UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Rodrigo Rafael Melaré Corrêa

Síntese do subsistema de hardware para comunicação de dados com Gigabit Ethernet para o espectrômetro digital do CIERMag

São Carlos
2014

RODRIGO RAFAEL MELARÉ CORRÊA

Síntese do subsistema de hardware para comunicação de dados com Gigabit Ethernet para o espectrômetro digital do CIERMag

Dissertação apresentada ao Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para obtenção do título de Mestre em Ciências.

Área de concentração: Física Aplicada
Opção: Física Computacional
Orientador: Prof. Dr. Alberto Tannús

Versão Corrigida
(Versão original disponível na Unidade que aloja o Programa)

São Carlos
2014

Ficha catalográfica elaborada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Correa, Rodrigo Rafael Melare
Sintese do subsistema de hardware para comunicação de dados com Gigabit Ethernet para o espectrometro digital do CIERMag / Rodrigo Rafael Melare Correa; orientador Alberto Tannus - versão corrigida -- São Carlos, 2014.
147 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Física Básica) -- Instituto de Física de São Carlos, Universidade de São Paulo, 2014.

1. Espectrometro digital. 2. Gigabit Ethernet. 3. FPGA. I. Tannus, Alberto, orient. II. Título.

À minha esposa, com amor, admiração e gratidão por sua compreensão
e apoio ao longo da elaboração deste trabalho.

AGRADECIMENTOS

A Deus por me dar a sabedoria e capacidade necessária para realização deste trabalho e por me guiar profissionalmente.

A minha família pelo amor e carinho que sempre tiveram comigo.

A toda a equipe do CIERMag pela orientação e apoio, sem os quais este trabalho não teria sido concluído.

Ao Instituto de Física de São Carlos, pela oportunidade de realização do curso de mestrado.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pelo apoio financeiro para a realização desta pesquisa.

À Fundação de Amparo a Pesquisa do Estado de São Paulo sob o Projeto FAPESP 2005/56663-1.

RESUMO

CORRÊA, R. R. M. **Síntese do subsistema de hardware para comunicação de dados com Gigabit Ethernet para o espectrômetro digital do CIERMag**. 2014. 147 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.

Neste trabalho, é apresentado o desenvolvimento de um IP de rede Ethernet com interface para o barramento Avalon para utilização em conjunto com o processador Nios II da Altera. O IPC foi adaptado do projeto Ethernet_tri_mode, e é capaz de transferir dados a velocidades de 1000, 100 e 10 Mbps. O desenvolvimento envolveu a adaptação do código para atingir os requisitos do projeto, feito segundo as diretrizes do CIERMag de manter todo o código em VHDL. Além disso, foi implementada uma interface de comunicação com o processador Nios II para tornar possível a configuração do sistema, bem como a transferência de dados através de um software sendo executado no processador. O IPC Ethernet foi projetado para ser aplicado no espectrômetro digital em desenvolvimento pelo CIERMag e teve como compromissos a baixa utilização de recursos lógicos do FPGA e, ao mesmo tempo, a disponibilização de uma alta taxa de transferência de dados para o espectrômetro. Como ferramenta de desenvolvimento, foi utilizada a plataforma Quartus II cujo fornecedor é a Altera. Já os testes em placa foram realizados em um kit de desenvolvimento DE3-150 da Terasic, o qual utiliza uma FPGA Stratix III, também da Altera. Com o intuito de testar e validar o sistema, foi desenvolvido um software para o processador Nios II capaz de receber e enviar dados através do IPC e com inteligência para responder pedidos do tipo ARP e PING. O subsistema de Gigabit Ethernet desenvolvido aqui já incorpora a versão corrente do Espectrômetro Digital de RM do CIERMag.

Palavras-chave: Espectrômetro digital. Gigabit Ethernet. FPGA.

ABSTRACT

CORRÊA, R. R. M. **Hardware subsystem synthesis for data communication with Gigabit Ethernet for the digital spectrometer of CIERMag.** 2014. 147 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.

In this work we expose the implementation of an Ethernet network core which interfaces to Avalon bus used along with the Nios II Altera processor. This core was adapted from the Ethernet_tri_mode project. It can transfer data at rates of 1000, 100 and 10 Mbps. The development involved the adaptation of the code to fullfil the project requirements, under the policy of the CIERMag to keep the whole coding in VHDL. Furthermore was implemented an interface to communicate with the Nios II processor to enable system configuration and data transfer through a software running on the processor. The core was projected to be applied with focus on the utilization of low FPGA logical resources with the availability of a high data transfer rate. It will be used in a digital spectrometer under development at the CIERMag. The Quartus II platform, supplied by Altera was used as the development tool. The tests on board where carried out on a DE3-150 development kit from Terasic, which has an FPGA Stratix III also from Altera. In order to test and validate the system, a software for the Nios II processor was developed, able to send and receive data via IPC and with intelligence to answer ARP and PING types requests. The developed Gigabit Ethernet subsystem is now part of the running version of the CIERMag Digital MR Spectrometer.

Keywords: Digital spectrometer. Gigabit Ethernet. FPGA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALM	Adaptive Logic Module
ALU	Arithmetic and Logic Unit
ARP	Address Resolution Protocol
ASIP	Application-specific instruction-set processor
BSP	Board Support Package
CD	Collision Detection
CIERMag	Centro de Imagens e Espectroscopia in vivo por Ressonância Magnética
CRC	Cyclic redundancy check
CS	Carrier Sense
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DMA	Direct memory access
EDS	Embedded Design Suite
EOP	End of Packet
FCS	Frame Check Sequence
FPGA	Field Programmable Gate Arrays
HDL	Hardware Description Language
GMII	Gigabit Media Independent Interface
GPP	General Purpose Processor
HSMC	High Speed Mezzanine Connector
ICMP	Internet Control Message Protocol
IEEE	Instituto de Engenheiros Elétricos e Eletrônicos
IFG	Interframe gap
IFS	InterFrame Gap
IP	Internet Protocol
IPC	Intellectual property cores
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
LAN	Local Area Network
LCC	Logical Link Control
LUT	LookUp Table
MA	Multiple Access
MAC	Media Access Control
MII	Media Independent Interface
MMU	Memory Management Unit
MPU	Memory Protection Unit
OSI	Open Systems Interconnection
PING	Packet Internet Grouper
PIO	Parallel input/output
PLD	Programmable Logic Device

PROM	Programmable read-only memory
RGMI	Reduced Gigabit Media Independent Interface
RISC	Reduced Instruction Set Computer
RM	Ressonância Magnética
RMII	Reduced Media Independent Interface
RTL	Register Transfer Level
RTOS	Real Time Operacional System
SFD	Start Frame Delimiter
SOP	Start Of Packet
VHDL	Very High Speed Integrated Circuits Hardware Description Language
VN	Von Neumann

SUMÁRIO

1 INTRODUÇÃO

1.1 Resumo dos objetivos

Este projeto tem como objetivo principal o desenvolvimento de um *link* de comunicação com as seguintes características:

1. Alta taxa de transferência de dados, capaz de suprir as necessidades do espectrômetro digital;
2. Utilizar poucos recursos lógicos do *Field Programmable Gate Arrays (*FPGA);
3. Baixo *overhead* sobre o protocolo de transmissão de dados utilizado;
4. Não utilizar propriedade intelectual (IPC) de pessoas ou empresas externas, tendo assim domínio completo do projeto;
5. Utilizar a linguagem de descrição de *hardware Very High Speed Integrated Circuits Hardware Description Language (*VHDL) mais robusta;
6. Fornecer um aprendizado para utilização de processadores utilizados em FPGAs.

1.2 Justificativa

O uso de técnicas não invasivas e não destrutivas na investigação de patologias têm sido um dos desafios da medicina desde seus primórdios. Na segunda metade do século passado houve consideráveis avanços neste sentido, com a criação de uma técnica radiológica capaz de tomar imagens do interior do corpo humano, a Ressonância Magnética Nuclear (RM). Diferentemente da radiologia convencional e da tomografia computadorizada que utilizam radiação ionizante, a RM utiliza campos magnéticos e ondas de rádio para obter as imagens.

Com base nas vantagens agregadas desta técnica, o Centro de Imagens e Espectroscopia in vivo por Ressonância Magnética (CIERMag) está desenvolvendo um Espectrômetro Digital de Ressonância Magnética com a proposta de utilização principal, entre outras, em um *scanner* de RM (Projeto ToRM15).

Uma das barreiras encontradas neste projeto é a necessidade, por parte do espectrômetro, de trocar uma quantidade alta de dados com o computador onde serão

processados os dados, chamado de console de operação. Para ilustrar o problema, considere a aquisição de dados a partir da técnica Spin-echo Multislice. Neste caso em particular, tem-se 8 receptores que recebem 256 pontos em cada *Eco,* serão necessários 16 *Ecos* a cada 40 milissegundos para realizar o experimento. Sabendo, ainda, que cada ponto coletado é composto por 4 bytes, pode-se obter a taxa de transmissão necessária para o procedimento, no caso, esta corresponde a aproximadamente 25 Mbps. Porém, é do interesse do grupo utilizar outras técnicas de aquisição, de forma a variar o tempo de repetição e o número de receptores, o qual pode chegar a até 128. Esta variação torna a transferência de dados um ponto crítico do sistema.

Assim, tendo como objetivo fornecer um dispositivo de comunicação de dados de alta velocidade cuja taxa de transferência não seja debilitada pelo protocolo utilizado, foi desenvolvido um módulo de comunicação em Gigabit Ethernet, utilizando o protocolo UDP. O módulo será incorporado ao lado dos subsistemas já existentes de USB *High Speed* e *Full Speed*. Desta forma, o *hardware* programável FPGA ao executar uma sequência de pulsos compilada utilizando a linguagem de programação "F" de sequências de pulsos, desenvolvida no grupo, poderá manipular os sinais de Ressonância Magnética e enviá-los ao console de controle para processamento e visualização.

Outra abordagem do trabalho foi a incorporação, ao módulo Gigabit Ethernet, de um processador capaz de resolver protocolos do tipo ARP e PING, desta forma, a coerência dos dados pôde ser testada.

O módulo de comunicação foi desenvolvido em *hardware* digital, utilizando a diretriz do grupo de manter toda a linguagem de descrição de *hardware em* VHDL, sem nenhuma dependência em Propriedades Intelectuais estranhas ao Centro. Tal desenvolvimento permitiu ao grupo deter a propriedade intelectual sobre toda a tecnologia envolvida na síntese do hardware do Espectrômetro Digital.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Tecnologia Ethernet

Atualmente a padronização internacional dos protocolos para interconexão de sistemas abertos (sistemas que estão abertos à comunicação com outros sistemas) fica a cargo do Instituto de Engenheiros Elétricos e Eletrônicos (IEEE - *Institute of Electrical and Electronic Engineers*). Os padrões IEEE são organizados de acordo com o modelo de referências para interconecção de sistemas abertos (OSI - *Open Systems Interconnection*). Este modelo foi desenvolvido em 1978 pela organização internacional de padrões (ISO - *International Organization for Standardization*) para prover um esquema de organização comum para padrões de rede. O modelo de referências é um método que descreve como o conjunto de hardware e software interligado pode ser organizado para trabalhar em uma só rede. Na verdade, o modelo OSI prove um meio de dividir as tarefas de rede em pedaços separados.

Para fazer isso, o modelo descreve sete camadas de funções de rede, como ilustrado na Figura 2.1. As camadas inferiores definem os padrões que descrevem como os bits de um sistema de rede de área local (LAN - *Local Area Network*) circulam. Já as camadas mais altas lidam com noções mais abstratas, como a confiança da transmissão de dados e como o dado é representado para o usuário (1).

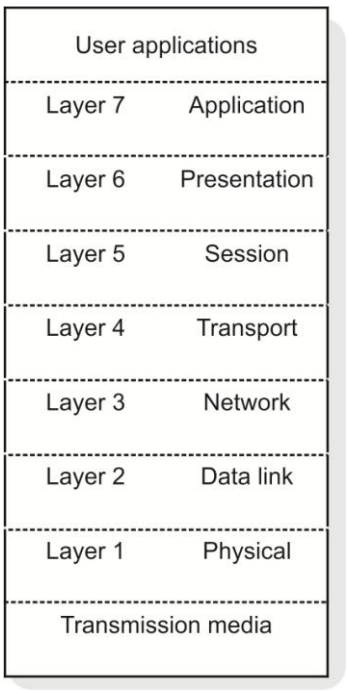


Figura 2.1 - As sete camadas do modelo OSI. Fonte: SPURGEON (1).

As camadas são descritas da seguinte forma:

i. Camada física

Lida com interfaces elétricas, mecânicas e de sincronização e com o meio físico de transmissão situado abaixo da camada física.

ii. Camada de enlace de dados

A principal tarefa desta é transformar um canal de transmissão bruto em uma linha que pareça livre de erros de transmissão não detectados para a camada de rede. Ela transmite e recebe *frames*, reconhece endereços de links, etc.

iii. Camada de rede

Responsável pelo endereçamento dos pacotes[*] de rede, também conhecidos por data gramas, esta associa endereços lógicos a endereços físicos, de forma que tais pacotes de rede consigam chegar corretamente ao destino. Ela também determina a rota na qual os eles irão percorrer para chegar ao sistema receptor.

iv. Camada de transporte

A função básica desta, em um sistema transmissor, é receber os dados da camada de seção, segmentar, enviar para a camada de rede e assegurar que todos os fragmentos chegarão corretamente à outra extremidade. Já em um sistema receptor o processo é inverso.

v. Camada de seção

Permite que diferentes máquinas estabeleçam seções entre elas. Fornecendo serviços como controle de diálogos, gerenciamento de token e sincronização.

[*] *O termo preciso* **definido** *no padrão Ethernet é "quadro", do inglês "frame", mas o termo "pacote" também é utilizado.*

vi. Camada de apresentação

Preocupa-se com a sintaxe e a semântica das informações. Converte o formato do dado recebido da camada de aplicação em um formato comum a ser usado na transmissão, ou seja, um formato entendido pelo protocolo usado.

vii. Camada de aplicação

Corresponde às aplicações (programas) no topo da camada OSI que serão utilizados para promover uma interação entre a máquina destinatária e o usuário da aplicação. A camada de aplicação também disponibiliza os recursos (protocolo) para que tal comunicação aconteça (2).

Os padrões Ethernet se restringem a descrever elementos nas camadas um e dois, o que inclui a camada de enlace de dados e a camada física. A Figura 2.2 mostra como as principais subcamadas das especificações IEEE estão organizadas.

No nível de enlace, existem as subcamadas de controle de link lógico (LCC - *Logical Link Control*) e o controle de acesso ao meio (MAC - *Media Access Control*), as quais são as mesmas para todos os tipos de Ethernet. A camada LLC é um mecanismo da IEEE para identificar os dados carregados no *frame* Ethernet enquanto que a subcamada MAC define o protocolo usado para controlar o acesso ao sistema Ethernet.

Já na camada física, as subcamadas do padrão IEEE variam de acordo com a padronização Ethernet e com a velocidade de operação (10 – 100 – 1000Mbps) (1).

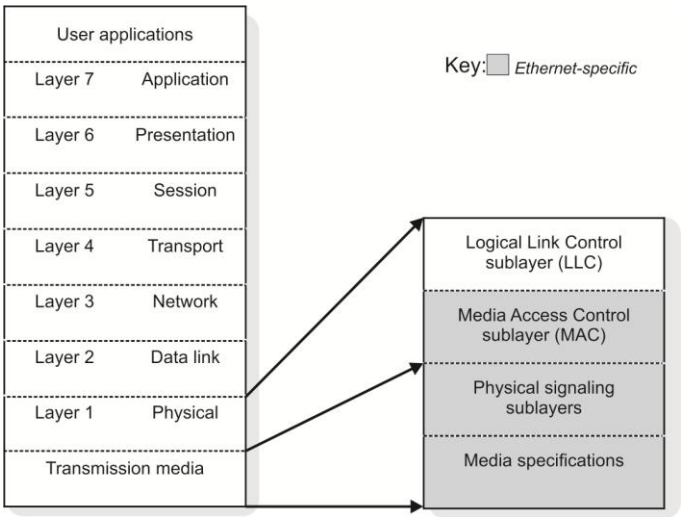


Figura 2.2 - As principais subcamadas do padrão IEEE. Fonte: SPURGEON (1).

Uma rede Ethernet de área local é composta por hardware e software trabalhando juntos para entrega de dados digital entre computadores. Para realizar esta tarefa são combinados quatro elementos, os quais formam um sistema Ethernet:

- Os **componentes de sinalização**, os quais são dispositivos eletrônicos capazes de receber e enviar sinais pelo canal Ethernet;

- O **meio físico**, o qual consiste de cabos e outros tipos de hardwares usados para carregar o sinal digital da Ethernet entre computadores conectados a rede;

- O ***frame***, o qual é um conjunto de bits padronizado utilizado para transportar os dados através do sistema;

- O **protocolo MAC**, que consiste de um conjunto de regras contidas em cada interface Ethernet que permite o acesso justo de vários computadores que compartilham o mesmo canal Ethernet.

Nas próximas subseções se abordará mais especificadamente a respeito de cada elemento.

2.1.1 Componentes de sinalização

Os componentes de sinalização para um sistema de par trançado incluem uma interface Ethernet localizada no computador, um *transceiver* e um cabo. A rede Ethernet é consistida de um par de estações ou mais, conectadas por segmentos de pares trançados, ou cabos, os quais interagem entre si utilizando a interface e o *transceiver*.

A interface Ethernet contém a eletrônica necessária para formar *frames* Ethernet de envio e recepção. Já o *transceiver* contém a eletrônica necessária para receber os sinais vindos da interface e transmiti-los pelo cabo, e para receber os sinais vindos do segmento de par

trançado e entregá-los a interface. A palavra *transceiver* vem da combinação das palavras em inglês *transmitter* e *receiver* (1).

2.1.2 O meio físico

Os cabos e outros componentes usados para construir a porção que transporta o sinal em um canal Ethernet compartilhado são chamados de meio físico. O sistema de cabeamento físico utilizado pode variar nos diferentes sistemas.

Uma mesma rede pode ser construída de modo que sistemas diferentes se comuniquem (1). Porém, neste trabalho o foco será o cabeamento por par trançado.

2.1.3 O *frame* Ethernet

A

Figura 2.3 mostra a divisão dos campos de um *frame* Ethernet de acordo com o padrão IEEE 802.3.

64 bits	***48 bits***	***48 bits***	***16 bits***	***46 to 1500 bytes***	***32 bites***
Preamble	*Destination Address*	*Source Address*	*Type/ Length*	*Data*	*Frame Check Sequence (CRC)*

Figura 2.3 - Divisão do frame Ethernet de acordo com o padrão IEEE 802.3. Fonte: SPURGEON (1).

O *frame* é dividido em seis campos distintos:

2.1.3.1 Preâmbulo

O *frame* começa com um preâmbulo de 64 bits, o qual permite a sincronização das interfaces Ethernet de 10Mbps, que estão se comunicando, antes que os dados importantes cheguem. Este campo existe para permitir que o começo do *frame* sofra perdas de alguns bits enquanto as interfaces estão em processo de sincronização, o que protege o resto do *frame* de perdas indesejadas de bits.

O preâmbulo foi mantido nos modos de operação *Fast* Ethernet e Gigabit Ethernet para fornecer compatibilidade com o *frame* Ethernet original. Entretanto, nesses dois casos existem outros mecanismos para a sincronização, não sendo necessário este processo.

Na especificação 802.3, este campo é formalmente dividido em duas partes, 7 bytes de preâmbulo, os quais tem o valor "01010101" e um byte que delimita o início do *frame* (SFD - *Start Frame Delimiter*), o qual tem o valor "11010101" (1).

2.1.3.2 Endereço do destino

Cada interface Ethernet tem um endereço único de 48 bits, chamado de endereço físico ou endereço de hardware da interface. Este campo contem 48 bits que correspondem ao endereço físico da interface de destino do *frame*. O campo pode ser preenchido também por um endereço *multicast*, o qual uma ou mais interfaces podem estar habilitadas para responder, ou o endereço broadcast[†] padrão.

Todas as interfaces Ethernet, conectadas a rede, leem pelo menos o campo de endereço de destino de todos os *frames* Ethernet transmitidos. Se o endereço de destino não for o mesmo da interface que o recebeu ou um dos endereços *multicast*, ou broadcast, que esta interface está programada para receber, então ela está livre para ignorar o resto do *frame* (1).

Na Ethernet, os 48 bits do endereço físico são escritos em 12 dígitos hexadecimais, os quais são separados em grupos de dois, representando a informação em grupos de 8 bits. A ordem de transmissão dessa informação é do grupo a esquerda para o grupo a direita, do mesmo modo como é escrito. Entretanto, a transmissão do octeto é iniciada no bit menos

[†] *O endereço broadcast é um endereço de 48 bits geral. Toda interface Ethernet que receber um frame com este endereço de destindo lerá o conteudo do pacote e o entregará ao software de rede que estiver rodando em seu computador.*

significativo e termina no bit mais significativo. Isso significa que o endereço Ethernet escrito na forma hexadecimal “F0-2E-15-6C-77-9B” é equivalente a sequência de bits abaixo (1), sendo enviadas da esquerda para a direita (1).

0000 1111 0111 0100 1010 1000 0011 0110 1110 1110 1101 1001 **(1)**

2.1.3.3 Endereço da fonte

Este campo é o endereço físico da interface que está enviando o *frame* e não é interpretado pelo protocolo MAC Ethernet. Tal campo é utilizado por protocolos de alto nível.

A estação Ethernet utiliza o seu próprio endereço físico como o endereço de fonte para todos os *frames* que ela transmite (1).

2.1.3.4 Campo de tipo ou de tamanho

Quando o padrão IEEE 802.3 foi primeiramente publicado no ano de 1985, o campo de tipo não foi incluído e em seu lugar as especificações IEEE colocaram o campo de tamanho, mas em 1997 o campo de tipo foi incorporado ao padrão. Desta forma, este campo pode conter tanto o tamanho do *frame* como o tipo de protocolo por ele carregado. Se o valor neste campo for numericamente igual ou menor que o tamanho máximo para um *frame*, ou seja, 1518_d, então ele é utilizado como um campo de tamanho. Se o tamanho dos dados for menor que o mínimo requerido para o campo, então serão adicionados automaticamente os octetos necessários para ajustar o tamanho quando este for enviado. Na recepção do *frame*, este campo é utilizado para determinar a quantidade de dados que são válidos e os octetos não válidos são descartados.

Se o valor deste campo for numericamente maior ou igual a 1536_d, então ele será usado como campo de tipo. Neste caso, o identificador hexadecimal neste campo, indicará o tipo de protocolo contido no campo de dados do *frame*. No caso do protocolo de resolução de

endereço (ARP – *Address Resolution Protocol*) o valor é 0806_h e no caso do protocolo Internet (IP – *Internet Protocol*) o valor é 0800_h (1).

2.1.3.5 Campo de dados

Este campo contém os dados transmitidos pelo pacote. Ele deve ter tamanho entre 46 e 1500 bytes. Se os dados transmitidos forem insuficientes para preencher o tamanho mínimo do campo, devem ser inseridos bytes de preenchimento para que este tamanho seja alcançado.

2.1.3.6 Campo de checagem da sequência do frame

Por último tem-se o campo de checagem de sequência do *frame* (FCS - *Frame Check Sequence)*. Este campo de 32 bits contém um valor que é usado para checar a integridade dos vários bits do *frame*, excluindo o campo do preâmbulo. O valor é computado utilizando a técnica conhecida como CRC, a qual é capaz de encontrar erros na transmissão, mas não pode realizar correções. O CRC é baseado em uma aritmética polinomial que consiste em tratar a mensagem como um grande número binário, e calcular o resto da divisão desta mensagem por um polinômio, previamente definido, chamado de polinômio gerador. Este polinômio é de uma única variável x e seus coeficientes são "0" ou "1". Adições e subtrações são feitas em módulo 2, ou seja, ambas são operações de "ou exclusivo". Por exemplo, tendo os polinômios (2 e 3) abaixo:

$$x^3 + x + 1 \quad \textbf{(2)}$$

$$x^4 + x^3 + x^2 + x \quad \textbf{(3)}$$

A soma ou a subtração destes polinômios é o polinômio (4):

$$x^4 + x^2 + 1 \quad \textbf{(4)}$$

O processo de verificação utilizado para transmissão em redes Ethernet é descrito da seguinte forma, sendo M a representação polinomial da mensagem a ser transmitida, a estação transmissora calcula o resto da divisão de M pelo polinômio gerador G, que no caso do CRC de 32 bits é definido como o indicado em (5).

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

Ou, em hexadecimal, como indicado em (6).

$$G = 04C11DB7h \quad \textbf{(6)}$$

Portanto, o resultado está indicado em (7).

$$R = (Mx^r)\ mod\ G \quad \textbf{(7)}$$

Na qual "r" é a ordem do polinômio onde, no caso, é 3.

A estação transmissora, então, acopla o valor do complemento de R ao final da mensagem. A estação receptora, por sua vez, deve calcular o CRC da mensagem juntamente com o complemento de R, e comparar esse valor com um valor constante como demonstrado a seguir:

Denotando o complemento do polinômio R como $\overline{R}$, têm-se o indicado (8).

$$\left(Mx^r + \overline{R}\right) mod\ G = \left(Mx^r + (x^{r-1} + x^{r-2} + \cdots + 1 - R)\right) mod\ G \quad \textbf{(8)}$$

Uma vez que a soma e a subtração em módulo 2 são operações idênticas fica-se com o observado (9).

$$\left(Mx^r + \overline{R}\right) mod\ G = ((Mx^r + R) + x^{r-1} + x^{r-2} + \cdots + 1)\ mod\ G. \quad \textbf{(9)}$$

Sabendo que $(Mx^r + R)\ mod\ G = 0$, encontra-se (10).

$$\left(Mx^r + \overline{R}\right) mod\ G = (x^{r-1} + x^{r-2} + \cdots + 1)\ mod\ G. \quad \textbf{(10)}$$

O que é um valor constante, chamado de resíduo e para o CRC de 32-bits com o polinômio gerador mencionado anteriormente se tem o indicado (11).

$$x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1 \quad \textbf{(11)}$$

Ou, em hexadecimal (12).

$$C704DD7Bh \quad \textbf{(12)}$$

Então, se o valor calculado pela estação receptor for idêntico ao valor apresentado em (12) a estação receptora assume, com uma alta probabilidade de acerto, que nenhum erro ocorreu durante a transmissão pelo canal Ethernet (1-3).

2.1.4 O protocolo de controle de acesso ao meio

Os padrões Ethernets originais descrevem um modo de operação no qual apenas uma estação pode enviar dados em um mesmo instante, esse modo é chamado de *half-duplex.* Ele utiliza um protocolo MAC que dispõe de um conjunto de regras usadas para arbitrar o acesso ao canal compartilhado entre o conjunto de estações. O funcionamento deste protocolo é bem simples, cada dispositivo equipado com uma conexão Ethernet opera independentemente de todos os outros, ou seja, não há um controle central.

A Ethernet utiliza um mecanismo de entrega broadcast, no qual cada *frame* transmitido é ouvido por todas as estações. Enquanto isto pode parecer ineficiente, a vantagem é manter o meio físico tão simples o quanto possível. Em uma Ethernet LAN, tudo que o sistema deve fazer é ver se os bits estão sendo transmitidos para todas as estações corretamente. A interface Ethernet instalada na estação realiza os demais trabalhos.

Os sinais Ethernet são transmitidos de uma interface e enviados pelo canal compartilhado para todas as estações conectadas àquela estação. Para realizar uma transmissão, a estação primeiramente escuta o canal e, se este estiver ocioso, ela transmite os dados na forma de um *frame* Ethernet.

Como todos os *frames* Ethernets são enviados pelo canal compartilhado, todas as interfaces Ethernet conectadas a esse canal devem receber o pacote e verificar o segundo campo do *frame*, o qual contém o endereço da estação de destino. A interface compara este endereço de 48 bits com o seu próprio e com o endereço broadcast padrão. Se o endereço contido no *frame* for um broadcast ou se for o mesmo da interface que o está analisando, o pacote é entregue ao software de rede que estiver rodando em no computador. Todas as outras interfaces irão interromper a leitura assim que descobrirem que o endereço de destino não confere.

Após a transmissão de cada *frame*, todas as estações na rede, com dados a enviar, precisam competir de um modo justo pelo canal, e isso é possível através do protocolo de múltiplo acesso de sentido de transporte com detecção de colisão (CSMA/CD - *Carrier Sense Multiple Access with Collision Detection*).

2.1.4.1 O protocolo CSMA/CD

O protocolo CSMA/CD funciona de forma semelhante a um jantar em uma sala escura, onde os participantes podem apenas ouvir uns aos outros. Todos devem ouvir um silêncio por um período de tempo antes de falar (CS - *Carrier Sense*). Uma vez que este espaço de tempo ocorre, todos têm iguais chances para dizer algo (MA - *Multiple Access*). Se duas pessoas começarem a falar no mesmo instante, elas perceberão este fato e pararão de falar (CD - *Collision Detection*).Traduzindo isto em termo da Ethernet, a porção CS do protocolo significa que antes de transmitir, cada interface deve esperar até não haver sinal no canal. Quando isso ocorre ela poderá começar a transmitir. Se outra interface está transmitindo, então haverá um sinal no canal cuja condição é chamada de "*Carrier*". Todas as outras interfaces devem esperar até que o *Carrier* cesse e o canal esteja ocioso antes de tentar transmitir. Tal processo é chamado *deferral*. Com o acesso múltiplo, todas as interfaces Ethernet tem a mesma prioridade quando começam a enviar dados na rede, e todas as interfaces podem tentar acessar o canal a qualquer momento.

A próxima parte do protocolo de acesso é chamada de detecção de colisão. Dado que todas as interfaces Ethernet têm oportunidades iguais para acessar a Ethernet, é possível que múltiplas interfaces percebam que o canal está ocioso e comecem a transmitir seus *frames* simultaneamente. Quando isso ocorre, o dispositivo de sinalização Ethernet, conectado ao canal compartilhado, percebe a colisão de sinais e informa para as interfaces Ethernet para que estas parem de transmitir. Cada interface escolherá um tempo randômico antes de retransmitir àquele pacote, este processo é chamado de *backoff*. Escolher tempos randômicos para retransmitir ajuda a evitar que as estações colidam na retransmissão (1).

2.1.4.2 Regras para o controle de acesso ao meio

Quando está transmitindo um *frame*, a estação passa pelos seguintes estados:

i. Quando um sinal está sendo transmitido pelo canal, a condição é chamada de *Carrier.*

ii. Quando uma estação, que está conectada a Ethernet, quer transmitir um *frame*, ela espera até que o canal esteja ocioso, indicado como "ausência de *Carrier*".

iii. Quando o canal se torna ocioso, a estação espera por um breve período de tempo chamado de "espaço entre *frames*" (IFG – *Interframe gap*), e depois transmite o *frame*.

iv. Se acontecer de duas estações transmitirem o *frame* simultaneamente, elas detectaram a colisão de sinais e reagendam a transmissão. Esta ocorrência é chamada de detecção de colisão.

As regras definidas quando uma interface pode transmitir um *frame* são simples:

1. Se o canal estiver ocioso e o período sem *Carrier* continuar por certo tempo, que é igual ou maior que IFG, então o *frame* é transmitido imediatamente. Se a estação deseja transmitir múltiplos *frames*, ela deve esperar por um período de tempo igual à IFG entre as transmissões.

2. Se houver *Carrier*, então a estação continua escutando o canal até que o *Carrier* cesse. Tão cedo quanto o canal se torne ocioso, a estação pode começar o processo de transmissão do *frame*, o qual inclui a espera pelo intervalo de espaço entre *frames*.

3. Se uma colisão for detectada durante a transmissão, a estação irá prosseguir com o envio de 32 bits de dados para garantir que todas as interfaces percebam que houve uma colisão. Se a colisão é detectada muito cedo na transmissão do *frame*, então a estação irá continuar o envio até que o preâmbulo seja transmitido, e depois disso ela enviará os 32 bits de garantia.

Completar o envio do preâmbulo e transmitir os 32 bits garante que o sinal fique no sistema tempo o bastante para que todas as estações envolvidas na colisão reconheçam a colisão e reajam corretamente. Neste caso, duas considerações devem ser feitas:

a. Depois de enviar a sequência de garantia, a estação espera por um período de tempo escolhido com a ajuda de um gerador de números randômicos e, então recomeça a transmissão (*backoff*). A escolha randômica de tempo faz com que as estações esperem por diferentes intervalos de tempo, assim elas não estarão propensas a colidirem novamente.

b. Se a próxima tentativa de transmitir o *frame* resultar em outra colisão, então a estação executará o procedimento de *backoff* novamente, mas desta vez os intervalos de tempo utilizados pela escolha randômica serão incrementados. Isso reduz a probabilidade de ocorrer outra colisão e provê um mecanismo de ajuste automático para o trafego intenso.

4. Uma vez que, na velocidade de 10Mbps ou 100Mbps, uma estação tenha transmitido 512 bits de um *frame* excluindo o preambulo sem detectar uma colisão, então é dito que a estação adquiriu o canal. O tempo de 512 bits é conhecido como o tempo de abertura (*slot time*) de um canal Ethernet.

   Uma vez que a estação adquira o canal e transmita o seu *frame*, isso zera o contador de colisões, o qual é utilizado para gerar o tempo de *backoff*. Se for detectada uma colisão na transmissão do próximo *frame*, ele começará o calculo de *backoff* novamente (1).

Configurando o tamanho mínimo do *frame* para 512 bits, excluindo o preâmbulo, significa que o campo de dados terá pelo menos 46 bytes. Um *frame* com esse tamanho não será considerado um fragmento de colisão (1).

2.1.5 Full-Duplex Ethernet

Com o modo *full-duplex* habilitado, as duas estações podem transmitir e receber simultaneamente, isso dobra a capacidade agregada ao link. Por exemplo, um segmento *half-duplex* de *Fast* Ethernet provê um máximo de 100Mbps de largura de banda. Quando operando em *full-duplex*, o mesmo seguimento 100BASE-TX pode prover um link com largura de banda total igual a 200Mbps (1).

Quando mandando um *frame* no modo *full-duplex*, a estação ignora o CS e não atrasa o tráfego sendo recebido no canal. Entretanto, a estação ainda espera pelo período de IFG entre a transmissão de *frames*, assim como as interfaces Ethernet foram projetadas para esperar um intervalo entre cada *frame*. Prover este tempo assegura que as interfaces em cada ponta do link podem continuar com a taxa máxima de *frames* do link. No modo *full-duplex*, as estações das duas pontas ignoram qualquer sinal de colisão detectado pelo *transceiver*.

