Pontifícia Universidade Católica do Rio Grande Do Sul
Faculdade de Engenharia - Faculdade de Informática
Engenharia de Computação

Utilização do Protocolo Ethernet para a Interface de Comunicação em Sistemas Digitais Embarcados em FPGAs

Augusto Martins Moraes

Volume Final do Trabalho de Conclusão de Curso
Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre, Junho de 2017

Utilização do Protocolo Ethernet para a Interface de Comunicação em Sistemas Digitais Embarcados em FPGAs

RESUMO

O aumento da utilização de FPGAs em sistemas digitais embarcados faz com que seja necessário desenvolver interfaces de comunicação com o hardware nestes dispositivos. O objetivo da utilização dessas interfaces de comunicação é tanto para envio/recepção de dados quanto para depuração. Dentre as formas usuais de comunicação com FPGAs, citam-se as interfaces: USB, PCI, Ethernet. A comunicação através do protocolo Ethernet representa uma vantagem significativa em relação às demais interfaces de comunicação, que é a possibilidade de acesso ao hardware implementado no FPGA de forma remota. Este volume final de TCC apresenta o desenvolvimento de um *Core-IP* que realiza essa interface, utilizando como base de desenvolvimento um *Core-IP* MAC disponível publicamente. Como prova de conceito, o *Core-IP* desenvolvido foi conectado a um *IP* de usuário, permitindo a comunicação desse IP de usuário com um computador externo.

Using the Ethernet Protocol as the Communication Interface on FPGA-based Embedded Digital Systems

ABSTRACT

The increased use of FPGAs in embedded digital systems makes necessary to develop communication interfaces for the hardware implemented on these devices. These communication interfaces enable to send/receive data to/from the FPGA, as well as debugging purposes. The communication protocols usually employed with FPGAs are USB, PCI, and Ethernet. Communication through the Ethernet protocol represents a significant advantage, which is the possibility of accessing the hardware implemented in the FPGA remotely. This end-of-term work presents the development of an IP-core to make this interface, using as a reference for the development a publicly available MAC Core-IP. As a proof-of-concept, the developed IP-core is connected to an user-IP, allowing the communication of this user-IP with an external computer.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

BD	-	Buffer Descriptor
BRAM	-	*Block Random Access Memory*
CRC	-	*Cyclic Redundancy Check*
CSMA/CD	-	*Carrier Sense Multiple Access* / *Collision detection*
FCS	-	*Frame Check Sequence*
FF	-	*Flip Flop*
FIFO	-	*First In First Out*
FMC	-	*FPGA Mezzanine Card*
FPGA	-	*Field-Programmable Gate Array*
FSM	-	*Finite State Machine*
IEEE	-	Institute of Electrical and Electronics Engineers
IO	-	*Input Output*
IP	-	*Intellectual Property*
LUT	-	*Lookup Table*
MAC	-	*Media Access Control*
MII	-	Media Independent Interface
MMCM	-	*Mixed-Mode Clock Manager*
PCI	-	Peripheral Component Interconnect
SFD	-	*Start Frame Delimiter*
SERDES	-	*Serializer-Deserializer*
WHS	-	*Worst Hold Slack*
WNS	-	*Worst Negative Slack*

1 INTRODUÇÃO

Dispositivos FPGA (do inglês *Field Programmable Gate Array)* são circuitos integrados que podem conter milhares de unidades lógicas programáveis idênticas. Esses dispositivos também provêm interfaces de alta velocidade, e diversos *Core-IPs* disponíveis junto à lógica programável (e.g. multiplicadores, blocos de memória, controladores de relógio, SERDES) reduzindo o custo dos produtos [XIL15]. A utilização de FPGAs cresce cada vez mais em sistemas digitais embarcados e a utilização dessa tecnologia requer que esses sistemas se comuniquem com o mundo externo. Dependendo da quantidade de dados que devem ser transferidos, diferentes tipos de protocolos de comunicação podem ser utilizados.

Uma interface de comunicação largamente utilizada no mercado de FPGAs é a *PCIe* (*Peripheral Component Interconnect Express)*, por se tratar de uma interface escalável e serial com elevadas taxas de transmissão. Porém, esta é uma interface complexa, pois requer *drivers* específicos para cada fabricante e baixa portabilidade, visto que é necessário um computador que possua um conector com esse padrão [PCI10].

O IEEE (*Institute of Electrical and Electronics Engineers*) desenvolveu e mantém o padrão Ethernet e, apesar de não ser o único protocolo de comunicação existente, tornou-se um protocolo amplamente utilizado devido a seu baixo custo de implementação e a alta flexibilidade para se adequar a diferentes topologias de rede [IEE15]. Devido à necessidade de aumentar as taxas de transmissão de dados, o padrão Ethernet recebeu melhorias ao longo dos anos. Assim, este padrão fornece a largura de banda necessária para a maioria das aplicações utilizadas em FPGAs [KHA11].

1.1 Motivação

No atual cenário de sistemas embarcados digitais, para que um produto se destaque é necessário que tenha alto desempenho e flexibilidade. Algumas interfaces de comunicação fornecem elevadas taxas de transmissão, porém o custo de implementação e a necessidade de utilização de um hardware específico dificultam atender esses requisitos.

Através do protocolo de comunicação Ethernet é possível realizar uma transmissão de dados eficiente e de alta velocidade entre o FPGA e dispositivos externos permitindo, por exemplo, uma troca de mensagens de forma remota entre um computador hospedeiro e o sistema prototipado.

A motivação deste trabalho é aprofundar os conhecimentos obtidos ao longo do curso, tais como sistemas de comunicação, desenvolvimento de fluxo de projeto

baseado em FPGA, linguagens de descrição de hardware e prototipação, necessários para o desenvolvimento de tal plataforma.

1.2 Objetivos

O objetivo deste Trabalho de Conclusão de Curso (TCC) é projetar um *Core-IP* que permita que um FPGA se comunique com dispositivos externos através de uma conexão Ethernet, a fim de que uma aplicação prototipada possa ser controlada por um *host* qualquer. Para esta finalidade foi utilizado um *core MAC* disponível no *site OpenCores* [OPC16].

Este projeto exige o domínio do *core MAC*, o estudo de dispositivos FPGA, domínio das ferramentas para desenvolvimento e conhecimentos técnicos de prototipação. A placa de prototipação utilizada nesse projeto foi a TB-7K-325T-IMG [INR13]. Esta placa possui um FPGA da família *Kintex-7*.

1.3 Arquitetura Conceitual

A Figura 1 apresenta a arquitetura conceitual do projeto desenvolvido. O projeto contém dois módulos principais: a interface *Core-IP* Ethernet e o hardware do usuário (*IP-user*) conectado à essa interface.

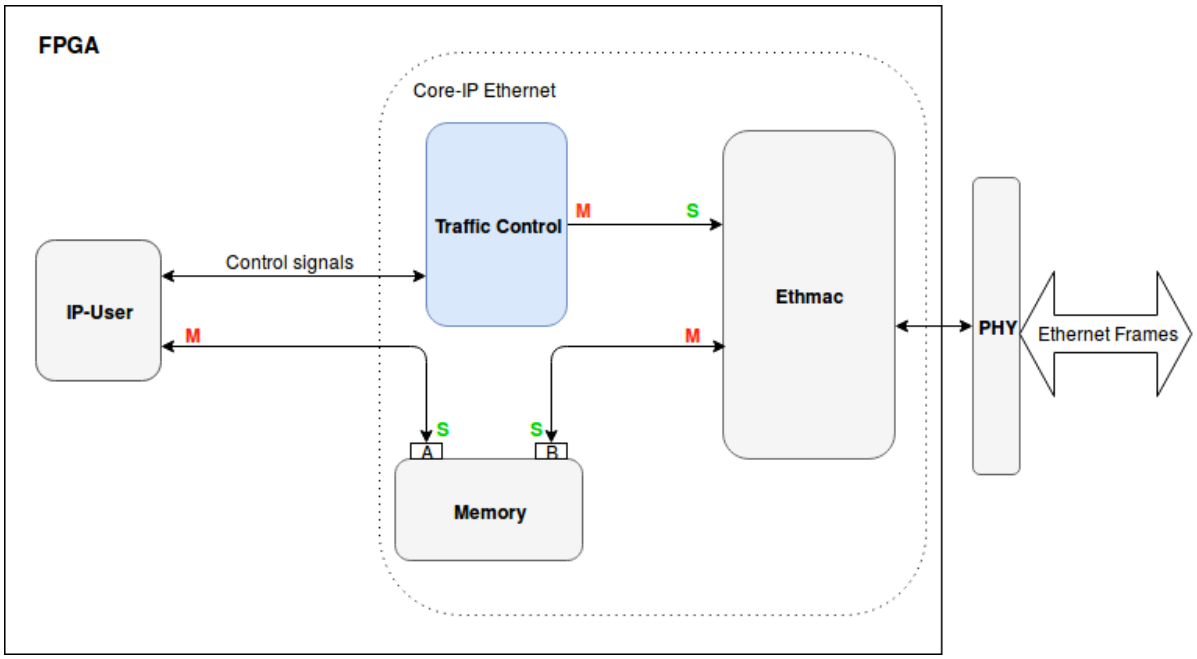

Figura 1- Arquitetura conceitual do projeto (M: *master*, S: *slave*).

A interface Core-IP Ethernet é responsável por capturar os *frames* Ethernet do PHY através do core denominado *Ethmac* [OPC16], processá-los e determinar qual o tipo de resposta o dispositivo deverá enviar. Em azul está representado o módulo

desenvolvido durante o trabalho e que é a principal contribuição, chamado *Traffic Control*, responsável pela configuração de registradores e *Buffers Descriptors* (BDs) do *MAC*, e também encarregado de controlar se existem *frames* a serem recebidos ou transmitidos. Uma memória BRAM *dual port* foi instanciada para realizar a leitura e escrita de dados, tanto do *MAC* quanto do IP do usuário. Esse IP do usuário consiste em uma aplicação que inverte os campos MAC *source* e MAC *destination* do *frame* Ethernet para realizar *loopback*, e que também pode realizar operações sobre os dados dos pacotes.

1.4 Estrutura do Documento

Este volume final de TCC é organizado como segue. O Capítulo 2 descreve o *Core-IP* utilizado como referência, suas características e funcionalidades. O Capítulo 3 apresenta uma descrição do módulo *Traffic Control*. O Capítulo 4 apresenta a descrição da implementação de um *testbench* e de uma aplicação, assim como a validação através de simulação e prototipação em FPGA. O Capítulo 5 conclui este documento.

2 CORE MAC

Neste Capítulo é apresentado o core Ethernet *MAC* (*Ethmac*) [OPC16] e suas principais características.

O *Ethmac* é um MAC (*Media Access Controller)* que se conecta ao módulo Ethernet *PHY* para comunicação com a rede de dados através do barramento MII, e ao barramento *wishbone* para a comunicação com o *hardware* do usuário. Este *core* possui alta flexibilidade para diversos tipos de aplicações que utilizam Ethernet e é capaz de operar em taxas de transferência de 10 e 100 *Mbps*.

2.1 Arquitetura do IP-core Ethmac

A Figura 2 apresenta a arquitetura do *Ethmac*, onde os principais módulos dessa arquitetura são: *Host Interface, Tx Ethernet MAC, Rx Ethernet MAC, MAC Control Module* e *MII management module.*

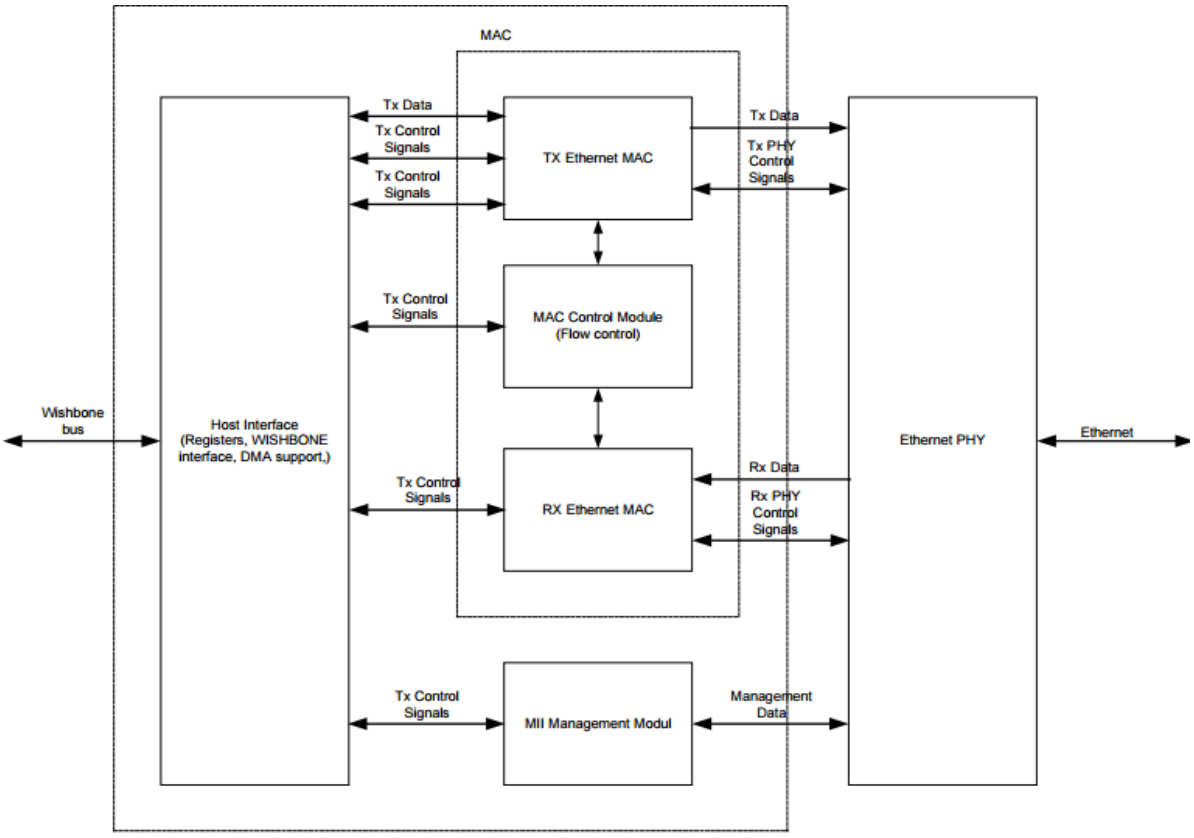

Figura 2- Arquitetura do core *Ethmac* [MOH02].

2.1.1 Host Interface

O módulo *Host Interface* possui internamente diversos módulos, descritos a seguir.

