UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

DESENVOLVIMENTO DE UM IP CORE DE PRÉ-PROCESSAMENTO DIGITAL DE SINAIS DE VOZ PARA APLICAÇÃO EM SISTEMAS EMBUTIDOS

DANIELLA DIAS CAVALCANTE DA SILVA

ORIENTADORES

ELMAR UWE KURT MELCHER

JOSEANA MACÊDO FECHINE

CAMPINA GRANDE

JULHO – 2006

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DESENVOLVIMENTO DE UM IP CORE DE PRÉ-PROCESSAMENTO DIGITAL DE SINAIS DE VOZ PARA APLICAÇÃO EM SISTEMAS EMBUTIDOS

Daniella Dias Cavalcante da Silva

Orientadores
Elmar Uwe Kurt Melcher
Joseana Macêdo Fechine

Dissertação submetida à Coordenação do Curso de Pós-graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de mestre em Informática.

Área de concentração: Ciência da Computação.

Linha de pesquisa: Redes de Computadores e Sistemas Distribuídos.

CAMPINA GRANDE
JULHO – 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S586d Silva, Daniella Dias Cavalcante da
2006 Desenvolvimento de um IP Core de pré-processamento digital de sinais de voz para aplicação em sistemas embutidos/ Daniella Dias Cavalcante da Silva. ─ Campina Grande, 2006.
92fs.: il.

Referências.
Dissertação (Mestrado em Informática) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Elmar Uwe Kurt Melcher e Joseana Macêdo Fechine.

1─ Sistemas Embutidos (embedded) 2─ IP Core 3 ─ Processamento Digital de Sinais de Voz 4 ─ Prototipação I ─ Título

CDU 004.031.6

DESENVOLVIMENTO DE UM IP CORE DE PRÉ-PROCESSAMENTO DIGITAL DE SINAIS DE VOZ PARA APLICAÇÃO EM SISTEMAS EMBUTIDOS

Daniella Dias Cavalcante da Silva

Elmar Uwe Kurt Melcher
Orientador

Joseana Macêdo Fechine
Orientadora

Benedito Guimarães Aguiar Neto
Componente da Banca

José Antônio Gomes de Lima
Componente da Banca

José Eustáquio Rangel de Queiroz
Componente da Banca

Campina Grande, Paraíba, Brasil

"As máquinas, um dia, talvez
venham a pensar.
Mas nunca terão sonhos."
Theodor Heuss

“Nada vem de graça.
Nem o pão, nem a cachaça.”
Zeca Baleiro

AGRADECIMENTOS

São tantas pessoas para agradecer, que qualquer ordem de citação que eu utilize, será injusta. Então, vou tentar usar uma ordem mais ou menos cronológica.

Primeiramente a Deus, pela vida e por mais uma grande conquista.

Aos meus queridos pais, D. Deri e Alex, pela constante confiança e incentivo, carinho, paciência e compreensão nas minhas muitas ausências físicas, mas nunca espirituais.

Aos Caselas: Anderson (Fofinho), Luciana (Luly), Severo (Severinho), Reinaldo (Reizinho) e Ramide (Ramufa), pelo companheirismo, amizade e apoio tanto na batalha para ingressar no mestrado, quanto na luta para concluir.

Ao amigo Valterivan, sem a ajuda do qual não teria chegado em tempo para a entrevista de seleção. Também por sua constante confiança e incentivo.

Aos meus orientadores e verdadeiros mestres, Joseana e Elmar, pela orientação, críticas, confiança, presteza e incentivo nos momentos mais difíceis e nebulosos. Elmar, não esquecerei sua frase "Tenha fé!". Joseana, entre os vários momentos de incentivo, o daquela apresentação dos três leões, faltando cinco minutos para uma reunião, ficará para sempre na memória.

A todos do LAD: Karinona, Isaac, Karinina, Fagner, Wilson, George, Zurita e Matheus. Em especial, a **Henrique** (em negrito, conforme prometido), pela constante ajuda e paciência, e que mesmo diante de tantas tarefas a cumprir, parava para esclarecer todas as minhas dúvidas. E Karinona que, mesmo estando na Alemanha e sem nos conhecermos pessoalmente, me transmitia sua confiança, força e energia.

Aos amigos e anjos, que Deus me apresentou aqui em Campina Grande: Nadir, Nara, Nilton, Leidjane e Geo. A aqueles que reencontrei: Fabiana e Isabela. Isa e Geo, aquela disciplina de PDS sem vocês... E a Artur, meu guia em Campina Grande.

A Silvana que, de minha professora transformou-se em uma grande amiga, conselheira e confidente.

Ao colega José Gomes Cipriano pelos esclarecimentos prestados e material disponibilizado, que foram muito importantes para o desenvolvimento deste trabalho.

Ao meu primão e irmão de coração Leandro, que me fazia enxergar cada passo dado como mais um degrau em direção ao topo!

A Aninha, da Copin, sempre disposta a ajudar e resolver todos os problemas que surgissem.

À Capes, pelo financiamento deste trabalho.

Não foi nada fácil chegar até aqui, mas certamente teria sido *impossível* sem a ajuda de todos vocês.

Muito obrigada!

RESUMO

A fala é o meio de comunicação comumente utilizado pelo homem, que o distingue dos demais seres vivos, permitindo-lhe a troca de idéias, expressão de opiniões ou revelação de seu pensamento. Diante do avanço tecnológico e conseqüente surgimento de equipamentos eletrônicos cada vez mais sofisticados, a possibilidade de permitir a interação homem-máquina através da voz tem sido objeto de grande interesse, tanto do meio acadêmico quanto dos fabricantes de tais equipamentos. Pesquisas na área de Processamento Digital de Sinais de Voz têm permitido o desenvolvimento de sistemas de Resposta Vocal, Reconhecimento de Voz e Reconhecimento de Identidade Vocal. Entretanto, requisitos de processamento ainda dificultam a implementação desses sistemas em dispositivos com baixo poder computacional, como celulares, *palmtops* e eletrodomésticos. O trabalho desenvolvido consiste do estudo e adaptação de técnicas de processamento digital de sinais de voz, resultando em uma biblioteca de pré-processamento, incluindo as funções de pré-ênfase, divisão em quadros e janelamento, de maneira a permitir sua utilização no desenvolvimento de aplicações embutidas de reconhecimento de voz ou locutor. Foram realizadas adaptações dos modelos necessários à realização das funções, implementação em uma linguagem de descrição de *hardware,* verificação funcional da biblioteca e, por fim, prototipação em um dispositivo de *hardware*.

ABSTRACT

Speech is the most common way of communication used by human beings, which distinguishes it from other living beings, allowing the exchange of ideas, expression of opinions or revelation of thought. In face technology advance and consequently appearance of electronics equipments more and more sophisticated, the possibility to allow the man-machine interaction through speech have been object of interesting as to academic environment as to electronic equipment developers. Research in the area of Speech Processing has been allowing the development of Speech Synthesis Systems, Speech Recognition Systems and Speaker Recognition Systems. However, processing requirements still difficult the implementation of those systems in devices with low computational power, as mobile phone, palmtops and home equipments. This work consists of the study and adaptation of digital processing speech signals techniques, resulting in an optimized library of preprocessing including preemphasis, division into frames and windowing, allowing this use in development of speech or speaker recognition embedded applications. It was made adaptations in the models, implementation in a *hardware* description language, library functional verification and finally the prototyping in a *hardware* device.

ÍNDICE

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

A/D	Analógico/Digital
ALU	A*ritmetic Logic Unit*
API	*Application Programming Interface*
CLB	*Configurable Logic Block*
CPLD	*Complex Programmable Logic Device*
D/A	Digital/Analógico
DSP	*Digital Signal Processor*
DUV	*Design Under Verification*
DVD	*Digital Versatile Disc*
FIFO	*First In First Out*
FPGA	*Field-Programmable Gate Array*
GPP	*General Purpose Processor*
HDL	*Hardware Description Language*
HLA	*High Level Architecture*
HMM	*Hidden Markov Model*
IOB	*Input Output Block*
IP C*ore*	*Intellectual Property Core*
LPC	*Linear Prediction Coding*
LUT	*Lookup Table*
NoC	*Network-on-chip*
OCP-IP	*Open Core Protocol – International Partnership*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
PDSV	Processamento Digital de Sinais de Voz
RAM	*Random Access Memory*
ROM	*Read Only Memory*
RTL	*Register Transfer Level*
RTOS	*Real Time Operation System*
SoC	*System-On-a-Chip*
SRAM	*Static Random Access Memory*
TCL	*Tool Command Language*

VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

Capítulo 1

Introdução

1.1 Comunicação Vocal Homem-Máquina

A fala é o meio de comunicação utilizado por excelência pelo homem e que o distingue dos demais seres vivos, permitindo-lhe a troca de idéias, expressão de opiniões ou revelação de seu pensamento. A possibilidade de realizar a comunicação homem-máquina através da voz torna essa interação mais fácil e produtiva [Rabiner 1978, Doddington 1985, Vieira 1989, Fagundes 1993, Minker 2004], uma vez que além de ser a forma mais natural de comunicação humana, permite que as mãos e os olhos dos usuários fiquem disponíveis para outras tarefas, o que oferece inúmeras vantagens, como por exemplo:

- **Velocidade:** a maioria das pessoas pode falar facilmente 200 palavras por minuto, mas poucas digitam em um teclado mais de 60 palavras por minuto [Costa 1994].
- **Não requer treinamento:** o uso da fala não exige que a pessoa tenha determinadas habilidades [Martins 1997] e não pode ser esquecida.
- **Mobilidade:** o usuário pode acessar o sistema mesmo que esteja se movimentando ou executando outra operação com as mãos ou com os olhos.
- **Acesso Remoto:** é possível a utilização da rede telefônica para transmissão de informação [Martins 1997].

Os primeiros trabalhos sobre máquinas que conseguiam reconhecer algumas palavras datam de 1952 [Davis 1952]. Graças às descobertas de algumas propriedades da voz e das facilidades oferecidas pelos computadores digitais, nos anos 60 surgiu uma grande quantidade de trabalhos sobre o assunto [Koenig 1946, Rabiner 1978]. Posteriormente, desejou-se a criação de máquinas que, além de serem capazes de reconhecer o que estava sendo dito, também pudessem responder ao que lhes era perguntado.

Diante do crescente interesse por aplicações de *softwares* e equipamentos que "compreendam", reconheçam e simulem a voz humana, pesquisas têm sido realizadas na área de Processamento Digital de Sinais de Voz (PDSV), buscando o desenvolvimento de técnicas que possibilitem a construção de padrões, a partir da fala que, permitam modelar de forma eficiente as características vocais únicas de cada locutor, bem como a produção da voz sintetizada [Kleijn 1998, De Lima 2000, Benzeghiba 2003, Da Cunha 2003, Lemmetty 2004].

O avanço tecnológico na área de PDSV permite o desenvolvimento de sistemas de comunicação vocal homem-máquina, que podem ser divididos em três grandes áreas [Rabiner 1978, Campbell 1997, Cipriano 2001, Ibnkahla 2005]:

1. Sistemas de Resposta vocal;
2. Sistemas de Reconhecimento de voz;
3. Sistemas de Reconhecimento de locutor.

Sistemas de resposta vocal, ou sistemas de síntese de voz, são projetados para responder a um pedido de informação utilizando mensagens faladas [Lemmetty 2004]. A maneira mais simples de produzir voz sintetizada consiste em reproduzir amostras pré-gravadas da voz natural, tais como sentenças ou palavras simples. Esta concatenação gera alta qualidade e naturalidade, mas tem um vocabulário limitado, uma vez que não é possível criar uma base de dados com todas as palavras e nomes comuns no mundo. Além disso, processamento adicional pode ser necessário para melhorar a continuidade entre palavras e a entonação das frases [Deller 1993]. Para uma síntese de voz irrestrita, devem ser utilizados trechos mais curtos do sinal de voz, tais como sílabas, fonemas ou, até mesmo, segmentos mais curtos [Kleijn 1998, Lemmetty 2004]. Sistemas de resposta vocal podem ser utilizados em várias aplicações, tais como sistemas automáticos de informação de preços, de vôos, de divulgação de produtos em supermercados ou lojas especializadas, sistemas bancários, auxílio a portadores de necessidades especiais, etc. [Kleijn 1998]. Esse tipo de sistema também pode ser utilizado no acesso a menus de equipamentos eletrônicos, eletrodomésticos e dispositivos móveis como, por exemplo, celulares, PDA e *palmtops.* No desenvolvimento de aplicações

para esse tipo de equipamento, características como consumo de energia, capacidade de processamento e armazenamento precisam ser considerados [Ibnkahla 2005]. Apesar de progressos recentes, alguns aspectos ainda precisam ser melhorados no que se refere à síntese de voz, tais como naturalidade, entonação correta de frases, combinações de palavras e pronúncia de outras mais complexas [Paula 2000, Tatham 2005].

Sistemas de reconhecimento de voz têm como objetivo determinar qual a palavra, a frase ou a sentença que foi pronunciada, e podem ser subdivididos em um grande número de subáreas, dependendo de alguns fatores, tais como tamanho do vocabulário e população de locutores [Rabiner 1978, Benzeghiba 2003]. De uma forma geral, são considerados como pertencentes a uma das seguintes categorias: Sistemas de Reconhecimento de Palavras Isoladas; Sistemas de Reconhecimento de Palavras Conectadas; Sistemas de Reconhecimento de Fala Dependente do Locutor e Sistemas de Reconhecimento de Fala Independente do Locutor [Rabiner 1978]. A diferença entre os sistemas dependentes e os independentes de locutor é que no primeiro, o sistema é treinado para um locutor específico, enquanto que o segundo está preparado para reconhecer sentenças, independente de quem as pronuncie. Sistemas de reconhecimento de fala podem ser utilizados em diversas aplicações, dentre as quais se pode citar: atendimento telefônico automatizado, acesso a menus de equipamentos eletrônicos (celular, fornos de microondas, aparelhos de DVD, etc.), transcrição de sentenças pronunciadas para um arquivo texto, etc. Os sistemas para fala contínua (palavras conectadas) são mais difíceis de implementar do que os sistemas para fala descontínua (com pronúncia isolada das palavras), devido, por exemplo, à dificuldade de se localizar o início e o fim de cada palavra, e também à tendência das línguas faladas de fundir o último e o primeiro fonema de duas palavras consecutivas [Rabiner 1978, Dias 2000, Lin 2003].