O algoritmo CSMA/CD utilizado em canais compartilhados no modo *half-duplex* não são mais usados em um link que está no modo *full-duplex.* Uma estação em um link *full-duplex* pode transmitir quando ela quiser, ignorando o CS. Não há múltiplo acesso, uma vez que cada estação tem um canal próprio de transmissão. Como não há disputa por acesso, não há colisões também, assim as estações em cada ponta do link podem ignorar a detecção de colisão (1).

2.2 Protocolos de rede

A Figura 2.4

Figura 2.4 exemplifica a diferença entre protocolos de rede e a Ethernet. Esta mostra um dado de determinado protocolo de rede viajando de uma estação “A” para uma estação “B”. O dado em “A” é representado pela carta que é colocada dentro de uma carta, um pacote de protocolo de alto nível que tem um endereço do protocolo de rede. Esta carta é colocada em um *frame* Ethernet, representado pelo saco de cartas. A analogia não é exata porque na verdade cada *frame* Ethernet transporta apenas uma “carta” por vez e não varias como mostrado na analogia. Então o *frame* Ethernet é colocado no meio físico, representado pelo caminhão, para que seja levado até a estação “B” (1).

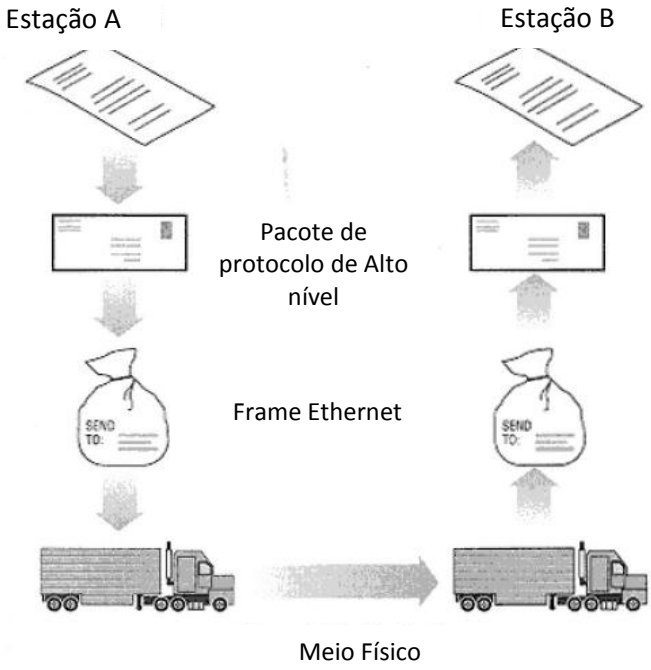


Figura 2.4 - Analogia entre a transferência de dados através de uma carta e através dos protocolos de rede Ethernet. Fonte: SPURGEON (1).

Dados enviados de um computador a outro são transportados no campo de dados do *frame* (quadro) Ethernet e estruturados como protocolos de rede de alto nível. Essa informação é o que realmente estabelece uma comunicação entre aplicações rodando em computadores conectados a rede (1).

2.2.1 O protocolo ARP

Protocolos de rede de alto nível têm seus próprios endereços de sistema, como o endereço IP de 32 bits usado na internet. O software que realiza comunicação IP em um determinado computador tem conhecimento do endereço IP de 32 bits associado àquele computador e pode, também, ler o endereço Ethernet de sua interface de rede. Entretanto, ao mandar o primeiro pacote pela rede, este não sabe o endereço Ethernet dos outros sistemas conectados. Então, para descobrir este endereço, é utilizado o protocolo ARP (1).

O funcionamento do protocolo ARP é mostrado na Figura 2.5. A estação A tem o IP “192.0.2.1”, e deseja mandar dados pelo canal Ethernet para uma estação B, a qual tem o IP “192.0.2.2”. A primeira envia um pacote broadcast contendo um pedido ARP. Nesta

mensagem está escrito basicamente "*A interface que tem o IP 192.0.2.2, por favor, me diga qual é o endereço de hardware da sua interface Ethernet*".

Desde que o pedido de ARP é enviado em um *frame* broadcast, todas as interfaces Ethernet conectadas àquela rede receberão a mensagem, mas somente a estação "B", que tem o endereço requisitado, responderá. Ela enviará um pacote de resposta contendo seu endereço Ethernet para a estação requisitante.

Portanto o protocolo ARP fornece um link entre o endereço IP de 32 bits e o endereço Ethernet de 48 bits. Os computadores envolvidos nas comunicações constroem uma tabela na memória chamada de cache ARP, na qual ficam salvos os endereços das interfaces com as quais aquele computador se comunicou recentemente, a fim de utilizar o endereço obtido para comunicações futuras (1).

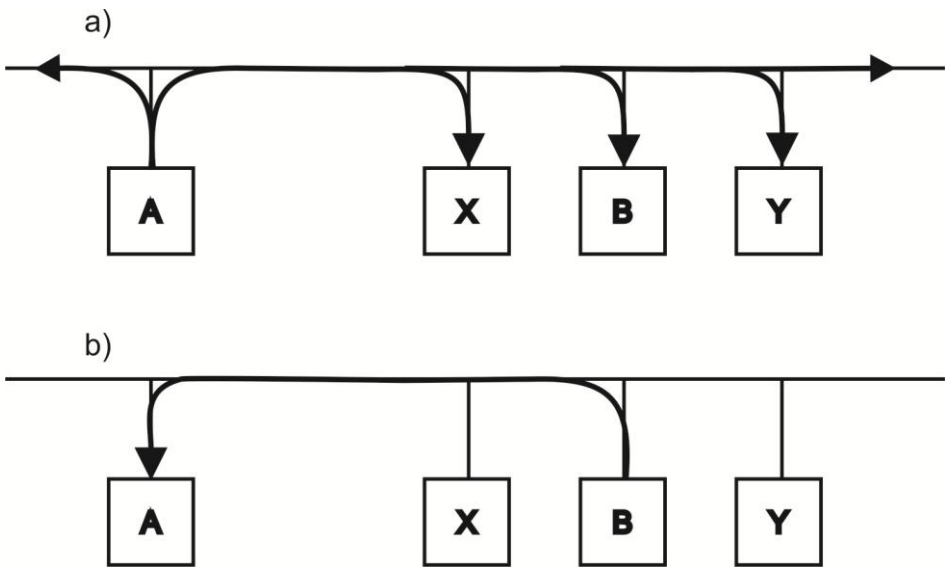


Figura 2.5 - Exemplo do funcionamento do protocolo ARP. (a) A envia um broadcast para descobrir o endereço físico de "B". (b) "B" responde ao pedido de ARP de "A". Fonte: SPURGEON (1).

Assim como indicado na Figura 2.5, a ideia por trás da resolução dinâmica utilizando ARP é simples: Quando "A" deseja encontrar o endereço físico do portador do IP I_B, ela envia um pacote broadcast, o qual diz para o dono do IP I_B enviar um pacote de resposta no qual esteja inserido o seu endereço físico. Todos os dispositivos conectados a rede, incluindo "B", iram receber a mensagem, mas apenas "B" reconhecerá seu endereço de IP e enviará uma resposta contendo seu endereço físico. Quando "A" receber a resposta, ele usará este endereço para se comunicar diretamente com "B" (4).

O ARP, portanto, é um protocolo de baixo-nível que esconde o endereço físico de rede, permitindo atribuir um endereço IP arbitrário para todas as máquinas. Nós pensamos no ARP como parte do sistema físico de rede, e não como parte do protocolo internet.

2.2.1.1 Encapsulação e identificação ARP

Quando as mensagens ARP viajam de uma máquina para outra, elas precisão ser transportadas em *frames* físicos e para identificar esses *frames* como transportadores de mensagens ARP, o transmissor atribui um valor especial para o campo de tipo no cabeçalho do *frame*, e coloca a mensagem ARP no campo de data do *frame*. Quando o *frame* chega a um computador, o software de rede usa o campo de tipo para determinar o seu conteúdo. Na rede Ethernet, *frames* carregando mensagens do tipo ARP tem o campo de tipo preenchido com o valor 0806_h (4).

2.2.1.2 Formato do protocolo ARP

Como a maioria dos protocolos, o dado em um pacote ARP não tem um formato fixo de cabeçalho. De fato, o formato da mensagem ARP é geral o bastante para permitir que ela seja usada com um endereço físico arbitrário e um endereço de protocolo arbitrário. O exemplo indicado na

Figura 2.6 mostra os 28 bytes de uma mensagem ARP, utilizada no hardware Ethernet, quando estão resolvendo endereços do protocolo IP.

O campo de tipo de hardware especifica o tipo da interface de hardware para a qual o transmissor procura uma resposta, ele possui o valor 0001_h para a Ethernet. Similarmente, o campo de tipo do protocolo especifica o tipo de endereço do protocolo de alto-nível que o

transmissor forneceu, o qual é 0800_h para o endereço IP. O campo operação especifica se a mensagem é um pedido ARP, uma reposta ARP, um pedido RARP, ou uma resposta RARP, para os quais os valores são 0001_h, 0002_h, 0003_h e 0004_h respectivamente.

0	8	16	24	31
HARDWARE TYPE		PROTOCOL TYPE		
HLEN	PLEN	OPERATION		
SENDER HÁ (octets 0-3)				
SENDER HÁ (octets 4-5)		SENDER IP (octets 0-1)		
SENDER IP (octets 2-3)		TARGET HÁ (octets 0-1)		
TARGET HÁ (octets 2-5)				
TARGET IP (octets 0-3)				

Figura 2.6 - Divisões utilizadas para transferência de dados através do protocolo ARP. Fonte: COMER (4).

Os campos HLEN e PLEN permitem que o protocolo ARP seja usado em uma rede arbitrária, porque eles especificam o tamanho do endereço de hardware e do endereço de IP, usados nos campos SENDER HA e SENDER IP.

Quando está fazendo um pedido, o transmissor também fornece o endereço de hardware, no caso da mensagem RARP, ou o endereço IP, no caso da mensagem ARP, do alvo, utilizando os campos TARGET HA ou TARGET IP. Ao responder uma mensagem, a máquina preenche o campo de endereço que está faltando, troca os campos de transmissor e alvo, e altera a operação para o modo resposta. Assim, a resposta carrega o endereço IP e de hardware do transmissor original, como os endereços para a qual o pedido de ARP havia sido enviado (4).

2.2.2 Mensagem ICMP

O protocolo ICMP (*Internet Control Message Protocol*) é utilizado para comunicação das informações da camada de rede, como por exemplo, a comunicação de erros. Mensagens deste tipo são encapsuladas no protocolo IP, portanto elas contêm o cabeçalho e os primeiros 8 bytes do data grama IP (5), como mostrado na Figura 2.7. No cabeçalho, os primeiros quatro bits representam a versão do protocolo IP utilizada, os próximos quatro bits fornecem o tamanho do cabeçalho do datagrama. Já o

campo *Service Type* especifica como o datagrama deve ser utilizado, sendo utilizado por serviços diferenciados. O campo seguinte fornece o tamanho total do datagrama em octetos.

Os bytes seguintes representam uma identificação única utilizada na montagem de datagramas fragmentados. O campo de *flags* indica algumas características do datagrama, como a possibilidade de fragmentação ou não, enquanto que o campo seguinte especifica um valor para cada fragmento no processo de desfragmentação. O *time to live* determina o tempo de vida do pacote na rede. O campo *protocol* diz qual protocolo está sendo carregado pelo pacote como, por exemplo, o UDP (valor 17) e o ICMP (valor 01). O *hearder checksum* é um campo de controle de erros no cabeçalho e os dois campos seguintes representam o endereço IP da fonte e do destino, respectivamente.

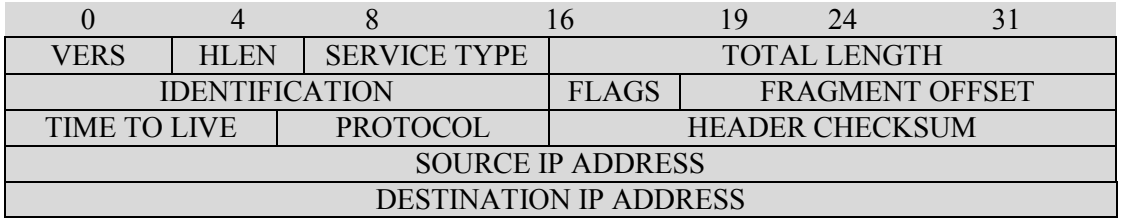


Figura 2.7 - Divisões contidas no cabeçalho e nos primeiros 8 bytes de um datagrama Internet. Fonte: COMER (4).

Para completar, uma mensagem ICMP recebe o cabeçalho ICMP e um campo de dados opcional, indicados na

Figura 2.8. O primeiro byte deste cabeçalho indica o tipo de mensagem ICMP, sendo que para o caso de um pedido Ping o valor é 08 e em uma resposta Ping o valor é 00. O campo *code* leva um código que descreve o problema encontrado. Na sequência aparece, novamente, um campo para checagem de erro dentro da mensagem ICMP. Já os campos *identifier* e *sequence number* são utilizados para associar os pedidos de eco às suas respectivas respostas, eles ajudam a identificar os pacotes quando diversos pedidos são enviados ao mesmo destino.

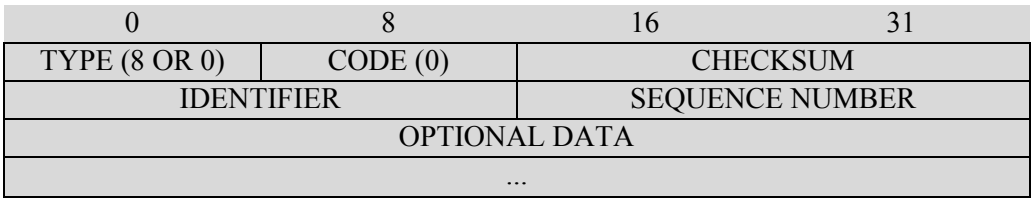


Figura 2.8 - Divisões de um frame que representa um pedido ou resposta de eco do protocolo ICMP. Fonte: COMER (4).

A

Figura 2.9 mostra a encapsulação de uma mensagem ICMP.

Um dos programas que faz uso de mensagens ICMP é o Ping. Ele envia uma mensagem do tipo oito com código zero para uma determinada estação, a qual, ao ver o pedido Ping, responde com uma mensagem ICMP do tipo zero e código zero sendo, portanto, uma resposta Ping.

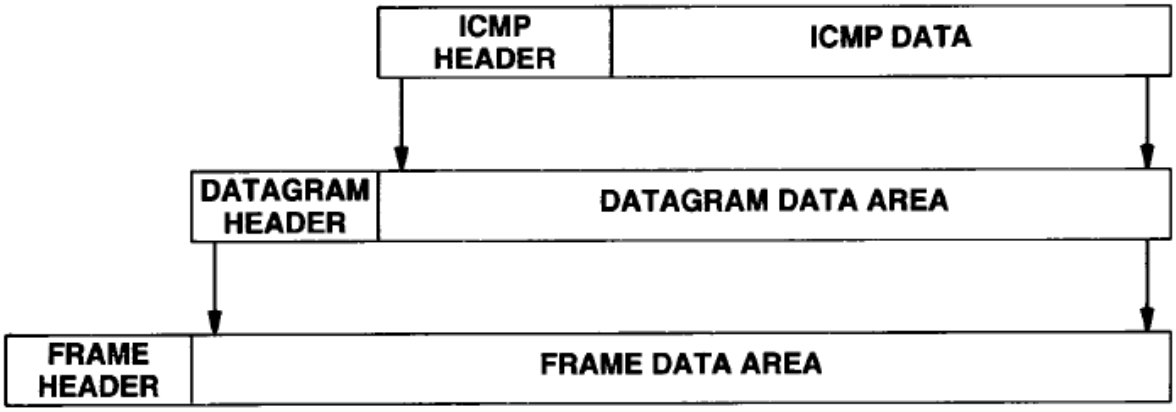


Figura 2.9 - Dois níveis de encapsulação do protocolo ICMP. Fonte: COMER (4).

2.3 Computação reconfigurável

É possível identificar duas características principais dos processadores, a flexibilidade e a desempenho. Os computadores baseados na arquitetura de Von Neumann (VN), por exemplo, são flexíveis e podem executar qualquer tipo de tarefa, por isso recebem o nome de processadores para uso geral (GPP - *General Purpose Processor*). Eles não fornecem muito desempenho por não executarem cálculos em paralelo, mas são muito flexíveis, exigindo que a aplicação seja adaptada para o hardware.

Um exemplo oposto aos GPPs são os *application-specific instruction-set processor* (ASIPs), os quais fornecem alta performance pois são otimizados para uma determinada aplicação e o seu conjunto de instruções pode ser construído em um chip. Neste caso, o hardware é sempre adaptado para a aplicação.

É possível ver na Figura 2.10 um gráfico entre a performance e a flexibilidade de diferentes processadores.

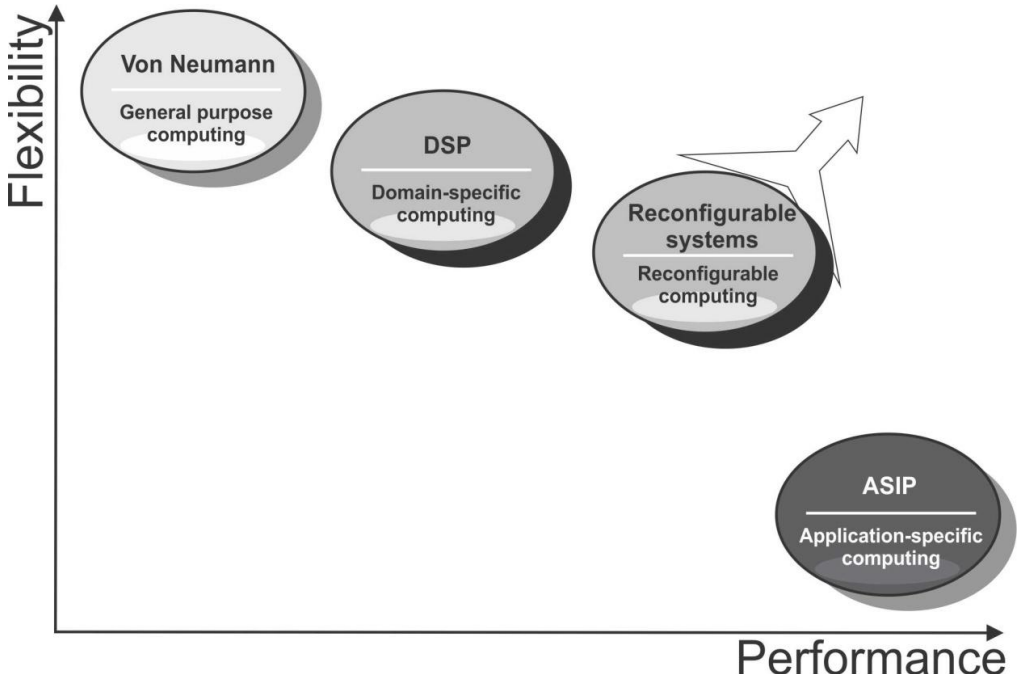


Figura 2.10 - Gráfico de Flexibilidade x Performance das classes de processadores. Fonte: BOBDA (6).

Como é mostrado na Figura 2.10, existe entre os GPPs e os ASIPs, um grande número de processadores que, dependendo das variáveis, podem ser colocados perto ou longe dos GPPs nas duas escalas. O ideal seria ter a flexibilidade de um GPP e contar com a performance de um ASIP em um mesmo dispositivo. A tecnologia que mais se aproxima deste panorama é o chamado hardware reconfigurável, no qual para uma data aplicação em um determinado tempo, a estrutura espacial do dispositivo pode ser modificada, tornando possível a utilização da aproximação computacional com melhor desempenho para realizar a aplicação. Ao contrário dos computadores de VN, os quais são programados por um conjunto de instruções que serão executadas sequencialmente, a estrutura de dispositivos reconfiguráveis é alterada modificando todo o hardware ou parte dele durante a compilação ou a execução (6).

A classe de hardware configurável que será utilizada neste projeto é o FPGA.

2.3.1 Field-programmable gate array (Fundamentals of Digital Logic)

Introduzido em 1985 pela companhia Xilinx, um FPGA é um dispositivo lógico programável que suporta a implementação de circuitos lógicos relativamente grandes, com mais de um milhão de portas lógicas. A estrutura geral de um FPGA é ilustrada na Figura 2.11 e mostra três recursos principais: blocos lógicos, blocos de entrada e saída (*Input/Output* – I/O) e fios de interconexões com chaves (*switches*). Os blocos lógicos são dispostos em um vetor bidimensional e os fios de interconexão são organizados como canais de roteamento horizontais e verticais entre as linhas e colunas de blocos lógicos. Estes canais contêm fios e chaves programáveis que permitem interconectar os blocos lógicos de milhares de maneiras diferentes. É mostrado na Figura 2.11 duas localizações para as chaves programáveis, onde a primeira são os blocos adjacentes aos blocos lógicos, os quais possuem chaves que conectam os terminais de entrada e saída dos blocos lógicos com os fios de interconexões e a segunda são os blocos que estão na diagonal, entre os blocos lógicos, os quais conectam um fio de interconexão ao outro (assim como os fios verticais aos fios horizontais). Conexões programáveis também existem entre os blocos de I/O, utilizados para realizar a conexão com os pinos da FPGA, e os fios de interconexão. O número total de chaves programáveis e fios em uma FPGA varia de acordo com os chips disponíveis no mercado. Cada um dos três componentes básicos de um FPGA pode ser programado pelo usuário (6-7).

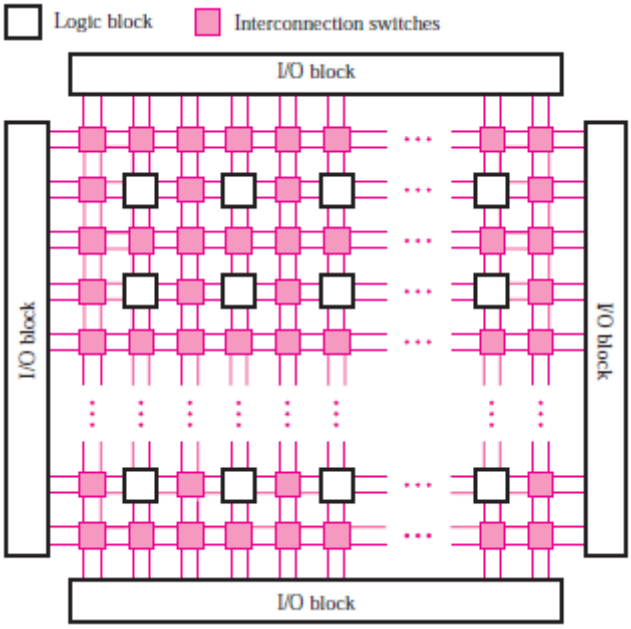


Figura 2.11 - Estrutura geral de uma FPGA. Fonte: BROWN; VRANESIC (7)

O bloco lógico mais utilizado em FPGAs é a *lookup table* (LUT), o qual contêm células de armazenamento que são usadas para implementar pequenas funções lógicas capazes de armazenar um único valor lógico, zero ou um, o qual é produzido como saída da célula. LUTs de vários tamanhos podem ser criadas, onde o tamanho é definido pelo número de entradas. É mostrado na Figura 2.12 a estrutura de uma pequena LUT com duas entradas, x_1 e x_2, e uma saída, f, na qual é possível implementar qualquer função lógica de duas variáveis.

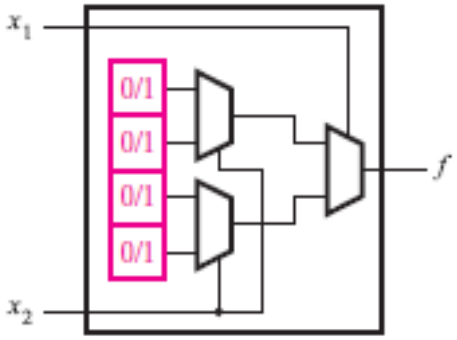


Figura 2.12 - Circuito de uma LUT de duas entradas. Fonte: BROWN; VRANESIC (7)

Está LUT tem quatro células de armazenamentos, o mesmo número de linhas de uma tabela verdade de duas variáveis, de modo que o valor em cada célula corresponde a um valor de saída de uma das linhas da tabela verdade daquela função. As variáveis de entrada, x_1 e x_2, são utilizadas como as entradas de três multiplexadores, os quais, dependendo do valor das

variáveis, podem selecionar o conteúdo de uma das quatro células de armazenamento como saída da LUT.

Para entender melhor o funcionamento de uma função lógica implementada na LUT, considere a tabela verdade da

Tabela 2.1. A função f_1 desta tabela pode ser armazenada em uma LUT como ilustrado na Figura 2.13, na qual a organização dos seus multiplexadores produz corretamente o resultado. Por exemplo, quando $x_1 = x_2 = 0$, a saída da LUT é a primeira célula de cima para baixo com valor lógico igual a um. Similarmente, para todos os valores de x_1 e x_2 há um valor lógico correspondente que será utilizado como saída da LUT.

Tabela 2.1- Tabela verdade para uma LUT de duas entradas representando uma função XNOR.

X_1	X_2	f_1
0	*0*	*1*
0	*1*	*0*
1	*0*	*0*
1	1	1

Fonte: BROWN;VRANESIC (7).

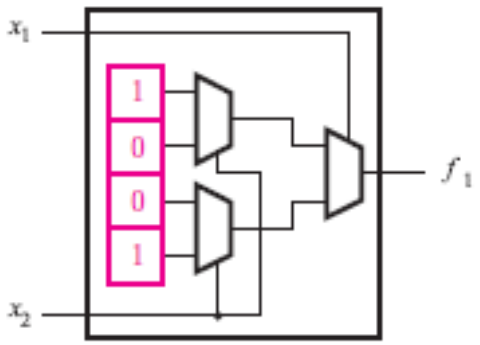


Figura 2.13 - Conteúdo da célula de armazenamento da LUT exemplificada na

Tabela 2.1.Fonte: BROWN; VRANESIC (7).

Uma LUT de *n* entradas pode ser usada para implementar até $2^{2^{\wedge}n}$ funções diferentes e cada uma delas pode ter 2^n saídas diferentes. Portanto, uma LUT de *n* entradas deve prover 2^n células de armazenamento.

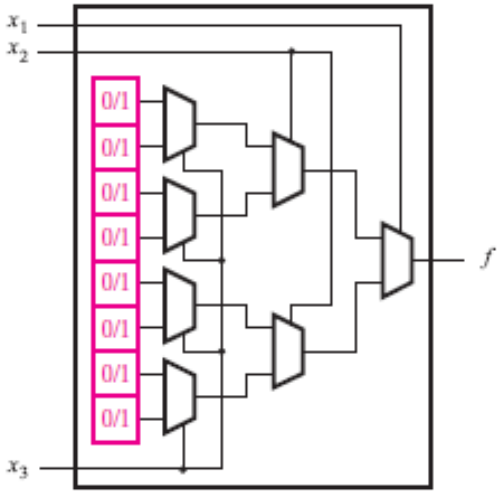


Figura 2.14 - Circuito de uma LUT de três entradas. Fonte: BROWN; VRANESIC (7).

Então, ao considerar uma LUT de três entradas, serão necessárias oito células de armazenamento correspondendo às oito linhas de uma tabela verdade de três variáveis, como ilustra a Figura 2.14. Nos FPGAs comerciais, as LUTs têm normalmente quatro ou cinco entradas, o que requer dezesseis ou trinta e duas células de armazenamento, respectivamente. Além disso, também é comum que um FPGA tenha circuitos extras em um mesmo bloco lógico, como é o caso da Figura 2.15, a qual ilustra um bloco lógico contendo um *Flip-Flop*, circuito eletrônico controlado pelo sinal de relógio que é utilizado para armazenar o valor de entrada *D*. Outra característica de FPGAs comerciais é o agrupamento de várias LUTs em um único módulo em conjunto com outros elementos funcionais, como *Flip-flops* e multiplexadores. A vantagem dessa abordagem é que a conexão entre as LUTs dentro do módulo é mais rápida do que as conexões feitas pelo roteamento da rede uma vez que são utilizados fios dedicados.

Ao implementar um circuito lógico dado pelo usuário, as ferramentas CAD automaticamente o convertem em pequenas funções capazes de se encaixar nas LUTs do FPGA. Além disso, os canais de roteamento são programados para realizarem as

interconexões necessárias entre os blocos lógicos, de modo a interligar todas essas pequenas funções e formar o circuito dado pelo usuário.

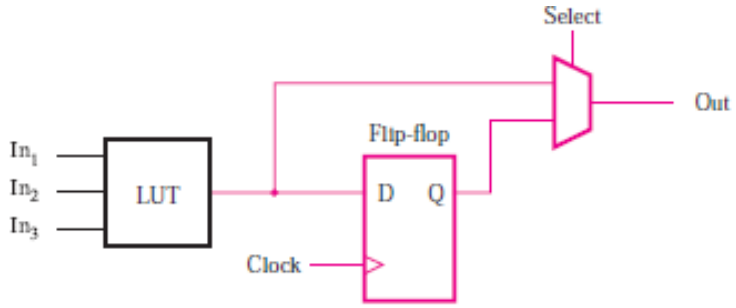


Figura 2.15 - Circuito de um bloco lógico com um Flip-Flop. Fonte: BROWN; VRANESIC (7).

Como exemplo, considere um circuito implementado em um pequeno FPGA como ilustrado na Figura 2.16, a qual mostra uma seção programada do FPGA onde os blocos lógicos estão representados por quadrados, e as chaves por *X*. Este FPGA tem LUTs de duas entradas e quatro fios em cada canal de roteamento. Cada chave mostrada na cor rosa está ativa e realiza uma conexão entre os fios horizontais e verticais, enquanto que as chaves em preto estão desativadas. As tabelas verdade programadas nos blocos lógicos na linha de cima da FPGA correspondem as funções $f_1 = x_1.x_2$ *e* $f_2 = not(x_2.x_3)$. Desta forma, o bloco lógico a direita da linha inferior está programado para implementar a função $f = f_1$ *or* f_2.

As células de armazenamento em um FPGA são voláteis, ou seja, elas perdem o seu conteúdo toda vez que o fornecimento de energia é cortado. Por essa razão, cada uma delas deve ser programada toda vez que o fornecimento de energia é ativado. Frequentemente, um chip de memória, chamado de memória programável de leitura (PROM - *Programmable Read-Only Memory*), que guarda a programação permanentemente, é incluído no circuito da placa que aloja o FPGA. Assim, os valores das células de armazenamento são carregados automaticamente da PROM no momento em que o chip é ligado (6-7).

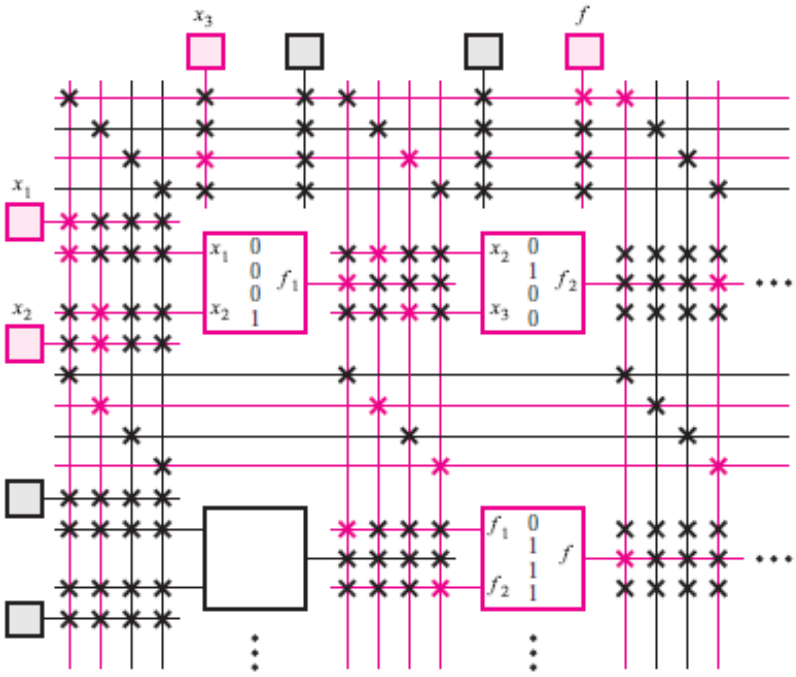


Figura 2.16 - Exemplo de uma seção de FPGA programada. Fonte: BROWN; VRANESIC (7).

2.3.2 Stratix III

O FGPA utilizado neste trabalho é da família Stratix III da Altera, a qual oferece dispositivos de 47.500 a 338.00 elementos lógicos. O seu elemento lógico é mais complexo do que o de outros FGPAs e é chamado de módulo lógico adaptativo (ALM - *Adaptive Logic Module*). Cada ALM é composto por uma lógica combinacional e dois *Flip-flops* programáveis. Com até oito entradas para as duas LUTs adaptativas combinacionais, um ALM pode implementar funções lógicas de até 7 variáveis, ou até uma combinação de duas funções. Na Figura 2.17 é mostrado algumas combinações possíveis de um ALM, como duas LUTs de quatro entradas, uma LUT de três entradas e outra de cinco entradas, e assim por diante. Outra possibilidade permitida por esse sistema é a sua configuração para que opere como um pequeno bloco de memória (7-8).

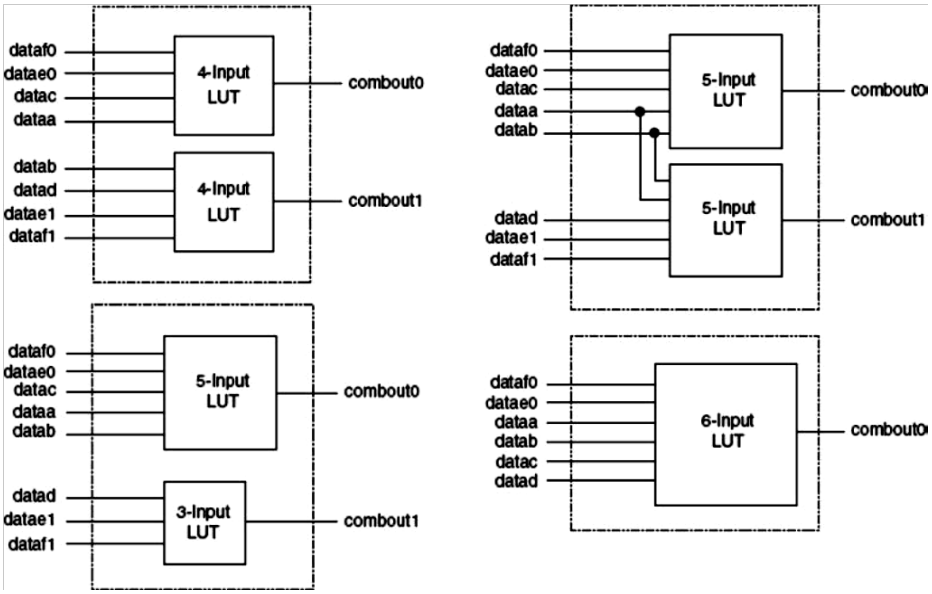


Figura 2.17 - Algumas configurações possíveis de um ALM. Fonte: ALTERA CORPORATION (9).

