PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA – FACULDADE DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

COMUNICAÇÃO DE ALTO DESEMPENHO EM DISPOSITIVOS FPGAS ATRAVÉS DE INTERFACES ÓPTICAS

Ernani Vinicius Thum

Volume Final do Trabalho de Conclusão de Curso

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre 2015

ABSTRACT

The need for communication interfaces to support the required traffic throughput is tightly connected with the search for low-cost solutions. This demand requires the development of new technologies to support high-speed traffic. The use of modern FPGAs devices are being used on a large scale for development of high-speed interfaces, due to their support for high-speed serial transmission and reception, in a flexible manner, with low-cost devices and reduced use of pins, simplifying the final product design.

The goal of this manuscript is to present the development of a 10 Gigabit Ethernet communication system through optic fiber using an FPGA device. The manuscript addresses the communication protocol and the resources available on the FPGA device for high-speed serial communication (transceivers). In addition, due to the complexity of the design steps in modern FPGAs devices, it is presented an introduction to these technologies to enable the description of the project development.

From the necessary knowledge for the development of the high-speed serial communication interface, the hardware infrastructure is created from the resources available on the device and is carried out tests to verify the system functionality. From this manuscript, the reader is expected to obtain the necessary knowledge for the development of communication interfaces on modern FPGA devices. This manuscript brings information that allows the reader to understand about the development of a 10 Gigabit communication systems through optic fiber using FPGA devices. Also, some of the information can also be used for other communication protocols, which have similar structures.

RESUMO

A necessidade de interfaces de comunicação cada vez mais rápidas para suportar a demanda de tráfego exigida nos dias de hoje está ligada com a busca de soluções de baixo custo para esse tipo de aplicação. Essa demanda faz com que seja necessário o desenvolvimento de novas tecnologias para dar suporte a esses sistemas. O uso de dispositivos FPGAs modernos está sendo usado em larga escala para desenvolvimento de interfaces de alta velocidade, devido a seus recursos que realizam transmissão serial de alta velocidade, de forma flexível, com baixo custo e utilização reduzida de pinos de transmissão.

O objetivo desse documento é apresentar o desenvolvimento de um sistema de comunicação 10 Gigabit Ethernet utilizando um dispositivo FPGA através de fibra óptica, além dos conhecimentos necessários para o desenvolvimento deste tipo de interface. Para isso o documento abordará as informações sobre o protocolo de comunicação Ethernet 802.3ae e os recursos disponíveis no dispositivo FPGA modernas para comunicação serial (*transceivers*). Além disso, devido à grande quantidade de etapas de desenvolvimento em dispositivos FPGAs, é apresentada uma introdução a essas tecnologias que servirá de base para a descrição do desenvolvimento do projeto.

A partir dos conhecimentos necessários para o desenvolvimento da interface de comunicação, são apresentados a infraestrutura de hardware desenvolvida a partir dos recursos disponíveis no dispositivo e são realizados testes para verificação do funcionamento do sistema. A partir deste documento, espera-se que o leitor obtenha os conhecimentos necessários para o desenvolvimento de interfaces de comunicação em dispositivos FPGAs modernos. A leitura deste documento permite ao leitor não só a obtenção de conhecimento sobre desenvolvimento de sistemas de comunicação em FPGAs através de interfaces ópticas, mas sobre aspectos relacionados à transmissão serial que são compatíveis com diversos protocolos de comunicação.

SUMÁRIO

LISTA DE ABREVIAÇÕES

10GbE	10 Giga bit Ethernet.
10Gbps	10 Giga bits por Segundo
BRAM	Block Random Access Memory
CPLL	Channel-Phase Locked Loop
CRC	Cyclic Redundancy Check
DRC	Design Rule Check
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GAPH	Grupo de Apoio ao Projeto de Hardware
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
ILA	Integrated Logic Analyzer
IP	Intellectual Property
I/O	Input/Output
LUT	Lookup Table
MAC	Media Access Control
MMCM	Mixed-Mode Clock Manager
OAM	Operation, Administration and Management
OSI	Open Systems Interconnection Model
PCS	Physical Coding Sublayer
PISO	Parallel In - Serial Out
PLL	Phase Locked Loop
PMA	Physical Medium Attachment
PMD	Physical Medium Dependent
QoS	Quality of Service
QPLL	Quad-Phase Locked Loop
SIPO	Serial In – Parallel Out
UUT	Unit Under Test
XDC	Xilinx Design Constraint
XGMII	10 Gigabit Medium Independent Interface

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

A cada dia o mundo está mais conectado e interagindo por diversos dispositivos através da Internet. É a geração da Internet das Coisas, onde cada vez mais os dispositivos eletrônicos se tornam capazes de se comunicar e trocar informações. Porém não é somente o número de dispositivos conectados que aumenta, mas também a quantidade de dados que trafegam simultaneamente. Por exemplo, o *Cloud Computing* criou uma maneira de virtualizarmos nossos dados, para que se possamos acessá-los de qualquer local, vídeos podem ser acessados diretamente da Internet com alta definição e celulares são como computadores, permitindo efetuar qualquer tipo de operação na internet [CIS15]. Para acompanhar o aumento na demanda de tráfego de dados, é necessário que os sistemas de comunicações acompanhem esse crescimento, e para isso, melhorias nas infraestruturas de telecomunicações e de novas tecnologias de transmissão de dados se fazem necessárias.

O IEEE (*Institute of Electrical and Electronics Engineers*) mantém e desenvolve o padrão de comunicação Ethernet, protocolo mais utilizado no mundo. Quando o primeiro protocolo Ethernet 803.2 foi desenvolvido, em 1985, ele era definido como uma interface de comunicação que permitia tráfego de dados à 10 Mbps. Devido à demanda por velocidades mais altas, o padrão recebeu aprimoramentos ao longo dos anos, como a versão 803.2ae, de até 10 Gbps, em 2002 e a 802.3bm, de até 100 Gbps em 2015 [SPU14]. Apesar de o padrão Ethernet não ser o único modelo de comunicação disponível no mercado, este tem se tornando cada vez mais popular devido ao seu baixo custo de implementação e flexibilidade em se adequar a diferentes topologias de rede. Nas versões mais atuais, houveram melhorias de qualidade de serviços (QoS) e operações de administração e gestão (OAM), que garantiram melhor confiabilidade de conexão, permitindo que o protocolo seja também utilizado como solução para sistemas de comunicação metropolitanos (MANs), antes implementados no padrão SONET/SDH [TAN15].

Para o desenvolvimento de sistemas de comunicação, não basta apenas um protocolo eficiente, mas também dispositivos que permitam a sua implementação. Atualmente dispositivos FPGAs são a principal escolha para este tipo de desenvolvimento [FUH13], devido a suas interfaces de altas velocidades, como os modelos de *transceivers* GTX, GTH e GTZ, que atingem velocidades de 12,5, 13,1 e 28.05 Gbps respectivamente. Além disso, FPGAs provêm flexibilidade de desenvolvimento, desempenho e alta densidade, reduzindo assim o custo de desenvolvimento final dos produtos [XIL15b].

1.1 Motivação

O desenvolvimento de novos sistemas de comunicação de alta velocidade é necessário para suprir as demandas de tráfego de dados. Atualmente as interfaces Ethernet 10 GbE desenvolvidas em dispositivos FPGAs e transmitidas a partir de fibra óptica são apresentadas como uma ótima solução em relação à custo e benefício para desenvolvimento de interfaces de comunicação. Além da oportunidade de estudo da operação de um sistema de comunicação altamente utilizado, o projeto de conclusão de curso oportunizou o desenvolvimento de um projeto que utilizou um dispositivo FPGA específico para sistemas de comunicação. Além disso, esse projeto foi fruto de um trabalho que foi

realizado em conjunto com a empresa TERACOM Telemática Ltda, empresa brasileira de telecomunicações e financiadora dessa pesquisa, e o Grupo de Apoio ao Projeto de Hardware (GAPH), pertencente a Faculdade de Informática (FACIN) da Pontifícia Universidade Católica do Rio Grande do Sul. Com isso, o projeto deu início ao desenvolvimento de um produto que poderá futuramente estar disponível no mercado pela empresa financiadora da pesquisa.

1.2 Objetivos

Este trabalho de conclusão teve por objetivo o desenvolvimento de uma interface de alta velocidade utilizando um dispositivo da família de FPGAs Virtex 7. A interface de comunicação é compatível com o padrão 10GBASE-R do protocolo Ethernet 803.2ae através do uso de *transceivers GTH*, os quais já são integrados ao dispositivo. Em projeto anterior foi desenvolvido para a empresa TERACOM um testador para validação de interfaces 10GbE denominada XGETH_TESTER a qual foi desenvolvida em um dispositivo Virtex 5. Esse módulo é responsável por gerar, receber e validar tráfego de dados de acordo com as normas da RFC2544 [IET99]. Esse testador foi então utilizado para validação da nova interface de comunicação 10GbE desenvolvida.

A vantagem do desenvolvimento de interfaces 10GBASE-R em dispositivos Virtex 7 é justificada pela reduzida utilização de recursos do FPGA quando comparada com a interface 10GbE utilizada em projeto anterior. Como pode-se observar na Figura 1 o projeto legado foi desenvolvido no dispositivo Virtex 5. Para funcionamento dessa interface eram necessários 64 pinos de E/S para transmissão de dados, 32 buffers diferenciais, e uma interface DDR devido a necessidade de paralelização dos dados provenientes do meio físico. A partir desses recursos os dados passam a ser transmitidos em pacotes de 64 bits em uma frequência aceitável para operação em dispositivos FPGAs, já que nos atuais FPGAs não se consegue atingir a mesma frequência utilizada na comunicação serial internamente.

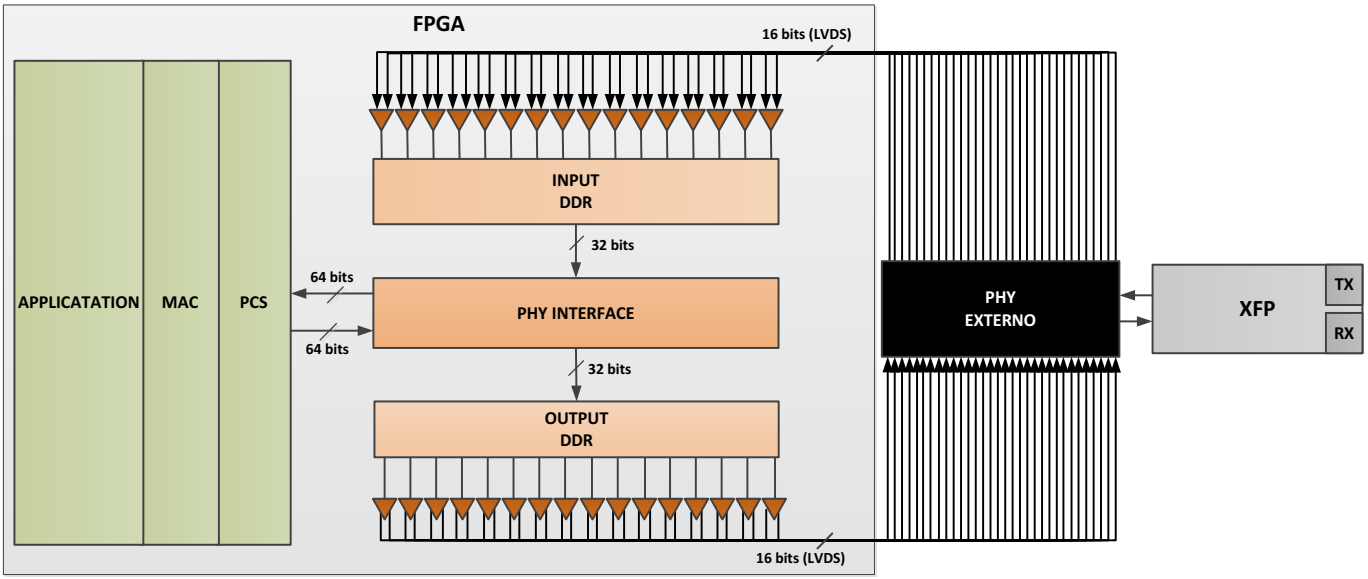


Figura 1 - Estrutura do projeto legado (Virtex 5).

Nos dispositivos Virtex 7, os *transceivers GTH* permitem a conexão da interface serial diretamente com o dispositivo FPGA, reduzindo para quatro o número de pinos necessários para transmissão de dados. Além disso, os *transceivers* já possuem recursos para paralização desses dados internamente, eliminando a necessidade de interfaces internas para essa função. Além disso, recursos

para a geração das referências de *clock* e sincronizadores não são necessários para a mudança entre os domínios de *clock*, pois os mesmos são tratadas internamente.

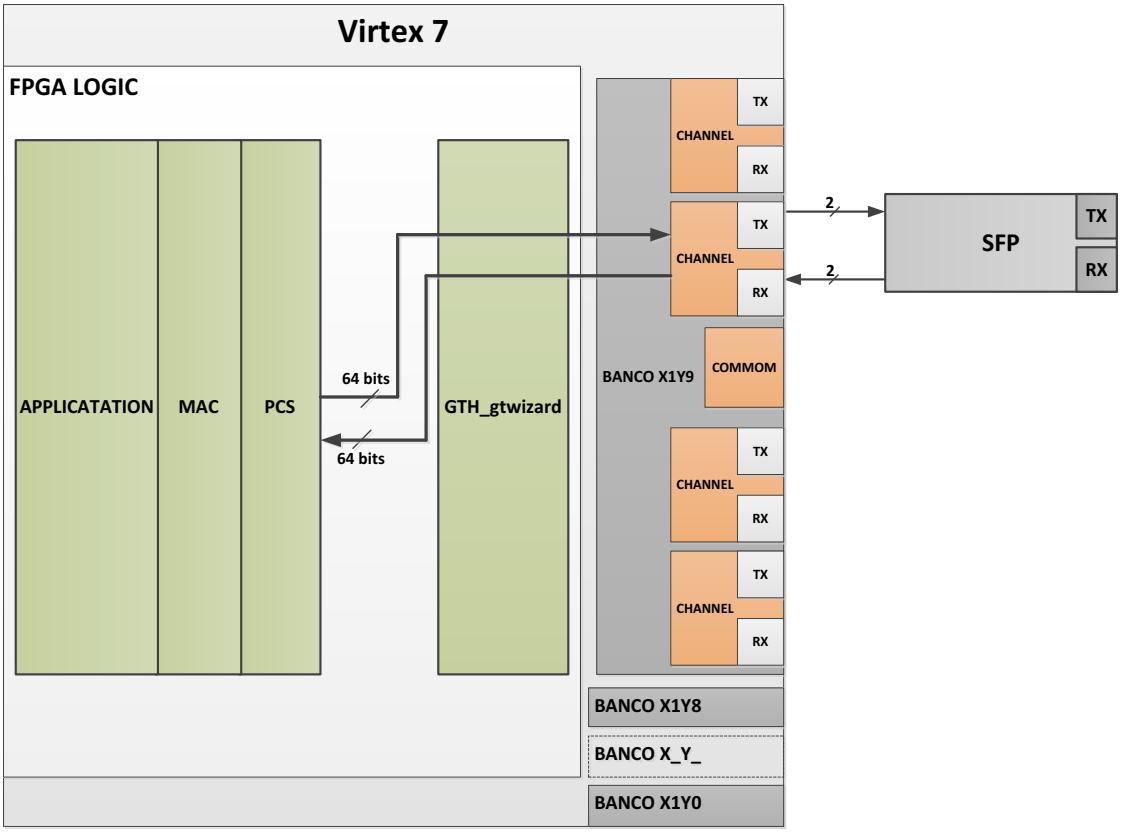


Figura 2 - Estrutura do projeto de TCC (Virtex 7).

Para o desenvolvimento do projeto foi exigido o estudo do protocolo Ethernet 803.2ae, estudo do fluxo de projeto em dispositivos FPGAs e um estudo sobre os *transceiver GTH* utilizados na interface serial de alta velocidade. Também foi necessária a integração do módulo testador responsável pelo tráfego de dados através da interface desenvolvida. Por fim foram realizados para validação e teste da interface desenvolvida.

1.3 Estrutura do Documento

Este documento é organizado como segue. O **Capítulo 2** descreve o padrão Ethernet 803.2ae, introduzindo os conceitos necessários para o entendimento do fluxo de dados na interface de alta velocidade. O **Capítulo 3** apresenta os *transceivers* GTH, bem como os recursos providos pelos mesmos, e que foram utilizados no desenvolvimento do projeto. O **Capítulo 4** provê informações quanto ao fluxo de projeto de um desenvolvimento em FPGAs que utilizam *transceivers GTH*. O **Capítulo 5** apresenta a estrutura de hardware desenvolvida durante a etapa de projeto. O **Capítulo 6** apresenta a implementação da interface desenvolvida em um dispositivo e a validação da mesma por meio de diferentes métodos de validação e teste. O **Capítulo 7** apresenta a conclusão do trabalho e projetos futuros.

2 PADRÃO ETHERNET 802.3AE

Este Capítulo tem por objetivo apresentar o protocolo de comunicação Ethernet 802.3ae, também denominado padrão 10GBASE. Esse padrão foi desenvolvido pelo IEEE para dispositivos que atuam com velocidade de 10 Gbps. Nele são definidas as interfaces que atuam nas camadas física e de enlace do modelo de referência OSI (*Open Systems Interconnection model*) [MIC14], e que tem como principais funções a recepção e transmissão de dados através de um canal e a formação de um enlace entre os dois nodos conectados, respectivamente [COM13].

Devido a existência de várias formas de implementação do padrão 10GBASE, são definidas famílias de padrões que variam conforme o tipo de sinal e o meio físico utilizado [SPU14]. A família em estudo pelo Autor é a 10GBASE-R, a qual especifica a camada física para interfaces que utilizam meio de transmissão óptico. Essa família será apresentada mais detalhadamente neste Capítulo. A Figura 3 ilustra os quatro diferentes modelos de implementação da família 10GBASE e suas respectivas camadas conforme o modelo de referência OSI. Além da família 10GBASE-R, existem protocolos específicos para interfaces de redes WAN's (*Wide area Networks*), o qual também realiza comunicação por meio de fibra óptica e é denominado 10GBASE-W. Por último os padrões 10GBASE-X e 10GBASE-T especificam as interfaces necessárias para transmissão através de fio de cobre e par trançado respectivamente[IEE12].

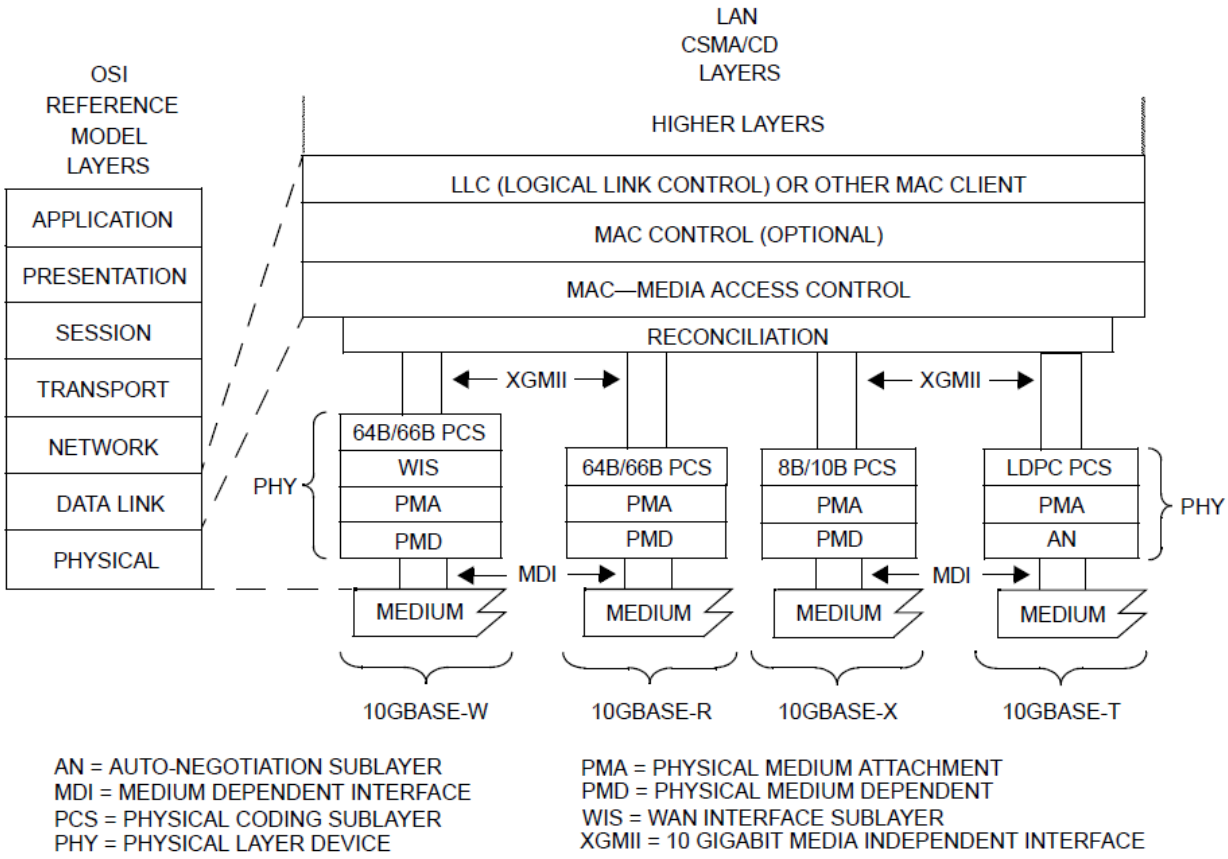


Figura 3 – Família de padrões 10GBASE [IEE12].

2.1 Família 10GBASE-R

A família 10GBASE-R define as interfaces necessárias para a implementação da camada física do modelo OSI. Ela é baseada no modelo de codificação 64b/66b e definida através de três subcamadas e uma interface para comunicação com a camada de enlace de dados, essas são denominadas: PMD (*Physical Medium Dependent*), PMA (*Physical Medium Attachment*), PCS (*Physical Coding Sublayer*), e interface XGMII (*10 Gigabit Media Independent Interface*) [IEE12].

Dentro dessa família também existe uma classificação dos dispositivos ópticos quanto ao tipo de comprimento de onda emitido pelo transmissor através da fibra óptica. No projeto desenvolvido, foram utilizados dispositivos ópticos da classe 10GBASE-LR (*long reach)*. Esses dispositivos são especificados para interfaces de longo alcance através de fibra multimodo (unidirecional) e com comprimento de onda de 1310 nm. Esses dispositivos são amplamente utilizados em redes LAN de até 10km de distância [SPU14].

2.1.1 Physical Medium Dependent (PMD)

A camada PMD é definida como a interface de acesso do sistema ao meio físico. O padrão 10GBASE-R é responsável pela tradução dos sinais ópticos para sinais elétricos no receptor, e a tradução de sinais elétricos para sinais ópticos no transmissor, permitindo tráfego de dados seriais nos dois sentidos [IEE12].

Existem diferentes modelos de dispositivos que realizam a função da camada PMD. O dispositivo utilizado no desenvolvimento deste projeto é o SFP+ (*Enhanced Small Form-factor Pluggable)*. Esse dispositivo é especificado pelo *SFF Committee*, comitê responsável pelo desenvolvimento do padrão SFF-8431 e que define as características elétricas e ópticas do dispositivo [SFF09].

2.1.2 Physical Medium Attachment (PMA)

A camada PMA realiza a de-serialização dos dados provenientes do módulo PMD e a transferência dos mesmos em formato de pacotes para a camada PCS. O processo inverso acontece no sentido de transmissão, onde ocorrem a serialização dos dados recebidos da camada PCS e a transferência serial dos mesmos para a camada PMD. Na Figura 4 é apresentado o modelo de funcionamento de um módulo PMA com interface de 16 bits, conforme é previsto no padrão Ethernet 802.3ae [IEE12]. No entanto, o tamanho dos pacotes enviados e recebidos em cada transferência pode variar conforme a tecnologia utilizada, como na implementação dos *transceivers GTH,* descritos no Capítulo 4, que utilizam interfaces de 32 ou 64 bits [XIL15a].

Essa camada também é responsável pela recuperação do *clock* do transmissor, no qual os dados recebidos são originados. Para isso, é preciso que durante a inicialização do dispositivo esse *clock* seja monitorado e recuperado. Essa função é necessária para o ajuste de fase entre os dois dispositivos, que apesar de operarem na mesma frequência, enfrentam sempre uma variação de fase, mesmo que pequena, e que necessita ser tratada, evitando possíveis erros de sincronismo [IEE12].

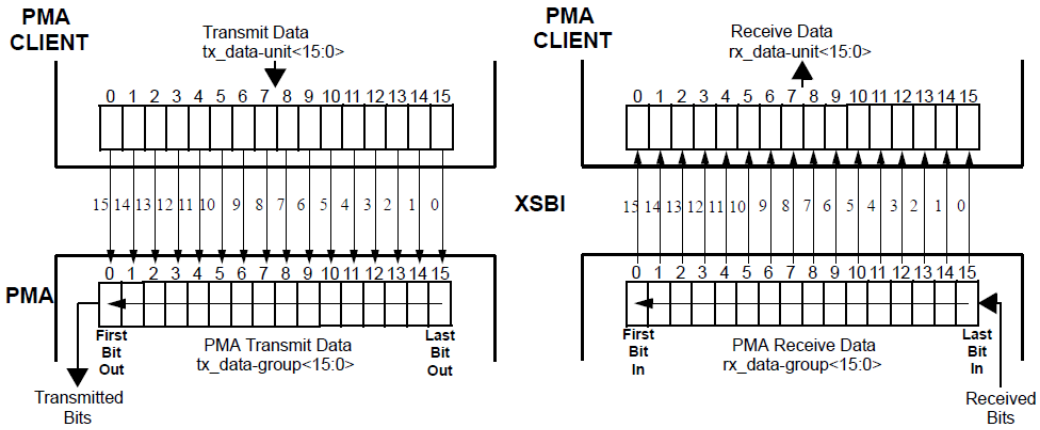


Figura 4 - Interface entre camadas PMD e PMA [IEE12].

Em resumo, os itens abaixo definem as funções da camada PMA para os dois sentidos de transmissão [IEE12]:

Para dados provenientes da camada PCS para a camada PMD;

- Prover o *clock* de transmissão para a camada PCS (*PMA Client*);
- Realizar a serialização de pacotes de dados;
- Transmissão dos dados seriais à camada PMD.

Para dados provenientes da camada PMD para a camada PCS:

- Recuperação do *clock* proveniente dos dados seriais recebidos do PMD;
- Paralelização dos dados e geração de pacotes de dados;
- Transmissão dos pacotes de dados à camada PCS (*PMA Client*);
- Prover informação de status do enlace de comunicação.

2.1.3 Physical Coding Sublayer (PCS)

A camada PCS é responsável pelo processo de codificação dos pacotes de dados provenientes da camada de enlace. O objetivo desse processo é codificar os pacotes de forma a facilitar a recepção e alinhamento dos mesmos no dispositivo ao qual a interface está conectada. A implementação da camada PCS é feita de forma bidirecional:

1. No sentido de transmissão, pacotes de dados em formato XGMII são codificados, embaralhados e transmitidos através de um módulo denominado *gearbox*;
2. No sentido de recepção, ocorrem processos de alinhamento dos pacotes recebidos, desembaralhamento e decodificação para o formato XGMII [IEE12].

Cada um desses processos é realizado por um módulo específico. Os detalhes sobre esses módulos serão abordados nas próximas Seções.

A Figura 5 apresenta o modelo de referência especificado pelo padrão IEE802.3ae para a camada PCS [IEE12]. A implementação apresentada na Figura 5 sugere a realização de duas transferências de pacotes de 32 bits (TXD) e 4 bits de controle (TXC) na interface XGMII por ciclo de relógio, no entanto neste documento será considerada uma implementação que realiza transferências de 64 bits e 8 bits de controle. Justifica-se o emprego de 64 bits neste projeto devido às restrições de *clock* no dispositivo FPGA empregado.

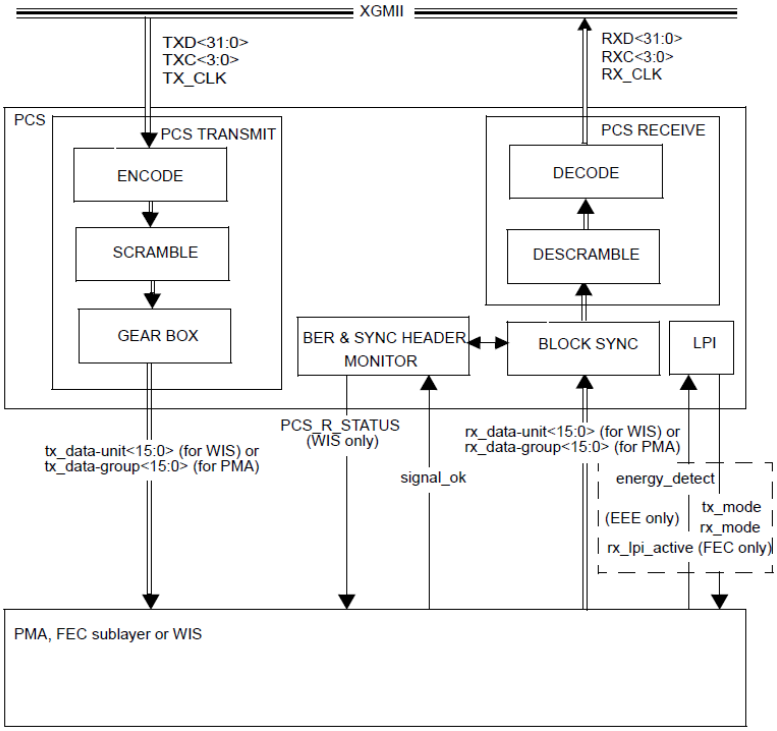


Figura 5 - Diagrama da camada PCS [IEE12].

