UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

IMPLEMENTAÇÃO DE ARQUITETURAS DE PILHA UDP/IP EM HARDWARE RECONFIGURÁVEL BASEADO NO DESEMPENHO DE VAZÃO, LATÊNCIA E TAXA DE PERDA DE QUADROS

DISSERTAÇÃO DE MESTRADO

Fernando Luís Herrmann

Santa Maria, RS, Brasil
2010

IMPLEMENTAÇÃO DE ARQUITETURAS DE PILHA UDP/IP EM HARDWARE RECONFIGURÁVEL BASEADO NO DESEMPENHO DE VAZÃO, LATÊNCIA E TAXA DE PERDA DE QUADROS

por

Fernando Luís Herrmann

Dissertação apresentada ao Programa de Pós-Graduação em Informática, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Computação.

Orientador: Prof. Dr. João Baptista dos Santos Martins

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

IMPLEMENTAÇÃO DE ARQUITETURAS DE PILHA UDP/IP EM HARDWARE RECONFIGURÁVEL BASEADO NO DESEMPENHO DE VAZÃO, LATÊNCIA E TAXA DE PERDA DE QUADROS

elaborada por
Fernando Luís Herrmann

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

João Baptista dos Santos Martins, Dr.
(Presidente/Orientador)

Diógenes Cecilio da Silva Jr, Dr. (UFMG)

Raul Ceretta Nunes, Dr. (UFSM)

Roseclea Duarte Medina, Dr. (UFSM)

Santa Maria, 11 de Março de 2010

Dedico este trabalho à minha noiva Márcia Schons por todo o carinho e compreensão.

AGRADECIMENTOS

Agradeço principalmente à minha noiva, Márcia Schons. Sem ela esse caminho com certeza não poderia ter sido trilhado. Muito Obrigado por todas as palavras de carinho e incentivo. Agradeço também aos meus pais, Alberto e Lovani Herrmann, por sempre terem me apoiado.

Um agradecimento especial ao meu orientador, João Baptista dos Santos Martins, pela orientação deste trabalho e por todas as palavras de motivação e oportunidades oferecidas. Agradeço também aos demais professores do Gmicro, que de uma forma ou outra colaboraram para o desenvolvimento deste trabalho.

Também agradeço, pela amizade e pelo apoio, a todos os colegas do Gmicro. Cabe ressaltar, Josué Paulo José de Freitas pela excelente co-orientação e pela revisão deste trabalho. Guilherme Perin, pelas implementações iniciais e por toda ajuda oferecida. E também todos aqueles que trabalharam de alguma forma comigo durante esse tempo: Gustavo Fernando Dessbesell, Mateus Beck Fonseca, Leonardo Londero de Oliveira, Paulo César Comassetto de Aguirre, Rafael Bertagnolli, Taimur Gibran Rabuske Kuntz, Lucas Teixeira e Cristian Müller.

Agradeço também os professores Diógenes Cecilio da Silva Jr, Raul Ceretta Nunes e Roseclea Duarte Medina por seus comentários extremamente valiosos durante a banca de avaliação desta dissertação.

Também merecem ser citados e tem toda a minha graditão, Marinelma Aimi de Carvalho, secretária do PPGI da UFSM, por sua colaboração durante este período. FINEP, SEBRAE e CNPQ pelo suporte oferecido. SUN pela disponibilização da placa XUPV5-LX110T. Xilinx pelo suporte das ferramentas. Cadence e em especial ao programa CI BRASIL. Universidade de Santa Cruz do Sul (UNISC) pelo empréstimo da ferramenta de teste JDSU. Cristiano Both e Gustavo Araújo por toda ajuda prestada na utilização das ferramentas no laboratório da UNISC. Felipe Henes, pelas implementações iniciais do MII.

E finalmente, mas acima de tudo, a Deus e a Meishu Sama por me ensinar uma forma diferente de ver a vida e todas as coisas em sua volta.

“Quem deseja ser feliz, deve primeiramente tornar feliz seu semelhante pois a Divina recompensa que disto provém, será a Verdadeira Felicidade.”

Meishu Sama

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria, RS, Brasil

IMPLEMENTAÇÃO DE ARQUITETURAS DE PILHA UDP/IP EM HARDWARE RECONFIGURÁVEL BASEADO NO DESEMPENHO DE VAZÃO, LATÊNCIA E TAXA DE PERDA DE QUADROS

AUTOR: FERNANDO LUÍS HERRMANN
ORIENTADOR: JOÃO BAPTISTA DOS SANTOS MARTINS

Local da Defesa e Data: Santa Maria, 11 de Março de 2010.

Este trabalho apresenta a implementação de três arquiteturas da pilha de comunicação UDP/IP em *hardware* reconfigurável. Também apresenta o desenvolvimento de um Testador baseado na metodologia da RFC 2544 e implementado em uma placa dotada de dispositivo FPGA. Esse Testador foi utilizado na obtenção dos resultados de vazão, latência e taxa de perda de quadros. O desempenho do projeto apresentou, em média, 89% a mais de vazão, para quadros de 64 *bytes*, que uma pilha de comunicação implementada em *software* (PC) e executada sobre um microprocessador de propósito geral. Em termos de latência, o projeto apresentou uma latência 389 vezes menor para quadros de 64 *bytes* e 13 vezes menor para quadros de 1518 *bytes*, que o PC. E em relação à taxa de perda de quadros, o projeto não apresentou perda para nenhum dos tamanhos de quadros utilizados durante os testes, enquanto o PC apresentou perda de quase 98% para quadros de 64 *bytes*.

Palavras-chave: Pilha de comunicação UDP/IP; FPGA; RFC 2544; Vazão; Latência; Taxa de perda de quadros.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Federal University of Santa Maria, RS, Brazil

IMPLEMENTATION OF UDP/IP STACK ARCHITECTURES IN RECONFIGURABLE HARDWARE BASED ON THROUGHPUT, LATENCY AND FRAME LOSS RATE PERFORMANCE

AUTHOR: FERNANDO LUÍS HERRMANN
ADVISOR: JOÃO BAPTISTA DOS SANTOS MARTINS

Place and Date: Santa Maria, March 11th, 2010.

This work presents the implementation of three architectures of UDP/IP network stack in reconfigurable hardware. Also, presents the development of a Tester based on the RFC 2544 methodology and implemented it in FPGA. This Tester was used to obtain the throughput, latency and frame loss rate results. The performance of the project shows, in average, throughput results 89% better in comparison with a network stack implemented in software (PC) and running over a general purpose microprocessor, for frames with 64 bytes. Regarding latency, the project is 389 times lower for frames with 64 bytes and 13 times lower for frames with 1518 bytes, than the PC. On behalf of frame loss rate, the project doesn't loss frames for any frame sizes used during the tests, while the PC has presented a frame loss of almost 98% for frames with 64 bytes.

Keywords: Network UDP/IP stack; FPGA; RFC 2544; Throughput; Latency; Frame loss rate.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
BRAM	Block Random Access Memory
CMOS	Complementary Metal-Oxide Semiconductor
CRC	Cyclical Redundancy Check
DCM	Digital Clock Manager
DF	Bit Don't Fragment
DFS	Digital Frequency Synthesizer
DiffServ	Differentiated Services
DNS	Domain Name System
DUT	Device Under Test
ECN	Explicit Congestion Notification
EDK	Embedded Development Kit
FDDI	Fiber Distributed Data Interface
FIFO	First In First Out
FPGA	Field Programmable Gate Array
fps	Frames per Second
FSL	Fast Simplex Link
FTP	File Transfer Protocol
Gbps	Gigabit por segundo
GMII	Gigabit Media Independent Interface
GPP	General-Purpose Processor
HDL	Hardware Description Language
HTTP	Hypertext Transfer Protocol

I/O	Input/Output
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IFG	InterFrame Gap
IP	Internet Protocol
IPG	InterPacket Gap
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local area network
LSB	Least Significant Bit
LLC	Logical Link Control
LUT	Look-up table
MAC	Medium Access Control
Mbps	Megabit por segundo
MF	Bit More Fragments
MII	Media Independent Interface
MSB	Most Significant Bit
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIC	Network Interface Card
ns	nanosegundos
OSI	Open Systems Interconnection
PHY	Physical Layer
PLB	Processor Local Bus
PPC	PowerPC
RARP	Reverse Address Resolution Protocol
RFC	Request For Comments
RGMII	Reduced Gigabit Media Independent Interface
RTAI	Real Time Application Interface

RTP	Real-time Transport Protocol
SDK	Software Development Kit
SFD	Start Frame Sequence
SGMII	Serial Gigabit Media Independent Interface
TCP	Transmission Control Protocol
IETF	The Internet Engineering Task Force
TOE	TCP/IP Offload Engine
TTL	Time to live
UDPIPv1	Arquitetura da pilha de comunicação UDP/IP desenvolvida em Verilog Versão 1
UDPIPv2	Arquitetura da pilha de comunicação UDP/IP desenvolvida em Verilog Versão 2
UDPIPv3	Arquitetura da pilha de comunicação UDP/IP desenvolvida em Verilog Versão 3
us	microsegundos
VoIP	Voice Over Internet Protocol
XGMII	10 Gigabit Media Independent Interface
UDP	User Datagram Protocol

LISTA DE LISTAGENS

SUMÁRIO

1 INTRODUÇÃO

Atualmente, com a crescente popularidade da Internet, os usuários necessitam de uma largura de banda cada vez maior para transmitir a crescente quantidade de dados e prover serviços mais confiáveis e de alta qualidade (WEIDONG, 2003). Isso também colaborou para que surgissem novas formas de comunicação, mais acessíveis e com custo reduzido, como por exemplo, *Voice over IP* (VoIP) e videoconferência (DAVIDSON et al., 2006). Ao mesmo tempo, o desenvolvimento da tecnologia tem permitido que as velocidade de transmissão de dados alcancem 1, 10, 40 e até 100 Gbps (IEEE P802.3ba, 2009), fazendo com que o gargalo para redes de alta velocidade não mais esteja atrelado ao meio de transmissão e sim ao processamento dos dados da rede (YEH et al., 2002). Essa tarefa de processamento dos dados da rede cabe às funções da pilha de comunicação TCP/IP, ou mais especificamente UDP/IP no caso de comunicações VoIP e videoconferência (KUROSE; ROSS, 2009).

Tradicionalmente, as principais funções da pilha TCP/IP são executadas em *software* rodando sobre um microprocessador de propósito geral. Como a velocidade de transmissão vem aumentando drasticamente, o microprocessador acaba ficando sobrecarregado com a enorme quantidade de processamento requerido pelas funções da pilha TCP/IP. Dessa forma, existe uma urgente necessidade de retirar essas funções, ou parte destas, do microprocessador, a fim de conseguir suportar as altas taxas de transmissão de dados.

A retirada de parte do processamento da pilha TCP/IP do microprocessador de propósito geral também é conhecida como *TCP/IP Offload Engine* (TOE) (YEH et al., 2002). Um dispositivo bastante conhecido e utilizado para este procedimento são os dispositivos *Field Programmable Gate Array* (FPGAs), que são *hardwares* reconfiguráveis e geralmente são configurados utilizando *Hardware Description Language* (HDL). FPGAs podem fornecer alta velocidade de processamento bem como flexibilidade para adição de novos recursos no caso de implementações de *offload*[1] parciais para mais completas.

Esta dissertação aborda o estudo e a implementação de arquiteturas da pilha UDP/IP em

[1] *Offload* é uma técnica de diminuição da carga de processamento através da retirada das funções da pilha TCP/IP do microprocessador. Arquiteturas que utilizam essa técnica serão detalhadas na Seção 2.3.2.

uma placa dotada de dispositivo FPGA, afim de prover uma alta vazão de dados, baixa latência e baixa taxa de perda de quadros.

1.1 Objetivos

O principal objetivo deste trabalho é o desenvolvimento e implementação de arquiteturas *Gigabit* da pilha de comunicação UDP/IP em *Hardware Description Language* (HDL). A implementação é efetuada em um dispositivo FPGA dotado de interface *Gigabit Ethernet* e integrada com uma camada de aplicação com o intuito de prover uma maior vazão de dados, menor latência e menor taxa de perda de quadros do que uma implementação tradicional da pilha UDP/IP em *software*.

Essa implementação usa como base a tradicional arquitetura de pilha de comunicação TCP/IP conforme demonstrado na Figura 1(a). A pilha de comunicação no estilo proposto por este trabalho, com ênfase às camadas que são desenvolvidas (enlace (MAC), transporte (UDP) e rede (IPv4)), pode ser visualizada na Figura 1(b). Ainda na Figura 1(b), pode ser observado que a função da camada de transporte é encapsular e desencapsular apenas os pacotes de protocolo UDP. Outros protocolos, como por exemplo, TCP, são repassados diretamente para a camada de aplicação, ou seja, o datagrama destes pacotes não sofre influências da camada de transporte.

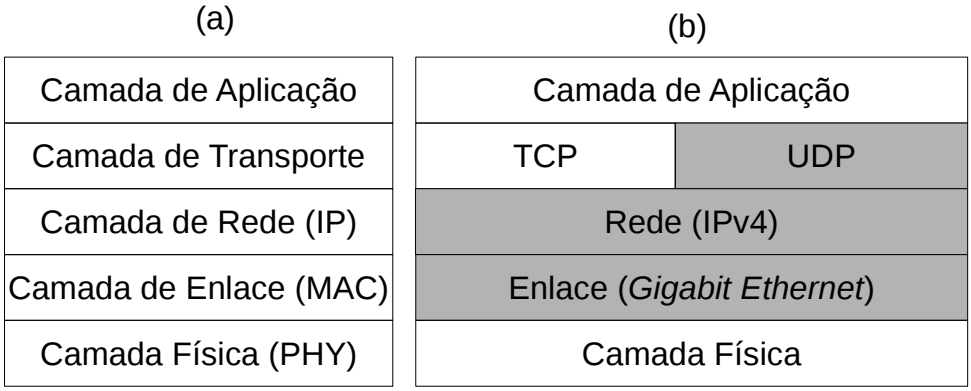


Figura 1: (a) Arquitetura tradicional da pilha de comunicação TCP/IP. (b) Arquitetura proposta por este trabalho e quais camadas são implementadas. Baseado em (TANENBAUM, 2002).

Por fim, também é objetivo deste trabalho realizar a implementação de um dispositivo que permita testar as arquiteturas da pilha de comunicação UDP/IP desenvolvidas, em relação à vazão, latência e taxa de perda de quadros. Esse dispositivo, que também é implementado em um FPGA, será baseado na metodologia e no conjunto de testes proposto pela RFC 2544.

1.2 Justificativa

Segundo (CLARK et al., 1989), (KAY; PASQUALE, 1996), (CLARK et al., 2002), (BINKERT et al., 2005) e (WU; CHEN, 2006), a pilha TCP/IP implementada no *software* dos sistemas operacionais, não consegue suportar de maneira apropriada uma alta vazão de dados, gastando assim vários ciclos de processamento para codificar e decodificar informações relativas aos protocolos de comunicação.

Atualmente com as taxas de transmissão chegando na faixa de 10 a 40 Gbps, microprocessadores de propósito geral não mais conseguem atender a demanda necessária de processamento, gerando assim um gargalo na transmissão de dados. Por exemplo, segundo (YEH et al., 2002) um processador de 20 GHz com utilização de 100% é necessário para suportar um *link Ethernet* de 10 Gbps.

Segundo (KAY; PASQUALE, 1996) e (GADELRAB, 2007) as principais causas da sobrecarga no processamento da pilha de comunicação estão relacionadas ao processamento dos protocolos, validação de *checksums*, cópia de dados, processos do Kernel do sistema operacional e gerenciamento de interrupções e *buffers*. Além disso, o sistema operacional e a placa de rede necessitam trocar um grande volume de dados e informações de controle por meio do barramento de entrada e saída, o que é outro fator importante para a sobrecarga no processamento da pilha de comunicação segundo (ORTIZ et al., 2009).

Para aferir o baixo desempenho de pilhas de comunicação quando implementadas no *software* dos sistemas operacionais, conforme argumentado pelos autores acima citados, este trabalho realizou os testes de vazão e taxa de perda de quadros num computador IBM *System* x3200 dotado de: CPU *Dual core* Intel Xeon 3040 1.86GHz, 1GB DDR2 667 MHz de memória RAM, interface de rede Broadcom NetXtreme *Gigabit Ethernet* e sistema operacional Linux Kernel 2.6.28-16-generic Ubuntu 9.04 x86_64 GNU/Linux. Os testes de vazão e taxa de perda de quadros são baseados na metodologia proposta pela RFC 2544 que estabelece um padrão para obtenção dos resultados relacionados ao desempenho de dispositivos de comunicação *Ethernet*.

O gráfico da Figura 2 apresenta os resultados de vazão para a tecnologia *Gigabit*, utilizando a arquitetura tradicional da pilha TCP/IP em *software* rodando num PC. No eixo X deste gráfico estão listados os tamanhos de quadros em *bytes* utilizados para a realização deste teste. E no eixo Y a vazão máxima *Gigabit* suportada em (%).

É possível observar, no gráfico da Figura 2, que o principal problema de uma pilha TCP/IP em *software* está relacionado com quadros que possuem um tamanho pequeno. Por exemplo, para quadros de 64, 128 e 256 *bytes*, a vazão máxima atingida pelo PC foi de apenas 11, 20 e 47

%, respectivamente, da capacidade total da tecnologia *Gigabit*. Esse problema se torna menos perceptível em quadros maiores, como por exemplo, quadros com tamanho a partir de 1024 *bytes* já atingem uma vazão igual ou superior a 95%. Isso se deve ao fato de que para transmitir a mesma quantidade de informação transmitida em quadros maiores são necessários mais quadros de menor tamanho, ocasionando, dessa forma, uma sobrecarga no microprocessador do PC que consequentemente não consegue atender à demanda de quadros que chegam pela interface de rede.

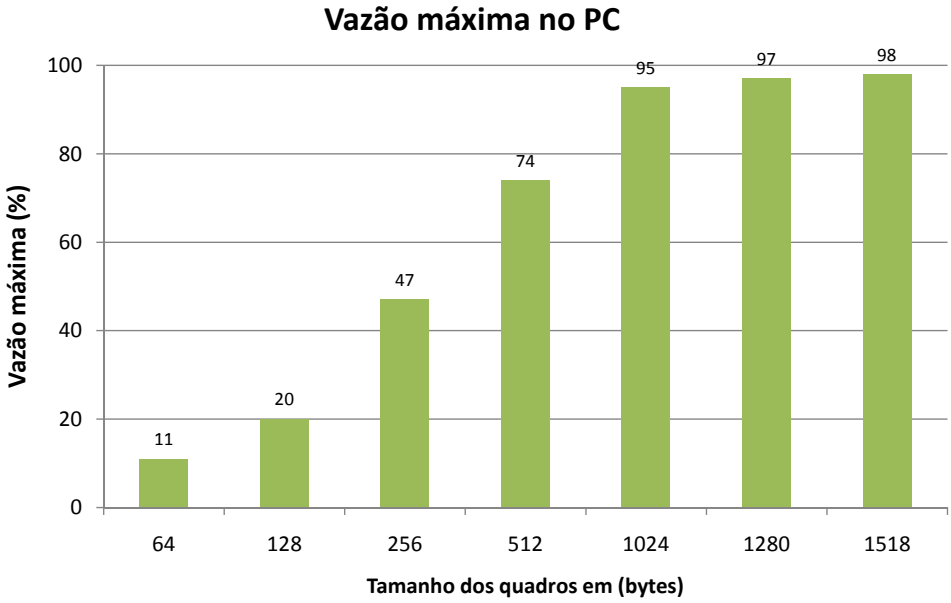


Figura 2: Vazão máxima *Gigabit* numa arquitetura tradicional da pilha TCP/IP em *software* rodando num PC.

Outro teste realizado foi em relação à taxa de perda de quadros e os resultados obtidos podem ser visualizados no gráfico da Figura 3. No eixo X deste gráfico estão listados os percentuais referentes à taxa de envio da vazão, com uma granularidade de 5%. O eixo Y exibe a taxa de perda de quadros também em (%). E cada curva do gráfico representa um tamanho de quadro em *bytes* para o qual o teste foi efetuado. Analisando o gráfico da Figura 3 é possível observar que a maior taxa de perda de quadros também está relacionada a quadros de tamanho menor. Por exemplo, quadros com 64 e 128 *bytes* apresentam taxas de perda de quadros acima de 80% mesmo para uma vazão de somente 55% da capacidade da tecnologia *Gigabit Ethernet*.

Considerando as análises expostas, a principal justificativa para este trabalho é a utilização de *hardware* dedicado para o processamento de informações referentes à pilha de comunicação TCP/IP, permitindo que parte do processamento, antes realizado pelo microprocessador de propósito geral, seja realizado por um *hardware* dedicado a este fim.

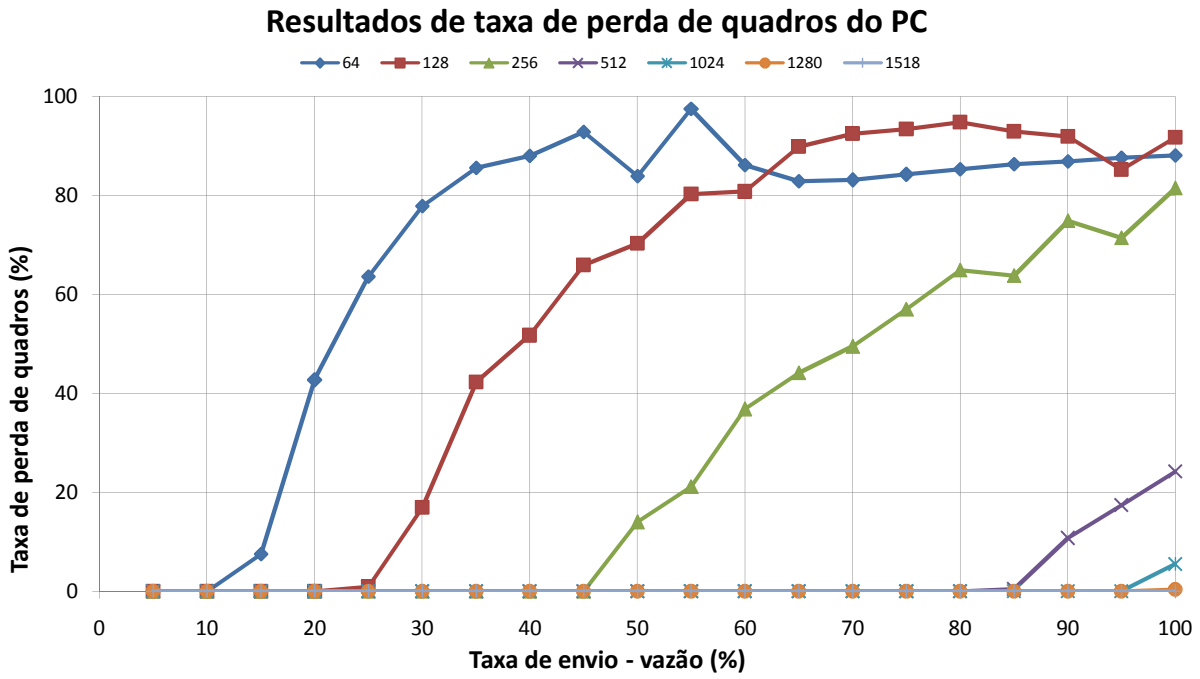


Figura 3: Taxa de perda de quadros numa arquitetura tradicional da pilha TCP/IP em *software* rodando num PC.

1.3 Organização

A dissertação encontra-se organizada conforme descrito a seguir.

O Capítulo 1 realiza uma Introdução ao tema, estabelece os Objetivos e a Justificativa para a realização deste trabalho.

A Revisão Bibliográfica sobre redes de computadores e pilha de comunicação TCP/IP é apresentada no Capítulo 2. Este capítulo também apresenta detalhadamente cada uma das camadas que compõem a pilha TCP/IP e como ocorre a comunicação entre elas. Também é apresentada a metodologia[2] existente que pode ser utilizada para a realização de testes de desempenho em dispositvos *Ethernet*. E ao final do Capítulo também são apresentadas arquiteturas e implementações existentes relacionados com este trabalho.

O Capítulo 3 apresenta a arquitetura e detalhes relacionados ao desenvolvimento do dispositivo Testador, bem como dos tipos de testes implementados: vazão, latência e taxa de perda de quadros, baseado na metodologia proposta pela RFC 2544. Este capítulo também apresenta as arquiteturas da pilha UDP/IP implementadas em *hardware* reconfigurável. E ao final é apresentada a metodologia e o ambiente utilizado no desenvolvimento e na validação tanto do Testador

[2]A metodologia utilizada para a realização de testes de desempenho em dispositivos *Ethernet* é um padrão estabelecido pela RFC 2544 e será detalhada na Seção 2.2.

quanto das pilhas de comunicação UDP/IP.

Os resultados de vazão, latência e taxa de perda de quadros obtidos através do Testador são apresentados no Capítulo 4. São comparadas as arquiteturas da pilha de comunicação UDP/IP em *hardware* reconfigurável e também a pilha TCP/IP rodando num computador com sistema operacional Linux. Ao final deste capítulo também é apresentada a utilização dos recursos do FPGA pelas arquiteturas da pilha de comunicação UDP/IP bem como uma comparação com outras arquiteturas e implementações existentes relacionados com este trabalho.

Por fim, o Capítulo 5 apresenta as conclusões desta dissertação, bem como propostas para trabalhos futuros.

O Apêndice A complementa a dissertação exibindo uma tabela contendo a taxa de quadros[3] e o tamanho do *InterFrame Gap*[4] (IFG) para a tecnologia *Gigabit Ethernet* com uma granularidade de 10% para todos os tamanhos de quadro sugerido pela RFC 2544. E no Apêndice B estão listados os artigos publicados que foram produzidos no decorrer do desenvolvimento deste trabalho.

[3] O termo taxa de quadros se refere a número de quadros por segundo e será detalhado na Seção 2.2.2 do Capítulo 2.

[4] O termo *InterFrame Gap* (IFG) é o tempo existente entre um quadro e outro e será detalhado na Seção 2.1.3.2 do Capítulo 2.

2 REVISÃO BIBLIOGRÁFICA

Este Capítulo apresenta uma revisão sobre diversos conceitos envolvidos em redes de computadores e pilhas de comunicação. É dado especial enfoque às camadas da pilha de comunicação e todos os conceitos envolvidos no desenvolvimento das arquiteturas implementadas por este trabalho. Também é apresentada a metodologia[1] existente que pode ser utilizada para a realização de testes de desempenho em dispositivos *Ethernet*. E ao final deste Capítulo também são apresentadas arquiteturas e implementações existentes relacionados com este trabalho.

2.1 Redes de computadores

Redes de computadores são estruturas físicas e lógicas que permitem que dois ou mais computadores possam compartilhar informações entre si, que podem ser: dados, impressoras, *e-mails*, etc. Quando um computador está conectado a uma rede de computadores, ele pode ter acesso às informações que chegam a ele e as informações presentes nos outros computadores ligados na mesma rede, o que permite um número muito maior de informações possíveis através daquele computador.

Para reduzir a complexidade, a comunicação em rede foi dividida em camadas. O objetivo de cada camada é oferecer determinados serviços às camadas superiores, isolando essas camadas dos detalhes de implementação desses recursos. A idéia fundamental é que um determinado item de *software* ou *hardware* forneça um serviço para seus usuários, mas mantenha oculto os detalhes de seu estado interno e de seus algoritmos. Estas camadas quando empilhadas formam uma suíte de protocolos de comunicação, ou em outras palavras, uma pilha de protocolos.

A pilha de protocolos padrão para comunicação na Internet é a TCP/IP (TANENBAUM, 2002) (KUROSE; ROSS, 2009) (STALLINGS, 2007). O nome TCP/IP é obtido dos dois protocolos fundamentais dessa pilha, *Internet Protocol* (IP) e *Transmission Control Protocol* (TCP). Todos os protocolos da pilha TCP/IP são definidos por documentos chamados de *Request For*

[1] A metodologia utilizada para a realização de testes de desempenho em dispositivos *Ethernet* é um padrão estabelecido pela RFC 2544 e será detalhada na Seção 2.2.

Comments (RFC) e podem ser encontrados em (RFC-EDITOR, 2009) ou (FAQS-RFC, 2010). A pilha de comunicação TCP/IP é apresentada na Figura 4(a), onde é possível observar as cinco camadas que a compõe: Aplicação, Transporte, Rede, Enlace e Física.

Na Figura 4(b) pode ser observado como cada camada da pilha TCP/IP realiza o processo de encapsulamento e desencapsulamento dos dados de saída e entrada, respectivamente. Esses dados são chamados de pacotes e geralmente possuem uma área de controle (cabeçalho) e uma área de dados. A área de dados carrega a informação relevante para a aplicação, e o cabeçalho carrega informações relevantes para a comunicação entre os computadores.

No encapsulamento ocorre a adição de informações dos protocolos em questão, geralmente um cabeçalho contendo estas informações, seguido pelo repasse destes dados encapsulados para a camada inferior. O envio de uma informação passa por quatro processos de encapsulamento, cada um responsável por um aspecto da comunicação, acrescentando sempre as informações necessárias para que a informação original chegue até seu destino, não se importando com o que está sendo transmitido na área de dados, pois a área de dados não é de responsabilidade do nível corrente mas do nível superior.

No desencapsulamento ocorre o processo inverso, onde o cabeçalho do protocolo é removido e os dados repassados à camada superior.

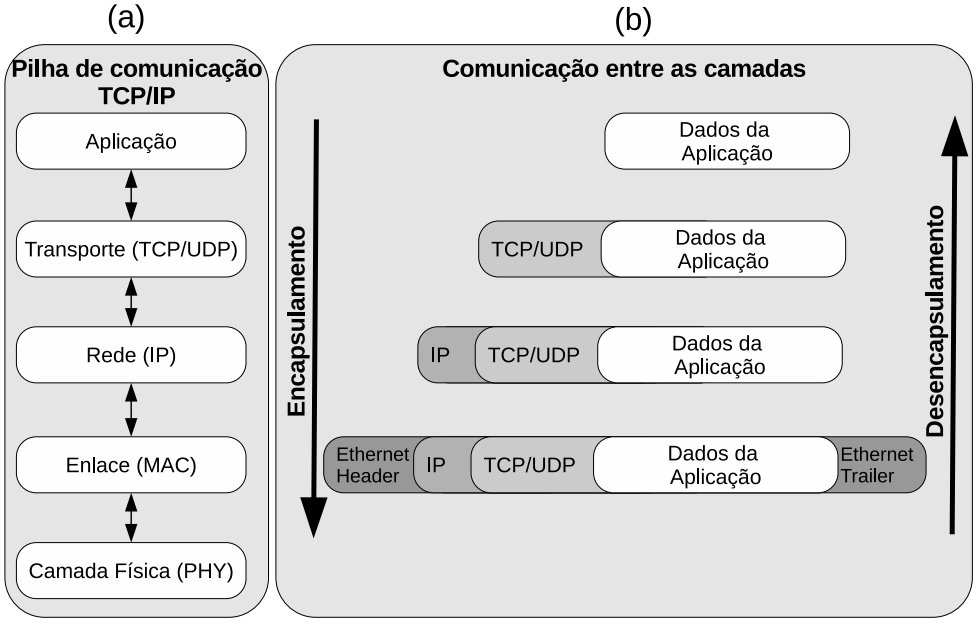


Figura 4: (a)Pilha de comunicação TCP/IP. (b) Comunicação entre as camadas e o processo de encapsulamento e desencapsulamento. Baseado em (TANENBAUM, 2002).

A camada de aplicação é responsável por realizar a comunicação entre o protocolo de transporte e os aplicativos, fornecendo uma interface direta com os aplicativos finais de usuário. Uma aplicação interage com os protocolos da camada de transporte para enviar ou receber dados.

Cada aplicação escolhe o tipo de protocolo de transporte, podendo ser TCP ou *User Datagram Protocol* (UDP). Alguns protocolos que representam essa camada são: *Hypertext Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP), *Domain Name System* (DNS).

A camada física ou *Physical Layer* (PHY) são os componentes de *hardware* envolvidos no processo de comunicação. Em termos de redes, a camada física diz respeito aos meios de conexão através dos quais os pacotes irão trafegar, tais como interfaces seriais, *switches*, cabos coaxiais ou ópticos. Na tecnologia *Gigabit Ethernet*, a camada física recebe o sinal através do cabo e o converte para um sinal digital onde o barramento de dados possui 8 *bits* que são fornecidos a cada ciclo de 125 MHz para a camada acima. No caminho inverso, a camada física recebe 8 *bits* da camada de enlace a cada ciclo de 125 MHz e os envia no formato analógico pelo cabo de rede.

As camadas de transporte, rede e enlace encontram-se explanadas nas subseções a seguir.

2.1.1 Camada de transporte

É a camada responsável pela comunicação dos dados de dois aplicativos que estão executando em diferentes computadores. Essa camada também é responsável pela transferência eficiente e confiável dos dados entre a máquina de origem e a máquina de destino, independente do tipo, topologia ou configuração das redes físicas existentes entre elas, garantindo ainda que os dados cheguem sem erros e na sequência correta (COMER, 2005).

Essa comunicação pode ser orientada à conexão ou não orientada à conexão. O serviço orientado à conexão visa prestar um serviço de transporte confiável, ocultando imperfeições do serviço de rede, de modo que a camada de aplicação possa simplesmente supor a existência de um fluxo de dados livre de erros. Já o serviço não orientado à conexão, é um serviço não confiável, pois a camada de transporte somente mapeia o pedido de transmissão de dados em pacotes para a transmissão pela camada de rede (DOSTÁLEK; KABELOVÁ, 2006).

