RICARDO BRANDÃO SAMPAIO

PROJETO DE UM NÚCLEO DE CONVERSÃO DA INTERFACE PCI PARA INTERFACE OCP-IP™

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, em cumprimento às exigências para obtenção do grau de Mestre em Engenharia Elétrica com concentração na área de Processamento da Informação.

Orientadores:
Prof. Dr. Raimundo Carlos Silvério Freire
Prof. Dr. Elmar Uwe Kurt Melcher

CAMPINA GRANDE – PB

2004

S192p Sampaio, Ricardo Brandao
Projeto de um nucleo de conversao da interface PCI para interface OCP-IPtm / Ricardo Brandao Sampaio. - Campina Grande, 2004.
142 f. : il.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal de Campina Grande, Centro de Ciencias e Tecnologia.

1. PCI 2. OCP-IP 3. FPGA 4. Interface 5. Nucleo 6. Dissertacao I. Freire, Raimundo Carlos Silverio, Dr. II. Melcher, Elmar Uwe Kurt, Dr. III. Universidade Federal de Campina Grande - Campina Grande (PB) IV. Título

CDU 004.5:621.3(043)

PROJETO DE UM NÚCLEO DE CONVERSÃO DA INTERFACE PCI PARA INTERFACE OCP-IP™

RICARDO BRANDÃO SAMPAIO

Dissertação Aprovada em 20.02.2004

ELMAR UWE KURT MELCHER, Dr., UFCG
Orientador

RAIMUNDO CARLOS SILVÉRIO FREIRE, Dr., UFCG
Orientador

JOSEANA MACEDO FECHINE, D.Sc., UFCG
Componente da Banca

JOSÉ ANTONIO GOMES DE LIMA, D.Sc., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Fevereiro - 2004

Dedico com todo meu amor
a minha mãe,
ao meu pai
e as minhas filhas.

AGRADECIMENTOS

A Deus, por colocar em meu caminho pessoas e instituições que me auxiliaram e apoiaram nesta jornada.

Institucionais:

CEFET-AM
Pela liberação para realizar estudos em outra região e ajuda financeira para o deslocamento.

UFCG
Pela disponibilidade da vaga e uso de sua infra-estrutura para realização do curso, em especial a Coordenação de Pós-Graduação em Engenharia Elétrica – COPELE, ao Departamento de Engenharia Elétrica – DEE e ao Departamento de Sistemas e Computação – DSC.

CNPq
Pelo apoio financeiro através da bolsa de estudo.

Pessoais:

Quero expressar ao Professor Dr Raimundo Carlos Silvério Freire, toda minha estima e reconhecimento pelas oportunidades de realização deste mestrado e a outros momentos de estudos proporcionados por seu intermédio (CBA2002, PROCAD 2001/02/03 e curso de micro-fabricação de *chip* UNICAMP/CCS 2003), agradeço também por seus valiosos ensinamentos nas disciplinas sob sua orientação, e por fim, pelos muitos momentos agradáveis e descontraídos que nos proporcionou.

Ao professor Dr Elmar Uwe Kurt Melcher, todo meu reconhecimento pelos valiosos ensinamentos, apoio, compreensão, e enorme desprendimento para ajudar-me nos momentos mais difíceis, sem o qual teria sido impossível a realização desta dissertação.

Professores Benedito Antônio Luciano, Gurdip Singh Deep, José Sérgio da Rocha Neto e José Gutembergue de Assis Lira, pelos ensinamentos relevantes através de suas aulas e palestras.

Aos funcionários Pedro e Ângela (COPELE) e Eleonora (MiniBlio/DEE) pelo apoio operacional.

Aos amigos Luiz Brunelli pelas valiosas colaborações no trabalho e Lívia Lima por sua ajuda em muitos momentos e a todos os colegas do LIMC pelos apoios diretos ou indiretos através de incentivos e muitos momentos de descontração.

RESUMO

Nesta dissertação se descreve todas as etapas necessárias ao desenvolvimento do projeto de um Núcleo de Propriedade Intelectual e como elaborar sua validação funcional, também se descreve todas as plataformas necessárias para a elaboração de cada etapa do projeto. Esse Núcleo projetado tem por função desempenhar o papel de conversão da interface do barramento local PCI, de largura de 32 *bits* e freqüência 33 MHz, para uma interface cujo barramento faça uso do protocolo OCP-IP™ de mesma largura e freqüência da interface do barramento PCI. Para desenvolvimento da Interface foi necessário elaborar uma unidade PCI Destino e uma unidade OCP™ Mestre. O projeto do Núcleo de Propriedade Intelectual foi totalmente realizado na linguagem de descrição de dispositivos *SystemC™*, bem como o ambiente de teste por simulação e a aplicação de dispositivo do usuário. A implementação do Núcleo foi especialmente desenvolvida para adaptar-se a dispositivo reconfigurável como a FPGA FLEX 10KE da empresa *Altera* que foi usada na etapa de prototipagem. Os resultados obtidos em ambiente de simulação e implementação satisfizeram as exigências inicias do projeto. Ao fim do projeto, na etapa de simulação, tinha-se respostas para os ciclos de leitura e escrita de configuração e ciclos de leitura e escrita na memória no modo simples. Na etapa de implementação em FPGA, somente uma parte do código, o trecho referente aos ciclos de leitura e escrita nos registradores de configuração da PCI, foi implementado. Nessa fase de implementação, procurou-se atender a todos os requisitos de tempo exigidos pelo padrão PCI revisão 2.2.

Palavras –chaves: PCI, OCP-IP, FPGA, Interface, Núcleo.

ABSTRACT

In this work we describes all the necessary stages to the development of design a Core of Intellectual Property (IP SoftCore) and how to elaborate your functional validation, also we describe all the necessary platforms for the elaboration of each stage of the project. That design Core has for function perform the conversion play of the interface of PCI local bus, of width of 32 *bits* and frequency 33 MHz, for a the interface whose the bus does use of the protocol OCP-IP™ of same width and frequency of the interface of PCI bus. For development of the Interface was necessary to elaborate a PCI Target unit and an unit OCP™ Master. The design of IP SoftCore was totally accomplished in the Hardware Description Language SystemC™, as well as the test environment for simulation and the device application of the user. The implementation of the Core was especially developed to adapt itself for device reconfiguration as FPGA FLEX 10KE of the company Altera that it was used in prototype stage. The results obtained in simulation environment were satisfactory, at the end of the project had itself answers for setup and cycles reading and writing cycles of reading and writing in the memory. The implementation stage was incomplete, only the setup environment was implemented in FPGA (reading and writing cycles in the setup configuration space).

Keywords: PCI, OCP-IP, FPGA, Interface, SoftCore.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABEL	*Advanced Boolean Equation Language*
ASIC	*Application-Specific Integrated Circuit*
BIOS	*Basic Input and Output System*
Bit	*Binary Digit*
CAD	*Computer Aided Design*
CI	Circuito Integrado
CIF	*Common Intermediate Format*
CLB	*Configurable Logic Block*
CMOS	*Complementary Metal-Oxide Semiconductor*
DCR	*Device Control Register Bus*
DH	*Design House*
DIMM	*Double In Line Memory Module*
DMA	*Direct Memory Access*
DOS	*Disk Operation System*
EDA	*Electronic Design Automation*
EDIF	*Electronic Design Interchange Format*
EISA	*Extended Industry Standard Architecture*
EPLDs	*Electrically Programmable Logic Devices*
E/S	Entrada/Saída
FPGA	*Field Programmable Gate Array*
GB	*Giga Bytes*
GDSII	*Graphic Design System II* (GDS2)
GHz	*Giga Hertz*
HDL	*Hardware Description Language*
IEEE	*Institute of Electrical and Electronic Engineers*
I/O	*Input/Output*
IP	*Intellectual Property*
ISA	*Industry Standard Architecture*
ISP	*Instruction Set Processor*
LE	*Logic Element*
LM	*Logic Modules*
Mb/s	*Mega bits por segundo*

Lista de Abreviaturas e Siglas (continuação)

MB/s	*Mega Bytes por segundo*
MCA	*Micro Channel Architecture*
MHz	*Mega Hertz*
MIPS	Milhões de Instruções Por Segundo
OCP	*Open Core Protocol™*
OPB	*On-Chip Peripheral Bus*
OS	*Operation System*
PAL	*Programmable Array Logic*
PC	*Personal Computer*
PCI	*Peripheral Component Interconnect Bus*
PLB	*Processor Local Bus*
PMC	*PCI Mezzanine Card*
PNM	Programa Nacional de Microeletrônica
PNP	*Plug-and-Play*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
RTL	*Register Transfer Level*
SoC	*Systems on a Chip*
SDRAM	*Synchronous Dynamic Random Access Memory*
SRS	*Semiconductor Reuse Standards*
SCSI	*Small Computer System Interface*
SVN	*Subversion*
UART	*Universal Asynchronous Receiver Transmitter*
Verilog	*Verifying Logic in general*
VESA	*Video Electronics Standards Association*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VLB	*VESA Local Bus*
VLSI	*Very Large Scale Integration*
VSI	*Virtual Socket Interface*
WWW	*Word Wide Web*
HTTPs	*Hyper Text Transfer Protocol - secure*

1. INTRODUÇÃO

A evolução de sistemas digitais, nas últimas duas décadas, colocou novos requisitos para projetistas de sistemas. Eles agora precisam projetar interfaces que sejam compatíveis com sistemas de outros fabricantes. O assunto de compatibilidade foi solucionado por sistemas com interfaces de barramento que são padrões na indústria, como ISA, EISA, VESA (VLB) e MCA. Um padrão de interface chamado PCI (*Peripheral Component Interconnect*) foi desenvolvido para atender aos novos requisitos dos sistemas de computadores digitais. As características do padrão PCI incluem freqüências de 33 e 66 MHz em versões de 32 e 64 *bits*, alcançando 533 mega octetos[1] por segundo em sua taxa de transferência máxima.

Apresenta-se nesta dissertação as estratégias de projeto e validação de um programa de Núcleo de Propriedade Intelectual (*IP Soft Core*), destinado a desempenhar o papel de controle de um barramento local padronizado de alto desempenho, (*PCI Local Bus*). O PCI é um padrão de barramento extensamente aceito e utilizado em muitos aplicativos incluindo telecomunicações, sistemas embutidos, placas de periféricos de alto desempenho e em rede. A implementação deste núcleo será realizada através da linguagem de descrição de dispositivo[2] *(HDL) SystemC*, sendo esta flexível, portável e personalizável para aplicações específicas. A implementação é especialmente desenvolvida para adaptar-se bem ao ambiente onde o núcleo reside em dispositivo reconfigurável como o FPGA (*Field Programmable Gate Array*) baseado em RAM (***R**andom **A**ccess **M**emory*).

As realizações e implementações de projetos são de importância relevante para o universo acadêmico, pois é através da montagem e da prática com uma interface que o projetista ganha o real sentimento das dificuldades envolvidas no projeto de um circuito para computador. Nos centros acadêmicos, tão importante quanto realizar experimentos, é realizá-los com o que há de mais moderno disponível. Hoje, tratando-se de projetos de circuitos digitais de rápida

[1] Usa-se freqüentemente a palavra inglesa "***byte***" para designar um conjunto constituído de 8 *bits*, mas nesse texto vamos usar a palavra portuguesa "**octeto**".

implementação, tem-se o FPGA que permite um projeto rápido e versátil. Com um FPGA, pode-se testar rapidamente um circuito digital de relativa complexidade, além de oferecer diversas facilidades de projeto.

Há algum tempo que os sistemas projetados em uma única pastilha (*SoC - Systems on a Chip*) podem alcançar mais de 20 milhões de portas, a uma freqüência operacional de 1 GHz[01]. A fim de implementar tais sistemas, projetistas contam com o reuso de blocos de Propriedade Intelectual (IP - *Intellectual Property*). Desde que os blocos de IPs estão pré-projetados e pré-verificados, o projetista pode se concentrar no sistema completo sem ter que se preocupar com as características individuais dos componentes.

A flexibilidade de um projeto com FPGA, conjugada com a vasta área de aplicações das interfaces para PC, mais o desafio do uso de uma nova linguagem de descrição de dispositivos, o *System*C, para gerar um bloco de IP, somando à participação da UFCG no Programa Nacional de Microeletrônica[02] e no projeto FENIX do programa *Brazil-IP*[03] motivaram a execução deste trabalho.

O restante deste documento, pode ser descrito em linhas gerais, da seguinte forma: no Capítulo 2 são feitas considerações gerais sobre o projeto do núcleo, no Capítulo 3 é apresentada a estrutura da interface, (arquitetura, funcionamento e as conexões das diversas partes usadas para a implementação do dispositivo); no Capítulo 4 comenta-se o algoritmo de programação do núcleo; no Capítulo 5 os recursos de laboratórios usados na implementação do projeto; no Capítulo 6, são apresentados os resultados obtidos na realização dos testes, medições e simulações; no Capitulo 7 tem-se as conclusões e sugestões para trabalhos futuros, após este, tem-se as referências e uma série de Apêndices com resumos das teorias abordadas na dissertação.

---

[2] Nessa dissertação usaremos a palavra portuguesa "**dispositivo**" em substituição a palavra inglesa

2. CAPÍTULO 2 – CONSIDERAÇÕES GERAIS

2.1. Núcleo de IP

A idéia do uso de núcleos de IP é permitir que usuários possam buscá-los na Internet e executá-los no dispositivo FPGA, comunicando-os uns com os outros, e possibilitar que determinado núcleo possa ser removido, quando seu uso não for mais necessário, para que outro seja inserido em seu lugar (Figura 2.1).

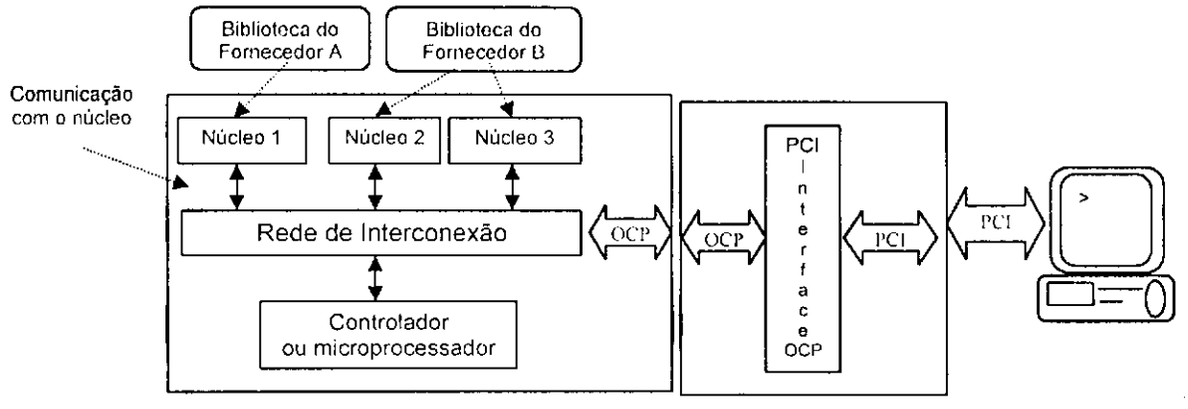


Figura 2.1– Diagrama da aplicação: SoC ←→ Interface OCP_PCI ←→ PC.

Para que esta idéia seja posta em prática, é necessário atender alguns requisitos, como por exemplo, o que trata da distribuição de núcleos através da Internet. É necessária a existência de uma ferramenta que auxilie o projetista na organização e distribuição de seus projetos, mas que também proteja seus direitos autorais[3] (Nesta dissertação, se fez uso da ferramenta SVN - *Subversion*[04], usada no projeto FENIX e na rede Brazil-IP[03]).

A comunicação entre os núcleos pode ser realizada usando-se duas estratégias: **Arquiteturas de Barramento** ou uso de **uma Interface.**

O uso desta estratégia, **Arquiteturas de Barramento**, torna-se atualmente complexa, pois existem poucas, publicamente disponíveis, para guiar os fabricantes, tais como a *CoreConnect* da IBM[05], AMBA da ARM[06] e *WISHBONE* da *Silicore*[07]. Estas arquiteturas de barramento são geralmente vinculadas à arquitetura de um processador[01], tal como o *PowerPC* ou o ARM (Maiores detalhes no Apêndice A).

"***hardware***" comumente usada para designar uma unidade física, um equipamento eletrônico.

Estratégia do uso de uma Interface: Esta abordagem de padronização diferente consiste em não definir um barramento, mas somente uma interface de comunicação, o que é vantagem por não ser vinculada a uma arquitetura específica de um processador. Exemplos desta abordagem são VSI/VIA, SRS/Motorola e OCP-IP™ [4]. Nesta dissertação optou-se pelo uso da interface OCP-IP™ [09] [5]. Maiores detalhes do OCP-IP estão descritos no Apêndice E.

2.2. Características da Interface

Para desenvolver a implementação do núcleo IP, necessita-se de uma placa de circuito impresso com dispositivo FPGA e conector PCI. A interface projetada tem como característica a simplificação dos blocos em estrutura básica com os sinais necessários à implementação de uma Interface PCI Alvo mínima e uma Interface OCP Mestre mínima. É necessário desenvolver um *Backend*[6] *de teste*, o *Device Driver*[7], o *Software*[8] e um *TestBench*[9].

Apresenta-se através da Figura 2.2 o diagrama simplificado do sistema computacional em implementação e o ambiente onde o mesmo está inserido.

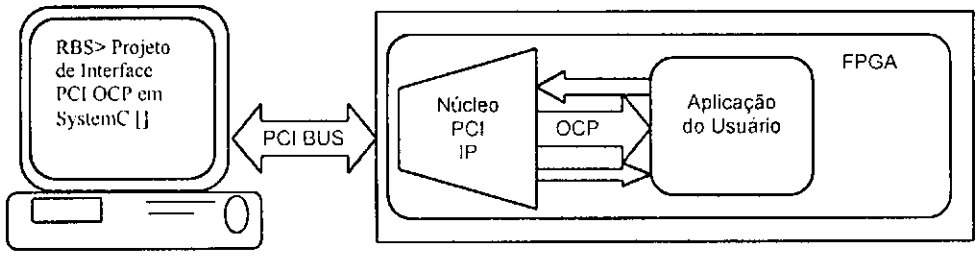


Figura 2.2 – Diagrama simplificado do projeto desenvolvido.

---

[3] Maiores informações na publicação de Dalpasso [08], "*Hardware/Software IP Protection*".
[4] *Open Core Protocol Specification* é de propriedade de *OCP-IP Association*, todos os direitos reservados [09]
[5] A escolha deste padrão foi devido ao fato deste ser o padrão normalizado para uso no projeto *Brazil IP*.
[6] Usa-se freqüentemente a palavra inglesa ***Backend*** para designar o programa de aplicação do usuário, mas nesta dissertação adota-se doravante a expressão em português **Aplicação de Dispositivo do Usuário**.
[7] Camada de *Software* que representa o dispositivo no ambiente do sistema operacional.
[8] Usa-se freqüentemente a palavra inglesa ***Software*** para designar um suporte lógico, conjunto de programas, neste caso trata-se apenas do programa de aplicação do usuário, doravante nesta dissertação usa-se a palavra portuguesa **Programa**.

2.2.1. Metas e Limitações

Toda a arquitetura elaborada visa conformidade com os padrões adotados nas especificações PCI revisão 2.2 e OCP-IP™ [10] revisão 1.0. A seguir são descritas as metas que foram consideradas no desenvolvimento do projeto para a dissertação:

- Velocidade do Relógio [11] da interface PCI é 33MHz, optou-se por esta freqüência por ser a freqüência do barramento PCI em uso no laboratório.
- Interfaces de E/S PCI de largura de banda de 32 *bits* para serem compatíveis com os computadores disponíveis no laboratório.
- Interfaces de E/S OCP-IP de 32 *bits*, foi usado este padrão para tornar mais simples o projeto (compatível como PCI) e por atender a maior parte das Aplicações de Usuários.
- Suporte para uma região de endereço de base (Tipo 0).
- Implementação dos registros de configuração de PCI exigidos.
- Suporte para ciclos únicos (mínimo obrigatório) e ciclos de rajada (para dar suporte às tecnologias mais recentes tipo multimídia) para ciclos de leitura e escrita na memória.
- A geração de paridade exigida.
- Implementação com FPGA da Altera modelo FLEX 10K, que está disponível no laboratório e atende às necessidades do projeto.
- Projeto em *SystemC™* hierárquico, para modificações simples do usuário final.
- Suportar sinal de interrupção do dispositivo para o barramento PCI [12].
- Como verificação do estado da arte em *SystemC*, tem-se o emprego da metodologia *transaction-level random-constrained coverage-driven self-checkingTestBench*.

---

[9] Usa-se freqüentemente a palavra inglesa ***TestBench,*** porém nesta dissertação usa-se a expressão em português **Ambiente de Teste por Simulação** em sua substituição.

[10] O uso da revisão 1.0 deu-se por ser a versão em uso atualmente, embora desde fevereiro de 2003 tenha sido publicada a nova revisão 2.0. Esta só teve a liberação oficial em setembro de 2003.

[11] Doravante nesta dissertação usa-se a palavra portuguesa **Relógio** em substituição à palavra inglesa ***Clock.*** **O** sinal de *Clock* é um pulso regular usado para propósitos de regulagem de tempo ou sincronização, proveniente do circuito de *Clock* que é um circuito que gera pulsos usados para sincronizar equipamento.

[12] O uso desta condição foi de caráter opcional nesta dissertação, não sendo implementada.

2.2.2. Sinais de Entrada e Saída da Interface

Na Figura 2.3, observa-se na ilustração algumas diferenças das principais características entre os sinais provenientes do barramento PCI e do OCP™.

Tem-se como um dos objetivos desta interface, facilitar o projeto do usuário, tornando imperceptível ao projetista essas diferenças. O projetista deve apenas verificar se seu projeto é compatível com o protocolo OCP™. O que torna o projeto do usuário final mais simples.

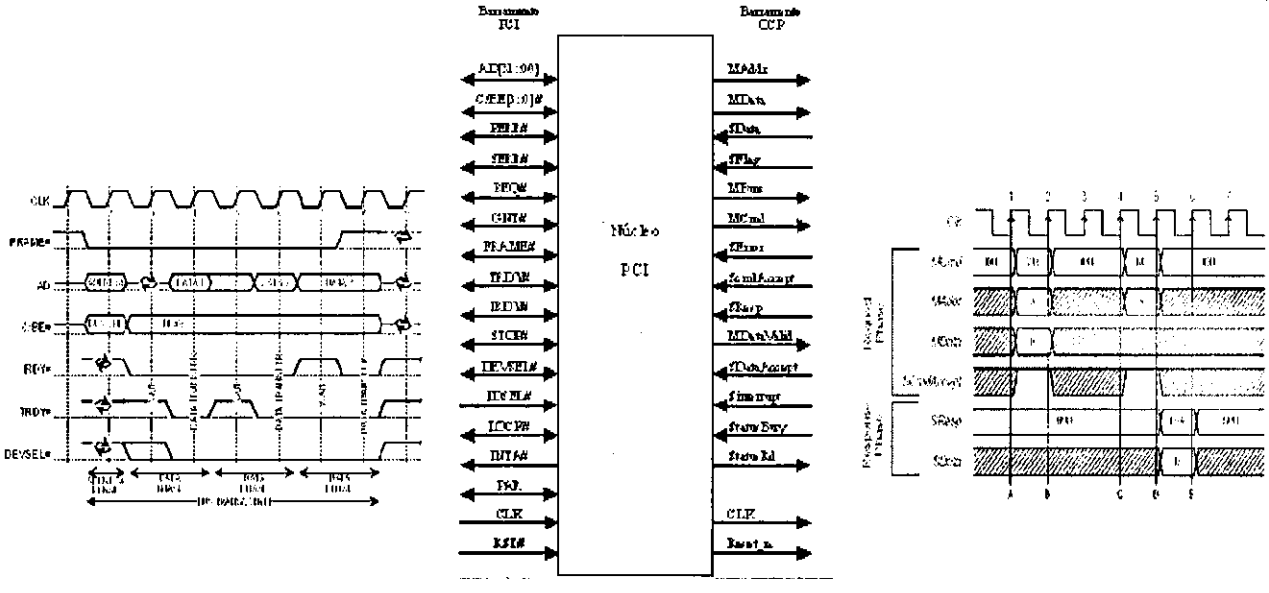


Figura 2.3 – Interface PCI – OCP, ciclos de leitura e escrita.

Na Tabela 2.1, tem-se uma relação simplificada das principais diferenças entre o barramento PCI e o OCP™.

Tabela 2.1 – Principais diferenças entre os barramentos PCI e OCP.

PCI	OCP
Barramentos bidirecionais (ciclo de inversão de barramento para leitura)	Barramentos unidirecionais (barramento de escrita e de leitura separados)
Barramentos multiplexados AD, C/BE	Barramentos simples (endereço, dados, comandos separados)
Níveis lógicos alto, baixo, terceiro estado e terceiro estado sustentado	Níveis lógicos alto, baixo
Tempo de acesso ao barramento crítico	Tempo de acesso ao barramento "seguro"

Na Figura 2.4 mostra-se a descrição dos sinais básicos usados na interface, de um lado estão presentes os sinais do barramento PCI Alvo e do outro os sinais provenientes da Aplicação de Dispositivo do Usuário, nesse caso interpretados pelo protocolo OCP-IP™, deixando totalmente "transparente" a interface com o barramento PCI para a aplicação do usuário, o que torna o projeto do usuário mais simples.

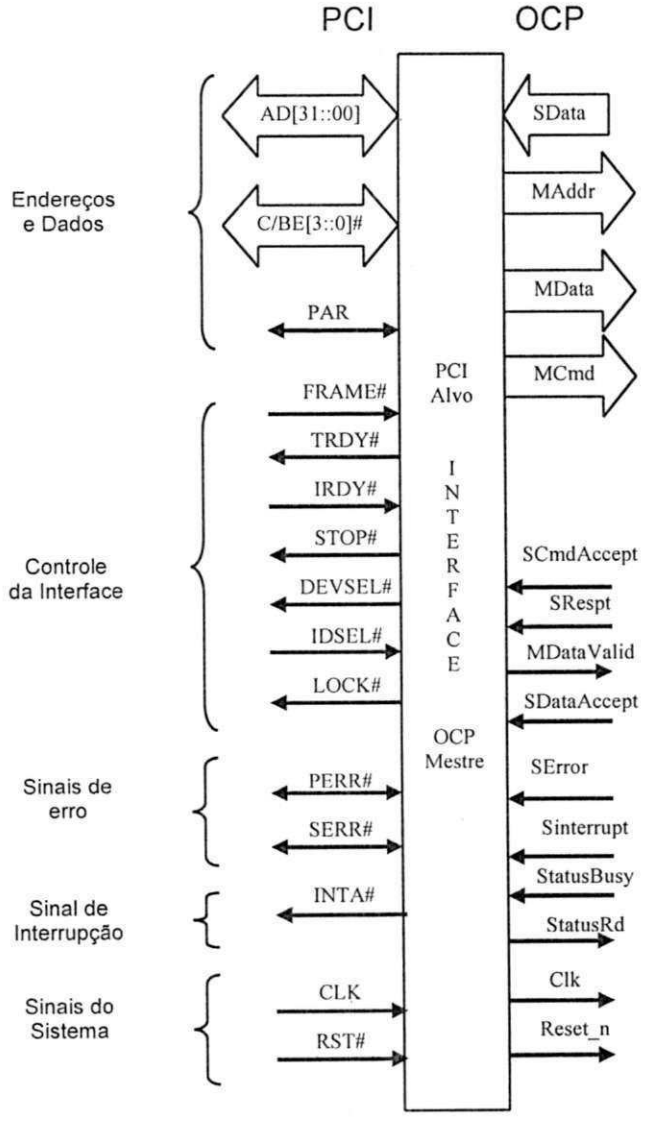


Figura 2.4 – Interface PCI OCP, sinais básicos.

A seguir, tem-se a descrição da padronização usada para os sinais:

→ Sinal é ativado em nível lógico 0 (NL0)

In → Entrada padrão

Out → Saída padrão ativa

t/s → Sinal bidirecional do tipo Alta Impedância[13]

o/d → *Open Drain* permite múltiplos dispositivos partilhar um único fio (*wire-OR*).

s/t/s → Alta Impedância Sustentado é um sinal ativo em nível lógico baixo mantido por um e apenas um agente por vez. O agente deve deixá-lo em nível lógico alto durante um ciclo de relógio antes de colocá-lo em Alta Impedância. Um novo agente não pode afirmar o sinal, antes que decorra um período de relógio, respeitando assim a duração da Alta Impedância.

Na Figura 2.5 mostra-se um detalhe do sinal do tipo Alta Impedância Sustentado.

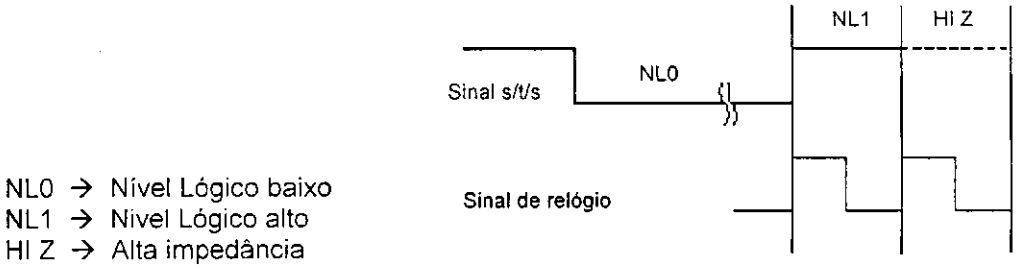


Figura 2.5 – Detalhe de um sinal do tipo Alta Impedância Sustentado[18].

[13] Nessa dissertação adota-se a expressão em português **Alta Impedância** em substituição à expressão inglesa ***Tri-State*** freqüentemente usada para designar um sinal em terceiro estado.

Na Tabela 2.2 estão descritos os sinais básicos do barramento PCI usados no projeto da interface nesta dissertação (Maiores detalhes estão disponíveis no Apêndice D).

Tabela 2.2 – Descrição dos sinais básicos do barramento PCI Alvo.

Sinal	Tipo	Descrição
AD[31::00]	t/s	**Endereços** e **Dados** são multiplexados. Uma transação de barramento consiste de uma fase de endereço seguida de uma ou mais fases de dados.
C/BE[3::0]#	t/s	**Comando** e **Dígitos de Habilitação** são multiplexados. Durante a fase de endereço os pinos definem o ciclo que irá ser executado e durante a fase de dados que octetos serão habilitados.
PAR	t/s	*Bit* de **paridade** par dos Sinais AD[31::00] e C/BE[3::0]#.
FRAME#	s/t/s	Ciclo de ***Frame*** [14] é colocado pelo dispositivo mestre indicando o início e o fim de uma transação.
IRDY#	s/t/s	**Iniciador Pronto** indica que o mestre do barramento está pronto para completar a atual fase de dados da transação.
TRDY#	s/t/s	**Destino Pronto** indica que o dispositivo selecionado está pronto para completar a atual fase de dados da transação.
STOP#	s/t/s	**Pare** indica que o Destino corrente está requisitando ao mestre que encerre a transação corrente.
DEVSEL#	s/t/s	O **Selecionador de dispositivo** quando ativado indica que o dispositivo decodificou seu endereço e se tornou o Destino da corrente transação.
IDSEL#	In	O **Selecionador de dispositivo de inicialização** é usado como seletor de placa durante as transações de leitura/escrita de configuração.
PERR#	s/t/s	**Erro de paridade** indica erros de paridade de dados exceto no Ciclo Especial de Comando.
SERR#	o/d	**Erro do Sistema** indica erros de paridade nos endereços e erro de paridade dos dados no Ciclo Especial de Comando.
CLK#	In	**Relógio** fornece a temporização para todas as transações do PCI. Com exceção dos sinais RST#, INTA#, INTB#, INTC# e INTD# os restantes são amostrados na borda de subida do sinal do CLK.
RST#	In	***Reset*** [15] é usado para colocar os registradores específicos do PCI, máquinas de estados e sinais em um estado consistente.
INTA#	o/d	**Interrupção A** é usada para colocar um pedido de interrupção (sensível a nível).

[14] Usa-se a palavra inglesa ***Frame*** para designar o sinal que indica o inicio e o término de uma transação, período que ocorre a transmissão de um pacote de dados incluindo informações de controle e rota, doravante em substituição a esta, adota-se nesta dissertação a palavra em português **Quadro**.

[15] Sinal de *Reset* é um sinal usado para restaurar, reiniciar; retornar o sistema a seu estado inicial, para permitir que um programa ou processo seja reiniciado. Doravante nessa dissertação será usada a palavra portuguesa **Reiniciar** em substituição à palavra inglesa ***Reset.***

A seguir, na Tabela 2.3, tem-se uma breve descrição dos sinais OCP usados nesta dissertação (Maiores detalhes são apresentados no Apêndice E).

Tabela 2.3 – Descrição dos sinais OCP.

Grupo	Sinal	Parâmetro para adicionar sinal para interface	Parâmetro para controle de largura	Controlador	Função
Básico	Clk	Exigido	Fixo 1	Varia	Relógio do OCP
	MAddr	Exigido	Faixa-dados 1-32	Mestre	Transfere endereço
	MCmd	Exigido	Fixo 3	Mestre	Transfere comando
	MData	Exigido	Faixa-dados 8/16/32/64/128	Mestre	Escreve dados
	SCmdAccept	Exigido	Fixo 1	Escravo	Aceita transferência
	SData	Exigido	Faixa-dados 8/16/32/64/128	Escravo	Lê dados
	SResp	Exigido	Fixo 2	Escravo	Transfere resposta
Simples	MBurst	Estouro	Fixo 3	Mestre	Código de Estouro
	MByteEn	Grupo de dígitos de Habilitação	Faixa-dados 1/2/4/8/16	Mestre	Dígitos ativos
	MDataValid	*datahandshake*	Fixo 1	Mestre	Escreve para dados válidos
	SDataAccept	*datahandshake*	Fixo 1	Escravo	Escravo aceita escreve dados
Banda Lateral	Reset_n	Reinicializar	Fixo 1	varia	Reinicializa Síncrono
	SError	Erro	Fixo 1	Escravo	Erro do escravo
	SInterrupt	Interrupção	Fixo 1	Escravo	Interrupção do escravo

O projeto da Interface elaborado nesta dissertação, fará uso do Espaço de Configuração da PCI do Tipo 0. A seguir, na Tabela 2.4, descreve-se os principais registradores utilizados.

Tabela 2.4 – Descrição dos Registradores de Configuração Espacial utilizados.

Device ID 0004		Vendor ID 1172		00h
Status 0400		Command 0002		04h
Class Code 058000			Revision ID 01	08h
BIST 00	Header Type 00	Latency Timer 00	Cache Line Siza 00	0Ch
Endereço Base				10h
Max Lat 00	Min Gnt 00	Interrupt Pin 01	Interrupt Line 0^A	3C

Registro 00h Identifica o fabricante e o modelo

Device ID → Identifica o dispositivo → 0004*

Ventor ID → Identifica o fabricante → 1172*

Registro 04h fornece o estado e o comando.

Status → 0400 → sinaliza erro de sistema.

Command → 0002 → Espaço de memória.

Registro 08h Identifica a classe.

Classe de base → 05 → Controlador de memória;

Sub-classe → 80 → Outra controladora de memória de massa;

Interface → 00 → Outra controladora de memória de massa;

Programação da Interface → 01 → Modo operacional primário

Registro 3C fornece o pino e a linha de interrupção

Int_Pin → 01 → pino de interrupção.

Command → 0A → linha de interrupção.

* Usado valor comercial da empresa Altera[10].

Nota: Descrições mais detalhadas dos registradores de configuração podem ser encontradas no documento *PCI Local BUS Speciification revision.2.2*[13] (páginas 189 - Capítulo 6 e página 257 no Anexo D). Nesta dissertação, pode-se verificar no Apêndice D sub-item D.2 - Operação Conectar e Usar, página 93.

2.2.3. Blocos de Funções

O núcleo PCI foi projetado para oferecer uma solução mínima que vá de encontro às necessidades de vários aplicativos diferentes a fim de garantir todos os requisitos de contagem de tempo do PCI.

O projeto foi dividido em dois blocos principais: Bloco Mutiplexador e Bloco de Controle. Veja a ilustração 2.6.

Bloco Multiplexador é responsável pelo transporte dos dados, endereços, comando e dígitos de habilitação[16]. Neste bloco é realizada a mutiplexação e demultiplexação entre o barramento PCI e OCP™, onde se realiza a separação dos dados e endereço provenientes do barramento PCI AD[31::00] para enviar ao barramento "MData", "SData" e "MAddr" no OCP™ e vice-versa.

Bloco de Controle é responsável por todo o controle da interface, realizando o acoplamento entre os sinais PCI e OCP™. As principais funções deste bloco são: Comando, Interface, Paridade, Erro, Interrupção e Sistema.

- **Comando** é responsável pela conversão do comando PCI C/BE[3::0] para os comandos OCP (MCmd[2::0]).
- **Interface** é responsável pelo processamento de todos sinais de troca de informação entre PCI e OCP™.
- **Paridade** e **Erro** realiza a verificação da paridade nos dados e endereço e sinaliza quando houver erro.
- **Interrupção** sinaliza o pedido de interrupção do OCP™ para o PCI.
- **Sistema** envia os sinais de relógio e de reinicializar provenientes do PCI para o OCP™.

[16] Em substituição a expressão inglesa "***Byte Enable***" para designar um conjunto de 4 ou 8 *bits* de habilitação do barramento PCI de largura 32 ou 64 *bits*, respectivamente. Nesta dissertação usou-se expressão portuguesa "**Dígitos de Habilitação**", que se aplica tanto para 32 *bits* "**Quarteto de Habilitação**" como no caso de 64 *bits* "**Octeto de Habilitação**".

