unesp Departamento de Engenharia Elétrica FEIS

“TAB2VHDL: UM AMBIENTE DE SÍNTESE LÓGICA PARA MÁQUINAS DE ESTADOS FINITOS”

LEANDRO DE OLIVEIRA TANCREDO

UNIVERSIDADE ESTADUAL PAULISTA – UNESP
FACULDADE DE ENGENHARIA
CAMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DISSERTAÇÃO DE MESTRADO

"TAB2VHDL: UM AMBIENTE DE SÍNTESE LÓGICA PARA MÁQUINAS DE ESTADOS FINITOS"

19 DE SETEMBRO DE 2002

UNIVERSIDADE ESTADUAL PAULISTA – UNESP
FACULDADE DE ENGENHARIA
CAMPUS DE ILHA SOLTEIRA

“TAB2VHDL: UM AMBIENTE DE SÍNTESE LÓGICA PARA MÁQUINAS DE ESTADOS FINITOS”

_____

Prof. Dr. Carlos Magnus Carlson Filho

Banca

_____

Prof. Dr. Norian Marranghello

Banca

_____

Prof. Dr. Alexandre César Rodrigues da Silva

Orientador

Dissertação de Mestrado submetida ao DEE – FEIS – UNESP como exigência para obtenção do título de Mestre em Engenharia Elétrica.

ILHA SOLTEIRA – SP

AGRADECIMENTOS

Dedico esta dissertação aos meus amigos e familiares, cada um a seu tempo e a seu modo...

Aos meus verdadeiros orientadores da vida, meu pai João e minha mãe Eliana;

Á minha amada, persistente, carinhosa, sensível e companheira Cristiane;

Aos meus irmãos batalhadores Adriano e Liamara, minha cunhada Sandra e minha afilhada Nathália;

Ao meu orientador que abriu os olhos para outro mundo, e também a sua família, que foi essencial para a existência desse trabalho;

Meus amigos professores estimuladores Clinton, Silvio, Elizandra, Cláudia Hidalgo;

Aos coordenadores responsáveis, sérios e porém acolhedores Maurício e Carlos;

Á coordenadora pedagógica da FAIJALES dona Ruth correta, competente e afetuosa;

Meus avôs amorosos, Amélia, Vick, Marcos e Sebastião (falecido);

Meus tios sempre presentes e amorosos Mariza ,Paschoal e Antônia;

Minhas primas seus maridos e filhos Cláudia, Isamara e Família;

Meus amigos dedicados, acolhedores e acima de tudo profissionais Tereza, Getúlio, , Neide, Cidinha Botaro, Cidinha (Barretos), Marivaldo, João Marcelo, João Roberto, Edward, Leônidas e Kleber ;

Aos amigos do DEE FEIS Fábio, Helder, Marcos e Tércio;

Aos professores rigorosos e persisitentes Nobuo, Aparecido, Edvaldo e Kitano que contribuíram para esse trabalho.

A minha melhor gratidão é trazer um pouco de cada um deles dentro de mim e levar os frutos de suas sementes para a vida e para o resto de minha vida, se por vezes pareci distante, não estava.

Fiz, faço e farei por vocês.

Tento retribuir a força que vocês me concedem.

[1] Ver glossário na página 78.

Frases

P O R T A S

Se você encontrar uma porta a sua frente, pode abri-la ou não.
Se você abrir a porta, pode ou não entrar em uma nova sala.
Para entrar, você vai ter de vencer a dúvida, o titubeio ou o medo.
Se você venceu, dá um grande passo: nesta sala, vive-se.
Mas tem um preço: inúmeras outras portas que você descobre.

O Grande Segredo é Saber:
"Quando e qual porta deve ser aberta"

A Vida não é Rigorosa: ela propicia erros e acertos.
Os erros podem ser transformados em acertos quando com eles se aprende.

Não existe a segurança do "Acerto Eterno".

Içami Tiba

[1] Ver glossário na página 78.

RESUMO

Este trabalho apresenta uma nova ferramenta de síntese para projetos de sistemas digitais denominada TAB2VHDL. A partir da descrição em diagrama de transição de estados de uma máquina finita, representada no modelo de Mealy, é gerada uma descrição otimizada do sistema na linguagem de VHDL. Elimina-se dessa forma a tarefa árdua com detalhes de projeto.

A TAB2VHDL foi comparada com duas outras ferramentas disponíveis comercialmente. Foram projetados diversos chip-set de códigos de transmissão digital utilizados no setor de telecomunicações. Os resultados comprovaram o desempenho satisfatório com relação ao custo de implementação, ao tempo de execução e uso de memória.

Palavra Chave: FPGA, VHDL, FSM, HDL, Síntese Lógica

ABSTRACT

This paper presents a new synthesis tool for digital system projects called TAB2VHDL. From the description in states transition diagram of a finite machine, represented in Mealy's model, an optimized system description in VHDL language is generated. Therefore, it is eliminated an arduous task with project details.

The TAB2VHDL was compared with two other available commercial tools. It was projected a sort of chip-set digital transmission codes, used in telecommunication sector. The results proved the satisfactory performance related to the implementation cost, to the time of execution and memory use.

Keywords: FPGA, FSM, VHDL, HDL, Logic Synthesis

[1] Ver glossário na página 78.

LISTA DE ABREVIATURAS

2B1Q	2 *Binary/ 1 Quaternary*
3B4B	3 *Binary/2 Binary*
ADSL	*Asymmetric Digital Subscriber Line*
AHDL	*Altera Hardware Description Language*
AMI	*Alternate Mark Inversion*
ASIC	*Aplication-Specific Integrated Circuits*
CAD	*Computer-Aided Design*
CAP	*Carrierless Amplitude/Phase Modulation*
CCS	*Calculus of Communicating System*
CCITT	*Comité Consultatif International Telegraphique et Telephonique*
CI	Circuito Integrado
CPLD	*Complex Programmable Logic Devices*
CSP	*Communicating Sequencial Process*
DE	*Discrete Event*
EPLD	EPROM *Programmable Gate Array*
EPROM	*Erase Programmable Ready Only Memory*
ERB	Estação Rádio-Base
FDDI	*Fiber Distributed Data Interface*
FPGA	*Field Programmable Gate Array*
FSM	*Finite State Machine*
HDB3	*High Density Bipolar of order 3 code*
HDL	*Hardware Description Language*
HDLC	*High Level Data Link Control*

[1] Ver glossário na página 78.

IEEE	*Institute of Electrical and Electronic Engineers*
HDL	*Hardware Design Language*
HDSL	*High bit rate Digital Subscriber Line*
ISO	*International Standards Organization*
ITU	*International Telecommunication Union*
LAPB	*Balanced Link Acess Procedure*
LAPD	*Link Acess Procedure for the D channel*
MIPS	Milhões de Instruções por Segundo
MLT-3	*Multi-Level Transmission-3*
MOC	*Model of Computation*
OSI	*Open System Interconnection*
PCM	*Pulse Code Module*
RDSI	Rede Digital de Serviços Integrados
RTL	*Register Transfer Level*
SDL	*Specification and Description Language*
SEQ2VHDL	*Sequential To* VHDL
VADSL	*Very High Bit-rate Asymmetric Digital Subscriber Line*
VDSL	*Very Data Digital Subscriber Line*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

[1] Ver glossário na página 78.

INDICE GERAL

[1] Ver glossário na página 78.

[1] Ver glossário na página 78.

ÍNDICE DE FIGURAS

[1] Ver glossário na página 78.

[1] Ver glossário na página 78.

INDICE DE TABELAS

[1] Ver glossário na página 78.

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo descreve-se o desenvolvimento do setor de telecomunicações, referenciando cenários antigos, atuais e futuros, para em seguida, apresentar-se às tendências de serviços mundiais de telecomunicações, e as novas tecnologias emergentes, que utilizam como meio condutor o par de fios metálicos, meio físico existente onde ocorre a maioria das transmissões de dados locais das empresas concessionárias de telefonia, condutor que é um dos responsáveis pelas novas Redes Digitais Integradas (RDI), que permitem altas taxas de velocidade de transmissão digital.

Essas redes apresentam como um dos seus componentes, códigos de linha, que são circuitos integrados que serão designados como estudo de casos neste trabalho, vários códigos de linha foram projetados e implementados em componentes FPGA[1] (do inglês *Field Programmable Gate Array*) da ALTERA. Utilizou-se de dois ambientes comerciais, cujos resultados serviram para avaliar a ferramenta desenvolvida.

Apresentam-se também as possíveis formas de representação dos circuitos digitais, que contém seus domínios e seus níveis de abstração, a partir do diagrama Y de Gajski & Kahn [GAJ 83], muito importante para que se entendam, quais são as possíveis formas de descrição de um chip.

---

[1] Ver glossário na página 78.

Logo em seguida, temos as ferramentas para modelagem, que abrange sua evolução e as principais linguagens formais de descrição de sistemas digitais ou HDLs (abreviatura do inglês *Hardware Description Languages*) comerciais como ALTERA e XILINX.

Finalmente, apresenta-se a proposta de pesquisa, contendo o objetivo, a organização da tese, e suas contribuições.

1.1 CENÁRIO DO SETOR DE TELECOMUNICAÇÕES

A telecomunicação nasceu da necessidade de comunicação entre duas pessoas, levando suas vozes a distâncias cada vez maiores [TEL 94].

No início da implantação dos sistemas telefônicos a interligação entre os assinantes, era feita pela telefonista, tornou-se lento por se tratar de um sistema manual, com o passar do tempo as centrais manuais foram substituídas por centrais analógicas de comutação automática, e mais tarde por centrais digitais.

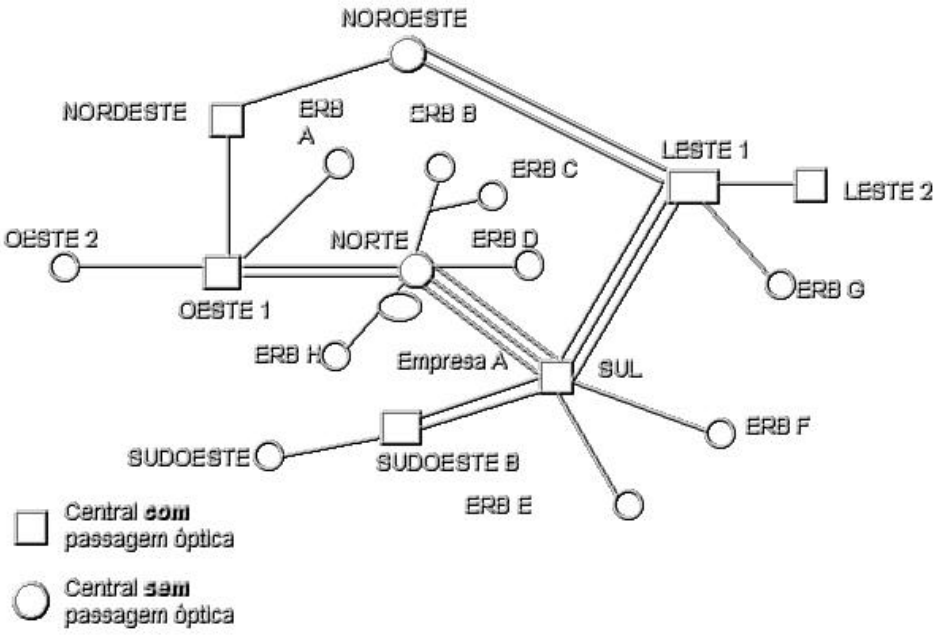

Figura 1 - Representação da Cidade A com Centrais Telefônicas e ERBs.

Com o passar do tempo, o aumento dos usuários tornou necessária a existência de várias centrais, as quais seriam, ligadas por fibras óticas ou através de rádios, como exemplifica a Figura 1. Apesar do emprego de novos meios de transmissão, durante todos esses anos, o meio de transmissão presente na maioria das interligações entre os usuários finais e as centrais, é o par trançado.

Além dos aumentos de centrais, outro fator importante, no setor de telecomunicações, na última década, foi a massificação e um crescimento cada vez maior de sua gama de serviços que podem ser classificados como [UEH 97]:

a) Telecomunicação Multimídia apresenta como característica explorar todas as formas de comunicação e tratamento de imagens, elegendo a visão, som e movimento como um dos principais órgãos sensoriais para fins de comunicação;

b) Telecomunicações Inteligentes são os serviços que agregam acessos a banco de dados com múltiplas facilidades, rapidez e flexibilidade;

c) Telecomunicações Pessoais é quando o ser humano precisa movimentar-se e a necessidade de se comunicar deverá adaptar-se ao seu próprio deslocamento.

Outro fator relevante foi a inversão no movimento do homem em direção à busca da informação, ou seja, não nos deslocamos mais fisicamente até a informação. Agora é a informação que se movimenta em nossa direção. O passo inicial surgiu com a criação da Internet [TAS 99].

Tais mudanças comportamentais e evoluções das centrais, só foram possíveis, graças à digitalização dos serviços de telecomunicações existentes e ao próprio progresso da eletrônica.

Contudo, uma das características relevantes pelo aumento de velocidade foi a transformação ocorrida nas três primeiras camadas do modelo OSI/ISO (abreviatura do inglês *Open System Interconnection / International Standards Organization*) a camada Física (Nível 1), Enlace (Nível 2) e Rede (Nível 3), que são as principais camadas de referência utilizadas em uma rede pública [MAR 01].

Nos anos 70, os níveis 2 e 3 eram todos implementados em *software*, somente a primeira camada ou a camada física era composta de *hardware*. A partir do final de 70 e

início dos anos 80, surgem padrões para os sistemas de comunicação em *hardware*. O protocolo responsável pela implementação foi o HDLC (abreviatura do inglês *High Level Data Link Control*) publicado pela ISO. Conseqüentemente surgiram implementações de diversos *firmwares* oferecidos como *chip-set* como LAPB (abreviatura do inglês *Balanced Link Acess Procedure*) e LAPD (abreviatura do inglês *Link Acess Procedure for the D channel*).

Assim, no final dos anos 80 e início dos anos 90, os membros do corpo de padronização reconhecem a existência de redundância entre a camada de enlace e a camada de rede, tais aplicações como teste de erro e controle de fluxo, com isso houve a migração das camadas de enlace e rede para o *hardware*, possibilitando que os sistemas operem de uma forma mais rápida.

Dessa forma, unindo a infra-estrutura atualmente utilizada para os serviços de telefonia, ou seja, o par trançado com inovações tecnológicas de *hardware*, uma grande quantidade de serviços digitais de telecomunicações, estão disponíveis e novas aplicações surgirão com o tempo [BOR 99].

Atualmente, com o advento da fabricação de microcomputadores em grande escala, impulsionou-se a implementação de novas tecnologias, empregados em diversos segmentos de rede como nas Redes Locais ou LAN (abreviatura do inglês *Local Area Network*), ou nas Redes Privadas ou WAN (abreviatura do Inglês *Wide Area Network*) e nas Redes Comerciais ou Redes de Valores Agregados ou VAN (abreviatura do inglês *Value-Added Networks*), todas contendo como parte da sua placa de rede ou modem, um circuito integrado responsável pelo código de linha.

Com relação às VANs, apresenta-se a seguir uma breve descrição das tecnologias atuais de digitalização:

a. RDSI – Rede Digital de Serviços Integrados

Rede de telefonia evoluída para rede digital integrada, que proporciona conectividade digital fim-a-fim, suporta uma variedade de serviços vocais e de dados, aos quais os usuários têm acesso através de um conjunto limitado de conexões usuário-rede. Uma das padronizações, ou seja, a interface S possui acesso básico no cliente caracterizado

por utilizar uma taxa de transmissão de 192 Kbps e o código de linha AMI (abreviatura do inglês *Alternate Mark Inversion*).

Outro exemplo está na Interface U, ou seja, um acesso primário de 2048 Kbps, segmentado em 30 canais de dados ou voz a 64 Kbps, acrescido de um canal de sinalização de 64 Kbps. Com relação ao código foi inicialmente empregada a técnica de transmissão PCM (do inglês *Pulse Code Modulation*) que utiliza o código de linha HDB3 (do inglês *High Density Bipolar of order 3 code*). Atualmente é possível a substituição pela técnica HDSL (abreviatura do inglês *High bit rate Digital Subscriber Line)*.

b. HDSL – Linha do Assinante Digital com Alta taxa de transmissão

Transmissão de dados digitais em redes para implementação de acessos de 2 Mbps na rede existente, utilizando 1, 2 ou 3 pares metálicos com velocidade de transmissão de 2336 Kbps, 1168 kbps ou 784 Kbps, formado por um *chip-set* com técnicas de codificação de linha 2B1Q (do inglês *2 Binary/ 1 Quaternary*).

Descreve-se no capítulo 4 o comportamento dos códigos de linhas, que foram implementados em FPGA para avaliar a ferramenta desenvolvida.

1.2 CONCEITOS DE REPRESENTAÇÃO DE UM PROJETO DIGITAL

Nesta sociedade tecnológica a qual vive-se, é fácil arruinar-se em detalhes quando se representa um projeto de circuito eletrônico, pela complexidade enfrentada. Para que se possa dominar esta complexidade, deve-se limitar a quantidade de informação analisada, em um determinado momento, "trabalhe em níveis de árvores ao invés de florestas" [ARM 89]. Essa frase é análoga a situação enfrentada por projetistas, devido a complexidade dos *chips*, por exemplo, no mercado mundial tem-se máquinas como o Pentium 4, que pode trabalhar com até 42 milhões de componentees, com uma freqüência de relógio máxima de 2.530 MHZ, com microprocessadores de 32 bits e executam instruções a 2.530 MIPS (Milhões de Instruções por Segundo) [INT 02a].

Durante a década de 60 os circuitos integrados, não ultrapassavam a ordem das centenas de componentees, trabalhava-se com freqüência de relógio não superior a 1 MHZ, alcançando uma taxa máxima de execução de instruções na ordem de frações de MIPS

[DeM 94]. Os projetos eram elaborados a partir de papel e lápis [MAR 95] o que passou a ser o fator limitante do projeto na época.

A necessidade de criar novas ferramentas, e as evoluções tecnológicas descritas acima, dobram o número de portas que de dois em dois anos, fez surgir, editores de *"layouts"*, e algumas outras ferramentas de desenho, todos sujeitos a erros [CAL 98].

Nas décadas de 70 e 80, com o aumento da complexidade cada vez maior, tinha-se que garantir a funcionalidade do sistema, pois o custo do projeto era considerado elevado, proliferaram programas de auxílio ao projeto de sistemas digitais e ferramentas de síntese.

No cenário atual, KHOSLA et alli [KHO 01] descreve que para sistemas elétricos existe uma grande variedade de linguagens e simuladores. Este último é muito utilizado para avaliar o comportamento dos circuitos analógicos, sendo o SPICE o mais popular, enquanto que para os circuitos digitais as linguagens VHDL e a Verilog HDL são as mais utilizadas.

Na atualidade para se implementar um circuto digital, deve-se respeitar as etapas de um projeto que consiste essencialmente de dois subconjuntos que se agregam formando a Descrição Inicial (*front-end*) e a Descrição Final (*back-end*), conforme o fluxograma representado na Figura 2 [ROC 99].

A partir da Descrição Inicial especifica-se o projeto, buscando uma implementação do circuito, escolhe-se uma linguagem de descrição de *hardware*, e logo após simula-se as dependências tecnológicas. Caso ocorra algum erro, descreve-se novamente o circuito utilizando a linguagem de descrição até que o circuito esteja validado. Assim, parte-se para a síntese lógica que é novamente analisada por uma simulação lógica e temporal, caso não corresponda ao esperado, é corrigida novamente através da mesma linguagem de descrição, ou refeita a síntese, até que a simulação lógica e temporal valide o circuito. Dessa forma, insere-se a estrutura de vetor de teste, logo em seguida, a geração de lista de ligações e a geração dos vetores de teste. Posteriormente, passa-se para a fase de descrição final, que não será objeto do nosso trabalho, pois o sistema será implementado em estrutura FPGA; desse modo o circuito não passará pelo processo de fabricação.

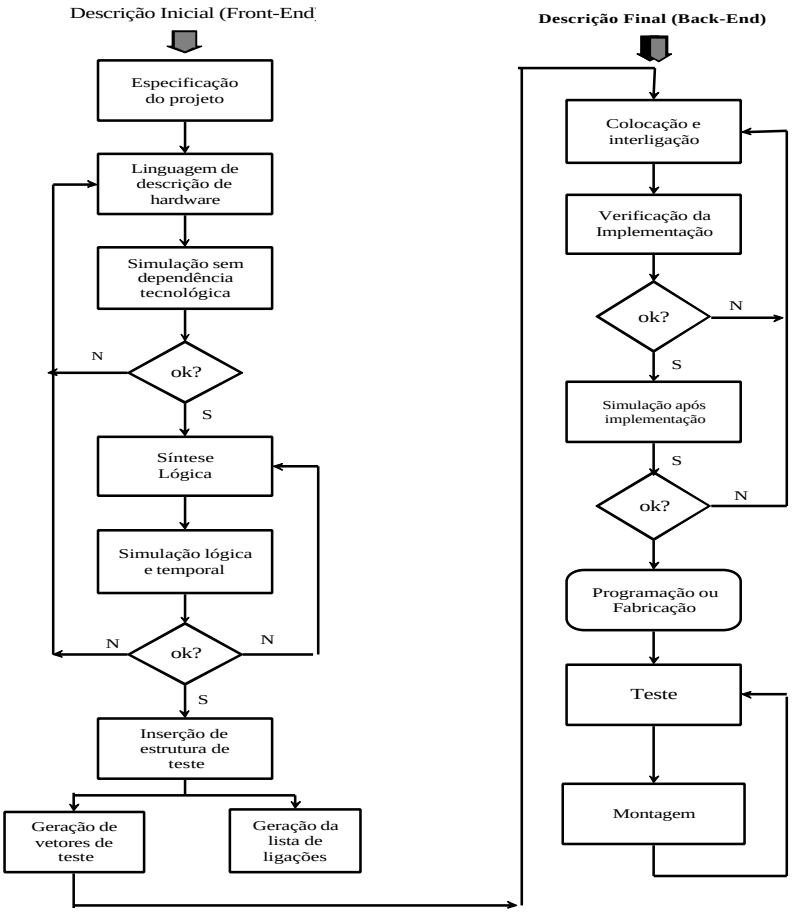

Figura 2 - Fluxo de um processo de projeto de circuito integrado

Conseqüentemente, nesta seção, conforme [CAL 98] [MAR 95], além das etapas do projeto, descreve-se formas de representações voltadas para sistemas digitais, freqüentemente classificados por seus níveis de abstração e decomposto de forma hierárquica, baseado na quantidade de informação contida na descrição associada a cada passo.

Assim, em descrições de mais alto nível, a sintaxe é próxima da linguagem humana, possuindo poucas informações associadas e deve, se possível, ser a descrição inicial, enquanto que a de mais baixo nível, está próxima a linguagem de máquina, localizado na descrição final, ou seja, no ponto de origem do diagrama Y [GAJ 83], apresentado na Figura 3.

Pela figura 3 pode-se notar que um sistema digital pode ser classificado com três [DeM 94] quatro [CAL 98] ou cinco níveis de abstrações distintas [MAR 95] e ordenado de forma ascendente:

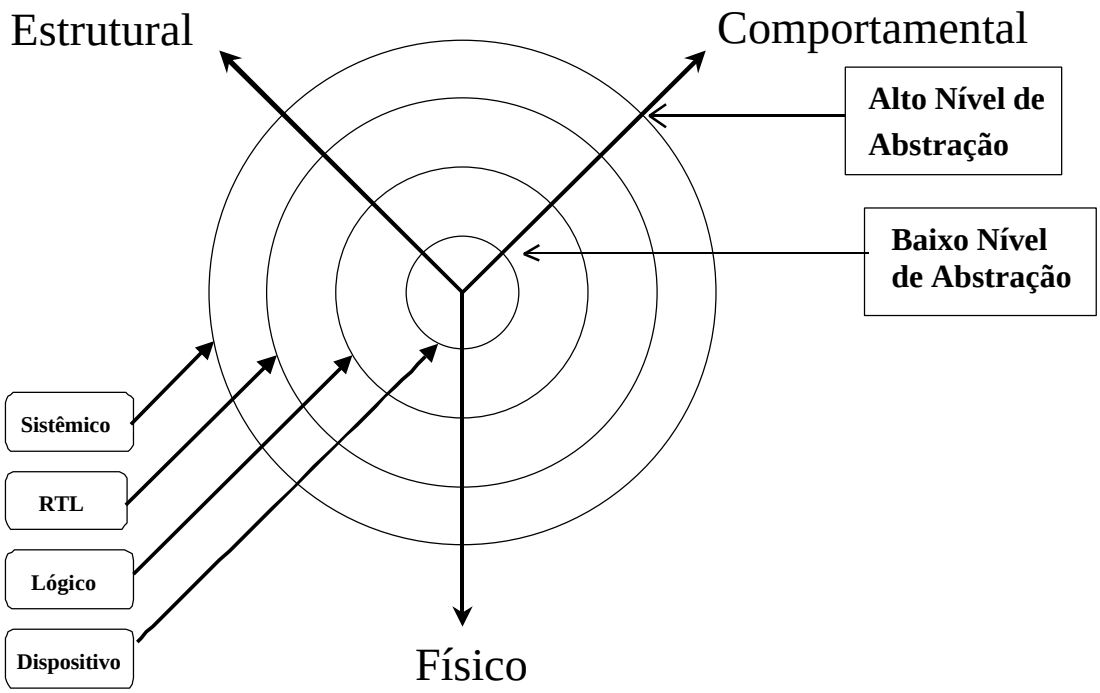

Figura 3 - Níveis de Abstração do Diagrama Y

- Nível Dispositivo - Os modelos dos componentes provêm da teoria de circuitos elétricos e eletrônicos, e/ou da física de semicondutores, isto é componentees, resistores, capacitores, equações diferenciais, diagramas elétricos, etc;

- Nível Lógico - O sistema digital é descrito a partir de noções elementares da teoria de circuitos digitais: portas lógicas, *flip-flops*, equações booleanas, etc;

- Nível RTL (abreviação do inglês *Register Transfer Level*) - Os modelos de base são os componentes mais complexos da teoria de circuitos digitais e as formas de descrevê-los são: registradores, decodificadores, ULAs, multiplexadores, linguagens de transferência entre registradores, grafos de fluxo de dados e de controle, etc;

- Nível Sistêmico - O sistema digital é descrito como um conjunto de algoritmos e módulos capazes de executar aqueles que podem ou não ter um mapeamento direto para um

hardware existente: processadores, memórias, componentes abstratos implementados no *software* básico do sistema, tais como um pacote de ponto flutuante em um sistema sem coprocessador aritmético, ou monitores de sistemas operacionais;

Os domínios podem ser classificados como:

- Domínio Comportamental, o qual mostra diversos formalismos para a representação da funcionalidade do sistema, destacando-se pela utilização de linguagens de descrição de *hardware*. O diagrama Y apresentado na Figura 4, destaca o domínio comportamental.

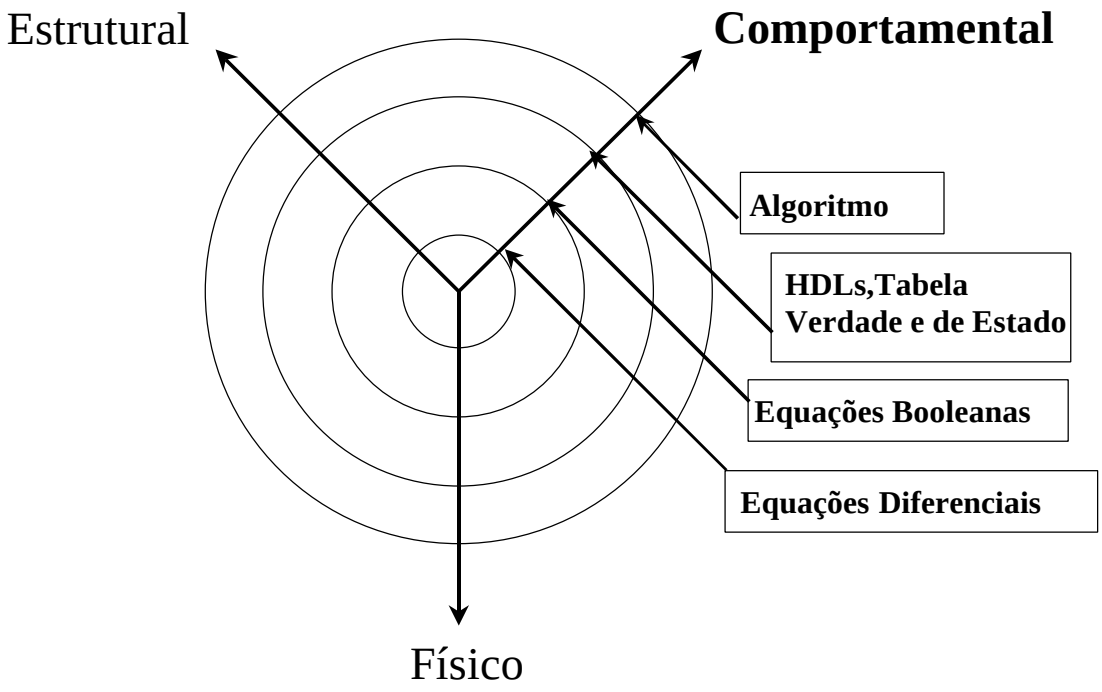

Figura 4 - Diagrama Y - Domínio Comportamental

- Domínio Estrutural ou Funcional contém informações sobre conjuntos diferentes de primitivas, ou seja, como deve-se interconectar blocos de base de comportamento conhecidos, sem preocupar-se com a disposição física destes no sistema.

- Domínio Físico contém informações geométricas a respeito dos componentes e módulos e/ou da disposição espacial no sistema a ser fabricado.

A Figura 5 apresenta o diagrama Y destacando os Domínios Estrutural e Físico.

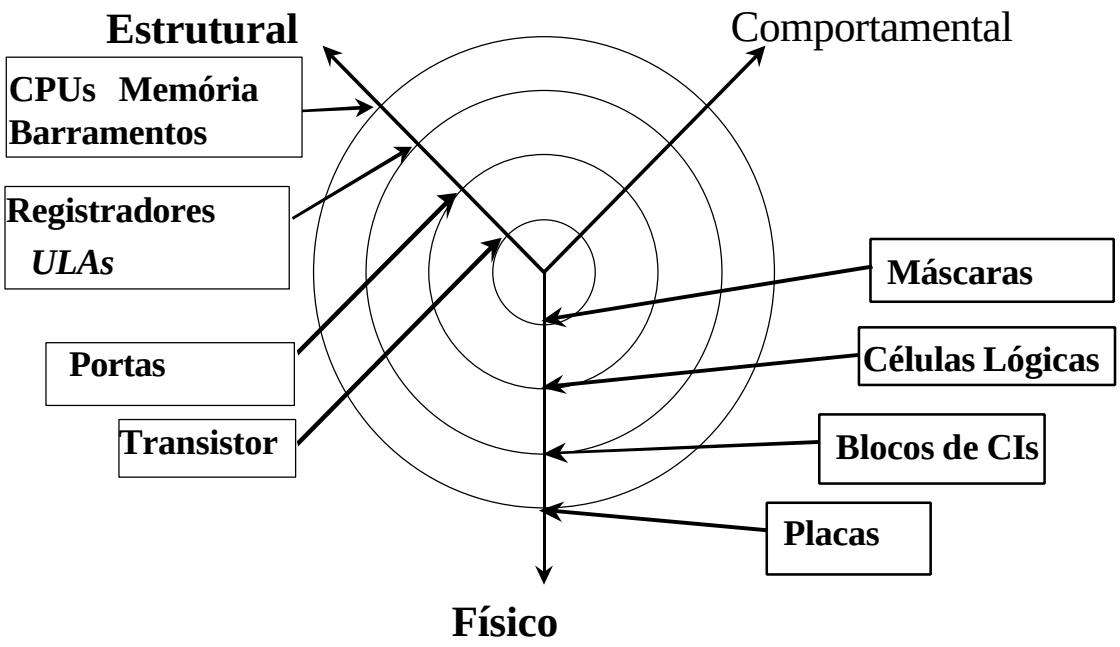

Figura 5 - Diagrama Y - Domínio Estrutural e Físico

A ferramenta de síntese desenvolvida opera no domínio comportamental, no nível RTL visando uma implementação otimizada com componentes programáveis.

1.2.1.1 CIRCUITOS PROGRAMÁVEIS

O conhecimento e o domínio da tecnologia de circuitos programáveis torna-se assim uma importante estratégia para o projetista de sistemas eletrônicos. Este item propõe uma visão geral das arquiteturas comerciais utilizadas na atualidade.

Os Dispositivos Lógicos Programáveis (do inglês *Programmable Logic Array*) são dispositivos digitais configuráveis, usados para implementar um conjunto de funções lógicas seqüenciais ou combinatórias.

Devido a complexidade de sua estrutura interna divide-se em duas categorias conforme apresenta Figura 6:

- Arranjos Lógicos Programáveis que são formados por meio de um estrutura interna de conjuntos de portas AND-OR, divididos em dois tipos PLA e PAL,e por não possuir elementos de memória, não será objeto de estudo.

- Arranjos de Portas Programáveis (do inglês *Programmable Gate Array*) são formadas por estruturas mais genéricas e versáteis chamadas de blocos lógicos, destacando as FPGAs e os CPLDs, os quais serão comentados nas seções seguintes.

