PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Núcleo IP de uma Bridge Ethernet Baseado em Lógica Reconfigurável e Processador SoftCore

FABIO SIDIOMAR ZAMPERETTI DUARTE

PORTO ALEGRE
2007

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Núcleo IP de uma Bridge Ethernet Baseado em Lógica Reconfigurável e Processador SoftCore

FABIO SIDIOMAR ZAMPERETTI DUARTE

Orientador: Prof. Dr. Fabian Luis Vargas

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica, da Faculdade de Engenharia da Pontifícia Universidade Católica do Rio Grande do Sul, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica.

PORTO ALEGRE
2007

Núcleo IP de uma Bridge Ethernet Baseado em Lógica Reconfigurável e Processador SoftCore

CANDIDATO: FABIO SIDIOMAR ZAMPERETTI DUARTE

Esta dissertação foi julgada para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul.

_____

Prof. Dr. Daniel Ferreira Coutinho
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

BANCA EXAMINADORA

_____

Prof. Dr. Fabian Luis Vargas – Presidente

_____

Prof. Dr. Eduardo Augusto Bezerra – PUCRS

_____

Prof. Dr. Rubem Fagundes – PUCRS

AGRADECIMENTOS

Agradeço primeiramente a meu professor e orientador, Fabian Luis Vargas, por seu apoio, dedicação e profissionalismo.

A todos os meus colegas de trabalho, em especial ao colega Alexandre Cheuiche, pelas horas de orientação sobre o desenvolvimento de meu trabalho.

A meus pais, por terem me dedo a oportunidade de um bom ensino, sem o qual não chegaria até aqui.

Por fim, mas não menos importante, à minha esposa, que neste período de desenvolvimento do trabalho teve a paciência em admitir a minha ausência, principalmente agora que carrega em seu ventre a personificação de nossa felicidade.

RESUMO

O constante aumento na densidade dos dispositivos de lógica programável (FPGA's), aliado à diminuição dos preços destes circuito integrados, tem viabilizado a implementação de sistemas complexos, que antes necessariamente implicavam no uso de circuitos integrados dedicados.

Em projetos onde um FPGA já é utilizado, justifica-se ainda mais facilmente a integração de novas funcionalidades ao projeto de lógica programável, uma vez que os custos envolvendo as ferramentas de desenvolvimento, tanto de hardware quanto de software, já foram contabilizados.

Este trabalho implementa uma *bridge* ethernet através de um sistema composto por um *softprocessor*, onde as funções relativas à classificação e encaminhamento dos pacotes são realizadas em software, o que torna o sistema mais acessível à mudanças na implementação e de fácil manutenção. Além do *softprocessor*, implementados em VHDL ainda temos um controlador de acesso ao meio físico ethernet (MAC) e um controlador HDLC o qual é utilizado como ponto de ligação entre as bridge local e a *bridge* remota.

A prototipagem do sistema, para avaliação e análise de desempenho, é feita com o uso das ferramentas de software e placas de desenvolvimento de hardware da Xilinx, por serem de fácil acesso e que oferecem o núcleo de *softprocessor* MicroBlaze, um microprocessador RISC de 32 bits com arquitetura *harvard*.

A análise de desempenho do sistema, realizada com o auxílio de ferramentas de software (Iperf) e hardware (SmartBits), mostrou que a bridge consegue atingir taxas acima de 1Mbps com pacotes pequenos (64 bytes), típicos das aplicações VoIP. Para pacotes maiores, o desempenho se aproximou dos 2 Mbps, que representam a taxa típica máxima onde este dispositivo será usado na prática.

Devido à sua natureza extremamente maleável, em vista da utilização de lógica programável e de funções de software, o sistema permite a inclusão de novas facilidades em atividades futuras, tais como a filtragem de pacotes, redes locais virtuais (VLAN's) e o protocolo *Spanning Tree.* Além de novas funcionalidades de software, novos módulos do hardware sintetizável também podem ser incorporados, sejam para desempenhar novas funções, tais como o aumento das interfaces WAN, como para otimizar módulos já existentes.

ABSTRACT

The constant increase of density in today´s programmable logic devices (FPGA's), together with the lowering of prices of these integrated circuits, has been making possible the implementation of complex systems which, some time ago, would require dedicated integrated circuits.

In designs where an FPGA is already in use, it is even easier to justify the integration of new functionalities to the programmable logic project, as the costs involving the software and hardware development tools have already been used.

This work implements an Ethernet *bridge* using a system composed by softprocessor, where the functions related to the packet classification and forwarding are executed in software, what makes the system far more versatile and friendly to implementation changes in the future, as well as easy maintenance. Besides the softprocessor, implemented in VHDL there are the media access controller (MAC) and an HDLC controller, which is used as the connection point between the local and remote *bridge*s.

The prototyping of the system, to evaluate the performance, has been done using the software tools and development boards from Xilinx, since they were easily accessible and offer the MicroBlaze softprocessor IP core, a 32 bit RICS processor with harvard architecture.

The performance analysis of the system, done with use of software tools like Iperf and hardware tools like SmartBits, has shown that the *bridge* was fast enough to handle small packets at a rate over 1Mbps. For larger packets, the performance was close to the 2Mbps, which represent the maximum typical rate where this *bridge* will be inserted in the real applications.

Due to its extremely versatile nature, having been implemented using programmable logic and software functions, the system can handle the inclusion of new features in future activities, such as packet filtering, virtual LAN's and the Spanning Tree Protocol. Besides these new software functionalities, new hardware modules can also be inserted, be it either to implement new features, such as the increase in the number of WAN interfaces, or to simply optimize existing logic blocks.

Sumário

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

2B1Q	2 Binários 1 Quaternário
ATM	*Asynchronous Transfer Mode*
BRAM	*Block RAM*
CAM	*Content Addressable Memory*
CPLD	*Complex Programmable Logic Device*
CRC	C*yclic Redundancy Check*
CSMA/CD	C*arrier Sense Multiple Access / Collision Detection*
DSL	*Digital Subscriber Line*
EDK	*Embedded Development Kit*
FCS	*Frame Check Sequence*
FPGA	*Field Programmable Gate Array*
GNU	*GNU's Not Unix (acrônimo recursivo)*
HDLC	*High-level Data Link Control*
h(K)	Função de hash '*h*' aplicada na chave 'K'
IETF	*Internet Engineering Task Force*
IP	*Intellectual Property*
ISO	*International Organization for Standardization*
ITU	*International Telecommunication Union*
LAN	*Local Area Network*
MAC	*Media Access Control (controller)*
MSDSL	*Multi-rate Symmetric DSL*
OUI	*Organizationally Unique Identifier*
OSI	*Open Systems Interconnect*
RSTP	*Rapid Spanning Tree Protocol*
SHDSL	*Single-Pair High-Bit-Rate DSL*
SOHO	S*mall Office/Home Office*
STP	*Spanning Tree Protocol*
TCP	*Transmission Control Protocol*
TDM	*Time division Multiplexing*
UDP	*User Datagram Protocol*
VLAN	*Virtual LAN*
VDSL	*Very-high-bit-rate DSL*
WAN	*Wide Area Network*

1. INTRODUÇÃO

Redes Ethernet estão se tornando o padrão de convergência de inúmeros serviços. Temos como exemplos os serviços de voz sobre IP, a emulação de circuitos síncronos sobre redes Ethernet bem como o transporte de TV sobre redes IP (IPTV).

O princípio de funcionamento das redes Ethernet foi definido ainda na década de 1970 [45] e, dada sua simplicidade e eficácia, seu uso se tornou muito difundido. Ao longo do tempo, a introdução de protocolos adicionais, bem como o aumento da velocidade de transmissão de dados, tornaram as redes Ethernet ainda mais atrativas aos usuários e fabricantes de equipamentos.

Tão importantes quanto os equipamentos terminais (ex.: computadores) em uma rede Ethernet, são aqueles responsáveis pela interligação destas redes, como por exemplo, as *bridges*, *switches* e roteadores.

1.1. MOTIVAÇÃO

Há muitos anos estamos presenciando um constante crescimento do mercado de redes de computadores e Internet para usuários domésticos. Da mesma forma, no mercado corporativo, atendido pelas grandes empresas de telecomunicações do país, como Telefónica, Brasil Telecom e EMBRATEL, a demanda por equipamentos para interconexão de redes de computadores tem crescido bastante, dado o aumento do uso destas redes.

A forma mais usual pela qual uma empresa privada está conectada a uma empresa de telecomunicações é através de links de 2Mbits por segundo que carregam 30 canais de voz de 64k bits/s [14]. Uma vez que a conexão física, entenda-se o par de fios de cobre, já está estabelecida, e que em geral o link de dados ainda pode ter sua taxa acrescida, fica evidente a possibilidade de uso de equipamentos que possam multiplexar o uso deste link de dados de forma a prover mais serviços.

Cientes desta possibilidade, as empresas de telecomunicações se voltaram aos fabricantes de equipamentos e colocaram como requisito técnico a inclusão da funcionalidade de *bridge* ethernet nos modens de 2Mbits/s. Com estes equipamentos, é possível prover serviços de voz e dados IP sobre o mesmo par de fios através da multiplexação no tempo (TDM). Com o advento dos modens SHDSL sob 2 pares de fios [35] é possível alcançar taxas de dados de até 2304k bits/s em cada par, ou seja, uma taxa total de 4608k bits/s. Desta forma, mantendo-se o serviço de

voz de 30 canais, que demanda uma taxa de 2048k bits/s, ainda é possível prover ao cliente um serviço de dados de 2560k bits/s utilizando-se o mesmo link de dados.

1.2. JUSTIFICATIVA

A PARKS foi a primeira empresa brasileira a incorporar este tipo de funcionalidade em um modem digital. O equipamento em questão foi o Power512 MSDSL no ano de 2000. Modem digital para transporte de dados sobre um ou 2 pares de fios até 512k bits/s, o Power512 MSDSL utilizava codificação de linha 2B1Q (2 binários 1 quaternário) e possuía, além da interface Ethernet, também a interface G.703 (para transporte fracionado de voz) bem como as interfaces V.35 e V.36. Para a solução de *bridge* o modem utilizava o RJ-017 da RAD.

Dado o alto preço deste componente e por uma estratégia de mercado, a empresa optou por não continuar mais a oferecer este tipo de característica em um modem de acesso e passou a implementá-la em equipamentos de maior valor agregado: os roteadores com modens digitais de alta velocidade integrados Powerlink 510 e Powerlink 514. Neste caso a função de *bridge* era uma característica extra, implementada em software, através de uma plataforma Linux, o que requeria uma quantidade considerável de código.

Atualmente, novamente por solicitação do mercado, a implementação da característica de *bridge* em modens de acesso, como os modens SHDSL tornou-se mais uma vez obrigatória.

O mercado brasileiro de telecomunicações é extremamente competitivo. Soluções inovadoras que permitam redução nos custos de produção e no *time-to-market* de um novo produto ou mesmo de uma nova característica adicionada a um produto existente são de extrema importância e podem ser fator decisivo na sobrevivência e no faturamento de uma empresa.

Com o aumento cada vez maior na densidade dos circuitos integrados de lógica programável (FPGA's), também tem aumentado a viabilidade de se utilizar sistemas microprocessados baseados em *softcores* para desempenhar tarefas complexas nas áreas de telecomunicações e redes de computadores.

O que antes necessitava de um projeto de hardware extremamente complexo e de difícil manutenção e escalabilidade, agora poderá ser dividido em blocos software, apoiados sobre uma plataforma de hardware amplamente disponível e bem documentada.

Além de contribuir com um aumento na experiência de utilização de *soft processors* para desempenho de funções ligadas à área de redes de computadores e de telecomunicações, este

projeto propõe-se a dar o passo inicial para que no futuro possam ser desenvolvidas alternativas às soluções em hardware existentes no mercado, que não possuem muita flexibilidade, tais como as *bridges* da RAD e da ADMTek descritas no capítulo 4.

1.3 OBJETIVOS

O principal objetivo do trabalho consiste em implementar e testar as funcionalidades básicas de um conversor Ethernet/HDLC com funções de *bridge*, de duas portas, baseado em um *soft processor* instanciado em FPGA's Xilinx.

Um *soft processor* consiste em um microprocessador implementado através de lógica programável (FPGA), sendo que a maior vantagem neste processo consiste na flexibilidade oferecida por estes dispositivos. A *bridge* deve implementar suas funções em software, o qual é executado pelo microprocessador embarcado, o que proporcionará maior liberdade e facilidade na colocação de novas características que venham a ser solicitadas no futuro.

Uma *bridge* ethernet é um dispositivo utilizado para a interligação de dois segmentos de uma rede de computadores. Tais dispositivos trabalham na camada 2 do modelo OSI, muito embora hoje em dia já existam equipamentos que além de desempenharem as funções de *bridge*, também realizam roteamento de pacotes IP, no nível 3 da camada OSI.

Uma das principais características de uma *bridge* é o aprendizado de endereços. Este mecanismo permite que apenas os pacotes que têm como destino o segmento adjacente passem de uma porta a outra evitando, assim, que tráfego desnecessário seja gerado no segmento adjacente. Além disso, no caso de bridges interligadas através de *links* de dados com taxa limitada, usualmente menor do que a máxima taxa gerada pelo segmento de rede, a função de aprendizado de endereços evita o desperdício da banda de dados que interliga as duas portas da *bridge* (através de modens por exemplo).

O projeto em questão propõe o desenvolvimento de um dispositivo de duas portas que estarão interligadas, na prática, através de um link digital de alta velocidade. Este link poderá ser implementado a partir do uso de modens digitais banda-base ou com tecnologia mais avançada, tais como MSDSL, SHDSL, VDSL, etc.

A figura 1.1 apresenta um modelo simplificado da *bridge*. Através deste modelo, fica claro que esta é composta por dois dispositivos completamente independentes. Cada um destes dispositivos executa as mesmas funções, porém conectados a segmentos de rede que estão

separados fisicamente, podendo estar a vários a quilômetros de distância um do outro, dependendo da tecnologia empregada na transmissão dos dados e da distância física entre os equipamentos.

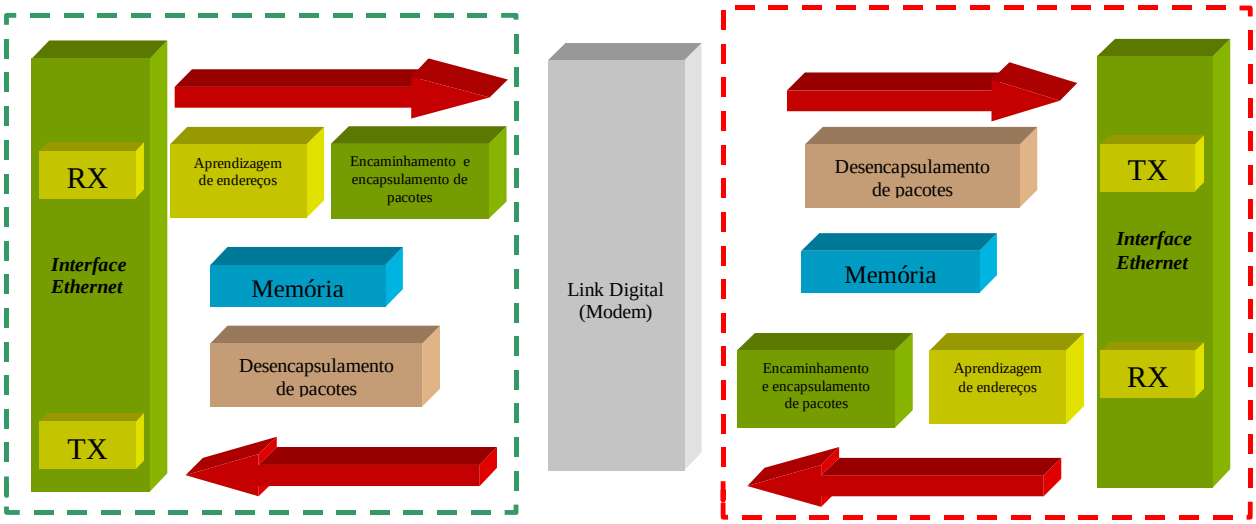


Figura 1.3.1 - Modelo simplificado da bridge

Funcionamento básico do modelo

No modelo simplificado acima, os pacotes que chegam pela interface Ethernet (RX) inicialmente passam pelo bloco de aprendizagem de endereços, que "aprende" quais estações estão conectadas a este segmento de rede através da leitura do endereço de origem do pacote ethernet.

Em seguida, o pacote é repassado ao bloco de encaminhamento, que decide se o mesmo deve ser enviado ao outro lado da *bridge* ou não, baseado na informação do endereço de destino do pacote. Caso positivo, o pacote é encapsulado em um quadro de dados tipo HDLC e enviado ao outro lado da *bridge*. Visto que este dispositivo possui apenas duas portas, a tarefa da parte de transmissão resume-se apenas a checar a integridade do pacote recebido através do link de dados, remover o cabeçalho HDLC e enviar o pacote através da interface ethernet ao segmento de rede conectado a esta.

Durante o desenvolvimento do trabalho, incluindo a fase de testes, o link de dados que interliga os dois dispositivos que contemplam a *bridge* foi implementado através do uso de cabos de interconexão, simplificando o desenvolvimento e validação do projeto.

As *bridge* realiza as funções descritas abaixo:

No sentido Ethernet => HDLC

- aprendizado de endereços;
- envelhecimento de endereços na tabela;
- encaminhamento de pacotes baseando-se nos endereços de destino dos mesmos;
- encapsulamento dos pacotes ethernet em um frame HDLC;

No sentido HDLC => Ethernet

- desencapsulamento dos pactotes;
- encaminhamento dos pacotes à interface Ethernet;

PARTE I - ESTADO DA ARTE, PROBLEMAS E SOLUÇÕES

2. SWITCHES ETHERNET

2.1. HISTÓRICO DE REDES ETHERNET

A figura 2.1.1 é o primeiro esboço, feito por Robert Metcalfe no ano de 1976, de uma rede Ethernet.

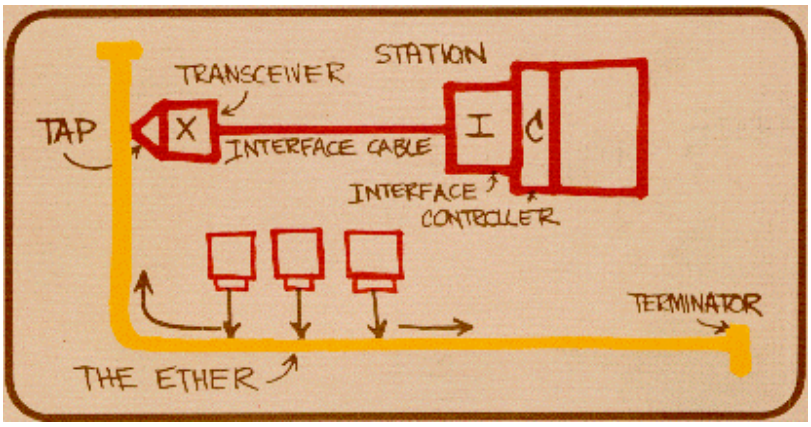


Figura 2.1.1 - Primeiro esboço de uma rede ethernet

O conceito apresentado por Metcalfe é utilizado até os dias de hoje. O controlador de interface descrito por ele é representado pelos controladores MAC e o transceptor são os PHY's (responsáveis pela camada física).

Em seu artigo [45], Metcalf e Boggs descreveram um sistema de comunicação de dados que podia crescer sem maiores problemas à medida em que mais estações fossem sendo adicionadas. Isso foi possível pois Metcalf e Boggs propuseram a distribuição do controle através de todos os computadores do sistema. Isso também eliminou os problemas de confiabilidade causados pela colocação do controle em um só nó do sistema, assim como a criação de gargalos em um sistema que é rico em paralelismo. A chave para o sucesso deste tipo de sistema reside na escolha de seus criadores em fazê-lo com custo baixo e fácil de ser mantida.

A primeira rede experimental deste tipo foi desenvolvida para interconectar os computadores da Alto da Xerox, em 1972. Na época, o relógio para a geração da taxa de transmissão de dados foi derivado do próprio relógio de operação do Alto, o que resultou em uma taxa de 2,94 Mbits/s. Inicialmente a rede criada por Metcalfe chamava-se *Alto Aloha Network,*

pelo fato de ser utilizada para conectar os computadores *Alto* e por ser inspirada em uma outra rede experimental chamada *Aloha Network.* Já em 1973, Metacalfe mudou o nome da rede para Ethernet, de forma a deixar claro que a rede poderia ser utilizada por qualquer computador, não somente os *Alto's.*

No ano de 1977, Robert Metacalfe e outros engenheiros da Xerox receberam uma patente Norte Americana para a Ethernet como um "Sistema de comunicação de dados multiponto com detecção de colisão". O primeiro padrão relativo à Ethernet a 10M bits/s foi publicado no ano de 1980 por um consórcio de empresas composto pela DEC, Intel e Xerox e ficou conhecido como padrão DIX. Esta norma continha as especificações de operação da Ethernet bem como as especificações para um sistema de transmissão em um único meio baseado em um cabo coaxial.

Após a publicação do padrão DIX, o IEEE começou a despender esforços para o desenvolvimento de um padrão aberto. Em 1985, o primeiro padrão do IEEE foi publicado sob o nome de *IEEE 802.3Carrier Sense Multiple Access with Colision Detection (CSMA/CD) Access Method and Physical LAyer Specifications.*

2.2. A DISTRIBUIÇÃO DE ENDEREÇOS MAC PELO IEEE

Todo o encaminhamento de pacotes em uma rede ethernet feita pelos switches e bridges é feito baseando-se no endereço de máquina das estações envolvidas. O número MAC é, ou pelo menos deveria ser, único para cada equipamento no mundo. A entidade que faz a distribuição destes endereços para as empresas interessadas é o IEEE (*Institute of Electrical and Electronics Engineers*) através do ***IEEE Registration Authority*** [36].

Para que uma empresa que fabrica equipamentos para redes de computadores que necessitam de endereços MAC obtenha uma gama de endereços é necessário o registro da mesma no IEEE. Após o registro, a empresa pode solicitar um OUI (*Organizationally Unique Identifier*). Este identificador é um número de 24 bits que identifica a empresa. Após a obtenção do OUI, a empresa obtém a gama de endereços MAC concatenando o OIU com mais 24 bits, formando assim, os endereços para uso em seus equipamentos.

A tabela 2.2.1 demonstra a maneira que o OUI deve ser concatenado com os demais 24 bits para formar o endereço ethernet completo.

Tabela 2.2.1 – Endereço MAC e o OUI

Endereço MAC					
Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
OUI			Gama de endereços para a empresa		

Como podemos observar, com apenas um OUI é possível a obtenção de 2^{24} endereços diferentes, ou seja mais de 16,7 milhões de combinações.

No endereço do ***IEEE Registration Authority*** é possível fazer a consulta aos OUI's públicos concedidos pelo IEEE. Empresas de pequeno porte normalmente só precisarão de um OUI durante muito tempo. Porém, para termos uma idéia do tamanho de grandes empresas desta área, como a CISCO Systems, a mesma possui algumas centenas de OUI's registrados no IEEE, algo em trono de 300 em Dezembro de 2006. A figura 2.2.1 mostra como os OUI's são listados na base de consultas do ***IEEE Registration Authority***.

```
00-04-16   (hex)          Parks S/A Comunicacoes Digitais
000416     (base 16)      Parks S/A Comunicacoes Digitais
                          Av. Pernambuco, 1001
                          Porto-Alegre-RS CEP 90240-004
                          BRAZIL
                          BRAZIL
```

Figura 2.2.1 – OUI da Parks S/A

2.3. TOPOLOGIAS DE REDES ETHERNET

As redes Ethernet mais antigas trabalhavam com apenas um domínio de colisão, em outras palavras, compartilhavam o mesmo meio físico. Dessa forma, um protocolo de controle de acesso ao meio físico precisava ser utilizado. Neste caso, o CSMA/CD (*carrier sense multiple access / collision detection)*.

No protocolo CSMA/CD, as estações que desejam transmitir esperam até que o meio físico esteja livre para iniciar a transmissão de seus dados (*carrier sense*). Se várias estações tentam transmitir em um mesmo instante (*multiple access*), simultaneamente, uma colisão ocorre

e então cada estação espera por um intervalo de tempo randômico para tentar transmitir novamente (*collision detection*). O acesso ao meio é, portanto, aleatório e assíncrono, ou seja, quando os dados chegam ao destino, a estação receptora não envia nenhuma confirmação de que recebeu o pacote.

Um segmento de rede Ethernet é definido como um domínio de colisão, arbitrado pelo protocolo CSMA/CD, formando uma topologia lógica em barramento e fisicamente uma topologia em barra ou estrela.

Um domínio de colisão pode ser estendido com o uso de repetidores (*repeaters*) na topologia em barra e com o uso de *hubs* (nó central) na topologia em estrela, mas não haverá segmentação da rede, pois as estações dos dois trechos de rede conectados pelo repetidor ou *hub* irão competir pelo acesso ao mesmo local, constituindo ainda, um único domínio de colisão.

Os repetidores e *hubs* são classificados como dispositivos da camada 1, no modelo de referência OSI da ISO, porque eles atuam apenas no nível físico e não consideram nenhuma outra informação. A função básica é estender o alcance da rede, copiando bits de um trecho para outro. A figura 2.3.1 ilustra as diversas camadas do modelo OSI.

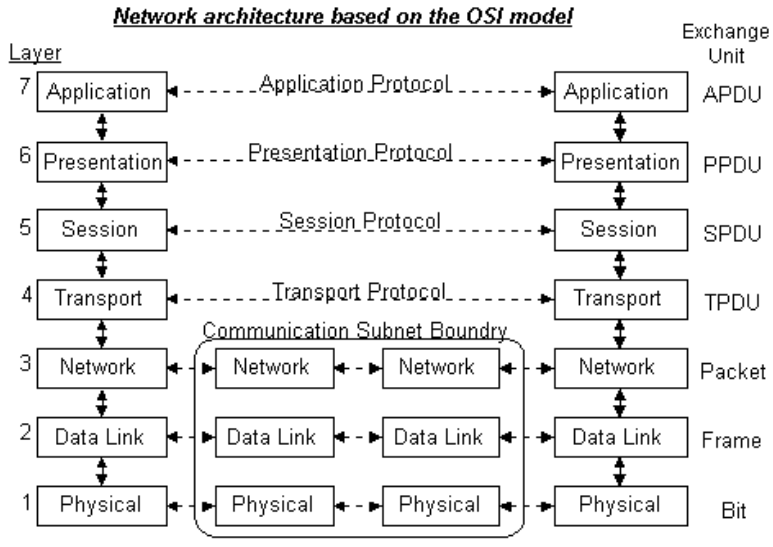


Figura 2.3.1 – Camadas do modelo de referência OSI

2.4. SWITCH ETHERNET: MECANISMO DE COMUTAÇÃO DE PACOTES

A medida em que o tamanho das redes aumentava, e a tecnologia disponível provinha meios, novos e mais complexos dispositivos foram sendo desenvolvidos. Os dispositivos de

comutação de pacotes (switches) são chaveadores de pacotes que operam na camada 2 do modelo de referência OSI/ISO, em nível de quadros Ethernet, tendo como principal função isolar segmentos de rede Ethernet compartilhados, de modo a evitar congestionamentos, operando em diferentes velocidades e controlando o fluxo de tráfego através de um sistema, além de melhorar a confiabilidade e aumentar bastante a largura de banda Ethernet disponível para uso. O aumento da confiabilidade se dá pelo fato de que cada *switch* em uma rede pode operar no modo *store-and*-forward, o que significa que cada pacote recebido (*store)* somente será encaminhado à porta de destino (*forward)* após a verificação de sua integridade através da checagem do código de CRC (*cyclic redundacy check)*. Caso o código de CRC indique erro no pacote de dados, o mesmo é descartado sem o envio de qualquer tipo de confirmação. O aumento da largura de banda efetiva se dá pelo fato de que o tráfego de pacotes entre duas estações conectadas a duas portas não é enviado a nenhuma outra porta, deixando, assim, as demais portas disponíveis para servir o tráfego entre outras estações conectadas a outras portas.

A figura 2.4.1 mostra uma topologia simples onde dois domínios de colisão estão conectados a cada uma das portas de um *switch*.

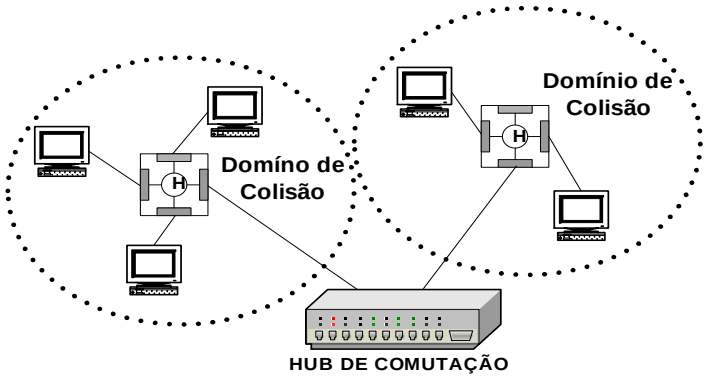


Figura 2.4.1 – Domínios de colisão separados por um *switch*

Fica claro pela 2.4.1 que as estações que não estão dentro de um mesmo domínio dependem do *switch* para se comunicarem. Cabe a ele, de forma rápida e eficiente, tomar a decisão de quais pacotes de dados devem passar de um domínio a outro. Esta inteligência se dá através do aprendizado de endereços e do armazenamento destas informações em uma base de dados para futura consulta no momento do encaminhamento dos pacotes.

A figura 2.4.2 serve para ilustrar o mecanismo de aprendizagem de endereços.

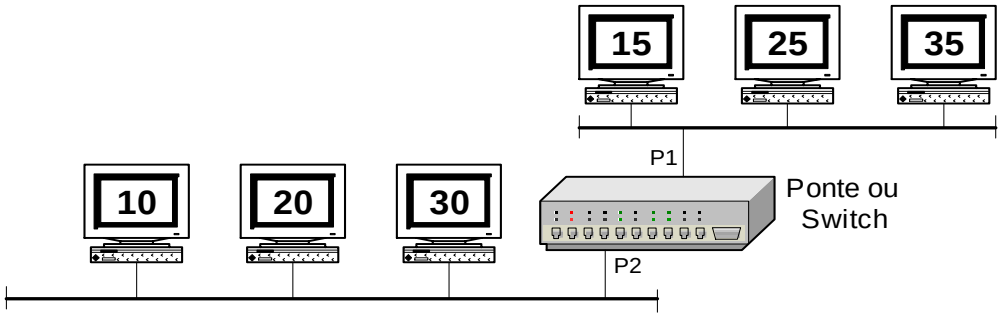


Figura 2.4.2 – Aprendizagem de endereços

Quando determinada estação, por exemplo a de número ‘10’ envia um pacote, o *switch*, ao examinar o endereço de origem, fica sabendo que esta estação está conectado ao segmento de rede da porta P2. Desta forma, se um pacote chegar na porta P2 com endereço de destino sendo a estação ‘10’, este pacote não será encaminhado aos outros segmentos de rede, diminuindo, assim, o congestionamento. Por outro lado, se a estação ‘15’ (conectada à porta P1) enviar um pacote para a estação ‘10’, este será encaminhado somente para a porta P2, e para nenhuma outra porta.

Ao longo do tempo, uma base de dados é desenvolvida contendo: o endereço da estação, a porta do *switch* à qual ela se encontra conectada, um contador de tempo medindo o tempo desde a última atividade detectada desta estação.

Estação – Endereço MAC	Porta	Idade
15	P1	38s
25	P1	15s
35	P1	2s
10	P2	84s
20	P2	7s
30	P2	119s

Figura 2.4.3 – Exemplo de tabela de endereços

O campo ‘idade’ é utilizado pelo mecanismo de envelhecimento. Este mecanismo é importante no sentido que evita o bloqueio do envio de pacotes a uma determinada estação, caso

esta seja deslocada de um segmento para outro. Este mecanismo também evita que posições na base de dados fiquem ocupadas por entradas que não estão sendo utilizadas por determinado período de tempo, liberando, assim, o uso desta posição por outros endereços. Isso se torna mais importante em sistemas que possuem uma base de dados limitada em memória.