O TCP é o protocolo mais utilizado na camada de transporte e oferece uma comunicação orientada à conexão. Outro protocolo da camada de transporte é o UDP que não é orientado à conexão. Entretanto, o protocolo UDP possui um desempenho melhor em aplicações de tempo real como áudio e vídeo, ou em aplicações que exijam baixa latência, e que não obrigatoriamente necessitem de um fluxo de dados livre de erros (LAM; LIEW, 2004).

2.1.1.1 Protocolo UDP

Uma área na qual o protocolo UDP é especialmente útil, é a de situações cliente/servidor. Com frequência, o cliente envia uma pequena solicitação ao servidor e espera uma pequena resposta de volta. Se a solicitação ou a resposta se perderem, o cliente simplesmente chegará a um tempo limite de espera (*timeout*) e tentará novamente (PARZIALE et al., 2006). Podem ser citadas, nessa área, aplicações como *Voice Over Internet Protocol* (VoIP), Videoconferência e outras aplicações que utilizem o protocolo *Real-time Transport Protocol* (RTP), ou seja, aplicações que necessitam de um fluxo contínuo de dados, especialmente aqueles que admitem perda ou corrompimento de parte de seu conteúdo (RODRIGUEZ; GATRELL; PESCHKE, 2001) (WANG; LIEW; LI, 2005).

O protocolo UDP está descrito na RFC 768 (POSTEL, 1980) e consiste de um cabeçalho de 8 *bytes* seguido pela carga útil (dados) proveniente da camada de aplicação, conforme pode ser observado na Figura 5. O cabeçalho é formado pelos campos: porta de origem, porta de destino, tamanho e *checksum*. Tanto a porta de origem quanto a porta de destino são campos de 16 *bits* e servem para identificar os pontos extremos nas máquinas de origem e destino. Quando um pacote UDP chega, sua carga útil é entregue ao processo associado a porta de destino. É por intermédio destas portas que a aplicação realiza o acesso à camada de transporte. As portas mais comumente utilizadas possuem números pré-definidos e a listagem completa das portas TCP ou UDP pode ser visualizada em (IANA, 2009).

	Protocolo UDP			
Bits	**0 – 7**	**8 – 15**	**16 – 23**	**24 – 31**
0	Porta de Origem		Porta de Destino	
32	Tamanho		Checksum	
64 +	Dados			

Figura 5: Formato do cabeçalho do protocolo UDP. Figura adaptada de (POSTEL, 1980).

Os campos tamanho e *checksum* também possuem 16 *bits* cada um. O primeiro representa o tamanho em *bytes* de todo o protocolo incluindo cabeçalho e dados. E o segundo é utilizado para detecção de erros. Do lado do remetente, o cabeçalho e os dados, são organizados em blocos de 16 *bits*. Esse blocos são somados e sobre o resultado é efetuado o complemento de um, que então é armazenado no campo *checksum* do protocolo UDP. Do lado do destinatário, o cálculo efetuado pelo remetente é refeito e o resultado é somado ao *checksum*. Se o resultado dessa soma for 'FFFF' em hexadecimal, então considera-se que não houve erros no pacote. A RFC 1071 apresenta detalhes referentes à implementação do *checksum* (BRADNER; BORMAN; PARTRIDGE, 1988).

Em todos os campos do cabeçalho o *bit* mais significativo, *Most Significant Bit* (MSB), é o bit mais alto. Por exemplo, o campo Porta de Origem deve ser lido ou gravado do *bit* 15 ao *bit* 0.

2.1.2 Camada de rede

A camada de rede está relacionada à transferência de pacotes da origem para o destino. Chegar ao destino pode exigir vários saltos em roteadores intermediários ao longo do percurso. Essa função contrasta claramente com a função da camada de enlace de dados, que tem o objetivo mais modesto de apenas mover quadros de uma extremidade de um fio até a outra. Portanto, a camada de rede é a camada mais baixa que lida com a transmissão fim a fim (STEVENS, 1994) (PARKER; SIYAN, 2002) (TANENBAUM, 2002).

A camada de rede é responsável por vários protocolos, dentre eles podem ser destacados os seguintes:

- *Internet Protocol* (IP): É por este protocolo que os dados são enviados de um computador para outro na Internet.
- *Internet Control Message Protocol* (ICMP): é um protocolo que fornece informações sobre o estado operacional da rede. O *software ping*, presente na maioria dos sistemas operacionais, é um exemplo de programa que utiliza pacotes do tipo ICMP.
- *Address Resolution Protocol* (ARP): é um protocolo utilizado para resolver o endereço *Medium Access Control* (MAC), também conhecido como endereço físico, de um determinado endereço IP.
- *Reverse Address Resolution Protocol* (RARP): é um protocolo utilizado para resolver o endereço IP de um determinado endereço MAC. É exatamente o inverso do ARP.

2.1.2.1 Protocolo IP

O protocolo IP é o protocolo mais importante da camada de rede e é por meio deste que os dados são enviados de um computador para outro na Internet. Cada computador na Internet possui um endereço IP que o identifica dentre os outros computadores. Quando a aplicação envia dados, por exemplo, um *e-mail*, a mensagem é dividida em pequenos pedaços chamados de pacotes ou datagramas. Cada um desses pacotes contém o endereço IP de origem e destino e são enviados primeiramente para o *gateway*. O *gateway* lê o endereço IP de destino e repassa

o pacote para um *gateway* adjacente. Esse processo se repete até que o pacote chegue no computador destino.

Como a mensagem é dividida em vários pacotes, cada pacote pode, se necessário, ser enviado por uma rota diferente através da Internet e com isso chegar fora de ordem até o seu destino. Como o protocolo IP apenas envia os pacotes através da Internet, cabe à camada de transporte cuidar do aspecto referente ao recebimento de pacotes fora de ordem.

A versão do protocolo IP mais utilizada atualmente é a *Internet Protocol Version 4* (IPv4). Entretanto, a versão *Internet Protocol Version 6* (IPv6) também já é suportada pelos sistemas operacionais. O protocolo IPv6 suporta endereços IP mais longos e com isso possibilita mais usuários na Internet. Contudo, esta Seção realizará apenas um estudo do protocolo IPv4 por se tratar do protocolo que será implementado em *hardware* reconfigurável neste trabalho.

<table>
<tr><th></th><th colspan="5">Protocolo IPv4</th></tr>
<tr><th>Bits</th><th>0 – 3</th><th>4 – 7</th><th>8 – 15</th><th>16 – 18</th><th>19 – 31</th></tr>
<tr><td>0</td><td>Versão</td><td>Tamanho do cabeçalho</td><td>Tipo de serviço</td><td colspan="2">Tamanho Total</td></tr>
<tr><td>32</td><td colspan="3">Identificação</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td>64</td><td colspan="2">Time to live</td><td>Protocolo</td><td colspan="2">Checksum do cabeçalho</td></tr>
<tr><td>96</td><td colspan="5">Endereço IP de origem</td></tr>
<tr><td>128</td><td colspan="5">Endereço IP de destino</td></tr>
<tr><td>160</td><td colspan="5">Opções</td></tr>
<tr><td>160 ou 192 +</td><td colspan="5">Dados</td></tr>
</table>

Figura 6: Formato do cabeçalho do protocolo IPv4. Figura adaptada de (POSTEL, 1981).

O protocolo IPv4 se encontra descrito na RFC 791, que pode ser encontrada em (POSTEL, 1981). A Figura 6 apresenta a estrutura e todos os campos que fazem parte do protocolo IPv4. Cada um destes campos encontra-se detalhado a seguir:

- Versão: Campo com 4 *bits* de tamanho que indica a versão do protocolo IP. Para o protocolo IPv4 esse campo é preenchido com o valor ’4’.

- Tamanho do cabeçalho: Campos com 4 *bits* de tamanho que indica quantas palavras de 32 *bits* (4 *bytes*) formam o cabeçalho. Um cabeçalho mínimo tem 20 *bytes* de tamanho, logo o valor mínimo em decimal neste campo é ’5’ se for considerado um pacote IP que não contenha um cabeçalho com o campo opções.

- Tipo de Serviço: Campo de 8 *bits* originalmente utilizado para especificar preferência de como os datagramas poderiam ser manuseados assim que circulassem pela rede. Por exemplo, um computador poderia definir o campo tipo de serviço dos datagramas IPv4 para preferir pequeno atraso, enquanto que outros poderiam preferir alta confiabilidade.

Na prática, esse campo não foi largamente implementado e por isso geralmente vem preenchidos com zeros. Porém, atualmente este campo é implementado como *Differentiated Services* (DiffServ) (NICHOLS et al., 1998) (BLAKE et al., 1998) (BABIARZ; CHAN; BAKER, 2006) e *Explicit Congestion Notification* (ECN) (RAMAKRISHNAN; FLOYD; BLACK, 2001) para prover qualidade de serviço e controle de congestionamento.

- Tamanho total: Campo de 16 *bits* que informa o tamanho de todo o datagrama IP, considerando cabeçalho e dados. O datagrama de tamanho mínimo possui 20 *bytes* e o máximo possui 65535 *bytes*.
- Identificação: Campo de 16 *bits* utilizado para identificar fragmentos do datagrama IP original.
- *Flags*: Campo de 3 *bits* que é utilizado para controlar ou identificar os fragmentos. Respectivamente esses 3 *bits* representam *Bit Reserved*, *Bit Don't Fragment* (DF), *Bit More Fragments* (MF).
- *Fragment Offset*: Este campo possui 13 *bits*, e permite que um receptor determine a posição de um fragmento em particular no datagrama IP original.
- *Time to live* (TTL): Possui 8 *bits* de tamanho e ajuda a prevenir que os datagramas persistam numa rede. Este campo representa o número máximo de saltos de um determinado pacote na rede. Cada roteador ou *switch* que um datagrama atravessa, o valor do TTL é decrementado em um. Quando o campo TTL chegar a zero, o pacote será descartado pelo roteador.
- Protocolo: Define o protocolo que é utilizado pela camada acima, neste caso, segundo o modelo TCP/IP é a camada de transporte, possui 8 *bits* de tamanho. Exemplos de protocolos: UDP ou TCP.
- *Checksum do cabeçalho*: Campo de 16 *bits* que armazena a verificação do cabeçalho. Este valor é ajustado ao longo do caminho e verificado a cada novo salto.
- Endereço IP de origem: Campo de 32 *bits* que armazena o endereço IP de origem.
- Endereço IP de destino: Campo de 32 *bits* que armazena o endereço IP de destino.
- Opções: Campo opcional do cabeçalho. Pode ser utilizado para superar possíveis deficiências do protocolo ou implementar campos que não são padrão para todas as redes. As principais finalidades são as passagens de parâmetros como: nível de segurança da informação em tráfego, depuração e testes de algoritmos de roteamento e escolha arbitrária de rotas para o datagrama seguir.

- Dados: São os dados provenientes da camada de transporte.

Em todos os campos do cabeçalho o MSB é o *bit* mais alto, exatamente como acontece no protocolo UDP. Por exemplo, o campo tamanho total deve ser lido ou gravado do *bit* 31 ao *bit* 16.

2.1.3 Camada de enlace de dados

Essa camada também é conhecida como MAC, por se tratar do protocolo mais utilizado para passar pacotes da camada de rede de um dispositivo para a camada de rede de outro dispositivo. Esses dispositivos irão executar as funções da camada de enlace de dados como adicionar um cabeçalho e um *trailer* ao pacote para prepará-lo para transmissão, e então de fato transmitir o quadro ou *frame* através da camada física. Do outro lado, a camada de enlace irá receber os quadros de dados, retirar o cabeçalho e *trailer* adicionados e encaminhá-los para a camada de rede. O *trailer* consiste geralmente de um *Cyclic Redundancy Check* (CRC) que é utilizado para verificar se o quadro recebido é válido, maiores informações sobre o CRC podem ser verificados na Seção 2.1.3.1.

Atualmente a *Ethernet*, padronizada pelo *Institute of Electrical and Electronics Engineers* (IEEE) 802.3 (IEEE, 2008) e formada pelas subcamadas *Logical Link Control* (LLC) e MAC, é a tecnologia mais amplamente utilizada para conexão de redes locais - *Local area network* (LAN). Este padrão define cabeamento e sinais elétricos para a camada física, e formato de quadros e protocolos para a camada MAC (SPURGEON, 2000).

2.1.3.1 Quadros *Ethernet*

O formato de quadro *Ethernet* mais utilizado na Internet é o *Ethernet* II que foi criado em 1978 pelas empresas Xerox, Intel e Digital, e por isso também é conhecido como DIX *Ethernet* (PARZIALE et al., 2006). A Figura 7 apresenta o formato do quadro DIX Ethernet que é formado pelos seguintes campos: Preâmbulo, *Start Frame Sequence* SFD, Endereço MAC de destino, Endereço MAC de origem, *EtherType*, Dados e *Cyclical Redundancy Check* (CRC) *Checksum*.

O preâmbulo é um valor fixo composto por 7 *bytes* cada um com o valor '10101010' em binário. Esse campo é utilizado pelo receptor para realizar a sincronização entre os dois dispositivos envolvidos na transmissão. O campo *Start Frame Sequence* (SFD) é transmitido imediatamente após o preâmbulo, indica o começo do quadro e possui 1 *byte* de tamanho. O valor que deve ser enviado é a sequência '10101011' em binário.

Os campos endereço MAC de destino e endereço MAC de origem possuem ambos 6 *bytes* de tamanho. O endereço MAC também é conhecido como endereço físico e deve ser único para cada dispositivo na Internet.

Quadro DIX Ethernet	
Bytes	**Campo**
7	Preâmbulo
1	SFD
6	Endereço MAC de destino
6	Endereço MAC de origem
2	EtherType
46 – 1500	Dados
4	CRC Checksum

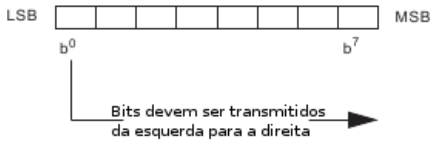


Figura 7: Formato padrão do quadro DIX *Ethernet*. Figura adaptada de (IEEE, 2008).

O campo *EtherType* possui 2 *bytes* de tamanho e indica o protocolo associado à camada de rede do quadro. Por exemplo, se o protocolo associado for IPv4, então o valor desse campo será '0800' em hexadecimal e se o protocolo associado for ARP, então o valor desse campo será '0806' em hexadecimal. A lista completa desses protocolos pode ser verificada em (IEEE, 2010).

O campo dados possui tamanho mínimo de 46 *bytes* e contém os dados do quadro oriundos da camada de rede. O tamanho mínimo é fixado em 46 *bytes* de dados, pois o tamanho mínimo de um quadro *Ethernet* é de 64 *bytes*. Os outros 18 *bytes* para completar os 64 *bytes* do tamanho mínimo do quadro são: 12 *bytes* do endereço MAC destino e origem, 2 *bytes* do *EtherType* e 4 *bytes* do CRC. O tamanho máximo de 1500 *bytes* exibido na Figura 7 pode variar de acordo com o *Maximum Transmission Unit* (MTU) suportado pelos dispositivos envolvidos na comunicação (MOGUL; DEERING, 1990).

O último campo do quadro DIX *Ethernet* apresentado na Figura 7 é o *Cyclical Redundancy Check* (CRC) *Checksum*. Esse campo contém um CRC de 32 *bits* e é utilizado na detecção de erros decorrentes de algum erro na transmissão do quadro. O valor deste campo é calculado por uma função que abrange desde o campo Endereço MAC de destino até o campo dados do quadro *Ethernet*. É definido pelo polinômio gerador com grau de 32 *bits* ($m = 32$) apresentado na Equação 2.1.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1 \quad (2.1)$$

$$P = \{p_{32}, p_{31}, ..., p_1, p_0\} = \{10000010011000001000111010110111\} \quad (2.2)$$

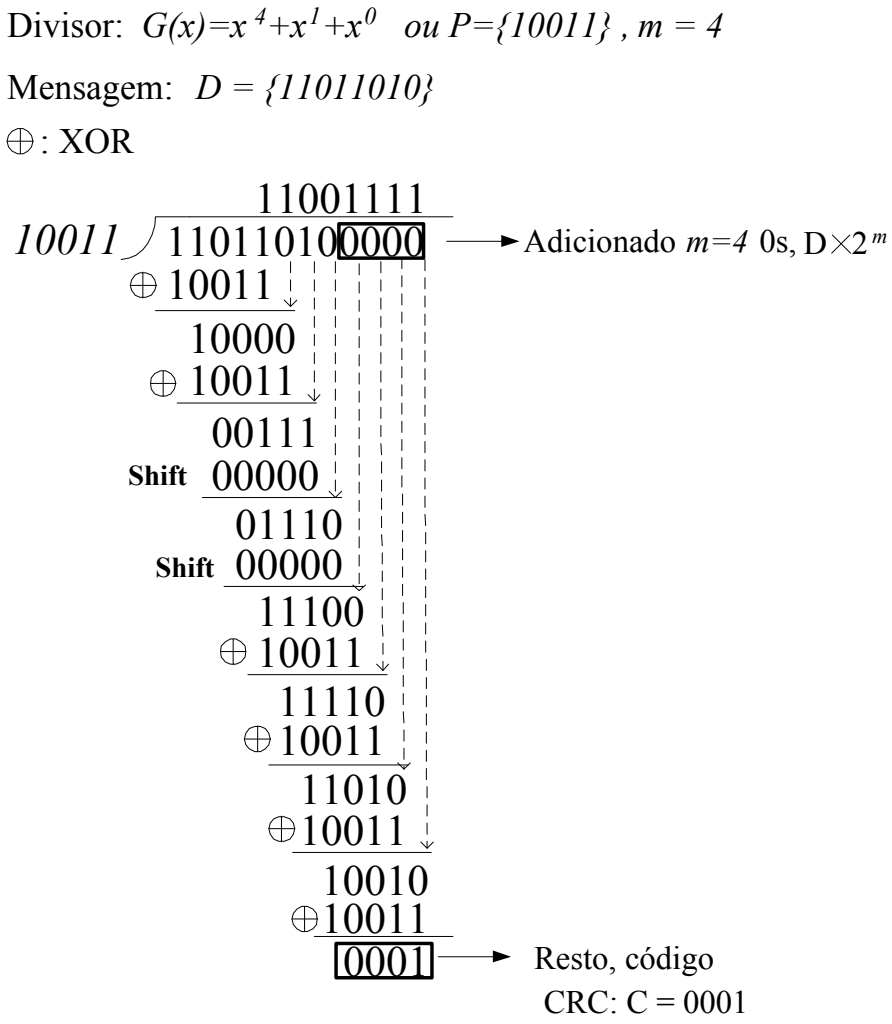
O polinômio gerador do CRC, *G(x)*, também pode ser representado na forma binária, como demonstra a Equação 2.2. O *bit* mais significativo do gerador P, p_{32}, corresponde ao coeficiente x^{32} do polinômio *G(x)* apresentado na Equação 2.1. Da mesma forma, p_{31} corresponde ao coeficiente x^{31}, que neste caso possui o valor '0', e assim por diante para os outros *bits*.

Assumindo que a mensagem constante no quadro seja representado por D, e o CRC representado por C com o tamanho m, então a mensagem a ser transmitida com o CRC calculado pode ser representada pela função exibida na Equação 2.3. A mensagem, $D \times 2^m$, é dividida pelo gerador P utilizando aritmética de módulo 2, que é realizada utilizando *shifts* e ou exclusivos (XORs).

$$T = \{DC\} = D \times 2^m + C \quad (2.3)$$

Um exemplo de cálculo de CRC considerando um polinônimo gerador de 4 *bits* pode ser observado na Figura 8. Neste exemplo, a mensagem a ser transmitida é: $D = \{11011010\}$. O polinômio gerador é $G(x) = x^4 + x^1 + x^0$, ou na forma binária, gerador $P = \{10011\}$. E $m = 4$ é o grau do polinômio gerador. No começo, 4 0s (zeros) são adicionados na mensagem a ser transmitida. Na sequência, a mensagem é dividida pelo polinômio gerador $P = \{10011\}$. O resto '1000' é obtido aplicando um XOR sobre '11011' utilizando o divisor $P = \{10011\}$. O próximo *bit* da mensagem é concatenado ao resto, e outro XOR é aplicado utilizando o divisor P. Quando o bit mais à esquerda do resto for '0', a operação *shift* será executada. Esse cálculo prossegue até que todos os bits da mensagem tenham sido calculados, o resto da divisão será o valor do CRC calculado, que para este exemplo é $C = 0001$. O CRC é então adicionado ao final da mensagem formando assim os dados que serão transmitidos para a rede, $T = \{DC\} = D \times 2^m + C = 110110100001$. Se algum erro ocorrer na transmissão do quadro T, o dispositivo responsável pelo recebimento do quadro não conseguirá calcular o mesmo CRC que foi enviado.

Em todos os campos do cabeçalho do quadro *Ethernet* o MSB é o *bit* mais alto, exatamente como acontece nos outros protocolos já exibidos. Porém, diferentemente das outras camadas da pilha TCP/IP, na camada MAC o *byte* é transmitido na ordem inversa, ou seja, do *Least Significant Bit* (LSB) para o MSB, exatamente como é exibido na parte inferior da Figura 7.



A mensagem protegida pelo código CRC:

$$T = D \times 2^m + C = \underbrace{110110100001}_{\text{Mensagem}\quad\text{CRC}}$$

Figura 8: Exemplo de cálculo de CRC considerando um polinômio gerador de 4 *bits* (WEIDONG, 2003).

Por exemplo, o campo Endereço MAC de destino possui 48 *bits* e o acesso deve ser feito do *bit* 47 ao 0. Mas ao realizar o envio do *byte*, 47 ao 40 por exemplo, o *bit* mais significativo será o 40.

Um quadro DIX *Ethernet* completo de 64 *bytes* pode ser visualizado na Figura 9. Essa Figura exibe um quadro com o preâmbulo, SFD, cabeçalho MAC, cabeçalho IPv4, cabeçalho UDP, Dados e CRC *checksum*. Todos os valores informados nos campos do quadro *Ethernet* II aparecem em hexadecimal.

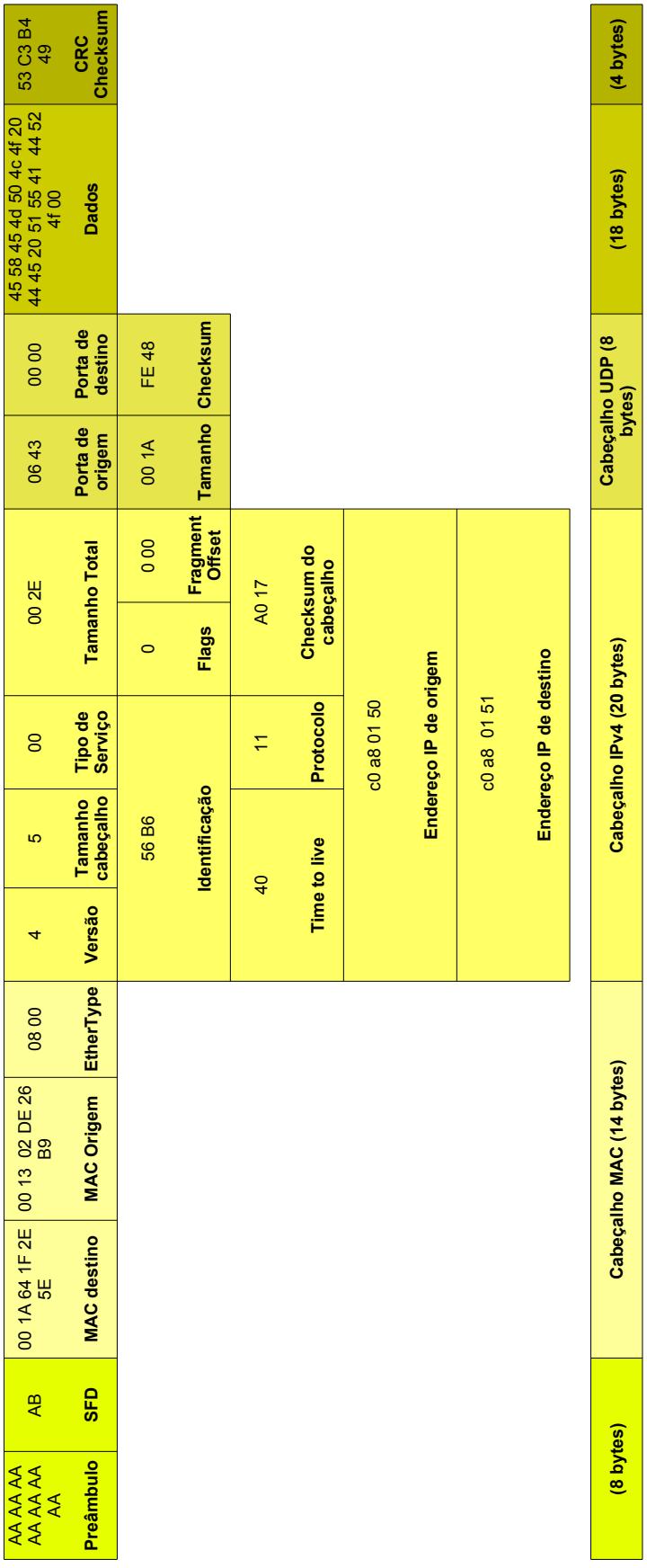


Figura 9: Exemplo de quadro *Ethernet* II (DIX) com 64 bytes de tamanho.

2.1.3.2 *InterFrame Gap*

Logo após a transmissão de um quadro, o dispositivo responsável pelo envio deve aguardar um tempo mínimo para realizar o envio de um novo quadro. Esse procedimento é demonstrado na Figura 10, onde são transmitidos 10 quadros de diferentes tamanhos e entre um quadro e outro existe um tempo mínimo conhecido como *InterFrame Gap* (IFG) ou *InterPacket Gap* (IPG). O valor mínimo do IFG é de 12 *bytes*, ou seja, o dispositivo responsável pelo envio deve aguardar o tempo necessário para realizar o envio de 12 *bytes* para a rede. Conforme pode ser observado a seguir, esse tempo difere dependendo da tecnologia que está sendo utilizada:

- 9.600 ns para 10 Mbps *Ethernet*.
- 960 ns para 100 Mbps (*fast*) *Ethernet*.
- 96 ns para 1 Gbps (*gigabit*) *Ethernet*.
- 9,6 ns para 10 Gbps (10 *gigabit*) *Ethernet*.

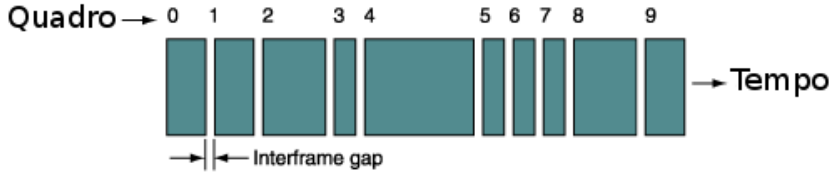


Figura 10: Exemplo de *InterFrame Gap*.

2.1.3.3 Interfaces MAC - PHY

São interfaces implementadas na PHY responsáveis por realizar a comunicação entre a camada MAC e a camada PHY. São exemplos de interfaces MAC: *Media Independent Interface* (MII), *Gigabit Media Independent Interface* (GMII), *Reduced Gigabit Media Independent Interface* (RGMII), *Serial Gigabit Media Independent Interface* (SGMII) e 10 *Gigabit Media Independent Interface* (XGMII).

A Figura 11 exibe o relacionamento das interfaces MAC, MII e GMII, com o modelo de referência *Open Systems Interconnection* (OSI) e o padrão IEEE 802.3 (IEEE, 2008). Na Figura 11(a) é possível observar as camadas que compõem o modelo de referência OSI e na Figura 11(b) as sub-camadas que compõem a camada de enlace (data link) e a camada PHY. Ainda na Figura 11(b), na área demarcada, é possível verificar onde as interfaces MII e GMII estão implementadas dentro da PHY.

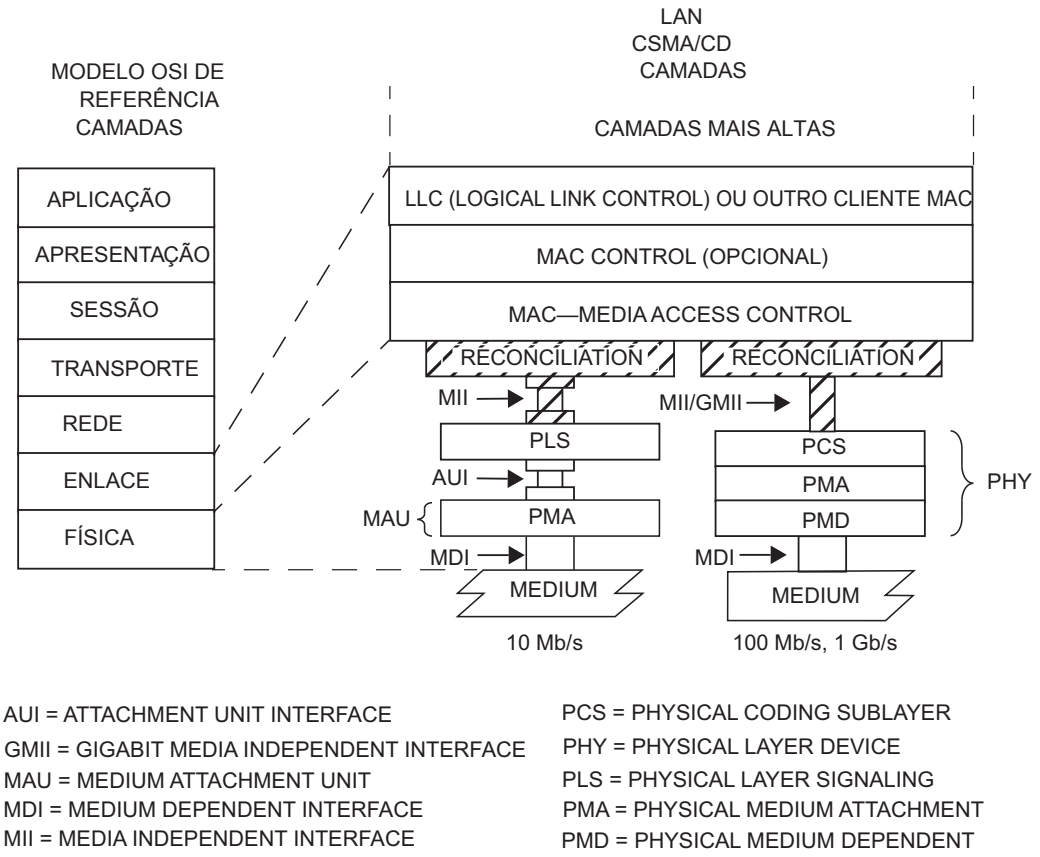


Figura 11: Relacionamento das interfaces MAC, MII e GMII, com o modelo de referência OSI e o padrão IEEE 802.3 (IEEE, 2008).

Para comunicação *Gigabit*, por exemplo, pode ser utilizada a interface GMII que está definida em (IEEE, 2008). Esta interface define velocidades de até 1000 Mbps, implementada usando 8 *bits* de dados a um *clock* de 125 MHz. Essa interface é compatível com MII e portanto opera também a 10/100 Mbps.

A interface GMII possui os seguintes pinos físicos que devem ser utilizados para a realização do envio do quadro gerado pela camada MAC:

- GTXCLK: Sinal de *clock* para *Gigabit* (125 MHz).
- TXCLK: Sinal de *clock* para 10/100 Mbps (2,5 MHz para 10 Mbps e 25 MHz para 100 Mbps).
- TXD[7:0]: Barramento de 8 *bits* de dados.
- TXEN: Sinal que habilita a transmissão.
- TXER: Sinal utilizado para forçar um erro na transmissão do quadro.

Para envio dos dados existem dois sinais de *clock* que dependem da configuração da PHY,

um sinal de *clock* é utilizado para conexões *Gigabit* e o outro sinal de *clock* é utilizado para conexões de 10/100 Mbps. Para velocidade *Gigabit*, o sistema deve fornecer o *clock* GTXCLK para a PHY e sincronizar a este os sinais TXD, TXEN e TXER. Para velocidades 10/100 Mbps o TXCLK é fornecido pela PHY e usado para sincronizar os sinais. O *clock* RXCLK para recebimento é mais simples, pois existe somente um pino que é fornecido pela PHY com os sinais de recebimento já sincronizados. Os sinais de *clock*, GTCLK e RXCLK, não são síncronos.

Tabela 1: *Bits* de definição do registrador 0, *Control*, da interface de gerenciamento GMII (IEEE, 2008).