2.3.3 Desenvolvimento do hardware

As linguagens de descrição de hardware (HDL - *Hardware Description Language*) se tornaram importantes com o aumento no nível de abstração no desenvolvimento de hardware. Elas foram desenvolvidas originalmente para construir simulações eficientes de sistemas digitais, ou seja, descrever execuções paralelas dos componentes de hardware e, por isso, são diferentes daquelas consideradas sequenciais. Existem algumas diferenças entre linguagem de programação sequencial e de linguagem de simulação, uma delas é que as declarações não são executadas de forma sequencial durante a simulação, ou seja, o código que utiliza uma linguagem de simulação pode descrever uma série de portas lógicas, as quais podem ter suas saídas alteradas simultaneamente (10).

Neste trabalho utilizaremos o VHDL que, assim como o Verilog, foi desenvolvida inicialmente como linguagem de simulação. Entretanto, ambas passaram a ser muito utilizadas como linguagens de síntese. Um modelo em VHDL ou Verilog pode ser utilizado para definir a funcionalidade de componentes para sistemas lógicos e antes de ser sintetizado, ele pode ser simulado a fim de checar o seu funcionamento.

Verilog é uma linguagem que difere do VHDL em muitos aspectos. Sua sintaxe é mais simples, pois foi desenvolvido para uma simulação eficiente, porém, para projetos de alto

nível, o VHDL se destaca uma vez que é uma linguagem fortemente tipificada e bem estruturada. Estas características do VHDL tornam a documentação do projeto mais organizada (10-11).

Para o desenvolvimento dos módulos de hardware, foi utilizada a ferramenta Quartus II que provê uma interface gráfica para o usuário acessar ferramentas e editar arquivos (11). Tal desenvolvimento envolve, normalmente, questões de propriedade intelectual, por isso, tais módulos são conhecidos como núcleos IPC (*intellectual property cores*). Um núcleo IPC pode ser desenvolvido em código HDL de alto nível ou em um *layout* no nível de transistor (11).

Para facilitar o projeto do sistema embarcado, a Altera disponibiliza um conjunto de *softcore*, incluindo o processador Nios II e uma coleção de periféricos comumente usados. Para definir o sistema e gerar os códigos HDLs, a Altera também disponibiliza um aplicativo conhecido como *SOPC Builder*, através do qual é possível instanciar os componentes. A aplicação salva a configuração em um arquivo de extensão .sopc, além de exportar as informações para um arquivo de extensão .sopcinfo. O primeiro é utilizado para programar o FPGA e o segundo fornece informações sobre a configuração do sistema desenvolvido. Ambos são utilizados no desenvolvimento do software (11).

2.3.4 Visão global do processador Nios II

O processador *softcore* Nios II utilizado neste trabalho para processar dados é de propriedade da Altera e de implementação específica em dispositivos FPGAs do fabricante. Ao contrário de um processador pré-fabricado, um processador *softcore* é descrito em código HDL e então mapeado em elementos lógicos genéricos de uma FPGA. Está abordagem permite maior flexibilidade, uma vez que o processador pode ser configurado e ajustado, sendo possível adicionar ou remover características para se alcançar os objetivos de desempenho ou custo.

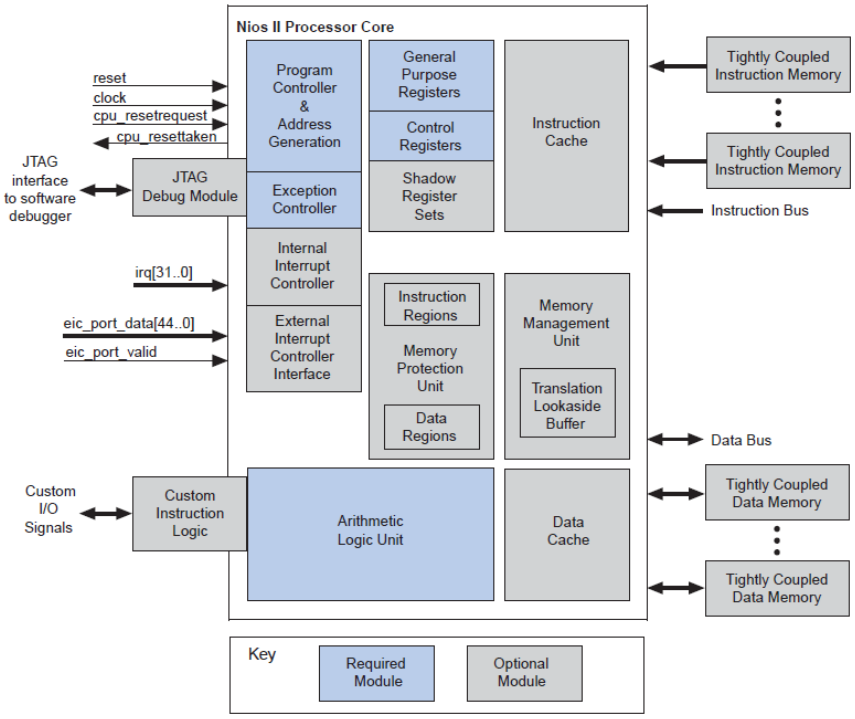


Figura 2.18 - Diagrama de blocos de um processador Nios II. Fonte: ALTERA CORPORATION (9).

.

O Nios II segue o principio de projeto básico de uma arquitetura RISC (*Reduced Instruction Set Computer*) e utiliza um conjunto de instruções pequeno e otimizado. As suas características principais são:

- Arquitetura *Load-Store.*
- Formato de instrução fixo de 32 bits.
- Caminho de dados interno de 32 bits.

- Espaço de endereço de 32 bits.
- Espaço de I/O de memória mapeada.
- 32 níveis de pedido de interrupção.
- 32 registradores de propósito geral.

O diagrama de blocos conceitual de um processador Nios II é mostrado na Figura 2.18 e os blocos principais são:

- Arquivo de registradores (registradores de propósito geral) e controle de registradores.
- *Arithmetic and logic unit* (ALU).

  Uma operação ALU é realizada em um ou dois registradores de entrada e armazena o resultado em outro registrador. As operações podem ser aritméticas, lógicas ou de deslocamento, lembrando que a implementação da multiplicação e/ou divisão é mais complexa e requer mais recursos de hardware, então esta pode ser incluída ou excluída do projeto de acordo com suas necessidades.

- Controlador de exceção e interrupção.
- Opcional - Cache de instrução e cache de dados.

  A memória cache é uma memória pequena e rápida entre o processador e a memória principal, como mostrado na Figura 2.18. No processador Nios II, a *cache* é implementada em um módulo de memória interna da FPGA. Se a maioria dos acessos a memória for a dados localizados na cache, o tempo médio de acesso será mais próxima da latência do *cache* do que da latência da memória principal. Desta forma, sabendo que a latência da cache é menor, é possível diminuir o tempo médio de acesso aos dados.

- Opcional - M*emory Management Unit* (MMU).

  MMU é um mecanismo opcional no processador Nios II/f que permite a utilização de memória virtual para aumentar a capacidade de memória. Na verdade, este mecanismo traduz um endereço virtual para um endereço físico, tornando possível armazenar dados em um disco externo como se fosse parte da memória do computador.

- Opcional – *memory protection unit* (MPU).
- Opcional – Módulo para eliminação de erros (*debug*) JTAG.

O módulo se conecta aos sinais dentro do processador e pode assumir o controle deste. Desta forma, um computador pode utilizar a porta JTAG do FPGA para se

comunicar com o módulo de debug e realizar atividades de verificação, como enviar programas para a memória, adicionar pontos de quebra (*breakpoints*), examinar registradores, memória e coletar dados da execução. O módulo pode ser incluído no projeto durante a fase de desenvolvimento e removido quando o produto estiver pronto.

Existem três versões básicas do Nios II:

- Nios II/f: O núcleo rápido é projetado para alto desempenho. Ele tem um *pipeline* de seis estágios, um *cache* de instruções, um *cache* de dados e uma previsão de desvios dinâmica.
- Nios II/s: O núcleo padrão é projetado para ter um tamanho pequeno, mas mantendo um bom desempenho. Ele tem um *pipeline* de cinco estágios, um *cache* de instruções e uma previsão de desvios estática.
- Nios II/e: O núcleo econômico é projeto para ocupar pouco espaço. Ele não tem *pipeline* nem *cache.*

As características principais fornecidas pela Altera estão listadas na Tabela 2.2.

Tabela 2.2 - Comparação entre as versões do Nios II.

Características	**Núcleo**		
	Nios II/e	Nios II/s	Nios II/f
Frequência de relógio máxima	200 MHz	165 MHz	185 MHz
Desempenho em DMIPS	31	127	218
Área do FPGA ocupada	< 700 LEs < 350 ALMs	< 1400 LEs < 700 ALMs	Sem MPU ou MMU: < 1800 LEs < 900 ALMs Com MMU: < 3000 LEs < 1500 ALMs Com MPU: < 2400 LEs < 1200 ALMs
Estágios do pipeline	1	5	6
Espaço para endereço externo	2 GB	2 GB	2 GB sem MMU 4 GB com MMU
Cache de instruções	-	De 512 bytes a 64KBytes	De 512 bytes a 64KBytes
Cache de dados	-	-	De 512 bytes a 64KBytes
Multiplicador por Hardware	-	3 ciclos	1 ciclo
Divisor por Hardware	-	Opcional	Opcional

Fonte: Adaptado CHU (11).

O processador pode ser configurado entre cada uma das versões, podendo-se incluir ou excluir certas características, como a unidade JTAG, e ajustar o tamanho e desempenho de certos componentes, como o tamanho do cache. Mesmo tendo tamanho e desempenho diferentes, as três versões compartilham o mesmo conjunto de instruções, o que as torna idênticas do ponto de vista do software, não sendo necessária a sua alteração caso haja necessidade de troca da versão do núcleo.

Embora o processador Nios II seja descrito em código HDL, os arquivos estão encriptados e a sua organização interna não pode ser modificada pelo usuário através dos códigos, sendo assim ele deve ser tratado como uma caixa preta que executa a instrução especificada (9, 11).

2.3.5 Organização das portas de entrada e saída do Nios II (I/O)

Para realizar operações de entrada e saída entre o processador e os dispositivos periféricos o Nios II utiliza um método de mapeamento das portas I/O pela memória. Estes dispositivos contém, normalmente, um conjunto de registradores que definem seus comandos, seu estado e seus dados. No esquema de mapeamento de I/O pela memória, o processador utiliza o mesmo espaço de endereços para acessar tanto a memória como esses registradores (11).

No SOPC Builder, a atribuição dos endereços é feita automaticamente. Ao adicionar uma porta de I/O, a interface checa o número de registradores desta porta e, então, aloca o espaço necessário na memória (11).

Estas portas são normalmente descritas em código HDL e implementadas como *softcores* pelo SOPC Builder, o qual disponibiliza os periféricos mais conhecidos, de modo que é necessário apenas estanciar o módulo na construção do sistema Nios II (11).

Um exemplo de um periférico disponibilizado é o núcleo PIO (*parallel input/output*), o qual fornece uma interface de interconexão para uma porta de I/O de propósito geral que

pode conectar o sistema a uma lógica dentro do chip ou a um dispositivo externo através dos pinos de I/O do FPGA.

Outro exemplo é o núcleo de *timer* interno, ao qual se pode atribuir varias tarefas, como medir o tempo entre eventos ou gerar pulsos periódicos. A parte principal deste núcleo é um contador capaz de contar decrescentemente de certo valor até zero, este valor é conhecido como período de *timeout* e é armazenando em um registrador especifico. Assim que o contador chega ao valor zero, um bit específico é levado a "1" e um pedido de interrupção opcional é acionado, gerando um pulso de saída opcional. Após chegar a "0", o contador pode permanecer com esse valor ou começar novamente a contar, dependendo do modo de operação selecionado. Adicionalmente podem ser adicionados sinais para parar, começar e recomeçar a operação do contador.

2.3.6 Estrutura de interconexão Avalon

Para interligar o processador ao módulo de memória, aos periféricos de I/O e aos aceleradores de hardware, a plataforma SOPC Builder define um conjunto de interfaces padrões conhecidas como interfaces Avalon. A Figura 2.19 mostra um diagrama de blocos de um sistema Nios II utilizando o Avalon.

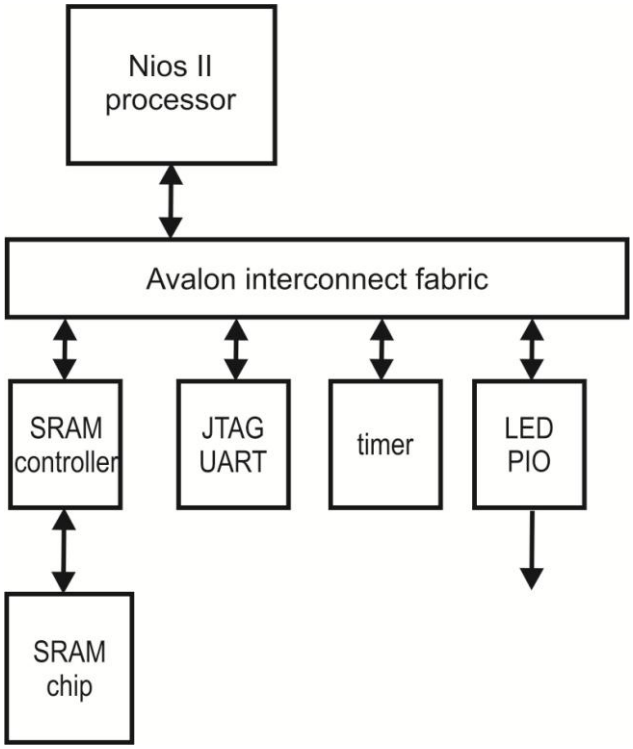


Figura 2.19 - Diagrama de blocos conceitual de um sistema com o processador Nios II. Fonte: CHU (11).

Para integrar uma porta periférica de I/O a um sistema baseado no Avalon, é preciso mapear os seus sinais de interfaceamento e ajustar suas propriedades temporais para que, então, a estrutura de interconexão realize a conexão entre as partes. Esta interconexão é feita através de decodificadores, multiplexadores, árbitros e lógica temporal e é construída de uma maneira distributiva, utilizando algumas vantagens da programação em FPGAs, além de ser customizada para ter as configurações individuais do sistema. O objetivo é eliminar a disputa por recursos centralizados e melhorar a performance e a escalabilidade do sistema. A implementação conceitual desta estrutura de interconexão é mostrada na Figura 2.20.

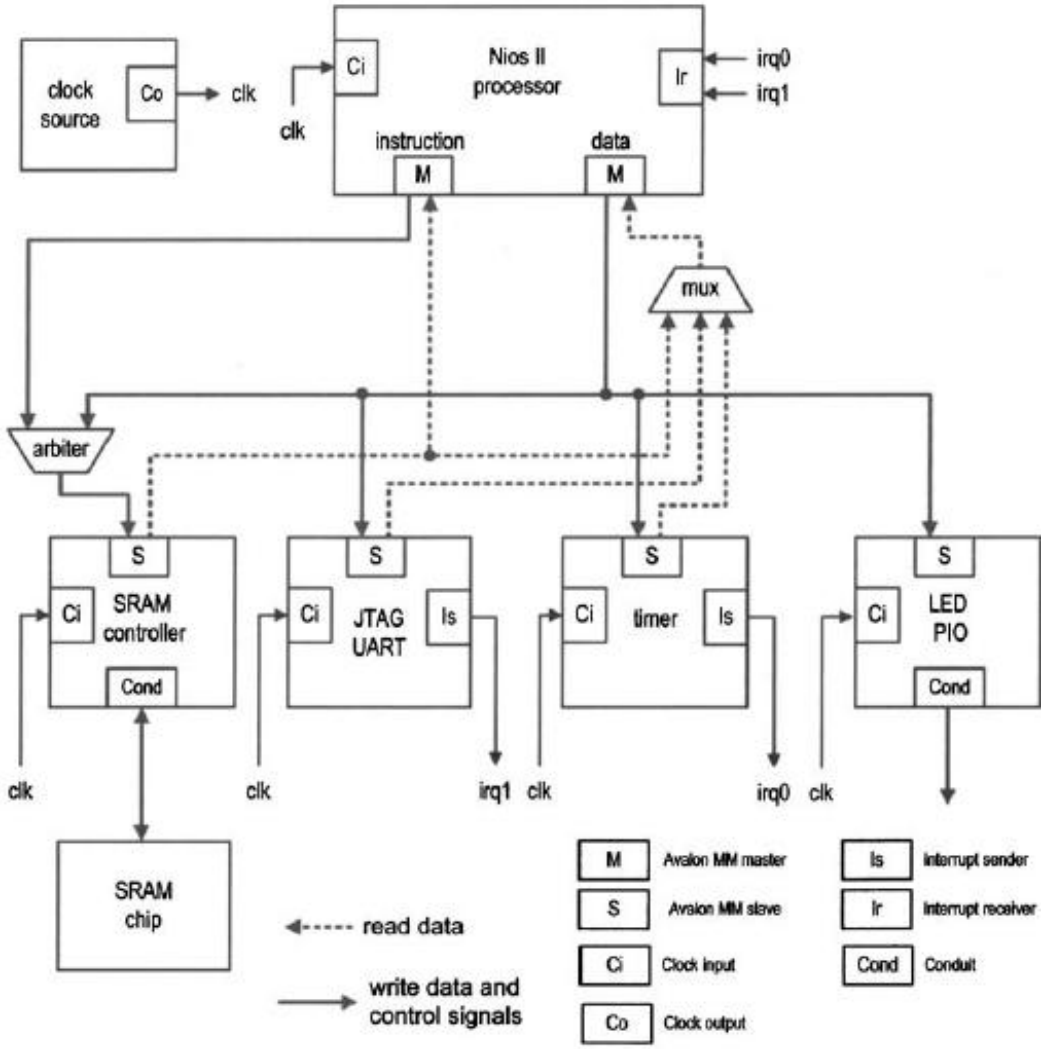


Figura 2.20 - Implementação conceitual de uma estrutura de interconexão Avalon utilizada no processador Nios II. Fonte: CHU (11).

Alguns exemplos de interfaces que utilizam o Avalon são:

- Interface de mapeamento de memória Avalon (Avalon MM): Esta interface define uma conexão baseada em endereço mestre-escravo. Um Avalon MM mestre usa um endereço para identificar um Avalon MM escravo e pode realizar operações de leitura ou escrita.

- Relógio Avalon (*Clock*): Esta interface define os sinais de *clock* e *reset* usados pelos componentes. Uma interface Avalon para saída de c*lock* gera o sinal de *clock* e uma interface Avalon para entrada de *clock* recebe o sinal de *clock.*

- Interrupção Avalon (*Interrupt*): Esta interface permite ao componente escravo enviar sinais para o componente mestre. Uma interface Avalon que envia a interrupção gera o sinal para pedido de interrupção enquanto que a receptora aceita e processa o pedido.

- Canal Avalon (*Conduit*): Esta interface agrupa e envia os sinais para fora de um sistema SOPC Builder.

Um único componente pode incluir múltiplas instâncias de um mesmo tipo de interface. No sistema representativo mostrado na Figura 2.20, a interface Avalon MM é a estrutura principal utilizada pelo processador Nios II para acessar os registradores de controle e de estado dos componentes *on-chip.* As interfaces de interrupção estão incluídas em alguns componentes e as duas interfaces de *clock*, geradas por uma PLL (*phase-locked loop*) externa, são distribuídas por todos os componentes. O sistema também contém uma interface de canal para acessos fora do chip FPGA (11).

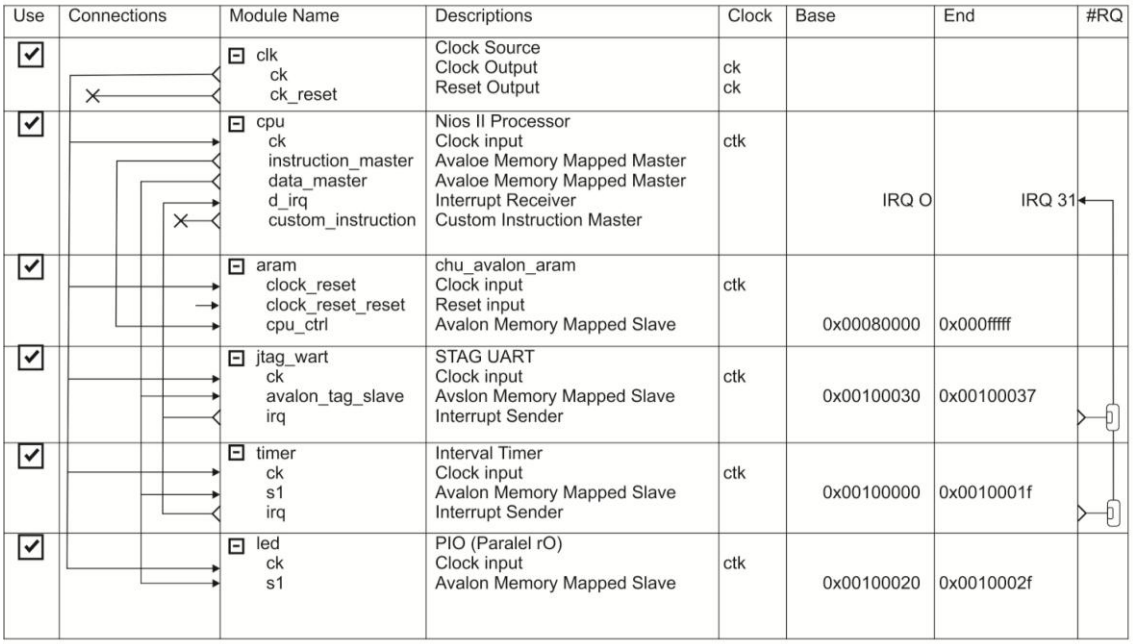


Figura 2.21 - Conexões detalhadas do Avalon, montadas através do SOPC Builder. Fonte: CHU (11).

Para construir um sistema embarcado na estrutura Avalon, é necessário apenas selecionar os núcleos IPC desejados e conectar as portas da interface. A ferramenta, SOPC Builder, irá automaticamente gerar as estruturas de interconexão de acordo com a configuração do sistema, como podemos ver na Figura 2.21 (11).

2.3.7 Desenvolvimento do software

Incluir um software, ao projeto, visa permitir o controle sobre os pacotes Ethernet, que são recebidos e enviados pela rede, além de permitir a configuração do MAC sem a necessidade de uma nova síntese do hardware. Então, para o desenvolvimento deste, a Altera disponibiliza uma ferramenta conhecida como Nios II EDS (*embedded design suite*) baseada em uma ferramenta GNU e customizada para o ambiente do processador Nios II, desta forma o software para o processador Nios II pode ser escrito em C ou C++.

Um projeto de software para o Nios II contêm duas partes principais: a aplicação do usuário e o BSP (*board support package*). O primeiro é o programa desenvolvido pelo usuário, enquanto que o segundo são os códigos de suporte para a configuração específica do sistema baseados nas informações do arquivo ".sopcinfo" do projeto. Os códigos de ambas as partes são compilados e *linkados* em uma única imagem com extensão ".elf" e então são carregados na memória principal do Nios II (11).

2.3.8 ChibiOS/RT

Para permitir o desenvolvimento modular do software, facilitando a este o controle sobre varias operações simultaneamente através de *threads* distintas, foi integrado ao projeto o sistema operacional de tempo real (RTOS – *Real Time Operacional System*) ChibiOS/RT. Este sistema foi projetado para aplicações embarcadas, nas quais uma execução eficiente e um código compacto estão entre os principais requisitos. Entre as características deste sistema estão a alta portabilidade e o tamanho compacto, como dito anteriormente, além disso, ele fornece suporte para projetos *multi-threads*, com uma arquitetura otimizada para trocas de contexto.

A arquitetura citada permite alguns estados a *thread,* como mostrado na Figura 2.22. A troca de um estado para o outro é feito ao se chamar uma função específica de cada transição, como as mostradas ao lado das setas de transição.

Além disso, todos os objetos do sistema, como *threads*, semáforos e *timers*, podem ser criados e deletados durante a execução do programa e o único limite é em relação a memória disponível.

O sistema dispõe também de cinco níveis para controle sobre a prioridade de execução de cada thread, os quais são listados a seguir, na ordem do nível de mais baixa prioridade para o nível de mais alta prioridade:

- IDLEPRIO
- LOWPRIO
- NORMALPRIO
- HIGHPRIO
- ABSPRO

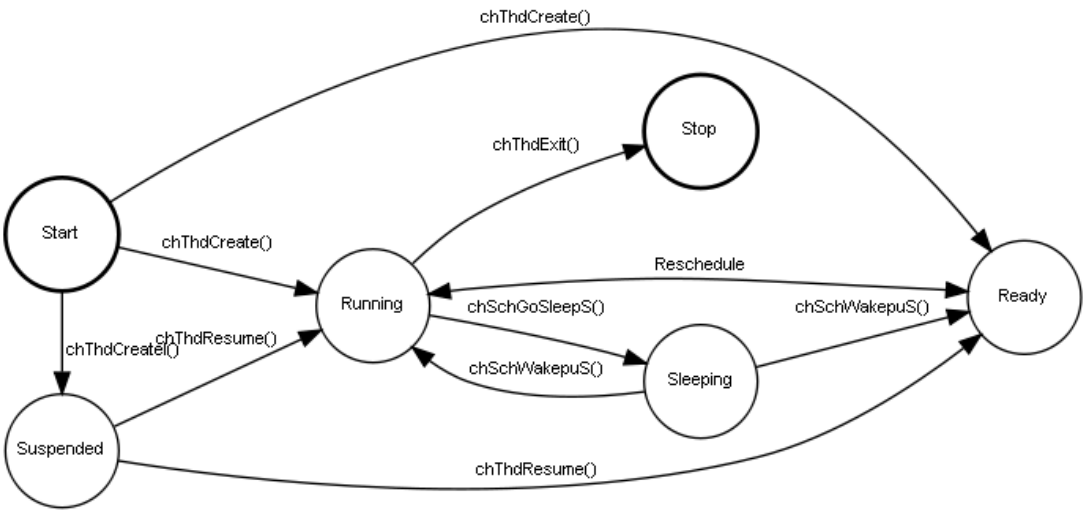


Figura 2.22 - Estados permitidos a uma thread. Fonte: CHIBIOS/RT (12).

Cada *thread* tem ainda um espaço na memória chamado de *thread working area*, no qual todas as suas estruturas são armazenadas enquanto ela não está rodando. Este é o único local da memória acessado pela *thread*, com a exceção de quando ela está acessando um dado compartilhado (12-13).

A versão do sistema portada para o Nios II foi obtida na referência (14).

2.4 HSMC-NET

A placa HSMC-NET da Terasic, mostrada nas Figura 2.23 e Figura 2.24, oferece uma conexão para a transferência de dados pela rede de até 1Gbps. Ela pode ser acoplada a um kit de desenvolvimento utilizando um conector Mezzanine de alta velocidade (HSMC -High Speed Mezzanine Connector) mostrado na

Figura 2.23 e fornece suporte para conexão de dois conectores RJ-45 (J2 e J3), como mostrado na

Figura 2.24. Além disso, para que seja possível a transferência de dados por uma rede Ethernet, esta placa dispõe de dois chips físicos 88E1111, um para cada conector. Desta forma, a placa permite que o kit de desenvolvimento possa se comunicar com outro dispositivo através de uma rede Ethernet.

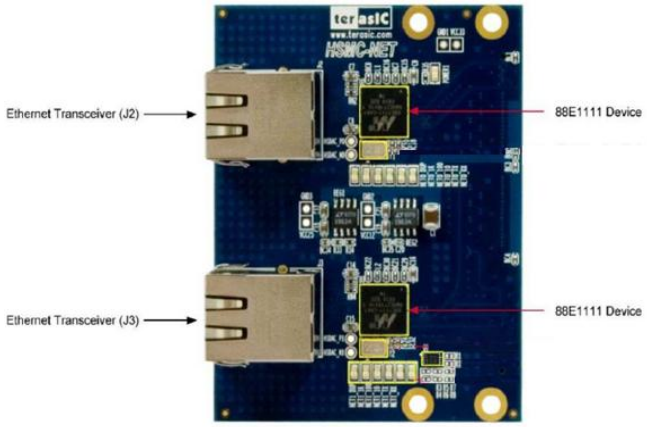


Figura 2.23 - Componentes da placa HSMC-NET. Fonte: Terasic (15).

O *chip* físico, disponível na placa, opera na camada física de rede mencionada no modelo OSI e é responsável por implementar as funções de transferências de pacotes Ethernet nesta camada. Ele interfaceia a comunicação entre a camada física e o MAC, recebendo os sinais vindos do MAC na forma binaria e enviando-os pela camada física utilizando modulações de frequência e amplitude para o caso da transmissão de dados. No caso da recepção de dados o processo é inverso, (14-15).

Figura 2.24 - Conector da placa HSMC_NET. Fonte: Terasic (15).

O chip utilizado pela placa HSMC é o *Alaska Ultra 88E1111 Gigabit Ethernet Transceiver*, o qual permite a transferência de dados em 10/100/1000 Mbps através de diferentes modos de operação. Os modos de operação utilizados neste trabalho, juntamente com suas velocidades e pinos utilizados para troca de dados entre o MAC e o PHY, estão descritos nas próximas seções.

Tanto o modo quanto a velocidade de operação do *chip* podem ser determinados através de dois meios diferentes. O primeiro é através da auto-negociação, um recurso disponibilizado pelo *chip* que, ao se conectar pela primeira vez a uma rede, realiza alguns testes afim de identificar as características desta rede, definindo sua velocidade e modo de operação. O segundo meio de configurar essas duas características do *chip* é escrevendo diretamente no registrador zero do *chip*, além disso, é necessário desabilitar o recurso de auto-negociação para que a alteração no registrador tenha efeito.

2.4.1 Gigabit Media Independent Interface (GMII/MII)

A Tabela 2.3 mapeia os sinais do dispositivo 88E1111 para as interfaces GMII e MII que correspondem a velocidades de 1Gbps e 100/10Mbps, respectivamente. Para operar a

GMII os quatro bits menos significativos do registrador zero, chamados de HWCFG_MODE, devem ter o valor “1111”, já para o modo MII, o valor de HWCFG_MODE deve ser “0111”.

Tabela 2.3 - Sinais necessários para transferência de dados para o dispositivo 88E1111, considerando as interfaces GMII e MII.

Pinos do dispositivo 88E1111	Interface GMII	Interface MII
GTX_CLK	GTX_CLK	-
TX_CLK	-	TX_CLK
TX_ER	TX_ER	TX_ER
TX_EN	TX_EN	TX_EN
TXD[7:0]	TXD[7:0]	TXD[3:0]
RX_CLK	RX_CLK	RX_CLK
RX_ER	RX_ER	RX_ER
RX_DV	RX_DV	RX_DV
RXD[7:0]	RXD[7:0]	RXD[3:0]
CRS	CRS	CRS
COL	COL	COL

Fonte: Adaptado MARVELL (16).

É mostrado na Figura *2.25* o diagrama de sinais para o modo GMII, onde é possível ver o sentido de operação de cada sinal. A frequência do sinal de *clock* GTX_CLK é de 125MHz, assim como a frequência do sinal de *clock* RX_CLK. Portanto, a cada transição positiva, quando o sinal passa do valor ‘0’ para o valor ‘1’, do *clock* são enviados ou recebidos 8 bits através das portas TXD[7:0] ou RXD[7:0]. Tem-se, então, $125*10^6*8$ bits sendo transferidos em um segundo, de modo que a velocidade de operação é de 1Gbps.

Tratando dos sinais de controle que partem do MAC para o PHY, considere os sinais TX_EN e TX_ER. Aquele indica quando o dado está disponível para leitura e este indica se houve algum erro durante a transmissão do pacote.

Analisando agora os sinais que realizam o caminho inverso, no caso RX_ER e RX_DV. O primeiro indica se ouve erro durante a recepção do dado, enquanto que o segundo é o sinal de dado válido, ou seja, quando seu estado é “1”, o dado está presente no barramento RXD[7:0] e pode ser lido.

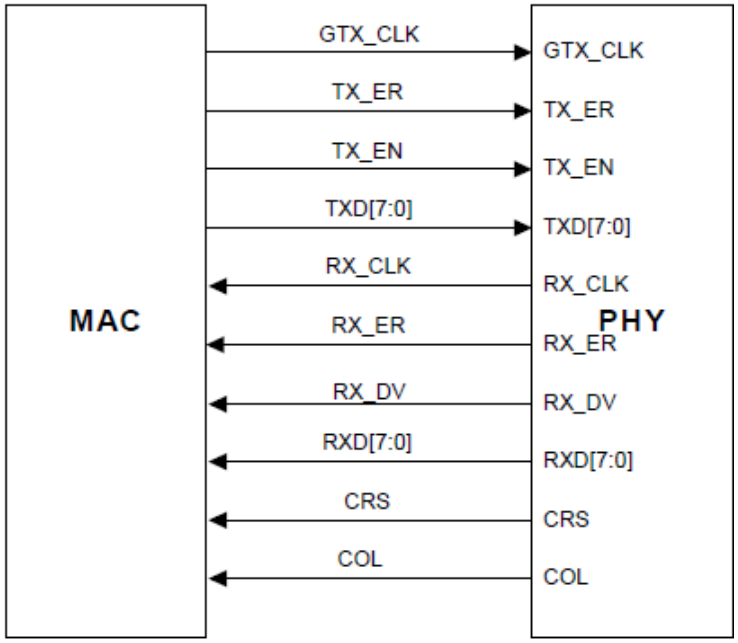


Figura 2.25 - Diagrama de sinais para a interface GMII. Fonte: elaborada pelo autor.