2.5. VLAN's (Virtual Local Area Networks)

Através do uso do conceito de VLAN's (*Virtual Local Area Networks*), é possível a criação de várias redes lógicas dentro de uma mesma rede física. Existem várias maneiras possíveis para a criação de LAN's virtuais:

VLAN's baseadas em portas

Neste tipo de topologia, as VLAN's são criadas com o agrupamento das portas do *switch*. Por exemplo, as estações conectadas a uma ou mais portas do equipamento fazem parte de uma mesma VLAN. Através do estabelecimento de regras internas ao equipamento é possível determinar que pacotes originados de uma determinada VLAN podem, ou não, ser encaminhados à outras VLAN's. Assim, formam-se redes virtuais, dentro de uma única rede local. Este tipo de agrupamento é muito usado em *switches* que concentram vários usuários em um único *uplink* e por questões de segurança e privacidade devem ser impedidos de trocar pacotes entre si.

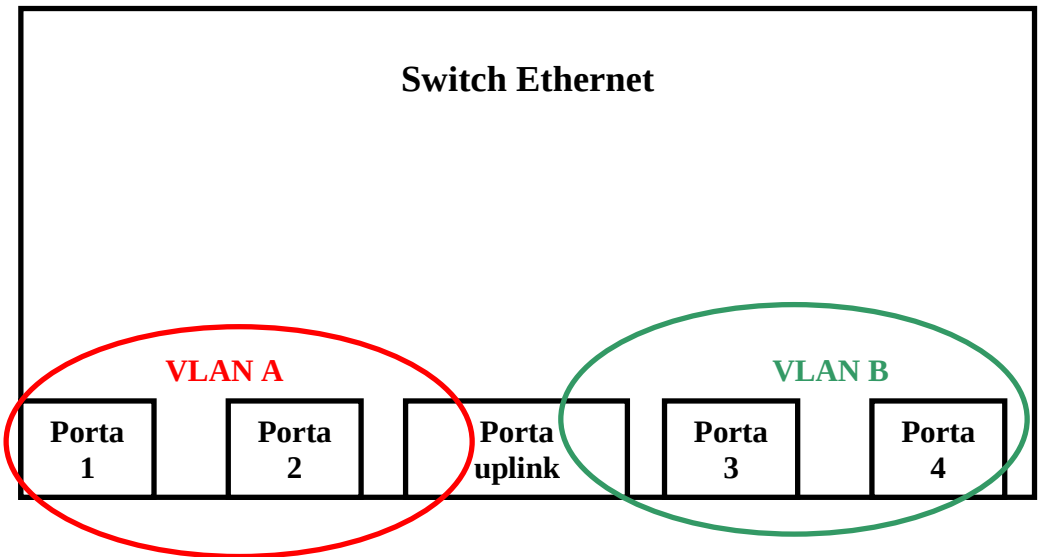


Figura 2.5.1 – VLAN's baseadas em portas

Uma mesma porta pode pertencer a duas VLAN's diferentes, como é o caso da porta *uplink* no exemplo acima. Em geral, a porta de *uplink*, que faz a conexão dos vários segmentos de rede ao mundo externo (ex.: roteador de saída) pertence a todas as VLAN's. No exemplo acima, as portas do *switch* foram agrupadas de acordo com a seguinte distribuição de VLAN´s:

- **VLAN A**: portas 1, 2 e *uplink*
- **VLAN B**: portas 3, 4 e *uplink*

Isso significa dizer que pacotes originados das portas 1 e 2 podem ser comutados para a porta de saída (*uplink*) e seguir para o mundo externo, pois pertencem à mesma VLAN. O mesmo acontece para os pacotes originados das portas 3 e 4. Porém, pacotes originados das portas 1 e 2 não podem ser comutados para as portas 3 e 4, criado uma barreira de segurança pelo *switch*.

Note que neste tipo de redes locais virtuais, os pacotes ethernet que circulam nos diferentes segmentos de rede não sofrem nenhuma alteração. A divisão dos segmentos em redes virtuais é feita internamente ao equipamento.

VLAN's baseadas em endereço MAC

Para a criação de redes virtuais, o equipamento pode fazer a divisão das redes baseado nos próprios endereços de máquina das estações. Desta forma, o *switch* ethernet pode fazer com que apenas um grupo de estações possa trocar pacotes entre si, independente de qual porta estão conectados. Este tipo de rede virtual é mais complexa de ser criada e gerenciada, uma vez que envolve o uso dos endereços de máquina das estações, os quais podem mudar caso a interface de rede seja substituída, como por exemplo, em caso de manutenção.

VLAN's baseadas em protocolos de rede

Neste tipo de mecanismo, as redes virtuais são criadas levando-se em consideração o tipo de protocolo de rede carregado pelo pacote ethernet, por exemplo: IPv4, ARP, IPx.

O IEEE propôs, com a publicação da norma P802.1Q [56], uma extensão ao quadro de dados ethernet até então definido. Nesta extensão quatro octetos são adicionados ao cabeçalho do pacote ethernet, os quais formam o etiqueta (*tag*) de VLAN. A figura 2.5.2 mostra a estrutura do cabeçalho ethernet contento os 4 bytes adicionais relativos à etiqueta de VLAN.

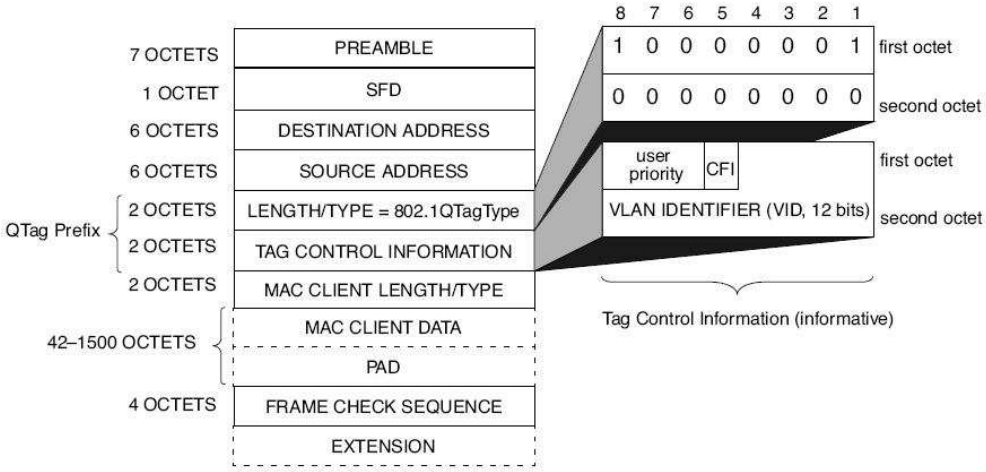


Figura 2.5.2 – Cabeçalho do pacote ethernet contento etiqueta de VLAN

Normalmente, logo após o endereço de origem do pacote, o campo contendo o tipo de pacote encapsulado pelo pacote ethernet ou o tamanho do próprio pacote é enviado (campo *type/length*). Quando um pacote Ethernet contém uma etiqueta de VLAN, o conteúdo do campo *type/length* é preenchido com o valor reservado 8100h, de forma a identificá-lo como um pacote contendo informação de redes locais virtuais. Em seguida, mais dois octetos contendo as informações da VLAN propriamente dita são recebidos e só então o campo *type/length* original do pacote Ethernet é colocado.

Os dois bytes referentes ao TCI (*tag control information*) são formados por três campos, que são descritos a seguir:

- ***user priority***: este campo tem um tamanho de 3 bits, e deve ser interpretado como um número binário. A prioridade do usuário, então representada por este número, é dividida em 8 níveis de prioridade, 0 a 7. De acordo com a prioridade de usuário, os switches Ethernet podem dar preferência a determinados pacotes em relação a outros em situações de tráfego intenso, por exemplo. O uso e a interpretação deste campo estão especificadas em detalhe na IEEE Std 802.1D, 1998 Edition.
- ***Canonical Format Indicator (CFI)***: em quadros Ethernet, este bit indica a presença ou ausência do campo E-RIF (*embedded RIF*) no quadro com a etiqueta de VLAN. Usualmente em redes Ethernet este bit está na condição *off* , indicando que o campo E-RIF não está presente no quadro Ethernet.

- ***VLAN identifier (VID)****:* este campo de 12 bits identifica a qual VLAN o pacote pertence. Alguns valores de VID são reservados, como 0, 1 e FFFh. Sendo que com os 12 bits do campo é possível a criação de 4093 VLAN's diferentes.

2.6. STP (SPANNING TREE PROTOCOL)

Normalmente em uma rede ethernet local, duas estações estão logicamente interligadas entre si através de um caminho único. Em tempos passados, este tipo de situação era mandatório sob pena de ocorrerem *loops* na rede, bem como atrapalhar o mecanismo de encaminhamento de pacotes dos switches ethernet conectados à rede, uma vez que os mesmos poderiam detectar o mesmo endereço de máquina chegando em duas portas diferentes, o que normalmente não é possível, pois tais endereços são únicos.

Porém, em alguns casos, podem haver duas *bridges* interligando segmentos da rede local. A figura 2.6.1 ilustra esta situação.

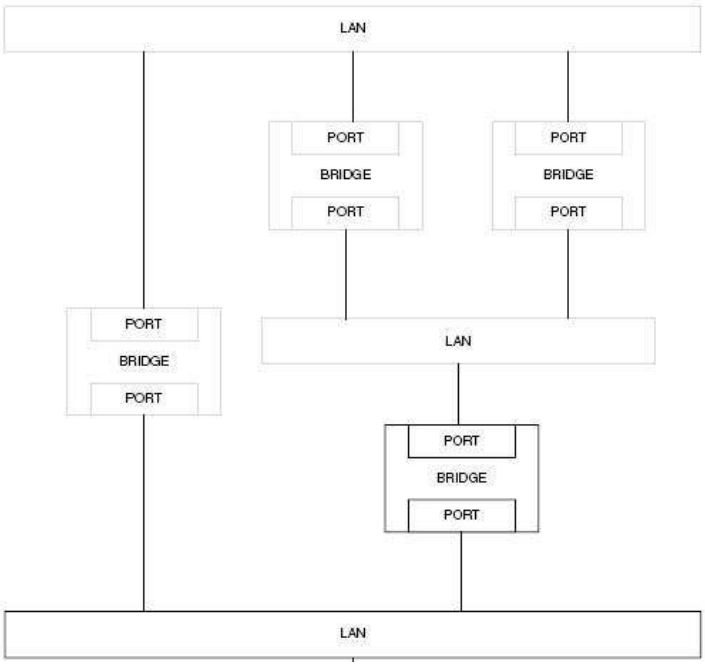


Figura 2.6.1 - Rede local interligada por bridges

Pode-se claramente notar pela figura 2.6.1 que os segmentos de rede estão ligados entre si por vários caminhos, três no caso exemplificado. A vantagem deste tipo de ligação reside no fato

de criarem-se caminhos alternativos entre os segmentos. Esta redundância torna-se útil quando ocorre a falha de um *switch* na rede. Assim, os pacotes podem ser encaminhados pelo *switch* que está no caminho redundante.

Para que uma rede ethernet funcione corretamente, só deve existir um caminho que conecte duas estações. Para resolver o problema dos *loops* causados na situação da figura 2.6.1 e ainda utilizar a vantagem da redundância dos caminhos, o IEEE propôs o *Spanning Tree Protocol and Algorithm* (STP), definido em IEEE 802.1D – MAC bridges [22]. O protocolo consiste em eleger um *switch* denominado "raiz", a partir do qual serão traçados todos os caminhos possíveis da rede. O protocolo força todos os caminhos redundantes a um estado bloqueado. Uma vez estabelecidos os caminhos que devem ser bloqueados, quando um segmento de rede tornar-se inacessível ou a topologia da rede mudar devido à inserção de novos caminhos, o protocolo automaticamente reconfigura a topologia da "árvore" de forma a tornar o segmento acessível através da ativação de um caminho antes bloqueado.

Uma questão importante a ressaltar é que o protocolo é totalmente transparente para as estações conectadas na rede.

O algoritmo está dividido em algumas etapas principais. A primeira etapa é a escolha do *root switch,* que será o centro da rede. A definição deste *switch* é baseada no endereço de máquina do mesmo (aquele que tem o menor endereço MAC é eleito o *root switch*) e em um número de prioridade, que pode ser atribuído a cada *switch* em particular pelo gerenciador da rede de forma a dar preferência a um determinado *switch* que, por questões de tráfego, número de portas ou velocidade das portas, possa ser o mais adequado a se tornar o *root switch.*

Todos os outros switches da rede irão então calcular o custo que cada porta tem até chegar ao *root switch.* A porta que tiver o menor custo é então escolhida como a porta de trabalho sendo que as outras são desabilitadas. Da mesma forma como acontece na escolha do *switch* principal, a escolha da porta também pode ser baseada em uma prioridade. Por exemplo, pode-se atribuir maior prioridade a portas com maior velocidade, pois nem sempre a porta que possui o menor custo é aquela mais adequada a ser a porta ativa do *switch.*

Durante a execução do algoritmo *Spanning Tree,* as portas dos equipamentos podem estar em cinco estados diferentes, são eles: *disabled, blocking, listening, learning, forwarding.* As portas podem migrar de estado em estado durante a execução do algoritmo. A passagem de estados segue os seguintes caminhos:

- na inicialização o estado inicial é *blocking*
- de *blocking* para *listening* ou *disabled*
- de *listening* para *learning* ou *disabled*
- de *learning* para *forwarding* ou *disabled*
- de *forwarding* para *disabled*

A comunicação entre os switches da rede que participam do protocolo é feita através de pacotes especiais denominados BPDU's (*Bridge Protocol Data Units*). As BPDU's não são encaminhadas entre as portas como pacotes normais são encaminhados, mas a informação contida neles é usada pelo *switch* para "aprender" sobre a topologia da rede a também gerar BPDU's para os *switches* adjacentes.

O protocolo *Spanning Tree* foi definido em uma época em que o tempo de recuperação menor que um minuto após uma interrupção no serviço era considerado uma performance adequada [23]. Uma evolução do STP original é o *Rapid Spanning Tree Protocol*, originalmente definido em IEEE 802.1w e que mais tarde, mais especificamente na revisão de 2004, foi incorporada a IEEE 802.1D. O RSTP foi desenvolvido para ser totalmente compatível com o STP. As portas de um *switch* que implementam o RSTP se ajustam automaticamente para prover interoperabilidade caso elas estejam conectadas a portas de outro equipamento que implementa apenas o STP.

No RSTP, os cinco estados possíveis que uma porta poderia se encontrar foram reduzidos a apenas 3. A tabela 2.6.1 mostra a inter-relação dos estados das portas no STP e no RSTP.

Tabela 2.6.1 – Estados do STP e do RSTP

Estado da porta no STP (802.1D)	**Estado da porta no RSTP (802.1w)**
Disabled	Discarding
Blocking	Discarding
Listening	Discarding
Learning	Learning
Forwarding	*Forwarding*

2.7. ESTADO DA ARTE EM SWITCHES ETHERNET

2.7.1 Cisco Systems

No mercado de equipamentos para redes de computadores, o destaque principal é sem dúvida a CISCO Systems [24], fabricante de equipamentos e também desenvolvedor de novas tecnologias na área, tais como a definição de novos protocolos e aprimoramento de protocolos já utilizados. A CISCO foi fundada no ano de 1984, por um grupo de cientistas da computação da Universidade de Stanford

Na área de switches ethernet a CISCO possui equipamentos das mais variadas capacidades, contemplando o mercado SOHO (*Small Office Home Office*) até as grandes operadoras que visam oferecer serviços na emergente tecnologia de Metro Ethernet. A CISCO é membro participante do Metro Ethernet Forum [25], um grupo formado por mais de 90 empresas, operadoras e fabricantes de componentes e equipamentos, com o objetivo de acelerar a adoção das redes ethernet como convergência no transporte de dados.

A menor família de switches ethernet da CISCO é a série Catalyst Express 500, que oferece equipamentos de 8 a 24 portas, como mostra a tabela 2.7.1.1:

Tabela 2.7.1.1 – Família Cisco Catalyst Express 500

Nome do produto	Descrição
Cisco Catalyst Express 500-24TT Switch (WS-CE500-24TT)	• 24 portas 10/100 para conectividade de desktops • 2 portas 10/100/1000BASE-T para uplink ou conectividade de servidores
Cisco Catalyst Express 500-24LC Switch (WS-CE500-24LC)	• 20 portas 10/100 para conectividade de desktops • 4 portas 10/100 PoE para desktop, pontos de acesso wireless, telefonia IP ou circuito fechado de TV • 2 portas 10/100/1000BASE-T ou Small Form-Factor Pluggable (SFP) para conexão flexível com servidores ou uplink
Cisco Catalyst Express 500-24PC Switch (WS-CE500-24PC)	• 24 portas 10/100 PoE para desktop, wireless, telefonia IP ou circuito fechado de TV • 2 portas 10/100/1000BASE-T ou para conexão flexível com servidores ou uplink
Cisco Catalyst Express 500G-12TC Switch (WS-CE500G-12TC)	• 8 portas 10/100/1000BASE-T e 4 portas 10/100/1000BASE-T ou SFP para conexão flexível com servidores ou uplink

Na linha de produtos de maior porte (*core switches*) a CISCO apresenta a linha Catalyst 6500. Nestes tipos de equipamentos, são usados chassis (sub-bastidores) com um determinado número de *slots* onde podem ser inseridas placas de acordo com a demanda. A linha oferece chassis com até 13 *slots*. O *backplane* tem capacidade para até 720 Gbps através do uso de *switch-fabric.*

Com o uso de placas com maior densidade, o número de portas 10/100Mbps pode chegar até 1152. No caso de portas 10/100/1000, o número máximo é de 577. A tabela 2.7.1.2 mostra a capacidade máxima de portas de cada equipamento da série 6500.

Tabela 2.7.1.2 – Capacidade de portas da família 6500

Número de Portas	Catalyst 6503	Catalyst 6503-E	Catalyst 6506 e 6506-E	Catalyst 6509 e 6509-E	Catalyst 6509-NEB e 6509-NEB-A	Catalyst 6513
10 Gigabit Ethernet	2	8	20	32	32	20
Gigabit Ethernet (SFP optics)	8	98	242	386	384	410
Gigabit Ethernet (GBIC)	34	43	82	130	130	194
10/100/1000 Ethernet	97	97	241	385	385	577
10/100 Fast Ethernet	192	192	480	768	768	1152
100BASE-FX	96	96	240	384	384	576
FlexWAN (DS0 a OC3)	4 POS 2 DPT	4 POS 2 DPT	10 POS 5 DPT	16 POS 8 DPT	16 POS 8 DPT	24 POS 12 DPT

2.7.2 Broadcom Corporation

No mercado de fabricação de componentes eletrônicos para switches ethernet, o destaque principal fica com a americana Broadcom Corporation [26]. Empresa fundada em 1991, é uma das líderes de mercado na fabricação de componentes para comunicação com e sem fio (*wireless*).

Na linha de produtos da Broadcom, destaca-se o BCM53718, um *switch* multi-layer com 48 portas Gigabit Ethernet. Assim como a maioria dos produtos finais oferecidos pela Cisco, a tabela de endereços MAC tem capacidade para armazenamento de 8000 endereços. A figura 2.7.2.1, retirada do documento de descrição do produto, mostra o diagrama em blocos do dispositivo.

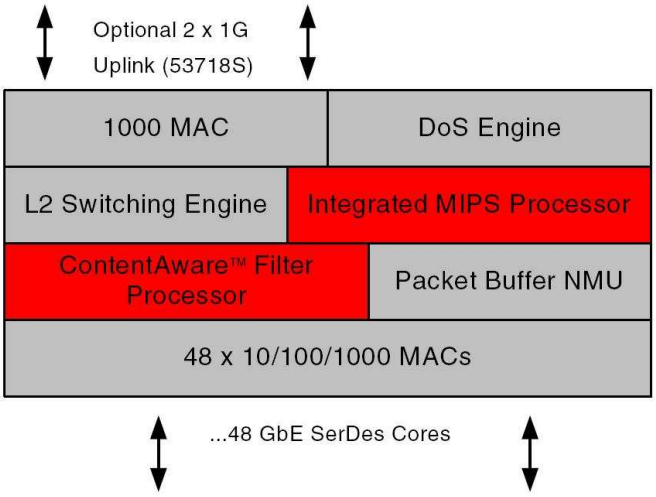


Figura 2.7.2.1 – Diagrama em blocos do BCM53718

Algumas das principais características do dispositivo:

- duas portas opcionais Gigabit ethernet para uplink
- Controladores MAC integrados com suporte a pacotes *jumbo* de 9216 bytes
- Interfaces SGMII/SerDes integradas
- *Switch fabric* integrada não bloqueante com buffer de 4M bits
- Processador MIPS de alta performance para aplicações web
- Classificação dos pacotes usando 4 filas de prioridade segundo IEEE 802.1p
- Suporte a 4K VLAN’s
- Suporte a 8K endereços MAC unicast
- Controle de fluxo half e full duplex (IEEE 802.3x)
- Suporte a Spanning Tree e Rapid Spanning Tree
- Controle de taxa por porta com resolução de 1 Mbps
- *Core* de 0,13um de baixo consumo

2.8 CONCLUSÕES

Nos dias de hoje vemos as redes Ethernet sendo largamente utilizadas. Uma das questões chave para a boa aceitação deste tipo de tecnologia, tanto pelos fabricantes de equipamentos quanto pelos usuários, é a simplicidade do protocolo e o baixo custo de produção e aquisição dos equipamentos.

O mecanismo de comutação de pacotes desempenhado pelos *switches* Ethernet, embora seja baseado em um conceito simples, pode ser incrementado com a adição de novos protocolos que facilitam a operação e a manutenção das redes Ethernet, tais como o protocolo *Spanning Tree*.

3. BRIDGES EM SOFTWARE

Neste capítulo, iremos abordar algumas soluções de *bridges* ethernet implementadas em software.

3.1. IMPLEMENTAÇÃO DE BRIDGE ETHERNET EM PLATAFORMA LINUX

Com o advento do software livre, a utilização do *kernel* do Linux como ferramenta para o desenvolvimento de equipamentos para interligação de redes de computadores tornou-se muito comum. Até mesmo em sistemas embarcados, onde se faz uso de microprocessadores dedicados, graças ao trabalho de equipes em todo o mundo no intuito de realizar o porte do sistema operacional para várias plataformas. Uma vasta gama de equipamentos comerciais hoje em dia, tais como roteadores, *firewalls* e modens, utilizam o *kernel* do Linux como plataforma.

Em geral, a utilização do *kernel* está voltada para o tratamento de pacotes IP. Porém, existe a possibilidade de se utilizar o *kernel* do linux para encaminhar pacotes como se fosse um *switch*. Esta função pode ser implementada utilizando-se uma ou mais interfaces de rede.

O apelo principal para a utilização do Linux como *bridge* é ao mesmo tempo podemos habilitar as funções de segurança já conhecidas do Linux, como as funções de *firewall*. Uma topologia interessante é colocar uma estação executando Linux com função de *bridge* junto a um servidor em uma rede de computadores. Desta forma, podemos adicionar segurança a este servidor sem a necessidade de reconfigurar o mesmo, o que muitas vezes pode ser custoso e delicado, como por exemplo, em casos onde não se tem privilégios administrativos do servidor.

Como desvantagem desta implementação, temos o alto custo de uma estação de trabalho aliado ao baixo desempenho que uma *bridge* em software tem na comutação de pacotes.

Na implementação do Linux, não existe a necessidade de se dar um endereço IP à *bridge*, visto que o encaminhamento de pacotes é feito através do endereço MAC e não do endereço IP. Porém, para se obter conectividade com a estação que implementa a *bridge* de forma a reconfigurá-la, obter relatórios ou até mesmo executar programas de diagnóstico da rede como *sniffers*, o Linux permite dar um endereço IP à *bridge*.

A figura 3.1.1 mostra de maneira simplificada a conexão de um laptop a uma rede de computadores através de uma estação Linux executando código de *bridge*.

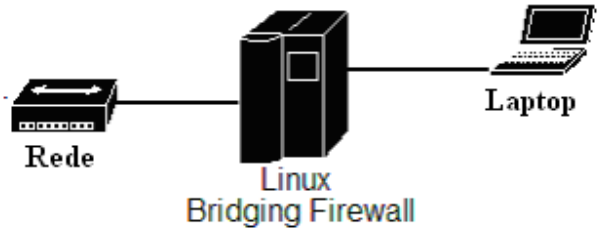


Figura 3.1.1 - Kernel do Linux em modo *bridge*

Normalmente, na maioria das distribuições, as funções de *bridge* do *kernel* do Linux são escritas no pacote *bridge*-utils. Jim Robinson mostra em detalhe como configurar o *kernel* do Linux [20] para executar a função de *bridge* aliada à filtragem de pacotes com *firewall*. De uma forma bem simplificada, a configuração do *kernel* como *bridge* consiste em criar uma "entidade" *bridge* através da linha de comando e em seguida adicionar as duas ou mais interfaces de rede à esta *bridge*. Como mostra a seqüência de comandos abaixo:

```
#> brctl addbr br0
#> brctl addif br0 eth0
#> brctl addif br0 eth1
#> ip link set br0 up
```

Para implementar as funções de *firewall* na *bridge* criada é necessária a utilização de um pacote de software que roda no espaço de usuário do Linux. Este pacote é o *ebtables* [27] distribuído gratuitamente. A sintaxe para uso do *ebtables* é bastante similar ao já bastante difundido *iptables*. Como todas as funções são implementadas em software, existe também o suporte ao STP (*Spanning Tree Protocol*). Além disso, a adequação a novos protocolos e tecnologias é bastante viável e simples.

3.2. ESTUDO DA BRIDGE DE KHAN

O projeto descrito por Amir A. Khan e outros em [13] propõe-se a implementar uma *bridge* em software através do uso de um PC e duas interfaces de rede. A implementação é feita

sob o sistema operacional DOS. Além das funções de comutação de pacotes, o projeto também propõe-se a coletar informação de estatísticas e implementar a filtragem de pacotes.

Em seu artigo, os autores discursaram sobre as principais questões envolvidas no projeto:

- definição de *drivers* a serem utilizados de forma a executar a interface entre o software e o hardware;
- gerenciamento dos *buffers* de dados, algoritmo de aprendizagem e consulta a base dados;
- filtragem de pacotes baseado no conteúdo dos mesmos (*firewall*) e coleta e análise de estatísticas (monitoração da rede)

Drivers de rede

Originalmente, o DOS não provê suporte à interfaces de rede. Assim, de forma a utilizar a plataforma DOS, o pacote de software '*packet drivers*' é utilizado [8] [6]. As principais razões para a utilização deste pacote são que ele permite a captura de todo o quadro ethernet mantendo a informação dos cabeçalhos intacta. O acesso a todo o quadro faz com que a tarefa de filtragem de pacotes seja mais fácil e eficiente. Além disso, estes *drivers* são de domínio público e as informações são disponibilizadas gratuitamente.

Tabela de endereços MAC

Deve-se escolher entre duas formas de acesso à base de dados (tabela de endereços MAC) são elas: indexação ou *hashing*.Qualquer tipo de indexação é sempre mais lenta pela sua natureza indireta. A função de *hash* escolhida deve ser eficiente no que se refere ao número de entradas na tabela e também fácil de ser computada, de forma a não degradar demasiadamente o desempenho da *bridge*.

Três mecanismos de *hashing* dos endereços foram testados:

1. hashing nos 8 bits menos significativos do endereço ethernet resultando em um total de 256 posições possíveis na tabela de endereços;
2. hashing nos 16 bits menos significativos do endereço ethernet resultando em um total de 65536 (64K) posições possíveis na tabela de endereços;
3. hashing na soma de todos os bytes do endereço ethernet resultando em um total de 1536 posições possíveis na tabela de endereços;

Das opções acima, segundo os autores, as duas últimas tiveram desempenho satisfatório na rede testada, a qual possuía em torno de 150 estações.

O autor afirma que nos para a avaliação de desempenho dos mecanismos de *hashing* propostos acima em casos práticos, o critério adotado foi o número de pacotes que foram encaminhados de uma porta a outra mas que na verdade não deveriam ser encaminhados. Dado o número de estações presentes na rede de teste e o número de posições da tabela *hash* em cada situação testada (sempre maior que o número de estações), podemos observar que o não foi implementado nenhum mecanismo de tratamento de colisões nesta esquema de *hashing*. Assim, quando uma colisão acontece, ou seja, a posição consultada na tabela já esta sendo ocupada por um outro endereço MAC, o pacote que está sendo tratado é enviado à outra porta de qualquer forma. Mais adiante iremos entrar em detalhe nos mecanismos de tabelas *hash* e o problema do tratamento de colisão nestas estruturas. É importante ressaltar que, dependendo da performance dos processos utilizados no equipamento, é possível que o custo de tratar uma colisão seja tão grande que o envio do pacote às demais portas por inundação ainda seja a melhor opção no caso de colisões. Em switches ethernet, a informação de a qual porta uma determinada estação está conectada não torna inviável a operação do equipamento, porém, o uso de tabelas *hash* não se resume a esse tipo de aplicação e em outros casos o descarte de uma entrada na tabela pode ser desastroso.

Comutação de pacotes

Como existem somente duas portas na *bridge*, de forma a facilitar o acesso a base de dados e evitar o compartilhamento da mesma, duas tabelas de endereços foram criadas, uma para cada porta. Desta forma, quando um pacote é recebido em uma determinada porta, apenas a tabela de endereços referente a esta porta necessita ser consultada. Isto facilita a implementação em ambientes *multitask* ou acionados por interrupção, onde existe o risco de dois processos estarem acessando a mesma base de dados. O mecanismo funciona da seguinte maneira: quando um pacote é recebido, a tabela de endereços da porta em questão é consultada, se o endereço de destino do pacote pertence a esta tabela (indicando que a estação está conectada a esta porta) então o pacote é descartado, caso contrário, o pacote é enviado para a outra porta. Além disso, a cada novo pacote que é recebido, o endereço de origem do mesmo é armazenado na tabela de endereços de forma a ser consultado no próximo pacote recebido.

A *bridge* proposta também coloca na tabela de endereços um controle visando o mecanismo de envelhecimento do endereço. Periodicamente a tabela é consultada de forma a excluir os endereços que não tenham atividade por um determinado intervalo de tempo. O tempo de envelhecimento pode ser configurado por software e em geral fica fixado em 10 minutos.

Filtragem de pacotes

A *bridge* proposta pode filtrar pacotes IP em função do endereço de destino, ou seja, tem a capacidade de analisar até o nível 3 do modelo OSI. A *bridge* supõe que o formato dos pacotes segue o padrão DIX (Digital, Intel, Xerox) também conhecido como Ethernet II. A forma de entrada dos endereços a serem filtrados não é muito prática e é feita através da leitura de um arquivo armazenado na máquina.

Coletor de estatísticas

A coleta de estatísticas do nível físico é feita diretamente através da leitura dos registros da placa de rede, enquanto que as estatísticas geradas pela filtragem de pacotes é feita pelo próprio software desenvolvido.

Todas as estatísticas são encapsuladas em um pacote UDP e enviadas periodicamente. O endereço IP e a portas de destino dos pacotes de estatísticas são configuráveis sendo que o conteúdo dos pacotes é proprietário. O programa cliente que recebe as estatísticas apenas mostra na tela as mesmas à medida em que es pacotes provenientes da *bridge* são recebidos.

3.3 BRIDGE EM SOFTWARE DIDÁTICA

Para fins didáticos, foi desenvolvida uma *bridge* implementada em software a qual gerou o trabalho publicado em [18]. Esta *bridge* consiste na utilização de um PC executando a plataforma Linux com duas ou mais interfaces de rede Ethernet. Em nível de usuário, um software foi desenvolvido de forma a capturar os pacotes ethernet provenientes de uma das interfaces e em seguida tomar a decisão para qual porta o mesmo deveria ser encaminhado.

Duas das principais razões para o uso da plataforma Linux foram:

- é comum na comunidade do software livre a criação de listas de discussão onde usuários dos mais diferentes meios (acadêmicos ou profissionais) trocam conhecimentos,

dificuldades e expõe novas idéias em torno de um determinado pacote de software, quer seja ele um aplicativo, um *device driver*, ou até mesmo o código fonte de um sistema operacional;

- através do uso de software livre é possível, para qualquer desenvolvedor, aplicar modificações ao código fonte de um determinado pacote de forma a este suprir suas necessidades específicas;

O código fonte foi totalmente desenvolvido em linguagem C. O compilador utilizado foi o ´gcc´, que é amplamente divulgado e faz parte das mais variadas distribuições do Linux, incluindo o Mandrake e o Conectiva, distribuições mais utilizadas durante o desenvolvimento.

