UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Alex Sandro Moretti

Um Modelo de Controle de Dispositivos Através do Barramento PCI: Core PCI

Dissertação submetida à Universidade Federal de Santa Catarina como requisito final para a obtenção do titulo de Mestre em Ciência da Computação

Professor João Bosco da Mota Alves, Dr.
Orientador

Florianópolis, Março de 2003.

Um Modelo de Controle de Dispositivos Através do Barramento PCI: Core PCI

Alex Sandro Moretti

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração da Computação Aplicada e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Professor Fernando A. Ostuni Gauthier, Dr (Coordenador)

Banca Examinadora

_____

Professor João Bosco da Mota Alves, Dr (orientador)

_____

Professor Luiz Fernando Jacinto Maia, Dr

_____

Professor Luiz Alfredo Soares Garcindo, Dr

Agradecimentos

Agradeço de coração aos meus pais, João Moretti Neto e Marilea da Luz Moretti, que são exemplos de lutadores e vencedores, tanto em aspectos morais, como profissionais. Eles me deram força e condições para continuar e finalizar o mestrado, além de dar apoio moral, conselhos e principalmente acreditar no meu potencial. Agradeço as minhas irmãs, Sabrina da Luz Moretti e Cristini da Luz Moretti que me incentivaram e acompanharam o trabalho, sempre com uma visão otimista do projeto. Agradeço a minha noiva Helen Monteiro, que sempre esteve ao meu lado nos piores momentos e me ergueu quando eu estava mais fraco. Agora vejo, que por traz de um grande homem existe uma grande mulher. Ao meu orientador João Bosco de Mota Alves, por ter me aceitado orientar-me para o mestrado. Além de ser meu orientador, foi e é um amigo e companheiro na minha jornada de trabalho, sendo bem acessível, compreensível e flexível. Ao Augusto Einsfeldt, que sempre foi paciente ao me fornecer informações técnicas sobre VHDL e FPGA. Ao Ewerton Artur Cappellati, que forneceu algumas informações sobre simulação e partes técnicas em VHDL. E agradeço também às pessoas e aquelas que, direta ou indiretamente, sempre me ajudaram e contribuíram.

Um agradecimento muito especial a DEUS e seus súditos por me confortar nas horas difíceis e nas noites mais frustradas. Agradecer por me ter renovado o ânimo para continuar e concluir meu trabalho, que em certos momentos pensei em desistir.

Resumo

Este trabalho é direcionado ao desenvolvimento de um protocolo PCI, o qual foi aqui denominado, Core PCI. O Core PCI é um núcleo que contém os controles básicos de acesso a um dispositivo PCI de 32bits e 33Mhz. Ele foi desenvolvido para uso geral, por isso, somente foram implementados, os acessos a dispositivos de I/O, mapeados como RAM. Neste trabalho são descritos detalhes e características do barramento PCI, tais como: sinais do barramento, funcionamento do protocolo de transação de dados, configuração plug and play e inicialização dos dispositivos PCI no *boot* do computador. Também se encontram, detalhes de uma aplicação usando o Core PCI e um software de alto nível para controle da aplicação. O Core PCI foi desenvolvido em um chip FPGA, utilizando-se de uma linguagem de descrição de hardware VHDL e utilizando-se também de simuladores para o código VHDL. Ainda neste trabalho encontram-se os cuidados e passos a serem seguidos para a elaboração do hardware do aplicativo e do Core PCI.

Abstract

This work was directed to development of a PCI protocol, witch was denominated here, Core PCI. The Core PCI is a nucleus that has the access basic controls to the PCI device of 32bits, 33Mhz and 5 volts power supply. It was developed for general use, so only was implemented the accesses to I/O's devices mapped as RAM. In this work (dissertation) are described details and characteristics of the PCI bus. Such as: signals of the bus; functioning of the transaction of information protocol; plug and play configuration and initialization of the PCI devices in the *boot* of the computer. Also are found, application details of an using the Core PCI and a high level software for application control. The Core PCI are developed in the FPGA chip, and are used a description language VHDL. Still in this dissertation are found the careful and steps to be followed to elaboration of the hardware of the application, and of the Core PCI.

Sumário

Resumo iv
Abstract v
Índice de Figuras x
Índice de Tabelas xii
Lista de Abreviaturas xiii
1 Introdução 1
2 Definições de Símbolos e Sinais 3
2.1 Tipos de Estados de Sinais 3
2.2 Representações de Sinais Invertidos 3
2.3 Representação de Grupos de Sinais 4
2.4 Representações das Bases Numéricas 4
2.5 Simbologias de Representações de Estados dos Sinais 4
2.5 Símbolos Usados nos Algoritmos em VHDL 5
2.6 5
3 Barramento PCI 6
3.1 Comentários 6
3.2 Características Principais 7
3.3 Características Físicas dos *Slots* e Dispositivos PCI's 7
3.4 Velocidade e Largura do Barramento de Dados 11
3.5 Arquitetura dos Barramentos 12
3.6 Componentes de um Barramento 13
3.7 Sinais do Barramento 14
3.7.1 Sinais de Sistema 15
3.7.2 Sinais de Data e Endereço 16
3.7.3 Sinais de Controle de Transferência de Dados 16
3.7.4 Sinais de Erro de Paridade 17
3.7.5 Sinais de Interrupções 18
3.7.6 Sinais Adicionais 18
3.8 Definição de Transação 18
3.8.1 Fase de Endereço 18
3.8.2 Fase de Dados 19
3.8.3 Ciclo de Inversão do Barramento 19
3.9 Comandos de Tipo de Transações 19
3.9.1 Reconhecimento de Interrupção 20
3.9.2 Ciclo Especial 20
3.9.3 Leitura e Escrita em I/O 20
3.9.4 Leitura e Escrita em Memória 21
3.9.5 Leitura e Escrita no Espaço de Configuração 21
3.9.6 Comando de Leituras Múltiplas 21
3.9.7 Ciclo de Endereço Duplo 21
3.9.8 Comando de Leitura de Linha em Memória 21
3.9.9 Comando de Escrita e Invalidade na Memória 21

3.10 Modos de Endereçamentos 22
3.10.1 Modo de Endereçamento a Registradores de Configuração 22
3.10.2 Modo de Endereçamento em I/O 23
3.10.3 Modo de Endereçamento em Memória 23
3.11 Inicio de uma Transação 24
3.11.1 Tempo de Seleção do Dispositivo 25
3.12 Transação de Leitura 25
3.13 Transação de Escrita 26
3.14 Interrupção e o Reconhecimento de Interrupção 28
3.15 Fim de uma Transação 30
3.15.1 Terminação da Transação pelo Alvo 30
3.15.1.1 Tente Novamente (Retry) 30
3.15.1.2 Desconecte (Disconnect) 31
3.15.1.3 Aborte (Abort) 32
3.15.2 Terminação da Transação pelo Iniciador 32
3.15.2.1 Terminação por Preempção 33
3.15.2.2 Terminação por Aborto 33
3.16 Geração de Paridade 33
3.17 Checagem de Paridade 34
3.17.1 Checagem de Paridade em Erro em Sistema 35
3.17.2 Checagem de Paridade em Erro em Dados 36
3.17.3 Tratamento de Erro 37
3.18 Tempo de Latência 38
3.19 Transação Rápida (Fast Back to Back) 38
3.20 Resposta a uma Violação do Protocolo 38
3.21 Exemplos de Transações 38
3.21.1 Transação de Leitura nos Registradores de Configuração 39
3.21.2 Transação de Escrita nos Registradores de Configuração 40
3.21.3 Transação de Leitura em Memória ou em I/O de Modo Simples 41
3.21.4 Transação de Escrita em Memória ou em I/O de Modo Simples 42
3.21.5 Transação de Leitura em Memória ou em I/O em Rajadas 43
3.21.6 Transação de Escrita em Memória ou I/O em Rajadas 43
3.22 Registradores de Configuração 44
3.22.1 Registrador de Identificação do Fornecedor 45
3.22.2 Registrador de Identificação do Dispositivo 45
3.22.3 Registrador de Comando 46
3.22.4 Registrador de Status 48
3.22.5 Registrador de Identificação de Revisão 50
3.22.6 Registrador de Código de Classe 50
3.22.7 Registrador de Tamanho de Linha de Cache 51
3.22.8 Registrador de Tempo de Latência 51
3.22.9 Registrador de Tipo de Cabeçalho 52
3.22.10 Registrador para Auto Teste 52
3.22.11 Registradores de Endereços Bases 52
3.22.11.1 Registrador de Espaço de Endereçamento de I/O 53
3.22.11.2 Registrador de Espaço de Endereçamento de Memória 53
3.22.11.3 Como a BIOS Determina o Tamanho do Endereçamento 54
3.22.12 Registrador de Identificação do Fornecedor de Subsistema 55
3.22.13 Registrador de Identificação de Subsistema 55
3.22.14 Registrador de Endereço Base para Expansão da ROM BIOS 56
3.22.15 Registrador de Linha de Interrupção 56

3.22.16 Registrador do Pino de Interrupção 56
3.22.17 Registrador de Mínimo Concedido 57
3.22.18 Registrador de Máxima Latência 57

3.23 Reconhecimento dos Dispositivos PCI's no *Boot* do Computador 57

4 O Protocolo PCI Proposto 62

4.1 Visão Geral 62

4.2 Funcionamento 62

4.3 Aplicações para o Uso do Core PCI 62

4.4 Registradores 63

4.4.1 Registradores de Aplicação 63

4.5 Sinais da Aplicação 64

5 Prototipação 65

5.1 Estudo Sobre os Chips Re-configuráveis FPGA's e CPLD's 65

5.1.1 FPGA 65
5.1.2 CPLD 67
5.1.3 Escolha do FPGA para o Core PCI 67

5.2 Definição de VHDL 67

5.3 Definição do Hardware para o Core PCI 68

5.4 Tipo de Aplicação 69

5.5 Ferramentas Usadas para o Desenvolvimento 70

5.6 Criação do Hardware do Protótipo 71

5.6.1 Criação do Esquema Elétrico do Protótipo 71
5.6.2 Criação da Parte Física do Protótipo 72
5.6.2.1 Especificações dos Tempos 72
5.6.2.2 Distribuição dos Pinos no FPGA 73
5.6.2.3 Posicionamento do FPGA na Placa de Circuito Impresso 74
5.6.2.4 Considerações de Alguns Sinais do Barramento PCI 75
5.6.2.5 Elaboração da Placa de Circuito Impresso 75

5.7 Descrevendo o Core PCI em VHDL 77

5.7.1 Componentes para Um Core PCI 77
5.7.1.1 Estado de Máquina 77
5.7.1.2 Algoritmo de Estado de Máquina 79
5.7.1.3 Acesso aos Registradores de Configuração 80
5.7.1.4 Algoritmo de Acesso aos Registradores de Configuração 80
5.7.1.5 Acesso aos Registradores de Aplicação 81
5.7.1.6 Algoritmo de Acesso aos Registradores de Aplicação 81
5.7.1.7 Controle de Paridade 82
5.7.1.8 Algoritmo de Geração de Paridade 82
5.7.1.9 Suporte de Interrupção 82
5.7.1.10 Caminhos Multiplexados 82
5.7.1.11 Controle 82
5.7.1.12 Algoritmo de Controle 83
5.7.2 Escrevendo o Core PCI 84

5.8 Características do Core PCI Implementado 88

5.9 Softwares das Aplicações 89

5.9.1 Software em Ambiente DOS 89
5.9.2 Software para o AVR 89
5.9.3 Protocolo Comunicação entre Software e *Firmware* 92

5.10 Executando o Projeto 97
5.10.1 Adaptações nos Computadores para Teste do Protótipo 97
5.10.2 Gravando o Core PCI no FPGA 99
5.10.3 Testando o Core PCI 99
6 Conclusão 102
Glossário 103
Referências Bibliográficas 105
Anexos 106
1 Regras de Comportamento dos Sinais do Barramento PCI 106
2 Regras na Fase de Endereço e no Inicio de uma Transação 106
3 Regras nas Fases de Dados 107
4 Regras para Finalização de uma Transação 109
4.1 Regras para o Iniciador 109
4.2 Regras para o Alvo 110
5 Regras para Geração de Paridade 111
6 Regras para Checagem de Paridade 111
6.1 Regras para Erro de Paridade na Fase de Endereço 112
6.2 Regras para Erro de Paridade na Fase de Dados 113
7 Regra para Tratamento de Erro 114
8 Regra para o Core PCI em VHDL 114
8.1 Regra para o Core PCI nos Processos em VHDL 114

Índice de Figuras

Figura 3.1: Dispositivo PCI de 5 volts de 32 bits 8
Figura 3.2: Dispositivo PCI de 5 volts de 64 bits 8
Figura 3.3: Dispositivo PCI de 3.3 volts de 32 bits 8
Figura 3.4: Dispositivo PCI de 3.3 volts de 64 bits 9
Figura 3.5: Dispositivo PCI de 3.3 e 5 volts de 32 bits 9
Figura 3.6: Dispositivo PCI de 3.3 e 5 volts de 32 e 64 bits 9
Figura 3.7: *Slots* do barramento PCI com seus sinais [Mendonça 2002- pg 633] 10
Figura 3.8: Arquitetura dos barramentos 12
Figura 3.9: Sinais do barramento PCI 15
Figura 3.10: Endereçamento do tipo 0 para acesso aos registradores de configuração 22
Figura 3.11: Endereçamento do tipo 1 para acesso aos registradores de configuração 22
Figura 3.12: Tempo de seleção do dispositivo 25
Figura 3.13: Transação de leitura 26
Figura 3.14: Transação de escrita 27
Figura 3.15: Reconhecimento de interrupção 30
Figura 3.16: Terminação por tente novamente (retry) da transação PCI 31
Figura 3.17: Terminação por desconexão (disconnect) da transação PCI 32
Figura 3.18: Terminação por aborto (abort) da transação PCI 33
Figura 3.19: Exemplo de cálculo da paridade 34
Figura 3.20: Checagem de erro de paridade 37
Figura 3.21: Transação de leitura nos registradores de configuração 39
Figura 3.22: Transação de escrita nos registradores de configuração 41
Figura 3.23: Transação de leitura em memória ou em I/O de modo simples 42
Figura 3.24: Transação de escrita em memória ou em I/O de modo simples 42
Figura 3.25: Transação de leitura em memória ou em I/O em rajadas 43
Figura 3.26: Transação de escrita em memória ou I/O em rajadas 44
Figura 3.27: Registradores de configuração dos dispositivos PCIs 45
Figura 3.28: Registrador de comando 46
Figura 3.29: Registrador de status 48
Figura 3.30: Registrador de código de classe 50
Figura 3.31: Registrador de endereço base de I/O 53
Figura 3.32: Registrador de endereço base de memória 53
Figura 3.33: Pino de interrupção 56
Figura 3.34: Inicialização da BIOS PCI 59
Figura 3.35: Tela do *boot* do computador 60
Figura 4.1: Arquitetura do Core PCI 62
Figura 4.2: Registradores de aplicação 63
Figura 5.1: Estrutura interna do FPGA xc2s15 66
Figura 5.2: Diagrama da estrutura do protótipo 68
Figura 5.3: Interface do software 70
Figura 5.4: Software para gerar esquema eletrônico do protótipo 72
Figura 5.5: Tempos de propagação dos sinais no barramento PCI 73
Figura 5.6: Distribuição dos sinais do barramento PCI no FPGA 74
Figura 5.7: Posicionamento do FPGA sobre a placa de circuito impresso 75
Figura 5.8: Software para gerar a placa de circuito impresso 76
Figura 5.9: Placa roteada e pronta para fabricação 76
Figura 5.10: Estado de máquina do Core PCI 78
Figura 5.11: Ambiente de desenvolvimento do Core PCI 84
Figura 5.12: Ambiente de simulação do Core PCI 86
Figura 5.13: Distribuição da área interna do FPGA recomendada para desenvolvimento 86
Figura 5.14: Analise de propagação de sinal no FPGA 87
Figura 5.15: Especificando área no FPGA com o Floorplanner 87
Figura 5.16: Fluxograma do software de aplicação DOS 90
Figura 5.17: Fluxograma do software de aplicação AVR 91
Figura 5.18: Gravador do microcontrolador AVR 92
Figura 5.19: Estrutura dos registradores do protocolo MDH 93
Figura 5.20: Fluxograma de gravação de dados no *firmware* através do protocolo MDH 95
Figura 5.21: Fluxograma de leitura de dados no *firmware* através do protocolo MDH 95

Figura 5.22: Fluxograma do protocolo MDH de leitura e escrita no *firmware* 96
Figura 5.23: Fluxograma de sincronismo do protocolo MDH no software 97
Figura 5.24: Ambiente de desenvolvimento do protótipo 98
Figura 5.25: Microcomputador adaptado para comportar o hardware do Core PCI 98
Figura 5.26: Software para configuração do FPGA 99
Figura 5.27: Reconhecimento do Core PCI pelo Windows 100
Figura 5.28: Aplicação está acessando o relógio I2C 101
Figura 5.29: Monitoramento, via porta serial, da comunicação entre software e *firmware* 101

Índice de Tabelas

Tabela 3.1: Comandos de tipos de transações do barramento PCI 20
Tabela 3.2: Combinações dos sinais #CBE em acesso em I/O 23
Tabela 3.3: Tipos de acesso em memória em rajadas de dados em dados (burst) 23
Tabela 3.4: Códigos de interrupções 29
Tabela 3.5: Tempos do acionamento do sinal #DEVSEL 49
Tabela 3.6: Códigos de classe 51
Tabela 3.7: Tipo de acessos de memória em registrador base de endereço 54
Tabela 3.8: Espaço de endereçamento requeridos no registrador BAR para acesso em I/O 54
Tabela 3.9: Espaço de endereçamento requeridos no registrador BAR para acesso em memória 55
Tabela 3.10: Pinos de interrupções 57
Tabela 5.1: Indicação de potência de consumo do periférico conectado no barramento PCI 75
Tabela 5.2: Principais serviços da interrupção 1Ah do barramento PCI utilizados pelo software 91
Tabela 5.3: Comandos do protocolo MDH 93
Tabela 5.4: Posições de memórias no *firmware* para o protocolo MDH 94

Lista de Abreviaturas

AGP	Accelerated Graphics Port
BAR	Base Address Registers
BIOS	Basic Input and Output System
CAD	Computer Aided Design
CLB	Configurable Logic Block
CMOS	Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Process Unit
DIP	Dual Inline Package
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GPIB	General Purpose Interface Bus
HEX	Hexadecimal
ID	Identificador
I/O	Input / Output
IOB	Input Output Block
JTAG	Joint Test Action Group
LVCMOS	Low Voltage Metal Oxide Semiconductor
LVTTL	Low Voltage Transistor Transistor Logic
ISA	Industry Standard Architecture
MHz	Mega Hertz
MSI	Message Signaled Interrupts
NA	*Nibble* Alto
NB	*Nibble* Baixo
PC	Personal Computer
PCI	Peripheral Component Interconnect
PLCC	Plastic J-leaded Chip Carrier
PNP	Plug and Play
RAM	Random Access Memory
SRAM	Static Random Access Memory
TCK	Test Clock (JTAG)

TDI	Test Data Input (JTAG)
TDO	Test Data Output (JTAG)
TMS	Test Mode Select
TQFP	Thin Plastic Gull Wing Quad Flat Package
TTL	Transistor – Transistor Logic
VESA	Bus - Video Electronics Standards Association
VHDL	Very High Speed Integrated Circuits Hardware Description Languagem
VME	Bus - VersaModule Eurocard bus

1 Introdução

Por muito tempo, projetistas de aplicação de hardware usaram o barramento local do processador para criarem aplicações integrando hardware e software, como o barramento ISA do computador PC AT. Até então, os mesmos não tinham grandes problemas, por ser, o barramento ISA, um barramento local do processador com os acessos diretos e simples. Para o desenvolvimento deste hardware, os níveis de complexidade são baixos por três razões: a) o hardware não requer tantos circuitos para controle. b) o barramento trabalha com uma freqüência baixa de 8 Mhz. c) a largura de barramento de dados é de 8 ou 16 bits. Por estas razões, a construção do hardware, principalmente a placa de circuito impresso com suas disposições de componentes, não é crítica e possibilita a qualquer projetista, com um pouco de experiência, desenhar o projeto.

Atualmente o quadro é diferente, o projeto e as arbitrariedades da modernidade nos forçam a criar equipamentos mais rápidos e softwares mais complexos para atender as demandas de velocidades. Um barramento de 8Mhz é um gargalo imenso no tráfego de dados entre os dispositivos. Com a evolução dos barramentos tais como VESA, VME, MicroChannel e outros, chegou-se ao barramento, PCI sendo um padrão universal para todas as plataformas computacionais. Este barramento é dotado de um protocolo complexo de transferência de dados de alta performance. Com a grande difusão do protocolo PCI, o barramento ISA foi sendo, aos poucos, eliminado do mercado. Hoje existem poucos fabricantes que incorporam o barramento ISA em seus computadores. Com esta constante diminuição do uso do barramento ISA, tornou-se inviável produzir estes dispositivos. Então, esses mesmos projetistas, desenham seus novos dispositivos a usar o barramento PCI. Porém, o barramento PCI é mais complexo para uma construção própria, principalmente pelos seguintes motivos:

- Um protocolo de comunicação que requer milhares de portas lógicas;
- Uma freqüência de trabalho de até 66Mhz, aumentando o risco de interferências entre trilhas sobre hardware e um tempo crítico na transferência de dados;
- Por ser de 32 ou 64 bits, aumenta a densidade das trilhas e contribui no fator das reatâncias e de ruídos elétricos sobre hardware.

Pelos motivos acima citados, a construção de um protocolo PCI se torna complexa e cara, por isso existem, hoje, empresas especializadas no ramo de construção de Cores PCI's. Contudo, nada impede que um projetista inicie a sua pesquisa para criar o seu próprio protocolo. Porém é aconselhável fazer uma análise da relação custo / benefício, previsto que o desenvolvimento exige gastos, ferramentas adequadas e tempo.

A motivação desse trabalho deu-se à possibilidade de, futuramente, amenizarmos o custo de adesão dos protocolos PCI, e dos circuitos integrados de alta integridade re-configuráveis (FPGA) e (CPLD), com suas respectivas ferramentas de desenvolvimento.

Essa dissertação está organizada em 6 capítulos.

O capítulo 2 descreve as representações e comportamentos dos sinais do hardware utilizados na dissertação, junto com seus símbolos e bases numéricas.

O capítulo 3 apresenta todo o comportamento do barramento PCI de 32 bits a 33Mhz, junto com seus componentes, definições e características. Também estão representados as arquiteturas e tipos de registradores.

O capítulo 4 apresenta uma aplicação utilizando o protocolo PCI desenvolvido.

O capítulo 5 exibe comentários a respeito dos principais componentes utilizados no protótipo e a respeito também, da linguagem de descrição de hardware (VHDL). São descritos ainda, os passos dados, tanto na utilização das ferramentas e elaboração do hardware, como na criação dos softwares e testes do protótipo.

Cabe ao capitulo 6, concluir essa dissertação, descrevendo as dificuldades obtidas no desenvolvimento e justificando os passos críticos. Seu desfecho sugere possibilidades de trabalhos futuros.

Em anexo contém as regras levantadas, com finalidade de descrever o hardware em VHDL o funcionamento do protocolo PCI.

2 Definições de Símbolos e Sinais

Este capítulo serve de suporte para esclarecimentos e exemplificações em relação aos tipos de sinais e símbolos utilizados no decorrer do documento.

2.1 Tipos de Estados de Sinais

O tipo de estado do sinal representa o sentido da propagação do sinal no seu respectivo pino no componente eletrônico. O sinal de entrada indica que este pino receberá a transação do sinal. Os tipos de estados dos sinais encontrados no projeto são:

- **Entrada:** O sentido do sinal no pino é de recepção da transação no componente eletrônico;
- **Saída:** O sentido do sinal no pino é de transmissão no componente eletrônico;
- **Três Estados:** O pino assume três estados. O estado de entrada e saída e o estado de alta impedância. O estado de alta impedância é como se o pino estivesse desconectado fisicamente do resto do hardware;
- **Especial Três Estados:** É um sinal de três estados, com um tratamento especial sobre o barramento, ou seja, após sua ativação, é necessário retornar à sua desativação, no mínimo um clock antes de ir à alta impedância. Somente um dispositivo por vez pode acionar este pino;
- **Coletor Aberto:** Neste pino não se encontra um componente ativo entre a saída e a tensão de alimentação (VCC ou GND). Permite interligar várias saídas entre si. Assim, o possível acionar múltiplos dispositivos simultaneamente através da lógica AND, ou ler um dos dispositivos através da lógica OU. Para garantir a desativação desse pino, é necessário um resistor de *pull up* ou *pull down*.

2.2 Representações de Sinais Invertidos

Os sinais invertidos são representados no documento pelo símbolo “#”. Este sinal indica que os pinos são ativados em nível baixo. Por exemplo:

#FRAME = 1 > Sinal desativado

#FRAME = 0 > Sinal ativado
IDSEL = 1 > Sinal ativado
IDSEL = 0 > Sinal desativado

2.3 Representação de Grupos de Sinais

Os sinais do mesmo tipo são conjugados em uma só representação indicando a quantidade de sinais através dos números internos aos colchetes. Por exemplo: o sinal #CBE[3:0] é a conjunção dos sinais #CBE[3], #CBE[2], #CBE[1] e #CB[0].

2.4 Representações das Bases Numéricas

As bases numéricas estão representadas por letras à direita do número. A letra “h” representa a base hexadecimal, a letra “b” representa a base binária, a letra “o” representa a base octal e a letra “d”, a ausência de letras, representa a base decimal. Por exemplo:

ABCDh = Número na base hexadecimal
1274o = Número na base octal
11011b = Número na base binária
2002 ou 2002d = Número na base decimal

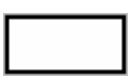
2.5 Simbologias de Representações de Estados dos Sinais

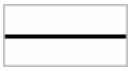
Os diagramas que representam as transações no barramento PCI são representados pelos seguintes símbolos:


O sinal está em nível alto por sustentação do chip;


O sinal está em nível alto por *pull up*;


O sinal está em nível baixo por sustentação do chip;


O sinal é indeterminado;



O pino referente ao sinal está em alta impedância;

O sinal está em inversão no barramento.

2.5 Símbolos Usados nos Algoritmos em VHDL

2.6

pClk ↑ Indica que o procedimento será ativado na borda de subida do clock.

pClk ↓ Indica que o procedimento será ativado na borda de descida do clock.

AND Operador lógico AND.

OR Operador lógico OU.

= Sinal de atribuição.

/= Sinal de diferença.

== Sinal de comparação

3 Barramento PCI

Os estudos realizados neste trabalho sobre o barramento PCI tiveram como objetivo, adquirir as informações necessárias para a construção do protótipo PCI. Foram omitidas quaisquer informações referentes aos dispositivos mestres e pontes, e ainda dos serviços do protocolo PCI não utilizados pelo protótipo ou não julgados importantes para implementações futuras no protótipo [1].

3.1 Comentários

A arquitetura PCI é uma arquitetura universal para todos os tipos de computadores, ou seja, ela não especifica para um só tipo de arquitetura, como para Macintosh ou para o PC AT. O barramento PCI é por natureza plug and play dotado por registradores de configuração e passou por quatros revisões, a revisão 1.0 foi em Junho de 1992, a revisão 2.0 foi em abril de 1993, a revisão 2.1 foi nos primeiros trimestres de 1995 e a revisão 2.2 foi em fevereiro de 1999 [Abbott 2000].

O protocolo PCI incorpora especificações mantidas e revisadas pelo grupo de interesses especiais para PCI (*PCI SIG*), representando todos os aspectos da industria de computadores como [Abbott 2000]:

- Fornecedores de chips
- Fabricantes de placas mães
- Fornecedores de sistema operacionais e BIOS
- Fornecedores de dispositivos PCI's
- Ferramentas de suportes

[1] Para um estudo mais detalhado sugere-se um estudo na literatura PCI System Architecture de Tom Shaley e Don Anderson da Série PC System Architecture.

3.2 Características Principais

As principais características do barramento PCI são [Finkelstein 1997]:

- **Alta taxa de transferência de dados:** O barramento com velocidade de 33Mhz, a taxa de transferência de dados é de 132MBytes por segundo usando 32 bits de largura de dados, ou é de 264Mbytes por segundo usando 64 bits de largura de dados. Em um barramento com velocidade de 66Mhz, a taxa de transferência de dados é de 264MBytes por segundo usando 32 bits de largura de dados, ou é de 528Mbytes por segundo usando 64 bits. Estes valores não consideram os estados de espera.
- **Expansão:** O barramento PCI pode ser expandido para mais *slots* PCI's utilizando pontes PCI – PCI, originando vários barramentos locais. Quando o tráfego é somente no barramento local, mais de um barramento pode ser executado concorrentemente.
- **Baixo consumo:** A placas mães podem reduzir o consumo de energia diminuindo a taxa de clock do barramento PCI a nível tão baixo que pode chegar a 0 Hz. Todos os dispositivos PCI's devem suportar a freqüência de trabalho entre 0Hz a 33Mhz ou de 0 a 66Mhz dependendo do dispositivo no barramento.
- **Dispositivos de multifunções:** Um dispositivo PCI pode ser de uma ou no máximo oito funções, ou seja, cada função representa um dispositivo lógico. Significa que em um só encapsulamento se pode ter até oito dispositivos independentes.
- **Configuração automática:** Todos os dispositivos PCI's devem ser configurados automaticamente por intermédio de registradores de configuração, tornando-os plug and play sem a intervenção humana.
- **Portabilidade:** O barramento PCI não é dedicado a um tipo de processador. O barramento é um padrão universal para conexão de periféricos PCI nas plataformas computacionais.
- **Inter operação com outros padrões diferentes:** Através das pontes, os barramentos podem-se intercomunicar com outros tipos de barramento, tal como um dispositivo no barramento PCI pode se intercomunicar com um dispositivo no barramento ISA usando uma ponte PCI – ISA.

3.3 Características Físicas dos *Slots* e Dispositivos PCI's

Os primeiros *slots* foram desenvolvidos para tensões de trabalho de 5 volts. Com as novas revisões do barramento PCI foi incluída a tensão de 3.3 volts para suportar freqüência de trabalho de 66Mhz.

Nas figuras 3.1 a 3.6 são mostrados vários tipos de *slots* e dispositivos em relação a tensões de trabalho e à largura do barramento de dados. A tensão de trabalho do dispositivo é indicada pelo guia de encaixe, o guia mais próximo da extrema direita indica um dispositivo de 3.3 Volts. O guia a mais próximo da extrema esquerda indica dispositivo de 5 Volts. Dispositivos de 64 bits são mais largos que os de 32 bits. A figura 3.7 mostra os *slots* num computador com suas disposições dos seus sinais.

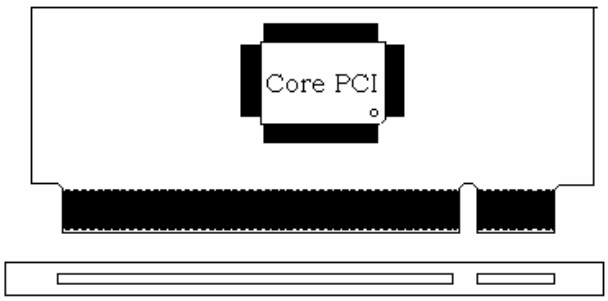


Figura 3.1: Dispositivo PCI de 5 volts de 32 bits

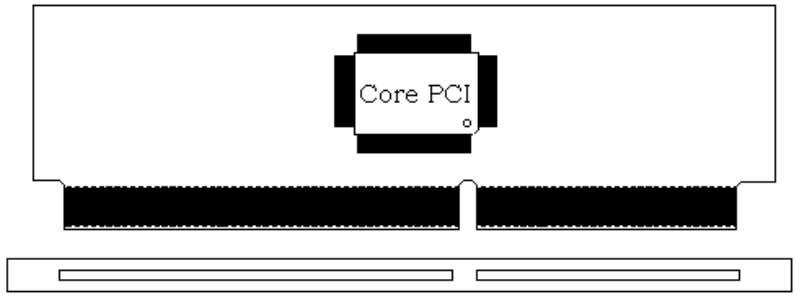


Figura 3.2: Dispositivo PCI de 5 volts de 64 bits

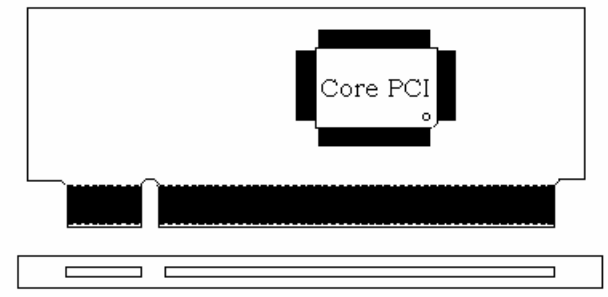


Figura 3.3: Dispositivo PCI de 3.3 volts de 32 bits

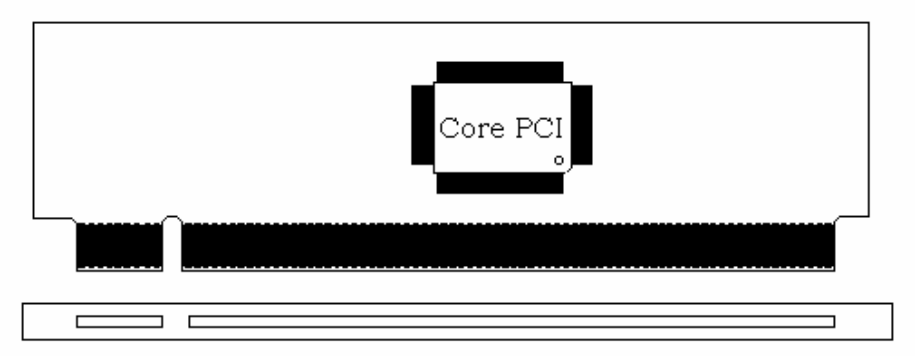


Figura 3.4: Dispositivo PCI de 3.3 volts de 64 bits

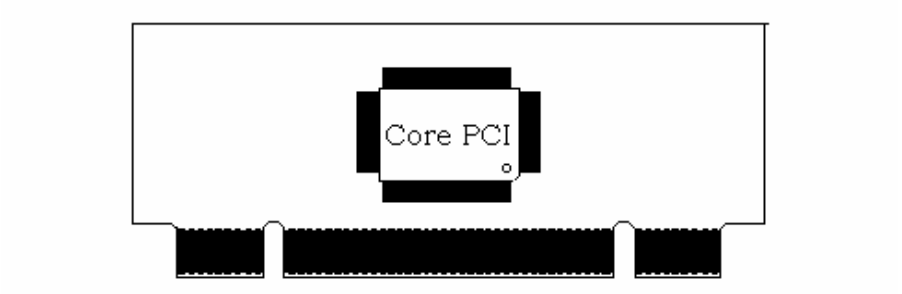


Figura 3.5: Dispositivo PCI de 3.3 e 5 volts de 32 bits

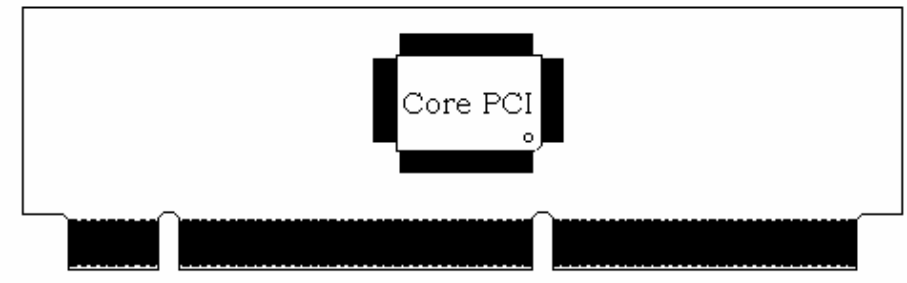


Figura 3.6: Dispositivo PCI de 3.3 e 5 volts de 32 e 64 bits

Cabe salientar que as características aqui apresentadas se aplicam perfeitamente em dispositivos PCIs *onboards*.

