único ciclo de relógio[3];
- Compilador otimizado e confiável.

Os temas que abordam estas duas filosofias de projetos envolvem diversos tipos de pesquisas, tais como aplicações de baixo consumo, metodologias para desenvolvimento de arquiteturas superescalares, aplicações de tempo real processamento digital de imagens, projeto de sistemas assíncronos entre outras [Charlot et. al. 1995], [Chang et. al. 1999], [Hu et. al. 2006] e [Lee et. al. 2002]. Todos estes trabalhos possuem um ponto em comum, apesar estarem dirigidos a áreas de estudos distintas. O crescimento tecnológico e científico, no que diz respeito às formas de construção e qualidade dos produtos que são fabricados atualmente, nos permitiu quebrar paradigmas e mudar a forma de pensamento para concepção dos sistemas embarcados. A alta densidade dos circuitos de hoje e a reconfiguração dinâmica são duas características que, entre as décadas de 70 e 80, limitavam bastante o escopo dos projetos e suas funcionalidades. A sessão 2.1.2 abordará aspectos sobre a atual situação e tecnologia disponíveis para a concepção de sistemas embarcados

2.1.2. FPGAs e Dispositivos Reconfiguráveis

Muito embora as arquiteturas e filosofia de desenvolvimento de sistemas embarcados sejam objetos de estudo de grande importância, a forma com que projetos de sistemas embarcados eram desenvolvidos pressupunha uma grande limitação e dependência do sistema pré-fabricado. Neste caso, a arquitetura do *core* do processador escolhido, muitas vezes, não era especificamente projetada para a aplicação ao qual era proposto, fossem estes processadores RISC ou CISC. Isto

[3] Neste caso, relógio deve ser entendido como o dispositivo responsável pela imposição do ritmo de processamento do sistema e sincronização das operações/tarefas

significava que o projetista estava limitado a um conjunto genérico de instruções de máquina que o processador era capaz de realizar e a uma limitação física de recursos de *hardware*, tais como memória, tamanho das palavras a serem processadas e quantidade de portas de entrada e saída [Stallings, 2002]. Os dispositivos reconfiguráveis estão inseridos dentro uma classe de sistemas digitais que, especialmente no Brasil, obteve grandes incentivos financeiros por parte do governo federal e empresas da área privada nos últimos anos. Um dos exemplos que deste investimento é o Programa Brasil-IP (Intelectual Property) que tem por objetivo formar um corpo expressivo e qualificado de profissionais de sistemas embarcados (sistemas digitais, analógicos e rádio frequência). Apesar de o país estar investindo a pouco tempo neste tipo de tecnologia, o primeiro dispositivo reconfigurável (*Programmable Read-Only Memory* - PROM) apresentado pela comunidade científica já remonta quase 4 décadas . Como forma de contextualizar o leitor sobre a plataforma de implementação a qual esta proposta se destina, façamos uma breve revisão histórica a cerca dos *Dispositivos Reconfiguráveis*.

Os Dispositivos Lógicos Programáveis (*Programmable Logic Device* - PLD) surgiram no início da década de 70 e tinham o propósito de fornecer certa flexibilidade de configuração em nível de *hardware*. A *Field-Programmable Logic Array* (FPLA ou apenas PLA, ou ainda PAL) também foi desenvolvido no início da década de 70 e era constituído por um plano fixo de portas ANDs seguido uma coluna de portas ORs de diversos tamanhos. Estes primeiros dispositivos tinham o inconveniente de permitir apenas a aplicação de lógicas combinacionais, mas logo adveio a idéia de interligar-lhes às suas saídas uma bateria de flip-flops [Brown & Rose, 1996]. A Figura 2.2 ilustra um exemplo de da arquitetura da PAL.

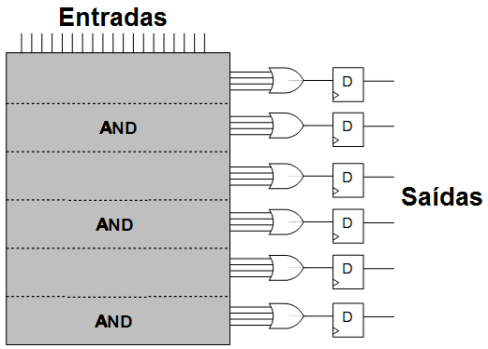

Figura 2.2: Arquitetura de uma PAL [Brown & Rose, 1996]

Claramente, as funções que eram possíveis de serem implementadas no PAL eram somas de produtos. A empresa responsável pela criação dos primeiros PALs foi a *Monolithic Memories* e, já no início dos anos 80, foram acrescentados novos blocos de circuitos lógicos aos PALs, o que lhes concebera maior poder de processamento. Estes novos blocos foram chamados de Macrocélulas e eram dotados de *flip-flops*, multiplexadores e portas lógicas.

Os CPLDs (Complex PLD) surgiram posteriormente aos PALs e destacam-se como sendo o dispositivo que promoveu a agregação de vários PLDs em um único *chip*. Esta agregação era promovida por meio de uma rede de interligações físicas e um mecanismo de roteamento entre os diversos blocos de PLDs. Além disso, a qualidade dos dispositivos semicondutores que eram utilizados na fabricação dos CPLDs era superior à dos antigos PALs. Outra importante característica que foi adicionada aos CLPDs foi o padrão de testes e varredura de circuitos, o *Joint Test Action Group* (JTAG), a fim de descobrir pontos de falhas. O processo natural de evolução tecnológica fez com que os sistemas embarcados se tornassem cada vez mais compactos, complexos e densos (maior número de transistores por unidade de área ocupada). A tecnologia atual permite que sistemas sejam produzidos em uma densidade de área de um transistor para cada $10^{-9}m^2$. Esta tecnologia é chamada de nano-tecnologia e está sendo fortemente estudada para o desenvolvimento de soluções que podem ser aplicadas nas mais diversas áreas tais como: industrial têxtil, engenharia militar e principalmente na área bio-médica, a bio-tecnologia.

A forma conceitual que caracteriza a arquitetura estrutural de uma *Field-Programmable Gate Arrays* (FPGA) está centrada na alta densidade circuitos, milhares de pinos de entrada e saída (o que lhes confere uma grande vantagem para criação de barramentos de diversos tamanhos) e sua capacidade de reconfiguração. Apesar de serem dispositivos semelhantes aos CPLDs, existem algumas diferenças que distinguem as FPGAs destes últimos. A título de exemplificação, podemos citar como diferença entre os dois dispositivos a quantidade de transistores existente nas FPGAs, que é maior, o processo de fabricação e a tecnologia que empregada. Neste sentido, ela pode ser entendida como uma imensa matriz de blocos lógicos configuráveis (*Configurable Logic Block* - CLB) interconectados ente si por diversas

malhas de conexão e dispositivos de entrada e saída que possibilitam a transferência de dados entre estes blocos. Além disso, o controle da configuração destes CLBs é realizado através de um conjunto de células de memória que estão localizados abaixo da estrutura dos CLBs. A Figura 2.3 ilustra um exemplo desta configuração.

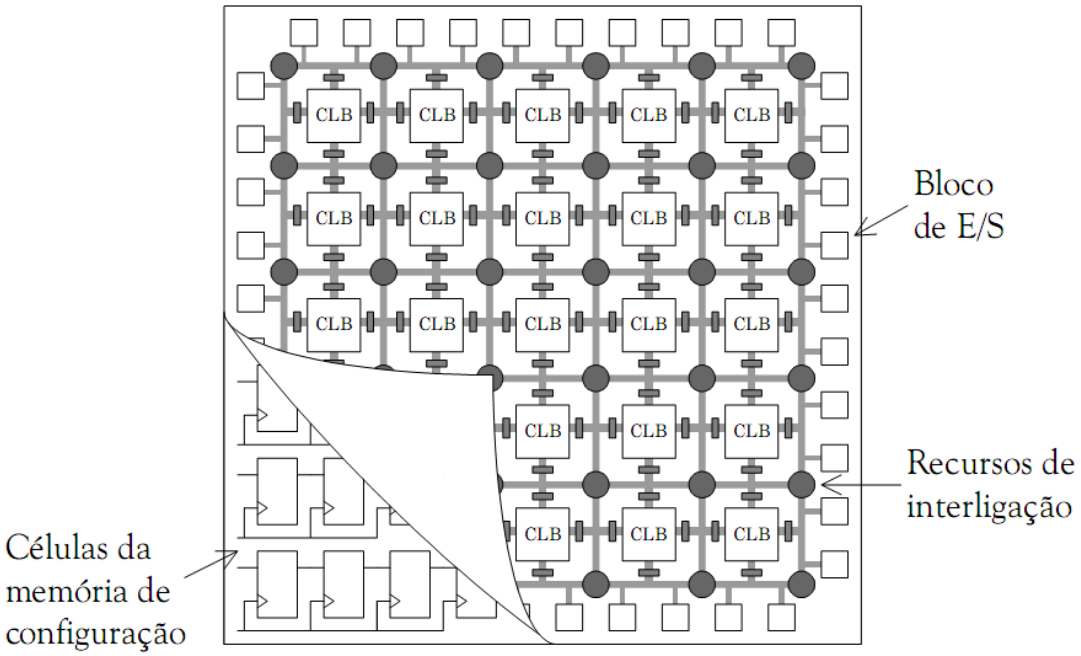

Figura 2.3: Arquitetura de uma FPGA [Gericota, 2003]

Diferentemente dos blocos que são interconectados pelo CPLD, o CLB não se baseia na utilização de soma de produtos, mas na utilização de *Lookup-Table* (LUT), ou seja, implementação de lógicas baseada em memória. As FPGAs constituem o principal grupo de dispositivos reconfiguráveis e nos últimos tempos adquiriu uma característica que proporcionou aos projetistas uma nova forma de pensar e desenvolver sistemas embarcados que é a reconfiguração dinâmica. Esta característica de algumas FPGAs permite ao *hardware* a capacidade de ser remodelado em tempo de execução. Diante disto, o desenvolvimento de sistemas com características de reconfiguração dinâmica possibilita a agregação de funcionalidades que podem tornar o sistema mais seguro através da implementação de técnicas de tolerância à falhas [Matos & White, 1998] e [Yu & Koren, 1994]. A despeito desta característica, as FPGAs são excelentes para prototipação de

projetos de sistemas embarcados devido à sua facilidade de reprogramação e redução de custos com erros de projeto não percebidos na fase de concepção, haja vista a característica de reconfiguração que lhe é assegurada. Entretanto, os resultados de desempenho, ocupação de área ou potência consumida obtidos em uma FPGA não devem ser comparados ao de um Circuito Integrado de Aplicação Específica (*Application-Specific Integrated Circuit* - ASIC), pois a maioria dos circuitos que são desenvolvidos em uma FPGA não está organizado em um arranjo ótimo para tal.

As FPGAs podem ser classificadas segundo o tipo de reconfiguração que lhe é assegurada dentro de 4 grupos segundo Ribeiro [Ribeiro, 2002]. As FPGAs baseadas na tecnologia *antifuse* são somente programáveis, ou seja, não são passíveis de reconfiguração. As EEPROMs (*Electrically-Erasable Programmable Read-Only Memory*) estão classificadas dentro do grupo reconfiguráveis, no entanto, o ato da reconfiguração implica na alteração de todas as células que compõem o dispositivo neste processo. No terceiro grupo estão inseridas as FPGAs parcialmente reconfiguráveis as quais são capazes de reter uma parte inativa de sua configuração enquanto outra sofre a reconfiguração. Isto resulta no ganho de tempo, haja vista que somente o circuito de interesse sofre a reconfiguração. O último grupo é designado de dinamicamente reconfigurável. Este grupo distingue-se do anterior pelo fato de permitir a reconfiguração seletiva de parte de seus circuitos em tempo de execução, ou seja, a reconfiguração ocorre com a FPGA em seu modo ativo. A Figura 2.4 ilustra a organização destes grupos, para outras informações [Skliarova & Ferrari, 2003].

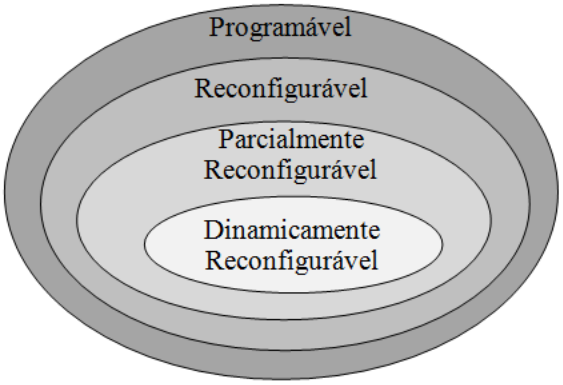

Figura 2.4: Classificação das FPGAs segundo a reconfigurabilidade [Ribeiro, 2002]

2.2. Tolerância a Falhas e Sistemas Redundantes

A revolução digital ocorrida no final do século passado promoveu o surgimento de sistemas que vieram a automatizar diversos processos, tais como: transações bancárias, controle de tráfego (aéreo, terrestre ou marítimo), plataformas ou refinarias petroquímicas e ambientes hospitalares. Na maioria dos casos, estes processos envolvem um fator de criticidade o qual necessita de especial atenção a fim de evitar a ocorrência de danos à saúde (física ou mental) dos envolvidos no processo, ao meio ambiente ou aos recursos financeiros. As tecnologias e pesquisas envolvendo soluções para ambientes hospitalares têm sido fortemente fomentadas no meio acadêmico [Soares, 2008], [Iantovics, 2007], [Zhang et. al. 2006] e [Bartocci et. al. 2007]. Este conjunto de fatores acaba por demandar do sistema um alto grau de confiança e disponibilidade, os quais podem ser alcançados através do emprego de técnicas de tolerância a falhas e, em alguns casos, *Redundância em Hardware*.

2.2.1. Tolerância a Falhas

O conceito de tolerância a falha foi primeiramente cunhado em 1967, por Avizienis, o qual fora motivado pelo projeto da *National Aeronautics and Space Administration* (NASA) que pretendia construir uma nave espacial não tripulada para enviar ao espaço [Avizienis, 1997]. O principal desafio deste projeto consistia no desenvolvimento de um sistema que fosse confiável e capaz de atuar de forma automática por um período de dez anos. Desde então, as tecnologias e os métodos para desenvolvimento de sistemas embarcados evoluíram bastante. A quantidade de transistores que podem ser implantados dentro de um chip (*System-on-Chip* - SoC), atualmente, é da ordem de dezenas de milhões. Isto acaba por permitir a construção de um sistema com um maior poder computacional, pois mais funcionalidades podem ser agregadas ao SoC.

As falhas são acontecimentos aos quais os sistemas estão sujeitos e são impossíveis de serem evitadas, toda via, os efeitos que elas podem causar sobre o sistema computacional podem ser tratados de forma a gerarem danos mínimos ou nenhum. Para tanto, existem diversas técnicas de tolerância à falhas já conhecidas no meio acadêmico que podem ser utilizadas em sistemas permitindo que estes

possam atuar de forma limitada, em alguns casos, ou até mesmo sendo intertravados para evitar a ocorrência de um dano maior. Na industrial de mineração, petroquímica ou siderúrgica, por exemplo, existem diversos processos que podem ser considerados extremamente críticos e que envolvem vidas humanas. A ocorrência de falha em um processo crítico destas indústrias pode causar a perde de vidas humanas, ou danos irreparáveis a algum equipamento, se não forem corretamente tratadas. Dependendo do tipo de falha, a parada do processo, mesmo que signifique a diminuição da produção, pode ser a melhor opção, caso seja constatado a ocorrência de uma falha que avarie o sistema e comprometa a segurança do processo. Assim como existem processos automatizados na indústria, de uma forma geral, a área biomédica também é dotada de alguns equipamentos especializados para monitoração de pacientes [Kida et. al. 2008], [Hong et. al. 2008] e [Winey & Yan, 2006]. Para um melhor entendimento dos conceitos de tolerância a falha passemos a definição de alguns conceitos clássicos.

2.2.2. Dependabilidade

A dependabilidade é uma propriedade do sistema que engloba um conjunto de propriedades que servem para definir a qualidade do serviço que é oferecida pelo sistema [Lapri, 1985]. Dentre estas propriedades Pradhan, em [Pradhan, 1996], destacam-se as seguintes: confiabilidade (*reliability*), disponibilidade (*desponibility*), testabilidade (*testability*), mantenabilidade (*mantenability*), segurança de funcionamento (*safety*), segurança da informação (*security*) e desempenho (*performability*). A seguir são apresentados os conceitos de cada uma destas propriedades.

- **Confiabilidade**: É a capacidade que o sistema tem de executar suas tarefas atendendo as especificações predefinidas do sistema, por um determinado período de tempo, sendo necessário estar operável no início da execução processo.

- **Disponibilidade**: É dada pela probabilidade que o sistema tem de estar disponível para execução de uma dada tarefa quando solicitado.

- **Testabilidade**: É dada pela possibilidade da realização de teste sobre o sistema a fim de verificar a consistência do mesmo. Esta propriedade será melhor tanto quanto for mais fácil a realização dos testes.

- **Mantenabilidade**: Esta propriedade esta relacionada com a capacidade de recuperação do sistema após a entrada deste em um estado de erro. Esta propriedade será melhor tanto quanto menor for o tempo para reparo e recuperação do sistema após sua parada. A mantenabilidade do sistema também está relacionada à probabilidade do sistema ser reparado dentro de um determinado período de tempo pré-estipulado.

- **Segurança de funcionamento**: Esta propriedade esta relacionada à probabilidade que o sistema tem de executar suas tarefas, ou dirigir o sistema para um estado de segurança, quando uma falha que comprometa a segurança for constatada. Neste sentido, ou o sistema executa suas tarefas sem causar danos aos usuários e outros sistemas ao qual ele esteja ligado, ou interrompe suas operações.

- **Segurança da informação**: Esta propriedade relacionada à confidencialidade, integridade, autenticidade e proteção contra invasões no sistema que vem a comprometer os dados armazenados.

- **Desempenho**: Esta propriedade está relacionada à capacidade de o sistema continuar operando, mesmo que de forma degradada, quando uma falha ocorre.

Diante das propriedades inerentes às características da dependabilidade, podemos observar que a tolerância a falha mostra-se como uma solução para a resolução de alguns desafios que a norteiam. As subsessões seguintes apresentaram outros conceitos relacionados à tolerância a falhas, bem como, técnicas utilizadas para sua implementação.

2.2.3. Conceitos Clássicos: Falha, Erro e Defeito

O correto funcionamento de um sistema está relacionado ao atendimento de suas especificações (limites máximos e mínimos, tempo, quantidade de dados processados, etc.). O não atendimento a estas especificações podem decorrer de uma má especificação ou de uma falha. O conceito relacionado à definição do que vem a ser *falha*, *erro* e *defeito* é de fundamental importância para a compreensão dos assuntos abordados nesta proposta de qualificação. As definições aqui apresentadas seguem o pensamento adotado por Weber [Weber et. al. 1990], Laprie [Lapri, 1985] e Anderson [Anderson & Lee, 1981]. A *falha* é definida como o evento que faz o sistema desviar-se de seu curso normal de execução e está necessariamente associada ao universo físico. Este acontecimento pode ser originário de um fator lógico ou físico. Uma falha decorrente de um fator lógico, em alguns casos, está relacionada a uma parte do código que não foi corretamente elaborada ou a um valor de entrada imprevisto, como no caso de entrada de um valor do tipo *real* quando é esperado um valor do tipo *inteiro*. As falhas decorrentes de fatores físicos normalmente estão relacionadas a desgastes do equipamento ou problemas de interconexão entre estes. Embora a falha seja o acontecimento que faz com que o sistema desvie-se de sua especificação, a ocorrência de uma falha, em alguns casos, não significa que o sistema irá atuar de forma incorreta. Por exemplo, se algum dispositivo de sistema deixa de funcionar, mas ele não é utilizado, a percepção desta falha pode nunca acontecer.

O *erro*, em um sistema, nada mais é que a manifestação da falha, ou seja, quando uma falha faz com que o sistema atue de forma não conforme com sua especificação, diz-se que o sistema está em um estado de erro [Avizienis et. al. 2004]. Por exemplo, uma falha em um sensor de temperatura de uma caldeira poderia induzir o sistema a um estado errôneo quanto ao verdadeiro valor de temperatura. Sendo assim, a falha no sensor de temperatura provocou um erro de medição de temperatura.

O *defeito* pode ser entendido como a manifestação do *erro*, assim como o *erro* pode ser entendido como a manifestação da *falha*. Quando um sistema ou equipamento está com defeito, ou em um estado defeituoso, as especificações para

o qual este fora projetado não serão atendidas. Por exemplo, dado uma rede de sensores/atuadores inteligentes que se comunicam e efetuam a monitoração e aplicação de medicamentos em um leito hospitalar [Valentim et. al. 2008]. A perda da comunicação devido a falha na estrutura da rede pode fazer com que o sistema não receba informações sobre algum dado clínico do paciente, o que indica um *defeito* no sistema de comunicação ou no equipamento que deveria fazer a medição.

2.2.4. Falha e Sistemas Computacionais

Um sistema é dito ser tolerante a falha quando possui a capacidade de executar corretamente suas tarefas independentemente da ocorrência de falhas [Johnson, 1984]. Mesmo um sistema que tenha sido aparentemente desenvolvido corretamente está sujeito a falhas decorrentes do desgaste dos dispositivos ou erros de projetos que impliquem em uma lógica falha. Existem dois tipos de falhas que podem acontecer em um sistema: as falhas naturais causadas por um desgaste, de origem interna ou externa, de um componente; e as falhas decorrentes de uma ação humana acidental ou intencional [Avizienis, 1997]. A ocorrência de uma destas falhas não implica que outro tipo de falha não possa vir a acontecer. Como exemplo de uma situação em que estes dois tipos de falhas podem ocorrer é quando um operador configura os parâmetros de uma planta para executar uma tarefa de forma sobrecarregada. Esta operação sistemática fará com que a planta, inevitavelmente, entre em avaria devido ao desgaste de suas peças. Neste exemplo em particular, uma falha decorrente de uma ação humana gerou uma falta natural.

A falha é o fator originário que implica no surgimento de um *erro*. Quando um *erro* acontece, o sistema pode ser conduzido a um estado inconsistente/incoerente que caracteriza a existência de um *defeito*. A falha é o fator que irá fazer com que o sistema não atenda às especificações de seus requisitos. No entanto, os sistemas tolerantes a falhas devem ser capazes de conduzir o sistema a um estado seguro quando uma falta (erro) é detectada, ou permitir que o sistema possa atuar de forma degradada, mas atendendo aos requisitos mínimos do mesmo. Diante deste contexto, existem algumas técnicas que visam dar um tratamento correto à falha, são elas: estruturas redundantes para mascarar componentes que sofreram alguma

falha; códigos de controle de erros e duplicação ou triplicação com mecanismo de votação para descobrir ou corrigir erros de informação; técnicas de diagnóstico para localizar componentes falhos; e técnicas de conversões automáticas para substituir um subsistema que sofreu uma falha [Avizienis et. al. 1987]. Em 1984, Johnson definiu as três principais técnicas para tentar melhorar ou manter a operação normal de um sistema: Prevenção de Falha (*Fault Avoidance*), Mascaramento de Falha (*Fault Masking*) e Tolerância a Falha (*Fault Tolerance*) [Johnson, 1984].

A Prevenção de Falha visa atacar a falha antes mesmo de seu surgimento e pode ser conseguida através de boas práticas de gerenciamento de projetos e testes do sistema para detecção de alguma falha de projeto não percebida em sua fase de detalhamento. Isto tem por objetivo evitar que o sistema seja lançado para o mercado com algum "*bug*". Neste sentido, testes exaustivos da aplicação podem ajudar a descobrir estes "*bugs*", entretanto, o tempo gasto na fase de testes irá aumentar o "*time to market*" (quantidade de tempo desde a concepção até o lançamento do produto no mercado).

