Um Canal de Comunicação Inter-FPGA com Módulo de Detecção de Erro

Lucas Melo; Silvio Santana; Silva-Filho, A.G.; Manoel E. Lima
Centro de Informática (CIn)
Universidade Federal de Pernambuco (UFPE)
Recife, PE, Brazil
{ltm3, shcs, agsf, mel}@cin.ufpe.br

Victor Medeiros, Marcelo Marinho
Departamento de Estatística e Informática
Universidade Federal Rural de Pernambuco (UFRPE)
Recife, PE, Brazil
{victor, marinho}@deinfo.ufrpe.br

Resumo **— Atualmente, sistemas envolvendo múltiplos FPGAs são utilizados em diversas aplicações científicas. Tais sistemas requerem um barramento de dados dedicado para comunicação entre FPGAs, o qual pode ser feito por meio de interfaces do tipo LVDS (Sinalização Diferencial de Baixa Tensão). Outro fator importante é que o roteamento que interconecta os pinos LVDS na plataforma deve ser desenvolvido com precisão para evitar instabilidades na comunicação. Infelizmente, muitas plataformas disponíveis no mercado não observam tais restrições, limitando a taxa de transferência no barramento. Este trabalho apresenta um canal de comunicação bi-direcional inter-FPGAs baseado em uma interface *DDR* voltado para esse tipo de plataforma. Esta abordagem promove uma comunicação estável entre esses dispositivos sem a utilização de pinos LVDS. Um módulo de detecção de erro também foi desenvolvido para garantir a integridade das transferências e corrigir possíveis erros no barramento. O canal foi validado em uma plataforma comercial. Os resultados de síntese e desempenho também são apresentados nesse trabalho.**

Keywords — canal de comunicação Inter-FPGAs; comunicação full-duplex; CRC.

I. INTRODUÇÃO

Plataformas que envolvem múltiplos FPGAs têm sido alvo de diversas áreas como prototipação de MPSoCs, aceleração de algoritmos e criptografia [1] [2]. Para que esses sistemas possam funcionar de forma eficiente, utilizando paralelamente recursos existentes nos FPGAs, uma comunicação eficiente deve existir entre os FPGAs disponíveis na plataforma. Geralmente esse tipo de comunicação, em FPGAs de última geração, se dá por meio de interfaces tipo LVDS (Sinalização Diferencial de Baixa Tensão) [3]. Esse tipo de sinalização permite o envio de sinais em alta velocidade através de um par diferencial de fios paralelos. A utilização desse recurso permite que a transmissão de dados entre os dispositivos possa ser realizada de forma mais eficiente, possibilitando uma comunicação mais segura contra interferências eletromagnéticas. Esta configuração possibilita que barramentos alcancem taxas de transferências da ordem de 10Gbps com a utilização de dispositivos mais avançados, como é o caso dos FPGAs da família *Virtex*, da *Xilinx* [4] e *Stratix V*, da *Altera* [5].

Atualmente diversos FPGAs suportam interfaces LVDS e se devidamente alocados na plataforma podem prover uma comunicação de dados a uma alta taxa de transmissão com a utilização de seus transceptores. Entretanto, algumas plataformas disponíveis no mercado não foram projetadas para acomodar os pinos com recursos LVDS do FPGA em suas vias de comunicação, impossibilitando assim, o uso de transceptores LVDS e por conseguinte, prejudicando a implementação de canais de comunicação com alto desempenho. Outros fatores como distância entre trilhas, resistência e capacitância balaceadas são importantes e necessitam ser observados pelos fabricantes desse tipo de plataforma. Uma transmissão de dados realizada sem esses recursos, pode acarretar em erros na transferência de dados quando a taxa de transmissão for alta, uma vez que não há garantia de que será mantida a integridade dos dados. Isso ocorre porque todos os pinos de I/O dos FPGAs, estão, em geral, sujeitos a interferências físicas. Devido a este possível baixo desempenho na transmissão de dados, os sistemas que utilizam estes recursos terão que reduzir a velocidade de seus módulos lógicos para se adequar ao canal e assim diminuir as incidências de erro na comunicação.

Este trabalho apresenta um canal dedicado de comunicação bi-direcional (*full-duplex*) síncrono, independentemente do uso da interface LVDS. Para garantir uma boa taxa de transmissão o transmissor é baseado em uma interface *Double Data Rate* (*DDR*) e para assegurar a integridade dos dados foram desenvolvidos módulos de detecção de erro baseados na geração e verificação *Cyclic Redundancy Check* (CRC).

O trabalho está organizado como se segue: na Seção II alguns trabalhos relacionados são apresentados; na Seção III é apresentada a plataforma Gidel PROCStarIII utilizada nos experimentos; na Seção IV é apresentada a arquitetura desenvolvida; na Seção V, são exibidos os resultados dos experimentos realizados; por fim, na seção VI a conclusão é apresentada.

II. TRABALHOS RELACIONADOS

Nesta seção alguns trabalhos com foco em comunicação inter-FPGAs são apresentados. O trabalho [6] apresenta uma arquitetura para esta finalidade e o trabalho [7] propõe uma solução para otimização da comunicação entre diferentes dispositivos FPGAs.

Em [6] é exibida uma arquitetura para comunicação inter-FPGA *full-duplex* com 18 trilhas LVDS de transmissão. A arquitetura é baseada na interface *DDR* e é composta por um transmissor e um receptor. Os autores demonstram que é possível alcançar taxas de transferência na ordem de 10 Gbps utilizando a abordagem. A arquitetura foi desenvolvida para ser

implementada em um cluster de alto desempenho (*PARAMNet-3*) [8]. A arquitetura compõe um *switch* de rede permitindo que o roteamento de pacotes seja realizado em alta velocidade. Cada placa que faz parte do *switch* é formada por quatro FPGAs *Virtex-4* da *Xilinx* e os Transceptores (*Multi-Gigabit Transceivers* – MGT's) disponíveis em cada FPGA são utilizados para comunicação com as outas placas do sistema.