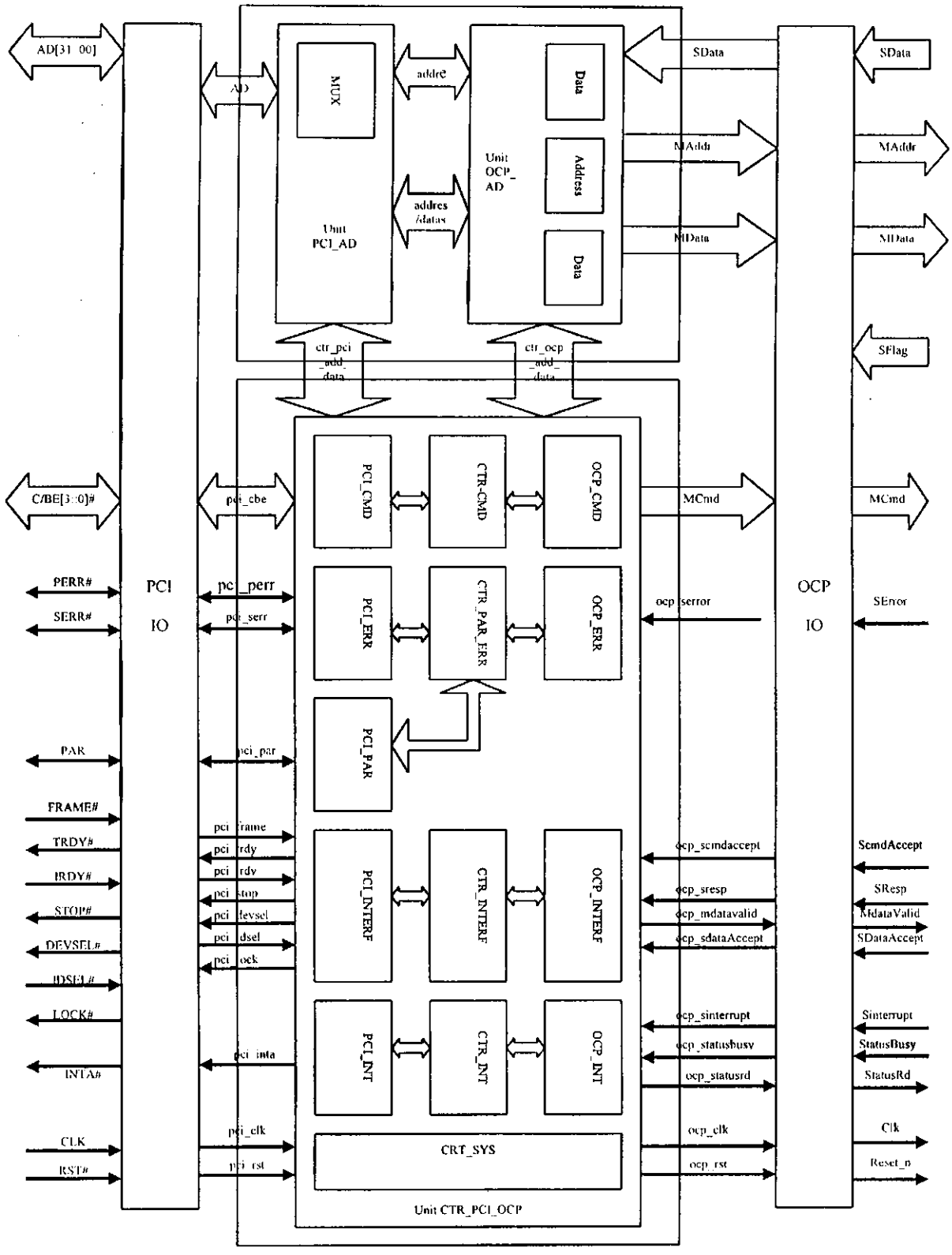


Figura 2.6 – Diagrama em bloco da interface PCI/OCP.

2.2.4. A Placa FPGA

A placa de desenvolvimento usada é a FLEX 10KE PCI, da empresa Altera[10], ilustrada na Figura 2.6.

As principais características da placa são descritas a seguir:

- Cartão de expansão PCI universal de 64 bits e 66 MHz.
- Módulo DIMM SDRAM de 32 MB com 144 pinos pequenos de saída na placa.
- Conector na placa é do padrão PMC (*PCI Mezzanine Card*).
- Possui uma área de prototipagem de E/S.
- Porta RS-232.
- Regulador de tensão gera 2.5 V, 3.3 V e 5.0 V.
- Portas típicas (Lógicas e RAM) 100.000.
- Elementos Lógicos (LE) 4.992.
- Pinos de entrada e saída (E/S) para usuários, máximo de 338.
- Total de RAM *bits* 49.152.

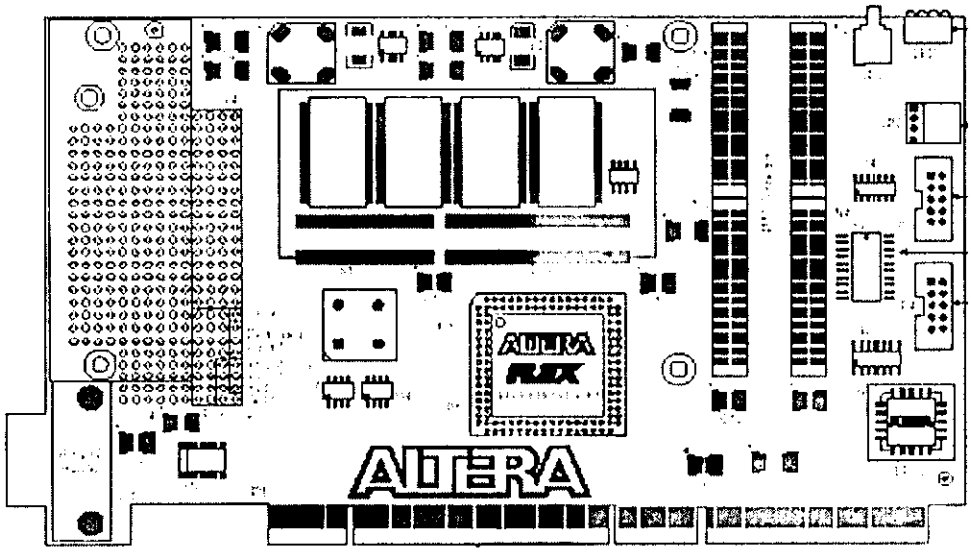


Figura 2.7 – Diagrama da placa FPGA – FLEX 10KE[10].

Nota: Informações suplementares podem ser obtidas nas publicações da ALTERA [10 -12].

3. CAPÍTULO 3 – ESTRUTURA DA INTERFACE

O projeto da Interface foi dividido em duas partes principais: o bloco Mutiplexador e bloco de Controle. Neste Capítulo descreve-se mais detalhadamente cada bloco.

3.1. Bloco Multiplexador

O bloco multiplexador (Figura 3.1) é responsável pelo transporte dos dados e endereços, neste bloco é realizada a mutiplexação e demultiplexação entre o barramento PCI e o barramento OCP™.

A principal função deste bloco é realizar a separação dos Dados e Endereço provenientes do barramento PCI AD[31::00] para enviá-los ao barramento "MData" "SData" e "MAddr" no OCP e vice-versa. Outra função deste bloco é fornecer os sinais MCmd e o MByteEn ao OCP, sinais estes provenientes do barramento de Comando e Dígitos de Habilitação C/BE[3::0] no PCI que foi previamente convertido para sinal de comando do OCP™ no bloco de comando. Por fim, este bloco garante o acesso aos registradores de configuração, usados na configuração espacial da PCI.

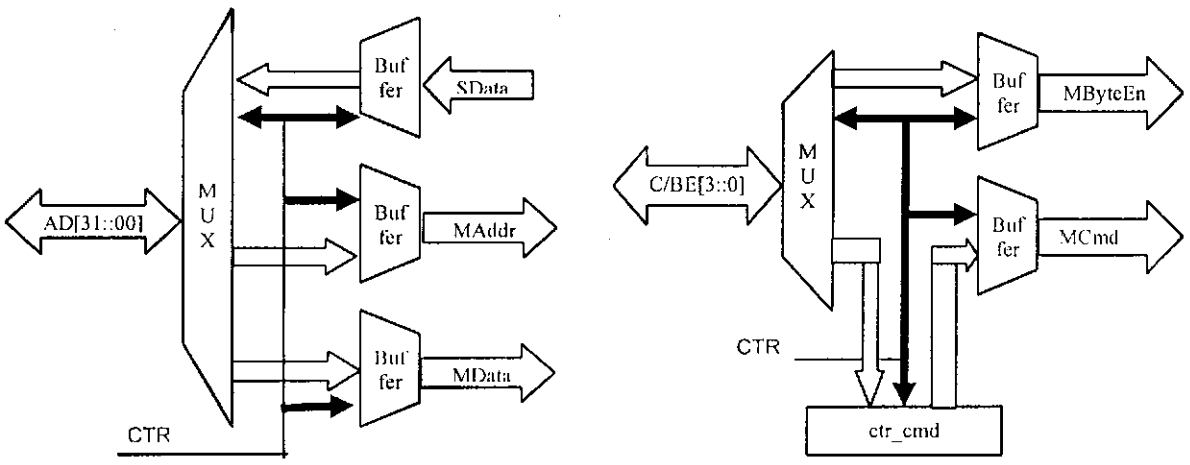


Figura 3.1 – Diagrama do Mux_AD e Mux_C/BE.

3.1.1. Descrição do funcionamento

Ciclo de Escrita e Leitura de Configuração

O ciclo de escrita e leitura de configuração é ativado pelo comando IDSEL, que atua como um seletor de dispositivo na inicialização do sistema[17]. Nesta etapa é realizada a leitura e a escrita nos registros de configuração. No primeiro ciclo de relógio o endereço do registro e o comando são enviados à interface que realiza imediatamente a codificação, recebendo os dados ou enviando-os em um ou dois ciclos, respectivamente.

Maiores detalhes sobre o ciclo de configuração da PCI estão expostos nos gráficos do Capítulo 6 e no Apêndice D - Registradores de Configuração.

Ciclo de Escrita e Leitura na Memória

No primeiro ciclo de relógio o endereço é armazenado em registradores internos enquanto o comando é decodificado pela **unidade de comando** para ser enviado posteriormente a este bloco que fará sua transferência juntamente com os dados e os dígitos de habilitação.

No caso do comando ser uma escrita na memória, no segundo ciclo de relógio é realizada a transferência dos dados e dos dígitos de habilitação para os registradores internos. No ciclo seguinte a transação com a PCI é terminada pelo bloco de controle. Neste momento o comando, os dígitos de habilitação, endereço e dados são repassados aos respectivos destinos no OCP, o MCmd, MByteEn, MAddr e MDtata.

No caso de ser um ciclo de leitura, tem-se uma operação um pouco mais complexa, pois é necessário manter comunicação com a PCI e OCP™ simultaneamente (PCI fica aguardando término de comunicação com OCP).

O primeiro ciclo de relógio é idêntico ao da escrita, no segundo ciclo o bloco de controle mantém a PCI em espera, enquanto o bloco multiplexador recebe os dígitos de habilitação, para só então, realizar a transferência do comando, dos dígitos de

habilitação e endereço para OCP™. Novamente o bloco de controle assume e fica aguardando o aceite do comando OCP (SCmdAccept) e o de dados válidos (SResp). Recebidos esses sinais o bloco de controle comunica a PCI para que somente neste instante o bloco multiplexador possa realizar a transferência dos dados do barramento SDtata no OCP™ para o barramento AD no PCI para então o bloco de controle finalizar a transação com a PCI.

Nota: No Capítulo 6 apresentam-se gráficos detalhados dessas transações de acesso ao barramento.

3.2. Bloco de Controle

O Bloco de Controle é responsável por todo o controle da interface, realizando a sincronização, o acoplamento e conversão entre os sinais PCI e OCP™. As princ ais unidades funcionais deste bloco são: Comando, Interface, Paridade, Erro, Interrupção e Sistema.

3.2.1. Unidade de Comando

A unidade de comando é responsável pela conversão do sinal C/BE de comando PCI para o sinal MCmd de comando OCP™, a conversão é realizada simultaneamente com a etapa de separação dos sinais no bloco multiplexador para entrega posterior do comando OCP ao bloco multiplexador que deverá enviá-lo ao barramento MCmd no OCP™ juntamente com os outros sinais daquele bloco.

Na Tabela 3.1 tem-se a relação usada para conversão dos comandos C/BE, provenientes do barramento PCI, em MCmd, enviados ao OCP™. Em destaque, os comandos implementados nesta dissertação.

[17] É comum o uso das expressões "***Boot time***", "***Power Up***" ou "***Power On Reset***" da língua inglesa para designar a etapa para inicializar o sistema.

Tabela 3.1 – Conversão dos comandos C/BE PCI para MCmd OCP.

Tipo de Comando PCI	C/BE[3::0]	MCmd	Tipo de Comando OCP
Reconhece Interrupção	0000	010	Lê*
Ciclo Especial	0001	000	INATIVO
Lê E/S	0010	010	Lê*
Escreve E/S	0011	001	Escreve*
Reservado	0100	000	INATIVO
Reservado	0101	000	INATIVO
Lê Memória	**0110**	**010**	**Lê**
Escreve Memória	**0111**	**001**	**Escreve**
Reservado	1000	000	INATIVO
Reservado	1001	000	INATIVO
Lê Configuração	**1010**	**010**	**Lê**
Escreve Configuração	**1011**	**001**	**Escreve**
Leitura Múltipla na Memória	1100	011	Leitura Múltipla*
Ciclo de Endereço Dual	1101	000	INATIVO
Leitura na Memória em Linha	1110	000	INATIVO
Escrita na Memória e Invalido	1111	000	INATIVO

* não implementados, permanecendo em INATIVO.

3.2.2. Unidade de Interface

A Unidade de Interface é responsável pelo processamento de todos os sinais que habilitam o controle para a troca de informações entre a PCI e o OCP™. Nesta unidade pode-se destacar os seguintes sinais:

- Sinais PCI: FRAME, TRDY, IRDY, STOP, DEVSEL, IDSEL.
- Sinais OCP™: CmdAccept, SResp, MDataValid, SDataAccept, MRespAccept

A seguir, tem-se uma seqüência de tabelas nos quais se ilustram as relações usadas para o acoplamento entre os sinais PCI e OCP™.

Na Tabela 3.2 pode-se verificar a dependência da saída do sinal TRDY da PCI com sinais de entrada na interface provenientes de comandos do OCP™. A interface só pode estar pronta para responder a PCI (TRDY = 0) se obteve resposta afirmativa do OCP™ (SCmdAccept e SResp forem ativados = 1), isto para o ciclo de leitura.

Tabela 3.2 – Sinal TRDY.

Entrada		Saída
Sresp	SCmdAccept	TRDY
DVA (01)	1	0

TRDY = 0 se {(SResp = DVA) **E** (SCmdAccept = 1)]}

DVA → Dados Validos / Aceite

Ainda no ciclo de leitura, tem-se na Tabela 3.3 a seguinte relação: Quando os dados estiverem disponíveis no barramento de saída do OCP™ (SData), esta unidade responderá com o comando de dados válidos (SResp [DVA]), a interface recebe o comando, e estando pronta, sinaliza que os dados foram aceitos (MrespAccept) disponibilizando-os à PCI se esta estiver pronta para receber (IRDY ativo [0])

Tabela 3.3 – Sinal MrespAccept.

Entrada		Saída
IRDY	SResp	MRespAccept
0	DVA (01)	1

MRespAccept = 1, se {(IRDY = 0) **E** (SResp = DVA)}

No ciclo de escrita verifica-se a relação para o sinal MDataValid do OCP™ que está mostrado da Tabela 3.4. Após a PCI disponibilizar os dados (IRDY = 0) a interface envia o comando (MCmd) e o endereço (MAddr), para tornar ativo o sinal (MDataValid) indicando que os dados estão disponíveis no barramento (MData) e fica aguardando o aceite dos dados por parte do OCP™ (SDataAccept).

Tabela 3.4 – Sinal MdataValid.

Entrada		Saída
IRDY	MCmd	out_MdataValid
0	Escrita(001)	1

MDataValid = 1 se {(IRDY = 0) **E** (MCmd = 001)}

Na Tabela 3.5 tem-se o sinal de STOP que poderá ser afirmado em duas condições: A primeira é um artifício temporário, pois visa o impedimento do ciclo rajada após os dados serem disponibilizados finalizando a transação, a segunda forma de ser ativado é na ocorrência de um erro por parte do dispositivo Destino no OCP™ que sinaliza com o comando de SError encerrando a transação.

Tabela 3.5 – Sinal STOP.

Entrada	Saída
SError	STOP
0	1
1	0

STOP = **NÃO** in_SError

Finalizando a Unidade de Interface tem-se a descrição do sinal DEVSEL, este sinal quando afirmado (DEVSEL = 0) sinaliza a PCI que decodificou o endereço e reconhece que a transação é com ela. Para ativá-lo é necessário a verificação de alguns parâmetros como o sinal de FRAME, que indica o início e o término de uma transação, o *bit* de acesso à memória no registrador *COMMAND* e a faixa de endereço se é válida, ou seja, compreendida pelo endereço base e o tamanho de memória requerido pela interface.

Tabela 3.6 – Sinal DEVSEL.

Entrada			Saída
FRAME	Bit 1 de COMMAND	Endereço na faixa válida	DEVSEL
0	[xxxxxxxxxxxxxx1x]	De [endereço base] até [endereço base + tamanho da memória requerido]	0

DEVSEL = 0 se {(frame=0) **E** (*Command* b1 = 1) **E** (endereço na faixa válida)

Nota: Maiores detalhes do funcionamento dos sinais poderão ser vistos nos gráficos do Capítulo 6.

3.2.3. Unidades de Paridade e Erro

Na Unidade de Paridade e Erro, são realizadas as verificações da paridade no barramento de Dados/Endereço (AD) e no barramento Comando/Dígitos de Habilitação (C/BE). A paridade é calculada em todos os ciclos de operação da PCI não importa o tipo de transação. A geração de paridade não é opcional e deve ser feita por todos os dispositivos compatíveis com o padrão PCI.

Nos ciclos de escrita de endereçamento e de dados, no ciclo relógio seguinte ao da informação, a PCI informa através do sinal de PAR o valor da paridade (número de 1s par) dos barramentos AD e C/BE. Após envio do sinal de PAR fica aguardando a resposta de confirmação, no ciclo de relógio seguinte, se houve erro ou não na transmissão através da informação nos sinais de PERR (erro de dados exceto no ciclo especial de comando) e SERR (erro de endereçamento e de dados no ciclo de sistema).

Para o ciclo de leitura, tem-se a operação inversa, ou seja, a PCI recebe a informação da paridade do barramento de endereço e dados, através do sinal de PAR, um ciclo após o endereço e os dados estarem disponíveis no barramento. Calcula a paridade e devolve a resposta nos sinais de PERR ou SERR um ciclo depois.

Nesta etapa, a unidade de paridade e erro realiza uma função redução ou-exclusivo (xor_reduce()) nos barramentos AD e C/BE para obter o *bit* de paridade de cada barramento, em seguida usa-se a função ou-exclusivo para cálculo da paridade.

paridade_AD = AD_**xor_reduce()**

paridade_CBE = CBE_**xor_reduce()**

paridade = (paridade_AD **xor** paridade_CBE)

Tabela 3.7 – Sinal PERR.

Entrada		Saída	Saída
PAR	paridade	SERR	PERR
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

PERR = PAR **XOR** paridade SERR = PAR **XOR** paridade

Nota 1: A função de redução ou-exclusivo (xor_reduce()) é facilmente implementada na compilação, entretanto a maioria dos sintetizadores não consegue executá-la, devendo-se utilizar outros artifícios para cálculo da paridade.
Nota 2: O sinal SERR é usado no ciclo Especial de Configuração, não foi implementado nesta dissertação.
Nota 3: Para maiores detalhes sobre o funcionamento do mecanismo de paridade veja publicação *PCI Local BUS Specification revision .2.2, cap.3.7.1 - Parity Generation* na página 94 [13].

3.2.4. Unidade de Interrupção

A Unidade de Interrupção somente realiza o acoplamento do pedido de interrupção proveniente do protocolo OCP™ para o barramento PCI, não realiza nenhum tratamento da interrupção, ficando por conta da interface PCI e da Aplicação do Usuário o tratamento dado ao pedido de interrupção.

A interrupção da interface PCI usada nesta dissertação é a INTA, sendo sua configuração realizada durante o ciclo de configuração inicial, registro de endereço 3Ch [*Interrupt Line, Interrup Pin*].

Após ser sinalizada a interrupção (Sinterrupt = 1,) a interface PCI envia em um ciclo de configuração o comando de Aceite da Interrupção (C/BE = 0000).

A relação dos sinais Sinterrupt e INTA estão expostos na Tabela 3.8.

Tabela 3.8 – Sinal INTA.

Entrada	Saída
SInterrupt	INTA
0	1
1	0

INTA = **NOT** SInterrupt

Nota 1: Não foi implementado o tratamento dado à interrupção, finalizando esta etapa.
Nota 2: O pedido de interrupção é sensível ao nível, sendo ativo na interface PCI em nível lógico baixo.

3.2.5. Unidade de Sistema

Na unidade de sistema são implementados dois sinais: o sinal de relógio CLK e o sinal para reinicializar RST.

O sinal de relógio CLK (Tabela 3.9) é um dos pontos críticos da interface, sendo fixado o mesmo sinal para o barramento PCI e OCP™ visando a simplificação da interface. Porém, o ideal seria que os sinais de relógio da interface PCI e da OCP™ fossem independentes, dando maior flexibilidade à implementação das Aplicações de Dispositivo do Usuário.

Tabela 3.9 – Sinal pci_clk e ocp_clk.

Entrada	Saída	
CLK	pci_clk	ocp_Clk
0	0	0
1	1	1

pci_clk = CLK — relógio da Interface

ocp_clk = CLK — relógio repassado ao OCP™

O sinal de Reset_n (Tabela 3.10 - *Reset enable*) gera um estado consistente para reinicializar a Interface após o sinal de RST da PCI ser ativado. Nesta etapa existe um travamento[18] do sinal para reiniciar o Aplicativo de Dispositivo do Usuário, evitando interpretação errônea por parte deste, caso o sinal de reiniciar (RST) no barramento PCI seja de pequena duração (poucos ciclos de relógio).

Tabela 3.10 – Sinal Reset_n.

Entrada	Saída
RST	Reset_n
0	0
1	1

Reset_n = RST

[18] Este sinal é mantido ativado por 16 ciclos de relógio, veja detalhes no gráfico *Reset* do Capítulo 6.

4. CAPÍTULO 4 – DESCRIÇÃO DO PROGRAMA

Conforme mencionado na introdução desta dissertação a Linguagem de Descrição de Dispositivo usada para desenvolvimento do programa foi o *SystemC*™.

O *SystemC*™ é uma linguagem de alto nível baseada na linguagem C++, tendo como um dos objetivos padronizar o uso de uma linguagem única nas etapas mais críticas do processo de projeto de um núcleo IP, passando pelas etapas de especificação, descrição comportamental, descrição RTL e descrição estrutural, não sendo necessário ao projetista o conhecimento de outras linguagens para conclusão de etapas intermediárias ao processo e nem na construção de um **Ambiente de Teste por Simulação** e de uma **Aplicação de Dispositivo do Usuário**.

Na Figura 4.1, tem-se uma apresentação dos principais blocos que compõem o programa. Neste Capítulo tem-se uma descrição resumida de parte do programa (apenas o trecho que foi implementado em FPGA). A listagem na íntegra do código em *SystemC*™ está disponível no CDROM anexo a este documento na biblioteca miniblio do bloco CG/DEE/UFCG.

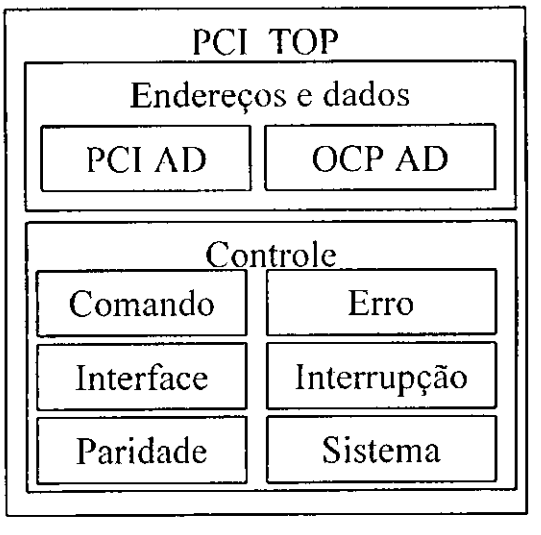


Figura 4.1 – Diagrama em Bloco do Programa.

4.1. Estrutura do Programa Implementado em FPGA

Nesta Dissertação, foi implementada apenas uma parte do código do programa em FPGA, para isto, houve mudança parcial da estrutura do programa, ficando organizada com segue:

Main.cpp
- controle.h
  - var_h
  - pci_io.h
  - ocp_io.h
  - count_clk.h
    - count_cyc
  - statemach.h
    - statemach
    - reset.h
    - reg_config.h
  - mux_ad.h
    - mux_add_cmd_rd
    - mux_data_be_rd
    - mux_add_cmd_wr
    - mux_data_be_wr
    - muxcfg_add_cmd_rd
    - muxcfg_data_be_rd
    - muxcfg_add_cmd_wr
    - muxcfg_data_be_wr
  - sys_clock
  - sys_reset
  - ctr_command
    - command.h
  - ctr_int
  - interf_frame
  - interf_irdy
  - interf_trdy

interf_idsel
interf_devsel
interf_stop
interf_mrespaccept
interf_mdatavalid
ctr_par
parity.h
master.h
slave.h

4.2. Descrição dos Arquivos do Programa

A seguir tem-se uma breve descrição dos arquivos do programa.

Arquivo main.cpp

O módulo, main.cpp contém a função sc_main() e realiza a ligação entre os módulos **control.h**, **master.h** e **slave.h**.

Arquivo control.h

O arquivo control.h contém o módulo *control*, onde se encontram inseridos todos os outros arquivos. Pode-se dividir esse módulo nos procedimentos: declaração de variáveis; declaração dos sinais PCI; declaração dos sinais OCP^{TM}; contador de ciclos de relógio; máquina de estado; sinais de sistema, comando, interface, interrupção, paridade e multiplexador.

Descrição dos arquivos que compõem o módulo control.h:

var.h

O arquivo var.h contém a declaração de todas as variáveis do tipo global do sistema.

pci_io.h

O arquivo pci_io.h contém a declaração de todos os sinais de entrada e saída para a interface PCI.

ocp_io.h

O arquivo ocp_io.h contém a declaração de todos os sinais de entrada e saída para a interface OCP-IP™.

count_clk.h

Neste arquivo tem-se a execução do processo **count_cyc** que é responsável pela contagem de tempo durante a realização de uma transação com a interface.

statemach.h

O arquivo statemach.h corresponde à máquina de estado, que controla o estado corrente da interface e aponta sempre para o próximo estado onde deverá ir a interface. É neste processo que se inicializa o arquivo de ***reset.h*** que contém os valores iniciais para todos os sinais e variáveis da interface, inclui também o arquivo ***reg_config.h,*** que contém os registros de configuração com os valores iniciais para o ciclo de configuração da interface PCI.

mux_ad

No arquivo mux_ad tem-se os processos de multiplexação do barramento AD e C/BE com os barramentos MAddr, MData, MCmd e MByteEn,

mux_add_cmd_rd a PCI entrega o endereço e o comando a interface no ciclo de leitura.

mux_data_be_rd a interface recebe os dígitos de habilitação da interface PCI e os dados do OCP-IP™ no ciclo de leitura.

mux_add_cmd_wr a interface PCI entrega o endereço e o comando a interface no ciclo de escrita.

mux_data_be_wr a interface PCI entrega os dados e os dígitos de habilitação à interface no ciclo de escrita.

muxcfg_add_cmd_rd a interface PCI entrega o endereço do registro e o comando à interface no ciclo de leitura de configuração.

muxcfg_data_be_rd a interface recebe os dígitos de habilitação da PCI e devolve os dados do registro de configuração.

muxcfg_add_cmd_wr a interface PCI entrega o endereço do registro e o comando à interface no ciclo de escrita de configuração.

muxcfg_data_be_wr a interface PCI entrega os dados a serem gravados nos registradores de configuração e os dígitos de habilitação à interface no ciclo de escrita de configuração.

sys_clock

Processo responsável pelo recebimento do sinal de relógio da interface PCI e repasse do sinal para a interface OCP-IP™.

sys_reset

O processo sys_reset garante a contagem de 16 ciclos de relógio ao OCP-IP™ para o comando Reset_n depois de ter recebido o sinal de RST da interface PCI.

ctr_command

O processo ctr_command executa o arquivo command.h, responsável pela conversão do comando C/BE da interface PCI em comando MCmd do protocolo OCP-IP™.

ctr_int

No processo ctr_int, tem-se o recebimento do comando SIterrupt e a saída do sinal INTA.

interf_frame

No processo interf_frame é feita a verificação do sinal de FRAME quando for ativado (nível lógico 0), dando início a uma transação.

interf_irdy

O processo interf_irdy verifica o sinal IRDY passando a informação à máquina de estado.

interf_trdy

O processo interf_trdy ativa o sinal TRDY que é dependente de vários sinais e do estado corrente da máquina de estado.

interf_idsel

O processo interf_idsel verifica quando for ativado [nível lógico alto] o sinal IDSEL identificando que a PCI deseja realizar um ciclo de configuração com a interface.

interf_devsel

No processo interf_devsel é verificada a validade da transação, ou seja, o reconhecimento de que a transação foi aceita. A PCI depois de afirmado o sinal de FRAME e colocado o endereço AD[31::00] e o comando C/BE[3::0] fica aguardando o reconhecimento da interface, que verifica esses sinais e a validade do endereço (se encontra dentro de faixa a qual foi destinada a interface) e se o *bit* 1 no registro de COMMAND está em 1, para só então ativar o sinal de DEVSEL (nível lógico baixo).

interf_stop

O sinal de STOP é ativado (nível lógico baixo) neste processo se receber um sinal de erro do OCP (SError) e estiver executando uma transação com a PCI (DEVSEL ativo).

Nota: Para evitar o modo de rajada, o sinal de STOP, torna-se ativo após transmissão do primeiro dado.

interf_mrespaccept

O processo interf_mrespaccept ativa o sinal de MRespAccept (nível lógico alto) depois da interface ter recebido o sinal Sresp, confirmando que os dados estão prontos para serem lidos no barramento SData.

interf_mdatavalid

O processo interf_mdatavalid, embora faça parte dos sinais da interface, o sinal nele contido, o MDataValid, é ativado no interior da máquina de estado, pois é dependente de uma resposta da PCI que é verificada na máquina de estados.

ctr_par

O processo ctr_par, que exerce o controle de paridade, executa o arquivo **parity.h,** neste arquivo encontra-se a implementação do cálculo de paridade e o acionamento dos sinais PERR e SERR que devem ser enviados à PCI como resposta à sinalização de paridade.

master.h

O arquivo *master.h* contém o módulo *master* que foi implementado apenas como módulo que representa a PCI, gerando os estímulos para verificação funcional do programa. Este módulo contém uma série de transações PCI, ou seja, transações de acesso de configuração de leitura e escrita, acesso à memória de leitura e escrita e da transação de reiniciar.

slave.h

O arquivo *slave.h* contém o módulo *slave* que foi implementado apenas como módulo que representa a interfane escravo, que faz uso do protocolo OCP-IP™, verificando os estímulos de saída da interface e gerando os estímulos de resposta para verificação funcional do programa.

5. CAPÍTULO 5 – IMPLEMENTAÇÃO

Neste Capítulo descreve-se o ambiente de desenvolvimento e as etapas necessárias à implementação do trabalho.

5.1. Ambiente de Desenvolvimento

Como ambiente de desenvolvimento teve-se a plataforma de trabalho para teste e depuração dos programas e dispositivos, o Laboratório de Arquiteturas Dedicadas do Departamento de Sistemas e Computação (LAD/DSC) e o Laboratório de Instrumentação Inteligente e Metrologia Científica do Departamento de Engenharia Elétrica (LIMC/DEE). Nestes locais disponibilizou-se uma placa PCI baseada em FPGA (*FLEX 10KE da Altera*) para prototipagem, um osciloscópio (*Tektronix TDS 220*) e dois computadores padrão IBM PC.

Um computador ficou com a função de Estação de Desenvolvimento na qual foram instalados os programas. O segundo computador teve a função de Estação de Produção onde ficou instalada a placa PCI com o Núcleo IP desenvolvido. Foram utilizados três sistemas operacionais:

- *LINUX REDHAT 7.2* – Instaladas as ferramentas de edição XEMACS, o compilador *SystemC™* e o conversor para *Verilog* da *Synopsys* versão 2003.03, programa para verificação lógica funcional e para o controle de versões o *Subversion* (*SVN*)[04].

- *MS Windows XP Professional* – este sistema permite o acesso aos recursos de dispositivo no ambiente do Windows. Dando suporte aos programas para síntese lógica (*Leonardo Spectrum* v20001b_106), o programa de posicionamento, roteamento e programação do FPGA (*Quartus* II v1.1 da *Altera*), para acessar os registradores das placas no barramento PCI foi utilizado o programa (*DiverWizard v 5.04 jun/2002)* e para visualizar e

capturar imagem do osciloscópio no computador utilizou-se o programa *WaveStar* do fabricante *Tektronix*.

- *MS-DOS Extender* – permite acesso ao modo protegido do processador, em *SystemC*, sem considerar o SO e o *device driver* da placa.

Nas Figuras 5.1 e 5.2, ilustram-se o ambiente de desenvolvimento, fotos no Laboratório de Arquiteturas Dedicadas do Departamento de Sistemas e Computação (LAD/DSC) .

Figura 5.1 – Ambiente de desenvolvimento, teste e depuração[18].

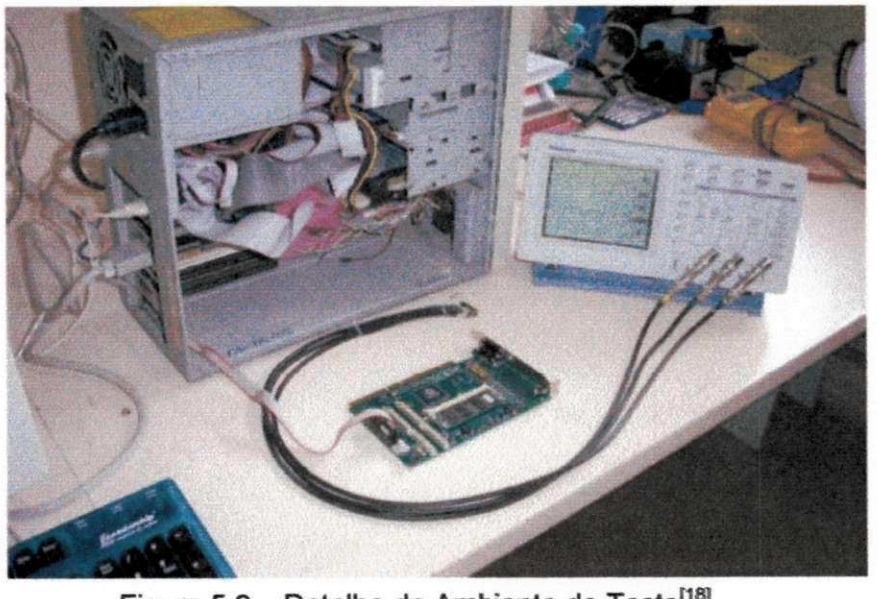


Figura 5.2 – Detalhe do Ambiente de Teste[18].

Na Figura 5.3 tem-se em detalhe a placa FPGA usada nesta dissertação.

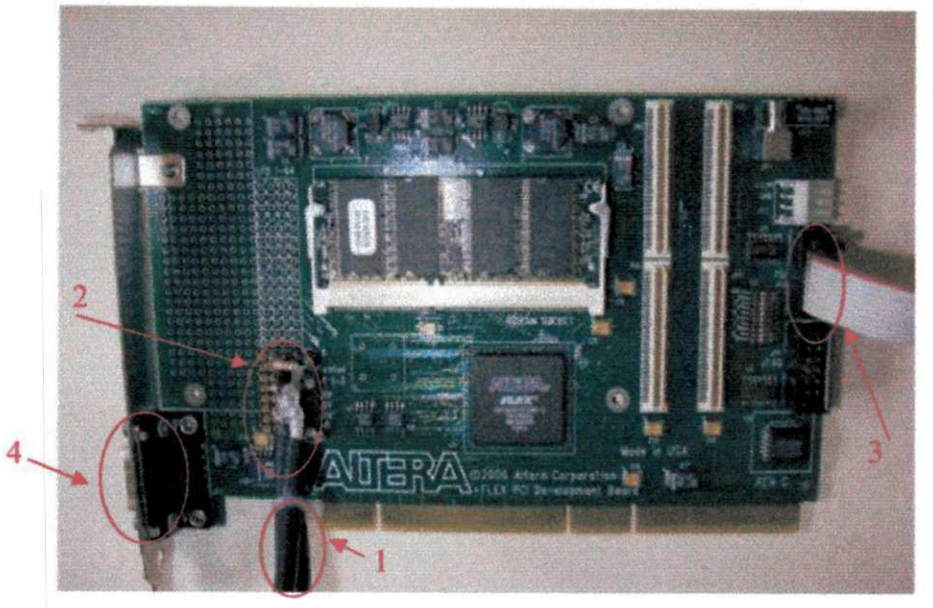


Figura 5.3 – Placa FLEX10KE da Altera[18].

Figura 5.3 (1), detalhe da montagem da ponta de prova: Cabo coaxial 50 Ω, resistores de 1 kΩ, resistores de 50 Ω, conectores BNC e terminadores (cargas).

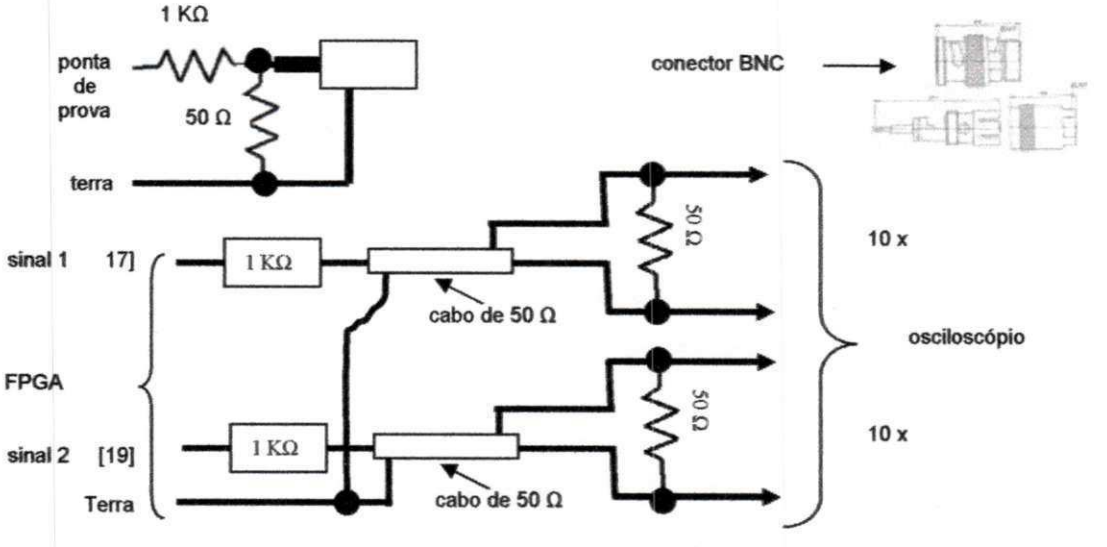


Figura 5.4 – Montagem da Ponta de Prova.

Na Figura 5.3 (2), detalhe da conexão da ponta de prova com área de entrada e saída da placa FPGA. Maiores detalhes da descrição dos pinos no manual da placa (EPF 10K100C484-1)[10].