O Mascaramento de Falha é aplicado em sistema de tempo real e tem por objetivo evitar que a falha seja propagada para outros componentes do sistema provocando a ocorrência de um erro, ou levando o sistema para um estado de erro. Uma das formas de se implementar esta técnica é por meio da implementação de votadores.

A Tolerância a Falha, ao contrário das demais, não tem por objetivo eliminar o surgimento da falha ou a sua propagação, mas sim conduzir o sistema para um estado que seja aceitável, em termos de execução, e seguro. Existem quatro princípios sobre os quais está fundamentado a utilização da Tolerância a Falha. São eles: a detecção, a localização, o confinamento e recuperação. A Figura 2.5 mostra um fluxograma onde as técnicas descritas por Johnson podem ser utilizadas para aumentar a garantia de funcionamento adequado do sistema e onde cada uma é empregada [Johnson, 1984]:

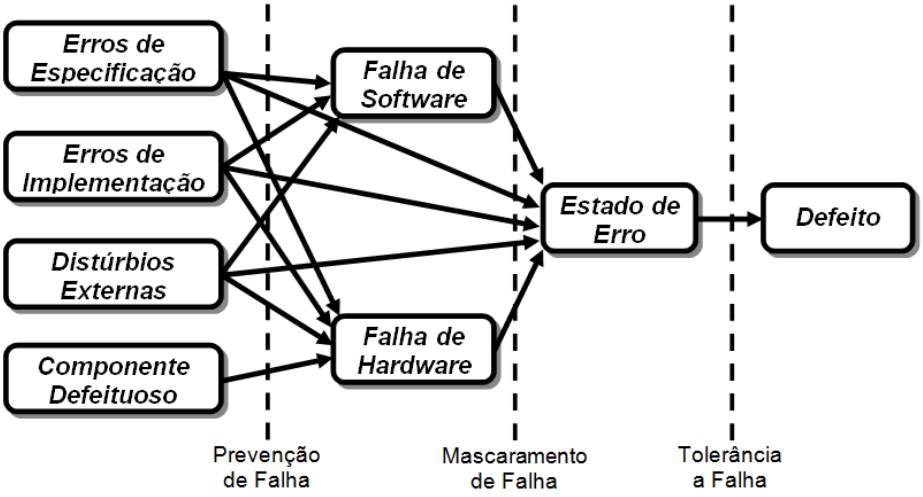

Figura 2.5: Fluxograma de Causa-Efeito de Falhas [Johnson, 1984]

2.2.5. Sistemas Redundantes

As técnicas de redundância em *hardware* estão, necessariamente, associadas ao aumento de custos financeiros decorrentes da maior quantidade de *hardware* que será empregada no sistema. Entretanto, as técnicas de redundância podem ocorrer em diversos níveis e/ou escopos do projeto [Johnson, 1984], dependendo do tipo do negócio e necessidade do cliente. Vejamos a seguir alguns tipos de redundância que comumente são implementadas em sistemas.

2.2.6. Redundância Temporal

Este tipo de técnica de redundância é implementada em processos onde o poder de processamento do sistema é suficientemente elevado, ao ponto de se poder processar uma mesma informação mais de uma vez, sem que o processo venha a sofrer algum prejuízo. Normalmente é utilizada em aplicações de tempo não crítico. Estas técnicas podem ser empregadas para a detecção de falhas transientes, ou seja, falhas que ocorrem de tempo em tempos, não necessariamente igualmente espaçados. Também é possível aplicá-la a fim de detectar uma falha permanente. Para este tipo de falha a detecção requer a aplicação de funções de codificadores e decodificadores as quais sejam inversíveis [Abd-El-Barr, 2007].

Para os casos de Redundância temporal, qualquer que seja o tipo de falha que se queira detectar, permanente ou transiente, faz-se necessário o uso de um

componente que seja capaz de armazenar o valor do dado em um tempo t e um comparador. Estes dois componentes serão utilizados para que os dados processados em um tempo $t+x$ e os processados em um tempo t possam ser comparados. Desta forma, a não equivalência entre os valores constatará a existência da falha.

2.2.7. Redundância de *Software*

A aplicação de técnicas de redundância de *software* não acontece simplesmente copiando dois programas idênticos e fazendo com que estes executem em paralelo. A própria literatura comenta sobre a grande dificuldade e a quantidade de erros que são ocasionados pela utilização deste tipo de redundância [Koren & Krishna, 2007]. Existem diversos tipos de técnicas de redundância que podem ser utilizadas com o auxílio de *software*. Por este não ser o objetivo principal desta dissertação, não nos ateremos a este tipo de redundância e maiores detalhes podem ser consultados em [Abd-El-Barr, 2007], [Koren & Krishna, 2007], [Pradhan, 1996] e [Jalote, 1994]. A seguir, daremos uma breve explicação sobre algumas das técnicas de redundância de *software* que podem ser aplicadas em um sistema computacional, as quais podem ser consultadas em [Abd-El-Barr, 2007].

Programação N-versões: Neste tipo de abordagem, o mesmo escopo de um sistema é passado para diversos grupos de desenvolvedores de *software*, os quais ficam encarregados de elaborar o programa. Após a finalização do programa, cria-se um mecanismo de votação que irá decidir sobre a saída do sistema com base nos valores que foram gerados por cada um dos programas. Normalmente são desenvolvidas 3 aplicações, as quais executam a mesma tarefa em paralelo. Esta técnica pode ser aplicada em diversas fases do desenvolvimento do *software* (especificação, implementação, teste), entretanto vários problemas estão associados a este tipo, tais como: identificação do erro por meio de correlação, sincronização entre os *software*s e aumento significativo da complexidade e custo do sistema. Não obstante, não existe um método formal para validar a eficácia deste tipo de técnica.

Blocos de Recuperação: Está técnica é bastante semelhante à Programação N-versões, mas com uma abordagem diferente. Tal qual para a técnica anterior, esta

também faz uso de várias versões de programas desenvolvidas por grupos distintos, no entanto, ao invés de utilizar um votador para decidir sobre o resultado processado é feito um teste de aceitação sobre cada uma dos programas. Os programas estão organizados de forma cascateada e para cada saída que não passar no teste de aceitação é verificada a saída do programa seguinte.

Precondições, Pós-condições e Asserções: Trata-se de testes de aceitação que são realizados sobre métodos, funções ou sub-rotinas de *software* a fim de realizar, por exemplo, um tratamento de exceção que ocorre quando se tenta realizar uma divisão por zero. Estes testes, que podem ocorrer antes, durante ou após a invocação de um método e tem por objetivo evitar a propagação do erro ou o "travamento" do programa.

As técnicas de tolerância a falhas aplicadas em *software*s não estão limitadas a estas breves descrições. Existem outras abordagens que exploram pontos de falhas diferentes dos aqui mencionados, não só para detecção como também para correção.

2.2.8. Redundância da Informação

Existem sistemas onde o mais importante é que a informação associada ao processo não seja perdida e, para tal situação, o emprego de redundância de dados armazenados (*backup*) em discos rígidos diferentes pode ser visto como uma boa solução para o problema. Toda via, torna-se mais proveitoso e seguro ter estes dois, ou mais, discos localizados em ambientes diferentes. Isto visa resguardar um destes discos rígidos no caso da ocorrência de um incêndio no prédio em que estes se encontrem, por exemplo, causando a destruição de ambos os dispositivos de armazenamento. Isto irá diminuir as chances de perda da informação, além de evitar a ocorrência de um evento que danifique os dois, ou mais, discos rígidos.

Neste caso, os custos envolvidos não estão relacionados somente à duplicação dos dados, mas também ao aluguel/compra/construção de outro local para armazenamento deste disco. Para tal situação, existem empresas especializadas que tratam do armazenamento e proteção (segurança) destas informações, o que evidencia mais um custo para o sistema. Não obstante, há

também a perda de tempo inerente à realização da duplicação da base de dados e, em algumas circunstâncias, o transporte, que também consome recursos. Esta é uma técnica que é utilizada com fins a prevenir a integridade dos dados, ou seja, trata-se de um mascaramento para o caso de ocorrência de alguma falha em um ou outro disco. No entanto, diferentemente das aplicações de armazenamento, existem outras formas de se aplicar a redundância da informação como meio para detecção de falhas. Nestes casos, costuma-se transmitir informações reduntantes em uma mesma mensagem para que o receptor possa comparar os campos e validar a qualidade da informação recebida. Exemplos clássicos de onde estas técnicas são implementadas podem ser encontrados nos campos de verificação da mensagem, utilizados em alguns protocolos, tais como bits de paridade e *checksum*. Estes campos de detecção de erros são muito importantes para verificação da mensagem que é recebida, no entanto, além destes ou, ao invés destes, existem também alguns campos que são utilizados para detecção e correção de erros, a saber, *Error Detection Code* (EDC) que além de detectar o erro realiza a correção deste.

2.2.9. Redundância de *Hardware*

Dentre todas as técnicas de redundância utilizadas para sistemas tolerantes a falhas, as técnicas de redundância em *hardware* são as mais pesquisadas no meio científico [Koren & Krishna, 2007]. O emprego deste método, entretanto, deve ser aplicado de acordo com o bom senso do projetista e a viabilidade financeira para qual o sistema é destinado. Neste cenário, existem três tipos de técnicas de redundância de *hardware* que são conceitualmente empregadas: redundância estática (ou passiva), redundância dinâmica (ou ativa) e redundância híbrida. A seguir são apresentados os conceitos dos tipos de técnicas de redundância em hardware segundo Koren e Krishna [Koren & Krishna, 2007].

Redundância Estática: O princípio básico desta técnica consiste em ocultar (mascarar) a existência da falha em um determinado nível do sistema de tal forma que sua ocorrência não seja identificada por outras partes do mesmo. Exemplos clássicos deste tipo de redundância são os sistemas *N-Modular Redundancy* (NMR) e o *Triple-Modular Redundancy* (TMR), o qual trata-se de uma particularidade do

NMR. Um sistema TMR é caracterizado pela existência de três módulos de processamento de *hardware* idênticos que devem receber e processar a mesma gama de dados em paralelo. A saída destes módulos está ligada a um módulo votador que irá decidir sobre a corretude dos valores recebidos baseado no valor médio ou pela maioria. Além disso, a correta especificação deste sistema irá depender da expectativa de falha dos módulos de processamento. Considerando que, em um determinado sistema, *n* módulos venham a falhar é fácil notar que para este será necessária a utilização de *2n+1* módulos [Abd-El-Barr, 2007]. Esta constatação pode ser feita, pois como o sistema baseia-se em um votador, é necessário que a maioria dos módulos esteja em bom estado para que o resultado da escolha seja correto. A Figura 2.6 demonstra um exemplo em diagrama de fluxo de um sistema TMR.

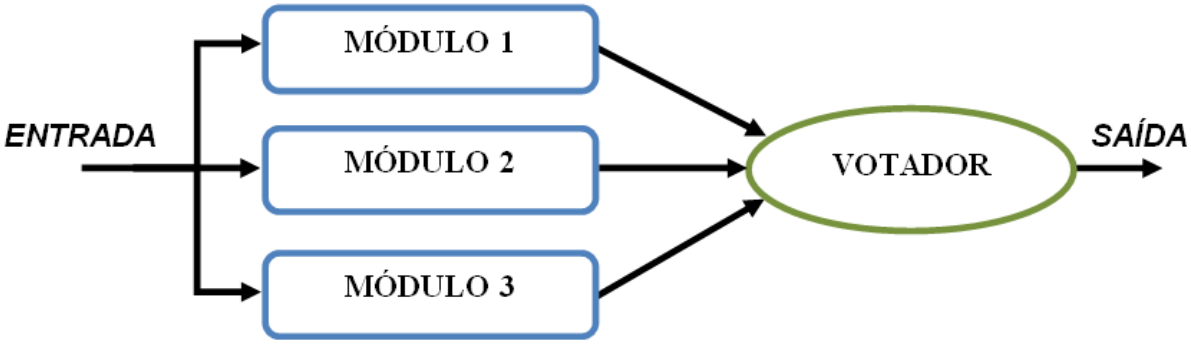

Figura 2.6: Modelo de Redundância TMR [Abd-El-Barr, 2007]

Observando este modelo podemos perceber que o módulo votador consiste no ponto crítico de falha, pois independentemente da quantidade de falhas que venham a acontecer nos módulos de processamento, se o módulo votador não estiver funcionando corretamente o sistema poderá atuar de forma errada. Como forma de minimizar esta deficiência, um esquema de redundância de votadores e armazenamento em memória podem ser utilizados [Koren & Krishna, 2007]. A Figura 2.7 demonstra um exemplo deste modelo que aumenta a confiabilidade do sistema. Vale salientar que este tipo de sistema não é capaz de detectar qual(is) dos módulos do sistema sofreu(ram) uma falha que tenha ocasionado um erro. Para isto, seria necessária a implantação de um detector de falhas para identificar o(s) módulo(s) em

desacordo(s). Pesquisas mostram que aplicações que requeiram a disponibilidade do sistema por um longo período, a utilização de um sistema NMR pode apresentar um desempenho pior que o de um sistema não redundante em termos de confiança.

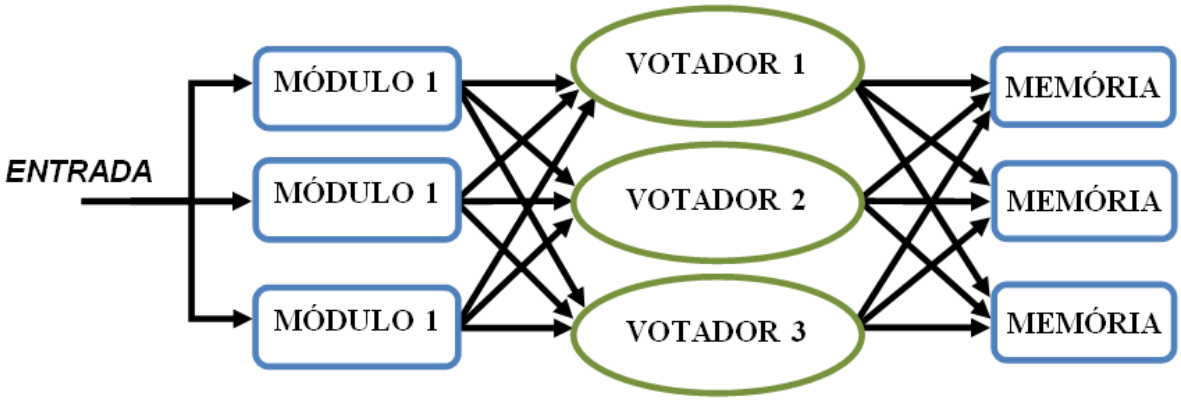

Figura 2.7: Modelo TMR com Redundância de Votadores [Koren & Krishna, 2007]

O Gráfico 2.1 demonstra o comportamento destes sistemas ao longo do tempo baseado em seu grau de confiança. Claramente podemos notar que, com o passar do tempo, a confiança de um sistema com redundância múltipla torna-se inferior a de um sistema que não emprega redundância. Parte desse declínio na confiança do sistema deve-se a complexidade envolvida para recuperação, quando da ocorrência de falha, em um sistema distribuído.

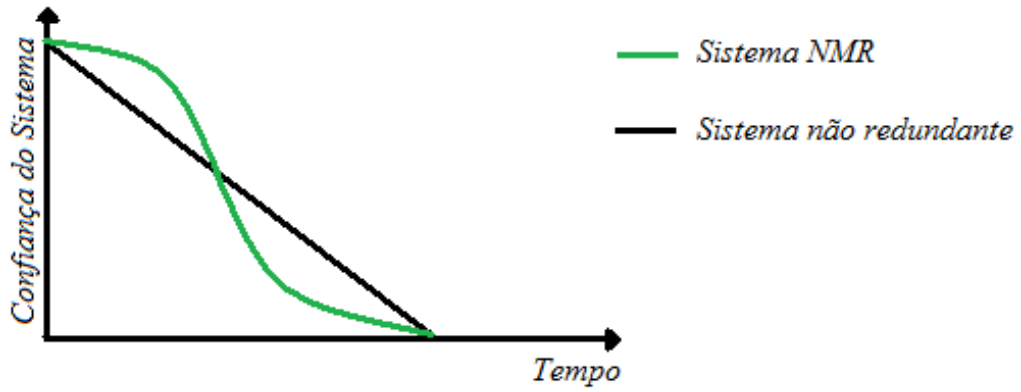

Gráfico 2.1: Comparação entre sistemas redundantes e não redundantes [Koren & Krishna, 2007]

Redundância Dinâmica: Este tipo de técnica deve ser aplicado em sistemas que não exijam uma resposta imediata da aplicação quando da ocorrência de um

erro, ou seja, é admissível que o sistema permaneça um período de tempo tolerável em um estado de falha. O fundamento na qual se baseia está técnica pressupõe a substituição ou remoção do componente que sofreu a falha (módulo ativo) por uma réplica deste (módulo passivo). Para tanto, é necessário que tais módulos sejam dotados de uma estrutura capaz de perceber a ocorrência da falha e, a partir desta detecção, recuperar o sistema. Esta estrutura é chamada de módulo detector/recuperador de falha e, pode ser visto como um gerenciador que bloqueia a passagem dos valores gerados pelos componentes com erro. Quando um erro é detectado, o módulo ativo em erro é bloqueado e o módulo seguinte passa a ser o responsável pela geração da informação e continuidade de operação do sistema. A Figura 2.8 ilustra um arranjo simples da implementação da técnica de redundância dinâmica.

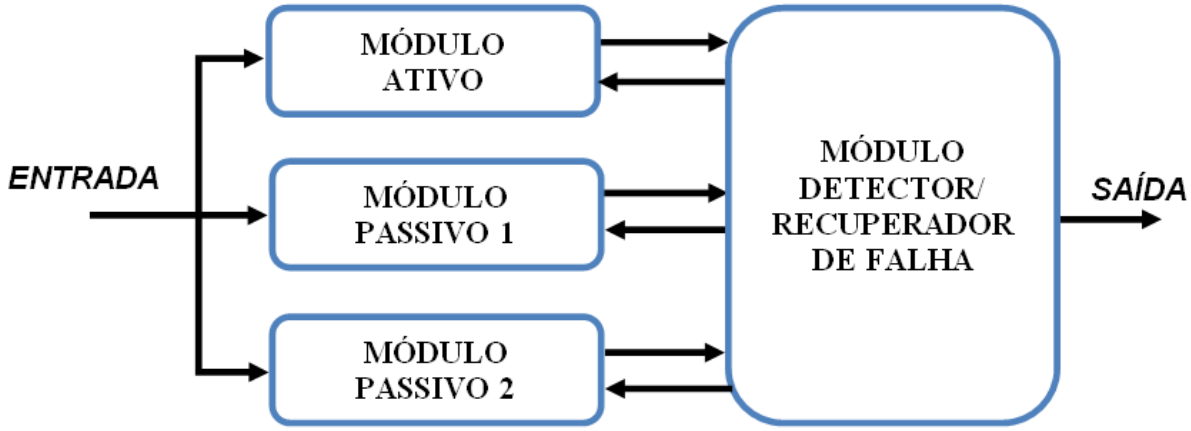

Figura 2.8: Modelo de Redundância Dinâmica [Abd-El-Barr, 2007]

A forma de operação dos componentes replicados determina o tipo de redundância dinâmica:

- ***Hot Standby*:** Nesta configuração, ambos os módulos estão operando de forma paralela e, no momento em que ocorre um erro no módulo que está em operação, o módulo seguinte, por estar operando em paralelo, encontra-se apto para enviar o valor processado.

- ***Cold Standby*:** Neste tipo de configuração, apenas um módulo processa a informação por vez. O módulo passivo é acionado somente quando um erro é detectado no módulo ativo. Este tipo de redundância requer um maior tempo de espera para recuperação do sistema, entretanto, o tempo de vida útil de forma geral é aumentado, posto que o módulo passivo só começa a atuar quando o módulo ativo entra um estado de erro. Isto permite que os módulos sofram um menor desgaste decorrente de sua não utilização.

Redundância Híbrida: Esta técnica é caracterizada pela utilização das duas abordagens anteriormente apresentadas, Redundância Estática e Redundância Dinâmica [Koren e Krishna, 2007]. Esta característica híbrida confere ao sistema a possibilidade de, além de continuar seu andamento a despeito da ocorrência de uma falha, identificar qual o módulo de processamento sofreu a falha, removendo ou substituindo-o do processo. Para isso, é necessária a criação de um bloco que será responsável pela detecção do módulo que sofreu a falha, o qual está ligado às saídas dos módulos de processamento e saída do módulo votador. A Figura 2.9 ilustra um exemplo deste tipo de técnica.

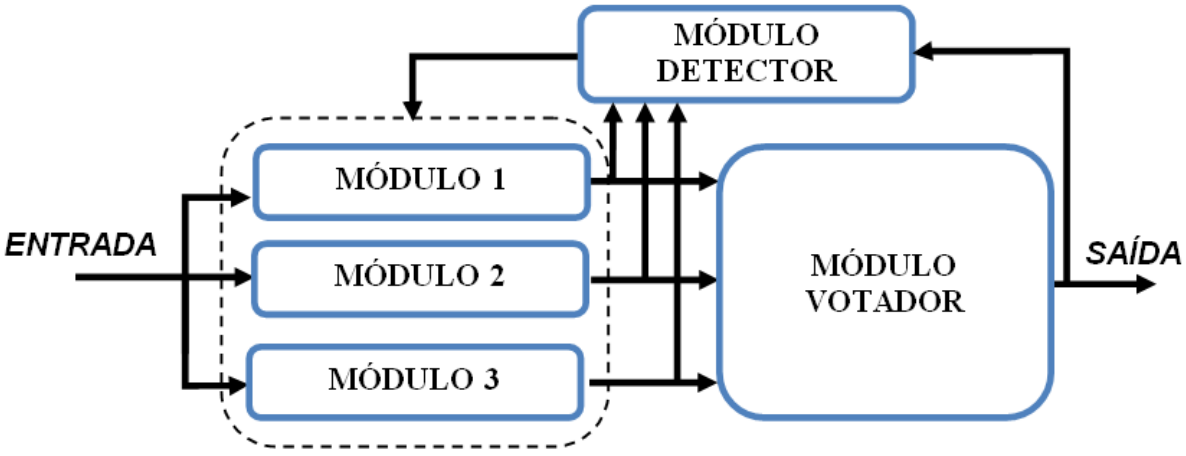

Figura 2.9: Modelo de Redundâncai Híbrida [Abd-El-Barr, 2007]

2.3. Sistemas Distribuídos

Os avanços das tecnologias de comunicação trouxeram ganhos expressivos que permitiram o desenvolvimento de grandes redes de computadores, como por

exemplo, a Internet[4]. Neste sentido, a aplicação de ferramentas que tem por objetivo a exploração dos benefícios proporcionados por estas redes acabaram por fomentar o desenvolvimento de sistemas distribuídos. Os sistemas distribuídos podem ser entendidos como um conjunto de módulos autônomos que se comunicam entre si por meio de troca de mensagens [Hariri et. al. 1992]. Entretanto, não é suficiente dizer que somente a troca de mensagem entre estes sistemas concebe a estes o valor agregado ao conceito de sistema distribuído. Faz-se necessário que estes módulos atendam alguns requisitos definidos por Colouris [Colouris et. al. 1994] e que as tarefas executadas estejam envolvidas em um propósito comum, tornando todos estes módulos um sistema singular e coerente para o usuário [Tanenbaum & Steen, 2002]. Estes requisitos são definidos como: compartilhamento de recursos, concorrência de processos, escalabilidade do sistema, tolerância a falhas e transparência em termos de aplicação para o usuário.

Sistemas distribuídos são idealmente empregados em aplicações onde se deseja aumentar o ganho do desempenho do sistema através da divisão de execução de tarefas a serem realizadas, porém, esta melhora de desempenho está associada a alguns desafios de desenvolvimento. Além da segregação física existente entre módulos autônomos, outros tipos de dificuldades devem ser observadas em face da criação de um sistema distribuído tais como: administração dos recursos compartilhados, sincronização entre os módulos distribuídos do sistema, manutenção da consistência da informação que é processada e armazenada entre outros [Hariri et. al. 1992]. Outros tipos de ganhos podem ser alcançados através da utilização de um sistema distribuído como, por exemplo, o aumento da vida útil do sistema por meio da realização do balanceamento de carga de processamento que este sistema pode fornecer.

