DELFIM LUIZ TOROK

PROJETO VISANDO A PROTOTIPAÇÃO DO PROTOCOLO DE ACESSO AO MEIO EM REDES ETHERNET

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre.
Curso de Pós-Graduação em Ciência da Computação, Faculdade de Informática,
Pontifícia Universidade Católica do Rio Grande do Sul.
Orientador: Prof. Ney Dr. Laert Vilar Calazans

PORTO ALEGRE

AGOSTO DE 2001

Av. Ipiranga, 6681 - Prédio 16
CEP 90619-900 - Porto Alegre - RS - Brasil
http://www.pucrs.br/biblioteca

Fone: 0 (xx) 51 3320-3544
FAX: 0 (xx) 51 3320-3548
e-mail: bceadm@pucrs.br

AGRADECIMENTOS

Agradeço aos meus pais, Delfim e Angelina Torok, pelo constante apoio e incentivo, que me permitiram a conclusão de mais esta etapa.

Ao professor Ney Laert Vilar Calazans, pela incansável orientação, constante dedicação e valiosos ensinamentos, ao longo do desenvolvimento deste trabalho e de todo o curso, suportando pacientemente os problemas das versões "beta".

Ao professor Fernando Gehm Moraes pelo seu oportuno apoio, antes e durante todo o desenvolvimento deste projeto, fornecendo ensinamentos e técnicas práticas para solução dos problemas encontrados no caminho.

Ao amigo, colega e parceiro de mestrado Ewerton Artur Capelati, que de certa forma participou do desenvolvimento deste projeto, com desenvolvimento correlato do seu trabalho de conclusão, pelo companheirismo e amizade dedicados ao longo desta jornada.

Aos professores e funcionários da Faculdade de Informática – FACIN da PUCRS, suporte fundamental ao longo destes anos de preparação. Em especial aos alunos, bolsistas e colegas do Grupo de Apoio ao Projeto de Hardware - GAPH, sempre atenciosos e dispostos a ajudar na busca de informações.

Aos amigos e colegas de mestrado, que de uma forma ou de outra forneceram estímulos para que este trabalho fosse realizado.

E ao Centro Universitário FEEVALE em Novo Hamburgo. Especialmente ao então Diretor do Centro de Ciência da Computação, Dr professor Cleber Cristiano Prodanov, que na época possibilitou meu ingresso no programa qualificação do Corpo Docente, desenvolvido nesta Instituição, promovendo a parceria PUCRS/FEEVALE na realização do curso de Mestrado em Informática da Pontifícia Universidade Católica do Rio Grande do Sul e por acreditar sinceramente neste projeto.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ADCCP	Advanced Data Communication Control Proceedings
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
ANSI	American National Standards Institute
API	Application Program Interface (Hardware Object Technology)
SA	Source Address
ASP	Application Specific Processor
ATM	Asynchronous Transfer Mode
AUI	Attachment Unit Interface
BLE	Basic Logic Element
BRAM	Block Random Access Memory
BSC	Binary Synchronous Protocol (IBM)
CAM	Content Addressable Memory
CCITT	Comité Consultatit International Télégraphique et Téléphonique
CCITT	Consultative Committee for International Telegraph and Telephone
CCM	Configuration Cache Manager
CI	Circuito Integrado
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Check
CSI	Configurable System Interconnect Bus
CSMA-CD	Carrier Sense Multiple Acess with Collision Detection
DA	Destination Address
DDCMP	Digital Data Communications Message Protocol (DEC)
DEC	Digital Equipment Corporation
DEC PRL	Paris Research Lab Digital Equipment Corporation
DF	Data Field
Dip	Dual In Line Package
DISC	Dynamic Instruction-Set Computer
DIX	Digital Equipment Corporation/Intel Corporation/Xerox Corporation
DLL	Delay Locked Loop
DMA	Direct Memory Access
DNA	Desoxiribonucleic Acid
DPGA	Dynamically Programmable Gate Array
DQDB	Distributed Queue Dual Bus
EAB	Embedded Array Block
EEPROM	Electrically Erasable Programmable ROM
EIA	Electronic Industries Association

EPROM	Erasable Programmable ROM
ESB	Embedded System Block
FB	Function Block
FCS	Frame Check Sequence
FIFO	First-In-First-Out
FPGA	Field-Programmable Gate Array
FPLD	Field Programmable Logic Device
FSM	Finite State Machine
FTP	File Transfer Protocol
GPP	General Purpose Processor
GRM	General Routing Matrix
HDLC	High Level Data Link Control
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic Engineers
IOB	Input-Output Block
IP	Intellectual Property
IP Core	Intellectual Property Core
IPG	Inter Packet Gap
ISA	Industry Standard Architecture
ISO	International Standards Organization
ISP	Instruction Set Processor
JTAG	Joint Test Action Group
LAB	Logic Array Block
LAN	Local Area Network
LE	Logic Element
LED	Light Emitting Diode
LLC	Logical Link Control
LSB	Least Significant Bit
LUT	Look-Up Table
LVDS	Low Voltage Differential Signaling
MAC	Medium Access Control
MAN	Metropolitan Area Network
MIPS	Millions of Instructions Per Second
MOS	Metal Oxide Semiconductor
MPGA	Mask-Programmable Gate Array
NIC	Network Interface Controller/Card
NTSC	National Television Standards Committee
ONO	Oxygen-Nitrogen-Oxygen
OSI	Open Systems Interconnect
OSI-RM	Open Systems Interconnect Reference Model

PAL	Programmable Array Logic
PAM	Programmable Active Memory
PCI	Peripheral Component Interconnect
PLL	Phase Locked Loop
PLS	Physical Signaling
PRISM	Processor Reconfiguration through Instruction-Set Metamorphosis
PROM	Programmable Read Only Memory
PSM	Programmable Switch Matrix
RAM	Random Access Memory
RAMDAC	Random Access Memory Digital Analog Converter
RENCO	Reconfigurable Network Computer
RGB	Red-Green-Blue
ROM	Read Only Memory
RPC	Remote Procedure Call
SDLC	Synchronous Data Link Control
SECAM	Séquentielle Couleur Avec Mémoire
SERDES	Serializer/Deserializer
SFD	Start Frame Delimiter
SLI	System-Level Integration
SMTP	Simple Mail Transfer Protocol
SOC	System-On-a-Chip
SORC	System-on-a-Reconfigurable Chip
SRAM	Static Random Access Memory
SRC	Supercomputing Research Center
USB	Universal Serial Bus
Utopia	Universal Test & Operations PHY Interface for ATM
VCC	Virtual Computer Corporation
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration
VME	Versa Module Europa
VSB	VME Subsystem Bus
WAN	Wide Area Network
WWW	World Wide Web

RESUMO

Este trabalho descreve as estratégias de projeto e validação de um Núcleo de Propriedade Intelectual (IP Soft Core) destinado a desempenhar o papel de protocolo de Controle de Acesso ao Meio para redes locais do tipo Ethernet. O projeto deste IP Soft Core foi completamente implementado na linguagem VHDL, sendo assim flexível, portável e personalizável para aplicações específicas. A implementação foi especialmente desenvolvida para adaptar-se bem a ambientes onde o IP Core reside em dispositivos de hardware reconfigurável tais como FPGAs baseados em RAM. Também se descreve aqui a estratégia de validação funcional do projeto do IP Core Ethernet em detalhe. Finalmente, o trabalho introduz uma proposta para vários ambientes de validação, mediante uso de plataformas de prototipação rápida comerciais, visando testar e caracterizar completamente o IP Soft Core Ethernet em redes locais reais.

Palavras-chave - Redes de Computadores, redes locais, LAN, Ethernet, dispositivos reconfiguráveis, sistemas digitais, FPGAs, modelo de referência OSI, protocolo de acesso ao meio, MAC, IP Core, Soft Core.

ABSTRACT

This work describes the design and validation strategies for an Intellectual Property (IP) Soft Core deemed to fulfill the role of Medium Access Control protocol for Ethernet local area networks. The design of the IP Core was implemented in the VHDL language, being thus flexible, portable and customizable to specific applications. The implementation was specifically developed to fit in environments where the IP Core resides in reconfigurable hardware devices like RAM-based FPGAs. The functional validation strategy of the Ethernet IP Soft Core using commercial reconfigurable boards is described in detail. Finally, the work introduces a proposal for several validation environments, in order to fully test and characterize the Ethernet IP Soft Core in real world local networks.

Keywords – Computer networks, local networks, LAN, Ethernet, reconfigurable devices, digital systems, FPGAs, OSI reference model, medium access protocol, MAC, IP Core, Soft Core.

1. Introdução

Provavelmente, o grande avanço na primeira metade do século XX foi a introdução de sistemas computadores como um novo recurso para geração e troca de informações. O computador e a comunicação através de sinais elétricos, que na época alcançava considerável evolução, veio revolucionar o mundo em que vivemos. Da conjunção destas duas tecnologias surgiu uma nova forma de comunicação entre pessoas e até mesmo entre máquinas. Com esta nova forma de comunicação, nasce o conceito de *rede de computadores*, que atualmente é definido por um conjunto de módulos processadores capazes de trocar informações e compartilhar recursos interligados por um sistema de comunicações [SOA95].

Dentro deste novo contexto, a diversificação das atividades e recursos cresceu, passou-se a considerar o uso das redes de computadores como um meio para distribuir com maior eficácia a utilização informações espalhadas por todo planeta. Quando a distância entre os módulos processadores estão na faixa de metros até alguns poucos quilômetros, as redes de computadores são denominadas de *Redes Locais* (Local Area Networks – LANs). Redes locais surgiram, assim, para viabilizar a troca e o compartilhamento de informações e dispositivos periféricos, preservando a independência das várias estações ou módulos processadores, e permitindo a integração em ambientes de trabalho cooperativo [SOA95]. Quando a distância de ligação entre os vários módulos processadores atinge distâncias metropolitanas, as redes de computadores são denominadas de *Redes Metropolitanas* (Metropolitan Area Networks – MANs). Por outro lado, as *Redes de Longo Alcance* (Wide Area Networks – WANs) surgiram da necessidade de se compartilhar recursos especializados por uma maior quantidade de usuários geograficamente dispersos [SOA95]. Por terem um custo associado dos seus meios de comunicação bastante elevado, tais redes são em geral públicas, isto é, o sistema de comunicação é mantido, gerenciado e de propriedade de grandes operadoras. Como exemplo temos a rede mundial de computadores, a *Internet*, que interliga redes da maioria dos países, oferecendo uma série de recursos, tais como: o protocolo de transferência de arquivos (FTP), o correio eletrônico e as "páginas" da internet.

Diversas tecnologias para a implementação de redes locais foram desenvolvidas e/ou estão em desenvolvimento. Exemplos são as redes *ATM* (modo de transferência assíncrono), *Token Ring* (passagem de permissão em anel), *Token Bus* (passagem de permissão em barra) e *CSMA/CD* (acesso múltiplo por sensoriamento de portadora e detecção de colisões). Todas estas tecnologias constituem-se como protocolos de acesso ao meio ou *MAC* (em inglês, Medium Access Control).

Uma das tecnologias habilitadoras de redes de computadores são os protocolos de comunicação de equipamentos fisicamente conectados, tais como o protocolo *Ethernet* [MET76]. O protocolo Ethernet foi projetado para transferência de dados em alta velocidade, limitado a pequenas distâncias. Atualmente, a tecnologia Ethernet é aplicada nas redes locais, que são redes de computadores restritas a uma sala, um prédio ou uma corporação. Redes Ethernet não têm tamanho físico determinado, porém seu alcance é restrito ao local de operação. Assim, embora possam interligar, em tese, qualquer quantidade e computadores, normalmente um número limitado provê melhor desempenho na comunicação. Este protocolo surgiu no ambiente de institutos de pesquisa e universidades, com o objetivo de viabilizar a troca e o compartilhamento de informações e dispositivos periféricos (recursos de hardware e software), permitindo a integração em ambientes de trabalho cooperativo [SOA95]. O protocolo CSMA/CD é a base do protocolo Ethernet.

Observando-se as diferentes aplicações que utilizam redes locais, percebemos a grande importância dos *controladores* utilizados nestas redes. Atualmente, grande parte destes controladores são baseados na tecnologia Ethernet. Esta tecnologia tornou-se um padrão internacional e a partir deste

padrão foram desenvolvidos módulos de hardware em dispositivos eletrônicos específicos, para executarem as funções necessárias aos controladores de redes.

Há uma tendência no desenvolvimento de módulos de hardware reutilizáveis e portáveis a diferentes tecnologias de implementação, que permitam a rápida construção de sistemas digitais, denominados por *Núcleo de Propriedade Intelectual* reutilizável (do inglês, Intellectual Property core, ou IP core). Estes módulos de hardware reutilizáveis são desenvolvidos em *dispositivos reconfiguráveis*. Os dispositivos reconfiguráveis, conhecidos como *FPLDs* (Dispositivos Lógicos Programáveis no campo, em inglês, *Field Programmable Logic Devices* ou *FPLDs*), são componentes eletrônicos digitais cujo comportamento pode ser definido pelo usuário após a fabricação, num processo denominado *configuração*. Estes componentes habilitam a construção de sistemas digitais eletrônicos de uma forma mais flexível. Isto é devido à característica de reconfigurabilidade, que permite resolver, em alguns casos de forma mais eficiente, os problemas encontrados no desenvolvimento e implementação de um sistema digital.

1.1 Objetivos

Motivado pela crescente importância da tecnologia de redes locais e da rápida evolução de dispositivos com características de reconfigurabilidade, propõe-se aqui o projeto e a prototipação do protocolo de acesso ao meio em redes Ethernet, sob forma de um núcleo processador de propriedade intelectual reutilizável. Este núcleo processador executará parte do protocolo de comunicação em uma rede local de tecnologia Ethernet.

O presente trabalho teve como objetivo estratégico dominar a tecnologia de implementação de controladores de redes Ethernet, concentrando-se nos níveis inferiores de abstração (hardware e software básico mínimo). Ênfase foi colocada, como indicado no título do trabalho, no projeto e prototipação de protocolos de acesso ao meio em redes Ethernet usando como estudo de caso o MAC Ethernet. Para tal, os objetivos específicos alocados ao longo desenvolvimento deste controlador consistiram em:

1. Projetar e validar a parte fundamental de um controlador de rede local para o padrão Ethernet, endereçando procedimentos de software e hardware no nível de enlace do modelo OSI-RM[1];
2. Implementar um controlador baseado no módulo mencionado no item anterior sobre componentes reconfiguráveis em uma plataforma de prototipação escolhida, fazendo-a interagir com um computador hospedeiro e com uma rede local;
3. Disponibilizar o módulo resultante sob forma de um núcleo processador de propriedade intelectual do tipo Soft (IP Soft Core) reutilizável;
4. Prover subsídios para efetuar migrações entre software e hardware dos módulos em questão, com o intuito de maximizar desempenho e/ou minimizar custos dos sistemas digitais e respectivo software.

1.2 Motivação

As justificativas para empreender o presente trabalho, malgrado a disponibilidade de circuitos integrados (CIs) comerciais que desempenham a função de controle de acesso ao meio Ethernet, e até mesmo de outros trabalhos de pesquisa [FRA00] e desenvolvimentos comerciais já possuem disponíveis IP Soft Cores deste tipo, são múltiplas. Além dos benefícios acadêmicos de dominar

1 O modelo OSI-RM é apresentado em mais detalhes no Capítulo 4.

uma tecnologia importante na área estratégica de transmissão de dados, existem benefícios potenciais para empresas nacionais usuárias da tecnologia Ethernet. Para estas, o custo de CIs em geral, e controladores para Ethernet é relativamente alto. Além disso, os CIs comerciais são inflexíveis quanto às funções que podem desempenhar, ou seja, funções a mais não podem ser acrescentadas, exceto pela adição de outros CIs ao sistema, e funções desnecessárias presentes no CI não podem deste ser removidas. O desenvolvimento de um IP Soft Core para a camada MAC Ethernet remove a inflexibilidade dos CIs comerciais, permitindo adaptar o hardware ao subconjunto de funções estritamente necessárias no produto que o emprega, melhorando a relação custo benefício de produtos tecnológicos. Por outro lado, o acesso à tecnologia de implementação de IP cores Ethernet permite, por exemplo, liberar empresas da necessidade de importar parte dos insumos de seus produtos, aumentando a competitividade destes produtos num mercado globalizado.

1.3 A Estrutura do Volume

O presente trabalho está estruturado em três partes. A primeira parte, contituida dos Capítulos 2 e 3, mostra um estudo do estado da arte de dispositivos e arquiteturas reconfiguráveis, apresentando os conhecimentos necessários para a prototipação do IP Soft Core. Na segunda parte, que envolve os Capítulos 4 e 5, introduz-se o Modelo de Referência OSI-RM da ISO, e estuda-se redes locais Ethernet. Na continuação da segunda parte, o Capítulo 5 apresenta o projeto IP Soft Core para o MAC Ethernet, descrevendo sua estrutura e a funcionalidade dos módulos básicos. Na terceira parte, que consiste nos Capítulos 6 ,7 e 8, apresenta-se, no Capítulo 6, a estratégia de validação funcional do projeto IP Soft Core. No Capítulo 7, mostra-se detalhes do processo de prototipação e uma proposta de ambientes de teste sugeridos para prototipação do núcleo processador de propriedade intelectual. Finalmente, o Capítulo 8 apresenta algumas conclusões e sugere trabalhos futuros.

2. Hardware Reconfigurável

2.1 Introdução

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

No caso de implementações em software existe um hardware subjacente normalmente composto por um processador de conjunto de instruções (em inglês, *instruction set processor* ou ISP) associado a uma memória. ISPs podem ser programados para executar uma ou mais aplicações específicas preenchendo a memória de instruções com software que implementa as aplicações.

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

A arquitetura genérica de FPGAs, ilustrada pela Figura 2.1, consiste em uma matriz de elementos agrupados em *blocos lógicos* configuráveis, que podem ser interconectados, por *barramentos de interconexão* configuráveis. Semelhante a uma PAL (*Programmable Array Logic*), as interconexões entre os elementos são implementadas por *blocos de chaves* configuráveis pelo usuário. Através de *blocos de entrada/saída* configuráveis é realizado o interfaceamento com o mundo externo. Os FPGAs foram introduzidos em 1985 pela empresa Xilinx. Desde então, grande variedade de FPGAs foi desenvolvida por várias outras companhias, entre elas: Actel, Altera, Atmel, Plessey, Plus Logic, Advanced Micro Devices (AMD), Quicklogic, Algotronix, Concurrent Logic, e Crosspoint Solutions.

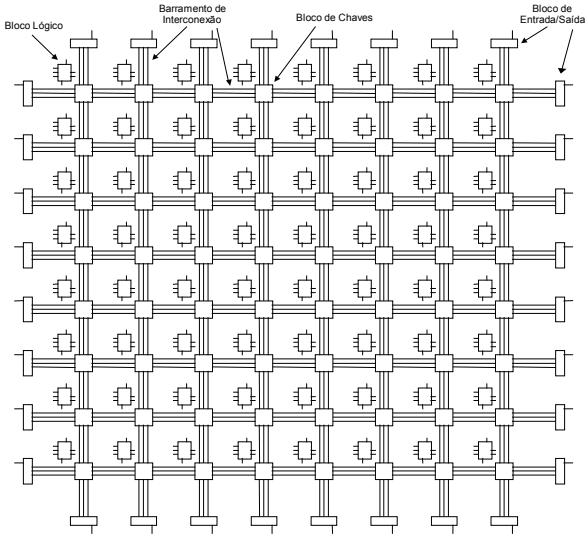


Figura 2.1 - Arquitetura de um FPGA genérico.

Nas Seções seguintes são apresentadas as formas de configurar FPGAs, as arquiteturas de blocos lógicos e de roteamento, bem como as noções de granularidade, densidade e desempenho de FPGAs. Apresenta-se também a definição de Núcleos de Propriedade Intelectual e uma visão do estado da arte em FPGAs.

2.2 Formas de Configurar FPGAs

Em um FPGA, a matriz de blocos lógicos é configurada e interconectada eletricamente através de chaves eletrônicas configuráveis. As propriedades destas chaves, tais como tamanho, resistência (em ohms), e capacitância (em farads), delimitam as características elétricas destes circuitos integrados. Nas três seções a seguir serão descritas as tecnologias de chaves mais comumente utilizadas [ROS93].

2.2.1 Tecnologia de Configuração SRAM

A tecnologia SRAM (*Static Random Access Memory*) usa bits de memória RAM estática para programar e controlar as chaves eletrônicas, baseadas em transistores de tecnologia CMOS (*Complementary Metal Oxide Semiconductor*) ou multiplexadores como ilustra a Figura 2.2.

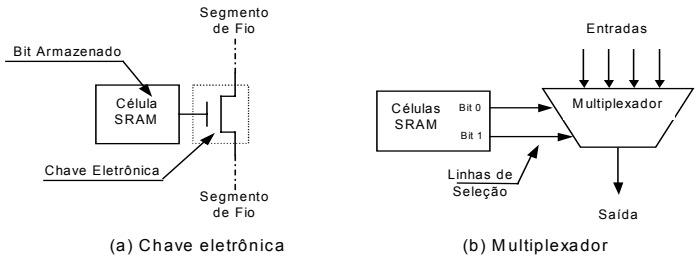


Figura 2.2 –Duas opções de tecnologia de configuração de FPGAs.

Quando um bit é armazenado na célula SRAM vista na Figura 2.2(a), a chave eletrônica funciona como um circuito aberto ou fechado, conforme o valor lógico ("0" ou "1") do bit armazenado. Desta forma a chave é usada para fazer uma conexão entre dois segmentos de fio. Quando um "0" é armazenado, a chave está aberta, e o transistor apresenta uma resistência que tende ao infinito, a qual impede a conexão entre os dois segmentos.

No caso de multiplexador, (Figura 2.2(b)), as saídas das células SRAM estão conectados às linhas de seleção, escolhendo uma das entradas do multiplexador para ser conectada à saída. Considerando que a SRAM é volátil, o FPGA deverá ser configurado a cada vez que for alimentado. Isto exige uma memória externa permanente como PROM, EPROM, EEPROM ou outro meio de armazenamento, para prover o vetor de configuração.

A tecnologia SRAM é usada em dispositivos da Xilinx [XIL99], Plessey [PLE89], Algotronix [ALG89], Altera [ALT01a], Concurrent Logic [CON91], Toshiba [MUR91] e Atmel [ATM99].

A necessidade de uma grande área na pastilha de silício que compõe o circuito integrado FPGA é a maior desvantagem da tecnologia de programação SRAM. Consome-se cinco transistores para implementar uma célula de SRAM e um transistor para a chave programável. Porém, esta tecnologia tem duas vantagens importantes, a reconfigurabilidade rápida e ilimitada devida às características das SRAMs e a possibilidade de utilizar um processo relativamente simples de fabricação do circuito integrado.

2.2.2 Tecnologia de Configuração Antifusível

Um *antifusível* é um dispositivo que apresenta uma resistência muito alta entre seus terminais. Quando uma tensão de 11 a 20 volts (dependendo do tipo de antifusível) é aplicada entre de seus terminais, o antifusível "queima," criando uma ligação de baixa de resistência (contrário ao fusível

convencional que se interrompe), tornando esta ligação permanente, conforme ilustrado na Figura 2.3.

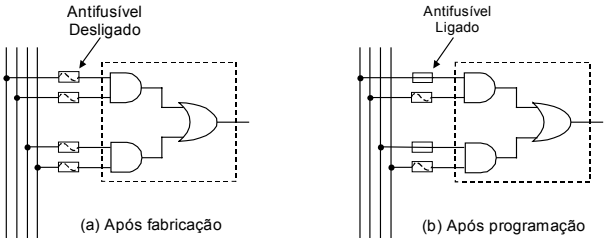


Figura 2.3 - Tecnologia de programação antifusível.

A programação de um antifusível requer um circuito extra para fornecer a tensão de configuração e uma corrente de 5 mA ou mais. Isto é uma desvantagem, pois exige transistores de maior potência para prover a queima de cada antifusível. Uma vantagem do antifusível é seu tamanho pequeno em relação a componentes de outras tecnologias de configuração. Outra é sua resistência série relativamente baixa e uma pequena capacitância parasita (para o antifusível não configurado), significativamente mais baixa que em outras tecnologias de configuração. Dois tipos de antifusíveis são mais utilizados, os fabricados pelo processo Oxigênio-Nitrogênio-Oxigênio (ONO) e o de Silício amorfo. A tecnologia de antifusível é usada nos dispositivos FPGAs da Actel [GAM89], Quicklogic [BIR91] e Crosspoint [MAR92].

2.2.3 Tecnologia de Configuração de Porta Flutuante

Esta tecnologia é baseada em configuração por armazenamento de cargas, da mesma forma que utilizada nas memórias EPROM (*Erasable Programmable ROM)*, EEPROM (*Electrically Erasable Programmable ROM*) e Flash RAMs. Cada bit da memória, conforme ilustra a Figura 2.4, possui um transistor MOS (*Metal Oxide Semiconductor***)** com duas portas*,* uma delas flutuante, não conectada ao barramento da memória (S_0) e isolada por material de alta impedância.

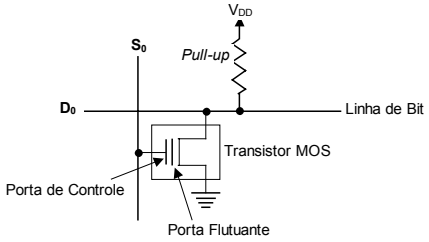


Figura 2.4 - Tecnologia de programação porta flutuante.

Em estado "desligado", como são fabricados, esses transistores não conduzem quando selecionados (via linha S_0) e o conteúdo das posições de memória (via linha D_0) é levado a "1" por resistores de *pull-up*. Para gravar um valor "0" em determinada posição, aplica-se uma tensão elevada, entre a porta de controle (não flutuante) e o dreno, o que causa uma ruptura no material isolante e permite o acúmulo de cargas na porta flutuante, as quais permanecem, mesmo após o término do pulso de tensão, devido à alta impedância do material isolante. A presença dessas cargas na porta flutuante mantém o transistor em condução quando a posição daquele bit for selecionada. Com isso, a linha de bit é levada para "0". Estas cargas podem ser removidas da porta flutuante expondo o componente à luz ultravioleta, desfazendo-se desta forma a configuração.

Como no caso de uso de SRAM, a vantagem principal da tecnologia de porta flutuante é sua reconfigurabilidade. Entretanto, esta tecnologia tem uma vantagem a mais, não é necessária memória externa permanente para programação do conjunto de bits de configuração no momento da

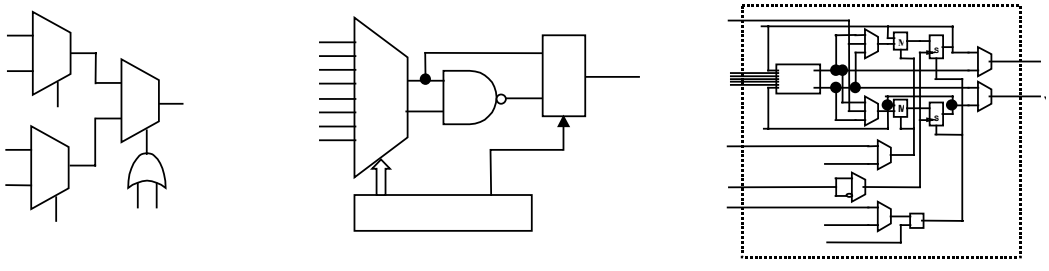
inicialização. Porém, a tecnologia de construção de dispositivos de porta flutuante requer três passos a mais do que o processo usual de fabricação CMOS. Duas outras desvantagens são: a alta resistência do transistor quando no estado de condução, e o alto consumo de energia devido ao resistor de *pull-up*, que é conectado à fonte de alimentação do dispositivo FPGA.

A tecnologia de porta flutuante baseada em EEPROM é usada em dispositivos fabricados pela AMD [AMD90], Lattice [BAK91] e Altera [WON89]. Esta tecnologia é semelhante a das EPROMs, a não ser pela remoção das cargas da *porta flutuante*, que pode ser feita eletricamente, no circuito, sem luz ultravioleta. Isto dá uma vantagem a mais, a fácil e mais rápida reconfigurabilidade, que pode ser muito útil em algumas aplicações. Porém, existe uma desvantagem, a célula de EEPROM é aproximadamente duas vezes o tamanho de uma célula de EPROM.

Desenvolvimentos de transistores MOS com base na tecnologia "double-poly silicon" habilitaram o uso de memórias eletricamente apagáveis com células de tamanho equivalente às EPROMs. A velocidade de acesso destes novos dispositivos de memória também é comparável com o desempenho de EPROMs CMOS, com tempos de acesso na ordem de 110 ns [KYN88]. Estes novos dispositivos de memória são denominados de *RAM Flash*. A tecnologia RAM Flash utiliza o processo de fabricação *CMOS Flash*, derivada do mesmo processo padrão das EPROMs CMOS. Por exemplo, a tecnologia FastFLASH, com espessura característica de 0,35 micron, utilizada no dispositivo FPGAs XC9500XL da Xilinx [XIL00], proporciona maior resistência na retenção da configuração (estimada em 20 anos) e uma durabilidade na ordem de 10 000 ciclos de configurações.

2.3 Arquiteturas de Blocos Lógicos

Uma definição possível para o *bloco lógico* é uma estrutura complexa contendo vários circuitos combinacionais, de múltiplas entradas e uma ou mais saídas. A maioria dos blocos lógicos também contém algum tipo de dispositivo de armazenamento, para viabilizar a implementação de circuitos seqüenciais. A Figura 2.5 ilustra exemplos de blocos lógicos.



(a) Função Lógica - Actel Act-1 (b) Multiplexador - Plessey (c) LUT – Xilinx 3000

Figura 2.5 - Exemplos das estruturas de blocos lógicos.

Como ilustram os exemplos da Figura 2.5, em alguns FPGAs os blocos lógicos possuem estrutura bastante simples, tal como um multiplexador e uma porta NAND de duas entradas. Em outros, são uma estrutura bem mais complexa, tais como tabelas verdade de 3, 4 ou 5 entradas, denominadas de *look-up tables* (LUTs) e outros módulos. Em alguns FPGAs, um bloco lógico configurável corresponde a uma estrutura semelhante a uma PAL (Programmable Array Logic) [BOS96].

Uma LUT de n entradas é implementada como um registrador com 2^n biestáveis e um esquema de seleção de um dos bits deste registrador para colocar na saída. O controle desta seleção é realizado pelas variáveis de entrada da função, como ilustrado na Figura 2.6 para o caso de LUTs de 3 variáveis. Este tipo de bloco lógico possui dimensão relativamente elevada, mas fornece a máxima

flexibilidade, muito maior que a dos blocos implementados com lógica de dois níveis ou com multiplexadores.

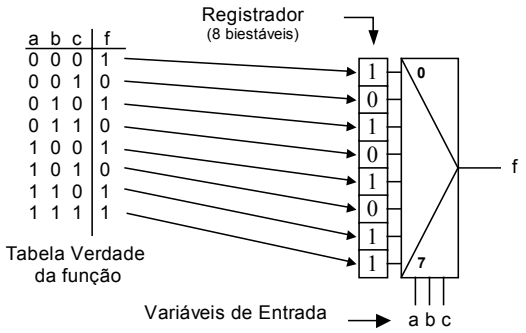
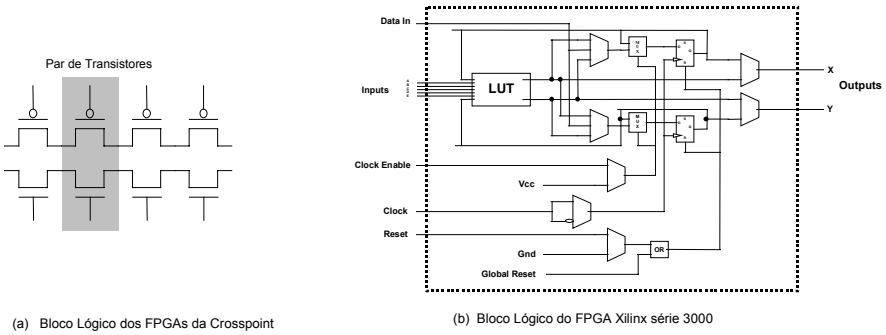


Figura 2.6 - LUT para tabelas de 3 variáveis, implementando a função $f = ab + \overline{c}$.

Rose, em [ROS93], observa que os blocos lógicos em FPGAs, como ilustra a Figura 2.7, diferem em tamanho e capacidade de implementação. O bloco lógico de dois transistores usado no FPGA da Crosspoint [MAR92], pode implementar apenas um inversor, porém é muito pequeno em tamanho, enquanto que o bloco lógico implementado com LUTs, usado nos FPGAs da Xilinx série 3000 pode implementar qualquer função lógica de até cinco variáveis, mas é significativamente maior em tamanho.



(a) Bloco Lógico dos FPGAs da Crosspoint

(b) Bloco Lógico do FPGA Xilinx série 3000

Figura 2.7 - Blocos lógicos (a) Crosspoint e (b) Xilinx série 3000.

Observando as diferentes estruturas que constituem os blocos lógicos, Rose [ROS93] classifica as arquiteturas dos FPGAs pela sua granularidade. *Granularidade* é um termo que pode ser definido usando vários critérios. Exemplos de critérios são o número de funções booleanas que o bloco lógico pode implementar, o número equivalente de portas NAND de duas entradas, o número total de transistores, o total de área normalizada, ou o número de entradas e saídas.

O assunto da granularidade da arquitetura FPGA é bastante complexo, porque em algumas arquiteturas, tais como a dos FPGAs Altera [WON89] ou a dos FPGAs AMD [AMD90], a lógica e o roteamento estão fortemente mesclados e é difícil separar as diferentes características de granularidade. Para simplificar, Rose escolhe classificar a granularidade das arquiteturas comerciais em duas categorias: grão pequeno e grão grande. Ver-se-á a seguir sua proposta de classificação de arquiteturas de FPGAs.

2.3.1 Arquiteturas de Grão Pequeno

O FPGA fabricado pela Crosspoint [MAR92], por exemplo, usa um único par de transistores no bloco lógico, como ilustrado na Figura 2.7(a). A função $f = ab + \overline{c}$ é implementada com pares de transistores conforme mostrado na Figura 2.8. Considerando que os transistores são conectados lado

a lado, alternando filas de transistores P e transistores N, portas lógicas CMOS quaisquer podem ser configuradas, bastando desligar um par de transistores nos seus limites laterais para criar uma região de isolamento entre portas adjacentes.

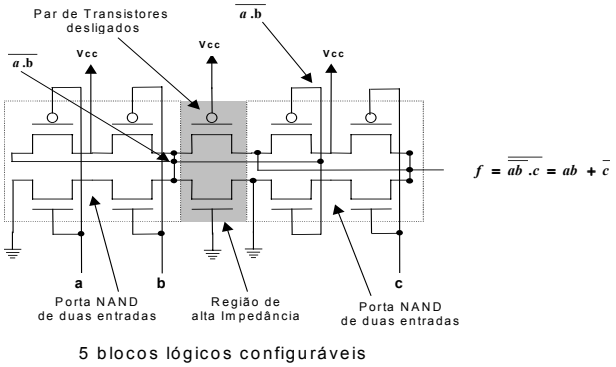


Figura 2.8 - FPGA Crosspoint configurado para a função $f = ab + \overline{c}$ implementando $f = \overline{\overline{ab}.c}$.

Um segundo exemplo de uma arquitetura de grão pequeno é o FPGA ERA60100 da Plessey Semiconductor [PLE89]. Neste, o bloco básico é uma porta NAND de duas entradas como ilustrado na Figura 2.9. A lógica é formada do modo habitual, conectando-se as entradas da NAND por meio de um multiplexador 8x2 para implementar a função $f = ab + \overline{c}$ desejada. Esta função é definida na Figura 2.10(a), e o bloco lógico é configurado para implementar, a função equivalente, mostrada na Figura 2.10(b). Se o *Latch* na saída do bloco não é necessário, então a RAM de configuração é programada para mantê-lo permanentemente transparente.

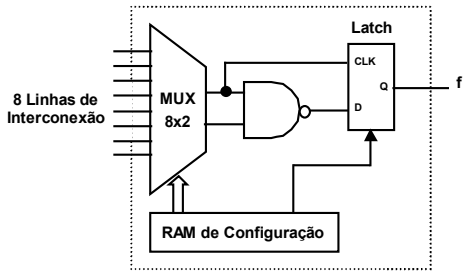


Figura 2.9 - Bloco lógico do FPGA ERA60100 da Plessey Semiconductor.

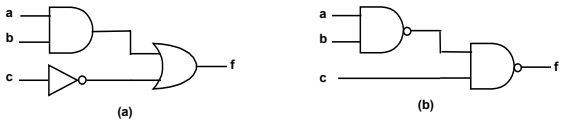


Figura 2.10 – Implementações estruturais da função $f = ab + \overline{c}$. (a) Direta. (b) Equivalente.

Vários outros fabricantes usam blocos de grão pequeno. A Algotronix [ALG89] usa um bloco lógico que pode executar qualquer função de duas variáveis de entrada. Isto é implementado usando um conjunto de multiplexadores configuráveis. Os blocos lógicos dos FPGAs da Concurrent Logic [CON91] e Toshiba [MUR91] contêm portas AND e NAND de duas entradas.

A vantagem principal de uso de blocos lógicos de grão pequeno é que os blocos usados para implementar as funções projetadas são utilizados com muita freqüência de forma completa. A exemplo dos blocos utilizados em tecnologias MPGAs (*Mask-Programmable Gate Arrays*) convencionais e/ou de células padronizadas (*standard cells*) [MIC96], blocos lógicos pequenos podem ser usados mais eficazmente, pois as técnicas de síntese lógica e física, para tais blocos, são semelhantes. A desvantagem principal de blocos lógicos de grão pequeno é que eles exigem um número relativamente grande de segmentos de fios e chaves programáveis para o roteamento. Estes recursos de roteamento são dispendiosos em atraso e área do circuito integrado. Como resultado,

FPGAs empregando blocos de grão pequeno são em geral mais lentos e têm mais baixas densidades em relação aos que empregam blocos de grão grande.

2.3.2 Arquiteturas de Grão Grande

O bloco lógico da família Act-1 da Actel está baseado em multiplexadores para implementar diferentes funções lógicas. Ele funciona conectando cada de suas entradas a uma constante ou a um sinal. Este bloco consiste em três multiplexadores e uma porta OR. O Act-1, conforme é ilustrado na Figura 2.11(a), tem um total de 8 entradas (S_1, S_2, S_3, S_4, x, y, z e w) e uma saída (f), e implementa a função genérica $f = \overline{(s_3 + s_4)}(\overline{s_1}w + s_1x) + (s_3 + s_4)(\overline{s_2}y + s_2z)$.

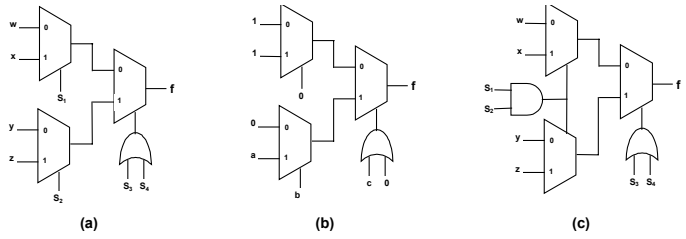


Figura 2.11 – Blocos lógicos Actel (a) Bloco lógico da Act-1. (b) Função $f = ab + \overline{c}$ implementada no Bloco Act-1. (c) Bloco lógico da Act-2.

Conectando cada variável a um sinal de entrada, ou a uma constante, 702 funções lógicas diferentes podem ser implementadas [LIN94]. Por exemplo, a função lógica $f = ab + \overline{c}$ é implementada fixando-se as variáveis como mostrado na Figura 2.11(b): w = 1, x = 1, S_1 = 0, y = 0, z = a, S_2 = b, S_3 = c, e S_4 = 0. O bloco lógico da família Act-2 [AHR90] é semelhante ao da Act-1. A diferença é que a seleção dos dois multiplexadores que configuram as entradas do bloco são unidas e conectadas à saída de uma porta AND de duas entradas, como mostrado em Figura 2.11(c). Esta mudança na estrutura do bloco lógico da família Act-2 permite implementar 766 funções lógicas, ou seja, 64 a mais que na Act-1.

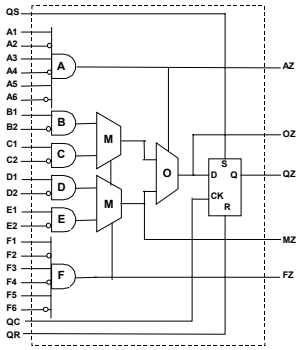


Figura 2.12 - Bloco lógico da QuickLogic.

Semelhante à lógica da Actel, o bloco lógico dos FPGAs da QuickLogic [BIR91] empregam multiplexadores 2x1. Cada entrada do multiplexador é alimentada por uma porta AND, como ilustra a Figura 2.12. Estes blocos baseados em multiplexadores têm a vantagem de permitir um alto grau de funcionalidade para um número relativamente pequeno de transistores. Porém, isto é alcançado às custas de um número grande de entradas (variáveis), 8 no caso da Actel e 14 no caso da QuickLogic estas entradas, quando utilizadas, impõem elevada exigência nos recursos de roteamento. Assim, este tipo de bloco é mais adequado para FPGAs que usam chaves programáveis de pequeno tamanho, tais como antifusíveis.

Os blocos lógico das famílias 3000, 4000 e Virtex da Xilinx [XIL99] entre outras, são células baseadas em tecnologia de configuração SRAM. Os blocos lógicos de FPGAs da Xilinx usam LUTs. Uma tabela verdade, para funções lógicas de K entradas, é armazenada em uma SRAM 2^K x 1 bit. As linhas de endereço da SRAM funcionam como entradas, e a saída da SRAM fornece o valor da função lógica. Por exemplo, considere a tabela verdade da função lógica $f = ab + \overline{c}$. Esta função lógica é implementada utilizando uma LUT de 3 entradas conforme representado na Figura 2.6, para tal a SRAM armazena "1" nos endereços "000," "010," "100," "110", "111" e "0" nos demais como, especificado pela tabela verdade.

Uma LUT de K entradas pode implementar qualquer 2^{2^K} funções de K variáveis. A vantagem de LUTs é que estas exibem alta funcionalidade. A desvantagem é que elas tornam-se muito grandes para funções de mais de cinco entradas. Além disso são necessárias 2^K células de memória para uma LUT de K entradas. Mesmo que o número de funções que podem ser implementadas cresça rapidamente, muitas das funções adicionais não são usadas freqüentemente em projetos práticos, além de serem de difícil utilização por ferramentas de síntese lógica. Desta forma, é freqüente o caso que uma LUT grande ser em grande parte subtilizada. A Seção 2.5 apresenta algumas informações quantitativas sobre este assunto.

O bloco lógico da família 3000 de FPGAs da Xilinx [XIL99] contém uma LUT de 5 entradas e 1 saída. Este bloco pode ser reconfigurado em duas LUTs de quatro entradas, possibilitando uma maior flexibilidade na utilização do bloco lógico, visto que funções lógicas frequentemente não exigem mais de quatro variáveis de entrada. O bloco também permite a implementação de lógica seqüencial e possui vários multiplexadores. Estes últimos permitem conectar as entradas combinacionais com as saídas diretamente ou passando por elementos biestáveis. Os multiplexadores são controlados por células de SRAM carregadas durante a configuração do FPGA.

Na família 4000 de FPGAs Xilinx o bloco lógico contém duas LUTs de 4 entradas e suas saídas são conectadas a uma outra LUT de 3 entradas, como apresentado na Figura 2.13. Este bloco introduz duas mudanças significativas comparadas ao bloco lógico da série 3000. A primeira, dois tamanhos de LUTs são utilizadas, permitindo um melhor compromisso entre desempenho e densidade lógica (ver mais detalhes na Seção 2.5). A segunda é o uso de duas conexões diretas, que ligam as saídas das duas LUTs de quatro entradas para as entradas da LUT de três entradas. Estas duas conexões são significativamente mais rápidas que qualquer interconexão programável, pois nenhuma chave programável é utilizada em série. Com o uso adequado destas conexões rápidas, pode-se melhorar o desempenho do FPGA. Porém há uma desvantagem para este tipo de conexão. Já que a conexão é permanente, a LUT de três entradas tem sua flexibilidade limitada e freqüentemente não pode ser utilizada, reduzindo a densidade lógica global.

O bloco lógico Xilinx 4000 incorpora várias características adicionais. Cada LUT pode ser utilizada diretamente como um bloco de SRAM de pequenas dimensões (32x1 bits com porta simples ou 16x2 ou 16x1 bits porta dupla). Isto permite implementar memórias pequenas mais eficazmente. Outra característica é a inclusão de circuitos dedicados para a implementação de somadores rápidos, via linha de propagação rápida de vai uns (*fast carry addition circuits*).

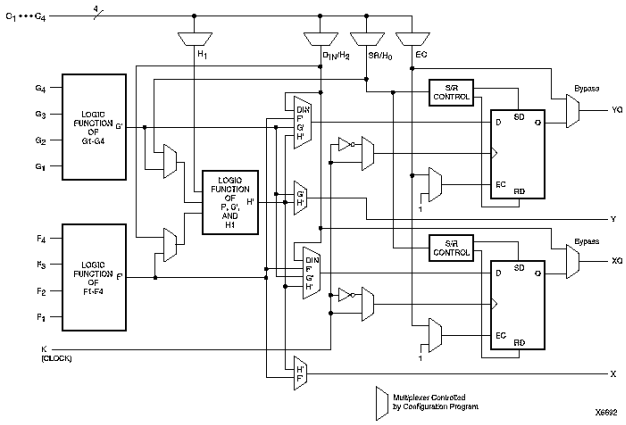
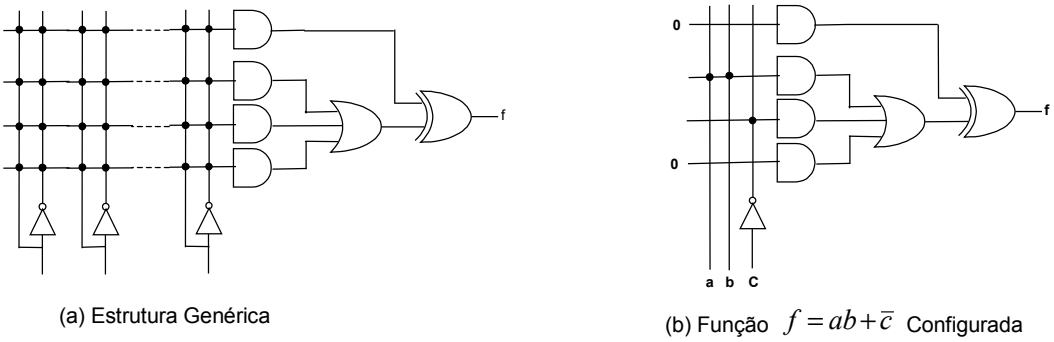


Figura 2.13 - Bloco lógico Xilinx série 4000.

A arquitetura da maioria dos CPLDs e FPGAs Altera [WON89] evoluiu da arquitetura PLA baseada em PLDs tradicionais de baixa densidade [LAL90]. Seu bloco lógico possui um grande número de entradas (de 20 a mais de 100). Este é constituído de portas AND que alimentam portas OR. A Figura 2.14(a) ilustra o bloco lógico do CPLD Altera MAX série 5000. As entradas das portas AND podem ser conectadas a qualquer linha vertical do barramento por meio de chaves eletrônicas com tecnologia de configuração de porta flutuante, vista na Seção 2.2.3. A Figura 2.14(b) ilustra a implementação da função lógica $f = ab + \overline{c}$. As conexões na figura indicam quais chaves eletrônicas estão fechadas para configurar o bloco lógico com a função desejada.



(a) Estrutura Genérica

(b) Função $f = ab + \overline{c}$ Configurada

Figura 2.14 - Bloco lógico Altera série 5000 e seu emprego.

A vantagem deste tipo de bloco é que o grande número de entradas pode ser usado para formar funções complexas com poucos blocos lógicos, reduzindo a necessidade de muitas interconexões programáveis. Isto porém, resulta em perda de densidade, pois é difícil fazer uso eficiente de todas entradas e todas portas. Esta perda não é tão significativa, pois é compensada pela alta densidade das portas *Wired-AND*. As próprias conexões servem também para o roteamento, diferente do que ocorre em outras arquiteturas, onde a configuração da lógica e o roteamento estão separados. Uma desvantagem das portas *Wired-AND* é o uso de resistores *pull-up*, que apresentam alto consumo de energia na configuração.

O bloco lógico dos CPLDs Altera MAX 7000 [VIJ92] é semelhante ao dos MAX 5000. Os blocos lógicos são implementados com dois termos produto a mais que os da série 5000 e também maior flexibilidade, porque blocos adjacentes podem utilizar termos produto um do outro. Vários outros FPGAs fazem o uso de grandes blocos lógicos, estilo AND-OR, entre os quais alguns produzidos pela Concurrent Logic [CON91], AMD [AMD90] e Lattice [BAK91].

2.4 Arquiteturas de Roteamento

A arquitetura de roteamento de um FPGA é a maneira através da qual são configuradas as chaves e segmentos de fios para realizar as interconexões dos blocos lógicos. Nesta Seção apresenta-se uma arquitetura utilizada comercialmente como referência para o estudo das arquiteturas de roteamento de forma genérica, observando-se que cada fabricante possui um modelo particular.