Bit(s)	**Nome**	**Descrição**	**R/W (*)**
15	*Reset*	1 = PHY *reset* 0 = operação normal	R/W SC
14	*Loopback*	1 = habilitar modo *loopback* 0 = desabilitar modo *loopback*	R/W
13	*Speed Selection* (LSB)	6 13 1 1 = reservado 1 0 = 1000 Mbps 0 1 = 100 Mbps 0 0 = 10 Mbps	R/W
12	*Auto-Negotiation Enable*	1 = habilitar processo de auto-negociação 0 = desabilitar processo de auto-negociação	R/W
11	*Power Down*	1 = *power down* 0 = operacão normal (**)	R/W
10	*Isolate*	1 = isolamento elétrico da PHY para MII ou GMII 0 = operação normal (**)	R/W
9	*Restart Auto-Negotiation*	1 = reinicia processo de auto-negociação 0 = operação normal	R/W SC
8	*Duplex Mode*	1 = *full-duplex* 0 = *half duplex*	R/W
7	*Collision Test*	1 = habilitar sinal de teste COL 0 = desabilitar sinal de teste COL	R/W
6	*Speed Selection* (MSB)	6 13 1 1 = reservado 1 0 = 1000 Mbps 0 1 = 100 Mbps 0 0 = 10 Mbps	R/W
5	*Unidirectional enable*	Quando *bit* 12 é 1 ou o *bit* 8 é 0, este sinal é ignorado Quando *bit* 12 é 0 ou o *bit* 8 é 1: 1 = habilitar transmissão mesmo não existindo conexão válida. 0 = habilitar transmissão somente se existir uma conexão válida	R/W
4:0	*Reserved*	Escrito como 0, leitura ignorada	R/W
(*) R/W = Leitura/Escrita, SC = *Self-Clearing*, setado automaticamente para 0.			
(**) Para operação normal, os *bits* 10 e 11 devem estar em 0.			

Pinos para recebimento da interface GMII:

- RXCLK: Sinal de *clock* para recebimento.
- RXD[7:0]: Barramento de 8 *bits* de dados.

- RXDV: Indica que o barramento *RXD* possui dado válido.
- RXER: Erro nos dados.
- COL: Colisão, utilizado somente para conexões *half-duplex*.
- CRS: *Carrier Sense*, utilizado somente para conexões *half-duplex*.

Pinos de gerenciamento interface GMII:

- MDC: Sinal de *clock* da interface de gerenciamento. Segundo (IEEE, 2008), o sinal fornecido para este pino deve possuir período de *clock* entre 400 ns e 160 ns, ou seja, entre 2,5 MHz e 6,25 MHz.
- MDIO: Sinal de *Input/Output* (I/O) para gerenciamento.

A interface de gerenciamento controla o funcionamento da PHY. Existem 32 registradores e cada um contém 16 *bits* de tamanho. Os primeiros 16 registradores estão definidos em (IEEE, 2008) e os outros são específicos de cada dispositivo. O primeiro registrador, *Control*, é utilizado para configurar o dispositivo, por exemplo, para operar em *gigabit*, *full-duplex* ou *half duplex*. A Tabela 1 apresenta todas as possíveis configurações do primeiro registrador.

2.2 *Request For Comments* (RFC) 2544

As *Request For Comments* (RFC) são um conjunto de documentos de referência que descrevem, especificam, estabelecem padrões e debatem a maioria das normas e padrões, de tecnologias e protocolos ligados à Internet e às redes em geral. A pilha de protocolos TCP/IP, por exemplo, é representada por um conjunto de RFCs estabelecidos pelo órgão *The Internet Engineering Task Force* (IETF) (IETF, 2010). Este órgão publica oficialmente as RFCs, que também podem ser encontrados em (RFC-EDITOR, 2009) ou (FAQS-RFC, 2010).

Além de estabelecer normas e padrões, as RFCs também podem propor uma metodologia, esse é o caso da RFC 2544 que define uma metodologia de testes para dispositivos *Ethernet* (BRADNER; MCQUAID, 1999). E, juntamente com a RFC 1242 (BRADNER, 1991), que define a maioria dos termos que são utilizados pela RFC 2544, estabelece um conjunto de testes que pode ser utilizado para descrever e reportar as características de desempenho de um dispositivo *Ethernet*.

Esse conjunto de testes é usualmente implementado num dispositivo chamado de Testador, com portas para envio e recebimento de quadros. O Testador está conectado a um Dispositivo

sob Teste, ou do inglês *Device Under Test* (DUT), ou seja, ao dispositivo que está sendo testado. A Figura 12 apresenta os três cenários de teste sugeridos pela RC 2544. O cenário mais usual é o cenário apresentado na Figura 12(a) em que o Testador envia quadros por uma porta do dispositivo e coleta os quadros para análise dos resultados em outra porta do mesmo dispositivo, ou simplesmete envia e coleta os quadros para análise dos resultados utilizando a mesma porta do dispositivo.

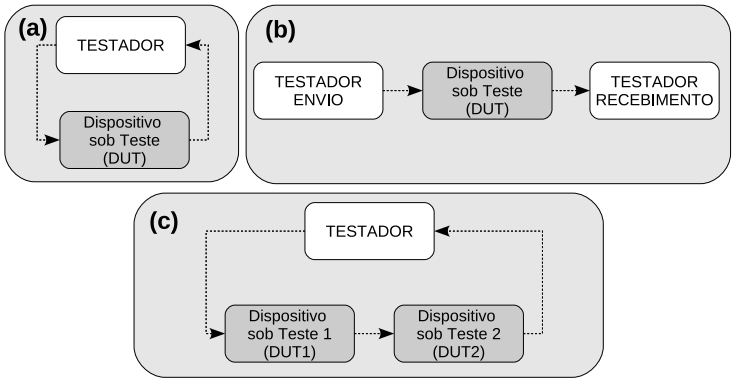


Figura 12: Cenários de teste propostos pela RFC 2544. Figura adaptada de (BRADNER; MCQUAID, 1999).

A Figura 12(b) apresenta o segundo cenário composto por dois Testadores, um que envia os quadros (Testador Envio), e outro que coleta e reporta os resultados obtidos (Testador Recebimento). Este caso é o mais indicado quando se trata de dispositivos *Ethernet* já instalados e com pontos para envio e recebimento separados impedindo o uso de um único Testador.

E o terceiro cenário, apresentado na Figura 12(c), consiste de múltiplos DUTs ligados serialmente. Este é o caso que mais se aproxima das redes existentes no mundo real. Por exemplo, clientes numa *Local area network* (LAN) que interagem com um servidor num *backbone Fiber Distributed Data Interface* (FDDI).

Além dos cenários de teste apresentados, a RFC 2544 também define o formato, o tamanho e a taxa de envio dos quadros de teste. As Seções a seguir apresentam estes termos e também o conjunto de testes que pode ser executado para medir o desempenho de um dispositivo *Ethernet*.

2.2.1 Formato e tamanho dos quadros de teste

Para realizar o conjunto de testes proposto pela RFC 2544 são utilizados quadros do tipo UDP *echo request* e o formato destes é apresentado na Tabela 2.

A Tabela 3 apresenta os sete tamanhos de quadros em *bytes* sugeridos pela RFC 2544 para testar dispositivos *Ethernet*, são eles: 64, 128, 256, 512, 1024, 1280 e 1518. Nessa tabela também é possível visualizar os valores que devem ser utilizados nos campos tamanho total do

IP e tamanho total do UDP dependendo do tamanho do quadro que está sendo testado. Esses dois campos podem ser visualizados na Tabela 2.

Tabela 2: Formato do quadro de teste utilizado para realizar o conjunto de testes da RFC 2544.

Byte	**Dados em Hexadecimal**	**Descrição**
Cabeçalho do quadro MAC		
00	xx xx xx xx xx xx	Endereço MAC de destino (*)
06	xx xx xx xx xx xx	Endereço MAC de origem (*)
12	08 00	*EtherType*
Cabeçalho do protocolo IP		
14	45	Versão do protocolo IP: 4. Tamanho do cabeçalho: 5
15	00	Tipo de serviço
16	00 2E	Tamanho total (**)
18	00 00	Identificação
20	00 00	*Flags* (3 bits): 0. *Fragment Offset*: 0
22	0A	TTL
23	11	Protocolo: 11 (UDP)
24	C4 8D	*Checksum* do cabeçalho (**)
26	xx xx xx xx	Endereço IP de origem (*)
30	xx xx xx xx	Endereço IP de destino (*)
Cabeçalho do protocolo UDP		
34	C0 20	Porta de origem
36	00 07	Porta de destino: 07 = *Echo Request*
38	00 1A	Tamanho total (**)
40	00 00	*Checksum* do cabeçalho
Dados do protocolo UDP		
42	00 01 02 03 04 05 06 07	Dados (***)
50	08 09 0A 0B 0C 0D 0E 0F	
(*) - Deve ser alterado levando em consideração os endereços do Testador e do DUT		
(**) - Deve ser alterado para quadros com tamanhos diferentes. Verificar Tabela 3		
(***) - Deve ser preenchido com dados até atingir o tamanho do frame		

Tabela 3: Tamanhos dos quadros de teste e valores referentes aos protocolos IP e UDP.

***Tamanho do Quadro* (*Bytes*)**	**Tamanho total do IP (Hexadecimal)**	**Tamanho total do UDP (Hexadecimal)**
64	00 2E	00 1A
128	00 6E	00 5A
256	00 EE	00 9A
512	01 EE	01 9A
1024	03 EE	03 9A
1280	04 EE	04 9A
1518	05 DC	05 C8

Para calcular o tamanho do quadro a RFC 2544 leva em consideração a estrutura exibida

na Figura 13. Esta Figura apresenta um quadro de 64 *bytes* que é composto por: Cabeçalho MAC (14 *bytes*), Cabeçalho IP (20 *bytes*), Cabeçalho UDP (8 *bytes*), Dados (18 *bytes*) e CRC (4 *bytes*). Para outros tamanhos de quadros a estrutura permanece a mesma, o campo Dados é o único que sofrerá alterações em seu tamanho.

O formato dos quadros de teste apresentado nessa Seção não é obrigatório, mas é essencial que esteja descrito no relatório de resultados do DUT.

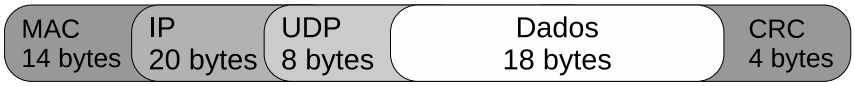


Figura 13: Estrutura do quadro de 64 *bytes* segundo a RFC 2544.

2.2.2 Taxa de quadros - *Frame rate*

O termo *frame rate* é utilizado para definir qual a taxa de quadros que um determinado dispositivo consegue processar (CLARK; HAMILTON, 1999). Essa taxa é expressa em *Frames per Second* (fps) e a taxa máxima de quadros por segundo que uma determinada tecnologia consegue suportar é denominada *Maximum Frame Rate* e está expressa na Equação 2.4.

$$Maximum\ Frame\ Rate\ (fps) = \frac{Ethernet\ Speed\ /\ 8\ bits}{Preambulo\ +\ Tamanho\ quadro\ +\ IFG\ minimo} \tag{2.4}$$

Analisando as seguintes considerações:

- *Ethernet Speed*: 1000 *Megabits* por segundo (Mbps) ou 1 *Gigabit* por segundo (Gbps).
- Preâmbulo: 8 *bytes*. Considerando o preâmbulo (7 *bytes*) mais o SFD (1 *byte*).
- Tamanho do quadro: 64 *bytes*.
- IFG mínimo: 12 *bytes*.

Então é possível concluir que a *Ethernet* de 1 Gbps consegue suportar 1.488.095 fps se forem considerados quadros de 64 *bytes* com o IFG mínimo de 12 *bytes*. A Tabela 4 apresenta os valores da quantidade máxima de quadros para outros tamanhos de quadros e outras tecnologias *Ethernet*.

Os valores da quantidade máxima de quadros apresentados até agora nesta Seção são referentes a utilização de 100% da capacidade da tecnologia em questão. Para verificar, por

exemplo, quantos quadros são necessários para a utilização de 30% da capacidade do dispositivo, basta adicionar um percentual na Equação 2.4. Um dispositivo *Ethernet Gigabit* com utilização de 30% é capaz de processar 446.429 fps se forem considerados quadros de 64 *bytes*. Neste caso o tempo entre um quadro e outro (IFG) deixa de ser somente 12 *bytes*, uma vez que, menos informação está sendo transmitida.

Tabela 4: *Maximum Frame Rate*

***Tamanho do Quadro* (*Bytes*)**	**10 Mbits *Ethernet* (fps)**	***Fast Ethernet* (100 Mbits) (fps)**	***Gigabit Ethernet* (1 Gbps) (fps)**
64	14.880	148.809	1.488.095
128	8.445	84.459	844.594
256	4.528	45.289	452.898
512	2.349	23.496	234.962
1024	1.197	11.973	119.731
1280	961	9.615	96.153
1518	812	8.127	81.274

Taxas menores que 100% apresentam um IFG maior. E a forma para calcular o tempo do IFG se encontra detalhada nos três passos seguintes:

Primeiro passo:
O primeiro passo é calcular o tempo de envio de todos o quadros, expresso na Equação 2.5. Por exemplo, considerando os seguintes parâmetros para a taxa de quadros de 30%:

- Preâmbulo: 8 *bytes*. Considerando o preâmbulo (7 *bytes*) mais o SFD (1 *byte*)
- Tamanho do quadro: 64 *bytes*.
- Ciclos *byte*: É a quantidade de ciclos necessária para enviar 1 *byte*. Para *Ethernet* 10/100 Mbps esse tempo é '2', pois a cada ciclo essa tecnologia envia 4 *bits*. Para *Gigabit Ethernet* esse tempo é '1', pois a cada ciclo essa tecnologia envia 8 *bits*. Então o valor desse parâmetro para este exemplo é '1'.
- Período *clock*: O período de *clock* do *Gigabit Ethernet* é 0,008 microsegundos (us)
- Max. Frame Rate: Para uma taxa de quadros de 30% o valor desse parâmetro é 446.429 fps.

Com base nos parâmetros acima é possível concluir que o tempo de envio de todos os quadros para uma taxa de 30% é 257.142,82 us.

$$Tempo\ envio(us) = (Preambulo\ +\ Tam.\ quadro) \times Ciclos\ byte \times Periodo\ clock \times Max.\ Frame\ Rate \quad (2.5)$$

Segundo passo:
O próximo passo é calcular o tempo de cada IFG, expresso na Equação 2.6. Por exemplo, para uma taxa de 30% o tempo de envio dos quadros é 257.142,82 us, então o tempo para cada IFG será de 1,66 us.

$$Tempo\ cada\ IFG(us) = \frac{1.000.000(us) - Tempo\ envio(us)}{Max.\ Frame\ Rate} \tag{2.6}$$

Terceiro passo:
O último passo é converter o tempo de cada IFG para *bytes*, expresso na Equação 2.7. Como o tempo para cada IFG é de 1,66 us e o período de *clock* é de 0,008 us então é possível concluir que o tamanho do IFG para a taxa de 30% é de 208 *bytes*.

A Equação 2.7 apresenta a forma para calcular tamanho do IFG.

$$Tamanho\ IFG(Bytes) = \frac{Tempo\ cada\ IFG(us)}{Periodo\ clock} \tag{2.7}$$

O Apêndice A apresenta uma listagem das taxa de quadros e o tamanho de IFG com uma granularidade de 10% para cada tamanho de quadro sugerido pela RFC 2544.

2.2.3 Conjunto de testes

A RFC 2544 estabelece um conjunto de seis testes que podem ser utilizados para medir o desempenho do dispositivo *Ethernet*, são eles: Vazão, Latência, Taxa de perda de quadros, *Back-to-back frames*, *System Recovery* e *Reset*.

2.2.3.1 Vazão

Teste utilizado para determinar a vazão máxima do DUT. Segundo a RFC 1242, vazão é a taxa máxima de quadros que é possível enviar para o DUT de forma que nenhum quadro seja perdido.

O teste se inicia enviando quadros a uma taxa de 100% para o DUT. Se algum quadro for perdido nessa operação o teste é repetido a uma taxa mais baixa e continua até que a vazão máxima seja encontrada. Geralmente esse processo consiste numa busca binária, aumentando ou diminuindo os valores da vazão a cada passo em: 50%, 25%, 12.5%, 6.25%, 3.125% etc. A taxa de quadros sofre um incremento ou decremento dependendo dos resultados obtidos no

teste anterior.

Este teste deve ser realizado para todos os tamanhos de quadro exibidos na Tabela 3 da Seção 2.2.1.

2.2.3.2 Latência

Latência é definido como o intervalo de tempo necessário para o quadro sair do Testador passar pelo DUT e retornar ao Testador. A RFC 2544 sugere que o teste de latência seja executado por 120 segundos para cada tamanho de quadro e repetido por 20 vezes. Os quadros devem ser enviados na vazão máxima suportada pelo DUT.

2.2.3.3 Taxa de perda de quadros

É o percentual de quadros que são perdidos pelo DUT durante o teste de uma determinada taxa de transmissão. Esse percentual é indicado pela Equação 2.8.

$$Taxa\ perda\ quadros(\%) = \frac{((quadros\_enviados - quadros\_recebidos) \times 100)}{quadros\_enviados} \tag{2.8}$$

No primeiro teste os quadros devem ser enviados a uma taxa de 100%. Essa operação deve ser repetida para 90%, depois 80% e assim por diante. O teste pode ser interrompido quando dois testes sucessivos não reportarem perda de quadros. A granularidade máxima deve ser de 10%, mas granularidades inferiores podem ser utilizadas.

2.2.3.4 *Back-to-back frames*

Consiste em determinar a capacidade do DUT de suportar rajadas de quadros com intervalo mínimo de IFG, ou seja, quadros sendo enviados a uma taxa de 100% sem que haja perda de quadros. O teste consiste em enviar um determinado número de quadros para DUT e repetir o processo aumentando ou diminuindo o número de frames dependendo do resultado do teste anterior. Deve ser efetuado para todos os tamanhos de quadros. Esse teste também é conhecido como "*burstability*"ou "*maximum throughput test*".

2.2.3.5 *System Recovery*

Este teste visa determinar o tempo necessário para que o DUT se recupere de rajadas de quadros enviadas 10% acima da vazão máxima suportada.

Primeiramente deve-se determinar a vazão máxima suportada para cada tamanho de quadro. Após, deve ser enviada uma rajada de quadros 10% acima da vazão máxima que o DUT suporta por pelo menos 60 segundos. Então, deve-se reduzir a taxa de envio de quadros para 50% da atual, e verificar qual o tempo necessário para que o DUT volte a operar normalmente e não mais apresente perda de quadros.

2.2.3.6 *Reset*

Teste utilizado para determinar o tempo necessário para o DUT se recuperar de um reinício do *software* ou do próprio dispositivo. Consiste em enviar quadros de 64 *bytes* na máxima vazão suportada pelo DUT, e então, provocar o reinício do DUT e verificar qual o tempo necessário para que o DUT volte a encaminhar os quadros para o Testador.

2.3 Trabalhos Relacionados

Esta Seção apresenta arquiteturas existentes da pilha de comunicação, tanto em *hardware* reconfigurável quanto em dispositivos que visam realizar o *offload* da pilha de comunicação do processador através da técnica conhecida como TCP/IP *offload Engine* (TOE).

2.3.1 Pilhas de comunicação em *hardware* reconfigurável

Em (LÖFGREN et al., 2005) são apresentadas três propostas de implementação da pilha UDP/IP num *Field Programmable Gate Array* (FPGA) Spartan-3 xc3s200 da Xilinx (XILINX, 2010a). Nesse trabalho os autores descrevem e analisam o paralelismo de uma pilha UDP/IP implementada em FPGA para sistemas *Ethernet* embarcados. As três implementações da pilha UDP/IP são "*Minimum*", "*Medium*", "*Advanced*". Elas possuem a estrutura apresentada na Figura 14 e implementam a camada de transporte, de rede e de enlace.

Dentre as três, a implementação "*Advanced*"é a que apresenta melhores resultados de desempenho, mas também é a que ocupa mais área em relação as outras implementações. Detalhes dessas três implementações podem ser verificados na Tabela 5. (LÖFGREN et al., 2005) afirma que a implementação "*Advanced*"apresenta vazão em torno de 957 Mbps quando excluído o processamento relacionado ao preâmbulo, cabeçalhos UDP/IP, CRC e IFG, mas afirma também que o desempenho geral do sistema está diretamente relacionado à latência presente na camada de aplicação.

As implementações apresentadas por (LÖFGREN et al., 2005), atualmente estão vinculadas

à empresa *RealFast*, que apresenta UDP/IP *Cores* para comunicação *Ethernet* de tempo real (REALFAST, 2010).

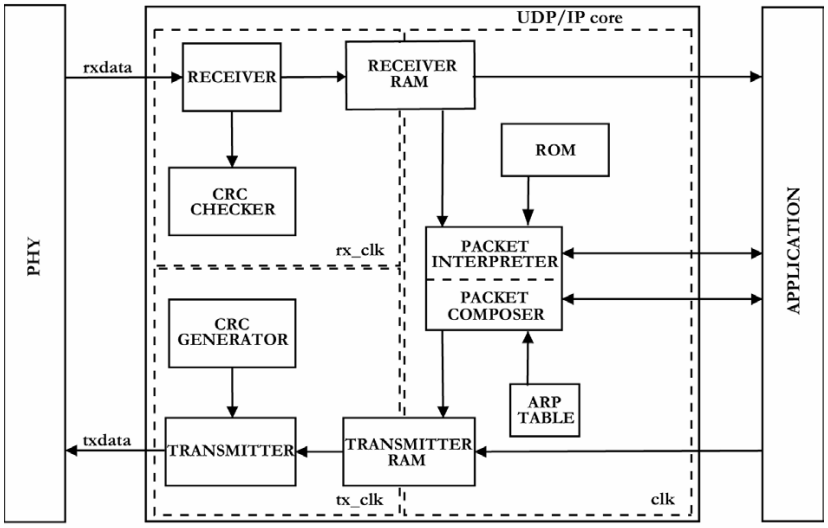


Figura 14: Estrutura da pilha UDP/IP em FPGA proposta por (LÖFGREN et al., 2005).

Tabela 5: Comparação das implementações propostas por (LÖFGREN et al., 2005).

Spartan-3, xc3s200-4ft256			
	"Minimum"	"Medium"	"Advanced"
Xilinx Slices	517	1022	1584
Xilinx BRAMs	3	3	5
Fmax (MHz)	90,7	60,3	105,6
Length (bytes) *	256	256	1518
Duplex mode	Full	Full	Full
Speed (Mbps)	10/100	10/100	10/100/1000
ARP	No	4 entries	4 entries
RARP	No	No	Yes
ICMP	No	Yes	Yes
TCP "channel"	No	No	Yes
Flexibility	Low	Medium	High

* Não considerando o preâmbulo.

(DOLLAS et al., 2005) apresenta uma arquitetura compreendendo um *open source TCP/IP core* capaz de processar pacotes ARP, ICMP, UDP, IP e TCP. A arquitetura, que pode ser visualizada na Figura 15, é composta pelos seguintes módulos internos:

- MII-INTERFACE: Interface MII responsável pela comunicação entre a camada MAC e a camada PHY, responsável também pelo controle dos sinais de envio (TX) e recebimento (RX) dos quadros pela rede.
- RxETHER: Implementa a camada de enlace e é responsável pelo recebimento dos quadros.

- ARP-RECV-REPLY: Bloco responsável por montar um pacote ARP *Reply*, quando acionado.
- ICMP-RECV-REPLY: Bloco responsável por montar um pacote ICMP *Reply*, quando acionado.
- IP-RECV: Implementa a camada de rede e é responsável pelo recebemento e validação de pacotes do protocolo IP.
- CRC-32: Blocos responsáveis pelo cálculo do CRC do quadro.
- TCP: Implementa a camada de transporte e realiza o processamento referente ao protocolo TCP.
- UDP: Implementa a camada de transporte e realiza o processamento referente ao protocolo UDP.
- ARP-TABLE: Mantém uma tabela ARP que relaciona um endereço IP com um endereço físico, MAC.
- TxRAM: Memória utilizada para armazenar o pacote a ser transmitido.
- TxETHER: Implementa a camada de enlace e é responsável pelo envio dos quadros.
- IP-SEND: Implementa a camada de rede e é responsável pelo encapsulamento do pacote a ser transmitido em um cabeçalho IP.
- ARP-REQUEST: Cria um datagrama do tipo ARP *Request* e o transmite. Este datagrama é responsável pela descoberta do endereço físico associado ao endereço IP.
- CHECKSUM: Realiza o *checksum* dos dados do datagrama.
- CONTROL: Responsável por sincronizar todos os módulos durante a recepção e envio de um datagrama e também por supervisionar o acesso à TxRAM.

A arquitetura proposta por (DOLLAS et al., 2005) é capaz de gerenciar 15 conexões UDP e 31 conexões TCP simultaneamente. Foi desenvolvida através de um fluxo de projeto ASIC, mas para fins de validação foi prototipada em FPGA Xilinx Virtex II XC2V8000. O projeto atinge uma frequência de 37.5 MHz e 700 Mbps de vazão em modo *full-duplex*.

Em (WEIDONG, 2003) é apresentada uma análise de todas as sub-funções da pilha TCP/IP que são consideradas críticas em termos de desempenho quando executadas num *General-Purpose Processor* (GPP), ou microprocessador de propósito geral. (WEIDONG, 2003) realiza

a implementação em VHDL das sub-funções, *checksum*, CRC e *Network Address Translation* (NAT), baseado no estudo realizado pelos trabalhos de (KAY; PASQUALE, 1996) e (HENRIKSSON; LIU, 2000).

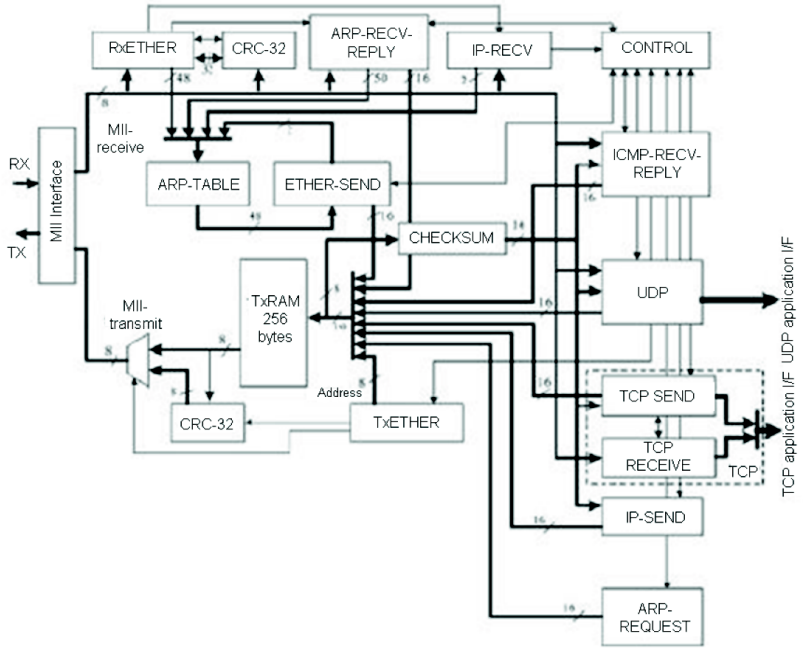


Figura 15: Arquitetura do *open source TCP/IP core* proposto por (DOLLAS et al., 2005).

2.3.2 TCP/IP *Offload Engine* (TOE)

Outros trabalhos também podem ser citados, estes não visam somente a implementação da pilha TCP/IP em *hardware*, mas também a diminuição da carga de processamento do microprocessador e do sistema de I/O com a pilha TCP/IP. Esse processo é efetuado retirando-se o processamento dos protocolos TCP/IP do processador do computador e movendo-os para um dispositivo/*hardware* auxiliar que passa a executar essa função, técnica também conhecida como TCP/IP *offload Engine* (TOE) (YEH et al., 2002) (SENAPATHI; HERNANDEZ, 2004).

Atualmente, TOE é largamente utilizado em servidores, como é o caso do servidor *Dell PowerEdge*, que foi desenvolvido pela *Dell*, *Broadcom* e *Microsoft* (GUPTA; LIGHT; HAMEROFF, 2006). A Figura 16 demonstra a idéia principal dessa implementação. Conforme pode ser observado, o lado esquerdo da Figura 16 apresenta a estrutura tradicional de uma pilha TCP/IP, com as camadas de transporte e rede implementadas em *software*. E o lado direito apresenta a estrutura do dispositivo desenvolvido, TOE *Network Interface Card* (NIC), onde as camadas de transporte e rede se encontram implementadas dentro de um dispositivo de *hardware*, enquanto que no *software* existe apenas uma camada responsável pela comunicação com

a aplicação.

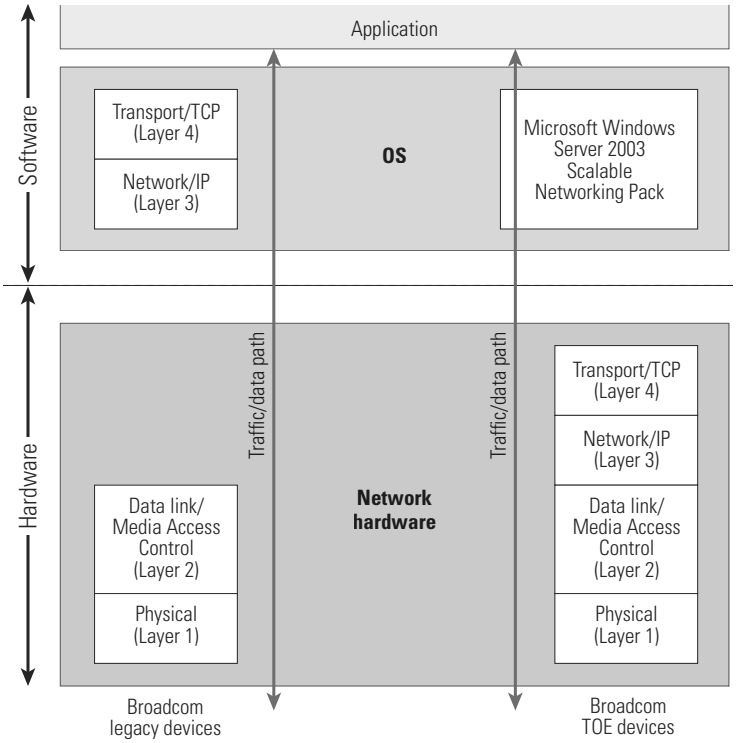


Figura 16: Estrutura tradicional das camadas da pilha TCP/IP e estrutura TOE demonstrada em (GUPTA; LIGHT; HAMEROFF, 2006).

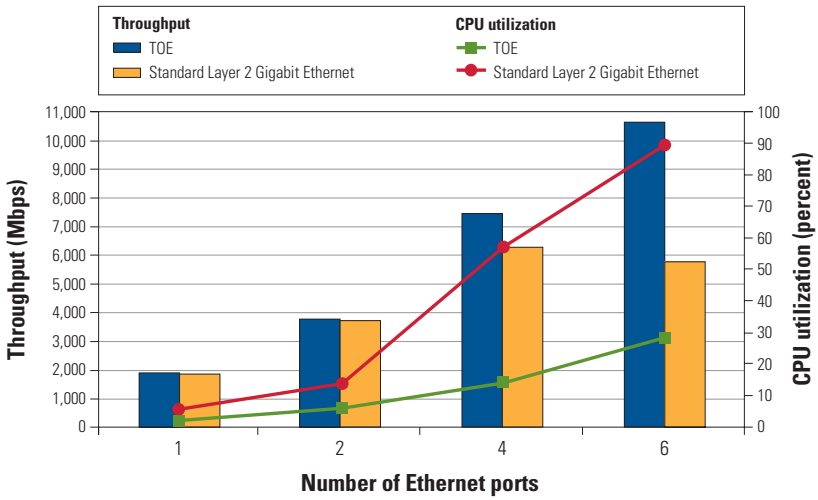


Figura 17: Vazão e efeitos na utilização da CPU: Estrutura tradicional comparada com estrutura TOE (GUPTA; LIGHT; HAMEROFF, 2006).

Ainda referente a implementação realizada em (GUPTA; LIGHT; HAMEROFF, 2006), a Figura 17 apresenta uma comparação entre o sistema com a estrutura tradicional (*Standard Layer 2 Gigabit Ethernet*) e um sistema com TOE . Ao utilizar uma ou duas portas *Ethernet* os dois sistemas apresentam um desempenho semelhante em relação à vazão e à utilização da CPU. Porém, ao utilizar quatro ou seis portas *Ethernet* essa diferença de desempenho se torna mais

acentuada. Com seis portas Ethernet, por exemplo, a estrutura tradicional utiliza três vezes mais a CPU que o sistema com TOE, e, no entanto só consegue atingir a metade da vazão alcançada pelo sistema com TOE.

Em (CHUNG et al., 2007) são propostas cinco diretrizes para utilização de TOE em FPGAs. Essas diretrizes são técnicas que permitem atingir velocidades de transmissão satisfatórias, mesmo utilizando FPGAs com baixo poder de processamento. Para testes comparativos, foi utilizado um computador dotado de CPU Intel Xeon 3GHz, 2GB DDR SDRAM e com Kernel Linux 2.4.22 contendo a implementação tradicional da pilha TCP/IP. E outro computador com as mesmas configurações, mas dotado de uma placa de prototipação Xilinx Virtex II Pro FPGA (XC2VP50) realizando o TOE da pilha. Neste FPGA, o autor implementou uma versão simplificada, que não atende a todos os RFCs, da pilha TCP/IP. Os resultados dessa comparação podem ser verificados na Tabela 6. Nessa tabela é possível observar que a implementação tradicional obtém melhores resultados comparados ao TOE em termos de vazão, porém a um custo de processamento na ordem de 2/3 a 3/4 mais elevado.

Tabela 6: Comparação das implementações propostas por (CHUNG et al., 2007).