O transmissor é formado por uma unidade de controle, buffers I/O LVDS e por um módulo *Serializer/Deserializer* (SERDES), esses dois ultimos são recursos de I/O já disponíveis nos FPGAs *Xilinx Virtex-4*. O módulo SERDES é um serializador/deserializador que recebe os dados da aplicação de forma paralela e os serializa. Os buffers de saída LVDS recebem esses dados serializados e implementam a saída em 18 trilhas de pares diferenciais. Cada trilha tem uma taxa de transferência de 625 Mbps, somando no total uma taxa de cerca de 10 Gbps.

O receptor tem seu relógio gerado no transmissor, garantido que o funcionamento de ambas arquiteturas estejam sincronizadas. Ele é composto de um módulo de controle, delays, buffers I/O LVDS e um módulo SERDES. Os módulos de *delay* são responsáveis em introduzir um *jitter* de 5-10 ps na entrada de relógio proveniente do transmissor. Essa estratégia respeita a janela válida de dados e é necessária para garantir que os sinais sejam recebidos no receptor corretamente. O relógio utilizado no receptor é de 312.5 MHz e está sincronizado com o transmissor.

A detecção de erro de comunicação é realizada na camada de rede do *switch* e os autores desenvolveram um *test engine* para gerar erros no canal de comunicação. Este *engine* é responsável por testar cada trilha LVDS e duas instâncias foram implementadas em cada FPGA. O ambiente de testes é formado por um processador *PowerPC* executando uma aplicação que se comunica com a arquitetura por meio de uma interface *UART*. É possível configurar parâmetros de teste e visualizar estatísticas na aplicação.

Em [7] é apresentada uma solução para otimização de comunicação inter-FPGAs utilizando um canal de adaptação. O trabalho demonstra que cada plataforma possui características próprias e que quase sempre, ao migrar-se o projeto para uma outra plataforma, é necessário alterar a arquitetura para se adaptar às mudanças necessárias. Largura do barramento de comunicação, transmissão ponto-a-ponto ou LVDS, são algumas das opções que poderiam ser parametrizadas abstraindo do usuário a necessidade de alterar sua arquitetura. É nesse aspecto que o autor do trabalho se concentra e propõe um canal parametrizável capaz de se auto-adaptar a arquitetura alvo.

Inicialmente é apresentada uma arquitetura base composta basicamente por um módulo de controle, um módulo *phy* e FIFOs de transmissão (*TX*) e recebimento (*RX*). A arquitetura suporta interfaces *SDR/DDR* e transmissão ponto-a-ponto ou LVDS. Também pode ser utilizado módulos SERDES para serialização/deserialização de dados e é possível definir se a comunicação será unidirecional ou bidirecional. Por fim, a arquitetura permite que a largura do barramento seja ajustável.

O canal de adaptação, chamado *chAdapt*, é formado por um módulo em hardware que auto-configura a transmissão de acordo com as características da plataforma. Para que isso aconteça, inicialmente uma transmissão de uma sequência de bits ocorre de forma repetida entre transmissor e receptor para identificação de possíveis atrasos na comunicação. Nesse mesmo tempo todos os controles de conexão são otimizados e o alinhamento do relógio é realizado através do ajuste dinâmico de sua fase.

A arquitetura foi testada em diferentes FPGAs utilizando-se como base módulos da plataforma *RAPTOR* [9]. Esta plataforma permite a adaptação de diferentes módulos reconfiguráveis que são compostos por FPGAs da *Xilinx*. Os testes foram realizados com os seguintes FPGAs: *Spartan-3A*, *Spartan-6*, *Virtex-4* e *Virtex-5*. Diversas combinações foram analisadas a fim de encontrar possíveis otimizações na comunicação. Como resultado, os autores demonstram uma tabela comparativa entre as combinações usadas destacando os percentuais de otimização após a adição do módulo *chAdapt* na arquitetura. Destacam-se os testes realizados com a comunicação entre dois *Virtex-4* e entre *Spartan-6* e *Virtex-4*. Foi possível notar uma melhora significativa na comunicação após a introdução do *chAdapt* nesses cenários. Os resultados indicam que apesar de não ter sido possível alinhar a largura de barramento para esses testes, o canal implementado oferece um ótimo caminho para a validação dos *links* de comunicação e identificação de melhores parâmetros de transferência de dados.

III. PLATAFORMA GIDEL PROCSTARIII

A implementação do canal de comunicação apresentada neste trabalho foi realizada em uma plataforma *PROCstarIII* (figura 1) da fabricante *Gidel* [10]. Ela é composta por quatro FPGAs *Stratix III* 260E da *Altera* e possui três bancos de memória: um banco de 512MB DDR2 SDRAM e dois bancos DDR2 que suportam até 32GB.

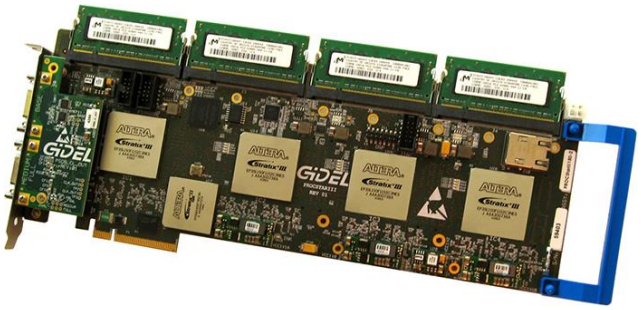

Figura 1. Plataforma *Gidel PROCStarIII*

Apesar da plataforma disponibilizar quatro FPGAs, ela não possibilita que a comunicação entre eles seja realizada utilizando-se os pares diferenciais disponíveis nesses dispositivos. Isto ocorre porque os pinos dos FPGAs utilizados no projeto da plataforma para serem os pinos de comunicação, ou seja, os pinos que estão diretamente conectados entre os FPGAs adjacentes, são pinos de I/O comuns. Outro aspecto, é que não há garantias relacionadas ao *layout* da plataforma. Características como resistência, capacitância, distância das

vias do barramento que interligam os FPGAs não são garantidas, inviabilizando uma comunicação estável. Por estas razões, uma comunicação de alta velocidade pode não ser alcançada. O barramento local é responsável por conectar todos os FPGAs ao *host* através do controlador *PCIex8*. O protocolo de comunicação para este barramento é automaticamente gerado pela ferramenta *ProcWizard* [11] disponibilizada pela *Gidel*. O barramento principal conecta todos os FPGAs entre si e tem uma largura de 40 bits.

