UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE DOS SANTOS MARRANGHELLO

Implementação de um Circuito Dedicado para Roteamento de Datagramas IPv4

Trabalho de Diplomação.

Prof. Dr. João César Netto
Orientador

Porto Alegre, dezembro de 2009.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CEITEC	Centro de Excelência em Tecnologia Eletrônica avançada
ASIC	Circuito Integrado de Aplicação Específica
IP	Internet Protocol
TTL	Time To Live
TL	Total Length
CIDR	Classless Inter-Domain Routing
BDD	Diagrama de Decisão Binária
CAM	Memória Endereçável por Conteúdo
TCAM	Memória Endereçável por Conteúdo Ternária
DRAM	Memória de Acesso Randômico Dinâmica
SFQ	Stochastic Fair Queuing
DRR	Deficit Round-Robin
MAC	Media Access Control
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
HDL	Linguagem de Descrição de Hardware
RTL	Register Transfer Level
VHDL	VHSIC Hardware Description Language
ATPG	Automatic Test Pattern Generator
DRC	Design Rules Constraints
FPGA	Field Programmable Gate Array
CPLD	Complex Programmable Logic Device
PLA	Programable Logic Device
POS	Product of Sums
SOP	Sum of Products

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O desempenho do encaminhamento de pacotes é essencial para determinar a velocidade que pode ser alcançado pelas redes de computadores. Não adianta o meio físico permitir grande fluxo de dados se os roteadores não conseguirem fazer o processamento das informações com a eficiência necessária.

Este trabalho propõe uma implementação de um circuito integrado de aplicação específica (ASIC) para encaminhamento que consiga alcançar a velocidade necessária para redes atuais. Todos os passos do fluxo de projeto de ASIC são analisados. O circuito será responsável por consultar a tabela de roteamento para decidir qual caminho um pacote deve seguir.

.

.

Palavras-Chave: Roteador, Circuito Dedicado, ASIC, Pacote IP, Descrição de Hardware.

.

Implementation of a Dedicated Circuit for IPv4 Packets Forwarding

ABSTRACT

The packets forwarding performance is essential to determine the maximum that can be achieved by computers networks. It is useless the physical layer provide a big data flow if the routers are unable to process the data with the necessary efficience.

This work propouses an Aplications Specific Integrated Circuit (ASIC) implementation to perform the routing task that can achieve the needed speeds for nowadays networks. All steps in the ASIC design flow are analised. The circuit is responsable for deciding wich way a packet will follow.

Key words: Router, Dedicated Circuit, ASIC, IP Packet Forwarding, Hardware Description.

1 INTRODUÇÃO

Como em todos os campos da informática, existe uma necessidade do desempenho das redes de computadores aumentar cada vez mais, permitindo assim maiores velocidades de acesso. Para que isto seja possível temos que possuir meios físicos que permitam maior fluxo de dados, como fibra óptica, além de equipamentos de hardware e software que consigam processar as informações. Além disso, a Internet expande-se cada vez mais, fazendo com que o número de computadores e, consequentemente, de roteadores conectados à rede cresça rapidamente.

Na Internet, roteadores são os elementos responsáveis por decidir por qual caminho um pacote deve seguir para chegar ao destino. Analisando o endereço da rede que deve ser alcançada, o roteador decide qual a próxima etapa no percurso do datagrama.

Como o encaminhamento de pacotes deve ser feito com a maior velocidade possível, é justificável a criação de um circuito dedicado para esta tarefa. Outras tarefas que existem em roteadores não são fundamentais em termos de velocidade, não sendo assim soluções em hardware necessárias. No entanto, é importante lembrar que existem programas que atingem o desempenho necessário.

O mercado de microeletrônica é, em termos de faturamento, um dos maiores do mundo. Praticamente todos os produtos comercializados possuem algum componente semicondutor. Infelizmente, o Brasil ainda não tem um lugar de destaque no desenvolvimento e produção de tais equipamentos. Existem atualmente diversas ações que visam mudar este cenário, uma delas é a criação do Centro de Excelência em Tecnologia Eletrônica avançada (CEITEC) que é uma empresa que fabrica Circuitos Integrados de Aplicação Específica (ASIC), a única empresa da América Latina com esta capacidade.

1.1 Objetivos do Trabalho

A primeira parte do trabalho tem como objetivo a criação de uma descrição de um circuito digital em uma linguagem de descrição de hardware. Ele deve fornecer uma solução que permita o encaminhamento de datagramas *Internet Protocol* (IP) versão 4 (IPv4). Esta descrição não é a de um roteador completo já que diversas funcionalidades dos roteadores são melhores feitas em software e, portanto, não são aqui implementadas. O trabalho foca apenas no encaminhamento de pacotes, pois é a função que precisa ser realizada o mais rapidamente possível para permitir conexões com altas

velocidades. Também se deseja que seja possível expandir o sistema para lidar com o IP versão 6 (IPv6).

A segunda parte do trabalho é passar pelo fluxo de projeto de ASIC para obter um layout do circuito que possa ser fabricado. Observar que a fabricação propriamente dita não faz parte da proposta. As ferramentas usadas nesta fase são as mesmas usadas na indústria, existem ferramentas acadêmicas que realizam as mesmas funções, porém elas não são consideradas. Os maiores desafios desta fase são determinar os requisitos para o sistema e determinar a melhor forma de se usar os programas disponíveis para obter-se um circuito com bom desempenho.

1.2 Datagrama IP

Um datagrama IP é o elemento sobre o qual o roteador atua diretamente. Por isso, faz-se necessário um entendimento dos campos contidos no datagrama. Primeiro, explica-se o datagrama da versão 4 do IP, depois, as principais diferenças para a versão seis são mostradas. A figura 1.1 mostra o formato do datagrama IPv4.

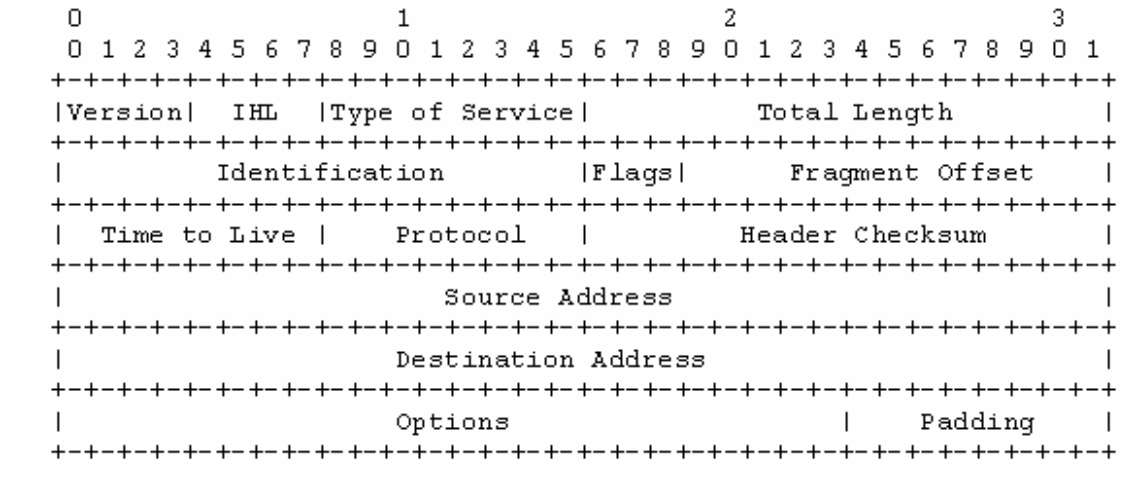

Figura 1.1 : Formato do Cabeçalho IP (RFC 791)

Para este trabalho, os campos mais importantes são o *Time to Live* (TTL), o *Total Length* (TL), *o Header Checksum* e os endereços de origem e destino (*Source Address* e *Destination Address*). Depois do cabeçalho, vem a parte dos dados que também tem tamanho variável.

O campo de versão possui quatro bits e indica qual o protocolo IP está sendo usado. Neste trabalho, este campo deverá ter o valor quatro. O *Internet Header Length* usa um byte e serve para indicar o tamanho do cabeçalho IP em palavras de 32 bits, é necessário um valor mínimo de cinco neste campo.

O campo *Type of Service* serve para determinar parâmetros para o tipo de serviço desejado. Para ver uma descrição dos possíveis valores, consultar a RFC 791. O campo *Total Length* mostra o tamanho total do datagrama em bytes. Como este valor tem 16

bits, um pacote IP pode ter no máximo 65536 bytes. *Identification* é um número de 32 bits que serve para ajudar o receptor na tarefa de remontagem dos pacotes.

O campo de *flags* possui 13 bits de alerta. Os principais são: o bit zero que é reservado, o bit 1 que indica que o datagrama não pode ser fragmentado (este bit deve ser setado quando, por exemplo, sabe-se que o receptor não tem a capacidade de desfragmentação) e o bit 2 que indica que ainda há mais fragmentos do pacote. Quando ocorre a fragmentação de um datagrama, os fragmentos não chegam, necessariamente, ordenadamente, então, usa-se a informação dos 13 bits do campo *Fragment Offset* para indicar qual a parte sendo recebida no momento.

Os oito bits do campo TTL determinam a quantidade de passos que ainda podem ser realizados pelo pacote na rede antes dele ser descartado, evitando assim que um datagrama entre em um laço infinito e prejudique o desempenho do sistema. Cada roteador por onde o pacote passa decrementa este campo e quando o valor chega em zero o datagrama é descartado

O campo *Protocol* serve para informar qual o protocolo usado no campo de dados. Os diversos valores possíveis podem conferidos na RFC 790. Os 16 bits do *Header Checksum* servem para detecção de erros no cabeçalho. Este valor é o complemento de um da soma dos complementos de um de cada 16 bits do cabeçalho.

Os endereços de origem e destino são determinados, respectivamente, pelos campos de *Source Address* e *Destination Address*. No IPv4, cada um tem 32 bits.

O campo Options pode ser usado para determinar diversas maneiras de como tratar o datagrama, os possíveis valores podem ser consultados na RFC 791. Este campo é opcional para os datagramas, ou seja, não é necessário enviar o pacote com esta parte. No entanto, os receptores devem ter a capacidade de tratar todos os valores possíveis. O último campo é o Padding que é uma sequência de bits em zero para garantir que o cabeçalho tem um tamanho múltiplo de 32 bits.

A principal diferença entre um datagrama IPv4 e um IPv6 é que o segundo usa 128 bits para endereços. Além disso, há mais opções que podem ser especificadas. Apesar deste trabalho ser desenvolvido visando o IPv4, é importante estar atento para o aumento do número de endereços já que muitas soluções que ignoram esta expansão podem ter o problema de pouca escabilidade, devido à falta de memória ou pela complexidade do algorítmo em função do tamanho do endereço, e, portanto, não sendo eficientes para o futuro quando os datagramas IPv6 forem mais utilizados.

1.3 Encaminhamento de Datagramas IP

Um endereço IP pode ser decomposto em duas partes, o endereço de rede e o de máquina. O encaminhamento é feito analisando apenas a parte relativa ao endereço de rede. Desta forma, todos os pacotes com destino a uma mesma rede são direcionados pela mesma interface de saída mesmo que a máquina final seja diferente.

Inicialmente, utilizavam-se cinco classes pré-determinadas de endereços IP, denominadas A, B, C, D e E. Um pacote de classe A usa oito bits para determinar a rede e sempre começa com um bit em zero. A classe B é identificada pelos dois primeiros

bits terem o valor 10 e usa, no total, 16 bits para identificar a rede. Finalmente, um pacote do tipo C usa 24 bits para o endereço de rede com os três primeiros com o valor 110. A classe D indica que deve ser realizado um multicast e tem o prefixo 1110. Existe também a classe E que é reservada e usa o prefixo 11110. O encaminhamento é feito identificando a classe do pacote e pesquisando na tabela de roteamento a interface de saída para aquele destino. Um problema que aparece é o desperdício de endereços já que, por exemplo, muito dificilmente uma rede A terá os 16.777.214 computadores permitidos. Existirão, então, diversos endereços que não especificam nenhuma máquina. Além disso, muito comumente, uma rede irá possuir mais do que os 256 computadores de uma classe C e então terá que receber um endereço classe B ocasionando assim uma superlotação dessa categoria.

Class	Leading Bits	Size of *Network Number* Bit field	Size of *Rest* Bit field	Number of Networks	Hosts per Network
Class A	0	8	24	128	16,777,214
Class B	10	16	16	16,384	65,534
Class C	110	24	8	2,097,152	254
Class D (multicast)	1110	not defined	not defined	not defined	not defined
Class E (reserved)	1111	not defined	not defined	not defined	not defined

Figura 1.2: Classes Básicas do IP

Para diminuir este problema criou-se o *Classless Inter-Domain Routing* (CIDR) que permite que uma quantidade qualquer de bits seja usada para determinar a rede destino. Desta maneira diminuí-se o número de endereços IP desperdiçados.

Em compensação, o CIDR aumenta significantemente a complexidade da tarefa de roteamento, pois não é possível saber quantos bits são usados para designar a rede de destino apenas olhando o endereço IP. É possível que, para alguns endereços, exista mais de uma entrada na tabela de roteamento que pode corresponder ao destino, sendo necessário mandar para a rede especificada com o maior prefixo. Por exemplo, se a tabela de roteamento tiver apenas as entradas 0110, 011011 e 011011 e um pacote com os oito primeiros bits do endereço de destino iguais a 01101100 precisar ser roteado, ambas as entradas 0110 e 011011 especificam redes de destinos válidas. No entanto, deve-se utilizar a entrada com o maior número de bits iguais, ou seja, o datagrama deve ser mandado para a interface de saída equivalente à entrada 011011 da tabela de roteamento.

Uma notação muito utilizada é a abc.def.ghi.jkl/x. Neste caso, a,b,c,d,e,f,g,h,i,j,k e l são números decimais representado o endereço IP e x é o número de uns na máscara. Também é possível usar números hexadecimais. A contagem sempre é feita a partir do bit mais significativo. Também é possível usar números decimais na notação. Durante o texto, o termo tamanho da máscara será usado como sinônimo de número consecutivos de uns da máscara a partir do bit mais significativo. Se as máscaras forem interpretadas como um número inteiro positivo, então é possível dizer que aquela com maior número de uns é maior que a outra.

1.4 Métodos de Projeto de Circuitos Dedicados

1.4.1 Programmable Logic Array (PLA)

Este dispositivos implementam as funções lógicas através de uma soma de produtos (SOP) ou, eventualmente, produto de somas (POS). No primeiro caso, tem-se dois planos, um que implementa todas os produtos possíveis entre as entradas e o segundo com todas as somas. Deve-se escolher então quais termos fazem parte da função desejada.

Estes equipamentos não são reprogramáveis e apresentam a vantagem de ser fácil estimar o atraso pela regularidade da estrutura. Na prática, seu uso limita-se a projetos simples, com algumas centenas de portas, e que tenham uma freqüência de operação perto de, no máximo, 100 MHz.

1.4.2 Complex Programmable Logic Device (CPLD)

Este dispositivo possuí células que implementam as funções lógicas, não sendo necessário expressá-las na forma de uma SOP. Desta maneira, consegue-se uma flexibilidade maior do que usando um PLA. Além disso, células projetadas para aritmética melhoram ainda mais o desempenho dos CPLD's.

1.4.3 Field Programmable Gate Array (FPGA)

Um FPGA consiste em um conjunto de células, chamadas de bloco lógico configurável, que podem ter a sua função lógica e as suas conexões com outras células programadas. O mais comum é que a configuração seja guardada em uma memória de acesso randômico, pode-se então programar o mesmo dispositivo várias vezes. Existem, porém, produtos, geralmente não atuais, que só podem ser programados uma vez. Também é normal FPGA's que possuem elementos mais complexos integrados como processadores, memórias e blocos otimizados para aritmética. A figura 1.3 mostra um esquema de um FPGA básico.

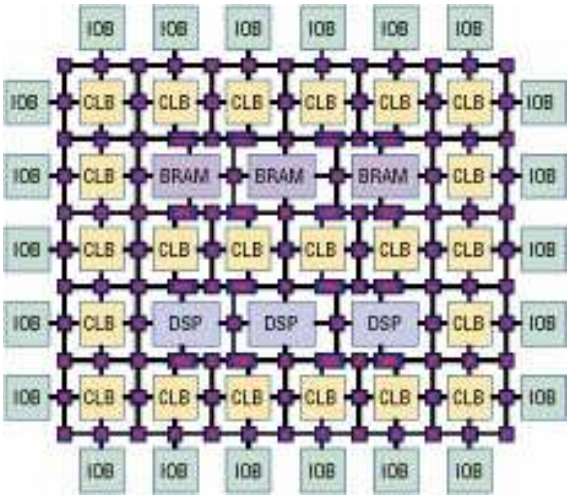

Figura 1.3: Configuração Básica de um FPGA

A programação do FPGA geralmente é feita através de uma descrição do circuito em uma HDL (Linguagem de Descrição de Hardware). Este texto é transformado em um vetor de bits que configura o FPGA.

A capacidade de reprogramação do FPGA permite uma grande flexibilidade o que torna este tipo de circuito interessante para vários tipos de problemas, processamento de imagens, visão computacional, processamento digital de sinais são alguns exemplos de aplicações. A reprogramação também é importante por permitir que melhorias no projeto sejam feitas rapidamente mesmo quando o circuito já esta em funcionamento no sistema final. Como os FPGA's modernos possuem, a não ser em alguns casos específicos, recursos suficientes para desempenhar qualquer lógica necessária, as maiores restrições na utilização deles são o consumo e a velocidade.

O custo de projeto depende simplesmente do valor do FPGA utilizado e do tempo necessário para a criação e validação da descrição HDL. O investimento total necessário é, portanto, proporcional à quantidade de unidades que serão vendidas. A regularidade da estrutura, as células têm posições fixas e apenas muda-se as conexões entre elas, faz com que seja fácil fazer análises dos desempenhos de temporização e consumo.

1.4.4 Gate Array

Os métodos até aqui expostos consistem na programação de um dispositivo para a implementação do circuito, que não é fabricado diretamente. O desenvolvimento usando Gate Array difere dos demais por necessitar a fabricação do projeto.

A posição dos transistores é pré-definida. O projeto consiste em fazer uma máscara que define a ligação entre os elementos. Esta abordagem permite uma liberdade maior no projeto do que as anteriores.

