

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



**ESPECIFICAÇÃO, DESENVOLVIMENTO E
PROTOTIPAGEM DE UM PROTOCOLO DE ALTA
DISPONIBILIDADE EM FPGA**

RÔMERSON DEINY OLIVEIRA

Uberlândia - Minas Gerais

2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



RÔMERSON DEINY OLIVEIRA

**ESPECIFICAÇÃO, DESENVOLVIMENTO E
PROTOTIPAGEM DE UM PROTOCOLO DE ALTA
DISPONIBILIDADE EM FPGA**

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Computação.

Orientador:

Prof. Dr. Daniel Gomes Mesquita

Co-orientador:

Prof. PhD. Pedro Frosi Rosa

Uberlândia, Minas Gerais

2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada “**Especificação, desenvolvimento e prototipagem de um protocolo de alta disponibilidade em FPGA**” por **Rômerson Deiny Oliveira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 21 de Agosto de 2013

Orientador:

Prof. Dr. Daniel Gomes Mesquita
Universidade Federal de Uberlândia

Co-orientador:

Prof. PhD. Pedro Frosi Rosa
Universidade Federal de Uberlândia

Banca Examinadora:

Prof. Dr. João Henrique de Souza Pereira
Algar Telecom

Prof. Dr. Vanderlei Bonato
ICMC USP-SC

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: 21 de Agosto de 2013

Autor: **Rômerson Deiny Oliveira**
Título: **Especificação, desenvolvimento e prototipagem de um protocolo de alta disponibilidade em FPGA**
Faculdade: **Faculdade de Computação**
Grau: **Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

Autor

O AUTOR RESERVA PARA SI QUALQUER OUTRO DIREITO DE PUBLICAÇÃO DESTE DOCUMENTO, NÃO PODENDO O MESMO SER IMPRESSO OU REPRODUZIDO, SEJA NA TOTALIDADE OU EM PARTES, SEM A PERMISSÃO ESCRITA DO AUTOR.

Dedicatória

A minha mãe... Pelos sonhos que sacrificou para que eu realizasse os meus.

Agradecimentos

A Deus, em primeiro lugar, pela possibilidade de conclusão de mais esta etapa.

Aos meus pais, irmãos e sobrinhos, por incondicionalmente terem apoiado cada passo da minha vida e por serem minha fonte diária de inspiração.

Ao meu orientador, Prof. Daniel Gomes Mesquita, pela oportunidade, constante orientação e por todas as exigências que levaram à conclusão deste trabalho. De forma especial, por acreditar em mim e pelo exemplo de integridade.

Ao meu co-orientador, Prof. Pedro Frosi Rosa, pela oportunidade, confiança e conselhos que tanto ajudaram no desenvolvimento da pesquisa. De forma especial, pela humanidade com que conduziu nossos cafés.

Ao Prof. Lásaro Jonas Camargos, pela ajuda efetiva na pesquisa e pela constante prestatividade.

Aos verdadeiros amigos, quer onde estejam, pela força e companheirismo incondicional.

Aos colegas de laboratório que, além de verdadeiros amigos, são companheiros de batalha. A eles, pela incessante ajuda e por todos os momentos de cooperação que não nos deixaram desistir. Em especial ao Cléber, Rafael, Diego, Geycy, Ana Maria, Hiran, Élder, Fabíola, Newarney, Taffarel, Luciane, Juliete e Joicy.

A CAPES pelo apoio financeiro.

"Eu gostaria de resumir este esquema, dizendo que a ciência começa e termina com problemas. (...) Eu tenho tentado desenvolver a tese de que o método científico consiste na escolha de problemas interessantes e na crítica de nossas permanentes tentativas experimentais e provisórias de solucioná-los." Marconi e Lakatos

Resumo

O crescente número de usuários conectados à Internet favoreceu que ela se tornasse um dos principais veículos de transações pessoais e empresariais nos últimos anos. Entretanto, sua indisponibilidade pode acarretar perdas, inclusive de caráter financeiro, aos seus usuários. Apesar dos esforços empenhados para manter a rede 100% do tempo disponível, pesquisas apontam que os protocolos de alta disponibilidade apresentam problemas algorítmicos conhecidos como Acéfalo e Cérebro Partido, que são causados por perdas e erros de mensagens e levam à indisponibilidade da rede. Tais pesquisas propõem, então, que alterações sejam feitas nas especificações dos protocolos existentes considerando que mensagens podem não chegar a seus destinos conforme previsto. Em virtude disso, este trabalho especifica e implementa um novo protocolo de alta disponibilidade, chamado *High Availability Router Protocol* (HARP), cuja operação acontece em ambientes sem preservação de estado. Adicionalmente, apresenta-se um sistema de validação para protocolos de alta disponibilidade que os testam segundo falhas nos canais de comunicação. A especificação do HARP concerne ao ambiente de operação, serviços, vocabulário, formato de mensagens e regras de procedimento especificadas através de máquinas de estados finitos. Ademais, a especificação é complementada pela descrição formal em TLA^+ e sua verificação no contexto de sistemas concorrentes para ratificar as boas propriedades do protocolo. A implementação do HARP consiste da prototipagem em FPGA e o sistema de validação é baseado em um *System on a Programmable Chip*.

Palavras chave: Alta Disponibilidade, Cérebro Partido, FPGA, HARP, Operação de Rede, VRRP.

Abstract

The increasing number of users connected to the Internet led it to become a major vehicle for personal and business transactions in the last years. Nevertheless, its unavailability can result in losses, including financial ones, for its users. Despite of all efforts to keep the network availability nearest to 100% of the time, reasearches have shown that the existing protocols have two algorithmic problems caused by message losses or disruption, named “No Brain” and “Split Brain”, which attack the network availability and lead it to crash. Thus, those researches propose that such protocols must be changed considering the possibility of message loss. In this way, this research specifies and implements the High Availability Router Protocol (HARP), which is a new high availability protocol that operates in stateless environments. Furthermore, a validation system is presented to test high availability protocols for the sake of link failures. The specification concerns to environment assumptions, services, vocabulary, format and procedure rules specified by finite state machine, moreover, the specification is complemented with a TLA⁺ formal description regarding concurrent systems context intending to ratify the HARP good properties. The HARP implementation consists of its prototyping on FPGA and the validation system based on a System-on-Programmable Chip (SOPC).

Keywords: FPGA, HARP, High Availability, Network Operation, Split Brain, VRRP.

Sumário

Lista de Figuras	xxi
Lista de Tabelas	xxiii
Lista de Abreviaturas e Siglas	xxv
1 Introdução	27
2 Estado da Arte	33
2.1 Fundamentação Teórica	33
2.1.1 Computação Reconfigurável	33
2.1.2 Alta Disponibilidade	37
2.1.3 A Linguagem de Especificação TLA ⁺	44
2.2 Literatura Correlata	46
2.2.1 Hardware Reconfigurável para Redes de Computadores	46
2.2.2 Alta Disponibilidade em Redes de Computadores	48
2.3 Conclusão do Capítulo	50
3 Método de Pesquisa	51
3.1 Caracterização do Problema	53
3.2 Desenvolvimento do HARP	54
3.3 Validação	57
3.3.1 Sistema de Validação	58
3.3.2 Características da aplicação de integração	61
3.3.3 Processos Avaliados	63
3.4 Resumo do Método	65
4 High Availability Router Protocol (<i>HARP</i>)	67
4.1 Suposições sobre o ambiente	68

4.2	Serviços e Vocabulário	68
4.3	Formato das Mensagens	69
4.4	Regras de Procedimentos	70
4.4.1	Keep Alive (KA)	70
4.4.2	Inform Node (INF)	71
4.4.3	Remove Node (REM)	71
4.4.4	Grant Master (GM)	72
4.4.5	Check Brain (CB)	73
4.4.6	Autômato Final	76
4.5	Especificação formal em TLA+	78
4.5.1	Resultados da verificação	83
4.5.2	Mapeamento entre especificações	85
4.6	Conclusão do Capítulo	85
5	Arquitetura do HARP	87
5.1	<i>MSG Receiver</i>	90
5.2	<i>Checksum</i>	93
5.3	<i>Type Identifier</i>	96
5.4	<i>Message Assembler</i>	98
5.5	<i>Fifo</i>	100
5.6	<i>HARP Control</i>	102
5.6.1	<i>Calculate Slaves to Election</i>	102
5.6.2	<i>Configure IP_NE</i>	102
5.6.3	<i>Control KA_IND_in</i>	104
5.6.4	<i>Request KA_REQ</i>	104
5.6.5	<i>Active Address Table</i>	105
5.6.6	<i>Respond ACTS_REQ</i>	105
5.6.7	<i>HARP FSM</i>	106
5.7	Resumo dos dados de prototipagem	109
5.8	Conclusão do Capítulo	112
6	Análise dos resultados	113
6.1	Operação funcional do HARP	113
6.2	Análise do Desempenho do HARP	116
6.2.1	Considerações de Tempo	116
6.2.2	Considerações de Frequência	119
6.3	Prototipagem em FPGA	121
7	Considerações Finais	125

Referências	129
A Mensagens HARP	135
B <i>Spec HARP</i>	137
C O Componente DM9000A na biblioteca SoPC	145
C.1 Integração com o <i>SoPC Builder</i>	147
C.2 Códigos	151

Lista de Figuras

2.1	Estrutura interna básica de um FPGA	35
2.2	Fluxo de Projeto para FPGA	36
2.3	Ambiente característico para um protocolo FHRP	39
2.4	Condição de Cérebro Partido	42
2.5	Condição de Acéfalo	43
3.1	Método: fluxograma de etapas	52
3.2	Esquema de conexão para a prova de conceito	56
3.3	Topologia do sistema de validação do HARP	58
3.4	SoPC em cada elemento de rede	59
3.5	Arquitetura da aplicação <i>IntegraSoPC</i>	62
3.6	Etapas do <i>co-design</i> do sistema de validação	66
4.1	Formato da mensagem HARP	69
4.2	Autômato inicial do HARP	71
4.3	Autômatos do serviço <i>Grant Master</i>	73
4.4	Autômatos do serviço <i>Check Brain</i>	74
4.5	HARP - Autômato completo	77
4.6	Organização modular da <i>Spec</i>	79
5.1	HARP - Diagrama em blocos da arquitetura compatível com IPv4	88
5.2	Componente <i>MSG Receiver</i>	90
5.3	Simulação do componente <i>MSG Receiver</i>	92
5.4	Componente <i>Checksum</i>	93
5.5	Simulação do componente <i>Checksum</i>	95
5.6	Componente <i>Type Identifier</i>	96
5.7	Simulação do componente <i>Type Identifier</i>	97
5.8	Componente <i>MSG Assembler</i>	98

5.9	Simulação do componente <i>MSG Assembler</i>	99
5.10	Componente <i>Fifo</i>	100
5.11	Simulação do componente <i>Fifo</i>	101
5.12	Componente <i>HARP Control</i>	103
5.13	Simulação do componente <i>HARP Control</i>	108
5.14	Posicionamento do hardware sintetizado no FPGA	110
5.15	Posicionamento do hardware sintetizado no FPGA	111
6.1	Autômatos do serviço <i>Grant Master</i>	114
6.2	Autômatos do serviço <i>Check Brain</i>	115
6.3	Custos das etapas na transmissão de uma MSG	118
6.4	Gráfico (t_{prot} x Quantidade de Escravos)	119
6.5	Ambiente real de experimentação	122
C.1	Diagrama de blocos interno do DM9000A Davicom (2005b)	145
C.2	Conexão de sinais com a interface do processador Davicom (2005b)	146
C.3	Controlador <i>DM9000A Controller</i> e suas interconexões	148
C.4	Representação da lógica interna do controlador DM9000A	148

Lista de Tabelas

2.1	Comparação entre os principais protocolos de HA	40
3.1	Processos a serem avaliados	64
4.1	Serviços e Vocabulário do HARP	68
5.1	Dados de prototipagem dos componentes	109
6.1	Custo das etapas da aplicação	118
6.2	Frequência mínima de operação	121
6.3	Utilização do FPGA	121
A.1	Nomes das mensagens e suas siglas	135
A.2	Formatação e conteúdo das mensagens HARP	136
C.1	DM9000A - Pinos para a interface do processador Davicom (2005b)	146
C.2	Classificação dos pinos do controlador DM9000A	149

Lista de Abreviaturas e Siglas

ACTS	<i>Active Slave</i>
ALUT	<i>Adaptative Look Up Table</i>
ARP	<i>Address Resolution Protocol</i>
ASIC	<i>Application Specific Integrated Circuit</i>
AVF	<i>Active Virtual Forwardes</i>
AVG	<i>Active Virtual Gateway</i>
BRAM	<i>Block Random-Access Memory</i>
CARP	<i>Common Address Redundancy Protocol</i>
CB	<i>Check Brain</i>
CRC	<i>Cyclic Redundancy Check</i>
CSMA/CD	<i>Carrier Sense Multiple Access with Collision Detection</i>
DE2	<i>Development Education Board 2</i>
DSP	<i>Digital Signal Processing</i>
DUT	<i>Design Under Test</i>
EDA	<i>Electronic Design Automation</i>
FHRP	<i>First Hop Redundancy Protocol</i>
FPGA	<i>Field-Programmable Gate Array</i>
GLBP	<i>Gateway Load Balancing Protocol</i>
GM	<i>Grant Master</i>
HA	<i>High Availability</i>
HAL	<i>Hardware Abstraction Layer</i>
HARP	<i>High Availability Router Protocol</i>
HCPLD	<i>High Complex Programmable Logic Device</i>
HDL	<i>Hardware Description Language</i>
HSRP	<i>Hot Standby Router Protocol</i>
IETF	<i>Internet Engineering Task Force</i>
INF	<i>Inform Node</i>

IP	<i>Internet Protocol</i>
IPS	<i>Intrusion Prevention System</i>
IPv4	<i>Internet Protocol version 4</i>
IPv6	<i>Internet Protocol version 6</i>
IRDP	<i>ICMP Router Discovery Protocol</i>
KA	<i>Keep Alive</i>
LAN	<i>Local Area Network</i>
LE	<i>Logic Element</i>
LUT	<i>Look-Up Table</i>
MAC	<i>Media Access Control</i>
MEF	<i>Máquina de Estados Finitos</i>
MTBF	<i>Medium Time Between Failures</i>
NE	<i>Network Element</i>
NRE	<i>Non-Recurring Engineering</i>
NSR	<i>Network Status Register</i>
NSRP	<i>NetScreen Redundancy Protocol</i>
PHY	<i>Physical Layer</i>
PLD	<i>Programmable Logic Device</i>
PLL	<i>Phase-Locked Loop</i>
REM	<i>Remove Node</i>
SoPC	<i>System-on-Programmable Chip</i>
SPLD	<i>Simple Programmable Logic Device</i>
TLA	<i>Temporal Logic of Actions</i>
TMR	<i>Triple Modular Redundancy</i>
TTM	<i>Time-to-market</i>
UDP	<i>User Datagram Protocol</i>
VHDL	<i>Very High Speed Integrated Circuits Hardware Description Language</i>
VRRP	<i>Virtual Router Redundancy Protocol</i>

Introdução

Nos últimos anos, a Internet se tornou um dos principais meios para transações pessoais e empresariais. Isso se deve principalmente ao seu alcance em escala global, ao número crescente de dispositivos que a acessam e às melhorias tecnológicas que permitiram tanto o aumento da velocidade de acesso quanto uma maior disponibilidade da rede. Segundo o *International Telecommunications Union*, o número de usuários conectados cresceu mais de 450% entre 2003 e 2013, o que reflete, em números absolutos, em uma ascensão de 603,5 milhões para 2,73 bilhões de usuários nos últimos dez anos (ITU, 2013).

É natural neste cenário que cresça também o número de usuários com exigências quanto à disponibilidade da Internet. Disponibilidade, neste contexto, refere-se a diminuir os riscos de falhas da rede e recuperá-la dentro de limites de tempo aceitáveis, que são ditados pelas regras de negócio (Woodberg *et al.*, 2007) (Hawkins e Piedad, 2000). Por exemplo, para um negócio que depende de Web sites *e-commerce*, as exigências de disponibilidade são consideravelmente diferentes de outros cujas transações dependam da Internet apenas para o envio e recebimento de *e-mails* e arquivos.

Em virtude disso, já há alguns anos, diversas empresas e universidades voltaram seus esforços para a alta disponibilidade de rede. A indisponibilidade da Internet pode trazer perdas em altíssima escala para as instituições. Recentemente, o grupo Aberdeen (2012) conduziu uma pesquisa feita com 134 empresas ao redor do mundo e constatou que as perdas de caráter financeiro decorrente dos *downtimes*¹ atingem, em média, 181,7 milhões de dólares por hora de indisponibilidade.

No intuito de diminuir as perdas devidas à indisponibilidade dos serviços de Internet, os requisitos de disponibilidade têm sido cada vez mais rigorosos. Por definição, um

¹Períodos em que uma rede permanece inativa, quer seja por falha de hardware ou por falha de software.

sistema de alta disponibilidade (HA)² é aquele que utiliza mecanismos de detecção, recuperação e mascaramento de falhas, visando manter o funcionamento dos serviços durante o máximo de tempo possível, inclusive no decurso de manutenções programadas (Gray e Siewiorek, 1991). Nesse contexto, disponibilidade refere-se à capacidade de um usuário de determinado sistema acessar, incluir ou modificar os dados existentes, assegurando a integridade de quaisquer alterações realizadas em qualquer intervalo de tempo (Sun *et al.*, 2003a).

Implementações proprietárias são largamente encontradas a este respeito. Lopes Filho (2008) cita três soluções proprietárias de HA para equipamentos de *Firewalls* e *Intrusion Prevention Systems* (IPS). As soluções em questão são oferecidas pelas empresas *Cisco Systems* (Cisco, 2013c), *Juniper Networks* (Juniper, 2013) e *Sun Microsystems* (Oracle, 2013). Elas se aplicam a equipamentos em diferentes níveis das camadas de rede e se concentram no contexto de comutadores³ e roteadores.

Segundo Woodberg *et al.* (2007), parte das soluções clássicas para de alta disponibilidade é caracterizada por mecanismos baseados em redundância de hardware, softwares inteligentes e protocolos para identificação de falhas de sistemas. Eles funcionam por meio de elementos físicos de rede redundantes que aparecem para o resto da rede como um único elemento abstrato, chamado elemento virtual. Este, por sua vez, é reconhecido por um único endereço virtual de rede e outro endereço virtual de máquina. Os elementos físicos que formam um elemento virtual operam na filosofia mestre/escravo, tendo sempre um nó como mestre (não mais e não menos que um) e quantos outros estiverem disponíveis como escravos.

A comunicação entre os elementos de rede redundantes se dá por um protocolo de alta disponibilidade. Este protocolo deve ser capaz de avisar periodicamente a existência de um mestre na rede e deve também ser capaz de eleger um novo mestre no caso de indisponibilidade do atual. Entre os protocolos de alta disponibilidade existentes, o *Virtual Router Redundancy Protocol* (VRRP) (IETF, 2004) destaca-se como o padrão *de facto* de mercado, por ser de especificação pública. A Subseção 2.1.2.1 aponta outros protocolos empregados a este respeito.

Lopes Filho (2008) afirma que situações de indisponibilidade foram percebidas em redes que operavam com o protocolo VRRP. Nestas situações, a rede permanecia indisponível, pois ora não havia equipamento mestre, ora se tinha mais que um nesta função. Nos dois casos, a continuidade dos serviços era impossibilitada, já que as requisições de serviços não eram respondidas ou eram respondidas multiplicadamente, o que gerava uma sobrecarga impossível de ser gerenciada. Após investigações, Hashimoto (2009) apon-

²Do inglês *High Availability*. Termo que designa um nível de disponibilidade normalmente esperado pelos usuários. Neste nível, uma vez estabelecido o cronograma de funcionamento de um sistema, não deve haver indisponibilidade neste período (Hawkins e Piedad, 2000).

³Neste trabalho, os comutadores referem-se a *switches* da camada 2 da arquitetura Internet (Tanenbaum, 2002).

tou as condições de “Acéfalo” e “Cérebro Partido” (Subseção 2.1.2.2) como as causas que afetavam diretamente o funcionamento do VRRP e estendeu os problemas como sendo aplicáveis também a outros protocolos de HA (Subseção 2.1.2.2).

Em virtude disso, Hashimoto (2009) apresentou uma proposta de extensão do VRRP para contornar estas condições (disponível também em (Hashimoto *et al.*, 2010)). A proposta é uma modelagem em redes de Petri (Petri, 1962) e gera um autômato para o elemento virtual, considerando ambientes com possíveis perdas de mensagens. Entretanto, a proposta não foi implementada e é uma asserção apenas para o comportamento do elemento virtual. Ela não define o comportamento dos elementos físicos de rede, tampouco especifica as condições para mudanças de estados entre eles. Ademais, por ter sido modelada em um alto nível de abstração, a proposta não permite detectar detalhes e particularidades da implementação, sendo necessário especificar um novo protocolo, com elementos bem definidos, para implementar e validar o seu funcionamento.

O processo de especificação do protocolo constitui-se da elaboração da sua máquina de estados finitos (MEF) (Hopcroft *et al.*, 2000) e de todas as regras que ditam sua execução (Holzmann, 1991) (Sharp, 2008). Enquanto isso, um sistema de validação capaz de demonstrar o funcionamento deste protocolo requer uma implementação que deixe clara a troca de mensagens que leva à recuperação da rede.

Do ponto de vista da implementação, muitos protocolos da arquitetura Internet (Kurose e Ross, 2006) têm sido implementados em hardware reconfigurável (Brown e Rose, 1996), especialmente em *Filed Programmabel Gate Array* (FPGA), conforme análise da literatura correlata. Isso se deve ao fato de que este tipo de plataforma atinge taxas de encaminhamento e transmissão compatíveis com as tecnologias de rede do estado-da-arte e, devido à reconfigurabilidade, os equipamentos de rede podem se adaptar a diferentes cenários de operação, obtendo-se assim, um hardware flexível e rápido.

Utilizar uma plataforma de hardware reconfigurável no processo de desenvolvimento significa adotar um modelo de projeto baseado em prototipagem rápida. O hardware permite manter o contato com os pormenores da especificação dos elementos do protocolo, atinge altas velocidades de processamento e a exploração do seu paralelismo espacial é adequado à estrutura multicampo das mensagens. Enquanto isso, a característica da reconfigurabilidade permite criar protótipos para teste simultaneamente à elaboração da especificação.

A prototipagem é um processo que dá ênfase à construção de protótipos durante o desenvolvimento e isso permite ter *feedback* mais rápido deste processo (Radatz, 1997) (Tioh *et al.*, 2008). O *feedback* mais rápido é benéfico para identificar problemas que podem reduzir os custos de produção, o que tem feito esta tendência de protótipos crescer desde o surgimento da reconfigurabilidade. Além disso, o processo resulta em um protótipo que pode ter seu hardware reconfigurado, atendendo a diferentes necessidades de re-especificações.

Adicionalmente, a possibilidade de integrar o hardware e o software em uma mesma plataforma, como no caso dos *Systems-On-Programmable Chip* (SoPC) (Fei-yu *et al.*, 2010), adequa-se à construção de um cenário de validação para demonstrar a operação do protocolo HARP. Os SoPCs valem-se da flexibilidade de processadores de uso geral e da velocidade do hardware dedicado e memória.

Assim, este trabalho pretende especificar um novo protocolo para alta disponibilidade baseado na proposta inicial de Hashimoto (2009), demonstrar suas boas propriedades, implementá-lo e prototipá-lo em FPGA. Objetiva-se chegar a uma implementação mais robusta que a do VRRP, garantindo a continuidade do sistema em caso de falhas. A implementação em hardware permite a detecção de detalhes e correção de falhas que escapam ao projeto de protocolos em alto nível de abstração, enquanto a utilização de FPGA agiliza o processo de prototipagem e de correção do hardware projetado.

Para este desenvolvimento, houve a especificação dos cinco elementos do novo protocolo a fim de, sequencialmente, demonstrar o seu funcionamento. Segundo Holzmann (1991), deve-se definir um protocolo pelo seu ambiente de operação, serviços, vocabulário, formato de mensagens e regras de procedimento. Em um baixo nível de abstração, ele é melhor descrito na forma de máquina de estados, pois ela define quais ações são permitidas em um processo, que eventos ele espera acontecer e como ele responderá a estes eventos.

Em suma, do ponto de vista científico, a investigação, o desenvolvimento e a demonstração de um novo protocolo que minimize a ocorrência das condições de Acéfalo e Cérebro Partido motivam este trabalho. O processo é guiado por diretrizes específicas que conduzem a estratégia de análise dos dados e estão elencadas como se segue:

1. Investigar as condições de Acéfalo e de Cérebro Partido e como elas afetam os protocolos de HA;
2. Utilizar FPGA no processo de especificação do novo protocolo, de forma a testar a viabilidade de sua implementação em baixo nível já nas etapas iniciais de projeto;
3. Prototipar o novo protocolo; e
4. Validar formal e experimentalmente o novo protocolo.

Este encadeamento lógico é materializado pelas ações adotadas no método de pesquisa que é abordado no Capítulo 3.

As condições de Acéfalo e Cérebro Partido são investigadas através de pesquisa bibliográfica. A redefinição da proposta de extensão do VRRP é efetivada com a criação dos elementos do protocolo e análise funcional da máquina de estados. Esta especificação dos cinco elementos gerou um novo protocolo para alta disponibilidade batizado de *High Availability Router Protocolo* (HARP) (Oliveira *et al.*, 2013b) (Oliveira *et al.*, 2013a).

A implementação do HARP foi feita em FPGA e sua operação é demonstrada com a utilização de um sistema baseado em SoPC.

É importante deixar claro que o HARP não é um protocolo de roteamento, logo, não se deve esperar resultados oriundos de métricas relativas ao encaminhamento de mensagens, mas sim quanto à continuidade da operação do sistema. Recuperar o sistema de forma transparente ao usuário e mantê-lo em operação é a forma de demonstrar o HARP em funcionamento e é uma das etapas desenvolvidas para validá-lo. Quanto à validação, por sua vez, esta é composta pelas etapas de análise da MEF do HARP, descrição formal em *Temporal Logic of Actions* (TLA⁺) (Lamport, 2003), implementação e simulação dos módulos arquiteturais e prototipagem.

O uso da linguagem TLA⁺ permite demonstrar que o HARP é consistente e bem formado. A técnica é baseada em descrever o protocolo em uma fórmula de lógica temporal que representa as possíveis execuções do sistema. Os comportamentos que satisfazem à descrição (chamada *Spec*)⁴ são os que representam o funcionamento correto da máquina de estados. Um comportamento que satisfaça à *Spec* representa uma sequência de estados para a qual o sistema se comporta corretamente. A *Spec* é construída de forma a considerar várias instâncias do protocolo, possibilitando um ambiente de simulação com execuções concorrentes. Valendo-se dos resultados da verificação da *Spec*, pode-se corroborar as boas propriedades do protocolo.

Por conseguinte, as principais contribuições desta pesquisa são a (i) especificação de um novo protocolo de alta disponibilidade e (ii) uma arquitetura que o permite ser implementado em hardware reconfigurável. Para a primeira delas, o Capítulo 4 apresenta o ambiente característico, os serviços, o vocabulário, o formato, as regras de procedimento e a descrição formal do protocolo HARP. Enquanto isso, para a segunda, o Capítulo 5 expõe a arquitetura elaborada para o protocolo HARP e traz os aspectos organizacionais da sua prototipagem em FPGA.

O HARP é parte de um projeto maior que pesquisa sobre Internet do Futuro, liderado pelo grupo MEHAR (2013). As pesquisas desenvolvidas por este grupo se dividem em algumas frentes, tais como endereçamento baseado em domínios e títulos (Pereira *et al.*, 2011), ontologia e reestruturação do modelo de camadas da Internet (Santos *et al.*, 2011) (Pereira *et al.*, 2010) e alta disponibilidade da rede e serviços (Filho *et al.*, 2008) (Hashimoto *et al.*, 2010). É no âmbito desta última vertente que esta pesquisa se enquadra.

O HARP é um protocolo para a camada de rede da Internet e a versão tratada neste trabalho é compatível com o endereçamento por IPv4. As versões compatíveis com IPv6 e próxima geração da Internet (e.g., *Clean Slate*) não são tratadas, mas são passíveis de desenvolvimento e prototipagem, tendo em vista a plataforma reconfigurável adotada para implementar o HARP. Desenvolver novas versões do HARP significa torná-lo compatível

⁴Do inglês *Specification*. Termo comumente utilizado para referir-se a descrições de sistemas em TLA⁺ (Lamport, 2003).

apenas em formato (de mensagens) com os outros padrões, uma vez que não é necessário alterar a especificação de serviços, vocabulário ou a própria máquina de estados.

O texto desta dissertação encontra-se organizado da seguinte forma:

- O Capítulo 2 apresenta uma visão geral das infraestruturas de alta disponibilidade, os principais protocolos existentes a este respeito, bem como explica as condições de Acéfalo e Cérebro Partido. Este capítulo traz também os fundamentos da linguagem de descrição formal TLA⁺. Por último, é feita uma introdução sobre computação reconfigurável e seus objetivos.
- O Capítulo 3 apresenta o método adotado para o desenvolvimento do trabalho. Ele aborda a caracterização do problema, o desenvolvimento do HARP, o sistema de validação construído e, por último, as métricas utilizadas para avaliação da qualidade da solução.
- O Capítulo 4 descreve os elementos do protocolo HARP, incluindo ambiente, serviços, vocabulário, formato, a máquina de estados e as regras de procedimento que a ditam. Neste capítulo, a especificação formal é apresentada, bem como as propriedades que foram verificadas sobre a corretude do protocolo.
- O Capítulo 5 apresenta a arquitetura elaborada para permitir a implementação do protocolo e como ela está prototipada no hardware reconfigurável. São mostrados seus módulos arquiteturais, a estrutura de interconexão entre eles e os dados de prototipagem no FPGA.
- O Capítulo 6 discorre sobre a análise dos resultados experimentais. As análises envolvem o processo de recuperação da rede, considerando aspectos tais como a frequência de operação e desempenho do hardware, com vistas aos parâmetros da prototipagem em FPGA.
- Por último, o Capítulo 7 aborda as considerações finais desta dissertação, ressaltando suas limitações e propostas de melhorias para trabalhos futuros.

Estado da Arte

Subdividido em duas seções, este capítulo tem como objetivos (i) apresentar o aparato teórico sobre o qual esta dissertação se baseia, bem como (ii) apresentar uma revisão do estado da arte relacionado aos tópicos investigados nesta pesquisa. Desta forma, a Seção 2.1 fornece uma visão geral sobre a reconfigurabilidade de hardware, introduz as tecnologias existentes nos principais protocolos de alta disponibilidade implementados por fornecedores de equipamentos e traz conceitos sobre a linguagem de especificação TLA⁺. Seguidamente, a Seção 2.2 apresenta os principais trabalhos relacionados à pesquisa aqui desenvolvida.

2.1 Fundamentação Teórica

Nesta seção, são apresentados conceitos que serão mencionados ao longo do trabalho. Inicialmente, é dada uma visão geral sobre as arquiteturas de alta disponibilidade, que retrata o ambiente clássico para um protocolo da natureza do HARP. Em seguida, é feita uma descrição do FPGA, o dispositivo de hardware escolhido para sua prototipagem.

2.1.1 Computação Reconfigurável

No universo da computação e da eletrônica, tradicionalmente os algoritmos vêm sendo executados por dois métodos clássicos. O primeiro deles é baseado na execução por software enquanto o segundo é baseado em hardware. Uma execução em software consiste de uma aplicação sendo executada em um microprocessador e tem a vantagem de oferecer flexibilidade ao processo, uma vez que é possível alterar as instruções do programa sem alterar o hardware do sistema computacional (Hauck e DeHon, 2010). Entretanto,

a complexidade computacional (Cormen *et al.*, 2009) destas aplicações pode introduzir gargalos no processamento e uma latência adicional indesejada ao sistema (Compton e Hauck, 2002) (Kachris e Vassiliadis, 2006).

Para exemplificar a ocorrência destes gargalos, cita-se a situação conhecida por Gargalo de Von-Neumann (Backus, 1978) em que o processador deve decidir entre executar um número elevado de instruções por segundo e ler um grande volume de dados. A cada ciclo do *clock* no computador, somente uma instrução ou um dado trafega pelo barramento entre a memória principal e o processador ao mesmo tempo, o que torna inviável a exploração do paralelismo de algumas operações e, conseqüentemente, inviabiliza também a aceleração do processamento através deste recurso.

Por outro lado, quando a execução é feita por um hardware de aplicação específica (ASIC)¹ (Deschamps *et al.*, 2006), isso incorre em um notório ganho de desempenho para o sistema. Um exemplo disso é que são eliminadas as interações no nível do sistema operacional e a troca de dados com a aplicação. Em contrapartida, esta solução extingue a característica de flexibilidade do sistema, requer longo período para fabricação do chip e pode ainda tornar o custo do sistema inviável para escalas de produção reduzidas.

Convergingindo estes dois paradigmas, os dispositivos de lógica programável (PLD)² introduziram a idéia de um circuito de hardware digital que pode ter sua lógica combinacional reprogramada (Brown e Rose, 1996). Baseados em tecnologia de memória reprogramável, os PLDs permitem aos projetistas mudar o circuito quantas vezes for necessário, até que ele opere satisfatoriamente.

Muitos tipos de PLDs estão disponíveis no mercado, indo desde os *Simple* PLD (SPLD) que são dispositivos de menor capacidade e capazes de implementar algumas equações lógicas, até os *High Complex* PLD (HCPLD) que são, por sua vez, dispositivos com grandes capacidades e capazes de comportar sistemas computacionais complexos, incluindo processadores e controladores de periféricos (Deschamps *et al.*, 2006) (Brown e Rose, 1996). Para este trabalho, escolheu-se um tipo de HCPLD como plataforma de prototipagem devido à grande quantidade de elementos lógicos disponíveis para prototipagem e pela facilidade de integração com interfaces de comunicação externas, a saber: *Field-Programmable Gate array* (FPGA).

Introduzido em 1984 pela companhia Xilinx (Trimberger, 1994), o FPGA é constituído de um arranjo bidimensional de blocos lógicos configuráveis e uma estrutura de interconexões que os interligam. Possui também blocos de entrada e saída que se conectam aos terminais de conexão com outros dispositivos. A Figura 2.1 mostra a estrutura interna de um FPGA.

Os blocos lógicos (ou elementos lógicos) são formados basicamente por *Look-Up Tables* (LUT), *flip-flops* e alguma lógica combinacional que varia de acordo com o fabricante (os

¹Do inglês *Application-Specific Integrated Circuit*.

²Do inglês *Programmable Logic Devices*.

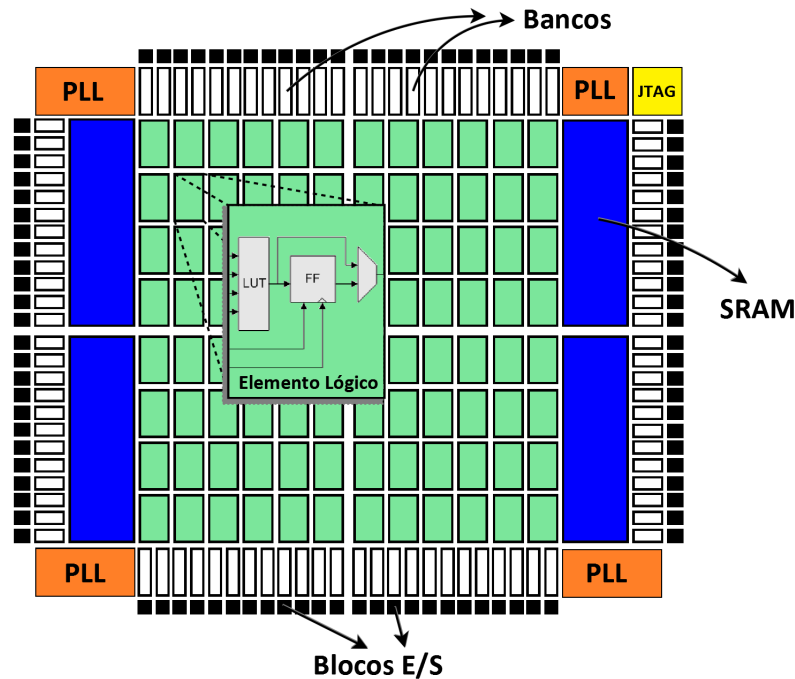


Figura 2.1: Estrutura interna básica de um FPGA

(Adaptada de <http://www.cresitt.com/technologies/HelpEpisipHtml/EDK/FPGA.htm>)

componentes do elemento lógico na Figura 2.1 são ilustrativos). Pela característica da reconfigurabilidade, a matriz de interconexão no FPGA introduz atrasos maiores que os ASICs, mas garante a flexibilidade do dispositivo. Graças às ferramentas de *Electronic Design Automation* (EDA), circuitos em FPGA podem ser implementados em um curto espaço de tempo, abstraem as especificações físicas, reduzem os custos inerentes ao projeto (NRE)³ e requerem baixos *time-to-market* (TTM) (Deschamps *et al.*, 2006).

Alguns outros elementos, tais como blocos de memória interna, *Phase-Locked Loop* (PLL)⁴, somadores, multiplicadores embarcados, *Digital Signal Processing* (DSP)⁵, entre outros, também podem ser encontrados internamente em um FPGA devido à grande capacidade de integração dos chips disponíveis no mercado. Os FPGAs são dispositivos que possuem elevada quantidade de elementos lógicos configuráveis. Ademais, eles se consolidaram como dispositivos que combinam os benefícios da eficiência do hardware com a flexibilidade do software.

Para grande parte das aplicações, a chave do FPGA está em sua reconfigurabilidade e adaptatividade a variadas aplicações. Conforme mostra a subseção 2.2, o FPGA tem alcançado velocidades compatíveis com o estado da arte em suas aplicações para implementação de protocolos de rede e, graças à redução do *time-to-market* e dos custos NRE, ele tem se difundido como plataforma de protipagem de protótipos em baixa escala.

³Do inglês *Non-Recurring Engineering Costs*. Termo aplicado aos custos envolvidos no processo de produção de um produto, incluindo as fases de pesquisa, desenvolvimento, projeto e testes.

⁴Circuitos utilizados na eletrônica para ajustar e gerar sinais osciladores.

⁵DSPs são circuitos que implementam funções específicas para o tratamento de sinais digitais.

Devido à disponibilidade de plataformas de prototipagem rápida, pesquisas tipicamente teóricas podem assumir a forma de investigação mais aplicada, que acaba por beneficiar tanto a comunidade acadêmica quanto mercadológica (Tioh *et al.*, 2008). O uso de placas de prototipagem que trazem dispositivos de entrada e saída conectados ao FPGA facilita o desenvolvimento de protótipos e soluções que interagem com tecnologias já difundidas em equipamentos utilizados pelas empresas e comunidade acadêmica.

O uso de prototipagem rápida, devido à facilidade trazida pelas plataformas que integram o FPGA com outras interfaces, tem permitido aos pesquisadores adquirir um entendimento mais complexo dos problemas de pesquisa e ainda tornam o ambiente mais propício ao surgimento de ideias para a formulação de soluções mais robustas (Tioh *et al.*, 2008).

2.1.1.1 Fluxo de Projeto em FPGA

A concepção e implementação da arquitetura se concretizou conforme o fluxo de etapas ilustrado pela Figura 2.2. Estas etapas configuram o fluxo clássico de projeto em FPGA. A ideia geral é cumprir a sequência de especificação, verificação, implementação e depuração do sistema.

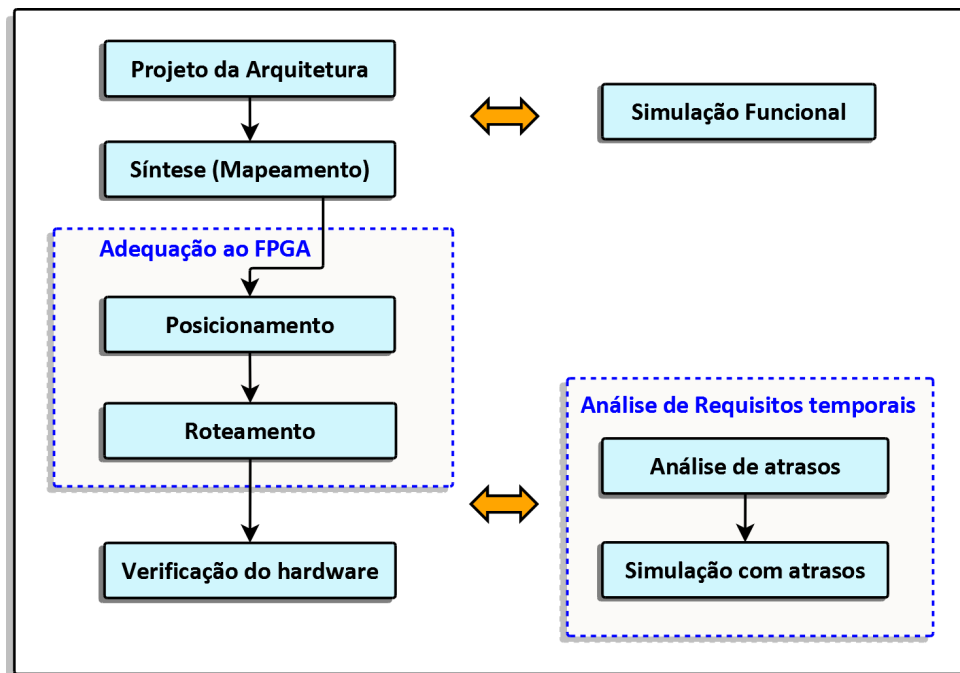


Figura 2.2: Fluxo de Projeto para FPGA

O *Projeto da Arquitetura* envolve analisar os requisitos do hardware, criar os módulos arquiteturais e os interligar organizacionalmente. A *Síntese (Mapeamento)* é o processo pelo qual uma descrição abstrata do comportamento de um circuito é implementada em termos de circuitos lógicos. Sendo assim, descrições em linguagem de descrição de hardware (HDL) são transformadas em circuitos descritos por lógica combinacional e

sequencial de portas lógicas.

A *Simulação Funcional* não compõe o processo de síntese, ela é parte da verificação do sistema. A simulação permite submeter o módulo sintetizado a testes de operação, simulando dados para os sinais de entrada através de *testbenches*. Os *testbenches* são ambientes virtuais utilizados para verificar a corretude dos módulos arquiteturais a serem prototipados. Neste caso, os *testbenches* são criados com arquivos descritos em HDL e se comportam como um módulo de hardware que instancia os outros módulos a serem testados. Eles geram estímulos para os sinais de entrada do módulo a ser testado e mede a resposta a estes estímulos nas saídas do componente. Para a simulação, foi utilizada a ferramenta *Modelsim Altera Starter Edition*, da *Mentor Graphics* (Graphics, 2013).

Na fase de *Adequação ao FPGA* é que ocorre o mapeamento dos resultados da síntese para especificar sua localização dentro do dispositivo de destino com referência a restrições de área e desempenho. Especifica recursos de roteamento a serem usados para conectar os elementos lógicos e demais componentes dentro do FPGA. Quando se obtém os resultados da *Posicionamento e Roteamento* pode-se partir para a *Análise de Requisitos Temporais*.

A fase de *Análise de Requisitos Temporais* também faz parte do processo de verificação do hardware. Nesta etapa, a ferramenta inclui restrições de atraso e frequência de operação que podem ser definidas pelo projetista, ou ainda, a ferramenta pode reportar estes limites ao projetista para que ele cuide que o circuito seja submetido a estas condições. Então, uma simulação novamente é feita, mas desta vez considerando os atrasos combinacionais e de roteamento introduzidas pelo FPGA.

Por fim, há a *Verificação do Hardware*. Para o FPGA, que é um dispositivo de hardware reconfigurável, uma vez que a simulação foi feita, os testes de operação são realizados diretamente no hardware. Assim, é feita a configuração do FPGA através do código executável (*bitstream*) e passa-se, então, à observar se a sua operação ocorreu conforme projetado.

Depois de concluir esta sequência para cada um dos módulos arquiteturais, todos eles foram integrados em um único projeto de hardware (ilustrado pela Figura 5.1) e prototipados no FPGA. A descrição destes módulos é dada no Capítulo 5.

2.1.2 Alta Disponibilidade

Na maioria das redes locais (LAN)⁶ (IEEE, 2001), os nós finais da rede são conectados a comutadores da camada de enlace e referidos como centros de comutação Ethernet. Através destes comutadores, os nós se conectam a um roteador predefinido. Este roteador é designado como o *gateway* padrão da rede e qualquer quadro Ethernet que não pertença àquela rede deve ser encaminhado para sua interface (Tanenbaum, 2002).

A indisponibilidade dos *gateways* pode, portanto, isolar os nós finais das redes. Para

⁶Do inglês *Local Area Network*.

evitar isto, métodos tais como protocolos de roteamento e ICMP⁷ *Router Discovery Protocol* (IRDP) (IETF, 1991) são empregados. Entretanto, alguns nós não podem executar esse tipo de protocolo ou mesmo estes protocolos poderiam tomar muito tempo para convergir usando IRDPs (Matsuda, 2012). Para superar estas deficiências, os protocolos *First Hop Redundancy Protocols* (FHRP) (Cisco, 2013a) têm sido largamente empregados em LANs.

Olhando da perspectiva de um nó final conectado a uma LAN, *First Hop* é o roteador IP agindo como *gateway* padrão da rede. FHRPs são os protocolos utilizados para proteção contra falhas deste *gateway* e permitem que o processo de recuperação de falha neste ponto da rede ocorra de forma transparente ao nó final. Os FHRPs são importantes para evitar que a rede se torne inoperante, uma vez que os *gateways* são os únicos pontos de conexão com as redes externas.

Os FHRPs pressupõem que os nós em geral conhecem o endereço IP do gateway, eliminando a necessidade de clientes finais executarem protocolos de roteamento. Tais protocolos são divididos em dois grupos. O primeiro grupo é composto pelos protocolos proprietários e tem o *Hot Standby Router Protocol* (HSRP) (Cisco, 2013d), da Cisco, como principal representante. Já o segundo grupo é representado pelo VRRP (IETF, 2004), que é um protocolo padrão do *Internet Engineering Task Force* (IETF)⁸.