A figura 2 apresenta a estrutura interna da plataforma *PROCstarIII.*

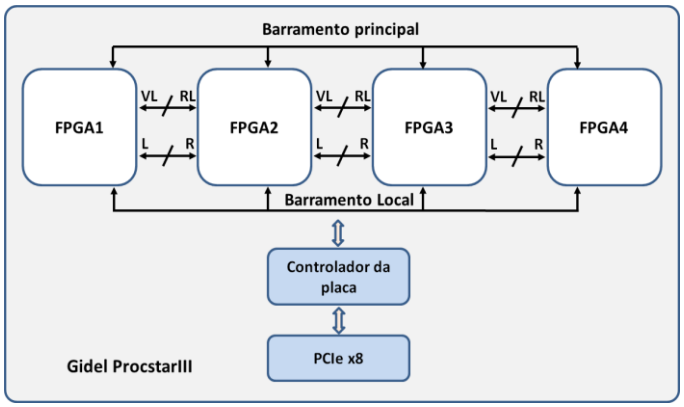

Figura 2. Diagrama da plataforma Gidel PROCStarIII.

Para comunicação de dados, dois barramentos estão disponíveis: o barramento *VL-RL* e o *L-R*. Cada um deles conecta os FPGAs adjacentes disponíveis na plataforma. O *VL-RL* é um barramento de I/O comum e tem largura de 10 bits. Os sinais utilizados nesse barramento possuem uma tensão de 1.8 V. O *L-R* possui largura de 100 bits e é também do tipo I/O comum. O barramento *L-R* além de ser o maior disponível, concentra um único banco de pinos que podem ser utilizados no canal de comunicação. Isto minimiza o impacto referente ao *data skew* uma vez que suas vias estão alinhadas. Diminui-se assim possíveis problemas de atraso de sinal. Devido a essas características, a arquitetura proposta neste trabalho utilizou este barramento para a implementação do canal de comunicação.

A plataforma também dispõe de conectores JTAG que podem ser utilizados para depuração e conta com conectores do tipo PSDB (*PROCStarIII Daughterboard's*) que são interfaces de alta velocidade utilizadas para conectar outras placas disponíveis pela fabricante (camera *links*, interfaces ethernet e etc).

IV. Canal de Comunicação

O sistema de comunicação proposto por esse trabalho é definido como um canal de comunicação paralelo *full-duplex* de 32 bits, síncrono e bi-direcional. Cada controlador é composto por dois módulos principais: um transmissor e um receptor. A comunicação no canal é sincronizada através de um *clock* comum. Esse sistema pode ser então acoplado em cada FPGA para permitir a comunicação de dados entre esses dispositivos na plataforma.

Todo o sistema foi idealizado de modo a facilitar a sua adaptação em diferentes projetos. Para que isso fosse possível, foi utilizada uma *FIFO* de entrada no transmissor e uma *FIFO* de saída no receptor, tornando o design transparente para o usuário. As adaptações para cada aplicação devem ser feitas através de pequenos ajustes nos parâmetros de entrada e saída de cada *FIFO*.

A Figura 3 exibe o canal de comunicação entre dois FPGAs envolvendo os principais sinais do transmissor e do receptor.

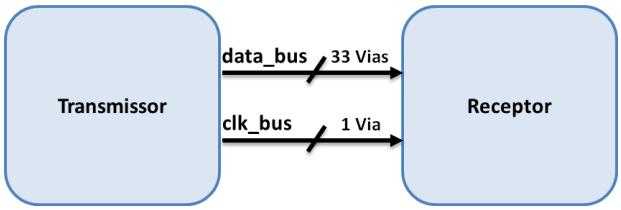

Figura 3. Principais sinais do canal de comunicação.

Uma via única de 33 bits (*data_bus*) foi definida para a transferência de dados entre os dispositivos. Um bit da via *data_bus* é destinado a um sinal *valid*, que indica para o receptor que os dados enviados são válidos e que estes podem ser recebidos e encaminhados para verificação do CRC e, posteriormente, para a *FIFO* de saída. Sendo assim, o canal de comunicação possibilita uma transferência de dados com uma largura de 32 bits entre os FPGAs.

A via de sinalização crítica *clk_bus* é destinada a alimentação do sinal de relógio para o receptor. O sinal de relógio (*clock*) é gerado no transmissor através de uma *PLL* e encaminhado para o receptor por meio de uma via reservada para esta finalidade. Isso garante que ambos, transmissor e receptor, estejam sincronizados permitindo que o canal de comunicação possa funcionar de forma estável. Os sinais de sincronismo e falha de transmissão que também incorporam o barramento serão discutidos nas próximas seções.

A. Transmissor

O módulo de transmissão, denominado *Transmitter*, é responsável por enviar os dados corretamente por meio do barramento. Também é de sua responsabilidade realizar o alinhamento do relógio com a disponibilização dos dados no barramento, aumentando assim o aproveitamento da janela válida, bem como re-transmitir os dados, caso ocorra algum erro em sua transmissão. O transmissor também possui um módulo de geração CRC para os dados enviados. Após a transmissão de um pacote de dados, um *checksum* que valida esse pacote é enviado pelo barramento para ser verificado no receptor. Caso seja encontrado erros durante a verificação, um pedido de retransmissão é realizado.

Para desempenhar essas tarefas, o módulo *Transmitter* foi subdividido em cinco módulos: uma *FIFO* de entrada (*FIFO_Input*), o Circuito de retransmissão de pacotes, o gerador de CRC (*CRC_Gen*), a camada física (*Trans_PHY*) e o Controle do transmissor (*Trans_CTRL*), conforme esquemático apresentado na Figura 4.

