UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EDUARDO GEMELLI EICK

PROJETO DE DIPLOMAÇÃO

DESENVOLVIMENTO DE INTERFACE DE COMUNICAÇÃO PARA INTEGRAÇÃO SW/HW EM PLACA FPGA

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE INTERFACE DE COMUNICAÇÃO PARA INTEGRAÇÃO SW/HW EM PLACA FPGA

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Altamiro Amadeu Susin

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EDUARDO GEMELLI EICK

DESENVOLVIMENTO DE INTERFACE DE COMUNICAÇÃO PARA INTEGRAÇÃO SW/HW EM PLACA FPGA

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Altamiro Amadeu Susin, UFRGS

Doutor pelo INPG – Grenoble, França

Banca Examinadora:

Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble – Grenoble, França

Prof. Dr. Marcelo Soares Lubaszewski, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble – Grenoble, França

Prof. Msc Ronaldo Husemann, UNIVATES
Mestre pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Porto Alegre, dezembro de 2010.

DEDICATÓRIA

Dedico este trabalho aos meus pais, em especial por acreditarem no valor da educação e por me apoiarem sempre em todas as minhas escolhas.

AGRADECIMENTOS

Aos irmãos pelo apoio em todos os momentos difíceis.

Aos colegas pela amizade e pelo apoio.

Aos professores Altamiro Amadeu Susin e Ronaldo Husemann pela orientação e acompanhamento ao longo deste projeto.

Ao projeto FINEP REDE H.264, em especial aos membros do laboratório PRAV-UFRGS, pela oportunidade de desenvolver este projeto.

RESUMO

Este projeto propõe o desenvolvimento de uma interface de comunicação voltada para suportar arquiteturas de co-projeto Software/Hardware (aplicação em software apoiada por uma placa FPGA de hardware) com o objetivo de disponibilizar ao software os recursos do hardware como forma de agilizar a execução de tarefas de elevado custo computacional. O objetivo final é dar suporte a um codificador de vídeo na padrão H.264. É apresentado inicialmente o funcionamento dos protocolos PCI e PCI Express utilizados na comunicação bem como a maneira com que eles foram integrados ao codificador. Por fim, é feito o teste de desempenho e verifica-se que apenas a comunicação por PCI Express é capaz de atender as necessidades do codificador.

Palavras-chaves: FPGA. PCI. PCI Express. DMA. Codificador H.264

ABSTRACT

This project proposes the development of a communication interface to support co-design Software/Hardware architectures (software application supported by a FPGA hardware board) in order to provide the software the resources of the hardware as a way of expediting the execution of high cost computing tasks. The ultimate goal is to support a H.264 video encoder. It is presented at first the operation of PCI and PCI Express protocols used in the communication as well as the way they have been integrated into the encoding chain. Finally, performance tests have been made and it has been proved that only PCI Express communication is able to meet the needs of the encoder.

Keywords: FPGA. PCI. PCI Express. DMA. H.264 Encoder.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CNET: Chipset NetFPGA

CPCI: Chipset PCI

DMA: *Direct Memory Access*

DWORD: *Double Word*

FIFO: First In First Out

FPGA: *Field Programmable Gate Array*

HDTV: *High Definition TeleVision*

PCI: *Peripheral Component Interconnect*

QWORD: *Quad Word*

SDTV: *Standard Definition TeleVision*

SNR: *Signal Noise Ratio*

TLP: *Transaction Layer Packet*

1 INTRODUÇÃO

Vídeos digitais são sequências de fotos estáticas também chamadas de quadros ou *frames*. Essas sequências devem ser obtidas e apresentadas a uma taxa entre 25 e 30 quadros por segundo para que o olho humano possa ter uma boa sensação de movimento (REDIESS, 2006).

Por isso a transmissão de vídeo tende a exigir elevadas bandas do canal de transmissão. Tomando-se, por exemplo, como base um vídeo SDTV (*Standard Definition TeleVision*), com resolução de 720x480 pixels a uma taxa de 30 quadros por segundo, utilizando 3 cores primárias com um byte por cor (24 bits por pixel), chega-se a uma taxa de 248.832.000 bits por segundo. Já para um vídeo HDTV (*High Definition TeleVision*), que tem resolução de 1920x1080 pixels, esse valor sobe para 1.492.992.000 bits por segundo. Esse valor pode ficar ainda maior com a utilização de vídeos em resoluções superiores, como por exemplo vídeos 4k de cinema (4096x2160 pixels) ou UHDV (7680x4320 pixels), entre outros, conforme ilustrado na Figura 1 (SUGAWARA, 2008).

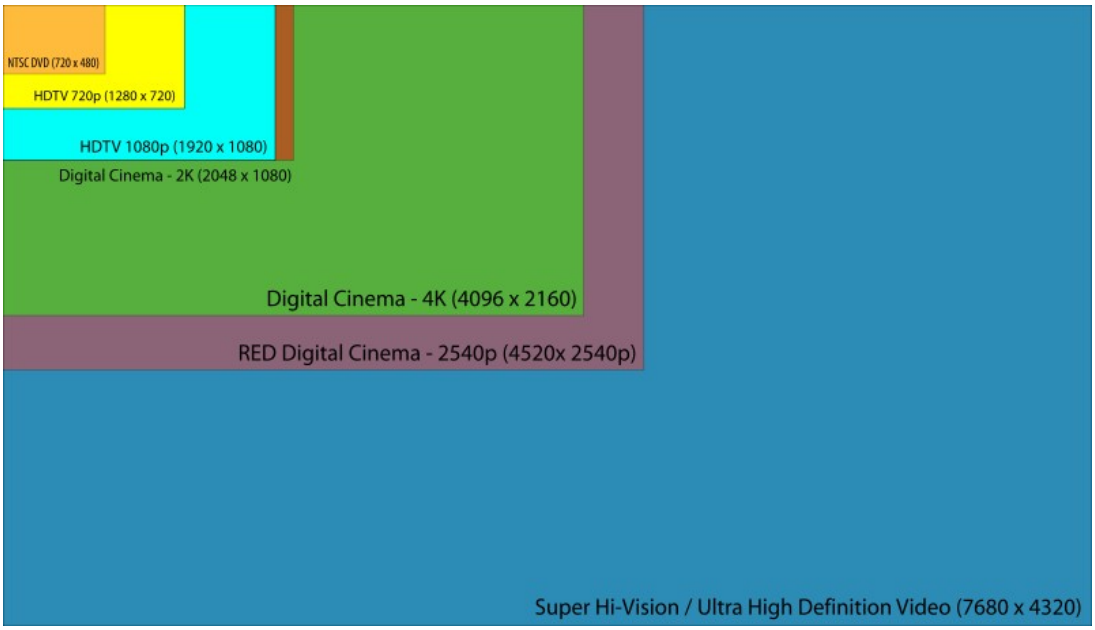

Figura 1: Formatos de Vídeo

A Figura 1 demonstra o crescimento de um problema real: o alto custo de armazenamento e transmissão de altas quantidades de dados. Felizmente imagens carregam muita redundância de informação, sendo que muitas vezes pequenas diferenças entre essas informações não são perceptíveis ao olho humano. São justamente essas propriedades que são exploradas pelos codificadores de vídeo, a fim de que se possa comprimir essas sequências para tamanhos menores, exigindo então menos recursos de armazenamento e de transmissão (REDIESS/2006).

Explorar essas propriedades, no entanto, demandam um significativo custo computacional, principalmente quando se visa obter elevadas taxas de compressão.

Atualmente, um dos mais modernos padrões de codificação de vídeo é definido pelo padrão ITU H.264, também chamado de MPEG-10/AVC (*Advanced Video Coding*). Esse padrão, publicado em 2000, introduziu um conjunto de técnicas avançadas e inovadoras com o objetivo de obter elevadas taxas de compressão de vídeo. De fato este codificador consegue atingir o dobro da taxa de compressão de seu predecessor, o MPEG-2, ao custo, entretanto, de quadruplicar a complexidade computacional (REDIESS, 2006).

Devido a essa crescente complexidade computacional, fica cada vez mais difícil tratar a compressão de vídeo em software. Como forma alternativa existe a possível abordagem do uso de arquiteturas em hardware para apoiar implementações em software. Para tanto é necessária uma eficiente interface de comunicação entre software e hardware, que seja capaz de transmitir dados com elevadas taxas efetivas de dados, de forma a não representar um novo gargalo para a solução.

A solução desenvolvida fez uso de interfaces PCI e PCI Expresse como forma de prover um canal de comunicação de alto desempenho.

2 FUNDAMENTOS TEÓRICOS

2.1 VÍDEOS EM ESTADO BRUTO

Antes de serem comprimidas, as sequências de vídeos são gravadas em um formato que representa um espaço colorimétrico[1] de três componentes:YCbCr. Neste tipo de representação, o termo Y representa o brilho ou "luminância" da imagem, Cb representa a cor (crominância) azul e Cr representa a cor vermelha.

No sistema YCbCr os pesos de cada componente são ajustados de forma a melhor se adequar a sensibilidade humana, mais sensível à luminância do que às crominâncias (RICHARDSON,2003).

2.2 BARRAMENTO PCI

O barramento PCI (*Peripheral Component Interconnect*) é um tipo de barramento que conecta os componentes do computador ao processador por meio de uma interface paralela.

Entre as características do PCI destacam-se a universalidade (independência do processador), leituras e escritas realizadas em modo rajada (taxa de comunicação próxima a 132MB/s a 33Mhz e 32 bits), entre outros (ZELENOVSKY,2006).

Os principais mecanismos de transferência adotados pelo PCI podem ser representados na Figura 2.

[1]Modelo matemático que descreve as cores por meio de conjuntos de números.

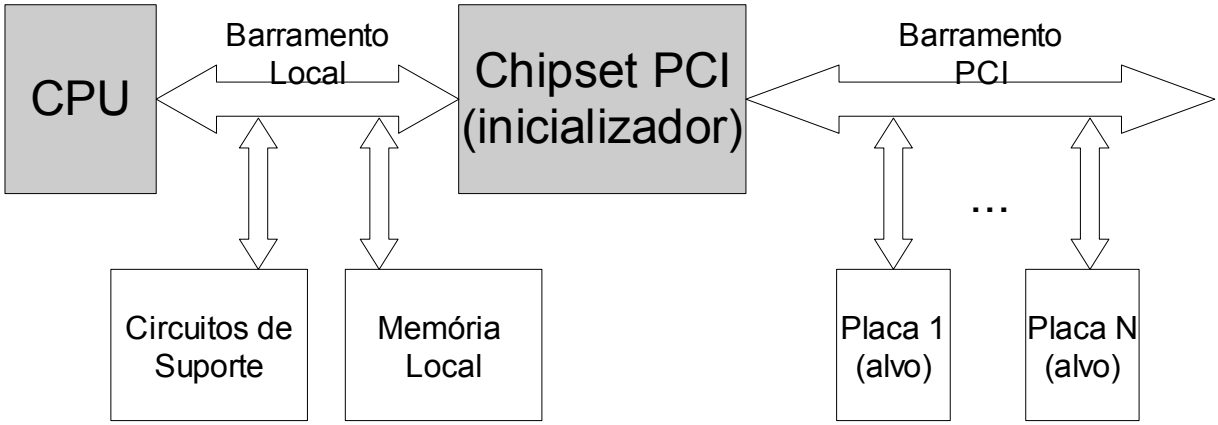

Figura 2: Mecanismos do PCI

O inicializador é o dispositivo que inicia a transação, gerando sinais de endereço e controle para que seja solicitado às placas alvo correspondentes a realização de uma transação. Neste projeto, o alvo será uma placa de hardware responsável pela codificação das sequências de vídeo.

Uma transação PCI se inicia por uma fase de endereços, onde o inicializador transmite um endereço para identificar o dispositivo alvo além do tipo de transação requisitada. Havendo casamento entre o endereço do alvo com o endereço enviado pelo iniciador, o alvo irá indicar que foi selecionado, para estabelecimento da comunicação.

Na fase seguinte, a fase de dados, os dados são transmitidos do iniciador ao alvo, ou vice-versa, dependendo do sentido da transferência. Ao fim do processo, a transação é finalizada pelo inicializador.

2.3 O PCI Express

O PCI Express, comumente abreviado como PCIe, foi desenvolvido para substituir o barramento PCI. Embora seja usual chamá-lo de barramento PCI Express, no sentido estrito da palavra o PCI Express não é um barramento, já que cada dispositivo possui um canal

exclusivo de comunicação com o chipset da placa-mãe, ao contrário do PCI, onde todos os dispositivos compartilham o canal ou barramento (BUDRUK,2003).

Outra diferença essencial em relação ao PCI, é que enquanto que o PCI é uma forma de comunicação paralela, o PCI Express é serial, podendo possuir várias vias. De fato cada periférico pode possuir 1, 2, 4, 8, 16 ou 32 vias (PCIe x1, x2, x4, x8, x16 ou x32). Cada uma dessas vias é roteada na placa-mãe por um *switch* que permite que vários dispositivos utilizem o barramento simultaneamente.

No PCIe versão 1.1 cada via possui uma taxa de dados de 250 Mbytes/s, dessa forma pode-se alcançar taxas de dados de até 8GBytes/s em cada sentido (considerando o PCIe x32).

Além disso, deve-se destacar que o PCIe é um protocolo constituído de 3 camadas principais, a camada Física, a camada de Enlace de Dados e a camada de Transações.

2.3.1 Camada Física (*Physical Layer*)

A Camada Física faz o interfaceamento entre a Camada de Enlace de Dados com a troca de dados propriamente dita no meio físico, podendo ser subdividida em uma camada lógica e uma camada elétrica. A camada lógica é responsável pela formação dos pacotes e a camada elétrica define as características elétricas e temporais para possibilitar recepção/transmissão de dados (entrada e saída).

2.3.2 Camada de Enlace de Dados (*Data Link Layer*)

A Camada de Enlace de Dados funciona como uma camada intermediária entre a Camada Física e a Camada de Transações. Sua principal função é fornecer um mecanismo confiável para a troca de pacotes entre os dois componentes da ligação.

2.3.3 Camada de Transações *(Transaction Layer)*

A Camada de Transações é a camada mais alta da arquitetura PCI Express e sua principal função é receber, armazenar e transmitir os pacotes ou TLPs *(Transaction Layer Packet)* oriundos da aplicação do usuário (PCI Express Base Specification, 2003).

3 CONTEXTO DO PROJETO

Este projeto foi desenvolvido dentro do laboratório PRAV no instituto de Informática da UFRGS, e tem por objetivo dar suporte ao projeto FINEP REDE H.264, desenvolvido nessa universidade. Particularmente este laboratório tem por finalidade desenvolver uma solução paralela para o codificador de vídeo H.264.

Diante da dificuldade de se comprimir sequências de vídeo com características de tempo real, foi proposta, no contexto do projeto FINEP REDE H.264, uma solução inovadora que utiliza uma arquitetura híbrida (aplicação em software apoiada por uma placa de hardware). Nesta proposta, a aplicação em software é responsável pela gerência do codificador, algoritmos sequenciais e montagem das *streams* de saída, enquanto que o hardware será responsável pela implementação dos algoritmos mais complexos do codificador (transformadas, quantização, estimativa de movimento, etc) (HUSEMANN,2010b).

O ambiente de testes é um computador onde está instalada uma placa de hardware contendo uma FPGA conforme pode ser identificado na Figura 3.

Figura 3: Ambiente de Testes

4 PROPOSTA DE TRABALHO

A Figura 4 representa o fluxo dos dados desde o seu estado bruto (no computador), passando pela FPGA para codificação e retornando ao computador hospedeiro com os dados já codificados.