2.1.3.1 Codificador e Decodificador 66b/64b

O padrão 66b/64b é uma codificação de dados desenvolvida para sistemas de comunicação serial 10 Gigabit Ethernet. A codificação é composta por pacotes de dados de 64 bits precedidos por cabeçalhos de dois bits, formando então pacotes de 66 bits. O cabeçalho do pacote define dois tipos de pacotes distintos (Figura 6). O primeiro consiste em um pacote de cabeçalho '01' seguidos por 64 bits de dados. O segundo consiste em um cabeçalho '10', seguido de um byte de identificação e 54 bits de dados, sendo o último bloco dividido em sete *slots* de um byte, os quais podem conter bytes de dados ou de controle. Neste caso, o byte identificador define que informações estão sendo transmitidas dentro de cada *slot* [ATH15].

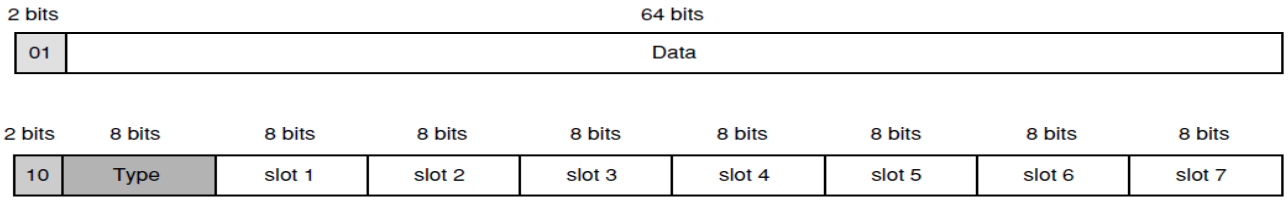


Figura 6 - Tipos de pacotes do padrão 66b/64b [ATH15].

Existem outros métodos de codificação que demandam implementações de camadas PCS distintas e que variam conforme o protocolo utilizado. Como exemplo, o protocolo PCIexpress utiliza uma codificação denominada 8b/10b, onde existe um overhead de 2 bits para cada 8 bits de dados.

2.1.3.2 Embaralhador e Desembaralhador

O embaralhamento de dados é um processo reversível que modifica o padrão dos pacotes antes do envio. Esse processo tem por objetivo quebrar possíveis sequências longas de zeros ou uns dentro dos pacotes, evitando que estas prejudiquem a capacidade do receptor para recuperar o sinal de *clock*.

O desembaralhamento é o processo inverso, onde a informação enviada pelo transmissor é reconstituída [ATH15].

Ambos os módulos utilizam um LFSR (*linear feedback shift-register*) para efetuar o embaralhamento ou desembaralhamento dos dados. Esses são constituídos de registradores em série, que realimentam a entrada de dados do módulo em portas lógicas do tipo ou-exclusivo (XOR), de acordo com um dado polinômio. A Figura 7 apresenta um modelo simples de embaralhador e desembaralhador, com o polinômio gerador $P(x) = 1 + x^2 + x^3$ [ATH15].

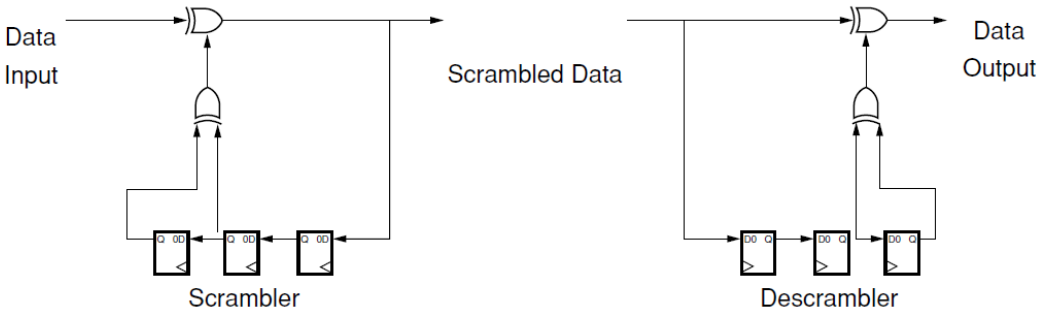


Figura 7 – Exemplos de circuitos embaralhador e desembaralhador [ATH15].

Os módulos exigidos pelo padrão Ethernet 802.3ae para embaralhadores e desembaralhadores utilizam o polinômio gerador $P(x) = 1 + x^{39} + x^{58}$, [IEE12]. A Figura 8 apresenta um exemplo de implementação do embaralhador dentro do padrão exigido.

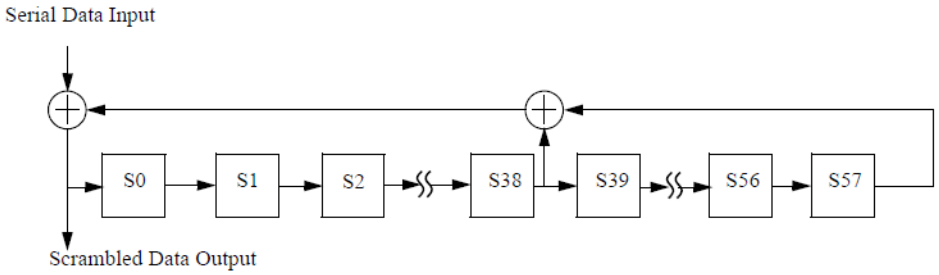


Figura 8 – Embaralhador de polinômio gerador $P(x) = 1 + x^{39} + x^{58}$ [IEE12].

O modelo apresentado na Figura 8 necessita de uma frequência altíssima para produzir um processo eficiente devido ao cascateamento de registradores, tornando-o impraticável em projetos de hardware. Para contornar esse problema é utilizada uma versão paralela deste módulo, acelerando assim o processo de embaralhamento e desembaralhamento. A arquitetura paralela deste módulo é apresentada na Figura 9.

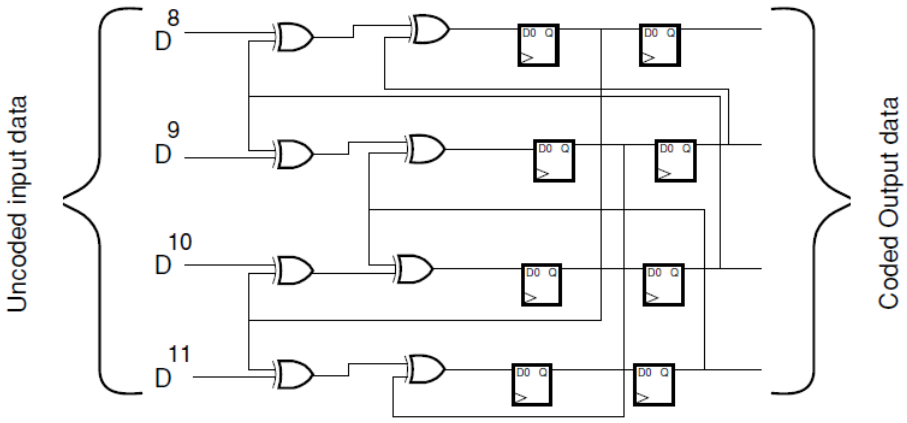


Figura 9 - Implementação de um Embaralhador Paralelo [FUH13].

2.1.3.3 Gearbox

Devido ao módulo PMA receber pacotes em transferências de 16, 32, ou 64 bits de acordo com a frequência de operação utilizada, ao fato que os pacotes transferidos do módulo de codificação 66b/64b estarem em um formato de 66 bits, é necessário que os mesmos sejam reagrupados, formando pacotes compatíveis com a interface PMA. O módulo *Gearbox*, realiza a formatação dos pacotes para o envio à camada PMA.

O módulo *Gearbox* gerencia os dados de forma a transferir os pacotes (codificados em 66 bits) à camada PMA (formatados em 64 bits). Inicialmente o header está nos dois primeiros bits do pacote, sendo apenas possível transferir 62 dos 64 bits de dados. No próximo pacote transfere-se o novo header (2 bits) e 60 bits de dados. Desta forma, a cada ciclo de clock, 2 bits são armazenados no *Gearbox*. Ao final do 32^{o} ciclo de clock, o *Gearbox* possui uma palavra completa de 64 bits armazenada. Esta palavra é transmitida no 33^{o} ciclo de clock, sendo após realizada uma pausa na transmissão de dados.

2.1.3.4 Sincronizador de Blocos (Block Synchronizer)

O bloco de sincronismo recebe os pacotes da camada PMA, reagrupando-os em pacotes de 66 bits. A análise dos cabeçalhos (dois bits mais significativos), é feita com objetivo de encontrar o alinhamento correto dos pacotes recebidos. Se o cabeçalho contiver os padrões ‘10’ ou ‘01’, é verificado se nos próximos 66 pacotes ou menos, dependendo da implementação, também haverá cabeçalhos válidos. Quando a sequência de cabeçalhos válidos é encontrada, o alinhamento é considerado correto, permitindo então o envio dos pacotes à camada de decodificação. Caso o padrão ‘11’ ou ’00’ seja encontrado, o alinhamento dos dados é deslocado e a verificação dos cabeçalhos reiniciada.

2.1.4 Interface XGMII

A interface XGMII (10 Gigabit Medium Independent Interface) tem por objetivo padronizar os pacotes de dados provenientes de diferentes formas de implementações de camadas físicas do protocolo Ethernet, adequando-as à um padrão único que possa ser interpretado nas camadas superiores.

O tamanho do pacote enviado e recebido por essa camada pode variar conforme a implementação. No padrão Ethernet, a interface é descrita utilizando barramentos de 32 bits de dados e 4 bits de controle, porém a abordagem utilizada nesse documento considerará uma implementação com 64 bits de dados e 8 de controle.

A interface utiliza um bit de controle para cada byte de dados, indicando que tipo de informação o byte contém. Quando o sinal de controle é ‘0’, dados normais são transmitidos, já o sinal de controle ‘1’ indica que caracteres especiais estão sendo transmitidos. Dentro desses estão inclusos os sinais de início e fim de pacote, canal de transmissão ocioso (*Idle*) e erro na transmissão [FRA00]. As codificações utilizadas para determinação dos dados de controle estão disponíveis na **Error! Reference source not found.**.

2.2 Medium Access Control (MAC)

O controle de acesso ao meio (MAC) é a primeira subcamada pertencente à camada de enlace (camada 2) e a mais próxima da camada física (camada 1), referente ao modelo de referência OSI [IEE12]. As funções dessa camada são:

- Recebimento dos dados provenientes da camada física;
- Delimitação e sincronismo dos pacotes de dados;
- Encapsulamento e Desencapsulamento: remoção dos delimitadores de início e fim de pacote e preâmbulo;
- Controle de erro a partir da utilização de FCM (Frame Check Sequence);
- Descarte de pacotes com erro;
- Interpacket Gap: o espaçamento mínimo necessário entre pacotes é 12 bytes.

O pacote Ethernet (Figura 10) tem por objetivo transportar, além dos dados úteis, os endereços de origem e destino do pacote, para permitir a comunicação ponto a ponto dos dispositivos conectados à uma mesma rede. O pacote é formatado em campos de bytes determinados pelo protocolo Ethernet para permitir a decodificação de cada informação dentro do pacote.

A subcamada MAC tem por objetivo determinar o início e o fim dos pacotes, bem como a verificação de sua integridade. Esses campos são descartados após o uso, não sendo repassados para as camadas superiores. Para isso são utilizados os seguintes campos:

- ***Preamble*:** o preâmbulo consiste em 7 bytes com valor igual a 0x55, identificando o início de uma transmissão.
- ***Start Frame Delimiter* (SFD):** um byte com a sequência 0xD5. O frame Ethernet começa imediatamente após essa sequência.
- **Frame Check Sequence (FCS):** esse campo consiste em um campo de 4 byte que armazenam o valor de CRC (*Cyclic Redudancy Check*), calculado através dos dados do pacote e utilizado para verificação da integridade do mesmo no receptor.

Os demais campos constituem-se de um cabeçalho com informações sobre o pacote e os dados úteis (*payload*), que serão tratados pelas camadas superiores. Essas possuem os seguintes campos:

- ***Destination*/*Source Addresses***: endereços MAC de 6 bytes, utilizados na identificação da origem e destino do pacote recebido.
- ***Length/Type:*** Este campo de 2 bytes é utilizado para dois propósitos diferentes. O primeiro é informar o número de bytes do pacote, quando este for inferior a 1500 bytes. Valores maiores que 1536 definem o tipo de protocolo ao qual pertence o pacote (*EtherType*). Como exemplo, o protocolo IPV4 utiliza o *EtherType* 0x0800 (valor em hexadecimal). Valores entre 1501 e 1535 não são utilizados.
- **Data/LLC:** Pacote de dados úteis, pode variar entre 46 bytes à 1500 bytes.

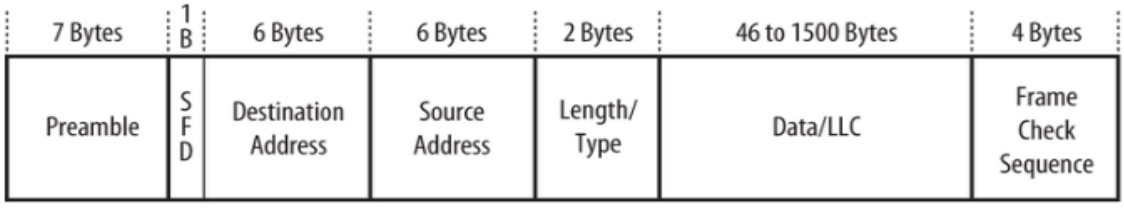


Figura 10 – Formato geral de um pacote Ethernet.

3 TRANSCEIVERS GTH

O crescimento da indústria de FPGAs e o avanço de novas tecnologias para esses dispositivos simplificou o desenvolvimento de interfaces Ethernet 10 Gbps. Os *transceivers* GTH são interfaces presentes nas gerações atuais de dispositivos FPGAs, desenvolvidas pela empresa Xilinx Inc., e presentes em modelos como a Virtex-7 e Kintex-7 [FUH13]. Este tipo de *transceiver* também pode ser encontrado nos dispositivos FPGAs desenvolvidos pela na empresa Altera Corporation, contudo utiliza-se uma nomenclatura distinta. Esse é o caso do *transceivers* GX [ALT15].

Os *transceivers* GTH permitem que sinais seriais de até 13,1Gbps sejam conectados diretamente aos dispositivos FPGAs, eliminando a necessidade de dispositivos físicos externos à FPGA para transmissão e recepção dos dados seriais. Em tecnologias anteriores, esses dispositivos eram necessários devido à incompatibilidade com a alta frequência de operação envolvida nesses processos. A desvantagem desses dispositivos externos é a necessidade de grandes quantidades de pinos de entrada e saída. Como exemplo, para uma interface externa de 16 bits, seriam necessários 16 pinos diferenciais para cada sentido de transmissão para uma única interface 10GbE, totalizando 64 pinos de I/O. A utilização de *transceivers* GTH otimizou o desenvolvimento de sistema de comunicação de alta velocidade, reduzindo o número de pinos de entrada e saída da placa para quatro (dois pares diferenciais), tornando a interface mais compacta [FUH13].

Os *transceivers GTH* podem ser utilizados em diferentes protocolos de transmissão serial, dentro destes, o padrão 10GBASE-R, conforme apresentado na Tabela1.

Tabela1 - Protocolos suportados por *transceivers GTH* [XIL14a].

Protocolo	Taxa de Transferência – Gbps
CEI 6G-SR	4,976 – 6,375
Interlaken	6,25
10GBASE-KR	10,3125
10GBASE-R	**10,3125**
XLAUI	10,3125
CEI-11	9,956 –11,1
CAUI	10,3125
OTU4	11,18, 12,5, 13,1
CPRI	0,6, 1,2, 2,4, 3,072, 4,9, 6,144, e 9,83
Gigabit Ethernet	1,25
OC-48	2,48832
OC-192	9,956
DisplayPort	1,620, 2,7, 5,4
JESD204	12,5
Optical-channel Transport Lane 3.4	10,7546
PCI Express Gen1	2,5
PCI Express Gen2	5
QSGMII	5
RXAUI	6,25
XAUI	3,125
Aurora 64B/66B	12,5
Aurora 8B/10B	6,6
Serial RapidIO Gen2	5,0

Esse Capítulo tem por objetivo apresentar os recursos disponíveis nesse dispositivo, apresentando as informações necessárias para o desenvolvimento de um projeto de interfaces de comunicação 10GBASE-R.

3.1 Recursos do Transceiver GTH

Os recursos dos *transceivers* GTH (Figura 11) abrangem as subcamadas PMA e PCS, porém com maior ênfase na camada PMA, já que esta contém os recursos de maior complexidade da interface de comunicação [FUH13]. A arquitetura do dispositivo para o padrão 10GBASE-R é baseada nas seguintes interfaces (os números de cada item estão indicados na Figura 11):

PMA:

1. **TX Configurable Driver:** driver de alta velocidade com recursos para maximizar a integridade do sinal de saída.
2. **Buffers do tipo PISO *(Parallel In/Serial Out)* e SIPO (*Serial In/Parallel Out):*** realizam a interface entre o meio serial e paralelo do sistema (serialização e de-serialização dos dados).
3. **PLL (*Phase Locked Loops):*** gerador do *clock* serial (alta frequência) e referência para os demais *clocks* do sistema (não representado na figura).
4. **RX/TX Clock Dividers:** divisores de *clock* para geração dos *clocks* necessários para o funcionamento do módulo.
5. **CDR (*Clock Data Recovery*):** recuperação do *clock* proveniente dos dados recebidos.
6. ***Pre-Emphasis (TX) e Equalization Blocks (RX):*** sistemas de compensação da degradação que ocorre nos sinais no canal óptico, como atenuação, reflexão e *crosstalk*.
7. ***Elastic Buffer* e *Phase Adjust FIFO***: buffers para transição entre domínios de *clock* interno distintos.

PCS:

8. ***Transmiter Gearbox:*** gerador de pacotes de 64 bits, formados a partir dos 66 bits recebidos da camada PCS devido a codificação 66b/64b. O controle dos pacotes (contador) é realizado através de um módulo externo ao *transceiver*.
9. ***Receiver Gearbox:*** gerador de pacotes de 66 bits, formados a partir dos pacotes de 64 bits provenientes da camada PMA, para a reconstituição dos pacotes codificados no padrão 66b/64b. O controle externo para determinação do alinhamento é realizado através de um módulo externo ao *transceiver*.

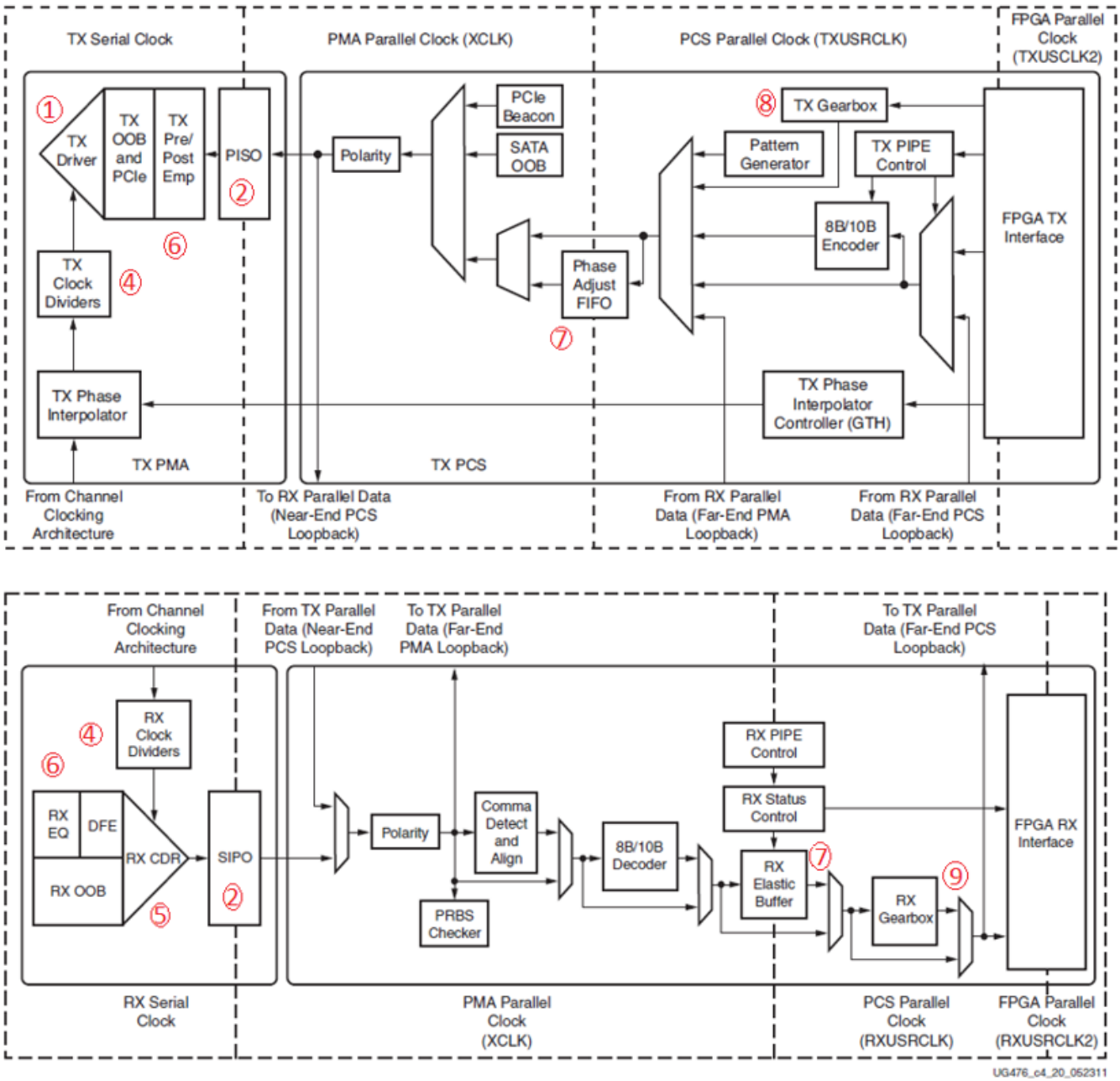


Figura 11 – Recursos do dispositivo GTH e domínios de Clock [XIL15a].

3.2 Encapsulamento dos Transceivers em dispositivos FPGAs

Os *transceivers* GTH são disponibilizados em bancos GTH Quad, os quais contêm quatro desses dispositivos. Cada *transceiver* constitui um canal denominado GTHE_CHANNEL, que contém todos os recursos necessários para o desenvolvimento dos protocolos descritos no tópico anterior. Além dos canais, cada banco possui um PLL (QPLL) de maior frequência de operação em comparação aos PLLs locais dos canais (CPLLs), e que é compartilhado entre os quatro canais do banco, esse denominado GTHE_COMMON. A Figura 12 apresenta um banco GTH Quad.

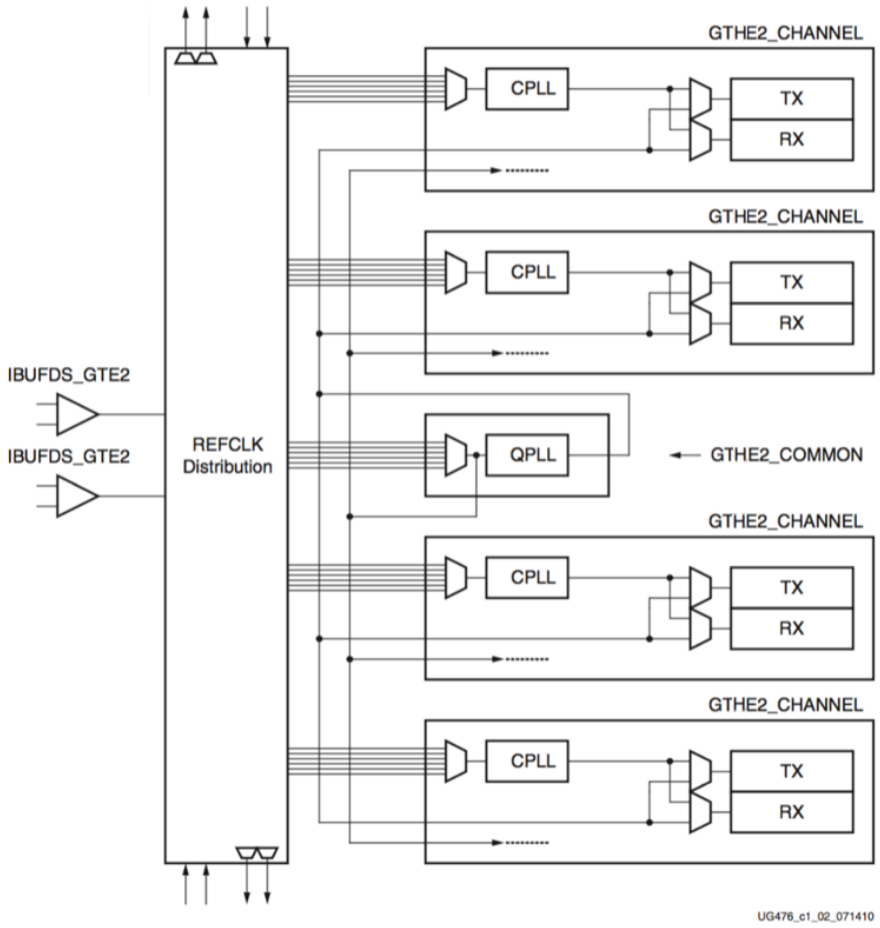


Figura 12 – Banco de Transceivers GTH [XIL15a].

3.3 Integração dos Transceivers GTH em Projetos de Hardware

Os módulos GTHE_CHANNEL e GTHE_COMMON são primitivas que podem ser instanciadas diretamente na descrição de hardware (HDL), pois esses são dispositivos já disponíveis no FPGA nos bancos GTH Quad. Entretanto, devido à grande quantidade de pinos de entrada e saída de cada módulo e as diferentes maneiras de configuração permitidas, sua configuração manual não é recomendada, pois demandaria grande quantidade de tempo e experiência sobre os recursos dos módulos. Como solução, a ferramenta Vivado disponibiliza uma interface (*wizard*) para geração automática das instanciações dos módulos necessários para o protocolo a ser desenvolvido. Além dos canais GTHE_COMMON e GTHE_CHANNEL propriamente configurados, também são fornecidos módulos auxiliares para controle da sequência de inicialização e resets do sistema e a geração dos *clocks* necessários para seu funcionamento. O *wizard* organiza as instanciações desses módulos no formato apresentado na Figura 13. As descrições dos principais módulos com referência em um projeto que utiliza o protocolo 10GBASE-R estão descritos abaixo:

- ***GT Wrapper*:** para cada GTHE_CHANNEL utilizado no protocolo é gerado um módulo com a instanciação para cada canal com a configuração dos parâmetros necessários para seu funcionamento.
- ***Multi GT Wrapper*:** módulo que engloba as instanciações de *GT Wrappers*.
- ***Initialization module*:** módulo com instanciações dos módulos *Multi GT Wrapper*, *TX/RX Reset FSM*.

- ***TX/RX Reset FSM*:** módulos de inicialização do transmissor e receptor de cada *transceiver*. É gerado um módulo de cada por canal GTHE2_CHANNEL.
- **Clock module:** módulo de geração dos *clocks* do sistema, com instanciação dos módulos MMCM (*Mixed-Mode Clock Manager*) [XIL09] e buffers diferenciais da referência de *clock* externa.
- **GT Common:** módulo com a instanciação do módulo GTHE2_COMMON.
- ***Common Reset*:** módulo de reset do GTHE2_COMMON.
- ***CSL*:** módulo com as instanciações de um banco de *transceivers* GTH.
- ***Core Top*:** topo das instanciações dos *transceivers* GTH.

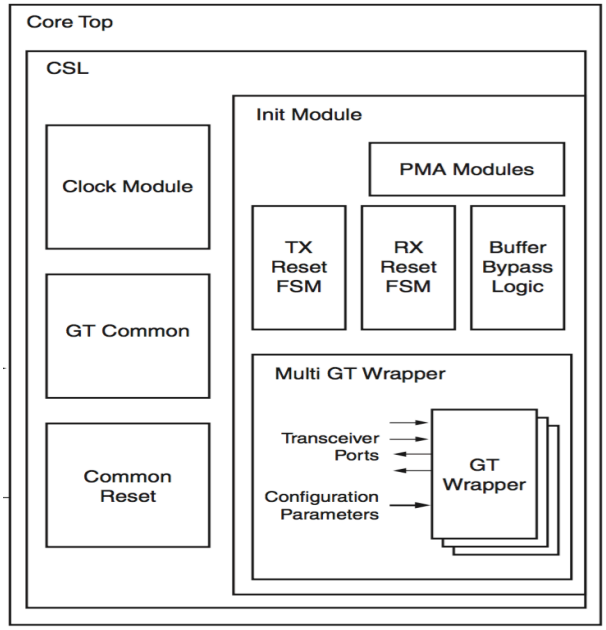


Figura 13 - Módulos instanciados pelo GTH Wizard.

O *wizard* de configuração é uma importante ferramenta para o desenvolvimento de projetos que utilizam *transceivers* GTH. No entanto, o projetista necessita de conhecimentos sobre o protocolo utilizado e os recursos necessários para implementação do mesmo, dentro dos recursos e limitações dos *transceivers*. Nesse contexto são necessários conhecimentos sobre os *clocks* utilizados para funcionamento do protocolo, a interface entre *transceiver* e a lógica desenvolvida na FPGA, o formato dos pacotes transmitidos e os processos de sincronismo e recepção dos dados recebidos. Por outro lado, os processos de tratamento de dados seriais, que são os de maior complexidade dentro da tecnologia envolvida, ficam transparentes ao projetista, salvo casos em que é necessário um maior controle sobre o hardware, e que demandam maior conhecimento sobre esses recursos. Os próximos tópicos desse documento apresentam os principais conceitos necessários para desenvolvimento de interfaces que utilizam transmissões de dados seriais a partir de *transceivers* GTH.

3.4 Referências de Clock

Devido à grande variedade de configurações necessárias para atender todos os protocolos suportados, cada canal GTHE_CHANNEL apresenta módulos configuráveis para a geração dos

diferentes *clocks* exigidos pelas interfaces de transmissão e recepção dos dados. A Figura 11 presenta as três regiões do sistema com domínios de *clock* distintos, para os dois sentidos de transmissão.