De forma geral, os mecanismos de alta disponibilidade são caracterizados por usarem soluções baseadas em redundância de hardware, softwares inteligentes e protocolos FHRPs para identificação de falhas de sistemas (Sun *et al.*, 2003b) (Matsuda, 2012). Eles funcionam por meio de elementos de rede redundantes que aparecem para o resto da rede como um único elemento abstrato chamado elemento virtual, que é reconhecido por um único endereço virtual de rede e outro endereço virtual de máquina (IETF, 2004). A Figura 2.3 ilustra um ambiente característico para um protocolo FHRP em operação, especificamente, o conjunto composto pelos três roteadores.

Na Figura 2.3, a LAN representa qualquer estrutura de interligação Ethernet entre os nós, podendo ser constituída, por exemplo, por comutadores redundantes. Os enlaces entre os roteadores podem conter ligações diretas entre si, dependendo da disponibilidade de portas e limitações geográficas. Os elementos físicos que formam um roteador virtual operam na filosofia mestre/escravo, tendo sempre um nó como mestre (não mais e não menos que um) e quantos outros estiverem disponíveis como escravos. A comunicação entre os elementos de rede redundantes se dá por um protocolo FHRP. Este protocolo deve ser capaz de avisar periodicamente a existência de um mestre na rede e deve também ser capaz de eleger um novo mestre no caso de falha do atual.

⁷ICMP refere-se ao protocolo *Internet Control Message Protocol*.

⁸O *Internet Engineering Task Force* (IETF) é uma grande comunidade internacional de profissionais e pesquisadores de rede que trabalham em prol da arquitetura Internet, criando e monitorando padrões que mantêm o funcionamento desta arquitetura e sua evolução (IETF, 2013).

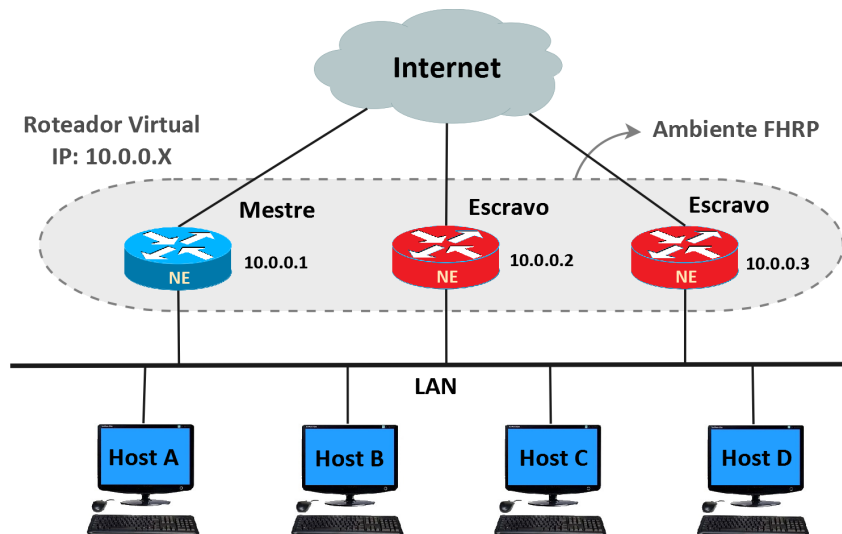


Figura 2.3: Ambiente característico para um protocolo FHRP

2.1.2.1 Principais protocolos de alta disponibilidade

Os protocolos *Virtual Router Redundancy Protocol* (VRRP) e o *Hot Standby Router Protocol* (HSRP) são os mais populares FHRPs existentes (Hashimoto *et al.*, 2010) (Matsuda, 2012). Entretanto, há também alguns outros com notoriedade a este respeito: *Gateway Load Balancing Protocol* (GLBP) (Cisco, 2013b), *NetScreen Redundancy Protocol* (NSRP) (Woodberg *et al.*, 2007) e *Common Address Redundancy Protocol* (CARP) (OpenBSD, 2013). Conceitualmente, todos estes protocolos são semelhantes e usam a mesma ideia do roteador virtual.

Para o HSRP e o VRRP, um roteador é eleito para possuir o endereço IP virtual e é responsável por gerenciar todas as requisições enviadas a este endereço. Todos os outros roteadores permanecem inativos. O roteador ativo detém o endereço IP virtual e o endereço MAC, também virtual. Para divulgar informações de estado do roteador ativo, mensagens de aviso (*advertisements*) são trocadas periodicamente entre todos os roteadores participantes do grupo. A percepção de falha do roteador ativo é acusada pelo não recebimento de três *advertisements* consecutivos.

No HSRP, o roteador eleito é chamado roteador ativo (*active router*). Um grupo HSRP tem no máximo um roteador ativo, pelo menos um roteador em espera (*standby router*) e talvez mais que um roteador em escuta (*listening router*). Já no VRRP, o roteador eleito é chamado roteador mestre (*master router*) e um ou mais roteadores chamados roteadores reserva (*backup router*).

No caso do GLBP, ele realiza balanceamento de carga através do conceito de *Active Virtual Gateway* (AVG) e *Active Virtual Forwarders* (AVF), que permitem responder as requisições utilizando diferentes endereços *Media Access Control* (MAC) (IEEE, 2001). O AVG eleito atribui endereços MAC para cada um dos membros do grupo GLBP, incluindo ele mesmo, habilitando então os AVFs. O segundo AVG permanece no estado de espera

(*Standby*) e os outros são colocados no estado escuta (*Listening*).

O NSRP é um protocolo proprietário da Juniper, voltado essencialmente para a redundância de *Firewalls* da plataforma NetScreen (da própria fabricante). Para o NSRP, os roteadores são intitulados mestres (*Masters*) e reservas (*Backups*). Segundo Woodberg *et al.* (2007), este protocolo nunca foi implementado para mais de dois equipamentos e provavelmente nunca será. Isso faz com que o monitoramento seja direcionado a apenas um equipamento, pois não há redundância com mais de dois deles e, já que não há eleição, a probabilidade de ocorrer as condições de Acéfalo e Cérebro Partido aumentam.

O CARP é uma implementação da comunidade OpenBSD e possui o mesmo princípio do VRRP, entretanto as denominações para roteadores ativos e inativos são mestre (*Master*) e reserva (*Backup*), respectivamente (OpenBSD, 2013). Como dito anteriormente, são protocolos muito parecidos em suas especificações. A Tabela 2.1 compara, dentre todos, os protocolos mais difundidos para alta disponibilidade.

Tabela 2.1: Comparação entre os principais protocolos de HA

	HSRP	VRRP	GLBP	NSRP
Proprietário	Cisco	IETF RFC	Cisco	Juniper
Nomenclatura	<i>Active</i> <i>Standby</i> <i>Listen</i>	<i>Master</i> <i>Backup</i>	<i>Active Virtual Gateway</i> <i>Active Virtual Forwarder</i>	<i>Master</i> <i>Backup</i>
Intervalo de aviso	3 seg	1 seg	3 seg	1 seg
Intervalo de falha	10 seg	3 seg	10 seg	3 seg
<i>Preemption</i> <i>Default</i>	Desligado	Ligado	Desligado	Desligado
Balanceamento de carga	Não nativo	Não nativo	Sim	Não nativo
Equipamentos	Roteadores Cisco	Roteadores dedicados ou plataformas Linux	Roteadores Cisco	Firewall NetScreen
Endereçamento	IP e MAC virtuais idênticos em cada roteador do grupo.	Um ou mais IP e MAC Virtuais. Os IP e MAC reais também podem ser utilizados.	Um IP Virtual e vários MAC virtuais que identificam os roteadores do grupo.	IP e MAC virtuais para o grupo.

De acordo com a Tabela 2.1, percebe-se que não existem grandes variações entre os protocolos. As diferenças mais importantes são os intervalos de aviso e falha que correspondem. O primeiro corresponde ao intervalo entre o recebimento de duas mensagens de aviso (*advertisements*) sucessivas, enquanto o segundo diz respeito ao intervalo total para a percepção de uma falha no grupo. Além disso, eles variam em características adicionais,

tais como balanceamento de carga e compatibilidade com equipamentos. Para a opção de *Preemption*, se ele está ativado, quando um roteador volta de uma queda ele reassume a função de mestre se tiver maior prioridade. Quanto ao balanceamento de carga, ele é nativo apenas no GLBP, que só é executado em equipamentos fabricados pela Cisco.

O VRRP tornou-se o protocolo FHRP padrão *de facto* para os equipamentos de alta disponibilidade e o mais empregado no mercado, seguido do HSRP (Hashimoto *et al.*, 2010) (Matsuda, 2012). Segundo Hashimoto (2009), os problemas percebidos no VRRP podem ser estendidos a todos os outros citados nesta seção, em virtude de análise realizada sobre a máquina de estados de cada um deles.

2.1.2.2 Inconsistências dos protocolos existentes

Os protocolos apresentados nesta seção se destinam essencialmente a gerenciar a comunicação entre elementos redundantes e sustentar arquiteturas de alta disponibilidade. Apesar disso, dois problemas algorítmicos foram apontados nestes protocolos e podem levar à indisponibilidade da rede (Woodberg *et al.*, 2007) (Hashimoto, 2009) (Pereira Júnior, 2010). Estes problemas resultam em duas situações indesejadas que foram intitulados por condição de Acéfalo e condição de Cérebro Partido. Os itens a seguir, acompanhados das Figuras 2.4 e 2.5, explicam estas condições.

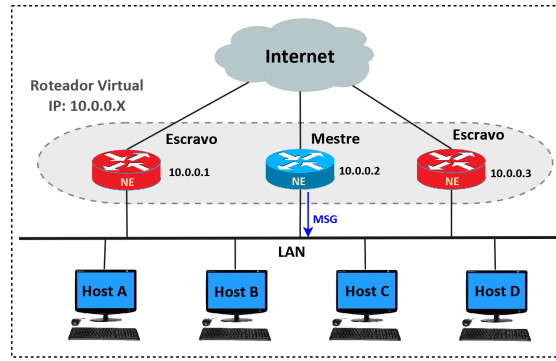
Condição de Cérebro Partido

É a condição em que um ou mais nós de uma infraestrutura de alta disponibilidade (HA) se tornam mestres, por julgarem que o nó mestre atual se tornou inoperante. Ela pode ocorrer quando os nós de uma arquitetura de HA perdem a comunicação entre si, por exemplo quando os enlaces HA são interrompidos (Woodberg *et al.*, 2007) (Hashimoto, 2009). Além disso, esta condição pode ser gerada em ambientes onde as primitivas não são entregues de um nó a outro ou são recebidas com erros, levando cada um dos nós a concluir que não há mestre no grupo e, portanto, se autoeleger mestre.

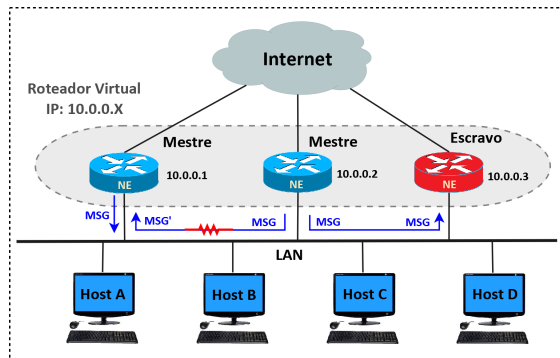
A Figura 2.4 mostra o resultado desta situação. Na figura, as mensagens (MSG) enviadas pelo(s) mestre(s) chegam aos outros elementos com valores corrompidos, ou podem mesmo nem chegar, concluindo pela falta de mestre.

Na Figura 2.4(a), o mestre envia suas mensagens de aviso (*heartbeats*) aos escravos do grupo. A mensagem (MSG) deixa a interface do mestre sem nenhum erro. Entretanto, ao atingir o escravo do lado esquerdo na Figura 2.4(b), a MSG sofre alguma alteração não detectável pela camada de enlace e é recebida e aceita pelo escravo. O processo se repete suficientemente para que este escravo julgue não ter mestre no grupo. Logo, ele se torna mestre mesmo com outro mestre ativo e começa também a disparar suas MSGs.

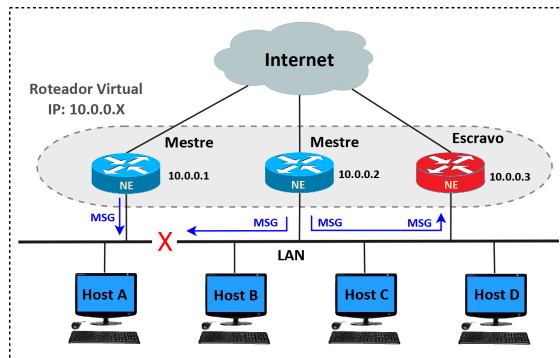
Caso as conexões entre este escravo e o mestre não estejam disponíveis (seja por quebra de canal ou por instabilidade da rede), como representado na Figura 2.4(c), as mensagens



(a) Envio da MSG pelo mestre



(b) Recebimento de MSG corrompida



(c) Não recebimento de MSG

Figura 2.4: Condição de Cérebro Partido

podem não chegar a este escravo e ele se tornar mestre também neste caso, deixando a rede com dois mestres mais uma vez.

Serviços não orientados à conexão são mais suscetíveis a este tipo de erro, uma vez que eles não são orientados à conexão e não operam com mensagens de confirmação de recebimento. Este é o caso dos protocolos Ethernet, *Internet Protocol* (IP) e *User Datagram Protocol* (UDP) (protocolos das camadas inferiores que suportam o VRRP).

Condição de Acéfalo

É a condição em que um nó mestre de uma infraestrutura de alta disponibilidade se torna inoperante e nenhum outro nó se habilita a assumir tal papel (Pereira Júnior, 2010). Este problema é consequência de falhas múltiplas na rede. Segundo Lopes Filho (2008) se, por exemplo, ocorre falha de comutadores independentes ao mesmo tempo, onde as interfaces de rede de *firewalls*/IPS estão conectadas, ambos os equipamentos detectarão a falha e entrarão em estado inoperante.

Outra causa é o recebimento de mensagens de aviso quando não enviadas pelo mestre. Isso pode acontecer em virtude de ataques por invasores. A Figura 2.5 mostra que mensagens de aviso são recebidas por todos os equipamentos. Na Figura 2.5(a), o mestre envia a mensagem (MSG) de aviso aos escravos, entretanto logo em seguida (Figura 2.5(b)) ele também recebe uma mensagem de aviso e julga que há outro mestre na rede. Neste caso,

se a prioridade no *advertisement* recebido seja maior que a dele, ele se torna escravo e a rede fica sem *gateway*.

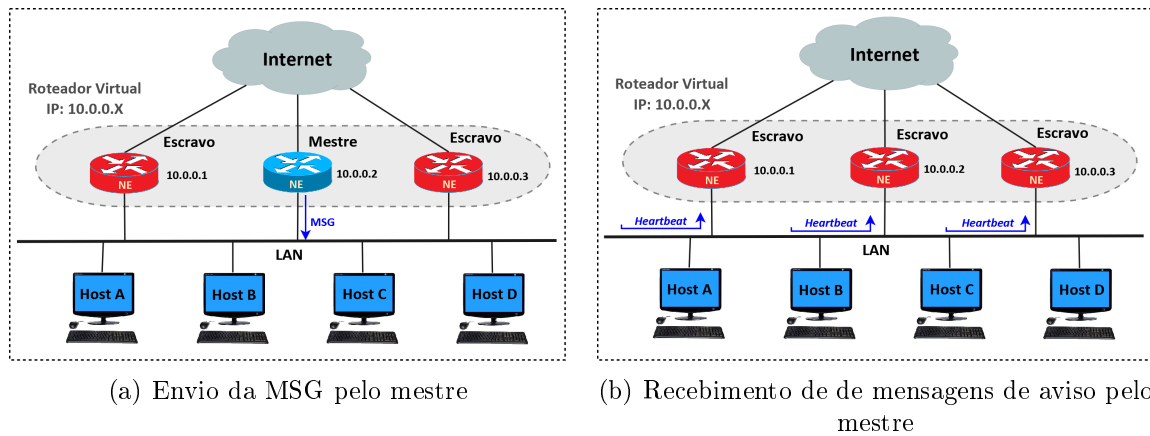


Figura 2.5: Condição de Acéfalo

Em (Woodberg *et al.*, 2007), os autores expõem um cenário onde a condição de Acéfalo pode ocorrer, mesmo com o protocolo NSRP em execução. Neste caso, dois equipamentos de *firewall* estão conectados cada um a um comutador. Estes, por sua vez, estão conectados tanto à zona protegida quanto à zona desprotegida da rede. No pior caso de operação, os dois comutadores se tornam inoperantes por uma falha na zona desprotegida. Assim, as duas instâncias do protocolo NSRP em execução nos *firewalls* vão ao estado de inoperante e, mesmo com conectividade à Internet, a zona protegida se torna indisponível.

No âmbito do VRRP, Pereira Júnior (2010) afirma que do ponto de vista da segurança, é possível para qualquer máquina situada na mesma rede local do roteador virtual se passar por um roteador VRRP impostor, alegando participar do mesmo domínio e, ademais, possuir prioridade superior ao roteador virtual atual. Neste caso, o roteador VRRP atual poderá ir ao estado reserva (*backup*) e o grupo ficar sem líder (Acéfalo).

Para estas duas condições, quando se trata de falhas de canal e *interface*, há uma probabilidade muito maior de ocorrer a condição de Cérebro Partido em relação à condição de Acéfalo (Hashimoto, 2009). Os projetistas do VRRP consideram que a única razão para não chegar uma primitiva de anúncio é a falha do mestre e isso é extensível aos protocolos citados na Subseção 2.1.2.2 (Hashimoto, 2009). Entretanto, ambientes de rede complexos mostram que esta não é a única razão. Falhas de enlace e erros detectáveis pelo protocolo Ethernet inutilizam ou descartam os quadros de mensagens enviados periodicamente pelo mestre, resultando em uma eleição na condição de Cérebro Partido (Hashimoto, 2009).

2.1.3 A Linguagem de Especificação TLA^+

A linguagem TLA^+ (Lamport, 2003) é uma linguagem baseada na combinação da Lógica Temporal de Ações (TLA)⁹ (Lamport, 1994) e a Teoria de Conjuntos (Enderton, 1977) e que foi projetada para construir especificações em alto nível de sistemas distribuídos e concorrentes, em particular, sistemas assíncronos (Merz, 2003). Segundo Camargos (2009), ambientes assíncronos (Lynch, 1996) são mais simples e gerais que os demais, o que quer dizer que soluções encontradas nestes ambientes também funcionarão em outros mais restritos.

Por combinar a TLA e a teoria clássica dos conjuntos, a linguagem TLA^+ provê um formalismo expressivo para especificações e suporta verificação de asserções e implicações lógicas. O formalismo do TLA^+ é proposto como uma ferramenta apropriada para formalizar a semântica de programas e sistemas concorrentes. Os sistemas e suas propriedades são representados na mesma lógica e ambos são expressos por implicações lógicas (Lamport, 1994).

Em TLA^+ , uma especificação (*Spec*) de um sistema consiste de matemática ordinária combinada com Lógica Temporal (Lamport, 2003). Uma *Spec* é um modelo matemático de alguma parte do sistema. O propósito primordial de uma especificação é ajudar a evitar erros. Deve-se especificar a parte do sistema para a qual a especificação é mais provável de revelar erros. Neste contexto, TLA^+ é indicado para revelar erros que surgem através da interação de componentes assíncronos (Lamport, 2003).

Conforme exposto, a linguagem foi construída para descrever propriedades temporais para sistemas com execução concorrente. Por temporal entende-se construir proposições que se referem a comportamentos desejáveis do sistema após uma sequência de transições entre estados. As variáveis que definem estes estados podem tanto manter seus valores atuais como assumirem valores futuros. Desta forma, os operadores *necessário* ($\Box$) e *possível* ($\Diamond$) - que na modalidade da Lógica Temporal significam, respectivamente, *Sempre no futuro* e *Em algum lugar do futuro* - formam expressões com propriedades que podem ser verificadas pela evolução dos estados ao longo do tempo. Características de sintaxe da linguagem são encontradas em Lamport (2000), alguns pontos relevantes a este respeito são apresentados na Seção 4.5, que traz também um exemplo de uso da linguagem TLA^+ aplicado ao HARP.

Há duas classificações fundamentais para as propriedades que se pretende descrever e demonstrar com TLA^+ : *Safety* e *Liveness* (Lamport, 1977). A primeira delas cuida do que o sistema é projetado para não fazer, enquanto a segunda afirma que algo considerado comportamento benéfico deve acontecer em algum momento.

As propriedades de *Safety* são as que dizem que nada de ruim acontece no sistema,

⁹Do inglês *Temporal Logic of Actions*. Ela inclui operadores da Lógica Temporal (Pnueli, 1979) para referir-se a valores de variáveis no passado e no futuro.

i.e., não há violação de nenhuma garantia preestabelecida. Elas são o ponto inicial de qualquer tentativa de verificação. Já as propriedades de *Liveness* servem para indicar que algo vai acontecer em algum momento ou, em outras palavras, em algum momento algo bom acontece no sistema. Somente examinando todos os comportamentos possíveis é que se pode demonstrar propriedades desta natureza. Elas são expressas como fórmulas temporais.

Para exemplificar, considere o problema da exclusão mútua (Dijkstra, 1965), onde o sistema deve garantir que dois processos (ou *threads*) não entrem na região crítica ao mesmo tempo. Garantir que dois processos nunca entram na região crítica ao mesmo tempo refere-se a propriedades de *Safety*. Enquanto isso, garantir que um processo entra na região crítica em algum ponto no futuro diz respeito às propriedades de caráter *Liveness*.

As especificações (*Specs*) são verificadas utilizando-se o *TLC Model Checker* (Lamport e Yu, 2011), que é uma ferramenta desenvolvida para encontrar erros em especificações TLA^+ . Posto que as especificações são construídas com fórmulas temporais que descrevem comportamentos e mudanças de estados, o verificador percorre todos os possíveis estados visitados em virtude de eventos de entrada e procura por *deadlocks* e demais inconsistências da especificação. Devido a um problema de explosão de estados, é necessário restringir as condições de teste, por isso diz-se que o TLC verifica as *Specs* e não que as prova.

Um sistema pode ser especificado em diferentes níveis de abstração, desde uma descrição das suas propriedades no mais alto nível até uma descrição da sua implementação em termos de microcódigos e circuitos. Assim, existem relações estabelecidas entre duas especificações de forma a demonstrar que uma especificação S1 implementa uma outra S2. Elas são correspondentes se cada comportamento externamente visível permitido por S1 também é permitido por S2. A este respeito, existem métodos para provar os chamados *Refinement Mappings* (Abadi e Lamport, 1991), ou seja, provar o mapeamento entre duas especificações e sua equivalência.

Desta forma, as características de uma *Spec* em TLA^+ são aplicáveis ao contexto de validação do HARP, um protocolo parcialmente síncrono¹⁰ e com execuções concorrentes. Em virtude disso, a Seção 4.5 traz a *Spec* do protocolo HARP, escrita em TLA^+ e como ela corrobora as boas propriedades deste protocolo. Além disso, naquela seção é feito um mapeamento entre as duas especificações do protocolo (especificação dos elementos e *Spec* em TLA^+) demonstrando como elas estão relacionadas entre si.

¹⁰Sistemas parcialmente síncronos são aqueles onde podem existir relógios parcialmente sincronizados, limites de discrepância predefinidos ou, ainda, limites aproximados para a troca de mensagens (Lynch, 1996).

2.2 Literatura Correlata

Esta seção apresenta uma revisão de trabalhos correlatos que apontam o estado da arte encontrado por esta pesquisa. Cabe lembrar que o presente trabalho situa-se como um ponto de convergência entre duas áreas bem definidas: hardware reconfigurável e alta disponibilidade. Desta forma, a seção se divide em duas subseções que refletem cada uma destas áreas.

2.2.1 Hardware Reconfigurável para Redes de Computadores

Sob a luz da reconfigurabilidade, os trabalhos mostram o uso do FPGA na resolução de problemas de redes de computadores, seja pelo uso de uma plataforma reconfigurável para desenvolvimento ou por valerm-se de uma arquitetura reconfigurável em um protótipo. Muitos protocolos Internet são implementados em FPGA, pois a possibilidade de reconfiguração de dispositivos fornece os meios para corrigir problemas de implementação ou reagir a variações, refletindo diretamente no conceito de protocolo.

Não há na literatura, até agora, protocolos de alta disponibilidade implementados em hardware. Em virtude disso, primeiramente são analisadas as implementações de outros protocolos Internet para tomar ciência de como suas arquiteturas estão sendo implementadas, bem como se certificar de que o hardware gerado é compatível com as velocidades de transmissão dos canais disponíveis atualmente. Em segundo lugar, faz-se um estudo sobre como as implementações concernes à tolerância a falhas melhoram os requisitos de disponibilidade dos sistemas.

Por outro lado, tangendo à alta disponibilidade, são apontadas aqui pesquisas que analisam a operação dos protocolos de alta disponibilidade, exploram problemas inerentes ao seu funcionamento e trazem dados sobre a execução destes protocolos em cenários específicos.

Jedhe *et al.* (2008) propõe uma arquitetura para classificação de pacotes em aplicações de segurança de rede e implementa um *firewall* com esta arquitetura em hardware reconfigurável. O uso do FPGA é devido ao grande conjunto de regras de classificação dos pacotes que se deve tratar e à necessidade de alta vazão. Apresenta um sistema embarcado baseado em um processador Power PC 32 bits com o módulo *Packet Classification Engine* (para filtrar pacotes segundo inspeção dos campos de cabeçalhos) implementado em FPGA. O módulo é capaz de lidar com uma taxa de 50 milhões pcts/s, o correspondente a 16 Gbps. A implementação foi feita em um FPGA Xilinx Virtex-II Pro XC2VP30 e ocupa 24% das LUTs e 36% dos blocos BRAMs.

Este trabalho mostra que o FPGA é adequado para trabalhar com análise de regras de pacotes e atinge velocidades aceitáveis em relação às tecnologias disponíveis em redes de computadores. Esta característica é aplicável ao cenário do HARP, pois além da sua

característica multi campo das mensagens, as versões futuras acrescentarão ainda mais exigências quanto às regras de classificação dos pacotes.

Em Ravindran *et al.* (2005), os autores desenvolveram um sistema multiprocessado, implementado em FPGA Xilinx Virtex-II Pro, para aplicações de alto desempenho. Um módulo de encaminhamento de pacotes IPv4 foi desenvolvido e incorporado como periférico em um sistema que tem processadores MicroBlaze em seu núcleo. O módulo alcançou uma taxa de transmissão de 1.8 Gbps e ocupa 2% da área total de um FPGA Xilinx Virtex - II Pro 2VP50, onde foi implementado.

Este trabalho deixa claro que outros protocolos da arquitetura IPv4 têm sido implementados em FPGA e que o uso de processadores embarcados e aceleradores de hardware para este fim são uma alternativa factível para sistemas de alto desempenho. O trabalho de Hamerski *et al.* (2007) também é baseado na implementação de protocolos Internet em FPGA, o que permite concluir que esta é uma solução viável de implementação dos protocolos, posto que há a intenção de integrá-los entre si, completando a arquitetura de camadas da Internet.

Kachris e Vassiliadis (2006) desenvolveram um comutador em uma plataforma reconfigurável para resolver o gargalo no encaminhamento baseado em conteúdo. Há coprocessadores implementados em FPGA que auxiliam o processamento em tarefas como gerenciamento de conexões TCP e análise das cadeias de caracteres de *Uniform Resource Locators* (URLs). O esquema proposto com o processador MicroBlaze em FPGA (100 MHz) pode ser programado em C, enquanto as funções de maior demanda são implementados em FPGA e podem ser reconfigurados. O esquema atinge até 927 Mbps de vazão, consumindo menos de 1 Watt em um FPGA Xilinx Virtex4. Ele ocupa uma área de 31% dos blocos lógicos e 71% dos blocos de memória BRAM.

Este trabalho provê uma combinação entre a flexibilidade de processadores e a performance de ASICs. Ele já integra vários módulos de processamento TCP/IP em um único FPGA e ainda indica que módulos adicionais para processamento de carga útil (criptografia, compressão de dados ou detecção de intrusos) poderiam ser acoplados neste mesmo FPGA.

Entende-se destes trabalhos que é indicado implementar protocolos de rede em FPGA, pois pode-se utilizar este FPGA para implementar módulos de rede completos. Todos os trabalhos citados foram desenvolvidos utilizando algum kit de prototipagem com interfaces de entrada e saída conectados ao FPGA, pois isso diminui a lacuna entre a pesquisa teórica e a prototipagem.

O trabalho desenvolvido por Straka e Kotasek (2009) apresenta uma metodologia de desenvolvimento de sistemas tolerante a falhas baseada em FPGA. São arquiteturas baseadas em redundância modular tripla (TMR)¹¹ para melhorar a detecção de falhas.

¹¹Do inglês *Triple Modular Redundancy*. É uma técnica de implementação baseada na replicação de circuitos para garantir resultados corretos.

É mostrado como os parâmetros de disponibilidade¹² podem ser afetados pelo ambiente de funcionamento em que o sistema tolerante a falhas é implementado. A unidade de hardware é triplicada e cada uma delas executa a mesma tarefa. O resultado é determinado por votação. Os autores afirmam que utilizar FPGA oferece novas possibilidades para as atividades que visam sistemas de tolerância a falhas. Em FPGAs, o módulo com defeito pode ser reparado por reconfiguração do chip para que um elemento lógico ou recurso de roteamento danificado não seja usado no novo modelo. Esta idéia de votação é trazida para o processo de eleição do protocolo HARP, no intuito de eliminar falhas pontuais.

Como se pode ver, na literatura encontram-se estudos que tratam a tolerância a falhas paralelamente à alta disponibilidade. Cabe salientar que a alta disponibilidade é um dos objetivos que se espera alcançar através do uso de técnicas de tolerância a falhas, bem como confiabilidade e manutenção da segurança (Johnson, 1988). A tolerância a falhas significa serviços ininterruptos, então quase sempre se refere à energia, discos, rede e outros componentes que podem assumir o papel de um outro dispositivo com problemas.

Nos ambientes sem preservação de estado, que é o caso de protocolos IP e UDP, muitas vezes a alta disponibilidade se confunde à tolerância a falhas, pois toda requisição cliente/servidor é tratada independentemente e as conexões prévias não são mantidas para uso em conexões futuras.

Assim, o trabalho de Straka e Kotasek (2009) ilustra as situações onde a alta disponibilidade é melhorada através de técnicas de tolerância a falhas, especificamente redundância de elementos de hardware. Seguindo esta linha e voltando a atenção para gerência e operação de redes, cabe lembrar que mesmo com enlaces e equipamentos redundantes, não é possível explorar a redundância se não houver mecanismo para redefinir os *gateways* para os nós finais.

2.2.2 Alta Disponibilidade em Redes de Computadores

No caso do trabalho aqui desenvolvido, o foco é o gerenciamento da comunicação entre os equipamentos redundantes, isto é, os protocolos de alta disponibilidade. Os trabalhos referenciados a seguir situam o estado da arte sob este ponto de vista.

O trabalho de Bhagat (2011) trata do rastreamento do enlace entre o roteador que tem o VRRP em execução e o servidor. O VRRP não permite monitoramento deste enlace diretamente, por isso, o autor propõe que ao monitorar o enlace, o roteador mestre com enlace indisponível diminua a sua prioridade para que o escravo com maior prioridade assuma a função de mestre. O trabalho compara o VRRP e o HSRP e conclui que o primeiro é mais eficiente, pois o VRRP tem menor intervalo de percepção de falha (*hello time e holdtime interval*) e é de especificação aberta.

¹²Parâmetros tais como por exemplo tempo médio entre falhas (MTBF - *Medium Time Between Failures*) e taxa de recuperação

O trabalho de Matsuda (2012) implementa um método para diminuir o tempo de convergência do VRRP, de forma que um roteador escravo não necessite esperar pelo intervalo para detectar a falha do mestre. Se aplica a um cenário específico onde há dois roteadores e um comutador. O comutador deve verificar se o seu canal com o mestre está ativo e, em caso de falha, ele envia uma mensagem VRRP de aviso (*advertisement*) que força o escravo a se tornar mestre. Neste caso, um equipamento de camada 2 (IEEE, 2001) deve interferir no processo e enviar uma mensagem do protocolo VRRP. O método não se aplica a outro cenário, ou seja, não se poderia implementá-lo quando os elementos redundantes forem comutadores.

Esta pesquisa situa-se como a continuidade de uma série de trabalhos realizados e publicados por Lopes Filho (2008), Hashimoto (2009) e Pereira Júnior (2010), que descrevem as sequências de hipóteses e conclusões acerca dos problemas de indisponibilidade de rede. As pesquisas citadas concluíram que os problemas residem nas condições de Acéfalo e Cérebro Partido percebidas no protocolo VRRP.

O trabalho apresentado por Lopes Filho (2008) investiga a suspeita de que o principal problema dos protocolos de alta disponibilidade seria a camada de transporte, devido ao uso de protocolos não orientados a conexão, o que levou à proposta de uma camada de transporte baseada no protocolo SCTP. No entanto, ele conclui que o problema não residia na camada de transporte e a hipótese passou a pairar sobre a camada de enlace, devido à possível transmissão para as camadas superiores de mensagens com falso positivo (neste caso, falso positivo é um pacote com erros não reconhecidos pelas camadas mais baixas da rede), o que acarretaria em comportamento inconsistente dos protocolos de camadas superiores.

Hashimoto (2009) atribui as falhas de canal e os erros não detectados pelos algoritmos de controle de erro da camada de enlace como causa para o problema da condição de cérebro partido e conclui pela necessidade de desenvolvimento de um novo protocolo de alta disponibilidade, ou a extensão de um já existente. Foi demonstrado naquele trabalho que o autômato do VRRP é incompleto, pois ele não considera erros não detectáveis pela camada de enlace. O autor afirma que o problema encontrado no VRRP também se aplica aos outros protocolos apresentados na Seção 2.1.2.2. O autor apresenta uma modelagem em redes de Petri que especifica as perdas de mensagens de anúncio em função dos fenômenos da camada de enlace, que resultou em um novo autômato. Este autômato define o comportamento do roteador virtual, mas não o comportamento dos elementos físicos que compõem esta abstração, tornando a proposta abstrata e carente de uma especificação para ser implementada.

Pereira Júnior (2010) discute as condições concorrentes para as situações de Acéfalo e Cérebro Partido e define uma especificação de serviço que compõe o projeto de um protocolo de alta disponibilidade. O trabalho apresenta suposições de um ambiente onde um serviço de alta disponibilidade deve operar e como os protocolos de alta disponibilidade

podem resolver os desafios que surgem nesses ambientes.

2.3 Conclusão do Capítulo

Estes trabalhos relacionados permitiram constatar que a especificação de um novo protocolo de alta disponibilidade se fazia necessária devido às inconsistências dos protocolos existentes. Além disso, mostram que sistemas de rede têm sido construídos com hardware reconfigurável e comprovam tanto o baixo custo do desenvolvimento quanto o bom desempenho destes sistemas.

Fica evidente, então, que a existência e a adequação de uma plataforma baseada em FPGA vem ao encontro da necessidade de desenvolvimento do HARP. Uma implementação em hardware permite um nível de detalhamento do protocolo que não se pode obter em análises somente teóricas. A característica da reconfigurabilidade do FPGA habilita o desenvolvimento de protótipos a cada versão, permitindo que se atenda às necessidades vindouras das pesquisas em Internet do Futuro.

Haverá um hardware que poderá ser reconfigurado e, conseqüentemente, ter sua arquitetura atualizada para as versões futuras do HARP. Neste ponto, Hauck e DeHon (2010) ratificam que sistemas de rede criados com hardware reconfigurável são flexíveis e facilmente modificáveis para prover novas funcionalidades.

Método de Pesquisa

Um método de pesquisa deve ser adotado para direcionar o trabalho de forma lógica (Lakatos e Marconi, 2003). Para Wazlawick (2009) o método consiste na sequência de passos necessários para demonstrar que o objetivo proposto foi atingido, i.e., se os passos definidos no método forem executados, os resultados obtidos deverão ser convincentes. Neste contexto, este capítulo se atenta ao método adotado para o desenvolvimento do HARP e tem como principais contribuições (i) descrever a sequência de passos realizados na construção desta pesquisa, bem como (ii) delinear seu escopo.

É importante que o objetivo do trabalho esteja bem definido para que se tome qualquer decisão relacionada ao método. Assim, uma vez que este trabalho pretende desenvolver um protocolo livre das situações de Acéfalo e Cérebro Partido dos protocolos de alta disponibilidade, o método deve contemplar as etapas de especificação, implementação e teste de um novo protocolo, bem como um sistema capaz de demonstrá-lo em funcionamento.

De acordo com as classificações de Wazlawick (2009), esta pesquisa tem caráter exploratório e experimental e segue a vertente de apresentação de “algo presumivelmente melhor”, onde o próprio autor do trabalho cria e realiza os testes que demonstram que sua abordagem é melhor que as outras. Quanto ao método de abordagem, a pesquisa é construída de acordo com o que Lakatos e Marconi (2003) classificam como Método Hipotético-Dedutivo, que reflete o processo de percepção de um problema, proposição de uma solução e realização de testes pela observação e experimentação.

A Figura 3.1 traz um fluxograma que sintetiza o método e descreve visualmente o processo de construção de conhecimento aqui realizado. Percebe-se, pela figura, três etapas principais e estas, por sua vez, são subdivididas em tarefas elencadas ora de forma sequencial ora simultaneamente.

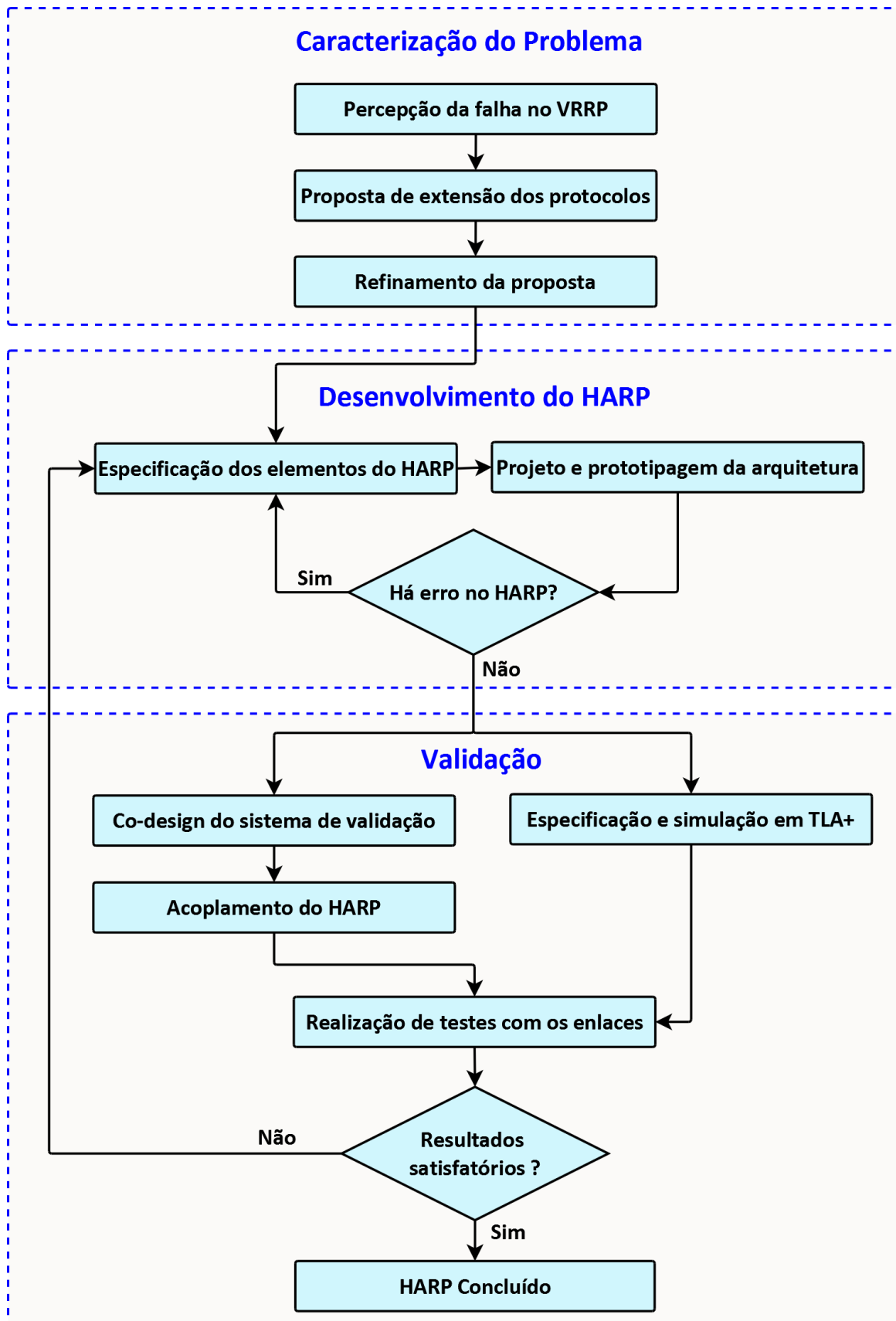


Figura 3.1: Método: fluxograma de etapas

A etapa de Caracterização do Problema estabelece o vínculo entre a percepção da indisponibilidade na rede e a necessidade de um novo protocolo de alta disponibilidade, o que também inclui os trabalhos que precederam a este e a necessidade de refinar a proposta existente. Já as duas últimas etapas incluem unicamente as contribuições desta dissertação e se voltam para a especificação, implementação e demonstração do *High Availability Router Protocol* (HARP). O detalhamento de cada uma destas etapas é feito nas seções subsequentes.

3.1 Caracterização do Problema

Não fugindo aos padrões clássicos da academia, esta pesquisa se iniciou com a percepção de um problema. Segundo Hashimoto (2009), situações de indisponibilidade foram percebidas no tráfego mantido por equipamentos de rede que operavam com o protocolo VRRP. Ao observar a situação, Hashimoto (2009) reproduziu o fenômeno e foram realizadas coletas e análises do tráfego de *Firewall* e *Intrusion Prevention Systems* (IPS) operando em ambiente de teste no *Data Center* da empresa de telecomunicações Algar Telecom.

As condições de Acéfalo e Cérebro Partido (Subseção 2.1.2) foram apontadas como causas do problema e percebeu-se que a segunda era mais frequente. Nos dois casos as requisições de serviços não eram respondidas ou eram respondidas por mais de um equipamento e, neste último caso, a duplicação de respostas não só gerava uma sobrecarga de mensagens impossível de ser gerenciada, como também deixava a rede com referência inconsistente de *gateway*.

Após estabelecer as causas da indisponibilidade e apontá-las como sendo problemas algorítmicos inerentes ao protocolo, Hashimoto (2009) apresentou uma especificação abstrata, modelada em rede de Petri (Petri, 1962) e representada por um autômato para contornar as condições citadas. O autômato não foi implementado e trata-se de uma descrição na mesma filosofia da especificação em rede de Petri: a visão do comportamento do elemento virtual (Subseção 2.1.2). Entretanto escapou-lhe a especificação do comportamento individual de cada um dos elementos físicos que compõem o elemento virtual.

Em virtude disso e para que se pudesse partir para a implementação de um protocolo que resolvesse estes problemas, fez-se necessária uma especificação dos cinco elementos (Holzmann, 1991) de cada instância deste protocolo e em um nível de abstração mais baixo. Foi necessário refinar a proposta de Hashimoto (2009) (que também aparece em (Hashimoto *et al.*, 2010)) a fim de gerar este novo protocolo, implementá-lo e validar o seu funcionamento. O processo de refinamento se inicia quando da constatação de que a proposta não é implementável e é concluído com a especificação do protocolo HARP. Os elementos listados a seguir mostram o porquê de a proposta carecer de refinamento:

- Não há especificação de serviço.
- Elementos a serem considerados no autômato, como por exemplo condições para transições entre estados, suas entradas, saídas e o vocabulário não são especificados.
- Na rede de Petri descrita por Hashimoto (2009), é dito que caso o mestre falhe, habilita-se a sequência de transições do estado *Standby* para um estado de *Disponível* e logo após para *Master*. Entretanto, não há regras de como o processo de eleição é feito. Ademais, o autor diz que "o estado S0 representa o estado inicial do autômato, onde podem existir equipamentos esperando para se tornar *Master*", i.e., uma abstração que não contempla cada elemento físico.
- O estado S2 representa um equipamento designado como *Master* e outro como *Standby*. Mas do ponto de vista de cada um destes dois equipamentos, eles estarão em estados diferentes (um mestre e um escravo), não se pode apenas chamar os dois estados de S2. Isso é uma questão de ponto de vista, pois olhar para o grupo como sendo o elemento virtual não dita as regras de como cada elemento físico deve agir.
- O S10 é um exemplo de estado desnecessário (sob a luz do elemento físico) pois é um estado visitado pelo *Master* para enviar mensagens de aviso *advertisements*. Um equipamento mestre não precisa ir a outro estado específico para enviar este tipo de mensagem, basta que ele seja mestre e ativo.

A partir destes exemplos, percebe-se que os elementos cruciais para a especificação do protocolo devem ser elaborados. Neste cenário é que nasce o *High Availability Router Protocol* (HARP), um protocolo que apresenta as regras de comportamento dos elementos físicos que integram um grupo de HA.

3.2 Desenvolvimento do HARP

Nesta fase ocorreu efetivamente a especificação do HARP, bem como a sua implementação e prototipagem em hardware. A especificação completa de um protocolo requer que cinco elementos sejam tratados: suposições sobre ambiente, serviços, vocabulário, formato e regras de procedimento (Holzmann, 1991). A especificação destes elementos ocorreu paralelamente à sua prototipagem em hardware, conforme a Figura 3.1. Os elementos do HARP são apresentados detalhadamente no Capítulo 4 e sua arquitetura no Capítulo 5, já as características de implementação para prova de conceito são tratadas aqui.