Quanto aos sistemas de reconhecimento de locutor, estes podem ser de dois tipos: verificação ou identificação. O primeiro consiste em decidir se a voz é mesmo de quem alega ser, enquanto que o segundo deve ser capaz de identificar quem é o locutor, dentre um conjunto de locutores. Esses sistemas são muito úteis para realizar controle de acesso a ambientes restritos e na área de criminalística podem ser utilizados com o mesmo propósito que hoje é dado às impressões digitais [Rabiner 1978, Campbell 1997]. A autenticação vocal se baseia no fato de que a voz é única para cada pessoa e pode ser utilizada para identificar alguém [Fechine 2000]. A grande dificuldade encontrada nesse tipo de sistema é a existência de diversos fatores que podem contribuir para erros no processo de reconhecimento do locutor. Como exemplo desses fatores, pode-se citar: estado emocional ou de saúde do

locutor, ruído no ambiente, variação na posição do microfone, dentre outros [Rabiner 1978, Campbell 1997, Dias 2000, Fechine 2000].

É importante destacar que, quando se fala em interface homem-máquina, deve-se ficar claro que a máquina pode ser qualquer equipamento eletrônico e, no âmbito deste trabalho, pode ser um celular, um *palmtop*, uma máquina de lavar, um forno de microondas, um veículo, uma máquina industrial, dentre outros equipamentos que apresentam arquitetura e limitações (capacidade de processamento e armazenamento, etc) diferentes de um computador. Diante disso, torna-se necessário o desenvolvimento de *software* adaptado para este ambiente. Surge então, o conceito de sistema embutido (embarcado), que é a combinação de *hardware* e *software* e, algumas vezes, peças mecânicas, desenvolvidos para realizar uma função específica [Barr 1999, Francia 2001]. Através da implementação em *hardware,* é possível alcançar maior eficiência e rapidez na execução de determinadas tarefas e, a partir do *software,* reduzir o tempo de desenvolvimento do sistema.

1.2 Motivação

Com o advento da tecnologia, as máquinas predominam em quase todos os cenários do cotidiano das pessoas, sejam essas máquinas computadores, eletrodomésticos, dispositivos portáteis, etc. Com isso, nada melhor do que dotá-las com a capacidade de percepção e compreensão da voz humana, que é a forma mais simples, natural e eficaz do ser humano expressar seus pensamentos [Rabiner 1978, Doddington 1985, Vieira 1989, Fagundes 1993, Minker 2004]. Tendo em vista o vigente interesse pelo desenvolvimento de aplicações em PDSV para sistemas embutidos (*embedded systems*) e também as dificuldades enfrentadas neste tipo de projeto, torna-se clara a importância da definição e implantação de ferramentas de *hardware* e *software* que otimizem a produção dessas aplicações.

O interesse pela interação homem-máquina através da fala tem aumentado consideravelmente, dando origem a uma demanda muito grande por sistemas capazes de reconhecer o que está sendo dito, bem como quem está falando ou responder ao que está sendo solicitado [Rabiner 1978, Doddington 1985, Vieira 1989, Fagundes 1993, Vassali 2000, Minker 2004]. Devido à existência de várias questões que ainda não foram solucionadas, muito estudo ainda vem sendo realizado na área de Processamento Digital de Sinais de Voz [Lin 2003, Krishna 2004, Ibnkahla 2005, Nedevschi 2005]. Com a disponibilidade das linhas telefônicas e microfones acoplados aos computadores, o custo de uma aplicação na área da

comunicação vocal homem-máquina está relacionado, basicamente, ao projeto do *software*. Algoritmos com bom desempenho, em termos de taxa de reconhecimento, demandam um certo poder computacional, o que já se tem disponível em máquinas tipo PC (*Personal Computer*). No entanto, em sistemas embutidos como um telefone celular, o desafio não representa apenas o desenvolvimento de *software*, mas também de *hardware* que se adapte às características inerentes de tais dispositivos (baixo consumo de energia, baixo poder de processamento e armazenamento, dimensões reduzidas, portabilidade, etc.) [Lin 2003, Krishna 2003, Krishna 2004, Ibnkahla 2005, Nedevschi 2005].

Diante da pressão mercadológica em um mercado mundial globalizado, somada à contínua evolução tecnológica, as empresas precisam projetar novos sistemas embutidos dentro de janelas de tempo cada vez mais estreitas, de poucos meses. Além disso, novos produtos têm uma vida útil cada vez mais curta, de modo que o retorno financeiro de seu projeto deve ser obtido também em poucos meses. Sem deixar de considerar que atrasos de poucas semanas no lançamento de um produto podem comprometer seriamente os ganhos esperados [Carro 2003].

Com a automação do projeto de *hardware* levada a efeito pelo reuso de plataformas e componentes, a automação do projeto do *software* se torna o principal objetivo a ser alcançado para a diminuição do tempo de projeto. Também aqui é essencial o reuso de componentes de *software* adaptados às limitações de recursos desse ambiente, de modo que o projeto do sistema embutido concentre-se apenas na configuração e integração dos componentes de *hardware* e *software* previamente desenvolvidos [Shandle 2002, Carro 2003].

1.3 Objetivos do Trabalho

Em uma aplicação de comunicação vocal homem-máquina, o sinal de voz emitido pelo usuário precisa ser tratado para que então possa ser realizado o processo de reconhecimento. A etapa responsável por esse tratamento denomina-se pré-processamento, e tem o objetivo de reduzir efeitos indesejados incorporados ao sinal de voz pelo ambiente de gravação ou canal de comunicação, além de prepará-lo para as etapas seguintes do processo de reconhecimento [Rabiner 1978, Furui 1981, Shaughnessy 2000].

O objetivo principal deste trabalho consiste no desenvolvimento de uma biblioteca de funções para pré-processamento digital de sinais de voz (pré-ênfase, segmentação e

janelamento) em sistemas embutidos. A possibilidade de ter essas funções já implementadas em uma biblioteca otimizará a etapa de projeto e implantação de aplicações de processamento digital de sinais de voz, reduzindo consideravelmente o custo final do produto.

Para atingir tal objetivo, tornou-se necessário estudar técnicas que permitem a otimização de rotinas de pré-processamento de sinais de voz para implantação em dispositivos com poucos recursos computacionais (capacidade de processamento, armazenamento, etc.). Alguns estudos comprovam que existem estruturas de *hardware* dentro dos processadores que consomem muitos recursos, mas que são muitas vezes subutilizadas [Carro 2003]. A identificação de técnicas que permitam a execução das tarefas de pré-processamento digital de sinais de voz utilizando, de maneira eficiente, o mínimo de recursos possível, proporciona a otimização de tais tarefas e, conseqüentemente, a sua implantação em um sistema embutido.

Depois da implementação da biblioteca, foi realizada sua verificação funcional, com o objetivo de verificar todas as funcionalidades de projeto e assegurar que estas ocorram da maneira especificada. Esta tarefa chega a consumir cerca de 70% do tempo total de desenvolvimento [Silburt 1995, Beizer 1995, Bergeron 2003, Da Silva 2004, Fine 2004]. Diante disso, ferramentas que possibilitem a geração rápida e eficiente de um ambiente de simulação podem e devem ser utilizadas. Para cobrir essa etapa, neste trabalho foi utilizada a ferramenta VeriSC [Da Silva 2004] que realiza a geração automática de um ambiente de simulação (*testbench*) implementado na linguagem SystemC, oferecendo várias vantagens como, por exemplo, a inclusão de nível de transação, cobertura dirigida, checagem automática e construção de verificação funcional randômica.

Para facilitar a integração dos componentes de *hardware* entre si, como também com componentes de outras bibliotecas, foi desenvolvida uma interface para os componentes de *hardware* da biblioteca.

1.4 Organização do Trabalho

Este documento está organizado da seguinte forma: no Capítulo 2, são apresentados conceitos de Processamento Digital de Sinais de Voz, incluindo a descrição sobre o papel e as funções da etapa de pré-processamento, objeto de estudo deste trabalho. No Capítulo 3, são discutidos os principais aspectos do contexto de sistemas embutidos, incluindo suas particularidades, limitações e dificuldades de implementação, como também, são apresentados trabalhos que tratam de PDSV em sistemas embutidos. No Capítulo 4, é

realizada a descrição do sistema desenvolvido neste trabalho. No Capítulo 5, são apresentados e discutidos os resultados obtidos ao final do trabalho. Por fim, no Capítulo 6, são apresentadas as considerações finais e sugestões para trabalhos futuros.

Capítulo 2

Processamento Digital de Sinais de Voz

2.1 Apresentação

O desenvolvimento de aplicações na área da comunicação vocal envolve a utilização de técnicas de Processamento Digital de Sinais de Voz. Para a aplicação adequada dessas técnicas, é importante que se compreenda o mecanismo de produção da fala, bem como os fundamentos do Processamento Digital de Sinais de Voz.

Na Seção 2.2 deste capítulo, é descrito o mecanismo de produção da fala e, na Seção 2.3, a produção de um sinal digital de voz. Em seguida, na Seção 2.4, são apresentadas as principais características de uma aplicação de comunicação vocal homem-máquina. Finalmente, na Seção 2.5, são introduzidos os conceitos e técnicas de pré-processamento digital de sinais de voz, alvo/objeto deste trabalho.

2.2 Mecanismo de Produção da Fala

Os sinais de voz são compostos de uma seqüência de sons que servem como uma representação simbólica da mensagem produzida pelo locutor para o ouvinte [Rabiner 1978]. Para gerar o som desejado, o locutor exerce uma série de controles sobre o aparelho fonador (Figura 2.1), produzindo a configuração articulatória e a excitação apropriadas.

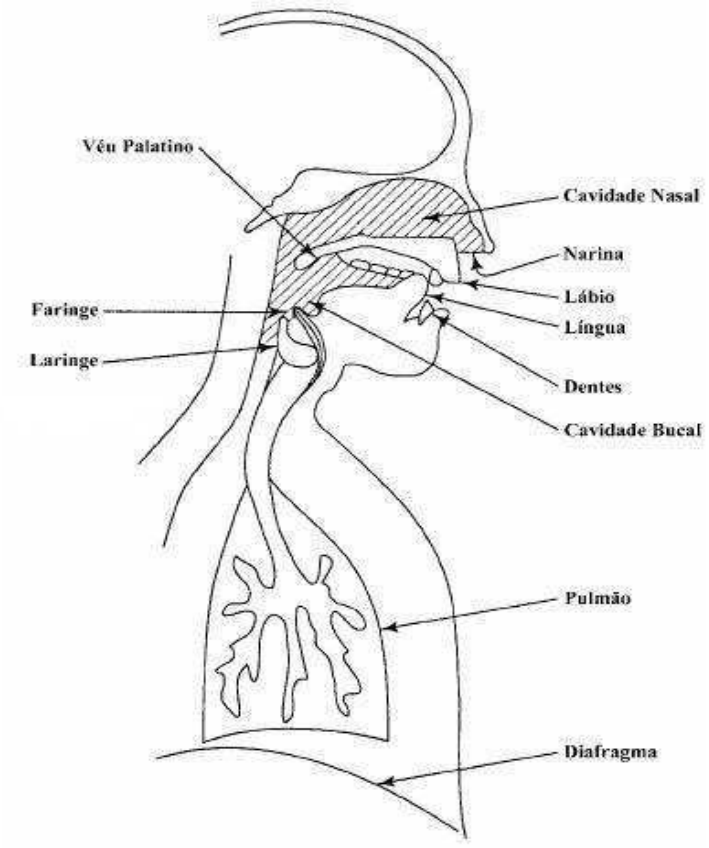


Figura 2.1 – Aparelho fonador humano (Adaptado de [Deller 1993]).

O trato vocal é o nome genérico dado ao conjunto de cavidades e estruturas que participam da produção sonora, tendo como limite inferior a região glótica e como limite superior os lábios. O trato nasal, responsável pela produção dos sons nasais, começa na úvula, ou véu platino, e termina nas narinas [Rabiner 1978].

Após a inalação do ar nos pulmões, os sinais de fala são produzidos durante a fase de exalação. Este fluxo de ar, depois da eventual vibração das cordas vocais, situadas na laringe, excitam o trato vocal, produzindo sons denominados sonoros. Para produção de sons nasalados, o véu palatino abre-se permitindo a passagem do ar pelo trato nasal e a sua radiação pelas narinas.

Na Figura 2.2, é apresentado um modelo mecânico para a produção de voz. Nesse modelo os tratos oral e nasal são representados por tubos acusticamente acoplados, de seção transversal não uniforme [Rabiner 1978, Shaughnessy 2000]. O som se propaga através desses tubos e o espectro de freqüência é modelado pela seletividade de freqüência do tubo. Esse efeito é muito similar aos efeitos de ressonância observados em instrumentos de sopro. No contexto da produção da voz, as freqüências de ressonância do tubo do trato vocal são chamadas de freqüências formantes ou simplesmente *formantes*. As freqüências formantes

dependem, sobretudo, da forma e dimensões do trato vocal. Cada forma é caracterizada por um conjunto de freqüências formantes. Sons diferentes são formados em função das variações da forma assumida pelo trato vocal. Assim, as propriedades espectrais do sinal de voz variam com o tempo e com a forma do trato vocal [Flanagan 1978, Shaughnessy 2000].