Como podem ser observados, os desafios que são impostos pelo desenvolvimento de sistemas distribuídos estão fortemente atrelados aos conceitos de tolerância a falha. Toda via, nem todo sistema tolerante a falha pode ser dito um sistema distribuído. A tolerância a falha surge como uma ferramenta que pode ser utilizada para garantir o correto funcionamento do sistema, em conseqüência de uma

[4] Neste texto, Internet refere-se à rede mundial de computadores.

falha ocorrida em algum dos módulos distribuídos garantindo a transparência de execução para aplicação ou usuário.

2.4. Considerações

Nesta Sessão foi possível observar a evolução das tecnologias utilizadas no desenvolvimento de sistemas embarcados. Estes avanços tecnológicos permitiram que a indústria da microeletrônica criasse dispositivos cada vez mais compactos e densos em termos de silício. Durante bastante tempo os projetistas de sistemas embarcados estavam envoltos sobre qual tipo de arquitetura utilizar para o desenvolvimento de seus sistemas. As filosofias de desenvolvimento de sistemas baseados em RISC e CISC foram motivo de várias pesquisas. Paralelamente a estes avanços mencionados, a quantidade de pesquisas sobre as formas de tornar estes sistemas cada vez robustos seguiram a mesma tendência de crescimento. Neste sentido, várias técnicas foram apresentadas ao longo desta evolução através do emprego de técnicas de tolerância a falhas. Tais técnicas podem ser utilizadas em diversos níveis um sistema como também sobre quatro variáveis: tempo, informação, *software* e *hardware*. A escolha pela melhor técnica e nível de aplicação está intimamente relacionada à característica e restrições do sistema. A utilização destas técnicas serve principalmente para melhorar o grau de dependabilidade do sistema.

Os custos com o emprego destas técnicas impactam sobre diversos fatores do desenvolvimento de um sistema tais como: custo financeiro, aumento da área de silício ocupada, aumento do consumo de energia e potência, aumento da complexidade total, entre outros. Todos estes fatores devem ser levados em consideração quando do desenvolvimento de uma arquitetura que empregue técnicas de tolerância a falha para aumento da confiança do sistema. Além do aumento da confiança, estas técnicas também podem ser utilizadas com o intuito de melhorar a disponibilidade do sistema, ou no melhor caso, melhorar estes dois parâmetros.

Os sistemas distribuídos surgiram como uma boa técnica para aproveitamento dos ganhos fornecidos pela melhoria das tecnologias de redes de computadores. Neste sentido, estes sistemas podem melhorar a velocidade de processamento geral

do sistema através da divisão da execução de tarefas. Além disso, estes sistemas também podem ser utilizados com o objetivo de aumentar o tempo de vida do sistema através do balanceamento de carga, comutando a execução das tarefas entre os processadores. Outra importante característica destes sistemas é direcionada ao compartilhamento de recursos, que em alguns casos serve para garantir autenticidade da informação compartilhada. Porém, os benefícios fornecidos pelas aplicações dos sistemas distribuídos estão relacionados a diversos desafios que tornam o seu correto desenvolvimento uma difícil e complicada tarefa para os projetistas.

Sessão 3

3. Estado da Arte e Trabalhos Correlatos

Nesta Sessão foram analisados os trabalhos desenvolvidos por outros autores e que possuem alguma relação com o objeto de estudo desta dissertação. Sendo assim, foi feita uma revisão das principais arquiteturas, metodologias e modelos de desenvolvimento de sistemas embarcados.

3.1. Modelagens e Arquiteturas de Sistemas

As arquiteturas para desenvolvimento de sistemas tolerantes a falhas são objetos de grande interesse no que diz respeito ao emprego desta prática. O trabalho desenvolvido por Hariri [Hariri et. al. 1992] apresenta uma arquitetura de suporte para o desenvolvimento de sistemas distribuídos tolerantes a falhas. A idéia principal para a elaboração desta arquitetura está baseada na utilização de dois níveis hierárquicos para o armazenamento de dados e um algoritmo de votação para decidir sobre o valor processado. A Figura 3.1 ilustra um exemplo do modelo do nó.

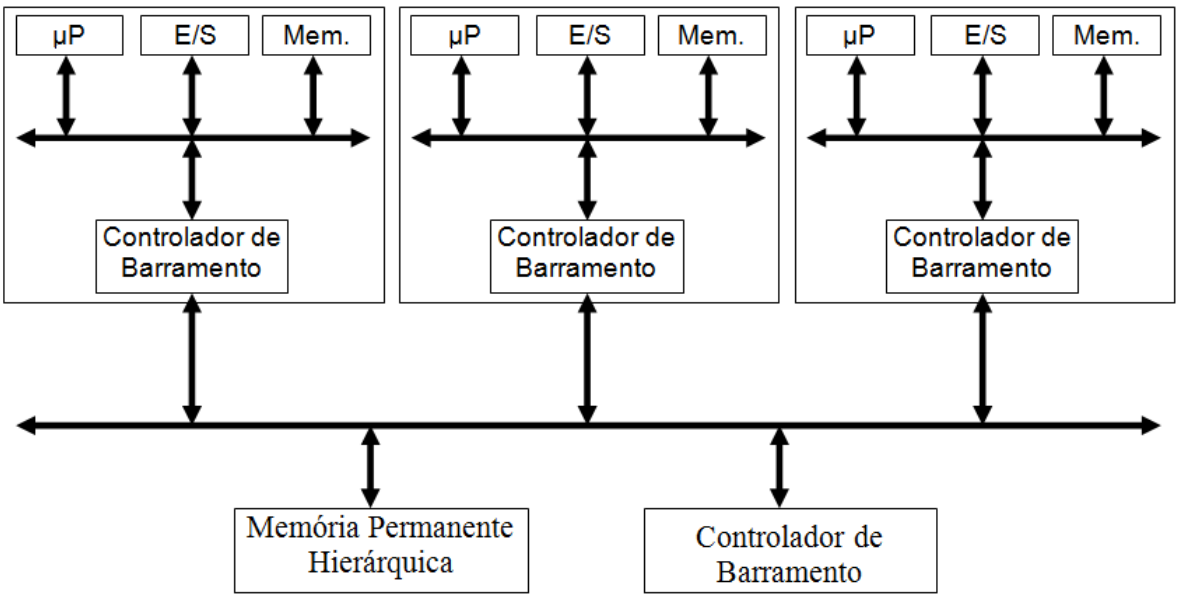

Figura 3.1: Nó da arquitetura proposta por Hariri

A hierarquização do armazenamento de dados consiste na utilização de uma memória de silício, em um primeiro nível funcionando como um *buffer*, e uma memória do tipo magnética em um segundo plano. Além do armazenamento dos dados processados, estas memórias também servem para guarda as informações referentes ao estado de processamento do processador. Neste trabalho, os autores propõem a criação de vários *clusters* os quais são compostos por diversos nós que se comunicam via *gateway*. O modelo apresentado mostrou-se bastante interessante, porém, alguns pontos críticos devem observados como o sincronismo entre os módulos de processadores, a forma de preempção de tarefas críticas, e a fragilidade na adoção de um único módulo de roteamento, o *gateway.*

Além dos trabalhos referentes à definição de arquiteturas tolerantes a falhas para sistemas embarcados, o processo de elaboração de uma arquitetura figura outras questões que também merecem especial atenção, como por exemplo, a forma comunicação entre os módulos computacionais. Neste sentido, Palma [Palma et. al. 2002] propõem em seu trabalho uma arquitetura, baseada em FPGAs comerciais, para criação de interfaces de comunicação entre IP Cores. Esta arquitetura tem como pressuposto um modelo análogo ao de uma interface *Peripheral Component Interconnect* (PCI) e a virtualização dos pinos de conexão. A fim de alcançar o objetivo proposto, faz-se necessária a criação de um árbitro de barramento o qual é responsável pelo gerenciamento de pinos e interconexões. Claramente, podemos perceber que a utilização de um único árbitro de barramento desta arquitetura implica em um ponto de fragilidade da mesma. Como forma de aumentar a confiabilidade do sistema poderia ser implementada uma estrutura de arbitragem redundante, onde uma desta estaria atuando em contingência.

A arquitetura de sistema proposta por Jeferry e Figueiredo [Jeferry & Figueiredo, 2007] consiste na utilização de *Virtual Machine Monitor* (VMM) e réplicas de *cores* a fim de contornar problemas decorrentes de falhas intermitentes ou transientes. A abordagem proposta pelos autores tem como alvo principal PCs e servidores, tendo como premissa a inviabilidade de réplica total do sistema. Neste sentido, como forma de alcançar os objetivos de sua abordagem é sugerida a criação de uma camada lógica intermediária entre o sistema operacional da aplicação e os

diversos *cores* de *hardware*. O número de réplicas a ser criado é dependente do grau de confiabilidade que se deseja alcançar e requer o uso de uma parcela da memória principal do computador e dos *cores* disponíveis. Diante disto, os autores afirmam que esta arquitetura é capaz de proteger o sistema contra erros de *softwares* podendo realizar detecção de saídas inconsistentes e detecção e recuperação de falhas arbitrárias em *hardware* e sistema operacional por meio da utilização de algoritmos de Byzantine. Apesar dos benefícios alcançados por meio da implementação desta arquitetura existem alguns problemas decorrentes de sua utilização. O principal destes problemas reside no *overhead* de processamento que é gerado pela copia das réplicas. Uma análise mais apurada a cerca deste processamento adicional ainda não foi realizada. Na Figura 3.2 podemos observar uma ilustração da arquitetura proposta.

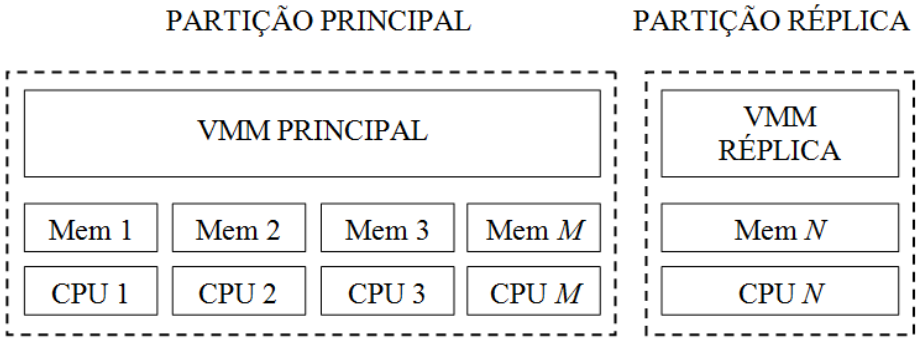

Figura 3.2: Arquitetura proposta por Jeferry e Figueiredo

3.2. Metodologias de Desenvolvimento

A complexidade e o expressivo custo associado ao desenvolvimento de sistemas torna indispensável a utilização de uma metodologia de projeto como forma de atenuar as dificuldades envolvidas na criação de tais sistemas. Além disso, o *time-to-market*[5] é outro fator importante que fomenta a adoção de uma metodologia para desenvolvimento de projeto. Neste sentido, Qi [Qi et. al. 2001] apresenta uma metodologia para desenvolvimento de sistemas baseada na reutilização de IP (*Intellectual Property*). Para resolução do problema proposto, os autores sugerem a utilização de duas metodologias de sistema: uma baseada na utilização de Interface

[5] Termo originário do inglês referente ao tempo necessário para que um produto seja lançado no mercado.

IP (IPI) e outra no desenvolvimento de sistemas baseados em plataformas. A metodologia baseada em IPI tem por filosofia a separação entre os barramentos de comunicação e o barramento de processamento. Desta forma, a mudança dos módulos de computacionais não interfere na arquitetura de comunicação pré-elaborada. Para o caso da interface IP, os autores utilizaram o padrão aberto da Motorola, o *Motorola Semiconductor Reuse Standards* (MSRS), e para o desenvolvimento de projetos baseado em plataformas o seguinte fluxo de projeto (Figura 3.3) foi utilizado. O objetivo final que deve ser alcançado pela implementação deste modelo é a reusabilidade de módulos IP e da arquitetura do SoC. Como forma de validação da proposta, foi implementado um estudo de caso segundo as metodologias apresentadas, resultando no desenvolvimento de uma plataforma baseada em *Mcore*. Esta plataforma consiste de um microcontrolador, unidade de memória, um módulo de integração, arquitetura de barramento hierárquica e módulos periféricos de IP. Com a possibilidade da reusabilidade de módulos IP e da arquitetura do sistema, os ganhos alcançados estão relacionados à diminuição de tempo execução do projeto e consequentemente um menor tempo para que o produto seja lançado para o mercado.

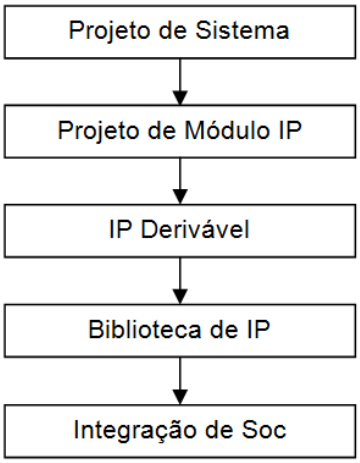

Figura 3.3: Fluxo de projeto de SoC baseado em plataforma

As necessidades pelo aumento da escalabilidade dos sistemas tornaram as pesquisas por redes em *chip* (*Network-on-Chip* - NoC) objeto de estudo, por estas se mostrarem uma solução plausível e viável para resolução destes desafios. Neste

sentido, tais pesquisas promoveram os estudos de metodologias para desenvolvimento de NoC de forma a atender aos requisitos de consumo de energia, área de silício e desempenho do sistema em um melhor caso. Chem [Chem et. al. 2007] apresenta um trabalho sobre o estudo destas metodologias e sugerem a utilização de uma metodologia heterogênea, chamada de RO2, para o desenvolvimento deste tipo de rede. Esta metodologia busca absorver características de uma rede *ad hoc* (rede irregular a qual possui o melhor desempenho, mas torna o desenvolvimento de IP cores impraticável, posto que não pode ser reutilizado) e o de uma rede regular (a qual pode ser utilizado para desenvolvimento de IP cores, porém possui desempenho insatisfatório quando comparado a uma rede *ad hoc*). A RO2 é composta por três topologias (Figura 3.4) regulares que podem ser configuradas de acordo com a necessidade do Sistema.

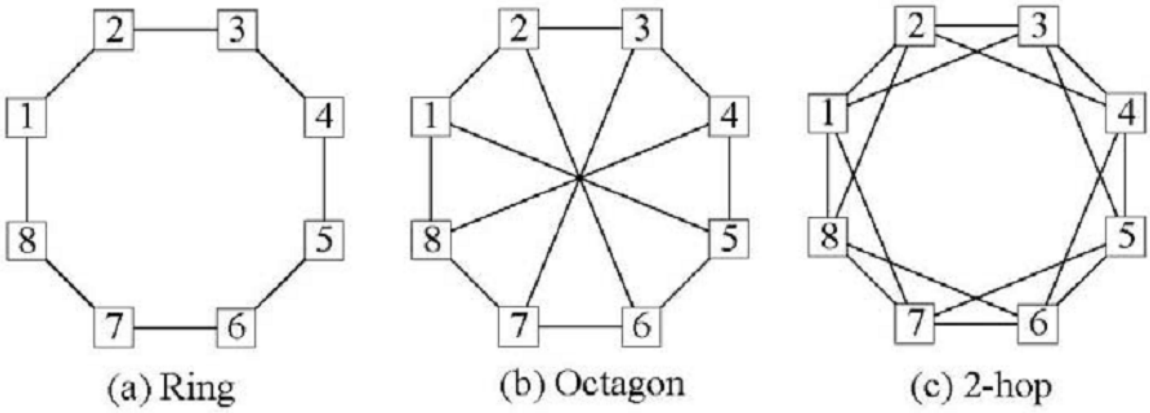

Figura 3.4: Conjunto de Topologias do RO2 [Chem et. al. 2007]

A escolha do tipo de topologia é feita de acordo com as necessidades de consumo de energia e conexões do sistema. Entretanto, esta escolha deve ser feita no momento da implementação do sistema pela ferramenta EDA (*Eletronic Design Automation*) através de um parâmetro que deve ser setado. Neste trabalho, os autores fizeram uma análise em termos de *throughput* e a complexidade das interconexões.

Ainda sobre as metodologias para desenvolvimento de sistemas, Wu [Wu et. al. 2008], sugere a criação de uma metodologia de projetos em nível de sistema (*System Level Design*) para FPGAs baseada em MPSoC (Micro-Processor System-

on-Chip). Esta metodologia tem por objetivo permitir aos desenvolvedores a exploração da diversidade de projetos de arquitetura, alternativas para alocação de *hardware-software* e opções de sincronização. As idéias que regem os princípios do trabalho proposto estão fundamentadas em: suporte para múltiplas manifestações de processamento de tarefas; virtualização de tarefas simples ou processos sem base para implementação física; ampla utilização do modelo de redes de processo de Khan (*Khan Process Network* - KNP); e um modelo unificado chamado de *Integer Linear Programing* (ILP) o qual permite a utilização de soluções simultâneas para projetos de arquitetura, *biding* e problemas de sincronização. Neste sentido, o processo de escolha do espaço de projeto para implementação híbrida de *hardware* e *software* é facilitado pelo desenvolvimento de um ILP baseado em um *Design Space Eploration* (DSE) automatizado. O DSE tem por objetivo escolher o projeto de arquitetura de acordo com os problemas de sincronização do sistema e *binding*,e a flexibilidade do modelo ILP permite que os projetistas insiram restrições de projeto que visam otimizar a escolha da arquitetura. O resultado obtido a partir do DSE são especificações em XML para arquitetura e, *binding* e sincronização. A Figura 3.5 ilustra o fluxo de projeto para o nível de sistema.

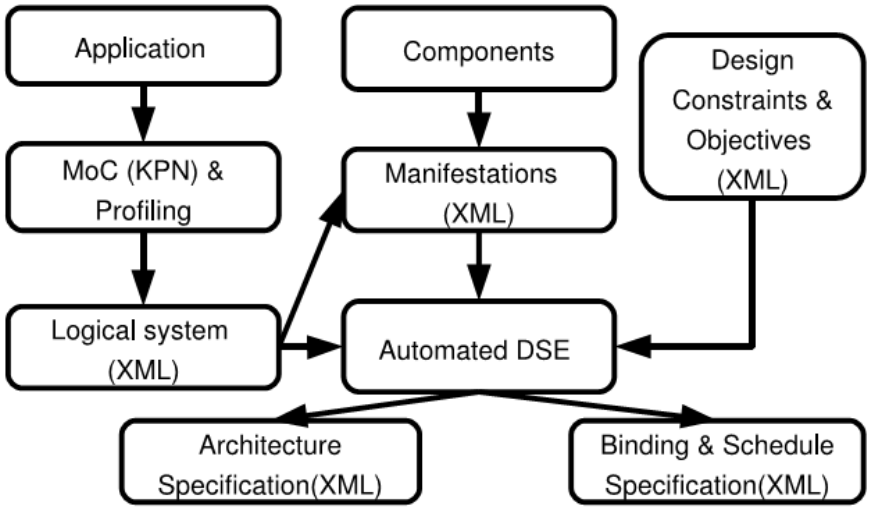

Figura 3.5: Fluxo de Projeto em Nível de Sistema [Wu et. al. 2008]

O sistema lógico da aplicação é definido de forma manual pelos projetistas, baseado na especificação KNP. Esta especificação permite a verificação do comportamento funcional da aplicação e mensurar os requisitos mínimos para tarefas de processamento de dados e filas de comunicação.

Os resultados de um estudo de caso apresentado pelos autores demonstraram-se satisfatórios, entretanto, para uma modelagem real de sistemas embarcados, faz-se necessária a modelagem do sistema considerando os atrasos de tempo inerentes a este tipo de sistema.

3.3. Considerações

Existem na literatura diversas modelagens, metodologias e arquiteturas voltadas para o desenvolvimento de sistemas embarcados. A maior parte destes trabalhos está preocupada com a metodologia de desenvolvimento arquitetural e maneiras para se obter um produto final que atendam às restrições físicas e temporais (*time-to-market*) de forma eficiente. Dentre estas metodologias, algumas estão relacionadas ao reaproveitamento de IP *cores*, enquanto outras se destinam ao fluxo de desenvolvimento de sistemas. Alguns trabalhos relacionados a arquiteturas de sistemas tolerantes a falha buscam explorar o conceito de uma técnica (redundância híbrida, modelo com votadores, etc.) para se alcançar a tolerância a falha. Na Sessão 4, será apresentado um modelo de referência para desenvolvimento destas arquiteturas.

Sessão 4

4. eOSI: Embedded Open Systems Interconnection

Nesta Sessão é descrito o modelo de referência em camadas, o qual é o trabalho principal desta Dissertação. Aqui são discutidas as formas pela qual o modelo é desenvolvido.

4.1. Contextualização do Trabalho

Em meados da década de 1980, apenas um engenheiro projetista era responsável pela criação de todos os RTL (*Register Transfer Level*) do dispositivo a ser desenvolvido [Maxfield, 2004]. Esta feita nos leva a conclusão que, o tempo de desenvolvimento de um projeto seria maior tanto quanto fosse a complexidade do mesmo e quanto menor fosse a experiência do engenheiro projetista encarregado. Assim, os projetos eram desenvolvidos como um único grande bloco e posteriormente sintetizado em um único chip. Desde essa época, as grandes empresas desenvolvedoras de sistemas em silício e pesquisadores do meio acadêmico estudam formas de melhoria no desenvolvimento destes sistemas.

O processo de desenvolvimento de sistemas embarcados em geral é realizado pela agregação de diversos componentes tais como: blocos de comunicação, blocos de processamento, blocos de conversão Analógico/Digital e Digital/Analógico, unidades de armazenamento, etc. A complexidade envolvida neste desenvolvimento requer a utilização de boas práticas de gerenciamento de projeto como forma de melhorar o processo de fabricação destes sistemas. Em sua maioria, os projetos de sistemas embarcados utilizam como premissa o modelo de arquitetura proposto por John von Neumann o qual é baseado em um modelo para desenvolvimento componentizado. Este modelo é muito interessante, pois dá ao projetista uma visão global do projeto do ponto de vista funcional de cada componente, o que lhe confere uma maior facilidade para poder definir sobre as estratégias que serão utilizadas no desenvolvimento do sistema. Estas estratégias

podem ser entendidas como as ferramentas que o projetista pode utilizar para atender ao escopo de projeto com base em seus *trade-off*. Por exemplo, o desenvolvimento de projeto que requeira um baixo consumo de energia e pouca ocupação de área de circuito indubitavelmente causará um impacto sobre o desempenho da velocidade deste SoC. O projeto de desenvolvimento de sistemas digitais proposto em [Vahid, 2007] pressupõe a existência de uma Parte Operativa (PO) e uma Parte de Controle (PC). A complexidade e o tamanho de cada um destes dois componentes determina, em partes, o cumprimento dos *trade-off* estabelecidos.

4.2. O Modelo em Camadas

O modelo proposto surgiu com base na idéia de desenvolvimento de dispositivos embarcados em FPGA, que atendessem a requisitos de alta confiabilidade e disponibilidade, voltados para aplicações em ambientes hospitalares. Além disso, outro fator que serviu de fonte motivadora foi o trabalho de doutorado elaborado por Valentim [Valentim, 2008]. Valentim propôs em sua Tese de Doutorado a criação de um protocolo multiciclos baseado em IGMP Snooping voltado para aplicações em ambientes hospitalares. As idéias que nortearam o desenvolvimento deste protocolo destinavam-se a: utilização de tecnologias de rede de baixo custo, no caso Ethernet; desenvolvimento de um protocolo determinístico; e criação de um protocolo que proporcionasse um baixo tráfego sobre a estrutura de rede. Todas essas abordagens mostraram-se satisfatórias, entretanto, o trabalho referente às questões de disponibilidade dos dispositivos e aumento da confiança do sistema necessitava do emprego de técnicas de tolerância a falha. Desta forma, o desenvolvimento de um modelo de referência para criação de sistemas embarcados tolerantes a falhas, como forma de aumentar o grau de disponibilidade do sistema, mostrou-se como uma solução factível para a resolução desta questão.