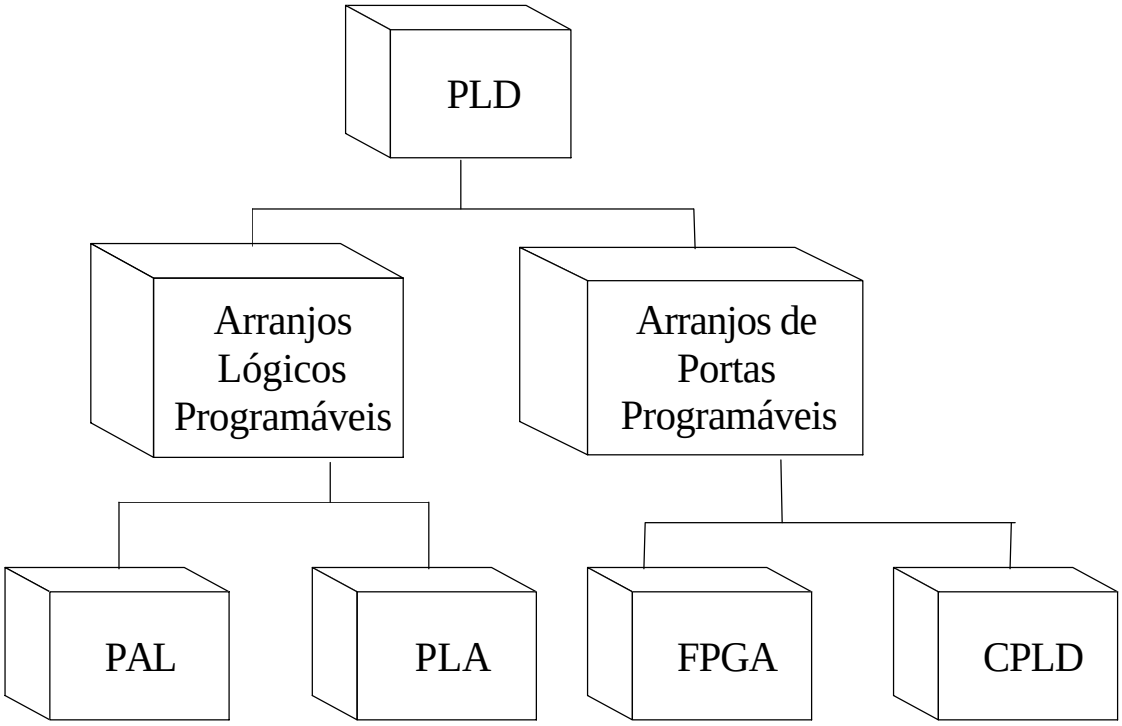

Figura 6 - Divisão de um PLD conforme sua estrutura.

Com o passar dos tempos, os dispositivos lógicos e programáveis tornaram-se extremamente densos, rápidos e complicados, exigindo-se dessa forma ferramentas CAD (do inglês *Computer-Aided Design*) cada vez mais complexas.

Dois importantes fabricantes de dispositivos programáveis disputam o mercado atualmente, os quais serão apresentados nas seções seguintes.

1.2.1.1.1 ALTERA

As arquiteturas programáveis da ALTERA [ALT 01] são conhecidas como CPLDs (*Complex Programmable Logic Devices*). A maior diferença destas arquiteturas para a arquitetura FPGA da XILINX da Actel está no roteamento dos blocos lógicos. Enquanto que as arquiteturas FPGA utilizam técnicas de roteamento onde os fios possuem comprimento variável e com atrasos dependentes do roteamento, as CPLDs utilizam matrizes de interconexão onde os atrasos nas conexões são previsíveis e, uma vez os blocos alocados independem do roteamento.

Do ponto de vista da tecnologia de programação dos componentes, as arquiteturas podem ser classificadas em quatro grupos: LUT[2], SOP[3], SRAM[4], FLASH[5], EEPROM[6] e EPROM[7].

Na Tabela 1 apresenta-se a classificação das arquiteturas da Altera de acordo com as seguintes características: Família de Componentes, Estrutura de Blocos Lógicos[8] e Tecnologia de Programação.

Família de Componentes	**Estrutura de Bloco Lógico**	**Tecnologia de Programação**
Stratics	LUT	SRAM
APEX II (1 Gbps)	LUT	SRAM
APEX 20 K	LUT	SRAM
FLEX 10K	LUT	SRAM
FLEX 8000	LUT	SRAM
MAX 9000	SOP	EEPROM
MAX 7000	SOP	EEPROM
MAX 5000 Classic	SOP SOP	EPROM EPROM
FLASHlogic	SOP	SRAM&FLASH

Tabela 1 - Classificação das Arquiteturas da Altera

[2] [3] [4] [5] [6] [7] [8] Ver glossário.

Conforme apresentado na Figura 7 podemos observar, que com a evolução do ponto de vista da arquitetura, a disposição dos blocos lógicos é cada vez mais envolvida por linhas de intersecção e roteamento, o que permite uma maior interação com outros componentes, além do aumento de portas e blocos que não foram representados na figura.

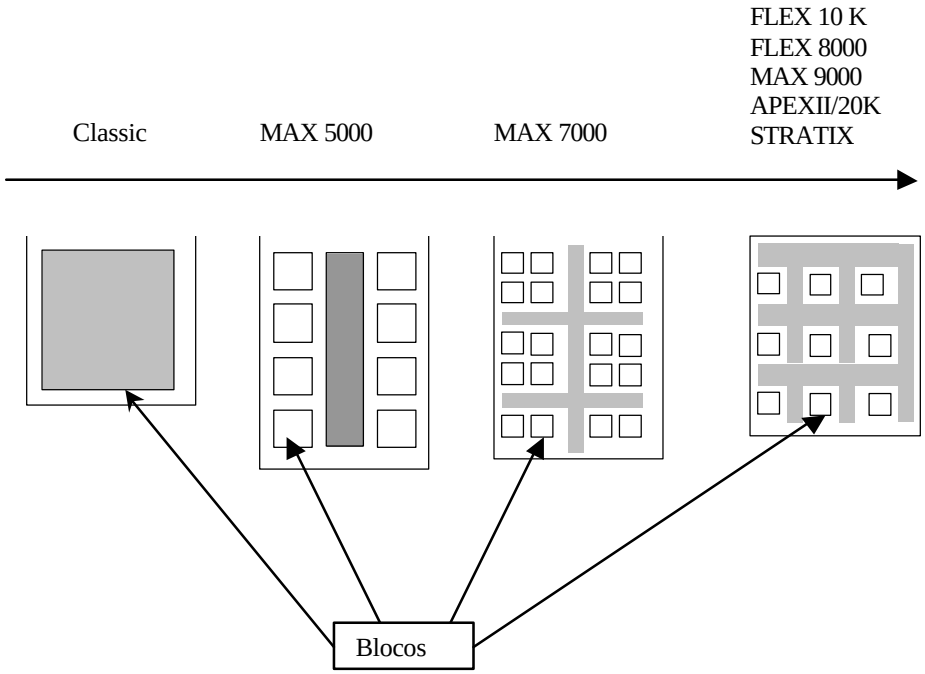

Figura 7 - Evolução da Arquiteturas da Altera

Nesta dissertação utilizou o ambiente MAX + PLUS II da Altera [ALT 01].

1.2.1.1.2 XILINX

A XILINX foi criada em 1984 com o objetivo de produzir circuitos programáveis pelos usuários. Em 1985 produziu o primeiro circuito FPGA. Em 1992, tornou disponíveis circuitos EPLD[9] (do inglês *Eraseble Programmable Logic Device*). Na atualidade fornece além dos componentes programáveis, um conjunto de ferramentas de *software* que roda em PCs e estações de trabalho, destinado a programação dos componentes [XIL 01].

[9] Ver glossário

Conforme a Figura 8 podemos observar a matriz de blocos lógicos, que pela disposição apresenta os blocos lógicos e o roteamento:

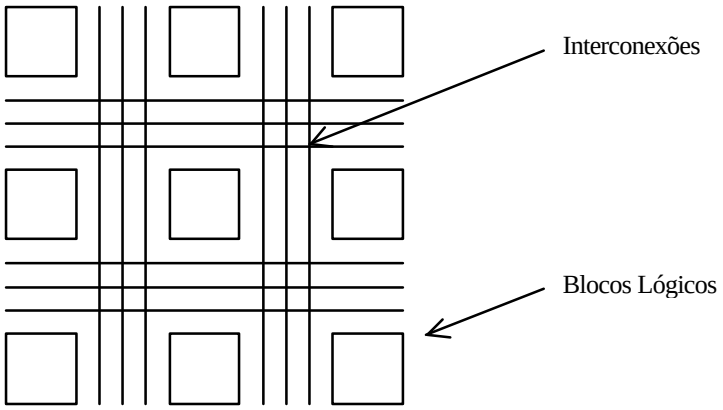

Figura 8 - Matriz Simétrica (Symetrical Array).

1.3 PROPOSTA DE PESQUISA

Este trabalho tem como proposta criar uma ferramenta de síntese de alto nível (do inglês *high-level synthesis*) denominada TAB2VHDL (TABELA TO VHDL) e a partir dela implementar vários circuitos integrados de códigos de linhas utilizados em transmissão de dados, no setor de telecomunicações, condicionados dos conceitos estudados sobre o cenário, diagrama Y, utilizando a linguagem VHDL [ARM 89] [PER 91] [SHA 86]. Criou-se desta forma, um ambiente amigável para a síntese de sistemas digitais que veio facilitar a tarefa do projetista, eliminando a descrição detalhada como no exemplo de uma metodologia de projetos digitais seqüenciais apresentada em [TAN 02a].

Atualmente dispõe-se de dois importantes tradutores disponíveis comercialmente que interpretam uma máquina de estado finito, descrita através de seu diagrama de estado.

Estas ferramentas são denominadas: Statecad [XIL 00] da Actel e Active-HDL [CYP 02] da Cypress.

Esses *softwares* destacam-se pelo uso do diagrama de estados para descrever o comportamento de uma máquina de estados finitos, traduzido-as para sua representação em

código VHDL. Nenhum processo de otimização é efetuado nesta etapa, deixando a cargo das ferramentas de síntese, ou seja, o MAX + Plus II ou XILINX. Estas ferramentas apresentam entradas gráficas que possibilitam expressar idéias em um modo natural, bem próximo da linguagem humana.

As etapas do projeto é apresentada na Figura 9, onde parte-se do nível RTL do domínio comportamental e chega-se ao nível lógico do mesmo domínio, ou seja, parte-se do diagrama de transição e estado e chega-se nas correspondentes funções booleanas em suas formas mínimas. Vários algoritmos de minimização de funções booleanas poderiam ser empregados nesta etapa do projeto. O programa TABELA emprega o método Quine-Mc Cluskey [DaS 89].

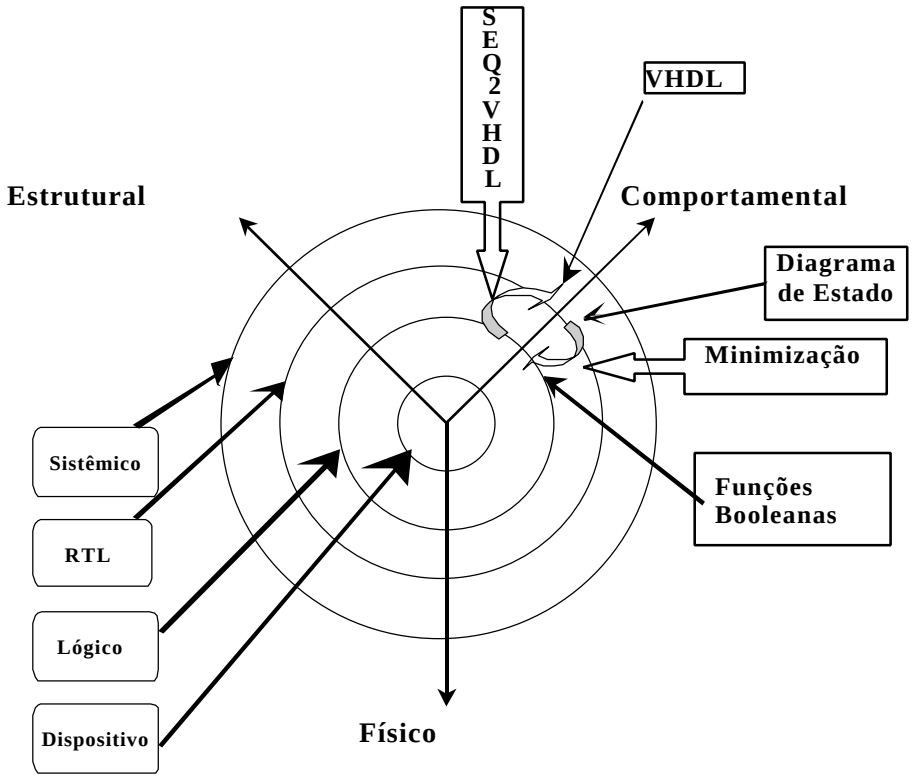

Figura 9 - Diagrama Y apresentando a ferramenta de síntese TAB2VHDL

Dadas as equações booleanas otimizadas, a ferramenta SEQ2VHDL as interpreta e gera um modelo RTL correspondente na linguagem VHDL, ou seja, parte-se do nível lógico e chega-se no nível RTL do domínio comportamental, conforme apresentado na Figura 9.

Como a descrição em VHDL independe da tecnologia, as ferramentas de síntese comerciais apropriadas levariam para o nível de implementação de estado no domínio físico.

Conforme GENOE [GEN 95] existem aproximadamente 6000 licenças para sintetizadores lógicos no mercado, entretanto, há uma carência de ferramentas que interpretem os níveis de abstração altos.

MACHADO et alli [MAC 97] empregam uma metodologia de especificação de sistemas digitais, baseada em Rede de Petri orientada por objetos para obter de forma rápida e simplificada um modelo VHDL do sistema pretendido.

LI e LEESER [LIY 00] desenvolveram uma nova linguagem de descrição denominada HML, baseada na linguagem de programação funcional SML e um tradutor para gerar códigos VHDL a partir da descrição HML.

LOHSE et alli [LOH 94] descrevem um tradutor para gerar um subconjunto da VHDL partindo de diagrama de decisão binário.

MARRA et alli [MAR 99] desenvolveram o C2VHDL, um tradutor que converte código C para o código VHDL.

Um tradutor de linguagem, foi proposto por BONATTI [BON 95] e implementado em um pacote de domínio público denominado Stoht.

CONSTANTINIDES et alli [CON 99] et alli desenvolveu uma ferramenta de síntese automática que a partir de uma FSM otimiza uma FPGA, usando uma técnica de decomposição a qual divide a FSM em várias sub-máquinas e usa uma técnica de Algoritmo Genético para explorar o espaço de possíveis partições.

FUHRER et alli [FUH 97] descrevem uma solução para minimização de estados de uma FSM para circuitos lógicos.

Portanto, muitas outras ferramentas de síntese de Alto Nível são descritas na literatura, comprovando a importância desse tipo de abordagem.

1.3.1 OBJETIVO

Este trabalho visa desenvolver uma ferramenta de síntese que gera um modelo funcional de síntese na linguagem VHDL.

Das especificações do comportamento de uma máquina de estados finitos através de um diagrama de estados, descrito no modelo de Mealy, o programa TABELA [DaS 89] gera equações booleanas dos controles dos elementos de memórias e de saídas.

1.3.2 ORGANIZAÇÃO DA DISSERTAÇÃO DE MESTRADO

No Capítulo 1 – INTRODUÇÃO

No Capítulo 2 – MODELAGEM E LINGUAGEM DE DESCRIÇÃO DE SISTEMAS DE TELECOMUNICAÇÕES

No Capítulo 3 – Projeto TAB2VHDL (TABELA E SEQ2VHDL).

No Capítulo 4 – ESTUDOS DE CASOS.

Apêndice I –ANÁLISE ESTRUTURAL DO AMBIENTE TABELA E SEQ2VHDL

Apêndice II – PROGRAMA SEQ2VHDL.

Apêndice III – LISTAGENS DOS CÓDIGOS DE LINHA.

1.3.3 CONTRIBUIÇÕES

O trabalho em questão trará contribuições de grande importância para as áreas educacionais no ensino de ferramentas de síntese no curso de Engenharia Elétrica do DEE – FEIS – UNESP - Ilha Solteira, no curso de Tecnologia de Produção Moveleira do CEUV – Votuporanga e no curso de Sistemas de Informação da FAIJALES – Jales, pois pretende-se utilizá-la em disciplinas ministradas na graduação e na pós-graduação.

A partir da documentação apresentada nos Apêndices I e II, que é de domínio público ao contrário dos tradutores que possuem direitos autorais, os pesquisadores poderão desenvolver novas ferramentas de otimização e criar ferramentas através de outras

linguagens de *softwares* e *hardwares*, respeitando suas características segundo o modelo adotado, conforme capítulo 2.

Permitirá aos alunos implementar sistemas autônomos, utilizando diagrama de estados, sem preocupar-se com detalhes de projetos digitais, pois com a ferramenta de síntese desenvolvida, a partir de uma FSM, criam-se protótipos de circuitos que se integrarão a sistemas de telecomunicações e de automação industrial, cujo custo de implementação será minimizado .

CAPÍTULO 2

MODELAGEM E LINGUAGEM DE SISTEMAS DE TELECOMUNICAÇÕES

Neste capítulo, descreve-se sobre o enfoque tecnológico computacional, o estado da arte dos modelos de sistemas de telecomunicações, e em seguida, as principais linguagens de descrição de *hardwares* e *softwares*, para finalizar, tem-se as características relevantes a serem consideradas para a escolha da linguagem utilizada na modelagem de um projeto de telecomunicações.

2.1 MODELOS DE SISTEMAS DE TELECOMUNICAÇÕES

A definição de um modelo de sistemas de telecomunicações não é simples, pois se pode ter vários enfoques: do usuário, do planejador de redes, da concepção teórica ou da concepção tecnológica.

Portanto o modelo funcional conforme o ponto de vista do usuário, pode ser representado por uma "nuvem" e duas interfaces de acesso usuário-rede, de acordo com a Figura 10.

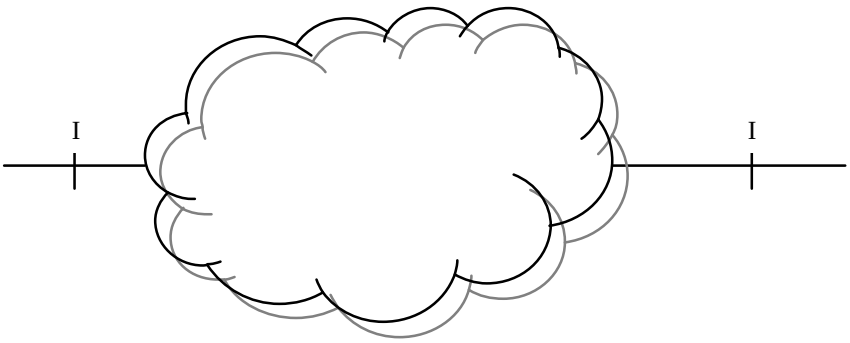

Figura 10 - Modelo Funcional representados por uma nuvem

Com relação aos serviços, pode ocorrer uma grande confusão, por exemplo, ao definir o modelo, sobre o enfoque tecnológico, sob o ponto de vista da empresa concessionária de telecomunicações o serviço é visto como uma troca de informações de rede que é capaz de transportar informações na faixa de 0 até 3,1 KHZ, podendo ser utilizada para telefonia (transporte de voz), fax e comunicação de dados, entretanto sobre o enfoque do usuário, não pode ser definida como tal, por não incluir a funcionalidade do aparelho telefônico [UEH 97].

Durante o desenvolvimento de sistemas complexos de telecomunicações a dedução, a modelagem e análise de requisitos é o principal desafio, pois tem-se uma rede heterogênea, que apresenta uma distribuição mundial. Portanto muitas propostas de pesquisas têm sido feitas através da descrição de sistemas comportamentais [AMY 00].

O modelo adotado neste trabalho será semelhante ao adotado por [LEE 99], ou seja, do ponto de vista tecnológico que é utilizado na concepção de sistemas digitais, originado de sistemas computacionais, o modelo de computação (em inglês *model of computation* ou MOC), é a "lei da física" dos componentes concorrentes, incluindo o que eles são (ontologia), como eles se comunicam, como seu controle de fluxos são relatados (protocolos), e que informação contém seus elementos (epistemologia). Estas são suas semânticas concorrentes.

2.1.1 EXEMPLOS DE MODELOS [LEE 99]

Existe uma grande variedade de modelos computacionais, os modelos usuais utilizados em sistemas embarcados, que por definição são computadores mascarados, ou melhor, uma caixa preta que na maioria das execuções tem um conjunto de tarefas com relação custo/benefício reduzido.

Todos os sistemas realizam tarefas que envolvem comunicação e processamento de sinais, necessitando para isso de interfaces. A representação desses sistemas utiliza-se de bolas, arcos e diagramas com setas e blocos.

Descreve-se a seguir alguns desses modelos.

2.1.1.1 MODELO PARA EQUAÇÕES DIFERENCIAIS

No modelo para equações diferenciais, uma possível semântica para a sintaxe da equação diferencial é representada na Figura 11. Os arcos representam funções contínuas e um determinado tempo. As bolas representam a relação entre essas funções. Dessa forma, o objetivo do modelo é encontrar um ponto fixo, isto é, um conjunto de funções do tempo que satisfaça a relação. Portanto, são excelentes para modelagem de circuitos analógicos e muitos sistemas físicos. Utiliza-se a linguagem VHDL-AMS [SAS 97], Simulink e SPICE.

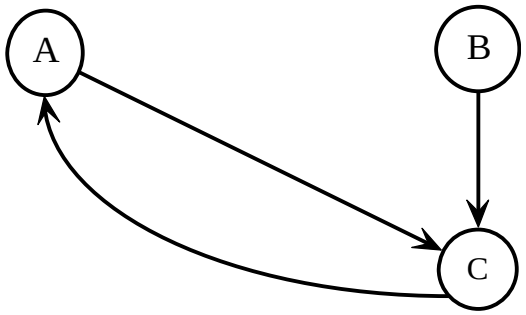

Figura 11 -Sintaxe simples (bolas e arcos ou diagrama de blocos e setas).

2.1.1.2 MODELO EQUAÇÃO DIFERENÇA

Outro modelo seria a Equação diferencial discretizada, ou seja, em um modelo a equação diferença, popularmente usada para modelos computacionais, utiliza-se de sinais digitais.

2.1.1.3 MODELO DE SINCRONISMO / REATIVO

No Modelo de Sincronismo / Reativo (do inglês *synchronous / reactive* ou SR), os arcos representam o valor dos dados que são alinhados pelo sinal do *clock* global. Portanto, seus sinais são discretos, como ocorre na equação diferença, mas diferente da equação diferencial, onde o sinal não precisa possuir o mesmo valor durante o sinal do *clock*. Assim as bolas representam a relação entre entrada e saída.

2.1.1.4 MODELO DE EVENTO DISCRETO

No modelo computacional de evento discreto (do inglês *discrete-event* ou DE), o arco representa o conjunto de eventos em um determinado tempo. Um evento consiste de um valor e um tempo fixo. Este modelo computacional é popular para especificar *hardware* e simulação de sistemas de telecomunicação, e tem sido utilizado em um grande número de simuladores.

2.1.1.5 MODELO CICLO DIRIGIDO

Alguns sistemas nos quais o tempo do evento é conduzido primariamente pelo *clock*, ou seja, sinais com eventos repetidos indefinidamente em um intervalo de tempo fixo, através da modelagem de eventos discretos possibilitaria a sua implementação, no entanto se tornaria uma solução cara.

2.1.1.6 PASSAGEM DE MENSAGEM SÍNCRONA

Na passagem de mensagem síncrona, os componentes são processos, e as comunicações desses processos devem ser atômicas (ações instantâneas chamadas de *"rendezvous").* Se dois processos estão prontos para se comunicar, e por uma razão, o primeiro atinge o ponto esperado, esse protela, até que o outro processo, que espera se comunique. Portanto, atômico significa que dois processos estão envolvidos em uma troca, e que essa troca é iniciada e completada em um simples passo ininterrupto. Nesta categoria têm-se vários exemplos de modelos que incluem os Processos Seqüenciais de Comunicação (do inglês *Communicating Seqüencial Process* ou CSP) e Sistemas de Comunicação de Cálculos (do inglês *Calculus of Communicating System* ou CCS).

2.1.1.7 MÁQUINAS DE ESTADOS FINITOS

Na Máquina de Estados Finitos (do inglês *finite-state machine* ou FSM), bolas representam o estado do sistema e arcos representam a transição de estado, este modelo não é concorrente, pois as execuções são seqüências ordenadas de transições de estado. Assim, sistemas de transição são versões gerais, no qual uma bola possibilita representar mais que um estado do sistema (e pode existir um número infinito de bolas).

Uma grande variedade de ferramentas para FSM é apresentada em [AT& 02], para as mais diversas aplicações como pesquisas sobre FSM.

Pode-se notar que FSM são excelentes para controle lógico em sistemas embarcados, motivo pelo qual foi adotado como modelo neste trabalho. Sendo assim apresenta-se a seguir a representação de um sistema digital para uma máquina de estado finito.

2.1.1.7.1 REPRESENTAÇÃO DE UM CIRCUITO DE UMA FSM [TAN 02a]

Uma máquina de estados finitos tem a representação conforme delineado na Figura 12 [KOH 78].

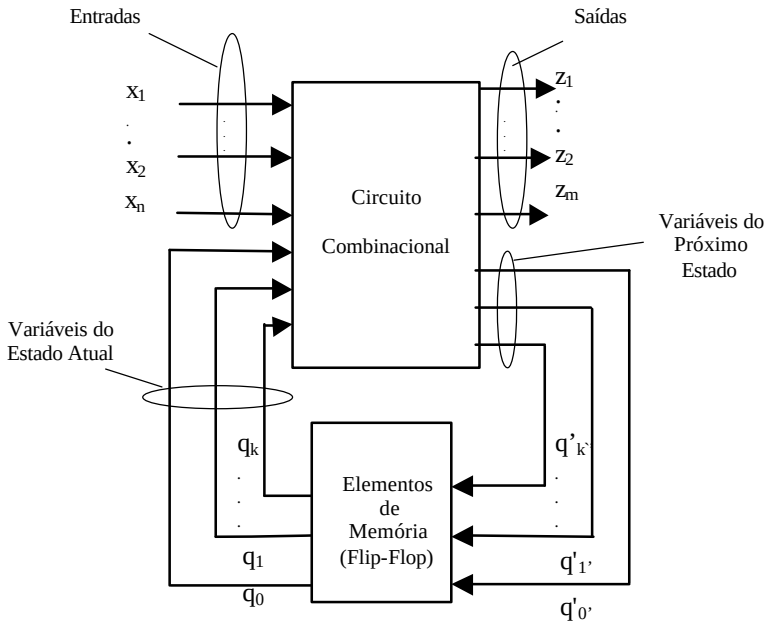

Figura 12 - Representação de um circuito de uma máquina seqüencial síncrona

O comportamento de uma FSM é representado como uma seqüência de eventos que ocorrem em instantes discretos, designados de t= 1,2,3, etc. Suponha que uma máquina M tenha recebido sinais de entrada (x_1, x_2, ..., x_n), uma n-upla ordenada na forma de vetor de entrada, e tenha respondido produzindo sinais de saída (z_1, z_2, ..., z_m). Uma m-upla ordenada destes valores é o vetor de saída.

Se agora, no tempo t, aplica-se um sinal x(t) para M, ela responde com z(t) que depende de x(t) e de sua entrada passada em M. O valor da variável de memória é chamado de variável do estado atual (q_0, q_1, ..., q_k). Os valores armazenados nos k elementos de memória definem o estado interno atual da máquina.

As saídas z_1, z_2, ..., z_m e as variáveis do próximo estado q_0', q_1', ..., q_k' são funções das entradas externas e do estado interno e são definidas através do circuito combinacional representado na Figura 12. Os valores de q que aparecem na saída do circuito

combinacional no instante t, determinam os valores das variáveis de estado no instante t+1, e portanto definem o próximo estado.

2.1.1.7.2 MODELOS DE MÁQUINAS SÍNCRONAS

A. MÁQUINA DE MEALY

Na máquina de Mealy [WAK 00] as saídas e o próximo estado dependem do estado atual e da entrada no tempo t, conforme Figura 13.

$$z(t) = H(y(t), x(t)) \quad (1.1)$$

$$\text{próximo estado} = H(y(t), x(t)) \quad (1.2)$$

sendo que, $x(t)$ é o estado de entrada e $y(t)$ o estado atual

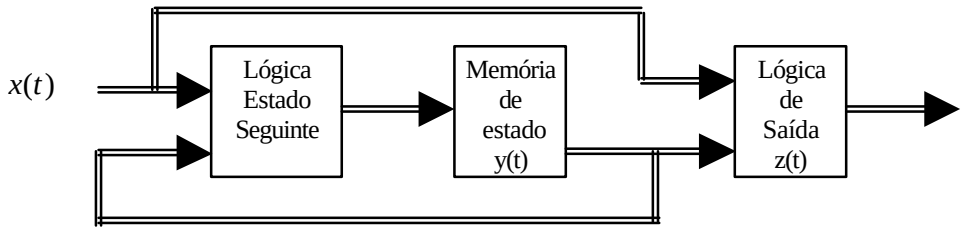

Figura 13 - Máquina de Mealy

B. MÁQUINA DE MOORE

A máquina de Moore [WAK 00] é um sistema seqüencial onde as saídas dependem apenas do estado atual do circuito, conforme Figura 14 .

$$z(t) = H(x(t)) \quad (1.3)$$

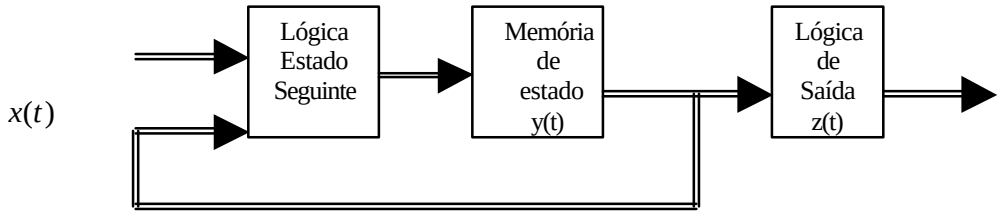

Figura 14 - Máquina de Moore

Outros tipos de modelos de descrição são abordados em [LEE 99] como Passagem de Mensagem Assíncrona, e Tempo CSP (abreviação do inglês Communicating Seqüencial Process) e tempo PN (abreviação do inglês *Publish* e *Subscribe* e *Unstructured events).*

2.1.2 EVOLUÇÃO E ESCOLHA DO MODELO DE DESCRIÇÃO

Uma grande variedade de MOC tem surgido no mercado entretanto, muitos projetistas não aproveitam essas oportunidades de escolha, pois preferem dominar um ou dois modelos como conseqüência da complexidade das mesmas.

Esta realidade está mudando paulatinamente, principalmente pela oferta de ferramentas que trabalham com diferentes níveis de abstração e em diferentes domínios do projeto.

Dessa forma a escolha de um modelo de descrição de sistemas de telecomunicações auxiliará muito o projetista. Dentre aos vários modelos, deve-se salientar, novamente, que a FSM foi escolhida pela facilidade de interpretação, o que facilita a entrada de dados para a ferramenta de síntese.

Apresenta-se a seguir as principais linguagens de descrição de *softwares* e *hardwares* para sistemas de telecomunicações.

2.2 LINGUAGENS DE DESCRIÇÕES

EDWARDS [EDW 01] descreve que para cada modelo em estudo, apresenta-se uma diversidade de características que podem ou não ser adaptáveis entre as várias linguagens, em que determinadas ocasiões não estão aptas a serem utilizadas.

Um sistema embarcado é uma linguagem que tem como propósito comum e suficiente resolver tarefas que precisam ser escritas, analisadas e compiladas.

Os sistemas embarcados que possuem componentes envolvidos por *hardware* e *software* têm recebido muita atenção nos últimos anos [ONL 96].

Atualmente uma grande variedade de linguagens está evoluindo. Cada vez mais para um domínio específico.

Por exemplo, a linguagem assembly é conveniente para o processamento final, porém inadequada para ser empregada no domínio comportamental, por possuir uma quantidade limitada de instruções de controle.

Faz-se em seguida, um breve comentário sobre as principais linguagens de descrição envolvidas em um projeto de telecomunicações.

2.2.1 LINGUAGEM DE DESCRIÇÃO DE HARDWARE

As linguagens de descrição de *hardware* ou HDL (abreviatura do inglês *Hardware Description Language*) permitem ao projetista desenvolver sistemas através de circuitos digitais, sem a necessidade do conhecimento de detalhes da teconologia de implementação, facilitando dessa forma as modificações e a implementação do sistema.

Está disponível uma grande variedade de HDL, como por exemplo, a AHDL [ALT 02] o Verilog HDL [GOL 86] e a VHDL, entretanto algumas foram padronizadas pelo IEEE.

Adotou-se como padrão do ambiente desenvolvido TAB2VHDL a linguagem VHDL. Descreve-se a seguir de forma sucinta os principais componentes da linguagem. Na seção seguinte faz-se um breve comentário sobre o Verilog HDL.

2.2.1.1 VHDL

A VHDL é uma linguagem de descrição de *hardware* usada para programar o comportamento e a estrutura de um sistema digital. (O significado de VHDL é a abreviatura do inglês *VHSIC Hardware Description Language* sendo que o termo VHSIC significa *Very High Speed Integrated Circuit*).

A VHDL é uma linguagem de descrição de *hardware* de propósito geral, pois pode ser empregada para descrever e simular a operação de uma grande variedade de sistemas digitais, cujo grau de complexidade pode variar de algumas portas lógicas a interconexões de muitos circuitos integrados complexos. Esta linguagem teve origem nos laboratórios militares com o propósito de uniformizar métodos para a especificação de sistemas digitais, que após ter sido padronizada pelo IEEE passou a ser utilizada pela indústria de eletrônica.

Com relação ao comportamento pode-se descrever os sistemas digitais em três níveis de abstração, ou seja, no comportamental, no funcional também conhecido como *Data Flow* e no estrutural. Por exemplo, um somador binário pode ser descrito no nível comportamental em termos de suas funções de adição de dois números binários, sem especificar detalhes de implementação. O mesmo somador pode ser descrito no nível funcional através das equações lógicas (booleanas) para o somador e finalmente pode-se descrevê-lo no nível estrutural pela especificação da interconexões das portas que compõem o somador.

Como a ferramenta de síntese desenvolvida utiliza-se da descrição em nível funcional para implementar as funções combinacionais de controle dos elementos de memória e das saídas dar-se-á mais ênfase nesse tipo de modelagem, porém para ilustrar a potencialidade da linguagem, alguns exemplos em nível comportamental também serão apresentados. Isto se torna necessário, pois esta é uma linguagem natural para a metodologia de projeto *top-down*, em que o sistema é inicialmente especificado em alto nível de abstração e testado, utilizando um simulador. Após o sistema ter sido depurado nesse nível, o projeto é gradativamente refinado em sua especificação, levando eventualmente a uma descrição estrutural totalmente detalhada com a tecnologia de implementação.