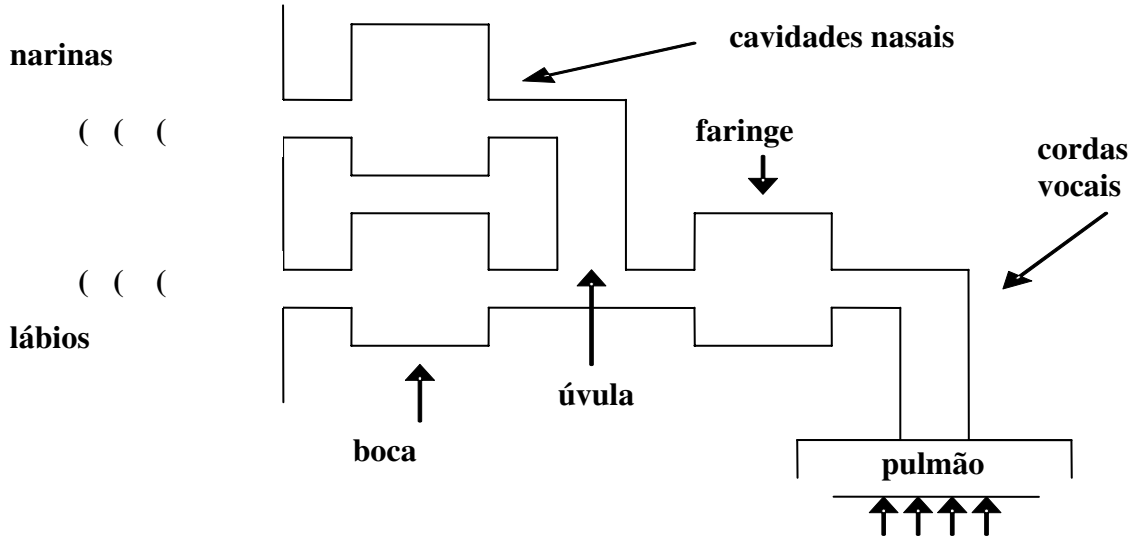


Figura 2.2 – Modelo acústico do aparelho fonador.

Em virtude das limitações dos órgãos humanos de produção de voz e do sistema auditivo, a faixa de comunicação humana típica tem como limite superior 7-8 kHz [Rabiner 1978, Shaughnessy 2000].

Para que seja obtido um modelo detalhado do processo de produção da voz, os seguintes efeitos devem ser considerados [Rabiner 1978]:

1. Variação da configuração do trato vocal com o tempo;
2. Perdas próprias por condução de calor e fricção nas paredes do trato vocal;
3. Maciez das paredes do trato vocal;
4. Radiação do som pelos lábios;
5. Junção nasal;
6. Excitação do som no trato vocal, etc.

Um modelo detalhado para geração de sinais de voz, que leva em conta os efeitos da propagação e da radiação conjuntamente pode, em princípio, ser obtido através de valores adequados para excitação e parâmetros do trato vocal. A teoria acústica sugere uma técnica simplificada para modelar sinais de voz, a qual é bastante utilizada [Rabiner 1978, Deller

1993, Shaughnessy 2000]. Essa técnica apresenta a excitação separada do trato vocal e da radiação. Os efeitos da radiação e o trato vocal são representados por um sistema linear variante com o tempo. O gerador de excitação gera um sinal similar a um trem de pulsos glotais, ou sinal aleatório (ruído). Os parâmetros da fonte e sistema são escolhidos de forma a se obter na saída o sinal de voz desejado [Rabiner 1978]. Considerando-se todos os fatores mencionados, é obtido o modelo apresentado na Figura 2.3 [Rabiner 1978], sendo $u(n)$ o sinal de excitação, e $A_s(n)$ e $A_f(n)$ o controle da intensidade da excitação do sinal de voz e do ruído, respectivamente.

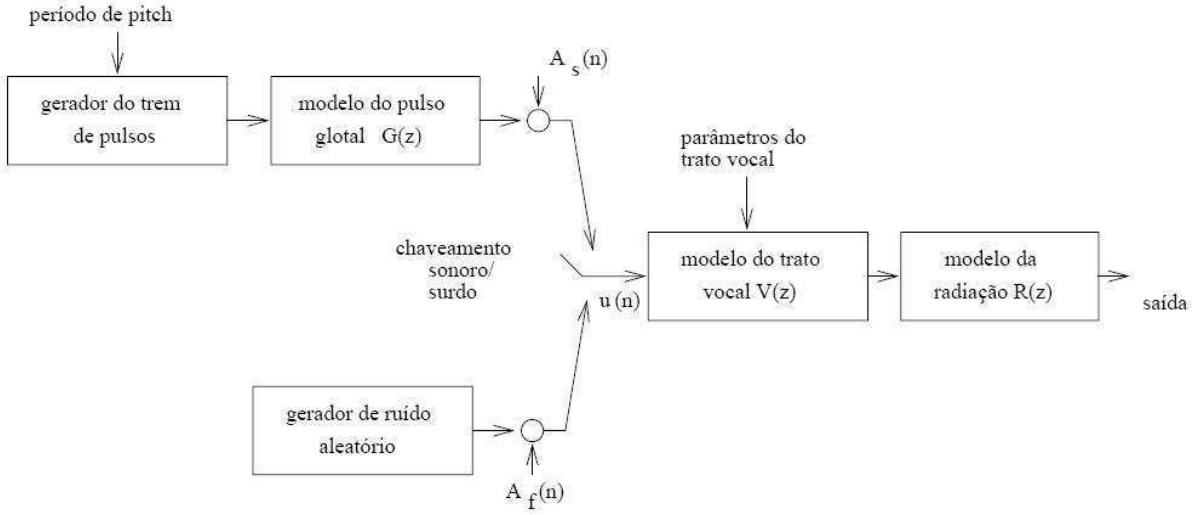


Figura 2.3 – Modelo discreto da produção da fala.

Chaveando entre geradores de excitação sonora e não sonora, alterna-se o modo de excitação. O trato vocal pode ser modelado em uma grande variedade de formas. Em alguns casos, é conveniente combinar o pulso glotal e modelos de radiação em um sistema simples. No caso da análise por predição linear, é conveniente fazer essa combinação juntamente com os componentes do trato vocal, representando-os através de uma simples função de transferência dada por [Rabiner 1978]:

$$H(z) = G(z)V(z)R(z) \tag{2.1}$$

Sendo:

G(z) – Transformada-z do modelo do pulso glotal;

V(z) – Transformada-z do modelo do trato vocal;

R(z) – Transformada-z do modelo da radiação.

2.3 Produção do Sinal Digital de Voz

Um sinal analógico varia continuamente no tempo assumindo um número infinito de valores de amplitude dentro de uma faixa finita de valores, denominada faixa dinâmica do sinal. Por outro lado, um sinal digital é caracterizado por assumir apenas valores distintos de amplitude e são representados por seqüências de números, possibilitando o uso de circuitos lógicos para o seu processamento [Aguiar 2000].

O sinal de voz produzido pelo homem é naturalmente analógico e para permitir seu processamento por computadores digitais é necessário que seja realizada uma conversão analógico-digital (A/D) sobre o mesmo. Essa conversão é apresentada na Figura 2.4.

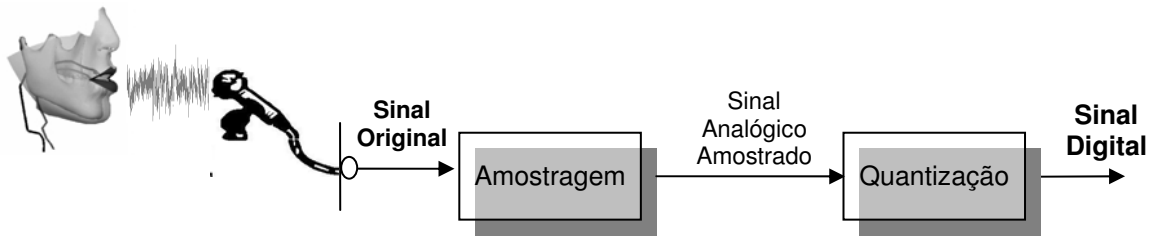


Figura 2.4 – Conversão analógico-digital do sinal de voz.

A aquisição dos sinais de voz é realizada por um microfone, o qual converte as variações que a fala causa na pressão do ar em variações de tensão elétrica. O próximo passo consiste na amostragem do sinal.

Nyquist [Lathi 1983] provou que um sinal contínuo pode ser completamente representado e recuperado através do conhecimento de suas amostras igualmente espaçadas no tempo. Para isso, basta que o intervalo (Ts) entre a captura das amostras seja:

$$Ts \leq 1 / (2.f_m) \qquad \therefore f_m \text{ é a maior freqüência do sinal analógico,} \tag{2.2}$$

ou seja, a taxa de amostragem (ou freqüência de amostragem) deve ser, no mínimo, igual ao dobro da máxima freqüência existente no sinal (Teorema de Nyquist). Na Figura 2.5, tem-se um exemplo de um processo de amostragem.

Tipicamente, um sinal de voz é limitado pela faixa de 3 a 5 kHz. Diante disso, considerando-se o Teorema de Nyquist, um valor ideal para amostragem de um sinal de voz típico deveria ser em torno de 10 KHz [Diniz 1997]. Neste trabalho foi utilizada uma

freqüência de 11025 Hz, uma das taxas de amostragem padrão disponíveis nas placas de som atuais [Bloch 2004].

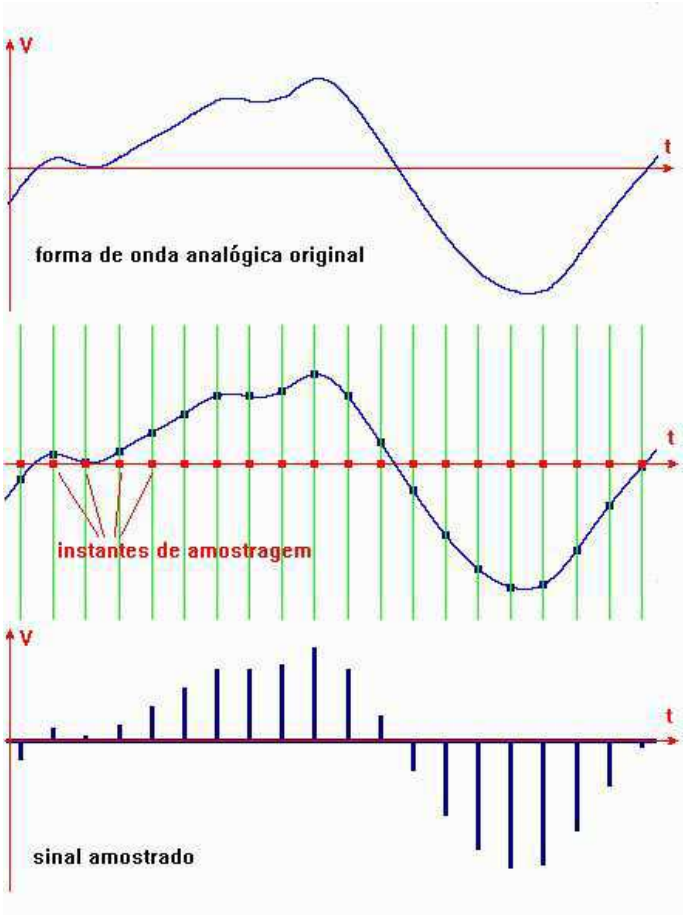


Figura 2.5 – Processo de Amostragem.

O passo seguinte à amostragem é a quantização (Figura 2.6) do sinal amostrado, que se refere à discretização da intensidade (amplitude) do sinal, permitindo a sua representação por uma quantidade finita e pré-definida de bits, obtendo-se então um sinal digital. Quanto maior a quantidade de bits utilizada nesse processo, maior o número de valores permitidos e menor o de aproximações necessárias, sendo assim o sinal quantizado mais fiel ao original.

A taxa de amostragem e a quantização podem variar de acordo com a aplicação. Quanto maior a taxa de amostragem e a quantidade de bits para a quantização, melhor será a qualidade do sinal obtido. Contudo, esse aumento é diretamente proporcional ao tamanho dos arquivos e, conseqüentemente, à demanda por maior banda de transmissão, de capacidade de processamento, de armazenamento, etc. Diante disso, é preciso realizar uma análise de necessidade de qualidade *versus* custo e encontrar valores adequados para cada situação.

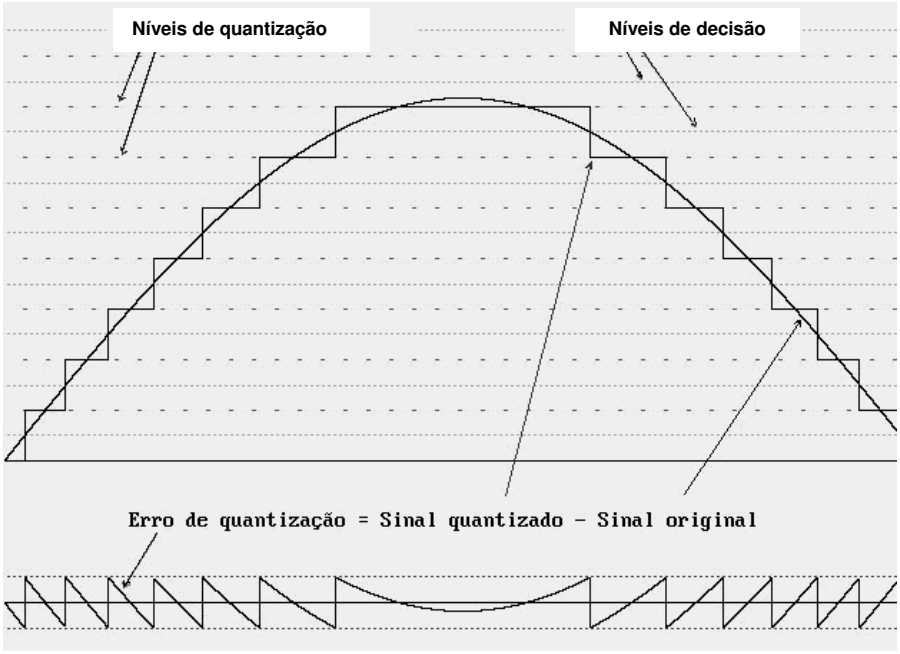


Figura 2.6 – Quantização de um sinal.