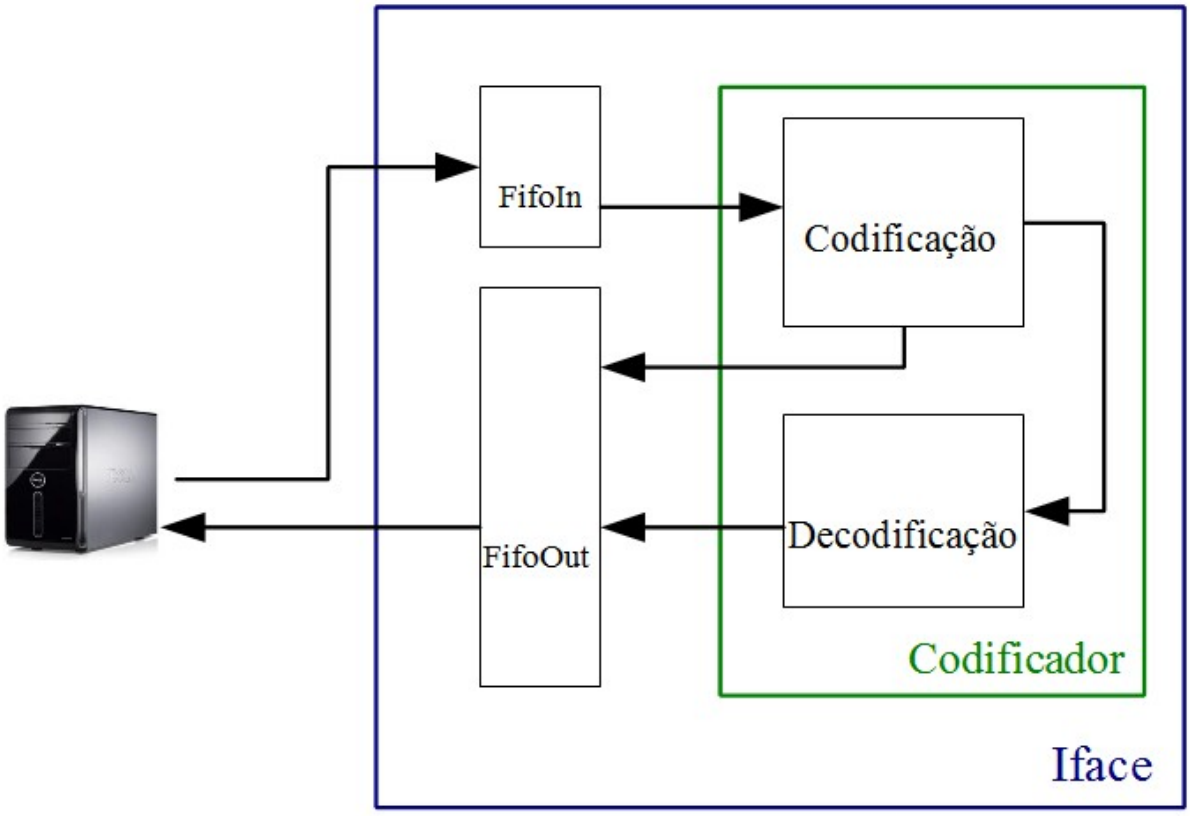

Figura 4: Fluxo dos dados

Os dados chegam na FPGA, são capturados pela interface de comunicação (módulo IFACE) e armazenados em uma FIFO de entrada presente nesse módulo (FifoIn). Em seguida os dados são transferidos para o codificador onde é feita a codificação H.264 dos dados. Após a codificação tem-se dois fluxos de dados distintos. O primeiro segue diretamente para a FIFO de saída (FifoOut) e o segundo passará pelo processo inverso, sendo decodificado a fim

de se remontar o vídeo bruto e só então segue para a FifoOut. Em seguida, o resultado final é transmitido de volta ao computador (HUSEMANN, 2010a).

A etapa final, realizada em software, consiste em comparar o vídeo original em estado bruto com o vídeo reconstruído. Dessa maneira é possível calcular o PSNR (Peak *Signal Noise Ratio*) e verificar se a compressão efetuada possui a qualidade desejada.

4.1 Objetivo do Projeto

Para a análise global do processo, o desempenho da solução proposta em hardware deve ser comparado com uma solução já existente em software. Para tanto utiliza-se o software de referência JSVM (JVT[2] *Scalable Video Model*, codificador de referência H.264/SVC disponibilizado pela *Joint Video Team*) (JSVM Software Manual,2006).

Resultados experimentais no software de referência utilizando amostras de 256 bytes forneceram um tempo de execução de 9 us por amostra, ou aproximadamente 27 Mbytes/s.

Como o objetivo dessa arquitetura híbrida software/hardware é superar o desempenho da solução unicamente em software, o objetivo desse projeto será construir uma interface de comunicação capaz de fornecer uma taxa de dados de no mínimo 27 Mbytes/s.

4.2 Planejamento do Trabalho

Este projeto descreve a implementação do bloco IFACE da figura 4, responsável por integrar software e hardware e permitir assim que os dados fluam entre as duas plataformas de maneira a obter maior rapidez na compressão das sequências de vídeo.

[2] *Joint Video Team*

Como plataformas validação foram utilizadas duas placas de desenvolvimento de FPGA (*Field Programmable Gate Array*) disponibilizadas no laboratório PRAV-UFRGS, local onde este projeto foi realizado.

Uma destas placas é a NetFPGA Versão 2, comercializada pela Digilent, que conta com uma FPGA Virtex2P50 e barramento PCI (NETFPGA, 2010) .

A segunda placa é a XUPV5-LX110T fabricada pela Xilinx e contém uma FPGA Virtex5 e barramento PCIe x1 (XILINX,2007).

O estudo iniciou desenvolvendo-se um módulo de comunicação PCI, que foi sintetizado na placa NetFPGA. Foram criadas duas versões deste módulo: versão com escrita sem DMA e versão com suporte a DMA.

A seguir foi desenvolvido um módulo de comunicação PCI-Express, que foi sintetizado na XUPV5. Neste caso todas as operações de leituras e escritas foram realizadas utilizando-se o recurso de DMA.

Os resultados obtidos com cada um destes módulos, bem como as comparações entre eles está apresentada na seção 7 .

5 COMUNICAÇÃO VIA BARRAMENTO PCI

Para a realização da interface de comunicação via barramento PCI foi utilizada a placa de hardware NetFPGA cujas características técnicas são descritas a seguir.

5.1 Descrição da Placa

A NetFPGA é uma placa de desenvolvimento mais comumente utilizada para aplicações de rede (WATSON, 200X). Ela possui dois dispositivos de FPGAs. Uma FPGA é voltada para fins de comunicação (Xilinx Spartan II xc2s200) a qual é responsável pelo provimento do canal PCI v1 da placa. Além disso é disponibilizada uma Xilinx Virtex2-Pro 50 para as aplicações do usuário. É nessa FPGA que será implementada toda a lógica descrita neste trabalho.

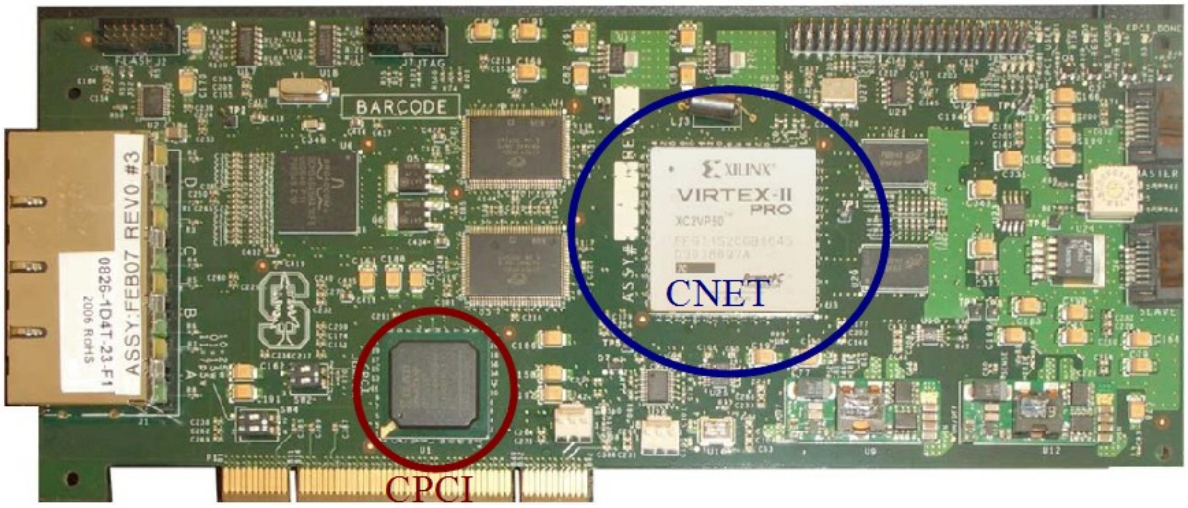

Figura 5: NetFPGA

5.2 Visão Global

A comunicação entre a Virtex2P50, a partir de agora chamada Chipset NetFPGA (CNET) e o computador hospedeiro será indireta, fazendo o uso de uma outra FPGA de

menor capacidade (Spartan2), que foi chamada de Chipset PCI (CPCI). Esta FPGA é pré-programada pelo fabricante para prover o canal PCI.

As figuras 6 e 7 descrevem a constituição da bloco FPGA da figura 4. Uma análise detalhada desta figura é apresentada na Figura 11 que se encontra na página 31.

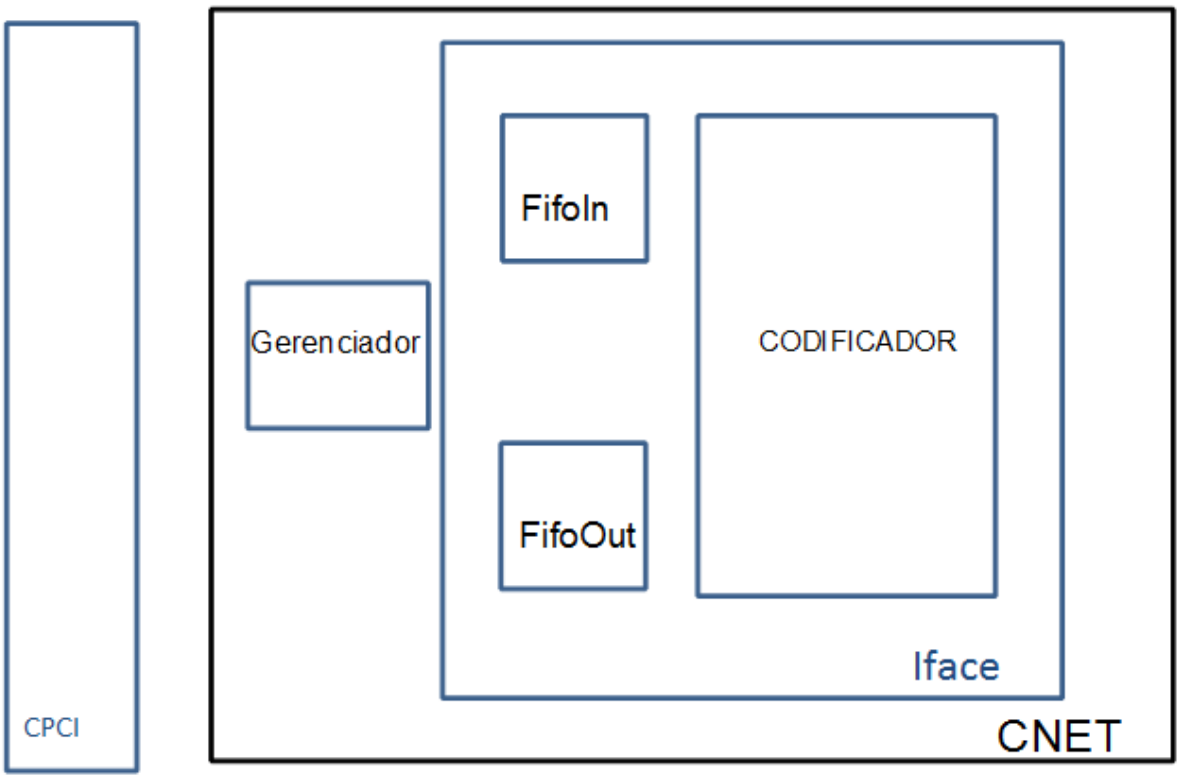

Figura 6: Diagrama de Blocos da CNET

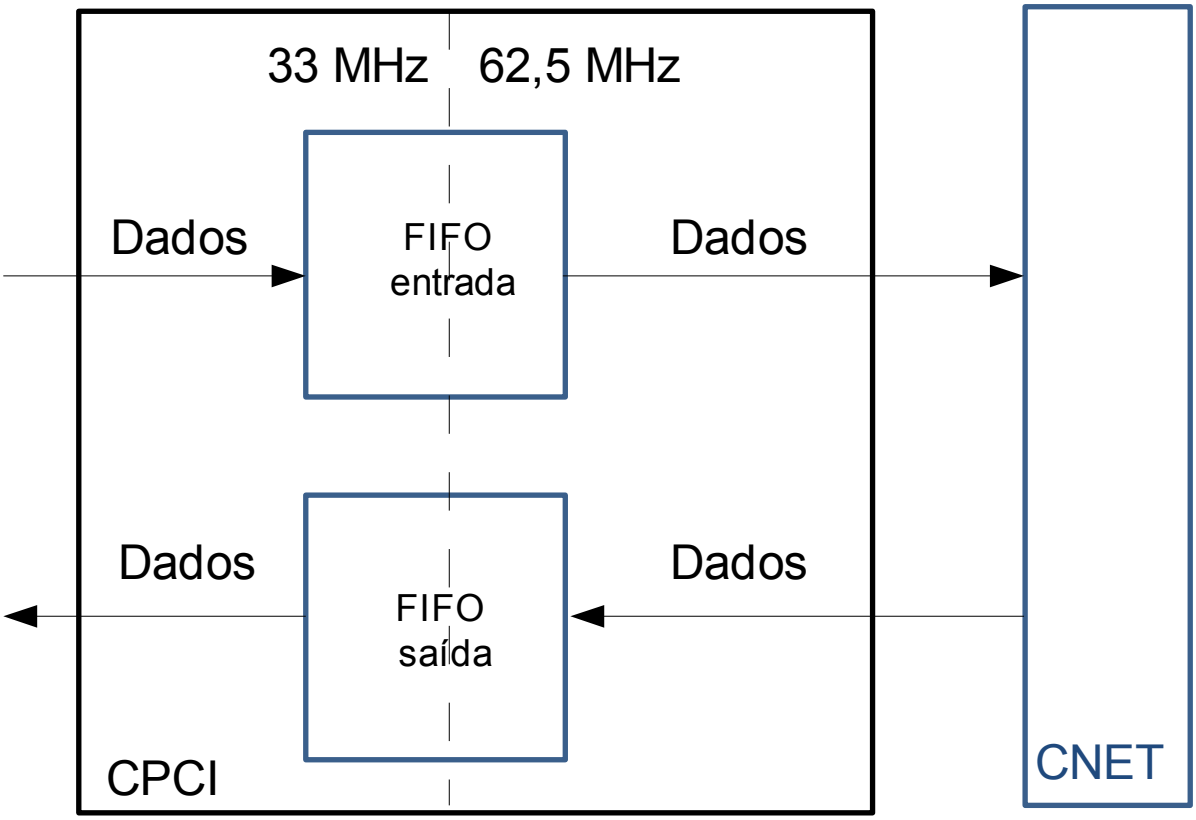

Figura 7: Diagrama de Blocos do CPCI

Sob o comando da aplicação em software, os dados transitam entre o computador hospedeiro e a placa de hardware. Uma vez feita uma solicitação de escrita na placa de hardware, os dados são transmitidos do computador e ficam armazenados nas FIFOS de entrada do CPCI, representadas na figura 7. Quando a CNET, representada na Figura 6, estiver pronta para receber os dados, esta indicará ao CPCI que gerará uma requisição de transferência no sentido C2N[3], dando início ao processo de transferência. Ao final do processo, a FIFO de entrada do CPCI é esvaziada indicando que não existem mais dados a serem transferidos, ficando esses dados armazenados na FifoIn da CNET (Figura 6). Quando o módulo de compressão estiver pronto, os dados lhe são repassados e a FifoIn esvaziada.