É importante salientar que a VHDL foi concebida para ser independente da tecnologia, o que significa que se o modelo especificado for implementado em uma tecnologia utilizada nos dias atuais, também poderá ser implementado em tecnologias que serão desenvolvidas no futuro.

A descrição de qualquer projeto em VHDL consiste de um par de declarações denominadas de entidade (do inglês *ENTITY*) e arquitetura (*ARCHITECTURE*). A declaração de entidade descreve os portos de entrada e saída do projeto e o corpo da arquitetura descreve o comportamento do sistema. O modelo VHDL apresentado a seguir representa uma porta E (AND) contendo como entrada os sinais denominados de A e B e como saída o sinal denominado de F, sendo que o tipo de dados permitido é do tipo BIT.

```
ENTITY Ex_and2 IS
PORT (
       A,B : IN  BIT;
         F : OUT BIT );
END Ex_and2;

ARCHITECTURE Porta_and OF Ex_and2 IS
BEGIN
       F <= A AND B;
END Porta_and;
```

A entidade pode ser considerada como sendo uma caixa preta contendo portos, que são pontos de comunicação com a parte externa da caixa. Esses portos são muitas vezes associados com os pinos dos dispositivos. Cada porto tem associado a ele um nome, um modo e um tipo.

O modo de um porto refere-se à direção para onde o dado é transferido. Os modos permitidos são:

- **IN** : Os dados entram na entidade;
- **OUT** : Os dados saem da entidade. Eles não podem ser utilizados internamente;
- **INOUT**: Os dados são bidirecionais. Vão para dentro e para fora da entidade;
- **BUFFER**: Dados que saem da entidade e que também podem ser utilizados como realimentação.

O tipo BIT é um tipo definido na biblioteca padrão (*Standard library*) e pode assumir somente os valores ‘0’ e ‘1’. Existem, porém, várias outras bibliotecas sendo que a biblioteca IEEE.std_logic_1164 define os seguintes tipos de dados:

- ‘0’ : força a zero;
- ‘1’ : força a um;
- ‘Z’ : alta impedância;
- ‘L’ : zero fraco;
- ‘H’ : um fraco;
- ‘-‘ : *don’t care*.

A arquitetura especifica o que está dentro da caixa preta, ou seja, qual a estrutura ou o comportamento da entidade. A descrição de uma arquitetura pode ser uma combinação dos estilos estruturais, funcionais e comportamentais.

No estilo estrutural os componentes que são parte do sistema, são instanciados no modelo e as conexões são estabelecidas. É uma sistemática muito semelhante à captura de esquemático. Apresenta-se a seguir um modelo estrutural para comparar dois números A e B.

```
ENTITY compara IS
PORT (
  A, B   : IN  std_logic_vector ( 3 DOWNTO 0)
  IGUAL : OUT std_logic);
END compara;

USE WORK.pkg_portas.ALL;

ARCHITECTURE estrutural OF compara IS
  SIGNAL X : std_logic_vector ( 3 DOWNTO 0);
BEGIN
  uo: xnor2 PORT MAP ( A(0),  B(0), X(0));
  u1: xnor2 PORT MAP ( A(1),  B(1), X(1));
  u2: xnor2 PORT MAP ( A(2),  B(2), X(2));
  uo: xnor2 PORT MAP ( A(3),  B(3), X(3));
  u4: and4  PORT MAP ( X(0),  X(1), X(2),X(3),IGUAL);
END estrutural;
```

Neste modelo as instâncias xnor2 e and4 foram previamente definidas na *package* denominada pkg.portas, onde por meio de descrições individuais, essas portas foram descritas e modeladas. Dessa forma, podem ser instanciadas por meio do mapeamento dos portos (do inglês *PORT MAP*).

Neste tipo de descrição, utilizam-se componentes desenvolvidos pelos fabricantes dos ambientes de sínteses. Por exemplo, a Altera disponibiliza uma vasta biblioteca de componentes que foram projetados, otimizados e testados para serem implementados nos FPGA e CPLD por ela fabricados. Sempre que possível esses componentes devem ser utilizados.

O mesmo exemplo pode ser descrito, no estilo funcional, também chamado de *Data Flow* ou RTL. Neste modelo o circuito é codificado por meio das funções booleanas que descrevem o funcionamento do sistema. Logo abaixo, exemplifica-se o estilo funcional para o circuito comparador.

```
ENTITY compara IS
PORT (
       A, B : IN  std_logic_vector( 3 DOWNTO 0)
       IGUAL: OUT std_logic);
END compara;

ARCHITECTURE funcional OF compara IS
BEGIN
  IGUAL <= '1' WHEN A = B ELSE '0';
END funcional;
```

O estilo comportamental é caracterizado pelo alto nível de abstração na descrição dos modelos. As descrições têm a forma IF ... THEN. O circuito do exemplo, que está sendo estudado, é representado pelo seguinte modelo:

```
ENTITY compara IS
PORT (
       A, B :    IN   std_logic_vector( 3 DOWNTO 0)
       IGUAL:    OUT  std_logic);
END compara;

ARCHITECTURE comportamental OF compara IS
BEGIN
  Comp: PROCESS(A,B)
  BEGIN
       IF A = B THEN
            IGUAL <= '1';
       ELSE
            IGUAL <= '0';
       END IF;
  END PROCESS;
END comportamental;
```

Existem dois tipos de declaração em VHDL, as concorrentes e as seqüenciais.

As declarações concorrentes estão fora do *PROCESS* e são avaliadas concorrentemente durante o processo de simulação. Os *PROCESS* também são instruções concorrentes. São exemplos de declarações concorrentes as equações booleanas, as atribuições de sinais condicionais e seletivos (*WHEN/ELSE*, *WITH/SELECT*) e as declarações de instanciação.

Apresenta-se a seguir exemplos de declarações concorrentes:

-- Exemplos de equações booleanas.

X <= (A **AND** (**NOT** SEL1) **OR** (B **AND** SEL1);

G <= **NOT** (Y **AND** SEL2);

-- Exemplo de atribuição condicional.

Y <= D **WHEN** (SEL1 = ‘1’) **ELSE** C;

H <= ‘0’ **WHEN** (X = ‘1’ AND SEL2 = ‘0’) **ELSE** ‘1’;

-- Exemplo de instanciação.

Inst: nand2 **PORT MAP** (H, G, F);

Apesar do *hardware* ser concorrente, ele pode ser modelado através de um algoritmo por uma série de declarações seqüenciais. Por definição, as declarações seqüenciais são agrupadas utilizando-se a declaração *PROCESS*.

A declaração *PROCESS* é usada para construir algoritmos permitindo-se dessa forma agrupar declarações seqüenciais. Durante a simulação as declarações incluídas dentro de um processo são executadas seqüencialmente.

Uma arquitetura pode conter qualquer número de processos sendo que cada processo é executado concorrentemente. Os processos podem ser utilizados para modelar lógicas combinacionais ou seqüenciais (síncronas).

O processo é executado quando ocorrer um evento em qualquer um dos sinais que estiverem contidos na lista de sensibilidade. Assim, define-se evento a qualquer alteração do valor lógico sobre um sinal. As declarações *IF-THEN-ELSE* e *CASE-WHEN* são declarações seqüenciais que especificam situações condicionais.

O exemplo de atribuição de sinal condicional *IF-THEN-ELSE* é apresentado como parte da arquitetura de um circuito multiplexador.

```
ARCHITECTURE  ..
...
Mux: PROCESS(A, B, S)
BEGIN
  IF S = ‘0’ THEN
        X <= A;
  ELSE
        X <= B;
  END IF;
END PROCESS;
...
END architecture;
```

Um outro exemplo ilustrativo envolvendo descrição comportamental é um contador de 4 bits com *reset* síncrono. Neste exemplo também se apresenta somente a declaração *PROCESS* contida na arquitetura.

É importante ressaltar que a declaração CLK'EVENT AND CLK = '1' implica que a declaração de sinal subseqüente ocorre na borda de subida do sinal CLK, pois a condição IF só será verdadeira se ocorrer um evento (alteração do valor lógico) no sinal CLK e o valor for igual a 1.

```
ARCHITECTURE
...
...
...
upcounter: PROCESS(CLK)
BEGIN
 IF CLK'EVENT AND CLK = '1' THEN
     IF RESET = '1' THEN
      COUNT<= "0000";-- poderia ser empregada a notação x"0"
     ELSE
               COUNT = COUNT + 1;
     END IF;
 END IF;
END PROCESS;
...
...
END architecture;
```

Um outro exemplo relevante é o apresentado num contador de 4 bits com *reset* assíncrono.

```
upcounter: PROCESS(CLK, RST)
BEGIN
  IF RST = '1' THEN
       COUNT <= X "0";
  ELSIF CLK'EVENT AND CLK = '1' THEN
       COUNT = COUNT + 1;
  END IF;
END PROCESS;
```

Observe que esse processo é sensível tanto a mudança de valor no sinal CLK quanto no sinal RST.

A linguagem VHDL, assim como qualquer outra linguagem de descrição, também disponibiliza alguns operadores nativos (próprios da linguagem), são eles:

- Lógicos:

  AND, NAND, OR, NOR, XOR, XNOR e NOT.

- Relacionais:

  = (igual), /= (não igual), < (menor que), <= (menor que ou igual),
  > (maior que), >= (maior que ou igual).

Operadores complexos podem ser desenvolvidos e acrescentados nos modelos através das *PACKAGES*, que são desenvolvidas pelos usuários e algumas já são disponibilizadas pelo fabricante do ambiente de síntese.

Muito sobre VHDL ainda teria que ser apresentado para que o leitor possa ter um perfeito entendimento sobre a linguagem. Boas revisões biográficas na área de Sistemas Digitais são apresentadas em [WAK 00] e [ERC 00], Ferramentas comerciais em [ALT 01] e [XIL 01]. A abordagem aqui apresentada tem como objetivo dar uma noção sobre os conceitos básicos necessários para que se possa entender o tipo de ferramenta desenvolvida.

2.2.1.2 VERILOG HDL [MAG 92]

O Verilog HDL é uma linguagem originalmente desenvolvida pela *Gateway Design Automation*, empresa que mais tarde, passou a pertencer a Cadence Design System.

Inicialmente, era uma linguagem utilizada somente no simulador *Cadence Verilog-XL*, face à concorrência do VHDL, em 1990 passou a ser de domínio público. O nome "Verilog" é registrado pela Cadence. O uso do Verilog HDL foi promovido pelo OVI

(abreviatura do inglês *Open Verilog International*), tendo a primeira versão do Manual de Referência de Descrição de *Hardware* sido publicada em Outubro de 1991.

Atualmente o Verilog HDL é padronizado pelo IEEE *Standard Verilog Hardware Description Language* ou IEEE Std 1364-2001 (Revisão do IEEE Std 1364-1995) [IEE 01].

Nas Tabela 2, 3 e 4, a seguir, são comparados as principais características do VHDL e do Verilog HDL nas.

	VHDL	**Verilog**
Origem	Padrão IEEE	Cadence Design System, Inc.
Status	Domínio Público	Proprietário, mais tarde de domínio público
Aceitação dos Projetistas	Emergente	Largamente usada

Tabela 2 - Origem e Status

	VHDL	**Verilog**
Sintaxe Próxima	ADA	C, Pascal
Nível de Complexidade	Difícil	Fácil
Significado Direto do Hardware	Não Verdadeiro	Sim
Requer Conhecimento de Software	Sim	Não

Tabela 3 - Principais Características

NÍVEL DE DESCRIÇÃO	**VHDL**	**Verilog**
Nível Funcional	BOM	REGULAR
Nível Comportamental	BOM	BOM
Register Transfer Level (RTL)	BOM	BOM
Nível Lógico	BOM	BOM
Nível De Chaves	NÃO	BOM
Nível Elétrico	NÃO	NÃO

Tabela 4 - Níveis de Descrição

2.2.2 LINGUAGEM *SOFTWARE* [TAN 02b]

As Linguagens de *softwares* descrevem uma seqüência de instruções e regras, que de certa forma, envolvem a tradução das tarefas que o processador deve executar em uma ordem pré-determinada.

Um breve resumo das principais linguagens de *softwares* utilizadas para o projeto de sistemas de telecomunicação é apresentado abaixo:

- **Linguagem C** é uma linguagem orientada para procedimentos e de objetivos gerais, que pode ser utilizada em aplicações comerciais e científicas. Suas características são próximas à da linguagem *Assembly*, o que a torna perfeitamente adequada para escrever *softwares* complexos de sistemas de telecomunicações, além do padrão ISO e ANSI (do inglês *American National Standards Institute*), detalhes dos comandos da linguagem C é apresentado em [TAN 02b].

- **Assembly** é uma linguagem de baixo nível, ou seja, uma linguagem de programação que é uma lista de instruções que será executada pelo processador, escrita de uma forma simbólica, de acordo com sentenças que representam instruções em linguagem de máquina, obtendo como conseqüência execuções mais rápidas. Esta linguagem é considerada de difícil implementação, por estar distante da linguagem natural humana, e próxima a linguagem de máquina, conforme o diagrama Y.

- **Linguagem C++** é um aperfeiçoamento da linguagem C, e conseqüentemente, tem sua mesma aplicabilidade, no entanto usa a abordagem de orientação por objetos.
- **Linguagem SystemC** é um subconjunto da linguagem C++, tendo como objetivo a descrição de *hardware.*
- **Java** tem várias características que a tornam vantajosa frente a outras linguagens, tais como Orientada a Objeto, Independente de Plataforma, sem ponteiros, sendo que sua grande desvantagem é a baixo rendimento relativa a outras linguagens tais como C e a impossibilidade de acessar o *hardware* da máquina real.

Notamos que cada linguagem de programação usa seu próprio conjunto de símbolos e regras específicos para descrever determinadas tarefas, veremos agora algumas das características que foram predominantes para a escolha das linguagens utilizadas na elaboração do ambiente e da especificação do *hardware*.

2.2.3 CARACTERÍSTICAS RELEVANTES NA ESCOLHA DAS LINGUAGENS

Neste item faz-se uma comparação dos fatores relevantes para a escolha das linguagens do programa SEQ2VHDL, sendo que a linguagem C será utilizada como ferramenta para a elaboração do programa, enquanto que a linguagem VHDL especificará os projetos que serão sintetizados a partir da ferramenta MAX+Plus II da Altera.

Conforme Tabela 5, apresenta-se às especificações das linguagens de descrição de *hardware* e *software* adotadas:

Características	**Linguagens Adotadas**	
	VHDL	**C**
Idealização da Linguagem[10]	*Hardware*	*Software*
Implementação de Sistemas em Baixo Nível [11]	Excelente	Alto
Transição de Estados [12]	Sim	Sim
Representação Lógica [13]	Alta	Não
Sincronismo [14]	Alto	Alto
Facilidade de Corrigir Erros e Manutenção [15]	Média	Alta
Público Alvo [16]	Médio	Muito
Ferramentas Disponíveis [17]	Muitas	Muitas
Aprendizagem [18]	Alta	Média
Documentação [19]	Médio	Baixo

Tabela 5 - Comparação das especificações das linguagens adotadas

Logo abaixo se faz um breve comentário de duas principais ferramentas de projetos existentes no mercado geradas a partir de uma Máquina de Estado Finito.

10 11 12 13 14 15 16 17 18 19 Ver glossário

2.2.4 FERR MENTAS PARA ENTRADAS FSM DE PROJETOS HDL

2.2.4.1 STATECAD

O StateCAD é um *software* que usa o diagrama de bolas para descrever uma máquina de estado, gerando o mesmo projeto nas linguagens VHDL ou Verilog. Podem apresentar representações lógicas e máquinas de Mealy/Moore.

O StateCAD otimiza o código gerado em função da velocidade, da área e outros.

2.2.4.2 ACTIVE-HDL

O Active-HDL é um ambiente de projeto HDL específico para Xilinx, para FPGA, este *software* de verificação e entrada de projetos VHDL e Verilog é capaz de manipular os dispositivos Xilinx.

O ambiente oferece a opção de simulação VHDL ou Verilog. O simulador VHDL é compatível com IEEE 1076-87/93 enquanto que o simulador Verilog cumpre com o padrão IEEE 1364-95.

Na Tabela 6, apresentamos algumas características do software Active-HDL.

Características	**Active-HDL**
Editor HDL	SIM
Editor de Máquina de Estado	SIM
Editor de Esquema	SIM
Editor de Diagrama de Bloco	SIM
Gerenciador de Fluxo de Síntese	SIM
Geração de "Testbench"	SIM
Visualizador de Formas de Ondas	SIM

Tabela 6 - Características do VHDL.

No próximo capítulo, descreve-se o ambiente TAB2VHDL, formado pelos programa TABELA e o SEQ2VHDL.

CAPÍTULO 3

AMBIENTE TAB2VHDL (TABELA E SEQ2VHDL)

Descreve-se neste capítulo um método sistemático para projetos que a partir de uma descrição textual de uma máquina de estados finitos síncronas (1), máquinas de estados (circuitos seqüenciais ou FSM), origina-se uma segunda descrição (2), onde os elementos de memória são claramente identificáveis (*flip-flops*) e finalmente uma última descrição (3) em VHDL nos quais está presente uma entrada particular, denominada de relógio, que sincroniza os eventos aos quais a máquina esta sujeita, e pode-se gerar um circuito integrado como representado na Figura 15.

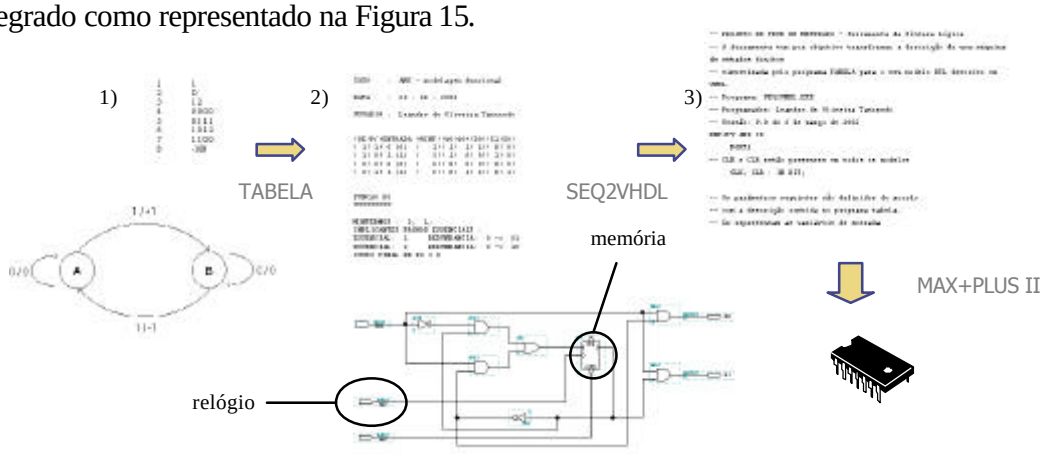

Figura 15 - Resumo dos arquivos textos gerados pelo ambiente TAB2VHDL

O comportamento de um circuito seqüencial pode ser descrito por um diagrama de estados. Tal diagrama define as transições entre os estados do sistema e suas saídas, quando submetidos às entradas.

3.1 PROGRAMA TABELA

O programa TABELA [DaS 89] elaborado pela UNICAMP gera a tabela de transição de uma máquina seqüencial a partir de seu diagrama de estados, e minimiza as funções de transições internas correspondentes aos elementos de memória utilizados e as funções de saída do circuito.

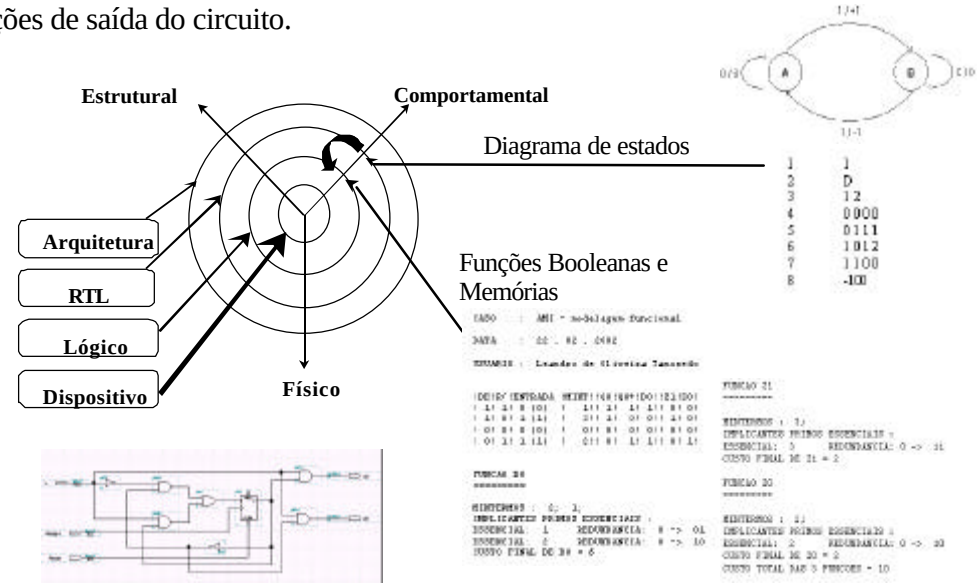

Figura 16 - Diagrama Y que representa os arquivos textos do Programa TABELA

Os dados solicitados pelo programa são:

- nome do dispositivo de saída para os resultados (não deve conter extensão);
- número de *flip-flops*;
- tipo de cada um dos *flip-flops* (podem ser do tipo D ou JK);
- número de variáveis de entradas seguido pelo número de variáveis de saída;
- a tabela de próximo estado deve ser descrita na seguinte forma:

  Estado atual Próximo estado Entrada Saída.

Os estados, as entradas e as saídas devem estar na forma decimal. As máquinas podem ser completa ou incompletamente especificadas, e devem estar na representação do modelo de Mealy.

O programa monta a tabela de transição armazenando-a no arquivo de saída. A partir desta tabela são obtidos os mintermos [20] e os *don't care states*[21] das funções internas (controle) de todos os *flip-flops* e da saída do circuito. Utilizando-se do algoritmo de Quine-McCluskey [McC 56] (Algoritmo de minimização de funções booleanas) estas funções são obtidas nas suas fórmulas mínimas.

O método de Quine-McCluskey é um método clássico que possui duas fases: a obtenção dos implicantes primos, e a cobertura irredutante, onde a partir do conjunto de implicantes primos são obtidos os implicantes essenciais para a realização da função. [DaS 89].

O diagrama de blocos do programa TABELA é apresentado na Figura 17.

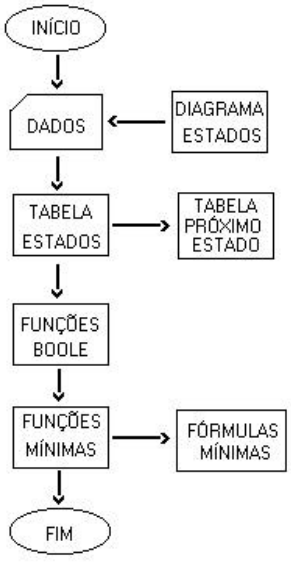

Figura 17 - Diagrama de blocos do programa TABELA

[20] [21] Ver glossário

Como exemplo da aplicação do programa TABELA descreve-se a seguir o projeto do código de linha AMI. Ele codifica os pulsos 0 em “0”, cuja alocação foi adotada como “00”, e os pulsos 1, alternadamente em pulsos “+1” e “-1” onde adotou-se respectivamente as alocações “01” e “10”. O objetivo do código AMI é eliminar o nível DC na linha de transmissão.

O projeto tem início na representação do funcionamento do sistema por meio do diagrama de transição de estado.

O diagrama de transição de estados para o código AMI é apresentado na Figura 18.

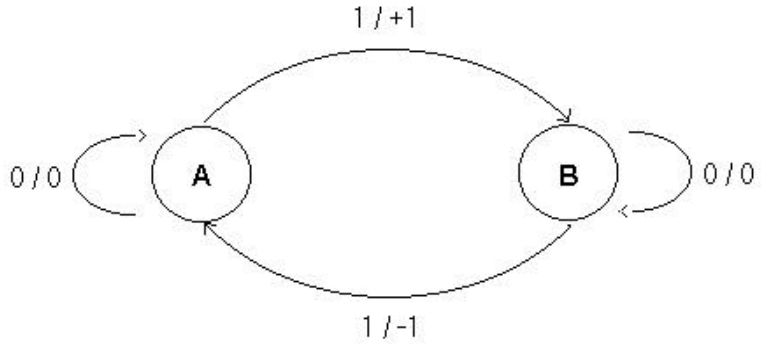

Figura 18 - Diagrama de estados do Código AMI

As informações contidas no diagrama de estado são transcritas por meio de notações convenientes para gerar um arquivo, com extensão “.TAB”, que é o arquivo de entrada para o programa TABELA.

Apresenta-se mais adiante, o arquivo AMI.TAB, conforme Figura 19 que descreve o diagrama de estado apresentado na Figura 18 para ser lido pelo programa.

```
1       1
2       D
3       1 2
4       0 0 0 0
5       0 1 1 1
6       1 0 1 2
7       1 1 0 0
8       -100
```

Figura 19 – Arquivo AMI.TAB de Entrada do Programa TABELA

A numeração na coluna, mais à esquerda, será utilizada para se referir às linhas do arquivo de descrição do diagrama de estados. Essa numeração não poderá estar inserida no arquivo AMI.TAB para o correto funcionamento do programa TABELA.

Na linha 1 especifica-se a quantidade de elemento(s) de memória(s). No caso necessita-se de somente um elemento de memória, visto que o sistema contém somente dois estados. Na linha imediatamente abaixo especifica-se o tipo de elemento de memória que será utilizada, no caso o *flip-flop* tipo D.

Na linha 3 especifica-se a quantidade de entradas e de saídas. O sistema projetado tem 1 porto de entrada e 2 portos de saídas.

Nas linhas subseqüentes especificam-se as transições dos estados e a saída gerada. A linha 4 é lida da seguinte forma, estando a máquina no estado atual ‘0’ ela transita para o estado ‘0’ quando tiver entrada ‘0’ e gerar saída ‘0’. Na linha seguinte, ao estar a máquina no estado atual ‘0’ ela transita para o estado ‘1’ quando tiver entrada ‘1’ e gera saída ‘1’.

Deve-se lembrar que na especificação está se utilizando a notação decimal. Como a saída tem dois bits a notação saída ‘1’ significa que o bit mais significativo assume o valor binário ‘0’ e o bit menos significativo o valor binário ‘1’. Observe que se utilizou a alocação “01” para representar um sinal com polaridade positiva “+1”.

O final da descrição é representado pela notação “-100”.

Ao executar o programa TABELA, utilizando o arquivo AMI.TAB como entrada, o arquivo gerado inicia-se com um cabeçalho que permite identificar o caso rodado, o usuário e a data. Posteriormente é apresentada a tabela de transição de estados da qual são extraídas as funções de controle dos elementos de memória e das saídas.

São apresentados todos os mintermos das funções booleanas e os respectivos implicantes primos. Por exemplo, a função D0 é composta pelos mintermos 2 e 1. Os implicantes primos geram função mínima representada por X’.Q + X.Q’. A função Z1 é composta pelo implicante X.Q e a função Z0 é composta pelo implicante X.Q’. O custo total para a implementação das três funções combinacionais é igual a 10, considerando-se como critério de custo a quantidade de entrada das portas lógicas AND e OR utilizadas.

O programa Tabela gera o arquivo de saída AMI.TXT apresentado a seguir:

```
CASO    :  AMI - modelagem funcional

DATA    :  22 . 02 . 2002

USUARIO :  Leandro de Oliveira Tancredo


!DE!P/!ENTRADA !MINT!!Q0!Q0+!D0!!Z1!Z0!
! 1! 1! 0 (0)  !   1!! 1!  1! 1!! 0! 0!
! 1! 0! 1 (1)  !   3!! 1!  0! 0!! 1! 0!
! 0! 0! 0 (0)  !   0!! 0!  0! 0!! 0! 0!
! 0! 1! 1 (1)  !   2!! 0!  1! 1!! 0! 1!


FUNCAO D0
=========

MINTERMOS :  2;  1;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  1     REDUNDANCIA:  0 ->  01
ESSENCIAL:  2     REDUNDANCIA:  0 ->  10
CUSTO FINAL DE D0 = 6

FUNCAO Z1
=========

MINTERMOS :  3;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  3      REDUNDANCIA: 0 ->  11
CUSTO FINAL DE Z1 = 2

FUNCAO Z0
=========

MINTERMOS :  2;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  2      REDUNDANCIA: 0 ->  10
CUSTO FINAL DE Z0 = 2
CUSTO TOTAL DAS 3 FUNCOES = 10
```

A Figura 20 apresenta o esquemático obtido pela interpretação do arquivo gerado pelo programa TABELA e a Figura 21 apresenta a simulação do circuito que implementa o código de linha AMI, utilizando-se o ambiente Max + Plus II da Altera.

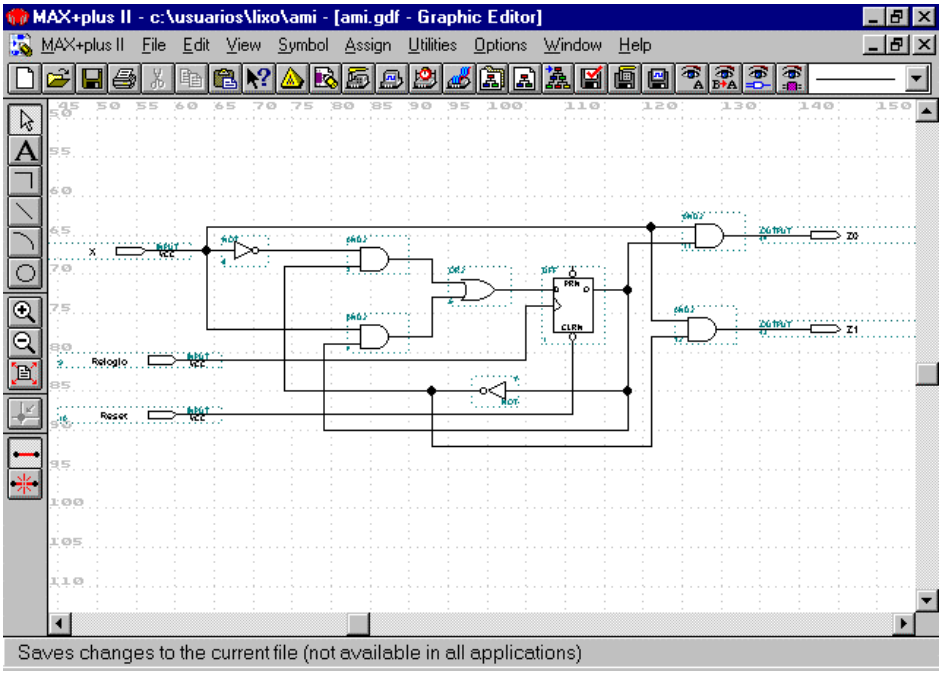

Figura 20 - Esquemático do circuito que implementa o código de linha AMI.

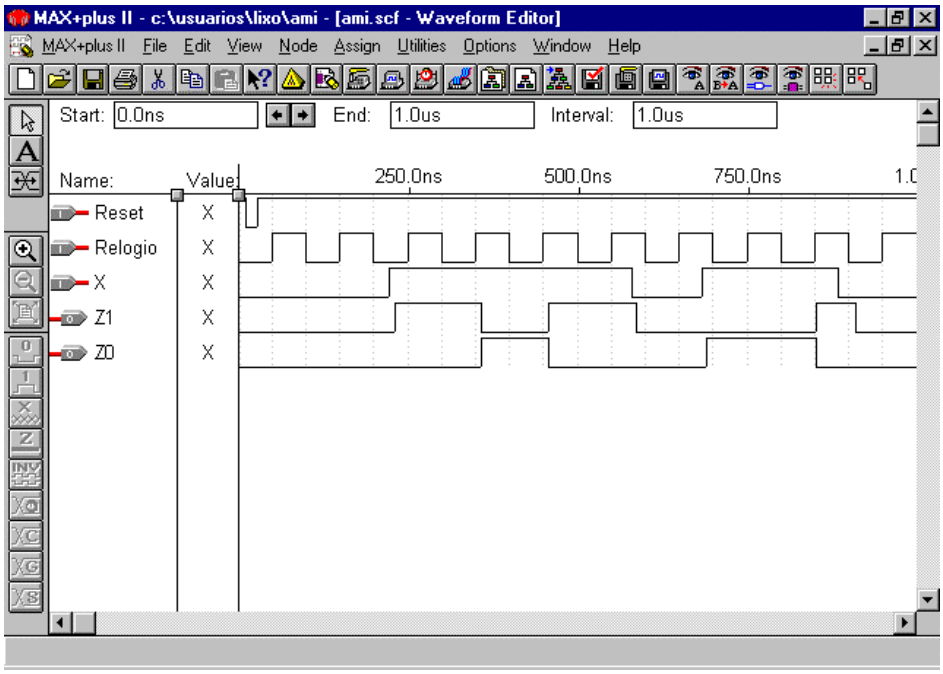

Figura 21 - Simulação do circuito apresentado na Figura 20.

A tecnologia de circuitos integrados vem avançando a cada dia com o objetivo de permitir que mais e mais componentes possam ser inseridos em uma simples pastilha de silício, assim os sistemas digitais também têm aumentado em complexidade. Como os sistemas digitais tem se tornado complexos, os detalhes de projeto de um sistema no nível de portas lógicas e *flip-flops* têm se tornado tediosa e passaram a consumir muito tempo do projetista. A descrição utilizando captura de esquemático está em desuso.

A linguagem de descrição de *hardware* permite que um sistema digital seja projetado e depurado em alto nível de abstração, antes de ser convertido para o nível de portas lógicas ou de *flip-flops*. O emprego de ferramentas de síntese, também denominadas de CAD, para este tipo de atividade está cada vez mais em evidência, conforme já mencionado.

Com o objetivo de automatizar os passos de projeto desenvolveu-se o programa TAB2VHDL (TABELA para VHDL) que recebe como entrada a saída gerada pelo programa Tabela, e cria um modelo funcional do circuito projetado na linguagem de descrição VHDL. Dessa forma, o arquivo contendo a descrição VHDL do circuito poderá ser sintetizado por ambientes de sínteses comerciais.

O ambiente que se está utilizando é o MAX + PLUS II da Altera [ALT 02], que está licenciado para trabalhar com o VHDL. Na realidade, trata-se de um pseudo VHDL, pois não disponibiliza todas as implementações especificadas no padrão IEEE. No ambiente disponível não se dispõe de um simulador VHDL e sim um sistema que dado o modelo descrito em VHDL, gera-se um *netlist* que é utilizado por um simulador de circuito digital.