O modelo de referência eOSI modifica a forma de abstração para desenvolvimento de projetos de sistemas embarcados. Diferentemente de uma visão componentizada, o modelo de sistema em camadas tem inspiração no modelo de referência em camadas *Open Systems Interconnection* (OSI) estabelecido pelo *International Organization for Standardization* (ISO). O modelo de referência em

camadas permite que o projetista possa desenvolver o sistema obedecendo aos requisitos de suporte que cada camada exige. Diante disto, podemos destacar como vantagens deste modelo de abstração os seguintes pontos:

- Permitir a segmentação funcional para o desenvolvimento de um sistema tolerante a falhas atentando para critérios de disponibilidade do sistema;
- Aumento do desacoplamento entre os componentes do sistema;
- Melhorar a visualização das técnicas a serem empregadas no sistema com base nos requisitos deste;
- Estabelecimento de uma estrutura hierárquica para o fluxo de dados;
- Desenvolvimento de aplicações com base nas premissas de cada camada do modelo, ou seja, cada funcionalidade do sistema será realizada por uma camada específica;
- Orientar de forma sistemática o desenvolvimento de um sistema distribuído tolerante a falhas;

Estudos realizados sobre tecnologias de redes, modelos de desenvolvimento de sistemas e técnicas de tolerância a falha acabaram resultando na criação do modelo de referência eOSI. Tal qual o modelo OSI, o eOSI é constituído por um conjunto de camadas que descrevem as funcionalidades de serviços de tolerância a falhas que podem ser implantadas em um sistema. Entretanto, ao contrário do modelo OSI o qual define um modelo de referência orientado a protocolos e serviços, o modelo de referência eOSI é orientado a funcionalidades e conceitos relacionados a tolerância a falhas.

Sendo assim, o modelo de referência eOSI foi definido como tendo seis camadas: Aplicação, Detecção de Falha, Recuperação de Falha, Gerência de Redundância, Comunicação e Enlace. Este modelo, assim como a implementação definida por blocos, permite a criação de grupos interdisciplinares que podem ser responsáveis pela implementação de cada uma das camadas. Dentro de cada uma

das camadas o projetista pode realizar a componentização necessária para alcançar o objetivo definido para esta camada. Cada camada corresponderá a um nível do sistema que deve ser implementado para alcançar a dependabilidade requerida. Desta forma, o modelo de referência eOSI não tem a intenção de confrontar a forma de desenvolvimento de sistemas embarcados em componentes, mas sim estabelecer uma nova metodologia no desenvolvimento de sistemas embarcados. O modelo eOSI deve ser utilizado como referência no desenvolvimento da arquitetura do sistema. As sessões seguintes são destinadas ao esclarecimento das funcionalidades de cada camada do modelo. A Figura 4.1 ilustra a representação em camada deste modelo.

Figura 4.1: Camadas do Modelo de Referência eOSI [Morais et. al. 2009]

Comparando-se o modelo eOSI ao modelo OSI, as quatro primeiras camadas deste último (Física, Enlace, Rede e Transporte) correspondem à camada de Enlace e Comunicação do modelo eOSI. Neste caso, Física e Enlace correspondem à camada de Enlace (eOSI) e Rede e Transporte correspondem à camada de Comunicação.

4.2.1. Camada de Aplicação

Assim como no modelo OSI, a Camada de Aplicação é destinada ao desenvolvimento dos dispositivos responsáveis pela execução das tarefas relacionadas ao processo. Tendo como alusão o ambiente hospitalar proposto por Valentim [Valentim et. al. 2008], tais dispositivos podem ser um infusor de insulina, monitor cardíaco ou um oxímetro de pulso, pro exemplo. O desenvolvimento destes dispositivos pode ser realizado de forma independente entre as equipes do projeto. Isto significa que a equipe ou projetista responsável pelo desenvolvimento do módulo de processamento da Camada de Aplicação não necessita se preocupar com requisitos e técnicas de tolerância a falhas. Todo o esforço desta camada é destinado ao melhor desenvolvimento deste módulo de processamento, com isto, uma equipe especializada no desenvolvimento de processadores para oxímetro de pulso, por exemplo, pode fazer parte de um projeto baseado neste modelo, sem que para isso sejam necessários os conhecimentos de técnicas de tolerância a falhas.

Esta forma de desenvolvimento orientada a funcionalidades, além de permitir uma nova forma de visão abstrata de alto nível do sistema, também possibilita que este seja desenvolvido com um maior grau de desacoplamento. A única consideração que deve ser obedecida diz respeito ao pacote de dados que deve ser enviado à camada inferior.

4.2.2. Camada de Detecção

A Camada de Detecção destina-se ao desenvolvimento de componentes do sistema que serão responsáveis pela detecção de falhas. As formas que o sistema pode utilizar para detectar a ocorrência da falha sobre o módulo de processamento da Camada de Aplicação são diversas. Cabe ao desenvolvedor/idealizador desta camada decidir sobre qual técnica será utilizada para a detecção de falha.

Baseados nos conceitos e técnicas de tolerância a falha apresentados na Sessão 2, uma das formas para se detectar a ocorrência de uma falha seria por meio da utilização de um comparador ligado a saída de dois módulos de processamento idênticos, os quais estariam recebendo o mesmo dado de entrada. A divergência entre os valores processados por estes dois módulos da Camada de Aplicação

evidencia a ocorrência da falha. Esta forma de detecção também poderia fazer uso de outras técnicas, como por exemplo, uma técnica baseada no conhecimento prévio do valor de saída do módulo da Camada de Aplicação, ou melhor, baseada na faixa de valores que este poderia produzir em um determinado instante. De posse desta informação, o detector pode ser implementado com limites inferior e superior, e no caso de o valor resultante do módulo da Camada de Aplicação não estar contido dentro deste intervalo, significa que uma falha aconteceu. Outra abordagem que também pode ser implementada é por meio da utilização de uma rede neural capaz de realizar esta função. A rede neural implementada pode ser treinada e, conforme for aprimorada poderá ser utilizada para a estimação de valores. No caso em que uma falha é detectada, esta deve ser reportada à Camada de Recuperação para as ações corretivas possam ser tomadas.

4.2.3. Camada de Recuperação

Esta camada é responsável por receber as informações geradas pela Camada de Detecção e agir de forma a recuperar o sistema caso seja constatada a ocorrência de falha. Assim como nas formas de implementações descritas para a camada anterior, os mecanismos utilizados para recuperação do sistema são muitos. Cabe ao projetista a escolha da implementação mais adequada.

A forma de recuperação assegurada ao sistema é, de certa forma, dependente do tipo de reportação que é enviada pela Camada de Detecção. Por exemplo, se informação enviada para a Camada de Recuperação contiver dados suficientes para informar o tipo de falha, e não somente que uma falha aconteceu, o sistema poderá ser levado para um estado de operação onde a falha não cause danos. Sendo assim, o sistema seria levado para o melhor estado de operação de acordo com o tipo de falha, ou seja, somente as operações afetadas pela falha deixariam de ser cumpridas. Esta abordagem mostra-se dispendiosa do ponto de vista de *overhead* de processamento, posto que se faz necessária a análise do tipo de falha reportada. Entretanto, em sistemas onde o acesso, substituição ou reparo sejam impossíveis e se deseje ter um alto grau de disponibilidade, mesmo perdendo-se algumas funcionalidades, torna-se viável a aplicação desta técnica. Em uma

abordagem mais simples, a recuperação poderia ser realizada pela substituição do módulo em estado de falha.

Ainda sobre esta camada, outras técnicas voltadas para a garantia de segurança operacional podem ser empregadas. Situações em que requeiram a parada de operação do sistema (*fail-stop*), para, em contrapartida, garantir que este não cause danos às aplicações que deste dependam ou até mesmo a segurança física do usuário. Mecanismos para avanço ou retorno de falha também podem ser utilizados a fim de prover mais uma funcionalidade a esta camada. Estes mecanismos são chamados de recuperação por avanço de falha (*forward error recovery*) e recuperação por retorno de falha (*backward error recovery*). Considerações sobre o efeito dessas técnicas devem ser feitas devido aos problemas recorrentes que elas podem gerar. Em sistemas distribuídos, a utilização destas técnicas pode ser bastante complicada pelo fato de algumas mensagens ficarem sem respostas. Isto faz com que outros processos que dependam destas respostas tenham que retroceder o processamento que podem gerar *overhead* sobre outros criando um efeito em cascata [Jansch-Porto & Weber, 1997]. Uma forma de minimizar a ocorrência deste problema é através da limitação da troca de mensagens entre processo. Desta forma, o avanço de falha visa desconsiderar o efeito desta.

4.2.4. Camada de Redundância

Esta camada é responsável pelo estabelecimento das regras de trocas de mensagens e pela aplicação do controle de acesso ao barramento específico para este fim, ou seja, ela realiza o gerenciamento necessário entre os dispositivos redundantes. Para esta camada são previstas as estratégias de comunicação para implementação do sistema distribuído. Estas estratégias possibilitam ao projetista a criação de uma rede de módulos de processamento que podem atuar tanto em paralelo sobre uma dada tarefa (balanceamento de carga) como em modo de redundância para aplicação de técnicas de tolerância a falhas. As complexidades envolvidas no desenvolvimento desta camada fazem com que ela se torne uma das mais críticas do modelo e, por esta razão, requer maior atenção. Problemas relacionados a sincronismo e consenso devem ser tratados por esta camada. Para o

caso em que se deseje aumentar a vida útil do sistema, uma estratégia baseada em módulos de reposição pode ser implantada. Neste caso, a ocorrência de falha em um módulo resultará na ativação do módulo de reposição, que por ventura se encontre inativo. Claramente, como o módulo estava inativo, podemos imaginar que seus componentes não sofreram desgastes devido ao uso, entretanto, estes módulos devem estar armazenados dentro de algum invólucro que seja capaz de fornecer elevado grau de proteção contra poeira e umidade. Estas medidas de proteção têm por objetivo minimizar a ocorrência de desgastes oriundos de uma possível, oxidação ou curto circuito, decorrentes destes fatores externos. Cuidados também devem ser tomados contra a influência de outros tipos de interferências externas, tais como térmica, eletromagnética e vibratória.

Esta camada proporciona ao modelo eOSI [Morais et. al. 2009] a criação de um sistema tolerante a falha que seja capaz de realizar tanto o mascaramento de falhas como as técnicas para detecção, confinamento e recuperação de falhas. O suporte que esta camada fornece às demais permite que neste nível do modelo seja criado um protocolo de troca de mensagens para realização do mascaramento, o que também implica na criação de um sistema distribuído. A idéia desta camada é criar um barramento, independente do barramento de comunicação, para estabelecimento desta política de gerenciamento. Isto significa que o tráfego sobre a rede de dados não será afetado pelas mensagens trocadas entre os diversos módulos distribuídos. Isto também é revertido em ganho de desempenho do sistema, pois, de forma análoga, o barramento dedicado para mensagens de controle não é utilizado para envio de mensagens de dados.

4.2.5. Camada de Comunicação

O desenvolvimento dos dispositivos desta camada é dependente do tipo de protocolo de comunicação utilizado entre os módulos de processamento. No caso específico do PM-AH, todas as regras para envio e recebimento de mensagens devem ser implementadas em um dispositivo embarcado que ficará localizado nesta camada. Protocolos destinados ao gerenciamento do sistema distribuídos devem ser trabalhados na camada de Gerenciamento de Redundância.

Uma das vantagens dessa abordagem esta no fato de facilitar o desenvolvimento do protocolo de comunicação, ou até mesmo permitir a utilização de um protocolo já existente para o desenvolvimento desta aplicação, sem que para isso sejam necessários maiores cuidados sobre tolerância a falhas. Entretanto, a fim de aumentar a confiabilidade sobre o sistema distribuído, protocolos capazes de estabelecer conexões seguras e confiáveis, como é o caso do TCP, devem ser utilizados. A recomendação para a utilização deste tipo de protocolo reside no fato de que estes requerem a confirmação de entrega e recebimento da mensagem. Além de conferir uma maior robustez ao sistema, permite a detecção de falhas devido à perda do link de comunicação.

Uma segunda abordagem que pode ser utilizada no desenvolvimento dos mecanismos de segurança (*safety*) para esta camada, diz respeito aos mecanismos para detecção e correção de erros. Códigos para detecção e correção de erro tais como paridade, *checksum* e *Cyclic Redundancy Check* (CRC) podem ajudar o sistema conferindo-lhe maior confiabilidade.

4.2.6. Camada de Enlace

A Camada de Enlace deste modelo assemelha-se bastante à camada de enlace do modelo OSI. A função desta camada é converter os sinais digitais que serão enviados para o tipo de sinal que o meio possa transmiti-lo. De forma análoga, devem existir dispositivos transdutores capazes de captar estes sinais oriundos do meio de transmissão e convertê-los para o formato digital. Neste sentido, diversos são os meios ao qual o sistema poderá ser conectado (metálico, fibra óptica, ar). Diante disto, o desenvolvimento de conversores A/D e D/A deve ser voltado para esta camada. Mais uma vez, os projetistas desta camada não precisam necessariamente estar preocupados com a lógica para tolerância a falha, esta questão será suportada pelas camadas superiores.

4.2.7. Considerações sobre o Fluxo de Dados

Uma característica importante deste modelo e que o difere da filosofia do modelo de referência OSI deve-se ao fato deste não possuir um fluxo de dados

bilateral entre suas camadas. Os dados somente deverão possuir um fluxo descendente partindo da Camada de Aplicação até a Camada de Enlace, entretanto, o sentido contrário não é permitido entre todas as camadas. No modelo OSI os dados são empacotados e desempacotados de acordo com o fluxo que se segue, envio ou recepção respectivamente. O objetivo deste modelo é permitir o emprego de técnicas de tolerância a falhas, logo não faz sentido os dados que se encontram, em determinado momento, na Camada de Redundância “subirem” para a Camada de Recuperação. Todos os dados que forem destinados à Camada de Aplicação serão tratados pela Camada de Comunicação e enviados por esta diretamente. Desta forma, pode-se deduzir que o caminho a ser percorrido pelos dados da Camada de Aplicação para o barramento de comunicação é mais lento que o contrário, ou seja, o caminho a ser percorrido desde a Camada de Comunicação até a Camada de Aplicação é mais rápido que o caminho inverso. A Figura 4.2 ilustra o fluxo de dados que ocorre neste modelo.

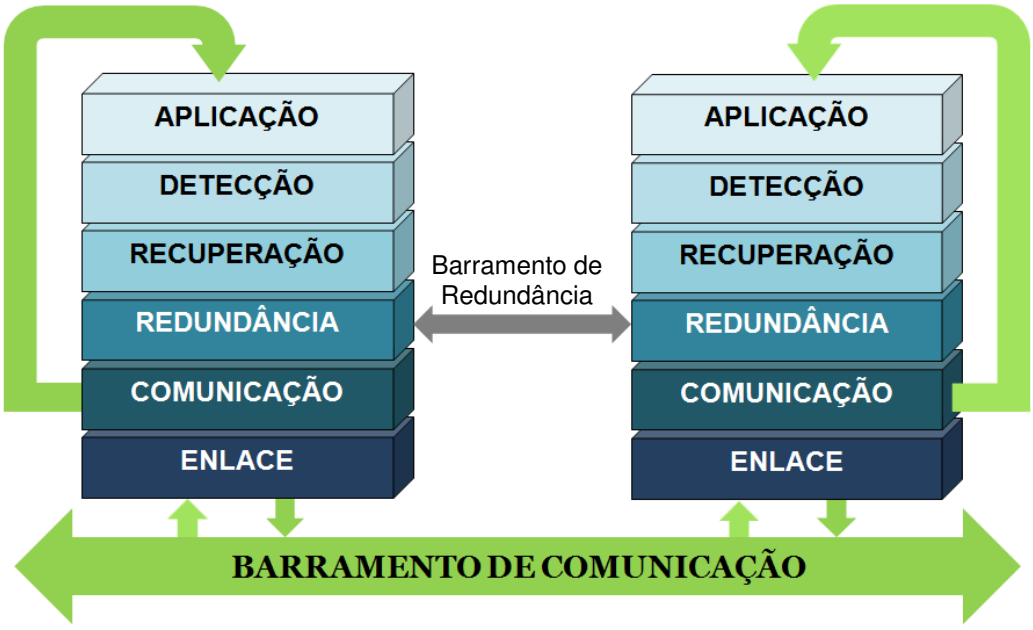

Figura 4.2: Fluxo de dados para o modelo eOSI

4.3. Estudo de caso

Como forma de consolidar a validação deste modelo, a implementação de uma arquitetura, em FPGA, baseada no modelo de referência eOSI será realizada.

Desta forma, a fim de dar continuidade aos estudos sobre sistemas embarcados para ambientes hospitalares, este trabalho de dissertação desenvolveu uma arquitetura de suporte que será utilizada para aumentar a confiabilidade e disponibilidade dos dispositivos biomédicos. Sendo assim, não é considerado escopo desta Dissertação a elaboração um sistema completo fazendo uso de todas as camadas apresentadas, ou seja, esta Dissertação não tem o objetivo de desenvolver os dispositivos da Camada de Aplicação, Comunicação e Enlace.

4.3.1. Camada de Aplicação, Detecção e Recuperação

A arquitetura de suporte leva em consideração a existência de dois módulos de processamento operando em paralelo, no entanto, um destes módulos atua no modo inativo somente entrando em operação no caso da ocorrência de falha no módulo de processamento principal. Cada um destes módulos é composto por dois dispositivos de processamento (Camada de Aplicação) idênticos e a detecção da falha acontece através da comparação dos resultados destes dispositivos. Neste caso, a Camada de Detecção implementa um comparador que é capaz de verificar a saída de ambos os módulos e havendo divergência entre os resultados de saída destes é constatada a existência da falha.

A implementação de um dispositivo que fosse capaz de controlar a execução dos dispositivos da Camada de Aplicação é dependente da arquitetura e forma de operação destes. Para que o módulo da Camada de Recuperação possa conduzir tais dispositivos a um estado de operação seguro é necessário conhecer os estados de operação. Além disso, os módulos da Camada de Aplicação devem ser desenvolvidos tendo como premissa o controle exercido pela Camada de Recuperação. Outra forma de realizar a recuperação do sistema consiste na implementação de outro(s) bloco(s) contendo também dois dispositivos biomédicos idênticos. Neste caso, seria implementada uma Redundância Dinâmica em nível de *hardware*, a qual poderia ser do tipo *hot* ou *cold standby*. A implementação da *Redundância Dinâmica Cold Standby* implica na utilização de uma maior quantidade de *hardware*, haja vista a necessidade de transferência de contexto. Além disso, o tempo necessário para recuperação do sistema também seria maior. A Figura 4.3

ilustra o exemplo da arquitetura de suporte para as duas camadas superiores do modelo de referência eOSI.

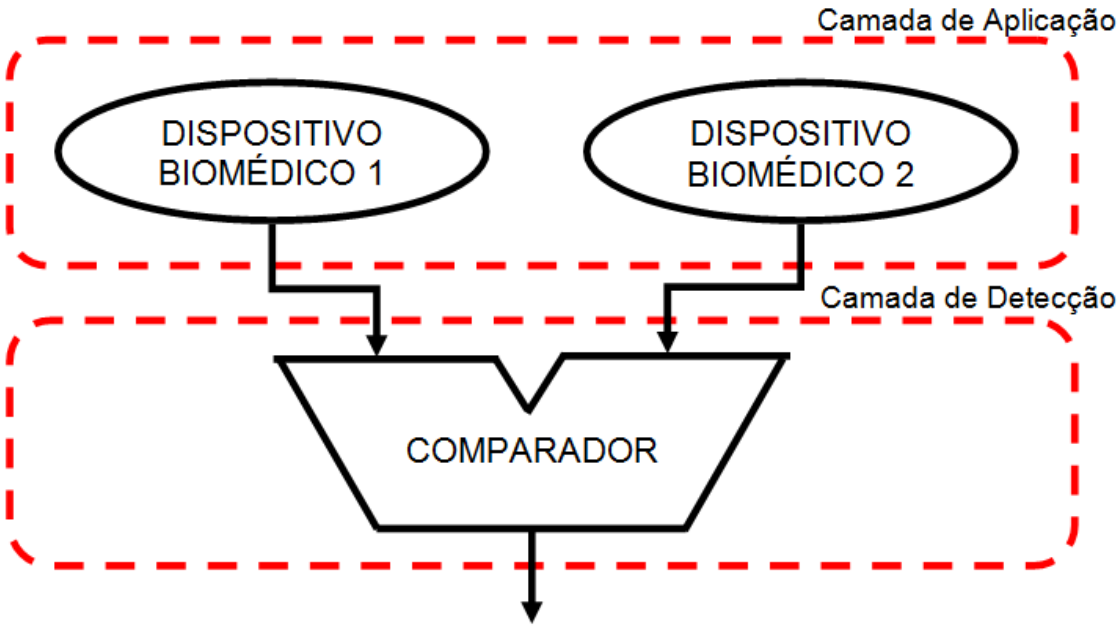

Figura 4.3: Arquiterura da Camada de Aplicação e Detecção

Como forma de reduzir a área que será ocupada e a potência consumida pelo sistema. A arquitetura de suporte proposta de neste estudo de caso não implementará os módulos referentes à Camada de Recuperação. Além disso, outra razão para a não implementação do módulo da Camada de Recuperação reside no fato de que este sistema deve ter um tempo de resposta adequado aos requisitos gerais do protocolo de comunicação e a sua forma de operação não pode ser degradada (falha que cause restrições sobre algumas operações). Desta forma, para que a interferência sobre este tempo de resposta seja mínima a recuperação do sistema através de transferência de contexto não será utilizada, e a detecção da falha será encaminha diretamente para a Camada de Redundância. A escolha pela implementação ou não desta Camada de Recuperação é de escolha do projetista, entretanto conceitos referentes à recuperação de falhas serão tratados pela Camada de Redundância.

4.3.2. Camada de Redundância

Esta camada é composta por um dispositivo de processamento que é responsável pelo gerenciamento da Camada de Comunicação, ou seja, pelo envio dos dados que devem ser envelopados pelos protocolos que esta camada implementa e pela troca de mensagens com os demais dispositivos que compõe o sistema distribuído (*cluster*). Neste sentido, este dispositivo de controle é o mais simples possível para que o *overhead* de processamento gerado devido à realização deste controle interfira o mínimo possível no desempenho final do sistema. Partindo deste pressuposto, o desenvolvimento deste controlador é baseado em dois algoritmos para o gerenciamento do fluxo de dados: um para o Módulo Principal e outro para o Módulo Redundante. Vejamos a seguir os algoritmos que servem de base para a criação dos controladores da Camada de Redundância do Módulo Principal e do Módulo Redundante.