Para esta etapa e para a validação, as ferramentas utilizadas para a construção dos protótipos são as seguintes¹:

¹Ferramentas Altera®. Disponível em: <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html>

- Quartus II versão 7.2, para edição de códigos em linguagem de descrição de hardware (HDL), síntese e configuração do FPGA;
- Megacore IP versão 7.2, biblioteca fornecida pela Altera que contém os *Intellectual Property Cores* que serão utilizados no sistema, por exemplo o processador embarcado e os controladores de memórias SRAM e *Flash*.
- NIOS II IDE versão 7.2, ambiente de desenvolvimento para construir projetos cujos softwares serão executados sobre o NIOS II. Inclui o Nios II *Viewer* para visualização do processamento dos dados no software embarcado.
- Modelsim Altera-Starter Edition versão 6.1, para simulação funcional dos elementos de hardware

É importante ressaltar que este projeto não funciona plenamente nas versões posteriores à 7.2, pois há incompatibilidade destas versões com a versão da pilha de protocolos TCP/IP implementada pelo controlador Ethernet. Nas versões posteriores, o TCP/IP inclui a pilha de rede do *Nichestack*, entretanto, este projeto foi construído sobre a pilha *LightWeight IP*. O *device driver* para o controlador Ethernet DM9000A é diferente para cada uma delas. Assim, a menos que seja escrito um novo *device driver* para o controlador do chip DM9000A, o projeto não será compatível com a pilha TCP/IP versão *Nichestack*.

A escolha por uma implementação em hardware foi motivada por dois fatores principais. O primeiro deles é que a implementação em hardware permite a detecção de detalhes e correção de falhas que escapam ao projeto de protocolos em níveis mais elevados de abstração. Já o segundo é que em se tratando de protocolos de rede, o paralelismo espacial do hardware adequa-se cabalmente à característica de mensagens multicampo, eliminando assim a latência incluída pelo processamento sequencial do software. Ainda que desempenho não seja um dos requisitos do protocolo nesta etapa do desenvolvimento, prototipá-lo em hardware facilita a continuidade do processo, uma vez que, em trabalhos futuros, pretende-se prototipar os outros protocolos da arquitetura Internet e mais módulos de segurança e acoplá-los para formar um equipamento de alta disponibilidade.

No entanto, o custo de produção de um hardware específico é muito alto. Por isso, o uso de um dispositivo flexível faz-se necessário, sendo o FPGA o mais indicado para esta situação. O uso do FPGA possibilitou que correções fossem feitas após cada prototipagem do HARP, permitindo perceber erros do hardware e retornar à sua especificação para realizar as correções. Este processo de iterações permitiu especificar o HARP e prototipar um hardware que atendesse às necessidades de um protocolo de alta disponibilidade. Além dos benefícios da plataforma reconfigurável para desenvolvimento, ter uma arquitetura final reconfigurável permite efetuar as mudanças necessárias para um hardware compatível com IPv6 ou que atenda exigências da Internet do Futuro.

Como descrito no Capítulo 4, o HARP vale-se de cinco serviços para manter a característica de alta disponibilidade na rede. Cada um deles foi especificado, implementado e

testado antes que se começasse a especificar um outro. Estas iterações refletiram diretamente na evolução da arquitetura, para que ela comportasse cada vez mais serviços. Este é o ponto que define a simultaneidade entre as tarefas (1) Especificação dos elementos do HARP e (2) Projeto e prototipagem da arquitetura da etapa de *Desenvolvimento do HARP*, na Figura 3.1.

Uma plataforma foi montada para realizar a prova de conceito do HARP, conforme ilustrado pela Figura 3.2. Para tal, foi montado um esquema com três placas de prototipagem DE2 (Altera, 2006) simulando elementos de rede (NE)² conectados entre si em topologia estrela (Soares *et al.*, 1995). Cada elemento de rede possui um FPGA de baixo custo Cyclone II como núcleo de processamento. O FPGA recebe e envia dados a qualquer um dos outros através de conexão dedicada sobre um cabo *flat* de 36 vias conectado a um dos cabeçalhos de expansão contidos na placa de prototipagem. Estas 36 vias são divididas em três canais de doze vias, onde cada um destes conecta um bloco receptor de um elemento de rede(NE) a um bloco emissor de outro NE.

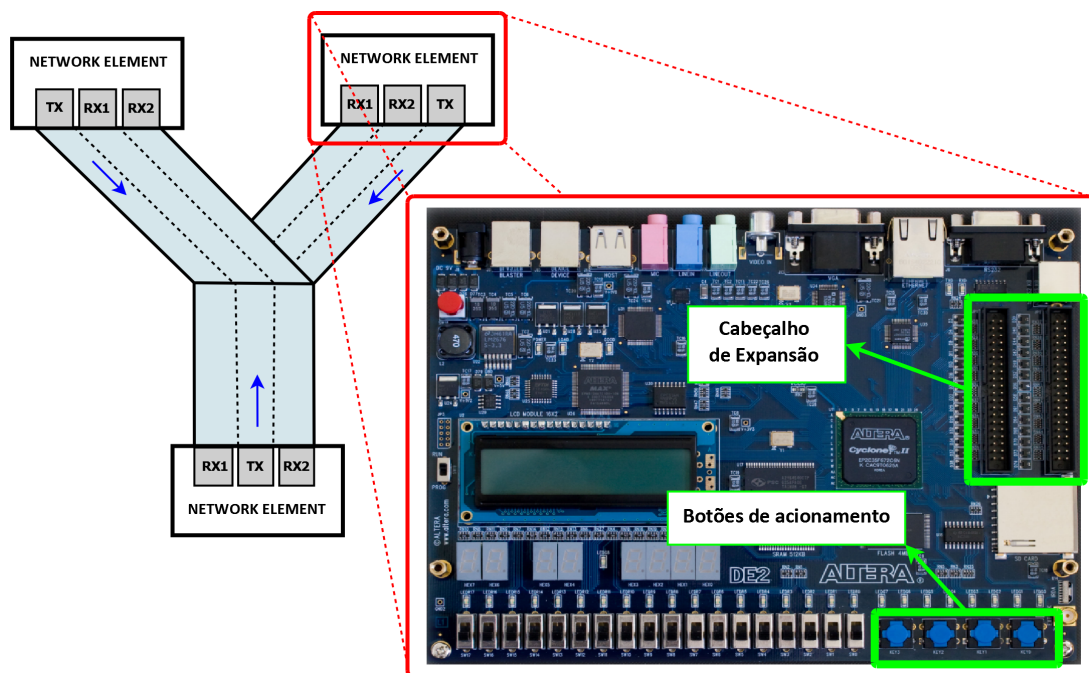


Figura 3.2: Esquema de conexão para a prova de conceito

Os testes neste caso são realizados com uma versão do HARP adaptada para experimentação. O endereçamento não é feito no formato IP, mas sim uma adaptação para a interconexão mostrada pela figura 3.2. As mensagens não têm nenhum tipo de encapsulamento por trafegarem em enlaces diretos e dedicados. O formato das mensagens é de apenas 12 bits, pois podem ser transmitidas em um só frame pelo canal, e carregam informações de endereço e tipo de mensagem.

²Do inglês *Network Elements*.

Cada solicitação de serviço é feita ao se pressionar um dos quatro botões de acionamento conectados ao FPGA (*pushbuttons* do kit DE2). Os serviços foram testados, i.e., as trocas de mensagens e estados foram confirmadas exatamente como previstas. Após uma sequência de iterações entre especificação e prototipagem, o sistema demonstrou o funcionamento do HARP conforme sua especificação.

Dado que o HARP é baseado em *timeouts* (Seção 4.4), como é o caso dos estados S2, S3, S4 e S6 da Figura 4.5, inserir atrasos entre estados e atender todos os possíveis casos de perdas de primitivas foram os principais benefícios da reconfigurabilidade do FPGA, que permitia revisar o HARP em cada implementação.

Como dito anteriormente, a Figura 3.2 representa a prova de conceito do HARP, que foi concluída com êxito. Entretanto, ainda não trata do tempo de recuperação do sistema e nem traz dados sobre o tempo que cada primitiva leva para ser processada dentro de cada instância do HARP.

A adaptação do HARP nesta etapa foi feita para possibilitar um sistema de prototipação simples que reduzisse o tempo de projeto. Este sistema não deveria introduzir nenhuma complexidade adicional ao processo de desenvolvimento, por exemplo ao utilizar interfaces com circuitos controladores que precisem de gerência. Percebe-se pela Figura 3.2 que nenhuma interface com redes externas é feita. Este ponto é retomado na etapa de validação, onde é apresentado um sistema mais complexo para demonstração de protocolos de alta disponibilidade.

O sistema da Figura 3.2, apesar de ser suficiente para a prova de conceito do protocolo HARP, não é expansível e também não há nenhuma interação com as camadas inferiores da arquitetura Internet, uma vez que não há qualquer tipo de interação com interfaces para sistemas externos. Depois da prova de conceito ter sido feita, um sistema de validação foi construído e o HARP foi submetido à experimentação com enlaces Ethernet.

3.3 Validação

Esta é a última etapa da pesquisa e contribui para o processo de duas formas. De um lado, foi elaborada uma especificação formal do HARP utilizando a linguagem de descrição formal TLA^+ (Lamport, 2003), que permite analisá-lo sob a ótica de sistemas paralelos e concorrentes. De outro, um sistema de validação em hardware e software foi coprojetado para demonstrar o funcionamento do HARP em sua forma padrão e permitir sua integração com outros protocolos.

Apesar de o diagrama de estados descrever completamente o comportamento do HARP, TLA^+ permite descrever protocolos consensuais para execução de forma paralela, simular a execução de diversas instâncias deste protocolo e encontrar falhas na execução dos seus processos. O resultado desta especificação é retomado na Subseção 4.5.

Quanto ao cenário experimental para validação, este é fornecido nas subseções a se-

guir. Uma visão geral do ambiente de experimentação é dada, incluindo equipamentos utilizados, a configuração interna de cada elemento de rede e uma descrição dos processos avaliados.

3.3.1 Sistema de Validação

Após a prova de conceito do HARP, fez-se necessário observar sua operação em um cenário mais próximo de aplicações reais. Neste sentido, foi construído um sistema que integra hardware e software em uma mesma plataforma e é capaz de estabelecer a comunicação entre diferentes instâncias do HARP em execução. O HARP está implementado conforme regras do Capítulo 4, utiliza Ethernet como enlace de dados e é encapsulado segundo regras dos protocolos MAC e IP.

Este sistema deve demonstrar a capacidade de o protocolo se recuperar em virtude de falhas nos enlaces. Ele não trata de possíveis ataques de invasores, sendo este um tópico para desenvolvimentos futuros. Protocolos que preservam o estado (*stateful*) também estão fora do escopo deste trabalho que, por sua vez, trata de protocolos sem preservação de estado (*stateless*), como é o caso do IP. É importante citar que para esta pesquisa, o VRRP tem sido o *benchmark*.

A Figura 3.3 ilustra a topologia do sistema de validação construído, de acordo com os recursos disponíveis no Laboratório de Redes de Computadores do PPGCC/UFU. São utilizados três computadores pessoais, três placas de prototipagem DE2 (Altera, 2006) e um comutador camada 2. Em cada um dos computadores, o software Wireshark (2013) executa a captura de pacotes que chegam e saem das suas interfaces.

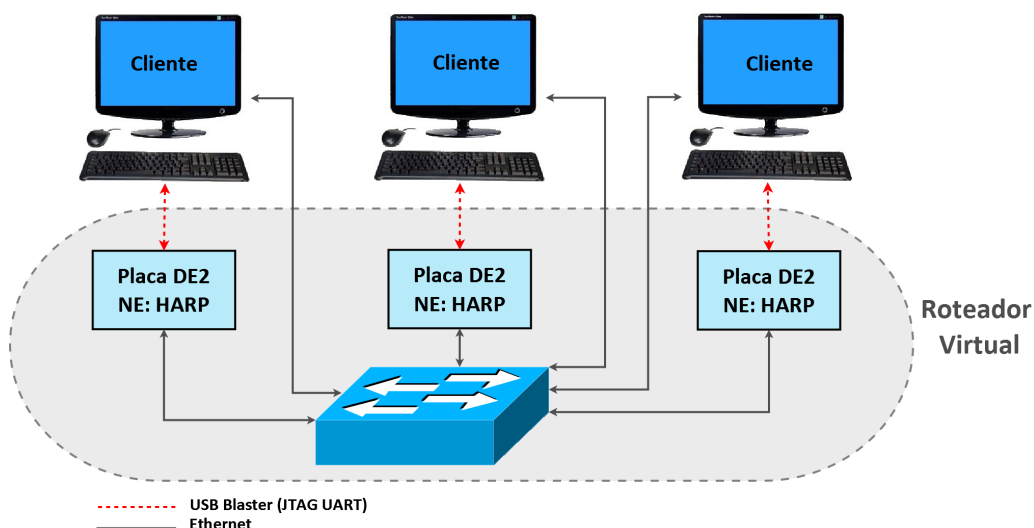


Figura 3.3: Topologia do sistema de validação do HARP

Cada computador está conectada ainda a uma das placas DE2 através de uma conexão serial JTAG UART. Esta via serve de caminho de dados entre o processador e o aplicativo

NIOS II *Viewer*. Este aplicativo, que é parte da ferramenta NIOS II IDE, exibe na tela do computador o processamento em software realizado na placa. Ele recebe os dados do processador através da conexão serial JTAG UART e, assim, é possível acompanhar em tempo real as mensagens que chegam e saem das placas.

Cada uma das placas DE2 na Figura 3.3 representa um elemento físico de rede que compõe o roteador virtual. Neste experimento, o comutador é considerado também parte do roteador virtual, mas não como um elemento do grupo de alta disponibilidade, pois serve apenas como uma forma de conectar todas as interfaces entre si, dada a limitação de que cada placa e cada PC têm apenas uma interface Ethernet.

Em cada elemento de rede há uma instância do HARP em execução. Uma vez que ele está implementado em hardware e a interface Ethernet não pode ser acessada diretamente desta forma, um *System-on-Programmable Chip* (SoPC) (Fei-yu *et al.*, 2010) foi configurado para permitir a interconexão entre NEs e consequentemente a demonstração do HARP. A Figura 3.4 representa o diagrama de blocos do SoPC, bem como a integração do HARP a este sistema. O SoPC é baseado no processador NIOS II (Altera, 2013b), um RISC com arquitetura de 32 bits, cuja versão utilizada foi a *Standard NIOS II fast* (Altera, 2013c). Os blocos possuem interfaces *M*, *S* e *E*, que significam interface *Master*, *Slave* e *External*, respectivamente.

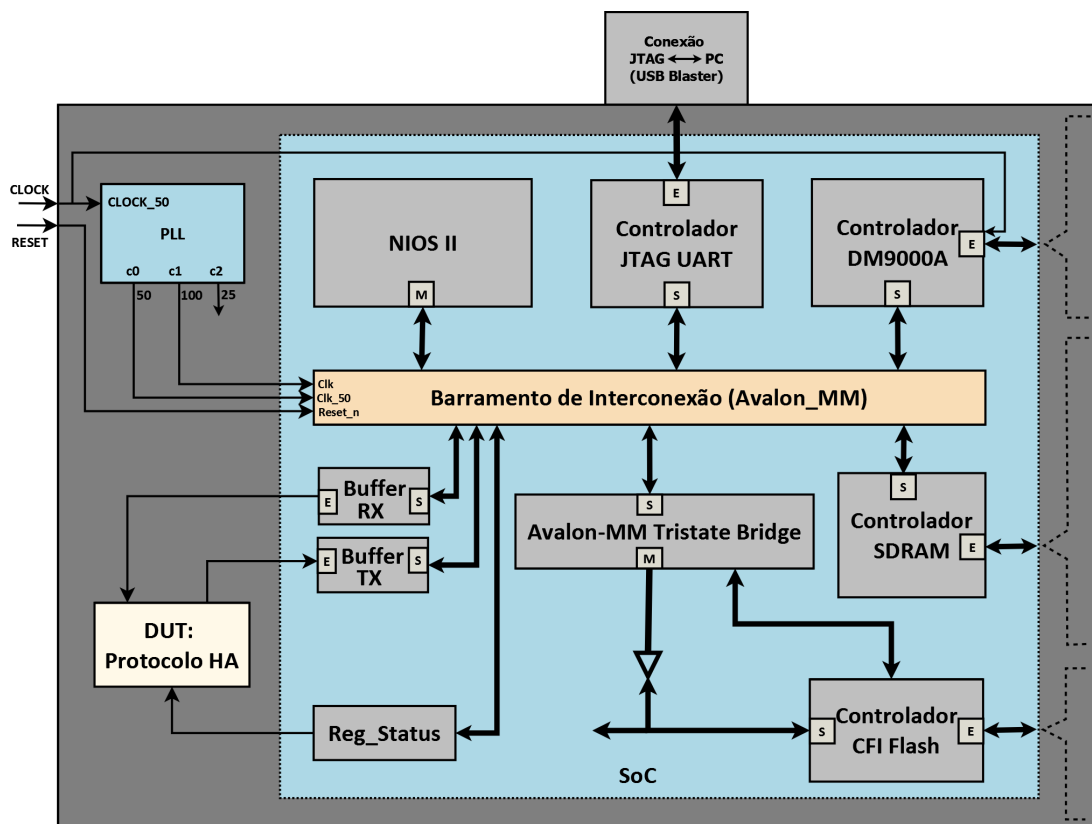


Figura 3.4: SoPC em cada elemento de rede

O JTAG UART é o módulo que permite a comunicação com o computador, para exibir

os dados processados pela aplicação sendo executada neste sistema. O *Barramento de Interconexão* é o controlador do barramento interno Avalon® (Altera, 2013a), e recebe os sinais de *clock* vindos do PLL para conectá-los de acordo com as especificações de frequência de cada chip³. O NIOS II opera a 100 MHz.

Os blocos *Controlador CFI Flash* e *Controlador SDRAM* são os controladores de memória. O primeiro se conecta a uma memória *Flash* de 4 MB para armazenar as configurações da aplicação de integração do SoPC quando carregadas na placa (recebe o endereço do *reset vector* do processador). Já o segundo se conecta a um chip de memória SDRAM de 8 MB que recebe o *exception vector* do processador e é onde o programa é carregado para execução. O *Avalon-MM Tristate Bridge* controla o uso de memórias *flash*, de tal forma que caso se tenha mais de uma memória em utilização ele é o árbitro de barramento entre elas.

O *Controlador DM9000A* é o bloco que se comunica com o chip DM9000A (Davicom, 2005b). Este chip contém o *Physical Layer* (PHY) e MAC da interface Ethernet da placa DE2. Ele não é um elemento nativo da biblioteca Megacore IP e, portanto, teve que ser criado e inserido manualmente. Durante o desenvolvimento, houve dificuldades para acessar e gerenciar este componente. Não foi possível acessá-lo diretamente sem o uso do processador devido ao grande número de registradores internos que devem ser configurados e os manuais de utilização (Davicom, 2005a) (Davicom, 2005b) não trazem informações claras sobre a sequência e temporização dos comandos no nível do hardware. Isso levou à necessidade de integrá-lo à biblioteca da ferramenta SoPC Builder. O Apêndice C explica como configurar este elemento para ser inserido nesta biblioteca.

Os módulos *Buffer RX*, *Buffer TX* e *Reg-Status* compõem uma camada de abstração de hardware para o HARP se comunicar com sistemas de uma forma geral. O primeiro recebe do sistema as mensagens destinadas ao HARP. Já o segundo envia as mensagens geradas pelo HARP ao sistema. Por fim, o terceiro armazena o estado da conexão do enlace Ethernet. Estes blocos devem obedecer às exigências de cada sistema, de forma a garantir a compatibilidade com o barramento. Como exemplo disso, neste caso o *REG-Status* é implementado pelo registrador interno *Network Status Register* (NSR) do chip DM9000A, assim, o NIOS II deve ler o conteúdo deste registrador e copiar para o *Reg-Status* através do barramento Avalon. Devido a estas restrições, eles também são criados e adicionados à biblioteca.

O HARP está inserido como um módulo de hardware externo ao SoPC, para processar somente as mensagens de entrada destinadas à comunicação do protocolo de alta disponibilidade, emitindo ou não resposta, dependendo da situação especificada no protocolo. Na Figura 3.4, ele é prototipado onde o módulo DUT está conectado. Este módulo significa

³Um controlador é a lógica digital implementada que se conecta ao chip físico de determinado componente. Por exemplo, o Controlador DM9000A é o módulo no SoPC que se conecta externamente ao chip DM9000A da placa de prototipagem.

Design Under Test e é onde o protocolo é acoplado para ser testado.

A interoperabilidade entre os módulos funcionais do SoPC da Figura 3.4 é controlada por uma aplicação escrita em linguagem C e executada pelo processador NIOS II. Esta aplicação é que realiza, por exemplo, o encaminhamento dos dados endereçando-os a cada módulo de hardware. A arquitetura desta aplicação é explicada na subseção a seguir.

3.3.2 Características da aplicação de integração

Para fazer a interface entre o hardware do HARP e os canais de comunicação via Ethernet, uma aplicação foi construída e é executada sobre o processador NIOS 2. A execução da aplicação influencia no tempo de convergência do protocolo neste cenário. A discussão deste custo será retomada no Capítulo 6.

A aplicação de integração, intitulada *IntegraSoPC*, está codificada em linguagem C e foi desenvolvida utilizando o NIOS II IDE versão 7.2. As instruções e funções que a compõem fazem referência a arquivos de configuração do controlador Ethernet DM9000A e de descrição do SoPC, que são incluídos no ambiente de desenvolvimento. A ferramenta *SoPC builder* (nativa do Quartus II) gera os arquivos de descrição do SoPC, que incluem, por exemplo, os endereços para acesso a cada um dos módulos acoplados ao barramento.

A aplicação *IntegraSoPC* recebe os dados gerados pelo HARP e os encapsula conforme especificações da arquitetura Internet. Com o pacote pronto para o envio, a aplicação o entrega à interface Ethernet, através do controlador DM9000A. O fluxo oposto acontece quando uma mensagem chega pela rede. A aplicação *IntegraSoPC* recebe a mensagem, desencapsula-a e verifica se ela se destina àquele NE. Em seguida, a mensagem é formatada conforme a interface do HARP e então entregue ao módulo de alta disponibilidade.

A Figura 3.5 apresenta a arquitetura da aplicação *IntegraSoPC*, segundo seus blocos funcionais. Ela é composta por uma sequência de passos para configuração do elemento de rede, tanto do ponto de vista arquitetural quanto das definições de rede, obedecendo padrões dos protocolos *Address Resolution Protocol* (ARP) e IP. Há o laço principal, que é onde a aplicação se mantém em execução permanente, incluindo a verificação do estado da conexão Ethernet e as rotinas controladoras de interrupção do módulo HARP e do controlador DM9000A.

As rotinas controladoras de interrupções gerenciam o envio e recebimento de mensagem entre os elementos do grupo de alta disponibilidade neste cenário. A transmissão de cada uma das mensagens HARP entre os elementos de rede obedece a uma sequência de etapas. Esta sequência está destacada na Figura 3.5 e é composta pelos seguintes blocos:

1. *Configura IP*: Envia uma mensagem ao módulo HARP, informando-o sobre o endereço IP atribuído a este elemento de rede.
2. *Configura String TX*: Atribui elementos como IP, MAC e comprimento de mensagem à estrutura que receberá a mensagem HARP em seu corpo de dados.

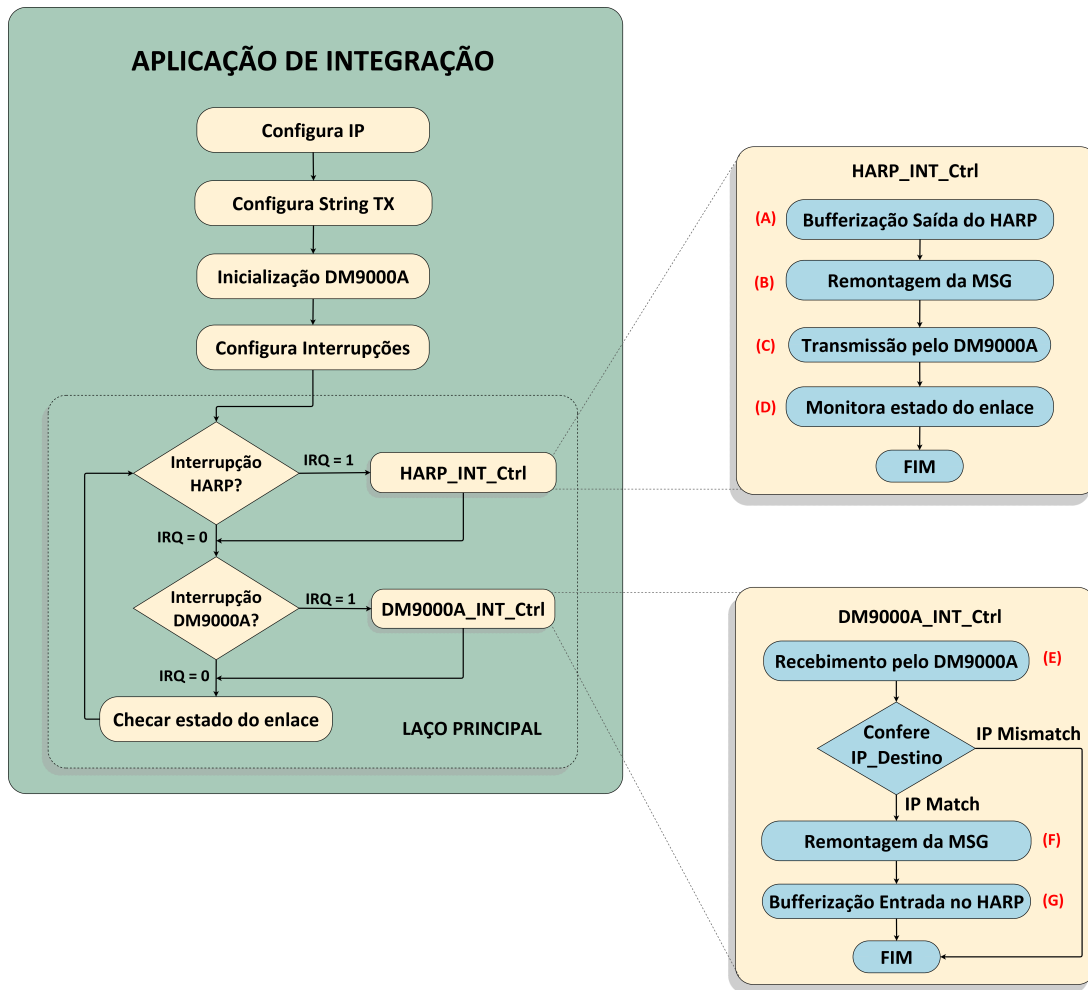


Figura 3.5: Arquitetura da aplicação *IntegraSoPC*

3. *Inicialização DM9000A*: Executa a rotina de inicialização do chip DM9000A, que contém o PHY e o MAC da interface Ethernet.
4. *Configura Interrupções*: Avisa ao processador que ele deve atender as interrupções geradas por dois dos módulos acoplados a ele, o Controlador DM9000A e Buffer TX. Após estas configurações, a aplicação está pronta para entrar no laço principal.
5. *Laço Principal*: A aplicação se mantém constantemente verificando a ocorrência de interrupções e checando o estado do enlace de rede. As manipulações de interrupção compreendem o envio e recebimento de mensagens. Tais funções de manipulação são formadas por:

- 5.1. *HARP_INT_Ctrl*: Controlador de interrupções geradas pelo HARP. Ele sempre será invocado quando o HARP gerar uma mensagem e ela estiver pronta para ser enviada. Desde o momento do atendimento da interrupção até o envio da mensagem, o seguinte fluxo é obedecido:

- *Bufferização Saída do HARP (A)*: Recebe os frames gerados pelo HARP,

uma vez que o canal de comunicação com o barramento Avalon é de 32 bits e a mensagem HARP tem 128 bits de comprimento.

- *Remontagem da MSG* (B): Redistribui os caracteres recebidos em frames para o campo de dados da *string* de transmissão.
- *Transmissão pelo DM9000A* (C): Carrega a mensagem a ser transmitida na SRAM interna do chip DM9000A e espera pela sua transmissão.
- *Monitora estado do enlace* (D): Lê o registrador *Network Status Register* (NSR) do chip DM9000A e escreve o valor no bloco *Reg_Status*, para atualizar o do enlace.

5.2. *DM9000A_INT_Ctrl*: Controlador de interrupções geradas pelo DM9000A. Será sempre executado quando uma mensagem chegar pela interface Ethernet e obedece às seguintes etapas:

- *Recebimento pelo DM9000A* (E): Espera pelo recebimento completo da mensagem. Confere a existência de erro ao nível de enlace. Copia a mensagem para um vetor.
- *Remontagem da MSG* (F): Redistribui os caracteres da mensagem recebida na forma de frames.
- *Bufferização Entrada no HARP* (G): Entrega os frames ao HARP.

5.3. *Checar Estado do Enlace*: Executa novamente o item *Monitora estado do enlace* (D).

A Figura 3.5 possui marcações de (A) a (G) que rotulam os contadores de T_A a T_G , referentes à cada etapa. Estes contadores indicam o tempo que cada uma das etapas consome no processo de transmissão de uma mensagem entre os módulos do HARP nos diferentes elementos de rede. Algumas etapas ocorrem de forma sequencial, enquanto outras acontecem simultaneamente. Por exemplo, parte das etapas C e E acontece simultaneamente e parte não. Enquanto isso, a etapa D é executada depois da etapa C e concomitantemente às etapas E e F. Estes dados são úteis para analisar o desempenho do HARP no sistema de validação construído (Figura 3.3) e, por isso, serão retomados no Capítulo 6 que trata da análise dos dados experimentais.

3.3.3 Processos Avaliados

As situações a serem testadas neste experimento estão relacionadas à indisponibilidade do mestre do grupo, por razões previstas ou não e em momentos esperados ou não. Alguns processos são monitorados para demonstrar a capacidade de recuperação do HARP em face às falhas de enlace. Os processos a serem monitorados no experimento são descritos na Tabela 3.1. A partir deles, infere-se variáveis experimentais (E) e variáveis medidas

(M) para controlar a análise dos resultados. Cinco processos foram enumeradas para serem observadas e medidas.

Tabela 3.1: Processos a serem avaliados

Nome	Variáveis	Descrição
<i>P1</i>	<i>E1: Transferência do Mestre</i> <i>M1: Novo Mestre resolvido</i>	Resposta ao serviço de transferência espontânea da função de mestre.
<i>P2</i>	<i>E2: Queda do Mestre</i> <i>M1: Novo Mestre resolvido</i>	Eleição de novo mestre a partir da queda do atual.
<i>P3</i>	<i>E3: Queda do canal do Mestre</i> <i>M1: Novo Mestre resolvido</i>	Eleição de novo mestre a partir da queda do canal de comunicação do mestre atual.
<i>P4</i>	<i>E4: Queda do Escravo</i> <i>M2: Escravo permanece Escravo</i>	Caso haja falha somente em um escravo, garantir que ele não se auto elegerá mestre.
<i>P5</i>	<i>E1: Queda do canal do Escravo</i> <i>M2: Escravo permanece Escravo</i>	Caso haja falha no canal de somente um escravo, garantir que ele não se auto elegerá mestre.

Para gerar as situações descritas de P1 a P5, deve-se alterar o valor atual das variáveis experimentais relacionadas a cada um deles e analisar as modificações das variáveis medidas. Cada processo tem um modo diferente de ser iniciado.

O processo P1 refere-se à situação onde o equipamento mestre do grupo de alta disponibilidade deve ser retirado para manutenção programada. Neste caso, a variável E1 (*Transferência do mestre*) deve ser setada para iniciar o processo. Como resultado, espera-se que somente o escravo escolhido (por algum parâmetro definido pelo administrador) assuma o papel de mestre, o que gera o valor 1 para a variável M1 (*Novo mestre resolvido*). Qualquer outro resultado que seja mais ou menos que um mestre no fim deste serviço, o valor de M1 será considerado não satisfatório (zero).

Já no processo P2, avalia-se a eleição de um novo mestre a partir da queda do atual. Como resultado, espera-se praticamente o mesmo do processo P1 (variável M1 na saída) diferenciando-se no fato de que agora não se pode predizer qual o escravo que assumirá como mestre. Para o processo P3, espera-se os mesmos resultados conseguidos em P2, a diferença neste caso está na causa do problema. Em P3, a falha é no enlace de comunicação, enquanto em P2 a falha pode ser no equipamento.

No processo P4, há uma situação onde o escravo tem problemas no recebimento das mensagens, o seu módulo de recepção falhará. Isso será feito alterando E4 (*Falha de escravo pontual*) e a verificação será pela variável M2 (*Escravo permanece escravo*), que deve assumir o valor um para indicar que o escravo continua escravo e não se autoelegeu mestre. Em P5, o resultado esperado é o mesmo descrito para P4.

Os processos são iniciados por interferência humana, para forçar os problemas nas interfaces, bem como iniciar o processo de transferência espontânea de mestre.

3.4 Resumo do Método

Em suma, o método adotado para o desenvolvimento deste trabalho é dividido em três etapas: (1) Caracterização do Problema, (2) Desenvolvimento do HARP e (3) Validação. É oportuno voltar à Figura 3.1 e ver como estas etapas estão elencadas ao longo do método.

A primeira delas se inicia em trabalhos que precederam a este, quando se percebeu o problema da indisponibilidade em uma empresa de telecomunicações. Em seguida, após uma sequência de hipóteses e conclusões parciais, o protocolo de HA foi apontado como a causa da indisponibilidade. Foi modelada, então, o que Hashimoto (2009) chamou de proposta de extensão para o VRRP, que contempla erros de mensagens durante sua operação. É neste ponto que o presente trabalho começa a atuar. Fez-se uma análise da proposta e percebeu-se que ela não ditava as regras para um elemento físico e também não era implementável. Partiu-se então para a segunda etapa.

Durante a segunda etapa, houve a especificação dos cinco elementos do HARP, o projeto da sua arquitetura e a prototipagem de uma versão de testes desta última. O processo se caracterizou por iterações entre a especificação e a prototipagem. A inclusão de cada serviço na especificação do HARP incorreu na atualização do seu vocabulário e formato, refletindo diretamente na organização dos módulos que implementam esta arquitetura. Desta forma, esta etapa se caracterizou pela sequência especificação/prototipagem em hardware reconfigurável. Seguidamente, partiu-se para a construção do sistema de validação e a verificação formal das boas propriedades do protocolo.

Por fim, a etapa de validação é caracterizada pelo co-projeto do sistema de validação e pelo desenvolvimento de uma especificação formal para o protocolo HARP. As etapas do *co-design* do sistema de validação são resumidas na Figura 3.6. Os projetos de software e hardware a que esta figura se refere são, respectivamente, o SoPC da Figura 3.4 e a aplicação de integração *IntegraSoPC*, da Figura 3.5.

Paralelamente ao coprojeto do sistema de validação, a especificação formal em TLA^+ foi construída para demonstrar que o HARP mantém as boas propriedades de um protocolo para execuções concorrentes. Submeteu-se, então, o HARP aos testes com enlaces, para demonstrar a recuperação do sistema através da troca de primitivas. O HARP foi concluído e não foi necessário voltar à sua especificação.

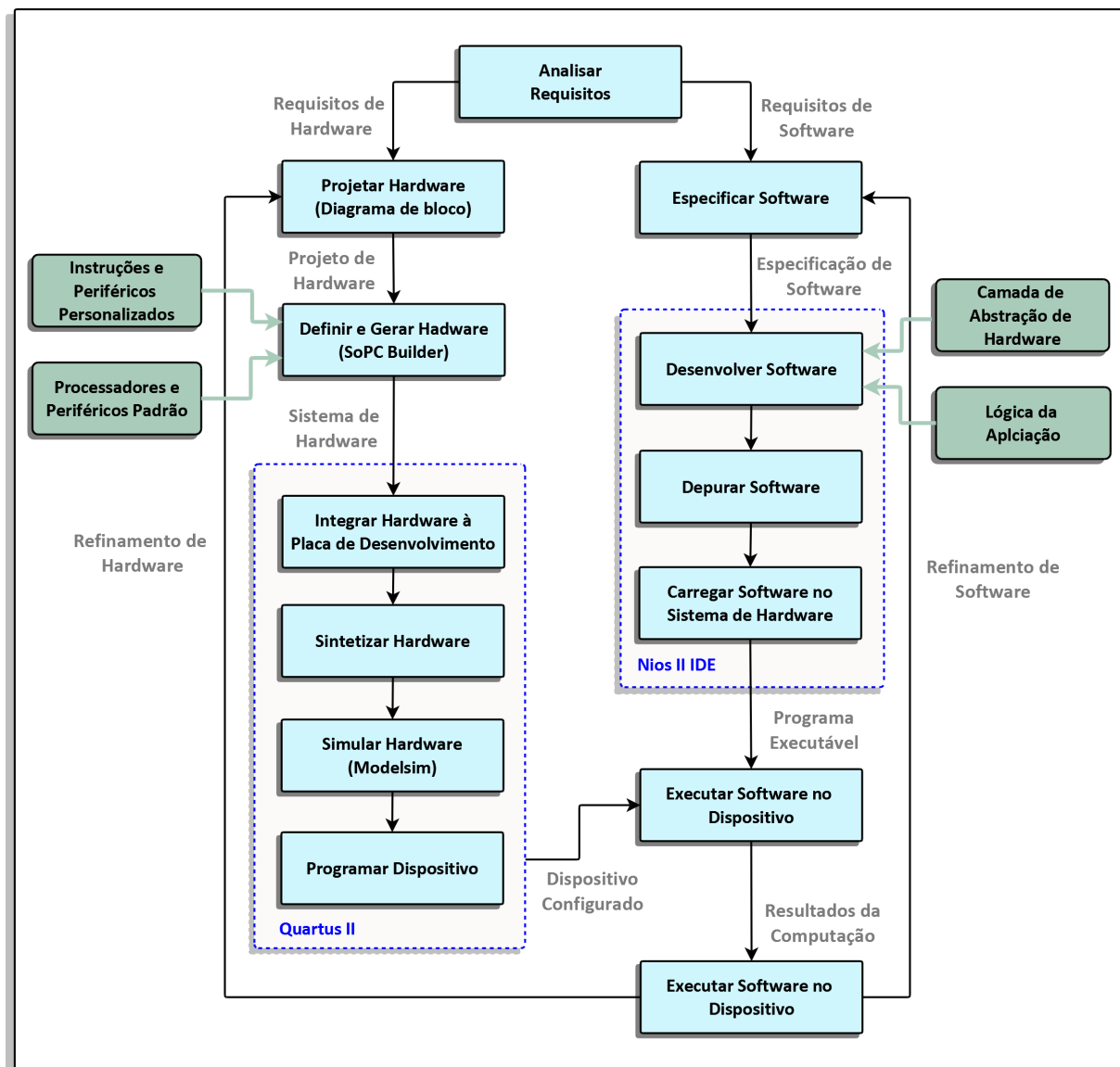


Figura 3.6: Etapas do *co-design* do sistema de validação

High Availability Router Protocol (HARP)

O HARP é um protocolo desenvolvido para cuidar do processo de obtenção de novo mestre de um grupo de alta disponibilidade. O novo mestre pode ser definido por eleição entre os escravos do grupo ou por transferência programada. Projetado para atuar em ambiente *stateless*, o HARP não trata da transferência de estados entre os equipamentos, sua função é determinar o novo líder e redirecionar automaticamente o *gateway* da rede.

A especificação de um protocolo consiste de cinco partes distintas (Holzmann, 1991). Para ser completa, cada especificação deve incluir:

1. Os serviços a serem providos pelo protocolo;
2. As suposições sobre o ambiente em que o protocolo é executado;
3. O vocabulário de mensagens usado para implementar o protocolo;
4. O formato de cada mensagem no vocabulário; e
5. As regras de procedimento que garantem a consistência da troca de mensagens.

Assim, este capítulo apresenta os cinco elementos do HARP. A Seção 4.1 traz diretrizes sobre o ambiente em que ele se enquadra. Já a seção 4.2 apresenta o vocabulário e os serviços oferecidos, relacionando cada serviço e seu grupo de mensagens envolvidas. Em seguida, a Seção 4.3 mostra o formato das mensagens e explica a função dos seus campos. Na Seção 4.4, os serviços são retomados e explicados paralelamente aos seus autômatos, para deixar claras as trocas de mensagens que ocorrem em cada um deles. Por fim, a Seção 4.5 traz a especificação formal do HARP em TLA⁺ e as conclusões inferidas dela.

4.1 Suposições sobre o ambiente

O HARP, como protocolo de alta disponibilidade, deve operar em elementos individuais de um grupo de equipamentos redundantes, conforme ilustrado na Figura 2.3. Este grupo forma o elemento virtual e é visto pelo restante da rede como um único ponto de encaminhamento de pacotes. Ele está projetado para trabalhar na camada de rede da arquitetura Internet, juntamente com o protocolo IP.

4.2 Serviços e Vocabulário

O HARP engloba um total de cinco serviços e quatorze mensagens em seu vocabulário. Naturalmente, as mensagens são trocadas para prover os serviços. Para que o leitor tenha uma visão geral do protocolo, estes dois elementos são introduzidos e relacionados pela Tabela 4.1, sendo retomados detalhadamente na Subseção 4.4. As mensagens obedecem à taxonomia *Request/Indication* e *Response/Confirmation*, i.e., uma *MSG Request* (MSG_REQ) tem o mesmo conteúdo que a *MSG Indication* (MSG_IND), diferenciando-se semanticamente no momento em que elas aparecem (IETF, 2008).

Tabela 4.1: Serviços e Vocabulário do HARP

Nome	Mensagens	Descrição
<i>Keep Alive</i> (KA)	<i>KA Request (KA_REQ)</i>	Serviço não confirmado. Atua como <i>heartbeat</i> usado pelo mestre para avisar sua disponibilidade.
<i>Grant Master</i> (GM)	<i>GM Request (GM_REQ)</i> <i>GM Response (GM_RESP)</i> <i>GM Failed Request (GMFAIL_REQ)</i> <i>GM Ready Request (GMRDY_REQ)</i>	Serviço confirmado. Utilizado pelo mestre para indicar sua intenção de transferir o seu papel a um escravo específico.
<i>Informe Node</i> (INF)	<i>INF Request (INF_REQ)</i> <i>INF Response (INF_RESP)</i>	Serviço confirmado. Utilizado para indicar que um nó está se tornando membro do grupo de HA.
<i>Remove Node</i> (REM)	<i>REM Request (REM_REQ)</i> <i>REM Response (REM_RESP)</i>	Serviço confirmado. Utilizado por um escravo para indicar ao mestre do grupo HA que está deixando a rede.
<i>Check Brain</i> (CB)	<i>CB Request (CB_REQ)</i> <i>CB Response Positive (CB_RESP+)</i> <i>CB Response Negative (CB_RESP-)</i>	Serviço confirmado. Utilizado por um escravo para se certificar de que não há mestre no grupo e evitar a situação do Cérebro Partido.

A Seção 4.4 retoma esta discussão e detalha a execução de cada um dos serviços apresentados na Tabela 4.1, bem como a função das mensagens e seu devido uso. Há

ainda duas mensagens que não foram citadas nesta tabela, pois não são necessariamente integrantes de um serviço, são elas:

- *Active Slave Request* (ACTS_REQ): enviada pelo mestre, em *broadcast*, para atualizar a sua tabela de escravos ativos;
- *Active Slave Response* (ACTS_RESP): enviada pelo escravo, ao mestre, para responder sua atividade.

Há outra mensagem, utilizada apenas para configuração do hardware, que é nomeada *Config_IP*. Ela se destina a transferir ao HARP o endereço IP atribuído a este NE. Ela contém apenas informações do tipo de mensagem e endereço IP de origem. O Apêndice A mostra os dados de preenchimento dos campos para esta mensagem e para todas as outras apresentadas nesta seção.

4.3 Formato das Mensagens

Para atender às necessidades dos serviços apontados anteriormente e por operar na camada de rede da arquitetura Internet, a mensagem do HARP tem dez campos predefinidos para sua primeira versão, que opera com IPv4. As mensagens têm 128 bits de cabeçalho e o campo de dados ainda não foi utilizado, mas pode ser nas próximas versões. Cabe salientar que outras versões do HARP são passíveis de alterações no formato de mensagens, justamente para atender a outras tecnologias. Entretanto, não há necessidade de que seja feita nova especificação de serviço ou da máquina de estados.

As mensagens HARP são essencialmente mensagens de controle, tendo por isso o cabeçalho maior que o campo de dados, em todos os casos previstos. O formato das mensagens é ilustrado pela Figura 4.1, que mostra o identificador do campo e o seu comprimento em bits.

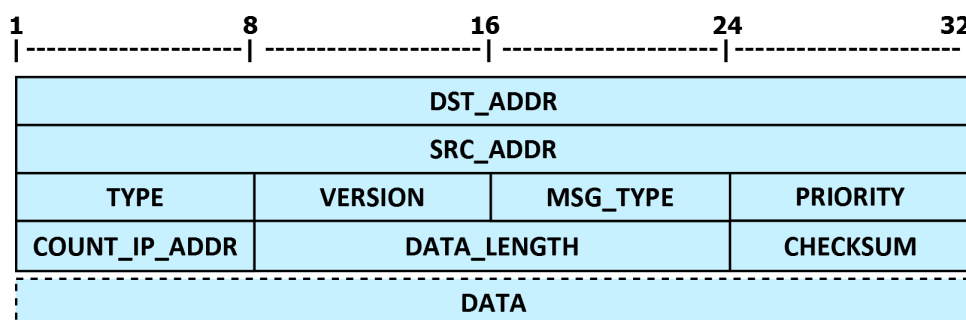


Figura 4.1: Formato da mensagem HARP

O significado de cada campo pode ser conferido segundo a lista a seguir:

1. DEST_ADDR: Endereço IP de destino. Pode ser de um elemento ou *broadcast*;

2. SRC_ADDR: Endereço IP de origem. Especifica o elemento que enviou a mensagem;
3. TYPE: Tipo do protocolo, i.e., número reservado à identificação do HARP;
4. VERSION: Versão do protocolo, para este caso será usada somente a versão um;
5. MSG_TYPE: Indica qual o tipo de mensagem reconhecida pelo vocabulário que aquela primitiva está carregando;
6. PRIORITY: Prioridade recebida pelo administrador para saber o comportamento inicial ao entrar no grupo (mestre ou escravo);
7. COUNT_IP_ADDR: Em uma mensagem do tipo *KA_Request* indica quantos escravos ativos há no grupo. Pode servir também (em outras primitivas) para dizer quantos endereços IP estão sendo enviados no campo DATA.
8. DATA_LENGTH: Indica comprimento do campo de dados;
9. CHEKSUM: Soma de verificação para detecção de erro;
10. DATA: Campo de dados para parâmetros das mensagens, quando necessário.