A Figura 2.15 define um modelo genérico de arquiteturas de roteamento de FPGAs comerciais. Neste modelo podemos identificar várias características. Primeiro, um *segmento de fio* é contínuo. Uma ou mais chaves podem ser ligadas a segmentos de fio; Cada final de um segmento de fio está tipicamente ligado a uma chave programável; Uma *trilha* é uma sucessão em linha de um ou mais segmentos de fio e um *canal de roteamento* é um grupo de trilhas paralelas.

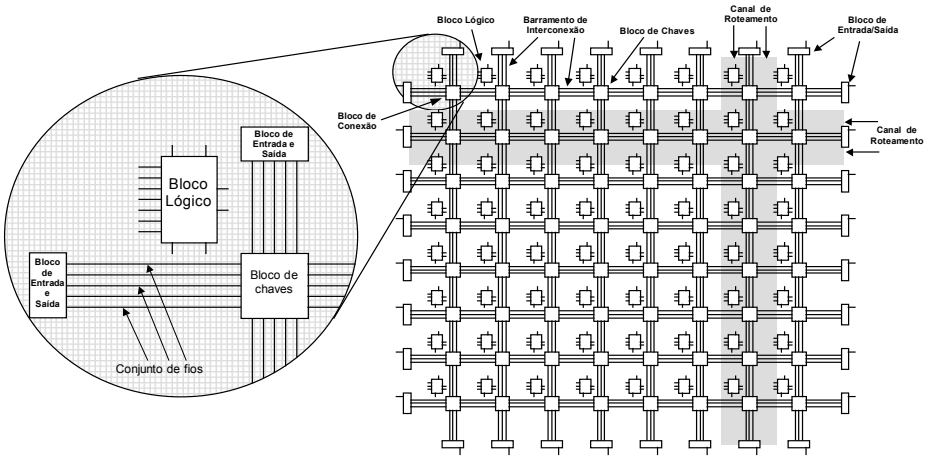


Figura 2.15 - Arquitetura de roteamento genérica.

A partir da Figura 2.15, nota-se que o modelo contém três estruturas básicas. A primeira é o *bloco de conexão* (não mostrado na figura) que aparece em todas as arquiteturas. Um bloco de conexão provê a interligação entre entradas e saídas de um bloco lógico e os segmentos de fio no canal de roteamento. Embora não mostrado na figura, podem haver blocos de conexão tanto na vertical como na horizontal. A segunda estrutura é o *bloco de chaves*, o qual provê a conectividade dos segmentos horizontais com os segmentos verticais. O bloco de chaves provê conectividade entre os segmentos incidentes em seus quatro lados. Em algumas arquiteturas, o bloco de chaves é intercalado com o bloco de conexão, e em outros eles são combinados em uma única estrutura. Um último bloco de relevante importância é o *bloco de entrada/saída*. Através deste é realizada a conexão do FPGA com o mundo externo. Neste bloco, os pinos terminais do componente FPGA são configurados para a transferência de sinais do interior do componente ou para o exterior do componente.

Um exemplo típico de arquitetura de roteamento é a utilizada na família 4000 de FPGAs da Xilinx. As conexões entre os blocos lógicos são feitas no canal de roteamento com segmentos de metal unidos por uma matriz de chaves programáveis. A Figura 2.16, obtida do manual do fabricante [XIL99], ilustra um conjunto de linhas horizontais e verticais interconectadas em um bloco PSM (Programmable Switch Matrix), onde cada matriz de chaves consiste em transistores de passagem configuráveis, utilizada para estabelecer a conexão entre as linhas. O CLB (Configurable Logic Block) executa as funções lógicas do circuito. As interconexões dos recursos são configuradas para formar trilhas de fios, que transportam os sinais lógicos entre os blocos, semelhante às trilhas em placas de circuito impresso convencionais. As funções dos blocos lógicos são implementadas em LUTs, estabelecidas via um arquivo de configuração. Este arquivo é carregado para o interior da matriz, onde é armazenado em células de memória SRAM de configuração.

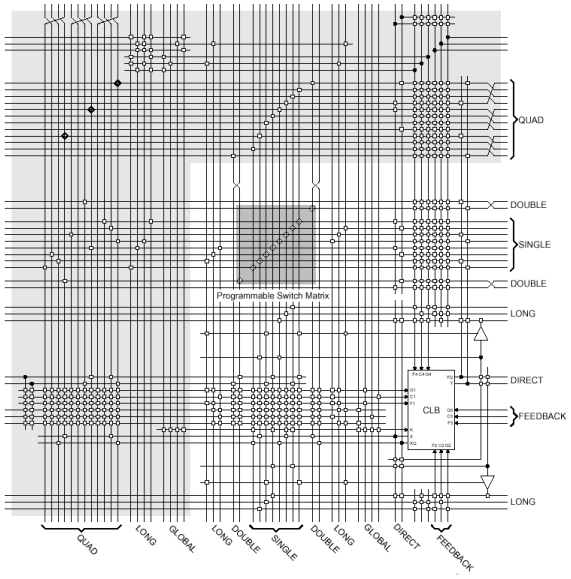


Figura 2.16 - Arquitetura de roteamento Xilinx 4000.

2.5 Granularidade, Densidade e Desempenho de FPGAs

Quando o tamanho do grão do bloco lógico aumenta, espera-se que o número de blocos necessários para implementar um projeto diminuia. Por outro lado, um bloco lógico mais poderoso (de grão maior) exige mais interconexões configuráveis para implementá-lo, e por conseqüência, ocupa mais área. Este situação antagônica sugere a existência de um ponto ótimo para a granularidade de bloco lógico, no qual a área de FPGA dedicada à implementação da lógica é minimizada. A porção de área para a implementação da lógica é relativamente fácil calcular, enquanto que o efeito de granularidade no roteamento não é tão simples, e tem grande influência na área de roteamento de todo o FPGA.

O efeito da funcionalidade do bloco sobre a área de lógica é facilmente observado, por exemplo, a implementação da função de lógica $f = abd + bc\overline{d} + \overline{a}\overline{b}\overline{c}$, utilizando blocos lógicos de três tamanhos do grão diferentes, como ilustrado na Figura 2.17, supondo que cada bloco lógico contém uma LUT.

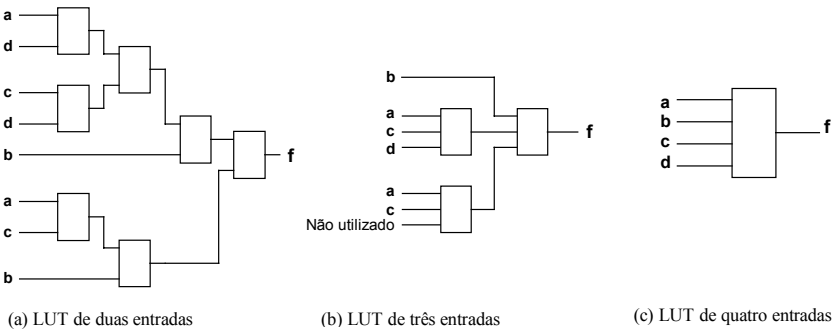


Figura 2.17 - Implementação da função lógica $f = abd + bc\overline{d} + \overline{a}\overline{b}\overline{c}$.

Cada um dos blocos lógicos são LUTs de duas entradas, na Figura 2.17(a), três entradas, na Figura 2.17(b) e quatro entradas, na Figura 2.17(c). A implementação da função com LUTs de duas entradas requer sete blocos lógicos, com LUTs de três entradas requer três blocos, e com LUTs de quatro um único bloco. Considera-se como uma medida de área o número de *bits* de memória necessários para implementar a função lógica *f*, utilizando LUTs de **K** entradas. Considera-se ainda que cada LUT de **K** entradas requer $\mathbf{2^K}$ *bits*, a implementação da função com tabelas de duas

entradas exige um total de 28 *bits*, as de três exigem 24 *bits* e as de quatro 16 *bits*. Conclui-se que, utilizando esta medida de área como referência, as LUTs de quatro entradas exigem menor área de lógica para a implementação da função.

A Figura 2.18 dá um exemplo de resultados experimentais que apontam para um determinado tamanho de grão de bloco lógico, tamanho este que requer a menor área de lógica para a implementação da função [ROS93, ROS90]. Um determinado número de projetos foram mapeados em FPGAs com diferentes tamanhos de grão de bloco. A área total do bloco lógico, bem como a área de roteamento são determinadas para cada mapeamento. São calculadas ainda as médias dos resultados, que então são comparados.

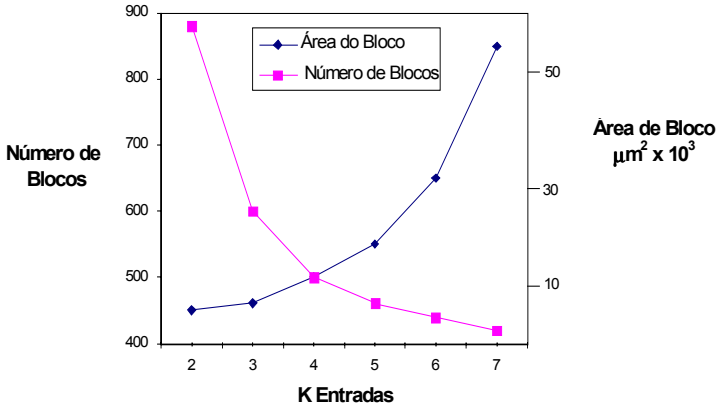


Figura 2.18 - Número de blocos e área do bloco para um circuito [ROS93].

O número de blocos lógicos diminui rapidamente como o aumento de **K** (número de entradas do bloco lógico), enquanto que o tamanho de bloco aumenta exponencialmente com **K**. A área total do bloco lógico (o produto das duas curvas) alcança um mínimo em **K=4**. A área mínima total do bloco lógico possui uma fraca dependência com tamanho da chave programável discutido em detalhes nas referências [GAM89] e [ROS93].

A Figura 2.19 mostra dois parâmetros de um circuito exemplo variando em função do tamanho de LUT empregado. Estes parâmetros são o número total de blocos do circuito e a área de roteamento por bloco. A área de lógica ativa é parte da área total. A área para o roteamento é normalmente maior que a área ativa. Em FPGAs, o roteamento representa de 70 a 90% da área total.

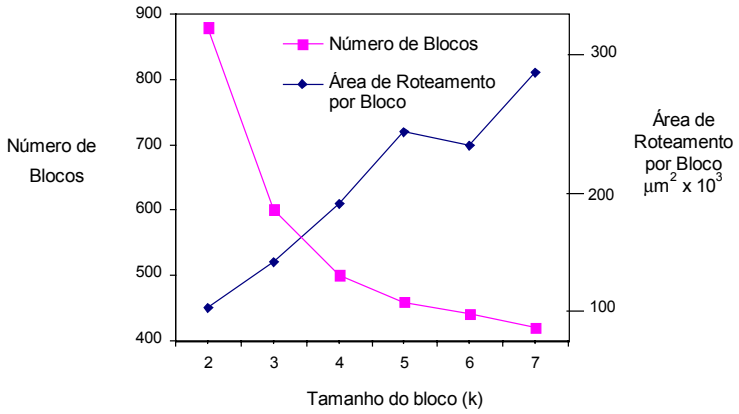


Figura 2.19 - Número de blocos e área de roteamento versus tamanho do bloco para um circuito exemplo [ROS93].

2.6 Núcleos de Propriedade Intelectual (IP Cores)

Um *Núcleo de Propriedade Intelectual* é um sistema digital complexo pré-projetado e pré-validado, cuja utilização como módulo componente de um sistema maior é licenciado a terceiros pelo desenvolvedor [VAH01]. O nome deriva da nomeclatura em inglês, Intellectual Property core, ou IP core.

Tecnologias de implementação submicrônicas introduziram a possibilidade de integrar todo um sistema complexo em um único CI, criando o conceito de Integração a nível de sistema (em Inglês, System-Level Integration – SLI, ou mais coloquialmente , System-On-a-Chip – SOC). O conceito de IP core surgiu [CAS97] como forma de reduzir o tempo de desenvolvimento de produtos complexos viabilizados pela crescente taxa de integração de CIs VLSI (do inglês, *Very Large Scale Integration*) em SOCs.

A comercialização de propriedade intelectual sob a forma de IP cores é hoje uma alternativa à comercialização de CIs complexos para os desenvolvedores de componentes. Para aquele que adquire um IP core, existem vantagens importantes. Primeiro, a melhoria do desempenho do produto final pode ser importante, devido à integração de diversos componentes poder ocorrer em um único CI. Segundo, aumentando-se o sigilo do projeto, pela implementação de CIs dedicados. Terceiro e mais importante, o uso de IP cores pode levar a uma redução significativa no tempo de desenvolvimento do projeto, capitalizando na pré-validação do IP core licenciado.

Contudo, também podem haver desvantagens para quem licencia a para quem desenvolve sistemas a serem comercializados sob a forma de propriedade intelectual. O desenvolvedor, por um lado, expõe seu projeto até certo ponto, sem o que pode inviabilizar a integração do IP core por quem licencia. Esta exposição implica um risco maior ao sigilo do projeto do IP core que a comercialização dos CIs. No lado do licenciador de um IP core, as desvantagens principais são o alto custo, comprado com um CI encapsulado equivalente e a eventual inflexibilidade do IP core a adaptar-se aos requisitos de um projeto específico. O alto custo deriva do fato de se tratar de propriedade intelectual reutilizável. Assim, considerações de volume de produção e número de produtos que podem empregar um dado IP core são importantes antes do compromisso de adquirir tais módulos. A necessidade de flexibilidade de um IP core, por outro lado, leva os desenvolvedores a criar compromissos de grau de validação, grau de sigilo e custo do IP core, gerando diferentes formas de disponibilizar estes componentes.

Os FPGAs habilitam o desenvolvimento de SOCs mediante arquiteturas reconfiguráveis e/ou plataformas de prototipação. O desenvolvedor pode fazer uso de IP cores da mesma forma dos que estão disponíveis para o desenvolvimento de CIs complexos. Estes IP cores também podem ser disponibilizados visando o uso em dispositivos FPGAs. Na Seção a seguir, explora-se uma classificação das formas de disponibilização de IP cores.

2.6.1 Classificação de IP Cores

Conforme Case em [CAS97], quatro critérios podem ser usados para diferenciar e classificar os IP cores. Um destes é a rigidez (hardness) dos blocos, ou seja o grau em que o IP core pode ser modificado para um processo de implementação particular. Outros três critérios são: a modelagem; a flexibilidade e a previsibilidade. Estes critérios são usados na Tabela 2.1, produzindo três classes de IP cores. Estas classes são:

a) **Hard Core:** otimizado para uma tecnologia específica, não podendo ser modificado pelo projetista. Possui uma organização pré-definida e uma planta baixa (do inglês,

Floorplaning) incluída com a arquitetura do projeto. As vantagens são uma temporização garantida e poder ser tratado como célula elemento de uma biblioteca durante o projeto. A desvantagem é que o usuário poderá não ser capaz de personalizar as funções ou sintonizar a temporização do módulo ao restante do projeto.

b) **Firm Core:** apresentado como uma mistura de código fonte e código objeto (sob forma de um Netlist) que depende de uma determinada tecnologia. Neste tipo de core, o código fonte é aberto ao projetista e partes específicas podem ser personalizadas. Contudo a parte especializada com netlist possui tecnologia específica e o usuário não pode facilmente substituir o dispositivo de hardware pelo de outro fabricante.
c) **Soft Core:** fornecido normalmente sob a forma de uma descrição em com uma linguagem para descrição de hardware (HDL – do inglês, Hardware Description Language). Um IP Soft Core oferece liberdade quanto à tecnologia do fabricante, permite grande flexibilidade ao projetista. Este pode facilmente modificá-los ou resintetizá-los para diferentes tecnologias, por exemplo substituindo o dispositivo de hardware pelo de outro fabricante, de acordo com a conveniência do projeto. A desvantagem deste tipo de core é que a temporização é um aspecto crítico. Não é garantido que o IP core possa ser sintetizado, mapeado, localizado e tenha suas rotas definidas para cada uma das possíveis implementações.

Tabela 2.1 – Três classificações de IP cores baseado em quatro critérios.

	Hard core	Firm core	Soft Core
Rigidez	A organização é predefinida.	Combinação de código fonte e netlist dependente de tecnologia.	Apresenta um código fonte comportamental independente da tecnologia.
Modelagem	Modelado como um elemento de biblioteca.	Combinação de blocos sintetizáveis fixos. Permite compartilhar recursos com outros cores.	Sintetizável com diversas tecnologias.
Flexibilidade	Não pode ser modificado pelo projetista.	A personalização de funções específicas é dependente da tecnologia.	O projeto pode ser modificado e independe da tecnologia.
Previsibilidade	Temporização é garantida.	Caminhos críticos com temporizações fixas.	A temporização não é garantida, podendo não atender os requisitos do projeto.

Um exemplo típico de um IP core comercial bastante difundido é o PCI LogiCORE da Xilinx, comentado na Seção 3.4.3 Este IP core permite ao projetista construir e personalizar um sistema de barramento compatível com o padrão PCI de 32 bits e freqüência de operação de 33 MHz.

2.7 Famílias Recentes de FPGAs e CPLDs

FPGAs têm evoluído de forma acelerada, em geral de forma mais acelerada que os segmentos de mercado de memória ou processadores. Novas famílias surgem a todo momento para endereçar os mais diferentes aspectos de sistemas eletrônicos tais como alta capacidade, baixo custo, sistemas completos em um único CI (em inglês, *System On a Chip* ou SOC), dotação de módulos especiais, etc.

As características de FPGAs modernos diferem dos discutidos até aqui sobretudo em quatro aspectos. O primeiro destes é a *forma do bloco lógico básico*, que mudou de um bloco monolítico para um conjunto de blocos com roteamento local de alto desempenho, introduzindo o conceito denominado em inglês de *cluster*. O segundo aspecto é a adição de *módulos com funcionalidade específica* aos dispositivos, visando suprir a incapacidade de implementar tal funcionalidade de maneira eficiente usando blocos lógicos convencionais. Entre os blocos mais difundidos encontram-se memórias de acesso simples ou duplo, controles de relógio tais como DLLs ou PLLs e multiplicadores. O terceiro aspecto é a adição de *capacidades especiais ao dispositivo*, tal como a reconfigurabilidade parcial, visando aumentar ainda mais as vantagens mais marcantes de FPGAs em relação a outros dispositivos de hardware. O último aspecto é a agregação de *IP hard/Soft Cores de processadores*, proprietários ou de outros fabricantes estabelecidos diretamente no mesmo substrato de silício que um FPGA. Isto produz o conceito de sistemas em um CI reconfigurável, ou SORC (do inglês, *System-on-a-Reconfigurable Chip*).

As Seções a seguir dedicam-se a explorar cada um destes aspectos, e a descrever como tais aspectos são endereçados por algumas das principais famílias recentes de FPGAs e CPLDs.

2.7.1 Blocos Lógicos do tipo Cluster e Módulos com Funcionalidade Específica

A grande desvantagem de FPGAs com relação a outros tipos de ASICs é o atraso de propagação. As chaves configuráveis, como foi discutido na Seção 2.2, provocam a redução da velocidade nos circuitos interconectados, e com isso reduzem também o desempenho de todo o sistema configurado no FPGA. A arquitetura do FPGA é outro fator do desempenho. Por esta razão, muitas das novas famílias de FPGAs foram desenvolvidas com uma arquitetura baseada em *clusters lógicos* [MAR00]. Um cluster lógico é definido como um grupamento de elementos lógicos, ligados a um sistema de interconexões local de alta velocidade. Fabricantes como Aletra, Xilinx, Actel e outros, implementam FPGAs com diversos tipos clusters lógicos. Estes clusters são formados por conjuntos de elementos lógicos básicos (basic logic elements – BLEs). Por definição, um BLE é uma unidade lógica mista pequena e indivisível, seqüencial e combinacional, conforme representado pela Figura 2.20(a).

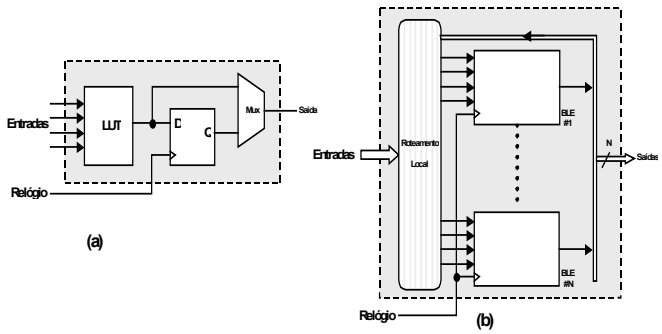


Figura 2.20 – Estrutura interna de blocos lógicos em FPGAs modernos: (a) elemento lógico básico e (b) cluster lógico.

O BLE, geralmente, é constituído de uma LUT de 4 entradas e um registrador, ambos conectados a um multiplexador como saída do bloco, como ilustrado na Figura 2.20(a). Um cluster lógico é visto, na Figura 2.20(b), como um bloco lógico (análogo a um CLB) combinando um ou mais BLEs e interconectados por um sistema especializado de roteamento local, que permite a qualquer entrada de um BLE conectar-se a qualquer entrada do cluster ou a qualquer saída de um BLE. Além de clusters lógicos, dedicados e com funções pré-definidas. Estes blocos podem ser utilizados, como dispositivos periféricos, tais como: memórias, DLLs ou PLLs (do inglês, *Delay Locked Loops e Phase Locked Loops*, respectivamente).

A título de exemplo, aqui são discutidas seis famílias de PLDs que apresentam blocos lógicos dedicados. Inicialmente veremos duas famílias que se utilizam estruturas do tipo clusters lógicos. A Figura 2.21 apresenta a estrutura da família FLEX de FPGAs da Altera. Dois blocos matriciais embutidos (*embedded array block*, EAB) no interior de FPGAs das famílias ACEX 1K e FLEX 10K da Altera [ALT01a, ALT01b]. O EAB é um bloco flexível de memória RAM, com registradores nas portas de entrada e saída, podendo ser configurado como uma matriz de portas lógicas para implementar mega-funções, tais como vetores, multiplicadores e circuitos para correção de erros.

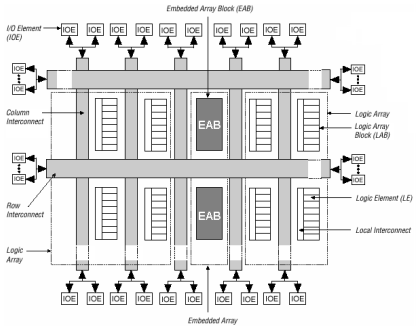


Figura 2.21 –Arquitetura da família FLEX 10K da Altera.

A família ACEX 1K permite implementar blocos de memória RAM de dupla porta com largura de até 16 bits por EAB. Cada EAB da família FLEX 10K pode ser configurado em bancos de memória RAM de 256 × 8, 512 × 4, 1,024 × 2, ou 2,048 × 1. O cluster lógico destas famílias (denominados Logic Array Blocks, ou LABs) apresenta 8 CLBs (denominados Logic Elements, ou LEs).

O EAB, alega a Altera, traz algumas vantagens sobre os FPGAs que possuem outros blocos de memória RAM, tais como os de outra família, as de FPGAs Spartan-II da Xilinx, que usa o conceito de Block-Select RAM [XIL00a], exemplificados na Figura 2.22. Nesta família a distribuição dos blocos de RAM na periferia do dispositivo provoca atrasos que crescem proporcionalmente com o aumento do tamanho total da memória configurada. Além disto, os blocos de RAM são propensos a problemas de roteamento, porque pequenos blocos devem ser interconectados para constituírem um bloco maior. Em contraste, ainda segundo a Altera, os EABs podem ser usados para implementar grandes blocos de RAM dedicados. Desta forma, geram menor atraso e menos problemas de roteamento em relação aos Block-Select RAM da Xilinx.

Os blocos de memória do FPGA XC2S15 da família Spartan-II podem implementar uma memória RAM de 16K bits, interligando-se os quatro blocos de 4K bits, conforme ilustra a Figura 2.22. A família Spartan-II além de implementar memórias de até 56K bits (usando o maior FPGA da família, o XC2S200), também possui quatro blocos de função pré-definida, dispositivos DLLs para controle de gerenciamento de sinais de relógio. Estes permitem manter o escorregamento de relógio dentro do chip dentro de limites razoáveis, além de prover possibilidade de dividir ou multiplicar a freqüência de operação dentro de certos limites. O cluster lógico da família Spartan II possui ou equivalente a 4 CLBs do modelo discutido acima.

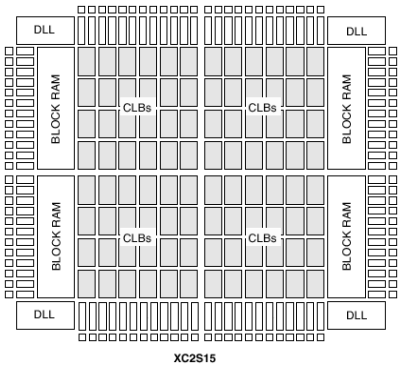


Figura 2.22 – Planta baixa básica da família de FPGAs Spartan-II da Xilinx. ilustrada com o dispositivo XC2S15.

Juntas, as empresas Xilinx e Altera detêm mais de 70% do mercado mundial de FPGAs, donde a importância de estudar prioritariamente famílias destes. A família FLEX 10K da Altera é mais antiga e encontra-se sendo substituída em aplicação pelas famílias APEX 20K e APEX II, a serem discutidas a seguir. As famílias ACEX e Spartan/Spartan II competem no segmento de FPGAs de baixas densidade e custo (alguns dólares por dispositivo com cerca de 5000 a 150000 portas lógicas equivalentes). Por outro lado, o mercado de FPGAs de alta densidade é dividido pelas famílias APEX da Altera e Virtex da Xilinx. A primeira inclui as subfamílias APEX 20K e APEX II (estado da arte em alta densidade). A última inclui as subfamílias Virtex, Virtex-E (mais densidade que a Virtex), Virtex EM (mais memória que as anteriores) e Virtex-II (estado da arte em alta densidade).

A família APEX 20K, representada no diagrama de blocos na Figura 2.23, possui um sistema de blocos embutidos (em inglês, *embedded system block* ou ESB), utilizados para implementar funções de memória e lógica do tipo soma de produtos (como discutido na Seção 0) em grandes funções combinacionais.

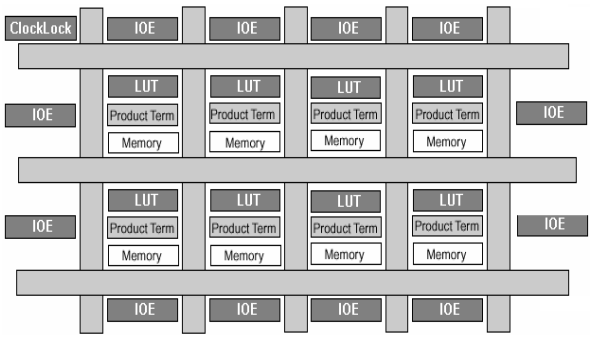


Figura 2.23 – Planta baixa da família de FPGAs APEX 20K da Altera.

Os dispositivos APEX 20K são compostos de uma série de grandes estruturas (MegaLAB), cada uma composta por 16 matrizes de blocos, um ESB, e um canal de interconexões para roteamento dos sinais. O ESB pode implementar várias funções de memória, desde CAMs (Content Addressable Memories), ROMs, RAMs, RAMs de dupla porta e memórias do tipo FIFO. Além disso, o conjunto de ESBs permite ao APEX 20K implementar grandes bancos de memória. A capacidade da família vai de 60 mil até 2,5 milhões de portas equivalentes na subfamília APEX 20K e de 600 mil a 7 milhões de portas na subfamília APEX II, esta última lançada em abril de 2001. O cluster lógico destas famílias (denominados Logic Array Blocks, ou LABs) apresenta 10 CLBs (denominados Logic Elements, ou LEs).

A família de FPGAs Virtex da Xilinx [XIL00a], ilustrada na Figura 2.24, é composta de três blocos principais: lógicos configuráveis (CLBs), de entrada/saída (IOBs) e de memória RAM (BRAMs). Os CLBs são interconectados por uma matriz de roteamento geral (*general routing matrix* - GRM) composta de uma matriz de chaves localizadas na intercessão dos canais de roteamento verticais e horizontais.

Na arquitetura Virtex, um conjunto de CLBs é denominado pela Xilinx por VersaBlock e representado como uma das colunas (CLBs e BRAMS) na Figura 2.24. Cada CLB de uma das colunas possui recursos para o roteamento local e conexão com a matriz de roteamento geral (GRM). Um anel de roteamento periférico, denominado de VersaRing permite um roteamento adicional para os VersaBlocks com os blocos de entrada e saída (IOBs). Esta arquitetura apresenta blocos de memória RAM dedicados (BRAMs) de 4096 bits cada um, e conta com 4 a 8 blocos dedicados, que implementam as funções de DLLs para o controle, distribuição e compensação de atrasos do relógio.

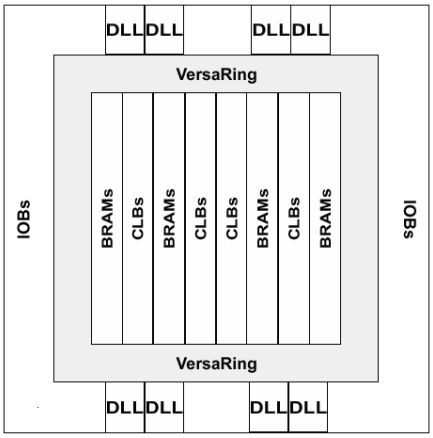


Figura 2.24 –Planta baixa típica dos dispositivos da família Virtex da Xilinx. Exemplo é a subfamília Virtex-E.

A capacidade da família vai de 50 mil até 1 milhão de portas equivalentes na subfamília Virtex e de 40 mil a 10 milhões de portas na subfamília Virtex II. existem ainda subfamílias intermediárias que aumentam a quantidade de recursos de controle de relógio e a velocidade (Virtex-E) e a quantidade de memória (Virtex-EM). A subfamília Virtex II vem de fato para substituir a Virtex, o que não ocorre com APEX 20K e APEX II, que são complementares. O cluster lógico da família Virtex possui ou equivalente a 4 CLBs do modelo discutido acima (idêntico ao da família Spartan).

Cabe ainda citar um outro tipo de dispositivos de sucesso comercial, os CPLDs (c*omplex programmable logic devices)* tal como os da família XC9500 da Xilinx, baseados em Flash-RAM. A arquitetura da família aparece na Figura 2.25, e é também baseada em clusters lógicos.

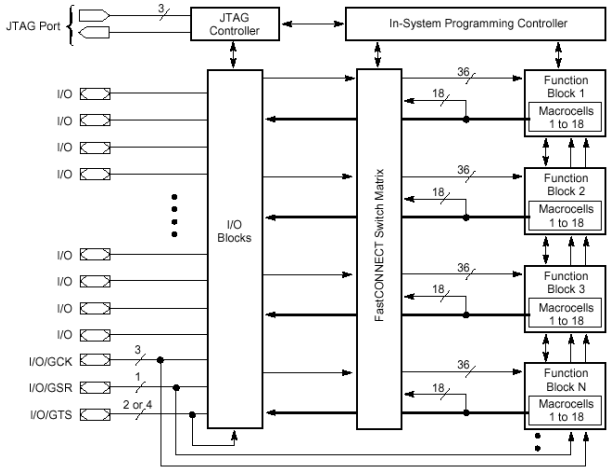


Figura 2.25 – Arquitetura da família XC9500 da Xilinx.

Nesta, o dispositivo é um subsistema constituído de múltiplos blocos de funções (function blocks – FBs) e blocos de entrada/saída (IOBs). Estes blocos estão totalmente interconectados por uma matriz de chavamento rápido, denominada pelo fabricante de "FastCONNECT switch matrix". Os

IOBs servem como buffers para outros dispositivos conectados as entradas e saídas do CPLD. Nesta arquitetura os FBs são um tipo de cluster lógico com a capacidade de até 36 entradas e 18 saídas. Cada um dos FBs, como mostrado na Figura 2.26, é composto de 18 macro células independentes, capazes de implementar funções combinacionais e/ou registradores. Estes FBs permitem configurar circuitos lógicos implementados com soma de produtos. A matriz de chaveamento rápido permite interconectar todas as entradas e saídas de sinais a todas entradas e saídas possíveis (de 12 a 18 saídas, dependendo do encapsulamento) dos FBs e/ou aos IOBs. Todas as macro células também recebem um relógio e o set/reset globais.

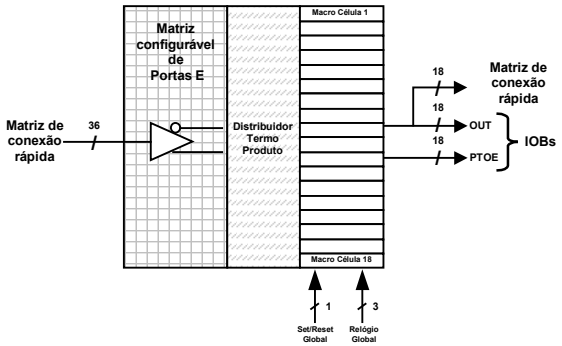


Figura 2.26 – Bloco de funções (FB) da família XC9500 da Xilinx.

As 36 entradas estão conectadas a uma matriz configurável de portas E. A matriz possibilita configurar 90 termos produto, que podem ser alocados por qualquer macro célula pelo distribuidor de termo produto. Cada um dos FBs suporta conexões de realimentação local, que permitem a qualquer saída ser conectada a sua própria matriz de portas E, mantendo os caminhos da conexão no interior do FB. Estas conexões de realimentação local são utilizadas contadores rápidos e máquinas de estados cujos registradores de estados encontram-se no interior do FB.

2.7.2 Capacidades Especiais e IP Hard/Soft Cores

Como explicado no início da Seção 2.7, existem capacidades especiais agregadas a algumas das famílias modernas de FPGAs. Duas destas capacidades têm potencial para permitir a construção de sistemas mais poderosos que os possíveis até então: a *reconfigurabilidade parcial* de dispositivos e a disponibilidade de módulos especiais capazes de aumentar significativamente a *freqüência de operação* para níveis inviáveis de serem obtidos com o estado da arte em tecnologia CMOS, a mais usada na construção de dispositivos VLSI do tipo FPGA. Discute-se aqui estas capacidades especiais e os dispositivos que habilitam seu uso, além de se discutir a integração em FPGAs de módulos processadores de propriedade intelectual de alta densidade, gerando o conceito de *sistemas reconfiguráveis implementados em um único circuito integrado*, ou SORC (do inglês System On a Reconfigurable Chip).

A primeira capacidade citada, reconfigurabilidade parcial de dispositivos consiste de permitir que uma parte do FPGA seja reconfigurado enquanto outras partes encontrem-se executando alguma tarefa em hardware. O potencial desta capacidade é tido como imenso, mas resta provar sua viabilidade. Vários trabalhos de pesquisa têm mostrado este potencial, mas as ferramentas de software e hardware para permitir o controle de configurações parciais ainda são por demais incipientes. Resta também demonstrar de forma inequívoca as vantagens que podem ser obtidas pela utilização de reconfigurabilidade parcial sobre reconfigurabilidade total ou não reconfigurabilidade em aplicações práticas. Um primeiro passo teórico neste sentido é a definição do conceito de *densidade funcional*, devido a Wirthlin e Hutchings [WIR98]. Estes autores definem uma métrica que relaciona tempo (de execução de tarefas e de configuração e/ou reconfiguração do hardware/software do sistema) e área de silício ocupada, procurando medir a eficiência global de

uma implementação. Experimentos realizados demonstram que o uso de dispositivos reconfiguráveis aumenta a densidade funcional de sistemas, e que potencialmente a reconfigurabilidade parcial pode aumentar ainda mais a densidade funcional.

A reconfigurabilidade parcial tem pouco suporte em dispositivos comerciais, aparecendo no momento em apenas duas famílias, a Virtex da Xilinx e a AT6000 da Atmel [ATM99]. Um dos problemas principais de acrescentar esta característica em FPGAs é a sobrecarga de hardware para viabilizar a reconfiguração parcial, que define a necessidade de se fazer um compromisso entre esta sobrecarga e a flexibilidade da reconfiguração. Tal compromisso reflete-se na granularidade da reconfiguração parcial, ou seja, no tamanho mínimo do bloco do FPGA que pode ser reconfigurado de forma independente. Na família Virtex, por exemplo, já mencionada anteriormente, esta unidade corresponde a 1/48 de uma coluna inteira de blocos lógicos no FPGA. A arquitetura de reconfiguração da Virtex é extremamente complexa, e infelizmente precisa ser dominada em grande detalhe para permitir tirar proveito da reconfigurabilidade parcial. Isto se deve ao fato de que o suporte à reconfiguração parcial é extremamente limitado, com ferramentas de software de baixíssimo nível de abstração sendo o único suporte disponível. Exemplo de ferramenta é o software JBits da Xilinx, um conjunto de classes Java para auxiliar no processo de desenvolvimento de ferramentas de reconfiguração parcial. Por outro lado, a família AT6000 da Atmel usa uma abordagem mais moderna de chaveamento de contexto de configuração introduzido em pesquisa pelos dispositivos DPGA (do inglês Dynamically Programmable Gate Array) [TAU95]. A família AT6000, baseada em tecnologia SRAM, possui a capacidade de reconfiguração parcial, e é ilustrada na Figura 2.27. Nela, uma memória de configuração interna armazena novas aplicações reconfiguráveis de forma parcial no FPGA.

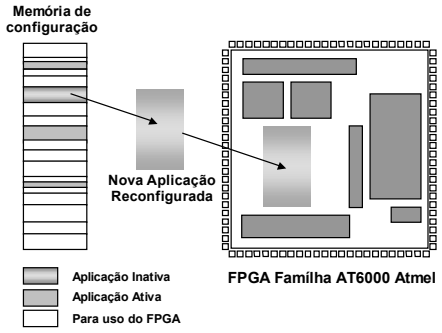


Figura 2.27 – Família AT6000 da Atmel utiliza a tecnologia SRAM e permite reconfiguração parcial.

Famílias não mais disponíveis de FPGAs introduziram o conceito de reconfiguração parcial antes das citadas. Os dois exemplos mais relevantes são a família CLAy da National Semiconductor, de baixa densidade e usada na arquitetura reconfigurável DISC, citada no Capítulo 3, e a família XC6200 da Xilinx, descontinuada pela empresa devido à incapacidade de desenvolver ferramentas eficientes para a síntese física.

A segunda capacidade especial citada, a possibilidade de aumento significativo da freqüência de operação aparece em pelo menos um dispositivo comercial, a família QuickSD de FPGAs da Quicklogic. Através do uso de multiplicadores de relógio internos (PLLs), interfaces diferenciais do tipo LVDS e blocos serializadores/desserializadores (SERDES) os dispositivos desta família conseguem em seus pinos freqüências de transmissão de dados na ordem do Gbps [BUR00]. Esta é uma característica altamente desejável ao suporte de aplicações tais como redes de alta velocidade.

No que concerne a integração em FPGAs de módulos processadores de propriedade intelectual de alta densidade, existem hoje várias opções no mercado. Estas opções podem ser classificadas em três categorias, com base no grau de complexidade do SORC resultante:

- *SORCs de baixa densidade* - envolvem dispositivos compostos por um IP hard core de um microcontrolador comercial tal como o 8051 da Intel, o 68HC11 da Motorola ou outro de porte similar, uma certa quantidade de memória, dispositivos periféricos tais como timers e controladores de DMA e uma matriz de blocos lógicos configuráveis com capacidade entre 10 mil a 100 mil portas equivalentes;

- *SORCs de média densidade* - normalmente envolvem o uso de FPGAs de alta densidade (de 200 mil a alguns milhões de portas equivalentes), combinados com licenças de uso Soft Cores de processadores de média capacidade, que podem em geral ser instanciados múltiplas vezes dentro do FPGA. Isto pode gerar inclusive sistemas multiprocessados;

- *SORCs de alta densidade* - combinam a mesma estrutura de SORCs de baixa densidade, trocando contudo a escala e a complexidade dos componentes. Ou seja, os hard cores de processadores simples são trocados por hard cores de processadores complexos e poderosos, o porte dos periféricos passa a ser maior, e a disponibilidade de hardware reconfigurável passa a ser equivalente ou superior ao dos SORCs de média densidade.

Exemplos de SORCs de baixa densidade são a família E5 da Triscend e a série AT94K da Atmel. Exemplos da segunda classe são a família de processadores embarcados Nios da Altera, parte da iniciativa denominada Excalibur pela empresa, e os processsadores embarcados MicroBlaze da Xilinx, parte da iniciativa Empower desta empresa. Finalmente, exemplos de SORCs de alta densidade existem dentro das mesmas iniciativas Excalibur e Empower da Altera e da Xilinx, respectivamente. No caso da Excalibur, a Altera possui CIs contendo hard cores licenciados de processadores comerciais MIPS e ARM das empresas de mesmo nome, enquanto que Empower cita a disponibilidade próxima de CIs contendo hard cores do processadores PowerPC da IBM.

Outro aspecto interessante de SORCs de todas as classes é a necessidade de definição de padrões para barramentos de comunicação internos dentro de SORCs. Exemplos são os barramentos padrão AMBA da ARM, o CoreConnect da IBM e o Configurable System Interconnect Bus (CSI) da Triscend. Estes barramentos unificam a forma de comunicação entre módulos de propriedade intelectual internos ao SORC, facilitando a integração de sistemas complexos. A desvantagem dos barramentos citados reside no fato de serem todos padrões *de facto*, propostos por fabricantes específicos. Iniciativas como a OpenCores (http://www.opencores.org) procuram incentivar o uso de padrões abertos e gratuitos, entre eles os padrões de barramento internos de SORCs, como o Wishbone.

3. Arquiteturas Reconfiguráveis/Prototipação Rápida

O Capítulo 2 apresentou o conceito de dispositivos reconfiguráveis, discutindo sobre tudo FPGAs baseados em RAM, sua funcionalidade e arquitetura.

O presente Capítulo visa discutir três tipos de sistemas ou subsistemas digitais tipicamente constituídos em torno ou usando dispositivos reconfiguráveis: arquiteturas reconfiguráveis, plataformas de prototipação rápida e núcleos de propriedade intelectual. Cabe salientar que tais sistemas não implicam necessariamente o emprego de dispositivos reconfiguráveis, sobre tudo do último da lista.

A Seção 3.1 apresenta algumas definições básicas. As duas Seções seguintes versam sobre arquiteturas reconfiguráveis, enquanto a Seção 3.4 discute plataformas de prototipação rápida.

3.1 Definições Básicas

Um *sistema computacional*, mostrado esquematicamente na Figura 3.1, é uma composição de um sistema digital e um software que executa sobre este, tal como apresentado por Calazans em [CAL98]. Um *sistema digital* pode ser visto como uma estrutura com entradas e saídas capaz de processar entradas e gerar saídas.

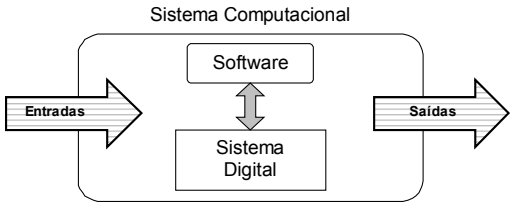


Figura 3.1 - Sistema computacional.

O desempenho de um sistema computacional pode ser definido a partir da avaliação de um conjunto de parâmetros entre os quais destacam-se: o tempo de execução; a energia consumida no tempo (potência consumida) e o preço do sistema como um todo.

Os sistemas computacionais têm seu valor agregado medido por critérios específicos do campo de aplicação, tais como, desempenho, facilidade de configuração e custo de fabricação. Os sistemas têm diferentes aplicações, e cada aplicação específica requer a definição de uma *arquitetura*, e esta por sua vez, deve ser projetada para atender aos objetivos da aplicação. Os requisitos de desempenho e custo de sistemas computacionais complexos tornam difícil a obtenção de um produto com valor agregado alto, transformando a fase de projeto destes em uma arte.

Muitas vezes, um produto com alto valor agregado só pode ser obtido construindo-se um sistema computacional especializado, com uso uma arquitetura dedicada. Uma das formas recentemente introduzidas para construir arquiteturas dedicadas eficientes e flexíveis é através do emprego de *arquiteturas reconfiguráveis*. Uma arquitetura reconfigurável, para a finalidade deste trabalho, é uma composição de um ou mais processadores de propósito geral (em inglês, general purpose processors ou GPPs), um ou mais ASPs (em inglês, Aplication Specific Processors) e uma estrutura de memória compartilhada ou não, entre o(s) GPP(s) e ASP(s). A Figura 3.2 mostra uma estrutura típica para o modelo arquitetura reconfigurável, onde um barramento padrão (endereços, dados e controle) interconecta GPPs, a estrutura de memória e os ASPs. Os ASPs são os dispositivos de hardware cuja funcionalidade pode ser alterada durante o uso do sistema [ADA97].

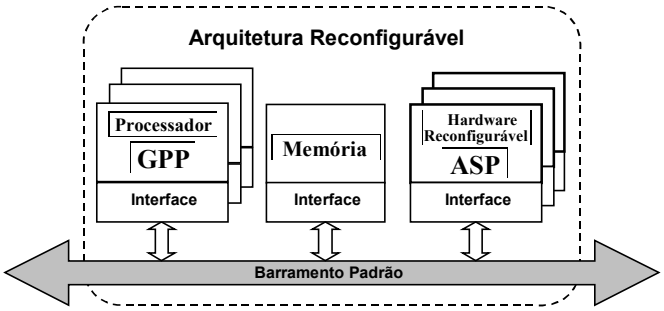


Figura 3.2 – Modelo de arquitetura reconfigurável.

Se uma dada arquitetura não possui parte reconfigurável, ela é dita *fixa*. Quando é necessário executar uma determinada tarefa computacional que exija alto desempenho, observa-se que o emprego de uma arquitetura reconfigurável, para implementar o sistema computacional, pode vir a superar algumas limitações. Entre estas, pode-se citar o tempo de execução as dimensões ou a flexibilidade geral do sistema.

Neste Capítulo, para fins de classificação, limita-se o escopo ao estudo de uma classe específica de arquiteturas reconfiguráveis de grande utilidade prática: aquelas compostas de apenas um GPP, um ASP e um subsistema de memória.

3.2 Classificações das Arquiteturas Reconfiguráveis

3.2.1 Introdução

Componentes de hardware reconfigurável usados em combinação com GPPs habilitaram um novo método para implementar aplicações. A combinação de um processador e um hardware reconfigurável define uma *arquitetura reconfigurável*, que aponta a um futuro no qual a dinâmica dos sistemas computacionais muda radicalmente. Esta mudança denota o quão bem e flexivelmente, ou não, este modelo poderá servir a solucionar problemas cada vez mais complexos.

Existem diversas classes de sistemas computacionais que podem ser caracterizadas como arquiteturas reconfiguráveis, tais como os descritos em [PAG96], [SAN99], [WIR95] e [ELD94]. Estes sistemas podem ser classificados segundo um conjunto de critérios. Nesta Seção, revisa-se duas das classificações propostas na bibliografia consultada, e se propõe alguns novos critérios e classificações de arquiteturas reconfiguráveis baseadas nestes.

3.2.2 Classificação de Page

De acordo com Page [PAG96], os FPGAs estão hoje presentes em inúmeras das implementações de hardware complexo. A função destes pode ser mudada sob controle do software, oferecendo a possibilidade de arquiteturas que se reconfiguram para apoiar uma aplicação. Estes componentes de hardware reconfiguráveis são usados em combinação com processadores tradicionais (GPPs). Combinações de GPPs e algum hardware reconfigurável implementam diferentes tipos de aplicações que podem ser classificados por diferentes critérios. Page cita quatro critérios de classificação para arquiteturas reconfiguráveis, os quais serão apresentados a seguir.

3.2.2.1 Critério Forma de Interação entre o GPP e o ASP

Este critério diz respeito ao modo como a parte reconfigurável, em geral um ASP[2], se comunica com o processador principal do sistema, que na maioria das aplicações é um GPP. O modelo de interação entre os processadores ASP e GPP prevê quatro modos de comunicação:

a) **coprocessamento:** a parte reconfigurável recebe as instruções provenientes do processador principal do sistema, de forma similar às instruções enviadas a um coprocessador;
b) **chamada de procedimento remoto (RPC, remote procedure call):** o microprocessador emite uma instrução ou sucessão de instruções que são interpretadas pela parte reconfigurável como uma chamada de procedimento remoto, semelhante ao coprocessamento, podendo operar como um sistema multitarefa;
c) **cliente-servidor:** este modo possui o algoritmo da parte reconfigurável funcionando como um processo servidor, sendo que, num dado momento, este pode ser chamado por um processo cliente no processador;
d) **execução paralela:** os processos na parte reconfigurável e no GPP são executados independentes um do outro. O algoritmo da parte reconfigurável é executado como um processo paralelo. A comunicação entre o ASP e o GPP pode acontecer, a qualquer momento, via troca de mensagens.