[3] No sentido CPCI para CNET.

No caminho inverso, quando os dados estiverem tratados e a FifoOut disponível, esta recebe os dados processados e aguarda a disponibilidade da FIFO de saída do CPCI para enviá-los. Uma vez que a FifoOut contiver dados válidos, a CNET indicará ao CPCI que sua FIFO de saída não está mais vazia. O CPCI irá então gerar uma requisição de transferência no sentido N2C[4], dando início ao processo de transferência. Os dados são então repassados para ao CPCI e a FifoOut é esvaziada.

O CPCI então sinaliza à aplicação em software que ele possui dados em uma FIFO de saída e aguarda a solicitação para a transferência da placa em direção ao computador.

5.3 Desenvolvimento DA APLICAÇÃO PCI

Uma vez compreendido o encadeamento das etapas de transmissão, é necessário detalhar o procedimento adotado.

5.3.1 Escritas e Leituras Programadas

A primeira solução desenvolvida envolveu apenas a escrita de dados e endereços nos barramentos. Neste método, a comunicação se faz diretamente entre a CNET e o computador, sem o uso do CPCI, a um clock de 125 Mhz. Os itens 5.3.1.1 e 5.3.1.2 descrevem esse procedimento.

5.3.1.1 Transferências no Sentido C2N

Os sinais envolvidos neste procedimento são listados na Tabela 1.

[4]No Sentido CNET para CPCI.

Tabela 1: Sinais Envolvidos na transferência C2N

Sinal	Bits	Sentido	Descrição
CLK	1		Relógio do Sistema
CPCI_RD_WR	1	C2N	0: Transferência C2N 1: Transferência N2C
CPCI_REQ	1	C2N	1: Requisição de uma transferência
CPCI_ADDR	32	C2N	Endereço dos Dados
CPCI_DATA	32	Bidirecional	Dados
CPCI_WR_RDY	1	N2C	1: Transferência C2N habilitada

A Figura 8 ilustra o procedimento, toda a vez que a CNET estiver pronta para receber dados, ela habilita o sinal CPCI_WR_RDY, indicando ao CPCI que a transferência pode começar. O CPCI então gera uma requisição (CPCI_REQ = 1) de escrita (CPCI_RD_WR = 0), carregando os sinais CPCI_ADDR e CPCI_DATA com a endereço e os dados, respectivamente. A CNET pode interromper o processo desabilitando CPCI_WR_RDY.

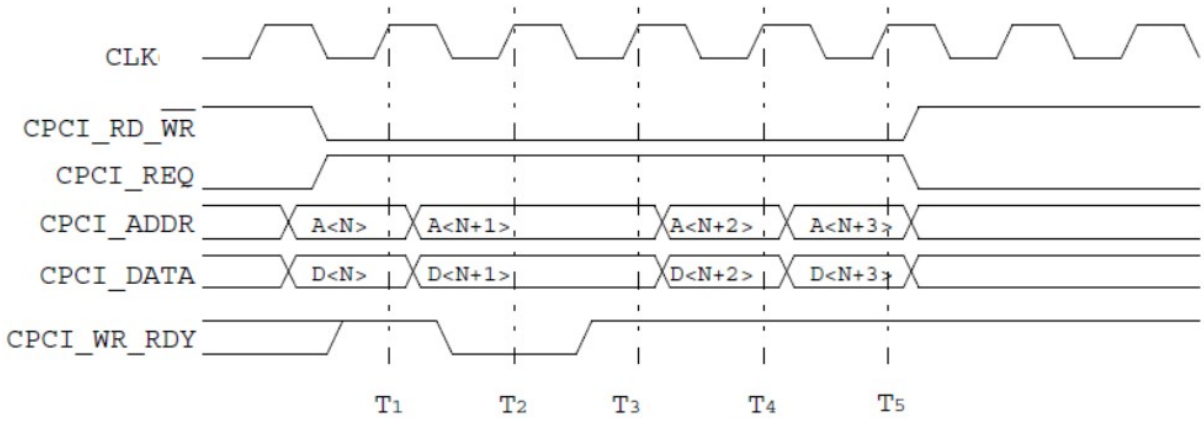

Figura 8: Transferência C2N

5.3.1.2 Transferência no Sentido N2C

Os sinais envolvidos neste procedimento são listados na Tabela 2.

Tabela 2: Sinais Envolvidos na transferência N2C

Sinal	Bits	Sentido	Descrição
CLK	1		Relógio do Sistema
CPCI_RD_WR	1	C2N	0: Transferência C2N 1: Transferência N2C
CPCI_REQ	1	C2N	1: Requisição de uma transferência
CPCI_ADDR	32	C2N	Endereço dos Dados
CPCI_DATA	32	Bidirecional	Dados
CPCI_RD_RDY	1	N2C	1: Transferência N2C habilitada

A Figura 9 ilustra o procedimento inverso, toda a vez que o CPCI desejar fazer uma leitura na CNET, ele deve gerar uma requisição (CPCI_REQ=1) de leitura (CPCI_RD_WR=1), carregando em CPCI_ADDR o endereço onde os dados devem ser lidos. Quando os dados especificados por CPCI_ADDR estiverem prontos para serem lidos pelo CPCI, a CNET carrega CPCI_DATA com os dados válidos e habilita o sinal CPCI_RD_RDY.

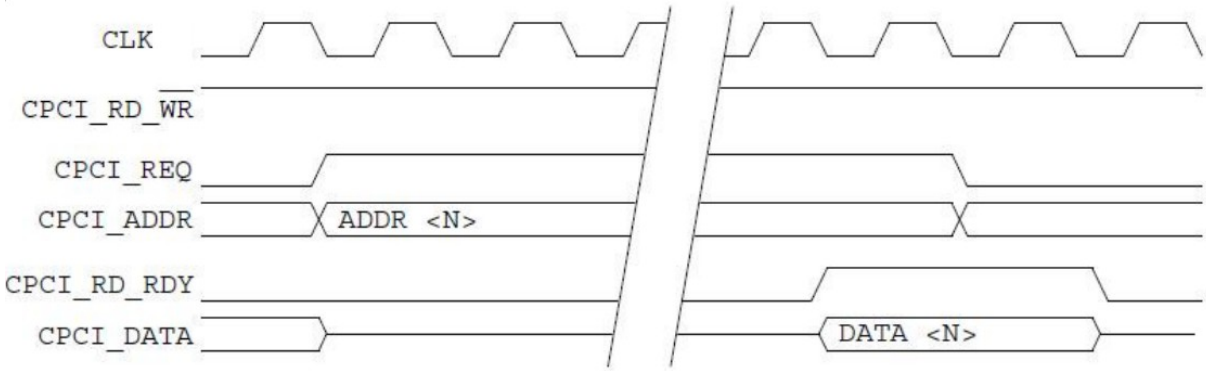

Figura 9: Transferência N2C

5.3.1.3 Resultados Práticos

Para analisar os resultados, foi utilizada a ferramenta ChipScope Pro Analyser® que é capaz de monitorar os sinais internos da FPGA em tempo de execução (CHIPSCOPE PRO, 2010).

A Figura 10 ilustra os resultados obtidos. É possível verificar que entre cada transferência existe um intervalo ocioso de aproximadamente 350 ciclos, intervalo de tempo no qual são transferidos 4 bytes, (ou seja a transferência de uma única palavra de 32 bits). Como a frequência de operação da CNET é de 125 MHz, isso representa um intervalo de tempo de 2,8µs, ou 0,7µs por byte. Em outras palavras, esse método fornece uma taxa de transferência de aproximadamente **1,4 MBytes/s**.

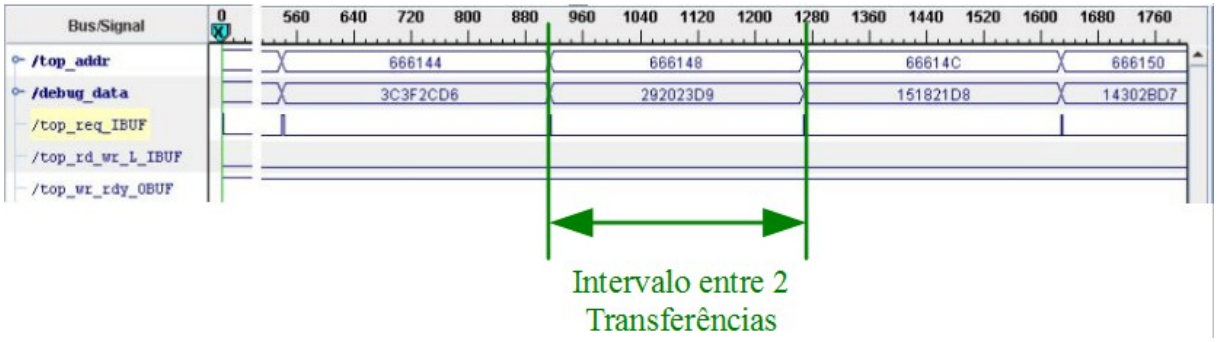

Figura 10: Transferência por leituras e escritas programadas

Percebe-se claramente que essa não é uma solução adequada, já que a taxa de dados obtida é bem inferior aos 27 Mbytes/s desejados, Dessa forma, o ideal é que se faça uma transferência por pacotes de dados, assim o tempo ocioso entre as transferências terá menor impacto no resultado final.

A solução adotada é descrita no item 5.3.2 onde se optou pelo uso do DMA[5] como forma de resolver esse problema.

[5] Direct Memory Access (Acesso Direto a Memória)

5.3.2 Solução PCI Com Suporte de DMA

O DMA (*Direct Memory Access*) é um mecanismo disponível para processadores de alto desempenho que permite que subsistemas de hardware do computador acessem a memória principal para leitura e/ou escrita de forma independente da CPU, ou seja podendo estabelecer um interface direta entre o periférico e a memória. As transferências por DMA são realizadas pelo controlador de DMA, que é um dispositivo de hardware que faz parte da placa mãe dos computadores. Sem o DMA, a CPU teria que intermediar cada dado da fonte até o destino ficando indisponível para processamento durante toda a tarefa de transferência. A situação ficaria ainda mais grave no caso de comunicação com dispositivos periféricos, já que acesso de I/O por um barramento periférico é geralmente mais lento do que a memória RAM.

Uma transferência por DMA evita esse problema possibilitando a cópia direta de blocos de memória de um dispositivo a outro. A CPU irá apenas iniciar a transação informando ao controlador de DMA a origem, destino, tamanho do bloco, entre outras configurações. Em seguida, o controlador de DMA assume o comando da operação realizando as transferências enquanto que o processador está livre para executar suas tarefas. Quando a transferência é finalizada, o controlador irá interromper a CPU devolvendo-lhe o comando.

Por este novo método, a comunicação por PCI fica dividida entre as duas FPGAs: CPCI e CNET. O suporte DMA da NetFPGA fica no CPCI que também é responsável pelo provimento do canal PCI, já a interface de comunicação e o codificador ficam na CNET. O suporte de DMA já é fornecido pela NetFPGA e não precisou sofrer alterações. Já a interface de comunicação com o DMA foi totalmente desenvolvida com o propósito de interligar o CPCI com a aplicação de codificação.

A principal diferença entre este método e o descrito na seção anterior é que não será mais necessário fornecer o endereço para cada um dos dados, bastará indicar o endereço do primeiro dado e o tamanho do pacote a ser transferido.

5.3.2.1 Diagrama de blocos

O diagrama da Figura 11 representa os módulos desenvolvidos bem como os sinais envolvidos para a realização dos processos de leitura e escrita na placa.

O software inicia o procedimento escrevendo o endereço de início do bloco de dados em um registrador de DMA. Com o driver da placa devidamente instalado, o CPCI armazena os dados em sua FIFO de entrada (ver Figura 7). Em seguida, o CPCI assume o controle do processo realizando uma escrita na placa (transferência C2N descrita no item 5.3.2.2). Os dados são então capturados pela CNET por meio de uma máquina de estados representada na Figura 11 pelo bloco *CnetProc*. Esta máquina de estados é o ponto central de todo o processo, fazendo a gestão dos dados que fluem em ambos os sentidos.

Saindo da máquina de estados, os dados entram no módulo *Iface*, este módulo contém as FIFOS onde ficam armazenados os dados antes e depois da codificação. Este módulo possui duas funções principais, fazer a gerência das FIFOS e facilitar a integração com o bloco de comunicação (CNET), o processo *IfaceProc* apenas encaminha os sinais da máquina de estados (CnetProc) para as FIFOS.

Em seguida, os dados são repassados à FifoIn pelo sinal FifoIn_WrData_in o qual é válido apenas quando o sinal FifoIn_WrVld_in estiver habilitado. Conforme a FifoIn for sendo preenchida ela deixa de estar vazia e habilita o sinal FifoIn_Empty_out. Este sinal indica ao codificador (representado na Figura 11 pelo bloco *Codificador*) que ele pode

começar o processo de codificação. A FifoIn possui ainda o sinal FifoIn_Full_out que é habilitado quando a FifoIn estiver cheia, esse sinal é repassado à máquina de estados (sinal Iface_Full_out) que paralisa a transferência até que a FifoIn possua espaço para armazenar os dados.

Uma vez que o codificador estiver apto a iniciar a codificação (FifoIn não está mais vazia), ele requisita uma leitura dos dados da FifoIn pelo sinal FifoIn_RdReq_in e recebe esses dados dois ciclos depois no sinal FifoIn_RdData_out, esse processo se repete até a FifoIn se esvaziar novamente.