Os endereços 0x0000 e 0xFFFF são reservados. O primeiro indica que não há endereço sendo transmitido, enquanto o segundo destina-se ao *broadcast*. As próximas versões do HARP podem vir com mensagens de formatos diferentes para atender diferentes propostas de arquiteturas de redes.

4.4 Regras de Procedimentos

Cabe às regras de procedimentos explicar a dinâmica de troca de mensagens e o comportamento da máquina de estados finitos (MEF) (Hopcroft *et al.*, 2000) do HARP durante a execução dos procedimentos dos serviços. Cada serviço apresentado na Tabela 4.1 é aqui representado como um autômato que especifica de forma não ambígua o comportamento da instância do HARP durante sua execução. Na Subseção 4.4.6, todos os autômatos de serviços (ainda considerados parciais) são combinados em um único, tanto do ponto de vista do emissor, quanto do receptor.

São utilizados diagramas de estados para descrever os serviços. Nestes diagramas, cada transição é representada pela combinação de entrada e saída, na forma $\frac{\text{entrada}}{\text{saida}}$. Os operadores "&", "|" e "==" significam, respectivamente, as operações "e", "ou" e "igual".

4.4.1 Keep Alive (KA)

Cada instância do protocolo se inicia no estado *Idle*, conforme Figura 4.2. Baseado em sua prioridade¹, cada nó pode ir de *Idle* a *Master* (tornando-se mestre) ou a *Slave*

¹Prioridade igual a zero é prioridade de mestre. Valores maiores que zero são prioridades de escravos e podem ser recebidos do último octeto do IP. Escravos têm prioridades sequenciais e diferentes entre si.

(tornando-se escravo). Se um nó assume o papel de mestre, automaticamente ele inicia o serviço KA, enviando um KA_REQ a cada intervalo de tempo t para informar seu estado aos demais. O intervalo t é gerenciável, podendo ser, por exemplo, 1 seg ou 100 ms. A figura 4.2 mostra que um escravo só se torna membro do grupo ao receber um INF_CONF, i.e., quando ele recebe a confirmação do mestre.

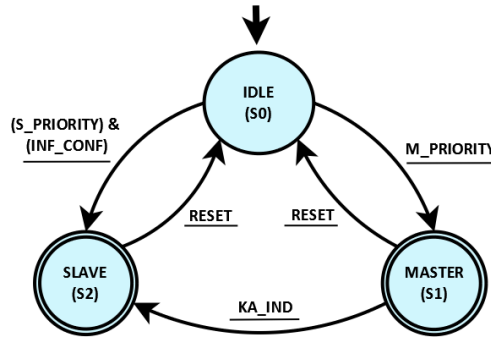


Figura 4.2: Autômato inicial do HARP

Se um mestre recebe um KA_IND significa que por alguma razão outro elemento de rede se tornou mestre e, então, o mestre receptor imediatamente vai a escravo deixando a arquitetura com apenas um mestre em atividade. Caso aconteça de os dois mestres receberem um KA_IND² ao mesmo tempo (o que ocorrerá com uma probabilidade muito menor em relação ao caso anterior), os dois irão ao estado *Slave*, encadeando o serviço *Check Brain* (Seção 4.4.5).

4.4.2 Inform Node (INF)

Quando um nó escravo entra na rede, ele envia um INF_REQ ao grupo e aguarda um INF_CONF do mestre. O mestre o inclui, então, em sua tabela de ativos. Um mestre não precisa enviar INF_REQ já que ele é o primeiro equipamento a ser inserido no grupo. Ele só envia esta primitiva se ele for ao estado de *Slave*, mas não precisa aguardar confirmação, dado que a comunicação já está estabelecida e o novo mestre já tem o seu endereço armazenado.

4.4.3 Remove Node (REM)

Um nó escravo envia ao mestre um REM_REQ para solicitar sua saída da rede e espera pela confirmação REM_CONF. Ao receber o pedido, o mestre envia um REM_RESP ao solicitante autorizando a saída. Recebendo a confirmação, o nó pode deixar a rede.

²Cabe lembrar que a mensagem KA_REQ apresentada na tabela 4.1 contém o mesmo conteúdo da mensagem KA_IND. Elas diferem entre si, semanticamente, no momento em que aparecem para cada elemento, i.e., uma MSG_REQ é enviada do emissor e ao ser recebida no receptor é referida como MSG_IND. O mesmo ocorre com MSG_RESP e MSG_CONF. Esta terminologia se aplica à todas as mensagens apresentadas na tabela 4.1.

4.4.4 Grant Master (GM)

Se um mestre deseja transferir sua função a um escravo, ele solicita o serviço GM. A escolha do escravo se dá por escolha do administrador da rede (na implementação deste trabalho adotou-se o primeiro endereço na tabela de ativos do mestre). O sinalizador *Grant Master Start* (GM_Start) é usado nas especificações para representar quando este serviço deve ser iniciado, a manutenção programada é um exemplo desta necessidade. Há quatro tipos de mensagens especificamente para este serviço:

- *Grant Master Request* (GM_REQ): O mestre solicita a um escravo predefinido que se torne mestre.
- *Grant Master Response* (GM_RESP): O escravo que recebeu a solicitação para se tornar mestre usa esta primitiva para responder positivamente ao mestre que está disposto a mudar sua função. Não há confirmação negativa, o que significa que um escravo sempre se disponibilizará a assumir como mestre quando solicitado.
- *Grant Master Failed* (GMFAIL_REQ): O mestre diz que o processo GM falhou e a primitiva GM_CONF não chegou a ele, então todos os envolvidos voltam aos seus estados originais.
- *Grant Master Ready* (GMRDY_REQ): Esta primitiva garante que a condição de Cérebro Partido não ocorrerá no caso de falha no canal de transmissão durante o serviço GM. O mestre a envia para dizer que entendeu que o escravo está pronto para tornar-se um mestre, e agora pode passar a escravo. Ao recebê-la, o escravo deve mudar sua função para mestre.

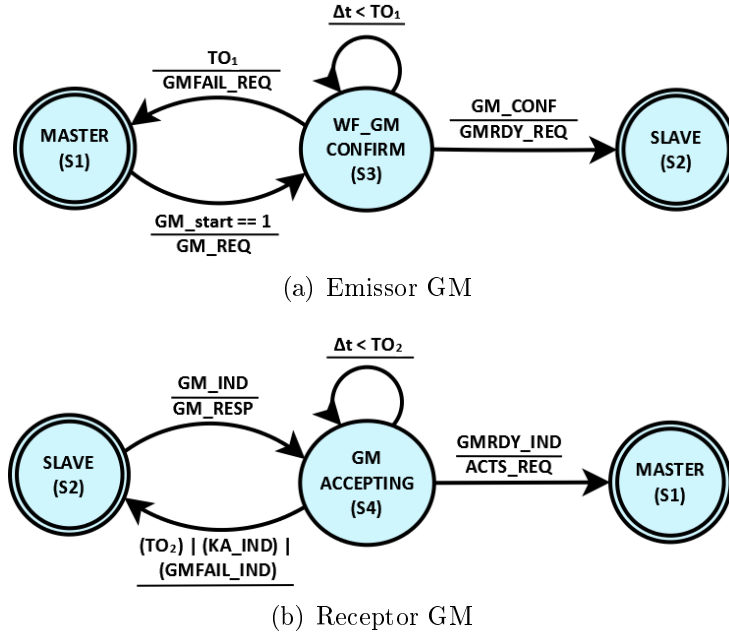
A Figura 4.3 ilustra, respectivamente, o comportamento do emissor e do receptor em um processo GM.

O mestre envia um GM_REQ a um escravo predeterminado e vai para o estado *Wait For Grant Master Confirm* (S3). Se ele recebe um GM_CONF, ele vai para o estado *Slave* e envia um GMRDY_REQ informando a conclusão do processo. Caso contrário, se ocorre o TO_1^3 (intervalo t), ele envia o um GMFAIL_REQ informando erro no processo e volta ao estado *Master*.

Do lado do receptor, quando o escravo recebe o GM_IND, ele envia um GM_RESP e vai ao estado *Grant Master Accepting* (S4), esperando um intervalo para garantir que não há outro elemento de rede enviando mensagens *Keep Alive*. Se um GMRDY_IND é recebido, acontece a transição ao estado *Master*, mas se ocorre o TO_2 , um KA_IND ou GMFAIL_IND, ele retorna ao estado *Slave*, evitando a condição de Cérebro Partido.

Dentro do contexto do GM, algumas situações são ilustradas para demonstrar a capacidade de recuperação do protocolo no caso de perdas de mensagens:

³ TO_n representa *timeout* e é utilizado nos estados com tempo de permanência máximo preestabelecido.

Figura 4.3: Autômatos do serviço *Grant Master*

- Se a primitiva GM_REQ for perdida, então o escravo receptor não recebe o GM_IND e tampouco envia o GM_RESP, logo o mestre emissor deve voltar ao seu estado de origem antes que os outros escravos percebam a sua falta. Por isso, o TO_1 é menor que o TO_3 (*timeout* de não recebimento do KA_IND) para garantir que não haverá condição de Acéfalo na rede.
- Já no caso de perda do GM_RESP, não haverá o envio da primitiva GMRDY_REQ, não permitindo que o receptor vá ao estado *Master*. O receptor, que estava no estado *Grant Master Accepting* (S4), conseqüentemente voltará ao estado *Slave* pela ocorrência do TO_2 ⁴, enquanto esperava o GMRDY_IND. Ele também volta ao estado *Slave* ao receber o GMFAIL_IND ou um KA_IND.
- Caso haja perda da primitiva GMFAIL_REQ não haverá problema, pois o TO_2 e o KA_IND contornam esta perda no estado *Grant Master Accepting* (S4).
- Caso a primitiva GMRDY_REQ se perca no caminho, há a situação em que o mestre vai a *Slave* e o escravo continua em *Slave*, gerando a condição de Acéfalo. Este é o pior caso quando se trata de perda de mensagens no processo GM, mas caso isso ocorra, o serviço *Check Brain* é iniciado automaticamente por algum escravo e um novo mestre será eleito.

4.4.5 Check Brain (CB)

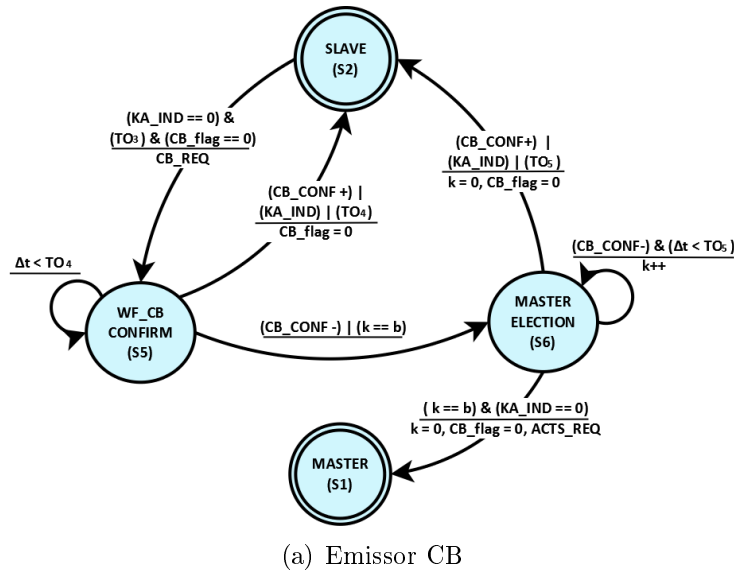
Se o escravo detecta a falta do mestre, o processo para eleger um novo é iniciado. Este é um ponto de destaque na especificação, pois garante a não ocorrência da condição de

⁴O TO_2 e o TO_1 são menores que o TO_3 e ambos valem t .

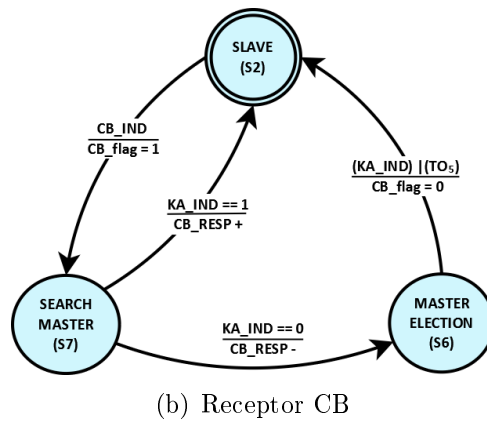
Cérebro Partido e resolve diretamente a condição de Acéfalo. O sinalizador (CB_Flag) é usado para inibir que o processo CB seja iniciado por algum escravo quando já tiver sido iniciado e não acabado por outro. Este serviço é usado entre os escravos considerando o seguinte:

- *Check Brain Request* (CB_REQ): Um escravo pergunta, em *broadcast*, se os outros escravos estão se comunicando com o mestre.
- *Check Brain Response Positive* (CB_RESP+): Um escravo envia em *unicast* ou uma confirmação positiva para dizer que há mestre na rede.
- *Check Brain Response Negative* (CB_RESP-): Um escravo envia em *unicast* uma confirmação negativa dizendo que também não está em comunicação com o mestre.

A Figura 4.4 descreve o comportamento de uma instância durante o serviço *Check Brain*, tanto para o emissor quanto para o receptor.



(a) Emissor CB



(b) Receptor CB

Figura 4.4: Autômatos do serviço *Check Brain*

O tempo necessário para que se perceba uma falha do mestre é o TO_3 e baseia-se no não recebimento de KA_IND. $TO_3 = (2 + a) \star t$, onde a é uma constante baseada na

prioridade. Isso diminui a probabilidade de a falha do mestre ser percebida no mesmo momento por vários escravos. O TO_3 é maior que todos os outros *timeouts* da MEF, garantindo que o *Check Brain* não seja iniciado durante a execução de um outro serviço.

Um nó escravo percebe que há um intervalo de tempo TO_3 sem o recebimento do KA_IND e percebe ainda que o sinalizador CB_flag está resetado, certificando-se de que o processo CB ainda não foi iniciado. O nó envia então um CB_REQ em *broadcast* para os outros escravos e vai ao estado *Wait For Check Brain Confirm* (S5), para aguardar pela confirmação de outros escravos sobre a não existência do mestre na rede.

Caso o escravo emissor receba uma confirmação positiva CB_CONF(+) ele volta ao estado de *Slave*, resetando o contador do TO_3 , que é naturalmente resetado a cada recebimento do KA_IND. O retorno ao estado de *Slave* pode ocorrer também se houver o recebimento de um KA_IND (indicando que há mestre) ou caso nenhuma mensagem chegue e ocorra o TO_4 ⁵.

Em chegando um CB_CONF(-), entende-se que outro elemento escravo também não está recebendo mensagens do mestre. O emissor vai então ao estado de *Master Election* (S6) para definitivamente certificar-se de que não há mestre na rede. Ele espera receber um total b de mensagens CB_CONF(-) proporcional ao número total de escravos no grupo. O valor b é dado pelo teto de $b = \frac{totaldeescravos}{2}$.

Se ele recebe as confirmações negativas esperadas, ele vai ao estado de *Master* e envia KA_REQ aos outros escravos da rede. Ele deve então zerar a CB_flag e o contador k de CB_CONF(-), que espera atingir o valor b . Ao receber um KA_IND os nós receptores zeram todos os seus contadores e o CB_flag, continuando escravos e reconhecendo o novo mestre.

Do lado dos receptores, quando um CB_IND é recebido, ele armazena o endereço de origem e seta sua CB_flag, indicando que o processo CB foi iniciado e está em andamento. Então, ele vai para o estado *Search Master* para conferir a quanto tempo não recebe o KA_IND. Caso tenha recebido KA_IND há no máximo $2t$ ele envia resposta CB_RESP(+) e volta para o estado de *Slave*, pois para ele tudo está operando dentro da normalidade e certamente houve uma falha pontual do emissor.

Entretanto, se o receptor perceber no estado *Search Master* que não houve recebimento de KA_IND no último intervalo $2t$, ele envia um CB_RESP(-) atestando a falta do mestre e vai ao estado de *Master Election* para atrasar sua volta e dar tempo para o emissor concluir o processo. Ele só deixa o estado *Master Election* para ir ao *Slave*, o que ocorre pelo recebimento de um KA_IND ou pelo TO_5 , quando ele reseta seu CB_flag. O TO_5 é igual a t para emissor e $2t$ para receptor.

⁵O TO_4 tem o valor de t . Quando ele ocorre o escravo volta ao seu estado inicial, resetando o TO_3 e o sinalizador CB_flag, daí recomeça a esperar o KA_IND novamente e, depois disso, pode repetir todo o processo CB, caso necessário. Este procedimento garante que o escravo permanece escravo se, por exemplo, houver falha somente na interface dele, seja ela permanente ou temporária.

Cabe destacar que apesar de o intervalo necessário para iniciar um CB seja TO_3 , o intervalo $2t$ já é suficiente para gerar respostas CB_RESP. Isso porque o escravo de menor TO_3 poderia perceber a falha e os outros ainda não terem atingido seus *timeouts* TO_3 (por causa da constante a) e deixariam o emissor sem resposta atualizada.

O uso de um TO_3 variável minimiza o risco de mais de um escravo iniciar o CB ao mesmo tempo. Ainda que isso ocorra, uma das mensagens atingirá o canal antes da outra, baseado no princípio de compartilhamento de canal (CSMA/CD). É a esta mensagem que os escravos responderão, pois eles ignorarão mensagens subsequentes por já terem setado o CB_flag. Este processo faz com que um dos escravos emissores volte ao estado de *Slave* quando ocorrer o TO_4 no estado *Wait For Check Brain Confirm*.

Para evitar mais uma vez a condição de Cérebro Partido, deve-se prever a situação em que o mestre fica indisponível um tempo suficientemente grande para a eleição de um novo mestre e depois disso volta a operar normalmente, deixando dois mestres no grupo. Para isso, ao receber um KA_IND os mestres envolvidos param de enviar KA_REQ e vão ao estado de *Slave*. Isso evita o Cérebro Partido e, no máximo, gera a condição de Acéfalo, que é solucionável com o processo *Check Brain*.

Ao se concluir um processo de *Grant Master* ou *Check Brain*, o novo mestre envia um ACTS_REQ ao grupo para atualizar a sua tabela de ativos e, conseqüentemente, atualizar o grupo de quanto escravos estão inseridos. A tabela de ativos é sempre atualizada com o recebimento de um ACTS_CONF ou de um INF_IND.

Todos estes procedimentos descritos são executados por qualquer instância do protocolo e a qualquer tempo. Portanto, as regras são combinados em uma única MEF capaz de gerenciar a comunicação.

4.4.6 Autômato Final

A implementação de um protocolo requer que a instância em execução esteja preparada para todas as possíveis sequências comportamentais, seja ela receptora, emissora, mestre ou escrava. Desta forma, é possível gerar um único diagrama de estados que descreva o comportamento de uma instância do HARP, a partir da combinação dos autômatos anteriores. A Figura 4.5 relaciona todos os eventos e ações decorrentes desta união.

A descrição dos estados que compõem a MEF da Figura 4.5 já foi vista, bem como os eventos reconhecíveis para a troca entre os estados. O que caracteriza uma instância como mestre ou escrava é a sua prioridade ao entrar no grupo e a conclusão de um serviço de eleição ou transferência de papel. Se ela será emissora ou receptora em um serviço, depende da sua iniciativa para desencadear a troca de mensagens.

Desde S1 a S7, todos os estados voltam ao estado S0 se resetados, por uma questão de organização visual esta transição não é apresentada. Os estados da parte superior da figura (S4, S5, S6, e S7) são considerados estados de escravo, bem como o estado da parte

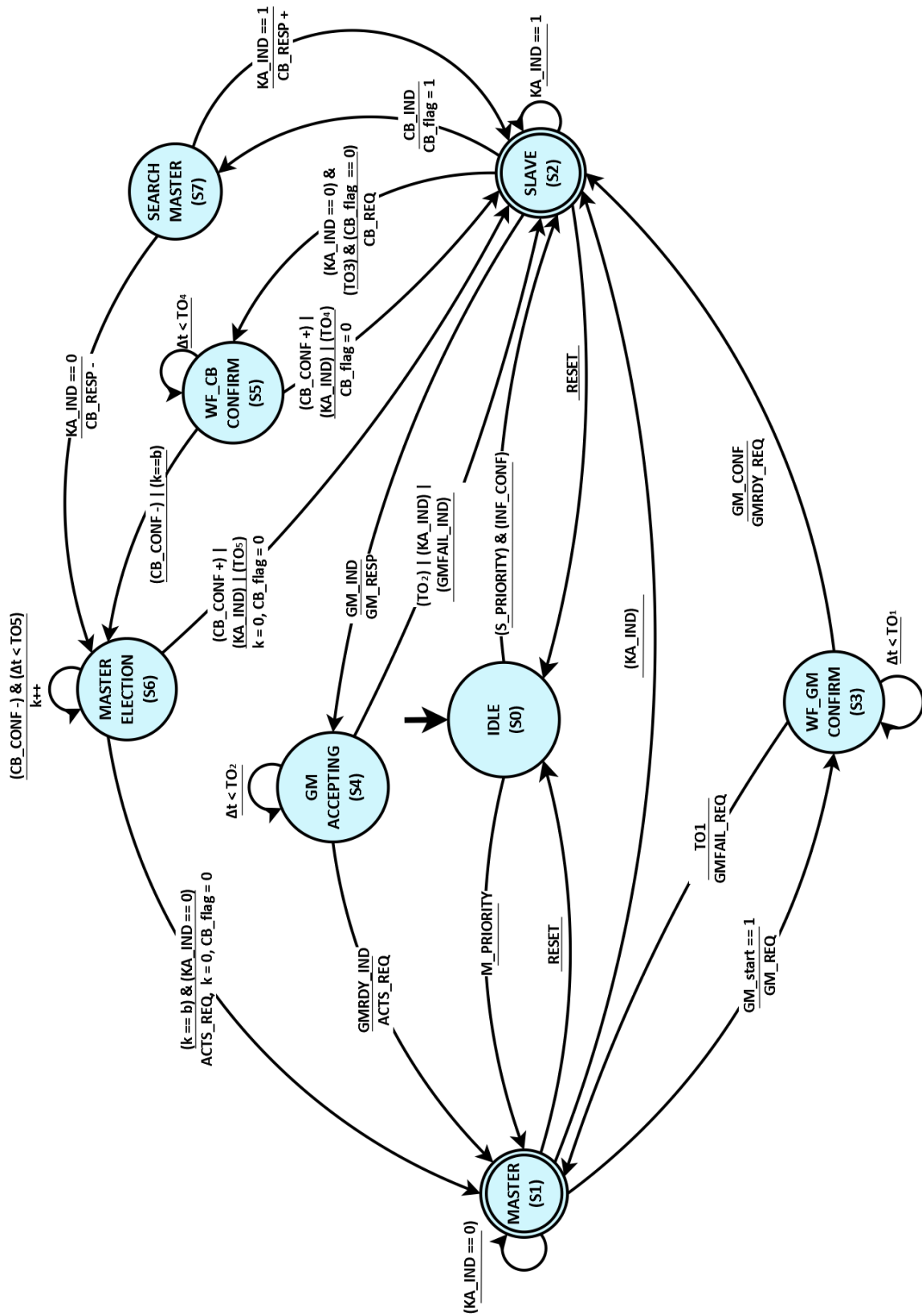


Figura 4.5: HARP - Autômato completo

inferior (S3) é um estado de mestre.

O autômato do HARP é simples e modular, uma vez que é construído com um número pequeno de partes bem formadas. Autômatos complexos necessariamente requerem simulação computacional para que se tenha certeza das suas boas propriedades. Depois de analisar a MEF do HARP e apontar suas boas propriedades, uma descrição em TLA^+ é apresentada na próxima seção.

4.5 Especificação formal em TLA^+

Uma vez conhecidos os elementos do protocolo, a atenção agora volta-se para uma outra especificação do HARP, desta vez mais abstrata, generalista e formal. Neste contexto, esta seção começa por dar uma visão geral de como a especificação em TLA^+ (*Spec*) foi construída e segue trazendo trechos que ajudam a explicar como ela funciona.

Como dito na seção 2.1.3, a linguagem TLA^+ é baseada na Lógica Temporal de Ações e na Teoria de Conjuntos. Assim, as *Specs* são constituídas por expressões matemáticas ordinárias que contêm variáveis e constantes. Se estas expressões não englobam operadores temporais, elas são chamadas funções de estados, caso contrário, são classificadas como funções de transição.

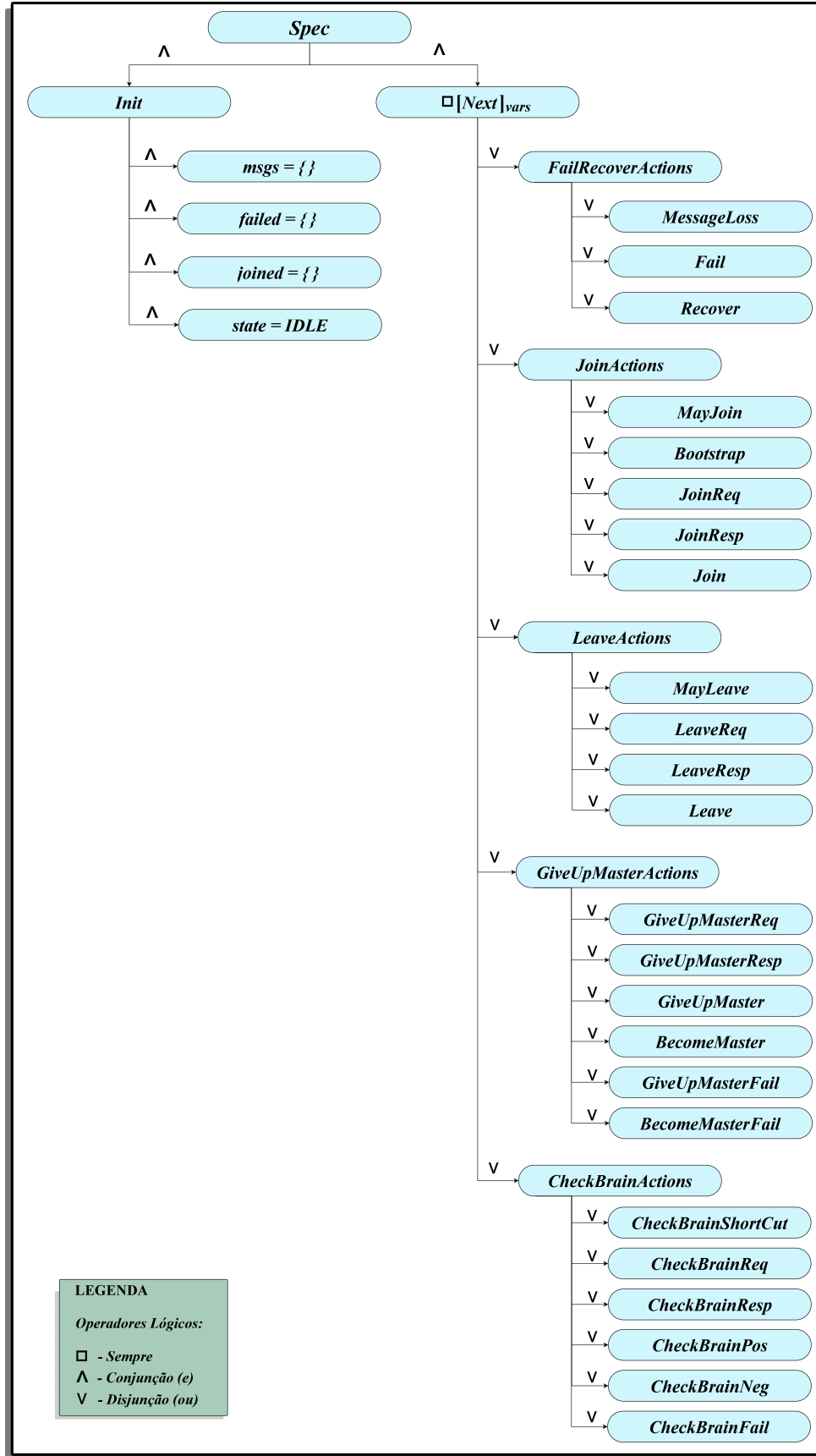
Matematicamente, um sistema é descrito por equações que determinam como seus estados evoluem com o tempo. Formalmente, define-se um comportamento como sendo uma sequência de estados, onde um estado representa determinada atribuição de valores para variáveis. Logo, a *Spec* descreve o conjunto de possíveis comportamentos que representam uma correta execução do mesmo (Lamport, 2003).

Uma visão geral da *Spec* do HARP aparece na Figura 4.6. Como se pode perceber no início do fluxograma, a *Spec* é satisfeita por duas condições principais e cada uma delas leva a um conjunto de ações que dizem respeito à possíveis comportamentos de uma instância do protocolo.

Como se pode perceber, a *Spec* é uma disjunção de duas expressões: *Init* e $\Box[Next]_{vars}$. *Init* descreve o estado inicial do sistema. $\Box[Next]_{vars}$ afirma que sempre é possível executar uma transição de estados descrita em *Next*. Todos os estados alcançáveis por HARP são baseados nas quatro variáveis *state*, *joined*, *msgs* e *failed* declaradas conforme o trecho *Spec* HARP - Fragmento 1.

A variável *state* refere-se aos estados do autômato da Figura 4.5; *joined* é o conjunto de elementos que podem ser vistos ativos no grupo; *msgs* refere-se ao conjunto de mensagens definidas no vocabulário (Tabela 4.1) e sendo transmitidas e, por último, *failed* é o conjunto de processos falhos. A constante *PID* define o conjunto de processos (e suas IDs) que se pretende validar concorrentemente. Ela pode assumir qualquer subconjunto dos números naturais $\mathbb{N}$.

Em TLA^+ as variáveis não são tipadas como em linguagens de programação, elas

Figura 4.6: Organização modular da *Spec*

MODULE *HARP*

EXTENDS *Naturals, Integers, TLC, FiniteSets*

Processes

VARIABLES *state*, Am I the master, a slave, or idle? Am I receiving mastership?
 Electing a leader?
 Am I part of the system?

joined

Communication channels

VARIABLE *msgs*

Global state

VARIABLE *failed*

vars $\triangleq$ $\langle state, joined, msgs, failed \rangle$

CONSTANT *PID*

Spec HARP - Fragmento 1: Início do módulo e declaração de variáveis

podem assumir os valores que são convenientes de acordo com cada especificação. Assim, o trecho *Spec* HARP - Fragmento 2 mostra como as variáveis da *Spec* HARP devem ser interpretadas.

PStates $\triangleq$ { "IDLE", "MASTER", "SLAVE",
 "MASTER_GM_CONFIRM", "SLAVE_GM_ACCEPT",
 "SLAVE_EL_CONFIRM", "SLAVE_EL_ELECTION" }

Msgs $\triangleq$ [*type* : { "GM_REQ" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "GM_RESP" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "GM_FAIL_REQ" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "GM_RDY_REQ" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "INF_REQ" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "INF_RESP" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "REM_REQ" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "REM_RESP" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "CB_REQ" }, *from* : *PID*] $\cup$
 [*type* : { "CB_RESP+" }, *from* : *PID*, *to* : *PID*] $\cup$
 [*type* : { "CB_RESP-" }, *from* : *PID*, *to* : *PID*]

TypeInvariant $\triangleq$ $\wedge msgs \in \text{SUBSET } Msgs$
 $\wedge state \in [PID \rightarrow PStates]$
 $\wedge failed \in \text{SUBSET } PID$
 $\wedge joined \in \text{SUBSET } PID$

Spec HARP - Fragmento 2: Invariantes para variáveis na *Spec*

Os conjuntos *PStates* e *Msgs* são essencialmente a descrição da especificação dos elementos do HARP para a *Spec*. A asserção *TypeInvariant* afirma que em cada comportamento que satisfaz a *Spec*, as variáveis *msgs*, *state*, *failed* e *joined* devem ser subconjuntos, respectivamente, de *Msgs*, *PStates*, *PID* e *PID*.

Conforme a Figura 4.6, a *Spec* se desdobra na conjunção das expressões *Init* e $\Box[Next]_{vars}$. Estas expressões definem comportamentos que garantem as propriedades de *Safety* do pro-

protocolo HARP (Lamport, 2003). O trecho *Spec* HARP - Fragmento 3 mostra a fórmula principal *Spec*⁶, bem como as ações que a definem.

$$\begin{aligned} Init &\triangleq \wedge msgs = \{\} \\ &\quad \wedge failed = \{\} \\ &\quad \wedge state = [p \in PID \mapsto "IDLE"] \\ &\quad \wedge joined = \{\} \end{aligned}$$

$$\begin{aligned} Next &\triangleq \vee FailRecoverActions \\ &\quad \vee JoinActions \\ &\quad \vee LeaveActions \\ &\quad \vee GiveUpMasterActions \\ &\quad \vee CheckBrainActions \end{aligned}$$

$$\begin{aligned} Spec &\triangleq \wedge Init \\ &\quad \wedge \Box [Next]_{vars} \end{aligned}$$

Spec HARP - Fragmento 3: Fórmula principal e suas relações

O predicado inicial da fórmula *Spec* especifica os possíveis valores iniciais para as variáveis. Entende-se pelo excerto *Spec* HARP - Fragmento 3 que o estado inicial do HARP deve satisfazer à asserção *Init*. Em outras palavras, para a *Spec*, todos os processos iniciam-se no estado *IDLE*, nenhum falho e sem mensagens nos canais.

Enquanto isso, o termo $\Box [Next]_{vars}$ significa a relação de próximo estado e define como os valores das variáveis podem mudar em um passo⁷. Em particular, $\Box [Next]_{vars}$ é a asserção que diz que *Next* é verdadeiro para cada passo no comportamento. Assim, o predicado $\wedge Init \wedge \Box [Next]_{vars}$ é verdade se e somente se o estado inicial satisfaz *Init* e cada passo satisfaz *Next*.

Tanto a Figura 4.6 quanto o excerto *Spec* HARP - Fragmento 3 apontam que o termo $\wedge \Box [Next]_{vars}$ do predicado da fórmula *Spec* é formado pela disjunção de uma série de ações relativas aos processos. Estas ações são a representação de (i) como uma instância do HARP pode se comportar quando da execução de algum serviço ou de (ii) alguma intercorrência inesperada habilitada por outras partes do sistema. De uma forma sucinta, elas são apresentadas na lista a seguir. Os detalhes da sua implementação e asserções lógicas que as compõem podem ser consultados no Apêndice B, que traz a *Spec* por completo.

1. *FailRecoverActions*: Manipulam ações de falha e recuperação de processos;

- 1.1. *MessageLoss*: Mensagem se perde, i.e., ela é interpretada pelo *Model Checker* como tendo saído de um processo *p* e não chegado nenhum outro do conjunto *PID*.

⁶A fórmula *Spec* recebe o mesmo nome da especificação *Spec* em virtude de ser a fórmula que relaciona todas as ações do seu comportamento.

⁷Passo é definido como uma sequência de transição entre dois estados.

- 1.2. *Fail*: Processo falha e passa a fazer parte do conjunto de processos inativos.
- 1.3. *Recover*: Processo retorna a *IDLE* no próximo passo.
2. *JoinActions*: Relativas à tentativa de ingresso no grupo por um processo em *PID*.
 - 2.1. *Bootstrap*: Um processo que forçadamente vai a *MASTER*.
 - 2.2. *MayJoin*: Processo não falho e que ainda não faz parte do *joined*.
 - 2.3. *JoinReq*: Processo envia uma mensagem solicitando sua entrada.
 - 2.4. *JoinResp*: Processo *MASTER* responde a solicitação.
 - 2.5. *Join*: Processo assume estado *SLAVE* e é incluído no *joined*.
3. *LeaveActions*: Ações para a saída de um processo *SLAVE*.
 - 3.1. *MayLeave*: Processo não falho e visto no *joined* que está apto a pedir para deixar o grupo.
 - 3.2. *LeaveReq*: Processo envia mensagem solicitando saída, mas não há mudança de estado.
 - 3.3. *LeaveResp*: Processo *MASTER* envia resposta para que ele deixe o grupo.
 - 3.4. *Leave*: O processo está apto a sair. Recebe uma mensagem de autorização e, no próximo estado, seu estado é alterado para *IDLE*. A variável *joined* é alterada.
4. *GiveUpActions*: Cuidam da transferência espontânea da função de mestre entre processos.
 - 4.1. *GiveUpMasterReq*: Se existe um processo cujo estado é *MASTER* e outro cujo estado é *SLAVE*, uma mensagem é enviada para um escravo aleatório solicitando a transferência de função e o emissor vai a *MASTER_GM_CONFIRM*.
 - 4.2. *GiveUpMasterResp*: O processo receptor da solicitação, se ativo e participante do *joined*, envia resposta de aceitação. e vai a *SLAVE_GM_ACCEPT*.
 - 4.3. *GiveUpMaster*: Processo mestre cujo estado atual é *MASTER_GM_CONFIRM* recebe resposta e vai a *SLAVE* no próximo passo.
 - 4.4. *BecomeMaster*: Processo escravo cujo estado atual é *SLAVE_GM_ACCEPT* recebe confirmação e vai a *MASTER*.
 - 4.5. *GiveUpMasterFail*: Processo mestre solicitante decide pela não conclusão do serviço e envia mensagem de falha, voltando de *MASTER_GM_CONFIRM* a *MASTER*.
 - 4.6. *BecomeMasterFail*: Processo escravo decide pela não conclusão do processo e volta de *SLAVE_GM_ACCEPT* a *SLAVE*.

5. *CheckBrainActions*: Referem-se ao processo de consenso e eleição de um novo mestre.
- 5.1. *CheckBrainShortCut*: Se há apenas um processo ativo visto pelo *joined*, o escravo vai direto a *MASTER*.
- 5.2. *CheckBrainReq*: Processo escravo decide pelo início da eleição e envia mensagem solicitando confirmação da ausência de mestre. Vai então a *SLAVE_EL_CONFIRM*.
- 5.3. *CheckBrainResp*: Processo que ainda não recebeu solicitação de *CB* a recebe e envia resposta positiva ou negativa.
- 5.4. *CheckBrainPos*: Processo solicitante recebe resposta positiva e volta de *SLAVE_EL_CONFIRM* a *SLAVE*.
- 5.5. *CheckBrainNeg*: Processo solicitante recebe respostas negativas suficientes para concluir o processo e vai de *SLAVE_EL_CONFIRM* a *SLAVE*.
- 5.6. *CheckBrainFail*: Processo participante da eleição desiste do processo e retorna ao estado *SLAVE*.

As ações listadas na *Spec* estão diretamente relacionadas aos comportamentos expostos na Seção 4.4 (Regras de Procedimentos). No entanto, estas ações se relacionam de forma mais geral e abstrata que as daquela seção. Isso é esperado, uma vez que a *Spec* é uma descrição mais geral do sistema, de forma que as afirmações validadas com esta descrição em TLA⁺ se aplicam à especificação dos elementos do HARP em cenários mais restritos.

4.5.1 Resultados da verificação

Ao submeter a *Spec* à verificação pelo *TLC Model Checker*, a principal asserção que deve ser verificada é a *TypeInvariant* mostrada no excerto *Spec HARP* - Fragmento 2, pois ela garante que todos os comportamentos que satisfazem à especificação do *HARP* atribuem às variáveis valores previstos. É importante lembrar que a ferramenta TCL é utilizada como um verificador de especificações e não no intuito de prová-las. Isso porque os testes de verificação das *Specs* são limitados por determinadas restrições que inibem uma explosão de estados, o que torna, então, o conjunto de possibilidades comportamentais finito.

Para a *Spec HARP*, considerou-se como restrição que as mensagens no *buffer* do canal não ultrapassem o limite de 30, o que é suficiente para que cada processo envie 3 mensagens a cada outro processo. Além disso, considerou-se ainda que a variável *failed* é sempre um conjunto vazio, i.e., os processos não falham. Por último, o número de processos ativos e vistos pela *Spec* é sempre igual ao número em *joined*, tornando perfeita a visão de quantos processos estão ativos e com estado válido. Desta forma, criou-se um ambiente que tende ao ideal para testar a execução do protocolo.

Sabe-se que os ambientes livres de falhas não existem na realidade (Camargos, 2009), entretanto, em termos práticos não há como provar todas as condições possíveis, o que leva então, a concluir que a prova de que o protocolo é bem formado para ambientes ideais é o parâmetro suficiente para iniciar qualquer análise. A consistência do protocolo se mantém em ambientes reais, proporcionalmente à qualidade da infraestrutura onde foi implementado.

Durante a verificação da *Spec*, o conjunto $\{1,2,3\}$ foi atribuído à constante *PID*, o que significa que os testes foram realizados para três processos. Durante sua verificação, as ações não consideram os aspectos de implementação. Elas também não consideram *timeouts* e enviam mensagens sem nenhum histórico, ou seja, apesar de os processos responderem a mensagens recebidas com uma outra mensagem, elas podem tanto ser enviadas em qualquer passo de um processo a outro como podem também ser ignoradas.

O excerto em *Spec HARP* - Fragmento 4 mostra uma asserção definitiva para a verificação das propriedades do HARP. Um *THEOREM* em TLA^+ é uma formula temporal satisfeita por cada estado. Assim, a asserção neste trecho diz que a fórmula principal *Spec* da especificação implica que para cada comportamento descrito nas ações (Figura 4.6) as variáveis assumem os valores definidos pelo fórmula *TypeInvariant* (do *Spec HARP* - Fragmento 2) em todos os estados.

THEOREM *Spec* $\implies \Box TypeInvariant$ Types are always correct.

Spec HARP - Fragmento 4: Teorema de invariância implicado pela *Spec*

Como resultado da verificação, o *Model Checker* ratificou que o teorema *Spec* $\implies \Box TypeInvariant$ está correto e é uma implicação verificável da fórmula principal *Spec*. Logo, se uma implementação obedece à esta especificação, as terminações dos seus processos serão definidas sempre por valores conhecidos e pertencentes aos conjuntos declarados nas variáveis, uma vez que há garantias de *Safety* (Seção 2.1.3) desta invariante para a *Spec*. Além disso, o verificador concluiu também que não há *deadlocks* na *Spec*, ou seja, todos os caminhos levam um processo a um estado válido.

Um ambiente de implementação real requer que requisitos mais complexos sejam considerados. A exemplo disso, cita-se (i) o quão eficiente é o mecanismo de detecção de falha de um processo (caso haja um) e (ii) a probabilidade de mensagens sofrerem alterações. Os resultados da convergência do protocolo serão diretamente proporcionais à confiabilidade destes recursos.

Para (i), Camargos (2009) define um detector de falha como sendo um oráculo distribuído, com módulos acoplados aos processos (ou objetos) do sistema para determinar o estado funcional dos outros processos e eles ajudam a contornar o problema do consenso em ambiente assíncronos. Discussões a este respeito podem ser consultadas em Bertier

et al. (2002), Chandra *et al.* (1996) e Chen *et al.* (2002). Quanto à (ii), Holzmann (1991) demonstra como calcular a probabilidade de mensagens serem alteradas em função da taxa de erro dos ambientes. Assim, abre-se um leque para um trabalho futuro neste quesito, que é sugerido no Capítulo 7.

4.5.2 Mapeamento entre especificações

Uma vez que foram mostradas duas especificações para o protocolo HARP (especificação dos elementos e *Spec*), é conveniente estabelecer a relação entre elas e deixar claro como elas são compatíveis entre si, mesmo em diferentes níveis de abstração.

Ao escrever uma especificação, deve-se primeiro escolher a abstração para criá-la. A especificação dos cinco elementos de um protocolo requer a criação dos serviços providos, mensagens trocadas, formato de mensagem, ambiente de execução e regras de procedimento. Em TLA^+ , especificação significa escolher as variáveis que representam os estados do sistema e a granularidade de passos que mudam os valores destas variáveis.

Desta forma, é intuitivo apontar as variáveis declaradas na *Spec* (*Spec HARP* - Fragmento 1) como sendo as representações das grandezas apontadas nas Seções de 4.1 a 4.4. Estas variáveis são a parte da *Spec* visíveis entre os processos, o que as faz serem a descrição do comportamento do HARP para o mundo externo. Este comportamento é refletido diretamente nas Regras de Procedimento apresentadas na Seção 4.4.

De forma específica, o conjunto *Msgs* compreende o conjunto de mensagens da Tabela 4.1. A variável *state*, por sua vez, abstrai a MEF que o autômato da Figura 4.5 representa. Já a variável *joined* diz respeito ao número de nós que passaram do estado *IDLE* a *MASTER* ou *SLAVE*, conforme o autômato parcial da figura 4.2, o que significa a visão que o mestre tem do grupo de ativos do sistema (parâmetro este que é enviado em todo KA_IND aos escravos). Por último, a variável *failed* representa os processos com problemas de interface ou de outra natureza e, em decorrência disso, influenciam diretamente na convergência da eleição.

Conclusivamente, a especificação dos elementos se atém a criar o protocolo e permitir algum tipo de análise da MEF, enquanto sua descrição formal em TLA^+ atenta-se para a verificação das boas propriedades. Uma implementação derivada de qualquer umas destas especificações resultará em comportamentos que satisfazem às exigências de um protocolo para alta disponibilidade do tipo FHRP, levando-se em consideração as características do ambiente de implementação.

4.6 Conclusão do Capítulo

Este capítulo apresentou os elementos do protocolo HARP e seus comportamentos previstos. Para um protocolo, há fatores que o classificam como bem formado e consistente

(Holzmann, 1991). Pode-se concluir que o HARP atende estes fatores ao analisar a MEF da Figura 4.5 e devido às propriedades e teorema demonstrados pela *Spec*. Em virtude disso, a lista a seguir enumera estes fatores.