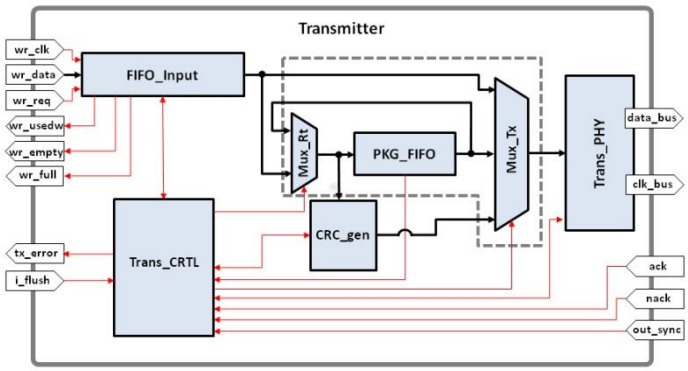

Figura 4. Arquitetura interna do transmissor

A entrada *FIFO_Input* é um módulo disponível no pacote *MegaWizard* que acompanha o software de desenvolvimento *Quartus II* da *Altera*. Ela tem o objetivo de padronizar a interface do módulo de transmissão com a arquitetura na qual ele será inserido, e de receber os dados a serem transferidos. Sua arquitetura permite que a mesma possa trabalhar internamente com uma frequência diferente da arquitetura ao qual o módulo transmissor está acoplado. Comumente, sua frequência interna é mais alta que a utilizada pela arquitetura, e assim, a velocidade com que os dados são retirados da *FIFO* pode ser maior que a velocidade com que eles são inseridos.

Para maior flexibilidade no processo de transmissão, tanto a largura dos dados que serão inseridos quanto a profundidade da *FIFO* são ajustáveis conforme a necessidade do uso por meio de parâmetros de configuração do *Transmitter*. Por padrão, a largura dos dados corresponde a 128 bits e a profundidade da *FIFO* é de 256 palavras.

Alguns sinais são disponibilizados pela *FIFO* de entrada e devem ser utilizados pela arquitetura para a manipulação das escritas de dados na entrada do transmissor. O sinal *wr_clk* corresponde à entrada do relógio da arquitetura que fará as escritas de dados serem encaminhadas para o canal de comunicação. Como citado anteriormente, este relógio é independente do relógio interno. O sinal *wr_req* é utilizado para indicar que os dados na entrada *wr_data* são válidos e que devem ser escritos na *FIFO*. O sinal *wr_usedw* indica para a arquitetura quantas palavras existem internamente na *FIFO*. Por fim, o sinal *wr_empty* indica quando a *FIFO* está vazia e o sinal *wr_full* indica quando a *FIFO* está cheia. Todos esses sinais foram disponibilizados para que o usuário possa monitorar o estado da *FIFO* de entrada e ter total controle de escrita de dados para a transmissão.

A comunicação ocorre no barramento por meio de pacotes. Por padrão, o tamanho do pacote a ser transferido é de oito palavras de 32 bits, uma vez que o barramento só suporta transmissões com essa largura de dados. Neste caso, para formar um pacote são necessários dois dados da *FIFO* de entrada (256 bits). Tomando como base esse exemplo, o transmissor retira os dados da *FIFO* de entrada e os divide internamente. A primeira divisão ocorre na saída da *FIFO* de entrada. Por padrão, a saída é de 64 bits. A segunda é realizada dentro do módulo *Trans_PHY* que divide cada dado de 64 bits em dois de 32 bits. Cada dado do pacote é enviado para o barramento e logo em seguida um *checksum* é enviado para a verificação da integridade do pacote no receptor. Se houver correspondência durante a verificação do *checksum* no receptor, o mesmo envia um sinal *ack* para o transmissor indicando que o pacote foi recebido com sucesso, caso contrário, um sinal *nack* é enviado e uma retransmissão acontece.

O circuito de retransmissão em destaque na Figura 4, é responsável por manter o pacote de dados a ser transmitido em sua estrutura e, em caso de falha, retransmiti-lo da forma mais rápida possível. Para isso, esse circuito conta com uma *FIFO* (*PKG_FIFO*), que armazena o pacote a ser transmitido, e dois multiplexadores (*Mux_Rt* e *Mux_Tx*) que selecionam o caminho correto dos dados a serem enviados. O pacote de dados a ser transmitido, depois de requisitado na *FIFO_Input*, segue para a camada física (*Trans_PHY*) através do componente *Mux_Tx*. Simultaneamente, uma cópia desse pacote é armazenada temporariamente na *FIFO* de re-transmissão (*PKG_FIFO)* por meio do componente *Mux_Rt*. Caso haja algum problema na transmissão, e o sinal *nack* assumir nível lógico alto, os dados previamente armazenados na *PKG_FIFO* são lidos e transmitidos para o receptor via os módulos *Mux_Tx* e *Trans_PHY*, respectivamente. Uma retroalimentação garante que esse pacote possa ser retransmitido outras vezes, até atingir o valor de *MAX_RETX*, previamente estipulado no projeto. Por padrão esse valor é de 10.

A geração do CRC foi implementada utilizando-se uma ferramenta disponível na web chamada *CRC Tool* [12]. Essa ferramenta disponibiliza o código de geração CRC tanto em *Verilog* quanto em *VHDL*. Ela permite que se escolha o polinômio, o tipo de CRC, a largura dos dados e a linguagem alvo. Escolheu-se o polinômio $p = x^{16} + x^{15} + x^2 + 1$, o padrão "CRC-16/USB Data", largura dos dados de 64 bits, e a linguagem *Verilog*.

A função gerada pela ferramenta foi adaptada para dentro do módulo *CRC_gen* que possui uma máquina de estados capaz de gerar o *checksum* de um pacote de dados. Ao passo que os dados do pacote vão sendo transmitidos, eles também vão sendo analisados pelo *CRC_gen* que calcula o CRC do pacote e logo em seguida disponibiliza um *checksum* de 16 bit's para ser transmitido pelo barramento. Cada pacote corresponde a dois dados de entrada na *FIFO* (256 bits) que depois de divididos internamente tornam-se oito dados para a transferência (32 bits cada). Por padrão, para cada pacote é gerado um código CRC para verificação. É possível também modificar a quantidade de dados do pacote para 4, 8, 16 ou 32 palavras.

Através do *MegaWizard* é possível criar um módulo gerador de CRC. Porém, esse módulo é proprietário, o que inviabiliza o seu uso. Dessa forma, o módulo *CRC_gen* foi desenvolvido com as mesmas entradas e saídas do módulo disponibilizado pela *Altera*, de modo a manter a compatibilidade entre as versões. As Figuras 5 e 6 a seguir, ilustram o protocolo de comunicação desenvolvido para a arquitetura proposta.