Apresenta-se logo após o modelo VHDL funcional gerado pelo programa SEQ2VHDL que teve como arquivo de entrada a descrição gerada pelo programa TABELA.

3.2 PROGRAMA SEQ2VHDL

Neste item, conforme Figura **22**, utiliza-se a ferramenta SEQ2VHDL, ver Apêndice II, que recebe como entrada às funções booleanas e memórias do arquivo AMI.TXT (a saída gerada pelo programa TABELA) e cria um modelo funcional do circuito projetado na linguagem de descrição VHDL para o ambiente MAX+PLUS II.

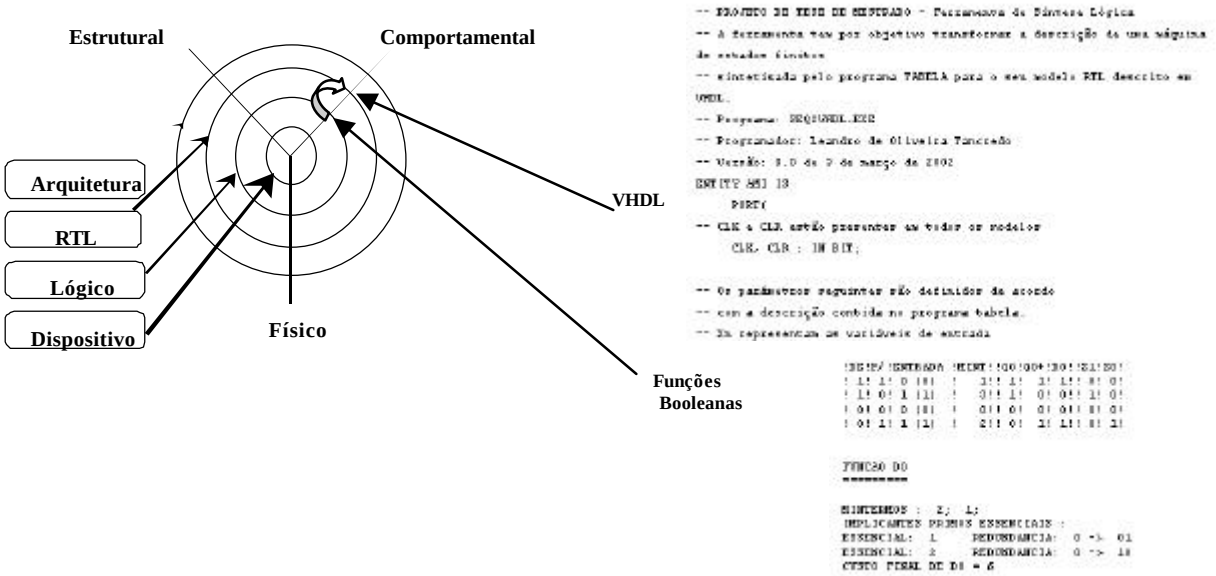

Figura 22 - Diagrama Y que representa a ferramenta SEQ2VHDL

A listagem abaixo descreve o arquivo AMI.VHD gerado pelo programa SEQ2VHDL:

```
-- PROJETO DE TESE DE MESTRADO - Ferramenta de Síntese Lógica
-- A ferramenta tem por objetivo transformar a descrição de uma máquina
de estados finitos
-- sintetizada pelo programa TABELA para o seu modelo RTL descrito em
VHDL.
-- Programa: SEQ2VHDL.EXE
-- Programador: Leandro de Oliveira Tancredo
-- Versão: 9.0 de 3 de março de 2002
ENTITY AMI IS
     PORT(
-- CLK e CLR estão presentes em todos os modelos
     CLK, CLR : IN BIT;
-- Os parâmetros seguintes são definidos de acordo
-- com a descrição contida no programa tabela.
-- Xn representam as variáveis de entrada
```

```
-- Qn representam as variáveis de estado
-- Zn representam as funções de saída
     X0     : IN BIT;
     Q0     : OUT BIT;
     Z0, Z1 : OUT BIT );
END AMI;
-- RTL e a designação para todas as arquiteturas
ARCHITECTURE RTL OF AMI IS
-- VEn são sinais auxiliares que assumem os mesmos valores
-- das variáveis de estado. Eles são utilizados para permitir
-- um melhor modelamento do sistema
SIGNAL  VE0: BIT;
-- Jn, Kn e Dn representam as funções de controle e sao definidas
-- no arquivo gerado pelo Programa TABELA
SIGNAL D0 : BIT;
 BEGIN
-- Nesta parte do modelo tem-se a descrição de cada um dos Flip-flops.
-- Deve-se observar que os sinais auxiliares são utilizados, sendo que
-- no final de cada processo seus valores são transferidos para as
-- variáveis de estados
-- Importante salientar que em relação ao software, existem duas
-- procedures,
-- uma que implementa o flip-flop D e outra que implementa o JK e estas
-- procedures
-- recebem os índices que representam o respectivo elemento de memória.
-- Tais índices estão definidos no arquivo gerado pelo programa TABELA
-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE0 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE0 <= D0;
    END IF;
    Q0 <= VE0;
END PROCESS;
-- Processos que implementam as funções combinacionais de controle
-- dos Elementos de memória e de Saídas.
  D0 <=  ( NOT(X0) AND (VE0)) OR ((X0) AND NOT(VE0));
  Z1 <=  ( (X0) AND (VE0));
  Z0 <=  ( (X0) AND NOT(VE0));
END RTL;
```

Pode-se notar que na entidade do modelo apresentado foram definidos os portos de entrada e saída e os sinais de sincronismo (CLK e CLR). O porto denominado Q_0 não é necessário para o perfeito funcionamento do sistema. Este foi inserido para permitir que o projetista observe, através de simulação, os estados pelos quais a máquina está transitando. Cabe salientar que o TAB2VHDL foi desenvolvido para ser empregado no ensino de sistemas digitais. Se a avaliação das transições de estado não forem necessárias, as variáveis de estado podem ser omitidas, conforme apresentado na Figura 21.

São definidos sinais auxiliares denominados V_{En} cujo objetivo é evitar que se gerem vários *drivers* para um mesmo sinal, o que exigiria o desenvolvimento de uma função de resolução. Esta função define qual será o valor do sinal em caso de conflitos.

É criado um processo para cada um dos *flip-flops* utilizados. Dois tipos de processos foram criados, um para modelar *flip-flop* D, sensível a transição de subida, e o outro para modelar *flip-flop* JK, sensível a transição de subida. Vários outros modelos de *flip-flop* poderiam ser definidos na ferramenta SEQ2VHDL.

Por fim, são definidas as funções de controle dos elementos de memória, D_0, e de saída, Z_1 e Z_2.

Estas funções foram obtidas do arquivo gerado pelo programa TABELA.

Na simulação apresentada na Figura 21 verifica-se o perfeito funcionamento do código AMI, cujo modelo foi gerado pelo programa SEQ2VHDL.

Vários códigos de linhas foram projetados, empregando três diferentes ambientes de projeto. No próximo capítulo, ao implementar os códigos, compara-se o projeto gerado pelo SEQ2VHDL com outras ferramentas.

CAPÍTULO 4

PROJETOS DE CÓDIGOS DE LINHA

Escolheu-se para estudo de casos a implementação de alguns chip-sets de códigos de transmissão digitais, utilizando-se duas ferramentas e o ambiente TAB2VHDL, este último apresentado no Capítulo 3. A maioria desses exemplos são padrões, todavia, com o passar do tempo surgem novos serviços, que obrigam a adaptação do código à nova tecnologia utilizada.

Logo, os projetos devem permitir uma maior flexibilidade de adaptação, sendo que uma solução adequada seria a implementação em *software.* Contrariamente, por estes códigos estarem localizados nas camadas inferiores da OSI/ISO, conforme visto no Capítulo 1, e por isso necessita-se de velocidade nas suas aplicações em tempo real, adotando-se a implementação em *hardware*.

Conforme apresentado no Capítulo 1, o desenvolvido de projetos eletrônicos tem impulsionado o setor de telecomunicações nos últimos anos, permitindo assim a criação de vários serviços digitais, como exemplo, temos várias tecnologias RDSI, HDSL, ADSL, VDSL e várias outras que estão surgindo no mercado. A utilização de componentes lógicos programáveis, como FPGAs para implementação de hardware, tem proporcionado uma melhoria no desempenho do circuito, ou seja, a redução de custos e uma integração física maior com relação aos projetos convencionais.

Os códigos de linha HDB3, HDB1, 2BQ1, 3B4B, MLT-3, AMI e algumas técnicas alternativas de detecção de sincronismo foram projetados utilizando-se as ferramentas comerciais aqui denominadas simplesmente de SC e AC, e os resultados foram comparados com os obtidos pelo ambiente TAB2VHDL.

Todos os códigos VHDL gerados foram sintetizados no ambiente de projeto Max + Plus II da altera.

Do arquivo com extensão RPT (Report) gerado pode-se extrair parâmetros como Circuitos Lógicos (LC) [22], Porcentagem Útil (% Útil) [23], Fan-In [24], Custo [25], Memória [26] para avaliar o desempenho do ambiente proposto.

4.1 CODIFICAÇÃO

Conforme PÉRICO [PÉR 95], a codificação de linha em um sistema de telecomunicação consiste no uso de códigos que atuam sobre um conjunto de bits que unidos formam originalmente uma palavra, essa palavra é alterada em uma nova palavra transformada, e transportada através de um meio de transmissão, de tal forma que sua principal característica é procurar preservar a faixa de freqüência original dos pulsos gerados, sendo que na recepção a palavra transformada é decodificada preservando-se assim a palavra original.

O objetivo da codificação é preservar o código nas seguintes situações: otimizar o espectro de energia do sinal transmitido, eliminar componentes do sinal com nível DC, prover mecanismo de preservação de sincronismo no próprio sinal.

4.1.1 CÓDIGO HDB3

O código HDB3 trabalha como o AMI, até a limitação de 3 zeros, a partir do quarto zero, obriga a introdução de falsos “um” na seqüência de zeros, cuja finalidade é limitar a seqüência de zeros.

Este código tem sido prescrito através de muitos anos pelo CCITT para sistemas de linha digitais primários de 2.048 Mbit/s.

[22] [23] [24] [25] [26] Ver Glossário

Para compreender algumas regras, deve-se definir os conceitos de bipolaridade e violação de uma cadeia de 4 zeros:

- **Sinal bipolar** é aquele que possui duas polaridades denominadas de B+ e B- e o estado zero, conforme Figura 23 .

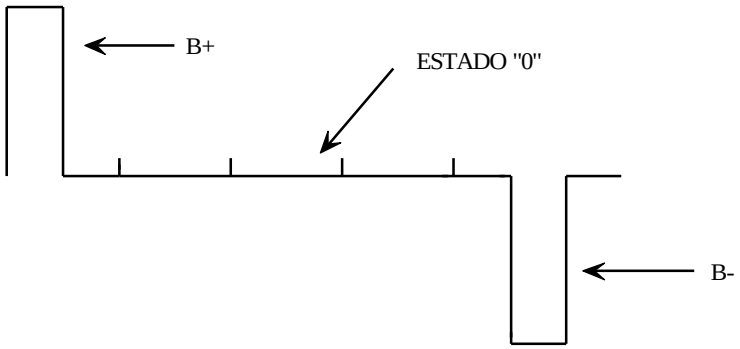

Figura 23 - Sinal Bipolar

- **Violação** da regra AMI, são pulsos que tem a mesma polaridade do pulso anterior, podendo ser positivas, chamadas de violação positivas (V+), ou negativas, chamadas violação negativa (V-), conforme Figura 24.

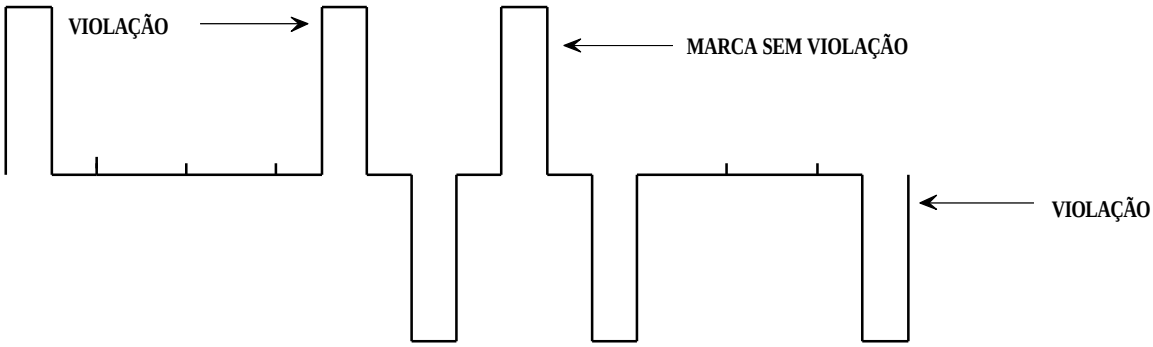

Figura 24 - Aspectos do sinal HDB-3 que envolvem violação.

As regras de codificação HDB-3, segundo a recomendação ITU-T de novembro de 1971, são:

1. O sinal HDB-3 é bipolar, pois produz três valores de intensidades que são representados por B+ e B- e 0;

2. Os sinais binários zeros são sempre codificados como zeros no sinal HDB-3 até a cadeia de três zeros consecutivos. Para seqüências a partir de quatro zeros consecutivos ou superior, aplica-se a regra 4;

3. Os uns do sinal binário são codificados alternadamente como B+ e B- no sinal HDB-3, tal como no código AMI. Violações da regra AMI só serão introduzidas quando uma seqüência de quatro espaços consecutivos aparecerem, conforme regra 4;

4. Na ocorrência de quatro zeros consecutivos, serão seqüencialmente numerados como 1º, 2º, 3º e 4º zero, deve-se portanto, proceder como nos sub-itens abaixo:

    a. Para o primeiro zero da seqüência será codificado como zero, se a marca precedente do sinal HDB-3 tiver polaridade oposta à violação precedente, será codificado como marca sem violação (B+ ou B-), caso as marcas de violação precedente tiverem a mesma polaridade;

    b. O segundo e terceiro zero da seqüência são codificados como zero;

    c. O último ou 4º espaço da seqüência é codificado como marca e a polaridade devem ser tal a regra AMI seja violada. Tais violações podem ser positivas ou negativas.

O fluxograma representado através da Figura 25 mostra os passos que deve-se tomar quando aparecem quatro espaços consecutivos, ou seja a regra 4.

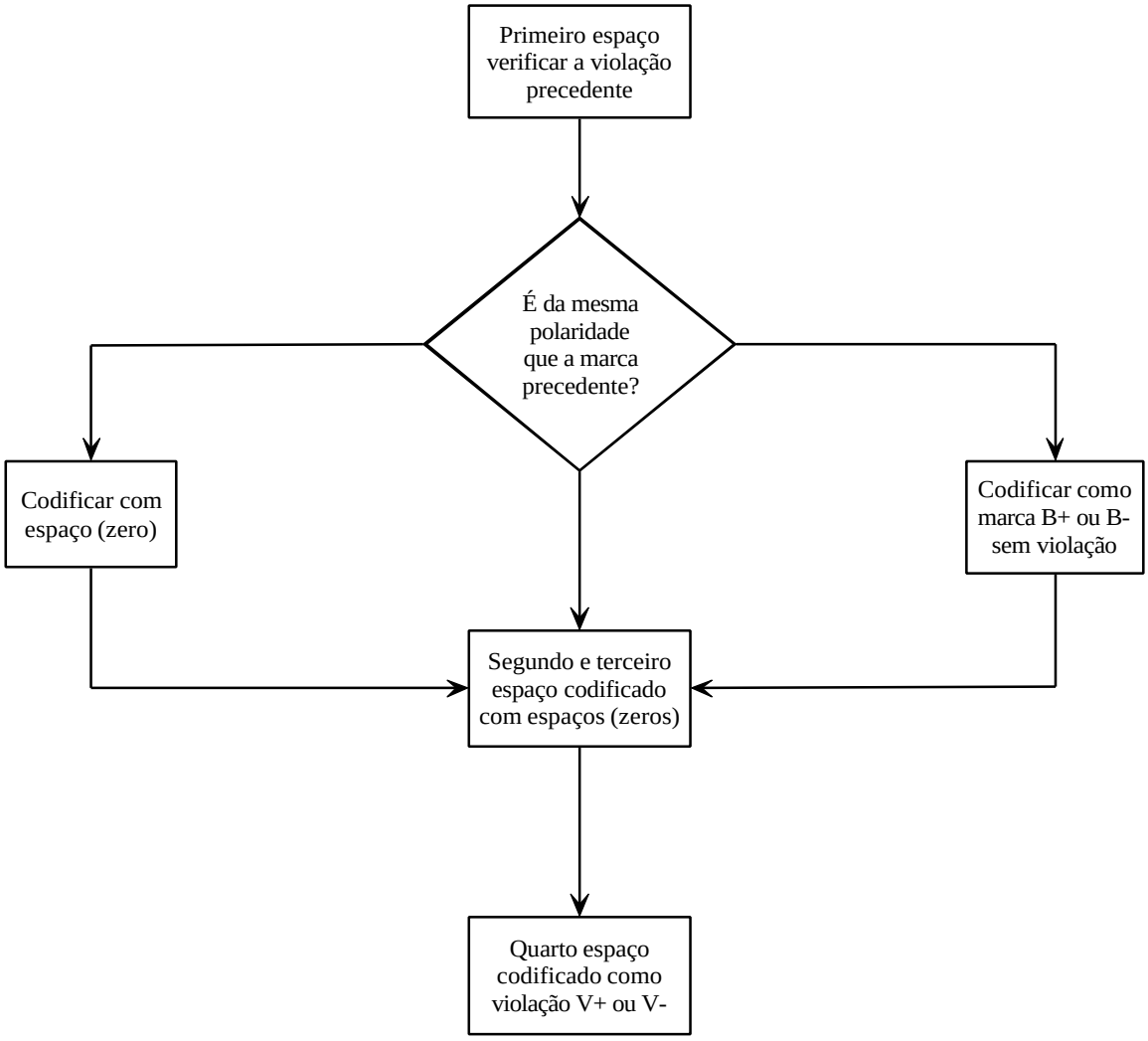

Figura 25 - Fluxograma da Regra 4 da codificação HDB-3

O diagrama de estado de uma máquina de Mealy, que gera o código de linha HDB3 é apresentado e descrito no trabalho de KEOGH [KOH 78].

O diagrama de estado ou a tabela de estado do código HDB3 será o ponto de partida para o projeto do código, existem muitos fabricantes comerciais que geram esse código de circuitos integrados como o CD22103A da RCA [INT 02b].

Como se pode notar anteriormente, o código HDB3 pode ser considerado como uma modificação do *alternate mark inverte* (AMI). Em AMI os dados 0 são transmitidos como 0 e os dados 1 como B+ (representado como +) ou B- (representando como -), como o AMI . O HDB3 busca corrigir o efeito de *strings* compridas formadas por mais de 4 "zeros", alterando a codificação de saída, para que não ocorra perda de informações de sincronização de relógio, substituindo quatros 0's sucessivos, por uma seqüência que pode ser B00V ou 000V, conforme regra 4, vista anteriormente.

Da descrição física do código, é possível observar o número de estados requerido. O codificador necessitou de 3 partes de memória para armazenar os dados prévios, ou seja, identificar quatro sucessivos O´s. Em seguida, um flip-flop é exigido para armazenar a polaridade do pulso bipolar (B) anterior (0 para + ou 1 para -), e finalmente, um outro flip-flop é requerido para armazenar a polaridade da Violação. Assim, o codificador requer um mínimo de $2^5 = 32$ estados.

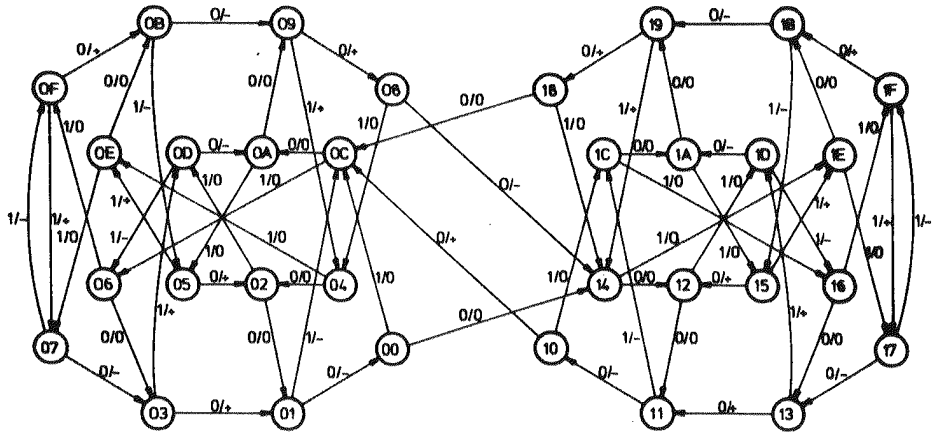

Figura 26 - Diagrama de estado do código HDB3 em Hexadecimal.

Empregando os processos convencionais para a construção e minimização de diagramas de estado para máquinas síncronas de estado finito, o diagrama com 32 estados do HDB3, pode ser visto na Figura 26. Este tem simetria aproximada sobre dois planos, um horizontal e o outro vertical.

Todos os dados representados por 1 causam uma transição através do plano horizontal, o qual corresponde à memória do codificador da polaridade do pulso bipolar previsto.

Os dois pares de estados 07 e 0F no lado esquerdo e 1F e 17, correspondentes aos pares de estados do lado direito do diagrama, são empregados na descrição AMI. Suas longas strings de sinais "uns" ocasionam a codificação alternada entre os estados + e -.

É importante notar que o diagrama de estado é um codificador que produz um símbolo de saída atrasada de três pulsos de relógio.

A codificação representada da máquina de estado está em hexadecimal. Partindo do diagrama de estado apresentado se projeta o código HDB3 no ambiente TAB2VHDL.

Transforma-se o diagrama de estado (de representação hexadecimal) para a representação decimal, para tanto utilizou-se o programa TABELA. A Tabela 7 apresenta essa transformação.

HEX	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**	**HEX**	**DEC**
00	0	**04**	4	**08**	8	**0C**	12
01	1	**05**	5	**09**	9	**0D**	13
02	2	**06**	6	**0A**	10	**0E**	14
03	3	**07**	7	**0B**	11	**0F**	15
10	16	**14**	20	**18**	24	**1C**	28
11	17	**15**	21	**19**	25	**1D**	29
12	18	**16**	22	**1A**	26	**1E**	30
13	19	**17**	23	**1B**	27	**1F**	31

Tabela 7 - Tabela de Conversão de Hexadecimal para Decimal

Para a conversão do nível de tensão + , – e 0 utilizou o binário 0 para representar o sinal + e o binário 1 para representarmos o sinal -, portanto +1 seria 01 e –1 seria 11 em binário, convertendo para decimal, tem-se 1 e 3.

As informações contidas no diagrama de estado são transcritas através de notação conveniente para gerar um arquivo, com extensão ".TAB", que é o arquivo de entrada para o programa TABELA.

Apresenta-se a seguir o arquivo HDB3.TAB, que descreve o diagrama de estado apresentado na Figura 26 .

```
1 5
2 D
3 D
4 D
5 D
6 D
7 1 2
8 0 20 0 0
9 0 12 1 0
10 1 0 0 3
11 1 12 1 3
12 2 1 0 0
13 2 13 1 0
14 3 1 0 1
15 3 13 1 1
16 4 2 0 0
17 4 14 1 0
18 5 2 0 1
19 5 14 1 1
20 6 3 0 0
21 6 15 1 0
22 7 3 0 3
23 7 15 1 3
24 8 20 0 3
25 8 4 1 0
26 9 8 0 1
27 9 4 1 1
28 10 9 0 0
29 10 5 1 0
30 11 9 0 3
31 11 5 1 3
32 12 10 0 0
33 12 6 1 0
34 13 10 0 3
35 13 6 1 3
36 14 11 0 0
37 14 7 1 0
38 15 11 0 1
39 15 7 1 1
40 16 12 0 1
41 16 28 1 0
42 17 16 0 3
43 17 28 1 3
44 18 17 0 0
45 18 29 1 0
46 19 17 0 1
47 19 29 1 1
48 20 18 0 0
49 20 30 1 0
50 21 18 0 1
51 21 30 1 1
52 22 19 0 0
53 22 31 1 0
54 23 19 0 3
55 23 31 1 3
56 24 12 0 0
57 24 20 1 0
58 25 24 0 1
59 25 20 1 1
60 26 25 0 0
61 26 21 1 0
62 27 25 0 3
63 27 21 1 3
64 28 26 0 0
65 28 22 1 0
66 29 26 0 3
67 29 22 1 3
68 30 27 0 0
69 30 23 1 0
70 31 27 0 1
71 31 23 1 1
72 -100
```

Podem ser observadas em três colunas. Em cada coluna há a numeração do arquivo de descrição do diagrama de estados. Essa numeração não poderá estar inserida no arquivo, deve-se descrever o diagrama em uma única coluna.

Na linha 1 especifica-se a quantidade de elementos de memórias. No caso necessita-se de 5 elementos de memória, visto que o sistema contém 32 estados. Na linha imediatamente abaixo, especifica-se o tipo de elemento de memória que será utilizada, no caso o *flip-flop* tipo D, nas linhas de 2 a 6.

Na linha 7 especifica-se a quantidade de entradas e de saídas. O sistema projetado tem 1 porto de entrada e 2 portos de saídas.

Nas linhas subseqüentes especificam-se as transições dos estados e a saída gerada. A linha 8 é lida da seguinte forma: Estando a máquina no estado atual '0' ela transita para o estado '2' quanto tiver entrada '0' e gera saída '0'. Na linha seguinte tem-se: Estando a máquina no estado atual '0' ela transita para o estado '1' quanto tiver entrada '1' e gera saída '1'.

Lembrando que na especificação utiliza-se a notação decimal. Como a saída tem três *bits*, a notação de saída '0' significa que o *bit* assume o valor binário '0'. Observe que se utilizou a alocação "01" para representar um sinal com polaridade positiva "+1", no qual representaremos pela notação "1" e a alocação "11" para representar um sinal com polaridade negativa "-1", no qual representaremos pela notação decimal "3"

O final da descrição é representado pela notação "-100".

No apêndice III tem-se o arquivo de saída HDB3.TXT, gerado pelo programa TABELA, logo após é feito um pequeno comentário sobre o arquivo, e finalmente o arquivo HDB3.VHD que gera o chip-set utilizando o MAXI+Plus II .

Comparou-se a ferramenta TAB2VHDL com as ferramentas SC, AC conforme Tabela 8.

HDB3	**SC**	**AC**	**TAB2VHDL**
CÉLULAS LÓGICAS (LC)	7	15	7
% UTIL	21	46	21
FAN-IN (porta)	34	101	39
CUSTO (porta)	99	301	50
TEMPO (s)	11	06	03
MEMÓRIA (Kb)	4,492	4,162	2,363

Tabela 8 - Comparação entre o SC, AC e o TAB2VHDL para o código HDB3

4.1.2 CÓDIGO DE LINHA MLT-3

O MLT-3 (do inglês Multi-Level Transmission-3) é um código de linha empregado para melhorar a eficiência de uma rede FDDI (do inglês Fiber Distributed Data Interface) utilizando par trançado para transmissões 100 BASE-TXmode [EVE 98].

No diagrama de estado representado através da ferramenta SC pode-se notar que a tensão varia entre +V, -V e 0, com isso, provocando um equilíbrio entre as tensões.

A descrição do projeto utilizado no TAB2VHDL demostrado logo abaixo, visa desprezar a primeira coluna, pois refere-se a linha da listagem, assim utilizando-se 2 flip-flop, definidos na linha 1, pois tem-se 4 estados, e dois flip-flop tipo D.

A seguir na linha 4 especifica-se que o código gerado apresenta 1 porto de entrada e dois de saída, e da linha 5 a 12, utilizou-se a alocação “01” para representar um sinal com polaridade positiva “+V”, no qual representa-se pela notação “1” e a alocação “11” para designar um sinal com polaridade negativa “-V”, que se representa-se pela notação decimal “3”.

```
1 2
2 D
3 D
4 1 2
5 1 1 0 1
6 1 0 1 0
7 0 0 0 0
8 0 3 1 3
9 3 3 0 3
10 3 2 1 0
11 2 2 0 0
12 2 1 1 1
13 -100
```

Logo a seguir descreve-se o diagrama de estado representado a partir da ferramenta SC, conforme Figura 27:

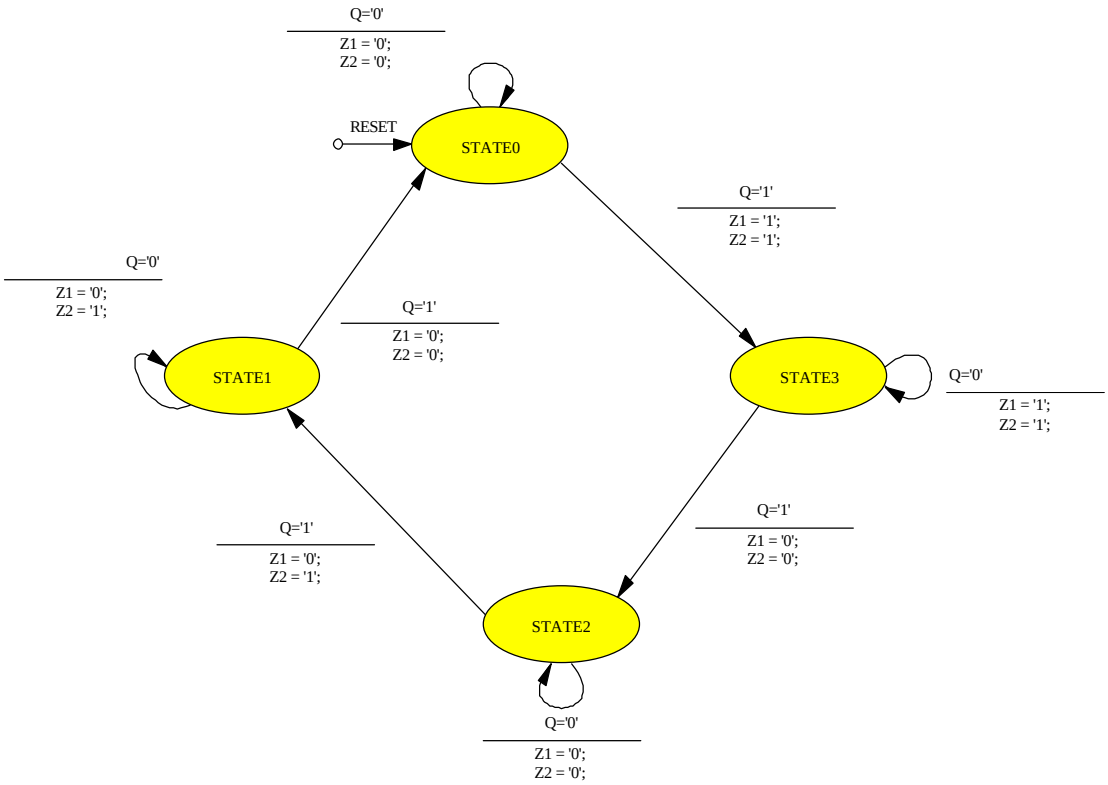

Figura 27 - Representação do Diagrama de Estado do código MLT-3

Posteriormente demonstra-se o comportamento do código MLT-3 ao utilizar a ferramenta MAX+Plus II, a partir do VHDL gerado pelo SC, AC e TAB2VHDL representado pela Tabela 9 :

MLT-3	SC	AC	TAB2VHDL
CÉLULAS LÓGICAS (LC)	6	4	4
% UTIL	18	12	12
FAN-IN	24	13	14
CUSTO (porta)	24	13	14
TEMPO (s)	03	02	02
MEMÓRIA (Kb)	2,963	3,025	2,912

Tabela 9 – Comparação do código de linha MLT3 utilizando SC, AC e TAB2VHDL

4.1.3 CÓDIGO AMI

O código AMI, apresentado no Capítulo 4, transforma a informação digital em um sinal ternário onde os "Zeros" são codificados pelo nível de tensão 0, enquanto que os "Uns" são alternadamente codificados por dois níveis de tensão simétricos +V e –V. Logo em seguida transformamos os sinais em binários para que possam ser processados, conforme o Capítulo 3.

A Tabela 10 que compara as características do código AMI ao utilizar-se o SC, AC e o TAB2VHDL:

AMI	SC	AC	TAB2VHDL
CÉLULAS LÓGICAS (LC)	3	3	3
% UTIL	9	4	9
FAN-IN (porta)	8	6	8
CUSTO (porta)	1	1	2
TEMPO (s)	01	04	02
MEMÓRIA (Kb)	3,032	3,396	2, 324

Tabela 10 - Comparação entre SC, AC e TAB2VHDL na criação do código AMI.

Este código, apesar de simples em sua descrição não pôde ser sintetizado diretamente pelo ambiente Altera com a descrição VHDL gerada pela ferramenta AC.

Devido a isso, foi necessário o emprego de um programa de otimização denominado Leonardo Spectrum para, a partir do arquivo com extensão EDIF correspondente, ser gerado uma descrição sintetizável em FPGA.

Sendo assim, os parâmetros apresentados na Tabela 10, referentes à ferramenta AC, com relação ao código AMI, é resultado de um pré-processo de otimização, o que invalida sua comparação com as outras duas ferramentas. Afinal, não foi objetivo desse estudo abordar a ferramenta Leonardo Spectrum. Esta foi utilizada somente para permitir gerar a síntese do modelo do código AMI desenvolvido pelo AC. Para todos os outros códigos foi possível efetuar a síntese dos modelos VHDL obtidos diretamente pela ferramenta AC.

Um aspecto importante que foi observado, e pode ser demonstrado analisando o arquivo report das sínteses geradas, é que no caso do ambiente TAB2VHDL, na grande maioria das vezes, as funções de controle dos elementos de memórias, funções booleanas especificadas a dois níveis de portas lógicas (soma de produto), facilitou muito o processo de otimização. Houve caso que a função tornou-se desnecessária com a substituição apropriada do elemento de memória (*flip-flop*), simplificando ainda mais o custo de implementação do circuito.