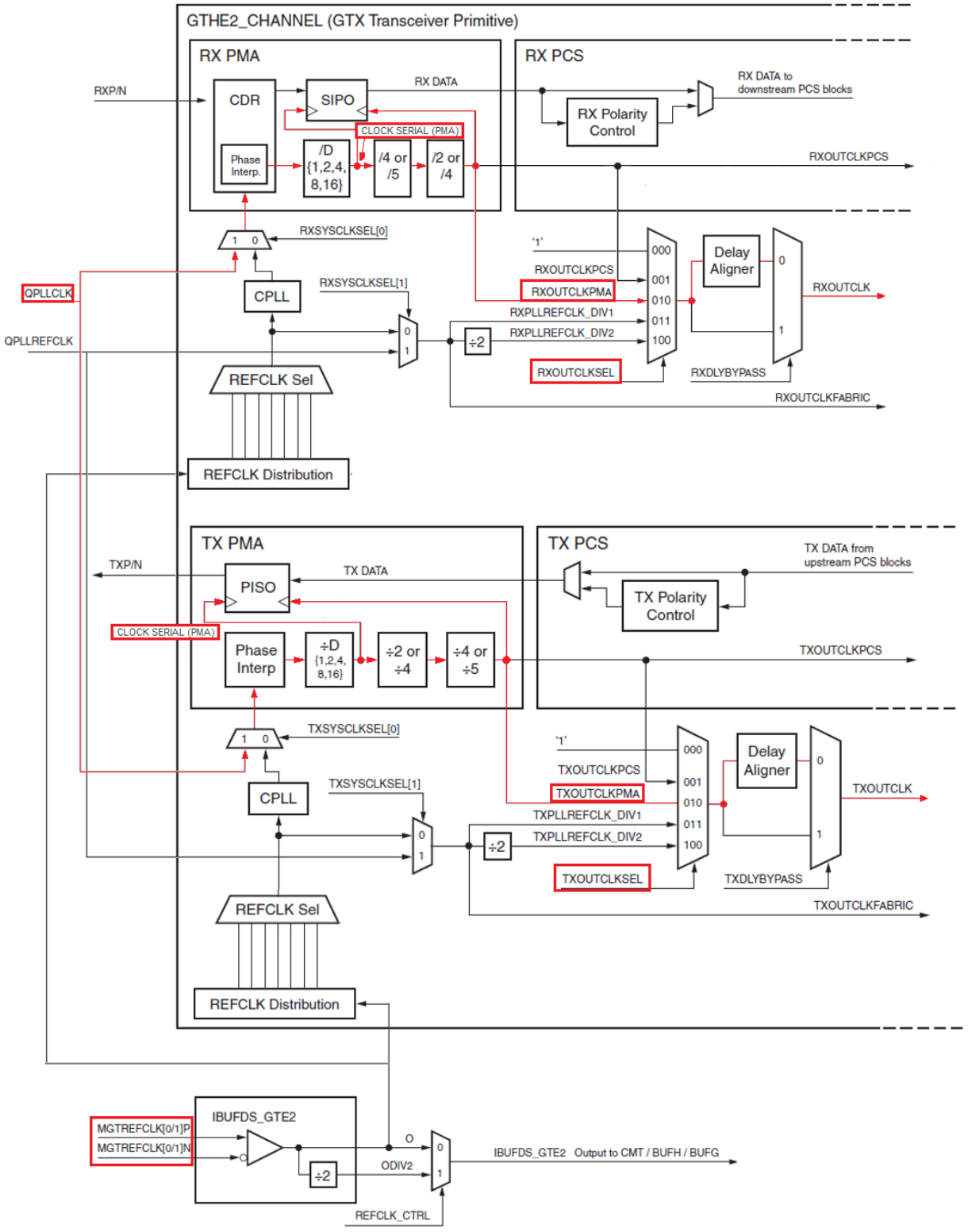


Figura 14 - Referência de clock internos – GTHE2_CHANNEL [XIL15a].

Essas referências internas são geradas a partir de um *clock* de referência externo à FPGA e é base para a construção dos domínios de *clock* dos canais. Esses domínios são divididos nos seguintes grupos:

- Referências de *Clock* Serial: *clocks* gerados internamente para referência da comunicação serial referentes à camada PMD.
- Referências de *Clock* Paralelo (XCLK): *clocks* gerados para referência da camada PCS e interface com a camada PMD.
- Referencias de *Clock* do Usuário (TXUSRCLK/RXUSRCLK): *clocks* gerados fora dos canais GTHE_CHANNEL, mas com referência de *clocks* gerados dentro dos canais para interface entre o canal do *transceiver* e a lógica da FPGA.

A construção dos diferentes domínios de *clock* é feita através de multiplexadores e divisores de *clock* configurados pelo usuário e controlados através de sinais de controle. Alguns dos sinais de controle podem ser alterados dinamicamente através da lógica desenvolvida na FPGA. Os próximos tópicos apresentam com mais detalhes os diferentes domínios de clock dos *transceivers*.

3.4.1 Referências de Clock Externas (MGTREFCLK)

Cada banco de *transceivers* possui duas entradas diferenciais para referência de *clock* externas ao FPGA e denominadas MGTREFCLK0_P/N e MGTREFCLK1_P/N. Dentre elas é escolhida uma como referência para cada canal. As referências podem ser utilizadas de forma independente pelos canais GTHE2_CHANNEL, ou através do módulo GTHE2_COMMON, esse permitindo a geração de uma referência compartilhada entre os canais. Como os *transceivers* são posicionados em formato de coluna (Figura 15), as referências de clock também podem ser compartilhadas entre os bancos adjacentes, ou seja, com os bancos acima (*south*) ou abaixo (*north*) do banco em uso. As referências externas são conectadas em um demultiplexador e selecionadas para o uso em PLLs dentro dos canais GTHE2_CHANNEL e GTHE2_COMMON para a geração de maior frequência do *transceiver* (*clock* serial).

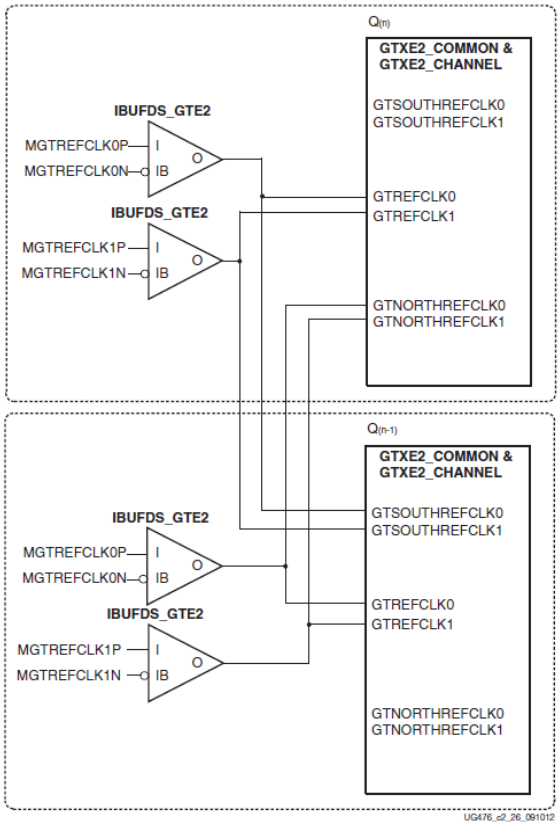


Figura 15 - Conexão do Clock de Referência Externa aos Canais [XIL15a].

3.4.2 Referência de Clock Serial (PMA)

A partir do clock de referência externo, é gerado o *clock* para a transmissão e recepção serial do dispositivo. Para isso, são utilizados PLLs internos (Figura 16) presentes nos canais GTHE_CHANNEL, os quais possuem um PLL do tipo *ring-based* (CPLL), capaz de gerar um *clock* com frequências que variam entre 1.6GHz e 5.16GHz. Outro recurso é a utilização do módulo GTHE_COMMON, esse é equipado com um PLL do tipo LC-Tank e que permite a geração de *clocks* de baixo jitter com frequências que variam entre 8.0GHz e 13.1GHz. [XIL15a].

O *clock* gerado através de uma das opções de PLLs disponíveis, é submetido a um módulo divisor de *clock*, para que protocolos com diferentes taxas de transmissão possam ser implementados a partir de uma mesma referência externa. Esse módulo divisor (D) permite as divisões do *clock* serial por 1, 2, 4, 8 ou 16. O *clock* resultante desse processo é conectado ao modulo PISO e SIPO e serve como referência para a transmissão e recepção de dados seriais respectivamente (Figura 14).

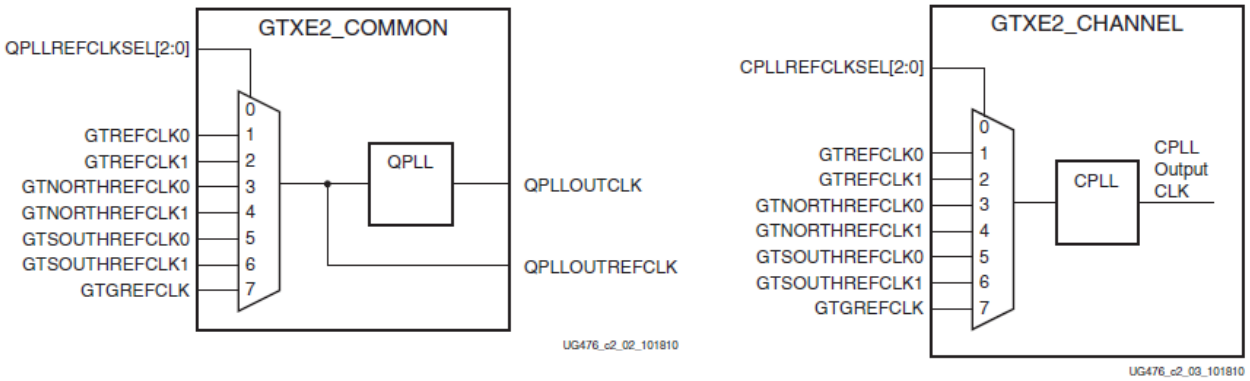


Figura 16 – Referência de Clock dos QPLLs e CPLLs [XIL15a].

No canal GTHE2_COMMON é necessário observar que, diferente do GTHE2_CHANNEL, o clock de referência gerado na saída do PLL é o valor configurado dividido por dois, devido a um divisor na saída do módulo.

O caminho destacado em vermelho na Figura 14 apresenta o caminho realizado pelo *clock* de referência externo gerado através QPLL do módulo GTHE2_COMMON, o qual alimenta o canal GTHE2_CHANNEL através do sinal QPLLCLK.

3.4.3 Referências de Clock Paralelo (XCLK)

Os *clocks* paralelos (XCLKs) definem as referências necessárias para a transferência de pacotes dados entre as interfaces internas do dispositivo. Dentre essas interfaces estão os módulos pertencentes à camada PCS e a lógica implementada na FPGA.

A referência de *clock* paralelo referente à camada PCS tem originem no *clock* serial, e é dividida através de dois módulos, já disponíveis em cada canal, para a redução da frequência de operação. Esses módulos permitem a divisão do *clock* serial pelos fatores 8, 10, 16 ou 20, gerando então a referência de *clock* paralelo para as interfaces de transmissão e recepção, denominados TXOUTCLKPMA e RXOUTCLKPMA respectivamente (Figura 14). O objetivo desse *clock* paralelo é fornecer uma referência, síncrona com o *clock* serial, para a transferência dos dados de forma paralela provenientes da serialização e de-serialização dos dados entre a camada PMA e PCS.

Outra referência de *clock* paralelo é necessária para a geração do clock do usuário, que em outras palavras, é o *clock* de sincronismo entre o canal do *transceiver* e a lógica desenvolvida na FPGA. Essa referência varia conforme o protocolo utilizado e se origina de *clocks* internos existentes no canal GTHE_CHANNEL. Essas referências podem ser originadas tanto da camada PMA (TX/RXOUTCLKPMA) e PCS (TX/RXOUTCLKPCS) quanto diretamente da referência externa utilizada no PLL. A seleção entre esses clocks é feita através de um de-multiplexador, e controlado através do sinal TX/RXOUTCLKSEL (Figura 14).

3.4.4 Referências de Clock do Usuário (Lógica da FPGA)

Os *clocks* do usuário são utilizados para sincronismo entre o *transceiver* e a interface desenvolvida. A geração desses *clocks* depende do protocolo implementado, e, para isso, podem ser utilizados diferentes recursos da FPGA. Como exemplo, a implementação sugerida pela Xilinx para o padrão 10GBASE-R utiliza um módulo MMCM (*Mixed-Mode Clock Manager*), conforme apresentado na Figura 17. Esse módulo é um recurso disponível em FPGAs para geração de múltiplos *clocks* e de diferente fase entre eles [XIL09].

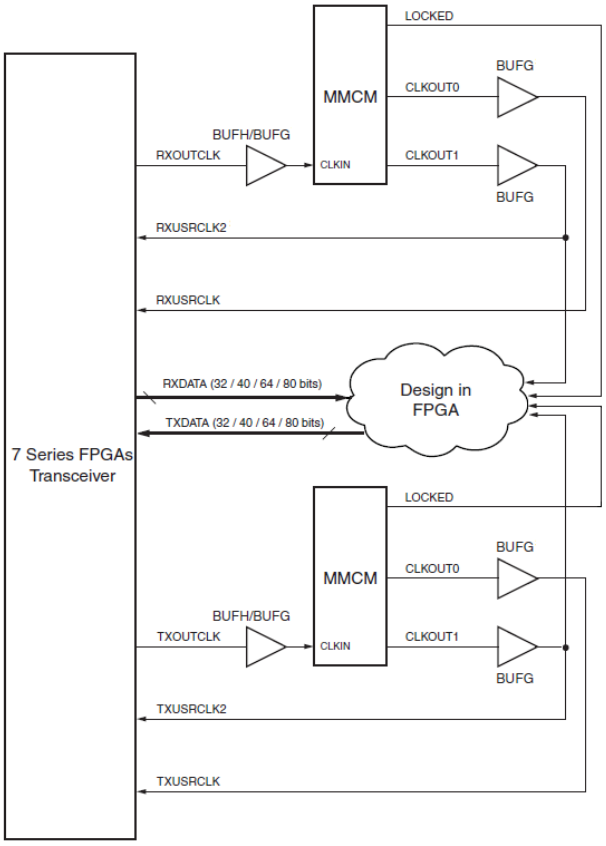


Figura 17 - Modelo de geração dos clocks do usuário [XIL15a].

Existem dois *clocks* de usuário que são necessários para cada sentido de transmissão, esses denominados TX/RXUSRCLK e TX/RXUSRCLK2. Estes sinais de clock precisam ser fornecidos ao canal do *transceiver* para que o mesmo possa ter o controle de fase entre os domínios XCLK e USRCLK. Os primeiros operam na mesma frequência *clock* paralelo da camada PCS (XCLK). Já o segundo, depende do tamanho do pacote a ser transferido para a lógica desenvolvida na FPGA. Caso o mesmo tamanho seja utilizado na interface da camada PCS, então as frequências TX/RXUSRCLK2 são iguais às frequências TX/RXUSRCLK. Caso contrário, o tamanho do pacote na lógica da FPGA

é o dobro do tamanho utilizado na camada PCS, logo as frequências TX/RXUSRCLK2 devem ser a metade das frequências TX/RXUSRCLK.

3.4.5 Clocks necessários para o Protocolo 10GBASE-R

Como apresentado anteriormente, para o desenvolvimento das diferentes camadas do protocolo Ethernet, são necessárias diferentes referências de *clock* devido à paralelização da transmissão de dados durante o processo. Considerando a interface XGMII com transmissão de pacotes de 64 bits, é necessário que as interfaces de codificação, decodificação, embaralhamento e desembaralhamento operem com *clock* de 156,25 MHz (Equação 1).

$$F_{xgmii} = \frac{10GHz}{64bits} = 156{,}25\ MHz \tag{1}$$

O uso da codificação 64b/66b causa um overhead de 3,125%, tornando-se necessário que a taxa transmissão compense essa diferença. Acrescendo este overhead à taxa de transmissão, conclui-se que é necessária uma taxa de transmissão de 10,3125 Gbps (2) para produzir uma carga útil de 10 Gbps, e que os módulos *gearbox* e *block synchronizer* descritos no protocolo 10GBASE-R (Seção 2.1.3.4) operem a 161,1328 MHz (3).

$$T_{10GbE} = \frac{10Gbps \times 66bits}{64bits} = 10{,}3125\ Gbps \tag{2}$$

$$F_{gearbox} = F_{blocksync} = \frac{10{,}3125Gbps}{64bits} = 161{,}1328\ MHz \tag{3}$$

Para determinar os valores dos *clocks* necessários, deve-se considerar as restrições impostas pelo hardware disponível para suas gerações. Primeiramente o PLL do canal GTHE_CHANNEL não consegue produzir um *clock* de referência para 10.3125 Gbps a partir de uma referência externa de 156.25 MHz, sendo então necessário o uso do canal PLL compartilhado GTHE_COMMON. Como visto anteriormente, o QPLL gera na sua saída metade da frequência nominal, ou seja, para o *clock* serial de 10,3125 GHz, a saída do PLL é 5,156 GHz. A partir desse, a menor redução para o clock paralelo (XCLK) compatível com o protocolo 10GBASE-R é a divisão por 16, gerando um clock de frequência 322,2265 MHz e uma interface de 32 bits. Por último, os *clocks* do usuário TX/RXUSRCLK e TX/RXUSRCLK2 são gerados com as referências de 322,2265 MHz e 161,1132 MHz respectivamente. A Tabela 2 apresenta as referências de *clock* para o protocolo 10GBASE-R.

Tabela 2 - Domínios de Clock para o Protocolo 10GBASE-R.

Referência de Clock		Frequência de Operação
Clock Externo		156,25 MHz
Clock Serial		5,156 GHz
Clock Paralelo (XCLK)		322,265625 MHz
Clock do Usuário	USRCLK	322,265625 MHz
	USRCLK2	161,1328125 MHz

3.5 Sequência de Inicialização

O módulo do *transceiver* gera, através de uma máquina de estados, a sequência de inicialização dos recursos necessários para o seu funcionamento (Figura 18). Os sinais TX_RESET_FSM_DONE e RX_RESET_FSM_DONE notificam quando a cadeia de inicialização foi concluída para os sentidos de transmissão e recepção respectivamente. Essa sequência verifica se os *clocks* gerados pelo PLL e que são necessários para o funcionamento do *transceiver* estão estáveis. Na inicialização do receptor também é verificado se o *clock* proveniente dos dados recebidos pelo CDR [XIL15a].

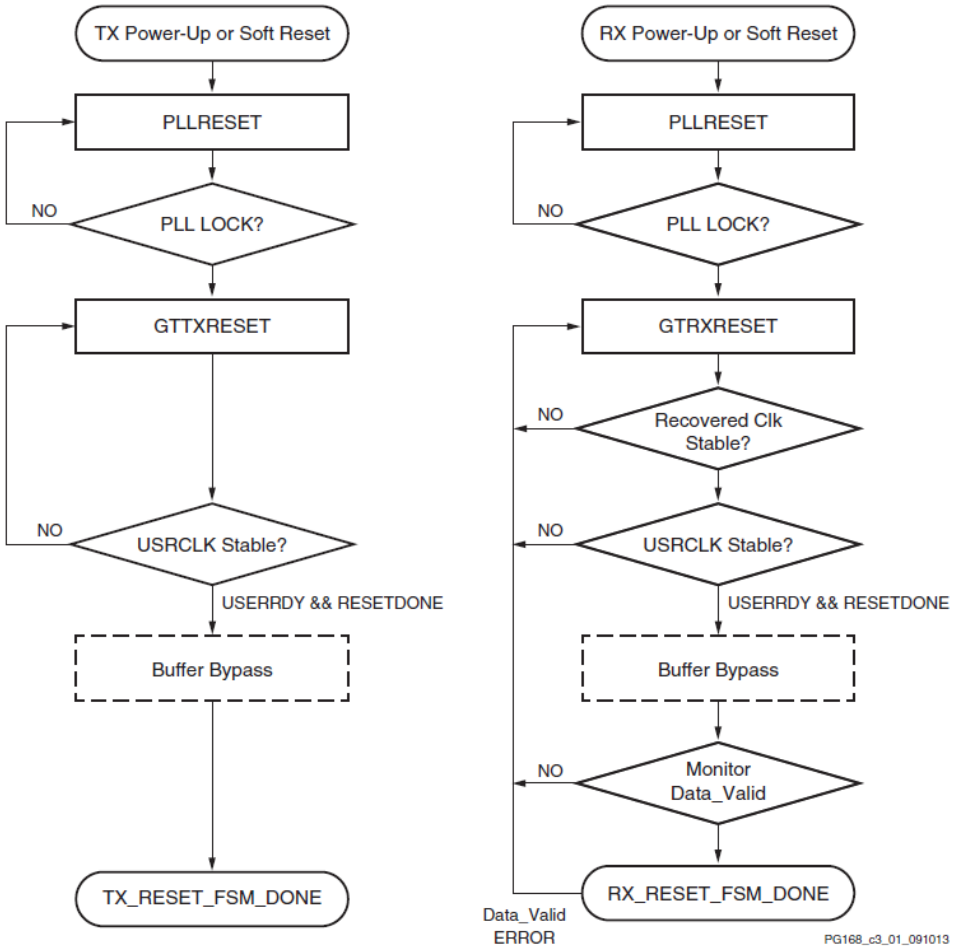


Figura 18 – Sequência de Resets do Transceiver GTH [XIL15a].

3.6 Transmissão de Pacotes

No processo de transmissão de dados, os pacotes necessitam estar codificados no padrão 64b/66b e embaralhados, antes de sua inserção no módulo do *transceiver*. Estes processos são de responsabilidade da interface a ser desenvolvida. O header do pacote é inserido na entrada TXHEADER[1:0] e o pacote de 64 bits na entrada TXDATA[63:0], ambos síncronos com o *clock* TXUSRCLK2. É necessário um contador externo para identificação da ordem dos pacotes enviados pelo TX *Gearbox*. A Figura 19 apresenta os recursos necessários para envio de pacotes para o TX Gearbox [XIL15a].

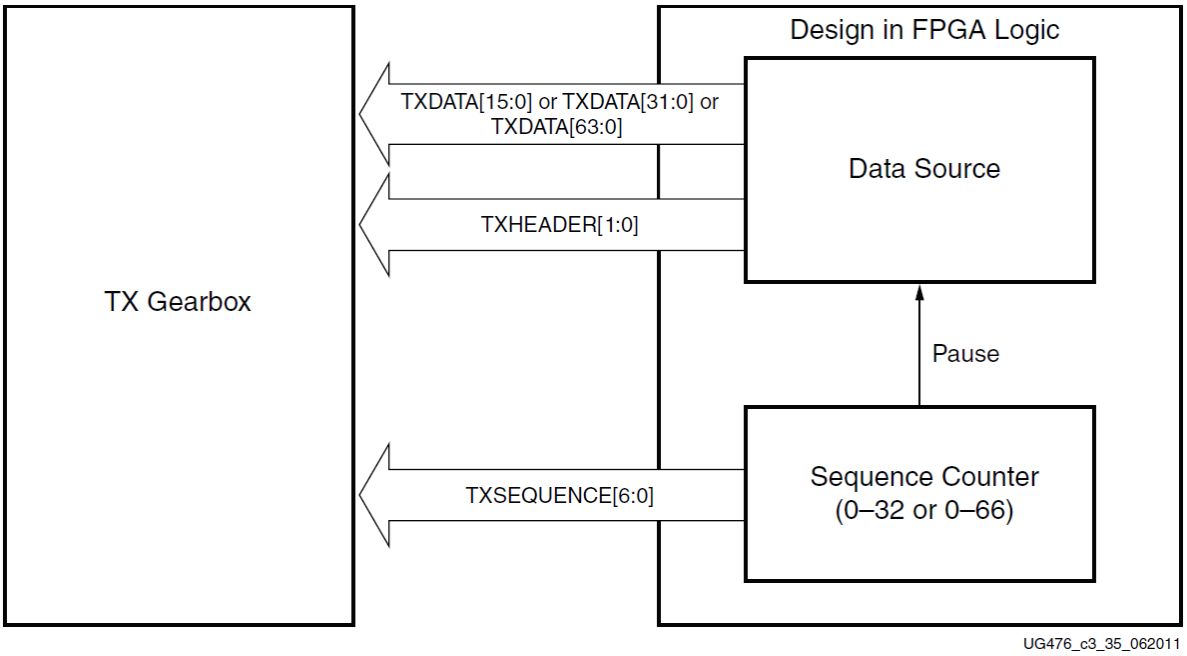


Figura 19 – Diagrama de transmissão do transceiver GTH [XIL15a].

Para o envio de pacotes de 64 bits, o contador necessita ser incrementado a cada pacote de dado enviado, até que 32 pacotes sejam enviados. Nesse instante é feita uma pausa no envio de pacotes ao *TX Gearbox*, permitindo que os 64 bits excedentes da redução realizada no *TX Gearbox*, e que foram causados pela codificação 66b/64b sejam enviados. Como podemos observar na Figura 20, no ciclo 32 os dados presentes no registrador TXDATA são ignorados [XIL15a].

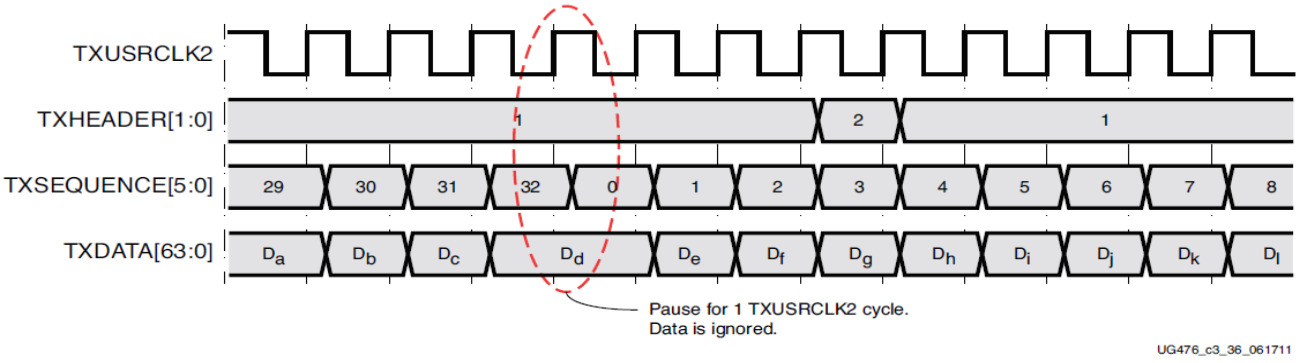


Figura 20 – Exemplo de pausa do processo de transmissão do transceiver GTH [XIL15a].

3.7 Recepção de pacotes

Os pacotes de dados recebidos pelo *transceiver* estão codificados e embaralhados no padrão 64b/66b. Cabe a interface desenvolvida o desembaralhamento e decodificação desses dados. O header e o pacote de dados são transferidos, pelo módulo *RX Gearbox,* através dos vetores RXHEADER[1:0] e RXDATA[63:0] respectivamente. Esses são sincronizados com o *clock* RXUSRCLK2. O módulo do *transceiver* também informa se o header e o pacote de dados são válidos através dos sinais RXHEADERVALID e RXDATAVALID respectivamente. A Figura 21 apresenta a interface entre a lógica interna do módulo e a recepção dos dados pela lógica desenvolvida [XIL15a].

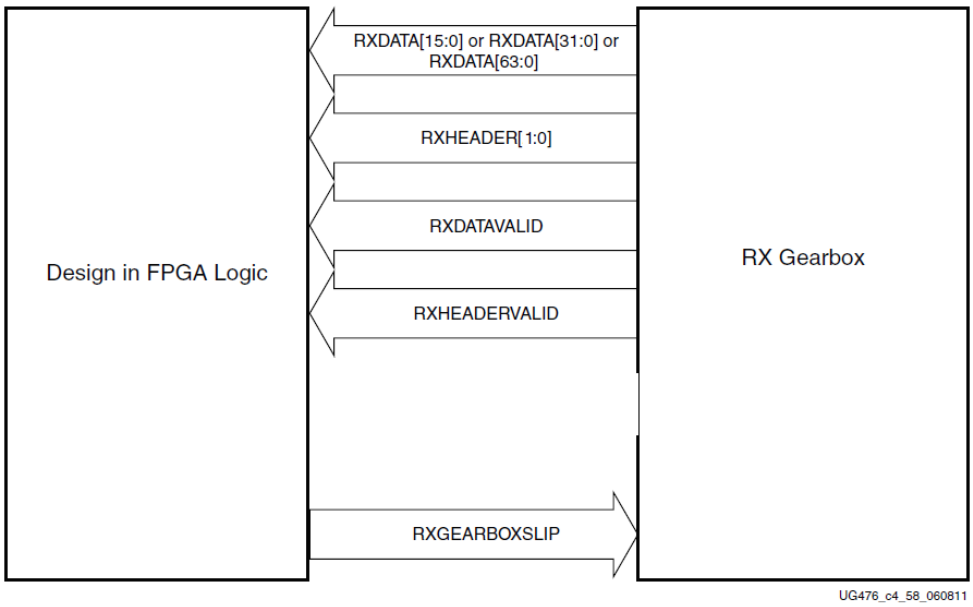


Figura 21 – Diagrama de transmissão do transceiver GTH [XIL15a].

Durante a inicialização do receptor, é necessário que o módulo de sincronismo de blocos (*Block Synchronizer*), desenvolvido na lógica da FPGA, encontre o alinhamento correto dos pacotes provenientes do módulo *RX Gearbox*, esse pertencente ao canal do *transceiver*. A Figura 22 apresenta o processo de alinhamento dos dados. Durante esse processo o *Block Synchronizer* busca no cabeçalho RXHEADER os valores '10' e '01' (cabeçalhos válidos). Quando os valores '11' ou '00' são encontrados, o sincronizador gera um pulso de um período de *clock* no sinal RXGEARBOX, para que o *RX Gearbox* desloque a ordem dos dados, pois a ordem atual dos pacotes transferidos não está correta. Quando uma sequência suficientemente grande de cabeçalhos válidos é encontrada, o *Block Synchronizer* indica aos demais módulos de recepção que o bloco de dados recebidos está alinhado através do sinal *block_sync*. [XIL15a].