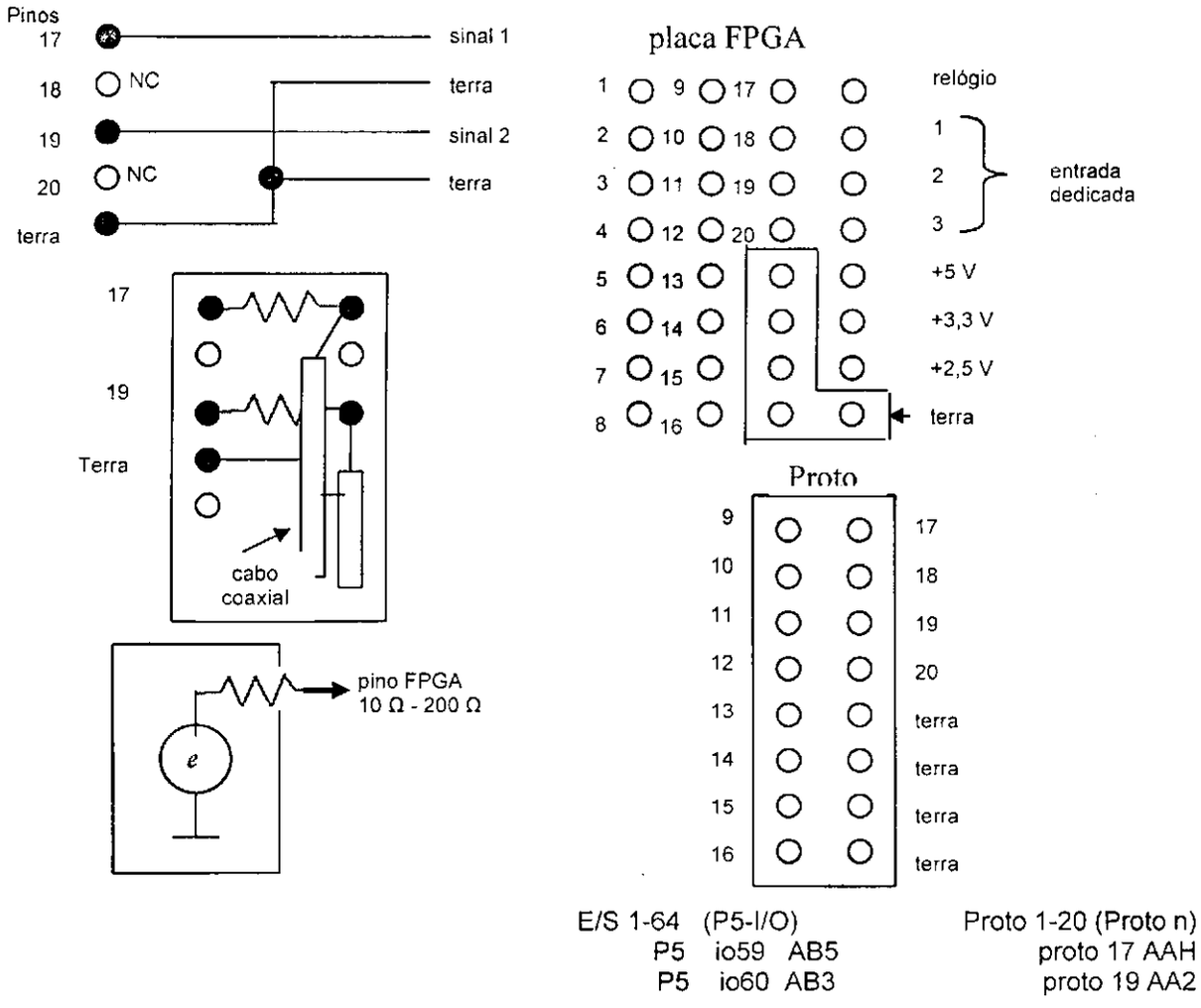


Figura 5.5 – Montagem da conexão da ponta de prova com a FPGA.

Figura 5.3 (3), detalhe do cabo para conexão da FPGA com o computador, usado para programação da placa, pode ser feita via cabo com conexão JTAG ou cabo PS (*Passive Serial*), sendo usado nesta dissertação o PS.

Figura 5.3 (4), detalhe da conexão do osciloscópio com o computador é feita pela porta ***COM 1*** via canal serial RS 232 com o conector DB 9 (cabo TDS 220).

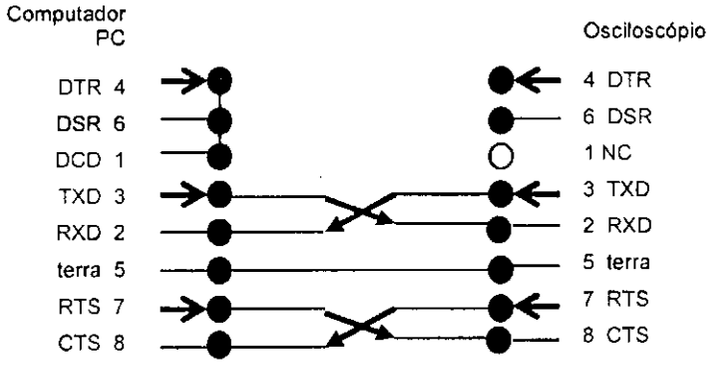


Figura 5.6 – Conexão Computador Osciloscópio - Cabo TDS 2CM.

5.2. Etapas de Implementação

As etapas necessárias à implementação do sistema são descritas a seguir.

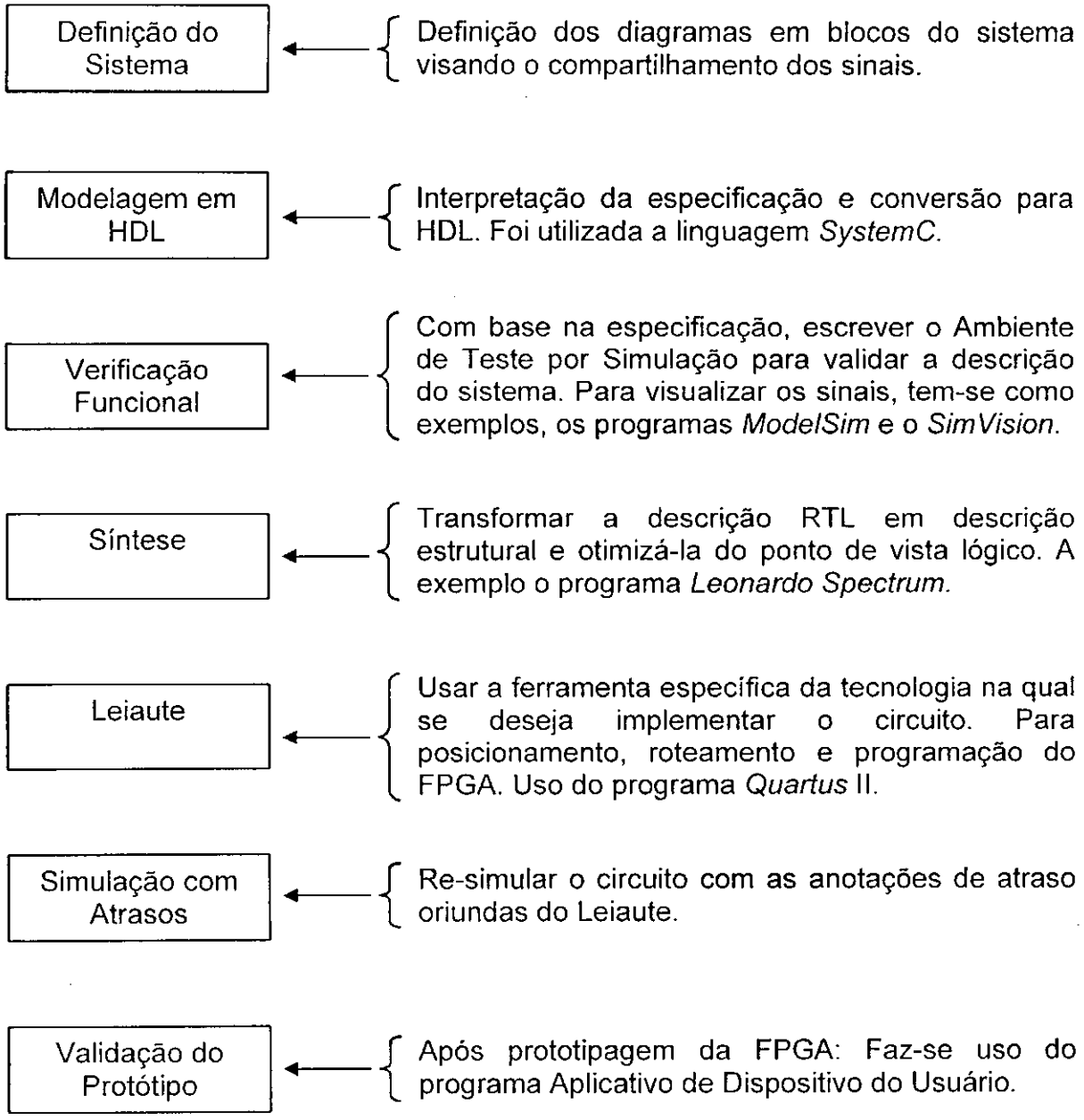


Figura 5.7 – Diagrama do fluxo de trabalho.

5.2.1. Definição do Sistema

Durante essa etapa de definição do sistema, foram estudadas algumas interfaces comerciais, para poder se ter um ponto inicial de referência e um apoio posterior para comparação com o modelo proposto. Uma dificuldade extra neste ponto foi o fato de não encontrar disponível nenhum trabalho referente à interface PCI-OCP. Só após este estudo, definiram-se quais sinais estariam presentes na interface e a estratégia de implementação a ser usada.

Faz-se neste ponto uma descrição sintetizada dos modelos de interface PCI verificados ao longo do estudo.

5.2.1.1. O *Softcore* da Empresa *Lattice*.

Na Figura 5.8 apresentam-se os detalhes de um núcleo *PCI Core da Lattice*[14][15]. Características do projeto são especificadas a seguir.

- Independência entre a largura de barramento PCI 32/64-bit e a largura do barramento da Interface Local e 32/64 Bit
- Compatível com especificação PCI ver.2.2
- Suporta endereçamento de 64 bits
- Detecção de Erro de paridade
- Até seis Registros de Endereço de base (BARs)
- Arquitetura de barramento local unidirecional até 66MHz PCI
- Projeto completamente síncrono
- O projeto do IP Core Inclui
  - Guia do Usuário de núcleo
  - Gabaritos codificados (*Netlist* e *VHDL/Verilog*)
  - *Testbench* e Modelos de Simulação

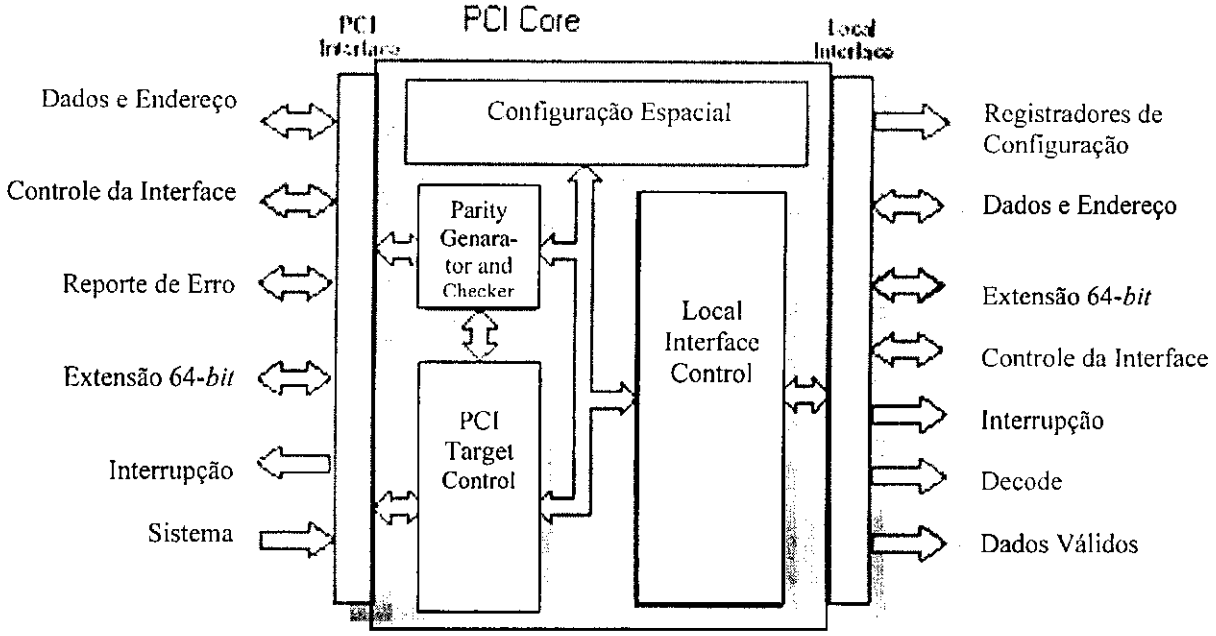


Figura 5.8 – Núcleo PCI – *Lattice*[14].

5.2.1.2. O *PCI Bridge IP Core* da *OpenCore*.

As principais características do núcleo IP da ponte PCI (*PCI Bridge IP Core*) da *OpenCore*[16] estão descritas a seguir:

- Interface de PCI de 32 *bits;*
- Completamente compatível com PCI 2.2 (66 MHz);
- Interface *Wishbone* 32 *bits*, 42 MHz;
- Suporta comando e funções:
  - Para ciclos único e rajada;
  - Interrupção;
  - Ponte Anfitriã (*Host Bridging*);
  - Configuração Espacial – Cabeçalho;
  - Geração de paridade, Detecção de Erro de Paridade (PERR# e SERR#)
- Registros de Status de comando completos Interconexão WISHBONE SoC rev. B.

Na Figura 5.9 tem-se ilustrado o diagrama em bloco das principais características do núcleo IP da ponte PCI da *OpenCore*.

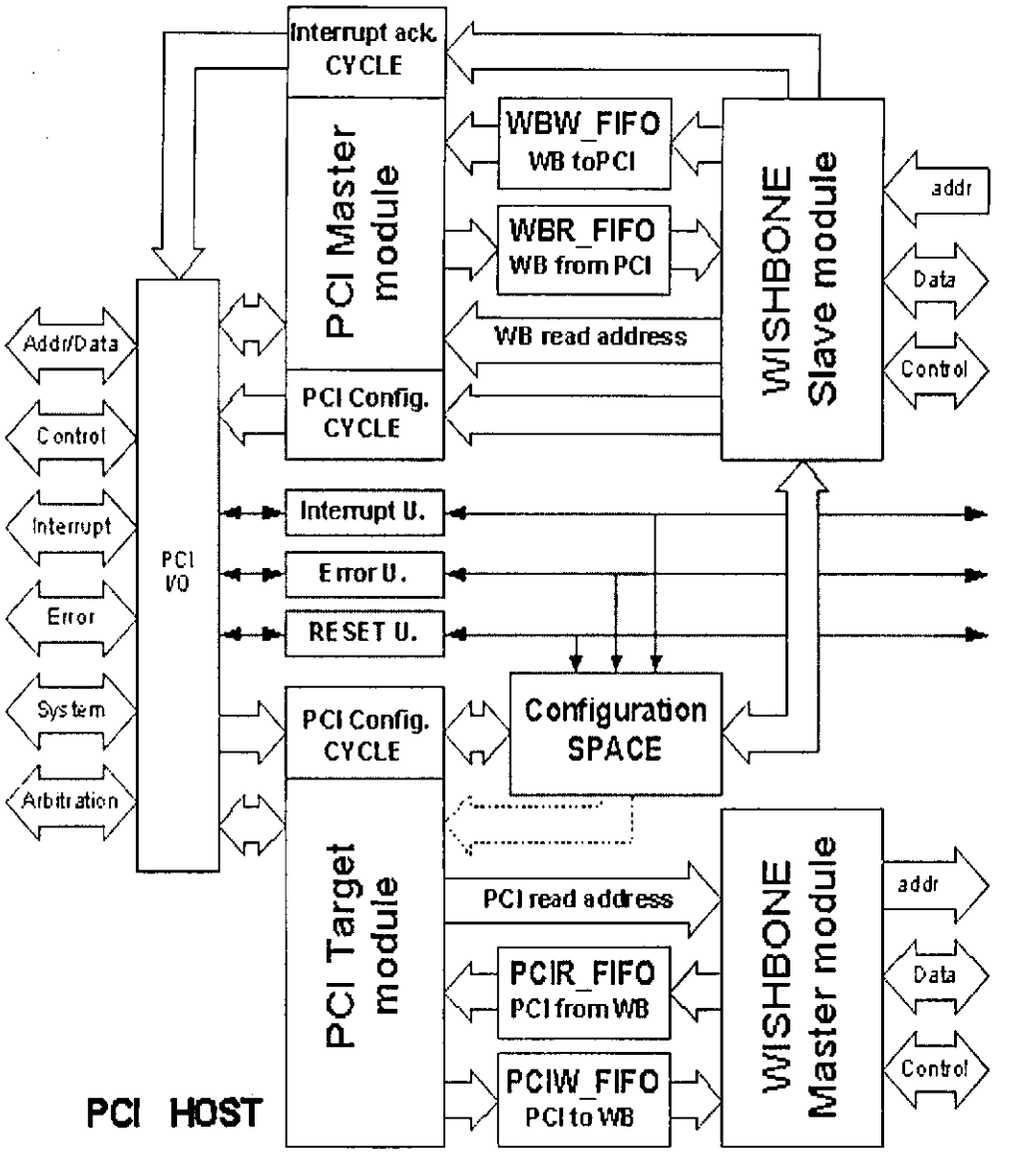


Figura 5.9 – Ponte PCI Anfitriã[16].

Nota: Os programas e manuais desse projeto da *OpenCore*[16] estão disponíveis na íntegra na Internet em: http//www.opencore.org/projects/pci.

5.2.1.3. Projeto de uma Interface PCI mínima

As principais características do projeto de Interface PCI Mínima[18] estão descritas a seguir:

- Interface de PCI de 32 *bits*
- Completamente compatível com PCI 2.2 (33 MHz)
- Suporta comando e funções:
  - Para ciclos único e rajada;
  - Configuração Espacial – Cabeçalho;
  - Geração de paridade, Detecção de Erro de Paridade (PERR#).
- Registros de Status de comando básicos

5.2.2. Modelagem em HDL

Após a interpretação da especificação, teve início a etapa de definição da estrutura da interface, descrevendo-se os blocos, para isso utilizou-se a conversão para HDL através da linguagem *SystemC™*.

As descrições dos blocos podem ser melhor compreendidas se visto o conteúdo exposto no Capítulo 2, no qual os sinais utilizados estão definidos. Os blocos usados foram detalhados no Capítulo 3.

Nesta fase, fez-se apenas a codificação dos sinais e blocos já definidos, por isso não será dado maior ênfase a este processo, pois trata-se apenas do uso da linguagem *SystemC™* para conversão desses blocos em HDL (Maiores informações sobre esta fase estão apresendados nos Apêndice F ou nas referências Melcher[19, 20] e *SystemC™* [21 - 26]).

Nas Figuras 5.10 e 5.11 é ilustrado o ambiente usado nesta etapa. Os programas utilizados nesta fase foram descritos para o ambiente do sistema operacional *Linux RedHat 7.2:* Edição XEMACS, compilação/conversão para *Verilog: SystemC™*, e controle de versões: o *SVN*.

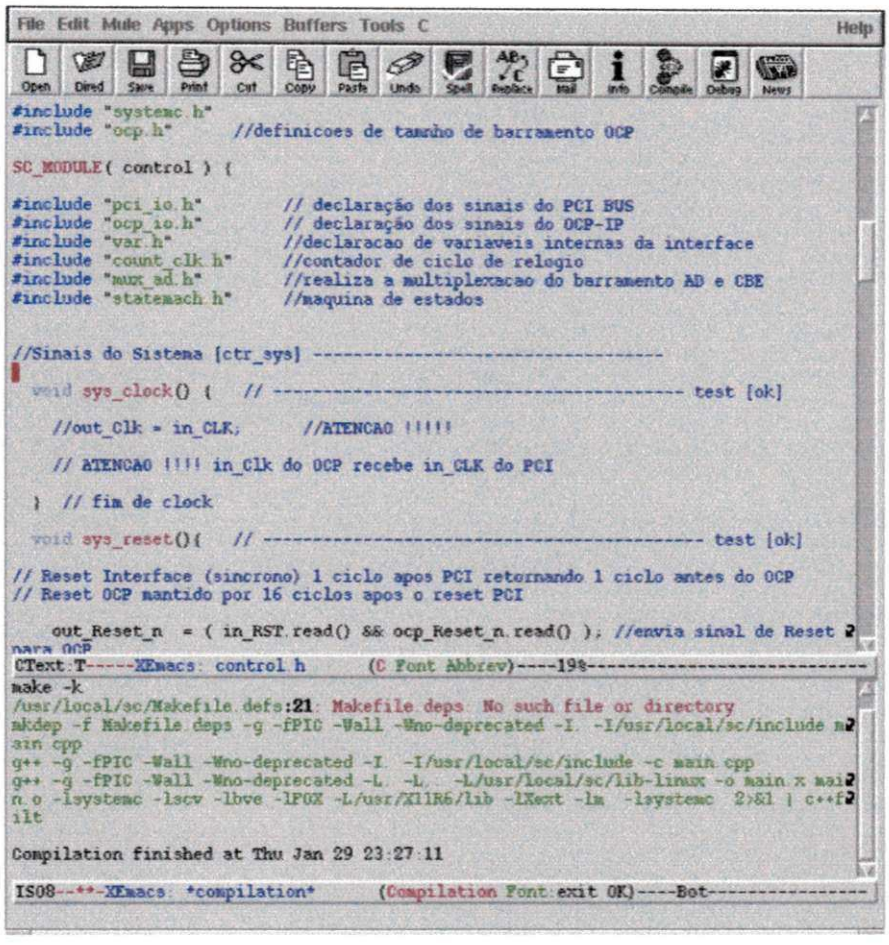


Figura 5.10 – Editor XEMACS.

O XEMACS é um editor extremamente versátil. A seguir é feito um resumo dos comandos básicos:

^W cortar

^Y colar

^X U desfazer

^D apaga letra a direita

[Delete] apaga letra a esquerda

^K apaga todo alinha a direita

^X ^F abre arquivo do disco ou cria novo

^X ^D abre diretório

^X ^W salvar como

^S [palavra] procura pela palavra

^X 1 uma janela ativa

^X 2 duas janelas ativas

^X O alterna janela ativa

Para a automatização da compilação faz-se uso do comando *Make* que lê e executa o arquivo *Makefile*, o *Make* só recompila os arquivos modificados. Este processo pode ser realizado do *XEMACS* ou no ambiente de comando. Após término da compilação tem-se o **arquivo.x** que é executado.

Para conversão de *SystemC™* é necessário o programa *SystemC Compiler* (Figura 5.11). A seguir a seqüência de comandos usados.

[] dc_shell
[] compile_systemc –rtl –rtl_format verilog arquivo.h

O resposta desta conversão para *Verilog*, caso não apresente erro, é o numeral 1. São gerados os arquivos **arquivo.v, *filenames.log*** e ***command.log***. O **arquivo.v** será usado na etapa de síntese.

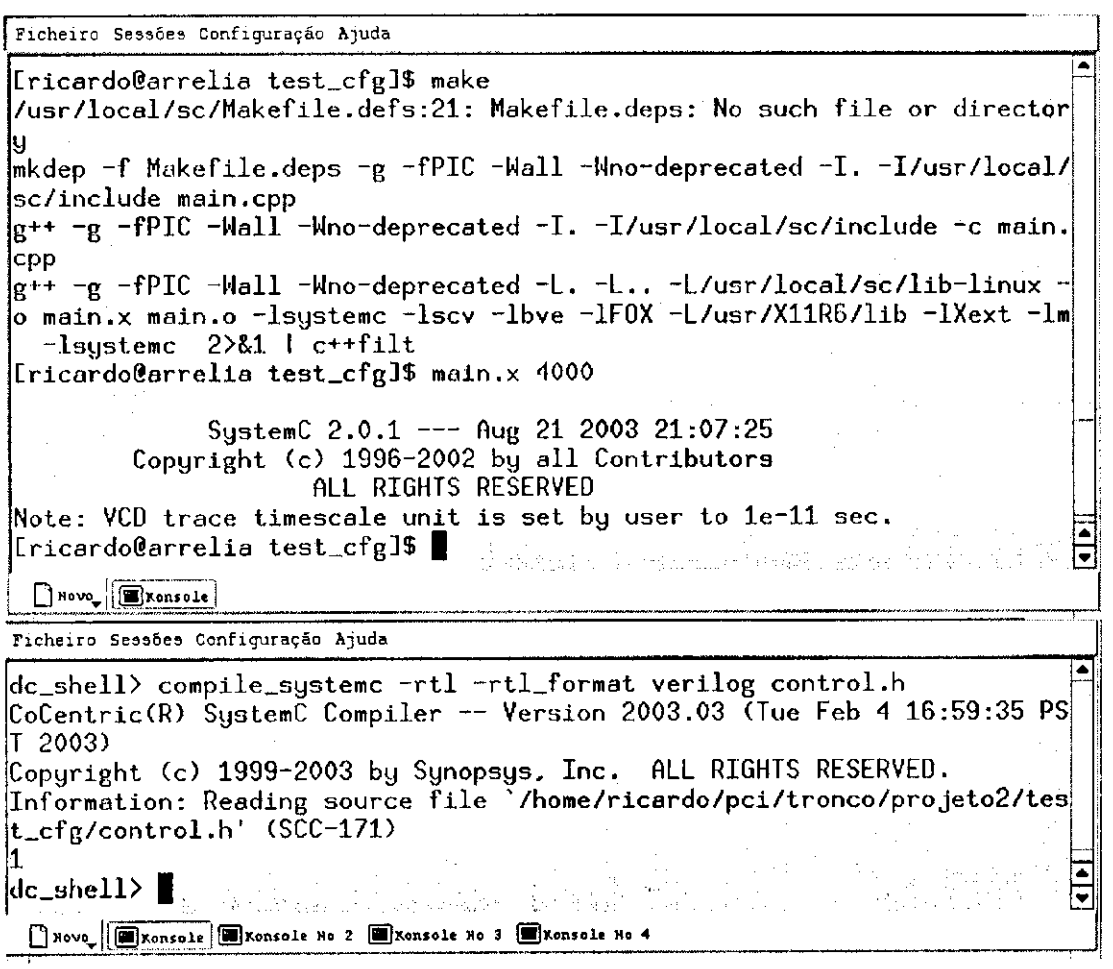


Figura 5.11 – Compilador/Conversor *SystemC™*.

5.2.3. Verificação Funcional

Com base na especificação, escreve-se o Ambiente de Teste por Simulação para validar a descrição do sistema. Nesta dissertação foi realizada apenas a etapa de visualização lógica dos sinais por intermédio de programas de verificação lógica.

Tem-se na Figura 5.12, como exemplo, programas que podem ser utilizados nesta fase para verificação lógica (*ModelSim* e *SimVision*). Para automação dessa fase é necessária a criação de um arquivo com todos os sinais a serem apresentados.

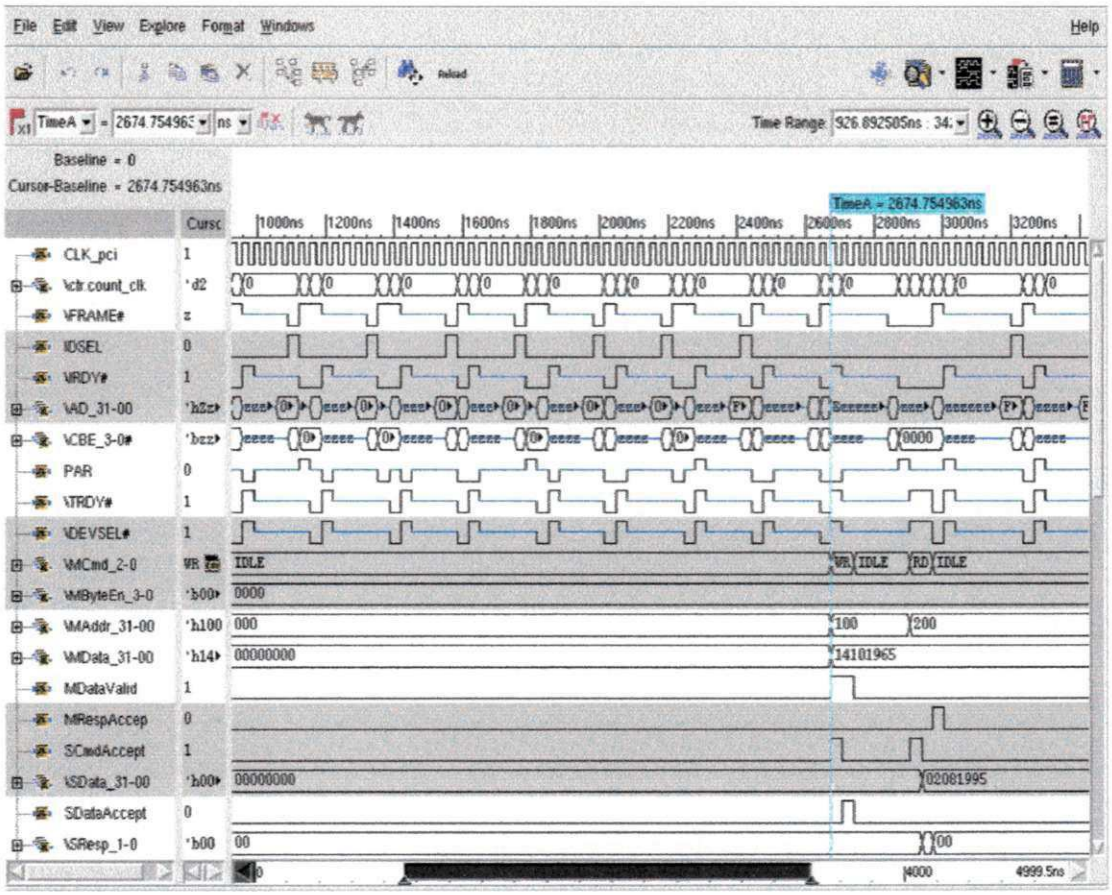


Figura 5.12 – Verificação Lógica.

5.2.4. Síntese

Na etapa de síntese, tem-se por tarefa, transformar a descrição RTL em descrição estrutural e otimizá-la do ponto de vista lógico. Na realização desta tarefa fez-se uso do programa *Leonardo Spectrum* (Figura 5.13).

Neste ambiente, de posse do arquivo gerado em etapa anterior (**arquivo.v**), dá-se início a síntese. Para automação desta fase pode-se editar os **arquivo.tcl** **arquivo.acf** (neste trabalho tem-se o **gera.tcl** e o **conrol.acf**) com informações necessárias ao ambiente de trabalho para o programa *Leonardo Spectrum* e descrição dos pinos para a placa FPGA.

Após o término desta etapa são gerados os arquivos **arquivo.edf**, **arquivo.lsp, arquivo.scr**, **arquivo.xdb, exemplar.his** e **exemplar.log**.

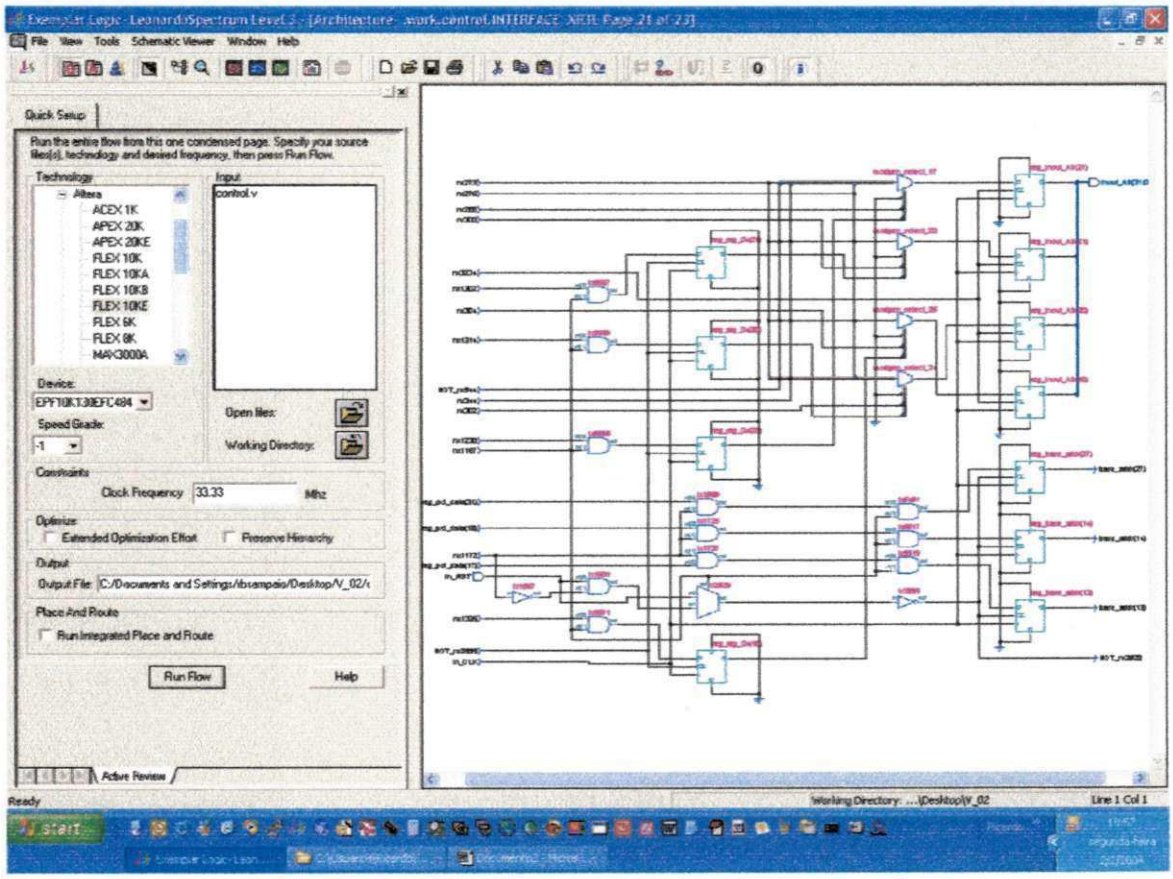


Figura 5.13 – Síntese com *Leonardo Spectrum*.

A seguir é feita uma descrição sintética dos procedimentos necessários à implementação desta etapa.

- Executar o programa *Leonardo Spectrum*
    - *Leonardo Spectrum Level 3*
    - *Run license selection next time*

Na barra de *menu* selecione:

- *File*
    - *New Project*

Nos quadros de diálogos selecione;

- *Technology*
    - ***FPGA/CPLD***
        - ***FLAX 10KE***
- *Device*
    - ***EPF10K100EFC484***
- *Speed Grade*
    - ***-1***
- *Clock Frequency:*
    - **33.33 MHz**
- *Working Directory:* (selecione o diretório de trabalho)
- *Open File:* (selecione arquivo *Verilog* com sua descrição RTL "**arquivo.v**)
- No barra de *menu* selecione:
    - *File*
        - *Run Scrip* (selecione seu **arquivo.tcl**)

Para salvar o projeto selecione na barra de *menu*:

- *File*
    - *Save Project*

De posse do **arquivo.edf** dar-se início a nova etapa, leiaute, conforme descrito a seguir.

Nota: Os textos em negrito deverão ser selecionados conforme necessidade do projeto do usuário.

5.2.5. Leiaute

Na fase de leiaute usa-se uma ferramenta específica da tecnologia na qual se deseja implementar o circuito.

Nesta etapa são descritos o posicionamento, roteamento e a programação da FPGA. Para esta etapa utiliza-se o programa *Quartus* II (Figura 5.14) da empresa Altera, fabricante da placa FPGA "FLEX 10KE".

Para automação desta fase, são necessários os arquivos: **arquivo.acf,** no qual, tem-se a descrição de todos os pinos da placa FPGA e o **arquivo.edf** que foi gerado na etapa anterior.

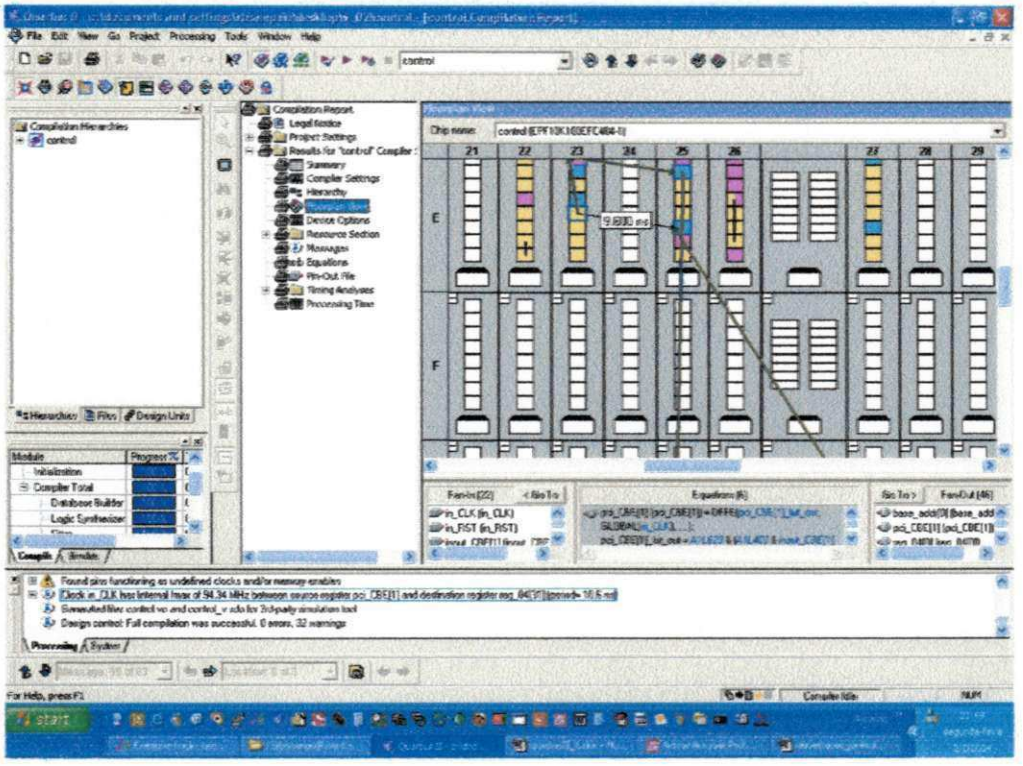


Figura 5.14 – Leiaute com *Quartus II*.