	Transfer Size	Sender (Mbps)/ CPU load	Receiver (Mbps)/ CPU load
Using a GbE NIC with conventional TCP/IP stack (linux 2.4.22)	200MB	742.21 / 37.4%	838.38 / 46.9%
	400MB	733.31 / 41.6%	815.45 / 41.8%
	800MB	732.80 / 39.4%	808.50 / 48.8%
	1.6GB	733.75 / 41.4%	822.35 / 44.2%
	3.2GB	741.77 / 35.8%	829.01 / 40.0%
	6.4GB	745.17 / 35.9%	837.06 / 41.5%
Using the proposed TOE	200MB	722.43 / 15.1%	665.83 / 31.7%
	400MB	744.35 / 12.1%	747.20 / 25.8%
	800MB	778.35 / 12.2%	801.72 / 20.8%
	1.6GB	782.97 / 12.1%	830.05 / 19.1%
	3.2GB	788.45 / 11.7%	851.85 / 16.9%
	6.4GB	793.45 / 9.1%	858.82 / 16.7%

(KIM et al., 2008) implementa o TOE para *Ethernet* de 1 *Gigabit* utilizando uma placa PCI-Express dotada de dois processadores *PowerPC* (PPC) 405, um FPGA Xilinx Virtex II Pro e PHY *Gigabit Ethernet* GPHY88E111 (MARVELL, 2009). Esse placa implementa as camadas de transporte e rede da pilha TCP/IP. Experimentos demonstram uma queda de 50% a 79% no tempo de execução do *Kernel* do sistema operacional com a utilização do TOE para o envio e recebimento de pacotes.

(KIM et al., 2009) também implementa TOE, porém, essa implementação é efetuada através da utilização de um ASIC 0.13um CMOS contendo dois ARM922T (ARM, 2010) operando a 250 MHz. Em seus testes experimentais, o autor demonstra que é possível reduzir o uso da

CPU em torno de 30% se comparado com uma implementação tradicional. Os resultados desse trabalho são exibidos na Figura 18.

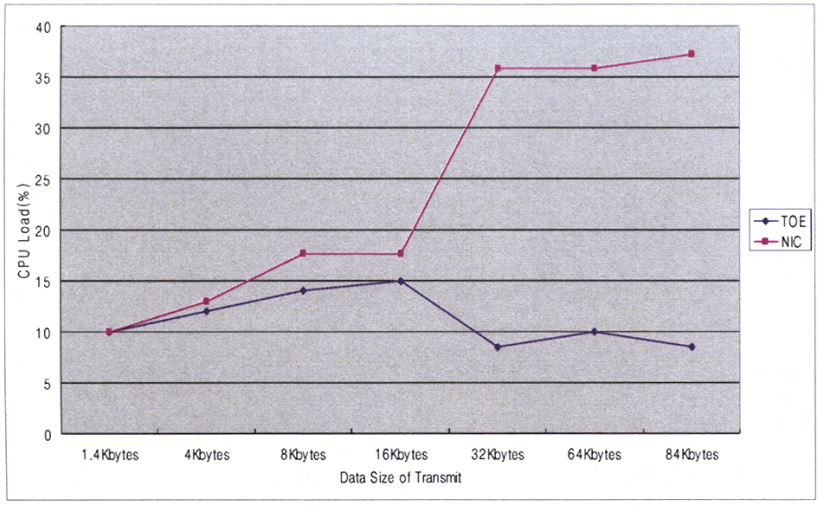


Figura 18: Resultados da utilização da CPU, comparando TOE com a estrutura tradicional (KIM et al., 2009).

Com os resultados dos trabalhos apresentados nesta Seção é possível observar que recursos que retiram do processador principal a função de transmissão de dados, como TOE, apresentam um ganho significativo de desempenho em relação a soluções em que o processador é utilizado para o processamento da pilha TCP/IP.

2.4 Resumo

Este Capítulo apresentou conceitos acerca de redes de computadores, bem como abordou detalhes e protocolos que compõem as camadas de transporte, de rede e de enlace da pilha de comunicação TCP/IP. Também foram apresentados os testes e conceitos envolvidos no conjunto de testes sugeridos na metodologia proposta pela RFC 2544 para medição de desempenho de dispositivos *Ethernet*. E por final foram apresentadas arquiteturas existentes de pilhas de comunicação em *hardware* reconfigurável, e também implementações comerciais e acadêmicas utilizando a técnica TOE.

3 ARQUITETURA PROPOSTA DO TESTADOR E DAS PILHAS DE COMUNICAÇÃO UDP/IP

Este Capítulo apresenta as arquiteturas propostas das pilhas de comunicação UDP/IP e do Testador, bem como o seu desenvolvimento e implementação em *hardware* reconfigurável. São também apresentados detalhes sobre a estrutura e funcionamento interno das arquiteturas. E ao final é apresentada a metodologia e o ambiente utilizado no desenvolvimento e na validação tanto do Testador quanto das pilhas de comunicação UDP/IP.

3.1 Testador

O Testador consiste de uma implementação *Gigabit* para a metodologia de testes de dispositivos *Ethernet* da RFC 2544. O conjunto de testes dessa implementação engloba os testes de vazão, latência e taxa de perda de quadros. Esse dispositivo foi implementado na placa de desenvolvimento ML402 (XILINX, 2006) dotado de um FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4 e desenvolvido com o auxílio das seguintes ferramentas da Xilinx: ISE *Design Suite* 10.1, *Embedded Development Kit* (EDK) 10.1 e *Software Development Kit* (SDK) 10.1 (XILINX, 2010b). Essa implementação se encontra dividida em duas camadas, são elas: *software* e *hardware*. Essa divisão e os blocos internos de cada camada podem ser observados na Figura 19.

As Seções seguintes visam fornecer detalhes referentes à estrutura e ao funcionamento dos blocos que compõem as camadas de *hardware* e *software*.

3.1.1 Blocos do *Hardware*

Os blocos que compõem a parte do *hardware*, ilustrado na Figura 19, são: *Top Testador*, *Sender*, *Receiver*, *CRC Generator e Checker*, DCM e GMII *Management*. Eles foram desen-

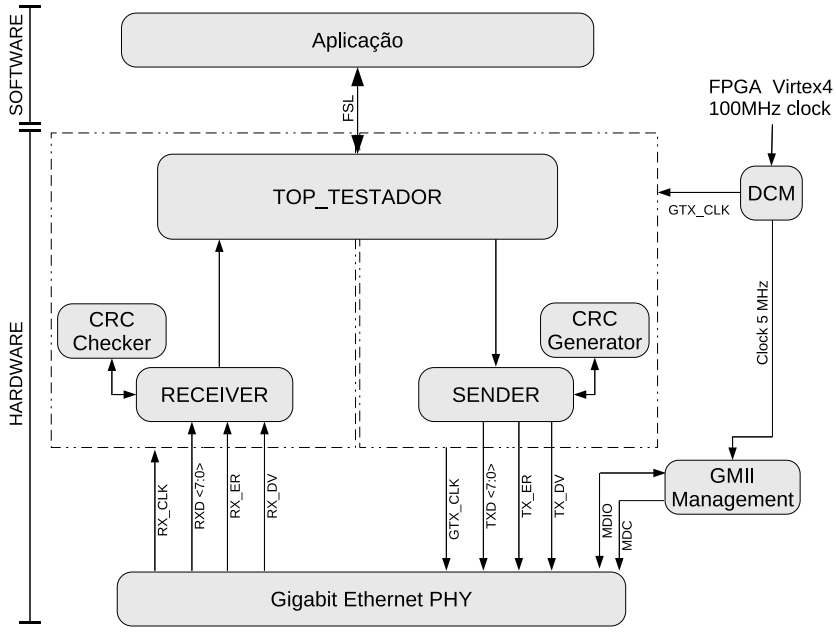


Figura 19: Arquitetura do Testador.

volvidos na linguagem Verilog e os detalhes de cada um se encontram nas Seções a seguir.

3.1.1.1 *Top Testador*

Esse bloco é responsável por gerenciar os blocos *Sender* e *Receiver* e também por realizar a comunicação com a aplicação através do barramento *Fast Simplex Link* (FSL) (XILINX, 2009c).

Para receber dados oriundos da aplicação, o bloco *Top Testador* monitora o pino de saída *FSL_S_Exists* do barramento FSL que indica a existência (*FSL_S_Exists*='1') ou não (*FSL_S_Exists*='0') de dados a serem lidos. O controle de fluxo de dados do protocolo FSL é realizado através da porta de leitura (*FSL_S_Read*). Assim, sempre que esta porta estiver desativada (*FSL_S_Read*='0') o barramento FSL não fornecerá dados. Os dados são fornecidos através do barramento *FSL_S_Data[0:31]*. Mais detalhes sobre o barramento FSL encontram-se na Seção 3.1.2.1.

O bloco *Top Testador* possui os seguintes pinos de entrada que estão conectados ao barramento FSL e são recebidos exatamente nessa ordem:

- *frames_p_second[23:0]*: Quantidade de quadros que devem ser enviados por segundo.
- *frame_size_i[15:0]*: O tamanho de cada quadro, por exemplo, 64 *bytes*.
- *cycles_ifg_i[19:0]*: Quantidade de ciclos entre um quadro e outro, ou seja, o IFG.

- *repeat_test_i[7:0]*: Quantas vezes este teste deve ser executado, uma vez que os parâmetros acima são configurações para testes referentes a um segundo.

- *new_test_i*: Quando o valor deste pino for '1' indica que um novo teste deve ser iniciado.

Para enviar os resultados do teste para a aplicação, o bloco *Top Testador* ativa o pino *FSL_M_Write* e grava os dados no pino *FSL_M_Data[0:31]* do barramento FSL. Juntamente com esta operação deve ser monitorado o pino *FSL_M_Full* para verificar se o *buffer* do barramento está ou não cheio. Caso não esteja cheio o dado é colocado no barramento e a porta *FSL_M_Write* é ativada. Caso o *buffer* torne-se cheio o bloco *Top Testador* efetua uma pausa na gravação de dados (*FSL_M_Write*='0') até que o *buffer* do barramento FSL possua espaço novamente, o que acontecerá quando a aplicação efetuar a leitura de dados.

O bloco *Top Testador* possui os seguintes pinos de saída que estão conectados ao barramento FSL e são enviados para a aplicação exatamente nessa ordem:

- *finish_o*: Quando valor deste pino for '1' indica que o teste está finalizado e os resultados serão enviados.

- *rcv_error_frame_o[31:0]*: Quantidade de quadros que não foram recebidos.

- *max_latency_o[31:0]*: Latência máxima encontrada durante o período de duração do teste, expressa em quantidade de ciclos de 8 ns.

- *min_latency_o[31:0]*: Latência mínima encontrada durante o período de duração do teste, expressa em quantidade de ciclos de 8 ns.

- *total_latency_o1[31:0]*: MSB da soma de todas as latências, expresso em quantidade de ciclos de 8 ns.

- *total_latency_o2[31:0]*: LSB da soma de todas as latências, expresso em quantidade de ciclos de 8 ns.

- *cnt_invalid_packets_o[31:0]*: Quantidade de quadros que não foram computados devido a erros de *checksum*, versão de protocolo, CRC ou endereços MAC e IP inválidos. Para que o teste seja válido o valor de retorno deste pino deverá ser zero.

A estrutura interna do bloco *Top Testador* possui as seguintes máquinas de estados: *FSM_snd*, *FSM_rcv* e *FSM_cnt*. Essas máquinas de estados são responsáveis por gerenciar e realizar a comunicação com os blocos *Sender* e *Receiver*.

FSM_snd é responsável pela comunicação com o bloco *Sender* e também por gerenciar a quantidade de quadros que devem ser enviados e o IFG entre eles. Também informa ao bloco *Sender* o tamanho de cada quadro. Ao final, informa à maquina de estados *FSM_rcv* que todos os quadros foram enviados.

A máquina de estados *FSM_rcv* é responsável pela comunicação com o bloco *Receiver*. Possui registradores para o número de quadro recebidos com sucesso e o número de pacotes inválidos recebidos pelo *Receiver*. Possui também registradores para as latências máxima, mínima e total, que são atualizadas a cada quadro recebido com sucesso. O processo termina quando todos os quadros forem recebidos com sucesso ou quando, após o término do envio de todos os quadros pela *FSM_snd*, durante cinco segundos nenhum novo quadro for recebido, garantindo que o Testador não permaneça esperando infinitamente o recebimento de quadros que possam ter sido perdidos pelo DUT. Ao término, os resultados são enviados para aplicação através dos pinos de saída.

FSM_cnt implementa um contador de tempo de 40 *bits* que é utilizado para calcular a latência dos quadros. Quando um novo teste se inicia, esse contador é zerado e após incrementado a cada ciclo de *clock*.

3.1.1.2 *Sender*

O bloco *Sender* envia um quadro, com a formatação exibida na Figura 13 da Seção 2.2.1, para a interface GMII da PHY. No entanto, para facilitar a obtenção dos resultados de vazão, latência e taxa de perda de quadros foi desenvolvido um novo protocolo. Esse protocolo, cujo nome é *HerrTester*, é utilizado em conjunto com o formato padrão sugerido pela RFC 2544 e seu encapsulamento é MAC/IP/UDP. A Figura 20 apresenta o formato do cabeçalho do protocolo para um quadro de 64 *bytes*. Para quadros maiores, o formato do protocolo permanece o mesmo, o único campo que sofre alterações é o campo de dados.

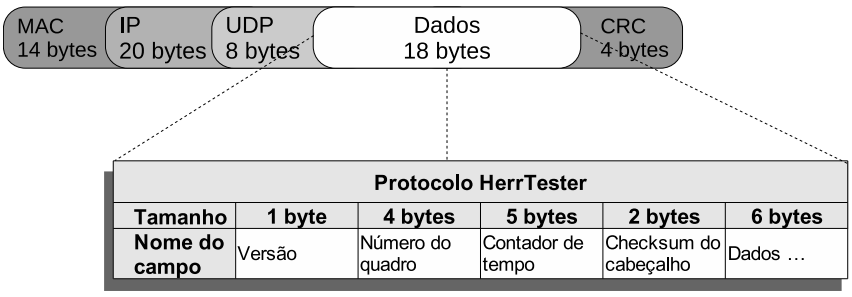


Figura 20: Formato do cabeçalho do protocolo *HerrTester* para um quadro de 64 *bytes*.

O cabeçalho do protocolo *HerrTester* possui os seguintes campos:

- Versão (1 *byte*): Versão do protocolo, deverá ser preenchido com o valor em hexadecimal '01'.
- Número do quadro (4 *bytes*): Número sequencial do quadro de teste.
- Contador de tempo (5 *bytes*): Armazena o tempo no momento do envio do quadro. Quando esse quadro for novamente recebido será possível calcular a sua latência baseado no valor do contador de tempo atual implementado pela máquina de estados *FSM_cnt*.
- *Checksum* do cabeçalho (2 *bytes*): Utilizado para detecção de erros. A implementação desse campo é baseada na RFC 1071 (BRADNER; BORMAN; PARTRIDGE, 1988).
- Dados: Esse campo é utilizado para *padding*, ou seja, apenas para completar o tamanho do quadro. Por exemplo, no formato apresentado na Figura 20 para um quadro de 64 *bytes*, esse campo possui 6 *bytes* para *padding*.

3.1.1.3 *Receiver*

O bloco *Receiver* é responsável por receber o quadro da interface GMII da camada PHY. Esse bloco realiza a validação do CRC, *checksum* de cabeçalhos, versões de protocolos e verificação de endereço MAC e IP. Se algum erro for detectado no quadro, este será computado como quadro inválido e repassado para a máquina de estados *FSM_rcv* do bloco *Top Testador*. Se a estrutura estiver correta, o bloco *Receiver* realizará o cálculo da latência, baseado no conteúdo do campo *contador de tempo* do protocolo *HerrTester* e no contador de tempo atual do bloco *Top Testador* implementado na máquina de estados *FSM_cnt*. Ao final, esse quadro será computado como válido e repassado para a máquina de estados *FSM_rcv* do bloco *Top Testador* para que as latências máxima, mínima e total sejam determinadas.

3.1.1.4 CRC *Generator* e *Checker*

Esse blocos são idênticos e ambos calculam o CRC baseado no polinômio gerador de 32 *bits* apresentado na Seção 2.1.3.1 do Capítulo 2.

O CRC provê detecção de erros decorrentes de algum erro na transmissão do quadro. Qualquer quadro que contenha um CRC inválido será descartado pelo bloco *Receiver* e computado como quadro inválido pela *FSM_rcv* do bloco *Top Testador*.

3.1.1.5 DCM

Digital Clock Manager (DCM) são estruturas internas que provêem funcionalidades de *clock* nos FPGAs da Xilinx (XILINX, 2009). DCMs também podem solucionar problemas relacionados ao *clock*, como por exemplo, *clock skew* em frequências altas.

O bloco DCM, apresentado na Figura 19, implementa dois *Digital Frequency Synthesizer* (DFS). DFSs são estruturas que fornecem uma frequência de saída baseada na relação de dois valores, um multiplicador (*CLKFX_MULTIPLY*) e um divisor (*CLKFX_DIVIDE*). As frequências de saída (*CLKFX* e *CLKFX180*) são derivadas do *clock* de entrada (*CLKIN*) e destes dois parâmetros (XILINX, 2009b). O primeiro DFS é utilizado para gerar um *clock* de 125 MHz, veja pino *GTX_CLK* na Figura 19, para os blocos *Sender*, *Top Testador*, *CRC Generator* e para a interface GMII da PHY. O outro DFS gera um *clock* de 5 MHz que é utilizado pelo bloco GMII *Management*.

Os blocos responsáveis pelo recebimento utilizam o *clock RX_CLK* fornecido pela interface GMII da PHY. Isso pode ser observado na Figura 19.

3.1.1.6 GMII *Management*

Esse bloco gerencia a interface GMII da PHY para operar em modo *full-duplex* e 1000 Mbps, utilizando o pino *MDIO*. O pino *MDC* é o *clock* de 5 MHz gerado pelo bloco DCM.

Maiores detalhes sobre a interface GMII podem ser encontrados na Seção 2.1.3.3.

3.1.2 Blocos do *Software*

Os blocos do *Software* compreendem a aplicação e o barramento FSL. Através do FSL é possível realizar a comunicação dos blocos desenvolvidos em *hardware* com a aplicação que está desenvolvida em linguagem C para o microprocessador *Microblaze* versão 7.10.d (XILINX, 2008).

3.1.2.1 Barramento *Fast Simplex Link* (FSL)

O barramento FSL é um barramento unidirecional utilizado para implementar comunicação entre dois blocos de *hardware* em FPGA (XILINX, 2009c).

Uma vez que o FSL é unidirecional, duas instâncias do *core* fornecido pela Xilinx foram utilizadas para integração dos blocos de *hardware* com o *Microblaze*. Uma instância é utilizada

para os pinos de entrada do bloco *Top Testador* e outra para os pinos de saída. O tamanho do barramento de dados de cada instância FSL pode variar de 1 a 32 *bits*. Para esta implementação foi utilizado o tamanho máximo do barramento, 32 *bits*. Cada instância FSL também possui uma *First In First Out* (FIFO) interna com até 8192 posições de profundidade. Como o Testador não utiliza um fluxo contínuo de dados e o tamanho dessa FIFO não interfere no desempenho do DUT, foi utilizada uma profundidade de apenas 16 posições.

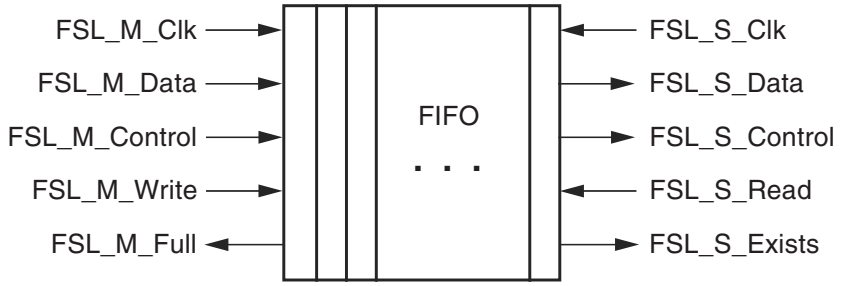


Figura 21: Pinos de entrada e saída do barramento FSL (XILINX, 2009c).

A Figura 21 ilustra os pinos de entrada e saída de cada interface do barramento FSL. Como é possível observar, através do barramento *Master* (pinos *FSL_M_**) são realizadas as gravações de dados, e através do barramento *Slave* (pinos *FSL_S_**) são realizadas as leituras. Desta maneira a integração do bloco *Top Testador* com as duas instâncias do FSL (fsl_0 e fsl_1) ocorre conforme ilustrado na Figura 22.

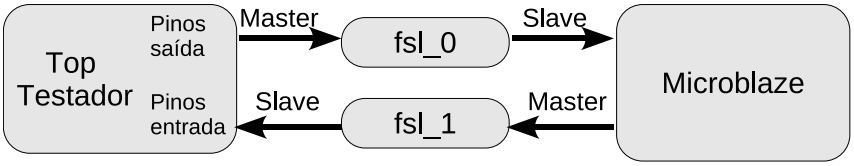


Figura 22: Conexão entre os blocos *Top Testador* com as instâncias dos barramentos FSL e o microprocessador *Microblaze*.

3.1.2.2 *Software* desenvolvido

Como todas as funcionalidades referentes ao envio e recebimento dos quadros de teste está implementado em *hardware*, o *software* implementa somente a comunicação com o barramento FSL e os algoritmos relacionados à vazão, latência e taxa de quadros perdidos.

A escrita das configurações dos quadros de teste no barramento FSL, bem como a leitura dos resultados é realizada por intermédio das funções de acesso ao barramento FSL. Estas funções encontram-se descritas no arquivo *header* fsl.h. O código na Listagem 1 demonstra como as escritas (*putfsl*) e leituras (*getfsl*) no barramento FSL devem ser realizadas.

Listagem 1: Código fonte C para escrita e leitura de dados no barramento FSL

```
#include "mb_interface.h"
#include "stdio.h"
#include "fsl.h"

int main()
{
    xil_printf("Inicializando...\r\n");
    //escrevendo os dados(configurações do teste) no barramento
    putfsl((Xuint32)frames_p_second, fsl_1,FSL_DEFAULT); //frames_p_second
    putfsl((Xuint32)fram_size_i,     fsl_1,FSL_DEFAULT); //frame_size_i
    putfsl((Xuint32)cycles_ifg_i,    fsl_1,FSL_DEFAULT); //cycles_ifg_i
    putfsl((Xuint32)repeat_test_i,   fsl_1,FSL_DEFAULT); //repeat_test_i
    putfsl((Xuint32)1,               fsl_1,FSL_DEFAULT); //new_test_i
    //lendo os resultados do barramento FSL
    getfsl(res_finish,           fsl_0,FSL_DEFAULT); //FINISH
    getfsl(res_rcv_error_frames, fsl_0,FSL_DEFAULT); //rcv_error_frames_o
    getfsl(res_max_latency,      fsl_0,FSL_DEFAULT); //max_latency_o
    getfsl(res_min_latency,      fsl_0,FSL_DEFAULT); //min_latency_o
    getfsl(res_total_latency_o1, fsl_0,FSL_DEFAULT); //total_latency_o1
    getfsl(res_total_latency_o2, fsl_0,FSL_DEFAULT); //total_latency_o2
    return 0;
}
```

As funções *putfsl* e *getfsl* possuem dois parâmetros: *val* e *id*. O primeiro parâmetro é o valor, o segundo é o barramento FSL que será utilizado. Essas duas funções são do tipo bloqueantes, ou seja, só serão executadas se existir espaço na FIFO, no caso da função *putfsl*, e se existirem dados para leitura, no caso da função *getfsl*.

A função *xil_printf* exibida na Listagem 1 pode ser utilizada para a visualização dos dados lidos. A interface padrão de I/O de dados do microprocessador Microblaze é a RS232, fato que não compromete o desempenho do Testador desenvolvido, pois após a gravação das configurações referentes ao teste (*putfsl*) a camada do *hardware* trabalha de forma independente e só volta a se comunicar, com a aplicação, ao final para reportar os resultados.

Além da comunicação com o barramento FSL, o *software* também implementa o algoritmo para a descoberta da vazão e latência de um dispositivo *Ethernet*, ilustrado no fluxograma da Figura 23. Para que o *hardware* efetue o teste para uma determinada taxa de quadros, o *software* calcula a quantidade de quadros para o pino *frames_p_second* do bloco *Top Testador* baseado na Equação 2.4, e o IFG para o pino *cycles_ifg_i* do bloco *Top Testador* baseado na Equação 2.7. Essas Equações se encontram na Seção 2.2.2.

O teste de vazão consiste em executar o algoritmo, ilustrado no fluxograma da Figura 23, para todos os tamanhos de quadros sugeridos pela RFC 2544 por 60 segundos para cada taxa de quadros que está sendo analisada. Esse tempo de 60 segundos é informado ao pino *repeat_test_i* do bloco *Top Testador*. Isso significa dizer que, para que um dispositivo *Gigabit Ethernet* suporte uma vazão de 100% para quadros de 64 *bytes*, por exemplo, ele deve ser capaz de processar e não perder nenhum dos 89.285.700 quadros enviados durante a realização do teste.

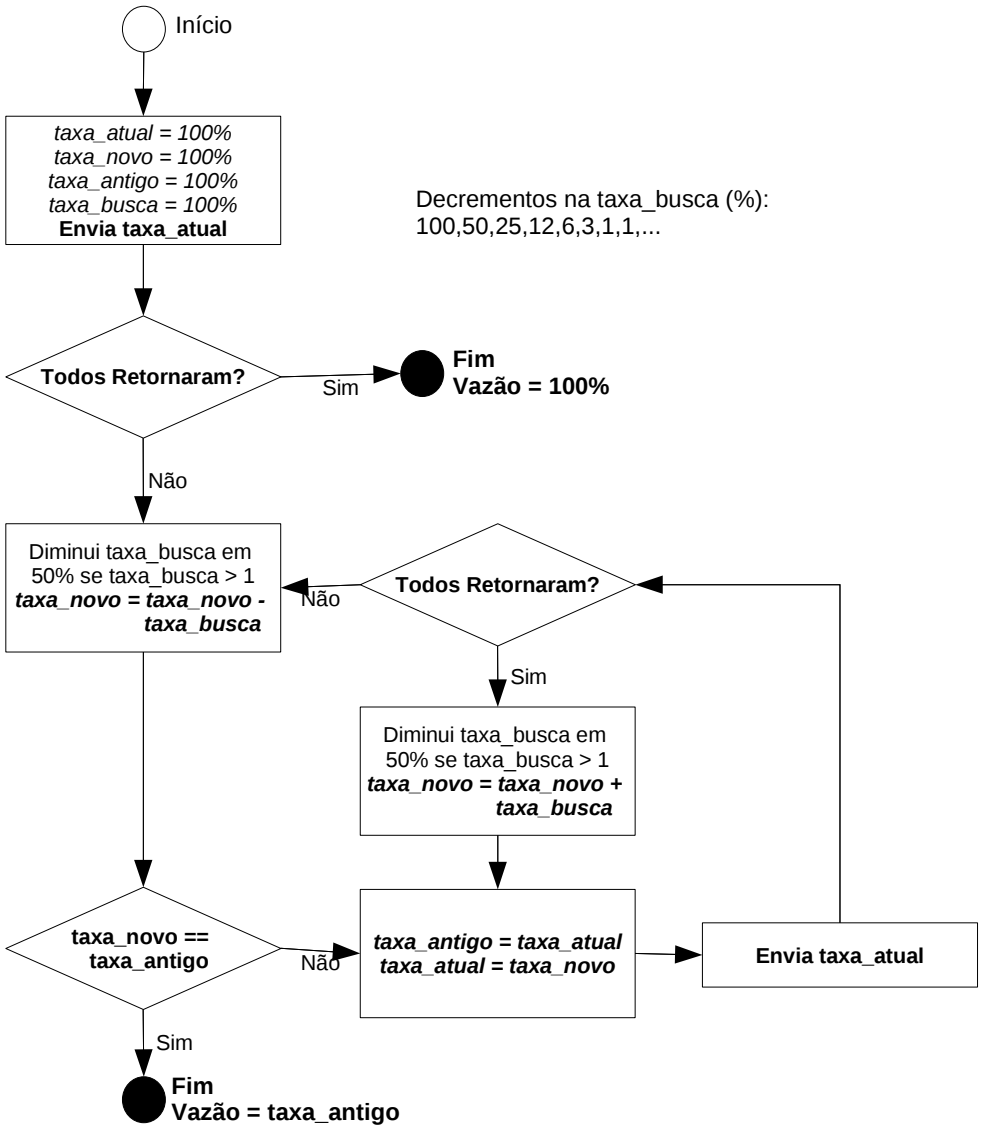


Figura 23: Fluxograma do algoritmo para a descoberta da vazão e latência de um dispositivo *Ethernet*.

O teste de vazão é repetido 5 vezes e a média dos resultados encontrados é calculada. Juntamente com o teste de vazão também é executado o teste de latência. Como o Testador considera para cálculo da latência somente os quadros que foram recebidos, é possível calcular a latência para vazão de 100% mesmo em dispositivos que não suportam tal vazão. Para o cálculo da latência o Testador faz uso do campo Contador de tempo do protocolo *HerrTester*

apresentado na Figura 20 da Seção 3.1.1.2

Outro algoritmo executado pelo Testador é o que calcula o percentual de quadros que são perdidos pelo DUT. O fluxograma deste algoritmo pode ser visualizado na Figura 24. Esse teste também é executado para todos os tamanhos de quadros sugeridos pela RFC 2544, repetido 5 vezes, e a média das taxas de perda calculada.

No primeiro teste os quadros são enviados a uma taxa de 100%. Essa operação é repetida para 95%, depois 90% e assim por diante. O teste é interrompido quando dois testes sucessivos não reportarem perda de quadros. A granularidade utilizada por esse algoritmo é de 5%.

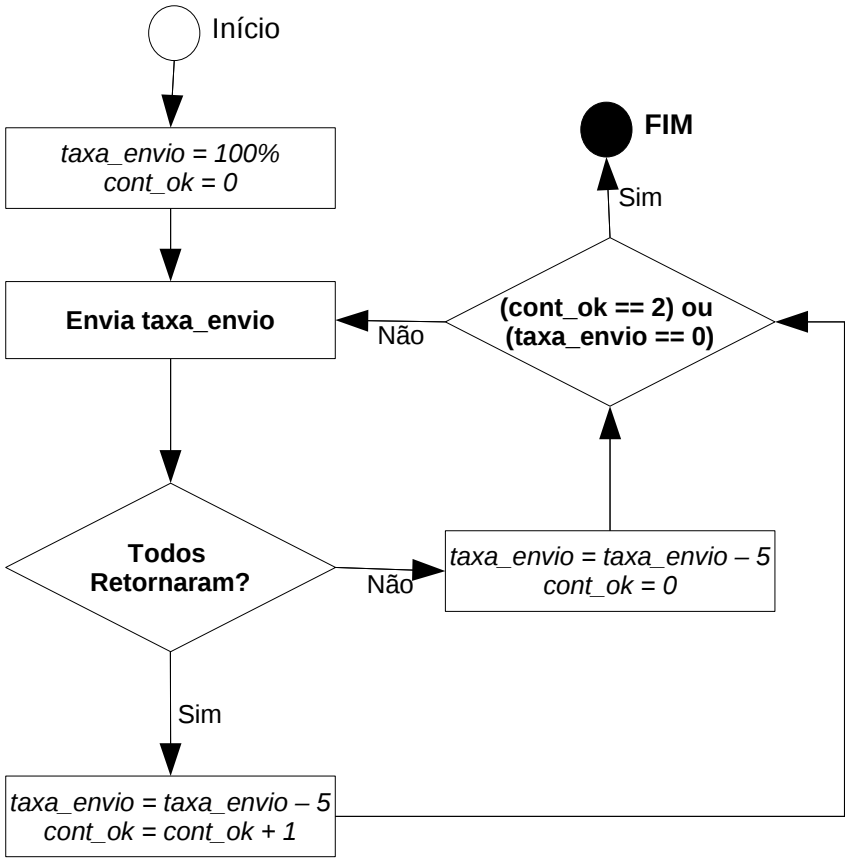


Figura 24: Fluxograma do algoritmo para calcular a taxa de perda de quadros de um dispositivo *Ethernet*.

3.1.3 Utilização dos recursos do FPGA

A utilização dos recursos internos do FPGA Virtex-4 XC4VSX35-10ff668 pelo Testador pode ser conferida na Tabela 7. Essas informações foram retiradas dos relatórios de síntese da ferramenta EDK da *Xilinx*.

Na Tabela 7 também é possível observar algumas estruturas específicas dos FPGAs, como

as *Block Random Access Memory* (BRAMs) que são memórias internas do FPGA (XILINX, 2009d). As BRAMs são utilizadas pelas FIFOs do barramento FSL, pelo microprocesssador *Microblaze* e pelo barramento *Processor Local Bus* (PLB) que possibilita a conexão do *Microblaze* com os dispositivos de I/O, como por exemplo a RS232 disponível na placa de desenvolvimento ML402. O FPGA Virtex-4 XC4VSX35 possui 192 blocos de BRAM de 18 Kb cada, o que totaliza 3.456 Kb ou 432 KB de memória interna.

Já outras estruturas como as DSP48 são utilizadas para implementar funções matemáticas básicas, como por exemplo, somadores, subtratores, acumuladores, multiplicadores, contadores e divisores (XILINX INC., 2005).