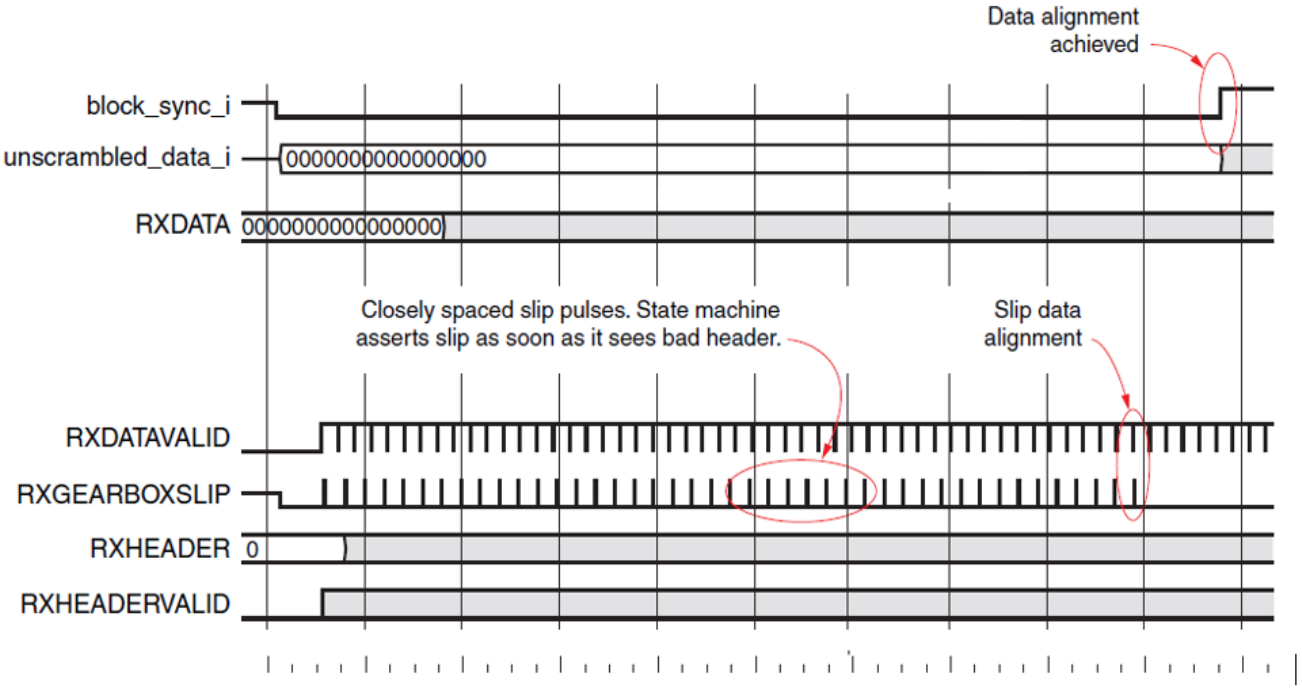


Figura 22 - Exemplo de processo de alinhamento de dados [XIL15a]

Assim como no processo de transmissão, o *RX Gearbox* realiza uma pausa no envio de dados a cada 32 ciclos de *clock*. Essa pausa é indicada através do sinal RXDATAVALID, e nesse período, os dados recebidos devem ser ignorados pelo receptor, inclusive na fase de alinhamento dos pacotes realizada através do *Block Synchronizer*. A Figura 23 apresenta o processo de pausa na recepção de dados [XIL15a].

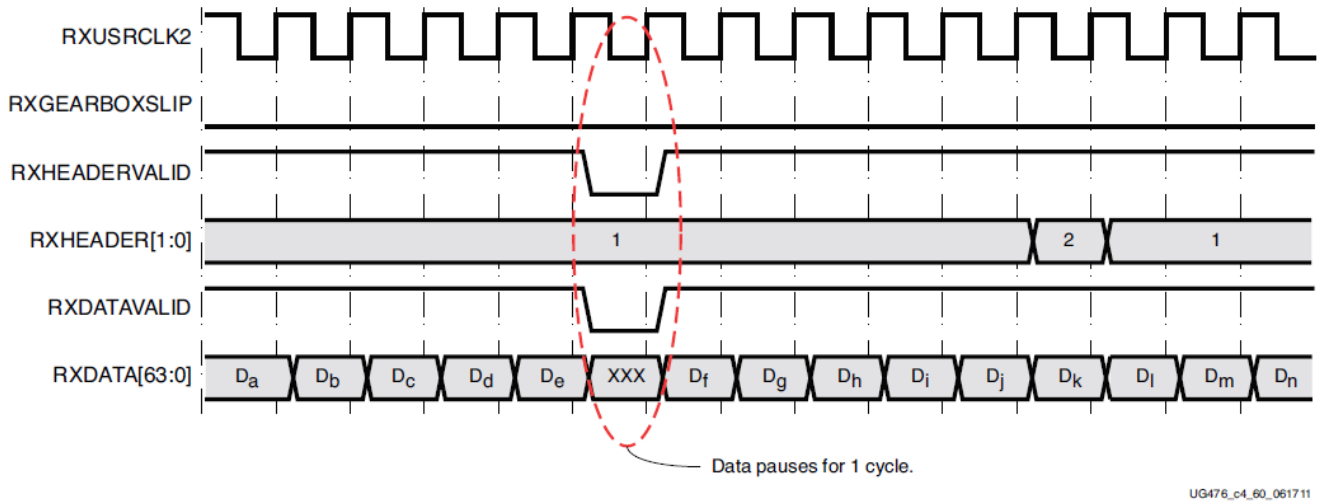


Figura 23 – Exemplo de recepção de dado inválido [XIL15a].

4 FLUXO DE PROJETO UTILIZANDO TRANSCEIVERS GTH

Este Capítulo apresenta o fluxo de projeto utilizado no desenvolvimento de interfaces que utilizam transceivers GTH. A ferramenta utilizada no desenvolvimento do projeto é o Vivado Design Tool 2015.2 desenvolvida pela empresa Xilinx Inc. As informações trazidas nesse Capítulo são referentes a um projeto que utiliza como plataforma de desenvolvimento a placa NetFPGA Sume [DIG15]. O dispositivo FPGA utilizado nessa placa é o Virtex 7 XCV690T.

4.1 Desenvolvimento do Projeto (Design Entry)

O primeiro passo no desenvolvimento de um projeto de hardware é a sua descrição lógica a partir de uma linguagem de descrição de hardware (HDL) ou através de um esquemático do projeto.

A partir da descrição de hardware é desenvolvido o projeto do sistema do usuário. Nesta etapa os diversos módulos com funções específicas que são necessários são integrados em um projeto único. Através da compilação da captura de projeto podem ser verificados erros de sintaxe e violações lógicas da implementação, facilitando o desenvolvimento do projeto.

Para auxiliar no projeto de desenvolvimento do projeto de entrada, a ferramenta Vivado disponibiliza módulos básicos, denominados IPs (*Intellectual Properties*), que já descrevem componentes específicos, podendo então serem diretamente instanciados dentro do projeto [XIL14a]. No desenvolvimento de um projeto que utiliza *transceivers* GTH, a utilização de IPs disponibilizados pela ferramenta é essencial para o desenvolvimento do projeto, pois devido à complexidade dos mesmos seria extremamente custoso desenvolver com esse tipo de recurso.

4.1.1 LogicCore IP 7 Series Transceiver Wizard

A ferramenta Vivado, apresenta uma solução para configuração de *transceivers* denominada *LogicCore IP 7 Series* Transceiver *Wizard*. Essa interface cria automaticamente módulos IPs configurados e conectados à *transceivers* GTH para diferentes tipos de protocolo como apresentado no Capítulo 3 desse documento. Os passos para configuração são simples, mas demandam conhecimento sobre o protocolo utilizado para a correta configuração dos recursos necessários [XIL14a].

A sequência de passos para configuração dos *transceivers* é apresentada na Figura 24, e consiste na seleção do protocolo, escolha dos *transceivers* a serem utilizados conforme a localização física na placa, seleção dos pinos que fornecem os *clocks* de referência e, se necessário, ajuste de parâmetros específicos para a aplicação, como pinos de controle opcionais. Finalizadas as configurações do *transceiver* via *wizard*, o IP GTH *Wrapper* é gerado e pode ser instanciado diretamente na interface desejada. Para facilitar, um exemplo é fornecido com base no protocolo do módulo gerado, facilitando a compreensão de funcionamento do mesmo [XIL14a].

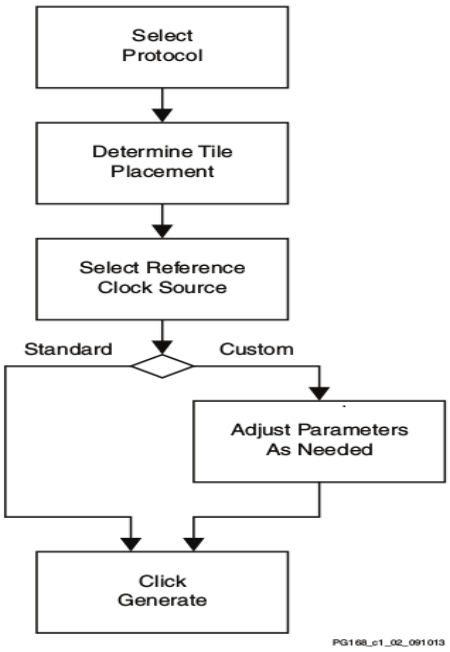


Figura 24 – Sequência para geração do IP GTH Wrapper [XIL14a].

4.1.2 Exemplo de Projeto com Transceiver GTH no padrão 10GBASE-R

O exemplo disponibilizado pelo Transceiver W*izard* auxilia a verificação de funcionamento do módulo gerado através do mesmo. Além do módulo principal (GTH Wrapper), são instanciados os componentes necessários para a geração, envio, recebimento e verificação dos pacotes. Também é disponibilizado um *testbench* para simulação lógica, permitindo a validação do sistema. Para a sequência do fluxo de projeto, será utilizado como referência um módulo gerado a partir do Wizard com a instanciação de um *transceiver* GTH, configurado para o protocolo 10GBASE-R [XIL14a].

Para validação do fluxo de dados do GTH Wrapper, o exemplo provê os seguintes módulos (Figura 25):

- ***GTH Wrapper**:* Instanciação do core do *transceiver* a ser utilizado no desenvolvimento do projeto.
- ***Frame Generator**:* lê pacotes pré-determinados armazenados em uma BRAM (Block RAM) e envia ao embaralhador. Os pacotes de dados armazenados já estão codificados no padrão 66b/64b.
- ***Scrambler**:* embaralha os 64 bits de dados provenientes do *frame generator*.
- ***Descrambler**:* desembaralha os pacotes recebidos pelo módulo do *transceiver*.
- ***Frame Synchronizer**:* valida os *headers* recebidos pelo módulo do *transceiver* e determina quando o sincronismo de blocos foi alcançado.
- ***Frame Checker**:* verifica se os dados recebidos são idênticos aos dados enviados.
- ***Sequence Counter**:* contador de pacotes enviados para determinação da posição do cabeçalho e pausa do envio de pacotes.
- ***Reset FSM**:* monitora o término da inicialização do GTH *Wrapper* e dá início ao fluxo de dados.

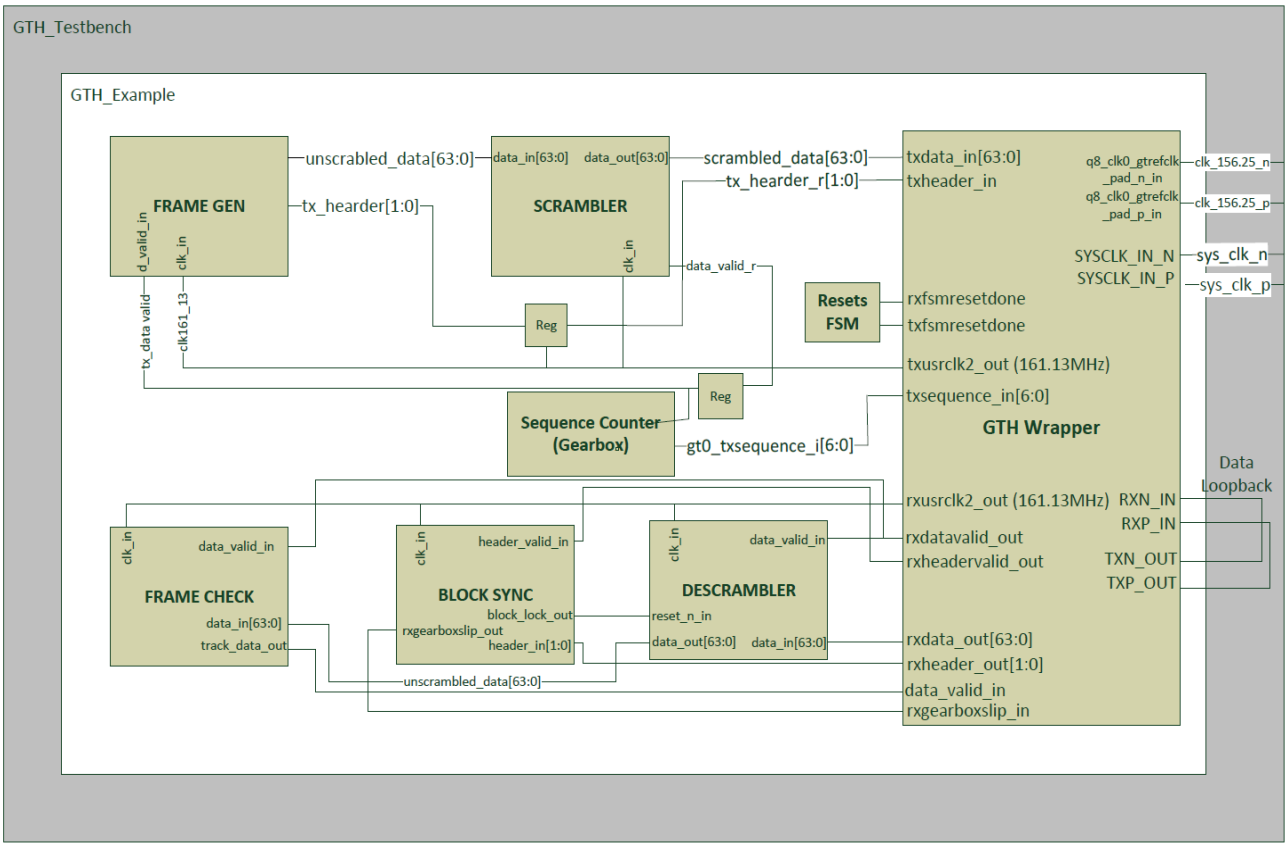


Figura 25 – Exemplo de projeto 10GBASE-R [XIL14a].

4.2 Simulação do Projeto

Após a correta compilação do projeto, é possível verificar o comportamento lógico do sistema através de simulação. O Vivado permite a simulação do projeto através de um *testbench*, código que instancia o topo do projeto e gera estímulos necessários para verificação do comportamento lógico do sistema. Abaixo são apresentados pontos da simulação do exemplo do módulo GTH *Wrapper* [XIL14a].

Como ilustração de simulação do exemplos do GTH *Wrapper* a Figura 26, Figura 27 e Figura 28 apresentam a verificação de pausa do processo de transmissão, o ajuste de alinhamento do RX *Gearbox* e a verificação do ciclo de pausa na recepção de dados no receptor, respectivamente. Esse processo permite validar os recursos dos *transceivers* abordados no capítulo 3, antes de iniciar o processo de desenvolvimento com o módulo.

Na Figura 26 pode-se observar a realização da pausa necessária do processo de transmissão. Essa pausa é realizada no ciclo 32 do *TX Gearbox*, e controlado através do sinal *gt0_txsequence*. Nesse período o sinal *gt0_pause_data_valid_r* indica que o processo de envio deve ser pausado, e que o pacote da interface *gt0_txdata_i* será ignorado.

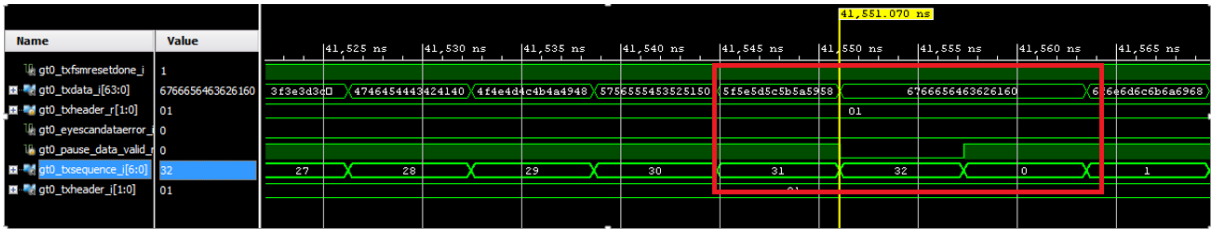


Figura 26 – Verificação da pausa de envio do transmissor.

Na Figura 27 é apresentado o processo de sincronismo do módulo *RX Gearbox* realizado através do módulo BLOCK_SYNC (Figura 25). No instante 1 o sinal *gt0_block_lock_i* vai a zero indicando perda de sincronismo dos pacotes recebidos, dando início no processo realinhamento. O módulo gera um pulso no sinal *gt0_rxgearboxslip_i*, para que o módulo *RX Gearbox* desloque os pacotes dados recebidos, em uma nova tentativa de alinhamento. Durante o passo 2, os cabeçalhos recebidos são verificados, e quando detectado um cabeçalho inválido é dado um no novo pulso no sinal *gt0_rxgearboxslip_i*. Quando uma sequência de cabeçalhos válidos for detectada, o sinal *gt0_block_lock_i* volta à '1', indicando que os dados estão alinhados.

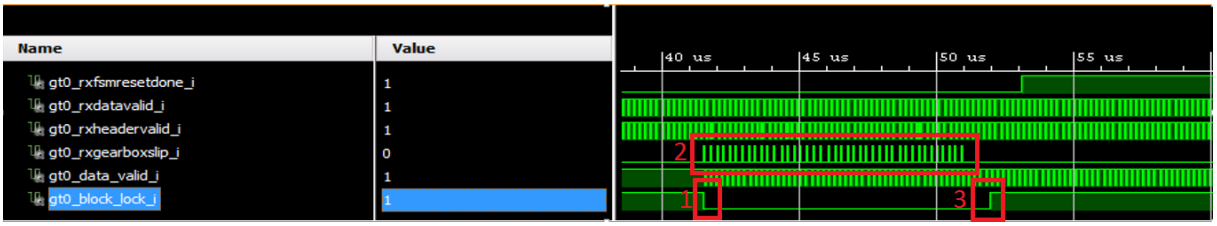


Figura 27 – Processo de sincronismo do RX Gearbox.

Na Figura 28 é apresentado o processo de pausa no sentido da recepção. No instante destacado em vermelho, pode-se observar que o sinal *gr0_rxdata_valid_i* indica o momento em que a pausa na recepção deve ocorrer, pois o pacote de dados transferido pelo barramento *gt0_rxdata_i* não deve ser utilizado.

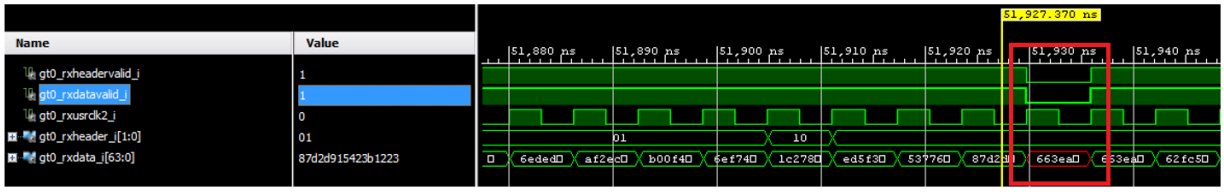


Figura 28 – Verificação do ciclo de pausa na recepção de pacotes.

4.3 Restrições de Projeto – *Constraints*

Restrições são parâmetros adicionados ao projeto para que os processos de síntese levem em consideração condições críticas que precisam ser atendidas, no momento em que o circuito é gerado. Esses parâmetros são adicionados a um arquivo com extensão XDC (*Xilinx Design Constraints*) ou através da ferramenta Vivado, onde os parâmetros são incluídos no arquivo de forma automaticamente, de acordo com as configurações realizadas através da interface gráfica. Os demais detalhes sobre os tipos de *constraints* existentes são abordados nos tópicos relacionados a síntese lógica e física.

4.4 Síntese Lógica

A síntese lógica gera a descrição do circuito no nível de portas lógicas e as conexões necessárias para rotear o circuito (*netlist*) a partir do projeto desenvolvido. Entretanto, independentemente do projeto, a descrição obtida inicialmente tem grandes probabilidades de não ser a mais otimizada e inclusive pode não atender os requisitos de temporização do sistema. Para solucionar esse problema, *constraints* podem ser utilizadas para prover informações de requisitos temporais que devem ser atendidos, para que a ferramenta sintetize o circuito atendendo a esses requisitos [BRO09].

4.4.1 Constraints: Síntese Lógica

As *constraints* são utilizadas no processo de síntese lógica, e determinam requisitos temporais para os circuitos. A partir das referências de *setup* e *hold* dos registradores a ferramenta de síntese determina o *slack time* dos caminhos críticos. O *slack time* é o tempo de sobra para que os dados consigam ser transferidos de um registrador para o outro antes da borda da borda de *clock* do domínio utilizado. Um *slack time* negativo representa violações de temporalização [XIL13c]. Dentro das *constraints* de tempo utilizadas destacam-se:

- ***create_clock:*** define a frequência de uma referência de *clock* primária do sistema.
- ***create_generated_clocks:*** define a frequência de uma referências de *clock* originadas de uma referência primária do sistema.
- ***create_false_path:*** define um caminho assíncrono entre dois pontos, portanto que não constituem um caminho crítico.
- ***set_input_delay:*** define um atraso de fase entre uma determinada borda de *clock* da interface e sinal recebido em um pino de entrada da FPGA.
- ***set_output_delay:*** define um atraso de fase entre uma determinada borda de *clock* da interface e o sinal recebido em um pino de saída da FPGA.

4.5 Síntese Física

A síntese física é responsável por implementar o circuito gerado pela síntese lógica em um determinado dispositivo FPGA. A implementação varia conforme o tipo de dispositivo utilizado, pois os recursos variam conforme o modelo do mesmo. Nessa etapa são posicionados os módulos desenvolvidos em determinadas regiões da FPGA e roteados os caminhos entre os circuitos lógicos. Esse processo é feito através de diferentes algoritmos pertencentes a ferramenta de síntese. Também são realizadas verificações de desempenho, ocupação de área, DRC (*Design Rule Check*) e a estimativa de dissipação de potência também nesta etapa, e os resultados são reportados ao projetista, que determinará se os requisitos de projeto foram atingidos. Caso a síntese não gerar um resultado satisfatório, recursos como *floorplaning* e realocação dos módulos através de *contraints* podem ser utilizadas para reposicionar a lógica dentro da FPGA de forma a facilitar uma nova execução de síntese [XIL13b].

4.5.1 Constraints: Síntese Física

A especificação das *constraints* físicas são pertinentes à localização dos pinos de entrada e saída da FPGA e dos módulos GTHE_CHANNEL, GTHE_COMMON, MMCMs, BRAMs, LUTs e registradores disponíveis internamente. A maioria das *contraints* físicas são definidas a partir do comando *set_property* [XIL13a].

4.5.2 Floorplanning

O processo de *floorplaning* tem por objetivo posicionar manualmente blocos de lógica desenvolvidas ao longo do projeto no FPGA com intuito de reduzir o atraso de caminhos críticos. Com esta ferramenta é possível determinar a posição de cada módulo dentro do FPGA. Normalmente

esse recurso é utilizado quando os algoritmos da ferramenta de síntese não consegue gerar um resultado que garanta os requisitos temporais do projeto.

4.6 Prototipação

A partir de uma síntese física que atenda aos requisitos do projeto, o passo seguinte é a geração do arquivo *bitstream* de programação do dispositivo. A partir da programação, o FPGA é configurado com o circuito desenvolvido e seu funcionamento pode ser verificado fisicamente.

4.6.1 Verificação lógica através da ferramenta Chipscope

A verificação física pode ser realizada através da ferramenta Chipscope. Esse recurso é fornecido pela ferramenta Vivado e permite a inserção de circuitos que permitem a leitura e escrita dos registradores do sistema em tempo real.

O Chipscope funciona em duas fases. Durante a fase de síntese são inseridos pontos de observação do hardware. A partir desses pontos a ferramenta gera estruturas de monitoração e armazenamento de dados. Após a configuração da FPGA os pontos podem ser monitorados a partir de um computador conectado ao FPGA, permitindo a visualização dos sinais em tempo real no dispositivo. A leitura dos registradores é feita através da configuração de *triggers, os quais* são ativados a partir de um evento determinado.

5 INFRAESTRUTURA DE HARDWARE

Os Capítulos anteriores corresponderam ao estudo necessário para a execução do presente projeto. Este Capítulo apresenta a *principal contribuição* deste TCC, que é o desenvolvimento do hardware contendo as interfaces de alta velocidade.

Este Capítulo apresenta as interfaces de hardware necessárias para o funcionamento e teste de um módulo Ethernet 10GBASE-R que utiliza como recurso *transceivers* GTH. Dentro dessas interfaces estão os módulos para comunicação Ethernet, os módulos auxiliares para gerenciamento do sistema, configuração, geração de pacotes para teste e de comunicação com usuário.

O projeto foi desenvolvendo em uma placa NetFPGA Sume, a qual utiliza um dispositivo FPGA de modelo Virtex 7 XC7VX690T, com capacidade de até quatro interfaces 10GbE através de dispositivos ópticos SFP+, esses conectados a *transceivers* GTH [ATH15]. O diagrama de projeto com a interface para um canal 10GbE e seus respectivos módulos de teste, gerenciamento e configuração são apresentados na Figura 29. As funções dos módulos apresentados serão descritas ao longo deste Capítulo.

O circuito conectado à interface de alta velocidade é denominado XGETH_TESTER, que tem por objetivo o teste de redes 10 GbE de acordo com vários critérios de desempenho. O trabalho consiste em não apenas desenvolver a interface de alta velocidade, mas também compreender sua interface com o projeto do usuário, e realizar as modificações necessárias para a integração com a nova interface de alta velocidade.

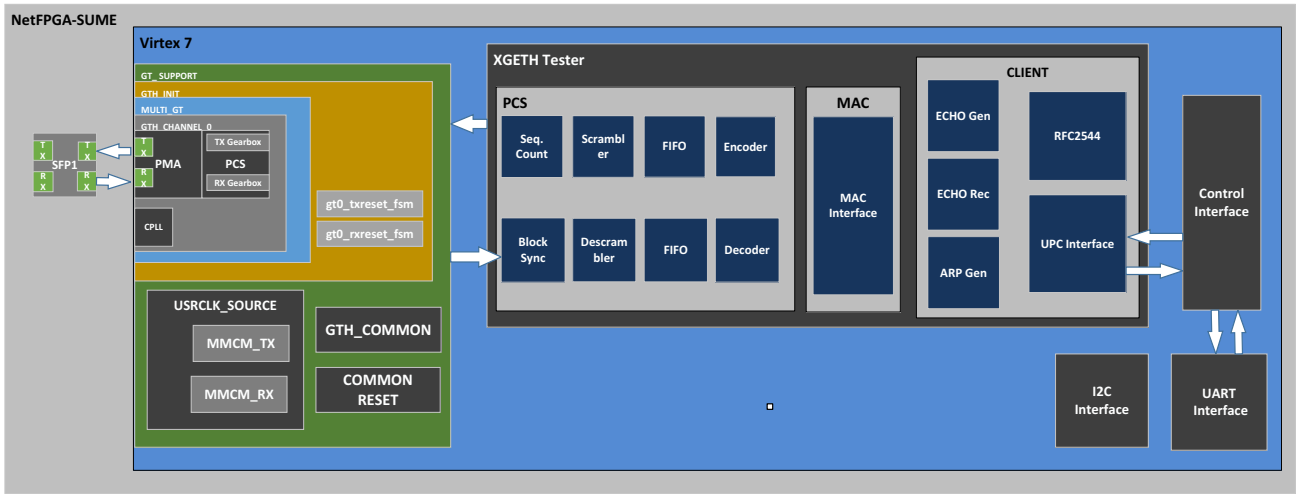


Figura 29 – Diagrama do projeto da interface 10GbE.

5.1 Interface 10GBASE-R

A interface 10GBASE-R foi desenvolvida através da integração dos módulos GTH_gtwizard e XGETH_TESTER. O primeiro foi criado através do *Wizard* apresentado na Seção 4.1.1, para configuração e inclusão de um *transceiver GTH* no projeto. Esse módulo foi integrado com o módulo XGETH_TESTER, desenvolvido em projeto de parceria entre a empresa Teracom e o grupo de apoio ao projeto de hadware (GAPH), e que consiste em um testador para interfaces 10 Gigabit Ethernet. O

testador inclui os módulos PCS e MAC, pertencentes as camadas física e de enlace respectivamente, entretanto. Entretanto, foram necessárias adequações deste dispositivo, devido a integração com os *transceivers* GTH.

Esta Seção aborda os requisitos de projeto necessários para a configuração e posicionamento do *transceiver* GTH de acordo com a posição do SFP+ da placa NetFPGA SUME. Também é apresentado o processo integração entre os módulos GTH_gtwizard e XGETH_TESTER.

5.1.1 Módulo GTH_gtwizard

Para a configuração e geração do módulo GTH_gtwizard através do *Wizard* de configurações, são necessários os conhecimentos sobre a posição física do banco o qual módulos SFP+ estão conectados e a referência de *clock* externa, conforme o roteamento físico dos recursos na placa NetFPGA SUME. Nesta Seção também são apresentadas as configurações necessárias para o funcionamento de um *transceiver* no padrão 10GBASE.

5.1.1.1 Localização do banco de transceivers GTH conectados aos SFP+.

Segundo o esquemático da placa (Figura 30) os dispositivos SFP+ estão localizados no banco 119, o que indica a área X1Y9 da FPGA. Já a referência de clock externa é proveniente do banco X1Y8 através da entrada diferencial MGTREFCLOCK0P/N. A utilização dessa referência só é possível devido aos recursos de *clock* poderem ser compartilhados com os bancos adjacentes, como apresentado na Seção 3.4.1.