1.4.5 Cell-based

Este é o método utilizado neste trabalho. Este tipo de projeto usa um conjunto de células, criadas para uma tecnologia específica, como blocos elementares para a criação do circuito. Esta biblioteca pode possuir desde células mais simples como inversores, *nands* e *nors* até multiplicadores e somadores que são estruturas mais complexas. Também é comum que existam diversos elementos com a mesma função lógica, mas com características de área, consumo e atraso diferentes. Isto é importante já que uma célula maior tende a ser mais rápida e consumir mais energia. Desta maneira, os circuitos projetados conseguem melhor desempenho do que as estratégias anteriores.

As células que foram escolhidas para implementar o circuito são posicionadas no chip e as conexões entre elas são realizadas. No final, tem-se um desenho do circuito que é mandado para a fábrica que desenvolveu a tecnologia escolhida para a fabricação.

Este tipo de solução tem um custo inicial muito grande. No entanto, o valor de cada unidade fabricada é muito baixo. O investimento total necessário pode ser visto então como sendo quase constante.

Esta abordagem possui um fluxo de projeto mais complexo, que será detalhado no capítulo 4, do que usando um FPGA. No início do projeto, desconhece-se a posição de cada célula no *chip*. Além disso, as conexões entre elas e até mesmo os circuitos de

distribuição de relógio e de alimentação precisam ser criados. Como resultado, é necessário verificar que estas ligações foram geradas de uma maneira que possibilite o funcionamento correto do circuito. No resto do trabalho, o termo ASIC será usado para referenciar um projeto feito de acordo com esta metodologia.

1.4.6 Full-custom

O *layout* do circuito é feito diretamente pelo projetista. Ou seja, ele desenha o circuito da maneira desejada, desde que respeite as regras de fabricação da tecnologia alvo. Apesar de possibilitar que o projeto seja feito de maneira totalmente personalizada para cada circuito, a complexidade desta abordagem faz com que ela seja usada somente em partes de projetos maiores onde não se consegue o desempenho necessário usando o método baseado em células.

1.5 Circuitos Síncronos e Assíncronos

Um circuito digital também pode ser classificado como sendo síncrono ou assíncrono. O que difere os dois grupos é a presença ou ausência de um relógio global para sincronização do circuito.

Em um circuito síncrono, os elementos de memórias têm os seus valores atualizados, idealmente, no mesmo instante. O sinal de relógio é responsável por determinar o momento da atualização. Como conseqüência, em um circuito com vários caminhos, é necessário esperar até que o mais devagar tenha a saída estável para atualizar os elementos de memória..Mesmo nos circuitos que usam mais de um sinal de relógio, as considerações apresentadas são válidas. A figura 1.4 é um exemplo de um circuito síncrono simples.

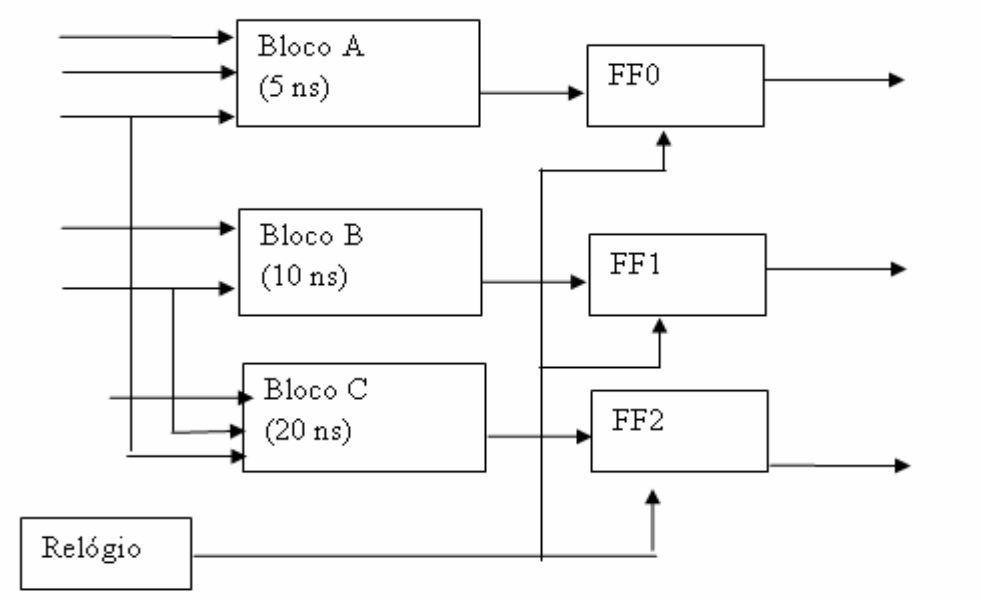

Figura 1.4: Exemplo de circuito síncrono

Os blocos A, B e C são circuitos puramente combinacionais o número entre parênteses especifica o tempo necessário para o valor da saída ser calculado. Os

elementos FF0, FF1 e FF2 são *flip-flops* controlados pelo sinal de relógio. Se todas as entradas forem válidas no instante zero, então nos instantes 5, 10 e 20, os blocos A,B e C terminam, respectivamente, a computação dos valores de saída. Como o relógio só poderá mandar os *flip-flops* atualizarem os seus valores quando todas as entradas dos *flip-flops* estiverem prontas, o sinal só poderá ser enviado aos 20ns. Como conseqüência, os sinais A e B devem esperar 15 e 10ns respectivamente. O atraso de um circuito síncrono é o maior tempo para um bloco combinacional fazer o cálculo do valor da saída.

Um circuito assíncrono não utiliza um sinal de relógio para fazer a sincronização dos dados. Essencialmente, os blocos se comunicam para saber quando podem mandar as informações adiante. Portanto, há uma independência entre caminhos paralelos no circuito. A figura 1.5 mostra um exemplo de um circuito assíncrono.

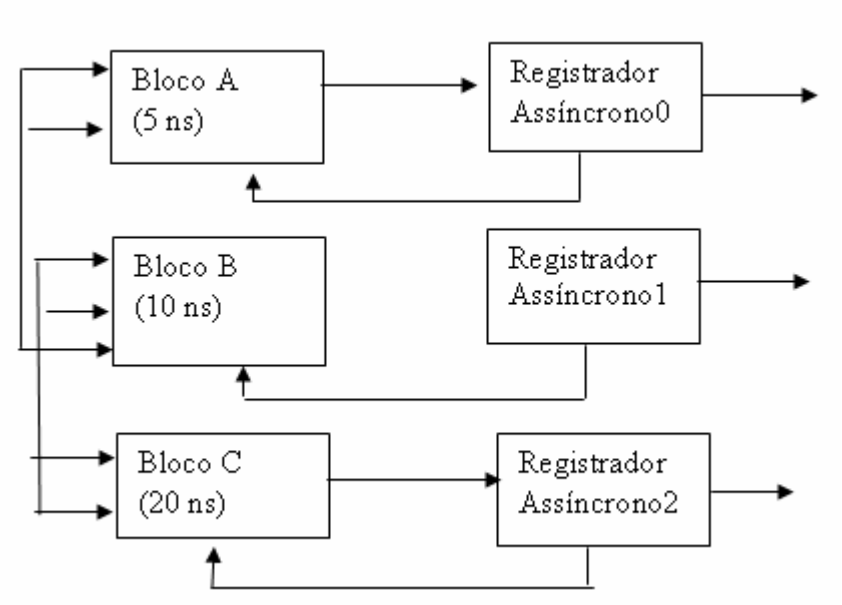

Figura 1.5: Exemplo de circuito assíncrono

Os blocos A,B e C são circuitos combinacionais assíncronos e os *flip-flops* foram substituídos por registradores assíncronos, ou seja, que não utilizam um sinal de relógio. Quando o bloco A tiver definido o valor da saída, ele manda um aviso para o registrador e espera até que chegue uma resposta confirmando o armazenamento do dado. O bloco A é então liberado para processar um novo valor de entrada. Os comportamentos dos blocos B e C são semelhantes. É importante observar que, neste caso, cada par formado por um bloco combinacional e registrador é independente dos demais, o registrador assíncrono0 não espera que o bloco C termine a sua computação para armazenar o valor calculado pelo bloco A. Em um circuito assíncrono, o atraso é o atraso médio dos blocos.

A ausência de um sinal de relógio traz diversas vantagens para o projeto. O consumo de energia diminui já que o sinal de relógio de um circuito síncrono consome energia para ser transmitida pelos fios. Além disso, mesmo que um *flip-flop* não esteja

sendo utilizado naquele ciclo de relógio, ele atualiza a sua saída o que caracteriza um consumo de energia mesmo que o novo valor seja igual ao anterior. Além disso, o relógio é um sinal que pode interferir e sofrer interferência dos outros sinais do circuito. Este ruído gera correntes elétricas que também são consumo. Também há a possibilidade de um circuito assíncrono ser mais rápido que um síncrono já que os blocos não precisam esperar pelos demais.

No entanto, a realização de um projeto síncrono tende a ser mais simples que a de um projeto assíncrono. O relógio garante uma sincronização dos dados que permite que o projetista saiba mais facilmente o que está acontecendo no circuito. É interessante que a criação da rede de distribuição de relógio é considerada uma tarefa bastante complicada, especialmente para tecnologias mais modernas.

Neste trabalho, o projeto feito será um circuito síncrono já que as ferramentas que serão utilizadas foram projetadas para este tipo de projeto. No ambiente comercial, há um predomínio de projetos síncronos, fazendo com que os projetos assíncronos sejam quase exclusivos do meio acadêmico.

1.6 Tecnologia e Formatos de Arquivo

A tecnologia de um projeto é definida pelo comprimento mínimo do canal dos transistores. Transistores com canais menores podem operar em freqüências mais altas, com tensões e correntes menores e ocupam menos área. No entanto, são mais sensíveis a ruídos e problemas decorrentes de variabilidade de processo. Neste projeto, é utilizada a tecnologia 0.35um da XFab que não é uma tecnologia de ponta, mas é a utilizada pelo CEITEC.

As ferramentas utilizadas são genéricas, ou seja, não desenvolvidas para uma tecnologia específica. Portanto, é preciso passar informações para elas para que seja possível fazer o circuito de acordo com as especificações da tecnologia escolhida. Os principais formatos utilizados são brevemente descritos a seguir. O objetivo desta seção é, simplesmente, mostrar qual a função dos arquivos no fluxo sem entrar em detalhes quanto à sintaxe deles.

1.6.1 Arquivo de Tecnologia

Este arquivo descreve as regras geométricas que o *layout* do circuito deve seguir para que ele possa ser fabricado. Há dimensões mínimas de largura (wmin) e comprimento (lmin) para cada elemento do *layout.* Além disso, há também distâncias mínimas (smin) entre os retângulos e também regras de *overlap* (omin) que especificam quanto dois retângulos podem se sobrepor e *enclosure* (emin) que especifica quanto um retângulo deve estar dentro de outro. A figura a seguir mostra exemplos destas regras.

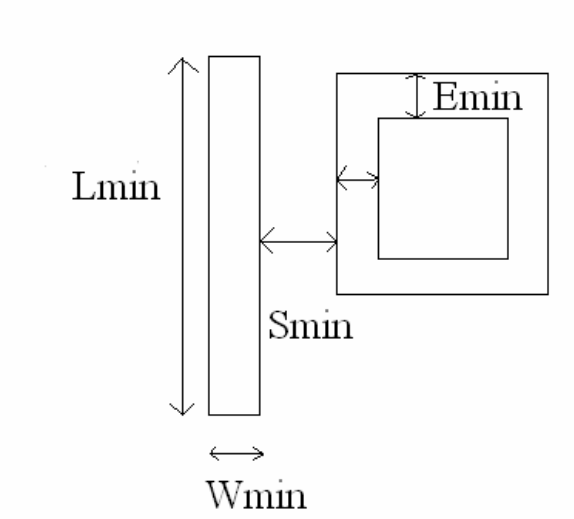

Figura 1.6: Exemplo de Regras de desenho

As regras são usadas para desenhar as células que serão usadas para a síntese lógica do circuito. Este processo pode ser feito tanto manualmente como de maneira automática. Além disso, há uma fase do fluxo específica para a correção de violações das regras de desenho que podem aparecer durante o fluxo.

1.6.2 Library Exchange Format

São dois tipos de *Library Exchange Format* (LEF), um deles contém as informações sobre os pinos das células. Ele informa as coordenadas das entradas, saídas e alimentação de cada célula. Desta maneira, a ferramenta responsável pelo roteamento do circuito sabe exatamente onde estão os pontos que devem ser conectados. Ele também contém informações sobre o roteamento interno da célula para que a ferramenta saiba que espaços já estão ocupados. O outro, chamado de arquivo LEF de tecnologia, tem as informações sobre os níveis de metais e vias que são usados no roteamento das conexões das células no *chip.*.

1.6.3 Liberty File

Um arquivo *liberty* é usado em praticamente todas as etapas do fluxo, ele contém as informações elétricas sobre as células. Para cada célula, são fornecidas informações sobre a função lógica, a área, o consumo e atraso. Também são definidos valores para a máxima capacitância de saída e o maior tempo de transições aceitáveis para que as células funcionem corretamente.

Outras informações contidas neste arquivo são diferentes condições de operação, que permitem estimar o comportamento do circuito para diferentes valores de temperatura, alimentação e variabilidade do processo. Há também diferentes modelos de resistência e capacitâncias de fios para estimar os atrasos das conexões. Durante o texto, este arquivo é chamado, usualmente, de biblioteca de células.

1.6.4 Synopsys Design Constraint (SDC)

Este formato de arquivo traz restrições definidas pelo projetista que devem ser respeitadas. As principais informações contidas nele não são específicas para uma determinada tecnologia. Por exemplo, informações referentes aos sinais de relógio como o período dele e qual a porcentagem deste tempo que ele fica em cada nível são determinadas aqui. Além disso, informações sobre atrasos de entradas e saídas em relação ao sinal de relógio e definições de valores máximos para a área e consumo do circuito fazem parte deste arquivo.

Este arquivo também pode conter redefinições dos valores de capacitância máxima e máximo tempo de transição das células. Neste caso, como estes valores já estão definidos na biblioteca de células, como parte de regras da tecnologia, os valores definidos pelo arquivo SDC só são utilizados se forem mais restritivos do que os já existentes.

1.6.5 Geometric Data Stream (GDS II)

Este formato de arquivo descreve o *layout* final do circuito. A empresa responsável pela fabricação o utiliza para criar as máscaras necessárias. Pode-se considerar que o objetivo do fluxo é gerar este arquivo da melhor maneira possível.

2 TRABALHOS RELACIONADOS

O grande gargalo dos roteadores é a tabela de roteamento. O desempenho é limitado pelo tamanho dela, que indica quantos endereços de destino podem ser armazenados e pela velocidade de acesso, que determina a freqüência máxima de transmissão de pacotes. Deseja-se uma maneira de diminuir o tamanho da memória necessária e da quantidade de acessos feitos para cada pacote.

O problema de decidir qual a rede de destino de um datagrama IP é, usualmente, tratado como um problema de busca em uma árvore binária. Cada caminho entre a raiz e uma folha representa uma entrada da tabela de roteamento. Então, a rede de destino do pacote de entrada é determinada pelo maior caminho cujo valor iguala o endereço de destino do pacote IP. Os algoritmos desenvolvidos, em hardware ou software, fazem otimizações nesta estrutura para obterem melhores resultados.

Pao et al(2003) propõem uma implementação em hardware para achar o endereço de destino de um pacote IP. A árvore é dividida em sub-árvores de tamanho constante, cada uma com 255 nodos, possibilitando processamento paralelo. A figura a seguir mostra um exemplo de uma árvore binária para endereços de quatro bits. Os nodos pretos indicam que aquela máscara está presente na tabela de roteamento.

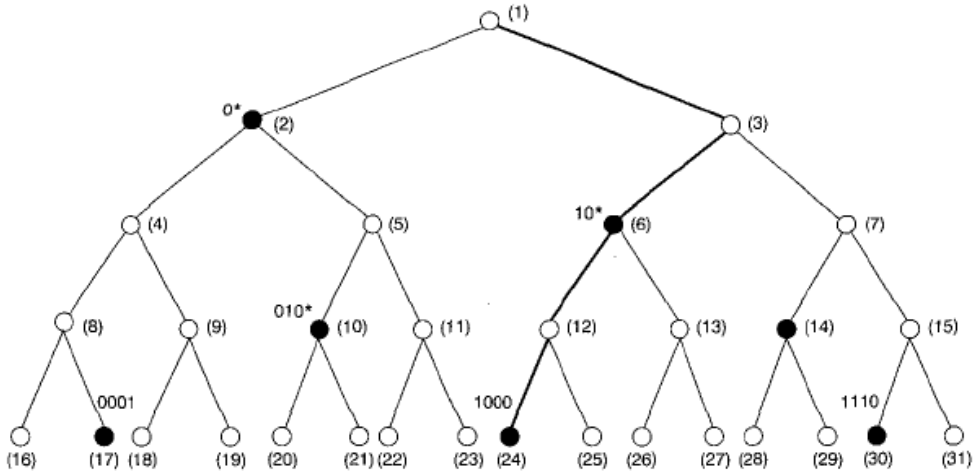

Figura 2.1: Árvore Binária para Roteamento IP

A rota percorrida para o endereço 1000 é destacada. Neste caso, existem dois prefixos que podem ser usados, o correspondente ao nodo seis e o correspondente ao nodo 24, deve-se usar o segundo por ser o mais longo. Juntamente com a árvore binária monta-se um outro vetor que armazena um identificador para determinar a interface de saída. Este esquema também permite a utilização de IPv6 sem perda significativa de desempenho. A única restrição é o total de memória necessária.

Chang e Lim(2004) desenvolveram uma arquitetura implementada em um ASIC que consegue tanto reduzir a quantidade de memória como obter elevada taxa de transmissão. O endereço IP de 32 bits é dividido em seis partes de 14, 4, 4, 4, 4 e 2 bits. Cada partição representa um nível e é associada com uma memória que contem um campo para indicar que o próximo nível pode ser ignorado, um ponteiro para ser usado como indexador no próximo nível e um conjunto de próximos destinos.