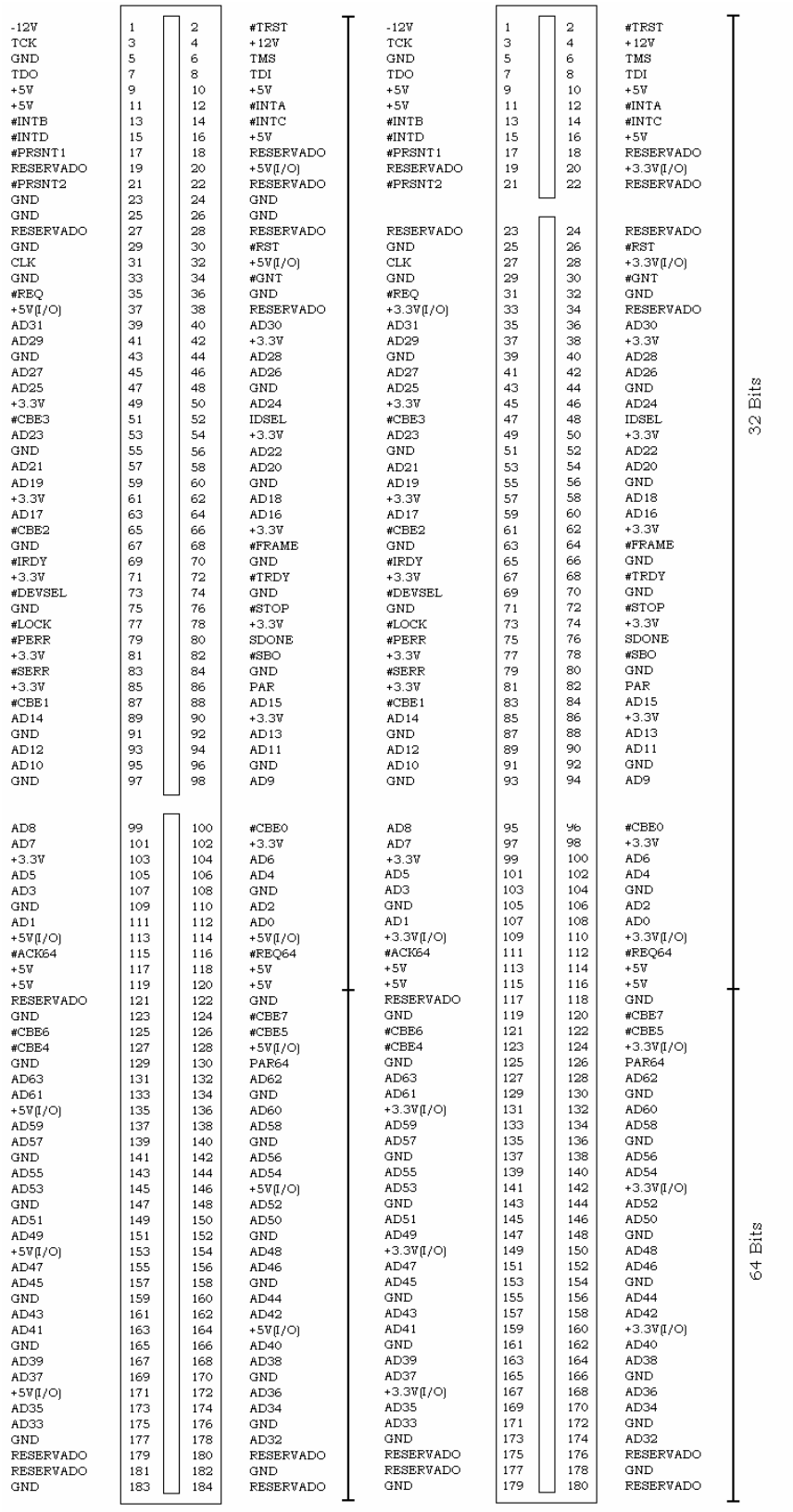


Figura 3.7: *Slots* do barramento PCI com seus sinais [Mendonça 2002- pg 633]

3.4 Velocidade e Largura do Barramento de Dados

A velocidade do barramento pode ser de 33Mhz ou 66Mhz dependendo da arquitetura do computador e do tipo de dispositivo. Atualmente os iniciadores são dotados para trabalharem com as duas velocidades, ficando somente a critério do tipo do dispositivo para seleção adequada da freqüência de trabalho. Esta seleção é feita via software durante a configuração do dispositivo, tanto no *boot* do computador como em qualquer outro momento para configuração via *drive*. Isto pode ser feito por dois modos [Anderson 1999]:

- Modo via software: O software checa o bit 66Mhz capable do registrador de status, ver secção 3.23.4. Se o bit for igual a 1, o dispositivo trabalha a 66Mhz.
- Modo via hardware: No barramentos para dispositivos de 66Mhz existe um novo sinal M66EN em *pull up*, quando este sinal for igual a 1, o dispositivo suporta uma freqüência de trabalho de 66Mhz.

No modo via hardware, o *chipset* seleciona a velocidade de forma automática, sem que o software acesse os registradores de configuração.

No modo via software, o *chipset* acessa, a 33Mhz, os registradores de configuração questionando a velocidade de funcionamento do dispositivo. Assim o *chipset* ajusta a velocidade de acordo com o dispositivo. Seja ele, 33 ou 66Mhz. [Anderson 1999]

Devido ao alto custo, os dispositivos, atualmente, mais utilizados são de velocidade de 33MHz com a largura do barramento de dados de 32 bits. O custo se torna mais alto conforme o aumento da velocidade, ou conforme o aumento da largura do barramento de dados. Este custo adicional refere-se à necessidade de adquirimos chips mais rápidos e um aumento considerável na área do chip e do hardware para um barramento de dados de 64 bits. A justificativa deste custo adicional são os cuidados indispensáveis na construção do hardware contra ruídos elétricos e reatância sobre o hardware em relação ao aumento da velocidade e de área do hardware, principalmente no desenvolvimento do Core PCI no chip. Estes fenômenos ocorrem devido à velocidade de propagação do sinal internamente do mesmo, pois quanto maior a área do chip, maior será a área de propagação dos sinais e menor será a velocidade do dispositivo. Um passo crítico na construção do Core PCI para que funcione adequadamente e permanecendo dentro dos parâmetros de velocidade exigidos pelo *PCI SIG* [Anderson 1999].

Entretanto, em um dispositivo de 66MHz com um barramento de dados de 64 bits, a banda passante de dados é bem alta. Ideal para aplicações onde requer grandes fluxos de dados, ficando a critério do consumidor, fazer uma relação entre custos e benefícios ao adquirir tais dispositivos.

3.5 Arquitetura dos Barramentos

A figura 3.8 exemplifica como são distribuídos os barramentos em um computador do tipo PC AT. Na maioria das arquiteturas computacionais o hospedeiro dos protocolos PCI e AGP já estão embutidos em um único *chipset* norte.

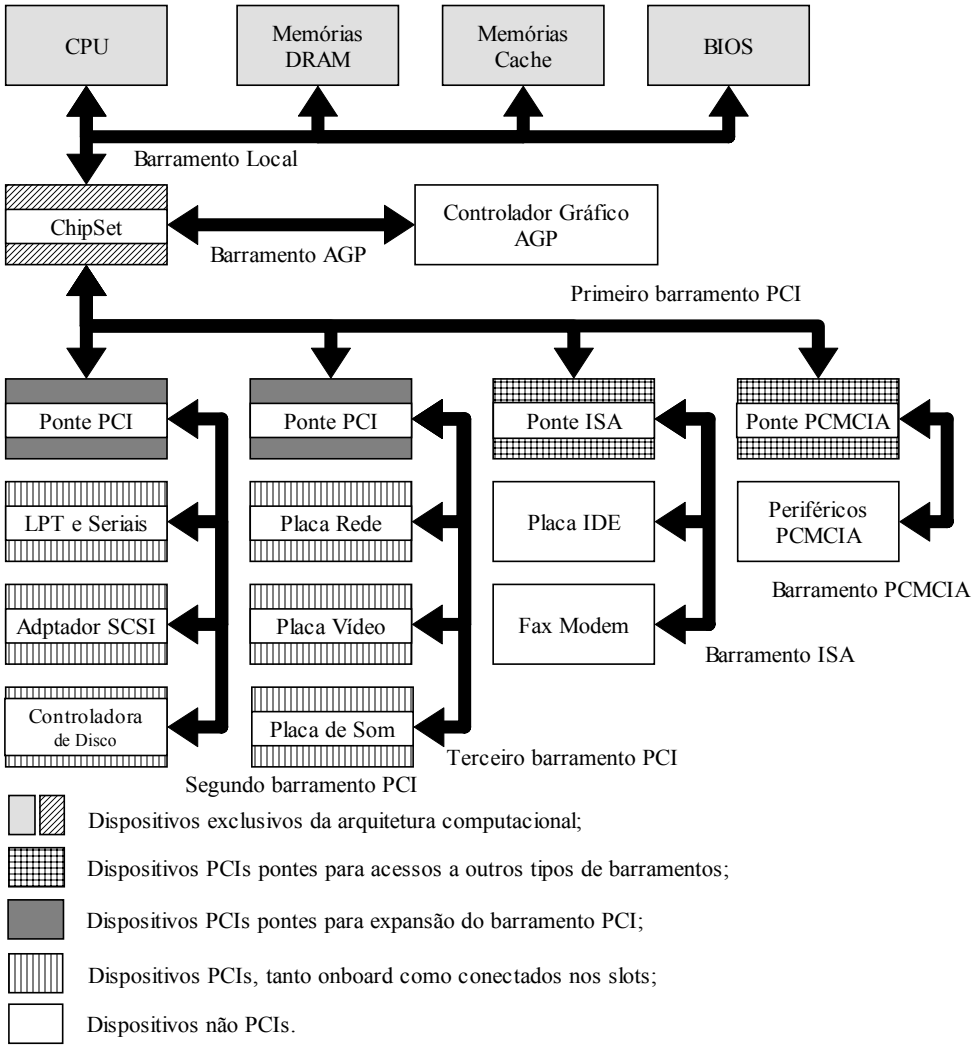


Figura 3.8: Arquitetura dos barramentos

O protocolo AGP mantém compatibilidade com o protocolo PCI. O mesmo é uma extensão da PCI, e tem acesso exclusivo ao *chipset* com objetivo de aumentar a troca de dados e desempenho, ou seja, o barramento não é compartilhado com outros dispositivos PCI's.

Ao observar a figura 3.8, existe mais de um barramento PCI em um único computador, que são interligados por intermédio de pontes PCI-PCI. Cada barramento é identificado por um número da BIOS durante o *boot* do computador e comentado com mais detalhe na secção 3.24.

3.6 Componentes de um Barramento

Existem dois tipos de classes para os dispositivos PCI's em relação a uma transferência de dados: a classe iniciadora e a classe alvo [Anderson 1999] [PCI 1998].

A classe iniciadora: É um dispositivo responsável para iniciar qualquer tipo de transferência de dados em um barramento PCI. Neste dispositivo está implementado, de acordo da revisão da época, todo o protocolo necessário para qualquer tipo de transferência de dados.

A classe alvo: É um dispositivo conectado ao barramento PCI, que será o alvo de uma transferência de dados.

Na arquitetura PCI existem quatro tipos de componentes PCI's referentes às classes anteriormente mencionadas, que são: os componentes Mestres; os componentes Alvos; os componentes Alvos / Iniciadores e os componentes Pontes.

Componente Mestre: Um tipo de dispositivo que pertencente, tanto à classe iniciadora, como à classe alvo, seu objetivo é intercomunicar o barramento local do processador com o primeiro barramento PCI. Dependendo da arquitetura computacional, o iniciador mestre pode estar incorporado junto ao *chipset* norte do computador mostrado na figura 3.8 pelo bloco com linhas transversais.

Componente Alvo: Um tipo de dispositivo pertencente somente à classe alvo. Estes dispositivos podem ser, tanto *onboards*, como conectados aos *slots* do computador, tal como placa de vídeo e placa de rede como mostrado na figura 3.8 pelos blocos com linhas verticais.

Componente Alvo/Iniciador: Um tipo de dispositivo que pertencem tanto à classe iniciadora como à classe alvo. Estes dispositivos podem ser, tanto *onboards*, como conectados aos *slots* do computador, tal como controladores de discos como mostrado na figura 3.8 pelos blocos com linhas verticais. A vantagem desse dispositivo em relação ao dispositivo Alvo, é que este pode tomar conta do barramento fazendo uma transferência de dados para qualquer dispositivo a desejar. Diferente do dispositivo Alvo, que a transferência de dados é somente para o Mestre e deste para o *driver* do sistema.

Componente Ponte: Um tipo de dispositivo que pertencente tanto à classe iniciadora, como à classe alvo, seu objetivo é intercomunicar outros tipos de barramento ao barramento PCI, como: barramento ISA, PCMCIA ou USB. As pontes estão representadas na figura 3.8 pelos blocos com linhas horizontais e verticais.

Componente Ponte PCI: Um tipo de dispositivo que pertencente, tanto à classe iniciadora, como à classe alvo. Seu objetivo é expandir o barramento PCI, originando outros barramentos PCI's locais com a finalidade de delimitar a quantidade de dispositivos por barramento. Reduzirá, assim, a sobrecarga e ruídos elétricos. As pontes estão representadas na figura 3.8 pelos blocos cinza escuros.

Os dispositivos Pontes e Alvos/Iniciadores são considerados classes iniciadoras escravas. Quando um iniciador escravo deseja fazer uma transferência de dados, o mesmo deve pedir ao arbitro do barramento a posse do mesmo, para que não haja conflitos entre os demais iniciadores.

3.7 Sinais do Barramento

Em um barramento PCI existem 124 sinais contendo linhas de endereçamento e de dados de 32 e 64 bits (AD[64:0], PAR e PAR64), linhas de comandos e bytes habilitados (#CBE[7:0]), linhas para reconhecimento de barramento de 64 bits (#REQ64 e #ACK64), linhas de controle (#FRAME, #TRDY, #IRDY, #STOP, #DEVSEL, IDSEL e #LOCK), linhas de paridade (#PERR e #SERR), linhas arbitragem (#REQ e #GNT), linhas de sistemas (CLK e #RST), linhas de interrupções (#INTA, #INTB, #INTC e #INTD) e as linhas de comunicação do padrão JTAG (TDI, TODO, TCK, TMS e #TRST) [Mendonça 2002].

Entretanto, somente 47 desses sinais são obrigatórios para todos os dispositivos PCI's alvos e 49 para os iniciadores em um barramento de 32 bits [PCI 1998]. No protótipo foram usados somente os sinais necessários como mostrado na figura 3.9

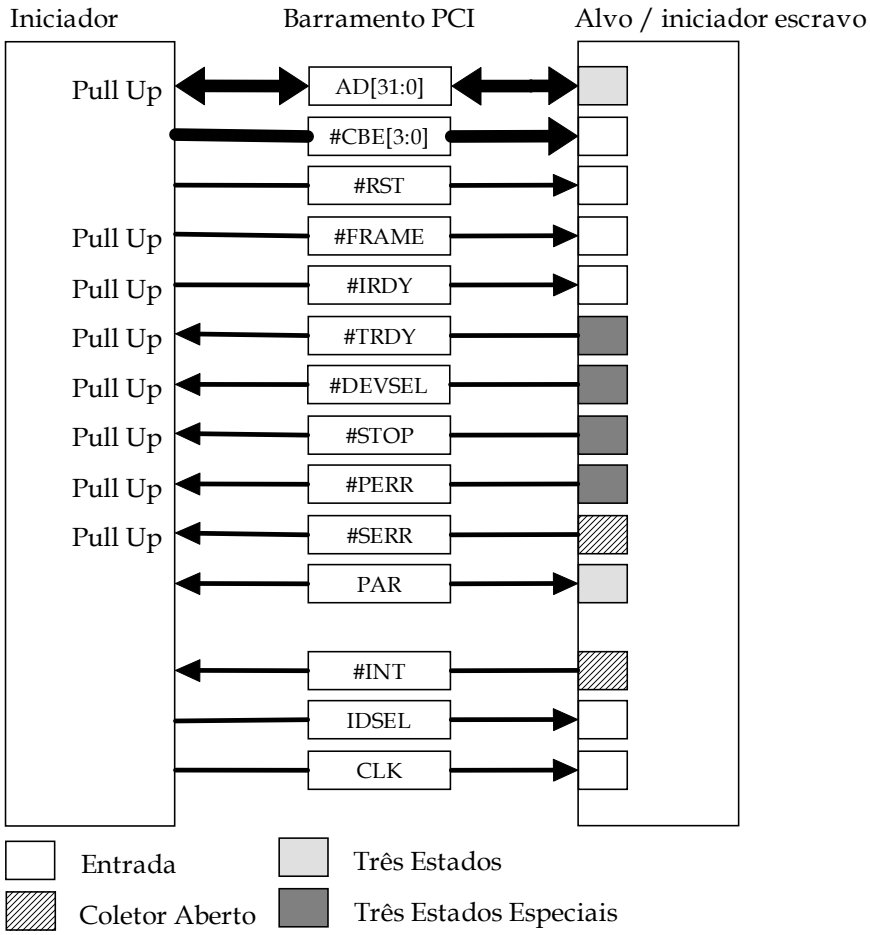


Figura 3.9: Sinais do barramento PCI

3.7.1 Sinais de Sistema

CLK - Clock (Relógio – Sinal de entrada) – Sinal de relógio do barramento PCI. Este sinal sincroniza todas as transações na borda de subida do clock. Opera entre 0 a 33 MHz ou de 0 a 66 MHz, dependendo do dispositivo. O clock pode mudar em qualquer momento dependendo do dispositivo ou até parar se o dispositivo PCI não estiver sendo utilizado. É uma estratégia de economia de energia [Anderson 1999] [Mendonça 2002].

#RST - *Reset* (Inicializar - Sinal de entrada) – Força todos os sinais e registradores dos dispositivos no barramento a irem a seus estados iniciais [Anderson 1999] [Mendonça 2002].

3.7.2 Sinais de Data e Endereço

AD[31:0] - Address and Data (Endereço e dados - Sinal de três estados) – Estes sinais são multiplexados entre: endereço e dados. Na fase de endereço, os sinais transportam o endereço do alvo. E na fase de dados, os sinais transportam dados entre iniciador e alvo [Anderson 1999] [Mendonça 2002].

#CBE[3:0] - Command and Byte Enable (Comando e Habilitador de byte - Sinal de entrada) – Estes sinais são multiplexados entre: tipo de comando de transação e tipo de bytes habilitados em AD[31:0]. Na fase de endereço, os sinais indicam o tipo de transação no barramento. E na fase de dados, os sinais indicam quais os bytes estão válidos em AD[31:0] [Anderson 1999] [Mendonça 2002].

PAR - Parity (Paridade - Sinal de três estados) – Carrega o valor do cálculo de paridade dos sinais AD[31:0] e #CBE[3:0]. Este é transmitido no sentido da transação. No cálculo não se leva em consideração o tipo de transação, nem quais bytes válidos em AD[31:0] por #CBE[3:0], ou seja, todos entram no cálculo em quaisquer circunstâncias. O sinal PAR é válido nos seguintes casos: 1 clock após a fase de endereço; 1 clock após a ativação do sinal #IRDY na transação de escrita (durante a fase de dados); 1 clock após a ativação do sinal #TRDY na transação de leitura (durante a fase de dados). E permanece válido por 1 clock após da fase completa. [Anderson 1999] [Mendonça 2002].

3.7.3 Sinais de Controle de Transferência de Dados

#FRAME - Frame (Amostra - Sinal de entrada) – Na ativação deste sinal o iniciador indica o inicio de uma transação e permanece acionado na mediada que o mesmo requer mais transferências de dados. A sua desativação ocorrerá quando o iniciador estiver pronto a realizar a última fase de dados ou quando a transação for finalizada por algum motivo [Anderson 1999] [Mendonça 2002].

#IRDY - Initiador Ready (Iniciador Pronto - Sinal de entrada) – O iniciador aciona este sinal indicando que está pronto para fase de dados, isto é, na transação de escrita o barramento contém dados válidos para o alvo e para transação de leitura o iniciador está pronto para ler os dados do alvo. Este sinal trabalha em conjunto com o sinal #TRDY, e quando um dos dois estiver desativado, indicará que existe um estado de espera na transação, e quando ambos

forem ativados juntos, indicará que a fase atual foi completa e houve transferência de dados entre iniciador e alvo [Anderson 1999] [Mendonça 2002].

#TRDY - Target Ready (Alvo pronto – Sinal especial de três estados) – O alvo aciona este sinal indicando que está pronto para fase de dados, ou seja, na transação de escrita o alvo está pronto para ler os dados do iniciador e para transação de leitura o dados estão prontos no barramento para o iniciador [Anderson 1999] [Mendonça 2002].

#STOP - Stop (Parar - Sinal especial de três estados) – Ao acionar este sinal o alvo solicita o término precoce da transação [Anderson 1999] [Mendonça 2002].

IDSEL - Initialization Device Select (Seleção de dispositivo para inicialização - Sinal de entrada) – Pela ativação desse sinal, o iniciador solicita uma transação de escrita ou leitura sobre os registradores de configuração do dispositivo [Anderson 1999] [Mendonça 2002].

#DEVSEL - Device Select (Seleção de dispositivo - Sinal especial de três estados) – Ao acionar este sinal, o dispositivo informa ao iniciador que o mesmo é o alvo da transação. Após 4 clock's, se nenhum alvo responder, a transação será abortada pelo iniciador [Anderson 1999] [Mendonça 2002].

3.7.4 Sinais de Erro de Paridade

#PERR - Parity Error (Erro de paridade - Sinal especial de três estados) – Sinal usado para acusar erro de paridade nos sinais AD[31:0] e #CBE[3:0] nas fases de dados. Com exceção da transação especial. Este sinal funciona como entrada e saída em um iniciador e somente de saída num dispositivo PCI alvo. Quando existir algum erro de paridade na fase de dados, o mesmo será acionado dois clocks após a fase completa de dados [Anderson 1999] [Mendonça 2002].

#SERR - System Error (Erro de Sistema - Sinal de saída em coletor aberto) – Sinal usado para acusar erro de paridade dos sinais AD[31:0] e #CBE[3:0] na fase de endereço, ou na fase de dados de uma transação especial ou em qualquer transação de sistema. Se existir um erro o mesmo é acionado dois clock após o AD[31:0] e #CBE[3:0] válidos [Anderson 1999] [Mendonça 2002].

3.7.5 Sinais de Interrupções

#INT[A, B, C e D] - Interruption (Interrupção – Sinal de saídas em coletor aberto) – Os dispositivos acionam estas linhas quando os mesmos requerem atenção dos *drivers* via iniciador. Estes sinais não são sincronizados com o clock do barramento. Quando os mesmos são acionados, eles permanecem até que os *drivers* atendam aos pedidos de interrupções [Anderson 1999] [Mendonça 2002].

3.7.6 Sinais Adicionais

#PRSNT[1:2] – Estes sinais estão presentes nos *slots* PCI's com a finalidade de indicar ao *chipset* do computador, o tipo de potência consumida pela placa conectada no *slot*. Estes sinais estarão em *pull up*. Caso nenhuma placa esteja conectada o *chipset* interpreta a não presença da mesma [Anderson 1999] [Mendonça 2002].

3.8 Definição de Transação

O mecanismo de transferência de dados no barramento PCI é denominado transação. Uma transação é composta por duas fases, fase de endereço e fase de dados [Ewerton 2001]. Estas duas fases existem por causa das mutiplexações dos sinais AD[31:0] e #CBE[3:0] do barramento (conforme secção 3.7.2). Seu objetivo é reduzir o número de trilhas, conseqüentemente diminuirá seu tamanho e o custo dos dispositivos [Anderson 1999] [Mendonça 2002].

3.8.1 Fase de Endereço

Toda transação PCI é iniciada pela fase de endereço e dura um clock do barramento PCI. Nesta fase o iniciador aciona o sinal #FRAME e todos os dispositivos PCI's devem capturar, na borda de subida do clock, o endereço nos sinais AD[31:0] e o tipo de comando de transação nos sinais #CBE[3:0] como exemplificado no clock 2 da figura 3.13 da secção 3.13. Caso o dispositivo seja alvo da transação, o mesmo deve informar ao iniciador acionado o seu sinal #DEVSEL. Após 4 clocks, se nenhum dispositivo responder, a transação será abortada pelo iniciador [Anderson 1999] [Mendonça 2002].

3.8.2 Fase de Dados

Em uma transação existe uma só fase de endereços e uma ou mais fases de dados. As fases de dados, denominadas rajadas (burst), aumentam a taxa de transferência de dados no barramento. A figura 3.13 secção 3.13. nos clocks de 4 a 8 mostra uma transação em rajada [Anderson 1999] [Mendonça 2002].

3.8.3 Ciclo de Inversão do Barramento

Existem momentos em que certas linhas do barramento entram em estado de inversão. Isso ocorre para que não haja contenção do barramento pelos dispositivos PCI. Os sinais e as circunstâncias em que acontece a inversão são [Anderson 1999] [Mendonça 2002]:

- Sinal #IRDY na fase de endereço;
- Sinal #TRDY na fase de endereço;
- Sinal #DEVSEL na fase de endereço;
- Sinal #STOP na fase de endereço;
- Sinais AD[31:0] logo após da fase de endereço numa transação de leitura e no final da transação;
- Sinais #CBE[3:0] no final da transação;
- Sinal #FRAME no final da transação.

3.9 Comandos de Tipo de Transações

Os comandos indicam o tipo de transação que vão ocorrer entre o iniciador e o alvo. O tipo de comando está contido no barramento, nos sinais #CBE[3:0] na fase de endereço [PCI 1998]. Os comandos descritos têm como referencia o iniciador, ou seja, se mencionado comando de escrita, o iniciador escreve no alvo. Todos os dispositivos PCI's devem ser obrigados a responder, pelo menos, os comandos de leitura e escrita nos registradores de configuração, com exceção das pontes. Os outros comandos são opcionais. Os tipos de comandos estão expostos na tabela 3.1 [Anderson 1999] [Mendonça 2002].

#CBE[3:0]	Comandos
0000	Reconhecimento de interrupção
0001	Ciclo especial
0010	Leitura em I/O
0011	Escrita em I/O
0100	Reservado
0101	Reservado
0110	Leitura em memória
0111	Escrita em memória
1000	Reservado
1001	Reservado
1010	Leitura em configuração
1011	Escrita em configuração
1100	Leitura múltipla em memória
1101	Ciclo de endereço duplo
1110	Leitura em linha de memória
1111	Escrita em memória e invalidade

Tabela 3.1: Comandos de tipos de transações do barramento PCI

3.9.1 Reconhecimento de Interrupção

Quando um dispositivo PCI gera uma interrupção em um dos sinais #INT do barramento PCI, é necessário indicar qual interrupção o mestre deve acionar no processador. Para isto existe uma transação especial chamada reconhecimento de interrupção. Isto será detalhado na secção 3.15 [Anderson 1999] [Mendonça 2002].

3.9.2 Ciclo Especial

Este comando fornece uma mensagem de *broadcast* para todos os dispositivos do barramento atual e não se propaga através das pontes [Anderson 1999] [Mendonça 2002].

3.9.3 Leitura e Escrita em I/O

Comandos para ler e escrever em dispositivos mapeados em espaços de endereços de I/O. Estes comandos servem para manter compatibilidade com os dispositivos antigos, em que haja expansões da CPU por mapeamento de I/O como RAM. É recomendo usar comandos de acesso à memória para os dispositivos futuros [Anderson 1999] [Mendonça 2002].

3.9.4 Leitura e Escrita em Memória

Comandos para ler e escrever em dispositivos mapeados nos espaços de endereços de memória [Anderson 1999] [Mendonça 2002].

3.9.5 Leitura e Escrita no Espaço de Configuração

Comandos para ler e escrever em registradores de configuração dos dispositivos PCI's, permitindo a programação dos recursos de hardware exigidos para serem plug and play [Anderson 1999] [Mendonça 2002].

3.9.6 Comando de Leituras Múltiplas

Este comando é usado para memórias que suportam uma pré busca. O iniciador indica ao alvo que deseja ler mais de uma linha de cache antes de se desconectar [Anderson 1999] [Mendonça 2002].

3.9.7 Ciclo de Endereço Duplo

O iniciador usa este comando para indicar o alvo que está usado o acesso ao espaço de endereçamento de memória de 64 bits acima de 4GB [Anderson 1999] [Mendonça 2002].

3.9.8 Comando de Leitura de Linha em Memória

Este comando é usado para memórias que suportam uma pré busca. O iniciador indica o alvo que deseja ler mais linhas de cache. Isto permite ao alvo, preencher a memória de cache de linha, antecipadamente, aumentado o desempenho do barramento [Anderson 1999] [Mendonça 2002].

3.9.9 Comando de Escrita e Invalidade na Memória

Este comando é usado quando o iniciador pretende executar uma escrita completa na linha de cache em uma transação PCI simples. Porém, a transação de escrita completa, dar-se-á, somente, em memórias que suportam uma pré busca. Caso contrário, a transação será de escrita simples.[Anderson 1999] [Mendonça 2002].

3.10 Modos de Endereçamentos

Existe um modo de endereçamento específico para cada tipo de transação. Os tipos de endereçamentos são divididos em: modo de endereçamento a registradores de configuração, modo de endereçamento em I/O e modo de endereçamento a memória [Anderson 1999] [Mendonça 2002].

3.10.1 Modo de Endereçamento a Registradores de Configuração

O acesso aos registradores de configuração dá-se quando a linha IDSEL do dispositivo é acionada na fase de endereço, tornando-o, alvo da transação para configuração. O tipo de acesso é classificado em dois tipos: tipo 0 e tipo 1. O tipo 1, acessa as pontes, e o tipo 0, acessa os demais dispositivos. O tipo de acesso é especificado na fase de endereço pelos sinais AD[1:0] como mostrados nas figuras 3.10 e 3.11 [PCI 1998].

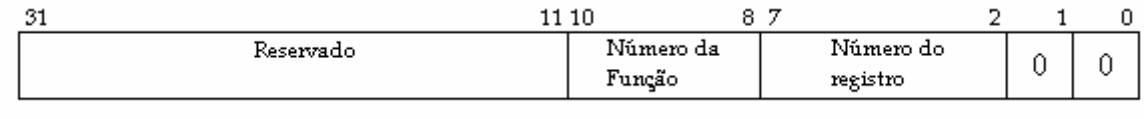


Figura 3.10: Endereçamento do tipo 0 para acesso aos registradores de configuração

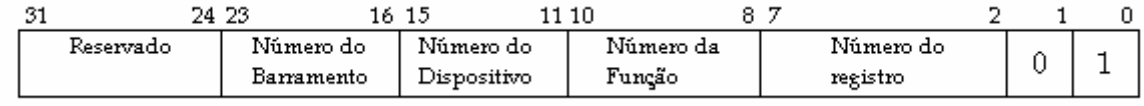


Figura 3.11: Endereçamento do tipo 1 para acesso aos registradores de configuração

Descrições dos grupos de bits:

- **Número de registro** - O endereço da *dword* do registrador de configuração;
- **Número de função** - Valor usado para selecionar a função do dispositivo de multifunções;
- **Número de dispositivo** - Valor usado para selecionar um dos 32 dispositivos em um dado barramento;
- **Número do barramento** - Valor usado para selecionar 1 dos 256 barramento no computador.

Somente a BIOS e ou o *driver* do dispositivo têm acesso aos registradores de configuração. A transação do tipo 0, somente é propagada no barramento PCI local e não se propaga para outros barramentos. Os acessos aos registradores são feitos em *dwords*.

Os primeiros 256 bytes de endereçamento de memória são reservados para os acessos aos registradores de configuração dos dispositivos PCI's. O objetivo é eliminar conflitos de acessos aos dispositivos durante a configuração. Suponhamos que não fossem reservados esses endereços, e o iniciador acessasse os registradores de configuração de um dispositivo. As conseqüências seriam desastrosas, pois qualquer outro dispositivo, já configurado, responderia a transação.

3.10.2 Modo de Endereçamento em I/O

O Endereçamento em I/O é usado para manter compatibilidade com os sistemas baseados na arquitetura PC-AT e não é recomendado para projetos futuros. Todos os 32 bits de endereçamento são para acessar um byte completo. Como mostrador na tabela 3.3. O valor dos sinais AD[1:0] na fase de endereço implica sobre os sinais #CBE[3:0] na fase de dados.

Endereço[1:0]	**#CBE[3:0]**	**Bytes Válidos**
00	xxx0 ou 1111	3, 2, 1, 0
01	xx01 ou 1111	3, 2, 1
10	x011 ou 1111	3, 2
11	0111 ou 1111	3

Tabela 3.2: Combinações dos sinais #CBE em acesso em I/O

Se o acesso da transação não for compatível com o tamanho da porta de I/O, o alvo é livre em abortar a transação.

3.10.3 Modo de Endereçamento em Memória

No modo de endereçamento em memória, os valores nas linhas AD[31:2] contêm o endereço da *dword* alinhada. Os valores das linhas AD[1:0] contêm o tipo de acesso de dados em rajadas (burst) requerido pelo iniciador. A tabela 3.4 mostra os tipos de acessos pelo iniciador [PCI 1998].

AD[1:0]	**Tipo de rajada**
00	Incremento linear
01	Reservado (Alvo deve desconectar)
10	Modo de linha cache wrap (Opcional)
11	Reservado (Alvo deve desconectar)

Tabela 3.3: Tipos de acesso em memória em rajadas de dados em dados (burst)

Todos os alvos devem checar o tipo de comando de acesso em rajadas de dados (burst) na memória. Se o alvo não suporta, o mesmo deverá não aceitar a transação, ou ainda, aceitar com desconexão. O tipo de desconexão pode acontecer de duas maneiras: desconectar com transferência de dados na primeira fase de dados ou desconectar sem transferência de dados na segunda fase de dados, ver secção 3.16.1.1. Essa desconexão serve para os alvos que não suportam a transação em rajadas, mas se o alvo suportar o modo em rajada o mesmo deve no mínimo suportar o incremento linear (AD[1:0] = 00). Em uma transação em rajadas de 32 bits, o incremento do endereço faz-se em uma *dword* ou em duas *dwords* para uma transação de 64 bits. Uma transação com o comando de escrita em memória ou invalidade somente suporta o incremento linear. O dispositivo pode restringir o acesso em espaço de memória, ou abortar a transação, caso seja violado o tipo de acesso. Um alvo pode abortar uma transação caso exista um acesso diferente de byte a um registrador de byte [PCI 1998].

No alvo, a implementação dos comandos depende da aplicação. Mas se forem implementar acessos à memória, o mesmo deve suportar todos os tipos de acessos, mesmo que o comando não seja implementado, forçando-o a substituí-lo por um comando básico. Por exemplo: o iniciador pretende fazer leituras múltiplas no alvo, o qual não suporta, assim o alvo deve executar o comando de leitura simples e desconectar logo em seguida [PCI 1998].

A principal diferença entre o espaço de endereçamento de I/O e espaço de endereçamento de memória, é que o primeiro suporta leituras de pré busca sem efeitos secundários com os comandos opcionais [Abbott 2000]. Os comandos opcionais são:

- Comando de leitura de linha em memória
- Comando de leituras múltiplas a memória
- Comando de escrita e invalidade na memória

3.11 Inicio de uma Transação

Qualquer transação é originada pelo iniciador, o mesmo aciona o sinal #FRAME iniciando a fase de endereço como mostrado no clock 2 da figura 3.13.

Como visto, na secção 3.10 existem três tipos de espaço de endereçamentos, I/O, memória e registradores de configuração. Então o iniciador pode começar uma transação em duas categorias: acesso em registradores de configuração acionando o sinal IDSEL na fase de endereço do dispositivo desejado, tornando-o, alvo da transação, ou fornecendo o endereço do

dispositivo desejado nos sinais AD[31:0] na fase de endereço, onde todos os dispositivos devem decodificá-lo e compará-lo como seus registradores de configuração BAR's, aquele que coincidir é o alvo da transação [Anderson 1999] [PCI 1998].

3.11.1 Tempo de Seleção do Dispositivo

Se o dispositivo for alvo da transação, o mesmo deve sinalizar ao iniciador, o reconhecimento do pedido de transação pela ativação do seu sinal #DEVSEL. Existem três tipos de classe de dispositivos referentes ao tempo de acionamento do sinal #DEVSEL: o dispositivo do tipo rápido, médio e lento, como mostrado na figura 3.12. O do tipo rápido, aciona o sinal #DEVSEL 1 clock após a fase de endereço, o dispositivo do tipo médio, aciona o sinal #DEVSEL 2 clocks após a fase de endereço e o dispositivo do tipo lento, aciona o sinal #DEVSEL 3 clock após a fase de endereço. No quarto clock se nenhum dispositivo responder, o iniciador aborta a transação [Anderson 1999] [PCI 1998].

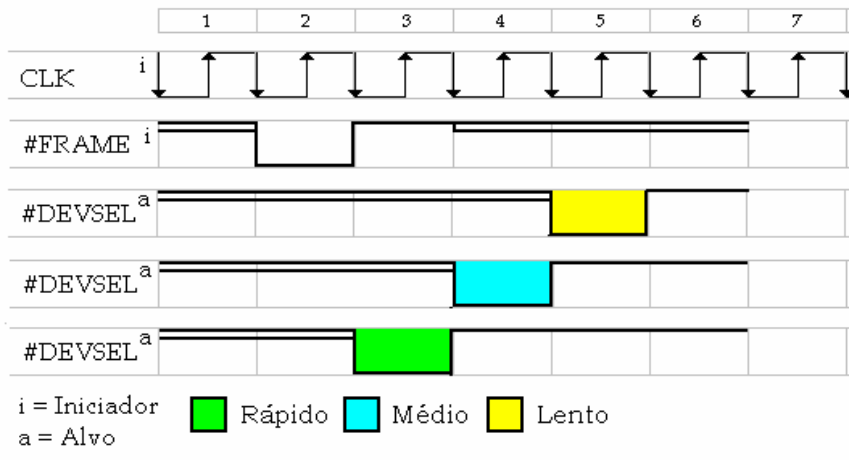


Figura 3.12: Tempo de seleção do dispositivo

3.12 Transação de Leitura

A transação de leitura é mostrada na figura 3.13. O iniciador aciona o sinal #FRAME, indicando o início da transação, e no mesmo instante coloca o endereço do dispositivo desejado e o tipo de comando nos respectivos sinais AD[31:0] e #CBE[3:0], originando a fase de endereço como mostrado no clock 2. Esta ilustração serve para qualquer tipo de leitura [Anderson 1999] [PCI 1998].

Logo após a fase de endereço, existe um clock extra para inversão do barramento mostrado no clock 3, é o tempo necessário para que os dispositivos invertam os seus sinais AD[31:0] de entrada para saída.

Na fase de dados, os sinais #CBE[3:0] indicam quais dos bytes são válidos nos sinais AD[31:0] a serem lidos.