5.1.1.2 Configuração do transceiver através do Wizard

Nesta etapa são configurados os recursos de funcionamento do *transceiver* GTH necessários para o seu funcionamento no protocolo 10GBASE-R. Inicialmente são determinados o protocolo utilizado, as taxas de transferência e a localização dos *transceivers* a serem utilizados (Figura 31).

Quando selecionado o protocolo desejado, as configurações padrão para o protocolo já são automaticamente configuradas, sendo necessário apenas um ajuste fino. Para o padrão 10GBASE-R são automaticamente selecionados a taxa de transferência de 10.3125Gbps e o *clock* de referência externo de 156.25MHz. Também é selecionado automaticamente o recurso GTHE_COMMON, pois o CPLL (PLL interno do canal) não possui os recursos necessários para atingir a frequência necessária do *clock* de referência serial a partir de uma referência de 156.25MHz. O menor *clock* exigido nesse caso seria 206.25MHz.

Nessa página de configuração também são selecionados os *transceivers* que são habilitados de acordo com a sua localização física. Para esse projeto foi selecionado o primeiro *transceiver* do banco X1Y9 ou Quad9, localizado na posição X1Y39. Além da posição, é necessário especificar a origem do *clock* de referência para cada *transceiver* habilitado. Nesse caso é preciso selecionar o *clock* proveniente do pino MGTREFCLK0_Q8, localizado no banco Quad8, de acordo com o esquemático da NetFPGA SUME.

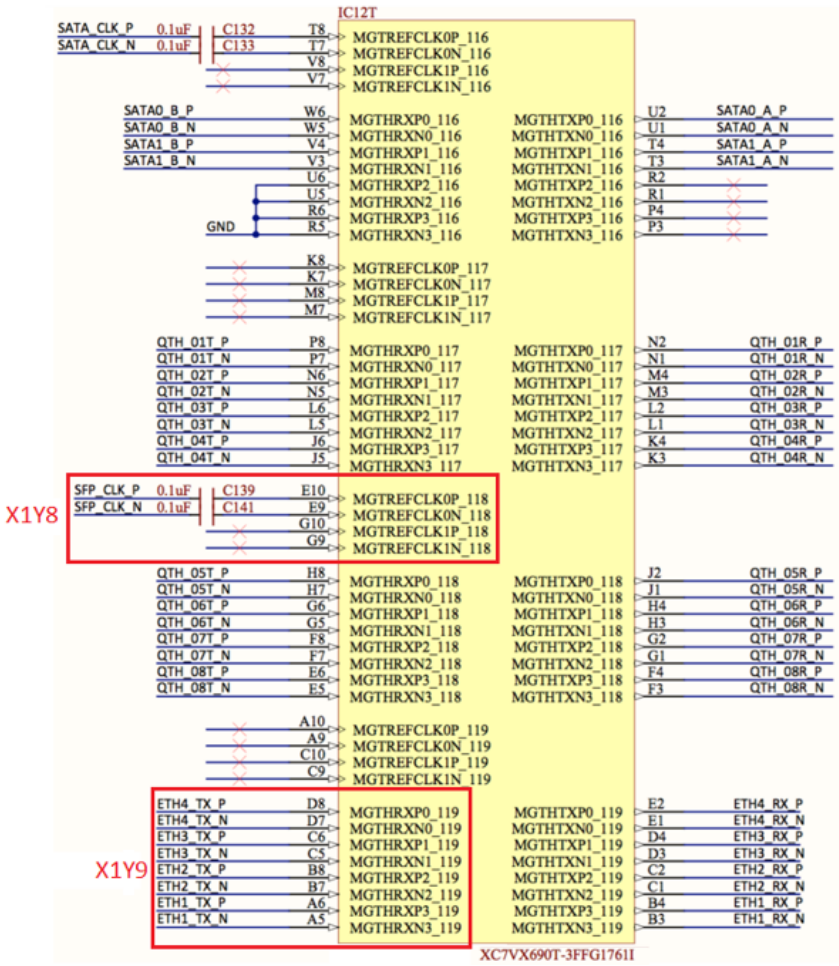


Figura 30 - Localização dos Pinos dos SFP+ no esquemático na NetFPGA SUME.

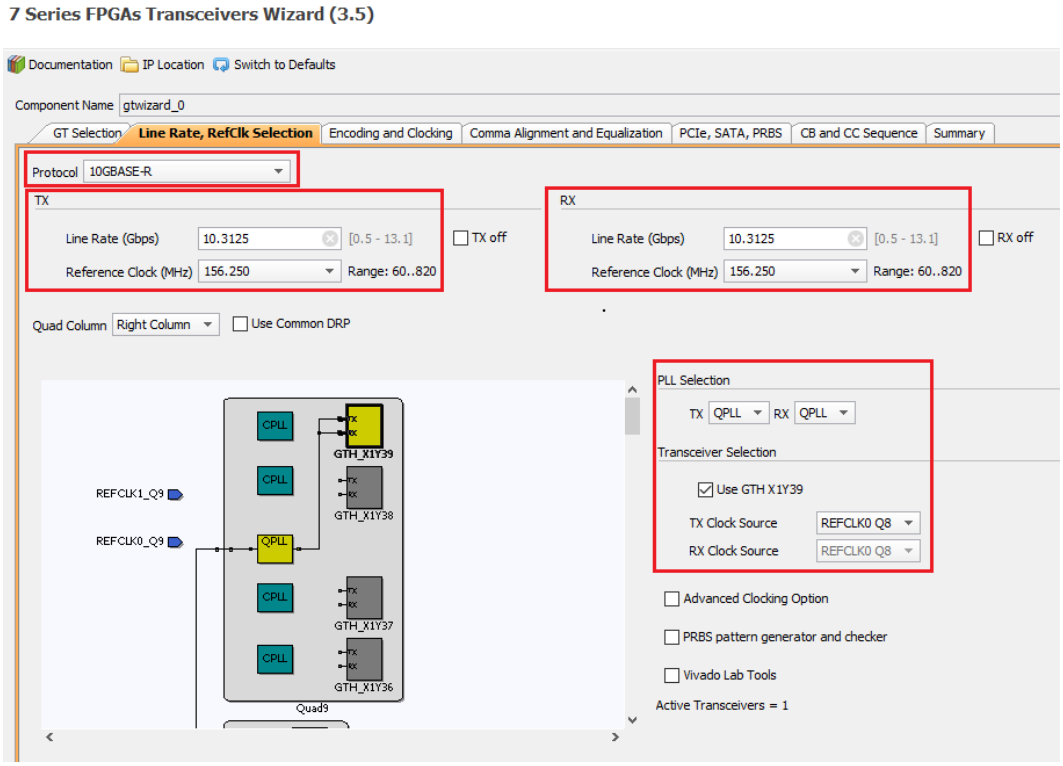


Figura 31 – Wizard de configuração – Protocolo, referência de clock e posicionamento.

Na segunda etapa de configuração (Figura 32) são determinados o padrão de codificação utilizado e o tamanho do barramento para interface com a lógica do FPGA. Como o módulo XGETH_TESTER utiliza uma interface de 64 bits, então esse tamanho de barramento deve ser selecionado. Observar que o *internal data width* é de 32 bits pois o *transceiver* não possui recursos para trabalhar com um barramento de 64 bits. Devido a seleção do protocolo 10GBASE-R, a codificação já é pré-configurada para o formato 64b/66b com controle de envio externo, ou seja, com a utilização de um contador externo para determinação dos períodos de pausa de transmissão, já que o *transceiver* não possui essa lógica implementada internamente como apresentado na Seção 3.6.

Nessa etapa foram desabilitados alguns recursos opcionais para reduzir a quantidade de pinos gerados no módulo GTH_gtwizard, já que esses não são utilizados no projeto. Essas configurações incluem o DRP (*Dynamic Reconfiguration Port*), que habilita os recursos para modificação dinâmica das configurações do *transceiver* e todas as demais portas opcionais disponíveis.

Por último, o *clock* global do sistema foi mantido em 100MHz, já que o máximo suportado é 175,01MHz e o *clock* de referência da FPGA opera a 200MHz, facilitando a sua divisão para uso na interface.

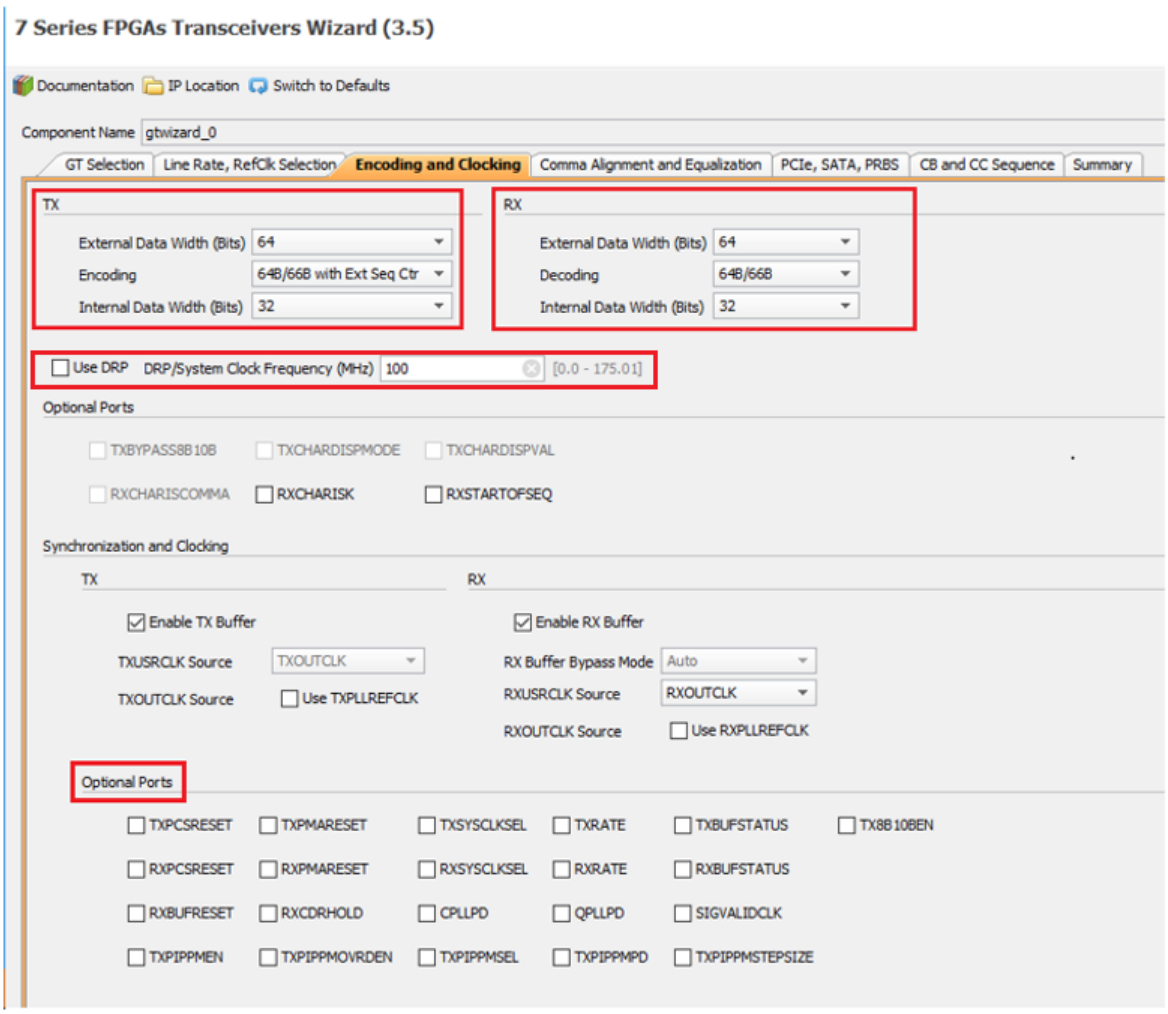


Figura 32 – Wizard de configuração: Codificação e tamanho das interfaces.

Ao final da configuração um sumário é apresentado com as principais informações relativas ao módulo que será gerado, conforme apresentado na Figura 33. Podemos observar que as referências de *clock* deduzidas no capítulo 3 estão de acordo com as referências apresentadas pelo sumário.

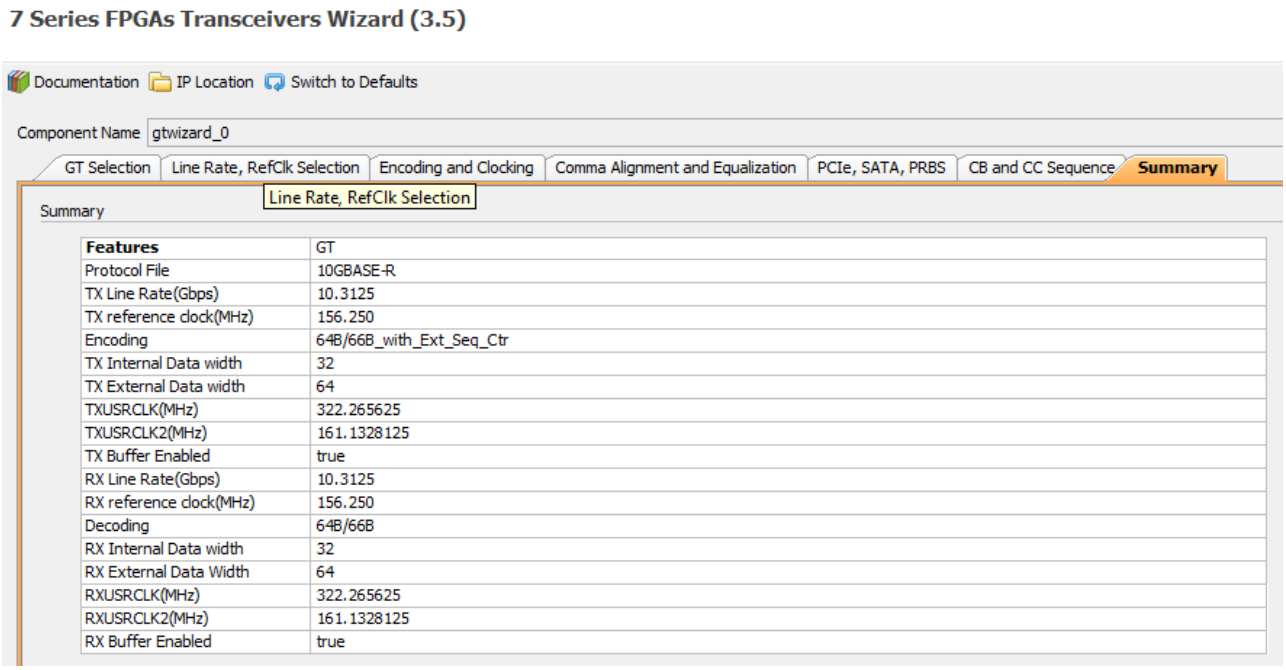


Figura 33 – Wizard de configuração: Sumário.

5.1.1.3 Instanciação do módulo GTH_gtwizard

Após o processo de geração do módulo GTH_gtwizard ser concluído, a instanciação apresentada na Figura 34 é gerada e pode ser adicionada ao projeto na ferramenta Vivado. Apesar do módulo ter sido configurado da forma mais simples possível, a instanciação necessita a conexão de diversos pinos. Dentro desses pinos são destacados em vermelho os que são necessários ao projeto e esses se encontram numerados. As funções desses pinos estão descritas abaixo:

Pinos de entrada:

1. **gt0_data_valid_in:** sinal proveniente do cliente para informação de que os pacotes recebidos são válidos.
2. **gt0_gthrxn_in/gt0_gthrxp_in:** par diferencial para recepção dos dados seriais do módulo SFP+.
3. **gt0_rxgearboxslip_in:** sinal utilizado no envio dos pulsos para o deslocamento dos pacotes das tentativas de alinhamento.
4. **gt0_txdata_in[63:0]:** barramento para transferência dos pacotes de 64 bits a transmitir
5. **gt0_txheader_in[1:0]:** barramento para transferência dos cabeçalhos dos pacotes a transmitir.
6. **gt0_txsequence[6:0]:** barramento para envio do contador para o TX Gearbox. Contador de 0 a 32.
7. **q8_clk0_gtrefclk_pad_n_in/ q8_clk0_gtrefclk_pad_p_in:** par diferencial de entrada da referência de *clock* externa.
8. **soft_reset_rx_in/soft_reset_tx_in:** reset global dos módulos de recepção e transmissão respectivamente.
9. **sysclk_in:** *clock* de referência para a lógica do módulo GTH_gtwizard (100MHz)**.**

Pinos de saída:

10. **gt0_gthrxn_in/gt0_gthrxp_in:** par diferencial para envio dos dados seriais.

11. **gt0_rx_fsm_done_out:** sinal de conclusão da máquina de estados do processo de reset dos módulos de recepção
12. **gt0_rx_data_out[63:0]:** barramento para transferência dos pacotes de 64 bits recebidos.
13. **gt0_rxdatavalid_out:** sinal que indica que o pacote de dados transferido pelo receptor é válido.
14. **gt0_rxheader_out[1:0]:** barramento para transferência dos cabeçalhos dos pacotes recebidos.
15. **gt0_rxheader_valid_out:** sinal que indica que o cabeçalho do pacote recebido é válido.
16. **gt0_rxresetdone_out:** sinal de conclusão do reset do módulo de recepção do *transceiver*.
17. **gt0_rxusrclk2_out:** *clock* de referência para transferência dos pacotes de dados recebidos (161.13MHz).
18. **gt0_tx_fsm_reset_done_out:** sinal de conclusão da máquina de estados do processo de reset dos módulos de transmissão.
19. **gt0_txresetdone_out:** sinal de conclusão do reset do módulo de transmissão do *transceiver*.
20. **gt0_txusrclk2_out:** *clock* de referência para transferência dos pacotes de dados a serem transmitidos (161.13MHz).
21. **q8_clk0_refclk_out:** *clock* de referência externo (156.25MHz).

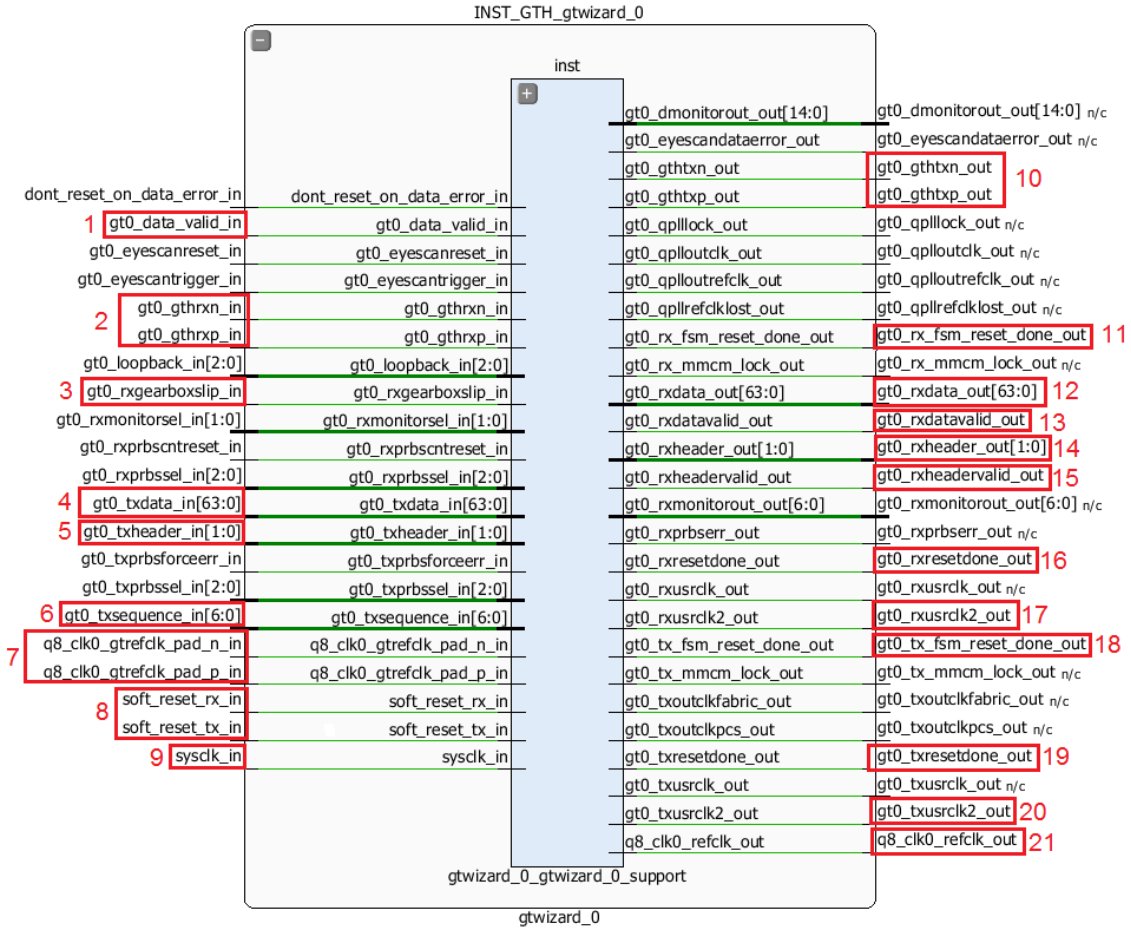


Figura 34 - Instanciação GTH_gtwizard.

5.2 Integração do GTH_gtwizard ao PCS do XGETH_TESTER

Algumas das funções exercidas na camada PCS do testador XGETH_TESTER já são realizadas internamente no canal do *transceiver,* sendo então necessária a reestruturação dos módulos que exercem essa função no projeto. Os ajustes necessários para a integração dos dispositivos são apresentados nas próximas Seções.

5.2.1 XGETH TESTER – Interfaces PCS e MAC Originais

No projeto em que o módulo XGETH_TESTER foi desenvolvido, as interfaces da camada física foram implementadas através de recursos diferentes do projeto atual. Anteriormente, não foram utilizados *transceivers* GTH e sim uma interface PMD proprietária. Por isso alterações nas interfaces internas da implementação do módulo XGETH_TESTER foram necessárias, para tornar o testador compatível com a interface dos *transceivers*. As alterações realizadas estão relacionadas com a camada PCS. A camada MAC permaneceu inalterada.

A Figura 35 apresenta a integração entre os módulos MAC e PCS do testador original.

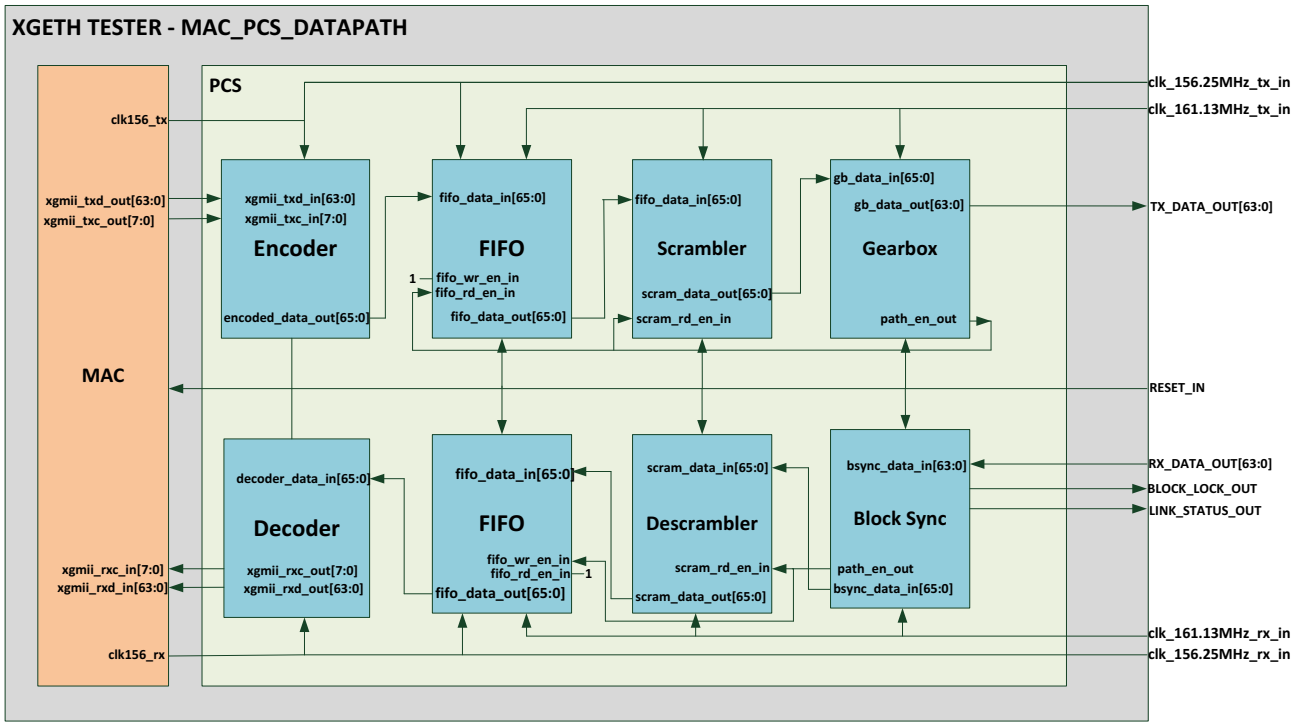


Figura 35 - MAC e PCS originais do módulo xgeth_tester.

O módulo PCS possui todas as funções exigidas pelo padrão 10GBASE-R para essa camada, sendo eles: codificador, decodificação (*decoder*), embaralhador (*scrambler*), desembaralhador (*descrambler*), *gearbox* e bloco de sincronismo (*block synchronizer*). Além desses módulos, são implementadas memórias do tipo FIFO entre o codificador e embaralhador, no sentido de transmissão, e decodificador e desembaralhador, no sentido de recepção. O uso dessas memórias é necessário devido à mudança na frequência de operação entre esses módulos, que variam entre 156,25MHz e 161,13MHz. No entanto apenas a utilização da FIFO causaria problemas de *starvation* e *overflow* de dados entre esses módulos nos sentidos de transmissão e recepção respectivamente, pois a quantidade de dados entrando e saindo da FIFO a cada ciclo de *clock* é a mesma em ambos os

domínios de *clock*. Como solução, além do uso das FIFOs, é necessária a realização de uma pausa na transferência de dados entre esses módulos a cada 32 ciclos de *clock*. O controle dessa pausa é feito nos módulos *gearbox* e *block sync*, pois esses reduzem e aumentam, respectivamente, o tamanho dos pacotes transmitidos, e por isso devem inferir uma pausa na transferência de dados a cada 32 ciclos de *clock* para compensar esta diferença. A utilização de 32 dois ciclos é deduzida através das seguintes equações:

A partir das frequências de ambos os domínios de frequências 156,25MHz e 161,13MHz são calculados os períodos dos mesmos nas equações (4) e (5) respectivamente.

$$T_{156,25MHz} = \frac{1}{156,25MHz} = 6.4ns \quad (4)$$

$$T_{161,13MHz} = \frac{1}{161,13MHz} = 6.2ns \quad (5)$$

O tempo necessário para que sejam completados 32 ciclos de *clock* em ambos os domínios de *clock* é dado por (6) e (7) respectivamente.

$$T_1 = 32\ Ciclos \times T_{156,25MHz} = 204.8ns \quad (6)$$

$$T_2 = 32\ Ciclos \times T_{161,13MHz} = 198,4ns \quad (7)$$

A diferença entre (6) e (7) é dada pela equação (8).

$$T_{dif} = T_1 - T_2 = 6.4ns = T_{156,25MHz} \quad (8)$$

Como podemos observar, a diferença de tempo para que no domínio de maior frequência sejam transferidos 32 pacotes é igual a um período de *clock* de menor frequência, logo ao final de 32 ciclos existirá um pacote a mais de dados acumulado na FIFO. Esse pacote será transferido para o domínio de menor frequência durante o período de pausa na transferência no domínio de maior frequência. Esse processo permite que a mudança de domínio de *clock* se torne transparente para os módulos codificador e decodificador.

O módulo MAC utilizado no testador XGETH_TESTER é uma interface de distribuição gratuita disponibilizada pelo grupo OpenCores [OPE08]. Esse módulo possui a interface XGMII necessária para a comunicação com os módulos de codificação e decodificação da camada PCS. O módulo MAC realiza o controle de fluxo de dados, inserindo o espaçamento necessário entre os dados com inclusões de pacotes *Idle*. Também é realizado o controle de erro através do preenchimento e validação do campo CRC de cada pacote. Os pacotes válidos são transferidos para o testador, sendo então descartados os pacotes com erro. Esse recurso é de grande importância para os métodos que são utilizados para os testes da interface.

5.2.2 Modificações no PCS do XGETH TESTER

Dois módulos exigem adaptação à nova interface do GTH_gtwizard: gearbox e bloco de sincronismo.

No sentido de transmissão, a única função utilizada do módulo *gearbox* da interface atual é a de um contador de 32 ciclos para controle da transferência do módulo *TX Gearbox* do módulo do *transceiver* para o deslocamento de dados, como apresentado na Seção 3.6. No período 32 do contador deve-se gerar um sinal de pausa da transferência (*data_pause_out*) para referência dos módulos Scrambler e FIFO, impedindo que os mesmos realizem transferência de dados nesse

período. No projeto, o módulo *gearbox* foi inteiramente substituído pelo contador, não sendo necessária nenhuma das funções exercidas pelo mesmo.

O bloco de sincronismo implementado no testador possui dois submódulos. O primeiro, denominado bloco de controle, exerce o controle de alinhamento dos dados, através da verificação dos cabeçalhos dos pacotes recebidos. Também é realizada a geração do pulso (*slip*), para alteração do alinhamento dos dados e detecção de erros de transmissão (BER). O segundo módulo (*FrameSync FIFO*) realiza o agrupamento dos pacotes de 64 bits recebidos, provenientes da camada PMD em pacotes de 66 bits. Quanto necessária a mudança de alinhamento dos dados, o deslocamento dos mesmos é realizado, através do sinal *slip, o qual é gerado* pelo bloco de controle. Esse módulo também gera o sinal de pausa do ciclo de transmissão (*wen_r*). Esse bloco realiza as mesmas funções do *RX gearbox* do *transceiver GTH* apresentado na Seção 3.7, sendo então necessária sua remoção do projeto e os sinais de controle utilizados e enviados por ele passaram a ser integrados com o módulo GTH_gtwizard.