Tabela 7: Utilização dos recursos do FPGA Virtex-4 XC4VSX35-10ff668 pelo Testador.

Recurso	**Utilizado**	**Total disponível**	**% utilizado**
Slices	5.124	15.360	33%
Flip Flops	5.524	30.720	17%
Look-up table (LUTs)	6.943	30.720	22%
Block Random Access Memory (BRAMs)	32	192	16%
Global Clock Buffer (GCLKs)	6	32	18%
Digital Clock Manager (DCMs)	1	8	12%
DSP48s	9	192	4%

3.2 Pilha de comunicação UDP/IP em *hardware* reconfigurável

Foram desenvolvidas três arquiteturas (versões) da pilha UDP/IP em *hardware* reconfigurável. As três versões foram desenvolvidas em linguagem Verilog e prototipadas na placa de desenvolvimento XUPV5-LX110T (XILINX, 2010c) dotada de um FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5.

Todas as versões implementam as camadas de enlace (MAC), de rede (IPv4) e de transporte (UDP). A primeira arquitetura (versão) da pilha UDP/IP em *hardware* (UDPIPv1) é a versão básica. Baseado nos resultados dos testes realizados na UDPIPv1 pelo Testador demonstrado na Seção 3.1, foram desenvolvidas as outras arquiteturas, UDPIPv2 e UDPIPv3, com o intuito de melhorar o desempenho da pilha UDP/IP nos testes de vazão, latência e taxa de perda de quadros.

A camada de aplicação consiste de uma camada, também desenvolvida em Verilog, somente com o intuito de fazer um *loopback*, ou seja, repassar os dados recebidos para as camadas responsáveis pelo envio dos dados. A comunicação do *core* UDP/IP com a camada de aplicação, bem como a camada de aplicação estão detalhadas nas Seções de cada arquitetura da pilha

UDP/IP.

3.2.1 Arquitetura da pilha de comunicação UDP/IP Versão 1 - UDPIPv1

A primeira versão da pilha de comunicação UDP/IP implementa as camadas de enlace (MAC), de rede (IPv4) e de transporte (UDP). A estrutura dessa versão da pilha é apresentada na Figura 25. Nessa Figura também é possível observar o *core* UDP/IP contendo as camadas da pilha UDP/IP e o bloco de controle. Como os blocos *Receiver* e *Transmitter* trabalham de forma simultânea e independente, a implementação dessa pilha é *full-duplex*.

As próximas Seções visam detalhar o funcionamento de cada bloco dessa versão da pilha, bem como a comunicação do core UDP/IP com a camada de aplicação.

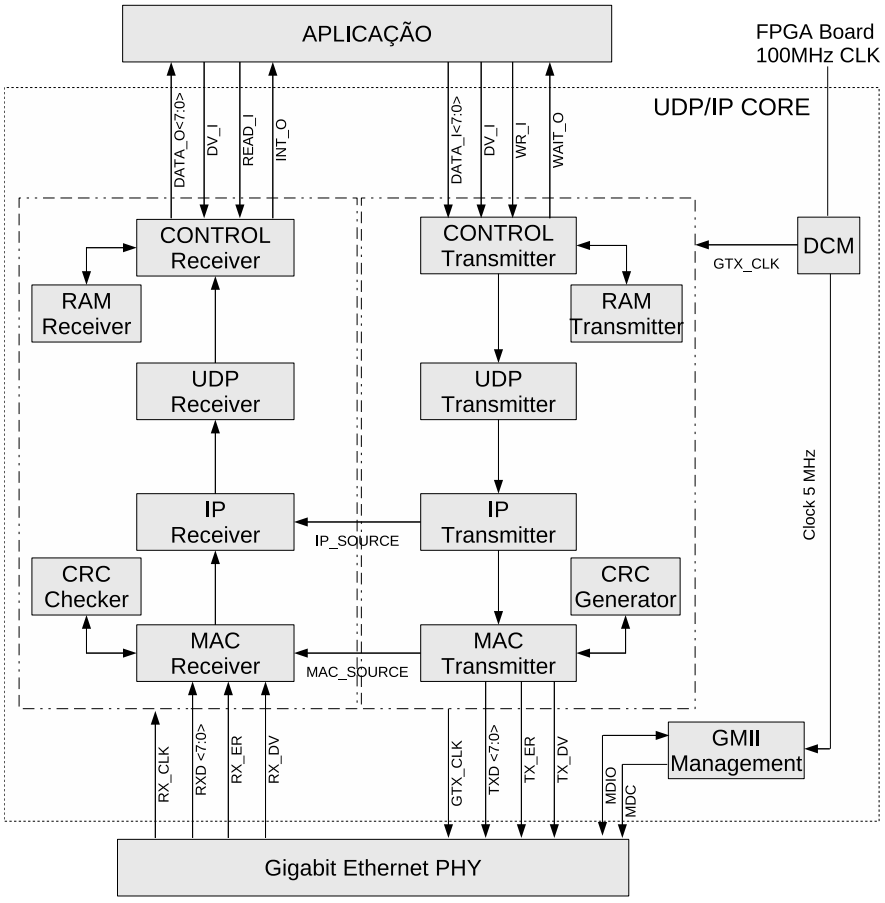


Figura 25: Diagrama de blocos da pilha UDP/IP versão 1 (UDPIPv1) desenvolvida em Verilog.

3.2.1.1 *Control Receiver/Transmitter*

Esses blocos são responsáveis por realizar a comunicação do *core* UDP/IP com a camada de aplicação. O bloco *Control Receiver* recebe o pacote do bloco *UDP Receiver* e grava-o

na memória RAM *Receiver*. Após recebê-lo completamente, este pacote é repassado para a camada de aplicação. O bloco *Control Transmitter* recebe o pacote da aplicação, armazena na RAM *Transmitter* e depois envia para o bloco UDP *Transmitter*.

A comunicação do *core* UDP/IP com a camada de aplicação, através dos blocos *Control Receiver* e *Control Transmitter*, é baseada na troca de pacotes de configuração e dados. Conforme exibido na Tabela 8 existem três pacotes de configuração e dois pacotes de dados.

Tabela 8: Tipos de pacotes utilizados na comunicação do *core* UDP/IP com a camada de aplicação.

Pacote	**Tipo**	**Valor (Hexadecimal)**
UDP	Dados	11
TCP	Dados	06
MAC Origem	Configuração	81
IP Origem	Configuração	82
MAC *Gateway*	Configuração	88

Os pacotes de configuração são enviados da aplicação para o *core* UDP/IP e são utilizados somente para configurar os endereços MAC de origem, IP de origem e MAC *gateway* do *core*. A estrutura desses três pacotes de configuração pode ser visualizada na Figura 26.

Tipo (8 bits) Hexadecimal	**Configuração**
81	MAC Origem (48 bits)
82	IP Origem (32 bits)
88	MAC Gateway (48 bits)

Figura 26: Estrutura dos pacotes de configuração do *core* UDP/IP.

Os pacotes de dados são enviados da aplicação para o *core* UDP/IP e também do *core* UDP/IP para a aplicação. Existem dois pacotes de dados, TCP e UDP, e eles possuem a estrutura ilustrada na Figura 27. A Figura 27 exibe a estrutura dos pacotes de dados que são enviados da aplicação para o *core* UDP/IP, contendo os seguintes campos:

- Tipo de pacote (1 *byte*): Esse campo é utilizado para diferenciar pacotes de dados e pacotes de configuração. Neste caso, para identificar pacotes de dados, este campo deve conter o valor ’11’ em hexadecimal para pacotes UDP ou o valor ’06’ para pacotes TCP.
- Tamanho do pacote (2 *bytes*): Tamanho total do pacote que está sendo transmitido para o *core* UDP/IP. Através deste campo se torna mais simples para determinar o tamanho do pacote que a aplicação está transmitindo.
- Porta de origem (2 *bytes*): Porta de origem do protocolo UDP da camada de transporte.

- Porta de destino (2 *bytes*): Porta de destino do protocolo UDP da camada de transporte.
- Endereço IP de destino (4 *bytes*): Endereço IP de destino do pacote. Esse campo será utilizado no protocolo IP pela camada de rede.
- Reservado (1 *byte*): Reservado para uso futuro.
- Dados (XX *bytes*): Esse campo irá conter os dados da aplicação. Se o pacote de dados for do tipo UDP, esse campo irá conter os dados da aplicação. E se for do tipo TCP, esse campo deverá conter o cabeçalho do protocolo TCP e os dados da aplicação.

Pacotes de dados que são enviados do *core* UDP/IP para a aplicação possuem a mesma estrutura exibida na Figura 27, mas ao invés do campo endereço IP de destino esses pacotes possuem o campo endereço IP de origem.

Pacote de dados – Aplicação para core UDP/IP							
Tamanho	**1 byte**	**2 bytes**	**2 bytes**	**2 bytes**	**4 bytes**	**1 byte**	**XX bytes**
Nome do campo	Tipo de pacote	Tamanho do pacote	Porta de origem	Porta de destino	Endereço IP de destino	Reservado	Dados ...

Figura 27: Estrutura dos pacotes de dados utilizados na comunicação da aplicação com o *core* UDP/IP.

O envio de pacotes da aplicação para o *core* UDP/IP é efetuado através dos seguintes pinos, que podem ser visualizados na Figura 25:

- *gtx_clk*: *Clock* de 125 MHz utilizado para o envio de pacotes.
- *wr_i*: Pino utilizado para iniciar o processo de escrita no *core* UDP/IP (*wr_i* = '1').
- *dv_i*: Pino de *data valid*. Se este pino estiver com o valor '1' indicará a presença de um dado válido no barramento *data_i*.
- *data_i[7:0]*: Barramento de dados de 8 *bits*. O *core* UDP/IP lê este campo somente quando o pino *dv_i* estiver em '1'.
- *wait_o*: Se o valor deste pino for '1', significa que o *core* está ocupado e não pode receber outro pacote no momento.

O diagrama de tempo ilustrado na Figura 28 demonstra como a aplicação deve enviar pacotes para o *core* UDP/IP. Nessa Figura é possível observar o envio de um pacote de configuração MAC *gateway* para o *core*. Para realizar o envio de um pacote de dados o comportamento do diagrama permanece o mesmo, alterando somente a estrutura do pacote que será enviado.

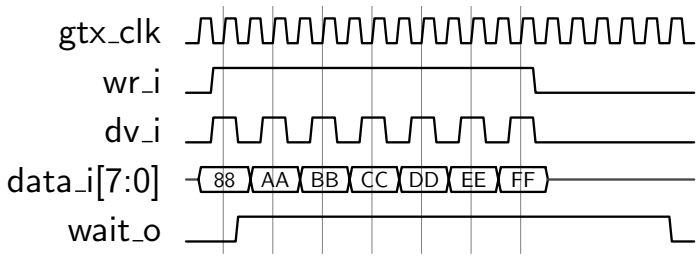


Figura 28: Diagrama de tempo da UDPIPv1 - Pacotes da aplicação para o *core* UDP/IP.

O recebimento de pacotes do *core* UDP/IP pela aplicação é efetuado através dos seguintes pinos, que também podem ser visualizados na Figura 25:

- *rx_clk*: *Clock* de 125 MHz utilizado para o recebimento de pacotes.
- *read_i*: Pino utilizado para iniciar o processo de leitura do *core* UDP/IP (*read_i* = '1').
- *dv_i*: Pino de *data valid*. Se este pino estiver com o valor '1' indicará que o *core* deve enviar o próximo dado, através do barramento *data_o*, pertencente ao pacote.
- *data_o[7:0]*: Barramento de dados de 8 *bits*. O *core* UDP/IP atualiza esse barramento após o recebimento de um *dv_i = '1'*.
- *int_o*: Se o valor deste pino for '1', significa que o *core* possui um novo pacote disponível e sua leitura pode ser realizada.

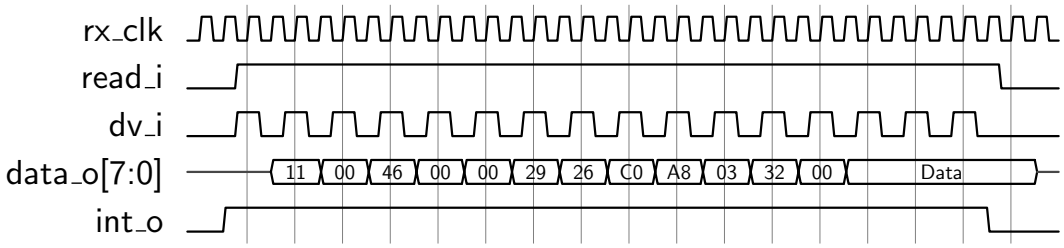


Figura 29: Diagrama de tempo da UDPIPv1 - Pacotes de dados do *core* UDP/IP para a aplicação.

O diagrama de tempo ilustrado na Figura 29 demonstra como a aplicação deve proceder para efetuar a leitura de um pacote de dados do *core* UDP/IP.

3.2.1.2 UDP *Receiver/Transmitter*

Esses blocos representam a camada de transporte e gerenciam pacotes UDP. Se o pacote recebido for do tipo TCP, esse dois blocos somente o repassam, não realizando nenhuma altera-

ção e nem remoção de cabeçalho. O bloco UDP *Receiver* realiza o desencapsulamento, ou seja, a verificação e remoção do cabeçalho UDP, e envia os dados para o bloco *Control Receiver*.

O bloco UDP *Transmitter* encapsula o pacote com o cabeçalho do protocolo UDP e envia para o bloco IP *Transmitter*.

3.2.1.3 IP *Receiver/Transmitter*

Representa a camada de rede e gerencia pacotes IPv4. O bloco IP *Transmitter* calcula o *checksum* e encapsula o pacote recebido do bloco UDP *Transmitter* com o cabeçalho IP e o envia para o bloco MAC *Transmitter*.

O bloco IP *Receiver* verifica o *checksum* e o endereço IP de destino do cabeçalho. Somente pacotes com o mesmo endereço IP de destino que o core UDP/IP e pacotes *broadcast* são aceitos e enviados para o bloco UDP *Receiver*. Se o pacote apresentar um *checksum* inválido será descartado.

3.2.1.4 MAC *Receiver/Transmitter*

O blocos MAC *Receiver* e MAC *Transmitter* representam a camada de enlace e gerenciam o recebimento e envio de quadros, respectivamente. O bloco MAC *Transmitter* envia o quadro para a interface GMII da PHY. No começo é enviado o preâmbulo, seguido do *start frame*. Em seguida, o cabeçalho MAC e os dados do quadro são enviados de 8 em 8 *bytes*. Cada *byte* é enviado para o bloco CRC *Generator* que calcula progressivamente o CRC do quadro. Este CRC é enviado para a PHY assim que o bloco MAC *Transmitter* terminar de enviar os dados do quadro.

O bloco MAC *Receiver* verifica se um novo quadro será recebido através da análise do preâmbulo e do pino *RXDV* da interface GMII da PHY. Se um novo quadro for detectado, o bloco CRC *Checker* é notificado e passa a calcular o CRC. Ao final do recebimento se CRC presente no quadro for diferente ao CRC calculado pelo bloco CRC *Checker*, o quadro será rejeitado. Somente quadros com o mesmo endereço MAC de destino que o core UDP/IP e quadros com endereço MAC *broadcast* são aceitos e repassados para o bloco IP *Receiver*.

3.2.1.5 RAM *Receiver/Transmitter*

Esses dois blocos são utilizados para armazenar os pacotes temporariamente e ambos possuem 1500 *bytes* de tamanho. O bloco RAM *Receiver* armazena o pacote até que a leitura seja

efetuada pela aplicação. E o bloco RAM *Transmitter* armazena o pacote até que o envio seja realizado pelos blocos responsáveis pela transmissão.

A memória desses dois blocos é implementada por *Block Random Access Memory* (BRAMs), que são memórias internas do dispositivo FPGA (XILINX, 2009d).

3.2.1.6 CRC *Checker/Generator*

Esse blocos são idênticos e ambos calculam o CRC baseados no polinômio gerador de 32 *bits* apresentado na Seção 2.1.3.1 do Capítulo 2.

O CRC provê detecção de erros decorrentes de algum erro na transmissão do quadro. Qualquer quadro que contenha um CRC inválido será descartado pelo bloco *Receiver*.

3.2.1.7 DCM

O bloco DCM, exibido na Figura 25, possui a mesma funcionalidade do bloco DCM do Testador apresentado na Seção 3.1.1.5, ou seja, implementa dois *Digital Frequency Synthesizer* (DFS). O primeiro DFS é utilizado para gerar um *clock* de 125 MHz para todos os blocos responsáveis pelo envio de pacotes e também para a interface GMII da PHY, conforme pode ser observado no pino *GTX_CLK* da Figura 25. O outro DFS gera um *clock* de 5 MHz que é utilizado pelo bloco GMII *Management*.

Os blocos responsáveis pelo recebimento utilizam o *clock RX_CLK* fornecido pela interface GMII da PHY. Isso pode ser observado na Figura 25.

3.2.1.8 GMII *Management*

Esse bloco gerencia a interface GMII da PHY para operar em modo *full-duplex* e 1000 Mbps, utilizando o pino *MDIO*. O pino *MDC* é o *clock* de 5 MHz gerado pelo bloco DCM.

Maiores detalhes sobre a interface GMII podem ser conferidos na Seção 2.1.3.3.

3.2.2 Arquitetura da pilha de comunicação UDP/IP Versão 2 - UDPIPv2

A segunda versão da pilha de comunicação UDP/IP (UDPIPv2) é baseada na UDPIPv1 e o intuito dessa versão é melhorar o desempenho da comunicação do *core* UDP/IP com a camada de aplicação, através do incremento do barramento dos pinos de dados (*data_i* e *data_o*) de 8 *bits* para 32 *bits*. Essa alteração e a estrutura da UDPIPv2 podem ser observadas na Figura 30.

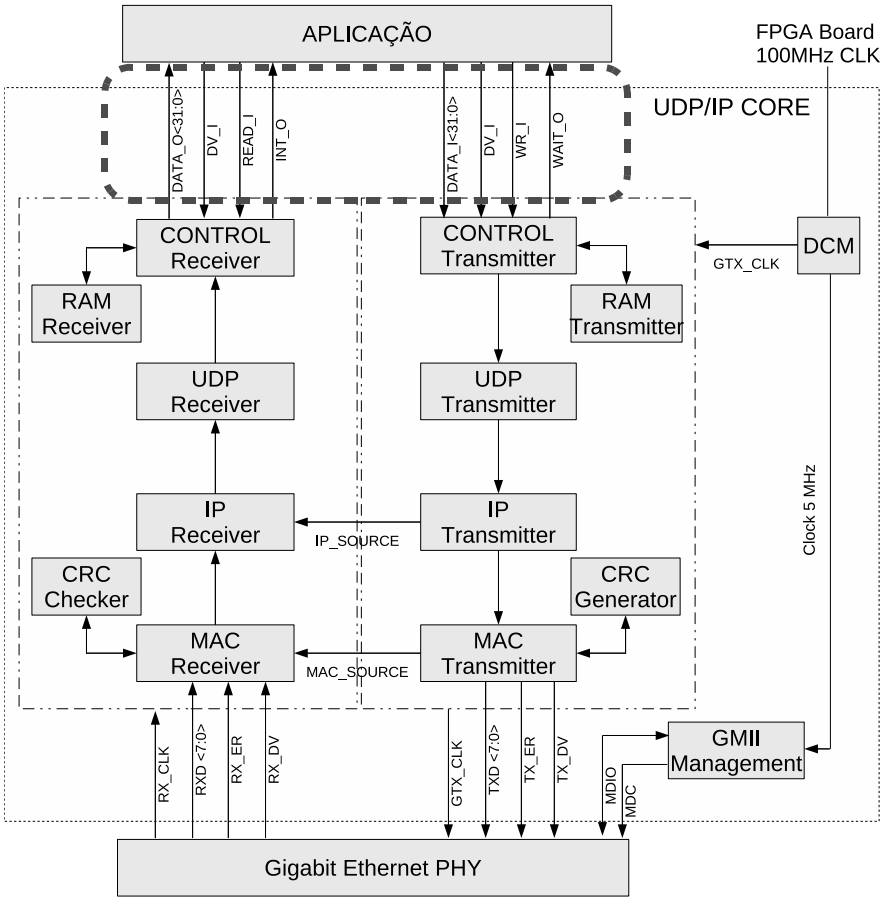


Figura 30: Diagrama de blocos da pilha UDP/IP versão 2 (UDPIPv2) desenvolvida em Verilog.

Conforme pode ser visualizado na Figura 30, a única alteração da UDPIPv1 para a UDPIPv2 é em relação ao barramento de dados utilizado na comunicação com a aplicação. Contudo, o tamanho do barramento de comunicação das memórias RAM *Receiver* e RAM *Transmitter* com os blocos *Control Receiver* e *Control Transmitter* também foi alterado para 32 *bits*. Todos os outros blocos permanecem inalterados.

3.2.2.1 Comunicação do *core* UDP/IP com a camada de aplicação

A comunicação do *core* UDP/IP com a camada de aplicação da UDPIPv2 continua baseada na troca de pacotes de configuração e dados. Porém, agora utilizando um barramento de 32 *bits*. As Figuras 31 e 32 apresentam os novos diagramas de tempo implementados nessa versão para a comunicação do *core* UDP/IP com a camada de aplicação.

A Figura 31 demonstra como a aplicação deve enviar pacotes para o *core* UDP/IP. E o diagrama de tempo da Figura 32 exibe como a aplicação deve proceder para efetuar a leitura de um pacote de dados do *core* UDP/IP. Conforme também pode ser observado nessas Figuras, a estrutura dos pacotes de configuração e dados permanece inalterada.

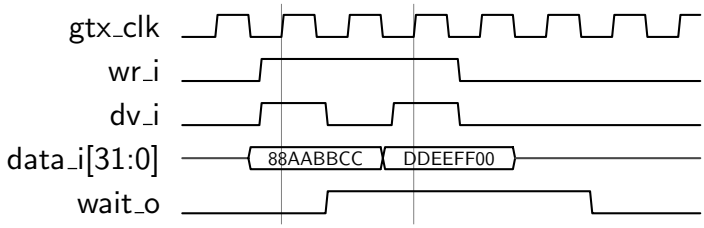


Figura 31: Diagrama de tempo da UDPIPv2 - Pacotes da aplicação para o *core* UDP/IP.

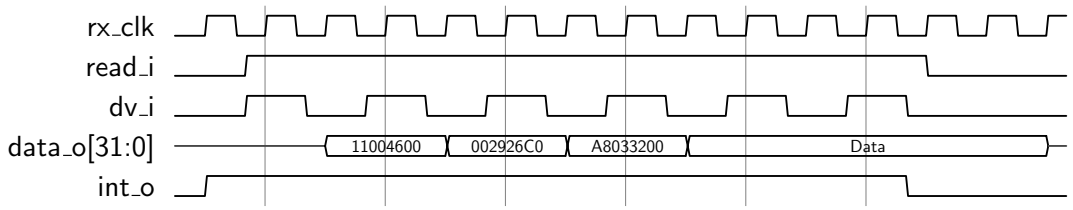


Figura 32: Diagrama de tempo da UDPIPv2 - Pacotes de dados do *core* UDP/IP para a aplicação.

3.2.3 Arquitetura da pilha de comunicação UDP/IP Versão 3 - UDPIPv3

A terceira versão da pilha de comunicação UDP/IP (UDPIPv3) é baseada na UDPIPv2 e o intuito dessa versão é garantir uma comunicação mais constante de dados, através da implementação de FIFOs para o recebimento e envio de pacotes. Dessa forma, o bloco *Control Transmitter* não mais necessita esperar até que o envio do pacote anterior tenha sido efetuado para só então receber um novo pacote da camada de aplicação. Da mesma forma, o bloco *Control Receiver* não mais necessita esperar que a camada de aplicação realize a leitura do pacote para só então receber um novo quadro da camada PHY. A alteração provocada na implementação dessas FIFOs e a estrutura da UDPIPv3 podem ser observadas na Figura 33.

3.2.3.1 FIFO *Receiver/Transmitter*

Esses dois blocos implementam FIFOs para o envio e recebimento de pacotes. Ambas as FIFOs utilizam, internamente, BRAMs para armazenar os pacotes, representadas na Figura 33 pelos blocos RAM. Essas BRAMs possuem uma profundidade de 1125 de 32 *bits* cada. As FIFOs *Receiver/Transmitter* dividem essas BRAMs em 3 partes com profundidade de 375 cada. Dessa forma, cada FIFO consegue armazenar 3 pacotes, sendo que o primeiro pacote ocupa da posição 1 até a 375, o segundo da posição 376 até a 750 e o último ocupa da posição 751 até a 1125.

Uma vez que a FIFO não armazena a quantidade de dados de um pacote, o controle de tamanho deve ser efetuado pelos blocos que instanciam as FIFOs. Neste caso o controle de

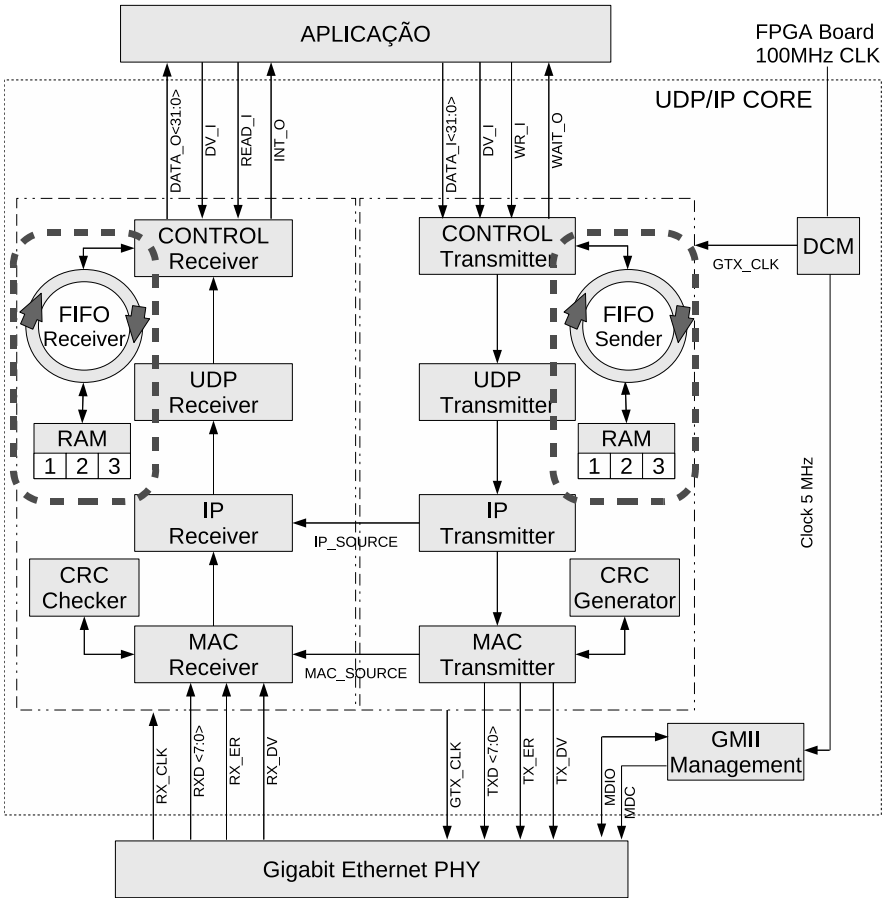


Figura 33: Diagrama de blocos da pilha UDP/IP versão 3 (UDPIPv3) desenvolvida em Verilog.

tamanho é efetuado pelos blocos *Control Receiver* e *Control Transmitter*, através do campo tamanho do pacote da estrutura exibida na Figura 27 da Seção 3.2.1.1.

A escrita e a leitura de pacotes nas FIFOs ocorre através dos pinos listados a seguir. Pinos de entrada:

- *clk*: Pino de *clock*.
- *rst*: Pino de *reset*.
- *wr_ena*: Quando o valor deste pino for '1' significa que o dado presente no barramento *wr_data* será gravado.
- *wr_new_packet*: Este pino é utilizado para indicar que um novo pacote será gravado. Deve permanecer com o valor '1' por somente um ciclo de *clock*. Porém, antes de efetuar a escrita de um pacote, o pino *full* deve ser verificado para garantir que a FIFO tenha espaço para receber um novo pacote (*full* = '0').
- *wr_data[31:0]*: Barramento de dados para serem gravados.

- *rd_ena*: Setando o valor deste pino para '1' indicará a leitura de um novo dado do pacote. A leitura do dado deve ser efetuada através do barramento *rd_data*.

- *rd_new_packet*: Pino utilizado para iniciar a leitura de um novo pacote armazenado na FIFO. Deve permanecer com o valor '1' por somente um ciclo de *clock*. Porém, antes de efetuar a leitura de um pacote, o pino *empty* deve ser verificado para garantir que existam pacotes na FIFO (*empty* = '0').

- *cancel_last_packet*: Este pino é utilizado para cancelar a escrita do último pacote. Por exemplo, quando um novo quadro começa a ser recebido pelo bloco MAC *Receiver*, este realiza o desencapsulamento e transmite os dados para o bloco IP *Receiver*. Esse processo de desencapsulamento e transmissão se repete até o bloco *Control Receiver*, que por sua vez inicia a armazenagem desse pacote no bloco FIFO *Receiver*. Porém, erros de CRC e *checksum* de cabeçalhos só são descobertos ao final da recepção de todo o quadro. Dessa forma, esse pacote precisa ser cancelado mantendo o valor desse pino em '1' por durante um ciclo de *clock*. O mesmo acontece com quadros recebidos da camada de aplicação pelo bloco *Control Transmitter*, onde podem ocorrer erros no tamanho do pacote que está sendo transmitido.

- *confirm_write_packet*: Mantendo o valor deste pino em '1' por durante um ciclo de *clock* indica que o pacote que foi gravado está correto e com dados válidos.

- *confirm_read_packet*: Ao final da leitura de pacote da FIFO, deve-se setar o valor deste pino para '1' por durante um ciclo de *clock* para confirmar a sua leitura.

Pinos de saída:

- *rd_data[31:0]*: Barramento de dados de saída da FIFO.

- *empty*: Esse pino indica que a FIFO está vazia quando o valor for '1'. Por exemplo, supondo que a FIFO contenha um ou mais pacotes (*empty* = '0'). Para efetuar a leitura de um pacote deve-se setar o pino *rd_new_packet* para '1' durante um ciclo de *clock*. Após este procedimento, o pino *rd_ena* deve ser setado para '1' para que a leitura dos dados do pacote seja efetuada. E ao final para confirmar a leitura do pacote o pino *confirm_read_packet* deve ser setado para '1'. Nesse momento a FIFO atualiza o pino *empty*, para '1' se ela estiver vazia ou para '0' se existirem mais pacotes presentes.

- *full*: Quando o valor deste pino for '1' indica que a FIFO está cheia. Antes de efetuar a gravação de um novo pacote, a situação deste pino deve ser consultada. Este pino

também só tem seu valor atualizado, pela FIFO, em função dos pinos de entrada *cancel_last_packet*, *confirm_write_packet* e *confirm_read_packet*.

3.3 Metodologia

As três arquiteturas da pilha de comunicação UDP/IP e o Testador desenvolvidos por este trabalho utilizaram a metodologia apresentada no fluxograma exibido pela Figura 34.

A etapa de especificação consiste do levantamento de todos os requisitos do projeto e foi baseada nos seguintes pontos propostos por (WESTE; HARRIS, 2004):

- Representação matemática do algoritmo: A especificação do algoritmo determina a complexidade do projeto e fornece uma idéia básica da área requerida.
- Número de pinos de entrada e saída e o número de *bits* de cada um: Com esta especificação é possível definir qual o tipo de interface que será utilizada no projeto, por exemplo, barramento de dados, controles de entrada e saída. Este item também determina o número de pinos que serão utilizados pelo projeto.
- Número de *bits* utilizados nas operações aritméticas internas: Este item previne as chances de ocorrer um *overflow* ou *underflow* e também reduz o efeito de acúmulo de erros para manter a precisão requerida pelo projeto. Também minimiza o custo e auxilia para que o projeto atenda aos requisitos de desempenho, pois é possível conhecer o número de somadores, multiplicadores e registradores necessários para a implementação do algoritmo.
- Número de sinais de *clock*: Este item é necessário porque o sinal de *clock* requer estruturas exclusivas do FPGA.
- Frequência máxima de *clock*: Define a velocidade de operação do projeto/circuito.

O *software* HDL Designer, fornecido pela empresa Mentor Graphics, foi utilizado no desenvolvimento de grande parte dos sistemas. Esse software auxiliou na codificação HDL, bem como na organização do trabalho.

Na etapa de simulação funcional e verificação lógica foi utilizado o *software* Modelsim, também fornecido pela empresa Mentor Graphics. Foram desenvolvidos *testbenches*, em Verilog, que injetavam estímulos na entrada da pilha de comunicação e analisavam as respostas obtidas na saída da pilha comparando-as então com as saídas esperadas. Esses *testbenches* geravam pacotes com diferentes tamanhos, diferentes configurações no cabeçalho e dados distintos.

Desta forma, foi possível automatizar o processo de verificação, o que torna o processo menos suscetível a erros humanos comuns em inspeções visuais.

Após o término da etapa de simulação e verificação foi utilizado o *software* ISE *Design Suite* versão 10.1, fornecido pela empresa Xilinx, para realizar a síntese, o mapeamento e o roteamento do código HDL para o FPGA alvo. No caso das arquiteturas da pilha UDP/IP foi utilizado o FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5. E para o Testador foi utilizado o FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4.