Sangireddy e Somani (2003) apresentam uma estrutura baseada em Diagrama de Decisão Binária (BDD) que usa hardware reconfigurável para desempenhar a tarefa de encaminhamento de pacotes. A idéia é semelhante à proposta por Pao et al. (2003), mas diminui a quantidade de nodos necessários usando estratégias de reduções de BDD's. Por exemplo, um filho não precisa existir se tiver a mesma interface de saída que o pai.

Sun e Zhao (2005) propõem um algoritmo para comprimir os dados da tabela de roteamento. Desta maneira, eles conseguem diminuir a quantidade de memória necessária para o armazenamento da tabela. Cada entrada na tabela de roteamento determina um conjunto de valores de destino que são mandados para a mesma interface. Os limites superior e inferior de cada intervalo são armazenados e a decisão de roteamento é feita descobrindo em qual intervalo o endereço se encontra. O limite inferior é calculado fazendo todos os bits de host iguais a zero e o superior é calculado fazendo todos os bits de host iguais a um. Além disso, não é necessário armazenar todo o valor dos limites, pois eles, usualmente, apresentam valores iguais nos bits mais significativos e uma quantidade mínima igual de zeros no final que podem ser descartados.

A estrutura utilizada por Sun e Zhao é semelhante a uma árvore B, mas a ordem da árvore não é determinada, ou seja, não é definido o número máximo de elementos que podem ser guardados em um nodo. Uma restrição deste trabalho é que todas as rotas da tabela devem ser conhecidas antes da tabela ser montada para minimizar a quantidade de memória necessária. Além disso a estrutura de dados utiliza ponteiros para determinar os endereços dos filhos dos nodos, isto pode atrapalhar a implementação do algoritmo com uma estrutura de *pipeline*.

Também pode-se utilizar uma Memória Endereçável por Conteúdo (CAM) ou Ternary CAM (TCAM). Este tipo de memória se diferencia das outras devido ao seu princípio de funcionamento. Neste tipo de memória, coloca-se na entrada uma informação que se deseja recuperar, cada endereço verifica, paralelamente, se está armazenando aquele dado e responde positivamente ou negativamente. Se a resposta for positiva, ela pode conter todo o conteúdo daquela posição da memória A diferença entre uma CAM e uma TCAM é que a segunda suporta o valor lógico *don't care* além do um e zero. Os principais problemas desta estratégia são o alto custo de fabricação, o alto

consumo e a pouca utilização da memória devido a um método ineficiente de armazenamento.

Chu et al. (2005) usam uma função de *hashing* para organizar a tabela de roteamento para o IPv4 em diversas CAM's e DRAM's. Um conjunto destas memórias é pesquisado de acordo com o tamanho do endereço de rede. O tempo de busca é diminuído, pois algumas memórias são utilizadas paralelamente.

Outra decisão importante que deve ser feita é escolher qual dos datagramas que chegam nas interfaces de entrada num momento deve ser processado. Um algoritmo simples escolhe o primeiro datagrama que é recebido. Esta solução tem o defeito de depender da boa vontade dos elementos da rede para funcionar, pois uma fonte de dados mal-intencionada pode enviar muitos pacotes e congestionar a rede, prejudicando assim o desempenho. Além disso, as conexões devem diminuir a transmissão de datagramas quando perceberem um possível congestionamento na rede.

Os roteadores tentam, geralmente, colocar uma noção de justiça neste processo, que depende do algoritmo utilizado. Notar que não existe uma definição formal única para justiça e, então, pode-se usar a idéia intuitiva de não punir uma fonte devido ao comportamento de uma outra e possibilitar que fontes que se comportam semelhantemente possam transmitir a mesma quantidade de dados. Uma idéia é considerar um rodízio entre as interfaces, com cada uma tendo direito à uma certa quantidade de dados para transmitir por vez. No entanto, dependendo da configuração da rede, esta solução não irá conseguir um bom desempenho. A figura 2.2 mostra uma destas situações.

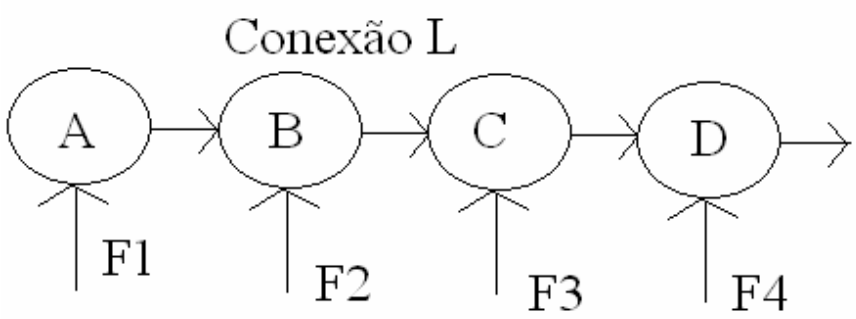

Figura 2.2: Topologia de Rede com Comportamento Injusto

Se todas as fontes (F1 até F4) transmitirem dados e o roteador D fizer a escolha do pacote a ser transmitido apenas considerando a interface de entrada, então, mesmo que nenhuma das fontes se comporte mal, não será possível um tratamento justo, pois F1, F2 e F3 compartilham a mesma interface e, então F4 tem metade da banda da rede, F3 um quarto e F1 e F2 um oitavo.

Nagle (1987) propôs uma solução para este problema. Ao invés de analisar a interface de entrada do pacote, ele classifica os pacotes em fluxos, que são conjuntos de datagramas com os mesmos endereços de fonte e destino. Cada fluxo tem uma fila

própria na interface de saída e as interfaces são tratadas com *round-robin*. Ou seja, cada uma recebe uma quantidade de informação que pode transmitir por rodada. Assim, um elemento mal-comportado, que manda pacotes com uma freqüência muito grande, simplesmente faz com que a sua fila aumente e só consegue transmitir a quantidade determinada. Porém, como o autor considerou que todos os pacotes têm o mesmo tamanho, esta idéia acaba beneficiando as fontes que enviam pacotes maiores.

Demers et al. (1989) propõem um algoritmo que consegue o melhor desempenho possível. A idéia é semelhante à de Nagle, porém ao invés de pacotes, cada fluxo transmite um bit por vez. Como esta solução não é possível de ser implementada, os autores sugerem simular o comportamento do algoritmo ideal. Para isso, calcula-se o tempo quando cada pacote seria transmitido e organizam-se as filas na saída de acordo com o tempo de partida.

McKenney (1991) criou o algoritmom conhecido como *Stochastic Fair Queuing* (SFQ). Ele utiliza uma função de *hash* para separar os diferentes fluxos e fazer o encaminhamento para a fila correspondente. Embora a utilização de um *hash* fosse esperada na implementação das idéias anteriores, McKenney sugere que diversos fluxos diferentes possam ser mapeados para a mesma fila de saída, recebendo o mesmo tratamento, ao contrário das abordagens anteriores que exigem uma fila para cada fluxo. Este esquema tem as vantagens de simplificar a função de *hash* e de diminuir os requisitos de memória. Enquanto isso, ele faz com que fluxos que sejam mapeados para a mesma fila sejam tratados injustamente.

Shreedahar e Varghese (1996) usam a idéia de McKenney com uma alteração, quando um fluxo não transmite todos os bits aos quais tinha direito em alguma rodada do *round-robin*, a sobra é adicionada ao limite que pode ser usado na próxima vez. Desta maneira, existe uma compensação que impede que algum fluxo seja prejudicado, mas ainda existem possíveis perdas devido a colisões na função de *hash*. Este esquema é chamado de *Déficit Round-Robin* (DRR).

Kompella e Varghese (2004) perceberam que, usando o DRR, pode ser necessária uma grande quantidade de elementos de memória apenas para armazenar a quantidade de dados que cada fila tem direito a transmitir. Foi observado que apenas um pacote por fila por rodada pode fazer com que o limite não seja respeitado. A idéia é determinar uma probabilidade de transmissão deste pacote. Ou seja, algumas vezes ele será transmitido e o limite não será respeitado e, às vezes, ele só sairá da fila na rodada seguinte. Desta maneira, consegue-se um desempenho semelhante ao do DRR tradicional, porém com menos uso de memória.

3 DESCRIÇÃO DO PROJETO

Este capítulo serão apresentados a arquitetura do circuito a ser implementado e os passos necessários para a criação do *layout* final partindo da especificação do projeto. A metodologia será a *cell-based*, descrita na seção 1.4.5.

3.1 Arquitetura do roteador

O objetivo de um roteador é determinar o destino de um pacote IP. Portanto, esta decisão deve ser feita da maneira mais rápida possível já que é ela que determina o desempenho do roteador. Por isso, é a funcionalidade que se deseja implementar em hardware. Um possível diagrama de blocos do circuito que realiza esta operação é mostrado na figura 3.1. Este projeto aborda o projeto da máquina de encaminhamento e da tabela de roteamento, porém é feita uma breve explicação de cada módulo para facilitar o entendimento do sistema e uma futura implementação deles.

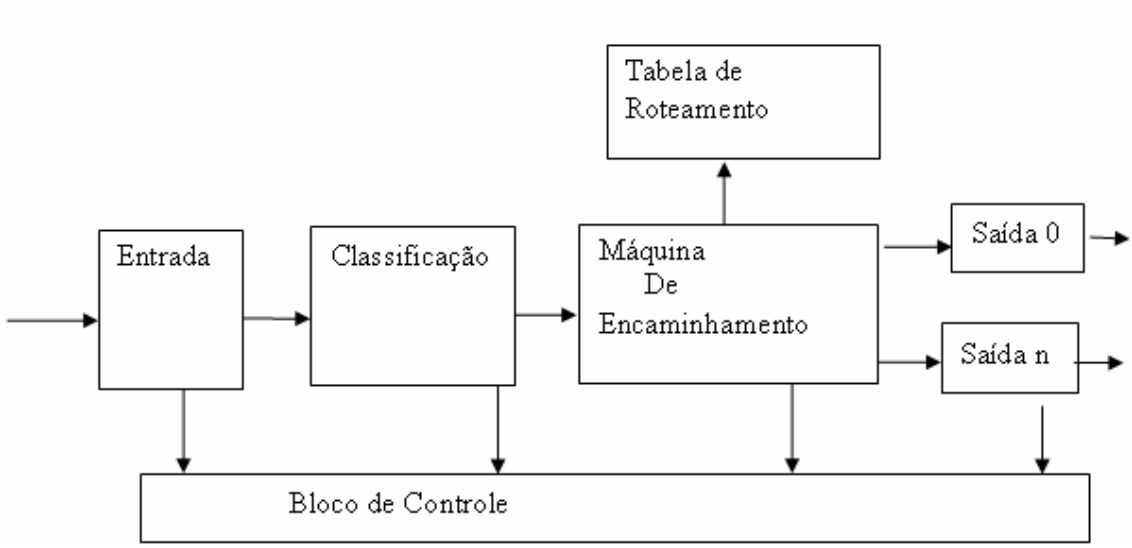

Figura 3.1: Diagrma de Blocos do Roteador

O bloco de controle é o responsável por fazer a comunicação com a parte implementada em software. As principais funções dele são receber sinais de erros dos demais blocos e receber as atualizações que devem ser feitas na tabela de roteamento.

Como o software não será implementado, será assumido que esta comunicação é feita através de registradores que são acessíveis tanto pelo hardware quanto pelo software

O bloco entrada é a junção de todas as interfaces do roteador. Será considerado que os pacotes são armazenados em uma memória compartilhada entre as entradas. Então, o circuito que será projetado poderá trabalhar apenas em cima dos cabeçalhos dos datagramas. Este bloco testa os campos TTL e *checksum* do datagrama e, em caso de erro, manda uma mensagem para o bloco de controle. O campo TTL também é decrementado.

.O bloco de classificação é responsável por analisar o datagrama e estabelecer uma prioridade para a sua transmissão. Por exemplo, pode-se decidir que cada máquina tem um limite de pacotes que podem ser transmitidos com uma prioridade. Também é possível decidir que algumas máquinas sempre terão a prioridade máxima.

A tabela de roteamento é a responsável por armazenar as rotas para as diferentes redes de destino. Infelizmente, não será possível criar a tabela de roteamento interna ao chip. O tempo de acesso à memória é um parâmetro importante para determinar a maior taxa de transmissão possível e uma memória interna apresentaria uma velocidade maior do que uma externa.

A máquina de encaminhamento tem a função de decidir qual a interface de saída de cada datagrama. Este módulo é considerado o gargalo do roteador já que a operação de decisão da rota é mais complicada de ser feita que as demais operações. A decisão é feita pesquisando, em uma tabela de roteamento, o melhor caminho para um pacote. Conforme já foi explicado, há a possibilidade de haver várias rotas possíveis para um datagrama e a escolha deve ser feita analisando todas as opções. Portanto, para cada transmissão a tabela deve ser acessada diversas vezes. O principal objetivo é tentar diminuir o número de acessos necessários à memória. Porém, muitas vezes, mais velocidade significa em custo de implementação maior e, então, deve-se procurar uma boa relação custo-benefício.

O algoritmo para fazer o encaminhamento, baseado em Shun e Zhao, é como segue: deseja-se mandar um pacote para um endereço IP w, então, é buscada na tabela uma palavra de memória. Esta palavra contém o endereço de uma rede destino r = ab.cd.ef.gh/x (notação explicada na seção 1.3) . É realizado uma operação E lógico da máscara de rede definida por x com w, se o resultado for igual a r, então r é uma possível rede de destino para w. Por exemplo, seja r = 25.34.98.00/24 e w = 25.34.98.F4, a máscara m é FF.FF.FF.00. Então, w E m é igual a 25.34.98.00 que é igual a r e, portanto, r é uma possível rede de destino.

No entanto, como há, possivelmente, mais de uma rede que pode ser endereço de destino, considerando o w anterior r’ = 25.34.98.80/25 e r’’ = 25.34.98.00/23 também podem ser o destino para w, deve-se decidir para qual delas enviar o pacote. A decisão é sempre mandar para aquela cuja máscara é maior. Neste exemplo, a rede r’ é a correta.

Como no início não se conhece nenhuma rede de destino possível, a primeira encontrada terá a maior máscara até aquele instante. Quando for encontrada uma outra rede candidata, os tamanhos das máscaras serão comparados e a maior será a escolhida. Desta maneira é garantido que a rede de destino será a correta.

Os blocos denominados saída representam as interfaces. Cada uma possui diversas filas que representam prioridades diferentes e um escalonador que decide de qual fila tirar o pacote para enviar para a saída. O escalonador escolhe sempre um pacote da fila de maior prioridade não vazia. A última etapa é passar pelo *Traffic Shapping*, nesta parte cuida-se para que cada interface não mande mais pacotes por segundo do que uma quantidade definida. Geralmente determinada pelo protocolo usado na interface. Os datagramas esperam até que seja possível fazer a transmissão. Por motivos de simplificação, apenas duas estão representadas no diagrama de blocos, porém mais serão previstas no projeto.

3.2 Organização da Memória

Este projeto utiliza uma árvore binária para organizar os dados na memória. Essencialmente, cada nodo terá o endereço de uma rede de destino e o número da interface de saída para aquela rede. Esta estrutura de dados foi escolhida, pois, apesar de necessitar de uma árvore com uma quantidade maior de nodos do que, por exemplo, uma árvore B, tem algoritmos mais simples para realizar as operações. Além disso, a desvantagem da velocidade durante a busca, quanto maior a altura da árvore maior o tempo necessário para efetuar a operação, pode ser compensada através de um *pipeline*.

Cada posição de memória tem 48 bits (seis bytes). O conteúdo de cada palavra é mostrado na figura 3.2.

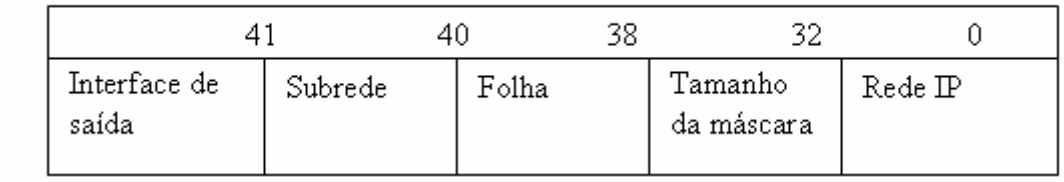

Figura 3.2: Formato de palavra de memória

Os primeiros 32 bits guardam o endereço IP da rede destino. Os próximos seis bits têm o tamanho da máscara desta rede. Optou-se por guardar o número de uns da máscara ao invés da máscara para diminuir o espaço necessário. Os próximos dois bits indicam se o nodo atual é uma folha ou não. Eles indicam se os filhos da esquerda e direita estão presentes. Estes bits são importantes, pois, quando o filho que deveria ser acessado não estiver na tabela, uma transmissão deve ser requisitada. O próximo bit é chamado de subrede e indica a presença ou ausência de subredes da rede IP armazenada neste nodo. Quando este bit estiver setado, não poderá ser feita a transmissão para a rede IP, pois é possível que haja uma outra possibilidade de interface de destino para o pacote sendo transmitido. Os sete bits restantes determinam o valor da interface de saída. Será feita uma explicação melhor sobre a utilização de cada um dos campos da memória quando for explicado o funcionamento do circuito.

Apesar da operação de inserção na tabela não ter sido implementada, é interessante considerar que deve-se ir para o filho esquerda apenas quando o endereço do nodo a ser inserido for estritamente menor que o do presente na árvore. Desta maneira, se o nodo a ser inserido for uma sub-rede do já existente, ele será sempre

colocado na sub-árvore da direita. Isto deve facilitar a determinação do valor correto do bit subnet. A tabela 3.1 mostra um exemplo de uma tabela de roteamento com nove entradas.

Endereço global de memória	Endereço local / nível do *pipeline*	IP/Máscara	Folha	Subnet	Interface de saída
0	0/0	A5.4F.59.00/24	10	1	1
2	2/0	A5.4F.59.80/25	00	0	3
5	5/0	A5.4F.59.40/26	11	0	6
6	6/0	B7.9D.60.40/26	01	1	7
13	6/1	B6.9D.60.C0/26	00	1	14
27	13/1	B5.00.00.00/8	11	0	28
28	14/1	B6.9D.60.CE/31	10	0	29
57	29/1	B6.9D.60.C1/32	10	0	58
116	60/2	C0.EF.00.00/16	11	0	117

Tabela 3.1: Exemplo de tabela de roteamento

A primeira coluna mostra o endereço de memória se for considerado que uma memória armazena toda a tabela. A segunda coluna mostra os endereços para uma tabela dividida e qual o estágio do *pipeline* associado com aquela palavra. Depois há a definição da rede IP, os bits que indicam se há filhos presentes na tabela, um bit para indicar se há sub-redes e o número da interface de saída.