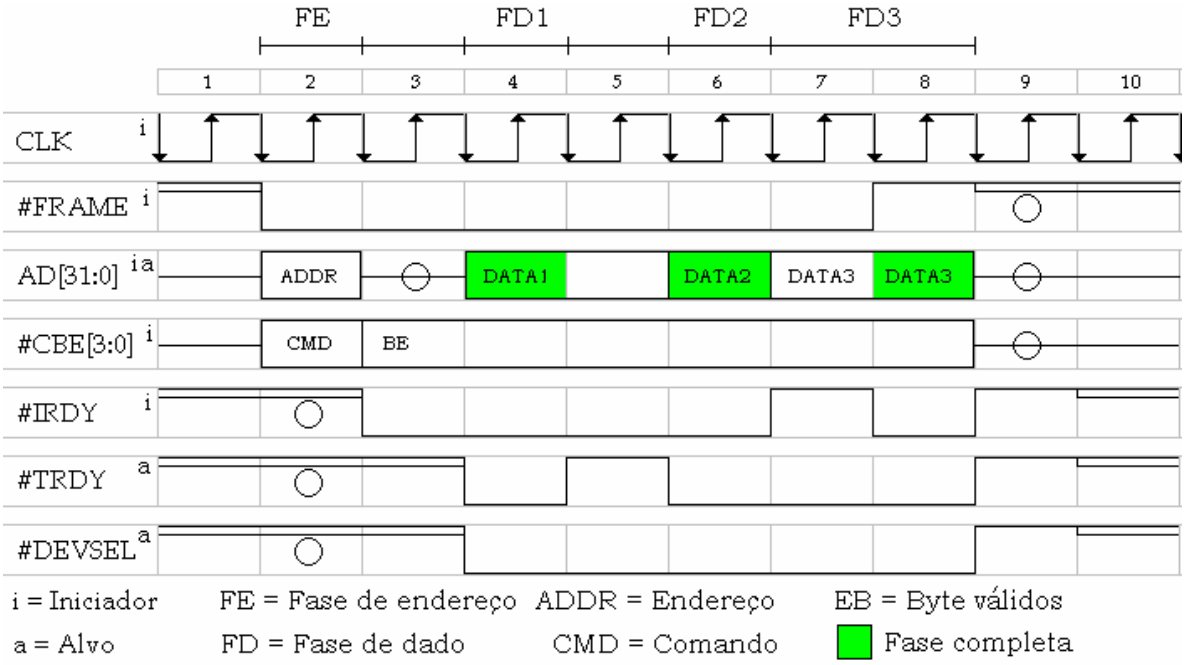


Figura 3.13: Transação de leitura

Em qualquer momento da transação podem ocorrer estados de espera tanto pelo iniciador, via desativação do sinal #IRDY, como pelo alvo, via desativação do sinal #TRDY mostrados nos clocks 5 e 7. Quando os sinais #IRDY e #TRDY são acionados juntos, a fase atual se torna completa e finalizada, ocorrendo à transferência de dados entre iniciador e o alvo como mostrados nos clocks 4, 6 e 8. Note que no clock 5, o alvo não está pronto e não coloca os dados nos sinais AD[31:0]. O caso é diferente no clock 7 onde o iniciador não está pronto, e o alvo sim, cujos dados já estão prontos nos sinais AD[31:0].

A duração da transação dá-se quando o sinal #FRAME permanece ativo a cada fase completa, requisitando mais dados para transferência. A finalização ocorre no inicio do clock 8, onde o sinal #FRAME é desativado junto com ativação do sinal #IRDY indicando ao alvo que a fase atual é a última fase requisitada.

Este tipo de transação tem a forma de rajadas de dados (burst), pois a transação é dividida em 3 fases de dados.

3.13 Transação de Escrita

A transação de escrita é mostrada na figura 3.14. O iniciador aciona o sinal #FRAME, indicando o inicio da transação e, simultaneamente, coloca o endereço do dispositivo desejado

e o tipo de comando de transação nos respectivos sinais AD[31:0] e #CBE[3:0], assim a fase de endereço é originada, como mostrado no clock 2. Esta ilustração serve para qualquer tipo de escrita [Anderson 1999] [PCI 1998].

Note que não existe o clock extra, como na transação de leitura. Isto porque os dispositivos não precisam inverter o barramento, pois, desde a fase de endereço, até o final da transação de escrita, os sinais AD[31:0] dos dispositivos permanecem como entrada de dados. Porém existe um clock extra em conseqüência de um estado de espera, devido ao fato de o dispositivo ser classificado como tipo médio, ver secção 3.11.1.

Na fase de dados, os sinais #CBE[3:0] indicam quais dos bytes são válidos para serem escritos nos sinais AD[31:0].

Em qualquer momento da transação podem ocorrer estados de espera, tanto pela parte do iniciador, via desativação do sinal #IRDY, como pela parte do alvo, via desativação do sinal #TRDY, mostrados nos clocks 6, 7 e 8. Quando os sinais #IRDY e #TRDY são ativados, simultaneamente, a fase de dados atual torna-se completa com transferência de dados, como mostra os clocks 4, 5 e 9. Note que nos clocks 6, 7 e 8, o alvo não está pronto para transferência e o iniciador está pronto nos clock 7 e 8, cujos dados já estão prontos nos sinais AD[31:0]. O caso é diferente no clock 6 onde o iniciador não atualizou os sinais AD[31:0] para transferência de dados.

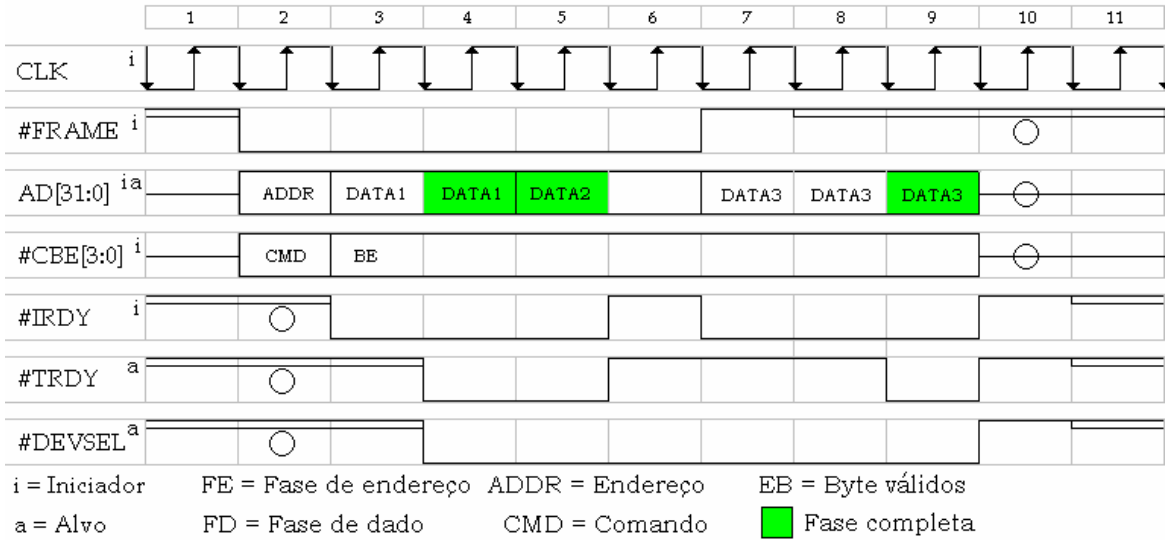


Figura 3.14: Transação de escrita

A duração da transação dá-se quando o sinal #FRAME permanece ativo a cada fase completa requisitando mais dados para transferência. A finalização ocorre no inicio de clock 7, onde o sinal #FRAME é desativado junto com ativação do sinal #IRDY indicando ao alvo que a fase atual é a última fase requisitada.

Este tipo de transação tem, também, forma de rajadas de dados (burst), pois a transação é dividida em 3 fases de dados.

3.14 Interrupção e o Reconhecimento de Interrupção

Em uma arquitetura computacional existem três modos de chamada de interrupção pelo dispositivo PCI [Anderson 1999] [PCI 1998]:

- Chamada via acionamento do pino de interrupção direta do processador;
- Chamada via acionamento do pino de interrupção através de um controlador de interrupção;
- Chamada via transação de escrita em memória, também chamada de MSI.

Os dois primeiros modos serão aqui abordados. O tratamento e o reconhecimento da interrupção de ambos são o mesmo, com diferença, somente, na arquitetura do *hardware.* Pela revisão 2.2, o modo MSI não utiliza pinos de interrupção, mas sim, uma transação de escrita. Entretanto, recomenda-se utilizar os dois mecanismos na construção de um novo *hardware*.

Quando um dispositivo necessita da atenção do *driver*, o mesmo aciona um dos sinais de interrupção e permanece até que a interrupção seja atendida [PCI 1998]. Se o dispositivo for simples, sempre deve utilizar o sinal #INT[A]. Caso os dispositivos sejam de multifunções, podem distribuir e compartilhar entre as funções, os sinais #INT[A], #INT[B], #INT[C] ou #INT[D]. Seguindo sempre nessa ordem, não se deve usar #INT[B] e #INT[C], ao passo que não fora usado o #INT[A]. Um dispositivo de quatro funções pode distribuir e compartilhar os sinais de interrupção da seguinte maneira:

- Todas as funções com o sinal #INT[A];
- Duas funções com o sinal #INT[A], e duas funções com o sinal #INT[B];
- Cada função utiliza cada pino individual. Por exemplo: função 1 utiliza o sinal #INT[A], função 2 utiliza #INT[B] e assim sucessivamente;
- Três funções com o sinal #INT[A] e uma função com o sinal #INT[B], etc...

Após a configuração dos dispositivos no momento do *boot* (ver secção 3.24). Os dispositivos que necessitarem de interrupção serão configurados. A tabela 3.4 mostra os códigos de interrupções, em que o iniciador deve chamar um processador de uma arquitetura PC-AT.

IRQ	**Linha Int**	**Int CPU**
0	0h	08h
1	1h	09h
2	2h	0Ah
3	3h	0Bh
4	4h	0Ch
5	5h	0Dh
6	6h	0Eh
7	7h	0Fh
8	8h	70h
9	9h	71h
10	Ah	72h
11	Bh	73h
12	Ch	74h
13	Dh	75h
14	Eh	76h
15	Fh	77h
Nenhum	FFh	

Tabela 3.4: Códigos de interrupções

A transação de reconhecimento de interrupção não suporta forma de rajada de dados. Somente os sinais #CBE[3:0] serão ativados, e os sinais AD[31:0] serão aleatórios na fase de endereço. Assim, somente o iniciador necessita pegar o IRQ do alvo devido que a BIOS já ter um mapa de todos os dispositivos com suas linhas de interrupção, feitas no momento do *boot* do computador. No inicio da transação de reconhecimento. O dispositivo que gerou a interrupção já sinaliza ao iniciador que o mesmo é o alvo da transação, ativando o seu sinal #DESVEL, sem a necessidade de comparações de endereços.

A figura 3.15 mostra um reconhecimento de uma interrupção e o acionamento da interrupção do processador de uma arquitetura PC-AT.

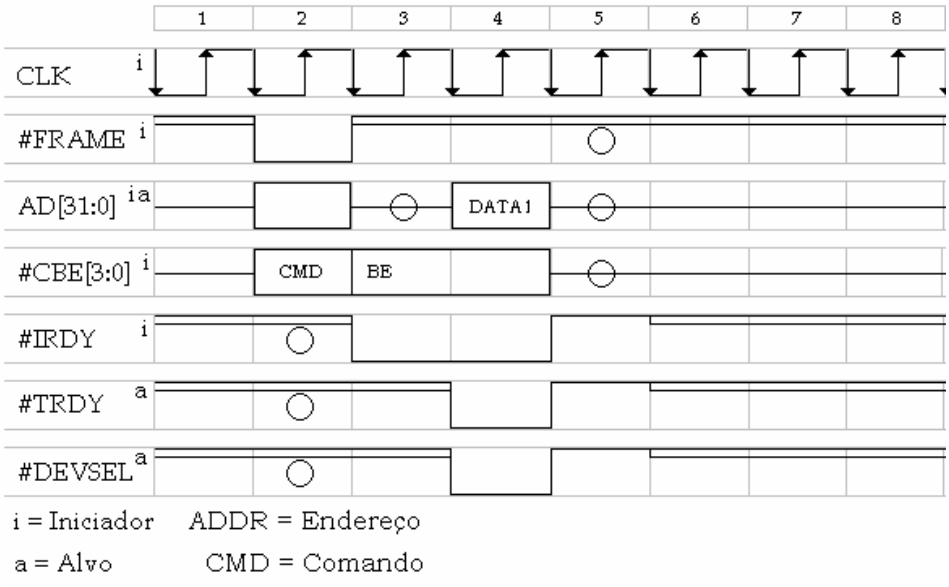


Figura 3.15: Reconhecimento de interrupção

3.15 Fim de uma Transação

A última transferência de dados em uma transação é indicada pela desativação do sinal #FRAME, na fase atual e após a fase de dados atual for completa. Mas podem existir circunstâncias em que possa ocorrer uma terminação precoce da transação, esta terminação tanto pode ser da parte do iniciador como pela parte do alvo [Anderson 1999] [PCI 1998].

3.15.1 Terminação da Transação pelo Alvo

Existem três tipos de terminação da transação do alvo. Cada uma é referente a certas circunstâncias que o alvo não suporta. Estas terminações são: tente novamente (retry), desconecte com transferência de dados (Disconect), desconecte sem transferência de dados (Disconect) e aborte (Abort). Todos estes tipos de terminação usam o sinal #STOP em combinação com os sinais #TRDY e #DEVSEL [Abbott 2000].

3.15.1.1 Tente Novamente (Retry)

A terminação, Tente Novamente, é gerada ao acionar o sinal #STOP e desativando o sinal #TRDY no início da fase de dados. Este procedimento termina a transação antes da transferência de dados, pois o alvo está ocupado ou temporariamente incapaz de efetuar a transação. Quando um alvo terminar com retry o iniciador deve, incondicionalmente, repetir a

mesma transação, até que seja completa. Isto não acontece com os outros tipos de terminações.

A figura 3.16 mostra uma tentativa do iniciador de escrever no alvo. Entretanto o alvo finaliza a transação com retry, por não estar pronto para a mesma.

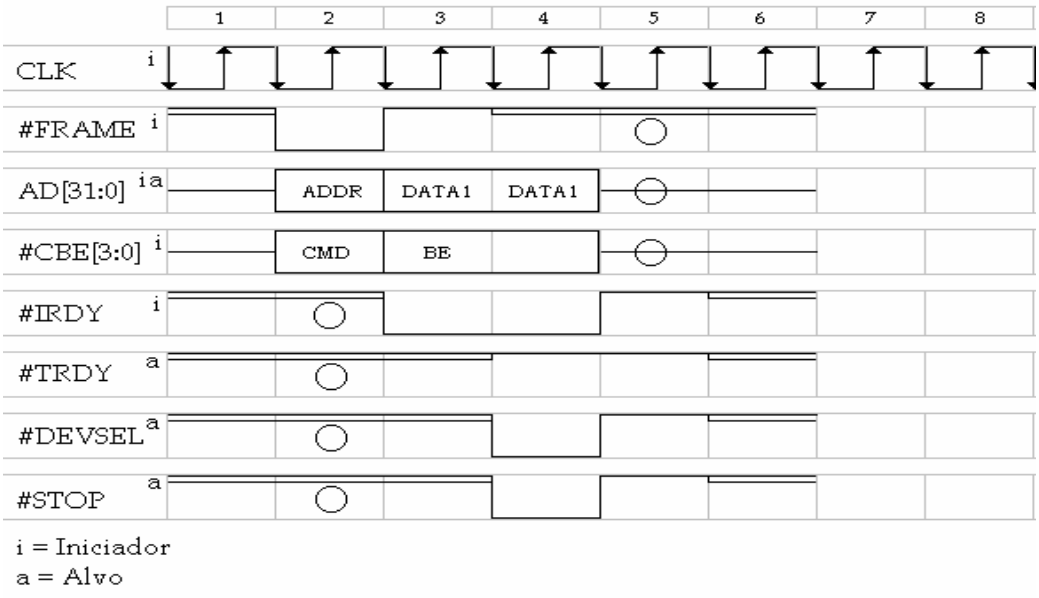


Figura 3.16: Terminação por tente novamente (retry) da transação PCI

3.15.1.2 Desconecte (Disconnect)

A terminação, Desconecte com transferência de dados, é gerada ao acionar os sinais #STOP e #TRDY juntos. Este procedimento termina a transação por três motivos: o alvo é incapaz de responder dentro do tempo de latência requerido, ou o alvo não suporta uma transferência em rajada de dados (burst), ou o alvo, temporariamente, não suporta uma transferência em rajada de dados (burst).

A terminação, Desconectando sem transferência de dados, é idêntica ao retry. A diferença é que este método é usado após a primeira fase de dados ocasionando uma transferência na primeira fase.

Na figura 3.17, o iniciador pretende ler em forma de rajada nos registradores de configuração. O alvo desconecta com transferência de dados, por o mesmo não suportar a transferência em modo de rajada. Note que no clock 4, o sinal #FRAME, ainda está ativo, requisitando mais dados (burst).

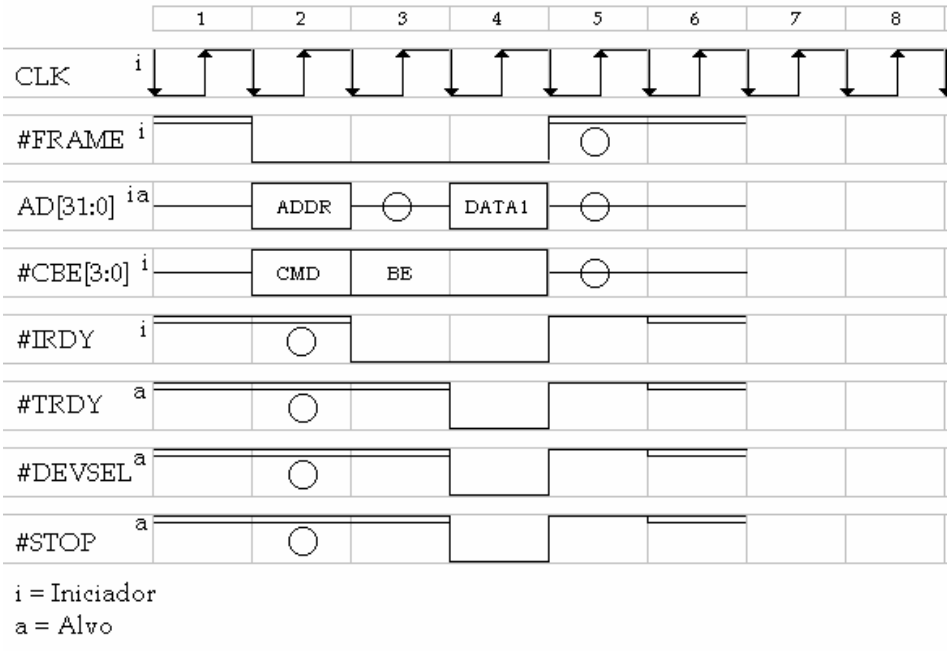


Figura 3.17: Terminação por desconexão (disconnect) da transação PCI

3.15.1.3 Aborte (Abort)

Aborto sem transferência de dados é feito gerado ao acionar o sinal #STOP e ao desativar o sinal #DESVEL, simultaneamente. Este procedimento termina quando uma transação anormal ocorre originando um erro fatal. Ocorre também quando o alvo não é capaz de completar o pedido da transação. Isso ocorre, caso o iniciador deseje ler uma *dword* no alvo em um registrador em byte.

Na figura 3.18 o iniciador deseja escrever um *word* em uma memória alinhada em bytes.

3.15.2 Terminação da Transação pelo Iniciador

Existem duas circunstâncias em que o iniciador termina a transação de forma forçada: a terminação por preempção e a terminação por aborto.

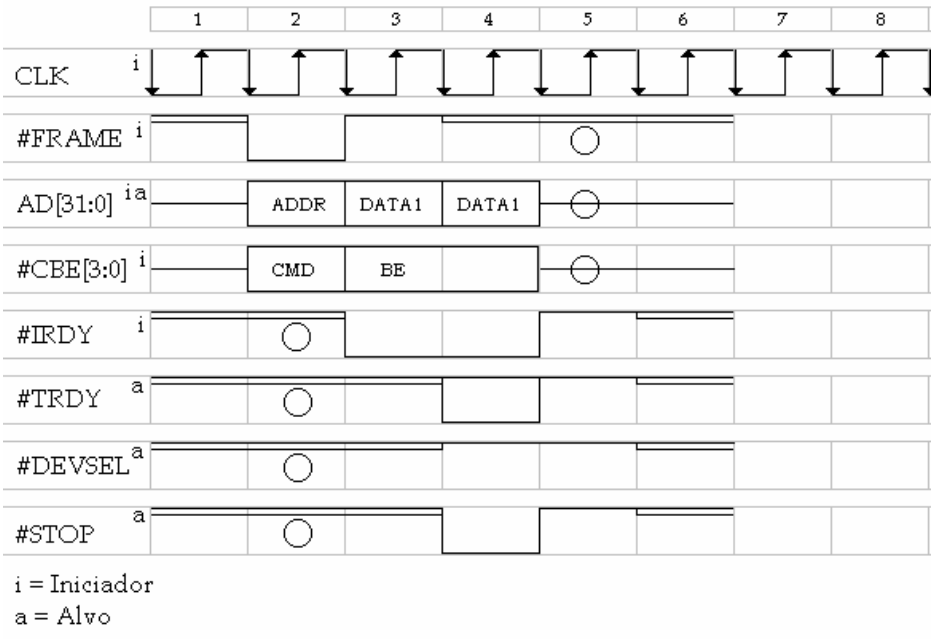


Figura 3.18: Terminação por aborto (abort) da transação PCI

3.15.2.1 Terminação por Preempção

A terminação por preempção ocorre quando o tempo de latência do iniciador atual já expirou, e exista um outro dispositivo PCI que deseje usar o barramento para transferência. Então o iniciador atual finaliza a transação e continua mais tarde, dando a vez para outro dispositivo.

3.15.2.2 Terminação por Aborto

A terminação por aborto ocorre quando nenhum dispositivo responde ao pedido de transação após 4 clocks do barramento. Este tipo de terminação normalmente representa um erro sério de condição.

3.16 Geração de Paridade

A geração de paridade garante a transferência de dados sem erros entre os dispositivos, caso haja, em uma transação lógica, uma troca do nível de tensão em qualquer um dos bits por motivos elétricos desconhecidos. O nível lógico 0 pode tornar-se nível lógico 1 por interferências. Assim, a função da paridade é identificar a ocorrência do erro e agir de acordo como programado. A geração de paridade não é um item opcional, e é sempre gerado por todos os dispositivos PCI [PCI 1998].

O cálculo da paridade é feito em qualquer tipo de transação. No cálculo sempre são incluídos os sinais AD[31:0] e #CBE[3:0] independente se existem ou não dados válidos. O cálculo da paridade permite identificar se o número de ocorrências de bits ligados nos sinais AD[31:0] e #CBE[3:0] estão em pares. Então, quando o resultado do cálculo for igual a 0, indica que a ocorrência foi em par, e caso o resultado seja 1, indica que a ocorrência foi impar. A lógica de cálculo é feita através do operador lógico ou exclusivo (XOR) em seqüência com os sinais, conforme a figura 3.19 [PCI 1998].

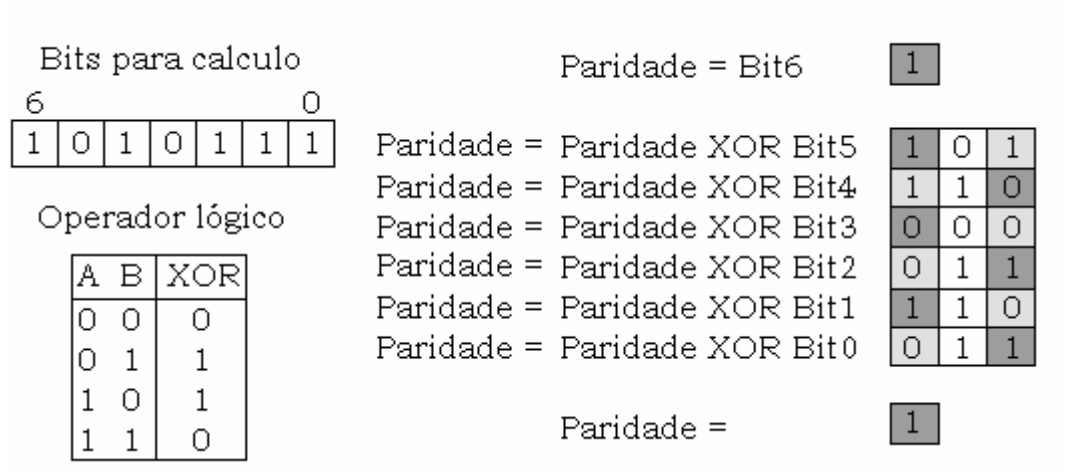


Figura 3.19: Exemplo de cálculo da paridade

O sinal PAR do dispositivo alvo lê o cálculo da paridade do iniciador na transação de escrita e ou fornece o cálculo da paridade na transação de leitura. O sentido do sinal PAR depende do tipo da transação, e é atrasado 1 clock permanecendo válido 1 clock após a fase completa.

Na figura 3.20, pela transação de leitura, o iniciador atualiza o sinal PAR no clock 3 com o cálculo de paridade gerado dos sinais AD[31:0] e #CBE[3:0] na fase de endereço no clock 2. Ainda nesta transação o sinal PAR é invertido no clock 4, para que o alvo atualize o sinal PAR no clock 5 com o cálculo da paridade dos sinais AD[31:0] e #CBE[3:0] no clock 4.

A figura 3.21 mostra uma transação de escrita onde o iniciador atualiza o sinal PAR nos clocks 9 e 11, com o cálculo de paridade dos sinais AD[31:0] e #CBE[3:0] nos clocks 8 e 10.

3.17 Checagem de Paridade

A checagem da paridade é feita por todos os dispositivos PCI's com exceções dos [PCI 1998]:

- Dispositivos exclusivos do computador como *chipsets*, não são dispositivos adicionais conectados ao *slots,* devido ao aumento do risco de erros pelas más conexões;
- Dispositivos com dados temporários e não críticos aos erros, como placa de vídeo ou de som. Caso exista algum erro, o sistema continua funcionando sem problemas. Os nossos sentidos dificilmente detectariam.

No barramento PCI existem dois tipos de checagem de erros de paridade: checagem de erro de paridade de sistema sinalizado pelo sinal #SERR, e a checagem de erro paridade de dados sinalizado pelo sinal #PERR. Ambos os sinais são atrasados dois clocks a partir da fase completa. No primeiro clock o sinal PAR é calculado e válido, no segundo clock o alvo calcula a paridade, compara o resultado com o sinal PAR do iniciador e atualiza o sinal #PERR ou #SERR. Assim, o mantém ativo por 1 clock [Abbott 2000].

3.17.1 Checagem de Paridade em Erro em Sistema

O sinal #SERR é usado para evidenciar erros de paridade dos sinais AD[31:0] e #CBE[3:0]. Isso na fase de endereço e em transações que não se referenciam a fase de dados, com exceção nas fases de dados na transação do comando especial. Neste sinal, outros dispositivos podem acionar, simultaneamente, pela lógica OU (OR) entre dispositivos.

Se o dispositivo for alvo da transação, e se existir um erro de paridade na fase de endereço, o alvo pode atuar em seguintes procedimentos:

- Aceitar a transação e ignorar o erro;
- Aceitar a transação e logo em seguida abortar se o bit 6 do registrador de configuração de comando [1] estiver acionado. Somente acionar o sinal #SERR se o bit 8 do registrador de configuração de comando estiver acionado [2];
- Não aceitar a transação pela não ativação do sinal #DEVSEL e somente acionar o sinal #SERR se o bit 8 do registrador de configuração de comando estiver acionado [2]. Com esse procedimento, o iniciador aborta a transação.

Independente dos bits 6 e 8 do registrador de comando [1-2], o bit 14 do registrador de status [3] deve ser acionado se existir um erro de paridade, e mantido até que o *driver* do alvo o limpe, ou o computador entre em *reset*.

A figura 3.20 mostra uma transação de leitura e outra de escrita, em que nos clocks 3 e 9, o iniciador fornece os cálculos da paridade via sinal PAR dos sinais AD[31:0] e #CBE[3:0] da fase de endereço. E nos clocks 4 e 10, o alvo compara o sinal PAR com seu cálculo de paridade dos sinais AD[31:0] e #CBE[3:0] da fase de endereço. Caso exista um erro de paridade, o alvo aciona o sinal #SERR e o *chipset* age para tratar o erro.

3.17.2 Checagem de Paridade em Erro em Dados

A checagem de paridade em erro em dados, como o próprio nome já diz, se faz nas fases de dados nas transações de escrita e leitura. O sinal #PERR é usado nas fases de dados para sinalizar erro de paridade dos sinais AD[31:0] e #CBE[3:0], com exceção do ciclo de transação especial. O sinal #PERR, no iniciador, assume os estados, tanto de entrada, como de saída, dependendo do sentido da transação. O sentido do sinal do alvo é somente de saída, no alvo, somente checa-se a paridade na transação de escrita. O alvo opcionalmente pode ou não tratar o erro de paridade de dados, dependendo da configuração estabelecida no bit 6 do registrador de configuração de comando [1]. Diferente do sinal #SERR o sinal #PERR não é compartilhado com outros dispositivos, somente um único dispositivo por vez pode acioná-lo.

Em uma transação de leitura, se o iniciador detectar algum erro de paridade, e se o bit 6 do registrador de configuração de comando [1] estiver acionado, o iniciador deve sinalizar o erro de paridade via sinal #PERR. Esta ação é somente entre iniciadores e pontes, para que seja tomada ação condizente ao erro. Já no alvo o sinal #PERR não tem efeito, pois é o *driver* do dispositivo que trata o erro de paridade.

Em uma transação de escrita, se o bit 6 do registrador de configuração de comando [1] estiver acionado e o erro de paridade for detectado pelo alvo, o mesmo deve acionar o sinal #PERR dois clocks após a fase de dados estar completa a partir de onde ocorrer o erro.

Independente do bit 6 registrador de comando [1] o bit 15 do registrador de status [4] é acionado se existir um erro de paridade, e é mantido até que o *driver* do alvo o limpe, ou o computador entre em *reset*.

Na figura 3.20, pela transação de leitura no clock 5, o alvo fornece os cálculos da paridade, via sinal PAR dos sinais AD[31:0] e #CBE[3:0] da fase de dados do clock 4. No clock 6 o iniciador compara o sinal PAR com seu cálculo de paridade dos sinais AD[31:0] e #CBE[3:0] da fase de dados do clock 4. Se o iniciador detectar um erro de paridade, o mesmo manda uma mensagem ao *driver* do dispositivo para agir como foi programado.

A figura 3.20 mostra uma transação de escrita, e no clock 11, o alvo fornece o cálculo da paridade, via sinal PAR dos sinais AD[31:0] e #CBE[3:0] da fase de dados do clock 10. Se existir algum erro de paridade, o alvo aciona o bit 15 do registrador de status [5] e aciona o sinal #PERR no clock 12, se estiver habilitado, e age de acordo com a programação do *driver* do alvo.

3.17.3 Tratamento de Erro

Somente o iniciador tem a responsabilidade de reportar o erro de paridade ao sistema operacional, via *driver* do dispositivo, para um tratamento do erro.

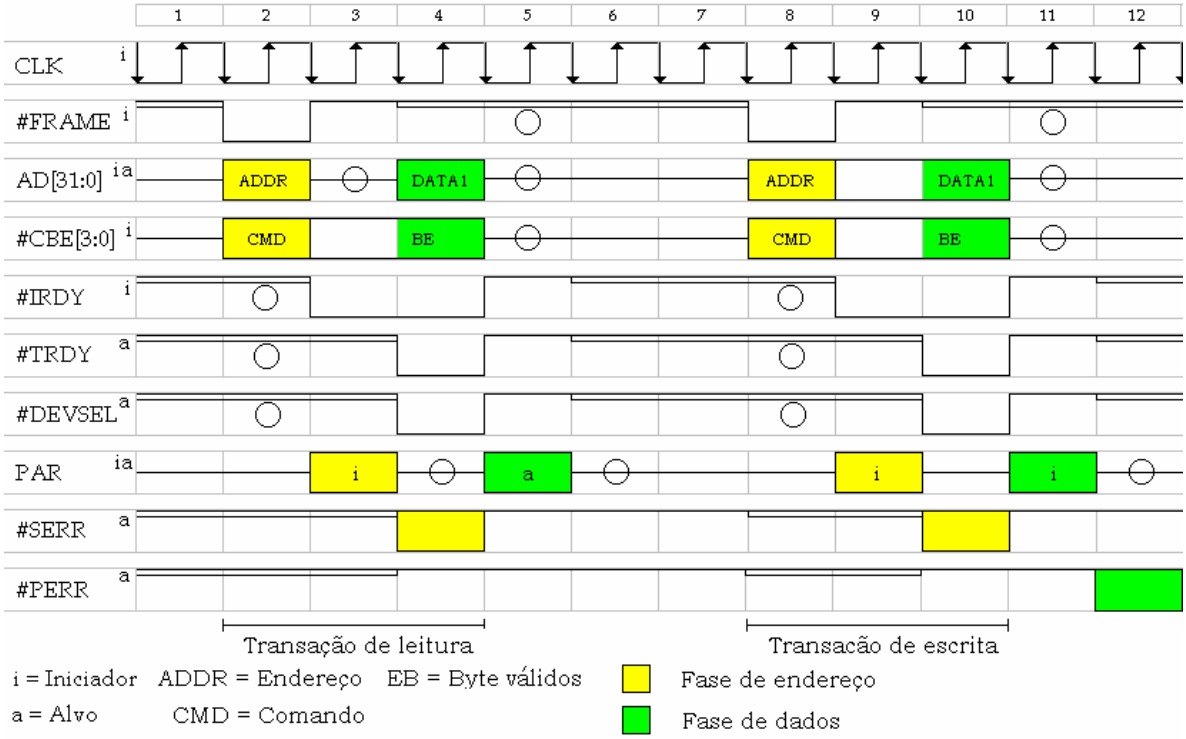


Figura 3.20: Checagem de erro de paridade

1 - Bit Parity Error Response
2 - Bit #SERR enable
3 - Bit Signaled System error
4 - Bit Detected Parity Error

3.18 Tempo de Latência

O tempo de latência é o tempo de estado de espera, tanto pelo iniciador, como pelo alvo em uma transação. O Iniciador deve ativar #IRDY até 8 clocks após o acionamento do sinal #FRAME na primeira fase de dados e com 8 clocks na desativação do sinal #IRDY nas subseqüentes fases de dados. O alvo deve acionar #TRDY até 16 clocks após o acionamento do sinal #FRAME, na primeira fase de dados e com 8 clocks para completar a fase de dados atuais [Anderson 1999] [PCI 1998].

3.19 Transação Rápida (Fast Back to Back)

A transação rápida é a eliminação do ciclo de inversão do barramento. O objetivo é aumentar a performance do barramento, que somente acontece em certas circunstancias, cuja contenção do barramento não existe [Abbott 2000].

3.20 Resposta a uma Violação do Protocolo

Se existir uma violação no barramento PCI, por parte do iniciador, o dispositivo alvo garantirá a consistência de dados e, imediatamente, entrará em um estado conhecido. Pois o controle de estados do dispositivo é feito de acordo com as regras PCI de funcionamento normais e se não prever nenhuma violação o mesmo ficará travado comprometendo seriamente o funcionamento do sistema. Apenas um *reset* restabelecerá o funcionamento normal. Por exemplo: O sinal #FRAME nunca pode ser desativado antes da ativação do sinal #IRDY. Se isso acontecer, além de ser um erro de condição, o alvo interpreta que o iniciador está pedindo o final de uma transação. Então, mesmo com a fase de dados não completa, o alvo finalizará sua transação e os demais dados não transferidos serão perdidos. Todavia, estes tipos de tratamento de violações não são encorajados pela *PCI SIG* a serem implementados [Anderson 1999].

3.21 Exemplos de Transações

Nestes subcapítulos serão exemplificadas algumas das principais transações do barramento PCI com comentários clock a clock.

3.21.1 Transação de Leitura nos Registradores de Configuração

As transações de leitura nos registradores de configuração são feitas em acessos em *dwords* pelo iniciador exemplificado na figura 3.21. Essa figura mostra dois tipos de transação, uma sem estado de espera e outra com estado de espera. Estes tipos de acessos podem ser feitos, tanto pela BIOS no *boot* do sistema, como pelo *driver* do dispositivo em qualquer momento.

Clock 2 – O iniciador encarrega-se de colocar no barramento, o endereço do registrador nos sinais AD[31:0]. Encarrega-se de ativar o sinal IDSEL do dispositivo desejado o tornando alvo da transação. Encarrega-se de colocar o comando de leitura do registrador nos sinais #CBE[3:0] e ativa o sinal #FRAME para indicar inicio da transação.