As alterações necessárias no bloco de sincronismo foram:

- Remoção do bloco FrameSync FIFO.
- O sinal de pausa gerado através do bloco FrameSync (*wen_r*) foi removido e substituído pelo sinal *rx_header_valid_in*, gerado no módulo GTH_gtwizard.
- O pulso para deslocamento do alinhamento *(slip*) agora foi integrado pelo bloco de controle diretamente no módulo GTH_gtwizard através do sinal *rxgearboxslip_out*.
- A recepção dos pacotes de dados através do barramento (*data_in*) não é mais necessária, pois o bloco FrameSync foi removido, e sua função passou a ser exercida pelo *RX Gearbox do módulo GTH_gtwizard,* não sendo mais necessário internamente. Na nova estrutura, apenas o cabeçalho dos dados é recebido para detecção do alinhamento, através do barramento *rx_header_in*, proveniente do módulo GTH_gtwizard,

A Figura 36 apresenta o diagrama de blocos do bloco de sincronismo com as adaptações necessárias. Os fios e o bloco FrameSync, que estão pontilhados, representam as interfaces removidas do projeto.

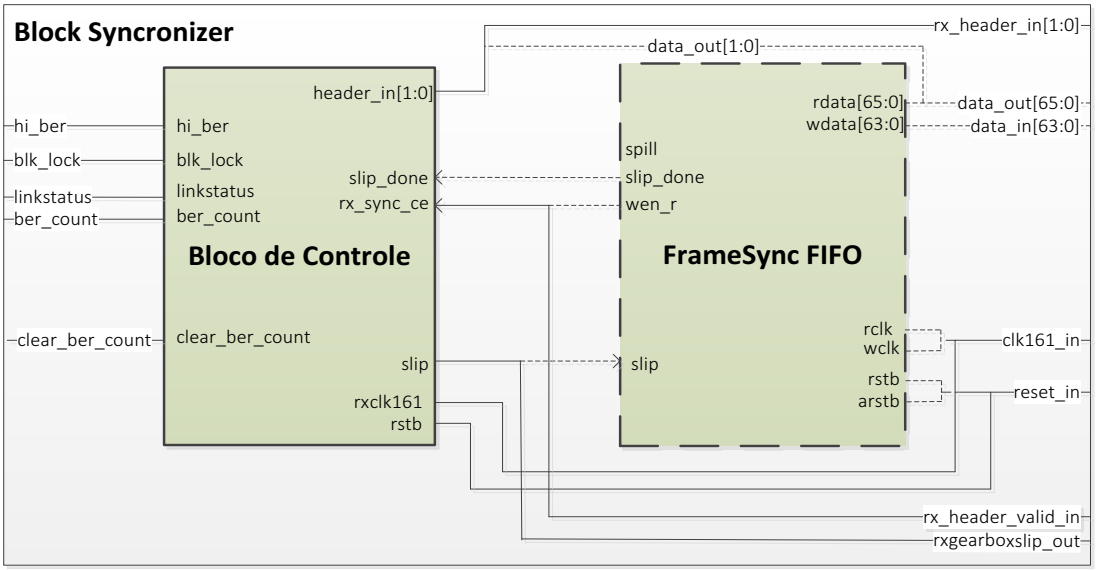


Figura 36 - Modificações no bloco de sincronismo da camada PCS.

5.2.3 Integração dos módulos GTH_gtwizard e XGETH TESTER

A partir da modificação dos módulos *gearbox* e *block synchronizer* foi possível realizar a integração do módulo XGETH TESTER com o GTH_gtwizard. A Figura 37 apresenta o diagrama de blocos da nova estrutura. Além das modificações apresentadas na Seção 5.2.2, foram necessárias as alterações descritas abaixo.

Com relação a interface de transferências de dados, a integração ocorreu da seguinte forma:

- No sentido de transmissão, o barramento scram_data_out do módulo *scrambler* foi separado nos barramentos txheader_in e txdata_in e conectados à interface GTH_gtwizard;
- No sentido de recepção, os barramentos rxheader_in e rxdata_in foram unidos e conectados ao modulo *descrambler* através do barramento descram_data_in;
- Em ambos os sentidos de transferência, os cabeçalhos estão localizados nos bits menos significativos dos barramentos scram_data_out e scram_data_in para compatibilidade com os módulos *Scrambler* e *Descrambler*, respectivamente;
- É necessária a inversão da ordem dos dados entre módulos *scrambler* e *descrambler*, e o módulo GTH_gtwizard, pois o primeiro utiliza a arquitetura *big endian* e o segundo *little endian*.

A sequência de resets passou a ser integrada entre os módulos XGETH TESTER e GTH_gtwizard e é realizada através de duas etapas:

- Primeiramente os sinais *txresetdone_out* e *rxresetdone_out* do módulo GTH_gtwizard liberam o reset do módulo PCS nos sentidos de transmissão e recepção respectivamente. Assim, inicia-se a transmissão e recepção de dados para início do processo de sincronismo entre o canal do *transceiver* e o dispositivo externo ao qual ele está interconectado;
- Após a primeira etapa, os sinais *txresetfsmdone* e *rxresetfsmdone* do módulo GTH_gtwizard, são utilizados para liberar o módulo MAC, permitindo o início da transmissão e recepção de pacotes no canal, respectivamente. Esses sinais indicam que o *transceiver* completou sua sequência de resets e está pronto para operar nos seus respectivos sentidos de transmissão.

Os *clocks* de referência fornecidos pelo canal GTH_gtwizard são conectados no módulo XGETH TESTER da seguinte forma:

- Os módulos *sequence counter*, *scrambler* do sentido de transmissão, e *block sync*, *descrambler* do sentido de recepção, utilizam como referência os *clocks* txusrclk2_out e rxusrclk2_out respectivamente, ambos de frequência 161.13MHz;
- Os módulos FIFO dos sentidos de transmissão e recepção utilizam dois *clocks* de referência: txusrclk2_out e q8_clk0_gtrefclk_pad_out, e rxusrclk2_out e q8_clk0_gtrefclk_pad_out, respectivamente;
- Os módulos MAC, *encoder* e *decoder* utilizam o *clock* q8_clk0_gtrefclk_pad_out.

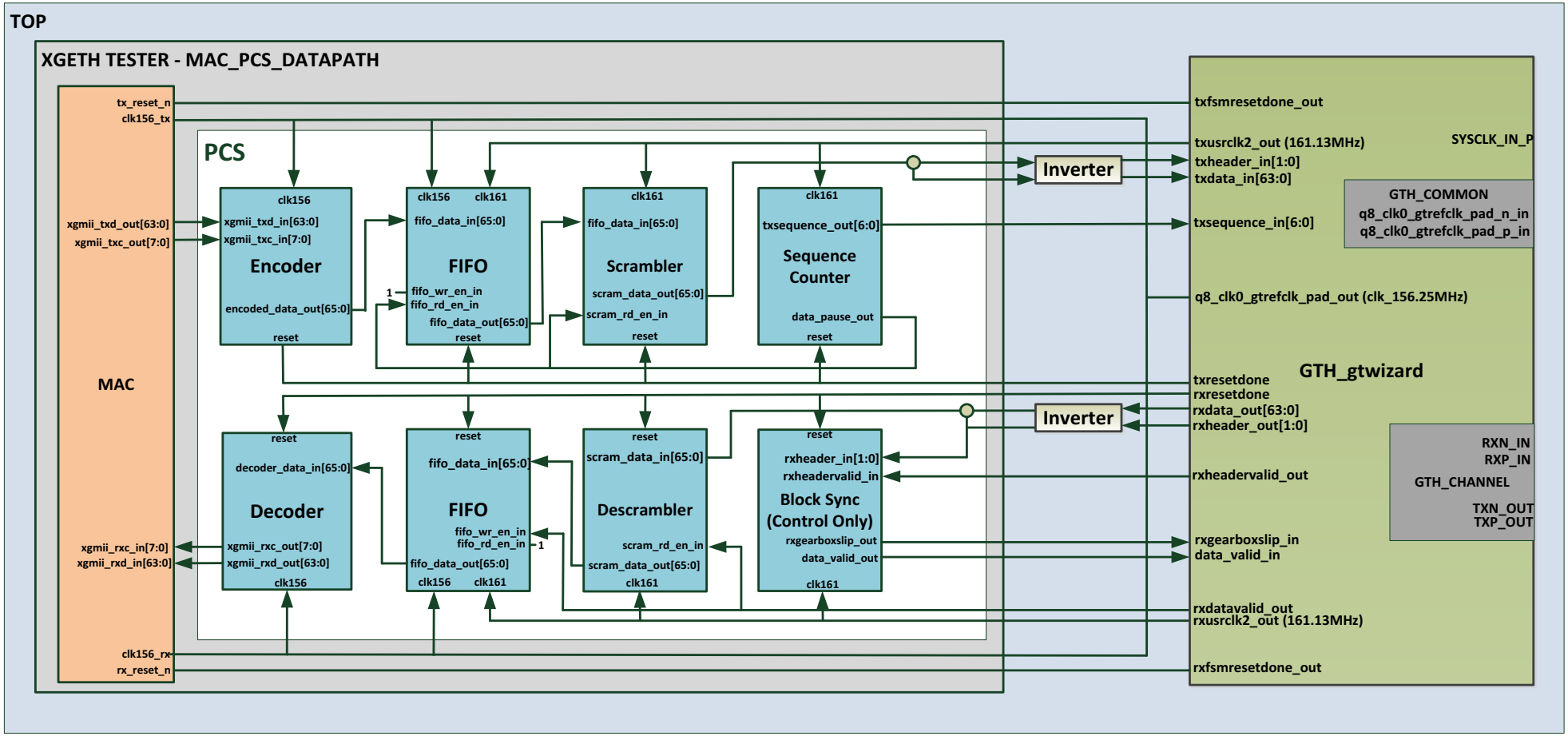


Figura 37 - Integração GTH_gtwizard e XGETH_TESTER

Quanto aos sinais de controle são utilizados conectados os seguintes sinais:

- O contador de transmissão é transferido a partir do barramento txsequencecounter_out, e conectado diretamente ao módulo GTH_gtwizard através do barramento txsequencecounter_in.
- O sinal de detecção de alinhamento *data_valid_in* é enviado pelo bloco de sincronismo ao módulo GTH_gtwizard através do sinal *data_valid_out*.
- Sinal para deslocamento do alinhamento *rx_gearboxslip_in* enviado pelo bloco de sincronismo para o módulo GTH_grwizard através do sinal *rxgearboxslip_out*.
- Sinal de indicação de cabeçalho válido *rxheadervalid_out* proveniente do módulo GTH_gtwizard é conectado ao bloco de sincronismo através do sinal *rxheadervalid_in*.

5.3 XGETH TESTER – Interface de Teste

Para geração e validação de pacotes de dados para fins de teste, foi utilizado o módulo testador da interface XGETH TESTER. Esse módulo realiza os testes de *throughput*, *latency*, *system recovery, frame loss rate,* e *back-to-back* conforme especificado no documento RFC2544, da comunidade *Internet Engineering Task Force* (IETF), e que têm por objetivo, verificar o desempenho de conexão entre interfaces de rede [IET99]. Os módulos testadores são interconectados aos dispositivos SFP+ através de módulos MAC e PCS, esses implementados internamente no testador, e o módulo GTH_gtwizard apresentado na Seção 5.2.3.

O testador XGETH TESTER permite a execução de rajadas de pacotes do tipo ECHO, pertencentes ao protocolo ICMP [IET81]. Esses pacotes são enviados e recebidos em diferentes taxas de transferência, tamanho e quantidade por rajada. Porém, os tamanhos de pacotes são limitados aos valores: 64, 128, 256, 512, 768, 1024 e 1518 bytes, como definido na RFC2544 para os testes implementados [IET99]. Outro recurso do testador é o envio de pacotes com *timestamps* que podem ser identificados quando recebidos pelo receptor. Os testes são realizados com a fibra óptica em *loopback* (transmissor ligado diretamente no receptor), ou através de um dispositivo capaz de retornar os pacotes recebidos. Os diferentes modelos de teste definidos pela RFC2544 e disponíveis na interface XGETH TESTER serão apresentados nas próximas Seções.

O testador é composto de quatro submódulos, os quais são apresentados no diagrama de blocos da Figura 38. As funções exercidas pelo mesmo estão descritas abaixo:

- **uPC Interface:** Interface de comunicação do testador. Através desse módulo são realizadas a leitura e escrita nos registradores do testador. Os registradores contêm os parâmetros de teste, e controle sobre a inicialização e término dos testes e os dados relativos ao teste realizado. A Tabela 3 apresenta os valores respectivos e funções dos registradores pertencentes ao módulo *uPC Interface.*
- **Traffic Injector** (na Figura identificado como *RFC2544*)**:** Interface de controle dos testes de acordo com a RFC2544. O módulo executa os testes de acordo com os parâmetros configurados no banco de registradores do módulo uPC interface. Esse módulo controla a geração e recepção de pacotes do tipo ECHO, e realiza a coleta dos dados necessários para diagnóstico dos testes, como número de pacotes recebidos e o *timestamp* de envio e retorno de um pacote.

- **Echo Generator:** Gerador de pacotes do tipo ECHO com recursos de envio com dados aleatórios ou com um identificador *timestamp* que pode ser identificado no receptor. Os pacotes são gerados de acordo com o tamanho especificado através do banco de registradores, e dentro das limitações de tamanho aceitas. Esse módulo é conectado diretamente ao módulo MAC.
- **Echo Receiver:** Receptor de pacotes do tipo ECHO. Esse módulo gera um indicador para o Traffic Generator quando um pacote válido é recebido.

Tabela 3 - Registradores do módulo uPC Interface.

Address	Registrador	Descrição
0x1000	STATUS	Inicializa a operação do XGETH_TESTER. Quando escrito o valor '1' no registrador a operação do mesmo é inicializada. No fim da operação no testador o valor desse registrador retorna ao valor '0'.
0x1001	TEST_CODE	Seleção do teste a ser executado: (0x01) Throughput; (0x02) Loss Rate; (0x03) Latency; (0x04) Back-to-Back; (0x05) System Recovery.
0x1002	PKT_length	Indica o tamanho do pacote a ser utilizado: (0x00) 64 Bytes; (0x01) 128 Bytes; (0x02) 256 Bytes; (0x03) 512 Bytes; (0x04) 768 Bytes; (0x05) 1024 Bytes; (0x06) 1280 Bytes; (0x07) 1518 Bytes.
0x1003	IDLE_number	Define o tamanho do inter-frame gap. O testador insere 8 bytes vezes <IDLE_number>. Para <IDLE_NUMBER> igual a 1, o hardware insere intercaladamente 8 e 16 bytes de intergap entre os pacotes, formando uma média de 12 bytes de intergap para atingir a vazão máxima.
0x1004	PKT_number[15:0]	Número de pacotes a serem transmitidos em uma rajada de dados.
0x1005	PKT_number[31:16]	
0x1006	TIMESTAMP_pos[15:0]	Indica a posição do pacote dentro da rajada de dados onde um identificador (timestamp) será inserido. Se o valor desse registrador for zero, não é gerado nenhum timestamp.
0x1007	TIMESTAMP_pos[31:16]	
0x1008	MAC_source[15:0]	MAC Origem
0x1009	MAC_source[31:16]	
0x100a	MAC_source[47:32]	
0x100b	MAC_destination[15:0]	MAC Destino
0x100c	MAC_destination[31:16]	
0x100d	MAC_destination[47:32]	
0x100e	IP_source[15:0]	IP Origem
0x100f	IP_source[31:16]	
0x1010	IP_destination[15:0]	IP Destino
0x1011	IP_destination[31:16]	
0x1012	TIMEOUT	Após todos os pacotes serem enviados, o testador aguarda <TIMEOUT> ciclos de *clock* no receptor para que o receptor receba os pacotes restantes antes de encerrar o teste.
0x1013	PKT_sequence	Parâmetros utilizados no teste de system *recovery* Indica o número mínimo de pacotes recebidos em sequência para estabelecer que a conexão é estável.
0x1014	PKT_rx[15:0]	Número de pacotes recebidos durante o teste. Este parâmetro é atualizado ao final de cada rajada.
0x1015	PKT_rx[31:16]	
0x1016	latency[15:0]	Parâmetro usado nos testes de latência e system *recovery*. Nesse registrador são armazenados os tempos de latência da conexão e recuperação do sistema, respectivamente.
0x1017	latency[31:16]	
0x1018	latency[47:32]	

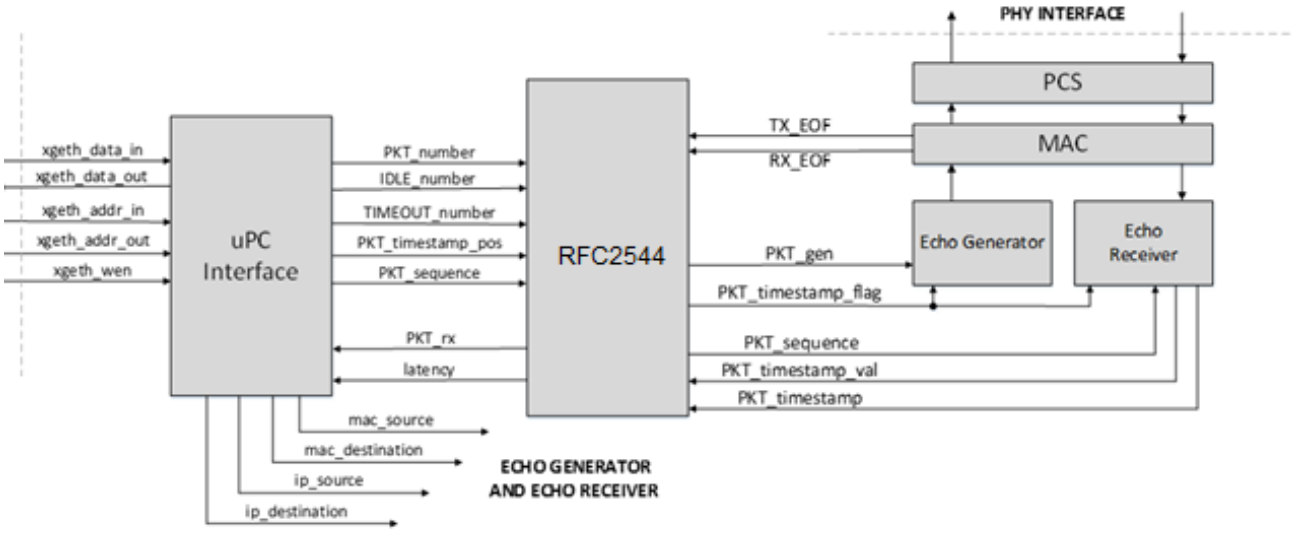


Figura 38 – Arquitetura do testador.

5.3.1 Throughput

O teste de vazão (*throughput*) verifica qual a taxa máxima de transferência suportada pelo canal. O teste é feito através da transmissão de rajadas de pacotes do tipo ECHO e a contabilização dos pacotes retornados. O teste é realizado a partir de uma taxa inicial determinada. A cada rajada executada com sucesso, ou seja, onde nenhum pacote é perdido, inicia-se um novo teste com uma taxa de transferência maior que a anterior. A taxa máxima do canal é determinada como a taxa de maior vazão onde não houveram perdas de pacotes [IET99].

5.3.2 Latency

O teste de latência (*latency*) tem por objetivo verificar o tempo que um pacote leva para ser transmitido e retornar à interface. O teste é feito através do envio de pacote do tipo ECHO. Uma rajada de dados de dados é então enviada à taxa máxima do sistema a qual foi determinada previamente através de um teste de *throughput*. Na metade da rajada de dados é enviado um pacote identificado por um *time-stamp*. O tempo que o pacote com *time-stamp* leva para retornar à interface é tido como o tempo de latência do sistema [IET99].

5.3.3 System Recovery

O teste de *system recovery* tem por objetivo verificar o tempo que o sistema demora para se recuperar de uma situação de sobrecarga. O teste inicia com uma rajada de dados a uma taxa 110% maior do que a maior taxa detectada através de um teste de *throughput*, forçando então, a perda de pacotes. Após um determinado período de tempo a taxa é reduzida à metade. O tempo de *system recovery* e determinado através da diferença entre o momento em que a taxa foi reduzida e o momento em que o último pacote da rajada foi perdido [IET99].

5.3.4 Frame Loss Rate

O teste *frame loss rate* tem por objetivo identificar a vazão máxima em que a interface não perde pacotes. O teste inicia-se com uma rajada de dados na taxa máxima suportada. A taxa é reduzida em 10% a cada teste. No momento em que é identificada uma taxa onde não há perdas, o teste é concluído e a taxa encontrada é dada como taxa de *frame-loss* [IET99].

5.3.5 Back-to-Back

O teste de *Back-to-Back* tem por objetivo encontrar a maior rajada de pacotes possível sem que haja perda de pacotes utilizando a maior taxa de transferência da interface. O teste inicia com a rajada de um a quantidade de pacotes. Caso a rajada tenha pacotes perdidos, a quantidade de pacotes é reduzida, caso contrário é aumentada.

5.4 Interface de Programação

Uma interface de comunicação serial realiza o envio e recebimento de dados através do protocolo RS-232, como apresentado na Figura 29 (módulo *UART interface*). Esse módulo permite a comunicação com o usuário através de software, via porta USB (Universal Serial Bus).

O módulo *UART interface* transmite os parâmetros de teste para o módulo *control interface*. Esse tem por objetivo identificar os parâmetros recebidos e realizar a escrita ou leitura do banco de registradores do módulo *uPC Interface*. No caso de leitura, os dados solicitados são lidos do banco de registradores e retornados para o usuário através do módulo *UART Interface*.

5.5 Interface I2C

A interface de I2C realiza a configuração de chip de modelo SI5324 externo ao FPGA que gera a referência de *clock* externa de 156.25MHz conectada ao banco de *transceivers GTH*. A configuração do chip é feita na inicialização no dispositivo FPGA através do protocolo I2C (*Inter-Integrated Circuit*). O processo de inicialização do *transceiver* ocorre a partir da identificação da existência desse *clock*.

6 IMPLEMENTAÇÃO E TESTES

No Capítulo anterior foram apresentadas as estruturas necessárias para o desenvolvimento da interface Ethernet 10GBASE-R utilizando *transceivers* GTH. Os aspectos apresentados até então estão relacionados apenas com o desenvolvimento do projeto do sistema. Nesse Capítulo são apresentadas as demais etapas de desenvolvimento, conforme o fluxo de projeto para dispositivos FPGAs apresentado no Capítulo 4. As etapas correspondem aos processos de síntese lógica, síntese física e prototipação da interface desenvolvida. A validação da interface é feita por meio da análise do comportamento lógico através de simulação, através da ferramenta *Chipscope* e a partir da execução de diferentes testes através do testador XGETH TESTER.

6.1 Desenvolvimento do Testbench para Simulação

No projeto em que o módulo XGETH_TESTER foi desenvolvido, o *testbench* implementado permitia a simulação apenas do testador (*UUT*), pois naquele projeto, o objetivo era a validação do módulo de teste de acordo com a RFC2544, e não a interface de comunicação 10GBASE-R. A necessidade de verificação da interface atual vai além do testador, e por isso surgiu a necessidade da readaptação da estrutura de simulação (*testbench*).

A estrutura do *testbench* original consiste na instanciação do módulo XGETH TESTER (*UUT*), um cliente (*Client*), uma interface MAC/PCS (*Mirror MAC_PCS*) e o controlador do *testbench* (*Testbench Control)*. O objetivo do cliente é receber os pacotes transmitidos pelo módulo XGETH TESTER e retransmiti-los com uma latência pré-determinada, permitindo então, diferentes valores de latência de transmissão. Além disso, o cliente permite a inserção forçada de perda de pacotes, através da não retransmissão um pacote recebido. Para integração entre o cliente e o módulo XGETH_TESTER é utilizado o módulo *Mirror MAC_PCS*, o qual possui a mesma interface MAC_PCS utilizada no módulo XGETH_TESTER. O diagrama de blocos da estrutura original do *testbench* é apresentado na Figura 39.

A módulo *Testbench Control* executa, a partir arquivo de configuração (RFC.conf), os testes a configurados para execução. Os testes são controlados diretamente através da escrita e leitura dos registradores do módulo uPC Interface pelo módulo *Testbench Control*. Ao final do teste é gerado um arquivo com os resultados (RFC.log).

No novo *testbench* (Figura 40) foram mantidas as funções relativas a execução dos testes da RFC2544, porém foi necessária a expansão do *testbench* para que ele abrangesse toda a interface de comunicação 10GBASE-R. Para isso, o *testbench* passou a instanciar o topo do projeto, o qual contém as instanciações não só do testador, mas também do módulo GTH_gtwizard e os demais módulos de controle apresentados no Capítulo 5. Também foram necessárias a utilização de uma interface UART (*UART Interface Mirror)* instanciada no *testbench,* para comunicação com o módulo *uPC Interface* do testador.

Para a utilização do cliente na nova interface seria necessário a inclusão de um módulo GTH_gtwizard adicional e sua integração com o módulo *MAC_PCS Mirror*. Essa inclusão implicaria no aumento da complexidade do testador. Por isso optou-se pela remoção do módulo *Client*, já que a

sua utilização não implicaria em um aumento significativo no tempo necessário para simulação. Assim, a saída serial do módulo GTH_gtwizard foi colocada em *loopback*.

Apesar da remoção do módulo *client* ser uma solução para o desempenho da simulação, sua remoção acarreta perda da capacidade de inserção de erros durante a execução dos testes. Esse recurso apesar de não obrigatório, elimina a capacidade de verificação do comportamento do testador quando detectada a perda de um pacote durante a simulação. Dado que o escopo do TCC é a interface de alta velocidade e não o testador em si, esta perda não implica em prejuízos ao trabalho.

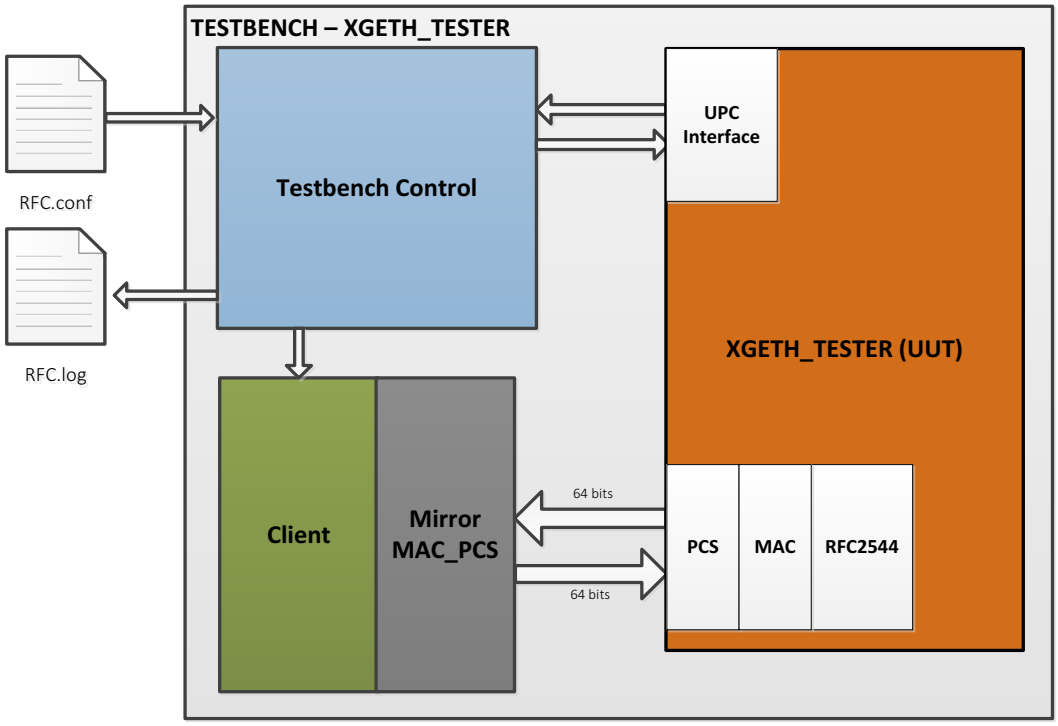


Figura 39 - Testbench Original.

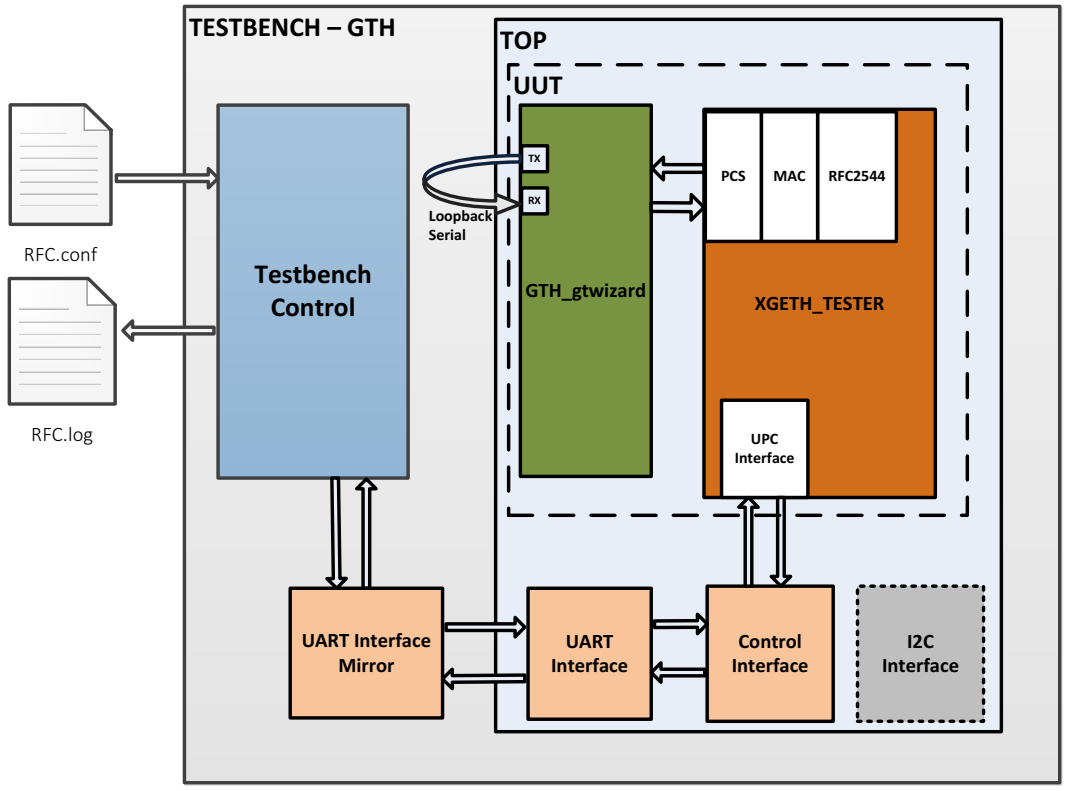


Figura 40 - Testbench para Interface 10GBASE-R.