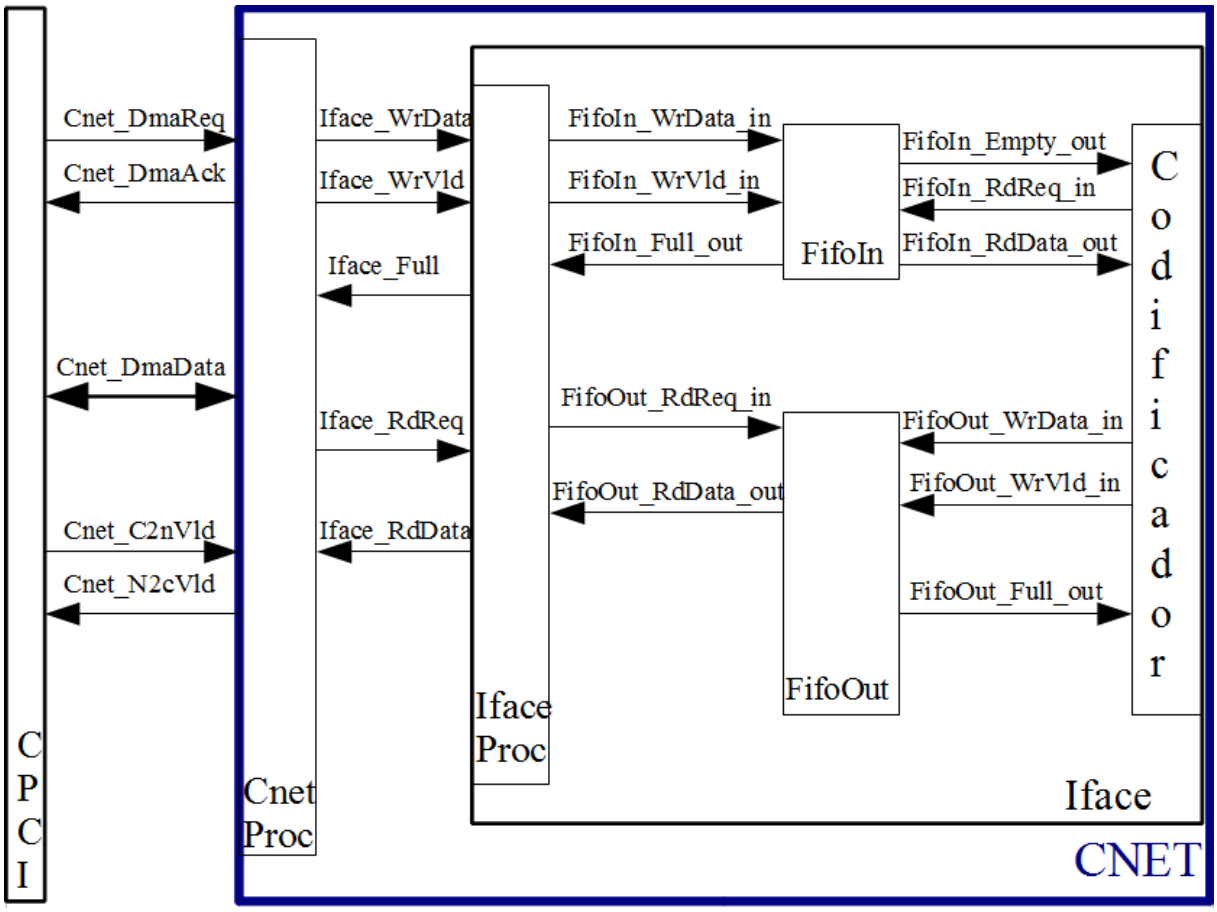

Figura 11: Diagrama de Blocos

No sentido inverso, o codificador preenche a FifoOut passando os dados codificados no sinal FifoOut_WrData_in habilitando FifoOut_WrVld_in a cada novo dado. O codificador deve paralisar a escrita na FifoOut quando ela estiver cheia (sinal FifoOut_Full_habilitado). Os dados codificados ficam armazenados na FifoOut aguardando o início da transferência N2C .

Durante a transferência N2C a FifoOut encaminha sua saída de dados (sinal FifoOut_RdData_out) à máquina de estados (sinal Iface_RdData) mediante a habilitação do sinal FifoOut_RdReq_in, gerado pela sinal Iface_RdReq da máquina de estados.

Ao contrário do que ocorre na transferência C2N, quando a aplicação em software requisita que o CPCI faça a transferência, na transferência N2C é a própria CNET que habilita o início do processo informando que possui dados na FifoOut. Quando a FifoOut possui dados codificados o sinal FifoOut_Empty_out está desabilitado, já que a FifoOut não está mais vazia. Por consequência o sinal Iface_Empty também estará desabilitado. A máquina de estados usa esse sinal para informar ao CPCI que existem dados codificados aguardando transferência. Quando o barramento de dados (sinal Cnet_DmaData) estiver disponível o CPCI dá início a transferência N2C.

Finalmente, os dados codificados ficarão armazenados no CPCI até que a aplicação em software requisite uma leitura.

Os itens 5.3.2.2 , 5.3.2.3 e 5.3.2.4 seguintes fornecem maiores detalhes ao funcionamento exposto neste item.

Observe ainda na Figura 11 que a FifoIn e a FifoOut foram representadas em tamanhos diferentes, isso porque de fato a FifoOut tem o dobro do tamanho da FifoIn, já que após a codificação dos dados o codificador também os reconstrói, devolvendo ambas as sequências para o aplicativo em software.

5.3.2.2 Transferência no Sentido C2N

A Tabela 3 apresenta os principais sinais intervenientes na operação.

Tabela 3: Sinais intervenientes na transação C2N do PCI DMA

Nome	Sentido	Bits	Descrição
Cnet_CpciClk_in	C2N	1	Relógio do sistema
Cnet_DmaReq_in	C2N	2	CPCI requisita da NETFPGA uma operação. De escrita (10), leitura (11), status (01).
Cnet_DmaAck_out	N2C	2	Confirma que a CNET recebeu o dma_op_code_req do CPCI.
Cnet_C2nVld_in	C2N	1	Especifica que o dato presente em dma_data é válido.
Cnet_DmaData_inout	bidirecional	32	O CPCI passa o tamanho do pacote no primeiro ciclo válido e o pacote de dados nos ciclos válidos subsequentes.
Cnet_N2cNearlyFull_out	N2C	1	Indica que a FifoIn está quase cheia.

A transferência de um pacote de dados do CPCI à CNET pode ser representada pela Figura 12. A transferência é iniciada pela aplicação em software que requisita do CPCI (mestre de DMA) um transferência C2N, o CPCI responde a essa solicitação enviando a CNET o sinal Cnet_DmaReq_in[1:0] = 10. Na próxima borda de subida do clock, a CNET deve responder ao CPCI atribuindo ao sinal Cnet_DmaAck_out[1:0] o mesmo valor que lhe passado por Cnet_DmaReq_in, confirmando assim o recebimento da requisição.

Neste momento, o CPCI já está habilitado a utilizar o barramento de dados (Cnet_DmaData_inout[31:0]). A existência ou não de informações válidas no barramento de dados é verificada pelo sinal Cnet_C2nVld_in, sempre que ele estiver habilitado na borda de subida do clock deve-se interpretar que o barramento contém informações válidas, do contrário, trata-se de um dado espúrio ou de um dado que permaneceu no barramento

aguardando ser capturado, e portanto deve ser descartado para evitar que ele seja avaliado mais de uma vez.

A transferência propriamente dita é composta de duas etapas principais. O primeiro valor válido no barramento de dados é o tamanho do pacote a ser enviado, este valor permite saber quando a transferência termina. Em seguida, um a um, são transferidos os dados acompanhados da habilitação do sinal Cnet_C2nVld_in. Os dados recebido são então armazenados em uma FIFO de entrada (FifoIn)

A CNET deve possuir além disso o sinal Cnet_N2cNearlyFull_out, sinal no sentido N2C que informa ao CPCI que a FifoOut da CNET está quase cheia, neste caso, o CPCI responde desabilitando o sinal Cnet_C2nVld_in e paralisando a transferência. Conforme o codificador for utilizando os dados da FifoIn ela é esvaziada novamente e o sinal de FIFO quase cheia é desabilitado novamente permitindo a continuação da transferência.

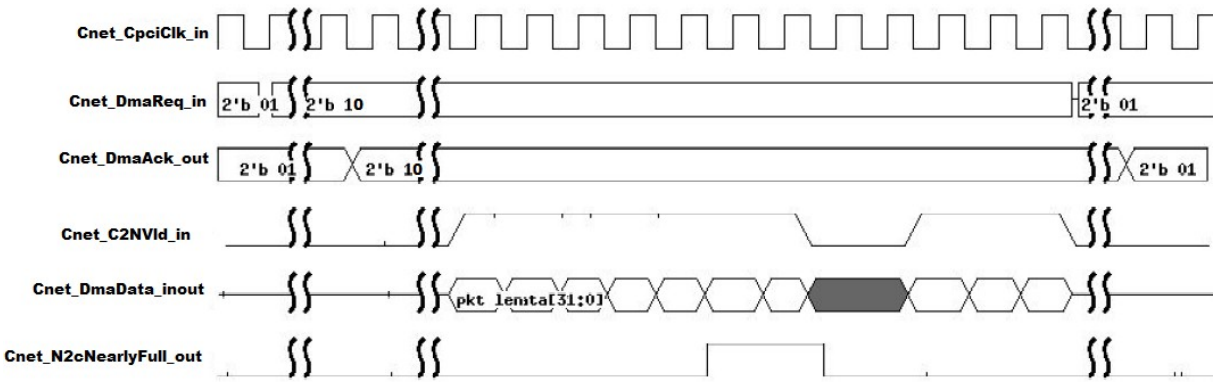

Figura 12: Transferência PCI C2N por DMA

5.3.2.3 Transferência no Sentido N2C

A Tabela 4 apresenta os principais sinais intervenientes na operação.

Tabela 4: Sinais intervenientes na transação N2C do PCI DMA

Nome	Sentido	Bits	Descrição
CLK	C2N	1	Relógio do Sistema
Cnet_DmaReq_in	C2N	2	CPCI requisita da NETFPGA uma operação. De escrita (10), leitura (11), status (01).
Cnet_DmaAck_out	N2C	2	Confirma que a CNET recebeu o dma_op_code_req do CPCI.
Cnet_N2cVld_out	N2C	1	Especifica que o dato presente em dma_data é válido.
Cnet_DmaData_inout	Bidirecional	32	A CNET passa o tamanho do pacote no primeiro ciclo valido e o pacote de dados nos ciclos validos subsequentes.

De forma semelhante ao exposto no item anterior, a transferência de uma pacote de dados da CNET ao CPCI pode ser representada pela Figura 13. A transferência se inicia com a requisição do CPCI por uma operação de transferência no sentido N2C: Cnet_DmaReq_in[1:0] = 11. Na próxima borda de subida do clock, a CNET deve responder ao CPCI atribuindo ao sinal Cnet_DmaAck_out[1:0] o mesmo valor que lhe passado por Cnet_DmaReq_in, confirmando assim o recebimento da requisição.

Neste momento, a CNET já está habilitada a utilizar o barramento de dados (Cnet_DmaData_inout[31:0]). De forma análoga ao item anterior, a cada novo dado válido, deve-se habilitar o sinal Cnet_N2cVld_out. Pode-se destacar duas etapas principais durante o processo de transferência. Num primeiro momento, a CNET deve informar o tamanho do pacote a ser transferido, carregando o sinal Cnet_DmaData_inout com o tamanho deste pacote e habilitando o sinal Cnet_N2cVld_out no mesmo ciclo. Em seguida, um a um, a CNET deve carregar o sinal Cnet_DmaData_inout com os dados, validando-os a cada vez (durante um único ciclo apenas).

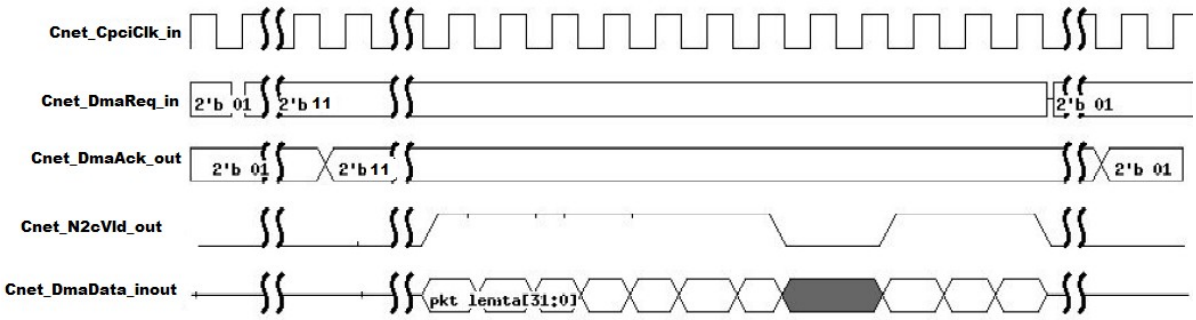

Figura 13: Transferência PCI N2C por DMA

5.3.2.4 Máquina de Estados Implementada

O processo *CnetProc* da Figura 11 é melhor detalhado na Figura 14.

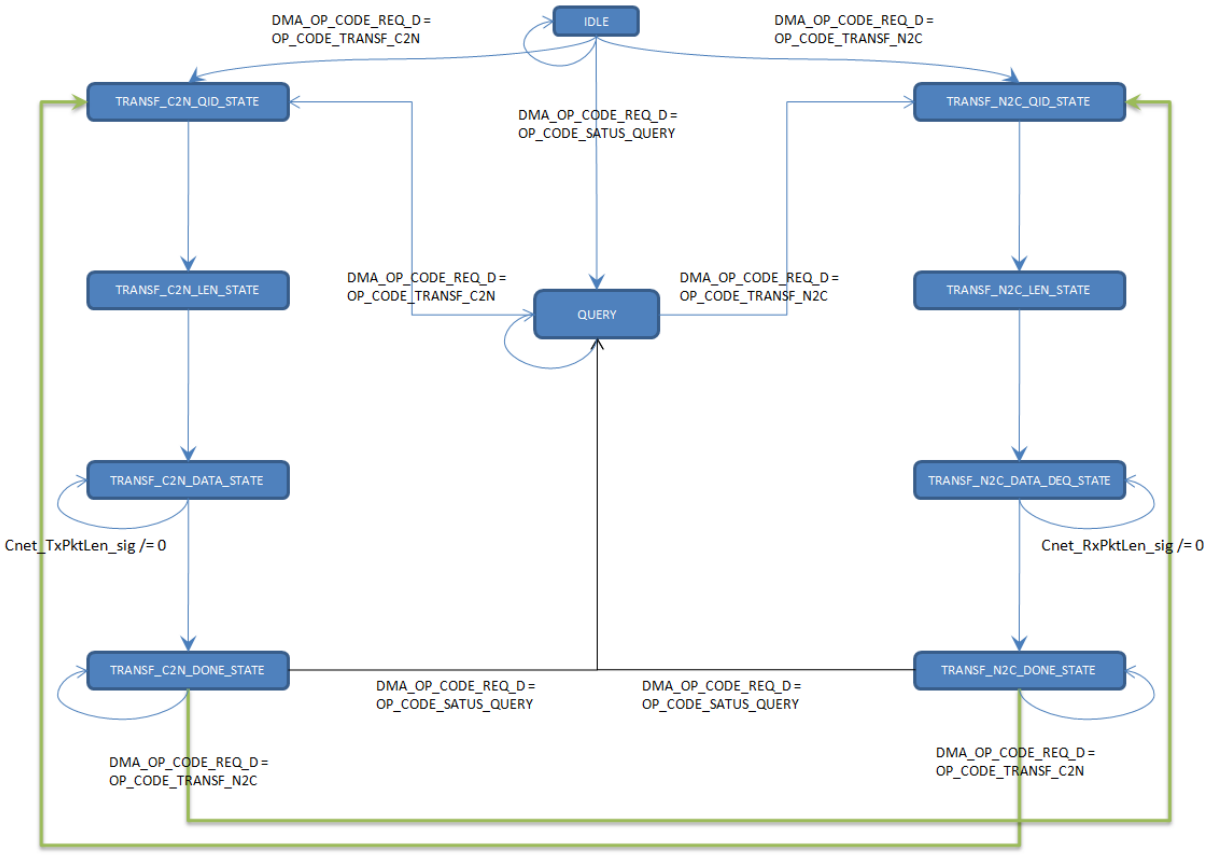

Figura 14: Máquina de Estados da comunicação PCI

Quando a placa é energizada ou o botão de reset é pressionado, o processo realiza as inicializações das variáveis necessárias e assume o estado de *IDLE*, estado ocioso. Neste ponto, avalia-se Cnet_DmaReq_in e decide-se o próximo estado.