Como exemplo tem-se a função de controle do *flip-flop* tipo D e as funções de saída Z0 e Z1 gerados pelo TAB2VHDL para o código AMI, as quais são apresentadas a seguir:

D0 = X'.Q + X.Q'

Z0 = X.Q'

Z1 = X.Q

O esquemático do circuito de controle do *Flip-Flop* tipo D é apresentado na Figura 28 e a simulação apresentada Figura 29.

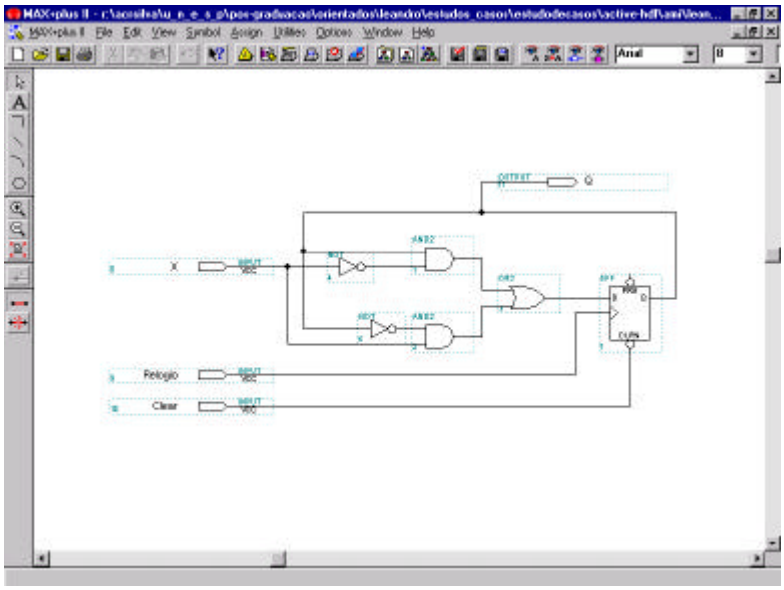

Figura 28 - Esquemático do circuito para a função D0 = X'.Q + X.Q'

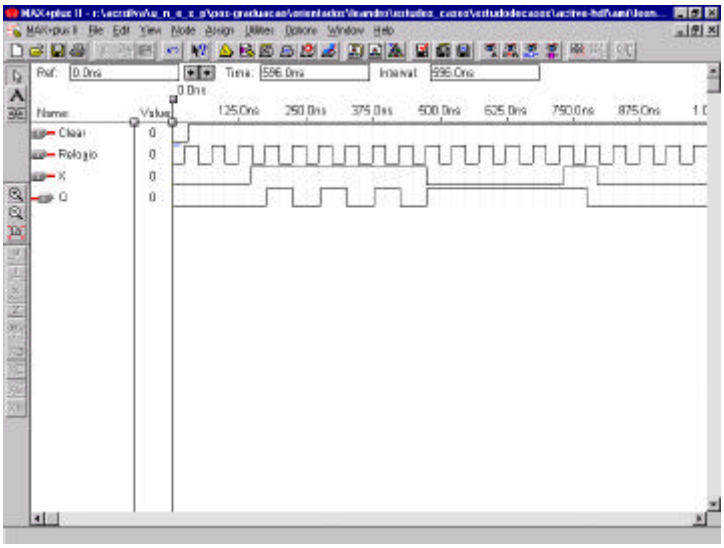

Figura 29 - Simulação do circuito apresentado na Figura 28.

Pode-se notar que o comportamento do circuito corresponde ao do flip-flop tipo T, ou seja, quando a entrada X é mantida em nível lógico “1” o estado do flip-flop, representado pela variável de estado Q, é complementado a cada pulso de relógio e quando a entrada X é mantida em nível lógico “0” o estado do flip-flop se mantém.

Quando esse modelo do circuito apresentado na Figura 28 foi sintetizado no ambiente Altera todo o circuito foi substituído por um único flip-flop T, sem nenhum circuito de controle, diminuindo, dessa forma o custo de implementação do código. O custo de implementação que inicialmente era 10, tornou-se 4 após a síntese, ou seja 60% menos.

Essa característica não foi observada na síntese efetuada com os modelos VHDL gerados pelas outras duas ferramentas. Isto justifica o fato de que o TAB2VHDL sempre obteve custo de implementação menor ou igual ao das duas outras ferramentas.

4.1.4 CÓDIGO 2B1Q

O código de linha 2B1Q transforma 2 bits em símbolos quaternários, o que pode ser visualizado com a apresentação da Tabela 11:

PRIMEIRO BIT	SEGUNDO BIT	QUATERNÁRIO
1	0	+3
1	1	+1
0	1	-1
0	0	-3

Tabela 11 - Tabela Verdade do código 2B1Q

A função principal deste código é efetuar a conversão de transmissão a 4 fios para transmissão a 2 fios e vice-versa.

Apresenta-se a seguir, a Tabela 12 comparando as características das ferramentas AC, SC e TAB2VHDL, utilizadas para projetar o código 2BQ1:

COD2BQ1	SC	AC	TAB2VHDL
CÉLULAS LÓGICAS (LC)	7	5	5
% UTIL	21	15	15
FAN-IN (porta)	34	18	10
CUSTO (porta)	43	22	9
TEMPO (s)	03	08	03
MEMÓRIA (Kb)	3,100	2,366	2,417

Tabela 12 - Comparação do projeto 2BQ1 ao utilizar o SC, AC e TAB2VHDL

4.1.5 CÓDIGO HDB1

A finalidade dos códigos HDBn é limitar o número de zeros em uma seqüência, pois o mesmo pode reduzir-se ao componente espectral da freqüência de um relógio a um valor muito pequeno, tornando-se portanto impossível ou difícil a recuperação das informações.

O código HDB1 (do inglês *High Density Bipolar of order 1 code*) foi o código antecessor dos códigos da família HDB, sendo praticamente o mesmo que o AMI, com exceção das transmissões em 0 (zeros). No caso da ocorrência de zeros isolados, Isto é, “0” entre dois “uns” consecutivos, a transmissão se faz da mesma forma que o AMI. O “zero” isolado é transmitido como 0.

Caso ocorram dois ou mais “0” consecutivos, para cada par de “0” transmite-se como +1+1, se o último sinal antecessor ao par apresentar polaridade –1, caso ocorra do último sinal apresentar polaridade +1 ou será transmitido como –1-1, e assim alternadamente.

A substituição desses zeros aumenta o número médio de pulsos de linha, facilitando a recuperação do “relógio” e do sincronismo na recepção.

O diagrama representado a partir do programa AC é observado na Figura 30:

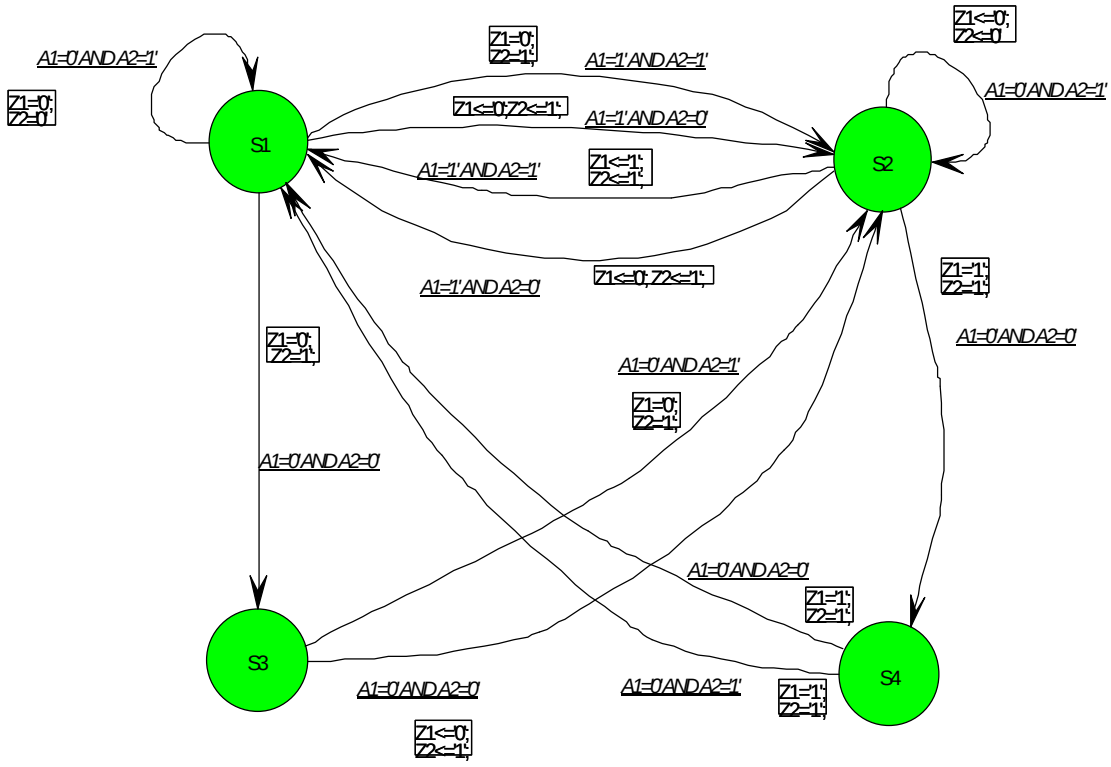

Figura 30 - Diagrama de Estado do Código HDB1

A linguagem de descrição do código HDB1 é referenciada, em seguida, onde os códigos binários são transformados em decimal para que se possa utilizar a ferramenta TAB2VHDL.

Logo após apresenta-se a Listagem do arquivo HDB1.TXT

```
2
D
D
1 2
0 1 1 2
0 2 0 2
1 0 1 1
1 3 0 1
2 0 0 1
2 1 0 1
2 1 1 0
3 0 0 2
3 0 1 0
-100
```

Ao utilizar o SC, AC e o TAB2VHDL, gerou-se o código do HDB1 em VHDL e usou-se o MAX+Plus II para chegar a seguinte tabela:

	SC	**AC**	**TAB2VHDL**
CÉLULAS LÓGICAS (LC)	6	4	4
% UTIL	18	12	12
FAN-IN (porta)	33	15	17
CUSTO (porta)	35	18	11
TEMPO (s)	02	03	06
MEMÓRIA (Kb)	3,020	3,095	2,925

Tabela 13 – Comparação do código HDB1 utilizando SC, AC e TAB2VHDL

Outros códigos utilizados no setor de telecomunicações como o Identificador da seqüência 10010 (S10010) , CCS, K em N e o Transu foram testados, conforme resultados apresentados na Tabela 14.

4.2 COMPARAÇÃO DOS RESULTADOS

Conforme o item anterior a interpretação do diagrama de estado foi inicialmente feita pelos ambientes AC, SC e TAB2VHDL (representados na tabela respectivamente como AC, SC e TAB), e posteriormente codificados em VHDL.

Em seguida, utilizou-se o *software* MAX + Plus II, para sintetizar o código VHDL, sendo que no caso do código gerado pelo ambiente TAB2VHDL ocorreu uma reotimização do circuito.

Notou-se que o dispositivo escolhido como conseqüência da complexidade do circuito foi o EPM 7032LC44-6 da Altera para todos os códigos utilizados.

Na Tabela 14 apresenta-se os resultados gerais obtido a partir da implementação feita pelo MAX + Plus II.

	HDB3			**HDB1**			**2BQ1**			**3B4B**		
	SC	**AC**	**TAB**	**SC**	**AC**	**TAB**	**SC**	**AC**	**TAB**	**SC**	**AC**	**TAB**
LC	7	15	7	6	4	4	7	5	5	12	7	7
% UTIL	21	46	21	18	12	12	21	15	15	37	21	21
FAN-IN	34	101	39	33	15	17	34	18	10	88	52	32
CUSTO	99	301	50	35	18	11	43	22	9	181	183	44
TEMPO (s)	11	6	3	2	3	6	3	8	3	8	10	2
MEMÓRIA (KB)	4,492	4,162	2,363	3,02	3,095	2,925	3,1	2,366	2,417	2,594	2,757	2,216

	MLT3			**AMI**			**S10010** [27]			**CCS** [28]		
	SC	**AC**	**TAB**	**SC**	**AC**	**TAB**	**SC**	**AC**	**TAB**	**SC**	**AC**	**TAB**
LC	6	4	4	3	3	3	8	4	4	8	4	4
% UTIL	18	12	12	9	4	9	25	12	12	25	12	12
FAN-IN	24	13	14	8	6	8	37	19	21	40	19	22
CUSTO	24	13	14	1	1	2	15	24	27	18	28	27
TEMPO (s)	3	2	2	1	4	2	1	0	1	0	4	1
MEMÓRIA (KB)	2,963	3,025	2,912	3,032	3,396	2,324	2,63	3,432	3,136	3,333	2,994	3,438

	KEMN [29]			**TRANSU** [30]		
	SC	**AC**	**TAB**	**SC**	**AC**	**TAB**
LC	14	5	5	9	4	4
% UTIL	43	15	15	28	12	12
FAN-IN	64	29	33	44	17	22
CUSTO	29	80	63	23	21	33
TEMPO (s)	1	6	1	1	6	1
MEMÓRIA (KB)	3,191	3,157	2,941	3,352	2,934	3,065

Tabela 14 - Resultado final da implementação

[27, 28, 29 e 30] Ver glossário

A partir da Figura 31, pode-se observar que na maioria dos códigos analisados a quantidade de memória alocada pelo TAB2VHDL (posta em destaque como uma linha contínua em negrito), na maioria dos códigos testados, o desempenho foi menor ou igual comparado com outras ferramentas, com exceção dos códigos S10010 e CCS. Relevando a diferença ocorrida com relação ao código HDB3 que possui uma quantidade reduzida de memória, quando comparada com os outros códigos.

Dessa forma, as ferramentas apresentaram um comportamento parecido com relação aos códigos HDB1 e MLT-3, ou seja, os picos de alocação de memória foram aproximadamente iguais.

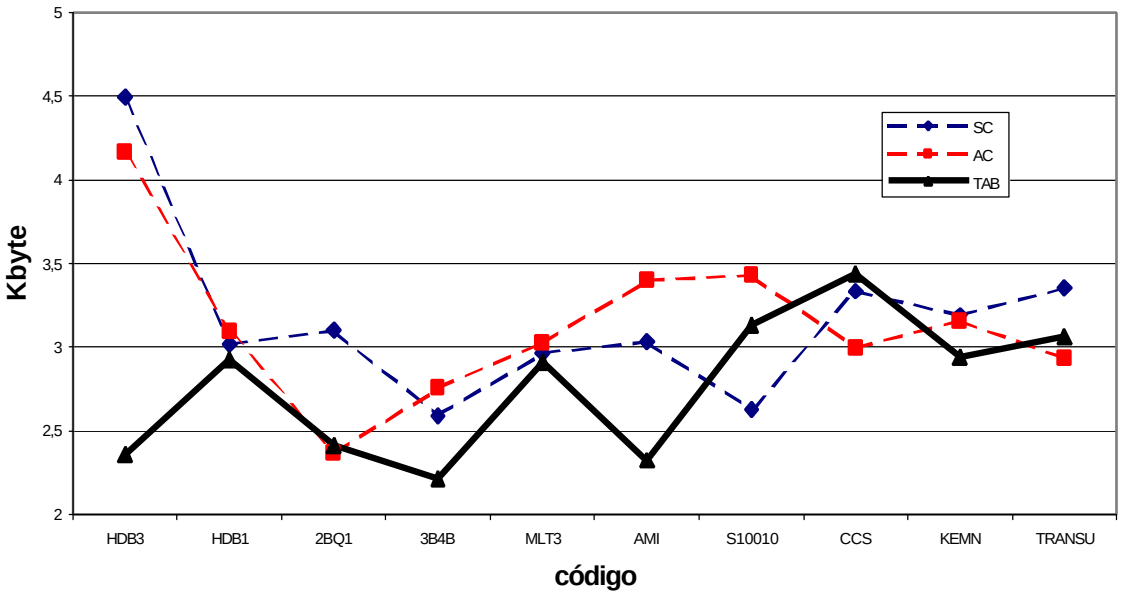

Figura 31 - Gráfico comparando a alocação de pico de memória

Conforme a Figura 32, ao comparar a variável tempo de processamento, verificou-se que a execução dos códigos gerados pelo TAB2VHDL com relação as outras ferramentas foi inferior, o mesmo não pode se afirmar com o ocorrido com o HDB1. Em alguns casos podemos observar que o tempo foi desprezível, ou seja casos que o tempo de execução foi inferior a 1 segundo.

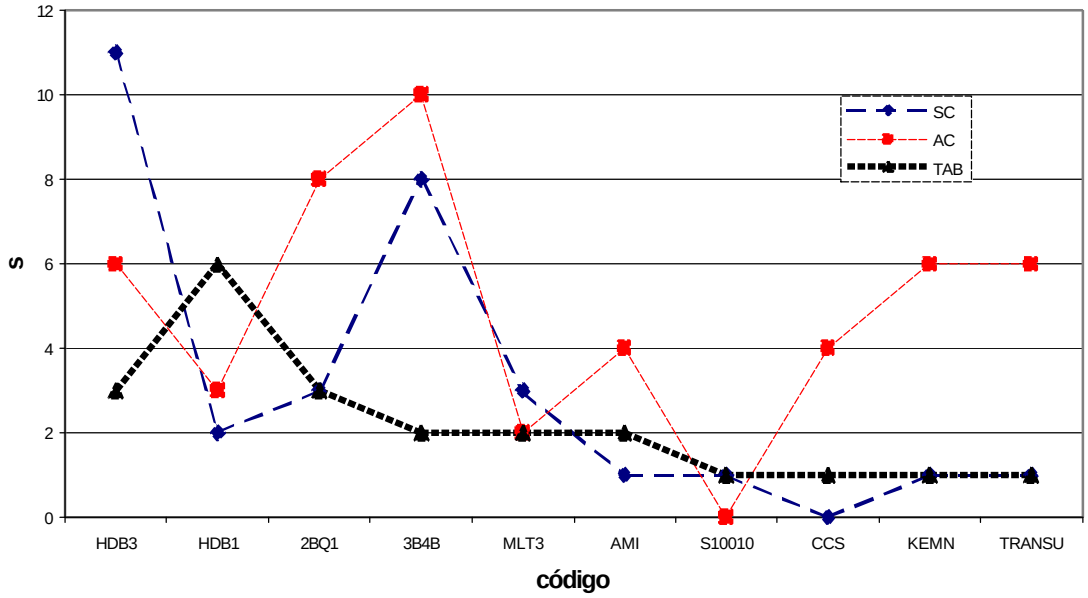

Figura 32 - Gráfico comparando o tempo de execução do código.

Quando se compara o custo, ou seja, a somatória aritmética dos números de terminais de entrada das portas lógicas AND e OR, usadas na representação de uma função, nota-se a eficiência do TAB2VHDL, com relação às outras ferramentas, Figura 33.

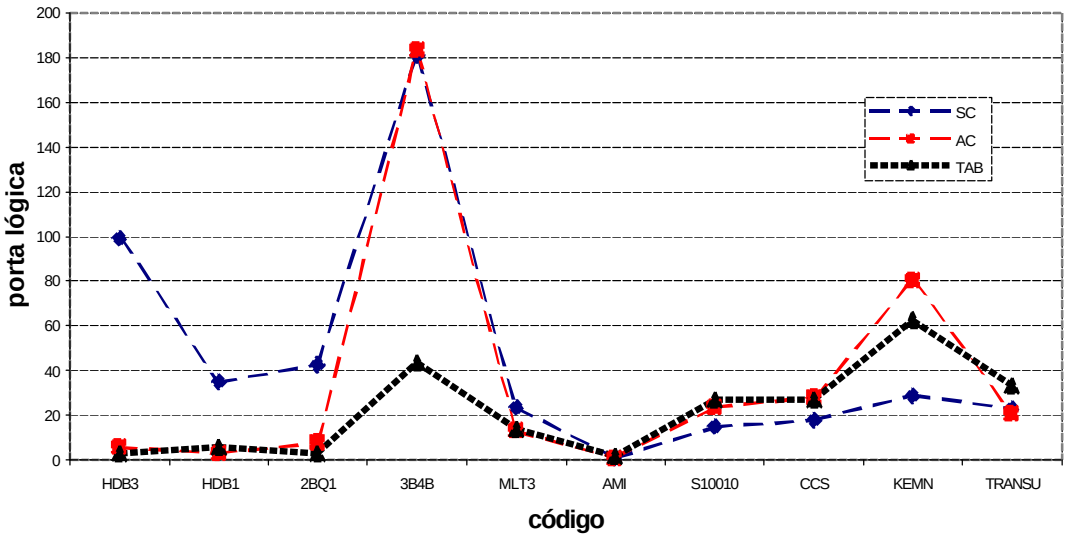

Figura 33 - Gráfico comparativo entre os códigos e o números de portas lógicas

Com relação à característica Fan-in o TAB2VHDL possui um número de componentes menor ou igual as outras ferramentas que serviram de comparação, conforme Figura 34.

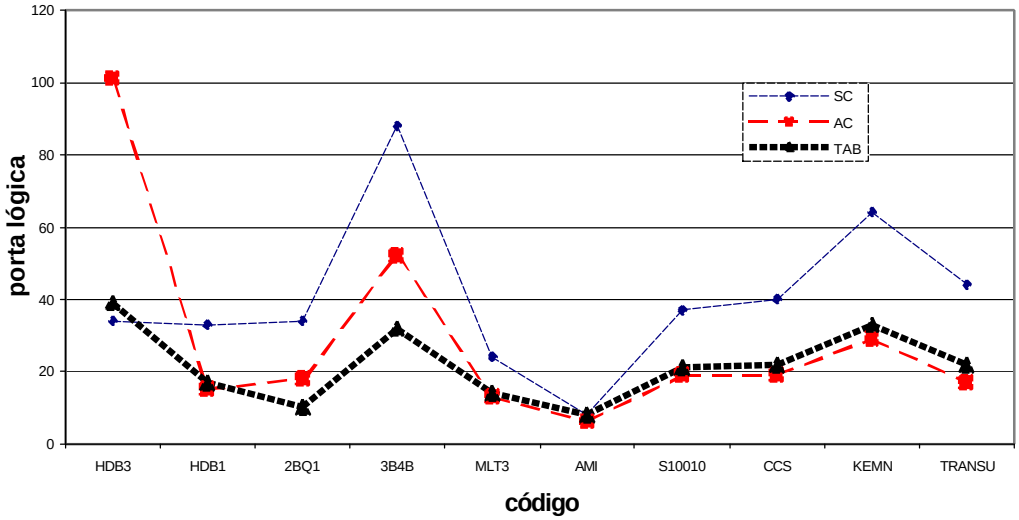

Figura 34 - Gráfico que compara o número de fan-in entre as ferramentas.

Comparando a porcentagem útil de silício do circuito integrado, na Figura 35 percebe-se que o TAB2VHDL comparado com o AC, teve valores muito parecidos com exceção do código HDB3, onde o AC apresentou uma porcentagem maior. O SC teve valores superiores à ferramenta TAB2VHDL, o mesmo não pode-se afirmar para o código HDB3, pois o TAB e o AC contém valores equivalentes.

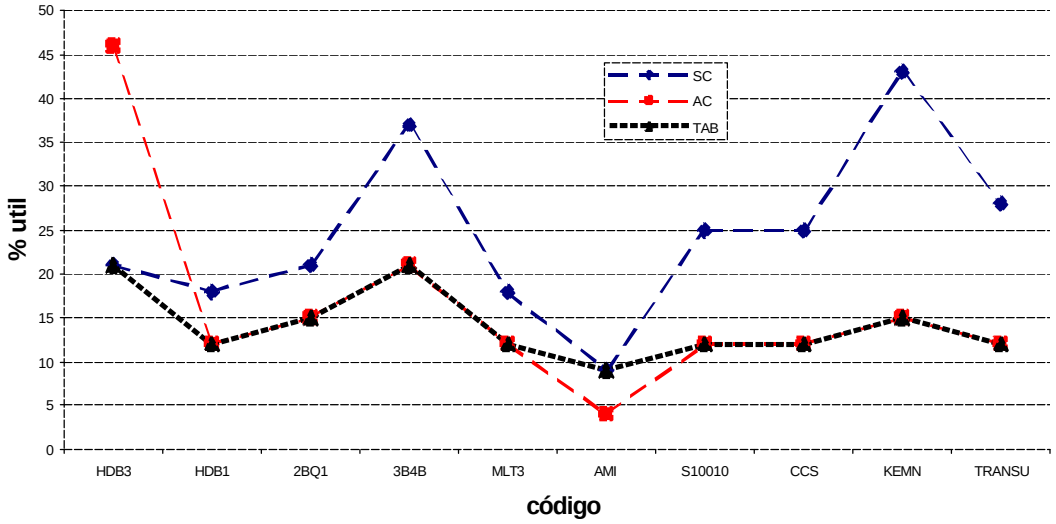

Figura 35 - Comparação entre a porcentagem útil de utilização e os códigos.

Contudo, podemos concluir através da análise dos gráficos apresentados, que o ambiente TAB2VHDL obteve um desempenho notável comparado com as ferramentas avaliadas.

Outro resultado importante, foi obtido ao comparar o custo do código gerado pelo TAB2VHDL ao utilizar como elemento de memória JK (TAB JK) ou D (TAB D), observa-se conforme Tabela 15 que em algumas situações o JK é a melhor solução como no 2BQ1 e 3B4B e MLT3 e que em outras o D foi a melhor solução como na AMI, S10010 E CCS.

	HDB3		**HDB1**		**2BQ1**		**3B4B**		**MLT3**		**AMI**	
	TAB D	**TAB JK**	**TAB D**	**TAB JK**	**TAB D**	**TAB JK**	**TAB D**	**TAB JK**	**TAB D**	**TAB JK**	**TAB D**	**TAB JK**
CUSTO	50	50	11	11	9	6	44	43	14	7	2	5

	S10010		**CCS**	
	TAB D	**TAB JK**	**TAB D**	**TAB JK**
CUSTO	24	29	27	28

Tabela 15 - Comparação utilizando elemento de memória JK e D no TAB2VHDL

O fato do ambiente TAB2VHDL, o projetista poder selecionar o elemento de memória convencional e optar pela seleção da alocação de estado, o torna-o flexível comparado com os outros dois ambientes avaliados.

É sabido que a alocação de estado e o elemento de memória empregado, está intimamente relacionado com a complexidade do circuito gerado.

CONCLUSÃO

Nesta dissertação mostrou-se o cenário passado, presente e procura-se antecipar o futuro dos serviços de telecomunicações, nota-se o surgimento de novos serviços, que deverão em um curto espaço de tempo, serem projetados e adaptados, impulsionados pelos grandes resultados apresentados pelas tecnologias de telecomunicações vista no capítulo 1, como exemplo o HDSL que a partir de um único cabo de par trançado se compartilha 30 canais de voz em um link de 2 Mbps. Portanto a necessidade de novas ferramentas de síntese como o TAB2VHDL, para otimizar *chip-sets* envolvidos nestes tipos de projetos, passa a ser fundamental.

Contudo, podemos observar que é necessário um estudo do estado da arte dos modelos, ferramentas e tipos de representações, para que se possa relacionar os aspectos envolvidos em um determinado projeto.

Assim, o estudo mostrou que é impossível ter uma única ferramenta, linguagem e modelo que substituísse todas as outras, entretanto, se existisse, ficaria muito complicado trabalhar, tendo como conseqüência o alto grau de complexidade que envolveria o projeto.

A partir dos estudos dos casos testou-se o TAB2VHDL com outras ferramentas comerciais, e comparando os resultados apresentados notou-se a eficiência dessa ferramenta que na maioria das experiências obtidas, apresentou-se de forma satisfatória com relação às outras ferramentas, conforme pode-se observar no Capítulo 4.

O projeto de sistemas digitais é uma área de pesquisa em constante evolução. O aumento contínuo da complexidade dos sistemas, cria a necessidade de se rever as técnicas de síntese e otimização já desenvolvidas.

O emprego cada vez maior das FPGAs tem motivado o desenvolvimento de ferramentas de síntese automatizada.

Foi neste contexto que se inseriu esta pesquisa, sempre visando a realidade, as necessidades do ensino da prática de projetos digitais.

A dissertação de mestrado originou um artigo que será apresentado durante o ICECE 2003 – International Conference on Engineering and Computer Education que acontecerá no mês de Março, em Santos.

O ambiente TAB2VHDL foi detalhado sistematicamente no capítulo 3, implementou-se vários códigos utilizados em telecomunicações no capítulo 4, documentou-se o ambiente utilizando análise estrutural no apêndice I, listou-se o programa SEQ2VHDL na linguagem C no apêndice II e finalmente no apêndice III pode-se comparar a listagem do TAB2VHDL e de outras ferramentas comerciais na VHDL, desvendando portanto, um ambiente de síntese, propiciando que novos trabalhos sejam gerados ou anexados.

Trabalhos futuros como: criar módulos para descrição em Verilog-HDL, outros tipos de *flip-flops*, novas bibliotecas padrões, e a substituição por outros métodos de minimização e a criação de um ambiente gráfico.

BIBLIOGRAFIA

[ALT 02] ALTERA. **Max+Plus II AHDL**. Disponível em: <www.altera.com/literature/lit-manual.html> em: 09/08/2002.

[ALT 01] ALTERA. **Altera Digital Library**. Disponível em: <www.altera.com> Acesso em: 09/12/2001.

[AMY 00] AMYOUT D. et alli. **An Evaluation of Scenario Notations for Telecommunication Systems Development**. School of Information Technology and Engineering, University of Ottawa, 20pp. 2000.

[ARM 89] ARMSTRONG, James R. **Chip-level modeling with VHDL.** 10 ed. New Jersey: Prentice Hall, 1989.148p.

[AT& 02] AT&T. **FSM Library**. Disponível em: <www.research.att.com/sw/tools/fsm> Acesso em: 02/04/02

[BON 95] BONATTI, Ivanil S**.. Stoht – An SDL-to-Hardware Translator**. Asia Pacific Design Automation Conference, Mukuhari Japan, 33-36pp. 1995

[BOR 99] BORTOLLI, Edson Jr. et alli: **A transmissão Digital na Rede Metálica de Assinantes**. Campinas: CPqD, 235f., 1999.

[CAL 98] CALAZANZ, Ney Laert Vilar. **Projeto lógico automatizado de sistemas digitais seqüenciais.**. Porto Alegre: Instituto de Informática. PUC RS, 1-41pp., 1998.

[CYP 02] CYPRESS. **An Introduction to Active-HDL.** FSM.pdf**.** São José, 28 de Janeiro de 2002. 1 arquivo (97 Kbytes). Disquete 3 ½. Acrobat Reader 5.0

[CON 99] CONSTANTINIDES, George A., et alli. **Synthia of Interacting Automata targeting LUT-base FPGAs**. Dept. of Electrical and Eletronic Engineering, Imperial College, London, UK, 10f., 1999.

[DaS 89] Da SILVA, A. C. R., **Contribuição à minimização e simulação de circuitos lógicos,** 1989. 113 f. Dissertação (Mestrado em Engenharia Elétrica) – UNICAMP, Campinas.

[DEM 89] DEMARCO, Tom. **Análise Estruturada e Especificação de Sistemas**. Rio de Janeiro: Campus. 333pp. 1989

[DeM 94] De MICHELI, G. **Synthesis and optimization of digital circuits**. Mc Graw-Hill, Inc., 1994

[EDW 01] EDWARDS. Stephen A. **Design Languages for Embedded System**, Mountain View: Synopsys, 2001. 10p.

[ERC 00] ERCEGOVAC, Milos et alli. **Introdução aos Sistemas Digitais**. Porto Alegre: Bookman, 2000, 453 p.

[EVE 98] EVERITT, J. Parker et alli. **A 10/100 Mb/s CMOS Ethernet Transceiver for 10BaseT, 100BaseTX, and 100 Base FX**. ISSCC 98. seção 13, IEEE International Solid State Circuits Conference. paper 13.6, 210 p, 1998.

[FUH 97] FUHRER, Robert M. et alli. **OPTIMISTA: State Minimization of Asynchronous FSMs for Optimum Output Logic.** NSF Award CCR-97-34803.DCS – Columbia University, New York .

[GAJ 83] GAJSKI, D.D. e KUHN, R.H.. **New VLSI Tools**. IEEE Computer, New York, Dec, 1983, 11-14p

[GEN 95] GENOE, Mark et alli. **On the use of VHDL-based behavioral synthesis for telecom ASIC design**. Eighth International Symposium On System Synthesis, 6 pp. 1995

[GOL 86] GOLSON S. **State Machine Design Techniques for Verilog and VHDL**, Synopsys Journal of Automation Conference, IEEE, 1986, 320-326p.

[IEE 01] IEEE Std 1364-2001 (Revisão da IEE Std 1364-1995) **IEEE Standard Verilog® Hardware Description Language**, IEEE Computer Society, Set, 2001.

[INT 02a] INTEL . **Bussiness Computing**. Disponível em: <www.intel.com> Acesso em : 13/05/2002

[INT 02b] INTERSIL**, CD22103A,** Disponível em: <www.intersil.com > Acesso em: 20/06/02

[KEO 84] KEOGH, D.B. The State Diagram. **IEEE Transaction on Communications**, l. Com 32 no. 11, pp. 1222-1224, nov de 1984.

[KHO 01] KHOSLA, K. Pradeep et alli. **Modeling and Simulation methods for design of Engineering Systems**. Journal of Computing and Information Science in Engineering, ASME, 13 f. , 2001.

[KOH 78] KOHAVI, Zvi. **Switching and Finite Automata Theory**, 2. ed. New York: Editora McGraw-Hill, 1978. 658 p.

[LEE 99] LEE, Edward A. **Embedded *Software* – An Agenda for Research**. UCB ERL Me-morandum M99/63. University of California at Berkeley. 16p. dez. 1999.

[LIY 00] LI, Y. and LEESER, M.. **HML, a novel hardware description language, and its translation to VHDL**, IEEE Transactions On Very Large Scale Integration (VLSI) System, VOL.8. No. 1, fev, 2000.

[LOH 94] LOHSE, Jörg. **VHDL-ranslator for BDD-Based Formal Verification**. Munich: Siemens Corporate R&D, 1-8pp., 1994

[MAC 97] MACHADO, Ricardo J., et alli. **Redes de Petri e VHDL na Prototipagem Rápida de Sistemas Digitais**. Anais da Engenharia e Tecnologia Electrotécnica, Revista do Colégio de Engenharia Electrotécnica, Ordem dos Engenheiros, ano II, no. 4, pp. 1-4, Lisboa, Portugal, July, 1997.