Após o término desta etapa são gerados muitos arquivos com relatórios completos do projeto (arquitetura, temporização, etc.), o arquivo de interesse imediato é o **arquivo.sof**, que é usado para a programação da placa FPGA.

Para verificar se houve a importação correta do **arquivo.acf,** pode-se fazer a verificação de alguns itens na barra de *menu*:

Project

Project Wizard

Next

Next (conferir dados)

family **FLEX 10KE**

Next

Device **EPF10KEFC484 -1**

Finish

Project

Timing Setting

*Clock Settings **33.33** MHz*

Other Requirements & Options

*tsu **7** ns*

th -- ns

*tco **11** ns*

*tpd **30** ns*

OK

Para gerar leiaute selecionar na barra de *menu*:

Processing

Start Analysis & Compilation

Start Compilation Start Timing Analysis

Para gravar o projeto na placa FPGA

Processing

Open Programmer

Mode [Passive Serial PS] foi usado cabo PS

*Add File... **[arquivo.sof]***

Start.

Nota: Os textos em negrito deverão ser selecionados conforme necessidade do projeto do usuário.

5.2.6. Simulação com Atrasos

Nesta etapa do projeto deve-se fazer a re-simulação com as anotações de atraso oriundas do leiaute, verificando se todos os tempos de atrasos requeridos pela PCI foram satisfeitos (*tsu = 7 ns, tco = 11 ns, tpd = 30 ns, clock = 33,33 MHz*), caso contrário, deve-se fazer modificações no código fonte para atender as necessidades de tempo e repetir todo o processo até que as condições de atrasos sejam satisfeitas.

tsu *clock setup time*
tco *clock to output time*
tpd *pin to pin delay time or point-to-point delay time*

5.2.7. Validação do Protótipo

A validação do núcleo IP se deu pelo emprego de programas que auxiliaram a análise visual da simulação (sinais no barramento PCI) e os valores contidos nos registradores de configuração espacial.

Na implementação em FPGA, após etapa de síntese e leiaute, a validação se faz pelo uso de um Aplicativo de Dispositivo do Usuário (aplicação básica de acesso à escrita e leitura em um endereço na memória). Nesta etapa foram realizados testes de leitura e escrita no espaço de configuração, nos ambientes dos sistemas operacionais *DOS Ex* e *Windows* (Figuras 5.15 e 5.16).

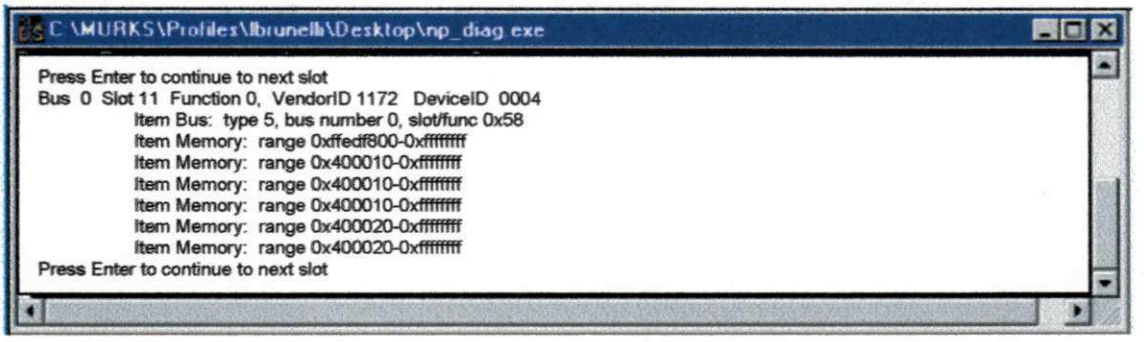


Figura 5.15 – Acesso aos registradores de configuração no ambiente DOS Ex.

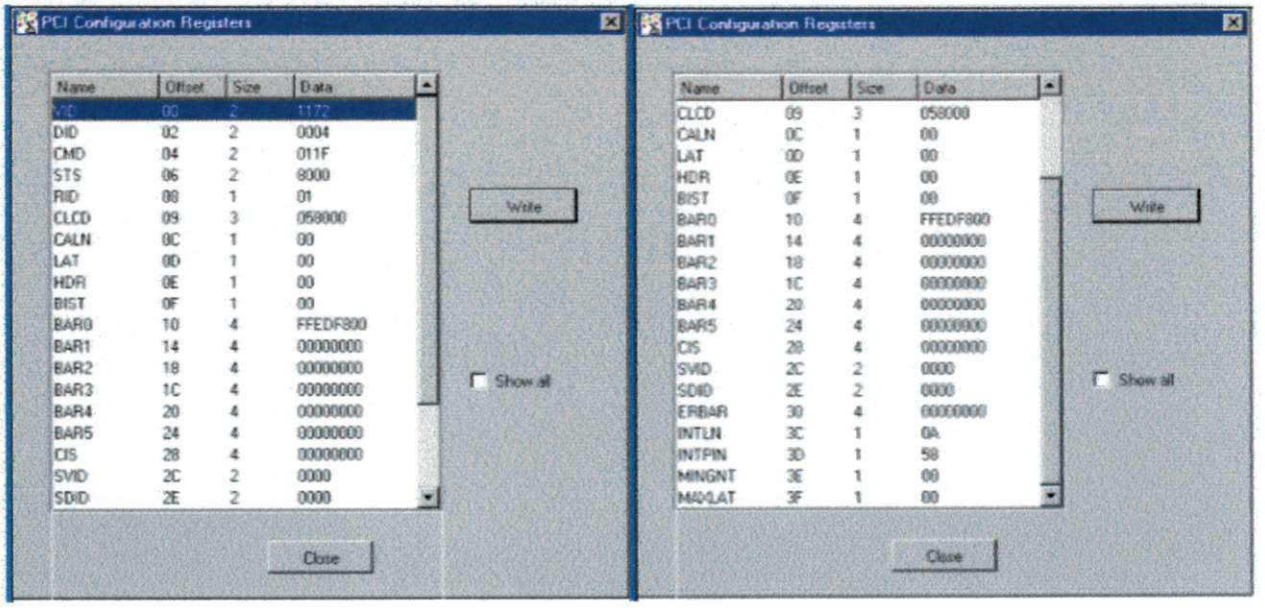


Figura 5.16 – Acesso aos registradores de configuração no ambiente Windows.

Também foram realizadas as medidas dos sinais no barramento, realizadas através do osciloscópio, para isto faz-se uso nesta etapa do programa *WaveStar* para aquisição no computador dos dados gerados no osciloscópio (Figura 5.17).

Para conectar a ponta de prova do osciloscópio ao barramento, é necessário realizar uma "janela" no programa, desviar os sinais presentes no barramento para pinos de saída da placa FPGA (pontos do *Proto* Figura 5.5), na qual são realizadas as medidas reais. Não se pode conectar a ponta de prova do osciloscópio diretamente ao barramento.

No Capítulo 6, são apresentados os gráficos com os resultados dos testes de depuração do programa núcleo.

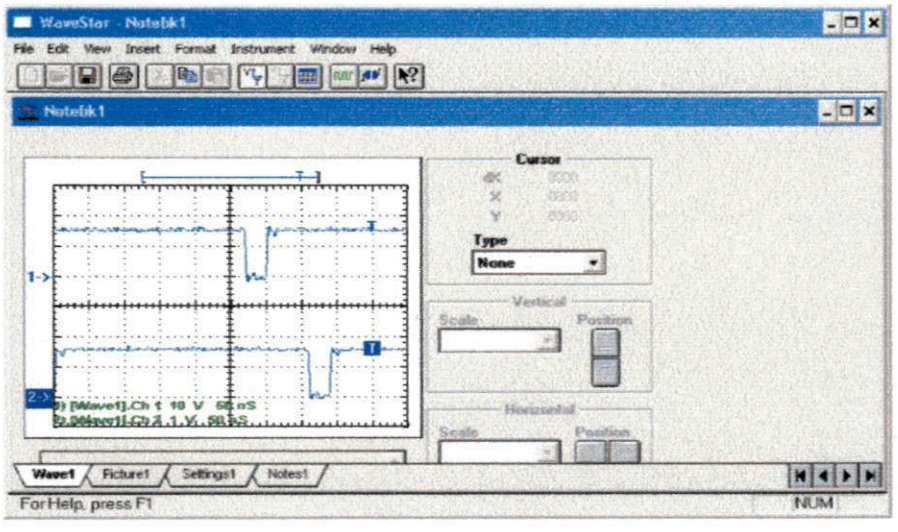


Figura 5.17 – Medições de sinais com osciloscópio - *WaveStar*.

Ainda, durante a fase de simulação, tem-se o emprego do Ambiente de Teste Funcional e o uso do programa *CoreCreator* para testar e validar o núcleo para o padrão OCP™.

6. CAPÍTULO 6 – RESULTADOS

Neste Capítulo é exposto o fruto do trabalho, ou seja, os resultados. Resultados estes obtidos por simulação no laboratório LIMC/DEE/UFCG, onde foram desenvolvidos e implementados os códigos HDL. Os diagramas de tempo das transações foram todos capturados na plataforma *Linux Red Hat 7.2* com auxílio de programas para visualização dos sinais. Os resultados obtidos correspondem à implementação em FPGA, no laboratório LAD/DSC/UFCG, onde foram realizadas as etapas de síntese, leiaute e validação do protótipo, realizados na plataforma Windows XP e DOS Ex.

6.1. Resultados em Simulação

Foram realizados testes de todo o programa núcleo apenas em ambiente de simulação, do qual foram extraídos os gráficos presentes nesta dissertação.

6.1.1. Ciclo de Reiniciar

No Gráfico 6.1 ilustra-se o sinal de *Reset*, no qual se tem as seqüências de eventos descritas a seguir.

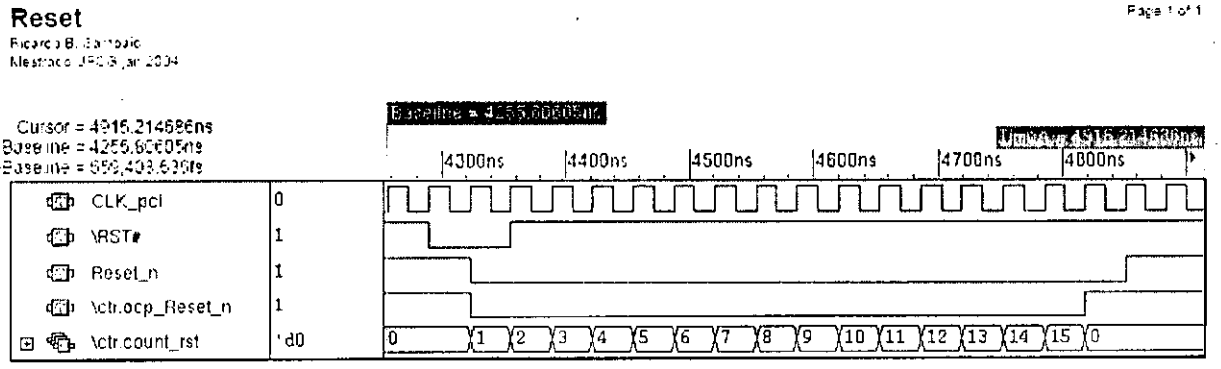


Gráfico 6.1 – Sinal de *Reset*.

count_clk	descrição do evento
[0]	A PCI aciona o sinal de RST.
[1]	O sinal ocp_Reset_n e Reset_n são acionados gerando o estado de reinicializar da interface e do Destino OCP™.
[2]	A PCI restaura seu estado inicial estando pronta para realizar as transações.
[3-14]	A interface e o Destino OCP™ permanecem em estado de reiniciar.
[15]7	A interface é ativada em um ciclo de relógio antes do Destino OCP™, preparando-se para as transações.
[0]	O Destino OCP™ é ativado.

Nota: Manteve-se o sinal de reiniciar ativo por 16 ciclos de relógio por sugestão acatada do protocolo OCP-IP™, que informa que alguns dispositivos mais lentos podem não responder ao sinal de reinicializar se este for muito rápido como no caso do PCI.

6.1.2. Transação de Leitura de Configuração

No Gráfico 6.2 tem-se uma leitura de configuração na qual se pode visualizar a leitura do registrador de endereço 00h [*Device ID, Vendor ID*]. A seqüência de eventos é descrita a seguir.

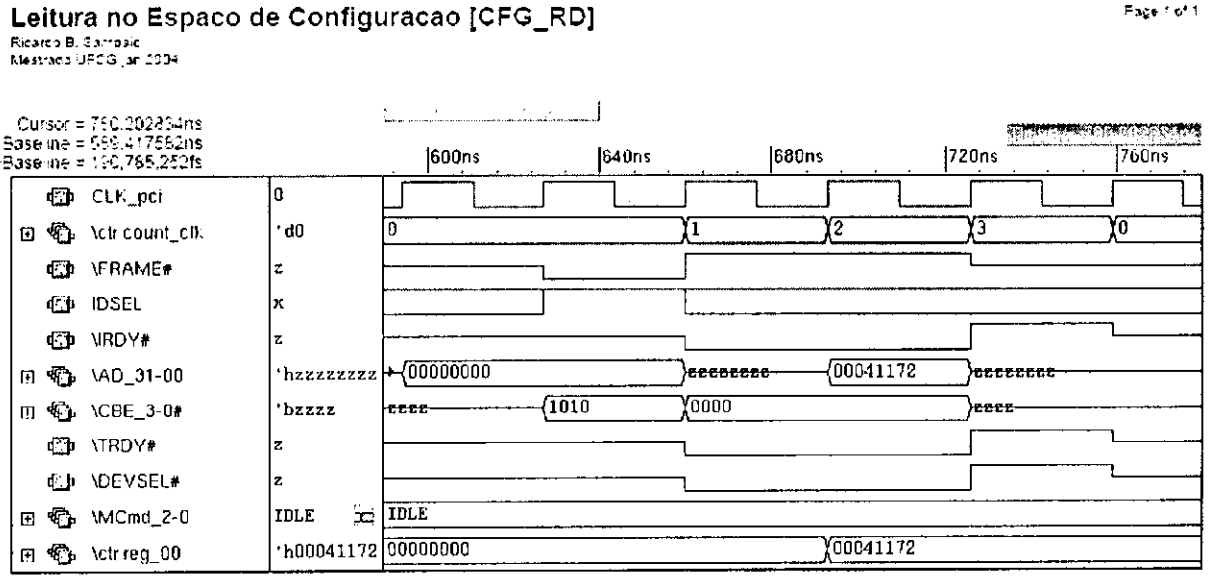


Gráfico 6.2 – Transação de Leitura de Configuração no Registro 00h.

count_clk descrição do evento

[0] A PCI coloca no barramento AD o endereço do registrador;

Ativa o sinal de FRAME (0);

Ativa o sinal de IDSEL (1);

Disponibiliza o comando [CFG_RD]

Tem-se a inversão dos sinais IRDY, TRDY e DEVSEL, estão em terceiro estado, alta-impedância.

[1] Atividades da PCI:

Inverte o barramento AD para receber os dados;

Coloca os dígitos de habilitação;

Retira o sinal de IDSEL, (não vai a zero);

Afirma o sinal de IRDY;

Coloca o sinal de FRAME em 1 e aguarda os dados;

Atividades da Interface:

Ativa o sinal de DEVSEL, indica que reconhece a transação;

Ativa o sinal de TRDY indicando que está pronta para a transação;

[2] A Interface transfere os dados do registrador para o barramento AD.

[3] A PCI coloca o sinal de FRAME, AD e CBE em alta-impedância e o sinal de IRDY em 1.

A Interface deixa DEVSEL e TRDY em nível alto por um ciclo.

[0] A Interface deixa os sinais de DEVSEL e TRDY em alta-impedância disponíveis para a PCI que faz o mesmo com o sinal IRDY.

Nota: Pode-se observar que durante a transação de configuração a Interface deixa o Destino em OCP™ INATIVO (o *MCmd* em *IDLE*).

6.1.3. Transação de Escrita de Configuração

No Gráfico 6.3 tem-se uma escrita de configuração no qual se pode visualizar a escrita do registrador de endereço 04h [*Status, Command*]. A seqüência de eventos é descrita a seguir.

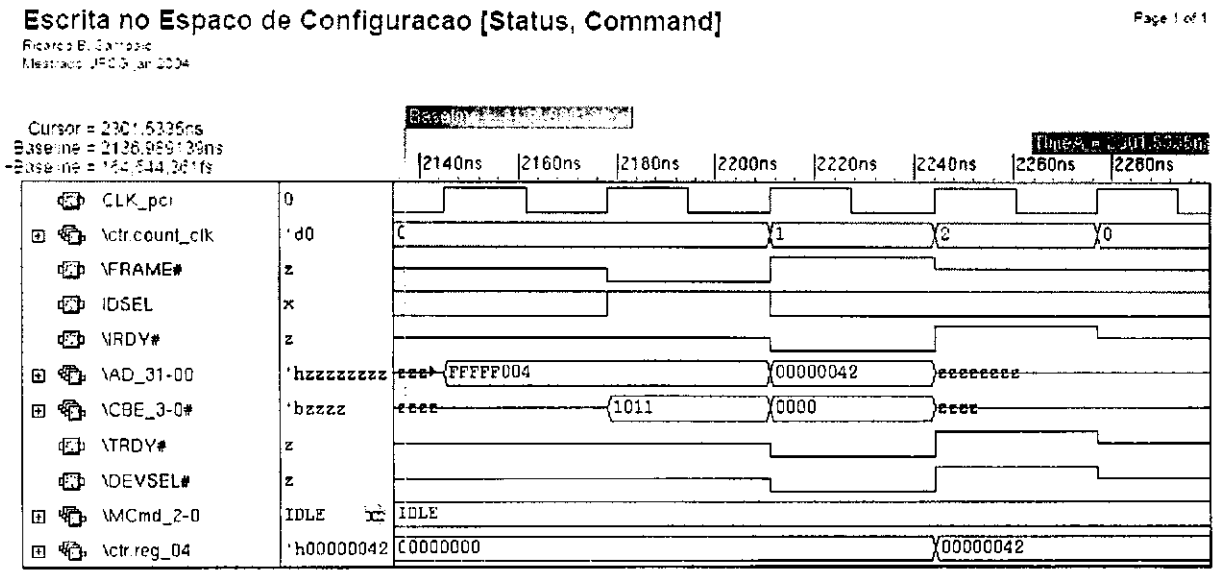


Gráfico 6.3 – Transação de Escrita de Configuração no Registro 04h.

count_clk descrição do evento

[0] A PCI coloca no barramento AD o endereço do registrador;
Ativa o sinal de FRAME (0);
Ativa o sinal de IDSEL (1);
Disponibiliza o comando [CFG_WR]
Tem-se a inversão dos sinais IRDY, TRDY e DEVSEL, estão em terceiro estado, alta-impedância.

[1] Atividades da PCI:
Coloca os dados no AD
Coloca os dígitos de habilitação;
Retira o sinal de IDSEL, (não vai a zero);
Afirma o sinal de IRDY;
Coloca o sinal de FRAME em 1 e aguarda os dados;

Atividades da Interface:

Ativa o sinal de DEVSEL, indica que reconhece a transação;

Ativa o sinal de TRDY indicando que está pronta para a transação;

A Interface recebe os dados.

[2] A PCI coloca o sinal de FRAME, AD e CBE em alta-impedância e o sinal de IRDY em 1.

A Interface deixa DEVSEL e TRDY em nível alto por um ciclo e grava os dados no registrador.

[0] A Interface deixa os sinais de DEVSEL e TRDY em alta-impedância disponíveis para a PCI que faz o mesmo com o sinal IRDY.

6.1.4. Transações de Configuração de Leitura e Escrita do Registro BAR0

No Gráfico 6.4 visualiza-se as transações de leitura e escrita, que são similares às transações descritas anteriormente, estando a maior diferença na escrita no registro de endereço 10h [BAR0] do valor de FFFFFFFFh, pois a escrita desse valor no BAR sinaliza à interface que a PCI deseja ler o tamanho da memória requerida pela Interface e outras informações. Após essa leitura a PCI faz uma nova escrita no BAR0, desta fez do valor do endereço base, o que permite o acesso aos recursos e outras informações.

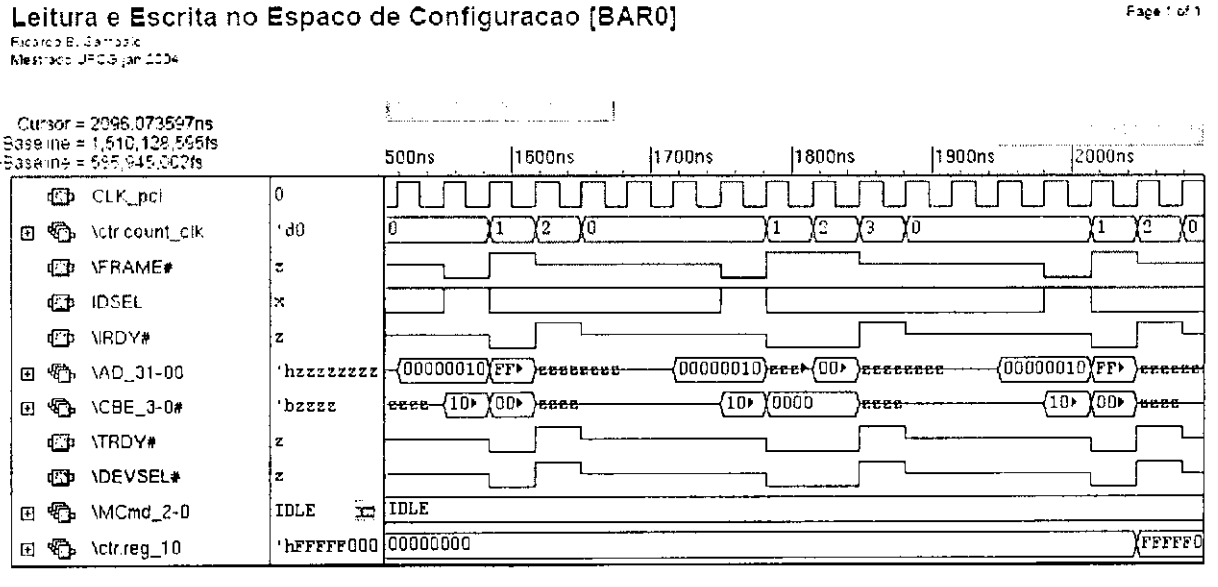


Gráfico 6.4 – Transações de Configuração no Registro BAR0.

6.1.5. Transação de Escrita na Memória

No Gráfico 6.5 tem-se ilustrada uma transação de escrita na memória. A seqüência de eventos é descrita a seguir.

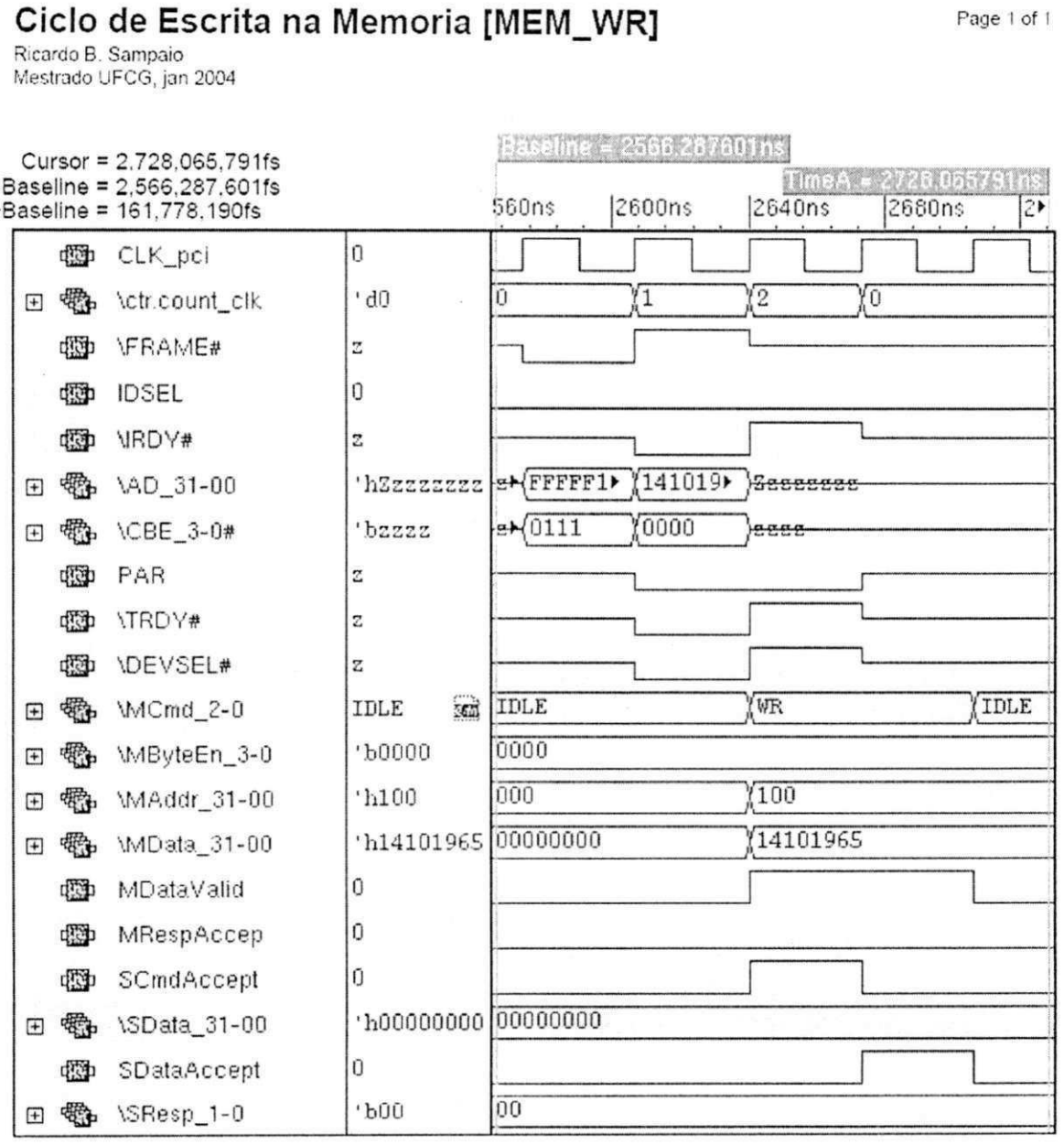


Gráfico 6.5 – Transação de Escrita na Memória.

count_clk descrição do evento

[0] A PCI ativa o sinal de FRAME (0); coloca o endereço em AD; disponibiliza o comando [MEM_WR] e inverte os sinais IRDY, TRDY e DEVSEL.

[1] Atividades da PCI:

O sinal de paridade PAR de endereço e comando é colocado;

Coloca os dados no barramento AD; os dígitos de habilitação em CBE; afirma o sinal de IRDY (0) e o sinal de FRAME em 1;

Atividades da Interface:

Ativa o sinal de DEVSEL e o sinal de TRDY, recebe os dados.

[2] Atividades da PCI:

Coloca o sinal de FRAME, AD e CBE em alta-impedância;

O sinal de paridade PAR de dados e dígitos de habilitação é colocado;

Coloca o sinal de IRDY em 1.

Atividades da Interface:

Coloca DEVSEL e TRDY em nível alto;

Disponibiliza o endereço em Maddr e os dados em Mdata.

Envia os sinal de escrita no MCmd, os dígitos de habilitação em MbyteEn;

Ativa o sinal de MdataValid

[0] A PCI deixa IRDY em alta-impedância.

A Interface deixa os sinais de DEVSEL e TRDY em alta-impedância Recebe o sinal de SResp e retira o sinal de MdataValid para no ciclo seguinte colocar o sinal de INATIVO (IDLE) no MCmd.

6.1.6. Transação de Leitura na Memória

No Gráfico 6.6 tem-se ilustrado uma transação de leitura na memória. A seqüência de eventos é descrita a seguir.

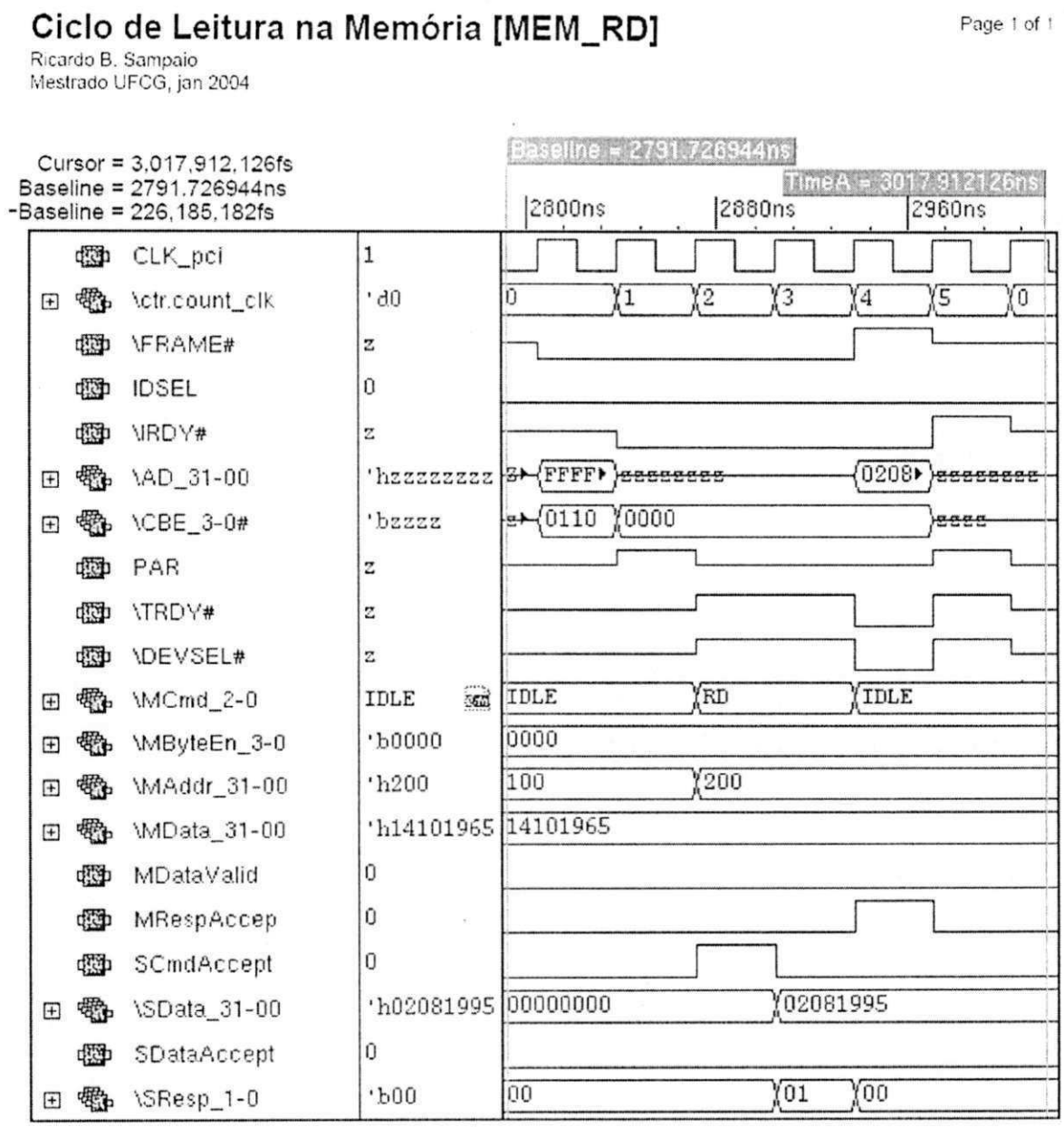


Gráfico 6.6 – Transação de Leitura na Memória.

count_clk	descrição do evento
[0]	A PCI ativa o sinal de FRAME (0); coloca o endereço em AD; disponibiliza o comando [MEM_RD] e inverte os sinais IRDY, TRDY e DEVSEL.
[1]	A PCI inverte o barramento AD; coloca os dígitos de habilitação em CBE; afirma o sinal de IRDY (0); O sinal de paridade PAR de endereço e comando é colocado;
[2]	A Interface: A Interface deixa DEVSEL em nível alto; Coloca TRDY em nível alto; Disponibiliza o endereço em Maddr e os dígitos de habilitação em MByteEn. Envia o comando de leitura (RD) no MCmd; A Interface fica aguardando o comando SCmdAccept do Destino OCP™.
[3]	O Destino OCP™ envia resposta de dados disponíveis (SResp=DVA) e disponibiliza os dados no barramento SData
[4]	A Interface responde que aceitou os dados MRespAccept (1); A Interface coloca o comando de Inativo (MCmd= IDLE) para o destino OCP™. A Interface ativa DEVSEL (0) e o sinal de TRDY (0) e transfere os dados para AD.; O Destino OCP™ retira o sinal Resp (NULL) A PCI sobe o sinal de FRAME para nível alto.
[5]	A PCI coloca FRAME, AD e CBE em alta impedância e o sinal TRDY em nível alto. A interface coloca o sinal de paridade PAR de dados e dígitos de habilitação são colocados; A Interface deixa os sinais de DEVSEL e TRDY em nível alto.
[0]	A Interface deixa os sinais de DEVSEL e TRDY em alta-impedância.

6.1.7. Verificação de Paridade e Sinalização de Erro

No Gráfico 6.7 tem-se ilustrado um exemplo de cheque de paridade, foi introduzido um valor de paridade errado na transação de endereço e comando (PAR = 1, valor correto "0"), o mesmo sendo feito na transação de dados e dígitos de habilitação (PAR = 0, valor correto "1") para poder demonstrar a resposta dos sinais de SERR e PERR ao se detectar uma inconsistência no cálculo da paridade. O sinal de SERR é do tipo dreno aberto com resistor de *pull up* sendo ativado em nível baixo e depois retorna a nível alto, já o sinal de PERR é do tipo terceiro-estado-sustentado, portanto, após ser ativado em nível baixo deve ir a nível alto por um ciclo de relógio e entrar em alta-impedância.

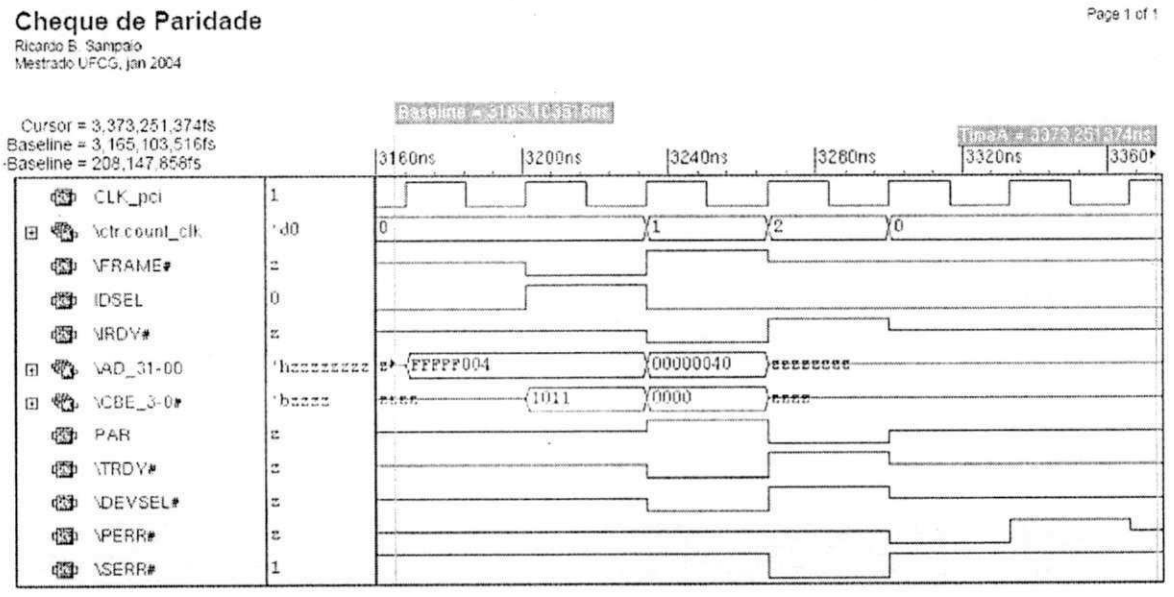


Gráfico 6.7 – Verificação de Paridade.

6.2. Resultados na Implementação

Nesta dissertação foi implementada em FPGA apenas uma parte do programa núcleo, a etapa de configuração da PCI. Detalhes do ciclo de configuração da interface PCI estão expostos no Apêndice D item "Registradores de configuração".

Os estados obtidos nesta etapa de leiaute, correspondem aos dados descritos nas Tabelas 6.1, 6.2, 6.3, 6.4 e 6.5. Conforme se pode observar, obteve-se êxito nesta etapa, alcançando a freqüência de trabalho desejada 33,3 MHz, os requisitos de tempo exigidos para o padrão PCI v2.2 de tsu 7 ns, tco 11 ns e tpd 30 ns também foram alcançados.

Tabela 6.1 – Sumário do processo de leiaute.

Summary	
Processing status	Fitting Successful
Timing requirements/analysis status	Requirements met
Chip name	Control
Device name	EPF10K100EFC484-1
Total logic elements	216 / 4992 (4 %)
Total pins	67 / 338 (19 %)
Total EAB bits	0 / 49152 (0 %)

Tabela 6.2 – Freqüência máxima.

fmax (not incl. delays to/from pins)		
Clock Name -- Destination Register Name -- Source Register Name	**Required fmax**	**Actual fmax (period)**
pci_clk	33.3 MHz	91.74 MHz (period = 10.900 ns)
-- reg_reg_04_1	33.3 MHz	91.74 MHz (period = 10.900 ns)
-- reg_state_3	33.3 MHz	91.74 MHz (period = 10.900 ns)
-- Timing analysis results restricted.		To change the limit use Timing Settings (Project menu)

Tabela 6.3 – Parâmetros de contagem de tempo para configuração usados no PCI.