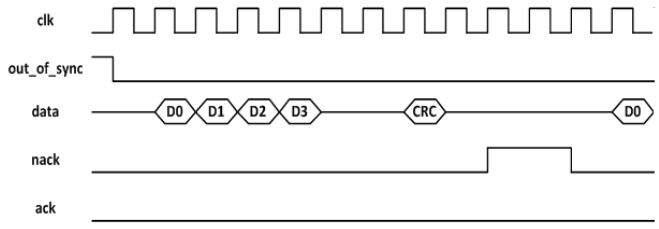

Figura 5. Transmissão com falha de verificação CRC no receptor (nack).

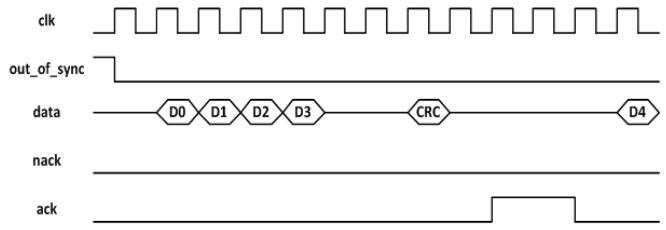

Figura 6. Transmissão realizada com sucesso (ack)

O sinal *out_of_sync* é proveniente do receptor e tem a finalidade de indicar ao transmissor quando seu relógio interno está sincronizado com o sinal *clk_bus*. Portanto, a transmissão só ocorre quando esse sinal está no nível baixo.

O módulo *Trans_PHY* é responsável por gerenciar e encapsular as soluções para o uso dos pinos de I/O do FPGA. O uso de uma camada física permite futuras alterações no padrão de transmissão sem grandes impactos no projeto.

Os dados ao saírem do *Mux_Tx* com 64 bits, são quebrados em dois blocos, formando assim a parte alta (*data_high*) e baixa (*data_low*). Logo em seguida os dados são transmitidos para o barramento juntamente com um sinal *valid*.

Seguiu-se o padrão *DDR* para transmissão [13]. Desta forma, os dados tanto são enviados na borda alta do relógio, quando na baixa, possibilitando uma maior taxa de transferência no canal de comunicação. A Figura 7 apresenta a arquitetura do *Trans_PHY*.

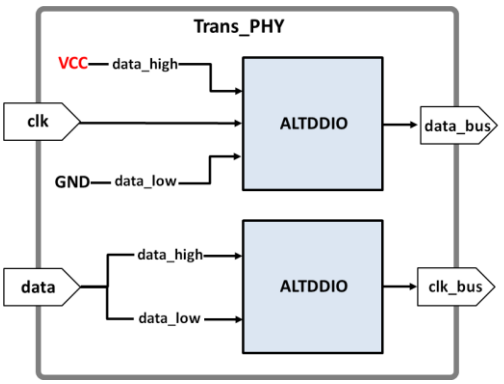

Figura 7. Arquitetura do Trans_PHY

Os módulos *ALTDDIO* também são gerados a partir da ferramenta *MegaWizard*. Eles possuem o propósito de gerar dados no padrão *DDR* e seus registradores internos são implementados em registradores *hard silicon*, ou seja, diretamente na microarquitetura do FPGA, garantindo menores taxas de *skew* na saída dos dados.

O primeiro módulo *ALTDDIO* é utilizado com duas entradas constantes, com *data_high* ligado ao *VCC* (nível lógico alto) e *data_low* ligado ao *GND* (nível lógico baixo). Essa configuração gera um relógio de saída que está em fase com os dados, ou seja, um alinhamento em borda de subida para o relógio com relação à janela válida de dados.

O segundo módulo *ALDDIO* é utilizado para envio dos dados, com suas entradas (*data_high* e *data_low*) recebendo uma porção distinta dos dados: para *data_high*, a parte mais significativa do dado (32 bits) e para *data_low*, a menos significativa (32 bits).

O módulo *Trans_CTRL* encapsula toda a lógica necessária que gera a sinalização de controle para todos os demais módulos. Essa lógica controla a geração do pacote de dados, o *checksum* CRC, a entrada de dados por meio da *FIFO_Input*, e as retransmissões, caso sejam necessárias. Também é de responsabilidade desse módulo indicar para a arquitetura por meio do sinal *tx_error* se houve algum erro na transmissão dos dados. Este é o caso, por exemplo, do número de retransmissões ter sido atingido. Por fim, O sinal *i_flush* indica ao controle do transmissor o encerramento de entrada de dados na *FIFO*. Dessa forma, é possível completar o pacote de dados com zeros, se necessário, mantendo o padrão de transmissão.

B. *Receptor*

O receptor, chamado de *Receiver*, é responsável por receber os dados através do barramento. Todos os fatores de degradação do sinal que atuarem sobre o canal de transmissão, se existirem, terão seus efeitos refletidos no sinal de entrada desse módulo. Dessa forma, a maior parte da complexidade para corrigir esses efeitos está localizada nele.

A arquitetura interna foi organizada com cinco módulos: camada física (*Recv_PHY*), duas *FIFOs* para recebimento dos pacotes (*PKG_BUFFER_1* e *PKG_BUFFER_2*), verificação do CRC (*CRC_chk*), *FIFO* de saída (*FIFO_Output*) e o Controle (*Recv_CTRL*) (Figura 8).

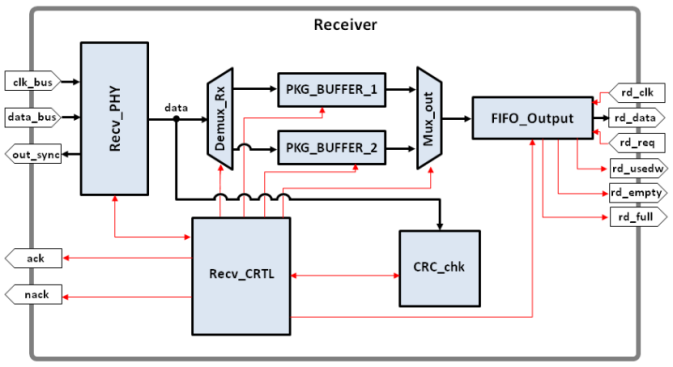

Figura 8. Arquitetura interna do receptor

O módulo *Recv_PHY* é responsável por receber os dados transmitidos e o relógio proveniente do barramento. Todo o

circuito que implementa o sincronismo do padrão *DDR* também está incluído nesse módulo.