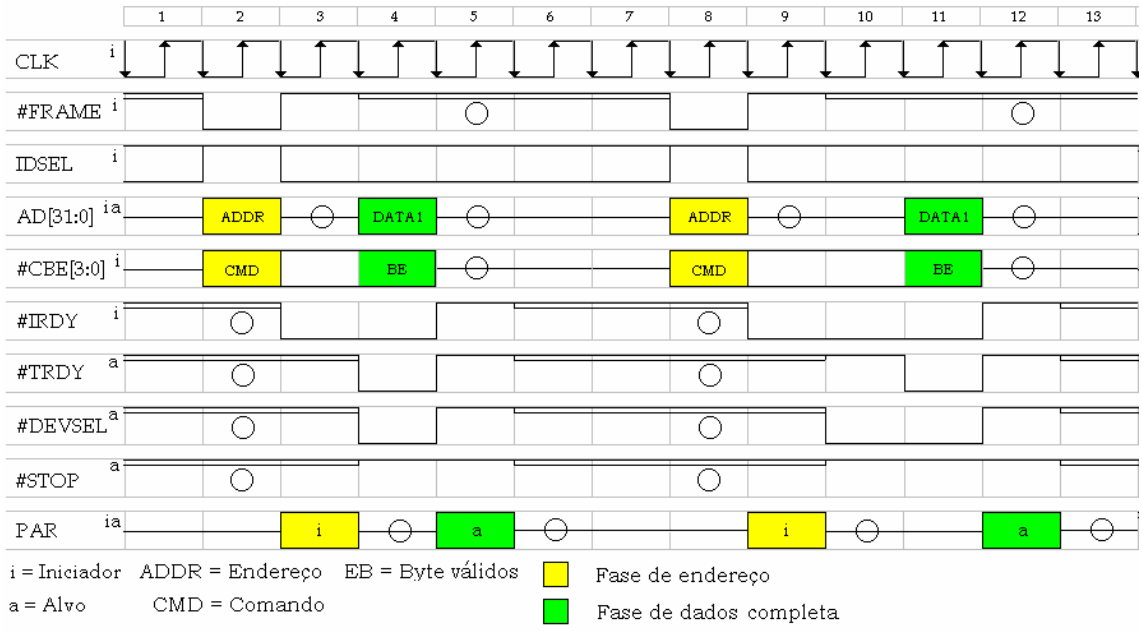


Figura 3.21: Transação de leitura nos registradores de configuração

Clock 3 – O dispositivo alvo inverte os seus sinais AD[31:0] de entrada para saída. O iniciador sinaliza que está pronto para receber dados e qual dos bytes são válidos nos sinais AD[31:0] para transação. Isto é feito pelos respectivos sinais #IRDY e #CBE[3:0]. O iniciador desativa o sinal #FRAME sinalizando que a transação não é em forma de rajada de dados e atualiza o sinal PAR com o cálculo da paridade da fase de endereço. Neste clock

existe um estado de espera, assim este tipo de alvo é classificado como do tipo médio, ver secção 3.11.1.

Clock 4 – O alvo sinaliza ao iniciador, que o mesmo é o alvo da transação via acionamento do sinal #DEVSEL, e atualiza os sinais AD[31:0] com os dados do registrador apontado pelo endereço. Também sinaliza ao iniciador, que os dados estão prontos para transferência através do sinal #TRDY. Neste clock a fase de dados é completa a transação é finalizada.

Clock 5 – O alvo atualiza o sinal PAR, com o cálculo de paridade dos sinais AD[31:0] e #CBE[3:0] da fase de dados do clock 4.

Clocks de 8 a 12 – Segue o mesmo raciocínio dos clocks 2 ao 5. Com exceção no clock 10, onde o barramento entra no estado de espera, visto que o alvo não está pronto. Neste caso a fase de dados não está completa.

3.21.2 Transação de Escrita nos Registradores de Configuração

As transações de escrita nos registradores de configuração são feitas em acessos em *dwords* pelo iniciador exemplificada na figura 3.22. Essa figura mostra dois tipos de transações, uma sem estado de espera e outra com estado de espera. Estes tipos de acessos podem ser feitos, tanto pela BIOS no *boot* do sistema, como pelo *driver* do dispositivo em qualquer momento.

Clock 2 – O iniciador encarrega-se de colocar no barramento, o endereço do registrador nos sinais AD[31:0], ativar o sinal IDSEL do dispositivo desejado, colocar o comando de escrita do registrador nos sinais #CBE[3:0] e ativar o sinal #FRAME para indicar inicio da transação.

Clock 3 – O iniciador sinaliza, que os dados estão prontos nos sinais AD[31:0] e quais os bytes são válidos para transferência pelos respectivos sinais #IRDY e #CBE[3:0]. Neste clock o iniciador desativa o sinal #FRAME indicando que a transação não é em forma de rajada de dados e atualiza o sinal PAR com o cálculo da paridade da fase de endereço. Ainda neste clock existe um estado de espera, devido que o alvo é classificado como do tipo médio, ver secção 3.11.1.

Clock 4 – O alvo sinaliza ao iniciador que o mesmo é o alvo da transação e está pronto para receber dados pelos respectivos sinais #DEVSEL e #TRDY. No final deste clock a fase de dados é completa e a transação é finalizada. Ainda neste clock, o iniciador atualiza o sinal PAR com o cálculo de paridade dos sinais AD[31:0] e #CBE[3:0] do clock 3 e permanece até o clock 5.

Clocks de 8 a 12 – Segue o mesmo raciocínio dos clocks 2 ao 5. Porém aqui, o barramento entra no estado de espera pela não ativação dos sinais #TRDY e #IRDY no clock 10. Isso indica, que nem o alvo e nem o iniciador estão prontos para transferência de dados. A transferência de dados somente ocorre no clock 11 com a fase completa.

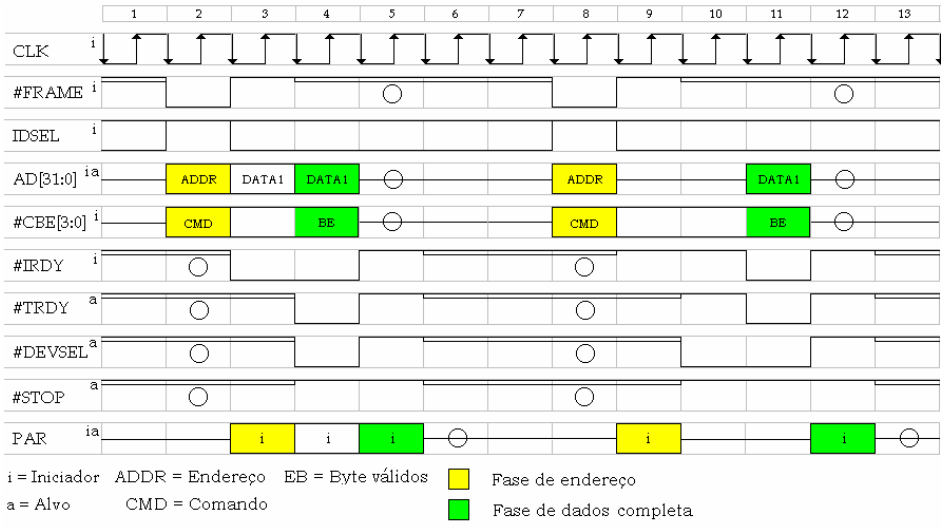


Figura 3.22: Transação de escrita nos registradores de configuração

3.21.3 Transação de Leitura em Memória ou em I/O de Modo Simples

A transação de leitura em espaçamento de endereçamento de memória, ou de I/O, ocorre quando o software de alto nível deseja ler o dispositivo PCI através de seu *driver*. Como mostrado na figura 3.23. Esta transação é idêntica à transação de leitura no registrador de configuração, exceto nos clocks 2 e 8, onde o iniciador não aciona o sinal IDSEL. Entretanto, o iniciador se encarrega de colocar no barramento, o endereço do dispositivo desejado. Neste clock, todos os dispositivos devem capturar o endereço. Aquele dispositivo que coincidir com o seu endereço, será o alvo da transação. O estado de espera no clock 10 pela parte do iniciador.

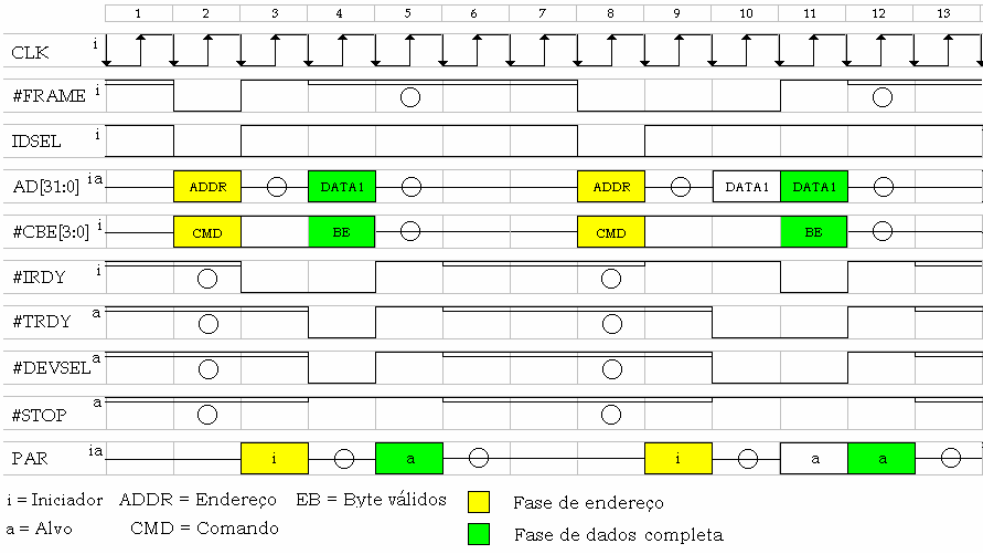


Figura 3.23: Transação de leitura em memória ou em I/O de modo simples

3.21.4 Transação de Escrita em Memória ou em I/O de Modo Simples

A transação de escrita em espaçamento de endereçamento de memória, ou de I/O ocorre quando o software de alto nível deseja escrever no dispositivo PCI através de seu *driver*. Como mostrado na figura 3.24. Esta transação é idêntica à transação de escrita no registrador de configuração, exceto nos clocks 2 e 7, onde o iniciador não aciona o sinal IDSEL. Entretanto, o iniciador se encarrega de colocar no barramento, o endereço do dispositivo desejado. Neste clock, todos os dispositivos devem capturar o endereço. Aquele dispositivo que coincidir com o seu endereço, será o alvo da transação. Os estados de espera no clock 9 pela parte do alvo e nos clocks 9 e 10 pela parte do iniciador.

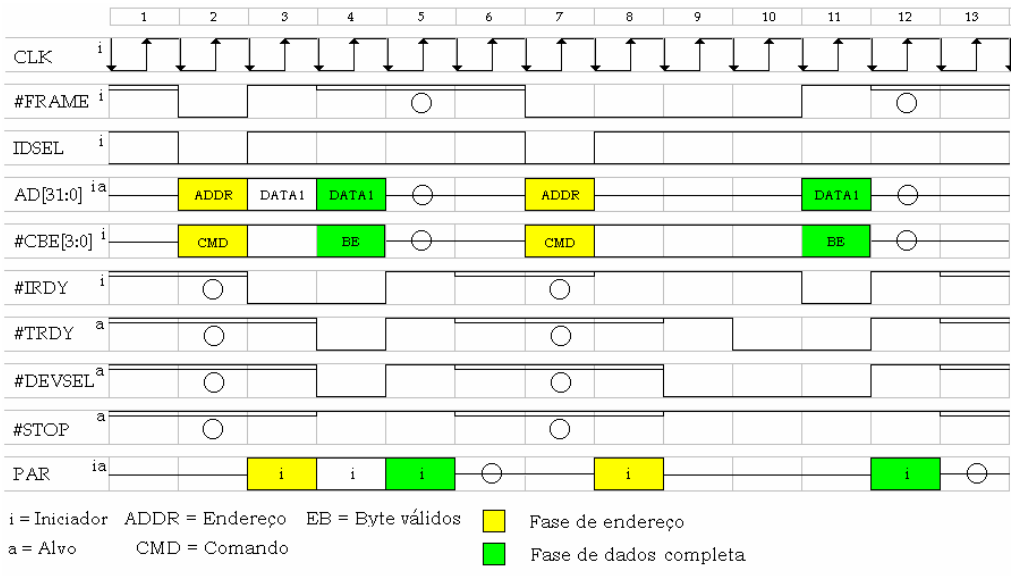


Figura 3.24: Transação de escrita em memória ou em I/O de modo simples

3.21.5 Transação de Leitura em Memória ou em I/O em Rajadas

A transação de leitura em rajada em espaçamento de endereçamento de memória, ou de I/O, ocorre quando o software de alto nível deseja ler um banco de registradores ou de memória no dispositivo PCI através de seu *driver* em uma única transação. Como mostrado na figura 3.25. A transação é idêntica à transação de leitura simples em espaçamento de endereçamento memória ou I/O. Nesse caso, a única exceção é que o sinal #FRAME permanece ativo em cada fase de dados requisitando mais dados para transferência. No clock 11, o sinal #FRAME é desativado indicando a última fase de dados para transferência. Os estados de espera pela parte do alvo nos clocks 4 e 9 e pela parte do iniciador no clock 7.

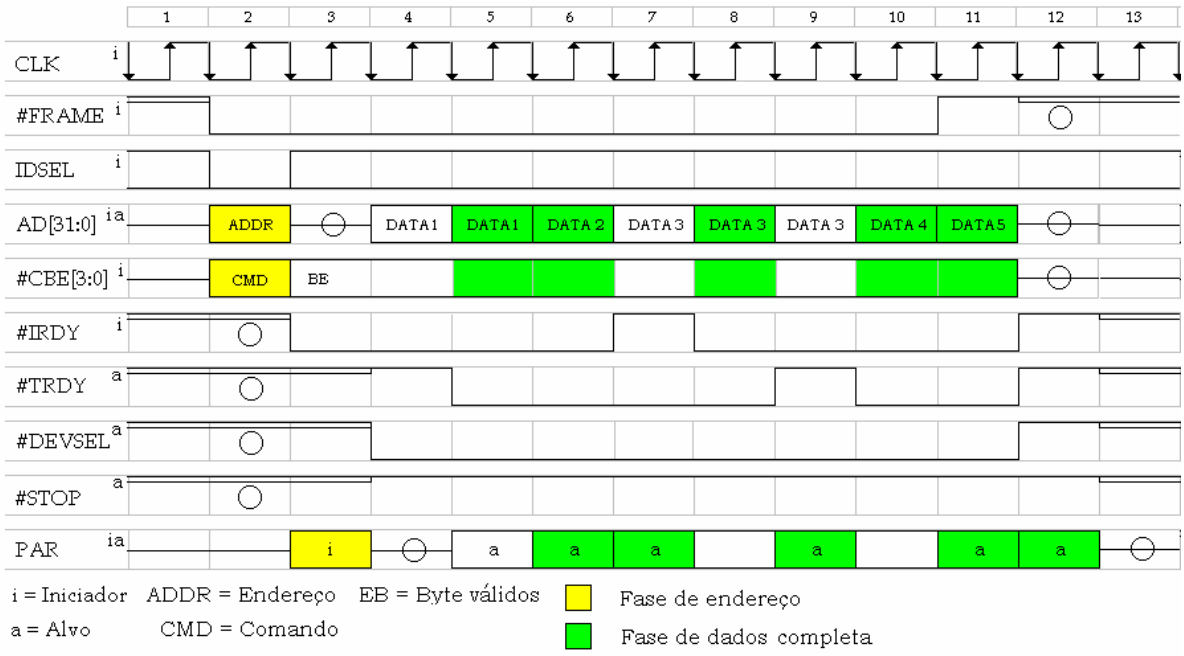


Figura 3.25: Transação de leitura em memória ou em I/O em rajadas

3.21.6 Transação de Escrita em Memória ou I/O em Rajadas

A transação de escrita em rajada em espaçamento de endereçamento de memória, ou de I/O, ocorre quando o software de alto nível deseja escrever em um banco de registradores, ou de memória no dispositivo PCI através de seu *driver* em uma única transação. Como mostrado na figura 3.26. A transação é idêntica á da transação de escrita simples em espaçamento de endereçamento memória, ou I/O. Nesse caso, a única exceção é que o sinal #FRAME permanece ativo em cada fase de dados, requisitando mais dados para transferência. No clock 11, o sinal #FRAME é desativado, indicando a última fase de dados para

transferência. Os estados de espera pela parte do alvo nos clocks 8 e 9 e pela parte do iniciador nos clocks 6 e 8.

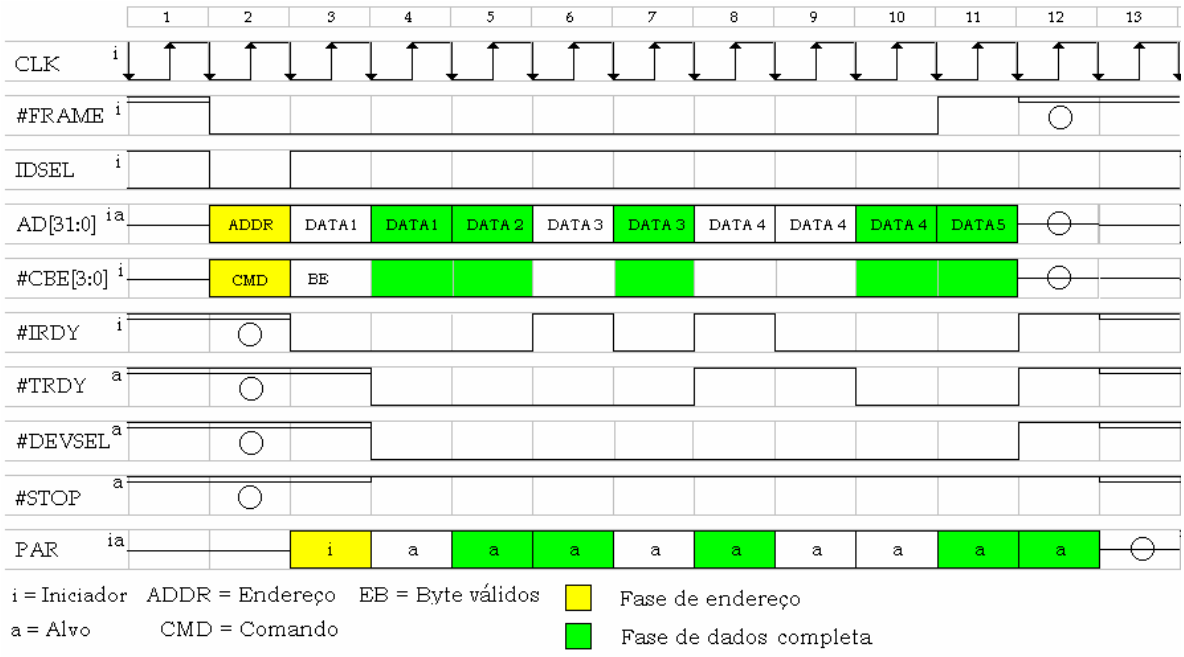


Figura 3.26: Transação de escrita em memória ou I/O em rajadas

3.22 Registradores de Configuração

Para que o barramento PCI funcione adequadamente e torne-se plug and play por natureza, cada dispositivo PCI [5] contém um o banco de 256 bytes de registradores para configuração. Estes registradores foram definidos pela *PCI SIG*. Os acessos, em *dwords*, aos registradores são feitos pelos comandos de leitura e escrita em registradores de configuração, tanto pela BIOS no *boot* do computador, como em qualquer momento pelo *driver* do dispositivo. Na figura 3.27 se encontram as disposições dos registradores de configuração e seus respectivos endereços. Os registradores aqui descritos referem-se aos dispositivos comuns e não para pontes e iniciadores, fato que estes dois últimos dispositivos necessitam de mais registradores de controle. Os seis primeiros registradores são obrigatórios para todos dispositivos PCI's e os demais são opcionais e de acordo de cada aplicação. Cabe lembrar que existem outros tipos de barramentos que são plug and play além do barramento PCI [PCI 1998].

1 - Incluído os dispositivos de multifunções

3.22.1 Registrador de Identificação do Fornecedor

O registrador de identificação do fornecedor contém o número de identificação do fabricante do dispositivo PCI. Esta identificação é distribuída pela *PCI SIG*. É um registrador de identificação exclusiva e individual para reconhecimento e configuração do dispositivo no momento do *boot* do computador. No Core PCI o registrador assume o valor ABCDh de forma aleatória por ser um protótipo e não coincidir com os demais dispositivos.

Nome do registrador : Vendor ID
Endereço de OffSet: 00h – 01h
Valor inicial = ABCDh
Atributo: Somente leitura
Tamanho: 16 bits

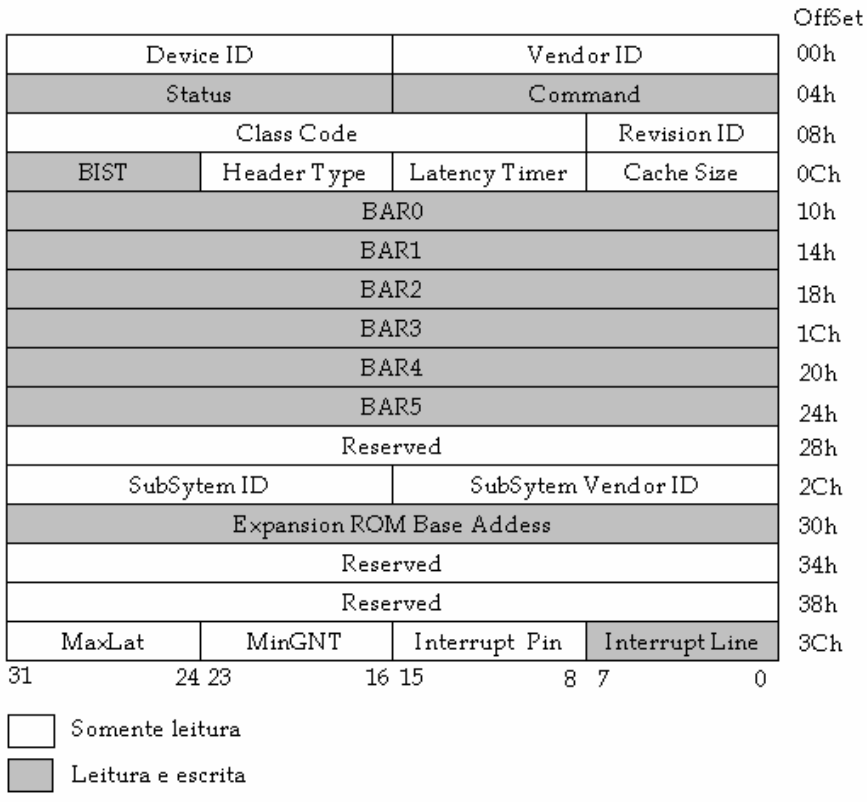


Figura 3.27: Registradores de configuração dos dispositivos PCIs

3.22.2 Registrador de Identificação do Dispositivo

O registrador de identificação do dispositivo é reservado para o número do dispositivo de identificação do fabricante. Cada fabricante é responsável por distribuir um identificador

exclusivo para cada dispositivo. Isso porque o identificador é o complemento para identificação do dispositivo no momento do *boot* do computador. Isto é, age em paridade ao registrador de identificação do fornecedor. No Core PCI o registrador assume o valor 0001h por ser o primeiro protótipo.

Nome do registrador : Device ID
Endereço de OffSet: 02h – 03h
Valor inicial = 0001h
Atributo: Somente leitura
Tamanho: 16 bits

3.22.3 Registrador de Comando

O registrador de comando fornece os comandos básicos para transações no barramento PCI. Quando escrito 0 no registrador, o dispositivo é, logicamente, desconectado do barramento. Com exceção dos acessos de configuração dos registradores. Cada bit é implementado de acordo com o tipo de aplicação e os recursos a serem usados. Por exemplo: O dispositivo não é implementado no espaço de endereçamento de I/O, então não se implementa o bit 0 para escrita, permanecendo 0 para leitura. No Core PCI os valores lógicos X e 0 da figura 3.28, representam respectivamente, implementado e somente leitura. Para os bits implementados, a BIOS preenche com o valor lógico 1, indicando a habilitação do serviço.

Nome do registrador : Command
Endereço de OffSet: 04h – 05h
Valor inicial = 0000h
Atributo: Leitura e escrita
Tamanho: 16 bits

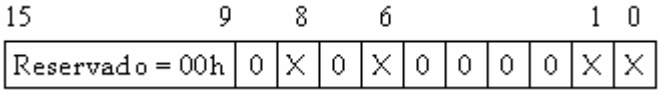


Figura 3.28: Registrador de comando

Bit 0 I/O Space – Controle de acesso ao dispositivo via espaçamento de endereço de I/O. Os valores lógicos 0 e 1 indicam, respectivamente, acesso desabilitado e habilitado. Após *reset* assume o valor lógico 0.

Bit 1 Memory Space - Controle de acesso ao dispositivo via espaçamento de endereço de memória. Os valores lógicos 0 e 1 indicam, respectivamente, acesso desabilitado e habilitado. Após *reset* assume o valor lógico 0.

Bit 2 Bus Master – Possibilita ao dispositivo, agir como mestre no barramento. O valor lógico 1 permite que o dispositivo se comporte como mestre no barramento. Após *reset* assume o valor 0.

Bit 3 Special Cycles – Controla a ação do dispositivo numa transação especial. O valor lógico 0 faz com que o dispositivo ignore a transação especial e o valor lógico 1 permite que o dispositivo monitore a transação especial. Após *reset* assume o valor lógico 0.

Bit 4 Memory Write and Invalidate Enable – Este bit indica ao iniciador, que o dispositivo é capaz de executar a transação com o comando de ESCRITA e INVALIDADE. No valor lógico 1 o iniciador pode gerar os comandos, senão gera somente o comando básico de ESCRITA.

Bit 5 VGA Palette Snoop – Este bit controla a compatibilidade da paleta VGA com dispositivos gráficos. Quando apresentar o valor lógico 1, a paleta de procura é habilitada e quando apresentar o valor lógico 0, o dispositivo deve tratar a paleta com acessos de escrita. Como as demais transações.

Bit 6 Parity Error Response – Este bit controla a resposta de erro de paridade. Quando apresentar o valor lógico 1, o dispositivo deve agir frente ao erro de paridade. E quando apresentar o valor lógico 0, o dispositivo não deve acionar o sinal #PERR. Após o *reset* assume o valor lógico 0. O dispositivo que vai checar o erro de paridade deve implementar este bit.

Bit 7 Stepping Control – Este bit é usado para controlar o endereçamento do dispositivo de modo em andamento. Quando apresentar o valor lógico 0, o dispositivo não fará o endereçamento.

Bit 8 #SERR Enable – Este bit tem a função de habilitar o sinal #SERR quando existir um erro de paridade de sistema e se o bit 6 [1] for ativado. O valor lógico 0 desabilita a ativação do

sinal #SERR, e o valor lógico 1 habilita a ativação do sinal #SERR. Após *reset*, o bit assume valor lógico 0. Todos os dispositivos que tenham o pino #SERR devem ser implementados este bit.

Bit 9 Fast Back to Back Enable – Este bit indica ao iniciador que o dispositivo está apto a executar uma transação de escrita e leitura em acessos rápidos (ver seção 3.20). O valor lógico 1, indica que o dispositivo é capaz, e o valor lógico 0, indica que o dispositivo não suporta acessos rápidos.

Bits 10 a 15 – Reservados.

3.22.4 Registrador de Status

O registrador de status fornece o status do dispositivo PCI no barramento. Quando o bit estiver acionado indicado à ocorrência de algum evento, somente uma escrita com valor lógico 1 o desativará. No Core PCI, os valores lógicos X e 0, mostrados na figura 3.29, representam respectivamente implementado e somente leitura.

Nome do registrador: Status
Endereço de OffSet: 06h – 07h
Valor inicial = 0000h
Atributo: Leitura e escrita
Tamanho: 16 bits

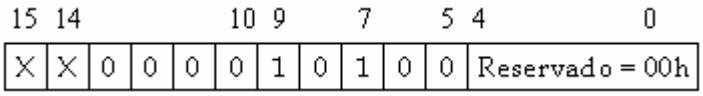


Figura 3.29: Registrador de status

Bit 0 a 3 – Reservados.

Bit 4 Capabilities List – Este bit somente de leitura indica a implementação do dispositivo com o ponteiro para uma lista de capacidades. O valor lógico 0 indica que a lista não está disponível.

Bit 5 66Mhz Capable – Este bit somente de leitura indica a capacidade do dispositivo em suportar o barramento de 66Mhz. No valor lógico 1, o dispositivo suporta o barramento de 66 Mhz.

Bit 6 – Reservado.

Bit 7 Fast Back to Back Capable - Este bit somente de leitura indica a capacidade do dispositivo de aceitar, ou não, transações rápidas. No valor lógico 1, o dispositivo aceita a transação rápida, e no valor 0, o dispositivo não aceita.

Bit 8 Master Data Parity Error – Este bit é implementado somente para o iniciador.

Bits 9 e 10 #DEVSEL Timing – Estes bits somente de leitura indicam ao iniciador o tempo de acionamento do sinal #DEVSEL quando o dispositivo é o alvo da transação. Exceto nas transações de leitura e escrita nos registradores de configuração (ver mais detalhes na secção 3.11.1). A tabela 3.5, mostra os tempos de acionamento do sinal #DEVSEL do alvo.

Bits[10:9]	**Tempos de acionamento**
00	Acesso rápido
01	Acesso médio
10	Acesso Lento
11	Reservado

Tabela 3.5: Tempos do acionamento do sinal #DEVSEL

Bit 11 Signaled Target Abort – Este bit deve ser acionado pelo dispositivo se o mesmo é terminado com aborto (ver secção 3.16.1.3). Não é necessário implementar este bit, pois o alvo nunca utilizará este tipo de terminação.

Bit 12 Received Target Abort – Este bit é implementado somente para o iniciador.

Bit 13 Received Master Abort - Este bit é implementado somente para o iniciador.

Bit 14 Signaled System Error – Este bit é ativado somente se o alvo acionar o sinal #SERR. Não é necessário implementar este bit se o dispositivo nunca acionará o sinal #SERR.

Bit 15 Detected Parity Error – Este bit deve ser acionado somente se o alvo detectar um erro de paridade na fase de dados. Essa ação independe do estado do bit 6 do registrador de configuração de comando [1].

3.22.5 Registrador de Identificação de Revisão

O registrador de identificação de revisão é reservado para o número da revisão feita sobre o dispositivo do fabricante. No Core PCI o registrador assume o valor 1 por ser o primeiro protótipo.

Nome do registrador: Revision ID
Endereço de OffSet: 08h
Valor inicial = 01h
Atributo: Somente leitura
Tamanho: 8 bits

3.22.6 Registrador de Código de Classe

O registrador de código de classe classifica o tipo de dispositivo conectado ao barramento PCI. O registrador está dividido em três campos: classe base, sub classe e interface de programação. Como mostrado na figura 3.30. No Core PCI o tipo de dispositivo é genérico e sua classe não é identificada. Isto é, o Core PCI não pertence a nenhuma classe mostrada na tabela 3.6. Então o registrador assume valor FF0000h.

Nome do registrador: Class Code
Endereço de OffSet: 09h – 0Bh
Valor inicial = FF0000h
Atributo: Somente leitura
Tamanho: 24 bits

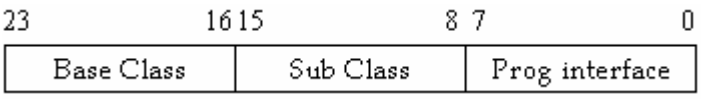


Figura 3.30: Registrador de código de classe

Classes de Bases	Dispositivo
02h	Controlador de Rede
03h	Controlador de Display
0Bh	Processador
...	...
FFh	Não identificado

Tabela 3.6: Códigos de classe

3.22.7 Registrador de Tamanho de Linha de Cache

O registrador de cache é usado pelo barramento mestre junto com o comando de leitura e escrita em memória com o suporte a pré busca. O tamanho é definido em *dwords* e incrementado em 64 bits. No Core PCI não é necessário linha de cache, assim o registrador assume o valor 00h.

Nome do registrador: Cache Size
Endereço de OffSet: 0Ch
Valor inicial = 00h
Atributo: Somente leitura
Tamanho: 8 bits

3.22.8 Registrador de Tempo de Latência

O registrador de tempo de latência define o tempo mínimo que o barramento PCI mestre pode reter o próprio barramento na transação. No Core PCI não requer tempo extra no barramento, assim o registrador assume o valor 00h.

Nome do registrador: Latency Timer
Endereço de OffSet: 0Dh
Valor inicial = 00h
Atributo: Somente leitura
Tamanho: 8 bits

3.22.9 Registrador de Tipo de Cabeçalho

O registrador de cabeçalho é composto de dois campos. Os bits de 0 a 6 definem o tipo de cabeçalho e o bit 7 define se o dispositivo é de simples ou de multifunções. O Core PCI é de função simples, então o registrador assume o valor 00h.

Nome do registrador: Header Type
Endereço de OffSet: 0Eh
Valor inicial = 00h
Atributo: Somente leitura
Tamanho: 8 bits

3.22.10 Registrador para Auto Teste

O registrador de auto teste permite a implementação personalizada de diagnósticos. No Core PCI não foi implementado.

Nome do registrador: BIST
Endereço de OffSet: 0Fh
Atributo: Leitura e escrita
Tamanho: 8 bits

3.22.11 Registradores de Endereços Bases

Estes registradores são usados para determinar a quantidade de endereçamento que o dispositivo PCI necessita da memória do computador. Para cada dispositivo, incluído os de multifunções, há seis registradores para sub dividindo a memória. Isso gera uma capacidade de especificação de até seis posições diferentes de memória para cada dispositivo. Um dispositivo será alvo de uma transação se o endereço de destino da transação coincidir com qualquer um dos intervalos dos registradores bases implementados no dispositivo alvo.

O endereçamento base é determinado pela BIOS durante o *boot* do computador. O espaço de endereçamento tanto pode ser para espaço de endereçamento para dispositivo de I/O, como de memória. No Core PCI, somente foi implementado 4 *dwords* de espaço de memória para a RAM, utilizando o registrador de endereço base 0 (BAR[0]). O registrador assume o valor FFFFFFF0h para requisitar os 4 *dwords* de espaçamento de memória.

Nome do registrador: BAR[x] (x = 0, 1, 2, 3, 4 e 5)

Endereço de OffSet: 10h – 14h – 18h – 1Ch – 20h – 24h

Valor inicial = FFFFFFF0h para o BAR0 e os demais são 00000000h

Atributo: Leitura e escrita

Tamanho: 32 bits

No registrador de configuração, o bit 0 identifica se o registrador é mapeando em um espaço de endereçamento de memória, ou de I/O.

3.22.11.1 Registrador de Espaço de Endereçamento de I/O

A figura 3.31 mostra o tipo de registrador para espaço de endereçamento de I/O com o bit 0 de valor lógico 1.

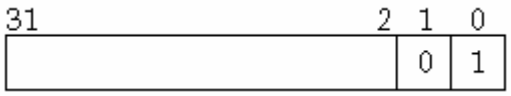


Figura 3.31: Registrador de endereço base de I/O

Bits 31 a 2 - Endereço base para 32 bits de largura

Bit 1 - Reservado - Somente leitura

Bit 0 - Identifica espaço IO - Somente leitura

3.22.11.2 Registrador de Espaço de Endereçamento de Memória

A figura 3.32 mostra o tipo de registrador de espaço de endereçamento de memória com o bit 0 de valor lógico 0.

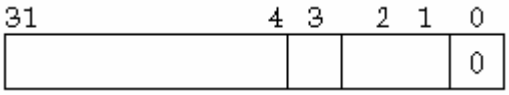


Figura 3.32: Registrador de endereço base de memória

Bits 4 a 31 - Endereço base para 32 ou 64 bits de largura

Bit 3 - Somente leitura

Bits 1 e 2 - Vejam tabela 3.7 - Somente leitura

Bit 0 - Identifica espaço de memória - Somente leitura

Bits[2:1]	Significado
00	Registrador base de 32 bits de largura e o mapeamento pode ser feito em qualquer posição no espaço de memória de 32 bits
01	Reservado
10	Registrador base de 64 bits de largura e o mapeamento pode ser feito em qualquer posição no espaço de memória de 64 bits
11	Reservado

Tabela 3.7: Tipo de acessos de memória em registrador base de endereço

3.22.11.3 Como a BIOS Determina o Tamanho do Endereçamento

O número de bits implementados no registrador depende do espaço de endereçamento requerido pelo dispositivo. Os demais bits são somente de leitura e assumem os valores lógicos 0.

No *boot*, a BIOS escreve nível lógico 1, em todos os bits dos registradores de endereços base (BAR's), se o registrador estiver no espaço de endereçamento de I/O, a BIOS escreverá somente nos bits [31:2]. E o registrador estiver no espaço de endereçamento de memória, a BIOS escreverá somente nos bits [31:4]. Em seguida a BIOS lê os mesmos registradores. Aqueles bits que retornarem com o valor 0 indicam o tamanho requerido para o espaço de endereçamento. Se todos os bits retornarem com o valor 0, o registrador não fora implementado e o espaço de endereçamento não é requerido.

A tabelas 3.8 e 3.9, respectivamente, mostram os valores dos registradores BAR's requerendo espaço de endereçamento de I/O e de memória.

No espaço de endereçamento de I/O, para cada BAR, recebe no máximo 256 bytes, então os bits [31:16] são sempre zero.

Após a BIOS acessar todos os registradores BAR's de todos os dispositivos PCI, a mesma calcula, aloca fisicamente o espaço de memória e atualiza todos registradores BAR's dos respectivos dispositivos (ver mais detalhes na secção 3.24).

BAR	Espaço de endereçamento em I/O
00000000h	Nenhum
FFFFFFFDh	4 Bytes (1 *dword*)
FFFFFFF9h	8 Bytes (2 *dword*s)
FFFFFFF1h	16 Bytes (4 *dword*s)
FFFFFFE1h	32 Bytes (8 *dword*s)
FFFFFFC1h	64 Bytes (16 *dword*s)

Tabela 3.8: Espaço de endereçamento requeridos no registrador BAR para acesso em I/O

BAR	Espaço de endereçamento em memória
00000000h	Nenhum
FFFFFFF0h	16 Bytes (4 *dword*s)
FFFFFFE0h	32 Bytes (8 *dword*s)
FFFFFFC0h	64 Bytes (16 *dword*s)
FFFFFF80h	128 Bytes (32 *dword*s)
FFFFFF00h	256 Bytes (64 *dword*s)
FFFFC000h	16 Kbytes (4K *dword*s)
E0000000h	512 MBytes (128 *dword*s)

Tabela 3.9: Espaço de endereçamento requeridos no registrador BAR para acesso em memória

Por exemplo:

Antes do *boot* do sistema, o dispositivo requer 80h posições de memória, então o BAR[0] assume o valor FFFFFF80h. Depois do *boot* e do cálculo da BIOS, o mesmo registrador assume o valor 00000300h com o limite de acesso de até 0000037Fh, originando 80h posições de memória de *dwords*.