Dentre as ferramentas utilizadas, destacam-se duas bibliotecas que desempenham um papel decisivo na implementação. São elas: a LIBPCAP [29] e a LIBNET [28]. A primeira, no ambiente Linux, dedica-se exclusivamente à captura de pacotes das interfaces de rede do sistema. A segunda, LIBNET, dedica-se à construção e envio de pacotes através destas interfaces.

Através do uso destas bibliotecas, foi possível a criação de um software de nível de usuário que capturasse os pacotes Ethernet de forma bem simplificada e clara, em seguida os enviando à interface apropriada.

O hardware da implementação consiste em um PC, com duas ou mais interfaces de rede do tipo Ethernet, cada uma conectada a diferentes segmentos de rede. Como só estamos preocupados com o nível de enlace, o protocolo que está sendo transportado através dos pacotes Ethernet é irrelevante e não é analisado.

Podemos dividir o código fonte em três partes distintas:

- a recepção, validação e análise dos pacotes;
- a transmissão dos pacotes para a interface ou interfaces apropriadas de acordo com o resultado da análise prévia;
- o gerenciamento da tabela de endereços;

Quando o programa é iniciado, este inicia uma *thread* para o gerenciamento de cada interface de rede do *switch*. *Threads* em Linux são semelhantes a processos e entram dentro do escalonamento de tarefas do sistema operacional. Diferentemente de processos, a memória é compartilhada entre todas as *threads* iniciadas por um determinado processo. Assim, é necessário

que se tome cuidado com a acesso a bases de dados, como a tabela de endereços MAC. Tal gerenciamento pode ser facilmente implementado através do uso de MUTEX, que de certa forma funcionam como semáforos.

Cada *thread* iniciada será responsável por desempenhar as duas primeiras partes do software. Além destas ´n´ *threads*, também é iniciada uma que trata do gerenciamento da tabela de endereços, sendo sua principal função a de apagar as entradas que estão inativas, sem enviar pacotes, a um tempo pré-determinado. Este mecanismo denomina-se envelhecimento, e faz com que os endereços que geram tráfego com maior freqüência fiquem mais tempo armazenados na tabela, diminuindo a probabilidade de geração de tráfego adicional pelo *switch*.

A tabela de endereços MAC é o centro de informações do software. Ela é na verdade um lista de estruturas alocada estaticamente que contém: um *array* de bytes que representa um endereço Ethernet, um byte identificando a qual das portas do *switch* este endereço pertence, um byte identificando se esta entrada da tabela esta ocupada ou vaga e, por fim, um inteiro que informa o tempo decorrido desde o último pacote enviado por este endereço.

Através da consulta à tabela de endereços é tomada a decisão se um pacote será descartado, enviado a apenas uma das portas ou enviado a todas as portas por broadcast ou por inundação. O armazenamento e consulta à tabela de endereços foi implementado de forma linear, sem o uso de tabelas *hash*, que serão abordadas mais adiante. Deste modo, quanto maior a utilização da tabela de endereços, pior será a performance da *bridge*, visto que mais posições da tabela terão de ser consultadas até que o endereço em questão seja encontrado.

Na parte da recepção de pacotes, quando a *thread* responsável pela captura de pacotes de uma determinada interface é iniciada, esta trata de inicializar o descritor de captura da biblioteca através da função *pcap_open_live()*. A seguir, com o uso da função *pcap_loop()* indicamos à biblioteca, qual função deve ser chamada cada vez que um pacote Ethernet for recebido ou enviado pela interface em questão. Este tipo de função denomina-se função *call back.* Tal função, que é escrita pelo usuário, quando chamada, recebe como parâmetros, dentre outros, um ponteiro para uma estrutura de dados com informações sobre o pacote capturado bem como um ponteiro para o próprio pacote.

O código fonte original da LIBPCAP, quando capturando pacotes Ethernet, não provê informação ao usuário sobre a direção do pacote, ou seja, não é possível distinguir se o pacote capturado foi recebido por determinada interface ou se foi enviado pela mesma. Na sua

implementação para a plataforma Linux, é possível agregar tal funcionalidade através da aplicação de um *patch* de modificações simples. Porém, em outras plataformas esta alteração não é tão trivial. Como a implementação em questão tem um objetivo puramente didático, e na intenção de a tornar portável a outras plataformas, foi decidido contornar esta limitação da *pcap* de uma forma diferente.

Quando um pacote é recebido, primeiro sua consistência é verificada, por exemplo: o endereço de origem, este não pode ser do tipo *multicast* nem *broadcast*. Após parte-se para a análise dos endereços propriamente dita. Nesta etapa, basicamente o que é feito consiste em varrer-se a tabela de endereços à procura dos endereços de origem e destino. Nesta implementação, de forma a garantir um mínimo de desempenho, a tabela é varrida apenas uma vez, procurando-se pelos endereços de origem e destino ao mesmo tempo. A informação sobre o endereço de destino será utilizada mais adiante na etapa de envio do pacote.

Supondo que estamos no início da execução do programa e a tabela de endereços está totalmente vazia. Do ponto de vista da recepção, quando varre-se a tabela à procura do endereço de origem, somente uma situação pode ocorrer: o endereço não pertence à tabela, logo adiciona-o à tabela como sendo pertencente à esta porta e informa a parte de transmissão a qual porta o pacote deve ser enviado, neste caso, à todas as portas. A partir daí, três situações podem ocorrer quando um pacote é recebido:

a) o endereço de origem não consta na tabela. Neste caso, adiciona-o na tabela como sendo pertencente a esta porta.

b) o endereço de destino consta na tabela e está associado à esta porta. Neste caso, apenas atualiza o contador de envelhecimento do endereço.

c) o endereço de destino consta na tabela mas está associado à outra porta.

A análise da situação ´c´ é o que permite resolver o problema de limitação da *pcap* em não informar a direção do pacote. Se o pacote capturado tem o endereço de origem associado a outra porta, isto significa que a *thread* responsável pela mesma é que está enviando o pacote, portanto, este deve ser ignorado.

Para que este mecanismo funcione, é necessário que apenas uma *thread* esteja manipulando a tabela de endereços por vez. Para garantir esta premissa, foi necessário o uso de *mutexes* para a proteção do acesso à tabela.

Na transmissão de pacotes, ao varrer-se a tabela à procura do endereço de origem, o endereço de destino também é analisado. Diferentemente da recepção, quando na transmissão, quatro situações distintas podem ocorrer:

a) o endereço não existe na tabela. Neste caso, envia-se o pacote para todas as outras portas.

b) o endereço de destino é *broadcast* ou *multicast*. Envia o pacote para todas as outras portas.

c) o endereço de destino existe, mas está associado a esta porta. Nesta caso, não faz nada.

d) o endereço existe na tabela, e está associado a uma outra porta. Neste caso, encaminha o pacote apenas para a porta ao qual o endereço está associado.

Cada vez que um pacote vai ser enviado, uma sessão de envio da LIBNET é criada através da função *libnet_init()*. Esta biblioteca permite variados tipos de injeção de pacotes na rede, mas visto que apenas queremos replicar o pacote recebido sem manipulação, utilizamos o tipo de injeção LIBNET_LINK_ADV. Após iniciada a sessão, a função *libnet_adv_write_link()* é chamada para o envio do pacote. Esta função recebe como parâmetros, dentre outros, um ponteiro para o pacote a ser enviado bem como o tamanho do pacote.

De forma a proporcionar um nível mínimo de controle ao usuário do software, uma pequena linha de comando com foi adicionada. Através de comandos digitados é possível visualizar a tabela de endereços MAC, inserir endereços estáticos na tabela, além de criar regras para a filtragem de endereços.

3.4 CONCLUSÕES

Em aplicações onde a performance do encaminhamento de pacotes não representa um grande problema, como em interfaces WAN de baixa velocidade, o uso de *bridges* em software se torna viável. O próprio kernel do Linux provê uma maneira fácil de configurar um sistema, que possua duas ou mais interfaces de rede, como uma *bridge* Ethernet. Também é possível, através do uso de bibliotecas de software livre, implementar sua própria bridge em software, tendo assim, liberdade para tornar o dispositivo mais otimizado.

4 SINGLE PORT BRIDGES COMERCIAIS EM HARDWARE

Atualmente no mercado existem dois principais circuitos integrados que realizam a função de *bridge* necessária para a aplicação proposta neste trabalho. São eles: RJ-017 da Israelense RAD e o ADM6993X da antiga ADMTek, empresa de Taiwan que foi recentemente incorporada pela alemã Infineon.

4.1. RAD RJ-017

"Criada em 1981, a RAD Data Communications é reconhecida internacionalmente como uma das líderes de mercado na fabricação de equipamentos para aplicações de acesso a redes, comunicação de dados e telecomunicações."

4.1.1. Descrição funcional

As características implementadas pela RAD em seu chip, com ênfase no encapsulamento simples e direto dos pacotes Ethernet sobre o protocolo HDLC, formaram uma solução muito eficiente para o transporte de dados de redes de pacotes a grandes distâncias através de modens, utilizando a infra-estrutura das redes determinísticas vastamente presentes no país.

Tal solução foi tão bem sucedida que acabou se tornado um "padrão de fato", sendo que outras empresas, com a ADMTek, desenvolveram seus próprios circuitos integradas com o aberto intuito de serem interoperáveis com o protocolo proposto anteriormente pela RAD.

Abaixo temos uma descrição do dispositivo da RAD, diretamente retirada do manual do componente.

"O ChipBridge é um ASIC altamente integrado que combina subsistemas LAN e WAN para implementar uma *bridge* Ethernet remota completa em um único circuito integrado. Ele aprende automaticamente os endereços MAC da LAN a qual está conectado e encaminha somente os pacotes destinados a outras LAN's. Ele contempla completamente a norma IEEE 802.3. A interface LAN do ChipBridge incorpora um codificador/decodificador Manchester que permite a operação nos modos *half-duplex* e *full-duplex*. Além disso, todos os sinais da LAN estão disponíveis na forma NRZ decodificada para a conexão com outros dispositivos, tais como

repetidores multi-porta ou controladores MAC. No total, 25 modos de operação podem ser selecionados através dos pinos de entrada LMODE.

O subsistema WAN contém um controlador HDLC síncrono e assíncrono, capaz de operar a até 40M bits/s no modo síncrono e 115.2k bits/s no modo assíncrono. No modo assíncrono, uma fonte externa de relógio pode ser usada ou o gerador interno pode ser configurado para gerar freqüências entre 9600 bits/s e 115.2k bits/s.

A tabela de endereços MAC do ChipBridge pode armazenar até 10.000 endereços e é automaticamente atualizada. O mecanismo de envelhecimento automaticamente retira da tabela os endereços cuja estação não teve atividade nos últimos 5 minutos. A filtragem e o encaminhamento dos pacotes é realizada a uma taxa máxima teórica de 15.000 pacotes por segundo. O buffer interno do ChipBridge pode acomodar até 256 pacotes, com uma latência de 1 pacote. A filtragem pode ser desabilitada para aplicações em que isso seja necessário.

O ChipBridge opera independentemente, sem a necessidade de um *host*. Os únicos componentes externos necessários são uma memória DRAM de 256k x16 bits, um cristal oscilador e alguns resistores e capacitores. O ChipBridge contém um controlador de memória DRAM que permite a conexão direta a memórias deste tipo. O ChipBridge é fabricado com um processo de baixo consumo de 0.6 microns, com tecnologia CMOS, e está disponível no encapsulamento tipo PQFP de 100 pinos".

4.1.2 Características

Abaixo temos listadas as principais características do ChipBridge".

- Bridge Ethernet de alta performance em um só chip, que contempla totalmente a IEEE 802.3
- Interfaces AUI (DTE ou DCE) ou UTP dentro do mesmo chip
- 10 Mbits/s no modo half-duplex ou 20 Mbits/s no modo full-duplex;
- Inversão automática de TIP e RING
- Múltiplos modos de operação da LAN
- Link da WAN de até 40 Mbits/s
- Link da WAN de até 115.2 kbits/s usando gerador interno de freqüências
- Tabela MAC de 10.000 endereços

- Capacidade do buffer de até 256 pacotes
- Taxa de filtragem e encaminhamento de até 15.000 pacotes por segundo
- Aprendizado e envelhecimento de endereços automático
- Suporte a 7 leds de status
- Tecnologia CMOS de baixo consumo de 6 microns
- Encapsulamento PQFP de 100 pinos

4.1.3. Diagrama em blocos

A figura 4.1.2.1 mostra um diagrama em blocos do chip *bridge* da RAD

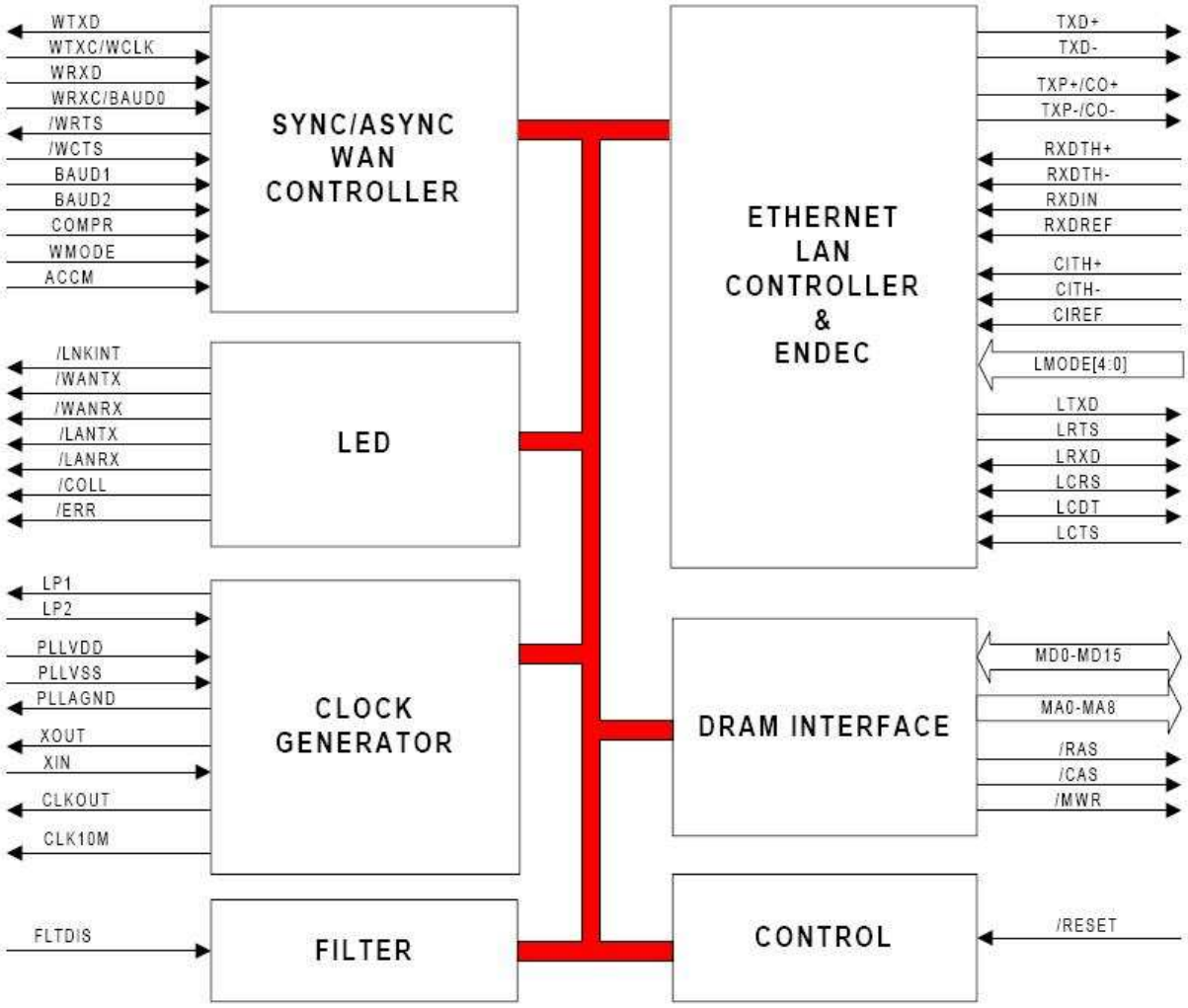


Figura 4.1.2.1 – Diagrama em blocos do RJ-17

4.2. ADMTEK ADM6993

Conforme foi mencionado anteriormente, a ADMTek foi adquirida pela empresa Infineon. Desta forma, poucas informações referentes à antiga empresa estão disponíveis na rede, pois o processo de absorção da tecnologia já está bem adiantado. Sendo assim, algumas informações sobre o perfil da Infineon são colocadas a seguir.

"A Infineon é um líder e inovador no cenário internacional das indústrias de semicondutores. Nós criamos, desenvolvemos, produzimos e vendemos uma vasta gama de semicondutores e sistemas com a solução completa focada em indústrias selecionadas. Nossos produtos servem as aplicações de comunicações sob par metálico e sem fio, setor automotivo, industrial, computadores e segurança. Nosso portifólio de produtos é composto por memórias, lógica em geral e inclui circuitos integrados digitais, híbridos (*mixed signal*) e analógicos bem como semicondutores discretos e soluções em nível de sistema."

4.2.1. Descrição funcional

Abaixo temos uma descrição do dispositivo da ADMTek/Infineon, diretamente retirada do manual do componente.

"O ADM6993/X é um único chip que integra dois transceptores de 10/100 Mbits auto MDI/X TX/FX, um controlador Ethernet de 3 portas com função de *switch*, e possui o modo conversor para ser utilizado nas mais diversas aplicações, incluindo conversores de fibra/Ethernet, switches de 2 ou 3 portas, gateways VoIP, bem como roteadores. O ADM6993/X é a versão "*green*" do ADM6993.

O ADM6993/X implementa características de priorização dos pacotes baseados na porta, etiqueta de VLAN ou tipo de serviço IP. A prioridade dos pacotes pode ser dada a partir da porta TCP no caso de aplicações multimídia.

A segunda interface MAC pode ser usada como TP/FX (par trançado ou fibra óptica) ou MII/RMII/GPSI para ser conectada diretamente a dispositivos de ponte a meios diferentes. A terceira interface MAC pode ser usada como MII/RMII/GPSI/HDLC. O canal HDLC dedicado suporta taxas de 64 kbits/s a 50 Mbits/s.

As portas 0 e 1 do ADM6993/X suportam auto MDIX 10Base-T/100Base-TX e 100Base-FX conforme especificado na IEEE 803.3 através do uso de conversores A/D de alta velocidade.

O ADM6993/X suporta gerenciamento serial através da interface SMI de forma a permitir que um processador pequeno e de baixo custo possa-o inicializar e configurar. Ele também provê status das portas para gerenciamento remoto através de um agente bem como estatísticas através de contadores.”

4.2.2. Características

As principais características estão listadas abaixo:

- Switch integrado de 3 portas 10/100M sendo 2 com transceptor interno e a terceira com controlador HDLC
- Provê modo conversor de mídia com capacidade de redundância usando 2 ADM6993/X
- Pequena latência no modo conversor
- Buffer de dados de 6kx64 bits em SRAM incorporado no chip
- Até 2.000 endereços MAC
- Tabela de endereços com função de envelhecimento
- Duas filas por porta para QoS
- Prioridade baseada em porta, TCP/IP ToS e segundo 802.1p
- Arquitetura tipo *store and forward*
- Controle de fluxo segundo 802.3x para operação no modo *full-duplex* e *back-pressure* para operação no modo *half-duplex*
- Suporta auto negociação
- Tamanho dos pacotes de até 1536 bytes
- *Broadcast storming filter*
- VLAN baseada em porta ou em etiqueta de VLAN
- 16 entradas para classificação de pacotes
- Interface de gerência serial para uso com processadores de baixo custo
- Provê contadores para implementação de estatísticas
- Provê status para monitoramento remoto
- Encapsulamento PQFP de 128 pinos com tensões de alimentação de 2.5 e 3.3V

4.2.3 Diagrama em blocos

A figura 4.2.3.1 mostra um diagrama em blocos do ADM6993/X

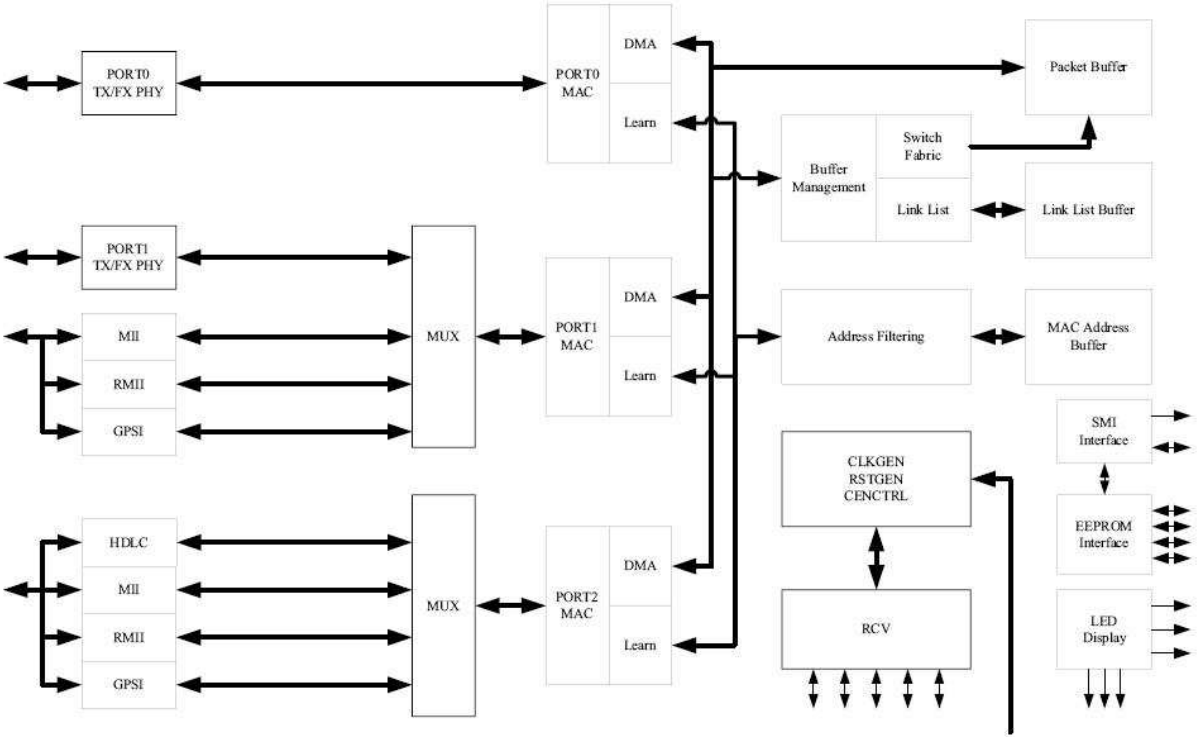


Figura 4.2.3.1 – Diagrama em blocos do AMD6993/X

4.3 CONCLUSÕES

No mercado existem duas opções de circuitos integrados que realizam a função de *bridge* Ethernet através do encapsulamento dos pacotes em quadros HDLC. O uso deste protocolo requer a adição de um pequeno cabeçalho, o que não representa uma queda significativa no na eficiência do canal de dados entre as duas bridges, local e remota.

Estes dispositivos possuem flexibilidade limitada, uma vez que grande parte de suas funções são executadas em hardware e, aquelas que são executadas em software, não permitem atualizações.

5. DESEMPENHO DE SWITCHES ETHERNET

Existem vários tipos de equipamentos no mercado que podem ser utilizados para testes de equipamentos de comutação de pacotes, sejam eles IP (nível 3) ou puramente Ethernet (nível 2). Uma das ferramentas mais convenientes para esse tipo de teste são as ferramentas de software disponíveis gratuitamente na Internet. Destacam-se entre estes tipos de softwares o IPERF [38] e o NetPerf [39]. Ambos os projetos podem ser executados sobre a plataforma Linux, sendo que o IPERF já possui porte para Windows disponível na rede. Com elas é possível geras pacotes ethernet de tamanhos diversos e com taxas de transmissão variadas. Desta forma, pode-se testar o desempenho dos equipamentos com os mais diferentes tipos de pacotes.

Com o uso destes softwares é possível utilizar PC's comerciais como plataformas de teste de redes. O problema neste tipo de aplicação é que o desempenho da própria ferramenta de teste pode ter uma variação significativa e inclusive influenciar no resultado do teste. Visto que a maior parte do trabalho é feito por pelo software do sistema operacional. A figura 5.1 exemplifica uma plataforma de testes de uma *bridge* ethernet através de uso de duas máquinas separadas, executando a ferramenta de teste.

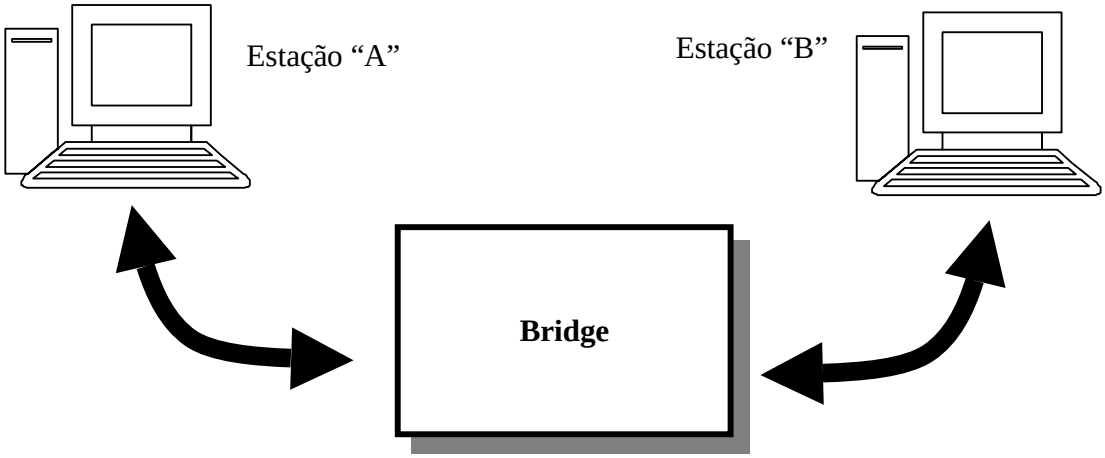


Figura 5.1 - Teste de uma bridge usando IPERF

O mecanismo consiste na geração de pacotes em uma estação e a recepção dos mesmos na outra estação. O teste pode ser unidirecional, de A para B por exemplo, como também pode ser bidirecional, fazendo com que o equipamento em teste seja mais exigido.

O IETF [40] define em duas RFC's [42] e [43] (Request For Comments) a terminologia e também os testes que devem ser executados quando na avaliação da performance de equipamentos de comutação de pacotes para redes locais. As duas RFC's em questão são:

- RFC2285 - Benchmarking Terminology for LAN Switching Devices
- RFC2889 - Benchmarking Methodology for LAN Switching Devices

No caso da segunda RFC, que define a metodologia de testes, é interessante notar que um dos autores da mesma, na época da formulação do documento, representava a Spirent Communications, uma das maiores fabricantes de equipamentos de teste de equipamentos no mundo ao lado de outras gigantes como a Agilent (derivada da HP).

5.1. MÉTRICAS

5.1.1 Throughput

Esta métrica indica a capacidade de encaminhamento de pacotes pelo equipamento. Para equipamentos de comutação de pacotes em redes locais (*switches*) o IETF define algumas variações desta métrica.

Forwarding Rate (FR)

Esta é a taxa de pacotes por segundo dados observada na porta de saída do equipamento em resposta a uma determinada carga na porta de entrada, não importando a perda de pacotes neste processo. A unidade de medida desta métrica é em pacotes de N-bytes por segundo.

Forwarding rate at maximum offered load (FRMOL)

Esta métrica indica a taxa de pacotes por segundo que um equipamento pode encaminhar sendo que a taxa oferecida ele pelo equipamento de teste é a maior possível. A 'taxa máxima de transmissão na maior carga oferecida' em alguns casos pode ser menor do que a taxa máxima de transmissão que um equipamento apresenta. Isso se deve ao fato de a performance dos equipamentos ser afetada quando uma taxa muito alta é apresentada em suas portas. Um exemplo

desse problema está nas implementações de rede baseadas em sistemas Linux. A cada pacote ethernet recebido pela interface de rede, uma interrupção de hardware é gerada. Essa interrupção indica à CPU que existe um pacote a ser tratado. Quando o tamanho dos pacotes diminuem, a taxa de interrupções por segundo aumentam (para uma mesma taxa de transmissão de bits) e isso pode fazer com que a CPU fique mais tempo atendendo interrupções (o que em algumas plataformas pode ser um processo custoso) do que tratando os pacotes que devem ser encaminhados. Algumas placas de redes possuem um recurso armazenamento temporário de pacotes que somente interrompe a CPU após que um determinado número de pacotes seja recebido ou que um determinado período de tempo tenha passado (evitando que um pacote fique perdido para sempre no buffer). Assim, o intervalo entre as interrupções fica maior e a cada interrupção a CPU lê da interface de rede um maior número de pacotes.

Maximum forwarding rate (MFR)

Esta métrica é obtida através de um processo iterativo que mede a mais alta taxa de transmissão sem que hajam pacotes perdidos. Na tabela 5.1.1.1, representamos um caso hipotético retirado da RFC2285 e que exemplifica a relação entre MFR e FRMOL.

Tabela 5.1.1.1 - Relação entre MFR e FRMOL

	Taxa oferecida pelo equipamento de teste	**Taxa de transmissão do equipamento testado**
1	14.880 pps – MOL	7.400 pps - FRMOL
2	13.880 pps	8.472 pps
3	12.880 pps	12.880 fps - MFR

Na tabela 5.1.1.1 observamos que para uma taxa oferecida de 12.880 pacotes por segundo a taxa de transmissão observada na saída foi a mesma. Isso indica que não houveram perdas de pacotes. Quando a taxa foi aumentada para 13.880 pps, observou-se que houveram perdas, desta forma ficou determinado que a máxima taxa de transmissão do equipamento foi de 12.880 pps.

Ainda pela 5.1.1.1, podemos observar que a taxa de transmissão diminui à medida em que a taxa oferecida aumenta, conforme os motivos que havíamos mencionado anteriormente, sendo que na máxima carga possível oferecida pelo equipamento de testes, a taxa de transmissão do

equipamento testado foi de 7.400 pps, o que determinou a FRMOL (taxa de transmissão na máxima taxa oferecida).

Embora o tamanho dos pacotes não esteja especificado na tabela 5.1.1.1, o IETF sugere que o mesmo seja sempre indicado, pois os resultados acima podem mudar de acordo com seu aumento ou diminuição. Em geral, a performance de equipamentos de comutação diminui com a diminuição do tamanho dos pacotes.

É importante ressaltar que o teste de taxa de transmissão pode ser influenciado pelo tamanho dos *buffers* do equipamento, para que este efeito seja minimizado, é necessário que o teste tenha uma duração grande o suficiente para fazer com que os *buffers* sejam totalmente preenchidos, o que não é uma tarefa fácil de se determinar. A presença dos *buffers* para armazenamento dos pacotes recebidos é importante para que equipamento tenha um bom desempenho frente às rajadas (*bursts*) de dados que são comuns de ocorrer em redes de computadores.

5.1.2. Latência

Esta métrica indica o tempo que um pacote leva para ser encaminhado da porta de entrada até a porta de saída do equipamento. Na RFC1242 [41], o IETF define latência de duas formas distintas de acordo com o tipo de equipamento:

> ***Equipamentos do tipo "armazenagem e encaminhamento"***
> O intervalo de tempo iniciado quando o último bit do frame de entrada alcança a porta de entrada e encerrado quando o primeiro bit do frame de saída é visto na porta de saída.
>
> ***Equipamentos de encaminhamento de bit***
> O intervalo de tempo iniciado quando o final do primeiro bit do frame de entrada alcança a porta de entrada e encerrado quando o início do primeiro bit do frame de saída é visto na porta de saída.