Tem-se, ainda, o sinal de COL que indica se houve uma colisão de pacotes, ou seja, quando ambos os meios, de transmissão e recepção, tentaram operar ao mesmo tempo. Isso só é possível quando o modo halfduplex está habilitado. E por último, o sinal de CRS, que é elevado a "1" quando o meio de recepção está sendo utilizado e, caso o módulo esteja operando em halfduplex, quando o meio de transmissão está sendo utilizado, de modo a indicar quando o meio está sendo utilizado.

Já para o caso da interface MII tem-se poucas diferenças, como é possível ver no diagrama de sinais mostrado na

Figura 2.26. Neste caso o *clock* é regido pelo PHY, o qual manda dois sinais, um para a transmissão de pacotes, TX_CLK, e outro para a recepção de pacotes, RX_CLK, ambos operando a frequência de 25MHz ou 12,5MHz, dependendo da velocidade de operação que pode ser de 100Mbps ou de 10Mbps respectivamente. Outra diferença está na quantidade de bits destinada tanto para transmissão quanto para recepção de dados. No caso anterior eram oito bits para esse propósito, enquanto que para esse caso são apenas quatro, o que multiplicado pela frequência de *clock* fornecem as velocidades de transferência de 100 e 10Mbps. Os demais sinais funcionam de maneira idêntica ao modo anterior.

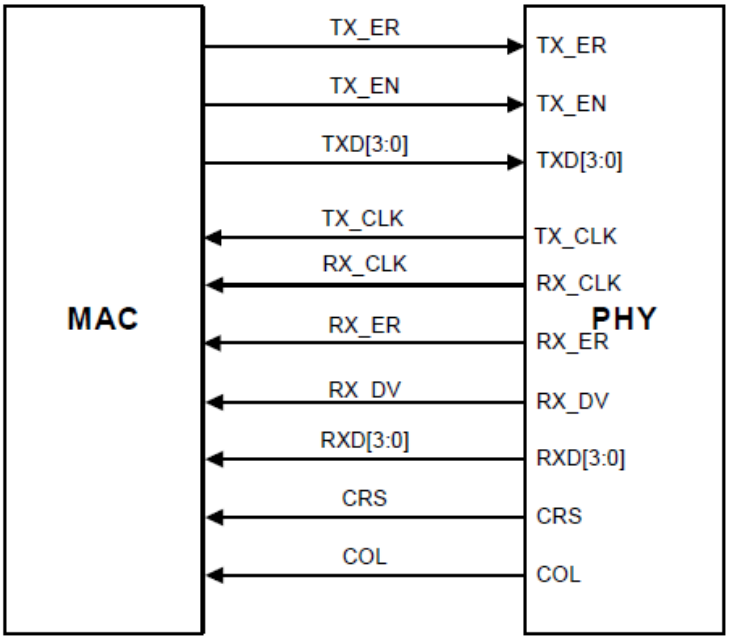


Figura 2.26 - Diagrama de sinais para a interface MII. Fonte: Elaborada pelo autor.

2.4.2 Reduced Pin Count GMII (RGMII)

O mapeamento dos sinais para o caso do modo de operação de contagem de pinos reduzido (*Reduced Pin Count*) é mostrado na Tabela 2.4, na qual é possível ver as interfaces para os modos de RGMII e RMII que correspondem a velocidades de 1000Mbps e 100/10Mbps, respectivamente. Para operar em um dos dois modos é necessário que o estado de HWCFG_MODE esteja em "1011", já a opção de um ou de outro depende da velocidade configurada no registrador zero.

Tabela 2.4 - Sinais necessário para transferência de dados para o dispositivo 88E1111, considerando as interfaces RGMII e RMII.

Pinos do dispositivo 88E1111	Interface RGMII/RMII
GTX_CLK	TXC
TX_EN	TX_CTL
TXD[3:0]	TD[3:0]
RX_CLK	RXC
RX_DV	RX_CTL
RXD[3:0]	RD[3:0]

Fonte: Adaptado MARVELL (16).

A quantidade de sinais utilizados para comunicação entre o MAC e o PHY é reduzida a seis nestes modos de operação e o diagrama desses sinais é mostrado na Figura 2.27. É possível subdividir esses sinais em dois grupos onde um estabelece a transferência de dados do MAC para o PHY e o outro estabelece a transferência de dados do PHY para o MAC. No primeiro, se tem os sinais TXC, TX_CTL e TD[3:0]. O TXC estabelece o *clock* de transmissão que pode variar entre 125MHz, 25MHz e 12,5MHz, já o sinal TD[3:0] define um barramento de transmissão de 4 bits, os quais transmitem dados tanto em transições positivas quanto em transições negativas do *clock*, isso acontece quando a velocidade de operação é de 1Gbps, já quando a velocidade de operação for de 100 ou 10Mbps a transmissão acontece apenas nas transições de subida. Se tem, ainda neste grupo, o sinal TX_CTL que define um sinal de controle de transmissão do qual é possível obter duas informações: a primeira é encontrada na transição positiva do *clock* e é equivalente ao sinal de TX_EN e a segunda é encontrada na transição negativa do *clock* e é codificada por um XOR com o sinal TX_EN e equivale ao sinal TX_ER.

No grupo que estabelece a transferência de dados para o MAC se tem os sinais RXC, RX_CTL e RD[3:0]. O RXC é o sinal de *clock* de recepção que, como o *clock* de transmissão, pode variar entre 125MHz, 25MHz e 12,5MHz, o sinal RD[3:0] funciona de maneira idêntica ao sinal TD[3:0] com a diferença de que este estabelece um canal para recepção de dados. Por fim se tem o sinal RX_CTL, do qual se pode obter os sinais TX_EN e TX_ER através do mesmo procedimento utilizado no sinal TX_CTL.

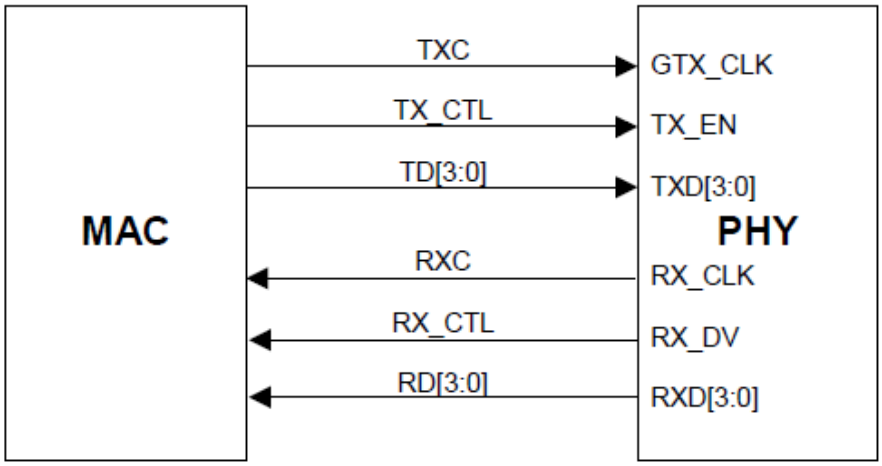


Figura 2.27 - Diagrama de sinais para as interfaces RGMII e RMII. Fonte: Elaborada pelo autor.

3 DESENVOLVIMENTO

Para o desenvolvimento dos códigos em HDL deste trabalho, foi utilizada a ferramenta de programação de dispositivos lógicos (PLD- Programmable Logic Device) Quartus II. Este software foi produzido pela Altera (17) para análise e síntese de projetos em HDL, com o qual é possível compilar projetos, examinar diagramas no nível de transferência de registrador (RTL- Register Transfer Level) e configurar um dispositivo alvo. Além disso, foi utilizado o software Eclipse adaptado com um compilador GNU para o Nios II para o desenvolvimento de um software embarcado para o processador Nios II.

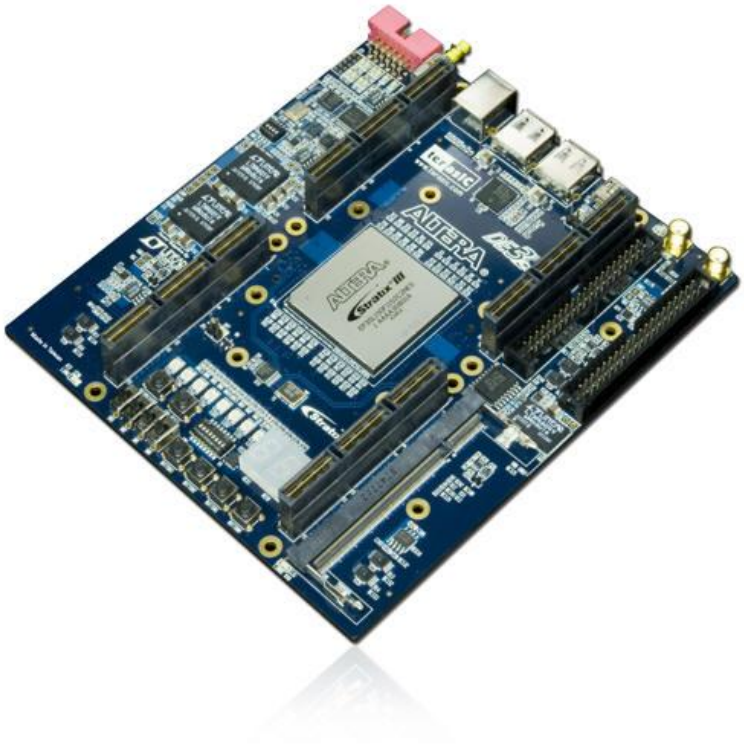


Figura 3.1 - Kit de desenvolvimento DE3-150 da Terasic, utilizado para verificação e validação do sistema desenvolvido. Fonte: TERASIC-SOCKIT (18).

A verificação e validação do sistema foram feitas em um kit de desenvolvimento DE3-150 da Terasic (19) mostrado na Figura 3.1, o kit contém um FPGA Stratix III EP3S150F1152C2 da Altera e permite a integração de quatro periféricos externos através de dutos de alta velocidade HSMC. Através desta última funcionalidade, foi possível acoplar ao kit uma placa HSMC-NET descrita na seção 1.6. A placa HSMC-NET acoplada ao kit é

mostrada na Figura 3.2. Esta permite conectar o sistema a uma rede local ou diretamente a um computador através de um cabo de par trançado.

Figura 3.2 - Placa HSMC-NET da Terasic, mostrada conectando-se ao kit DE3 através do duto HSMC. Fonte: TERASIC-SOCKIT (18).

A proposta do projeto é desenvolver um módulo Ethernet que possibilite a transferência de dados nas velocidades de 1000 Mbps, 100 Mbps e 10 Mbps e que possa operar nos modos de *Media Independent Interface* (MII), *Gigabit* MII (GMII), *Reduced* MII (RMII), e RGMII. Também faz parte da proposta a integração de um processador ao sistema, que será utilizado para controlar a transferência de dados, viabilizando a validação do sistema.

Nas seções seguintes os módulos do sistema são apresentados e detalhados.

3.1 Top file

Os módulos da arquitetura do sistema embarcado proposto são apresentados na Figura 3.3, cuja arquitetura conta com três módulos auxiliares, IOV_VoltConfig, gen_reset_n e enet_tx_clk_pll, os quais estão descritos a seguir.

- IOV_VoltConfig

O kit fornece vários barramentos para integrar dispositivos externos, os quais podem operar a tensões diferentes, cabendo ao desenvolvedor configurar corretamente a placa. Desta forma, o primeiro módulo auxiliar é também o primeiro a ser iniciado, ele conversa diretamente com o regulador de voltagem do kit para fazer a configuração da voltagem de cada barramento da placa para que a tensão fornecida aos dispositivos periféricos seja a correta.

O módulo recebe como entrada um sinal de *clock* vindo da placa FPGA, um sinal de dados vindos do regulador de voltagem e quatro valores que definem a voltagem de cada um dos quatro barramentos da placa. A tensão nesses barramentos pode assumir valores distintos dependendo do valor de configuração, como mostra a Tabela 3.1.

Tabela 3.1 - Tabela que indica o valor da tensão no barramento para cada valor de configuração do bloco IOV_VoltConfig.

Valor de configuração	**Tensão no barramento**
00b	1,5V
01b	1,8V
10b	2,5V
11b	3,3V

Fonte: Elaborada pelo autor.

Como saída o módulo envia ao regulador de voltagem um sinal de *clock*, um outro de controle e um outro de dados para a configuração do mesmo. Um sinal indicando o término da operação de configuração também é fornecido ao módulo gen_reset_n, o qual tem nível lógico “0”, enquanto a configuração da placa está sendo feita, e nível lógico “1”, quando a placa já está configurada.

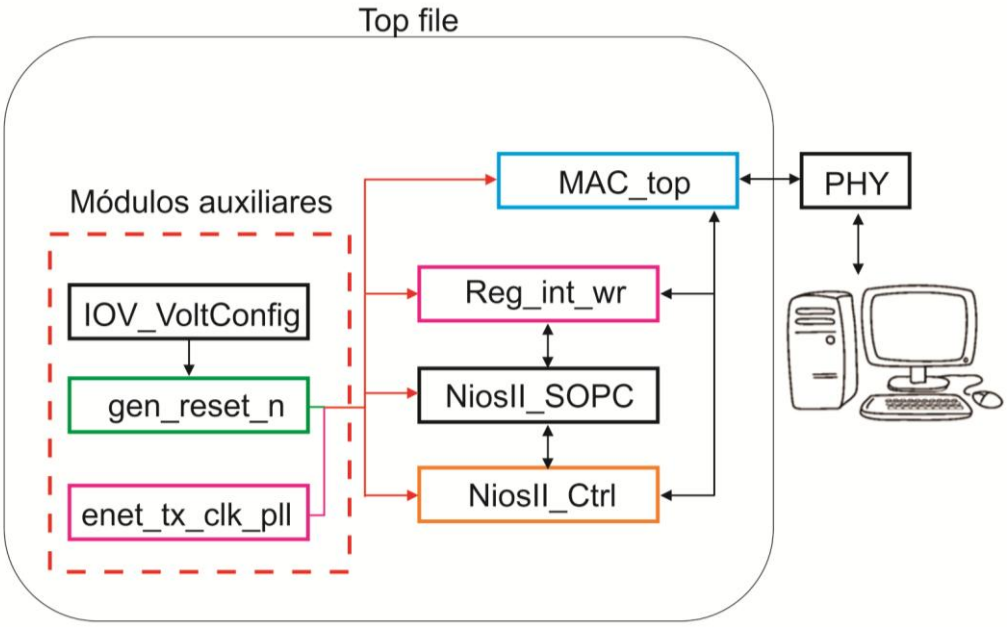


Figura 3.3 - Diagrama de blocos dos módulos do sistema no nível mais alto da arquitetura. Fonte: Elaborada pelo autor.

- gen_reset_n

Este módulo realiza a distribuição do sinal de *reset* para o sistema, desta forma ele é capaz de retardar o início de cada módulo de modo a aguardar que a configuração de voltagem dos barramentos seja feita. O *reset* pode ser realizado através do botão *cpu_reset* do kit de desenvolvimento.

É realizado um *AND* lógico entre o sinal recebido do módulo IOV_VoltConfig e o sinal vindo do botão de reset da placa antes que estes cheguem ao gen_reset_n, dessa forma, caso o sinal recebido pelo módulo permaneça em nível lógico "1" por mais de um ciclo de *clock*, a saída do módulo também terá nível lógico "1", liberando todo o sistema para operar, caso contrário, o sistema permanecerá em estado de *reset* até as condições necessárias serem satisfeitas.

- enet_tx_clk_pll

O terceiro módulo auxiliar, chamado de enet_tx_clk_pll é o gerador de *clocks* do sistema, ele recebe um clock de 50MHz vindo da DE3-150 e, através deste *clock*, gera os clocks de 125MHz com 0º de fase, 125MHz com -90º de fase, 250MHz com -15º de

fase, 25MHz com 0º de fase e 2,5MHz com 0º de fase, os quais são utilizados nos diferentes módulos do sistema. As fases mencionadas acima são medidas em relação ao sinal de entrada.

Nesta camada, tem-se também o módulo MAC_top, o qual representa o módulo Ethernet propriamente dito. Este módulo se comunica com o chip físico (PHY), localizado na placa HSMC-NET, para enviar e receber dados pelo meio físico. O MAC é controlado pelo processador Nios II, aqui representado pelo módulo NiosII_SOPC. O processador é capaz de trocar dados pela rede utilizando o MAC, imprimindo na tela as mensagens recebidas e transmitidas, além de ser capaz de responder pacotes do tipo Ping e ARP. Tal processador necessita de mais dois módulos complementares, o NiosII_Ctrl e Reg_int_wr, para interfacear a comunicação com o MAC_top. O primeiro módulo realiza a intermediação dos pacotes recebidos e transferidos pela rede enquanto que o segundo auxilia na leitura e escrita nos registradores internos do MAC_top. A necessidade desses dois últimos módulos vem do fato de estar sendo executado um software dentro do processador, o qual é mais lento que a execução em hardware, portanto é necessário esperar que o software realize a leitura ou escrita dos dados atuais e avise o término desta para que então um novo dado possa ser recebido ou transmitido.

Está indicado na Figura 3.3 o chip físico que faz a conversão do sinal digital, vindo do FPGA, para o sinal analógico utilizado para transferência de dados na camada física. Este também faz a conversão oposta de modo que é possível receber e enviar dados a um determinado computador.

3.2 MAC_top

O módulo de Ethernet MAC foi adaptado do projeto Ethernet_tri_mode (20), retirado da referência (21).

O módulo, inicialmente em Verilog, foi traduzido para VHDL. Na sequência foram realizadas as adaptações necessárias para que se alcançasse a proposta inicial do projeto. A seguir, se tem uma descrição detalhada do funcionamento do módulo e de seus submódulos.

Este módulo realiza toda a comunicação para transferência de dados com PHY, desde a sua configuração até a transferência propriamente dita. Seu diagrama de blocos é mostrado na Figura 3.4, na qual se observa as comunicações dos submódulos deste módulo.

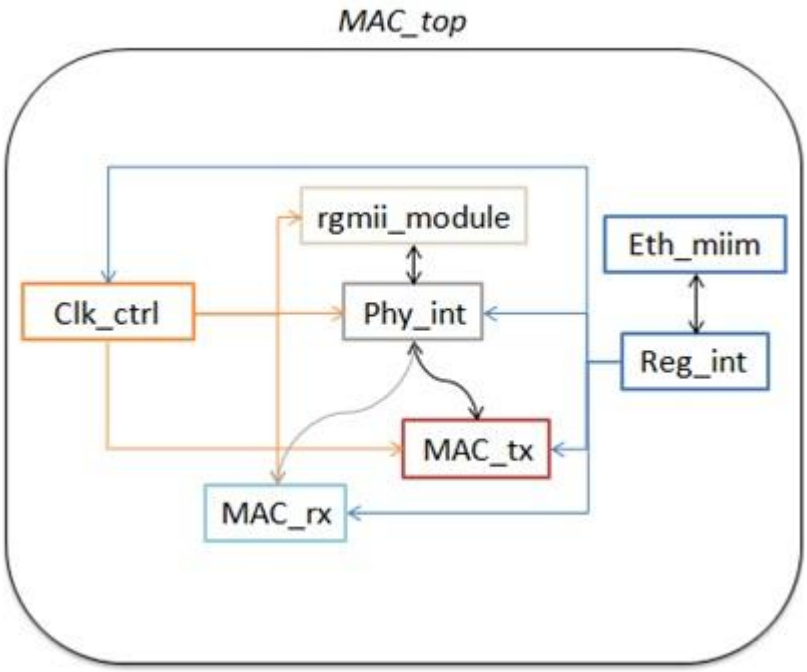


Figura 3.4 - Diagrama de blocos do bloco MAC_top, o qual realiza toda a comunicação para transferência de dados com PHY. Fonte: elaborada pelo autor.

Entre os submódulos estão o Clk_ctrl, que faz o controle e distribuição de todos os *clocks* utilizados internamente, o Rgmii_module, que faz a conversão do sinal de GMII para RGMII ou de MII para RMII dependendo do modo de operação do sistema, o PHY_int, que faz a conversão do barramento de 8 bits para 4 bits quando necessário, o Eth_miim, que realiza operações de escrita e leitura nos registradores internos do chip físico. Tem-se ainda os submódulos MAC_rx e MAC_tx que realizam o controle sobre o sistema de recepção e transmissão de pacotes, respectivamente. E, por último, se tem o submódulo Reg_int que define um conjunto de registradores internos capaz de configurar vários aspectos do MAC. Cada um desses submódulos está detalhado nas próximas seções.

3.2.1 Eth_miim

Este módulo é utilizado para controlar a interface de gerenciamento do chip físico, através da qual é possível acessar os registradores internos, tanto para leitura como para escrita. Através deste é possível configurar, por exemplo, a velocidade e o modo de operação desejado, além de poder realizar a leitura desses registradores quando a função de auto negociação está habilitada.

Para a operação de leitura o módulo recebe como entrada o endereço do registrador a ser lido através do barramento Rgad e o sinal RStat indicando que uma operação de leitura deve ser realizada. Na sequência, ele envia o sinal UpdateMIIRX_DATAReg, indicando que o pedido de leitura foi recebido e está em execução. O sinal RStat deve, então, ser limpado e o sinal Busy indica se a operação ainda está em execução. Quando Busy estiver com o valor "0", o valor do registrador está presente no barramento Prsd. A operação descrita está esquematizada no Figura 3.5.

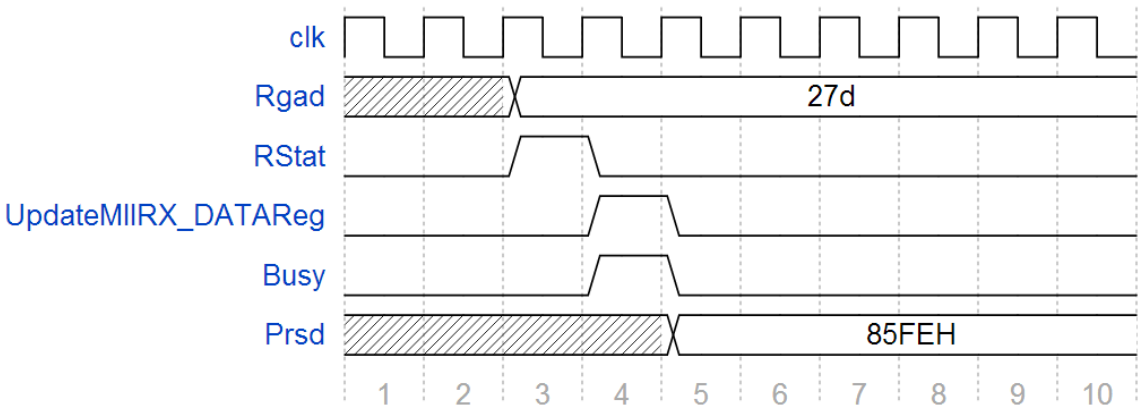


Figura 3.5 - Formas de onde de uma operação de leitura de um registrador do chip físico, utilizando o módulo Eth_miim. Fonte: elaborada pelo autor.

Já para a operação de escrita o processo é semelhante, o módulo recebe como entrada o endereço do registrador a ser lido através do barramento Rgad, o valor a ser escrito neste registrador pelo barramento CtrlData e o sinal WCtrlData indicando que uma operação de escrita deve ser realizada. Na sequência, ele envia o sinal WCtrlDataStart, indicando que o pedido de escrita foi recebido e está em execução. O sinal WCtrlData deve, então, ser limpado

e o sinal Busy indica se a operação ainda está em execução. Quando o valor de Busy for “0”, significa que o dado foi escrito no registrador. A operação descrita está esquematizada no Figura 3.6.

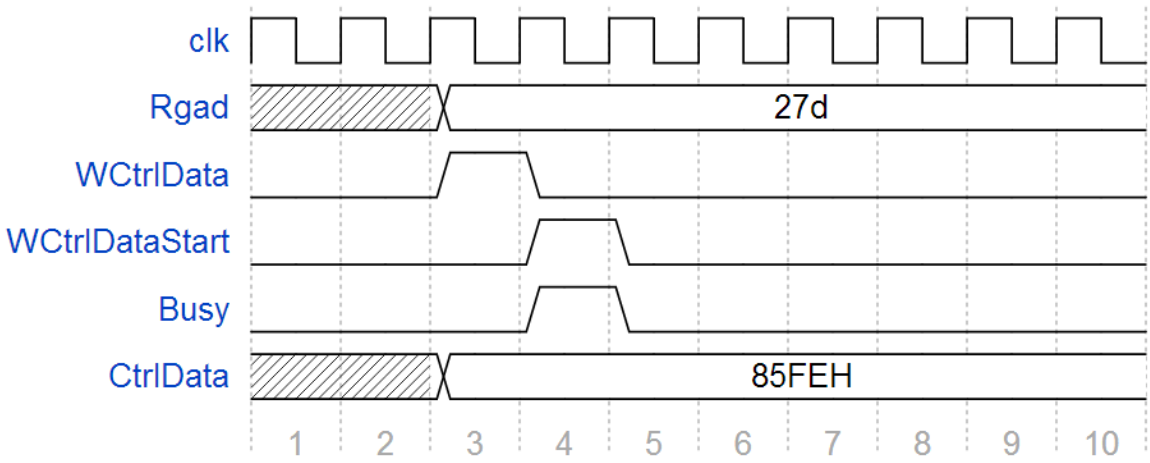


Figura 3.6 - Forma de onda de uma operação de escrita em um registrador do chip físico, utilizando o módulo Eth_miim. Fonte: elaborada pelo autor.

3.2.2 Clk_ctrl

O módulo Clk_ctrl controla toda a distribuição de *clocks* dentro do MAC_top, ele recebe como entrada todos os *clocks* que podem ser utilizados e, observando o registrador Speed, define qual será o *clock* de saída. Além disso, ele também protege o sistema contra pulsos aleatórios (*glitchs*) que podem ocorrer quando há alteração dos *clocks* utilizados. Um exemplo de geração de pulsos aleatórios é mostrado na Figura 3.7.

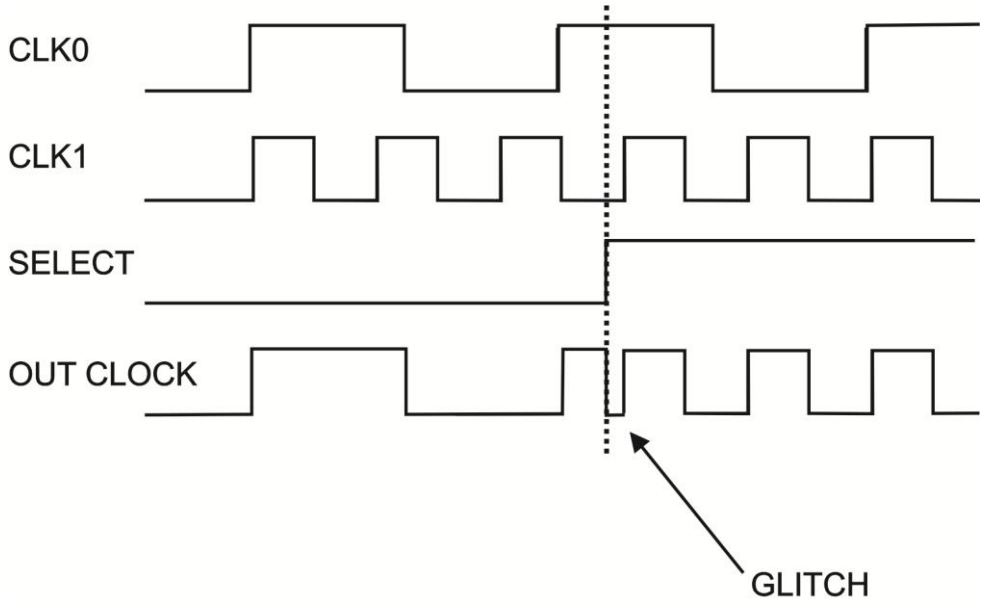


Figura 3.7 - Exemplo da forma de onda da geração de um pulso aleatório gerado na troca de clocks.
Fonte: MAHMUD (22).

Uma implementação simples de troca de *clocks* pode ser construída como indicado na Figura 3.8, a qual utiliza apenas um multiplexador lógico do tipo *AND-OR*. Esse multiplexador tem um sinal, chamado de SELECT, que define qual dos *clocks* irá propagar. Nesta forma de implementação, um pulso aleatório pode ser gerado quando o sistema, que estava operando com o "*clock*0", passar a operar com o "*clock*1" como é mostrado na Figura 3.7. Esse pulso aleatório pode gerar danos a todo o sistema, uma vez que uma parte do sistema pode interpretá-lo como um sinal de *clock* e outra parte não, sendo impossível prever os danos a serem causados por este erro.

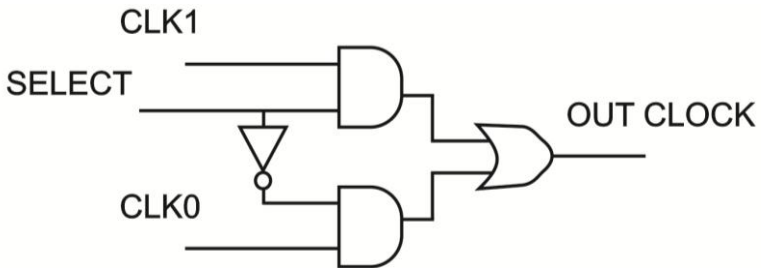


Figura 3.8 - Exemplo de um circuito que pode gerar pulsos aleatórios durante a troca de clocks.
Fonte: MAHMUD (22).

No caso do sistema apresentado neste trabalho, todos os clocks utilizados tem relação entre si, ou seja, eles são múltiplos uns dos outros. Sendo assim, um circuito que soluciona esse problema, para clocks é apresentado na Figura 3.9.

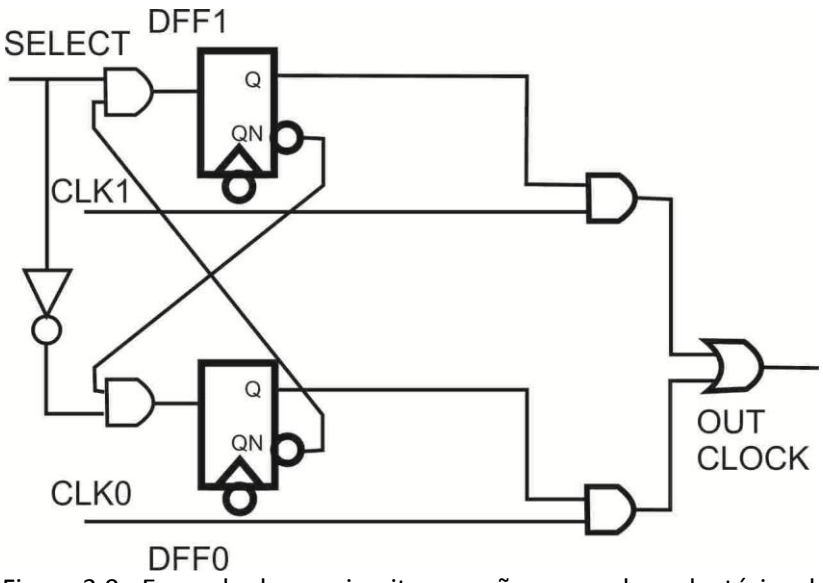


Figura 3.9 - Exemplo de um circuito que não gera pulsos aleatórios durante a troca de clocks, se os clocks de entrada forem relacionados. Fonte: MAHMUD (22)

Nesse circuito foi adicionado no caminho de seleção de ambas as fontes um *flip-flop* do tipo D que é acionado na borda de descida do *clock*. Isso faz com que a troca ocorra apenas na borda de descida do *clock* e depois que o *clock* que estava funcionando anteriormente estava com nível lógico "0".

O sinal de saída desta nova configuração é mostrado na Figura 3.10 (22). Esse circuito é utilizado em todas as trocas de *clock* realizadas no sistema.

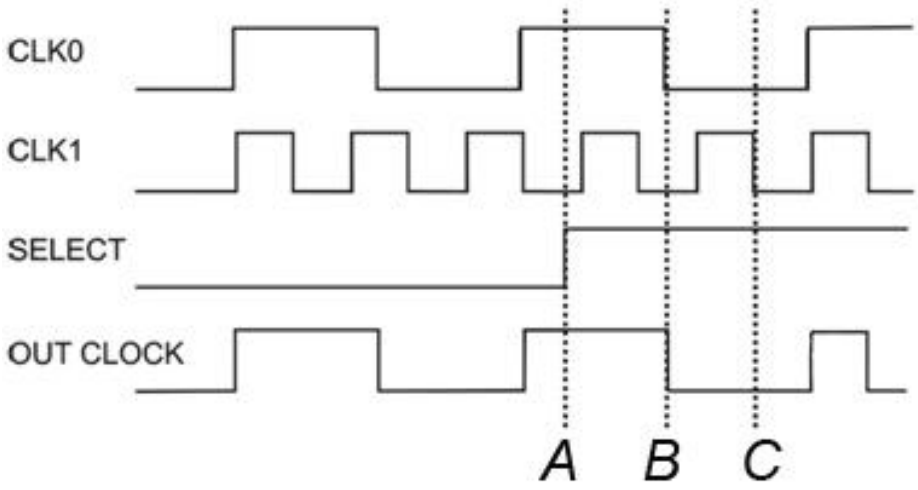


Figura 3.10 - Exemplo da troca de clock sem a geração de pulsos aleatórios, onde “A” representa a borda de subida do pulso select, “B” a borda de descida do clock 0 e “C” a borda de descida do clock 1. Fonte: MAHMUD (22).

3.2.3 Rgmii_module

Todos os sinais de comunicação entre o MAC_top e o chip físico passam por esse módulo. Se o sistema não estiver funcionando em RGMII, este módulo não realiza operação alguma nos sinais, mas caso esteja funcionando em RGMII, este módulo faz a conversão dos sinais de GMII para RGMII. Assim, a alteração no modo de operação, é transparente para o resto do sistema.

Para fazer a conversão do sinal de RGMII para GMII na recepção de pacote, por exemplo, o sistema soma os 4 bits que chegam na transição positiva do *clock* com os 4 bits que chegam na transição negativa e os coloca em um vetor de 8 bits para, então, repassar os dados para o sistema. Essa operação está representada na Figura 3.11. Observe que a operação gera um atraso de um ciclo de *clock* na recepção de dados.