1. O HARP é um protocolo bem formado:

- Limitado: Ele não sobrecarrega a capacidade do sistema, tal como capacidade limitada de mensagens;
- Autoestabilizável: Se um erro arbitrário muda o estado do protocolo, ele sempre retorna a um estado desejável com um número finito de transições;

2. O HARP é um protocolo consistente:

- Livre de *Deadlocks*: Não há estados sem sucessores;
- Livre de *Livelocks*: Não há sequências de execução que podem ser repetidas indefinidamente sem nenhum progresso efetivo;
- Livre de terminações inadequadas: A execução do protocolo retorna aos estados finais e satisfaz às condições para terminação adequada, ainda que os erros atrasem a convergência à medida que tendem ao pior caso.

Conforme exposto no início deste trabalho, além da especificação do HARP, ele foi implementado para fins de validação e prova de conceito. Visto que os elementos do HARP foram detalhadamente expostos, o próximo capítulo trata das questões arquiteturais para sua implementação.

Arquitetura do HARP

O processamento das mensagens nos equipamentos de rede é feito observando-se a ordem de encapsulamento dos dados em cada camada. Se a extração ou inclusão de cabeçalhos e dados é realizada por software, as instruções devem obedecer uma execução sequencial de passos para este processamento, acompanhando a sequência de camadas da rede. Quando o processamento destas mensagens multicampo é feito por hardware, a penalidade causada pelo processamento sequencial de dados pode ser dirimida. Desta forma, uma arquitetura que explora o paralelismo espacial do FPGA se aplica plenamente às especificações do HARP.

Ainda que as exigências iniciais da implementação do HARP não sejam rigorosas quanto ao desempenho, é conveniente que se possa explorar este recurso no FPGA a fim de conseguir que o HARP opere compativelmente às tecnologias de rede disponíveis e visando obter um hardware que possa ser utilizado em um equipamento de rede. Ainda assim, o uso do FPGA nesta pesquisa não se resume à possibilidade de ganho no desempenho. O FPGA possui um função crucial no processo de refinamento do projeto, pois a sua reconfigurabilidade permitiu que a arquitetura do HARP fosse evoluindo simultaneamente à especificação, com prototipagens para teste a cada especificação de serviço.

O fim deste processo iterativo resultou numa arquitetura implementada em hardware que atende aos requisitos de um protocolo de alta disponibilidade. Em virtude disso, esta seção expõe a arquitetura do HARP em sua primeira versão, que foi projetada e implementada em FPGA. O protocolo HARP está prototipado como um módulo de hardware no FPGA e sua arquitetura é mostrada na Figura 5.1. Nesta figura, o módulo HARP é a parte do sistema responsável pela comunicação entre os elementos que garantem a alta disponibilidade (HA) da rede, gerando e processando as mensagens de HA.

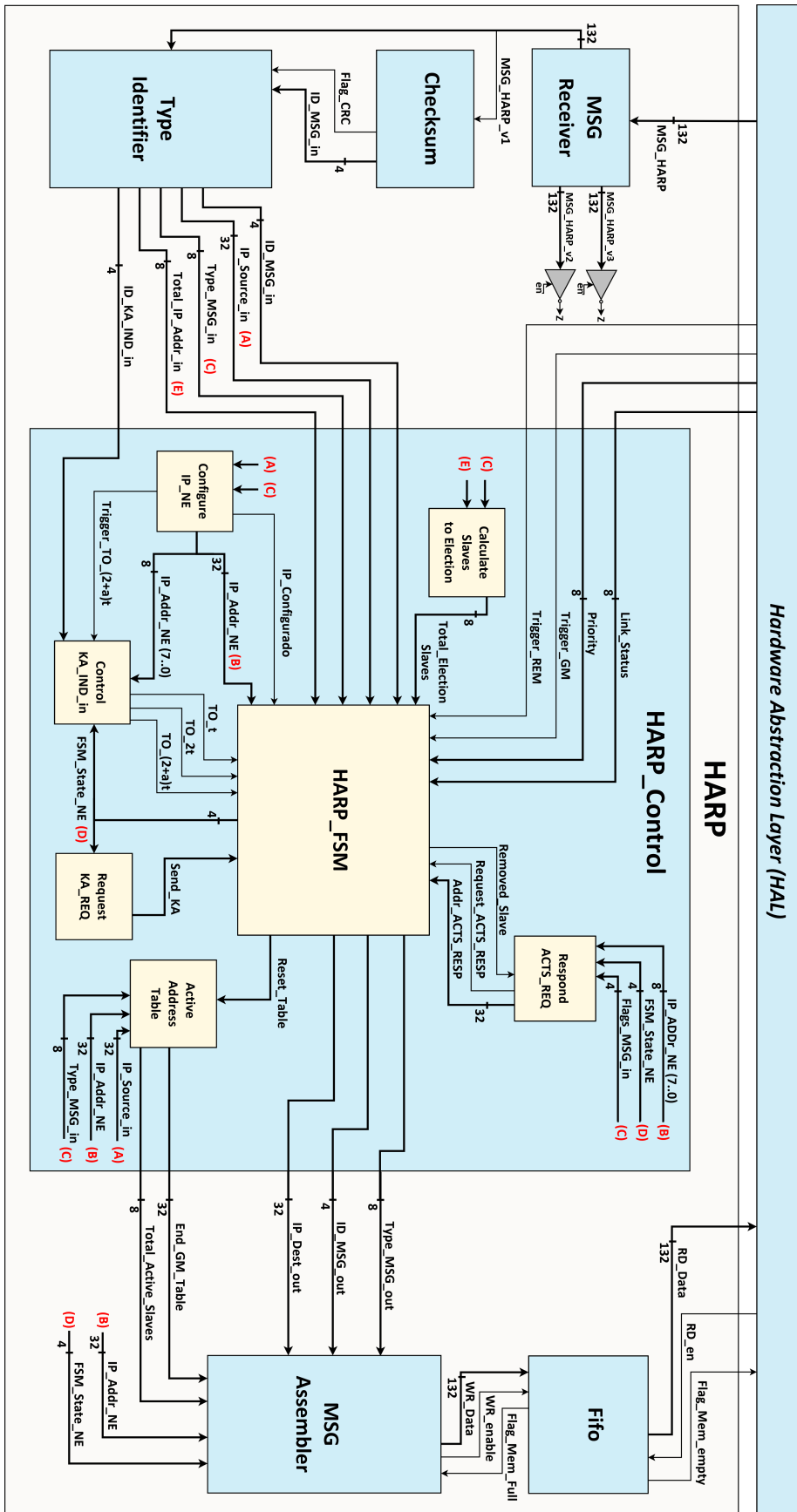


Figura 5.1: HARP - Diagrama em blocos da arquitetura compatível com IPv4

O módulo HARP pode ser acoplado a qualquer outra estrutura de hardware ou software para dar continuidade ao processamento das mensagens da arquitetura Internet. Tomando como exemplo o SoPC da Figura 3.4, outros componentes que implementam protocolos da Internet podem ser acoplados ao SoPC assim como o HARP o foi. Então, a pilha TCP/IP poderia valer-se de um acelerador em hardware. Por outro lado, pode-se implementar os demais protocolos em aplicações de software e fazê-los se comunicarem com o HARP através de um processador, tal qual o NIOS II faz por meio da aplicação de integração *IntegraSoC*.

Na parte superior da Figura 5.1, o módulo *Hardware Abstraction Layer* (HAL) representa uma camada criada para tornar transparente ao HARP questões de implementação e compatibilidade com ferramentas e outros sistemas, pois não se pode prever, por exemplo, como será o acesso ao registrador do enlace de estado da rede ou a qualquer um dos sinais com origem externa ao componente HARP.

Para uma melhor contextualização a este respeito, cabe voltar à Figura 3.4, onde o HARP ocupa o espaço destinado à DUT e a camada HAL está imediatamente antes do barramento *Avalon*. Nesta figura, o módulo *Reg_Status* é o registrador que armazena o estado do enlace do controlador de rede DM9000A e o coloca na via de dados *Link_Status* da arquitetura do HARP. O acesso a este registrador é dependente de cada equipamento, assim, ele ilustra bem a função da camada HAL.

A arquitetura é modular e formada por componentes que realizam funções independentes entre si. Seus canais de interligação internos atendem às especificações da mensagem do HARP, de forma que eles carregam campos e parâmetros das mensagens cujo formato é dado ilustrado pela Figura 4.1. O principal canal de dados de entrada é de 132 bits (da camada HAL para o *MSG Receiver*), que comporta uma mensagem de 128 bits adicionada de um identificador interno de 4 bits. O mesmo ocorre com o canal na saída do HARP (da *Fifo* para o HAL), este canal também possui 132 bits de largura. As demais vias internas se adaptam às exigências de cada módulo.

Os módulos, por sua vez, recebem os campos da mensagem de acordo com sua tarefa específica. Como se pode perceber pela Figura 5.1, há o módulo central *HARP Control* que é onde está implementado o autômato do HARP, na forma de uma máquina de estados finitos (MEF) (referente à Figura 4.5). É neste módulo que o controle principal do HARP é feito. O módulo *HARP FSM* implementa a troca de estados do autômato do HARP, recebendo sinais de *status* e gerando sinais de controle, enquanto os outros módulos se conectam a ele para realizar tarefas paralelas que diminuem a carga do módulo principal.

A maioria dos módulos da arquitetura geram sinais de controle para a máquina de estados, os que não os geram realizam algum processamento que indiretamente influencia nos valores destes sinais. Tais módulos são conectados entre si de forma que isso explora o paralelismo do hardware, para tornar o processamento mais rápido. A apresentação e descrição de cada um deles é dada nas próximas subseções, ainda neste capítulo.

Cada um dos módulos arquiteturais foi desenvolvindo de acordo com o fluxo de projeto apresentado na Subseção 2.1.1.1. Todos eles foram integrados em um único projeto de hardware (ilustrado pela Figura 5.1) e prototipados no FPGA. A descrição destes módulos é dada nas subseções a seguir, dados de simulação também são apresentados, no intuito de ilustrar a operação de cada um deles. Uma vez que os módulos da arquitetura do HARP estão combinados organizacionalmente como na Figura 5.1, esta figura é repetida em cada subseção para apresentar os componentes e suas interfaces, bem como as estruturas de interconexão entre eles, com destaque para o módulo em questão.

5.1 *MSG Receiver*

Este componente recebe as mensagens destinadas ao HARP. É composto por três módulos internos que encaminham as mensagens de acordo com a implementação do protocolo. São três implementações previstas: (1) compatível com IPv4, (2) compatível com IPv6 e (3) compatível com endereçamento baseado em identificador e localizador, por exemplo, *Identifier Locator Network Protocol* ILNP¹ (Atkinson e Andrews, 2012). Para o momento, o primeiro deles é implementado, enquanto os demais são passíveis de desenvolvimento.

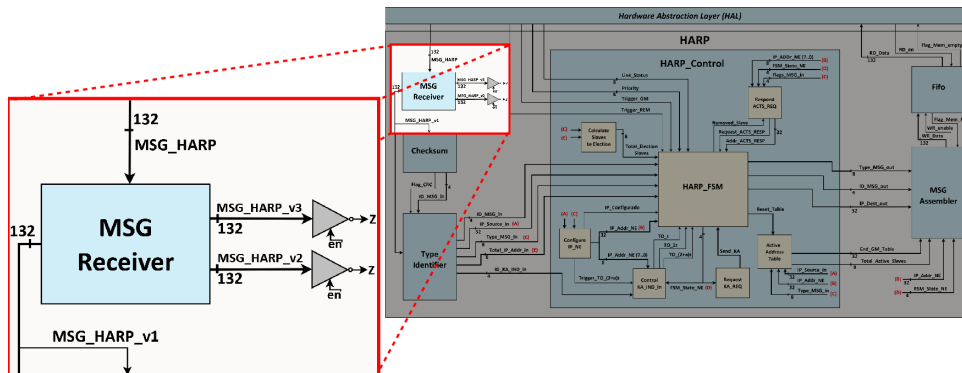


Figura 5.2: Componente *MSG Receiver*

A Figura 5.2 mostra a interface deste componente. Além dos pinos de *Clock* e *Reset*, este módulo possui dois barramentos em sua interface, cada um deles com 132 pinos. Pelo canal de entrada *MSG_HARP*, ele recebe a mensagem do *Buffer RX* (Figura 3.4) acrescentada de quatro bits à esquerda para a identificação desta mensagem. Através do canal de saída *MSG_HARP_v1*, ele entrega a mensagem ao módulo *Type Identifier*, que identifica o tipo da mensagem e extrai parâmetros em seus campos, e a entrega também ao módulo *Checksum*, que confere se a mensagem chegou sem erro.

¹ILNP é um protocolo experimental que provê, basicamente, as mesmas capacidades do IP. O ILNP propõe algumas melhorias em relação ao IP e redefine o espaço destinado ao endereçamento IPv6 para o conceito de Identificador e Localizador.

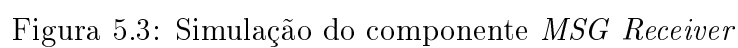
Ele é controlado por uma máquina de estados que permanece em execução ciclicamente por três estados depois do estado de inicialização. Durante este ciclo, o componente confere se há mensagem nova na entrada e, em caso positivo, controla a geração e propagação do identificador de mensagem (*ID_MSG*) para colocá-la na saída. É importante transmitir o identificador para o restante da arquitetura, pois a maioria das mensagens de entrada serão idênticas (*Keep Alive Indications*) e, então, este recurso possibilita diferenciá-las durante o processamento.

Na Figura 5.2, aparecem dois canais saindo da interface do *MSG Receiver* sem serem conectados a outros módulos. Estes canais indicam que a mensagem do HARP pertence a versões diferentes da atual. Isso é um indicativo de como ficaria o componente quando ele tiver que lidar com mensagens de outras versões do HARP. O *MSG Receiver* foi submetido à simulação para testar a saída no canal *MSG_HARP_v1* e os resultados aparecem na Figura 5.3.

A Figura 5.3 mostra que uma *string* de caracteres é colocada no canal de dados da entrada *MSG_HARP*, acompanhada do seu identificador *ID_MSG* nos quatro bits mais significativos. Na simulação, as mensagens são inseridas com identificadores de 0x1 a 0x4. Após três ciclos de *clock*, os canais de saída são atualizados.

A mensagem de entrada com identificador 0x3 contém uma versão diferente da versão 0x01, o que não é tratado por esta implementação. Como se pode ver na Figura 5.3, o campo de versão foi alterado para 0x02. Percebe-se, como resultado, que não há a atualização esperada da mensagem nos canais de saída onde a figura aponta o tempo transcorrido de 195 ns^2 (onde ocorre a mudança de estado). Assim, as duas saídas são atualizadas somente aos 245 *ns*, pelo conteúdo da quarta mensagem. O identificador *ID_MSG* que na entrada era 0x4, é corrigido para 0x3, para continuar com a sequência de mensagens entregues aos demais componentes internos.

²As marcações de tempo são utilizadas para fins de simulação. Nas figuras deste capítulo, elas não se referem aos tempos reais de operação do HARP.



5.2 Checksum

Este é o componente responsável por recalcular a soma de verificação e conferir se a mensagem chegou com ou sem erro. Suas interfaces são mostradas na Figura 5.4, onde vê-se um canal de entrada da mensagem e duas vias de saída com os resultados da operação.

Ele se conecta ao *MSG Receiver* pela via *MSG_HARP_v1* de 132 bits, já que este é o tamanho da mensagem acrescida do identificador. Possui também os pinos de *Clock* e *Reset* em sua interface, bem como dois canais de saída que o ligam ao módulo *Type Identifier* (*Flag_CRC* e *ID_MSG_in*).

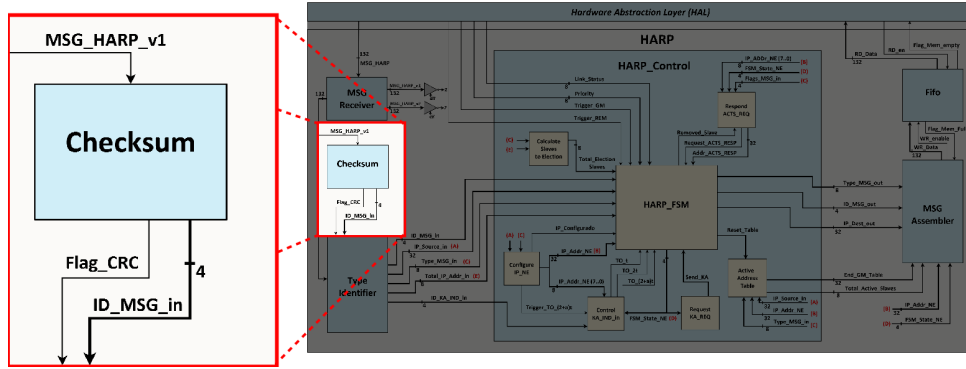


Figura 5.4: Componente *Checksum*

Pelo canal de entrada de 132 bits (*MSG_HARP_v1*) ele recebe a mensagem HARP com seu devido identificador interno. Do outro lado, os canais de saída *ID_MSG_in* e *Flag_CRC* carregam, respectivamente, o identificador da última mensagem processada e a ocorrência ou não de erro no seu recebimento. Para a *Flag_CRC*, o valor 1_2 indica que a mensagem está íntegra e não foi detectado erro, enquanto o valor zero 0_2 indica que houve erro de recebimento na mensagem e por isso, não se pode confiar em seu conteúdo.

O *Checksum* também é controlado por uma MEF de cinco estados (S0 a S4), onde o primeiro é apenas para inicialização de sinais. No segundo estado, o módulo verifica se há mensagem nova na entrada. Quando detecta nova mensagem, ele então leva mais três ciclos de *clock* para realizar a soma de verificação e apontar o erro nos canais de saída.

A Figura 5.5 mostra os resultados da simulação para o componente *Checksum*. O canal de entrada de dados *MSG_HARP_v1*, que recebe a mensagem e o seu identificador, está exibido em hexadecimal. Nesta simulação, foram inseridas sequencialmente seis mensagens, com as seguintes características:

1. *ID_MSG*: 0x1. Mensagem com o valor do *checksum* correto.
2. *ID_MSG*: 0x2. Mensagem com o mesmo conteúdo da mensagem 0x1, mas com o valor do *checksum* contendo erro de bit.
3. *ID_MSG*: 0x3. Mensagem com conteúdo diferente da 0x2 e com o valor do *checksum* correto.

4. *ID_MSG*: 0x3. Repete a mensagem anterior, inclusive o identificador.
5. *ID_MSG*: 0x4. Repete a mensagem anterior, porém com identificador diferente.
6. *ID_MSG*: 0x5. Nova mensagem com *checksum* errado.

A Figura 5.5 mostra os conteúdos das mensagens que foram colocadas no canal de entrada de dados. As seis primeiras mensagens possuem configurações conforme exposto na lista anterior. As mensagens com identificador 0x1 e 0x2 ilustram, respectivamente, os resultados nos canais para entrada livre de erro e com erro. Na parte destacada da figura é possível ver a borda descendente do sinal *Flag_CRC*, quando a mensagem 0x2 é colocada na saída.

Onde o cursor aponta 165 ns, não houve alteração na saída, isso porque o vetor de entrada *MSG_HARP_v1* manteve a mensagem anterior (0x03), o que leva o componente a entender que não houve atualização de conteúdo na entrada. Quando a mensagem 0x04 chega à entrada, há a atualização das saídas normalmente depois dos 240 ns, na transição para o estado S1, o que mostra que o componente consegue lidar com mensagens de mesmo conteúdo que cheguem sequencialmente para o HARP.

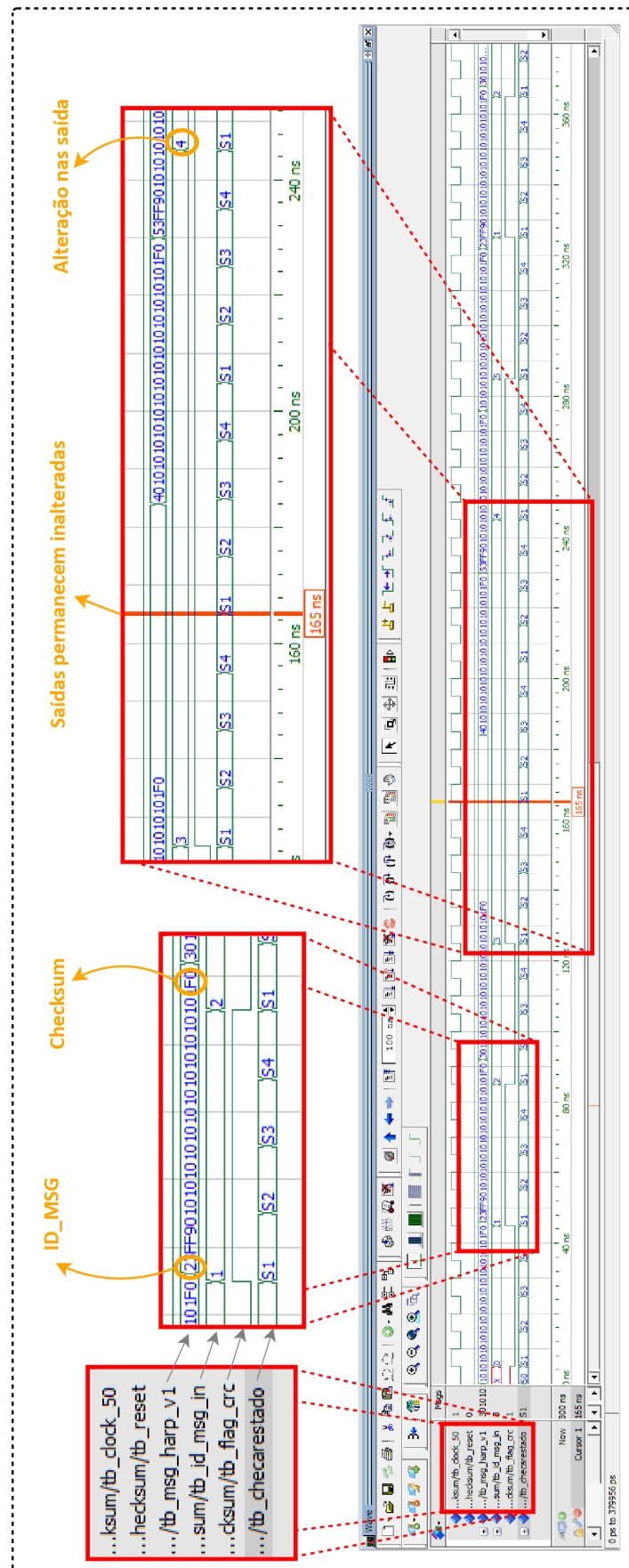


Figura 5.5: Simulação do componente *Checksum*

5.3 Type Identifier

Este componente recebe a mensagem do *MSG Receiver* e espera pela confirmação do módulo *Checksum* para certificar-se de que a mensagem está íntegra. A Figura 5.6 mostra os canais em sua interface que o conectam a estes dois módulos, bem como ao *HARP Control*.

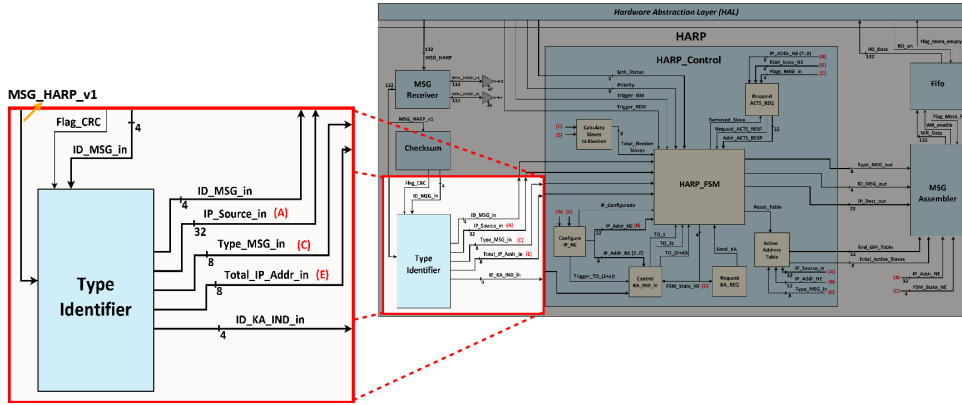


Figura 5.6: Componente *Type Identifier*

Ele identifica se o tipo de mensagem é compatível com a arquitetura implementada e extrai alguns parâmetros da mensagem para entregar ao controle da máquina de estados do HARP. Como se pode ver, ele possui vários canais de entrada e saída em sua interface. Estes canais foram criados para comportarem, separada e paralelamente, os campos das mensagens. É através deles que o componente *Type Identifier* consegue estabelecer a ligação entre os componentes de recebimento de mensagem e o controle da máquina de estados. A seguir, estes canais são explicados e é esclarecida a função de cada um:

- *MSG_HARP_v1*: Recebe do componente *MSG Receiver* a mensagem HARP com seu identificador. Conecta-se ao componente pela sua esquerda em uma via de 132 bits.
- *ID_MSG_in*: Recebe do módulo *Checksum* o identificador da mensagem processada por ele.
- *FLAG_CRC*: Indicação de que a mensagem processada pelo *Checksum* chegou com ou sem erro.
- *IP_Source_in*: Endereço de origem da mensagem.
- *Type_MSG_in*: Tipo da mensagem e a que serviço ela se refere.
- *ID_KA_IND_in*: Para o caso de uma mensagem *Keep Alive Indication*, este módulo também gera identificador próprio para permitir o controle de recebimento do *KA_IND*.

- *Total_IP_Addr_in*: Ainda no caso de um *KA_IND*, este canal indica quantos escravos estão ativos no grupo de HA.

Além do estado de inicialização (S0), sua máquina de estados permanece transitando sequencialmente entre dois estados (S1 e S2). Nestes estados ele verifica se há atualização dos dados de entrada (indicando que houve novo recebimento) e os coloca nos canais de saída. É importante lembrar que este módulo já extrai os dados da mensagem mesmo sem saber se ela está ou não livre de erros, entretanto somente os propaga depois que o módulo *Checksum* permitir que isso seja feito. Este é um exemplo dos casos em que o processamento de protocolos de rede se adequam à exploração do paralelismo espacial do FPGA.

A Figura 5.7 mostra o resultado da simulação deste componente. As mensagens 0x1 e 0x2 são de conteúdo idêntico, ambas sem erro de *checksum*, neste caso os sinais de saída recebem os parâmetros da mensagem. Já a mensagem 0x3 foi recebida com erro, como indicado pela *Flag_CRC* (na figura: *flag_crc_from_checksum*) que está em zero.

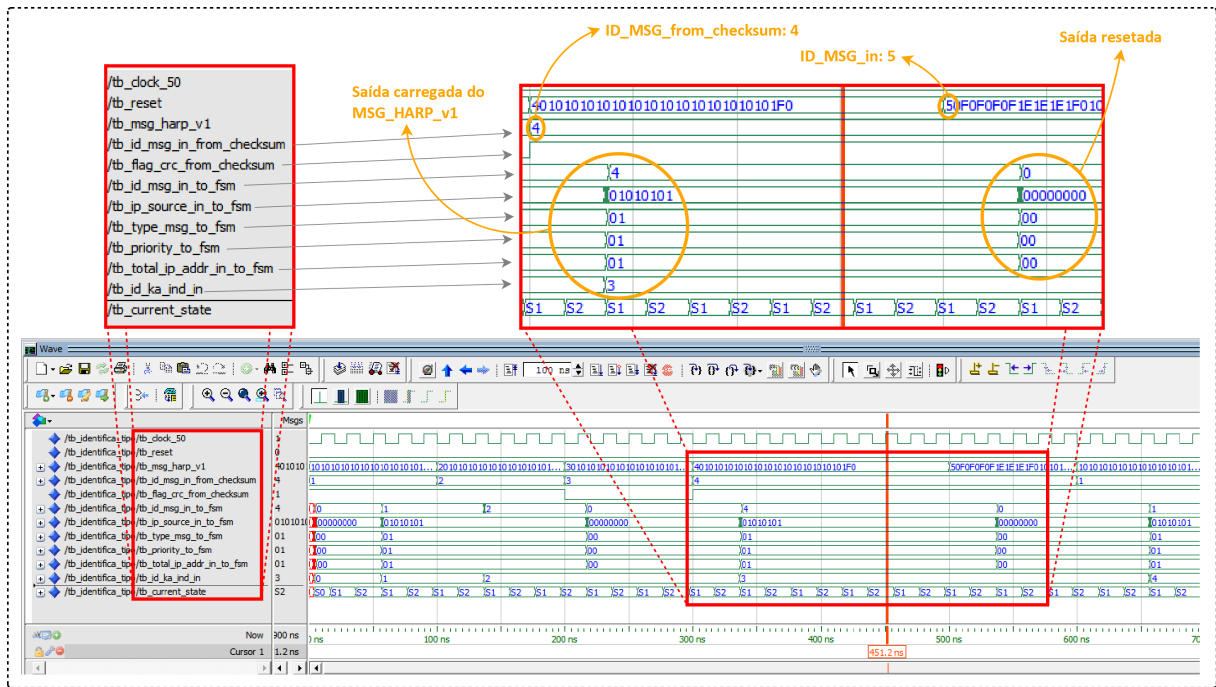


Figura 5.7: Simulação do componente *Type Identifier*

Outro fato importante ocorre quando a mensagem 0x5 é inserida na entrada. Apesar de o seu conteúdo e *checksum* estarem corretos, o canal *ID_MSG_in_from_Checksum* não atualiza seu valor para 0x5. Logo, pela incompatibilidade entre os identificadores recebidos dos módulos *Checksum* e do *MSG Receiver*, todos os canais de saída são resetados, evitando que a mensagem seja propagada sem que se tenha certeza da sua integridade.

5.4 Message Assembler

Este é o módulo responsável por montar a mensagem solicitada pela máquina de estados do HARP e colocá-la na memória circular de saída (*Fifo*) para envio. Os pinos da sua interface são melhor explicados se divididos em duas categorias: a interface com a *Fifo* e a interface com o *HARP Control*. A Figura 5.8 mostra esta distribuição.

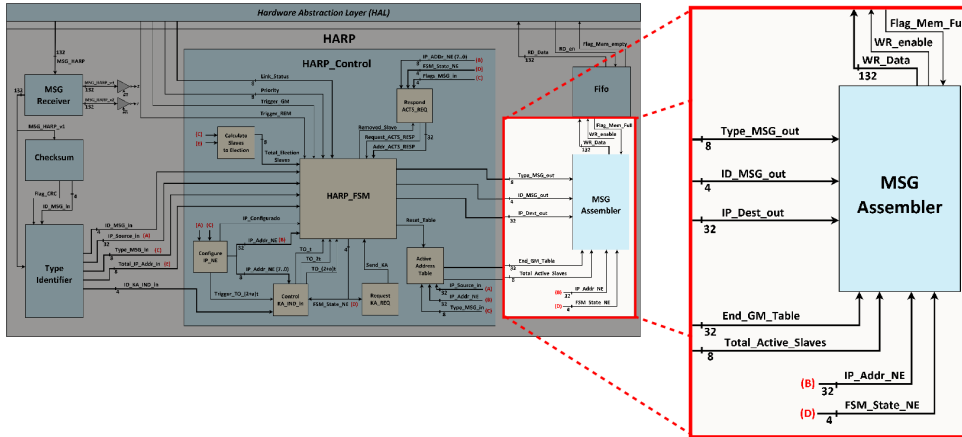


Figura 5.8: Componente *MSG Assembler*

As três vias de dados que na Figura 5.8 estão se conectando ao *MSG Assembler* pelo seu lado superior formam a interface com a memória *Fifo*. Enquanto isso, todas as outras vias de dados são conectadas ao *HARP Control*. A seguir, tem-se uma descrição de todas estas vias:

1. Interface com a *Fifo* para escrita da mensagem a ser enviada:

- *WR_enable*: Este pino é usado para indicar que uma operação de escrita será realizada na memória no próximo ciclo de *clock*.
- *WR_Data*: Canal para enviar à *Fifo* a mensagem e seu identificador que devem ser passados para a saída.

2. Interface com a máquina de estados (*HARP FSM*) e recebimento dos parâmetros para montagem da mensagem:

- *IP_Addr_NE*: Recebe o próprio endereço IP, que fica armazenado no módulo *Configure IP NE*.
- *FSM_State_NE*: Canal de controle que indica qual o estado atual do componente *HARP FSM*, no *HARP Control*. Os estados possíveis são os contidos no autômato da Figura 4.5.
- *ID_MSG_out*: Recebe o identificador da mensagem solicitada.
- *IP_Dest_out*: Recebe o endereço de destino da mensagem.

- *Type_MSG_out*: Recebe o tipo de mensagem que deve ser gerada (e serviço a que se refere).
- *Total_Active_Slaves*: Quantidade de escravos com endereços armazenados na tabela de ativos, i.e., quantidade de escravos que estão inseridos no grupo.

A máquina de estados que controla este módulo, além do estado de inicialização de sinais (S0), permanece sequencialmente transitando entre quatro estados (S1 a S4). Durante este período, ela realiza as seguintes operações: atualizar os dados dos pinos de entrada, verificar o estado da máquina de estados do HARP e o tipo de mensagem solicitada, concatenar parâmetros da mensagem, calcular *checksum* e incluí-lo no último campo do cabeçalho, gerar identificador para a mensagem e escrevê-la na *Fifo*.

O *MSG Assembler* tem sempre uma mensagem em sua saída, ele somente apaga a última mensagem transferida quando a substitui por outra. Entretanto, ele só ativa o pino de *WR_enable* durante um ciclo de *clock*, para que a escrita na memória ocorra apenas uma vez. A mensagem leva quatro ciclos para ser montada a partir do momento em que foi solicitada pela máquina de estados do HARP.

A Figura 5.9 mostra os resultados da simulação para este componente. Na transição para o estado S1, os parâmetros da mensagem são atualizados dos canais de entrada para formarem a nova mensagem de saída depois de mais três ciclos. Nesta mesma transição, a mensagem solicitada anteriormente é montada na saída. Como se pode ver na figura, o sinal que habilita a escrita na memória permanece em nível alto apenas por um ciclo, para evitar que a escrita ocorra em mais de uma posição na *Fifo*.

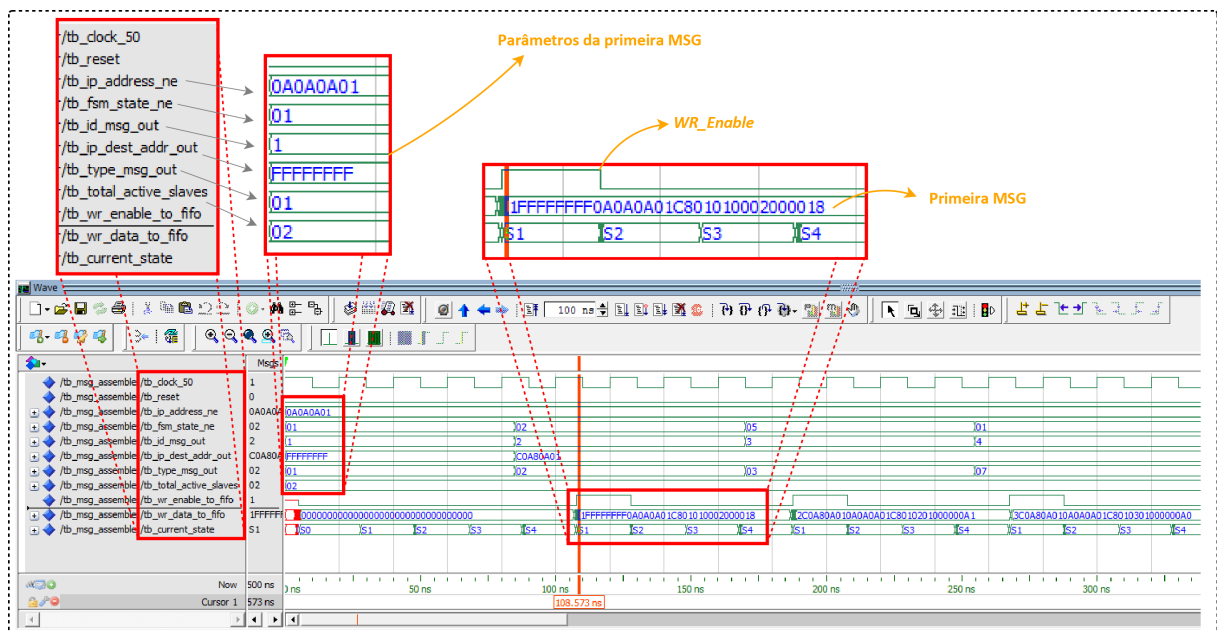


Figura 5.9: Simulação do componente *MSG Assembler*

A região destacada entre 0 ns e 25 ns na Figura 5.9 mostra que os parâmetros da mensagem 0x01 estão prontos para serem enviados. Alguns ciclos depois, a região destacada

entre 100 ns e 200 ns mostra estes mesmos campos concatenados em uma *string* para transmissão. O próprio componente calcula o *checksum* e o insere no fim do cabeçalho. O campo *Priority* depende do estado do autômato do HARP (na Figura 4.5). Se o estado do autômato for de mestre, a prioridade é zero, senão, este valor assume o último octeto do IP (*IP_Addr_NE*).

O componente *MSG Assembler* ocupa um total de 384 LEs no FPGA *Cyclone 2*. Este valor significa 12,5% do total de LEs ocupado pelo HARP e uma porção de aproximadamente 1,2% do total disponível no FPGA. Em relação à frequência de operação, o *MSG Assembler* consegue operar no *Cyclone 2* a 98,7 MHz de forma a manter a integridade dos dados.

5.5 Fifo

Esta é uma memória de escrita e leitura utilizada para enfileirar as mensagens que esperam para serem transmitidas ao restante da rede. Sua interface é conforme a Figura 5.10. As vias de dados da parte superior se conectam a componentes externos, enquanto os outros se conectam ao *MSG Assembler*.

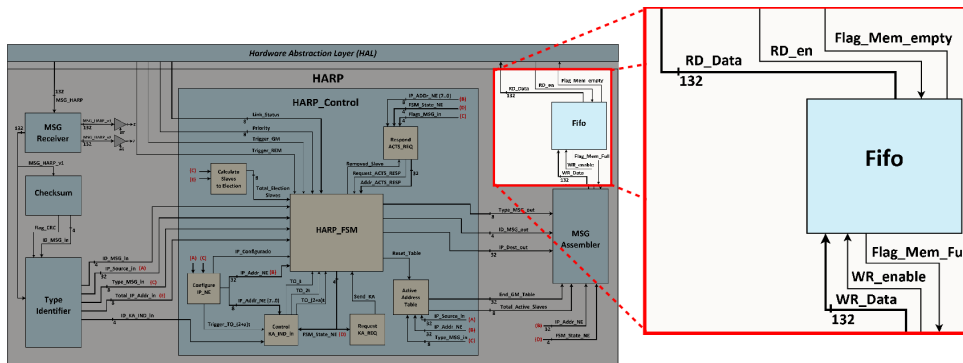


Figura 5.10: Componente *Fifo*

Sua escrita é controlada pelo módulo que monta as mensagens para o envio (*MSG Assembler*), enquanto a leitura é controlada pelo componente que recebe as mensagens do lado de fora. Neste caso, o componente é o *Buffer_TX*, conforme a Figura 3.4. Os pinos de dados e controle em sua interface, além do *Clock* e *Reset*, são:

- *WR_enable*: Habilita a escrita de dados na memória.
- *RD_enable*: Habilita a leitura de dados na memória.
- *WR_Data*: Canal de dados para escrita da mensagem.
- *RD_Data*: Canal de dados para leitura de mensagens armazenadas.
- *Flag_Mem_Empty*: Pino que indica que a memória está vazia e não há mais dados para serem lidos.

- *Flag_Mem_Full*: Pino que indica que a memória está cheia e não há mais espaço para inserir novas mensagens, a menos que outras sejam sobrescritas.

A *Fifo* é percorrida circularmente e está implementada como uma RAM 8x132 bits, embora possa ser configurada de acordo com a capacidade do hardware que se tem e pelo fluxo de mensagens solicitadas pela máquina de estados (*HARP FSM*). Ela é controlada por uma máquina de estados de dois estados: S0 (*Emptying*) e S1 (*Filling*). O primeiro deles indica que a última operação foi de leitura, já o segundo indica que a última operação foi de escrita.

O controle de posições é feito por dois contadores internos, um para escrita e outro para leitura, que são incrementados após uma operação de escrita ou leitura, respectivamente. Os pinos *Flag_Mem_Empty* e *Flag_Mem_Full* são controlados pelo estado da máquina de estados e a posição dos contadores. Quando estes se encontram, se o estado é *Emptying*, a memória está vazia e se o estado é *Filling*, ela está cheia. A Figura 5.11 mostra o resultado da simulação para este componente.

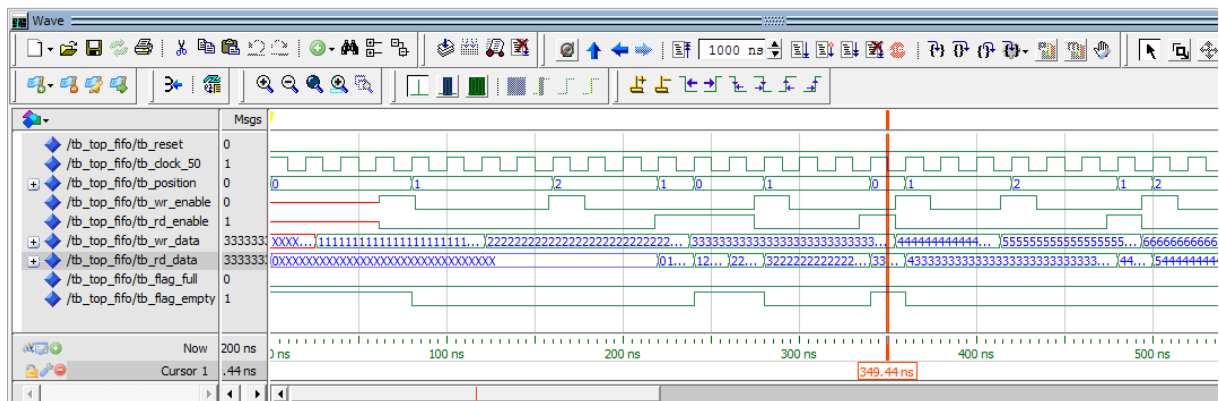


Figura 5.11: Simulação do componente *Fifo*

As mensagens são inseridas sequencialmente sempre que o pino *WR_enable* habilita a escrita. No canal *RD_Data*, as mensagens são lidas sequencialmente sempre que o pino *RD_enable* habilita a leitura. Se qualquer um destes dois sinais de *enable* estiverem em nível alto, há uma alteração na saída *position*, que indica a posição atual onde deve ser escrito o próximo dado na memória.

Os sinais *Flag_Full* e *Flag_Empty* são alterados de acordo com a leitura e escrita. No início da simulação, a memória estava vazia e, por isso, a *Flag_Empty* começou com o valor 1. Sempre que o endereço de leitura alcança o endereço da escrita, este sinalizador é setado, indicando que não há mais conteúdo. Estes dois sinais são levados para fora da entidade da memória para que o componente *MSG Assembler* veja que não há mais espaço para gravar dados e também para que a HAL veja que não há dado para ser lido.

5.6 *HARP Control*

De acordo com a Figura 5.1, percebe-se que há um conglomerado de módulos que recebe o nome de *HARP Control*. Este conjunto de módulos é o responsável pelo controle das transições da máquina de estados do HARP. O módulo *HARP FSM* é que efetivamente implementa a máquina de estados finitos (MEF) representada na Figura 4.5, mas para que isso ocorra, uma série de outros componentes realizam tarefas menos complexas, tirando a carga total do módulo *HARP FSM*, por exemplo, calcular número de escravos em uma processo de eleição e calcular o tempo decorrido desde que a última mensagem de aviso foi recebida.

Tal abordagem reduz o tempo que o HARP leva para gerar uma mensagem em seu hardware, pois compreende um conjunto de ações processadas paralelamente às operações da MEF, deixando que o módulo *HARP FSM* preocupe-se apenas com as transições entre estados. A MEF implementada é uma máquina de estados de *Mealy* (Mealy, 1955), pois suas transições dependem tanto do estado atual da MEF quanto dos valores dos sinais de entrada que trazem, entre outros, os parâmetros de mensagens.

A Figura 5.12 retoma a arquitetura e mostra detalhadamente a organização do componente *HARP_Control*. Ele é formado pela combinação de sete componentes e estes são explicados nas subseções a seguir.

O módulo *HARP FSM* é o principal deste conjunto, pois trata da mudança de estados do elemento de rede. Os outros seis realizam tarefas mais simples e, em virtude disso, os resultados da simulação do *HARP Control* são apresentados na Figura 5.13, quando todos os componentes menores já tiverem sido explicados.