3.22.12 Registrador de Identificação do Fornecedor de Subsistema

O registrador de identificação do fornecedor de subsistema é usado para identificar o usuário da placa ou usuário de um subsistema. No Core PCI não foi implementado, então o registrador assume o valor 0000h.

Nome do registrador: Subsystem Vendor ID
Endereço de OffSet: 2Ch – 2Dh
Atributo: Somente leitura
Tamanho: 16 bits

3.22.13 Registrador de Identificação de Subsistema

O registrador identificação de subsistema é usado para identificar uma placa adicional ou um subsistema. No Core PCI não foi implementado.

Nome do registrador: Subsystem ID
Endereço de OffSet: 2Eh – 2Fh
Atributo: Somente leitura
Tamanho: 16 bits

3.22.14 Registrador de Endereço Base para Expansão da ROM BIOS

Este registrador fornece uma expansão da ROM da BIOS em um dispositivo PCI. No Core PCI não é necessário, então o registrador assume o valor 00000000h.

Nome do registrador: Expansion ROM Base Address
Endereço de OffSet: 30h
Valor inicial = 00000000h
Atributo: Leitura e escrita
Tamanho: 32 bits

3.22.15 Registrador de Linha de Interrupção

O registrador de linha de interrupção indica qual das linhas de interrupção do processador o iniciador deve acionar (ver mais detalhes na secção 3.9.1). No Core PCI não foi implementado o requisito de interrupção.

Nome do registrador: Interrupt line
Endereço de OffSet: 3Ch
Atributo: Leitura e escrita
Tamanho: 8 bits

3.22.16 Registrador do Pino de Interrupção

O registrador de pino de interrupção indica qual dos sinais de interrupções (#INT[A:B:C:D]) serão usados pelo dispositivo. A figura 3.33 mostra o formato do registrador. Somente os três primeiros bits são implementados e os demais são reservados. A tabela 3.10 contém os valores válidos para o registrador. No Core PCI não foi implementada a interrupção.

Nome do registrador: Interrupt Pin
Endereço de OffSet: 3Dh
Atributo: Somente leitura
Tamanho: 8 bits

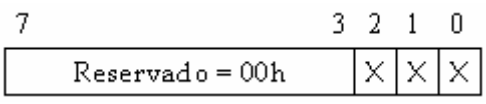


Figura 3.33: Pino de interrupção

Bits [2:0]	Linhas interrupções
000	Nenhum
001	#INTA
010	#INTB
011	#INTC
100	#INTD
10x	Reservado

Tabela 3.10: Pinos de interrupções

3.22.17 Registrador de Mínimo Concedido

Este registrador pode ser usado opcionalmente pelo barramento mestre para especificar o tamanho necessário da transação em rajada de dados. O valor zero indica a não especificação. No Core PCI, o registrador não foi implementado.

Nome do registrador: MinGnt
Endereço de OffSet: 3Eh
Atributo: Somente leitura
Tamanho: 8 bits

3.22.18 Registrador de Máxima Latência

O registrador de máxima latência pode ser usado opcionalmente pelo barramento mestre para especificar a freqüência de acesso pelo dispositivo no barramento. O valor zero indica a não especificação. No Core PCI o registrador não foi implementado.

Nome do registrador: MaxLat
Endereço de OffSet: 3Fh
Atributo: Somente leitura
Tamanho: 8 bits

3.23 Reconhecimento dos Dispositivos PCI's no *Boot* do Computador

O sistema plug and play foi desenvolvido para que os dispositivos se configurem por software automaticamente através da BIOS e *drivers* sem a intervenção humana, eliminando qualquer conflito com outros dispositivos. Em dispositivos antigos, as configurações eram

feitas manualmente através de chaves e jumpers, o que aumenta os riscos de conflitos entre os dispositivos no mesmo barramento como: linhas de interrupção, DMA, espaçamento de endereçamento, entre outros. Para que o barramento PCI seja plug and play, o sistema requer de alguns componentes. Primeiramente, os registradores de configuração, vistos na secção 3.23. Em segundo, o suporte da BIOS PCI a fim de controlar de todos os recursos necessários para acessar e configurar os dispositivos. Em terceiro o *driver* do dispositivo anexado ao sistema operacional. A função do *driver* é controlar os tipos de acessos ao dispositivo de acordo com a aplicação, controlar o tipo de tratamento de interrupção e o tipo de tratamento de erro de paridade [Ewerton 2001].

Em um computador que existe, pelo menos, um barramento PCI a BIOS já é dotada de PCI BIOS, e no *boot* do computador, a BIOS PCI cria um mapa de recursos dos dispositivos PCIs, tal como a versão da BIOS PCI e a quantidade de barramentos no computador. Em seguida a BIOS PCI lê o registrador de configuração VendorID de cada terminal do barramento e para todos os barramentos, incluído os dispositivos de multifunções. Se existir um dispositivo ausente ou registrador VendorID inválido o retorno será FFFFFFFFh devido às linhas dos sinais AD[31:0] estarem em estados de *pull up*. Se o registrador for diferente de FFFFFFFFh, a BIOS PCI reconhece a existência do dispositivo e adiciona-o ao mapa de recursos. Este mapa é composto de informações de cada dispositivo, tais como: o número do barramento, número do dispositivo, número de funções, classe do dispositivo, serviço de interrupção e outros a mais. O procedimento de reconhecimento dos dispositivos é exemplificado no fluxograma da figura 3.34. A figura 3.35 mostra a tela inicial de um computador o reconhecendo os dispositivos no momento do *boot*.

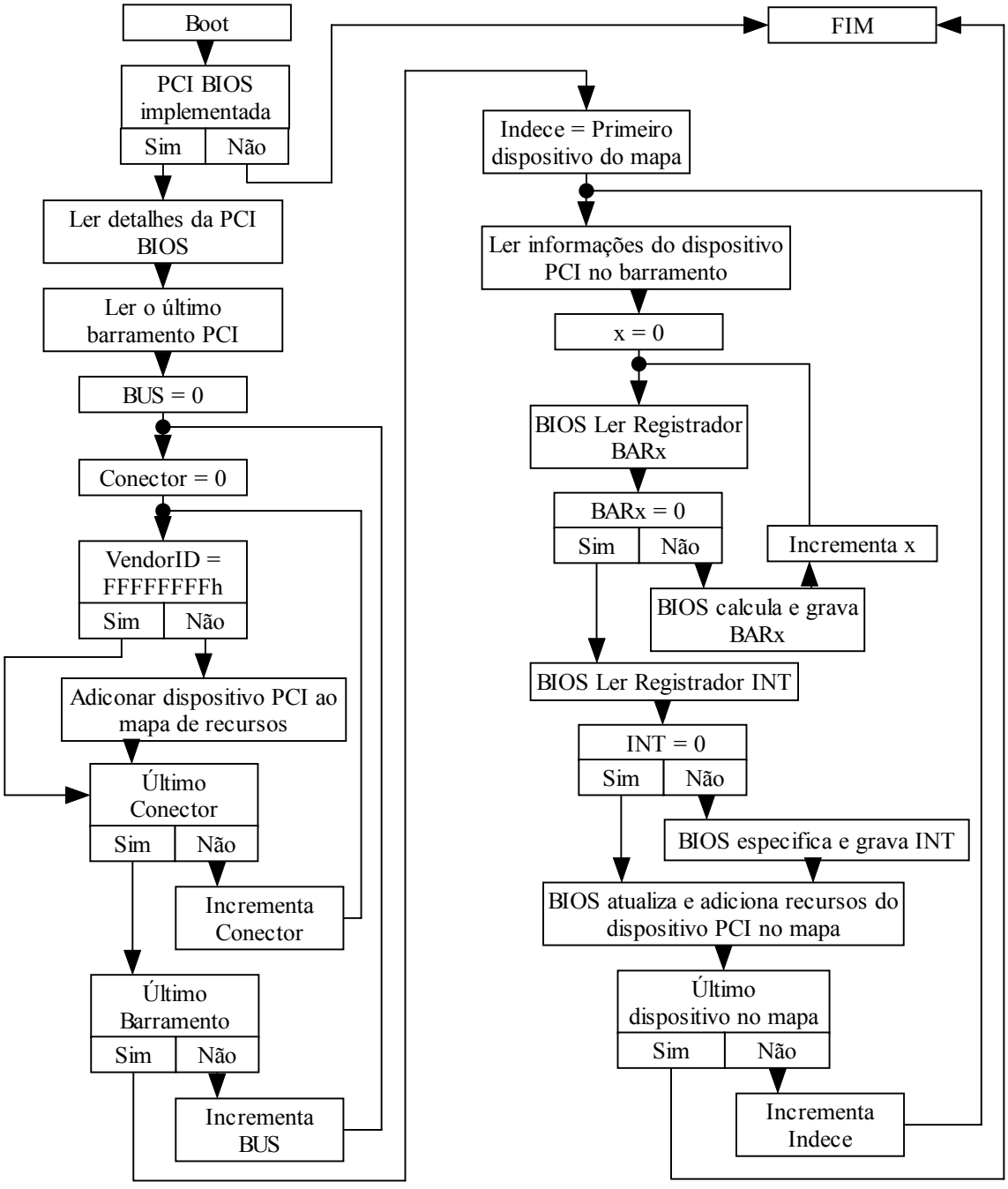


Figura 3.34: Inicialização da BIOS PCI

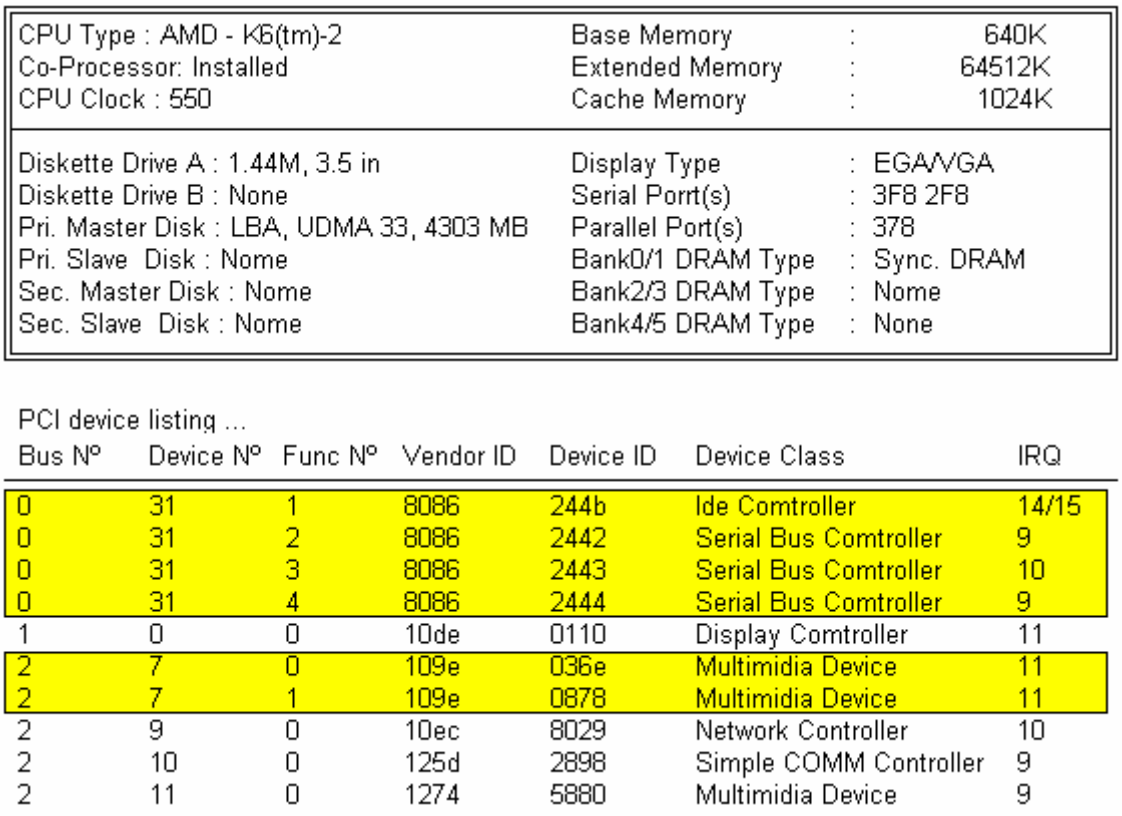


Figura 3.35: Tela do *boot* do computador

Comentários dos recursos dos dispositivos encontrados pela BIOS PCI são:

Bus Nº - Em cada barramento PCI existe um número identificador para distingui-lo dentre os demais barramentos. Já que em um computador existe mais de um barramento PCI.

A figura 3.35 mostra um computador constituído de três barramentos PCI's. No barramento de número 0 estão conectados dispositivos *onboards*, já no barramento de número 1 está conectado uma placa de vídeo AGP e no barramento de número 2 se encontram os dispositivos conectados nos *slots* do computador.

Device Nº e Func Nº- A BIOS PCI atribui dois números identificadores para cada dispositivo no barramento. Um número é destinado à cada dispositivo, conectado ao barramento (Device Nº) e o outro é para cada função (Func Nº) do mesmo dispositivo começando do zero. Na figura 3.35 as tarjas mostram dois dispositivos de multifunções no computador.

O conjunto dos identificadores Bus Nº, Device Nº e Func Nº são extremamente importantes para acessos aos dispositivos PCI's. Isso proporciona um controle mais flexível.

Vendor ID e Device ID – São identificadores particulares para cada dispositivo, ver secção 3.23.1 e 3.23.2. A BIOS PCI necessita deles para que se possa encontrar os dispositivos no barramento e mapeá-los para acesso.

Device Class – O fabricante classifica o seu dispositivo em relação a uma lista de classes determinada pela *PCI SIG*, ver secção 3.23.6. O objetivo desta classificação é para que cada dispositivo possa ser tratado adequadamente pala BIOS PCI e softwares, usufruindo todos os recursos possíveis.

IRQ – A BIOS verifica quais os dispositivos necessitam da interrupção e atribui as linhas de interrupções disponíveis do processador aos mesmos dispositivos, ver secção 3.23.15.

4 O Protocolo PCI Proposto

4.1 Visão Geral

O Core PCI tem a função de fazer a intercomunicação, entre o software de aplicação de alto nível, e o dispositivo de hardware projetado pelo desenhista via barramento PCI. Devido à alta complexidade e custo do desenvolvimento do protocolo PCI, para uma empresa seria mais viável adquirir os protocolos de outras empresas especializadas a implementar o seu próprio protocolo. Lucra-se investindo mais na aplicação.

4.2 Funcionamento

O funcionamento do Core PCI se resume em fazer a configuração plug and play no momento do *boot* do computador e fornecer recursos de acesso ao hardware através do barramento PCI. A figura 4.1 mostra a arquitetura do Core PCI e uma pequena aplicação.

No Core PCI, basicamente, existem os registradores de configuração descritos na secção 3.23 e os registradores de aplicação. Os registradores de aplicação são posições de memórias de 32 bits que podem ser endereçadas e acessadas, tanto pela aplicação do usuário, como pelo barramento PCI, através do registrador de endereço base 0 (BAR[0]), ver secção 3.23.11.

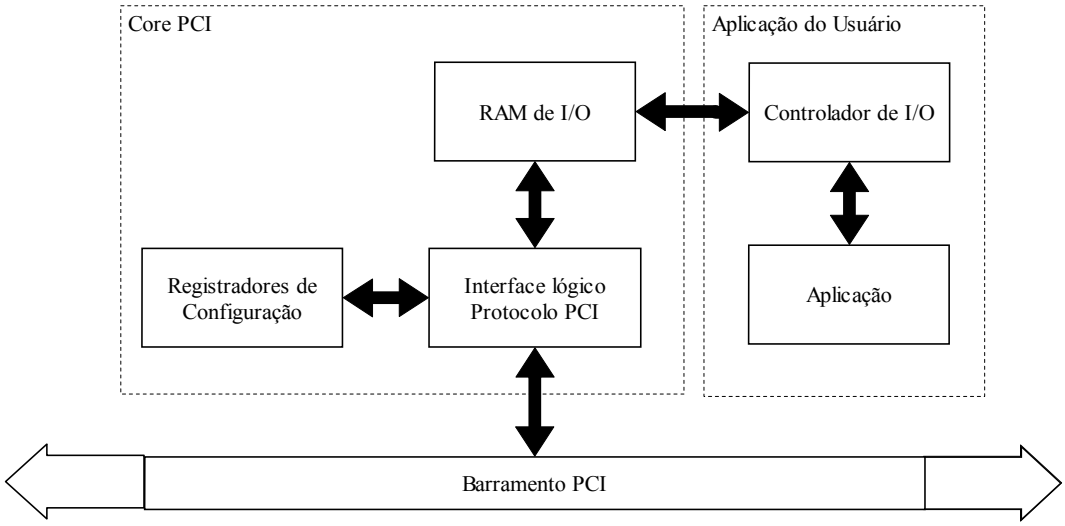


Figura 4.1: Arquitetura do Core PCI

4.3 Aplicações para o Uso do Core PCI

O Core PCI tem a finalidade de fazer a intercomunicação entre um software do computador e um hardware através do barramento PCI. As aplicações são inúmeras, desde

que as saídas e entradas digitais sejam compatíveis em níveis de tensões e correntes de consumo. Eis algumas aplicações:

- Ler sensores externos do computador, tais como: sensores de presença, chaves de toque, sensores de luz e sensores de consumo e outros mais;
- Escrever em atuadores tais como: acionamento de lâmpadas, acionamento de motores elétricos, acionamento de relês, acionamento de válvulas e outras mais;
- Transferência de dados digitais entre o computador e outro aparelho, tal como um osciloscópio digital via porta GPIB ou um outro computador que tenha a mesma placa de interface.

4.4 Registradores

No Core PCI existem dois tipos distintos de registradores de acordo com suas funções. Os primeiros tipos são os registradores de configuração PCI (secção 3.23). Os segundos tipos são os registradores da aplicação denominados de RAM de I/O. Estes registradores de aplicação são para a aplicação do usuário.

4.4.1 Registradores de Aplicação

Os registradores de aplicação são registradores mapeados em memória pelo computador com a função de realizar uma intercomunicação, entre Core PCI e o controlador de I/O da aplicação. No Core PCI existem quatro registradores de aplicação apontados pelos endereçamentos de 0 a 3 como mostra a figura 4.2.

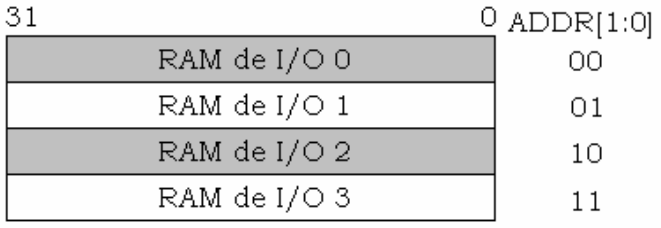


Figura 4.2: Registradores de aplicação

Os endereços 0 e 2 são destinados à escrita pelo Core PCI, à leitura pela aplicação. Os endereços 1 e 3 são destinados à leitura pelo Core PCI, e à escrita pela aplicação. A divisão desses registradores acontece por não existir um controlador de acesso, com finalidade de evitar conflitos de escrita entre o Core PCI e a aplicação, no mesmo tempo e no mesmo endereço.

4.5 Sinais da Aplicação

Os acessos aos registradores de aplicação são feitos através de sinais onde o projetista pode conectar a sua aplicação ao Core PCI. Os sinais são:

- **#IOWR - I/O Write** (Escrita em I/O – Estado de entrada): Este sinal é usado pelo controlador de I/O de aplicação para escrever na RAM de I/O.
- **#IORD - I/O Read** (Leitura em I/O - Estado de entrada): Este sinal é usado pelo controlador de I/O de aplicação para ler a RAM de I/O.
- **IOADDR[1:0] - I/O Address** (Endereço de I/O - Estado de entrada): Estes sinais são usados pelo controlador de I/O da aplicação para apontar quais o registradores serão usados na RAM de I/O.
- **IODATA[31:0] - I/O Data** (Dados de I/O – Três estados): Estas sinais são usados pelo controlado de I/O da aplicação para fazer a transferência de dados entre o controlador de I/O e a RAM de I/O.

5 Prototipação

Neste capitulo estão descritos os mecanismos, ferramentas e componentes utilizados para o desenvolvimento do protótipo. Vale, inicialmente, salientar, que o Core PCI, aqui desenvolvido, é simples em comparação aos Cores PCI's profissionais, que são encontrados em placas de periféricos, tais como: placas de vídeo, placas de rede, *chipsets* e outras mais. Apesar deste Core PCI ser simples, existe um pequeno grau de complexidade. Entretanto, por limitações não foi possível utilizar todos os recursos desejados do barramento, como será justificado neste capitulo.

5.1 Estudo Sobre os Chips Re-configuráveis FPGA's e CPLD's

Antes de mencionar a escolha do chip para conter o Core PCI, é interessante comentar sobre dois tipos de chip de lógicas programáveis muito difundidos nos tempos atuais. São eles: o FPGA e o CPLD. [Einsfeldt 2002]

5.1.1 FPGA

Um chip FPGA é constituído por vários CLB's, IOB's e RAM's, que variam sua quantidade dependendo do tipo e de família do FPGA. Os CLB's contém portas lógicas e flip-flops que podem assumir dos tipos D ou Q. Os IOB's são interfaces de ligações entre os CLBs com os pinos de ligação externa ao FPGA. Os IOB's podem assumir características especiais, dependendo das ligações com outros componentes para manter a compatibilidade, como as ligações com componentes convencionais TTL, CMOS, LVTTL e LVCMOS, ou ligações com certos tipos de barramento como AGP, PCI e GTL. Os IOB's. Além da compatibilidade dos IOB's, seus pinos de saídas dos IOB's podem assumir internamente, estados de *pull up* e *pull down* de acordo com as necessidades. A RAM de uso geral disponível para a aplicação é muito utilizada para processamentos de dados e CPU's. Entre CLB's, existem linhas de conexão que atravessam o chip e permitem que as portas lógicas se interliguem entre si e entre CLB's. As conexões são mantidas em uma RAM estática e precisam ser recarregadas todas vezes que o FPGA é iniciado, porque a RAM perde o seu conteúdo ao desligar o circuito. Existe uma característica não determinística do tempo de atraso entre a interconexões dos CLB's, pois o projeto do usuário é distribuído em uma matriz de CLB's interconectados

por conexões, e estas conexões podem ser diferentes a cada reprodução do projeto. Este processo chama-se roteamento. Por essa característica não determinística, os projetos baseados em FPGA, devem ser síncronos e orientados por um ou mais clocks, garantido a chegada de todos os sinais ao destino no tempo previsto [Einsfeldt 2002]. A figura 5.1 mostra as disposições dos CLB's, IOB's e RAM's em um FPGA xc2s15 da Xilinx.

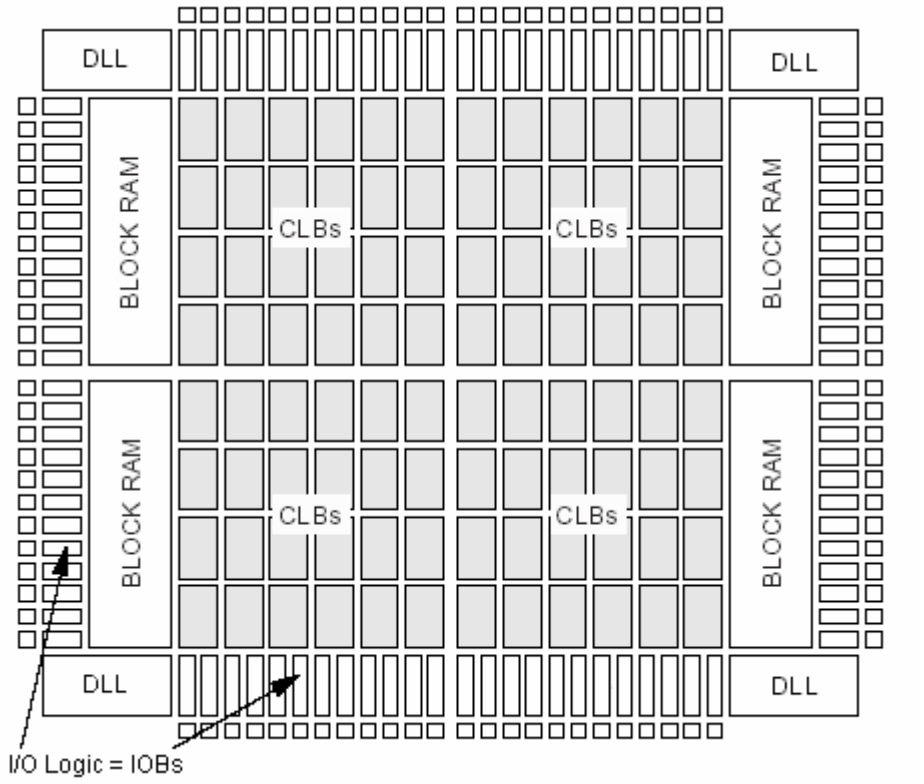


Figura 5.1: Estrutura interna do FPGA xc2s15

Existem quatros tipos de modos de carregamento do FPGA com as suas configurações de conexões dos CLB's gravadas na RAM. Os modos são:

- O modo JTAG é uma interface serial padronizada que utiliza quatro linhas de sinais (TCK, TMS, TDI e TDO) para intercomunicação entre dispositivos. Inicialmente, o modo JTAG foi desenvolvido, somente, para testes de componentes complexos, como processadores e DSP. Atualmente é utilizado para outros fins, como a configuração do FPGA, ou a leitura e a gravação de microcontroladores. Como essa interface é bem difundido este modo é o mais utilizado para configuração. Após a configuração essa porta serve opcionalmente para testes.
- O modo Slave Serial necessita de um dispositivo externo para configurar o FPGA, tal como um microcontrolador.

- O modo Master serial, o próprio FPGA que auto se configura acessando um dispositivo serial, tal como uma *eeprom* serial.
- O modo Slave parallel, é similar ao Slave Serial, porém sua configuração é de oito bits, aumentando a velocidade de configuração.

Os FPGA's são ideais para construções de dispositivos que necessitam de muitos registradores, memórias, operações aritméticas e processamentos, pois estes processos requerem muitas portas lógicas. Por exemplo: compressão e descompressão de áudio e vídeo, implementação de CPU's e processamento na comunicação de dados e telefonia.

5.1.2 CPLD

Os CPLD's em relação ao FPGA possuem poucos flip-flops e poucos recursos aritméticos, mas em compensação pode chegar até 48 entradas por porta lógica, contra no máximo 6 no FPGA. Os CPLD's são ideais na construção de circuitos seqüenciais, máquinas de estados, decodificadores de estados e endereços. As configurações das conexões são mantidas em flash *eeprom* interna do CPLD e não se perde ao desligar o circuito. A estrutura interna de um CPLD é bem diferente do FPGA, porém os detalhes serão omitidos aqui, por ter sido escolhido um FPGA para conter o Core PCI no projeto.

5.1.3 Escolha do FPGA para o Core PCI

Existem dois fatores importantes para escolha do FPGA. Um deles é o valor de custo, pois em comparação aos CPLD's são mais acessíveis financeiramente. O outro fator é composição da sua estrutura. Há uma grande quantidade de portas lógicas, assim, a sua estrutura é ideal para prototipação.

5.2 Definição de VHDL

VHDL (**V**HSIC **H**ardware **D**escription **L**anguage) é uma linguagem elaborada para descrever sistemas eletrônicos digitais, ou melhor, uma linguagem para descrição de hardware. Foi originada pelo programa americano VHSIC (Very High Speed Integrated Circuits) no inicio dos anos 80 para descrever hardware dos componentes. Primeiro padrão pela IEEE (Institute of Electrical and Electronics Engineers) para o VHDL foi em 1987, seguida com uma revisão em 1993. Atualmente existem muitas outras linguagens para

descrição de hardware, tais como Verilog, SDL, ISP e muitas outras, contudo as mais populares são o VHDL e o Verilog. Estes tipos de linguagem foram adotados mundialmente por empresas de CAD para simulação e síntese dos circuitos, nos quais algumas ferramentas são livres para aquisição como WebPack da Xilinx, e outras são comerciais como Active-HDL. Cabe salientar aqui duas principais vantagens de utilizar este tipo de linguagem. A primeira é a possível de fazer simulação do hardware, testar e analisar antes da implementação física. A segunda é a capacidade de fazer a descrição do hardware em diversos níveis de abstração, deixando livre o usuário, seguir a aplicação sem se preocupar de regras e conceitos na construção de baixo nível do hardware [Moraes 2001].

O uso do VHDL traz muitas vantagens como: flexibilidade do projeto, abstração de códigos, independência de tecnologias, reutilização do código, reduz o custo de desenvolvimento, elimina erros de baixo nível e a mais importante, a possibilidade de se fazer simulações. Entretanto existem também algumas desvantagens. O hardware gerado é menos otimizado e falta de pessoal treinado para lidar com a linguagem [Moraes 2001].

5.3 Definição do Hardware para o Core PCI

O hardware desenvolvido é, de forma genérica, o mais abrangente aos vários tipos de exigências de aplicações, seja um alvo que requer controle de interrupção, seja um alvo que requer um controle de paridade, seja um alvo que requer controle de acesso a memória, enfim um hardware flexível para construção do Core PCI de acordo com a aplicação desejada. Os sinais utilizados estão descritos nas secções 3.7 e 4.5.

O diagrama da estrutura do protótipo é mostrado em blocos na figura 5.2.

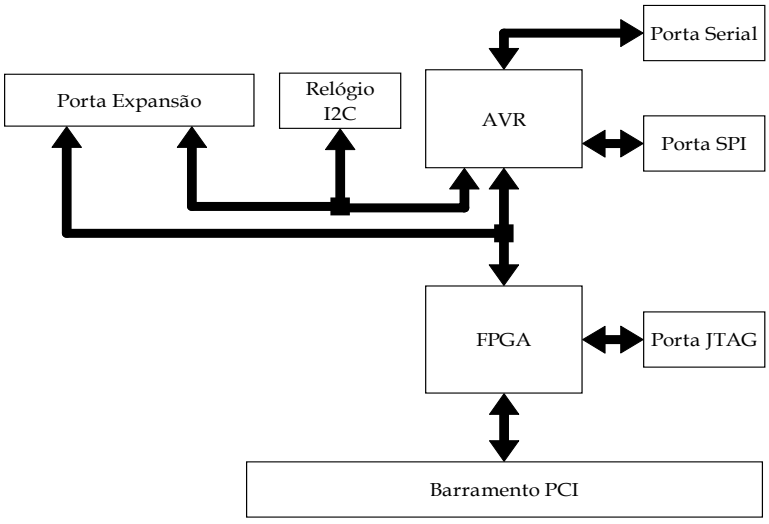


Figura 5.2: Diagrama da estrutura do protótipo

Descrição dos blocos do protótipo:

- **Barramento PCI**.
- **FPGA:** Contém o Core CPI para intercomunicar entre a aplicação e o barramento PCI. Componente da empresa Xilinx, família SpartanII modelo xc2s50 com encapsulamento TQFP144.
- **Porta JTAG:** Usado para configurar o Core PCI no FPGA.
- **AVR:** Microcontrolador usado, tanto para aplicação, como para testar o Core PCI. Componente da empresa Atmel modelo AT90s8515 com encapsulamento PLCC44.
- **Porta Serial:** Usado para fazer o debug da aplicação e da comunicação entre Core PCI e a aplicação.
- **Porta SPI:** Usado para gravar o *firmware* no microcontrolador
- **Relógio I2C:** Um componente com barramento I2C que controla a data e hora. Componente da empresa Dallas, modelo DS1307 com encapsulamento DIP8.
- **Porta de expansão:** Usada para expandir as aplicações futuras.

5.4 Tipo de Aplicação

A aplicação aqui apresentada é simples. Foi implementado um protocolo de comunicação entre o software do computador com o *firmware* do microcontrolador nomeado de protocolo MDH (Protocolo de Manutenção de Data e Hora). A interface do software é mostrada na figura 5.3 e os eventos do menu são:

- **Botão ligar:** Inicia a captura da data e hora no relógio I2C através do barramento PCI.
- **Botão informações:** Exibe os valores dos registradores de configuração do Core PCI.
- **Botão ajustar:** Permite entrar com novos valores de data e hora para atualização no relógio I2C.
- **Botão Sair:** Finaliza o software

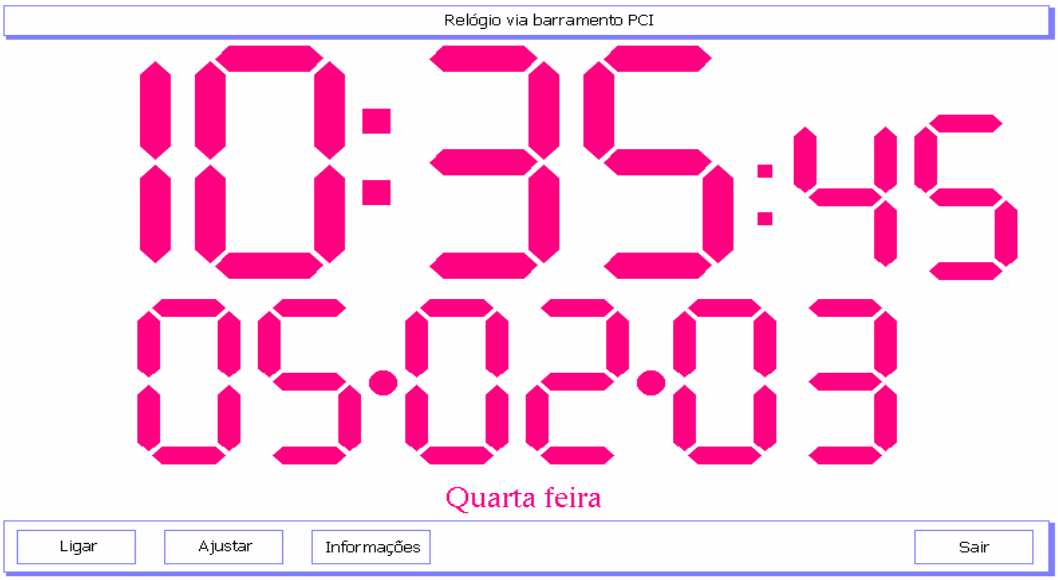


Figura 5.3: Interface do software

5.5 Ferramentas Usadas para o Desenvolvimento

Neste sub capitulo se encontram as ferramentas e os comentários do desenvolvimento do Core PCI. As ferramentas utilizadas são:

- Microsoft Word XP: Documentar todo projeto no decorrer do desenvolvimento;
- SmartDraw 6.0: Usado para criação dos fluxogramas de dados e hardwares;
- Paint: Criar desenhos;
- Tango 2000 P-CAD: Usado para projetar o hardware;
- Ferramentas para bancada de eletrônica: Usadas para a montagem e teste do hardware. Dentro elas incluem-se estação de solda, osciloscópio digital, material antiestática e ferramentas em geral;
- WebPack 5.1i: Usado para o desenvolvimento do Core PCI em VHDL e configuração do FPGA;
- ModelSim 5.6a: Usado para simular o Core PCI;
- Floorplanner 5.1i: Usado para delimitar as áreas criticas, CLB's, do Core PCI no FPGA;
- Timing Analyser 5.1i Usado para analisar tempos de propagação dos sinais internos do Core PCI internamente ao FPGA;

- Editor de texto UltraEdit 6.20b: Usado para escrever os softwares e *firmwares*.
- Compilador C, GCC para DOS: Usado para compilar o software de aplicação.
- Compilador C, DJGPP para AVR: Usado para compilar o *firmware* de aplicação;
- PonyProg 2000 2.05a Beta: Usado para gravar o *firmware* no AVR;
- Mttty 4.11: Usado para comunicação com o microcontrolador AVR para debug do Core PCI;
- Dois Computadores: Um Pentium 4 1.6 Ghz para desenvolvimento do Core PCI e um K6 550 Mhz para funcionamento e teste do protótipo.

5.6 Criação do Hardware do Protótipo

A parte física do projeto foi elaborada no Tango 2000 ou P-CAD. Esta é uma ferramenta de CAD usada na área de eletrônica para criar esquemas eletrônicos e para projetar as placas de circuitos impressos. O P-CAD é composto de várias ferramentas para uso em eletrônica, as principais são:

- Programa para geração de esquema eletrônico;
- Programa para fazer as disposições dos componentes e re-trabalhar os detalhes na placa de circuito impresso;
- Programa para roteamento das trilhas automático sobre a placa de circuito impresso;
- Programa para criar novos componentes.

5.6.1 Criação do Esquema Elétrico do Protótipo

O esquema eletrônico do hardware foi criado a partir do programa Schematic do P-CAD como mostrado na figura 5.4.