2.1.1.1 Wishbone Interface

Este é o módulo que possui o maior número de funcionalidade*s:*

- Realiza a interface entre o *core* e outros dispositivos (memória, *host)*, através de duas interfaces (mestre e escravo), descritas mais detalhadamente nas próximas subseções.
- Contém os *Buffers Descriptors* (BDs) na RAM interna. Os *BDs* são ponteiros para onde estão armazenados os *frames* Ethernet.
- Contém filas de recepção e transmissão de *frames*.
- Contém lógica de sincronização para sinais que utilizam diferentes domínios de relógio*.*
- Faz a leitura dos BDs de transmissão e inicia a interface *master* do *Wishbone,* preenche as FIFOs para realizar a transmissão dos dados, e por fim atualiza o *status* dos BDs.
- Faz a leitura dos BDs de recepção, monta as palavras com os *bytes* de entrada e escreve nas FIFOs de recepção. Por fim, escreve os dados na interface *master* do Wishbone.

2.1.1.2 Registers

Este módulo é responsável pela leitura dos valores dos registradores que determinam os modos de operação que serão configurados. Os registradores são alocados a partir do endereço 0x00 até o endereço 0x50 e são apresentados mais detalhadamente na seção 2.3.1.

2.1.1.3 Tx e Rx FIFO

Para transmissão e recepção de pacotes são utilizadas duas FIFOs, ambas com profundidade de 16 palavras.

Na transmissão, assim que o *buffer descriptor* de transmissão é lido (*status* e *pointer)* o dado é lido da interface *master* do *wishbone* e é armazenado na TX FIFO. A transmissão ocorre assim que a FIFO fica cheia. Enquanto há espaço para pelo menos uma palavra a leitura da FIFO continua.

Na recepção assim que o *buffer descriptor* de recepção é lido (*status* e *pointer*) e existe algum dado sendo armazenado na FIFO, uma escrita na interface *master* do

wishbone é realizada. A recepção do próximo *frame* ocorre após todo o dado ser escrito na memória (FIFO vazia).

2.1.2 Tx Ethernet MAC

Este módulo é responsável pelo controle da transmissão de dados. Ele obtém dados em formato de *bytes* do módulo *wishbone* que precisam ser transmitidos. Além disso, recebe sinais de controle informando o início do *frame (TxStartFrm),* o final do *frame (TxEndFrm)*, e informa ao wishbone que o módulo precisa de um novo byte do dado (*TxUsedData*). O módulo TX é responsável por informar ao *PHY* e ao wishbone sobre o estado de operação de transmissão através dos sinais:

- *MTxD* – É o dado em *nibbles (4 bits)* que será enviado para o mundo externo através do PHY Ethernet.
- *MTxEn* – Informa ao PHY que o dado *MTxD* é válido e a transmissão pode iniciar.
- *MTxErr* – Informa ao PHY que um erro ocorreu durante a transmissão.
- *TxDone* – Informa ao wishbone que a transmissão foi realizada com sucesso.
- *TxRetry* – Informa ao wishbone que a transmissão deve ser repetida. Isso acontece geralmente no modo *half-duplex* quando uma colisão é detectada.
- *TxAbort* – Informa ao wishbone que a transmissão foi abortada, por exemplo por um pacote muito grande, ou por ter um número de colisões muito elevado.

Além desse controle, o TX Ethmac possui alguns sinais internos: *Data_Crc, Enable_Crc* e *Initialize_Crc* responsáveis pela geração de CRC no *frame* a ser transmitido, e o sinal *WillTransmit* que informa ao receptor que a transmissão iniciará.

Este módulo possui quatro sub-módulos, sendo os mais importantes:

- eth_crc – Este sub-módulo calcula o CRC do frame que será enviado e o adiciona no final do *frame.* Também é utilizado na recepção.
- eth_txcounters – Este sub-módulo contém alguns contadores importantes na transmissão do *frame*, tais como contador de nibbles (*NibCnt)* e contador de bytes (ByteCnt), importantes no controle de que o *frame* será transmitido sem perder dados.
- eth_txstatem – Este módulo possui uma máquina de estados responsável pelo controle da transmissão do *frame*. Sob operação normal, o estado inicial da máquina é *idle*. Quando a interface *wishbone* seta o sinal *TxStartFrm* por dois ciclos de *clock,* junto com o primeiro byte a ser transmitido, a máquina vai para o estado *preamble* e o sinal MTxEn é setado para ‘1’, informando que a transmissão iniciará. O registrador *MTxD* recebe o valor do preâmbulo e o valor do SFD e a FSM vai para o estado Data0. O sinal *TxUsedData* é então setado para ‘1’ para informar ao *wishbone* que um novo *byte* pode ser enviado. A parte baixa do *byte*

é enviada em *nibbles* e a máquina vai para o estado Data1, para então enviar a parte alta. Este processo ocorre até que o sinal TxEndFrm (que indica o último *byte* do *frame)* seja ‘1’.

2.1.3 RX Ethernet MAC

Este módulo é responsável pelo controle da recepção de dados. O *PHY* externo recebe os dados seriais da camada física (cabo), monta-o em *nibbles* e o envia para o módulo de recepção através do sinal *MRxD[3:0]*, juntamente com o sinal de dado válido MRxDV. Em seguida, monta os dados em *bytes* e envia para a interface do *wishbone,* junto com alguns sinais que indicam o início e fim do *frame*. Esse módulo também é responsável por remover o preâmbulo e o CRC do pacote.

O módulo de recepção contém quatro sub-módulos:

- *eth_crc* – Este sub-módulo é utilizado para validar o CRC dos pacotes que estão sendo recebidos. Também é utilizado na geração do CRC para a transmissão.
- *eth_rxaddracheck* – Este sub-módulo faz o reconhecimento do endereço de destino e verifica quais pacotes devem ser recebidos ou não. O *Ethmac* inicialmente recebe todos os pacotes, independente do endereço de destino associado, porém o endereço de destino é verificado. A recepção dos *frames* acontece após a verificação de algumas condições:
    - Se o bit *r_Pro* é setado no registrador MODER (descrito na seção 2.3.1) indicando uma operação em modo promíscuo, todos os pacotes são recebidos, independente do seu endereço de destino.
    - Se *r_Bro* é setado no registrador MODER todos os *frames* contendo endereço de destino *broadcast* são descartados (*r_Pro* deve ser zero neste caso).
    - Quando o registrador MAC é configurado, todos os endereços recebidos são comparados a ele. O *frame* é recebido somente quando os valores dos endereços forem iguais. (*r_Pro* deve ser zero neste caso).

Se nenhuma das opções acima for satisfeita, o pacote é descartado e o sinal *RxAbort* é setado.

- eth_rxcounters – Este sub-módulo possui contadores para leitura do pacote, tais como contador do *interframe gap, bytes* recebidos e *delay* do CRC.
- eth_rxstatem – Este sub-módulo consiste em uma máquina de estados responsável pelo controle do recebimento do *frame.* Possui seis estados: *Idle, drop, preamble*, SFD (*standard frame delimiter), Data0* e *Data1* e, assim que o sinal MRxDV é setado indicando um dado válido no MRxD, a máquina de estados vai do estado *idle para* o estado *drop.* Se o dado recebido iniciar com o valor 0X5

a máquina vai para o estado *preamble* e depois para o estado SFD onde espera o *nibble* 0xD*. S*e iniciar com 0xD (opção de *frame* sem preâmbulo) vai para o estado de SFD diretamente, seguindo a norma. Quando o *interframe gap* possui um valor adequado, a máquina de estados vai para o estado Data0 onde a parte baixa do dado é recebida e depois para o estado Data1 onde a parte alta do dado é recebida. Isso ocorre até o sinal MRxDV baixar, indicando que o *frame* chegou ao fim.

2.1.4 Módulo MAC control

Este módulo é responsável pelo controle do fluxo de dados quando o *Core-IP* está no modo de operação full-duplex de 100 Mbps. Ele consiste em uma lógica de multiplexação e é dividido em dois sub-módulos: eth_transmitcontrol e eth_receivecontrol. Quando o dispositivo conectado à interface do wishbone não consegue processar todos os frames que estão sendo recebidos, ele solicita uma pausa para quem está enviando os frames. Os frames voltam a ser transmitidos após o sinal pause ser setado para zero ou ocorrer um timeout.

2.1.5 Módulo MII

Este é o módulo de gerência do barramento *MII*, possuindo três sub-módulos: *eth_clockgen, eth_shiftreg* e *eth_outputcontrol*. Ele é responsável por ler os registradores do módulo PHY e realizar a operação desejada (leitura ou escrita do *PHY).* A funcionalidade deste módulo é descrita com mais detalhes na próxima seção.

2.2 Interfaces de Comunicação

Esta Seção descreve as interfaces de comunicação com o hardware do usuário, barramento *wishbone*, e a interface com o *PHY*, barramento MII.

2.2.1 Barramento Wishbone

O padrão de barramento *wishbone* define uma arquitetura de comunicação *on-chip*, sendo um padrão de código aberto que propõe a especificação de um barramento síncrono de alta velocidade.

O barramento *wishbone* permite que um usuário personalize sinais para dar suporte a requisitos específicos da sua aplicação [PAS08]. Esse barramento é responsável por realizar a comunicação entre um dispositivo mestre e um dispositivo escravo utilizando uma topologia específica que atende aos requisitos do sistema. Na arquitetura de referência, a topologia utilizada é a ponto-a-ponto, representada na Figura 3. Nessa topologia, os dados podem trafegar em qualquer direção, porém o

dispositivo mestre requisita ou transmite os dados, enquanto o dispositivo escravo é sempre passivo.

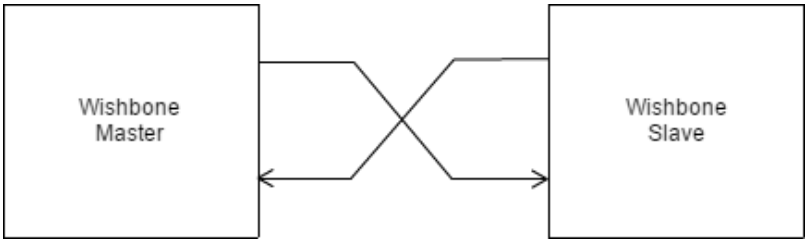

Figura 3- Topologia *Wishbone* ponto-a-ponto [PAS08].

A seguir são descritas as interfaces *wishbone master* e *wishbone slave* utilizadas neste trabalho.

2.2.1.1 Interface Wishbone Slave

Esta interface é responsável por realizar o acesso e configuração dos registradores e *buffers descriptors (BDs).*

2.2.1.2 Interface Wishbone Master

Esta interface é responsável por realizar o acesso ao endereço de memória em que os *buffers* (*frames ethernet*) estão armazenados.

O acesso ao dado da transmissão e recepção ocorre no mesmo barramento. Portanto, para este controle existe uma máquina de estados que multiplexa o acesso do TX e RX.

2.2.1.3 Fluxo de operação

Em ambas as interfaces citadas nas seções 2.2.1.2 e 2.2.1.3 o barramento *wishbone* possui os seguintes sinais:

- *Address* – Endereço de entrada da interface.
- Write enable – Indica um ciclo de escrita quando em nível lógico ‘1’ e leitura quando em nível lógico ‘0’.
- *CYC* – Indica que um dado válido está disponível no barramento.
- *STB* – Indica o início de uma transferência de dados válida.
- *Data out* – Dado de saída da interface.
- *Data in* – Dado de entrada da interface.
- *ACK* – Indica o final de um ciclo de operação normal.
- *Error* – Indica um erro no ciclo de operação.

- *Clock* e *Reset*

O wishbone opera da seguinte forma: o mestre habilita os sinais *CYC* e *STB* para iniciar uma transferência de dados. O escravo realiza uma leitura ou escrita e quando completa a transação ativa o sinal *ACK* (ou um erro). O mestre aguarda então o *ACK* ou sinal de erro para realizar uma leitura do dado disponível no *Data in* ou uma escrita no *Data out*.

A Figura 4 apresenta a tabela contendo as portas de entrada e saída e a descrição das mesmas, tanto da interface wishbone *slave* quanto da interface wishbone *master*.

Port	Width	Direction	Description
CLK_I	1	I	Clock Input
RST_I	1	I	Reset Input
ADDR_I	32	I	Address Input
DATA_I	32	I	Data Input
DATA_O	32	O	Data Output
SEL_I	4	I	Select Input Array Indicates which bytes are valid on the data bus. Whenever this signal is not 1111b during a valid access, the ERR_O is asserted.
WE_I	1	I	Write Input Indicates a Write Cycle when asserted high or a Read Cycle when asserted low.
STB_I	1	I	Strobe Input Indicates the beginning of a valid transfer cycle.
CYC_I	1	I	Cycle Input Indicates that a valid bus cycle is in progress.
ACK_O	1	O	Acknowledgment Output Indicates a normal Cycle termination.
ERR_O	1	O	Error Acknowledgment Output Indicates an abnormal cycle termination.
INTA_O	1	O	Interrupt Output A.
M_ADDR_O	32	O	Address Output
M_DATA_I	32	I	Data Input
M_DATA_O	32	O	Data Output
M_SEL_O	4	I	Select Output Array Indicates which bytes are valid on the data bus. Whenever this signal is not 1111b during a valid access, the ERR_I is asserted.
M_WE_O	1	O	Write Output Indicates a Write Cycle when asserted high or a Read Cycle when asserted low.
M_STB_O	1	O	Strobe Output Indicates the beginning of a valid transfer cycle.
M_CYC_O	1	O	Cycle Output Indicates that a valid bus cycle is in progress.
M_ACK_I	1	I	Acknowledgment Input Indicates a normal cycle termination.

Figura 4 - Portas de entrada e saída *wishbone slave* e *wishbone master* [MOH02].

2.2.2 Interface MII

A *MII (Media Independent Interface)* é uma interface que se conecta ao *PHY* externo ao FPGA. Essa interface é usada para configurar registradores e realizar leitura/escrita de/para o *PHY* através dos seguintes sinais:

- MTxClk – *Clock* de transmissão dos *nibbles. O* PHY gera um clock de 25 MHz para 100 Mbps e 2.5 MHz para 10 Mbps. É usado como referência de tempo para os demais sinais de transmissão.
- MTxD[3:0] – Dado transmitido em *nibbles*. Esse dado é sincronizado segundo a borda de subida do *MTxClk.* Quando o *sinal MTxEn* é setado o *PHY* aceita o *MTxD.*
- MTxEn – Sinal que indica que o dado *MTxD* é válido e a transmissão pode iniciar. É desabilitado na primeira borda de subida do *MTxClk* após o último *nibble* ser transmitido.
- MTxErr – Sinal que indica que ocorreu um erro na transmissão.
- MRxClk – *Clock* de recepção dos *nibbles. O* PHY gera um clock de 25 MHz para 100 Mbps e 2.5 MHz para 10 Mbps. É usado como referência de tempo para os demais sinais de recepção.
- MRxDV – Sinal que indica que o *nibble* atribuído ao MRxD é um dado válido. Este sinal é setado na borda de subida do MRxClk no início do preâmbulo até o último *nibble* recebido.
- MRxD[3:0] – Dado recebido em *nibbles*. Esse dado é sincronizado segundo a borda de subida do *MRxClk.* Quando o *sinal MRxDV* é setado o *PHY* envia os *nibbles* para o RX Ethmac. Para que ocorra a recepção, devem ser enviados o número correto de *bytes* de preâmbulo e SFD, segundo o padrão *IEEE* 802.3.
- MRxErr – Sinal que informa ao RX Ethmac que ocorreu um erro na transmissão do dado. Esse sinal é síncrono ao MRxClk.