3.2.2.2 Critério Estrutura de Memória

O algoritmo executado por um ASP normalmente requer algum espaço variável de memória temporária para sua operação. O tamanho deste espaço pode variar desde alguns registradores até uma estrutura organizada de memória, incluindo o uso de caches. Page classificou arquiteturas reconfiguráveis quanto à estrutura de memória em:

a) **sem acesso a memória:** em algumas circunstâncias, o algoritmo executado no ASP pode operar sem memória externa. Esta situação só é aceitável quando o ASP precisa de poucos estados para sua operação, pois a maioria dos ASPs atuais tem relativamente pouca memória interna disponível;
b) **acesso compartilhado:** o ASP pode usar qualquer memória associada ao barramento por ele compartilhado. Contudo, isto envolve tempo e sobrecarga no espaço de memória, necessitando desta forma de uma estrutura de arbitragem de acesso à memória. Pode ser necessário existir um controlador de endereçamento, o que tende a reduzir a velocidade de acesso a esta memória para processamento intensivo de dados;
c) **acesso à memória local:** o algoritmo da parte reconfigurável é organizado com uma memória local, necessitando de um mínimo de ciclos de relógio para cada dado armazenado. Isto aumenta o desempenho durante um acesso à memória. Neste caso, contudo, pode existir duplicação de dados em relação ao GPP. Logo, também é necessário um controle de coerência e um protocolo para troca de mensagens.

[2] Como já foi citado antes, neste trabalho considera-se ASPs implementados usando hardware reconfigurável.

3.2.2.3 Critério Forma de Operação da Memória Local do ASP

Segundo o presente critério, a memória local do ASP pode ser usada de três formas diferentes:

a) **memória de bloco:** a memória local é grande o bastante para conter o conjunto de dados completo para processamento. Por exemplo, a memória poderia conter uma página completa de vídeo a ser processada por um algoritmo de compressão de imagens.
b) **memória de fila (FIFO):** a memória age como fila, enquanto dados e resultados são trocados com o GPP. Um exemplo seria o armazenamento da região completa de apoio para um filtro de tempo real.
c) **cache**: a memória local pode ser vista como uma "cache" sobre uma estrutura de dados maior, controlada pelo GPP. Esta pode ser administrada através de troca preditiva de dados pedidos no microprocessador, ou através de métodos tradicionais, tais como os métodos do tipo cache-line-on-demand [HEN96].

Salienta-se que as classificações baseadas nos critérios desta e da Seção 3.2.2 não são necessariamente mutuamente exclusivos, ou seja, os critérios não são ortogonais, interferindo uns nos outros.

3.2.2.4 Critério - Forma de Execução do Algoritmo pelo ASP

Page menciona a existência de cinco formas segundo as quais um algoritmo pode ser implementado numa arquitetura reconfigurável. Estas formas diferem no suporte à execução de algoritmos, explorando diferentes partes da relação custo/desempenho de implementação:

- **hardware puro** (Figura 3.3)**:** O algoritmo é convertido, totalmente para hardware, através de uma ferramenta de síntese. Isto gera um circuito que realiza integralmente a mesma função do algoritmo original e que será implementado no ASP. Esta é a tecnologia na qual tudo é construído sem a necessidade de qualquer outro elemento extra para conter tal como um GPP o código executável.

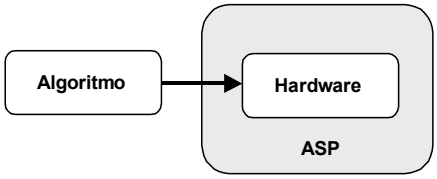


Figura 3.3 – Classificação de arquiteturas reconfiguráveis segundo o critério forma de execução do algoritmo pelo ASP: Hardware puro.

- **processador de aplicação específica:** o algoritmo é compilado em um código de máquina abstrato para um processador abstrato, e os dois são então co-otimizados para produzir a descrição de um ASP propriamente dito e um programa em código de máquina para execução. Este último é um software carregado na memória do ASP conforme ilustrado na Figura 3.4. A descrição do processador é sintetizada então para uma implementação na parte reconfigurável usando técnica de *hardware/software codesign* [MIC96].

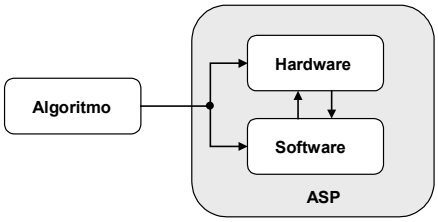


Figura 3.4 - Classificação de arquiteturas reconfiguráveis segundo o critério forma de execução do algoritmo pelo ASP: Processador de aplicação específica.

- **reuso seqüencial:** o algoritmo alvo pode ser muito grande para implementar no dispositivo reconfigurável disponível. Particiona-se o algoritmo, de forma a configurar um bloco de partição sobre o ASP a cada intervalo de tempo, ora utilizando um ou outro bloco, conforme necessário (ver Figura 3.5). As vantagens advindas da reutilização da uma mesma parte reconfigurável devem ser, neste caso, avaliadas contra o tempo gasto para a reconfiguração.

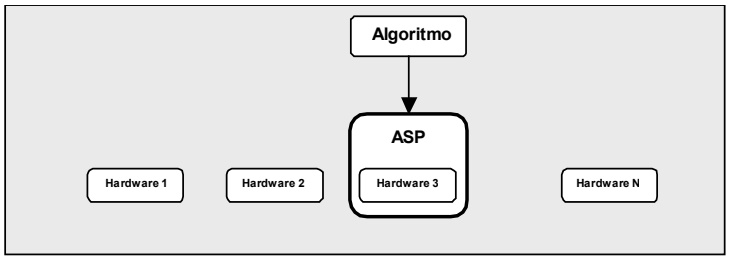


Figura 3.5 - Classificação de arquiteturas reconfiguráveis segundo o critério forma de execução do algoritmo pelo ASP: Reuso seqüencial.

- **uso múltiplo simultâneo**: se os recursos da parte reconfigurável são consideravelmente extensos, é possível a vários algoritmos coexistirem de forma residente, executando suas funções simultaneamente, cada um deles trocando dados entre si ou interagindo com o processador hospedeiro (GPP), conforme mostra a Figura 3.6.

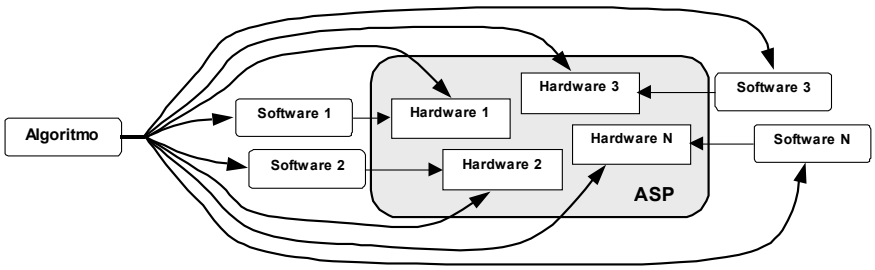


Figura 3.6 - Classificação de arquiteturas reconfiguráveis segundo o critério forma de execução do algoritmo pelo ASP: Uso múltiplo simultâneo.

- **uso sob demanda:** aqui há uma coleção relativamente grande de circuitos que podem ser carregados na parte reconfigurável e a atividade atual do sistema depende, a qualquer momento, de um subconjunto destes circuitos (ver Figura 3.7). Analogamente a sistemas de memória virtual, podemos nos referir a esta arquitetura como a um hardware virtual. O uso sob demanda cria a necessidade de estruturas complexas, onde o tempo real exige-se um controle eficiente de qual sistema de hardware deve ser construído para a necessidade do momento.

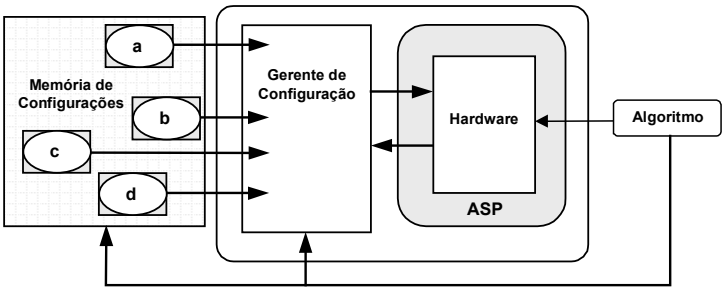


Figura 3.7 - Classificação de arquiteturas reconfiguráveis segundo o critério forma de execução do algoritmo pelo ASP: Uso sob demanda.

3.2.2.5 Considerações sobre a Taxonomia de Page

Page classifica arquiteturas reconfiguráveis a partir de características comportamentais e estruturais. Sua classificação dá um enfoque que privilegia processadores de aplicação específica (ASP) [PAG96]. Além disto, os critérios utilizados nas Seções 3.2.2.3 e 3.2.2.2 não são ortogonais. Observa-se que os modelos propostos compõem-se de uma parte reconfigurável e uma parte baseada em um processador de propósito geral (GPP).

3.2.3 Classificação de Sanchez

Analisando arquiteturas reconfiguráveis de um outro ponto de vista, Sanchez [SAN99] classificou-as levando em consideração os critérios número de configurações possíveis para o circuito e o instante de mudança de configuração. Ele dividiu as arquiteturas reconfiguráveis em duas grandes classes de reconfigurabilidade, discutidas nas Seções a seguir.

3.2.3.1 Sistemas Estáticos

Nesta classe de arquiteturas, existe pouca ou nenhuma flexibilidade. O sistema é desenvolvido para executar somente uma aplicação específica. Duas subclasses podem ser identificadas:

- **configuráveis estaticamente:** o circuito possui apenas uma configuração, que nunca é alterada, nem durante e nem após o processamento. Ou seja, o componente é configurado integralmente para realizar apenas uma função que não é mais alterada durante toda a atividade do sistema. Estritamente falando, esta subclasse não constitui uma arquitetura reconfigurável.
- **reconfiguráveis estaticamente:** há várias configurações possíveis para o circuito, Entretanto, a reconfiguração da arquitetura ocorre apenas ao final do processamento de uma dada tarefa. Os dispositivos programáveis podem ser reconfigurados integral (a forma mais comum) ou parcialmente.

3.2.3.2 Sistemas Dinâmicos

É nesta classe de aplicações que a arquitetura reconfigurável é utilizada da maneira mais flexível. Em um sistema **reconfigurável dinamicamente** existem várias configurações para o circuito, e a reconfiguração ocorre durante a execução das tarefas do sistema (em inglês, *run-time reconfiguration*). Para a implementação, podem ser utilizados tanto dispositivos parcialmente reconfiguráveis, quanto dispositivos totalmente reconfiguráveis com tarefas de reconfiguração e execução alocadas em instantes de tempo diferentes. Neste último caso, enquanto alguns

dispositivos executam, outros estão sendo reconfigurados e vice-versa obrigando, no caso de uso de dispositivos totalmente reconfiguráveis, a existência de mais de um destes dispositivos na composição do sistema.

3.2.3.3 Considerações sobre a Taxonomia de Sanchez

Na análise de arquiteturas reconfiguráveis feitas por Sanchez em [SAN99], descreve-se quatro projetos de arquiteturas reconfiguráveis, separando-os nas duas classes distintas descritas acima. O autor considera dois sistemas estáticos: SPYDER, um sistema de desenvolvimento para um processador reconfigurável e RENCO, uma rede de computadores reconfiguráveis, além disso o autor estuda classificando-os como dois outros sistemas dinâmicos: Firefly, uma máquina evolutiva, e o BioWatch, um relógio capaz de consertar a si próprio. Estes sistemas enquadram-se nas classes discutidas anteriormente. A classificação de Sanchez é estritamente comportamental, observa os sistemas quanto à dinamicidade da sua reconfiguração. Comparada à classificação de Page, a última não distingue as arquiteturas reconfiguráveis do ponto de vista estrutural.

3.2.4 Proposta de Novos Critérios de Classificação

Tanto a classificação de Page [PAG96] quanto a de Sanchez [SAN99] baseiam-se em diferentes critérios, tais como esquema de comunicação entre GPPs e ASPs, a arquitetura de memória, os modos de execução e as classes de reconfigurabilidade dos projetos. Os critérios utilizados nem sempre são ortogonais.

Com o objetivo de apresentar um estudo comparativo do tema arquiteturas reconfiguráveis, propõe-se aqui um conjunto adicional de critérios, observando-se a reconfigurabilidade de outro ponto de vista. Sob esta nova óptica utilizar-se-ão critérios orientados ao modo como a configuração é implementada nas arquiteturas reconfiguráveis. Gera-se novas classificações, que podem ser usadas de forma alternativa ou complementar às classificações apresentadas anteriormente.

As novas classificações serão baseadas em quatro critérios e definidas sob o ponto de vista do conjunto de dados de configuração, normalmente um vetor de bits. Este vetor de bits é aqui denominado *vetor de configuração*. Outro ponto vista a ser empregado é o modo como o vetor de configuração implementa a funcionalidade, ou seja, a configuração do hardware.

3.2.4.1 Critério Número de Vetores de Configuração

Baseado neste critério, é possível distinguir duas classes de arquiteturas reconfiguráveis:

a) Simples, aquelas onde um vetor de configuração específico é carregado uma única vez na arquitetura configurando-a para uma única aplicação;
b) Múltiplas, caracterizadas por usarem mais de um vetor de configuração durante a execução. Cada nova aplicação corresponde a uma nova configuração, o que implica uma arquitetura reconfigurável.

3.2.4.2 Critério Dimensão do Vetor Mínimo de Configuração

Este critério considera a quantidade de bits que compõem um vetor de configuração válido qualquer, comparado com o tamanho máximo deste. Implica duas classes de arquiteturas reconfiguráveis:

a) Total, quando qualquer vetor de configuração válido possui todos dados para configurar completamente o dispositivo;

b) Parcial, quando, ao contrário da classe anterior, existem vetores de configuração válidos que não configuram todo o dispositivo, mas apenas parte deste. De acordo com o tamanho mínimo do vetor de configuração, pode-se ainda classificar vetores de configuração parciais em duas subclasses: os de *grão pequeno* e os de *grão grande.*

3.2.4.3 Critério Ordem de Configuração

Refere-se à ordem na qual um conjunto de vetores de configuração é carregado no ASP. O critério implica três classes de arquiteturas reconfiguráveis:

a) Serial, quando um conjunto de vetores configura a arquitetura para diversas aplicações. Estas são utilizadas uma a uma, de forma seqüencial em uma mesma arquitetura;
b) Circular, semelhante à classificação Serial, o conjunto de vetores usado para configurar a arquitetura é limitado, com a particularidade que as tarefas especializadas são executadas de forma seqüencial e repetitiva;
c) Aleatória, não apresenta uma ordem predefinida para a carga de diferentes vetores de configuração.

3.2.4.4 Critério Atividade da Configuração

Este último baseia-se no tipo de atividade que o vetor impõe na configuração da arquitetura, permitindo ou não sua alteração durante o uso. Para este critério, tem-se duas classes possíveis de arquiteturas reconfiguráveis:

a) Passiva, o vetor configurado na arquitetura permanece inalterado até que uma nova configuração seja carregada por uma entidade externa;
b) Ativa, o vetor de configuração da arquitetura é reconfigurado pelo próprio hardware que ele define durante o uso.

A última classe implica a capacidade de reconfiguração dinâmica da arquitetura, permitindo que o hardware execute novas funções, mantendo o seu funcionamento normal durante a reconfiguração. Um exemplo desta classe de atividade de configuração é apresentada, mais adiante, na Seção 3.3.2 ao discutir a arquitetura DISC [WIR95].

Os critérios propostos e as classificações por eles implicadas estão resumidos na Tabela 3.1.

Tabela 3.1 - Classificação de arquiteturas reconfiguráveis baseadas nos novos critérios propostos.

CRITÉRIO	CLASSIFICAÇÕES
Número de vetores de configuração	Simples (Arquitetura Configurável) Múltipla (Arquitetura Reconfigurável)
Dimensão do vetor mínimo de configuração	Total Parcial (grão grande ou grão pequeno)
Ordem de configuração	Serial Circular Aleatória
Atividade da configuração	Passiva Ativa

3.3 Estudos de Caso de Arquiteturas Reconfiguráveis

As primeiras arquiteturas reconfiguráveis surgiram a partir do final da década de 80. Estas arquiteturas foram desenvolvidas para aplicações específicas, normalmente uma classe de problemas bem definida, tais como o processamento de imagens [WIR95], aritmética especializada, criptografia [LUC86] e processamento de informações genéticas [GOK90]. Algumas destas arquiteturas permitiram resolver alguns problemas com maior desempenho que arquiteturas genéricas.

Nas Seções a seguir serão apresentados quatro exemplos de arquiteturas reconfiguráveis, oriundas de trabalhos acadêmicos, para ilustrar as classificações propostas na Seção 3.2.4.

3.3.1 PRISM - Processor Reconfiguration through Instruction-Set Metamorphosis

O PRISM [ATH93] é um processador cuja estrutura geral é ilustrada na Figura 3.8. Esta arquitetura consiste de um compilador de configurações, interativo e especializado, que recebe como entrada uma especificação sistêmica em linguagem de alto nível.

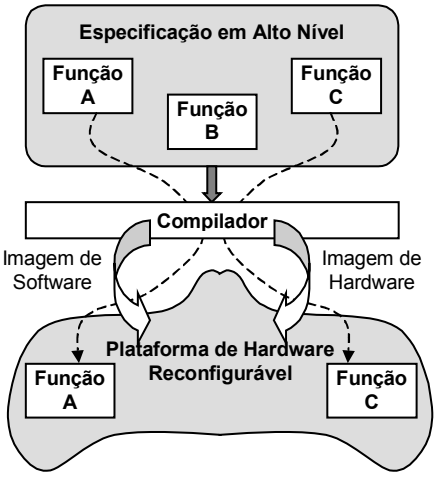


Figura 3.8 - Arquitetura reconfigurável PRISM.

O compilador analisa a especificação e apresenta ao usuário uma lista funções candidatas a serem sintetizadas em hardware. O usuário indica quais funções aceitar para produzir a imagem de hardware. A imagem de hardware consiste em especificações configuráveis. Por outro lado, a imagem de software é transformada para uma imagem executável, produzida por um compilador convencional.

Existe um conjunto de ferramentas de desenvolvimento que, dada uma especificação, instruções são sintetizadas em software e/ou hardware para o processador, sendo que esse processo é repetido para cada nova aplicação. Dois protótipos, o PRISM-I e o PRISM-II foram construídos, utilizando FPGAs Xilinx, baseadas nos dispositivos XC3090 e XC4010, respectivamente.

Na PRISM, a cada nova aplicação corresponde uma reconfiguração. Isto implica em uma arquitetura reconfigurável classificada como Múltipla. O processo de particionamento do sistema em hardware e software no PRISM ocorre a partir de uma especificação de alto nível, escrita em linguagem C, contendo as funções candidatas a serem sintetizadas. O compilador constrói duas descrições, uma que implementa a imagem de hardware e outra a imagem de software, contendo os conjuntos de procedimentos ou funções respectivos da especificação. Estes são usados para

configurar o hardware e gerar o código executável para a plataforma de hardware reconfigurável. O tempo de compilação e configuração da arquitetura é da ordem de unidades a dezenas de minutos, dependendo da função a ser implementada. A classificação de configuração Total é aplicável, pois o vetor de configuração, apesar de duplo, reconfigura a plataforma como um todo. Cada nova função é implementada em hardware através de intervenção do usuário, constituindo uma arquitetura com ordem de configuração Aleatória. Quanto à atividade da configuração, uma vez que a mesma é configurada, permanece inalterada, qualificando a arquitetura como Passiva.

3.3.2 DISC - Dynamic Instruction-Set Computer

O DISC [WIR95] é um processador programável, capaz de carregar novas instruções no hardware reconfigurável onde é implementado, conforme a necessidade específica de uma aplicação. A carga de instruções é feita por demanda, e a estrutura geral da arquitetura é representada na Figura 3.9. O projeto utiliza FPGAs CLAy da National, que são não apenas reconfiguráveis dinamicamente (baseados em SRAM) mas também reconfiguráveis parcialmente, o que viabiliza a abordagem de manter o hardware em funcionamento durante sua alteração.

Um exemplo de aplicação DISC na implementação de um algoritmo de refinamento de imagens, alcançou melhorias de até 10 vezes em relação à mesma implementação em software num PC486 a 66 MHz [WIR95].

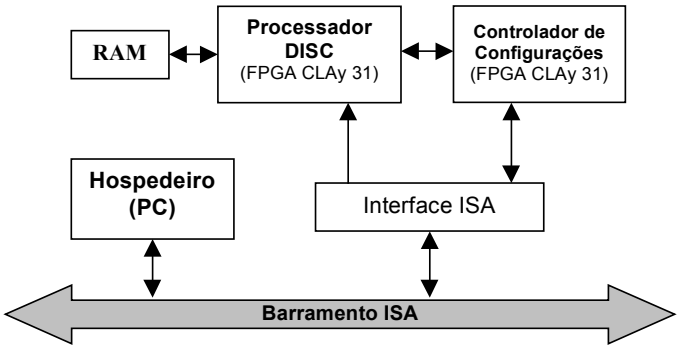


Figura 3.9 - Estrutura geral arquitetura DISC

Observa-se, a partir da Figura 3.9, que o DISC possui um controlador de configurações. Este controlador dinamicamente altera a configuração do processador, carregando o hardware para executar novas instruções e/ou substituindo hardware de instruções não usadas. Isto implica uma arquitetura reconfigurável Múltipla. Os recursos de hardware específicos para cada instrução são implementados como uma configuração Parcial. Isto é feito configurando cada instrução individualmente no DISC.

O hardware é reconfigurado usando conjuntos de linhas contíguas de blocos lógicos dentro do FPGA que implementa o processador. A unidade de reconfiguração mínima é uma linha horizontal de blocos lógicos do FPGA, ilustrada na Figura 3.10. A reconfiguração só ocorre para atender demandas do programa de aplicação, caracterizando desta forma uma arquitetura com as classificações Aleatória e Ativa. As linhas verticais na Figura 3.10 representam barramentos de comunicação entre recursos de hardware específicos para cada instrução e o controlador global, configurado de forma permanente na arquitetura.

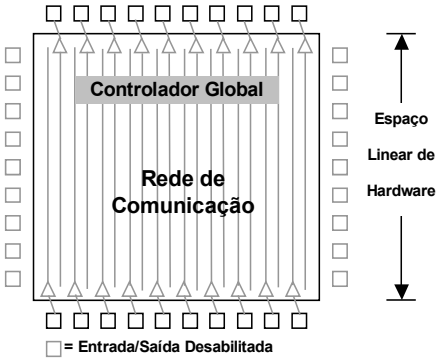


Figura 3.10 - Descrição física da distribuição interna de recursos de hardware do processador DISC.

3.3.3 SPLASH

SPLASH [GOK91, GOK90] é um arranjo linear sistólico reconfigurável desenvolvido pelo Supercomputing Research Center (SRC), em 1988. Sua principal área de aplicação foi em Biologia, oferecendo um bom desempenho em tarefas como a comparação e emparelhamento de seqüências de DNA. O arranjo do SPLASH possui 32 estágios, conforme detalhado na Figura 3.11. Cada estágio é composto de um FPGA Xilinx XC3090 e uma memória SRAM de 128KB. A segunda versão do SPLASH, o SPLASH-2, é um processador matricial reconfigurável mais flexível que o SPLASH, e foi desenvolvido em 1992 [BUE96].

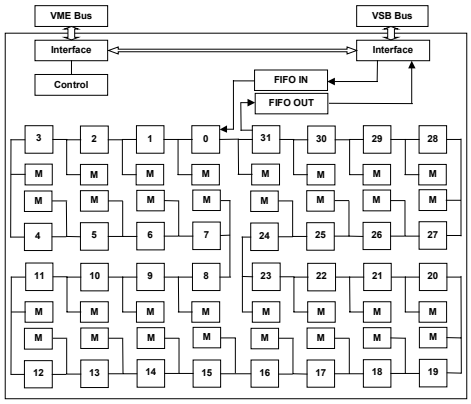


Figura 3.11 - Arquitetura SPLASH, mostrando as interfaces VME, VSB e a matriz de 32 estágios.

O hardware do SPLASH, representado na Figura 3.12, consiste de duas placas conectadas a um computador hospedeiro (SUN/UNIX), que compartilham um barramento VSB. Uma das placas é o processador SPLASH ilustrado na Figura 3.11, com uma interface VME para o hospedeiro, uma interface VSB para organizar a memória e uma matriz linear de 32 estágios. O outro processador é uma placa com 8 Mbytes de memória conectada aos barramentos VSB e VME.

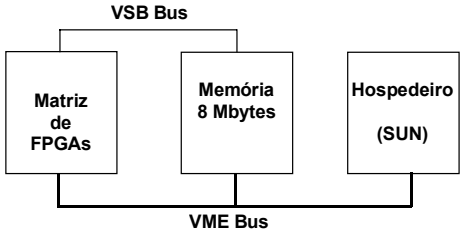


Figura 3.12 - Sistema SPLASH.

A máquina SPLASH é configurada a partir das imagens individuais para cada FPGA, conectada em uma cadeia de 32 estágios. A configuração é realizada através do barramentos VME quando o

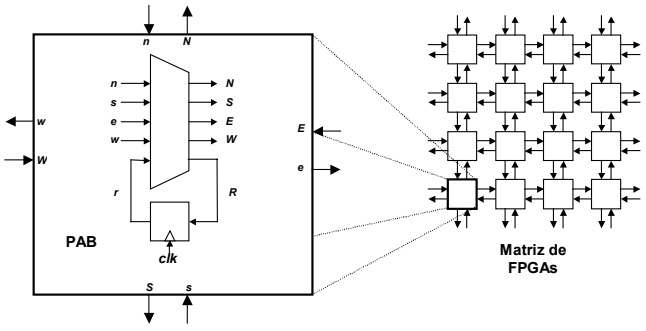
SPLASH é inicializado. Paralelamente, um programa supervisor controla a execução dos algoritmos sistólicos no SPLASH. Este programa é de autoria do usuário, sendo executado na estação de trabalho SUN.

Com estas características de reconfiguração, podemos classificar esta arquitetura como Múltipla, pois pode ser reconfigurada a cada novo algoritmo. Uma linguagem interpretada, denominada *Trigge*], permite aos usuários carregar e depurar os programas. Esta linguagem permite a acesso a uma biblioteca básica de procedimentos, bem como o acesso a funções escritas pelos usuários, e permite que o SPLASH seja visto realmente como um conjunto de 32 processadores.

No SPLASH, cada FPGA pode ser configurado através da interconexão serial de todos os processadores. Deste modo, o SPLASH tem a propriedade de reconfigurabilidade Parcial. Esta propriedade é melhor explorada no SPLASH 2, onde é possível a reconfiguração da arquitetura da própria cadeia de interconexão. A escolha do algoritmo sistólico é realizada de acordo com a área de aplicação. Desta forma, a ordem de reconfiguração não é determinada, classificando a arquitetura como Aleatória. Uma vez que as rotinas são definidas e o algoritmo configurado na matriz, esta permanece inalterada até uma nova aplicação, caracterizando uma arquitetura de classificação Passiva.

3.3.4 DEC-Perle

DEC-Perle [BER89] é um projeto desenvolvido pelo DEC PRL (*Paris Research Lab* da *Digital Equipament Corporation*). Trata-se de implementações de uma arquitetura genérica denominada PAM (*Programmable Active Memories*), uma matriz de células idênticas conectadas ortogonalmente, onde cada célula é um PAB (*Programmable Active Bit*), como ilustrado na Figura 3.13. Cada PAB pode realizar simultaneamente funções Booleanas quaisquer de até 5 entradas. Um protótipo de arquitetura reconfigurável baseada em PAMs, denominado DEC-Perle-0, foi construído e testado utilizando uma matriz de 25 FPGAs Xilinx XC3020. Entre as aplicações implementadas nesta arquitetura reconfigurável estão problemas como aritmética especializada (implementação de multiplicadores complexos, com até 512 bits), compressão de dados e processamento de imagens. Esse protótipo foi sucedido pelo DEC-Perle-1, que utiliza uma matriz de 24 FPGAs XC3090.



Cada PAB tem 4 entradas (n,s,e,w), 4 saídas (N,S,E,W), um registrador (flip-flop) com entrada R e saída r, e um circuito combinacional g(n,s,e,w,r) = (N,S,E,W,R).

A tabela verdade de g é especificada por 160 = 5 x 32 bits

Figura 3.13 - Arquitetura da célula básica da arquitetura DEC-Perle, mostrando a matriz de FPGAs e detalhe da célula PAB.

A estrutura genérica de uma PAM é vista na Figura 3.14. Ela é conectada por uma interface de entrada e saída a um processador hospedeiro. Uma das funções do hospedeiro é carregar o vetor de configuração na PAM. Observa-se que uma PAM é conectada a um barramento de alta velocidade

do computador de hospedeiro, como qualquer módulo de memória de RAM. O processador pode escrever e/ou ler da PAM, porém ao contrário de uma RAM, uma PAM processa dados entre instruções de escrita e leitura, caracterizando PAMs como *memórias ativas*.

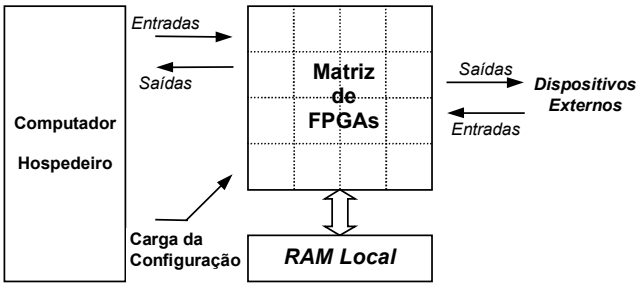


Figura 3.14 - Arquitetura genérica de um processador baseado em PAMs.

Em algumas aplicações, tais como criptografia, não é necessário o uso de memória local. Contudo a maioria das aplicações exigem alguma quantidade de memória para reordenar dados.

Nas PAMs, um programa é composto de três partes:

- O software, que controla o hardware e é executado no computador hospedeiro;
- A configuração dos blocos lógicos, que descreve o hardware síncrono implementado na PAM;
- As diretivas de posicionamento e roteamento, que orientam a implementação da lógica no interior do PAM.

O software de controle é escrito em C/C++ e é integrado a uma biblioteca que encapsula o "driver" do dispositivo. A configuração dos blocos lógicos e as diretivas de posicionamento e roteamento são geradas no programa em C++. Um FPGA extra implementa o *Firmware* (interface de alta velocidade, 1.2Gbps). Este, analogamente a uma ROM, não pode ser configurado pelo usuário e sua função é prover, através do hospedeiro, o controle da PAM, bem como o protocolo para ajudar a reconfiguração da PAM durante execução.

O propósito da PAM é implementar uma máquina virtual, que pode ser configurada dinamicamente em um grande número de dispositivos de hardware específicos. Isto demonstra uma arquitetura configurável que pode ser classificada como Simples. Depois da configuração, a PAM comporta-se, elétrica e logicamente, como um ASIC (em inglês, *Application-Specific Iintegrated Circuits*) definido por um vetor de configuração específico. Desta forma, a classificação para a dimensão do vetor é Total. Ela pode operar de forma autônoma, conectada a algum dispositivo externo ou operar como um co-processador sob controle do hospedeiro. A PAM também pode operar de ambas formas, autônoma ou coprocessador, conectada ao hospedeiro e a algum dispositivo externo, como um dispositivo de áudio ou de vídeo, ou alguma outra PAM. Não existe ordem de reconfiguração definida. Portanto, pode-se classificar a arquitetura da PAM, quanto a ordem de configuração, como Aleatória. Quanto a atividade, PAMs podem ser classificadas como de configuração Ativa ou Passiva dependendo da aplicação.

3.4 Plataformas de Prototipação Rápida para Sistemas Digitais

3.4.1 Introdução

Como foi visto na Seção 0, arquiteturas reconfiguráveis são uma nova forma de compor tecnologias para implementar sistemas computacionais. Estas combinam a programabilidade dos GPPs com a flexibilidade de hardware reconfigurável. O resultado desta combinação muda os limites de flexibilidade do hardware e do software. Nesta Seção, estuda-se uma classe especial de arquiteturas reconfiguráveis, as plataformas de prototipação rápida para sistemas digitais. Estas plataformas são orientadas ao desenvolvimento de projetos de sistemas digitais para as mais diversas aplicações. Esta característica dá ao projetista de sistemas novos graus de liberdade, não disponíveis em dispositivos convencionais. O hardware destas arquiteturas pode ser mudado tão facilmente quanto o software de um processador, e pode até mesmo ser alterado múltiplas vezes para executar funções diferentes em diferentes situações [BRO92]. Esta classe de arquiteturas reconfiguráveis facilita a depuração de sistemas digitais e são uma excelente forma para *prototipar* sistemas, devido à facilidade de alterá-los. As plataformas de prototipação rápida trazem com isto um benefício importante, a redução do tempo necessário para um novo produto chegar ao mercado.

Aqui são revisadas três plataformas comerciais disponíveis para a realização deste trabalho. O número de plataformas existentes nas áreas comercial e de pesquisa é extremamente elevado, sendo difícil abordar todas existentes em único estudo. Repositórios dedicados a coletar informações sobre uma maior quantidade destas plataformas existem em http://www.io.com/~guccione/HW_list.html/, uma página gerenciada por Steve Guccione e http://www.optimagic.com/boards.html, uma página da empresa Optimagic.

3.4.2 Plataforma de Prototipação Virtual Workbench

A plataforma de prototipação *Virtual Workbench (VW300),* da empresa *Virtual Computer Corporation* (VCC) proporciona um meio para o desenvolvimento de projetos com base em FPGAs de alta densidade [VIR99]. Esta plataforma é uma placa construída em torno de um FPGA XCV-300 da família Virtex da empresa Xilinx. Esta placa possui os seguintes recursos:

- 1 FPGA XCV-300 da Xilinx com capacidade para implementar sistemas digitais com capacidade para até cerca de 300.000 portas lógicas equivalentes;
- 2Mbits de Memória Flash;
- 256Kx18 bits de SRAM;
- 8 Mbytes de SDRAM;
- 1 módulo gerador de relógio programável (de 360KHz até 100MHz);
- 2 osciladores fixos com freqüências de 33 MHz e 50 MHz, respectivamente;
- 1 porta serial assíncrona RS-232 e uma interface para monitorar temperatura;
- 8 Displays matriz de pontos, 8 LEDs e um bloco de 8 chaves Dip Switch;
- 2 conectores para configuração do FPGA, usando interfaces JTAG, Xchecker ou MultiLinx.

A plataforma permite razoável flexibilidade em conexões com o ambiente externo. Os sinais de entrada e saída, presentes nos pinos do FPGA, podem ser facilmente analisados a partir de dois conectores (I/O MOD 1 e 2, Figura 3.15 à direita) e utilizados para interfaceamento com outros dispositivos externos, tais como um conversor analógico digital ou um analisador lógico.

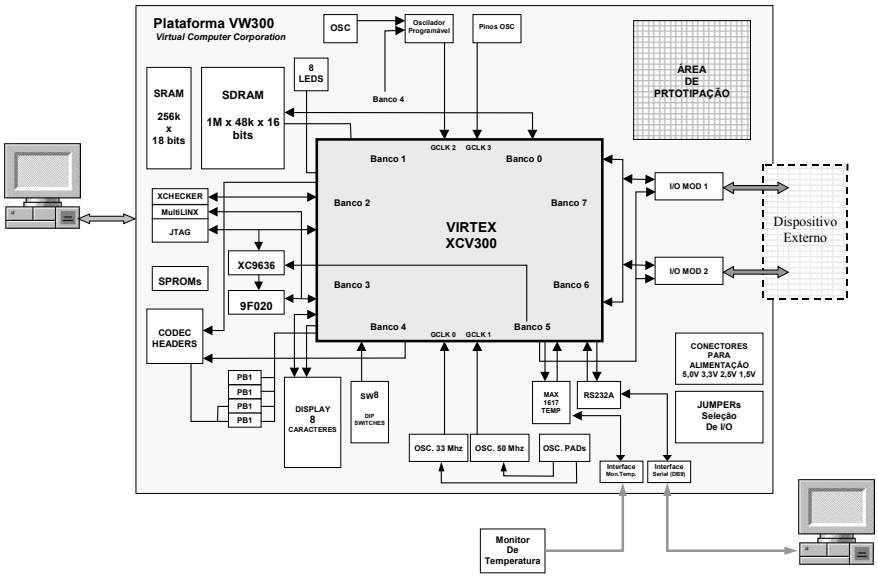


Figura 3.15. Diagrama de blocos da plataforma VW300.

O dispositivo XCV300, o coração da plataforma de prototipação VW300 possui uma matriz 32x48 de CLBs, ou seja, 6 912 células lógicas, até 316 portas de entrada/saída programáveis e 64 Kbits de memória RAM distribuídos em duas colunas de bancos de 4 Kbits (BRAMs na Figura). Os blocos de memória RAM podem ser configurados como memória de porta simples ou porta dupla para escrita e leitura síncronas. Estes blocos são ainda configuráveis para diferentes larguras (bits de dados) e diferentes profundidades (número de palavras).

3.4.3 Plataforma HOT2-XL

A plataforma de prototipação HOT2-XL mostrada esquematicamente na Figura 3.16, também da empresa VCC [VIR98], é dotada de características bem distintas. A sua principal diferença em relação à plataforma VW300 reside no fato dela possuir um barramento PCI, possibilitando seu acoplamento ao computador hospedeiro por uma via de alto desempenho. Utiliza um FPGA XC4062XLT da Xilinx (62 mil portas equivalentes), um gerenciador de configuração da memória "cache" (Configuration Cache Manager-CCM), dois bancos de 2Mbytes de RAM estática, dois conectores para placa de expansão tipo mezanino e interface com o barramento PCI do computador hospedeiro (inclui PCI Target e Initiator). Existe uma versão HOT2 standard, usada para desenvolvimento, personalização e prototipação de projetos usando o núcleo de propriedade intelectual LogiCORE PCI da empresa Xilinx. Ela permite validar pequenos projetos (até 5000 portas lógicas) do IP core. A versão XL é ampliada para habilitar a implementação de projetos maiores, acima de 45000 portas lógicas.

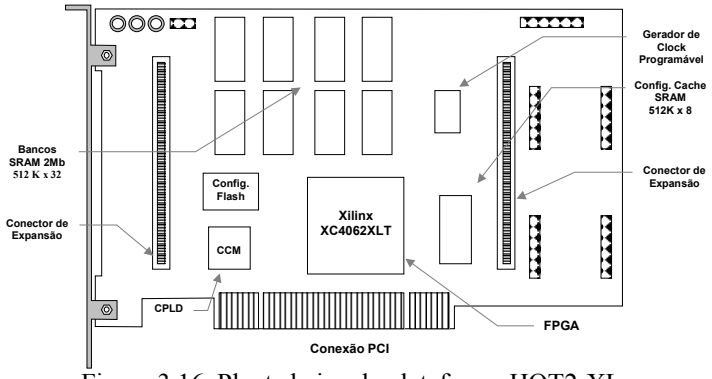


Figura 3.16. Planta baixa da plataforma HOT2-XL.

O sistema de desenvolvimento HOT2-XL oferece um método para validação de projetos com interface PCI, projetos back-end, drivers e software de controle. A avaliação pode proceder em tempo real com o respectivo software de aplicação. O principal elemento, um FPGA–XC4062XLT, representado no diagrama de blocos da Figura 3.17, contém uma Macro Interface Xilinx PCI LogiCORE (Núcleo PCI de propriedade da Xilinx), personalizado pelo fabricante da plataforma, direcionando a comunicação via dois bancos independentes de RAM estática, cada um, organizado em 112 K palavras de 32 bits.

Após a alimentação elétrica da placa ser estabelecida, o FPGA é inicializado a partir da Macro Interface HOT2 (Configurador de Partida na Figura 3.17), que contém o Xilinx PCI LogiCORE. O CCM controla em tempo real a configuração, bem como a reconfiguração do sistema. Ele pode configurar o FPGA a partir de duas fontes, da Flash de Configuração ou da RAM de Configuração. Quando o FPGA torna-se ativo, o CCM sinaliza ao software de controle, o qual faz a recarga dos cabeçalhos de informações PCI no Núcleo PCI (LogiCORE PCI Core).

Um banco de RAM de configuração pode armazenar até três vetores de configuração para o FPGA XC4062XLT, no caso da plataforma HOT2-XL. O usuário pode carregar a RAM de configuração a partir do barramento PCI. Através de uma interface de aplicação de hardware (Hardware Object Technology API), fornecida pelo fabricante da plataforma, as configurações são controladas e carregadas. Os vetores de configuração são convertidos por um programa codificador para serem carregados na plataforma, mediante comandos simples na interface de aplicação. A plataforma PCI HOT2-XL também possui dois barramentos independentes de 32 bits de dados e 24 Bits de endereços, com respectivos conectores, para acesso externo.

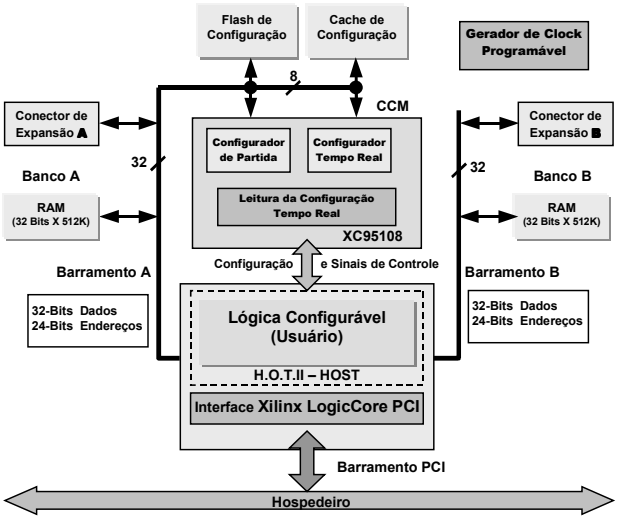


Figura 3.17. Diagrama de blocos da plataforma HOT2-XL.

3.4.4 Plataforma XSV

A plataforma XSV é centrada em um FPGA Xilinx da família Virtex. A placa pode alojar um FPGA Virtex com capacidade para implementar sistemas digitais compostos de 50.000 (usando o dispositivo XCV50) até 800.000 (usando o dispositivo XCV800) portas lógicas. O FPGA é um componente de 240 pinos, sendo ele o principal repositório para a lógica programável da plataforma. Um dispositivo CPLD XC95108, instalado na plataforma serve de controlador da configuração e do fluxo de dados entre o FPGA e o computador hospedeiro.

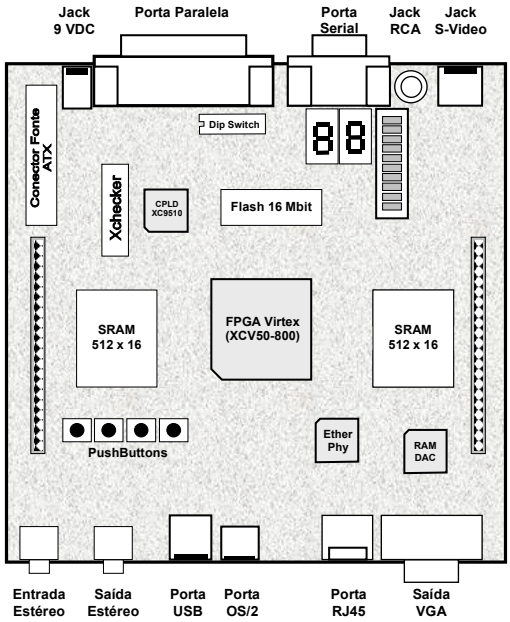


Figura 3.18. Vista simplificada da plataforma XSV.

A plataforma XSV [XES00] está representada na Figura 3.18. Seu projeto é dedicado, sobretudo, ao processamento de som e imagem. Ela possui recursos para o processamento de sinais de áudio e vídeo nos sistemas PAL, SECAM ou NTSC com a resolução de 9 bits para os canais RGB, bem como gerar sinais de vídeo (110 MHz, 24 bits RAMDAC). A XSV também permite o processamento de sinais de áudio estereofônicos, com uma amostragem de até 20 bits e uma banda de 50 KHz.

Na plataforma XSV, estão disponíveis 2 bancos independentes de 512K x 16 de SRAM para uso geral. Uma RAM Flash de 16 Mbits permite armazenar múltiplos vetores de configuração para o FPGA e dados de uso geral aos quais o FPGA pode ter acesso.

Nesta plataforma existe uma grande variedade interfaces para comunicação com dispositivos externos. A maior parte destas interfaces são as mesmas interfaces de um computador pessoal moderno, como por exemplo:

- Portas paralela e serial, para enviar e receber dados através do CPLD;
- Portas para Mouse e Teclado (PS/2), que permitem ao FPGA ter acesso aos dispositivos comuns de entrada do PC;
- Uma porta USB, que provê o FPGA com um canal de entrada/saída serial com largura de banda entre 1.5 e 12 Mbps.
- Uma interface com a camada física, 10BASE-T/100BASE-TX Ethernet, o que possibilita à plataforma ter acesso a uma Rede Local (LAN) com taxa de transferência de até 100 Mbps.

Existem dois conectores de expansão para uso geral independentes, cada um com 38 pinos de entrada/saída diretamente conectados ao FPGA. Este conjunto de conexões implementa barramentos típicos para acessos a periféricos. Linhas de endereços, de dados e controle estão distribuídas entre os dispositivos presentes na plataforma. Desta forma, o FPGA pode ter acesso aos dois bancos de memória SRAM e à RAM Flash.

O CPLD e o FPGA compartilham o acesso à RAM Flash. Normalmente, o CPLD carrega a Flash RAM com dados enviados através da porta paralela ou serial. Caso os dados sejam o vetor de configuração, então o CPLD transfere-os para o FPGA. Quando é utilizado o cabo Xchecker para enviar o vetor de configuração ao FPGA, este permite executar as operações de configuração e "readback" (leitura da configuração existente no FPGA), útil na depuração de hardware. Também existe um oscilador programável simples que provê o sinal de relógio ao FPGA e ao CPLD. Este relógio é gerado por divisão (de 1 a 2052) de uma freqüência básica de 100 MHz, proveniente de um cristal.

- Outros dispositivos também estão disponíveis na plataforma, tais como:
- Quatro interruptores do tipo Push-button;
- Uma chave seletora de oito pontos tipo DIP switch;
- Dois Displays de 7 segmentos;
- Um conjunto de 10 LEDs tipo Bargraph.

A plataforma XSV possui uma arquitetura muito semelhante a um computador pessoal, possibilitando desta forma a prototipação rápida de núcleos processadores. Sua característica mais marcante é a flexibilidade, aliada à grande variedade de interfaces. Isto faz desta plataforma um meio atraente e facilitador para o desenvolvimento e a prototipação de sistemas digitais.

De particular interesse para o presente trabalho é o dispositivo LXT970A do fabricante Level One Communications [LEV98], disponível na plataforma XSV. Trata-se de um transceptor, semelhante aos utilizados por controladores comerciais para interface de redes locais. Este transceptor executa a tarefa de detectar colisões, dados trafegando na rede, sincronização da recepção e transmissão, bem como a codificação e decodificação de informação proveniente de pacotes Ethernet trafegando no meio físco. Estas funcionalidades habilitam o desenvolvimento de projetos de sistemas digitais em arquiteturas reconfiguráveis, dedicados ao estudo e a pesquisas de tecnologias de redes locais Ethernet. Como exemplos destas tecnologias, tem-se o protocolo de acesso ao meio para redes locais Ethernet, passível a ser implementado e validado na plataforma XSV. A implementação e validação deste protocolo serão discutidos em detalhes nos Capítulos 6 e 7.

4. Redes de Computadores

4.1 Introdução

As tecnologias de hardware e software usadas para redes de computadores são projetadas visando-se o compartilhamento de recursos e informações. Elas permitem que múltiplos computadores e dispositivos sejam conectados diretamente a uma rede única e compartilhada. Como o meio de comunicação é compartilhado, os computadores devem alternar-se no uso do mesmo [COM01].

Este Capítulo apresenta algumas definições básicas relacionadas a redes de computadores, o modelo de referência para interconexão de sistemas abertos e discuti alguns aspectos importantes relacionados a uma classe particular de redes, denominadas de redes locais.

Em geral as redes de computadores são classificadas dependendo do seu tamanho em nós ou elementos processadores, da abrangência geográfica e da tecnologia de comunicação. Uma classificação baseada no critério de abrangência geográfica muito usada é:

- *Rede local* (Local Area Networks – LAN) apresenta seus módulos processadores interligados através um sistema de comunicação localizado em uma área restrita. Atualmente, considera-se área restrita uma região delimitada por distâncias máximas entre 100 m e 25 Km [SOA95]. Redes locais são ainda caracterizadas pela taxa de transmissão de bits associada. As distâncias e as taxas de bits atuais mais encontradas são 0.1 a 10 Km e 10 a 100 Mbps, respectivamente. O produto destes valores pode ser usado para definir os limites desta tecnologia, embora estes sejam mutáveis ao longo do tempo. Pode-se assim caracterizar redes locais como sendo aquelas com uma vazão por limite de distância variando hoje em 1Mbps.m e 1Gbps.m [SOA95]. Com tudo o limite superior vem sendo paulatinamente ultrapassado com novas tecnologias, tais como redes locais do tipo *Fast Ethernet, ATM* (Asynchronous Transfer Mode) e as *Gigabit Ethernet*.
- *Rede metropolitana* (Metropolitan Area Network - MAN) pode abranger uma única cidade, em geral cobrem distâncias maiores que LANs e operam em velocidades maiores. Tipicamente, uma MAN consiste em sub-redes DQDB[3] (Distributed Queue Dual Bus) interconectadas, cujo propósito é prover serviços integrados, tais como texto, voz e vídeo, em uma área metropolitana [COM01].
- *Rede de longo alcance* (Wide Area Network – WAN) pode abranger locais em múltiplas cidades, países ou continentes. A WAN deve ser capaz de crescer o quanto for necessário para conectar muitos *locais* (em inglês, *sites*) espalhados através de distâncias geograficamente grandes, com muitos computadores em cada local interligando redes metropolitanas e redes locais. Como exemplo, temos a rede mundial *Internet*, que interliga redes da maioria dos países. A Internet oferece um conjunto de serviços, tais como: *FTP* (File Transfer Protocol), um protocolo para transferência de arquivos; *TELNET* que permite a um usuário de um computador estabelecer uma sessão interativa com um outro computador na rede; *SMTP* (Simple Mail Transfer Protocol) conhecido como correio eletrônico (em inglês, *e-mail)* e a *WWW* (World Wide Web) desenvolvida para permitir o acesso a informações organizadas na forma de páginas, que englobam documentos espalhados em servidores por toda a rede [SOA95].