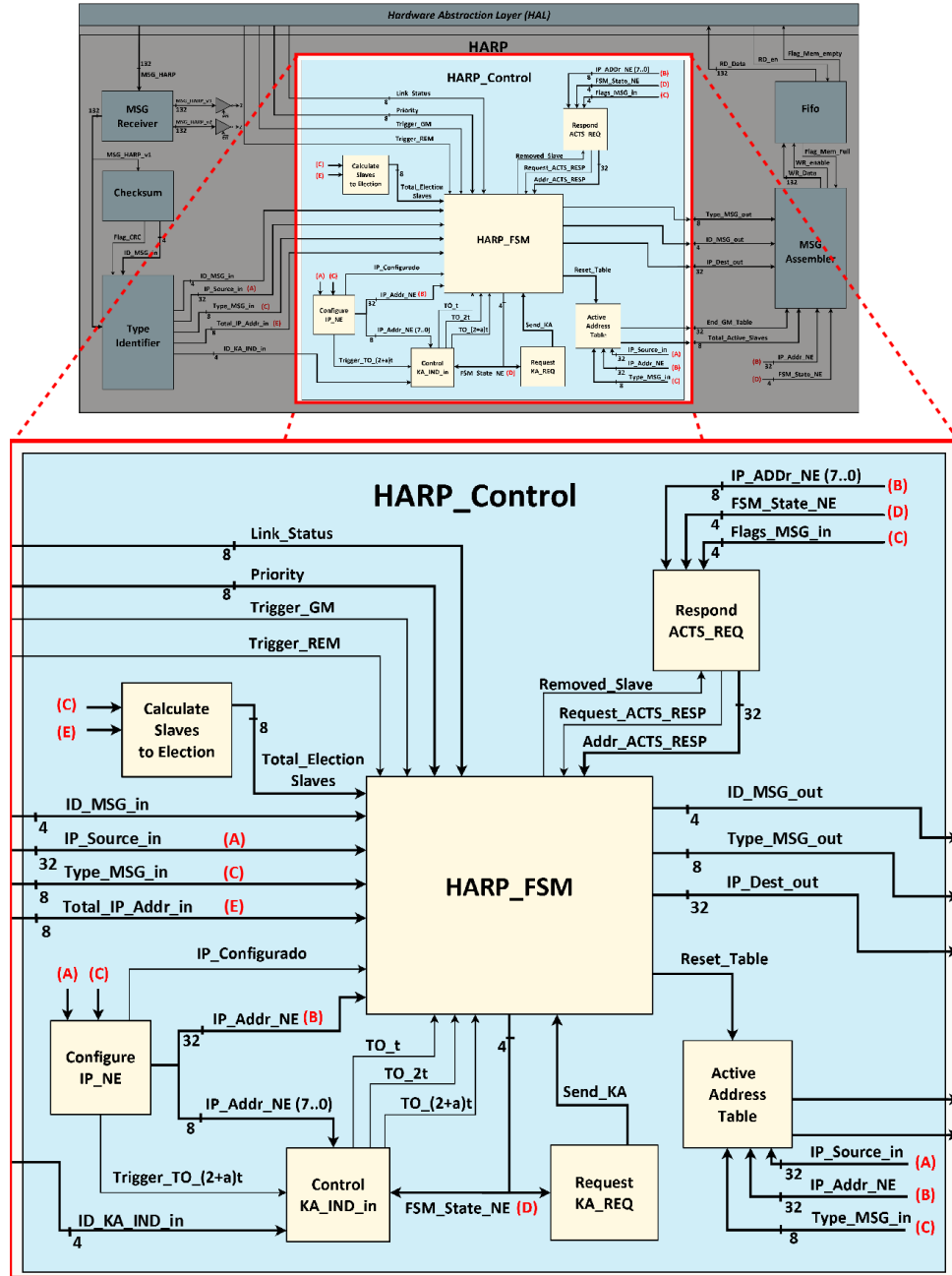
5.6.1 *Calculate Slaves to Election*

Este componente calcula o número de respostas que se deve esperar em um serviço de eleição a novo mestre (*Check Brain*). Ele recebe dois canais de dados do componente *Type Identifier*, um indica o tipo de mensagem recebida (*Type_MSG_in*) e o outro recebe a quantidade de escravos na tabela de ativos do mestre (*Total_IP_Addr_in*). Este último é extraído de um dos campos das mensagens *Keep Alive Indication*.

A cada ciclo de *clock*, o componente confere se chegou outra mensagem do tipo *KA_IND* e, então, atualiza o valor de escravos ativos. Em sua interface, há o canal de saída *Total_Election_Slaves* que se conecta à máquina de estados do HARP, por onde ele entrega o valor de respostas que se deve esperar em um serviço *Check Brain*.

5.6.2 *Configure IP_NE*

Este componente é o que trata da atribuição do endereço IP para o elemento de rede. Na verdade, o endereço IP já terá sido atribuído ao elemento de rede pelo protocolo IP,

Figura 5.12: Componente *HARP Control*

mas não há interação direta com o hardware do HARP. Assim que configurado o IP, este componente avisa à máquina de estados e aos outros módulo que podem começar a calcular os *timeouts* por recebimento de KA_IND. Os seguintes canais compõem sua interface:

- *IP_Source_in*: Recebe o endereço IP atribuído a este elemento.
- *Type_MSG_in*: Recebe o tipo de mensagem que foi recebida.
- *IP_Configurado*: Indica para a máquina de estados se o IP já foi, ou não, configurado.
- *IP_Addr_NE*: Repassa o endereço IP a outros módulos.

- *Trigger_TO_{(2+a)t}*: Indica ao *Control KA_IND_in* que já pode começar a contabilizar os intervalos sem recebimento de *KA_IND*.

Através dos canais de entrada *IP_Source_in* e *Flags_MSG_in*, ele identifica a chegada de uma mensagem *Config_IP* (Seção 4.2) e armazena o endereço IP deste NE. Em seguida, ele carrega o endereço no canal *IP_Addr_NE* e seta os outros pinos de saída, avisando à máquina de estados e ao módulo *Control_KA_IND_in* que o hardware está com o IP configurado. Assim, eles podem começar a calcular os intervalos entre sucessivos recebimentos de mensagens de aviso *heartbeats*, bem como iniciar o processo de eleição.

5.6.3 *Control KA_IND_in*

Módulo que calcula os *timeouts* e controla as *flags* do serviço *Keep Alive*, i.e., controla se o recebimento do *KA_IND* (*heartbeat*) tem acontecido conforme as configurações iniciais do protocolo. Ele possui três pinos de saída que se conectam ao módulo *HARP FSM*. O primeiro é para indicar se houve recebimento do *KA_IND* no último intervalo t , a segunda no último intervalo $2t$ e a terceira no intervalo $(2 + a) * t$, conforme estabelecido na especificação do HARP (Seção 4.4). Sempre que cada um dos *timeouts* ocorre, ele seta a via correspondente.

O módulo *HARP FSM* se liga ao módulo *MSG Type Identifier* através do canal *ID_KA_IND_in*. Este, por sua vez, armazena a identificação do último *KA_IND* recebido. Desta forma, o componente sempre reinicia todos os sinais de saída (que indicam *timeout*) a cada recebimento do *KA_IND*. Do outro lado, ele recebe um canal do *HARP FSM* indicando qual o estado em que a MEF se encontra, este canal é nomeado *FSM_State_NE*. Caso o NE seja mestre, o módulo não realiza nenhum processamento e zera todos os contadores, mas para todos os estados classificados como escravos, os contadores são incrementados a cada ciclo de *clock*.

Através do canal *IP_Addr_NE*, ele recebe o último octeto do IP deste NE. Este é o mecanismo utilizado para indicar a prioridade do nó para iniciar uma eleição. Assim, o componente utiliza este valor para calcular o termo $(2 + a) * t$ e setar o pino correspondente a este *timeout*, na interface com o *HARP_FSM*.

5.6.4 *Request KA_REQ*

Este módulo é utilizado apenas quando a máquina de estados desta instância do HARP está no estado de mestre. Este componente controla a solicitação do envio de *KA_REQ* a cada intervalo t . Em sua entrada, além dos pinos de *Clock* e *Reset*, há o canal *FSM_State_NE* que indica qual o estado atual da MEF do HARP. Em sua saída há um pino que é setado a cada intervalo t , indicando à MEF do HARP que ela deve solicitar um envio de *KA_REQ*.

5.6.5 *Active Address Table*

Trata-se de uma tabela mantida pelo mestre com todos os endereços de escravos ativos no grupo. Sua interface é composta pelos seguintes canais e pinos (além do *Clock* e *Reset*):

- *Reset_Table*: Além do *Reset* do sistema, a MEF utiliza este pino para limpar o conteúdo da tabela de ativos, por mudança de estado, por exemplo.
- *IP_Source_in*: Recebe do *Type Identifier* o endereço de origem das mensagens.
- *IP_Addr_NE*: Recebe o endereço IP deste NE.
- *Type_MSG_in*: Lê o tipo das mensagens que chegam ao *Type Identifier*.
- *End_GM_Table*: Passa ao módulo *MSG Assembler* o endereço do escravo que receberá o serviço de *Grant Master*.

Com estes canais, o componente verifica a cada ciclo de *clock* qual a mensagem que foi recebida. Ao receber uma das mensagens que indicam que um escravo está ativo no grupo (INF_IND ou um ACTS_INS), este módulo adiciona o endereço de origem da mensagem em sua memória. Caso seja recebida uma mensagem do tipo REM_IND, este endereço é retirado da memória e decrementa-se o contador de escravos.

Endereços repetidos não são acrescentados na tabela, por isso que a cada recebimento de mensagem, toda a tabela é percorrida. Os escravos que entram no grupo são inseridos sequencialmente na tabela. Assim, o contador de posições é utilizado para gerar a saída do pino *Total_Active_Slaves*. Para uma exclusão de endereço, a tabela é percorrida até encontrar este endereço para deletá-lo. Então, o último endereço da tabela passa a ocupar a nova posição vazia e o contador de posições é decrementado.

5.6.6 *Respond ACTS_REQ*

Componente utilizado quando a MEF do HARP está em estados diferentes de mestre, i.e., qualquer um dos estados classificados como estados de escravo (Seção 4.4.6). Este componente indica à máquina de estados quando ela deve solicitar o envio de uma mensagem respondendo ao mestre que o escravo está ativo. Para isso, os seguintes pinos e canais compõem sua interface:

- *IP_Source_in*: Recebe o IP deste NE.
- *FSM_State_NE*: Lê do módulo HARP_FSM o estado atual do autômato.
- *Flags_MSG_in*: Recebe do *Type Identifier* o tipo de mensagem que foi recebida.
- *Removed_Slave*: A máquina de estados avisa que, por razão programada ou não, este NE não está mais ativo.

- *Request_ACTS_RESP*: Solicita à máquina de estados que envie uma resposta de atividade ao mestre.
- *Addr_ACTS_RESP*: Endereço para resposta ao ACTS_REQ.

O módulo lê o estado da MEF do HARP e, no caso de escravo, ele monitora o canal que indica o tipo de mensagem recebida. Quando um ACTS_REQ é recebido, ele armazena o endereço da mensagem e, no ciclo seguinte, solicita à MEF que envie a mensagem ACTS_RESP. Esta solicitação é feita ao setar o pino *Request_ACTS_RESP*. Se o escravo tiver sido removido por algum motivo, este módulo não realiza nenhuma operação, permanecendo em *Idle*.

5.6.7 HARP FSM

Este é o módulo que realiza as operações relativas à mudança de estados do autômato na Figura 4.5. As tarefas paralelas à transição entre estados, bem como a geração de sinais de controle foram rateadas entre os outros módulos do *HARP Control* e apresentados anteriormente. Os resultados destas tarefas são entregues ao módulo *HARP FSM* através da sua interface.

Como se pode ver na Figura 5.1, a interface do módulo *HARP FSM* se conecta aos seguintes módulos da arquitetura do HARP: *Type Identifier*, *MSG Assembler*, *Calculate Slaves to Election*, *Configure IP NE*, *Control KA_IND_in*, *Request KA_REQ*, *Active Address Table*, e *Respond ACTS_REQ*. Todos os pinos e canais que os interligam foram apresentados juntamente com os outros módulos ao longo deste capítulo. Restam as vias que entregam dados (e controle) à MEF, vindos de componentes externos. Na Figura 5.12, eles se conectam à camada HAL (Figura 5.1) e são os seguintes:

- *Link_Status*: Estado da conexão do enlace de rede do elemento.
- *Priority*: Prioridade do nó no grupo HA.
- *Trigger_GM*: Indica a solicitação de um serviço *Grant Master*.
- *Trigger_REM*: Indica a solicitação de um serviço *Remove Node*.

Este componente é controlado por uma MEF de oito estados, conforme o autômato da Figura 4.5. Ele exporta um canal que indica qual o seu estado atual (*FSM_State_NE*), para que os outros módulos tomem as decisões relacionadas.

A última mensagem que chegou à sua interface fica sempre disponível na interface com o *Type Identifier*. Quando ocorre uma mudança no identificador de mensagem (*ID_MSG_in*), o *HARP FSM* lê os campos de dados desta mensagem e solicita ao *MSG Assembler*, que emita a resposta apropriada. O mesmo ocorre com sinais que vêm dos módulos auxiliares dentro do *HARP Control*. Se eles setam os pinos de *Send_KA*, *Request_ACTS_RESP* ou $TO_{(2+a)t}$ uma nova mensagem correspondente é solicitada ao *MSG Assembler*.

Como exposto, as regras que ditam a troca entre os estados neste componente são as especificadas na Seção 4.4. Tanto este módulo quanto os outros apresentados mantêm em suas interfaces os últimos valores colocados até que novos os substituam. Os componentes da arquitetura, ao perceberem a mudança de dados, tomam a decisão relacionada.

A Figura 5.13 mostra os resultados da simulação do componente *HARP Control*. Como se pode ver, o estado do enlace recebe o valor 0x40, pois é o valor no registrador de estado que indica que a conexão está ativa na placa DE2. A prioridade recebida inicialmente é zero, para que o nó se torne mestre.

O último dos vetores mostrados na simulação indica o estado do módulo *HARP FSM* correspondente ao autômato da Figura 4.5 (onde S1 é *Master* e S2 é *Slave*).

A cada 20 ciclos de *clock*, um *KA_IND* aparece nos canais de saída do módulo³. Este valor foi utilizado apenas na simulação para adaptar a visualização dos dados, o padrão é o *default* de 100 *ms* estabelecido na especificação. Vê-se que os novos dados substituem os anteriores e, então, permitem aos outros módulos trabalharem detectando os eventos nas vias de dados.

Inicialmente, envia-se os parâmetros da mensagem *Config_IP* ao elemento para que a operação seja habilitada nos componentes. Em seguida, ao receber uma mensagem que indica que um outro nó deseja entrar na rede (*INF_REQ*), o mestre envia a confirmação *INF_RESP* e, a partir de então, o campo *Total_Active_Slaves* passa a ser 1.

Uma solicitação de *Grant Master* é feita através do *Trigger_GM* (aproximadamente em 1250 *ns*) e, então, uma mensagem de *GM_REQ* é solicitada nos canais de saída e o estado da MEF vai ao S3 (espera pela confirmação do serviço GM). Ao receber a mensagem de confirmação (*GM_RESP*), ele continua com a troca de mensagens prevista pelo serviço e vai a *Slave* (S2), enviando também um *INF_REQ* em *broadcast*.

Transcorridos o período de 20 ciclos de *clock*⁴ sem nenhum recebimento de *KA_IND*, o nó que agora é escravo, começa o processo de eleição, passando pelos estados S5 e S6 (onde o cursor está apontando). Já que não há outros elementos escravos ativos na rede, ele não espera nenhuma resposta e vai diretamente ao estado de *Master S1*, quando, então, envia uma solicitação de confirmação de atividade dos escravos (*ACTS_REQ*) para montar sua tabela de ativos e começa o envio dos *KA_IND*.

³Valor utilizado apenas para fins de validação. O intervalo real entre mensagens de aviso é especificado no Capítulo 4.

⁴O valor 20 ciclos de *clock* foi adotado didaticamente para que o envio de sucessivos *KA_IND* pudessem ser vistos na Figura 5.13. O intervalo padrão em uma implementação do HARP refere-se ao parâmetro *t*, que aparece na Seção 4.4.

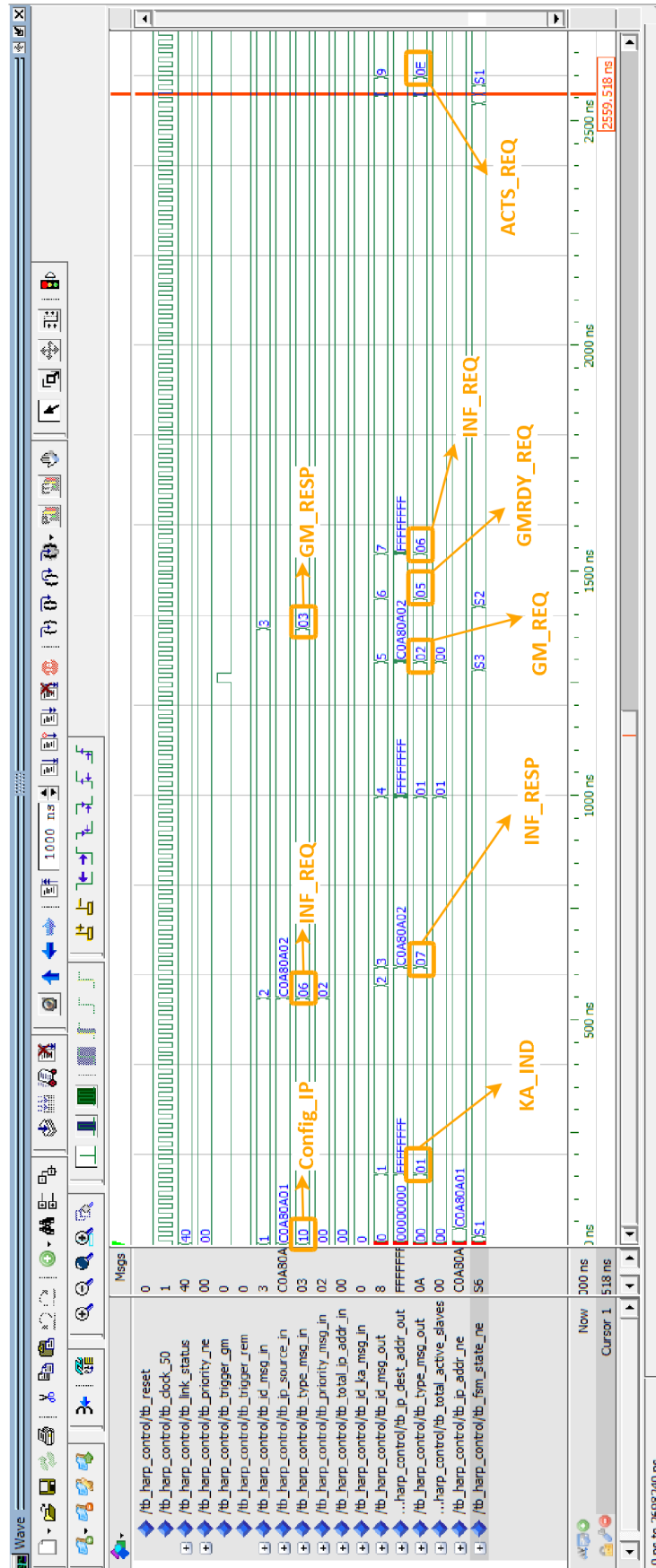


Figura 5.13: Simulação do componente *HARP Control*

5.7 Resumo dos dados de prototipagem

A arquitetura apresentada neste capítulo concerne ao HARP em sua primeira versão. Embora cada módulo da arquitetura recebe dados dos outros módulos, os processamentos realizados por cada um deles são diferentes entre si, o que implica em diferentes arranjos lógicos para sua implementação. Em virtude disso, eles são limitados por diferentes valores de frequência máxima de operação. O mesmo ocorre com a área ocupada. A Tabela 5.1 lista a ocupação do FPGA por cada componente e suas limitações de frequência.

Tabela 5.1: Dados de prototipagem dos componentes

Componente	Área Ocupada (LEs)	F_{MAX} (MHz)	Memória (bits)
<i>MSG Receiver</i>	599	179,31	0
<i>Checksum</i>	582	102,13	0
<i>Type Identifier</i>	308	261,99	0
<i>HARP Control</i>	1386	148,35	1024
<i>MSG Assembler</i>	384	98,7	0
<i>Fifo</i>	25	199,2	512

Como se pode ver, o componente *MSG Assembler* impõe que a frequência máxima de operação do sistema (F_{MAX}) seja de 98,7 MHz e por ser este o menor valor entre os listados, o hardware do HARP assume esta frequência como seu limite superior de operação.

Quanto à área de ocupação, a combinação dos componentes ocupa um total de 3284 LEs, o que significa menos de 10% dos 33216 LEs disponíveis no FPGA Cyclone 2, o que viabiliza a prototipagem de outros núcleos de processamento de rede no mesmo FPGA. Conforme exposto por Kachris e Vassiliadis (2006), núcleos de criptografia, compressão de dados ou detecção de intrusos são exemplos do que ainda pode ser prototipado neste mesmo FPGA.

Ao longo das simulações, foram utilizados em alguns componentes vias de dados e lógica adicional apenas para fins de validação. Um exemplo disso é o canal *Current_State*, que serve para acompanhar as transições da máquina de estados de cada componente. Isso gerou um aumento da área ocupada pelo HARP no FPGA de 221 elementos lógicos. Uma prototipagem do HARP puramente operacional atinge 3063 elementos lógicos, uma diferença de paroximadamente 7% em relação ao total anterior.

O uso de memória foi baixo e também não atingiu nem 1% do total disponível de blocos de SRAM interna. Na verdade, as memórias utilizadas pelo HARP são voláteis e, geralmente, pequenas. Assim, elas podem ser montadas utilizando apenas elementos lógicos para tornar o seu acesso ainda mais rápido.

Como exposto anteriormente, o FPGA Cyclone 2 tem 33216 elementos lógicos disponíveis para o hardware que se quer prototipar. O sistema de validação do HARP, cujo

hardware prototipado é descrito pela Figura 3.4 (SoPC), ocupa uma parcela de aproximadamente 30% (incluindo o HARP) de total de elementos disponíveis. Enquanto isso, o módulo HARP descrito pela Figura 5.1 ocupa uma área de aproximadamente 9% do total. Esta relação é claramente visualizada ao se observar a Figura 5.14.

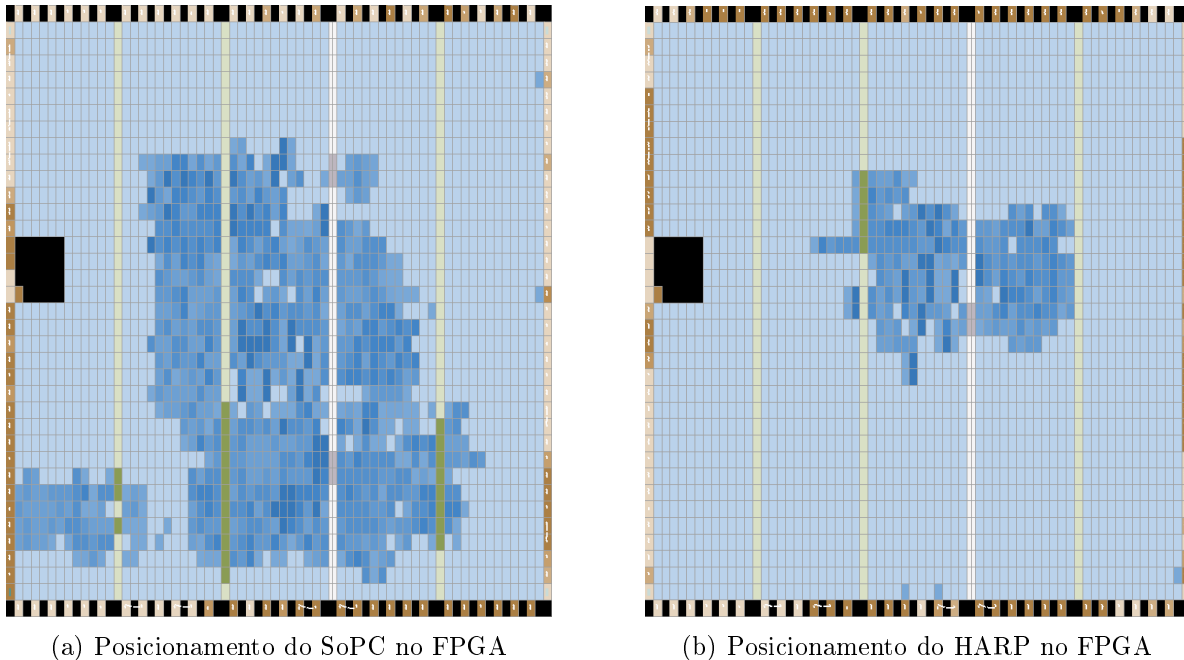
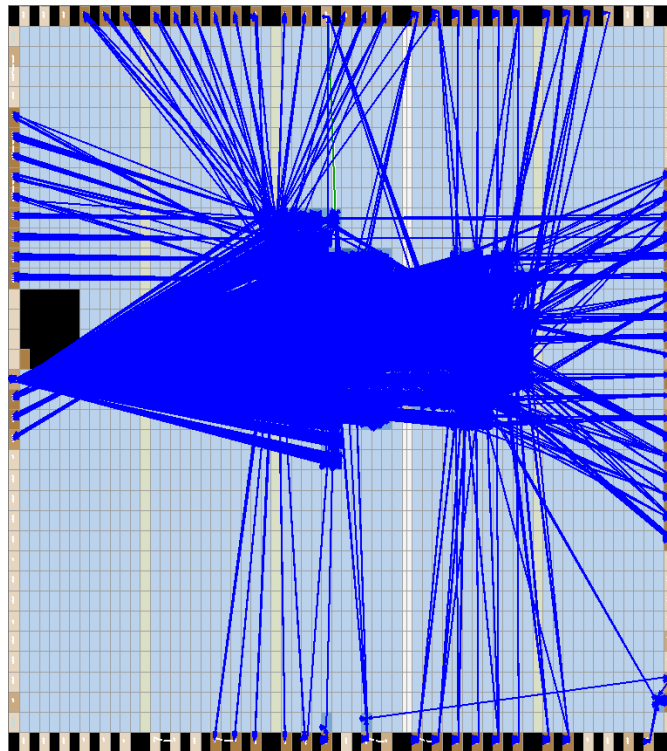


Figura 5.14: Posicionamento do hardware sintetizado no FPGA

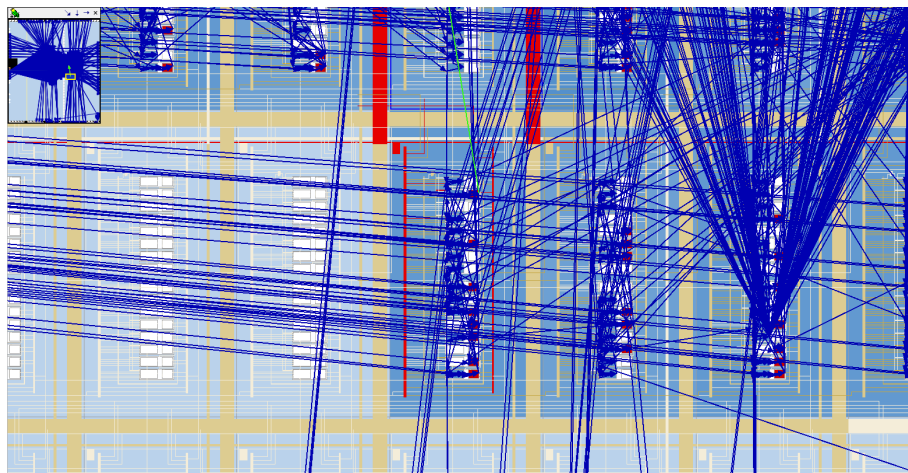
A Figura 5.14 mostra graficamente os detalhes do posicionamento de recursos no FPGA Cyclone 2. Os blocos em azul são elementos lógicos, enquanto as colunas intercaladas entre eles (geralmente com blocos verdes) são outros elementos como memória e DSP. Dos elementos lógicos, aqueles coloridos em tons mais escuros de azul são os utilizados pelo hardware em questão. O HARP ocupa uma área três vezes menor se prototipado sozinho, em relação ao seu sistema de validação. Vale lembrar que o FPGA Cyclone 2 é de baixo custo e, ainda assim, há muito espaço que pode ser utilizado simultaneamente ao HARP.

Na Figura 5.14(b), a região central que está destacada em azul mais escuro é a ocupada pelo HARP. A partir daquela distribuição de elementos lógicos no FPGA, o roteamento de dados na matriz de interconexão está configurado como na Figura 5.15. O emaranhado de caminhos em azul são as rotas de dados entre os elementos lógicos. Eles representam as interconexões internas e caminhos que os ligam aos blocos de entrada e saída. A densidade de ligações é grande o suficiente para cobrir a visão da área ocupada em elementos lógicos.

A Figura 5.15(b) aproxima a imagem do roteamento e mostra com mais detalhes as conexões. Nesta figura, há uma caixa em destaque no canto superior esquerdo que dá a visão geral do FPGA (a mesma Figura 5.15(a)) e um pequeno retângulo dentro desta caixa que diz qual a região cuja ampliação está sendo observada. Nesta figura, é possível



(a) Roteamento de dados do HARP no FPGA



(b) Ampliação da imagem de roteamento

Figura 5.15: Posicionamento do hardware sintetizado no FPGA

ver os elementos lógicos em blocos azuis, a matriz de interconexão em vias amarelas e os caminhos de dados em setas azul. A matriz de interconexão é que implementa as especificações de roteamento mostradas nos caminhos azuis de dados. A seta verde sobre as azuis é um caminho selecionado como exemplo, ele é implementado pela parte vermelha da matriz de interconexão.

5.8 Conclusão do Capítulo

Este capítulo apresentou a arquitetura do HARP projetada e prototipada em um FPGA de baixo custo Cyclone II. O hardware prototipado possui aspectos organizacionais que implementam a arquitetura da Figura 5.1. Esta arquitetura, por sua vez, é formada por módulos arquiteturais que se interconectam através de vias que se adaptam ao formato da mensagem do HARP. Estes módulos operam paralelamente entre si e exploram o formato multicampo das mensagens para acelerar o processamento no hardware.

A arquitetura foi construída conforme o fluxo de especificação, verificação, implementação e depuração do sistema, ou seja, todos os módulos passaram pelo processo de implementação e simulação antes de serem prototipados. Juntos, estes módulos compõem o hardware do HARP e este ocupa menos de 10% dos elementos disponíveis no Cyclone II. A frequência de operação do HARP atinge o valor máximo de 98,7 MHz.

Esta arquitetura modularizada permite que alterações sejam feitas nos módulos de operação auxiliar, sem que o módulo principal *HARP FSM* necessite de ser alterado. Da mesma forma, se houver alterações na especificação da máquina de estados que implementa o autômato da Figura 4.5, os outros módulos de operação auxiliar não precisam ser alterados. Desta forma, a etapa de construir um hardware para controlar a alta disponibilidade de um equipamento de rede foi concluída. A partir de agora, pode-se vislumbrar a construção do protótipo de um roteador que tenha o hardware do HARP como controle de alta disponibilidade e seguir, então, para os testes com operação em alto fluxo de mensagens.

Análise dos resultados

Em se tratando de resultados, salienta-se que os principais resultados desta dissertação foram apresentados nos Capítulos 4 e 5. A especificação do novo protocolo e sua arquitetura, por si só, são as máximas desta pesquisa.

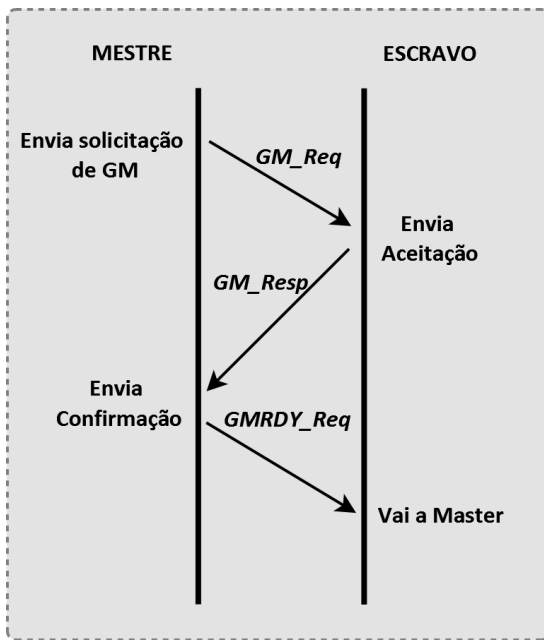
Este capítulo destina-se a apresentar uma análise da operação do HARP no sistema de validação montado, bem como deixa claro que o HARP está apto a operar em redes Gigabit. Primeiramente, a discussão é voltada para a execução funcional do protocolo (Seção 6.1). Em seguida, considera-se parâmetros tais quais o seu tempo de convergência e frequência de operação do hardware (Seção 6.2). Por último, a atenção se volta à prototipagem e uso do FPGA (Seção 6.3).

6.1 Operação funcional do HARP

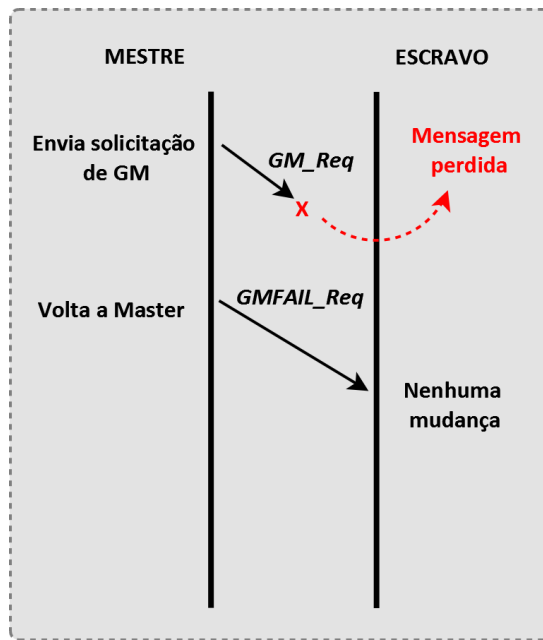
Retomando a discussão sobre a operação do HARP, convém observar a Tabela 3.1. São listados cinco processos para avaliação e que demonstram uma operação funcional do HARP. Os resultados destes processos listados são apresentados na forma de diagramas de sequência (para troca de mensagens) que descrevem exatamente as trocas de mensagens durante os serviços. Cabe lembrar, também, que o sistema é composto de três FPGAs representando elementos de rede. Desta forma, o total de nós esperado no grupo é um mestre e dois escravos.

1. Processo 1: Resposta ao serviço de transferência espontânea do mestre

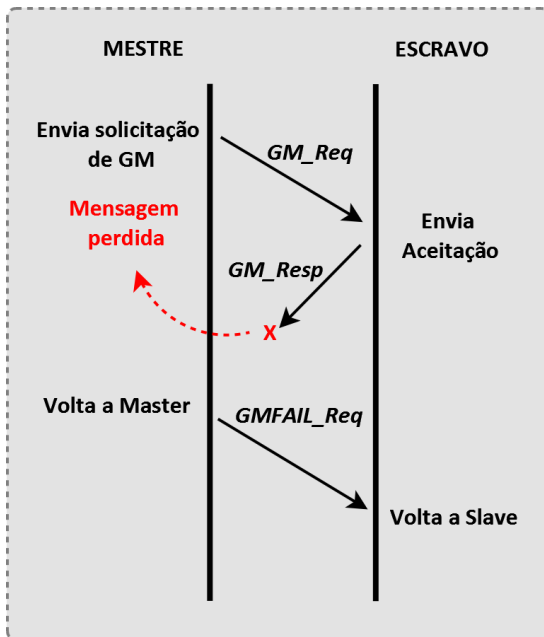
Os quatro comportamentos representados na Figura 6.1 foram constatados. O primeiro modelo ocorre predominantemente mais que os outros. Os três restantes apenas ocorrem se houver erro de mensagem e isso depende da taxa de erro do



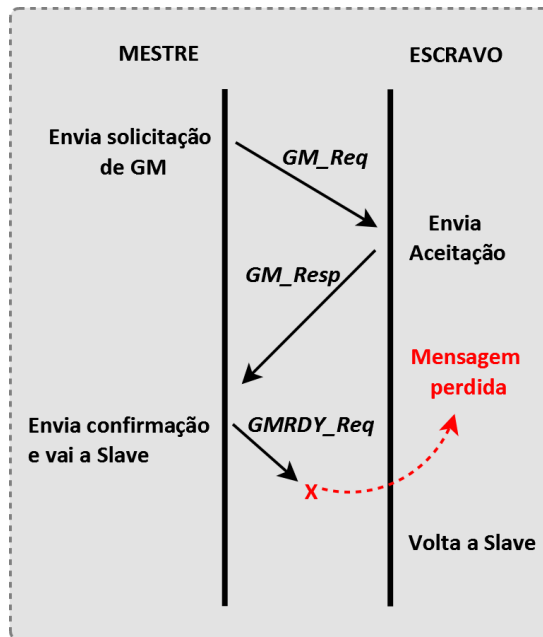
(a) Execução correta



(b) Execução com erro de mensagem



(c) Execução com erro de mensagem



(d) Execução com erro de mensagem

Figura 6.1: Autômatos do serviço *Grant Master*

sistema (Holzmann, 1991). Para este sistema de validação não foi possível medir a taxa de erro de forma confiável, pois não há fluxo intenso de mensagens em trânsito. Entretanto, verificou-se a ocorrência das quatro situações de forma não probabilística.

A Figura 6.1(d) deixa o sistema sem nenhum mestre, mas invoca automaticamente o serviço de *Check Brain*. Assim, as três primeiras situações satisfazem diretamente à variável M1 (*Novo Mestre Resolvido*), enquanto a situação quatro se une a outro

processo para satisfazê-la.

2. Processo 2: Eleição de novo mestre a partir de queda do atual

Os nós escravos foram configurados com prioridades 1 e 2 e nomeados Escravo 1 e Escravo 2, respectivamente. Os resultados constatados estão na Figura 6.2.

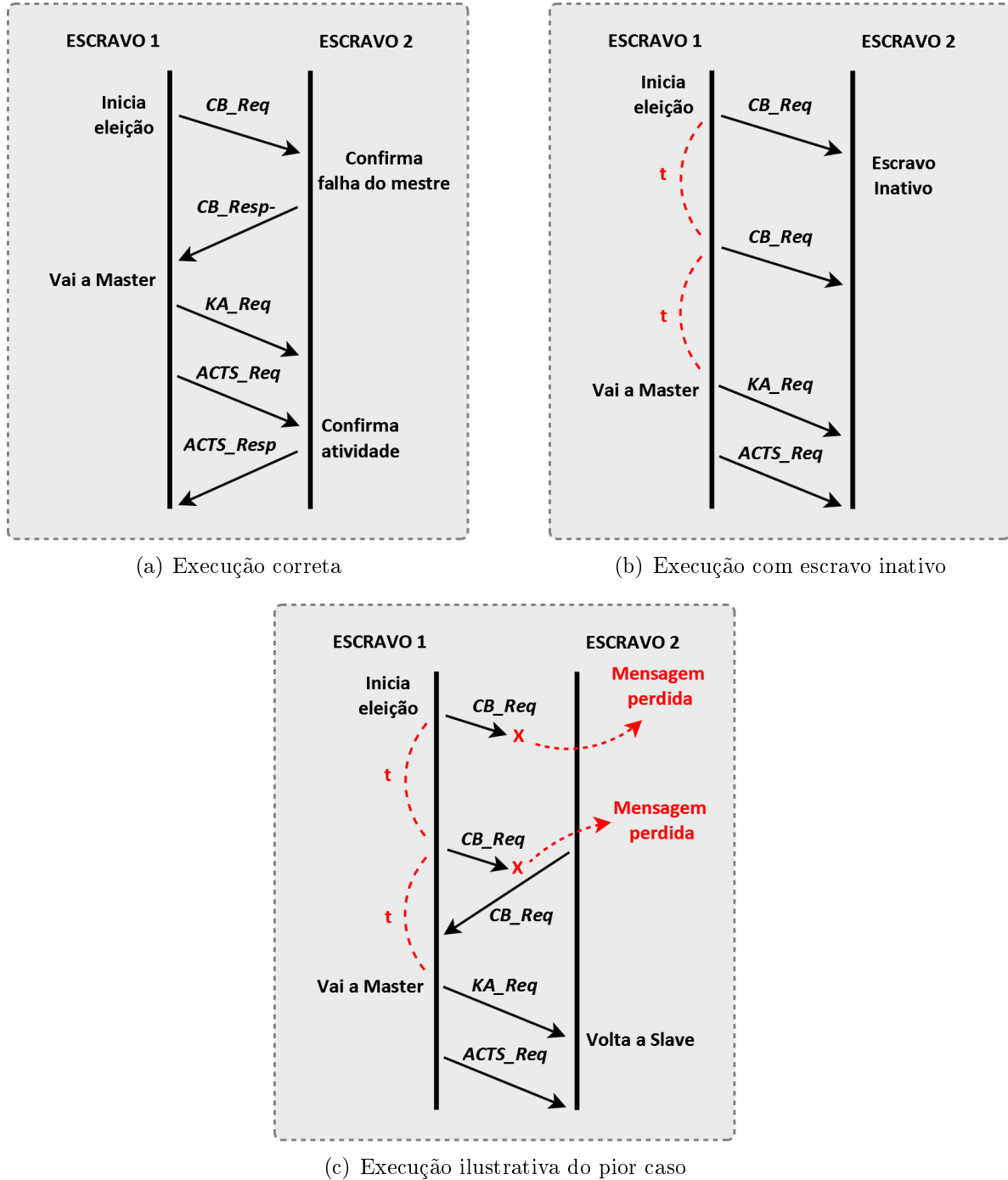


Figura 6.2: Autômatos do serviço *Check Brain*

As duas primeiras situações, nas Figuras 6.2(a) e 6.2(b), são os resultados experimentados. No primeiro caso, ao perceber a falha, o Escravo 1 começou o processo de eleição e recebeu a confirmação esperada, indo então a *Master*. Já no segundo caso,

o Escravo 2 estava inativo. Quando isso acontece, o escravo requisitante repete o processo e a cada repetição ele decrementa de uma unidade o número esperado de respostas, para que o processo tenha convergência. Quanto mais escravos tornarem-se indisponíveis imprevisivelmente, mais o processo demora para ser concluído.

No terceiro caso, da Figura 6.2(c), ocorre de a mensagem enviada pelo Escravo 1 solicitando confirmação da falta do mestre se perder. Assim, ao atingir seu TO_3 , o Escravo 2 também inicia o processo, mas não é respondido devido ao Escravo 1 já estar participando de um processo de eleição. Por julgar que não há outro escravo na disputa, o Escravo 1 repete o processo até tornar-se mestre. Ao enviar o *heartbeat*, o Escravo 1 faz o Escravo 2 voltar a escravo.

Este terceiro (Figura 6.2(c)) caso não ocorreu experimentalmente, mas introduz a ideia de pior caso. Se todas as primitivas se perderem, tanto requisições quanto respostas, os dois escravos se tornarão mestre. Daí, se eles recebem *heartbeats* uns dos outros, eles voltam a escravo e acionam o processo P2. O pior caso acontece em ambientes demasiadamente instáveis que, por fatores práticos, fogem à filosofia da estrutura provedora de alta disponibilidade.

Desta forma, a variável M1 continua sendo satisfeita. Percebeu-se que o protocolo pode levar um tempo maior que o esperado para convergir, mas ainda assim consegue um novo mestre.

O processo P3 (*E3: Queda de Canal do Mestre*), quando acionado gera os mesmos resultados que P2. Já os processos P4 (*E4: Queda de Canal do Escravo*) e P5 (*E4: Queda do Escravo*) não geram sozinho nenhum impacto no sistema, pois não há nenhuma troca de mensagem pelos canais. Todavia, estes dois últimos podem impactar se ocorrem em concomitância com P2 ou P3. Nestes casos, eles levam também à situação ilustrada na Figura 6.2(b), onde há eleição com tempo de convergência maior. A variável M2(*Escravo Permanece Escravo*) também é satisfeita.

6.2 Análise do Desempenho do HARP

Dividida em duas partes, esta seção realiza uma análise do desempenho do HARP. A Subseção 6.2.1 traz uma análise do tempo de convergência do protocolo, enquanto a Seção 6.2.2 atenta-se aos parâmetros de frequência de operação do hardware.

6.2.1 Considerações de Tempo

O tempo de percepção de falha para protocolo de HA na literatura tem seu melhor desempenho no VRRP, que considera um intervalo de 3 segundos para a percepção da

falha, mais um tempo esperado da ordem de $\ll 1\text{seg}$ para a convergência do protocolo (IETF, 2004) (Bhagat, 2011).

O HARP tem este intervalo flexível para que se possa observar o tempo médio de transmissão de mensagens no grupo (t_m). O valor padrão é 100 ms, reduzindo o tempo de percepção de falha para 300 ms. Do ponto de vista do roteamento, é importante que a perda de mensagens entre nós finais ocorra no menor número possível. Assim, propõe-se que os elementos mantenham *buffers* que armazenem as mensagens durante o período de percepção de falha, i.e., *buffers* que consigam armazenar as mensagens que transitaram no último período de 300ms, sempre atualizando-as. Desta forma, restaria apenas o tempo de convergência do protocolo para que estas mensagens fossem retransmitidas para seus destinos - pelo menos as de maior prioridade.

O tempo de convergência do protocolo HARP (t_{prot}) depende do número n de escravos na rede e o tempo médio para propagação das mensagens (t_m) entre os nós. A equação (6.1) descreve matematicamente este tempo de convergência.

$$t_{prot} = t_m + \frac{n}{2} \cdot t_m \quad (6.1)$$

O primeiro termo do lado direito da equação representa o t_m de uma mensagem enviada pelo iniciador do processo de eleição. Já o segundo, indica a soma dos t_m de metade dos escravos existentes no grupo, para ratificar a falta de mestre. O termo t_m pode definir a ordem de grandeza do tempo de convergência do protocolo e varia de ambiente para ambiente. O ambiente de operação do HARP, neste caso, é o construído para sua validação. Desta forma, um estudo foi feito com o sistema de validação para calcular qual o tempo médio de transmissão de mensagem no sistema e como os interfaceamentos hardware/software impactaram nestes resultados.

Dado que o sistema de validação integra processamento em hardware e software, várias etapas para interfaceamento são incluídas no processo de transferência de uma mensagem. A Subseção 3.3.2 mostra a aplicação de integração *IntegraSoPC* e deixa clara a sequência de execuções no sistema desde o momento da geração de uma mensagem até o seu recebimento na outra ponta da rede. Esta discussão é retomada aqui para uma análise quantitativa sobre o impacto de cada um destes passos no tempo médio de transmissão de uma mensagem.

É oportuno observar a Figura 3.5. Nota-se que a transmissão e o recebimento de mensagens são manipulados dentro das rotinas que atendem as interrupções por uma sequência das etapas. Estas, por sua vez, detalhadas na Subseção 3.3.2, estão rotuladas pelos contadores resumidos a seguir:

- T_m : Tempo médio de transmissão de mensagens.
- T_A : Interface entre o HARP e o barramento.
- T_B : Adapta mensagem para o formato da saída Ethernet.

- T_{CE} : Tempo decorrido entre a mensagem começar a ser enviada de um lado e ser recebida do outro (entre interfaces Ethernet).
- T_F : Adapta mensagem para o formato HARP.
- T_G : Interface entre o barramento e o HARP.