4.3.2.1. Algoritmo do Módulo Principal

Este algoritmo é especificamente desenvolvido para o Módulo Principal, pois existem características inerentes ao Módulo Secundário que não são necessárias e nem fazem sentido serem tratadas pelo Módulo Principal. A seguir é apresentada a descrição e comentários sobre o algoritmo (Quadro 4.1) que servirá de base para implementação da máquina de estados do controlador do Módulo Principal.

```
1.   Enquanto (!falha ou !timeout)
2.        Se (existe_informação_para_envio)
3.             deteccao.zerar_timeout();
4.             mp.despachar(informacao);
5.        Senão
6.             mp.aguardar_informacao();
7.   Fim enquanto;
8.   mp.ativar_redundante();
9.   mp.entrar_modo_inoperante();
```

Quadro 4.1: Algoritmo do módulo principal

Neste algoritmo, *mp* representa o Módulo Principal. A percepção da manifestação da falha é dada por um sinal enviado pela Camada de Detecção para a Camada de Redundância. Tal evento faz com que um o módulo de controle da Camada de Redundância envie um sinal de ativação para o módulo redundante e em seguida entre em um estado de inoperação. Neste estado inoperante, o módulo principal não é capaz de enviar mensagens de dados para o barramento de comunicação. O módulo principal também pode entrar em estado inoperante caso um sinal de *timeout* seja disparado. Desta forma, o Módulo Principal permanecerá no estado de espera até que uma falha se manifeste ou o contador atinja o limite máximo de contagem (linha 1). No instante em que alguma informação chega da Camada de Detecção para a Camada de Redundância (linha 2) e nenhuma falha é constatada, o Módulo principal se encarrega de encaminhar esta mensagem para a Camada de Comunicação (linha 4) e a Camada de Detecção zera o contador (linha 3). Caso nenhuma mensagem chega até esta camada, o Módulo Principal permanece no estado de espera aguardando a chegada dos dados (linha 6). Quando uma falha é detectada (linha 1) o Módulo principal ativa o Módulo Secundário (linha 8) e passa para o estado inoperante.

O *timeout* tem por objetivo a detecção de falhas devido à perda de comunicação entre as camadas que estão acima da Camada de Redundância. O início da contagem do *timeout* é iniciada quando alguma informação é enviada da Camada de Comunicação para a Camada de Aplicação. Sendo assim, o tempo que deve ser estipulado para este contador aguarde até a efetuação do disparo é dado pela Equação 4.1:

$$T_{timeout} = T_{c\_aplic} + T_{c\_detc} + T_{c\_rec} \quad \textbf{(4.1)}$$

onde,

$T_{timeout}$: Tempo total de espera

T_{c_aplic}: Tempo necessário para processamento na Camada de Aplicação

T_{c_detc}: Tempo necessário para processamento na Camada de Detecção

T_{c_rec}: Tempo necessário para processamento na Camada de Recuperação

No caso específico desta arquitetura, o tempo de processamento da Camada de Recuperação é desconsiderado, posto que nenhum módulo é implementado para esta camada.

Caso todo o processamento seja efetuado sem a ocorrência de falha, o módulo principal (*mp*) recebe um sinal de que a informação está pronta para ser enviada. Neste mesmo instante, a Camada de Detecção envia um sinal para que o contador (*timeout*) zerar sua contagem, impedindo o envio de um sinal de inoperação para o módulo de controle da Camada de Redundância. Em seguida, um sinal de controle, juntamente com a informação processada, é enviado para a Camada de Comunicação executar o devido empacotamento do dado a ser transmitido e volta a esperar pelo processamento de uma informação. Este processo é repetido de forma ininterrupta até a ocorrência de uma falha ser constatada.

4.3.2.2. Algoritmo do Módulo Redundante

Como já mencionado, o algoritmo proposto para o Módulo Redundante possui algumas peculiaridades que não fazem sentido serem tratadas pelo Módulo Principal, como por exemplo, a existência de um estado inativo. Vejamos a seguir a descrição e comentários sobre o algoritmo (Quadro 4.2) que servirá de base para implementação da máquina de estados do controlador do Módulo Redundante.

Para este algoritmo, a partir da linha 5 até a linha 13 a lógica trata-se da mesma já descrita para o algoritmo do Módulo Principal (Quadro 4.1). A primeira diferença que podemos observar entre as máquinas de estado que serão gerados é a existência de um estado inativo. Este estado é semelhante ao estado inoperante, entretanto, o que os difere é o estado do sistema. Quando o módulo encontra-se em um estado inativo significa que ele é o módulo redundante de outro e nenhuma falha lhe ocorrerá até o momento. Para a máquina de estados que será descrita na sessão 5.3.4.1, as linhas de 1 a 3 representam o estado inativo. No modo inativo, o módulo não possui permissão para enviar mensagens no barramento de comunicação. Quando o módulo entra no estado inoperante significa que alguma falha aconteceu e

ele não pode mais enviar mensagens através do barramento de comunicação. Sendo assim, o que difere um estado de outro é simplesmente a ocorrência da falha.

```
1.   Enquanto (!timeout1 ou !msg_ativacao)
2.        mr.permanecer_inativo();
3.   Fim enquanto;
4.   mr.inativar_timeout1();
5.   Enquanto (!falha ou !timeout2)
6.        Se (existe_informação_para_envio)
7.             deteccao.zerar_timeout2();
8.             mr.despachar(informacao);
9.        Senão
10.            mr.aguardar_informacao();
11.  Fim enquanto;
12.  mr.ativar_redundante();
13.  mr.entrar_modo_inoperante();
```

Quadro 4.2: Algoritmo do módulo secundário

Outra característica que diferencia as arquiteturas do Módulo Principal e Secundário é a existência de um *timeout1* (contador) extra na arquitetura do Módulo Secundário. O objetivo deste segundo *timeout1* é verificar a ocorrência de falha no Módulo Principal ou falha no meio de comunicação entre as Camadas de Redundância dos módulos. O tempo para o cálculo deste *timeout* é dado pela seguinte Equação 4.2:

$$T_{timeout1} = T_{proc\_mp} + T_{proc\_leitura} \qquad (4.2)$$

onde,

$T_{timeout1}$: Tempo total de espera para o processamento do Módulo Principal

T_{proc_mp}: Tempo necessário para processamento do Módulo Principal

$T_{proc\_leitura}$: Tempo necessário para processamento do dispositivo de leitura do barramento de comunicação

A partir da Equação 4.2 podemos deduzir que, o tempo de espera para detectar a ocorrência da falha no Módulo Principal ou no barramento de controle (barramento entre as Camadas de Redundância) é dado pelo tempo que é necessário para que o Módulo Principal realize todo o processamento e encaminhe o dado pelo barramento de comunicação acrescido do tempo necessário para que o dispositivo de leitura do barramento de comunicação realize o seu processamento. Este dispositivo deve ler o dado que está sendo enviado pelo barramento de comunicação e verificar se o mesmo foi enviado pelo Módulo Principal. Caso este dado tenha sido enviado pelo Módulo Principal, este dispositivo envia um sinal para que o *timeout1* seja zerado. Se o tempo referente ao processamento do Módulo Principal já tiver sido excedido e o dispositivo de leitura do barramento de comunicação não detectar o envio do dado pelo Módulo Principal, significa que este sofreu alguma falha e que a comunicação com o Módulo Redundante também falhou.

Caso a comunicação entre os módulos não falhe e o módulo principal sofra uma falha, então o Módulo Redundante receberá um sinal de ativação e o *timeout1* será zerado. Logo em seguida o Módulo Redundante, caso não tenha sofrido uma falha também, despachará a informação para ser enviada pela Camada de Comunicação. A partir deste momento, a forma de atuação do Módulo Redundante é semelhante a do Módulo Principal.

4.3.3. Camada de Comunicação e Enlace

O desenvolvimento destas duas camadas requer um estudo mais aprofundado sobre as técnicas de conversão Analógico/Digital e Digital/Analógico, além do estudo necessário para embarcar o protocolo de comunicação. Tais implementação, pela complexidade envolvida em cada uma, poderiam resultar em outra Dissertação de Mestrado. Por estes motivos, e por também fugir ao escopo que esta Dissertação sugere, tais componentes não serão implementados, sendo posteriormente sugeridos como um trabalho futuro.

4.3.4. Considerações sobre a Arquitetura

Para esta arquitetura foram sugeridas a implementação de dois módulos de processamento atuando de forma paralela em um mesmo processo e recebendo a mesma informação para ser processada. Esta arquitetura busca a simplicidade de redundância através da utilização de somente dois módulos (um principal e outro redundante). Entretanto, uma arquitetura composta por mais de dois módulos atuando de forma paralela é perfeitamente factível. Uma forma de se conseguir implementar esta estrutura de multi-módulos é através da criação de um esquema de envio de mensagens endereçáveis. Neste caso, a arquitetura seria composta por um Módulo Principal e diversos Módulos Redundantes. Para cada um dos módulos deve ser atribuído um endereço único capaz de distingui-los. Detalhes sobre os temporizadores foram omitidos como forma de simplificar a ilustração e valorizar o fluxo de dados do sistema.

A fim de permitir uma melhor visualização da arquitetura deste estudo de caso, a Figura 4.4 demonstra um arranjo lógico dos componentes desta ilustração e valorizar o fluxo de dados do sistema.

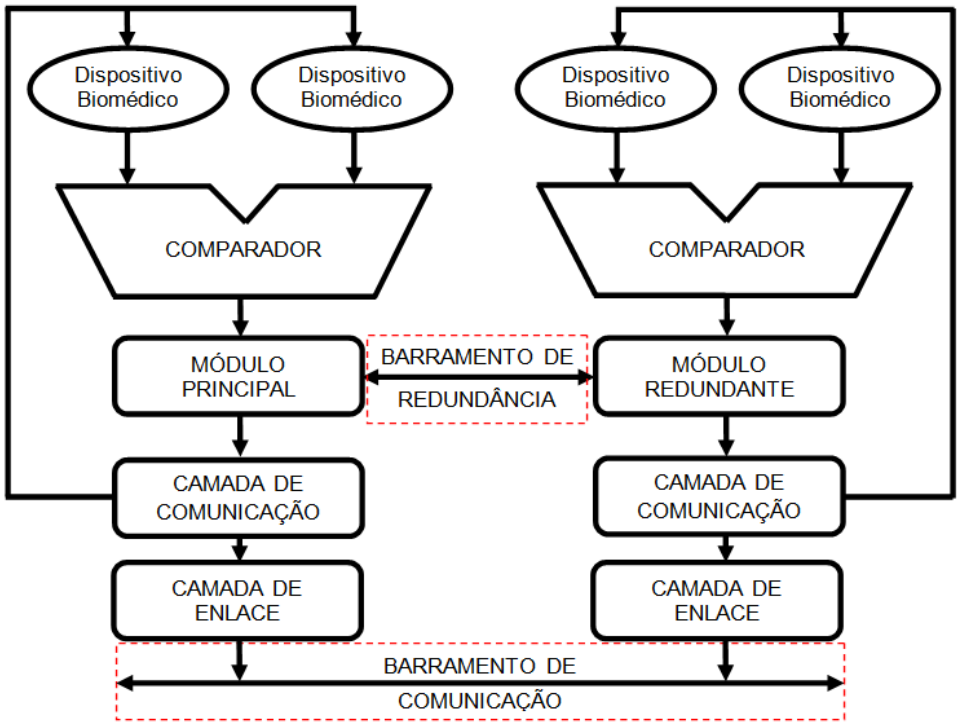

Figura 4.4: Arquitetura proposta para o estudo de caso

Este é apenas um exemplo de arquitetura que pode ser desenvolvida segundo o modelo de desenvolvimento proposto. As idéias acerca das arquiteturas podem ser exploradas de acordo com a finalidade do sistema. A abordagem seguida para o desenvolvimento desta arquitetura baseou-se nas premissas de confiança e disponibilidade com respostas rápidas à ocorrência de falhas.

Sessão 5

5. Implementação e Resultados

Nesta Sessão é apresentada a plataforma utilizada para implementação da arquitetura descrita no Estudo de Caso, bem como as máquinas de estado que representam os algoritmos para as Camadas de Redundância e a interligação entre os Diagramas de Blocos de cada uma das Camadas do Modelo.

5.1. Plataforma

A Plataforma utilizada para implementação do Estudo de Caso foi a DE2 Development and Education Board, modelo EP2C35F672C6 da família Cyclone II da fabricante Altera Devices. Este modelo foi escolhido por se tratar de uma FPGA que já havia sido utilizada em projetos anteriores, bastante didática e dotada de recursos suficientes para validação/implementação da arquitetura descrita no Estudo de Caso. A seguir, veremos as principais características de especificações de hardware e software suportados por esta plataforma.

5.1.1. Dispositivos de Entrada e Saída

No que diz respeito aos mecanismos de comunicação com meio externo, esta plataforma oferece suporte para uma grande variedade de dispositivos de entrada e saída. A lista destes mecanismos é demonstrada a seguir:

- 1 interface 10/100 Ethernet em total acordo com a especificação IEEE 802.3u;
- 1 serial entrada DB-9 para interface RS-232;
- 1 sensor infravermelho IrDA;
- 1 porta USB 2.0;
- 1 porta USB *Blaster* (configuração e programação da FPGA);

- 1 porta PS/2 (mouse e teclado);
- 3 portas para entrada e saída de áudio (*Line-in, Line-out, microphone-in 24-bit audio CODEC*);
- 1 saída de vídeo (VGA 10-bit DAC);
- 1 entrada de vídeo (NTSC/PAL/Multi-format);
- 2 expansões para até 76 pinos de entrada e saída;
- 4 *pushbuttons*, do tipo normalmente aberto, dotados de um circuito *Schmitt trigger* para diminuição de ruído devido ao chaveamento;
- 18 mini *switches* para chaveamento de sinais;
- 18 LEDs vermelhos e 9 LEDs verdes;
- 1 *display* de LCD 16x2;
- 8 *displays* de sete segmentos;

5.1.2. Memória

A forma de armazenamento de dados para esta plataforma pode se dar através de quatro tipos de mídias diferentes (SRAM, SDRAM, Flash e cartão SD). Cada uma destas mídias pode ser acessada tanto pelo processador embarcado Nios II (*softcore*) da *Altera Devices* quanto pelo painel de controle da plataforma. A configuração e especificação destas memórias são descrita a seguir:

- **SRAM:** Chip de memória com capacidade de armazenamento de 512K bytes organizados como uma matriz 256K x 16 bits;
- **SDRAM:** Chip de memória com capacidade de armazenamento de 8M bytes organizados em quatro bancos, como uma matriz de 1M x 16 bits;
- **Flash:** Memória com capacidade de armazenamento de 4M bytes que podem ser acessados via um barramento de 8 bits de dados;
- **Cartão SD:** Entrada para um cartão SD (capacidade de armazenamento dependente da capacidade do cartão SD).

5.1.3. Clocks

São disponibilizados três formas de *clocks* que podem determinar a frequência de processamento da aplicação. Dentre estas formas, dois *clocks* são fornecidos por osciladores internos, um de 27 MHz e outro de 50 MHz, e uma porta para um *clock* externo.

5.2. Linguagem de Descrição de Hardware

As linguagens de descrição de *hardware* são linguagens que, como se pode depreender do próprio nome, linguagens utilizadas para descrever a forma de interligação entre os componentes de *hardware* (portas lógicas, registradores, etc.) e seu comportamento. Dentre as diversas linguagens existem, destaca-se a linguagem SystemC, que na verdade é uma biblioteca de classes voltada para modelagem de *hardware* baseada em C/C++. Esta biblioteca permite que o projetista possa atuar em todos os níveis do processo de desenvolvimento de sistemas embarcados. A partir desta biblioteca, é possível alcançar níveis de abstração mais elevados que os conseguidos através de linguagens de descrição de *hardware* como VHDL ou Verilog.

Por se tratar de uma modelagem mais simples, optou-se pela implementação usando VHDL, pois esta Dissertação não tem como fim um modelo de abstração mais elevado posto a complexidade da arquitetura. Desta forma, os requisitos necessários para o desenvolvimento deste projeto são plenamente contemplados por esta linguagem. Outro fator importante para escolha da VHDL deve-se a sua excelência para projetos RTL, na qual esta se mostra superior a SystemC [Kojima, 2007].

5.3. Projeto em Nível de Transferência entre Registradores (RTL)

A concepção deste projeto tomou como base o modelo de referência em camadas eOSI, entretanto, a Camada de Recuperação do modelo foi abstraída por questões já discutidos na sessão 4.3.1. Neste sentido, as sessões posteriores irão abordar as implementações das camadas de Aplicação, Detecção e Redundância.

Foi implementada uma arquitetura para um sistema síncrono. Neste sentido, para todo o sistema foi concebido um único domínio de *clock*. Isso é crucialmente importante, pois evita possíveis problemas decorrentes de atrasos ou gatilhos de *clock* [Altera, 2009].

5.3.1. Camada de Aplicação

Para a Camada de Aplicação, foram desenvolvidos dois processadores simples capazes de realizar operações de soma ou subtração, de acordo com o comando que lhe é passado. Este processador possui um barramento de entrada de 20 bits e um barramento de saída de 17 bits. A forma como esta camada foi implementada não seria a mais apropriada para um dispositivo final. Alguns dos bits de entrada existem a mais para que seja possível implementar alguns casos de teste para injeção de falhas. Por este motivo, existem dois bits para indicar o comando a ser executado pelos processadores desta camada, soma ou subtração. A Camada de Comunicação é a responsável por enviar os dados para serem processados na Camada de Aplicação. A Figura 5.1ilustra o resultado da implementação para a Camada de Aplicação em RTL.

Neste RTL, percebe-se que existem duas saídas distintas para o processamento de dados, uma de cada processador e um sinal de envio de dados. As duas saídas de dados irão para a Camada de Detecção que realizará a comparação destas duas informações. O sinal de envio de dados serve para informar ao processador da Camada de Detecção que a comparação deve ser realizada. Os dois processadores são idênticos e possuem a mesma capacidade de processamento. Entretanto, para o processo de desenvolvimento de sistemas embarcados, devem ser observadas algumas restrições quanto ao caminho crítico de cada um dos processadores. Estas restrições são informadas através de ferramentas de análise temporal do próprio Quartus II[6] e cabe ao projetista utilizá-las e observar as restrições de tempo do sistema.

[6] Ambiente para desenvolvimento e simulação de aplicações para sistemas embarcados da *Altera Devices*

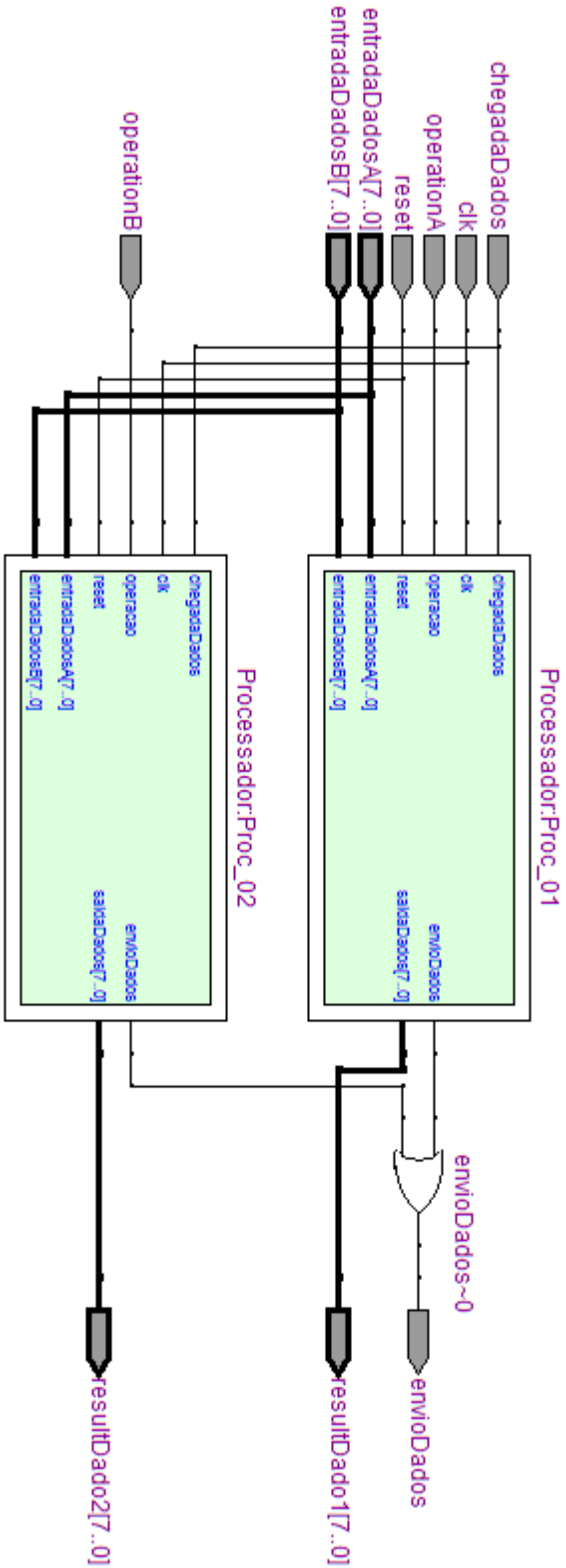

Figura 5.1: RTL da Camada de Aplicação

Não nos ateremos a detalhes minuciosos de implementação de sistemas embarcados, pois se trata de um assunto que foge ao escopo proposto por este trabalho, que no caso é criação de um modelo para desenvolvimento de sistemas tolerantes a falhas.

5.3.1.1. Máquina de Estados – Aplicação

Cada um desses processadores da Camada de Aplicação é responsável por operações simples, onde o objetivo é receber um dado da Camada de Aplicação, processá-lo e enviá-lo para Camada de Detecção. Estes processadores são idênticos é possuem uma parte operativa responsável por realizar operações de soma e subtração e uma parte de controle que comanda as atuações da parte operativa de acordo com a chegada de dados. Vejamos a seguir a máquina de estados e respectivas transições deste processador na Figura 5.2.

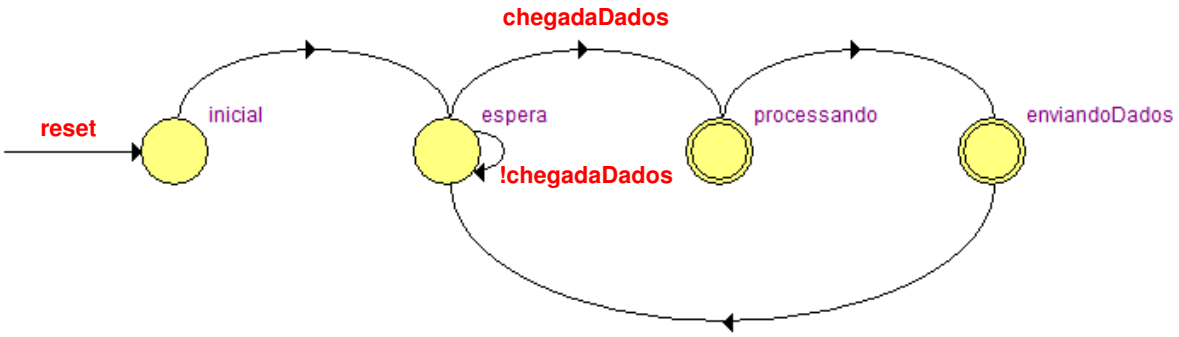

Figura 5.2: Máquina de Estados do Processador da Camada de Aplicação

Esta máquina de estados é composta por quatro estados: *inicial*, *espera*, *processando* e *enviandoDados*. Quando o sistema é reiniciado, o processador é configurado para o estado *inicial* e logo em seguida vai para o estado de espera. Neste ponto, o processador aguarda a chegada de dados da Camada de Aplicação. Enquanto estes dados não chegarem o processador permanece no estado de *espera*. O tempo de processamento é fundamental para determinação do *timeout* do temporizador da Camada de Redundância. Para esta camada, o tempo de processamento é de dois pulsos de *clock*.

5.3.2. Camada de Detecção

A Camada de Detecção é composta de uma controladora que aguarda um sinal de ativação para que a comparação seja realizada (sinal originado pela Camada de Aplicação) e um circuito que faz a comparação entre as duas entradas de dados. Com base na comparação entre essas entradas é feita a detecção de falha na Camada de Aplicação. A Figura 5.3 ilustra o resultado da implementação para a Camada de Detecção em RTL.

5.3.2.1. Máquina de Estados – Detecção

A Camada de Detecção, ao contrário da Camada de Aplicação, conta com apenas um único processador, que realiza o controle sobre o comparador e ativa a Camada de Redundância. Isto pode ser considerado um ponto crítico de falha para a arquitetura, entretanto, o objetivo é apenas provar a validade do modelo de referência eOSI.