[3] Segundo definição do grupo IEEE 802 a DQDB, padronizada pela norma IEEE 802.6, é uma sub-rede que pode ser usada como parte componente de uma MAN (poucas redes se encaixam nessa categoria)

4.2 Redes Locais (LANs)

As Redes Locais permitem a computadores por elas interligados trocar informações e compartilhar recursos interligados por conexões físicas (meio de transmissão). As conexões entre os vários *módulos processadores* ou *estações* que compõem uma rede de computadores são o *sistema de comunicação*. As conexões que compõem a LAN definem uma *topologia* [SOA95], que da a forma como as conexões físicas e os nós (estações) estão organizados. As topologias podem ser classificadas em dois tipos fundamentais, de acordo com o tipo de conexões que comportam: ponto a ponto ou multiponto, ambas ilustradas na Figura 4.1.

Topologias *ponto a ponto* são aquelas onde cada par de módulos processadores comunica-se por uma conexão física exclusiva, conforme mostrado na Figura 4.1(a). Esta topologia, em geral, facilita as funções de comunicação e o controle desta. Por outro lado, redes que conectam um grande número de módulos processadores tornam este tipo de conexão extremamente caro e de difícil manutenção. Por outro lado, topologias do tipo *multiponto*, ilustradas na Figura 4.1(b), são aquelas onde uma conexão física é compartilhada entre três ou mais módulos processadores. Esta topologia barateia o custo de hardware para implementar redes, mas em geral complica o controle da comunicação.

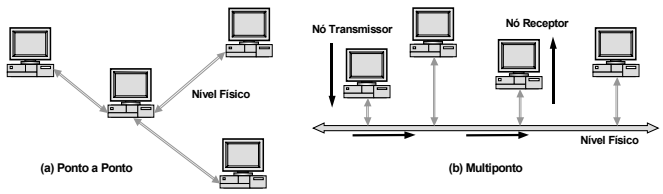


Figura 4.1 - Topologia (a) Ponto a Ponto e (b) Multiponto.

Cada uma das topologias básicas acima pode naturalmente ser combinada em topologias mistas, de forma hierárquica ou não. O controle das atividades de comunicação pode também ser dividido em dois tipos básicos: centralizado e distribuído. No *controle centralizado*, um, ou um pequeno conjunto de equipamentos é responsável por comandar as atividades de comunicação, podendo ou não este ser um dos módulos processadores da rede. No *controle distribuído*, as atividades de controle da comunicação são exercidas por um grande número de equipamentos, tipicamente todos os módulos processadores. Neste caso, todos os módulos concordam em utilizar um *protocolo* de comunicação único. O protocolo nada mais é que um conjunto de regras que regem a comunicação entre módulos processadores, mediante uso das conexões físicas que determinam a topologia da rede.

Soluções do tipo multiponto com controle distribuído da comunicação são largamente adotadas em redes locais, oferecendo as vantagens de baixo custo de implementação e a descentralização da comunicação. Esta descentralização traz consigo vantagens técnicas, tais como a diminuição da sobrecarga de processamento de informações de controle, e o aumento de tolerância a falhas da rede.

É preciso assumir a realidade de que não existe uma rede única que poderia corresponder às necessidades de todas as classes ou níveis de atividade existentes em uma organização (empresas, universidades, centros de pesquisas, etc.), sendo a solução mais utilizada, a adoção de várias redes interconectadas, cada rede servindo de suporte à comunicação no contexto de uma ou diversas atividades.

Cada nível da hierarquia da organização que faz uso de uma rede é representado por um conjunto de ações e processamentos que possuem requisitos de comunicação diferentes. A característica

predominante nos níveis hierárquicos inferiores é a transferência de mensagens curtas com alta freqüência, entre um número elevado de estações (nós ou módulos processadores). Nos níveis hierárquicos superiores há a predominância de transferência de mensagens longas entre um número menor de estações e a uma freqüência mais baixa, como representado na Figura 4.2.

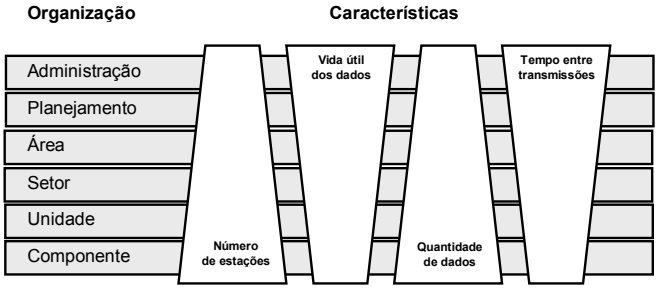


Figura 4.2 - Característica das comunicações em redes de computadores.

Deste modo, não existe um sistema de comunicação único capaz de atender a todas as aplicações existentes na organização, mas sim uma série de sub-redes locais adequada aos requisitos de comunicação de cada nível. As sub-redes serão conectadas à linha tronco (*Backbone*) através de conversores (*Gateways*), pontes (*Bridges*) e roteadores (*Routers*), de modo que todas as estações possam ter acesso umas às outras, formando um sistema de comunicação coeso que atenda toda a organização.

Dada a dificuldade de implementar comunicação eficiente entre dispositivos como computadores e aplicações complexas executando nestes, os dados transferidos em uma comunicação entre dois nós não são enviados diretamente pelas aplicações através das conexões físicas (ou *horizontalmente*). As informações a serem transmitidas evoluem *verticalmente*, através de uma seqüência de camadas de protocolos de comunicação até atingir a camada da conexão física, onde os dados são efetivamente enviados ao destino. No destino, os mesmos dados procedem de forma inversa até a aplicação alvo da informação transmitida. Uma *arquitetura da rede* é formada por *níveis* (as camadas de protocolo), *interfaces* e *protocolos*, e é definida como um conjunto de *camadas hierárquicas*, cada uma destas sendo construída utilizando as funções e serviços oferecidos pelas camadas inferiores [SOA95].

Nas seções que se seguem são apresentadas as camadas hierárquicas do modelo genérico OSI-RM, suas funções básicas, serviços, relações, bem como os padrões adotados em cada camada. A partir do Capítulo 5 discute-se o projeto, validação e a implementação de um módulo de hardware para implementar parte desta hierarquia em uma rede local com topologia multiponto e o controle da comunicação distribuído, mediante uso da tecnologia de rede conhecida como Ethernet.

4.3 O Modelo OSI-RM

O Modelo de Referência para Interconexão de Sistemas Abertos (em inglês, *Open Systems Interconnection Reference Model* ou OSI-RM), proposto pela International Standards Organization (ISO), permite a estratificação, o projeto e a implementação dos protocolos padronizados em redes de comunicação. O OSI-RM divide o tratamento de informações de rede em sete níveis de abstração. O mais baixo corresponde ao hardware, e os subseqüentes correspondem ao firmware ou software que usam o hardware, conforme enumerados e definidos a seguir:

- **Nível 1 (Físico):** define o meio de transmissão, suas características elétricas e mecânicas. Exemplo de normas aplicadas neste nível são, RS-232, RS-422A, RS-485, da EIA (Electronic Industries Association), ATM Utopia, etc;

- **Nível 2 (Enlace de Dados):** estabelece, mantém e permite acesso ao meio físico, empacota mensagens para transmissão e verifica a integridade das mensagens recebidas. Exemplo de normas e padrões aplicados a este nível são, HDLC(ISO), ADCCP(ANSI), DDCMP(DEC) e SDLC e BSC (IBM);
- **Nível 3 (Rede):** controla o encaminhamento das mensagens, gerenciando o seu fluxo dentro e entre redes. Exemplo de normas aplicadas a este nível são, X.20, X.21, X.25 da CCITT (International Consulting Committee on Telephone and Tetegraph);
- **Nível 4 (Transporte):** cria e mantém a conexão entre estações, permitindo a conversão do nível mais baixo entre estações;
- **Nível 5 (Sessão):** estabelece e controla a sessões (mensagens estruturadas) entre duas estações da rede;
- **Nível 6 (Apresentação):** proporciona uma representação e formato geral de informação trocada entre duas estações da rede;
- **Nível 7 (Aplicação):** define regras de sintaxe, semântica e gramáticas para troca de informações a nível de transporte de arquivos, acesso à bases de dados e uso compartilhado de recursos entre duas estações da rede.

A hierarquia proposta pelo OSI-RM é apresentada no lado esquerdo da Figura 4.3. Na direita da figura, comparativamente, observa-se que para os níveis 5 ao 7 ainda não existem normas e padrões bem estabelecidos. Os níveis 3 e 4 do OSI-RM podem ser equiparados ao conjunto de protocolos da Internet (TCP/IP) e os níveis 1 e 2 estão relacionados aos padrões definidos para redes locais com sua interface de rede e respectivos drivers de rede e protocolos.

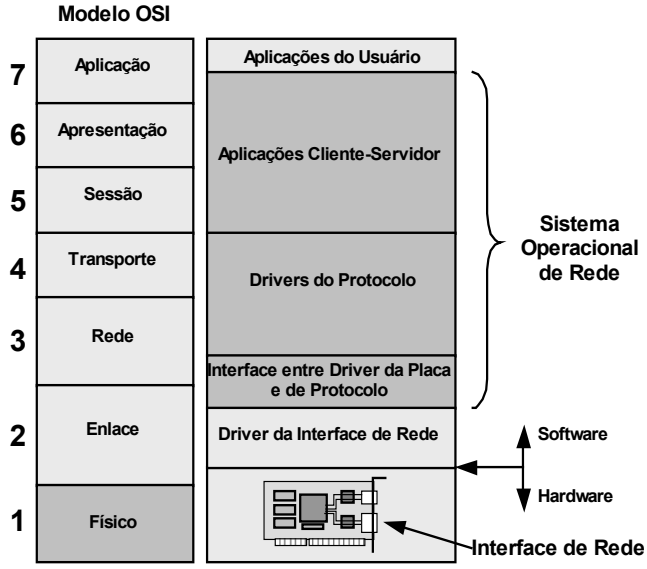


Figura 4.3 - O OSI-RM e um exemplo típico de sistema operacional de rede.

Em geral, para que um módulo processador possa operar em uma rede de computadores, devem ser instalados recursos de hardware e software que complementem seus dispositivos e seu sistema operacional local. Em uma rede local, o hardware adicional necessário se constitui, tipicamente, de uma placa de interface de rede e um software, que definimos como *sistema operacional de rede*. Os componentes típicos de um sistema operacional de rede são mostrados na Figura 4.3 à direita, comparados com o OSI-RM. Tal sistema engloba os seguintes componentes:

- As aplicações cliente-servidor de uso geral;
- Um conjunto de módulos implementando os protocolos;

- Um ou mais módulos de software básico (drivers) com implementações de protocolos de comunicação.

O controle e configuração da interface de rede são realizados pelo driver da interface de rede, que se comunica com o sistema operacional de rede através de uma interface padronizada.

4.4 A Tecnologia Ethernet e o Padrão IEEE 802.3

No início dos anos 70 começou a operar uma rede de rádio difusão via radiofreqüência denominada *Aloha*. Seu propósito era interligar o centro de computação da Universidade do Havaí, em Honolulu, a terminais espalhados por todas as ilhas do arquipélago. Embora a rede Aloha não possa ser considerada uma rede local, seu estudo é importante uma vez que de seu protocolo resultou o da tecnologia Ethernet [MET76].

A Ethernet foi inventada no Centro de pesquisas da Corporação Xerox em Palo Alto. Mais tarde, Digital Equipment Corporation, a Intel Corporation e a Xerox cooperaram para desenvolver um padrão de produção, que é informalmente chamado de *DIX Ethernet*, devido às iniciais das três companhias [COM01]. A Ethernet foi projetada para transferência de dados limitada a pequenas distâncias. Ela baseou-se originalmente no protocolo *Carrier Sense Multiple Acess with Collision Detection* (CSMA-CD).

Ao nível de redes locais, o padrão mais utilizado é o proposto pelo IEEE (Institute of Electrical and Electronic Engineers). O Projeto IEEE 802 nasceu com o objetivo de elaborar padrões para redes locais de computadores. Este projeto ficou a cargo de um comitê instituído em fevereiro de 1980 pela IEEE Computer Society. O comitê 802 publicou e continua a publicar padrões adotados nacionalmente nos Estados Unidos da América pelo American National Standards Institute (ANSI). Esses padrões são muitas vezes revisados e republicados como padrões internacionais pela ISO com a designação ISO 8802 [SOA95].

A Figura 4.4 apresenta a relação entre alguns dos principais padrões IEEE 802 e o OSI-RM. Conforme ilustrado, o padrão 802.2 especifica a compatibilidade com o nível 2 do OSI-RM, enquanto que 802.3 a 802.12 referem-se também ao nível 1 do OSI-RM. O protocolo de acesso ao meio baseado no CSMA/CD é batizado pela IEEE sob o número IEEE 802.3. Os padrões 802.4 e 802.5 consistem na especificação do protocolo tipo passagem de permissão, implementado em barramento (*token bus*) e em anel (*token ring*), respectivamente. O padrão 802.6 define um padrão para transporte de dados a alta velocidade em regiões metropolitanas e é chamado de *Distributed Queue Dual Bus* (DQDB).

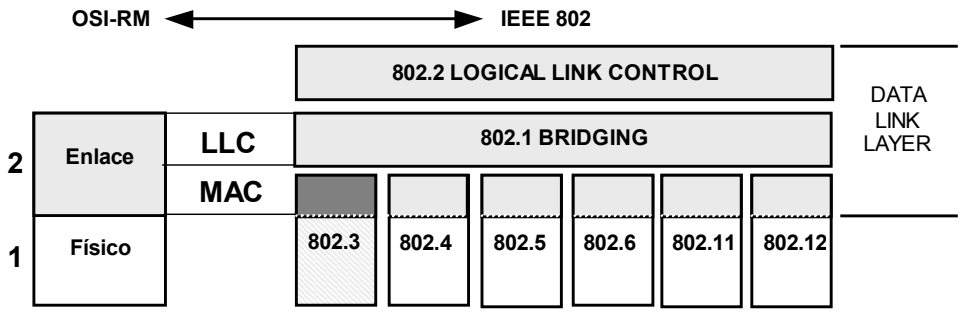


Figura 4.4 - Relação entre os padrões IEEE 802 e o OSI-RM.

Recentemente, dois novos padrões foram definidos para especificação das novas tecnologias de redes em desenvolvimento. Um é o padrão IEEE 802.11, que especifica as camadas *MAC* (Medium Access Control*)* e *PHY* (Physical Layer) para redes locais sem fios (em inglês, *wireless*). O

propósito deste padrão é prover a conexão sem fios para equipamentos ou estações móveis, dentro de uma área local, que podem ser equipamentos portáteis ou instalados em veículos. Este padrão também especifica o acesso ao meio para as bandas de freqüência utilizadas na comunicação dentro da área local. Outro é o padrão IEEE 802.12 (conhecido em inglês por *Demand Priority Access Method, Physical Layer and Repeater Specifcations*), que surgiu de uma proposta das empresas AT&T, IBM e HP denominada de *100VG-AnyLAN*. Esta proposta está baseada em uma topologia de rede em *estrela*[4] com método de acesso por contenção, onde a comunicação é realizada por meio de um dispositivo denominado *HUB* (concentrador) [COM01]. As estações só podem transmitir quando o HUB concede a permissão. Este padrão especifica redes de alta velocidade, tipicamente 100Mbps, suportando diferentes tecnologias de acesso ao meio, Ethernet e Token Ring/Bus, no mesmo ambiente de rede.

O método de acesso ao meio CSMA/CD pertence ao subnível de controle de acesso ao meio (MAC) [IEE00]. O subnível MAC e o controle de enlace lógico (Logical Link Control - LLC) juntos englobam a funcionalidade necessária para o nível de enlace definido no modelo OSI-RM. Os níveis físico e de enlace dados pelos padrões IEEE 802.2 e 802.3 correspondem aos níveis mais baixos do OSI-RM conforme ilustra a Figura 4.5.

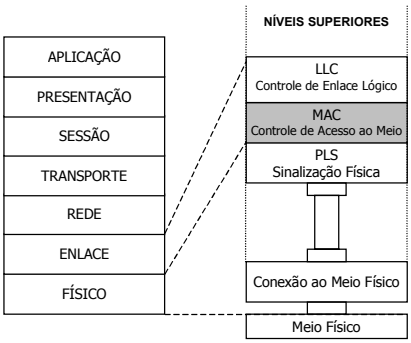


Figura 4.5. O Padrão IEEE 802.3 e sua relação com o OSI-RM.

4.4.1 Protocolo da Camada MAC

Os termos *pacote*, *quadro* e respectivas siglas serão aqui definidos, procurando manter coerência com o padrão Ethernet IEEE 802.3 pois serão utilizadas constantemente, neste Capítulo e nos que o seguem. Conforme representado na Figura 4.6, o termo *pacote* (em inglês, *packet*) refere-se a todo fluxo serial de bits que trafega no meio físico e o termo *quadro* (em inglês, *frame*), quadro de dados ou mensagem, se refere à porção do pacote que é transferida de ou para o nível superior ao subnível MAC.

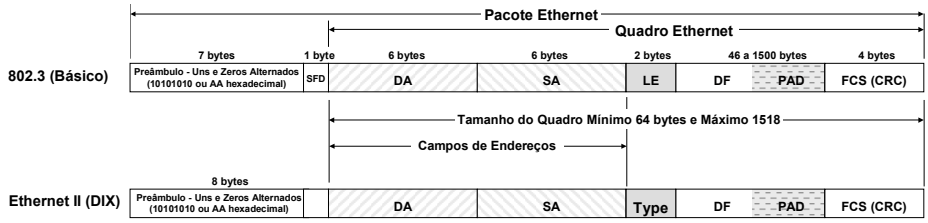


Figura 4.6. Quadros IEEE 802.3 (básico) e Ethernet II (DIX).

A Figura 4.6 detalha os dois tipos de quadro usados na transmissão de dados via tecnologia Ethernet. Há uma diferença sutil entre o padrão Ethernet II (DIX) anterior à padronização e a norma IEEE 802.3 na estrutura dos seus quadros. No quadro Ethernet II há um campo chamado *tipo*,

[4] Uma rede em estrela possui a topologia básica da ligação ponto a ponto, sendo o controle comunicação é centralizado. Cada nó é interligado a um nó central denominado *mestre*.

porém no padrão IEEE 802.3 o campo correspondente contém a informação *tamanho do quadro*. Esta diferença contudo não causa problemas para implementar o método CSMA/CD padrão IEEE 802.3, devido à codificação empregada. O primeiro byte à direita do campo é o byte mais significativo e os dois bytes são interpretados da seguinte forma:

- Se o valor é menor ou igual ao tamanho máximo dos dados, 1500 bytes, então o campo é interpretado como tamanho de quadro e representa a quantidade de bytes de dados contidos no campo seguinte (campo de dados).
- Se o valor é maior ou igual a 1536 bytes (0600 hexadecimal) então o campo indica a natureza do protocolo utilizado e o campo é interpretado como de tipo.

As interpretações de tamanho e tipo são mutuamente exclusivas, quando é utilizada a codificação de tipo, sua correta operação para o tamanho do quadro é responsabilidade da camada superior (LLC) ao subnível MAC.

A Figura 4.6 ilustra os sete campos de um quadro 802.3 (básico), comparado com um quadro Ethernet II DIX, um padrão industrial *de facto* que precedeu a norma IEEE 802.3 [SPU00]. As informações contidas em cada um dos campos do quadro padrão 802.3 são as seguintes:

- **Preâmbulo**: Indica o início da transmissão do quadro. Este campo de 7 bytes é usado para permitir ao circuito de sinalização física (em inglês, *Physical Signaling* ou PLS) sincronizar-se com o quadro recebido. Cada byte deste campo é formado pela seqüência 10101010;
- **Delimitador de início de quadro** (em inglês, *Start Frame Delimiter* ou SFD): O delimitador é a seqüência binária 10101011 e indica o começo de um quadro imediatamente após os dois últimos bits (11);
- **Endereço destino**: O endereço da estação ou estações, a quem o quadro é destinado (em inglês, *Destination Address* ou DA) e seu tamanho é de 48 bits (6 bytes);
- **Endereço fonte**: Este campo indica o endereço da estação (em inglês, *Source Address* ou SA) que iniciou a transmissão. Este campo possui tamanho de 48 bits (6 bytes);
- **Tamanho do quadro**: Este campo indica o tamanho do quadro (em inglês, *length* ou LE), exceto os campos Preâmbulo e SFD [SPU00]. O tamanho do quadro possui 16 bits (2 bytes) de comprimento. Para o padrão Ethernet II DIX, este campo é denominado *tipo* (em inglês, *type*). O campo de tipo contém um código de tipo específico de protocolo do quadro;
- **Dados**: Possui um tamanho mínimo de 46 bytes e máximo de 1.500 bytes, denominado campo de dados (em inglês, *Data Field* ou DF). Este campo contém os dados a enviar através da rede. Se os dados a enviar são menos de 46 bytes, então um conjunto de bits de enchimento (denominados em inglês de *padding bits* ou PAD) é acrescentado ao final dos dados para completar o tamanho mínimo;
- **Seqüência de verificação de quadro**: O *Frame Check Sequence* (FCS) implementa um código de verificação de erro do tipo *Cyclic Redundancy Check* (CRC). Este valor é computado como uma função dos conteúdos dos campos endereço destino (DA), endereço fonte (SA), tamanho do quadro (LE) e dados (DF+PAD, quando este último existir). O FCS possui 32 bits (4 bytes) de comprimento.

Para que pacotes sejam adequadamente transmitidos e recebidos, cada estação ou dispositivo conectado à rede deve possuir um endereço MAC que o identifique de um modo único. De modo geral, quando duas ou mais redes são interconectadas, deve-se garantir que os endereços dos dispositivos continuem identificando-os de forma única. Para garantir a individualidade dos endereços, o padrão IEEE 802.3 permite que sejam utilizadas duas formas de endereçamento [SOA95].

O antigo uso de endereços de 16 bits está especificamente excluído no padrão IEEE 802.3 de publicação mais recente [IEE00].

No campo DA, o primeiro bit (LSB) deve ser usado como designador do tipo de endereço, *individual* ou de *grupo*. Conforme representado na Figura 4.7, se o bit for "0", trata-se de um endereço individual, caso contrário é um endereço de grupo, indicando que o campo contém um endereço que identifica um, mais de um ou todas as estações conectadas a LAN. Neste caso ha dois tipos de endereços, um é o endereço de grupo com todos os bits iguais a "1", este é reservado para o grupo a que todas as estações pertencem denominado *endereço de difusão* (em inglês broadcast address). O segundo tipo é qualquer outro grupo que não o global e deve ser definido por regras em níveis mais altos do protocolo. No campo SA o primeiro bit é reservado e sempre é "0".

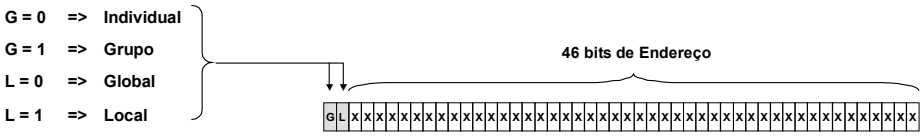


Figura 4.7 – Formato do campo de endereço DA.

O segundo bit deve ser usado para distinguir entre um endereço administrado *localmente* ou *globalmente*. Para um endereço de *administração global* este bit (L) deve ser mantido em "0", caso contrário o endereço é de *administração local*. Nessa forma, conjuntos de endereços são distribuídos aos fabricantes que se responsabilizam pela atribuição de endereços aos controladores de rede que fabricam. Essa forma de alocação de endereços garante que não haja duplicação mesmo quando redes distintas são interligadas. Os fornecedores de interfaces de rede devem decidir se usarão uma ou ambas as formas de endereçamento. Caso forneçam suporte a ambas as formas, cabe aos usuários a escolha da forma que irão utilizar em suas redes [IEE00]. Cada um dos bytes de cada campo de endereços (DA e SA) deve ser transmitido a partir do bit menos significativo (LSB).

Para o último campo do quadro, FCS, o padrão dita que se use para computar seu valor o código CRC32, que tem por base o seguinte polinômio gerador:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1.$$

Ainda segundo a norma IEEE 802.3, o valor do CRC é gerado pelo seguinte algoritmo:

a) Os primeiros 32 bits do quadro (início do campo DA) são complementados;
b) Os *n* bits do quadro são considerados como sendo os coeficientes de um polinômio M(x) de grau *n-1*. O primeiro bit do campo DA corresponde ao termo $x^{(n-1)}$ e o último bit do campo de dados (DF + PAD) corresponde ao termo $x^{(n-1)}$;
c) O polinômio M(x) é multiplicado por x^{32} e dividido por G(x), produzindo um polinômio resto R(x) de grau menor que 31;
d) Os coeficientes do polinômio R(x) são considerados como uma seqüência de 32 bits;
e) A seqüência é complementada e o resultado é o CRC;

De acordo com o exemplo da Figura 4.8, os 32 bits do CRC são colocados no final do quadro (logo após o campo de dados), no campo destinado ao FCS, de forma que o termo x^{31} seja o bit mais à esquerda do primeiro byte, e o termo x^0 seja o bit mais à direita do último byte. Os bits do CRC são então transmitidos na ordem $x^{31}, x^{30}, \ldots\ldots x^1, x^0$ e são recebidos de forma análoga.

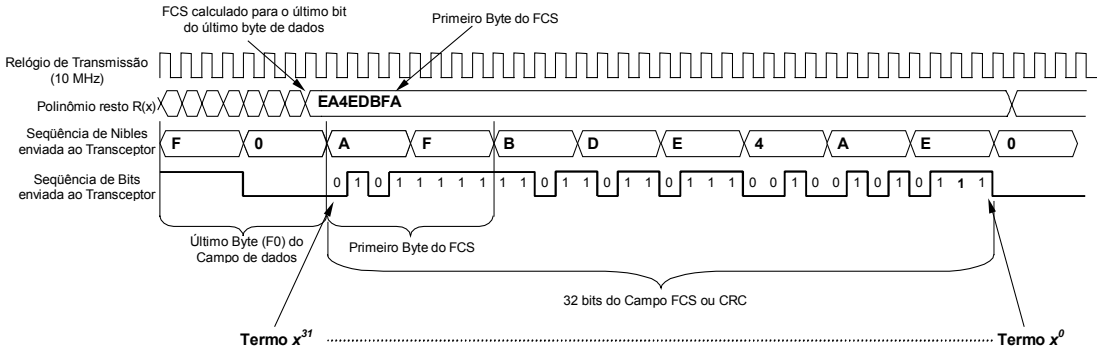


Figura 4.8 – Montagem do CRC no final do quadro.

4.4.2 Detecção de Colisões e Backoff no CSMA/CD

A rede Ethernet foi originalmente projetada com base no princípio de compartilhamento de um meio físico por todos os terminais em uma rede local. O mecanismo usado para coordenar a transmissão não é centralizado. Para tanto o protocolo exige que as estações monitorem todo o tráfego de informações no mesmo meio, antes de iniciar a transmissão de um pacote, evitando assim colisões.

Na verdade colisões são eventos esperados e absolutamente normais na Ethernet, e sua ocorrência simplesmente indica que o protocolo de CSMA/CD está funcionando como foi projetado. Quanto mais estações são agregadas a uma rede Ethernet, mais o tráfego de pacotes aumenta e mais colisões acontecerão como parte da operação normal da rede. As estações, ao detectarem uma colisão, esperam um determinado tempo para tentarem a retransmissão. Para evitar que todas as estações voltem a transmitir ao mesmo tempo, desta forma gerando novas colisões, foi definido no método CSMA/CD uma técnica conhecida como *espera aleatória exponencial truncada* (em inglês, truncated exponential back off), que doravante será referenciada simplesmente por *backoff*.

Nesta técnica de retransmissão, a estação, ao detectar uma colisão, espera um tempo aleatório que vai de zero a um limite superior, de forma a minimizar a probabilidade de colisões repetidas. Com a finalidade de manter a rede estável mesmo com tráfego intenso, o limite superior é dobrado a cada colisão sucessiva. Esta técnica também é conhecida como standard backoff [IEE00] e descrita pela equação:

$$0 \leq r \leq 2^k,$$

onde r é um valor aleatório inteiro que representa o número de intervalos de tempo (em inglês, *slot times*) que o MAC deve esperar, considerando-se que 1 slot time é igual ao tempo necessário para transmitir 512 bits. O slot time para uma taxa de transmissão de 10 Mbps é 51,2 μs. O expoente k é o menor valor entre o valor n e o valor 10, sendo $n \geq 3$ e o valor n é o número de tentativas de retransmissão.

A detecção da colisão é realizada durante a transmissão conforme representado na Figura 4.9. Ao transmitir, o módulo de transmissão fica escutando o meio e, detectada uma colisão, aborta a transmissão e espera por um determinado tempo para novamente tentar a transmissão. Devido ao tempo de propagação no meio ser finito, para que possa haver a detecção de colisão por todas as estações transmissoras, um quadro deve possuir um tamanho mínimo (64 bytes) conforme definido na Seção 4.4.1. Para quadros de grande tamanho, comparado com o tempo de ida e volta (em inglês, *Round Trip*) o método CSMA/CD apresenta considerável ineficiência na utilização da capacidade do meio [SOA95].

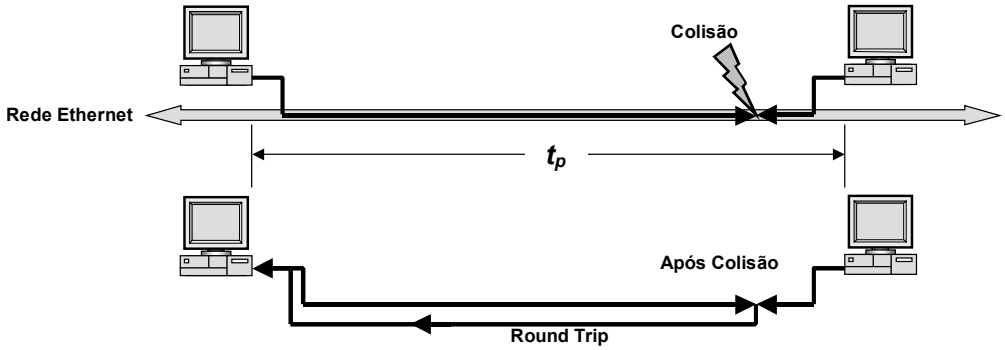


Figura 4.9 – Colisão e pós colisão na rede Ethernet.

Sendo t_p o tempo de propagação entre os dois nós mais distantes da rede, M o tamanho do quadro e C a taxa de transmissão, vale a relação:

$$M \geq 2\, C\, t_p.$$

Em todos os métodos de acesso CSMA, temos que, para um número elevado de estações, a probabilidade de ocorrência de colisões aumenta exponencialmente, de forma que o tempo de reação aumenta consideravelmente e não pode ser exatamente determinado.

Este é um fator limitante do método, cuja eficiência pode ser dada em uma primeira aproximação pela relação:

$$\text{Eficiência} = \frac{1}{1 + (3{,}4 \times t_p \times \frac{C}{M})}$$

Algumas relações podem ser tiradas desta igualdade e da desigualdade anterior:

- Quanto maior a distância, maior o tempo de propagação, menor a eficiência, e maior o tamanho mínimo do quadro para a detecção de colisão;
- Quanto maior a taxa de transmissão, maior é o tamanho mínimo do quadro e menor a eficiência;
- Quanto maior se queira a eficiência, maior deverá ser o tamanho do quadro.

4.5 Atividades do Subnível MAC

No quadro Ethernet, todos os campos são de tamanho fixo, menos o de dados que deve conter um número inteiro de bytes. O quadro Ethernet é definido como **não válido** quando uma das condições seguintes ocorrer:

- Um quadro não possuir um número inteiro de bytes no total seu comprimento;
- As partes de um quadro recebido não gerarem um valor de CRC idêntico ao valor do campo FCS recebido;
- O tamanho do quadro for menor que o tamanho de quadro mínimo.

Para que os quadros Ethernet sejam adequadamente transmitidos ou recebidos, o *subnível* MAC deve desempenhar diversas tarefas para garantir a transmissão dos dados com integridade. Listam-se abaixo estas tarefas, classificadas em atividades do subnível MAC durante a transmissão e durante a recepção de dados, respectivamente.

Na transmissão de quadros, o MAC:

- Aceita dados do *subnível* LLC e monta um quadro;
  - Acrescenta preâmbulo, delimitador de início de quadro para todos quadros de saída;
  - Computa e acrescenta o FCS para os quadros de saída e verifica o alinhamento de byte completo;
- Adia a transmissão de um fluxo de dados serial sempre que o meio físico está ocupado;
- Retarda a transmissão do fluxo de dados serial para garantir o intervalo de tempo mínimo entre quadros;
- Transmite um fluxo de dados serial para o nível físico;
- Detecta colisões e evita transmitir quando isto ocorre;
- Ao detectar uma colisão durante sua transmissão, reforça esta (ação em inglês denominada *jamming* ou simplesmente JAM) por algum tempo, para assegurar a propagação ao longo de cadeia e a detecção da colisão por todos os usuários conectados;
- Programa a retransmissão depois de uma colisão para um instante futuro calculado;

Na recepção de quadros, o MAC:

- Verifica erros de transmissão nos quadros recebidos por meio do FCS, e verifica o alinhamento de byte completo;
- Descarta quadros menores que o mínimo, os que possuem erros de CRC e os que não têm número inteiro de bytes;
- Remove e descarta preâmbulo, delimitador de início de quadro e FCS de todos quadros recebidos;
- Transfere a informação útil do quadro ao nível de protocolo superior (LLC).

Ethernet originalmente utilizava-se de sinais de radio freqüência como meio físico para implementar o protocolo CSMA/CD. Ela baseou-se no princípio da comunicação *half-duplex* onde cada estação pertencente a rede podia transmitir no meio físico para qualquer outra estação e em qualquer direção, porém apenas uma por vez. Atualmente, a tecnologia Ethernet é largamente aplicadas nas redes de computadores restritas a uma sala, um prédio ou uma corporação, onde o meio físico é composto de cabeamentos padronizados interconectados por dispositivo repetidores, concentradores (Hubs) ou chaveadores (Switches). Nas recentes redes locais a comunicação tipicamente é *full_duplex*, onde cada estação pertencente a rede pode transmitir utilizando-se de uma conexão e receber por outra. Desta forma as tarefas executadas pelo MAC, tanto na transmissão quanto na recepção, podem ser realizadas de forma concorrente e independente uma da outra.

4.6 Estudo de Caso de um Controlador de Acesso a Rede (CS8900A)

Um Controlador de Aceso a Rede (em inglês, Network Interface Controller ou NIC) é um hardware dedicado a fornecer serviços de envio e recebimento de dados através de uma rede. Dados são convertidos em cadeias de bits e fracionados em pacotes, sendo a seguir enviados ao sistema de destino. Também é responsabilidade do NIC recompor, a partir dos pacotes recebidos, os dados originais. Freqüentemente, NICs ainda implementam um mecanismo de controle de fluxo e de erros para garantir a integridade dos quadros recebidos ou enviados para ou do nível superior (LLC).

Usualmente, os controladores de rede executam funções que realizam o acesso físico ao meio de transmissão. Conforme ilustrado na Figura 4.10, um NIC se conecta ao meio de transmissão através do transceptor, cujas funções básicas são transmitir e receber sinais de e para o meio.

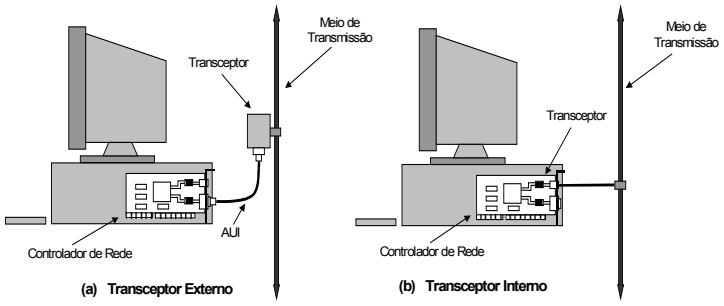


Figura 4.10 - Duas formas de conexão de NICs Ethernet ao meio físico de transmissão.

Existe hoje grande quantidade de circuitos integrados (CIs) comerciais que implementam a maior parte da funcionalidade de hardware de NICS Ethernet. A Figura 4.11 mostra um exemplo de um NIC comercial típico usado em computadores pessoais.

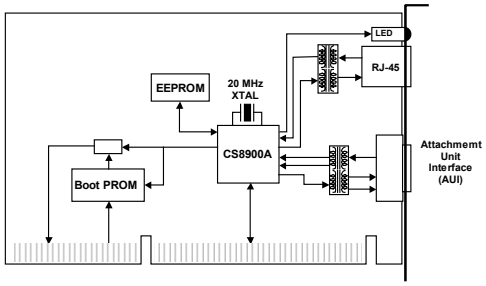


Figura 4.11 - NIC Ethernet construído em torno do CI CS8900A da Cirrus Logic, Inc.

O NIC é implementado com o CI CS8900A da empresa Cirrus Logic, Inc. Este CI dá suporte à interconexão com barramento padrão ISA. O controlador permite a conexão ao meio de transmissão diretamente (via par trançado, através do conector RJ-45) ou através do conector AUI a um transceptor externo [CIR99].

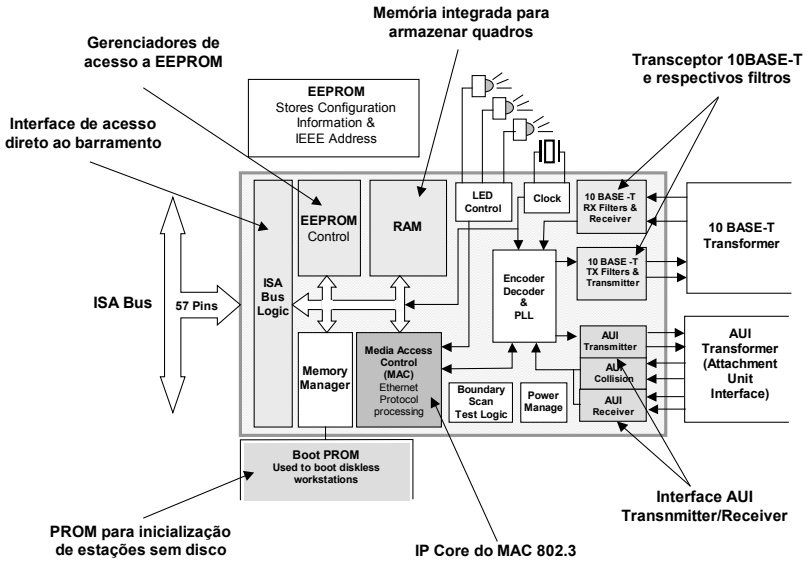


Figura 4.12 - CS8900A mostrando uma configuração básica de NIC Ethernet.

Este NIC incorpora uma memória externa de inicialização e configuração (Boot PROM) que possibilita sua utilização em computadores sem periféricos locais, tais como estações sem disco e/ou unidade de CD-ROM. A Figura 4.12 apresenta o diagrama de blocos do NIC, detalhando a estrutura interna e os principais blocos funcionais do CI CS8900A. Este CI incorpora todos os circuitos digitais e quase todos os analógicos necessários para implementar o NIC.

5. Desenvolvimento de NICs Ethernet sobre FPGAs

No Capítulo anterior foram citados diferentes padrões para redes locais e características importantes relacionadas aos sistemas de comunicações, dando-se ênfase ao método de acesso ao meio CSMA/CD de redes Ethernet. A meta deste Capítulo é apresentar a especificação de um núcleo processador de acesso ao meio em redes Ethernet que execute o método CSMA/CD, sob a forma de um IP Soft Core.

Um IP Soft Core, como foi visto na Seção 2.6, oferece liberdade quanto à tecnologia do fabricante e permite a máxima flexibilidade ao projetista para adaptar o hardware ao subconjunto de funções estritamente necessárias ao projeto em desenvolvimento. Além disto, um IP Soft Core como o desenvolvido aqui pode ser pode facilmente modificado ou resintetizado para diferentes tecnologias, podendo ser integrado a outros módulos em um mesmo CI configurável ou ASIC. Desta forma, aumenta-se o desempenho, possivelmente reduzindo custos de implementação, reduzindo potência e dimensões do produto final.

O projeto como um todo foi dividido em três módulos descritos em VHDL[MAZ92]. O primeiro módulo desenvolvido foi projetado para teste e validação funcional dos segundo e terceiro módulos, a interface MAC-Usuário e o IP Soft Core MAC Ethernet. O primeiro é uma bancada de testes, discutida em detalhes no Capítulo 6 e o segundo será detalhado no Capitulo 7.

Este Capítulo apresenta a estrutura geral e o projeto de um IP Soft Core MAC Ethernet. A partir da descrição de um ambiente típico de emprego do core explora-se o projeto dos dois módulos que compõe o IP core, o transmissor e o receptor de pacotes Ethernet. Estes módulos têm suas funções definidas a partir das especificações do protocolo padrão IEEE 802.3.

5.1 Ambiente do IP Soft Core Proposto

A Figura 5.1 mostra o ambiente típico de um sistema computacional completo onde se emprega o IP core desenvolvido. O diagrama de blocos é composto de um sistema usuário e um módulo de acesso à rede Ethernet. Este módulo de acesso comunica-se com o sistema usuário através de um barramento local. Esta via de acesso é constituída tipicamente, embora não de forma exclusiva, por um barramento padronizado tal como o ISA Bus [CIR99], o PCI Local Bus [SHA99] ou o USB [COM98] [USB99]. O módulo de acesso à rede emprega uma interface com o meio físico (PHY), que complementa as funções do módulo MAC.

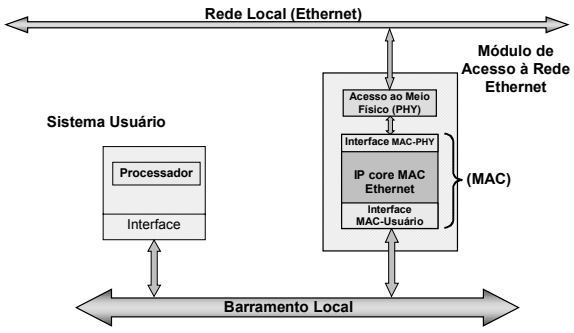


Figura 5.1 –Diagrama de blocos mostrando o ambiente típico de emprego do IP core desenvolvido.

Este ambiente mostra a necessidade de agregar duas interfaces ao IP core, visando adaptá-lo a ambientes específicos. Estas interfaces são com o barramento do sistema usuário (MAC-Usuário) e com o módulo de acesso ao meio físico (MAC-PHY). A conexão entre o IP core e estes módulos de

interface garante ao primeiro portabilidade e independência do tipo de meio físico (10/100 Mbps, serial ou paralelo, etc.), bem como do tipo de acesso ao sistema usuário (ISA, PCI, USB, etc.).

O termo *transferência* refere-se doravante ao movimento de mensagens pelo barramento local sob controle do sistema usuário. Uma *mensagem*, por sua vez, refere-se a um quadro Ethernet sem os bits de PAD e sem o FCS. Os termos *transmissão* e *recepção* referem-se à operação de envio ou captura do vetor de bits, denominado pacote, seja de forma serial ou paralela. O pacote encapsula um quadro IEEE 802.3 básico conforme definido na Seção 4.4.1. Quando este é enviado do IP core para o PHY caracteriza-se a operação de transmissão e do PHY para o IP core a operação de recepção, conforme detalhado nas Seções a seguir.

5.2 Especificação e Projeto Geral do IP Core MAC Ethernet

O IP core MAC Ethernet é composto pelos dois módulos de hardware representados na Figura 5.2. Um é o módulo de transmissão (TxMAC), cuja funcionalidade é a montagem e o controle do fluxo da transmissão do pacote Ethernet. O outro é o módulo de recepção (RxMAC), que analisa todos os pacotes que trafegam no meio físico e aceita o pacote que possui o campo DA idêntico ao seu endereço individual ou a um endereço de grupo do qual o módulo em questão participa. Isto é feito de acordo com uma configuração pré-estabelecida pelos níveis mais altos do protocolo, executados no sistema usuário. O módulo de recepção verifica três tipos fundamentais de erro de transmissão no quadro recebido. Primeiro, calcula o tamanho do campo de dados e o compara com o especificado no campo LE. Segundo, testa o alinhamento do quadro, ou seja, computa se este é composto por um número inteiro de bytes. Terceiro, recomputa o valor o CRC e o compara com o valor do campo FCS do quadro recebido.

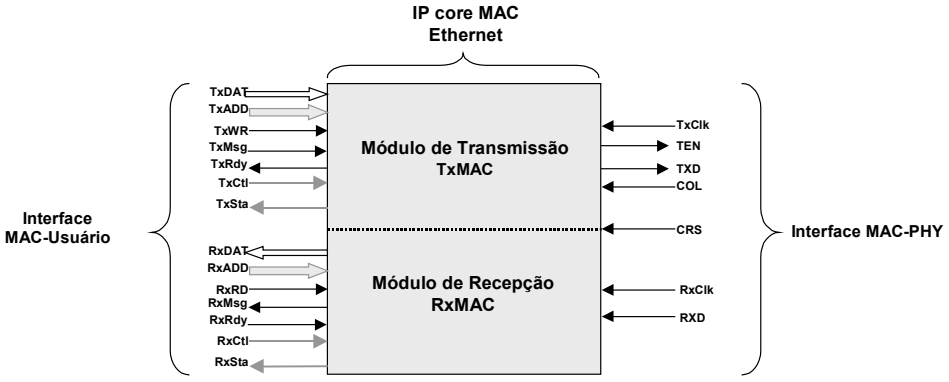


Figura 5.2 – Entradas e Saídas do IP core MAC Ethernet.

Os dois módulos trabalham de forma concorrente e independente, podendo ou não ocorrer transmissão simultânea à recepção. No caso da comunicação half-duplex[5], o sinal CRS (Carrier Sense), comum aos dois módulos, é fornecido pelo PHY, sendo sua função informar se o meio físico possui portadora. Em caso afirmativo, isto caracteriza a existência da transmissão de pacotes Ethernet no meio físico. Desta forma, o módulo de transmissão não poderá trabalhar simultaneamente ao de recepção. Caso a comunicação seja full-duplex[6], o sinal CRS é simplesmente ignorado pelo módulo transmissor.

Em cada um dos módulos TxMAC e RxMAC, são necessárias duas interfaces, uma com o lado do sistema usuário e outra com o lado PHY. A Figura 5.3 mostra um pré-detalhamento dos módulos do IP core, salientando a necessidade de buffers de memória tanto no lado TxMAC como no lado RxMAC. A função destes buffers é a de compatibilizar as taxas de transferência de mensagens do

[5] *Half-duplex*: o meio físico único entre as estações é utilizado nos dois possíveis sentidos de transmissão, porém em apenas um sentido por vez.
[6] *Full-duplex:* o meio físico entre as estações é duplicado e utilizado um para cada sentido de transmissão.

barramento local (a taxa varia de acordo com a tecnologia do barramento, ISA, PCI, USB, etc.), e de transmissão/recepção (padrão Ethernet 10/100 Mbps, etc.), utilizadas pelo IP core MAC Ethernet.

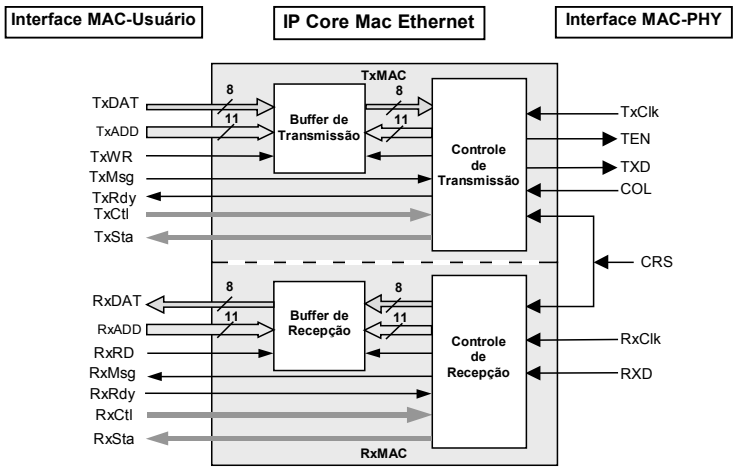


Figura 5.3 – Diagrama de blocos do IP core e interfaces.