A interface *MII* consiste ainda em dois sinais lógicos: sinal de *clock* (*MDC*) e sinal de dado (*MDIO*). Para o sinal *MDIO* tornar-se bidirecional e conectar-se ao PHY ele depende de um sinal de entrada Mdi, um sinal de saída Mdo e um sinal de habilitação *MdoEn*.

Para se realizar uma leitura ou escrita de dados do *PHY* os seguintes registradores devem ser configurados:

- Registrador *MIIMODER*
- Registrador *MIIADDRESS*
- Registrador *MIICOMMAND*
- Registrador *MIITX_DATA*

- Registrador *MIIRX_DATA*
- Registrador *MII_STATUS*

A operação do barramento *MII* inicia com os registradores *MIIMODER* e *MIIADDRESS* sendo configurados. No *MIIADDRESS* os endereços do *PHY* e do registrador associado devem ser indicados. *Após* o dado ser escrito no registrador *MIITX_DATA,* quando "*read status"* ou "*scan status"* estiverem habilitados no registrador *MIICOMMAND*, ou seja, se for requisitada uma leitura, o valor recebido pelo *PHY* pode ser lido do registrador *MIIRX_DATA.*

2.3 Operação do core Ethmac

O Ethmac pode operar em modo *half-duplex* ou *full-duplex* e é baseado no método de acesso *CSMA/CD* (*Carrier Sense Multiple Access* / *Collision detection*).

O *Ethmac* possui dois conjuntos de registradores internos que podem ser configurados. Os endereços 0x00 a 0x50 são destinados aos registradores de configuração geral, *flags* de interrupção, modo de operação e endereço de *MAC* interno (utilizado para filtrar pacotes destinados a outros endereços). Os endereços 0x400 a 0x7FF são reservados para *Buffers Descriptors* (BDs) de transmissão e recepção.

Os *BDs* possuem 64 bits que são divididos em duas partes. Os primeiros 32 bits contêm as informações (tamanho do *frame*, status, etc.) sobre um *frame* Ethernet que foi gravado na memória ou que será transmitido. Os últimos 32 bits são ponteiros para a localização de onde os *frames* serão alocados na memória. A seguir são descritas as características principais destes registradores internos e suas funções.

2.3.1 Registradores

Para realizar a transmissão de dados, alguns registradores devem ser configurados. A Figura 5 e a Figura 6 apresentam a descrição desses registradores, onde o campo *Address* indica o endereço relativo ao registrador em hexadecimal e *Width* indica o número de bits que o registrador possui. Todos os registradores possuem modo de acesso de leitura e escrita (RW).

Name	Address	Width	Access	Description
MIITX_DATA	0x34	32	RW	MII Transmit Data The data to be transmitted to the PHY
MIIRX_DATA	0x38	32	RW	MII Receive Data The data received from the PHY
MIISTATUS	0x3C	32	RW	MII Status Register
MAC_ADDR0	0x40	32	RW	MAC Individual Address0 The LSB four bytes of the individual address are written to this register.
MAC_ADDR1	0x44	32	RW	MAC Individual Address1 The MSB two bytes of the individual address are written to this register.
ETH_HASH0_ADR	0x48	32	RW	HASH0 Register
ETH_HASH1_ADR	0x4C	32	RW	HASH1 Register
ETH_TXCTRL	0x50	32	RW	Transmit Control Register

Figura 5- Descrição dos Registradores 1 [MOH02].

Name	Address	Width	Access	Description
MODER	0x00	32	RW	Mode Register
INT_SOURCE	0x04	32	RW	Interrupt Source Register
INT_MASK	0x08	32	RW	Interrupt Mask Register
IPGT	0x0C	32	RW	Back to Back Inter Packet Gap Register
IPGR1	0x10	32	RW	Non Back to Back Inter Packet Gap Register 1
IPGR2	0x14	32	RW	Non Back to Back Inter Packet Gap Register 2
PACKETLEN	0x18	32	RW	Packet Length (minimum and maximum) Register
COLLCONF	0x1C	32	RW	Collision and Retry Configuration
TX_BD_NUM	0x20	32	RW	Transmit Buffer Descriptor Number
CTRLMODER	0x24	32	RW	Control Module Mode Register
MIIMODER	0x28	32	RW	MII Mode Register
MIICOMMAND	0x2C	32	RW	MII Commend Register
MIIADDRESS	0x30	32	RW	MII Address Register Contains the PHY address and the register within the PHY address

Figura 6 – Descrição dos registradores 2 [MOH02].

A Tabela 1 descreve o principal registrador, MODER, responsável pela configuração dos modos de operação do *ethmac.*

Tabela 1- Registrador MODER.

Bit #	Acesso	Descrição
31-17	RW	Reservado
16	RW	RECSMALL – Receive Small Packets 0 = Pacotes menores que MINFL são ignorados. 1 = Pacotes menores que MINFL são aceitos.
15	RW	PAD – Padding Enable 0 = Não adiciona pads em *frames* pequenos. 1 = Adiciona *padding* em *frames* pequenos até o valor de MINFL.
14	RW	HUGEN – Huge Packets Enable 0 = O tamanho máximo do *frame* é MAXFL. Os *bytes* adicionais são descartados. 1 = *Frames* maiores que 64 KB são transmitidos.
13	RW	CRCEN – CRC Enable 0 = Tx MAC não adiciona CRC no final do *frame*. 1 = TX MAC adiciona CRC em todos os *frames.*
12	RW	DLYCRCEN – Delayed CRC Enable 0 = Operação normal (CRC inicia após SFD). 1 = Calculo do CRC inicia 4 *bytes* após o SFD.
11	RW	Reservado
10	RW	FULLD – Full Duplex 0 = Modo half-duplex. 1 = Modo full-duplex.
9	RW	EXDFREN – Excess Defer Enabled 0 = Quando o limite de espera for excessivo o pacote é abortado. 1 = MAC espera o portador indefinidamente.
8	RW	NOBCKOF – No Backoff 0 = Operação normal (Utiliza um algoritmo específico de *backoff).* 1= TX MAC começa a retransmitir imediatamente após uma colisão.
7	RW	LOOPBACK – Loop Back 0 = Operação normal. 1 = TX é espelhado para RX.
6	RW	IFG – *Interframe Gap* 0 = Operação normal (valor mínimo é requerido para o *frame* ser aceito). 1= Todos *frames* são recebidos independente do IFG.
5	RW	PRO – Promiscuous 0 = Confere o endereço de destino dos *frames.* *1 =* Aceita o *frame* independente do endereço.
4	RW	IAM – Individual Address Mode 0 = Operação normal (endereço físico é conferido quando o *frame* é recebido). 1 = É utilizada uma tabela para conferir os endereços recebidos.
3	RW	BRO – Broadcast Address 0 = Recebe todos os *frames* contendo o endereço *broadcast.* 1 = Rejeita todos os *frames* contendo o endereço *broadcast.*
2	RW	NOPRE – No preamble 0 = Operação normal (7 *bytes* de preâmbulo). 1 = Não envia preâmbulo.
1	RW	TXEN – Transmit Enable 0 = Desabilita transmissão / 1 = Habilita transmissão.
0	RW	RXEN – Receive Enable 0 = Desabilita Recepção / 1 = Habilita recepção.

2.3.2 Buffers Descriptors de Transmissão

Os *Buffers Descriptors* de transmissão, Figura 7, começam no endereço 0x400 e alguns devem ser configuradas para que a transmissão ocorra. Portanto, assim que um BD é setado como *Ready (*bit 15) a transmissão de pacotes pode iniciar.

ADDR = Offset + 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LEN															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD	IRQ	WR	PAD	CRC	Reserved		UR	RTRY[3:0]				RL	LC	DF	CS

ADDR = Offset + 4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXPNT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXPNT															

Figura 7- Buffer Descriptor de Transmissão [MOH02].

Os bits utilizados neste trabalho para que ocorra uma transmissão de *frames* são:

- LEN – Tamanho do *frame* que será transmitido no BD correspondente.
- TXPNT – Ponteiro onde o *frame* é armazenado.
- RD (Ready)
    - ‘0’: O *buffer* não tem nenhum dado associado, então pode ser escrito.
    - ‘1’: O *buffer* está pronto para transmissão e não deve ser modificado.
- WR (Wrap)
    - ‘0’: Indica que este não é o último BD.
    - ‘1’: Indica que este é o último BD.

2.3.3 Buffers Descriptors de Recepção

Os *Buffers Descriptors* de recepção, Figura 8, começam no endereço 0x600. De modo semelhante à transmissão, quando o BD de recepção é marcado como *Empty, o Ethmac* coloca o primeiro dado válido recebido no endereço de memória associado.

Os bits utilizados neste trabalho para que ocorra uma recepção de *frames* são:

- LEN – Tamanho do frame que será recebido no BD correspondente.
- RXPNT – Ponteiro onde o *frame* é armazenado.

- E (Empty)
    - ‘0’: O *buffer* está preenchido com dados ou ocorreu algum erro. Desta forma ele não deverá ser utilizado.
    - ‘1’: O *buffer* está disponível pronto para receber dados.
- WR (Wrap)
    - ‘0’: Indica que este não é o último BD.
    - ‘1’: Indica que este é o último BD.

ADDR = Offset + 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LEN															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	IRQ	WR	Reserved				CF	M	OR	IS	DN	TL	SF	CRC	LC

ADDR = Offset + 4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RXPNT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXPNT															

Figura 8- Buffer Descriptor de Recepção [MOH02].

2.3.4 Recepção de Dados

Para realizar a recepção dos *frames*, o *Core-IP* deve seguir o seguinte protocolo:

- Setar o BD de recepção para ser associado com o pacote a ser recebido e marcá-lo como *Empty*.
- Habilitar a recepção Ethernet setando o bit *RXEN* do registrador *MODER* para ‘1'.

O *Ethmac* lê o BD de recepção, e se este está marcado como *empty*, ele começa a receber os *frames*. Depois que o *frame* inteiro é recebido e armazenado na memória, o status do BD é atualizado. Então, o ponteiro é incrementado para o próximo BD e se este é marcado como *empty* a recepção continua. Caso contrário, a operação é encerrada.

2.3.5 Transmissão de Dados

Para transmitir o primeiro frame, o *core* deve:

- Armazenar o *frame* na memória.
- Associar o BD de transmissão no Ethmac com o *frame* escrito na memória.

- Habilitar a transmissão setando o bit *TXEN* do registrador *MODER* para ‘1’.

Assim que o *Ethmac* é habilitado, ele começa a fazer a leitura do BD de transmissão. Quando o BD é marcado como *ready,* o *Ethmac* lê o ponteiro da memória que armazena os dados associados e inicia a leitura de dados para uma FIFO interna. Quando a FIFO fica cheia a transmissão começa.

Assim que a transmissão é encerrada, o BD *status* é atualizado e podem ocorrer dois eventos, dependendo do bit *WRAP*:

- Se o *WRAP* é ‘0’, o endereço do BD é incrementado e o próximo BD é analisado; se marcado como *ready* a operação se repete.
- Se o WRAP é ‘1’, o endereço base do BD é carregado novamente. Quando for setado como *ready* a transmissão é reiniciada.

3 TRAFFIC CONTROL

Este Capítulo apresenta o módulo *Traffic Control,* responsável pela escrita de dados nos registradores e *Buffers Descriptors (BDs).* Esse módulo também possui uma interface com o IP-User, a fim de requisitar operações de envio e recebimento de *frames.* A Figura 9 detalha a interface do *Traffic Control* com os módulos *Ethmac* e IP-User. A seguir são descritas essas duas interfaces.

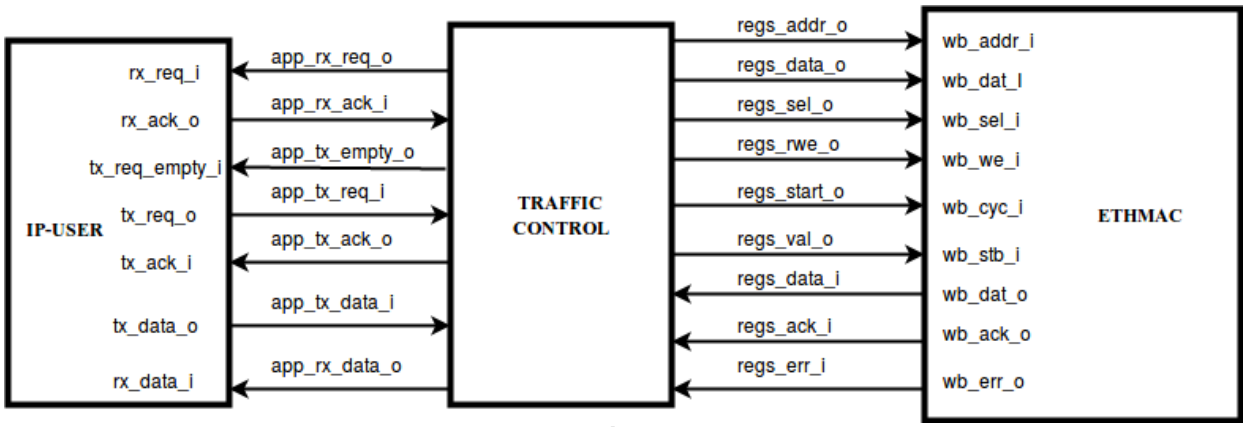

Figura 9 - Interfaces do módulo Traffic Control.

3.1 Interface com a aplicação do usuário

O *Traffic Control* possui uma interface com o IP-User, que pode ser uma aplicação qualquer. Essa interface com a aplicação possui os sinais:

- *app_rx_req_o* – Este sinal de saída informa à aplicação que existem *frames* na memória prontos para serem tratados.
- *app_rx_ack_i* – Este sinal de entrada indica que a aplicação já fez uma leitura e tratou o dado que estava armazenado na memória.
- *app_tx_empty_o* – Este sinal de saída informa à aplicação que uma transmissão pode ser realizada.
- *app_tx_req_i* – Este sinal de entrada indica que a aplicação deseja enviar um dado para o mundo externo.
- *app_tx_ack_o* – Este sinal de saída indica que a aplicação pode escrever os dados na memória.
- *app_tx_data_i* – Este sinal recebe um dado da aplicação contendo informações sobre o pacote, tais como tamanho, *ready,* crc, pad, etc.
- *app_rx_data_o* – Envia à aplicação informações sobre o pacote que foi recebido.