QUERY: Estado de Decisão. Neste estado, a máquina de estados informa ao CPCI sobre a existência ou não de dados na FifoOut da CNET prontos para serem lidos. A máquina ficará travada neste estado avaliando o sinal Cnet_DmaReq_in para decidir sobre seu próximo estado, conforme detalhado na Tabela 5

Tabela 5: Valores de Cnet_DmaReq_in

Cnet_DmaReq_in	Próximo Estado
1	QUERY
10	TRANSF_C2N
11	TRANSF_N2C

Destaca-se aqui que o CPCI dá prioridade às transferências no sentido N2C sobre as C2N, ou seja, sempre que houver dados já codificados eles são transferidos ao CPCI, dessa forma quando o software requisitar uma leitura da placa os dados já estarão prontos no CPCI, agilizando assim todo o processo. Por essa razão, as transferências entre as duas FPGAs não ocorrerão na mesma ordem que entre o PC e o CPCI, sem prejuízo à coerência dos dados.

- TRANSF_C2N: Inicio da transferência C2N. Habilita o sinal Cnet_DmaData_inout para receber os dados no sentido C2N e confirma o recebimento de Cnet_DmaReq_in (Cnet_DmaAck_out = 10). Permanece neste estado por apenas um ciclo e segue incondicionalmente para o estado TRANSF_C2N_LEN_STATE.
- TRANSF_C2N_LEN_STATE: Recebe a primeira informação, tamanho do pacote. Avalia o sinal Cnet_C2NVld_in, se ele estiver habilitado, significa que Cnet_DmaData_inout contém o tamanho do pacote a ser transferido. Este dado

é armazenado em um registrador (tx_pkt_len) para o controle da máquina de estados. Ao mesmo tempo, este valor é encaminhado para a FifoIn juntamente com Cnet_C2NVld_in para preencher a primeira posição da Fifo. Se tamanho do pacote for maior do que zero, o próximo estado será TRANSF_C2N_DATA_STATE senão o próximo estado será TRANSF_C2N_DONE_STATE.

- TRANSF_C2N_DATA_STATE: Transferência efetiva dos dados. Neste ponto, a cada novo dado válido, Cnet_DmaData_inout é encaminhado à FifoIn e o tamanho do pacote é decrementado. Quando o tamanho do pacote atinge zero, a Fifo está preenchida e a máquina passa para o estado TRANSF_C2N_DONE_STATE para finalizar a transferência.
- TRANSF_C2N_DONE_STATE: Fim da transferência C2N. Decide o próximo estado avaliando Cnet_DmaReq_in conforme Tabela 5.
- TRANSF_N2C: Início da transferência N2C. Habilita o sinal Cnet_DmaData_inout para receber os dados no sentido N2C e confirma o recebimento de Cnet_DmaReq_in (Cnet_DmaAck_out = 11). Permanece neste estado por apenas um clico e segue incondicionalmente para o estado TRANSF_N2C_LEN_STATE.
- TRANSF_N2C_LEN_STATE: Efetua a transmissão do tamanho do pacote para o CPCI. O primeiro dado da FifoOut (Iface_RdData_out) é o tamanho do pacote e é carregado no sinal Cnet_DmaData_inout e validado no mesmo ciclo pela habilitação do sinal Cnet_N2cVld_out. O tamanho do pacote é armazenado em um registrador (rx_pkt_len) para o controle da máquina de estados. O próximo estado será TRANSF_C2N_DATA_DEQ_STATE.

- TRANSF_C2N_DATA_DEQ_STATE: Transferência efetiva dos dados. Neste estado, cada dado da FifoOut é enviado ao CPCI pelo sinal Cnet_DmaData_inout sendo validado a cada ciclo ao mesmo tempo em que se decrementa o tamanho do pacote. Quando o tamanho do pacote atinge zero, a FifoOut está esvaziada e os dados transferidos ao CPCI. O próximo estado será TRANSF_N2C_DONE_STATE.
- TRANSF_N2C_DONE_STATE: Fim da transferência N2C. Decide o próximo estado avaliando Cnet_DmaReq_in conforme Tabela 5.

5.3.2.5 Resultados Práticos

A implementação dos procedimentos descritos anteriormente fornece o desempenho ilustrado na Figura 15. Na transferência entre as duas FPGAs, é possível verificar uma velocidade maior na transferência N2C. Isso ocorre pois durante grande parte da transferência C2N não há transferência efetiva de dados. Isso se explica pois a frequência de operação do barramento PCI (33 Mhz) é inferior à frequência do CPCI (62,5 Mhz), dessa forma, a CNET esvazia rapidamente a FIFO de entrada do CPCI e deve aguardar que novos dados estejam disponíveis. Isso no entanto não ocorre no sentido inverso pois a FIFO de saída do CPCI foi concebida para ser capaz de armazenar todos os dados do pacote

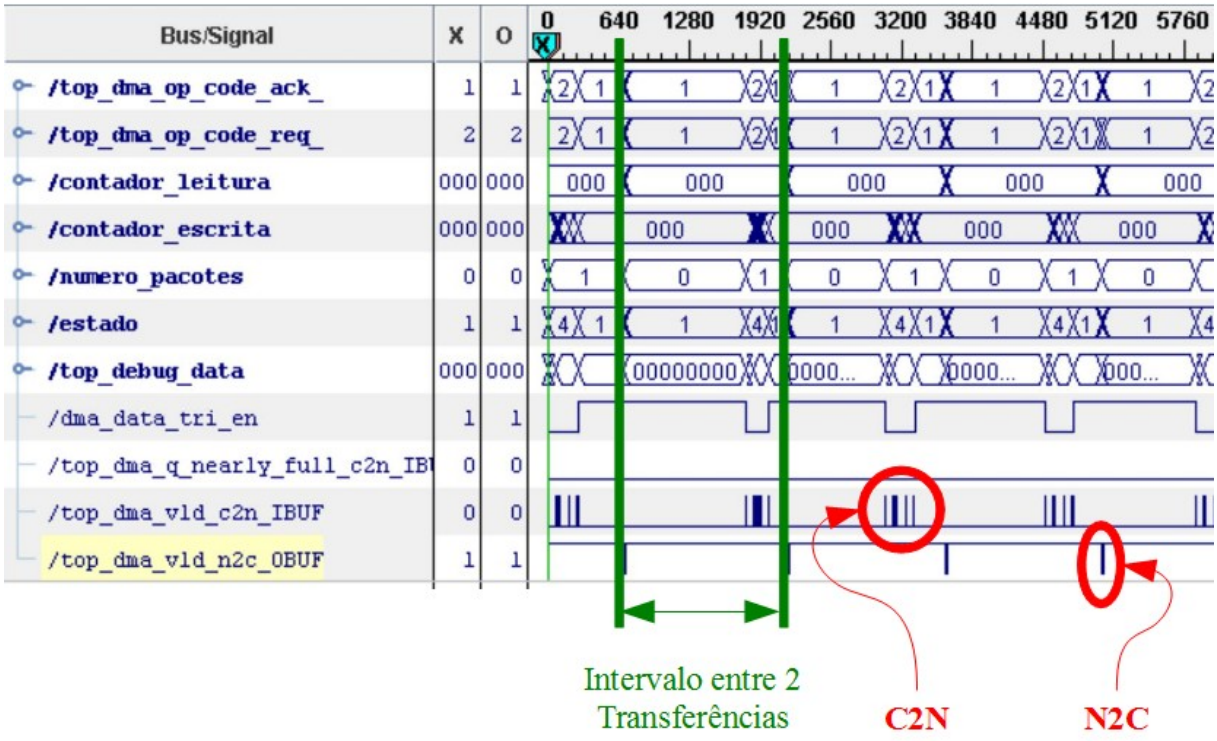

Figura 15: Múltiplas Transferências C2N e N2C

Na Figura 15 foram transferidos múltiplos pacotes de 256 bytes (tamanho de uma matriz de 16x16 pixels) e foi possível verificar que o tempo entre cada uma da transferências é de aproximadamente 24 us (1500 ciclos de 62,5 Mhz), período em que são transferidos 256 bytes, totalizando uma taxa de tratamento de **10,2 Mbytes/s**.

Pôde-se observar que o módulo de comunicação via barramento PCI constitui um gargalo para a codificação, por isso buscou-se uma nova maneira de estabelecer a comunicação entre o software e o hardware.

6 COMUNICAÇÃO VIA PCI EXPRESS

A segunda alternativa de projeto foi utilizar a placa de hardware XUPV5-LX110T da Xilinx que contém uma FPGA Virtex5, essa placa possui PCI Express que será utilizado para o desenvolvimento da aplicação.

6.1 Descrição da Placa

A XUPV5-LX110T *(Xilinx University Program XUPV5-LX110T Development System)* é uma placa de uso geral que conta as seguintes características (XILINX, 2007):

- FPGA Xilinx Virtex5 XC5VLX110T;
- Duas PROMs Xilinx XCF32P para a configuração da placa;
- Um controlador de cartão Compact Flash Xilinx SystemACE;
- Conexão PCI Express x1
- Controlador USB;
- Diversas portas de I/O.

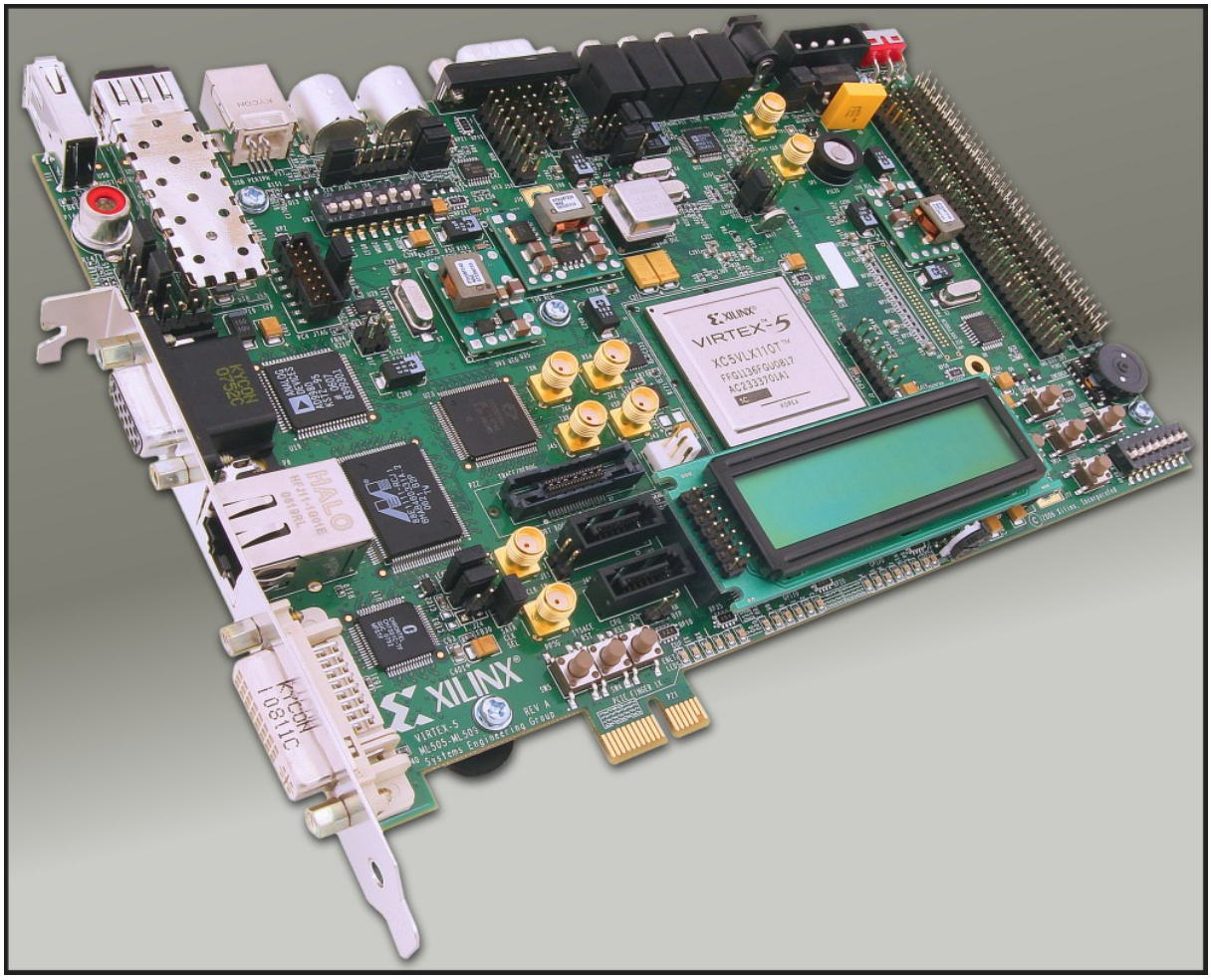

Figura 16: XUPV5-LX110T

6.2 Visão Global

Conforme discutido no item 2.3 a comunicação PCI Express é composta das camadas Física, de enlace de Dados e de Transação, conforme representadas da Figura 17.

A camada física (Xilinx CORE) é a camada responsável pelo provimento do canal PCI Express e pelo suporte do DMA. A camada de Enlace de Dados (Xilinx End Point) faz a ligação entre a camada física e a camada de transações. Tanto a camada Física quanto a de Enlace de Dados foram geradas usando o Xilinx CoreGen® (XILINX, 200X), ferramenta da Xilinx para gerar cores otimizados para suas FPGAs. A camada de transação é a responsável

pela gerência dos pacotes enviados e recebidos, é a ela que deverá ser conectada a interface do codificador.

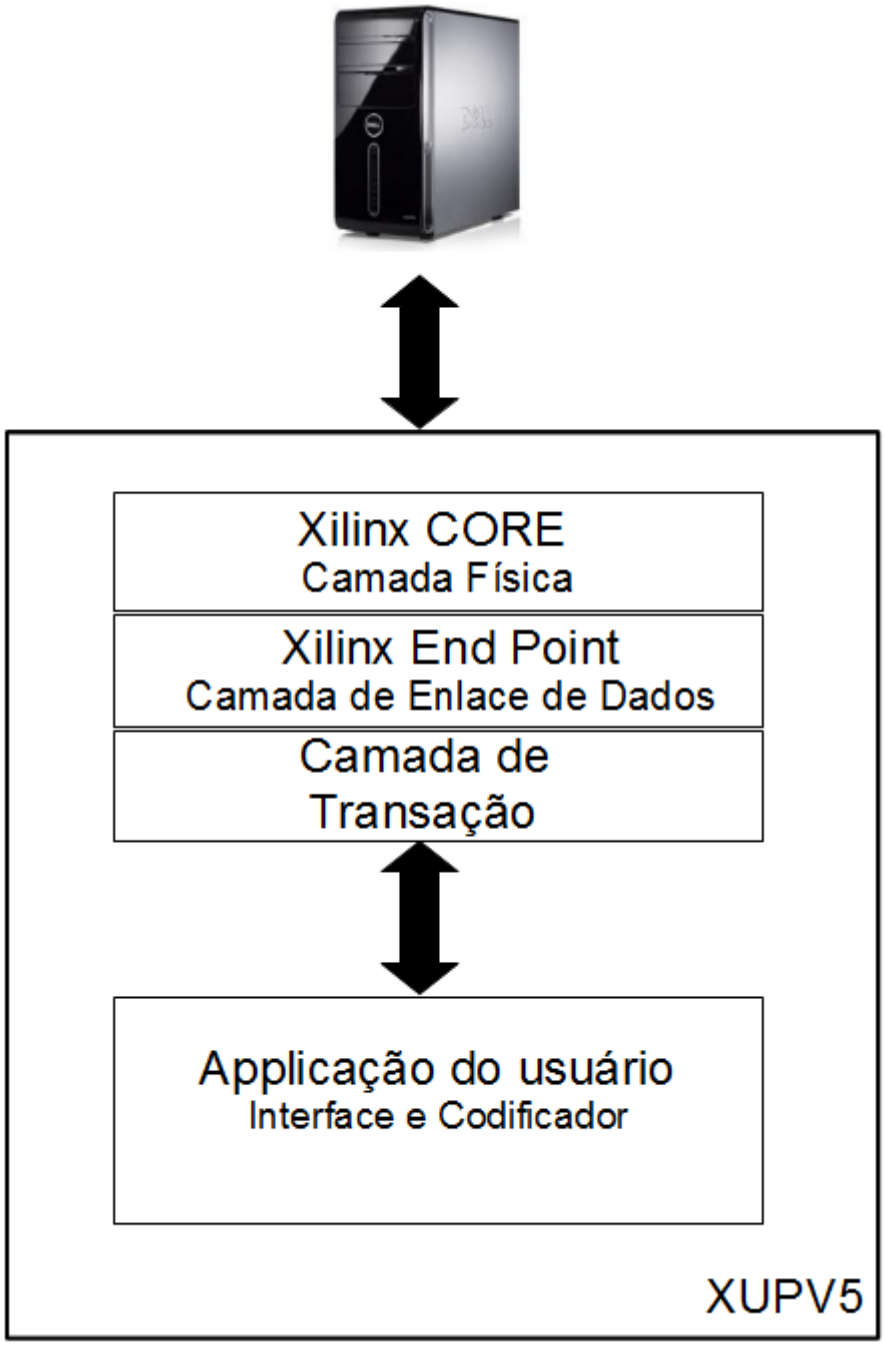

Figura 17: Camadas do PCI Express

6.2.1 Camada de Transação

As transmissões são realizadas por meio de pacotes de transação ou TLPs (*Transaction Layer Packets*), cada um deles com uma largura de 64 bits, conforme representado na Figura 18. Além disso, a quantidade de dados presente em cada TLP (*payload*) é variável, neste caso o *payload* é de 3 palavras de 32 bits ou 3 DWORDS.