Na etapa de síntese são associados os pinos de entrada e saída do projeto com os pinos disponíveis no FPGA e também o mapeamento, posicionamento e roteamento do código HDL para as estruturas internas (células ou *slices*) do FPGA. Nesta etapa também foram utilizados os *softwares* EDK 10.1 e SDK 10.1 para implementação do Testador, ambos fornecidos pela empresa Xilinx. O EDK foi utilizado para integrar o blocos da camada de *hardware* com o barramento FSL e o *Microblaze* versão 7.10.d, conforme já foi explanado na Seção 3.1. E o *software* SDK foi utilizado para desenvolver os algoritmos, em linguagem C, que compõem o Testador, conforme pode ser verificado na Seção 3.1.2.2.

A etapa de simulação de *timing*, também conhecida como simulação *Post Place and Route*, utilizou os mesmos *testbenches* utilizados na etapa de simulação funcional e verificação lógica. Porém, esta etapa leva em consideração os atrasos gerados a partir do posicionamento e roteamento da lógica, gerada na etapa de síntese, para as células internas do FPGA. Dois caminhos para solução de qualquer problema encontrado por esta simulação foram adotados. O primeiro consistiu em tentar alterar algum parâmetro da etapa de síntese do projeto, e então realizar a síntese e novamente executar a simulação de *timing*. E o segundo, caso o problema não fosse solucionado pelo primeiro caminho, consistiu em retornar à etapa de codificação e tentar solucionar o problema encontrado pela simulação de *timing*.

Uma vez a simulação de *timing* não apresentando nenhum problema o próximo passo foi a geração do arquivo (*.bit*), contendo o *bitstream* para a configuração do FPGA. A gravação do arquivo *bitstream* foi efetuado através do *software* Impact fornecido pela empresa Xilinx.

A última etapa realizada foi a verificação funcional de teste em FPGA dos sistemas. Quando algum problema era detectado nesta etapa, retornava-se para a etapa de síntese, mapeamento e roteamento ou mesmo para a etapa de desenvolvimento HDL se o problema não fosse solucionado pela primeira opção. Esta última etapa também fez uso de um analisador lógico Agilent 1682AD. Detalhes sobre a verificação funcional das arquiteturas da pilha de comunicação UDP/IP e do Testador podem ser conferidos nas Seções 3.3.1 e 3.3.2, respectivamente.

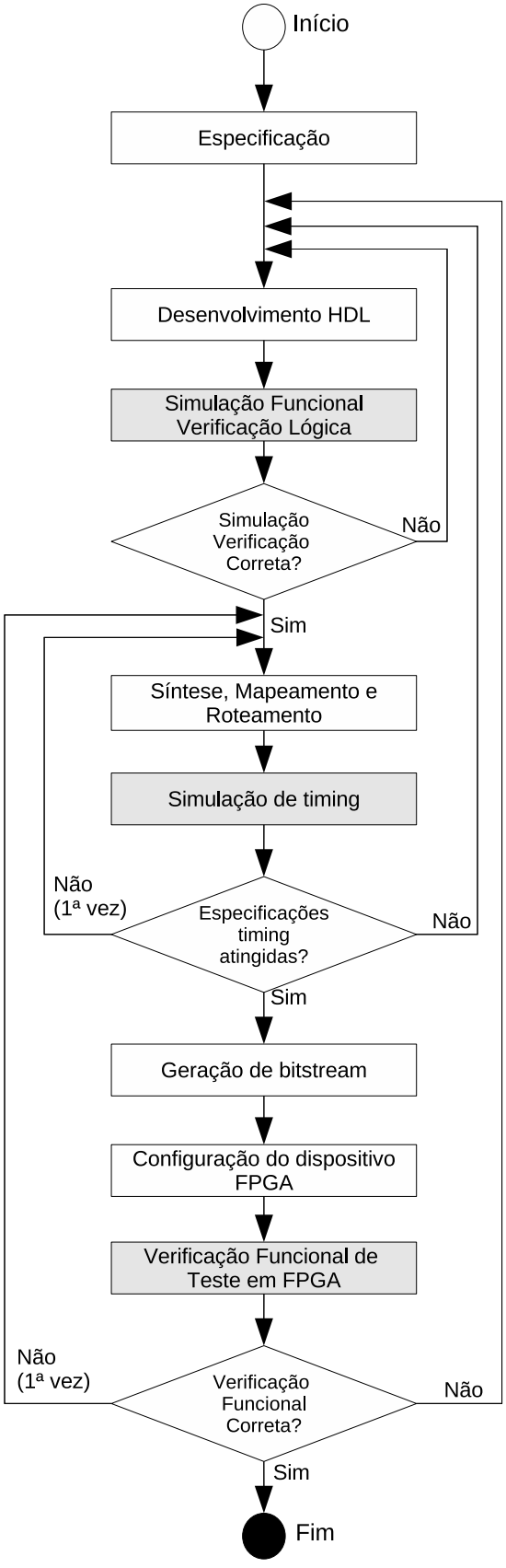


Figura 34: Fluxograma da metodologia utilizada por este trabalho para o desenvolvimento das arquiteturas da pilha de comunicação UDP/IP e o Testador. Figura adaptada de (WESTE; HARRIS, 2004).

3.3.1 Verificação Funcional de teste em FPGA das arquiteturas da pilha de comunicação UDP/IP

A verificação funcional das arquiteturas da pilha de comunicação UDP/IP consistiu em validar a funcionalidade dessas. Para isso, utilizou-se um computador com interface de rede *Gigabit Ethernet* conectado por cabo à interface de rede *Gigabit* do FPGA. No computador foi instalado o *software* hping (SANFILIPPO, 2010), que gera diversos tipos de pacotes baseado nos parâmetros fornecidos pelo usuário. Esses parâmetros podem ser: tamanho, *checksum* válido/inválido, tipo de protocolo, arquivo texto para transmissão, etc. Através deste *software* foram gerados pacotes e transmitidos pela interface de rede para o FPGA.

Uma vez que as arquiteturas implementadas num FPGA possuem a camada de aplicação realizando um *loopback*, então todos os pacotes enviados para o FPGA deveriam retornar para o computador. Com base nisso foi desenvolvido um *software*, em linguagem C, utilizando *sockets* do *kernel* do Linux, que foi responsável por receber os pacotes enviados pelo FPGA e analisar os dados recebidos. Se os dados recebidos pelo *software* no PC fossem diferentes dos dados que foram transmitidos para o FPGA, a verificação funcional falhava e o erro então era depurado.

3.3.2 Verificação Funcional de teste em FPGA do Testador

Na verificação funcional do Testador também foi utilizado um computador com interface de rede *Gigabit Ethernet* conectado por cabo à interface de rede *Gigabit* do FPGA. Porém, nessa verificação do Testador o computador não realiza o envio de pacotes para o FPGA e aguarda uma resposta, mas sim o Testador envia pacotes para o computador e aguarda uma resposta válida. Com base nisso foi desenvolvido um *software*, em linguagem C, utilizando *sockets* do *kernel* do Linux que foi responsável por receber os pacotes oriundos do Testador, analisar e validar a sua estrutura e então reenviá-los para o Testador para que este computasse os dados.

Porém, como o computador não possui capacidade para o recebimento de todos os quadros se o Testador utilizar uma vazão de, por exemplo, 100% foi desenvolvida em Verilog um contador e validador de pacotes e implementado em outro FPGA. Este FPGA foi conectado por cabo à interface de rede *Gigabit* do FPGA contendo o Testador e então os detalhes finais puderam ser testados com o auxílio de um analisador lógico Agilent 1682AD.

Para validação final, o Testador desenvolvido neste trabalho foi comparado com o dispositivo *JDSU SmartClass Ethernet* (JDSU, 2010). Esse Testador comercial, da empresa JDSU, implementa todos os testes sugeridos pela RFC 2544 para redes *Ethernet* de 10/100 Mbps e *Gigabit Ethernet*. Os resultados obtidos pelo Testador, implementado neste trabalho, para os

testes de vazão, latência e taxa de perda de quadros foram iguais aos resultados obtidos pelo Testador comercial.

3.3.3 Ambiente de Desenvolvimento e Obtenção dos Resultados

O ambiente de desenvolvimento utilizado na implementação das arquiteturas das pilhas de comunicação UDP/IP e do Testador, e também na obtenção dos resultados é apresentado na Figura 35. Esse ambiente é composto pelos seguintes itens:

1. Analisador lógico: Analisador lógico Agilent 1682AD utilizado na etapa de verificação funcional de teste em FPGA para depuração de erros relacionados ao Testador ou às pilhas de comunicação UDP/IP. Através dos pinos de teste do analisador lógico conectados ao FPGA também foi possível efetuar a verificação e validação do formato dos protocolos desenvolvidos, tanto para o Testador quanto para as pilhas de comunicação.

2. Pinos de teste: Pinos de teste do analisador lógico conectados à placa de desenvolvimento XUPV5-LX110T.

3. DUT: Placa de desenvolvimento XUPV5-LX110T dotada de um FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5 utilizado para implementação das arquiteturas da pilha de comunicação UDP/IP.

4. Testador: Placa de desenvolvimento ML402 dotada de um FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4 utilizado para implementação do Testador.

5. Cabo serial: Cabo RS232 conectado ao computador e utilizado na obtenção dos resultados reportados pelos algoritmos implementados no Testador.

6. Cabo de rede: Cabo de rede que conecta a interface de rede *Gigabit Ethernet* do Testador à interface de rede *Gigabit Ethernet* do DUT.

7. Computador: Computador utilizado durante as etapas de especificação, desenvolvimento, simulação e verificação do projeto. Este computador também é utilizado na captura dos resultados oriundos do Testador, através de um programa de monitoramento da porta RS232. Todos os dados capturados são armazenados em um arquivo texto para posterior verificação.

Através da análise dos itens 3, 4 e 6 da Figura 35 é possível concluir que o cenário utilizado para obtenção dos resultados através do Testador, é baseado no cenário mais usual proposto pela

RFC 2544, onde um único Testador envia os quadros para o DUT e coleta os quadros recebidos do DUT utilizando a mesma interface de rede. Este cenário e os outros cenários propostos pela RFC 2544 são apresentados na Figura 12 da Seção 2.2.

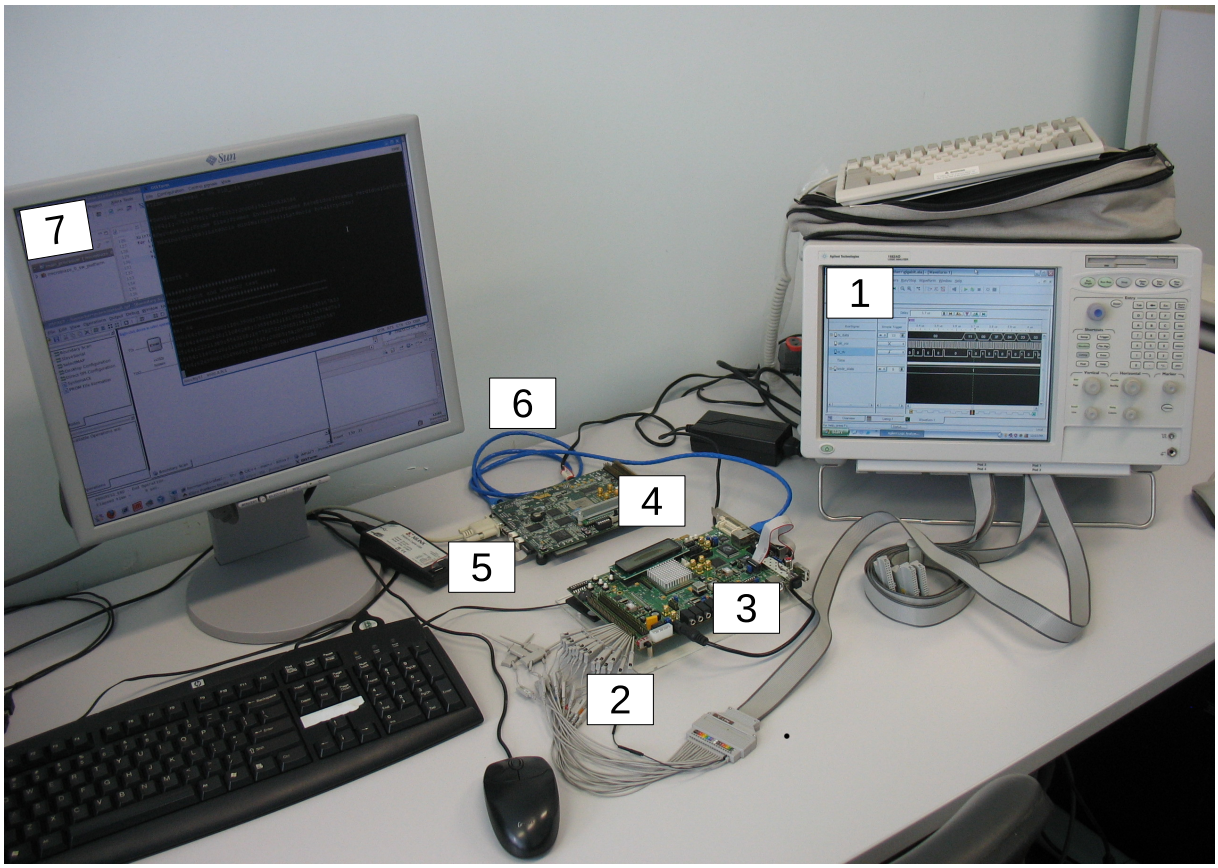


Figura 35: Ambiente de desenvolvimento e de obtenção dos resultados.

3.4 Resumo

Este Capítulo expôs detalhamente as arquiteturas propostas das pilhas de comunicação UDP/IP e do Testador. Foram apresentados os algoritmos desenvolvidos no Testador que implementam os testes de vazão, latência e taxa de perda de quadros sugeridos pela metodologia da RFC 2544. Também foram apresentadas as arquiteturas UDPIPv1, UDPIPv2 e UDPIPv3 da pilha de comunicação UDP/IP implementadas em *hardware* reconfigurável, bem como detalhes referentes à comunicação da camada de aplicação com essas três arquiteturas. Ao final do Capítulo foram apresentadas todas as etapas envolvidas no desenvolvimento, verificação e validação tanto do Testador quanto das pilhas de comunicação UDP/IP.

4 RESULTADOS OBTIDOS

Este capítulo apresenta os resultados de vazão, latência e taxa de perda de quadros, nas Seções 4.1, 4.2 e 4.3, respectivamente. Esses resultados são baseados na metodologia da RFC 2544 implementada neste trabalho pelo dispositivo Testador apresentado na Seção 3.1. São comparadas as três arquiteturas da pilha UDP/IP implementadas em FPGA: UDPIPv1, UDPIPv2 e UDPIPv3. E também uma aplicação desenvolvida na linguagem C que utiliza a pilha de comunicação do *kernel* do sistema operacional Linux. Essa aplicação, que nas comparações recebe o nome de PC, realiza somente um *loopback*, ou seja, reenvia os dados recebidos para a rede.

As características do computador, a versão do *kernel* e do sistema operacional utilizados para obtenção dos resultados de vazão, latência e taxa de perda de quadros referentes ao PC, podem ser visualizados na Tabela 9.

Tabela 9: Características do computador, versão do *kernel* e do sistema operacional utilizados para obtenção dos resultados de vazão, latência e taxa de perda de quadros relacionados ao PC.

Computador	IBM *System* x3200
CPU	*Dual core* Intel Xeon 3040 1.86GHz
Tamanho da *Cache*	2 X 2048 KB
Memória RAM	1 GB DDR2 667 MHz
Sistema Operacional	Kernel Linux 2.6.28-16-generic Ubuntu 9.04 x86_64 GNU/Linux
Interface de rede	Broadcom NetXtreme *Gigabit Ethernet* 10/100/1000

Este capítulo também apresenta, na Seção 4.4, os resultados relacionados à utilização dos recursos do FPGA, bem como uma comparação com os trabalhos relacionados apresentados na Seção 2.3 do Capítulo 2.

4.1 Resultados de Vazão

Para obter os resultados referentes à vazão, cada dispositivo testado (DUT) foi submetido ao Testador apresentado na Seção 3.1. O fluxograma do algoritmo para descoberta da vazão foi apresentado na Figura 23 da Seção 3.1.2.2. O teste se inicia enviando quadros a uma taxa de

100% para o DUT por durante 60 segundos. Se algum quadro for perdido nessa operação o teste é repetido a uma taxa mais baixa e continua até que a vazão máxima suportada pelo DUT seja encontrada. Consiste de uma busca binária, aumentando ou diminuindo os valores da vazão a cada passo em: 50%, 25%, 12%, 6%, 3%, 1%, 1%, etc. A taxa de quadros sofre um incremento ou decremento dependendo dos resultados obtidos no teste anterior.

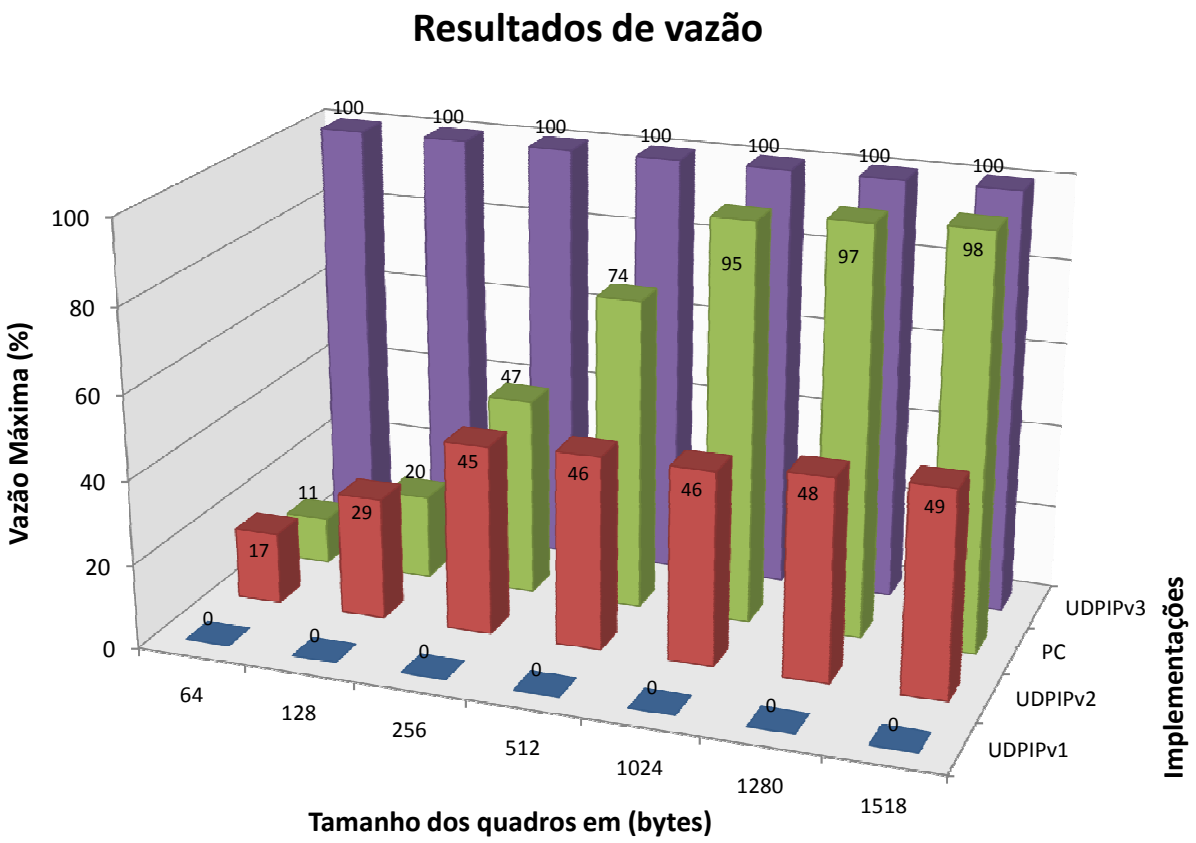


Figura 36: Comparação dos resultados de vazão entre os dispositivos testados para cada um dos tamanhos de quadro sugeridos pela RFC 2544.

O teste de vazão foi repetido 5 vezes para cada um dos quatro dispositivos testados PC, UDPIPv1, UDPIPv2 e UDPIPv3. Os melhores resultados obtidos por cada dispositivo para cada tamanho de quadro sugerido pela RFC 2544 podem ser visualizados no gráfico da Figura 36. Esse gráfico apresenta no eixo X o tamanho dos quadros, no eixo Y os valores referentes à vazão máxima atingida por cada dispositivo e no eixo Z as quatro implementações/dispositivos testados.

É possível observar que a única implementação que conseguiu atingir a vazão de 100% para cada um dos tamanhos de quadros é a UDPIPv3, ou seja, a UDPIPv3 não perdeu nenhum quadro durante os testes de vazão. Para a UDPIPv1 não foi possível determinar a vazão, pois essa versão apresentou perda de quadros para todas as taxas e para todos os tamanhos de quadros,

mesmo utilizando taxas de envio de somente 1%. Isso se deve ao fato dessa versão possuir um barramento muito pequeno (8 *bits*) para comunicação com a camada de aplicação, fazendo, portanto, com que não haja tempo hábil para efetuar a leitura de um pacote antes que um novo quadro chegue à camada MAC dessa arquitetura.

Por outro lado, a pilha de comunicação UDPIPv2, que possui um barramento de 32 *bits*, apresenta resultados melhores se comparada à UDPIPv1. Atingindo quase 50% de vazão para quadros a partir de 256 *bytes* de tamanho. No entanto, para quadros de 64 e 128 *bytes*, a UDPIPv2 também não consegue uma vazão satisfatória, atingindo somente 17 e 29%, respectivamente. Mesmo não atingindo uma vazão satisfatória, a UDPIPv2 possui vazão 6 e 9% maior que a vazão do PC para quadros de 64 e 128 *bytes*, respectivamente. Porém, para quadros a partir de 256 *bytes*, o PC começa a apresentar uma vazão superior, chegando a possuir 49% a mais de vazão que a UDPIPv2 para quadros de 1024, 1280 e 1518 *bytes*.

O principal problema que pode ser observado nos resultados do PC está relacionado com quadros de 64 *bytes*, onde a vazão máxima atingida é de somente 11%. No caso do PC, esse problema está relacionado ao processamento exigido para validação dos cabeçalhos dos quadros. Em quadros a partir de 1024 *bytes* de tamanho esse problema não ocorre, resultando em vazão superior a 95%, uma vez que, são necessários menos quadros para enviar a mesma quantidade de dados e consequentemente ocasionando uma menor sobrecarga no processamento. Isso se deve ao fato do IFG entre um quadro e outro ser muito pequeno para quadros de menor tamanho. Por exemplo, baseando-se no Apêndice A, para quadros de 64 *bytes* sendo enviados à uma vazão de 10% são necessários 148.810 quadros por segundo para transmitir a mesma quantidade de informação que 8.127 quadros com 1518 *bytes* conseguem transmitir.

4.2 Resultados de Latência

Juntamente com o teste de vazão é executado o teste de latência. Os resultados reportados nesta Seção comparam a latência considerando uma vazão de 100% para todos os dispositivos testados. Para dispositivos que não atingem a vazão de 100% é considerada a latência somente dos quadros recebidos. O teste é executado por 60 segundos e repetido 5 vezes. A latência média (ms) e o desvio padrão (ms) encontrados para os quatro dispositivos testados podem ser visualizados na Tabela 10.

Na Tabela 10 é possível observar que somente os testes realizados no PC apresentaram latência com desvio padrão significativo. Os testes realizados no FPGA com as implementações UDPIPv1, UDPIPv2 e UDPIPv3 apresentaram desvio padrão na faixa de nanosegundos (ns) e

Tabela 10: Resultados de Latência para os dispositivos testados e para cada um dos tamanhos de quadro sugeridos pela RFC 2544.

Tamanho dos quadros (bytes)	Vazão	Latência (ms)							
		PC		UDPIPv1		UDPIPv2		UDPIPv3	
		Média	Desvio padrão	Média	Desvio padrão	Média	Desvio padrão	Média	Desvio padrão
64	100%	2,6840	0,0057	0,0072	-	0,0052	-	0,0069	-
128	100%	3,2628	0,1329	0,0131	-	0,0071	-	0,0106	-
256	100%	2,4440	0,0111	0,0154	-	0,0086	-	0,0148	-
512	100%	0,7647	0,3946	0,0317	-	0,0098	-	0,0172	-
1024	100%	0,8257	0,4261	0,0645	-	0,0190	-	0,0253	-
1280	100%	0,7181	0,1564	0,0809	-	0,0236	-	0,0314	-
1518	100%	0,5404	0,2709	0,0961	-	0,0279	-	0,0419	-

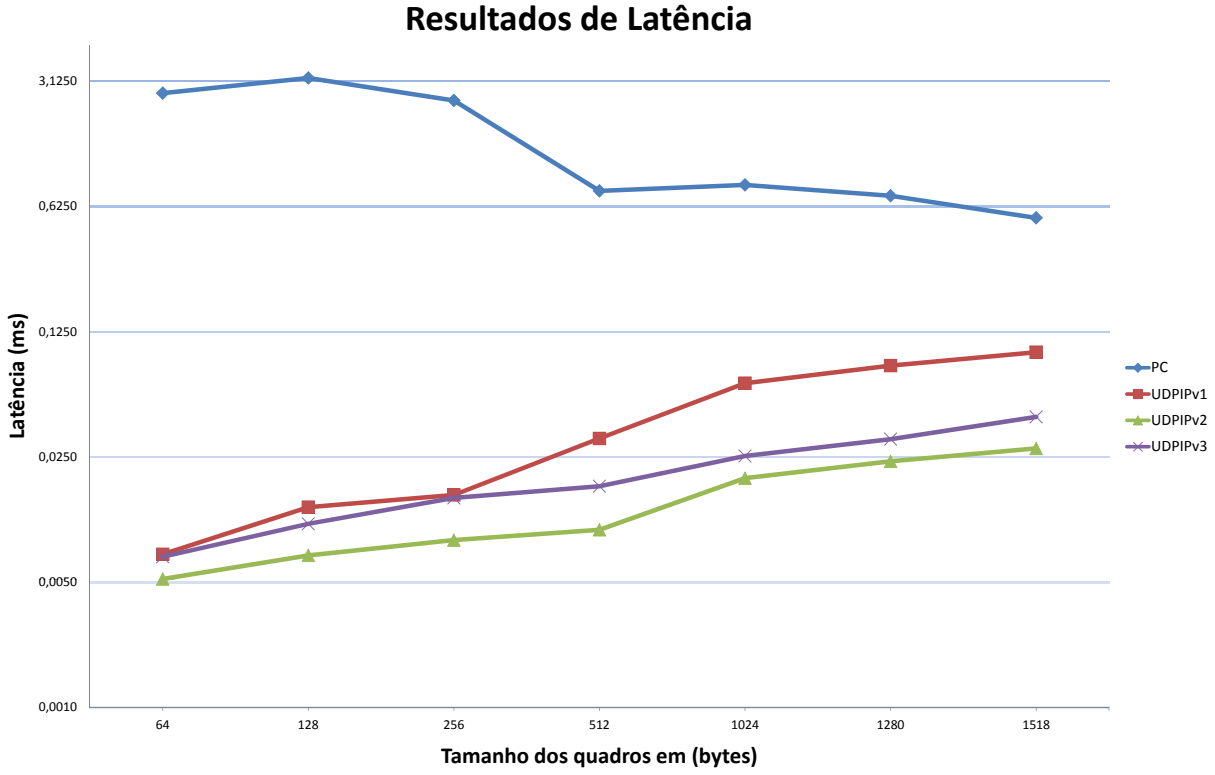


Figura 37: Comparação, em escala logarítmica, dos resultados de latência entre os dispositivos testados para cada um dos tamanhos de quadro sugeridos pela RFC 2544.

portanto não significativos para a escala de milisegundos (ms) utilizada nesta comparação.

O gráfico da Figura 37 apresenta, em escala logarítmica, a comparação das médias de latência obtidas para os dispostivos testados e para cada um dos tamanhos de quadro sugeridos pela RFC 2544. Esse gráfico apresenta no eixo X o tamanho dos quadros, e no eixo Y, em escala logarítmica, os valores referentes à latência média obtida por cada dispositivo testado.

Analisando os resultados apresentados no gráfico da Figura 37, é possível observar que, no caso do PC, quanto maior o tamanho do quadro, menor é a latência média obtida. Isso se deve à sobrecarga ocasionada no processamento para validação dos cabeçalhos de quadros

de tamanho pequeno, e também devido a cópia de dados no contexto do PC da memória de kernel para memória de usuário e depois para memória de kernel novamente para por fim ser enviado (CLARK et al., 1989), (KAY; PASQUALE, 1996), (CLARK et al., 2002). E no caso das implementações em FPGA, quanto maior o tamanho do quadro maior é a latência média obtida, uma vez que a validação dos cabeçalhos possui um tempo fixo, esse incremento da latência está relacionado ao tempo necessário para que a aplicação realize a leitura do quadro e o reenvie para os blocos responsáveis pelo envio do quadro para a rede.

Dentre as implementações em FPGA, a UDPIPv2 apresenta os melhores resultados de latência por possuir um barramento de 32 *bits* de comunicação com a aplicação e possuir *buffer* interno para armazenamento temporário de somente um único quadro. Porém, pelo fato de possuir *buffer* para somente um quadro, essa arquitetura não apresenta resultados de vazão e taxa de perda de quadros satisfatórios, conforme pode ser verificado nas Seções 4.1 e 4.3, respectivamente. Comparando a UDPIPv2 com a UDPIPv3 é possível observar que, para quadros de 1518 *bytes*, por exemplo, a UDPIPv3 é somente 1,5 vezes mais lento (0,014 ms) e no entanto consegue atingir a vazão máxima da tecnologia *Gigabit Ethernet*. A UDPIPv3 é mais lenta que a UDPIPv2 por possuir FIFOs que armazenam os quadros temporariamente, dessa forma, nenhum quadro é perdido por esta arquitetura, fato que não ocorre na UDPIPv2 pois esta arquitetura simplesmente descarta os quadros quando a memória estiver cheia, resultando portando, em uma latência menor. Porém, em uma aplicação real a perda de quadros acaba resultando numa retransmissão de quadros o que acaba provocando, consequentemente, uma latência maior. Dessa forma, é possível concluir que a UDPIPv3 é a melhor arquitetura também em termos de latência.

A UDPIPv1 mesmo possuindo a latência mais alta dentre as arquiteturas implementadas no FPGA, apresenta resultados melhores que a implementação no PC. Chegando a ser 374, 248 e 159 vezes melhor para quadros de 64, 128 e 256 *bytes*, respectivamente. No entanto, o problema da UDPIPv1 está relacionado à vazão, cujos resultados apresentados na Seção 4.1 demonstram que essa arquitetura apresenta vazão de 0% para todos os tamanhos de quadros.

Comparando a latência da UDPIPv3 com o PC, é possível afirmar que o ganho mais significativo se concentra nos quadros com 64, 128 e 256 *bytes* de tamanho. Para quadros de 64 *bytes*, a UDPIPv3 é em torno de 389 vezes mais rápida que a implementação no PC. E para quadros de 128 e 256 *bytes* este ganho é de 308 e 165 vezes, respectivamente. O ganho em latência diminui para quadros maiores, mas ainda se mantém alto. Por exemplo, quadros com 1518 *bytes* da UDPIPv3 apresentam latência 13 vezes menor que a latência do PC.

4.3 Resultados de Taxa de perda de quadros

Os resultados da taxa de perda de quadros foram obtidos através do Testador apresentado na Seção 3.1. O fluxograma do algoritmo para a taxa de perda de quadros foi apresentado na Figura 24 da Seção 3.1.2.2. O teste se inicia enviando, para cada tamanho de quadro, os quadros a uma taxa de 100% por durante 60 segundos. Essa operação é repetida para 95%, depois 90% e assim por diante. O teste é interrompido quando dois testes sucessivos não reportarem perda de quadros ou ao final da execução do teste de 5%. Cada dispositivo testado foi submetido a este teste por 5 vezes, as médias e o desvio padrão calculados podem ser visualizados na Tabela 11.

A Tabela 11 apresenta todos os resultados de taxa de perda de quadros para os quatro dispositivos testados. Nas linhas dessa Tabela constam as taxas de envio (vazão) utilizados no teste, de 100 a 5 % com granularidade de 5%. Essa Tabela está dividida em sete colunas principais que representam os tamanhos de quadro em *bytes* utilizados no teste, são eles: 64, 128, 256, 512, 1024, 1280 e 1518. Cada uma dessas colunas está dividida em cinco outras colunas representando as taxas de perda de quadros em percentual para os dispositivos testados. O PC ocupa duas colunas, pois exibe o valor do desvio padrão também em percentual para os 5 testes realizados.

Como é possível observar na Tabela 11, somente os testes realizados no PC apresentaram taxa de perda de quadros com desvio padrão significativo para os 5 testes realizados. Pois, além da pilha de comunicação, o sistema operacional também é responsável por atender outras tarefas, não garantindo, dessa forma, prioridade sobre a execução da pilha de comunicação. Os testes realizados nas arquiteturas da pilha UDP/IP UDPIPv1, UDPIPv2 e UDPIPv3, não apresentaram desvio padrão significativo. A diferença máxima encontrada entre um teste e outro foi de apenas 5 quadros, portanto, não significativa para estas comparações onde dezenas de milhares de quadros são trocados.