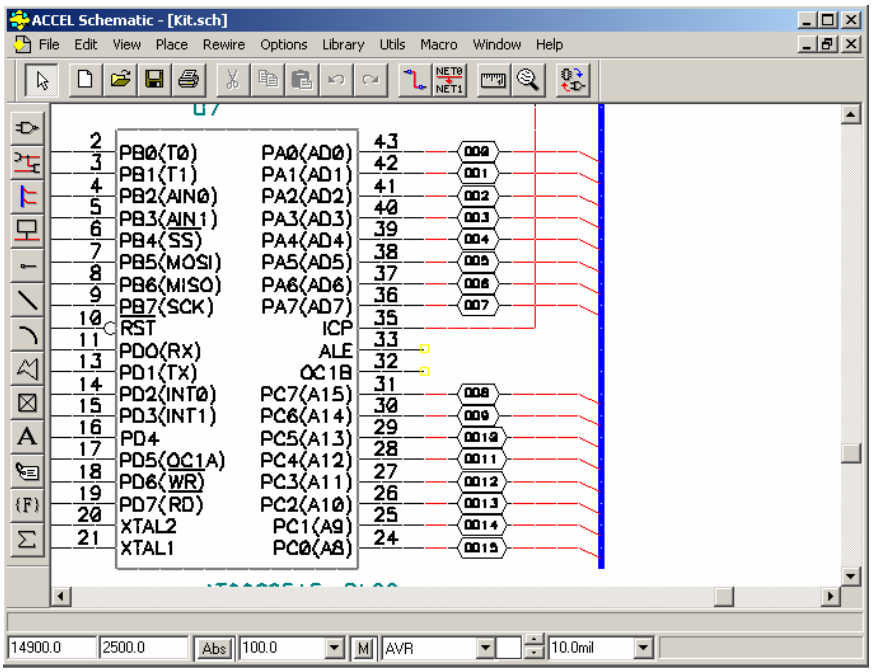


Figura 5.4: Software para gerar esquema eletrônico do protótipo

5.6.2 Criação da Parte Física do Protótipo

Antes de continuar com a parte física do hardware, dois fatores devem ser revistos e considerados. São eles os tempos de propagações dos sinais e as disposições dos componentes na placa de circuito impresso.

5.6.2.1 Especificações dos Tempos

Na construção do hardware que se referente ao Core PCI, os tempos de propagação dos sinais são críticos e a integridade dos dados devem ser mantida. Para que haja uma integridade dos dados garantida. Serão mostradas nos parágrafos abaixo importantes especificações sobre estes tempos.

Pela especificação PCI num clock de 30ns (33Mhz) o tempo de propagação do sinal no barramento está dividido em três categorias (mostradas na figura 5.5). A primeira categoria é o tempo de propagação do sinal de 10ns nas trilhas da placa de circuito impresso (T_prog), que somado ao tempo de escorregamento do clock (T_skew) de 2ns, totaliza 12ns de tempo de propagação do sinal. A segunda categoria, denominada o clock-to-out (T_cto) é de 11ns. É o tempo máximo para que os dados estejam válidos nos pinos de saída do dispositivo PCI, após receber o sinal de clock. A terceira categoria, denominada o setup-timing (T_st), possui

o tempo de, pelo menos, 7ns para que os dados estejam válidos na entrada da porta lógica antes da borda de subida do clock.

No FPGA adotado, todas as exigências referentes aos pinos para o barramento PCI são de responsabilidade do fabricante, tais como: capacitâncias, indutâncias e IOB's compatíveis com o barramento (secção 5.1.1). Outras exigências também são de responsabilidade do fabricante, são elas: o tempo de reposta e de gatilho das portas lógicas internos, contribuindo para diminuir os tempos de T_st e T_cto. Contudo, para que os tempos estejam dentro das especificações requisitadas, três procedimentos devem ser adotados na construção do hardware:

- Melhor utilização dos CLB's para manter as exigências dos tempos T_st e T_cto.;
- Distribuição dos pinos do FPGA para contribuir no agrupamento dos CLB's auxiliando nas exigências dos tempos T_st e T_cto. Visto neste sub capitulo;
- Posicionamento do FPGA na placa de circuito impresso para manter as exigências dos tempos T_prog e T_Skew. Visto neste sub capitulo.

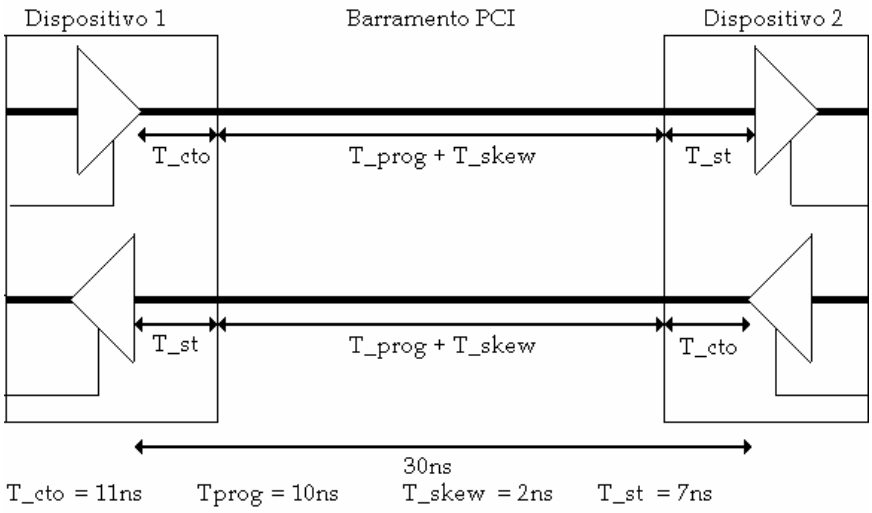


Figura 5.5: Tempos de propagação dos sinais no barramento PCI

5.6.2.2 Distribuição dos Pinos no FPGA

De acordo com as disposições dos sinais, no *slot* do barramento PCI, como mostrado na figura 3.7 da secção 3.3, é recomendável a distribuição dos sinais nos pinos FPGA na

seguinte ordem: #INT[A], CLK, #RST, #GNT, #REQ, AD[31:4], #CBE[3], IDSEL, AD[23:16], #CBE[2], #FRAME, #IRDY, #TRDY, #DEVSEL, #STOP, #LOCK, PERR, SERR, PAR, #CBE[1], AD[15:8], #CBE[0] e AD[7:0].

Ao analisar a arquitetura do FPGA em quadrantes, a melhor distribuição dos pinos é mostrada na figura 5.6. As distribuições dos sinais do barramento PCI para os pinos devem ser simétricas. A simetria se aplica aos sinais que são compartilhados diretamente com outros dispositivos no barramento, exceto os sinais CLK, #INT, #GNT, #RST, #REQ e #GNT, os quais não se ajustam á regra de simetria. A simetria dos sinais pode ser observada abaixo da linha horizontal mostrada na figura 5.6. Com estas distribuições dos pinos, o agrupamento dos CLB's são os mais próximos e suas conexões são as mais curtas, contribuindo para a diminuição dos tempos de propagações dos sinais (T_st e T_cto).

5.6.2.3 Posicionamento do FPGA na Placa de Circuito Impresso

Os pinos do FPGA devem estar no máximo alinhados com o conector do barramento. A figura 5.7 mostra o alinhamento recomendado para o FPGA.

O objetivo do alinhamento é uniformizar a distância de todos os sinais entre o barramento PCI e o FPGA. Este alinhamento é exemplificado na figura 5.7 e as linhas laterais do FPGA representam duas trilhas com máxima uniformidade. Quanto melhor for o alinhamento, mais os tempos de propagação dos sinais serão parecidos.

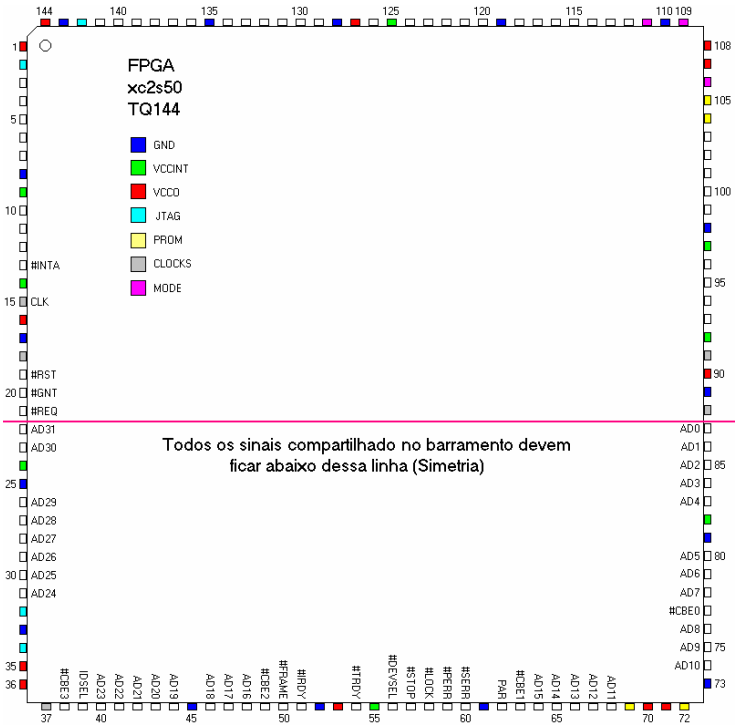


Figura 5.6: Distribuição dos sinais do barramento PCI no FPGA

Pela *PCI SIG*, as distâncias máximas das trilhas, para garantir os tempos T_prog e T_skew entre o barramento e o FPGA são:

- A distância máxima de 6.35cm (2.5 “) para o clock.
- A distância máxima de 3.81cm (1.5”) para as demais trilhas do barramento.

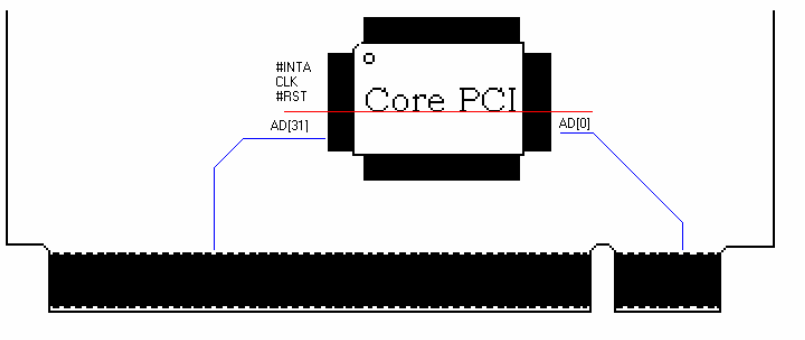


Figura 5.7: Posicionamento do FPGA sobre a placa de circuito impresso

5.6.2.4 Considerações de Alguns Sinais do Barramento PCI

O sinal IDSEL deve ficar entre o sinal #CBE[3] e o sinal AD[23] via regra técnica do barramento PCI.

A placa mãe deve ser notificada sobre a potência consumida pelo hardware conectado ao *slot* através dos sinais #PRSNT[1] e #PRSNT[2] como mostrado na tabela 5.1.

#PRSNT[1]	#PRSNT[2]	Configuração
Aberto	Aberto	Placa ausente
GND	Aberto	Consumo placa no máximo 25W
Aberto	GND	Consumo placa no máximo 15W
GND	GND	Consumo placa no máximo 7.5W

Tabela 5.1: Indicação de potência de consumo do periférico conectado no barramento PCI

Se o barramento JTAG não for implementado, os sinais TDO e TDI do conector PCI devem ser colocados em curto circuito.

5.6.2.5 Elaboração da Placa de Circuito Impresso

Ao usar o programa P-CAD, junto com as informações dos subcapítulos anteriores, a construção da placa de circuito impresso está apta a continuar. No programa PCB do P-CAD, os componentes são posicionados adequadamente (figura 5.8). Neste ponto a placa está pronta

para ser roteada. O roteamento pode ser feito por qualquer programa que aceite o formato do arquivo de rede do P-CAD, tal como o Specctra ou Pro Route. No projeto foi usado o Pro Route, por ser nativo do P-CAD. A figura 5.9 mostra a placa já roteada. A placa roteada é enviada para fabricação, montagem e testes.

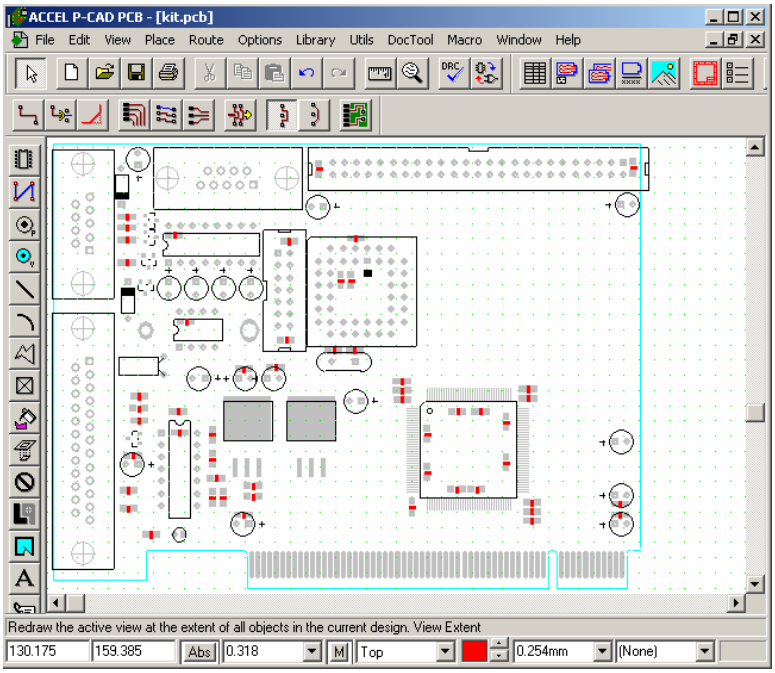


Figura 5.8: Software para gerar a placa de circuito impresso

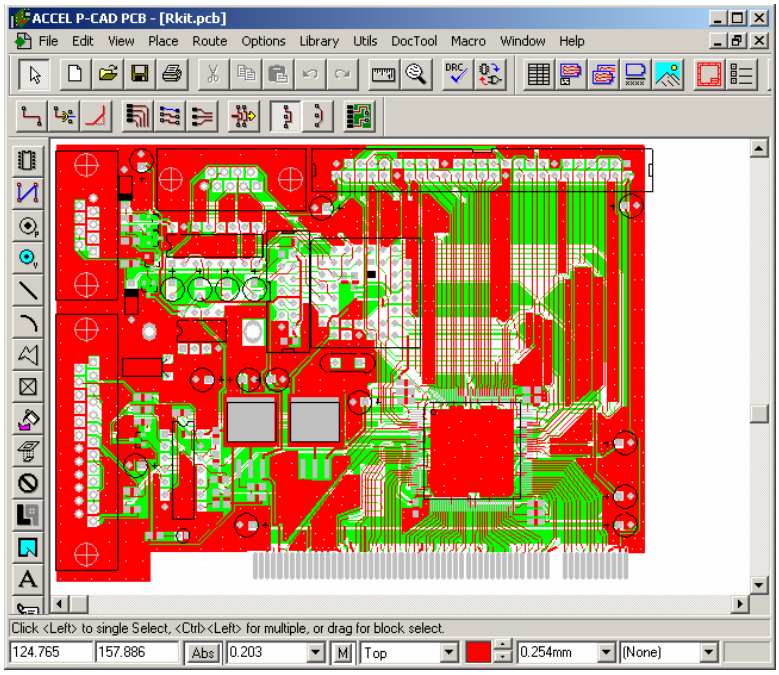


Figura 5.9: Placa roteada e pronta para fabricação

5.7 Descrevendo o Core PCI em VHDL

Com o hardware pronto cabe, agora descrever o comportamento do Core PCI em VHDL mediante ao barramento PCI e simular seus eventos.

5.7.1 Componentes para Um Core PCI

Para um funcionamento satisfatório do Core PCI no barramento, os seguintes módulos são necessários na construção lógica do hardware [PLD 2001]:

- **Estado de máquina:** Controla todo PCI interno e a operação de interface local do Core PCI. Este estado reporta o sequenciamento da operação do barramento PCI;
- **Acesso aos registradores de configuração:** Blocos de cabeçalho de acesso de leitura e escrita nos registradores de configuração;
- **Acesso aos registradores de aplicação:** Blocos de cabeçalho de acesso de leitura e escrita nos registradores de aplicação. Permitindo ao Core PCI, responder a uma transação, quando os endereços são mapeados em memória ou espaço de I/O;
- **Controle de Paridade:** O modulo calcula a paridade dos dados, caso exista um erro e se habilitado, o alvo reporta o erro ao iniciador;
- **Suporte de interrupção:** Um controle lógico para processamento de requisição de interrupção sobre o barramento PCI;
- **Caminhos multiplexados:** Interfaces com o barramento de dados / endereços e comandos / bytes habilitados para os seus respectivos registradores internos do Core PCI;
- **Controle:** O Core PCI assegura uma confiável transferência de dados e sinaliza ao iniciador, o reconhecimento do pedido de transação.

5.7.1.1 Estado de Máquina

O estado de máquina utilizado no Core PCI é mostrado na figura 5.10, adequado para as exigências atuais, porém não o suficiente para utilizar recursos avançados havendo a necessidade de mudar a topologia e adicionar novos estados.

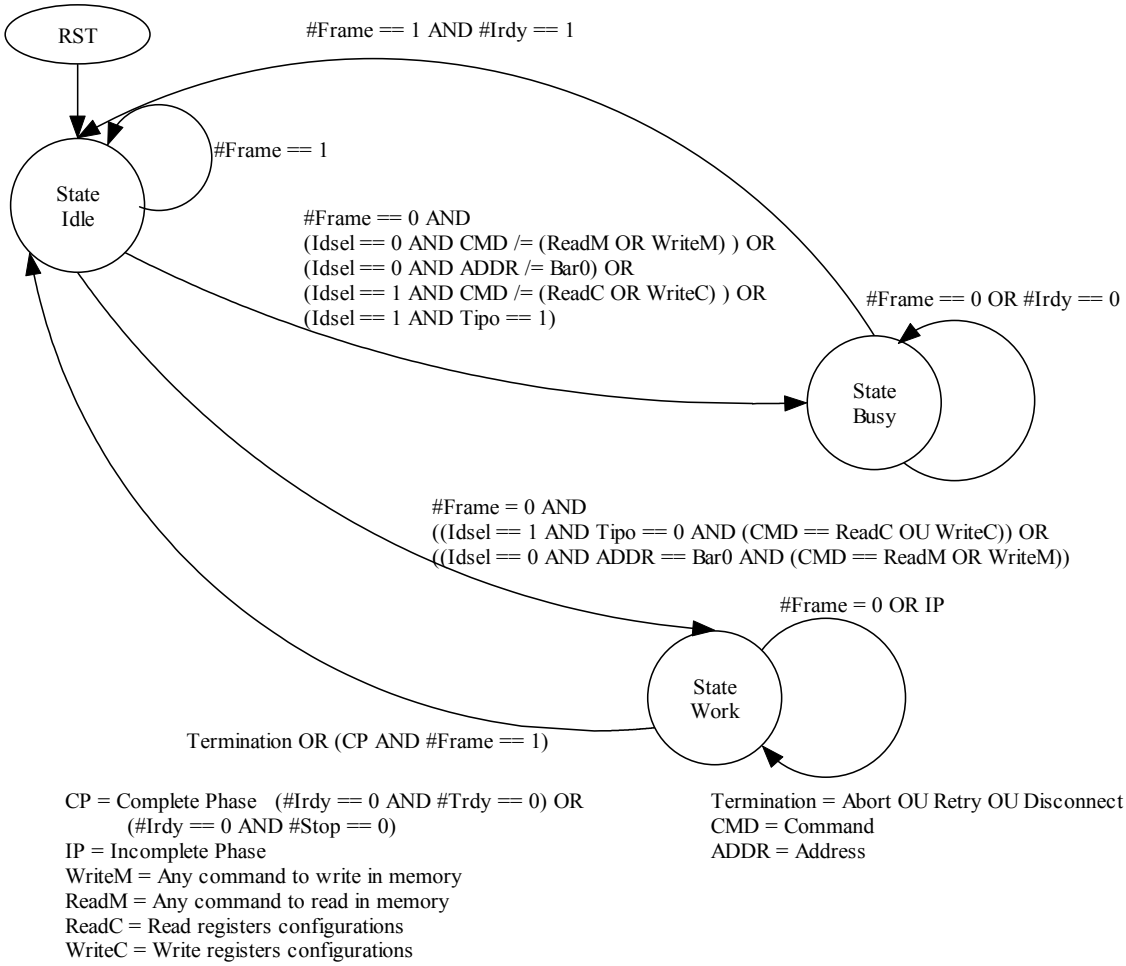


Figura 5.10: Estado de máquina do Core PCI

Neste Core PCI existem somente três estados de máquina, são eles: OCIOSO (idle), OCUPADO (busy) e TRABALHO (Work). O funcionamento do estado de máquina inicia-se após o *reset* do computador. Neste momento o Core PCI fica no estado OCIOSO monitorando o sinal #FRAME do barramento. Quando o sinal #FRAME for ativado, o Core PCI captura o comando e endereço do barramento. Se o sinal IDSEL estiver ativado o dispositivo é alvo para configuração, ou se o sinal IDSEL não estiver ativado e o endereço capturado coincidir com alguns dos registradores de configuração BAR's, o dispositivo é o alvo para transação. Caso contrário o dispositivo não é o alvo da transação e o Core PCI entra no estado OCUPADO, para esperar que o sinal #FRAME seja desativado para entrar no estado OCIOSO novamente.

Hipoteticamente temos o dispositivo como alvo da transação, conseqüentemente, o estado de máquina entra no estado de TRABALHO e somente sai quando a transação é finalizada retornando ao estado OCIOSO. É importante salientar que o estado de inversão do barramento (secção 3.8.3), para este Core PCI não se faz necessário, porque o sinal

#DEVSEL é acionado dois clocks após a fase de endereço (ver secção 3.11.1), dando o clock necessário para inversão do barramento. Caso o dispositivo seja classificado como tipo rápido, novos estados devem ser criados. Um estado de ESCRITA, um estado de LEITURA e um estado de INVERSÃO. O estado de INVERSÃO é usado na transação de leitura para que entre neste estado antes do estado de LEITURA, assim, há tempo necessário para inversão do barramento. Como existe o estado de LEITURA, é criado também um estado de ESCRITA para um controle adequado do estado de máquina no Core PCI.

5.7.1.2 Algoritmo de Estado de Máquina

O algoritmo abaixo mostra o comportamento do estado de máquina em VHDL do Core PCI.

```
/*Determina quando o dispositivo é alvo da transação e controla os pinos de controle e
estado de máquina de operação do dispositivo*/

Se pRST = 0
    sStatePciChange = Idle
Senão Se pClk ↑
     sStatePciTemp       = sStatePci
     sParityTemp         = Parity(pAd & pCbe)
     sTrdyTemp= sTrdy

     caso  sStatePci =
          Idle
               Se  pFrame = 0
                    sAddress = pAd

                    /*Se acesso a registradores e do tipo 0 ou acesso a memória com o
                    endereço dentro do intervalo de endereço de bar0 então o dispositivo é alvo da
                    transação senão é
                    entra em ocupado pois é outro dispositivo que alvo da transação */
                    Se  (pCbe = WriteConfig OU ReadConfig) AND pIdsel = 1 AND
                              pAd[1:0] = 0) OR
                         (pCbe qualquer acesso a memória AND endereço dentro dos limites
                              de BAR0)
                              sStatePciChange = Work
                              sCommand        = pCbe
                    Senão
                              sStatePciChange = Busy
          Work
                    /*Somente o dipositivo sai do estado de trabalho quando o #Frame for
                    desativado e a fase atual for completa e nenhuma finalização pela parte
                    do dispositivo*/
                    Se (pFRAME = 1 AND sPhaseData = Complete) OR sStateFinish /= Normal
                              sStatePciChange = Idle
          Busy =>
                    /*Quando outro dispositivo terminar a transação o barramento entra em estado
```

```
ocioso*/
Se pFRAME = 1 AND pIrdy = 1
        sStatePciChange = Idle
```

5.7.1.3 Acesso aos Registradores de Configuração

O acesso aos registradores de configuração é feito no mesmo clock do barramento quando o sinal #FRAME é ativado. Neste momento, o Core PCI analisa se o dispositivo é alvo da transação, monitorando do sinal IDSEL.

5.7.1.4 Algoritmo de Acesso aos Registradores de Configuração

O algoritmo abaixo mostra o comportamento dos registradores de configuração em VHDL do Core PCI.

```
*****************************************************************************
/*Acesso ao nos registradores de configuração*/
Processo
Se Reset
        /*Inicializar os sinais do Core no momento do reset */
        sRegConfigBAR0                  = X"FFFFFFF0"
        sRegConfStDetectParityError     = 0
        sRegConfStSignaledSystemError   = 0
        sRegConfCmSerrEnable            = 0
        sRegConfCmParityErrorResponse   = 0
        sRegConfCmMemorySpace           = 0

/*Na borda de subida do clock captura ou atualiza os sinais dos registradores de configuração*/
Senão Se pClk ↑
        Se pCommand == ReadConfig AND pStatePci == Work
                Caso pAddress for
                        000000  sAd =  cRegConfigDeviceID + cRegConfigVendorID;
                        000001  sAd =   Status + Command
                        000010  sAd =    cRegConfigClassCode + cRegConfigRevisionID;
                        000011  sAd = Test + func + latency + cache
                        000100  sAd =    sRegConfigBAR0;
                        outros endereços  sAd =  0

        Senão Se pPhaseData == Complete AND pCommand == WriteConfig AND
                pStatePci == Work
                Caso pAddress é
                        000001 Comando + status = pAdIn
                        000100  sRegConfigBAR0[31:4] = pAdIn[31:4];

Processo
/* Na borda de descida do clock se atualiza o sinal temporário para que na borda de subida o sinal
lido pelo iniciador esteja estável*/
```

```
Se pClk ↓
        Se  pCommand == ReadConfig AND pStatePci == Work
                pAdOut = sAd

        pBar0 = sRegConfigBAR0
```

5.7.1.5 Acesso aos Registradores de Aplicação

O acesso aos registradores de aplicação é feito no mesmo clock do barramento onde o sinal #FRAME é acionado. Neste momento, o Core PCI analisa se o dispositivo é alvo da transação, fazendo isto analisando a não ativação do sinal IDSEL e o endereço é igual a um dos registradores de configuração BAR's.

5.7.1.6 Algoritmo de Acesso aos Registradores de Aplicação

No algoritmo abaixo mostra o comportamento dos registradores de aplicação em VHDL do Core PCI.

```
*****************************************************************************
/*Acesso ao RAM de I/O*/

Processo
/*Na borda de subida do clock captura ou atualiza os sinais dos registradores de aplicação*/
Se pClk ↑
        Se pCommand == ReadMemory AND pStatePci == Work
                sAd = RamIO (pAddress*2)

        Se pWriteApl == 1 E pReadApl == 0
                pDataOut = RamIO (pAddressApl*2 + 1)

Processo
/*Na borda de subida do clock captura ou atualiza os sinais dos registradores de aplicação*/
Se pClk ↑
        Se pPhaseData == Complete AND pCommand == WriteMemory AND pStatePci == Work
                RamIO(pAddress*2 + 1) = pAdIn

        Se pWriteApl == 0 E  pReadApl == 1
                RamIO(pAddressApl*2) = pDataIn

Processo
/* Na borda de descida do clock se atualiza o sinal temporário para que na borda de subida o sinal
lido pelo iniciador esteja estável*/
Se pClk ↓
        Se  pCommand == ReadMemory AND pStatePci == Work
                pAdOut = sAd
```

5.7.1.7 Controle de Paridade

O controle de paridade é feito quando o Core PCI se encontra no estado de TRABALHO nos comandos de escrita e leitura. Porém, no Core PCI, o controle de paridade não foi implementado por dois motivos: Primeiro, o software de baixo nível faz todo o controle de erro de paridade, e com isto, diminuiu o tempo de desenvolvimento. O segundo motivo, será justificado na secção 5.8. Todavia, o sinal PAR deve ser gerado por todos os dispositivos PCI's.

5.7.1.8 Algoritmo de Geração de Paridade

O algoritmo abaixo mostra o mecanismo de geração de paridade em VHDL do Core PCI.

```
/*Atualiza o sinal PAR  na fase completa e na transação de leitura*/
pPar =
        sParity quando
                sStatePciPrior = Read AND sTrdyPrior = 0 AND (sCommandPrior = ReadConfig
                 OR ReadMemory)
        Senão
                Z
```

5.7.1.9 Suporte de Interrupção

O mecanismo de interrupção do Core PCI não foi implementado por dois motivos: Primeiro, a aplicação não necessita de interrupção. O segundo motivo, será justificado na secção 5.8.

5.7.1.10 Caminhos Multiplexados

Conforme explicado na secção 3.7.2, o caminho do barramento de dados e comandos são multiplexados. Na fase de endereço, o Core PCI captura e salva o endereço e comando nos seus respectivos registradores. Na fase de dados o Core PCI captura e salva os dados e bytes habilitados nos seus respectivos registradores.

5.7.1.11 Controle

O controle se resume em outros pequenos controles para garantir uma segura transferência de dados. Tais como:

- Identificar quando uma fase de dados é completa para encerrar com segurança a transferência de dados atual;
- Controla o tipo de finalização de dados que pode ser: Retry, Abort e Disconnect;
- Sinaliza ao iniciador, o reconhecimento de pedido de transação, se o dispositivo for alvo da transação.

5.7.1.12 Algoritmo de Controle

No algoritmo abaixo mostra os mecanismos de controle em VHDL do Core PCI.

```
/*sTarget é verdadeiro se o estado atual for igual a Work (trabalho) e o estado anterior for diferente de
Idle (Ocioso)*/
sTarget = sStatePci == Work AND  sStatePciPrior /= Idle

/*sEnd é verdadeiro se o estado atual for igual a Idle (Ocisoso) e o estado anterior for diferente de Idle
(Ocioso) e Busy (Ocupado)*/
sEnd = sStatePci == Idle AND sStatePciPrior /= Idle AND sStatePciPrior /=  Busy

/*Atribuido os estados dos sinais #Devsel, #Stop e #Trdy de acordo com os estados atuais*/
sDevsel  =
        0 quando
                sTarget
        1 quando
                sEnd
        Senão
                Z
sTrdy  =
        0 quando
                sTarget
        1 quando
                sEnd
        Senão
                Z

sStop   =
        0 quando
                sTarget AND pIrdy == 0 AND pFrame == 0
        1 quando
                sEnd OR
                (sTarget E AND
                        ( (pIrdy == 1 AND pFrame == 0) OR (pIrdy == 0 AND pFrame == 1)))
        Senão
                Z

*****************************************************************************
/* Verifica quando a fase é completa de acordo com as condições dos sinais em teste*/
sPhaseData =
        Complete quando
                (pIrdy == 0 AND sTrdy == 0) OR (pIrdy == 0 AND sStop == 0)
```

```
        Senão
                Incomplete

/* Testa o tipo de finalização do alvo se é Retry ou Disconnect ou Abort ou normal de acordo com as
condições dos sinais em teste*/
sStateFinish  =
        Retry  quando
                (sStop == 0 AND sDevsel == 0 AND sTrdy == 1)
        Disconnect quando
                (sStop == 0 AND sDevsel = 0 AND sTrdy == 0)
        Abort quando
                (sStop == 0 AND sDevsel = 1)
        Senão
                Normal
```

5.7.2 Escrevendo o Core PCI

Com os estudos, conceitos e algoritmos do Core PCI já especificados, basta agora descrever o hardware em VHDL e simulá-lo seu comportamento. A ferramenta WebPack é fornecida pela empresa Xilinx para descrever hardware (FPGA's e CPLD's) no formato EDIF, VHDL ou Verilog. Esta ferramenta é o centro do desenvolvimento do Core PCI, nela encontram-se sub menus para acesso a outras ferramentas de auxilio, tais como: Floorplanner, FPGA Editor, Timing Analyzer, XPower, gravador para o FPGA, simuladores para código e muitas outras ferramentas.

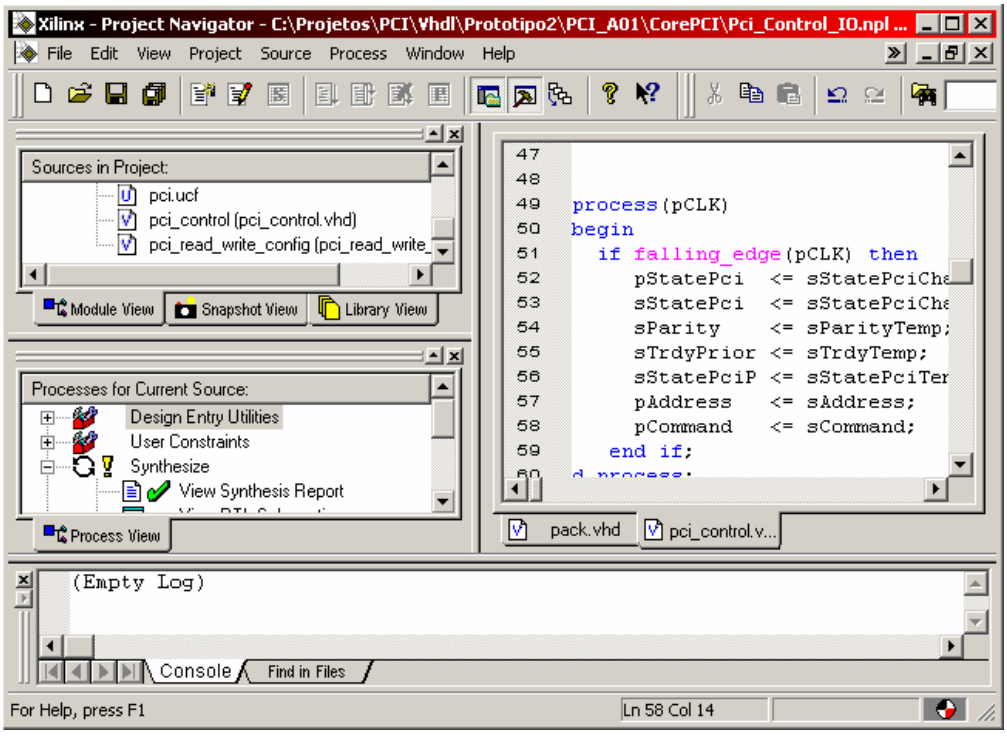


Figura 5.11: Ambiente de desenvolvimento do Core PCI

No WebPack foram descritos dois tipos de comportamento em VHDL, um deles é o próprio Core PCI e o outro é uma seqüência de comandos para simular o funcionamento do protocolo PCI. Essa seqüência de comandos tem por objetivo testar o Core PCI no simulador. A figura 5.11 mostra o ambiente WebPack e na figura 5.12 o ambiente de simulação ModelSim.

ModelSim é uma ferramenta poderosa, embora limitada para simulação que acompanha o WebPack. Para aquisição da versão completa, é necessário o pagamento do registro.

Com auxilio da ferramenta Timer Analyser, se pode analisar e identificar os tempos de resposta e propagações dos sinais internos do FPGA (T_st e T_cto). Com auxilio da ferramenta Floorplanner é possível delimitar área para alcançar metas, que no caso do Core PCI, são as disposições dos CLB's críticos que comprometem a velocidade. A figura 5.14 mostra o resultado da analise do Core PCI e no circulo mostra que a velocidade média de trabalho é de 26.068ns ou 38.361209MHz acima do recomendado (30ns ou 33Mhz), porém as análises não se resumem a isto. Existem muitos outros fatores que devem ser estudados e analisados.

Com o Floorplanner, pode-se delimitar as áreas internas do FPGA para usos adequados dos CLB's. Em um FPGA, que contém o Core PCI junto com a aplicação, é recomendável criar três grupos de disposições dos CLB's. Um deles seria o grupo Core PCI, o outro o grupo interface PCI e finalmente o grupo da aplicação (como mostrado na figura 5.13). Nesta distribuição, as portas lógicas tendem a ficarem mais próximas e com suas conexões mais curtas possíveis. Esse tipo de distribuição reduz o tempo de propagação do sinal. Entretanto, o Core PCI desenvolvido contém somente o grupo Core PCI e o grupo interface.

O software Floorplanner é uma ferramenta gráfica de posicionamento dos CLB's e pinos no FPGA. Esses são representados em forma hierárquica, e opcionalmente, podem ser reagrupadas de acordos com as necessidades (como mostrado na janela superior esquerda da figura 5.15). Estes grupos podem ser organizados e identificados através de cores para facilitar o trabalho de posicionamento e análise. No Floorplanner é possível criar áreas de uso de CLB's para cada grupo descrito anteriormente. A janela esquerda da figura 5.15, mostra a áreas delimitada no FPGA para o Core PCI.