Cada processo de transmissão ou recepção de dados pode envolver vários TLPs.

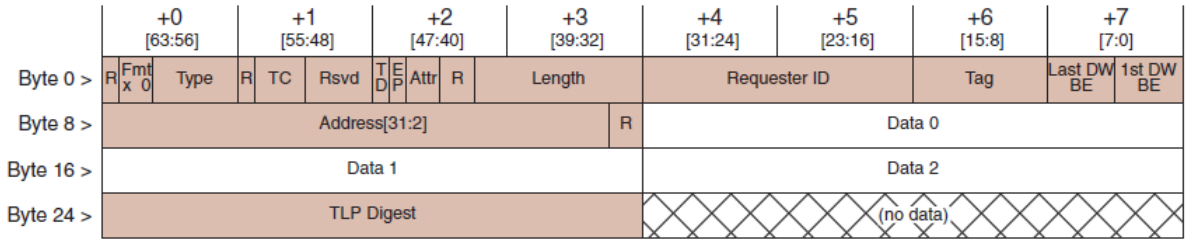

Figura 18: Organização do TLP

A tabela 6 descreve a função das principais entidades do pacote.

Tabela 6: Composição do TLP

Entidade	Bits	Descrição
R	1	Reservado
Fmt x 0	2	Formato do TLP 00 : cabeçalho de 3 DWORDs, sem dados; 01 : cabeçalho de 4 DWORDs, sem dados; 10 : cabeçalho de 3 DWORDs, com dados; 11 : cabeçalho de 4 DWORDs, com dados;
Type	5	Tipo de TLP: configuração, IO
EP	1	Indica TLP corrompido
Length	10	Número de DWORDs de payload
Requester ID	16	Identificação da placa de hardware
Data 0 ~ Data 3	32	Dados
TLP Digest	32	Verificação de erro

6.2.2 Gerência do Processo

A gerência do processo de codificação é realizada por software. O processo se inicia pela configuração do DMA conforme descrito no item 6.3.1 , seguido de duas tarefas, uma delas compõe o pacote de dados a ser transmitido e a outra reconfigura o DMA a cada novo pacote. Uma vez que a configuração estiver pronta e o pacote montado o DMA realizará as transferências da maneira determinada durante a configuração.

6.3 Desenvolvimento da Interface PCI Express

O processo de transferência é dividido em 3 etapas distintas. Inicialmente devem ser configuradas uma variáveis do processo, em seguida os dados começam a ser transferidos no sentido placa para o barramento PCI Express (*B2P*) e/ou no sentido barramento PCI Express para a placa (*P2B*), dependendo da configuração.

6.3.1 Fase de Configuração

A fase de configuração é significativamente mais lenta do que as etapas seguintes já que neste momento o DMA não está configurado, obrigando as transferências a serem executadas diretamente pelo software por meio de escritas e leituras programadas.

Existem duas etapas distintas de configuração. A configuração inicial ocorrere uma única vez para diversos pacotes, nessa fase são transmitidas as configurações que permanecem constantes durante todo o processo como por exemplo os dados relativos a placa de hardware utilizada, o tamanho dos TLPs e o endereço de origem e destino dos dados.

Na fase de configuração intermediária, são transmitidos os dados que são alterados ao longo do processo de comunicação, como o sentido da transferência (*B2P* e/ou *P2B*) e os contadores de tamanho dos TLPs.

A Figura 19 representa o espaço de configuração, é nele que ficam armazenadas as informações referentes à configuração.

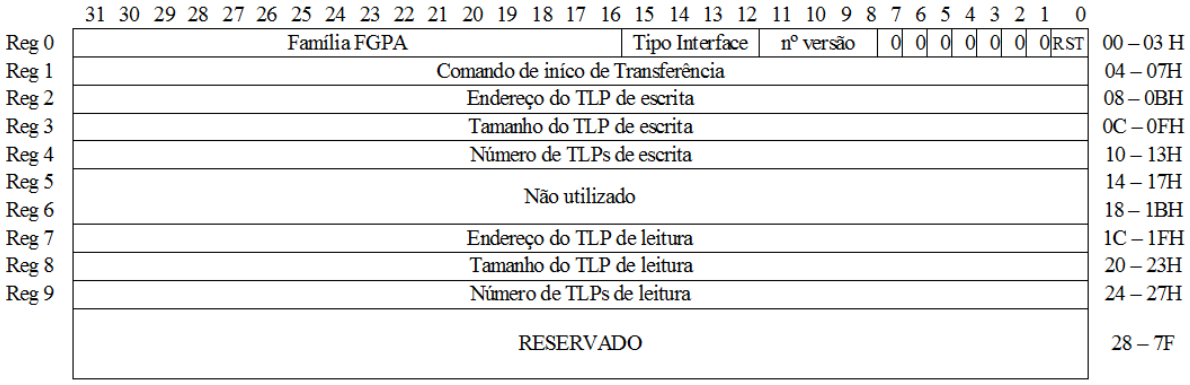

Figura 19: Espaço de Configuração

A última parte de configuração é a determinação do sentido da transferência, que é representado na Figura 19 pelo registrador *Reg 1* (Comando de início de transferência). Após a escrita neste registrador, o DMA assume o controle e as transferências são realizadas de acordo com essa configuração e da maneira descrita nos itens 6.3.2 e 6.3.3 .

6.3.2 Transferência no Sentido B2P (leitura da placa)

Os sinais envolvidos neste procedimento são dados na Tabela 7.

Os sinais trn_tsof_n, trn_teof_n, trn_tsrc_rdy_n e trn_tdst_rdy_n são definidos em lógica negada.

Tabela 7: Sinais envolvidos na transferência B2P

Sinal	Bits	Sentido	Descrição
trn_clk	1	Entrada	Relógio do Sistema
trn_td	64	Saída	Saída de dados
trn_tsof_n	1	Saída	Início do TLP
trn_teof_n	1	Saída	Fim do TLP
trn_trem	8	Saída	Dados remanescentes • 00: dados válidos em trd_td[63:0] • 0F: dados válidos apenas em trn_td[63:32]
trn_tsrc_rdy_n	1	Saída	Dados enviados são válidos (0)
trn_tdst_rdy_n	1	Entrada	Destino pronto para receber dados (0)

Conforme ilustrado na Figura 20, o procedimento se inicia pela aplicação do usuário habilitando o sinal trn_tsof_n indicando que um TLP está iniciando. Este sinal deve permanecer habilitado (em nível baixo) até que o usuário habilite o sinal trn_tsrc_rdy_n, indicando que a origem dos dados está disponível. Neste momento o QWORD (palavra de 64 bits) a ser enviado deve estar pronto em trn_td. Se neste momento o CORE estiver habilitando o sinal trn_tdst_rdy_n, o dados é aceito imediadamente, senão a aplicação deve manter o dado até que trn_tdst_rdy_n seja habilitado pelo CORE.

No ciclo imediatamente após a aceitação do primeiro dado, o sinal trn_tsof_n deve ser desabilitado (em nível alto). A cada novo ciclo, a aplicação deve habilitar trn_tsrc_rdy_n a cada novo QWORD transmitido (desde que o CORE esteja habilitando trn_tdst_rdy_n).

No último QWORD, a aplicação deve habilitar trn_teof_n, indicando que a sequência de dados está terminando. Além disso, é preciso informar se os 8 bytes do QWORD são válidos (trn_trem = 0x00) ou se apenas os 4 primeiros (trn_trem = 0x00). No ciclo seguinte, deve-se desabilitar trn_teof_n para permitir a transferência dos demais TLPs.

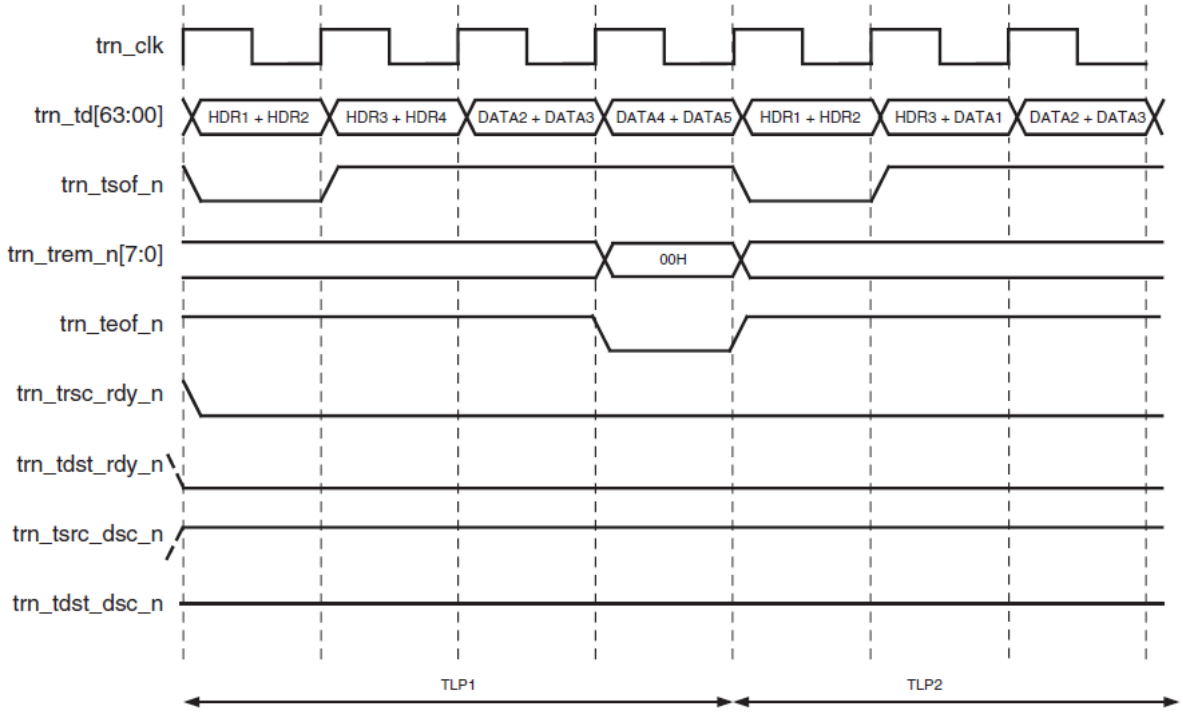

Figura 20: Comunicação PCI Express - B2P

Após o final da transferência de um TLP, outro é transmitido até que todos os TLPs definidos na configuração tenham sido transmitidos.

6.3.3 Transferência no Sentido P2B (escrita na placa)

Os sinais envolvidos neste procedimento são dados na Tabela 8.

Os sinais trn_rsof_n, trn_reof_n, trn_rsrc_rdy_n e trn_rdst_rdy_n são definidos em lógica negada.

Tabela 8: Sinais envolvidos na transferência P2B

Sinal	Bits	Sentido	Descrição
trn_clk	1	Entrada	Relógio do Sistema
trn_rd	64	Entrada	Entrada de dados
trn_rsof_n	1	Entrada	Início do TLP
trn_reof_n	1	Entrada	Fim do TLP
trn_rrem	8	Entrada	Dados remanescentes • 00: dados válidos em trd_td[63:0] • 0F: dados válidos apenas em trn_td[63:32]
trn_rsrc_rdy_n	1	Entrada	Dados recebidos são válidos (0)
trn_rdst_rdy_n	1	Entrada	Destino pronto para receber dados (0)

Conforme ilustrado na Figura 21, o procedimento se inicia pelo CORE, quando ele estiver pronto para enviar um TLP ele deve habilitar trn_rsof_n e trn_rsrc_rdy_n (indicando que o dados é valido) e apresentar o QWORD em trn_rd. Esses sinais permanecem neste estado até que a aplicação habilite trn_rdst_rdy_n (aceitação dos dados).

No ciclo seguinte, o CORE desabilita trn_rsof_n e habilita trn_rsrc_rdy_n a cada novo QWORD. Para paralisar a transferência deve-se desabilitar trn_rdst_rdy_n, o CORE então mantem os sinais em seu estado atual atém que trn_rdst_rdy_n seja desabilitado pela aplicação.

No último QWORD, o CORE desabilita trn_reof_n, indicando que o TLP está terminando. O CORE também informa quantos bits do QWORD contém dados válidos. Se trn_rrem for 0x00, então os 8 bits contém dados válidos (trn_rd[63:0]), do contrário, se trn_rrem for 0x0F então apenas os 4 primeiros bytes contém dados válidos (trn_rd[63:32]). No ciclo seguinte o CORE desabilita trn_reof_n finalizando do TLP.

Após o final da transferência de um TLP, outro será transmitido até que todos os TLPs definidos na configuração tenham sido transmitidos.