Timing Settings

Assignment File	Source Name	Destination Name	Option	Setting
control.psf			Include external delays to/from device pins in fmax calculations	Off
control.psf			RUN_ALL_TIMING_ANALYSES	On
control.psf			Ignore user-defined clock settings	On
control.psf			FMAX_REQUIREMENT	33.3MHz
control.psf			tco Requirement	11ns
control.psf			tsu Requirement	7ns
control.psf			tpd Requirement	30ns
control.psf			Default hold multicycle	Same As Multicycle
control.psf			Cut off feedback from I/O pins	On
control.psf			Cut off clear and preset signal paths	On
control.psf			Cut off read during write signal paths	On
control.psf			Cut paths between unrelated clock domains	On
control.psf			Number of source nodes to report per destination node	10
control.psf			Maximum Strongly Connected Component loop size	50

Tabela 6.4 – Requisitos *tsu*.

tsu Requirements

Source Name	Destination Name	Destination Clock Name	Required tsu	Actual tsu	Slack
pci_ad[16]	reg_reg_pci_data_16	pci_clk	7.000 ns	6.800 ns	0.200 ns
pci_idsel	reg_state_1	pci_clk	7.000 ns	6.200 ns	0.800 ns
pci_idsel	reg_state_0	pci_clk	7.000 ns	6.200 ns	0.800 ns
pci_idsel	reg_state_2	pci_clk	7.000 ns	6.000 ns	1.000 ns
pci_ad[25]	reg_reg_pci_data_25	pci_clk	7.000 ns	5.300 ns	1.700 ns
pci_ad[26]	reg_reg_pci_data_26	pci_clk	7.000 ns	5.200 ns	1.800 ns
pci_ad[28]	reg_reg_pci_data_28	pci_clk	7.000 ns	5.200 ns	1.800 ns
pci_ad[30]	reg_reg_pci_data_30	pci_clk	7.000 ns	5.200 ns	1.800 ns
pci_ad[31]	reg_reg_pci_data_31	pci_clk	7.000 ns	5.100 ns	1.900 ns
pci_rst_l	reg_reg_04_8	pci_clk	7.000 ns	4.600 ns	2.400 ns
Timing analysis restricted to 10 rows.	To change the limit use Timing Settings (Project menu)				

Tabela 6.5 – Requisitos *tco*.

tco Requirements

Source Name	Destination Name	Source Clock Name	Required tco	Actual tco	Slack
reg_state_3	pci_devsel_l	pci_clk	11.000 ns	9.000 ns	2.000 ns
reg_state_3	pci_trdy_l	pci_clk	11.000 ns	9.000 ns	2.000 ns
reg_state_3	proto[17]	pci_clk	11.000 ns	8.800 ns	2.200 ns
reg_state_1	pci_devsel_l	pci_clk	11.000 ns	8.500 ns	2.500 ns
reg_state_1	pci_trdy_l	pci_clk	11.000 ns	8.500 ns	2.500 ns
reg_state_0	pci_devsel_l	pci_clk	11.000 ns	8.400 ns	2.600 ns
reg_state_0	pci_trdy_l	pci_clk	11.000 ns	8.400 ns	2.600 ns
reg_state_2	pci_devsel_l	pci_clk	11.000 ns	8.300 ns	2.700 ns
reg_state_2	pci_trdy_l	pci_clk	11.000 ns	8.300 ns	2.700 ns
reg_state_1	proto[17]	pci_clk	11.000 ns	8.300 ns	2.700 ns
Timing analysis restricted to 10 rows.	To change the limit use Timing Settings (Project menu)				

Tabela 6.6 – Requisitos *tpd*.

tpd Requirements

Source Name	Destination Name	Required Longest P2P Time	Actual Longest P2P Time	Slack
pci_idsel	proto[18]	30.000 ns	11.600 ns	18.400 ns
pci_frame_l	proto[19]	30.000 ns	7.100 ns	22.900 ns

Nota: As tabelas 6.1 à 6.6 apresentadas neste item, estão em língua inglesa, por tratar-se de tabelas originais extraídas do **arquivo.csf.htm** (nesta dissertação "**control.csf.htm**"), gerados através do programa *Quartus II*, após o término do processo de leiaute.

A seguir, tem-se uma seqüência de gráficos com os resultados obtidos após o término desta implementação, os sinais foram medidos com o osciloscópio *Tektronix TDS 220* e transmitidos ao computador via *RS 232*, onde foram capturados através do programa *WaveStar*.

No Gráfico 6.8 tem-se uma transação de leitura, o sinal DEVSEL aguarda um ciclo de relógio para ser ativado, ou seja, será ativado 60 ns depois do sinal FRAME ter sido ativado. Tem-se no canal 1 o sinal FRAME e no canal 2 o sinal DEVSEL.

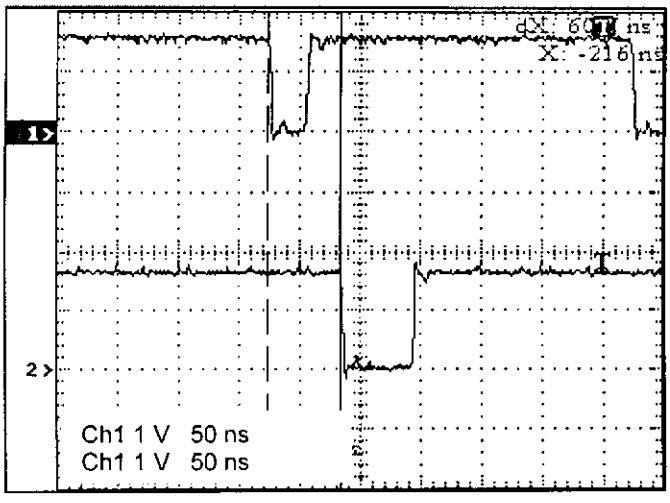


Gráfico 6.8 – Sinais FRAME e DEVSEL.

No Gráfico 6.9 tem-se no canal 1 o sinal FRAME e no canal 2 o sinal IDSEL. O sinal IDSEL funciona como um circuito seletor de dispositivo convencional, quando ativo (1) indica que o dispositivo é o alvo da transação. Deve ser retirado depois do sinal FRAME ser ativado.

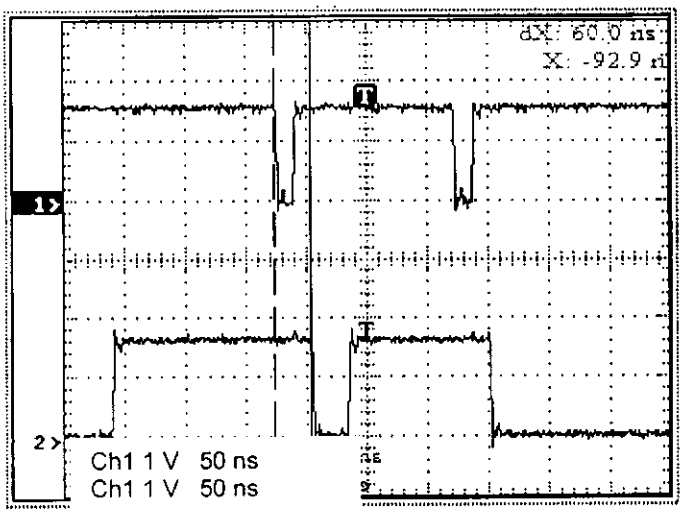


Gráfico 6.9 – Sinais FRAME e IDSEL.

No Gráfico 6.10 tem-se no canal 1 o sinal FRAME e no canal 2 o sinal IRDY. O sinal IRDY indica que o mestre do barramento está disponível para realizar a transação, nota-se que este sinal IRDY é ativo logo após o FRAME, permanecendo ativo durante a transação, caso contrário gera um estado de espera para o dispositivo alvo.

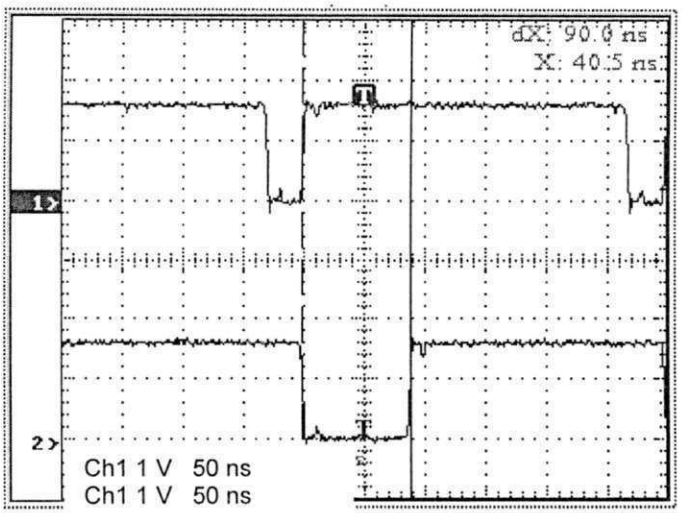


Gráfico 6.10 – Sinais FRAME e IRDY.

No Gráfico 6.11 tem-se no canal 1 o sinal FRAME e no canal 2 o sinal TRDY. O sinal TRDY indica que o dispositivo alvo está disponível para realizar a transação.

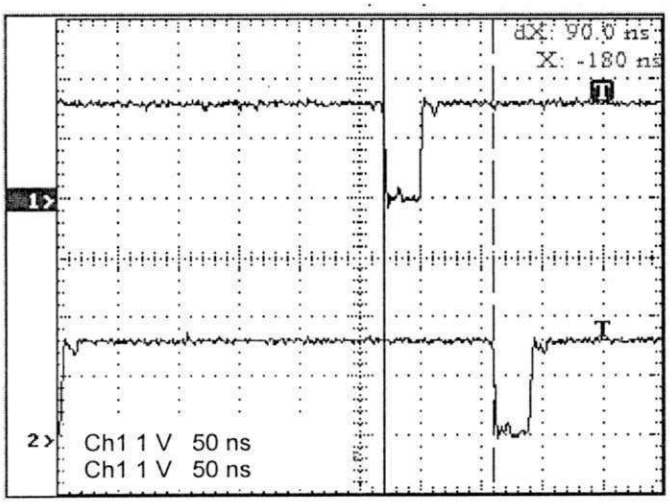


Gráfico 6.11 – Sinais FRAME e TRDY.

No Gráfico 6.12 tem-se no canal 1 o sinal IRDY e no canal 2 o sinal TRDY. Verifica-se um ciclo de leitura de configuração, o dado será lido somente quando IRDY e TRDY estiverem ativos (nível baixo).

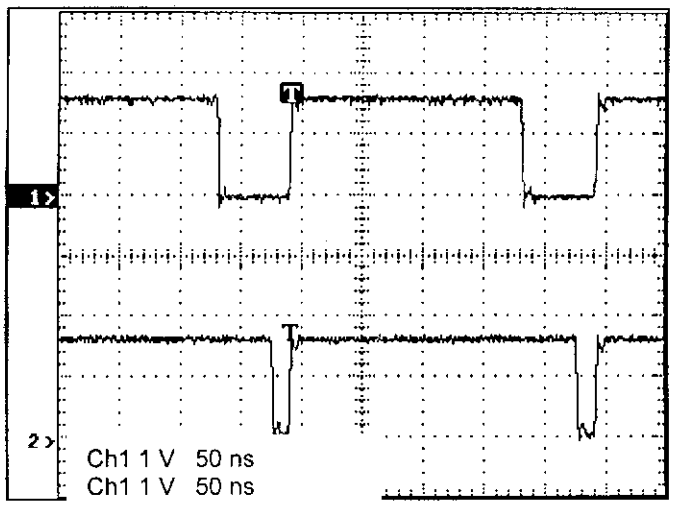


Gráfico 6.12 – Sinais IRDY e TRDY.

No Gráfico 6.13 tem-se no canal 1 o sinal TRDY e no canal 2 o sinal PAR. O sinal de Par é ativado pelo mestre do barramento durante o ciclo de endereçamento e comando e escrita dos dados, somente no ciclo de leitura de dados o sinal de PAR é ativado pela interface logo após ter disponibilizado os dados no barramento.

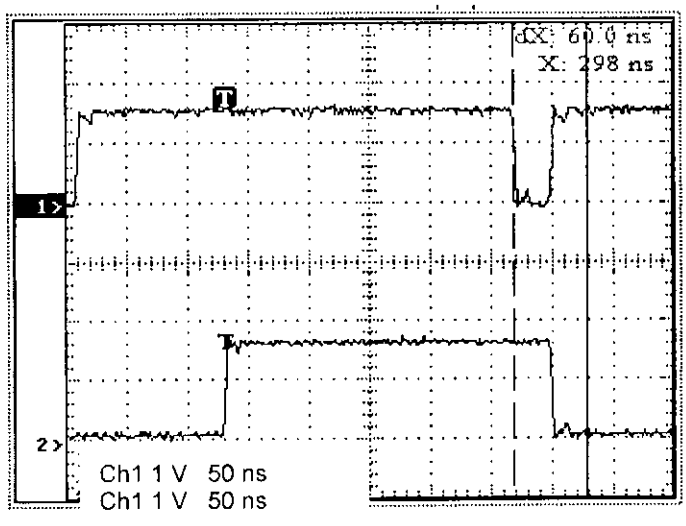


Gráfico 6.13 – Sinais TRDY e PAR.

7. CONCLUSÕES

Esta dissertação teve por meta servir de base para estudos de implementação de um projeto de Núcleo de Propriedade Intelectual de conversão da Interface PCI para a Interface OCP-IP™. Com isto pôde-se adquirir, ao longo do período, novos conhecimentos e aperfeiçoamento de metodologias de estudo e pesquisa que levaram ao aprendizado de como gerar um projeto para um IP em HDL conduzindo:

- Ao domínio básico em projeto e desenvolvimento de aplicações para barramento com interface PCI.
- Ao domínio básico em projeto e desenvolvimento de aplicações para o protocolo OCP-IP™.
- Desenvolvimento de Núcleo de Interface PCI para Interface OCP-IP™ em linguagem *SystemC™*.
- Desenvolvimento de um gerador de estímulos para PCI e um monitor para sinais OCP.

Quanto aos resultados obtidos nas simulações, foi verificado que a interface responde de forma eficiente aos ciclos de configuração e escrita na memória, onde se obteve o menor número de ciclos possíveis para realização das transações. No ciclo de leitura teve-se um baixo desempenho, vários ciclos extra de relógio em espera, visto que a interface necessitava de uma resposta do OCP™ para concluir suas transações. Uma forma de melhorar o rendimento seria a substituição dos registradores (armazena apenas um dado por vez), permitindo realizar apenas acesso no modo simples, por um banco de memória que ajudaria no desenvolvimento para implementação do acesso no modo rajada, melhorando o desempenho da interface na leitura de múltiplos dados seqüenciais.

Com relação à implementação em FPGA, obteve-se êxito, ainda que parcial, uma vez que, o projeto inicial da dissertação era de se implementar, na íntegra, todo núcleo da interface em FPGA. Devido, principalmente, ao fator tempo, implementou-

se apenas parte do núcleo referente ao ciclo de configuração da PCI. Com este fato, não foi possível fazer o levantamento completo sobre o desempenho do projeto da interface, uma vez que os testes ficaram limitados ao ciclo de configuração, porém pode-se fazer uma projeção, no sentido de se afirmar que a interface responderá de forma satisfatória, pois foram vencidos os caminhos críticos de tempo de resposta para os sinais levantados, sendo os principais "gargalos" os tempos de 7 ns para alguns sinais de resposta tsu da PCI. Todos os requisitos de tempo foram atendidos nesta etapa, de forma a satisfazer a especificação PCI v2.2 a 33 MHz.

De forma geral, se pode considerar que os resultados obtidos foram muitos bons (foram atendidos os requisitos de projeto). Além de terem sido de extrema relevância para o aperfeiçoamento acadêmico, fica como grande contribuição à instituição (UFCG), o domínio da técnica que consiste em realizar programas de acesso e configuração ao barramento PCI e implementação do mesmo em FPGA, através de seu corpo Docente.

7.1. Sugestões para Trabalhos Futuros

Como atividades futuras, que poderão ser implementadas, tem-se:

- Desenvolvimento de um IP para interface PCI 66 MHz 64 *bits* que atenda às exigências do padrão da PCI 2.3 e OCP-IP 2.0, que tenha uma memória interna para armazenamento de dados aumentando o desempenho e que aceite freqüência livre para o OCP™ a ser determinada pelo projetista, bem como a largura do barramento. A seguir, na Figura 7.1, tem-se uma ilustração da proposta.

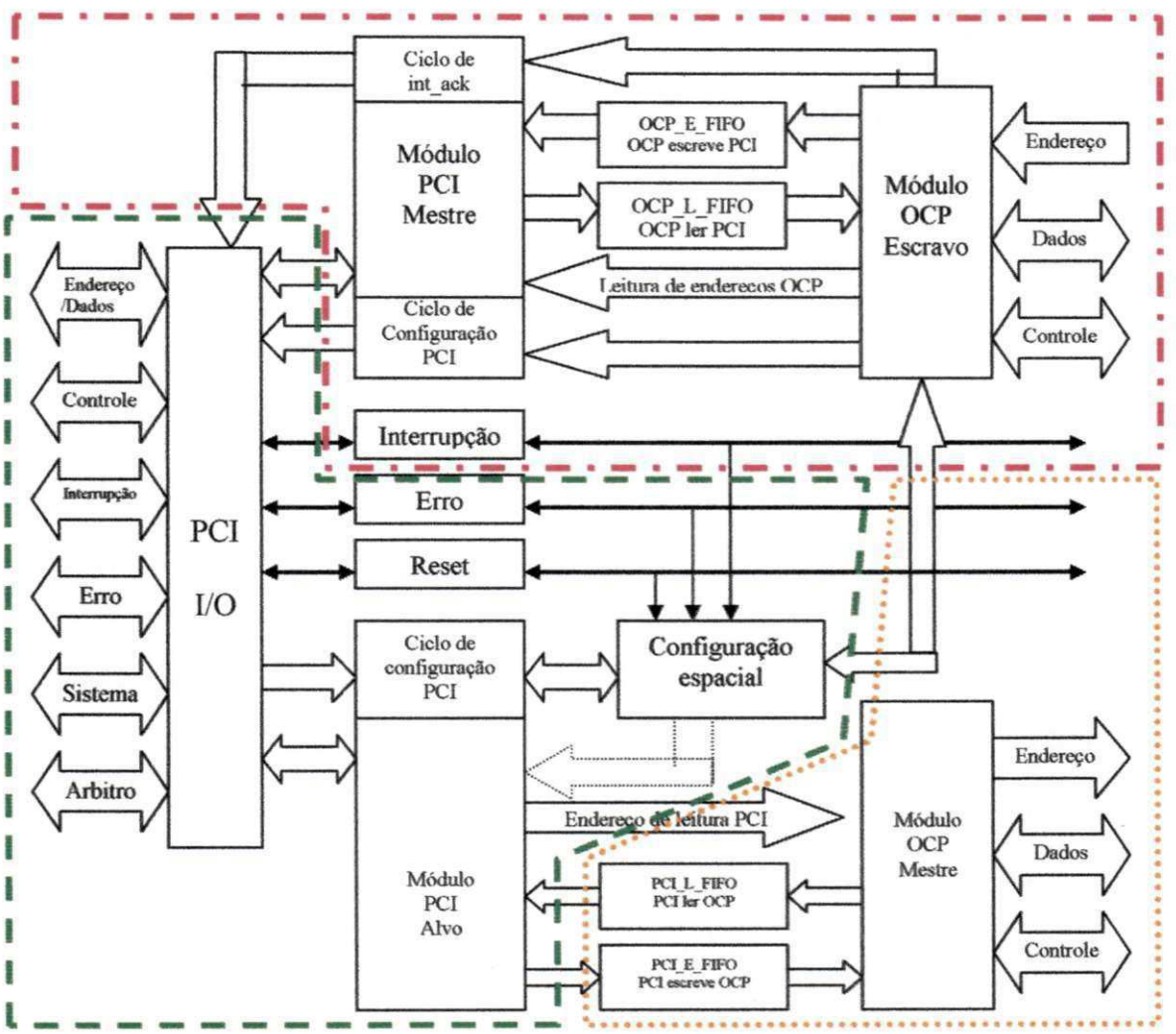


Figure 7.1 – Diagrama de Proposta para Interface PCI-OCP (Baseado na estrutura da OpenCore[16]).

Implementação Futura
Implementação em FPGA (Realizada para 33 MHz e 32 *Bits*)
Implementação em Programa (Simulada para 33 MHZ e 32 *Bits*)

- Desenvolvimento de um IP para barramento padrão *PCI Extended*

  O protocolo *Extended* PCI (PCI-X) é um projeto avançado do padrão de barramento PCI,permite que projetos com 64 *bits* de largura de barramento operem com freqüência de até 133 MHz. Este desempenho é alcançado pela implementação de um protocolo registrador-a-registrador. O PCI-X opera indistintamente no protocolo PCI V2.2 ou no próprio padrão PCI-X.

- Desenvolvimento de uma Aplicação de Dispositivo do Usuário de memória RAM, de acesso ao modo simples, ao modo rajada e interrupção para largura de barramento de 32 e 64 *bits* e freqüência de relógio 33 e 66 MHz.

- Desenvolvimento de uma estrutura completa de um Ambiente de Teste por Simulação para interface PCI rev. 2.3.

- Desenvolvimento de uma estrutura completa de um Ambiente de Teste por Simulação para o protocolo OCP-IP™ rev 2.0.

- Implementação em FPGA dos modelos apresentados e estudos para aplicações em SoC e ASIC.

8. REFERÊNCIAS

[01] Bergamaschi R. A.; Lee W. R., "Designing Systems-on-Chip Using Cores". Design Automation Conference – DAC'2000, Los Angeles, California, USA, pp. 420-425, 2000.

[02] "Programa Nacional de Microeletrônica – Design Atração, Fixação e Crescimento de Empresas de Projeto de Componentes Microeletrônicos no Brasil", disponível na Internet, http://www.mct.gov.br/Temas/info/Palestras/ProgrMic.pdf, 19.mar.2003.

[03] *Brazil IP- The Brazil IP network*, disponivel na Internet, http://www.brazilip.org, 19.mar.2003.

[04] Subversion: The Definitive Guide, Draft: Revision 5113, 26 February, 2003, disponível na internet, http://svnbook.red-bean.com/book.pdf.

[05] IBM. "The CoreConnectTM Bus Architecture", disponível na Internet, http://www.chips.ibm.com/product/coreconnect/docs/crcon_wp.pdf, jun.1999.

[06] ARM. "AMBASpecification Overview", disponível na Internet, http://www.arm.com/Pro+Peripherals/AMBA, jul.2000.

[07] SILICORE Co, "WISHBONE". Disponível na Internet, http://www.silicore.net/pdfiles/wishbone.pdf, Jul 2001.

[08] Dalpasso M. et al, "Hardware/Software IP Protection". Design Automation Conference – DAC'2000, Los Angeles, California, USA, pp.593-596, Outubro 2000.

[09] OCP - *Open Core Protocol Specification realese1.0,* disponível na Internet, http:// www.ocpip.org, 25-05-2003, www-micrel.deis.unibo.it/~benini/NOC/ OCP-IP_OpenCoreProtocolSpecification-1.0.pdf, junho, 2003.

[10] ALTERA, "FLEX 10KE PCI Development Board", *DataSheet,* disponível por ftp em www.altera.com/literature, V1.0, August 1999.

[11] ALTERA, "FLEX – 10K – Data Sheet". Disponível por ftp em www.altera.com/literature, ver 4.3, Jajuary 2003.

[12] ALTERA, "PCI Testbench – User Guide". Disponível por ftp em www.altera.com/literature, doc 2.3.0 ver 1, February 2003.

[13] PCI Local Bus Specification, rev2.2, disponível na Internet, http://www.pcisig.com, 15.fev.2003.

[14] LATTICE, " PCI Target" , *IP Data Sheet (ip1004_02)*, July 2002, disponível na Internet, http://www.latticesemi.com, 22.abr.2003.

[15] LATTICE, " Designing a 33MHz, 32-Bit PCI Target Using ispMACH Devices", (Reference Design RD1008), July 2002, disponível na Internet, http://www.latticesemi.com, 22.abr.2003.

[16] OPENCORE, "PCI Bridge IP Core", rev 5113 de 26-02-2003, disponível na Internet, http://www.opencore.org/projects/pci, julho de 2003.

[17] OPENCORES.ORG. Wishbone SoC Interconnection. Disponível na Internet, http://www.opencores.com/press/ pr_8jan2001.shtml,julho de 2001.

[18] Brunelli L., Estudo, projeto e implementação de um backend para o barramento PCI com suporte ao modo rajada, Projeto de pesquisa de Doutorado – DEE/UFCG, fevereiro de 2003.

[19] Melcher E. U. K., "SystemC" – Apostila do Curso do projeto Brazil-IP, DSC/UFCG, disponível na Internet, http://lad.dsc.ufpb.br/ip, 27.mar.2003.

[20] Melcher E. U. K., "Verificação funcional por simulação" – Apostila do curso do projeto Brazil-IP, disponível na Internet, http://lad.dsc.ufpb.br/ip, 12.maio.2003,

[21] Open SystemC™ Initiative, "SystemC 2.0.1 Language Reference Manual", revision 2.0 2003, disponível na Internet, http://www.ocpip.org/home

[22] Bhasker J., "A SystemC™ Primer", Star Galaxy Publishing, 2002, 268p.

[23] Martim G., Swan S. "System Design with SystemC™", Kluwer Academic Publishers, 2002, 216p.

[24] Ayough L. M., et al, "Verilog2SC: A Methodology for Converting Verilog® HDL to SystemC", disponível na Internet http://down.dsg.cs.tcd.ie/sendt/background/papers/systemc/verilog2sc.pdf, outubro de 2003.

[25] SystemC™ Based SoC Communication Modeling for the OCP™ Protocol, disponível na Internet, http://www.systemc.org/ ,outubro de 2003.

[26] Bergeron J., "Writinng Testbenches: Functional Verification of HDL Models", Second Edition, Klumer Academic Publishers, 476p.

[27] Palma J. C. et.al. "Interface de Comunicação de *Cores* em FPGAs", Publicação, PUCRS – FACIN, 2001, disponível na Internet, http://www.inf.pucrs.br/~moraes/papers/2002_iberchip_cores.pdf, 13.mar.2003.

[28] Palma J. C. S. "Métodos de Distribuição e Conexão de *IP Cores* para Dispositivos Programáveis FPGA". Dissertação de Mestrado, PUCRS/FACIN, Porto Alegre, 2001, disponível na Internet, http://www.inf.pucrs.br/~moraes/papers/diss_jpalma.pdf, março de 2003.

[29] Rose J. et al. "Architecture of Field-Programmable Gate Arrays". Proceedings of the IEEE, Vol. 81, No. 7, pp. 1013-1028, June, 1993.

[30] Apostila sobre Max-PlusII e FPGA/VHDL, disponível na Internet, http://diana.ee.pucrs.br/~terroso/Apostilas/ body_apostilas.html, 25.maio.2003.

[31] Cappelatti E. A., "Implementação do Padrão de Barramento PCI para Interação *Hardware/Software* em Dispositivos Reconfiguráveis". Dissertação de mestrado PUC-RG, 2001

[32] Solari E; Willse G. PCI Hardware and Software, Architecture & Design Annabooks. Fourth Edition, 1998.

[33] Calazans N. L. V. et al, "Projeto para Prototipação de um IP Soft Core MAC Ethernet". FACIN/PUCRS, Porto Alegre, RS, Brasil, p19, 2001, disponível na Internet, www.inf.pucrs.br/~moraes/papers/2001iberether.pdf, 27.mar.2003.

Arquitetura de Barramentos para Integração de Núcleos

Historicamente, no estágio inicial da metodologia de projeto de um sistema em uma única pastilha, SoC (maiores detalhes no Apêndice B), os núcleos eram projetados com várias interfaces diferentes e protocolos de comunicação. Para contornar o problema de integração entre núcleos, foram criados padrões para estruturas internas ao circuito integrado tais como a *CoreConnect*, AMBA e *WISHBONE*[01].

A arquitetura *CoreConnect* da IBM fornece três barramentos para interconectar núcleos e lógica personalizável:

- Barramento Local do Processador (*Processor Local Bus - PLB*): usado para interconectar núcleos com alto desempenho, grande largura de banda, tais como o *PowerPC*, controladores DMA e interfaces de memória externa.
- Barramento Periférico (*On-Chip Peripheral Bus - OPB*): usado para interconectar periféricos que trabalham com baixas taxas de transmissão de dados, tais como portas seriais, portas paralelas, UARTs (*Universal Asynchronous Receiver Transmitter*) e outros núcleos com pequena largura de banda.
- Barramento de Registradores de Controle de Dispositivos (*Device Control Register Bus - DCR*): caminho de baixa velocidade, usado para passar configuração e informações de estado entre o núcleo processador e outros núcleos.

A arquitetura *WISHBONE* é análoga a um barramento de microcomputador, sendo que:

- Oferece uma solução flexível para integração que pode ser facilmente adaptada a uma aplicação específica.
- Oferece uma variedade de ciclos de acesso ao barramento e de largura de caminhos de dados para atender a diferentes sistemas.
- Permite que os núcleos sejam projetados por vários fornecedores. É um barramento flexível e simples. É completamente aberto, pois sua criadora, a empresa *Silicore Corporation*, tornou-o aberto ao domínio público[17].

APÊNDICE B. – Sistema em uma Única Pastilha

Hoje em dia a realidade de mercado em projeto de VLSI é caracterizada pelo pequeno tempo para comercializar, com grande custo e um alto desempenho. Este ambiente exigente está forçando as mudanças de princípio no modo como sistemas de VLSI são projetados. O uso de blocos IP pré-projetados ficou essencial para o projeto de Sistemas em uma Única Pastilha (*System on a Chip – SoC*) a fim de se construir a exigida complexidade em pouco tempo para comercialização.

Na Figura B.1 ilustra-se uma arquitetura genérica de um SoC. Os núcleos são integrados através de uma rede de interconexão comercial ou adaptada, com um controlador e funções de interface com o meio externo. Caso os núcleos sejam obtidos de diferentes fontes, a integração dos módulos e os testes do sistema podem ser difíceis, podendo até haver necessidade de que os núcleos sejam reprojetados, para adequá-los a um protocolo de interface comum.

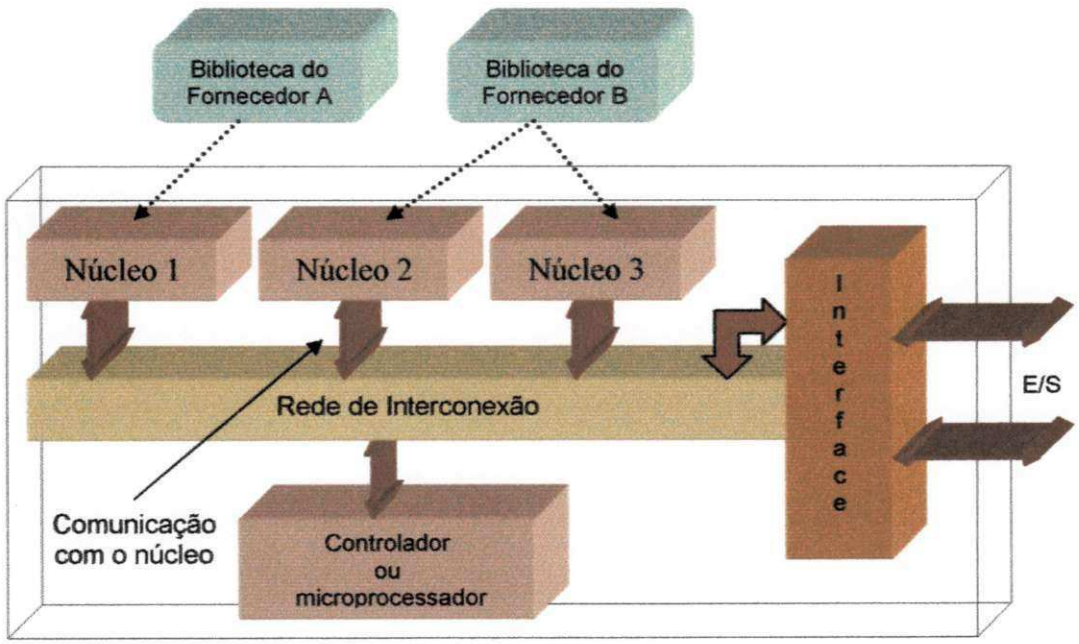


Figura B.1 – Arquitetura de um SoC genérico.

Porém, na prática, o uso de núcleos ainda não se tornou uma realidade plena por várias razões, dentre elas destacam-se as seguintes:

- Arquitetura do sistema é uma tarefa complexa, que exige dos projetistas responder perguntas como, (1) que CPU deve ser usada? (2) que funções devem ser feitas em Dispositivos ou em Programas? (3) que taxa de MIPS deve o sistema alcançar e é esta suficiente para os aplicativos de destino? etc.
- A integração de núcleos em um SoC é, amplamente, um processo manual propenso de erros, porque exige dos projetistas um completo entendimento da funcionalidade, interfaces e características elétricas de núcleos complexos, como microprocessadores, memória controladoras, árbitros de barramento, etc.
- Alcançar um fechamento de contagem de tempo completo é muito difícil devido à grande complexidade do sistema. Em muitos casos exige ajuste fino nos núcleos o que afeta sua reusabilidade.
- O projeto físico de grandes sistemas gera um problema significativo. Ainda que cada núcleo seja predefinido, quando postos juntos o itinerário de ligações pode gerar grandes imprevistos como o ruído e uma dupla capacitância que degradam a apresentação do sinal.
- A verificação do sistema é um dos "gargalos" importantes. Ainda que os núcleos estejam pré-verificados, não quer dizer que o sistema inteiro funcione quando eles forem postos juntos. Várias interfaces e contagem de tempo dos eventos podem fazer o sistema falhar, embora os núcleos individuais estejam corretos. A verificação formal corrente, como também as técnicas de simulação por programas não têm a capacidade total de avaliar os sistemas de grande velocidade, executando em tempo real.
- Unir IPs de diferentes provedores para ser integrado no mesmo SoC torna-se uma tarefa difícil, por falta de um único padrão estabelecido nas indústrias e/ou a falta de ferramentas de síntese de interface eficientes.
- A integração de Dispositivos e Programas é outro problema importante que afeta diretamente o tempo para comercialização, enquanto as partes de Dispositivos são mais estáveis, as partes do processo de projeto são

normalmente mais atrasadas. As ferramentas de projeto auxiliado por computador (CAD) ajudam a acelerar a modelagem e simulação.Porém, os Programas não são diretamente dirigidos para projetos de sistemas usando núcleos. Os CAD tradicionalmente enfocaram assuntos de projeto de nível mais baixo, como síntese, contagem de tempo, plano e simulação. Recentemente, foi desenvolvida uma modelagem usando variações de linguagens de alto nível.

B.1 Arquitetura de SoC alvo

Historicamente, no início dos projetos de SoCs, os núcleos eram projetados com muitas interfaces e diferentes protocolos de comunicação. A integração de tais núcleos em um SoC freqüentemente exigiam sub-utilização da lógica a ser implementada. A fim de evitar este problema foram desenvolvidas estruturas padrões para barramentos de circuito integrado.

Na Figura B.2 apresenta-se um sistema em uma única pastilha (SoC) baseado na arquitetura de barramento de núcleos conectados (*CoreConnect*). Embora os núcleos sejam projetados para terem uma interface com os barramentos quase direta, o projetista ainda tem que conectar centenas de ligações e definir os parâmetros para todos os núcleos a fim de criarem um nível superior correto.

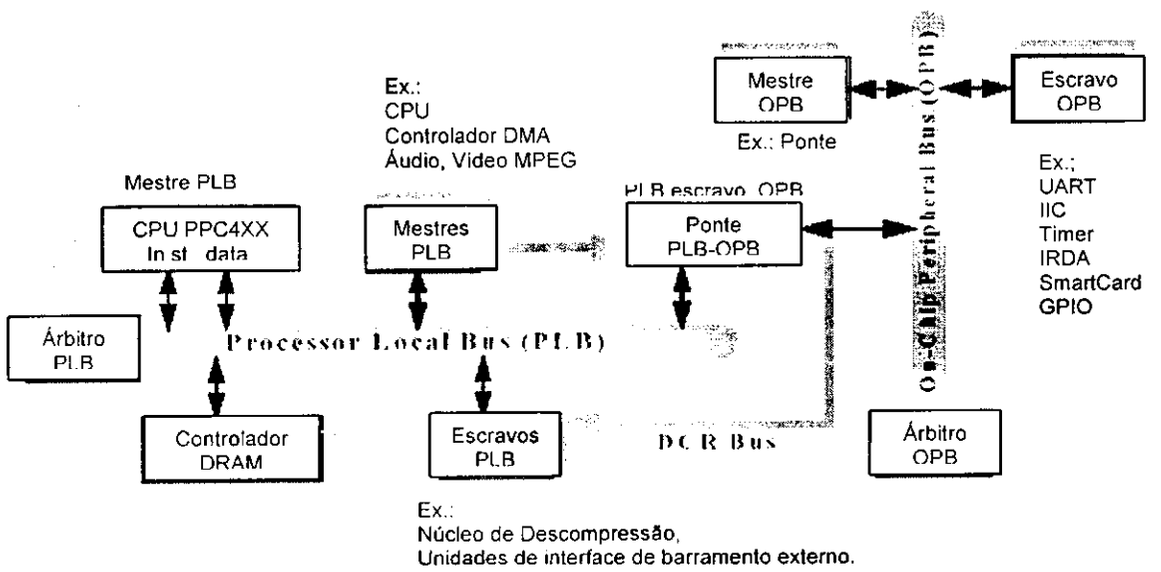


Figura B.2 – Arquitetura de SoC usando barramento de núcleos conectados[01].

Para o projeto de SoC deve-se seguir vários passos, dentre eles destacam-se os seguintes:

- Definir a funcionalidade desejada a ser implementada em todos os núcleos. Este processo é uma combinação de identificar pré-projeto dos núcleos a serem usados com ou sem modificação e identificar novos núcleos para serem projetados. Estas escolhas são tipicamente feitas dentro das restrições de uma dada relação preço/desempenho. O projetista deve escolher entre barramentos de 32, 64 ou 128 *bits* e características de processadores, trocas entre Dispositivos e Programas, etc. Em muitos casos, as escolhas só podem ser avaliadas mais tarde, depois de simulação e análise.
- Entender a funcionalidade de todos os pinos em todos os núcleos e determinar quais pinos devam ser conectados juntos. Embora este problema seja aliviado com o uso de arquiteturas de barramento predefinidas, é ainda um processo de trabalho manual intensivo, exige dos projetistas uma leitura da documentação afim que entenda a função de todos os pinos em todos os núcleos. Mesmo conectar uma única pinagem de barramento padrão em um único núcleo pode demorar semanas se for descrito de forma incongruente com as especificações. Qualquer integrador de SoC chamará o suporte do projetista do IP para agrupar o núcleo, ou pior, incorretamente conectará este pino, e achar este problema mais tarde durante a simulação e depuração é demorado.
- Definir as prioridades de pedidos de interrupção nos barramentos para os mestres e o processador. As prioridades são específicas da aplicação e podem afetar dramaticamente o desempenho do sistema.
- Interconectar a pinagem de acordo com as suas prioridades, quando possível, deixando espaço no projeto para mudança de interrupções (por exemplo, pode-se deixar ser decidido em uma fase posterior as prioridades de interrupção, ou que uma nova interrupção seja adicionada).
- Definir quais núcleos podem acessar memória por uma controladora de DMA e executar a tarefa de canal de acordo com a prioridade dos dispositivos solicitantes. Quando o número de DMA requeridos exceder o número de

canais de DMA, o canal compartilhado pode ser utilizado e/ou pode ser adicionado uma controladora de DMA adicional.