3.3 Arquitetura da Máquina de Encaminhamento

Esta seção detalha o projeto da máquina de encaminhamento do roteador. O desejável é obter um projeto que possa ser facilmente modificado para outros tamanhos de tabela, de interfaces de saída e até para utilização com IPv6.

Apenas foi implementada a operação de busca. Como as operações de inserção e remoção de rotas acontecem com freqüência muito menor e o tempo para executá-las não é crítico para o desempenho do sistema, elas podem ser realizadas por software. No entanto, o projeto foi feito prevendo uma futura criação destas operações.

A seguir é mostrado o diagrama de blocos da máquina de encaminhamento. As setas indicam a direção dos dados. Setas sem origem são entradas primárias do circuito. Setas sem destino são saídas. Para simplificação não são mostrados todos os fluxos de dados já que os módulos serão detalhados mais adiante. Deve-se considerar que este diagrama mostra somente as comunicações existentes.

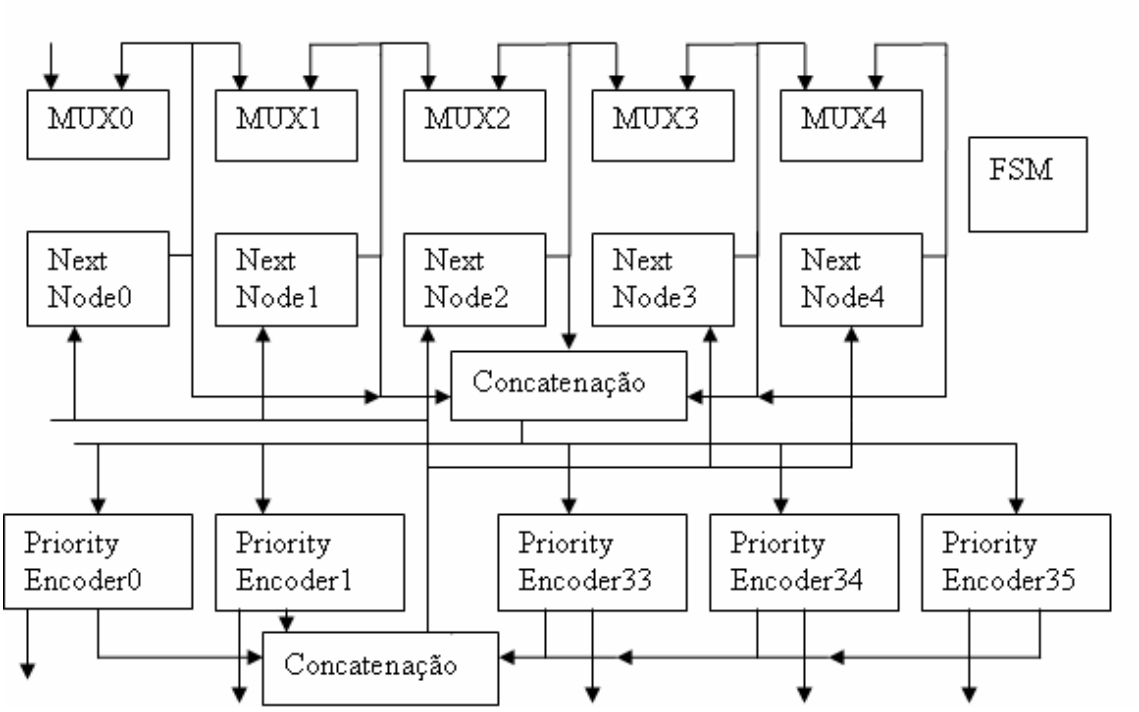

Figura 3.3: Diagrama de blocos da máquina de encaminhamento

Os módulos MUX são simples multiplexadores e não merecerão maior detalhamento. Da mesma maneira, os blocos concatenação simplesmente unem os sinais dos blocos e também não precisam ser explicados.

Os blocos *NextNode* recebem os dados da memória e decidem qual a interface de destino de um pacote. Os blocos *PriorityEncoder* estão ligados com uma interface de saída e decidem qual pacote transmitir. O bloco FSM é uma máquina de estados finitos que manda sinais de controle para os demais do circuito. Já que ela se comunica com todos os blocos *NextNode* e MUX, as comunicações foram omitidas. O objetivo dela é controlar o momento quando um pacote deve ser mandado para o próximo estágio. A chegada de um pacote de um estágio é a condição para trocar o sinal de seleção do multiplexador. Este controle é feito através de um contador para cada estágio.

Também é importante ressaltar que, seria possível criar até 128 interfaces de saída. No entanto, por questões de tempo, a síntese lógica demora muito para ser realizada com todas as interfaces, cada interface demora mais de um minuto para ser sintetizada, inicialmente apenas 36 foram criadas e apenas cinco são representadas no diagrama acima.

Para que um novo pacote possa ser transmitido, é necessário esperar que o sinal de pronto do circuito esteja no nível lógico alto. Quando isto acontecer, deve-se colocar as informações do pacote na entrada do circuito e setar o sinal que indica um novo dado. Este sinal deve ser zerado no próximo ciclo de relógio ou será feita uma nova transmissão que será ilegal, pois o pino de pronto estará com o valor zero. Quando o pacote estiver pronto para ser transmitido, será mandado um sinal para a interface de saída para que ela faça a transmissão. A tabela a seguir mostra os pinos de entrada e saída do projeto.

Pino	Descrição	Direção	Tamanho (bits)
iAddress	Endereço IP	Entrada	32
iValid	Sinaliza que existe um dado novo na entrada do circuito	Entrada	um
iClk	Sinal de Relógio.	Entrada	um
iRst	Reset Assíncrono.	Entrada	um
iPid	Identificador do pacote.	Entrada	32
oValid	Indica que há pacote para ser transmitido pela interface.	Saída	36 bits
oInterfaceOut	Identificador dos pacotes para cada interface.	Saída	36*8
Done	Indica se o primeiro estágio do *pipeline* está liberado.	Saída	um bit
tMemRd	Requisição de leitura da memória.	Saida	cinco
tMemRdWord	Dados lidos das memórias	Entrada	5*48 bits

Tabela 3.2: Descrição dos pinos de entrada e saida

Quando acontece uma borda de relógio, o valor de *iValid* é checado. Se tiver o valor um então deve acontecer uma nova transmissão para o endereço determinado por *iAddress*, que indica qual o endereço IP de destino do pacote. Ele também poderia ser usado nas operações de inserção e remoção da tabela. Neste caso, indicaria, respectivamente, qual o endereço de rota a ser adicionado ou removido da tabela. Enquanto o sinal *done* apresentar o valor lógico zero, *iValid* não deve ser setado.

É considerado que existe uma outra memória onde ficam armazenados os pacotes que esperam para serem transmitidos. O sinal *iPid* simplesmente identifica o pacote nesta memória. Desta maneira, o circuito pode se preocupar apenas com o IP de destino que é campo do datagrama que interessa.

O sinal *tMemRdWord* possuí os dados que são lidos da memória. Assim como *oInterfaceOut* e *oValid*, ele é a concatenação dos dados lidos de cada memória existente.

Os vetores sinais *oInterfaceOut* e *oValid* são a concatenação dos sinais que são mandados para cada interface de saída de modo que cada uma lê apenas alguns bits destes sinais. Quando um pacote é direcionado para a sua interface de destino, os sinais *oInterfaceOut* e *oValid* recebem, nos bits correspondentes àquela interface, respectivamente, os valores *iPid* do pacote e um lógico que serve para indicar que deve acontecer uma transmissão. Os demais bits dos vetores recebem o valor zero.

.O tempo necessário para fazer uma busca na árvore é no pior caso igual à altura da árvore. Isto pode ser um problema para tabelas grandes já que o número máximo de rotas que se deseja armazenar determina a altura da árvore. Lembrando que uma árvore

binária com altura h pode armazenar até 2^h -1 nodos, com o símbolo ^ indicando a operação de potenciação.

A árvore que armazena a tabela tem altura 15. Como conseqüência, é possível armazenar até 32767 rotas e o tempo de espera entre duas transmissões é de no máximo 15 ciclos de relógio. Para melhorar o desempenho, foi decidido criar uma estrutura de *pipeline*. Para isso, divide-se a tabela em memórias independentes de tal maneira que cada estágio do *pipeline* recebe dados de uma memória. Foram criados cinco estágios de *pipeline* com cada um realizando acesso a três níveis da árvore. Desta maneira, o tempo máximo de espera para que seja possível uma nova transmissão é de três ciclos de relógio. A figura 3.4 mostra a forma de onde para a busca da rota de destino de um pacote com endereço IP de destino A = B6.9D.60.CE considerando a memória da tabela 3.1. Apenas os sinais principais são mostrados.

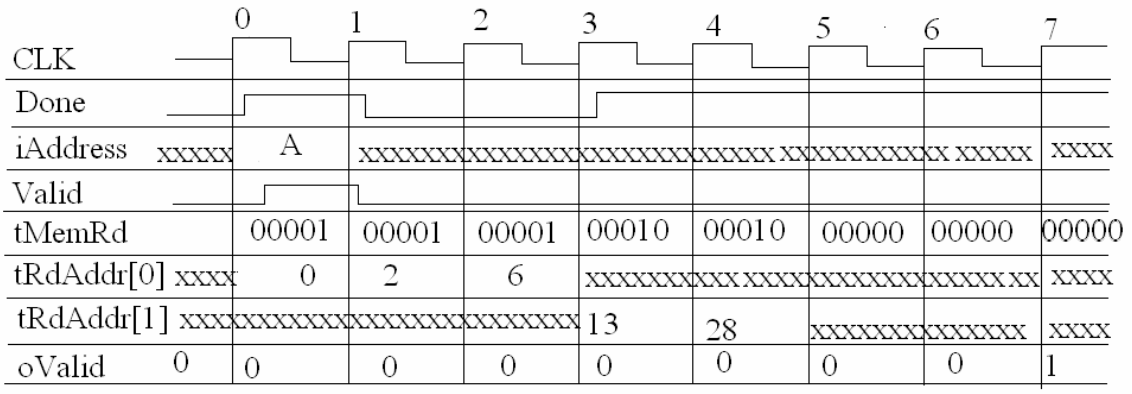

Figura 3.4: Formas de onda da operação de busca na tabela

No ciclo zero, o sinal *done* vai para um possibilitando que o sinal *valid* seja acionado e que o endereço IP seja colocado na entrada do circuito. Também é feita a requisição de leitura do endereço zero do primeiro estágio do *pipeline* que é a raiz da árvore. No segundo ciclo, os sinais de *done* e *valid* são desativados e, como o endereço de destino B6.9D.60.CE é maior que A5.4F.59.00/24 que é a rede de destino na posição zero da tabela e ela não é um possível destino para o pacote, faz-se apenas a requisição de leitura do endereço 2 do primeiro nível da memória. No ciclo dois, novamente apenas acontece a requisição de leitura da próxima posição da memória, que é o filho da direita. No ciclo seguinte, o pacote deve ir para o próximo nível e, portanto, a leitura é feita da segunda memória. Quando o nodo 13 estiver for analisado, o circuito perceberá que a rede destino B6.9D.60.C0/26 é uma possível rota para o pacote IP, porém como o bit *subnet* esta ligado não é possível pedir a transmissão do pacote. No ciclo cinco, o nodo processado é 28 cuja rede é B6.9D.60.CE/31. Esta rede é uma candidata melhor que a anterior, pois tem uma máscara com uma quantidade maior de uns e, como o bit *subnet* esta desligado, o pedido de transmissão pode ser feito. No ciclo seis, o módulo *PriorityEncoder* processa o pedido e, no ciclo sete, a interface 29 é avisada que é há um pacote para ser transmitido por ela.

3.4 HIERARQUIA

Esta seção apresenta e explica a hierarquia do projeto. Existem dois blocos principais chamados *NextNode* e *PriorityEncoder* que já foram apresentados na figura 3.3 que mostra o diagrama de blocos do projeto.

3.4.1 NextNode

Este é o bloco que é responsável por determinar a interface de saída do pacote. Ele determina se o endereço de rede salvo na palavra da memória pode ser o destino do pacote e, em caso positivo, se deve substituir algum destino previamente calculado. Além disso, o bloco também determina qual o próximo nodo da árvore que deve ser visitado. Como cada nível do *pipeline* tem uma memória independente, este bloco também aparece uma vez em cada estágio.

O princípio de funcionamento deste módulo já foi brevemente explicado na seção 3.1. O circuito deve pedir uma transmissão para o pacote quando: houver um pedido anterior não atendido ou quando o próximo nodo a ser pesquisado não estar presente na árvore ou se a rota da memória for melhor do que a que está sendo usada e não houver uma outra rota possível para o pacote. Quando for achada uma nova interface de destino, mas houver a possibilidade dela não ser a final, então as informações referentes à interface de destino e ao tamanho da máscara são atualizadas sem haver o pedido de transmissão. O diagrama de blocos é mostrado na figura 3.5.

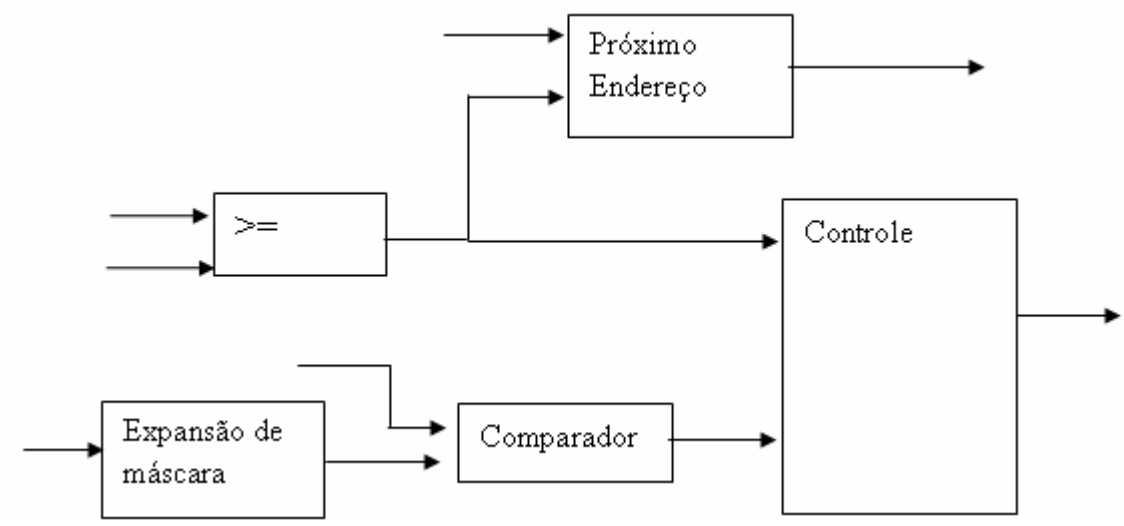

Figura 3.5: Diagrama de blocos do módulo NextNode

O bloco de expansão de máscara é o responsável por gerar a máscara de rede a partir da quantidade de uns que é guardada na memória. Ele poderia ser removido caso a máscara inteira fosse armazenada, porém seriam necessários 26 bits a mais para cada posição. Optou-se por aumentar o tamanho do circuito já que é mais importante otimizar a quantidade de memória utilizada. A saída é mandada para o comparador que determina se o endereço de rede salvo na memória é compatível com o destino do pacote. O bloco com o símbolo >= compara se o endereço de destino do pacote é maior ou igual ao guardado na tabela. Esta informação é, depois, usada para determinar qual o

próximo endereço a ser buscado. O bloco controle é o responsável por determinar se deve ser requisitada uma transmissão ou não.

A tabela 3.3 mostra a descrição dos pinos de entrada e saída. Após, é feita uma explicação do funcionamento do circuito. Por questões de simplicidade, os sinais de relógio e *reset* não são mostrados. Um sinal com tamanho variável tem uma quantidade diferente de bits para cada estágio do *pipeline*. Um endereço do primeiro nível usa três bits para ser determinado já que são sete nodos existentes e sempre que se passa para o próximo nível são necessários três bits a mais do que anteriormente, o segundo necessita de seis, o terceiro de nove, o quarto de doze e o quinto de quinze.

Pino	Descrição	Direção	Tamanho
pid	Identificação do pacote.	Entrada	32 bits
memWord	Palavra lida da memória.	Entrada	48 bits
currentAddr	Endereço da memória atual.	Entrada	Variável
dstIp	Endereço IP de destino.	Entrada	32 bits
mask	Tamanho atual da máscara.	Entrada	6 bits
res	Transmissão realizada.	Entrada	1 bit
eth	Atual interface de destino.	Entrada	8 bits
ctl	Pedido de transmissão.	Entrada	1 bit
change	Troca de estágio do *pipeline*	Entrada	1 bit
valid	Dados válidos	Entrada	1 bit
rdAddr	Próximo endereço.	Saída	Variável
oDstIp	Endereço IP de destino.	Saída	32 bits
oMask	Tamanho da máscara de rede.	Saída	6 bits
oPid	Identificador do datagrama	Saída	32 bits
Octl	Pedido de transmissão.	Saída	1 bit
oeth	Nova interface de destino	Saída	8 bits

Tabela 3.3: Pinos de entrada e saida do módulo NextNode

Se pelo menos um dos sinais *reset* e *res*, que indica que o pacote sendo processado já foi transmitido, estiver com o valor um, as saídas tem o valor zerado. Caso contrário, é checado se o valor do *ctl*. Se este sinal estiver no nível lógico alto, significa que a interface de saída já determinada anteriormente e deve-se apenas transmitir os valores *dstip, mask, eth, pid e ctl* para as saídas *odstip, oeth, omask, opid* e *octl*. Quando *ctl* apresentar o valor zero, deve-se decidir se uma transmissão deve ser requisitada.

Uma transmissão será requisitada se o nodo for uma folha ou se a rede determinada pela palavra da memória for uma rota possível e tiver uma máscara maior

do que a definida pelo sinal e *mask* e não houver sub-redes para esta rede, bit *subnet* com o valor 1. Nestes casos, os valores de *omask* e *oeth* devem ser atualizados de acordo com a informação do nodo.