Para contar quanto tempo cada etapa consome, são utilizados um nó emissor e outro receptor de mensagens. Inseriu-se um contador de ciclos de *clock* em cada um deles (acoplado ao SoPC) para calcular todas as etapas de um mesmo processo de transmissão/recepção. Os valores são armazenados em vetores no FPGA e então lidos. A Tabela 6.1 mostra o t_m (ou T_m) neste cenário, bem como os tempos médios consumidos por cada etapa, com uma certeza de 99,881% sobre cada valor.

Tabela 6.1: Custo das etapas da aplicação

Contador	Tempo Médio (ms)
T_m	3,172200
T_A	0,301892
T_B	0,116248
T_{CE}	2,429328
T_F	0,176670
T_G	0,144084

Como se pode observar, de acordo com a Tabela 6.1, o t_m neste sistema é de aproximadamente $3,2\text{ ms}$, tornando o tempo de convergência do HARP $t_{HARP} = 6,6\text{ ms}$, bem mais preciso que o padrão $t_{prot} \ll 1\text{ seg}$. Como dito anteriormente, é evidente que o t_m depende do cenário. Para que se tenha uma visão proporcional destas etapas no processo completo, os dados da Tabela 6.1 são apresentados na Figura 6.3 denotando a parcela do custo que cada uma detém. Inclui-se, neste caso, o tempo que o HARP leva para gerar uma mensagem em seu hardware (T_{HW}).

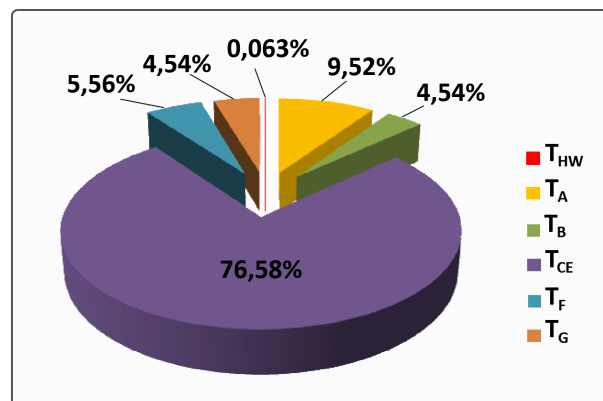


Figura 6.3: Custos das etapas na transmissão de uma MSG

De acordo com a tabela, fica evidente que o que mais custa neste processo é o tempo que uma mensagem leva para ser transmitida entre duas interfaces de rede (T_{CE}), que ocupa mais de 75% do tempo. Isso é dependente de cada tecnologia. As outras etapas, que juntas somam aproximadamente 23,3% do tempo, também são dependentes da implementação e serão menores se os outros protocolos da arquitetura Internet estiverem também implementados em hardware. Ainda segundo a Figura 6.3, o tempo que o HARP consome, efetivamente, para gerar uma mensagem não atinge nem 1% do tempo total requerido pelo sistema para transmissão de uma mensagem.

Para o sistema de validação, o gráfico da Figura 6.4 apresenta as previsões de tempo, considerando um aumento progressivo no número de elementos de rede.

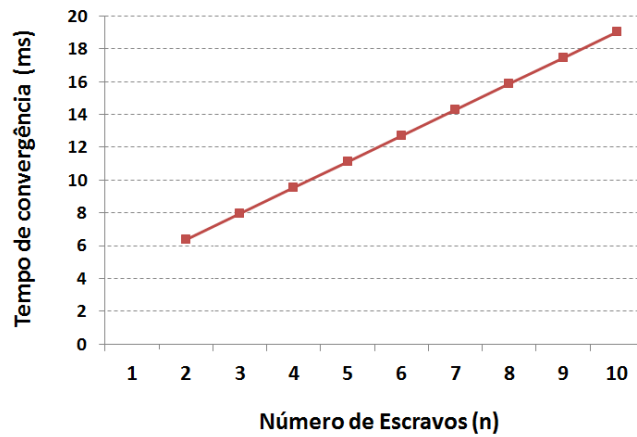


Figura 6.4: Gráfico (t_{prot} x Quantidade de Escravos)

Neste gráfico, o eixo das abscissas apresenta o número de escravos na rede, enquanto o eixo das ordenadas traz o tempo de convergência, em milissegundos, considerando o T_m da Tabela 6.1. O gráfico atém-se a um limite máximo de 10 escravos, uma vez que não é comum ter arquiteturas de alta disponibilidade com quantidades maiores que esta de elementos redundantes (Woodberg *et al.*, 2007). Conclui-se, então, que o t_{prot} continua na escala de milissegundos, demonstrando o HARP em operação compatível com o estado da arte.

6.2.2 Considerações de Frequência

Sabendo que o HARP não é protocolo de roteamento, ou qualquer nível de encaminhamento de pacotes, não é possível medir a sua eficiência quanto a este quesito. Por isso, objetiva-se fazer com que o HARP acompanhe a velocidade de processamento dos protocolos de encaminhamento e às potencialidades das linhas de transmissão. Assim, adotou-se para esta análise a condição de que o hardware do HARP deve ser capaz de

gerar as mensagens, no mínimo, em consonância com a taxa de transmissão dos canais de comunicação. Isto é, se um enlace pode transmitir a 100 Mbps, as mensagens devem ser geradas a 100 Mbps. Isso garante que o HARP, quando acoplado a outros módulos de rede, terá um desempenho que tenda ao ótimo.

A equação (6.2) foi desenvolvida para determinar a frequência mínima a que o circuito do HARP deve operar para que os requisitos de transmissão, segundo a velocidade do canal, sejam obedecidos. Assim, a descrição da relação é que o HARP leva para gerar um bit, o mesmo tempo que o canal demora a transmiti-lo. Na equação, o termo 208 é adicionado em virtude dos 26 bytes acrescentados no encapsulamento Ethernet, significando: preâmbulo (7 bytes), *Start of Frame* (1 bytes), MAC de origem (6 bytes) e de destino (6 bytes), Tamanho (2 bytes) e CRC (4 bytes).

$$f_{HARP} = \left(\frac{(n_{bits} + 208) \cdot t_{bit}}{n_{cycles}} \right)^{-1} \quad (6.2)$$

Onde:

n_{bits} : número de bits em uma mensagem HARP;

t_{bit} : tempo para transmitir um bit no canal;

n_{cycles} : ciclos de *clock* requeridos pelo HARP para gerar uma mensagem.

O numerador da equação significa o tempo necessário para que uma mensagem seja transmitida pelo canal. Este valor é considerado o tempo máximo que o HARP deve utilizar para gerar uma mensagem. Dividindo-o pelo número de ciclos que o HARP leva para gerar a mensagem, tem-se o período requerido por ciclo e, assim, a frequência mínima de operação.

As mensagens do HARP possuem um tamanho fixo de 128 bits. De acordo com a arquitetura da Figura 5.1, uma mensagem leva 10 ciclos de clock para chegar à saída do HARP a partir da sua solicitação, desconsiderando a necessidade de filas. É importante lembrar que o número de ciclos é fixo, pois o tamanho de mensagem é fixo. Caso a mensagem tivesse tamanhos maiores, o hardware poderia ser forçado a dispendiar mais ciclos para o processamento.

Como exemplo, considere um canal de transmissão de 10 Mbps. Neste canal, o t_{bit} é de $0,1\mu s$. Assim, a Equação (6.2) se reduz a $(33,6\mu s/10)^{-1}$, resultando em uma frequência mínima de 0,2976 MHz. A mesma análise é feita para canais com velocidades maiores e os resultados são apresentados na Tabela 6.2.

A Tabela 6.2 sugere que o HARP opere à mesma velocidade do canal, se equiparmos a geração e a transmissão de um bit. Em termos práticos, estes valores de frequência podem ser menores, pois foi desconsiderado o tempo que o sistema leva para encapsular mensagens. A partir deste ponto, cabe analisar o hardware prototipado para estimar em que patamar o HARP se encontra.

Tabela 6.2: Frequência mínima de operação

Velocidade do Canal	Tempo de bit	Frequência mínima
10 Mbps	0.1 us	0.2976 MHz
100 Mbps	0.01 us	2.976 MHz
1 Gbps	1 ns	29.76 MHz
10 Gbps	0.1 ns	297.6 MHz

6.3 Prototipagem em FPGA

Para esta discussão, cabe lembrar que o hardware disponível para prototipagem é o kit de desenvolvimentno DE2, equipado com um FPGA de baixo custo Cyclone II. O relatório de síntese foi gerado pelo software Quartus II. A Tabela 6.3 apresenta os recursos utilizados pelo módulo HARP em sua prototipagem no *Cyclone II*. Para efeitos de comparação, o mesmo código foi sintetizado para um FPGA mais poderoso, o Altera Stratix EP1S80F1508I7.

Tabela 6.3: Utilização do FPGA

	FPGA Cyclone II EP2C35F672C6	FPGA Stratix IV EP1S80F1508I7
Logic utilization	3063 LEs (9.2%)	1911 ALUTs (1%)
LC combinational	2556	1911
LC Registers	1697	1700
Block Memory Bits	1536 (1%)	1536 (< 1%)
DSP Elements	4 (9-bit)	4 (18-bit)
PLL	0	0
F_{max} (MHz)	97.68	128.45

Prototipado no FPGA Altera Cyclone II EP2C35F672C6, o HARP ocupou uma área pequena dos recursos disponíveis, alcançando a marca do 9,2% de uso dos 33216 elementos lógicos (LE) disponíveis. Enquanto isso, segundo os dados da síntese, no *Altera Stratix EP1S80F1508I7* a área total utilizada atingiu apenas 1% das 182400 *Adaptative LUTs* (ALUTs) disponíveis no FPGA.

A respeito do uso de memória, o total de blocos não foi alterado na migração entre FPGAs. Os módulos *Fifo* e *Active Address Table*, da Figura 5.1, são responsáveis pelo uso de memória. A primeira ocupa um total de 8x128 bits, enquanto a segunda requer 16x32 bits de memória interna *Block Ram* (BRAM). Outras memórias podem ser utilizadas, para armazenar mensagem, por exemplo. Isso também pode ser feito utilizando-se SRAM externas. A máxima frequência que mantém a integridade do circuito é 98,68 MHz para o Cyclone II e 128,45 MHz para o Stratix IV.

O SoPC da Figura 3.4 foi prototipado no FPGA Cyclone II. Ele ocupa uma área total de 9892 LEs, sendo 30% do total disponível no FPGA. A frequência do processador é de

100 MHz e as dos componentes integrados variavam entre 25 MHz e 50 MHz, dependendo do chip a que se conectam.

É importante lembrar que o HARP é independente do restante do sistema e devido ao baixo recurso demandado, torna-se viável seu acoplamento a outros módulos quando da construção de um núcleo de roteamento e alta disponibilidade. Observa-se ainda pela Tabela 6.2, que o HARP alcançou uma frequência máxima que atende às exigências de canais das redes 1 Gbps.

Este resultado é satisfatório neste patamar da pesquisa, pois embora não haja exigências quanto ao desempenho, canais de transmissão 1 Gbps são ainda são utilizados em determinadas categorias de redes. Como se pode notar pela Tabela 6.2, o HARP atende os requisitos de frequência para 1 Gbps com mais que o triplo da exigência, classificando-o como satisfatório à operações mais rápidas que 3 Gbps. Estes valores podem ainda ser melhorados depois de uma otimização do código fonte em VHDL.

Por se tratar de uma pesquisa com aplicação prática, os resultados foram analisados tanto com simulações parciais no Modelsim, bem como diretamente no hardware. Assim, o esquema proposto pela figura 3.3 está em destaque na Figura 6.5, que mostra o ambiente de experimentação real construído.

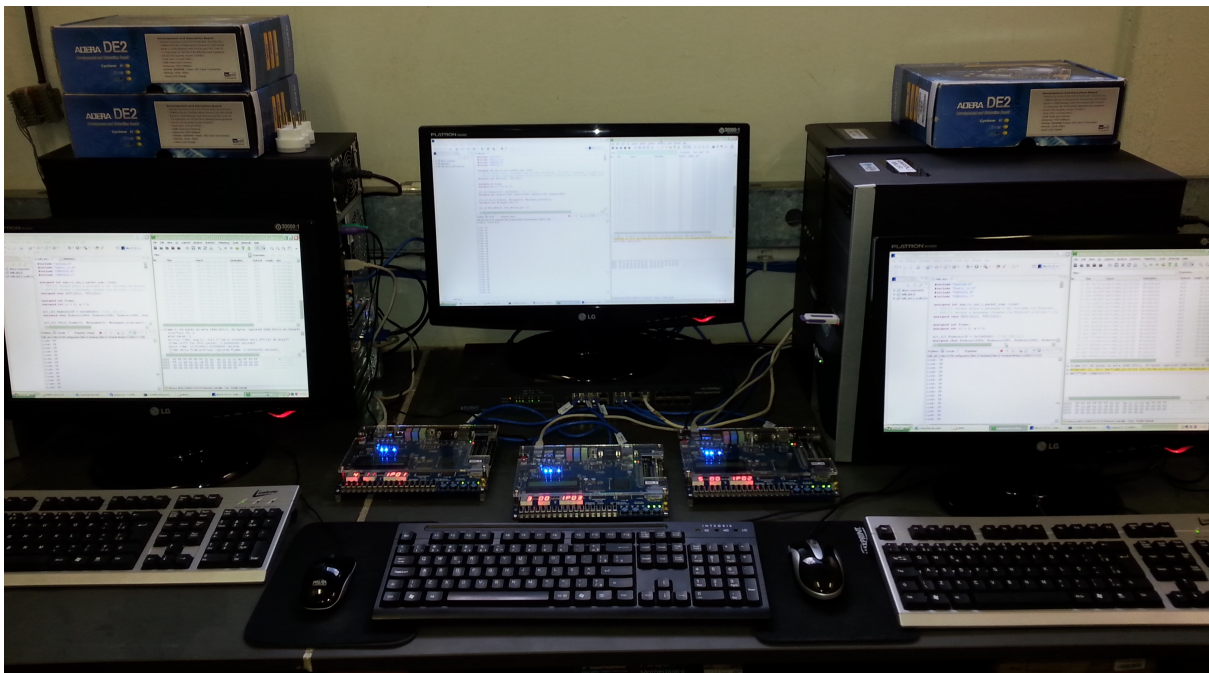


Figura 6.5: Ambiente real de experimentação

Os monitores exibem o processamento do software embarcado através do NIOS II *Viewer* e a captura de pacotes nas interfaces de rede através do software *Wireshark*. Os kits de prototipagem são equipados também com leds, chaves de comutação e displays de sete segmentos que ajudam na depuração do sistema e acompanhamento em tempo

real dos dados no hardware. O comutador que está atrás dos três kits de prototipagem é que permite que todos os nós da rede (computadores de placas) sejam interconectados via Ethernet.

Considerações Finais

Esta dissertação apresenta o *High Availability Router Protocol* (HARP) (Oliveira *et al.*, 2013b) (Oliveira *et al.*, 2013a) e como ele corrige os problemas algorítmicos que atacam os protocolos de alta disponibilidade existentes. São apresentadas a sua especificação formal e sua arquitetura, bem como o resultado de uma simulação no contexto de sistemas paralelos.

Para o desenvolvimento do HARP, partiu-se de uma proposta não implementável, modelada em redes de Petri, que considera um ambiente com a ocorrência de perda de mensagens e corrupção de dados. A partir de então, procedeu-se com a especificação dos cinco elementos do protocolo, sua implementação e prototipagem para prova de conceito. Desta forma, foi construído um sistema de validação para demonstrar o funcionamento do HARP integrando-o a versões *LightWeight* dos protocolos IP e ARP. Concluiu-se, em virtude das trocas de mensagens em cada serviço, que operacionalmente o HARP funciona a contento e evita a indisponibilidade.

As condições de Acéfalo e Cérebro Partido foram apontadas como as causas da indisponibilidade da rede, devido, principalmente, às falhas das interfaces de comunicação. Este problemas são corrigidos pelo HARP, uma vez que sua MEF possui estados e transições suficientes para cobrir estas falhas.

A especificação de serviços introduz a possibilidade de transferência espontânea de função entre um mestre e um escravo através do serviço *Grant Master*. Para o caso de eleição, o serviço *Check Brain* exige que os nós se comuniquem entre si em um processo consensual, a fim de minimizar a ocorrência das condições que atacam o protocolo.

Uma vez que os elementos foram especificados, uma descrição formal em TLA+ permitiu simular uma execução de várias instâncias do HARP, demonstrando sua corretude.

Verificou-se que o protocolo chega a um consenso e que o autômato é estável. Esta verificação ratificou que a MEF mantém as boas propriedades por ser (i) bem formada e (ii) consistente. Para a primeira propriedade, o HARP é limitado, autoestabilizável e autoadaptativo. Para a segunda, o autômato é livre de *Deadlocks*, livre de *Livelocks* e livre de terminações inadequadas.

Do ponto de vista arquitetural, o HARP está implementado em FPGA com uma arquitetura modularizada, compatível com IPv4 e com a capacidade de responder mensagens utilizando, em média, 10 ciclos de clock. A implementação em FPGA permitiu executar atualizações sem custos adicionais, devido à sua reconfigurabilidade. O uso de prototipagem rápida tornou possível a especificação do protocolo simultaneamente ao projeto e implementação da sua arquitetura. Além disso, ter uma arquitetura que seja reconfigurável permite que as outras versões do HARP sejam habilitadas sem a necessidade da fabricação de ASICs.

As ferramentas de desenvolvimento foram cruciais para o protótipo. O Quartus II fornece o aparato necessário para o desenvolvimento com FPGA e provê relatórios de utilização do espaço e tempo de operação dos circuitos. O Modelsim permite realizar simulações de cada módulo da arquitetura, garantindo as respostas esperadas antes de acoplá-los ao sistema. A adição de módulos à biblioteca do *SoPC Builder* permitiu a construção do SoPC e a integração com a rede local.

O módulo de hardware do HARP ocupa uma área de aproximadamente 10% elementos lógicos em um FPGA de baixo custo Cyclone II e estas proporções diminuem à medida que os FPGAs possuem mais recursos. O módulo pode operar a até 98,68 MHz de frequência de forma a manter a integridade dos dados. Este valor demonstra que mesmo sem exigências quanto ao desempenho, no estágio atual o HARP pode operar em redes Gigabit, mantendo o tempo de convergência do protocolo na ordem de milissegundos. Este valor é superestimado pois foram ignoradas todas as tarefas entre a geração da mensagem e sua transmissão pelo enlace.

O tempo para percepção de uma falha em um protocolo desta natureza tem no VRRP seu menor valor, que é de 3 segundos. O HARP deixa este valor a ser configurando observando-se o tempo médio de transmissão de mensagens nos cenários específicos. Entretanto, o HARP não está apto a funcionar em ambientes com preservação de estado, uma vez que não trata do processo de transferência de estado. Ademais, ele não faz nenhuma consideração sobre balanceamento de carga. Uma análise do HARP em um ambiente com alto fluxo de mensagens também se faz necessário, para uma conclusão fidedigna dos limites de desempenho.

O capítulo anterior mostrou que a troca de mensagens observada no sistema de validação ocorre de forma previsível, permitindo concluir que o HARP comporta-se conforme sua especificação. Não é comum, por exemplo, que se tenha mais de dez equipamentos em um roteador virtual, por isso, o tempo de convergência do protocolo se mantém

$\ll 1$ segundo e não chega a um custo algorítmico exponencial.

É importante salientar que, devido à limitação laboratorial de não haver quantidade suficiente de placas de prototipagem com múltiplas interfaces Ethernet, um monitoramento em uma rede complexa não pode ser feito com este experimento, uma vez que cada placa possui apenas uma interface de rede, impossibilitando a redundância de canais. Além disso, submeter o HARP à avaliação em um ambiente com alto fluxo de mensagens mostraria suas limitações quanto à capacidade de processamento. Espera-se que isso seja feito posteriormente, quando houver a acoplação do HARP aos outros módulos dos protocolos TCP/IP e no laboratório de uma empresa de telecomunicações parceira.

Assim, esta dissertação traz resultados sobre um novo protocolo de rede. Tais resultados foram atingidos depois de um processo de testes e prototipagem em hardware reconfigurável. A partir de agora, pode-se vislumbrar o desenvolvimento de equipamentos de rede para alta disponibilidade que atendam à exigências de novos modelos e arquiteturas. Neste contexto, há alguns pontos classificados como potenciais trabalhos futuros que podem aprofundar esta pesquisa, a saber:

- Acoplar o HARP aos outros protocolos da arquitetura Internet (possivelmente até a camada de transporte) e submetê-lo a um ambiente com alto fluxo de mensagens para identificar as limitações de desempenho. Os módulos acoplados podem implementar também outras funções além de processamento de protocolos, tais como criptografia de dados.
- Realizar um estudo sobre a utilização do HARP em ambiente *stateful* e incluir na especificação considerações que tratem da transferência de estado.
- Incluir adendos na especificação do HARP que trate do balanceamento de carga, através de classificação por endereços, prioridade ou parâmetros semelhantes.
- Introduzir negociações de autenticação e segurança entre os membros do grupo de alta disponibilidade, bem como implementar módulos criptográficos para dados HA.
- Modificar o elemento formato da especificação do HARP e implementá-lo em suas versões compatíveis com IPv6 e abordagens de Internet do Futuro (e.g., *Clean Slate*).
- Analisar a operação do HARP em infraestruturas típicas de alta disponibilidade, de forma que se possa estudar a probabilidade de perdas e erros de mensagens nestes ambientes e implementar detectores de falhas para concluir estatisticamente a intensidade com que isso afeta a operação do HARP.
- Considerar os aspectos levantados no item anterior e demonstrar propriedades de Liveness na descrição formal em TLA⁺.

Referências

- Abadi, M. e Lamport, L. (1991). The existence of refinement mappings. *Theoretical Computer Science Journal*, 82(2):253 – 284.
- Aberdeen, G. (2012). Datacenter Downtime: Hou Much It Really Cost? URL: <http://www.stratus.com/~media/Stratus/Files/Library/AnalystReports/AberdeenDatacenterDowntimeCost.pdf>. Acesso em 12/06/2013.
- Altera (2006). DE2 Development and Education Board User Manual. URL: <http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html>. Acesso em 17/05/2013.
- Altera (2013a). Avalon Bus Specification. URL: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf. Acesso em 17/05/2013.
- Altera (2013b). NIOS II Processor. URL: <http://www.altera.com/devices/processor/nios2/ni2-index.html>. Acesso em 17/05/2013.
- Altera (2013c). Nios II/f Core: Fast for Performance-Critical Applications. URL: <http://www.altera.com/devices/processor/nios2/cores/fast/ni2-fast-core.html>. Acesso em 12/06/2013.
- Atkinson, R. e Andrews, U. S. (2012). Identifier-Locator Network Protocol (ILNP) Architectural Description. URL <http://tools.ietf.org/html/rfc6740>.
- Backus, J. (1978). Can Programming be Liberated from the Von Neumann Style?: A Functional Style and its Algebra of Programs. *Transactions on Communications of the ACM*, 21(8):613–641.
- Bertier, M., Marin, O., e Sens, P. (2002). Implementation and Performance Evaluation of an Adaptable Failure Detector. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks. DSN '02*, páginas 354–363, Washington, DC, USA. IEEE Computer Society.
- Bhagat, N. H. (2011). Virtual Router Redundancy Protocol-A Best Open Standard Protocol in Maintaining Redundancy. *IJCA Proceedings on the International Conference on Web Services Computing (ICWSC)*, ICWSC(1):67–70.
- Brown, S. e Rose, J. (1996). FPGA and CPLD architectures: a tutorial. *Transactions on IEEE Design & Test of Computers*, 13(2):42 –57.

- Camargos, L. J. (2009). *Protocolos Multicoordenados de Acordo e o Serviço de Log*. Tese de Doutorado, Programa de Pós Graduação em Ciência da Computação. Universidade Estadual de Campinas.
- Chandra, T. D., Hadzilacos, V., e Toueg, S. (1996). The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722.
- Chen, W., Toueg, S., e Aguilera, M. K. (2002). On the Quality of Service of Failure Detectors. *IEEE Transactions on Computers*, 51(5):561–580.
- Cisco (2013a). First Hop Redundancy Protocol (FHRP). URL: http://www.cisco.com/en/US/products/ps6644/products_ios_protocol_option_home.html. Acesso em 17/05/2013.
- Cisco (2013b). Gateway Load Balancing Protocol (GLBP). URL: http://www.cisco.com/en/US/docs/ios/12_2s/feature/guide/fs_glb2.html. Acesso em 17/05/2013.
- Cisco (2013c). High Availability. URL: http://www.cisco.com/en/US/products/ps6550/products_ios_technology_home.html. Acesso em 12/06/2013.
- Cisco (2013d). Hot Standby Router Protocol (HSRP). URL: http://www.cisco.com/en/US/tech/tk648/tk362/tk321/tsd_technology_support_sub-protocol_home.html. Acesso 17/05/2013.
- Compton, K. e Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys Journal (CSUR)*, 34(2):171–210.
- Cormen, T. H., Stein, C., Rivest, R. L., e Leiserson, C. E. (2009). *Introduction to Algorithms*. Mit Press.
- Davicom (2005a). DM9000A Application Notes. URL: <http://www.jedec.org/standards-documents/docs/jesd-6801>. Acesso em 12/06/2013.
- Davicom (2005b). DM9000A Datasheet. URL: <http://www.davicom.com.tw/page1.aspx?no=143762>. Acesso em 12/06/2013.
- Deschamps, J.-P., Bioul, G. J. A., e Sutter, G. D. (2006). *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. Wiley.
- Dijkstra, E. W. (1965). Solution of a problem in concurrent programming control. *Transactions on Communications of the ACM*, 8(9):569–.
- Enderton, H. B. (1977). *Elements of Set Theory*. Elsevier, London, UK.
- Fei-yu, L., Xi-xiang, J., Yu-Hui, G., Jian-chuan, Z., Wei-ming, Q., Lan, J., Yan-Yu, W., e Yun-hai, M. (2010). System on Programmable Chip Development System. In *Proceedings of the Second International Workshop on Education Technology and Computer Science (ETCS), 2010*, volume 1, páginas 471–474.
- Filho, E., Hashimoto, G., e Rosa, P. (2008). A High Availability Firewall Model Based on SCTP Protocol. In *IEEE Proceedings of the Systems and Networks Communications, 2008. ICSNC '08*, páginas 202–207.

- Graphics, M. (2013). Modelsim. URL: <http://www.mentor.com/products/fpga/simulation/modelsim>. Acesso em 12/06/2013.
- Gray, J. e Siewiorek, D. (1991). High-availability Computer Systems. *Computer Journal*, 24(9):39–48.
- Hamerski, J. C., Reckziegel, E., e Kastensmidt, F. (2007). Evaluating memory sharing data size and TCP connections in the performance of a reconfigurable hardware-based architecture for TCP/IP stack. In *Proceedings of the IFIP International Conference on Very Large Scale Integration, 2007. VLSI - SoC 2007*, páginas 212–217.
- Hashimoto, G., Filho, E., Pereira, J., e Rosa, P. (2010). High Availability: A Long-Term Feature in Network Elements. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC), 2010*, páginas 201 –206.
- Hashimoto, G. T. (2009). Uma proposta de extensao para um protocolo para arquiteturas de alta disponibilidade. Dissertação de Mestrado, PPGCC - Universidade Federal de Uberlândia.
- Hauck, S. e DeHon, A. (2010). *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Systems on Silicon. Elsevier Science.
- Hawkins, M. e Piedad, F. (2000). *High Availability: Design, Techniques and Processes*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition.
- Holzmann, G. J. (1991). *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Hopcroft, J. E., Motwani, R., e Ullman, J. D. (2000). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition.
- IEEE (2001). IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture. *IEEE Standard 802-2001*.
- IETF (1991). ICMP Router Discovery Messages. RFC 1256. URL <http://www.ietf.org/rfc/rfc1256.txt>.
- IETF (2004). Virtual Router Redundancy Protocol (VRRP). RFC 3768. URL: <http://www.ietf.org/rfc/rfc3768.txt>.
- IETF (2008). Unified Layer 2 (L2) Abstractions for Layer 3 (L3)-Driven Fast Handover. RFC 5184. URL: <http://tools.ietf.org/html/rfc5184>.
- IETF (2013). The Internet Engineering Task Force. URL: <http://www.ietf.org/>. Acesso em 12/06/2013.
- ITU (2013). Internet Users. URL: <http://www.itu.int/ITU-D/ict/statistics/index.html>. Acesso em 17/05/2013.
- Jedhe, G., Ramamoorthy, A., e Varghese, K. (2008). A Scalable High Throughput Firewall in FPGA. In *Proceedings of the 16th International Symposium on Field-Programmable Custom Computing Machines, 2008. FCCM '08*, páginas 43 –52.

- Johnson, B. W., editor (1988). *Design & analysis of fault tolerant digital systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Juniper (2013). Understanding High Availability Features on Juniper Network Routers. URL: http://www.juniper.net/techpubs/en_US/junos13.1/topics/concept/high-availability-features-in-junos-introducing.html. Acesso em 12/06/2013.
- Kachris, C. e Vassiliadis, S. (2006). Design of a web switch in a reconfigurable platform. In *Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications systems, 2006. ANCS 2006.*, páginas 31–40.
- Kurose, J. F. e Ross, K. W. (2006). *Redes de computadores e a Internet: uma abordagem top-down*. Pearson Addison Wesley.
- Lakatos, E. M. e Marconi, M. A. (2003). *Fundamentos de Metodologia Científica*. Atlas, São Paulo, SP, Brasil, 5 edition.
- Lamport, L. (1977). Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143.
- Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923.
- Lamport, L. (2000). A Summary of TLA⁺. URL: <http://research.microsoft.com/en-us/um/people/lamport/tla/summary.pdf>. Acesso em 12/06/2013.
- Lamport, L. (2003). *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley, USA.
- Lamport, L. e Yu, Y. (2011). TLC – The TLA⁺ Model Checker. URL: <http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html>. Acesso em 12/06/2013.
- Lopes Filho, E. (2008). Arquitetura de Alta Disponibilidade para Firewall e IPS baseada em SCTP. Dissertação de Mestrado, PPGCC - Universidade Federal de Uberlândia.
- Lynch, N. A. (1996). *Distributed Algorithms*. Data Management Series. Morgan Kaufmann Publishers.
- Matsuda, H. (2012). L2 Switch Feature for Virtual Router Redundancy Protocol Fast Convergence. *International Journal of Computer Applications*, 55(11).
- Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079.
- MEHAR (2013). Mondial Entities Horizontally Addressed by Requirements. URL: <http://www.mehar.facom.ufu.br/>. Acesso em 17/05/2013.
- Merz, S. (2003). On the Logic of TLA⁺. *Computers and Informatics Journal*, 22:351–379.
- Oliveira, R., Mesquita, D., e Rosa, P. (2013a). HARP: A Split Brain free protocol implemented in FPGA. In *Proceedings of the Ninth Advanced International Conference on Telecommunications, AICT 2013*, volume 9, páginas 197–203.

- Oliveira, R., Mesquita, D., e Rosa, P. (2013b). Um novo protocolo de alta disponibilidade implementado em FPGA. In *Proceedings of the 31 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. SBRC 2013*, volume 31, páginas 105–118.
- OpenBSD (2013). PF: Firewall Redundancy with CARP and pfsync. URL: <http://www.openbsd.org/faq/pf/carp.html>. Acesso em 17/05/2013.
- Oracle (2013). High Availability. URL: http://docs.oracle.com/cd/E27363_01/doc.121/e25143/high_availability.htm. Acesso em 12/06/2013.
- Pereira, J., Santos, E., Pereira, F., Rosa, P., e Kofuji, S. (2010). Layers Optimization Proposal in a Post-IP Network. *International Journal on Advances in Networks and Services*, 3(3 and 4):361–369.
- Pereira, J. d. S., de Oliveira Silva, F., Filho, E., Kofuji, S., e Rosa, P. (2011). Title Model Ontology for Future Internet Networks. In J. Domingue, A. Galis, A. Gavras, T. Zahariadis, D. Lambert, F. Cleary, P. Daras, S. Krco, H. Möller, M.-S. Li, H. Schaffers, V. Lotz, F. Alvarez, B. Stiller, S. Karnouskos, S. Avessta, e M. Nilsson, editores, *The Future Internet*, volume 6656 of *Lecture Notes in Computer Science*, páginas 103–114. Springer Berlin / Heidelberg. 10.1007/978-3-642-20898-0_8.
- Pereira Júnior, J. E. (2010). Especificação de serviço e suposições sobre o ambiente para um protocolo de alta disponibilidade. Dissertação de Mestrado, PPGCC - Universidade Federal de Uberlândia.
- Petri, C. (1962). *Kommunikation mit Automaten*. Schriften des Rheinisch-Westfälischen Instituts für Instrumentelle Mathematik an der Universität Bonn. Rhein.-Westfäl. Inst. f. Instrumentelle Mathematik an der Univ. Bonn, Germany.
- Pnueli, A. (1979). The Temporal Semantics of Concurrent Programs. In *Proceedings of the International Symposium on Semantics of Concurrent Computation*, páginas 1–20, London, UK, UK. Springer-Verlag.
- Radatz, J. (1997). *The IEEE Standard Dictionary of Electrical and Electronics Terms*. IEEE Standards Office, New York, NY, USA, 6th edition.
- Ravindran, K., Satish, N., Jin, Y., e Keutzer, K. (2005). An FPGA-based soft multiprocessor system for IPv4 packet forwarding. In *Proceedings of the International Conference on Field Programmable Logic and Applications, 2005.*, páginas 487 – 492.
- Santos, E., Pereira, F., Pereira, J., Theodoro, L., Rosa, P., e Kofuji, S. (2011). Meeting Services and Networks in the Future Internet. In J. Domingue, A. Galis, A. Gavras, T. Zahariadis, D. Lambert, F. Cleary, P. Daras, S. Krco, H. Möller, M.-S. Li, H. Schaffers, V. Lotz, F. Alvarez, B. Stiller, S. Karnouskos, S. Avessta, e M. Nilsson, editores, *The Future Internet*, volume 6656 of *Lecture Notes in Computer Science*, páginas 339–350. Springer Berlin / Heidelberg. 10.1007/978-3-642-20898-0_24.
- Sharp, R. (2008). *Principles of Protocol Design*. Springer Publishing Company, Incorporated, Germany, 1 edition.
- Soares, L., Souza Filho, G., e Colcher, S. (1995). *Redes de computadores: das LANs, MANs, e WANs às redes ATM*. Campus.

- Straka, M. e Kotasek, Z. (2009). High Availability Fault Tolerant Architectures Implemented into FPGAs. In *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009. DSD '09.*, páginas 108 –115.
- Sun, H., Han, J. J., e Levendel, H. (2003a). Availability requirement for a fault-management server in high-availability communication systems. *IEEE Transactions on Reliability*, 52(2):238–244.
- Sun, H., Han, J., e Levendel, H. (2003b). Availability requirement for a fault-management server in high-availability communication systems. *IEEE Transactions on Reliability*, 52(2):238–244.
- Tanenbaum, A. (2002). *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition.
- Tioh, J.-W., Bahuguna, R., Vanderhorn, N., Mina, M., Weber, R., e Somani, A. (2008). Reprogrammable high-speed platform : Bridging the gap between research, education and engineering. In *Proceedings of the IEEE International Conference on Electro/Information Technology, 2008. EIT 2008.*, páginas 145 –147.
- Trimberger, S. (1994). *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers.
- Wazlawick, R. S. (2009). *Metodologia de Pesquisa para Ciência da Computação*. Elsevier, Rio de Janeiro, RJ, Brasil, 2 edition.
- Wireshark (2013). Wireshark. URL: <http://www.wireshark.org/>. Acesso em 17/05/2013.
- Woodberg, B., Madwachar, M. K., Swarm, M., Wyler, N. R., Albers, M., e Bannell, R. (2007). *Configuring Juniper Networks NetScreen & SSG Firewalls*. Syngress Publishing.

Mensagens HARP

As tabelas nesta seção são relacionadas ao conteúdo das mensagens HARP. A Tabela A.2 apresenta o conteúdo especificado para formatar as mensagens. Alguns deles são valores padrão, enquanto outros são atribuídos na implementação, conforme explicado no Capítulo 4. Para uma melhor visualização, a Tabela A.2 utiliza siglas no lugar dos nomes das mensagens. São as mesmas siglas utilizadas ao longo do texto e, por uma questão didática, elas são relacionadas na Tabela A.1.

Tabela A.1: Nomes das mensagens e suas siglas

<i>NOME DA MENSAGEM</i>	<i>SIGLA</i>
<i>Keep Alive Request</i>	<i>KA_REQ</i>
<i>Grant Master Request</i>	<i>GM_REQ</i>
<i>Grant Master Response</i>	<i>GM_RESP</i>
<i>Grant Master Failed Request</i>	<i>GMFAIL_REQ</i>
<i>Grant Master Ready Request</i>	<i>GMRDY_REQ</i>
<i>Inform Node Request</i>	<i>INF_REQ</i>
<i>Inform Node Response</i>	<i>INF_RESP</i>
<i>Remove Node Request</i>	<i>REM_REQ</i>
<i>Remove Node Response (+)</i>	<i>REM_RESP</i>
<i>Check Brain Request</i>	<i>CB_REQ</i>
<i>Check Brain Response (+)</i>	<i>CB_RESP⁺</i>
<i>Check Brain Response (-)</i>	<i>CB_RESP⁻</i>
<i>Active Slave Request</i>	<i>ACTS_REQ</i>
<i>Active Slave Response</i>	<i>ACTS_RESP</i>
<i>Configure IP</i>	<i>Config_IP</i>

Tabela A.2: Formatação e conteúdo das mensagens HARP

	<i>DESTINATION ADDRESS</i>	<i>SOURCE ADDRESS</i>	<i>TYPE</i>	<i>VERSION</i>	<i>MSG TYPE</i>	<i>PRIORITY</i>	<i>COUNT IP ADDRESS</i>	<i>DATA LENGTH</i>	<i>CHECKSUM</i>
<i>KA_REQ</i>	0xFFFF	<i>IP NE</i>	0xC8	0x01	0x01	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>GM_REQ</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x02	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>GM_RESP</i>	<i>unicast (master)</i>	<i>IP NE</i>	0xC8	0x01	0x03	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>GMRDY_REQ</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x04	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>GMFLD_REQ</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x05	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>INF_REQ</i>	0xFFFF	<i>IP NE</i>	0xC8	0x01	0x06	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>INF_RESP</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x07	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>REM_REQ</i>	<i>unicast (master)</i>	<i>IP NE</i>	0xC8	0x01	0x08	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>REM_RESP</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x09	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>CB_REQ</i>	0xFFFF	<i>IP NE</i>	0xC8	0x01	0x0A	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>CB_RESP(+)</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x0B	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>CB_RESP(-)</i>	<i>unicast</i>	<i>IP NE</i>	0xC8	0x01	0x0C	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>ACTS_REQ</i>	0xFFFF	<i>IP NE</i>	0xC8	0x01	0x0D	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>ACTS_RESP</i>	<i>unicast (master)</i>	<i>IP NE</i>	0xC8	0x01	0x0E	<i>nonstandard</i>	0x00	0x00	<i>calculated</i>
<i>Config_IP</i>	0x00	<i>IP NE</i>	0xC8	0x01	0x10	0x00	0x00	0x00	<i>calculated</i>

*Spec HARP*MODULE *HARP*EXTENDS *Naturals, Integers, TLC, FiniteSets*

Processes

VARIABLES *state*,

Am I the master, a slave, or idle? Am I receiving mastership?

Electing a leader?

joined

Am I part of the system?