3.2 Interface com o *core MAC*

O *Traffic Control* possui uma interface com o Core-IP através da interface *wishbone slave,* que realiza a escrita dos registradores e BDs. Essa interface possui os seguintes sinais:

- regs_addr_o
- regs_data_i e regs_data_o
- regs_err_i e regs_ack_i
- regs_sel_o e regs_rwe_o
- regs_start_o e regs_val_o

Estes sinais possuem o mesmo comportamento dos sinais descritos na seção do barramento *wishbone.* Os mais importantes são *regs_addr_o* e *regs_data_o* que indicam endereço e valor que são atribuídos aos registradores e BDs.

No tratamento dos endereços deve ser levado em consideração que o *core Ethmac* possui uma lógica interna de mascaramento, visto que são necessários mais de dez bits para representar alguns endereços e o *Ethmac* possui um barramento de endereços de apenas dez bits. Portanto, esses endereços foram deslocados previamente antes de serem atribuídos à interface. A Figura 10 apresenta a declaração dos endereços dos registradores e dos BDs que são utilizados.

```
-- REGISTERS ADDRESS
constant addr_moder      : std_logic_vector(9 downto 0) := "00" & x"00"; --0x00
constant addr_ipgt       : std_logic_vector(9 downto 0) := "00" & x"03"; --0x0C
constant addr_mac_addr0  : std_logic_vector(9 downto 0) := "00" & x"10"; --0x40
constant addr_mac_addr1  : std_logic_vector(9 downto 0) := "00" & x"11"; --0x44
constant addr_tx_bd0     : std_logic_vector(9 downto 0) := "01" & x"00"; --0x400
constant addr_rx_bd0     : std_logic_vector(9 downto 0) := "01" & x"80"; --0x600
```

Figura 10- Endereço dos registradores e BDs.

3.3 Máquina de estados de controle

O controle de recepção e transmissão de *frames* é feito através de uma máquina de estados finita (FSM) que é responsável pela configuração dos registradores e *Buffers Descriptors* do *MAC*, através da interface *slave* do *wishbone*. Também é responsável por fazer *pooling* nos *BDs de RX,* perguntando se existem *frames* armazenados para que possam ser lidos da memória pela aplicação, ou nos BDs de TX, perguntando se existe algum BD livre para armazenar o dado que será enviado. A FSM possui cinco estados. A Figura 11 ilustra a máquina de estados.

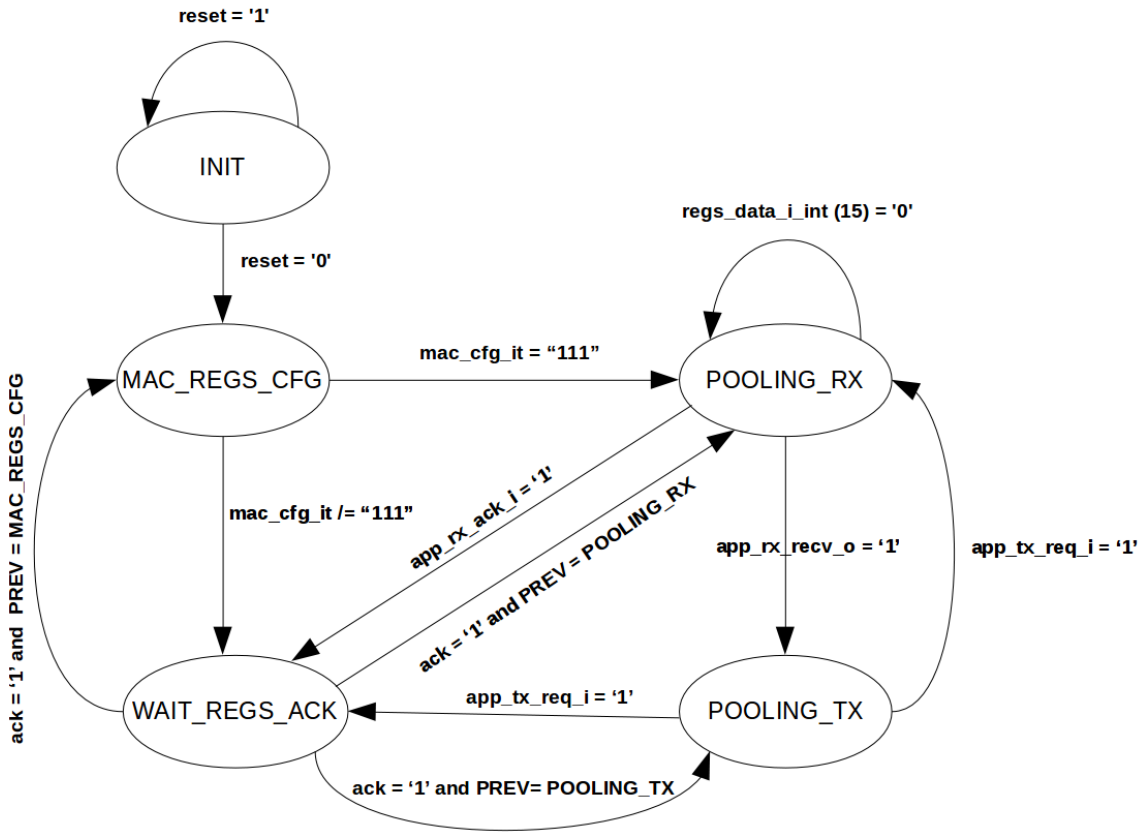

Figura 11- FSM Traffic Control.

- INIT – Estado inicial quando reset é igual a ‘1’.
- MAC_REGS_CFG – Nesse estado é feita a configuração dos registradores e BDs, onde os endereços e valores desses registradores são atribuídos à saída de um multiplexador. Nesse estado são atribuídos os endereços dos BDs de recepção e transmissão e os *BDs* são setados como *Empty*.

  Os registradores configurados neste estado são:

  - MODER – É configurado de acordo com o tipo de pacote que se espera receber.
  - IPGT – É modificado para operar como *full-duplex* (valor 0x00000015).
  - ADDR0 – Correspondente à parte alta do endereço *MAC* destinado à placa.
  - ADDR1 – Correspondente à parte baixa do endereço *MAC* destinado à placa.

- WAIT_REGS_ACK – Sempre que ocorre uma escrita ou leitura nos registradores a FSM vai para este estado, para garantir que uma nova escrita não seja realizada enquanto não é recebido um *ack* do *wishbone*. Quando o *ack* é recebido a FSM retorna para o estado anterior.
- POOLING_RX – Este estado verifica se existe algum dado que deve ser lido (bit 15 do RX BD igual a ‘0’). Se há algum dado armazenado em memória, ele informa à aplicação através do sinal app_rx_req_o que o dado pode ser lido da memória, e fica aguardando o sinal app_rx_ack_i, indicando que o dado foi tratado pela aplicação e o BD pode ser liberado.
- POOLING_TX – Este estado verifica se existe algum BD disponível para realizar um envio. Enquanto não existem requisições de envio por parte da aplicação, o sinal *app_tx_ready_o* recebe o valor do bit 15 do TX BD. Desta forma, enquanto este sinal estiver em zero o *traffic control* permite que a aplicação requisite um novo envio. Quando a aplicação envia o sinal de requisição app_tx_req_i juntamente com as informações sobre o pacote (*length, pad, crc, ready),* o *traffic control* escreve naquele BD o valor do *app_tx_data_i* e retorna um *app_tx_ack_o* para a aplicação.

4 IMPLEMENTAÇÃO E VALIDAÇÃO

Nos Capítulos anteriores foram apresentados os módulos necessários para o desenvolvimento da interface *Core-IP Ethernet*. Neste Capítulo são apresentadas as demais etapas de desenvolvimento. A validação do projeto é feita através de simulação, analisando o comportamento lógico dos sinais de referência. O sistema também é parcialmente validado através de prototipação, seguindo o fluxo de projeto de FPGAs, através da ferramenta *Chipscope* da Xilinx.

4.1 Desenvolvimento do Testbench para Simulação

Para realizar a simulação da interface foi desenvolvido um *testbench*. Esse *testbench* gera um pacote que é enviado para o Ethmac através de um PHY simulado e age sobre os sinais dos demais módulos instanciados, os quais incluem a memória, *Traffic Control* e *uma aplicação do usuário*.

4.1.1 Interface entre os módulos

A Figura 12 apresenta os módulos instanciados no *testbench* para simular a recepção e transmissão de pacotes.

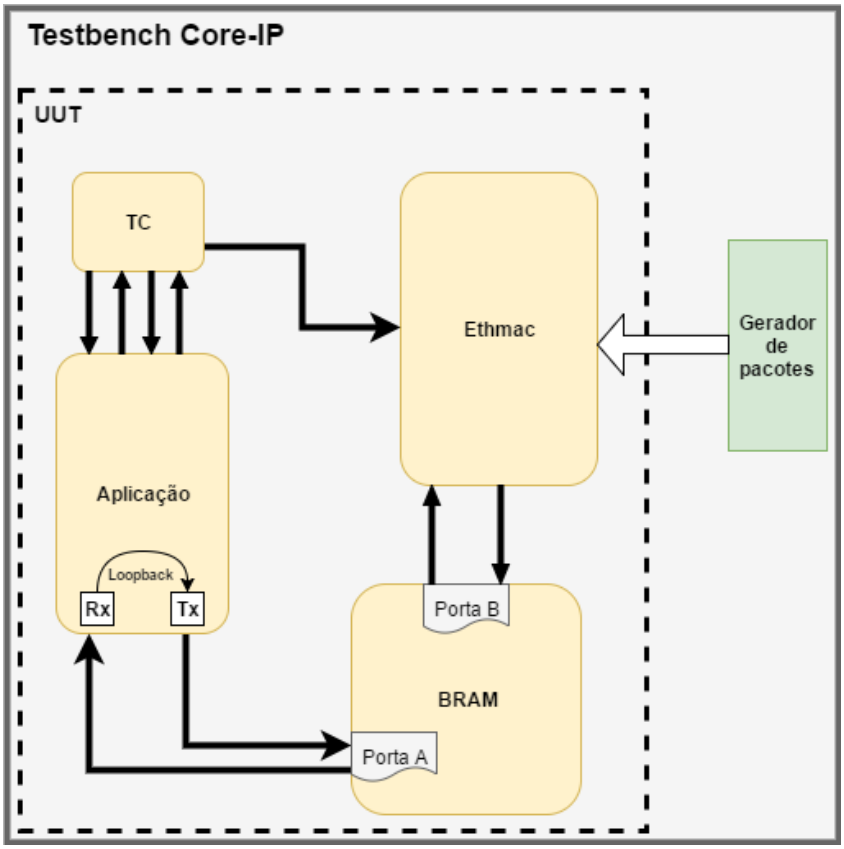

Figura 12- Testbench para a interface *Core-IP.*

Os pacotes são enviados para o Ethmac assim que o *Traffic Control* termina a configuração do mesmo através de um sinal que indica o fim da configuração. O *frame* é armazenado na memória a partir da porta B, no endereço destinado à recepção e é lido desse mesmo endereço a partir da porta A. Assim que o dado é tratado e armazenado no endereço destinado à transmissão e a aplicação requisita um envio, o Ethmac realiza a leitura daquele endereço a partir da porta B. O dado é então transmitido através do PHY simulado.

4.1.2 Gerador de pacotes

O Ethmac utiliza o padrão IEEE 802.3. Nesse padrão o pacote Ethernet possui campos para realizar a comunicação entre os dispositivos conectados em uma mesma rede. O MAC tem por objetivo determinar o inicio e fim dos *frames* que serão enviados às camadas superiores [IEE15]. Desta forma, o formato do pacote gerado deve ser o apresentado na Figura 13.

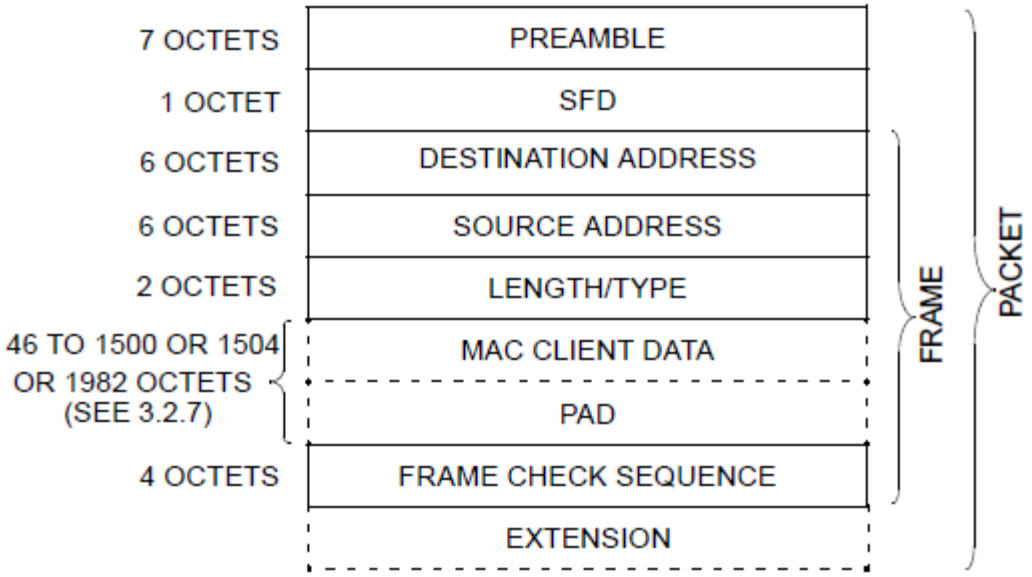

Figura 13- Formato do Pacote Ethernet [IEE15].

Os campos representam:

- Preamble – Possui 7 bytes contendo o valor 0x55 e indica o início de uma transmissão (sequência de ‘1’s e ‘0’s).
- *Start Frame Delimiter (SFD)* – Possui 1 byte contendo o valor 0x5D. Ele indica que o *frame* que será enviado às camadas superiores irá iniciar.
- *Destination Address* – Endereço MAC de 6 bytes utilizado na identificação do dispositivo de destino.

- *Source Address* – Endereço MAC de 6 bytes utilizado na identificação do dispositivo fonte.
- Lenght/type – Quando o pacote possui um tamanho inferior a 1500 este identifica o tamanho do pacote. Para tamanhos superiores a 1536 este campo indica o tipo de protocolo utilizado. Ocupa 2 bytes.
- MAC Client Data – Este campo possui tamanho que pode variar de 46 a 1500 e carrega os dados úteis.
- *Frame Check Sequence (FCS)* – Este campo possui 4 *bytes* e armazena o CRC calculado através dos dados do pacote.