Quando a rota determinada pelo nodo for possível e tiver uma máscara maior, mas houver sub-redes na árvore, então apenas se atualiza os valores de *omask* e *oeth*, porém não se requisita uma transmissão.

Este módulo também é responsável por determinar o próximo a ser acessado. Em uma árvore binária normal, o filho da esquerda do endereço k tem endereço 2*k + 1 e o filho da direita 2*k+2. Neste projeto, porém, deve-se fazer o cálculo de uma maneira diferente. Como a tabela é uma árvore dividida em várias memórias, existem diversas árvores. Então, as memórias de todos os níveis, exceto o primeiro, não terão apenas um nodo raiz. Se houver n nodos no primeiro nível da árvore, o endereço do filho da esquerda é 2*k + n e 2*k+n+1 para o filho da direita.

Outra consideração importante é que, sempre que o pacote for para o próximo nível do *pipeline*, será necessário acessar uma memória diferente, cujo endereço é zero e, portanto, não é contínuo com a atual memória. Como conseqüência, o cálculo do próximo endereço deve compensar este fato. Isto é feito subtraindo do resultado usual o total de nodos do estágio atual. Por exemplo, quando um pacote vai do primeiro para o segundo estágio deve-se subtrair sete do resultado. A figura 3.6 ilustra esta situação.

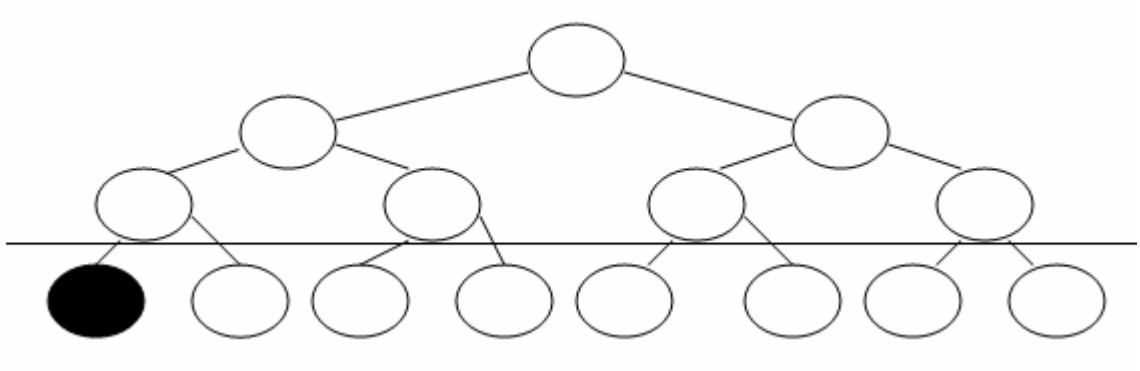

Figura 3.6: Exemplo de corte da tabela

A linha separa os nodos que pertencem a memórias distintas. Ou seja, quando um pacote for para um nodo de baixo, ele trocará de estágio no *pipeline* e, portanto, deve arrumar o valor de endereço para que o endereçamento fique correto. O nodo preto ocupa a posição zero e não oito. Os seus filhos estão nas posições oito e nove.

3.4.2 PriorityEncoder:

Há um bloco deste para cada interface de saída. Este bloco recebe os pedidos de transmissão de pacotes de cada estágio do *pipeline* e decide qual pacote deve ser mandado para a interface. É usada uma fila de prioridades, os pacotes que percorreram uma altura maior na árvore têm uma prioridade maior do que os demais. A maior prioridade é dada para um pacote que está no último estágio. O funcionamento deste bloco consiste em, simplesmente, verificar quais estágios querem transmitir por aquela interface e ver qual tem a maior prioridade. Por vezes no texto, este bloco será referido

simplesmente como interface ou interface de saída. O diagrama de blocos do circuito é mostrado a seguir na figura 3.7.

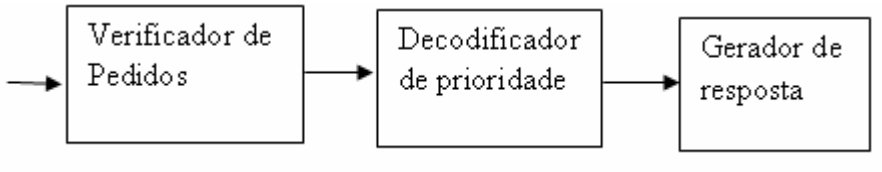

Figura 3.7: Diagrama de blocos do módulo PriorityEncoder

O bloco verificador de pedidos analisa quais estágios do *pipeline* querem uma transmissão por aquela interface. O resultado desta computação é passado para o próximo módulo que decide qual nível terá direito à transmissão de acordo com o sistema de prioridade e o gerador de resposta que cria os sinais de saída adequados para a computação realizada. A tabela 3.4 mostra os pinos deste módulo. Novamente, os sinais de relógio e *reset* assíncrono não são mostrados.

Nome	Descrição	Direção	Tamanho
Sel	Pedidos de transmissão de cada estágio.	Entrada	5 bits
Eth	Número das interfaces de saídas	Entrada	40 bits
Ipid	Identificador do pacote	Entrada	32*5 bits
Iaddr	Endereços IP de destino	Entrada	32*5 bits
Res	Resposta para os estágios.	Entrada	5 bits
Opid	Identificador do pacote	Saída	32 bits
Addr	Endereço IP de destino	Saída	32 bits
Valid	Dados de saída são válidos	Saída	1 bit

Tabela 3.4: Pinos de entrada e saida do módulo PriorityEncoder

Em cada ciclo de relógio, o módulo verifica quais bits do vetor *sel* são iguais a um. Estes são os estágios que querem transmitir um pacote. No entanto, Como cada bloco *PriorityEncoder* está conectado com uma interface de saída, é necessário decidir se aquela é a correta. Como cada módulo sabe com qual interface está conectado, basta uma comparação com os bits do sinal *eth* correspondentes aos estágios que querem transmitir. Os pacotes que passarem nos dois testes são os candidatos a transmissão. Entre eles é escolhido o de maior prioridade. Ou seja, aquela que estiver mais próximo do fim da árvore. Como o pacote que foi para a interface continua no *pipeline*, é necessário mandar uma resposta as instâncias do módulo *NextNode* avisando se o pacote sendo processado já foi enviado, o sinal *Res* é o responsável por este aviso.

Uma característica importante do projeto é a capacidade de transmitir pacotes em uma ordem diferente da recepção. Por exemplo, se um pacote tiver a interface de destino determinada já no primeiro estágio e o pacote imediatamente anterior a ele só tiver a sua interface decidida no último estágio, o datagrama que chegou depois será transmitido antes. É importante lembrar que o protocolo IP não exige que a ordem seja respeitada.

Há um problema de sincronização entre os módulos *PriorityEncoder* e *NextNode*. Se no ciclo zero, o módulo *Nextnode* determinar a interface de saída de um pacote, então este pedido chegará ao *PriorityEncoder* no ciclo um e a resposta só chegará no *NextNode* no ciclo dois . No entanto, no ciclo um o pacote foi processado por um bloco *NextNode* e um novo pedido de transmissão foi feito, mesmo que o pacote tenha sido transmitido, já que não havia uma resposta ainda.

Para evitar este problema é possível retirar o *flip-flop* de saída de um dos blocos, aumentando o atraso do circuito. Desta maneira, a resposta seria dada no próximo ciclo e não haveria o segundo pedido. Outra abordagem, que é a adotada, é fazer com que o módulo *PriorityEncoder* guarde se foi feita alguma transmissão e qual pacote foi transmitido, desta maneira, basta que ele ignore o segundo pedido.

3.5 VERIFICAÇÃO DE FUNCIONAMENTO

Após o código VHDL ser escrito, é necessário testá-lo para verificar se o que foi escrito apresenta o comportamento desejado. No entanto, apenas em projetos muito simples é possível testar todos os vetores de entrada possíveis. Como conseqüência, não é possível ter certeza que o circuito irá funcionar sempre corretamente. Para aumentar a confiança no correto funcionamento do projeto, deve-se, então, escolher vetores de teste que cubram de melhor maneira possível os casos de funcionamento do circuito.

Outro fator importante é projetar o teste de maneira a identificar de maneira fácil o local do erro. Se o código sendo testado for muito grande e apresentar erros, pode ser difícil de encontrar o local da falha. É, portanto, indicado testar cada módulo separadamente antes do teste do circuito final.

Neste projeto, cada bloco descrito anteriormente foi testado separadamente e, apenas quando todos foram aprovados, o circuito final foi testado. A metodologia utilizada foi igual para cada módulo, porém respeitando as diferenças entre eles. Para cada módulo, um conjunto de vetores de teste foi escolhido manualmente de maneira a garantir que alguns casos especiais fossem testados e, depois, os demais casos foram escolhidos de maneira aleatória.

Neste projeto, os testes foram escritos em VHDL, mas poderiam ter sido feitos em alguma outra linguagem qualquer como C, Java ou Verilog. No restante do texto, o termo *testbench*, nomenclatura de VHDL, é utilizado como sinônimo de teste.

Em ambas as situações, o *testbench* determina o resultado esperado e caso ele seja diferente da resposta obtida é mandado um sinal de alerta. É interessante observar que, como o *testbench* também é um código ele também pode conter erros. Portanto, não é possível determinar automaticamente qual dos dois está errado. No entanto, como o *testbench* não precisa ser sintetizado, a descrição da unidade que prevê o resultado

acaba sendo mais simples que a do circuito sob teste. Então, as chances de erros ocorrerem são menores no *testbench*. De qualquer maneira, as descrições devem ser checadas quando as respostas divergem. Neste projeto, considerou-se que quando as duas respostas são iguais, o comportamento do circuito está correto. O *testbench* pode ser visto como o seguinte diagrama de blocos.

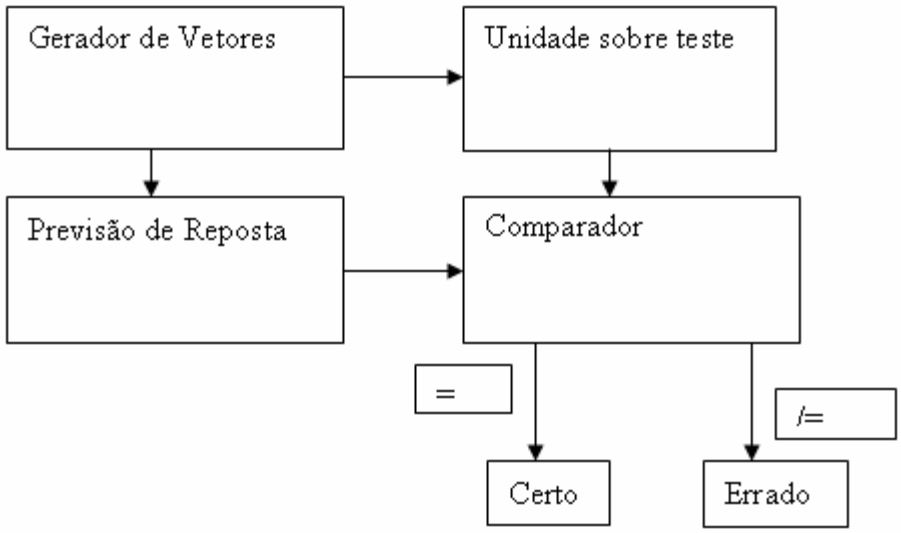

Figura 3.8: Diagrama de blocos do Teste

O módulo gerador de vetores cria os casos de testes que são transmitidos para os blocos unidade sobre teste, que é o circuito sendo testado, e previsão de resposta, que determina a resposta esperada. As saídas destes dois módulos são mandadas para o comparador, se elas forem iguais, a computação é considerada correta. Senão, houve um erro. O restante da seção detalha o procedimento de teste utilizado para cada módulo.

Os casos de teste são decididos estudando os comportamentos possíveis do circuito e definindo valores de entradas pertinentes para cada situação. Considerando o bloco *NextNode* e o funcionamento do algoritmo descrito anteriormente podemos determinar os casos de teste que se encaixam, essencialmente, em seis categorias. Lembrando que, como mencionado no capítulo, cada entrada na tabela determina um intervalo [a,b] de endereços IP que podem ter aquela interface como saída. Nos itens a seguir, w representa o endereço IP de destino do pacote, x e y representam, respectivamente, o tamanho da máscara da entrada da tabela e o tamanho da máscara de uma rede IP candidata visitada anteriormente.

a)O endereço de destino do nodo é um possível destino para o pacote e a memória tem uma máscara maior do que a sendo usada (a <= w <= b e x>y) .

b)O endereço de destino do nodo é um possível destino para o pacote e não tem uma máscara maior do que a sendo usada (a <= w <= b e x<y).

c)O endereço de destino do nodo não é um possível destino para o pacote, mas tem uma máscara maior do que a sendo usada ((a > w ou w > b) e x>y).

d)O endereço de destino do nodo não é um possível destino para o pacote e não tem uma máscara maior do que a sendo usada ((a > w ou w > b) e x < y).

e)O próximo nodo é o filho da esquerda (a > w).

f)O próximo nodo é o filho da direita (a < w ou a = w).

Também devem ser testados os comportamentos com as entradas *ctl*, *change*, *res*, *valid* com valores diversos. Além disso, os campos folha e *subnet* da palavra de memória também têm influência.

A primeira etapa do teste foi criar um vetor de teste para cada um dos itens. Desta maneira, adquire-se confiança que o circuito funciona corretamente em algumas situações. A segunda etapa é a parte automática do teste. Uma palavra de memória era gerada aleatoriamente e, então, os demais sinais eram gerados para testar os casos de ‘a’ a ‘f’. Este processo foi repetido para diversas palavras de memória.

Os vetores de teste randômicos podem ser gerados da seguinte maneira: inicialmente escolhe-se um tamanho x para a máscara e um número de 32 bits. A máscara é aplicada a este número e o resultado forma o endereço da rede de destino armazenada na tabela. Para testar os casos ‘a’ e b modificam-se alguns da parte de *host* para formar o endereço IP de destino e diminui-se ou aumenta-se o valor da máscara dependendo de qual caso deseja-se testar. Para os casos ‘c’ e ‘d’ os bits modificados são os do endereço de rede e, novamente, o tamanho da máscara é trocado de acordo com o caso desejado. Os casos ‘e’ e ‘f’ são testados juntamente com os demais. No entanto, os casos ‘a’ e ‘b’ sempre testarão o caso ‘f’. Os valores dos demais bits folha e *subnet* podem ser gerados através de um contador.

A tabela 3.5 mostra exemplos de teste para este módulo. A primeira coluna indica qual dos casos acima esta sendo testado, as próximas quatro são valores dos sinais de entrada, um X indica que aquele bit não influencia no resultado e as próximas duas indicam se haverá uma requisição de transmissão do pacote e se a interface de saída do pacote será modificada.

Caso	Ip memória	Ip destino/ Máscara atual	Folha	Subnet	Requisição transmissão	Atualização interface
A,f	143.54.09.00/24	143.54.09.145/20	X0	1	0	1
A,f	143.54.09.00/24	143.54.09.145/20	XX	0	1	1
B,f	143.54.09.00/24	143.540.9.145/28	X0	X	0	0
B,f	143.54.09.00/24	143.54.09.145/28	X1	X	1	0
C,e	143.54.09.00/24	115.00.00.04/28	11	X	1	0
c,f	143.54.09.00/24	150.00.00.04/28	00	X	0	0
d,e	143.54.09.00/24	115.00.00.04/20	01	X	0	0
d,f	143.54.09.00/24	150.00.04/20	10	X	1	0

Tabela 3.5: Casos de teste para módulo *NextNode*

Um pedido de transmissão só pode ocorrer no caso a, se o bit de *subrede* estiver zerado, ou se o próximo nodo que deveria ser visitado não estiver na tabela. A atualização da interface de saída só ocorre no caso a. Os sinais *valid, ctl, res e reset* não aparecem na tabela porque eles têm um comportamento que predomina sobre os demais sinais. Ou seja, quando estes sinais estão ativos, os dados lidos da memória não influenciam na determinação dos valores da saída. O sinal *change* só influencia no cálculo do próximo endereço e também pode ser testado separadamente.

O módulo *PriorityEncoder* é um pouco mais simples de ser testado. Basicamente, deve-se testar se o módulo consegue identificar corretamente se aquela é a interface de saída correta e se esta aplicando as prioridades corretamente.

A primeira etapa do teste foi verificar a capacidade do circuito em atribuir as prioridades corretamente. Como são cinco estágios de *pipeline*, há cinco sinais de requisição de transmissão e, portanto, são apenas 32 possibilidades que podem ser testadas exaustivamente. O outro dado importante é se a interface de saída do pacote é a mesma que está associada com a instância do módulo. Portanto, o valor da interface de saída pode ser considerado como sendo apenas um bit significando igual ou diferente. Há, então, 32 possibilidades para este sinal. Combinando com as 32 possibilidades do sinal de requisição de transmissão, há 1024 casos de teste.

Para não dificultar o procedimento, o sinal de *reset* foi testado separadamente, sendo setado de maneira aleatória durante uma segunda bateria de testes. A tabela a seguir mostra alguns exemplos de teste.

Requisição de transmissão (sel)	Número da interface de saída	Interface de saída	Resposta (Res)
00000	5	{5,5,5,5,5}	00000
00001	5	{5,5,5,5,5}	00001
00011	5	{5,5,5,5,5}	00010
00111	5	{5,5,5,5,5}	00100
01111	5	{5,5,5,5,5}	01000
11111	5	{5,5,5,5,5}	10000
00001	5	{5,5,5,5,4}	00000
00011	5	{5,5,5,4,4}	00000
00111	5	{5,5,4,5,5}	00010
01111	5	{5,4,5,5,5}	00100
11111	5	{4,4,4,4,5}	00001

Tabela 3.6: Casos de teste para módulo *PriorityEncoder*

Após, foi realizado o teste do projeto completo. É importante ressaltar que este teste é bem mais complicado de ser elaborado já que a saída do circuito em um

determinado ciclo de relógio depende dos últimos dezesseis valores de entradas. Já que o tempo máximo para um pacote ser processado é quinze ciclos que é a altura da árvore mais um ciclo para ser mandado para a interface correta.

Foi criada uma pequena memória com valores determinados arbitrariamente. Após, foram escolhidos vetores de teste de maneira a ser possível fazer uma verificação manual dos resultados.