- Definir mapas de endereço para todos os núcleos e passar os valores como parâmetros para cada núcleo, assegurando que um conflito de endereço não seja criado entre quaisquer dos núcleos.
- Definir os domínios de relógio válido no circuito integrado e conectar os relógios certos para cada núcleo, como também a lógica de controle de *relógio* apropriado.
- Inserir qualquer lógica de encapsulamento exigida entre núcleos.
- Definir todas as E/S do circuito integrado, projetar a lógica de E/S que inclua qualquer compartilhamento de pinagens e lógica de controle de teste requerida pelo fabricante.
- Checar se os núcleos que estão sendo usados são compatíveis com respeito à freqüência operacional, largura de *bit*, número da versão, etc.
- Documentar o sistema (por exemplo, mapas de endereço, prioridades de interrupção, canais de DMA, E/S de *Chip*, etc.) para uso futuro por programa e desenvolvedores de placa de circuitos impressos. Esta lista não exaustiva é suficiente para mostrar que no projeto ainda existe um número grande de tarefas complexas que precisam ser consideradas. Estas tarefas são resolvidas manualmente porque hoje metodologias e ferramentas são ineficientes e propensas a erro.

B.2 Automatizando a Integração de SoC

Com a finalidade de automatizar muitas das tarefas manuais descritas na seção anterior, foram desenvolvidas novas ferramentas de projeto como por exemplo a chamada "Coral" ou a ferramenta "SocCreator" do OCP^{TM}, que contêm novos algoritmos e metodologias para projetar SoC usando núcleos baseados no conceito de "projeto virtual" sintetizável.

Estas ferramentas acrescentam produtividade criando um nível de abstração no qual os projetistas trabalham seus projetos de SoC no meio virtual, escondendo toda a complexidade desnecessária associada aos núcleos, o que diminui erros e aumenta a produtividade.

Estes programas e suas metodologias associadas são baseados nos seguintes elementos:

- Projeto virtual;
- Encapsulamento da interface;
- Núcleo e Propriedades dos pinos;
- Engenharia de Interconexão;
- Engenharia de Síntese Virtual para Real;
- Engenharia de Configuração.

Nota: Para maiores informações sobre o tema consulte as referências Bergamaschi[01] e Palma[28].

APÊNDICE C. Descrição Geral de um FPGA

O FPGA (*Field Programmable Gate Array*) foi introduzido em 1985 pela empresa Xilinx, desde então, grandes variedades de FPGAs foram desenvolvidas por várias outras companhias como as americanas Altera, Actel, e outras empresas como Atmel, *Plessey, Plus Logic, Advanced Micro Devices (AMD), Quicklogic, Algotronix, Concurrent Logic,* e *Crosspoint Solutions*, que seguem expandindo suas fatias no mercado de *chips*. O impacto mais significativo desta inovação tecnológica foi a introdução de uma rota alternativa, mais barata em muitos casos, porém sempre de menor desempenho para a mesma função de um *chip* dedicado.

Os FPGAs aparecem como uma solução intermediária entre os circuitos integrados genéricos (por exemplo: microprocessador) e circuitos integrados dedicados para aplicações específicas (ASICs). Esta solução apresenta um bom equilíbrio entre custo *versus* versatilidade[29]. Estes circuitos integrados configuráveis podem ser personalizados como diferentes ASICs[29]. Esta tecnologia permite o projeto, teste e correção de circuitos integrados com um baixo custo de implementação de protótipos[29].

Muitas aplicações computacionais necessitam alteração freqüente de sua funcionalidade ou grande flexibilidade de comportamento. Isto é atualmente possível não apenas via implementação em programa, mas também em dispositivo, graças à existência de dispositivos reconfiguráveis.

No caso de implementações em dispositivo existe um dispositivo subjacente, normalmente composto por um processador com conjunto de instruções (*ISP - Instruction Set Processor*) associado a uma memória. ISPs podem ser programados para executar uma ou mais aplicações específicas preenchendo a memória de instruções com os programas que implementam as aplicações.

No caso de implementação em dispositivo, as aplicações flexíveis são obtidas principalmente através de uso de dispositivos FPGAs. De fato, os FPGAs modificaram a distinção tradicional entre o dispositivo e o programa, visto que sua funcionalidade em dispositivo pode ser alterada de forma total ou parcial ou até mesmo dinâmica.

Note-se que os FPGAs:

- Foram viabilizados pelo avanço das tecnologias CMOS em silício e são beneficiados continuamente por esta.
- São alternativas tornadas simples e baratas para implementação em baixo volume devido ao poderio e baixo custo do programa de EDA associado ao "mapeamento tecnológico" ou programação do dispositivo.
- São importantes para *breadboarding*[19] de alguns sistemas, antes do projeto elétrico e físico de um ASIC ou SoC.
- Têm sempre desempenho elétrico inferior aos ASICs customizados e fabricados nas *foundries*[20], porém são mais baratos para séries de 100 ou 1000 peças, dependendo da complexidade do projeto.
- Na medida que evoluem e incorporam acima de 1 milhão de portas lógicas, ocupam a capacidade de produção das *foundries* de silício.
- Evitam a interação do usuário dos FPGAs com a *foundry* diretamente; a detentora do *copyright* das máscaras do FPGA e do *copyright* do programa de automação do projeto para FPGAs faz esta encomenda a poucas (tipicamente duas) *foundries* e atende a dezenas de milhões de clientes.

[19] Montagem funcional de circuito eletrônico.
[20] Empresas envolvidas na fabricação (ou fundição) do circuito integrado.

C.1 Arquitetura de um *chip* FPGA

A arquitetura genérica de FPGAs é ilustrada na Figura C.1, que consiste em uma matriz de elementos agrupados, células lógicas ou blocos lógicos, alocados, configuráveis, que podem ser interconectados, por barramentos de interconexão configuráveis. Semelhante a uma PAL (*Programmable Array Logic*), as interconexões entre os elementos são implementadas por blocos de chaves configuráveis pelo usuário. Através de blocos de entrada/saída configuráveis é realizada a interface com o mundo externo. Em geral, a funcionalidade destes blocos, assim como o seu roteamento, são configuráveis por *software*. A palavra *Field* indica que a configuração do circuito pode ser feita pelo usuário final sem a necessidade da utilização de recursos de *foundries*.

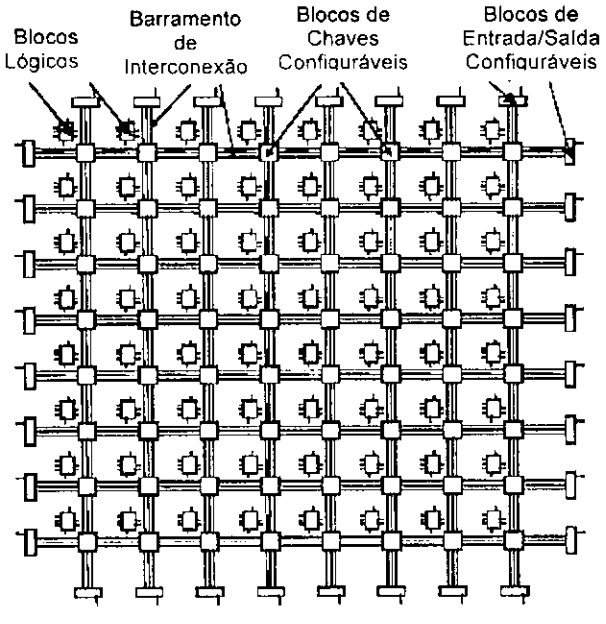


Figura C.1 – Arquitetura de um FPGA genérico[29].

C.2 Definição de Blocos Lógicos

As funções lógicas são implementadas no interior dos Blocos Lógicos. Em algumas arquiteturas os Blocos Lógicos possuem recursos seqüenciais tais como *flip-flop* ou registradores. O fabricante Xilinx chama seu Bloco Lógico de CLB (*Configurable Logic Block*), enquanto que a Actel usa o termo LM (*Logic Modules*), já a Altera utiliza o termo LE (*Logic Element*), conforme descrito na Figura C.2.

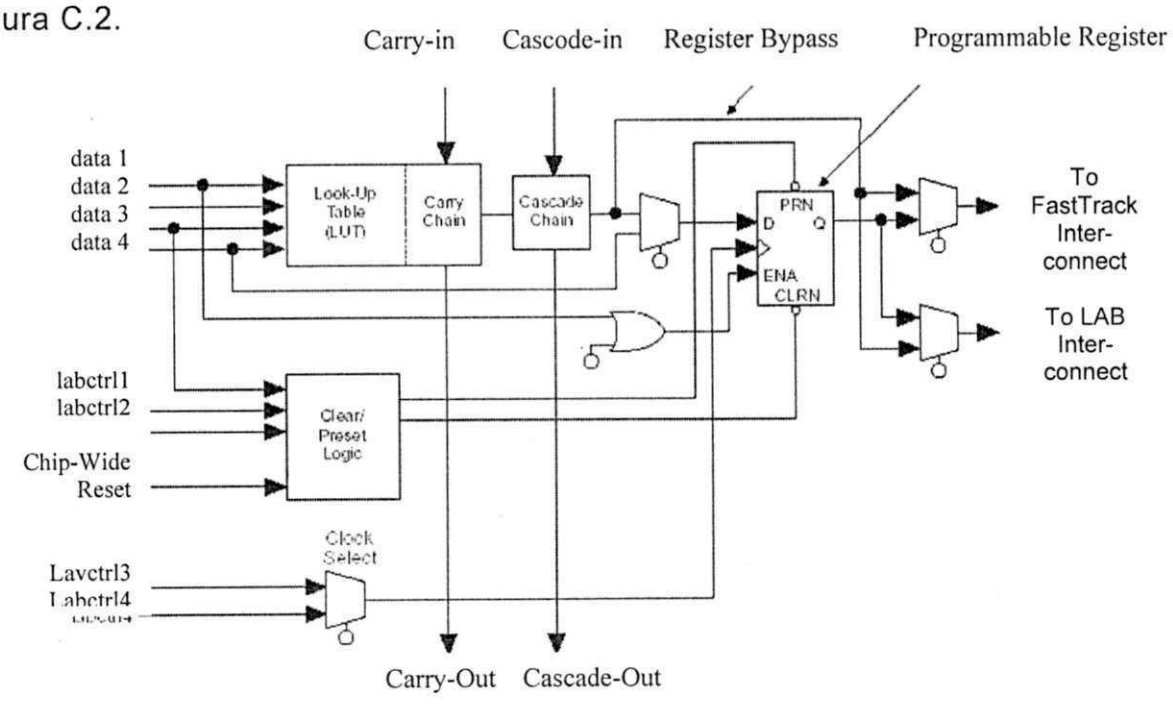


Figura C.2 – A estrutura de um Elemento Lógico dentro da FPGA – Altera[11].

C.3 Definição de roteamento

A interconexão entre os blocos é feita através de uma rede de cinco ou mais camadas de metal. As conexões físicas entre os fios são feitas ora com transistores de passagem controlados por *bits* de memória ora com chaves de interconexão nas matrizes de conexão.

Os recursos de roteamento de algumas FPGAs possuem:

- **Conexões globais**

  Formam uma rede de interconexão em linha e colunas de cinco fios de metal cada, que se ligam através de chaves de interconexão. Esta rede circunda os blocos lógicos (*CLBs*) e os blocos de E/S (*IOBs*);

- **Matrizes de conexão**

  As matrizes de conexão são chaves de interconexão que permitem o roteamento entre os Blocos Lógicos através das Conexões Globais (Figura C.3). Estas conexões são programáveis na fase de roteamento automático, executada pelo programa de projeto do fabricante ou manualmente.

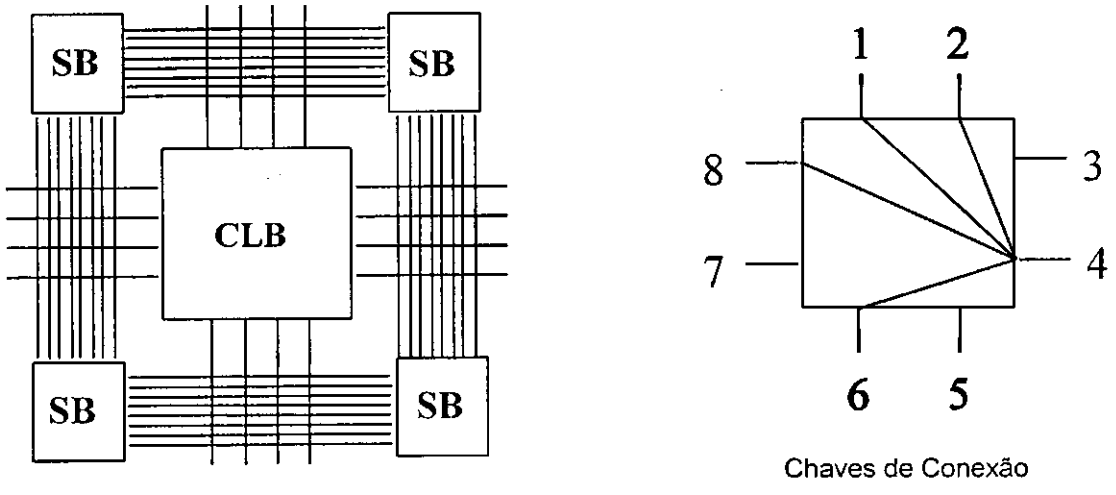


Figura C.3 – Representação de uma Matriz de Conexão[30].

- **Conexões diretas**

São conexões entre CLBs vizinhos e permitem conectar blocos com menor atraso, pois não utilizam os recursos globais de roteamento.

- **Linhas longas**

São conexões que atravessam todo o circuito sem passar pelas matrizes de conexão e são utilizadas para conectar sinais longos e com restrições de distorções entre múltiplos destinos (Figura C.4).

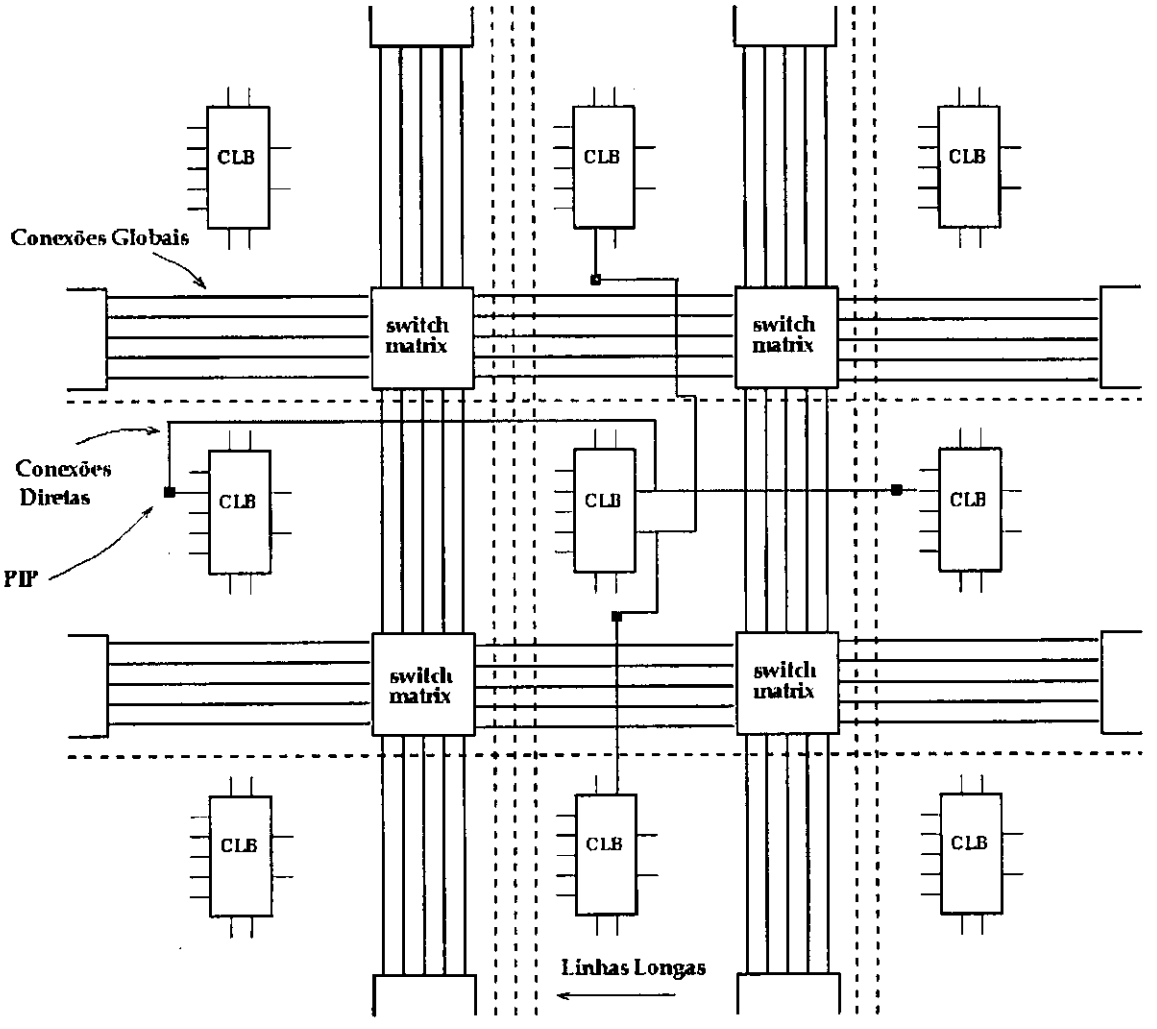


Figura C.4 – Roteamento de uma FPGA[30].

C.4 Definição de elementos entrada e saída (E/S)

Um Elemento de E/S contém um *buffer* de E/S bidirecional e um registrador que pode ser usado como um registro de entrada para dados externos que exige um tempo de configuração rápida, ou como um registro de saída para dados que exige tempo rápido de apresentação de saída. Para implementação bidirecional de E/S o Compilador usa a opção de inversão programável para inverter sinais. Na Figura C.5 ilustram-se os registros de E/S bidirecional da placa FLEX 10K.

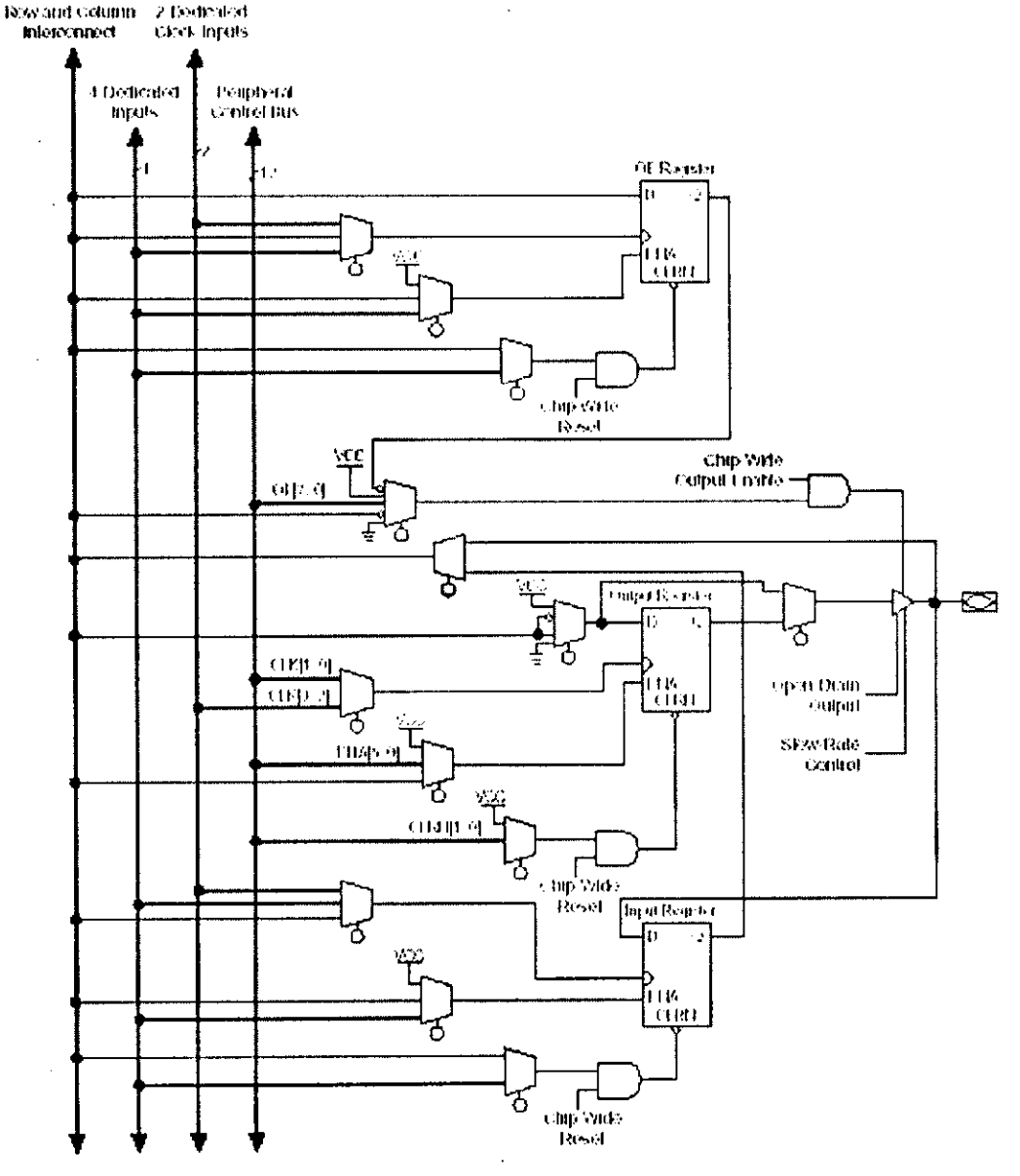


Figura C.5 – Representação de elementos entrada e saída[11].

APÊNDICE D. – Barramento de Interconexão de Componentes Periféricos

Nesta seção é descrito o Barramento de Interconexão de componentes Periféricos (*Peripheral Component Interconnect Bus – PCI*). Inicialmente apresentam-se as características gerais deste barramento, seguido pelos conjuntos de sinais presentes neste e finalizando com detalhes dos ciclos de operações do barramento. Esta compreensão do funcionamento do barramento PCI é fundamental para o desenvolvimento do núcleo IP.

Na Tabela D.1 apresentam-se as principais características de algumas arquiteturas que antecederam o padrão PCI.

Tabela D.1 – Arquitetura de barramentos mais comuns em computadores PC[31].

Arquitetura	Sistema típico	Largura do barramento (bits)	Freqüência de operação (MHz)	Conectores de expansão	Característica
ISA (1984)	IBM PC AT	8/16	8	7	Foi o padrão mais comum, em desuso atualmente
SCSI-1 (1986) Fast SCSI, Ultra SCSI, Ultra-2 SCSI, (SCSI-3)	PCs de alto desempenho a partir do 80386	8/16	5/10/20/40	7/15	Comum, muitos padrões, freqüências de operação e larguras de barramentos de dados
MCA (1987)	IBM OS/2	16/32	10	3-7	Raro, antigo padrão da IBM
EISA (1988)	PCs baseados em 80386	32	8	7	Parcialmente compatível com ISA
VLB (1992) (VESA)	PCs baseados em 80486	32/64	40/50	2	Foi comum em computadores baseados em i486
PCI (1992)	PCs baseados em Intel Pentium e estações de trabalho	32/64	33/66/133	4, mais pontes	Encontrado nos padrões: CompactPCI, PC/104-Plus, Small PCI, CardBus, PCI Mezzanine Card (PMC)

D.1 Características gerais do barramento PCI

O barramento PCI foi proposto pela empresa INTEL em junho de 1992 e tornou-se um padrão industrial aberto, orientado pelo PCI SIG[21]. Em abril de 1993

[21] PCI SIG *Special Interest Group*, **http://www.pcisig.com** [13].

chegou a versão rev2.0 do PCI. A revisão usada neste trabalho é a especificação denominada PCI rev2.2, que data de dezembro de 1998. Atualmente, o padrão PCI encontra-se na especificação PCI rev2.3.

O barramento PCI é de alto desempenho, suporta 32 ou 64 *bits* multiplexados de linhas de endereçamentos e dados. Possui paridade nos barramentos de dados e endereços. Empregado para conectar componentes em uma placa mãe, e componentes desta às placas de extensão de um computador. Isto torna este padrão independente do processador, pois o processador tem seus sinais acoplados ao barramento PCI através de uma Ponte[22] que conecta o barramento PCI ao barramento do processador (*PCI-to-host-bridge*), fazendo a interface entre eles, o que permite que o PCI seja utilizado por diferentes fabricantes de computadores.

Na Figura D.1 é mostrado um sistema genérico de um barramento PCI no qual os periféricos PCI podem ser conectados diretamente ao barramento PCI, sendo sua detecção automática, conectar e usar, (*PNP- Plug-and-Play*). Uma vez que a ponte é um componente presente na placa mãe, qualquer processador poderá ter acesso a todos componentes PCI do computador. A ponte *host-to-PCI* isola o conjunto processador/*cache*[23] dos periféricos o que torna o sistema inalterado, independente do processador, mesmo que se venha a substituir o processador por um outro mais rápido.

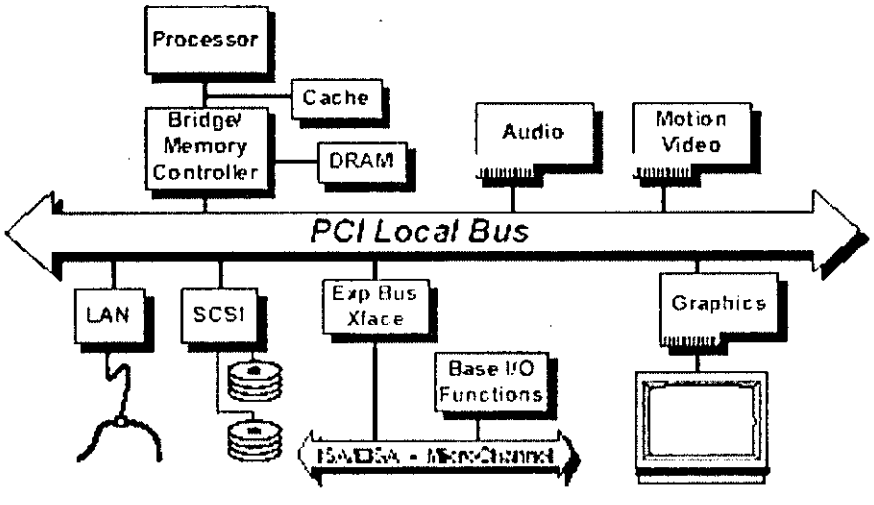


Figura D.1 – Diagrama em bloco de um sistema PCI[13].

[22] Usa-se freqüentemente a palavra inglesa "***Bridge***" para designar um tipo de lógica que conecta um barramento PCI de um computador ao barramento do processador, permitindo qualquer processador ter acesso a todos os componentes PCI do computador, desta forma um agente pode acessar outro agente em outro barramento, mas nesta dissertação, usa-se a palavra portuguesa "**Ponte**".

[23] *Cache*, seção de memória de alta velocidade que armazena as próximas instruções/dados a serem executados por um processador (para acelerar a operação).

O PCI permite acesso ao barramento para transferência de dados (leitura/escrita) no modo simples e no modo rajada[24]. Barramentos anteriores ao PCI só tinham acesso no modo simples, em que havia a necessidade de se informar um novo endereço a cada transferência de dados. No modo rajada são realizadas múltiplas transferências de/para endereços consecutivos, tendo um comprimento indefinido, permanecendo até que um dispositivo PCI solicite o final da transferência.

Na Tabela D.2 apresenta-se o desempenho do barramento PCI. Apesar de haver normas para freqüências de 33, 66 e 133 MHz, neste trabalho adotou-se apenas a freqüência de 33 MHz.

Tabela D.2 – Desempenho do padrão PCI na transferência de dados.

Freqüência do sistema (MHz)	Largura do barramento (bits)	Taxa de transferência no pico (MB/s)
33,33	32	133
33,33	64	266
66,66	32	266
66,66	64	533
133,33 *	64	1066

* *PCI Extended (PCI-X)*

D.2 Operações Conectar e Usar do barramento PCI[25]

O termo Conectar e Usar, refere-se à capacidade do sistema computacional de determinar automaticamente os recursos necessários para cada dispositivo instalado no barramento PCI. Estes recursos são mapeados de forma a evitar conflitos no sistema, o que acontece com os dispositivos que não são PNP, os chamados legados, que necessitam ser configurados por *jumpers*[26]. São exemplos de dispositivos legados os controladores de portas serial/paralela e alguns controladores conectados aos conectores ISA.

---

[24] Usa-se freqüentemente a palavra inglesa "***burst***", mas nesta dissertação, usa-se a palavra portuguesa "**rajada**", para designar um tipo de transferência de dados em alto desempenho, que consiste em uma fase de endereçamento seguida por uma ou mais fases de dados.

[25] As seções "Operações Conectar e Usar" e "Arbitragem do Barramento" foram extraídas da dissertação de Cappelatti [31].

[26] ***Jumper***, (a) ponte, fio, conector; conexão temporária de fio em uma placa de circuito. (b) ponte selecionável = circuito ou dispositivo cujas opções podem ser selecionadas através de conexões de fio.

Existem quatro tipos de registradores que são importantes para o sistema PNP sendo estes:

- ***Device* e *Vendor ID***. São empregados juntos para identificar o fabricante de um dispositivo em particular. O *Vendor ID* é atribuído pelo PCI SIG, o que garante um único número de identificação para cada fabricante. Cada fabricante atribui a seu *Device ID* valores que garantem um único identificador para cada um de seus produtos.

- ***BAR***. Possui um duplo propósito. A partir da inicialização, o BAR identifica os recursos do sistema necessários para cada dispositivo. Cada dispositivo pode utilizar até seis registradores BAR para identificar até seis espaços de endereços individuais de vários tamanhos para serem empregados pelo dispositivo. O valor inicial armazenado em cada BAR por ocasião da inicialização, identificando qual o tipo de recurso está sendo solicitado. O *bit* menos significativo classifica o dispositivo como sendo de entrada/saída ou um espaço de memória. Os *bits* restantes indicam quanto espaço é solicitado. Para completar a configuração PNP o *host* escreve o valor do endereço base para o dispositivo no BAR.

- ***Interrupt Line Register***. Este registrador é utilizado para identificar a linha de interrupção a ser utilizada pelo dispositivo PCI compatível. Em sistemas Computador Pessoal (PC), este registrador recebe o valor da interrupção de *hardware* utilizada pelo dispositivo.

Outros registradores do espaço de configuração, que não são necessários para as funções PNP, contêm outras informações: O *Class Code*, serve para identificar o tipo de dispositivo; os registradores para a requisição do barramento, como *Minimum Grant* e *Latency timer* (temporizador de latência) são parâmetros necessários para o arbitramento do barramento PCI.

PCI *BIOS*

A configuração PNP dos dispositivos conectados ao barramento PCI é controlada pelo PCI BIOS no momento da inicialização do sistema. Na Figura D.2 apresenta-se o diagrama do PCI BIOS.

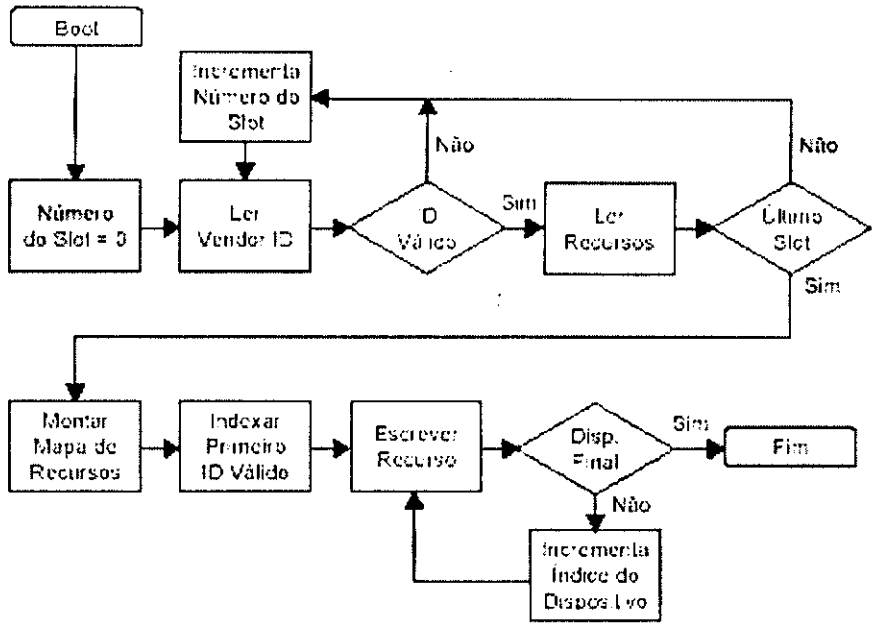


Figura D.2 – Fluxograma da configuração PNP do PCI BIOS[31].

O BIOS inicia a seqüência de configuração PNP endereçando individualmente cada conector PCI da placa mãe, de maneira a determinar se um dispositivo válido está nele inserido. A verificação do dispositivo é feita através da leitura do registrador *Vendor ID*. Um valor FFFF (Hexadecimal) (devido a resistores *pull up*) indica um componente inválido, ou nenhum instalado. Se outro valor for retornado pelo registrador *Vendor ID*, o sistema fará a leitura do BAR e do Registrador de Interrupção do dispositivo.

Este procedimento se repete até que todos os *slots* PCI tenham sido acessados, e com isso o sistema torna-se informado sobre *I/O*, memória e interrupções solicitados individualmente pelos dispositivos conectados ao barramento.

Assim, o sistema gera um mapa para cada dispositivo, garantindo a exclusividade deste espaço a cada um deles, evitando conflitos. O PCI BIOS escreve o endereço base de cada espaço atribuído no BAR apropriado e a interrupção é escrita no Registrador de Interrupção. Terminadas estas tarefas, encerra-se a configuração PNP do dispositivo.

D.3 Arbitragem do Barramento

Toda operação PCI ocorre entre dois componentes distintos: o mestre[27] e o alvo[28]. O mestre é o dispositivo que inicializa e gerencia o barramento PCI, podendo ser o processador ou uma ponte. Já o alvo é o dispositivo de destino, aquele que responde a um acesso, como por exemplo, uma controladora de vídeo, memórias e E/S. Na Figura D.3 ilustra-se estes componentes.

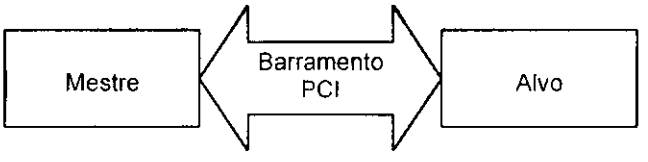


Figura D.3 – O mestre comanda o barramento e troca informações com o alvo.

Somente um mestre pode controlar o barramento PCI em cada ciclo de barramento que esteja sendo executado. Outro mestre, que não aquele ativo no momento, pode solicitar a arbitragem[29] através do sinal LOCK#. Esta arbitragem deve ser feita por qualquer dispositivo mestre que necessite do barramento. Esta solicitação é feita para o árbitro central, que é a entidade que está gerenciando o barramento no momento da solicitação.

Dois sinais estão envolvidos na solicitação da arbitragem do barramento: REQ# e GNT#. O sinal REQ# é o sinal utilizado por um mestre para solicitar o barramento, que recebe como resposta positiva à solicitação a ativação do sinal GNT#. O dispositivo mestre que deseja gerenciar o barramento deve ativar o sinal REQ# somente quando ele estiver pronto para iniciar, imediatamente, um novo ciclo. As linhas dos sinais REQ# e GNT# são individuais para cada dispositivo e roteadas ponto a ponto até o árbitro central, que é uma ponte. A ativação dos sinais REQ# e GNT# são sincronizadas com o sinal de relógio (CLK).

Existe mais de um algoritmo de distribuição de arbitragem pela ponte no PCI. Por exemplo, supondo que existem 3 dispositivos que podem operar como mestre (A,

---

[27] Usa-se freqüentemente a palavra inglesa "***master***" para designar a "unidade controladora, origem", mas nesta dissertação, usa-se a palavra portuguesa "**mestre**".
[28] Usa-se freqüentemente a palavra inglesa "**target**" para designar a "unidade controlada, destino", mas nesta dissertação, usa-se a palavra portuguesa "**alvo**".
[29] Dá-se o nome de arbitragem ao controle e gerenciamento do barramento PCI durante os ciclos de barramento.

B e C), o árbitro central garantirá primeiramente ao dispositivo **A** o acesso ao barramento. Se **A** não quiser assumir o barramento ou já concluiu sua utilização, o árbitro central passará o controle para **B**. Após **A** e **B** concluírem suas tarefas o árbitro passará o controle para **C**. Dado que **A**, **B** e **C** já utilizaram o barramento, o ciclo volta para **A**, e assim por diante.

Outro algoritmo associa a cada dispositivo um nível de prioridade. Quando dois mestres requisitarem o barramento simultaneamente, aquele que tiver a maior prioridade assume o controle do barramento. Para garantir que os outros mestres tenham seu direito garantido, o árbitro central não cede a um mestre o controle do barramento sem que todos os outros tenham sido servidos.

Existem outros protocolos de arbitragem, porém todos têm o cuidado de não deixar de atender à solicitação de algum mestre (latência muito grande), de garantir o tempo necessário na utilização do barramento (garantir latência) e de, independente da solicitação, oferecer o controle do barramento a qualquer mestre.

D.4 Descrição dos sinais básicos do barramento PCI

Como já mencionado, o PCI tem duas larguras de barramento, 32 e 64 *bits*, e freqüências de operação de 33 e 66 MHz. Os dispositivos PCI com barramento de 32 *bits* possuem 124 pinos, e os com barramento de 64 *bits* tem 188 pinos. Um dispositivo alvo, 32 *bits*, deve possuir no mínimo 47 pinos disponíveis à interface do barramento (todos os pinos obrigatórios menos GNT# e REQ#), e um dispositivo mestre deve ter um mínimo de 49 pinos (todos os pinos obrigatórios). Para se chegar ao número total de pinos, devem ser considerados diversos pinos de alimentação 3,3 Vcc, 5,0 Vcc e terra[32].

Na Figura D.4 ilustra-se os sinais obrigatórios (para mestre e alvo), bem como os sinais para operação em barramento de *64 bits*. Os sinais que estão em negrito fazem parte do conjunto mínimo de sinais que um dispositivo compatível com o padrão PCI deve manipular em modo alvo.