Nenhum dos testes realizados na arquitetura UDPIPv3 apresentou perda de quadros para qualquer tamanho de quadro, por isso não consta nenhum percentual informado para a UDPIPv3 na Tabela 11. O mesmo ocorre para algumas taxas de envio para os dispositivos PC e UDPIPv2. Por exemplo, para quadros de 64 *bytes*, o PC não perde quadros até uma vazão de 10% e a UDPIPv2 não perde quadros até uma vazão de 15%. Para quadros de 1518 *bytes*, o PC não perde quadros até uma vazão de 95% e a UDPIPv2 consegue suportar somente uma vazão até 45%.

A UDPIPv1 é a única implementação que apresenta taxa de perda de quadros para todas as taxas de envio testadas e para todos os tamanhos de quadros. Essa taxa de perda está acima

Tabela 11: Resultados de Taxa de perda de quadros para os dispositivos testados: PC, UDPIPv1, UDPIPv2 e UDPIPv3.

Taxa de envio - vazão (%)	Tamanho dos quadros (bytes)																							
	64				128				256				512											
	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3
	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)
100	88,10	0,04	90,73	89,53	-	91,71	3,56	83,66	82,45	-	81,52	0,08	86,00	66,67	-	24,25	0,24	87,75	66,67	-				
95	87,59	0,09	90,73	89,43	-	85,23	6,40	83,67	82,44	-	71,46	9,45	86,00	66,67	-	17,42	0,44	86,00	66,67	-				
90	86,87	0,05	88,87	86,97	-	91,93	7,13	80,40	79,17	-	74,86	0,16	83,66	50,00	-	10,75	0,32	86,00	50,00	-				
85	86,29	0,06	88,88	86,98	-	92,95	5,76	80,41	79,18	-	63,83	8,05	83,66	50,00	-	0,45	0,24	85,99	50,00	-				
80	85,27	0,04	88,87	86,97	-	94,80	0,04	80,40	79,17	-	64,92	0,16	83,66	50,00	-	0,03	0,01	83,65	50,00	-				
75	84,23	0,06	88,87	86,97	-	93,36	0,07	80,40	79,17	-	56,98	5,19	80,39	50,00	-	0,02	0,02	83,65	50,00	-				
70	83,11	0,07	86,09	84,59	-	92,43	0,03	75,49	75,49	-	49,49	7,63	80,38	50,00	-	-	-	83,65	50,00	-				
65	82,84	1,91	86,09	84,59	-	89,84	0,06	75,49	75,49	-	44,11	2,85	80,38	50,00	-	-	-	80,38	50,00	-				
60	86,15	6,48	86,08	84,58	-	80,78	5,65	75,48	75,48	-	36,82	0,42	75,48	50,00	-	-	-	80,38	50,00	-				
55	97,49	0,26	81,44	79,94	-	80,24	0,07	67,30	73,57	-	21,17	10,42	75,48	50,00	-	-	-	76,78	50,00	-				
50	83,90	6,96	81,44	79,94	-	70,29	8,29	67,30	66,22	-	14,01	0,79	75,47	50,00	-	-	-	75,47	50,00	-				
45	92,83	2,01	81,44	79,94	-	65,96	0,45	67,30	60,20	-	-	-	67,29	-	-	-	-	75,48	-	-				
40	87,97	5,27	81,45	79,95	-	51,73	8,06	67,31	50,00	-	-	-	67,29	-	-	-	-	67,30	-	-				
35	85,57	0,13	72,16	70,66	-	42,28	0,76	50,95	50,00	-	-	-	67,30	-	-	-	-	67,30	-	-				
30	77,87	0,28	72,16	68,33	-	16,97	7,45	50,95	0,12	-	-	-	50,95	-	-	-	-	67,30	-	-				
25	63,60	0,27	72,16	62,64	-	0,94	0,63	50,96	-	-	-	-	50,95	-	-	-	-	50,95	-	-				
20	42,70	0,13	72,17	53,69	-	-	-	50,96	-	-	-	-	50,96	-	-	-	-	50,95	-	-				
15	7,51	0,39	50,95	-	-	-	-	50,95	-	-	-	-	50,96	-	-	-	-	50,96	-	-				
10	-	-	44,34	-	-	-	-	1,93	-	-	-	-	1,91	-	-	-	-	1,90	-	-				
5	-	-	44,35	-	-	-	-	1,96	-	-	-	-	1,94	-	-	-	-	1,86	-	-				

Taxa de envio - vazão (%)	Tamanho dos quadros (bytes)																							
	1024					1280					1518													
	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3	PC		UDPIPv1	UDPIPv2	UDPIPv3
	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)	Taxa (%)	Desvio %	Taxa (%)	Taxa (%)	Taxa (%)
100	5,49	0,01	87,75	66,67	-	0,39	0,49	87,75	66,67	-	0,03	0,03	87,75	66,67	-									
95	-	-	87,74	66,67	-	-	-	87,75	66,67	-	-	-	85,99	50,00	-									
90	-	-	85,99	50,00	-	-	-	85,99	50,00	-	-	-	85,99	50,00	-									
85	-	-	85,99	50,00	-	-	-	85,99	50,00	-	-	-	85,99	50,00	-									
80	-	-	85,99	50,00	-	-	-	85,99	50,00	-	-	-	85,99	50,00	-									
75	-	-	83,65	50,00	-	-	-	83,65	50,00	-	-	-	83,65	50,00	-									
70	-	-	83,65	50,00	-	-	-	83,65	50,00	-	-	-	83,66	50,00	-									
65	-	-	80,38	50,00	-	-	-	83,66	50,00	-	-	-	83,66	50,00	-									
60	-	-	80,38	50,00	-	-	-	80,38	50,00	-	-	-	80,38	50,00	-									
55	-	-	80,38	50,00	-	-	-	80,39	50,00	-	-	-	80,39	50,00	-									
50	-	-	75,47	50,00	-	-	-	75,47	50,00	-	-	-	75,47	50,00	-									
45	-	-	75,48	-	-	-	-	75,47	-	-	-	-	75,49	-	-									
40	-	-	75,47	-	-	-	-	75,48	-	-	-	-	75,48	-	-									
35	-	-	67,31	-	-	-	-	67,30	-	-	-	-	67,29	-	-									
30	-	-	67,31	-	-	-	-	67,30	-	-	-	-	67,31	-	-									
25	-	-	50,95	-	-	-	-	50,94	-	-	-	-	50,94	-	-									
20	-	-	50,94	-	-	-	-	50,95	-	-	-	-	50,95	-	-									
15	-	-	50,96	-	-	-	-	50,96	-	-	-	-	50,94	-	-									
10	-	-	1,85	-	-	-	-	1,86	-	-	-	-	1,85	-	-									
5	-	-	1,89	-	-	-	-	1,87	-	-	-	-	1,86	-	-									

de 75% para as taxas de envio (vazão) superiores ou iguais a 60% para todos os tamanhos de quadro, e se mantém alta mesmo para uma vazão de somente 15% onde as taxas de perda ainda são superiores a 50% para todos os tamanhos de quadro.

No PC, os maiores percentuais de perda de quadros se concentram nos quadros com 64, 128 e 256 *bytes* de tamanho. Isso se deve ao processamento exigido para validação dos cabeçalhos dos quadros, exatamente o mesmo problema que ocorre nos testes de vazão e latência. Quanto ao desvio padrão, é possível observar que os índices mais altos também se concentram nestes tamanhos de quadro, chegando a 10,42% para quadros de 256 *bytes* e vazão de 55%. Já em quadros superiores a 512 *bytes* o desvio padrão se mantém abaixo de 0,5% para todas as taxas de envio.

Os gráficos exibidos nas Figuras 38 e 39, apresentam uma comparação da taxa de perda de quadros entre os dispositivos PC, UDPIPv1, UDPIPv2 e UDPIPv3 para quadros de 64 e 1024 *bytes* de tamanho, respectivamente. Esses dois gráficos apresentam no eixo X as taxas de envio, em percentual, utilizadas no teste, e no eixo Y as taxas de perda de quadros também em percentual para cada um dos dispositivos testados.

A UDPIPv3, conforme já citado anteriormente, não apresenta perda de quadros, portanto nos gráficos das Figuras 38 e 39 esta arquitetura aparece com as taxas de perda de quadros sobre o eixo X.

Para quadros de 64 *bytes*, no gráfico da Figura 38, a taxa de perda de quadros para o PC, a UDPIPv1 e a UDPIPv2 se mantém entre 80 e 90 % para taxas de envio superiores ou iguais a 60%. Porém, para taxa de envio de 55% a taxa de perda de quadros para o PC chega a quase 98% dos quadros transmitidos. E, analisando em conjunto com os dados exibidos na Tabela 11, é possível observar que o desvio padrão para os 5 testes executados para essa taxa de envio foi de somente 0,26%. Então, é possível concluir que este fato está claramente relacionado à maneira como o kernel do sistema operacional atende aos quadros recebidos e gerencia as interrupções e *buffers* em diferentes taxas de envio (BENVENUTI, 2005).

A UDPIPv1 e UDPIPv2 apresentam taxas de perda de quadros muito próximas para taxas de envio superiores ou iguais a 35%. Mas, para taxas de envio inferiores a 35%, a UDPIPv2 começa a apresentar resultados melhores, chegando a 0% de taxa de perda de quadros considerando uma vazão de 15%, enquanto a UDPIPv1 ainda apresenta taxa de perda de quadros superior a 50% considerando esta vazão.

Para quadros de 1024 *bytes*, no gráfico da Figura 39, o PC apresenta taxa de perda de quadros somente para vazão superior a 95%. Chegando, dessa forma, a resultados muito próximos aos apresentados pela UDPIPv3.

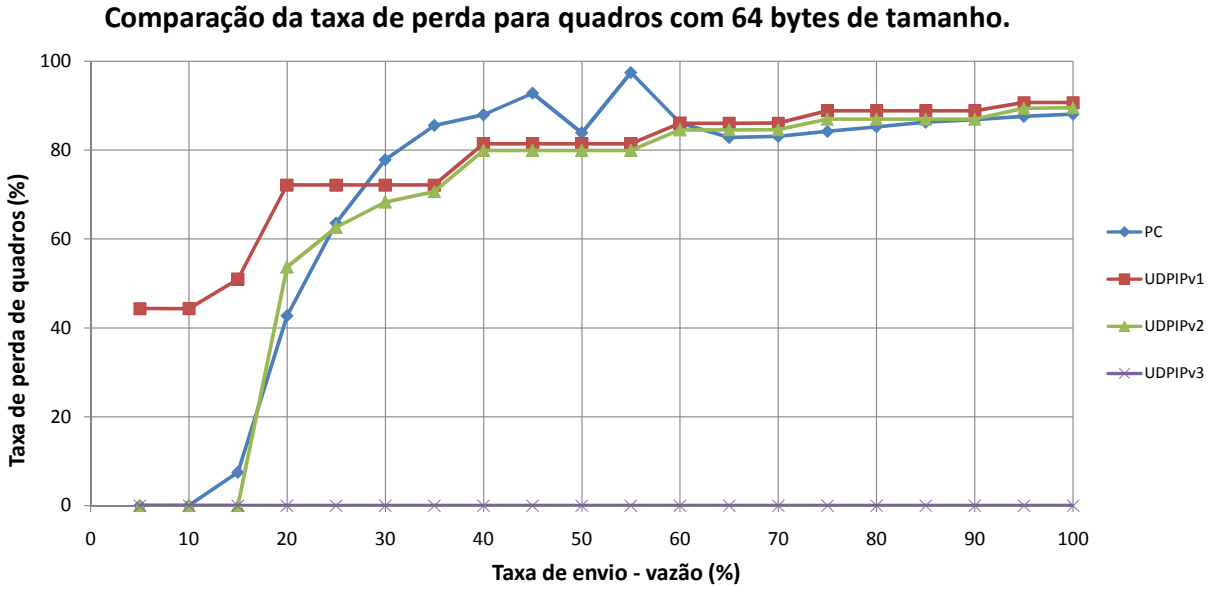


Figura 38: Comparação da taxa de perda para quadros com 64 *bytes* de tamanho.

Por outro lado, a UDPIPv1 apresenta os maiores índices de taxa de perda de quadros dentre os dispositivos testados, permanecendo acima de 50% de perda para vazão igual ou superior a 15%. E chegando próximo a 90% de perda considerando uma vazão de 100%.

Já a UDPIPv2 não apresenta perda de quadros para vazão de até 45%. Para vazão de 50% até 90% a taxa de perda de quadros se mantém próximo de 50%. E para vazão superior ou igual a 95% a taxa de perda atinge quase 70% dos quadros enviados.

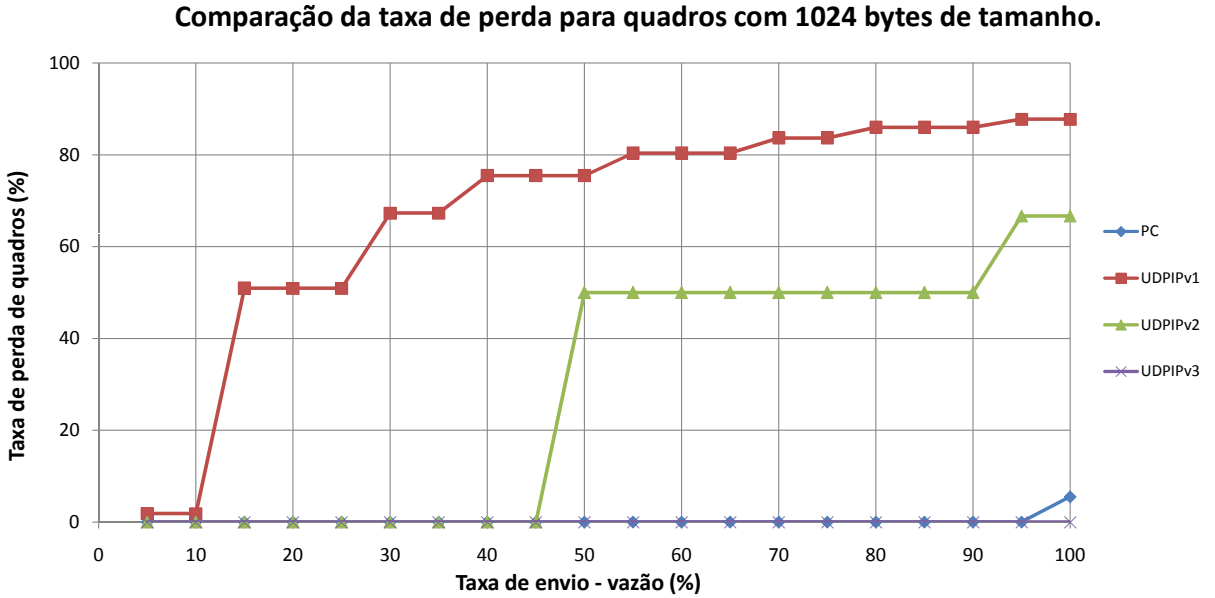


Figura 39: Comparação da taxa de perda para quadros com 1024 *bytes* de tamanho.

4.4 Utilização dos recursos do FPGA

A utilização dos recursos internos do FPGA pelas três arquiteturas da pilha de comunicação UDP/IP desenvolvidas neste trabalho, bem como a comparação com os trabalhos relacionados apresentados na Seção 2.3 do Capítulo 2 podem ser conferidos na Tabela 12. Além das três arquiteturas, UDPIPv1, UDPIPv2 e UDPIPv3 prototipadas num FPGA Virtex-5 XC5VLX110T, a Tabela 12 também apresenta resultados referentes as implementações "Minimum", "Medium"e "Maximum"de Löfgren (LÖFGREN et al., 2005) prototipadas num FPGA Spartan-3 XC3S200-4ft256, e também da implementação de Dollas (DOLLAS et al., 2005) prototipada num FPGA Virtex II XC2V8000-4FF1517.

As três arquiteturas implementadas por este trabalho apresentam a utilização de 1 DCM e 5 *Global Clock Buffers* (BUFGs). Conforme já foi mencionado anteriormente os DCMs são estruturas internas que provêem funcionalidades de *clock* nos FPGAs. Os BUFGs são utilizados automaticamente pela ferramenta de síntese para distribuir sinais de *clock* que possuam um elevado *fanout*. Nenhum dos trabalhos relacionados apresenta informações referentes a DCMs e BUFGs.

Quanto às BRAMs, o FPGA Virtex-5 XC5VLX110T possui 148 blocos de BRAM de 36 Kb cada, o que totaliza 5.328 Kb ou 666 KB de memória interna. A UDPIPv1 e a UDPIPv2 utilizam somente 1 dessas 148 BRAMs disponíveis. Já a arquitetura UDPIPv3 utiliza 4 BRAMs, porém apresenta resultados de vazão muito superiores e taxa de perda nulos se comparada aos resultados apresentados pelas arquiteturas UDPIPv1 e UDPIPv2. No FPGA Spartan-3 XC3S200-4ft256, utilizado por Löfgren, existem 12 blocos de BRAM de 18 Kb cada, o que totaliza 216 Kb ou 27 KB de memória interna. As implementações "Minimum"e "Medium"utilizam 3 BRAMs, enquando que a implementação "Advanced"utiliza 5 das 12 BRAMs disponíveis neste FPGA. No FPGA Virtex II XC2V8000, utilizado por Dollas, existem 168 blocos de BRAM de 18 Kb cada, o que totaliza 3024 Kb ou 378 KB de memória interna. A implementação de Dollas utiliza 10 das 168 BRAMs disponíveis neste FPGA.

Quanto à área ocupada, em termos de *xilinx slices* e LUTs, não é possível realizar uma comparação direta entre os trabalhos pois esses resultados variam de um FPGA para outro. O mesmo se aplica à frequência atingida pelas arquiteturas. Comparando as três arquiteturas desenvolvidas neste trabalho, pode-se observar que a UDPIPv1 ocupa 1.741 *xilinx slices*. As arquiteturas UDPIPv2 e UDPIPv3 utilizam 3,39% e 13,15%, respectivamente, a mais de *xilinx slices* que a UDPIPv1. Em relação às LUTs, a UDPIPv1 ocupa 1.825 das 69.120 disponíveis. E as arquiteturas UDPIPv2 e UDPIPv3 utilizam 0,55% e 27,95%, respectivamente, a mais de LUTs que a UDPIPv1.

O tamanho máximo de quadro suportado pelas implementações, UDPIPv1, UDPIPv2, UDPIPv3 e Löfgren "Advanced"é de 1518 *bytes*. Löfgren "Minimum"e "Medium"suportam quadros de no máximo 256 *bytes* e Dollas não apresenta informações relacionadas ao tamanho máximo de quadro. Quanto à velocidade *Ethernet*, as implementações, UDPIPv1, UDPIPv2, UDPIPv3 e Löfgren "Advanced"suportam 1 Gbps enquanto que Löfgren "Minimum", Löfgren "Medium"e Dollas suportam somente 10/100 Mbps.

Tabela 12: Comparação na utilização dos recursos do FPGA pelas arquiteturas da pilha de comunicação UDP/IP

Projetos	**FPGA Prototipação**	**Xilinx Slices**	**LUTs**	**BRAMs U / D**(*)	**BUFGs**	**DCMs**	**Tamanho máximo de quadro (*bytes*)**	**Velocidade *Ethernet***	**Frequência (MHz)**
UDPIPv1	Virtex-5 XC5VLX110T-3ff1136	1.741	1.825	1 / 148	5	1	1.518	1 Gbps	125
UDPIPv2	Virtex-5 XC5VLX110T-3ff1136	1.800	1.835	1 / 148	5	1	1.518	1 Gbps	125
UDPIPv3	Virtex-5 XC5VLX110T-3ff1136	1.970	2.335	4 / 148	5	1	1.518	1 Gbps	125
Löfgren "Minimum"	Spartan-3 XC3S200-4ft256	517	—	3 / 12	—	—	256	10/100 Mbps	90.7
Löfgren "Medium"	Spartan-3 XC3S200-4ft256	1022	—	3 / 12	—	—	256	10/100 Mbps	60.3
Löfgren "Advanced"	Spartan-3 XC3S200-4ft256	1584	—	5 / 12	—	—	1.518	1 Gbps	105.6
Dollas(**)	Virtex II XC2V8000-4FF1517	1557	—	10 / 168	—	—	—	10/100 Mbps	77
(*) U - Blocos Utilizados, D - Blocos Disponíveis (**) Deste projeto são considerados somente os valores referentes aos blocos UDP/IP, uma vez que este trabalho realiza a implementação de um *core* TCP/IP									

5 CONCLUSÃO

Este trabalho apresentou a implementação de três arquiteturas da pilha UDP/IP para *Gigabit Ethernet*, são elas: UDPIPv1, UDPIPv2 e UDPIPv3. Essas arquiteturas foram desenvolvidas na linguagem de descrição de *hardware* Verilog e implementadas em uma placa de desenvolvimento XUPV5-LX110T dotada de um FPGA Xilinx XC5VLX110T-3ff1136 da família Virtex-5. Para efeitos comparativos também foi desenvolvida uma aplicação na linguagem C que utiliza a pilha de comunicação do *kernel* do sistema operacional Linux. Essa aplicação realiza somente um *loopback*, ou seja, reenvia os dados recebidos para a rede.

Além da implementação das arquiteturas da pilha UDP/IP e da aplicação rodando sobre o PC, também foi desenvolvido um Testador baseado na metodologia da RFC 2544 e implementado em uma placa de desenvolvimento ML402 dotada de um FPGA Xilinx XC4VSX35-10ff668 da família Virtex-4. O conjunto de testes implementados no Testador engloba os testes de vazão, latência e taxa de perda de quadros.

Os resultados apresentados no Capítulo 4 demonstraram que a única implementação que conseguiu atingir 100% de vazão para todos os tamanhos de quadros testados é a UDPIPv3. Os resultados também demonstraram que o principal problema do PC está relacionado com quadros de tamanho pequeno. Por exemplo, o PC atingiu uma vazão de somente 11 e 20% para quadros de 64 e 128 *bytes*, respectivamente. Já em quadros a partir de 1024 *bytes* de tamanho esse problema não ocorreu, resultando em vazão superior a 95%. Para a UDPIPv1 não foi possível determinar a vazão, pois essa versão apresentou perda de quadros para todas as taxas e para todos os tamanhos de quadros, mesmo utilizando taxas de envio de 1%. Utilizando a arquitetura UDPIPv2 foi possível atingir vazão igual ou superior a 45% para quadros a partir de 256 *bytes*.

Em relação à latência, todas as arquiteturas implementadas em FPGA apresentaram latência muito menor do que uma pilha de comunicação implementada em um PC. A arquitetura UDPIPv3 apresentou 389 vezes menos latência que o PC para quadros de 64 *bytes*. E 13 vezes menos latência para quadros de 1518 *bytes*. Dentre as implementações em FPGA, a UDPIPv2 apresentou os melhores resultados de latência. Porém, foi possível observar que a UDPIPv3,

para quadros de 1518 *bytes* por exemplo, foi somente 1,5 vezes mais lenta (0,014 ms) que a UDPIPv2 e no entanto conseguiu atingir a vazão máxima da tecnologia *Gigabit Ethernet*. Já a arquitetura UDPIPv1 apresentou a latência mais alta dentre as implementações em FPGA.

Em relação à taxa de perda de quadros, a arquitetura UDPIPv3 foi a única implementação que não apresentou perda de quadros para nenhum dos tamanhos de quadro testados. A arquitetura UDPIPv1 apresentou taxa perda de quadros para todos os tamanhos de quadros e inclusive superiores a 50% para vazão de somente 15%. A arquitetura UDPIPv2 não perdeu quadros até uma vazão de 15 e 25% para quadros de 64 e 128 *bytes*, respectivamente. Para quadros a partir de 256 *bytes*, a UDPIPv2 não apresentou perda de quadros até uma vazão de 45%. No PC, os maiores percentuais de perda de quadros se concentraram nos quadros com 64, 128 e 256 *bytes* de tamanho. Já para quadros a partir de 1024 *bytes* de tamanho, o PC não perdeu quadros até uma vazão de 95%.

Os resultados relacionados à área no FPGA demonstraram que ocupando apenas 13,15 % e 27,95 % a mais de *xilinx slices* e LUTs do que a UDPIPv1, respectivamente, a arquitetura UDPIPv3 conseguiu atingir a vazão máxima da tecnologia *Gigabit Ethernet*, com latência menor e não apresentando perda de quadros.

5.1 Trabalhos Futuros

- Otimizar o aproveitamento de espaço das FIFOs da UDPIPv3. Atualmente a memória de cada FIFO está dividida em três partes iguais com profundidade de 375 posições e cada posição com 32 bits, ocupando dessa forma 1.500 *bytes* por parte. No entanto, pacotes de 64 *bytes*, por exemplo, ocupam apenas 4,26% desse espaço, fazendo com o restante dos 1.436 *bytes* não sejam utilizados. Uma forma de solucionar esse problema é a implementação de um módulo que funcionaria em paralelo às FIFOs, controlando o início e o fim de cada pacote dentro da FIFO. Com isso, no mesmo espaço onde antes eram alocados somente 3 pacotes de 64 *bytes*, agora seriam alocados 70 pacotes. Fato que deve ser levado em consideração, principalmente se a camada de aplicação, que atualmente está implementada em Verilog, for substituída por uma camada em *software* rodando sobre um microprocessador Microblaze, por exemplo.

- Implementação de uma *offload engine* utilizando UDPIPv3 em um computador. Realizar a implementação da pilha de comunicação em uma placa que seja conectável com um PC. Esse procedimento pode ser realizado utilizando a placa de desenvolvimento XUPV5-LX110T, ou seja, a mesma placa que foi utilizada para a implementação das

arquiteturas da pilha UDP/IP. Essa placa possui um barramento PCI *Express* que poderia ser utilizado para comunicação com a pilha implementada no FPGA, retirando dessa forma essa funcionalidade do microprocessador e consequentemente diminuindo a sobrecarga no processamento. A camada de aplicação, para efetuar comunicação em rede, seria responsável apenas por ler e escrever no barramento PCI de modo que todo o encapsulamento seria feito pela placa XUPV5-LX110T e não pelo sistema operacional como é realizado em um pilha de comunicação em *software*.

- Fragmentação de pacotes. O procedimento de fragmentação é utilizado pelo roteador toda vez que este precisa enviar um pacote para uma rede cujo MTU é menor que o tamanho do pacote a ser transmitido. Nenhuma das arquiteturas da pilha de comunicação UDP/IP implementadas neste trabalho possuem a fragmentação de pacotes implementada no protocolo IP da camada de rede. Se as arquiteturas implementadas receberem pacotes fragmentados, estas simplesmente irão tratá-los como pacotes únicos não provendo a junção dos dados para a camada de aplicação. Este fato impede, parcialmente, a utilização dessas arquiteturas em um ambiente real e que não fosse controlado. Parcialmente, pois a fragmentação de pacotes ocorre pouco em redes locais e muito menos em aplicações VoIP ou videoconferência que sempre utilizam pacotes pequenos para o envio de dados (GUO; LEE; LIN, 2007), (ABU-ALHAJ et al., 2009).

- Tabela ARP. *Address Resolution Protocol* (ARP) é um protocolo utilizado para encontrar um endereço MAC a partir do endereço IP. Geralmente cada máquina mantém uma tabela de resolução em cache para reduzir a latência e carga na rede. Porém, nenhuma das três arquiteturas deste trabalho possui tabela ARP implementada. Atualmente o endereço MAC da máquina destino é fornecido pela própria camada de aplicação através do pacote de configuração MAC *Gateway* apresentado na Seção 3.2.1.1. O fato de não possuir implementada uma tabela ARP pode, no caso de comunicação com vários endereços IPs diferentes, gerar uma carga extra na aplicação e consequentemente uma menor vazão e maior latência, uma vez que para cada endereço IP de destino diferente a aplicação deverá configurar o MAC *Gateway* do *core* UDP/IP.

- Jumbo Frames. Implementação do suporte a quadros do tipo *jumbo* (9.000 *bytes*) ou *super jumbo* (16.000 - 64.000 bytes) (RUTHERFORD et al., 2007) nas arquiteturas da pilha UDP/IP e no Testador. Através dessas alterações será possível verificar algumas tendências como, por exemplo, até que ponto a latência do PC diminui enquanto o tamanho do quadro aumenta.

- Implementação dos testes: *Back-to-back frames*, *System Recovery* e *Reset* no Testador. E

dessa forma efetuar esses testes em todos os dispositivos testados: PC, UDPIPv1, UDPIPv2 e UDPIPv3.

- Utilização da biblioteca de tempo real *Real Time Application Interface* (RTAI) (RTAI, Team, 2010) no PC, para garantir que a execução da pilha sofra o mínimo de interrupções e execute com prioridade máxima no sistema operacional. Um dos objetivos na utilização desta biblioteca é a redução do desvio padrão encontrado em quase todos o testes executados no PC.

- Desenvolvimento de um nova arquitetura movendo as FIFOs e os *buffers* (RAM) para a camada de enlace. Através dessa nova arquitetura seria possível aumentar a frequência (*clock*) das camadas superiores, ficando somente a camada de enlace com os 125 MHz necessários para comunicação *Gigabit Ethernet*. O intuito dessa nova arquitetura seria melhorar o desempenho em termos de latência.

REFERÊNCIAS BIBLIOGRÁFICAS

ABU-ALHAJ, M.; KOLHAR, M.; HALAIYQAH, M.; ABOUABDALLA, O.; SURESWARAN, R. Muxcomp - a new architecture to improve voip bandwidth utilization. In: *Future Networks, 2009 International Conference on*. [S.l.: s.n.], 2009. p. 212 –215.

ARM. *ARM922T*. 2010. Disponível em: <http://www.arm.com/products/CPUs/ARM922T.html>.

BABIARZ, J.; CHAN, K.; BAKER, F. *RFC 4594 - Configuration Guidelines for DiffServ Service Classes*. Agosto 2006. Disponível em: <http://www.rfc-editor.org/rfc/rfc4594.txt>.

BENVENUTI, C. *Understanding Linux Network Internals*. 1st edition. ed. [S.l.]: O'Reilly Media, 2005. Paperback. ISBN 0596002556.

BINKERT, N.; HSU, L.; SAIDI, A.; DRESLINSKI, R.; SCHULTZ, A.; REINHARDT, S. Performance analysis of system overheads in tcp/ip workloads. In: *Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on*. [S.l.: s.n.], 2005. p. 218 – 228. ISSN 1089-795X.

BLAKE, S.; BLACK, D.; CARLSON, M.; DAVIES, E.; WANG, Z.; WEISS, W. *RFC 2475 - An Architecture for Differentiated Service*. Dezembro 1998. Disponível em: <http://www.rfc-editor.org/rfc/rfc2475.txt>.

BRADNER, S. *RFC 1242 - Benchmarking Terminology for Network Interconnect Devices*. Julho 1991. Disponível em: <http://www.rfc-editor.org/rfc/rfc1242.txt>.

BRADNER, S.; BORMAN, D.; PARTRIDGE, C. *RFC 1071 - Computing the Internet Checksum*. Setembro 1988. Disponível em: <http://www.rfc-editor.org/rfc/rfc1071.txt>.

BRADNER, S.; MCQUAID, J. *RFC 2544 - Benchmarking Methodology for Network Interconnect Devices*. Março 1999. Disponível em: <http://www.rfc-editor.org/rfc/rfc2544.txt>.

CHUNG, S.-M.; LI, C.-Y.; LEE, H.-H.; LI, J.-H.; TSAI, Y.-C.; CHEN, C.-C. Design and implementation of the high speed TCP/IP offload engine. In: *Proceedings of the 7th International Symposium on Communications and Information Technologies*. [S.l.: s.n.], 2007. p. 574–579.

CLARK, D.; JACOBSON, V.; ROMKEY, J.; SALWEN, H. An analysis of tcp processing overhead. *Communications Magazine, IEEE*, v. 40, n. 5, p. 94–101, May 2002. ISSN 0163-6804.

CLARK, D. D.; JACOBSON, V.; ROMKEY, J.; SALWEN, H. An analysis of tcp processing overhead. *IEEE Communications Magazine*, v. 27, p. 23–29, 1989.

CLARK, K.; HAMILTON, K. *Cisco LAN Switching (CCIE Professional Development series)*. 1st edition. ed. [S.l.]: Publishing, 1999. Paperback. ISBN 1578700949.

COMER, D. E. *Internetworking with TCP/IP*. 5th edition. ed. [S.l.]: Prentice Hall, 2005. Paperback. ISBN 0131876716.

DAVIDSON, J.; PETERS, J.; BHATIA, M.; KALIDINDI, S.; MUKHERJEE, S. *Voice over IP Fundamentals*. 2nd edition. ed. [S.l.]: Cisco Press, 2006. Paperback. ISBN 1587052571.

DOLLAS, A.; ERMIS, I.; KOIDIS, I.; ZISIS, I.; KACHRIS, C. An open tcp/ip core for reconfigurable logic. In: *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*. [S.l.: s.n.], 2005. p. 297–298.

DOSTÁLEK, L.; KABELOVÁ, A. *Understanding Tcp/ip: A Clear And Comprehensive Guide*. [S.l.]: Packt Publishing, 2006. Paperback. ISBN 190481171X.

FAQS-RFC. *Internet RFC/STD/FYI/BCP Archives*. Janeiro 2010. Disponível em: <http://www.faqs.org/rfcs/>.

GADELRAB, S. 10-gigabit ethernet connectivity for computer servers. *Micro, IEEE*, v. 27, n. 3, p. 94 –105, May-June 2007. ISSN 0272-1732.