A priori, poder-se-ia tentar reconhecer a forma de onda representada logo após a conversão A/D, mas isto custaria um tempo de processamento indesejavelmente longo, uma vez que, a forma de onda contém variações redundantes e irrelevantes. Assim, sistemas de PDSV utilizam técnicas para reduzir a redundância e enfatizar as características mais importantes do sinal de voz, reduzindo consideravelmente o volume de dados a serem processados e, conseqüentemente, o tempo de processamento necessário [Costa 1994, Shaughnessy 2000].

2.4 Reconhecimento de Padrões da Fala

O desenvolvimento de aplicações na área de comunicação vocal homem-máquina (reconhecimento de fala ou locutor) consiste na realização de uma tarefa de reconhecimento de padrões, nesse caso padrões de fala. Esse tipo de sistema (Figura 2.7) inclui duas fases [Chou 2003]: treinamento e reconhecimento. Com base nos dados de treinamento, que consistem de sentenças (palavras ou frases), são gerados os modelos de referência, aos quais são atribuídos rótulos que identificam cada padrão (sentença ou locutor). Na fase de reconhecimento, através dos dados de teste (sinais de voz) são obtidos padrões de teste que, em seguida, são comparados com os modelos gerados durante o treinamento, e utilizando-se

uma regra de decisão, é identificado o que mais se assemelha ao padrão de entrada desconhecido [Rabiner 1978, Deller 1993, Shaughnessy 2000].

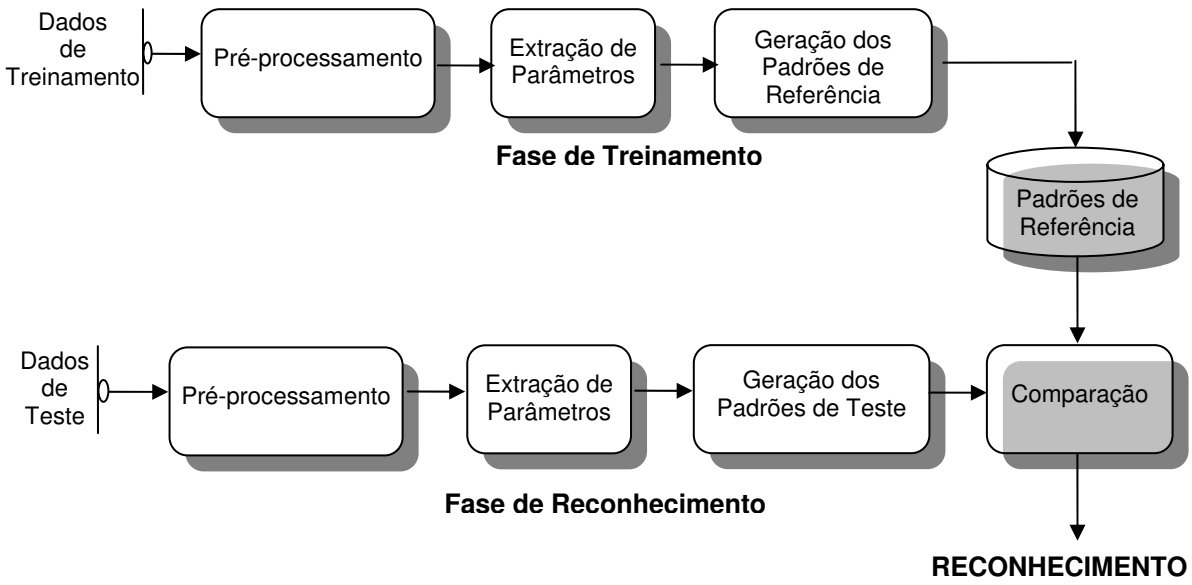


Figura 2.7 – Sistema de Reconhecimento de Padrões (Adaptada de [Rabiner 1978]).

A etapa de pré-processamento é responsável pelo tratamento do sinal de voz com relação ao ambiente de gravação e canal de comunicação utilizado. O objetivo desse tratamento é reduzir efeitos indesejados incorporados ou presentes no sinal de voz, além de prepará-lo para as etapas seguintes do processo de reconhecimento. Dentre vários aspectos que podem ser tratados nessa etapa, podem ser citados [Rabiner 1978, Furui 1981, Shaughnessy 2000]:

- Variações relacionadas ao estilo do falante;
- Ruído no ambiente, no meio de comunicação, etc;
- Variações no momento de gravação do sinal como, por exemplo, distância entre o locutor e o microfone;
- Considerações quanto ao tipo de unidade da fala a ser processada nas etapas posteriores.

As etapas típicas do pré-processamento são [Shaughnessy 2000]: normalização, filtragem, segmentação e janelamento. Neste trabalho, foram implementados o filtro de pré-

ênfase, segmentação e janelamento. Informações mais detalhadas sobre essas etapas são apresentadas na Seção 2.5.

A etapa de extração de características é de extrema importância em sistemas de reconhecimento, uma vez que nesta etapa são obtidos elementos que possibilitam a geração de padrões. Esses elementos também podem ser parâmetros obtidos a partir de modelos de produção de voz. Assim, essa etapa também poder ser chamada de *extração de parâmetros.*

A análise por predição linear, análise LPC (*Linear Prediction Coding*) é uma das técnicas mais importantes para análise de voz e tem sido a técnica predominante para estimar os parâmetros básicos da voz [Rabiner 1978, Dias 2000, Lee 2003, Zheng 2004, Atal 2006]. Esses parâmetros podem ser obtidos a partir da análise LPC, denominados coeficientes LPC, ou a partir de técnicas derivadas dessa análise. Dentre os coeficientes mais utilizados, destacam-se [Furui 1981, Mammone 1996, Lee 2003, Zheng 2004, Sanderson 2004]: coeficientes LPC, Cepestrais, Cepestrais Ponderados, Delta Cepestrais, Delta Cepestrais Ponderados e Mel Cepestrais, dentre outros.

Dependendo do grau de redução de dados, as características podem ser divididas em dois grupos [Shaughnessy 2000]:

- *Características fonéticas* – possuem uma faixa discreta, pois associam sons a categorias lingüísticas como, por exemplo, sons vocais ou sons fricativos. Essas características representam uma redução significativa da quantidade de dados, visto que transforma um sinal de voz em uma decisão fonética.
- *Características acústicas* – representam uma etapa intermediária entre parâmetros e características fonéticas. Como exemplo de características acústicas, podem ser citadas: as formantes e a freqüência fundamental.

A maioria dos sistemas de reconhecimento de fala utiliza apenas a parametrização, sem a geração do conjunto de características. Em geral, a quantidade de características é menor do que a quantidade parâmetros e, conseqüentemente podem se tornar mais eficientes para sistemas de reconhecimento de fala. Por outro lado, características são dependentes de um dado sinal de voz e, portanto, podem produzir classificações errôneas em um sistema de reconhecimento automático da fala independente do locutor [Shaughnessy 2000].

A definição de um bom conjunto de parâmetros e/ou características é um processo complexo e essencial para que se obtenha bons resultados quanto ao reconhecimento dos padrões.

Durante a fase de treinamento, com base nas características extraídas, são gerados padrões de referência, os quais serão comparados com o padrão de teste na fase de reconhecimento. De uma forma geral, os métodos utilizados para a construção dos padrões representativos da fala (ou do locutor) podem ser divididos em dois grupos: métodos paramétricos e métodos estatísticos [Huang 2001]. Exemplos de métodos paramétricos são: Alinhamento Dinâmico no Tempo, Quantização Vetorial e Redes Neurais [Haykin 1994, Madeiro 1999, Benzeghiba 2002]. Nos métodos estatísticos, a construção dos padrões é obtida por meio de modelos estatísticos, tais como Modelos de Markov Escondidos (HMM) [Rabiner 1989, Benzeghiba 2002]. É possível, ainda, a utilização de métodos híbridos, quando são combinados métodos paramétricos e estatísticos [Benzeghiba 2002, Fechine 2000].

Vale salientar que as etapas de pré-processamento e extração de características das fases de treinamento e reconhecimento devem ser equivalentes, para que seja possível a correta comparação entre o padrão testado e o(s) padrão(ões) já conhecido(s) pelo sistema.

A seção a seguir descreve o pré-processamento digital de sinais, visto que esta etapa é objeto de estudo deste trabalho.

2.5 Pré-processamento Digital de sinais de voz

Após a aquisição e digitalização do sinal de voz, é realizado o pré-processamento nas amostras, a fim de prepará-las para a extração de seus parâmetros e/ou características. Tais parâmetros e/ou características são utilizados no algoritmo de reconhecimento de padrões. O pré-processamento, realizado neste trabalho, inclui as etapas de pré-ênfase, segmentação e janelamento.

2.5.1 Pré-ênfase

Conforme discussão apresentada na Seção 2.2, a voz produzida pelo aparelho fonador humano sofre perdas durante sua passagem pelo trato vocal, inclusive na sua radiação através dos lábios. A distorção provocada pelos lábios produz uma queda na envoltória espectral de, aproximadamente, 6dB/oitava. Uma vez que o sinal de voz apresenta baixas amplitudes nas altas freqüências, essa tendência as torna especialmente vulneráveis ao ruído, comprometendo o processo de reconhecimento.

Para solucionar esse problema é aplicado um filtro, de resposta aproximadamente +6dB/oitava, que ocasiona um nivelamento no espectro [Petry 2000]. A esse processo de tratamento do sinal de voz, dá-se o nome de *pré-ênfase*.

A função de transferência da pré-ênfase consiste de um sistema de primeira ordem fixo, cuja função é dada por:

$$H(z) = 1 - a.z^{-1}, \quad 0 \leq a \leq 1 \tag{2.3}$$

Neste caso, a saída da pré-ênfase $s_p(n)$ está relacionada à entrada $s(n)$ pela equação diferença [Petry 2000]:

$$s_p(n) = s(n) - \alpha.s(n-1) \tag{2.4}$$

Sendo:

$s_p(n)$ – amostra pré-enfatizada;

$s(n)$ – amostra original;

α – fator de pré-ênfase, $0{,}9 \leq \alpha \leq 1$.

Um valor típico usado é $\alpha = 0.95$, o que significa 20 dB de amplificação para as mais altas freqüências [Da Cunha 2003]. Na Figura 2.8, é ilustrado o processo de pré-ênfase para a palavra "aplausos".

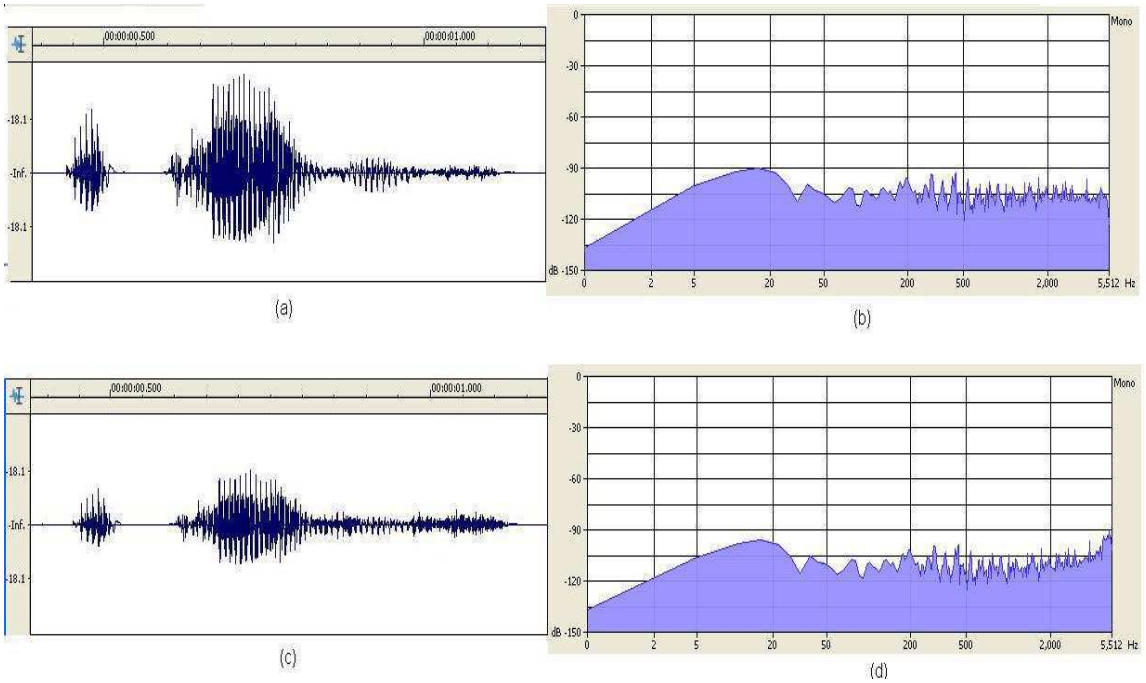


Figura 2.8 – Palavra "aplausos" – (a) Antes da Pré-ênfase e (b) seu espectro (c) após a Pré-Ênfase e (d) seu espectro.