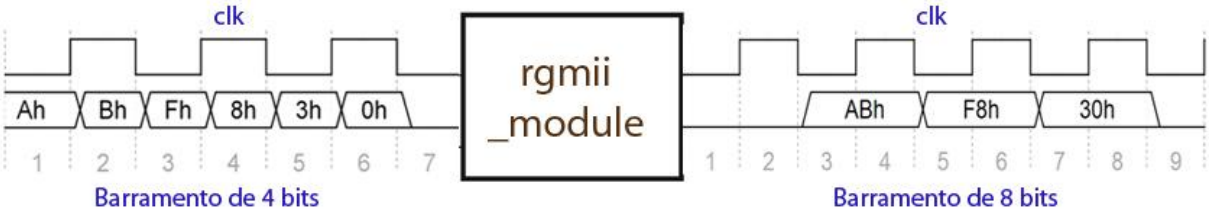


Figura 3.11 - Exemplo de funcionamento do modulo rgmii_module para transmissão de pacotes, quando o modo de operação do sistema está em RGMII. Fonte: elaborada pelo autor.

Para realizar a conversão do sinal GMII para RGMII na transmissão de dados, o sistema realiza a operação inversa, dividindo os 8 bits de dados a serem transmitidos em dois grupos de 4 bits cada, os quais são enviados, um na transição positiva e outro na transição negativa do *clock*.

3.2.4 Phy_int

Uma vez que os módulos de recepção e transmissão operam com um barramento de transferência de dados de 8 bits, é necessário que, para velocidades de 100Mbps e 10Mbps, a transmissão desse vetor seja quebrada em duas transmissões subsequentes de 4 bits cada. Já para a recepção, é necessário que as recepções de 4 bits sejam agrupadas em pares, formando um vetor de 8 bits que deve ser enviado para o sistema de recepção de dados. A operação de transmissão mencionada acima é mostrada na Figura 3.12. Essa operação só é necessária quando o sistema está funcionando a uma velocidade de 100Mbps ou 10Mbps, para os quais são recebidos 4 bits a cada transição positiva do *clock*. Já para a velocidade de operação de 1000Mbps, o módulo é transparente para o sistema, não realizando operações sobre os dados. Essa arquitetura só é possível porque os módulos de recepção e transmissão de dados operam com um *clock* igual ao *clock* de operação do Phy_int dividido por dois, quando a velocidade de operação é de 100Mbps ou 10Mbps, de forma a disponibilizar tempo para agrupar ou dividir o byte.

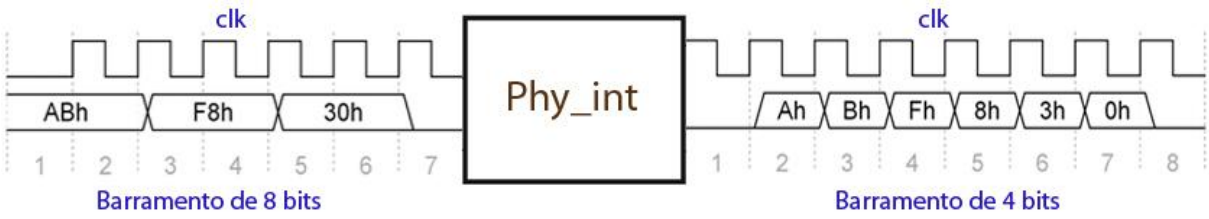


Figura 3.12 - Exemplo de funcionamento do módulo Phy_int para transmissão de pacotes, quando a velocidade de operação é de 100Mbps ou 10Mbps. Fonte: elaborada pelo autor.

3.2.5 Mac_rx

Este módulo é responsável por toda a parte de recepção de pacotes, incluindo a checagem do CRC, verificação do endereço para o qual o pacote foi encaminhado, entre outros. Ele é composto por cinco submódulos, como mostrado Figura 3.13, dos quais cada um é responsável por uma tarefa distinta.

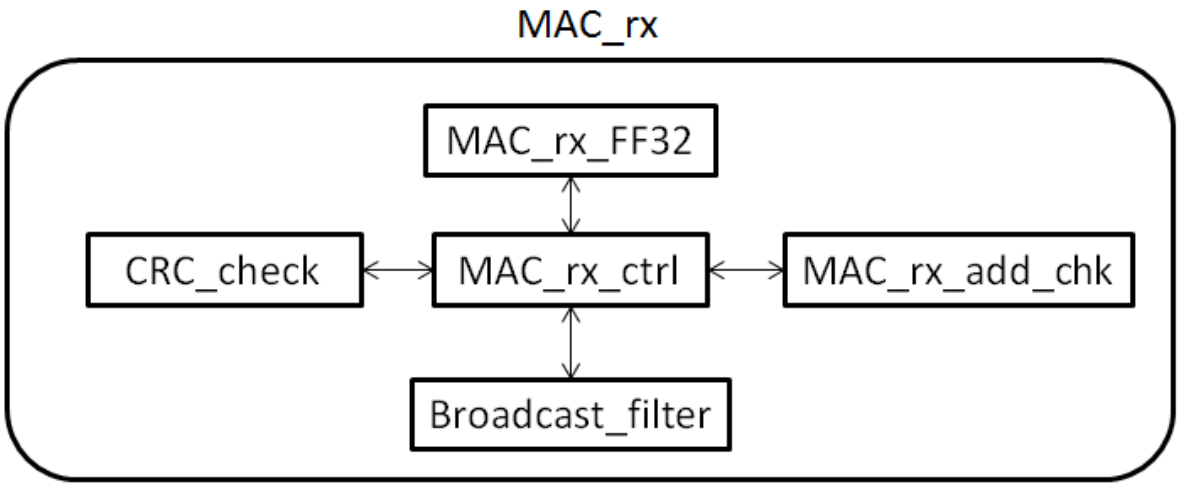


Figura 3.13 - Diagrama de blocos do módulo MAC_rx, o qual é responsável pela parte de recepção de pacotes do sistema. Fonte: elaborada pelo autor.

3.2.5.1 MAC_rx_ctrl

Esse módulo realiza o controle sobre a recepção de pacotes e dispõe de duas máquinas de estado para gerenciar as diferentes tarefas, uma máquina de estado principal e outra auxiliar. A máquina principal, mostrada na

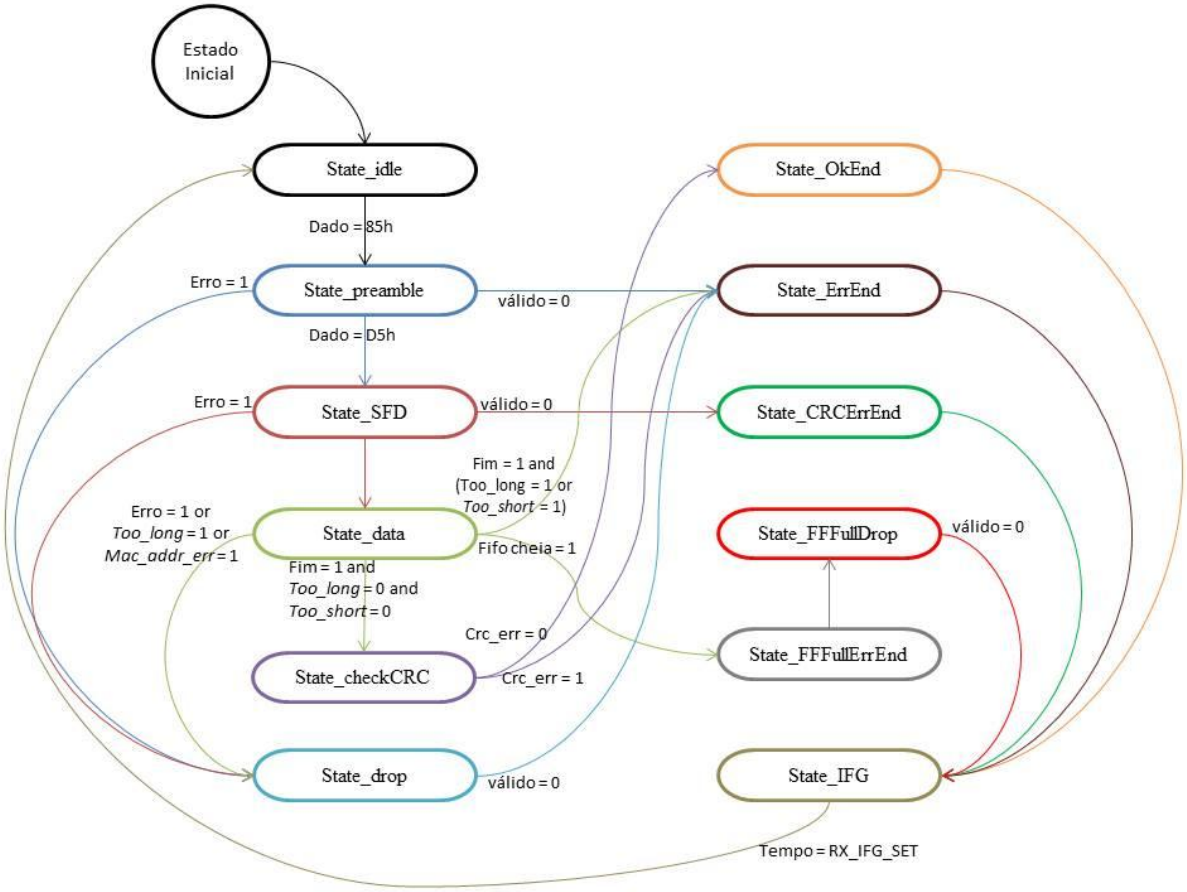


Figura 3.14, é responsável por armazenar os dados recebidos em uma fila. Ela tem o estado **State_idle** como inicial e evolui como descrito a seguir:

- **State_idle**

  Nesse estado a máquina aguarda o inicio da recepção de um pacote, esperando pelo seu preâmbulo.

- ✓ Caso o vetor de 8 bits de dados recebidos seja equivalente ao preâmbulo do pacote Ethernet (“01010101b”) e o sinal de dado válido esteja em “1”, a máquina passa para o estado **State_preamble**.
- ✓ Caso contrário, a máquina permanece do estado atual.

- **State_preamble**

  Neste estado a máquina aguarda o término da recepção do preâmbulo, verificando se nenhum erro ocorreu durante esse intervalo.

  - ✓ Caso o sinal de dado válido baixe para “0”, a máquina passa para o estado **State_ErrEnd**.
  - ✓ Caso o sinal de erro tenha valor “1”, a máquina passa para o estado **State_drop**.
  - ✓ Caso o dado recebido seja equivalente ao SFD (“11010101b”), a máquina passa para o estado **State_SFD**.
  - ✓ Caso o dado recebido permaneça com valor equivalente ao preâmbulo o estado atual é mantido.
  - ✓ Caso nenhuma das opções seja satisfeita, a máquina passa para o estado **State_drop.**

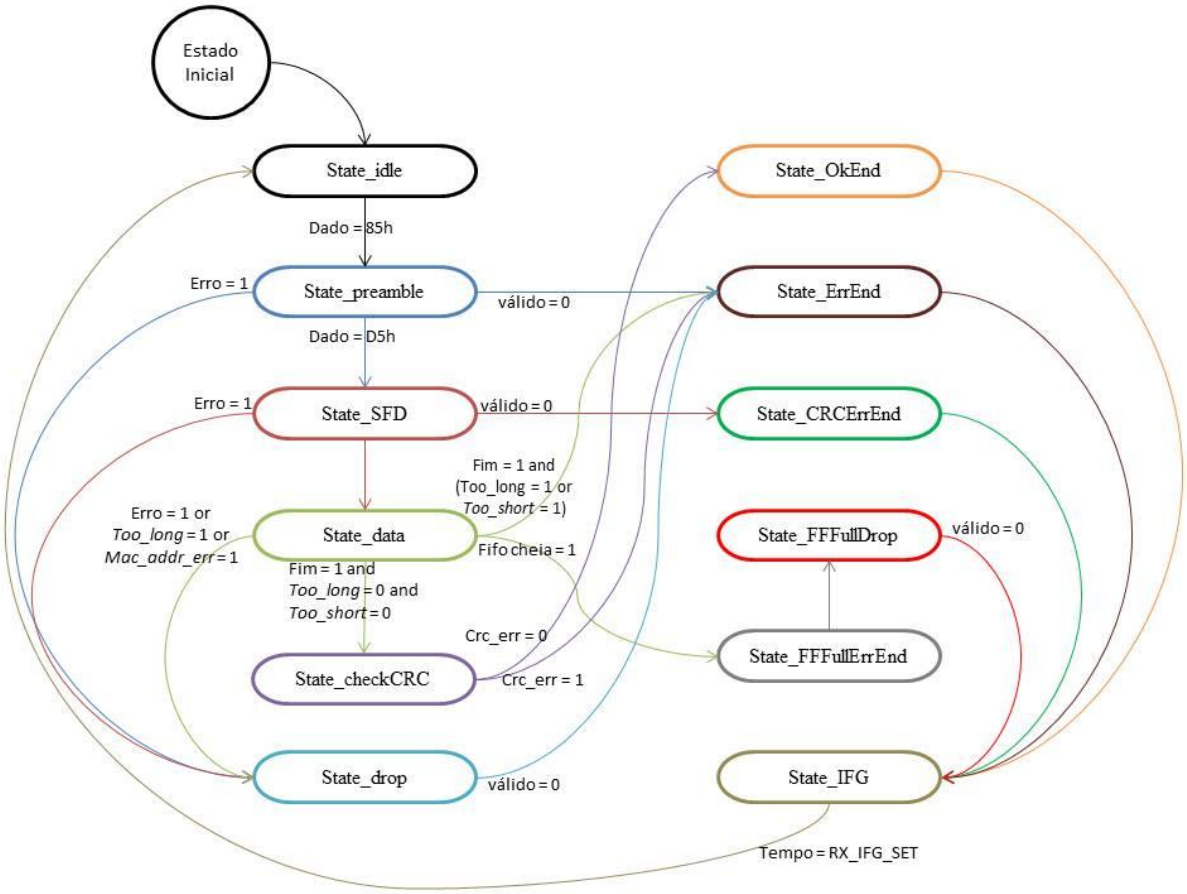


Figura 3.14 - Diagrama de transições da máquina de estados principal do módulo MAC_rx_ctrl, responsável por armazenar, em uma fila, os dados recebidos. Fonte: elaborada pelo autor.

- **State_SFD**

  Neste estado a máquina define o início da recepção de dados.

  - ✓ Caso o sinal de dado válido baixe para "0", a máquina passa para o estado **State_ErrEnd**.
  - ✓ Caso o sinal de erro tenha valor "1", a máquina passa para o estado **State_drop**.
  - ✓ Caso nenhuma das opções seja satisfeita, a máquina passa para o estado **State_data**.

- **State_data**

  Neste estado a máquina aguarda pela recepção de todos os dados do pacote, armazenando-os na *fifo* do módulo MAC_rx_FF32 e verificando a ocorrência de algum erro.

- ✓ Caso o pacote tenha sido totalmente recebido e seu tamanho esteja na faixa especificada pelos registradores RX_MAX_LENGTH e RX_MIN_LENGTH, a máquina passa para o estado **State_checkCRC**.
- ✓ Caso o pacote tenha sido totalmente recebido e seu tamanho não esteja na faixa especificada pelos registradores Rx_Hwmark e Rx_Lwmark, a máquina passa para o estado **State_Errend**.
- ✓ Caso a Fifo de recepção de dados esteja cheia, a máquina passa para o estado **State_FFFullErrEnd**.
- ✓ Caso o sinal de erro seja equivalente a "1" ou o pacote não tenha sido endereçado a este módulo ou o pacote seja grande demais, a máquina passa para o estado **State_drop**.
- ✓ Caso nenhuma das opções seja satisfeita, a máquina permanece no estado atual.

- **State_checkCRC**

  Neste estado a máquina verifica se ouve erro durante a recepção do pacote através do calculo do CRC.
  - ✓ Caso a verificação do pacote seja inválida, a máquina passa para o estado **State_CRCErrEnd**.
  - ✓ Caso contrário a máquina passa para o estado **State_OkEnd**.

- **State_drop**

  Neste estado a máquina aguarda a recepção de todo o pacote, descartando os dados recebidos.
  - ✓ Caso o sinal de dado válido esteja com o valor "0", a máquina passa para o estado **State_ErrEnd**.
  - ✓ Caso contrário, o estado atual é mantido para que o pacote seja totalmente descartado.

- **State_OkEnd**

  Neste estado a máquina define o final da recepção de pacotes.
  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_IFG**.

- **State_ErrEnd**

  Neste estado a máquina define um erro na recepção do pacote, que se deve a uma interrupção na recepção ou ao pacote exceder o tamanho mínimo ou máximo, previamente definidos.

  ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_IFG**.

- **State_CRCErrEnd**

  Neste estado a máquina define um erro na recepção do pacote, que se deve a um erro na checagem do CRC.

  ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_IFG**.

- **State_FFFullDrop**

  Neste estado a máquina aguarda a recepção de todo o pacote, descartando os dados recebidos.

  ✓ Caso o sinal de dado válido esteja com o valor "0", a máquina passa para o estado **State_IFG**.

  ✓ Caso contrário, o estado atual é mantido para que o pacote seja totalmente descartado.

- **State_FFFullErrEnd**

  Neste estado a máquina define um erro na recepção do pacote, que se deve à falta de espaço na *fifo* do módulo MAC_rx_FF32, que realiza o armazenamento dos dados.

  ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_FFFullDrop**.

- **State_IFG**

  Neste estado a máquina aguarda por um período de tempo, definido no registrador RX_IFG_SET, antes de voltar ao estado inicial para a recepção de um novo pacote. Esse período equivale ao intervalo entre a recepção de dois pacotes.

- ✓ Caso o intervalo definido no registrador RX_IFG_SET seja satisfeito, a máquina volta ao estado inicial, ou seja, o estado **State_idle**.
- ✓ Caso contrário, o estado atual é mantido até que o intervalo prescrito seja satisfeito.

Neste mesmo módulo, tem-se ainda uma máquina de estados auxiliar, a qual implementa o *hardware* para a recepção de um pacote de pausa. Este pacote representa um pedido para que a transmissão seja interrompida por certo período de tempo definido no pacote. A máquina de estados é representada pela

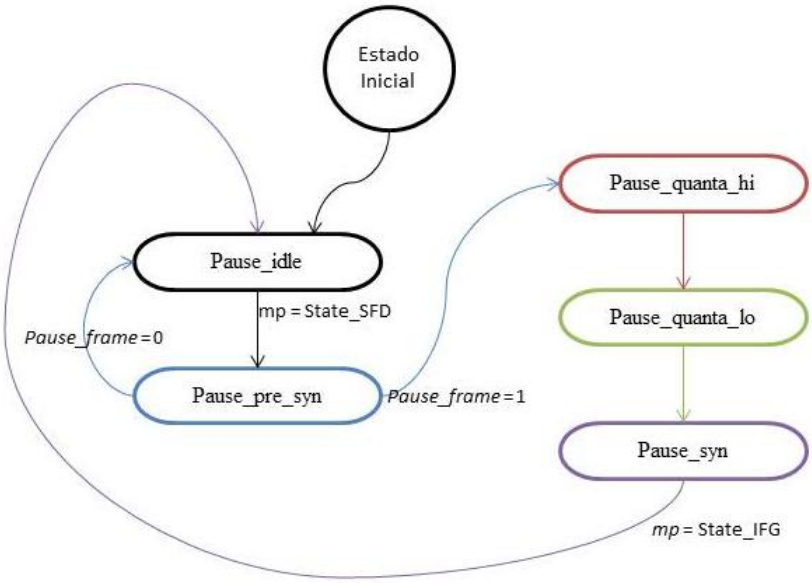


Figura 3.15, seu estado inicial é o **Pause_idle** e sua evolução é descrita a seguir.

- **Pause_idle**

  Neste estado a máquina aguarda pelo inicio da recepção de dados de um pacote.

  - ✓ Se o estado da máquina principal for o **State_SFD**, esta máquina passa para o estado **Pause_pre_syn**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **Pause_pre_syn**

  Neste estado a máquina verifica se os dados recebidos são equivalentes a um pedido de pausa.

- ✓ Se os dados recebidos são equivalentes aos dados de um pacote de pausa, esta máquina passa para o estado **Pause_quanta_hi.**
- ✓ Caso contrário, a máquina volta para o estado **Pause_idle**.

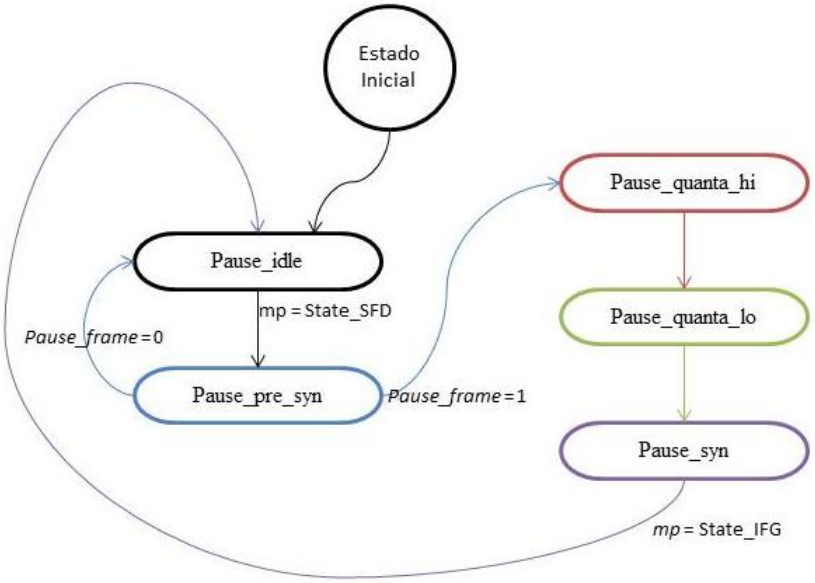


Figura 3.15 - Diagrama de transições da máquina de estados auxiliar do módulo MAC_rx_ctrl, a qual implementa o hardware necessário para tratar a recepção de um pacote de pausa. Na figura, "mp" representa o estado da máquina de estados principal do módulo. Fonte: elaborada pelo autor.

- **Pause_quanta_hi**

  Neste estado a máquina define a aquisição dos 8 bits mais significativos que representam o período de pausa requerido.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **Pause_quanta_lo**.

- **Pause_quanta_lo**

  Neste estado a máquina define a aquisição dos 8 bits menos significativos que representam o período de pausa requerido.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **Pause_syn**.

- **Pause_syn**

Neste estado a máquina aguarda pelo final da recepção do pacote.

- ✓ Se o estado da máquina principal for o **State_IFG**, esta máquina volta para o estado **Pause_idle.**
- ✓ Caso contrário, a máquina permanece no estado atual.

3.2.5.2 MAC_rx_FF32

Este módulo implementa uma fila do tipo *first in first out*, que armazena os dados recebidos e os dispõem para serem lidos por uma aplicação, que no caso deste projeto é um software rodando no processador Nios II. Os dados são recebidos de 8 em 8 bits, então este módulo aguarda a chegada de 32 bits e os armazena conjuntamente. Portanto, o barramento de transferência de dados entre o MAC e o processador é de 32 bits.

O módulo é regido por duas máquinas de estados principais, uma para escrever os dados na fila e a outra para ler os dados da fila. Na ordem de acontecimentos, os dados que chegam ao sistema são primeiramente armazenados nesta fila. O processo é controlado pela máquina de estados mostrada na

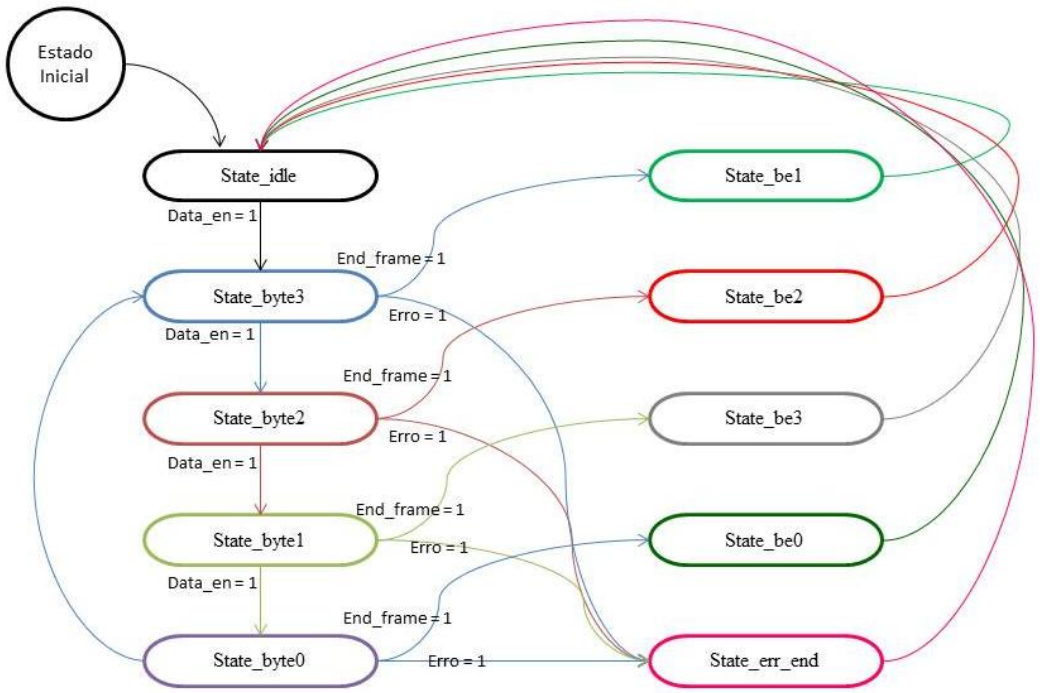


Figura 3.16, e seu estado inicial é o estado **State_idle**.

State_idle

Neste estado a máquina aguarda pelo inicio da recepção de dados do pacote.

- ✓ Se o sinal que indica o inicio da recepção de dados estiver com valor “1”, a máquina passa para o estado **State_byte3**.
- ✓ Caso contrário, a máquina permanece no estado atual.

- **State_byte3**

  Neste estado a máquina define a recepção do byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de dados sendo recebidos estiver com valor “1”, a máquina passa para o estado **State_byte2**.
  - ✓ Caso contrário, se o sinal de erro estiver com valor “1”, a máquina passa para o estado **State_err_end**.
  - ✓ Caso contrário, se o sinal que indica o final do pacote estiver com valor “1”, a máquina passa para o estado **State_be1**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **State_byte2**

  Neste estado a máquina define a recepção do segundo byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de dados sendo recebidos estiver com valor “1”, a máquina passa para o estado **State_byte1**.
  - ✓ Caso contrário, se o sinal de erro estiver com valor “1”, a máquina passa para o estado **State_err_end**.
  - ✓ Caso contrário, se o sinal que indica o final do pacote estiver com valor “1”, a máquina passa para o estado **State_be2**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

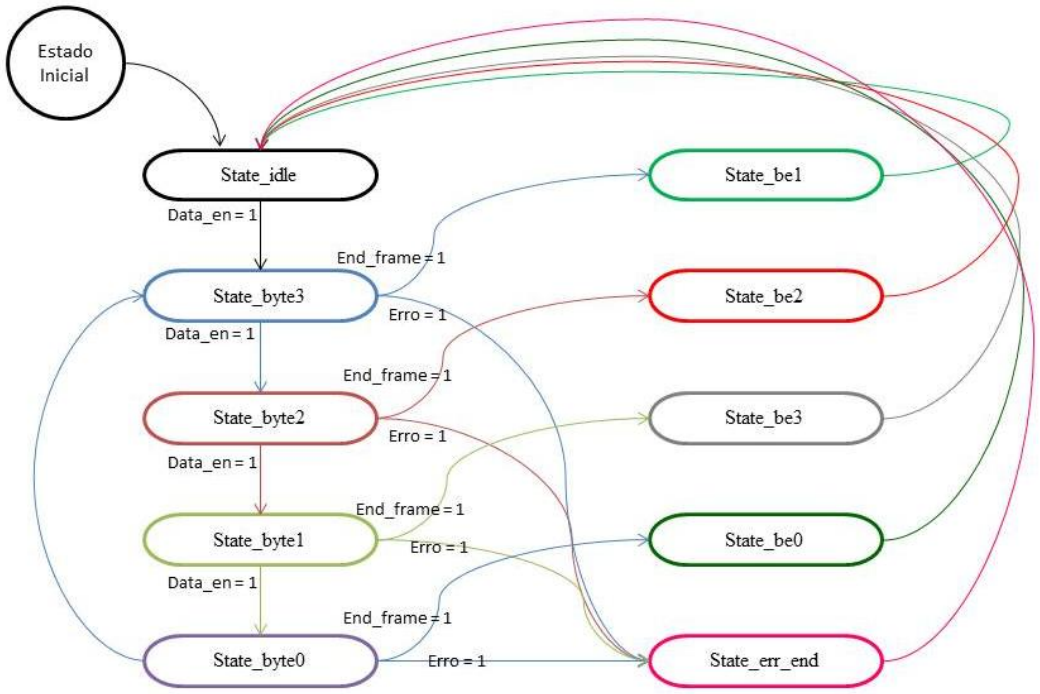


Figura 3.16 - Diagrama de transições da máquina de estados que efetua a escrita dos dados na fila do módulo MAC_rx_FF32. Fonte: elaborada pelo autor.

- **State_byte1**

  Neste estado a máquina define a recepção do terceiro byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de dados sendo recebidos estiver com valor “1”, a máquina passa para o estado **State_byte0**.
  - ✓ Caso contrário, se o sinal de erro estiver com valor “1”, a máquina passa para o estado **State_err_end**.
  - ✓ Caso contrário, se o sinal que indica o final do pacote estiver com valor “1”, a máquina passa para o estado **State_be3**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **State_byte0**

  Neste estado a máquina define a recepção do quarto byte mais significativo dos 32 bits que são agrupados no armazenamento e define o armazenamento dos 32 bits.

  - ✓ Se o sinal de dados sendo recebidos estiver com valor “1”, a máquina passa para o estado **State_byte3**.

- ✓ Caso contrário, se o sinal de erro estiver com valor “1”, a máquina passa para o estado **State_err_end**.
- ✓ Caso contrário, se o sinal que indica o final do pacote estiver com valor “1”, a máquina passa para o estado **State_be0**.
- ✓ Caso contrário, a máquina permanece no estado atual.

- **State_be1**

  Neste estado a máquina define o final da recepção do pacote e define o armazenamento do byte mais significativo, preenchendo com zeros o restante dos 32 bits.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_idle**.

- **State_be2**

  Neste estado a máquina define o final da recepção do pacote e define o armazenamento dos dois bytes mais significativos, preenchendo com zeros o restante dos 32 bits.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_idle**.

- **State_be3**

  Neste estado a máquina define o final da recepção do pacote e define o armazenamento dos 3 bytes mais significativos, preenchendo com zeros o restante dos 32 bits.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_idle**.

- **State_be0**

  Neste estado a máquina define o final da recepção do pacote e define o armazenamento dos últimos 32 bits de dados.

- ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_idle**.

- **State_err_end**

  Neste estado a máquina define um erro na recepção do pacote e o descarte dos dados armazenados.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **State_idle**.

Uma vez que a *fifo* tenha algum dado armazenado, ela pode transferi-los para o processador, quando esta operação for requisitada. Então, para o controle da leitura dos dados na fila, se tem a máquina de estados mostrada na Figura 3.17, para a qual o estado inicial é o estado **SYS_idle** e o seu funcionamento é descrito a seguir.

- **SYS_idle**

  Neste estado a máquina aguarda por um pedido vindo do processador para realizar a leitura dos dados armazenados na fila, e enviá-los para o software que está rodando no processador.

  - ✓ Se a operação de leitura de dados for requisitada e a *fifo* tenha dados disponíveis para transmissão, a máquina passa para o estado **SYS_read**.
  - ✓ Caso contrário, se a operação de leitura de dados for requisitada, mas a *fifo* não tenha dados disponíveis para transmissão, a máquina passa para o estado **FF_empty_err**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **SYS_read**

  Neste estado a máquina transmite os dados do pacote que está na fila.

  - ✓ Se o final do pacote que está sendo transmitido foi alcançado, a máquina passa para o estado **SYS_wait_end**.

- ✓ Caso contrário, se um sinal de pausa na transferência for enviado, a máquina passa para o estado **SYS_pause**.
- ✓ Caso contrário, se a fila estiver vazia, a máquina passa para estado **FF_empty_err**.
- ✓ Caso contrário, a máquina permanece no estado atual.

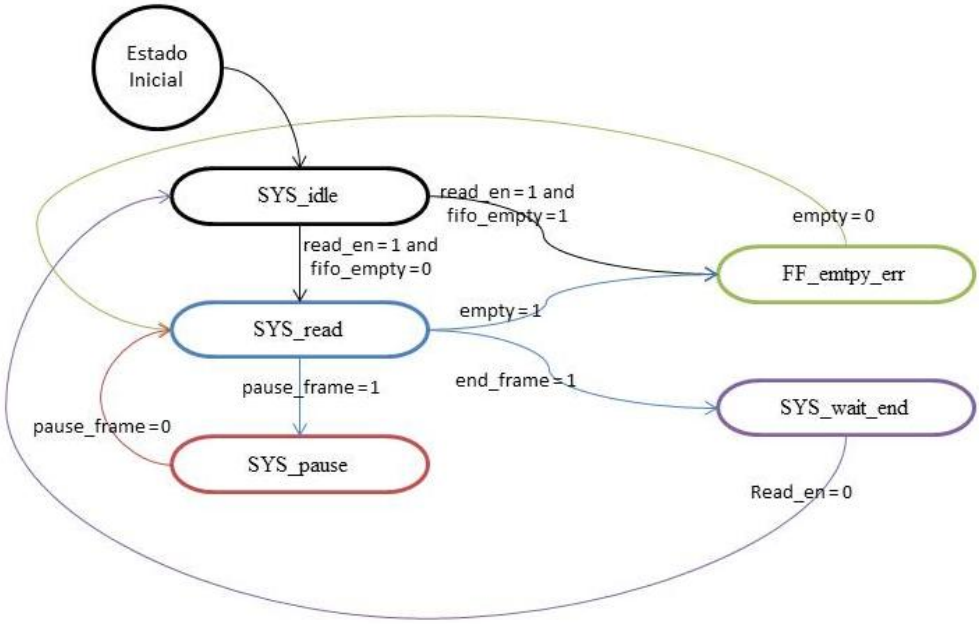


Figura 3.17 - Diagrama de transições da máquina de estados que efetua a leitura dos dados na fila do módulo MAC_rx_FF32. Fonte: elaborada pelo autor.