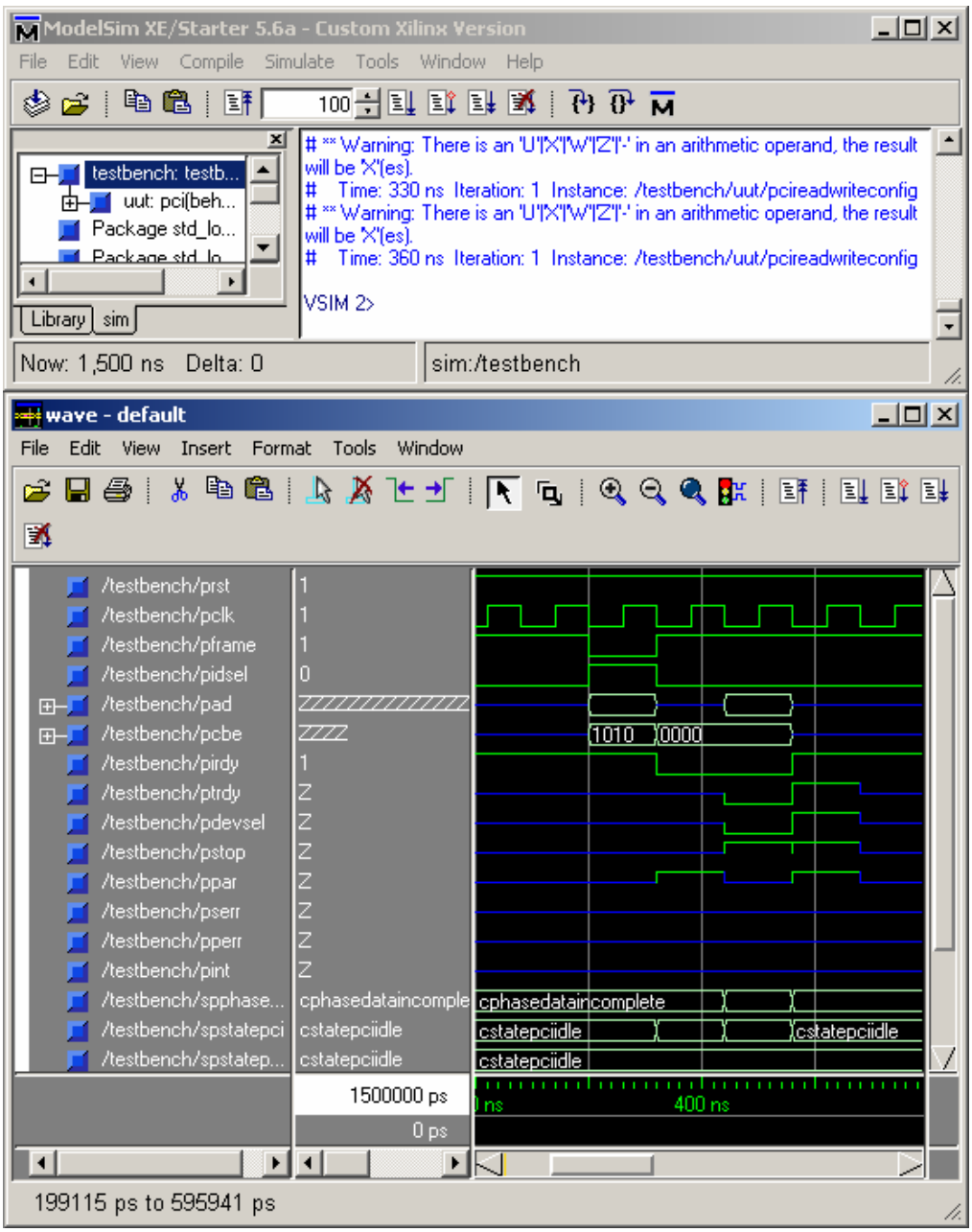


Figura 5.12: Ambiente de simulação do Core PCI

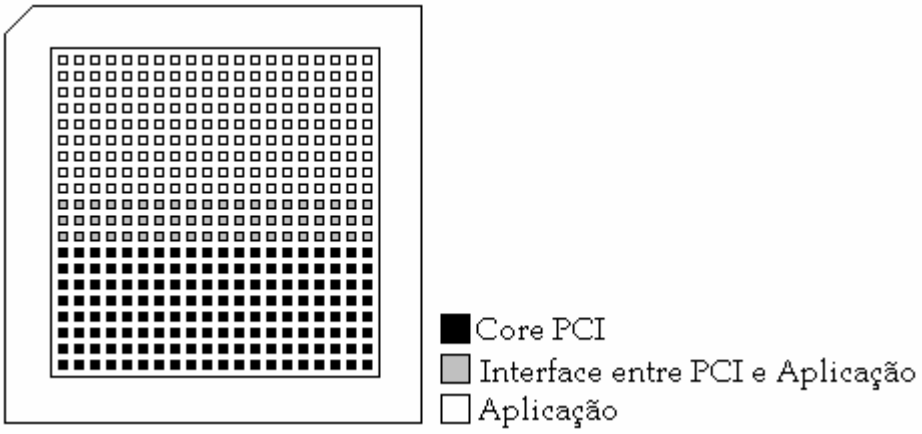


Figura 5.13: Distribuição da área interna do FPGA recomendada para desenvolvimento

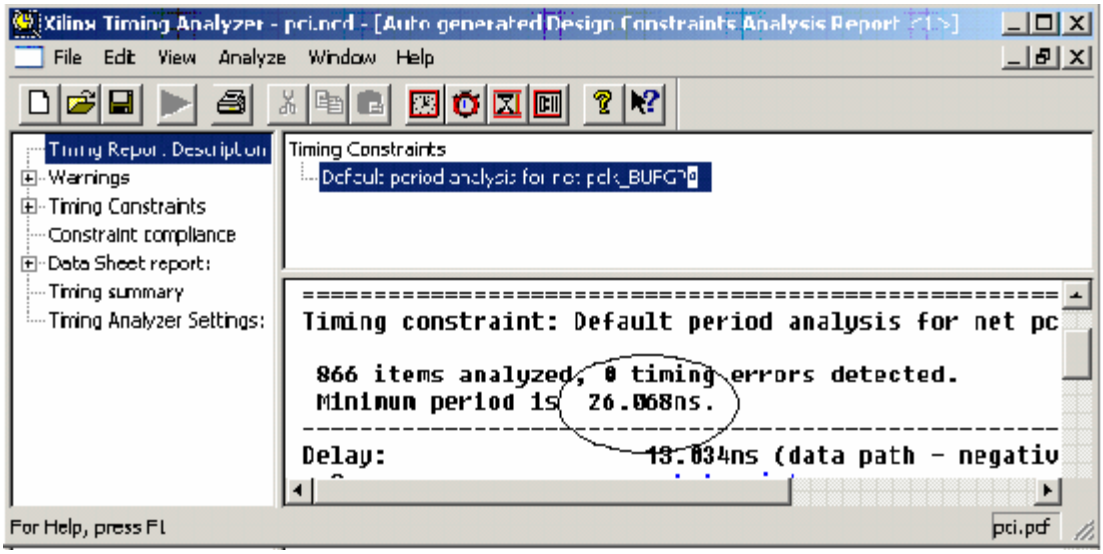


Figura 5.14: Analise de propagação de sinal no FPGA

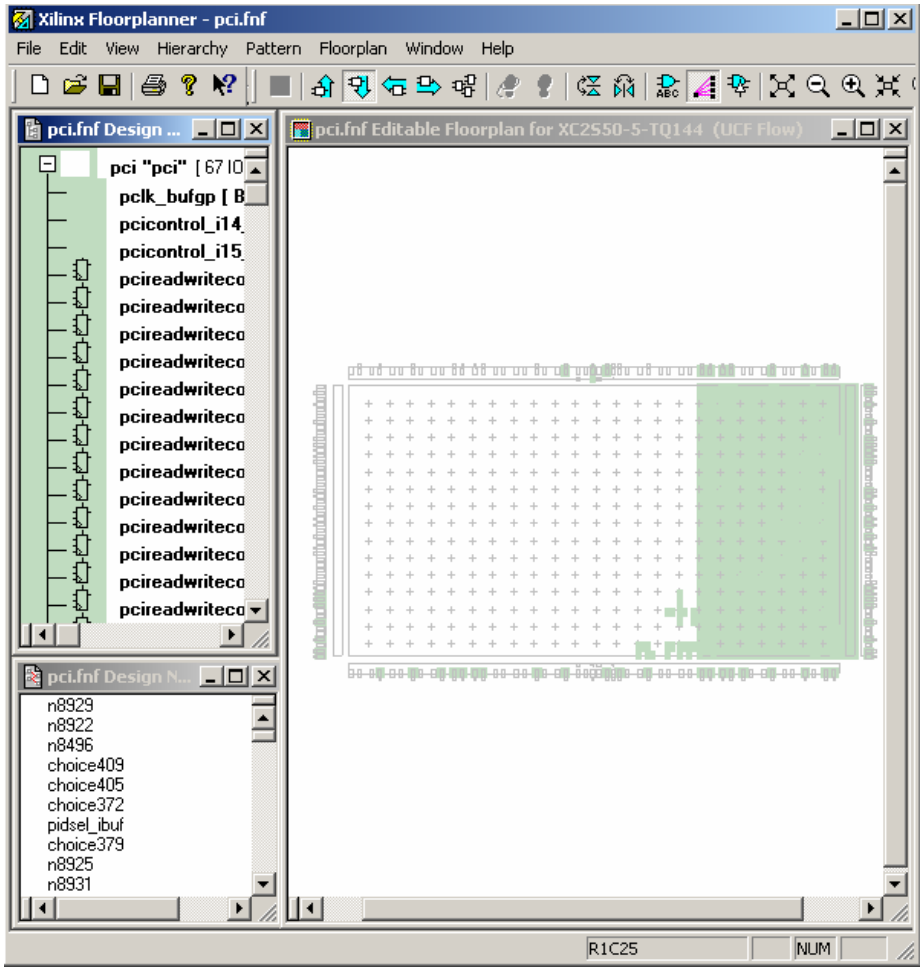


Figura 5.15: Especificando área no FPGA com o Floorplanner

5.8 Características do Core PCI Implementado

Mediante a necessidade simples da aplicação e, principalmente, pelos tempos de propagação dos sinais internamente do FPGA, as características do Core PCI são:

- **Não contém o controle de paridade:** O software de alto nível constantemente lê a aplicação. Caso exista alguma inconsistência nos dados por erros de paridade, o próprio software se encarrega de filtrar as informações lidas, baseadas nos dados lidos anteriormente. No procedimento de escrita o próprio *driver* se encarrega de confirmar o que escreveu. Caso haja algum erro o mesmo retransmite os dados;
- **Não contém controle de interrupção:** A aplicação não necessita da atenção do processador. Isto porque o software está constantemente monitorando a aplicação;
- **Acesso a registradores de configuração:** Estes registradores são fundamentais para todos os dispositivos PCI's;
- **Acesso a registradores de aplicação à memória:** Registradores de dados da aplicação são tratados como RAM's pelo processador;
- **Não implementado JTAG:** O Core PCI não necessita de depuração ou teste via JTAG. Isto acontece através de porta serial do microcontrolador AVR, por ser mais prático e simples;
- **Não suporta modo em rajada de dados (burst):** A aplicação é composta por poucos registradores de dados e independentes entre si, sem a necessidade do busrt. O caso não é o mesmo para uma memória de vídeo, pois com uma única transação em modo burst se pode capturar uma imagem composta por várias posições subseqüentes de memórias;
- **O alvo é de função simples:** O Core PCI é composto de um único dispositivo, ou seja, só há uma função;
- **Foi somente implementada a finalização desconecte com transferência de dados (Disconect):** O Core PCI sempre está pronto para transação e aceita qualquer tipo de acesso. Então não há necessidade de se implementar os outros tipos de finalizações. Somente foi usado este tipo de desconexão para cancelar uma transação em modo de rajada pedida pelo iniciador;
- **A velocidade de resposta de transação do alvo é classificada como tipo médio:** A velocidade de resposta é classificada como tipo médio, porque o sinal #DESVEL é acionado dois clock's após a fase de endereço, devido a uma limitação no

desenvolvimento do Core PCI. Para decodificar, tanto o endereço, quanto o comando, e prontificar-se para a transação. O Core PCI necessita um clock após a fase de endereço.

5.9 Softwares das Aplicações

Os softwares utilizados para a programação das aplicações foram feitos pela linguagem de programação C. O compilador GCC é usado para sistema operacional DOS e o compilador DJGPP para o microcontrolador AVR.

5.9.1 Software em Ambiente DOS

A linguagem de programação utilizada no desenvolvimento do software da aplicação foi o compilador livre GCC da GNU. O software de aplicação é mostrado pelo fluxograma na figura 5.16.

Para que o software funcione no barramento PCI, a BIOS deve ser dotada de suporte para o mesmo, chamado de BIOS PCI. A BIOS PCI fornece recursos para acesso aos dispositivos PCIs e estes recursos são os serviços da interrupção 1Ah. Os recursos utilizados pelo software são mostrados e comentados na tabela 5.2.

5.9.2 Software para o AVR

A linguagem de programação utilizada no *firmware* da aplicação foi o compilador livre DJGPP para o AVR. O *firmware* da aplicação é mostrado no fluxograma na figura 5.17.

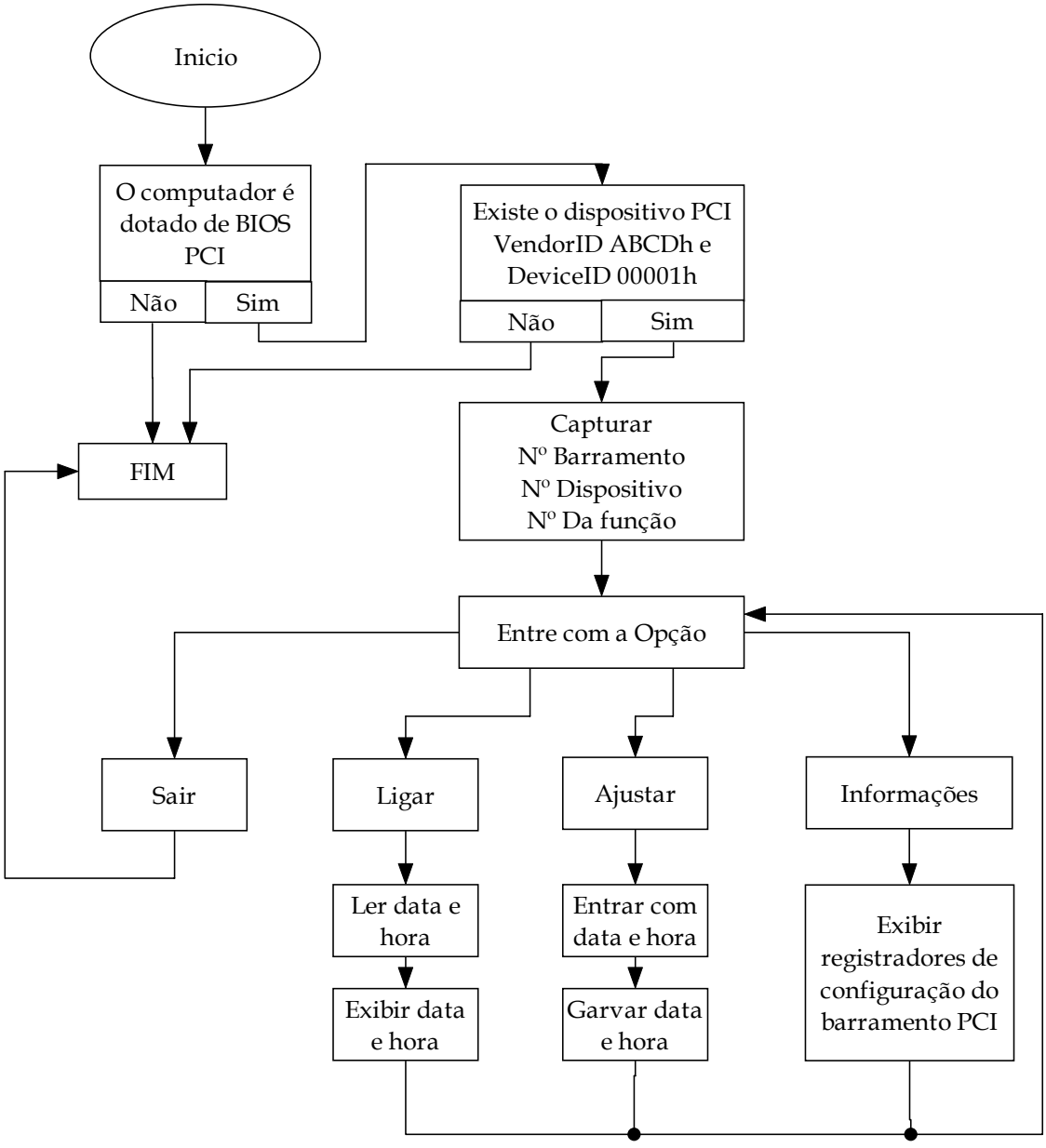


Figura 5.16: Fluxograma do software de aplicação DOS

Serviço	Comentário	Entrada	Principal Retorno
B101h	Verifica se a BIOS é dotada de BIOS PCI.		Instalado ou não.
B102h	Encontra dispositivo no barramento PCI.	Nº VendorID. Nº DeviceID. Nº da função.	Instalado ou não. Nº barramento. Nº Dispositivo e função.
B10Ah	Lê o valor em *dword* nos registradores de configuração.	Nº Barramento. Nº Dispositivo. Nº Da função. Nº Do registrador.	Valor em *dword* e se o processo foi bem sucedido.
B10Bh	Grava o valor em *dword* nos registradores de configuração.	Nº Barramento. Nº Dispositivo. Nº Da função. Nº Do registrador. Valor em *dword* a ser escrito.	Se o processo foi bem sucedido.

Tabela 5.2: Principais serviços da interrupção 1Ah do barramento PCI utilizados pelo software

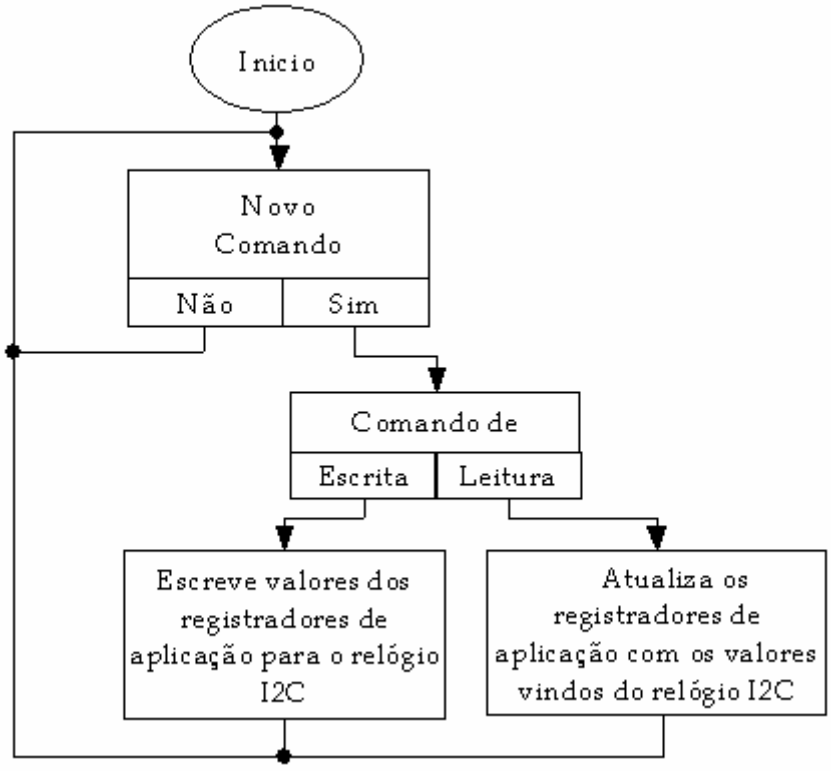


Figura 5.17: Fluxograma do software de aplicação AVR

O *firmware* é gravado no microcontrolador AVR pelo software PonyProg mostrado na figura 5.18.

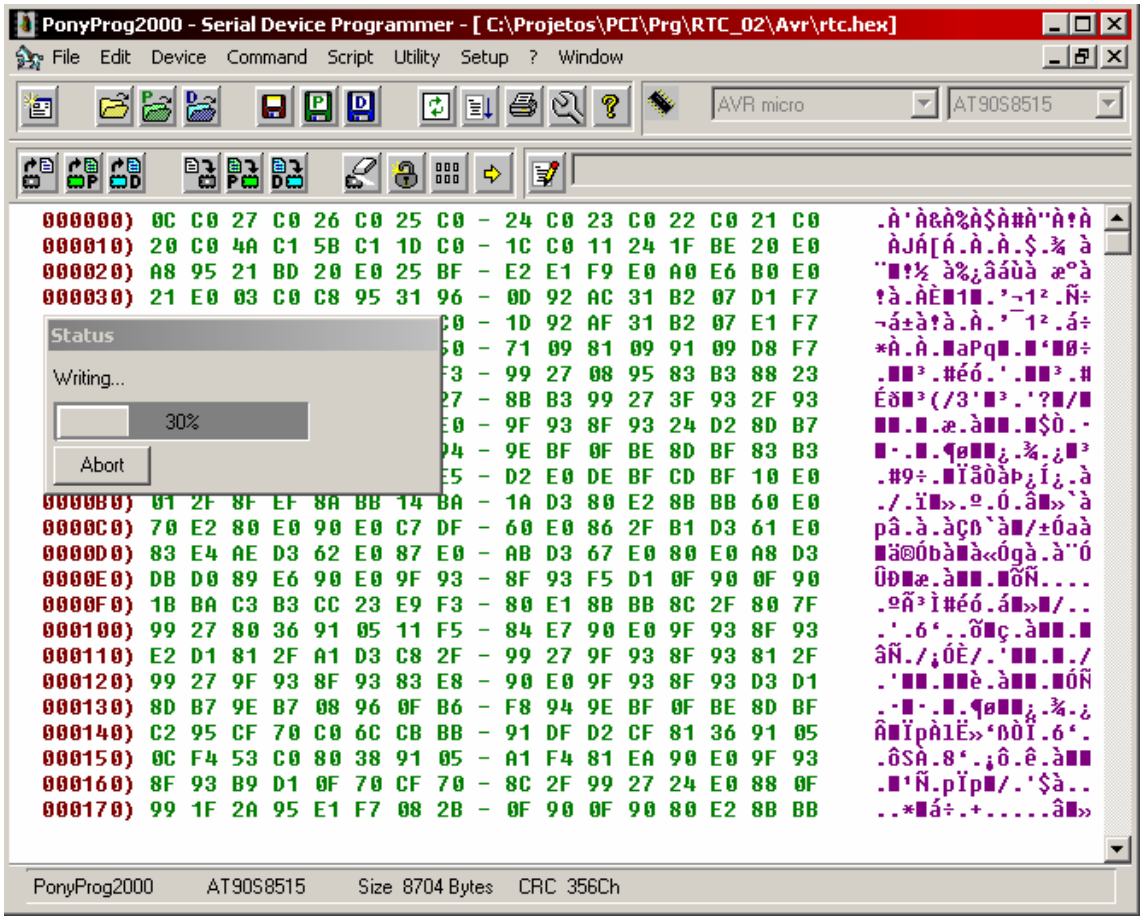


Figura 5.18: Gravador do microcontrolador AVR

5.9.3 Protocolo Comunicação entre Software e *Firmware*

Os fluxogramas, por si só, dão uma noção intuitiva de como funcionam as aplicações. Será feito um breve comentário sobre o funcionamento do protocolo de comunicação MDH entre o software alto nível com o *firmware* da aplicação. O protocolo MDH somente utiliza dois dos quatro registradores de dados do Core PCI, um registrador de saída e outro registrador de entrada. Os comandos utilizados em C, no software para acessarem a esses registradores são, _farpeekl para leitura e _farpokel para escrita, são utilizados seletores de memória. Porém, existem outros modos para acessarem as memórias. Dos 32 bits do registrador do Core PCI, somente foram utilizados os 8 primeiros bits por motivo da compatibilidade com a arquitetura do microcontrolador AVR de 8 bits. A figura 5.19 mostra a interligação entre os registradores de aplicação como as portas de I/O do AVR.

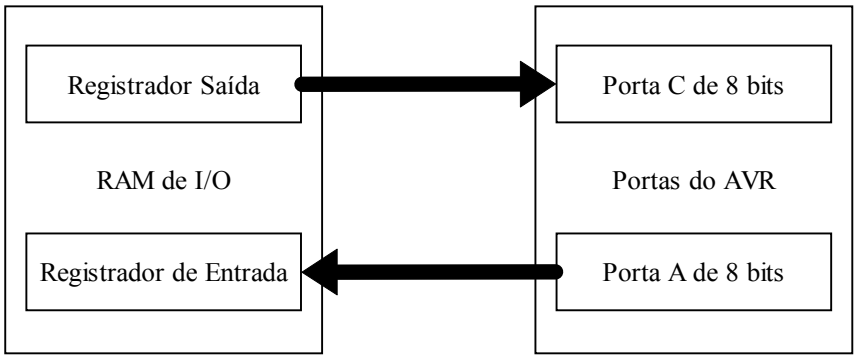


Figura 5.19: Estrutura dos registradores do protocolo MDH

Para funcionamento do protocolo MDH são necessários três critérios: O primeiro é a elaboração de comandos para intercomunicação como descritos na tabela 5.3. O segundo é a divisão do byte de comunicação entre comandos e dados, ou seja, o *nibble* alto pertence aos comandos e o *nibble* baixo pertence aos dados. E no terceiro, o AVR é tratado como um escravo e não tem permissão de requisitar transferências de dados para o Core PCI.

Comandos	**Ações**
0x	AVR ou Software pronto
1x	AVR ocupado
2x	AVR terminou a tarefa
3x	AVR captura o *nibble* baixo do endereço na porta C
4x	AVR captura o *nibble* alto do endereço na porta C
5x	AVR atualiza porta A como o valor de *nibble* baixo do registrador apontado pelo endereço
6x	AVR atualiza porta A como o valor de *nibble* alto do registrador apontado pelo endereço
7x	AVR captura na porta C o *nibble* baixo do valor a ser gravado
8x	AVR captura na porta C o *nibble* alto do valor a ser gravado
9x	AVR atualiza o registrador apontado pelo endereço com o valor capturado
Bx	AVR como dados de *nibble* alto do valor prontos na porta A
Cx	AVR como dados de *nibble* baixo do valor prontos na porta A

Tabela 5.3: Comandos do protocolo MDH

A aplicação do *firmware* no AVR é dedicada à manutenção de data e hora. Contudo no AVR foram reservadas posições de memória com a finalidade de fazer uma interface de troca de dados entre o Core PCI com o relógio I2C (como mostrado na tabela 5.4). Por exemplo: Se o software necessita ler os minutos do relógio, então o *firmware* atualiza a posição de memória de endereço 1 e sinaliza ao software que esta posição está atualizada, e logo em seguida o software faz a leitura desta posição. E para escrita do minuto o procedimento é o inverso.

Endereço memória	Descrição
0	Segundos
1	Minutos
2	Horas
3	Dia da semana
4	Dia do mês
5	Mês
6	Ano

Tabela 5.4: Posições de memórias no *firmware* para o protocolo MDH

Para o *firmware* acessar as posições de memória, o protocolo de comunicação precisa ser composto de dois registradores, um registrador para capturar o endereço de acesso e outro registrador para guardar temporariamente o valor a ser gravado na memória (fluxograma da figura 5.22).

Os fluxogramas mostram o funcionamento do protocolo MDH entre o software e o *firmware*. O fluxograma da figura 5.20 mostra como o protocolo grava um valor no registrador de memória no *firmware*. A figura 5.21 mostra o fluxograma do software requerendo dados de uma posição de memória no *firmware*. Note que em todos os procedimentos há uma chamada da função de sincronismo, se os pedidos dos comandos forem nos tempos certos, o microcontrolador AVR será bem mais lento que o barramento PCI. (O fluxograma do sincronismo é mostrado na figura 5.23). (O protocolo no *firmware* é mostrado na figura 5.22). Os acessos feitos ao relógio são feitos por intermédio de outro protocolo no barramento I2C. Embora isto não seja abordado neste trabalho. Orientamos àqueles que desejam saber mais, consultar as revistas Saber eletrônica exemplar 338 março de 2001 e exemplar 330 julho de 2000.

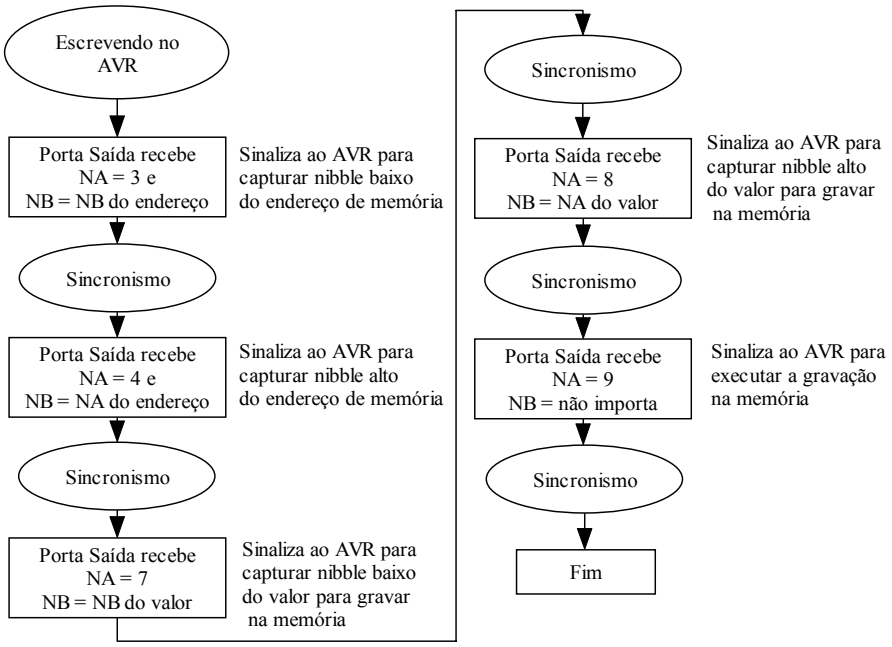


Figura 5.20: Fluxograma de gravação de dados no *firmware* através do protocolo MDH

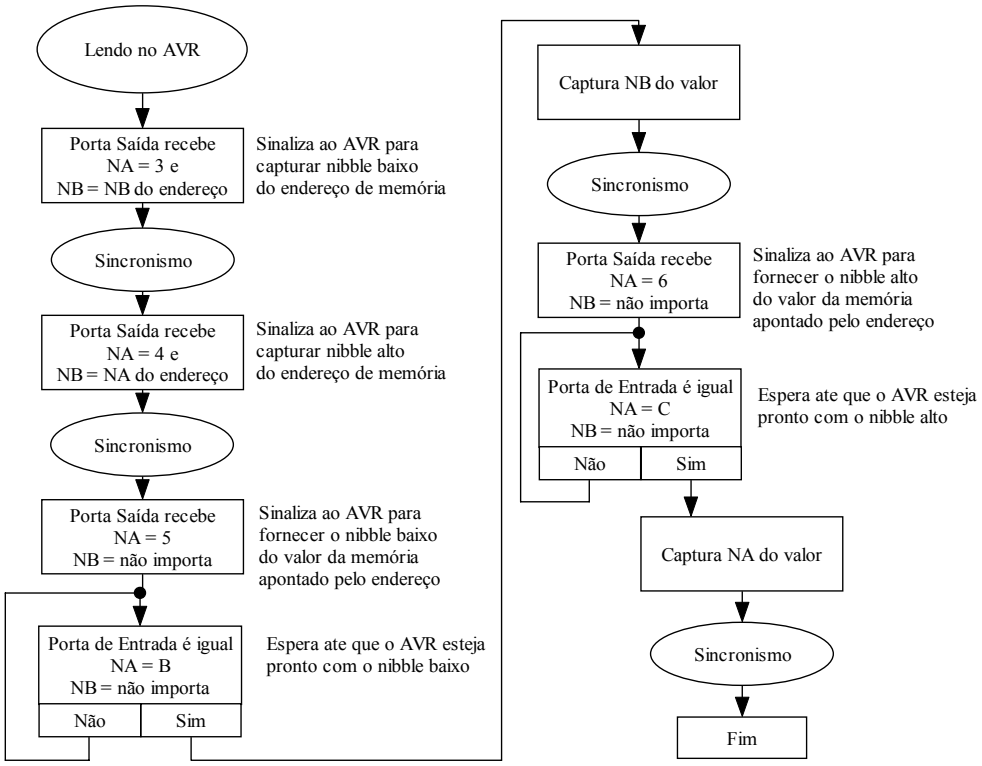


Figura 5.21: Fluxograma de leitura de dados no *firmware* através do protocolo MDH

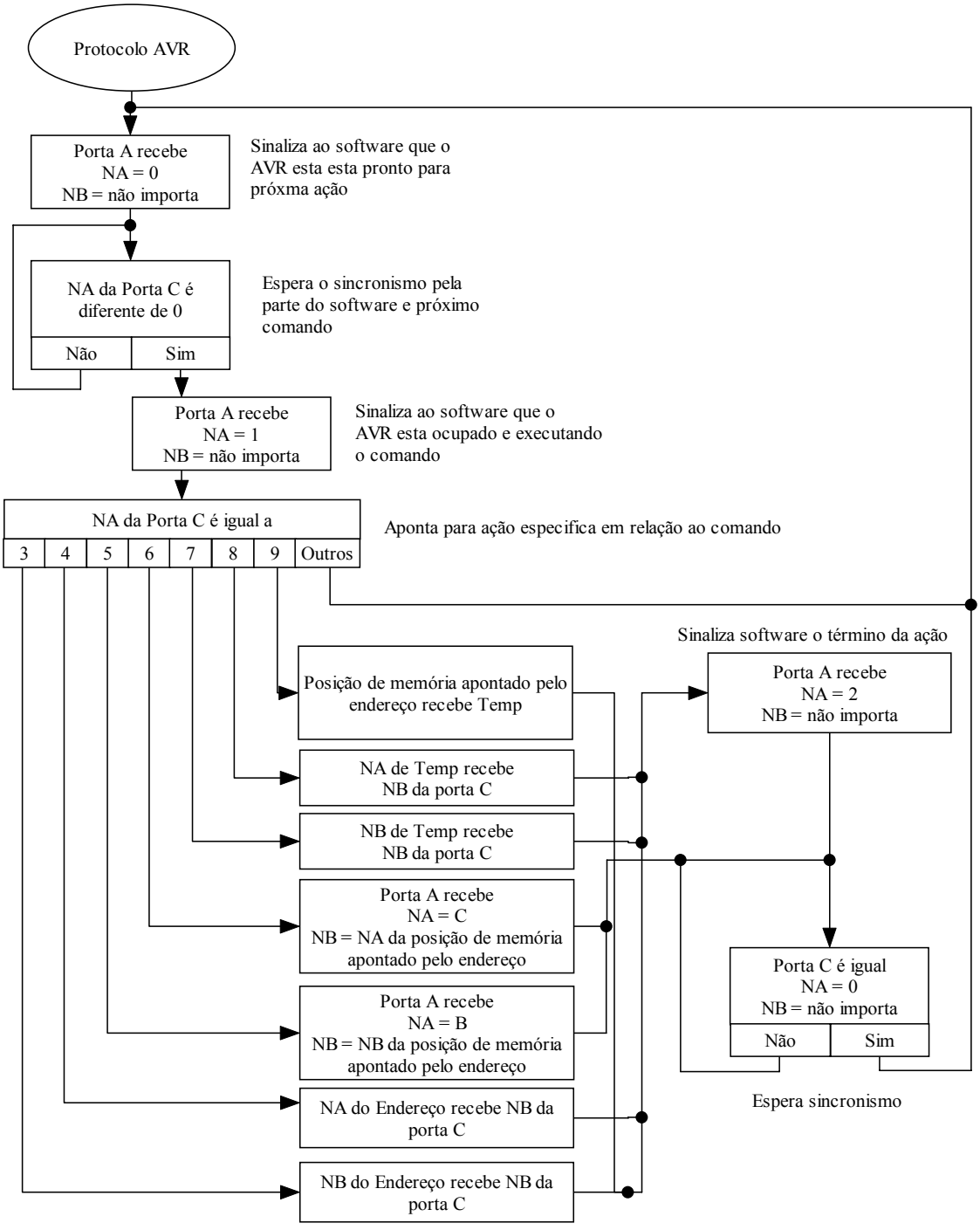


Figura 5.22: Fluxograma do protocolo MDH de leitura e escrita no *firmware*

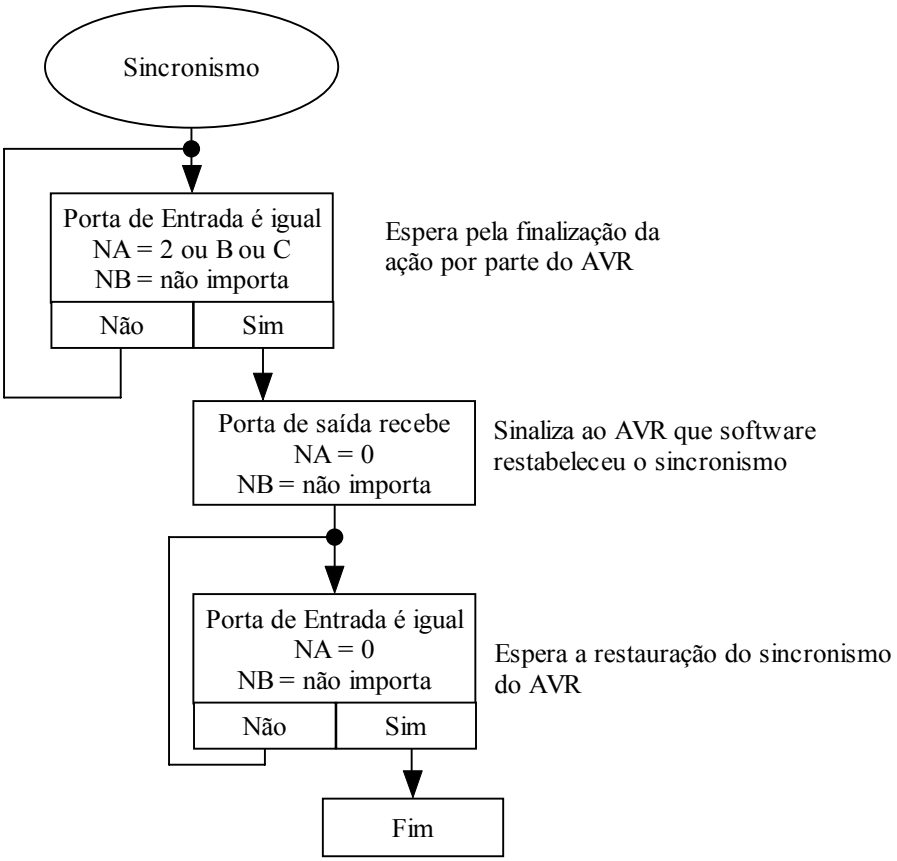


Figura 5.23: Fluxograma de sincronismo do protocolo MDH no software

5.10 Executando o Projeto

5.10.1 Adaptações nos Computadores para Teste do Protótipo

No desenvolvimento e testes foram utilizados dois microcomputadores (figura 5.24). O computador da esquerda teve a função de desenvolver o Core PCI, depurar a comunicação via porta serial e atualizar o Core PCI no FPGA. O micro da direita, quase imperceptível, teve a função de testar e funcionamento do protótipo. O computador destinado aos testes do protótipo e demais dispositivos são *onboards,* deixando os *slots* PCI's livres para que não entre em conflito com o protótipo, caso algo esteja errado. Facilita, assim, a detecção do erro, por este procedimento não afetar o funcionamento vital do computador. Outras duas adaptações foram feitas no computador para elaboração do protótipo. Uma delas foi mudar o tipo de botão de *reset* de aperte e solta para liga e desliga, com a finalidade de se manter o computador em *reset* no momento da configuração do FPGA. A segunda foi a montagem do

computador, eliminou-se o gabinete para que o hardware ficasse o mais exposto possível, facilitando as medições e monitoramento dos sinais com os instrumentos apropriados.