Cada um dos buffers tem capacidade de 1536 bytes de memória RAM, podendo armazenar um quadro de tamanho máximo (1518 bytes). A transferência de mensagens de e para os buffers é realizada por meio de uma *operação de transferência*. Durante uma operação de transferência a interface com o sistema usuário, após armazenar uma mensagem no buffer de transmissão, informa ao TxMAC, que a mensagem está no buffer, pronta para ser encapsulada em um pacote e transmitida ao meio físico. A interface faz isto conforme representado na Figura 5.4, mantendo *ativado* o sinal TxMsg (nível lógico "1") até que o sinal TxRdy, fornecido pelo TxMAC seja *desativado* (nível lógico "0"), informando ao sistema usuário que o módulo TxMAC está em processo de acesso ao meio para a transmissão do pacote. O TxMAC, ao término da transmissão, ativa o sinal TxRdy informando à interface do sistema usuário que está pronto para nova transmissão, liberando o buffer para uma nova mensagem ser armazenada.

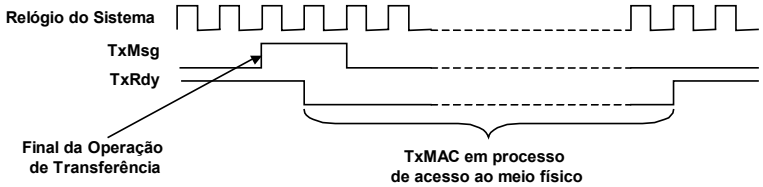


Figura 5.4 - Diagrama de tempos da operação de transferência de mensagens do sistema usuário ao TxMAC.

Esta operação é denominada *handshake* (aperto de mão) de transferência e também é utilizada pela interface de forma inversa na transferência de mensagens em sentido contrário, do buffer de recepção para o sistema usuário. De forma análoga, o RxMAC, após receber um pacote e armazenar a mensagem no buffer de recepção, mantém ativo o sinal RxMsg até que o sinal RxRdy, fornecido pela interface do sistema usuário, seja desativado. Isto irá informar ao RxMAC que o sistema usuário está realizando a transferência da mensagem armazenada no buffer de recepção.

As trocas de informação com o sistema usuário se fazem através de uma estrutura de barramentos de endereço de 11 bits (TxADD e RxADD) e dados de 8 bits (TxDAT e RxDAT), além de sinais de controle de escrita e leitura (TxWR, RxRD). Toda a troca de dados de transmissão se faz através dos buffers respectivos. Os buffers possuem dois caminhos de acesso independentes. Contudo, como os sinais de handshake garantem acesso exclusivo aos buffers, estes podem ser implementados tanto com memórias de acesso duplo como memórias de acesso simples, tornando o IP core mais flexível.

A interface com o sistema usuário apresenta mais quatro barramentos de controle. Dois destes são para controle de configuração (TxCtl e RxCtl), utilizados pelo sistema usuário na configuração dos modos de operação do IP core. Outros dois são os barramentos de status (TxSta e RxSta), cuja função é informar o resultado das operações de transmissão e recepção dos módulos do IP core. Estes barramentos são compostos por sinais individuais que serão explicitados nas Seções a seguir.

O projeto de cada um dos módulos de controle da Figura 5.3 caracteriza-se por um conjunto de blocos específicos agrupados em torno de um bloco controlador. Cada um dos blocos de controle implementa o método de acesso ao meio CSMA/CD, um para a transmissão e o outro para a recepção. Os blocos específicos são tipicamente contadores, registradores de deslocamento, comparadores, multiplexadores, registradores e outros. Nas duas Seções a seguir detalha-se a funcionalidade dos blocos de controle de transmissão e recepção, respectivamente. Nas mesmas Seções, inclui-se descrições sucintas da estrutura de cada bloco e de seus componentes internos. Para fins de facilidade de referência, empregam-se nas Seções os termos TxMAC e RxMAC para referir-se aos controles de transmissão e recepção, embora esta seja a denominação dos módulos de transmissão e recepção completos, incluindo os buffers de memória.

5.3 Especificação e Projeto do Controle de Transmissão (TxMAC)

O controle de transmissão tem como base um algoritmo que define uma seqüência de eventos dividida em duas fases, detalhadas respectivamente nos fluxogramas da Figura 5.5 e da Figura 5.6.

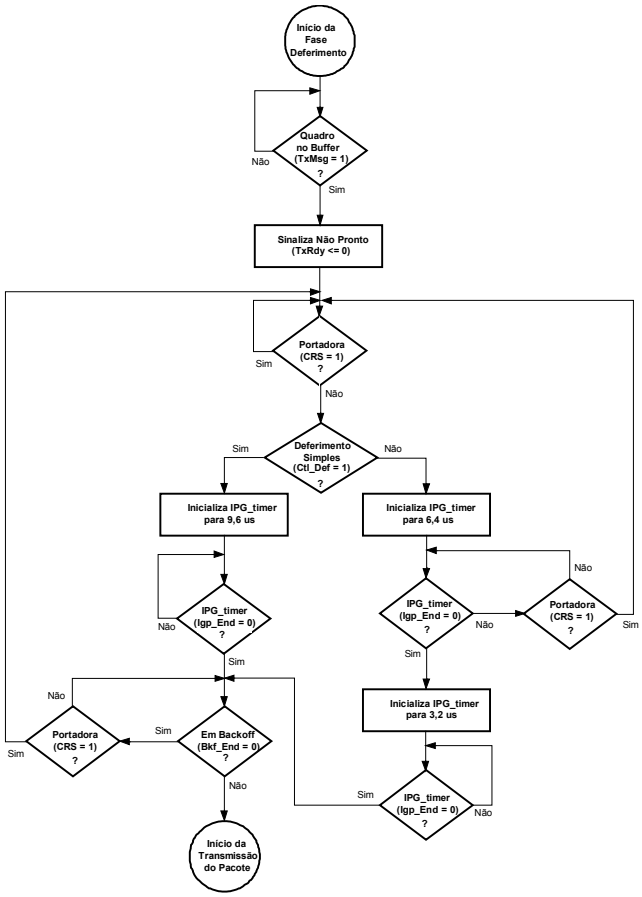


Figura 5.5 - Fluxograma da fase deferimento.

O *Deferimento* é a fase inicial, onde é adiada a transmissão sempre que o meio físico está ocupado. A transmissão do pacote através da interface com o meio físico é retardada para garantir o intervalo de tempo mínimo entre pacotes (em inglês, *Inter Packet Gap* ou IPG). A este tempo é acrescido o tempo de espera aleatória exponencial truncada (backoff), conforme especificado na Seção 4.4.2.

A fase deferimento inicia-se após uma mensagem ser inteiramente armazenada no buffer de transmissão. O sinal TxMsg, fornecido pela interface com o sistema usuário, libera o módulo de transmissão, que sinaliza o início do processo de acesso ao meio, antes de executar a transmissão do pacote. Isto é feito desativando o sinal TxRdy para então passar-se ao monitoramento do tráfego de pacotes Ethernet. Este tráfego é verificado pela presença de portadora no meio físico (sinal CRS enviado pelo PHY ao módulo de transmissão). Se o sinal CRS estiver *ativo* (nível lógico "1"), isto indica a presença de portadora. Neste caso, sabe-se que o meio físico está ocupado e aguarda-se a ausência de portadora para tentar a transmissão.

É possível configurar o controle de transmissão em dois modos, de acordo com a técnica para determinar quando poderá ser iniciada a transmissão do pacote Ethernet. Um sinal Ctl_Def, parte do barramento de controle TxCtl e proveniente do sistema usuário é utilizado para selecionar o deferimento *simples* (em inglês, simple deferral) ou deferimento *em duas partes* (em inglês, two-part-deferral). Quando o sinal Ctl_Def estiver ativo (nível lógico "1"), o controle assume deferimento simples, inicializando um temporizador (IPG_timer) para 9,6 μs, esperando pelo término deste intervalo de tempo. Após o IPG_timer sinalizar o término do tempo mínimo entre quadros colocando o sinal Igp_End em nível lógico "1", o TxMAC verifica se o controle encontra-se em estado de backoff (sinal Bkf_End do barramento de controle TxSta em nível lógico "0"). Caso contrário, inicia a transmissão do pacote sem verificar a presença de portadora no meio físico.

O sinal Bkf_End, proveniente de um gerador de temporizações aleatórias para backoff (Backoff_timer), será desativado caso tenha ocorrido pelo menos uma tentativa de transmissão deste pacote Ethernet, indicando que está decorrendo o tempo de backoff gerado. O TxMAC permanece em laço enquanto espera o fim do tempo de backoff, verificando a presença de portadora. Estando em backoff e se verificada a presença de portadora, retorna-se ao início da fase deferimento, para uma nova tentativa de transmissão do atual pacote armazenado no buffer. Contudo, se o término do tempo de backoff (sinal Bkf_End em nível lógico "1") ocorrer sem a presença de portadora, então passa-se ao início da fase de transmissão.

Na outra configuração, deferimento em duas partes, o TxMAC inicializa o IPG_timer para 6,4 μs e espera pelo término deste tempo em laço, verificando a presença de portadora. Caso seja verificada a presença de portadora, retorna-se ao início da fase deferimento. Caso contrário, inicializa-se novamente o IPG_timer para 3,2 μs e espera pelo término deste tempo em laço. Desta vez porém, sem verificar a presença de portadora. Neste ponto, quando o IPG_timer sinalizar o término do tempo, a rotina segue os mesmos passos finais da rotina executada na configuração deferimento simples.

A *Transmissão* é a segunda fase do controle de transmissão, onde realiza-se a montagem e o controle da transmissão do pacote Ethernet, através do módulo PHY e para meio físico. O funcionamento desta fase é detalhado no fluxograma da Figura 5.6. A interface com o meio físico foi projetada sendo como alvo primário, embora não exclusivo as especificações para o padrão 10BASE-T/100BASE-TX Ethernet. Conforme ilustrado na Figura 5.2, o PHY irá fornecer os sinais de RxClk (relógio de recepção) e o TxClk (relógio de transmissão) utilizados na sincronização do fluxo serial de bits de e para o IP core. O fluxo serial de bits é transmitido do IP core ao meio físico pelo sinal TxD e pelo sinal RxD no sentido inverso, do meio físico ao IP core. O sinal TEN (Transmit Enable) será usado para o PHY iniciar a transmissão do fluxo serial de bits gerado no IP core. O sinal de portadora CRS (Carrier Sense) é distribuído tanto ao módulo de transmissão quanto ao de recepção, informando a presença de tráfego presente no meio físico. O sinal COL (Collision Detect) indica ao módulo de transmissão quando ocorre uma a colisão.

Na fase de transmissão, imediatamente após o preâmbulo e o SFD seguem os bits da mensagem armazenada no buffer de transmissão. Estes são transmitidos através da linha TxD ao PHY. Após cada bit ser transmitido, verifica-se se houve ou não uma colisão a partir do valor do sinal COL do PHY. No final do último bit de dados ou PAD são acrescentados os 32 bits do FCS, resultantes da operação do gerador de CRC cuja função foi descrita na Seção 4.4.1. A seqüência de operações e as ações resultantes da fase de transmissão são sincronizadas pelo sinal TxClk.

Conforme representado pelo fluxograma da Figura 5.6, a fase de transmissão inicia-se com a geração do preâmbulo, seguido do delimitador de início de quadro, SFD. Ao final da transmissão do último bit do SFD, um contador de bits é inicializado (Ctr_Bit) e o gerador de CRC é habilitado a operar.

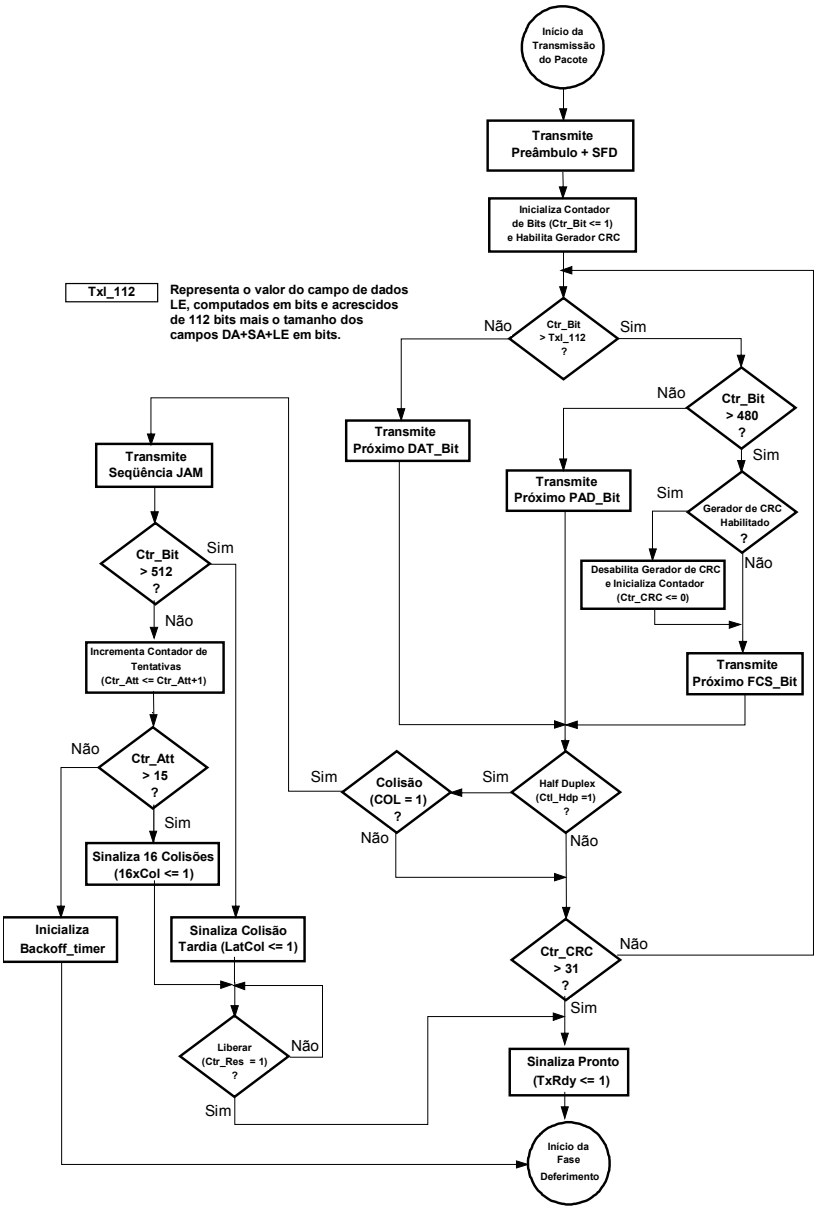


Figura 5.6 – Fluxograma da fase de transmissão.

O contador de bits serve para delimitar o campo de dados em relação ao PAD[7] (definido na Seção 4.4.1), Um sinal auxiliar, denominado Txl_112, é usado no processo de delimitação de dados e

[7] O campo de dados deve ser estendido com a incorporação de bits extras (pad bits = "0") até atingir o comprimento mínimo de 368 bits, caso a mensagem a enviar possua menos de 46 bytes (46*8=368).

PAD. Este sinal resulta da comparação do valor do contador de bits transmitidos com o resultado da soma da constante 112 (a soma do tamanho, em bits, dos campos DA, SA e LE) com o valor, convertido em bits, do campo LE. O sinal Txl_112 é ativado (nível lógico "1") caso o contador de bits possua neste momento uma contagem menor ou igual a 480, informando quando é transmitido o último bit do campo de dados (DAT_Bit). Inicia-se então a transmissão dos bits de enchimento (PAD_Bit) para completar o tamanho mínimo do quadro.

No caso da contagem de bits ser maior que 480 após a transmissão do último bit do campo de dados ou PAD, o gerador de CRC é desabilitado e inicializa-se o contador Ctr_CRC que informa se foi completada a transmissão dos 32 bits do FCS. O valor correspondente ao FCS calculado fica registrado no gerador, e neste momento os bits registrados (FCS_Bit) são transmitidos conforme definido na Seção 4.4.1 e ilustrado na Figura 4.8. Isto é feito imediatamente após o último DAT_Bit ou PAD_Bit, conforme representado no fluxograma da Figura 5.6.

Após a transmissão de cada um dos bits pertencentes aos dados, PAD ou FCS, é verificado se houve colisão. A verificação de colisão somente será realizada caso o sinal de controle Ctl_Hdp (barramento de controle TxCtl), proveniente do dispositivo usuário, tenha sido ativado para configurar comunicação Half-Duplex no módulo de transmissão. Não sendo verificada colisão, é verificado se o Ctr_CRC atingiu a contagem de 32 bits transmitidos. Isto indica que o último bit do FCS foi transmitido e o módulo sinaliza que está pronto, ativando o sinal TxRdy, liberando o buffer de transmissão para o sistema usuário poder armazenar nova mensagem. Desta forma, é encerrada a fase de transmissão com sucesso e volta-se ao início da fase deferimento, aguardando que o sistema usuário sinalize que uma nova mensagem foi armazenada no buffer de transmissão.

Até este ponto observa-se que os bits são transmitidos sem a ocorrência de colisão. Caso haja uma colisão após a transmissão de qualquer um dos bits, o fluxo serial é modificado para uma seqüência padrão de 32 bits iguais a zero, configurando a operação JAM (Seção 4.4.1). Isto está ilustrado na Figura 5.6. Desta forma, é reforçada a colisão por um período de tempo igual a 3.2 μs (no caso de um IP core operando a 10 Mbps), para então ser interrompida a transmissão do pacote Ethernet.

Logo que a transmissão é interrompida verifica-se se a colisão ocorreu após o envio dos primeiros 512 bits (Ctr_Bit > 512). Esta situação configura uma *colisão tardia* e é considerado um erro de transmissão grave. O bloco de controle registra o erro, ativando o sinal LatCol (no barramento de status, TxSta), informando ao sistema usuário a ocorrência de uma colisão tardia, Aguarda-se então o sinal Ctl_Res (ativado pelo sistema usuário via o barramento TxCtl) para após reiniciar o processamento na fase deferimento.

Contudo, se é verificada a ocorrência de uma *colisão normal*, isto é, quando a colisão ocorre dentro dos primeiros 512 bits, o contador de tentativas é incrementado (Ctr_Try). Na seqüência, verifica-se se o contador excedeu 15 tentativas. Neste caso, está configurado um erro de *colisões excessivas*, o qual também é considerado um erro de transmissão grave. O módulo registra o erro, ativando o sinal *16xCol* (no barramento de status, TxSta) e segue o mesmo fluxo que sucede o registro de uma colisão tardia. Caso o número de tentativas não exceda a 15, é inicializado o Backoff_timer e a rotina volta ao início da fase deferimento sem ativar o sinal TxRdy, mantendo o sistema usuário impossibilitado de transferir uma nova mensagem ao buffer de transmissão. O sinal TxRdy irá permanecer desativado até o instante em que uma fase de transmissão seja encerrada com sucesso ou sejam excedidas 15 tentativas de transmissão consecutivas ocorrendo colisões normais.

A Figura 5.7 mostra o diagrama de blocos do TxMAC que implementa os algoritmos descritos pelos fluxogramas acima. Nesta Figura, o controlador é o principal responsável pela seqüência de operações executadas pelos blocos restantes. Trata-se obviamente de uma máquina de estados finita.

Como já foi salientado antes, a implementação como de todos os módulos do IP core Ethernet foi realizada em VHDL [MAZ92], sendo a divisão em blocos um reflexo razoavelmente fiel da organização em pares de estruturas entidade/arquitetura da linguagem. Existe dois modos possíveis de transmissão no IP core implementado. A primeira é a transmissão serial pura, usando o sinal Txds. Uma segunda possibilidade de transmissão consiste em usar transmissão de 4 em 4 bits. Isto se deve ao fato de muitos módulos de acesso ao meio físico aceitarem dados neste formato, realizando eles mesmo a serialização. Na Figura 5.7, isto corresponde a empregar o sinal Txd ao invés do sinal Txds.

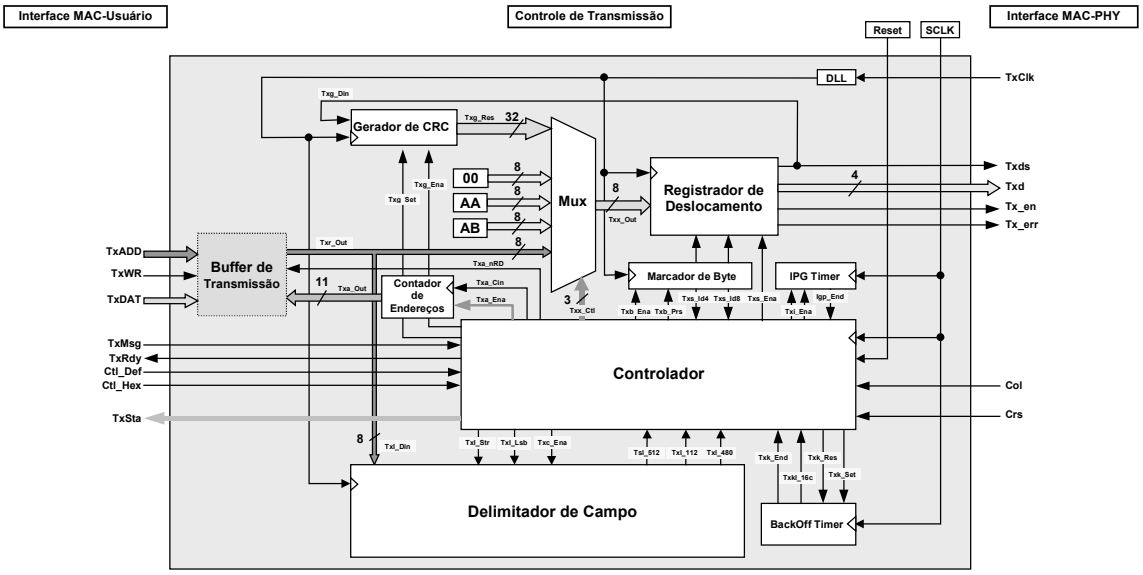


Figura 5.7 – Diagrama de blocos do controle de transmissão do IP Soft Core MAC Ethernet.

A Figura 5.8 mostra o primeiro de uma série de diagramas de tempos parciais ilustrando a funcionalidade da implementação do processo de transmissão do IP core. Para tanto, assume-se que o IP core está configurado para operar com deferimento simples. Quando é armazenada uma nova mensagem no buffer de transmissão (ver indicação de handshake na Figura, via sinais TxMsg e TxRdy), o controlador entra na fase deferimento, inicializando e habilitando o bloco IGP Timer. O controlador aguarda, monitorando o sinal de portadora e a ativação do sinal Ipg_End (sinal interno oriundo do IPG Timer). O IPG Timer utiliza o relógio do sistema (SCLK) como base para contar os tempos (9.6 µs, 6.4 µs ou 3.2 µs) determinados pelo controlador.

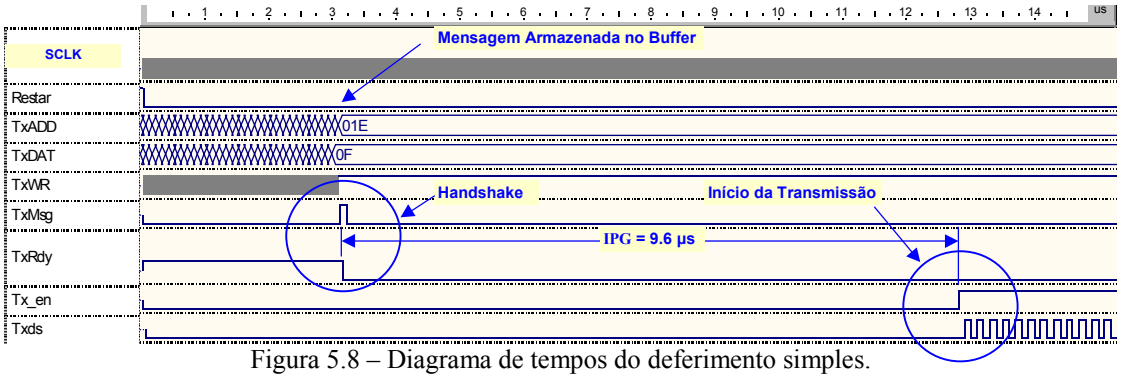


Figura 5.8 – Diagrama de tempos do deferimento simples.

Na Figura 5.8, pode-se ainda observar que o início da transmissão ocorre após o IPG, com a ativação do sinal Tx_en, que habilita o PHY para transmitir o fluxo serial de bits presente no sinal Txds. A segunda parte do processo de transmissão normal está ilustrada na Figura 5.9. O controlador, antecipado em oito pulsos do relógio de transmissão (TxClk) habilita (sinal Txs_ena interno ao IP core) o bloco registrador de deslocamento para conversão dos dados paralelos (entrada de 8 bits, Txs_din) em um fluxo serial Txds, conforme ilustrado na Figura 5.9.

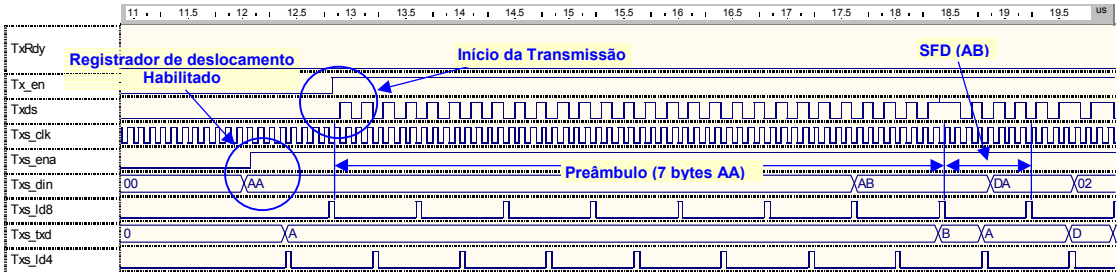


Figura 5.9 – Diagrama de tempos do início da transmissão, preâmbulo e SFD.

Os bytes são carregados no registrador de deslocamento por um pulso (Txs_ld8) proveniente do bloco marcador de byte. O marcador é um contador de 3 bits sincronizado com o relógio de transmissão, habilitado pelo controlador juntamente com o registrador de deslocamento. Este marcador fornece também um sinal (Txs_ld4) que informa exatamente o meio do byte (em inglês, *nibble*) a ser serializado. Este pulso é utilizado pelo registrador para converter, os bytes de sua entrada em pares de nibbles (primeiro o menos significativo) transferidos pelo sinal de 4 bits (Txd) para módulos PHY que trabalhem neste modo.

Os dados carregados no registrador de deslocamento são selecionados pelo controlador através do bloco MUX (Figura 5.7). Os bytes da mensagem armazenada no buffer, o preâmbulo, o SFD, o PAD, bem como os quatro bytes do FCS são todos transmitidos via MUX para o registrador de deslocamento, cada um no exato instante necessário para a montagem do pacote.

A estrutura interna do módulo delimitador de campo é detalhada na Figura 5.10. O bloco controlador utiliza-se do módulo delimitador de campo para selecionar o MUX, de forma que seja possível montar pacotes de tamanho mínimo para mensagens com campo de dados contendo menos de 46 bytes. Isto é feito selecionando-se um byte com valor 0 para enchimento do campo de dados, no momento que o último byte de dados armazenado no buffer for transmitido. Paralelamente à transmissão do preâmbulo, o controlador endereça o buffer de transmissão e carrega os dois bytes correspondentes ao tamanho do campo de dados no registrador LenBit do bloco delimitador de campo, conforme representado no diagrama de tempos da Figura 5.11.

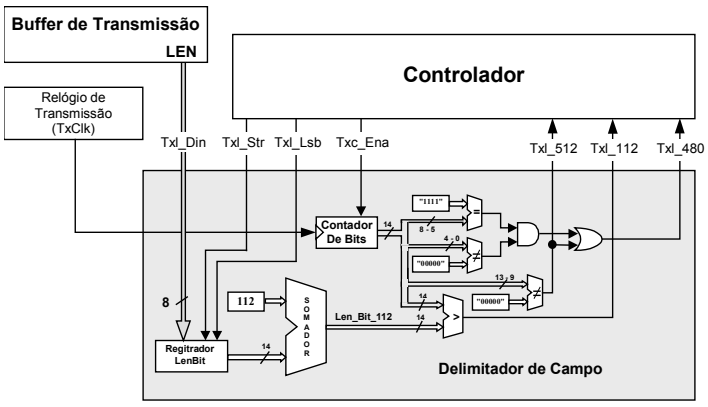


Figura 5.10 - Diagrama de blocos do delimitador de campo do módulo de transmissão.

O delimitador de campo, via o registrador LenBit converte os dois bytes do campo LEN da mensagem a transmitir para um valor em bits e a este soma a constante 112. O valor da soma é comparado com o valor do contador de bits. Caso este seja maior que a soma é ativado o sinal Txl_112, que informa ao controlador que os próximos bits serão de enchimento, conforme ilustrado no diagrama de tempos da Figura 5.11.

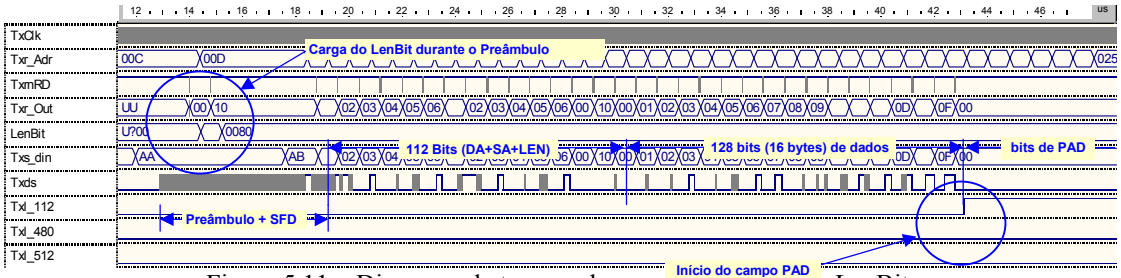


Figura 5.11 – Diagrama de tempos da carga do registrador LenBit.

De acordo com a Figura 5.10, o bloco delimitador de campo também fornece os sinais Txl_480 e Txl_512, que informam ao controlador o final do campo de PAD e o final de quadro mínimo, respectivamente. O último destes sinais também é utilizado para detectar a posição de uma colisão, conforme anteriormente discutido.

Finalmente, o diagrama de tempos da Figura 5.12 mostra a simulação funcional do final de uma transmissão normal de pacote, para o mesmo exemplo. No final da transmissão dos bits de enchimento, o controlador desabilita o gerador de CRC e o valor correspondente ao campo FCS calculado fica registrado no gerador. O controlador seleciona então o MUX para transmitir os quatro bytes que compõe o FCS calculado, conforme ilustrado no diagrama de tempos. Concluindo, o controlador encerra uma transmissão sem colisões, desabilitando o sinal Tx_en e habilitando o sinal TxRdy.

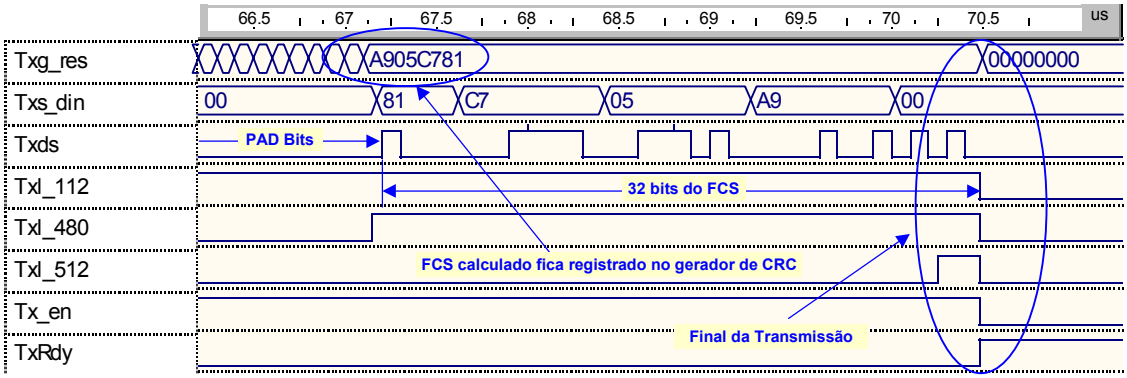


Figura 5.12 – Diagrama de tempos do final da transmissão: final do campo PAD, transmissão do FCS e liberação do buffer de transmissão.

5.4 Especificação e Projeto do Controle de Recepção (RxMAC)

O controle de recepção é responsável pela *operação de recepção*, onde somente a mensagem é transferida do IP Core para o sistema usuário. O preâmbulo, o SFD e o campo de PAD, quando este existir, são removidos pelo controle de recepção. O FCS pode ou não ser transferido, dependendo da configuração. Na recepção, o RxMAC apresenta um comportamento reverso ao do controle de transmissão. De forma semelhante ao TxMAC, este também monitora a presença de portadora no meio físico, verificando o mesmo sinal CRS fornecido pelo PHY. Desta forma, o receptor observa todo o tráfego de pacotes do meio físico, inclusive os transmitidos pelo módulo TxMAC adjacente. O RxMAC seleciona alguns dentre os pacotes que transitam pelo meio físico através de um filtro de endereços de destino, denominado *filtro DA*. O quadro correspondente é desencapsulado do pacote selecionado e armazenado no buffer de recepção, para após ser transferido do buffer ao sistema usuário, caso o quadro tenha sido identificado como um quadro Ethernet válido. O filtro DA identifica o endereço contido no campo DA do quadro durante processo de recepção. O filtro pode ser configurado via sinais do barramento de controle RxCtl. O filtro compara os bytes recebidos do campo DA com os bytes contidos no registrador Fil_DA, para identificar e aceitar qualquer quadro

cujo endereço destino seja um endereço individual ou de grupo correspondente ao conteúdo do registrador Fil_DA, o qual foi previamente carregado com uma codificação como definida na Figura 4.7, selecionando uma das formas de endereçamento válidas.

Durante a operação de recepção podem ocorrer erros tais como colisões ou interferências externas no meio físico. Estes erros podem corromper o conteúdo dos pacotes. Três tipos de erros podem ser detectados durante a recepção de um pacote:

- *Erro no comprimento do quadro*, é detectado quando o pacote recebido contém um quadro cujo total de bytes recebidos não é consistente com valor informado no campo LE. Este pode ocorrer sob duas formas específicas: erro de *comprimento mínimo* (Err_Minl), quando o quadro é fragmentado antes de atingir o tamanho mínimo de quadro (64 bytes), indicando uma possível colisão, e o erro *comprimento máximo* (Err_Maxl), quando o quadro ultrapassa o tamanho de 1518 bytes;
- *Erro de alinhamento do quadro,* é detectado quando um quadro recebido possui um comprimento válido, porém o FCS, comparado com o valor do gerador CRC é inválido e além disto, após o FCS possui bits excedentes (*Extra Bits*);
- *Erro de FCS* é detectado pelo receptor quando, apesar deste desencapsular o quadro sem a ocorrência de nenhum dos erros anteriores, o valor computado no gerador de CRC para o pacote corrente não é igual ao campo FCS do quadro recebido.

O controle de recepção implementa o algoritmo ilustrado no fluxograma da Figura 5.13 de forma bastante simplificada. O fluxograma do RxMAC apresenta uma seqüência de eventos que ocorrem sincronizados com o relógio de recepção RxClk gerado pelo PHY. este relógio varia de acordo com a taxa de recepção (10/100 Mbps) do pacote Ethernet que trafega no meio físico, como é descrito seguir.

No início da recepção do pacote, o RxMAC verifica se o buffer de recepção está livre através do sinal RxRdy, fornecido pela interface com o sistema usuário. Este não estando ativo, o RxMAC aguarda a sua ativação para então verificar a presença de portadora via o sinal CRS. Caso RxRdy já esteja ativo o RxMAC espera até ocorrer ausência da portadora. Esta situação caracteriza o início do intervalo de tempo existente entre os pacotes que trafegam no meio físico (IPG). Desta forma, o receptor pode sincronizar-se ao início do próximo pacote a ser recebido.

Quando o sinal CRS é novamente ativado, o RxMAC sinaliza *não pronto*, desativando o sinal RxMsg. Isto informa para a interface com o sistema usuário que está iniciando a operação de recepção e que de agora em diante o buffer está reservado para armazenar os bytes do quadro a ser recebido. A seguir, o RxMAC entra na fase em que os bytes do preâmbulo são verificados e removidos até que seja encontrado o byte SFD. Se algum erro ocorrer neste ponto, ocasionando a perda do SFD (o padrão AB em hexadecimal), algum mecanismo deve ser implementado para detectar o problema, evitando um laço até que um byte AB surja, possivelmente em local incorreto neste ou em outro pacote subseqüente. Isto se faz pela contagem de bytes do preâmbulo, até um máximo de 7, após o que o SFD *deve* ser o próximo. Salienta-se aqui que o tamanho do preâmbulo, segundo a norma IEEE 802.3, pode ser variável, sendo 7 bytes um tamanho típico e máximo. Este processamento não foi incluído no fluxograma para mantê-lo simples.

Embora não apareça no fluxograma por questões de simplicidade, existem vários pontos em que está implícita a recepção de um byte do pacote proveniente do meio físico. Na Figura 5.13, isto ocorre antes de cada teste pelo término do campo DA (Ctr_Byt=6), antes de cada teste pelo término do campo SA (Ctr_Byt=12) e antes do primeiro teste pela presença de portadora durante a recepção do campo de dados DF (CRS=1, losango ao centro da Figura). Em verdade, o uso de um

fluxograma é apenas ilustrativo, uma vez que muitas das tarefas aqui descritas estão sendo realizadas em paralelo.

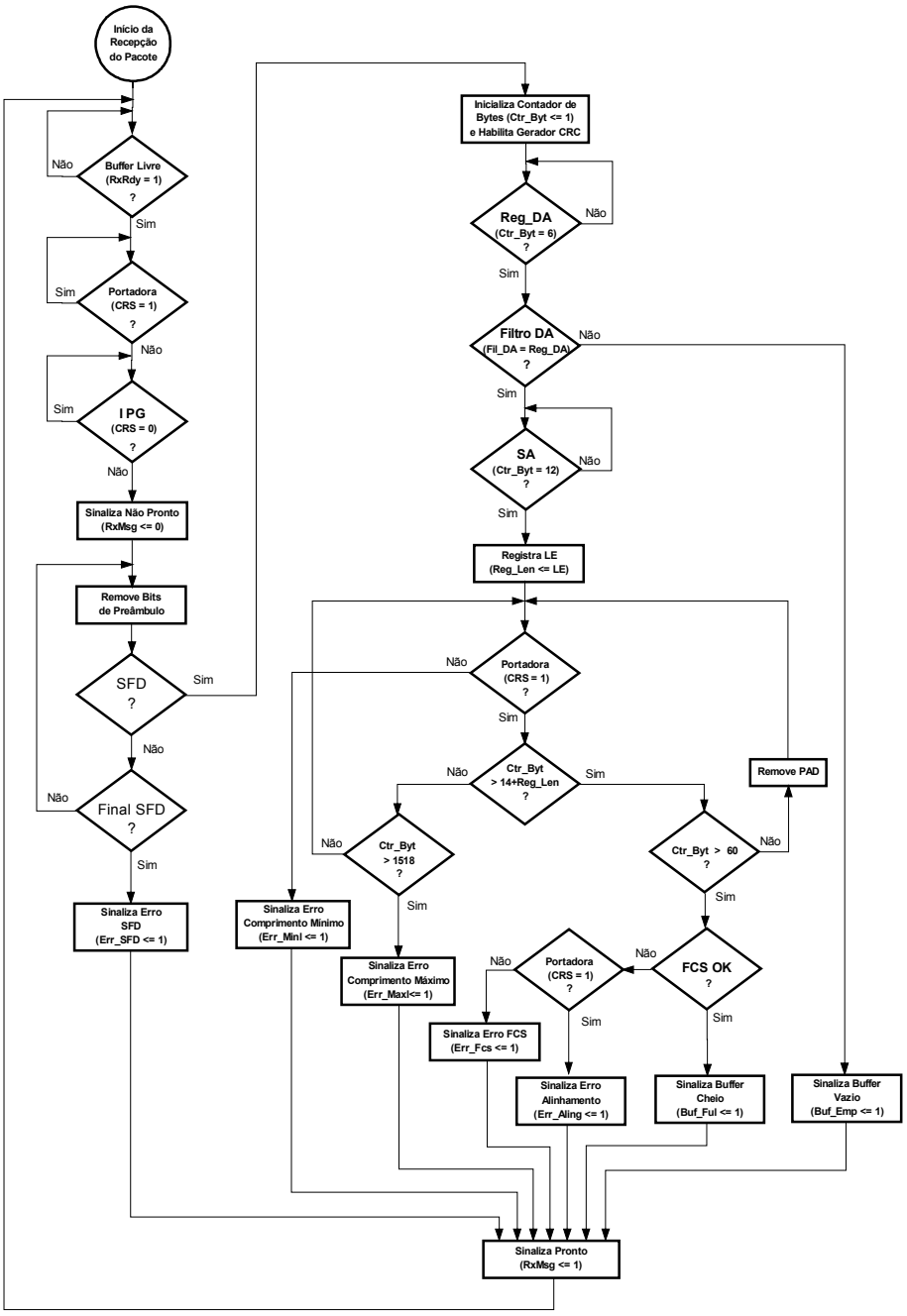


Figura 5.13 – Diagrama de fluxo do módulo receptor.

Após o último byte do campo DA ser recebido o Reg_DA é comparado com o registrador Fil_DA para identificar o endereço de destino do quadro corrente. Caso a comparação resulte falsa, o pacote é descartado e o RxMAC sinaliza que está *pronto*, ativando o sinal RxMsg, e voltando ao início da rotina de recepção para aguardar o próximo pacote.

Contudo, se o endereço de destino for validado pelo Filtro DA o RxMAC segue recebendo bytes e verificando a contagem até atingir 12 bytes recebidos. Destes, os últimos seis são referentes ao campo SA e os próximos dois bytes são o campo LE da mensagem armazenada no registrador

Reg_Len. A partir desta situação o RxMAC irá verificar constantemente a existência de portadora, após cada byte recebido.

Nesta fase da operação de recepção são detectados os possíveis erros de transmissão. O RxMAC, após verificar a existência da portadora compara o contador de bytes (Ctr_Byt) com o total de 14 bytes (total dos campos DA+SA+LE do quadro corrente) somados ao valor do Reg_Len, previamente armazenado. Caso o Ctr_Byt seja menor ou igual ao valor 14+Reg_Len o RxMAC verifica se o valor do Ctr_Byt não atingiu ao tamanho máximo de quadro. Não tendo sido atingida a contagem de 1518 bytes, a rotina volta a verificar a existência de portadora. Contudo, se a contagem de bytes ultrapassou este valor, então o RxMAc sinaliza um erro de comprimento de quadro, ativando o sinal Err_Maxl (do barramento de status RxSta). Isto informa ao sistema usuário a ocorrência do erro de comprimento máximo. Após, o RxMAC sinaliza que está pronto, ativando o sinal RxMsg e retorna ao início da operação de recepção.

Por outro lado, se após a verificação da existência de portadora o contador de bytes for verificado maior que 14+Reg_Len, o RxMAC segue para verificar se o Ctr_Byt é maior que 60. Caso isto não se verifique, trata-se o quadro como um de tamanho mínimo (64 bytes) e cujo campo de dados possui bits de enchimento (PAD bits). Faz-se isto removendo o campo de PAD e voltando a verificar a existência de portadora até a contagem ultrapassar 60 bytes, para então proceder-se à verificação do FCS recebido, que é comparado com o valor computado no gerador de CRC. Sendo válido o FCS, o RxMAC considera que os bytes armazenado no buffer correspondem a um quadro válido, sinaliza que está pronto, ativando o sinal RxMsg e retorna ao início da rotina de recepção. Desta forma é realizado o handshake com o sistema usuário que transfere a mensagem armazenada e libera o buffer, ativando o sinal RxRdy, para o RxMAC iniciar uma nova operação de recepção.

A ocorrência de um FCS inválido implica que o quadro foi corrompido por colisão ou interferência externa, mesmo que o RxMAC tenha identificado o valor do comprimento do campo de dados (LE) como válido. Para identificar o tipo de erro ocorrido, o RxMAC após verificar um FCS inválido, verifica mais uma vez a existência de portadora. Caso o teste resulte positivo, o RxMAC sinaliza erro de alinhamento, ativando o sinal Err_Align. Caso contrário, sinaliza-se erro de FCS, ativando o sinal Err_Fcs. Nos dois casos, o RxMAC sinaliza pronto, retornando ao início da recepção de pacotes.

A sinalização de erro de comprimento mínimo ocorre quando durante a operação de recepção é verificada a falta de portadora, sendo a contagem de bytes interrompida antes de atingir o valor correspondente ao tamanho mínimo de quadro (64 bytes).

A Figura 5.14 mostra o diagrama de blocos funcionais do controle de recepção RxMAC. Ela apresenta o conjunto de blocos funcionais que implementam operações, sob o comando do bloco principal, o controlador. De forma semelhante ao módulo transmissor, o controlador é responsável pela seqüência de operações executadas pelo RxMAC. O controlador também é uma máquina de estados finita e sua seqüência de estados segue as etapas descritas no fluxograma do RxMAC da Figura 5.13.

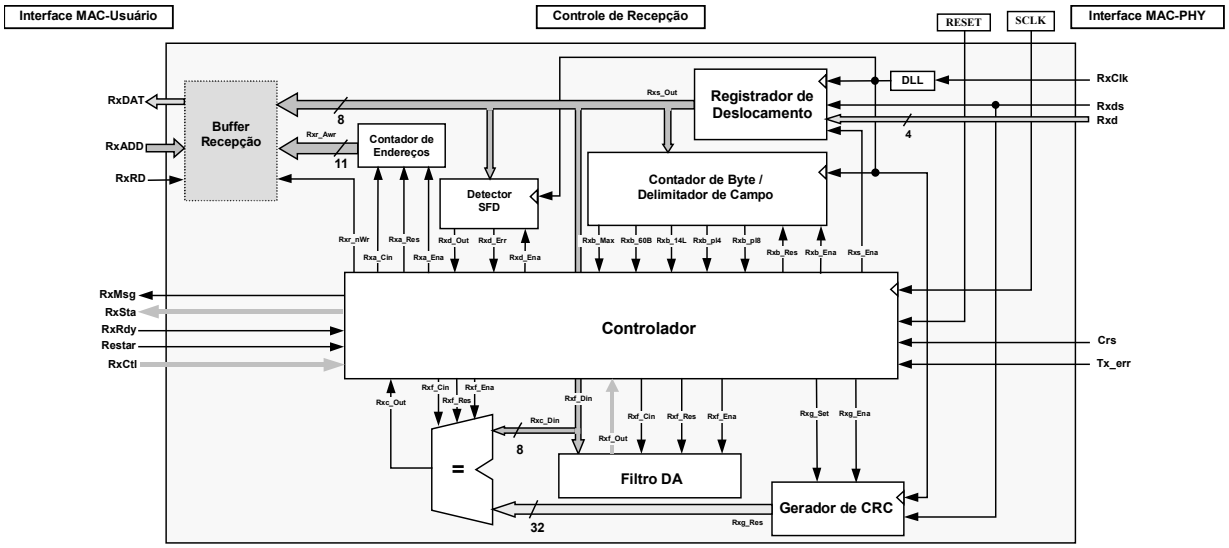


Figura 5.14 - Diagrama de blocos do módulo de recepção do IP Core MAC Ethernet.

O RxMAC recebe pacotes fornecidos pela interface MAC-PHY através do sinal Rxds na forma serial pura. Contudo, com pequenas modificações no registrador de deslocamento, é possível implementar também a opção de recepção de 4 em 4 bits, a exemplo do que ocorre no TxMAC. A implementação serial pura é utilizada para ilustrar a funcionalidade do receptor na situação mais desfavorável, pois neste caso a velocidade do registrador de deslocamento deve ser quatro vezes maior que a de 4 em 4 bits.

A título de exemplo, assume-se que o pacote recebido pelo RxMac é exatamente o mesmo utilizado como exemplo na descrição do TxMAC, de tamanho mínimo. O campo DA possui a seqüência hexadecimal "DA0203040506" e o SA a seqüência "FA0203040506". O tamanho do campo de dados contém a seqüência "0010" informando um comprimento de 16 bytes para este campo, cujos dados são representados pela seqüência hexadecimal "000102030405060708090A0B0C0D0E0F".