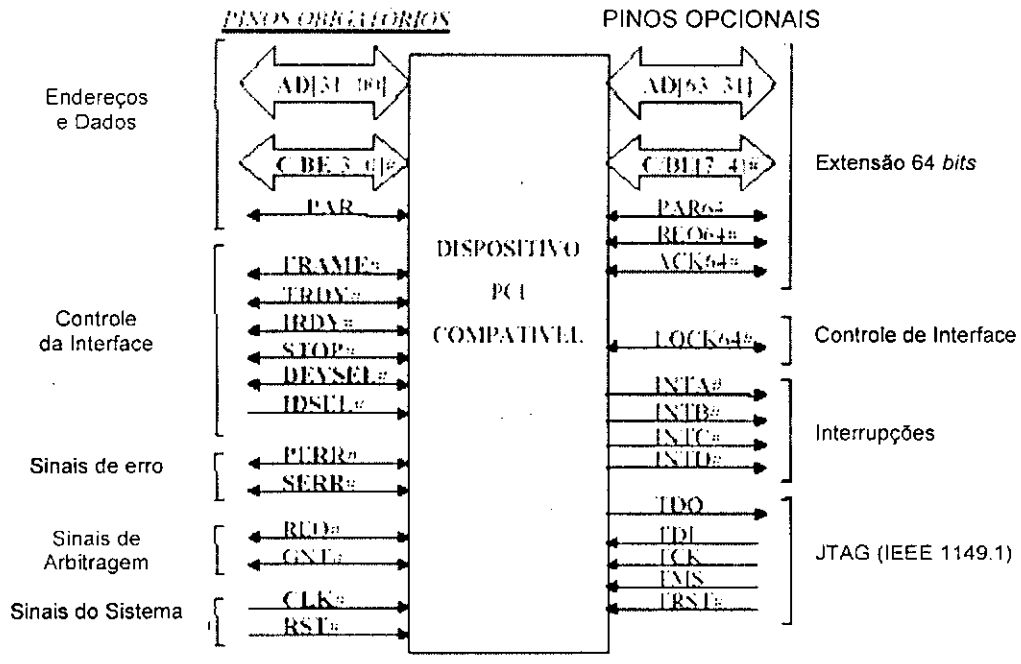


Figura D.4 – Sinais básicos do barramento PCI.

Uma observação importante quanto à multiplexação das linhas de dados e endereços é que quando o mestre solicita dados do alvo, necessita-se de um ciclo extra no protocolo para a inversão da direção do barramento. Este fato fica mais claro na discussão sobre os ciclos básicos de operação (escrita e leitura), abordados na seção Ciclos Básicos de Operação em 32 *Bits*.

A especificação PCI permite que o tamanho dos dados em suas vias seja identificado dinamicamente. Não há troca de dados (*swap*) nas vias de dados. Logo, todos os dispositivos PCI compatíveis devem dar suporte a uma largura de barramento de dados de 32 *bits*.

O barramento AD[31::0] contém 4 vias de dados com largura de um octeto. AD[7::0] compreende a primeira via, AD[15::8] a segunda, AD[23::16] a terceira e AD[31::24] a quarta via. Estas vias podem ser acessadas independentemente, conforme o valor especificado em C/BE#.

Sinal de paridade

- **PAR** e **PAR64** - A especificação do padrão PCI provê uma capacidade de detecção de erros limitada. As linhas C/BE# e AD são protegidas com paridade par sinalizadas pelos sinais PAR (versão 32 *bits*) e PAR64 (utilizado nas operações em 64 *bits*). As linhas PERR# e SERR# são utilizadas para sinalizar erros de paridade e erros com a plataforma hospedeira. As informações de erro são armazenadas em registradores e podem ser acessadas pelo BIOS, pelo sistema operacional e pelo programa de aplicação.

A paridade é utilizada em todos os ciclos do barramento PCI. A geração da paridade não é opcional e deve ser computada e sinalizada em uma das linhas PAR/PAR64. Contudo, a paridade não precisa ser verificada por todos os dispositivos conectados ao barramento. Os dispositivos que não verificam e reportam erros de paridade são aqueles que satisfazem às seguintes condições:

- Recursos que estão inseridos diretamente na placa mãe e não em placas de inserção (*add-on cards*).

- Recursos que não processam dados permanentemente, por exemplo, placas de vídeo.

Em outras palavras, recursos que não causarão perda nos dados no processamento da plataforma quando um erro de paridade ocorrer e não suportar verificação de paridade.

A primeira fase de uma transação de barramento chama-se de fase de endereçamento, na qual o mestre coloca um endereço nas linhas AD e um comando nas linhas C/BE. Um ciclo de relógio depois, o mestre coloca na linha PAR o *bit* de paridade correspondente, calculado sobre os *bits* de AD e C/BE#. Este atraso de um ciclo de relógio deve-se ao fato de que o alvo precisa calcular a paridade entre AD e C/BE recebidos e depois receber o sinal PAR para poder comparar com a paridade que calculou. Se o *bit* recebido na linha PAR for igual ao da paridade calculada pelo alvo, a linha PERR# (*Parity Error*) é mantida desativada, isto é, em nível lógico 1. A segunda fase de uma transação de barramento é a fase de dados. A partir desta fase a paridade é calculada entre os dados colocados em AD e os sinais do dígito de

habilitação colocados em C/BE. Para cada dado e dígito de habilitação, a paridade é calculada e um sinal PAR é gerado, seja em ciclos de leitura ou de escrita.

D.5 Transações do barramento

As transações são indicadas pelo Comando e fornecidas ao barramento PCI durante a fase de endereçamento. Os comandos de barramento indicam ao alvo o tipo de transação que o mestre está requisitando. No início de cada fase de endereçamento, as linhas C/BE[3::0]# carregam o código do comando. Na Tabela D.4 apresentam-se os tipos de comandos e são destacados os comandos mínimos para um dispositivo operar somente como alvo.

Tabela D.4 – Comandos do barramento PCI.

C/BE[3::0]#	Tipo de Comando	Ação
0000	Interrupt Acknowledge	Retorna o vetor de interrupção.
0001	Special Cycle	Mecanismo de *Broadcast*[30].
0010	**I/O Read**	**Leitura no espaço de I/O.**
0011	**I/O Write**	**Escrita no espaço de I/O.**
0100	Reservado	O alvo não deve responder.
0101	Reservado	O alvo não deve responder.
0110	**Memory Read**	**Leitura no espaço de memória.**
0111	**Memory Write**	**Escrita no espaço de memória.**
1000	Reservado	O alvo não deve responder.
1001	Reservado	O alvo não deve responder.
1010	**Configuration Read**	**Leitura no espaço de configuração.**
1011	**Configuration Write**	**Escrita no espaço de configuração.**
1100	Memory Read Multiple	Lê uma linha da *cachê*.
1101	Dual Address Cycle	Suporte para 64 bits de end.
1110	Memory Read Line	Lê varias linhas da *cache*.
1111	Memory Write and Invalidate	Transfere no mínimo uma linha da *cache*.

Todos os dispositivos PCI (com exceção das pontes do barramento hospedeiro) devem responder como alvo no comando configuração (escrita ou leitura). Todos os outros comandos são opcionais.

D.6 Ciclos básicos de operação em 32 *bits*

O barramento PCI suporta transferências de dados em modo rajada, tanto para transferência de dados para uma aplicação de dispositivo do usuário, ou para os registros de configuração. Uma alta taxa de transferência é alcançada devido ao

fato de que múltiplas fases de dados são realizadas para cada fase de endereçamento, ao invés de uma fase de dados para uma fase de endereçamento para o modo simples (não rajada).

Os ciclos descritos a seguir são:

- Ciclo de acesso à Configuração;
- Ciclo de acesso de Leitura em Modo Simples;
- Ciclo de acesso de Escrita em Modo Simples;
- Ciclo de acesso de Leitura em Modo Rajada;
- Ciclo de acesso de Escrita em Modo Rajada.

Nota: Neste Apêndice descreve-se somente o ciclo de acesso à configuração, maiores detalhes consultar as referências PCI[13], Cappelatti[31].

Ciclo de acesso à Configuração

A especificação do protocolo PCI suporta registradores de configuração em cada dispositivo[31] PCI. Este ciclo executa seus acessos da mesma forma que os ciclos de leitura e escrita, podendo ser, também, em modo simples ou rajada. Todo dispositivo PCI que é acessado por um ciclo de acesso à configuração possui 64 registradores de 4 octetos conforme mostrado na seção Registradores de Configuração. Estes registradores são acessados através das linhas AD[31::11] (*Initialization Device Select* - IDSEL), que atuam como um circuito selecionador tradicional. Durante os ciclos de configuração, o dispositivo alvo do acesso corrente pode não saber seu endereço, dado que ainda não foi configurado. Neste caso, o alvo é selecionado através do sinal IDSEL na inicialização do computador hospedeiro.

O ciclo de acesso à configuração tem como objetivo determinar as necessidades de cada dispositivo PCI conectado ao sistema.

Existem dois tipos de ciclos de acesso à configuração: Tipo 0 e Tipo 1. O Tipo 0 é efetuado em um dispositivo que se encontra no mesmo barramento em que o ciclo

[30] *Broadcast* -Transmissão por difusão; transmissão de dados para vários receptores.
[31] Barramento PCI Mestre, Alvo ou Ponte são definidos como dispositivos PCI.

de configuração está sendo executado. O Tipo 1 é executado somente entre PCI/PCI ponte.

O ciclo de acesso à configuração começa por iniciativa da unidade central de processamento do computador hospedeiro, tendo como componente ativo as pontes, que funcionam como conexão entre os barramentos PCI, operando assim como mestres. O ciclo inicia com o mestre (ponte) ativando o sinal FRAME#, colocando um comando válido em C/BE[31::0]# e estabilizando as linhas AD[31::0]#. Para um acesso à configuração, sinais em AD[31::11] são anexados às linhas IDSEL de cada dispositivo. Durante a fase de endereçamento, somente um dispositivo é selecionado por IDSEL.

Logo após a fase de endereçamento segue-se a fase de dados. O protocolo da fase de dados, tanto para o modo de acesso simples ou modo rajada, é o mesmo que nos acessos à memória ou E/S.

A ativação do sinal FRAME# no ciclo de configuração ocorre mais tarde do que nos ciclos de acesso à memória ou E/S, o que leva as linhas IDSEL do alvo acessado anteriormente a um estado de alta impedância antes de utilizar estas linhas para o alvo atual.

O acesso ao espaço de configuração é feito através de dois ciclos de comando: o de leitura e o de escrita da configuração. No Gráfico D.1 é mostrado o ciclo de leitura do espaço de configuração.

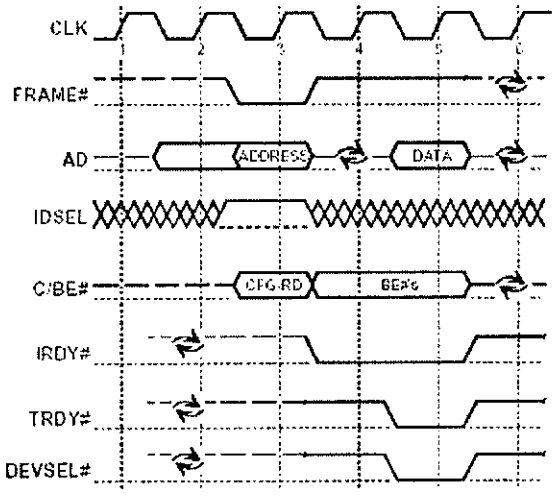


Gráfico D.1 – Ciclo de leitura da configuração[13].

Descrição dos eventos do ciclo:

Subida do sinal de relógio – [1]

- O mestre coloca o endereço do registrador a ser lido.
- Existe a inversão de sentido dos sinais IRDY#, TRDY# e DEVSEL#.

Subida do sinal de relógio – [2]

- O mestre ativa o sinal FRAME# dando início a transação.
- O comando (C/BE[3::0]#=1010) a ser executado é colocado pelo mestre.
- O mestre coloca o sinal IDSEL[32] em NL1 indicando o conector físico PCI onde está a placa a ser configurada[33].

Subida do sinal de relógio – [3]

- Os dispositivos do barramento reconhecem o sinal FRAME#.
- O ciclo do barramento é identificado pelos dispositivos como sendo de leitura da configuração.
- O mestre ajusta os sinais BE#s.
- O sinal IDSEL é retirado (não é colocado em NL0).
- O mestre ativa o sinal IRDY# indicando que está pronto para a leitura dos dado.

Subida do sinal de relógio – [4]

- Ocorre a inversão de sentido do barramento A/D (para evitar a contenção do barramento).
- O sinal DEVSEL# é ativado pelo dispositivo que o recebeu e este se torna o alvo da transação.
- O alvo afirma o sinal TRDY# indicando que está pronto para fornecer o dado.

Subida do sinal de relógio – [5]

- O mestre lê o valor do registrador de configuração, pois os sinais IRDY# e TRDY# estão ambos ativados (se um deles estiver negado será gerado um ciclo de espera).
- O mestre nega o sinal IRDY#(s/t/s).
- O alvo nega os sinais TRDY# e DEVSEL# (s/t/s).

---

[32] Um sinal independente por conector e válido apenas durante o ciclo de configuração [18].

Os sinais Frame#, IRDY#, TRDY# e DEVSEL# são do tipo *Tri-State* Sustentado e devem respeitar o protocolo PCI quanto a sua desativação, isto é: Tomando-se como exemplo o sinal FRAME#, verifica-se que na subida do sinal de clk_3 ele é negado, indicando que o mestre deseja ler apenas um dado de 32 *bits*, na subida do clk_4 ele é mantido em NL1 e na subida do clk_5 ele é flutuado (Gráfico D.1).

[33] Somente passará a "existir" (responder aos ciclos de leitura e escrita) no ambiente após ser configurada [18.]

APÊNDICE E. – Protocolo de Núcleo Aberto

O Protocolo de Núcleo Aberto (*Open Core Protocol - OCP™*) é um protocolo para interface de barramento. Neste trabalho usa-se como referência do Protocolo de Núcleo Aberto a especificação OCP-IP™ revisão 1.0.

E.1 Avaliação do OCP™

O OCP™ define uma interface de alto desempenho independente do barramento entre núcleos de IP, que reduz tempo de projeto, risco de projeto, e os custos de engenharia para projetar SoCs.

Um núcleo de IP pode ser um controlador de periférico simples, um microprocessador de alto desempenho, ou um subsistema de comunicação como um barramento *on-chip*.

- A meta é o reuso de projetos de IP. O OCP™ torna os núcleos de IP independentes da arquitetura e projeto dos sistemas em que eles são usados.
- Características necessárias para comunicação dos núcleos são implementadas na área de configuração da interface do OCP™.
- Simplifica a verificação criando um limite ao redor de cada núcleo de IP que pode ser observado, controlado, e validado.

E.2 Característica do OCP™

O OCP™ define uma interface ponto-a-ponto entre duas entidades. Uma entidade é definida como o mestre e o outro como o escravo. Só o mestre pode apresentar comandos. O escravo responde aos comandos apresentados, aceitando os dados do mestre ou apresentando dados para o mestre. Duas entidades se comunicam usando o processo par-a-par. Para que haja uma conexão é preciso duas instâncias do OCP, a primeira entidade é um mestre e a segunda entidade é um escravo.

Na Figura E.1 ilustra-se um sistema simples contendo um barramento encapsulado e três núcleos de entidades IP (instâncias OCP™): um é o iniciador do sistema (mestre), um outro é o alvo do sistema (escravo), e uma outra entidade com ambas as funções (mestre e escravo).

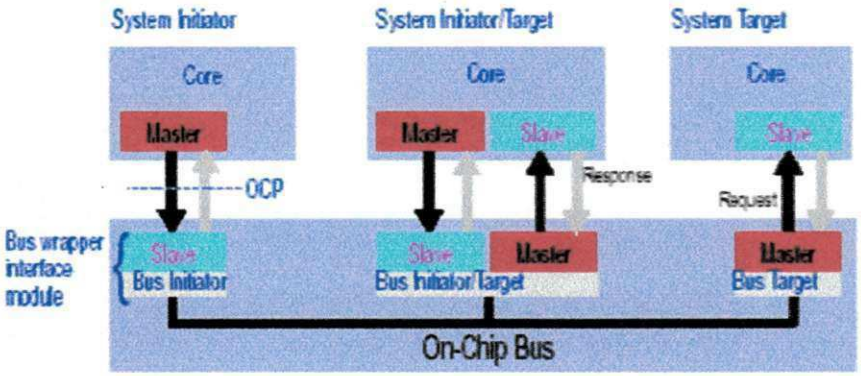


Figura E.1 – Exibição de Sistema barramento encapsulado e Instanciais OCP[09].

As características do núcleo de IP determinam se existem necessidades de núcleos mestre, escravo ou ambos. Os módulos de interface de envoltura devem agir como o lado complementar do OCP™ para cada entidade conectada.

Uma transferência através deste sistema acontece como segue:

Um iniciador de sistema (mestre como é denominado no OCP™) apresenta o comando, controle, e possivelmente dados para seu escravo conectado (módulo de interface no barramento de envoltória). O módulo de interface transfere o pedido através do barramento do sistema. O OCP™ não especifica a funcionalidade do barramento embutido. Ao invés, o projetista da interface converte o OCP™ para solicitar uma transferência ao barramento embutido. O módulo da interface do barramento de envoltória do receptor (mestre) converte a operação de barramento embutido em um comando legal do OCP™. O alvo do sistema (OCP™ escravo) recebe o comando e toma a ação solicitada.

Cada instância do OCP™ é configurada (escolhendo sinais ou larguras de *bit* de sinais particulares) baseada nos requisitos das entidades conectadas e sendo independentes uns dos outros. Por exemplo, iniciadores de sistema podem exigir mais *bits* de endereço em suas instâncias de OCP™ que fazem os alvos de sistema;

os *bits* extras de endereço poderiam ser usados pelo barramento embutido para selecionar que destino de barramento é tratado pelo iniciador de sistema.

O OCP™ é flexível, existem vários modelos úteis para os diversos núcleos de IP existentes se comunicarem uns com os outros: Um emprego é o da canalização[34] para melhorar a largura de banda e características de latência. Outro modo de acessar é usar ciclos múltiplos, no qual sinais são mantidos estáveis para vários ciclos de relógio para permitir que um IP mais lento tenha tempo para reagir. É possível suportar grande variedade de comportamentos pelo uso de *handshaking*[35] de sinais síncronos que permitem ambos, o mestre e escravo, controlar as operações.

E.3 Teoria de Operação

A comunicação entre unidades funcionais (ou núcleos de IP) incluídas em SoCs funcionam com endereços da interface do Protocolo de Núcleo Aberto. O OCP™ aceita independência de protocolos de barramento sem ter que sacrificar desempenho de acesso.

A metodologia estruturada do OCP™ inclui todos os sinais exigidos para descrever comunicações dos núcleos de IP inclusive fluxo de dados, controle, verificação e sinais de teste.

Neste ponto, faz-se uma avaliação dos conceitos do OCP™, introduzindo a terminologia usada para descrever a interface e oferecer uma visão do protocolo.

Interface Síncrona Ponto-a-Ponto

Para simplificar análise de contagem de tempo, projeto físico, e compreensão geral, o OCP™ é composto de sinais unidirecionais completamente síncronos com a

---

[34] Referente ao termo em inglês "***pipelining***", (a) arquitetura computacional encadeada; construída em blocos e executa instruções em passos onde cada bloco trata de uma parte da instrução e dessa forma acelerando a execução do programa; (a) iniciar o processamento de uma segunda instrução enquanto a atual ainda está processando de forma a aumentar a velocidade de execução de um programa [dicionário Michaelis]

[35] *Handshaking* - Refere-se a uma troca de informação entre mestre e escravo sinalizada, ou seja, controlando quando os sinais de permissão devem mudar para pedido e confirmação de transmissão.

borda de subida do relógio e não contém nenhum caminho de contagem de tempo de multiciclo. Todos os sinais diferentes do relógio e do reiniciar são estritamente ponto-a-ponto.

Os sinais da Interface Síncrona Ponto-a-Ponto do OCP™ são agrupados em um conjunto denominado *Dataflow*[36], complementar, e sinais de Teste. Um conjunto pequeno dos sinais do grupo Fluxo de Dados é chamado de OCP™ básico e é exigido em todas as configurações do OCP™. Um outro grupo, os sinais opcionais podem ser configurados para suportar requisitos de comunicação de núcleo adicional. O grupo de sinais opcionais Fluxo de Dados é dividido em extensões Simples e Complexas.

Independência de Barramento

Um núcleo utilizando o OCP™ pode ser interfaceado para qualquer barramento (*Bus Independence*). Técnicas de seleção dos dispositivos variam muito nos barramentos, alguns usam decodificadores de endereços, outros geram sinais de seleção de dispositivo independente (análogo a um circuito de seleção em uma placa). Esta complexidade deve ser escondida dos núcleos de IP, especialmente no caso da conexão direta em que não exista nenhuma lógica de decodificação/seleção. No OCP™ os escravos recebem informações de seleção dos dispositivos integradas no campo básico do comando.

Os esquemas de arbitramento variam extensamente. Desde a inexistência de arbitramento no caso da conexão direta, até o arbitramento para qualquer recurso compartilhado, a responsabilidade da lógica é somente no lado do barramento do OCP™. Estas "licenças" do OCP™ aos mestres passam por um campo de comando através do OCP™ que na interface de barramento são convertidos da lógica em uma seqüência de pedido de arbitramento.

Comandos

Existem dois comandos básicos, Leia (*Read*) e Escreva (*Write*) e duas extensões de comando:

---

[36] *Dataflow* ou fluxo de dados, é um conjunto de sinais usados para movimentar os dados através de um sistema.

O comando de Transmissão Pública tem a mesma semântica do protocolo de Escrita; a diferença é que o mestre indica que está tentando escrever para vários ou todos dispositivos de destino remoto.

A segunda extensão de comando, Leitura Exclusiva (ReadEx), tem no protocolo a semântica que é semelhante ao comando de Leitura (RD), mas garante bloqueio de recurso suficiente para suportar o atômico leia-modifique-escreva ou semântica de troca. Ao receber um comando de leitura exclusiva, o escravo tenta adquirir acesso exclusivo ao recurso tratado. Uma vez que o escravo retorna dados daquele endereço, o mestre pode assumir que obteve um acesso exclusivo e emitir o comando de escrita. O comando de escrita notifica o escravo para atualizar o endereço (que deve combinar o endereço de ReadEx), e então liberar acesso exclusivo para a posição de memória.

Endereços e Dados

Larguras características dos barramentos de Endereço e Dados fazem com que só aqueles *bits* de endereço que são significantes para o núcleo IP cruzem a interface OCP™-IP. O espaço de endereçamento de OCP™ é plano e composto de 8 *bits* (octeto).

Para acrescentar eficiência as transferências, muitos núcleos de IP têm larguras de campo de dados significativamente maior que um octeto. O OCP™ suporta diretamente até 128 *bit* de dados, permitindo a ele transferir 16 octetos simultaneamente. O OCP™ se refere à largura de campo de dados escolhidos como o tamanho de palavra[37] do OCP™. Por exemplo, um núcleo de DSP de 12 *bits* tipicamente empregaria uma palavra de 16 *bits* OCP™ (dois octetos) e provê zeros nos *bits* superiores (0000bbbb.bbbbbbbb).

As transferências de dados menores que uma palavra completa são suportadas ativando informações que especifiquem que octetos são transferidos e especificando a montagem de octetos em agregados seguindo as regras que tratam o agregado no OCP™.

Canalização

O OCP™ permite Canalização de transferências. Para suportar esta característica, o retorno de "leia dados" e a provisão de "escreva dados" podem estar atrasados depois da apresentação do pedido associado.

Resposta

O OCP™ separa pedidos de respostas. Um escravo pode aceitar um comando solicitado de um mestre em um ciclo e responder em um ciclo mais tarde.

Estouro

Para simplificar a comunicação para IPs que recebem ou produzem fluxos de dados contínuos (exemplo: FIFO) existe o modo Estouro[38]. Este modo exige, apropriadamente, que uma seqüência de endereços acompanhe cada comando sucessivo no estouro. Isto simplifica os requisitos de endereço para o processamento no escravo.

Encadeamentos e Conexões

Para suportar processamento concorrente de transferências fora de ordem o OCP™ ESTENDIDO suporta a noção de múltiplas Cadeias[39]. Transações dentro das Cadeias diferentes têm requisitos de não ordenado e, então, podem ser processadas fora de ordem. Porém, em uma única cadeia de fluxo de dados, toda transferência no OCP™ deve permanecer ordenada.

Interrupções, Erros, e outra Sinalização complementar

Tipos diferentes de sinalização de controle são exigidos para transferências de dados de coordenadas (por exemplo, fluxo de controle de alto nível) ou eventos de

---

[37] O termo "palavra" é usado no contexto de sistema de computador tradicional; isto é, uma palavra é a unidade natural de transferência do bloco OCP.

[38] Usa-se a palavra "**Estouro**" neste tópico (OCP™) para diferenciar do modo "Rajada" do tópico (PCI), em substituição a palavra inglesa "***Burst***", que neste caso representa uma curta seqüência isolada de sinais transmitidos; ± burst mode = modo intermitente = transmissão de dados em grupos separados de dados feita de modo intermitente. [Dicionário Michaelis].

[39] Usa-se a palavra "**Cadeia**" em substituição a palavra inglesa "***Thread***", para designar um arquivo encadeado, arquivo no qual uma entrada vai conter dados e um endereço para a próxima entrada

sinalização do sistema (como interrupção), são exigidas muitas vezes nas comunicações de dados dedicadas ponto-a-ponto. Muitos dispositivos também exigem a habilidade de notificar o sistema de erros.

O OCP™ se refere a toda comunicação como sinalização complementar, desde que não esteja diretamente relacionada ao protocolo do fluxo de dados do OCP™.

Os erros são divulgados através do OCP™ usando dois mecanismos. O código de resposta de erro, no campo de resposta descreve erros resultantes da transferência de resposta no OCP™. O segundo método para reportar erros através do OCP™ é um sinal de saída de faixa de erro.

E.4 Semântica de Protocolo

Neste tópico define-se a semântica do protocolo OCP™ atribuindo significados para as codificações dos sinais descritos anteriormente. Na Figura E.2 tem-se uma visão gráfica da hierarquia de elementos que compõem o OCP™.

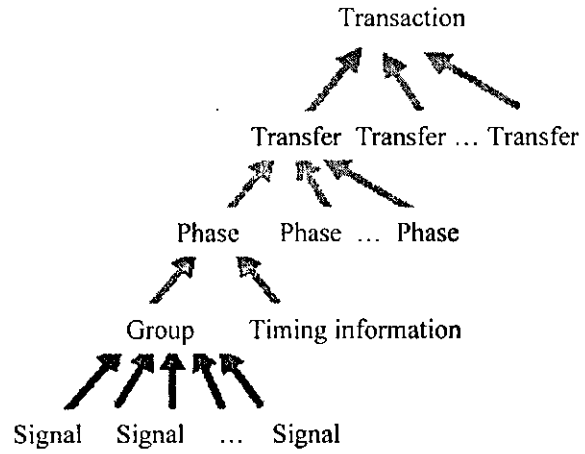


Figura E.2 – Hierarquia de Elementos[09].

Grupos de sinais

Campos de OCP™ são agrupados juntos porque devem ser ativos ao mesmo tempo. Os sinais de fluxo de dados são divididos em três grupos de sinais: pedido de

---

que tem o mesmo conteúdo de dados (permitindo a recuperação rápida de todos os registros idênticos). [Dicionário Michaelis].

sinais, sinais de resposta, e sinais *datahandshake*. Uma lista dos grupos de sinais é mostrada na Tabela E.1.

Tabela E.1 – Grupo de sinais OCP.

Groupo	Sinal	Condição
Pedido	MAddr	Sempre
	MAddrSpace	Sempre
	MBurst	Sempre
	MByteEn	Sempre
	MCmd	Sempre
	MConnID	Sempre
	MData*	datahandshake = 0
	MThreadID	Sempre
Resposta	SData	Sempre
	SResp	Sempre
	SThreadID	Sempre
Datahandshake	MData*	datahandshake = 1
	MDataValid	Sempre
	MDataThreadID	Sempre

Dependência Combinacional

É possível para algum sinal ou grupos de sinais que as saídas sejam derivadas diretamente de entradas, isto é combinacional. Descreve-se na Figura E.3 algumas possibilidades de dependência combinacional.

Os caminhos combinacionais não são permitidos com os sinais complementares e com os sinais de teste. A única dependência combinacional legal está dentro dos sinais de controle. Porém, sinais de fluxo de dados podem ser do tipo combinacional derivado de Reset_n.

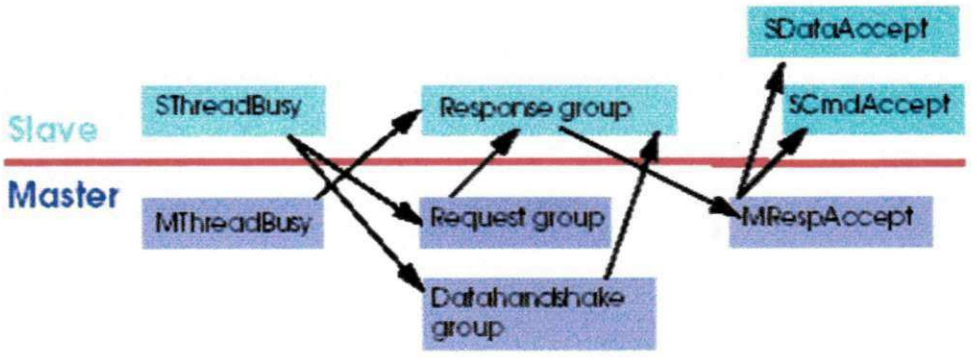


Figura E.3 – Dependência combinacional legal entre sinais e grupos de sinais[09].

Contagem de tempo e fases de sinais de Protocolo

Especifica quando um sinal pode ou deve ser válido.

Sinais de fluxo de dados

- O grupo de sinais é válido sempre que um comando diferente de Inativo é apresentado no campo de MCmd.
- O grupo de resposta é válido sempre que uma resposta diferente de Nula é apresentada no campo de SResp.
- O grupo *datahandshake* é válido sempre que 1 é apresentado no campo de MDataValid.

O aceite de sinal associado com um grupo de sinal é válido só quando este grupo for válido.

- O sinal de SCmdAccept é válido sempre que um comando diferente de Inativo é apresentado no campo de MCmd.
- O sinal de MRespAccept é válido sempre que uma resposta diferente de Nula é apresentada no campo de SResp.
- O sinal de SDataAccept é válido sempre que 1 é apresentado no campo de MDataValid.

Uma maneira de mapear os grupos de sinais individuais para fases de protocolo é retê-los fixos desde o início de uma fase de protocolo até o fim da fase. Fora de uma fase de protocolo, todos os sinais nos correspondentes grupos (com exceção do sinal que define o início da fase) são "irrelevantes".

- Uma fase de pedido começa sempre que o grupo de pedido for ativo. Quando o sinal de SCmdAccept for ativado durante um pedido de fase.
- Uma fase de resposta começa sempre que o grupo de resposta fica ativo. Isto quando o sinal de MRespAccept for ativado durante uma fase de resposta.
- Uma fase *datahandshake* começa sempre que o grupo de sinal *datahandshake* for ativo. Isto quando o sinal de SDataAccept for ativado durante uma fase *datahandshake*.

Fases em uma Transferência

Uma transferência no OCP™ consiste em várias fases mostradas na Tabela E.2. Toda transferência tem uma fase de pedido. Dependendo do tipo de transferência e da configuração do OCP™, o *datahandshake* ou a fase de resposta são opcionais.

Tabela E.2 – Fases OCP em uma transferência OCP.

Comando	Fase	Condição
Read, ReadEx	Pedido, resposta	Sempre
Escreve, *Broadcast*	Pedido	*datahandshake* = 0
Escreve, Broadcast	Pedido, *datahandshake*	*datahandshake* = 1

Ordem de fase dentro de uma Transferência

O OCP™ é causal: dentro de cada transferência uma fase de pedido deve preceder a fase *datahandshake* associada que na sua vez, deve preceder a fase de resposta associada. As restrições específicas são:

- Uma fase *datahandshake* não pode começar antes que comece a fase de pedido associado, mas pode começar no mesmo ciclo de *relógio* (Clk).
- Um *datahandshake* não pode terminar antes da fase de pedido associado terminar, mas pode terminar no mesmo ciclo de relógio.
- Uma fase de resposta não pode começar antes que a fase de pedido associado comece, mas pode começar no mesmo ciclo de relógio.
- Uma fase de resposta não pode terminar antes da fase de pedido associado terminar, mas pode terminar no mesmo ciclo de relógio.

Ordem de fase entre transferências

A ordenação de transferências é determinada pela ordem das fases de seu pedido.

- Caso duas fases do mesmo tipo, que pertencem a transferências diferentes, usem os mesmos sinais, a fase de uma transferência subseqüente não pode começar antes que a fase da transferência prévia termine. Se a primeira fase termina em x ciclos, a segunda fase pode começar em x+1 ciclos.

- As ordens das fases *datahandshake* do pedido devem seguir a ordem configurada pela fase solicitada.
- As ordens das fases *datahandshake* de resposta devem seguir a ordem de configuração das fases de pedido.
- Quando nenhuma ordem de fase é especificada, o efeito de duas transferências, que são endereçadas na mesma posição, deve ser o mesmo que se as duas transferências fossem executadas no mesmo pedido, mas sem qualquer sobreposição.

Sinais não agrupados

Os sinais não cobertos na descrição de grupos de sinais e fases são *MThreadBusy* e *SThreadBusy*. O ciclo de contagem de tempo da transição de cada *bit* que compõe cada um destes dois campos não é especificado em relação a outros sinais *dataflow*. Isto significa que não existe nenhum tempo específico para um OCP™ mestre ou escravo para este *drive* de sinais, nem um tempo específico para os sinais terem efetivo controle do fluxo. Segue que *MThreadBusy* e *SThreadBusy* só podem ser tratados como uma "sugestão".

Para prevenir blocagens encadeadas múltiplas das interfaces OCP™, o remetente de um encadeado tem necessidades de sinais ocupados para produzir o sinal no ciclo depois do último pedido de aceite ou fase de resposta. O receptor deve tomar o sinal em resposta por uma seleção da cadeia do ciclo corrente.

Banda lateral e Sinais de Teste

Um dispositivo de sistema afirma o sinal de reinicializar (Reset_n) na borda de subida do relógio, mestre e escravo devem transitar para um estado no qual não existe nenhuma pendência solicita/respostas OCP™.

Reiniciar

Um dispositivo de sistema ativa o Reset_n para o reinicializar do mestre e escravo da interface OCP™. Uma vez Reset_n sendo afirmado pela borda de subida de Clk, ambos o mestre e escravo devem ter transição para um estado no qual não exista nenhuma pendência OCP™ de pedidos ou respostas.

Reset_n deve ser ativado pelo menos por 16 ciclos de Clk para assegurar que o mestre e escravo alcancem um estado interno consistente. O mestre e escravo devem, cada um, poder alcançar seu estado de reinicializar não importando os valores apresentados na sinalização OCP™. Se o mestre ou escravo exigirem mais de 16 ciclos de Reset_n para ativação, o requisito deve ser documentado nas especificações de núcleo de IP.

Na mesma borda do *relógio* em que o Reset_n é ativado negado, em toda interface OCP™ os sinais devem ser válidos. Em particular, é legal para o mestre começar o seu primeiro pedido de fase no mesmo ciclo de *relógio* em que o Reset_n seja negado.

Interrupção, Erro e Sinalização do Núcleo

Não existe nenhuma contagem de tempo específica associada com os sinais de interrupção, erro e sinalização[40] do núcleo (SInterrupt, SError, MFlag e Sflag). A contagem de tempo destes sinais é específica do núcleo.

Estados e Controle

As regras a seguir asseguram que o controle das informações de estados podem ser permutadas através do OCP™ sem quaisquer caminhos combinacionais das entradas até as saídas.

- O controle deve ser retido fixo por um ciclo completo depois do ciclo em que tenha ocorrido uma transição, que significa não poder mais transitar para todos outros ciclos. Se ControlBusy fosse mostrado ativo no fim do ciclo atual, o controle não deveria ter transição no ciclo corrente. Depois do reiniciar ter sido negado, o controle teve ser retido fixo no primeiro ciclo de relógio.
- Se o controle de transições em um ciclo, ControlWr (se presente) estiver ativo deve ser dirigido para aquele ciclo. ControlWr seguindo as regras para controle, pode ser ativado em dois ciclos sucessivos.
- ControlBusy permite um núcleo forçar o sistema para reter o controle fixo. ControlBusy só pode começar a ser ativado logo depois do reinicializar, ou em

[40] É comum o uso da palavra inglesa "***Flag***" para indicar *bits* de sinalização, um indicador.

Solicitação de *Handshake*

Ilustra-se no Gráfico E.2 o mecanismo básico de controle de fluxo para a fase de pedido usando SCmdAccept. Existem três transferências de escrita, cada uma com um pedido diferente de aceite de latência.

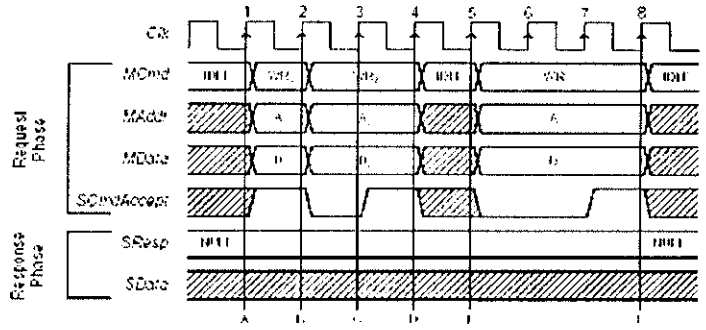


Gráfico E.2 – Solicitação de *Handshake*[09].

Seqüência

A - o mestre começa o pedido de escrita por WR de direção em MCmd e endereço válido e dados em MAddr e MData, respectivamente. O escravo afirma SCmdAccept no mesmo ciclo, para um pedido aceita de latência 0.

B - o mestre começa uma nova transferência no próximo ciclo. O escravo captura a escrita de endereço e dados. O SCmdAccept é negado, indicando que ainda não está pronto para um novo pedido.

C - Reconhecendo que SCmdAccept não é afirmado, o mestre segura todo pedido implantando os sinais (MCmd, MAddr, e MData). O escravo afirma SCmdAccept no próximo ciclo, para um pedido aceita de latência 1.

D - O escravo captura a escrita de endereço e dados.

E - após 1 ciclo inativo, o mestre começa um novo pedido de escrita. O escravo SCmdAccept é negado.

F - Desde então SCmdAccept é afirmado, a fase de pedido termina. SCmdAccept fica baixo por 2 ciclos, então o pedido aceita de latência para esta transferência é 2. O escravo captura a escrita de endereço e dados.

Solicitação de *Handshake* e Resposta em Separado

Ilustra-se no Gráfico E.3 um único ciclo transferência de leitura em que um escravo introduz espera no pedido e fases de resposta. O pedido aceita de latência 2, corresponde para o número de ciclos de relógio que SCmdAccept era negado. A solicitação para resposta latência 3, corresponde ao número de ciclos de *relógio* do fim da fase de pedido (D) para o fim da fase de resposta (F).