- **SYS_pause**

  Neste estado a máquina permanece pausada até que um sinal, para que a transmissão continue, seja enviado.

  - ✓ Se um sinal para continuar a transmissão de dados for recebido, a máquina volta para o estado **SYS_read**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **FF_empty_err**

  Neste estado a máquina define um erro devido a falta de dados na fila e aguarda que a fila não esteja mais vazia.

  - ✓ Se a fila não estiver mais vazia, a máquina volta para o estado **SYS_read**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **SYS_wait_end**

  Neste estado a máquina espera por um sinal que diz se o software rodando no processador terminou de fazer a leitura dos dados desse pacote.

  - ✓ Se a operação para leitura de dados não estiver mais sendo requisitada, a máquina volta para o estado **SYS_idle**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

3.2.5.3 MAC_rx_add_chk

O módulo MAC_rx_add_chk verifica o endereço de destino do pacote recebido. Como foi visto na seção 2.1.3, os primeiros seis bytes do pacote Ethernet representam o endereço físico de destino do pacote. Sendo assim, ao receber tal parte do frame, o módulo MAC_rx_ctrl envia um sinal, juntamente com os dados, para o módulo MAC_rx_add_chk. Então, caso a checagem do endereço esteja habilitada, este último módulo compara o endereço recebido com o valor guardado no registrador MAC_add_prom_data e retorna um sinal com valor “0”, caso os valores sejam compatíveis, ou “1”, caso contrário. Se os valores

não forem compatíveis, ou seja, o sinal retorna tenha valor “1”, o frame é descartado pelo módulo MAC_rx_ctrl.

O sistema também pode receber pacotes *broadcast* juntamente com os pacotes endereçados para o sistema diretamente. Sendo assim, ao comparar o endereço de destino do pacote, este módulo também compara o valor recebido com o valor do endereçamento *broadcast* (“FFFFFFh”). Então, sendo o pacote endereçado para o sistema ou para todos os dispositivos na rede, ele será armazenado, caso contrário, será descartado.

3.2.5.4 Broadcast_filter

Este módulo filtra o número de pacotes *broadcast* recebidos pelo sistema. O filtro foi projetado como um balde que se enche à medida que chegam pacotes *broadcast* pela rede. A quantidade de bytes de pacotes *broadcast* que o balde pode armazenar é definida pelo registrador broadcast_bucket_depth e, além disso, o balde é esvaziado ao se passarem certo número de ciclos de *clocks* definido pelo registrador broadcast_bucket_interval.

Então quando o balde se enche, o que é equivalente a dizer que chegaram broadcast_bucket_depth bytes, todos os pacotes que chegarem depois são descartados. Porém, o balde é esvaziado toda vez que decorrem broadcast_bucket_interval ciclos de clock. Desta forma, a razão entre broadcast_bucket_depth e broadcast_bucket_interval determina o número máximo de bytes permitidos em um segundo. Por exemplo, se a razão é 0,1, o fluxo de pacotes broadcast é limitado a 10Mbps quando a velocidade de operação é de 100Mbps.

Então, se o filtro está habilitado e o balde cheio, ele envia um sinal com valor “1”, neste caso os dados subsequentes são descartados, mas se o filtro está habilitado e o balde ainda não se encheu o valor do sinal enviado é “0”.

3.2.5.5 CRC_check

Este módulo executa a checagem do CRC do pacote recebido. O algoritmo descrito na seção 2.1.3.6, que verifica se houve erros durante a transmissão do pacote, é executado se o valor do registrador CRC_chk_en for “1”, caso contrário, se o valor do registrador for “0”, a

verificação de erros não é executado e este módulo não executa nenhuma operação sobre o sistema. Caso um erro no pacote seja identificado, um sinal é enviado ao módulo MAC_rx_ctrl que o faz descartar o pacote, caso contrário, os dados do pacote são armazenados na *fifo* até que sua leitura seja realizada.

3.2.6 MAC_tx

Este módulo é semelhante ao MAC_rx, porém ele é responsável pela parte de transmissão de pacotes. Isso envolve o cálculo do CRC, o acoplamento do endereço físico ao pacote a ser transmitido, entre outros. Ele é composto por seis submódulos, mostrados na Figura 3.18, dos quais cada um é responsável por uma tarefa distinta, as quais são descritas nas seções seguintes.

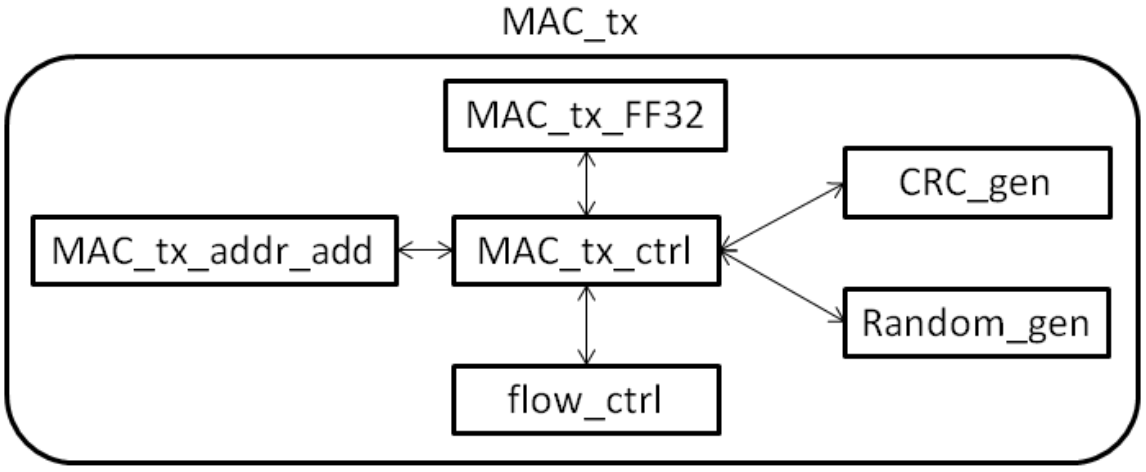


Figura 3.18 - Diagrama de blocos do módulo MAC_tx, o qual é responsável pela recepção de pacotes do sistema. Fonte: elaborada pelo autor.

3.2.6.1 MAC_tx_ctrl

Esse módulo realiza o controle da transmissão de pacotes e dispõe de duas máquinas de estados para gerenciar as diferentes tarefas, uma máquina de estado principal e outra auxiliar, semelhantemente ao módulo MAC_rx_ctrl. A máquina principal é mostrada na Figura 3.19, ela é responsável por controlar o envio de dados, quando estes estiverem prontos na fila. Seu estado inicial é o estado **State_Defer** e sua evolução é descrita a seguir:

- **StateDefer**

  Neste estado a máquina verifica se meio de transmissão está ocioso.

  - ✓ Se o módulo estiver funcionando em modo *fullduplex* ou se ele estiver em modo *halfduplex* e o sinal de meio ocioso estiver com valor "0", a máquina passa para o estado **StateIFG**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateIFG**

  Neste estado a máquina aguarda por um período de tempo defino no registrador IFGset, que equivale ao período de tempo que se deve aguardar entre a transmissão de dois pacotes consecutivos.

  - ✓ Se o módulo estiver em *halfduplex* e o meio estiver ocupado, a máquina volta para o estado **StateDefer.**
  - ✓ Caso contrário, se o sistema estiver apto a enviar dados e tenha decorrido, no estado atual, o tempo determinado pelo registrador IFGset, a máquina passa para o estado **StateIdle**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

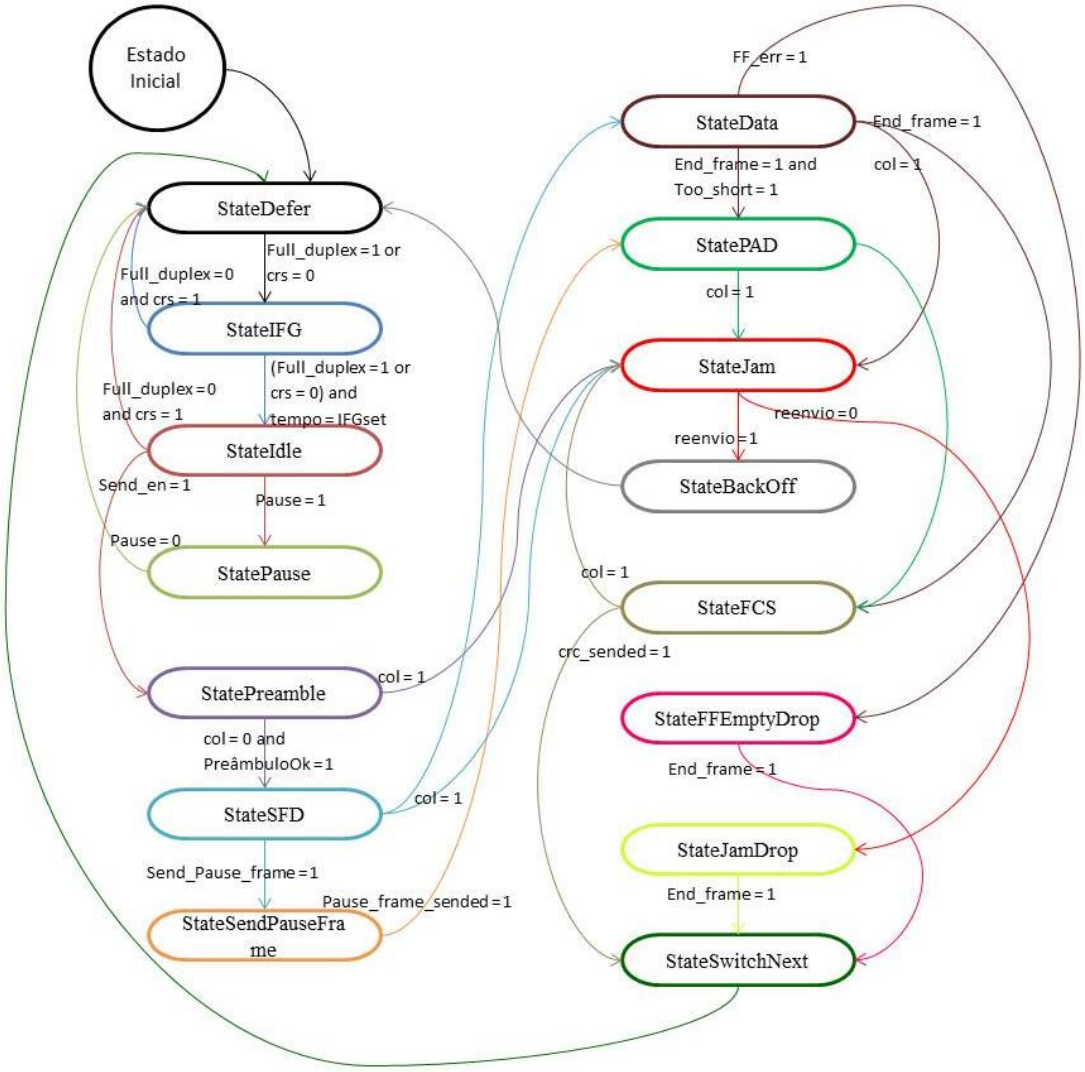


Figura 3.19 - Diagrama de transições da máquina de estados principal do módulo MAC_tx_ctrl, ela é responsável pela transmissão dos dados que chegam até a fila. A sigla crs índica se o meio está ocioso quando seu valor é zero e a sigla col indica se houve uma colisão quando seu valor é um. Fonte: elaborada pelo autor.

- **StateIdle**

  Neste estado a máquina aguarda por uma operação de transmissão.

  - ✓ Se o módulo estiver ocupado quando operando em *halfduplex*, a máquina volta para o estado **StateDefer**.
  - ✓ Caso contrário, se o sinal para que o sistema pause o envio de pacotes estiver com valor “1”, a máquina passa para o estado **StatePause**.

- ✓ Caso contrário, se o sistema estiver apto a enviar dados e tenha pacotes a enviar, seja esses pacotes de dados ou de pausa, a máquina passa para o estado **StatePreamble**.
- ✓ Caso contrário, a máquina permanece no estado atual.

- **StatePause**

  Neste estado a máquina permanece pausada por certo período de tempo.
  - ✓ Se decorrido o tempo de pausa, a máquina volta para o estado **StateDefer**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StatePreamble**

  Neste estado a máquina inicia a transmissão do pacote, enviando o preâmbulo.
  - ✓ Se o sistema estiver operando em modo *halfduplex* e for detectada uma colisão, a máquina passa para o estado **StateJam**.
  - ✓ Caso contrário, se o sistema estiver apto a enviar pacotes e o tempo de seis ciclos de *clock* tenham se passados no estado **StatePreamble**, a máquina passa para o estado **StateSFD**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateSFD**

  Neste estado a máquina transmite o delimitador de início do pacote e define o inicio da transmissão de dados.
  - ✓ Se o sistema estiver operando em modo *halfduplex* e for detectada uma colisão, a máquina passa para o estado **StateJam**.
  - ✓ Caso contrário, se o módulo MAC_ctrl receber um sinal para envio de pedido de pausa, a máquina passa para o estado **StateSendPauseFrame**.
  - ✓ Caso contrário, a máquina passa para o estado **StateData**.

- **StateSendPauseFrame**

  Neste estado a máquina define a transmissão de um pacote de pausa.
  - ✓ Se o frame de pausa tenha sido totalmente enviado, a máquina passa para o estado **StatePAD**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateData**

  Neste estado a máquina define a transmissão dos dados do pacote.

  - ✓ Se o sistema estiver operando em modo *halfduplex* e for detectada uma colisão, a máquina passa para o estado **StateJam**.
  - ✓ Caso contrário, se o sinal de erro vindo da *fifo* tenha valor "1", a máquina passa para o estado **StateFFEmptyDrop**.
  - ✓ Caso contrário, se os dados tenham sido totalmente transmitidos e o tamanho mínimo do pacote tenha sido alcançado, a máquina passa para o estado **StateFCS**.
  - ✓ Caso contrário, se os dados tenham sido totalmente transmitidos, a máquina passa para o estado **StatePAD**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StatePAD**

  Neste estado a máquina completa com zeros a seção de dados do pacote, até que este alcance o tamanho mínimo.

  - ✓ Se o sistema estiver operando em modo *halfduplex* e for detectada uma colisão, a máquina passa para o estado **StateJam**.
  - ✓ Caso contrário, se o tempo necessário para que o pacote seja completado com zeros tenha decorrido, a máquina passa para o estado **StateFCS**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateJam**

  Neste estado a máquina define se o pacote que sofreu colisão será descartado ou retransmitido.

  - ✓ Se o número máximo de tentativas de envio do pacote definido pelo registrador MaxRetry ainda não tenha sido alcançado e o tempo para que o módulo Random_gen gere o número aleatório tenha decorrido, a máquina passa para o estado **StateBackOff**.
  - ✓ Caso contrário, se o número máximo de tentativas de envio da mensagem tenha sido alcançado, a máquina passa para o estado **StateJamDrop**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateBackOff**

  Neste estado a máquina aguarda por um tempo aleatório, definido no módulo Random_gen, antes de retransmitir o pacote.

  - ✓ Se o tempo aleatório de espera tenha sido alcançado, a máquina volta para o estado **StateDefer**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateFCS**

  Neste estado a máquina define a transmissão dos 32 bits do CRC.

  - ✓ Se o sistema estiver operando em modo *halfduplex* e for detectada uma colisão, a máquina passa para o estado **StateJam**.
  - ✓ Caso contrário, se os 4 bytes respectivos ao CRC tenham sidos enviados no final da mensagem, a máquina passa para o estado **StateSwitchNext**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateFFEmptyDrop**

  Neste estado a máquina descarta os dados do pacote que está sendo recebido da *fifo*, até que o pacote, no qual o erro de fila vazia ocorreu, chegue ao fim.

  - ✓ Se o final do pacote foi alcançado, a máquina passa para o estado **StateSwitchNext**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateJamDrop**

  Neste estado a máquina descarta os dados do pacote que está sendo recebido da fifo, até que o pacote, para o qual o número de tentativas de envio chegou ao limite, chegue ao fim.

  - ✓ Se o final do pacote tenha sido alcançado, a máquina passa para o estado **StateSwitchNext**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateSwitchNext**

  Neste estado a máquina define o término da transmissão do pacote.

- ✓ Independente da situação, ao chegar a este estado a máquina volta automaticamente para o estado **State_Defer**.

3.2.6.2 MAC_tx_FF32

Este módulo implementa uma fila do tipo *first in first out*, semelhantemente ao módulo MAC_rx_FF32, porém, ela armazena os dados recebidos do processador e que devem ser enviados através da rede Ethernet. Os dados são recebidos do processador através de um barramento de 32 bits e devem ser particionados em blocos de 8 bits para que sejam transmitidos.

O módulo é regido por duas máquinas de estados principais, uma para escrever os dados na fila e a outra para ler os dados da fila. Na ordem de acontecimentos, os dados que chegam ao sistema são primeiramente armazenados nesta fila, este processo é controlado pela máquina de estados mostrada na Figura 3.20, para a qual o estado inicial é o estado **SYS_idle**.

- **SYS_idle**

  Neste estado a máquina aguarda por um evento para começar a receber os dados que serão guardados na *fifo*.
  - ✓ Se a fila não estiver cheia e este for o primeiro bloco do pacote a ser armazenado, a máquina passa para o estado **SYS_SOP.**
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **SYS_SOP**

  Neste estado a máquina define o inicio do armazenamento dos dados do pacote na fila.
  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **SYS_MOP**.

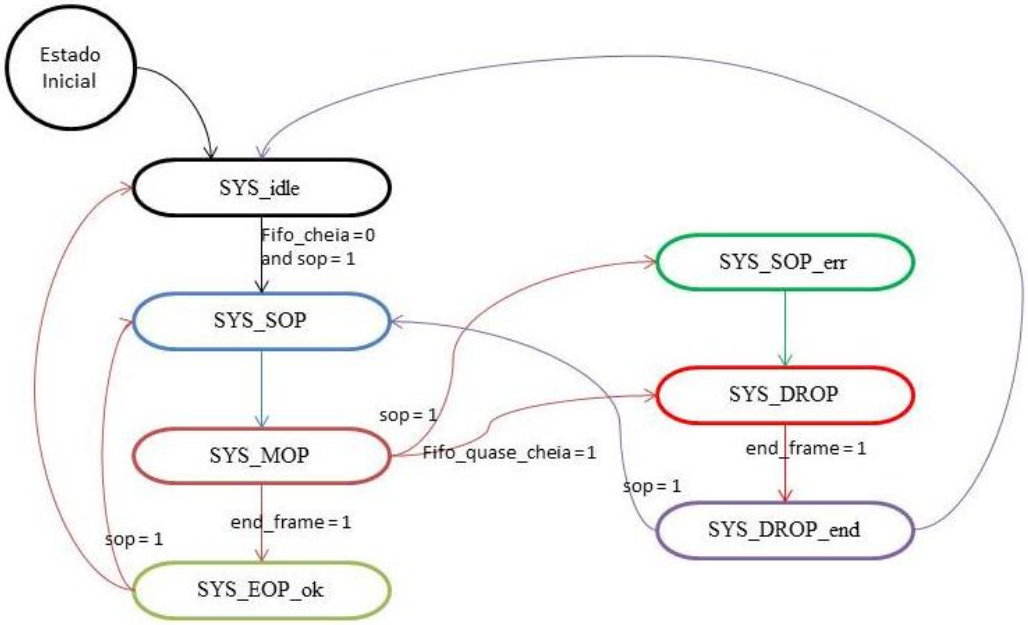


Figura 3.20 - Diagrama de transições da máquina de estados que efetua a escrita dos dados na fila do módulo MAC_tx_FF32. Fonte: elaborada pelo autor.

- **SYS_MOP**

  Neste estado a máquina define o armazenamento dos dados do pacote na fila.

  - ✓ Se a fila estiver quase cheia, a máquina passa para o estado **SYS_DROP**.
  - ✓ Caso contrário, se o dado a ser armazenado ainda pertencer ao primeiro bloco do pacote, a máquina passa para o estado **SYS_SOP_err**.
  - ✓ Caso contrário, se o dado a ser armazenado pertencer ao último bloco do pacote, a máquina passa para o estado **SYS_EOP_ok**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **SYS_EOP_ok**

  Neste estado a máquina define o término do armazenamento dados do pacote na fila.

  - ✓ Se houver dados a serem armazenados e o bloco a ser transmitido corresponder ao primeiro do pacote, a máquina volta para o estado **SYS_SOP**.
  - ✓ Caso contrário, a máquina volta para o estado **SYS_idle**.

- **SYS_SOP_err**

  Neste estado a máquina define um erro durante o inicio do armazenamento dos dados na fila.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **SYS_DROP**.

- **SYS_DROP**

  Neste estado a máquina descarta os dados do pacote.

  - ✓ Se o bloco a ser descartado corresponder ao ultimo bloco do pacote, a máquina passa para o estado **SYS_DROP_end**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **SYS_DROP_end**

  Neste estado a máquina define o final do descarte dos dados do pacote.

  - ✓ Se houver dados a serem armazenados e o bloco a ser transmitido corresponder ao primeiro do pacote, a máquina volta para o estado **SYS_SOP**.
  - ✓ Caso contrário, a máquina volta para o estado **SYS_idle**.

Uma vez que a *fifo* tenha um pacote inteiro armazenado ou o nível mínimo de dados tenha sido alcançado, a transmissão pela rede Ethernet é iniciada. Essa transmissão é controlada pela máquina de estados mostrada na Figura 3.21, para a qual o seu estado inicial é o **SYS_idle** e o seu funcionamento está descrito a seguir.

- **MAC_idle**

  Neste estado a máquina aguarda por um evento para dar inicio a transmissão dos dados armazenados na fila.

  - ✓ Se o sinal para iniciar a transmissão dos dados estiver em “1”, mas a fila estiver vazia, a máquina para o estado **MAC_FF_Err.**

- ✓ Caso contrário, se o sinal de transmissão de dados estiver em “1”, a máquina passa para o estado **MAC_byte3**.
- ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_byte3**

  Neste estado a máquina define a transmissão do byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de retransmissão de pacotes estiver com valor “1”, a máquina passa para o estado **MAC_retry**.
  - ✓ Caso contrário, se o bloco a ser transmitido representar o último bloco do pacote, a máquina passa para o estado **MAC_wait_finish**.
  - ✓ Caso contrário, se o bloco de oito bits atual for o último dado válido do bloco de 32 bits, a máquina permanece no estado **MAC_byte3**.
  - ✓ Caso contrário, se o sinal de transmissão de dados estiver em “1”, a máquina passa para o estado **MAC_byte2**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

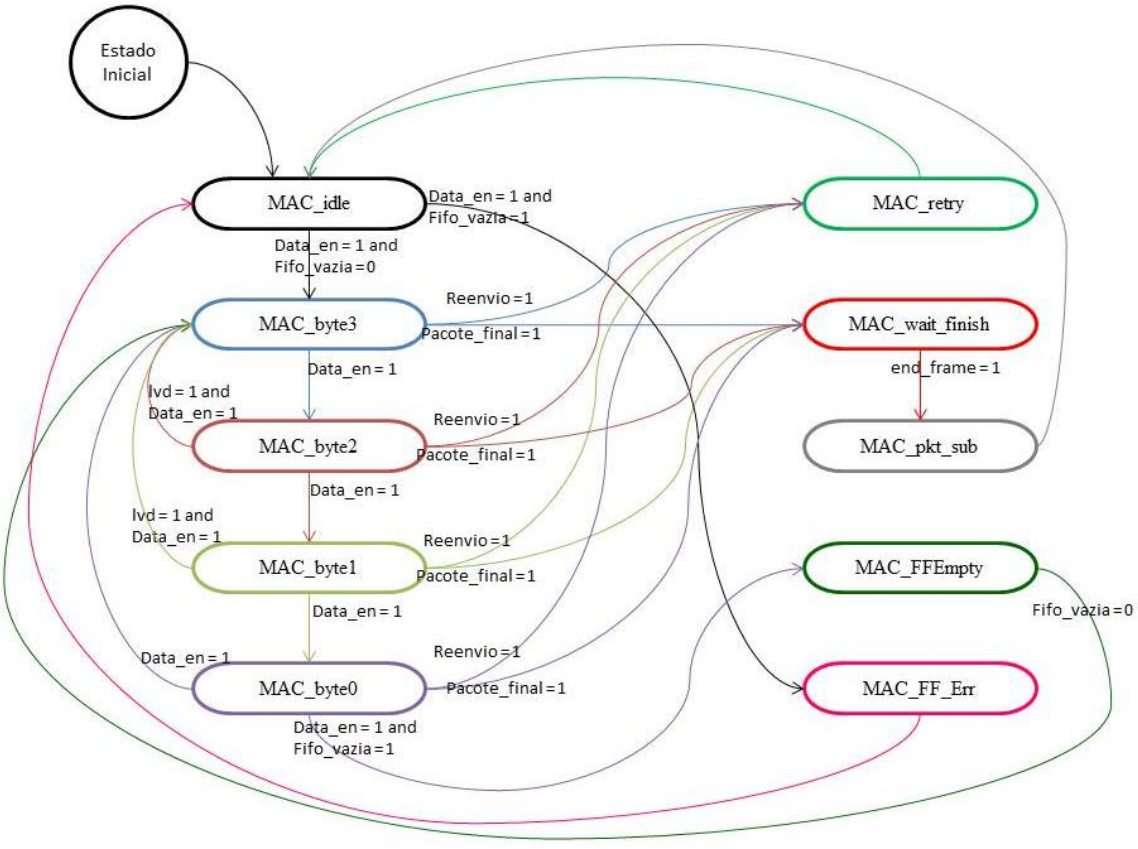


Figura 3.21 - Diagrama de transições da máquina de estados que efetua a leitura dos dados da fila do módulo MAC_tx_FF32. Fonte: elaborada pelo autor.

- **MAC_byte2**

  Neste estado a máquina define a transmissão do segundo byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de retransmissão de pacotes estiver com valor "1", a máquina passa para o estado **MAC_retry**.
  - ✓ Caso contrário, se o bloco a ser transmitido representar o último bloco do pacote, a máquina passa para o estado **MAC_wait_finish**.
  - ✓ Caso contrário, se o bloco de 8 bits atual for o último dado válido do bloco de 32 bits, a máquina passa para o estado **MAC_byte3**.
  - ✓ Caso contrário, se o sinal da transmissão de dados estiver em "1", a máquina passa para o estado **MAC_byte1**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_byte1**

  Neste estado a máquina define a transmissão do terceiro byte mais significativo dos 32 bits que são agrupados no armazenamento.

  - ✓ Se o sinal de retransmissão de pacotes estiver com valor "1", a máquina passa para o estado **MAC_retry**.
  - ✓ Caso contrário, se o bloco a ser transmitido representar o último bloco do pacote, a máquina passa para o estado **MAC_wait_finish**.
  - ✓ Caso contrário, se o bloco de 8 bits atual for o último dado válido do bloco de 32 bits, a máquina passa para o estado **MAC_byte3**.
  - ✓ Caso contrário, se o sinal de transmissão de dados estiver em "1", a máquina passa para o estado **MAC_byte0**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_byte0**

  Neste estado a máquina define a transmissão do byte menos significativo dos 32 bits que são agrupados no armazenamento.

- ✓ Se o sinal de retransmissão de pacotes estiver com valor “1”, a máquina passa para o estado **MAC_retry**.
- ✓ Caso contrário, se o bloco a ser transmitido representar o último bloco do pacote, a máquina passa para o estado **MAC_wait_finish**.
- ✓ Caso contrário, se o sinal de transmissão de dados estiver em “1” e a fila estiver vazia, a máquina passa para o estado **MAC_FFEmpty**.
- ✓ Caso contrário, se o sinal de transmissão de dados estiver em “1”, a máquina passa para o estado **MAC_byte3**.
- ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_retry**

  Neste estado a máquina define a retransmissão dos dados.
  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **MAC_idle**.

- **MAC_wait_finish**

  Neste estado a máquina aguarda o término da transmissão do pacote.
  - ✓ Se o sinal que indica o final da transmissão do pacote tenha valor “1”, a máquina passa para o estado **MAC_pkt_sub**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_pkt_sub**

  Neste estado a máquina define o final da transmissão do pacote, informando ao sistema que um pacote foi retirado da fila.
  - ✓ Independente da situação, ao chegar a este estado a máquina passa automaticamente para o estado **MAC_idle**.

- **MAC_FFEmpty**

  Neste estado a máquina aguarda até que a fila não esteja mais vazia, para prosseguir com a transmissão.
  - ✓ Se a fila não estiver vazia, a máquina volta para o estado **MAC_byte3**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **MAC_FF_Err**

  Neste estado a máquina define um erro no inicio da leitura.

  - ✓ Independente da situação, ao chegar a este estado a máquina passa, automaticamente, para o estado **MAC_idle**.

3.2.6.3 MAC_tx_addr_add

O módulo MAC_tx_addr_add adiciona o endereço físico do sistema ao pacote que será transmitido. Como foi visto na seção 2.1.3, os primeiros 6 bytes do pacote Ethernet representam o endereço físico do remetente do pacote. Sendo assim, ao enviar tal parte do *frame*, o módulo MAC_rx_ctrl envia um sinal, juntamente com os dados para o módulo MAC_tx_addr_add, que então, caso a acoplagem do endereço físico esteja habilitada, adiciona o endereço físico definido no registrador **MAC_add_prom_data** ao pacote. Tal registrador armazena o endereço de *hardware* do módulo.

3.2.6.4 MAC_flow_ctrl

Este módulo é responsável por realizar o controle de pausas do sistema. Através dele pode ser realizada uma pausa na transmissão de dados do módulo, ou pode ser enviado um *frame* Ethernet contendo um pedido de pausa ao destinatário. O primeiro caso depende do registrador tx_pause_en, quando o valor deste registrador é “1”, o módulo enviará um pedido de pausa com período de um *quanta*, se o período de pausa desejado for maior, o registrador deve ser mantido em “1” pelo período desejado. O período de pausa é definido em unidades de *quanta*, no qual para cada unidade o tempo decorrido é igual ao tempo de envio de 512 bits.

Já para o segundo caso, o envio de um pacote de pausa é controlado por quatro registradores. O registrador pause_framse_send_en define se o transmissor lógico está habilitado para enviar um *frame* de pausa, o registrador pause_quanta_set define o período de tempo a ser requisitado pelo *frame* de pausa que será enviado. Tem-se ainda o registrador

xon_cpu, para o qual a borda de subida do sinal define o começo da transmissão de um *frame* de pausa com o valor de *quanta* igual a pause_quanta_set, e o registrador xoff_cpu, para o qual a borda de subida do sinal é usado para começar a transmitir um *frame* de pausa com quanta igual a zero, o que equivale a um pedido para que o controlador de Ethernet remoto interrompa o estado de pausa.

A Figura 3.22 ilustra como esses registradores funcionam, onde os valores de xoff_gen e xon_gen representam se houve uma borda de subida dos registradores xoff_cpu e xon_cpu, respectivamente.

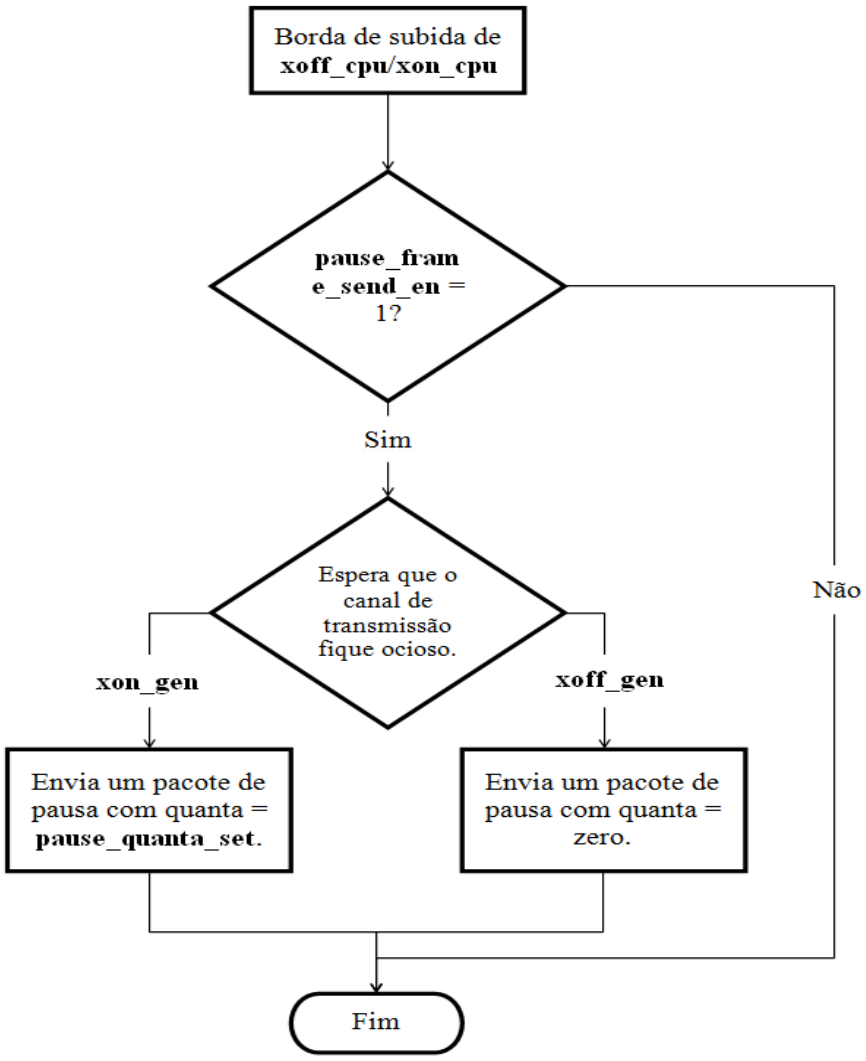


Figura 3.22 - Diagrama que exemplifica os passos necessários para o envio de um pacote de pausa.
Fonte: elaborada pelo autor.

3.2.6.5 CRC_gen

Assim como no módulo CRC_check, este módulo executa o algoritmo descrito na seção 2.1.3.6 para calcular o valor do CRC, mas deste caso o valor é acoplado ao final do *frame* a ser enviado. O módulo calcula o valor do CRC durante a transmissão do pacote e ao final desta, acopla os 32 bits.