Na Figura 14 é apresentada a simulação da geração de um pacote no *testbench.* Nesse trecho da simulação pode-se visualizar a geração em *nibbles* de um pacote com tamanho de 46 *bytes (0x2E).* O campo destacado representa o cabeçalho contendo os endereços MAC de destino e origem, e o campo indicando o tamanho do pacote, onde o MAC de destino é o mesmo que foi configurado no respectivo registrador. O restante é preenchido com dados aleatórios. O sinal *mrxdv_pad_i* é acionado no inicio do preâmbulo, indicando que esse é um dado válido a ser recebido.

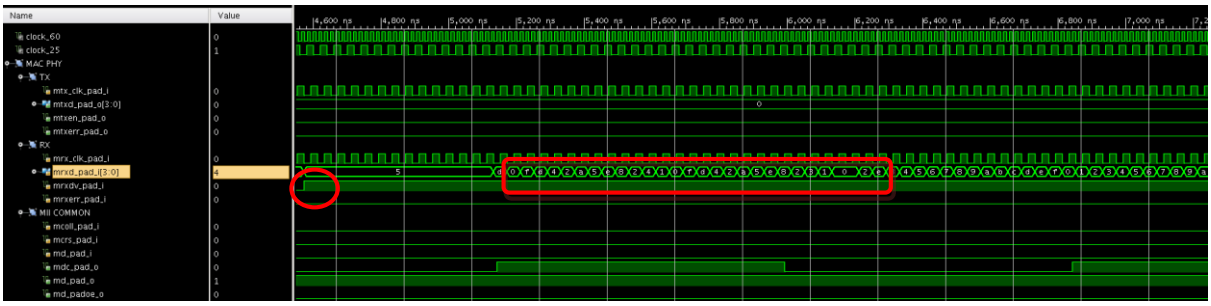

Figura 14- Recepção do pacote no PHY.

4.2 Simulação do Core-IP Ethernet

Para validar a interface desenvolvida neste trabalho, vários aspectos já destacados anteriormente devem ser levados em consideração: configuração dos registradores, configuração de endereços e conteúdo dos BDs, recepção dos *frames,* tratamento do dado a partir da aplicação e transmissão dos *frames* para o mundo externo.

A Figura 15 apresenta uma visão geral da inicialização da interface a partir do *Traffic Control*. No destaque 1 foram configurados os registradores importantes para que ocorra a recepção e transmissão de pacotes. No destaque 2 são setados os endereços dos BDs de recepção e transmissão. No destaque 3 é escrito o conteúdo nos BDs para iniciar uma recepção. No destaque 4 é iniciado o pooling de RX e TX.

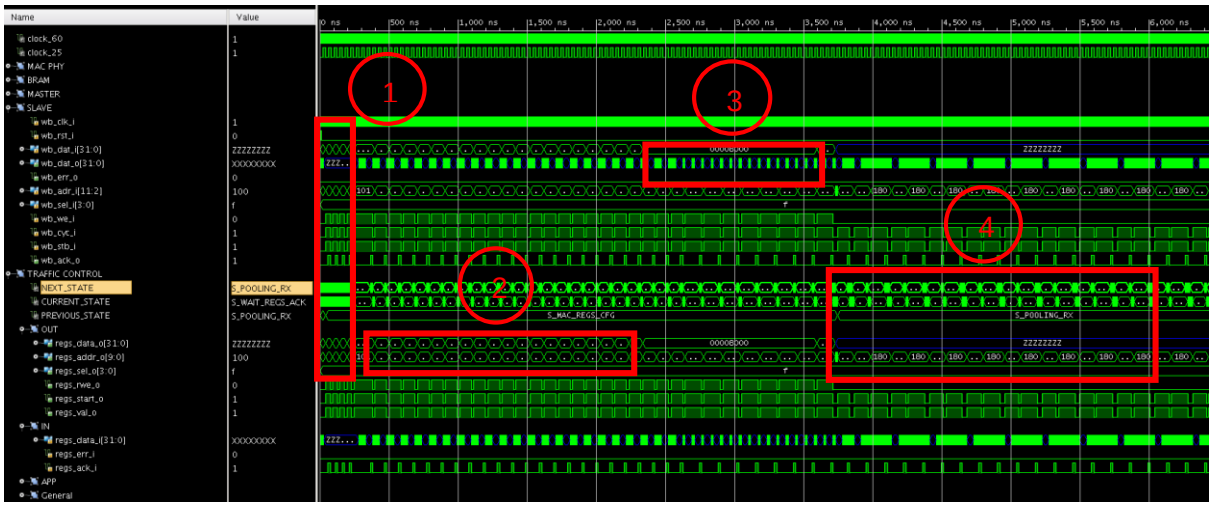

Figura 15- Visão geral da configuração de registradores e BDs.

As seções 4.2.1 a 4.2.5 apresentam as simulações da configuração, recepção e transmissão de *frames.*

4.2.1 Configuração de Registradores

A Figura 16 ilustra a configuração detalhada dos registradores, onde através de um multiplexador os endereços e dados são atribuídos aos sinais de saída *regs_addr_o_int* e *regs_data_o_int*, respectivamente. Para realizar uma operação de escrita, os sinais *regs_start_o_int*, *regs_val_o_int* e *regs_rwe_o_int* devem estar habilitados. Os destaques na forma de onda correspondem aos eventos:

- 1: representa a configuração do registrador *MAC_ADDR0* (endereço 0x40, que por mascaramento torna-se 0x11), correspondente à parte alta do endereço *MAC* destinado à placa (0x0000F04D).
- 2: representa a configuração do registrador *IPGT* (endereço 0x0C). Esse registrador é importante para que ocorra uma transmissão *full-duplex (*0X15).
- 3: representa a configuração do registrador *MAC_ADDR1* (endereço 0x44), correspondente à parte baixa do endereço *MAC* destinado à placa (0XA2E52814).
- 4: representa a configuração do registrador *MODER* (endereço 0x00). Além do valor *default* 0x0000A000, são setados os bits de *TxEn, RxEn, RECSMALL* (para receber pacotes pequenos) e operação *full-duplex;* desta maneira o valor final é 0x0001A403.
- 5: interface wishbone *slave* sendo configurada com os valores corretos.

Os registradores restantes permanecem com seus valores *default*.

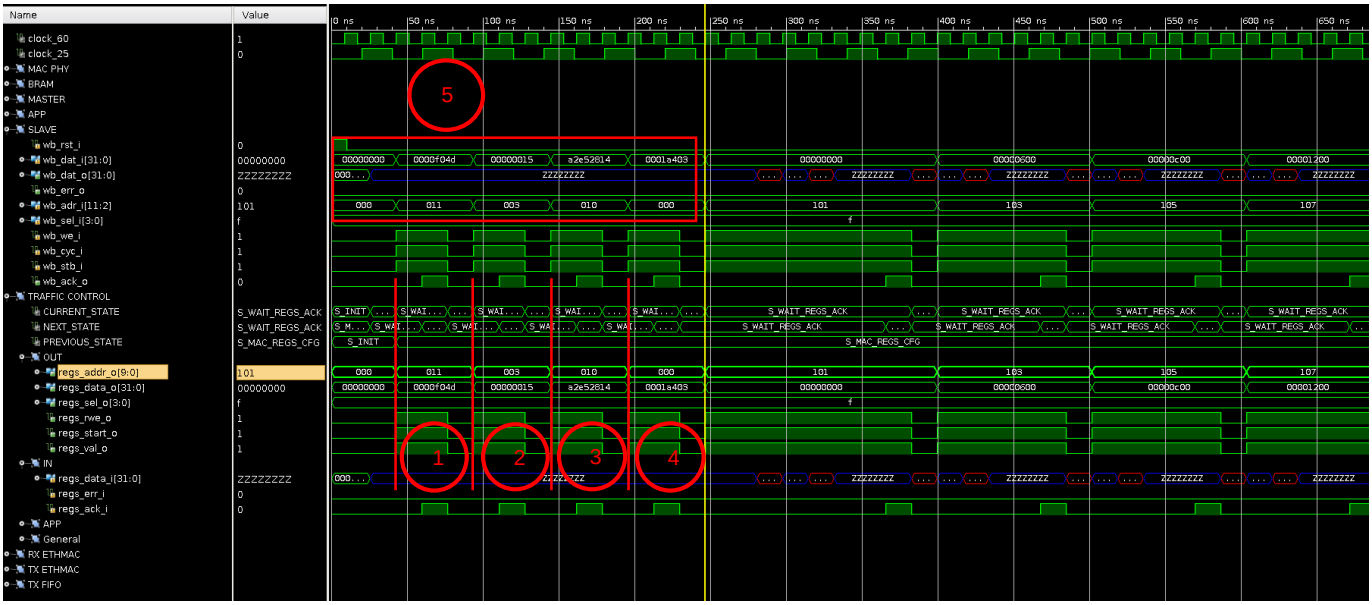

Figura 16- Configuração dos registradores.

4.2.2 Configuração dos Buffers Descriptors

Os *Buffers Descriptors* devem ser configurados associando-se com seus respectivos endereços de memória. A Figura 17 apresenta a escrita nos ponteiros dos BDs sendo realizada. Os BDs de transmissão são alocados em memória a partir do endereço 0x0000, enquanto os de recepção são alocados a partir do endereço 0x3C00. Na Figura são destacados os momentos em que os BDs de transmissão são associados a estes endereços. Inicialmente são setados os valores 0X0000 e 0x3C00, e os próximos endereços recebem os valores 0x0600 e 0x4200, portanto, para cada *BD* é somado o valor 1536 (0x600) que representa o tamanho máximo dos *frames*. O valor 0x101 representa o endereço base para o ponteiro de dados do *buffer.*

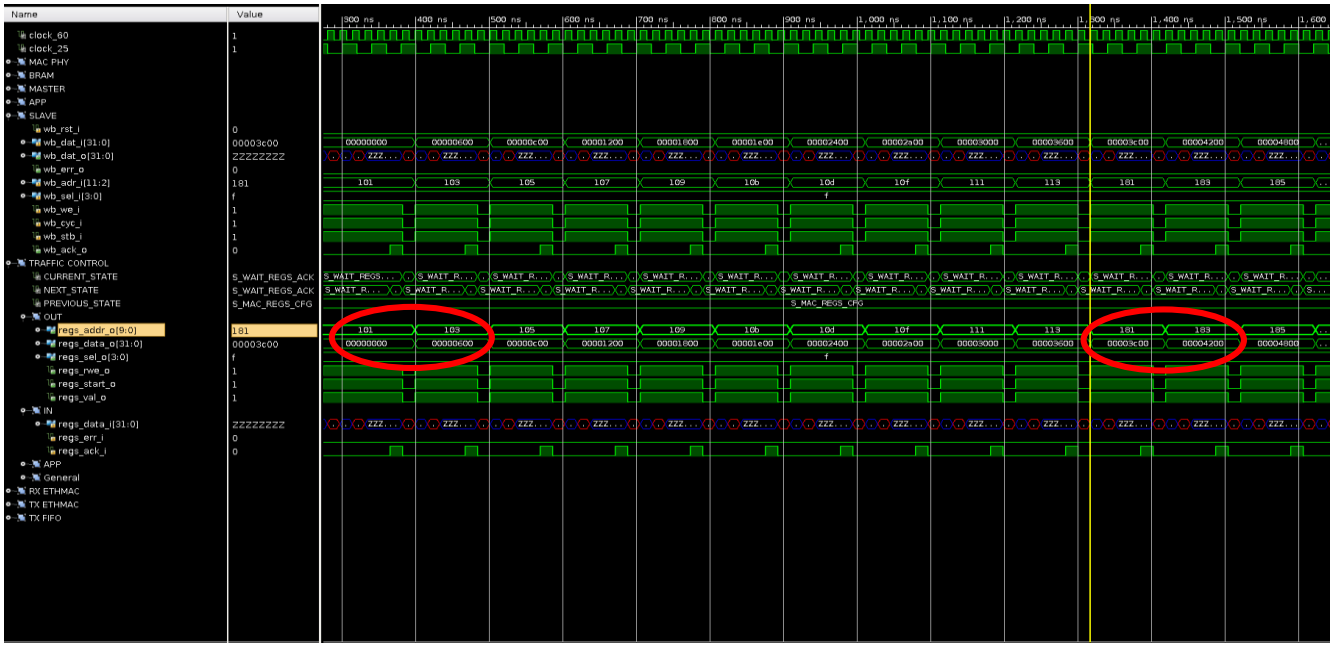

Figura 17- Inicialização dos endereços dos BDs

A Figura 18 apresenta a inicialização dos *BDs* de recepção no endereço 0x100 correspondente aos primeiros 32 bits de dados. Cada *BD* recebe o valor 0x00008000, indicando que esses BDs estão disponíveis para receber dados.

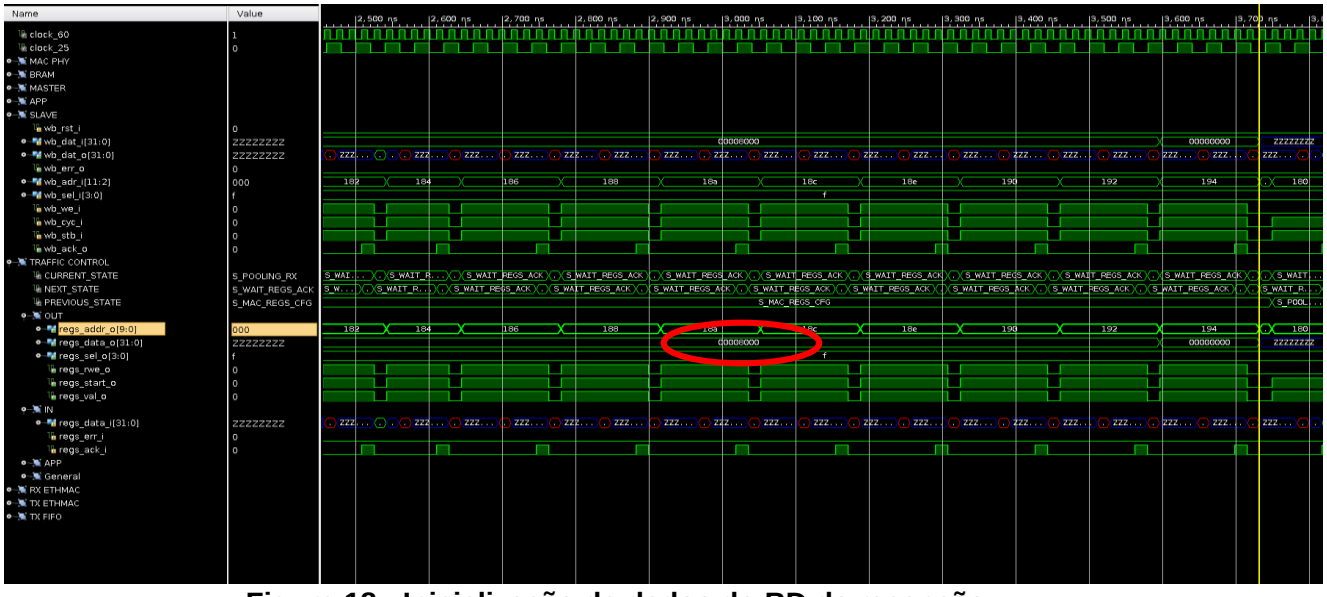

Figura 18 - Inicialização de dados do BD de recepção.