A inclusão do módulo GTH_gtwizard na simulação acarreta na drástica diminuição da velocidade de processamento devido ao aumento da máxima frequência utilizada. No projeto anterior, a maior frequência de *clock* em simulação era de 161.13MHz. A partir da instanciação do módulo GTH_gtwizard, a frequência máxima passou a ser 10.315GHz, pois passou a ser simulado a transmissão serial de alta velocidade, enquanto anteriormente as transferências eram em pacotes de 64 bits.

Também é necessário observar que a velocidade de transferência dos comandos através da interface UART, é de somente 115200 bits por segundo. Nessa taxa de transmissão a simulação se torna inviável devido ao tempo necessário para transferência dos comandos. Para contornar esse problema, foi adicionado um parâmetro que habilita o hardware em modo de simulação, e assim acelera velocidade de transmissão do módulo UART. Além disso o parâmetro impede a geração do módulo I2C Interface, pois durante a simulação o *clock* de 156,25MHz tem origem através de estímulos do próprio *testbench* e não é necessária a configuração do chip SI5324 externo ao FPGA, como apresentado na Seção 5.5.

6.2 Simulação da Interface 10GBASE-R

A partir das modificações do *testbench* original apresento na Seção 6.1 foi possível realizar a verificação do comportamento lógico de toda a interface desenvolvida. Nessa Seção é apresentada os principais pontos da simulação de um teste de *thoughput*.

Para inicialização da simulação o parâmetro *enable_simulation* localizado no topo do projeto deve ser configurado para *true,* para que a simulação opere com o módulo *UART Interface* acelerado.

```
entity GTH_tester is

generic (

    enable_simulation : boolean := true -- Enable simulation mode when asserted true.
);
```

Figura 41 - Habilitação do modo de simulação.

Para a simulação de um teste de *throughput* é necessário a configuração do arquivo RFC.conf. A Figura 42 apresenta a configuração utilizada. No arquivo são especificados o tipo de pacote ECHO, o tipo de teste (*thoughput*), transmissão de 100 pacotes por rajada, envio de 64 bytes por pacote, taxa inicial de 7000Mb/s, incremento de 100Mb/s a cada rajada, e um *timeout* de 1500 ciclos de 156MHz (9.6us).

RFC.conf

```
<N#TESTS>  1

    <TEST_throughput>
        <PKT_TYPE>          ECHO
        <TYPE>        throughput
        <N#PACKETS>          100
        <S#PACKETS>         64
        <INIT_TX>           1000
        <STEP_TX>            100
        <TX_LOST>           6000
        <CLIENT_LOST>          0
        <CLIENT_LATENCY>       0
        <TIMEOUT>           1500
    <END>
```

Figura 42 - Configuração do RFC.conf.

6.2.1 Inicialização das Interface 10GBASE-R

O primeiro aspecto importante a ser analisado em simulação é inicialização do módulo *GTH_gtwrapper*, pois este realiza a geração dos *clocks* de referência da interface de comunicação e inicia a transmissão e recepção. Nessa etapa também são realizados os processos de inicialização da transmissão, recepção e alinhamento dos pacotes recebidos.

A Figura 43 apresenta a simulação do processo de inicialização do módulo *GTH_gtwrapper*. Os pontos principais estão enumerados em vermelho e descritos abaixo:

1. Início da transmissão do *clock* de recepção do usuário (RXUSRCLK e RXUSRCLK2) gerado através do módulo MMCM.
2. Início da transmissão do *clock* do usuário (TXUSRCLK e TXUSRCLK2) gerado através do módulo MMCM.
3. Fim do processo de reset do receptor (*gt0_rxfsmresetdone* e *gt0_rxresetdone*). Nesse período também ocorre um falso sinal de alinhamento (*gt0_linkstatus*), pois o transmissor ainda não iniciou seu processo de transmissão durante esse processo. Durante esse período o canal transmite sequencias de zero e uns, fazendo com que o bloco de sincronismo considere os pacotes recebidos como válidos e alinhados.
4. Fim do processo de reset do transmissor (*gt0_txfsmresetdone_i* e *gt0_txresetdone_i*). Nesse período ocorre o início da transmissão de dados válidos no barramento *gt0_txdata_i*.
5. No momento em que os dados transmitidos começarem a retornar através da interface do receptor (*gt0_rxdata_i*), é detectado que o alinhamento não está correto e iniciado o processo de realinhamento. O bloco de sincronismo inicia o envio de pulsos através do sinal gt0_rxgearboxslip_i na tentativa de detectar o alinhamento.
6. Quando detectado o alinhamento correto o sinal *gt0_linkstatus* volta a '1' e o processo de reset do receptor e novamente concluído (*gt0_rxfsmresetdone_i*). A partir desse momento a interface está pronta para utilização.

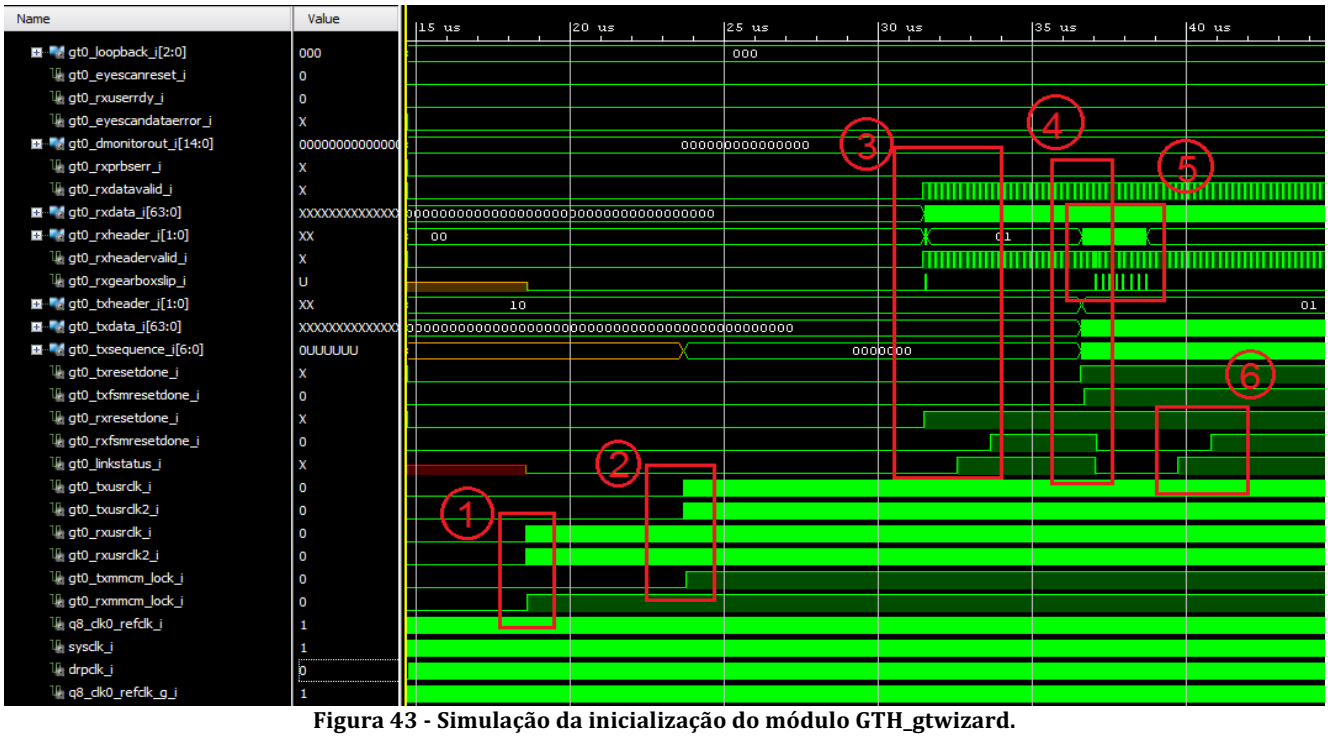


Figura 43 - Simulação da inicialização do módulo GTH_gtwizard.

6.2.2 Geração de um pacote

Na Figura 44 é apresentado a simulação da geração de um pacote no módulo *echo_generator*. Nesse trecho da simulação pode-se visualizar a transmissão de o pacote que contem oito blocos de 64 bits, totalizando os 64 bytes de dados por pacote configurado através do arquivo de configuração RFC.conf. Notar a geração de um pacote que constitui de um campo de cabeçalho, com informações sobre endereços de origem e destino e protocolo utilizado (S_HEADER), um campo com o *timestamp* para identificação do pacote no receptor (S_TIME_STAMP), e o restante do pacote é preenchido com dados gerados aleatoriamente (S_PAYLOAD).

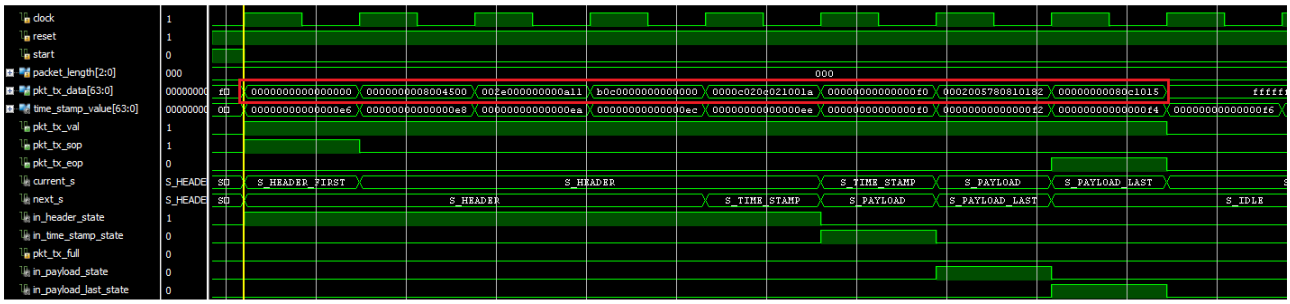


Figura 44 - Simulação da geração de um pacote.

6.2.3 Simulação do teste de Throughput

Na Figura 45 é apresentado o os sinais de controle de um teste de *throughput*. É possível observar que no teste são realizadas diversas rajadas de dados, as quais são iniciadas através do sinal *initialize* e seu termino indicado pelo sinal RFC_end. Por fim o sinal RFC_running indica o período em que o teste está em execução.

O tamanho do intergap entre pacotes (*IDLE_number*) é reduzido a cada teste. Como podemos observar na simulação, o teste atingiu a inserção de *inter-gap* mínimo entre pacotes (*IDLE_number* = 1). Esse resultado representa que a interface atingiu a maior taxa transferência possível, para o tamanho do pacote utilizado, sem perdas.

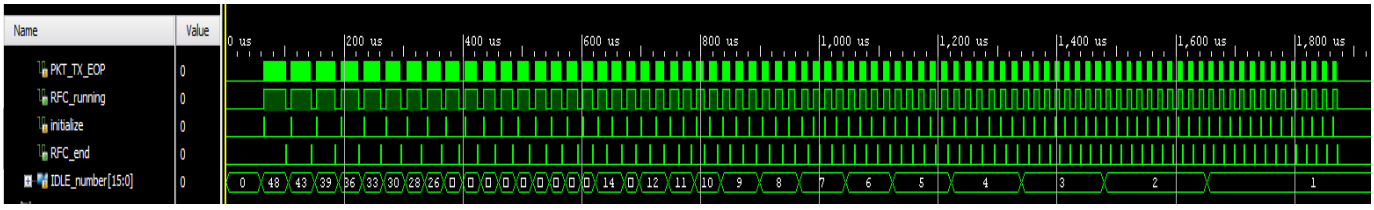


Figura 45 - Simulação de um teste de Thoughput.

6.3 Síntese Lógica

A utilização dos *transceivers* GTH demandam cuidados com relação às referências de *clock* utilizadas, as quais devem ser tratadas durante a fase de síntese lógica. Na determinação das *constraints* de tempo, é necessário que sejam adicionadas às diferentes referências de *clock* do sistema, bem como a relação de fase entre eles.

As *constraints* podem estar localizadas em diferentes arquivos. Nesse projeto, devido a utilização do *wizard* para geração do módulo *GTH_gtwizard*, são geradas automaticamente *constraints* pertinentes

a este módulo. As demais referências necessitam ser adicionadas manualmente para as corretas definições de *slack time* do projeto.

A Figura 46 apresenta as *constraints* de tempo utilizadas no projeto. Inicialmente são determinadas as referências de *clock* do sistema com seus respectivos períodos, a partir dos comandos *create_clock* e *create_generated_clock*. Nessa etapa foram adicionados os principais *clocks* do sistema, como o *clock* de referência externo (Q8_CLK0_GTREFCLK_PAD_P_IN) de 156,25MHz, o *clock* do sistema (SYSCLK_IN_P) de 200MHz e o *clock* interno do módulo GTH_gtwizard (*drpclk_i*) de 100MHz. Os comandos create_clock também são utilizados nas *constraints* internas do módulo *GTH_gtwizard*, os quais especificam os *clocks* de referência para a geração dos *clocks* do usuário (TXOUTCLK e RXOUTCLK).

Após a realização da síntese lógica foram detectados pontos onde haviam problemas de *slack time* negativo entre diferentes origens de *clock* do sistema. No entanto as relações entre essas referências podem ser desconsideradas, pois, não há relação de fase entre esses *clocks*. Para isso, foi utilizado o comando set_false_path essas referências que acusaram problemas de *slack time.* As relações de fase ignoradas através do comando *set_false_path* foram:

- Os *clocks* do usuário (*USRCLKs*) gerados através do módulo MMCM (*clkout0* e *clkout_0_1*) e o *clock* de referência externa (*Q8_CLK0_GTREFCLK_PAD_P_IN*);
- O *clock* do sistema (*SYSCLK_IN_P*) e o *clock* de referência externa (*Q8_CLK0_GTREFCLK_PAD_P_IN*) ;
- O *clock* interno do módulo GTH_gtwizard (*drpclk_i*) e o *clock* de referência externa (*Q8_CLK0_GTREFCLK_PAD_P_IN*).

C:/Users/Vini/Dropbox/NEWXGETH/NEWXGETH.srcs/constrs_1/new/contraints_stable.xdc

```
585
586 create_clock -period 6.400 -name Q8_CLK0_GTREFCLK_PAD_P_IN -waveform {0.000 3.200} [get_ports Q8_CLK0_GTREFCLK_PAD_P
587 create_clock -period 5.000 -name SYSCLK_IN_P -waveform {0.000 2.500} [get_ports SYSCLK_IN_P]
588 create_generated_clock -name drpclk_i -source [get_pins drpclk_i_reg/C] -divide_by 2 [get_pins drpclk_i_reg/Q]
```

c:/Users/Vini/Dropbox/NEWXGETH/NEWXGETH.srcs/sources_1/ip/gtwizard_0/gtwizard_0.xdc

```
 73 create_clock -period 3.10303 [get_pins -hier -filter {name=~*gt0_gtwizard_0_i*gthe2_i*TXOUTCLK}]
 74 create_clock -period 3.10303 [get_pins -hier -filter {name=~*gt0 gtwizard 0 i*gthe2 i*RXOUTCLK}]
```

C:/Users/Vini/Dropbox/NEWXGETH/NEWXGETH.srcs/constrs_1/new/contraints_stable.xdc

```
624
625 set_false_path -from [get_clocks Q8_CLK0_GTREFCLK_PAD_P_IN] -to [get_clocks clkout0]
626 set_false_path -from [get_clocks SYSCLK_IN_P] -to [get_clocks drpclk_i]
627 set_false_path -from [get_clocks clkout0] -to [get_clocks Q8_CLK0_GTREFCLK_PAD_P_IN]
628 set_false_path -from [get_clocks clkout0_1] -to [get_clocks Q8_CLK0_GTREFCLK_PAD_P_IN]
629 set_false_path -from [get_clocks SYSCLK_IN_P] -to [get_clocks Q8_CLK0_GTREFCLK_PAD_P_IN]
630 set_false_path -from [get_clocks drpclk_i] -to [get_clocks Q8_CLK0_GTREFCLK_PAD_P_IN]
```

Figura 46 - Constraints de tempo.

6.4 Floorplanning

A etapa de floorplaning foi realizada visando o posicionamento dos módulos de forma a ficarem o mais próximo dos canais GTHE2_CHANNEL e GTHE2_COMMON utilizados. Na Figura 47 é apresentado a localização dos módulos após a síntese física da interface. Dentre os módulos, o GTH_gtwizard (em verde) é o módulo mais próximo dos canais GTHE2_CHANNEL e GTHE2_COMMON, já que o mesmo realiza instanciação dos canais. O módulo XGETH_TESTER foi destacado em duas cores distintas, em vermelho os módulos pertinentes aos testes da RFC2544 e em azul os módulos PCS e MAC. Os demais módulos *Control Inteface* (em magenta), *Serial Interface,* (em

marrom) e I2C Interface (em amarelo), e as lógicas pertencentes ao topo do projeto (em branco) também são posicionados no mesmo quadrante (X1Y9).

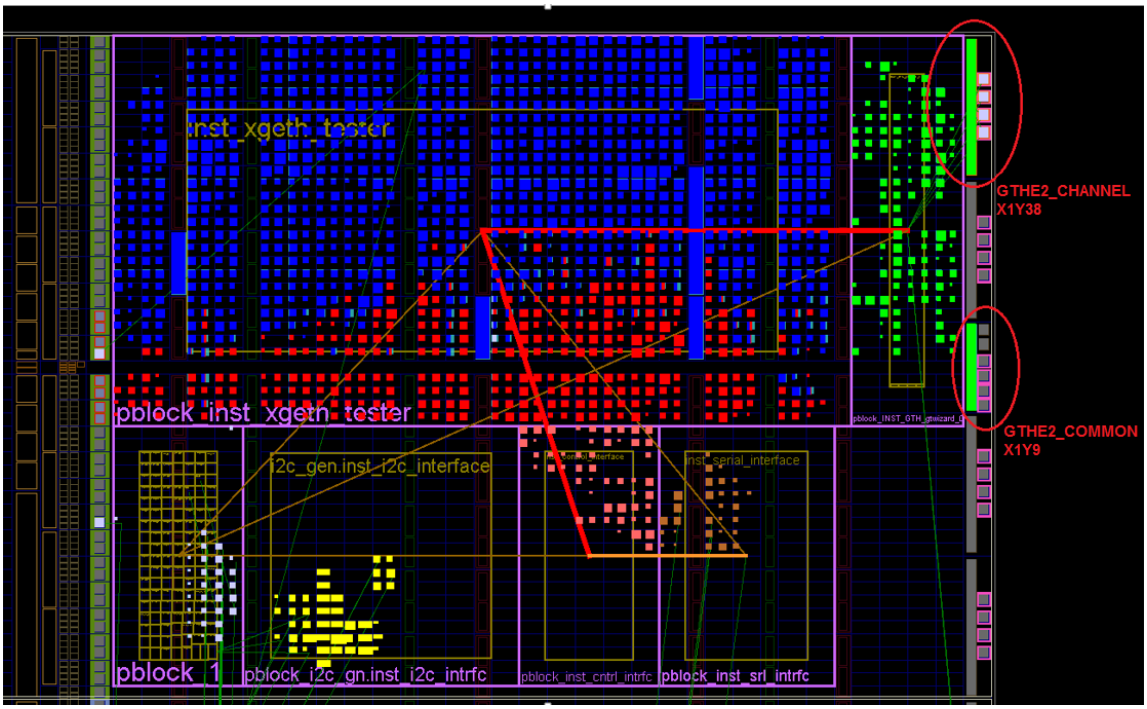


Figura 47 – Floorplaning.

6.4.1 Utilização de área

A partir do relatório gerado ao final da síntese física pode-se observar uma baixa utilização de lógica necessária para a implementação da interface 10GBASE-R e das demais interfaces de configuração e testes. Conforme apresentado na Tabela 4 a lógica utilizada representou em torno de 1% da área disponível na FPGA. Entretanto a utilização de buffers globais (BUFG) é em torno de 31,25% devido ao uso de diferentes referências de *clock* na interface.

A partir da Figura 48, pode-se ter uma melhor visualização da dimensão da utilizada em relação ao total de recursos disponíveis no dispositivo FPGA. Fica evidente que o custo em área de uma interface 10GBASE-R é pequeno.

É importante notar que a alta utilização de *buffers* globais é devido à grande quantidade de referências de *clock* do canal do *transceiver GTH*, no entanto, esses buffers podem ser compartilhados entre os demais canais conectados aos módulos SFP+.

Tabela 4 - Recursos utilizados na interface 10GbE

Tipo de Recurso	Utilizados	Total na FPGA	Utilização (%)
LUTs	4683	433200	1.08
Registradores – FF/Latch	4688	866400	0.54
BRAM	6	1470	0.41
GTHE2_CHANNEL	1	36	2.78
BUFG	10	32	31.25
MMCME2_ADV	2	20	10.00

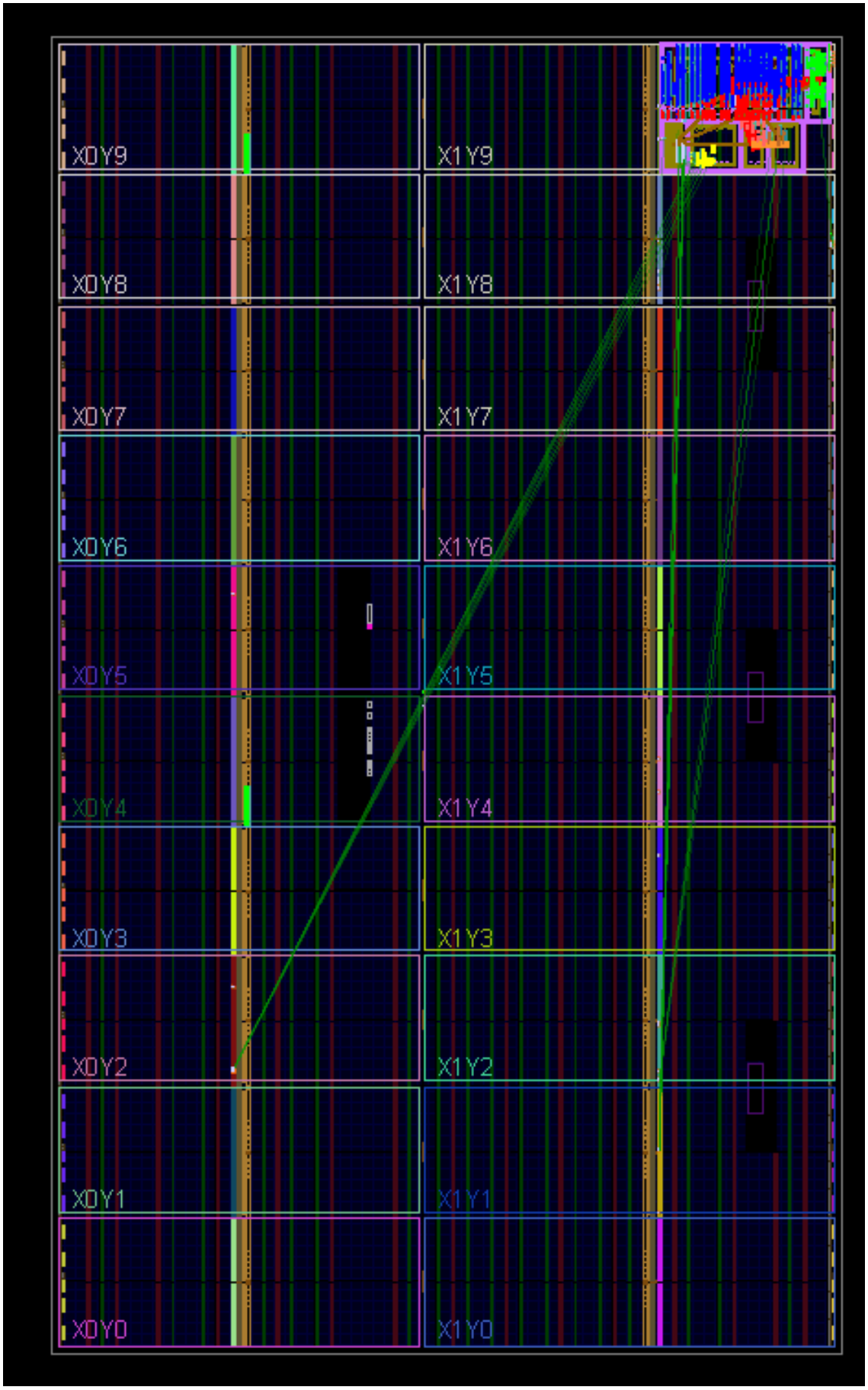


Figura 48- Posicionamento da interface no dispositivo FPGA.

6.5 Validação da Interface e Testes de Desempenho

Nesta Seção serão apresentados as ferramentas e métodos utilizados para a verificação do sistema. Essa etapa valida o funcionamento da interface desenvolvida através das da ferramenta *Chipscope*, um software de teste desenvolvido pelo Autor e o software de teste da RFC2544 desenvolvido em parceria entre a empresa Teracom e o GAPH.

A métrica fundamental para a verificação do funcionamento do sistema é através da determinação da vazão do sistema. Nessa Seção também será apresentada a derivação do cálculo de vazão utilizado nos testes a serem realizados conforme os recursos do hardware utilizados.

6.5.1 Cálculo da vazão da interface

O cálculo da vazão efetiva é baseado na quantidade de pacotes úteis recebidos na camada de enlace. Conforme apresentado na Seção 2.2, o pacote Ethernet possui 7 bytes de preâmbulo, 1 byte de delimitador de início de pacote (SFD) e 4 bytes de CRC que são removidos na camada MAC, os quais não são repassados à camada superior. Além disso, o mínimo *inter-frame gap* entre pacotes deve ser de 12 bytes, esse também devem ser contabilizados no cálculo da vazão. Esses dados são considerados overhead de transmissão e não devem ser considerados dados úteis. A vazão do sistema também deve considerar o tamanho dos pacotes transmitidos, pois quanto menor o pacote, maior será o overhead causado por esses campos, já que eles possuem o mesmo overhead independentemente do *payload* transmitido.

Outro ponto necessário para o cálculo da vazão é relativo à forma ao qual o número o *inter-frame gap* é gerado no testador XGETH TESTER. Conforme a apresentado na Seção 5.3, a inserção do *inter-frame gap* é feita através do registrador 0x1003. O *intergap* consiste em 8 bytes vezes o valor armazenado do registrador, com exceção do valor 1, onde são inseridos 12 bytes.

Com isso, o cálculo da vazão é dado pelas equações (9) e (10), onde T é a vazão em Mbps, I é o número <IDLE_NUMBER> armazenado no registrador 0x1003 e P_{size} o tamanho dos pacotes transmitidos, em bytes.

$$T = \frac{P_{size}}{P_{size} + (I \times 8) + 12} \times 10Gb/s \qquad \text{I} > 1 \tag{9}$$

$$T = \frac{P_{size}}{P_{size} + 24} \times 10Gb/s \qquad \text{I} = 1 \tag{10}$$

6.5.2 Software de teste

A o software desenvolvido pelo Autor escreve os parâmetros necessários no registrador via USB para dar início a um teste de *throughput*. Durante o teste são feitas leituras do registrador status (0x1001) aguardando o término no processo. Ao final do teste são lidos os registradores do testador e realizado o cálculo da vazão. A Figura 49 apresenta a configuração e execução do software de teste, com pacotes de 64 bytes e *intergap* igual a 1. Ao final da execução, os registradores 0x14 e 0x15 contém o número de pacotes da rajada que retornaram ao testador (parte baixa e alta respectivamente). Se a quantidade de pacotes enviados for igual ao de pacotes recebidos, então é calculado o valor que corresponde ao *thoughput* alcançado. Nesse exemplo todos os pacotes retornaram e o *thoughput*

calculado foi de 7272.73Mb/s (de acordo com a equação 10). A execução do teste foi realizada através do *loopback* da fibra óptica entre o transmissor e o receptor do módulo SFP+.

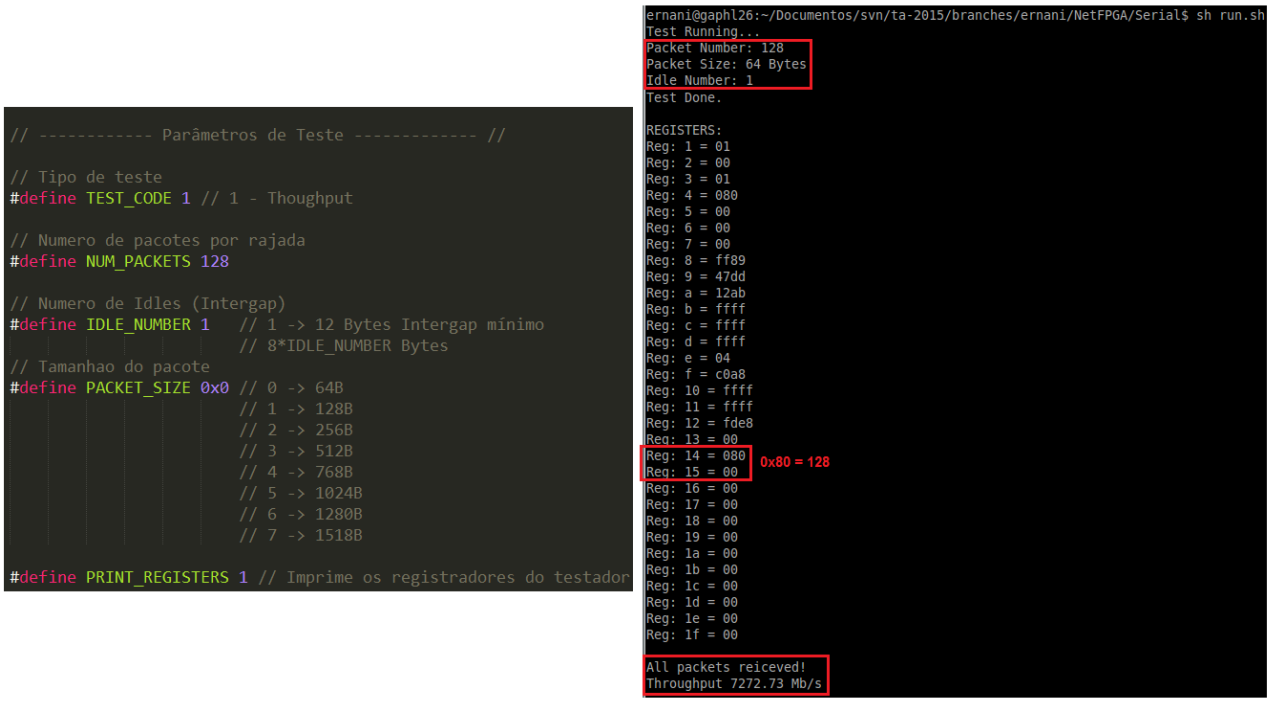


Figura 49 – Exemplo de configuração e execução do software de teste.