[MAG 92] MAGINOT, Serge. **Evaluation Criteria of HDLs: VHDL compared to Verilog, UDL/I & M**, IEEE, 746-751pp.,1992.

[MAR 99] MARRA, Felipe Walcarenghi et alli. **Tradutor de C para VHDL C2VHDL**. Rio de Janeiro: DESC - FEN – UERJ, agosto,1999

[MAR 95] MARRANGHELLO, Norian. **Apostila de Métodos de Análise de Sistemas Digitais**. IBILCE. UNESP. 1995, 101f.

[MAR 01] MARCON, César A. Missio, **Modelagem de Sistemas de Telecomunicação para o Projeto Integrado de *Hardware* e *Software***. 2001. 136 f. Trabalho Individual I (Ciência da Computação) - Faculdade de Informática - PUC RS. Porto Alegre.

[McC 56] McCluskey E.J.. **Minimization of Switching Function**. Bell System Tech. J., vol. 35, Nov. 1956, pp. 1417-1444.

[ONL 96] O´NILS, Mattias. **Hardware/Software Partitioning of Telecommunication System**, 1996. 55 f. Licentiate thesis (Eletronic System Design), Departament of Eletronics, Stockholm. Sweden.

[PÉR 95] PÉRICO, Luiz Carlos. **Técnicas de Transmissão de Dados**. Módulo U1. Brasília: TELEBRÁS. 103f. Julho, 1995

[PER 91] PERRY ,D. L.. **VHDL**. 4.ed. New York: McGraw-Hill Inc, 1991. 459p.

[ROC 99] ROCHA, José F. et alli**. Projecto de Circuitos Integrados Digitais Utilizando VHDL**. DEE e Comunicações – Instituto Superior de Engenharia de Lisboa. 1999, 6p.

[SAS 97] SASAKI, Hisashi et alli. **Semantic Validation of VHDL-AMS by an Abstract State Machine**, in Proceedings of BMAS'97 (IEEE/VIUF International Workshop on Behavioral Modeling and Simulation), Arlington, VA, October 20-21, 1997, 61-68.

[SHA 86] SHAHDAD, Moe. **An overview of VHDL Language e Technology**. IEEE 23rd Design Automation Conference, Paper 17.1, 320-326p, 1986.

[TAS 99] TASSI, Walter. **Sociedade da Informação – Montando o Quebra-Cabeça Digital**. In: Seminário de Redes, Campinas: CPqD,8p, 1999. CD-ROM.

[TAN 02a] TANCREDO, Leandro de O. Tancredo, **Métodos e Ferramentas de Projetos Digitais Seqüenciais e Programa TABELA**. 18 jan 2002. 82 f. Estudos Especiais I - DEE FEIS UNESP. Ilha Solteira.

[TAN 02b] TANCREDO, Leandro de O. Tancredo, **Ferramenta de Síntese TAB2VHDL**. 11 março 2002. 96 f. Estudos Especiais II – DEE FEIS UNESP. Ilha Solteira.

[TEL 94] TELEBRÁS, **Introdução à Teleinformática**, Brasília: CNTr. Nov/94

[UEH 97] UEHARA, José M. **Evolução Tecnológica em Telecomunicações**. In: SEMINÁRIO CENÁRIOS TECNOLÓGICOS DE TELECOMUNICAÇÕES, 1, 1997. Campinas: TELEBRÁS/CPqD. 28p.

[XIL 00] XILINX. **StateCAD User´s Guide**. Xilinx Inc, 2000.

[XIL 01] XILINX**. Xilinx Programmable Logic Book**. Disponível em: <www.xilinx.com > Acesso em 09/12/2001

[WAK 00] WAKERLY, John F. **Digital Design: Principles and Practices**, 3 .ed. New York : Editora Prentice-Hall, 2000.946 p.

GLOSSÁRIO

% Útil - informa a porcentagem útil de silício utilizado de um circuito integrado;

Aprendizagem – O tempo e a condição de abstrair uma linguagem, ou seja, esta característica envolve os custos relacionados ao tempo e, invariavelmente, à qualidade das especificações dos projetistas que estão iniciando o uso das linguagens. A linguagem C destaca-se como apoio em sistemas muito difíceis de serem especificados. O VHDL torna-se muito complexo para o aprendizado, devido a inúmeras construções de linguagens e ao significado descritivo para gerar o *hardware* desejado;

Blocos Lógicos - Funções Lógicas são implementadas no interior dos Blocos Lógicos. Em algumas arquiteturas os Blocos Lógicos possuem recursos seqüências tais como *flip-flops* ou registros. O fabricante Xilinx chama seus blocos lógicos de CLB (*Configurable Logic Block*), enquanto que a Actel usa o termo LM (*Logic Modules*). Já a altera usa o termo LE (Logic Element) para as séries 8000 e 10000 e Macrocell para as séries 5000,7000 e 9000 e outros;

CCS - Técnica de detecção de sincronismo representada através do diagrama de estado, utilizado em um sistema PCM de 120 canais;

Custo - é a somatória aritmética dos números de terminais de entrada das portas lógicas AND e OR;

Documentação – Declaração que auxilia outros programadores a desenvolver um *software*. A linguagem VHDL permite a implementação de descrições com interfaces bem definidas, a linguagem C é que apresenta maiores problemas relacionados ao item documentação. Conseqüentemente no apêndice I, documenta-se o SEQ2VHDL através da análise estruturada;

Don't Care State - Estado lógico cujo valor não é relevante para a determinação de uma função;

EEPROM - *Electrically Erasable* Programmable ROM;

EPLD (EPROM *Programmable Gate Array*) - são componentes que possuem EPROM para possibilitar sua programação.

EPROM possuem dois Gates:

- *Floating Gate*
- *Select Gate*

O *floating Gate* não está conectado ao circuito, daí o seu nome, e não possui nenhuma carga no seu estado normal. Neste estado, o componente pode ser chaveado normalmente pelo

Select Gate que na fase de programação do componente, o componente pode ser percorrido por uma corrente mais alta e uma carga fica armazenada sob o *Floating Gate.* Desta forma o componente fica permanentemente aberto (OFF). Para desprogramar o componente basta submete-lo a uma exposição de raios ultra-violeta que excitam os elétrons armazenados sob o *Floating Gate* até que estes atravessam a camada de óxido que os separa dos substrato;

As desvantagens desta tecnologia é que o componente *pull-up* está permanentemente conduzindo o que provoca um consumo de potência maior no circuito;

EPROM – *Erasable Programmable ROM*

Facilidade de Corrigir Erros e Manutenção - Otimizar e ter idéias de custos estimados, e durante a vida útil do projeto poder modificar e incluir novas tarefas. Assim sendo a linguagem C, apresenta um forte problema, pois confere uma certa liberdade ao programador, o que resultaria em descrições muito pouco flexíveis,enquanto que a linguagem VHDL permite a implementação com características bem definidas;

Fan-in representa a limitação física dos número de entradas de um circuito digital;

Ferramentas Disponíveis da Linguagem - com relação ao público alvo, pois isto acaba se tornando um fator importante no processo de formação dos circuitos, na VHDL há uma grande cultura difundida no mercado, que resulta na elevada quantidade de ferramentas para descrição, validação e síntese. Sendo que a linguagem C tem-se como enfoque, a especificação de *softwares* científicos e comerciais.

FPGA (*Field Programmable Gate Array*) - circuitos programáveis compostos por um conjunto de células lógicas ou blocos lógicos alocados em forma de matriz. Em geral, a funcionalidade destes blocos assim como o roteamento, são configuráveis por software. A palavra *field* indica que a configuração do circuito pode ser feita pelo usuário final;

Flash - É um tipo de memória cujo conteúdo pode ser apagado submetendo-se o componente a sinais elétricos apropriados;

HDL (*Hardware Description Language*) - Linguagem de Descrição de Hardware são linguagem que permitem a descrição de um projeto eletrônico em um ou mais representações;

Idealização da Linguagem - para que determinada finalidade o *software* foi criado, ou seja, o VHDL foi projetado para descrição de *hardware* e a linguagem C para elaboração de programas de médio e baixo nível de abstração, possuindo como característica uma ordem seqüencial de execução;

Implementação de Sistemas em Baixo Nível - é a capacidade de descrição das camadas físicas, de enlace e de rede do modelo OSI. Essas camadas trabalham essencialmente com protocolos orientados a *bit*, o VHDL permite detalhar com precisão qual *bit* será manipulado, enquanto que a C, tem alto poder de manipulação de *bits*, entretanto tem deficiência em representar instantes de tempo;

K em N – K certos em N testes – é um diagrama de estado que compara N quadros de sincronismo. Se das comparações resultarem K ou mais certas, o sistema permanece em sincronismo, caso contrário passa para o estado de fora do sincronismo;

LUT (*Look-Up Tables*) - são memórias RAM utilizadas para implementar funções lógicas. As entradas da função correspondem ao endereço enquanto que a saída corresponde ao conteúdo da memória. Uma look-up table de n entradas pode realizar todas as funções lógicas de até n entradas que são 2^{2^n} diferentes funções;

Memória - representa o pico de alocação de memória de um circuito digital;

Mintermos (de n variáveis) - termo de produto normal com n literais, em que cada variável aparece uma única vez ou na forma complementada ou na não complementada, quando a função lógica possui n variáveis resultará em 2^n mintermos;

Público Alvo - são as tendências das instituições de ensino, pesquisa e produção, no direcionamento de esforços para a aprendizagem e uso de uma determinada linguagem. Dessa forma, o VHDL apresenta um público menor que a C, mesmo sendo uma linguagem padrão para a descrição de *hardware*;

Representação Lógica - avalia a capacidade da linguagem de representar sinais booleanos através de vários níveis lógicos, ou seja, alta impedância, valores inválidos, zero fraco, zero forte, vários níveis lógicos e a tomada de decisão com relação a conflitos lógicos. Assim, a linguagens VHDL suportam um maior número de representações lógicas, com relação a linguagem C;

S10010 - Quando o diagrama de estado detecta uma seqüência 10010, e caso a afirmação for identificada como positiva, o diagrama de estado informa que a seqüência foi detectada.

SOP (do inglês Sum of Products) - ou Soma de Produtos, dois ou mais termos de produtos ligados pelo conectivo OR;

SRAM - Memória RAM estática, usada para realizar LUTs;

Sincronismo – quando ás linguagens dão suporte a descrição síncrona, as duas linguagens apresentam essa característica;

Transição de Estados - capacidade de representar os estados de uma máquina e suas transições, ambas as linguagens apresentam essa característica, mas pode-se notar a dificuldade em especificar os sistemas, os quais são naturalmente vistos pelos projetistas como algoritmos implementados através do diagrama do tempo;

TRANSU ou Transições Suaves - é um diagrama no qual o sistema está em sincronismo, quando uma palavra é detectada incorretamente, leva o sistema a uma situação de pré-alarme, onde cada nova palavra detectada leva o sistema à estados cada vez mais próximo do estado fora do sincronismo;

VLSI (VHSIC *Large Scale Integration*) - onde VHSIC significa (*Very High Speed Integrated Circuits*);

APÊNDICE I

ANÁLISE ESTRUTURAL DO AMBIENTE TABELA E SEQ2VHDL

A. DESCRIÇÃO DO SISTEMA TAB2VHDL

Conforme [DEM 89] documentar-se-á com uma especificação estruturada para o completo entendimento do programa SEQ2VHDL que é uma ferramenta de projeto de síntese de alto nível que a partir de um arquivo gerado pelo programa TABELA, contendo a descrição de um circuito lógico no Domínio Estrutural, Nível Lógico, converte este sistema para o Domínio Comportamental, Nível RTL, ou seja, em VHSIC Linguagem de Descrição de *Hardware* (VHDL) codificação de extrema versatilidade desenvolvida para auxiliar em muitos aspectos em um Projeto de Circuito Integrado.

B. OBJETIVO

O objetivo do sistema TAB2VHDL é facilitar a vida do projetista, o circuito é gerado a partir de uma máquina de estado finita fornece uma descrição do circuito na linguagem VHDL, e através de qualquer tecnologia FPGA ou outra tecnologia disponível se tem um circuito minimizado, com custos operacionais reduzidos na confecção do projeto. Proporcionará uma maior velocidade e segurança na implementação física do circuito integrado.

Neste caso, o circuito pode ser testado logicamente a partir de um microcomputador PC utilizando uma ferrramenta de Automação de Projetos Eletrônicos (EDA em Inglês), como Altera [ALT 01] e Xilinx [XIL 01], que interpreta a linguagem VHDL, e seguidamente permitir ao projetista avaliar o resultado para finalmente implementar o *chip.*

C. DESCRIÇÃO DOS PROCESSOS

Todas as informações devem ser armazenadas em um arquivo com extensão .TAB conforme Item 3.1 pois através dessa metodologia não necessitamos de reentrar com as informações toda vez que executarmos o programa (representado no processo P1). Assim podemos digitar o projeto através do programa TABELA como mostra o processo P1. No processo P3 é gerado um arquivo com extensão TXT, vide Item 3.1. Que será executado pelo Programa SEQ2VHDL, a partir do processo P4, que gera o arquivo final com extensão VHD.

Vamos detalhar os processos que envolvem o projeto:

PROCESSO P 1

Cadastra os dados de uma máquina de Mealy gerada a partir de uma tabela de estado e das unidades de memórias, conforme Figura 36, entrar com as informações, utilizar um editor de texto, segundo item 3.1, ARQUIVO.TAB;

O processo P 1, subdivide-se em vários subprocessos:

PROCESSO P 1.1

Número de *flip-flops*, conforme Item 3.1, ARQUIVO.TAB ou conforme tela representada na Figura 45;

PROCESSO P 1.2

Neste processo, o projetista deve selecionar, um dos tipos de *flip-flops* utilizados (podem ser do tipo D ou JK) , conforme item 3.1, ARQUIVO.TAB ou conforme tela representada na Figura 45;

PROCESSO P 1.3

Número de variáveis de entrada seguido pelo número de variáveis de saída, conforme Item 3.1, ARQUIVO.TAB ou conforme tela representada na Figura 44;

PROCESSO P 1.4

A tabela de próximo estado, conforme item 3.1, observar arquivo HDB3.tab, deve ser descrita na seguinte forma:

Estado atual Próximo estado Entrada Saída.

De acordo com item 3.1 , ARQUIVO.TAB ou conforme tela representada nas Figura 46, Figura 47 e Figura 48;

No final do cadastro dos dados deve-se colocar o número –100 para indicar o fim das entradas de informações do ARQUIVO.TAB

PROCESSO P 2

Cadastra os dados solicitados pelo programa TABELA para montagem do cabeçalho e especificação do nome do arquivo de saída e seleção da opção de saída dos resultados, conforme diagrama representado Figura 37 detalhados em vários subprocessos:

PROCESSO P 2.1

Cadastro do nome do arquivo de saída, conforme tela representada na Figura 42;

PROCESSO P 2.2

Cadastro da Identificação do caso a ser rodado conforme tela representada, na Figura 42;

PROCESSO P 2.3

Cadastro da data atual, conforme tela representada na Figura 42;

PROCESSO P 2.4

Cadastro do usuário, conforme tela representada na Figura 42;

PROCESSO P 2.5

Selecionar opção via terminal ou via arquivo, conforme tela representada por meio da Figura 43;

PROCESSO P 3

Cria os dados do programa TABELA com a especificação do nome do arquivo de entrada, conforme diagrama representado por meio da Figura 38. Com isso, o processo fragmenta-se em:

PROCESSO P 3.1

Cadastro do nome do arquivo; conforme tela representada pela Figura 42;

PROCESSO P 3.2

Monta o cabeçalho do arquivo TABELA.TXT;

PROCESSO P 3.3

Calcula a Tabela de Transição;

PROCESSO P 3.4

Utiliza-se do algoritmo de Quine-McCluskey [McC 56], obtendo-se a fórmula mínima;

PROCESSO P 3.5

Calcula FUNCÕES com mintermos e os implicantes primos;

PROCESSO P 4

Entrada dos dados do programa SEQ2VHDL, conforme diagrama representado na Figura 40. Neste processo, a divisão se faz em:

PROCESSO P 4.1

Entrar com o nome do arquivo de leitura ARQUIVO, conforme tela representada por meio da Figura 49;

PROCESSO P 4.2

Cria cabeçalho do TABELA.VHD;

PROCESSO P 4.3

Identificação dos estados;

PROCESSO P 4.4

Cria entidade de entrada, estado e saída, representada por meio do diagrama da Figura 40; Este processo decompõe-se em:

PROCESSO P 4.4.1

Cria entidades de entradas no arquivo TABELA.VHD;

PROCESSO P 4.4.2

Cria entidades de estado no arquivo TABELA.VHD;

PROCESSO P 4.4.3

Cria Entidades de saída(s);

PROCESSO P 4.5

Cria Arquitetura de sinais, modelos, funções booleanas, conforme diagrama representado por meio da Figura 41; E novamente o processo segmenta-se em:

PROCESSO P 4.5.1

Define Sinais, subdividindo-se em:

PROCESSO P 4.5.1.1

Fila Modelos;

PROCESSO P 4.5.1.2

Sinal de Controle;

PROCESSO P 4.5.2

Inferir Modelos

PROCESSO P 4.5.2.1

Ler filas. Representando a seguinte divisão:

PROCESSO P 4.5.2.1.1

Modelo JK

PROCESSO P 4.5.2.1.2

Modelo D

PROCESSO P 4.6

Cria a tabela em VHDL TABELA.VHD;

D. DESCRIÇÃO DOS DEPÓSITOS DE DADOS

D1 TABELA.TAB NÍVEL RTL, DOMINIO COMPORTAMENTAL

- Números de *Flip-flop*
- Tipos de cada um dos *Flip-flops* (D ou JK)
- Números de Entradas/Saídas
- Transições entre estados: Estado atual próximo estado entrada saída

D2 TABELA.TXT NÍVEL LÓGICO, DOMINIO COMPORTAMENTAL

- Caso
- Data
- Usuário
- Entradas
- Estados Atuais
- Próximos Estados
- Entradas De Excitação Do Flip-Flop
- Mintermos
- *Don't Care States*
- Implicantes Primos
- Custo Final

D3 DOMÍNIO COMPORTAMENTAL, NÍVEL RTL TABELA.VHD

- Entradas Xn
- Saídas Zn
- Variáveis de Estado Qn
- Sinais Auxiliares VEn
- Funções de Controle Jn, Kn e/ou Dn
- *Flip-Flops* Dn e/ou Jn, Kn
- Funções Combinacionais

E. MODELAGEM DE DADOS

Descreve-se aqui, as principais variáveis utilizadas pelos programas TABELA e pelo SEQ2VHDL.

Tabela do Diagrama de Fluxo de Dados P1:

P1 – CRIANDO ARQUIVO DE ENTRADA DO TABELA			
MODELAGEM	**TIPO**	**TAMANHO**	**DESCRIÇÃO**
Num_ff	Inteiro	1	Número de Flip-flop
Tipo_ff	Caracter	2	Tipo do flip-flop
Var_entr	Inteiro	1	Variáveis de entrada
Var_sai	Inteiro	1	Variáveis de saídas
Est_atual	Inteiro	1	Estado Atual da tabela de estado
Pro_Estado	Inteiro	1	Próximo Estado da tabela de estado
Entrada	Inteiro	1	Entrada da tabela de estado
Saída	Inteiro	1	Saída da tabela de estado

Tabela 16 - Criando arquivo de entrada do programa TABELA

Tabela do Diagrama de Fluxo de Dados P2:

P2 – ENTRADA DE DADOS TABELA DO PROGRAMA			
MODELAGEM	**TIPO**	**TAMANHO**	**DESCRIÇÃO**
Arq	Caracter	8	Nome do arquivo
Caso	String	30	Identificação do caso
Data	String	8	Data atual
Nome_usu	String	30	Nome do usuário
Opção_prog	Inteiro	1	Via terminal ou arquivo

Tabela 17 - Entrada de Dados do Programa TABELA

Tabela do Diagrama de Fluxo de Dados P4.3:

P4.3 – IDENTIFICA ESTADO DO PROGRAMA SEQ2VHDL			
MODELAGEM	**TIPO**	**TAMANHO**	**DESCRIÇÃO**
Linha	Caracter	1	Caracter de leitura do arquivo TABELA
FF	Inteiro	1	Contador de Q
Sai	Inteiro	1	Contador de Z
Ent	Inteiro	1	Contador de 01X
Num_estados	Inteiro	1	Número de Estados
Num_saida	Inteiro	1	Número de Saídas
Num_entrada	Inteiro	1	Número de entradas
Size_linha	Inteiro	1	Tamanho da linha

Tabela 18 - Identifica estado do programa SEQ2VHDL

Tabela do Diagrama de Fluxo de Dados P4.4:

P4.4 – CRIA ENTIDADE DO PROGRAMA SEQ2VHDL			
MODELAGEM	**TIPO**	**TAMANHO**	**DESCRIÇÃO**
Nome_entity	String	12	Nome da entidade
N_ent	Inteiro	1	Número de entradas
N_est	Inteiro	1	Número de Estados
N_sai	Inteiro	1	Número de Saídas

Tabela 19 - Cria entidade do Programa SEQ2VHDL

Tabela do Diagrama de Fluxo de Dados P4.5:

P4.5 – CRIA ARQUITETURA DO PROGRAMA SEQ2VHDL			
MODELAGEM	**TIPO**	**TAMANHO**	**DESCRIÇÃO**
Nome_entity	String	12	Nome da entidade
Tipo_ff	Caracter	1	Tipo de Flip-flop
N_est	Inteiro	1	Número de Estados
Tipo	Caracter	1	Tipo do insere_fila
Índice_ff	Inteiro	1	Índice do modelo
Size_pacote	Inteiro	1	Tamanho do Pacote
Conta_entrada	Inteiro	1	Conta as entradas
Conta_estado	Inteiro	1	Conta os estados
Conta_variável	Inteiro	1	Conta as variáveis
Aux_linha	caracter	1	Implicante ou 0 ou 1
Aux_2	caracter	1	Implicante VHDL VEn ou NOT(VEn)