4.2.3 Recepção de *frames*

Após a configuração, o Traffic Control inicia o pooling nos BDs de recepção e transmissão para saber se existe algum dado disponível neles ou se existe espaço alocado para realizar uma transmissão.

A Figura 19 apresenta os estados S_POOLING_RX e S_POOLING_TX sendo alternados enquanto o valor de *regs_data_i* é diferente de zero (1), ou seja, enquanto não há dados no BD. A Figura destaca ainda o instante em que um pacote chega (bit 15 do BD de recepção igual à zero) e o dado pode ser tratado (2). Após o dado ser recebido o endereço base do BD é incrementado para uma nova recepção e o conteúdo do BD atualizado indicando que foi liberado.

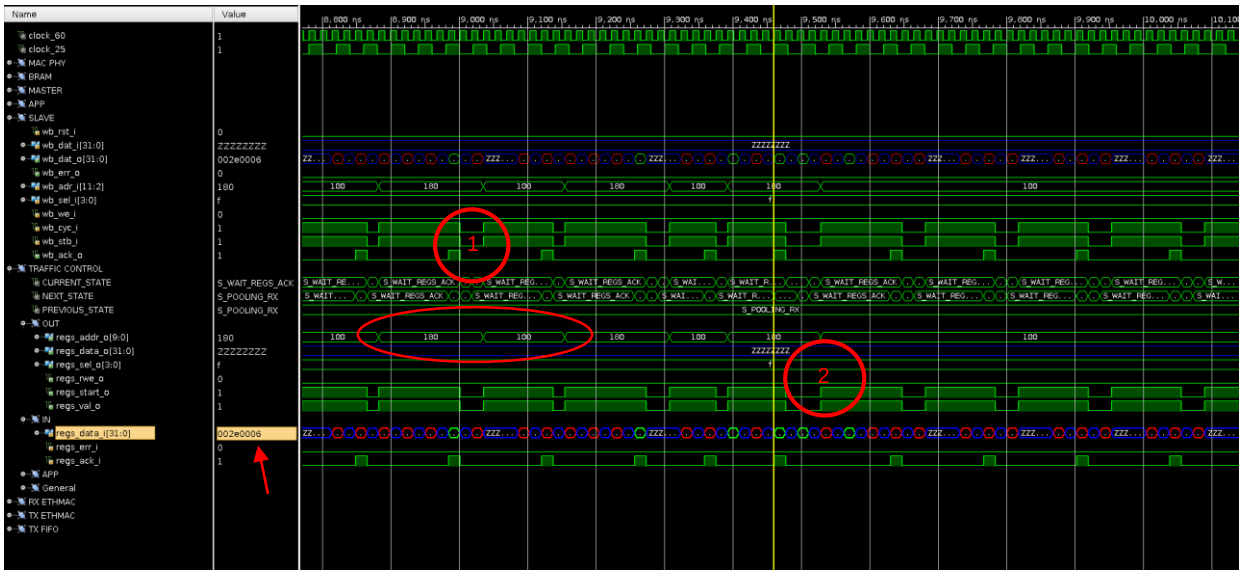

Figura 19- Pooling de RX e TX.

A Figura 20 apresenta a interface mestre do *wishbone* recebendo os pacotes (1). Como citado anteriormente, o endereço alocado para a recepção é inicialmente 0x3C00 e esse endereço é incrementado de 4 em 4. Os valores recebidos no PHY são montados em palavras de 32 bits e o dado é escrito na memória BRAM a partir da porta B (2).

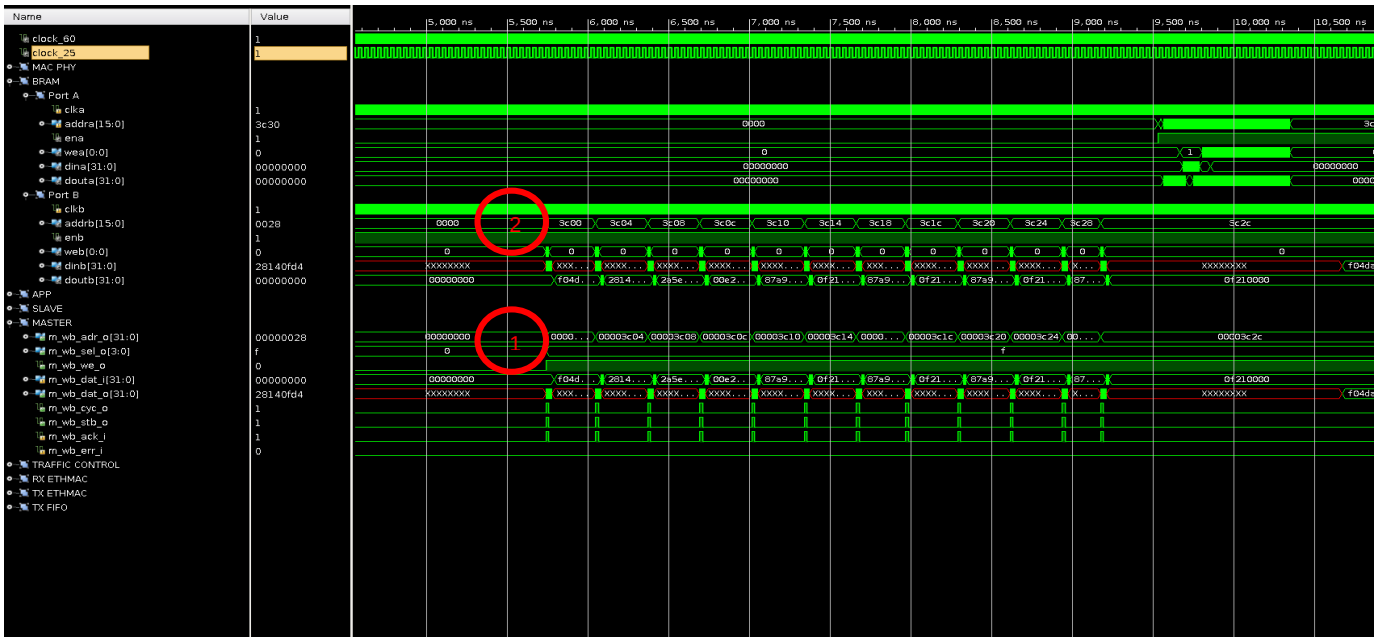

Figura 20- Recepção na interface Master do wishbone e escrita na memória.

4.2.4 Memória BRAM

Para armazenar os *frames* é utilizada uma memória BRAM *true dual port.* Desta forma, tanto o Ethmac quanto o IP-User podem realizar escrita e leitura de dados. Ambas portas da BRAM utilizam o *clock* do circuito (60 Mhz) sendo a porta A conectada ao IP-User para realizar uma aplicação e a porta B conectada com o Ethmac para armazenar os dados que são capturados da rede e que serão enviados. A instância dessa memória é apresentada no anexo A.

4.2.5 Aplicação do usuário

Para validar a transmissão é utilizada uma aplicação que faz loopback do *frame* recebido. Essa aplicação possui uma interface com a porta A da memória e, assim que recebe uma requisição através do sinal *rx_req_i*, realiza uma leitura no endereço alocado para o BD de transmissão. O cabeçalho é então alterado, invertendo-se os campos MAC *source* e MAC *destination* e, após a escrita do cabeçalho, a máquina de estados fica alternando entre leitura e escrita do payload até que o *frame* chegue ao fim.

Além da interface com a memória a aplicação possui uma interface com o *Traffic Control*. Assim que o dado é tratado pela aplicação e escrito na memória novamente, a aplicação envia ao Traffic Control o dado a ser escrito no BD de transmissão através do registrador *tx_data_o* contendo tamanho e status (ready,

CRC, pads). Desta forma o Ethmac inicia a leitura do dado no endereço do BD correspondente.

A Figura 21 apresenta o cabeçalho do *frame* sendo lido do endereço correspondente ao BD de recepção através do sinal de saída da porta A (*douta*) (1) e escrito no endereço correspondente ao BD de transmissão através do sinal de entrada da porta A (*dina*) (2). Após o cabeçalho ser tratado ocorre a leitura e escrita do restante do *frame* (3). Portanto o dado foi tratado pela aplicação e escrito novamente na memória.

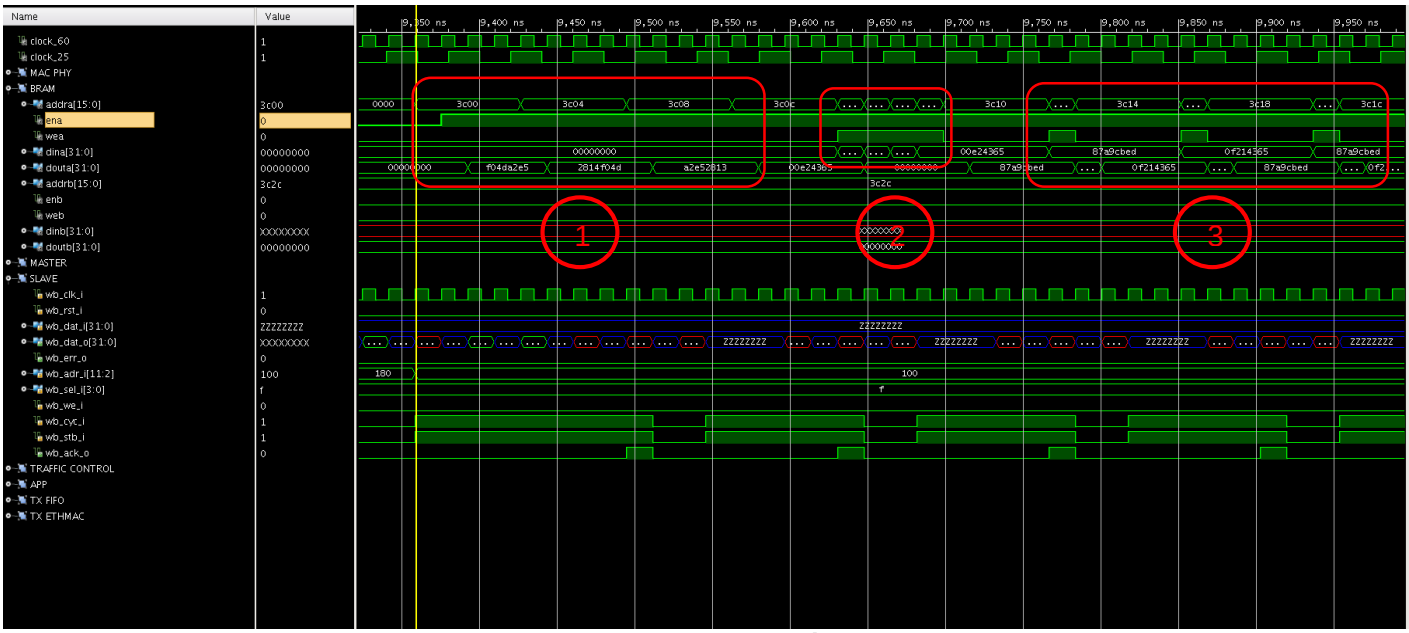

Figura 21- Leitura e escrita da memória feita pela aplicação.

4.2.6 Transmissão de *frames*

Quando a aplicação termina de escrever os dados na memória, é feita a escrita no wishbone *slave* através do *Traffic Control* indicando para o MAC que um dado está pronto para ser transmitido. A Figura 22 ilustra o momento em que o MAC começa a leitura do dado armazenado na memória, quando os sinais *m_wb_cyc_o* e *m_wb_stb_o* são setados para '1' e o *m_wb_we_o* é setado para '0' (1). A figura ilustra também que a leitura ocorre nos ponteiros de endereços de transmissão (a partir de 0x0000), e o que é amostrado no sinal *dinb* é o mesmo dado lido e registrado no *m_wb_dat_i (2).*

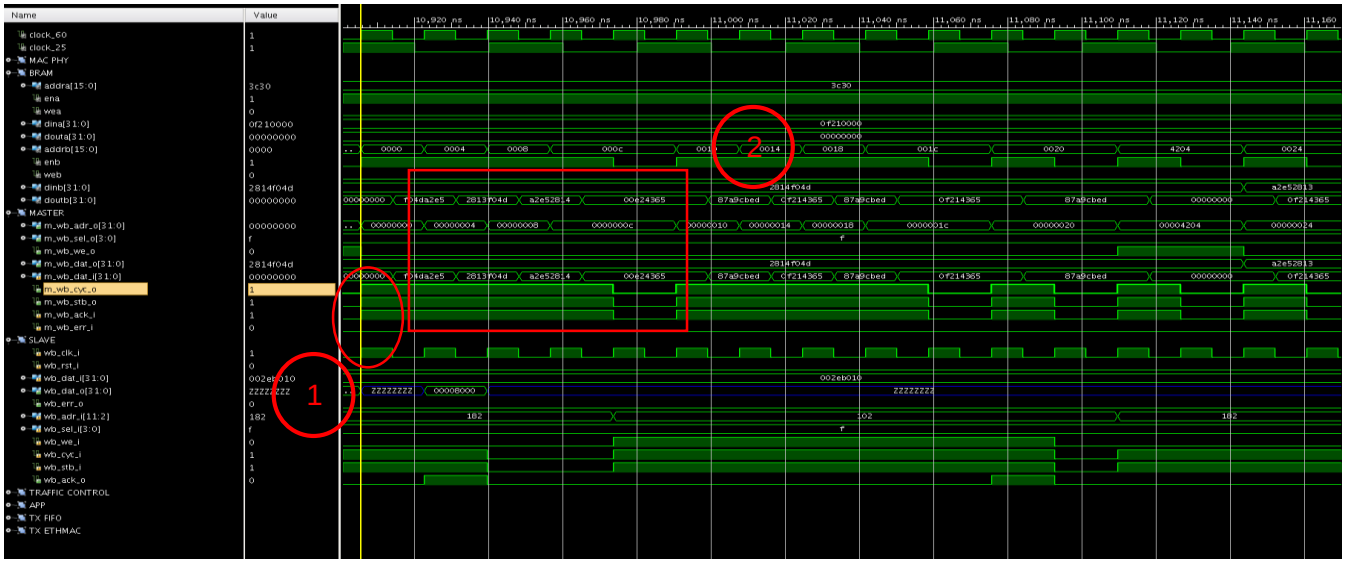

Figura 22- Leitura do *frame* feita pelo MAC.