Para exemplificar o funcionamento do teste, será usada a tabela de roteamento da seção 3.1. Para facilitar o teste, foi calculado, para cada entrada na tabela, o intervalo de endereços IP que têm aquela rede como a rede de destino. A tabela abaixo mostra os intervalos.

Endereço de memória	IP/Máscara	Intervalo IP
0	A5.4F.59.00/24	[A5.4F.59.00, A5.4F.59.3F]
2	A5.4F.59.80/25	[A5.4F.59.80, A5.4F.59.FF]
5	A5.4F.59.40/26	[A5.4F.59.40, A5.4F.59.7F]
6	B7.9D.60.40/26	[B7.9D.60.40, B7.9D.60.7F]
13	B6.9D.60.C0/26	[B6.9D.60.C0, B6.9D.60.FF]
27	B5.00.00.00/8	[B5.00.00.00, B5.FF.FF.FF]
28	B6.9D.60.CE/31	[B6.9D.60.CE, B6.9D.60.CF]
57	B6.9D.60.C1/32	B6.9D.60.C1

Tabela 3.7: Tabela de roteamento com intervalos de rotas

A tabela ser pequena ajuda a verificação manual dos resultados. No entanto, ela fornece uma boa cobertura de casos já que possui todos os valores possíveis para o campo folha, há a presença de sub-redes e há uma troca de estágio de *pipeline* para verificar se o cálculo do endereço é feito corretamente. Os valores da interface de saída foram escolhidos como sendo o endereço mais um, desta maneira, a interface zero será o destino de pacotes que não tem rota definida na tabela. Para fazer o teste basta escolher um dos intervalos, gerar algum endereço dentro dele e verificar se a transmissão é feita pela interface esperada.

A segunda etapa do teste foi criada com valores randômicos tanto para a tabela quanto para os sinais de entrada. Uma diferença importante em relações aos outros módulos é que a verificação dos resultados não é feita ao mesmo tempo em que a simulação. Pois o módulo de previsão precisaria avaliar as últimas entradas como já explicado. Portanto, as respostas do circuito foram salvas em um arquivo para serem comparadas com as respostas previstas pelo *testbench*. De fato foi apenas verificado se os pacotes estavam sendo direcionados para a interface correta.

É importante ressaltar que, se todos os nodos da árvore tiverem o bit *subnet* em um, os pacotes serão mandados para a interface correta. Então, para garantir a existência de uma tabela válida durante o teste, em um primeiro momento, este bit foi fixado em

um. Esta opção foi tomada para não ser necessário decidir o valor correto e simplificar o procedimento.

Depois, foi percebido que a tabela não ser válida não é realmente um problema para o teste. Basta que os resultados sejam coerentes com o conteúdo da memória. Então alguns bits *subnet* tiveram o seu valor alterado para aumentar a abrangência.

4 FLUXO DE PROJETO ASIC

Este capítulo apresenta o fluxo de projeto utilizado neste trabalho. A primeira seção apresenta uma visão geral dos passos envolvidos no projeto e a segunda explica o que foi realizado em cada etapa.

4.1 Descrição do Fluxo

Um circuito integrado pode ser definido em três domínios diferentes: comportamental, estrutural e físico. O comportamental define o objetivo que se deseja alcançar. O estrutural explica a conexão de componentes necessárias para se conseguir o comportamento necessário e o físico determina a posição dos elementos para conectá-los e obter o funcionamento esperado.

Um fluxo de projeto de ASIC's pode ser definido como sendo um conjunto de passos que têm como objetivo transformar a especificação do projeto em um circuito fabricável que funciona de acordo com o especificado. O fluxo a ser adotado neste trabalho é definido a seguir.

4.1.1 Fluxo de Projeto de Síntese Comportamental

Nesta etapa se consegue observar qual o comportamento que o circuito final irá ter sem se conhecer a implementação final. Esta é a fase que tem a maior independência em relação à tecnologia utilizada e pode ser dividida em diversos estágios. A figura 4.1 mostra o fluxograma desta etapa do projeto. Após, cada etapa é explicada de maneira mais precisa. Apesar de não aparecer na figura, há realimentações de cada etapa para as anteriores para consertar passos que não obtiveram resultado satisfatório.

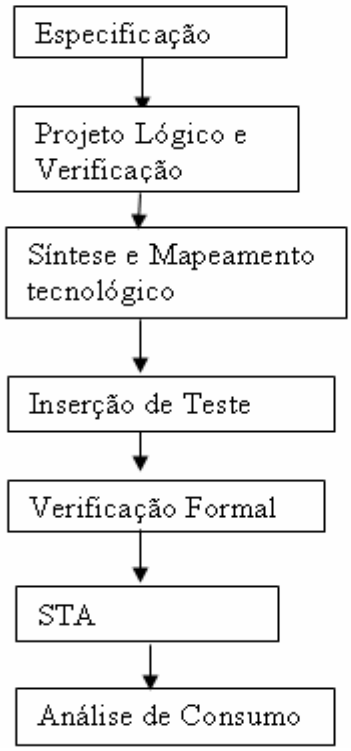

Figura 4.1: Fluxo de projeto de síntese comportamental

4.1.1.1Especificação

Esta fase determina o que o circuito deve fazer e os requisitos necessários para fazê-lo. Em um projeto industrial, é, possivelmente, a etapa mais complicada e que exige mais tempo para ser realizada, pois não há ferramentas que automatizem o processo. Além disso, os objetivos aqui estabelecidos terão impactos em todas as etapas posteriores já que eles são as referências para se comparar o resultado de cada fase do desenvolvimento. Uma especificação que estabeleça condições muito fáceis de serem atingidas fará com que, provavelmente, o produto final seja rapidamente obtido, mas não será capaz de competir no mercado. Por outro lado, uma especificação que seja demasiadamente exigente, pode impedir a realização do projeto em tempo hábil. Outra dificuldade é definir a função do circuito de um jeito formal, que não permita diferentes interpretações dos objetivos a serem alcançados.

4.1.1.2Projeto Lógico e Verificação

Partindo da especificação, escreve-se uma descrição *Register Transfer Level* (RTL) do circuito em alguma Linguagem de Descrição de Hardware (HDL). Após, cria-se um *testbench* para verificar que o projeto desempenha a função esperada. É importante lembrar que, como na maioria dos projetos não é possível testar todos os valores de entrada, apenas é possível afirmar que o circuito funciona para os casos que forem testados.

As duas principais HDL são VHDL e Verilog. A primeira é a mais usada pela indústria européia e a segunda é a principal nos Estados Unidos. Outras linguagens

foram desenvolvidas como SystemC ou HandelC, estas linguagens permitem que o circuito seja definido em um nível mais alto de abstração, assemelham-se mais com uma linguagem de programação, mais especificamente com C. A linguagem usada neste trabalho é a VHDL. Mais especificamente, a descrição do circuito a ser fabricado será feita usando apenas a parte da linguagem que é sintetizável.

Um *testbench* é uma descrição de um circuito que testa, automaticamente, se o projeto realiza a função lógica esperada. Como ele apenas é usado em simulações, podem-se utilizar construções da linguagem VHDL que não são sintetizáveis. O *testbench* estimula o circuito, observa a saída e compara o comportamento com o esperado, indicando se existe algum erro na lógica implementada. Para esta etapa do projeto, o ambiente ModelSim é utilizado.

4.1.1.3Síntese RT L e Mapeamento tecnológico

A descrição anterior é transformada em um conjunto de portas lógicas e registradores. Após, aplicam-se otimizações para melhorar os resultados de área, consumo e velocidade do projeto. As portas lógicas usadas nesta fase são genéricas, ou seja, não foram desenvolvidas para uma tecnologia alvo. O trabalho usa a ferramenta RTL Compiler da Cadence.

A descrição usando as portas lógicas genéricas não possibilita a manufaturação do circuito. Para que ela seja possível, é necessário determinar uma tecnologia para ser utilizada. Esta etapa converte a descrição genérica em uma que depende do processo a ser usado, utilizando uma biblioteca de células específica para a tecnologia. Este processo é conhecido por mapeamento tecnológico.

A biblioteca fornece informações de área, consumo e atraso sobre as células, assim é possível, a partir desta etapa, estimar o desempenho do circuito. No entanto, ainda não há informações sobre a ligação entre as células e isto limita significativamente a precisão de qualquer análise feita. O resultado desta fase é chamado de Verilog ou VHDL mapeado.

A criação da biblioteca pode ser considerada uma etapa separada, pois ela nem sempre é fornecida pela empresa responsável pela manufaturação do circuito. Quando se deseja gerar uma biblioteca, há vários pontos que precisam ser considerados como: as funções lógicas, tamanhos das células e o método de caracterização elétrica que será empregado.

Há vários circuitos distintos que calculam a mesma função lógica, cada um com características diferentes de área, velocidade e consumo. Geralmente, para uma mesma tecnologia, circuitos mais rápidos são maiores e consomem mais. Como não é possível gerar todas as implementações possíveis para escolher a melhor, é necessário tentar direcionar o processo de síntese para que a ferramenta ache mais rapidamente uma solução adequada. Como o preço de fabricação de um circuito depende do tamanho dele, faz sentido escolher a menor implementação que satisfaça as restrições de velocidade e potência. Embora também seja possível diminuir o desempenho desejado para obter um custo de manufaturação menor.

4.1.1.4Verificação Formal ou Funcional

É necessário verificar que o circuito mapeado ainda realiza a mesma operação lógica da sua descrição comportamental ou RTL. Apesar de esperar-se que os processos de síntese e mapeamento não alterem a função do projeto, é possível que ocorra um erro que cause uma mudança no comportamento do projeto. Este erro, geralmente, é causado por uma descrição mal feita. Por exemplo, se uma lista de sensibilidade de um processo VHDL não estiver completa, pode acontecer da ferramenta de síntese gerar um circuito que se comporte diferente do desejado.

Uma possível opção para fazer esta verificação é utilizar o mesmo *testbench* que foi usado anteriormente. Se a resposta do circuito for a mesma, então a função lógica não foi alterada para aqueles vetores de entrada. Deste jeito, consegue-se de uma forma simples, para projetos que não sejam muito grandes, mostrar a equivalência entre o comportamento das duas descrições. No entanto, com o aumento da complexidade do projeto, são necessários cada vez mais vetores de teste e o tempo necessário para esta abordagem aumenta consideravelmente.

Pode-se também usar técnicas de verificação formal. Esta estratégia compara diretamente a função lógica das duas descrições e prova, matematicamente, que elas são equivalentes. Apesar de ser um processo mais complexo que o anterior, esta estratégia permite ter-se uma certeza maior que, de fato, a síntese e o mapeamento foram feitos corretamente. Além disso, conforme o tamanho do projeto aumenta, pode ser mais rápido fazer uma verificação formal do que a simulação já que a primeira opção não necessita de vetores de teste.

Pode-se também fazer uma análise semântica e estrutural par verificar que algumas características mais simples não foram modificadas. Checa-se, por exemplo, se a largura de algum barramento foi modificada ou se existe alguma saída desconectada.

4.1.1.5Static Timing Analysis (STA)

Não adianta o circuito desempenhar a função correta se ela não for feita com a velocidade necessária. Esta etapa verifica se os requisitos temporais são satisfeitos. Essencialmente, é testado se os tempos de *setup* e *hold* dos *flip-flops* são respeitados. O *setup* especifica quanto tempo antes da borda de relógio a entrada deve estabilizar e o *hold* quanto tempo após a borda de relógio a entrada deve permanecer estável.

A análise é dita estática, pois é independente de vetores de entrada. Ou seja, não são feitas simulações. Como conseqüência, é possível que não exista algum vetor de entrada capaz de sensibilizar o caminho que foi identificado pela ferramenta como sendo crítico, esta situação é chamada de caminho falso. Neste caso, o atraso é irrelevante. A ferramenta de STA pode tentar identificar estes casos, porém, esta verificação nem sempre é precisa, já que ela é feita simulando o circuito para alguns vetores e observando se algum excita o caminho crítico. A abordagem mais utilizada é o projetista definir manualmente os caminhos falsos e, caso nenhum caminho deste tipo seja conhecido, então se considera que não há caminhos falsos.

Os tempos de *recovery* e *remova*l também são analisados. As definições são análogas, respectivamente, as dos tempos de *setup* e *hold*, porém para sinais

assíncronos. O tempo de *recovery* especifica quanto tempo antes da borda ativa do sinal de relógio o sinal de *reset* ou *set* assíncrono deve ser desligado para que o *flip-flop* armazene o valor da entrada. O tempo de *removal* especifica quanto tempo após a borda de *clock* o sinal deve se manter desligado para que o *flip-flop* não o interprete como ativo na borda anterior.

Outra característica importante é a capacidade de analisar o comportamento para diferentes condições de funcionamento. Os atrasos das portas lógicas são fortemente influenciados pelo ambiente onde operam. Aumentos de temperatura e quedas na tensão de alimentação, por exemplo, diminuem a velocidade de um circuito. Se forem conhecidas a melhor e a pior condição de operação, é possível testar o circuito para estes dois casos e, caso ele respeite as especificações, assumir que o projeto opera corretamente nos demais casos.

4.1.1.6Inserção de Teste

Imperfeições no processo de fabricação de circuitos fazem com que diversas unidades produzidas não funcionem corretamente. É essencial que seja possível identificar quais circuitos funcionam corretamente e quais apresentam defeitos. Nesta etapa, modificam-se alguns registrados para facilitar a tarefa de teste.

Os testes para ASIC são baseados no modelo de travado-em-zero (*stuck-at-zero*) ou travado-em-um (*sutck-at-one*). Nestes modelos, uma falha é representada como sendo um sinal que tem o seu valor constante em zero ou um independentemente dos valores da entrada. O teste é baseado em duas características do circuito: a controlabilidade (*controlabillity*), que é a capacidade de determinar o valor de algum nodo a partir das entradas primárias, e a observalidade (*observallity*), que é a capacidade de observar o valor de um nodo em algum instante de tempo. Quanto maiores forem a controlabilidade e a observalidade do circuito, maior será a abrangência do teste que será feito.

No projeto inicial do circuito, apenas as saídas e entradas primárias são observáveis e apenas o primeiro bloco combinacional é controlável, diretamente, a partir de entradas primárias. Para melhorar a qualidade do teste são adicionadas novas entradas e saídas primárias para controlar e observar os estados dos *flip-flops* do circuito. Como foi adicionado um novo fio na entrada do *flip-flop*, é preciso um método para escolher qual das duas entradas deseja-se utilizar, a nova entrada primária ou a que existia no circuito inicial. Isto pode ser feito adicionando um multiplexador na entrada do *flip-flop*, o circuito resultante é chamado de *scan flip-flop* e o seu esquemático é mostrado na figura 4.2.

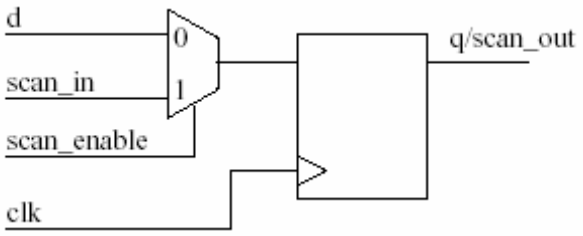

Figura 4.2: Scan Flip-Flop

Quando for desejável fazer o teste, o sinal *scan_enable* seleciona a entrada *scan_in* que é a entrada primária que foi adicionada. Quando o circuito estiver operando normalmente, a entrada do multiplexador selecionada é a *d*. A inserção destes *flip-flops* aumenta a área e o consumo do circuito. Portanto, deve-se escolher quais *flip-flops* devem ser trocados para possibilitar uma boa cobertura dos testes com um custo aceitável. Também é possível que haja uma redução na velocidade do circuito.

O circuito manufaturado é ligado a uma máquina de geração automática de geração de padrões, *Automatic Test Pattern Generator (*ATPG) que é responsável por gerar os estímulos de teste para o circuito. Os primeiros vetores de teste tendem a ser randômicos para tentar identificar rapidamente os locais onde pode haver erros. Os demais vetores são projetados para testar casos específicos.

4.1.1.7Análise de consumo

O consumo de um circuito é uma característica extremamente importante, podendo ser, muitas vezes, fator decisivo na escolha entre dois produtos semelhantes. Portanto, deve-se avaliar se o desempenho do circuito está dentro do aceitável.

Há dois tipos de consumo, o estático e o dinâmico. O estático é caracterizado pela energia gasta enquanto o circuito está parado devido as correntes de fuga.. O dinâmico corresponde à energia gasta devido ao chaveamento dos transistores, depende dos valores de entrada e do estado atual dos transistores, e é maior que o estático. A soma dos dois é o consumo total. De acordo com a tecnologia a relação entre os consumos estático e dinâmico pode ser tal que apenas o dinâmico é relevante para o circuito. No entanto, com a diminuição do tamanho dos transistores, a parcela correspondente ao consumo estático tende a ganhar importância para determinação do consumo total.

Como o consumo total depende da entrada do estado atual do circuito, uma análise precisa deve testar a maior quantidade possível de configurações. Obviamente, não é possível, a não ser em projetos pequenos, testar todas as possibilidades. É necessário escolher alguns vetores de entrada que avaliem adequadamente o consumo.

4.1.2 Geração automática de Layout

A segunda fase do fluxo também é conhecida por síntese física. O objetivo desta etapa é transformar o circuito mapeado anteriormente em uma descrição que possa ser fabricada de acordo com as regras de uma tecnologia. Os passos desta fase, com exceção da correção de regras de desenho e *layout* versus *schematic*, serão feitos no ambiente *Encounter* da *Cadence*. Esta parte do fluxo é representada pelo fluxograma da figura 4.3 e, novamente, não são mostradas as realimentações.