O circuito operativo desta camada é dotado apenas de um comparador que realiza esta comparação em um único pulso de *clock*. Sendo assim, o tempo de processamento desta camada é de dois pulsos clock, um para chegada de dados e outro para o processamento destes. Esta máquina de estados é composta por quatro estados: *inicial*, *espera*, *comparação* e *envioDados*. Quando o sistema é reiniciado, o processador é configurado para o estado *inicial* (neste estado as saídas do sistema são configuradas para nível lógico ‘0’) e logo em seguida vai para o estado de espera. Neste ponto, o processador aguarda a chegada de dados da Camada de Aplicação. Enquanto estes dados não chegarem o processador permanece no estado de *espera*. Quando é sinalizada a chegada de algum dado o processador muda para o estado de *comparação*. Neste instante, o processador envia um sinal para o comparador e a comparação é realizada. Logo após, o processador muda para o estado *envioDados* que envia um sinal para a Camada de Redundância informando que a comparação já foi realizada e que esta deve ser verificada. A seguir, o processador volta para o estado de *espera* de dados.

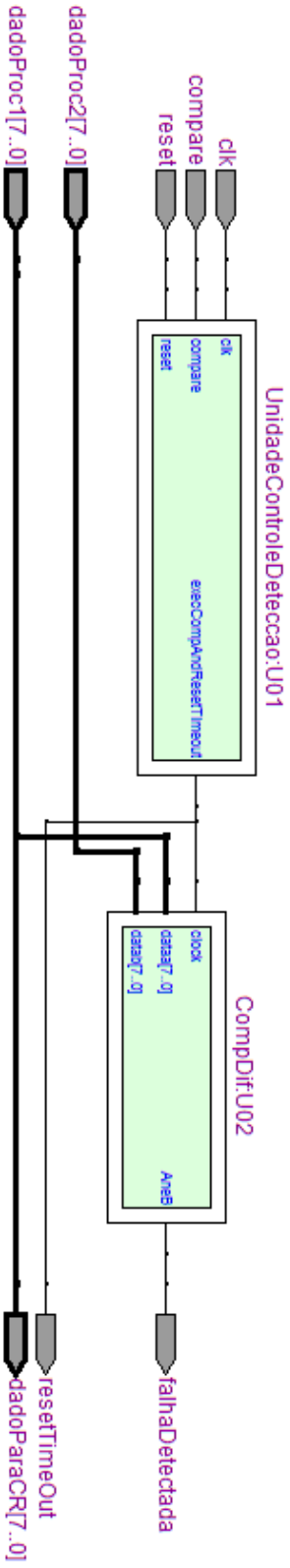

Figura 5.3: RTL da Camada de Detecção

A Figura 5.4 ilustra o modelo da máquina de estados finitos e suas respectivas transições.

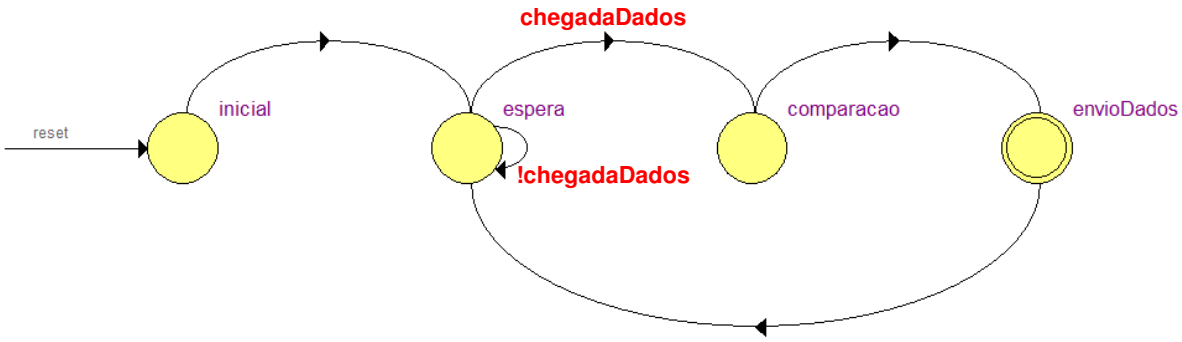

Figura 5.4: Máquina de Estados do Processador da Camada de Detecção

5.3.3. Camada de Redundância – Módulo Principal

A Camada de Redundância do Módulo Principal é constituída de um processador responsável pelo gerenciamento do fluxo de dados originário da Camada de Aplicação e Detecção, bem como todo o controle de falha. Neste sentido, conforme descrito na sessão 4.3.2, o algoritmo do módulo principal para este controle de falhas necessita de um temporizador que seja capaz verificar a ocorrência de falha devido a algum problema de conexão de barramentos entre a Camada de Redundância e as camadas superiores. Este temporizador também tem por objetivo cobrir outros tipos de falhas além da questão do barramento de comunicação, como por exemplo: uma possível falha no processador da Camada de Aplicação que o impeça de funcionar. Neste caso, a Camada de Aplicação recebe os dados para serem processados, porém estes não são encaminhados para a Camada de Detecção devido à falha em um destes processadores. Com isso, a Camada de Detecção não consegue realizar a comparação para detecção de falha, que é realizada através da divergência dos dados processados pela Camada de Aplicação. Esta mesma regra vale para o processador da Camada de Detecção. Caso este sofra alguma falha que o impeça de enviar os dados recebidos da Camada de Aplicação para a Camada de Redundância, o temporizador atuará na detecção desta falha. Neste caso, é necessário o conhecimento prévio do tempo de processamento

das camadas superiores para que se possa calcular o tempo para disparo do temporizador (*timeout*).

Para efeito de processamento digital, o tempo de processamento será calculado em termos de pulsos de *clock*. Conforme já mencionado, a quantidade de pulsos de *clocks* necessários para processamento da Camada de Aplicação é de dois pulsos de *clock,* assim como para a Camada de Detecção. Sendo assim, com base na Equação 4.1, o tempo de processamento necessário para que o temporizador desta camada dispare é o seguinte:

$$T_{timeout} = T_{c\_aplic} + T_{c\_detec}$$
$$T_{timeout} = 2 + 2$$
$$T_{timeout} = 4$$

Como forma de representação geral da Camada de Redundância do módulo principal, a Figura 5.6 ilustra essa representação em RTL.

5.3.3.1. Máquina de Estados – Módulo Principal

A máquina de estados desta camada é composta por cinco estados, a saber: *inicial*, *espera*, *falha*, *envio* e *erro*. Cada um destes estados tem por objetivo traduzir para uma linguagem de *hardware* o algoritmo apresentado na sub-sessão 4.3.2. A Figura 5.5 ilustra o modelo da máquina de estados finitos e suas respectivas transições.

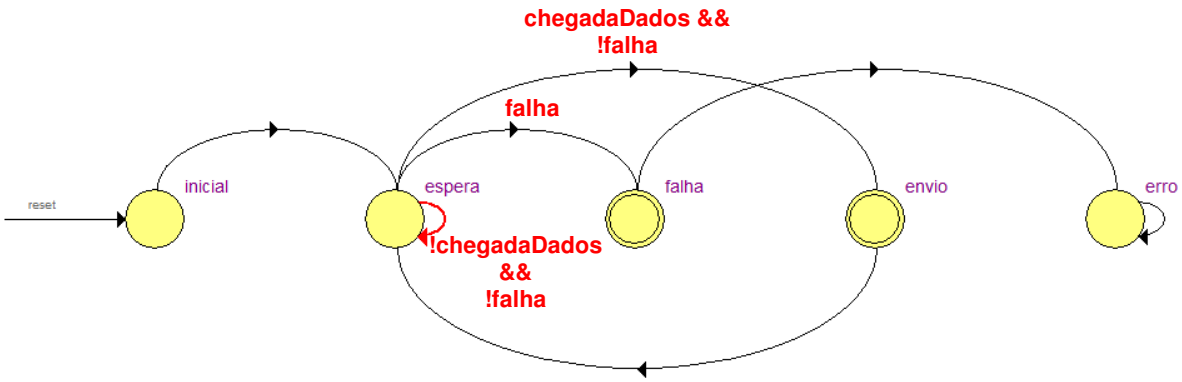

Figura 5.5: Máquina de Estados do Processador da Camada de Redundância – Módulo Principal

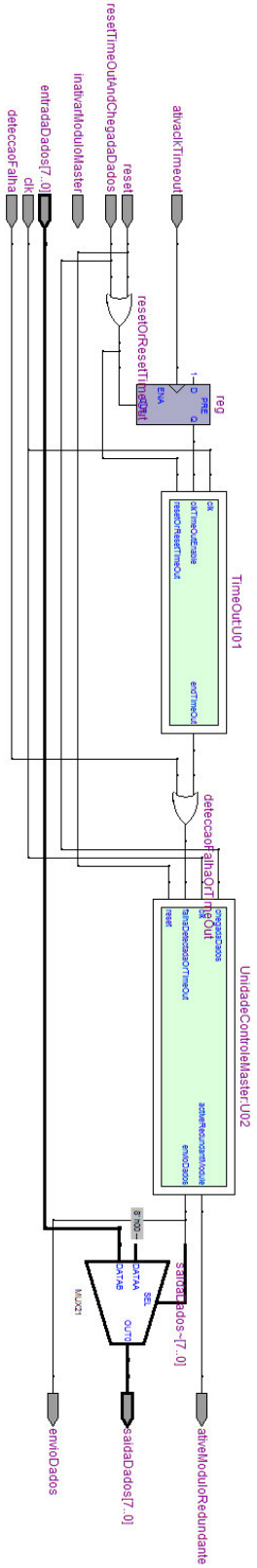

Figura 5.6: RTL da Camada de Redundância – Módulo Principal

Para a máquina de estados da Figura 5.5 quando o sistema é reiniciado, o processador é configurado para o estado *inicial* e logo em seguida vai para o estado de espera. O processador permanece no estado de *espera* enquanto não chegar nenhum sinal de chegada de dados ou um sinal de falha. Os sinais de chegada de dados e de falha são gerados pela Camada de Detecção, entretanto um sinal de falha pode ser gerado antes de um sinal de chegada de dados. O sinal de falha, não necessariamente é enviado pela Camada de Detecção, mas também pelo temporizador da Camada de Redundância. Quando um sinal de chegada de dados é recebido pelo processador e nenhum sinal de falha é gerado, então este passa para o estado de envio, onde os dados processados são enviados para a Camada de Comunicação e um sinal de bloqueio é enviado para a Camada de Redundância do módulo redundante. Se um sinal de falha é recebido, então o processador passa para o estado de *falha*. No estado de *falha*, o processador envia um sinal de ativação para o módulo redundante que passará a ser o módulo atuante, caso este também não tenha sofrido alguma falha. Logo após o envio do sinal de ativação para o módulo redundante, o processador passa para o estado de *erro* e permanecerá neste não podendo atuar até que o mesmo seja reiniciado.

5.3.4. Camada de Redundância – Módulo Secundário

O que difere o módulo principal do módulo secundário é a Camada de Redundância. Nesta camada, não somente os processadores são diferentes, mas também existe um temporizador a mais para a Camada de Redundância do módulo secundário. Este segundo temporizador é utilizado para prevenir possíveis falhas no módulo principal que o impeçam de comunicar-se com o módulo secundário. O princípio de atuação desta camada é baseado no algoritmo do módulo redundante apresentado na sessão 4.3.2. Neste sentido, a funcionalidade desta camada é idêntica a do módulo principal, ou seja, controlar o fluxo de dados para a Camada de Comunicação e ativar um módulo excedente, caso ocorra uma falha.

Além de um temporizado extra, esta camada é dotada de quatro registradores que servem para armazenar informações referentes ao processamento de dados, chegada de dados ou sinalização de falhas ocorridas em alguma das camadas

superiores. Como este módulo secundário necessita que o temporizador atinja o seu limite ou o módulo principal envie-lhe um sinal de ativação para ser ativado, o envio do sinal de chegada de dados da Camada de Detecção pode ser perdido. Neste sentido, esta informação precisa ser armazenada para que quando o sinal de ativação seja originado pelo temporizador a informação de chegada de dados não seja perdida. Se esta informação não fosse armazenada seria gerado um falso negativo para falha, pois como o sinal do temporizador de ativação sempre é gerado posteriormente ao sinal de chegada de dados, o sistema iria entender que o módulo secundário também teria sofrido uma falha. Analogamente ao sinal de chegada de dados, o registro do resultado do processamento também precisa ser armazenado para posterior verificação de falha. Desta forma, caso o módulo seja ativado pelo disparo do temporizador, o mesmo só enviará a informação para a Camada de Comunicação se nenhuma falha tiver acontecido. A Figura 5.8 ilustra a representação RTL da Camada de Redundância do módulo secundário. Os outros dois registradores servem para ativar a contagem dos temporizadores.

5.3.4.1. Máquina de Estados – Módulo Secundário

A máquina de estados desta camada, para o módulo secundário, é composta por seis estados, a saber: *inicial*, *inativo*, *espera*, *falha*, *envio* e *erro*. Estes estados representam em uma máquina de estado o algoritmo apresentado na sessão 4.3.2. A Figura 5.7 ilustra o modelo da máquina de estados finitos e suas respectivas transições.

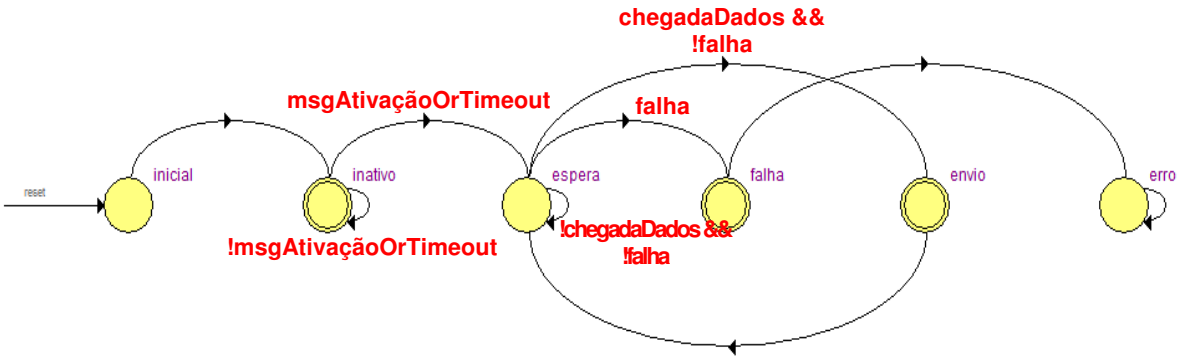

Figura 5.7: Máquina de Estados do Processador da Camada de Redundância – Módulo Secundário

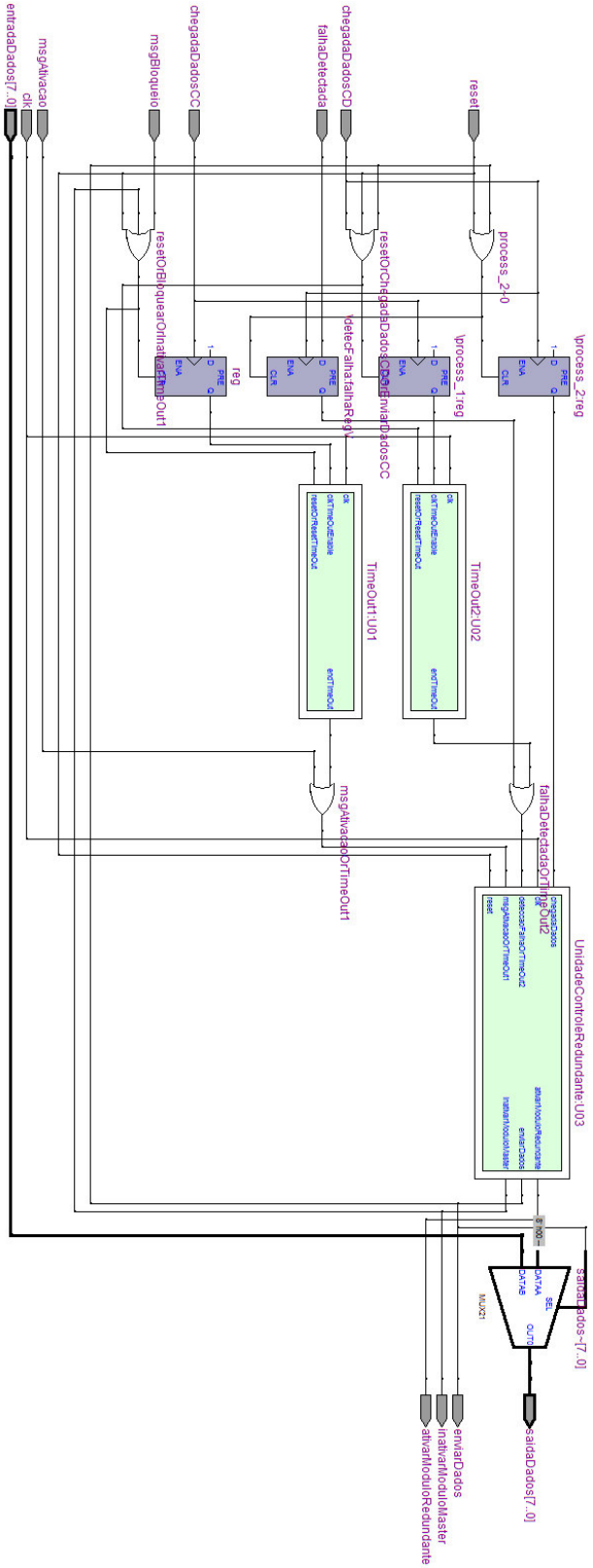

Figura 5.8: RTL da Camada de Redundância – Módulo Secundário

Para a máquina de estados da Figura 5.7, o estado *inicial* é primeiro estado deste processador que é setado ao ser reiniciado. Em seguida, o processador passa para o estado *inativo* e permanece neste até que um sinal de ativação ou *timeout* sejam disparados. O estado *inativo* é o que distingue esta máquina de estados do módulo secundário da mesma no módulo principal. Este estado *inativo* é o que impede que haja conflitos e colisões na escrita no barramento de dados, ou seja, para que o módulo secundário possa escrever sobre o barramento de dados é necessário que este não esteja no estado *inativo*. Os demais estados e lógica são iguais à máquina de estados apresentado na sessão 5.3.3.

5.3.5. Arquitetura Geral

O objetivo da arquitetura de suporte desenvolvida é introduzir o menor custo de processamento e aumentar o grau de dependabilidade do sistema. Desta forma, os atores envolvidos neste processo devem ter participação e interferência mínina, todavia, quando necessários devem atuar de forma efetiva e sem prejuízos para a aplicação final. A arquitetura que é apresentada neste trabalho retrata a implementação em RTL da junção das três camadas do modelo de referência eOSI: Aplicação, Detecção e Redundância. Neste sentido, a Figura 5.9 ilustra esta representação.

5.4. Casos de Teste

Os casos testes foram desenvolvidos tendo como premissa as possíveis falhas previstas para serem tratadas pela arquitetura de suporte. Deve-se levar em consideração que nenhum sistema está livre da ocorrência de falha e que por mais amplo que seja o espectro de cobertura dos casos de testes a construção de um sistema totalmente infalível é inconcebível [Anderson & Lee, 1981]. Neste sentido, este sistema tem por objetivo o tratamento de falhas físicas e lógicas, entretanto, casos os dois módulos sofram a mesma falha o sistema não será capaz de realizar o correto tratamento.

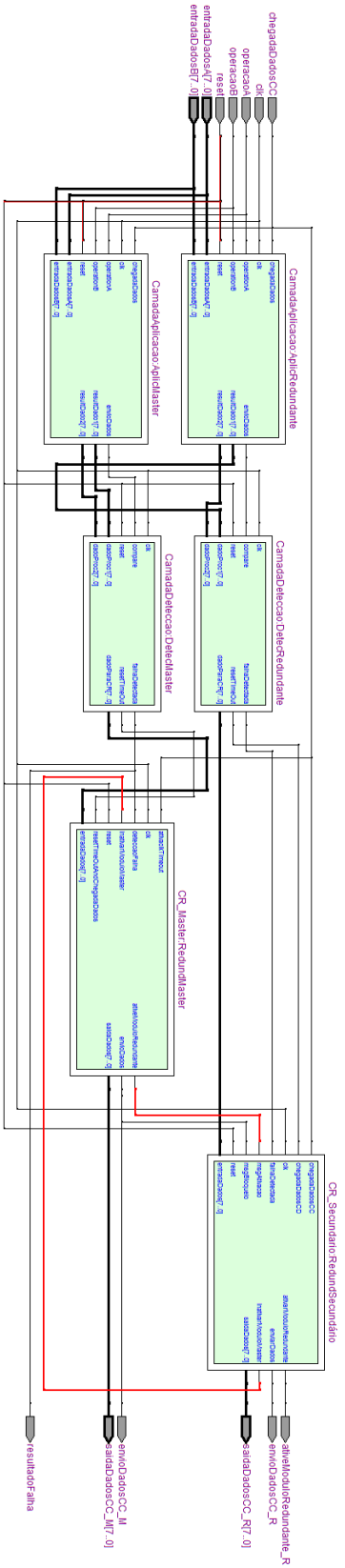

Figura 5.9: RTL da Arquitetura Suporte Geral

Como exemplos dos possíveis casos de falhas que podem ser tratados pela arquitetura de suporte destacam-se: falhas físicas e lógicas da Camada de Aplicação; falhas físicas da Camada de Detecção; falhas físicas e lógicas da Camada de Redundância. Detalhes sobre a transição de estados das camadas superiores foram abstraídos a fim de tornar a leitura do gráfico mais fácil e focada no ponto chave da arquitetura, que é a Camada de Redundância de ambos os módulos. As sessões seguintes apresentam os resultados dos casos de testes realizados para a arquitetura.

5.4.1. Primeiro Teste – Fluxo de Atuação Normal

Para o primeiro caso de teste é apresentado o fluxo de operação normal sem a ocorrência de falha. A Figura 5.10 ilustra este fluxo de operação através de sinais de chegada de dados, sinais de envio de dados, da transição de estados dos processadores da Camada de Redundância e dos eventos enumerados nesta figura. Neste gráfico é ilustrada a chegada de dados em dois instantes, o primeiro acontece aos 60 ns e o segundo aos 150 ns. Os sinais *chegadaDadosCD_M* e *chegadaDadosCD_R* representam o sinal de chegada de dados da Camada de Detecção para o módulo principal e secundário respectivamente através de um pulso positivo. Na Figura 5.10 percebemos que os dados chegam ao mesmo instante nas respectivas Camadas de Redundância, como era de se esperar, pois se trata de um sistema síncrono. O sinal *envioDadosCC_M* significa que o módulo principal enviou os dados processados para a Camada de Comunicação e, como o módulo secundário não foi ativado o mesmo não enviou dado algum para a Camada de Comunicação.