Analisando-se a Figura 2.8, é possível observar que, após a pré-ênfase, as amostras de freqüências mais baixas sofrem uma atenuação, enquanto as de freqüências mais altas, especialmente acima de 4 kHz, têm um ganho de amplitude.

Quando a porção do sinal analisado corresponde a um som fricativo como, por exemplo, /s/ ou /f/, a pré-ênfase produz um sinal que se aproxima, geralmente, do som original com uma distorção muito pequena. Isto ocorre, devido ao fato desses sons apresentarem, basicamente, componentes de alta freqüência. Por outro lado, muitos fonemas não possuem uma concentração significativa de energia nas altas freqüências [Shaughnessy 1995]. A característica básica desses tipos de fonemas é a posição das três primeiras formantes, geralmente abaixo de 4 kHz. Considerando que a pré-ênfase reforça as amostras de alta freqüência, um aumento excessivo da energia acima de 6 kHz pode introduzir uma alteração nos parâmetros espectrais, o que pode comprometer a taxa de reconhecimento [Shaughnessy 1995].

2.5.2 Segmentação e Janelamento

Um sinal é dito estacionário quando suas características estatísticas não variam com o tempo [Lathi 1998]. A segmentação consiste em particionar o sinal de voz em segmentos, selecionados por janelas ou quadros (*frames*) de duração perfeitamente definida. O tamanho desses segmentos é escolhido dentro dos limites de estacionariedade do sinal (duração média de 16 a 32 *ms*) [Rabiner 1978, Shaughnessy 2000].

Os tipos de janelas normalmente utilizados são [Rabiner 1978, Deller 1993, Shaughnessy 2000]:

- Janela Retangular – o sinal é simplesmente particionado em blocos consecutivos de mesmo tamanho. Sua equação é dada por:

$$J(n) = \begin{cases} 1 & , 0 \le n \le N_A - 1 \\ 0 & , \text{caso contrário} \end{cases}$$

- Janela de *Hamming* – proporciona a manutenção das características espectrais do centro do quadro e a eliminação das transições abruptas das extremidades. Sua equação é dada por:

$$J(n) = \begin{cases} 0,54 - 0,46\cos[2\pi n/(N_A - 1)] & , 0 \le n \le N_A - 1 \\ 0 & , \text{caso contrário} \end{cases}$$

- Janela de *Hanning* – assemelha-se à janela de *Hamming*, porém gera um reforço menor nas amostras do centro e uma suavização maior nas amostras da extremidade. Sua equação é dada por:

$$J(n) = \begin{cases} 2a\cos[\pi n/N_A] + b & , 0 \le n \le N_A - 1 \\ 0 & , \text{caso contrário} \end{cases}$$

$$\text{Sendo } 2a + b = 1 \ (0 \le a \le 0{,}25 \ , \ 0{,}5 \le b \le 1)$$

Na Figura 2.9, são apresentadas as janelas Retangular, de *Hanning* e de *Hamming* e suas respectivas respostas em freqüência. Como se pode observar, o lobo principal da janela retangular é aproximadamente a metade do lobo principal das janelas de *Hamming* e *Hanning*, enquanto os lobos secundários dessas últimas são bem menores que os da retangular. O primeiro lobo secundário da janela de *Hanning* é aproximadamente 20 dB maior do que o da de *Hamming*, mas os lobos seguintes diminuem mais rapidamente do que os da janela de *Hamming*.

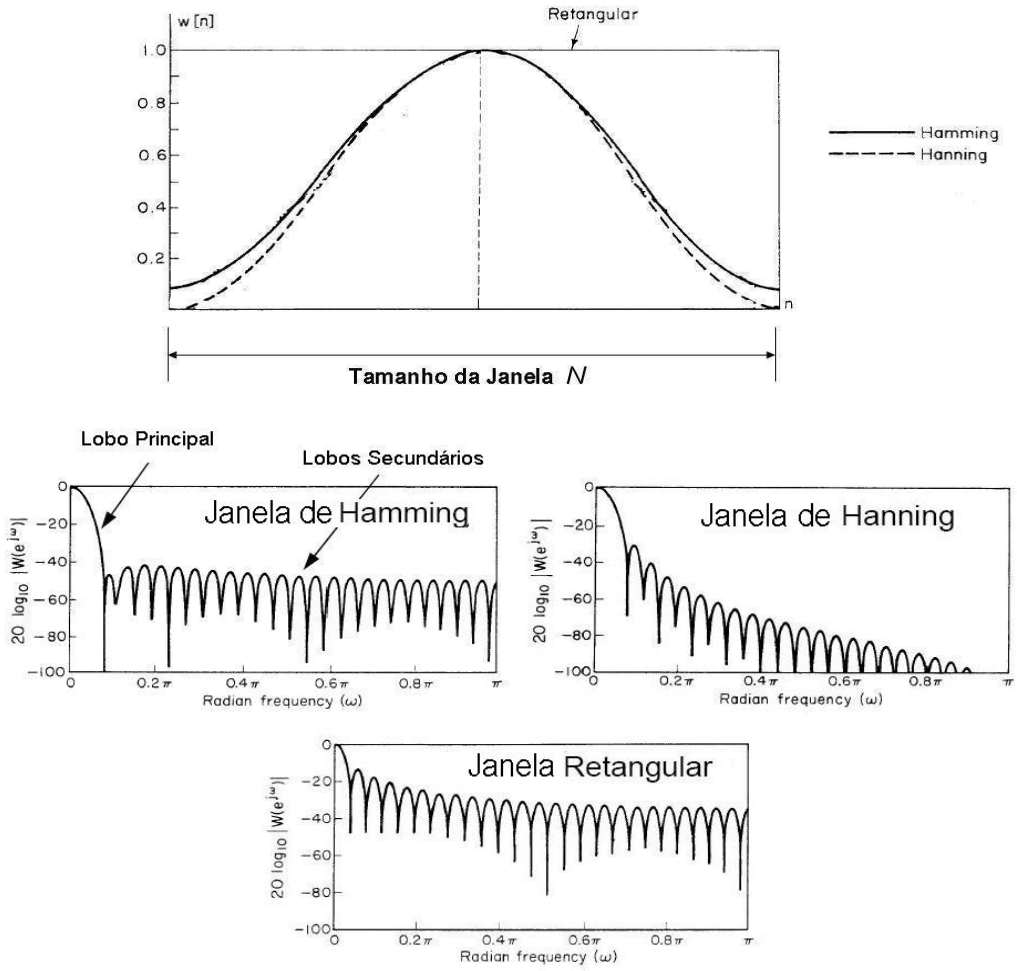


Figura 2.9 – Janelas de *Hamming*, *Hanning* e Retangular com suas respostas em freqüência [Lee 2005].

A escolha da janela a ser utilizada é uma questão de avaliação da relação custo-benefício entre um lobo principal estreito e lobos secundários pequenos. Um lobo principal

estreito melhora a resolução da freqüência, o que permite que componentes estreitos muito próximos sejam separados. No caso da janela retangular, no entanto, esse lobo principal estreito vem seguido de lobos secundários altos, o que adiciona uma aparência ruidosa ao sinal, devido à interferência de harmônicos adjacentes, dificultando a discriminação de componentes de baixa amplitude. Já nas janelas de *Hamming* e *Hanning* os lobos secundários menores permitem uma melhor detecção desses componentes.

A janela de *Hamming* apresenta, porém, uma característica nem sempre desejável, que corresponde à atribuição de um peso muito baixo às amostras da extremidade. Entretanto, estas amostras podem representar eventos importantes de curta duração do sinal de voz e multiplicá-las por um peso baixo representa pouca atenção no processamento subseqüente realizado no nível de blocos. Para assegurar que a tais eventos seja dado o peso necessário, blocos adjacentes são sobrepostos de modo que um evento seja "coberto" por outros blocos. Muitos trabalhos utilizam uma sobreposição de 50% [Dias 2000, Andreao 2001]. Outra justificativa para a sobreposição das janelas é que esta proporciona uma variação mais gradual dos parâmetros entre janelas sucessivas [Petry 2000, Picone 1993].

Para o contexto da produção da voz, as características apresentadas, referentes ao janelamento de *Hamming*, mostram que este tipo de janela é, portanto, mais eficiente, quando comparada às janelas Retangular e de *Hanning*, com uma boa aproximação da janela ideal. Assim sendo, essa foi a janela utilizada neste trabalho. Na Figura 2.10, é ilustrado o resultado das funções de pré-ênfase e janelamento na palavra "bola".

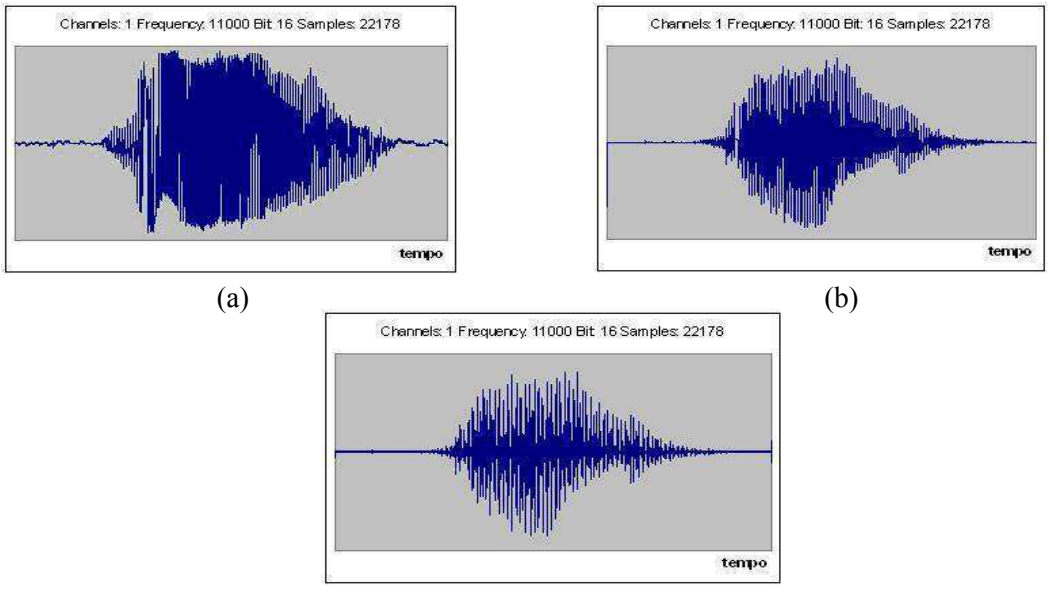


Figura 2.10 – Forma de onda da palavra /bola/. (a) Sinal Original (b) Sinal pré-enfatizado (c) Sinal Janelado.

2.6 Considerações Gerais

Neste capítulo, foi descrito o mecanismo de produção da voz do ser humano, bem como a sua representação a partir de um modelo mecânico e, em seguida, foi apresentada a modelagem matemática deste modelo. Em seguida, foi apresentado o processo de conversão do sinal de voz naturalmente analógico para um formato digital, de forma a permitir o seu processamento por computadores digitais.

Também foram descritas as etapas presentes em uma aplicação de comunicação vocal homem-máquina, detalhando-se as técnicas de pré-processamento digital de sinais de voz, alvo deste trabalho.

O conhecimento desses conceitos permite uma melhor compreensão da aplicação das técnicas empregadas, bem como suas especificidades e relevância.

Capítulo 3

Sistemas Embutidos

3.1 Apresentação

Atualmente, fabricantes de equipamentos eletrônicos têm procurado adicionar aos seus produtos, funcionalidades e diferenciais que atraiam o interesse de seus usuários. Como exemplo, pode-se citar televisores conectados à Internet; máquinas fotográficas que permitem "zoom" e alteram resolução, além de conexão a computadores e impressoras; fornos de microondas e máquinas de lavar roupas com controle "inteligente", etc. Tais equipamentos apresentam arquiteturas e limitações (capacidade de processamento e armazenamento, fonte de alimentação, etc) diferentes de um computador. Diante disso, torna-se necessário o desenvolvimento de *software* e/ou *hardware* específico para esse tipo de equipamento. Surge então, o conceito de sistema embutido (embarcado), que é a combinação de *hardware* e *software*, e algumas vezes peças mecânicas, desenvolvidos para realizar uma função específica [Barr 1999, Francia 2001]. Através da implementação em *hardware,* é possível alcançar maior eficiência e rapidez na execução de determinadas tarefas e, a partir do *software*, pode-se reduzir o tempo de desenvolvimento e aumentar a flexibilidade do sistema.

Na Seção 3.2 deste capítulo, serão apresentados os principais conceitos relacionados ao *hardware* de sistemas embutidos e, na Seção 3.3, serão descritas as etapas necessárias ao seu desenvolvimento. Também será apresentada a ferramenta *VeriSC* que foi utilizada neste trabalho para realizar a etapa de verificação funcional do sistema desenvolvido. Na Seção 3.4, são apresentados trabalhos que tratam de PDSV em sistemas embutidos.

3.2 Hardware de um sistema embutido

No que se refere à implementação em *hardware* de um sistema embutido, diferentes soluções podem ser adotadas. Para aplicações com produção em grande escala (mercado de consumo), o mais adequado é o uso de um SoC (*System-On-a-Chip* – sistema em uma única pastilha) [Carro 2003]. A arquitetura de *hardware* de um SoC embutido pode conter um ou mais processadores, memórias, interfaces para periféricos e blocos dedicados. Tais blocos, também chamados de IP Cores (*Intellectual Property Cores*), consistem de blocos em *hardware* que executam tarefas específicas e são definidos de modo a permitir o seu reuso em diferentes sistemas [Moraes 2004]. Os diversos IP Cores de um determinado SoC são interligados por uma estrutura de comunicação que pode variar de um barramento a uma rede complexa (NoC – *network-on-chip*) [De Micheli 2002].