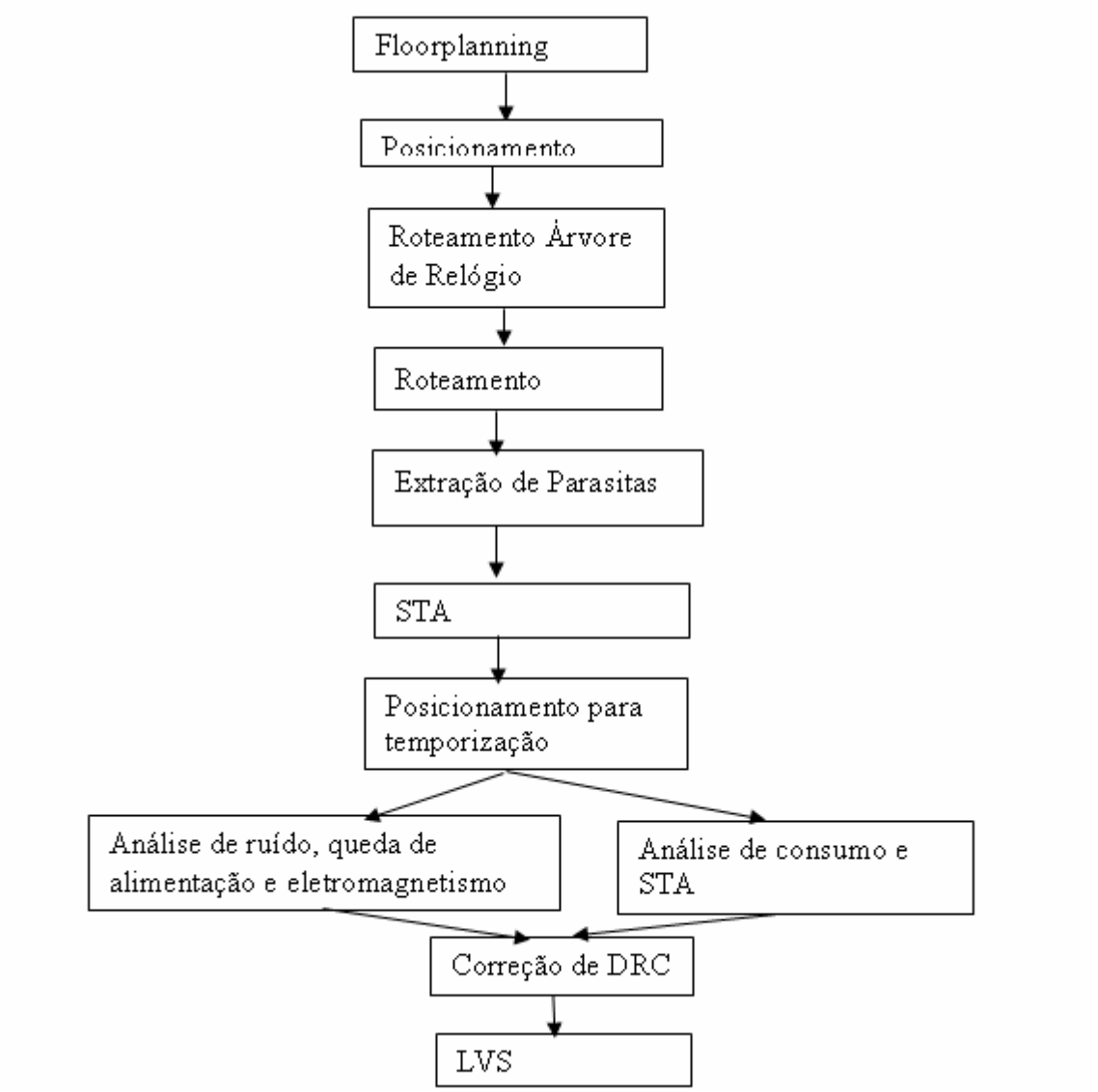

Figura 4.3: Fluxo de projeto síntese de layout

4.1.2.1Posicionamento Floorplanning e Roteamento

Os módulos podem ser colocados no chip de acordo com a sua necessidade de comunicação com os outros. Deseja-se que módulos que troquem informações estejam próximos de seus parceiros. Além disso, aqueles que têm conexões com o mundo exterior devem ficar próximos dos pinos de entrada e saída. O objetivo é fazer o posicionamento de maneira a diminuir o comprimento dos fios que conectarão as células.

A posição dos pinos de entrada e saída também precisa ser definida. É possível que estas localizações sejam definidas de acordo com a visão do sistema onde o circuito será usado. Neste caso, elas não devem ser alteradas.

As células são então posicionadas dentro dos módulos, nesta fase estima-se o comprimento total de fio necessário para as conexões entre as células e tenta-se

diminuí-la. Após, faz-se o roteamento, ou seja, realizam-se as conexões entre os elementos do projeto.

4.1.2.2Roteamento da árvore de relógio

Como os registradores do circuito ficam em locais diferentes, o sinal de relógio precisa percorrer caminhos distintos para chegar a cada elemento. Como conseqüência, ele não chega a todos os locais exatamente no mesmo instante de tempo. Este fenômeno é chamado de clock *skew* e pode ser problemático para o correto funcionamento do circuito já que pode causar erros de sincronização, especialmente quando a freqüência de relógio é alta.

Durante a etapa do roteamento da árvore de relógio, o objetivo é criar uma rede de distribuição do sinal que apresente o menor *skew* possível e que não sofra muitos problemas de degradação devido à propagação na linha. O sinal de relógio deve ter uma qualidade melhor que os demais sinais devido à sua função de sincronizar o circuito. Oscilações indesejadas deste sinal podem ser interpretadas pelos registradores como bordas e ativarem os elementos seqüências do projeto. Também é possível que o sinal tenha que percorrer um caminho muito longo e chegue ao *flip-flop* degradado de um jeito que ele não consiga ativar a carga do *flip-flop* no momento desejado.

O problema do *skew* pode ser minimizado tentando igualar as distâncias percorridas pelo relógio para chegar aos registradores. O problema de degradação de sinal é diminuído colocando *buffers* especiais, eles precisam possuir, na maioria das vezes, uma capacidade de fornecimento de corrente maior que os utilizados para lógica combinacional, para restaurar o sinal. No entanto, estes buffers possuem um atraso e, portanto, influenciam no *skew*. Além de ocuparem área e consumirem potência.

4.1.2.3Extração de Parasitas

Até agora, ou considerava-se que os fios não possuíam nem capacitância nem resistência ou era utilizado um modelo que tentava prever o comportamento físico dos fios. Como conseqüência os atrasos estimados não são muito precisos. Nesta fase, analisa-se o circuito com a intenção de se conseguir valores reais para esses parasitas. Existem três métodos para fazer a extração. O primeiro assume que os valores são constantes ao longo dos fios. O segundo utiliza tabelas para uma melhor aproximação. O terceiro e mais preciso resolve as equações diferenciais que determinam o comportamento dos fios. Como esperado, os métodos mais precisos necessitam de um tempo maior para executarem.

4.1.2.4Análise de temporização

Com todas as informações de posicionamento, roteamento e parasitas do circuito, é possível estimar mais precisamente o comportamento temporal do circuito. Como a tendência é que o desempenho seja deteriorado conforme o projeto passa pelas diversas etapas já que os modelos utilizados nos cálculos são cada vez mais precisos conforme o projeto avança no fluxo. Esta nova análise pode ser um novo STA ou até mesmo a simulação do circuito em nível de transistores. Embora a simulação forneça resultados que são mais precisos, o tempo de execução é bem superior ao do STA.

4.1.2.5Análise de Ruído, de Queda da alimentação e Eletromagnetismo

Linhas posicionadas próximas uma das outras podem ter problemas de ruído. Ou seja, parte do sinal pode ir de uma linha para outra através da capacitância existente entre eles. Esta interferência pode causar uma falha transiente, ou seja, que se extingue com o tempo, que pode comprometer o funcionamento do circuito.

Se o valor da alimentação diminuir durante o funcionamento do circuito, pode não ser possível fornecer a corrente elétrica necessária para o funcionamento do circuito. Novamente, existe uma falha transiente que o circuito deve suportar.

Além disso, há efeitos que eletromagnetismo que interferem no circuito. Variações de campos magnéticos podem gerar uma corrente elétrica indesejada que leve o circuito a operar de maneira errada.

É importante lembrar que todos estes problemas são aleatórios e, portanto, é necessário tentar estimar a força que eles terão e testar o circuito para os casos que são considerados os piores.

4.1.2.6Posicionamento dirigido à temporização

No primeiro posicionamento, não havia informações sobre parasitas. Por isso, não era possível estimar o atraso proveniente das conexões entre as células de maneira precisa, apenas através de modelos de fios. Se apenas o primeiro tipo de posicionamento for usado, é necessário pegar as informações de parasitas, fazer a análise de temporização e, caso necessário, fazer modificações no posicionamento original e refazer as análises. Em projetos grandes, pode acontecer que este ciclo tenha que ser repetido diversas vezes até que chegar-se em um momento quando novas melhorias no posicionamento acabam prejudicando uma otimização que havia sido feita anteriormente.

O posicionamento para temporização tem como estratégia priorizar as células que fazem parte de caminhos críticos. Elas são posicionadas primeiro de forma que o comprimento dos fios que as conectam seja o menor possível e, consequentemente, o atraso entre as células também será reduzido.

4.1.2.7Análise de Consumo

Pode-se optar por repetir a análise de consumo no fim do fluxo para aproveitar as informações sobre parasitas que agora estão disponíveis. Esta etapa ocorre da mesma forma que a primeira. Porém, como se tem informações mais detalhadas do circuito, o resultado é mais preciso.

4.1.2.8Correção das Regras de Desenho (DRC)

Nesta fase o circuito deveria estar pronto para ser produzido. No entanto, devido à imperfeições nas ferramentas, é necessário, na maioria dos casos uma correção no *layout* para que ele respeite as regras de desenho.

Para conseguir fazer um roteamento, a ferramenta pode, por exemplo, optar por não respeitar o espaçamento mínimo entre duas linhas de metal. Estes erros impossibilitariam a manufaturação do circuito, sendo necessário corrigi-los. Esta etapa

muitas vezes é realizada manualmente em um editor de *layout* como o Virtuoso da Cadence.

4.1.2.9Layout Versus Schematic

Nesta etapa do projeto, é feita a verificação que o *layout* que será fabricado desempenha a mesma função do esquemático. Se ambas as descrições forem equivalentes, então se pode considerar que o projeto esta pronto já que o *layout* é fabricável de acordo com as regras da tecnologia, cumpre os requisitos de consumo, área e potência e tem o desempenho funcional esperado.

O fluxo de projetado é cíclico. Quando não for obtido o resultado esperado em alguma etapa é necessário retornar para uma etapa anterior e melhorar o projeto. Além disso, outras etapas poderiam ser incluídas como a prototipação do projeto em um FPGA para fazer uma verificação funcional do circuito mais precisa, já que é possível observar o funcionamento real do circuito apesar de, provavelmente, em uma velocidade menor que a desejada para o produto final. Também pode haver uma etapa de simulação elétrica do *layout* que garante o funcionamento elétrico do projeto.

4.2 Utilização do fluxo no projeto

Este projeto não possui uma especificação formal como seria feito com um produto comercial. Apenas foi definido qual seria a função do circuito sem que outras características importantes como freqüência de operação, área e consumo fossem determinadas. O ambiente externo ao *chip* tem influência sobre ele durante a sua atuação. É possível especificar qual será a condição de operação do circuito, informando para a ferramenta a tensão de alimentação, a temperatura e uma característica relacionada com a variabilidade da tecnologia. Uma outra abordagem é caracterizar a biblioteca de células para diferentes condições de operação. Desta maneira, os caminhos longos do circuito devem utilizar as células da biblioteca caracterizada para a pior condição, os caminhos curtos para a melhor condição e os demais para a condição considerada típica. Como só há uma biblioteca disponível, o processo de síntese é feito para a condição com processo com variabilidade típica, temperatura de 25 graus Celsius e tensão de alimentação de 3,30V. Todos os mapeamentos realizados foram feitos usando como base o resultado da síntese feita sem nenhuma restrição ao circuito. O resultado deste primeiro mapeamento é resumido a seguir.

É importante destacar que das 128 interfaces que poderiam ser construídas, somente 36 foram incluídas no projeto final para economizar o tempo necessário pelas ferramentas. As conseqüências da inserção de outras interfaces são discutidas no capítulo cinco. Em todas as tabelas apresentadas neste e nos próximos capítulos, as grandezas padrões são: para área micrometro quadrado (um^2), para tempo nano segundos (ns), para consumo estático nano watts (mW) e para consumo dinâmico mili watts (mW).

Área (Células)	Timing	Consumo Estático	Consumo Dinâmico	Área (Fios)
2030859	10,447	119,057	69,27	1243546

Tabela 4.1: Resultado mapeamento sem restrição

Para estimar a freqüência de operação do circuito, o processo de síntese lógica foi realizado várias vezes. Em cada iteração, o valor do atraso desejado do circuito é determinado como sendo de 80% do atraso da iteração anterior. Como resultado a ferramenta tenta sempre aumentar a velocidade do circuito. No entanto, haverá um ponto a partir do qual não será possível aumentar a velocidade do circuito, este ponto é o que se deseja alcançar. A tabela 4.2 resume os resultados obtidos.

Rodada	Atraso	Sobra	Consumo Estático	Consumo Dinâmico	Área (Células)	Área (Fios)
1	7,938	0,420	119.673	86,48	2039089	1223498
2	6,341	0,010	109.416	79,02	2094602	1171446
3	5,073	0	117.603	84,91	2083086	1197568
4	4,058	0	124.809	80,41	2065106	1215046
5	3,246	0	146.875	89,67	2256812	1315204
6	3,235	-0,638	149.719	91,20	2247810	1306356
7	3,319	-0,731	148.232	85,64	2249642	1310009
8	3,211	-0,555	149.452	83,76	2245054	1306771
9	3,266	-0,697	150.621	93,17	2267348	1315504
10	3,363	-0,750	142.197	86,92	2221929	1304925

Tabela 4.2: Resultado mapeamento iterativo

A tabela mostra que a ferramenta consegue diminuir o atraso do circuito até perto de 3300 ps. Na rodada seis, o atraso é de 3235 ps e nas rodadas posteriores não é diminuído, portanto é possível estimar que este atraso esta perto do menor que a ferramenta consegue para o circuito. Este atraso possibilita o funcionamento do circuito em uma freqüência de até 310Mhz. Também é possível observar que o consumo estático é muito menor que o dinâmico, isto já era esperado por causa da tecnologia utilizada no projeto. Também é possível observar um aumento considerável do consumo quando comparado com o mapeamento inicial.

Outra observação é que o consumo dinâmico não é estimado de uma maneira muito precisa, já que não há informação sobre o relógio do circuito, uma análise de consumo do circuito da sexta rodada com freqüência de relógio igual a 300MHz mostra um consumo dinâmico de 696,424mW.

É interessante notar que durante as primeiras quatro rodadas a área do circuito permanece praticamente constantes, mas a velocidade dele diminui quase pela metade, na primeira rodada o atraso estimado é de 7938ps e na quarta de 4058ps. Na quinta rodada, há um acréscimo de quase 10% na área em relação à rodada anterior e a área é pouco alterada nas demais rodadas. A coluna denominada área de fios tem a estimativa da quantidade de linhas de metais que serão usadas no roteamento do circuito. Esta estimativa é importante para uma previsão mais precisa do atraso.

Para tentar fazer um fluxo mais parecido com o industrial, o processo de síntese foi repetido, mas com uma freqüência de operação determinada em 300MHz. Nas últimas três rodadas, a freqüência desejada foi aumentada para 312MHz, período do relógio igual a 3,2 ns. Também foi estipulado um atraso entre o pulso de relógio e a chegada das entradas e saídas do circuito de 166ps. O resultado deste mapeamento é mostrado na tabela 4.3 e a última versão do projeto, rodada cinco, será utilizada na próxima etapa do fluxo. A coluna atraso obtido é a soma do atraso do caminho combinacional com o tempo de *setup*. A coluna folga é a diferença entre o tempo desejado para a estabilização do sinal na entrada do *flip-flop* e o tempo quando isto acontece.

Rodada	Atraso	Sobra (300MHz)	Consumo Estático	Consumo Dinâmico	Área (Células)	Área (Fios)
0	3,469	-0,136	152.528	684,850	2418619	1424546
1	3,414	-0,081	144.977	672,50	2268828	1326800
2	3,359	-0,026	143.865	663,17	2254318	1332969
3	3,275	0,058	145.883	691,52	2240308	1315697
4	3,256	0,077	145.422	696,728	2234060	1312495
5	3,248	0,085	144.549	696,047	2230026	1312484

Tabela 4.3:Resultado mapeamento para 300MHz

O circuito mapeado foi comparado com a descrição VHDL e a ferramenta responsável pela verificação confirmou a equivalência das duas descrições. O projeto foi, então, submetido a uma análise de *timing*. A análise foi feita para diversas condições de operação com o objetivo de estudar o efeito delas no comportamento do circuito.

Condição de Operação	Atraso Obtido	Sobra (300MHz)	Sobra (250MHz)	Sobra (200MHz)	Sobra (150MHz)
Worst_2.60V_85	8,64	-5,31	-5,01	-3,65	-1,98
Worst_3.15V_70	7,44	-4,11	-3,81	-2,45	-0,78
Typ_3.30V_25	3,41	-0,08	0,59	1,59	3,25
Typ_3.00V_85	4,42	-1.09	-0,78	0,57	2,23
Best_3.45V_0	2,21	1,12	0,82	2,18	3,84
Best_3.60V_-40	1,93	1,40	1,097	2,46	4,12

Tabela 4.4: STA

Como o mapeamento foi realizado para a condição de operação *Typ_3.30V_25*, era esperado que o circuito não obtivesse bom desempenho para ambientes menos favoráveis. É possível observar que há uma folga razoável

O projeto passou então pela etapa de inserção de teste, substituição dos *flip-flops* pelos equivalentes *scan*, após esta etapa é esperado um aumento na área, no atraso e no consumo do circuito já que, essencialmente, esta etapa adiciona um multiplexador em

cada *flip-flop* e aumenta a quantidade de fios no circuito. Além disso, há a criação de novos pinos de entrada e saída. A tabela 4.5 mostra o resultado obtido após a inserção dos *flip-flops scan*. A análise de consumo é feita na tabela 4.6.

Atraso	Sobra (300MHz)	Sobra (250MHz)	Sobra (200MHz)	Sobra (150MHz)	Área (Células)	Área (Fios)
3,42	-0,11	0,58	1,58	3,24	2592476	1796950

Tabela 4.5: Resultado do mapeamento com scan

O aumento no atraso do circuito é desprezível. No entanto, a área sofreu um aumento de aproximadamente 15%. Observando os valores dos atrasos foi determinada a freqüência de operação desejada em 200MHz. Como as fases de posicionamento e roteamento adicionarão ainda mais atrasos no circuito é necessário que haja uma sobra significativa nesta etapa.