Tabela 20 - Cria arquitetura do programa TAB2VHDL

F. DIAGRAMA DE FLUXO DE DADOS (D.F.D)

D.F.D. 1- CRIAÇÃO DA TABELA DE ENTRADA DO PROGRAMA TABELA

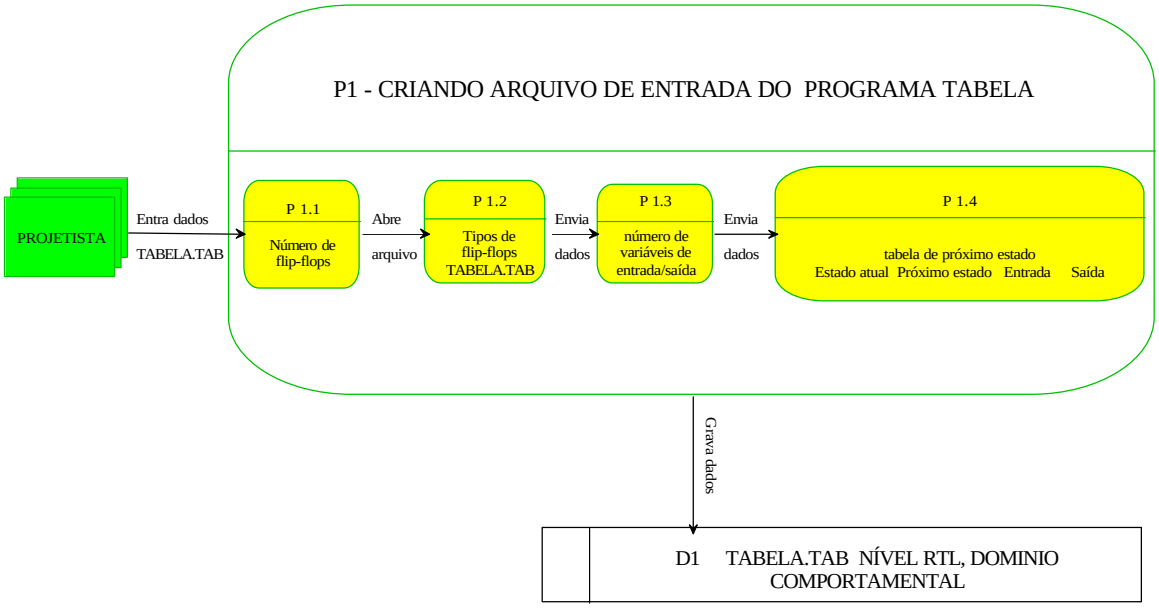

Figura 36 - Diagrama de Fluxo de Dados 1

D.F.D. 2 - ENTRADA DE DADOS DO PROGRAMA TABELA

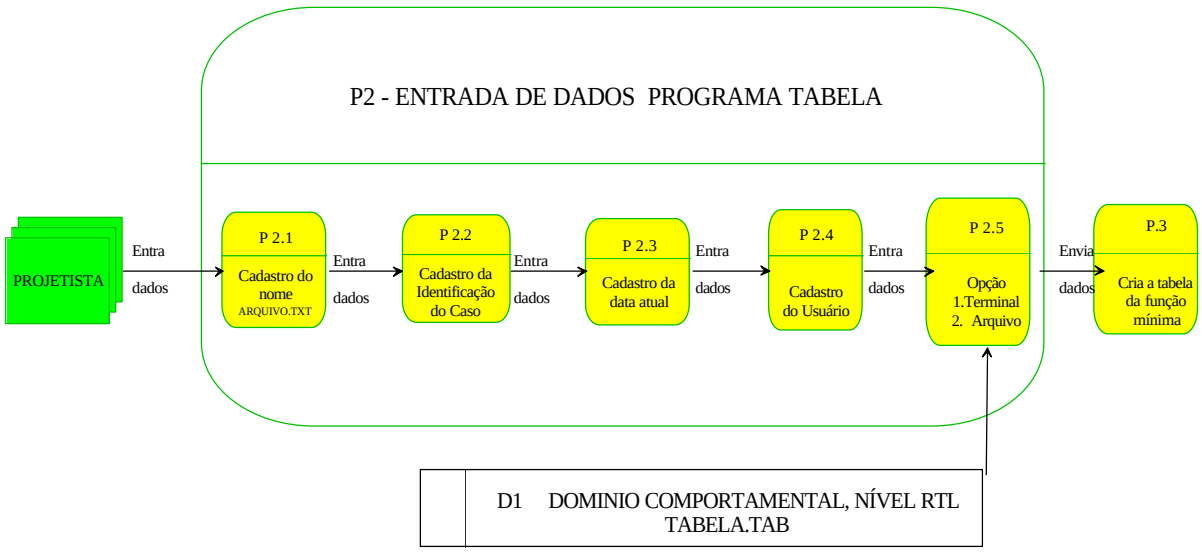

Figura 37 - Diagrama de Fluxo de Dados 2

D.F.D. 3 - CRIAÇÃO DA TABELA DE SAÍDA DO PROGRAMA TABELA

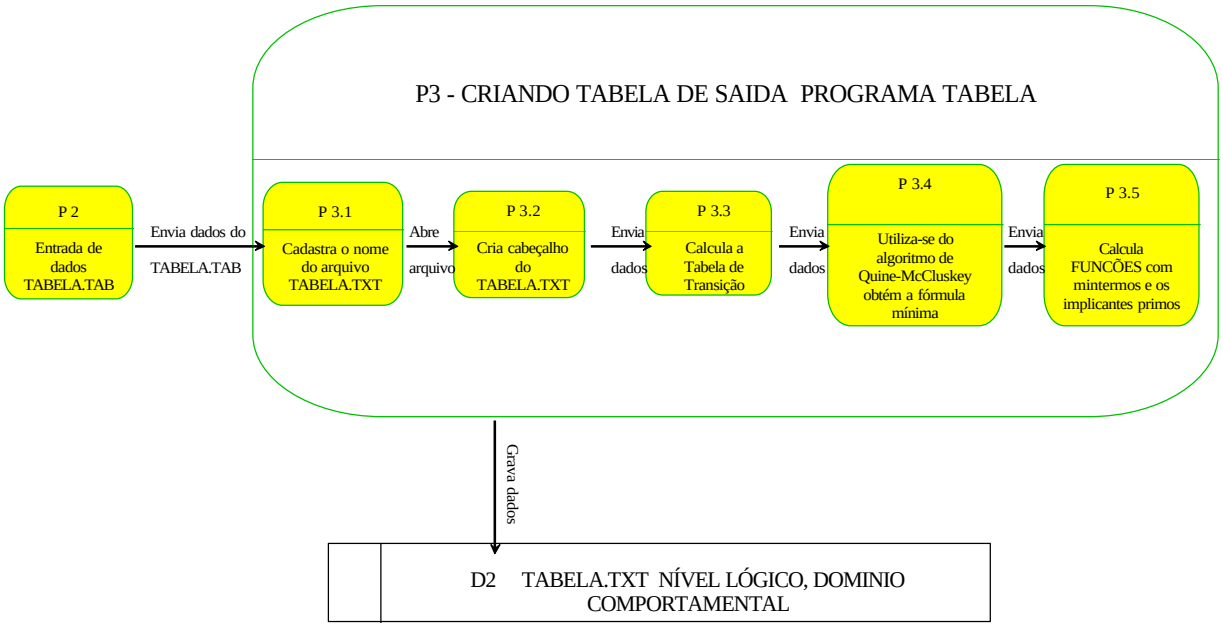

Figura 38 - Criando Tabela de saída do programa tabela

D.F.D. 4 - ENTRADA DE DADOS DO PROGRAMA SEQ2VHDL

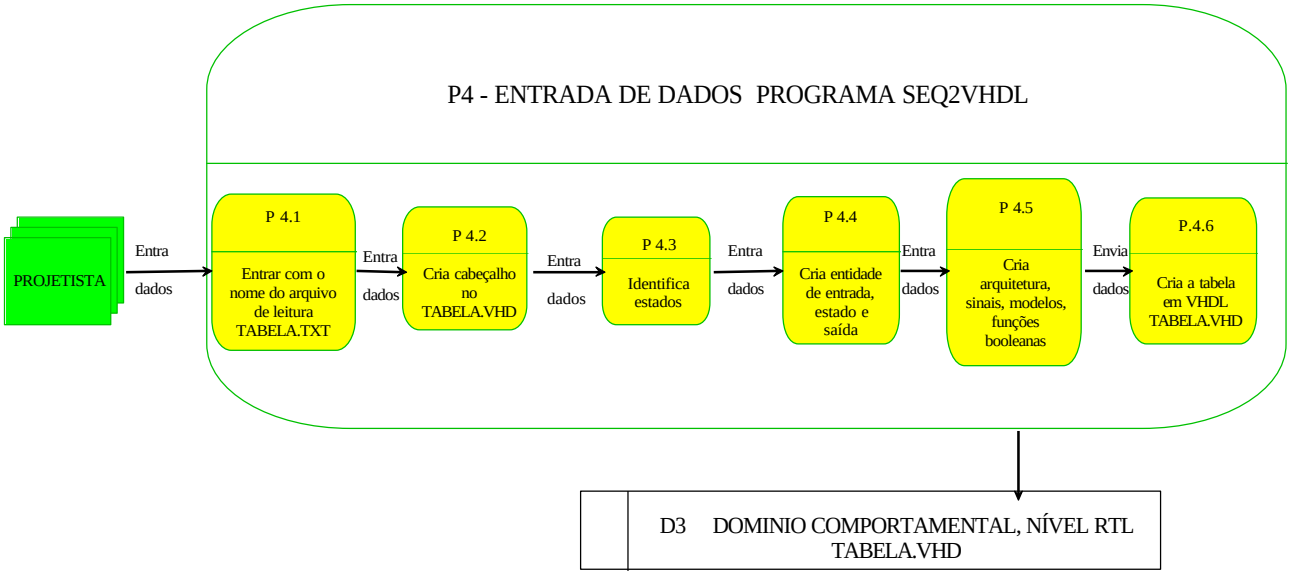

Figura 39 - Diagrama de Fluxo de Dados 4

D.F.D. 5 - RODA O PROGRAMA SEQ2VHDL CRIAÇÃO DE ENTIDADES EM VHDL

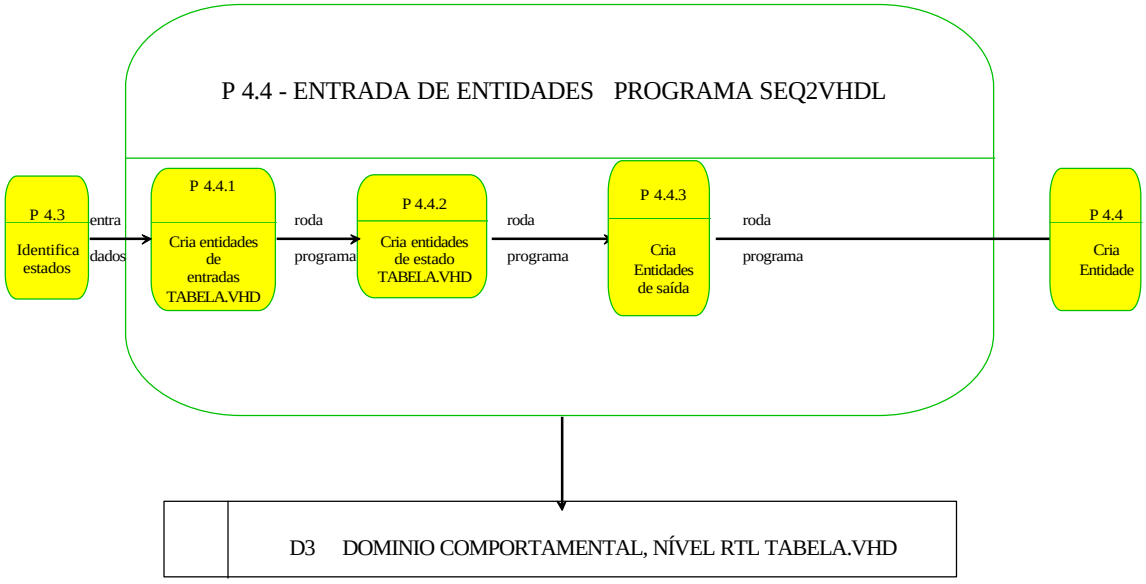

Figura 40 - Diagrama de Fluxo de Dados 5

D.F.D. 6 - RODA O PROGRAMA SEQ2VHDL CRIAÇÃO DE ARQUITETURAS EM VHDL

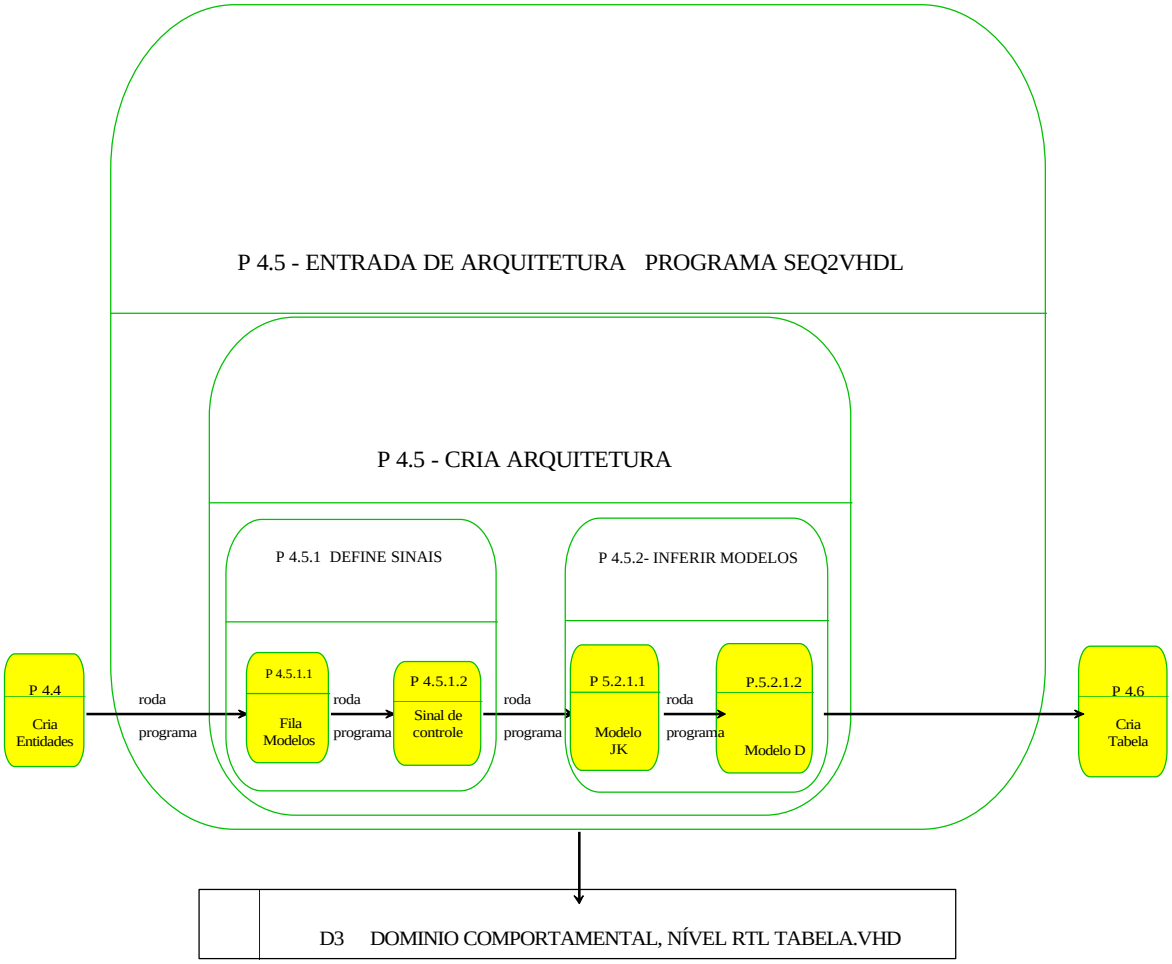

Figura 41 - Diagrama de Fluxo de Dados 6

G. DESCRIÇÃO DAS INTERFACES

Menu Cadastro do Arquivo de Saída do Programa TABELA

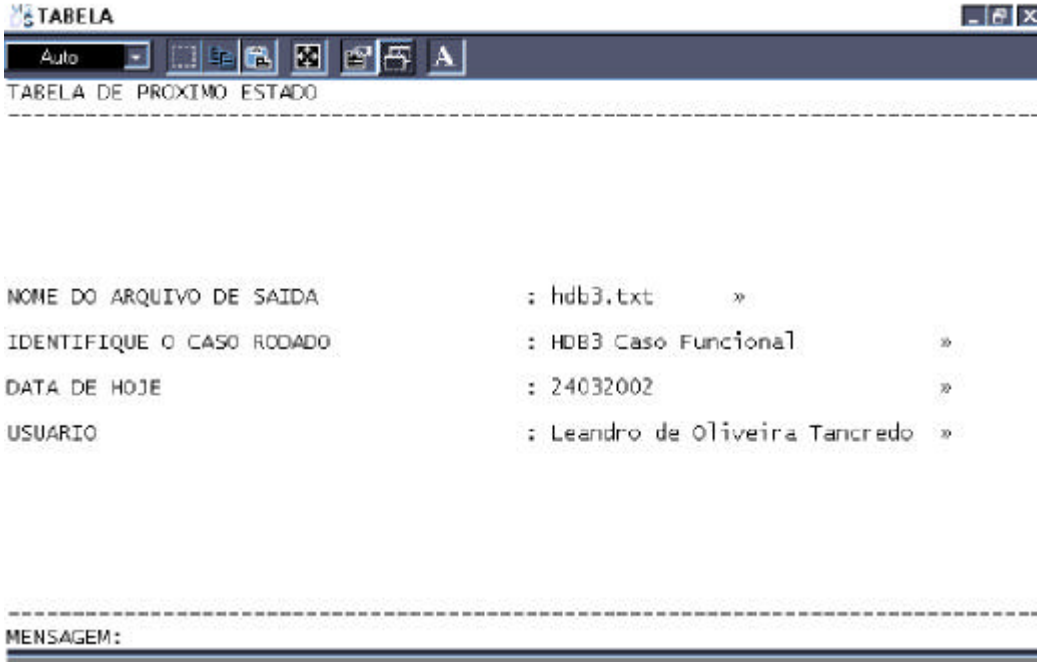

Figura 42 - Menu Cadastro do Arquivo de Saída do Programa TABELA

Menu das opções de entrada de dados do Programa TABELA

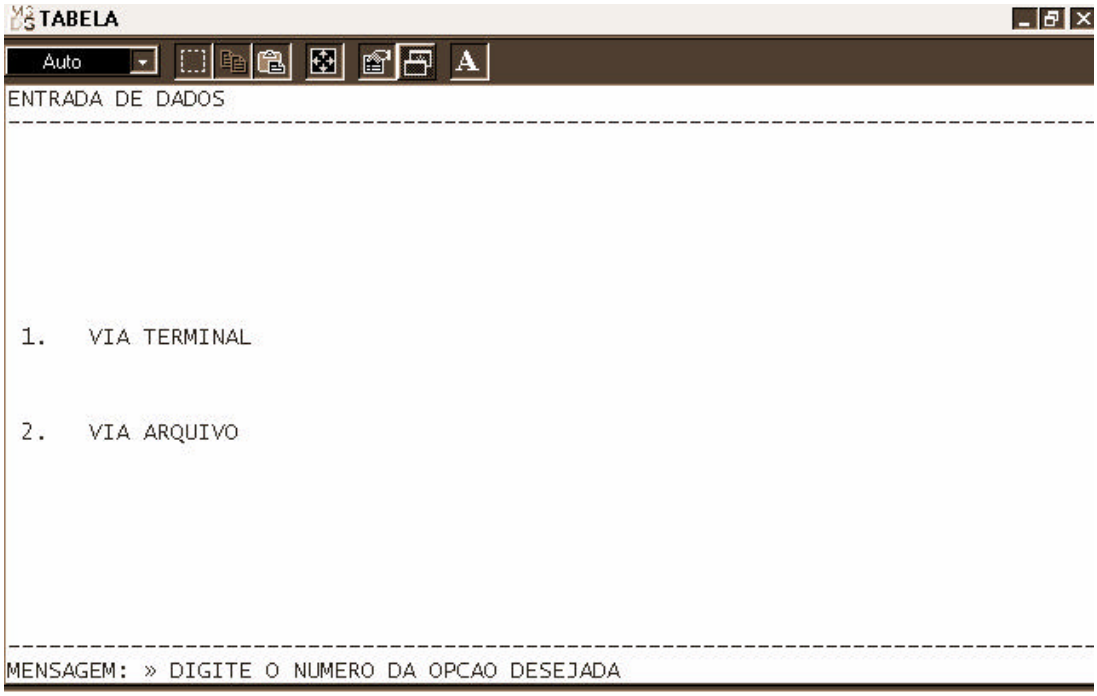

Figura 43 - Menu das opções de entrada de dados

Menu para informar o número de entrada e saída do circuito

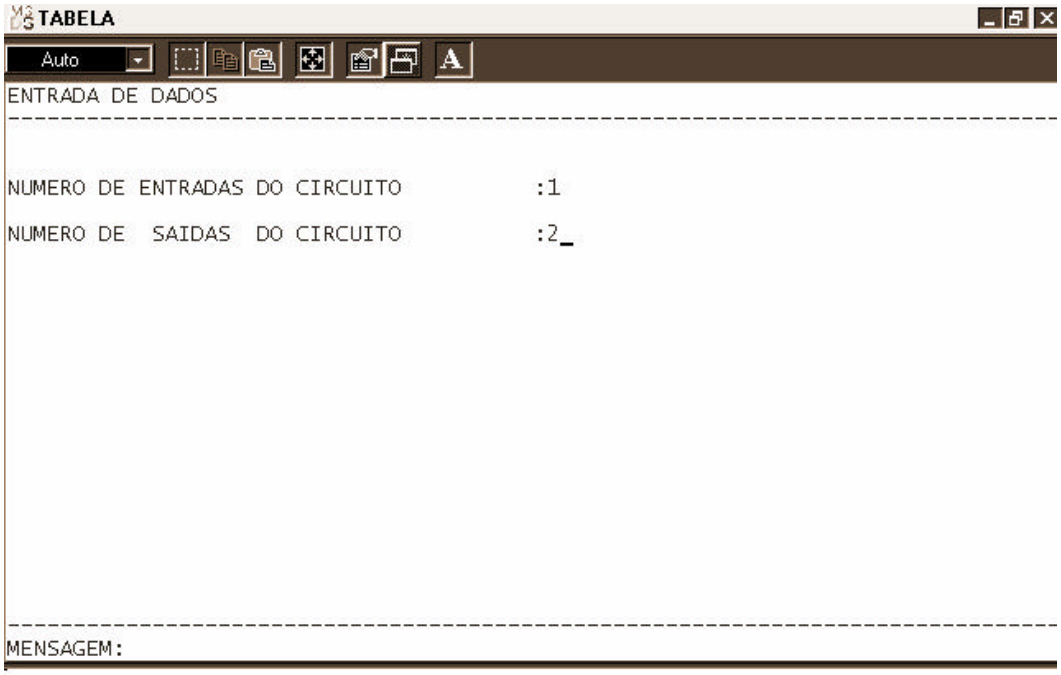

Figura 44 - Menu do número de entrada e saída

Menu inclusão do Número de flip-Flop e o seu tipo.

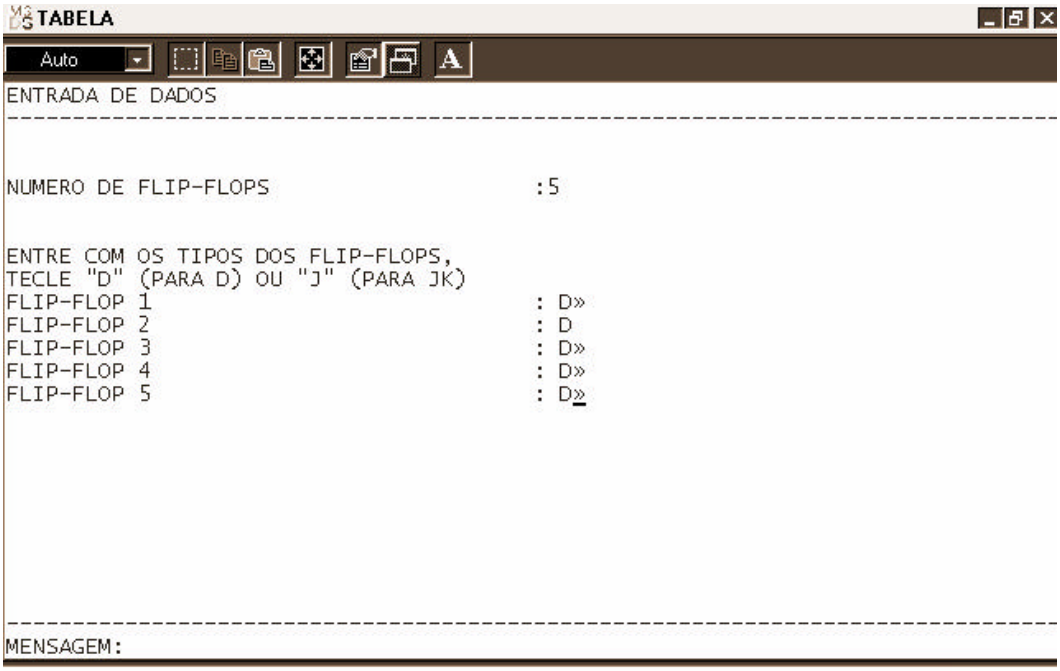

Figura 45 - Menu de inclusão do número de flip-flop e o tipo

Menu de entrada da tabela de estado

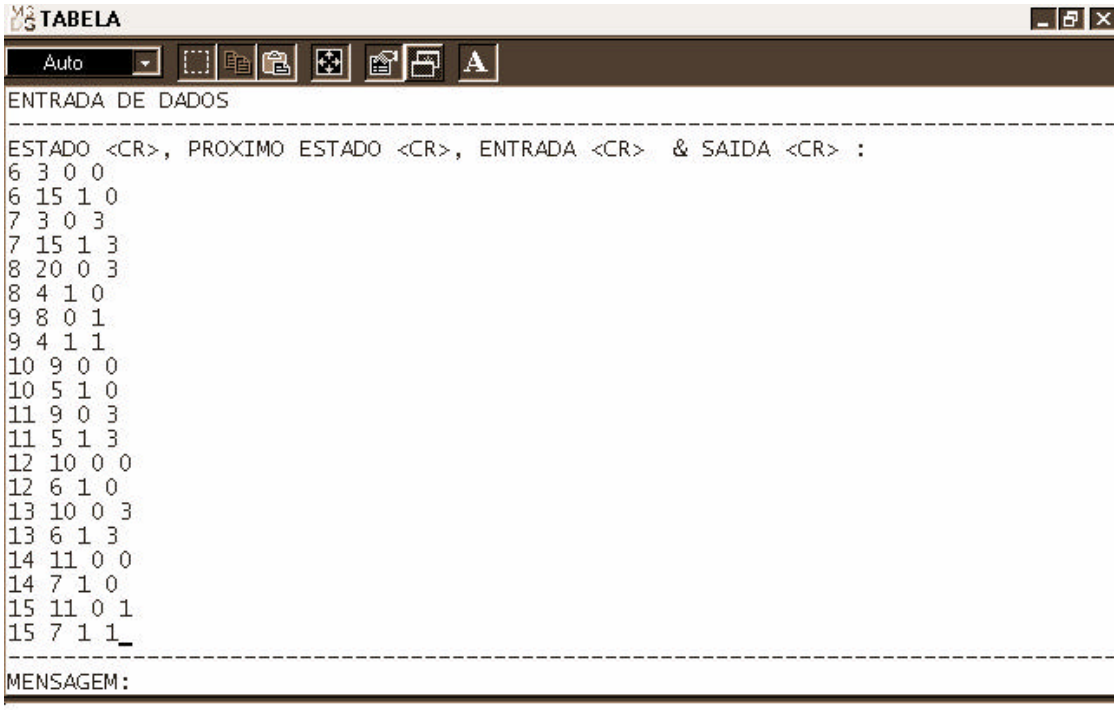

Figura 46 - Menu de inclusão da tabela de estado

Menu de Entrada de Dados – Continuação.

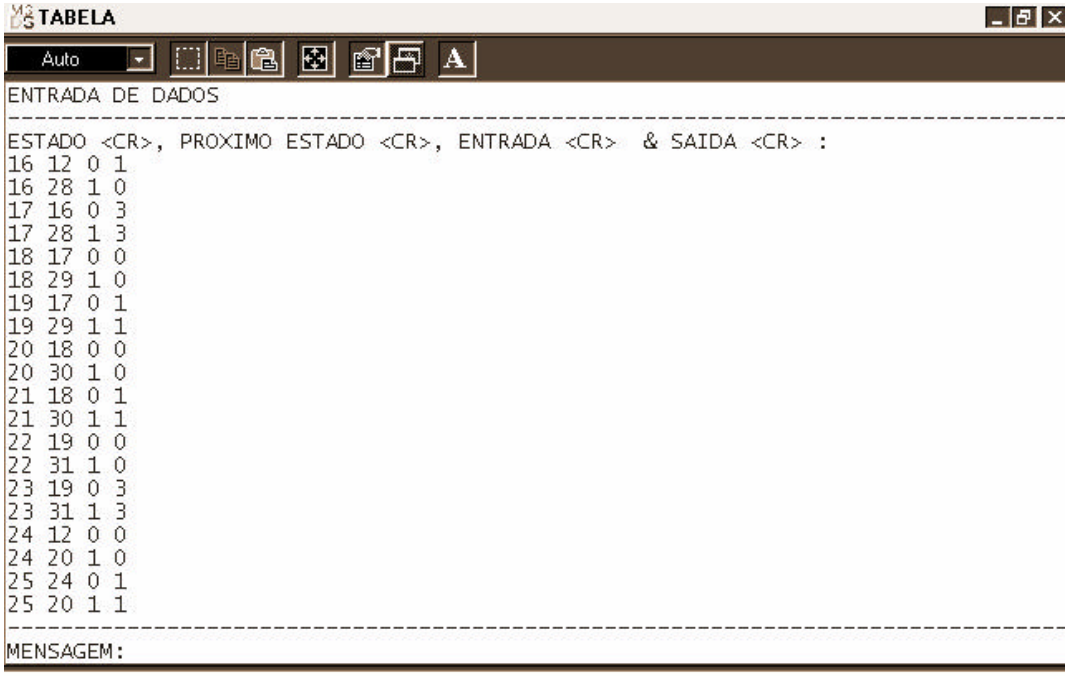

Figura 47 - Menu de entrada de dados - continuação

Menu de Entrada de Dados – Finalização

Figura 48 - Menu de entrada de dados - finalização

Menu SEQ2VHDL

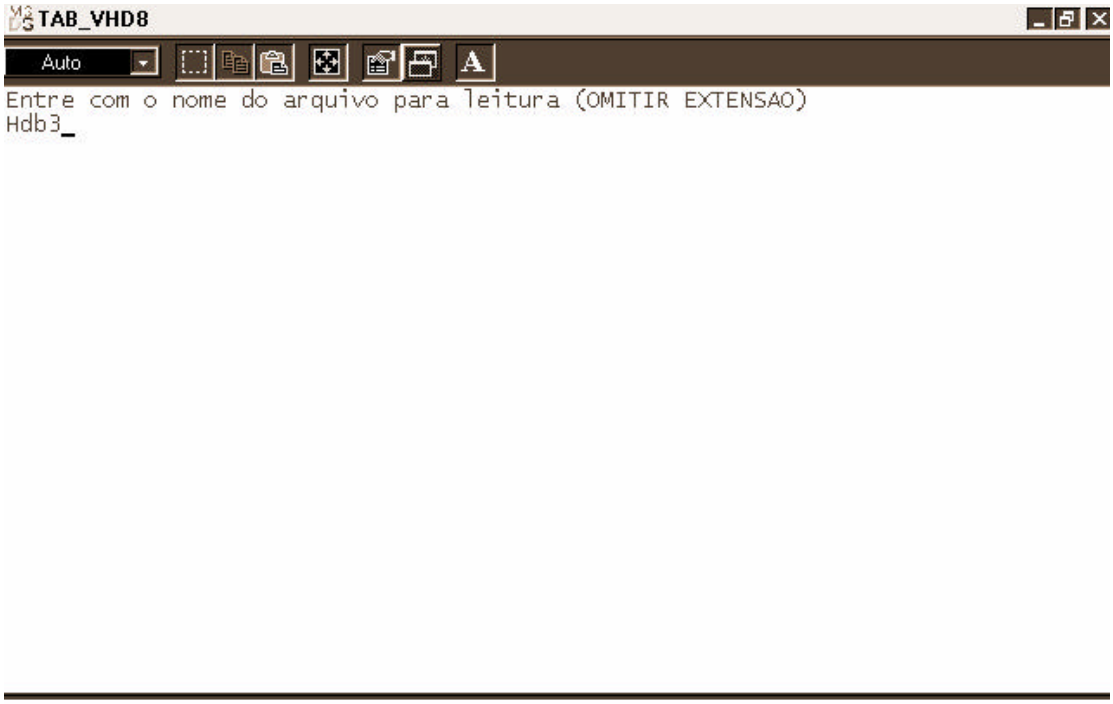

Figura 49 - Menu SEQ2VHDL

APÊNDICE II
LISTAGEM DO PROGRAMA SEQ2VHDL

```
       /* UNIVERSIDADE  ESTADUAL  PAULISTA  "JULIO DE MESQUITA FILHO"
                     Departamento de Engenharia Elétrica
                           Curso de Pós-Graduação

Programa: SEQ2VHDL.C
Objetivo: Gerar um modelo VHDL de uma m quina de estados finitos gerada
        a partir de sua descrição em diagrama de transição de estado.
        O programa SEQ2VHDL tem como entrada o arquivo gerado pelo pro-
        grama TABELA e gera o modelo correspondente.
Programador: Leandro de Oliveira Tancredo
Versão: 9.0
Desenvolvimento:
INICIO: 24/05/2001                   ULTIMA ALTERACAO: 20/02/2002      */

#include <stdio.h>
#include <string.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

struct fila_modelo
{
 char tipo[2];
 char indice[7];
 struct fila_modelo *proximo;
} size_modelo;

FILE *le, *escreve;
char ler_arq[12], esc_arq[12], nome_entity[12];
char funcao_controle[500] = " ", aux_2[250];
int  num_estados,num_saida,num_entrada;
struct fila_modelo *new_fila, *cabeca, *rabo;

abre_arq()
{
 char arq[8];
 printf("Entre com o nome do arquivo para leitura (OMITIR EXTENSAO) \n"
);
 gets(arq);
 stpcpy(ler_arq,arq);
/* strcat(ler_arq,".bom"); +++++++++++++++ utilizar quando definir a
extensao */
 if((le = fopen(ler_arq,"r")) == NULL)
 {
   printf(" Falha na abertura do arquivo de leitura \n");
   exit(0);
 }
 stpcpy(esc_arq,arq);
 stpcpy(nome_entity,arq);
```

```
 strcat(esc_arq,".vhd");
 if((escreve = fopen(esc_arq,"w")) == NULL)
 {
   printf(" Falha na abertura do arquivo de escrita \n");
   exit(0);
 }
}

fecha_arq()
{
  fclose(le);
  fclose(escreve);
  printf("\n Finalizando programa  ...  Pressione alguma tecla  \n");
  getch();
}

cria_cabecalho()
{
 fprintf(escreve,"-- PROJETO DE TESE DE MESTRADO - Ferramenta de Síntese
Lógica \n");
 fprintf(escreve,"-- A ferramenta tem por objetivo transformar a
descricao de uma maquina de estados finitos \n");
 fprintf(escreve,"-- sintetizada pelo programa TABELA para o seu modelo
RTL descrito em VHDL. \n\n");
 fprintf(escreve,"-- Programa: SEQ2VHDL.EXE \n");
 fprintf(escreve,"-- Programador: Leandro de Oliveira Tancredo\n");
 fprintf(escreve,"-- Versão: 1.0 de 9 de maio de 2001 \n");
}

identifica_estados()
{
 int i,j,t,v;
 int size_linha,size_pacote,ff=0,sai=0,ent=0;
 int controle=1;
 char linha[81], aux_linha;
 char *pacote;
 fseek(le,0L,SEEK_SET); /* ponteiro para o inicio do arquivo */
 for(i=0;i<=7;i++)
 {
 fgets(linha,80,le);
 }

/*  stpcpy(aux_linha,linha); */
 size_linha = strlen(linha);
 for(j=0; j < size_linha; j++)
 {
  if(linha[j] == 'Q')
  {
   ff++;
  }
  if(linha[j] == 'Z')
  {
   sai++;
  }
 }

 do
```

```
 {
  fgets(linha,80,le);
  size_linha = strlen(linha);
  for(t=0; t < size_linha; t++)
  {
   if(linha[t] == '>')
   {
    controle = 0;
   }
  }
 } while(controle == 1);
 strrev(linha);
 pacote = strtok(linha, "->");
 size_pacote = strlen(pacote);
 for(v=0; v < size_pacote; v++)
 {
  if(pacote[v] == '0' || pacote[v] == '1' || pacote[v] == 'X')
  {
   ent++;
  }
 }
 num_estados = ff/2;
 num_saida = sai;
 num_entrada = ent - num_estados;
}

entity_entrada()
{
 int n_ent = num_entrada;
 int i, t;
 char linha[50] = " ", aux[20];
 strcat(linha,"X");
 t = n_ent - 1;
 for(i=0; i <= t; i++)
 {
  itoa(i,aux,10);
  strcat(linha,aux);
  if (i != n_ent-1)
  {
   strcat(linha,", X");
  }
 }
 strcat(linha," : IN BIT;");
 fprintf(escreve,"    %s\n",linha);
}

entity_estado()
{
 int n_est = num_estados;
 int i, t;
 char linha[50] = " ", aux[20];
 strcat(linha,"Q");
 t = n_est - 1;
 for(i=0; i <= t; i++)
 {
  itoa(i,aux,10);
  strcat(linha,aux);
```

```
  if (i != n_est-1)
  {
   strcat(linha,", Q");
  }
 }
 strcat(linha," : OUT BIT;");
 fprintf(escreve,"    %s\n",linha);
}

entity_saida()
{
 int n_sai = num_saida;
 int i, t;
 char linha[50] = " ", aux[20];
 strcat(linha,"Z");
 t = n_sai - 1;
 for(i=0; i <= t; i++)
 {
  itoa(i,aux,10);
  strcat(linha,aux);
  if (i != n_sai-1)
  {
   strcat(linha,", Z");
  }
 }
 strcat(linha," : OUT BIT );");
 fprintf(escreve,"    %s\n",linha);
}


cria_entity()
{
  fprintf(escreve,"\n\nENTITY %s IS\n",nome_entity);
  fprintf(escreve,"     PORT(\n");
  fprintf(escreve,"     -- CLK e CLR estão presentes em todos os
modelos\n");
  fprintf(escreve,"     -- CLK e CLR estão presentes em todos os
modelos\n");
  fprintf(escreve,"     CLK, CLR : IN BIT; \n\n");
  fprintf(escreve,"     -- Os parâmetros seguintes são definidos de
acordo \n");
  fprintf(escreve,"     -- com a descrição contida no programa tabela.
\n");
  fprintf(escreve,"     -- Xn representam as variáveis de entrada \n");
  fprintf(escreve,"     -- Qn representam as variáveis de estado \n");
  fprintf(escreve,"     -- Zn representam as funções de saída \n");
  if (num_entrada != 0)
  {
   entity_entrada();
  }
  entity_estado();
  if (num_saida != 0)
  {
   entity_saida();
  }
  fprintf(escreve,"END %s;\n\n",nome_entity);
}
```

```
aloca_mem_fila()
{
 struct fila_modelo *info_fila;
/* char *malloc();*/

 info_fila = (struct fila_modelo *) malloc(sizeof(size_modelo));
 if (info_fila == 0)
 {
  printf("\n O sistema nao dispoe de memoria \n");
  return;
 }
 new_fila = info_fila;
 new_fila->proximo = NULL;
}

cria_fila_modelos()
{
 aloca_mem_fila();
 cabeca = new_fila;
 rabo = new_fila;
}

insere_fila(tipo)
char tipo[10];
{
 char letra[2]=" ", indice[3]=" ";
 letra[0] = tipo[0];
 strcat(letra,"\0");
 indice[0] = tipo[1];
 strcat(indice,"\0");
 if(cabeca == rabo)
 {
  strcpy(cabeca->tipo,letra);
  strcpy(cabeca->indice,indice);
  rabo = NULL;
 }
 else
 {
  aloca_mem_fila();
  strcpy(new_fila->tipo,letra);
  strcpy(new_fila->indice,indice);
  new_fila->proximo = cabeca;
  cabeca = new_fila;
 }
}

sinal_controle()
{
  char linha[81], *result, *pacote, tipo_ff[15],funcao_cont[80]="SIGNAL
";
  int result_1, result_2, cond_final;
  fseek(le,0L,SEEK_SET); /* Ponteiro para o inicio do arquivo */
  while(fgets(linha,80,le) != NULL)
  {
    if((result = strstr(linha,"FUNCAO")) != NULL)
    {
```

```
     strrev(linha);
     pacote = strtok(linha," ");
     stpcpy(tipo_ff,pacote);
     strrev(tipo_ff);
     pacote = strtok(tipo_ff,"\n");
     stpcpy(tipo_ff,pacote);
     if( tipo_ff[0] != 'Z')
     {
      strcat(funcao_cont,tipo_ff);
      if( (tipo_ff[0] == 'J') || (tipo_ff[0] == 'D') )
      {
       insere_fila(tipo_ff);
      }
      result_1 = strcmp(tipo_ff,"D0");
      result_2 = strcmp(tipo_ff,"K0");
      if( (result_1 == 0) || (result_2 == 0) )
      {
       strcat(funcao_cont," : BIT;\n");
      }
      else
      {
       strcat(funcao_cont,", ");
      }
     }
    }
  }
  fprintf(escreve,"%s\n",funcao_cont);
}

modelo_JK(indice_ff)
char indice_ff[7];
{
 char VE[5]="VE", J[5]="J", K[5]="K", Q[5]="Q";
 strcat(VE,indice_ff);
 strcat(J,indice_ff);
 strcat(K,indice_ff);
 strcat(Q,indice_ff);
 fprintf(escreve,"-- Inferindo flip-flop tipo JK\n");
 fprintf(escreve,"PROCESS(CLK, CLR)\n");
 fprintf(escreve,"BEGIN\n");
 fprintf(escreve,"    IF CLR = '0' THEN\n");
 fprintf(escreve,"       %s <= '0';\n",VE);
 fprintf(escreve,"    ELSIF CLK'EVENT and CLK = '1' THEN\n");
 fprintf(escreve,"       IF %s = '0' and %s = '1' THEN\n",J,K);
 fprintf(escreve,"          %s <= '0';\n",VE);
 fprintf(escreve,"       ELSIF %s = '1' and %s = '0' THEN\n",J,K);
 fprintf(escreve,"          %s <= '1';\n",VE);
 fprintf(escreve,"       ELSIF %s = '1' and %s = '1' THEN\n",J,K);
 fprintf(escreve,"          %s <= not(%s);\n",VE,VE);
 fprintf(escreve,"       END IF;\n");
 fprintf(escreve,"    END IF;\n");
 fprintf(escreve,"    %s <= %s;\n",Q,VE);
 fprintf(escreve,"END PROCESS;\n\n");
}

modelo_D(indice_ff)
char indice_ff[7];
```

```
{
 char VE[5]="VE", D[5]="D", Q[5]="Q";
 strcat(VE,indice_ff);
 strcat(D,indice_ff);
 strcat(Q,indice_ff);
 fprintf(escreve,"-- Inferindo flip-flop tipo D\n");
 fprintf(escreve,"PROCESS(CLK, CLR)\n");
 fprintf(escreve,"BEGIN\n");
 fprintf(escreve,"    IF CLR = '0' THEN\n");
 fprintf(escreve,"       %s <= '0';\n",VE);
 fprintf(escreve,"    ELSIF CLK'EVENT and CLK = '1' THEN\n");
 fprintf(escreve,"       %s <= %s;\n",VE,D);
 fprintf(escreve,"    END IF;\n");
 fprintf(escreve,"    %s <= %s;\n",Q,VE);
 fprintf(escreve,"END PROCESS;\n\n");
}

ler_fila_tipos()
{
 struct fila_modelo *auxilio;

 auxilio = cabeca;
 while(auxilio != rabo)
 {
  if(auxilio->tipo[0] == 'J')
  {
   modelo_JK(auxilio->indice);
  }
  if(auxilio->tipo[0] == 'D')
  {
   modelo_D(auxilio->indice);
  }
  auxilio = auxilio->proximo;
 }
}

inferir_modelos()
{
 fprintf(escreve,"-- Nesta parte do modelo teremos a descrição de cada um
dos Flip-flops.\n");
 fprintf(escreve,"-- Deve-se observar que os sinais auxiliares são
utilizados, sendo que\n");
 fprintf(escreve,"-- no final de cada processo seus valores são
transferidos para as variáveis\n");
 fprintf(escreve,"-- de estados\n\n");
 fprintf(escreve,"-- Importante lembrar que em relação ao software,
existem duas procedures,\n");
 fprintf(escreve,"-- uma que implementa o flip-flop D e outra que
implementa o JK e estas procedures\n");
 fprintf(escreve,"-- recebem os índices que representam o respectivo
elemento de memoria. Tal índice esta\n");
 fprintf(escreve,"-- definido no arquivo gerado pelo programa
TABELA\n\n");
 ler_fila_tipos();
}

define_sinais()
```

```
{
 int i, num_est = num_estados-1;
 char aux[10]=" ", aux1[50]=" ";
 char linha[60] = "SIGNAL ";
 for(i=0; i<=num_est; i++)
 {
  itoa(i,aux,10);
  if(i < num_est)
  {
   strcat(aux1,"VE");
   strcat(aux1,aux);
   strcat(aux1,", ");
  }
  else
  {
   strcat(aux1,"VE");
   strcat(aux1,aux);
   strcat(aux1,": BIT;");
  }
 }
 strcat(linha,aux1);
 fprintf(escreve,"\n-- VEn são sinais auxiliares que assumem os mesmos
valores\n");
 fprintf(escreve,"-- das variáveis de estado. Eles sao utilizados para
permitir\n");
 fprintf(escreve,"-- um melhor modelamento do sistema\n");
 fprintf(escreve,"%s\n\n",linha);
 fprintf(escreve,"-- Jn, Kn e Dn representam as funcoes de controle e sao
definidas\n");
 fprintf(escreve,"-- no arquivo gerado pelo Programa TABELA\n");
 cria_fila_modelos();
 sinal_controle();
 fprintf(escreve,"\n BEGIN \n");
}

obtem_mintermos()
{
 int i, j, controle, size_pacote, conta_entrada, conta_estado;
 int num_variavel, num_aux, controle_fluxo, controle_and;
 int cont_1, T_irrelevante;
 char linha[81], aux_linha[50], aux[5], lixo[1];
 char *pacote, *result;
 fpos_t curpos;

 num_variavel = num_estados + num_entrada;
 controle_fluxo = 1;
 stpcpy(aux_2,"");
 cont_1 = 0;
 T_irrelevante = 0;
 while( (fgets(linha,80,le) != NULL) && (controle_fluxo == 1) )
 {
   if((result = strstr(linha,"ESSENCIAL")) != NULL)
  {
   strrev(linha);
   pacote = strtok(linha, "->  ");
   size_pacote = strlen(pacote);
   size_pacote = size_pacote - 1;
```

```
   stpcpy(aux_linha,pacote);
   strrev(aux_linha);
   for( j = 0; j < size_pacote; j++)
   {
     if ( aux_linha[j] != 'X' )
     {
      T_irrelevante = 1;
     }
   }
   conta_entrada = num_entrada;
   conta_estado = num_estados;
   controle_and = 0;
   for( i = 0; i < num_variavel; i++)
   {
     if ( (controle_and == 1) && (aux_linha[i] != '\n') && (aux_linha[i]
!= 'X'))
     {
      strcat(aux_2, " AND ");   /* esta passando por essa rotina
indevidamente */
     }
     conta_entrada = conta_entrada - 1;
     if (conta_entrada >= 0)
     {
      num_aux = conta_entrada;
      itoa(num_aux,aux,10);
      if(aux_linha[i] != 'X')
      {
      if (cont_1 == 0)
      {
/*     controle_and = 1;     */
       cont_1 = 1;
       strcat(aux_2, "( ");
      }
/*    controle_and = 1;       */
      if(aux_linha[i] == '0')
      {
       strcat(aux_2,"NOT(X");
         controle_and = 1;
      }
      else
      {
       strcat(aux_2,"(X");
         controle_and = 1;
      }
      strcat(aux_2,aux);
      strcat(aux_2,")");
       }
      }
     else /* Trabalha com as variaveis de estados */
     {
      conta_estado = conta_estado - 1;
      if (conta_estado >= 0)
      {
/*    controle_and = 1;     */
      num_aux = conta_estado;
      itoa(num_aux,aux,10);
      if(aux_linha[i] != 'X')
```

```
      {
         if (cont_1 == 0)
       {
        cont_1 = 1;
        strcat(aux_2, "( ");
       }
       if(aux_linha[i] == '0')
       {
        strcat(aux_2,"NOT(VE");
          controle_and = 1;
       }
       else
       {
        strcat(aux_2,"(VE");
          controle_and = 1;
       }
       strcat(aux_2,aux);
       strcat(aux_2,")");
/*
   Verifica se a proxima linha contem essencial incluir OR caso contrario
   incluir um ;
*/
       }
       if (conta_estado == 0)   /*  local de possivel problema */
       {
        if( fgetpos(le,&curpos) != 0) {}
      fgets(linha,80,le);
      if((result = strstr(linha,"ESSENCIAL")) != NULL)
      {
       strcat(aux_2,") OR (");
      }
        else
      {
       if (T_irrelevante == 0)
       {
        strcat(aux_2," '1';");
        controle_fluxo = 0;
       }
       else
       {
        strcat(aux_2, ");");
        controle_fluxo = 0;
       }
      }
      if( fsetpos(le,&curpos) != 0){}
       }
     }
    }
   }
  }
 }
}

funcoes_booleanas()
{
 int i, controle, size_linha;
 char linha[81], aux_linha[20];
```

```
 char *pacote, *result;
 fprintf(escreve," -- Processos que implementam as funcoes combinacionais
de controle \n ");
 fprintf(escreve," -- dos Elementos de memoria e de Saidas. \n\n ");
 fseek(le, 0L, SEEK_SET);
 for(i = 0;i < 6; i++)
 {
  fgets(linha,80,le);
 }
 while(fgets(linha,80,le) != NULL)
 {
  controle = 0;

  /* verifica as funcoes de controle e funcoes de saida */
  if((result = strstr(linha,"FUNCAO")) != NULL)
  {
   strrev(linha);
   pacote = strtok(linha, " ");
   strrev(pacote);
   size_linha = strlen(pacote) - 1;
   strncat(aux_linha,pacote,size_linha);
   controle = 1;

   if (controle == 1)
   {
/*     fprintf(escreve," %s <= (",aux_linha); */
     fprintf(escreve," %s <= ",aux_linha);
     obtem_mintermos();
     controle = 0;
     fprintf(escreve," %s \n ",aux_2);
     stpcpy(aux_2,"");
     stpcpy(aux_linha,"");
   }
  } /* Fecha o looping da funcao de controle */
 }
}

cria_arquitetura()
{
 fprintf(escreve,"-- RTL e a designacao para todas as arquiteturas\n");
 fprintf(escreve,"ARCHITECTURE RTL OF %s IS\n",nome_entity);
 define_sinais();
 inferir_modelos();
 funcoes_booleanas();
 fprintf(escreve,"\n");
 fprintf(escreve,"END RTL;");
}
main()
{
  clrscr();
  abre_arq();
  cria_cabecalho();
  identifica_estados();
  cria_entity();
  cria_arquitetura();
  fecha_arq();
};
```