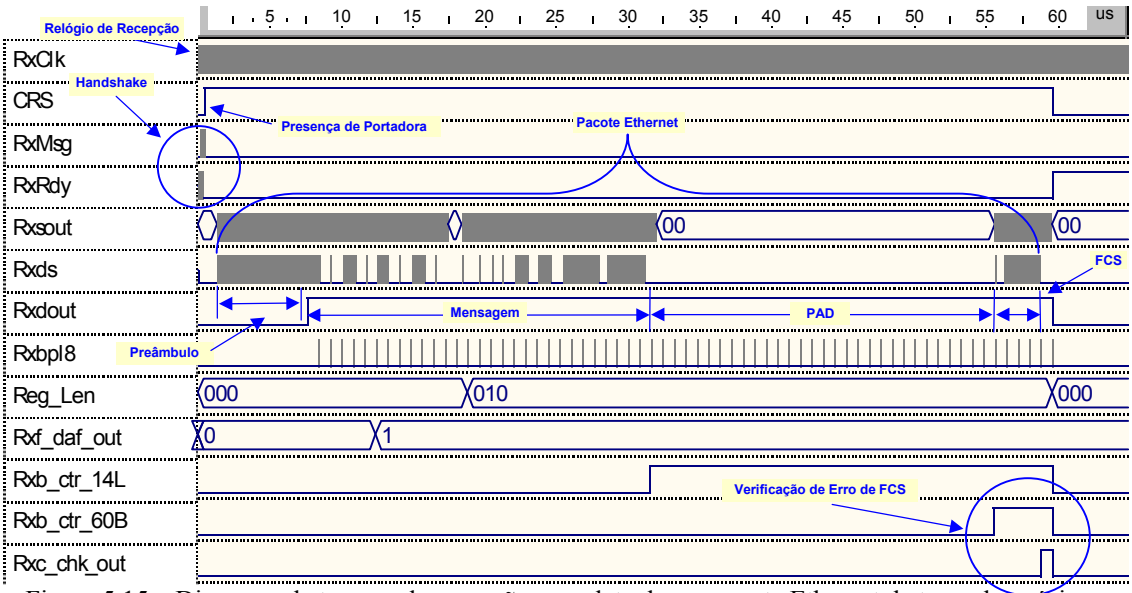


Figura 5.15 – Diagrama de tempos da recepção completa de um pacote Ethernet de tamanho mínimo.

A Figura 5.15 ilustra a recepção completa do pacote Ethernet. Neste diagrama de tempos pode-se observar que aproximadamente a metade do pacote (30 bytes de um total de 64 bytes) possui bits de enchimento (PAD), os quais serão removidos juntamente com o preâmbulo, SFD e FCS, para que somente a mensagem seja armazenada no buffer de recepção. Os sinais de controle e as diferentes

fases do processo executado pelo RxMAC representadas neste diagrama de tempos, serão a seguir detalhados e associados aos blocos funcionais responsáveis pela operação, que ocorre na seqüência das fases.

Durante o processo de recepção, todos os sinais envolvidos são sincronizados nas bordas de subida ou de descida do relógio de recepção, o sinal RxClk, fornecido pelo PHY. Na borda de subida do relógio, o sinal Rxds apresenta o início de cada bit do fluxo serial, enquanto que na borda de descida os bits são amostrados, exatamente na metade do tempo de duração de um bit. O tempo de duração de um bit é de 100 ns para uma taxa de 10 Mbps, totalizando 800 ns para a recepção completa de um byte.

O sinal Rxdout, quando é ativado, sinaliza, com exatidão, o término do SFD e o correspondente início do quadro (primeiro bit do campo DA). Este sinal é gerado pelo bloco detector de SFD (Figura 5.14) que é habilitado pelo controlador ao executar o handshake e detectar a presença de portadora via sinal CRS, conforme ilustrado no diagrama de tempos do início da recepção da Figura 5.16.

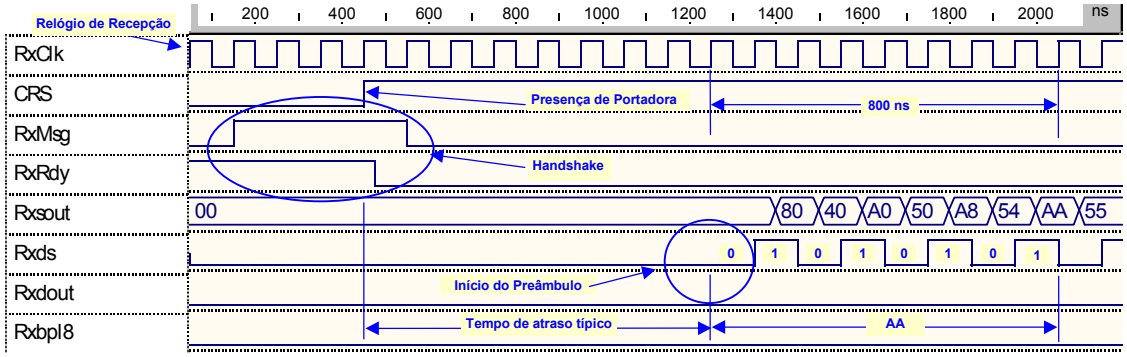


Figura 5.16 – Diagrama de tempos do início da recepção.

Neste diagrama, observa-se que existe um tempo de atraso entre a sinalização da presença de portadora e o efetivo início do preâmbulo, que é relativo ao tempo de aquisição do dispositivo decodificador pertencente ao PHY (tipicamente, 900 até 1500ns) [LEV98]. Em termos de simulação e testes, este atraso foi fixado em 800ns, representando um atraso de tempo equivalente a um byte. Este atraso não prejudica o correto funcionamento do detector de SFD, pois este foi projetado para aguardar o início do preâmbulo, contar o máximo de 7 bytes de valor hexadecimal "AA" e detectar o próximo, cujo valor deve ser "AB", desta forma sinalizando o início do campo DA, via o sinal Rxdout ativo na fronteira do byte SFD. Isto está representado na Figura 5.17.

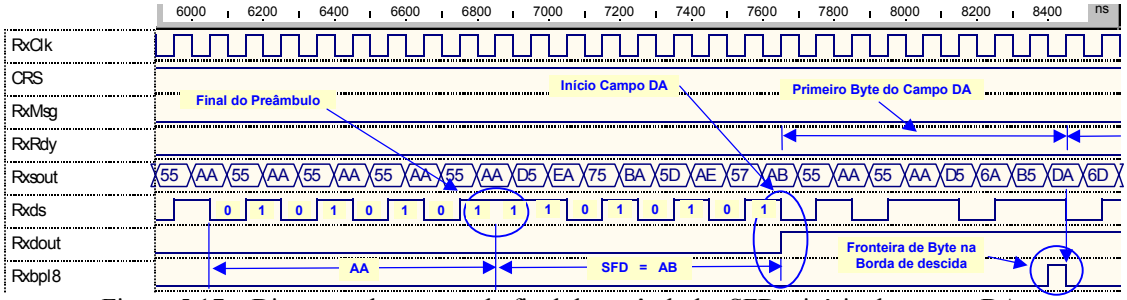


Figura 5.17 – Diagrama de tempos do final do preâmbulo, SFD e início do campo DA.

Neste momento, o controlador inicializa o bloco contador de bytes/delimitador de campo, habilita o gerador de CRC e o filtro DA. O bloco contador de bytes é responsável pela geração do sinal Rxbpl8, utilizado para determinar a fronteira de byte na sua borda de descida. Este sinal é utilizado por todos os blocos que recebem o barramento Rxsout, conforme a Figura 5.17 e a Figura 5.18. Os

blocos utilizam-se deste sinal para sincronizar a leitura dos bytes que são convertidos de serial para paralelo no registrador de deslocamento.

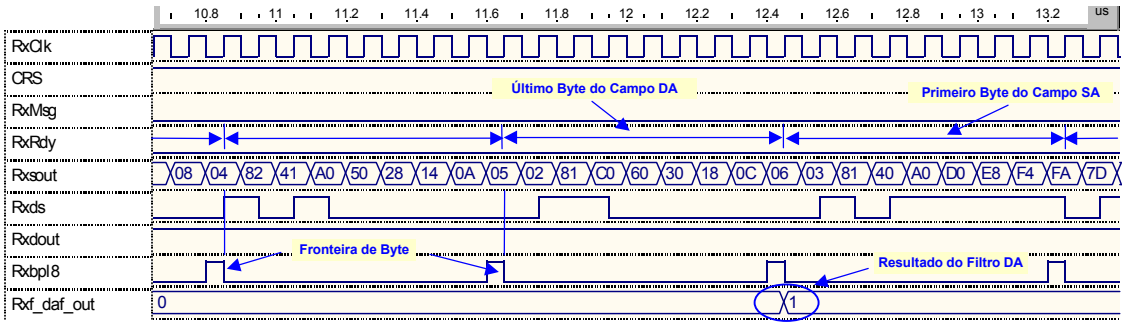


Figura 5.18 – Diagrama de tempos do final do campo DA, resultado do filtro DA e início do campo SA.

O filtro DA, por sua vez, utiliza a borda de descida do sinal Rxbpl8 para registrar cada um dos seis bytes de endereço do campo DA. Os seis bytes são então comparados com valores constantes, previamente armazenados em registradores internos ao filtro DA. Estes valores representam o endereço físico do controlador de rede implementado pelo IP Core. O resultado da comparação de cada um dos bytes registrados com o correspondente de endereço físico ou com a constante "FF", é computado. Ao final do sexto byte do campo DA, o resultado é apresentado ao controle de recepção pelo sinal Rxf_daf_out. Este sinal é composto de dois bits, que são combinados para representar o valor 1 (endereço individual) caso os seis bytes de endereço recebidos coincidam um a um com os seus correspondentes físicos, o valor 2 (endereço de difusão), caso todos os seis bytes sejam iguais a constante "FF" e o valor zero caso nenhum dos dois anteriores ocorra. Um valor zero como resultado é interpretado pelo controlador como um endereço não-válido. Isto provoca o descarte da operação de recepção do quadro. O RxMAC, sinaliza para a interface MAC-Usuário, via sinal RxSta=0, que não foi armazenada uma mensagem no buffer de recepção. Além disso, o RxMAC sinaliza pronto, ativando o sinal RxRdy e volta ao início da rotina de recepção, aguardando o IPG terminar para iniciar a recepção do próximo pacote.

Nesta fase do processo de recepção (Figura 5.18, final do campo DA), o contador de bytes registra a recepção de seis bytes e o bloco contador de endereço, habilitado pelo controlador de recepção no inicio do campo DA, indica que os seis bytes, correspondentes ao DA, estão armazenados no buffer de recepção. Caso o filtro DA apresente um valor não nulo no sinal Rxf_daf_out, o controlador continua a operação de recepção, armazenando os bytes subseqüentes no buffer.

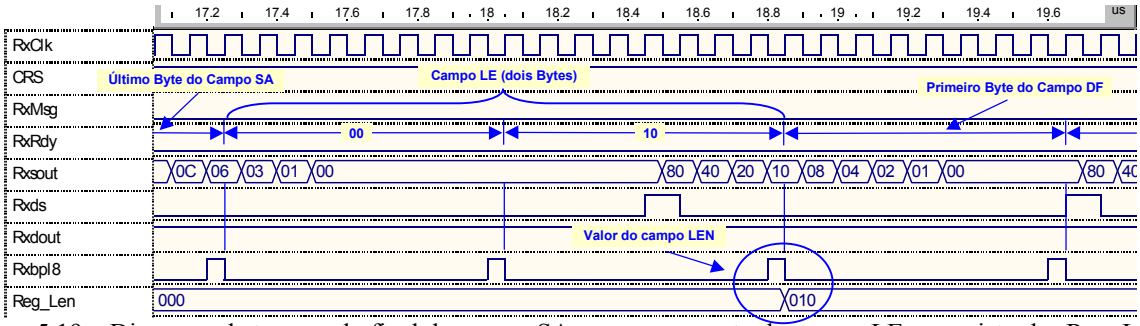


Figura 5.19 – Diagrama de tempos do final do campo SA, armazenamento do campo LE no registrador Reg_Len e início do campo DF.

O contador de bytes, interno ao bloco contador de bytes/delimitador de campo, compara a contagem dos bytes com valores constantes pré-determinados ("000D" e "000E", do campo LE) para localizar e registrar os dois bytes referentes ao campo LE, no registrador Reg_Len, conforme ilustrado na Figura 5.19. O Reg_len é acrescido da constante 14 e o total é comparado com o valor do contador

de bytes. Caso o contador seja maior, o sinal delimitador de campo Rxb_Ctr_14L é ativado, sinalizando um bit antes do término do campo de dados, conforme representado na Figura 5.20.

O contador de bytes/delimitador de campo é responsável pela geração de mais dois sinais de controle. O sinal Rxb_Ctr_60B, quando ativado sinaliza um bit antes do início do campo FCS. O sinal Rxb_Ctr_Max (não ilustrado nos diagramas de tempos), quando ativado sinaliza que o contador atingiu contagem de 1518 bytes. Os sinais Rxb_Ctr_60B, Rxb_Ctr_14L e Rxb_Ctr_Max são constantemente verificados pelo controlador, pois a combinação dos valores destes permite ao controlador sinalizar o tipo de erro ocorrido na recepção da mensagem atualmente armazenada no buffer, caso algum erro ocorra. A detecção de erros se dá pela análise, pelo controlador dos valores dos sinais mencionados acima combinado com o valor do sinal CRS, que informa a perda prematura de portadora antes do final da recepção do pacote completo.

A Figura 5.20 ilustra o momento em que o controlador verifica que o sinal Rxb_Ctr_14L foi ativado, informando que o penúltimo bit do último byte do campo DF foi recebido e que este é o último byte a ser armazenado no buffer de recepção.

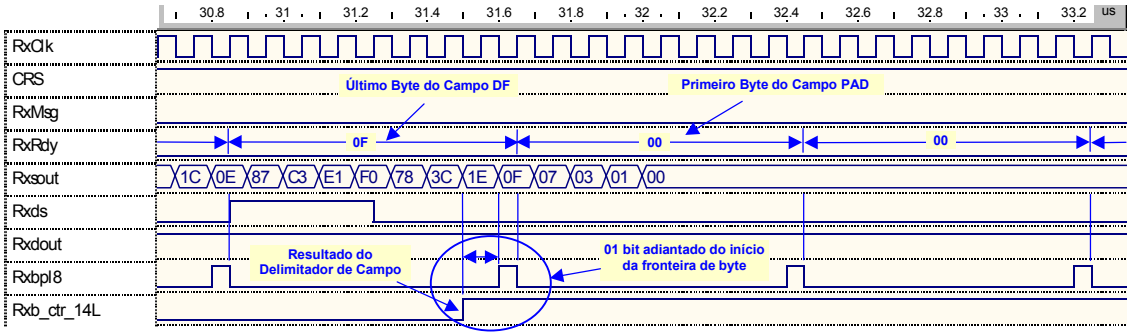


Figura 5.20 – Diagrama de tempos do final do campo DF, resultado do delimitador de campo e início do campo PAD.

Nesta fase, os bits de enchimento (PAD) são descartados e não são armazenados no buffer. Contudo, continuam sendo computados no gerador de CRC, para que ao final do campo de PAD, sinalizado pelo sinal Rxb_Ctr_60B, o gerador seja desabilitado para reter o valor computado no barramento de 32 bits (Rxg_crc_res), conforme ilustrado na Figura 5.21. Isto ocorre desta forma para o exemplo usado, onde existem bits de enchimento. Se o campo de dados contiver mais do que 46 bytes, o funcionamento depende do sinal Rxb_Ctr_14L.

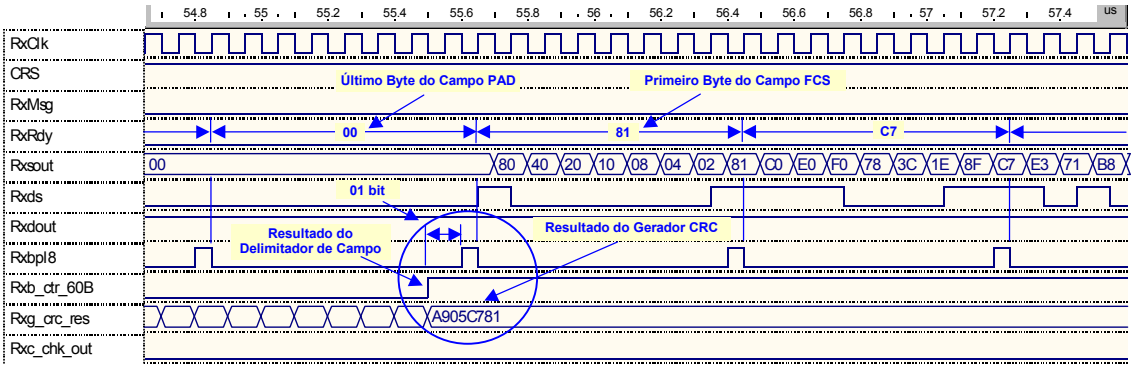


Figura 5.21 – Diagrama de tempos do final do campo PAD, resultado do delimitador de campo e início do campo FCS.

Na Figura 5.22, o sinal Rxc_chk_out informa o resultado da comparação de um a um dos bytes do campo FCS recebidos com as respectivas posições de oito bits do barramento Rxg_crc_res. Caso os quatro bytes confirmem igualdade, o controlador termina a recepção sinalizando para a interface

MAC-Usuário (via sinal RxSta=1) que foi armazenada uma mensagem válida no buffer de recepção. Ele também sinaliza pronto, ativando o sinal RxRdy e volta ao início da rotina de recepção, aguardando o IPG terminar para iniciar a recepção do próximo pacote.

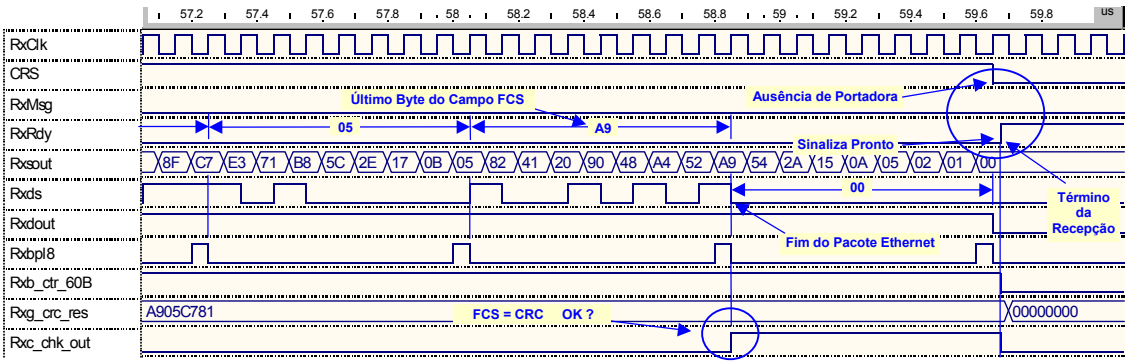


Figura 5.22 – Diagrama de tempos do resultado do CRC gerado comparado com os bytes do FCS e término da recepção.

Finalmente, caso tenha ocorrido erro de CRC (via sinal Rxc_chk_out=0 após o bit final do último byte do campo FCS), o controlador testa a existência de portadora para detectar qual o tipo específico de erro ocorrido, o erro de FCS ou de alinhamento.

6. Estratégia de Validação Funcional do Projeto

Em paralelo com o processo de desenvolvimento do IP Core MAC Ethernet, um conjunto de módulos de hardware e software necessários para prover a comunicação entre o hospedeiro e o MAC foram desenvolvidos e validados. Para facilitar o futuro teste do protótipo em hardware, a estratégia de validação funcional do projeto MAC empregou a estrutura já validada destes módulos. Estes compreendem um módulo de software e um módulo de hardware. O software, denominado *Ether_Serial*, é responsável pela comunicação via interface serial, do lado do hospedeiro. O módulo de hardware, denominado *MAC-Usuário*, permite a comunicação entre o MAC e o hospedeiro via uma interface serial RS-232 existente na plataforma de prototipação usada. Estes módulos são descritos em mais detalhe no Capítulo 7. Contudo, referências a estes aparecerão necessariamente neste Capítulo, dada a estratégia de validação escolhida, que tem como vantagem vetores de teste únicos utilizados tanto na fase de validação funcional como na fase de prototipação.

A estratégia de validação funcional do projeto do IP Core MAC Ethernet é baseada no uso de *testbenches* [BER00]. Um testbench é uma bancada de testes virtual, implementada como uma descrição em VHDL [MAZ92], que contém uma instância VHDL do módulo a testar, junto com a descrição de geradores de estímulos, capturadores de saídas e formatadores de resultados. Esta estrutura é ilustrada na Figura 6.1. Testbenches são projetados para executar testes de um módulo de forma automática ou semi-automática. O módulo sob teste funcional, em geral, necessita de sinais de relógio e inicialização para o sincronismo e seqüenciamento de operações. Estas últimas, para serem executadas, também necessitam de estímulos de entrada adequados para se obter as respostas em conformidade com o projeto na saída dos módulos em teste, como exemplificado na Figura 6.1.

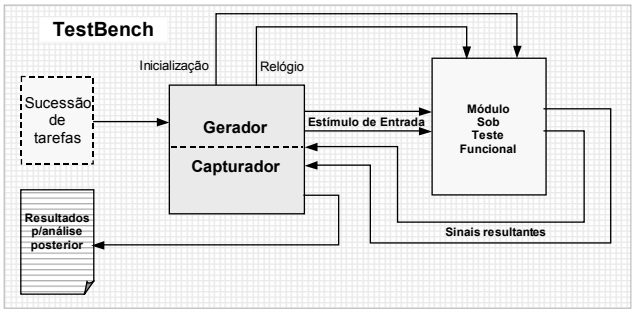


Figura 6.1 – Diagrama de blocos do método testbench.

O testbench é um sistema autônomo, que descreve o comportamento do ambiente externo, instanciando o módulo sob teste e interagindo com este. Os estímulos são produzidos a partir da especificação de uma sucessão de *tarefas* previamente preparadas para o módulo em teste funcional. As tarefas em geral são seqüências de dados que serão processados pelo módulo em teste. Estas devem ser criteriosamente elaboradas para representar as tarefas que o protótipo no futuro deverá realmente processar conforme foi especificado no projeto. Na Figura 6.1, os módulos *Sucessão de tarefas* e *Resultados para análise posterior* representam em geral informação armazenada em arquivos. Esta estrutura permite construir testbenches parametrizáveis. Os resultados do processamento, após adaptados pelos capturadores de sinais são armazenados para análise.

Na validação funcional do projeto do IP Core MAC Ethernet, devido à complexidade das tarefas e dos sinais envolvidos, surgiu a necessidade de utilização de dois testbenches separados. Dividiu-se em duas fases o teste funcional, uma de validação do TxMAC, que analisa a conformidade na transmissão de pacotes Ethernet e outra de validação do RxMAC que analisa a conformidade na recepção de pacotes Ethernet.

O módulo MAC-Usuário foi desenvolvido de forma a possibilitar a verificação funcional do IP Core tanto nas fases de validação do TxMAC e RxMAC como na protipação final, conforme será visto no Capítulo 7. Neste capítulo, o módulo MAC-Usuário é empregado na validação de ambos módulos, TxMAC e RxMAC, visto que ele realiza a função de interface entre o sistema hospedeiro e o IP Core. Outros dois módulos intermediários são os Buffers de Transmissão e Recepção, cuja validação funcional foi realizada de forma indireta, pois o testbench não atua diretamente sobre os mesmos.

Na fase de validação do TxMAC, o Gerador gera estímulos (incluindo as mensagens a serem transmitidas) para o módulo MAC-Usuário, que por sua vez gera novos sinais para transferir as tarefas ao módulo TxMAC do IP Core. O módulo TxMAC executa e gera os sinais (incluindo os pacotes Ethernet) que são capturados pelo Capturador, parcialmente analisados e armazenados. Na fase de validação do RxMAC, de forma semelhante, o Gerador gera estímulos (incluindo os pacotes Ethernet) para o módulo RxMAC do IP Core. Este, por sua vez, produz novos sinais para o módulo MAC-Usuário que gera novos sinais para transferir as mensagens recebidas ao Capturador que as analisa e armazena.

Este Capítulo tem como objetivo apresentar uma amostra do conjunto de testes realizados durante o desenvolvimento e a validação funcional dos módulos MAC-Usuário, TxMAC e RxMAC. Os testes aqui descritos são um apanhado resumido e simplificado, centrados nos eventos mais significativos que ocorrem no método de acesso ao meio CSMA/CD de redes Ethernet. O objetivo não é completude de descrição do processo de validação empreendido, mas demonstrar a complexidade do problema de validação e ilustrar as formas encontradas para efetuar esta. Uma explicação mais completa de todo o testbench está fora do escopo deste trabalho, mas o texto completo em VHDL do testbench aparece nos ANEXOS E e D. Salienta-se que todas as formas de onda mostradas neste Capítulo correspondem a resultados de simulação usando os testbenches descritos.

Na próxima Seção é descrito resumidamente o módulo MAC-Usuário, visto que este é a interface de comunicação do hospedeiro com o IP Core MAC Ethernet. Durante os testes de validação funcional do MAC, conseqüentemente, o módulo MAC-Usuário também é testado a validado, pois o mesmo é responsável pela transferência de mensagens do hospedeiro ao IP Core MAC e vice versa, que desta forma adapta o MAC a ambientes específicos, conforme proposto na Seção 5.1. As Seções 6.1 e 6.2, detalham a estrutura dos dois testbenches (TestBench-TX e TestBench-RX) elaborados de forma independente e isolada um do outro, tendo em comum apenas o módulo MAC-Usuário para adaptar o IP Core ao ambiente de teste. Maiores detalhes estão disponíveis nos Anexos, que contêm também descrições VHDL e máquinas de estados Finitas (FSMs) da interface MAC-Usuário.

6.1 Fase de Validação Funcional do TxMAC

O módulo *Testbench-TX* compõe-se de um módulo *Gerador/Capturador* conectado ao Controle deTransmissão de Mensagens da interface MAC-Usuário e ao Controle de Transmissão de Pacotes do módulo TxMAC. O módulo Gerador/Capturador, conforme ilustrado na Figura 6.2, é responsável pela geração da seqüência de bits de dados na linha TXD no padrão RS-232 (start bit, 8 bits de dados, um stop bit, sem paridade) que representam a mensagem a ser transferida ao buffer de transmissão. No Anexo D é descrito em VHDL o módulo Gerador/Capturador que foi desenvolvido para validar o módulo TxMAC.

O Gerador/Capturador produz internamente os sinais de relógio TxClk e SClk. O TxClk é uma forma de onda quadrada com freqüência de 10 Mhz e ciclo de serviço de 50%. Este está sincronizado com SClk de 20 Mhz, também uma onda quadrada com ciclo de serviço de 50% (Anexo D, linhas 111 a 121). O primeiro sinal é utilizado pelo Controle de Transmissão de Pacotes na geração da taxa de transmissão do pacote Ethernet (10 Mbps), enquanto o segundo é destinado aos dois módulos, MAC-Usuário e TxMAC como relógio global do sistema. Este último sinal aciona FSMs, contadores e comanda o sincronismo, e de forma indireta controla o Buffer de Transmissão.

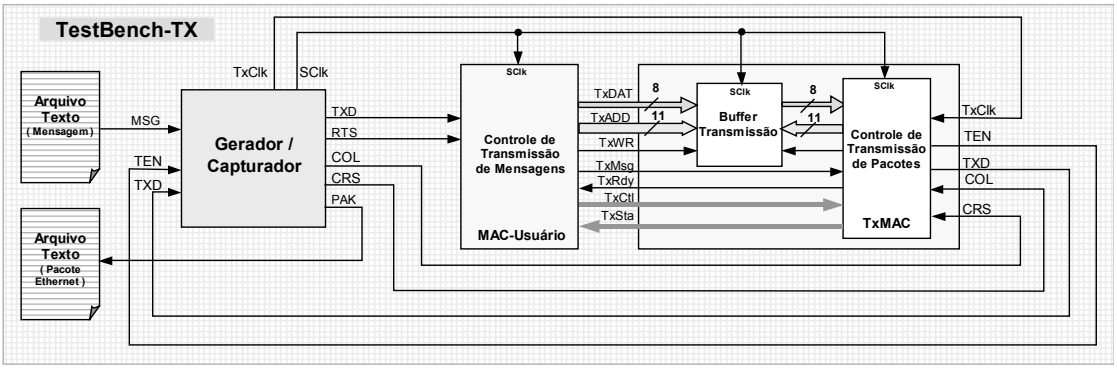


Figura 6.2 – Diagrama de blocos do TestBench_TX na fase de validação do TxMAC.

A mensagem é lida de um arquivo texto e convertida para uma seqüência de bytes, que é serializada em uma seqüência de bits com taxas de transmissão padronizadas (9,6/19,2/38,4/76,8 ou 115,2 Kbps, conforme Anexo D, linhas 123 até 127). Notar que esta tarefa, realizada aqui por simulação via primitivas de manipulação de arquivos de bibliotecas VHDL, equivale à execução do software Ether_Serial na prototipação (ver Capítulo 7). O arquivo texto contém inicialmente três bytes responsáveis pelo protocolo de comunicação do software com a interface MAC-Usuário, conforme ilustrado na Figura 6.3. O primeiro (55H) é utilizado somente na primeira transmissão, serve para inicializar o bloco Gerador de Clock Rxd/Txd, por intermédio do bloco Detector de Baud Rate (Seção 7.2, Figura 7.2 e Anexo C). O segundo é utilizado como preâmbulo do protocolo de comunicação da interface MAC-Usuário.

A interface MAC-Usuário, após receber o preâmbulo, interpreta o próximo byte como um byte de controle. Por exemplo, 01H, prepara o bloco Controle de Transmissão de Mensagens (Seção 7.2, Figura 7.2) para receber os bytes subseqüentes e armazená-los no buffer de Recepção. Outros códigos são reservados para programação e controle de configuração do IP Core MAC Ethernet.

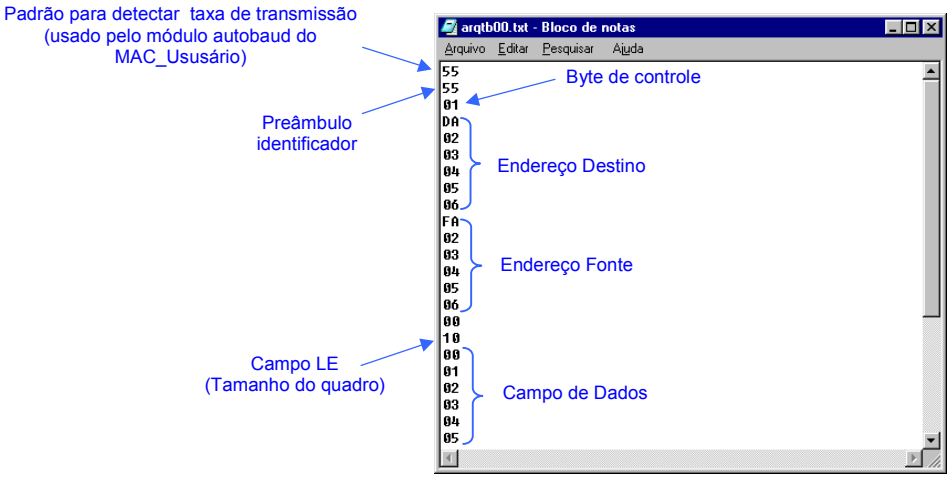


Figura 6.3 – Arquivo texto com a mensagem a ser transferida.

O módulo Gerador/Capturador, conforme ilustrado na Figura 6.4, possui um contador de ciclos do relógio SClk (SckgCtr, Anexo D, linhas 141 a 156), que é utilizado para simular e validar a fase de deferimento, o tempo de espera aleatória exponencial truncada (backoff) e a operação de reforço de colisão (JAM) com a geração do sinal Col. Este último sinal também é utilizado para determinar o instante no qual é iniciada a sucessão de tarefas a serem executadas nos módulos sob teste. O contador SckgCtr foi projetado para incrementar a contagem na borda de subida e na borda de descida para aumentar a precisão dos momentos em que os estímulos são aplicados.

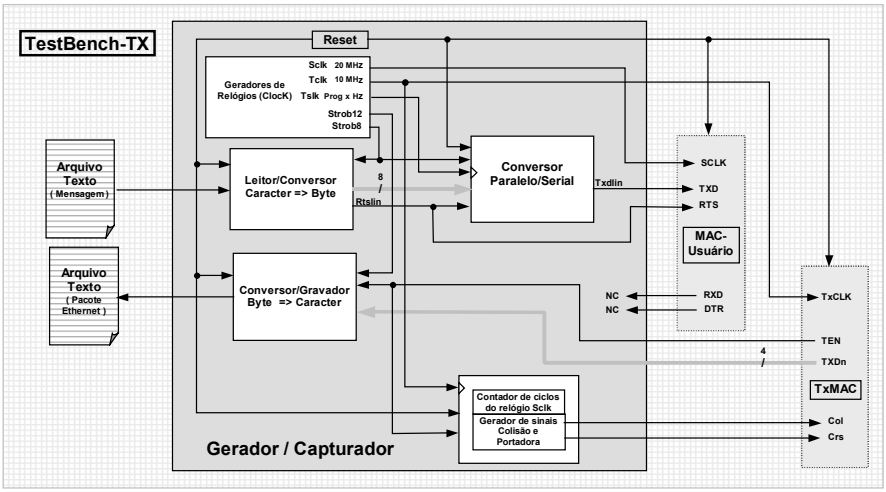


Figura 6.4 – Diagrama de blocos do módulo Gerador/Capturador doTestbench-TX.

Uma vez que a mensagem transmitida serialmente pelo sinal Txdlin esteja totalmente armazenada no buffer de Transmissão, o MAC-Usuário realiza o handshake (Seção 5.2) com o TxMAC (Anexo D, linhas 258 a 244). A partir deste ponto, o módulo Gerador/Capturador aguarda o sinal TEN (habilitação de transmissão, TxMAC, Figura 6.4), que indica o início da transmissão do pacote Ethernet e então inicia-se a recepção via o sinal TXDn (transmissão de 4 em 4 bits como visto da Seção 5.3) ou TXDs (de interface serial). O sinal TEN (Txen para simulação) habilita o contador SckgCtr para o bloco Gerador de sinais de Col e Crs.

Na Figura 6.5, por outro lado, pode-se observar uma colisão programada na contagem de 400 ciclos do relógio SClk (1 ciclo de SClk é igual a 0,05 µs), a partir do início da transmissão (txen = '1'), com duração de 130 ciclos de SClk (3,25 µs). Após a colisão, o TxMAC executa a operação de JAM (Seção 4.5) durante 3,2 µs e encerra a transmissão (txen = '0') em 531 ciclos de SClk.

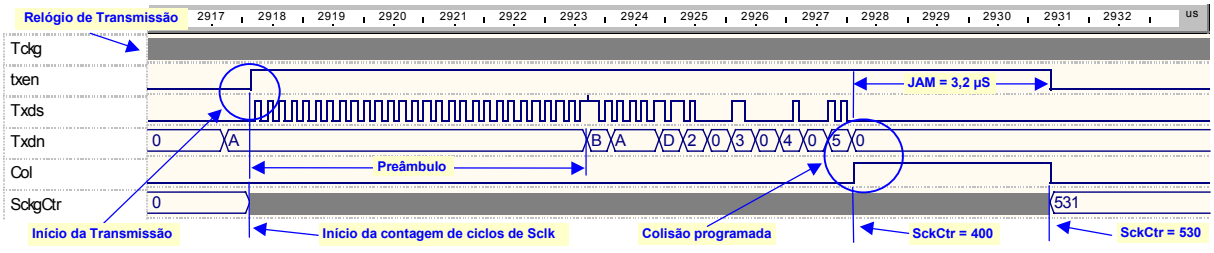


Figura 6.5 – Diagrama de tempos da simulação de colisão.

A colisão ocorre no início do sexto byte do campo DA. Após a seqüência JAM (32 bits em "0") o TxMAC reinicia a fase de deferimento e a respectiva retransmissão do pacote Ethernet para ser recebido no módulo Gerador/Capturador através do sinal Txdn (4 bits). Os nibbles gerados no TxMAC são concatenados e gravados byte a byte em um arquivo texto para posterior análise,

conforme ilustrado na Figura 6.6 (Anexo D, linhas 246 a 295). Na Figura, podemos observar que a seqüência de bytes do campo DA foi interrompida no byte 05H seguida pela seqüência de quatro bytes 00H (32 bits em "0") sucedida de 7 bytes do preâmbulo, SFD e subseqüentes bytes da mensagem contida no pacote em retransmissão.

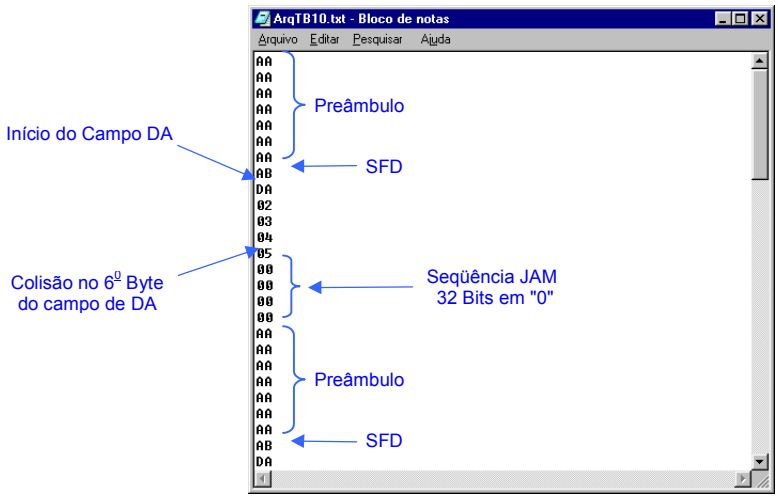


Figura 6.6 – Arquivo texto com pacote transmitido e retransmitido após colissão.

A técnica de simulação utilizando o SckgCtr é demonstrada na validação do tempo de espera aleatória exponencial truncada. Na Figura 6.7 pode-se observar que o sinal de colisão (Col) permanece em nível '1' para simular 16 colisões consecutivas. Desta forma, são avaliados os tempos de espera, representados pelo vetor de 11 bits (rtimes). Este vetor é atualizado de forma aleatória e limitado pelo vetor "maxrnd" que cresce de forma exponencial a cada nova colisão, maxrnd, por sua vez é controlado pelo contador de colisões (atmpts).

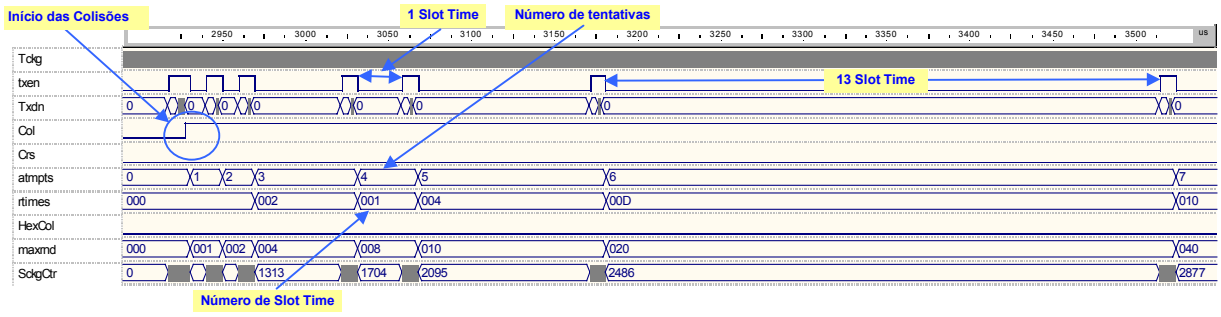


Figura 6.7 – Diagrama dos seis primeiros tempos de espera aleatória exponencial truncada.

Quando o contador atmpts atinge 16 colisões consecutivas, conforme ilustrado na Figura 6.8, ao final do décimo quinto tempo de espera, o sinal HexCol vai para o nível '1'. Isto informa ao controle de transmissão do módulo TxMAC para interromper o processo de transmissão do pacote e configura um erro de colisões excessivas que é considerado um erro de transmissão. Nesta situação especial, o controlador permanece aguardando um sinal de "restart" (proveniente do sistema usuário) para voltar novamente ao processo de transmissão.

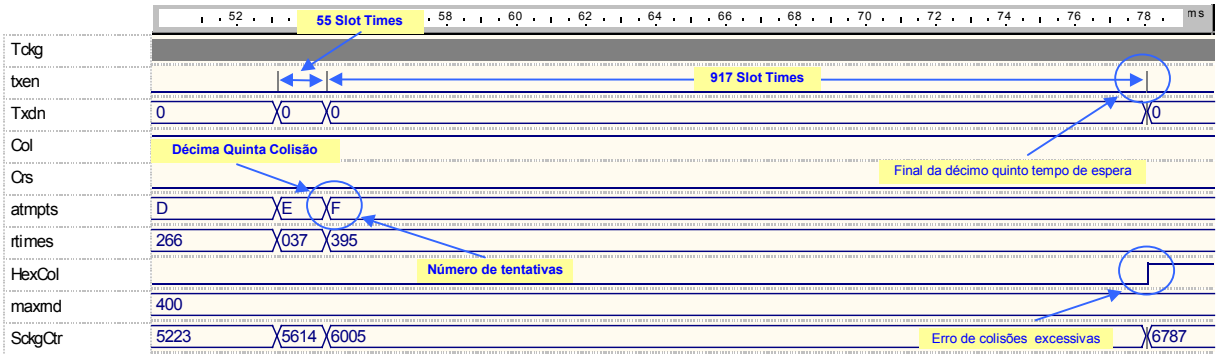


Figura 6.8 – Diagrama dos dois últimos tempos de espera aleatória exponencial truncada.

O sinal de portadora presente (CRS) permanece em nível '0', durante todas as dezesseis colisões, para simular e testar o gerador de backoff (Seção 4.4.2). Esta situação não corresponde estritamente à realidade. Porém, ela foi necessária para evitar que o módulo de controle de transmissão de mensagens fique aguardando o meio de transmissão se tornar livre para executar a fase deferimento e garantir o tempo mínimo entre pacotes (IPG, Seção 5.3), alongando por demais o tempo de simulação.

A técnica de simulação utilizando o SckgCtr é também empregada na validação funcional do RxMAC. Para validação do RxMAC foi necessária a utilização de um segundo contador (TskgCtr) de sinais de relógio Tskg. Este é o relógio para transmissão serial do Gerador/Capturador para o Controle Transmissão do módulo MAC-Usuário na inicialização da fase de validação funcional do RxMAC, como será descrito na próxima Seção.

6.2 Fase de Validação Funcional do RxMAC

Conforme pode-se notar pelo exame da Figura 6.9 o diagrama de blocos da fase de validação do RxMAC é semelhante ao da fase de validação do TxMAC. Na fase de validação do RxMAC, o *TestBench-RX* é constituído pelo módulo de Controle de Recepção de Pacotes do RxMAC interconetado, via buffer de recepção, ao Controle de Recepção de Mensagens da interface MAC-Usuário. A interface MAC-Usuário, por sua vez, está conectada ao módulo Gerador/Capturador e este foi especialmente desenvolvido para validar o módulo RxMAC através do emprego da interface MAC-Usuário.

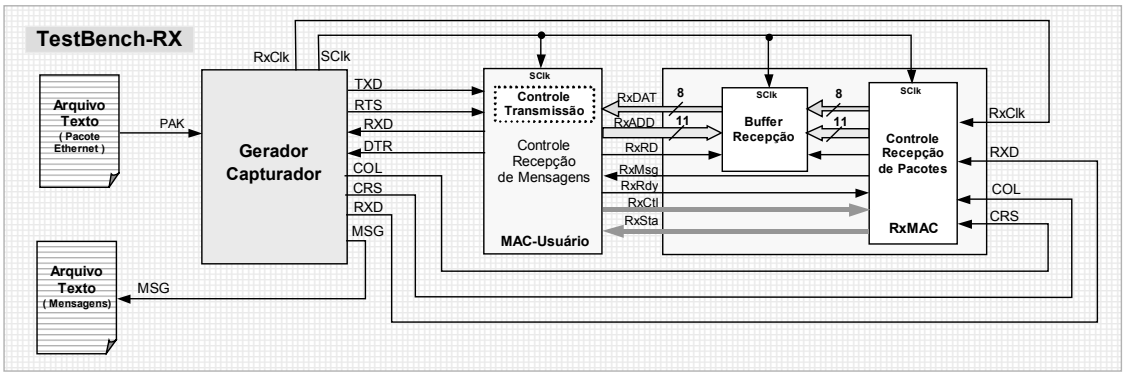


Figura 6.9 – Diagrama de blocos do TestBench-RX na fase de validação do RxMAC.

O módulo Gerador/Capturador, conforme ilustrado na Figura 6.9, é responsável pelos os sinais de relógio RxClk e Sclk, de forma análoga ao TestBench-TX. O RxClk é uma forma de onda quadrada de freqüência de 10 Mhz com ciclo de serviço em 50% e está sincronizado com SClk de 20 Mhz, também onda quadrada ciclo de serviço em 50% (Anexo D, linhas 111 a 121). O primeiro relógio é utilizado pelo Controle de Recepção de Pacotes na geração da taxa de transmissão do pacote Ethernet (10 Mbps). O segundo é destinado aos dois módulos, MAC-Usuário, TxMAC como relógio do sistema como um todo (relógio para as FSMs de controle, contadores e sincronismo) e de forma indireta do Buffer de Recepção (somente como sinal de sincronismo).

O bloco Gerador de Relógios, ilustrado na Figura 6.10 , além de produzir os sinais de relógio para os módulos MAC-Usuário, TxMAC e Buffer de Recepção, também é responsável pela criação dos sinais de relógio internos ao módulo Gerador/Capturador para geração da seqüência de bits (TXD) no padrão RS-232.

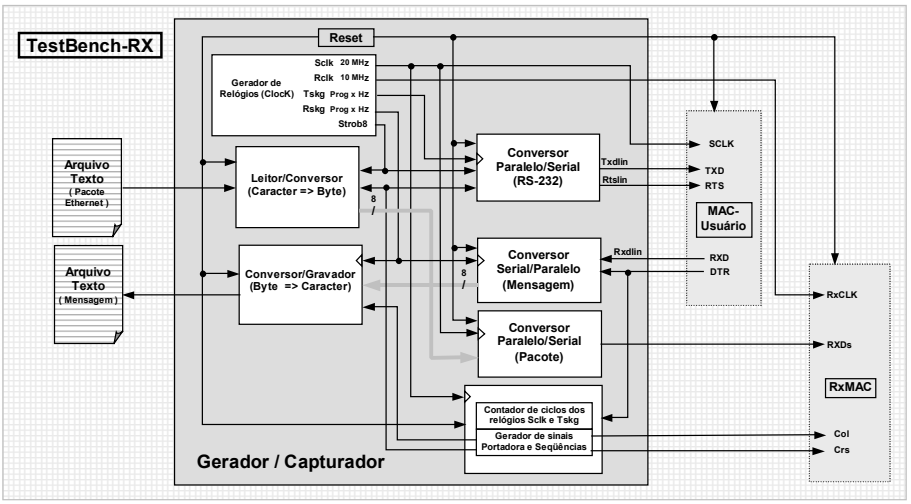


Figura 6.10 – Diagrama de blocos do módulo Gerador/Capturador do TestBench-RX.

Na fase de recepção, o bloco Conversor Paralelo/Serial (RS-232) é responsável pela transmissão de inicialização do módulo MAC-Usuário. Este conversor transmite (de forma parametrizável) o padrão para a detecção da taxa de transmissão, o padrão identificador, o byte de controle e o byte parâmetro conforme descrição em VHDL no Anexo E, linhas 206 a 243. Este procedimento é necessário apenas para a inicialização do gerador de relógio RXD/TXD do módulo MAC-Usuário conforme descrito na Seção 6.1. O gerador de relógio RXD/TXD é ajustado pelo bloco detector de baud rate com a contagem de tempo para a taxa de transferência previamente determinada pelo bloco Geradores de Relógios do módulo Gerador/Capturador (Anexo E, linhas 122 a 150).

A seqüência inicial de bytes representa um protocolo de comunicação utilizado pela interface MAC-Usuário na inicialização dos Módulos TxMAC e RxMAC, definindo os sinais TxCtl e RxCtl na configuração dos modos de operação do IP Core (Seção 5.2). Esta seqüência inicial, controlada pelo contador TskgCtr é ativada, via sinal Rtslin, pelo bloco contador de ciclos dos relógios Tslk e Tskg e o Gerador de sinais de estímulos, descrito em VHDL no Anexo E, linhas169 a 204.

Na Figura 6.11, pode-se observar que a transmissão da seqüência inicial é gerada após o estímulo Rtslin (contador TskgCtr em 2 ciclos de Tskg). Este estímulo representa o sinal RTS da interface RS-232. Ele informa ao controle de transmissão de mensagens do módulo MAC-Usuário que é chegado o momento de iniciar a transferência dos bytes de inicialização (55H, 55H, 02H, FFH), conforme parametrizado no módulo Gerador/Capturador (Anexo E, linhas linhas169 a 204).

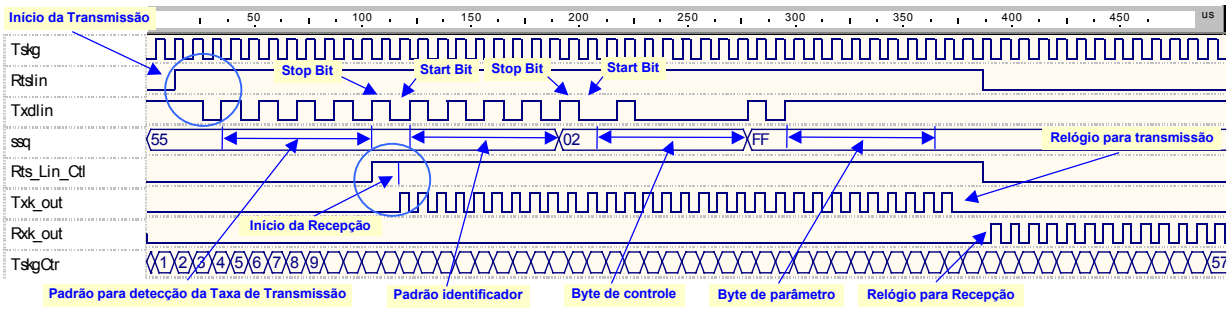


Figura 6.11 – Diagrama de tempos da simulação da transmissão de inicialização do módulo MAC-Usuário.