Communication channels

VARIABLE *msgs*

Global state

VARIABLE *failed* $vars \triangleq \langle state, joined, msgs, failed \rangle$ CONSTANT *PID* $NoPID \triangleq \text{CHOOSE } p : p \notin PID$ $View \triangleq \text{SUBSET } PID$ $RandomView \triangleq RandomElement(\text{SUBSET } joined) \text{ Perfect: } joined$ $JoinedCardEst \triangleq RandomElement(1 \dots Cardinality(PID)) \text{ Perfect: } Cardinality(joined)$ $PStates \triangleq \{ "IDLE", "MASTER", "SLAVE", \\ "MASTER_GM_CONFIRM", "SLAVE_GM_ACCEPT", \\ "SLAVE_EL_CONFIRM", "SLAVE_EL_ELECTION" \}$ $Msgs \triangleq [type : \{ "GM_REQ" \}, \quad from : PID, to : PID] \cup \\ [type : \{ "GM_RESP" \}, \quad from : PID, to : PID] \cup$

$$\begin{aligned}
& [type : \{ "GM_FAIL_REQ" \}, from : PID, to : PID] \cup \\
& [type : \{ "GM_RDY_REQ" \}, from : PID, to : PID] \cup \\
& [type : \{ "INF_REQ" \}, from : PID, to : PID] \cup \\
& [type : \{ "INF_RESP" \}, from : PID, to : PID] \cup \\
& [type : \{ "REM_REQ" \}, from : PID, to : PID] \cup \\
& [type : \{ "REM_RESP" \}, from : PID, to : PID] \cup \\
& [type : \{ "CB_REQ" \}, from : PID] \cup \\
& [type : \{ "CB_RESP+" \}, from : PID, to : PID] \cup \\
& [type : \{ "CB_RESP-" \}, from : PID, to : PID]
\end{aligned}$$

$$\begin{aligned}
TypeInvariant & \triangleq \wedge msgs \in \text{SUBSET } Msgs \\
& \wedge state \in [PID \rightarrow PStates] \\
& \wedge failed \in \text{SUBSET } PID \\
& \wedge joined \in \text{SUBSET } PID
\end{aligned}$$

$$\begin{aligned}
Init & \triangleq \wedge msgs = \{ \} \\
& \wedge failed = \{ \} \\
& \wedge state = [p \in PID \mapsto "IDLE"] \\
& \wedge joined = \{ \}
\end{aligned}$$

General Purpose Functions

$$\begin{aligned}
IsMaster(p) & \triangleq \wedge state[p] \in \{ "MASTER", "MASTER_GM_CONFIRM" \} \\
& \wedge p \notin failed \\
& \wedge p \in joined \\
IsIdle(p) & \triangleq \wedge state[p] \in \{ "IDLE" \} \\
& \wedge p \notin failed \\
& \wedge p \in joined \\
IsSlave(p) & \triangleq \wedge state[p] \in \{ "SLAVE", \\
& \quad "SLAVE_GM_ACCEPT", "SLAVE_EL_CONFIRM", "SLAVE_EL_ELECTION" \} \\
& \wedge p \notin failed \\
& \wedge p \in joined
\end{aligned}$$

MessageHandlingActions

$$\begin{aligned}
Send(m) & \triangleq msgs' = msgs \cup \{ m \} \\
Receive(m) & \triangleq \wedge m \in msgs \\
& \wedge msgs' = (msgs \setminus \{ m \}) \\
ReceiveNSend(n, m) & \triangleq msgs' = (msgs \setminus \{ n \}) \cup \{ m \}
\end{aligned}$$

FailRecoverActions

$$\begin{aligned}
MessageLoss & \triangleq \wedge \text{FALSE} \quad \text{Just for testing.} \\
& \wedge \exists m \in msgs : msgs' = msgs \setminus \{ m \} \\
& \wedge \text{UNCHANGED } \langle failed, state, joined \rangle
\end{aligned}$$

$$\begin{aligned}
Fail &\triangleq \wedge \exists p \in PID : \wedge p \notin failed \\
&\quad \wedge failed' = failed \cup \{p\} \\
&\quad \wedge joined' = joined \setminus \{p\} \\
&\quad \wedge \text{UNCHANGED } \langle msgs, state \rangle \\
Recover &\triangleq \wedge \exists f \in PID : \wedge f \in failed \\
&\quad \wedge failed' = failed \setminus \{f\} \\
&\quad \wedge state' = [state \text{ EXCEPT } ![f] = \text{"IDLE"}] \\
&\quad \wedge \text{UNCHANGED } \langle msgs, joined \rangle
\end{aligned}$$

JoinActions

$$\begin{aligned}
MayJoin(p) &\triangleq \wedge state[p] = \text{"IDLE"} \\
&\quad \wedge p \notin failed \\
&\quad \wedge p \notin joined
\end{aligned}$$

This action is the equivalent of a manual intervention forcibly bringing a node up to act as the master

$$\begin{aligned}
Bootstrap &\triangleq \wedge \forall p \in PID : state[p] = \text{"IDLE"} \vee p \in failed \\
&\quad \wedge \exists p \in PID : \wedge MayJoin(p) \\
&\quad \quad \wedge state' = [state \text{ EXCEPT } ![p] = \text{"MASTER"}] \\
&\quad \quad \wedge joined' = joined \cup \{p\} \\
&\quad \wedge \text{UNCHANGED } \langle msgs, failed \rangle
\end{aligned}$$

When joining, the node is informed about whom to contact (the master)

$$\begin{aligned}
JoinReq &\triangleq \wedge \exists p, q \in PID : \wedge MayJoin(p) \\
&\quad \quad \wedge IsMaster(q) \\
&\quad \quad \wedge Send([type \mapsto \text{"INF_REQ"}, from \mapsto p, to \mapsto q]) \\
&\quad \wedge \text{UNCHANGED } \langle failed, state, joined \rangle \\
JoinResp &\triangleq \wedge \exists p \in PID, m \in msgs : \wedge IsMaster(p) \\
&\quad \quad \wedge m.type = \text{"INF_REQ"} \wedge m.to = p \\
&\quad \quad \wedge ReceiveNSend(m, [type \mapsto \text{"INF_RESP"}, from \mapsto p, to \mapsto m.from]) \\
&\quad \wedge \text{UNCHANGED } \langle failed, state, joined \rangle
\end{aligned}$$

$$\begin{aligned}
Join &\triangleq \wedge \exists p \in PID, m \in msgs : \wedge MayJoin(p) \\
&\quad \quad \wedge m.type = \text{"INF_RESP"} \\
&\quad \quad \wedge m.to = p \\
&\quad \quad \wedge Receive(m) \\
&\quad \quad \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE"}] \\
&\quad \quad \wedge joined' = joined \cup \{p\} \\
&\quad \wedge \text{UNCHANGED } \langle failed \rangle
\end{aligned}$$

LeaveActions

$$\begin{aligned}
MayLeave(p) &\triangleq \wedge state[p] = \text{"SLAVE"} \\
&\quad \wedge p \notin failed \\
&\quad \wedge p \in joined
\end{aligned}$$

$$\begin{aligned}
\text{LeaveReq} \triangleq & \wedge \exists p, q \in \text{PID} : \wedge \text{MayLeave}(p) \\
& \wedge \text{IsMaster}(q) \\
& \wedge \neg \exists m \in \text{msgs} : \wedge m.\text{type} = \text{"REM_REQ"} \\
& \wedge m.\text{from} = p \\
& \wedge \text{Send}([type \mapsto \text{"REM_REQ"}, from \mapsto p, to \mapsto q]) \\
& \wedge \text{UNCHANGED } \langle \text{failed}, \text{state}, \text{joined} \rangle
\end{aligned}$$

$$\begin{aligned}
\text{LeaveResp} \triangleq & \wedge \exists p \in \text{PID}, m \in \text{msgs} : \wedge \text{IsMaster}(p) \\
& \wedge m.\text{type} = \text{"REM_REQ"} \wedge m.\text{to} = p \\
& \wedge \text{ReceiveNSend}(m, [type \mapsto \text{"REM_RESP"}, from \mapsto p, to \mapsto m.\text{from}]) \\
& \wedge \text{UNCHANGED } \langle \text{failed}, \text{state}, \text{joined} \rangle
\end{aligned}$$

$$\begin{aligned}
\text{Leave} \triangleq & \wedge \exists p \in \text{PID}, m \in \text{msgs} : \wedge \text{MayLeave}(p) \\
& \wedge m.\text{type} = \text{"REM_RESP"} \\
& \wedge m.\text{to} = p \\
& \wedge \text{Receive}(m) \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![p] = \text{"IDLE"}] \\
& \wedge \text{joined}' = \text{joined} \setminus \{p\} \\
& \wedge \text{UNCHANGED } \langle \text{failed} \rangle
\end{aligned}$$

GiveUpMasterActions

$$\begin{aligned}
\text{GiveUpMasterReq} \triangleq & \wedge \exists p, q \in \text{PID} : \wedge p \neq q \\
& \wedge \text{IsMaster}(p) \wedge \text{state}[p] = \text{"MASTER"} \\
& \wedge q \in \text{RandomView} \cup \{p\} \\
& \wedge \neg \exists m \in \text{msgs} : \wedge m.\text{type} = \text{"GM_REQ"} \\
& \wedge m.\text{from} = p \\
& \wedge m.\text{to} = q \\
& \wedge \text{Send}([type \mapsto \text{"GM_REQ"}, from \mapsto p, to \mapsto q]) \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![p] = \text{"MASTER_GM_CONFIRM"}] \\
& \wedge \text{UNCHANGED } \langle \text{failed}, \text{joined} \rangle
\end{aligned}$$

$$\begin{aligned}
\text{GiveUpMasterResp} \triangleq & \wedge \exists p \in \text{PID} : \wedge \text{IsSlave}(p) \wedge \text{state}[p] = \text{"SLAVE"} \\
& \wedge \exists m \in \text{msgs} : \wedge m.\text{type} = \text{"GM_REQ"} \\
& \wedge m.\text{to} = p \\
& \wedge \text{ReceiveNSend}(m, [type \mapsto \text{"GM_RESP"}, from \mapsto p, to \mapsto m.\text{from}]) \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![p] = \text{"SLAVE_GM_ACCEPT"}] \\
& \wedge \text{UNCHANGED } \langle \text{failed}, \text{joined} \rangle
\end{aligned}$$

$$\begin{aligned}
\text{GiveUpMaster} \triangleq & \wedge \exists p \in \text{PID}, m \in \text{msgs} : \wedge \text{IsMaster}(p) \\
& \wedge \text{state}[p] = \text{"MASTER_GM_CONFIRM"} \\
& \wedge m.\text{type} = \text{"GM_RESP"} \\
& \wedge m.\text{to} = p \\
& \wedge \text{ReceiveNSend}(m, [type \mapsto \text{"GM_RDY_REQ"}, from \mapsto p, to \mapsto m.\text{from}]) \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![p] = \text{"SLAVE"}] \\
& \wedge \text{UNCHANGED } \langle \text{failed}, \text{joined} \rangle
\end{aligned}$$

$$\text{BecomeMaster} \triangleq \wedge \exists p \in \text{PID}, m \in \text{msgs} : \wedge \text{IsSlave}(p)$$

$$\begin{aligned}
& \wedge state[p] = \text{"SLAVE_GM_ACCEPT"} \\
& \wedge m.type = \text{"GM_RDY_REQ"} \\
& \wedge m.to = p \\
& \wedge Receive(m) \\
& \wedge state' = [state \text{ EXCEPT } ![p] = \text{"MASTER"}] \\
& \wedge \text{UNCHANGED } \langle failed, joined \rangle
\end{aligned}$$

$$\begin{aligned}
GiveUpMasterFail & \triangleq \wedge \exists p, q \in PID : \wedge IsMaster(p) \\
& \wedge state[p] = \text{"MASTER_GM_CONFIRM"} \\
& \wedge \neg \exists m \in msgs : \wedge m.type = \text{"GM_FAIL_REQ"} \\
& \quad \wedge m.from = p \\
& \quad \wedge m.to = q \\
& \wedge Send([type \mapsto \text{"GM_FAIL_REQ"}, from \mapsto p, to \mapsto q]) \\
& \wedge state' = [state \text{ EXCEPT } ![p] = \text{"MASTER"}] \\
& \wedge \text{UNCHANGED } \langle failed, joined \rangle
\end{aligned}$$

$$\begin{aligned}
BecomeMasterFail & \triangleq \wedge \exists p \in PID : \wedge IsSlave(p) \\
& \wedge state[p] = \text{"SLAVE_GM_ACCEPT"} \\
& \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE"}] \\
& \wedge \text{UNCHANGED } \langle failed, joined, msgs \rangle
\end{aligned}$$

CheckBrainActions

$$\begin{aligned}
CheckBrainShortCut & \triangleq \wedge \exists p \in PID : \wedge IsSlave(p) \\
& \wedge state[p] = \text{"SLAVE"} \\
& \wedge JoinedCardEst = 1 \\
& \wedge state' = [state \text{ EXCEPT } ![p] = \text{"MASTER"}] \\
& \wedge \text{UNCHANGED } \langle failed, joined, msgs \rangle
\end{aligned}$$

$$\begin{aligned}
CheckBrainReq & \triangleq \wedge \exists p \in PID : \wedge IsSlave(p) \\
& \wedge state[p] = \text{"SLAVE"} \\
& \wedge \neg \exists m \in msgs : \wedge m.type = \text{"CB_REQ"} \\
& \quad \wedge m.from = p \\
& \wedge Send([type \mapsto \text{"CB_REQ"}, from \mapsto p]) \\
& \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE_EL_CONFIRM"}] \\
& \wedge \text{UNCHANGED } \langle failed, joined \rangle
\end{aligned}$$

$$\begin{aligned}
CheckBrainResp & \triangleq \wedge \exists p \in PID, m \in msgs : \\
& \quad \wedge IsSlave(p) \\
& \quad \wedge state[p] = \text{"SLAVE"} \\
& \quad \wedge m.type = \text{"CB_REQ"} \\
& \quad \wedge m.from \neq p \\
& \quad \wedge \neg \exists m2 \in msgs : \wedge m2.type = \text{"CB_RESP+"} \vee m2.type = \text{"CB_RESP-"} \\
& \quad \quad \wedge m2.from = p \\
& \quad \wedge \vee \wedge ReceiveNSend(m, [type \mapsto \text{"CB_RESP+"}, from \mapsto p, to \mapsto m.from]) \\
& \quad \quad \wedge \text{UNCHANGED } state \\
& \quad \vee \wedge ReceiveNSend(m, [type \mapsto \text{"CB_RESP-"}, from \mapsto p, to \mapsto m.from])
\end{aligned}$$

$$\begin{aligned} & \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE_EL_ELECTION"}] \\ & \wedge \text{UNCHANGED } \langle failed, joined \rangle \end{aligned}$$

$$\begin{aligned} CheckBrainPos \triangleq & \wedge \exists p \in PID, m \in msgs : \wedge IsSlave(p) \\ & \wedge state[p] = \text{"SLAVE_EL_CONFIRM"} \\ & \wedge m.type = \text{"CB_RESP+"} \\ & \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE"}] \\ & \wedge \text{UNCHANGED } \langle failed, joined, msgs \rangle \end{aligned}$$

$$\begin{aligned} CheckBrainNeg \triangleq & \wedge \exists p \in PID : \wedge IsSlave(p) \\ & \wedge state[p] = \text{"SLAVE_EL_CONFIRM"} \\ & \wedge Cardinality(\{m \in msgs : m.type = \text{"CB_RESP-"}\}) * 2 > JoinedCardEst \\ & \wedge state' = [state \text{ EXCEPT } ![p] = \text{"MASTER"}] \\ & \wedge \text{UNCHANGED } \langle failed, joined, msgs \rangle \end{aligned}$$

$$\begin{aligned} CheckBrainFail \triangleq & \wedge \exists p \in PID : \wedge IsSlave(p) \\ & \wedge state[p] = \text{"SLAVE_EL_CONFIRM"} \vee state[p] = \text{"SLAVE_EL_ELECTION"} \\ & \wedge state' = [state \text{ EXCEPT } ![p] = \text{"SLAVE"}] \\ & \wedge \text{UNCHANGED } \langle failed, joined, msgs \rangle \end{aligned}$$

Specification

$$\begin{aligned} FailRecoverActions \triangleq & \vee Fail \\ & \vee Recover \\ & \vee MessageLoss \end{aligned}$$

$$\begin{aligned} JoinActions \triangleq & \vee Bootstrap \\ & \vee JoinReq \\ & \vee JoinResp \\ & \vee Join \end{aligned}$$

$$\begin{aligned} LeaveActions \triangleq & \vee LeaveReq \\ & \vee LeaveResp \\ & \vee Leave \end{aligned}$$

$$\begin{aligned} GiveUpMasterActions \triangleq & \vee GiveUpMasterReq \\ & \vee GiveUpMasterResp \\ & \vee GiveUpMaster \\ & \vee BecomeMaster \\ & \vee GiveUpMasterFail \\ & \vee BecomeMasterFail \end{aligned}$$

$$\begin{aligned} CheckBrainActions \triangleq & \vee CheckBrainShortCut \\ & \vee CheckBrainReq \\ & \vee CheckBrainResp \\ & \vee CheckBrainPos \\ & \vee CheckBrainNeg \\ & \vee CheckBrainFail \end{aligned}$$

$$Next \triangleq \vee FailRecoverActions$$

$\vee JoinActions$
 $\vee LeaveActions$
 $\vee GiveUpMasterActions$
 $\vee CheckBrainActions$

$$Spec \triangleq \wedge Init \\ \wedge \square [Next]_{vars}$$

Interesting properties and Invariants

The master does not change.

$$StableMaster \triangleq \exists p \in PID : IsMaster(p) \implies IsMaster'(p)$$

Wanting idle, joins the system.

$$WantingJoins \triangleq \forall i \in PID : MayJoin(i) \rightsquigarrow i \in joined$$

Wanting master, gives up.

$$WantingMaster \triangleq \text{TRUE}$$

Election terminates.

$$ElectionTerminate \triangleq \forall x \in joined : state[x] \in \{\text{"SLAVE"}, \text{"MASTER"}\}$$

Split brain.

$$SomeBrain \triangleq \exists m \in PID : IsMaster(m)$$

No brain.

$$NoBrain \triangleq \neg SomeBrain$$

Split brain.

$$SplitBrain \triangleq \exists m1, m2 \in PID : m1 \neq m2 \wedge IsMaster(m1) \wedge IsMaster(m2)$$

There is a single leader in the system.

$$OneBrain \triangleq \wedge SomeBrain \\ \wedge \neg SplitBrain$$

No process fails, no process joins.

$$StableSystem \triangleq \text{UNCHANGED } \langle joined \rangle \\ \text{No losses}$$

Types are always correct.

THEOREM $Spec \implies \square TypeInvariant$

If the master stops changing, every node trying to join eventually succeeds.

THEOREM $Spec \implies StableMaster \rightsquigarrow WantingJoins$

If the system stabilizes then all elections cease and all processors are

slaves, masters, or out of the system.

THEOREM $Spec \implies StableSystem \leadsto ElectionTerminate$

THEOREM $Spec \implies JoinedCardEst \leq Cardinality(joined) \leadsto SomeBrain$

THEOREM $Spec \implies JoinedCardEst = Cardinality(joined) \leadsto (\Diamond SomeBrain \wedge \neg SplitBrain)$

O Componente DM9000A na biblioteca SoPC

O DM9000A é um chip controlador *Fast Ethernet* fabricado pela DAVICOM Davicom (2005b). É um circuito que integra a camada Física e a camada de Rede de uma conexão, conforme a Figura C.1. Como se pode ver, ele possui interface para processador geral, interface EEPROM, um 10/100 PHY e uma SRAM interna de 16KB (13KB-RX *Fifo* e 3KB-TX *Fifo*).

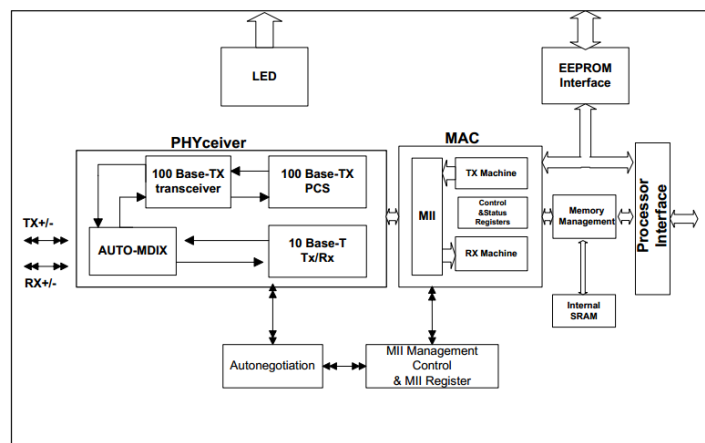


Figura C.1: Diagrama de blocos interno do DM9000A Davicom (2005b)

Ele suporta modo de operação de 8 ou 16 bits na interface com o processador para acessos à sua memória interna. Antes de se utilizar o DM9000A como interface de rede, deve-se inicializá-lo a partir de uma rotina pré-estabelecida. Além disso, para realizar a transmissão e/ou a recepção de um pacote, outras rotinas específicas devem ser exe-

cutadas. A execução dessas rotinas é feita acessando os registradores internos através da interface com o processador. A figura C.2 apresenta a os sinais de conexão entre o DM9000A e a interface do processador (pinos do lado esquerdo do chip na figura).

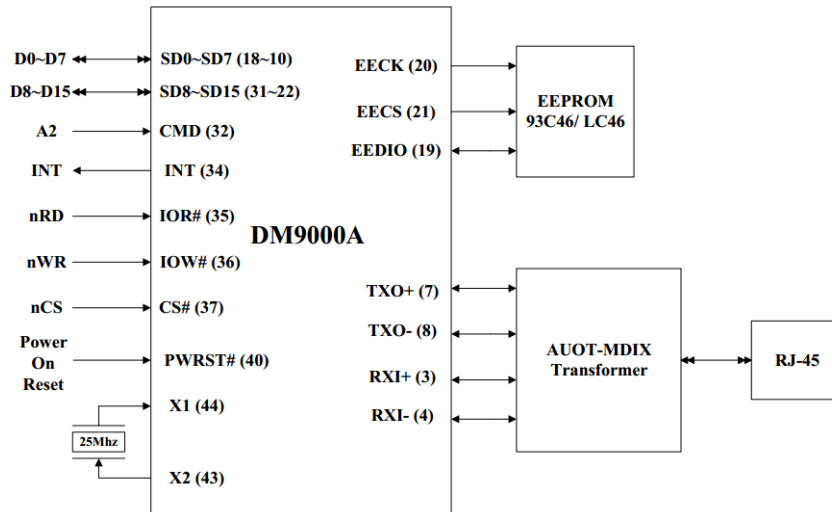


Figura C.2: Conexão de sinais com a interface do processador Davicom (2005b)

A tabela C.1 refere-se aos pinos mostrados na Figura C.2 e fornece características sobre aqueles que são acessados pelo processador.

Tabela C.1: DM9000A - Pinos para a interface do processador Davicom (2005b)

Sinal Processador	Sinal DM9000A	I/O	Descrição
nRD	IRD#	I	Comando READ do processador. Este pino é ativo em nível baixo, por padrão.*
nWR	IOW#	I	Comando WRITE do processador. Este pino é ativo em nível baixo, por padrão.*
nCS / nAEN	CS#	I	Chip Select. Um sinal ativo baixo é usado para selecionar o DM9000A.*
SD0 ~ 7	SD0 ~ 7	I/O	Barramento de dados 0 ~ 7
SD8 ~ 15	SD8 ~ 15	I/O	Barramento de dados 8 ~ 15
CMD	CMD	I/O	Tipo de comando. Quando baixo, o acesso do comando é INDEX <i>port</i> Quando alto, o acesso do comando é DATA <i>port</i>
INT	INT	I/O	Requisição de Interrupção. Este pino é ativo em nível alto.*

*Sua polaridade pode ser modificada pelas configurações da EEPROM

Todo o controle e configuração do DM9000A é feito acessando os registradores de controle e estado no bloco de controle de acesso MAC. São 255 registradores endereçados de 0x00h a 0xFF. Para transmitir um pacote, deve-se enviar os dados deste pacote para

determinados registradores e eles serão, de forma transparente ao usuário, transmitidos para a SRAM interna. Para receber um pacote, deve-se acessar determinados registradores para que estes busquem os dados na memória. Em suma, todas as operações e configurações do DM9000A são feitas acessando os registradores de estado e controle, disponíveis para consulta em Davicom (2005b).

No entanto, no DM9000A só há dois registradores que podem ser acessados diretamente: o *INDEX port* e *DATA port*. Estes dois registradores são distinguidos pelo dado de controle no pino *CMD#*. Quando o *CMD#* está baixo, é um ciclo de comando e o *INDEX* é acessado. Quando o *CMD* está alto, é um ciclo de troca de dados e o *DATA* é acessado. Todos os registradores de controle e estado do DM9000A são acessados indiretamente pelos *INDEX* e *DATA ports*. A sequência de comandos para acessar um registrador específico é: primeiramente, escreve-se o endereço do registrador no *INDEX port*, então, num segundo ciclo lê/escreve seu conteúdo do/no *DATA port*.

Em um ciclo de comando, o DM9000A é acessado ao setar o pino *CS#* e os pinos *IOR#* ou *IOW#*. Estes devem estar ativos (um ou outro, dependendo da operação) para ler ou escrever o valor no *INDEX* ou *DATA ports*. As rotinas de inicialização, transmissão e recebimento de pacotes podem ser consultadas no *Datasheet* Davicom (2005b) e o *Application Notes* Davicom (2005a) do controlador. Ademais, a função de cada registrador interno também é explicada nestes documentos.

C.1 Integração com o *SoPC Builder*

O DM9000A é um elemento que não tem controlador nativo na biblioteca *Megacore IP* e, por tal, deve-se projetá-lo e incluí-lo na biblioteca da ferramenta *SoPC Builder*. Para isso, é necessário ter um arquivo escrito em linguagem de descrição de hardware (HDL) que descreva o comportamento do controlador e suas interfaces, sendo uma interface para o barramento *Avalon* e outra para o chip DM9000A, conforme a Figura C.3

O controlador deve ter duas interfaces, uma *Avalon-MM* se conectando aos pinos de dados, endereço e controle com o barramento *Avalon* (para troca dados com o processador) e outra se conectando ao chip DM9000A para troca de dados com o ambiente externo. A Figura C.3 mostra um sistema que possui um controlador (*DM9000A_Controller*) estabelecendo a interconexão entre o chip DM9000A e o barramento *Avalon* que, por sua vez, trafega os dados do sistema embarcado e provê interconexão com todos os periféricos.

O arquivo *DM9000A_HDL_Controller.v* (Seção C.2) está escrito em na HDL *Verilog* e descreve o controlador DM9000A. Ao observar a entidade do controlador descrita no código C.1, pode-se inferir o esquema de interconexão mostrado na Figura C.3.

Já a Figura C.4 ilustra o que é feito segundo a lógica de controle interno no código que descreve o DM9000A no *SoPC Builder*. As setas que conectam as portas significam o que

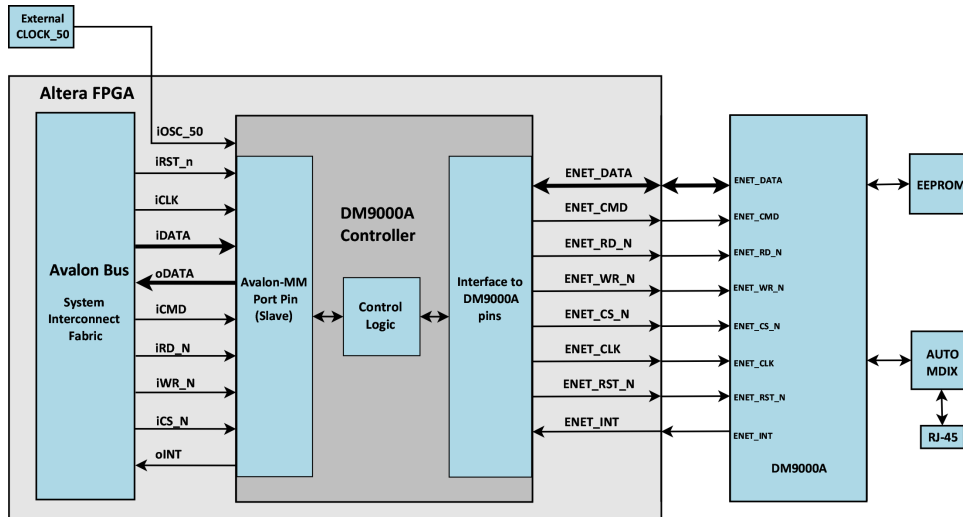


Figura C.3: Controlador *DM9000A Controller* e suas interconexões

é feito internamente no estado normal de execução, obviamente respeitando-se o tempo de um ciclo de *clock* para transferência de dados. Caso o *reset* seja acionado, todas as saídas assumirão os valores *default*, o que não é representado.

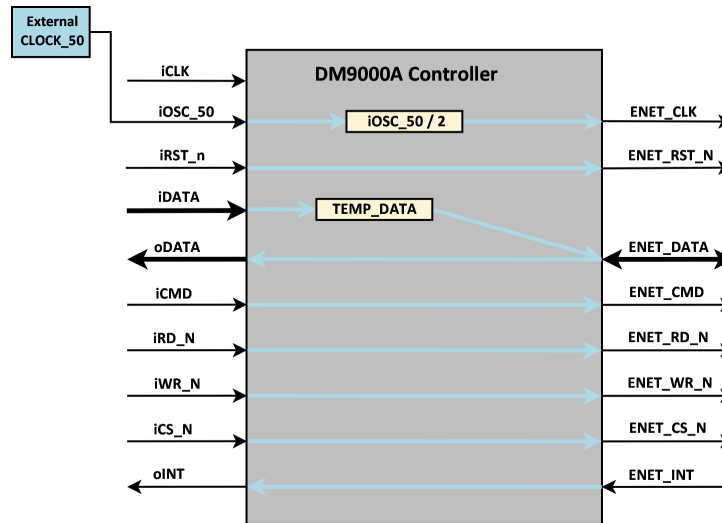


Figura C.4: Representação da lógica interna do controlador DM9000A

Ao inserir o código de descrição do controlador no *SoPC Builder*, é necessário configurar os tipos dos pinos que compõem a entidade do controlador (todas as interfaces) para que o processador possa saber qual o propósito de cada pino e direcionar corretamente os dados e o controle do componente. A Tabela C.2 mostra os tipos de sinais atribuídos a cada pino do *DM9000A_Controller* para acesso ao barramento e ao chip DM9000A.

A interface tipo *clock* deve receber pinos de *clock* e *reset*. Já a interface *avalon_slave* refere-se aos pinos que serão conectados ao barramento *Avalon* para transferência de dados, controle e endereço. Um pino deve ser classificado como *Interrupt_Sender* quando ele for a via pela qual as interrupções serão enviadas ao processador. Por fim, os pinos

Tabela C.2: Classificação dos pinos do controlador DM9000A

Sinal	Interface	Tipo	Direção
iCLK	clock	clock	I
iRST_N	clock	reset_n	I
iDATA	avalon_slave	write_data	I
oDATA	avalon_slave	read_data	O
iCMD	avalon_slave	address	I
iRD_N	avalon_slave	read_n	I
iWR_N	avalon_slave	write_n	I
iCS_N	avalon_slave	chipselect_n	I
oINT	interrupt_sender	irq	O
iOSC_50	avalon_slave_export	export	I
ENET_DATA	avalon_slave_export	export	BIDIR
ENET_CMD	avalon_slave_export	export	O
ENET_RD_N	avalon_slave_export	export	O
ENET_WR_N	avalon_slave_export	export	O
ENET_CS_N	avalon_slave_export	export	O
ENET_RST_N	avalon_slave_export	export	O
ENET_INT	avalon_slave_export	export	I
ENET_CLK	avalon_slave_export	export	O

com interface *avalon_slave_export* são aqueles que não serão conectados ao barramento, mas sim exportados para fora do sistema de forma que dê acesso a outros componentes (é por esses pinos que o controlador se conecta ao chip DM9000A externo).

Segundo a especificação, o DM9000A é inicializado seguindo uma rotina de acesso, leitura e escrita nos registradores específicos e só depois de um roteiro ele poderá funcionar corretamente. Por isso, os arquivos que especificam essas rotinas devem ser inseridos no projeto. Estes arquivos são os *Device Drivers* e, para este caso, os códigos *DM9000A.c* e *DM9000A.h* são inseridos para este fim.

Uma análise destes códigos deve ser feita em juntamente com o ao *Datasheet* Davicom (2005b) e o *Aplication Notes* Davicom (2005a), pois os códigos tratam do acesso a registradores que são explicados nestes documentos. Da forma em que estão, os códigos basicamente tratam de três funções principais: inicialização do chip, transmissão de pacote e recebimento de pacote. As funções devem ser chamadas no programa principal para serem executadas.

Nos códigos, a conteúdo a ser transmitido em um pacote deve ter um tamanho máximo de 1518 bytes para ser suportado pelo DM9000A. A função *DM9000A_init()* que é para a inicialização do chip. A função *ReceivePacket* confere se o pacote chegou sem erro (através de CRC), copia o conteúdo da SRAM interna para os registradores específicos. Após copiar todo o conteúdo da memória, a função *ReceivePacket()* limpa o conteúdo da SRAM. Já a função *TransmitPacket* copia o conteúdo dos registradores para a SRAM e procede com a transmissão.

As funções principais para acesso ao chip externo são IOWR e IORD. Quando se utiliza estas funções, a aplicação acessa os componentes que formam o SoC, tanto para escrita como para a leitura de valores. Para utilizá-las, três parâmetros são passados para as funções. Primeiro, o nome do componente que deve ser acompanhado do sufixo `_BASE`. Segundo, o valor 0 ou 1, para dizer se é um ciclo de comando ou de dados (para o pino CMD). Terceiro, o valor que se quer escrever no controlador (para a função IOWR, no caso da IORD não há o terceiro parâmetro).

O fluxo de dados é o seguinte: primeiro o dado é passado como parâmetro para função, em seguida ele é transmitido via JTAG-UART para o SoC. Ao atingir o JTAG-UART *Core*, os dados são enviados ao processador NIOS II pelo barramento *Avalon*. Então, o processador consulta sua tabela de endereços e envia os dados pelo mesmo barramento ao controlador DM9000A. Chegando ao controlador e de acordo com a lógica de controle interna do mesmo, os dados são transferidos à sua interface externa e conseguem chegar ao chip DM9000A para transmissão. O fluxo de dados durante a recepção é o inverso do explicado para o fluxo de transmissão de um pacote.

C.2 Códigos

Código C.1: DM9000A_HDL_Controller.v - Código de descrição do controlador DM9000A

```

1  module DM9000A_IF(//HOST Side
2
3      iDATA,
4      oDATA,
5      iCMD,
6      iRD_N,
7      iWR_N,
8      iCS_N,
9      iRST_N,
10     iCLK,
11     iOSC_50,
12     oINT,
13     //DM9000A Side
14     ENET_DATA,
15     ENET_CMD,
16     ENET_RD_N,
17     ENET_WR_N,
18     ENET_CS_N,
19     ENET_RST_N,
20     ENET_INT,
21     ENET_CLK);
22 // HOST Side
23 input    [15:0] iDATA;
24 input    iCMD;
25 input    iRD_N;
26 input    iWR_N;
27 input    iCS_N;
28 input    iRST_N;
29 input    iCLK;
30 input    iOSC_50;
31 output   [15:0] oDATA;
32 output   oINT;
33 // DM9000A Side
34 inout    [15:0] ENET_DATA;
35 output    ENET_CMD;
36 output    ENET_RD_N;
37 output    ENET_WR_N;
38 output    ENET_CS_N;
39 output    ENET_RST_N;
40 output    ENET_CLK;
41 input    ENET_INT;
42
43 reg      [15:0] TMP_DATA;
44 reg      ENET_CMD;

```

```

44 reg          ENET_RD_N;
45 reg          ENET_WR_N;
46 reg          ENET_CS_N;
47 reg          ENET_CLK;
48 reg    [15:0] oDATA;
49 reg          oINT;
50
51 assign ENET_DATA = ENET_WR_N ? 16'hzzzz : TMP_DATA;
52
53 always@(posedge iCLK or negedge iRST_N)
54 begin
55     if (!iRST_N)
56     begin
57         TMP_DATA <= 0;
58         ENET_CMD <= 0;
59         ENET_RD_N <= 1;
60         ENET_WR_N <= 1;
61         ENET_CS_N <= 1;
62         oDATA <= 0;
63         oINT <= 0;
64     end
65     else
66     begin
67         oDATA <= ENET_DATA;
68         oINT <= ENET_INT;
69         TMP_DATA <= iDATA;
70         ENET_CMD <= iCMD;
71         ENET_CS_N <= iCS_N;
72         ENET_RD_N <= iRD_N;
73         ENET_WR_N <= iWR_N;
74     end
75 end
76
77 always@(posedge iOSC_50)
78 ENET_CLK <= ~ENET_CLK;
79
80 assign ENET_RST_N = iRST_N;
81
82 endmodule

```

Código C.2: *Device Driver* DM9000A.c - Programa de configuração do DM9000A

```

1 #include <stdio.h>
2 #include "DM9000A.H"
3 #include "basic_io.h"
4 //-----
5 void iow(unsigned int reg, unsigned int data)
6 {
7     IOWR(DM9000A_0_BASE, IO_addr, reg);
8     usleep(STD_DELAY);
9     IOWR(DM9000A_0_BASE, IO_data, data);
10 }
11 //-----
12 unsigned int ior(unsigned int reg)
13 {
14     IOWR(DM9000A_0_BASE, IO_addr, reg);
15     usleep(STD_DELAY);
16     return IORD(DM9000A_0_BASE, IO_data);
17 }
18 //-----
19 void phy_write (unsigned int reg, unsigned int value)
20 {
21     /* set PHY register address into EPAR REG. 0CH */
22
23     /* PHY register address setting, and DM9000_PHY offset = 0x40 */
24     iow(0x0C, reg | 0x40);
25
26     /* fill PHY WRITE data into EPDR REG. 0EH & REG. 0DH */
27     iow(0x0E, ((value >> 8) & 0xFF)); /* PHY data high_byte */
28     iow(0x0D, value & 0xFF); /* PHY data low_byte */
29
30     /* issue PHY + WRITE command = 0xa into EPCR REG. 0BH */
31     iow(0x0B, 0x8); /* clear PHY command first */
32     IOWR(DM9000A_0_BASE, IO_data, 0x0A); /* issue PHY + WRITE command */
33     usleep(STD_DELAY);
34     IOWR(DM9000A_0_BASE, IO_data, 0x08); /* clear PHY command again */
35     usleep(50); /* wait 1~30 us (>20 us) for PHY + WRITE completion */
36 }
37 //-----
38 /* DM9000_init I/O routine */
39 unsigned int DM9000_init (void) /* initialize DM9000 LAN chip */
40 {
41     unsigned int i;
42
43     /* set the internal PHY power-on (GPIOs normal settings) */
44     iow(0x1E, 0x01); /* GPCR REG. 1EH = 1 selected GPIO0 "output"
45                     port for internal PHY */
46     iow(0x1F, 0x00); /* GPR REG. 1FH GEPIO0 Bit [0] = 0

```

```

47          to activate internal PHY */
48  msleep(5);          /* wait > 2 ms for PHY power-up ready */
49
50  /* software-RESET NIC */
51  iow(NCR, 0x03);      /* NCR REG. 00 RST Bit [0] = 1 reset on,
52                      and LBK Bit [2:1] = 01b MAC loopback on */
53  usleep(20);          /* wait > 10us for a software-RESET ok */
54  iow(NCR, 0x00);      /* normalize */
55  iow(NCR, 0x03);
56  usleep(20);
57  iow(NCR, 0x00);
58
59  /* set GPIO0=1 then GPIO0=0 to turn off and on the internal PHY */
60  iow(0x1F, 0x01);     /* GPR PHYPD Bit [0] = 1 turn-off PHY */
61  iow(0x1F, 0x00);     /* PHYPD Bit [0] = 0 activate phyxcer */
62  msleep(10);          /* wait >4 ms for PHY power-up */
63
64  /* set PHY operation mode */
65  phy_write(0, PHY_reset); /* reset PHY:
66                          registers back to the default states */
67  usleep(50);          /* wait >30 us for PHY software-RESET ok */
68  phy_write(16, 0x404); /* turn off PHY reduce-power-down mode only */
69  phy_write(4, PHY_txab); /* set PHY TX advertised ability:
70                          ALL + Flow_control */
71  phy_write(0, 0x1200); /* PHY auto-NEGO re-start enable
72                          (RESTART_AUTO_NEGOTIATION +
73                          AUTO_NEGOTIATION_ENABLE) to auto sense and
74                          recovery PHY registers */
75  msleep(5);          /* wait >2 ms for PHY auto-sense linking to partner */
76
77  /* store MAC address into NIC */
78  for (i = 0; i < 6; i++)
79  iow(16 + i, ether_addr[i]);
80
81  /* clear any pending interrupt */
82  iow(ISR, 0x3F);      /* clear the ISR status: PRS, PTS, ROS, ROOS 4 bits,
83                      by RW/C1 */
84  iow(NSR, 0x2C);      /* clear the TX status: TX1END, TX2END, WAKEUP 3 bits,
85                      by RW/C1 */
86
87  /* program operating registers ~ */
88  iow(NCR, NCR_set);   /* NCR REG. 00 enable the chip functions
89                      (and disable this MAC loopback mode back to
90                      normal) */
91  iow(0x08, BPTR_set); /* BPTR REG.08 (if necessary) RX Back Pressure
92                      Threshold in Half duplex moe only:
93                      High Water 3KB, 600 us */

```

```

94     iow(0x09, FCTR_set);  /* FCTR REG.09 (if necessary) Flow Control
95                           Threshold setting High/ Low Water Overflow
96                           5KB/ 10KB */
97     iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary) RX/TX Flow Control
98                           Register enable TXPEN, BKPM (TX_Half),
99                           FLCE (RX) */
100    iow(0x0F, 0x00);      /* Clear the all Event */
101    iow(0x2D, 0x80);      /* Switch LED to mode 1 */
102
103    /* set other registers depending on applications */
104    iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
105
106    /* enable interrupts to activate DM9000 ~on */
107    iow(IMR, INTR_set);    /* IMR REG. FFH PAR=1 only,
108                           or + PTM=1& PRM=1 enable RxTx interrupts */
109
110    /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
111
112    /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
113    iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
114
115    /* RETURN "DEVICE_SUCCESS" back to upper layer */
116    return (ior(0x2D)==0x80) ? DMFE_SUCCESS : DMFE_FAIL;
117 }
118 //-----
119 /* Transmit one Packet TX I/O routine */
120 unsigned int TransmitPacket(unsigned char *data_ptr, unsigned int tx_len)
121 {
122     unsigned int i;
123
124     /* mask NIC interrupts IMR: PAR only */
125     iow(IMR, PAR_set);
126
127     /* issue TX packet's length into TXPLH REG. FDH & TXPLL REG. FCH */
128     iow(0xFD, (tx_len >> 8) & 0xFF); /* TXPLH High_byte length */
129     iow(0xFC, tx_len & 0xFF);        /* TXPLL Low_byte length */
130
131     /* write transmit data to chip SRAM */
132     IOWR(DM9000A_0_BASE, IO_addr, MWCMD); /* Set MWCMD REG. F8H TX
133                                             I/O port ready */
134     for (i = 0; i < tx_len; i += 2)
135     {
136         usleep(STD_DELAY);
137         IOWR(DM9000A_0_BASE, IO_data, (data_ptr[i+1]<<8)|data_ptr[i] );
138     }
139
140     /* issue TX polling command activated */

```

```

141 iow(TCR , TCR_set | TX_REQUEST); /* TXCR Bit [0] TXREQ auto clear
142                                after TX completed */
143
144 /* wait TX transmit done */
145 while (! ( ior (NSR)&0x0C))
146     usleep (STD_DELAY);
147
148 /* clear the NSR Register */
149 iow (NSR,0x00);
150
151 /* re-enable NIC interrupts */
152 iow (IMR, INTR_set);
153
154 /* RETURN "TX_SUCCESS" to upper layer */
155 return DMFE_SUCCESS;
156 }
157 //-----
158 /* Receive One Packet I/O routine */
159 unsigned int ReceivePacket (unsigned char *data_ptr,unsigned int *rx_len)
160 {
161     unsigned char rx_READY,GoodPacket;
162     unsigned int Tmp, RxStatus, i;
163
164     RxStatus = rx_len[0] = 0;
165     GoodPacket=FALSE;
166
167     /* mask NIC interrupts IMR: PAR only */
168     iow (IMR, PAR_set);
169
170     /* dummy read a byte from MRCMDX REG. F0H */
171     rx_READY = ior (MRCMDX);
172
173     /* got most updated byte: rx_READY */
174     rx_READY = IORD (DM9000A_0_BASE,IO_data)&0x03;
175     usleep (STD_DELAY);
176
177     /* check if (rx_READY == 0x01): Received Packet READY? */
178     if (rx_READY == DM9000_PKT_READY)
179     {
180         /* got RX_Status & RX_Length from RX SRAM */
181         IOWR (DM9000A_0_BASE, IO_addr, MRCMD); /* set MRCMD REG. F2H RX I/O
182                                                port ready */
183         usleep (STD_DELAY);
184         RxStatus = IORD (DM9000A_0_BASE,IO_data);
185         usleep (STD_DELAY);
186         rx_len[0] = IORD (DM9000A_0_BASE,IO_data);
187

```

```

188      /* Check this packet_status GOOD or BAD? */
189      if ( !(RxStatus & 0xBF00) && (rx_len[0] < MAX_PACKET_SIZE) )
190      {
191          /* read 1 received packet from RX SRAM into RX buffer */
192          for (i = 0; i < rx_len[0]; i += 2)
193          {
194              usleep(STD_DELAY);
195              Tmp = IORD(DM9000A_0_BASE, IO_data);
196              data_ptr[i] = Tmp&0xFF;
197              data_ptr[i+1] = (Tmp>>8)&0xFF;
198          }
199          GoodPacket=TRUE;
200      } /* end if (GoodPacket) */
201      else
202      {
203          /* this packet is bad, dump it from RX SRAM */
204          for (i = 0; i < rx_len[0]; i += 2)
205          {
206              usleep(STD_DELAY);
207              Tmp = IORD(DM9000A_0_BASE, IO_data);
208          }
209          printf("\nError\n");
210          rx_len[0] = 0;
211      } /* end if (!GoodPacket) */
212  } /* end if (rx_READY == DM9000_PKT_READY) ok */
213  //

```

```

214      else if(rx_READY) /* status check first byte: rx_READY Bit[1:0]
215                          must be "00"b or "01"b */
216      {
217          /* software-RESET NIC */
218          iow(NCR, 0x03); /* NCR REG. 00 RST Bit [0] = 1 reset on,
219                          and LBK Bit [2:1] = 01b MAC loopback on */
220          usleep(20); /* wait > 10us for a software-RESET ok */
221          iow(NCR, 0x00); /* normalize */
222          iow(NCR, 0x03);
223          usleep(20);
224          iow(NCR, 0x00);
225          /* program operating registers~ */
226          iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions (and
227                              disable this MAC loopback mode back to normal)
228                              */
229          iow(0x08, BPTR_set); /* BPTR REG.08 (if necessary) RX Back Pressure
230                              Threshold in Half duplex mode only: High Water
231                              3KB,
232                              600 us */

```

```

231     iow(0x09, FCTR_set);  /* FCTR REG.09 (if necessary) Flow Control
232                           Threshold setting High/ Low Water Overflow
233                           5KB/ 10KB */
234     iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary) RX/TX Flow
                           Control
235
                           Register enable TXPEN, BKPM (TX_Half), FLCE (
                           RX) */
236     iow(0x0F, 0x00);      /* Clear the all Event */
237     iow(0x2D, 0x80);      /* Switch LED to mode 1 */
238     /* set other registers depending on applications */
239     iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
240     /* enable interrupts to activate DM9000 ~on */
241     iow(IMR, INTR_set);    /* IMR REG. FFH PAR=1 only, or + PTM=1& PRM=1
242                           enable RxTx interrupts */
243     /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
244     /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
245     iow(RCR, RCR_set | RX_ENABLE | PASS_MULTICAST);
246 } /* end NIC H/W system Data-Bus error */
247
248     return GoodPacket ? DMFE_SUCCESS : DMFE_FAIL;
249 }
250 //_____

```

Código C.3: DM9000A.h - Programa auxiliar de configuração do DM9000A

```

1 #ifndef __DM9000A_H__
2 #define __DM9000A_H__
3
4 #define IO_addr      0
5 #define IO_data      1
6
7 /* Network Control Register REG. 00 */
8 #define NCR          0x00
9 /* Network Status Register REG. 01 */
10 #define NSR          0x01
11 /* Transmit Control Register REG. 02 */
12 #define TCR          0x02
13 /* Receive Control Register REG. 05 */
14 #define RCR          0x05
15 /* TX early Control Register REG. 30 */
16 #define ETXCSR       0x30
17 /* RX FIFO I/O port command READ for dummy read a byte from RX SRAM */
18 #define MRCMDX       0xF0
19 /* RX FIFO I/O port command READ from RX SRAM */
20 #define MRCMD        0xF2
21 /* TX FIFO I/O port command WRITE into TX FIFO */
22 #define MWCMD        0xF8
23 /* NIC Interrupt Status Register REG. FEH */
24 #define ISR          0xFE
25 /* NIC Interrupt Mask Register REG. FFH */
26 #define IMR          0xFF
27
28 #define NCR_set       0x00
29 #define TCR_set       0x00
30 /* TCR REG. 02 TXREQ Bit [0] = 1 polling Transmit Request command */
31 #define TX_REQUEST   0x01
32 /* packet disable TX Jabber Timer */
33 #define TCR_long      0x40
34 /* skip CRC_packet and skip LONG_packet */
35 #define RCR_set       0x30
36 /* RCR REG. 05 RXEN Bit [0] = 1 to enable RX machine */
37 #define RX_ENABLE     0x01
38 /* packet disable RX Watchdog Timer */
39 #define RCR_long      0x40
40 /* RCR REG. 05 PASS_ALL_MULTICAST Bit [3] = 1: RCR_set value ORed 0x08 */
41 #define PASS_MULTICAST 0x08
42 /* BPTR REG. 08 RX Back Pressure Threshold:
43    High Water Overflow Threshold setting 3KB and Jam_Pattern_Time = 600 us
44    */
45 #define BPTR_set      0x3F
46 /* FCTR REG. 09 High/ Low Water Overflow Threshold setting 5KB/ 10KB */

```

```

46 #define FCTR_set          0x5A
47 /* RTFCR REG. 0AH RX/TX Flow Control Register enable TXPEN + BKPM(TX_Half)
   + FLCE(RX) */
48 #define RTFCR_set          0x29
49 /* Early Transmit Bit [7] Enable and Threshold 0~3: 12.5%, 25%, 50%, 75% */
50 #define ETXCSR_set         0x83
51 /* IMR REG. FFH: PAR +PRM, or 0x83: PAR + PRM + PTM */
52 #define INTR_set           0x81
53 /* IMR REG. FFH: PAR only, RX/TX FIFO R/W Pointer Auto Return enable */
54 #define PAR_set            0x80
55 /* PHY reset: some registers back to default value */
56 #define PHY_reset          0x8000
57 /* set PHY TX advertised ability: Full-capability + Flow-control (if
   necessary) */
58 #define PHY_txab           0x05e1
59 /* set PHY media mode: Auto negotiation (AUTO sense) */
60 #define PHY_mode           0x3100
61 /* standard delay 20 us */
62 #define STD_DELAY          20
63
64 #define DMFE_SUCCESS        0
65 #define DMFE_FAIL           1
66
67 #define TRUE                1
68 #define FALSE               0
69
70 #define DM9000_PKT_READY    0x01 /* packets ready to receive */
71 #define PACKET_MIN_SIZE     0x40 /* Received packet min size */
72 #define MAX_PACKET_SIZE     1522 /* RX largest legal size packet with fcs &
   QoS */
73 #define DM9000_PKT_MAX      3072 /* TX 1 packet max size without 4-byte CRC
   */
74 //-----
75 unsigned char ether_addr[6]={ 0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F };
76 //-----
77 void          iow(unsigned int reg, unsigned int data);
78 unsigned int  ior(unsigned int reg);
79 void          phy_write(unsigned int reg, unsigned int value);
80 /* DM9000_init I/O routine */
81 unsigned int  DM9000_init (void);
82 /* Transmit One Packet TX I/O routine */
83 unsigned int  TransmitPacket(unsigned char *data_ptr, unsigned int tx_len);
84 /* Receive One Packet I/O routine */
85 unsigned int  ReceivePacket (unsigned char *data_ptr, unsigned int *rx_len);
86 //-----
87
88 #endif

```