Para switches o IETF, referindo-se à *Broadcast latency*, define que o tempo de latência deve ser medido apenas para pacotes do tipo *broadcast*, e o resultado do teste deve indicar a latência do pacote para todas as portas do equipamento. Estas podem ser portas pertencentes à mesma placa, portas de placas diferentes interligadas em um mesmo chassis, ou portas de chassis diferentes interligadas através de um *backbone* no caso de teste de sistemas de maior porte. O IETF recomenda que *switches* sejam testados de acordo com o segundo grupo de equipamentos descritos acima.

5.1.3. Jitter

Em sistemas de comunicação de dados, *jitter* é a métrica que representa a variação de fase de um sinal. A variação de fase, ou atrase de um bit para outro, pode prejudicar o processo de recuperação dos dados e do relógio em um sistema de transmissão de dados. Em redes de computadores, o *jitter* em geral é definido como a variação da latência. Embora nestas RFC's abordadas aqui não definam a maneira como esta métrica deve ser mensurada, ela com certeza tem efeito negativo em parte das aplicações. Com o advento do VoIP por exemplo, que transmite a voz digitalizada sob a forma de pacotes IP, é importante que a cadência de chegada dos pacotes no receptor seja mantida constante, ou muito perto disso, caso contrário, podem haver perdas de dados devido à *overflows* ou *underflows* de *buffers* de dados.

Outra aplicação muito sensível à variações de atraso são as aplicações de emulação de circuitos (CES). Estas aplicações emulam circuitos síncronos através de redes assíncronas como ATM pelo AAL1 e mais recentemente pela emulação de circuitos através de redes IP, como a tecnologia TDMoIP (*Time Division Multiplexing Over IP*) da RAD e outras como PWE3 (Pseudo Wire Emulation Edge to Edge) da IETF.

5.2. OTIMIZANDO OS PROCESSOS PARA MAXIMIZAR A PERFORMANCE

Em sistemas de comutação de pacotes, uma das questões chave é a velocidade com que os mesmos são encaminhados. Neste mecanismo, uma das tarefas mais dispendiosas é a consulta a base de dados para determinar a qual ou quais portas um pacote deve ser enviado.

Dispositivos com hardware mais complexo dispõem de memórias do tipo CAM (*Content Addressable Memory*). Estas memórias são extremamente rápidas e úteis na consulta a bases de

dados pois se comportam no sentido inverso às memórias RAM. A RAM produz em sua saída o dado de um endereço de entrada, por outro lado, a CAM apresenta em sua saída o endereço de um determinado dado de entrada. A procura em uma lista baseada em memória RAM pode levar vários ciclos, dependendo do endereço em que a entrada se encontra. A CAM, por outro lado, procura todos os endereços em paralelo. O problema deste tipo de memória é a necessidade de uso de grandes recursos de hardware (silício) principalmente para memórias de grande densidade.

Para driblar este problema, uma solução implementada em software tem se mostrado muito eficiente. Esta solução emprega o conceito de "*hashing*". Através da aplicação deste conceito, temos as Tabelas Hash. Uma Tabela Hash é composta de duas partes, uma lista (que é a tabela, contendo os elementos, propriamente dita) e uma função de mapeamento, também denominada de função de *hash*. A função *hash* é um mapeamento do universo das entradas, sejam elas palavras ou endereços MAC de uma rede, para o universo dos números inteiros. Este universo de números inteiros é exatamente o que irá servir de índice para a lista. Em outras palavras, a função de *hash* provê uma maneira de relacionar números inteiros às entradas, que no caso em questão são endereços MAC, de forma que o dado possa então ser colocado na posição da lista correspondente ao número inteiro obtido.

Existem inúmeras funções de *hash* implementadas, cada uma delas podendo ser mais adequada a determinada aplicação, dependendo das entradas, do número gerado na saída, podendo variar desde uma simples soma, passando por CRC's (*cyclic redundancy check*), à algoritmos mais complexos.

Desta forma, tanto o armazenamento quanto a consulta à base de dados têm um custo igual a 1, pois o endereço MAC de entrada sempre irá gerar o mesmo *hash*, desde que a função que o calcula seja a mesma.

Uma função *hash* não garante que cada dado de entrada irá gerar uma saída diferente. Sempre existe a chance de que duas entradas diferentes irão gerar a mesma saída. Tenhamos como exemplo os endereços MAC em questão, onde usamos uma tabela de 256 endereços e uma função que mapeia os endereços MAC em números que variam de 0 a 255. Um endereço Ethernet possui 48 bits (6 bytes), ou seja, o universo de entradas é muito maior que o universo de saídas. Assim, é evidente que diferentes endereços de entrada poderão gerar saídas iguais.

Isso indica que os elementos que geram saídas iguais devem ser colocados na mesma posição da lista. Este fenômeno é denominado de colisão. A seguir, iremos discutir sobre os vários métodos conhecidos para tratar do problema de colisões em tabelas *hash.*

5.3 CONCLUSÕES

Ethernet é uma tecnologia estável a largamente utilizada. Em vista disso, é importante que se tenha uma maneira padronizada de medição da performance de equipamentos de comutação de pacotes (*switches*).

O IETF, através de duas RFC´s, realizou um esforço para a padronização destas medidas, as quais são adotadas pelos equipamentos de teste e medição existentes no mercado.

6. FUNÇÕES DE HASH

6.1. DEFINIÇÃO

Na língua inglesa o verbo "*to hash*" significa cortar em pedaços. O conceito por trás das funções de *hash* está em cortar fora alguns aspectos do elemento de entrada (denominado chave) e usar essa informação parcial como base para outras operações [48]. Existem vários tipos de funções de *hash*, desde as mais simples como as do tipo "extração de bit", onde o conceito apresentado inicialmente do simples "corte" da informação é utilizado, passando pelos populares somatórios e códigos de CRC e finalmente chegando aos complexos algoritmos utilizados em criptografia, tais como o MD5 [49] e o SHA-1 [50].

6.2. APLICAÇÕES

As operações realizadas com o resultado da aplicação das funções de *hash* sobre uma determinada entrada podem estar desde a procura pelo elemento em uma tabela (conceito de tabelas *hash*, abordado mais adiante neste trabalho), armazenamento de senhas de usuários de um sistema para futura autenticação, no caso de aplicações de segurança e criptografia, ou códigos de detecção e correção de erros em sistemas de comunicação de dados.

O uso de funções de *hash* nas aplicações de segurança é largamente difundido. É essencial que não somente as técnicas utilizadas nestas aplicações sejam eficientes, mas também é necessário que o algoritmo seja executado da forma mais rápida possível. Para aplicações embarcadas, existem no mercado muitos dispositivos comerciais de grandes fabricantes, como é o caso do Freescale MPC190 [53] (co-processador criptográfico de segurança). Ao mesmo tempo, muita pesquisa também é feita para o desenvolvimento de dispositivos embarcados de alta performance para aplicações de segurança e criptografia, como propõe Wang e outros em [12].

Em redes ATM temos uma aplicação muito interessante das funções de *hash*, uma dupla utilização. A função utilizada para detecção de erros no cabeçalho de uma célula também é utilizada pelo mecanismo de delineamento das mesmas, o qual tem como propósito a localização do cabeçalho das células dentro do feixe de bits. Como o cabeçalho da célula possui apenas 40 bits e o byte de detecção de erro cobre apenas os campos do cabeçalho, uma função

implementada em hardware calcula, a cada novo bit que chega pela interface, o código CRC dos últimos 40 bits recebidos, quando o valor obtido pelo cálculo for igual a zero, isso indica o início da célula ATM.

6.3. MÉTODOS DE HASH

Neste capítulo, iremos abordar alguns dos mais populares tipos de funções de *hash*, denominados métodos, para aplicações em tabelas *hash*, as quais iremos discutir mais adiante.

6.3.1. Extração de bits

As funções do tipo extração de bits consistem em escolher um número *j* de bits dentre os *i* bits existentes na chave. Não existe padrão em quais bits serão selecionados, sendo que a escolha poder ser contígua ou intercalada. As funções de extração de bits são extremamente simples de serem implementadas em hardware. O hardware necessário para gerar as duas funções de *hash* ilustradas nas figuras 6.3.1.1 e 6.3.1.2 é exatamente o mesmo.

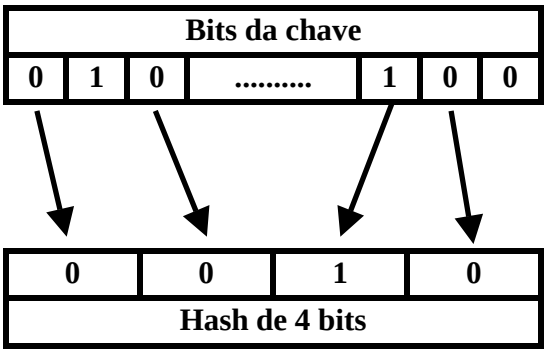


Figura 6.3.1.1 - Extração de bits intercalada

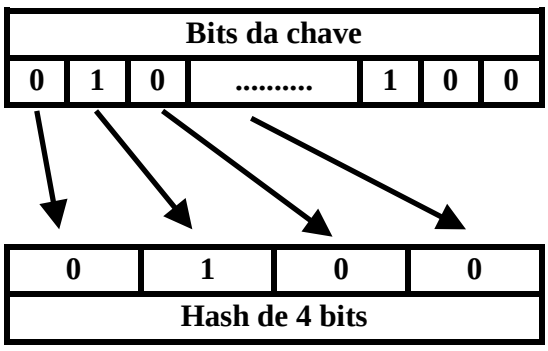


Figura 6.3.1.2 - Extração de bits contígua

Por outro lado, funções de *hash* do tipo extração de bits, quando implementadas em software, podem se tornar muito complexas, dependendo de quais bits necessitam ser extraídos da chave, exigindo vários ciclos de instruções no microprocessador.

6.3.2. Método da divisão

O método da divisão consiste em dividir a chave por um número *n*, e utilizar o resto da divisão como *hash*. O resto da divisão de um número por um inteiro *n* gera um outro número inteiro que pode ser maior ou igual a zero e menor do que *n*.

$$h(K) = MOD(K, n)$$
$$0 \leq h(K) < n$$

Desta forma, utilizando o método da divisão, podemos escolher o número máximo de saídas possíveis com esta função, o que é uma informação útil no dimensionamento dos recursos do sistema onde o *hash* será utilizado, como por exemplo, nas tabelas *hash* que abordaremos mais adiante.

Neste método é preciso tomar um cuidado especial na escolha de *n*. Por exemplo, sabemos que na representação binária de um número, a divisão deste número por 2^k é simplesmente o deslocamento de seus bits em *k* casas para a direita. Com isso, o resto da divisão deste número por 2^k é na verdade a extração dos *k* bits menos significativos do número, ou seja, um caso particular de do método de extração de bits que abordamos anteriormente. Para evitar isso o ideal é que a escolha de *n* recaia sobre o número primo que não seja próximo de nenhuma potência de 2.

É interessante ressaltar que no método da divisão, chaves consecutivas geram saídas consecutivas, o que pode, dependendo da aplicação, ser uma desvantagem. Outro aspecto importante a ser observado é quando pretende-se utilizar o método da divisão em software. Embora a codificação da operação em linguagem C seja extremamente simples, como mostra o código abaixo, operações de divisão e resto de divisão podem ter um alto custo associado, dependendo do microprocessador utilizado.

h(i) = i % n;

6.3.3. Hash usando operação lógica XOR

Funções de *hash* que utilizam a operação lógica “ou exclusivo” (XOR) são relativamente fáceis de serem implementadas em hardware, por isso são muito utilizadas. O método consiste em criar dois grupos de bits da chave de entrada e em seguida realizar a operação ou exclusivo, bit a bit, destes dois grupos. O resultado, de largura de bits menor do que a chave original, é o *hash*. Por exemplo, para a chave de 8 bits $x = x_1x_2x_3...x_8$, podemos calcular o *hash* de x como sendo:

$h(x) = (x_1 \oplus x_5)\ (x_2 \oplus x_6)\ (x_3 \oplus x_7)\ (x_4 \oplus x_8)$.

Este método muitas vezes é denominado de “dobramento”, fazendo referência a uma folha de papel, onde estão representados os bits da chave, sendo dobrada. A figura 6.3.3.1 ilustra este processo, porém, com uma operação de adição simples entre os algarismos da chave.

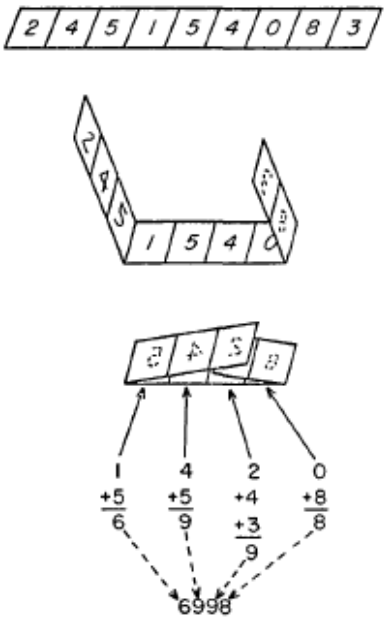


Figura 6.3.3.1 - Método de “dobramento”

Inúmeras combinações de bits podem ser feitas com o método, sendo que nem todos os bits devem necessariamente ser utilizados na operação. Por exemplo, os bits das chaves que tem uma tendência a se repetir no universo de entradas podem ser deixados de lado de forma a otimizar o processo. Voltando ao Capítulo 2, onde abordamos a distribuição de endereços MAC pelo IEEE para os fabricantes de equipamentos, podemos observar que em uma rede local existe uma tendência que a porção do endereço MAC correspondente ao OUI do fabricante e repita.

Dependendo da aplicação e do perfil das entradas, uma ou outra combinação entre os bits da chave pode ser mais eficiente. Em geral, os microprocessadores possuem em seu conjunto de instruções, as operações lógicas bit a bit, inclusive o XOR. Assim, este método torna-se muito eficiente também par aplicações em software. No caso específico de tratamento do campo de endereços de pacotes ethernet, os quais possuem 6 bytes de largura, com poucas linhas de código é possível realizar o XOR entre os 6 bytes gerando um *hash* de 8 bits, o que leva a uma tabela de 256 posições, tamanho suficiente de endereços para a maioria das aplicações SOHO.

6.3.4. Hashing Universal

Carter e Wegman definiram em [21] certas classes de funções de *hash* denominadas "Classes Universais de Funções de Hash". Antes de abordá-las, iremos especificar algumas notações. Se **S** é um universo de chaves, então **|S|** denota o número de chaves existentes neste universo. $\lceil \mathbf{x} \rceil$ indica o menor inteiro maior ou igual a **x.** Todas as funções de *hash* mapeiam um conjunto **A** para um conjunto **B,** sendo que **|A|** > **|B|**. Algumas vezes **A** é denominado de o conjunto de possíveis chaves, e **B** é o conjunto de índices de saída. Se **f** é uma função de *hash* e $x,y \in A$, então definimos que:

$\delta_f(x,y) = 1$ se $x \neq y$ e $f(x) = f(y)$
$\delta_f(x,y) = 0$ se $x \neq y$ e $f(x) \neq f(y)$

Em outras palavras, $\delta_f(x,y)$ é igual a 1 se duas chaves distintas x e y pertencentes ao conjunto **A** forem mapeadas para o mesmo índice usando a função de *hash* **f.** Se substituirmos f,

x, ou y em $\delta_f(x,y)$ por um conjunto, temos que somar todos os elementos no conjunto. Desta forma, se **H** é um conjunto de funções de *hash*, $x \in A$ e $S \subset A$ então temos:

$$\delta_H(x,S) = \sum_{f \in H} \sum_{y \in H} \delta_f(x,y)$$

As "Classes Universais de Funções de Hash" possuem algumas propriedades interessantes. A primeira delas diz que uma classe H de funções de *hash* que mapeiam de $A \rightarrow B$ é universal se para todo x,y em A tivermos:

$$\delta_H(x,y) \leq \frac{|H|}{|B|}$$

Em outras palavras, H é universal se nenhum par de chaves distintas x e y são mapeadas para o mesmo índice de B por mais de $\frac{1}{|B|}$ das funções.

Dada uma coleção de H de funções de *hash* (não necessariamente universal), existem duas chaves $x,y \in A$ tal que:

$$\delta_H(x,y) > \frac{|H|}{|B|} - \frac{|H|}{|A|}$$

Seja x qualquer elemento de A, e S seja subconjunto de A. Seja **f** uma função de *hash* escolhida aleatoriamente de uma classe universal de funções, então o número esperado de elementos de S com os quais x colide (possui o mesmo *hash*) é:

$$\delta_f(x, S) \leq \frac{|S|}{|B|}$$

Se reduzirmos S a apenas uma chave escolhida do aleatoriamente em A então temos que a probabilidade de existirem colisões é de $\frac{1}{|B|}$. Em uma classe universal de funções, existem aquelas que possuem um desempenho abaixo da probabilidade esperada, funções mais pobres.

Porém, o desempenho das funções mais adequadas ao conjunto de entradas acaba compensando o baixo desempenho das funções pobres e a probabilidade se mantém. A chave para uma aplicação bem sucedida com funções de *hash* de classes universais está na utilização de várias funções ao longo do tempo. O momento certo de mudar de função deve ser cuidadosamente escolhido, pois isso implica em realizar o *hash* de todas as entradas novamente.

Uma das classes universais de funções de *hash* mais interessante é a classe H_3. As funções de *hash* desta classe não requerem o uso de multiplicação e podem ser facilmente implementadas em hardware e representadas em software, pois podem são obtidas através do uso de operações de OU exclusivo e o uso de arrays de bits. Desta forma, a mudança de uma função para outra da mesma classe é facilmente realizada. Suponhamos que A é um conjunto de números de i dígitos na base α, e B um conjunto de números binários de largura j. Desta forma temos:

$|A| = \alpha^i$

$|B| = 2^j$

Seja M uma classe de arrays de tamanho $i \times \alpha$, dos quais cada elemento é um conjunto de bits de comprimento j. Para $m \in M$, seja m(k) o conjunto de bits correspondente k-ésimo elemento de m, e para $x \in A$, seja x_k o k-ésimo dígito de x na base α. Definimos:

$$f_m(x) = m(x_1 + 1) \oplus m(x_1 + x_2 + 2) \oplus .... \oplus m(\sum_{k=1}^{i} x_k + k)$$

A classe H_3 é o conjunto $\{f_m \mid m \in M\}$. Ramakrishna em [52] comprovou o desempenho das funções de *hash* da classe H_3 com dados reais, tais como arquivos com a hora de *log-in* de usuários em um sistema, palavras de um dicionário UNIX, id's de usuários em um sistema, sendo que os dados não numéricos foram convertidos em inteiros de 32 bits. Em seus experimentos, com a utilização de 500 funções de *hash* escolhidas aleatoriamente da classe H_3, o autor chegou muito próximo dos resultados teóricos esperados. Ramakrishna ainda comparou o desempenho das funções da classe H_3 com outros dois tipos de funções de *hash*, método de extração de bits e XOR, através da obtenção do tamanho da máxima seqüência de procura em uma tabela *hash* e mostrou que tais funções são muito mais eficientes.

Sistemas que utilizam funções de *hash* para endereçamento de tabelas estão sujeitos à ataques externos que enviam chaves que levam ao mesmo *hash*. Dependendo de como o sistema foi projetado, efeitos catastróficos podem ser ocasionados por tais ataques. O uso da classe H_3 permite que a função de *hash* seja trocada com muita facilidade pela aplicação, conforme sugere Ramakrishna em [52] através de bancos de registradores. Desta forma podemos deixar o sistema mais robusto e ao mesmo tempo diminuir a probabilidade de colisões, diminuindo o tamanho da máxima seqüência de procura na tabela.

6.3.5. Hash perfeito

Em alguns casos, é possível encontrar uma função de *hash* realize o mapeamento de todas as *n* chaves de um universo para *n* saídas diferentes, sem que haja repetição nos valores (colisões). Para que isso seja possível, devemos ter conhecimento antecipado de todos o universo de chaves possíveis na aplicação em questão. Tais funções são denominadas funções de *hash* perfeitas.

Funções de *hash* perfeitas não necessariamente geram saídas inteiras consecutivas, o que significa que a tabela utilizada para armazenamento das chaves poderá ter mais posições do que o número total de chaves. Em outras palavras, uma função de *hash* perfeita mapeia *n* chaves para uma faixa de 0...*k*-1 saídas. Nos casos em que $k = n$ a função é denominada “função de *hash* perfeita mínima”.

Funções de *hash* perfeitas são úteis quando temos um universo de chaves de entrada que não varia no tempo, ou pelo menos não varia com freqüência, dada a complexidade da geração da função, portanto tais funções têm bastante utilidade em dicionários e compiladores. Uma ferramenta bastante popular para a geração de funções de *hash* perfeitas é o GNU gperf [54]. A figura 6.3.5.1 ilustra a diferença entre uma “função de *hash* perfeita” e uma “função de *hash* perfeita mínima”.

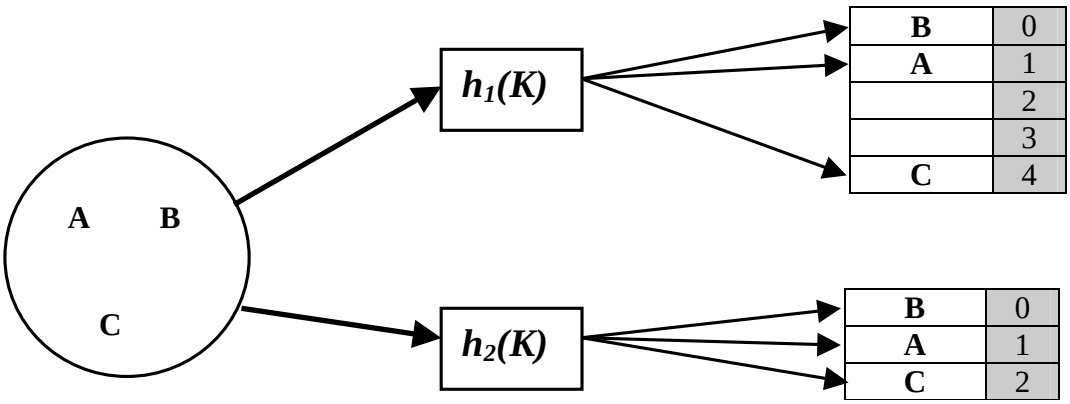


Figura 6.3.5.1 - Função de *hash* perfeita e perfeita mínima

Na figura 6.3.5.1, a função de *hash* $h_1(K)$ é perfeita para o universo *K*, porém não gera endereços contínuos, por isso não é mínima. Já $h_2(K)$ gera as saídas como números inteiros consecutivos, por isso a função é denominada função de *hash* perfeita mínima.

O desenvolvimento de algoritmos para a criação de funções de *hash* perfeitas é um assunto muito estudado e várias propostas neste sentido são publicadas, tais como os descritos em [5], [9] e [55].

6.4 CONCLUSÕES

Funções de *hash* são muito utilizadas em sistemas de telecomunicações e redes de computadores, seja para a detecção e correção de erros ou para a autenticação de usuários.

A vantagem no uso destas funções reside no fato das mesmas terem uma grande variedade. Desta forma, pode ser encontrada a função de *hash* que melhor se adapte à aplicação final, seja ela implementada em software ou em hardware.

7. TABELAS HASH

7.1. DEFINIÇÃO

Tabelas *hash* podem ser consideradas o equivalente em software das CAM's abordadas no capítulo 9. Estas estruturas de dados fazem uso de funções de *hash* para a geração do endereço onde as entradas, denominadas chaves, serão armazenadas na tabela. A figura 7.1.1 ilustra o mecanismo utilizado em uma tabela *hash*, onde o universo de chaves *K* = { A, B, C, D, E } é processado pela função de *hash h(K)*, gerando os endereços de armazenamento na tabela.

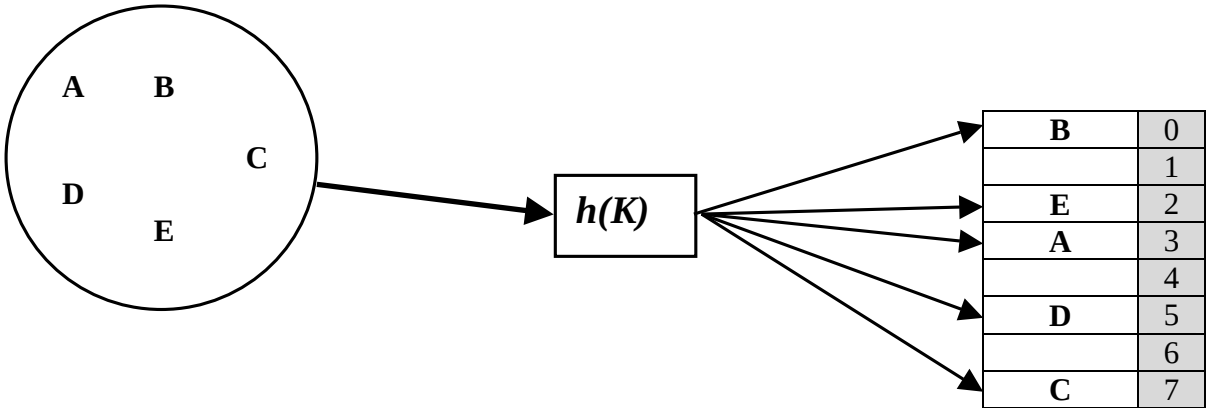


Figura 7.1.1 - Mecanismo de uma tabela *hash*

7.2. APLICAÇÕES

Uma das aplicações mais comuns das tabelas *hash* reside na classificação de pacotes. O kernel do Linux por exemplo, consulta determina a rota pela qual enviar um pacote que está sendo tratado através da consulta a sua tabela de roteamento. De forma a tornar este mecanismo mais rápido, as últimas consultas são armazenadas em uma memória cache denominada "*Routing Cache*". Esta memória é implementada em software através do uso de uma tabela *hash*.

Tabelas *hash* também são denominadas arrays associativos, ou seja, estruturas que servem para associar entradas a uma determinada saída ou ação, como por exemplo em compiladores, onde a tradução de mnemônicos para códigos de máquina deve ser feita.

7.3. O PROBLEMA DE COLISÕES EM TABELAS HASH

Para que uma tabela *hash* funcione corretamente é necessária a escolha de uma função de *hash* adequada para a aplicação. A função adequada é aquela que consigue agregar velocidade na execução e minimizar a ocorrência de *hash*'s idênticos para chaves distintas. Com o auxílio de componentes de hardware é possível resolver o problema da velocidade. Entretanto, o problema da duplicação de valores de saída é mais complexo. Quando uma função de *hash* gera valores iguais para duas chaves distintas, temos uma colisão.

Funções de *hash* que, ao serem aplicadas à um universo de chaves de entrada geram saídas totalmente distintas são surpreendentemente raras de serem encontradas. Em [48], Knuth demonstra isso com um caso onde um grupo de 31 chaves são, uma a uma, transformadas por uma função f(K) em números que variam de –10 a 30.

Um caso clássico e muito ilustrativo é o do "paradoxo do aniversário" que diz que em uma sala com 23 pessoas ou mais, a chance de haver pelo menos duas delas com exatamente o mesmo dia e mês de aniversário é em maior que 50%. Trazendo isso para o universo das tabelas *hash*, podemos descrever o caso como uma função de *hash* aleatória que mapeia um universo de 23 chaves para uma tabela de tamanho igual a 365. Neste caso a probabilidade de não haverem colisões é de aproximadamente 0,4927.

O tratamento das colisões em tabelas *hash* é um problema largamente estudado e é a chave para o bom desempenho destas estruturas. Nos sub-capítulos a seguir iremos abordar vários tipos de tabelas *hash* e como as colisões são tratadas em cada caso.

7.4. TABELAS HASH COM ENDEREÇAMENTO ABERTO

Um dos tipos de tabelas *hash* mais populares, dada sua simplicidade, são as tabelas com endereçamento aberto (*open addressing)*. Neste tipo de tabela, as chaves são diretamente armazenadas nas posições da tabela correspondentes ao seu *hash* (ao contrário do método de encadeamento - *chainning* - que veremos mais adiante). Quando uma determinada chave precisa ser armazenada na tabela e o *hash* gerado é um endereço que já está ocupado, a entrada e armazenada em posições alternativas da tabela. Caso a posição alternativa também não esteja disponível, procura-se uma outra até que encontre-se uma posição vaga. Este processo é

denominado amostragem (*probing)*, onde as possíveis posições de armazenamento são testadas uma a uma.

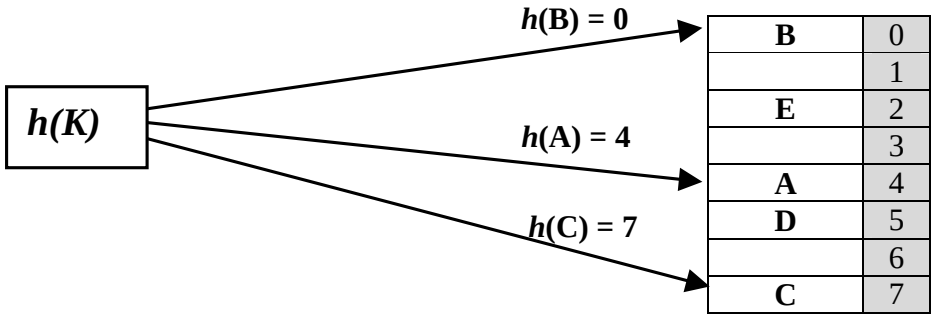


Figura 7.4.1 - Tabela *hash* com endereçamento aberto

7.4.1. Tratamento de colisões em endereço aberto

Muitas das técnicas para tratamento das colisões em tabelas *hash* com endereçamento aberto são bem conhecidas e definidas, tais como as abordadas em [11]. Para realizar a procura em uma tabela *hash* que foi preenchida através do uso do processo de amostragem, a mesma rota deve ser seguida até que se encontre a chave desejada ou uma posição vazia. Tenhamos o exemplo da figura 7.4.1.1 onde três chaves quaisquer geraram o mesmo *hash* e, por conseqüência, deveriam ser armazenados na mesma posição, que simbolizamos por "n".

$h(k_1) = h(k_2) = h(k_3)$

Posição	.....
	.....
n-1	k_3
n	k_1
n+1	k_2
	.....

Figura 7.4.1.1 - Colisão em uma tabela *hash* de endereçamento aberto

A chave k_1, a primeira das três a ser armazenada, foi realmente colocada na posição "n". Em seguida, a chave k_2 também gerou o mesmo *hash*. Porém, como a posição já estava ocupada, foi armazenada na primeira posição acima (n+1). Da mesma forma, k_3 gerou o mesmo *hash* e teve de ser armazenado na primeira posição abaixo de "n" uma vez que "n+1" também já estava ocupada.

Para realizar a procura na tabela, devemos seguir o mesmo caminho utilizado na armazenagem das chaves. Desta forma, para procurar a entrada k_3, precisamos consultar a posição "n", depois "n+1", "n-1" e assim sucessivamente até que a entrada seja encontrada ou que uma posição vazia o seja, indicando que a chave procurada não está mais na tabela.

É importante notar que quando uma entrada da tabela é apagada, a posição não pode ser marcada como vazia, mas sim como "apagada". Suponhamos que a entrada k_1 seja apagada. Se a posição "n" for marcada como vazia, então as entradas k_3 e k_2 nunca mais serão encontradas na tabela, pois como vimos acima, a procura por uma entrada termina quando uma posição vazia ou a própria entrada são encontradas. Desta forma, a posição deve ser marcada como "apagada" sinalizando para o mecanismo de escrita que a mesma pose ser utilizada novamente a ao mesmo tempo sinalizando para o mecanismo de procura que ainda podem existir chaves mais adiante.

Neste tipo de técnica, endereço aberto, podemos ver que o custo para procurar uma determinada chave na tabela é igual ao custo que foi para inseri-la, não importando o fator de ocupação da tabela. Mais adiante, iremos ver que com o emprego de técnicas mais sofisticadas, o custo de procura por uma determinada chave pode diminuir à medida em que o fator de ocupação da tabela diminui.