Internamente, uma *PLL* foi implementada para receber o relógio do barramento e alimentar todo o circuito do receptor. A *PLL* utilizada é um módulo gerado pela ferramenta *MegaWizard* e é chamada de *ALTPLL*. Ela é do tipo *Source Synchonous*, e garante que a geração do relógio feito por ela esteja em total sincronismo com o relógio recebido pelo barramento. A Figura 9 apresenta a arquitetura do *Recv_PHY*.

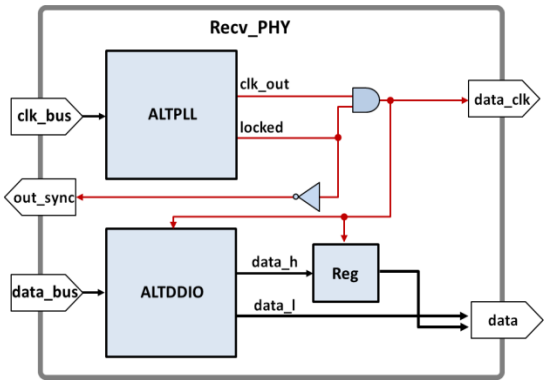

Figura 9. Arquitetura do Recv_PHY

A *PLL* alimenta o circuito do receptor por meio do sinal *data_clk*. Um sinal *locked* é utilizado para indicar quando a *PLL* atinge o sincronismo com o relógio de entrada. Assim, o sinal *out_sync* irá garantir que somente as transações possam ocorrer quando ambos transmissor e receptor estiverem sincronizados. A escolha de se inverter esse sinal para enviar ao barramento se deve à intenção de evitar o *crosstalking*.

A *PLL* possui um *offset* de fase de –90˚, proporcionando uma interface de captura de borda oposta (o dado lançado pela borda de subida no *Transmitter* é capturado pela borda de descida do *Receiver*) e *clock* alinhado com o centro da janela de dados válidos (Figura 10).

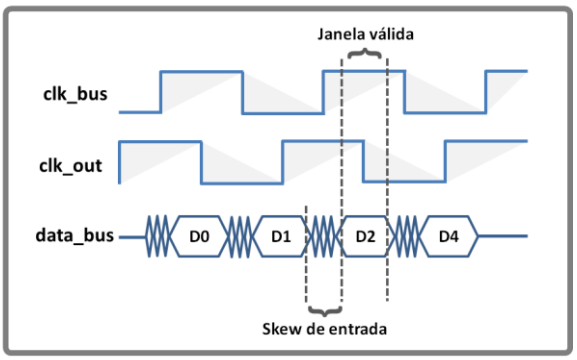

Figura 10. Captura de dados na borda oposta do clock.

A captura com borda oposta proporciona uma inversão na ordem normal de captura dos dados (recebidos pelo módulo *ALTDDIO*). Para ajustar a ordem correta de recebimento dos dados inverte-se assim a saída *data*, ou seja, os dados apresentados em *data_l* devem ser utilizados como a porção alta dos dados que chegam, e os apresentados em *data_h* devem ser os da porção baixa. O registrador associado ao módulo *ALTDDIO* é necessário para causar um atraso na saída *data_h*, uma vez que esse dado é disponibilizado, primeiramente. Assim, a porção alta e baixa do dado recebido segue para a saída *data* de forma sincronizada.

O circuito responsável por receber os pacotes é composto por um demultiplexador (*Demux_Rx*), duas *FIFOs* de pacotes (*PKG_BUFFER_1* e *PKG_BUFFER_2)*, e um multiplexador (*Mux_out*). Cada dado ao ser recebido inicialmente é armazenado em *PKG_BUFFER_1*. Quando um pacote completo é recebido, seu *checksum* é verificado e caso o pacote seja válido, os dados podem seguir para a *FIFO* de saída *FIFO_Output* por meio do *Mux_out*. Enquanto esse processo de transferência de dados entre *PKG_BUFFER_1* e *FIFO_Output* ocorre, o *Demux_Rx* é chaveado para que os próximos dados sejam armazenados em *PKG_BUFFER_2*. O pacote recebido é checado e caso seja válido, os dados são transmitidos para a *FIFO_output* e o *Demux_Rx* é chaveado novamente. É possível assim, obter-se um maior desempenho no recebimento dos dados uma vez que não há a necessidade de esperar uma *PKG_FIFO* ser esvaziada para o recebimento do próximo pacote. As *FIFOs PKG_BUFFER_1* e *PKG_BUFFER_2* têm largura de 64 bits e profundidade de 32 palavras cada.

O módulo *CRC_chk* é similar ao *CRC_gen* do transmissor. Ele possui a mesma função e é responsável por calcular o CRC dos pacotes que estão sendo recebidos. Ao final da transmissão de cada pacote, o *checksum* gerado pelo *CRC_chk* é comparado com o *checksum* recebido no barramento (gerado pelo *CRC_gen*). Nessa estrutura, caso a verificação do CRC falhe, os dados são descartados antes de serem inseridos na *FIFO* de saída e um sinal *nack* é enviado para o transmissor solicitando uma retransmissão. Caso a verificação do CRC seja positiva, os dados são encaminhados para a *FIFO* de saída e um sinal *ack* é enviado para o transmissor solicitando o próximo pacote.

A *FIFO* de saída é exatamente igual à *FIFO* de entrada, localizada no *Transmitter*: possui dois domínios de relógio, profundidade e largura parametrizáveis, e possui funções para ajustar seus parâmetros com base nos dois parâmetros anteriores. Por padrão ela tem uma largura de entrada de 64 bits, largura de saída de 128 bits e profundidade de 256 palavras. O sinal *rd_clk* corresponde à entrada do relógio da arquitetura que fará as leituras de dados. O sinal *rd_req* é utilizado para solicitar à *FIFO* uma leitura. O sinal *rd_usedw* indica para a arquitetura quantas palavras existem internamente. O sinal *rd_empty* indica quando a *FIFO* está vazia e o sinal *rd_full* indica quando a *FIFO* está cheia.