Após a receber o padrão para detectar a taxa de transmissão, o gerador de relógio RXD/TXD é ajustado pelo bloco detector de baud rate. O relógio Txk_Out é então utilizado para sincronismo de recepção dos próximos bytes da seqüência inicial (Padrão identificador, Byte de controle, Byte de parâmetro) conforme representado na Figura 6.11. Ao final da transmissão, o relógio Txk_Out é desabilitado e o relógio Rxk_Out é habilitado para ser utilizado pelo controle de recepção de mensagens do MAC-Usuário.

A Figura 6.12 ilustra a seqüência de eventos para validação funcional do RxMAC. Após a transmissão da seqüência inicial, o bloco Gerador de Sinais Portadora e Seqüências do módulo Gerador/Capturador, controlado pelos contadores TskgCtr e SckgCtr, leva o estímulo de portadora (Crs) ao nível '1'. Isto sinaliza ao RxMAC o início da recepção do pacote.

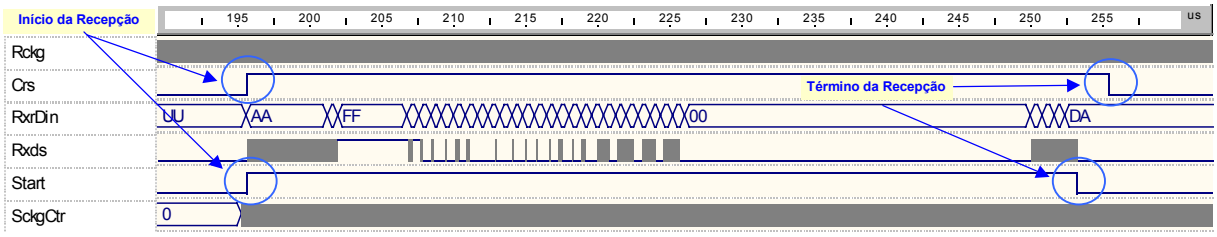


Figura 6.12 – Diagrama de tempos da recepção do pacote pelo RxMAC.

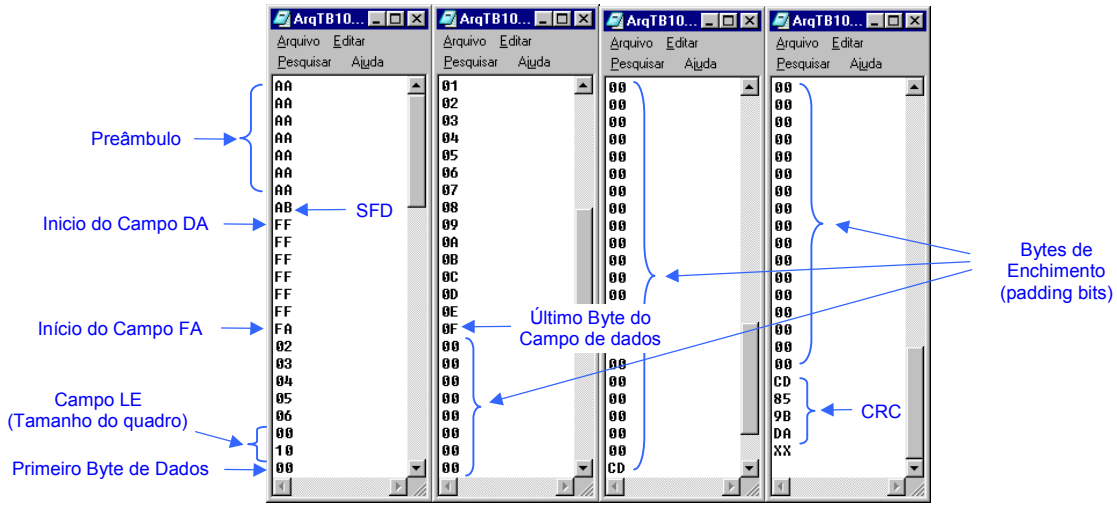


Figura 6.13 – Arquivo texto com o pacote Ethernet a ser recebido pelo RxMAC.

O estímulo "Start", interno ao módulo Gerador/Capturador, é simultaneamente levado ao nível '1,' habilitando os blocos Leitor/Conversor e Conversor Paralelo/Serial (Pacote), ilustrados na Figura 6.10, a iniciarem a seqüência de bits. Esta representa o pacote Ethernet previamente armazenado em um arquivo texto, conforme ilustrado na Figura 6.13.

Na fase de recepção, são detectados os possíveis erros de transmissão. O RxMAC, após verificar uma perda da portadora, interrompe a recepção e deixa de transferir os bytes recebidos para o buffer de recepção. Para validar esta funcionalidade do RxMAC é simulada uma perda de portadora no módulo Gerador/Capturador (Anexo E, linha 198, SckgCtr = 400) conforme ilustrado no diagrama de tempos da Figura 6.14.

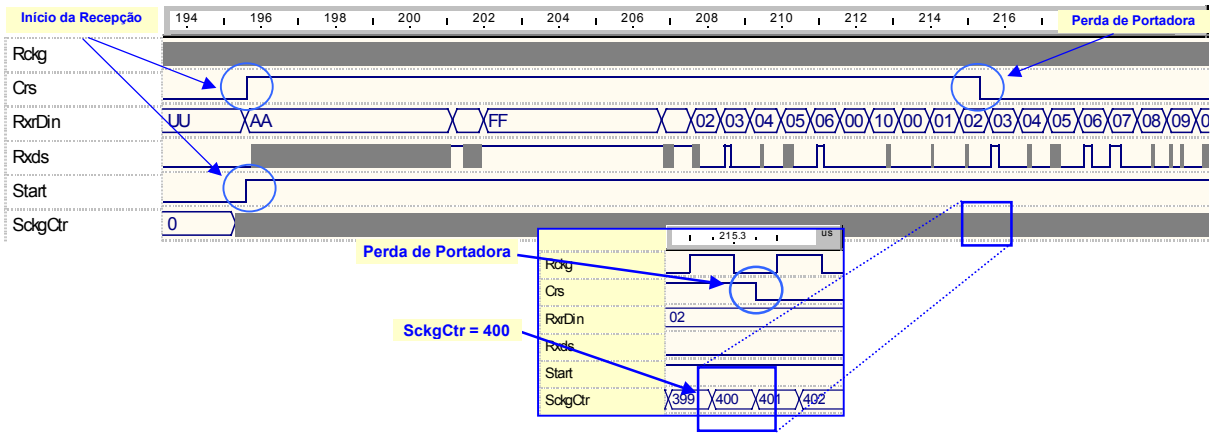


Figura 6.14 – Diagrama de tempos na recepção do pacote pelo RxMAC com perda de portadora.

O RxMAC, após interromper a recepção, realiza o handshake (Seção 5.2) com a interface MAC-Usuário sinalizando o erro, de acordo com o instante em que ocorreu a perda de portadora em relação ao pacote em recepção. Isto é feito codificando no barramento de status RxSta (Seção 5.4) com o valor 3H, indicando perda de portadora no campo de dados. Este valor é enviado pela serial usando o barramento Rxd_Dat_Byt ilustrado na Figura 6.15. Este valor (3H) é o nibble mais significativo do byte de status, composto com o valor codificado barramento de status do Controle de Transmissão (TxSta, valor 0H), sendo este o nibble menos significativo para completar o barramento Rxd_Dat_Byt. Esta composição de nibbles é transferida ao módulo Gerador/Capturador pelo controle de Recepção de Mensagens da interface MAC-Usuário, como o Byte de Status que precede o Preâmbulo Identificador (55H).

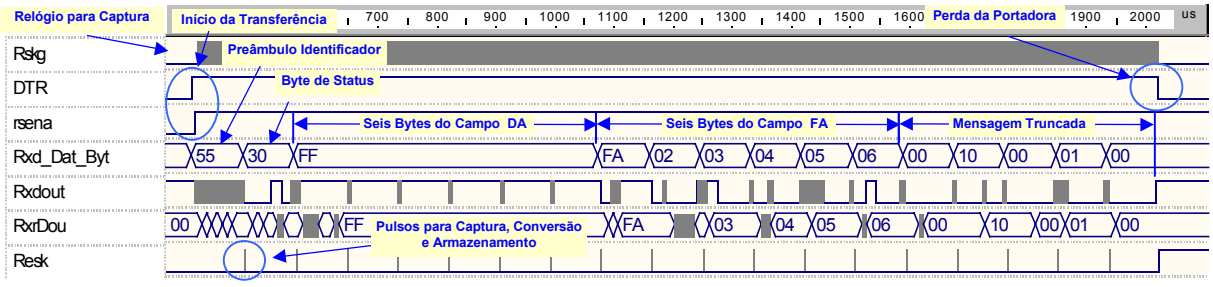


Figura 6.15 – Diagrama de tempos da tranferência do byte de status e a mensagen truncada.

Após o handshake com a interface MAC-Usuário via controle de Recepção de Mensagens, a interface ativa o sinal DTR (em nível '1') informando o início da transferência da mensagem (truncada) ao módulo Gerador/Capturador, conforme a Figura 6.15. Este módulo, por sua vez, recebe a seqüência de bits na linha RXD. O sinal RxdOut do conversor paralelo/serial do MAC-Usuário, simulando a interface RS-232, deverá ser utilizado para a validação na fase de prototipação do IP Soft Core conforme será descrito no próximo Capítulo.

Durante a transferência da mensagem, o módulo Gerador/Capturador utiliza os blocos Conversor Serial/Paralelo (mensagem) e Conversor/Gravador (Byte => Caracter), conforme o diagrama de blocos da Figura 6.10. O objetivo aqui é capturar, converter e armazenar em um arquivo texto a seqüência de bytes que representa a mensagem, conforme ilustrado na Figura 6.16. A mensagem é composta por um preâmbulo identificador (55H), um byte de status (3H, para esta situação de simulação) e a seqüência de bytes da mensagem extraída do pacote Ethernet truncado pela perda da portadora.

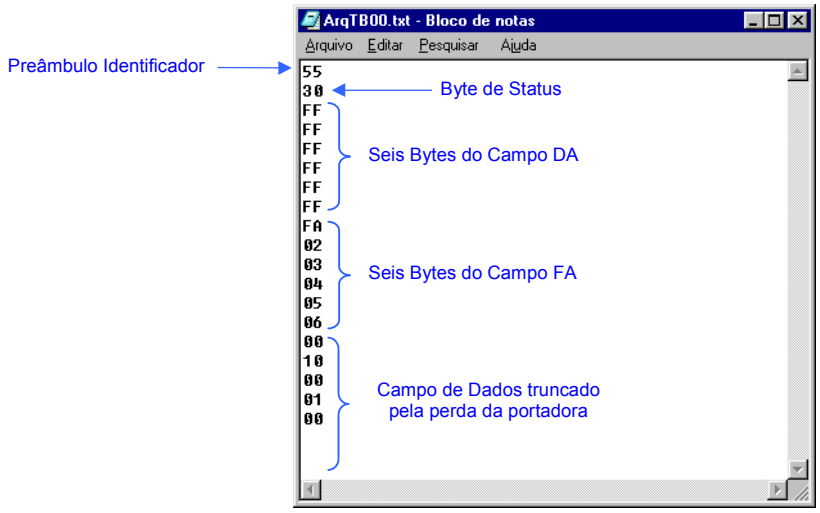


Figura 6.16 – Arquivo texto com a mensagem (truncada) armazenada pelo módulo Gerador/Capturador.

O módulo Gerador/Capturador utiliza um estímulo interno (ilustrado na Figura 6.16) denominado pulso Resk para sincronizar o relógio (Rskg) das operações de captura, conversão e armazenamento, com a taxa de transferência da interface MAC-Usuário (descrição em VHDL no Anexo E, linhas 152 a 167).

6.3 Conclusão

Durante o desenvolvimento e a validação funcional dos módulos MAC-Usuário, TxMAC e RxMAC observa-se a complexidade das tarefas e dos estímulos envolvidos. As tarefas e os estímulos são descritos em duas fases, uma de validação do TxMAC, na transmissão de pacotes Ethernet, e outra de validação do RxMAC, na recepção de pacotes Ethernet.

Na validação do TxMAC foi apresentada uma parte do conjunto de testes realizados, incluindo: simulação de colisão durante a transmissão do pacote para validar o método CSMA/CD; a captura do pacote com dados truncados e o pacote retransmitido após colisão; simulação de dezesseis colisões consecutivas para validar a operação em backoff. Também foi apresentada a simulação da inicialização do módulo MAC-Usuário na fase de validação do RxMAC. Nesta fase, foram descritas as operações de recepção do pacote pelo RxMAC e o respectivo armazenamento realizado pelo módulo Gerador/Capturador, a simulação de recepção do pacote com perda de portadora, a transferência e o armazenamento do byte de status e mensagens truncadas.

Nestas simulações foram utilizados pacotes de tamanho mínimo, especialmente elaborados a título de exemplo. Contudo, durante o desenvolvimento do projeto, os testes para validação foram realizados de forma exaustiva, utilizados pacotes de diferentes tamanhos, desde o tamanho mínimo ao máximo e com diferentes conjuntos de endereços (campos DA e FA) e tamanhos de dados

(campos LE e DF) encapsulados. Todos os pacotes foram planejados conforme a sintaxe do protocolo da camada MAC no padrão IEEE 802.3 (básico).

Também foram gerados estímulos nas mais diferentes situações de transmissão e recepção dos pacotes. Alguns destes estímulos destinaram-se a provocar colisões em diferentes posições dos pacotes, tais como nos campos de endereço, dados, enchimento (stuffing) e CRC. Também foram geradas simulações com estímulos para testar e validar as fases de deferimento (simples ou em duas partes), o tempo mínimo entre pacotes (IPG), os tempos de espera aleatória exponencial truncada (backoff), a detecção de preâmbulo e SFD, a seqüência JAM e a verificação erros de recepção do pacote (comprimento, alinhamento e FCS).

Em resumo, procurou-se aqui descrever de forma muito sucinta o comportamento do ambiente externo interagindo com os módulos sob teste e estes reagindo conforme especificado pelo protocolo da camada MAC. Inúmeras outras simulações parciais foram realizadas na validação de blocos intermediários, incluindo: os buffers de transmissão e recepção, o detector de baud rate, conversores, registradores, geradores de relógios e CRC, delimitadores de campo e todos os controladores envolvidos na execução das operações de transferências das mensagens e transmissão/recepção dos pacotes.

7. Prototipação do IP Soft Core para o MAC Ethernet

Neste Capítulo, mostra-se alguns detalhes do processo de prototipação do IP core MAC Ethernet sobre uma plataforma comercial construída em torno de FPGAs baseados em RAM.

A Figura 7.1 reproduz o ambiente típico de um sistema computacional completo onde se emprega o IP core desenvolvido para fins de referência no presente Capítulo. Ela também serve para descrever a estrutura geral do ambiente onde o IP core foi e vem sendo validado via prototipação. Revisando a apresentação do Capítulo 5, este diagrama de blocos mostra que o controlador de rede projetado é composto de quatro módulos principais: a interface MAC-Usuário, o IP core MAC Ethernet propriamente dito, a interface MAC-PHY e o módulo de acesso ao meio físico (PHY).

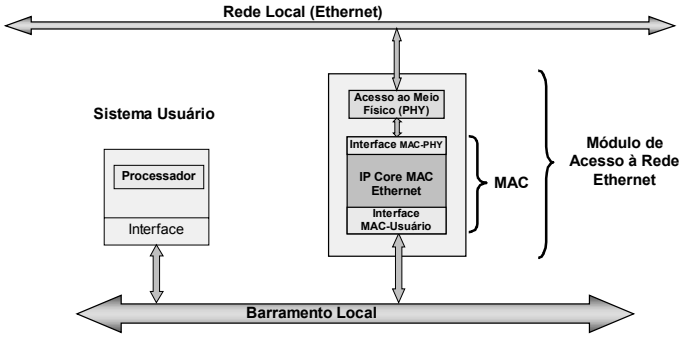


Figura 7.1 – Diagrama de blocos mostrando o ambiente típico de emprego do IP core desenvolvido.

Uma interface MAC-Usuário foi projetada e desenvolvida para permitir a comunicação entre o computador hospedeiro e o IP core MAC implementado no FPGA da plataforma de prototipação. Esta interface é de vital importância para os testes e a validação do IP core MAC. As plataformas de prototipação usadas neste trabalho são apresentadas na Seção 7.1, enquanto que a interface MAC-Usuário implementada é tratada na Seção 7.2. A interface MAC-PHY é um módulo em geral muito simples, uma vez que se trata de uma interface entre módulos definidos ambos no padrão IEEE 802.3. Contudo, ela é necessária para adaptar a interface genérica do core a dispositivos específicos de acesso ao meio físico, pois a norma IEEE 802.3 não padroniza a interface MAC-PHY. Este Capítulo conclui com uma apresentação de propostas de montagens de teste para validar o IP core de forma incremental. Este assunto é tratado na Seção 7.3.

7.1 Plataforma de Prototipação

A plataforma de prototipação que foi usada na maior parte do presente trabalho é a XSV800, o mais poderoso membro da família de plataformas XSV da empresa Xess, apresentada em mais detalhes na Seção 3.4.4. O motivo principal que levou à escolha desta plataforma, além de sua disponibilidade, foi o fato desta contar com um dispositivo de acesso ao meio físico de redes Ethernet, e com um conector RJ-45, usado para a conexão física a redes com meio físico do tipo par trançado, correspondente aos padrões físicos 10BASE-T e 100BASE-T da norma [IEE00]. Estas características facilitaram sobremaneira a prototipação e teste do IP core MAC.

Não foi objetivo deste trabalho a implementação de interfaces de comunicação de alto desempenho entre o IP core MAC Ethernet e o hospedeiro, sobretudo devido ao fato de que o domínio da alta tecnologia e complexidade de barramentos de alto desempenho como o PCI e o USB estar além do escopo deste trabalho. Assumiu-se durante a prototipação que a comunicação entre módulo IP core Ethernet em desenvolvimento e o sistema usuário se daria através da uma linha de comunicação serial RS-232, disponível na plataforma XSV800. Entretanto, o desenvolvimento do IP core MAC Ethernet foi direcionado para permitir que futuramente seja possível substituir a interface MAC-

Usuário corrente por uma outra, que implemente uma via de alto desempenho tal como a interface USB, também disponível na XSV800 ou a interface PCI, disponível, por exemplo na plataforma HOT2-XL, apresentada na Seção 3.4.3.

7.1.1 Limitações de Software e Hardware

Neste ponto é importante considerar que a taxa de transferência de dados, a ser utilizada na interface serial é no máximo da ordem de 115,2 Kbps. Esta taxa, comparada com a velocidade do IP core, padronizada em 10 ou 100Mbps, torna a interface serial o gargalo da comunicação, mesmo se a transmissão ocorrer a 10Mbps. Este gargalo impede a transmissão e recepção de quadros em intervalos de tempo menores que 5 ms. Este tempo é calculado para um pacote de tamanho mínimo (72 bytes, do Preâmbulo até o FCS), ou seja, trata-se ainda de uma análise de melhor caso. Esta situação não representa o tráfego normal de uma rede local, cuja a temporização entre quadros (Inter Packet Gap – IPG) chega a um mínimo de 9,6 μs [CIR99]. Entretanto, para prototipação e teste do IP core MAC, a transmissão de quadros será realizada de forma individual e o tempo entre quadros aumentado para valores adequados ao gargalo da interface serial. Ainda que esta estrutura de teste não permita validar plenamente as características de temporização do IP core, ela servirá para a maior parte das atividades de validação de temporização e para todas as de validação funcional em hardware.

Padrões de vias de alto desempenho tais como o PCI Local Bus [SHA99] e USB são possibilidades mais realistas de implementação da interface MAC-Usuário de NICs Ethernet. O PCI Local Bus, por exemplo, foi concebido para prover uma razoável largura de banda para a transferência de dados entre dispositivos periféricos e processadores conectados ao barramento [SOL98]. Este barramento local de alto desempenho tornou-se um dos padrões de interconexão mais populares na indústria, não só para computadores pessoais, mas também para computadores industriais, chaveadores e roteadores de comunicação e em instrumentação. Um outro padrão, este mais recente é a interface USB (Universal Serial Bus) [COM98], especificada para ser, na indústria, uma extensão padronizada de arquiteturas do tipo PC, com um enfoque na integração do computador com as tecnologias de telecomunicação. A interface USB é uma solução que suporta taxas de transferência de até 12 Mbps, podendo ser utilizada para transferência de quadros Ethernet com taxas de 10Mbps, quando configurada, a interface USB, para o modo de controle de rede Ethernet (Ethernet Networking Control Model) [USB99].

A partir de estudos sobre as especificações dos tempos e velocidades do IP core MAC Ethernet, concluiu-se que um núcleo processador para Ethernet com uma taxa de transferência de 10 Mbps, exige uma plataforma de hardware com uma freqüência de operação (sinal SClk) maior que 10MHz, sendo um valor suficiente estabelecer esta freqüência em 20 MHz. Esta escolha obviamente deve ser revista no caso de se querer trabalhar com taxas de transmissão de 100 Mbps. No desenvolvimento do IP core MAC Ethernet algumas decisões de implementação fazem com que o primeiro protótipo opere apenas a 10Mbps de taxas de transmissão e recepção. Não obstante, necessita-se poucas alterações e uma plataforma de hardware com freqüência de operação compatível para implementar taxas de 100 Mbps. A plataforma XSV é adequada para tanto, exceto, de novo, no que diz respeito à comunicação com um hospedeiro externo.

Uma outra restrição importante imposta pelo projeto é que a comunicação com o IP core se faz através de dois buffers com capacidade para armazenar apenas um quadro de tamanho máximo (1518 bytes**)**, implementada mediante o uso de blocos de memórias RAM de dupla porta, disponibilizadas no interior do dispositivo de hardware reconfigurável (FPGA XCV800 da família Virtex da Xilinx). O esquema de utilização dos buffers foi mantido extremamente simples, pela exclusividade de acesso garantida pelos sinais de handshake TxMsg/TxRdy e RxMsg/RxRdy (ver

Figura 5.3). Devido a esta simplicidade, a reserva do buffer pelo sistema usuário pode gerar a perda de pacotes transmitidos, caso haja demora daquele em recuperar uma mensagem do buffer. Os testes de validação vêm sendo cuidadosamente planejados para evitar esta situação. Na prática, podem ocorrer problemas de espaço de armazenamento, disponibilização e transmissão de mensagens entre o IP core e o sistema usuário. Na entanto, nenhuma destas limitações é parte do IP core. Por exemplo, caso o IP core não deva ser usado em uma arquitetura como a descrita na Figura 7.1, mas como um módulo de um sistema integrado, nenhum dos problemas citados aqui ocorre.

7.2 Módulo MAC-Usuário do Protótipo

O módulo MAC-Usuário, desenvolvido para validar o protótipo do IP core MAC Ethernet é composto de oito blocos principais, conforme representado na Figura 7.2. Os dois blocos mais importantes são os controladores de transmissão e de recepção de mensagens, descritos em diagramas de transição de estados nos Anexos A e B. Estes dois controladores são responsáveis pela transferência de quadros entre o sistema usuário e o IP core, mediante controle dos sinais TXD, RXD, RTS e DTR da interface RS-232. A escolha desta interface deu-se, inicialmente, devido a ser esta a única presente na plataforma de prototipação disponível no início deste trabalho, qual seja, a VW300 da empresa VCC [VIR99], descrita na Seção 3.4.2.

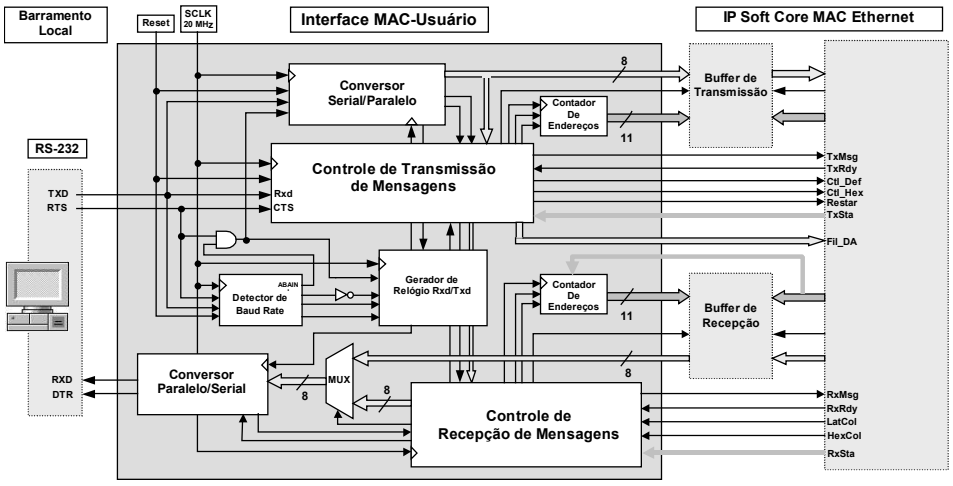


Figura 7.2 – Diagrama de blocos da Interface MAC-Usuário.

Um terceiro bloco, o *detector de baud rate*, descrito em diagrama de transição de estados no Anexo C é utilizado apenas no início do envio de dados do computador pessoal, para detectar automaticamente a taxa de transferência com que os dados serão transmitidos, e adaptar a recepção de forma automática. O quarto e o quinto blocos são registradores de deslocamento, responsáveis pela conversão dos dados da forma serial para paralela e vice versa. Estes realizam a sincronização da transferência dos dados entre o computador pessoal e os buffers de recepção e transmissão, respectivamente. Para isto, os blocos registradores utilizam-se do sexto bloco, o Gerador de Rrelógio Rxd/Txd, previamente ajustado pelo detector de baud rate com a mesma taxa de transferência inicialmente configurada na interface serial do PC. Os dois últimos blocos são os Contadores de Endereços, utilizados pelos controladores de recepção e transmissão para transferir bytes de forma ordenada, de e para os buffers, respectivamente.

Para permitir a comunicação entre o protótipo e um hospedeiro do tipo PC, foi desenvolvido um software (Ether_Serial) de comunicação com o módulo MAC-Usuário. Este software executa algumas funções específicas para dar apoio ao uso e à validação do IP core MAC Ethernet. Entre estas funções pode-se citar:

- Geração de pacotes de teste em geral e pacotes Ethernet, em particular, via um editor gráfico interativo;
- Transmissão e recepção de pacotes Ethernet;
- Configuração do IP core MAC Ethernet.

Um protocolo simples foi implementado para determinar o tipo de dados enviados e/ou recebidos pelo hospedeiro. Conforme exemplificado na Seção 6.1 no início do conjunto de dados denominado de "mensagem" (quadro Ethernet sem os bits de PAD e sem o FCS, Seção 5.1) são acrescentados dois bytes, conforme ilustrado na Figura 7.3 (Parâmetros Para Transmissão). A Tabela 7.1 apresenta o comando (preâmbulo e controle) e as ações tomadas pelo controlador de transmissão e recepção serial RS-232.

Tabela 7.1 – Parâmetros de comando para transmissão e recepção via interface serial.

Preâmbulo	Controle	Significado	Ações
55H	00H	Inicialização do Controlador TxMAC.	Gera o sinal de Restart e Sinaliza código "0".
55H	01H	Dados (mensagem) para transmissão.	Armazena a mensagem no buffer de transmissão e Sinaliza código "3" (caso não ocorra erro).
55H	02H	Configuração do Controlador TxMAC.	O byte subseqüente determina os modos de deferimento (simples ou em duas partes) e a comunicação em Half-duplex ou Full-duplex.
55H	03	Configuração do Controlador RxMAC. (ainda não implementado)	O byte subseqüente determina a configuração do "filtro DA" do controlador RxMAC.
55H	> 03H	Controle inválido.	Sinaliza código "2", controle inválido.
Diferente de 55H	Qualquer código	Comando inválido para controle.	Sinaliza código "0", preâmbulo inválido.

O primeiro byte (55H) representa um preâmbulo, identificador para o bloco controlador de transmissão serial RS-232 do módulo MAC-Usuário. O segundo é interpretado pelo controlador de transmissão ou de recepção (configuração do filtro DA), conforme especificado nas Seções 5.3 e 5.4, respectivamente. Este é um byte de controle para o tipo de operação a ser realizada com os bytes subseqüentes (se existirem).

O software Ether_Serial para realizar a interação entre o hospedeiro (PC) e o protótipo do IP core MAC Ethernet foi desenvolvido em linguagem de alto nível, orientada a objetos. A linguagem escolhida foi o Borland Pascal 7.0 [SWA93].

A Figura 7.3 mostra a interface gráfica do software. O exemplo de tela apresenta as janelas de dados enviados e recebidos do IP core MAC Ethernet, bem como uma janela para introdução de Parâmetros para Transmissão/Recepção na configuração do IP core.

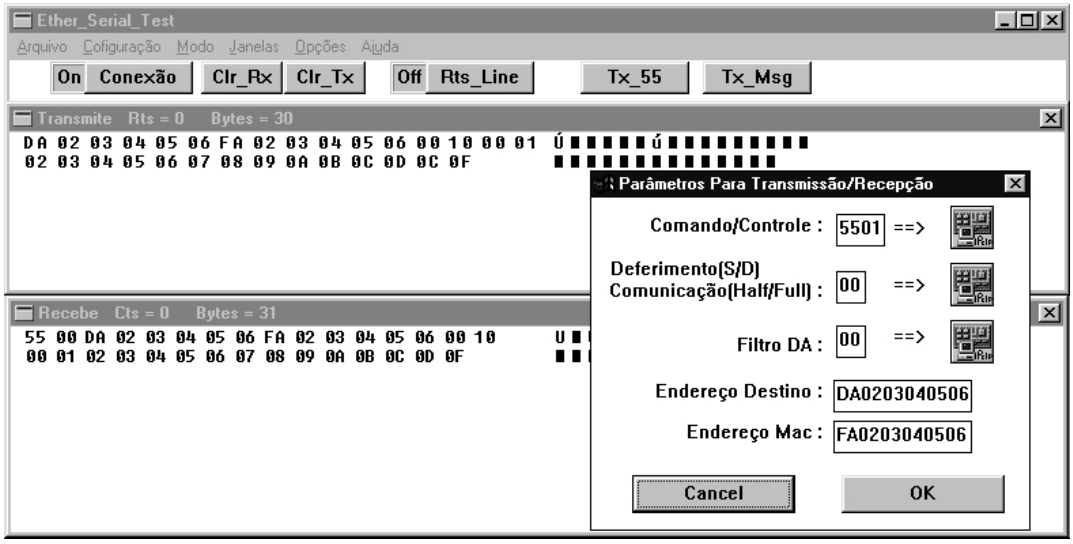


Figura 7.3 – Tela mostrando a execução do software de interação entre o hospedeiro (PC) e o protótipo do IP Core MAC Ethernet.

7.3 Proposta de Montagens para Validação a Nível de Protótipo

Foram elaboradas quatro propostas de montagens a serem usadas na validação do IP core MAC Ethernet a nível de prototipação. Esta Seção detalha a estrutura geral destas montagens.

Primeiro, para implementar e validar o protótipo duplica-se o IP Core MAC Ethernet, criando uma montagem (a). Como ilustrado na Figura 7.4(a), dois IP cores são configurados em duas plataformas idênticas, de forma a simular a comunicação entre dois nós de rede.

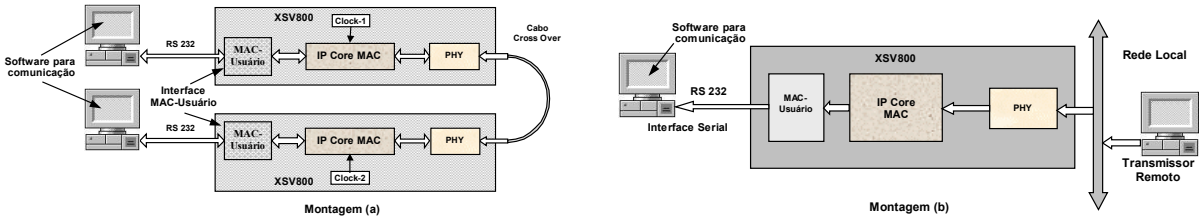


Figura 7.4 – Duas montagens de validação do protótipo do IP Core MAC Ethernet. (a) Valida transmissão e recepção internamente ao projeto. (b) Valida recepção em rede local real com estação remota transmitindo quadros para o protótipo com grande espaçamento entre estes.

Para o teste da recuperação do relógio na etapa de recepção do IP Core, se faz necessária a utilização de dois geradores de relógio independentes (Clock-1 e Clock-2), para evitar que o teste seja viciado pelo uso de sinais de relógio em fase dos dois lados da comunicação, algo que não ocorre na realidade da comunicação entre módulos processadores empregando tecnologia Ethernet. A montagem (a) possibilita validar a transmissão e recepção de quadros Ethernet de forma individual e isolada, evitando-se riscos de interferência ao realizar testes numa rede local real.

A proposta de montagem representada na Figura 7.4(b) consiste em configurar o IP core Ethernet no FPGA, funcionando apenas para recepção, mas conectado a uma rede local real. Nesta configuração, o IP core é programado para a recepção de quadros Ethernet provenientes de uma

rede real. Estes são enviados de um transmissor remoto conectado a uma rede local. Os quadros têm o campo de endereço destino DA previamente selecionado para evitar conflitos com o tráfego da rede local real. Este endereço coincide com o endereço MAC do IP core do protótipo. O tempo entre os quadros deve, neste caso, ser mantido em valores adequados à taxa de transferência da interface serial na recepção. Com a montagem (b) valida-se o módulo receptor do IP core no que concerne sobretudo a recuperação de sinal de relógio, o Controle de Recepção e o verificador de CRC.

A proposta de montagem (c), representada na Figura 7.5(c), implementa o IP core em uma configuração completa, possibilitando transmitir e receber quadros padronizados simultaneamente em uma rede local real, mantidos os mesmos cuidados com o endereçamento e o tempo entre os quadros da montagem anterior.

Como mencionado na Seção anterior, pretende-se futuramente utilizar uma nova plataforma de prototipação, que possibilite o uso uma via de alto desempenho (PCI) *de facto*, conforme ilustrado na Figura 7.5(d). Na montagem (d) pretende-se utilizar um IP core PCI, seja o disponível na plataforma HOT2–XL, seja o atualmente em desenvolvimento no trabalho correlato [CAP01].

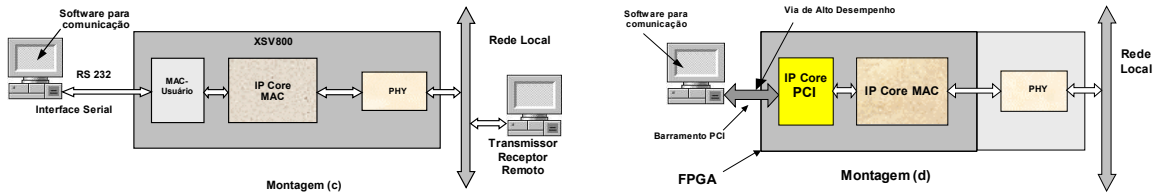


Figura 7.5 - Duas outras montagens de validação do protótipo do IP core MAC Ethernet. (c) Valida transmissão e recepção em rede local real com estação remota transmitindo quadros para o protótipo com grande espaçamento entre estes. (d) Plataforma com IP core Ethernet combinado com IP core PCI para implementar via de alto desempenho.

Com as quatro propostas de montagens propõe-se validar o desempenho da implementação, incluindo ambos, software e hardware. Também é possível utilizar as montagens para avaliar os módulos de software e hardware do IP core desenvolvido em um ambiente realista e sem interferir com o tráfego de quadros Ethernet da rede local em uso.

7.4 Resultados Preliminares da Prototipação

Para fins de avaliação, somente a síntese de uma versão completa do controlador foi realizada. A interface MAC-Usuário usada foi a descrita na Seção 7.2 e os módulos de Transmissão (TxMAC) e Recepção (RxMAC) do projeto do IP Core, utilizados conforme especificado no Capítulo 6. O conjunto de módulos (denominado UserCore), o bloco da direita ilustrado na Figura 7.1, foi implementado em um FPGA Xilinx da família Virtex (XCV800).

A versão completa do controlador, MAC-Usuário, TxMAC e RxMAC, soma aproximadamente 4000 linhas de VHDL. Na síntese desta versão não estão incluídas a linhas de Test Bench, visto que estas são necessárias somente na fase de validação funcional do projeto (Capítulo 6). O protótipo ocupou área de 7% do FPGA XCV800 cuja capacidade total é de aproximadamente 800.000 portas lógicas, desta forma, alcançou a contagem equivalente à 110.300 portas com buffers e aproximadamente 60.000 portas sem incluir os buffers. Dois buffers, de Transmissão e Recepção, totalizam uma capacidade de 24 Kbits e utilizaram 6 módulos de BlockRAM dos 28 disponíveis.

Registradores de configuração estão ausentes devido à configuração do FPGA ser realizada de forma manual.

Os resultados preliminares de dimensionamento, gerados pela ferramenta de síntese aparecem com mais detalhes na Figura 7.6.

```
Xilinx Mapping Report File for Design 'UserCore'
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line   : map -p xcv800-4-hq240 -o map.ncd pether01.ngd pether01.pcf
Target Device  : xv800
Target Package : hq240
Target Speed   : -4
Mapper Version : virtex -- D.26
Mapped Date    : Fri Jul 20 10:48:51 2001

Design Summary
--------------
   Number of errors:      0
   Number of warnings:   67
   Number of Slices:                695 out of  9,408    7%
   Number of Slices containing
      unrelated logic:                0 out of    695    0%
   Total Number Slice Registers:    637 out of 18,816    3%
      Number used as Flip Flops:                585
      Number used as Latches:                    52
   Total Number 4 input LUTs:       923 out of 18,816    4%
      Number used as LUTs:                      918
      Number used as a route-thru:                5
   Number of bonded IOBs:            15 out of    166    9%
   Number of Block RAMs:              6 out of     28   21%
   Number of GCLKs:                   4 out of      4  100%
   Number of GCLKIOBs:                3 out of      4   75%
Total equivalent gate count for design:  110,300
Additional JTAG gate count for IOBs:  864
```

Figura 7.6 – Relatório de mapeamento do IP Core (UserCore).

Do ponto de vista da validação do protótipo em si, uma primeira montagem do tipo (b) foi usada e permitiu validar completamente o módulo denominado Interface MAC-Usuário e a geração/verificação de CRC a partir de pacotes reais capturados da rede pela plataforma XSV disponibilizados pelo CI Ethernet PHY. A verificação usou um analisador lógico conectado entre o FPGA e o CI Ethernet PHY e o software Ether-Serial.

Está em andamento agora o uso da montagem (a) para validar mais amplamente o IP Core MAC Ethernet.

8. Conclusões e Trabalhos Futuros

Os objetivos estabelecidos ao início deste trabalho são reproduzidos aqui para facilitar a discussão. O objetivo estratégico foi dominar a tecnologia de implementação de controladores de redes Ethernet, concentrando-se nos níveis inferiores de abstração. Os objetivos específicos aparecem abaixo listados:

1. Projetar e validar a parte fundamental de um controlador de rede local para o padrão Ethernet, endereçando procedimentos de software e hardware no nível de enlace do modelo OSI-RM;
2. Implementar um controlador baseado no módulo mencionado no item anterior sobre componentes reconfiguráveis em uma plataforma de prototipação escolhida, fazendo-a interagir com um computador hospedeiro e com uma rede local;
3. Disponibilizar o módulo resultante sob forma de um núcleo processador de propriedade intelectual do tipo Soft (IP soft core) reutilizável;
4. Prover subsídios para efetuar migrações entre software e hardware dos módulos em questão, com o intuito de maximizar desempenho e/ou minimizar custos dos sistemas digitais e respectivo software.

Considera-se que o objetivo estratégico foi plenamente atendido. A tecnologia de controladores de rede Ethernet foi dominada, estando o grupo envolvido capacitado a proceder no futuro com trabalhos adicionais nesta área de conhecimento. A partir desta tecnologia, pode-se pretender abordar temas mais atuais como a implementação de controladores para redes sem fio (em inglês wireless networks) tais como o padrão 802.11 [IEE99] ou o padrão industrial Bluetooth, além de abordar outros temas da vasta área em expansão que é a transmissão de dados em alta velocidade.

O domínio da tecnologia de projeto e prototipação de tais circuitos sobre dispositivos reconfiguráveis é um subproduto importante deste trabalho. Os módulos críticos em desempenho foram identificados na tecnologia particular e estas informações servem hoje de subsídio para outros trabalhos em andamento.

Considera-se atendido o primeiro objetivo específico para os aspectos de hardware. Os procedimentos de software foram desenvolvidos para validar o protótipo de forma *ad hoc*, com rotinas básicas para envio e recepção de pacotes Ethernet de um computador hospedeiro para o IP core MAC Ethernet, através da interface MAC-Usuário, desenvolvida e descrita no Capítulo anterior. O desenvolvimento de drivers de dispositivo para um controlador usando o IP core MAC Ethernet descrito aqui está fora do escopo do presente trabalho. Ele é visto contudo, como um importante trabalho futuro, baseado na experiência adquirida na construção e validação do protótipo. Deve estar claro que estes drivers fazem sentido no momento em que exista uma interface MAC-Usuário que empregue um meio de comunicação mais realista com o controlador, tipicamente um barramento local PCI, ou equivalente.

O segundo objetivo foi atendido parcialmente. O ambiente de teste foi estabelecido e vários módulos de apoio foram desenvolvidos. O teste em hardware encontra-se em andamento para cada um dos módulos, transmissor e receptor, de forma separada. Testes após a integração destes em enlaces locais e sobre redes reais já estão sendo usado as montagens propostas no Capítulo anterior.

O terceiro objetivo também foi atendido de forma parcial, mas encontra-se muito próximo de seu atendimento total. Encontram-se validados funcionalmente os módulos TxMAC e RxMAC que compõem o IP core MAC Ethernet, incluindo os buffers de transmissão e recepção, a partir do uso de blocos especiais de memória RAM no dispositivo reconfigurável escolhido. Aliás, estes buffers são vistos como a única fonte possível de problemas de portabilidade. Para reduzir o problema, duas versões do IP core são mantidas de forma paralela, uma totalmente independente da tecnologia de implementação destes buffers, usando a construção *array* em VHDL e outra, específica para o protótipo desenvolvido, otimizada para a família de dispositivos usada, Virtex da Xilinx. O projeto funcional está completo, gerando quadros básicos para transmissão no padrão IEEE 802.3. O módulo de transmissão e a interface MAC-Usuário encontram-se validados. A interface MAC-PHY é praticamente inexistente para o caso da prototipação, dado o tipo de dispositivo de acesso ao meio físico disponível na plataforma.

O quarto objetivo é considerado atendido, baseado na experiência de projeto adquirida. Contudo, falta validar na prática as idéias desenvolvidas, pela adaptação do IP core desenvolvido a um ambiente prático, com restrições de custo e desempenho que justifiquem a migração de módulos entre hardware e software. Por exemplo, suponha-se a possibilidade de automação caseira de equipamentos, um assunto de crescente interesse comercial, justificando uma área de desenvolvimento conhecida como domótica [MOR01]. *Domótica* é uma área de aplicações voltadas para monitoração e controle de residências. Uma das necessidades fundamentais da domótica é a colocação em rede de utensílios caseiros, tais como máquinas de lavar, sistemas de alarme, banheiros, etc. Caso se queira usar o IP core Ethernet em questão para interligar cada artefato da casa a ser monitorado/controlado, existe a necessidade de reduzir drasticamente o custo do controlador de rede. Este poderia então ser parte de um SORC de baixa densidade (definido na Seção 2.6), onde na parte reconfigurável poderia ser implementado o IP core MAC Ethernet e na parte programável serem implementados os protocolos de nível mais elevado. Além disto, como estes dispositivos ocupam pouca área de circuito um reconfigurável, talvez seja necessário neste caso migrar parte da funcionalidade do hardware para software, dada a pouca necessidade de desempenho, e o forte acoplamento entre o sistema usuário e o IP core, pois trata-se de um barramento interno de um circuito integrado.

Uma das preocupações atuais vem sendo a homologação do IP core desenvolvido, ou seja uma certificação prática que o mesmo se conforma à especificação IEEE 802.3. Para tanto, é necessário o uso de equipamentos de teste indisponíveis e de custo extremamente elevado. Para tanto, se está lançando mão da cooperação do grupo onde o trabalho está sendo desenvolvido com empresa de comunicação de dados que possui tais equipamentos.

Finalmente, divisa-se uma série de desenvolvimentos futuros a curto e médio prazo para refinar e dar seqüência ao trabalho aqui descrito. Entre estes, citam-se:

1. Finalização do protótipo do IP core, incluindo sua homologação funcionando a 10Mbps, passando mais tarde a fazê-lo funcionar a 100Mbps;

2. Acrescentar a possibilidade de parametrização de reconfigurabilidade do IP core para permitir sintetizar versões específicas, onde se escolhe a velocidade de comunicação (10/100Mbps) a forma de interação com o dispositivo de acesso ao meio físico (série/paralelo), etc.;

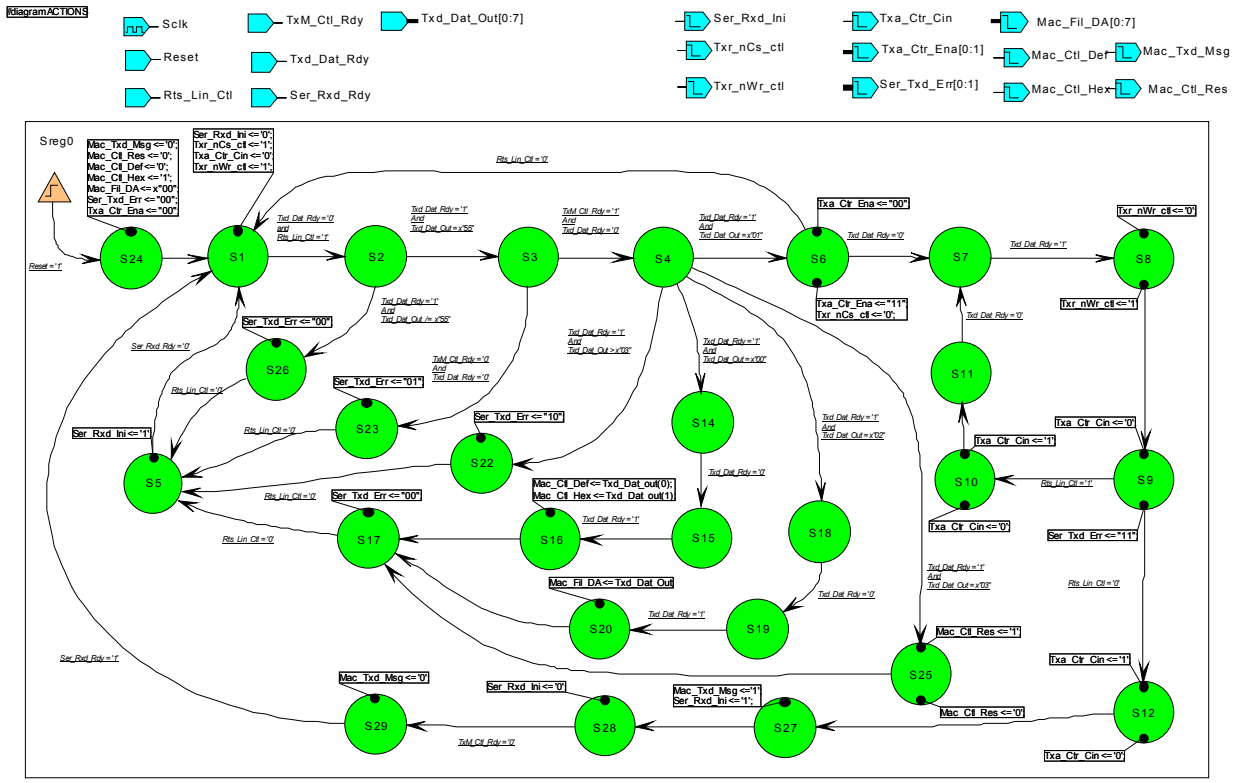
3. Integração a outros IP cores, marcadamente os que provêem comunicação com um hospedeiro via um barramento de alto desempenho, do tipo PCI, USB, etc.;

4. Flexibilização do acesso aos buffers para permitir escrita e leitura simultâneas, aumentando o desempenho da comunicação MAC-Usuário;

5. Incluir algum suporte no IP core desenvolvido para transferência de um bom volume de dados (mensagens grandes, obtidas, por exemplo, durante a transferência de arquivos) em alta velocidade para o hospedeiro. Isto inclui recursos para uso de interrupções e/ou acesso direto à memória (DMA).