Depois que a leitura dos dados é realizada, o mesmo é escrito na FIFO destinada à transmissão e quando a mesma fica cheia o *frame* é transmitido. A Figura 23 apresenta o *frame* sendo transmitido no PHY. O destaque mostra o valor 0XF04DA25E2813 como MAC *destination* e o valor 0XF04DA25E2814 como *MAC source* provando que os mesmos foram invertidos.

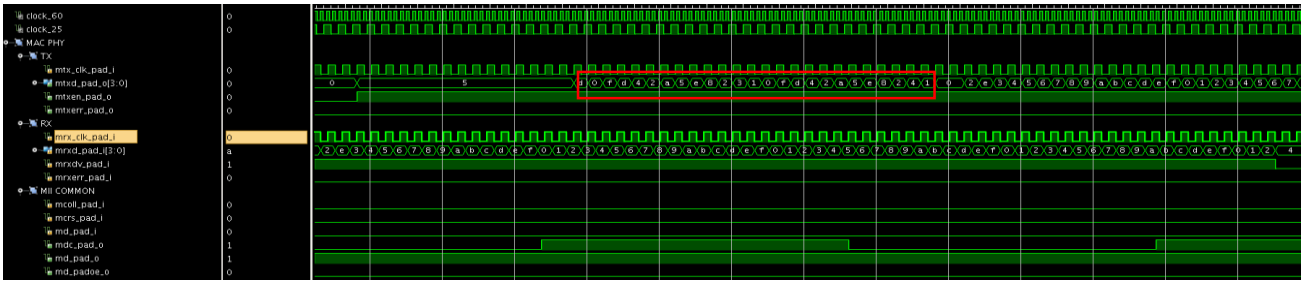

Figura 23- Transmissão do pacote no PHY.

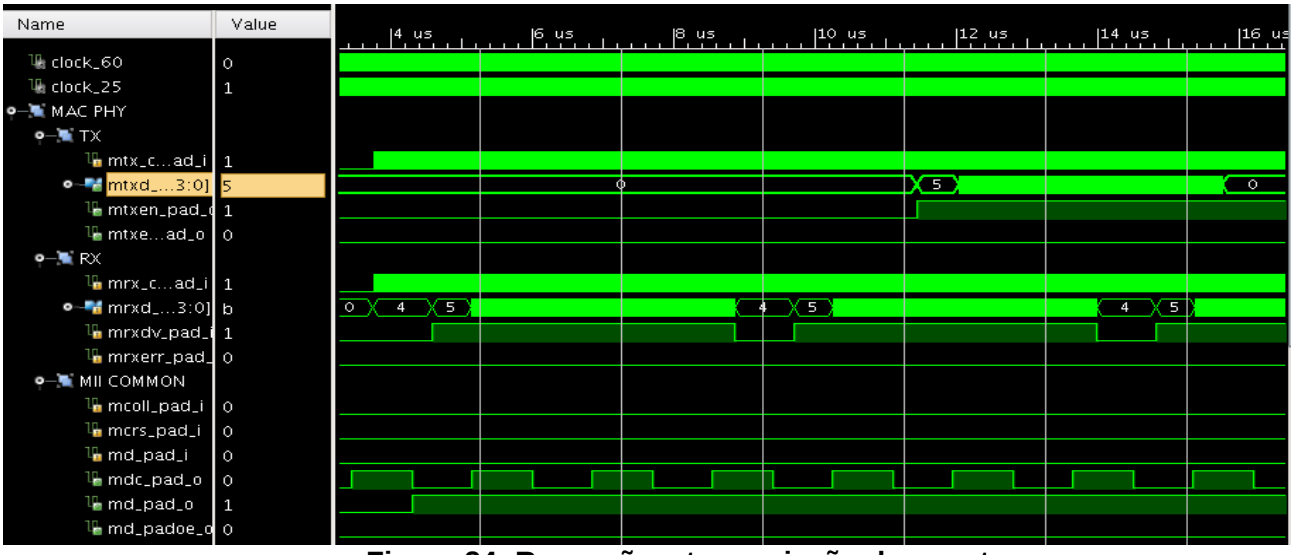

Figura 24- Recepção e transmissão do pacote.

4.3 Validação da interface via prototipação

Nesta seção são apresentadas as ferramentas e métodos utilizados para validar a interface em FPGA. Para isso foi seguido o fluxo de projeto de FPGAs, montando-se o ambiente para realização da prototipação. A verificação foi realizada através da ferramenta *Chipscope* e de um *software* desenvolvido pelo Autor.

4.3.1 Ambiente de prototipação

A validação do sistema é feita através de uma placa filha chamada TB-FMCL-INET que possui três portas Ethernet com suporte a 10/100 Mbps conectada à placa de prototipação através de um FMC (FPGA *Mezzanine Card)*, que fornece os conectores e interface modular necessários para comunicação com um FPGA localizado em uma placa base [XIL17]. A Figura 25 apresenta o diagrama de blocos dessa placa. Nessa figura é possível observar a interface dos sinais do MII e do PHY com a FMC. Além disso, a placa gera um *clock* de 25 MHz para uma operação a 100 Mbps, que é associado aos sinais RXC e TXC.

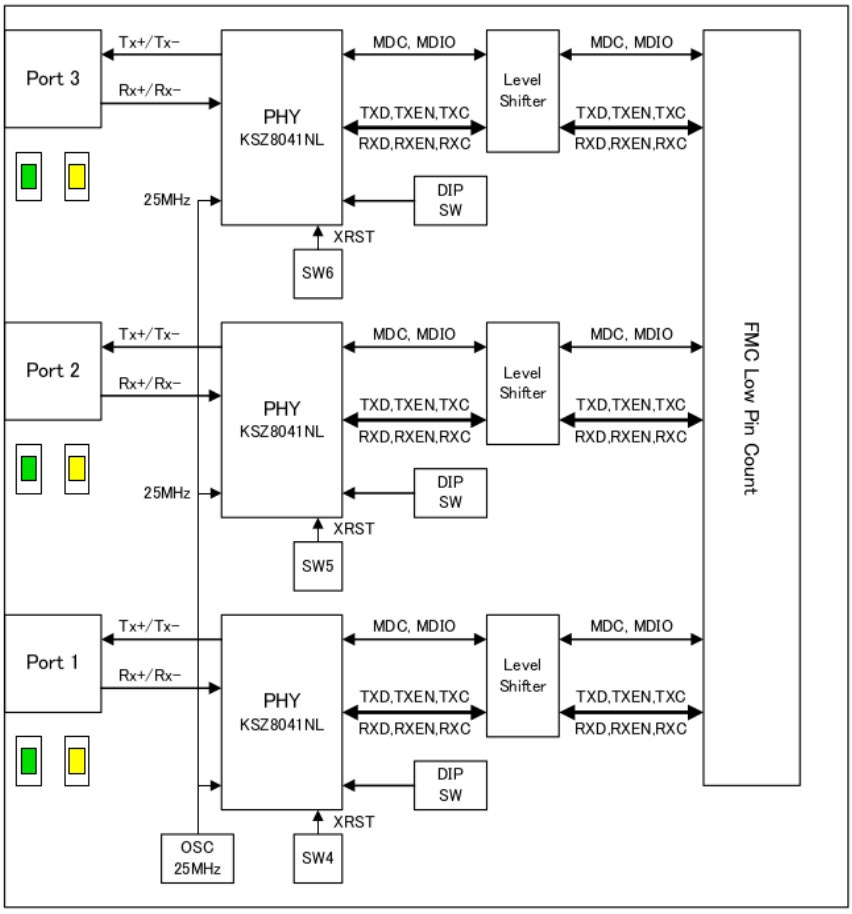

Figura 25- Diagrama de blocos FMC. [INR10]

Neste trabalho apenas uma porta é utilizada e os pinos da FMC são associados às portas do MAC através de uma tabela disponível na documentação [INR10]. Os pinos são declarados no arquivo de restrições (*constraints*) que é utilizado na síntese lógica. Por fim, a placa filha possui *DIP Switches* que devem ser configurados para habilitar a porta e habilitar comunicação *full-duplex.*

4.3.2 Software

A validação da recepção de frames necessita que sejam enviados pacotes pela rede. Esse envio é realizado através de um software que abre uma conexão via socket raw no *host* e envia os pacotes para FPGA através de um cabo *crossover*. Para realizar a validação foi utilizado o mesmo endereço MAC de destino utilizado na simulação e um payload qualquer. A Figura 26 ilustra esse pacote no software wireshark, onde o endereço de destino é o mesmo configurado inicialmente (F0:4D:A2:E5:28:14) e o frame possui um tamanho de 51 bytes. A Figura 27 ilustra o fluxo básico para validação do envio desse pacote para o Core-IP.

```
▾ Frame 1: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0
    Interface id: 0 (enp4s2)
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  5, 2017 14:18:55.342255351 BRT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1496683135.342255351 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 65 bytes (520 bits)
    Capture Length: 65 bytes (520 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:llc:data]
▾ IEEE 802.3 Ethernet
  ▸ Destination: Dell_e5:28:14 (f0:4d:a2:e5:28:14)
  ▸ Source: D-LinkCo_90:bd:d8 (00:21:91:90:bd:d8)
    Length: 51
▾ Logical-Link Control
  ▸ DSAP: Unknown (0xca)
  ▸ SSAP: ISO Network Layer (0xfe)
  ▸ Control field: I, N(R)=93, N(S)=111 (0xBADE)
▾ Data (47 bytes)
    Data: bacacacacacacacacacacacacacacacacacacacacacacaca...
    [Length: 47]
```

Figura 26- Pacote enviado do *host* para a placa de prototipação.

Figura 27- Fluxo básico de um envio de pacotes do host até o Core-IP.

4.3.3 Chipscope

A partir da ferramenta *Chipscope* foi possível validar a recepção do pacote. Através de um cabo *crossover* conectado a uma porta da placa filha é possível realizar o envio desses pacotes para a placa de prototipação e validar a recepção dos mesmos.

A Figura 28 mostra o pacote que foi enviado para a rede sendo capturado no PHY de RX, onde é destacado o endereço MAC de destino sendo o MAC atribuído à placa (F0:4D:A2:E5:28:14) e o MAC de origem sendo o da interface do *host* que envia o pacote (00:21:91:90:BD:D8). O tamanho do pacote também é o correto de 51 bytes (0x33).

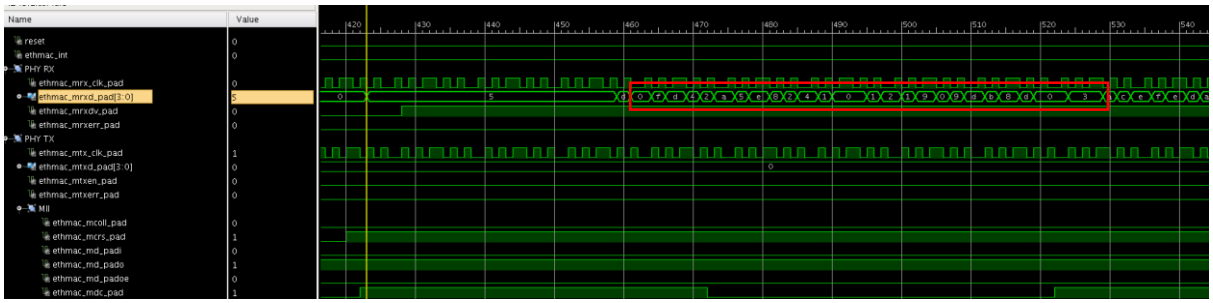

Figura 28- Pacote capturado no PHY RX.

A Figura 29 indica a escrita dos registradores e BDs para realizar a recepção de um *frame.* Após a configuração, a leitura acontece no *wishbone master*. Na Figura 30, os pontos destacados mostram a primeira palavra do *frame,* que indica a parte alta do MAC *destination (*0XF04DA2E5), sendo armazenada no endereço 0x3C00. O *chipscope* fez uma quebra no endereço, portanto o endereço destacado (0x00000f00) é a representação do endereço de referência para a recepção. Desta forma fica claro que a configuração nos registradores e BDs foram feitas corretamente e o *frame* foi aceito pela interface, validando uma recepção.

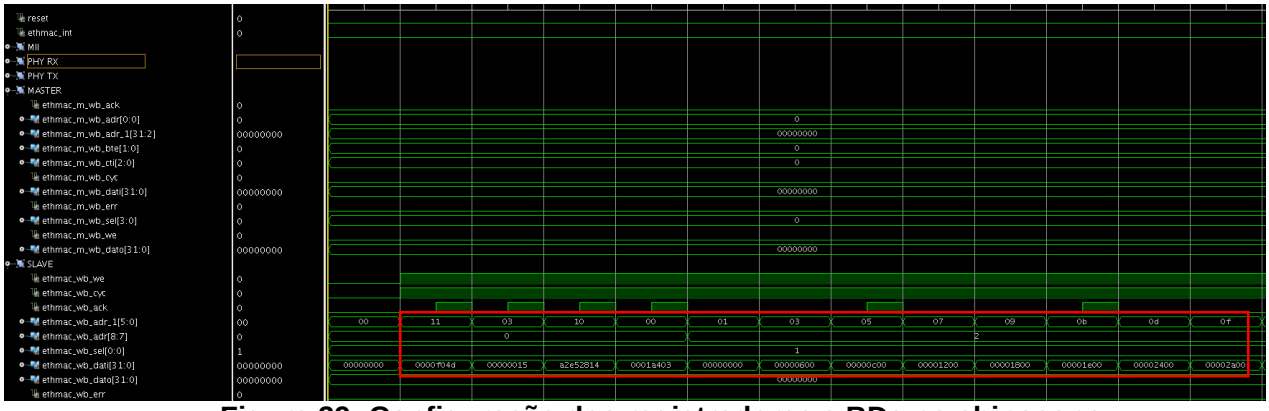

Figura 29- Configuração dos registradores e BDs no chipscope

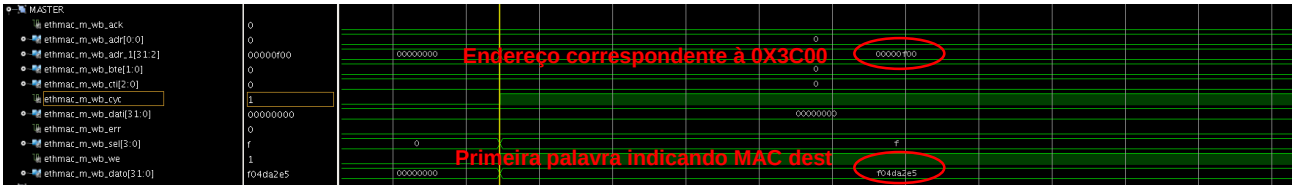

Figura 30- Recepção feita pelo *wishbone master*.