O módulo *Recv_CTRL* encapsula a solução que gera a sinalização de controle para todos os demais módulos, controlando assim a recepção do pacote de dados, a verificação do *checksum* CRC, a escrita do pacote verificado para a *FIFO* de saída, o esvaziamento da *FIFO* de pacotes no caso de erro na verificação do *checksum*, a requisição de uma retransmissão por meio de um *nack*, a indicação de uma transmissão realizada com sucesso por meio de um *ack* e o controle do *demux_rx* e *mux_out*.

V. RESULTADOS

O canal de comunicação foi testado na plataforma *PROCStarIII* conectada via barramento *PCIex8* a um

computador *Intel Xeon* 3 GHz com 4 GB de memória *DDR*. Diferentes frequências de relógio e tamanhos de pacote foram testados. Cada teste foi realizado considerando-se no máximo uma hora de transferência ininterrupta de dados entre os FPGAs da plataforma.

Os dados a serem transferidos foram gerados por uma aplicação em alto nível e enviados do *host* para a plataforma via barramento *PCIex8*. Um controlador de memória específico para as plataformas Gidel (*Gidel PROCMultiPort*) foi implementado para alimentar o transmissor com os dados encaminhados pela aplicação. Para fins de testes e análises de corretude, os dados recebidos pelo receptor foram encaminhados para o *host* para que a aplicação pudesse analisar se os dados enviados correspondiam com os dados recebidos.

O tempo apresentado nos resultados refere-se apenas ao tempo efetivo de transmissão de dados no barramento. Neste caso não se está levando em conta o tempo de envio de dados do *host* para a plataforma, o tempo de transferência de dados entre memória *DDR2* (localizada na plataforma) e FPGA transmissor, e o tempo de envio de dados do FPGA receptor para o *host*.

Foi adotado como base para os testes a quantidade de dados transferidos em uma hora para a configuração de tamanho de pacote de 8 palavras. Com a quantidade de dados transferidos nessa configuração, foi possível refazer os testes variando o tamanho dos pacotes e mensurando o tempo de transferência.

As tabelas I, II, III e IV apresentam os resultados obtidos para as transmissões com frequências de 50, 100, 120 e 150 MHz, respectivamente.

TABELA I. RESULTADOS OBTIDOS COM FREQUÊNCIA DE 50 MHZ.

Tamanho do Pacote	Taxa de Transferência (Gbps)	Quantidade de dados (Gb)	Tempo de transmissão (s)
8	0,7	316,65	3600
16	1,14	316,65	~2200
32	1,68	316,65	~1500

TABELA II. RESULTADOS OBTIDOS COM FREQUÊNCIA DE 100 MHZ.

Tamanho do Pacote	Taxa de Transferência (Gbps)	Quantidade de dados (Gb)	Tempo de transmissão (s)
8	1,4	633,30	3600
16	2,29	633,30	~2200
32	3,37	633,30	~1500

TABELA III. RESULTADOS OBTIDOS COM FREQUÊNCIA DE 120 MHZ.

Tamanho do Pacote	Taxa de Transferência (Gbps)	Quantidade de dados (Gb)	Tempo de transmissão (s)
8	1,58	715,25	3600
16	2,6	715,25	~2200
32	3,8	715,25	~1500

TABELA IV. RESULTADOS OBTIDOS COM FREQUÊNCIA DE 150 MHZ.

Tamanho do Pacote	Taxa de Transferência (Gbps)	Quantidade de dados (Gb)	Tempo de transmissão (s)
8	1,98	894,06	3600
16	3,24	894,06	~2200
32	4,76	894,06	~1500

Todas os testes com as configurações de transmissões apresentadas pelas tabelas foram realizados com sucesso e nenhum erro de transmissão foi detectado.

Para todos os resultados, observa-se um aumento da taxa de transferência de dados quando o tamanho do pacote é incrementado de 8 para 16 palavras, e de 16 para 32 palavras. Isso ocorre por conta dos ciclos de relógio necessários para geração e verificação de CRC no transmissor e receptor. Quanto menor o tamanho do pacote, mais ciclos para geração e verificação de CRC serão necessários, aumentando assim o tempo de transmissão. A Figura 5 exibe um fluxo de transmissão para um tamanho de pacote igual a 4. Nota-se no diagrama que após o envio dos dados alguns ciclos são necessários para o envio do CRC.

De acordo com as tabelas apresentadas, percebe-se que quanto maior a frequência utilizada mais quantidade de dados são transferidos. Os testes realizados com frequência de 150 MHz obtiveram as melhores taxas de transferências. Para as transmissões com tamanhos de pacotes iguais a 8, 16 e 32 foram obtidas taxas de transmissão de 1.98, 3.24, 4.76 Gbps, respectivamente.

Apesar dos resultados obtidos apresentarem taxas de transmissão na ordem de Gbps, é importante ressaltar que o barramento para comunicação utilizado possui 32 bits de largura. Para o caso de uma comunicação realizada com frequência de 100 MHz e pacotes de tamanho 8, cada trilha do barramento realiza transferências numa taxa de 44,8 Mbps.

Outro fator que merece destaque é que a mesma arquitetura desenvolvida e testada na *PROCStarIII* pode apresentar taxas de transmissão diferentes quando implementada em outras plataformas. Características inerentes ao projeto da plataforma podem influenciar diretamente no desempenho da comunicação.

A. Teste de erros na comunicação

Para testar a retransmissão de pacotes, foi implementado na arquitetura um gerador de erros que tem a função de modificar os dados de saída antes deles seguirem para o barramento de comunicação. Esse gerador foi posicionado na saída do *Trans_PHY* e altera os bits dos dados de acordo com uma taxa de inserção de erro configurada em milisegundos. Dessa forma, é possível simular erros na transmissão e testar o funcionamento do canal de comunicação nessas condições. Esse tipo de teste garante a funcionalidade do canal de comunicação caso instabilidades ocorram no decorrer de alguma transferência de dados.

O teste foi realizado modificando-se um bit do barramento de comunicação e foi definida uma taxa de inserção de erro de 4 ms. As transferências foram testadas com uma frequência de 100 MHz e tamanho de pacote igual a 8 palavras. A quantidade

de dados que foram transferidos foi de 633.30Gb, a mesma quantidade utilizada nos testes com 100 MHz em condições normais. A Tabela V apresenta os resultados obtidos.