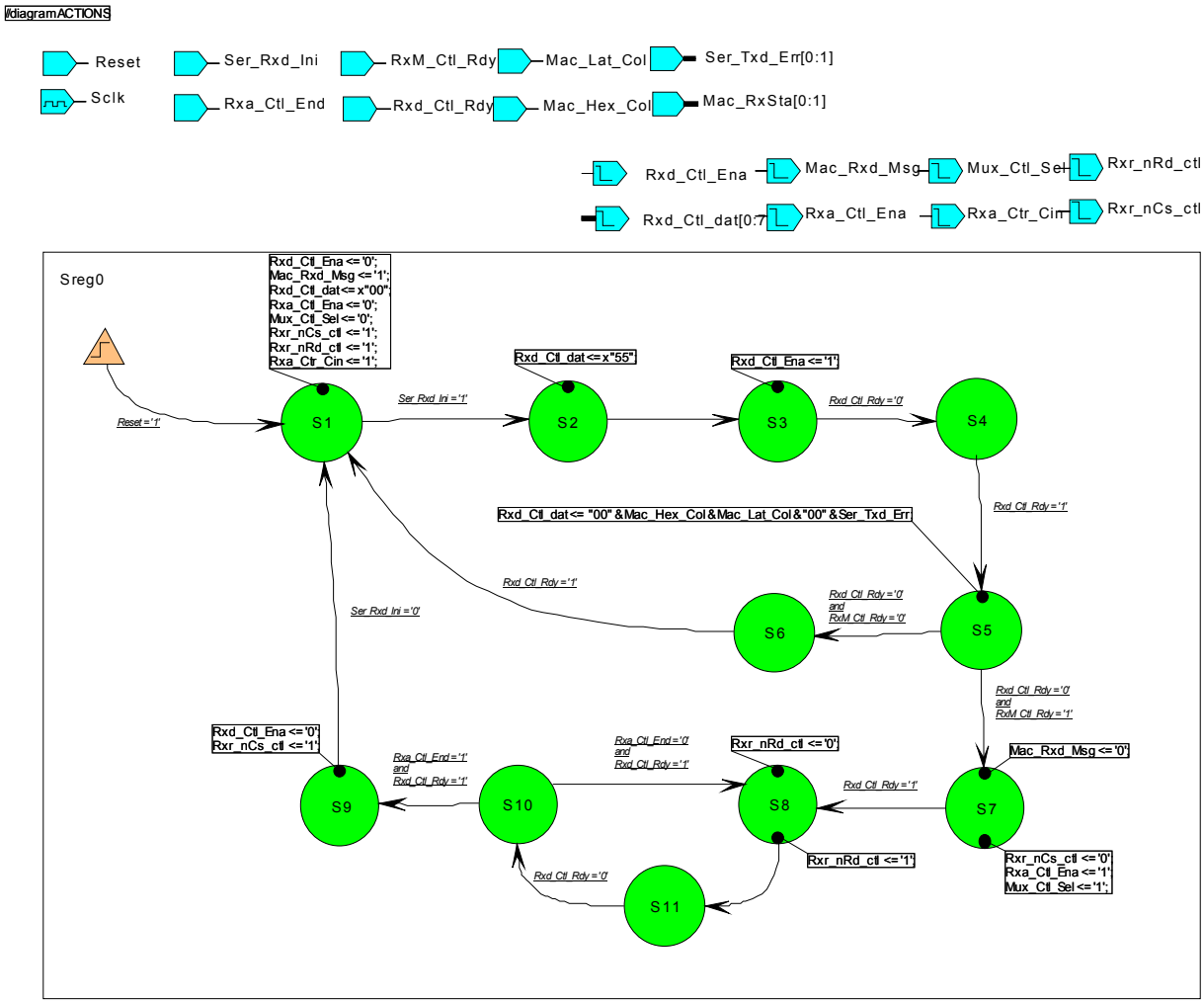
ANEXO A

Bloco controlador de transmissão serial RS-232



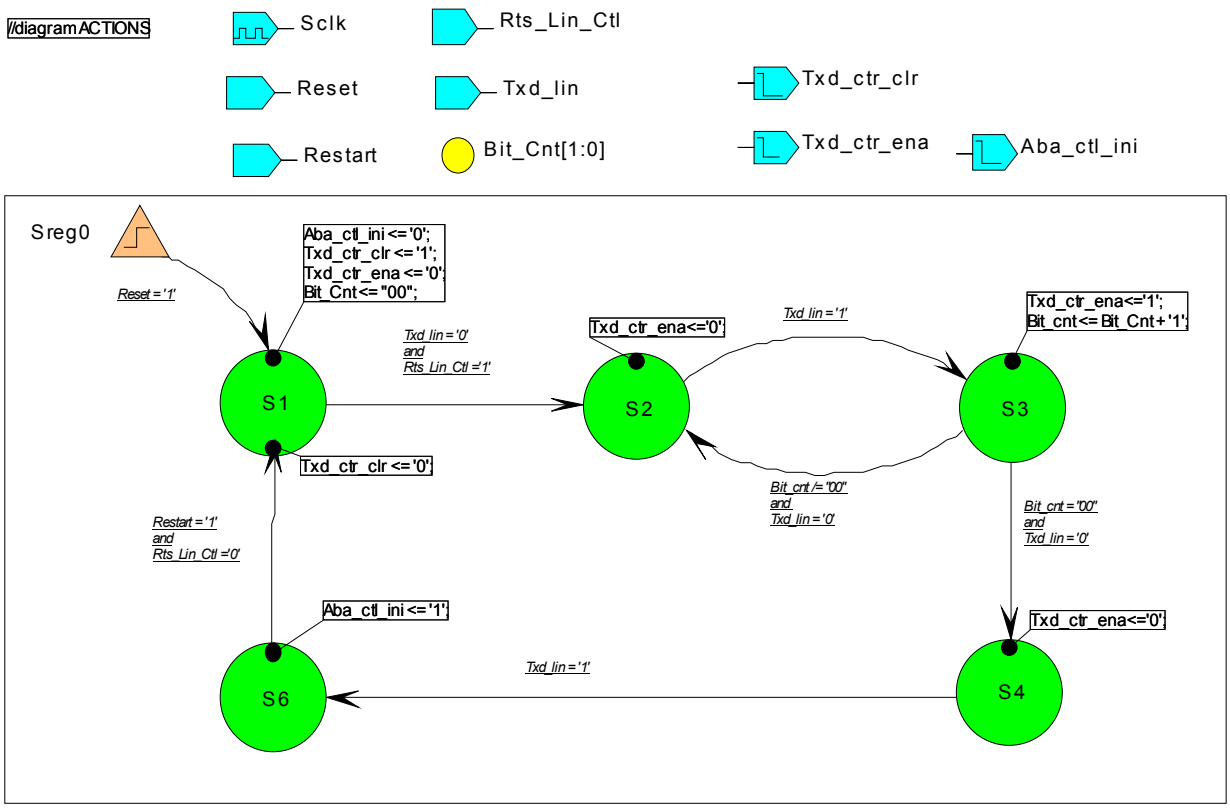
ANEXO B

Bloco controlador de recepção serial RS-232



ANEXO C

Bloco detector de baud rate



ANEXO D

Módulo Gerador/Capturador para o Testbench-TX

```
1   --     TestBench-TX
2   --  File: D:\CoreEther\Ether_Serial_0\SRC\MAC_User_TB.VHD
3   --
4   --
5   library IEEE;
6   use IEEE.Std_Logic_1164.all;
7   use IEEE.std_logic_unsigned.all;
8   use IEEE.Std_Logic_arith.all;
9   use STD.TEXTIO.all;
10
11  --{entity {MAC_User_TB} architecture {MAC_User_TB}}
12
13  entity MAC_User_TB is
14  end MAC_User_TB;
15
16  architecture MAC_User_TB of MAC_User_TB is
17
18  component UserCore is
19  port
20        (
21              -- EXTERNAL
22              Reset: in std_logic;
23              SCLK: in std_logic;
24              -- RS232
25              TXD: in std_logic;
26              RTS: in std_logic;
27              RXD: out std_logic;
28              DTR: out std_logic;
29              -- MAC-PHY
30              TxClk: in std_logic;
31              Txds: out std_logic;
32              Txdn: out std_logic_vector(3 downto 0);
33              Tx_en: out std_logic;
34              Tx_err: out std_logic;
35
36              RxClk: in std_logic;
37              Rxds: in std_logic;
38              Rxdn: in std_logic_vector(3 downto 0);
39              Rx_err: in std_logic;
40
41              Col: in std_logic;
42              Crs: in std_logic
43        );
44  end component;
45  ------------------------------------------------------------------
```

```
46  -- Test Bench
47  file INFILE                         : TEXT open READ_MODE is "ArqTB00.txt";
48  file OUTFILE                        : TEXT open WRITE_MODE is "ArqTB10.txt";
49
50  signal Rtslin                   :STD_LOGIC := '0';
51  signal Txdlin                   :STD_LOGIC := '1';
52  signal ln1                      :STD_LOGIC := '1';
53  signal Rxdout,Dtrout            :STD_LOGIC;
54  signal Reset,Strb8,Strb12       :STD_LOGIC;
55  signal Sckg,Tskg,Tckg           :STD_LOGIC;
56  signal srq,TxrDin               :STD_LOGIC_VECTOR (7 downto 0);
57  signal bitctr                   :std_logic_vector (3 downto 0);
58  signal SckgCtr                  :integer;
59
60  -- Baud rate Generator
61  constant T: time := 8680 ns;   --  =>115200 bps
62  --104164 ns   =>  9600 bps
63  -- 52082 ns   => 19200 bps
64  -- 26041 ns   => 38400 bps
65  -- 13020 ns   => 76800 bps
66  --  8680 ns    =>115200 bps
67
68  -- MAC-PHY
69  signal txds         :STD_LOGIC;
70  signal txdn         :std_logic_vector(3 downto 0);
71  signal txen         :STD_LOGIC;
72  signal txerr        :STD_LOGIC;
73
74  signal rxds         :STD_LOGIC;
75  signal rxdn         :std_logic_vector(3 downto 0);
76  signal rxerr        :STD_LOGIC;
77  signal col,crs      :STD_LOGIC;
78  ----------------------------------------------------------------------
79  ----------------------------------------------------------------------
80  begin
81
82  usercor: UserCore
83  port map(
84              -- EXTERNAL
85              Reset     => Reset,
86              SCLK      => Sckg,
87              -- RS232
88              TXD       => Txdlin,
89              RTS       => Rtslin,
90              RXD       => Rxdout,
91              DTR           => Dtrout,
92              -- MAC-PHY
93              TxClk     => Tckg,
94              Txds          => txds,
95              Txdn      => txdn, --: out std_logic_vector(3 downto 0);
96              Tx_en     => txen,
```

```
 97              Tx_err    => txerr,
 98
 99              RxClk     => Tckg,
100              Rxds      => rxds,
101              Rxdn      => rxdn, --: in std_logic_vector(3 downto 0);
102              Rx_err    => rxerr,
103
104              Col       => col,
105              Crs       => crs
106         );
107
108   rxds <= txds;
109   Reset <='1', '0' after 50 ns;
110
111   process -- 20 Mhz Clock generator
112         begin
113              Sckg <= '1' after 25 ns, '0' after 50 ns;
114              wait for 50 ns;
115      end process;
116
117   process  --- 10 Mhz Clock generator
118         begin
119              Tckg<='1' after 50ns, '0' after 100ns;
120              wait for 100ns;
121      end process;
122
123   process   -- Clock Serial  T: time := 8680 ns  =>115200 bps
124          begin
125               Tskg <= '0', '1' after T/2, '0' after T;
126               wait for T;
127       end process;
128
129          process   -- Strobe to byte from File "ArqTB00.txt"
130          begin
131               Strb8 <= '0', '1' after T*5, '0' after T*5.2;
132               wait for T*10;
133       end process;
134
135          process   -- Strob to nible 1/2 from File "ArqTB10.txt"
136          begin
137               Strb12 <= '0', '1' after 200ns, '0' after 201ns;
138               wait for 400ns;
139       end process;
140
141   process(Sckg, Reset, txen)  -- SckgCtr for elapsed time to sinal col, crs end ....
142    begin
143      if Reset = '1' then
144               SckgCtr <= 0;
145               col <= '0';
146               crs <= '0';
147          elsif txen = '1' then
```

```
148         sckgctr <= sckgctr + 1;
149           case sckgctr is
150           when 400 => col <= '1';
151         -- when 530 => col <= '0';
152         -- when 500 => crs <= '1';
153             when others => null;
154           end case;
155         end if;
156     end process;
157
158     process(Strb8)                          --- Load Message from File "ArqTB00.txt"
159
160         variable IN_LINE : LINE;   -- pointer to string
161         variable linha : string(1 to 2);
162         variable stg1,stg2 : integer;
163
164     begin
165         if Reset = '0' then
166         if Strb8'event and Strb8='1' and NOT(endfile(INFILE))then
167                 readline(INFILE,IN_LINE);           -- read line from a file
168           read(IN_LINE, linha);
169                 Rtslin <= '1';
170                case linha(1) is                -- convert character to byte
171             when '0' => stg1 := 0;
172             when '1' => stg1 := 1;
173             when '2' => stg1 := 2;
174             when '3' => stg1 := 3;
175             when '4' => stg1 := 4;
176             when '5' => stg1 := 5;
177             when '6' => stg1 := 6;
178             when '7' => stg1 := 7;
179             when '8' => stg1 := 8;
180             when '9' => stg1 := 9;
181             when 'A' => stg1 :=10;
182             when 'B' => stg1 :=11;
183             when 'C' => stg1 :=12;
184             when 'D' => stg1 :=13;
185             when 'E' => stg1 :=14;
186             when 'F' => stg1 :=15;
187             when others => null;
188                 end case;
189                 case linha(2) is
190             when '0' => stg2 := 0;
191             when '1' => stg2 := 1;
192             when '2' => stg2 := 2;
193             when '3' => stg2 := 3;
194             when '4' => stg2 := 4;
195             when '5' => stg2 := 5;
196             when '6' => stg2 := 6;
197             when '7' => stg2 := 7;
198             when '8' => stg2 := 8;
```

```
199             when '9' => stg2 := 9;
200             when 'A' => stg2 :=10;
201             when 'B' => stg2 :=11;
202             when 'C' => stg2 :=12;
203             when 'D' => stg2 :=13;
204             when 'E' => stg2 :=14;
205             when 'F' => stg2 :=15;
206             when others => null;
207                 end case;
208                 TxrDin <= conv_std_logic_vector(stg1*16+stg2,8);
209               end if;
210            if endfile(INFILE) then
211                 Rtslin <= '0';
212            end if;
213          end if;
214    end process;
215
216    process(Tskg, Strb8, Reset, Rtslin)            -- SHIFTREG Module
217
218          begin
219             if (Tskg'event and Tskg='1'and Rtslin = '1') then
220                     bitctr <= bitctr + '1';
221             end if;
222                if Strb8'event and Strb8 ='0' then
223                srq<=TxrDin;                              -- Load q register with data [7..0]
224                     bitctr <= "0000";
225                end if;
226                if Reset ='1' then
227                     srq <= (others => '1');
228                     bitctr <= "0000";
229             end if;
230
231                case bitctr is                          --  Transfer the left bit from q register
232                  when "0001" => Txdlin <= '0';         -- Start bit
233                   when "0010" => Txdlin <= srq(0);
234                   when "0011" => Txdlin <= srq(1);
235                   when "0100" => Txdlin <= srq(2);
236                   when "0101" => Txdlin <= srq(3);
237                   when "0110" => Txdlin <= srq(4);
238                   when "0111" => Txdlin <= srq(5);
239                   when "1000" => Txdlin <= srq(6);
240                   when "1001" => Txdlin <= srq(7);
241                   when "1010" => Txdlin <= '1';              -- Stop bit
242                   when others => null;
243                end case;
244    end process;
245
246    process(txen, Strb12)                                 --- Write Message to File "ArqTB00.txt"
247            variable OUT_LINE   : LINE;                   -- pointer to string
248           variable outln                  : string(1 to 2);
249            variable chr                : Character;
```

```
250          variable int              : integer;
251
252     begin
253          if txen = '1' then
254               if Strb12'event and Strb12='1'then
255                 int := CONV_INTEGER(Txdn);
256               case int is
257              when 0 => chr :='0';
258              when 1 => chr := '1';
259              when 2 => chr := '2';
260              when 3 => chr := '3';
261              when 4 => chr := '4';
262              when 5 => chr := '5';
263              when 6 => chr := '6';
264              when 7 => chr := '7';
265              when 8 => chr := '8';
266              when 9 => chr := '9';
267              when 10 => chr := 'A';
268              when 11 => chr := 'B';
269              when 12 => chr := 'C';
270              when 13 => chr := 'D';
271              when 14 => chr := 'E';
272              when 15 => chr := 'F';
273              when others => null;
274                  end case;
275
276                  if ln1 = '0' then
277                       outln(1):= chr;
278                       write( OUT_LINE, outln);
279                    writeline(OUTFILE, OUT_LINE); -- write line to a file
280                  else
281                       outln(2):= chr;
282                  end if;
283                  ln1 <= not ln1;
284          end if;
285          end if;
286          if txen'event and txen='0'then
287               if ln1 = '0' then
288                ln1 <= '1';
289                outln(2):= chr;
290                write( OUT_LINE, outln);
291                writeline(OUTFILE, OUT_LINE); -- write line to a file
292               end if;
293          --     FILE_CLOSE(OUTFILE); -- Error on colision
294          end if;
295     end process;
296
297     end MAC_User_TB;
```

ANEXO E

Módulo Gerador/Capturador para o Testbench-RX

```
 1
 2   --
 3   --  File: D:\CoreEther\Ether_Serial_0\SRC\MAC_User_TB.VHD
 4   --
 5   --
 6   library IEEE;
 7   use IEEE.Std_Logic_1164.all;
 8   use IEEE.std_logic_unsigned.all;
 9   use IEEE.Std_Logic_arith.all;
10   use STD.TEXTIO.all;
11
12   --{entity {MAC_User_TB} architecture {MAC_User_TB}}
13
14   entity MAC_User_TB is
15   end MAC_User_TB;
16
17   architecture MAC_User_TB of MAC_User_TB is
18
19   component UserCore is
20   port
21        (
22             -- EXTERNAL
23             Reset: in std_logic;
24             SCLK: in std_logic;
25             -- RS232
26             TXD: in std_logic;
27             RTS: in std_logic;
28             RXD: out std_logic;
29             DTR: out std_logic;
30             -- MAC-PHY
31             TxClk: in std_logic;
32             Txds: out std_logic;
33             Txdn: out std_logic_vector(3 downto 0);
34             Tx_en: out std_logic;
35             Tx_err: out std_logic;
36
37             RxClk: in std_logic;
38             Rxds: in std_logic;
39             Rxdn: in std_logic_vector(3 downto 0);
40             Rx_err: in std_logic;
41
42             Col: in std_logic;
43             Crs: in std_logic
44        );
```

```
45  end component;
46
47  ----------------------------------------------------------------
48
49  -- Test Bench
50  file INFILE                     : TEXT open READ_MODE is "ArqTB10.txt";
51  file OUTFILE                        : TEXT open WRITE_MODE is "ArqTB00.txt";
52
53  signal Rtslin                   :STD_LOGIC := '0';
54  signal Txdlin,rxena,rsena       :STD_LOGIC;
55  signal Resk                         :STD_LOGIC := '0';
56  signal Rxdout,Dtrout,ini                :STD_LOGIC;
57  signal Reset,Strb8                  :STD_LOGIC;
58  signal Sckg,Tskg,Tckg,Rckg,Rskg         :STD_LOGIC;
59  signal bctrtx,bctrrx                    :std_logic_vector (3 downto 0);
60  signal srq,ssq,RxrDin,RxrDou        :STD_LOGIC_VECTOR (7 downto 0);
61  signal SckgCtr,TskgCtr,bytctr           :integer;
62  signal Start                        :STD_LOGIC := '0';
63
64  -- Baud and RxClk Generators
65  constant T                  : time := 8680 ns; --  =>115200 bps     8680
66  constant R                  : time := 100 ns; --  =>10 Mbps
67
68  --104164 ns  =>  9600 bps
69  -- 52082 ns  => 19200 bps
70  -- 26041 ns  => 38400 bps
71  -- 13020 ns  => 76800 bps
72  --  8680 ns  =>115200 bps
73
74  -- MAC-PHY
75
76  signal txds             :STD_LOGIC;
77  signal txdn             :std_logic_vector(3 downto 0);
78  signal txen             :STD_LOGIC;
79  signal txerr            :STD_LOGIC;
80
81
82  signal rxds             :STD_LOGIC;
83  signal rxdn             :std_logic_vector(3 downto 0);
84  signal rxerr            :STD_LOGIC;
85  signal col,crs          :STD_LOGIC;
86
87  ----------------------------------------------------------------
88  ----------------------------------------------------------------
89  begin
90
91  usercor: UserCore
92  port map(
93              -- EXTERNAL
94              Reset     => Reset,
95              SCLK    => Sckg,
```

```
 96              -- RS232
 97              TXD     => Txdlin,
 98              RTS     => Rtslin,
 99              RXD     => Rxdout,
100              DTR             => Dtrout,
101              -- MAC-PHY
102              TxClk   => Tckg,
103              Txds            => txds,
104              Txdn     => txdn, --: out std_logic_vector(3 downto 0);
105              Tx_en   => txen,
106              Tx_err  => txerr,
107
108              RxClk   => Rckg,
109              Rxds    => rxds,
110              Rxdn    => rxdn, --: in std_logic_vector(3 downto 0);
111              Rx_err  => rxerr,
112
113              Col     => col,
114              Crs     => crs
115          );
116
117  txen <= '0';
118  rxerr <= '0';
119
120  Reset <='1', '0' after 50 ns;
121
122  process --- 20 Mhz Clock generator
123        begin
124              Sckg <= '1' after 25 ns, '0' after 50 ns;
125              wait for 50 ns;
126     end process;
127
128  process  --- 10 Mhz Clock generator
129        begin
130              Tckg<='1' after 50ns, '0' after 100ns;
131              wait for 100ns;
132     end process;
133
134  process   -- 10 MHz Clock
135         begin
136               Rckg <= '0', '1' after R/2, '0' after R;
137               wait for R;
138  end process;
139
140  process                              -- Strob8 from  byte to serial
141         begin
142               Strb8 <= '0', '1' after R*3.9, '0' after R*4;
143               wait for R*8;
144  end process;
145
146  process                              -- Clock Serial  T: time := 8680 ns  =>115200 bps
```

```
147          begin
148                Tskg <= '0', '1' after T/2, '0' after T;
149                wait for T;
150        end process;
151
152    process(Sckg, Resk, Reset)              -- Clock Serial  T: time := 8680 ns  =>115200 bps
153
154          variable T_Ctr : time;
155
156    begin
157           if rsena = '0' or  Resk = '1'Then
158             Rskg <= '0';
159             T_Ctr := 0 ns;
160        elsif Sckg'event and Sckg='1' then
161           T_Ctr := T_Ctr + 100 ns;
162           if T_Ctr > T then
163                Rskg <= not Rskg;
164                T_Ctr := 0 ns;
165             end if;
166           end if;
167    end process;
168
169    process(Sckg,Tskg,Reset)                -- SckgCtr for elapsed time to sinal crs, ini, Rtslin ....
170     begin
171       if Reset = '1' then
172                SckgCtr <= 0;
173                TskgCtr <= 0;
174                Rtslin <= '0';
175                rxena <='0';
176                rsena <='0';
177                crs <= '0';
178                ini <= '0';
179           else
180             if Rxdout = '0' and rsena ='0' Then
181                  rsena <='1';
182             end if;
183             if Tskg'event and Tskg='1' then
184                tskgctr <= tskgctr + 1;
185                case tskgctr is
186                       when 1 => Rtslin <= '1';
187                      when 22 => rxena <= '1';
188                      when 44 => Rtslin <= '0';
189                      when others => null;
190                end case;
191             end if;
192              if Sckg'event and Sckg='1' then
193                if rxena ='1' then
194                sckgctr <= sckgctr + 1;
195                      case sckgctr is
196                            when 5 => ini <= '1';
197                            when 6 => crs <= '1';
```

```
198                          when 1200 => crs <= '0';     -- when 400 => crs <= '1';  -- for CRS fault
199                         when others => null;
200                    end case;
201                 end if;
202                end if;
203          end if;
204    end process;
205
206    process(Tskg, Reset, Rtslin)              -- SHIFTREG-TX Module for serial initialization
207
208    begin
209          if Reset ='1' then
210                ssq <= x"55";                 -- For Baud rate detector
211                bytctr <= 0;
212                bctrtx <= "0000";
213                Txdlin <= '1';
214       elsif Rtslin = '1' and  bytctr < 4 then
215           if Tskg'event and Tskg='1' then
216             bctrtx <= bctrtx + '1';
217        end if;
218           if bctrtx = "1010" then
219                bctrtx <= "0000";
220              bytctr <= bytctr + 1;
221                case bytctr is
222                 when 0 => ssq <= x"55";          -- For identifier preamble
223                 when 1 => ssq <= x"02";          -- byte command
224                 when 2 => ssq <= x"FF";          -- byte parameter
225
226                 when others => null;
227                end case;
228           end if;
229             case bctrtx is      --  Transfer the left bit from q register
230           when "0001" => Txdlin <= '0';    -- Start bit
231                when "0010" => Txdlin <= ssq(0);
232                when "0011" => Txdlin <= ssq(1);
233                when "0100" => Txdlin <= ssq(2);
234                when "0101" => Txdlin <= ssq(3);
235                when "0110" => Txdlin <= ssq(4);
236                when "0111" => Txdlin <= ssq(5);
237                when "1000" => Txdlin <= ssq(6);
238                when "1001" => Txdlin <= ssq(7);
239                when "1010" => Txdlin <= '1';    -- Stop bit
240                when others => null;
241            end case;
242          end if;
243    end process;
244
245    process(Strb8)                                --- Load serial Rxd from File "ArqTB10.txt"
246
247           variable IN_LINE : LINE;               -- pointer to string
248         variable linha : string(1 to 2);
```

```
249         variable stg1,stg2 : integer;
250
251     begin
252          if Reset = '0'and ini = '1' then
253          if Strb8'event and Strb8='1' and NOT(endfile(INFILE))then
254                   readline(INFILE,IN_LINE);            -- read line from a file
255               read(IN_LINE, linha);
256                    start <= '1';
257                   case linha(1) is
258                when '0' => stg1 := 0;
259                when '1' => stg1 := 1;
260                when '2' => stg1 := 2;
261                when '3' => stg1 := 3;
262                when '4' => stg1 := 4;
263                when '5' => stg1 := 5;
264                when '6' => stg1 := 6;
265                when '7' => stg1 := 7;
266                when '8' => stg1 := 8;
267                when '9' => stg1 := 9;
268                when 'A' => stg1 :=10;
269                when 'B' => stg1 :=11;
270                when 'C' => stg1 :=12;
271                when 'D' => stg1 :=13;
272                when 'E' => stg1 :=14;
273                when 'F' => stg1 :=15;
274                when others => null;
275                    end case;
276                    case linha(2) is
277                when '0' => stg2 := 0;
278                when '1' => stg2 := 1;
279                when '2' => stg2 := 2;
280                when '3' => stg2 := 3;
281                when '4' => stg2 := 4;
282                when '5' => stg2 := 5;
283                when '6' => stg2 := 6;
284                when '7' => stg2 := 7;
285                when '8' => stg2 := 8;
286                when '9' => stg2 := 9;
287                when 'A' => stg2 :=10;
288                when 'B' => stg2 :=11;
289                when 'C' => stg2 :=12;
290                when 'D' => stg2 :=13;
291                when 'E' => stg2 :=14;
292                when 'F' => stg2 :=15;
293                when others => null;
294                    end case;
295                    Rxrdin <= conv_std_logic_vector(stg1*16+stg2,8);
296                   end if;
297               if endfile(INFILE) then
298                         start <= '0';
299               end if;
```

```
300         end if;
301     end process;
302
303     process(Rckg, Strb8, Reset,  Start, ini)  -- SHIFTREG Module for RX-packet transmission
304         begin
305           if Start ='0'or ini = '0' then         --  Start transmission of the RX-packet
306                 if (Rckg'event and Rckg='1') then
307                     srq <= (others => '0');
308                     Rxds <= '0';
309                 end if;
310           else
311             if (Rckg'event and Rckg='1') then
312                  Rxds <= srq(0);
313                  srq <= '0'& srq(7 downto 1);
314             end if;
315                if Strb8'event and Strb8 ='0' then
316               srq<=Rxrdin; -- Load q register with data [7..0]
317                end if;
318           end if;
319     end process;
320
321     process(rsena, Rskg, Sckg)                          -- Write Message to File "ArqTB00.txt"
322
323           variable OUT_LINE         : LINE;         -- pointer to string
324          variable outln                    : string(1 to 2);
325           variable chr0,chr1            : Character;
326           variable int,int0,int1        : integer;
327
328     begin
329          if rsena = '0' then
330           bctrrx <= "0000";
331           RxrDou <= (others => '0');
332          else                                           -- Start Message transfer
333           if Rskg'event and Rskg = '1'then
334             bctrrx <=  bctrrx + '1';
335         end if;
336           if bctrrx = "1010" and Rxdout = '0'then
337                 bctrrx <= "0000";
338                 Resk  <='1';
339           end if;
340           if bctrrx = "1010" and  Dtrout = '0'then
341                 Resk  <='1';
342           elsif Resk = '1' and Sckg = '0' then
343                  Resk  <='0';
344           end if;
345           if Rskg'event and Rskg = '1'then
346              int := CONV_INTEGER(bctrrx);
347               if int > 0 and int < 9 then
348                 RxrDou(int-1) <= Rxdout;
349               end if;
350           end if;
```

```
351
352             if Rskg'event and Rskg = '0'then
353                  if  int = 9 then
354                   int1 := CONV_INTEGER(RxrDou(3 downto 0));
355                   case int1 is
356              when 0 => chr1 :='0';
357              when 1 => chr1 := '1';
358              when 2 => chr1 := '2';
359              when 3 => chr1 := '3';
360              when 4 => chr1 := '4';
361              when 5 => chr1 := '5';
362              when 6 => chr1 := '6';
363              when 7 => chr1 := '7';
364              when 8 => chr1 := '8';
365              when 9 => chr1 := '9';
366              when 10 => chr1 := 'A';
367              when 11 => chr1 := 'B';
368              when 12 => chr1 := 'C';
369              when 13 => chr1 := 'D';
370              when 14 => chr1 := 'E';
371              when 15 => chr1 := 'F';
372              when others => null;
373                   end case;
374                   int0 := CONV_INTEGER(RxrDou(7 downto 4));
375                   case int0 is
376              when 0 => chr0 :='0';
377              when 1 => chr0 := '1';
378              when 2 => chr0 := '2';
379              when 3 => chr0 := '3';
380              when 4 => chr0 := '4';
381              when 5 => chr0 := '5';
382              when 6 => chr0 := '6';
383              when 7 => chr0 := '7';
384              when 8 => chr0 := '8';
385              when 9 => chr0 := '9';
386              when 10 => chr0 := 'A';
387              when 11 => chr0 := 'B';
388              when 12 => chr0 := 'C';
389              when 13 => chr0 := 'D';
390              when 14 => chr0 := 'E';
391              when 15 => chr0 := 'F';
392              when others => null;
393                   end case;
394                   outln(1):= chr0;
395                   outln(2):= chr1;
396                   write( OUT_LINE, outln);
397                   writeline(OUTFILE, OUT_LINE);          -- write line to a file
398                  end if;
399             if Dtrout'event and Dtrout = '0'then
400                   FILE_CLOSE(OUTFILE);
401             end if;
```

```
402             end if;
403           end if;
404     end process;
405
406     end MAC_User_TB;
407
```

ANEXO F

Descrição dos principais sinais do Protótipo do IP Core MAC.

16xCol	Sinal do barramento de status TxSta informa que o contador (Ctr_Try) excedeu 15 tentativas.
Bkf_End	Sinal que indica final do tempo de backoff (sinal Bkf_End em nível lógico "1").
COL	Sinal que indica ao módulo de transmissão quando ocorre uma a colisão (do inglês, Collision Detect).
CRS	Sinal sensor de portadora (Carrier Sense).
Ctl_Hdp	Sinal do barramento de controle TxCtl, proveniente do dispositivo usuário, ativado para configurar comunicação Half-Duplex no módulo de transmissão.
Ctl_Res	Sinal do barramento TxCtl para reiniciar o processamento na fase deferimento.
Ctr_Bit	Valor do contador de bits.
Ctr_Byt	Valor do contador de bytes.
Ctr_CRC	Valor do contador de CRC que informa se foi completada a transmissão dos 32.
Ctr_Try	Contador de tentativas.
DAT_Bit	Sinal que informa quando é transmitido o último bit do campo de dados.
DTR	Sinal padrão da RS-232, não utilizado.
Err_Align	Sinaliza erro de alinhamento.
Err_Fcs	Sinaliza erro de FCS.
Err_Maxl	Sinaliza erro comprimento máximo.
Err_Minl	Sinaliza erro de comprimento mínimo.
FCS_Bit	Valor do contador de bits do FCS
Fil_DA	Registrador para identificar e aceitar qualquer quadro cujo endereço destino seja um
IPG_timer	Sinaliza o término do tempo mínimo entre pacotes.
LatCol	Sinal do barramento de TxSta informa colisão tardia.
LenBit	Registrador de dois bytes correspondentes ao tamanho do campo de dados do bloco delimitador de campo.
PAD_Bit	Sinal informa quando inicia-se a transmissão dos bits de enchimento.
Reg_DA	Registrador do endereço destino.
Reg_Len	Registrador do campo tamanho de quadro.
RTS	Sinal que informa quando inicia e finaliza a transmissão de dados via RS-232.
RxADD	Barramento de endereço de 11 bits para o buffer de recepção.
Rxb_Ctr_14L	Sinal delimitador de campo sinalizando um bit antes do término do campo de dados.
Rxb_Ctr_60B	Sinaliza um bit antes do início do campo FCS.
Rxb_Ctr_Max	Sinaliza que o contador atingiu contagem de 1518 bytes.
Rxbpl8	Sinal para registrar cada um dos seis bytes de endereço do campo DA.
Rxc_chk_out	Sinal que informa o resultado da comparação de um a um dos bytes do campo FCS recebidos com as respectivas posições de oito bits do barramento Rxg_crc_res.

RxClk Sinal relógio de recepção.

RxCtl Barramento para sinais de controle na configuração do RxMAC.

RXD Linha para recepção de dados da interface RS-232

RxD Fluxo serial de bits proveniente do meio físico, recebido pelo RxMAC.

RxDAT Barramento de dados de 8 bits para o buffer de recepção.

Rxdout Sinal, quando é ativado, sinaliza, com exatidão, o término do SFD e o correspondente início do quadro (primeiro bit do campo DA).

Rxf_daf_out Sinal composto de dois bits, que são combinados para representar o valor 1 (endereço individual), o valor 2 (endereço de difusão) e o valor zero caso nenhum dos dois anteriores ocorra.

Rxg_crc_res Registrador para reter o valor computado no gerador de CRC.

RxMAC Módulo de controle para recepção no IP Core Ethernet.

RxMsg Sinal informando que a mensagem está no buffer recepção, pronta para ser está realizando a transferência da mensagem ao sistema usuário.

RxRD Sinal de controle de leitura para buffer de recepção.

RxRdy Sinal que informar ao RxMAC que o sistema usuário de recepção está realizando a transferência da mensagem armazenada no buffer de recepção..

RxSta Barramento para sinais que informam o resultado das operações de recepção.

SCLK Sinal relógio do sistema.

TEN Sinal usado para habilitar o módulo PHY para transmissão do fluxo serial de bits gerado no IP cor (do inglês, Transmit Enable).

TxADD Barramento de endereço de 11 bits para o buffer de transmissão.

TxClk Sinal relógio de transmissão.

TxCtl Barramento para sinais de controle na configuração do TxMAC.

TXD Linha para transmissão de dados da interface RS-232

Txd Barramento de 4 bits para entrada do módulo PHY.

TxD Fluxo serial de bits gerado no TxMAC.

TxDAT Barramento de dados de 8 bits para o buffer de transmissão.

Txds Fluxo serial gerado pelo registrador de deslocamento.

Txl_112 Sinal para o processo de delimitação de dados e PAD (ativo em nível lógico "1").

Txl_480 Sinal para o processo de delimitação do final do campo de PAD (ativo em nível lógico "1").

Txl_512 Sinal para o processo de delimitação do final de quadro mínimo (ativo em nível lógico "1").

TxMAC Módulo de controle para transmissão no IP Core Ethernet.

TxMsg Sinal informando que a mensagem está no buffer transmissão, pronta para ser encapsulada em um pacote e transmitida ao meio físico (ativo em nível lógico "1").

TxRdy Sinal informando ao sistema usuário que o módulo TxMAC está em processo de acesso ao meio para a transmissão do pacote (desativado em nível lógico "0").

Txs_din Barramento entrada de 8 bits do bloco registrador de deslocamento para conversão dos dados paralelos em um fluxo serial Txds.

Txs_ena Sinal interno ao IP core habilita o bloco registrador de deslocamento para conversão dos dados paralelos.

Txs_ld4 Sinal, proveniente do bloco marcador de byte, informa o meio do byte (em inglês, *nibble*) a ser serializado. Este também é utilizado pelo registrador para converter, os bytes de sua entrada em pares de nibbles para o módulo PHY.

Txs_ld8 Sinal, proveniente do bloco marcador de byte, para carga do registrador de deslocamento

TxSta Barramento para sinais que informam o resultado das operações de transmissão.

TxWR Sinal de controle de escrita para buffer de transmissão.

REFERÊNCIAS BIBLIOGRÁFICAS

[ADA97] A. M. S. Adário, "Implementação em FPGA de um Processador de Vizinhança para Aplicação em Imagens Digitais," *Dissertação de Mestrado*, UNICAMP, 1997.

[AHR90] M. Ahrens, et. al, "An FPGA family optimized for high densities and reduced routing delay," in *Custom Integrated Circuits Conference, CICC'90*, pp. 31.5.1-31.5.4, May, 1990.

[ALG89] Algotronix, Ltd, "CAL 1024 Datasheet," Edinburgh, Scotland, 1989.

[ALT01a] Altera, "ACEX 1K, Embedded Programmable Logic Device Family Data Sheet," version 4.1, March, 2001.

[ALT01b] Altera, "FLEX 10K, Embedded Programmable Logic Device Family Data Sheet," version 2.0, March, 2001.

[ALT01c] Altera, "APEX 20K, Embedded Programmable Logic Device Family Data Sheet," version 3.5, April, 2001.

[AMD90] AMD Inc., "Mach Devices High Density EE Programmable Logic Data Book," 1990.

[ATH93] P. M. Athanas and H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis," *IEEE Computer*, vol 26, no. 3, pp 11-18, March, 1993.

[ATM99] ATMEL Corporation, "Data Acquisition Systems Using Cache Logic FPGAs", Application Note, 1999.

[BAK91] S. Baker, "Lattice fields FPGA," *Eleci Fog Times.* no. 645, p. 1, June, 1991.

[BER89] P. Bertin, D. Roncin and J. Vuillemin, "Introduction to Programmable Active Memories," *Paris Research Lab*, Report 3, June, 1989.

[BER00] J. Bergeron, "Writing Testbenches - Functional Verification of HDL Models," *Kluwer Academic Publishers*, 2000.

[BIR91] J. Birkner, A. Chan, H. T. Chua, A. Chao, K. Gordon, B. Kleinrnan, P. Kolze, and R. Wong, "A very high-speed field programmable gate array using metal-to-metal anti-fuse programmable elements," in *New Hardware Product Introduction* at Custom Integrated Circuits Conference, CICC'91, 1991.

[BRO92] S. D. Brown, R. Francis, J. Rose and Z. Vranesic, "Field-Programmable Gate Array," *Kluwer Academic Publishers*, International Series in Engineering and Computer Science, June, 1992.

[BOS96] G. Bostock, "FPGA Combines Multiple Serial Interfaces and Logic", *Butterworth-Heinemann*, Oxford, 1996.

[BUE96] D. A. Buell, J. M. Arnold and W. J. Kleinfelder, "Splash 2: FPGAs in Custom Computing Machines," *IEEE Computer Society Press*, 1996.

[BUR00] D. Bursky, "Splash 2: FPGAs in Custom Computing Machines," *Electronic Design*, pp. 74-78, October, 2000.

[CAL98] N. L. V. Calazans, "Projeto Lógico Automatizado de Sistemas Digitais Seqüenciais," *Imprinta Gráfica e Editora Ltda*, 342 páginas, Rio de Janeiro, 1998.

[CAP01] E. A. Cappelatti, "Barramento de Alto Desempenho para Interação Hardware/Software". Dissertação de Mestrado, FACIN - PUCRS, junho, 2001.

[CAS97] J. Case, N. Gupta, J. Mittal and D. Ridgeway, "Design Methodologies for Core-Based FPGA Designs", *Xilinx white paper*, April, 1997. Disponível em http://www.xilinx.com /products/logicore/pci/docs/pciplus.pdf.

[CIR99] Cirrus Logic, Inc., "Cristal LAN ISA Ethernet Controller," *CS8900A Product Data Sheet*, 1999. Manual disponível em http://www.cirrus.com.

[CON91] Concurrent Logic, "CFA6006 Field-Programmable Gate Array Data Sheet," 1991.

[COM01] D. E. Comer, "Redes de Computadores e Internet," Bookman, Segunda Edição, 2001.

[COM98] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, NEC Corporation. "Universal Serial Bus Specification," 1998. Disponível em: http://www.usb-by-example.com/USBdocs/index.htm.

[ELD94] J. G. Eldredge, B. L. HUTCHINGS, "RRANN: The Run-Time Reconfiguration Artificial Neural Network," In: *Custom Integrated Circuits Conference, CICC'94*, pp. 77-80, 1994.

[FRA00] J. L. Fragoso, E. Costa, J. Rochol, S. Bampi, R. Reis, "Specification and Design of an Ethernet Interface Soft IP," *Journal of the Brazilian Computer Society, JSBC*, vol. 6, no. 3, April, 2000.

[GAM89] A. El Gamal, et al., "An architecture for electrically configurable gate arrays," *IEEE Journal of Solid-State Circuits - JSSC*, vol. 24, no.2, pp. 39-398, April 1989.

[GOK91] M. Gokhale, W. Holmes, A. Kopser and S. Lucas, "Building and Using a Highly Parallel Programmable Logic Array," *IEEE Computer*, vol 24, no. 1, pp. 81-89, January, 1991.

[GOK90] M. Gokhale, W. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minnich and P. Olsen. "SPLASH: A Reconfigurable Linear Logic Array". In: *Proceedings of the International Conference on Parallel Processing, ICPP´90*, August, 1990.

[HEN96] J. L. Hennessy and D. A. Patterson, "Computer Architecture: a quantitative approach," *Morgan Kaufmann Publishers*, Second Edition, 1996.

[IEE00] Institute of Electrical and Electronics Engineers, Inc. "IEEE Std 802.3,2000 Edition - Part 3:Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications," *IEEE Standards Association*, 2000.

[IEE99] Institute of Electrical and Electronics Engineers, Inc. "IEEE Std 802.11a-1999 - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHZ Band," *IEEE Standards Association*, 1999.

[KYN88] V. N. Kynett, A. Baker, M. L. Fandrich, G. P. Hoekstra, O. Jungroth, J. A. Kreifels, S. Wells, and M. D. Winston, "An In-System Reprogrammable 32Kx8 CMOS Flash Memory," *IEEE Journal of Solid-State Circuits, JSSC*, vol. 23. no. 5, October 1988.

[LAL90] P. K. Lala, "Digital System Design Using Programmable Logic Devices," *Prentice Hall*, 1990.

[LEV98] Level One Communications, "LXT901A/907A Universal Ethernet Transceiver Data Sheet," Revision 1.2, March 1998.

[LIN94] C. Lin, M. Marek-Sadowska and D. Gatlin, "Universal Logic Gate for FPGA Design," In: *IEEE/ACM International Conference on Computer-Aided Design, ICCAD´94,* pp. 164-168, 1994.

[LUC86] C. L. Lucchesi, "Introdução à Criptografia Computacional," *Escola Brasileiro-Argentina de Informática*, 1986.

[MAR92] D. Marple and L. Cooke, "An MPGA compatible FPGA architecture," in*, ACM First International Workshop on Field Programmable Gate Array*, *FPGA 92*, pp. 39-44, February, 1992.

[MAR00] A. Marquardt, V. Betz and J. Rose, "Speed and Area Tradeoffs in Cluster-Based FPGA Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no.1, pp. 84-93, February, 2000.

[MAZ92] S. Mazor and P. Langstraat, "A guide to VHDL," *Kluwer Academic Publishers*, Massachusetts, 1992.

[MET76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer," *Communications of the ACM*, Vol. 19, No. 5, pp. 395 – 404, July 1976. Disponível em: http//www.acm.org/classics/apr96/.

[MIC96] G. de Micheli and M. Sami, "Hardware/Software Co-Design," *Kluwer Academic Publishers*, 1996.

[MOR01] F. G. Moraes, A. M. Amory, N. L. V. Calazans, E. Bezerra e J. Petrini Jr., " Using the CAN protocol and reconfigurable computing technology for web-based smart house automation," In: *14th Symposium on Integrated Circuits and Systems Design, SBCCI'01*, pp. 38-43, 2001.

[MUR91] H. Muroga, H. Murata, Y. Sacki, T. Hibi, Y. Ohash. T. Nuguchi, and T. Nishimura, "A large scale FPGA with 10K core cells with CMOS 0.8 μm 3-layered metal process," In: *Custom Integrated Circuits Conference, CICC'91*, pp. 6.4.1-6.4.4, May, 1991.

[PAG96] I. Page, "Reconfigurable Processor Architectures," *Microprocessors and Microsystems*, vol. 20, no. 3, pp 185-196, May, 1996. Disponível em: ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Ian.Page/proc_arch.ps.gz.

[PLE89] Plessey Semiconductor, "ERA60100 preliminary data sheet," 1989.

[ROS90] J. S. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic functionality on area efficiency," *IEEE Journal of Solid-State Circuits, JSSC*, vol. 25, pp.1217-1225, October, 1990.

[ROS93] J. S. Rose and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings of the IEEE*, vol. 81 no. 7, July 1993.

[SAN99] E. Sanchez, M. Sipper, J. O. Haenni, J. L. Beuchat, A. Stauffer and A. P. Uribe, "Static and Dynamic Configurable System," *IEEE Transactions on Computers*, vol. 48 no. 6, pp. 556-564, June, 1999.

[SHA99] T. Shanley and D. Anderson, "PCI System Architecture," *Addison-Wesley Publishing Company,* Fourth Edition, PC System Architecture Series. 787 pages, June, 1999.

[SOA95] L. F. G. Soares, G. L. Souza Filho e S. Colcher, "Redes de computadores: das LANs, MANs e WANs às Redes ATM," *Editora Campus*, 2a Edição, 1995.

[SOL98] E. F. Solari, G. Willse, "PCI Hardware and Software," *Annabooks*, Fourth Edition, 1998.

[SPU00] C. E. Spurgeon, "Ethernet: the definitive guide." *O´Reilly and Associates, Inc.*, February, 2000.

[SWA93] T. Swan, "Programando em Pascal 7.0 Para Windows - Borland", *Editora Berkeley*, Tradução Rolf Reiger Jr. - Rio de Janeiro, 1993.

[TAU95] E. Tau, I. Eslick, D. Chen, J. Brown, and A. DeHon, "A First Generation DPGA Implementation," In: *Proceedings of the Third Canadian Workshop on Field-Programmable Devices*, pp. 138-143, May 1995. Disponível em: http://www.cs.caltech.edu /research/ic/transit/dpga/dpga-proto-fpd95.ps.Z.

[USB99] USB Implementers' Forum, "USB Class Definitions for Communication Devices," 1999. Disponível em: http://www.usb.org/developers/dwg/communications class_grp/.

[VAH01] F. Vahid and T. Givargis, "Platform tuning for embedded systems design," *IEEE Computer*, vol 34, no.3, pp. 112-114, March, 2001.

[VIJ92] S. Vij and B. Ahanim, "A high density, high speed, array-based erasable programmable logic device with programmable speed/power optimization," In: *ACM First Workshop on Field-Programmable Gate Arrays, FPGA´92*, pp.29-32, February, 1992.

[VIR98] Virtual Computer Corporation, "H.O.T. II & Hardware Object Technology - Hardware API Guide Version 2.1," *Virtual Computer Corporation*, 40pages, 1998.

[VIR99] Virtual Computer Corporation, "The Virtual Workbench Guide," *Virtual Computer Corporation*, Version 1.02.2, 57 pages, 1999.

[WIR95] M. J. Wirthlin and B. L. Hutchings, "DISC: The dynamic instruction set computer," In: *Field Programmable Gate Arrays for Fast Board Development and Reconfigurable Computing*, John Schewel, Editor, Proceedings of the SPIE 2607, pp. 92-103, 1995.

[WIR98] M. Wirthlin and B. Hutchings. "Improving Functional Density Using Run-Time Circuit Reconfiguration," *IEEE Transactions on Very Large Scale Integration Systems, ITVLSI*, pp. 247-256 Vol 6, no. 2, June, 1998.

[WON89] S. C. Wong et al, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proc. 1989 Custom Integrated Circuits Conference, CICC´89*, pp. 5.8.1-5.8.4. May 1989.

[XES00] Xess Corporation, "XSV Board V1.0 Manual", *Manual do fabricante*, Setembro, 2000.

[XIL99] Xilinx Corporation, "Xilinx Data Book". *Manual de fabricante,* 1999.

Disponível em: http://www.xilinx.com/partinfo/databook.htm

[XIL00] Xilinx Corporation, "FastFLASH XC9500XV High-Performance, Low-Power CPLD Family," A*dvance Product Information*, DS049 (v1.1) February, 2000.

[XIL00a] Xilinx Corporation, "Rev. 1 Second Quarter 2000 DataSource CD-ROM," *Manual do fabricante*, March, 200