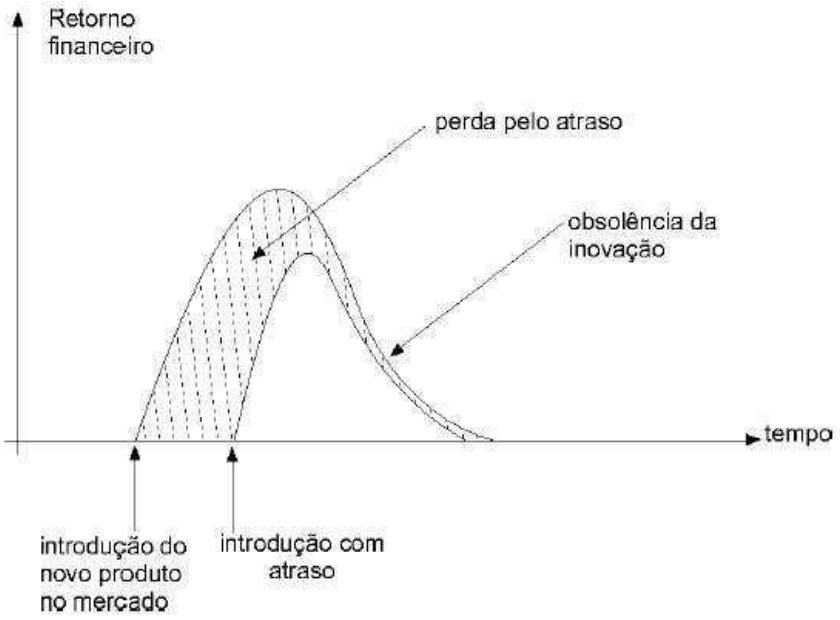


Figura 3.1 – Retorno financeiro e janelas de tempo [Carro 2003].

Além do longo tempo que pode ser gasto com uma exploração sistemática, deve-se considerar ainda o tempo necessário para o projeto e validação individual de todos os componentes dedicados do sistema – processadores, blocos de *hardware*, rotinas de *software*, RTOS (*Real Time Operation System*) – assim como o tempo de validação de sua agregação dentro de um mesmo sistema. Por outro lado, a grande pressão do mercado mundial, somada

à contínua evolução tecnológica, impõe às empresas a necessidade de projetar novos sistemas embarcados dentro de janelas de tempo cada vez mais estreitas, de poucos meses. Além disso, novos produtos têm uma vida cada vez mais curta, de modo que o retorno financeiro de seu projeto deve ser obtido também em poucos meses. Conforme mostrado na Figura 3.1, atrasos de poucas semanas no lançamento de um produto no mercado podem comprometer seriamente os ganhos esperados [Carro 2003].

Em aplicações com menor volume de produção, a implementação do sistema em FPGA *(Field-Programmable Gate Array*) é mais indicada. Apesar de ser viável também a implementação utilizando sistemas baseados em famílias de microprocessadores [Carro 2003], a implementação em FPGA apresenta como principal vantagem a possibilidade de modificação da estrutura de *hardware* do sistema através de um processo denominado reconfiguração, o qual permite o desenvolvimento incremental, correção de erros de projeto, além da adição de novas funções de *hardware* [Moraes 2004].

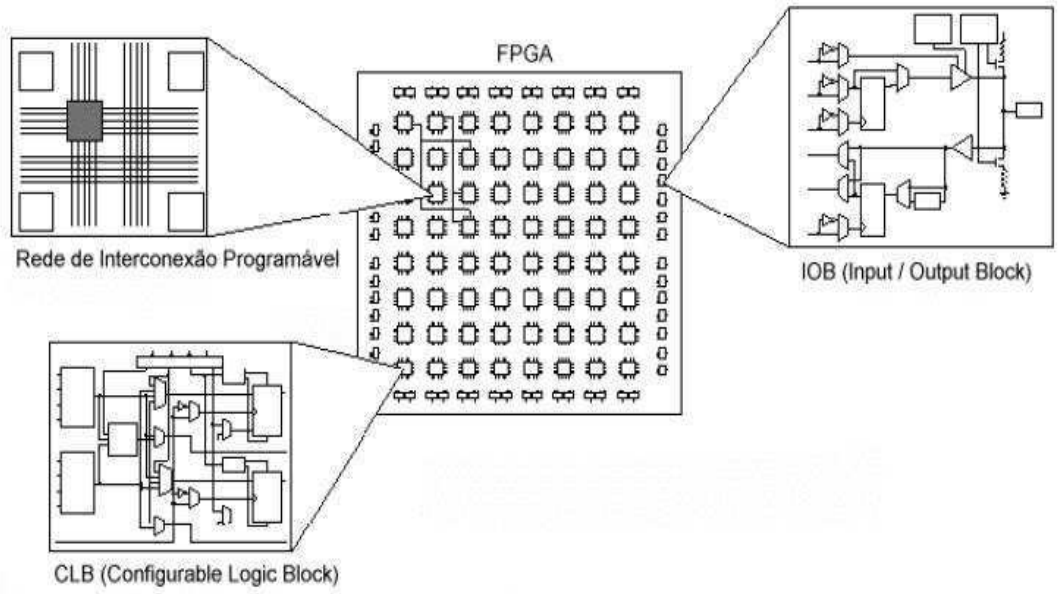


Figura 3.2 – Arquitetura interna de um FPGA [Martins 2003].

Um FPGA típico possui uma arquitetura interna (Figura 3.2) composta por uma matriz de blocos lógicos configuráveis (CLB - *Configurable Logic Block*) cercados por uma rede de interconexão programável, formada por blocos de interconexão [Martins 2003]. Circundando todo o circuito, existem os blocos de entrada e saída (IOB - *Input Output Block*), que também

são programáveis e que servem como interface entre o mundo exterior e a lógica interna do dispositivo.

A arquitetura de um CLB varia de família para família e de fabricante para fabricante, mas basicamente são compostos de pontos de entrada, que se conectam a blocos que implementam funções puramente combinacionais (LUT - *Lookup Table*), multiplexadores que direcionam o fluxo dos sinais internamente ao CLB; e de registradores (tipicamente *flip-flops*) que estão ligados às saídas e também podem realimentar as entradas dos geradores de funções combinacionais. Todos os elementos são configuráveis e propiciam grande flexibilidade para a implementação de funções. A rede de interconexão programável é composta por diferentes tipos de segmentos de conexão, capazes de interligar a maioria das entradas e saídas dos CLB entre si e aos IOB. Isso tudo permite que circuitos complexos, máquinas de estado e algoritmos sejam implementados nos FPGA [Martins 2003].

Além disso, existem os recursos de configuração ou reconfiguração do dispositivo, como conceitos e características arquiteturais, que podem apresentar as seguintes características:

- Capacidade de realizar uma única configuração ou muitas reconfigurações,
- Reconfiguração estática ou dinâmica, ou seja, pode ser realizada em tempo de compilação ou em tempo de execução,
- Reconfiguração parcial ou total,
- Modo de configuração único ou múltiplo,
- Quantidade de configurações armazenadas simultaneamente no dispositivo,
- Tempo gasto com a configuração ou a reconfiguração.

Na reconfiguração estática, o dispositivo é configurado antes de começar a fazer a computação (processamento) dos dados. Neste tipo de configuração, o dispositivo é primeiramente configurado e mantém o seu estado e funcionalidades enquanto se necessite destas características. Caso seja necessário reconfigurar para que se tenha outra funcionalidade, o dispositivo pára de executar as operações, é realizada uma nova reconfiguração para, só então, voltar à execução das operações de computação. Na reconfiguração dinâmica, o dispositivo é configurado e começa a processar os dados de entrada. Quando o dispositivo necessita de alguma nova funcionalidade, reconfigura-se uma área que não esteja processando nenhum dado e a reconfiguração ocorre paralelamente à

execução das respectivas operações de computação. Nem todos os dispositivos possuem reconfiguração dinâmica. Dessa forma, os dispositivos que não possuem essa característica, dependendo da aplicação, perdem em desempenho [Moraes 2001, Martins 2003, Moraes 2004].

Passada a etapa de configuração, os dispositivos reconfiguráveis estão prontos para serem usados. Normalmente, os sistemas reconfiguráveis híbridos usam um *hardware* reconfigurável junto a um processador de propósito geral (GPP – *General Purpose Processor*) que o auxilia nos processos [Martins 2003]. Em geral, o dispositivo reconfigurável é um FPGA. Este dispositivo serve para acelerar a execução de algoritmos, mapeando as partes que requerem uma demanda computacional grande para o substrato reconfigurável. O GPP serve para executar o algoritmo que não pode ser eficientemente acelerado pelo dispositivo reconfigurável. Assim sendo, os sistemas reconfiguráveis híbridos utilizam, nesses casos, os dispositivos reconfiguráveis como sendo um coprocessador para o GPP.

Cada elemento reconfigurável pode variar de uma simples *lookup table* de 3 bits até uma unidade lógica aritmética (ALU – A*ritmetic Logic Unit*) complexa de 4 ou mais bits. Esta diferença em termos de simplicidade/complexidade é chamada granularidade do dispositivo reconfigurável. Dispositivos mais simples (de grão fino) são indicados para aplicações de manipulação no nível dos bits, enquanto os mais complexos (dispositivos de grão grosso) são indicados para aplicações que envolvem computações mais complexas como manipulações de imagens, sinais de voz e outras aplicações típicas de caminho de dados, com manipulação de dados com largura (codificação) de vários bits [Moraes 2001].

Ao projetar o *hardware* de um sistema embutido, linguagens de descrição de *hardware* (HDL – *Hardware Description Language*) podem ser utilizadas para realizar a descrição dos circuitos eletrônicos. Essas linguagens permitem descrever a forma como os circuitos operam, possibilitando também a simulação desses circuitos antes mesmo de sua fabricação. Ao contrário de uma linguagem de programação para *software*, a sintaxe e a semântica das HDL incluem informações para expressar qual será o comportamento do *hardware* ao longo do tempo (*timed*). As linguagens mais populares são VHDL (VHSIC[1] *Hardware Description Language*) e Verilog, que apresentam a grande vantagem de possibilitar a sua utilização como entrada para simulação e síntese automática de circuitos descritos no nível da microarquitetura, através da utilização de ferramentas comerciais bastante difundidas [Carro 2003].

[1] Very High Speed Integrated Circuits

No entanto, essas linguagens não são apropriadas para descrições de *software* e especificações funcionais de alto nível [Carro 2003]. No intuito de sanar essa deficiência, foram realizadas tentativas com linguagens que possuíssem um maior grau de abstração, como C, C++ e Java. Nesse caso, ocorre exatamente o contrário, ou seja, a inadequação semântica dessas linguagens para descrição de aspectos de *hardware*. O uso da linguagem SystemC [SystemC 2006] visa solucionar essas deficiências, combinando as vantagens da popularidade de C++ e a sua adequação ao processo de geração de *software*, com uma semântica adicional (na forma de uma biblioteca de funções) apropriada para a descrição de *hardware*, através de construções como portas, sinais e relógios (*clocks*) que sincronizam processos [Carro 2003]. Detalhes sobre a linguagem SystemC são apresentados no Apêndice A.

3.3 Etapas para o desenvolvimento do *hardware* de um sistema embutido

Na Figura 3.3, é apresentado o fluxo de desenvolvimento do *hardware* de um sistema embutido. Nas seções seguintes, será apresentado o papel de cada etapa.

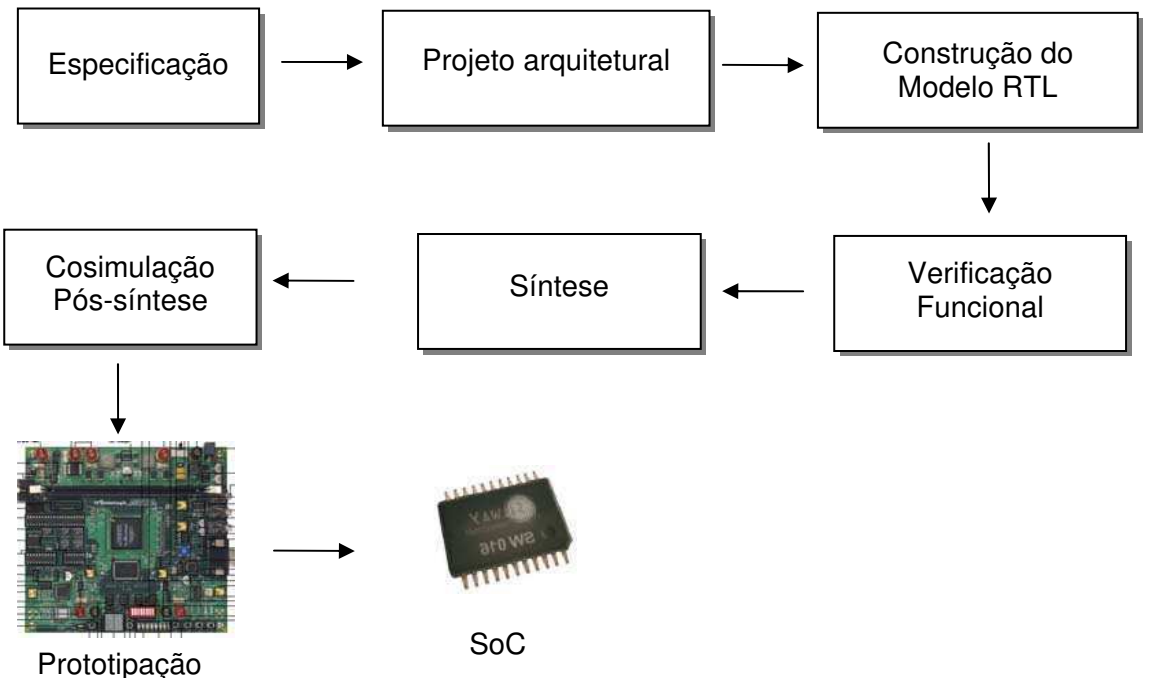


Figura 3.3 – Fluxo de desenvolvimento de um sistema embutido.