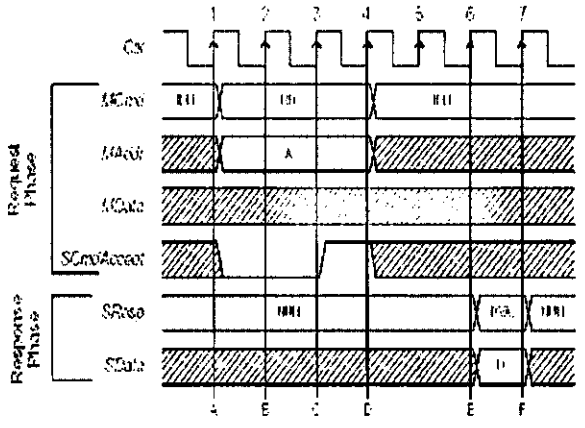


Gráfico E.3 – Solicitação de *Handshake* e Resposta em Separado[09].

Seqüência

A - O mestre começa uma fase de pedido emitindo o comando leitura (RD) no campo de MCmd. Ao mesmo tempo, apresenta um endereço válido em MAddr. O escravo não está pronto para aceitar o comando ainda, então ele negará SCmdAccept.

B - O mestre vê que SCmdAccept não é afirmado, então ele mantém todos os pedidos de sinais de fase fixo. O escravo pode estar usando estas informações para uma longa operação de decodificação, e ele espera o mestre para segurar-se que tudo esteja "fixo" até que ele afirme SCmdAccept.

C - O escravo afirma SCmdAccept. O mestre continua a segurar o pedido implantando sinais.

D - Desde que SCmdAccept é afirmado, a fase de pedido termina. O escravo captura o endereço, e embora a fase de pedido seja completa, não está pronto para prover a resposta, então ele continua direcionando NULO no campo de SResp. Por exemplo, o escravo pode estar esperando por dados para vir de volta de um dispositivo de memória com o circuito desativado.

E - O escravo está pronto para apresentar a resposta, então pega os conteúdos DVA no campo SResp e ler os dados em trânsito no SData.
F - O mestre vê a resposta de DVA e captura a leitura dos dados.

Pedido e Resposta Canalizada

Ilustra-se no Gráfico E.4 três transferências de leitura usando o pedido e resposta em transporte canalizado. Em cada caso, o pedido é imediatamente aceito, enquanto a resposta é retornada no mesmo ou num ciclo mais tarde.

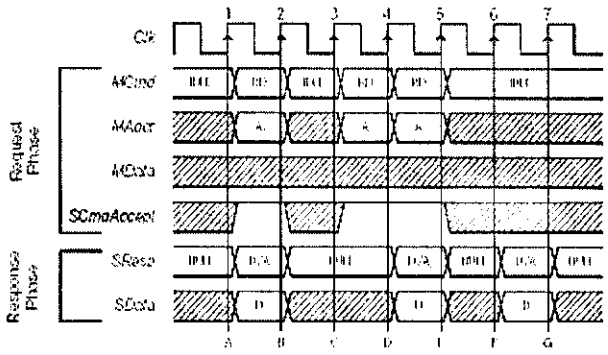


Gráfico E.4 – Pedido e Resposta Canalizada[09].

Seqüência
A - O mestre começa o primeiro pedido de leitura, dirigindo RD em MCmd e um endereço válido em MAddr. O escravo afirma SCmdAccept, para um pedido aceito de latência 0. Quando o escravo vier ler o comando, responde com DVA em SResp e dados válidos em SData. Este exige do escravo um caminho combinatório de MCmd e, possivelmente, outra fase de pedidos de campos, para SResp, e possivelmente outra fase de resposta de campos.
B - Então SCmdAccept é afirmado, a fase de pedido termina. O mestre vê que SResp é DVA e captura a leitura de dados de SData. Porque o pedido é aceito e a resposta é apresentada no mesmo ciclo, o pedido para resposta de latência é 0.
C - O mestre lança um pedido da primeira leitura e o escravo afirma SCmdAccept.
D - O mestre vê que SCmdAccept é afirmado, então ele pode lançar uma terceira leitura embora a resposta para a leitura anterior não foi recebida. O escravo captura o endereço da segunda leitura e começa o DVA de direção em SResp e a leitura de dados em SData.

E - Então SCmdAccept é afirmado, o terceiro pedido termina. O mestre vê que o escravo produziu uma resposta válida para a segunda leitura e captura os dados de SData. O pedido para resposta desta transferência têm latência de 1.

F - o escravo tem os dados para as três leituras, então este fornece o DVA em SResp e os dados em SData.

G - o mestre captura os dados para as três leituras de SData. O pedido para resposta de latência para esta transferência é 2.

Leitura não Canalizada

Ilustra-se no Gráfico E.5 três transferências de leitura para um escravo que não pede respostas de fonte de informação depois dos pedidos. Isto é o comportamento típico de protocolos de barramento de computador de legado com um sinal de espera (*Wait*) ou sinal de reconhecimento (ACK). Em cada transferência, SCmdAccept é ativado no mesmo ciclo que está SResp e DVA. Então, o pedido para resposta de latência é sempre 0, mas o pedido aceita latência varia de 0 até 2.

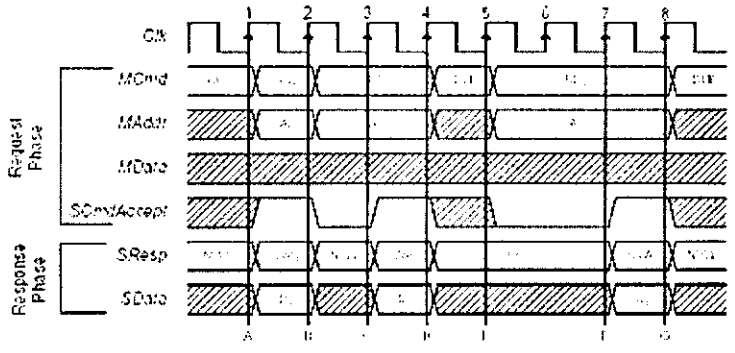


Gráfico E.5 – Leitura não Canalizada[09].

Seqüência

A - O mestre começa o primeiro pedido de leitura dirigindo RD em MCmd e um endereço válido em MAddr. O escravo afirma SCmdAccept, para um pedido de aceita de latência 0. Quando o escravo vier ler o comando, responde com DVA em SResp e dados válidos em SData. (Este exige uma combinação de caminho do escravo de MCmd e, possivelmente, outra fase de pedido de campos, para SResp e possivelmente, outra fase de resposta de campos.)

B - O mestre lança outro pedido de leitura. Também verifica que SResp é DVA e captura a leitura de dados de SData. O escravo não está pronto para responder para o novo pedido, então ele nega o SCmdAccept.

C - O mestre vê que SCmdAccept é baixo e estende a fase de pedido. O escravo está agora pronto para responder no próximo ciclo, então este simultaneamente afirma SCmdAccept e drives DVA em SResp e os dados selecionados em SData. Para o aceite do pedido têm-se a latência de 1.

D - Desde que SCmdAccept é afirmado, a fase termina. O mestre vê este SResp é agora DVA e captura os dados.

E - O mestre lança um terceiro pedido de leitura. O escravo negará SCmdAccept.

F - O escravo afirma SCmdAccept depois de 2 ciclos, então o aceite do pedido têm latência de 2. Ele também direciona DVA em SResp e ler os dados em SData.

G - O mestre vê que SCmdAccept é afirmado concluindo a fase. Ele também ver que SResp é agora DVA e captura os dados.

Leitura por Estouro

Ilustra-se no Gráfico E.6 uma transação de Leitura em Estouro que é composta de quatro transferências de leituras por estouros canalizados. Um campo adicional, MBurst, é adicionado ao pedido da fase, indicando o tipo do estouro e o número de transferências que o mestre espera. Neste diagrama, MData e SData são assumidos para ser 32 bits.

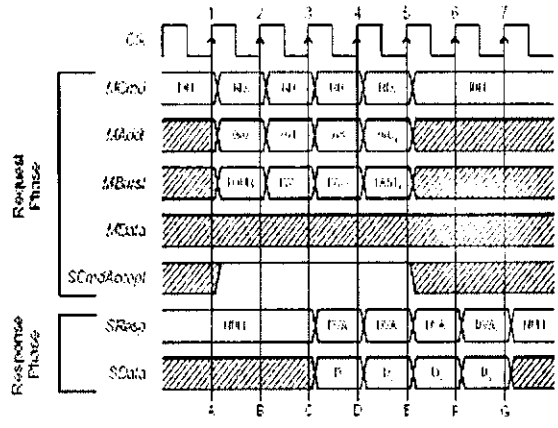


Gráfico E.6 – Leitura por Estouro[09].

Seqüência

A - O mestre começa a leitura por estouro inserindo o comando (RD) em MCmd, o primeiro endereço do estouro está no MAddr, e o código de estouro QUATRO em MBurst. O código de estouro indica que isto é um incremento de estouro e que quatro ou mais transferências são esperadas. O escravo está pronto para qualquer coisa, então ele afirma SCmdAccept.

B - O mestre emite a próxima leitura no estouro. MAddr é configurado para o próximo endereço de "palavra-alinhada". Para palavras de 32 bits, o endereço é incrementado por 4. O mestre também muda MBurst para DOIS, significando que duas ou mais transferências ainda permanecem na transação.

C - O mestre emite a próxima leitura no estouro, incrementando MAddr e deixando MBurst configurar para DOIS, porque restam duas ou mais transferências. O escravo está agora pronto para responder para a primeira leitura no estouro, então ele direciona DVA em SResp e dados válidos em SData. O pedido para aceita de latência para esta transferência é 2.

D - O mestre emite a leitura final no estouro, incrementando MAddr e configurando MBurst para durar. O mestre também captura os dados para as primeiras leituras do escravo. O escravo responde para a segunda transferência. O pedido para aceita de latência para esta transferência é 2, embora seja possível para o escravo introduzir mais latência para cada resposta em uma transação de estouro. (No OCP™, estouro não impõe quaisquer restrições adicionais de contagem de tempo de protocolo).

E - O mestre captura os dados para a segunda leitura do escravo. O escravo responde para a terceira transferência.

F - O mestre captura os dados para a terceira leitura do escravo. O escravo responde para a quarta e última transferência.

G - O mestre captura os dados de leitura pela última vez do escravo.

Aceite de Resposta

Ilustra-se no Gráfico E.7 exemplos da extensão de aceite de resposta (*Response Accept*) usada com duas transferências de leitura. Um campo adicional, MRespAccept, é adicionado a fase de resposta. Este sinal pode ser usado pelo mestre para controle de fluxo de fase da resposta.

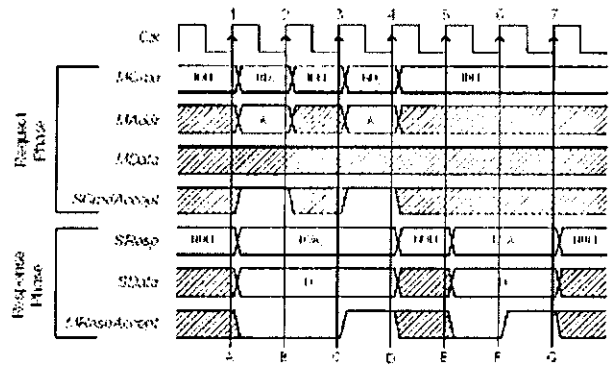


Gráfico E.7 – Aceite de Resposta[09].

Seqüência

A - O mestre começa o pedido de uma leitura por RD de direção em MCmd e um endereço válido em MAddr. O escravo afirma SCmdAccept imediatamente, e este direciona os DVA em SResp assim que vê o pedido de leitura. O mestre não está pronto para receber a resposta que acabou de solicitar, então este sinal permanece negado MRespAccept.

B - Então SCmdAccept é afirmado, a fase de pedido termina. O mestre, porém, continua negando MRespAccept. O escravo segura SResp e SData fixa.

C - O mestre começa um segundo pedido de leitura. Está finalmente pronto para a resposta de seu primeiro pedido, então ele afirma MRespAccept. Este corresponde a um aceite de resposta de latência de 2.

D - Desde que SCmdAccept é afirmado, a fase de pedido termina. O mestre captura os dados para a primeira leitura do escravo. Desde que MRespAccept é afirmado, a fase de resposta termina. O escravo não está pronto para responder para a segunda leitura, então ele direciona NULOS em SResp.

E - O escravo responde para a segunda leitura por DVA de direção em SResp e a leitura de dados em SData. O mestre não está pronto para a resposta, então ele negará MRespAccept.

F - O mestre afirma MRespAccept, para o aceite do pedido a latência é de 1.

G - O mestre captura os dados para a segunda leitura do escravo. Desde que MRespAccept é afirmado, a fase de resposta termina.

Extensão *Datahandshake*

Ilustra-se no Gráfico E.8 três transferências de escrita usando extensão do *datahandshake*. Esta extensão adiciona fase *datahandshake* que é completamente independente do pedido e fases de resposta. Dois novos sinais: MDataValid e SDataAccept, são adicionados e MData é reposicionada na fase de pedido até o fase do *datahandshake*.

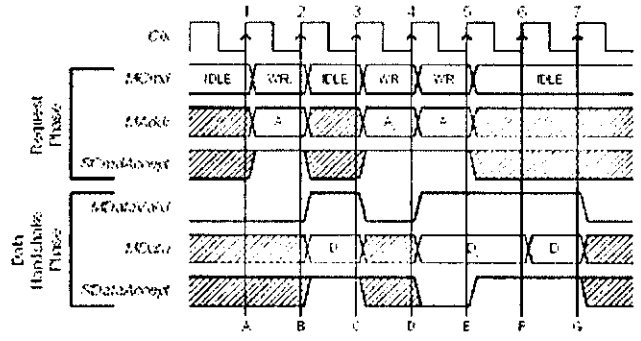


Gráfico E.8 – Extensão *Datahandshake*[09].

Seqüência

A - O mestre começa o pedido de escrita por WR de direção em MCmd e um endereço válido em MAddr. Ainda não tem a escrita de dados, então este negará MDataValid. O escravo afirma SCmdAccept. Não precisa afirmar ou negar SDataAccept ainda, porque MDataValid está quieto negado.

B - O escravo captura a escrita do endereço do mestre. O mestre está agora pronto para transferir a escrita de dados, então ele afirma MDataValid e direciona os dados em MData, começando a fase do *datahandshake*. O escravo está pronto para aceitar os dados imediatamente, então ele afirma SDataAccept. Isto corresponde, para os dados, um aceite de latência 0.

C - O mestre permanece negando MDataValid enquanto não tiver mais dados para transferir (Como MCmd e SResp, MDataValid devem estar sempre em um estado especifico válido). O escravo captura a escrita de dados de MData, completando a transferência. O mestre começa uma segunda escrita para pedido por WR de direção em MCmd e um endereço válido em MAddr.

D - Desde que SCmdAccept é afirmado, o mestre começa imediatamente um terceiro pedido de escrita. Também afirma MDataValid e apresenta a escrita de

dados da segunda escrita em MData. O escravo não está pronto para os dados ainda, então este permanece negando SDataAccept.

E - O mestre "vê" que SDataAccept permanece, então ele segura os valores MDataValid e MData. O escravo afirma SDataAccept, tempo de aceite para uns dados é de latência 1.

F - Então SDataAccept é afirmado e a fase do *datahandshake* termina. O mestre está pronto para entregar a escrita de dados para o terceiro pedido, então este mantém MDataValid afirmado e apresenta os dados em MData. O escravo captura os dados para a segunda escrita de MData e mantém SDataAccept afirmado, o aceite para os dados tem latência de 0 para a terceira escrita.

G - Então SDataAccept é afirmado, a fase do *datahandshake* termina. O escravo captura os dados para a terceira escrita de MData.

Leitura Encadeada

Ilustra-se no Gráfico E.9 uma conclusão de transferência de leitura fora de ordem usando a extensão de Leitura em Cadeia do OCP™. Este Gráfico é desenvolvido do Gráfico E.4. Os sinais *thread* IDs (MThreadID e SthreadID) foram adicionados e o pedido de duas das respostas foi mudado.

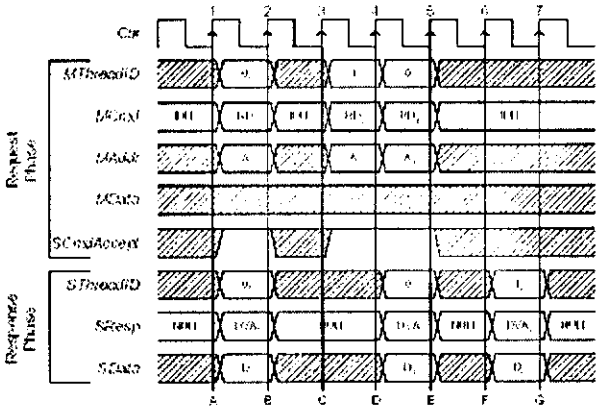


Gráfico E.9 – Leitura Encadeada[09].

Seqüência:

A - O mestre começa a primeira leitura de pedido, dirigindo RD em MCmd e um endereço válido em MAddr. Ele também direciona um 0 em MThreadID, indicando que esta leitura de pedido é para *thread* 0. O escravo afirma SCmdAccept, para o aceite do pedido a latência é de 0. Quando o escravo vier ler comando, responde com DVA em SResp e dados válidos em SData. O escravo também direciona uns 0 em SThreadID, indicando que esta resposta é para *thread* 0.

B - Então SCmdAccept é afirmado, a fase termina. O mestre vê se este SResp é DVA e captura leitura de dados de SData. Porque o pedido é aceito e a resposta é apresentada no mesmo ciclo, o pedido para resposta de latência é 0.

C - O mestre lança um novo pedido de leitura, mas este tempo é para *thread* 1. O escravo afirma SCmdAccept, porém, não está pronto para responder.

D - Desde que SCmdAccept é afirmado, o mestre pode lançar outro pedido de leitura. Este pedido é para *thread* 0, então MThreadID é trocado de volta para 0. O escravo captura o endereço da segunda leitura para *thread* 1, mas para isto começa direcionando DVA em SResp, os dados em SData e um 0 em SThreadID. Este significa que está respondendo para a terceira leitura, antes da segunda leitura.

E - Então SCmdAccept é afirmado, o terceiro pedido termina. O mestre vê que o escravo produziu uma resposta válida para a terceira leitura e captura os dados de SData. O pedido para resposta de latência para esta transferência é 0.

F - O escravo tem os dados para a segunda leitura, então direciona DVA em SResp, os dados em SData e um 1 em SThreadID.

G - o mestre captura os dados para a segunda leitura de SData. O pedido para resposta de latência para esta transferência é 3.

Nota: Para maiores informações consulte a referência OCP™ [09].

APÊNDICE F. – SystemC™

A linguagem *SystemC™* teve como primeira proposta uma versão apresentada pela *Synopsys: v0.9* em março de 1999, logo em seguida, em setembro de 1999, foi criado um grupo para organizar as diversas sugestões, surgindo o *Open SystemC™ Iniciative,* já no ano seguinte, em Julho 2000, surge o primeiro *Testbuilder-SC da Cadence*[20].

SystemC™ é uma linguagem de alto nível baseada na linguagem C++, tendo como um dos objetivos padronizar o uso de uma linguagem única nas etapas mais críticas do processo de projeto de um núcleo IP, passando pelas etapas de especificação, descrição comportamental, descrição RTL e descrição estrutural, não sendo necessário ao projetista o conhecimento de outras linguagens para conclusão de etapas intermediárias ao processo e nem construção de um Ambiente de Teste por Simulação e de uma Aplicação de Dispositivo de Usuário.

Mostra-se no diagrama da Figura F.1 o fluxo de um projeto, a parte destacada refere-se a abrangida pela linguagem *SystemC™*.

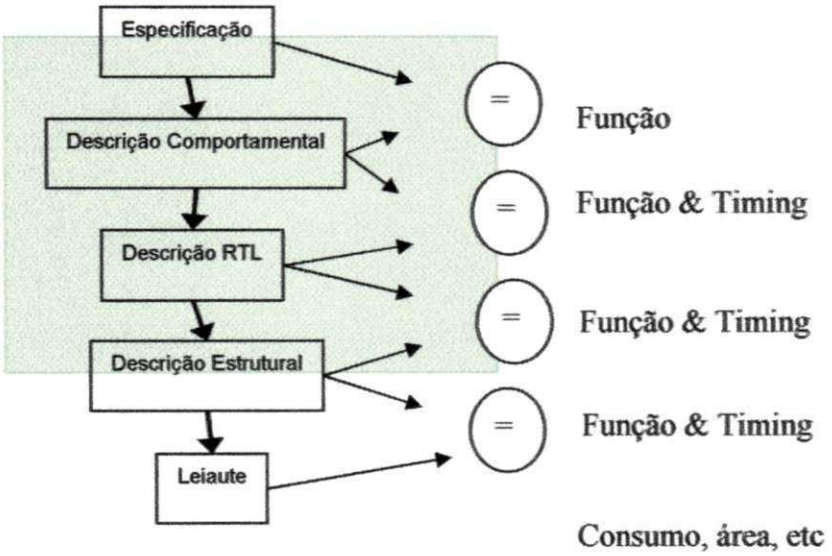


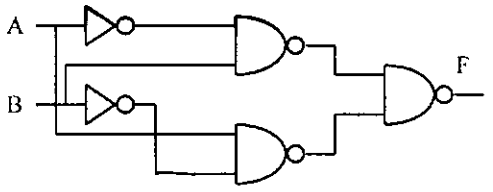
Figura F.1 – Diagrama de fluxo de projeto[20].

Representação gráfica *versus* Representação textual

Diferenças básicas entre a representação gráfica e a representação textual:

- Representação gráfica é melhor para o domínio estrutural, por ser mais intuitiva.
- Representação textual é melhor para o domínio comportamental, atingindo um nível de abstração maior. Apresenta maior dificuldade para aprender a linguagem, maior hierarquia, é rápida de fazer (mas necessita de comentários), mais fácil de modificar e mais fácil de ser processada automaticamente.

Na Figura F.2 ilustra-se uma representação gráfica e uma representação textual.




```
if (function==1)
        y = a-b;
else
        y = a+b;
```

Figura F.2 – Representação gráfica X textual[20].

Os projetistas de desenvolvimento usam atualmente alguns tipos de *HDLs* como por exemplo: *Verilog, VHDL, Abel, Palasm, Cupl, OCCAM, Handle-C, ELLA*, entre outras.

Tem-se como principais propriedades do *SystemC*™ as características expressas abaixo, podendo-se verificar que esta linguagem atende às necessidades de uma boa linguagem HDL.

As principais propriedades do *SystemC*™ são:

- Expressar ações concorrentes.
- Expressar tempo (atraso com restrições, relógio).
- Permitir as descrições comportamentais e estruturais, mas não físicas.
- Permitir mesclar diferentes vistas de diferentes subsistemas.
- Permitir simulação, síntese e verificação.
- Ser fácil e segura de usar.

Nota: Para maiores informações sobre *SystemC*™ consulte as referências [19 - 25].

APÊNDICE G. – Verificação Funcional

A Verificação Funcional por Simulação, geralmente se refere ao código utilizado para criar uma seqüência predeterminada de entradas para um circuito e, opcionalmente, observar suas saídas. É comumente implementado, através das linguagens VHDL ou *Verilog*, podendo incluir arquivos externos ou rotinas em C, sendo neste trabalho, especificamente, desenvolvido em *SystemC™*.

G.1 Verificação do Núcleo

O objetivo da verificação do núcleo é assegurar que o mesmo esteja correto tanto na funcionalidade quanto na temporização descritas na especificação funcional. A verificação é um dos maiores desafios no desenvolvimento de um projeto, principalmente quando se projeta um núcleo para ser reutilizado.

Embora esta etapa não faça parte do projeto em si, ela se torna essencial para a verificação do mesmo, sendo assim imprescindível a elaboração de um teste por simulação, completo durante a execução do trabalho[42].

A verificação deve garantir que o núcleo não apresente nenhum defeito. A ausência de defeitos deve ser garantida para todo tipo de configuração do núcleo, com todos os valores possíveis aceitos por seus parâmetros. Além disso, a equipe de integração deve ser capaz de reutilizar os ambientes de testes por simulações no nível do núcleo, pois o núcleo deve ser verificado tanto como um projeto isolado, quanto no contexto da aplicação final.

A motivação de se fazer uma verificação funcional por simulação está expressa diretamente no custo e tempo de corrigir um defeito, pois este cresce quando descoberto mais tarde no ciclo de vida do produto (Figura G.1).

---

[42] Esta tarefa ficou a cargo de outra equipe de desenvolvimento, dando maior funcionalidade e confiabilidade ao projeto desenvolvido nesta dissertação[20].

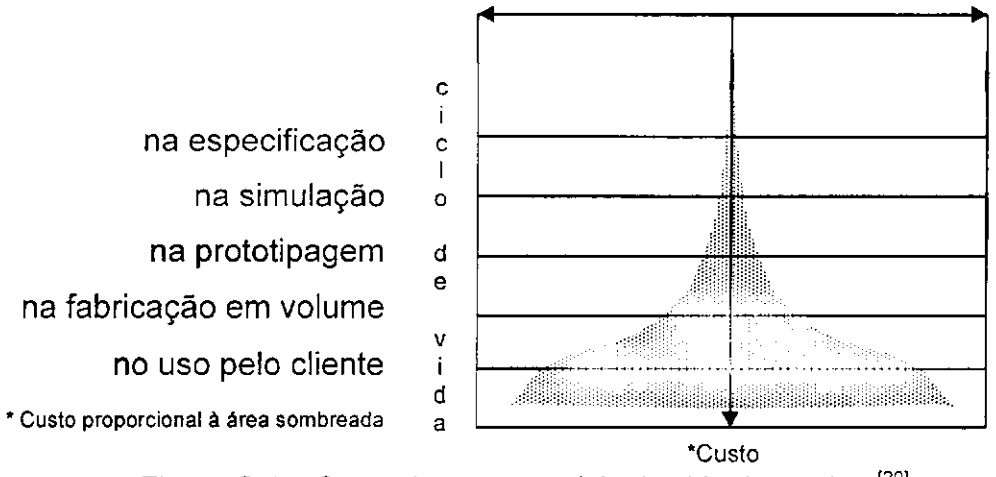


Figura G.1 – Curva de custo no ciclo de vida do produto[20].

O Ambiente de Teste por Simulação torna-se um problema, pois mais da metade do esforço de projeto está na verificação. Um Ambiente de Teste por Simulação muitas vezes contém mais linhas que a própria descrição do projeto. Todo o esforço sobre o Ambiente de Teste por Simulação torna-se um mal necessário, sempre leva tempo demais e custa caro demais, mas é indispensável porque afeta diretamente três requisitos básicos: cronograma, custo e qualidade.

G.2 Plano de Verificação

Devido à complexidade e escopo da verificação funcional, é essencial que um plano global de verificação funcional seja criado. Ter uma definição clara dos critérios que o núcleo deve obedecer ajuda a concentrar o esforço de verificação e ter uma noção mais exata de quando o núcleo está pronto para ser utilizado.

Os benefícios específicos do desenvolvimento do plano de verificação no início do projeto incluem:

(i) análise das tarefas que consomem mais tempo antes de executá-las;

(ii) concentração dos esforços nas áreas em que a verificação seja mais necessária;

(iii) minimização dos esforços redundantes;

(iv) o projetista pode compartilhar experiências e conhecimentos acumulados com o resto da equipe;

(v) o plano fornece um mecanismo formal para correlacionar as necessidades do projeto à testes específicos, garantindo a integridade da cobertura dos testes;

(vi) as informações contidas no plano permitem que uma equipe de suporte à verificação desenvolva o ambiente de verificação em paralelo às tarefas de captura do projeto, executadas pela equipe principal de projetistas. Isto pode reduzir significantemente o tempo de projeto.

A verificação de um núcleo consiste de três fases principais:

(i) verificação de módulos individuais;

(ii) verificação do núcleo;

(iii) e prototipagem.

O objetivo da primeira fase é atingir um nível bastante alto de cobertura dos testes no nível de módulos e, depois, concentrar a verificação em nível de núcleo, testando suas interfaces e sua funcionalidade. Esta abordagem de verificação começando do nível mais baixo até o mais alto (*bottom-up*) é baseada no princípio de localidade. É mais fácil detectar e consertar erros em módulos pequenos do que em módulos grandes.

A abordagem de verificação *bottom-up*, como um modelo de desenvolvimento em cascata, não é verdadeiramente eficiente. Na prática, uma abordagem em espiral envolvendo interações é a que realmente funciona. Embora esta abordagem garanta uma segurança bastante alta, não garante cem por cento de correção no funcionamento. A construção de um protótipo rápido do núcleo é o que permite que a equipe de projeto possa testar o núcleo em aplicações reais, aumentando sua confiabilidade e robustez. Esta necessidade de um protótipo do núcleo é atendida pelo uso dos dispositivos programáveis de alta densidade, como os FPGAs.

Em cada fase do projeto, a equipe deve decidir que tipos de testes serão utilizados e quais as ferramentas de verificação que serão necessárias para isso. Os tipos básicos de testes de verificação incluem:

Teste de adequação

O teste de adequação é usado para verificar se o projeto está de acordo com a especificação. Para um padrão da indústria, como uma interface PCI ou uma interface IEEE 1394[43], utiliza-se também este teste para verificar a adequação com a especificação publicada. Em todos os casos, a adequação do projeto com a especificação funcional é verificada da maneira mais completa possível.

Teste de casos extremos

Usa-se o teste de casos extremos para tentar encontrar situações complexas e casos extremos, em que o projeto provavelmente apresente falhas.

Teste aleatório

O uso do testes randômicos é útil para todo projeto, como um processador ou interfaces de barramento complexas, como complemento aos testes de adequação e de casos extremos. Estes são limitados à situação que os projetistas previram. Testes randômicos podem criar situações que os projetistas não previram e descobrir a maioria dos erros mais difíceis de serem detectados no projeto.

Teste de código real

Testar uma aplicação real, com código real é uma etapa importante na verificação de um projeto, porque sempre existirá a possibilidade de que a equipe de projeto entenda mal uma especificação, e acabe projetando e testando seu código com uma especificação errada.

Teste de regressão

À medida que os testes são desenvolvidos, eles devem ser adicionados ao conjunto de testes de regressão. Um dos problemas mais típicos encontrados durante a verificação é que, quando se conserta um erro, outro pode ser descuidadosamente introduzido. O conjunto de testes de regressão ajuda a verificar

[43] Um padrão de barramento externo muito rápido que suporta taxas de transferências de dados de até 400 Mbps (em 1394a) e 800 Mbps (em 1394b). Produtos suportando uma única porta 1394 podem ser usados para conectar 63 dispositivos externos. Além de sua alta velocidade, também suporta dados assíncronos -- entregando dados em uma taxa garantida. Sendo ideal para os dispositivos que precisam transferir grandes níveis de dados em tempo real, como vídeo dispositivos.

que em determinado ponto de referência, a funcionalidade continua sendo mantida à medida que novas características são adicionadas, e que todos os erros, até aquele ponto, foram corrigidos.

G.3 Abordagem de verificação

Existe três abordagens clássicas para se fazer a verificação funcional: a *Black Box*, a *Grey Box* e a *White Box*[20]. Neste trabalho sempre que possível a abordagem de verificação usada no Ambiente de Teste por Simulação será Caixa Preta (*Black Box*), na qual tem-se as Entradas, Saídas e Função, sendo a função bem documentada. Para verificar, é necessário entender a função e prever as saídas sabendo as entradas.

Exemplo de uma abordagem Caixa Preta é exposto da Figura G.2, em que são conhecidas as entradas, a função de transferência (*DUV – Design Under Verification*) e as saídas.

Figura G.2 – Caixa Preta[20].

G.4 Projeto de um Ambiente de Teste por Simulação

O projeto do Ambiente de Teste por Simulação difere dependendo da função do núcleo. O Ambiente de Teste por Simulação de um núcleo de uma interface de barramento deve usar modelos funcionais de barramentos e monitores de barramentos para aplicar estímulos e analisar os resultados.

A seguir, tem-se as definições do modelo do Ambiente de Teste por Simulação a ser desenvolvido.

- Montagem de teste para simulação.
- Código escrito em *SystemC*™.
- Criar estímulos e verificar a resposta.
- Não tem entrada nem saída.

- Um modelo do universo em volta do projeto.
- Imprime mensagens quando o DUV apresenta comportamento inesperado.
- Em verificando-se que tudo estar certo imprime uma única mensagem no final.

O projeto do Ambiente de Teste por Simulação de referência para o PCI 33 MHz de 32 bits contém os seguintes blocos funcionais: *Source*, *Reference Model*, *Driver*, Monitor e *Checker,* conforme ilustrado no diagrama da Figura G.3, o módulo a ser testado é o DUV (*Design Under Verification*).

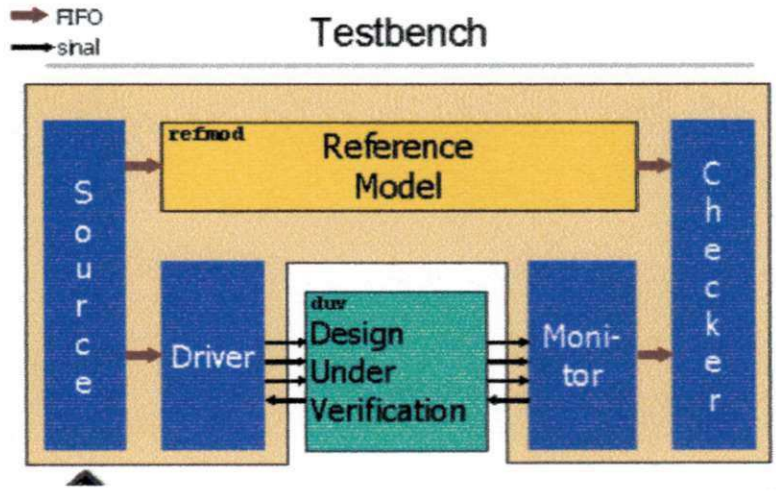


Figura G.3 - Diagrama do Ambiente de Teste por Simulação[20].

Source

Envia transações de entrada para o *driver*. É bom ser reutilizável, ou seja, depender pouco do DUV.

Reference model

Tipicamente independente do tempo (*timeless*).

Driver

Recebe transações de entrada, gera estímulos.

Monitor

Recolhe as respostas do DUV, verificando o protocolo de saída, produz transações de saída.

Driver e Monitor são totalmente independentes.

Checker

Compara as transações de saída recebidas do monitor com um modelo de referência. É bom ser reutilizável, ou seja, depender pouco do DUV.

Existem diferenças significativas entre o projeto do Ambiente de Teste por Simulação de sub-bloco e o projeto do Ambiente de Teste por Simulação no nível do núcleo. Em ambos os casos, o mais importante é garantir que a cobertura de testes seja adequada.

Ambiente de Teste por Simulação de Sub-bloco

Devido ao fato de que estes módulos quase nunca possuem interfaces bidirecionais, pode-se desenvolver um único Ambiente de Teste por Simulação que gere um conjunto de entradas para as portas do sub-bloco e verifique as portas de saída do mesmo. Na maioria dos sistemas digitais, estas entradas não são aleatórias, mas sim um conjunto de transações que devem ocorrer em uma determinada porta (Figura G.3).

- **Geração de Estímulos**

Quando se projeta um sub-bloco, se pode especificar os tipos de transações permitidas em determinada porta do sub-bloco. Por exemplo, a escrita em um registrador consiste em uma seqüência específica de dado, endereço e mudança de pinos de controle. Qualquer outra seqüência de ações é ilegal. No projeto do restante do núcleo, é necessário garantir que nenhum outro sub-bloco que envie sinais para esta porta gere transações ilegais.

Uma vez definido o conjunto de transações legais para as portas de entrada, é necessário gerar seqüências destas transações com valores apropriados para os dados e endereços, para testar o sub-bloco. Analisa-se a funcionalidade do sub-bloco para determinar seqüências úteis, que servirão para verificar se o comportamento do sub-bloco está de acordo com a especificação, e então são feitos os testes de casos extremos. Após serem feitos todos os testes, deve-se executar uma ferramenta de cobertura de código, que fornece uma boa indicação da integridade do conjunto de testes.

- **Verificação de Saídas**

A geração dos casos de teste é apenas a primeira parte da verificação. É necessário analisar as reações do projeto para verificar se ele está funcionando corretamente. Esta análise pode ser feita manualmente, através da monitoração das

saídas em um visualizador de formas de onda. Porém, este processo tende a produzir muitos erros, sendo necessário um verificador de saídas automático para o Ambiente de Teste por Simulação. Este verificador deve ser exclusivo para o sub-bloco a ser testado, mas existem alguns aspectos comuns à maioria deles:

(i) é possível verificar que somente transações legais são geradas pelas portas de saída do projeto;

(ii) é possível verificar que transações específicas são respostas corretas às transações de entrada geradas.

Ambiente de Teste por Simulação do Núcleo

É possível estender os conceitos utilizados em Ambiente de Teste por Simulações de módulos para Ambiente de Teste por Simulações usados para testar os núcleos. Estando os módulos integrados dentro do núcleo, constrói-se um Ambiente de Teste por Simulação que, novamente, gera automaticamente as transações para as entradas do núcleo e testa as transações das portas de saída. Existem algumas razões pelas quais deve-se desenvolver um Ambiente de Teste por Simulação mais poderoso e bem documentado neste nível:

(i) o projeto está mais complexo, exigindo cenários de teste também mais complexos;

(ii) mais projetistas (talvez toda a equipe que desenvolveu os módulos) estarão trabalhando na verificação do núcleo.

O Ambiente de Teste por Simulação deverá ser fornecido ao usuário juntamente com o núcleo, para que ele possa testar o núcleo.

Nota: Para maiores informações consulte as referências [20], [26] e [28].

APÊNDICE H. – Programa da Interface (Núcleo)

Para maiores informações procure o autor ou orientadores desta dissertação.

- O programa Núcleo da Interface está disponível em CDROM na miniblio/DEE/UFCG bloco CG;
- disponível na Internet (ano 2004): https://lad.dsc.ufcg.edu.br/svn/pci/tronco
- ou entre em contato com o autor: ricardo@cefetam.edu.br

APÊNDICE I. – Programas para Teste de Simulação (Mestre e Escravo)

Para maiores informações procure o autor ou orientadores desta dissertação.

- O programa de Teste de Simulação da Interface está disponível em CDROM na miniblio/DEE/UFCG bloco CG;
- ou entre em contato com o autor: ricardo@cefetam.edu.br

APÊNDICE J. – Diagrama do Circuito Elétrico da Interface

Apresenta-se neste Apêndice os circuitos elétricos referentes ao núcleo implementado em FPGA, o ambiente de configuração da Interface.

Para maiores informações procure o autor ou orientadores desta dissertação.

- Os Diagramas dos Circuitos Elétricos da Interface estão disponíveis em CDROM na miniblio/DEE/UFCG bloco CG;
- ou entre em contato com o autor: ricardo@cefetam.edu.br