Em uma tabela do tipo endereçamento aberto, a questão da utilização de memória é bem definida. Como os elementos são diretamente armazenados nas posições da tabela, o número máximo de elementos armazenados é igual ao número de posições. Em sistemas onde o uso de memória é crítico e alocação dinâmica de memória é uma tarefa dispendiosa, este tipo de abordagem é interessante. Veremos a mais adiante, através do mecanismo de encadeamento, como é possível que uma tabela possua mais elementos do que posições disponíveis.

Amostragem linear

No exemplo acima, fixamos que o caminho a ser seguido caso a posição desejada já estivesse ocupada seguia uma seqüência linear (n+1, n-1, n+2, n-2, ...). Este tipo de amostragem é denominado "linear", descrito pela expressão a seguir, que usa o método da divisão, visto no capítulo anterior.

$h(k) = (h_1(k)+i) \bmod m$ para i=0, 1, ..., m-1

O maior problema desta técnica é que quando colisões ocorrem grandes espaços da tabela com posições ocupadas podem ser criados. Esse fenômeno é denominado de aglomeração (*clustering)*. Isso, à medida em que o fator de uso da tabela cresce, faz com que a performance da mesma se aproxime de uma busca seqüencial.

Para amenizar este problema, existem duas outras maneiras bastante conhecidas de realizar o amostragem , são elas: o método quadrático e o duplo *hash*.

Amostragem quadrática

No método quadrático, o intervalo entre as consultas, que antes era linear, agora é incrementado linearmente entre elas, ou seja, os índices são descritos por uma função quadrática, conforme abaixo:

$h(k) = (h_1(k)+i^2) \bmod m$ para $i = 0, 1,2, 3, ..., m-1$

Mesmo com o método quadrático, caso $h(k_1) == h(k_2)$, as seqüências para procura das posições de k_1 e k_2 são exatamente as mesmas, o que irá gerar a aglomerações, porém os mesmos estarão mais distribuídos pela tabela. Estas aglomerações são denominadas aglomerações secundárias.

Método do quociente quadrático

De forma a eliminar o efeito das aglomerações secundárias, Bell propôs uma modificação no método quadrático [16]. Bell propôs que o método quadrático, outrora proposto por Maurer [57], utilizasse uma função de deslocamento que fosse variável de chave para chave, mesmo nos

casos em que o *hash* de chaves distintas colidem entre si.. Denominamos de função de deslocamento a parcela que é somada ao *hash* da chave propriamente dito quando existe uma colisão. Por exemplo, no caso específico do método quadrático proposto em [57] temos:

$h_i(K) = h_0(K) + a \times i + b \times i^2$

Neste caso, a função de deslocamento é $a \times i + b \times i^2$, onde *a* e *b* são constantes, o que torna a função constante para o mesmo indexador *i* e chaves diferentes. O que Bell propôs foi uma modificação no método obtendo a seguinte função de *hash*:

$h_i(K) = h_0(K) + a \times i + \mathbf{b(K)} \times i^2$

De forma a não degradar o desempenho do método, a função b(K) deve ser rapidamente calculada. Quando o *hash* inicial é obtido pelo método da divisão, ou seja, $h_0(K) = K \bmod N$ (onde N é o tamanho da tabela) temos o quociente da divisão disponível sem custo adicional, podendo ser utilizado como a função b(K), parte da função de deslocamento.

Duplo hash

Para eliminar de vez o problema da aglomeração, o método de duplo *hash* pode ser empregado. Neste método, a próxima posição a ser consultada é calculada por uma segunda função de *hash*, como é mostrado abaixo:

$h(k) = (h_1(k) + i\, h_2(k)) \bmod m$

Sempre existe a possibilidade de que duas chaves k_1 e k_2 gerem o mesmo *hash*. Porém, dado que $h_1(k_1) == h_2(k_2)$, a probabilidade de que $h_2(k_1) == h_2(k_2)$ é muito menor. Em outras palavras, o método de duplo *hash* essencialmente elimina a possibilidade de duas chaves que colidem em uma posição tenham a mesma seqüência de amostragem. Para que isso seja verdade, uma boa função de *hash* deve ser escolhida. Isso pode depender do tipo de aplicação que está se desenvolvendo e quais os tipos esperados das entradas.

Com o suporte de módulos de hardware, o método de *hash* duplo pode se tornar muito interessante, dada a possibilidade de executar o *hash* $h_1(k)$ e $h_2(k)$ paralelamente no hardware e só utilizar o segundo *hash* se necessário. Em aplicações puramente de software, o método torna-se dispendioso devido à execução das duas funções de *hash* ser feita serialmente e não paralelamente.

7.5. TABELAS HASH POR ENCADEAMENTO

Em um determinado ponto no aumento da ocupação da tabela, o número de colisões e erros na consulta aumenta abruptamente. O método de encadeamento descrito neste capítulo faz com que o desempenho da consulta na tabela tenha uma degradação mais suave a medida em que o fator de ocupação aumenta (*load factor*).

O método de encadeamento requer uma pequena modificação na estrutura de dados. Ao invés de armazenar as entradas diretamente na lista, elas são armazenadas em listas encadeadas. Cada posição na tabela então aponta para estas listas encadeadas. Quando uma entrada é traduzida para uma posição, através da função de *hash*, ela é adicionada à lista encadeada que está contida naquela posição da tabela. Pela natureza das listas encadeadas, que não tem problema com tamanho, as colisões não são mais problema. A figura 7.5.1 ilustra uma tabela *hash* composta por listas encadeadas.

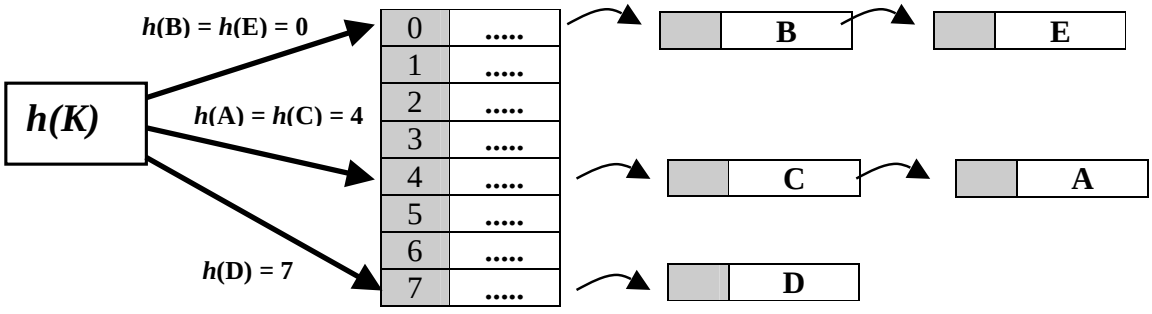


Figura 7.5.1 - Tabela *hash* com listas encadeadas

Em uma tabela que usa o método de encadeamento, podemos ter mais elementos armazenados do que posições disponíveis na tabela. Isso, na pior das hipóteses, atrasa o momento

em que haverá necessidade de redimensionar a tabela, o que é uma operação trabalhosa e com alto custo de tempo. Por outro lado, se muitos elementos forem mapeados para a mesma posição, teremos um aumento significativo do uso de memória, sem que a tabela esteja totalmente utilizada, ou seja, teremos muitas posições vazias (com memória evidentemente já alocada) mas com listas encadeadas compostas por muitos elementos.

7.6. MÉTODO HÍBRIDO - COALESCED HASH

A maior fraqueza das tabelas que empregam o método de endereçamento aberto está relacionada à aglomeração causada quando colisões ocorrem. A escolha de uma boa função de *hash* minimiza este problema, porém nem sempre isso é possível. Por outro lado, o método de encadeamento, pela própria natureza das listas encadeadas, insere um cabeçalho a cada uma das posições, que representa o ponteiro para onde a lista encadeada está localizada, aumentando a memória necessária para a implementação da tabela. Além disso, quando a memória reservada para o armazenamento das listas encadeadas é totalmente utilizada, todo o mecanismo entra em colapso, mesmo que nem todas as posições da tabela estejam ocupadas.

Olhando para as vantagens de cada método, temos um método híbrido [7] (*Coalesced hashing*) que aproveita o aspecto de otimização de espaço do endereçamento aberto e ainda conta com a minimização do efeito da aglomeração utilizando o encadeamento. Um método similar ao *Coalesced hashing*, denominado “Pseudo Encadeamento” é proposto por Halatsis e Philokyprouj em [10].

Como foi tratado anteriormente, no método de encadeamento, as posições da tabela servem não para armazenar as entradas, mas sim para armazenar ponteiros que indicam a localização de listas encadeadas, as quais contém as entradas propriamente ditas. Tais listas são armazenadas em locais diferentes da memória a qual tem sua ocupação incrementada à medida em que as colisões ocorrem.

No método *coalesced hashing*, também baseado em listas encadeadas, as próprias posições da tabela são usadas para armazenar as listas. Desta forma, temos uma melhor eficiência na utilização dos recursos de memória do sistema. O algoritmo para a inserção de um item na tabela funciona da seguinte forma: dada uma certa chave de entrada K, h(K) é calculada gerando o endereço onde K deve ser armazenada; se a posição em questão estiver vazia, então K é

armazenado nesta posição, caso contrário, *K* é armazenado na mais alta posição vazia da tabela e um link para esta posição é inserida em h(*K*).

Na tabela abaixo, as chaves do universo K foram inseridas na ordem alfabética, de A até H. As setas indicam o encadeamento das posições da tabela devido à ocorrência de colisões. Por exemplo: quando h(D) foi gerado para encontrar a posição onde a chave **D** deveria ser inserida, a posição em questão, posição 3, já estava ocupada pela chave **B.** Logo, **D** foi inserida na maior posição vazia da tabela, no caso, a posição 10, visto que a posição 11 já estava ocupada.

Chave K	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
h(K)	11	3	5	3	10	4	10	9

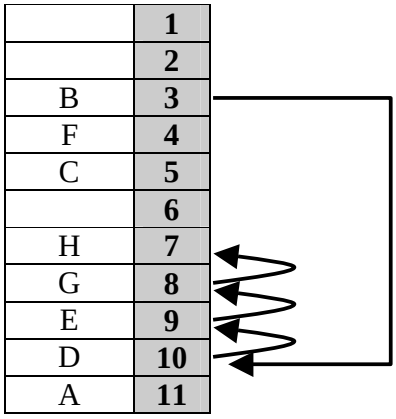


Figura 7.6.1 - Tabela *hash* com método coalesced hashing

Analisando a utilização da tabela da figura 7.6.1, podemos chegar a conclusão de que não há vantagem deste método em relação ao método clássico de endereçamento aberto com amostragem linear, uma vez que podemos claramente observar uma grande aglomeração de chaves. A aglomeração neste método é causada, assim como na amostragem linear, de duas formas: por chaves que colidiram pois geraram o mesmo *hash* e também por chaves cujas posições já estavam ocupadas por outras chaves já colididas.

Entretanto, as vantagens do *coalesced hashing* começam a aparecer quando um item da tabela é retirado. Na amostragem linear com endereçamento aberto, o comprimento máximo da seqüência de procura por um item não é alterada quando um item de uma aglomeração é retirado, salvo quando este item é o último da seqüência. Já neste caso, salvo quando o item retirado for o primeiro de uma cadeia, qualquer outro item retirado da cadeia diminui o custo máximo da procura em 1. Tenhamos como exemplo a mesma tabela preenchida anteriormente. Quando a chave E é retirada, a posição 9 é marcada como vaga e retirada da cadeia como mostra a figura

7.6.2. Desta forma, o custo para a procura pela chave G, que antes era de 3 passa a ser 2, uma vez que h(G) = 10.

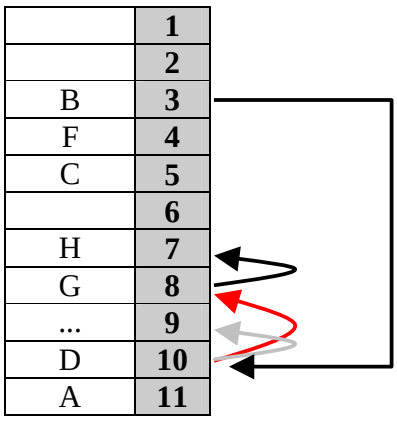


Figura 7.6.2 - Retirada de um item da tabela em *coalesced hashing*

A representação das posições de uma tabela *hash* que usa este método é bem simples, o que o torna ainda mais interessante. Cada campo pode ser composto como mostra a tabela 7.6.1, conforme sugere Vitter em [7].

Tabela 7.6.1 – Campos da tabela em *coalesced hashing*

Campos			
Indicador de campo vazio	*Chave*	*Outros campos*	*Link para encadeamento*

Na tabela 7.6.1 o "indicador de campo vazio" pode ter largura de apenas um bit para otimização de uso de memória e indica que a posição da tabela está vaga e uma nova entrada pode ser inserida neste endereço. O campo "chave" armazena a chave de entrada propriamente dita e o campo "link" serve para indicar o próximo item na cadeia.

Uma pequena variação no método descrito acima está na utilização de uma extensão da tabela *hash* com o intuito de armazenar somente as entradas que geraram colisão. Esta área é denominada "porão" (do inglês: *cellar*). Esta área previne que as entradas que geraram colisão

sejam armazenadas diretamente na tabela *hash* principal, ocasionando ainda mais colisões com chaves armazenadas mais tarde.

Se *M* é o número de endereços da tabela principal e *M'* é o número de endereços total da tabela, incluindo o porão, define-se a razão entre estas duas grandezas como fator de endereços $\beta = M / M'$. Para o caso apresentado acima, temos $\beta = 1$, ou seja, não existe área de porão. Vitter analisa em [7] os efeitos da variação de β, fator de endereços, na performance da tabela *hash* e conclui que, para a maior parte das aplicações, um fator de ocupação $\beta \cong 0{,}86$ é o valor ideal pois otimiza ao máximo a performance da procura em tabelas *hash* com uma vasta gama de fatores de ocupação.

7.7. TABELAS HASH EM REDES DE COMPUTADORES

Umas das tarefas mais críticas para o bom desempenho de equipamentos de comutação de pacotes em redes de computadores, seja no nível de enlace ou no nível de rede, é a classificação de pacotes [17]. No caso de roteadores, a escolha de qual rota o pacote deve seguir não é feita apenas pela consulta da existência do endereço IP de destino do pacote em uma tabela. Nestes casos é preciso pesquisar pelo prefixo mais longo referente ao endereço de destino. Ou seja, a entrada mais específica que coincida com o endereço. A procura pelo prefixo coincidente mais longo não é uma tarefa trivial e algumas propostas sugerem o uso de hardware auxiliar para a implementação deste mecanismo [3], [15].

Em equipamentos da camada 2, switches ethernet por exemplo, a tarefa de classificação dos pacotes é um pouco mais simples. Neste caso, uma tabela é consultada de forma a determinar se um endereço está ou não presente. Além da tarefa de comutação de pacotes entre as portas do equipamento, alguns equipamentos mais sofisticados disponibilizam a priorização de tráfego através da marcação dos pacotes em diferentes classes de serviços (CoS).

Assim como em roteadores, algumas propostas sugerem o uso de hardware para a implementação deste mecanismo de classificação dos pacotes, tais como memórias associativas (CAM's) ou projetos tais como [19].

7.8 CONCLUSÕES

Tabelas *hash* são o equivalente em software do que são as memórias CAM (*content addressable memory*) em hardware. Estas estruturas de dados oferecem uma grande vantagem no que se refere à custo em relação à suas equivalentes em hardware, por poderem ser implementadas utilizando algoritmos otimizados e com o uso de memórias RAM convencionais. Existem vários tipos de tabelas *hash*, sendo que cada tipo se adapta a um certo tipo de aplicação, o que torna o uso de tais estruturas ainda mais atrativo.

PARTE II – MATERIAIS E MÉTODOS

TECNOLOGIA DE COMPONENTES RECONFIGURÁVEIS

8. FIELD PROGRAMMABLE TECHNOLOGY – FPT

Nos dias de hoje o desenvolvimento de circuitos lógicos reconfiguráveis está muito avançado. Existem várias empresas nesta mercado, cada uma delas oferecendo uma grande gama de componentes em sua linha.

A utilização destes tipos de componentes no mercado eletro-eletrônico é bem ampla e variada. Em geral, quando uma grande porção de hardware dedicado a executar uma função específica é necessário, as FPGA's e CPLD's são a solução mais adotada. As FPGA's de altíssima densidade, estado da arte em componentes dessa natureza, em geral são utilizadas como plataforma de prototipação de ASIC's (*Application Specific Integrated Circuit*).

A diferença mais visível entre as FPGA's e CPLD's está no fato de neste último haver uma característica não volátil em relação à configuração do componente. Isso significa que CPLD's, após configuradas não perdem a configuração na falta de alimentação. Por outro lado, as FPGA's precisam que os bits de configuração sejam enviados ao componentes a cada vez que o mesmo é energizado, ou sua reconfiguração é forçada pelo usuário. Além disso, a arquitetura interna dos dois componentes também difere.

À medida em que a lógica interna destes componentes aumenta, também aumenta o número de bits de configuração dos mesmos. Desta forma, fica claro que para componentes de tamanho muito grande, as FPGA's levam vantagem, pois os bits de configuração do componente estão localizados fora do mesmo, em uma memória específica para este uso ou mesmo na memória do microprocessador principal do equipamento.

Como comparação, a tabela 8.1 mostra o número de bits de configuração de alguns componentes da Xilinx.

Tabela 8.1 – Número de bits de configuração

	FPGA Spartan3 XC3S50	**FPGA Spartan3E XC3S250E**	**FGPA Virtex4 XC4VLX200**
Bits de configuração	439.264 (~53k bytes)	1.353.728 (~165k bytes)	50.648.448 (~6M bytes)

8.1. FAMÍLIAS DE FPGA'S XILINX

A Xilinx [32], ao lado de Atera [31] e Lattice [33], é uma das líderes de mercado em circuitos de lógica programável. Foi fundada em 1984 por Ross Freeman, Bernie Vonderschmitt, e Jim Barnett.

Além de desenvolver e fabricar os circuitos integrados propriamente ditos, a empresa também investe pesado no desenvolvimento de ferramentas de software para disponibilizar suporte ao uso de seus circuitos lógicos programáveis. Além disso, a empresa disponibiliza suporte contínuo à seus clientes.

A empresa também disponibiliza, através de desenvolvimento próprio ou através de seus parceiros, *IP cores* para aplicações nos mais diferentes ramos do mercado, mais diferentes com as mais diferentes funções para aplicações como por exemplo: áudio, vídeo, processamento de imagens, interfaces, telecomunicações e redes de computadores, armazenamento de dados, operações matemáticas, processamento digital de sinais, entre outras.

A linha de circuitos programáveis da Xilinx vai desde as CPLD's (*Complex Programmable Logic Device*) até as densas FPGA's (*Field Programmable Gate Array*) da família Virtex.

8.1.1. CPLD's

A família de CPLD's está dividida em duas partes, a linha XC9500 e a linha *CoolRunner*. A linha XC9500 foi desenvolvida pela própria Xilinx e contempla os dispositivos de baixa densidade que podem ser usados como agregação de lógica em projetos de pequeno porte. Para cada componente, um determinado número de encapsulamentos está disponível para as aplicações que necessitam de mais pinos de entrada e saída. A tabela 8.1.1.1 mostra a densidade dos diferentes dispositivos desta família disponíveis nos dias de hoje.

Tabela 8.1.1.1 – Densidades das CPLD's Xilinx

Dispositivo	**XC9536XL**	**XC9572XL**	**XC95144XL**	**XC95288XL**
Macrocélulas	36	72	144	288
Portas úteis	800	1600	3200	6400

Como exemplo, em um dispositivo do tipo XC9572, com 1600 portas disponíveis, é possível implementar um codificador e um decodificador HDB3 com recuperador de relógio para uso em interfaces do tipo G.703 a 64k bits/s para aplicações em telecomunicações.

No final da década de 90, a Xilinx adquiriu o segmento de CPLD's da Philips, a família de CPLD's *CoolRunner*. Esta família de dispositivos utiliza uma tecnologia que provê baixo consumo e ao mesmo tempo alta performance. A tabela 8.1.1.2 mostra as opções de dispositivos disponíveis no mercado para a família *Cool Runner-II* .

Tabela 8.1.1.2 – Densidades das CPLD's *Cool Runner*

Dispositivo	**XC2C32A**	**XC2C32A**	**XC2C128**	**XC2C256**	**XC2C384**	**XC2C512**
Macrocélulas	32	64	128	256	384	512
Máximo de I/O's	33	64	100	184	240	270

8.1.2. FPGA's Spartan

A linha de FPGA's da Xilinx inicia-se com a família Spartan. Esta família vem se desenvolvendo ao longo do tempo. A família iniciou-se com a introdução dos dispositivos Spartan e em seguida a família Spartan2, Spartan2E, Spartan3 e Spartan3E. O mais novo grupo de dispositivos componentes da família Spartan é a Spartan3A, voltada a maior densidade de pinos de entrada e saída pois utilizam internamente 2 fileiras de pinos de I/O's.

Ao contrário das CPLD's, que não perdem o conteúdo de sua programação após o *power-down*, as FPGA necessitam ser configuradas a cada novo ciclo de alimentação. Os modos de configuração das FPGA's podem ser divididos em 2 grupos. Modos ativos (*master*) e modos passivos (*slave*), sendo que cada um deles ainda pode ser paralelo ou serial.

Em projetos onde existe um microprocessador externo os modos passivos são mais convenientes, uma vez que o momento mais conveniente para a configuração dispositivo pode ser escolhido. Além disso, várias configurações diferentes, podem estar armazenadas na memória do microprocessador central. Assim, dependendo do modo de operação em uso do equipamento, a CPU pode configurar a FPGA com um arquivo ou com outro, de acordo com a necessidade.

Nos modos ativos, após a estabilização da tensão de alimentação, o próprio dispositivo lê os bits de configuração de uma memória externa conectada a pinos dedicados. No passado, apenas as memórias PROM da própria Xilinx ofereciam suporte a este tipo de mecanismo. Em geral, estas memórias tinham um custo elevado em relação ao próprio dispositivo o qual estavam programando. Com o surgimento da família Spartan3E a Xilinx, após muitos anos, finalmente colocou em seus dispositivos um mecanismo capaz de ler os bits de configuração a partir de memórias *flash* paralelas de 8 ou 16 bits de dados. Além disso, a Xilinx disponibilizou também a configuração através de memórias *flash* do tipo SPI.

A família Spartan3E é composta por dispositivos de 5 diferentes densidades. Cada um deles disponível em vários encapsulamentos diferentes. A tabela 8.1.2.1 mostra as principais diferenças na capacidade de cada um dos componentes da família.

Tabela 8.1.2.1 – Comparativo entre FPGA's Spartan 3

Dispositivo	Portas	Células Lógicas	Bits de RAM Distribuída	Bits de Block RAM's	Multiplicadores	DCM's	Máx. pinos de I/O
XC3S100E	100K	2,160	15K	72K	4	2	108
XC3S250E	250K	5,508	38K	216K	12	4	172
XC3S500E	500K	10,476	73K	360K	20	4	232
XC3S1200E	1200K	19,512	136K	504K	28	8	304
XC3S1600E	1600K	33,192	231K	648K	36	8	376

8.1.3 FPGA's Virtex

Atualmente a linha Virtex de FPGA's da Xilinx está representada pela Virtex4 e Virtex5. Estas FPGA's são dispositivos, em sua maioria, com uma densidade muito superior às FPGA's da família Spartan. Além disso, são componentes com um grande espaço de memória RAM interna. O maior dos dispositivos da família Virtex4 possui mais de 200 mil células lógicas, aproximadamente 6M bits de ram interna (*block RAM)* e quando utilizada no encapsulamento FF1513, disponibiliza nada mais nada menos do que 960 pinos de entrada e saída para o usuário.

Tendo em vista a larga utilização destes componentes em aplicações onde é necessária a conexão com redes de computadores, a Xilinx acrescentou no design um controlado MAC

interno. Em dispositivos da família Spartan por exemplo, quando é necessária a utilização de um controlador MAC, um considerável espaço de lógica é utilizado para tal.

Os componentes da família de FPGA's Virtex4 são agrupados em 3 conjuntos de componentes. Cada conjunto tem foco em determinadas aplicações e em vista disso tem implementado em maior número uma determinada característica. Estas divisões são:

- Virtex4 LX: dispositivos voltados a alta densidade de lógica de alta performance. O maior dos dispositivos deste grupo possui mais de 200 mil células lógicas.
- Virtex4 SX: dispositivos principalmente voltados ao processamento digital de sinais (DSP).
- Virtex4 FX: os dispositivos deste grupo são voltados ao processamento embarcado e a conectividade serial.

No grupo de componentes FX da Virtex4, os dispositivos possuem internamente até dois núcleos do microprocessador PowerPC. Visto que neste grupo o suporte a processamento embarcado é a prioridade, as FPGA's Virtex4 FX oferecem ainda mais bits de RAM interna, podendo chegar a 10M bits no maior dispositivo.

A tabela 8.1.3.1 mostra as principais características dos componentes da família Virtex4 LX, a família mais popular e que possui dispositivos de uso geral.

Tabela 8.1.3.1 – Comparativo entre FPGA's Virtex 4

Dispositivo/ Característica	**XC 4VLX15**	**XC 4VLX25**	**XC 4VLX40**	**XC 4VLX60**	**XC 4VLX80**	**XC 4VLX100**	**XC 4VLX160**	**XC 4VLX200**
Células Lógicas	13,824	24,192	41,472	59,904	80,640	110,592	152,064	200,448
Bits de RAM	864k	1,296k	1,728k	2,880k	3,600k	4,320k	5,184k	6,048k
DCM's	4	8	8	8	12	12	12	12
Divisores de clock	0	4	4	4	8	8	8	8
XtremeDSP	32	48	64	64	80	96	96	96

Na tabela 8.1.3.1, podemos notar a presença dos componentes internos chamados XtremeDSP. Estes componentes são blocos modulares altamente específicos que dão

desempenho às funções de DSP dentro da FPGA. Estes *slices* só estão disponíveis na família de FPGA's Virtex4 sendo que dentre suas funções especializadas encontra-se hardware para aplicações de vídeo como compressão MPEG-4.

Recentemente a Xilinx lançou no mercado a família de dispositivos Virtex5. Algumas avanços tecnológicos foram incorporados a esta nova família, o que dão a ela uma pequena vantagem de desempenho em relação a sua antecessora, a Virtex4. Fabricada em um processo de 65nm contra os 90nm da Virtex4 e com tensão de alimentação do núcleo de 1volt apenas, a família Virtex5 pode ter seus blocos de DSP operando a uma freqüência de até 550MHz.

Os dispositivos mais densos desta família possuem mais de 300 mil células lógicas e memória interna de até 11M bits (*block RAM*). Quando utilizados no encapsulamento FF1760, podem disponibilizar até 1200 pinos entrada e saída para o usuário.

Além destas novas características, a família Virtex5 também incorporou a capacidade de obter os bits de configuração através de memórias paralelas de 8 ou 16 bits comerciais, além das memórias tipo SPI, assim como nas FPGA's da família Spartan3E.

8.2 SOFT PROCESSORS

Um *soft processor* é uma propriedade intelectual (IP) implementada usando primitivas de lógicas de FPGA [30]. Os maiores benefícios desta abordagem incluem a facilidade de mudança de configuração do sistema de forma a dar maior prioridade a preço ou performance, menor tempo de desenvolvimento, fácil integração com os circuitos da FPGA e a inexistência do risco da obsolescência. Em um determinado projeto, talvez seja interessante ter-se um número maior de portas seriais assíncronas (UART), sendo que os microprocessadores disponíveis no mercado e que possuem este número de portas não contemplam outras funcionalidades também necessárias ao projeto.

Assim como a facilidade de adicionar ou retirar periféricos, também é possível ajustar o desempenho do microprocessador através do dimensionamento de barramentos e com otimizações do núcleo IP disponibilizadas ao longo do tempo.

Da mesma forma não podemos deixar de destacar as vantagens oferecidas pelo microprocessadores convencionais existentes no mercado. Uma das maiores vantagens no uso de tais processadores é sua larga escala de utilização, o que acaba por gerar uma gama bastante

grande de fornecedores de ferramentas relacionadas e tais processadores, o que no caso dos *soft processors* fica mais limitada.

8.2.1. MicroBlaze

No segmento de processamento embarcado para aplicações variadas, a Xilinx disponibiliza o *soft processor* MicroBlaze [1]. A empresa investe bastante em marketing, seminários, documentação e desenvolvimento de ferramentas para atrair seus clientes ao uso deste tipo de microprocessador. O Microblaze é comercializado pela Xilinx como parte do software EDK [34]. Este software é na realidade um conjunto de vários aplicativos já desenvolvido pela Xilinx e que oferecem suporte ao desenvolvimento de projetos baseados em lógica programável, sejam eles projetos que envolvam o uso de *soft processors* ou não.

A ocupação de lógica do MicroBalze em um dispositivo é de aproximadamente 800 a 2600 *look-up tables* (LUT's). Alguns dos sistemas operacionais já portados para o MicroBlaze incluem o *Nucleus* da Mentor Graphics, o ThreadX, o uC/OS-II da Micrium, uClinux da LinuxWorks, dentre outros.

No caso específico de uma FPGA Spartan3E, uma LUT faz parte de um bloco maior denominado *Configurable Logic Block* (CLB). Uma LUT é a unidade básica responsável por desempenhar as funções de lógica da FPGA. Ela consiste em um gerador de funções baseado em memória RAM. Além de desempenhar as funções de lógica, a LUT pode ser configurada como uma memória distribuída dentro do dispositivo, caso o uso das memórias RAM's dedicadas (*block RAM's*) não seja possível ou o tamanho da memória seja muito pequeno que não justifique o uso de *block RAM's*. A LUT também pode ser configurada para ser utilizada como um registrador de deslocamento (*shift register*) de 16 bits. Cada LUT possui 4 entradas e uma única saída sendo que qualquer operação booleana de quatro entradas pode ser implementada com ela. Caso sejam necessárias mais entradas, é possível o uso de duas LUT's em cascata.

O MicroBalze é um processador RISC (*Reduced Instruction Set Computing*) de 32 bits, com arquitetura do tipo HARVARD. Este dispositivo é evidentemente otimizado para utilização nas FPGA's da Xilinx. A organização básica deste microprocessador consiste em 32 registros de uso geral, uma ULA (unidade lógica aritmética) e dois níveis de interrupções. A figura 8.2.1.1, retirada do guia de referência do MicroBlaze, mostra um diagrama em blocos do núcleo do microprocessador.