3.3.1 Especificação

O projeto de um sistema embutido é iniciado usualmente por uma especificação da funcionalidade desejada, feita através de uma linguagem ou formalismo adequado. Idealmente, esta especificação deve ter um alto nível de abstração, no qual ainda não tenham sido tomadas decisões em relação à implementação dessa funcionalidade, em termos da arquitetura-alvo a ser adotada, nem sobre os componentes de *hardware* ou *software* a serem selecionados. Esta especificação deve ser preferencialmente executável, para fins de validação [Carro 2003].

3.3.2. Projeto arquitetural

A seguir, é feita uma exploração do espaço de projeto arquitetural, de modo a se encontrar uma arquitetura que implemente as funções contidas na especificação inicial e que atenda aos requisitos de projeto, em termos de custo, desempenho, consumo de potência, área, etc. O resultado final desta etapa é uma macro-arquitetura (ou arquitetura abstrata), contendo um ou mais processadores de determinados tipos (DSP, microcontroladores) e outros componentes necessários (memórias, interfaces, blocos dedicados de *hardware*), todos interconectados através de uma infra-estrutura de comunicação (um ou mais barramentos ou uma NoC).

Entre a especificação funcional e a macro-arquitetura estabelece-se um mapeamento, através do qual cada função do sistema é atribuída a um processador ou a um bloco dedicado de *hardware*. Este mapeamento estabelece um determinado particionamento de funções entre *hardware* (blocos dedicados) e *software* (funções implementadas por um processador de instruções).

A exploração do espaço de projeto deve encontrar uma solução ótima para três questões básicas [Carro 2003]:

1) Quantos e quais são os processadores e blocos dedicados de *hardware* necessários?

2) Qual é o mapeamento ideal entre funções e componentes de *hardware*?

3) Qual é a estrutura de comunicação ideal para conectar os componentes entre si, tendo em vista as trocas de informações que devem ser realizadas entre as funções mapeadas para os componentes?

Para que esta exploração seja efetuada rapidamente, é fundamental a existência de estimadores que, a partir da especificação funcional do sistema, sejam capazes de informar, com um grau de precisão adequado, os valores de métricas importantes de projeto (desempenho, consumo de potência, área) que irão resultar de cada alternativa arquitetural (uma macro-arquitetura e um mapeamento de funções).

Tendo em vista um espaço quase infindável de soluções arquiteturais possíveis, com uma correspondente complexidade computacional para exploração do mesmo, em busca de uma solução ótima ou mesmo subótima, a etapa do projeto arquitetural é usualmente simplificada pela escolha prévia de uma plataforma arquitetural conhecida e adequada ao domínio da aplicação. Essa arquitetura pode conter um ou mais processadores de tipos conhecidos, além de outros componentes necessários, todos interconectados através de uma estrutura de comunicação também pré-definida.

Usualmente, diversas funções serão mapeadas para um mesmo processador, sendo, então, implementadas como tarefas concorrentes que precisarão ser escalonadas e gerenciadas por um sistema operacional e, caso a aplicação assim o requisite, este deverá ser um sistema operacional de tempo real (RTOS). Além da função de escalonamento de tarefas, o RTOS deve oferecer recursos para comunicação entre as tarefas, considerando que estas poderão estar distribuídas entre diversos processadores e mesmo blocos dedicados de *hardware*. Estes recursos devem oferecer uma abstração adequada ao *software* aplicativo, escondendo detalhes de mais baixo nível da infra-estrutura de comunicação. Também acionadores (*drivers*) dos periféricos devem ser oferecidos, ocultando, igualmente, detalhes das interfaces e da infra-estrutura de comunicação [Carro 2003].

Uma vez definida a macro-arquitetura, é necessária a geração do *software* para a mesma, a partir da especificação funcional do sistema. Idealmente, seria desejável uma síntese automática do *software*, incluindo tanto o *software* aplicativo como o RTOS. Esta geração do *software* é bastante facilitada se a especificação funcional inicial tiver sido feita sobre uma interface de programação da aplicação padronizada (API – *Application Programming Interface*) que ofereça recursos para comunicação entre as tarefas e para a qual exista uma implementação sobre a plataforma arquitetural (processadores e RTOS) selecionada. É também necessário um compilador, que traduza a especificação funcional para uma linguagem de programação adequada a cada processador adotado (a menos que a especificação funcional já tenha sido feita em uma determinada linguagem).

Componentes de *hardware* e *software* selecionados para a macro-arquitetura podem ter interfaces heterogêneas, implementando diferentes protocolos de comunicação. Neste caso, é necessária a síntese da comunicação entre os componentes. Esta síntese deve gerar adaptadores (*wrappers*) que fazem a conversão entre os diferentes protocolos. Adaptadores de *software* podem ser considerados como elementos de um RTOS dedicado gerado para a aplicação, enquanto que adaptadores de *hardware* são componentes dedicados que ajustam as interfaces dos componentes ao protocolo da infra-estrutura de comunicação (embora conexões ponto-a-ponto também sejam possíveis).

3.3.3 Construção do modelo RTL

O modelo RTL (*Register Transfer Level*) é a descrição da especificação do projeto em nível de fluxo de dados entre registradores, controlado por um *clock*. Esse modelo geralmente é escrito em uma linguagem de descrição de *hardware*, como VHDL ou Verilog.

3.3.3 Verificação Funcional

A melhoria contínua das metodologias de projeto e processos tem possibilitado a criação de grandes e complexos sistemas digitais. A verificação funcional é uma das principais tarefas do fluxo de desenvolvimento, e tem o objetivo de verificar todas as funcionalidades de projeto e assegurar que estas estão ocorrendo da maneira especificada. No contexto de *Intellectual Property* (IP) *cores design*, a verificação funcional é um elemento chave para garantia de sucesso em seu reuso [Bergeron 2003, Romero 2005].

Esta tarefa chega a consumir cerca de 70% do tempo total de desenvolvimento [Silburt 1995, Beizer 1995, Bergeron 2000, Bergeron 2002, Da Silva 2004, Fine 2004]. Diante disso, ferramentas que possibilitem a geração rápida e eficiente de um ambiente de simulação podem e devem ser utilizadas. Para cobrir esta etapa, foi utilizada neste trabalho a ferramenta *VeriSC* [Da Silva 2004] que realiza a geração automática de um ambiente de simulação (*testbench*) implementado na linguagem SystemC, a qual oferece várias vantagens como, por exemplo, a inclusão de nível de transação, cobertura dirigida, checagem automática e construção de verificação funcional randômica. A ferramenta VeriSC será descrita mais adiante.

Ao longo do processo de projeto, inúmeras descrições do sistema embutido são geradas, em diferentes níveis de abstração (especificação funcional, macro-arquitetura, micro-arquitetura), cobrindo diferentes aspectos (*software* e *hardware*) e eventualmente com a utilização combinada de diferentes linguagens (e.g. Java e VHDL). Estas descrições precisam ser validadas, o que exige a execução das descrições, no caso de linguagens de programação como Java e C, ou sua simulação, no caso de linguagens de descrição de *hardware* ou de sistemas como VHDL e SystemC [Carro 2003].

Conforme já discutido anteriormente, nenhuma dessas linguagens consegue cobrir simultaneamente todos os domínios de aplicação, níveis de abstração e aspectos de *hardware* e *software*. Assim, é bastante comum a necessidade de validação de descrições multi-linguagem em determinados passos do projeto. Um exemplo evidente é a validação combinada da microarquitetura descrita, por exemplo, em VHDL, e do *software* que irá rodar sobre um processador. Outro exemplo é a validação de um sistema embutido contendo partes de *hardware* digital, descritas em VHDL, *hardware* analógico, descrito por equações diferenciais em Matlab, e *software*, descrito em C.

Por outro lado, a validação de um sistema muito complexo pode ser bastante facilitada por um processo de refinamentos sucessivos, no qual apenas partes selecionadas do sistema são descritas num nível de abstração mais detalhado (e.g. micro-arquitetura), enquanto o restante do sistema permanece descrito de forma mais abstrata (p.ex. como macro-arquitetura ou mesmo como componente puramente funcional). Esta abordagem, que permite focalizar melhor as decisões de projeto que precisam ser validadas a cada novo passo de refinamento, também pode resultar em descrições multi-linguagem.

A simulação de sistemas que possuem módulos descritos em diferentes linguagens é chamada de cosimulação. Exemplos de cosimulação entre VHDL e C são encontrados nos simuladores VCI [Valderrama 1998] e SIMOO [Oyamada 2000]. Ferramentas de cosimulação atuais, como CoCentric System Studio, da Synopsys [Synopsys 2006], e Seamless CVE, da Mentor [Mentor 2003], permitem a integração de SystemC, C, simuladores de *software* no nível de instruções de máquina de um processador e HDL diversas, como VHDL e Verilog. MCI [Hessel 1999] é um exemplo de solução genérica, que permite a geração de modelos de cosimulação a partir de uma especificação multi-linguagem do sistema. No entanto, a cosimulação dá-se através de um mecanismo de comunicação proprietário, como nas soluções comerciais. O ambiente Ptolemy [Davis II 2001] adota uma abordagem orientada a objetos na modelagem do sistema e oferece um conjunto de classes explicitamente orientado à cosimulação de diferentes modelos de computação.

No contexto de reuso de componentes IP, torna-se interessante o desenvolvimento de simulações distribuídas, nas quais componentes IP sejam simulados remotamente, no *site* de seus fornecedores, visando a proteção da propriedade intelectual. O ambiente JavaCAD [Dalpasso 1999] oferece este recurso, mas está restrito a componentes descritos em Java, assim como o ambiente proposto em [Fin 2000] está restrito a componentes VHDL e Verilog. Soluções mais genéricas, abertas a diversas linguagens, são propostas pelos ambientes WESE [Dhananjai 2000], baseado em CORBA, e DCB [Mello 2002], inspirado no padrão HLA [HLA 2000] de simulação distribuída.

A validação por simulação apresenta duas grandes restrições. Em primeiro lugar, o número de casos de testes para uma validação exaustiva da descrição do sistema é muito grande, obrigando os projetistas na prática a limitarem-se a uma cobertura parcial dos mesmos. Em segundo lugar, quanto mais baixo o nível de abstração, maior o número de casos de teste e mais demorada é a simulação, pelo maior detalhamento da descrição. Técnicas de verificação formal [Edwards 1997], que realizam uma validação simbólica do sistema, e não numérica, prometem resolver este problema por cobrirem todos os possíveis casos de teste através de um único processamento. No entanto, tais técnicas ainda não atingiram um grau de maturidade suficiente que permita a sua aplicação em grande escala em todo o processo de projeto, estando limitadas a determinadas combinações de linguagens, estilos de descrição, domínios de aplicação e níveis de abstração.

VeriSC

A metodologia utilizada pelo *VeriSC* consiste de um ambiente de simulação composto de um *Source, um Driver, um Monitor, um Modelo de Referência e um Checker.* Os dados de entrada são enviados para o módulo que está sendo verificado, chamado de DUV (*Design Under Verification*), e para o Modelo de Referência. As saídas do DUV e do Modelo de Referência são coletadas e comparadas.

VeriSC é uma ferramenta que implementa essa metodologia de verificação e gera automaticamente um *template* do ambiente de simulação, de acordo com as características específicas do DUV. Todos os módulos do ambiente de simulação são gerados: *source, driver, monitor, modelo de referência, checker* e todas as FIFO que conectam esses módulos entre si, inclusive o DUV. Na Figura 3.4, é apresentada a estrutura do *testbench* gerado pelo VeriSC.

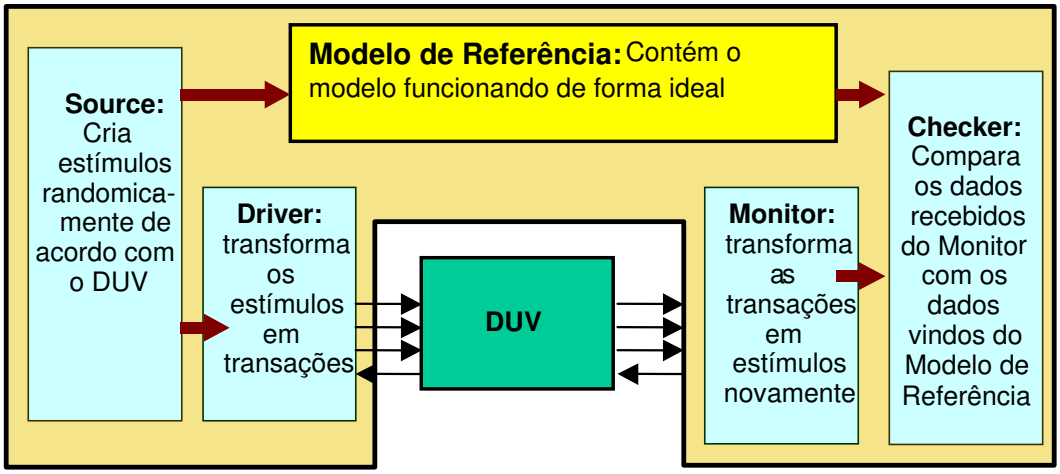


Figura 3.4 – Ambiente de simulação VeriSC [Da Silva 2004B].

O *template* é criado com base nas portas de saída e entrada do DUV e nas estruturas de nível de transação descritas em SystemC. As estruturas de nível de transação devem conter todas as informações sobre a semântica dos dados relevantes na comunicação com o DUV. O *signal handshake*, as métricas da cobertura funcional e a distribuição dos valores de entrada devem ser implementadas pelo engenheiro de verificação.