GUO, L.; LEE, W. C.; LIN, Y. ieh. On dynamic packet fragmentation for traffic integration over bandwidth-limited links. In: *Access Networks Workshops, 2007. AccessNets '07. Second International Conference on*. [S.l.: s.n.], 2007. p. 1 –8.

GUPTA, P.; LIGHT, A.; HAMEROFF, I. *Boosting Data Transfer with TCP Offload Engine Technology on Ninth-Generation Dell PowerEdge Servers*. 2006. Disponível em: <http://www.dell.com/downloads/global/power/ps3q06-20060132-Broadcom.pdf>.

HENRIKSSON, U. N. T.; LIU, D. CRC generation for protocol processing. *Norchip, Turku, Finland*, p. 288–293, 2000.

IANA. *Port Numbers*. 2009. Disponível em: <http://www.iana.org/assignments/port-numbers>.

IEEE. *IEEE 802.3 LAN/MAN Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. 2008. Disponível em: <http://standards.ieee.org/getieee802/download/802.3-2008_section1.pdf>.

IEEE. *Public EtherType Field listing*. 2010. Disponível em: <http://standards.ieee.org/regauth/ethertype/eth.txt>.

IEEE P802.3ba. *IEEE P802.3ba 40 Gbps and 100 Gbps Ethernet Task Force*. Setembro 2009. Disponível em: <http://www.webcitation.org/5k74bUEXk>.

IETF. *The Internet Engineering Task Force (IETF)*. Janeiro 2010. Disponível em: <http://www.ietf.org/>.

JDSU. *JDSU SmartClass Ethernet*. 2010. Disponível em: <http://www.jdsu.com/product-literature/smclassethgige_ds_acc_tm_ae.pdf>.

KAY, J.; PASQUALE, J. Profiling and reducing processing overheads in tcp/ip. *Networking, IEEE/ACM Transactions on*, v. 4, n. 6, p. 817–828, Dec 1996. ISSN 1063-6692.

KIM, D. W.; KWON, W. O.; PARK, K.; KIM, S. W. Internet protocol engine in tcp/ip offloading engine. In: *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*. [S.l.: s.n.], 2008. v. 1, p. 270–275. ISSN 1738-9445.

KIM, S.; PARK, C.; KIM, S.; CHUNG, Y. The offloading of socket information for tcp/ip offload engine. In: *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*. [S.l.: s.n.], 2009. v. 01, p. 826–831. ISSN 1738-9445.

KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach*. 5th edition. ed. [S.l.]: Addison-Wesley, 2009. Paperback. ISBN 0136079679.

LAM, P.-K.; LIEW, S. UDP-liter: an improved UDP protocol for real-time multimedia applications over wireless links. In: *1st International Symposium on Wireless Communication Systems, 2004.* [S.l.: s.n.], 2004. p. 314–318.

LÖFGREN, A.; LODESTEN, L.; SJOHOLM, S.; HANSSON, H. An analysis of fpga-based UDP/IP stack parallelism for embedded ethernet connectivity. In: *Proceedings of Norchip Conference, Oulu, Finland, 23rd*. [S.l.: s.n.], 2005. p. 94–97.

MARVELL. *88E1111 Product Brief Integrated 10/100/1000 Ultra Gigabit Ethernet Transceiver*. 2009. Disponível em: <http://www.marvell.com/products/tranceivers/alaska_gigabit_ethernet_transceivers/Alaska_88E1111-002.pdf>.

MOGUL, J.; DEERING, S. *RFC 1191 - Path MTU discovery*. Novembro 1990. Disponível em: <http://www.rfc-editor.org/rfc/rfc1191.txt>.

NICHOLS, K.; BLAKE, S.; BAKER, F.; BLACK, D. *RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. Dezembro 1998. Disponível em: <http://www.rfc-editor.org/rfc/rfc2474.txt>.

ORTIZ, A.; ORTEGA, J.; DIAZ, A.; PRIETO, A. A new offloaded/onloaded network interface for high performance communication. In: *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*. [S.l.: s.n.], 2009. p. 143 –150. ISSN 1066-6192.

PARKER, T.; SIYAN, K. *TCP/IP Unleashed*. 3rd edition. ed. [S.l.]: Publishing, 2002. Paperback. ISBN 0672323516.

PARZIALE, L.; BRITT, D. T.; DAVIS, C.; FORRESTER, J.; LIU, W.; MATTHEWS, C.; ROSSELOT, N. *IBM RedBook - TCP/IP Tutorial and Technical Overview*. Packt Publishing, 2006. Pdf. Disponível em: <http://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf>.

POSTEL, J. *RFC 768 - User Datagram Protocol*. Agosto 1980. Disponível em: <http://www.rfc-editor.org/rfc/rfc768.txt>.

POSTEL, J. *RFC 791 - Internet Protocol*. Setembro 1981. Disponível em: <http://www.rfc-editor.org/rfc/rfc791.txt>.

RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. *RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP*. Setembro 2001. Disponível em: <http://www.rfc-editor.org/rfc/rfc3168.txt>.

REALFAST. *RealFast HW/SW System Design - UDP/IP Cores High Speed Ethernet Communication Cores*. 2010. Disponível em: <http://www.realfast.se/rfipp/products/udpip/udpip.shtml>.

RFC-EDITOR. *Request For Comments Repository*. Dezembro 2009. Disponível em: <http://www.rfc-editor.org>.

RODRIGUEZ, A.; GATRELL, J.; PESCHKE, R. *TCP/IP Tutorial and Technical Overview*. 7th edition. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2001. ISBN 0130676101.

RTAI, Team. *RTAI - the RealTime Application Interface for Linux from DIAPM*. 2010. Disponível em: <https://www.rtai.org/>.

RUTHERFORD, W.; JORGENSON, L.; SIEGERT, M.; EPP, P. V.; LIU, L. 16 000-64 000 b pmtu experiments with simulation: The case for super jumbo frames at supercomputing '05. *Optical Switching and Networking*, v. 4, n. 2, p. 121 – 130, 2007. ISSN 1573-4277. Disponível em: <http://www.sciencedirect.com/science/article/B7GX5-4MD46C3-2/2/d52a67bbbed98b275d31fda645473ef5>.

SANFILIPPO, S. *Hping - Active Network Security Tool*. Fevereiro 2010. Disponível em: <http://www.hping.org/>.

SENAPATHI, S.; HERNANDEZ, R. *Introduction to TCP Offload Engines*. 2004. Disponível em: <http://www.dell.com/downloads/global/power/1q04-her.pdf>.

SPURGEON, C. *Ethernet: The Definitive Guide*. 1st edition. ed. [S.l.]: O'Reilly Media, 2000. Paperback. ISBN 1565926609.

STALLINGS, W. *Data and Computer Communications*. 8th edition. ed. [S.l.]: Prentice Hall, 2007. Paperback. ISBN 0132433109.

STEVENS, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. 5th edition. ed. [S.l.]: Addison-Wesley, 1994. Paperback. ISBN 0201633469.

TANENBAUM, A. S. *Computer Networks*. 4th edition. ed. [S.l.]: Prentice Hall, 2002. Paperback. ISBN 0130384887.

WANG, W.; LIEW, S. C.; LI, V. Solutions to performance problems in VoIP over a 802.11 wireless LAN. *Vehicular Technology, IEEE Transactions on*, v. 54, n. 1, p. 366–384, Jan. 2005. ISSN 0018-9545.

WEIDONG, L. *Designing TCP/IP Functions In FPGAs*. Dissertação (Mestrado), 2003.

WESTE, N.; HARRIS, D. *CMOS VLSI Design: A Circuits and Systems Perspective*. 3rd edition. ed. [S.l.]: Addison Wesley, 2004. ISBN 0321149017.

WU, Z.-Z.; CHEN, H.-C. Design and implementation of tcp/ip offload engine system over gigabit ethernet. In: *Computer Communications and Networks, 2006. ICCCN 2006. Proceedings.15th International Conference on*. [S.l.: s.n.], 2006. p. 245 –250. ISSN 1095-2055.

XILINX. *ML401/ML402/ML403 Evaluation Platform - User Guide*. [S.l.], Maio 2006.

XILINX. *MicroBlaze Processor Reference Guide*. 2008. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf>.

XILINX. *Digital Clock Manager (DCM) Module*. 2009. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/dcm_module.pdf>.

XILINX. *Spartan-3 Generation FPGA User Guide*. 2009b. Disponível em: <http://direct.xilinx.com/bvdocs/userguides/ug331.pdf>.

XILINX. Fast simplex link (FSL) bus (v2.11b). *Product Specification*, 2009c. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf>.

XILINX. *Block RAM (BRAM) Block (v1.00a) Data Sheet*. 2009d. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf>.

XILINX. *Xilinx, Inc. FPGA and CPLD Solutions*. 2010a. Disponível em: <http://www.xilinx.com>.

XILINX. *Xilinx Design Tools*. 2010b. Disponível em: <http://www.xilinx.com/tools/designtools.htm>.

XILINX. *Xilinx University Program XUPV5-LX110T Development System*. 2010c. Disponível em: <http://www.xilinx.com/univ/xupv5-lx110t.htm>.

XILINX INC. *DSP: Designing for Optimal Results, High-Performance DSP Using Virtex-4 FPGAs*. Xilinx, 2005. Pdf. Disponível em: <http://www.xilinx.com/publications/books/dsp/dsp-book.pdf>.

YEH, E.; CHAO, H.; MANNEM, V.; GERVAIS, J.; BOOTH, B. *Introduction to TCP/IP Offload Engine (TOE)*. 2002. Disponível em: <http://www.techonline.com/article/pdf/showPDF.jhtml?id=1931018381>.

APÊNDICE A – TAXA DE QUADROS E TAMANHO DO INTERFRAME GAP (IFG) PARA GIGABIT ETHERNET

Taxa de Quadros e Tamanho do IFG para Gigabit Ethernet (Página 1)

Parâmetros:

Ethernet Speed:	1.000.000.000 bps	Período clock:	0.008 us (8ns)
Preâmbulo:	8 bytes	Ciclos por byte:	1
IFG mínimo:	12 bytes		

Tamanho quadro (bytes)	Taxa de quadros (fps)									
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%
64	1.488.095	1.339.286	1.190.476	1.041.667	892.857	744.048	595.238	446.429	297.619	148.810
128	844.595	760.135	675.676	591.216	506.757	422.297	337.838	253.378	168.919	84.459
256	452.899	407.609	362.319	317.029	271.739	226.449	181.159	135.870	90.580	45.290
512	234.962	211.466	187.970	164.474	140.977	117.481	93.985	70.489	46.992	23.496
1024	119.732	107.759	95.785	83.812	71.839	59.866	47.893	35.920	23.946	11.973
1280	96.154	86.538	76.923	67.308	57.692	48.077	38.462	28.846	19.231	9.615
1518	81.274	73.147	65.020	56.892	48.765	40.637	32.510	24.382	16.255	8.127

Tamanho quadro (bytes)	100%			90%			80%		
	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)
64	857.142,86	0,10	12	771.428,57	0,17	21	685.714,29	0,26	33
128	918.918,92	0,10	12	827.027,03	0,23	28	735.135,14	0,39	49
256	956.521,74	0,10	12	860.869,57	0,34	43	765.217,39	0,65	81
512	977.443,61	0,10	12	879.699,25	0,57	71	781.954,89	1,16	145
1024	988.505,75	0,10	12	889.655,17	1,02	128	790.804,60	2,18	273
1280	990.769,23	0,10	12	891.692,31	1,25	156	792.615,38	2,70	337
1518	992.197,66	0,10	12	892.977,89	1,46	183	793.758,13	3,17	397

Tamanho quadro (bytes)	70%			60%			50%		
	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)
64	600.000,00	0,38	48	514.285,71	0,54	68	428.571,43	0,77	96
128	643.243,24	0,60	75	551.351,35	0,89	111	459.459,46	1,28	160
256	669.565,22	1,04	130	573.913,04	1,57	196	478.260,87	2,30	288
512	684.210,53	1,92	240	586.466,17	2,93	367	488.721,80	4,35	544
1024	691.954,02	3,68	459	593.103,45	5,66	708	494.252,87	8,45	1.056
1280	693.538,46	4,55	569	594.461,54	7,03	879	495.384,62	10,50	1.312
1518	694.538,36	5,37	671	595.318,60	8,30	1037	496.098,83	12,40	1.550

Taxa de Quadros e Tamanho do IFG para Gigabit Ethernet (Página 2)

Tamanho quadro (bytes)	40%			30%			20%		
	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)
64	342.857,14	1,10	138	257.142,86	1,66	208	171.428,57	2,78	348
128	367.567,57	1,87	234	275.675,68	2,86	357	183.783,78	4,83	604
256	382.608,70	3,41	426	286.956,52	5,25	656	191.304,35	8,93	1.116
512	390.977,44	6,48	810	293.233,08	10,03	1253	195.488,72	17,12	2.140
1024	395.402,30	12,62	1.578	296.551,72	19,58	2448	197.701,15	33,50	4.188
1280	396.307,69	15,70	1.962	297.230,77	24,36	3045	198.153,85	41,70	5.212
1518	396.879,06	18,55	2.319	297.659,30	28,81	3601	198.439,53	49,31	6.164

Tamanho quadro (bytes)	10%		
	Tempo envio (us)	Tempo cada IFG (us)	Tamanho IFG (Bytes)
64	85.714,29	6,14	768
128	91.891,89	10,75	1.344
256	95.652,17	19,97	2.496
512	97.744,36	38,40	4.800
1024	98.850,57	75,26	9.408
1280	99.076,92	93,70	11.712
1518	99.219,77	110,83	13.854

APÊNDICE B – ARTIGOS PUBLICADOS

B.1 Artigos publicados

HERRMANN, F. L.; PERIN, G.; DE FREITAS, J. P. J.; BERTAGNOLLI, R.; MARTINS, J. B. S.. **A Gigabit UDP/IP Network Stack in FPGA**. In: ICECS 2009 - 16th IEEE International Conference on Electronics Circuits and Systems, 2009, Yasmine Hammamet - Tunisia.

HERRMANN, F. L.; PERIN, G.; DE FREITAS, J. P. J.; BERTAGNOLLI, R.; MARTINS, J. B. S.. **An UDP/IP Network Stack in FPGA**. In: SFORUM 2009 - IX Microelectronics Students, 2009, Natal - Brazil.

TEIXEIRA, L.; AGUIRRE, P. C. C.; MÜLLER, C.; **HERRMANN, F. L.**; PIEPER, L. Z.; DE FREITAS, J. P. J.; DESSBESELL, G. F.; MARTINS, J. B. S.. **A Comparison between Hardware and Software Full Duplex Internet Protocol Version 4 Implementations**. In: XVI Iberchip Workshop, 2010, Iguaçu - Brazil.

AGUIRRE, P. C. C.; TEIXEIRA, L.; MÜLLER, C.; **HERRMANN, F. L.**; PIEPER, L. Z.; MARTINS, J. B. S.; DE FREITAS, J. P. J.; DESSBESELL, G. F.. **A Full Duplex implementation of Internet Protocol version 4 in a FPGA device**. In: SPL10 - VI Southern Conference on Programmable Logic, 2010, Porto de Galinhas, PE - Brazil.

B.2 Artigo publicado em ICECS 2009 - 16th IEEE International Conference on Electronics Circuits and Systems

A Gigabit UDP/IP Network Stack in FPGA

Fernando Luis Herrmann, Guilherme Perin, Josue Paulo Jose de Freitas, Rafael Bertagnolli and Joao Baptista dos Santos Martins

Federal University of Santa Maria (UFSM) - Microelectronics Group (Gmicro)
Post-Graduate Program in Informatics (PPGI)
Av. Roraima n. 1000, Santa Maria, Rio Grande do Sul, Brazil
Email: (herrmann, guilhermeperin, josue.freitas, rafaelbertagnolli)@mail.ufsm.br, batista@inf.ufsm.br

***Abstract*—This paper presents a proposal of a Gigabit UDP/IP network stack in FPGA, which is the stack of the widely used in VoIP and Video-conference applications. This network node implements the Network, Transport and Link Layer of a traditional stack. This architecture is integrated and developed using Xilinx ISE tool and synthesized to a Spartan-3E FPGA. We show architecture details, timing and area results of a practical prototyping. Also, we compare our prototype and results with other works in terms of area (Xilinx slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps). Comparing to these works our architecture obtained a intermediate solution in area and is the best implementation in terms of speed (MHz).**

***Index Terms*—UDP/IP, network stack, FPGA.**

I. Introduction

Nowadays, the great need of communication in society has collaborated to appear news forms of communication, that are more accessible and lower cost, for example Voice over IP (VoIP) or video conference. But, in a microprocessor of general purpose, this applications compete equally in processing time with other applications, causing a overload in the processing. In order to solve this problem, solutions implemented in dedicated hardware, ASICs or FPGAs become available. This solutions allow that part of the processing, instead of being realized by the microprocessor of general purpose, now can be executed by a dedicated hardware.

The UDP/IP protocol has applications on audio and video streaming of VoIP and Video Conference communications. Being real times conversations mode, the UDP protocol employment is explained by the need of low delay datagrams tranfer, due to unrealiable service [1] [2].

Some hardware UDP/IP stacks have already been realized. In [3], the author describes an analysis of FPGA-based UDP/IP stack parallelism for embedded Ethernet connectivity. In [4] an analysis of the TCP/IP sub-functions are made and the work describes the performance-critical functions that can be accelerated in FPGAs, how these sub-functions may be implemented and what speed-up gains that can be achieved. [5] Shown a RTP/UDP/IP protocol in Virtex FPGAs to accelerating VoIP applications. [6] Proposed five design guidelines and a corresponding architecture in TCP/IP Offload Engine (TOE). [7] Propose an implementation of UDP/IP protocol stack on FPGA and its performance evaluation.

This work presents a Gigabit UDP/IP network stack implementation in FPGA and makes a comparison with other existing works. For integration with the Application Layer, we also propose data and configuration packets formats. This paper is organized as follows. Section II presents an overview of the OSI model layers. The proposed implementation is detailed on Section III, while evaluation and results are exposed in Section IV. Finally, Section V presents our conclusion.

II. Network stack

The UDP/IP protocol is part of the Open Standards Interconnect (OSI) model. OSI is a theoretical model and is used to describe the behavior of a network and also to describe networking issues. The OSI model consists of seven layers and the layers are named (starting from the highest layer): Application, Presentation, Session, Transport, Network, Link and Physical Layer. From a TCP/UDP/IP viewpoint the Session and Presentation Layers are often included in the Application Layer. The OSI layers are frequently referred in this paper, but it's not further explained. For a detailed description of the layers and protocols, see [2].

A. Link Layer

As for serial links, the Link Layer provides data exchange between neighboring computers as well as data exchange between computers within a local network. For the Link Layer, the basic unit of data transfer is the data link packet frame. A data frame is composed of a header, payload, and trailer.

A frame carries the destination link address, source link address, and other control information in the header. The trailer usually contains the checksum of the transported data. By using the checksum, we can find out whether the payload has been damaged during transfer. The Network Layer packet is usually included in the payload.

B. Network Layer

The Network Layer is responsible for establishing the route to be used between the originating and destination computers. End-to-end reliability across several physical links is more of a function of the Transport Layer.

The basic unit of transfer is a datagram that is encapsulated in a frame. The datagram is also composed of a header and data field. Trailers are not very common in network protocols.

The datagram header, together with data (Network Layer payload), creates the payload or data field of the frame.

The Network Layer is used to establish communications with computer systems that lie beyond the local LAN segment. It can do so because it has its own routing addressing architecture, which is separate and distinct from the Link Layer. Such protocols are known as routed or routable protocols, for example Internet Protocol (IP) [8]. The Internet Protocol (IP) is the most important protocol of the Network Layer. IP attempts to deliver messages to the destination, which is selected by a unique IP address [9].

C. Transport Layer

The Network Layer make the connection between two remote computers easiear. As far as the Transport Layer is concerned, it acts as if there were no modems, repeaters, bridges, or routers along the way. The Transport Layer relies completely on the services of lower layers. It also expects that the connection between two computers has been established, and it can therefore fully dedicate its efforts to the cooperation between two distant computers. Generally, the Transport Layer is responsible for communication between two applications running on different computers [9].

In this case, the basic transmission unit is the segment that is composed of a header and payload. The Transport packet is transmitted within the payload of the network packet.

The Transport Layer provides end-to-end reliability by having the destination host communicate with the source host. The idea here is that even though lower layers of protocols provide reliable checks at each transfer, the end-to-end layer double checks to make sure that no machine in the middle failed [10].

The Transmission Control Protocol (TCP) is the most used protocol of Transport Layer which gives a connection-oriented communication with reliable data delivery, duplicate data suppression and flow control. Another Transport Layer protocol is the User Datagram Protocol (UDP), which provides an unreliable and connectionless communication service. However, UDP is very effective when TCP is not suited for the application needs, e.g. for real-time applications like audio and video or in applications where low latency and low delay is preferred over reliable data delivery [11].

III. Implementation

The hardware UDP/IP stack core that's described in this paper is shown in Figure 1. Figure 1(a) showed the architecture of a traditional TCP/IP stack. In Figure 1(b), as the shadowed area, is showed the UDP/IP stack which are entirely implemented in the FPGA. The Layers, Transport, Network and Link in UDP/IP stack are designed using Verilog and VHSIC Hardware Description Language (VHDL).

In our implementation the UDP protocol was implemented in the Transport Layer while TCP packets are passed directly to the Application Layer, to be processed by software.

For Network Layer the Internet Protocol version 4 (IPv4) is used, which gives a more area-effective design compared to the more recent IPv6 protocol.

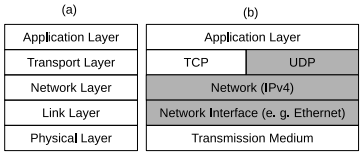


Fig. 1. (a) Architecture of a traditional TCP/IP stack. (b) Architecture used by this work and which layers are implemented. Redraw from [1].

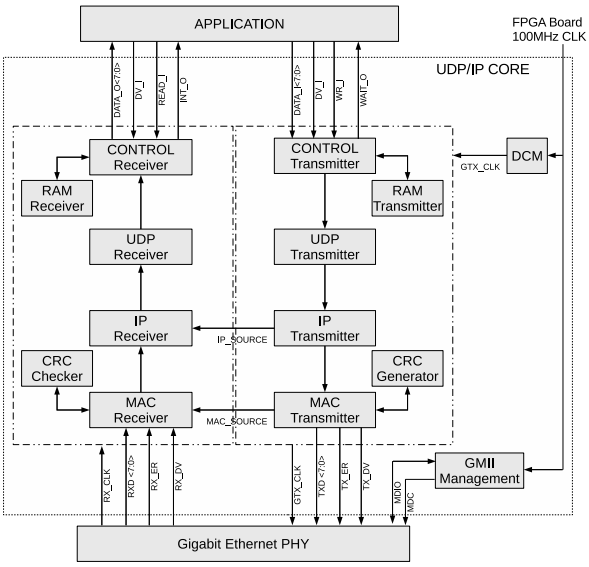


Fig. 2. Block diagram of the UDP/IP stack

The hardware UDP/IP stack implementation use the design structure which is showed in Figure 2. This implementation is full-duplex because the Transmitter and the Receiver works simultaneous and independent. The subsections below provide a detailed description of the functionality of each block.

A. Control Transmitter/Receiver

These blocks provide the communication with the Application Layer. The Control Transmitter receives the packet from the application and stores it in the RAM Transmitter for sending forward to the block UDP Transmitter.

The Control Receiver writes the packet from UDP Receiver in RAM Receiver and sends it to the application layer.

B. UDP Transmitter/Receiver

These blocks represent the Transport Layer and manage UDP packets. If the packet is a TCP, these two blocks just send the packet out. The block UDP Transmitter encapsulates the packet with the UDP header and sends out to block IP Transmitter.

The block UDP Receiver checks the packet and sends it to block Control Receiver without UDP header information.

C. IP Transmitter/Receiver

Represents the Network Layer manage IPv4 packets receiving and sending process. The IP Transmitter block calculates the checksum and encapsulate the packet with the IP header.

The IP Receiver block verifies the packet checksum and the destination IP-address. Only IP-address that matches the core's IP-address and broadcast IP-addresses are accepted and send to block UPD Receiver. If the packet check fails the packet will be rejected.

D. MAC Transmitter/Receiver

These blocks represent the Link Layer and manage the outcoming and incoming packets. The MAC Transmitter sends packets to the PHY. At the beginning, the preamble is sent, where the last nibble is a start of frame delimiter. The MAC transmitter then puts out the transmit packet to the PHY data bus and sets control signals. Each byte is sent to the CRC generator, which progressively calculates the CRC. When the packet end is reached the calculated 32-bit CRC is sent.

The MAC Receiver will check for a new packet (preamble from Ethernet PHY). Once a new packet is detected the CRC checker is notified and progressively calculates the checksum. When the end of frame is signaled from the Ethernet PHY the CRC check will be completed and the destination MAC-address will be verified. Only MAC-addresses that matches the UDP/IP stack MAC-address and broadcast MAC-addresses are accepted. If the packet check fails the packet will be rejected.

E. RAM Transmitter/Receiver

These blocks just temporarily store the packets and both has 1500 bytes in size.

F. CRC Checker/Generator

These blocks are identical and progressively calculate the Cyclic Redundancy Check (CRC). It uses the CRC32 polynomial for Ethernet. The polynomial is shown below:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} +$$

$$X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$$

A 32-bit CRC provides error detection in the case where line errors (or transmission collisions in Ethernet) result in corruption of the MAC frame. Any frame with an invalid CRC is discarded by the MAC Receiver without further processing. The MAC protocol does not provide any indication that a frame has been discarded due to an invalid CRC. The Link Layer CRC therefore protects the frame from corruption while being transmitted over the physical medium.

G. DCM

This is a Digital Clock Manager (DCM) and it implements a Digital Frequency Synthesizer (DFS) to generate a 125MHz clock, see pin GTX_CLK in Figure 2. The GTX_CLK clock is used by all the Transmitter blocks and also by the Gigabit Ethernet PHY.

H. GMII Management

This block manages the Gigabit Media Independent Interface (GMII) to full-duplex and 1000Mbps operations using the Management interface I/O pin (MDIO). This block also implements a DFS to generate a 5MHz clock for the Management Interface Clock pin (MDC). The GMII Standard interface is defined in IEEE Standard 802.3 [12].

I. Overall behaviour

The operation of the system is based on the exchange of configuration and data packets between the Application and UDP/IP stack, see Table I.

TABLE I
PACKETS TYPE

Packet	Type	Value (HEX)
UDP	Data	11
TCP	Data	06
MAC Source	Configuration	81
IP Source	Configuration	82
MAC Gateway	Configuration	88

The configuration packets are send by the Application to the UDP/IP stack and are used only for configure the IP Source, MAC Source and the MAC Gateway. The structure of this three configuration packets is showed in the Table II.

TABLE II
CONFIGURATION PACKETS

Type (8 bits)	Configuration
81	MAC Source (48 bits)
82	IP Source (32 bits)
88	MAC Gateway (48 bits)

The data packets are send by the Application to UDP/IP stack which in turn sends data packets to the Application. There are two data packets, TCP and UDP, and it has the structure as showed in Table III. The Table III shows the structure of the data packets that are send by the Application to the UDP/IP stack. The data packets that coming from the UDP/IP stack to the application has the same structure, but instead the Destination IP address it has the Source IP address.

TABLE III
DATA PACKETS - APPLICATION TO UDP/IP STACK

Bits	7 - 0	23 - 8	39 - 24
Field Name	Packet Type	Packet Length	Source Port
Bits	55 - 40	87 - 56	(...)
Field Name	Destination Port	Destination IP address	Data

The Figure 3 shows how the Application must be send the packets to the UDP/IP stack. This Figure shows the sending of a MAC Gateway configuration packet.

The Figure 4 shows how the UDP/IP stack send a packet to the Application. The pin int_o indicates that the UDP/IP stack has an packet available. Also, this Figure shows the a receiving of a UDP packet.

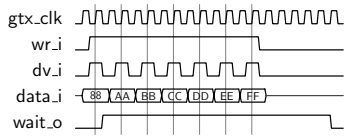


Fig. 3. Timing diagram - Application to UDP/IP stack

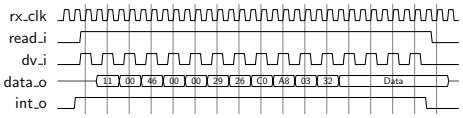


Fig. 4. Timing diagram - UDP/IP stack to Application

IV. Evaluation and Results

The UDP/IP stack was developed under Xilinx ISE 10.1 and passed through procedures of "Synthesis" and "Place and Route" for the Xilinx Spartan 3E, XC3S500e-4FG320 FPGA. Three simulation sub procedures were performed: "Functional", "Post Synthesis" and "Post Place and Route". Input values were supplied by macro files and test benches. Test benches also provided an overall report which includes signal comparison and signal sequence comparison, incoming TCP and UDP packets.

TABLE IV
Projects comparison

Projects	Xilinx Slices	Speed (MHz)	Max. Frame Length(bytes)	Max. Ethernet Speed(Mbps)
Our implementation	1580	125	1518	1000
Löfgren "Minimum"	517	90.7	256	100
Löfgren "Medium"	1022	60.3	256	100
Löfgren "Advanced"	1584	105.6	1518	1000
Dollas	1557	77	—	100

We compare our work in terms of area (Xilinx Slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps) with other implementations and we get the results that can be seen in the Table IV. The chosen implementations were Löfgren "Minimum", Löfgren "Medium" and Löfgren "Advanced" of [3] and Dollas in [13]. [13] Presents a complete TCP/IP stack implementation, therefore only related data to UDP/IP implementation were observed.

In occupied area terms, our implementation can be considered an intermediate solution. In speed terms our work presented the best performance. For the maximum Ethernet frame length only our implementation and Löfgren "Advanced" are capable to manage frames with 1518 bytes. Löfgren "Minimum" and Löfgren "Medium" just manage frames with maximum 256 bytes. Dollas doesn't provide information about maximum Ethernet frame length. For the maximum Ethernet speed only our implementation and Löfgren "Advanced" are support 1000Mbps connections. Löfgren "Minimum", Löfgren "Medium" and Dollas just support 100Mbps connections.

For example, if we compare our implementation with Löfgren "Minimum", we can observe that our work occupies three times more area than Löfgren Minimum and just 35 MHz better in speed, but our implementation is capable to processing frames with 1518 bytes and supports 1000Mbps connections while Löfgren "Minimum" only processing frames with 256 bytes and works just with 100Mbps connections.

V. Conclusion

This paper shows a Gigabit UDP/IP network stack in FPGA. We present a UDP/IP stack that is successfully implemented and verified in Xilinx Spartan 3E. The hardware UDP/IP network stack used 1,580 Xilinx slices of Xilinx Spartan 3E FPGA and its maximum frequency operation is 125MHz.

Also, we present a comparison with other four works, in terms of area (Xilinx slices), speed (MHz), maximum Ethernet frame length (bytes) and maximum Ethernet speed (Mbps). In occupied area, our implementation is an intermediate solution. But, in speed our work obtained the best results among the compared works. Depending on the application purposes, is well accepted more few elements in area terms to achieve a better performance in data communication. This is the case of VoIP commucation once the delay problem could be a relevant issue in many cases.

References

[1] A. Rodriguez, J. Gatrell, and R. Peschke, *TCP/IP Tutorial and Technical Overview*, 7th ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2001.

[2] A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, August 2002.

[3] A. Löfgren, L. Lodesten, S. Sjoholm, and H. Hansson, "An analysis of fpga-based UDP/IP stack parallelism for embedded ethernet connectivity," in *Proceedings of Norchip Conference, Oulu, Finland, 23rd*, Nov. 2005, pp. 94–97.

[4] L. Weidong, "Designing TCP/IP functions in FPGAs," Master's thesis, Code Number CE-MS-2003-09.

[5] A. Tavoularis, M. G. Manousos, D. Economou, and G. Lykakis, "Accelerating voip applications using virtex FPGAs," *FPGA and Structured ASIC Journal*, 2004.

[6] S.-M. Chung, C.-Y. Li, H.-H. Lee, J.-H. Li, Y.-C. Tsai, and C.-C. Chen, "Design and implementation of the high speed TCP/IP offload engine," in *Proceedings of the 7th International Symposium on Communications and Information Technologies*, Oct. 2007, pp. 574–579.

[7] K. Morita and K. Abe, "Implementation of UDP/IP protocol stack on FPGA and its performance evaluation," in *Proceedings of IPSJ General Conference*, 2001, pp. 157–158.

[8] K. S. Siyan and T. Parker, *TCP/IP Unleashed*, 3rd ed. Sams Publishing, August 2002.

[9] L. Dostálek and A. Kabelová, *Understanding Tcp/ip: A Clear And Comprehensive Guide*. Packt Publishing, April 2006.

[10] D. E. Comer, *Internetworking with TCP/IP*, 5th ed. Prentice Hall, July 2005, vol. 1.

[11] P.-K. Lam and S. Liew, "UDP-liter: an improved UDP protocol for real-time multimedia applications over wireless links," in *1st International Symposium on Wireless Communication Systems, 2004.*, Sept. 2004, pp. 314–318.

[12] *IEEE 802.3 LAN/MAN Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Std. 802.3, 2008.

[13] A. Dollas, I. Ermis, I. Koidis, I. Zisis, and C. Kachris, "An open TCP/IP core for reconfigurable logic," in *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2005. FCCM 2005.*, April 2005, pp. 297–298.