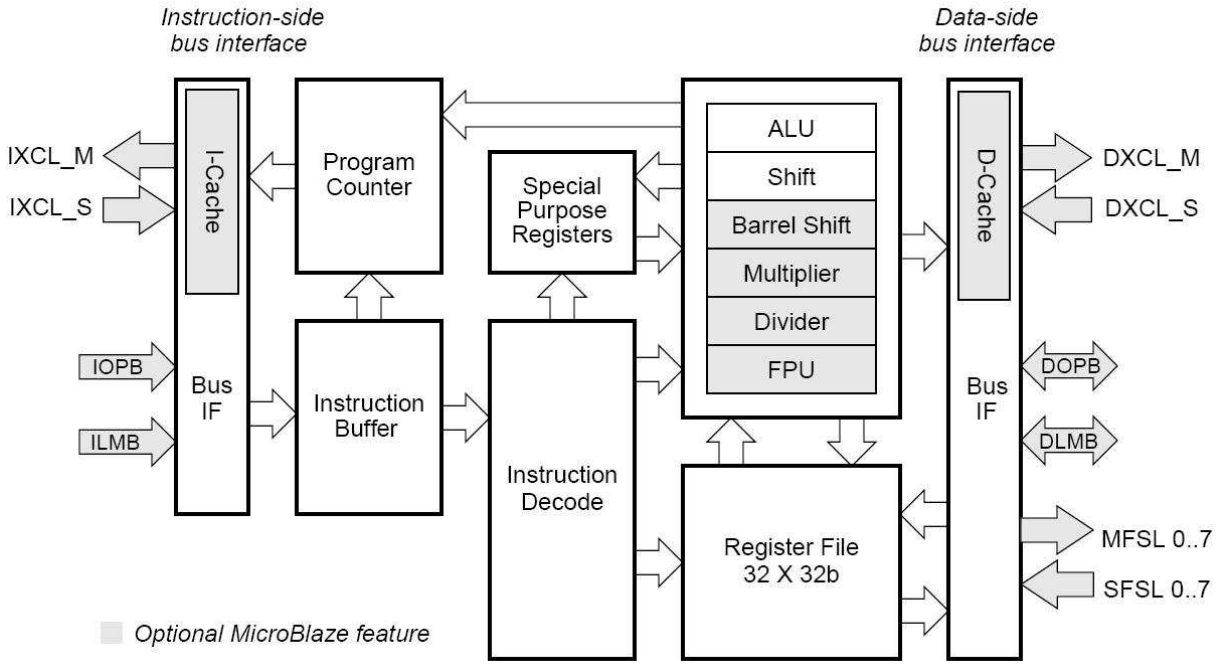


Figura 8.2.1.1 – Diagrama em blocos do MicroBlaze

Como abordamos anteriormente, a grande vantagem deste tipo de processador reside no fato de ser possível a inclusão de blocos de características de acordo com a necessidade da aplicação. Pelo diagrama em blocos, podemos observar que algumas partes do núcleo são opcionais. Estes blocos podem ser colocados ou retirados de acordo com a necessidade. Dentre estas características configuráveis estão:

- profundidade do *pipeline*
- lógica de debug de hardware
- memória cache de instrução
- memória cache de dados
- suporte à captura de exceções de hardware
- unidade de ponto flutuante
- divisor em hardware

A performance do *soft processor* MicroBlaze, varia de acordo com alguns pontos: configuração do processador (ex.: tamanho da cache), resultados da ferramenta de implementação

(problemas de *timing*), arquitetura da FPGA utilizada (Virtex, Spartan) e finalmente com a velocidade do próprio dispositivo, ou seja o *speed grade.* As FPGA's da família Spartan estão disponíveis em dois *speed grades*, -4 e –5, sendo que o –5 é representa um dispositivo mais rápido. A tabelas 11 e 12 mostram o melhor caso de performance das duas mais recentes versões do MicroBlaze executando Dhrystone [37] em diferentes dispositivos. Vale ressaltar que no casos abaixo a execução do código é feita a partir das memórias internas dos dispositivos (*block RAM's*) o que torna a execução extremamente rápida, porém, não representa a maior parte dos casos reais, onde o uso de memória interna ao dispositivo torna-se praticamente proibitivo dada a sua escassez.

Tabela 8.2.1.2 - Configuração do MicroBlaze v5.00 para FPGA's Virtex

FPGA	Ocupação	Freqüência de clock	Dhrystone 2.1	Performance
Virtex-5 (5VLX30-3)	960 LUTs	210 MHz	240 DMIPS	1.15 DMIPS/MHz
Virtex-4 Pro (4VLX25-12)	1,700 LUTs	160 MHz	184 DMIPS	1.15 DMIPS/MHz

Tabela 8.2.1.3 - Configuração do MicroBlaze v4.00 para FPGA's Spartan

FPGA	Ocupação	Freqüência de clock	Dhrystone 2.1	Performance
Spartan-3 (3S1500-5)	1,230 LUTs	100 MHz	92 DMIPS	0.92 DMIPS/MHz

Usando uma FPGA da família Virtex5, com muitas das funções do MicroBlaze v5.00 desabilitadas (não colocadas no hardware) pode-se chegar a uma ocupação de aproximadamente 800 LUT's. No caso da versão v4.00 do MicroBlaze, o mínimo utilizado são 950 LUT's

8.2.2. Picoblaze

Para aplicações mais simples, ou que sejam extremamente especializadas a Xilinx oferece o PicoBlaze [2]. O PicoBlaze é um processador com core RISC de 8 bits compacto voltado a custo. Ao contrário do MicroBlaze, este microprocessador é fornecido como um arquivo em

VHDL ou Verilog sem royalty para uso tanto em FPGA's como em CPLD's. Desta forma ele fica imune a obsolescência de produtos, uma vez que o código fonte é aberto.

O PicoBlaze é fornecido sob a forma do arquivo KCPSM3.vhd. A razão disso reside no fato de que o desenvolvimento original foi feito pelo engenheiro da Xilinx Ken Chapman e o projeto se chamava "Ken Chapman's Programmable State Machine", daí KCPSM.

A utilização de recursos da FPGA pelo PicoBlaze é tão pequena que é possível, em aplicações mais complexas, a instanciação de vários destes componentes dentro do dispositivo de modo a executar uma tarefa mais sofisticada, onde cada processador fica responsável por uma parte da tarefa.

A figura 8.2.2.1 mostra o diagrama em blocos do microprocessador PicoBlaze.

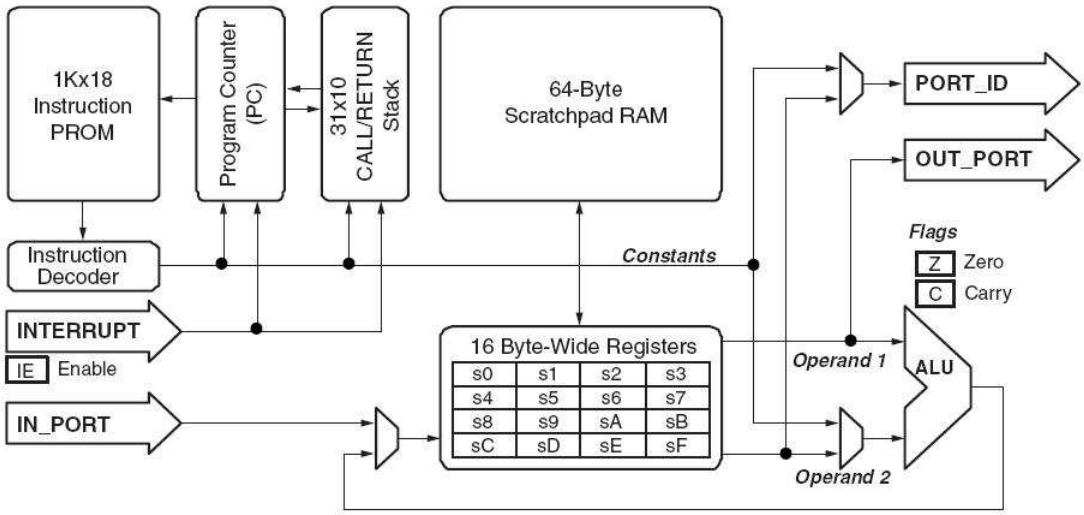


Figura 8.2.2.1 - Diagrama em blocos do PicoBlaze da Xilinx

As principais características do PicoBlaze são:

- 16 registradores de 8 bits para uso geral
- memória de dados de 1024 bytes automaticamente carregada durante a configuração do dispositivo
- ULA de 8 bits com flags de carry e zero
- 256 portas de entrada mais 256 portas de saída
- RAM interna de 64 bytes
- performance previsível, sempre 2 ciclos de clock por instrução

- resposta rápida à interrupções, máximo de 5 ciclos de clock

Assim como acontece no MicroBlaze, a performance do PicoBlaze é afetada principalmente pela família de FPGA's onde o mesmo está implementado. Existem 3 diferentes versões deste processador, cada uma delas otimizada para uma determinado grupo de famílias de componentes. A tabela 8.2.2.1 mostra as diferenças em performance e em recursos das diferentes versões do PicoBlaze.

Tabela 8.2.2.1 – Picoblaze implementado em diferentes dispositivos

Característica	PicoBlaze para Spartan3 Virtex-II/Pro e Virtex-4	PicoBlaze para VirtexE e SpartanII/E	PicoBlaze para CoolRunnerII
Memória de código	1024	256	256
Largura da instrução	18 bits	16 bits	16 bits
Registradores de 8 bits	16	16	8
Profundidade da pilha	31	15	4
Ocupação	96 slices na Spartan 3	76 Slices na Spartan II E	212 Macrocélulas na XC2C256
Performance	44 MIPS (Spartan-3) 76 MIPS (Virtex-II) 100 MIPS (Virtex-II Pro) 100 MIPS (Virtex-4 LX, SX) 102 MIPS (Virtex-4 FX)	37 MIPS (Spartan-IIE)	21 MIPS

É extremamente simples a instanciação do PicoBlaze dentro do código VHDL. O componente possui penas 11 sinais ou grupos de sinais de entrada e saída. Estes sinais devem então ser conectados aos sinais de interrupção, barramento da memória ROM (implementado através de uma memória do tipo block RAM), etc.

Uma das desvantagens do PicoBlaze está no fato de a Xilinx apenas fornecer um *assembler* para este processador, ou seja, o desenvolvimento do software deve ser feito com linguagem de mnemônicos *assembly*. Existem muito poucos compiladores em C para este processador. A maior parte dos compiladores disponíveis na Internet foram desenvolvidos por estudantes ou desenvolvedores autônomos.

8.3. MAC ETHERNET

Todos os grandes fabricantes de FPGA disponibilizam, gratuitamente ou não, núcleos IP que desempenham a função de controlador de acesso ao meio físico ethernet. É possível, ainda, encontrar no mercado empresas especializadas no desenvolvimento deste tipo de produtos, porém a um custo financeiro.

A Xilinx possui uma série de núcleos IP para o desempenho de funções relativas ao tratamento de pacotes ethernet, alguns deles são disponibilizados sem custo, porém outros necessitam de licença para serem utilizados. Os principais controladores ethernet na forma de núcleos IP são o *OPB Ethernet Lite Media Access Controller (EMAC Lite)* e o *OPB Ethernet Media Access Controller (EMAC).* Algumas das principais diferenças entre o EMAC e o EMAC Lite estão no fato que o último não permite a operação no modo promíscuo e também não possui suporte à recepção de pacotes com tamanho maior do que 1518 bytes (pacotes *jumbo*). A utilização do controlador MAC no modo promíscuo é imprescindível para a aplicação de *bridge* ethernet. No modo promíscuo, muito embora o controlador possa ter um endereço ethernet, todos os pacotes, não importando o endereço de destino, são recebidos pelo controlado e acessíveis pelo usuário.

Algumas das principais características do EMAC são listadas a seguir:

- operação a 10 ou 100M bits/s
- capacidade de *loopback* interna
- suporte a quadros *jumbo* com até 9k bytes de tamanho
- opção de inserção automática de FCS (*frame check sequence*) na transmissão
- processamento automático de quadros PAUSE
- suporte a recepção de quadros com etiqueta de VLAN
- suporte a gerenciamento do PHY externo
- contadores de erro e interrupções para estatísticas

A comunicação com ambos os controladores é feita através do barramento OPB (*on-chip peripheral bus*). O EMAC está disponível para todas as FPGA's das famílias Spartan e Virtex. A figura 8.3.1, mostra um diagrama em blocos do controlador MAC (EMAC) da Xilinx.

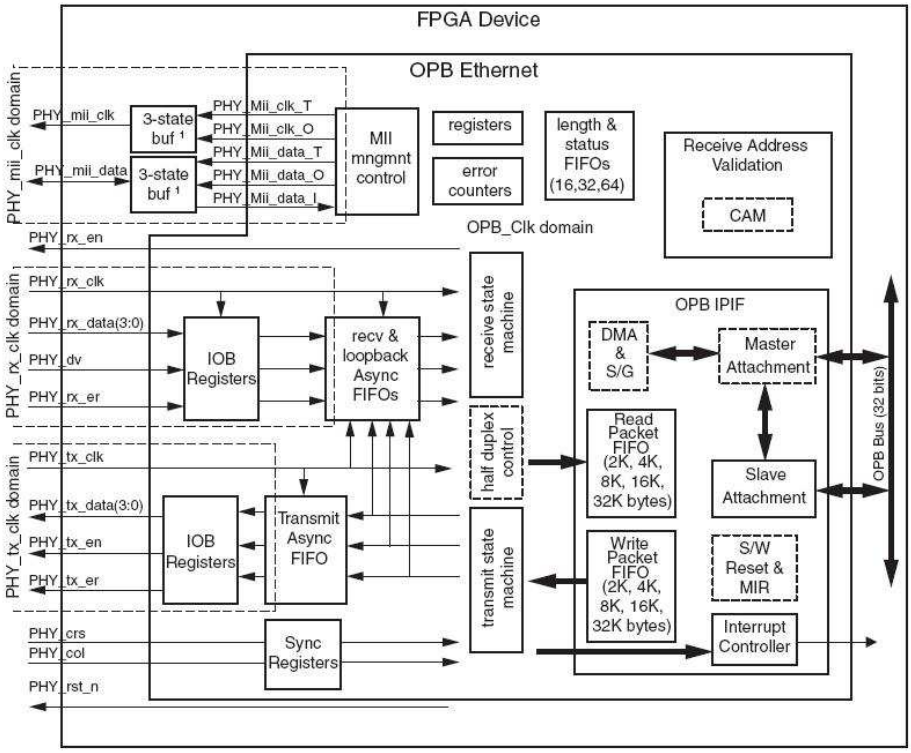


Figura 8.3.1 - Diagrama em blocos do MAC Ethernet da Xilinx

A tabela 8.3.1 mostra a utilização máximo e mínima de recursos pelo EMAC em uma FPGA.

Tabela 8.3.1 – Consumo de lógica pelo núcleo EMAC

Recursos utilizados		
	Mínimo	Máximo
IOBs	13	19
LUTs	1960	6157
FFs	1538	3198
Block RAM's	5	32

Cada vez mais os equipamentos eletrônicos tendem a ter suporte a protocolos de rede em vista do aumento da conectividade e facilidade de configuração e controle. Neste mercado, ethernet é a tecnologia dominante. Dentro da própria Pontifícia Universidade Católica do RS já

foram desenvolvidas pesquisas para o desenvolvimento de um núcleo IP com função de controlador de acesso ao meio físico ethernet [44]. O núcleo em questão foi desenvolvido totalmente em VHDL utilizando placas de desenvolvimento fornecidas por terceiros e foi dividido em duas partes principais. Um módulo de comunicação implementando uma interface com o sistema do usuário, no caso um computador utilizando porta serial assíncrona, e o controlador de acesso ao meio propriamente dito. O controlador tem suporte à taxas de 10 e 100 Mbits/s e tamanho de pacote de até 1500 bytes, não apresentando suporte à quadros jumbo, que podem ter tamanho de até 9000 bytes. As figuras 8.3.2 e 8.3.3 apresentam os diagramas em blocos da parte da recepção e da transmissão do controlador de acesso ao meio físico.

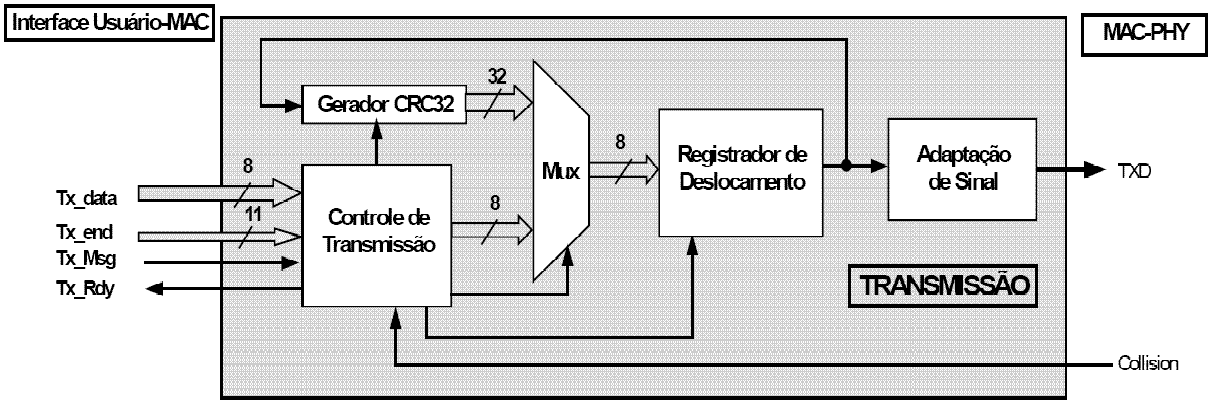


Figura 8.3.2 - Diagrama em blocos da transmissão do MAC

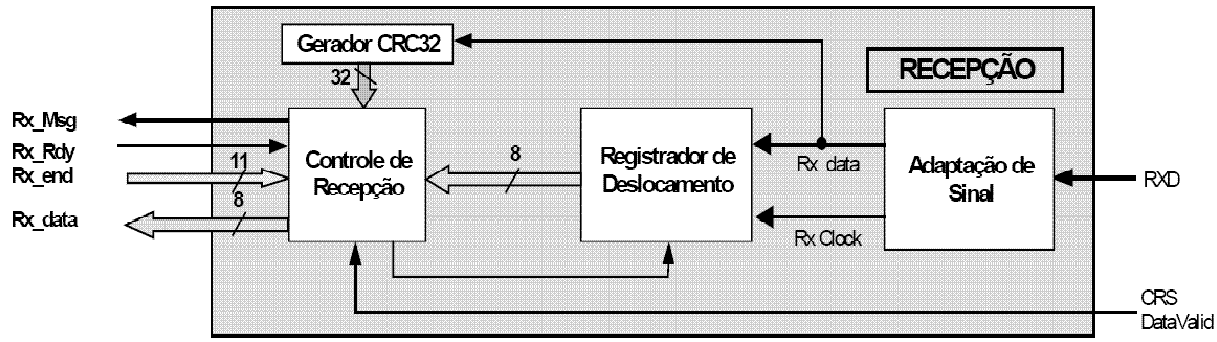


Figura 8.3.3 - Diagrama em blocos da recepção do MAC

8.4 CONCLUSÕES

A Xilinx, assim como outros fabricantes de circuitos integrados de lógica programável, provê uma gama de dispositivos das mais variadas capacidades (densidades) que podem ser utilizados nas mais diversas aplicações do mercado e de pesquisa.

Além da variada densidade dos componentes oferecidos, algumas características os fazem mais focados para uma determinada aplicação, como por exemplo, a presença de blocos de DSP. Dessa forma, é possível escolher o componente que melhor se enquadre na aplicação final de forma a otimizar a performance e minimizar os custos.

Nos dias de hoje, o apelo principal dos fabricantes de dispositivos de lógica programável é, dado o aumento na densidade dos componentes, a possibilidade da implementação de processadores através de núcleos de propriedade intelectual. Tais processadores podem variar desde um simples núcleo de 8 bits, até um complexo processador de 32 bits com arquitetura RISC.

9. CONTENT ADDRESSABLE MEMORY (CAM)

9.1. DEFINIÇÃO E APLICAÇÕES

Memórias de acesso randômico (RAM's) são matrizes de dados que são indexadas através de linhas de endereço. Em geral, cada posição do *array*, ao ser acessada, pode ser lida ou escrita, dependendo do estado dos sinais de controle da memória. Para realizar-se uma busca por um determinado dado em uma RAM, é necessária, no pior caso, a consulta a todos as posições do array (com sorte o dado procurado estará nas primeiras posições consultadas). Esse tipo de mecanismo acaba acarretando dois problemas em aplicações de alta velocidade como a comutação de pacotes.

O primeiro problema é bastante evidente, quanto maior a memória, maior o tempo máximo esperado para o término da consulta, uma vez que no pior caso toda a memória deve ser consultada. O segundo problema está no fato de o tempo de consulta não ser previsível, visto que o dado procurado pode estar em uma das primeiras posições consultadas assim como em uma das últimas. Este problema poderia ser resolvido através da inclusão de ciclos extras em casos de que o custo de procura foi pequena, porém, esta não parece uma solução elegante.

No caso das CAM's, ao invés de apresentarmos um endereço e a memória apresentar em sua saída o dado presente naquele endereço, a operação oposta é realizada. Um dado é apresentado à memória e a mesma informa se esta dado está, ou não, armazenado em alguma de suas posições. Esta operação é realizada através da consulta paralela a todos os endereços da memória.

Muito embora a CAM execute em hardware a procura em um array, de uma forma geral a utilização da CAM envolve o uso de alguma parcela de software. Quanto mais complexa a aplicação de procura ou organização de dados, maior a parcela de software utilizada em conjunto com a CAM. Vários estudos são feitos hoje em dia com o intuito de criar algoritmos otimizados, mais eficientes no que diz respeito à velocidade, utilizando a ajuda de CAM's, assim como aqueles descritos em [46], onde alguns problemas clássicos de aplicações em redes de computadores são abordados, tais como a procura do número de uma porta TCP ou UDP dentro de um conjunto de faixas de portas.

Existem basicamente dois tipos de CAM's, as binárias e as ternárias. As do tipo binárias suportam a consulta a dados compostos por dígitos binários, 0 ou 1 apenas. Em redes IP, para

determinar-se o caminho de destino de um pacote, a tabela de roteamento deve ser consultada. Esta tabela define a qual *gateway* um pacote IP deve ser encaminhado baseado na rede à qual pertence o endereço de destino do mesmo. Para determinar a qual rede o endereço pertence, apenas uma parte dele é levada em conta, aquela parte sinalizada pela máscara de rede.

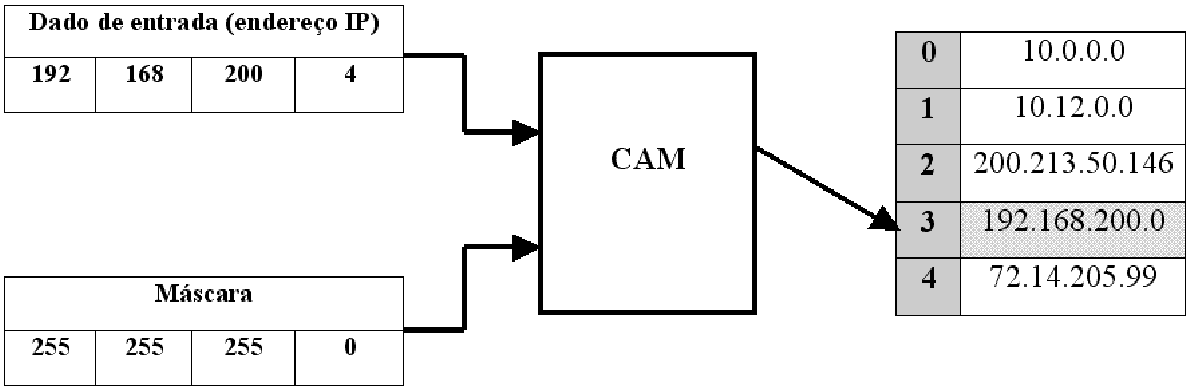


Figura 9.1.1 - Determinação do endereço de rede através da máscara

CAM's ternárias possuem, além do dado de entrada o qual se deseja consultar, uma segunda entrada representado uma espécie de máscara, o que permite o armazenamento e consulta de três tipos de dados, 0, 1 ou X (*don't care*). Desta forma, em aplicações de roteamento de pacotes IP, podemos apresentar à memória o endereço IP mas especificarmos que apenas uma parte do endereço é que deve ser levada em conta na hora da pesquisa.

Além do uso em sistemas de comutação de pacotes [3], CAM's ainda tem outras utilidades interessantes. Na compressão de dados, por exemplo, uma CAM pode ser usada para substituir seqüências comuns de serem encontradas por chaves, que na verdade representam o endereço onde as seqüências estão armazenadas. Como a consulta à memória é constante não importando o tamanho da mesma, pode-se aumentar o número de seqüências da base de dados sem a diminuição da performance.

Outra utilização clássica das CAM's é no mapeamento de memória em microprocessadores, como TLB's (*Translation Lookaside Buffer*).

Nada impede que um mesmo dado seja armazenado em duas posições diferentes da memória, logo, memórias com interfaces mais detalhadas podem informar se uma, mais de uma, ou nenhuma posição interna contém o dado procurado.

Dentre os usos clássicos de CAM's, está a comutação de células em switches ATM [51]. A comutação de células em um *switch* ATM se dá baseada na informação do indicador de caminho virtual (*virtual path indicator* – VPI) e do indicador de circuito virtual (*virtual circuit indicator* – VCI) presentes na célula ATM. O VPI possui o bits de tamanho enquanto que o VCI possui 16 bits. Este tipo de "endereçamento" da célula ATM é bem menor em tamanho do que o endereçamento de 48 bits que encontramos nos pacotes ethernet, o qual estamos tratando neste trabalho.

Assim como nas RAM's existe sempre uma contrapartida em relação ao aumento da capacidade da memória. No caso da RAM, o aumento acarreta em um maior tempo de procura, nas CAM's o problema está no aumento significativo no hardware necessário.

9.2. CAM'S EM FPGA'S

Todos os grandes fabricantes de FPGA's possuem suporte ao desenvolvimento de CAM's em seus dispositivos. No caso da Xilinx, o núcleo IP disponibilizado atualmente é o "Content-Addressable Memory v5.1". Na forma do componente mostrado na figura 9.2.1, a memória pode ser instanciada com largura de dados de 1 a 512 bits e profundidade de 16 a 4096 posições. É claro que estes números variam de acordo com a tecnologia e densidade do dispositivo utilizado.

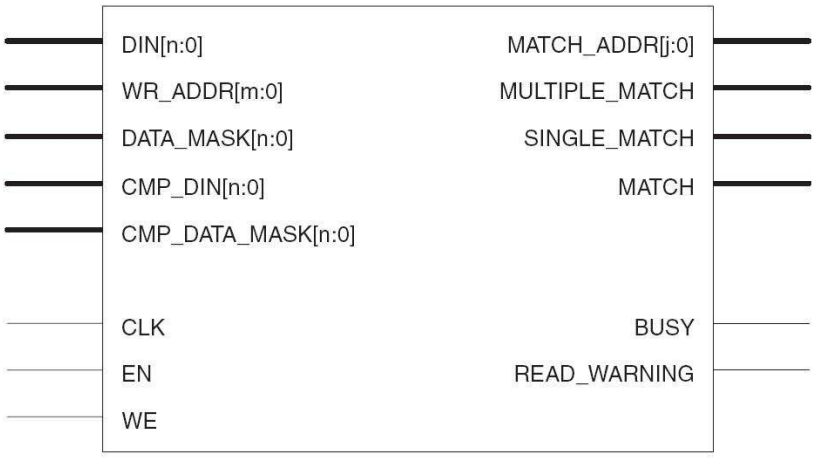


Figura 9.2.1 - Símbolo do núcleo IP CAM

Este núcleo IP está disponível para as famílias Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE e Spartan-3. Duas formas distintas de implementação da CAM são disponíveis com o núcleo IP em questão. A primeira é baseada no uso de BRAM's. Neste mecanismo, a CAM gerada é extremamente rápida e tem uma latência de apenas um ciclo de relógio para operações de escrita e leitura.

A segunda forma de implementação da CAM é baseada no uso de registradores de deslocamento. Esse sistema é indicado para projetos onde as BRAM's são um recurso escasso. Devido ao uso de registradores de deslocamento, a CAM gerada possui uma latência de um ciclo de relógio para operações de leitura e 16 ciclos de relógio para operações de escrita. Uma vantagem na implementação com registradores de deslocamento é que neste caso é possível que a CAM seja do tipo ternária.

9.3. CONSUMO DE LÓGICA E BRAM

Para ilustrar o consumo de lógica na instanciação de uma CAM em uma FPGA da família Spartan3 temos as tabelas abaixo. Estas tabelas resumem o uso de recursos na instanciação de CAM's das duas topologias descritas anteriormente em uma FPGA de 200.000 portas da Família Spatan3 da Xilinx.

A tabela 9.3.1 mostra a utilização de recursos na instanciação de uma CAM de 32 bits de largura de dados e com 64 posições apenas. Esta topologia utiliza BRAM's para CAM implementar a CAM.

Tabela 9.3.1 - CAM binária 64 x 32

	Ocupação	% do total disponível
Slices	252	13 %
LUT's	416	10 %
BRAM	8	66 %
GCLK	1	12 %

Na tabela 9.3.2 são mostrados os números para a topologia que utiliza registradores de deslocamento na implementação de uma CAM ternária com a mesma capacidade da anterior.

Neste caso podemos observar que o número de LUT´s utilizadas na implementação cresceu. Isso se deve ao fato destas LUT's serem utilizadas na como registradores de deslocamento.

Tabela 9.3.2 - CAM ternária 64 x 32

	Ocupação	% do total disponível
Slices	781	40 %
LUT's	1480	38 %
BRAM	0	0 %
GCLK	1	12 %

Na tabela a 9.3.3 temos o caso de uma CAM com as mesmas características da anterior, porém com 128 posições. Neste caso, vemos que a lógica necessária para a implementação da CAM extrapolou a capacidade do componente utilizado.

Tabela 9.3.3 - CAM ternária 128 x 32

	Ocupação	% do total disponível
Slices	2504	130 % (*)
LUT's	2916	75 %
BRAM	0	0 %
GCLK	1	12 %

(*) Capacidade máxima do componente foi ultrapassada

Em uma FPGA da família Spartan3, um *slice* é formado por dois geradores de funções lógicas, dois elementos de armazenamento, multiplexadores, lógica de *carry*, além de portas aritméticas.

Podemos notar pela análise das tabelas 9.3.1 a 9.3.3 que o crescimento do uso de lógica interna não é linear em relação ao crescimento da capacidade de armazenamento da memória.

9.4 CONCLUSÕES

Sumarizando o que foi exposto acima, podemos justificar o abandono da idéia de uso de CAM's para auxiliar na tarefa de classificação e encaminhamento dos pacotes ethernet em nossa *bridge*.

CAM's comerciais são dispositivos não muito comuns, poucos fabricantes as produzem. Desta forma, estes tipos de memórias têm um alto preço no mercado, o que torna seu uso proibitivo. É provável que em aplicações como switches ATM, as CAM's sejam implementadas no próprio silício, juntamente com as outras funções do dispositivo de comutação.

Além disso, como discutimos anteriormente, qualquer implementação de CAM's seja ela em FPGA's ou em circuitos integrados dedicados, requer o uso de RAM's. Estas memórias, por sua vez, tem um alto valor em FPGA's e não são bastante abundantes, principalmente se nestas FPGA's houver a necessidade de instanciação de dispositivos que sozinhos já empreguem o uso de RAM, como o controlador ethernet e o controlador HDLC, além do próprio *softcore* MicroBlaze que também emprega BRAM's.

Na aplicação clássica de CAM's, switches ATM, o tamanho dos campos identificadores do destino da célula (VPI e VCI) é de 8 e 16 bits. No caso de quadros ethernet, o tamanho dos endereços de origem e destino é de 48 bits, o que requer o uso de CAM's de maior porte e complexidade.

PARTE III - IMPLEMENTAÇÃO E VALIDAÇÃO DA BRIDGE ETHERNET

10. IMPLEMENTAÇÃO

10.1. PROPOSTA DE IMPLEMENTAÇÃO DE BRIDGE EM FPGA USANDO SOFTPROCESSOR

O objetivo final deste trabalho é implementar uma *bridge* ethernet através do uso de um núcleo IP de um *soft processor*. O processador utilizado é o MicroBlaze da Xilinx [32]. As funções de classificação dos pacotes ethernet são executadas totalmente por software.

Com o constante crescimento na densidade de dispositivos de lógica programável, começa a se tornar viável a execução de tarefas antes desempenhadas por circuitos integrados dedicados e de custo elevado.

A síntese do hardware, da mesma forma que a compilação do software de aplicação, foi executada com o auxílio da ferramenta EDK [34], que é na realidade uma interface gráfica que facilita a utilização de uma série de outras ferramentas, como o clássico ISE (para síntese e implementação do hardware), o Impact (para configuração dos dispositivos), o XMD (para debug das aplicações de software), dentre outras.

10.1.1. Porque implementar a bridge em software sobre lógica programável?

Muitos circuitos integrados existentes no mercado executam as funções de *bridge* ethernet, tais como aqueles apresentados nos capítulos 2 e 4. Entretanto, em grande parte das aplicações em produtos, o circuito integrado utilizado provê funcionalidades além daquelas necessárias à aplicação onde o mesmo será inserido. Assim, ao sub-utilizarmos um determinado dispositivo, estaremos arcando com custos desnecessários. Aliado a isso, novas tecnologias, protocolos e técnicas que por ventura surjam durante a vida útil do dispositivo não podem ser implementadas no mesmo. Em situações ainda mais drásticas, na eventual descoberta de um problema na concepção do dispositivo que impeça seu uso em determinada aplicação, o risco financeiro, no sentido de perda de contratos ou concorrências por parte da empresa que o utiliza é muito grande.