Ainda com relação ao gráfico da Figura 5.10, é possível perceber que os temporizadores não atingem o seu limite máximo de contagem e, por isso mesmo, estes não ativam os processadores das respectivas camadas. Com relação aos sinais de falha, nenhum dos módulos receberam esta sinalização o que pode ser verificado por meio dos sinais *falhaDetectadaCD_M* e *falhaDetectadaCD_M*. As máquinas de estados do módulo principal e secundário são representadas por *CR_Master* e *CR_Secundário* respectivamente, bem como seus estados que foram

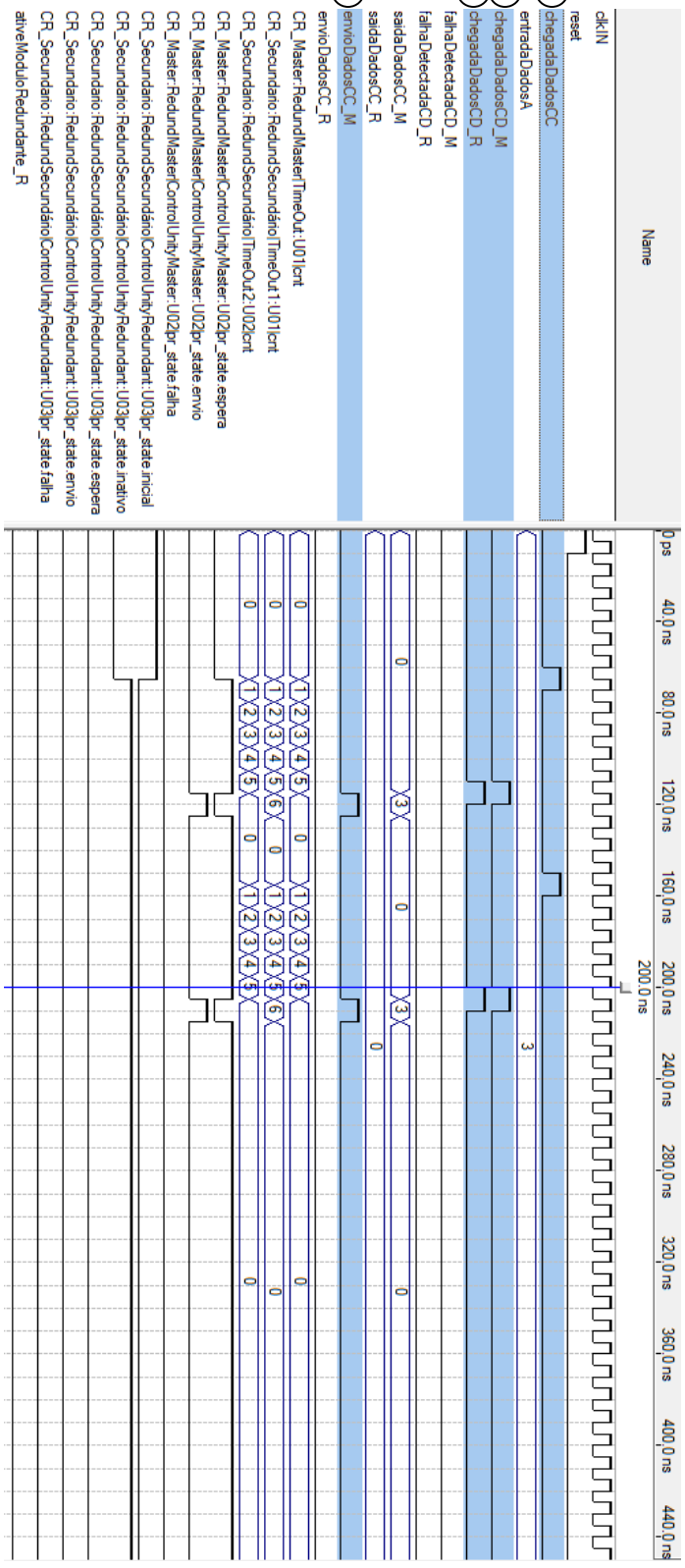

Figura 5.10: Fluxo normal de operação

apresentados nas sessões 5.3.3 e 5.3.4. Neste sentido, podemos observar a transição de estado de *espera* para *envio* na chegada de dados em *CR_Master* e a permanência no estado *inativo* em *CR_Secundário*. Estas mesmas observações a cerca dos estados e máquinas de estados serão adotadas para os casos de teste que se seguem.

5.4.2. Segundo Teste – Camada de Aplicação

Para este segundo teste, foi feita a simulação de uma falha ocorrida na Camada de Aplicação e detectada pela Camada de Detecção. Neste exemplo, temos que um sinal de falha detectada é enviado pela Camada de Detecção para a Camada de Redundância. No caso, este sinal é enviado juntamente com o sinal de chegada de dados. O gráfico da Figura 5.11 ilustra as transições de estados e comportamento do módulo principal ao detectar uma falha. A falha é detectada aos 110 ns e em seguida o processador do módulo principal vai para o estado de *falha*. Neste estado, o processador envia um sinal de ativação para o módulo secundário que muda seu estado de *inativo* para *espera*.

Esta transição para o estado de *espera* acontece para que seja possível a verificação de falha no módulo secundário. Podemos notar também que, como o módulo principal entrou em erro, este por sua vez deixa de escrever sobre o barramento de dados através do envio de dados para a Camada de Comunicação. Desta forma, mesmo que a Camada de Comunicação não tenha sofrido nenhuma falha, o módulo principal não enviará nenhum dado, pois estes não são encaminhados para a camada responsável pelo envio de dados. Sendo assim, somente o módulo secundário fica responsável pelo envio de dados, o que pode ser observado através do sinal *envioDadosCC_R* na Figura 5.11. Ainda no que se refere a este caso de teste, podemos observar que somente o módulo secundário envia os dados para a Camada de Comunicação através do sinal *envioDadosCC_R*. Este evento acontece aos 135 e 205 ns respectivamente. Outro importante evento que pode ser verificado no gráfico da Figura 5.11 é a sequência de transições de estados que ocorrem no processador do módulo secundário. A transição do estado *inativo*

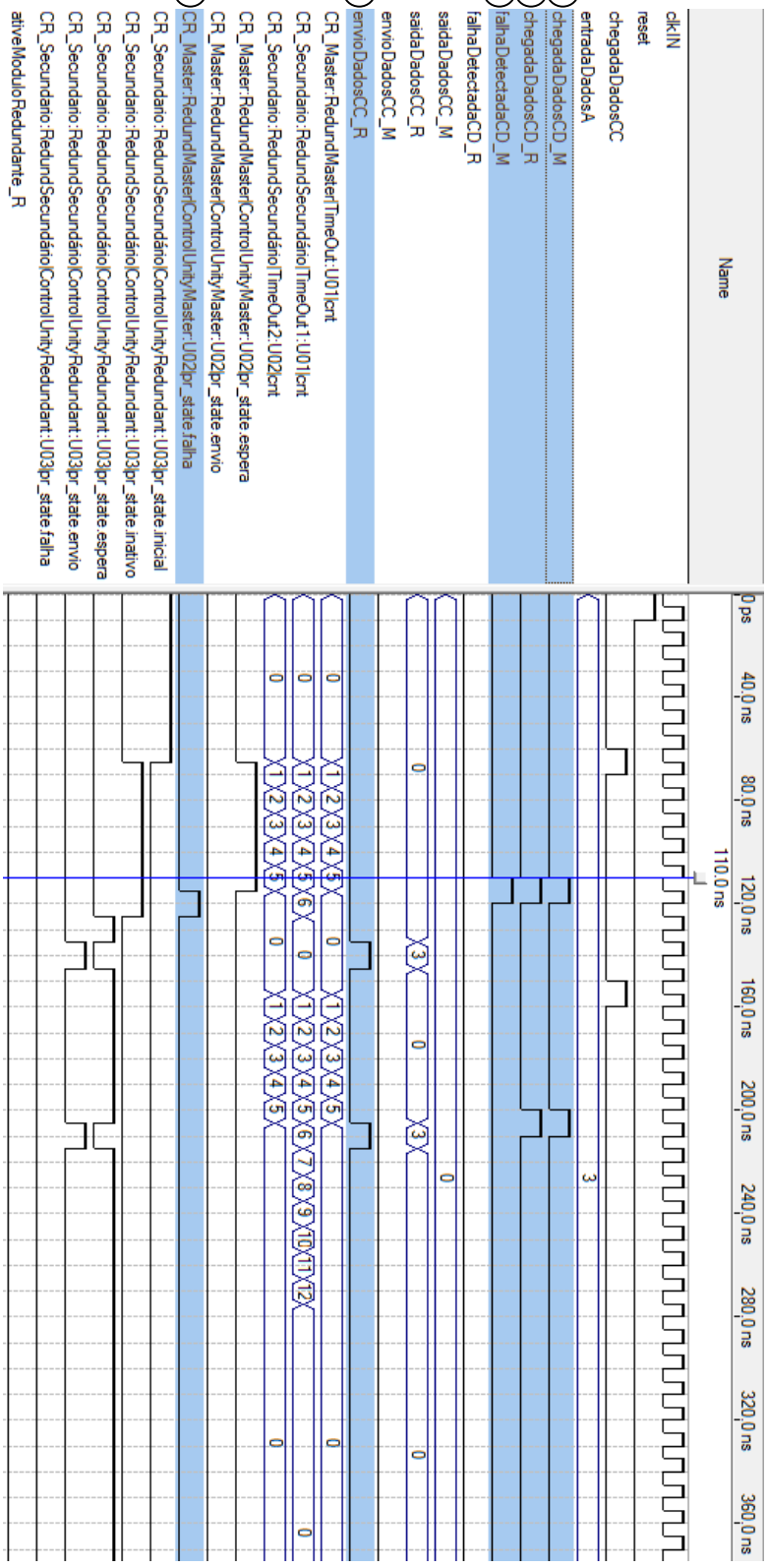

Figura 5.11: Falha detectada pela Camada de Detecção

para o estado de *espera* ocorre logo após o módulo principal ativar o estado de *falha*, o que indica o sincronismo entre os módulos.

5.4.3. Terceiro Teste – Camada de Aplicação ou Detecção

O terceiro caso de teste simula uma falha de comunicação entre o processador da Camada de Redundância e os demais processadores das camadas superiores, ou seja, o tempo estimado para processamento atingiu o limite máximo e nenhum dado fora recebido pela Camada de Redundância do módulo principal. Isto pode significar a ocorrência de diversas falhas, como por exemplo: falha de comunicação entre a Camada de Aplicação e a Camada de Detecção; falha de comunicação entre a Camada de Detecção e a camada de Redundância; falha nos dispositivos da Camada de Aplicação, o que pode impossibilitar o envio de dados para a Camada de Detecção; falha nos dispositivos da Camada de Detecção, o que pode impossibilitar o envio de dados para a Camada de Redundância.

Neste caso, o que ativa o módulo secundário é o temporizador do módulo principal. O temporizador atinge a sua contagem limite e neste instante o processador percebe que alguma falha aconteceu e muda seu estado de *espera* para o estado de *falha*. Neste mesmo instante, o processador envia um sinal de ativação para o módulo secundário que por sua vez entra em execução e envia os dados processados de forma transparente para a aplicação que os aguarda. A máquina de estado da Figura 5.12 ilustra as transições de estado.

A Figura 5.12 representa o gráfico que demonstra o comportamento do sistema frente à situação descrita. Podemos observar que o temporizador do módulo principal (*TimeOut*) atinge o seu limite e faz com o processador da Camada de Redundância vá para o estado de *falha*. Em seguida o módulo secundário é ativado e saindo do estado *inativo* e passando ao estado de *espera*.

Podemos observar também que o temporizador *TimeOut1* do módulo secundário, não tem efeito algum sobre este módulo, pois o sistema já fora ativado pelo módulo principal. A única funcionalidade deste temporizador é para o caso de falha no módulo principal e não envio de um sinal de ativação.

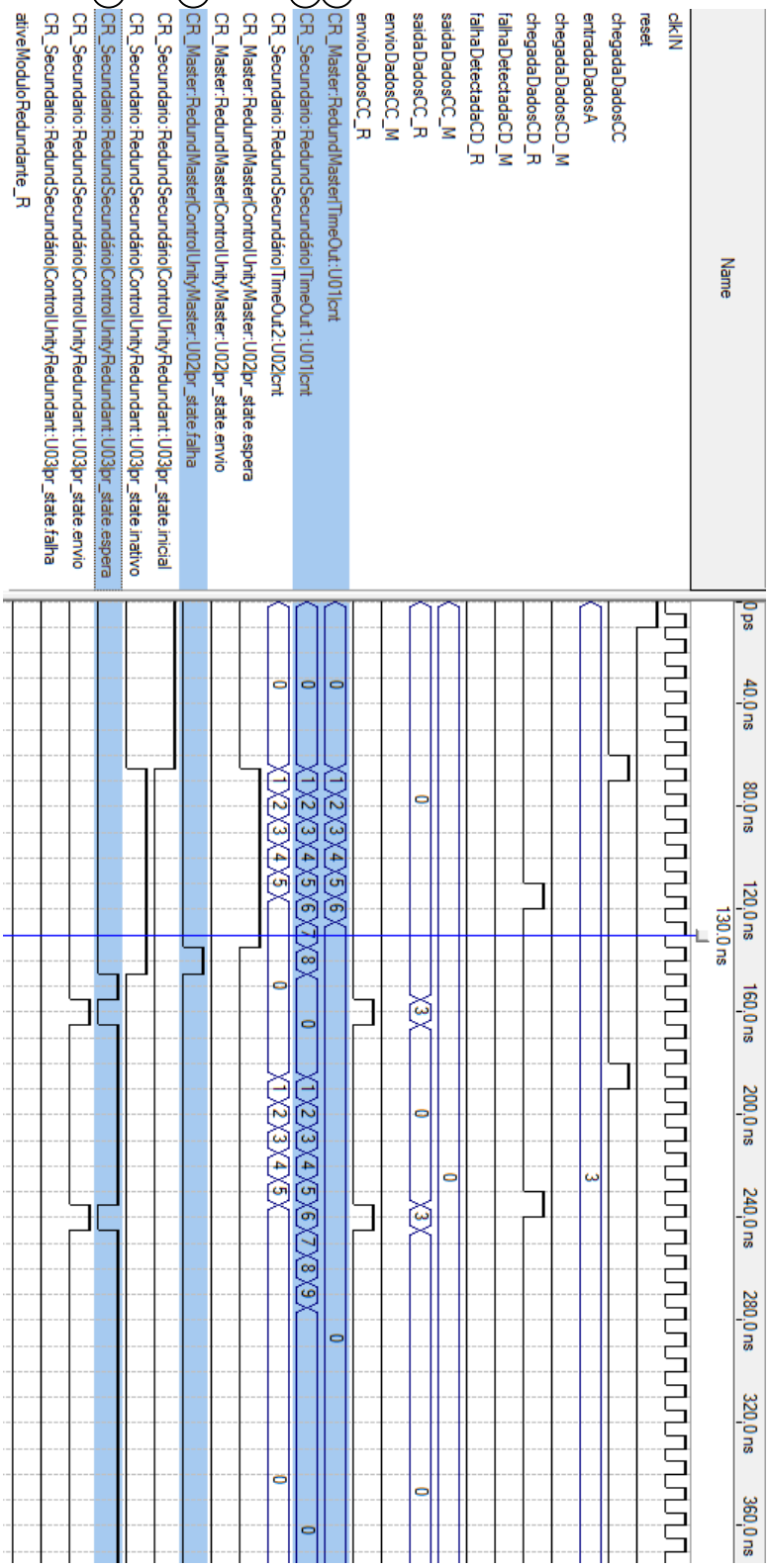

Figura 5.12: Falha de não chegada de dados para Camada de Redundância

Vale ressaltar que os tempos para ativação devem ser maiores que os calculados pelas equações 4-1 e 4-2, pois devemos levar em conta os atrasos decorrentes das “trocas de mensagens”, que neste caso acontece via sinais de ativação e o próprio atraso inerente ao circuito. Portanto, foram acrescentados dois pulsos de *clock* para os temporizadores. No caso do temporizador *TimeOut1* o seu limite de tempo é ainda maior pois prevê uma margem de erro para comunicação com o módulo principal.

5.4.4. Quarto Teste – Camada de Redundância Master

O quarto caso de teste avalia o comportamento do sistema frente à ocorrência de falha na Camada de Redundância do módulo principal, ou seja, o módulo que se encontra em operação. A Camada de Redundância é responsável pelo controle do fluxo de dados para a Camada de Comunicação e pelo controle e ativação do módulo secundário. Neste caso de teste, a falha acontece devido a não resposta para a chegada de dados proveniente da Camada de Detecção do módulo principal. Esta falha pode ter como causa dois motivos: falha no temporizador da Camada de Redundância do módulo principal ou falha no processador da Camada de Redundância. Neste caso, independentemente de qual falha tenha ocorrido, o módulo secundário será ativado pelo temporizador *Timeout1*, posto que o tempo para envio do dado atingiu o limite máximo de tempo e o módulo principal não enviou dado algum para a Camada de Comunicação, após a chegada de dados. A partir deste instante, o módulo secundário entra em ativação e verifica o resultado do processamento das camadas superiores para que enviá-lo para a Camada de Comunicação ou bloqueio.

O gráfico da Figura 5.13 ilustra a sequência de acontecimentos que são representados pelas transições de estados das máquinas do módulo principal e secundário, além dos sinais de dados de chegada de dados. Observemos também as transições ocorrentes na máquina de estado do módulo secundário e o *Timeout1*.

Analisando detalhadamente a Figura 5.13 podemos perceber que mesmo com a ocorrência da falha no módulo principal, o mesmo só passa para o estado de falha após a chegada da segunda massa de dados. Isto permite inferir que foi necessária a ocorrência de duas falhas para que o módulo principal passasse para o estado de

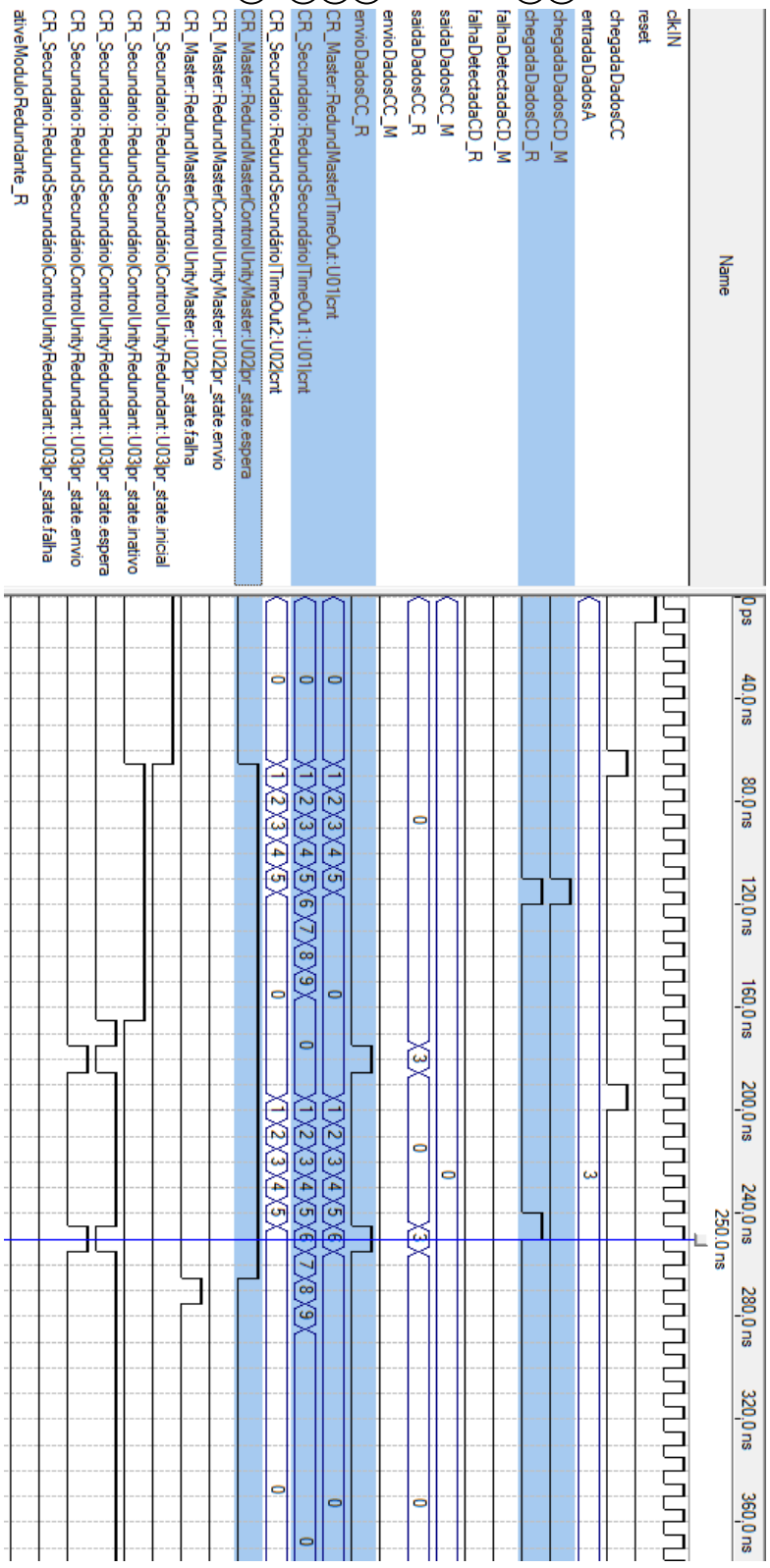

Figura 5.13: Falha na Camada de Redundância do módulo principal

falha, entretanto, a ocorrência de uma ou outra faz com o conjunto do sistema (módulo principal e módulo secundário) perceba e tome as ações adequadas como forma de tratar estas falhas. Sendo assim, o que de fato aconteceu nesta simulação foi primeiramente uma falha no barramento que leva o sinal de chegada de dados da Camada de Detecção para o processador da Camada de Redundância, fazendo com este permanecesse no estado de espera enquanto que o temporizador fosse resetado. No segundo momento, por volta dos 230 ns, o que aconteceu foi a falha no barramento de chegada de dados da Camada de Detecção como um todo, ou seja, nem o processador nem o temporizador foram informados sobre a chegada de dados. Desta forma, no instante em que o temporizador *Timeout* (módulo principal) atinge a contagem de seis pulsos de *clock* o processador da Camada de Redundância do módulo principal recebe um sinal informando a falha de não chegada de dados da Camada de Detecção.

5.4.5. Quinto Teste – Camada de Redundância Secundária

Para este quinto e último caso de teste, a situação de falha é explorada sobre o módulo secundário. Para os demais casos de teste, todas as suposições anteriores se aplicam para o módulo secundário. Neste exemplo, a falha acontece no barramento de chegada de dados da Camada de Detecção para a Camada de Redundância. Para que o par do sistema, módulo principal e módulo secundário, atue com maior confiabilidade e disponibilidade é necessário que na percepção de falha de algum destes, o mesmo seja substituído. Independentemente de qual módulo falhe, o módulo que irá substituir o defeituoso deverá ser do tipo secundário, posto que este, inicialmente, deverá permanecer em estado de inativação.