Figura 5.24: Ambiente de desenvolvimento do protótipo

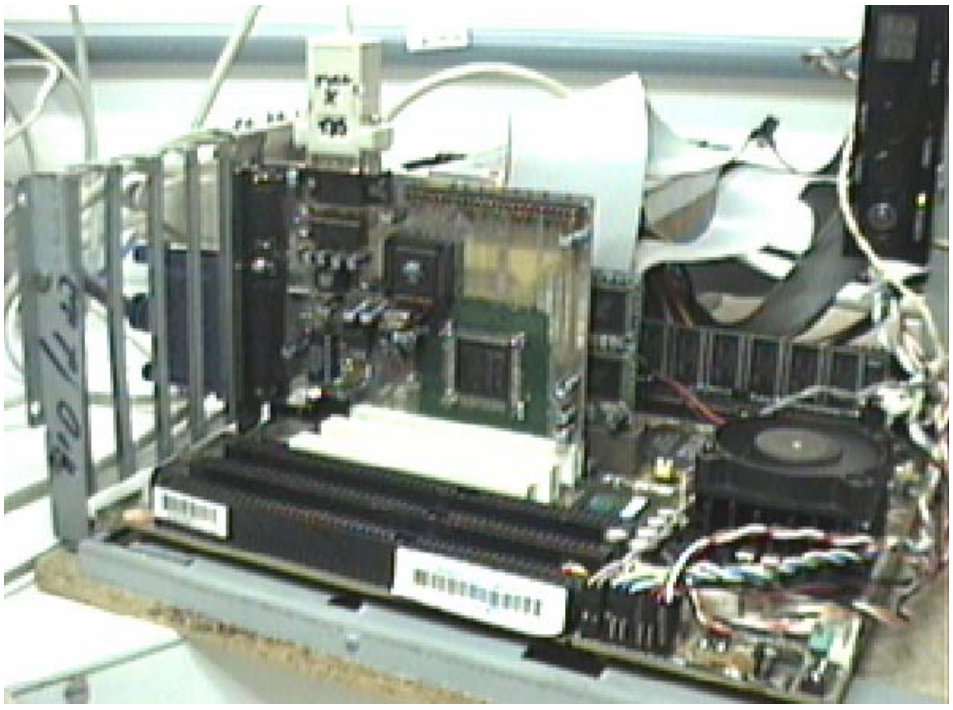


Figura 5.25: Microcomputador adaptado para comportar o hardware do Core PCI

5.10.2 Gravando o Core PCI no FPGA

A figura 5.26 mostra o software de configuração do FPGA incluso na própria ferramenta WebPack. Porém antes de se atualizar o Core no FPGA, o computador na qual contém o protótipo deve ser mantido em *reset*.

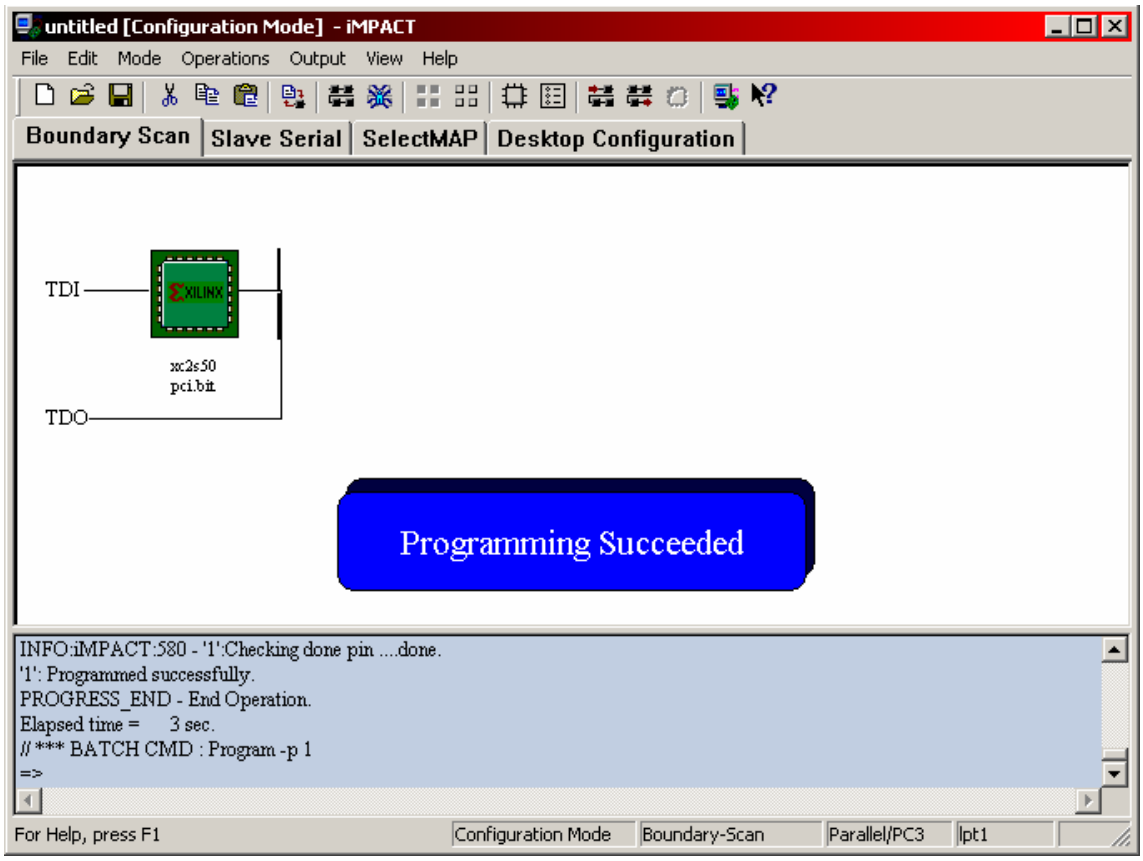


Figura 5.26: Software para configuração do FPGA

5.10.3 Testando o Core PCI

Agora com o *firmware* já gravado no AVR e com o Core PCI já configurado no FPGA, basta, somente, retirar o microcomputador do *reset* para inicialização. Após o *boot* do computador, a BIOS e o sistema operacional reconhece o novo dispositivo e começa o procedimento para instalação do *driver* (figura 5.27). Contudo, neste dispositivo não foi programado um *driver* para o Windows, porque os acessos são feitos via DOS, e basta somente a BIOS reconhecer o dispositivo. Neste caso, nenhum *driver* será instalado e o Windows o classificará como dispositivo desconhecido. Com o kit da Microsoft DDK (Driver Development Kit – Kit de Desenvolvimento de Drivers) se pode programar os *drivers* para Windows.

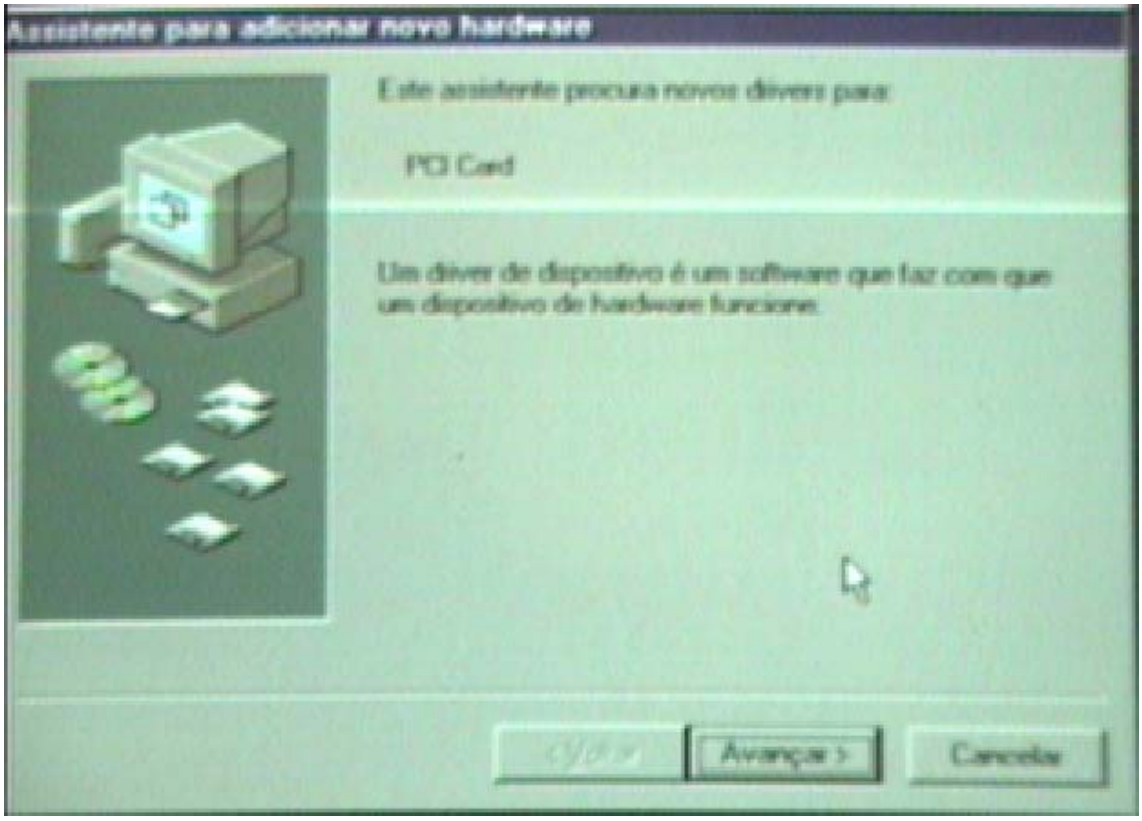


Figura 5.27: Reconhecimento do Core PCI pelo Windows

Após a inicialização do computador e do sistema operacional, cabe executar o software de aplicação para acesso ao relógio. A figura 5.28 mostra o software acessando o relógio, e na figura 5.29 o programa Mttty monitora o funcionamento da comunicação entre o software e *firmware*, isto é feito através da porta serial do microcontrolador AVR.

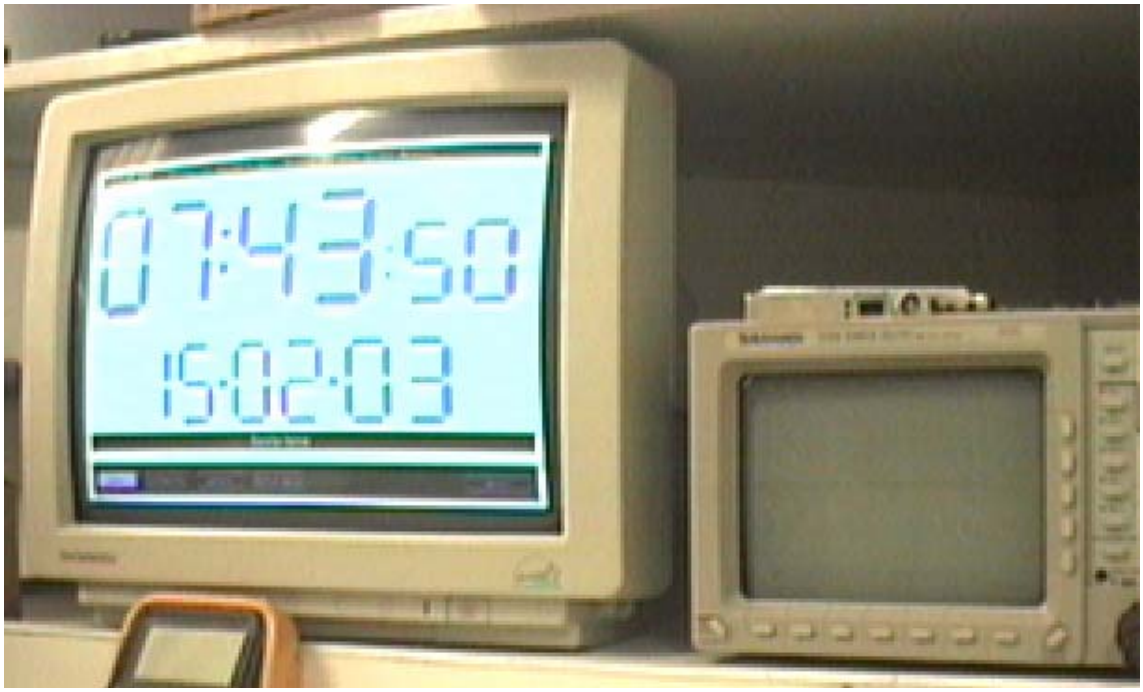


Figura 5.28: Aplicação está acessando o relógio I2C

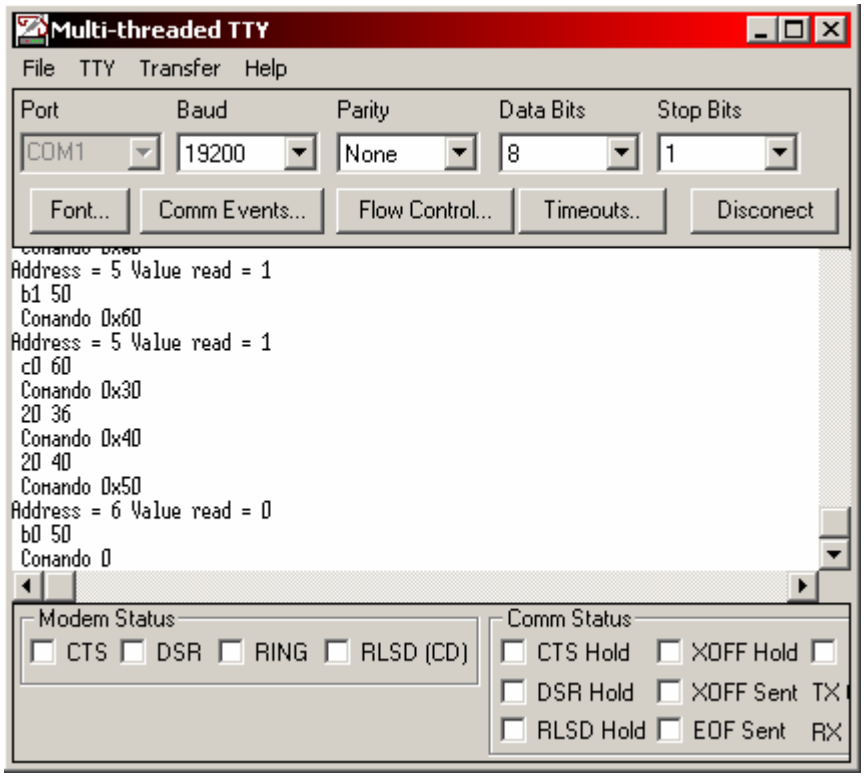


Figura 5.29: Monitoramento, via porta serial, da comunicação entre software e *firmware*

6 Conclusão

Grande parte das tecnologias nasce de uma necessidade. Esse trabalho não foi diferente. Com estudo e considerável desgaste, foi possível desenvolver o Core PCI em VHDL e suas simulações. A necessidade de controlar alguns dispositivos externos do computador e a necessidade de dominar a técnica do barramento PCI para projetos futuros, foram uns dos fatores primordiais para o desenvolvimento desse trabalho. O projeto aqui presente contém informações suficientes para as exigências das necessidades constatadas. Não menos importante, ele é o primeiro passo para projetos mais complexos.

A maior dificuldade encontrada no projeto foi alcançar a velocidade ideal dos circuitos internos do FPGA. O Core PCI continha controle de paridade e interrupção, foi preciso eliminá-los para que o projeto fosse o rápido suficiente para se manter funcional e dentro dos parâmetros determinados pela *PCI SI*. Assim para se construir Cores PCI's mais rápidos e mais complexos, deve se intensificar os estudos acerca das ferramentas Floorpanner e Timing Analyser. Para esse melhor performance aconselha-se, também, o uso de equipamentos mais sofisticados como o analisador lógico e osciloscópio digital. No intuito de que se obtenha melhores resultados, sugere-se a utilização, também, de outros tipos de FPGA's com suas respectivas ferramentas, tais como: FPGA da Altera com seu ambiente de desenvolvimento MaxPlus ou FPGA da Actel com seu ambiente de desenvolvimento Active-HDL.

Esse trabalho contém informações suficientes para que qualquer estudioso da área possa iniciar seu próprio Core PCI, seja ele simples ou complexo, de acordo com a necessidade de aplicação. O Core PCI aqui desenvolvido apresenta resultados satisfatórios em relação ao uso de dispositivos externos de acesso simples, tal como a leitura de sensores ou acionamento de atuadores. Portanto, qualquer, dispositivo que se possa controlar por intermédio de portas lógicas de entrada e saída contemplados nesse trabalho. Quanto a futuros desenvolvimento, pode-se pensar em diversas possibilidades de novas aplicações usando o Core PCI. Portanto, sugere-se a modificar o comportamento do Core PCI que possa incluir os controles de paridade e de interrupção, e obter um melhor desempenho em relação à velocidade.

Glossário

boot Indicada procedimento de inicialização do computador pela BIOS e sistema operacional.

broadcast Única transação feita pelo iniciador para todos os dispositivos do barramento local.

chipsets São chips como vários tipos de funções agregados num só encapsulamento. Muito utilizados em placas mães dos computadores. Estas funções são controladores de discos, controladores em acesso a memórias dinâmicas principais, controladores de acesso a barramentos, controladores de som e vídeo, entre outros. Estes se encarregam de fazer a parte sistemática do funcionamento do microcomputador deixando o processador livre para outras funções. Por serem chips pequenos, contribuem, tanto no custo, como no desempenho do computador, e proporcionam uma alta velocidade de trabalho.

drivers Os softwares que são responsáveis em acessar e controlar os dispositivos são classificados de drivers pelo sistema operacional.

dword Conjunto de 32 bits.

EEprom (Erase Electric Programam ROM – ROM que pode ser escrito e apagadas eletricamente). Componente eletrônico com função de ROM, que pode ser lido e gravado em qualquer momento diretamente pela CPU. Uma RAM que não perde os dados ao desligar o circuito.

Flash EEprom Uma EEprom com características bem mais rápidas que outras EEproms

firmware São softwares para sistemas embutidos. Por exemplo: Celulares, microondas, vídeo cassete, entre outras.

nibble Conjunto de 4 bits.

onboard São dispositivos inclusos na própria arquitetura computacional.

PCI SIG (PCI - Special Interest Group). Grupos que ditam as normas do barramento PCI.

pull down São pinos dos componentes que são forçados a assumir um nível lógico baixo por intermédio de algum componente externo.

pull up São pinos dos componentes que são forçados a assumir um nível lógico alto por intermédio de algum componente externo.

reset Indica que qualquer hardware entrará de forma forçada no estado inicial de funcionamento.

slot Conector para dispositivos de qualquer plataforma computacional.

Wire ORed Técnica usada em uma linha de sinal com o objetivo de unir varias outras linhas de mesma função para o acionamento simultâneo do sinal.

word Conjunto de 16 bits.

Referências Bibliográficas

[PLD 2001] PLD Applications – **PCI Core User's guide**, March 2001 http://www.plda.com.

[PCI 1998] PCISIG. **PCI local BUS Specifications**, Revision 2.2 , December 1998

[Moraes 2001] Moraes, Fernando; Calazans, Ney. **Apresentação da linguagem VHDL**. 31- Outubro 2001

[Finkelstein 1997] Finkelstein, Ehud. **Design And implementation of PCI BUS Based Systems**. Tese de dissertação de mestrado. Universidade Tel Aviv. Engenharia Elétrica Outubro de 1997.

[Anderson 1999] Anderson, Dom; Shanley, Tom. **PCI System Architecture Fourth Edition.** PC System Architecture Series – Addison-Wesley, 1999

[Abbott 2000] Abbott, Doug. **PCI Bus Demystified**. LLH Technology Publishing Eagle Rock, Virginia - 2000.

[Ewerton 2001] Ewerton Artur Cappelatti, **Implementação do Padrão de Barramento PCI para Interação Hardware / Software em Dispositivos Reconfiguráveis**. Dissertação de Mestrado. PUC. Faculdade de Informática, Porto Alegre, Junho de 2001

[Einsfeldt 2002] Einsfeldt, Augusto. Saiba Como Construir um Chip Dedicado Utilizando Lógica Programável VHDL. Saber Eletrônica. São Paulo. Editora Saber Ltda. Junho de 2002.Nº 353

[Mendonça 2002] Mendonça, Alexandre; Zelenovsky, Ricardo. **PC: um Guia Prático de Hardware e interfaceamento**. MZ Editora. 3. Ed, 2002.

[1] AMMC, **PCI Products Data Book**. http://www.amcc.com

[2] Xilinx. **ModelSim Tutorial**. Agosto de 2001.

[3] Xilinx. **XST User Guide**

[4] Mento Graphics. **Using the Xilinx Floorplanner to improve Results**. Application Note

[5] Ashenden. Peter. **The Stundent's Guide to VHDL**. Morgan Kaufmann Publishers, 1998.

[6] Sjoholm, Stefan; Lindth, Lennart. VHDL For Designers. Prentice Hall, 1997.

Anexos

Abaixo, encontram-se regras levantadas para o auxilio da construção dos comportamentos do Core PCI em VHDL e simulação.

1 Regras de Comportamento dos Sinais do Barramento PCI

Para descrever o comportamento do Core PCI é interessante determinar algumas regras de funcionamento dos sinais de acordo como o protocolo PCI.

Regra 1

Os sinais #RST e #INT[x] (x = A, B, C ou D) não são classificados e nem sincronizados pelo clock do barramento. No estado OCIOSO do barramento. Os sinais #FRAME, #IRDY, #TRDY, #DEVSEL, #STOP, #SERR, #PERR estão fracamente em nível alto por um resistor *pull up.* Os sinais AD[31:0], #CBE[3:0] e IDSEL são indeterminados.

Regra 2

Os sinais que devem estar estáveis na borda de subida do clock após o *reset* são: #IRDY, #TRDY, #FRAME, #DESVEL, #STOP, #PERR e #SERR (Somente na borda de descida do clock). #RST e #INT[x] (x = A ou B ou C ou D) não são classificador e não são sincronizados pelo clock.

2 Regras na Fase de Endereço e no Inicio de uma Transação

Nesta fase, os sinais #IRDY, #TRDY, #DEVSEL e #STOP estão em modo de inversão de barramento, que dura um ciclo de clock.

Regra 3

Iniciador começa uma transação acionando o sinal #FRAME.

Regra 4

Os sinais AD[31:0] contêm o endereço do alvo estáveis na borda de subida do clock e não muda até que seja completa a fase atual, em outras ocasiões, é indeterminado, com exceção da fase de dados.

Regra 5

Os sinais #CBE[3:0] contêm o comando estável do tipo de transação na borda de subida do clock e não muda até que seja completa a fase atual. Em outras circunstancias é indeterminado, com exceção da fase de dados. Os valores dos sinais #CBE[3:0] são controlados pelo iniciador, via comandos de software. Por exemplo: Ler ou escrever no dispositivo alvo.

Regra 6

Um dispositivo somente será alvo da transação, se o endereço mandado pelo iniciador coincidir com um dos registradores de configuração BAR[x] (x = 0, 1, 2, 3, 4 ou 5) do alvo. Ou se o sinal IDSEL do dispositivo estiver ativado na borda de subida do clock e o tipo de transação for zero (sinais AD[1:0] igual a "00") . Em outras situações, o sinal IDSEL é indeterminado.

3 Regras nas Fases de Dados

Regra 7

Um dispositivo quando é o alvo de uma transação o mesmo aciona o sinal #DEVSEL, isto não deve ocorrer depois da ativação de uns dos sinais #TRDY e #STOP. Depende como o alvo é classificado, seja como rápido, médio ou lento a sua ativação ocorre respectivamente em um clock, dois clocks ou três clocks após fase de endereço. Uma vez que o sinal #DEVSEL é acionado, não muda até fase atual seja completa. No quarto clock se nenhum alvo responder, o iniciador abortará a transação.

Regra 8

Os sinais #CBE[3:0] contêm os bytes habilitados dos sinais AD[31:0] estáveis na borda de subida do clock e serão permanentes para toda a transação independente do estados de espera do iniciador (#IRDY desativado). Em outras circunstancias, é indeterminado, com exceção da fase de endereço. O valor dos sinais #CBE[3:0] é controlado pelo iniciador pelos comandos via software. Ex: ler ou escrever um byte, ou uma *word*, ou uma *dword* no dispositivo alvo.

A fase de dados indica quais os bytes são válidos na transação. Normalmente permanecem sem mudanças por toda a transação. Mas o iniciador pode mudar desde que seja no inicio da fase e permaneça até que a fase esteja completa. O iniciador é livre em aderir para

#CBE ser ou não ser continuo, ou seja, não aciona nenhum dos sinais #CBE quando deseja pular algum endereço em uma transação em rajadas de dados, isto é muito útil na transação de escrita.

Regra 9

Nos sinais AD[31:0] os dados são válidos e estável na borda de subida do clock e não muda até que seja completa a fase atual. Em outras ocasiões é indeterminado com exceção da faze de endereço.

Regra 10

Uma fase é dita completa, quando estiver na borda de subida do clock e quando os sinais #IRDY e #TRDY ou #IRDY e #STOP estiverem acionados juntos na mesma fase. Neste momento é concluída a transferência de dados entre iniciador e alvo.

Regra 11

A última fase completa de dados é feita quando:

- O sinal #FRAME é desativado e o sinal #TRDY é acionado (modo normal);
- O sinal #FRAME é desativado e o sinal #STOP é acionado (Alvo terminando com "tente novamente" (retry) ou "desconectando" (disconnect));
- O sinal #FRAME é desativado e o tempo do alvo é extrapolado (iniciador aborta (abort));
- O sinal #DEVSEL é desativado e o sinal #STOP é ativado (Alvo aborta (abort)).

Regra 12

O alvo não tem permissão de mudar de ação uma vez que se comprometeu a completar a fase atual. O comprometimento para completar a fase é quando alvo aciona o sinal #TRDY ou o sinal #STOP. O alvo se compromete em:

- Modo de rajada (burst) que aciona o sinal #TRDY e não aciona o sinal #STOP;
- Desconectando (disconnect) por acionando ambos sinais #STOP e #TRDY;
- Nova tentativa (Retry) por acionando o sinal #STOP e desativando o sinal #TRDY.

Regra 13

Quando o sinal #IRDY ou o sinal #TRDY é desativado o barramento entre em estado de espera (wait cycles).

4 Regras para Finalização de uma Transação

A transação pode ser finalizada, tanto pelo iniciador, como pelo alvo. As transações são concluídas quando #FRAME e #IRDY são desativados e o barramento entra em modo OCIOSO (idle).

4.1 Regras para o Iniciador

O sinal #FRAME e seu sinal correspondente #IRDY definem o estado do barramento. Quando ambos são ativados, indica que o barramento está OCUPADO (busy) e quando ambos estão desativados, o barramento está OCIOSO (idle).

Regra 14

O sinal #FRAME não pode ser desativado, a menos que o sinal #IRDY seja ativado (O sinal #IRDY deve ser sempre ativado na primeira borda do clock que o sinal #FRAME é desativado).

Regra 15

O sinal #FRAME é desativado na penúltima fase de dados completa. Uma vez desativado, o mesmo não pode ser reativado na mesma transação.

Regra 16

Uma vez que o iniciador aciona o sinal #IRDY, o iniciador não pode mudar os estados dos sinais #IRDY e #FRAME até que a fase de dados atual seja completa.

Regra 17

O iniciador deve desativar o sinal #IRDY um clock após da última fase de dados completa e no clock seguinte levá-lo a alta impedância.

Regra 18

Quando os sinais #FRAME e #IRDY são desativados é o fim da transação.

4.2 Regras para o Alvo

Regra 19

Quando o alvo é incapaz de completar o pedido de transação, o mesmo pode finalizar acionado o sinal #STOP. O sinal #STOP não pode ser ativado no estado de INVERSÃO do barramento (turnaround) e nem antes do acionamento do sinal #DEVSEL. Uma vez que o alvo aciona o sinal #STOP, deve mantê-lo acionado até que o sinal #FRAME seja desativado pelo iniciador. O acionamento do sinal #STOP, em conjunto com outros sinais, determina o tipo de finalização, eis os tipos:

- Tipo “tente novamente” (retry) é feito acionado o sinal #STOP e desativado o sinal #TRDY no inicio da fase de dados. Este procedimento termina a transação antes da transferência de dados, pois o alvo está ocupado ou temporariamente incapaz para transação por algum motivo. Quando um alvo terminar com retry o iniciador deve incondicionalmente repetir a mesma transação até que seja completa. Isto não acontece com os outros tipos de finalizações;
- Tipo “desconectando com transferência de dados” (Disconect) é feito acionado os sinais #STOP e #TRDY juntos. Este procedimento termina a transação onde o alvo é incapaz de responder dentro do tempo de latência requerido, ou não suporta, ou temporariamente não suporta uma transferência em modo de rajada (burst);
- Tipo “desconectando sem transferência de dados” (Disconect) este funciona idêntico ao retry, só que é usado para finalizar uma transação em rajada após a primeira fase de dados;
- Tipo “Alvo aborta sem transferência de dados” (Abort) é feito acionado o sinal #STOP e desativando o sinal #DESVEL ao mesmo tempo. Este procedimento termina quando uma transação anormal ocorre originando um erro fatal ou alvo nunca é capaz de completar o pedido de transação. Por exemplo, iniciador quer ler uma *dword* no alvo em um registrador em byte de configuração.

Regra 20

Na última fase de dados se já não foram desativados os sinais #TRDY, #STOP e #DEVSEL devem ser desativados no clock seguinte e no próximo clock irem a alta impedância.

5 Regras para Geração de Paridade

O sinal PAR não é um item opcional e é sempre gerado por todos os dispositivos PCI. O sinal PAR indica se o número de ocorrências de bits acionados nos sinais AD[31:0] e #CBE[3:0] estão em pares, ou seja, quando o sinal PAR está ligado que dizer que a ocorrência de bits ligados é par. O cálculo do valor do sinal PAR não leva em consideração se os sinais AD[31:0] ou #CBE[3:0] contém dados úteis ou não. Por exemplo, mesmo que na fase de endereço na transação de reconhecimento de interrupção os sinais AD[31:0] são aleatórios, os mesmo entram para o cálculo de paridade.

Regra 21

O sinal PAR deve estar estável na borda de subida do clock nas seguintes situações:

- O sinal PAR é válido no sentido de iniciador para o alvo, após um clock da fase completa de endereço, ou na fase de dados após um clock quando acionado #IRDY na transação de escrita;
- O sinal PAR é válido no sentido de alvo para iniciador na fase de dados após um clock quando foi acionado #TRDY na transação de leitura.

Regra 22

Uma vez que o PAR é válido o mesmo permanece até um clock após a fase de dados ser completa, levando em consideração o atraso de um clock.

6 Regras para Checagem de Paridade

A checagem de paridade é feita por todos os dispositivos PCI, com exceção de:

- Dispositivos exclusivos na placa mãe, tal como *chipset*;

- Dispositivos com dados temporariamente e não critico a erro, tais como placa de vídeo ou de som.

6.1 Regras para Erro de Paridade na Fase de Endereço

O sinal #SERR é usado para sinalizar erro de paridade dos sinais AD[31:0] e #CBE[3:0] na fase de endereço. Este é requerido por todos os dispositivos PCI com exceção daqueles que não necessitam checagem de paridade. Neste sinal outros dispositivos podem ser acionados simultaneamente por causa da técnica de lógica OU (*wire ORed*).

Regra 23

Quando existir um erro de paridade e se o bit 8 do registrador de configuração de comando (bit #SERR enable) estiver acionado e o bit 6 do registrador command (bit Parity Error Response) estiver acionado, o dispositivo deve acionar o sinal #SERR.

Regra 24

Se um dispositivo detecta um erro de paridade, e se o bit 6 do registrador command (bit Parity Error Response) estiver acionado e o decodificador de endereço indicar que o dispositivo é o alvo da transação, o alvo pode atuar em seguintes procedimentos:

- Aceitar a transação e ignorar o erro;
- Aceitar a transação e logo em seguida o alvo abortar;
- Não aceitando a transação em não ativando o sinal #DEVSEL e permitir que o iniciador aborte a transação.

Regra 25

O sinal #SERR é ativado em um clock e depois vai a alta impedância.

Regra 26

Se existir um erro de paridade independente do bit 6 e 8 do registrador de configuração de comando, o bit 15 do registrador de configuração de status (bit Detected Parity Error) deve ser acionado.

6.2 Regras para Erro de Paridade na Fase de Dados

O sinal #PERR é usado para sinalizar erro de paridade dos sinais AD[31:0] e #CBE[3:0] na fase de dados com exceção do ciclo de transação do ciclo de especial e não deve ser acionado antes do sinal #DEVSEL. Este é requerido por todos os dispositivos com exceção daqueles que não necessitam checagem de paridade. O iniciador usa estas informações para gravar a ocorrência do erro para o driver do dispositivo. No iniciador o sinal #PERR é de entrada de saída e no alvo é somente de saída, ou seja, no alvo somente se checa a paridade na transação de escrita.

Regra 27

Se o bit 6 do registrador de comando estiver acionado e o erro de paridade for detectado pelo iniciador em uma transação de leitura, o iniciador deve acionar o sinal #PERR dois clocks depois que a fase de dados estiver completa onde ocorreu o erro. Isto só tem efeitos por outros iniciadores ou pontes, e não tem efeito sobre um alvo, pois o *driver* refaz a transação. Se o iniciador ativar o sinal #PERR antes da completar a fase de dados de leitura o mesmo deve acionar o sinal #IRDY para completar a fase.

Regra 28

Se o bit 6 do registrador comando estiver acionado e o erro de paridade foi detectado pelo alvo em uma transação de escrita, o alvo deve acionar o sinal #PERR dois clocks depois que a fase de dados estiver completa onde ocorreu o erro. Se o alvo acionar #PERR antes de completar a fase de dados de escrita, o mesmo deve ativar #TRDY para completar a fase de dados.

Regra 29

Se o alvo insere um estado de espera durante transação de escrita, o alvo é permitido acionar o sinal #PERR dois clocks depois dos dados válidos (sinal #IRDY acionado), porém antes da transferência dos dados (sinal #TRDY acionado).

Regra 30

Se o iniciador insere um estado de espera durante transação de leitura, o iniciador é permitido acionar o sinal #PERR dois clocks depois dos dados válidos (sinal #TRDY acionado), porém antes da transferência dos dados (sinal #IRDY acionado).

Regra 31

Uma vez que o sinal #PERR é acionado, o mesmo deve permanecer até dois clock's seguido à fase completa (sinais #IRDY e #TRDY acionados) ou quando finalizar transação.

Regra 32

Se for configurado, o alvo deve terminar a transação antes de acionar o sinal #PERR, ou seja, havendo o erro, então finalizará a transação, pois o iniciador somente vai tomar conhecimento do erro dois clocks posteriores.

Regra 33

No final da transação o sinal #PERR deve ser desativado e no clock seguinte ir à alta impedância.

7 Regra para Tratamento de Erro

Somente o iniciador tem a responsabilidade de reportar o erro de paridade ao sistema operacional por intermédio do *driver* do dispositivo para um tratamento do erro.

8 Regra para o Core PCI em VHDL

Quando o dispositivo for alvo da transação os estados dos sinais #TRDY, #STOP, #PERR e #SERR devem ser diferentes de alta impedância e se o sinal #DEVSEL estiver desativado os sinais #TRDY, #STOP, #PERR e #SERR também ficarão desativados na fase de endereços até que o alvo responda pela transação.

8.1 Regra para o Core PCI nos Processos em VHDL

Os sinais, que serão atribuídos em VHDL nos processos "rising_edge" e "falling_edge", somente ocorrerão respectivamente no momento de subida ou descida do clock e não durante o clock alto ou baixo. Então se deve levar em consideração a estabilidade dos sinais que pertence a operação da atribuição para que não haja uma inconsistência. Com no exemplo nos processos 1 e 2. No processo 1, a atribuição do sinal A se dá quando o clock permanece em nível em baixo e a atribuição do sinal D em nível alto. No processo 2, os sinais

PA e PB são atribuídos no momento de descida do clock, mas existe uma certa inconsistência na atribuição do sinal PA, pois neste momento o sinal A está em mudança de estado, já é diferente da atribuição do sinal PB, pois o sinal D está estável devido que o sinal D só muda no nível alto do clock.

```
Processo 1                      Processo 2
    if clk = '0' then           if falling_edge(clk)
        A = B e C                   PA = A
    else                            PB = D
        D = A
```