Pelas razões citadas acima, já é perfeitamente possível justificar a utilização de um circuito de lógica programável para o desenvolvimento de uma *bridge* ethernet, uma vez que:

- os recursos podem ser utilizados na medida certa da necessidade, sendo que a lógica extra pode ser utilizada para outras funções dentro da FPGA.
- na eventual necessidade de implementação de novos protocolos ou expansões, circuitos de lógica programável provêm a flexibilidade necessária.
- problemas de operação detectados no cliente, uma vez que solucionados em laboratório, podem ser facilmente corrigidos através de atualizações de "firmware".

Pelos mesmos motivos citados acima, que justificam o uso de dispositivos de lógica programável para implementação da *bridge* – utilização de recursos, expansões e facilidade nas correções – podemos justificar a implementação das funções de comutação de pacotes da *bridge* em software e não por hardware. Com a implementação das funções em software, a realização das ações impostas pelas justificativas acima fica muito mais simplificada. Em geral, implementações em software têm manutenção mais fácil, sendo que o projeto realizado poderá ser facilmente difundido e utilizado e também melhorado por outros.

Uma *bridge* em software possui um throughput relativamente baixo em relação às bridges totalmente por hardware. Porém, o propósito desta *bridge* é ser colocada diretamente em um link WAN, normalmente formado por modens digitais de até 2Mbps, sendo que assim não deverá se tornar o gargalo da aplicação.

10.1.2. Descrição funcional simplificada

A tarefa da *bridge* é interligar dois segmentos de rede que estejam distantes entre si. Para desempenhar esta tarefa, a *bridge* necessita encaminhar os pacotes recebidos por sua interface ethernet (conectada ao segmento de rede local) até a *bridge* remota (conectada ao segmento de rede remoto). A *bridge* encapsula os quadros ethernet em quadros HDLC, inserindo um cabeçalho, de forma a facilitar a sua recepção serialmente pela *bridge* remota. A transmissão dos quadros HDLC propriamente ditos pode ser feita através de modens digitais síncronos, tais como HDSL, SHDSL ou banda-base.

A interface ethernet da *bridge* funciona no modo promíscuo, de forma que todos os pacotes recebidos por esta interface, após a checagem de integridade através do campo FCS, são encaminhados ao processador local através de interrupção. Todos os pacotes válidos (com FCS

correto) recebidos pelo controlador ethernet, são copiados para a área de memória de dados do processador. O cabeçalho do pacote é então analisado pela CPU e a decisão de enviar o pacote à *bridge* remota, através do controlador HDLC, é tomada. Caso o pacote deva ser transmitido para o segmento remoto, pelas razões abordadas no capítulo 2, o mesmo é copiado da área de memória do processador para o controlador HDLC, o qual insere o cabeçalho apropriado e o transmite através da linha de comunicação.

No caminho inverso, os pacotes recebidos da *bridge* remota através do controlador HDLC têm sua integridade validada e, caso estejam íntegros através de checagem de CRC, uma interrupção é gerada ao processador. Visto que a *bridge* remota já realizou à consulta à sua tabela de endereços e o único destino possível neste ponto para o pacote recebido pelo controlador HDLC é o segmento de rede local, não é necessária uma segunda consulta à tabela de endereços. Desta forma, o processo pode ser otimizado e toda a vez que o processador atende uma interrupção do controlador HDLC, os pacotes recebidos da *bridge* remota são encaminhados diretamente ao controlador ethernet.

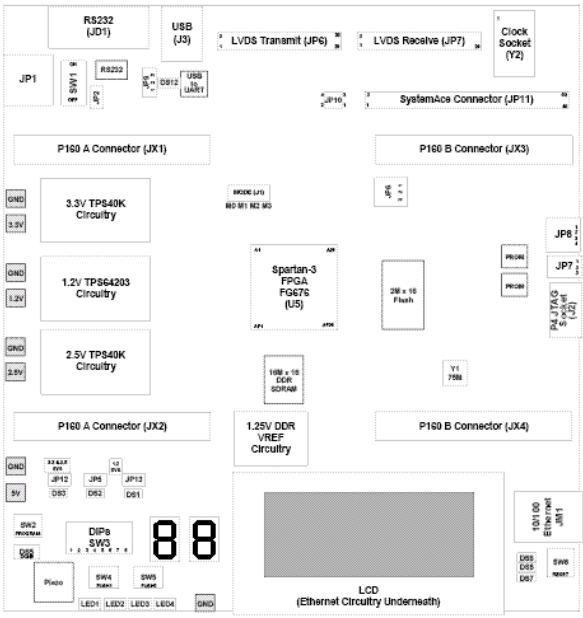
Tendo em vista a escassez de recursos de memória internamente à FPGA, a memória de dados e o código executado pela CPU são armazenados em um dispositivo externo de maior capacidade. Com isso, ganha-se em flexibilidade do código mas perde-se em velocidade de execução, uma vez que as memórias internas ao dispositivo podem ser acessadas muito mais rapidamente do que memórias externas, sujeitas à capacitâncias e indutâncias de linhas de transmissão das placas de circuito impresso.

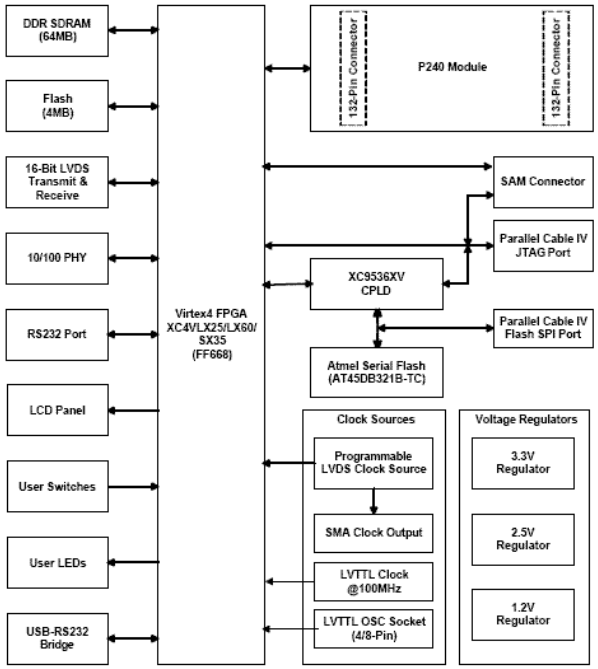
10.1.3. Detalhamento do hardware envolvido

A prototipagem da *bridge* foi realizada através do uso de dois kits com placas de desenvolvimento de FPGA's Xilinx. São elas: Spartan3 MB Development Board e Virtex-4 MB Development Board. Ambas fabricadas pela Memec.

Ambas as placas possuem todos os requisitos necessários à prototipagem da *bridge*. Espaço em lógica programável suficiente para acomodar todos os núcleos IP necessários, memória RAM DDR externa para execução do software, memória flash para eventual armazenamento do software, muito embora a interface de debug JTAG permita o download do software diretamente na RAM externa, PHY ethernet de 10/100 Mbits/s, interface RS232 para conexão de terminal, memórias flash seriais (Xilinx PROMs) para armazenamento da

configuração dos dispositivos, bem como sinais de uso geral ligados a conectores que facilitam a interconexão entre as duas placas, simulando um link digital. A figura 10.1.3.1 mostra os diagramas em bloco dos dois kits de desenvolvimento da Memec que foram utilizados.





Spartan3 MB Kit **Virtex-4 MB Kit**

Figura 10.1.3.1 – Diagrama em blocos das plataformas utilizadas

A figura 10.1.3.2 ilustra como a *bridge* foi prototipada, utilizando os kits em questão.

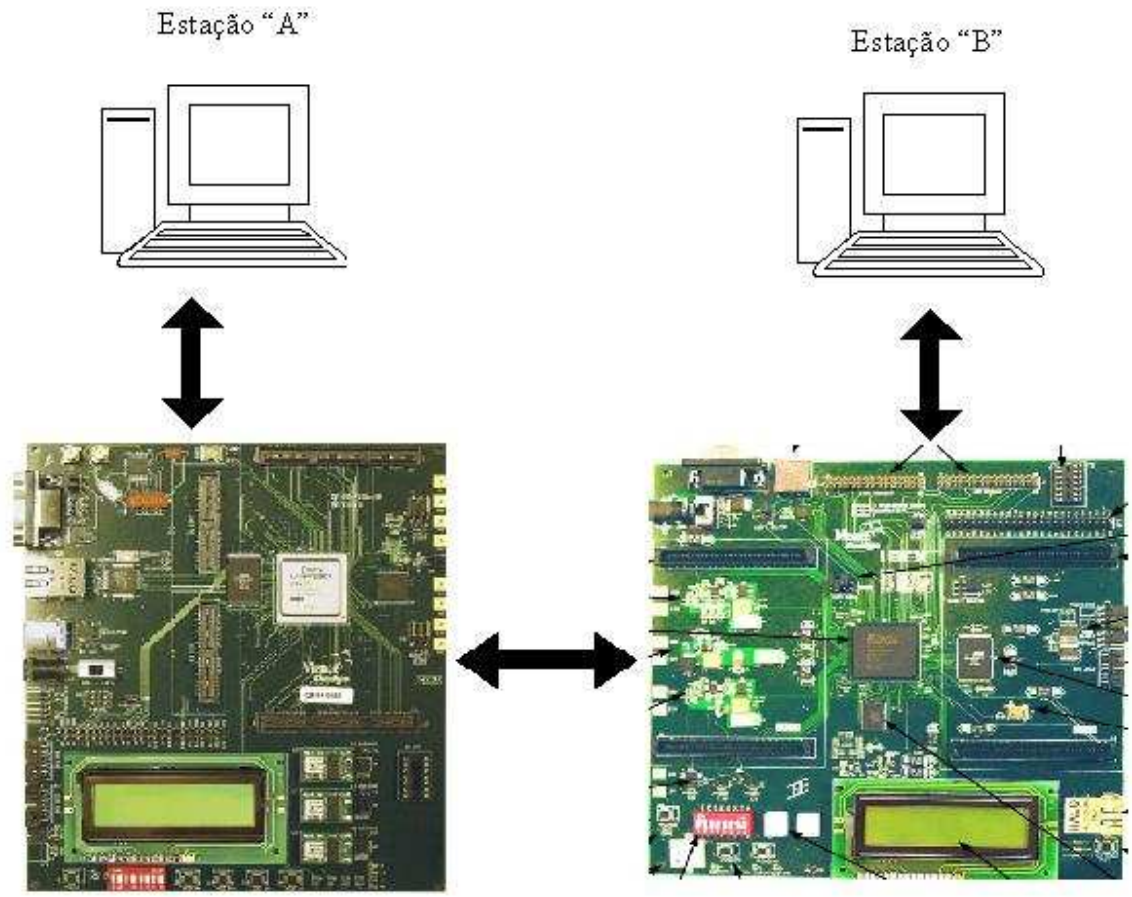


Figura 10.1.3.2 - Prototipagem da bridge

A simulação do link digital síncrono que interconecta as duas bridges foi feita usando-se um circuito muito simples. Um oscilador a cristal de 2048kHz - que serve como base de tempo para o circuito de transmissão e recepção dos controladores HDLC - e um par de fios de conexão que ligam os sinais de transmissão e recepção das duas bridges, conforme mostra a figura a 10.1.3.4.

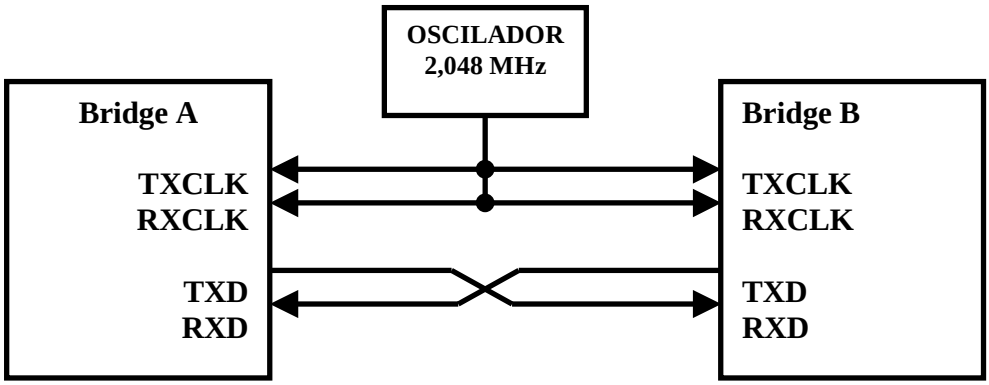


Figura 10.1.3.4 - Esquema para simulação do link digital

Internamente à FPGA, a *bridge* proposta é composta por 3 blocos principais:

- Microprocessador MicroBlaze
- MAC Ethernet
- Controlador HDLC

Todos os blocos estão ligados à CPU (MicroBlaze) através do barramento OPB (*on-chip peripheral bus*). Já abordamos as principais características dos núcleos IP EMAC e MicroBlaze. Assim, nos próximos parágrafos iremos detalhar o restante dos blocos envolvidos.

Controlador HDLC (OPB HDLC)

Umas das formas mais usuais de transmissão de pacotes IP/ethernet através de links WAN é através do encapsulamento dos mesmos em quadros HDLC. Na *bridge* proposta, a tarefa de encapsulamento dos pacotes tratados pela CPU em quadros HDLC é executada pela propriedade intelectual OPB HDLC, fornecido pela Xilinx através de seu kit de desenvolvimento EDK [34]. Este IP, na sua configuração mínima de hardware, ocupa 6 pinos de interconexão, 1203 LUTs, 604 flip-flops, e 2 blocos de memória RAM (block RAM).

A figura 10.1.3.5 mostra um diagrama em blocos do IP opb_hdlc

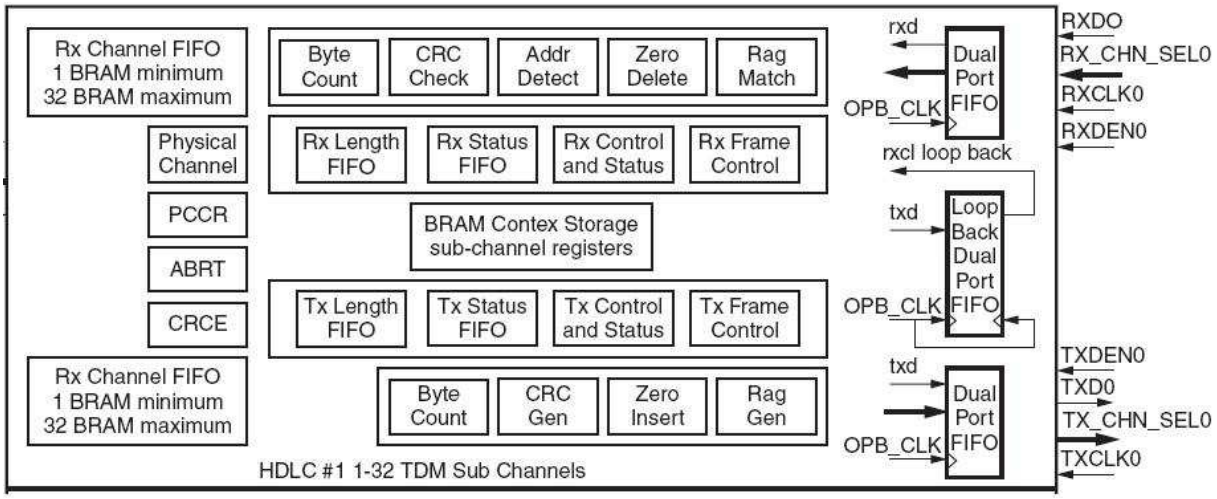


Figura 10.1.3.5 - Diagrama em blocos do controlador HDLC

A grande vantagem no uso deste núcleo IP para a tarefa de encapsulamento dos pacotes em quadros HDLC está no fato de o mesmo ser expansível no que se refere ao número de canais HDLC. Da forma como hoje o controlador está instanciado, apenas um canal está implementado, uma vez que a *bridge* possui apenas uma interface WAN. Porém, existem modens que podem ser conectados a dois equipamentos remotos simultaneamente através de duas linhas digitais. Assim, uma expansão de portas da *bridge* seria resumida a pequenas modificações no software e a instanciação de mais um canal no controlador HDLC. O número máximo de canais físicos é 8, sendo que cada canal físico pode ter até 32 sub-canais. O conceito de sub-canais permite a multiplexação no tempo do canal físico entre os vários sub-canais.

Outros dispositivos de hardware

De forma a otimizar o funcionamento do software, os controladores HDLC e ethernet funcionam através de interrupções. Quando um novo pacote é recebido, uma interrupção é gerada e o processador deve tratar o novo pacote. Da mesma forma, após a correta transmissão de um pacote, uma nova interrupção é gerada para informar à CPU da liberação dos buffers de envio. Como o MicroBlaze possui apenas uma entrada de interrupção externa, é necessário o uso de um controlador de interrupções. Este dispositivo, também conectado ao OPB, basicamente recebe todos os pedidos de interrupção dos periféricos e agrupa-os em apenas um sinal de interrupção, o qual é entregue à CPU. O núcleo IP utilizado para esta função é o ‘OPB Interrupt Controller’, que

possui suporte até 32 interrupções externas. Na configuração mínima de hardware, os recursos utilizados pelo controlador de interrupções são: 55 pinos de interconexão, 42 LUT's e 63 flip-flops.

A comunicação com o mundo externo, para obter informações de status e debug sobre a operação da *bridge* é feita através de uma porta serial. Como o núcleo do MicroBlaze não possui porta serial, um novo núcleo IP deve ser utilizado, neste caso o "OPB UART Lite". Este núcleo IP utiliza apenas 88 LUTs e 48 flip-flops. Embora não use recursos de BRAM, o núcleo IP possui duas FIFOs de 16 caracteres, uma para a transmissão e outra para a recepção. A desvantagem no uso deste núcleo é que a escolha do *baud rate* dos dados deve ser feita em tempo de síntese e não durante a operação.

A melhor performance no que tange à execução do software da *bridge* seria obtida através do uso da memórias interna do FPGA (BRAM), pois está integrada no mesmo silício onde reside o processador. Entretanto, este bloco de memória tem grande importância para uso em outras funções da *bridge*, como no controlador HDLC e controlador ethernet. Assim, devemos partir para a execução do software a partir de uma memória externa. A melhor opção, aquela que resulta em maior performance, é a execução a partir da memória RAM DDR externa ao FPGA. Para isso, é necessário a utilização de um controlador de memória SRAM DDR. Esta função está disponível no "OPB Double Data Rate Synchronous DRAM Controller". Este núcleo IP utiliza, na sua configuração mínima de hardware, 396 LUTs e 374 flip-flops.

De forma a facilitar o desenvolvimento do software, o MicroBlaze pode ser sintetizado juntamente com um hardware de debug. O núcleo IP "Microprocessor Debug Module (MDM)" é um hardware que provê suporte a debug do core do microprocessador através de ferramentas de software baseadas em conexões JTAG.

Com todos os periféricos descritos acima, temos o diagrama em blocos da *bridge* prototipada conforme a figura 10.1.3.6. Note que o diagrama mostra tanto os periféricos externos à FPGA quanto aqueles presentes internamente na forma de núcleos IP instanciados.

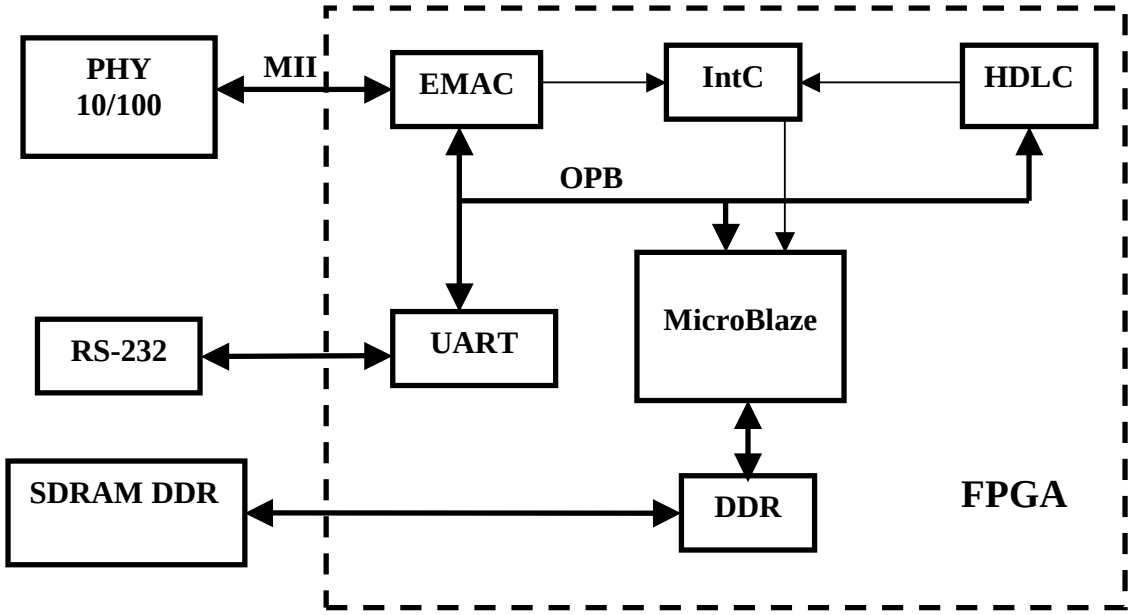


Figura 10.1.3.6 - Diagrama em blocos do hardware da bridge

Todos os núcleos IP utilizados acima são disponibilizados pela Xilinx em seu kit de desenvolvimento EDK. Alguns são desenvolvidos pela própria Xilinx enquanto outros são em parceria com terceiros e então incorporados ao software.

A ocupação de lógica interna ao FPGA para a implementação do hardware da *bridge* é mostrada na tabela 10.1.3.1. A tabela refere-se à implementação feita na placa de desenvolvimento com a FPGA Virtex4.

Tabela 10.1.3.1 - Ocupação de lógica de FPGA (Virtex4)

Item	Quantidade	% do total disponível
Slices	4491	29%
Flip-flops	4468	14%
LUT's	6062	19%
Pinos de E/S	70	15%
BRAM	12	6%

Na implementação do projeto em um FPGA da família Spartan3, a ocupação de lógica ficou conforme a tabela 10.1.3.2 indica.

Tabela 10.1.3.2 - Ocupação de lógica de FPGA (Spartan3)

Item	Quantidade	% do total disponível
Slices	4439	33%
Flip-flops	4648	17%
LUT's	6209	23%
Pinos de E/S	71	14%
BRAM	12	37%

10.1.4. Software da bridge

O software da *bridge* é totalmente escrito em linguagem C ANSI, sem o conceito de orientação à objetos. De forma a obter o máximo de desempenho possível no tratamento dos pacotes e no atendimento às interrupções de hardware, nenhum sistema operacional foi utilizado.

O acesso aos núcleos IP (descritos anteriormente) é feito através do uso do drivers automaticamente gerados pela ferramenta EDK da Xilinx.

Tabela hash

Em vista da simplicidade e do desempenho na implementação em software, a função de *hash* utilizada na tabela é a abordada no capítulo 6.3.3 (*hash* através de lógica XOR). Como foi abordado no capítulo 2.2, o endereço MAC é composto por um conjunto de 48 bits (ou 6 bytes), sendo que 24 destes bits (ou 3 bytes) representam o OUI do fabricante do dispositivo controlador de acesso ao meio. Desta forma, é provável que em redes locais a ocorrência de muitos endereços MAC cujos 3 bytes mais significativos sejam os mesmos, principalmente em redes onde os computadores são comprados em lote e do mesmo fabricante.

Tabela 10.1.4.1 – Endereço MAC e o OUI

Endereço MAC					
Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
OUI			Gama de endereços para a empresa		

Pelo exposto acima, os três bytes mais significativos do endereço MAC tendem a se repetir ao longo do universo de endereços existentes em uma rede local. Assim, a utilização deste campo na função de *hash* não gera resultados diferentes e pode ser descartada, facilitando a execução da função. Desta forma, somente os três primeiros podem ser utilizados no cálculo do *hash*. O impacto do uso de todos ou apenas 3 dos bytes do endereço MAC sobre o desempenho da *bridge* em encaminhar pacotes é mostrado no capítulo 11.

O método de tratamento de colisões adotado é o abordado no capítulo 7.6 (*Coalesced hash*). Segundo Knuth [48], este método não apresenta um desempenho melhor do que o de encadeamento tradicional, abordado no capítulo 7.5, que utiliza listas encadeadas externas à tabela *hash*. Por outro lado, no método de encadeamento tradicional, é ideal que se utilize alocação dinâmica de memória. À medida em que as listas encadeadas crescem devido às colisões, mais memória deve ser alocada. Se não ocorrem colisões, nenhuma memória necessita ser alocada. No método *coalesced hashing*, não há a necessidade de utilização de alocação dinâmica de memória, uma vez que as listas encadeadas, resultado das colisões, residem dentro do próprio espaço de memória da tabela. Isso elimina a necessidade de gerenciamento de memória do sistema e garante que a memória alocada para a tabela *hash* seja apenas aquela definida em tempo de compilação.

A tabela *hash* propriamente dita utilizada nos testes de validação da *bridge* consiste em um bloco de memória de 256 posições. Desta forma, o resultado obtido pela aplicação da função de *hash* sobre os endereços MAC é um inteiro de 8 bits. Assim, a implementação da função de *hash* em software se resume a uma operação “ou exclusivo”, byte a byte, do endereço MAC, como mostra o pseudo código C a seguir:

```
hash = mac[0] ^ mac[1] ^ mac[2] ^ mac[3] ^ mac[4] ^ mac[5];
```

Armazenamento dos pacotes

Inicialmente o software da *bridge* foi concebido sem o emprego do conceito de filas para o armazenamento (*buffering*) dos pacotes recebidos pela interface ethernet e enviados à interface HDLC. Em outras palavras, todos os pacotes recebidos pela interface ethernet (passíveis de envio à interface HDLC), durante o período de tempo em que o controlador HDLC estivesse ocupado com o envio de outro pacote através do link, não eram armazenados na memória e eram descartados. Isso obrigava que os pacotes recebidos pela interface ethernet seguissem uma cadência mínima, que depende do tamanho do pacote sendo transmitido pelo controlador HDLC à *bridge* remota e da taxa de transmissão deste circuito, que no caso da *bridge* implementa, é de 2048k bits/s. De uma forma geral, os protocolos de rede das camadas superiores, tais como o TCP, ajustam a cadência de envio dos pacotes de forma a não afogar a camada de enlace. Porém, pelo menos duas situações podem fazer com que esta cadência mínima de recebimento dos pacotes pela interface ethernet seja extrapolada:

- recebimento de pacotes de duas estações independentes;
- recebimento de pacotes IP fragmentados;

Assim, uma fila de pacotes foi implementada para o armazenamento dos mesmos. O impacto da utilização da fila no desempenho da *bridge* é exposto no capítulo 11.

10.2 CONCLUSÕES

Sem ainda levar em conta o desempenho, a implementação da *bridge* através de o núcleo IP de um *softprocessor* em uma FPGA mostrou-se perfeitamente viável, utilizando-se menos da metade da capacidade de lógica de um dispositivo de 1500k portas. Todos os núcleos IP utilizados são muito bem documentados e podem ser facilmente substituídos, no futuro, por núcleos mais enxutos e otimizados à aplicação final.

A facilidade de desenvolver o software de controle da *bridge* em linguagem C também contribuiu bastante para a bom andamento do trabalho.

11. VALIDAÇÃO DA BRIDGE ETHERNET

Neste capítulo, iremos abordar os testes funcionais e de desempenho efetuados com a *bridge* proposta, bem como expor os resultados obtidos com os mesmos. Em todos os testes realizados, não houve diferença perceptível no desempenho da *bridge* quando a consulta à tabela de endereços era desabilitada. Ou seja, o tempo gasto pelo software para a execução de outras tarefas, tais como a cópia dos pacotes de uma interface à outra (Ethernet para HDLC e vice-versa) somado ao tempo de latência de transmissão dos pacotes através do link digital que conecta as duas bridges, é muito maior do que o tempo necessário ao software realizar o acesso à tabela *hash*. Dessa forma, o uso de apenas 3 ou de todos os campos do endereço MAC no cálculo do *hash* do endereço, pelo exposto anteriormente, também não proporcionou aumento perceptível no desempenho da *bridge*.

11.1. TESTES FUNCIONAIS

De forma a validar a funcionalidade mínima da *bridge*, ou seja, a comutação de pacotes para a *bridge* remota, foi testada a conectividade entre duas estações interligadas através de duas *bridges*, conforme mostra a figura 11.1.1. Pacotes do protocolo ICMP tipo *ping* (*echo request* e *echo reply)* foram utilizados.

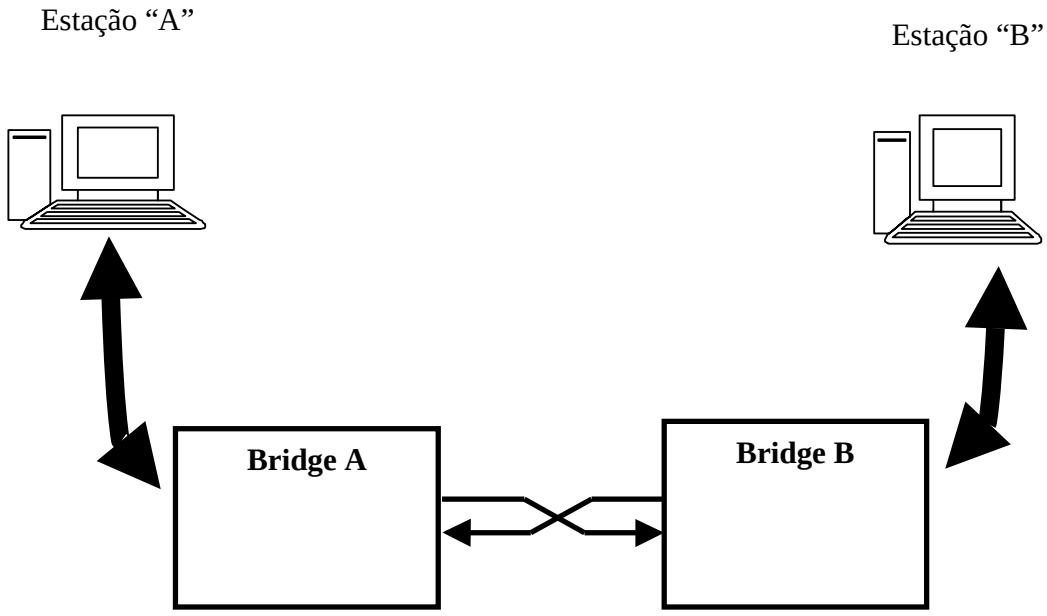


Figura 11.1.1 – Conectividade entre estações

O resultado do teste com *ping* foi satisfatório e a ferramenta de software acusou um tempo de ida e volta da solicitação de aproximadamente 2ms, para o tamanho padrão de pacote utilizado pelo software, que é de 32 bytes no campo de dados UDP.

A *bridge* também foi utilizada para conectar um estação à um roteador de acesso à internet. O teste também foi bem sucedido sendo que a estação navegou pela internet normalmente através da *bridge*.

11.2. THROUGHPUT

O teste de throughput consiste em determinar a máxima taxa de dados que a *bridge* pode encaminhar à *bridge* remota. Para a realização deste teste, foi utilizada a ferramenta de software Iperf [38], conforme descrito no capítulo 5. Em todos os testes, a ferramenta foi configurada para gerar pacotes UDP com tamanhos do campo de dados variando de 64 a 1450 bytes.

Inicialmente o mecanismo de comutação foi implementado sem a utilização de memórias para o armazenamento dos pacotes, conforme abordamos no capítulo 10. A tabela 11.2.1 mostra a perda de pacotes ocorrida na *bridge* nesta situação. A primeira coluna indica o tamanho do campo de dados dos pacotes, a segunda coluna indica a carga oferecida à *bridge* e a terceira indica a perda de pacotes neste teste.

Tabela 11.2.1 – Perda de pacotes sem memória de armazenamento

Payload UDP (bytes)	Carga (bps)	Perda (%)
1450	256k	0,00
	1200k	0,61
	1500k	1,00
1024	200k	0,00
	1024k	0,77
	1300k	0,99
512	150k	0,00
	512k	0,56
	1000k	0,93
128	28k	0,00
	256k	0,57
	490k	1,00
64	20k	0,00
	128k	0,52
	215k	0,93

Após a implementação no software do armazenamento de pacotes, os testes foram repetidos de forma a determinar a vazão máxima para a perda de 0% de pacotes. Os resultados estão indicados na tabela 11.2.2.