3.2.6.6 Random_gen

Este módulo define o intervalo de *backoff* que o sistema deverá esperar no caso da ocorrência de uma colisão. Ele contém um gerador de números pseudorrandômicos que é alterado a cada ciclo de *clock*, como mostrado na Figura 3.23, onde o vetor superior na figura representa o valor antes da alteração e o vetor inferior representa o valor depois da alteração. É possível ver que o novo valor é gerado através de uma rotação para a esquerda, seguido de um ou exclusivo entre o bit mais significativo e o terceiro bit menos significativo do vetor antigo, o bit gerado nesta operação é colocado no bit menos significativo do novo vetor, como mostrado na figura citada. O número aleatório gerado deve estar dentro de certo intervalo dependendo da sequência de colisões registrada, ou seja, quando ocorrer à primeira colisão, apenas o bit menos significativo desse número será usado como tempo de *backoff*, se a colisão ocorrer pela segunda vez no mesmo pacote, será utilizado os dois bits menos significativos e assim sucessivamente, até que o número de tentativas de envio do pacote seja atingida, neste caso o pacote é descartado. O número gerado pela lógica descrita acima define o intervalo de tempo que o sistema permanecerá em modo de espera antes de tentar retransmitir o pacote colidido.

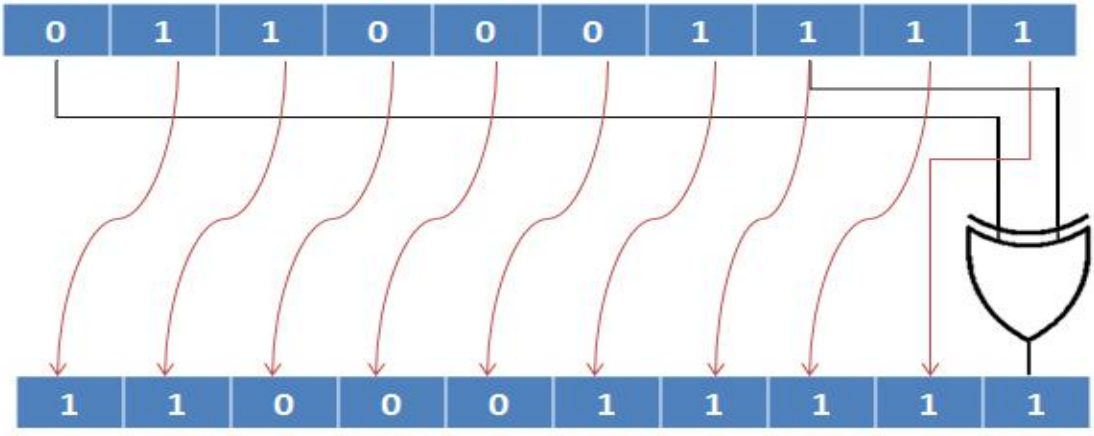


Figura 3.23 - Representação gráfica do algoritmo responsável por gerar números aleatórios. Note que a figura representa uma transição que ocorre a cada ciclo de clock. Fonte: elaborada pelo autor.

3.2.7 Reg_int

O módulo Reg_int é responsável por criar e inicializar registradores capazes de configurar o módulo MAC_top, além de disponibilizar um barramento especifico para realização de escrita e leitura nesses registradores. Cada um desses registradores tem seu endereço específico com tamanho de 6 bits e um valor padrão com tamanho de 16 bits. A lista de registradores disponíveis é mostrada na Tabela 3.2.

Tabela 3.2 - Tabela com todos os registradores de configuração do sistema, disponibilizados pelo módulo Reg_int.

Nome	Operações permitidas	Endereço (6 bits)	Valor padrão (16 bits)
Tx_Hwmark	R/W	(000)d	(001e)h
Tx_Lwmark	R/W	(001)d	(0019)h
pause_frame_send_en	R/W	(002)d	(0001)h
pause_quanta_set	R/W	(003)d	(0000)h
IFGset	R/W	(004)d	(0012)h
FullDuplex	R	(005)d	-
MaxRetry	R/W	(006)d	(0002)h
MAC_tx_add_en	R/W	(007)d	(0001)h
MAC_tx_add_prom_data_h1	R/W	(008)d	(0012)h
MAC_tx_add_prom_data_h2	R/W	(009)d	(3456)h
MAC_tx_add_prom_data_h3	R/W	(010)d	(7890)h
tx_pause_en	R/W	(011)d	(0000)h
xoff_cpu	R/W	(012)d	(0000)h
xon_cpu	R/W	(013)d	(0000)h
MAC_rx_add_chk_en	R/W	(014)d	(0001)h
MAC_rx_add_prom_data_h1	R/W	(015)d	(0012)h
MAC_rx_add_prom_data_h2	R/W	(016)d	(3456)h
MAC_rx_add_prom_data_h3	R/W	(017)d	(7890)h
broadcast_filter_en	R/W	(018)d	(0001)h
broadcast_bucket_depth	R/W	(019)d	(0080)h
broadcast_bucket_interval	R/W	(020)d	(0080)h
RX_APPEND_CRC	R/W	(021)d	(0000)h
Rx_Hwmark	R/W	(022)d	(001A)h
Rx_Lwmark	R/W	(023)d	(0010)h
CRC_chk_en	R/W	(024)d	(0001)h
RX_IFG_SET	R/W	(025)d	(0012)h
RX_MAX_LENGTH	R/W	(026)d	(0610)h
RX_MIN_LENGTH	R/W	(027)d	(002A)h

Line_loop_en	R/W	(028)d	(0000)h
Speed	R	(029)d	-
Mac_Rx_set	R/W	(030)d	(0000)h
Mac_Tx_set	R/W	(031)d	(0000)h

Fonte: elaborada pelo autor.

- Nível de água

✓ **Tx_Hwmark** e **Tx_Lwmark**

Estes dois registradores são usados para definir o nível de "água" máximo e mínimo da *fifo* de transmissão. Quando o número de bits armazenados na *fifo* de transmissão chega ao nível mínimo, o módulo de transmissão começará a ler os dados da *fifo* e enviá-los para o chip físico através da interface em operação (GMII, RGMII, MII ou RMII). Adicionalmente, os níveis de "água" máximos e mínimos estão relacionados com o sinal Tx_mac_wr, dessa forma, quando a *fifo* de transmissão atinge o nível máximo, Tx_mac_wr será ajustado para "0" de modo a avisar à aplicação do usuário que ela deve parar de transmitir o pacote. Já quando a *fifo* de transmissão atinge o nível mínimo, o sinal Tx_mac_wr é ajustado para "1", informando a aplicação que ela pode continuar com a transmissão do pacote.

✓ **Rx_Hwmark** e **Rx_Lwmark**

Os registradores **Rx_Hwmark** e **Rx_Lwmark** são usados para configurar o nível de "água" da *fifo* de recepção. Quando a *fifo* recebe um pacote completo ou os dados do pacote armazenados na *fifo* atingem o nível de "água" alto, o sinal Rx_mac_ra é ajustado para "1" e quando a *fifo* atinge o nível baixo o sinal é ajustado para "0". Quando há apenas um pacote completo recebido do chip físico, o sinal Rx_mac_ra será ajustado para "zero" apenas quando todo o pacote for lido da *fifo*.

- Controle de fluxo

✓ **pause_frame_send_en**, **pause_quanta_set**, **xoff_cpu** e **xon_cpu**

O registrador **pause_frame_send_en** define se o sistema pode enviar pedidos de pausa, quando seu valor é “1” ele está habilitado a executar esse serviço, caso contrário, não.

O registrador **pause_quanta_set** define o numero de *quantas* que será enviado no pedido de pausa. Já o registrador **xon_cpu** é utilizado para iniciar a transmissão de um pedido de pausa com *quanta* igual a **pause_quanta_set** e o registrador **xoff_cpu** é utilizado para iniciar a transmissão de um pedido de pausa com *quanta* igual a zero.

- ✓ **tx_pause_en**

Quando este registrador tem valor lógico “1”, o módulo responderá a um pedido de pausa na transmissão. A máquina de estado de transmissão entrará em um estado de pausa por um tempo determinado pelo valor de quanta recebido no pedido.

- IFG

- ✓ **IFGset** e **RX_IFG_SET**

O registrador **IFGset** define o número mínimo de ciclos de *clock* que devem ser aguardados no envio de dois pacotes consecutivos. De acordo com os padrões IEEE802.3, o valor mínimo do IFG é de 96 bits, desta forma, se o *clock* do sistema é de 125Mhz e o barramento é de 8 bits, o intervalo de tempo entre pacotes é de pelo menos dezoito ciclos de *clock*.

Já o registrador **RX_IFG_SET** é utilizado para definir o intervalo de tempo mínimo entre os pacotes recebidos consecutivamente. Se o intervalo de tempo entre dois pacotes recebidos for menor do que o valor deste registrador, o segundo pacote será considerado como um pacote inválido e, consequentemente, será descartado.

- Full Duplex

  - ✓ **FullDuplex**

Este registrador define se o sistema está operando em modo *full duplex* ou não. Quando o seu valor é “1”, o módulo estará operando em *full duplex*, caso contrário, se o seu valor é “0”, o módulo estará operando em *half duplex* e realizará operações de *back off,* detecção de colisão, entre outras.

  - ✓ **MaxRetry**

Quando o sistema está operando no modo *half duplex* e ocorre uma colisão, ele tentará retransmitir o pacote que sofreu a colisão. Se um mesmo pacote colidir **MaxRetry** vezes, ele será descartado.

- Endereço físico do remetente

  - ✓ **MAC_tx_add_en**

Quando este registrador tem valor lógico igual a “1”, o sistema está habilitado a adicionar seu endereço físico ao pacote que será enviado. Portanto, para todo pacote enviado, é adicionado o endereço físico do transmissor do *frame* Ethernet. Se o valor lógico deste registrador for igual a “0”, o módulo não realizará esta operação.

  - ✓ **MAC_rx_add_prom_data_h1, MAC_rx_add_prom_data_h2 e MAC_rx_add_prom_data_h3**

Esse conjunto de registradores define o endereço físico do módulo de forma que, se o valor padrão for utilizado, o endereço será 00-12-34-56-78-90, onde cada registrador armazena dois bytes. O registrador **MAC_rx_add_prom_data_h1** armazena os dois bytes mais significativos, o registrador **MAC_rx_add_prom_data_h2** armazena o terceiro e o quarto

byte mais significativos e o registrador **MAC_rx_add_prom_data_h3** armazena os dois bytes menos significativos.

- Filtro *broadcast*

  - ✓ **broadcast_filter_en**

O filtro *broadcast* limita o número de pacotes, deste tipo, lidos pelo sistema em um certo intervalo de tempo e o registrador **broadcast_filter_en** define se o filtro *broadcast* está habilitado, ou desabilitado.

  - ✓ **broadcast_bucket_depth e broadcast_bucket_interval**

O registrador **broadcast_bucket_depth** determina o número máximo de bytes referentes a pacotes *broadcast* que podem ser recebidos em certo período de tempo, esse período é definido pelo registrador **broadcast_bucket_interval,** o qual especifica a quantidade de ciclos de *clock* que devem decorrer antes que o contador de bytes seja zerado. O funcionamento do filtro esta descrito na seção 3.2.5.4.

- CRC

  - ✓ **RX_APPEND_CRC**

Em algumas condições, a aplicação do usuário precisa reter o valor do CRC do *frame* Ethernet recebido. Quando **RX_APPEND_CRC** tem valor lógico igual a “1”, o CRC do *frame* Ethernet será enviado para a aplicação requisitante, no caso o software rodando no processador Nios II, junto com os dados do pacote recebido.

  - ✓ **CRC_chk_en**

Por padrão, a lógica de recepção irá descartar qualquer pacote que contenha erro na checagem do CRC. Mas, configurando o registrador **CRC_chk_en** como zero, é possível desabilitar esta checagem,

consequentemente, o módulo passa a receber pacotes com erro na checagem do campo de FCS.

- *Loopback*

  - ✓ **Line_loop_en**

Quando este registrador está definido como “1”, o pacote transmitido para o chip físico irá ser redirecionado para a recepção do módulo e será recebido como se fosse um pacote externo.

- Velocidade de operação

  - ✓ **Speed**

Este registrador indica a velocidade e o modo de operação do módulo, o seu valor é definido pelo chip físico através da função de auto negociação.

A este módulo foi integrado também, uma máquina de estados capaz de realizar a configuração do chip físico. Quando o sistema é iniciado, esta máquina de estados acessa o chip físico utilizando o barramento do módulo Eth_miim e realiza tanto operações de escrita como operações de leitura nos registradores do chip. Esta é responsável por verificar se a auto negociação do chip foi completada, fazer a leitura da velocidade de operação determinada para aquela rede e guardar o valor desta velocidade no registrador **Speed**.

3.3 NIOSII_SOPC

Este módulo acrescenta o processador Nios II da Altera ao sistema, tal processador foi implementado utilizando a ferramenta SOPC Builder de modo a dispor de um núcleo e diversos dispositivos periféricos. O processador, em conjunto com seus periféricos, é mostrado na Figura 3.24, na qual é possível ver o barramento Avalon interligando os diversos dispositivos. A configuração de cada dispositivo é descrita a seguir:

- Nios II processor

Foi utilizado a versão rápida (Nios II/f) do processador Nios II, a qual permite a troca de dados pela interface JTAG UART e é a que retorna um maior desempenho em relação as três versões do Nios II.

- On-Chip Memory (RAM or ROM)

Foi utilizada uma memória RAM interna com 256 Kbytes de espaço para armazenamento do programa e dos dados.

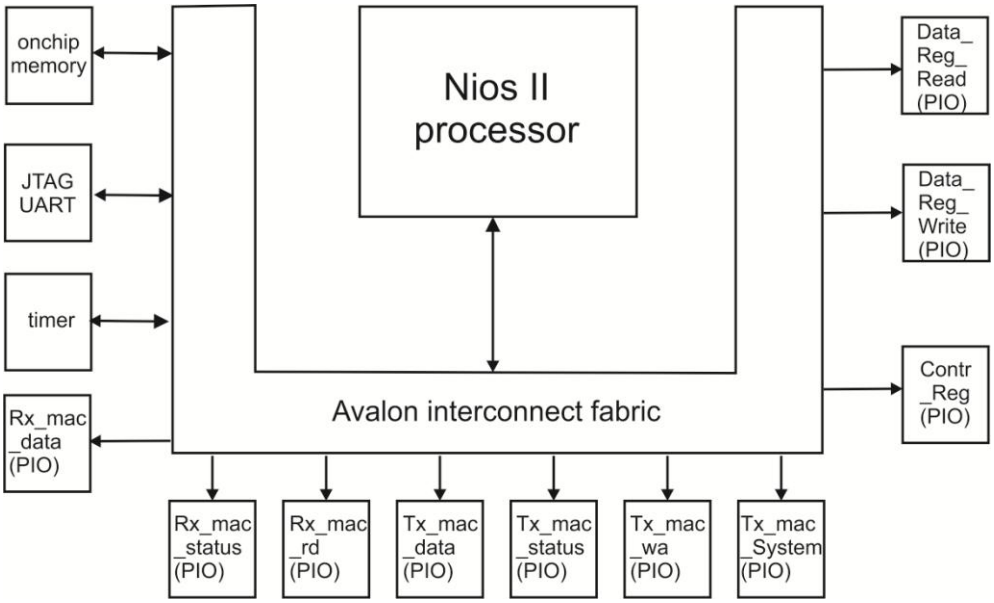


Figura 3.24 - Diagrama representando as ligações dos dispositivos periféricos com o processador Nios II, através do barramento Avalon. Fonte: elaborada pelo autor.

- JTAG UART

A interface JTAG UART é utilizada para carregar o software na memória do processador, além de fornecer uma ferramenta para verificação do sistema, permitindo a escrita de dados na tela do computador.

- Interval Timer

O timer foi configurado com período equivalente a 1ms e é usado para gerar interrupções no processador, permitindo o funcionamento de vários processos concorrentemente.

- Parallel I/O

As interfaces paralelas de entrada e saída são utilizadas para que o processador seja capaz de trocar dados com o módulo MAC, sendo assim, foram criadas dez interfaces para troca de dados, as quais estão descritas abaixo:

✓ Rx_mac_data

Os dados contidos em um *frame* são enviados ao processador em blocos de 32 bits através deste barramento.

✓ Rx_mac_Status

Esta porta de entrada de 5 bits recebe o estado dos dados que estão contidos na porta Rx_mac_data e tem o padrão indicado em (13), em ordem do bit mais significativo para o menos significativo.

$$\text{BE(2 bits), sop(1 bit), eop(1 bit), ra(1 bit)} \qquad \textbf{(13)}$$

Os dois bits mais significativos, BE, indicam quais dos 32 bits recebidos pela porta Rx_mac_data são válidos. Caso seu valor seja “00” todos os bits são válidos. Caso seu valor seja “01”, apenas o byte mais significativo é válido. Caso seu valor seja “10” apenas os dois bytes mais significativos são válidos e, por fim, caso seu valor seja “11”, apenas os três bytes mais significativos são válidos.

Já o terceiro bit mais significativo sop, indica se esse bloco corresponde ao primeiro bloco do pacote que está sendo recebido (*Start Of Packet*), caso seu valor

seja “1”, este é o primeiro bloco de dados do pacote, caso seu valor seja “0”, o bloco de dados recebido não corresponde ao primeiro.

O quarto bit mais significativo, eop, indica se o bloco recebido equivale ao ultimo bloco do pacote (*End Of Packet*), se o seu valor for “1”, ele é o último bloco do pacote, mas se o seu valor for “0”, este bloco não corresponde ao último.

Quanto ao bit menos significativo, ra, este indica se o MAC tem algum dado disponível para leitura. Seu valor é “1” quando há pelo menos um pacote disponível. Caso contrário, seu valor é “0”.

- ✓ Rx_mac_rd

Esta porta de saída de um bit faz a requisição de dados ao MAC. Uma vez que o dado esteja disponível, o bit referente a esta porta é elevado a um pelo software, e em seguida abaixado para zero, o hardware entende esse sinal como uma requisição de dados e envia o bloco de dados que está armazenado na fila, juntamente com seu estado.

- ✓ Tx_mac_data

Esta porta de saída de 32 bits envia os dados que serão transmitidos pelo MAC.

- ✓ Tx_mac_Status

Esta porta de saída de cinco bits envia o estado dos dados que estão contidos na porta Tx_mac_data e tem o padrão indicado em (14), em ordem do bit mais significativo para o bit menos significativo.

BE(2 bits), sop(1 bit), eop(1 bit), wr(1 bit) **(14)**

Os dois bits mais significativos, BE, indicam quais dos 32 bits enviados pela porta Tx_mac_data são válidos. Caso seu valor seja “00” todos os bits são válidos. Caso seu valor seja “01”, apenas o byte mais significativo é válido. Caso seu valor seja “10” apenas os dois bytes mais significativos são válidos e, por fim, caso seu valor seja “11”, apenas os três bytes mais significativos são válidos.

Já o terceiro bit mais significativo, sop, indica se esse bloco corresponde ao primeiro bloco do pacote que está sendo transmitido (*Start Of Packet*), caso seu valor

seja “1”, este é o primeiro bloco de dados do pacote, caso seu valor seja “0”, o bloco de dados transmitido não corresponde ao primeiro.

O quarto bit mais significativo, eop, indica se o bloco transmitido equivale ao último bloco do pacote (*End Of Packet*), se o seu valor for “1”, ele é o último bloco do pacote, mas se o seu valor for “0”, este bloco não corresponde ao último.

O bit menos significativo, wr, habilita o envio do bloco de dados ao MAC. Se o seu valor seja “1” o bloco está disponível e será enviado, caso contrário, seu valor seja “0”, não há bloco a ser enviado.

- ✓ Tx_mac_wa

Esta porta de entrada de um bit indica se o usuário fez uma requisição para envio de um pacote de teste pela rede. A requisição é feita ao pressionar o botão zero do FGPA.

- ✓ Tx_mac_System_wa

Esta porta de entrada de um bit indica se o sistema está disponível para transmissão, caso seu valor esteja em “1”, a transmissão está disponível, caso seu valor esteja em “0”, a transmissão não está disponível.

- ✓ Contr_Reg

Esta porta de saída de oito bits é utilizada para realizar operações de escrita e leitura nos registradores internos do MAC. Ela tem o padrão indicado em (15), em ordem do bit mais significativo para o bit menos significativo.

wr_CA(1 bit), rd_CA(1 bit), CA_Addr(6 bits) **(15)**

A ação de elevar o bit wr_CA a “1” e voltar o seu valor a “0” estabelece uma operação de escrita no registrador, que tem seu endereço definido nos seis bits CA_Addr. Essa mesma ação realizada no bit rd_CA define uma operação de leitura no mesmo registrador.

- ✓ Data_Reg_Write

Esta porta de saída de 16 bits é utilizada para escrever dados no registrador definido na porta Contr_Reg. Antes de realizar uma operação de escrita, é necessário definir o valor desta porta para que a escrita possa ser realizada corretamente no registrador.

- ✓ Data_Reg_read

Esta porta de entrada de 16 bits é utilizada para ler os dados do registrador definido na porta Contr_Reg. Uma vez realizada a operação de leitura no registrador, o dado contido nele é recebido nesta porta.

Como o processador e, consequentemente, o *software* que está rodando nele são mais lentos que o *hardware* desenvolvido, foi necessário implementar dois módulos que fazem o interfaceamento da comunicação entre o processador e o *hardware*. Um dos módulos realiza o interfaceamento do processador com os registradores internos e o outro, o interfaceamento do processador com os barramentos do MAC que correspondem à transferência de dados pelo canal Ethernet.

3.4 NiosII_Ctrl

Este módulo realiza o interfaceamento do barramento de transferência de dados entre o MAC e o processador Nios II. Sua necessidade vem do fato de que o MAC, desenvolvido em *hardware*, tem um tempo de resposta mais rápido que o *software* de controle implementado no processador.

Considerando primeiramente o controle sobre o barramento de recepção, o módulo aguarda pelo sinal de leitura enviado pelo processador através da porta Rx_mac_rd e, se a leitura estiver disponível, ele requisita um bloco de dados ao MAC. Esse bloco de dados e o seu respectivo estado é enviado ao processador através das portas Rx_mac_data e Rx_mac_Status, respectivamente. Na sequência, o módulo volta a aguardar por outro sinal de leitura. A sequência é mostrada na Figura 3.25, onde se tem o sinal de *clock* como referência,

o sinal de requisição de leitura vindo do processador, o sinal de requisição de leitura enviado ao MAC, os barramentos de estado e dados vindos do MAC e os barramentos de dados e estado enviados ao processador, respectivamente. Nesta figura é possível ver a diferença de velocidade entre o MAC e o processador, na qual o segundo demora vinte ciclos de *clock* para realizar um pedido de leitura, enquanto que o primeiro precisa apenas de um ciclo para receber o pedido.

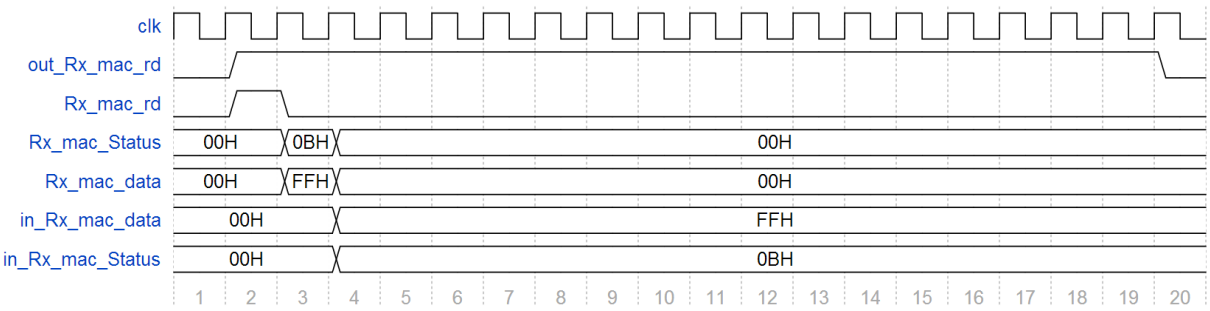


Figura 3.25 - Exemplo de uma leitura de dados realizada pelo processador. Fonte: elaborada pelo autor.

Considerando, contudo, o controle sobre o barramento de transmissão, o módulo aguarda por um sinal de transmissão vindo do processador através do bit menos significativo da porta Tx_mac_Status. Então, se o envio de pacotes estiver habilitado, o módulo enviará ao MAC os dados vindos do processador através da porta Tx_mac_dados juntamente com o estado destes dados, que são recebidos nos quatro bits mais significativos da porta Tx_mac_Status. O processo de transmissão é mostrado na

Figura 3.26, onde se tem o *clock* de referência, o bit de leitura habilitada, o bit de transmissão vindo do processador, os barramentos de transmissão dos dados e do estado vindos do processador, e os barramentos de transmissão dos dados e do estado enviados ao MAC, respectivamente.

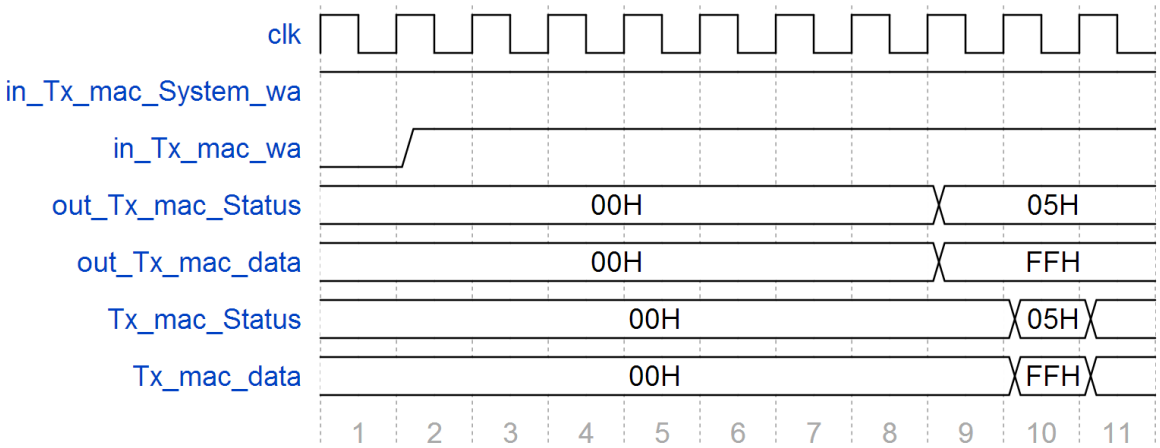


Figura 3.26 - Exemplo de uma transmissão de dados realizada pelo processador. Fonte: elaborada pelo autor.

É possível, ainda, enviar um pacote de testes ao pressionar o botão zero do kit de desenvolvimento. Então, este módulo verifica constantemente a condição desse botão e, caso seja pressionado, uma requisição de envio de pacote de testes é enviada ao processador.

3.5 Reg_int_wr

O módulo Reg_int_wr é responsável pelo interfaceamento dos barramentos para escrita e leitura nos registradores internos do MAC. Para fazer isso, o módulo dispõe de uma

máquina de estados que controla o acesso aos registradores, a qual é mostrada na

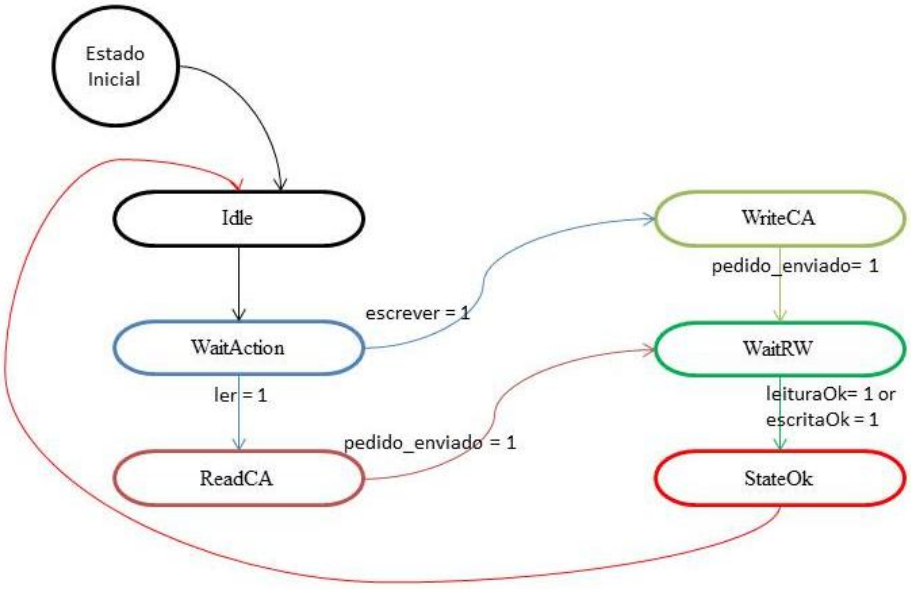


Figura 3.27. O estado inicial desta máquina é o estado **Idle** e sua evolução é descrita a seguir.

- **Idle**

  Nesse estado a máquina limpa os valores das operações anteriores.

  ✓ Ao chegar a este estado, a máquina passa automaticamente para o estado **WaitAction**.

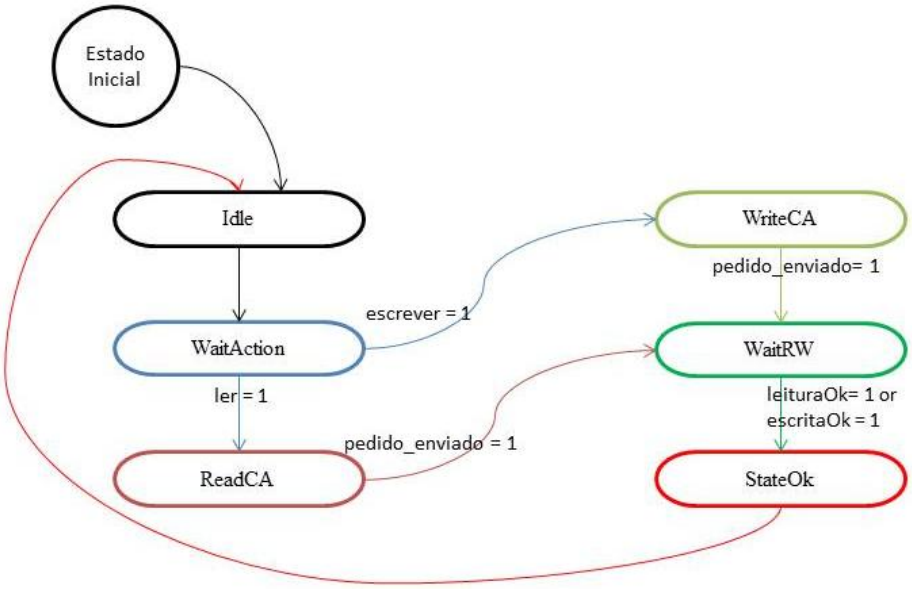


Figura 3.27 - Diagrama de transições da máquina de estados do módulo Reg_int_wr, a qual controla o acesso aos seus registradores. Fonte: elaborada pelo autor.

- **WaitAction**

  Nesse estado a máquina esta ociosa, aguardando por uma requisição de operação.

  - ✓ Caso uma operação de leitura tenha sido requisitada pelo processador, a máquina passa para o estado **ReadCA**.
  - ✓ Caso contrário, se um operação de escrita tenha sido requisitada pelo processador, a máquina passa para o estado **WriteCA**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **WriteCA**

  Nesse estado a máquina envia o endereço do registrador no qual será realizada a operação de escrita para o MAC, juntamente com o valor a ser escrito e o sinal que faz a requisição de escrita.

  - ✓ Caso o pedido de escrita tenha sido enviado ao MAC, a máquina passa para o estado **WaitRW**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **ReadCA**

  Nesse estado a máquina envia o endereço do registrador no qual será realizada a operação de leitura para o MAC, juntamente com o valor que faz a requisição de escrita.

  - ✓ Caso o pedido de leitura tenha sido enviado ao MAC, a máquina passa para o estado **WaitRW**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **WaitRW**

  Nesse estado a máquina zera os sinais referentes às requisições de escrita ou leitura.

  - ✓ Caso a operação de leitura ou de escrita tenha sido realizada, a máquina passa para o estado **StateOk**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

- **StateOk**

  Neste estado a máquina repassa o dado lido para o processador, se a operação realizada tenha sido de leitura.

  - ✓ Caso a operação realizada tenha sido de leitura e o dado requisitado já tenha sido passado ao processador, a máquina volta para o estado **Idle**.
  - ✓ Caso contrário, a máquina permanece no estado atual.

3.6 Software

O *software* projetado para controlar o MAC é capaz de:

- Ler e escrever nos registradores internos.

Com o auxilio do módulo Reg_int_wr descrito na seção 3.5, o software é capaz de realizar operações de leitura ou escrita nos registradores internos do MAC, podendo desta forma alterar valores como o endereço físico do sistema ou a profundidade do filtro *broadcast,* ou, ainda, ler o valor do registrador Speed para aferir a velocidade de operação do sistema.

- Mostrar na tela todos os pacotes recebidos e transmitidos.

A interface JTAG UART permite a troca de dados entre o *software* e um computador através de um link USB, isso permite que o *software* escreva os dados na tela desse computador, tornando possível a verificação dos dados que estão sendo tratados pelo sistema.

- Enviar um *frame* de teste.

O *software* verifica constantemente o botão zero do kit de desenvolvimento, para se caso este for pressionado, ele envie um *frame* de teste previamente definido.

- Responder pacotes PING e ARP.

O canal de recepção de dados é constantemente verificado pelo *software*, de forma que quando um pacote é recebido, ele é impresso na tela do computador e, caso corresponda a um pacote ARP ou PING, uma resposta adequada é gerada e enviada.

O *software* desenvolvido roda em cima do sistema operacional ChibiOS/RT, um sistema operacional de tempo real comentado na seção 2.3.8, e está dividido em uma *thread* principal e duas *threads* auxiliares. A primeira, primeiramente, faz a inicialização do sistema, configurando alguns registradores do MAC, como o registrador que define o endereço físico do sistema e o registrador que define se o filtro *broadcast* estará habilitado ou não.

Na sequência esta thread verifica alguns registradores internos e os imprime na tela, como é mostrado na Figura 3.28.