A Figura 5.14 ilustra um exemplo com a sequência de acontecimentos e o comportamento do sistema de acordo com o surgimento das falhas. Neste exemplo da, podemos perceber uma primeira falha que ocorre no módulo principal fazendo com que o módulo secundário seja ativado. No momento da segunda chegada de dados da Camada de Comunicação (*chegadaDadosCC*) os temporizadores são ativados, porém ocorre uma falha em uma das camadas superiores que impede que os dados sejam corretamente enviados para a Camada de Redundância. Devido a

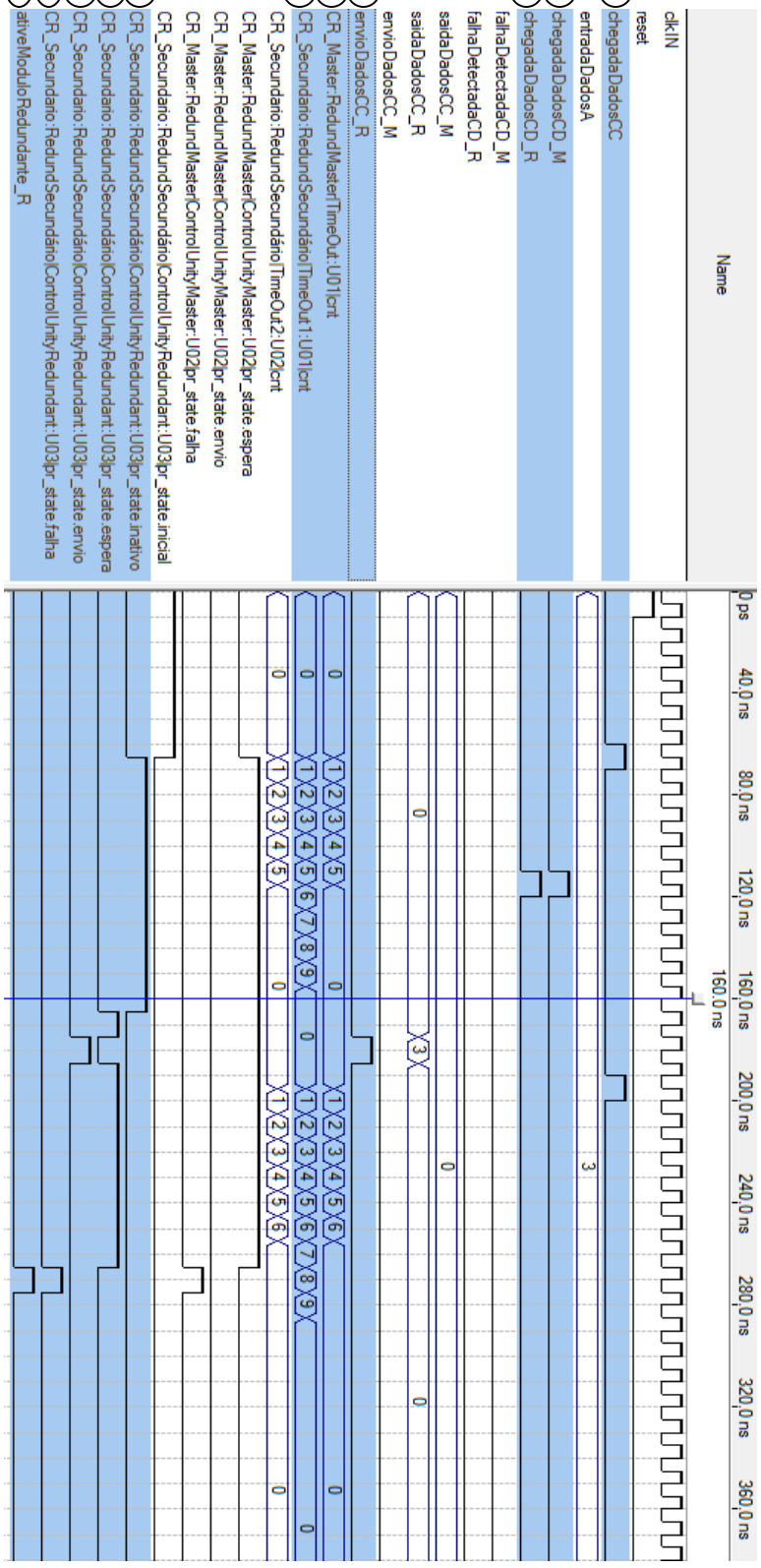

Figura 5.14: Falha pela Camada de Redundância do módulo secundário

isto, o temporizador *Timeout2* da Camada de Redundância do módulo secundário atinge o limite máximo de contagem fazendo com o processador passe para o estado de *falha*. Com isto, o módulo secundário envia um sinal de ativação (*ativeModuloRedundante_R*) para que o substituto do módulo principal seja ativado.

Para o estudo de caso proposto, estes casos de teste demonstram o espectro de cobertura de falha as quais podem ser tratadas pelo sistema deste estudo de caso.

Sessão 6

6. Conclusões

Nesta Sessão é apresentada a avaliação final sobre os resultados obtidos com este trabalho de Dissertação, bem como futuros trabalhos de pesquisa que podem ser gerados e continuados a partir deste. Apresentaremos também as consequências resultantes da implementação e emprego desta metodologia de desenvolvimento de sistemas tolerantes a falha.

6.1. Considerações Finais

Os avanços tecnológicos e incentivos a pesquisas para desenvolvimento de sistemas embarcados é uma crescente que se destaca ao longo das últimas décadas. As metodologias e arquiteturas de desenvolvimento para este tipo de aplicação dividem opiniões e, de certo, cada qual possui seu espaço, tanto para segmentos da indústria como para pesquisas acadêmicas. Esta gana por novas e melhoradas arquiteturas e sistemas deve-se principalmente à ampliação do espectro de utilização para sistemas embarcados. Os ambientes aos quais estes sistemas são expostos e suas aplicabilidades cresceram tanto quanto as inovações na forma de concepção destes dispositivos. Neste sentido, aplicações que estão associadas a condições que envolvem risco à saúde, ao meio ambiente e a sociedade devem dispor de um grau de confiabilidade e segurança compatível com o contexto de atuação no qual estes sistemas estão inseridos.

Consoante a estes fatos, as tecnologias e mecanismos para tratamento de falhas que ocorrem em sistemas mostraram-se como uma alternativa para melhoria do grau de dependabilidade de uma determinada planta. Desta forma, as técnicas e mecanismos de tolerância a falhas apresentados nesta Dissertação comprovaram ser bastante eficazes na resolução destes problemas. Todavia, estas ações para melhoria do grau de dependabilidade do sistema são tomadas de forma a parte. Nesta Dissertação, apresentamos uma nova visão para implantação, avaliação e

escolhas de um conjunto de técnicas que podem ser utilizadas para a criação de um projeto de sistemas embarcados tolerantes a falhas. O modelo de referência eOSI mostrou que pode ser possível o desenvolvimento de um sistema voltado para o tratamento de falhas sem que para isso seja necessário alterações sobre a aplicação final. O modelo de referência eOSI torna transparente para a aplicação o controle do fluxo de dados e a recuperação em situações de ocorrência de falhas.

A separação das técnicas de tolerância a falhas em camadas permite que o projetista tenha um maior grau de liberdade para a escolha da técnica e mecanismo de tolerância a falha. Isto confere ao projetista a capacidade de desenvolver o sistema segundo as necessidades que lhe são requisitadas. Outra vantagem que pode ser percebida com a utilização do modelo eOSI diz respeito a possibilidade de utilização de técnicas de mascaramento e recuperação de falhas em um mesmo sistema. As aplicações que podem ser desenvolvidas com base neste modelo de referência são muitas, bem como a forma de combinação de técnicas para alcance do objetivo comum para tratamento e recuperação de falhas. Desta forma, muitos outros modelos de arquiteturas podem ser explorados, desenvolvido e avaliados tendo como ponto de partir o modelo de referência eOSI.

O modelo de arquitetura de suporte apresentado nesta Dissertação apresentou resultados satisfatórios quanto à recuperação e atuação no tratamento e confinamento de falhas. Toda a arquitetura foi desenvolvida tendo como o base o modelo de referência eOSI que, neste projeto, serviu de guia para escolha e implementação das técnicas de tolerância a falhas apresentadas na Sessão 2. O objetivo deste modelo não é otimizar o processo de especificação e design de sistemas embarcado, entretanto, o modelo eOSI contribui de forma indireta para a melhoria destes fatores. Isto pode ser constatado através aspectos inerentes a capacidade humana de abstração que é alcançada com base neste modelo. Esta melhoria na forma de visualização da concepção do sistema permite que o projetista tenha uma maior facilidade para identificação e isolação de pontos de falhas o que contribui para o aperfeiçoamento do potencial de dependabilidade do sistema.

Os resultados dos testes demonstraram que é possível desenvolver um sistema segundo a modelagem sugerida pelo modelo eOSI, porém é possível

também criar um sistema sem que todas partes do modelo sejam implementadas, como foi o caso da arquitetura apresentada. Esta característica confere ao modelo um poder de flexibilidade que deverá ser explorado pelo projetista e equipe de desenvolvedores, ou seja, cabe a estes a escolha de quais camadas implementar de acordo com as necessidades e exigências do projeto.

Depreende-se desta implementação que a construção de um sistema distribuído para o controle, gerenciamento e tratamento de falha pode ser utilizado como uma abordagem eficaz para solução de problemas relacionados a falhas de *hardware*. Através do estudo de caso, o sistema comprovou a validade do modelo por meio de simulações e injeção de falhas que possivelmente podem ocorrer em um sistema real. Desta forma, para todos os casos de testes apresentados o sistema comportou-se da maneira esperada e os resultados dos testes evidenciaram esta validade. As camadas que comprovam o tratamento de falhas para o modelo eOSI estão concentradas na Camada de Recuperação (abstraída no estudo de caso apresentado por motivos de ganhos em desempenho), na Camada de Detecção e na Camada de Redundância. Os testes apresentados exploraram casos de falhas (lógica e física) que pudessem acontecer em todas as camadas superiores a Camada de Comunicação e, para a arquitetura proposta, o sistema foi capaz de atuar de forma a minimizar os efeitos provocados pelo surgimento destas falhas.

6.2. Trabalhos Futuros

No que tange o âmbito de implementação e desempenho do modelo de referência eOSI, frente às práticas de projetos que são utilizadas, sugere-se como um trabalho de pesquisa a avaliação comparativa dos ganhos proporcionados à um projeto real de desenvolvimento de sistemas embarcados. Neste caso, sugere-se que sejam feitas avaliações de tempo de desenvolvimento, integração entre equipes, e até mesmo avaliações subjetivas a cerca da compreensão e entendimento do sistema com esta abordagem. Entretanto, a aplicação do modelo não está restrita a projetos desta natureza e a mesma avaliação comparativa de desempenho em termos de retrabalho, manutenção e tempo de desenvolvimento podem ser

observados em sistemas que necessitem do emprego de técnicas de tolerância a falha e que podem fazer uso deste modelo.

Trabalhos de implementação que façam uso de todas as camadas do modelo destaca-se por sua importância e é sugerido como um trabalho futuro de Dissertação, ou continuação da arquitetura que foi apresentada nesta Dissertação. Neste contexto, podemos ainda destacar como possíveis trabalhos futuros os seguintes:

- Implementação da Camada de Comunicação e inserção do protocolo de comunicação PM-AH apresentado em [Valentim, 2008]; neste caso torna-se viável a avaliação do impacto que é causado pela implementação desta arquitetura de suporte sobre o desempenho que é esperado para a rede de automação hospitalar;
- Implementação e análise comportamental de modelos de arquiteturas mais complexos para as Camadas de Detecção e Recuperação. Como exemplo para estas novas implementações temos: testes com sistemas de tempo real, aplicações baseada em redes neurais, sistemas especialistas e sistemas preditivos que podem ser desenvolvidos. Em casos onde se deseja obter menor impacto sobre o tempo de resposta devido a arquitetura de suporte, torna-se mister analisar os resultados da implementação de uma rede neural para a Camada de Recuperação.
- Avaliação de desempenho do sistema e os déficits causados pela implementação da arquitetura de suporte em sistemas que já estejam em operação; Neste caso, pode ser feita uma avaliação comparativa com outras abordagens que tenham sido implementadas para sistemas em operação.

É fato que as tecnologias para tratamento e confinamento de falhas são objetos de estudo da comunidade acadêmica e por vezes aplicada em sistemas industriais. Todavia, a história recente nos mostra que a evolução das formas de concepções e desenvolvimento destes sistemas os torna, de um modo geral, cada

vez mais confiáveis e seguros. A busca por um sistema perfeito e totalmente tolerante a falha ainda não foi possível, pois a própria matéria sofre desgaste e não é indestrutível, o que nos leva ao desenvolvimento de sistemas redundantes. Os sistemas redundantes, atualmente, mostram-se como uma boa alternativa em termos de melhoria dos requisitos de dependabilidade e esta Dissertação teve como principal objetivo a apresentação de um novo modelo para desenvolvimento de sistemas que se baseia nas práticas para tratamento de falhas já consolidados na literatura. Sendo assim, os objetivos validados por meio do desenvolvimento de um estudo de caso comprovam a eficácia do modelo e testifica o seu emprego.

Referências Bibliográficas

Bibliográfia

[1] Abd-El-Barr, M. “Design And Analysis of Reliable And Fault-tolerant Computer Systems”. Ed. Imperial College Press, 2007.

[2] Anderson, T.; Lee, P. A. “Fault tolerance - principles and practice”. Englewood Cliffs, Prentice-Hall, 1981.

[3] ANSI/ISA–95.00.01–2000. Enterprise-Control System Integration Part I: Models and Terminology. ISA - The Instrumentation, Systems, and Automation Society. Julho de 2000.

[4] Avizienis, A. “Design of Fault-Tolerant Computers”. *Proc. 1967 Fall Joint Computer Conf., AFIPS Conf. Proc.*, Vol. 31, pp. 733-743, Thompson Books, Washington, D.C., 1967.

[5] Avizienis, A.; Kopetz, H.; Laprie, J.-C. “The Evolution of Fault-Tolerant Computing”. Ed. Springer-Verlag, New York, 1987.

[6] Avizienis, A. “Toward systematic design of fault-tolerant systems,” *IEEE Computer*, vol. 30, no. 4, pp. 51-58, Abril de 1997.

[7] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. “Basic Concepts and Taxonomy of Dependable and Secure Computing”. *IEEE Transactions on Dependable and Secure Computing*. Vol. 1, no. 1, pp. 11-33, Janeiro-Março de 2004.

[8] Bartocci, E.; Cacciagrano, D.; Cannata, N.; Corradini, F.; Merelli, E.; Milanesi, L.; Romano, P. “An Agent-Based Multilayer Architecture for Bioinformatics Grids”. *IEEE Transactions on Nanobioscience*. Vol. 6, no 2, Junho de 2007.

[9] Brown, S.; Rose, J. “FPGA and CPLD Architectures: A Tutorial”. *IEEE Design and Test of Computers*. Vol. 13, No. 2, pp. 42-57, 1996.

[10] Chang, Y.-S.; Park, B.-II; Park, I.-C.; Kyung, C.-M. “Customization of a CISC processor core for low-power applications”. *International Conference on Computer Design – ICCD*. pp. 152-157, 0-13 de Outubro de 1999.

[11] Charlot, D.; Bard, J.-M.; Canfield, B.; Cuney, C.; Graf, A.; Pirson, A.; Teichner, D.; Yassa, F. “A RISC controlled motion estimation processor for MPEG-2 and

HDTV encoding". *International Conference on Acoustics, Speech, and Signal Processing - ICASSP.* Vol. 5, pp. 9-12 de Maio de 1995.

[12] Chen, W.; Jin, D.; Zeng, L. "Heterogeneous design methodology with configurable regular topology set for scalable Network-on-Chip designs". *International Conference on ASIC*. pp. 1293-1296, 22-25 de Outubro de 2007.

[13] Coulouris, G.; Dollimore, J.; Kindberg, T. "Distributed Systems -- Concepts and Design". Addison-Wesley, 2nd Ed., 1994.

[14] Diguet, J.-P.; Evain, S.; Vaslin, R.; Gogniat, G.; Juin, E. "NOC-centric Security of Reconfigurable SoC". *First International Symposium on Networks-on-Chip*. pp. 223-232, Princeton, EUA, Maio de 2007.

[15] Ditzel, D. R.; Patterson, D. A. "Retrospective on HLLCA. 25 years of the international symposia on Computer Architecture", 1998.

[16] El-Aawar, H. "CISC vs. RISC Hardware and Programming Complexity Measures of Addressing Modes". *Proceedings of the 2nd International Conference Perspective Technologies and Methods in MEMS Design - MEMSTECH*. pp. 43-48, Maio de 2006.

[17] Fiddler, J.; Stromberg, E.; Wilner, D.N. "Software considerations for real-time RISC". *Thirty-Fifth IEEE Computer Society International Conference Compcon Spring '90. 'Intellectual Leverage'*. Digest of Papers. pp. 274-277, 26 de Fevereiro a 2 de Março de 1990.

[18] Gericota, M. G. O. "Metodologias de teste para FPGAs (Field Programmable Gate Arrays) integradas em sistemas reconfiguráveis". *Dissertação de Doutorado*. Faculdade de Engenharia da Universidade do Porto/Porto. Abril de 2003.

[19] Gervini, A.; Corrêa, E.; Carro, L.; Wagner F. "Avaliação de Desempenho, Área e Potência de Mecanismos de Comunicação em Sistemas Embarcados". *SEMISH'03 – XXX Seminário Integrado de Software e Hardware*. Campinas, Brasil, Agosto de 2003.

[20] Hariri, S.; Choudhary, A.; Sarikaya, B. "Architectural support for designing fault-tolerant open distributed systems". *IEEE Computer*. Vol. 25, pp. 50-62, Junho de 1992.

[21] Hong, J.-H.; Cha, E.-J.; Lee, T.-E. "Performance evaluation of cellular phone network based portable ECG device". *30th Annual International IEEE EMBS Conference*. pp. 763-766, Vancouver, British Columbia, Canada, 20-24 de Agosto de 2008.

[22] Hu, S.; Kim, I.; Lipasti, M. H.; Smith, J.E. "An approach for implementing efficient superscalar CISC processors". *The Twelfth International Symposium on High-*

Performance Computer Architecture, 2006. pp. 41-52, 11-15 de Fevereiro de 2006.

[23] Iantovics, B. “The CMDS Medical Diagnosis System”. *Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. pp. 246-253, 26-29 de Setembro de 2007.

[24] Jalote, P. “Fault tolerance in distributed systems”. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[25] Jamil, T. “RISC versus CISC”. *IEEE Potentials*. Vol. 14, pp. 13-16, Agosto-Setembro de 1995.

[26] Jansch-Porto, I. E. S; Weber, T. S. “Recuperação em Sistemas Distribuídos”. *XVI Jornada de Atualização em Informática, XVII Congresso da SBC*. pp 263-310, Brasília, 2-8 agosto de 1997. anais.

[27] Jeffery, C.M.; Figueiredo, R.J.O. “Towards Byzantine Fault Tolerance in Many-core Computing Platforms”. *IEEE International Symposium on Pacific Rim Dependable Computing*. pp. 256-259, 17-19 de Dezembro de 2007.

[28] Johnson, B.W. “Fault-Tolerant Microprocessor-Based Systems”. *IEEE Micro*, Vol. 4, pp. 6-21, Dezembro de 1984.

[29] Kida, M.; Hirayama, R.; Kajikawa, Y.; Tani, T.; Kurumi, Y. “An active noise control system for MR noise Implementation of an ANC system by digital signal processor”. *9th International Conference on Signal Processing – ICSP*. pp. 2689-2692, Pequim, China, 26-29 de Outubro de 2008.

[30] Kojima, L. “Metodologia de Projeto de Sistemas Dinamicamente Reconfiguráveis”. *Dissertação de Mestrado*. Escola Politécnica da Universidade de São Paulo/SãoPaulo. Maio de 2007.

[31] Koren, I.; Krishna, C. M. “Fault tolerant Systems”. Ed. Morgan Kaufmann, 2007.

[32] Lahtinen, V. “System Level Design Experiences and the Need for Standardization”. *International Symposium on System-on-Chip*. pp. 1-4, Tampere, Finlândia, Novembro de 2006.

[33] Laprie, J. C. “Dependable computing and fault-tolerance: concepts and terminology”. *International Symposium on Fault Tolerant Computing*, pp. 2-11, 19-21 de Junho de 1985.

[34] Lee, J.-H.; Lee, W.-C.; Cho, K.-R. “A novel asynchronous pipeline architecture for CISC type embedded controller, A8051”. The 2002 45th Midwest Symposium on Circuits and Systems - MWSCAS. Vol. 2, pp. II-675 - II-678, 4-7 de Agosto de 2002.

[35] Matos, G.; White, E. “Application of dynamic reconfiguration in the design of fault tolerant production systems”. *Fourth International Conference on Configurable Distributed Systems*. pp. 2-9, 4-6 de Maio de 1998.

[36] Maxfield, C. “The Design Warrior’s Guide to FPGAs – Devices, Tools and Flows”. Ed. Elsevier, 2004.

[37] Morais, A. H. F.; Brandão, G. B.; Valentim, R. A. M.; Guerreiro, A. M. G. “eOSI: a Layer Model for Developing Fault-Tolerant Embedded Systems”. *International Conference on Dependable Systems and Networks*. Estoril, Portugal, Junho-Julho de 2009.

[38] Palma, J. C. S.; Mello, A. V. de; Calazans, N. L. V.; Moraes, F. G. “Interface de Comunicação de Cores em FPGAs”. *VIII Workshop Iberchip*, v. 1, Guadalajara, 2002.

[39] Patterson, D. A.; Ditzel, D. R. “The case for the reduced instruction set computer". *Computer Architecture News*, 8(6), pp. 25-33, 1980.

[40] Pradhan, D. K. “Fault-Tolerant System Design”. Prentice Hall, New Jersey, 1996.

[41] Qi, H.; Jiang, Z.; Wei, J. “IP reusable design methodology”. *International Conference on ASIC*. pp. 756-759, 23-25 de Outubro de 2001.

[42] Ribeiro, A. A. L. “Reconfigurabilidade dinâmica e remota de FPGAs”. *Dissertação de Mestrado*. Universidade de São Paulo/São Carlos. Julho de 2002.

[43] Skliarova, I.; Ferrari, A. B. "Introdução à computação reconfigurável", *Electrónica e Telecomunicações*, v. 4, nº 1, pp. 103-119, Setembro de 2003.

[44] Soares, H. B. “Análise e classificação de imagens de lesões da pele por atributos de cor, forma e textura utilizando máquina de vetor de suporte”. *Tese de Doutorado*. Universidade Federal do Rio Grande do Norte/Natal. Fevereiro de 2008.

[45] Stallings, W. “Arquitetura e Organização de Computadores”. Ed. Pearson, 5ª Ed. Abril de 2002.

[46] Tanenbaum, A. S.; Steen, M. V. “Sistemas Distribuídos: Princípios e Paradigmas”. Prentice Hall, 2002.

[47] Vahid, F.; Lysecky, R. “VHDL for Digital Design”. Ed. Wiley, 2007.

[48] Valentim, R. A. M.; Morais, A. H. F.; Guerreiro, A. M.; Brandão, G. B.; Paz de Araújo, C. “MP-HA: Multicycles Protocol for Hospital Automation over multicast with IEEE 802.3”. *6th IEEE International Conference on Industrial Informatics – INDIN*, pp. 979-984, 13-16 de Julho de 2008.

[49] Valentim, R. A. M. “Protocolo Multiciclos para Automação Hospitalar sobre Multicast com IEEE 802.3 utilizando IGMP Snooping”. *Tese de Doutorado*. Universidade Federal do Rio Grande do Norte/Natal. Novembro de 2008.

[50] Vermeulen, B.; Kühnis, R.; Rearick, J.; Stolon, N.; Swoboda, G. “Overview of debug standardization activities”. Design and Test of Computers, IEEE. Vol. 25, pp. 258-267, Maio-Junho de 2008;

[51] Weber, T.; Jansch-Pôrto, I.; Weber, R. “Fundamentos de tolerância a falhas”. Apostila preparada para o IX JAI - Jornada de Atualização em Informática, no X Congresso da Sociedade Brasileira de Computação, Vitória, Espírito Santo, 1990

[52] Winey, B.; Yan Yu. “Oximetry considerations in the small source detector separation limit”. *Proceedings of the 28th IEEE EMBS Annual International Conference*. pp. 1941-1943 New York City, USA, 30 de Agosto a 3 de Setembro de 2006.

[53] Wu, J.; Williams, J.; Bergmann, N. “System Level Design Methodology for Hybrid Multi-Processor SoC on FPGA”. *International Symposium on Field-Programmable Custom Computing Machines*. pp. 312-313, 14-15 de Abril de 2008.

[54] www.altera.com, acessado em 3 de Junho de 2009.

[55] Yi, J. J.; Lilja, D. J. “Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies”. Editora Springer, 2006.

[56] Yu, K.; Koren, I. “Reliability enhancement of real-time multiprocessor systems through dynamic reconfiguration”. *Workshop on Fault-Tolerant Parallel and Distributed Systems - IEEE*. pp. 161-168, 12-14 de Junho de 1994.

[57] Zhang, J.; Tan, H.; Luo, W.; Yang, T.; Wang, S. “Robot Assistant Microsurgery System”. *Proceedings of the 6th World Congress on Intelligent Control and Automation*. Vol. 2, 21-23 de Junho de 2006, Dalian, China.