Tabela 11.2.2 – Vazão máxima de pacotes com 0% de perda

Payload UDP (bytes)	Taxa máxima (bps)
1450	1800k
1024	1690k
512	1620k
256	1420k
128	1100k
64	810k

Através da comparação dos resultados indicados nas tabelas 11.2.1 e 11.2.2, podemos observar a nítida melhora no desempenho da *bridge* quando utilizamos um mecanismo de armazenamento de pacotes. Sem o uso deste mecanismo, a *bridge* fica muito suscetível a

eventuais rajadas de pacotes geradas na interface ethernet que excedam a taxa em que os pacotes são transmitidos pela interface HDLC que, no caso desta implementação, é de 2048k bits/s, conforme mostrado no capítulo 10.

Nos testes anteriores, a freqüência de operação do barramento do microprocessador MicroBlaze foi de 100MHz. De forma a avaliar o impacto da diminuição desta freqüência no desempenho da *bridge*, uma nova implementação do núcleo da *bridge* foi gerada onde o barramento do MicroBlaze opera a 75MHz. O desempenho da *bridge* nesta situação é indicado na tabela 11.2.3.

Tabela 11.2.3 – Vazão máxima de pacotes (barramento a 75MHz)

Payload UDP (bytes)	Taxa máxima (bps)
1450	1690k
1024	1570k
512	1490k
256	1280k
128	950k
64	670k

Nos testes anteriores, a processador MicroBlaze sintetizado no FPGA não possuía memória cache. No teste cujo resultado é mostrado na tabela 11.2.4, o MicroBlaze foi sintetizado com uma memória cache de instrução de 2048 bytes. Como podemos observar, houve aumento significativo no desempenho da *bridge*.

Tabela 11.2.4 – Vazão máxima com cache de 2k bytes (barramento a 75MHz)

Payload UDP (bytes)	Taxa máxima (bps)
1450	1900k
1024	1890k
512	1780k
256	1280k
128	950k
64	1000k

Throughput na carga máxima oferecida

Conforme abordado no capítulo 5, uma das métricas que podem ser verificadas no desempenho de bridges ethernet é a vazão que a mesma pode ter quando lhe é oferecida a carga máxima possível em sua interface ethernet. A tabela 11.2.5 lista os resultados obtidos no teste de vazão (*throughput*) na carga máxima oferecida à *bridge* para vários tamanhos de pacotes UDP.

Tabela 11.2.5 – Vazão na carga máxima oferecida (100Mbps)

Payload UDP (bytes)	Taxa (bps)
1450	1,49M
1024	1,57M
512	1,49M
256	1,08M
128	827k
64	350k

11.3. LATÊNCIA

Os testes de latência foram executados com o uso do equipamento de teste SmartBits 200, uma versão reduzida da plataforma de testes SmartBits 2000, originalmente desenvolvido pela empresa NetCom Systems e mais tarde adquirida pela gigante Spirent Communications [4].

Conforme abordado no capítulo, existem duas definições diferentes desta métrica para equipamentos de comutação de pacotes. São elas:

> ***Para equipamentos do tipo "armazenagem e encaminhamento"(S&F)***
> O intervalo de tempo iniciado quando o último bit do frame de entrada alcança a porta de entrada e encerrado quando o primeiro bit do frame de saída é visto na porta de saída.

Para equipamentos de encaminhamento de bit (CT)

O intervalo de tempo iniciado quando o final do primeiro bit do frame de entrada alcança a porta de entrada e encerrado quando o início do primeiro bit do frame de saída é visto na porta de saída.

A *bridge* implementada se insere no grupo de equipamentos do tipo “armazenagem e encaminhamento” (*store and forward*), pois os pacotes recebidos são armazenados na memória e só então encaminhados à *bridge* remota. Entretanto, o equipamento de testes realiza a medição levando em conta as duas definições de latência. Os resultados estão indicados na tabela 11.3.1. A carga oferecida durante o teste de latência foi de aproximadamente 80% da taxa máxima que a *bridge* suporta, levantada nos testes anteriores, de forma a não haver perda de pacotes durante o teste.

Tabela 11.3.1 – Latência dos pacotes da bridge

Payload UDP (bytes)	Latência (us) - CT	Latência (us) - S&F
64	635,5	602,3
128	1044,8	942,4
256	1822,8	1618,0
512	3388,4	2978,8
1024	6514,3	5695,1
1450	9117,2	7957,2

11.4 CONCLUSÕES

As limitações de desempenho da bridge Ethernet cujas funções de comutação de pacotes são implementadas em software são evidentes. Para pacotes pequenos, 64 bytes, o desempenho ficou um pouco acima de 1Mbps. Este tipo de tráfego com pacotes, é normalmente observado somente em aplicações do tipo VoIP (Voice over IP), onde o desempenho alcançado pelo sistema é suficiente para algumas conexões simultâneas.

Ficou claro durante os testes que a otimização das funções de software é decisiva para o bom desempenho da *bridge*. Como foi o caso da utilização da memória de armazenamento dos pacotes.

Outra observação muito importante foi a que o mecanismo de comutação de pacotes, implementado com o auxílio de uma tabela *hash*, não teve papel decisivo no desempenho final do sistema, uma vez que a utilização ou não deste mecanismo não ocasionou mudança na performance. Isto nos leva a concluir que, dos processos de software envolvidos, a cópia dos pacotes do controlador MAC para a memória principal (SDRAM) e posteriormente da memória principal para o controlador HDLC é o maior limitante do desempenho. No caminho inverso, no sentido do controlador HDLC ao controlador MAC, também há esta limitação.

12. CONCLUSÕES FINAIS

Nos dias de hoje vemos as redes Ethernet sendo largamente utilizadas. Uma das chaves para a boa aceitação deste tipo de tecnologia, tanto pelos fabricantes de equipamentos quanto pelos usuários, é a simplicidade do protocolo e o baixo custo de produção dos equipamentos.

O mecanismo de comutação de pacotes desempenhado pelos *switches* Ethernet, embora seja muito simples, pode ser incrementado com a adição de novos protocolos que facilitam a operação e a manutenção de redes, tais como o protocolo *Spanning Tree,* o que torna a tecnologia Ethernet uma tecnologia atual.

De forma geral, as bridges Ethernet implementam suas funções em hardware. Entretanto, em aplicações onde a performance do encaminhamento de pacotes não representa um grande problema, como em interfaces WAN de baixa velocidade, o uso de *bridges* em software se torna viável. O próprio kernel do Linux provê uma maneira fácil de configurar um sistema, que possua duas ou mais interfaces de rede, como uma *bridge* Ethernet. Também é possível, através do uso de bibliotecas de software livre, implementar sua própria bridge em software sob uma plataforma PC, tendo assim, liberdade para tornar o dispositivo mais otimizado.

Para a otimização do processo de comutação de pacotes, podemos utilizar o conceito de tabelas *hash* ou o conceito de memórias CAM. O uso de tabelas *hash* oferecem a vantagem do custo, pois podem ser implementadas utilizando-se a própria memória principal do sistema. No caso das memórias CAM, o alto custo e a baixa disponibilidade dos circuito integrados no mercado, bem como o imenso uso de lógica para implementação das mesmas em FPGA, torna seu uso inviável.

A Xilinx, assim como outros fabricantes de circuitos integrados de lógica programável, provê uma gama de dispositivos das mais variadas capacidades (densidades), que podem ser utilizados nas mais diversas aplicações do mercado e de pesquisa.

Nos dias de hoje, o apelo principal dos fabricantes de dispositivos de lógica programável é, dado o aumento na densidade dos componentes, a possibilidade da implementação de processadores através de núcleos de propriedade intelectual. Tais processadores podem variar desde um simples núcleo de 8 bits, até um complexo processador de 32 bits com arquitetura RISC.

A implementação da *bridge* através de o núcleo IP de um *softprocessor* em uma FPGA mostrou-se perfeitamente viável, utilizando-se menos da metade da capacidade de lógica de um

dispositivo de 1500k portas. Todos os núcleos IP utilizados são muito bem documentados e podem ser facilmente substituídos, no futuro, por núcleos mais enxutos e otimizados à aplicação final.

As limitações de desempenho da bridge Ethernet cujas funções de comutação de pacotes são implementadas em software são evidentes. Para pacotes pequenos, 64 bytes, o desempenho ficou um pouco acima de 1Mbps. Este tipo de tráfego com pacotes, é normalmente observado somente em aplicações do tipo VoIP (Voice over IP), onde o desempenho alcançado pelo sistema é suficiente para algumas conexões simultâneas.

Ficou claro durante os testes que a otimização das funções de software é decisiva para o bom desempenho da *bridge*. Como foi o caso da utilização da memória de armazenamento dos pacotes.

Outra observação muito importante foi a que o mecanismo de comutação de pacotes, implementado com o auxílio de uma tabela *hash*, não teve papel decisivo no desempenho final do sistema, uma vez que a utilização ou não deste mecanismo não ocasionou mudança na performance. Isto nos leva a concluir que, dos processos de software envolvidos, a cópia dos pacotes do controlador MAC para a memória principal (SDRAM) e posteriormente da memória principal para o controlador HDLC é o maior limitante do desempenho.

Como principais contribuições deste trabalho, podemos citar:

abertura de oportunidades para novos projetos de pesquisa que levem à otimização do sistema de comutação de pacotes.
primeiro passo na construção de um sistema enxuto e reconfigurável para a comutação de pacotes Ethernet através de funções de software.
estudo e aplicação de técnicas de software para maximizar a performance da comutação de pacotes Ethernet.
avaliação do desempenho do processador MicroBlaze na tarefa de comutação de pacotes Ethernet.

13. TRABALHOS FUTUROS

Este trabalho teve como foco o desenvolvimento de uma *bridge* ethernet, através do uso de um *softprocessor*, com as funcionalidades mínimas possíveis, focando no desempenho da comutação de pacotes através do uso de tabelas *hash*. Este pode ser o ponto de partida e servir como base para muitos trabalhos desenvolvidos no futuro.

Aqui, a função de *hash* testada e utilizada foi implementadas totalmente em software. De forma a incrementar ainda mais a performance da *bridge*, as funções de *hash* poderão ser mapeadas para o hardware. Ainda focando no aumento de performance, a análise detalhada de cada bloco de hardware e software pode ser feita de forma a identificar pontos de melhoria.

O modelo de *bridge* proposto e implementado neste trabalho, contém uma porta conectada à rede local (porta LAN) e outra conectada à interface WAN. Assim, os pacotes recebidos pela interface WAN não precisam ser processados pela *bridge*, em relação ao endereço de destino, e podem ser diretamente encaminhados à porta WAN, após verificada sua integridade. Isso se deve ao fato dos pacotes já terem sido processados pela *bridge* remota quando recebidos por sua interface LAN. Uma adição ao projeto da *bridge* pose ser a inclusão de mais portas WAN, de forma a podermos ter uma *bridge* local conectada a várias bridges remotas, como mostra a figura 13.1. A inclusão de novas portas WAN irá requerer uma mudança no software da *bridge*, que agora terá que processar os endereços de origem e destino dos pacotes recebidos por suas interfaces WAN, de forma a decidir se os mesmos devem ser comutados à outra porta WAN ou mesmo para a porta LAN.

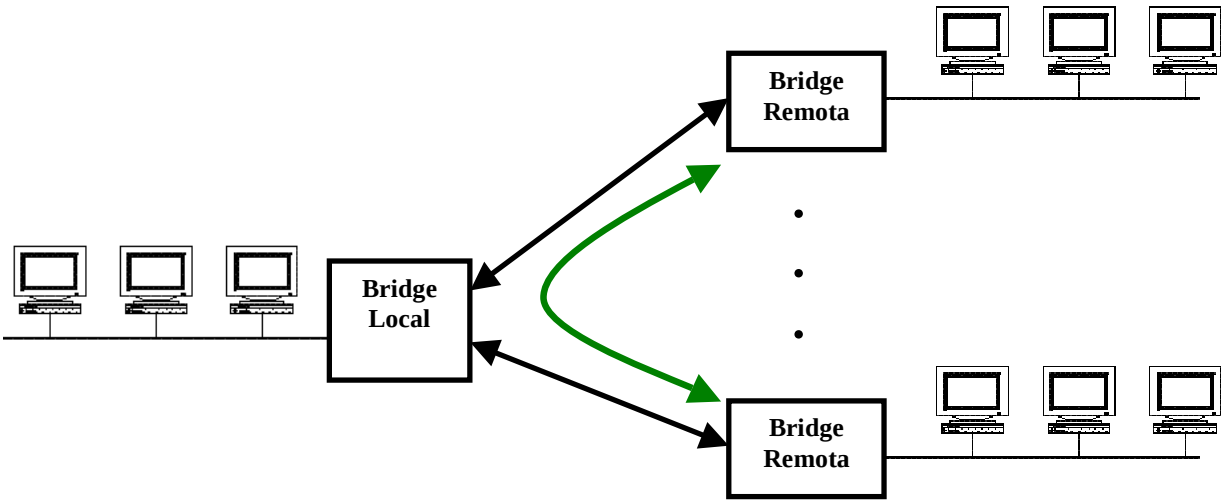


Figura 13.1 - Bridge Local conectado-se a duas ou mais bridges remotas

Durante os testes do sistema, foi observado que um dos limitantes do desempenho da bridge é a cópia dos pacotes dos controladores das interfaces LAN e WAN para a memória principal e vice-versa. Uma análise mais detalhada do real desempenho destes processos, de como afetam a performance do sistema, e de melhorias a serem feitas para maximizar a taxa de encaminhamento de pacotes é um trabalho muito importante a ser realizado. É muito provável que seja necessário a utilização de núcleos IP mais específicos para a otimização deste processos. No que se refere ao controlador ethernet, um trabalho pode ser encontrado em [44].

Por fim, um estudo sobre o impacto da utilização de um sistema operacional sobre o desempenho do software pode ser feito. Mesmo os sistemas operacionais voltados à aplicações embarcadas, como o μCOS-II [47], podem ter um impacto negativo na performance do software. Entretanto, os benefícios gerados pela facilidade de divisão das tarefas executadas pela aplicação podem ser decisivos para a inclusão de novas características no equipamento como, por exemplo, o suporte à RSTP [23]. A inclusão de um sistema operacional também pode trazer benefícios no que se refere ao gerenciamento dos *buffers* que guardam os pacotes recebidos à espera da transmissão.

14. REFERÊNCIAS BIBLIOGRÁFICAS

[1] MicroBlaze Softprocessor Core. Capturado em http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf, 2007.

[2] PicoBlaze 8-bit Embedded Microcontroller. Capturado em http://www.xilinx.com/bvdocs/userguides/ug129.pdf, 2007.

[3] Tong-Bi Pei and Charles Zukowski. Putting routing tables in silicon. *IEEE Network Magazine*, 6(1):42 50, 1992.

[4] Site oficial da *Spirent Communications*. Disponível em www.spirent.com

[5] G. Jaeschke. Reciprocal hashing: a method for generating minimal perfect hashing functions. *Communications of the ACM*, v.24 n.12, p.829-833, 1981

[6] Crynwr software. *PC/TCP Packet Driver Specification*. Disponível em http://www.crynwr.com/packet_driver.html, 2007.

[7] Jeffrey Scott Vitter. Implementations for coalesced hashing. *Communications of the ACM*, Volume 25 , Issue 12, Pages 911 - 926, 1982.

[8] J. Romkey and S. Fisher. All about Packet Drivers. *Byte Magazine*, p.297–306, 1991.

[9] Richard J. Cichelli. Minimal perfect hash functions made simple. *Communications of the ACM*, v.23 n.1, p.17-19, 1980.

[10] Constantine Halatsis and George Philokyprouj. Pseudochaining in hash tables. *Communications of the ACM*, Volume 21, Issue 7, *Pages:554 - 557*, 1978.

[11] J. Ian Munro and Pedro Celis. Techniques for collision resolution in hash tables with open addressing. *Proceedings of 1986 ACM Fall joint computer conference*, *Pages: 601 - 610*, 1986.

[12] Mao-Yin Wang, Chih-Pin Su, Chih-Tsun Huang, Cheng-Wen Wu. An efficient implementation of hash function processor for IPSEC. *Proceedings of IEEE Asia-Pacific Conference on ASIC*. Pages: 93-96, 2002.

[13] A. Khan, N. Al-Darwish, M. Guizani, M. Benten, H. Youssef. Design and implementation of a software bridge with packet filtering and statistics collection functions. *International Journal of Network Management*, Volume 7, Issue 5, *Pages: 251 - 263*, 1997.

[14] ITU-T G.704, Synchronous frame structures used at 1544, 6312, 2048, 8448 and 44 736 kbit/s hierarchical levels. Disponível em http://www.itu-org, 1998.

[15] P. Gupta, S. Lin, and N. McKeown. Routing Lookups in Hardware at Memory Access Speeds, *Proc. IEEE INFOCOM '98*, Apr. 1998, pp. 1240–47.

[16] James R. Bell. The quadratic quotient method: a hash code eliminating secondary clustering. *Communications of the ACM*, Volume 13 , Issue 2, *Pages: 107 - 109*, 2006.

[17] Ruiz-Sanchez, M.A, Biersack, E.W. Dabbous, W. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, Volume 15, Issue 2, 2001.

[18] F. Duarte, A. Vieira, L. Basso, F. Caminha, A. Kieling, Sistema para abordagem prática e didática do mecanismo de comutação de quadros ethernet. *COBENGE - Congresso Brasileiro de Ensino de Engenharia*, 2004.

[19] V. Papaefstathiou, I. Papaefstathiou. A hardware-engine for layer-2 classification in low-storage, ultra-high bandwidth environments. *Proceedings of the conference on Design, automation and test in Europe: Designers' forum, Pages: 112 - 117*, 2006.

[20] Jim Robinson. Kernel korner: Linux as an ethernet bridge. *Linux Journal*, Volume 2005, Issue 135, *Page:11, 2005*.

[21] J. Lawrence Carter, Mark N. Wegman. Universal classes of hash functions. *Annual ACM Symposium on Theory of Computing, Proceedings of the ninth annual ACM symposium on Theory of computing*. *Pages:106 – 112, 1977*.

[22] IEEE 802.3. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Capturado em http://standards.ieee.org/getieee802/download/802.1D-2004.pdf

[23] Cisco Systems White Papers. Understanding Rapid Spanning Tree Protocol (802.1w). Disponível em http://www.cisco.com/warp/public/473/146.html, *2006*.

[24] Cisco Systems Incorporation. Disponível em www.cisco.com

[25] Metro Ethernet Forum. Disponível em http://www.metroethernetforum.org

[26] Broadcom Corporation. Disponível em http://www.broadcom.com

[27] Ebtables Utility. Disponível em http://ebtables.sourceforge.net

[28] Libnet. Disponível em http://www.packetfactory.net/libnet

[29] *LibpCap* - TCPDUMP Public Directory. Disponível em http://www.tcpdump.org

[30] Xilinx MicroBlaze FAQ. Capturado em, http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/mb_faq.pdf

[31] Altera Corporation. Disponível em www.altera.com

[32] Xilinx Incorporation. Disponível em www.xilinx.com

[33] Lattice Semiconductor Corporation. Disponível em www.latticesemi.com

[34] Xilinx Platform Studio and the EDK. Disponível em http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm

[35] ITU-T G.991.2, Single-pair high-speed digital subscriber line (SHDSL) transceivers. Disponível em www.itu.org

[36] IEEE Registration Authority. Disponível em http://standards.ieee.org/regauth

[37] Reinhold P. Weicker. Dhrystone: A Synthetic Systems Programming Benchmark. *Communications of the ACM 27*, p.1013-1030, Oct. 1984.

[38] IPERF. Disponível em http://dast.nlanr.net/Projects/Iperf/

[39] NetPerf. Disponível em http://www.netperf.org/netperf/

[40] The Internet Engineering Task Force – IETF. Disponível em http://www.ietf.org/

[41] IETF - Benchmarking Terminology for Network Interconnection Devices. Capturado em http://www.ietf.org/rfc/rfc1242.txt

[42] IETF - Benchmarking Terminology for LAN Switching Devices. Capturado em http://www.ietf.org/rfc/rfc2285.txt

[43] IETF - Benchmarking Methodology for LAN Switching Devices. Capturado em http://www.ietf.org/rfc/rfc2889.txt

[44] CALAZANS, Ney Laert Vilar ; MORAES, Fernando Gehm ; TOROK, Delfim ; ANDREOLI, Andrey . Projeto para Prototipação de um IP Soft Core MAC Ethernet. *Revista de Informatica Teórica e Aplicada*, Porto Alegre, v. 8, n. 1, p. 23-41, 2001.

[45] Robert M. Metcalfe , David R. Boggs, Ethernet: distributed packet switching for local computer networks, *Communications of the ACM*. v.19 n.7, p.395-404, July 1976

[46] Rina Panigrahy , Samar Sharma, Sorting and Searching using Ternary CAMs. *IEEE Micro*, v.23 n.1, p.44-53, January 2003.

[47] µC/OS-II, The Real-Time Kernel. Disponível em http://www.micrium.com/products/rtos/kernel/rtos.html

[48] D. Knuth, The Art of Computer Programming, vol. 3. Reading, Mass.: Addison-Wesley, 1973.

[49] IETF, The MD5 Message-Digest Algorithm. Capturado em http://www.ietf.org/rfc/rfc1321.txt

[50] IETF, US Secure Hash Algorithm 1 (SHA1). Capturado em http://www.ietf.org/rfc/rfc3174.txt

[51] Chie Dou, Ming-Der Shieh. A CAM-Based VLSI Architecture for Shared Buffer ATM Switch with Fuzzy Controlled Buffer Management. *Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*. 1996.

[52] M. V. Ramakrishna, E. Fu, E. Bahcekapili, Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, Volume 46, Issue 12, Pages: 1378-1381, 1997.

[53] FreeScale MPC190 : Security Processor. Disponível em http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC190

[54] GNU gperf – Perfect hash function generator. Disponível em http://www.gnu.org/software/gperf/gperf.html

[55] Gary Haggard, Kevin Karplus, Finding minimal perfect hash functions. *Technical Symposium on Computer Science Education*. p. 191 – 193, 1986.

[56] IEEE 802.3Q, Virtual Bridged Local Area Networks. Capturado em http://standards.ieee.org/getieee802/download/802.1Q-2003.pdf

[57] MAURER, W. D. An improved hash code for scatter storage. *Communications of the ACM 11*, p. 35-38, 1968.

ANEXOS

A. MODEM POWER2048 SHDSL

Dentre os diversos produtos da PARKS, empresa gaúcha do setor de telecomunicações, destaca-se a linha de modens digitais SHDSL, para transmissão de dados a taxas de até 4608k bits/s sob 2 pares de fios. Este tipo de equipamento utiliza a codificação TC-PAM de 16 níveis para transmitir dados a uma velocidade de até 2304k bits/s sobre um par trançado de fios de cobre a distâncias que podem superar os 3 km. Usualmente tais equipamentos são utilizados para a interligação de centrais PABX utilizando-se de sua interface G.703.

A estrutura de dados transmitida pelas centrais PABX é de no máximo 2048k bits, representando 30 canais de voz a 64k bits/s mais 2 canais de sinalização, também de 64k bits/s cada. Dada a possibilidade dos equipamentos SHDSL transmitirem dados a uma taxa máxima de 4608k bits/s (quando operando sobre 2 pares de fios), abre-se a oportunidade de mesclar-se serviços sobre o mesmo link de dados. Por exemplo: uma operadora que forneça a uma empresa um link SHDSL a 2304k bits/s sob um par de fios pode utilizá-lo para a interligação das centrais PABX e ainda assim dispor de mais 256k bits/s (2304k – 2048k) para utilização em outro serviço, a transmissão de dados de uma segunda interface do equipamento. Os equipamentos produzidos pela PARKS possuem esta facilidade, são multi-interface, onde as mesmas podem operar simultaneamente. A figura A.1 ilustra este tipo de aplicação.

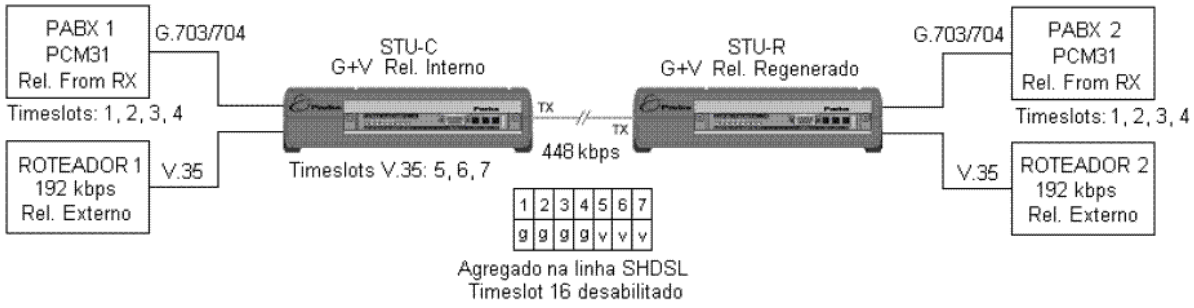


FIGURA A.1 – Aplicação multi-interface (V.35 e G.703)

Neste exemplo, um caso clássico, a operadora de telecomunicações provê um link de 448k bits/s à empresa contratante. Através deste link, o modem transmite os dados da central PABX, conectado a sua interface G.703, bem como os dados do roteador conectado a sua interface V.35 ou V.36 a 192k bits/s.

Ainda neste exemplo, os dados da rede local, Ethernet, são desencapsulados pelo roteador e encapsulados novamente sob um novo protocolo, que normalmente é o Frame-Relay, podendo ser também PPP ou ainda HDLC.

Existem casos em que, para o acesso de dados da rede local, é desejável entregar Ethernet diretamente ao cliente, seja para conectá-lo a uma rede Metro Ethernet ou mesmo para interligar duas LAN's de suas filiais próximas. Com o recente surgimento das redes Metro Ethernet, isto se tornará cada vez mais comum. Nestas situações, o modem necessita ter uma interface Ethernet e também prover um mecanismo de encapsulamento dos quadros Ethernet (uma rede de pacotes) sob os quadros SHDSL (rede de circuitos síncrona). A solução adotada para isso é o encapsulamento dos pacotes ethernet sob um segundo protocolo, que tem a função de adaptar a natureza assíncrona da rede de pacotes ao tipo de transmissão síncrona da rede SHDSL. O protocolo utilizado é um simples HDLC. Os quadros Ethernet provenientes da LAN são encapsulados em um novo quadro HDLC e este por sua vez é colocado dentro do quadro SHDSL. Quando não existem pacotes Ethernet a serem transmitidos, quadros HDLC vazios são gerados para preencher o espaço de tempo reservado a esta interface no feixe SHDSL. A figura A.2 mostra uma topologia multi-interface onde o tráfego de voz é feito pela interface G.703 do equipamento e o tráfego de dados é feito pela interface Ethernet.

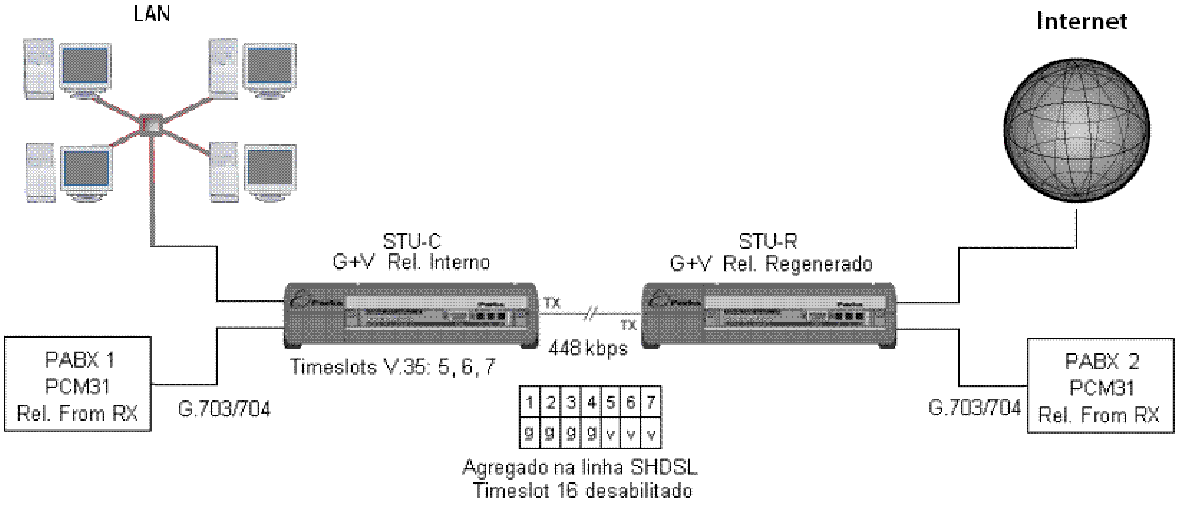


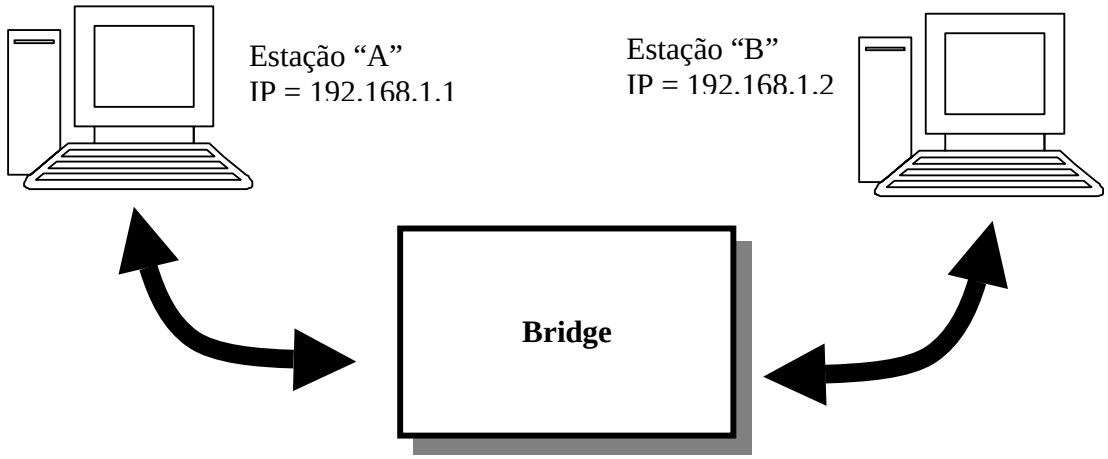
Figura A.2 – Aplicação multi-interface (Ethernet e G.703)

B. IPERF

Iperf é uma ferramenta de software largamente utilizada para a medição de desempenho de equipamentos de rede. Através desta ferramenta, é possível aplicar uma determinada carga de dados a um equipamento de comutação de pacotes e medir sua capacidade em encaminhá-los de uma porta à outra, medindo a perda de pacotes.

A ferramenta pode gerar pacotes do tipo TCP ou UDP, para testes mais severos, sendo que a taxa oferecida e o tamanho dos pacotes é configurável através de parâmetros inseridos na linha de comando.

O teste utilizando o IPERF consiste em gerar pacotes em uma estação (Estação "A") e medir a quantidade de pacotes que chegam na estação adjacente (Estação "B"). Através desta medição, é possível determinar se o equipamento sob teste (bridge ou roteador) é capaz de tratar todos os pacotes na taxa oferecida. A Figura B.1 ilustra um ambiente de teste utilizando a ferramenta IPERF.



A linha de comando abaixo exemplifica um caso prático onde pacotes UDP são gerados da estação A (cliente) para a estação B (servidor).

Estação A (cliente)

iperf –c 192.168.1.2 –u –t 300 –b 1000k

As opções da linha de comando especificam os seguintes parâmetros:

-c: especifica modo cliente e endereço IP do servidor
-u: especifica uso de pacotes UDP
-t: especifica o tempo do teste em segundos

-b: especifica a taxa de dados a ser transmitida

Estação B (servidor)

iperf –s –u

As opções da linha de comando especificam os seguintes parâmetros:

-s: especifica modo servidor
-u: especifica uso de pacotes UDP