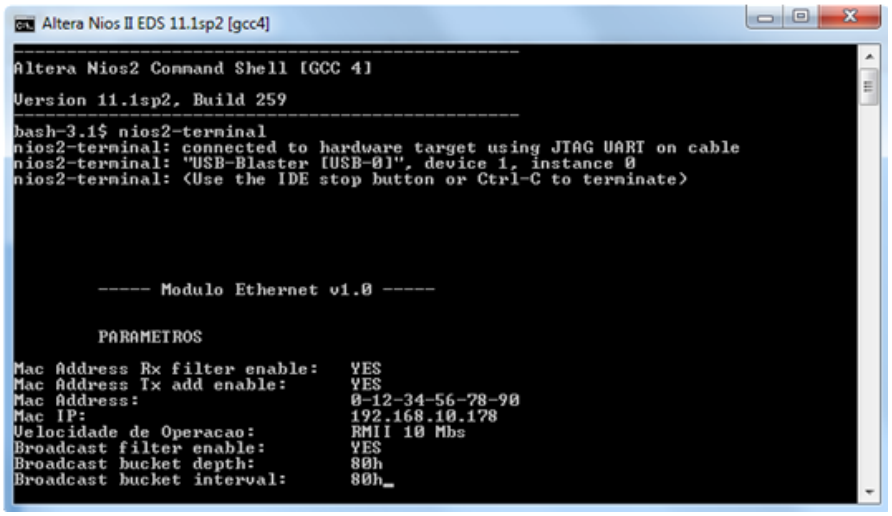


Figura 3.28 - Tela de início do software rodando no processador Nios II, onde são mostradosvalores de alguns registradores internos do sistema. Fonte: elaborada pelo autor.

É possível ver que o filtro do endereço físico para pacotes recebidos está habilitado, assim como a filtro que adiciona o endereço físico ao pacote a ser transmitido. Também é possível ver o valor do endereço físico e do endereço IP, além da velocidade e modo de operação, que no caso é o RMII com velocidade igual a 10 Mbps. Por último, é mostrado que o filtro *broadcast* está habilitado e o valor do seu intervalo e da sua profundidade equivale ao valor hexadecimal oitenta.

Em seguida, a *thread* principal executa a chamada das duas *threads* auxiliares e depois entra em um *loop* infinito, dentro do qual é feita a verificação do sistema de transmissão. A cada interação deste loop, a *thread* entra no estado *sleep* por quinhentos milissegundos.

Já a primeira *thread* auxiliar executa um *loop* infinito, dentro do qual o canal de recepção de pacotes é verificado, e caso um pacote tenha sido recebido, sua leitura é feita e os seus dados são impressos na tela. Na sequência, o conteúdo deste pacote é verificado e, caso este corresponda a um pedido do tipo ARP ou PING, um resposta ARP ou PING é gerada e enviada ao computador que a solicitou. O pacote enviado também é mostrado na tela. Após estas operações, a *thread* entre no estado *sleep* por um milissegundo antes que volte a executar a próxima interação do *loop*.

Por fim, a segunda *thread* auxiliar também executa um *loop* infinito, no qual é verificado se o botão zero da placa DE3_150 foi pressionado, em caso afirmativo um pacote de testes, previamente definido no software, é montado e enviado através da rede. A *thread* ainda entra no estado *sleep* por um milissegundo antes de executar outra interação.

Como ambas as *threads* auxiliares utilizam o canal de transmissão de pacotes, foi criado um semáforo que controla o acesso a este canal. Este semáforo é disponibilizado pelo sistema operacional e permite que apenas uma *thread* por vez utilize o canal. Se uma delas tentar transmitir um pacote enquanto a outra já o esta realizando, a primeira é colocada no estado *sleep* para aguardar o término da segunda, que ao terminar de utilizar o canal de transmissão, envia um sinal, acordando a primeira.

4 RESULTADOS

Uma vez tendo todo o sistema desenvolvido, alguns testes descritos nas seções abaixo foram executados para aferir o correto funcionamento deste. Entre as características testadas estão incluídas a correta transferência de dados, a máxima velocidade atingida para as três velocidades suportadas, a quantidade de recursos lógicos ocupados no FPGA e a capacidade de configuração do sistema através do processador.

Para esses testes foi conectado ao FPGA um computador de mesa equipado com um processador Intel Core i7 CPU 920 de 2.67GHz rodando no sistema operacional Windows 7©. Foi desenvolvido, também, um software em C que realiza a contagem dos pacotes recebidos e calcula a velocidade de transferência, utilizando como medida temporal a diferença entre os horários em que os pacotes chegaram ao computador.

Sabendo das dificuldades encontradas na transmissão de pacotes utilizando protocolo UDP em uma rede compartilhada, os quais incluem o monitoramento de todos os pacotes transmitidos para verificação de sua correta entrega, os testes foram realizados em uma rede local ponto a ponto, onde o sistema desenvolvido é conectado diretamente a um computador utilizando um cabo par trançado. Esta mesma abordagem será utilizada na instalação do espectrômetro afim de evitar problemas relacionados à transferência de dados.

4.1 Taxa de transferência

Um dos principais objetivos relacionados ao projeto é a sua capacidade de transmitir dados em alta velocidade para o computador que está conectado ao sistema através de uma rede ponto a ponto. Visando, então, aferir a capacidade de transmissão do sistema nas diferentes velocidades foi realizado o teste descrito.

Primeiramente, sabendo que o processador integrado ao MAC pode influenciar significativamente neste processo, a velocidade foi aferida sem a inclusão deste ao projeto. Foi desenvolvido, então, um módulo auxiliar chamado de UDP_Sender, o qual se comunica diretamente com o MAC para enviar um pacote UDP de 1512 bytes previamente definido. A escolha do protocolo UDP para transferência de dados tem como objetivo a diminuição do *overhead* relacionado a essa transferência. O pacote é enviado 69351 vezes, totalizando 100 MB de dados transferidos, e o intervalo entre a transmissão de dois pacotes consecutivos é definida como o intervalo padrão do módulo, que é de dezoito ciclos de *clock*, equivalente a 144 ns para 1000 Mbps.

É importante lembrar, também, que como o processador foi removido nesta arquitetura, as respostas por parte do sistema de pacotes PING e ARP não podem ser efetuadas. Dessa forma, foi necessário relacionar o IP do MAC com o seu endereço físico, manualmente, no *cache* do sistema operacional.

Outro ponto que deve ser destacado é a velocidade com que o programa desenvolvido consegue realizar a leitura dos pacotes recebidos. Uma vez que o *buffer* de recepção não chega a 100 MB, a aplicação deve ser capaz de ler os pacotes na velocidade em eles são transmitidos. Para isso, entre a recepção de dois pacotes consecutivos, o software não realiza operação alguma, com a exceção do incremento do contador de pacotes e da adição do tamanho do pacote recebido ao montante de dados já recebidos anteriormente. Dessa forma, apenas para o primeiro pacote e para o último, a leitura do horário em que foi recebido é realizada. Após a recepção de todos os pacotes, o cálculo do intervalo de tempo de recepção é feito, bem como a taxa de transferência conseguida.

Os testes foram feitos para velocidades de 1000 Mbps, 100 Mbps e 10 Mbps, os quais são mostrados na Tabela 4.1, aonde é possível observar que, em nenhum dos casos, a velocidade máxima foi atingida. Isso é explicado pela necessidade de um período de espera entre o envio de dois pacotes consecutivos (IFS - *InterFrame Gap),* o qual deve ser de no mínimo 9,6 µs para 10 Mbps (Ethernet), 0,96 µs para 100 Mbps (*fast* Ethernet), 0,096 µs para 1000 Mbps (Gigabit Ethernet), ou seja, não é possível manter a taxa de transferência na velocidade máxima quando é enviado mais de um pacote. Mesmo assim, a taxa atingida esteve bem próxima da taxa máxima para os três casos, variando entre 93,2% e 93,3% do máximo. Esses valores foram obtidos a partir da média de dez medidas para cada velocidade.

Ao comparar os resultados obtidos neste trabalho com os descritos na literatura (23), onde a velocidade atingida para 1000 Mbps foi de 115,6 MBps, nota-se que são próximos e a diferença se deve, provavelmente, ao fato de que no artigo citado os pacotes tinham tamanho de 1472 bytes, ou seja, 40 bytes a menos do que neste trabalho. Outra diferença é em relação ao intervalo de tempo aguardado entre dois pacotes enviados consecutivamente, enquanto neste trabalho o tempo foi de dezoito ciclos (144 ns), no trabalho citado foram aguardados quarenta ciclos (320 ns).

Tabela 4.1- Média da velocidade máxima de transferência de dados atingida para as velocidades de 1000, 100 e 10 Mbps sem a utilização do processador Nios II e sem a consideração do overhead referente ao protocolo UDP.

Velocidade de operação (Mbps)	Velocidade atingida (Mbps)	Desvio médio (Mbps)	Velocidade atingida (MBps)	Desvio médio (MBps)
1000	931,9	±0,1	116,49	±0,01
100	93,1572	±0,0003	11,64465	±0,00004
10	9,329198	±0,000002	1,1661498	±0,0000002

Fonte: elaborada pelo autor.

No entanto, essa análise está desconsiderando o *overhead* proveniente do protocolo de transmissão, no caso o UDP. Esse protocolo necessita de 42 bytes de controle para realizar a transferência. Dessa forma, refazendo os cálculos, a velocidade de transferência de dados realmente atingida é mostrada na Tabela 4.2, aonde se observa que a velocidade máxima atingida é de aproximadamente 906 Mbps, o que representa um aproveitamento de, aproximadamente, 91% da largura de banda total da tecnologia.

Tabela 4.2 - Média da velocidade máxima de transferência de dados atingida para as velocidades de 1000, 100 e 10 Mbps sem a utilização do processador Nios II, mas levando em consideração o overhead referente ao protocolo UDP.

Velocidade de operação (Mbps)	Velocidade atingida (Mbps)	Desvio médio (Mbps)	Velocidade atingida (MBps)	Desvio médio (MBps)
1000	906,0	±0,1	113,25	±0,01
100	90,5695	±0,0003	11,32118	±0,00003
10	9,070054	±0,000002	1,1337567	±0,0000002

Fonte: elaborada pelo autor.

Também foram feitos testes utilizando o projeto completo, ou seja, com o processador Nios II integrado ao MAC. O padrão utilizado anteriormente foi mantido, de forma que os pacotes continuaram sendo de 1512 bytes e a quantidade de dados enviados continuou sendo de 100 MB. Neste caso, foi possível ver a limitação do processador em termos da velocidade

de transferência de dados, uma vez que a taxa de transmissão não ultrapassou 81 Mbps como pode ser visto na Tabela 4.3. Esse comportamento era esperado, uma vez que o processador, que opera a 125 MHz, necessita alterar com muita frequência os bits das portas que fazem a comunicação entre o software e o MAC, processo este que demanda alguns ciclos de *clock* e varia de acordo com a versão utilizada do processador. Uma abordagem que pode acelerar esse processo, tirando do processador a responsabilidade pela transferência dos dados, é a utilização de DMA, a qual faria automaticamente a comunicação entre a memória do processador e as portas de entrada e saída. Dessa forma, o processador ficaria responsável apenas pelo processamento, melhorando o seu desempenho.

Tabela 4.3 - Média da velocidade máxima de transferência de dados atingida para as velocidades de 1000, 100 e 10 Mbps utilizando o processador Nios II/f, mas sem levar em consideração o overhead referente ao protocolo UDP.

Velocidade de operação (Mbps)	Velocidade atingida (Mbps)	Desvio médio (Mbps)	Velocidade atingida (MBps)	Desvio médio (MBps)
1000	80,7741	±0,0009	10,0968	±0,0001
100	80,7740	±0,0002	10,09675	±0,00002
10	9,32803	±0,00002	1,166004	±0,000002

Fonte: elaborada pelo autor.

4.2 Comparação de velocidade entre as versões e a porcentagem utilizada

Outro item testado é a velocidade de transmissão de dados para as diferentes versões do Nios II. Na Tabela 4.4 são mostradas as velocidades atingidas para as três versões disponíveis. A velocidade de operação do sistema era de 1000 Mbps e, assim como nos exemplos apresentados na seção 4.1, foram enviados 100 MB através da rede.

Tabela 4.4 - Média da velocidade máxima de transferência de dados atingida para a velocidade de 1000 Mbps utilizando as três versões do Nios II disponíveis, mas sem levar em consideração o overhead referente ao protocolo UDP.

Versão do Nios II utilizada	Velocidade atingida (Mbps)	Desvio médio (Mbps)	Velocidade atingida (MBps)	Desvio médio (MBps)
Nios II/f	80,7741	±0,0009	10,0968	±0,0001
Nios II/s	65,32716	±0,00002	8,165895	±0,000003
Nios II/e	18,777219	±0,00002	2,3471526	±0,0000005

Fonte: elaborada pelo autor.

Nos dados apresentados é possível observar a diferença entre as velocidades de transmissão de cada versão do processador. Como esperado, a variação na velocidade foi decrescente conforme a capacidade da versão. A vantagem, então, de se utilizar uma versão inferior do Nios II está relacionada à diminuição no número de recursos lógicos ocupados por ela.

A Tabela 4.5 mostra uma comparação entre a lógica utilizada pelo projeto, com quatro abordagens diferentes, na qual a primeira sem a implementação do Nios II e as três seguintes, com a implementação de cada versão do processador. Na tabela é notado que a quantidade de lógica necessária para adicionar o processador ao projeto é maior que o dobro da utilizada inicialmente, mesmo para a versão econômica do Nios II.

Tabela 4.5 - Percentual de utilização da FPGA, com e sem a adição de cada uma das três versões do processador Nios II.

	Porcentagem de lógica utilizada	**ALUTs combinacional**	**ALUTs Memória**	**Registros lógicos dedicados**
Sem Nios II	2 %	1 364/113 600	0/56 800	1 559/113 600
Nios II/e	5 %	3 348/113 600	128/56 800	4 920/113 600
Nios II/s	6 %	3 637/113 600	0/56 800	5 098/113 600
Nios II/f	6 %	3 914/113 600	0/56 800	5 430/113 600

Fonte: elaborada pelo autor.

4.3 Testes PING e ARP e registradores internos

Com o objetivo de testar o funcionamento correto do processador e a coerência dos dados enviados pela rede, foram utilizados os procedimentos de respostas a pacotes do tipo ARP e PING implementados por software. Neste teste, o computador envia um pedido ARP primeiramente e, em seguida, uma sequência de 501 pedidos de PINGs. Esses pedidos devem ser respondidos adequadamente.

O processo é mostrado na Figura 4.1, a qual representa uma sequência de dados impressos pelo software na tela do computador. Na figura, é possível ver os valores de alguns

registradores internos do MAC, como o endereço físico e o endereço IP do sistema. Acompanhando a sequência da figura, se tem um pacote ARP recebido pelo sistema, o qual é devidamente respondido pelo software. Por último, um pacote PING recebido pelo sistema, o qual também tem a sua resposta efetuada.

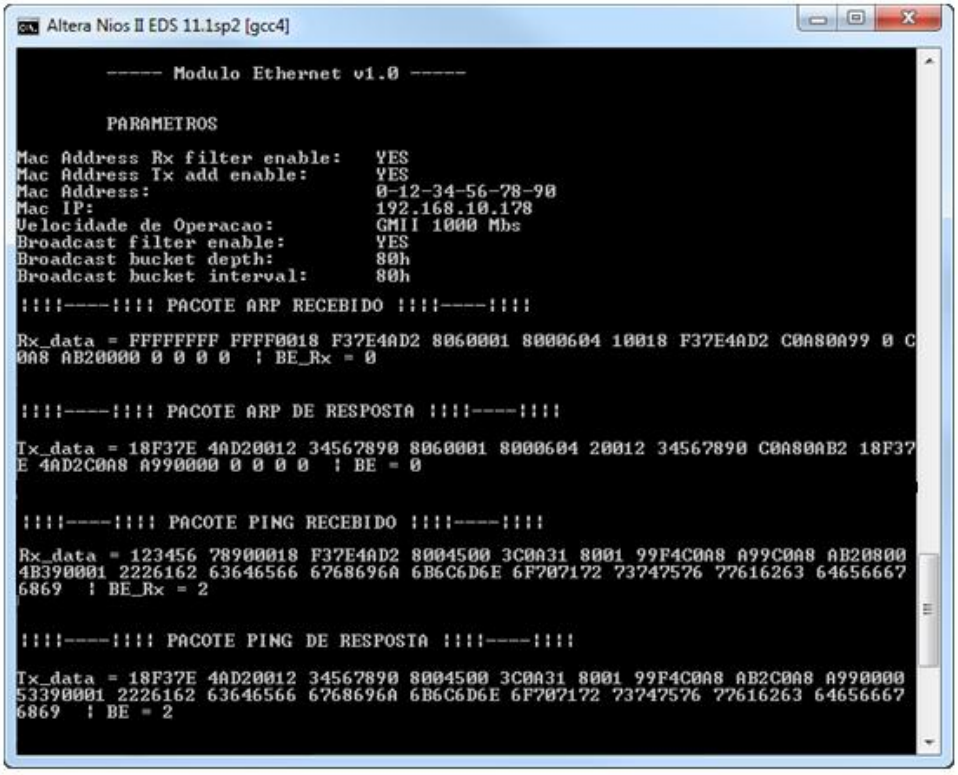
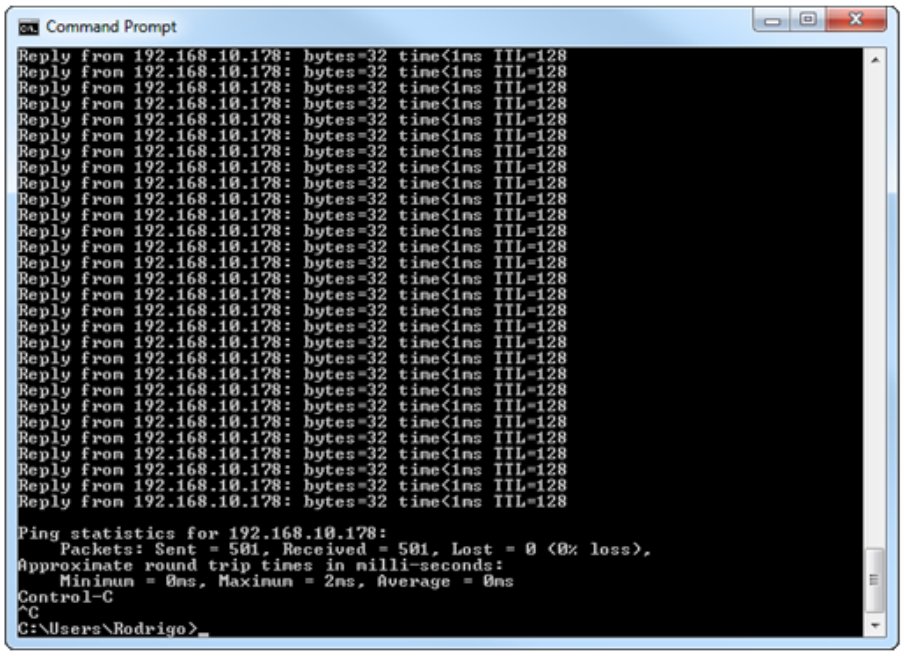


Figura 4.1- Exemplo de uma troca de pacote do tipo ARP e PING entre um computador e o sistema.
Fonte: elaborada pelo autor.

É possível observar esse processo por outro ângulo, o de quem realiza os pedidos de ARP e PING. Este está indicado na Figura 4.2, onde se tem uma sequência de pedidos do tipo PING respondidos pelo *software*. Desta forma, é constatado que foram feitos 501 pedidos, dos quais todos foram respondidos corretamente, com um intervalo de tempo entre a requisição e a resposta, efetuadas pelo computador e pelo software respectivamente, menor que um milissegundo.



```
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128
Reply from 192.168.10.178: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.178:
    Packets: Sent = 501, Received = 501, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 2ms, Average = 0ms
Control-C
^C
C:\Users\Rodrigo>_
```

Figura 4.2 - Pedidos consecutivos de PING para o sistema implementado utilizando o processador Nios II/f, na qual pode-se observar que 100% dos pedidos realizados foram respondidos. Fonte: elaborada pelo autor.

4.4 Testes no espectrômetro

Um protótipo do espectrômetro digital que está sendo desenvolvido no Centro de Imagens e Espectroscopia *in vivo* por Ressonância Magnética (CIERMag) já está em fase de testes, tanto do *software* quanto do *hardware* envolvido no projeto. Neste protótipo, está sendo utilizada uma versão do projeto deste trabalho, que consiste apenas do módulo MAC funcionando a velocidades de 1000 Mbps e 100 Mbps nos modos GMII e MII. O processador Nios II também não está incluso no protótipo. Isso se deve à necessidade de preservar a quantidade de recursos lógicos disponíveis no FPGA, uma vez que este é um dos fatores limitantes do desenvolvimento do espectrômetro. Em seu lugar, uma máquina de estados mais simplificada cuida do setup dos registros do MAC, define velocidade de comunicação e outros parâmetros necessários à sua operação.

A validação dos dados coletados e transmitidos pela rede, utilizando o MAC desenvolvido, é feita pelo software de aquisição no console de operação, denominado *Acquisition Server* devido à sua capacidade de operar com diversos consoles como clientes.

Tal validação mostrou um resultado satisfatório, no qual não foram encontrados erros durante a transmissão dos dados coletados.

Com relação ao problema abordado na introdução deste trabalho, o qual cita a quantidade de dados que o espectrômetro necessita trocar com o console de operação, se tem que a taxa máxima de transferência de dados real é de aproximadamente 906 Mbps, como mostrado na Tabela 4.2. Este valor permite a utilização de técnicas que variam o número de receptores e o tempo de recepção, como era esperado. Porém, não é possível fornecer valores máximos para essas variações, uma vez que outros fatores podem interferir na banda utilizada, como é o caso do modo de empacotamento dos dados de cada receptor.

Caso seja necessário aumentar a banda fornecida, o kit de desenvolvimento DE4® da Terasic, no qual o espectrômetro poderá ser instalado futuramente, irá auxiliar. Ele disponibiliza quatro links Gigabit Ethernet, como mostrado na Figura 4.3, de forma que, neste caso, poderiam ser instanciados quatro MACs. Cabe analisar, porém, outros fatores limitantes para se aumentar a taxa de dados transferidos como, por exemplo, a capacidade de recepção e processamento de dados no console de operação. Esta discussão será feita de forma mais detalhada quando o grupo do CIERMag adotar uma abordagem alternativa, considerando sistemas de recepção distribuídos, quando retomaremos esta análise.

Figura 4.3 - Kit de desenvolvimento DE4 da Terasic, o qual contém um FPGA Straix IV da Altera e quatro interfaces Gigabit Ethernet. Fonte: TERASIC-SOCKIT (18).

Outra opção que pode fornecer uma banda maior para comunicação entre o espectrômetro e o console de operação é a utilização de barramentos PCIe x8. Estes são disponibilizados, também, pelas placas DE4. A tecnologia citada permite uma taxa de transferência de até 5Gbps por canal e, como a placa citada tem oito canais disponíveis, a banda pode chegar a 40 Gbps Novamente, esta análise cuida apenas da capacidade operacional do link de comunicação, não levando em conta outros fatores que limitariam a utilização desta banda.

4.5 Testes com o LEON3

Em busca de um processador que permitisse ao grupo deter os direitos sobre sua propriedade intelectual, foi adicionado ao projeto o processador LEON3(Aeroflex Gaisler) no lugar do Nios II. A escolha deste levou em conta a sua ampla utilização no meio de desenvolvimento tecnológico (24) o que disponibiliza documentação e suporte aos desenvolvedores, outro ponto é a sua arquitetura baseada na SPARC V8, que promete um bom desempenho, mesmo a baixas frequências.

A síntese deste, porém, ocupou uma área maior do FPGA do que o Nios II, chegando a 10% da sua utilização. Outro ponto negativo deste processador foi a impossibilidade de responder pacotes PING adequadamente, fato que está provavelmente relacionado à velocidade de comunicação do processador com o hardware. Na figura 4.4 é mostrada uma sequência de quinze pedidos PING realizados por um computador, dos quais apenas sete foram respondidos corretamente, e com um atraso de 65 a 113 milissegundos. Dessa forma, foi optado pela utilização do processador Nios II neste trabalho.

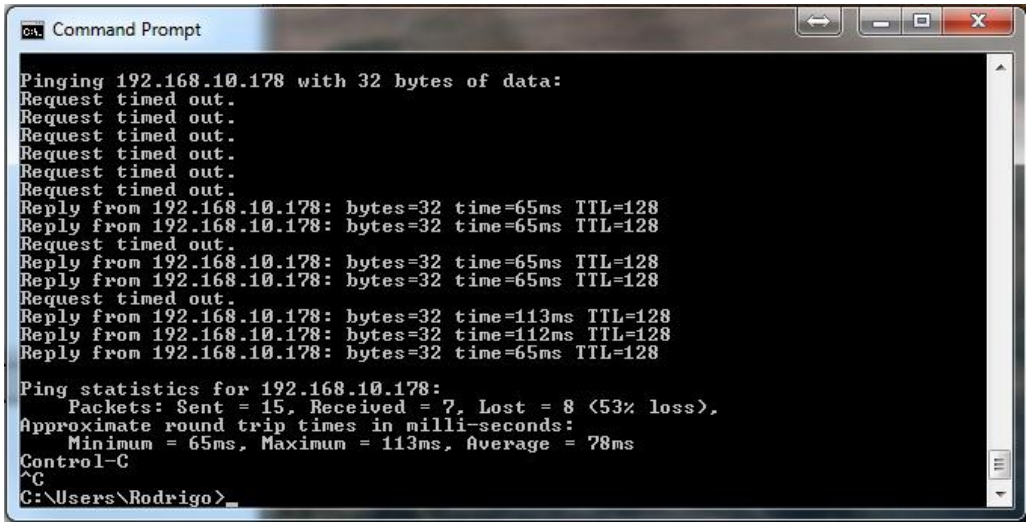


Figura 4.4 - Pedidos consecutivos de PING para o sistema implementado utilizando o processadorLEON3, na qual é possível ver que apenas 53% dos pedidos realizados foram respondidos. Fonte: elaborada pelo autor.

5 CONCLUSÃO

Analisando os objetivos deste trabalho conclui-se que a taxa de transferência alcançada com a comunicação Gigabit Ethernet satisfaz as necessidades atuais do espectrômetro, uma vez que as técnicas de aquisição que se esperam utilizar no momento se assemelham à do exemplo citado na introdução, de forma a necessitar de uma banda aproximada de 25 Mbps. Existem, porém, algumas técnicas de imagem, como é o caso da Sense ou da Smash, nas quais a taxa de transmissão pode exceder o limite fornecido de 906 Mbps, neste caso, ainda é possível utilizar mais de uma interface Gigabit Ethernet ou partir para outra abordagem de desenvolvimento, a utilização dos barramentos PCIe, como citado na seção 4.4. Entretanto, a limitação encontrada no momento para que tais técnicas sejam utilizadas, está na quantidade de recursos lógicos disponíveis no FPGA, além da capacidade de recepção e processamento de dados no console de operação.

Como mencionado no parágrafo acima, a quantidade de lógica utilizada é um dos pontos críticos no desenvolvimento do espectrômetro, sendo assim, observando os resultados relacionados a esta característica, pode-se dizer que ela atingiu o nível desejado, uma vez que, quando considerado apenas o módulo de comunicação, o projeto ocupa apenas 1364 ALUTs e 1559 registros lógicos dedicados, correspondendo a, aproximadamente, 2% do total dos recursos lógicos do kit utilizado.

A utilização do protocolo UDP para diminuir o *overhead* na transferência sem perder campos de checagem do pacote, como o campo de FCS, mostrou-se uma boa opção, uma vez que a diminuição na velocidade decorrente da utilização do protocolo foi inferior a 3% do valor total.

Já se tratando da propriedade intelectual e da tradução do código original de Verilog para VHDL, vale lembrar que ambos os itens são políticas de desenvolvimento do CIERMag. O primeiro permite a modificação de qualquer parte do código, bem como a sua utilização irrestrita, enquanto que o segundo fornece uma linguagem mais estruturada e com maior facilidade de detecção de incoerências. Erros como sinais declarados e não utilizados, blocos não funcionais e máquinas de estados com erros não aparentes foram encontrados com certa frequência no projeto original, porém, mediante a nova versão estes foram corrigidos.

Além destes, o trabalho permitiu a ampliação do conhecimento e de técnicas de desenvolvimento utilizando dispositivos reprogramáveis, bem como o aprendizado utilizando o Nios II e o Leon3, em relação à sintetização e utilização de processadores para esta tecnologia.

5.1 Trabalhos futuros

Como já mencionado neste trabalho, as novas DE4 trazem a possibilidade de aumentar a velocidade de transferência em duas frentes diferentes. A primeira consiste na utilização de até quatro interfaces Gigabit Ethernet em um mesmo dispositivo, permitindo, teoricamente, chegar a velocidades de aproximadamente 3700 Mbps. A segunda consiste no desenvolvimento de um módulo que realize a comunicação através da interface PCIe, a qual permite, também teoricamente, atingir velocidades de até 40 Gbps. Então, se houver necessidade, é possível aumentar consideravelmente a banda de transferência de dados.

Outro trabalho interessante seria a realização de um estudo mais aprofundado em relação aos processadores utilizados em FPGAs. Para este caso, o objetivo seria a utilização ou o desenvolvimento de um processador que permitisse a transferência de dados em maior velocidade e, de preferência, que tenha os direitos sobre a propriedade intelectual atrelados ao desenvolvedor do trabalho. Esta abordagem parece ser mais promissora com o advento de processadores *Hardcoded* nas FPGAs mais modernas da Altera (apesar desta prática ter sido adotada já há algum tempo pela Xilinx). Um exemplo disso é a nova placa de desenvolvimento SoCKit da terasic, nela está presente uma FPGA Cyclone V da altera, juntamente com um processador ARM Dual-Core CortexTM-A9 que opera a 800 Mhz (18).

REFERÊNCIAS

1 SPURGEON, C. E. **Ethernet**: the definitive guide. Cambridge: O'Reilly, 2000.

2 TABENBAUM, A. S. **Redes de computadores**. 4. ed. Rio de Janeiro: Elsevier, 2003.

3 WARREN JR., H. S. **Hacker's delight**. 2th ed. Upper Saddle River: Addison-Wesley Professional, 2013.

4 COMER, D. E. **Internetworking with TCP/IP:** principles, protocols, and architectures. 4th ed. Upper Saddle River: Prentice Hall, 2000.

5 ROSS, K. W.; KUROSE, J. F. **Redes de computadores e a internet:** uma abordagem Top-Down. 5. ed. Sao Paulo: Person Education do Brasil, 2010.

6 BOBDA, C. **Introduction to reconfigurable computing:** architectures, algorithms and applications. Dordech: Springer, 2007.

7 BROWN, S.; VRANESIC, Z. **Fundamentals of digital logic with VHDL design**. 3th ed. New York: McGraw-Hill, 2009.

8 ALTERA CORPORATION. **Stratix III device handbook.** San Jose: Altera, 2011. Disponivel em: <http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf>. Acesso em: 10 dez. 2013.

9 ALTERA CORPORATION. **Nios II processor reference handbook.** San Jose: Altera, 2011. Disponivel em: <http://altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf>. Acesso em: 10 dez. 2013.

10 WOLF, W. **FPGA-based system design**. Upper Saddle River: Prentice Hall, 2004.

11 CHU, P. P. **Embedded SOPC design with Nios II processor and VHDL examples**. Hoboken: Wiley, 2011.

12 CHIBIOS/RT in brief. 2013. Disponivel em: <http://www.chibios.org>. Acesso em: 19 fev. 2013.

13 CHIBIOS/RT. 2013. Disponivel em: <http://en.wikipedia.org/wiki/ChibiOS/RT>. Acesso em: 19 fev. 2013.

14 HOW to use ChibiOS/RT on a Nios II CPU based on an Altera DE1 Board. em4fun, 2012. Disponivel em: <http://www.emb4fun.de/fpga/niosii2/index.html>. Acesso em: 19 fev. 2013.

15 PHY (chip). 2013. Disponivel em: <http://en.wikipedia.org/wiki/PHY_(chip)>. Acesso em: 19 fev. 2013.

16 MARVELL. **Integrated 10/100/1000 ultra gigabit Ethernet transceiver**. Disponivel em: <http://html.alldatasheet.com/html-pdf/316946/MARVELL/88E1111/894/3/88E1111.html> Acesso em: 16 dez. 2013.

17 ALTERA. FPGA CPLD and ASIC from Altera. 2013. Disponivel em: <http://www.altera.com>. Acesso em: 19 fev. 2013.

18 TERASIC-SOCKIT**. SoCKit** - the development kit for new soc device**.** 2013. Disponivel em: <http://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&CategoryNo=165&No=816&PartNo=1>. Acesso em: 24 nov. 2013.

19. TERASIC. 2013. Disponivel em: <http://www.terasic.com>. Acesso em: 19 fev. 2013.

20. OPENCORES. 2013. Disponivel em: <http://www.opencores.org>. Acesso em: 19 fev. 2013.

21 OPENCORES. **10_100_1000 Mbps tri-mode Ethernet MAC:** overview. Disponivel em: <http://opencores.org/project.Ethernet_tri_mode>. Acesso em: 19 fev. 2013.

22 MAHMUD, R. **Techniques to make clock switching glitch free**. 2003. Disponível em: < http://www.design-reuse.com/articles/5827/techniques-to-make-clock-switching-glitch-free.html>. Acesso em: 10 dez. 2013.

23 ALACHIOTIS, N.; BERGER, S. A.; STAMATAKIS, A. Efficient PC-FPGA communication over gigabit Ethernet. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY (CIT 2010), 10., 2010. Bradford. **Proceedings...** Bradford: IEEE, 2010, p. 1727-1734. Disponivel em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5577968>. Acesso em: 10 dez. 2013.

24. LEON SPARC-V8. 2013. Disponivel em: <http://tech.groups.yahoo.com/group/leon_sparc>. Acesso em: 20 fev. 2013.

25 AEROFLEX GAISLER. **LEON 3,** multiprocessing CPU Core. Goteborg Sweden. 2010. Disponível em: < http://www.gaisler.com/doc/leon3_product_sheet.pdf>. Acesso em: 16 dez. 2013.

26 THOMPSON, G. **How 1000BASE-T works**, Montreal, 1997. Disponivel em: <http://grouper.ieee.org/groups/802/3/ab/public/nov97/geoff1.pdf>. Acesso em: 10 dez. 2013.

27 ALTERA CORPORATION. **Stratix IV devide handbook**.2012. Disponivel em:<http://www.altera.com/literature/hb/stratix-iv/stx4_5v1.pdf> Acesso em: 16 dez. 2013.

28 ALTERA DE4 development and education board. **Terasic**. Disponivel em: <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=138&No=501&PartNo=1#section>. Acesso em: 20 fev. 2013.