O circuito passou pela fase de análise de consumo. A estimativa é feita para as freqüências de 300MHz, 250MHz, 200MHz e 150MHz e para cada condição de operação. Como o consumo estático é muito menor do que o dinâmico, ele não é mostrado na tabela.

Condição operação	Consumo (300MHz)	Consumo (275MHz)	Consumo (250MHz)	Consumo (200MHz)	Consumo (150MHz)
Worst_2.60V_85	385,628	360,104	334,554	283,509	232,477
Worst_3.15V_70	641,040	597,801	554,519	468,047	381,596
Typ_3.00V_85	626,056	583,380	540,660	455,313	369,987
Typ_3.30V_25	762,850	710,579	658,255	553,721	449,211
Best_3.45V_0	932,707	867,734	802,695	672,757	542,851
Best_3.60V_-40	1001,444	931,671	861,827	722,290	582,787

Tabela 4.6: Análise de consumo

O consumo aumentou em torno de 10%, comparando o resultado para 300MHz e condição de operação *Typ_3.30V_25* com o resultado da rodada cinco da tabela 4.3. As condições de operação que apresentam um consumo maior são as mais rápidas e são, portanto, aquelas que terminam a tarefa antes. Por exemplo, o circuito operando a 150MHz necessita do dobro do tempo do que o circuito operando em 300MHz para realizar uma mesma tarefa. Portanto, o circuito operando a 300MHz gasta 385,628mJ para uma operação que custa 232,477mJ com freqüência de 150MHz.

O circuito respeita a restrição de temporização, então é possível avançar para a etapa de geração do *layout.* No entanto, prevendo o efeito que o posicionamento e o roteamento terão no circuito, a freqüência de operação desejada é reduzida. Caso fosse desejado que o circuito funcionasse em outras condições de operação seria necessário refazer o processo de síntese. A tabela 4.7 mostra o resultado da síntese com o menor atraso para as condições de operação *Worst_3.15V_70* e *Best_3.45V_0*.Os valores do consumo estático são omitidos, pois a ferramenta responsável pelo cálculo não conseguiu fazer a estimativa do valor da corrente de fuga.

Condição de operação	Atraso	Consumo Dinâmico	Área (Células)	Área (Fios)
Worst_3.15V_70	5,51	569,695	2388476	1397218
Best_3.45V_0	1,96	1048,06	2249301	1298355

Tabela 4.7: Mapeamento para outras condições de operação

A próxima etapa é o *floorplanning*, nesta etapa, as principais decisões a serem tomadas são: o tamanho do *chip*, quanto maior ele for mais caro será a produção, e o posicionamento dos módulos e pinos de entrada e saída. Inicialmente, pode parecer vantajoso determinar a área do *chip* como sendo igual a que foi obtida durante o mapeamento. No entanto, é necessária a existência de regiões sem células para possibilitar o roteamento dos sinais.

A geração da rede de alimentação pode ser feita de três maneiras. O método utilizado consiste em informar qual o consumo do circuito e a ferramenta gera as linhas de alimentação automaticamente. A segunda opção é definir de maneira manual a alimentação, isto é feito determinando as dimensões dos fios e espaçamento entre eles. Em ambos os casos, o resultado se assemelha com uma matriz. A terceira opção coloca anéis ao redor de cada bloco do projeto. Esta abordagem tem a vantagem de facilitar o projeto para circuitos que possuam múltiplos domínios de alimentação.

Foram feitos experimentos para as células ocupando 60%, 70% e 80% da área total do *chip*. Com utilização inicial de 60%, foi possível obter um circuito sem erros e que respeita as condições de temporização, apesar da sobra ser de apenas 0,287ns. Para 80% o sinal chegou 0,922 ns atrasado e ocorreram problemas de congestionamento de fios. Para 70% não ocorreram problemas, mas a sobra foi de -0,517ns.

Sobra	Área	Comprimento Total de Fio
0,287	4,3mm^2	5,198 um

5 ALTERAÇÕES NO PROJETO

O fluxo de projeto descrito anteriormente foi feito tendo como base uma árvore binária com altura quinze dividida em cinco níveis de profundidade três cada. No entanto, pode ser vantajoso utilizar outras estruturas para a implementação. Este capítulo traz algumas considerações sobre mudanças que podem ser feitas e as suas conseqüências.

5.1 Adição de interfaces de saída

O projeto da memória prevê a utilização de sete bits para determinar a interface de saída. No entanto, no projeto descrito anteriormente, 36 delas foram criadas. A adição de mais interfaces influência a área e o consumo do circuito. A velocidade do circuito é, a princípio, independente do número de interfaces de saída já que todas elas demoram aproximadamente o mesmo tempo para fazer a computação das suas entradas e trabalham em paralelo.

No entanto, há a resposta das interfaces para os módulos *NextNode* que é calculada através de um ou lógico entre o sinal de cada interface. Como cada módulo responde com um bit, seria necessária uma quantidade muito grande de interfaces de saída para que este caminho sofresse um aumento considerável no atraso. As áreas das interfaces criadas no projeto variam de aproximadamente 45000 um^2 até aproximadamente 50000 e a variação ocorre de forma não crescente. Ou seja, a interface x+1 não é, necessariamente, maior que a interface x. O consumo de cada uma varia entre 10mW e 11mW e, novamente, não é uma relação crescente.

5.2 Aumento da tabela de roteamento

A tabela utilizada anteriormente pode armazenar até 32767 rotas. Se for desejado adicionar mais entradas, a altura da árvore deve aumentar. Para isto, ou aumenta-se o número de estágios do *pipeline* ou muda-se o número de ciclos de relógio em cada nível.

Cada estágio do *pipeline* ocupa um pouco mais de área que o anterior já que precisa de mais bits para fazer o endereçamento da memória. O aumento da área pode ser estimado como sendo o tamanho do novo estágio individualmente mais a área necessária para colocar os multiplexadores.

Ao invés de aumentar a quantidade de estágios no *pipeline*, é possível aumentar a quantidade de níveis visitados por cada um. Esta abordagem é semelhante à anterior já que consiste em trocar um estágio por um outro que trabalhe com endereços maiores. O aumento da área é calculado como sendo a diferença entre a área dos níveis existentes e a área que será ocupa pelos novos módulos. Esta opção tem um lado negativo que é o aumento do tempo de espera para a transmissão de dois pacotes.

A tabela 5.1 mostra uma estimativa da área necessária para a adição de um novo estágio, considerando a configuração original de três níveis de árvore para cada estágio. Esta estimativa foi realizada fazendo a síntese individual de cada um dos módulos sem nenhuma restrição de temporização.

Estágio adicionado	Área células	Área Fios	Largura do endereço da memória (bits)
Sexto	46430	19881	18
Sétimo	46994	20197	21
Oitavo	47213	20368	24
Nono	47431	20538	27
Décimo	47650	20709	30
Décimo-Primeiro	52857	23171	33

Tabela 5.1: Estimativa de área pra acréscimo de estágio do *pipeline*

Como o IPv4 utiliza 32 bits para determinar o destino do pacote, não faz sentido ter mais de 2^32 entradas na tabela e o décimo-primeiro estágio indexaria até um trigésimo-terceiro bit de endereço. Os valores mostrados na tabela são simples estimativas, a tabela 5.2 mostra a área para cada estágio no mapeamento utilizado no fluxo.

Estágio	Área Células	Área Fios	Largura do endereço de memória (bits)
Primeiro	57296	17743	3
Segundo	54286	12414	6
Terceiro	56994	15851	9
Quarto	62320	18567	12
Quinto	53296	12148	15

Tabela 5.2: Área real de cada estágio

A diferença da estimativa para a área ocupada de verdade ocorre devido às restrições de temporização. Na estimativa, a síntese foi feita sem nenhuma preocupação com o tempo, porém no circuito afina havia requisitos que precisam ser obedecidos.

5.3 Migração para IPv6

Um dos objetivos descritos deste trabalho é a possibilidade de migrar facilmente para o IPv6. A estrutura modular permite que a descrição seja facilmente modificada para implementar este protocolo.

A principal diferença ocorre na quantidade de bits necessários por palavra de memória. A versão para o IPv4 necessita de 41 bits mais os bits para interface, que neste trabalho são sete. O IPv6 de 139 bits mais os sete bits para determinar interface, totalizando em 146 bits. Para as mesmas 32767 rotas, a tabela de roteamento terá tamanho de 4783982 bits, aproximadamente 600KB. A mesma quantidade de rotas é armazenada em, aproximadamente, 200MB com o IPv4.

Para o IPv6 pode ser uma boa idéia a utilização de uma árvore B para armazenar diversos endereços de destino em um mesmo nodo e aplicar um algoritmo de compreensão de dados para não ser necessário a utilização de 128 bits para cada endereço.

Para uma melhor comparação das diferenças entre o IPv4 e o IPv6 foi realizado o processo de síntese lógica para o IPv6 de maneira semelhante ao realizado no capítulo quatro. O primeiro mapeamento é feito sem restrição alguma e os demais são realizados e aumentando a restrição de tempo. A tabela 5.3 mostra os resultados obtidos.

Rodada	Atraso	Sobra	Consumo Estático	Consumo Dinâmico	Área (células)	Área (Fios)
0	27,99	--------	284.197	186,385	4782797	2273630
1	17,90	4,49	279.838	182,885	4827100	2198470
2	14,318	0,002	261.794	186,696	4978232	2096312
3	11,454	0	263.891	186,993	4992451	2102590
4	9,158	0,005	263,611	186,689	4994929	2106169
5	7,325	0,001	264,007	185,064	4996254	2104549
6	5,859	0,001	265,329	177,268	5006004	2112893
7	4,687	0	269.717	178,094	5028739	2135610
9	3,450	-300	317,658	199,338	5041742	2368204
10	3,458	-698	311,785	205,001	4984874	2357301

Tabela 5.3: Mapeamento para IPv6

O circuito para IPv6 consegue funcionar em uma velocidade próxima da do IPv4, isto era esperado já que, apesar dos endereços serem maiores, o processamento é essencialmente paralelo. No entanto, é possível perceber um aumento de mais de 100% na área do circuito. Além disso, para uma freqüência de 300MHz o consumo estimado é de 1,58W.

6 CONSIDERAÇÕES FINAIS

6.1 Resultados

O circuito projetado, para uma tecnologia de 0,35um, opera com uma freqüência de 200MHz, ocupa uma área de 4,3mm^2 e apresenta um consumo aproximado de 700mW. A tabela de roteamento armazena até 32767 rotas utilizando uma árvore de busca binária e ocupa 196602 bytes. São necessários seis bytes por rota, independentemente da tabela que será armazenada.

O tempo máximo necessário para a que um pacote seja encaminhado para a interface de saída é de 16 ciclos de relógio o que resulta em um total de 80ns de tempo de processamento por pacote. A estrutura de *pipeline* permite que o intervalo máximo que é necessário esperar para a transmissão de um segundo pacote seja de três ciclos de relógio ou 15ns. Portanto, espera-se que sejam transmitidos aproximadamente 66 milhões de pacotes por segundo. Se cada datagrama tiver 20 bytes, o valor mínimo para o IP, então a taxa de transmissão alcança 1,32GB/s.

Outra característica importante é a possibilidade do circuito transmitir os pacotes em uma ordem diferente da ordem de recepção. As soluções em software tendem a, naturalmente, respeitar esta ordem e nem as soluções em hardware costumam abordar esta possibilidade, mesmo que o protocolo IP permita que os datagramas sejam redirecionados em qualquer seqüência.

Além disso, o projeto pode ser alterado para trabalhar com tabelas de outros tamanhos e com outras configurações do *pipeline*. A migração para o IPv6 também é possível, porém com custos adicionais.

6.1.1 Comparação com outros trabalhos

Esta seção faz uma breve discussão entre este trabalho e alguns dos trabalhos estudados no capítulo dois. Como cada autor tem a liberdade para estruturar os seus testes e apresentar os resultados da maneira que achar melhor, nem sempre é possível uma comparação direta.

O trabalho de Pao et al precisa de uma quantidade diferente de memória de acordo com a tabela que é armazenada. Considerando os cinco exemplos fornecidos no texto, a média de utilização de memória é de 20,86 bytes por rota, o valor mínimo é de 4,25 bytes e o máximo de 32,13 bytes. Cada estágio do pipeline demora 12ns utilizando a tecnologia AMD C5N 0,5um.

Chang construiu a máquina de encaminhamento utilizando a tecnologia Compass 0,35um. O circuito consegue operar a 30MHz e só precisa de um acesso à memória para determinar a rota, dissipação de potência de 0,54W e utilização de 590 KB para armazenar uma tabela com aproximadamente 40.000 entradas, totalizando 14,75 bytes por rota, usando oito bits para a interface de saída. Se este trabalho utilizasse oito bits para armazenar a interface, necessitaria de 245KB para guardar 40000 rotas.

Sangireddy atingiu um melhor caso de 229,3 milhões de procuras por segundo, com um tempo de busca na tabela de 4,36ns. Ele não informa sobre o total de memória utilizado para armazenar a tabela.

Sun e Zhao testam várias configurações no mesmo texto e apresentam resultados para tabelas diferentes para cada alternativa de algoritmo. A tabela influência no total de memória utilizada. No modo que minimiza o espaço de armazenamento, são necessários de 2,60 bytes a 2,93 bytes por rota e seis acessos à memória. O modo com o menor número de acessos realiza três leituras da memória, a média de byte por rota é de 15,67 bytes, com os valores variando entre 11,5 até 21,6. É importante lembrar que este método necessita que todas as rotas sejam conhecidas quando a tabela for montada para que a compressão dos dados seja feita de maneira eficiente.

Trabalho	Média Bytes/Rota	Mínimo Bytes/Rota	Máximo Bytes/Rota	Acessos à memória	Pacotes/s
Este	6	6	6	15/3	666M
Pao	20,86	4,25	32,13	5	833M
Chang	14,75	----------	----------	1	30M
Sangireddy	--------	----------	----------	----------	229,3M
Sun e Zhao (1)	2,84	2,60	2,93	6	----------
Sun e Zhao(2)	16,73	11,5	21,6	3	----------

6.2 Conclusões e trabalhos futuros

Este trabalho apresentou o desenvolvimento de uma máquina de encaminhamento para pacotes IP em hardware. O circuito recebe o endereço de destino do datagrama, faz a busca na tabela e decide para qual das interfaces de saída redireciona-lo.

Apesar do fluxo ASIC não ter sido completo, o trabalho atingiu os principais objetivos:

- O projeto foi realizado de forma a ser facilmente modificável para outras configurações. Ou seja, o tamanho da tabela, o número de interfaces de saídas e até mesmo a escolhe entre qual tipo de pacote IP transmitir são simples de serem alterados.
- As etapas do fluxo são estudadas e aquelas realizadas foram feitas de maneira a obter bons resultados e, quando adequado, comparando diferentes possibilidades.

- Todas as etapas do fluxo, mesmo as não realizadas, foram estudadas de maneira a permitir o entendimento da importância de cada uma, do seu objetivo e de como elas se relacionam.

Como trabalhos futuros são sugeridos:

- A criação dos circuitos para inserção e remoção de nodos na tabela, levando em consideração a necessidade de balancear a árvore.
- A finalização do fluxo e a elaboração. Se as etapas restantes forem completadas, pode ser possível mandar o circuito para fabricação como um protótipo já que a tecnologia utilizada é comercial.
- A criação dos circuitos que completam o diagrama do roteador apresentado no capítulo três, incluindo a geração de memória interna ao chip. Após, fazer a integração de todos os módulos em um único chip.
- A migração do projeto para uma tecnologia mais recente para aumentar a velocidade de transmissão.

.

REFERÊNCIAS

WESTE, Neil H.E.; HARRIS, D. **CMOS VLSI DESIGN: A CIRCUITS AND SYSTEMS PERSPECTIVE** . 3rd ed. Boston: Addison-Wesley, 2005.

COMER, DOUGLAS E; STEVENS, DAVID L. **INTERLIGAÇÃO DE REDES COM TCP/IP VOLUME II: PROJETO, IMPLEMENTAÇÃO E DETALHES INTERNOS.** 3. ed. Rio de Janeiro: Campus, 1998**.**

COMER, DOUGLAS E. **INTERLIGAÇÂO DE REDES COM TCP/IP VOLUME I: PRINCÌPIOS, PROTOCOLOS E ARQUITETURA.** 8. ed.Rio de Janeiro : Campus, 1998.

CHU, YUAN-SUN et al. ASIC Design of Fast IP Lookup for Next Generation IP Router, IEEE, 2005.

KOMPELLA, RAMANA RAO; VARGHESSE, GEORGE. Reduced State Fair Queuing for Edge and Core Router, ACM, 2004

DEMERS, ALAN; SHENKER, SCOTT. Analysis and Simulation of a Fair Queuing Algorithm. ACM, 1989.

GUPTA, PANKAJ; LIN, STEVEN; MCKEOWN, NICK. Routing Lookups in Hardware at Memory Access Speeds.

SHREEDHAR, M. VARGHESE, GEORGE. Efficiente Fair Queuing Using Deficit Round Robin. **IEE/ACM TRANSACTIONS ON NETWORKING,** VOL 4. NO 3. JUNE 1996.

PAO, D. et al. Efficient Hardware Architecture for Fast IP Address Lookup. **IEE Proc.- Comput. Digit. Tech.** VOL 150. NO 1. January 2003.

CHANG, ROBERT C; LIM, BENG-HUAT. Efficient IP Routing Table VLSI Design for Multigigabit Router. **IEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I : REGULAR PAPERS**. VOL 51. NO 4. April 2004.

SANGIREDDY, RAMA. SOMANI, ARUN K. High-Speed IP Routing With Binary Decision Diagrams Based Hardware Address Lookup Engine. **IEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS.** VOL 21. NO 4. May 2003.

SUN, XUEHONG. ZHAO, YIQIANG Q. An On-Chip IP Address Lookup Algorithm**. IEEE TRANSACTIONS ON COMPUTERS**. VOL 54. NO 7. JULY 2005.

KESHAV, S; SHARMA, ROSEN. Issues and Trends in Router Design. **IEEE COMUNICATIONS MAGAZINE,** May 1998.

MÔCHO, R.U.R.. **Circuitos assíncronos na plataforma FPGA**. 2006. 132 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ATMEL, **Asic Design Rules**, disponível em: <http://www.atmel.com/dyn/resources /prod_documents/doc1205.pdf>, acesso em nov, 2009.