APÊNDICE III

EXEMPLO DA LISTAGEM DE ALGUNS CÓDIGOS DE LINHA

A. CÓDIGO HDB3

HDB3.TAB

```
1 5                 25 8 4 1 0          49 20 30 1 0
2 D                 26 9 8 0 1          50 21 18 0 1
3 D                 27 9 4 1 1          51 21 30 1 1
4 D                 28 10 9 0 0         52 22 19 0 0
5 D                 29 10 5 1 0         53 22 31 1 0
6 D                 30 11 9 0 3         54 23 19 0 3
7 1 2               31 11 5 1 3         55 23 31 1 3
8 0 20 0 0          32 12 10 0 0        56 24 12 0 0
9 0 12 1 0          33 12 6 1 0         57 24 20 1 0
10 1 0 0 3          34 13 10 0 3        58 25 24 0 1
11 1 12 1 3         35 13 6 1 3         59 25 20 1 1
12 2 1 0 0          36 14 11 0 0        60 26 25 0 0
13 2 13 1 0         37 14 7 1 0         61 26 21 1 0
14 3 1 0 1          38 15 11 0 1        62 27 25 0 3
15 3 13 1 1         39 15 7 1 1         63 27 21 1 3
16 4 2 0 0          40 16 12 0 1        64 28 26 0 0
17 4 14 1 0         41 16 28 1 0        65 28 22 1 0
18 5 2 0 1          42 17 16 0 3        66 29 26 0 3
19 5 14 1 1         43 17 28 1 3        67 29 22 1 3
20 6 3 0 0          44 18 17 0 0        68 30 27 0 0
21 6 15 1 0         45 18 29 1 0        69 30 23 1 0
22 7 3 0 3          46 19 17 0 1        70 31 27 0 1
23 7 15 1 3         47 19 29 1 1        71 31 23 1 1
24 8 20 0 3         48 20 18 0 0        72 -100
```

HDB3.TXT

```
      CASO    :  HDB3 – Modelagem Funcional
     DATA    :  03 03 2002
     USUARIO :  Leandro de Oliveira Tancredo

!DE!P/!ENTRADA
!MINT!!Q4!Q4+!D4!!Q3!Q3+!D3!!Q2!Q2+!D2!!Q1!Q1+!D1!!Q0!Q0+!D0!!Z1!Z0!
!31!23! 1 (1)  !  63!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 1!  1! 1!! 1!  1!
1!! 0! 1!
!31!27! 0 (0)  !  31!! 1!  1! 1!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 1!  1!
1!! 0! 1!
```

```
!30!23! 1 (1)  !  62!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!30!27! 0 (0)  !  30!! 1!  1! 1!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!29!22! 1 (1)  !  61!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 0!  1! 1!! 1!  0!
0!! 1! 1!
!29!26! 0 (0)  !  29!! 1!  1! 1!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 1!  0!
0!! 1! 1!
!28!22! 1 (1)  !  60!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!28!26! 0 (0)  !  28!! 1!  1! 1!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!27!21! 1 (1)  !  59!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 1!  0! 0!! 1!  1!
1!! 1! 1!
!27!25! 0 (0)  !  27!! 1!  1! 1!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 1!  1!
1!! 1! 1!
!26!21! 1 (1)  !  58!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 1!  0! 0!! 0!  1!
1!! 0! 0!
!26!25! 0 (0)  !  26!! 1!  1! 1!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 0!  1!
1!! 0! 0!
!25!20! 1 (1)  !  57!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 0!  0! 0!! 1!  0!
0!! 0! 1!
!25!24! 0 (0)  !  25!! 1!  1! 1!! 1!  1! 1!! 0!  0! 0!! 0!  0! 0!! 1!  0!
0!! 0! 1!
!24!20! 1 (1)  !  56!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!
!24!12! 0 (0)  !  24!! 1!  0! 0!! 1!  1! 1!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!
!23!31! 1 (1)  !  55!! 1!  1! 1!! 0!  1! 1!! 1!  1! 1!! 1!  1! 1!! 1!  1!
1!! 1! 1!
!23!19! 0 (0)  !  23!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 1!  1!
1!! 1! 1!
!22!31! 1 (1)  !  54!! 1!  1! 1!! 0!  1! 1!! 1!  1! 1!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!22!19! 0 (0)  !  22!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!21!30! 1 (1)  !  53!! 1!  1! 1!! 0!  1! 1!! 1!  1! 1!! 0!  1! 1!! 1!  0!
0!! 0! 1!
!21!18! 0 (0)  !  21!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 1!  0!
0!! 0! 1!
!20!30! 1 (1)  !  52!! 1!  1! 1!! 0!  1! 1!! 1!  1! 1!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!20!18! 0 (0)  !  20!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!19!29! 1 (1)  !  51!! 1!  1! 1!! 0!  1! 1!! 0!  1! 1!! 1!  0! 0!! 1!  1!
1!! 0! 1!
!19!17! 0 (0)  !  19!! 1!  1! 1!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 1!  1!
1!! 0! 1!
!18!29! 1 (1)  !  50!! 1!  1! 1!! 0!  1! 1!! 0!  1! 1!! 1!  0! 0!! 0!  1!
1!! 0! 0!
!18!17! 0 (0)  !  18!! 1!  1! 1!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 0!  1!
1!! 0! 0!
!17!28! 1 (1)  !  49!! 1!  1! 1!! 0!  1! 1!! 0!  1! 1!! 0!  0! 0!! 1!  0!
0!! 1! 1!
!17!16! 0 (0)  !  17!! 1!  1! 1!! 0!  0! 0!! 0!  0! 0!! 0!  0! 0!! 1!  0!
0!! 1! 1!
```

```
!16!28! 1 (1)  !  48!! 1!  1! 1!! 0!  1! 1!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!
!16!12! 0 (0)  !  16!! 1!  0! 0!! 0!  1! 1!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 1!
!15! 7! 1 (1)  !  47!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 1!  1! 1!! 1!  1!
1!! 0! 1!
!15!11! 0 (0)  !  15!! 0!  0! 0!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 1!  1!
1!! 0! 1!
!14! 7! 1 (1)  !  46!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!14!11! 0 (0)  !  14!! 0!  0! 0!! 1!  1! 1!! 1!  0! 0!! 1!  1! 1!! 0!  1!
1!! 0! 0!
!13! 6! 1 (1)  !  45!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 0!  1! 1!! 1!  0!
0!! 1! 1!
!13!10! 0 (0)  !  13!! 0!  0! 0!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 1!  0!
0!! 1! 1!
!12! 6! 1 (1)  !  44!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!12!10! 0 (0)  !  12!! 0!  0! 0!! 1!  1! 1!! 1!  0! 0!! 0!  1! 1!! 0!  0!
0!! 0! 0!
!11! 5! 1 (1)  !  43!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 1!  0! 0!! 1!  1!
1!! 1! 1!
!11! 9! 0 (0)  !  11!! 0!  0! 0!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 1!  1!
1!! 1! 1!
!10! 5! 1 (1)  !  42!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 1!  0! 0!! 0!  1!
1!! 0! 0!
!10! 9! 0 (0)  !  10!! 0!  0! 0!! 1!  1! 1!! 0!  0! 0!! 1!  0! 0!! 0!  1!
1!! 0! 0!
! 9! 4! 1 (1)  !  41!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 0!  0! 0!! 1!  0!
0!! 0! 1!
! 9! 8! 0 (0)  !   9!! 0!  0! 0!! 1!  1! 1!! 0!  0! 0!! 0!  0! 0!! 1!  0!
0!! 0! 1!
! 8! 4! 1 (1)  !  40!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!
! 8!20! 0 (0)  !   8!! 0!  1! 1!! 1!  0! 0!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 1! 1!
! 7!15! 1 (1)  !  39!! 0!  0! 0!! 0!  1! 1!! 1!  1! 1!! 1!  1! 1!! 1!  1!
1!! 1! 1!
! 7! 3! 0 (0)  !   7!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 1!  1!
1!! 1! 1!
! 6!15! 1 (1)  !  38!! 0!  0! 0!! 0!  1! 1!! 1!  1! 1!! 1!  1! 1!! 0!  1!
1!! 0! 0!
! 6! 3! 0 (0)  !   6!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 1!  1! 1!! 0!  1!
1!! 0! 0!
! 5!14! 1 (1)  !  37!! 0!  0! 0!! 0!  1! 1!! 1!  1! 1!! 0!  1! 1!! 1!  0!
0!! 0! 1!
! 5! 2! 0 (0)  !   5!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 1!  0!
0!! 0! 1!
! 4!14! 1 (1)  !  36!! 0!  0! 0!! 0!  1! 1!! 1!  1! 1!! 0!  1! 1!! 0!  0!
0!! 0! 0!
! 4! 2! 0 (0)  !   4!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 0!  1! 1!! 0!  0!
0!! 0! 0!
! 3!13! 1 (1)  !  35!! 0!  0! 0!! 0!  1! 1!! 0!  1! 1!! 1!  0! 0!! 1!  1!
1!! 0! 1!
! 3! 1! 0 (0)  !   3!! 0!  0! 0!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 1!  1!
1!! 0! 1!
```

```
! 2!13! 1 (1)  !  34!! 0!  0! 0!! 0!  1! 1!! 0!  1! 1!! 1!  0! 0!! 0!  1!
1!! 0! 0!
! 2! 1! 0 (0)  !   2!! 0!  0! 0!! 0!  0! 0!! 0!  0! 0!! 1!  0! 0!! 0!  1!
1!! 0! 0!
! 1!12! 1 (1)  !  33!! 0!  0! 0!! 0!  1! 1!! 0!  1! 1!! 0!  0! 0!! 1!  0!
0!! 1! 1!
! 1! 0! 0 (0)  !   1!! 0!  0! 0!! 0!  0! 0!! 0!  0! 0!! 0!  0! 0!! 1!  0!
0!! 1! 1!
! 0!12! 1 (1)  !  32!! 0!  0! 0!! 0!  1! 1!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!
! 0!20! 0 (0)  !   0!! 0!  1! 1!! 0!  0! 0!! 0!  1! 1!! 0!  0! 0!! 0!  0!
0!! 0! 0!

FUNCAO D4
=========

MINTERMOS :  0;  8; 48; 17; 49; 18; 50; 19; 51; 20; 52; 21; 53; 22; 54;
23; 55; 56; 25; 57; 26; 58; 27; 59; 28; 60; 29; 61; 30; 62; 31; 63;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 20     REDUNDANCIA:      43 ->  X1X1XX
ESSENCIAL: 18     REDUNDANCIA:      45 ->  X1XX1X
ESSENCIAL: 17     REDUNDANCIA:      46 ->  X1XXX1
ESSENCIAL: 48     REDUNDANCIA:      15 ->  11XXXX
ESSENCIAL:  0     REDUNDANCIA:       8 ->  00X000
CUSTO FINAL DE D4 = 18

FUNCAO D3
=========

MINTERMOS : 32; 33; 34; 35; 36; 37; 38; 39;  9; 10; 11; 12; 13; 14; 15;
16; 48; 49; 50; 51; 52; 53; 54; 55; 24; 25; 26; 27; 28; 29; 30; 31;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 16     REDUNDANCIA:       8 ->  01X000
ESSENCIAL: 12     REDUNDANCIA:      19 ->  0X11XX
ESSENCIAL: 10     REDUNDANCIA:      21 ->  0X1X1X
ESSENCIAL:  9     REDUNDANCIA:      22 ->  0X1XX1
ESSENCIAL: 32     REDUNDANCIA:      23 ->  1X0XXX
CUSTO FINAL DE D3 = 21
FUNCAO D2
=========

MINTERMOS :  0; 32; 33; 34; 35; 36; 37; 38; 39;  8; 40; 41; 42; 43; 44;
45; 46; 47; 16; 48; 49; 50; 51; 52; 53; 54; 55; 24; 56; 57; 58; 59; 60;
61; 62; 63;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 32     REDUNDANCIA:      31 ->  1XXXXX
ESSENCIAL:  0     REDUNDANCIA:      56 ->  XXX000
CUSTO FINAL DE D2 = 6

FUNCAO D1
=========

MINTERMOS :  4; 36;  5; 37;  6; 38;  7; 39; 12; 44; 13; 45; 14; 46; 15;
47; 20; 52; 21; 53; 22; 54; 23; 55; 28; 60; 29; 61; 30; 62; 31; 63;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  4     REDUNDANCIA:      59 ->  XXX1XX
CUSTO FINAL DE D1 = 1
```

```
FUNCAO D0
=========

MINTERMOS :  2; 34;  3; 35;  6; 38;  7; 39; 10; 42; 11; 43; 14; 46; 15;
47; 18; 50; 19; 51; 22; 54; 23; 55; 26; 58; 27; 59; 30; 62; 31; 63;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL:  2     REDUNDANCIA:      61 ->  XXXX1X
CUSTO FINAL DE D0 = 1

FUNCAO Z1
=========

MINTERMOS :  1; 33;  7; 39;  8; 11; 43; 13; 45; 17; 49; 23; 55; 27; 59;
29; 61;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 13     REDUNDANCIA:      48 ->  XX1101
ESSENCIAL: 11     REDUNDANCIA:      48 ->  XX1011
ESSENCIAL:  8     REDUNDANCIA:       0 ->  001000
ESSENCIAL:  7     REDUNDANCIA:      48 ->  XX0111
ESSENCIAL:  1     REDUNDANCIA:      48 ->  XX0001
CUSTO FINAL DE Z1 = 27

FUNCAO Z0
=========

MINTERMOS :  1; 33;  3; 35;  5; 37;  7; 39;  8;  9; 41; 11; 43; 13; 45;
15; 47; 16; 17; 49; 19; 51; 21; 53; 23; 55; 25; 57; 27; 59; 29; 61; 31;
63;
IMPLICANTES PRIMOS ESSENCIAIS :
ESSENCIAL: 16     REDUNDANCIA:       1 ->  01000X
ESSENCIAL:  8     REDUNDANCIA:       1 ->  00100X
ESSENCIAL:  1     REDUNDANCIA:      62 ->  XXXXX1
CUSTO FINAL DE Z0 = 14
CUSTO TOTAL DAS 7 FUNCOES = 88
```

O arquivo se inicia com um cabeçalho que permite identificar o caso rodado, o usuário e a data em que foi executado. Logo a seguir é apresentada a tabela de transição de estados da onde são extraídas as funções de controle dos elementos de memória e das saídas.

São apresentados todos os mintermos das funções booleanas e os respectivos implicantes primos. Por exemplo a função D_0 é composta pelos mintermos 2; 34; 3; 35; 6; 38; 7; 39; 10; 42; 11; 43; 14; 46; 15; 47; 18; 50; 19; 51; 22; 54; 23; 55; 26; 58; 27; 59; 30; 62; 31; 63;. Os implicantes primos gera a função mínima representada por Q_1, a função D_1 é composta pelo implicante Q_2, a função D_2 é composta pelo implicante X + $Q_2' Q_1' Q_0'$, a função D_3 é composta pelo implicante $X' Q_4 Q_2' Q_1' Q_0'$ + $X' Q_3 Q_2$ + $X' Q_3 Q_1$ + $X' Q_3 Q_0$ + Q_3',

, função D_4 é composta pelo implicante , $Q_4Q_2 + Q_4Q_1 + Q_4Q_0 + XQ_4 + XQ_4'Q_2'Q_1'Q_0'$,Z1 é composta pelo implicante $Q_3Q_2Q_1'Q_0 + Q_3Q_2'Q_1Q_0 + XQ_4'Q_3Q_2'Q_1'Q_0' + Q_4'Q_3Q_2Q_1 + Q_4'Q_3'Q_2'Q_1$ e a função Z0 é composta pelo implicante $X'Q_4Q_3'Q_2'Q_1' + X'Q_4'Q_3Q_2'Q_1' + Q_0$. O custo total para a implementação das três funções combinacionais é igual a 88, considerando-se como critério de custo a quantidade de entrada das portas lógicas AND e OR utilizadas.

HDB3.VHDL

Vamos executar o SEQ2VHDL que recebe como entrada a saída gerada pelo programa Tabela e criar um modelo funcional do circuito projetado na linguagem de descrição VHDL para o ambiente MAX + PLUS II

```
-- PROJETO DE TESE DE MESTRADO - Ferramenta de Síntese Lógica
-- A ferramenta tem por objetivo transformar a descrição de
uma maquina de estados finitos
-- sintetizada pelo programa TABELA para o seu modelo RTL
descrito em VHDL.

-- Programa: SEQ2VHDL.EXE
-- Programador: Leandro de Oliveira Tancredo
-- Versao: 1.0 de 03 de março de 2002


ENTITY Hdb3 IS
     PORT(
     -- CLK e CLR estão presentes em todos os modelos
     -- CLK e CLR estão presentes em todos os modelos
     CLK, CLR : IN BIT;
     -- Os parâmetros seguintes são definidos de acordo
     -- com a descrição contida no programa tabela.
     -- Xn representam as variáveis de entrada
     -- Qn representam as variáveis de estado
     -- Zn representam as funções de saída
     X0 : IN BIT;
     Q0, Q1, Q2, Q3, Q4 : OUT BIT;
     Z0 : OUT BIT );
END Hdb3;
-- RTL e a designação para todas as arquiteturas
ARCHITECTURE RTL OF Hdb3 IS
```

```
-- VEn são sinais auxiliares que assumem os mesmos valores
-- das variáveis de estado. Eles são utilizados para permitir
-- um melhor modelamento do sistema
SIGNAL  VE0, VE1, VE2, VE3, VE4: BIT;
-- Jn, Kn e Dn representam as funções de controle e são
definidas
-- no arquivo gerado pelo Programa TABELA
SIGNAL D4, D3, D2, D1, D0 : BIT;
Z0, Z1 : OUT BIT );
 BEGIN
-- Nesta parte do modelo teremos a descricao de cada um dos
Flip-flops.
-- Deve-se observar que os sinais auxiliares são utilizados,
sendo que
-- no final de cada processo seus valores sao transferidos
para as variáveis
-- de estados
-- Importante lembrar que em relação ao software, existem
duas procedures,
-- uma que implementa o flip-flop D e outra que implementa o
JK e estas procedures
-- recebem os índices que representam o respectivo elemento
de memória. Tal índice esta
-- definido no arquivo gerado pelo programa TABELA

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE0 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE0 <= D0;
    END IF;
    Q0 <= VE0;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE1 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE1 <= D1;
    END IF;
    Q1 <= VE1;
END PROCESS;
```

```
-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE2 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE2 <= D2;
    END IF;
    Q2 <= VE2;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE3 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE3 <= D3;
    END IF;
    Q3 <= VE3;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
    IF CLR = '0' THEN
       VE4 <= '0';
    ELSIF CLK'EVENT and CLK = '1' THEN
       VE4 <= D4;
    END IF;
    Q4 <= VE4;
END PROCESS;

 -- Processos que implementam as funções combinacionais de
controle
  -- dos Elementos de memória e de Saídas.
  D4 <=  ( (VE4) AND (VE2)) OR ((VE4) AND (VE1)) OR ((VE4)
AND (VE0)) OR ((X0) AND (VE4)) OR (NOT(X0) AND NOT(VE4) AND
NOT(VE2) AND NOT(VE1) AND NOT(VE0));
  D3 <=  ( NOT(X0) AND (VE4) AND NOT(VE2) AND NOT(VE1) AND
NOT(VE0)) OR (NOT(X0) AND (VE3) AND (VE2)) OR (NOT(X0) AND
(VE3) AND (VE1)) OR (NOT(X0) AND (VE3) AND (VE0)) OR ((X0)
AND NOT(VE3));
  D2 <=  ( (X0)) OR (NOT(VE2) AND NOT(VE1) AND NOT(VE0));
  D1 <=  ( (VE2));
  D0 <=  ( (VE1));
```

```
  Z1 <=  ( (VE3) AND (VE2) AND NOT(VE1) AND (VE0)) OR ((VE3)
AND NOT(VE2) AND (VE1) AND (VE0)) OR (NOT(X0) AND NOT(VE4)
AND (VE3) AND NOT(VE2) AND NOT(VE1) AND NOT(VE0)) OR
(NOT(VE3) AND (VE2) AND (VE1) AND (VE0)) OR (NOT(VE3) AND
NOT(VE2) AND NOT(VE1) AND (VE0));
  Z0 <=  ( NOT(X0) AND VE4 AND NOT(VE3) AND NOT(VE2) AND
NOT(VE1) OR NOT(X0) AND NOT(VE4)AND VE3 AND NOT(VE2) AND
NOT(VE1) OR VE);
END RTL;
```

B. CÓDIGO VHDL GERADO POR OUTRAS FERRAMENTAS

Logo abaixo, apresenta-se o código MLT-3 gerado pelo ambiente AC:

MLT3AC.VHD

```
--  File: c:\My Designs\mlt3ahdl\compile\component_1.vhd
-- created: 07/17/02 23:14:30
--  from: 'c:\My Designs\mlt3ahdl\src\component_1.VHD'
--  by fsm2hdl - version: 2.0.1.80
--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mlt3ac is
  port (CLK: in STD_LOGIC;
        Q: in STD_LOGIC;
        Z1: out STD_LOGIC;
        Z2: out STD_LOGIC);
end;

architecture component_1_arch of mlt3ac is

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S0, S1, S2, S3);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments


Sreg0_machine: process (CLK)
```

```
begin
if CLK'event and CLK = '1' then
     case Sreg0 is
          when S0 =>
               if Q='1' then
                    Sreg0 <= S3;
               elsif Q='0' then
                    Sreg0 <= S0;
               end if;
          when S1 =>
               if Q='1' then
                    Sreg0 <= S0;
               elsif Q='0' then
                    Sreg0 <= S1;
               end if;
          when S2 =>
               if Q='1' then
                    Sreg0 <= S1;
               elsif Q='0' then
                    Sreg0 <= S2;
               end if;
          when S3 =>
               if Q='0' then
                    Sreg0 <= S3;
               elsif Q='1' then
                    Sreg0 <= S2;
               end if;
          when others =>
               null;
     end case;
end if;
end process;

-- signal assignment statements for combinatorial outputs
Z1_assignment:
Z1 <= '1' when (Sreg0 = S0 and Q='1') else
      '0' when (Sreg0 = S0 and Q='0') else
      '0' when (Sreg0 = S1 and Q='1') else
      '1' when (Sreg0 = S1 and Q='0') else
      '0' when (Sreg0 = S2 and Q='1') else
      '0' when (Sreg0 = S2 and Q='0') else
      '1' when (Sreg0 = S3 and Q='0') else
      '0' when (Sreg0 = S3 and Q='1') else
      '0';
```

```
Z2_assignment:
Z2 <= '1' when (Sreg0 = S0 and Q='1') else
      '0' when (Sreg0 = S0 and Q='0') else
      '0' when (Sreg0 = S1 and Q='1') else
      '0' when (Sreg0 = S1 and Q='0') else
      '1' when (Sreg0 = S2 and Q='1') else
      '0' when (Sreg0 = S2 and Q='0') else
      '1' when (Sreg0 = S3 and Q='0') else
      '0' when (Sreg0 = S3 and Q='1') else
      '0';

end component_1_arch;
```

A seguir o código gerado pela ferramenta SC:

MLT3SC.VHD

```
--C:\ESTUDOS\MONOGRAFIA\ARTIGOS\ \MLT3.vhd
--  VHDL code created
--  Mon Jul 15 08:43:03 2002

--  This VHDL code was generated using:
--  one-hot state assignment with boolean code format.
--  Minimization is enabled,  implied else is enabled,
--  and outputs are manually optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- LIBRARY synopsys;
-- USE synopsys.attributes.all;

ENTITY MLT3sc IS
        PORT (CLK,Q: IN std_logic;
                Z1,Z2 : OUT std_logic);
END mlt3sc;

ARCHITECTURE BEHAVIOR OF MLT3sc IS
--      State variables for machine sreg
SIGNAL STATE0, next_STATE0, STATE1, next_STATE1, STATE2, next_STATE2, STATE3
                , next_STATE3 : std_logic;
BEGIN
        PROCESS (CLK, next_STATE0, next_STATE1, next_STATE2, next_STATE3)
        BEGIN
                IF CLK='1' AND CLK'event THEN
                        STATE0 <= next_STATE0;
                        STATE1 <= next_STATE1;
                        STATE2 <= next_STATE2;
                        STATE3 <= next_STATE3;
                END IF;
```

```
        END PROCESS;

        PROCESS (Q,STATE0,STATE1,STATE2,STATE3)
        BEGIN

        IF (( Q='0' AND  (STATE0='1')) OR ( Q='1' AND  (STATE1='1'))) THEN
                        next_STATE0<='1';
                ELSE next_STATE0<='0';
                END IF;

        IF (( Q='0' AND  (STATE1='1')) OR ( Q='1' AND  (STATE2='1'))) THEN
                        next_STATE1<='1';
                ELSE next_STATE1<='0';
                END IF;

        IF (( Q='0' AND  (STATE2='1')) OR ( Q='1' AND  (STATE3='1'))) THEN
                        next_STATE2<='1';
                ELSE next_STATE2<='0';
                END IF;

        IF (( Q='1' AND  (STATE0='1')) OR ( Q='0' AND  (STATE3='1'))) THEN
                        next_STATE3<='1';
                ELSE next_STATE3<='0';
                END IF;


        IF (( Q='1' AND  (STATE0='1')) OR ( Q='0' AND  (STATE3='1'))) THEN Z1<='1';
                ELSE Z1<='0';
                END IF;

        IF (( Q='1' AND  (STATE0='1')) OR ( Q='0' AND  (STATE1='1')) OR ( Q='1' AND
                          (STATE2='1')) OR ( Q='0' AND  (STATE3='1'))) THEN Z2<='1';
                ELSE Z2<='0';
                END IF;
        END PROCESS;
END BEHAVIOR;
```

Logo abaixo temos a listagem do código mlt3 gerado pelo tabela:

MLT3TAB.VHD

```
-- PROJETO DE TESE DE MESTRADO - Ferramenta de Síntese Lógica
-- A ferramenta tem por objetivo transformar a descrição de uma maquina de estados finitos
-- sintetizada pelo programa TABELA para o seu modelo RTL descrito em VHDL.

-- Programa: SEQ2VHDL.EXE
-- Programador: Leandro de Oliveira Tancredo
-- Versão: 1.0 de 9 de maio de 2001
```

```
ENTITY mlt3 IS
    PORT(
    -- CLK e CLR estão presentes em todos os modelos
    -- CLK e CLR estão presentes em todos os modelos
    CLK, CLR : IN BIT;

    -- Os parâmetros seguintes são definidos de acordo
    -- com a descrição contida no programa tabela.
    -- Xn representam as variáveis de entrada
    -- Qn representam as variáveis de estado
    -- Zn representam as funções de saída
    X0 : IN BIT;
    Q0, Q1 : OUT BIT;
    Z0, Z1 : OUT BIT );
END mlt3;

-- RTL e a designação para todas as arquiteturas
ARCHITECTURE RTL OF mlt3 IS

-- VEn são sinais auxiliares que assumem os mesmos valores
-- das variáveis de estado. Eles são utilizados para permitir
-- um melhor modelamento do sistema
SIGNAL  VE0, VE1: BIT;

-- Jn, Kn e Dn representam as funções de controle e são definidas
-- no arquivo gerado pelo Programa TABELA
SIGNAL D1, D0 : BIT;


 BEGIN
-- Nesta parte do modelo teremos a descrição de cada um dos Flip-flops.
-- Deve-se observar que os sinais auxiliares são utilizados, sendo que
-- no final de cada processo seus valores são transferidos para as variáveis
-- de estados

-- Importante lembrar que em relação ao software, existem duas procedures,
-- uma que implementa o flip-flop D e outra que implementa o JK e estas procedures
```

```
-- recebem os índices que representam o respectivo elemento de memória. Tal índice esta
-- definido no arquivo gerado pelo programa TABELA
-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
      VE0 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
      VE0 <= D0;
   END IF;
   Q0 <= VE0;
END PROCESS;

-- Inferindo flip-flop tipo D
PROCESS(CLK, CLR)
BEGIN
   IF CLR = '0' THEN
      VE1 <= '0';
   ELSIF CLK'EVENT and CLK = '1' THEN
      VE1 <= D1;
   END IF;
   Q1 <= VE1;
END PROCESS;

 -- Processos que implementam as funções combinacionais de controle
  -- dos Elementos de memória e de Saídas.

  D1 <=  ( NOT(X0) AND (VE1)) OR ((X0) AND NOT(VE1) AND NOT(VE0));
  D0 <=  ( (X0) AND NOT(VE0)) OR (NOT(X0) AND (VE0));
  Z1 <=  ( NOT(X0) AND (VE1) AND (VE0)) OR ((X0) AND NOT(VE1) AND NOT(VE0));
  Z0 <=  ( (X0) AND NOT(VE0)) OR (NOT(X0) AND (VE0));

END RTL;
```