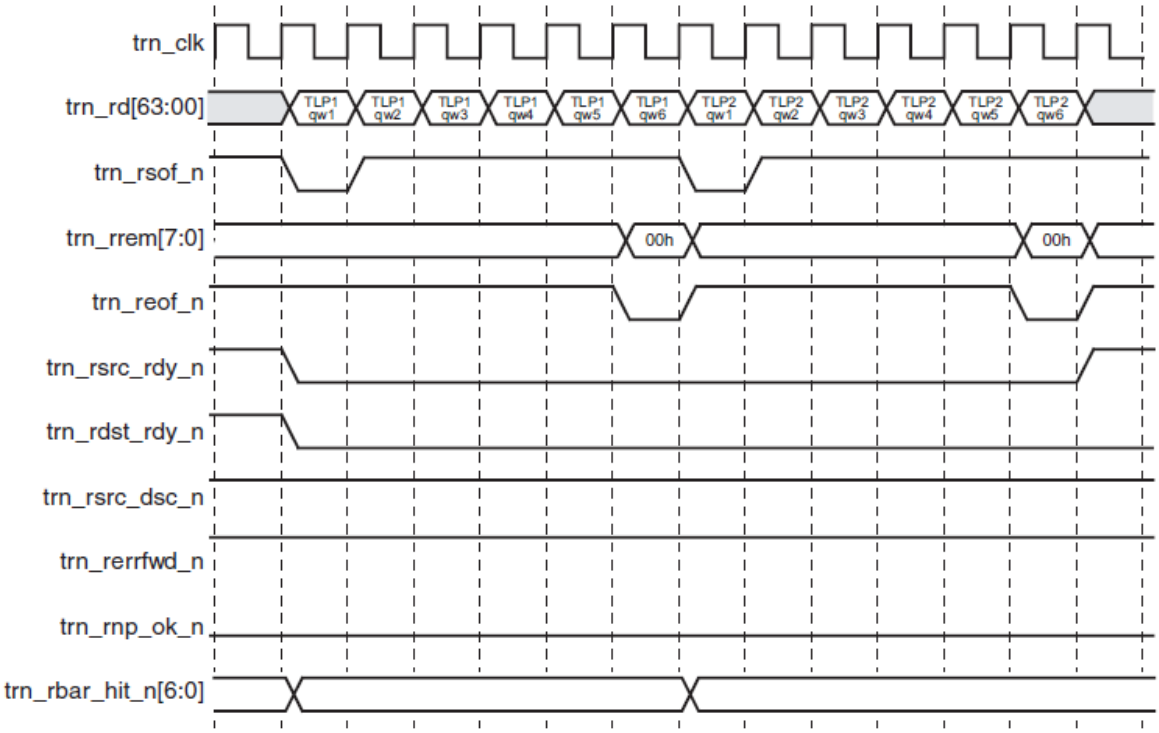

Figura 21: Comunicação PCI Express - P2B

6.4 Integração com o Codificador H.264

Para garantir a compatibilidade do codificador com ambas as interfaces de comunicação desenvolvidas (PCI e PCI Express), utilizou-se o mesmo codificador e a mesma interface de ligação ao codificador representadas na Figura 11 . A essa interface foi conectada um módulo de interconexão integrando a comunicação PCI Express à interface do codificador, conforme representado da Figura 22.

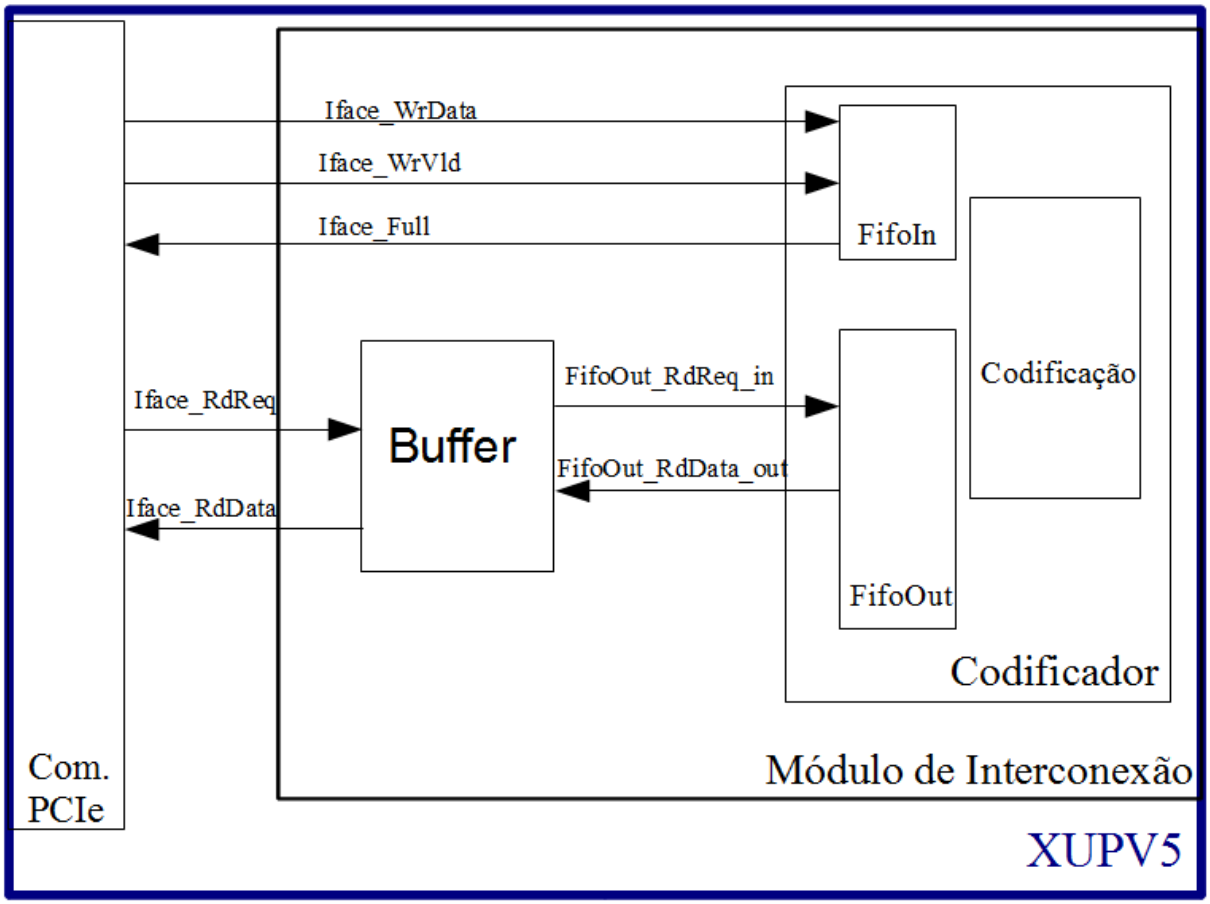

Figura 22: Integração com o Codificador H.264

Neste procedimento, o software gerenciador inicia a transferência de um pacote desde a sua origem, passando pela interface de comunicação PCI Express, até a interface do codificador. Neste módulo é realizada a codificação do pacote que é em seguida transmitido ao buffer. Os dados ficam armazenados no buffer até que o software gerenciador requeira os dados codificados. Neste ponto, a interface PCI Express lê o conteúdo do Buffer e transmite os dados nelo contido de volta ao computador de origem.

6.5 Resultados Práticos

6.5.1 Desempenho da Comunicação PCI Express

De início, foi analisado apenas o desempenho da comunicação, ou seja, os dados foram transmitidos à placa e devolvidos em seguida sem modificação. Esse ensaio permitiu verificar as dimensões ideais para o tamanho do pacote de dados a ser transmitido e o tamanho do quadro (TLP). Deve-se observar no entanto que o driver da placa impõe um limite máximo de 32 TLPs por pacote e 32 DWORDS por TLP, ou seja um limite máximo de 4096 bytes por pacote.

Para a realização do ensaio, o projeto gravado na placa é gerenciado pela aplicação em software fornecida pela Xilinx para o teste de desempenho (XILINX, 2010). Essa aplicação, cuja interface está ilustrada na Figura 23, permite variar o tamanho do pacote e dos TLPs e verificar a taxa de dados obtida.

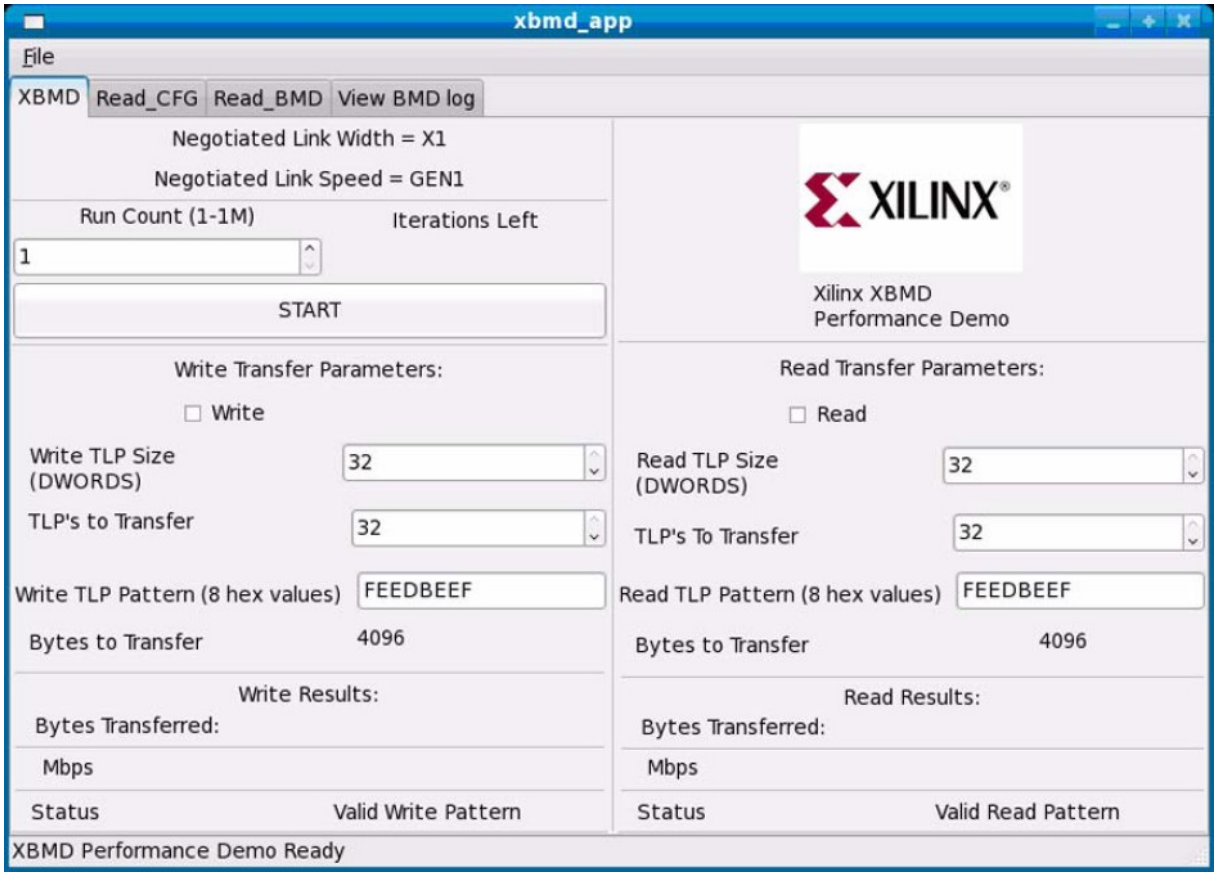

Figura 23: Aplicação Xilinx para o teste de desempenho

Após uma série de ensaios com a Aplicação Xilinx, pôde-se gerar a Tabela 9 onde se pode concluir que o melhor desempenho é obtido para quadros de 32 DWORDS e pacotes de 16 TLPs. Observa-se que esse resultado é coerente com o fato de que a cada novo pacote existe uma fase de configuração intermediária, conforme relatado no item 6.3.1 , assim, o uso de pacotes maiores minimiza o número de etapas de configuração necessárias, maximizando o desempenho, na medida que o tempo mais lento da configuração é menos significativo em relação ao tempo total do processo.

As taxas de dados mostradas na tabela foram obtidas a partir da média de 10 execuções.

Tabela 9: Taxa de dados para diferentes pacotes

DWORDS por TLP	TLPs por pacote	Bytes por pacote	Taxa de dados Mbits/s	Taxa de dados Mbytes/s
1	1	4	7,6	1,0
1	4	16	27,3	3,4
1	16	64	46,7	5,8
1	32	128	57,0	7,1
4	1	16	32,1	4,0
4	4	64	99,0	12,4
4	16	256	173,2	21,7
4	32	512	201,6	25,2
16	1	64	109,8	13,7
16	4	256	290,9	36,4
16	16	1024	412,7	51,6
16	32	2048	467,2	58,4
32	1	128	189,2	23,7
32	2	256	297,4	37,2
32	4	512	430,9	53,9
32	5	640	463,6	58,0
32	16	2048	596,8	74,6
32	32	4096	571,1	71,4

Nota-se que para quadros de 32 DWORDS e pacotes de 16 TLPs consegue-se atingir uma taxa de dados de **74,6 Mbytes/s**, portanto superior à meta de 27 Mbytes/s.

Deve-se ainda analisar porque pacotes com TLPs maiores possuem melhor desempenho do que pacotes com TLPs menores para um mesmo tamanho de pacote. Embora a Figura 21 admita a possibilidade de que os TLPs ocorram sem interrupções uns após os outros, na prática não há garantia de que isso irá ocorrer, conforme ilustrado na Figura 24. Por essa razão, este projeto utiliza sempre TLPs de 32 DWORDS, minimizando assim o número de TLPs necessárias por pacote.

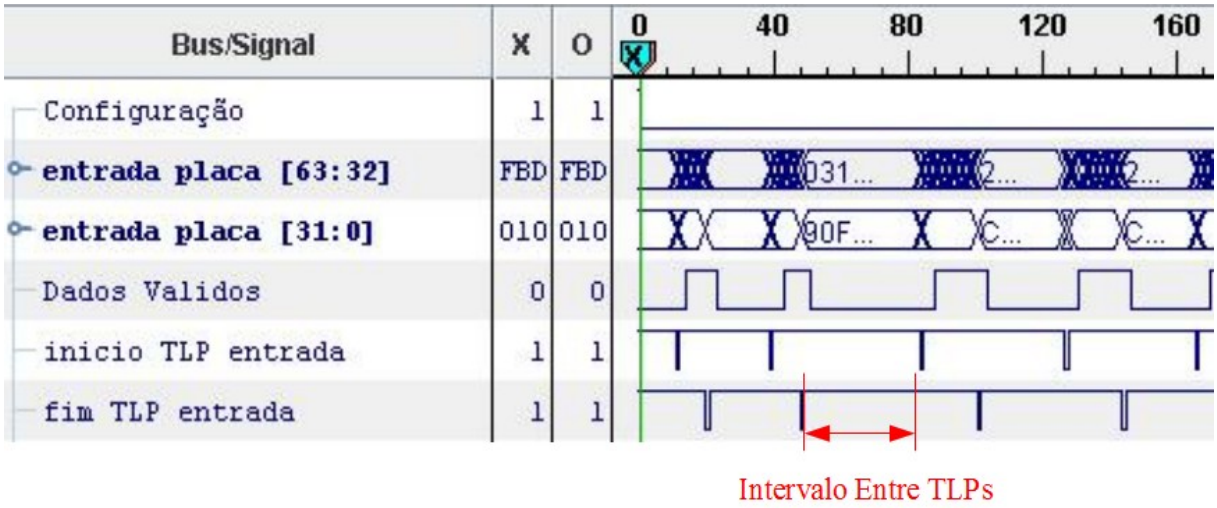

Figura 24: Sequência dos TLPs

6.5.2 Desempenho após integração com o Codificador H.264

Para o atendimentos das necessidades específicas do Codificador H.264 a interface de comunicação teve que sofrer algumas modificações que influenciam o seu desempenho.

O Codificador H.264 trata matrizes de 16x16 pixels, o que gera pacotes de 256 bytes. Dessa forma, seriam necessários 2 TLPs de 32 DWORDS na entrada do codificador. Pode-se observar na Tabela 9 que essa é uma situação de baixo desempenho para o módulo de comunicação.

Outra característica do codificador é que ele trabalha em 32 bits, enquanto que o módulo de comunicação trabalha em 64 bits para atendimento da norma PCI Express. Dessa maneira, a cada dois dados transmitidos, apenas um será dado útil, de tal forma que seriam necessários 4 TLPs de 32 DWORDS totalizando **512 bytes.**

Esse problema pode ser evitado conectando-se uma FIFO entre o módulo de comunicação e o módulo de codificação. Essa melhoria, no entanto, não foi implementada já que o codificador deve ser modificado de forma a trabalhar em 64 bits.