6.5.3 Chipscope

A partir da ferramenta *Chipscope* foi possível visualizar o fluxo de dados na interface XGMII, e os demais sinais pertinentes para avaliação do funcionamento físico da interface. Para gerar tráfego de dados, foi utilizado o software desenvolvido pelo Autor para execução rajadas de dados.

Para a visualização da transmissão e recepção dos pacotes, a ferramenta *Chipscope* adiciona ILAs (*Integrated Logic Analyzer*), que permitem o monitoramento do lógico através da coleta de dados em pontos do sistema determinados pelo usuário. Esses sinais começam a ser amostrados a partir de um evento pré-determinado configurado a partir dos sinais amostrados. Como disparador (*trigger*) de um evento, foi utilizado o barramento de controle de transmissão da interface XGMII. Quando qualquer valor diferente de 0xFF for detectado, indicando que algum dado útil começou a ser transmitido através do barramento, a ferramenta inicia a amostragem dos sinais, dentro de uma janela de tempo.

A Figura 50 apresenta uma rajada executada com a mesma configuração utilizada no exemplo da Figura 49. Nessa execução pode-se verificar os eventos numerados em vermelho:

1. O contador *echo_sequence_counter*: contabilizou os 128 pacotes recebidos.
2. Sinal *echo_received_packet* notificando através de pulsos cada pacote recebido pelo módulo *echo_receiver*.
3. Nenhum erro detectado nos pacotes recebidos durante a transmissão (pkt_rx_err).
4. Transmissão da rajada na interface XGMII.
5. Recepção da rajada de dados na interface XGMII.
6. Distância (latência) entre o término da transmissão dos pacotes e a recepção nos mesmos pelo receptor na interface XGMII
7. Barramento XGMII ocioso após a conclusão da rajada de dados.

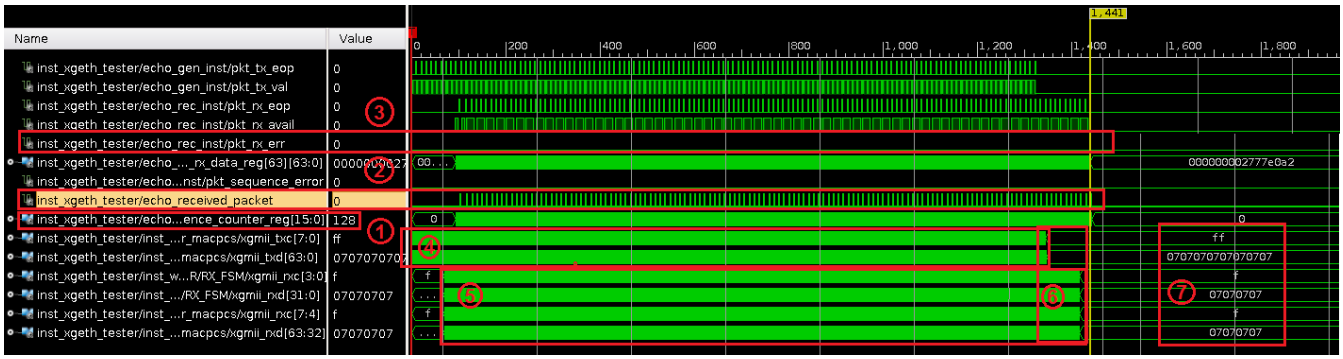


Figura 50 – Rajada de dados monitorado através ferramenta Chipscope.

Na Figura 51 é possível observar pacotes sendo transmitidos (1) e o *intergap* entre esses pacotes (2). Nessa configuração foram utilizados *intergaps* de 800 bytes e pacotes de 64 bytes. A vazão nessa rajada foi de aproximadamente 730Mb/s.

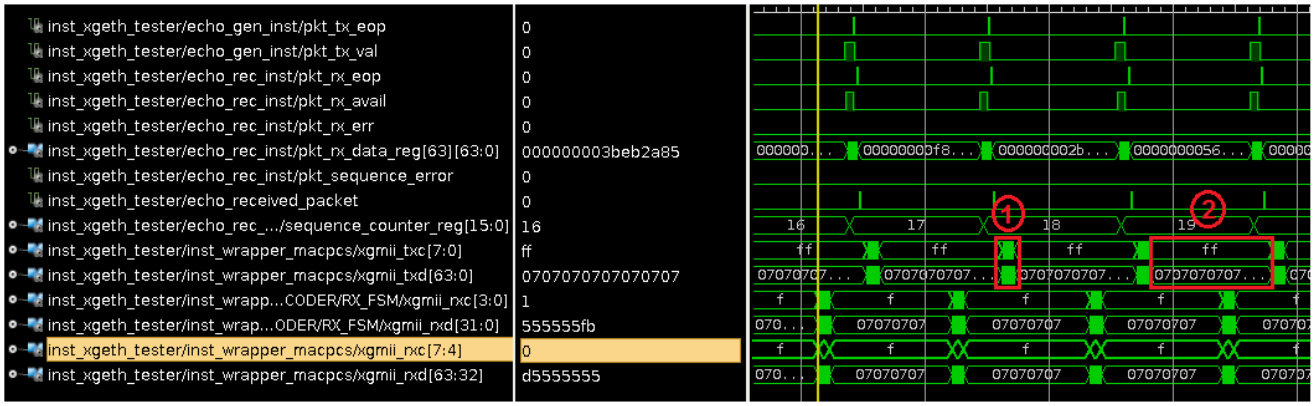


Figura 51 – Intergap entre pacotes.

6.6 Testes da RFC2544

Os testes realizados a partir do software de teste e a ferramenta *Chipscope* apresentados até então realizaram apenas a verificação física da interface. Com a utilização do software de testes da RFC2544 é possível validar o desempenho do canal com diferentes tamanhos de pacotes, explorando então a taxa máxima obtida na interface de alta velocidade.

Foram utilizadas duas formas de conexão com a interface. A primeira forma (Figura 52) consiste na conexão através fibra óptica em *loopback* entre o transmissor e receptor, como utilizado nos testes anteriores. Em uma segunda forma (Figura 53) o teste foi efetuado através da conexão do dispositivo com um *switch* 10GbE de modelo DM4100 e fabricado pela empresa Teracom. Esse teste permite então a verificação do dispositivo através de duas interfaces distintas. Nessa configuração o tráfego de dados é gerado através do testador, recebido pelo switch através da porta 25, retornado internamente pela porta 26 ao qual é configurada em *loopback* interno e retornado pela porta 25 para o testador.

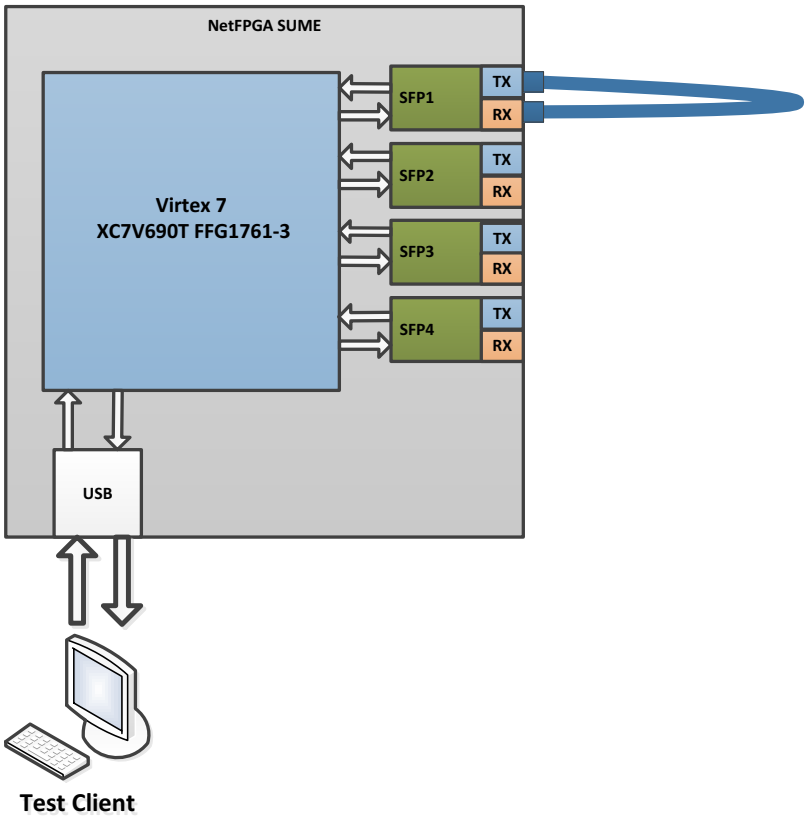


Figura 52 - Configuração para teste em loopback.

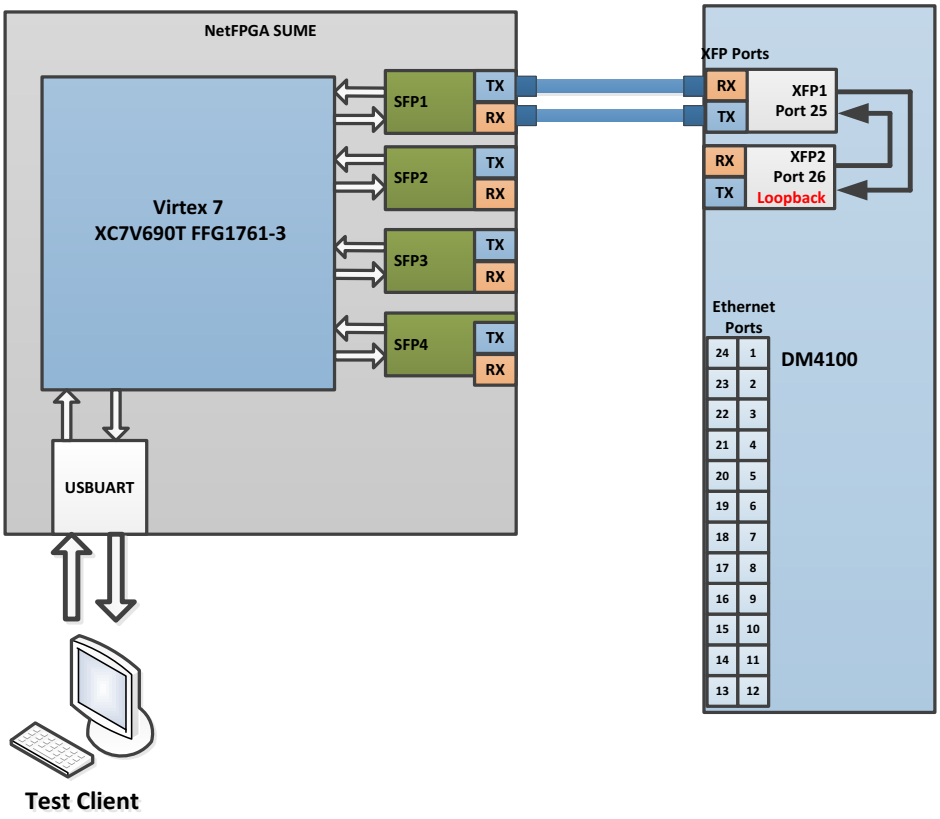


Figura 53 - Configuração para teste contra o switch.

A Figura 54 apresenta a foto do ambiente de teste. Nesta foto estão identificadas a placa de prototipação NetFPGA SUME com dispositivo Virtex 7 XCV690T, o switch DM4100, a fonte de alimentação e as fibras ópticas que formam o ambiente de teste. Para execução dos testes, um computador é conectado via USBUART para transferência dos parâmetros necessários ao teste. Notar que quando é feito o teste em loopback a fibra óptica é conectada ao transmissor e receptor do mesmo SFP+. Já no teste com o switch, como apresentado na figura, são conectados os módulos SFP+ da NetFPGA com módulo XFP do switch, o qual também utiliza o protocolo 10GBASE-R.

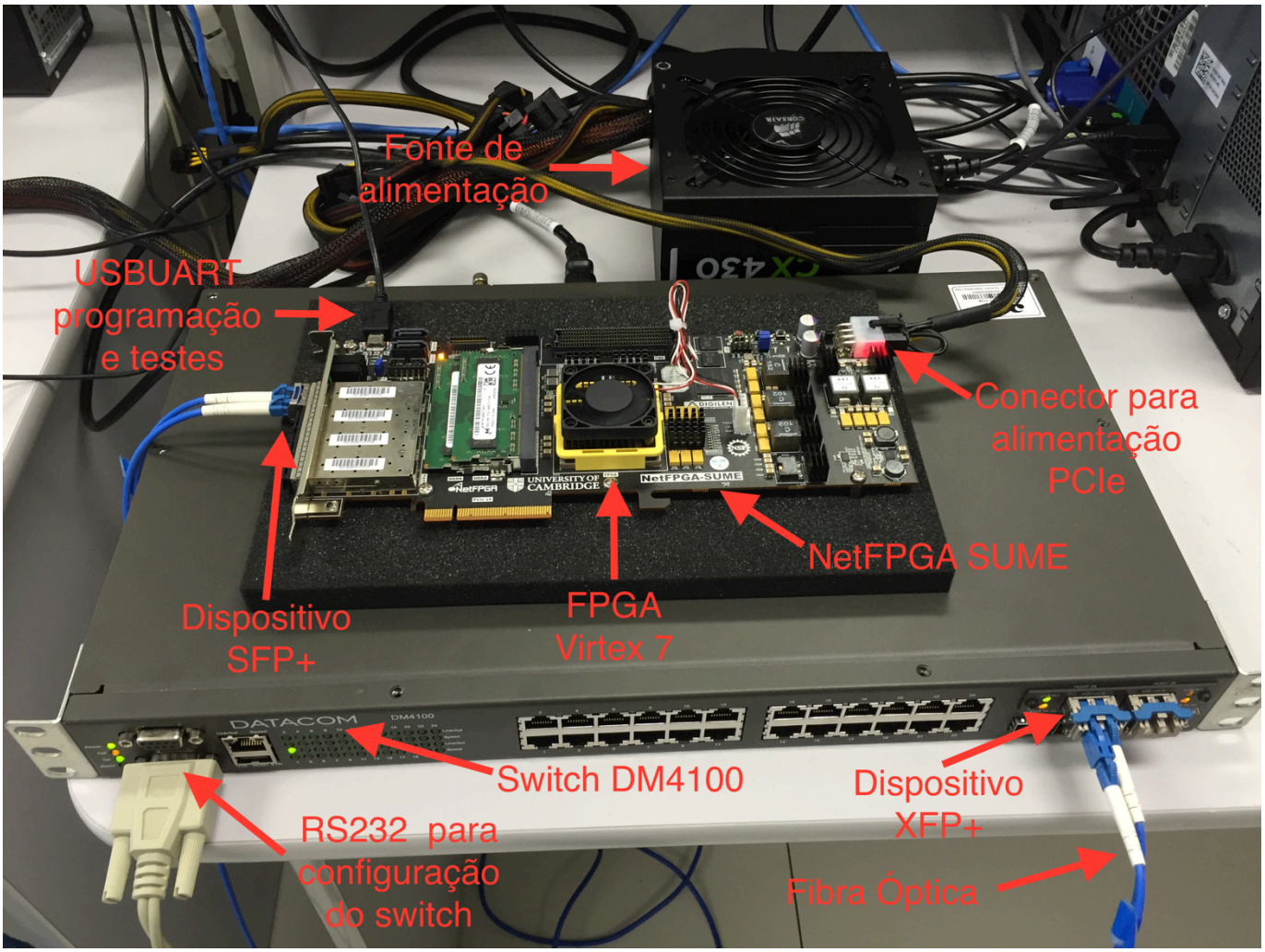
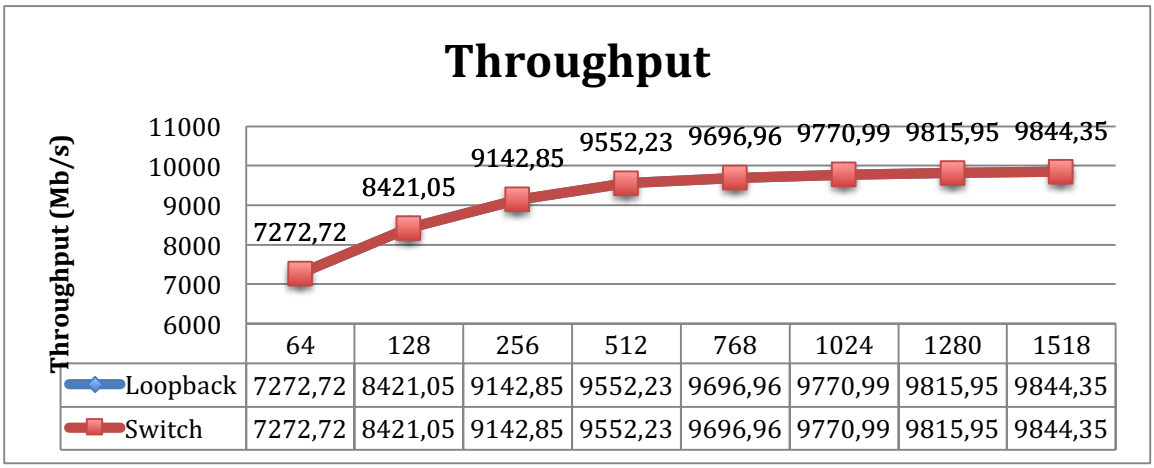


Figura 54 - Ambiente de teste.

A verificação foi realizada através dos testes de *thoughput*, latência e *back-to-back* e os resultados são apresentados nas Figura 55, Figura 56 e Figura 57 respectivamente, para testes em modo *loopback* e através do *switch*.

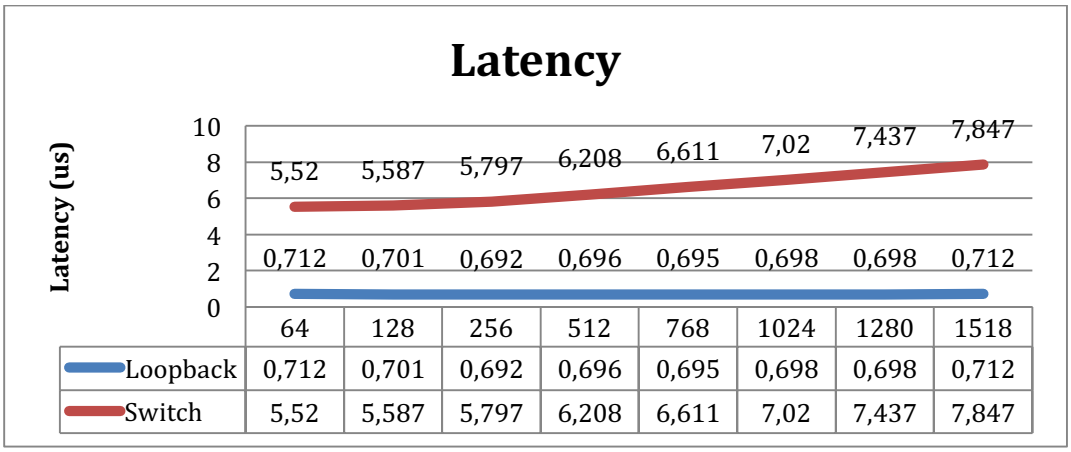
A Figura 55 apresenta o teste de vazão (*thoughput*). O eixo X apresenta o tamanho dos pacotes e o eixo Y a vazão obtida. Conforme observamos foi possível atingir a taxa máxima relativa a todos os tamanhos de pacote através do teste de *thoughput*, seja com a saída em *loopback* ou através do *swith.*



	64	128	256	512	768	1024	1280	1518
Loopback	7272,72	8421,05	9142,85	9552,23	9696,96	9770,99	9815,95	9844,35
Switch	7272,72	8421,05	9142,85	9552,23	9696,96	9770,99	9815,95	9844,35

Figura 55 - Teste de vazão (throughput).

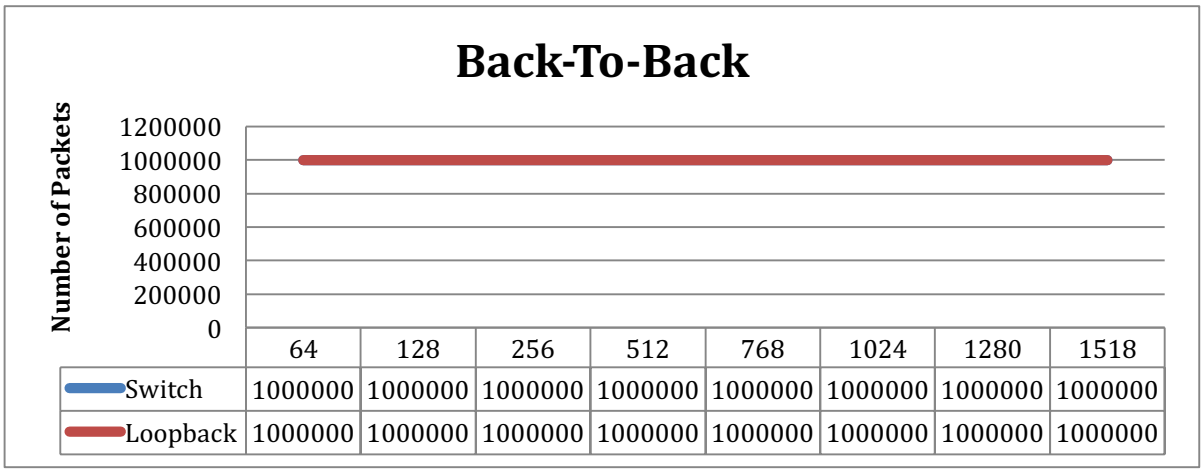
A Figura 56 apresenta o teste de latência. O eixo X apresenta o tamanho dos pacotes e o eixo Y a latência obtida. Como esperado, com o sistema em *loopback* a latência é praticamente constante, igual a 0,712 microssegundos. Quando realizados os testes através do *switch* a latência aumenta proporcionalmente ao tamanho do pacote. Este aumento deve-se ao armazenamento dos pacotes internamente no switch para sua interpretação e direcionamento para a devida porta de saída.



	64	128	256	512	768	1024	1280	1518
Loopback	0,712	0,701	0,692	0,696	0,695	0,698	0,698	0,712
Switch	5,52	5,587	5,797	6,208	6,611	7,02	7,437	7,847

Figura 56 - Teste de latência (latency).

A Figura 57 apresenta o teste de *back-to-back*. O eixo X apresenta o tamanho dos pacotes e o eixo Y o número de pacotes injetados. Neste teste foram transmitidos até um máximo de um milhão de pacotes para todos os tamanhos de pacotes, sem que houvessem perdas de pacotes.



	64	128	256	512	768	1024	1280	1518
Switch	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
Loopback	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000

Figura 57 - Teste de back-to-back.

7 CONCLUSÃO E TRABALHOS FUTUROS

Neste documento foram apresentados os conceitos necessários para a implementação e testes de uma interface de comunicação 10GbE em dispositivos FPGAs. Os conhecimentos desenvolvidos nesse documento complementam os estudos em áreas de grande importância dentro do curso de Engenharia da Computação, como a de redes de computadores e de desenvolvimento de sistemas em dispositivos FPGAs. A partir desses conhecimentos o projeto desenvolvido pelo Autor atingiu os objetivos propostos para o trabalho de conclusão de curso.

O projeto realizado não constituiu uma aplicação específica, provendo uma interface de comunicação de alta velocidade validada e que pode ser integrada a diversas aplicações que necessitam de transmissão de dados em taxas de 10Gbps. Além disso, os conhecimentos adquiridos podem ser aplicados para outros projetos que utilizam *transceivers* GTH para outros fins de comunicação.

Dentro dos assuntos relacionados com a área de redes de computadores, foram abordados conceitos avançados sobre as estruturas que compõem a camada física de um sistema de comunicação e que são utilizados em protocolos largamente utilizados nos dias de hoje. Esses conhecimentos são de grande importância para desenvolvimento de sistemas para a área de telecomunicações.

Com relação ao desenvolvimento em dispositivos FPGAs, foram aprofundados os conhecimentos sobre o fluxo de projeto. Esses conhecimentos podem ser aplicados à diferentes projetos, tornando-os não só aplicáveis a interfaces de comunicação, mas para qualquer tipo de projeto envolvendo FPGAs.

Esse estudo e desenvolvimento pode servir como base para outros projetos a serem realizados em trabalhos futuros. A partir da interface 10GbE desenvolvida, podem ser implementados dispositivos de comunicação como switches e roteadores com múltiplas portas de alta velocidade, ou até mesmo aplicações com protocolos proprietários integrados aos canais 10GbE. Por outro lado, os conhecimentos obtidos sobre *transceivers* GTH podem ser utilizados desenvolvimento de interfaces de comunicação que utilizam outros protocolos, como o protocolo PCIexpress para comunicação de alta velocidade através de barramentos.

8 REFERÊNCIAS

[ALT15] Altera Corporation "Trasnceivers Technology". Disponível em: https://www.altera.com/solutions/technology/transceiver/overview.highResolutionDisplay.html, novembro, 2015'

[ATH15] Athavale, A. "High-Speed Serial I/O Made Simple". Disponível em: http://arrc.ou.edu/~rockee/RIO/serialio-book.pdf, abril, 2015, 31p.

[BRO09] Brown, S. Vranesic. "Fundamentals of Digital Logic with VHDL Design", McGraw-Hill, Third Edition, 2009, pp. 57-62.

[CIS15] Cisco Systems. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014–2019". Disponível em: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf, fevereiro, 2015.

[COM13] Comer, D. "Internetworking with TCP/IP Volume One", Pearson, sixth edition, maio, 2013.

[DIG15] Digilent Inc. "Net FPGA Sume". Disponível em: http://digilentinc.com/sume, setembro, 2015.

[FRA00] Frazier, H. "IEEE P802.3ae - 10 Gigabit Ethernet Task Force: XGMII Update", Disponível em: http://www.ieee802.org/3/ae/public/jul00/frazier_1_0700.pdf, julho, 2000.

[FUH13] Fu, H. "Leveraging 7 Series FPGA Transceivers for High-Speed Serial I/O Connectivity". Disponível em: http://www.xilinx.com/support/documentation/white_papers/wp431-Trans-Serial-Connectivity.pdf, março, 2013.

[IEE02] The Institute of Electrical and Electronic Engineers Inc. (IEEE) "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architeture", Disponível em: http://www.ieee802.org/secmail/pdfYD89wBpRqH.pdf, março, 2002.

[IEE12] The Institute of Electrical and Electronic Engineers Inc. (IEEE) "IEEE Standard for Ethernet", sections one and four Disponível em: https://standards.ieee.org/about/get/802/802.3.html, agosto, 2012.

[IET81] Internet Engineering Task Force, "RFC792". Disponível em: https://tools.ietf.org/html/rfc792https://tools.ietf.org/html/rfc792, setembro, 1981.

[IET99] Internet Engineering Task Force, "RFC2544". Disponível em: https://www.ietf.org/rfc/rfc2544.txt, março, 1999.

[MIC14] Microsoft Support "The OSI Model's Seven Layers Defined and Functions Explained", Disponível em: https://support.microsoft.com/en-us/kb/103884, março, 2014.

[OPE08] Open Cores "Ethernet 10GE MAC", Disponível em: www.opencores.org/project,xge_mac, maio, 2008.

[SFF09] SFF Committee. "SFF-8431 SFP+ 10 Gb/s and Low Speed Electrical Interface". Disponível em: ftp://ftp.seagate.com/sff/SFF-8431.PDF, julho 2009.

[SPU14] Spurgeon, C., Zimmerman, J. “Ethernet: The Definitive Guide”, O’Reilly, second edition, abril, 2014,

[TAN15] Tang, S. “Advantages of Ethernet vs SONET/SDH”. Disponível em: http://www.tccomm.com/Content/pdf/Literature/White-Paper-Ethernet-Advantages.pdf, setembro, 2015.

[XIL09] Xilinx Inc. “Mixed-Mode Clock Manager (MMCM) Module (v1.00a)”, Junho, 2009. Disponível em: http://www.xilinx.com/support/documentation/ip_documentation/mmcm_module.pdf.

[XIL13a] Xilinx Inc. “Vivado Design Suite Tutorial - Synthesis”, UG901, junho, 2013, 8p. Disponível em: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug901-vivado-synthesis.pdf,

[XIL13b] Xilinx Inc. “Vivado Design Suite Tutorial - Implementation”, UG904 (v2013.4). Dinsponível em: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_3/ug904-vivado-implementation.pdf, dezembro, 2013, pp. 5-6.

[XIL13c] Xilinx Inc. “Vivado Design Suite Tutorial – Using Constraints”, UG903. Capturado em: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf, junho, 2013, pp. 33-39.

[XIL14a] Xilinx Inc. “LogicCORE IP 7 Series FPGAs Transceivers Wizard 3.3”, PG168. Capturado em: http://www.xilinx.com/support/documentation/ip_documentation/gtwizard/v3_3/pg168-gtwizard.pdf, junho, 2014.

[XIL15a] Xilinx Inc. “7 Series FPGAs GTX/GTH Transceivers User Guide”, UG476 (v1.11). Capturado em: http://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf, fevereiro, 2015.

[XIL15b] Xilinx Inc. “High Speed Serial”, outubro, 2015. Capturado em: http://www.xilinx.com/products/technology/high-speed-serial.html