4.3.4 Características da implementação no FPGA

Esta Seção apresenta as características do hardware desenvolvido em termos de *timing* e ocupação do FPGA.

4.3.4.1 Timing

Para implementação foram utilizados dois *clocks* de referência. O primeiro *clock* de 25 MHz gerado pela placa filha e associado aos pinos de RX e TX do PHY, e o *clock* do circuito de 60 MHz. A placa de prototipação opera a uma frequência de 200 MHz. Assim, foi utilizado um divisor de *clock* para fornecer 60 MHz, que é a frequência nominal do circuito utilizado.

A partir das referências de *setup* e *hold* dos registradores é possível determinar o *slack time* dos caminhos críticos do circuito. Esse *slack time* é a diferença de tempo entre a requisição e a chegada de transferências dos dados de um registrador para o outro antes da borda de *clock* do domínio utilizado. Um dado é transferido com segurança entre dois registradores se os tempos de *setup* e *hold* são respeitados, ou seja, se o *slack* é positivo [XIL13].

A Tabela 2 apresenta os valores de *Worst Negative Slack (WNS)* que corresponde a tempo de *setup* e *Worst Hold Slack* (WHS) que corresponde a tempo de *hold,* ambos obtidos após a síntese física para essas duas frequências. Desta forma podemos observar que não existem violações de temporização, visto que não ocorre *slack* negativo.

Tabela 2- Resultados de *slack time* para as frequências utilizadas no projeto.

Frequência	WNS	WHS
60 MHz	+12.153 ns	+0.081 ns
25 MHz (RXC)	+35.368 ns	+0.120 ns
25 MHz (TXC)	+35.537 ns	+0.133 ns

4.3.4.2 Área

A Tabela 3 apresenta a quantidade de recursos que o projeto desenvolvido ocupa no FPGA.

Tabela 3- Ocupação da área do FPGA.

Recursos	Utilização	Disponível	Utilização (%)
LUT	2450	203800	1.20
FF	2791	407600	0.68
BRAM	36	890	4.04
IO	26	500	5.20
BUFG	4	32	12.50
MMCM	1	10	10.00

A partir do relatório gerado é possível observar que a área ocupada pela lógica da interface é pequena em relação ao total disponível no FPGA. A lógica utilizada representou em torno de 1% dessa área. Porém existem ainda os recursos de *buffers globais (*BUFG) e MMCM, em torno de 12 e 10% respectivamente, que são utilizados na geração de diferentes referências de *clock* que a interface possui. São utilizados também 36 blocos de RAMB18.

A Figura 31 apresenta a disposição das células ilustrando a ocupação da área do projeto no FPGA. A região azul no canto inferior esquerdo indica a posição do circuito no FPGA, deixando evidente o baixo custo em área da interface desenvolvida. Na direita é feita uma ampliação dessa região.

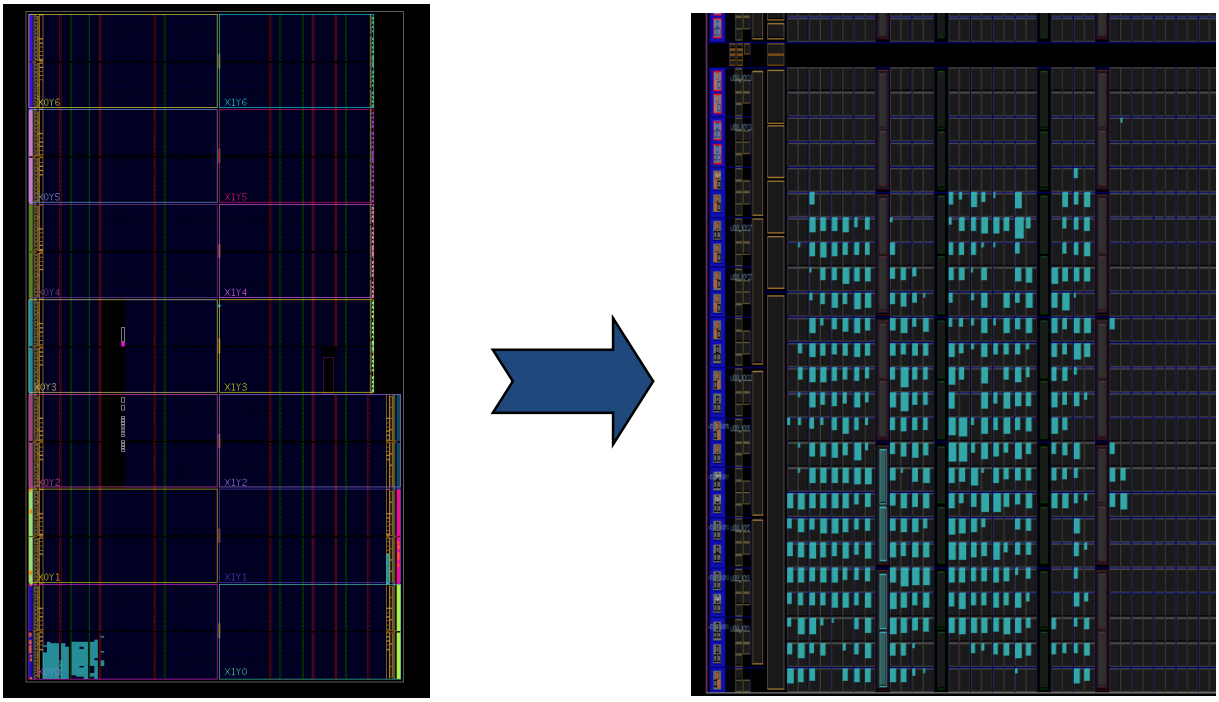

Figura 31- Posição das células do projeto no FPGA.

5 CONCLUSÃO

O trabalho descrito neste documento apresentou o desenvolvimento de uma interface *core Ethernet de 10/100 Mbs* para comunicação com dispositivos FPGAs. Essa interface pode ser utilizada para comunicação com diferentes IPs de usuário que forem embarcados em FPGAs, como por exemplo, processadores e sistemas multiprocessados.

Os conhecimentos adquiridos durante o desenvolvimento do trabalho são muito importantes para a área de Engenharia de Computação e compreendem: redes de comunicação, desenvolvimento de sistemas em dispositivos FPGAs, e desenvolvimento de projetos através da linguagem de descrição de hardware VHDL. Na área de redes foram abordados conceitos avançados de comunicação através das camadas mais baixas de um sistema de comunicação, que são muito utilizadas na área de telecomunicações. Foram aprofundados também os conhecimentos no fluxo de projeto de FPGAs, onde a ferramenta Vivado da *Xilinx* foi explorada.

O projeto alcançou parcialmente o objetivo inicial, visto que foram validadas as funcionalidades através de simulação na sua totalidade, porém apenas parte do processo foi validado em prototipação. Apesar disso, o *Core-IP Ethernet* desenvolvido apresenta resultados corretos de recebimento e transmissão de *frames* entre dispositivos, que podem ser realizados de forma remota, podendo ser utilizado em projetos futuros.

Como trabalhos futuros visa-se a validação do projeto completo através de prototipação e a comunicação com uma aplicação de maior porte tais como, um processador embarcado. Portanto, a contribuição desse trabalho é uma interface de comunicação flexível a alterações e importante para a área de pesquisa e indústria de telecomunicações.

6 REFERÊNCIAS

[IEE15] The Institute of Electrical and Electronic Engineers Inc. (IEEE) “IEEE Standard for Ethernet”. Disponível em: http://standards.ieee.org/about/get/802/802.3.html, setembro, 2015.

[INR10] Inrevium. “TB-FMCL-INET Hardware User Manual “. Disponível em: http://solutions.inrevium.com/products/pdf/pdf_TB-FMCL-INET_HWUserManual_01.00e.pdf, julho, 2010.

[INR13] Inrevium. “TB-7K-325T-IMG Hardware User Manual Rev. 1.07”. Disponível em: http://solutions.inrevium.com/products/pdf/pdf_TB-7K-325T-IMG_HWUserManual_1.07e.pdf, junho, 2013.

[KHA11] Khalilzad, N.; Pourshakour, S. “FPGA implementation of Real-time Internet Communication using RMII interface”. Disponível em: https://pdfs.semanticscholar.org/428f/14caf44283183cfc97bf4423c731a4e32145.pdf, fevereiro, 2011.

[MOH02] Mohor, I.”Ethernet Core-IP Specification”. Disponível em: http://www.cprover.org/firmware/doc/ethoc/eth_speci.pdf

[OPC16] *OpenCores*. “Ethernet MAC 10/100 Mbps Overview”. Disponível em: https://opencores.org/project,ethmac , janeiro, 2016.

[PAS08] Pasricha, S., Dutt, N. “On-Chip Communication Architectures: System on Chip Interconnect”, Morgan Kaufmann, 2008, pp. 541.

[PCI10] PCI-SIG Inc. “PCI Express Base Specification Revision 3.0”. Disponível em: http://composter.com.ua/documents/PCI_Express_Base_Specification_Revision_3.0.pdf, novembro, 2010, pp. 37.

[XIL13] Xilinx Inc. “Vivado Design Suite User Guide Using Constraints”. Disponível em: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf, março, 2013, pp. 51.

[XIL15] Xilinx Inc. “High Speed Serial”. Disponível em: http://www.xilinx.com/products/technology/high-speed-serial.html, outubro, 2015.

[XIL17] Xilinx Inc. “FMC Cards”. Disponível em: https://www.xilinx.com/products/boards-and-kits/fmc-cards.html, junho, 2017.

ANEXO A

Neste anexo são apresentadas as instâncias dos módulos da arquitetura da interface desenvolvida no TCC.

Instância do Ethmac

```
ethmac_inst : ethmac
  port map
  (
    wb_clk_i     => clock_60,
    wb_rst_i     => reset,

    -- control connections
    wb_adr_i     => ethmac_wb_adr,
    wb_dat_i     => ethmac_wb_dati,
    wb_dat_o     => ethmac_wb_dato,
    wb_sel_i     => ethmac_wb_sel,
    wb_we_i      => ethmac_wb_we,
    wb_cyc_i     => ethmac_wb_cyc,
    wb_stb_i     => ethmac_wb_stb,
    wb_ack_o     => ethmac_wb_ack,
    wb_err_o     => ethmac_wb_err,

    -- memory connections
    m_wb_adr_o       => ethmac_m_wb_adr,
    m_wb_dat_i       => doutb,
    m_wb_dat_o       => ethmac_m_wb_dato,
    m_wb_sel_o       => ethmac_m_wb_sel,
    m_wb_we_o        => ethmac_m_wb_we,
    m_wb_cyc_o       => ethmac_m_wb_cyc,
    m_wb_stb_o       => ethmac_m_wb_stb,
    m_wb_ack_i       => ethmac_m_wb_ack,
    m_wb_err_i       => ethmac_m_wb_err,
    m_wb_cti_o       => open,
    m_wb_bte_o       => open,



   -- PHY connections
    -- tx
    mtx_clk_pad_i    => ethmac_mtx_clk_pad,
    mtxd_pad_o       => ethmac_mtxd_pad,
    mtxen_pad_o      => ethmac_mtxen_pad,
    mtxerr_pad_o     => open,
```

```
    -- rx
    mrx_clk_pad_i     => ethmac_mrx_clk_pad,
    mrxdv_pad_i       => ethmac_mrxdv_pad,
    mrxd_pad_i        => ethmac_mrxd_pad,
    mrxerr_pad_i      => ethmac_mrxerr_pad,
    -- MII
    md_pad_i          => ethmac_md_padi,
    md_pad_o          => ethmac_md_pado,
    md_padoe_o        => ethmac_md_padoe,
    mdc_pad_o         => ethmac_mdc_pad,
    -- etc
    mcoll_pad_i       => ethmac_mcoll_pad,
    mcrs_pad_i        => ethmac_mcrs_pad,
    int_o             => ethmac_int
  );
```

Instância da Memória BRAM

```
mem: for i in 0 to 31 generate
    memory_inst : RAMB16_S1_S1
    port map (
    clka      => clock_60,
    addra     => addra(15 downto 2),
    ena       => ena,
    wea       => wea,
    DIA(0)    => dina(i),
    DOA(0)    => douta(i),
    SSRA      => '0',
    clkb      => clock,
    addrb      => addrb(15 downto 2),
    enb       => enb,
    web       => web,
    DIB(0)    => ethmac_m_wb_dato(i),
    DOB(0)    => doutb(i),
    SSRB      => '0'
    );
  end generate mem;
```

Instância do Traffic Control

```
traffic_control_inst : entity work.traffic_control
   port map
   (
      clock_60          => clock_60,
      reset             => reset,

      -- MAC
      regs_addr_o       => ethmac_wb_adr,
      regs_data_i       => ethmac_wb_dato,
      regs_data_o       => ethmac_wb_dati,
      regs_sel_o        => ethmac_wb_sel,
      regs_rwe_o        => ethmac_wb_we,
      regs_start_o      => ethmac_wb_cyc,
      regs_val_o        => ethmac_wb_stb,
      regs_ack_i        => ethmac_wb_ack,
      regs_err_i        => ethmac_wb_err,


      -- APP
      app_rx_recv_o     => app_rx_recv_i,
      app_rx_ack_i      => app_rx_ack_i,
      app_tx_req_i      => app_tx_req_i,
      app_tx_empty_o    => app_tx_empty_i,
      app_tx_data_i     => app_tx_data_i,
      app_tx_ack_o      => app_tx_ack_i,
      app_rx_data_o     => app_rx_data_i,

      --General
      leds              => leds_int,
      traffic_ready     => traffic_ready);
```

Instância do divisor de clock

```
clock_mmcm : clk_200_to_60
   Port map (
      clk_in_200_p   => clk200m_p,
      clk_in_200_n   => clk200m_n,
      clk_out_60     => clock_60
   );
```

```
application_inst : entity work.application
   port map
   (
      clock              => clock_60,
      reset              => reset,

      -- MEM
      mem_ren_o          => ena,
      mem_wen_o          => wea,
      mem_data_i         => douta,
      mem_data_o         => dina,
      mem_addr_o         => addra,

      -- APP
      rx_req_i           => app_rx_recv_i,
      rx_ack_o           => app_rx_ack_i,
      tx_req_o           => app_tx_req_i,
      tx_empty_i         => app_tx_empty_i,
      rx_data_i          => app_rx_data_i,
      tx_data_o          => app_tx_data_i,
      tx_ack_i           => app_tx_ack_i
   );
```

Instância da aplicação do IP-User