TABELA V. TESTE DE ERROS COM FREQUÊNCIA DE 100 MHZ E QUANTIDADE DE DADOS TRANSFERIDOS IGUAL A 633.30 GB.

Taxa de inserção de erro (erro/ms)	Quantidade de erros inseridos	Qnt. de dados retransmitidos(Mb)	Tempo de retransmissão (ms)
1/4	849.871	25,93	~144

O número de retransmissões realizadas foi de 849.871, indicando que esse foi o número de pacotes verificados no receptor que não corresponderam com o *checksum* enviado. Apesar da grande quantidade de erros, o montante de dados que necessitou ser retransmitido foi de 25,93 Mb, não causando grandes impactos em relação ao tempo na comunicação do pacote total de dados.

O tempo necessário para realizar essas retransmissões foi de aproximadamente 144 ms. Esse tempo foi obtido a partir da diferença entre o teste sem inserção de erros e o teste com inserção de erros nas mesmas configurações apresentadas.

Foi possível perceber após as análises dos resultados desse teste que a comunicação foi realizada com sucesso apesar dos erros inseridos propositalmente no barramento. Portanto, uma comunicação estável entre os FPGAs pode ser obtida nessas condições.

VI. CONCLUSÃO

A utilização de plataformas multi-FPGAs possibilita o aumento de desempenho na execução de aplicações, uma vez que vários dispositivos reconfiguráveis podem ser utilizados para executar uma mesma tarefa ou trabalharem de forma cooperativa. O uso de vários FPGAs em uma mesma plataforma de modo unificado necessita do estabelecimento de um meio de comunicação entre esses dispositivos.

O trabalho apresentado descreveu um canal de comunicação bi-direcional inter-FPGAs baseado em uma interface *DDR* para transmissão de dados. A arquitetura proposta foi projetada para ser utilizada como uma plataforma reconfigurável, necessitando apenas alguns ajustes de parâmetros do transmissor e do receptor.

Um módulo de detecção de erro baseado no método CRC também foi implementado de maneira a garantir uma comunicação estável. A arquitetura proposta foi testada e validada em uma plataforma *PROCStarIII* da Gidel.

Nesta plataforma em particular foi possível obter taxas de transferências na ordem de Gbps, destacando-se a transmissão realizada com frequência de relógio de 150 MHz, na qual foi possível alcançar taxas de transferências de até 4.76 Gbps. Outro aspecto a ser considerado é que apesar das taxas de transferências obtidas serem satisfatórias, o uso de pinos diferenciais tipo LVDS certamente proporcionariam transferências com taxas mais elevadas. Tomando como referência os dados apresentados em [6], caso fosse possível o uso de 32 trilhas no barramento de comunicação e cada uma com taxas de 625 Mbps, seria possível obter transferências de até 20 Gbps.

Os FPGAs mais modernos possuem transceptores dedicados para transferências de dados. Dispositivos como o Stratix V da Altera, conseguem atingir taxas de transferências de até 28.05 Gbps. Entretanto, é de fundamental importância que o roteamento das trilhas das plataformas que irão acomodar esses dispositivos seja feito cuidadosamente, evitando assim instabilidades na comunicação.

O canal de comunicação proposto obteve um excelente desempenho visto que é direcionado para plataformas que não possuem recursos avançados como interfaces LVDS e transceptores. Como continuidade deste trabalho, o barramento apresentado será implementado na plataforma mais recente da Gidel, a *PROCStarIV*. Essa plataforma possui FPGAs *Stratix* IV da Altera e recursos mais avançados. Espera-se assim que frequências mais altas possam ser alcançadas com o uso do canal de comunicação apresentado.

VII. AGRADECIMENTOS

Os autores gostariam de agradecer ao Centro de Pesquisa da Petrobras (CENPES), pelo suporte técnico em conceitos de sísmica, a CAPES e a FACEPE pelo suporte parcial financeiro ao projeto.

REFERENCES

[1] Hammami, O.; Li, X.; Larzul, L.; Burgun, L., "Automatic design methodologies for MPSOC and prototyping on multi-FPGA Platforms," SoC Design Conference (ISOCC), 2009 International , vol., no., pp.141,146, 22-24 Nov. 2009. doi: 10.1109/SOCDC.2009.5423895.

[2] Melnikova, O.; Hahanova, I.; Mostovaya, K., "Using multi-FPGA systems for ASIC prototyping," CAD Systems in Microelectronics, 2009. CADSM 2009. 10th International Conference - The Experience of RDesigning and Application of , vol., no., pp.237,239, 24-28 Feb. 2009.

[3] LVDS Owners's Manual. Texas Instruments. Disponível em: www.ti.com/lit/ml/snla187/snla187.pdf. Acessado: Julho 2013.

[4] Xilinx Inc. Disponível em : http://www.xilinx.com. Acessado: Julho 2013.

[5] Altera Corporation. Disponível em : http://www.altera.com. Acessado: Julho 2013.

[6] Godbole, P.; Batth, A.; Ramaswamy, N., "High speed multi-lane LVDS inter-FPGA communication link," Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on , vol., no., pp.1,4, 28-29 Dec. 2010.

[7] Inagi, M.; Takashima, Y.; Nakamura, Y., "Globally optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA systems," Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on , vol., no., pp.212,217, Aug. 31 2009-Sept.

[8] PARAMNet-3, C-DAC. Disponível em: http://www.cdac.in/html/htdg/products.aspx. Acessado: Julho 2013.

[9] Raptor Modules. Disponível em: http://www.ks.cit-ec.uni-bielefeld.de/projects/raptor-family/raptor-modules.html. Acessado: Julho 2013.

[10] Gidel PROCStarIII. Disponível em: http://www.gidel.com/PROCStar%20III.htm. Acessado em: Julho 2013.

[11] Gidel's ProcWizard. Disponível em: http://www.gidel.com/procwizard.htm. Acessado em Julho 2013.

[12] CRC tool. Disponível em: http://www.easics.be/webtools/crctool. Acesado: Julho 2013

[13] DDR Interface Design Implementation. Lattice Corporation. Disponível em: http://www.latticesemi.com/lit/docs/generalinfo/memory_ddr_interface_ wp.pdf. Acessado: Julho 2013.