O codificador necessita ainda receber no início de cada pacote um dado de configuração que define seus parâmetros de codificação. Dessa forma, mais 2 DWORDs devem ser transmitidos, já que um é perdido na adaptação 64/32 bits.

Para a transmissão desses dados adicionais, um novo TLP deve ser transmitido já que não é possível obter o número exato de DWORDs por TLP necessário. Esse TLP adicional, no entanto, não provoca uma queda de desempenho, conforme se pode observar na Figura 25.

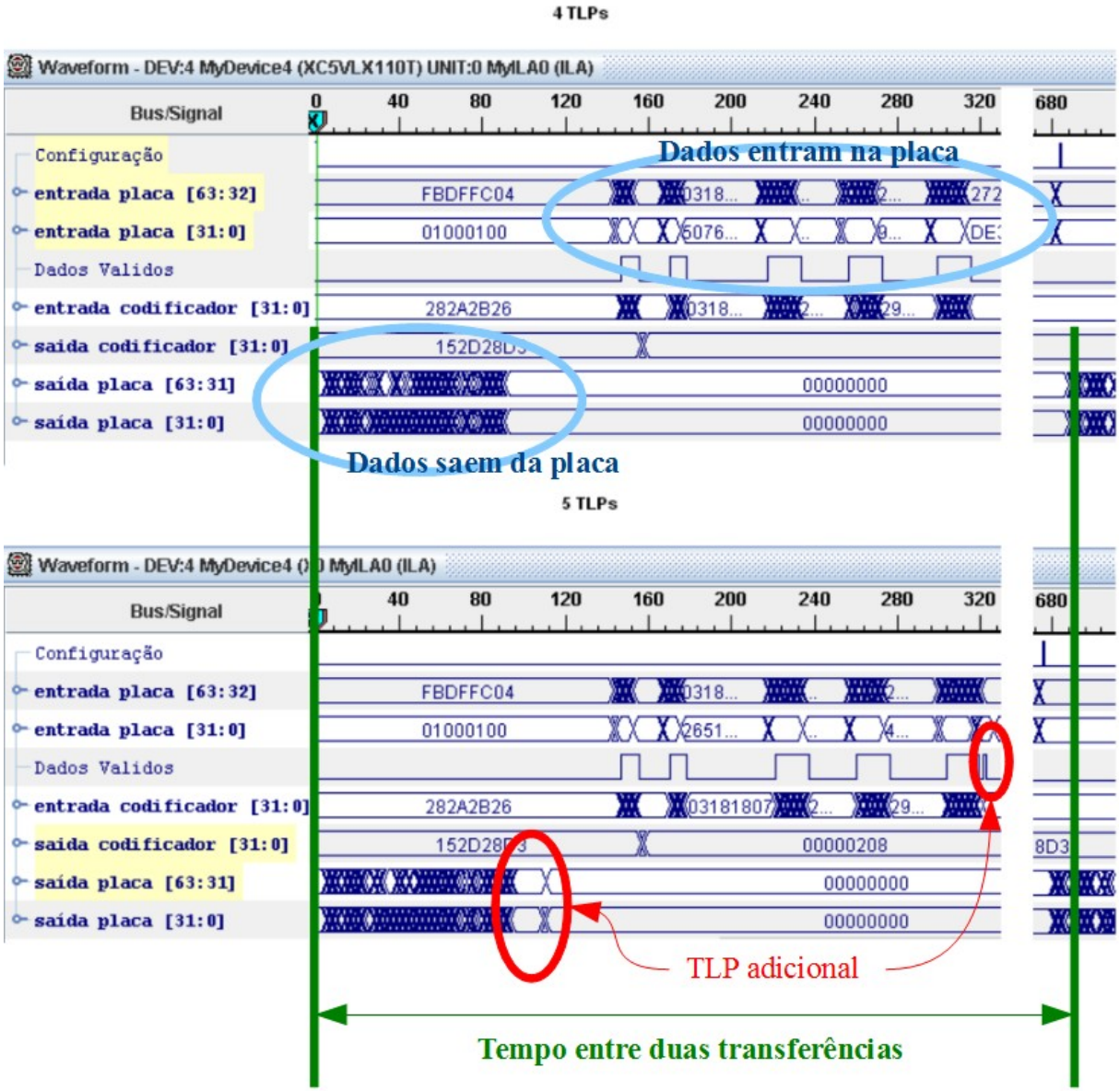

Figura 25: Impacto de um TLP adicional no desempenho da comunicação

Deve-se observar que a adição de um novo TLP praticamente não modificou o tempo da transferência. Em ambos os casos a transferência durou aproximadamente 680 ciclos de 62,5 Mhz, ou seja, aproximadamente **10,8 us** para se tratar um pacote de **520 bytes** de dados, o que corresponde a uma taxa de **48,1 Mbytes/s.**

Assim, o codificador foi ajustado para ser alimentado com uma entrada de dados de 5 TLPs de 32 DWORDS, totalizando uma transferência de 640 bytes dos quais apenas 520 bytes corresponde a dados úteis.

A saída de dados do módulo codificador, conforme a Figura 4 definida no capítulo 4 , deve conter os dados codificados e os dados reconstruídos, portanto se são recebidos na placa 256 bytes, 512 bytes deverão ser devolvidos. Logo seriam necessários no sentido inverso 4 TLPs de 32 DWORDS.

O codificador possui ainda uma última característica relevante. Entre a transmissão do dado codificado e a transmissão do dado reconstruído, o codificador transmite ainda um DWORD de controle. Dessa forma, aqui também um novo TLP será transmitido apenas para esse dado adicional.

Neste ponto, no entanto, não é necessário duplicar o número de TLPs em função da adaptação 64/32 bits entre comunicador e codificador, isso se deve à arquitetura do projeto. Conforme ilustrado na Figura 22, ao final da recepção dos dados na placa, eles são tratados e armazenados em um buffer até que seja requisitada a leitura. Assim quando for realizada a transferência da placa para o computador, basta transmitir dois elementos do buffer por vez.

Logo a saída do codificador contará com 5 TLPs de 32 DWORDs, totalizando uma transferência de 640 bytes dos quais apenas 516 correspondem a dados úteis.

A Figura 26 ilustra um processo de leitura e escrita. O procedimento se inicia ao final da configuração por uma leitura dos dados já tratados pela placa, em seguida, os próximos

dados são gravados na placa e transferidos ao codificador. Finalizado o processo de escrita, o codificador começa a trabalhar e a escrever os dados tratados em um buffer de saída. Enquanto os dados estão sendo codificados, o software de gerência realiza duas tarefas em paralelo: configuração do próximo pacote e montagem do próximo pacote. Quando o pacote estiver pronto para ser transferido, o software dá início ao próximo ciclo iniciando a leitura dos dados codificados transferidos no ciclo anterior.

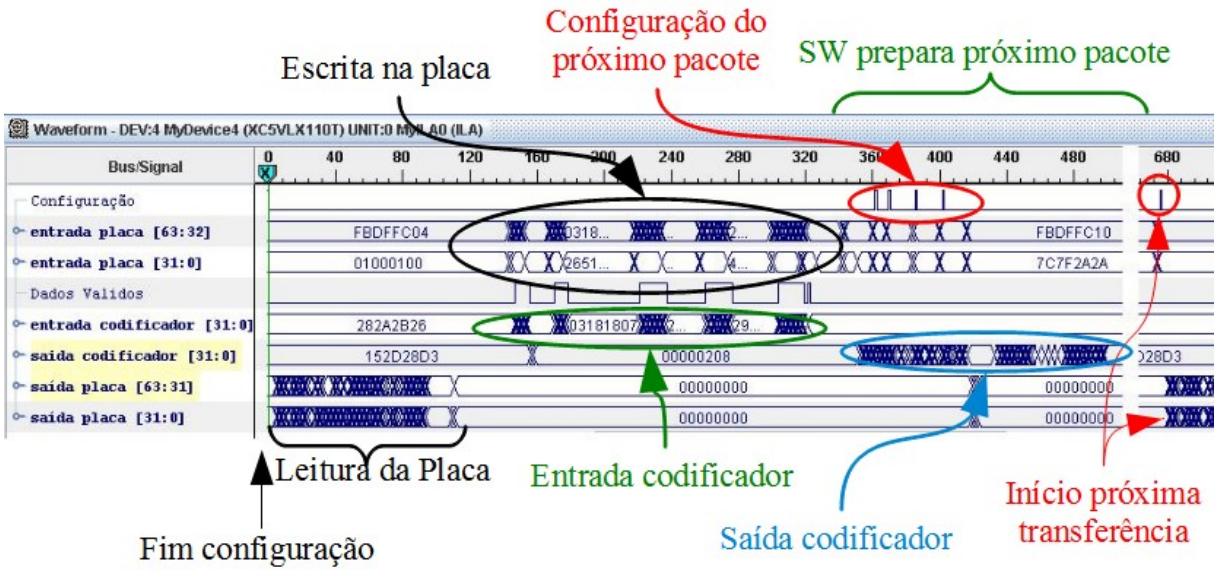

Figura 26: Integração ao codificador

Na Figura 26, assim como na Figura 25 é possível verificar que o tempo entre cada uma da transferências é de aproximadamente 10,8 us (680 ciclos de 62,5 MHz), período em que são codificados 520 bytes, totalizando uma taxa de **48,1 Mbytes/s**. Deve-se notar que essa taxa continua sendo superior à meta de 27 Mbytes/s proposta no capítulo 4 .

Logo, foi possível desenvolver uma interface de comunicação para integração entre Software e Hardware em FPGA capaz de atender aos objetivos propostos.

7 ENSAIOS

Para uma análise completa do desempenho, foram feitos também ensaios considerando transações de leituras e escritas separadamente. A Figura 27 permite verificar uma queda de desempenho, isso porque se adiciona uma etapa adicional de configuração.

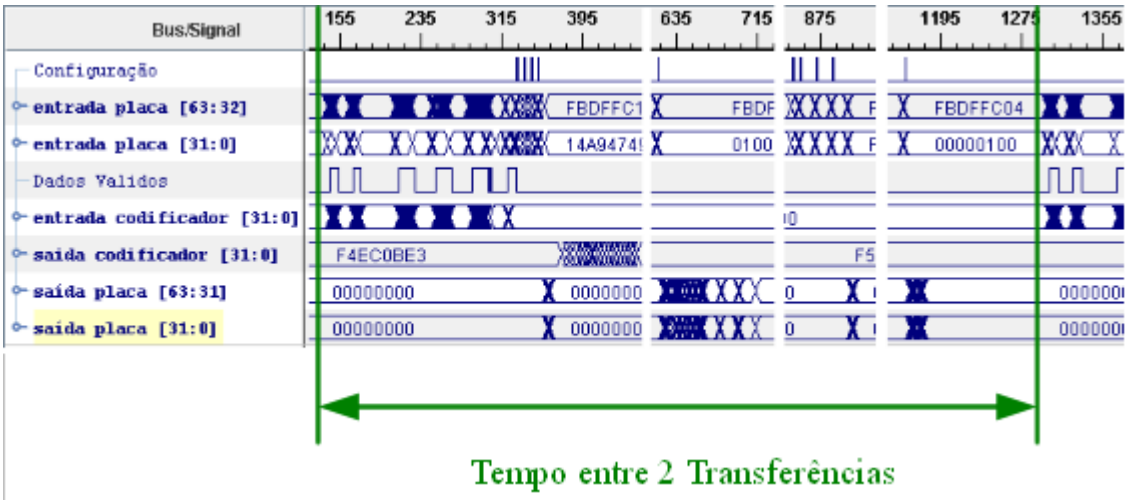

Figura 27: Escritas e leituras separadas

Na Figura 27 é possível verificar que o tempo entre cada uma da transferências é de aproximadamente 18,5 us (1160 ciclos de 62,5 MHz), período em que são transferidos 520 bytes, totalizando uma taxa de **28,1 Mbytes/s**.

Comparando com as demais interfaces de comunicação desenvolvidas pode-se obter a Tabela 10.

Tabela 10: Comparativo de Desempenho

Método	Taxa de Dados (Mbytes/s)
PCI sem DMA	1,4
PCI com DMA	10,2
PCI Express (leituras e escritas simultâneas)	**48,1**
PCI Express (leituras e escritas separadas)	28,1
Software	27

A análise da Tabela 10 permite concluir que a melhor interface de comunicação desenvolvida foi a interface PCI Express desenvolvida na placa XUPV5 mediante a realização de leituras e escritas durante uma mesma transação, ou seja é feita a leitura da codificação de um pacote seguido da escrita do pacote seguinte. Neste método, verifica-se uma taxa de dados 78% superior ao software.

8 CONCLUSÃO

Este projeto desenvolveu três interfaces de comunicação em duas placas de hardware diferentes: NetFPGA e XUPV5. A Tabela 10 permitiu verificar que o melhor desempenho foi obtido na placa XUPV5 por meio da comunicação PCI Express, de fato foi a única interface a apresentar melhor desempenho do que o codificador em software.

Deve-se lembrar que a interface de comunicação por PCI Express em sua melhor condição de operação (16 TLPs de 32 DWORDs) é capaz de fornecer uma taxa de dados de **74,6 Mbytes/s**, esse desempenho não se repetiu na integração com o codificador devido a limitação de trabalhar com pequenos pacotes de dados. Assim, ainda é possível obter desempenhos superiores adequando o codificador a processar maiores pacotes de dados.

REFERÊNCIAS

BUDRUK,R.; ANDERSON,D.;SHANLEY,T. PCI Express System Architecture. MindShare Inc. 2003. p.10.

CHIPSCOPE PRO 12.1. Software and Cores. User Guide. 2010

HUSEMANN R. et al. Hardware Integrated Quantization Solution for Improvement of Computational H.264 Encoder Module. 2010.

HUSEMANN R. et al. Proposta de Solução de HW/SW para o Módulo de Transformadas de um Codificador H.264/SVC. 2010.

JSVM Software Manual. JSVM 6.8.2. 2006.

NETFPGA, NetFPGA User Guide. Disponível em http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide. Acessado em 21/11/2010 às 17h20. 2010

PCI Express Base Specification. Revision 1.0a. 2003. p. 36-37.

REDIESS F. et al. Projeto de Hardware para a Compensação de Movimento do Padrão H.264/AVC de Compressão de Vídeo. 2006. p.1-2.

RICHARDSON, I.E.G. H.264 and MPEG-4 Video Compression. Video Coding for Next Generation Multimedia. Wiley. 2003. p. 15.

SUGAWARA,M. Super Hi-Vision – research on a future ultra-HDTV system. EBU Technical Review. 2008.p.2-3.

XILINX, Bus Master DMA Performance Demonstration. Reference Design for the Xilinx Endpoint PCIExpress® Solutions. Disponível em http://www.xilinx.com/support/documentation/application_notes/xapp1052.pdf. Acessado em 21/11/2010 às 20h46. 2010.

XILINX, Core Generator Guide. Disponível em http://www.xilinx.com/itp/xilinx6/books/docs/cgn/cgn.pdf. Acessado em 21/11/2010 às 21h15. 200X.

XILINX, XUPV5-LX110T User Manual. ML505/ML506/ML507 Evaluation Platform. 2007.

WATSON, G.; McKEOWN, N.; CASADO, M. NetFPGA: A Tool for Network Research and Education. 200X

ZELENOVSKY, R.; MENDONÇA, A. PC: Um Guia Prático de Hardware e Interfaceamento. MZ Editora Ltda. 2006. p.697-698.