A função de cada um dos módulos [Da Silva 2004B]:

- ***Source***: responsável por gerar estímulos para o Modelo de Referência e para o *Driver*. Esses estímulos devem ser enviados através de FIFO.
- ***Driver***: cria as transações que serão submetidas como estímulos e as envia para o módulo DUV através de uma interface definida. O *Driver* é responsável também por gerar os atrasos para o modelo. O acesso que esse módulo possui ao DUV se dá somente pela interface definida. A transação flui do *Source* para o *Driver*, mas nunca na direção oposta.
- ***Monitor***: recebe as transações de saída do DUV, via interface, transforma-as em transações novamente e passa os resultados para o módulo *Checker* via FIFO. O acesso ao DUV se dá somente pela interface definida. Ele faz verificação de protocolo e é independente do *Driver* ou *Checker*.
- ***Checker***: é o responsável por verificar se as respostas resultantes do *Monitor* e Modelo de Referência são equivalentes. Se os resultados forem equivalentes, o módulo DUV implementado está de acordo com as especificações, caso

contrário são emitidas mensagens de erro. No último caso, é necessário modificar o DUV para que este fique de acordo com as especificações.

- **Modelo de Referência**: deve conter a implementação ideal do sistema. Por isso ao receber estímulos deve produzir respostas corretas. Esse modelo não deve possuir um controle de tempo, sendo por isso considerado *timeless*. Para efeito de comparação dos dados, após a conclusão das operações, o Modelo de Referência deve passar os dados de saída para o módulo *Checker* via FIFO.

As FIFO devem ser responsáveis por controlar os dados que entram e saem delas. Para que não haja erros na comparação dos dados vindos do *Monitor* e do Modelo de Referência, estes devem ser retirados um a um e comparados, de forma que sempre estarão na ordem correta de comparação.

Quando se implementa um DUV, o mesmo está inserido em um meio, que o permite receber dados de *n* entradas e enviá-los para *m* saídas, sendo *m* e *n* maiores ou iguais a 1. Para implementar essa característica é necessário que, para cada comunicação de entrada seja criada uma interface e, para cada comunicação de saída seja criada uma outra interface. Sendo que ao final tem-se *n* interfaces de entrada e *m* interfaces de saída. Diante disso, é gerado um *Driver* para cada interface de entrada e um *Monitor* para cada interface de saída.

VeriSC oferece um aumento na produtividade dos engenheiros de verificação, uma vez que reduz o tempo gasto na criação de ambientes de simulação.

3.3.4 Síntese

Uma vez definidos e validados os componentes de *hardware* da macro-arquitetura, incluindo a infra-estrutura de comunicação e os eventuais adaptadores, poderá ser feita a síntese do *hardware*. Numa primeira etapa, a macro-arquitetura pode ser expandida para uma micro-arquitetura (ou arquitetura RTL), contendo o detalhamento de todos os componentes e suas interconexões, pino-a-pino e considerando o funcionamento do circuito com precisão de ciclo de *clock*. Numa segunda etapa, podem ser usadas ferramentas convencionais para síntese de *hardware*, que a partir da micro-arquitetura irão gerar o *layout* final do circuito. Para tanto, é necessário que a microarquitetura esteja descrita numa linguagem apropriada para estas ferramentas, como VHDL ou Verilog. A existência prévia de *layouts* para os componentes de *hardware* selecionados facilita bastante esta síntese, que se limita então ao posicionamento e roteamento de células.

3.3.5 Cosimulação pós-síntese

Na etapa de síntese, é definido qual o dispositivo que será utilizado para prototipação do sistema, o que implica que, a partir de então, aspectos específicos do dispositivo, como por exemplo, atrasos das portas lógicas passam a ser considerados durante a execução. Isso torna necessária uma nova cosimulação pós-síntese de todo o sistema, de modo a assegurar a manutenção de seu correto funcionamento.

3.3.6 Prototipação

Vencida a etapa de cosimulação pós-síntese, pode-se passar para etapa final, a prototipação. Essa etapa consiste na implantação do código gerado pela síntese em algum dispositivo de *hardware.* Neste trabalho foi utilizada a placa Stratix II EPS260F672C5ES da Altera [Altera 2006] para prototipação do sistema. Detalhes sobre a placa Stratix II são apresentados no Apêndice B.

3.4 Trabalhos Relacionados

Este trabalho tem o objetivo de obter um módulo de pré-processamento de voz sintetisável, submetido a um processo de verificação funcional, validado em um dispositivo de *hardware* e que possa ser utilizado, de maneira transparente, em aplicações de reconhecimento. A possibilidade de ter essas funções já implementadas proporcionará uma economia de tempo na construção de aplicações de reconhecimento de fala ou locutor.

Algoritmos com bom desempenho na área de Processamento Digital de Sinais de Voz, em termos de taxa de reconhecimento, demandam um certo poder computacional, o que já se tem disponível em máquinas tipo PC em tempo real. No entanto, em sistemas embutidos dependentes de baterias, como um telefone celular, o desafio não representa apenas o desenvolvimento de *software*, mas também de *hardware* que se adapte às características inerentes de tais dispositivos (baixo consumo de energia, baixo poder de processamento, etc.) [Krishna 2004, Nedevschi 2005].

Algumas técnicas de programação permitem a geração de um código mais eficiente no que se refere à quantidade de operações a serem executadas pelo processador (p.ex.:

substituição de uma multiplicação por deslocamento de bits), o que conseqüentemente proporciona menor consumo de potência e energia [Vassali 2000]. Sendo assim, através da otimização de algoritmos, utilização de estruturas de *hardware* específicas e um *pipeline* de instruções eficiente, é possível aumentar a velocidade, reduzir os recursos necessários, sem diminuir a taxa de reconhecimento.

Muitos trabalhos vêm sendo desenvolvidos no contexto de processamento digital de sinais de voz para sistemas embutidos. Nesta seção será realizada uma breve descrição de alguns deles.

[Cipriano 2001] tem como objetivo a definição de uma arquitetura de reconhecimento de fala baseada em HMM que possa ser utilizada na comunicação e controle de eletrodomésticos, robôs, instrumentos de teste, etc. São propostas simplificações nos modelos matemáticos, que permitem, por exemplo, substituições de números em ponto-flutuante por números inteiros e paralelismo de operações. Nesse trabalho todos os módulos do sistema são definidos de modo que possam ser utilizados de forma independente, bastando para isso que se conheça os sinais de entrada e saída de cada módulo. Apresenta ainda potencial para ampliar o tamanho do vocabulário, das palavras e do número de locutores, permitindo agregar mais módulos integrados, comunicáveis entre si.

O objetivo do trabalho proposto por [Li 2003] é o desenvolvimento de um sistema de reconhecimento de fala rápido e com poucos recursos para aplicação em dispositivos móveis. Para isso, é proposta uma unidade aritmética baseada em *Lookup Tables* de rápido acesso. O autor defende que, o uso de aritmética em ponto-flutuante requerido pelos Modelos Ocultos de Markov (HMM - *Hidden Markov Models*) é responsável pelo elevado custo computacional e consumo de energia.

[Lin 2003] propõe o desenvolvimento de um *hardware* dedicado para reconhecimento de fala contínua, independente de locutor e com grande vocabulário. Esse trabalho ressalta que muitos sistemas de reconhecimento de voz embutidos sacrificam a acurácia do processo de reconhecimento em nome da redução do custo computacional. Destaca-se ainda que, uma modelagem inadequada dos padrões e o processo de busca são os maiores responsáveis pela complexidade desse tipo de aplicação e, defende que se for realizada uma segmentação eficiente dos fonemas essa complexidade pode ser reduzida. É proposta uma nova estrutura de memória e unidades aritméticas, além de processamento paralelo e redução da velocidade do *clock*, o que proporciona uma economia no consumo de energia.

[Krishna 2003] e [Krishna 2004] enfatizam que para permitir a implantação de sistemas de reconhecimento de fala em dispositivos móveis é necessário que sejam exploradas características específicas desse tipo de aplicação. No primeiro trabalho, com base no elevado nível de paralelismo que pode ser alcançado, é proposta uma arquitetura multiprocessada, o que permite uma redução no consumo de energia. No entanto, devido à grande quantidade de dados envolvida nesse tipo de aplicação, a memória se torna outro empecilho a ser superado. O segundo trabalho apresenta um estudo de diferentes arquiteturas de memória, com o propósito de definir a mais adequada para aplicações de reconhecimento de fala em dispositivos móveis e dependentes de bateria.

[Nedevschi 2005] tem como objetivo o desenvolvimento de uma arquitetura para reconhecimento em tempo-real de um pequeno vocabulário em dispositivos com pouca capacidade de processamento e baixo custo. O sistema é projetado de modo que seja possível trabalhar com diferentes dialetos, bastando para isso que o modelo referente ao respectivo dialeto, seja "carregado" e o sistema retreinado. A arquitetura desenvolvida faz uso de paralelismo, aritmética em ponto-fixo, além de memórias SRAM e FLASH embutida no chip.

Na Tabela 3.1, são apresentadas algumas características dos trabalhos citados, referentes aos aspectos destacados.

Tabela 3.1 – Resumo dos trabalhos citados.

	Objetivo	Verificação Funcional	Prototipação	Reuso
[Li 2003]	Reconhecimento de fala contínua	Não	Não	Não
[Lin 2003]	Reconhecimento de fala contínua	Não	Não	Não
[Cipriano 2001]	Reconhecimento de palavras isoladas	Não	Não	Sim
[Krishna 2003] e [Krishna 2004]	Reconhecimento de fala contínua	Não	Não	Não
[Nedevschi 2005]	Reconhecimento de palavras isoladas	Não	Sim	Não

O objetivo deste trabalho é o desenvolvimento de uma biblioteca de *hardware* para pré-processamento, que possa ser reutilizada em aplicações embutidas de reconhecimento de fala. Assim, todos os módulos foram desenvolvidos de forma que a sua utilização requer apenas o conhecimento das suas interfaces. Além disso, modificações em qualquer um dos módulos, desde que a interface seja mantida, não interfere no funcionamento dos demais. Também foram realizadas as etapas de verificação funcional e prototipação, no intuito de validar a biblioteca desenvolvida.

O exposto acima, mostra portanto, o diferencial deste trabalho em relação aos trabalhos relacionados apresentados nesta seção.

3.5 Considerações Gerais

O projeto de sistemas embutidos é bastante complexo, por envolver conceitos pouco analisados pela computação de propósito geral [Krishna 2004, Nedevschi 2005]. Por exemplo, as questões de mobilidade, limite de consumo de potência, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede de comunicação e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embutidos uma área de pesquisa bastante abrangente [Wolf 2001, Carro 2003]. Além disso, novos produtos têm uma vida cada vez mais curta, e atrasos de poucas semanas no lançamento de um produto podem comprometer seriamente os ganhos esperados.

Com a automação do projeto de *hardware* caminhando na direção do reuso de plataformas e de componentes integráveis em SoC (IP *Cores*), a automação do projeto de *software* e sua integração com o projeto de *hardware* se torna o principal objetivo a ser alcançado para a diminuição do tempo total de projeto, sem sacrifício na qualidade da solução [Carro 2003]. Também é essencial o reuso de componentes de *software*, de modo que o desenvolvimento do sistema concentre-se apenas na configuração e integração desses componentes [Shandle 2002, Carro 2003].

Um outro problema diz respeito aos custos de engenharia não-recorrentes. O projeto de um sistema embarcado de grande complexidade é bastante caro para uma empresa, envolvendo equipes multidisciplinares (*hardware* digital, *hardware* analógico, *software*, teste) e a utilização de ferramentas computacionais de custo elevado. São especialmente elevados os custos de fabricação de sistemas integrados numa pastilha, o que obriga as empresas a investir

apenas em projeto de componentes que tenham garantidamente volume muito alto de produção, de forma a amortizar os custos de fabricação.

Muitos trabalhos vêm sendo desenvolvidos para o reconhecimento de fala em sistemas embutidos, apresentando redução do número de acessos à memória, paralelismo de operações, economia de recursos lógicos necessários para implementação, etc. No entanto, a maioria dos trabalhos não realiza a etapa de verificação funcional dos módulos, mas apenas da taxa de reconhecimento, ou ainda não realizam testes efetivos em algum dispositivo de *hardware.*

Outro aspecto que deve ser considerado, é que a maioria dos trabalhos é implementada para uma aplicação específica, sem visar uma possível reutilização de seus módulos no desenvolvimento de outros sistemas de reconhecimento, diferentemente da proposta deste trabalho.

Capítulo 4

Descrição do Sistema

4.1 Apresentação

O protótipo desenvolvido neste trabalho, cujo diagrama é apresentado na Figura 4.1, tem como base o projeto descrito por Cipriano em [Cipriano 2001].

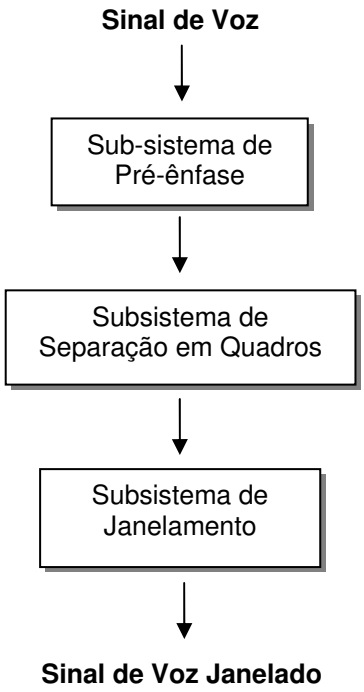


Figura 4.1 – Diagrama do Sistema Desenvolvido.

O projeto apresenta adequações dos modelos necessários ao processo de reconhecimento de voz para implementação em *hardware*. Para isso, é feita a utilização de ponto-fixo nas operações com número fracionários, simplificações nas funções matemáticas, substituição de operações de divisão e multiplicação por deslocamentos, paralelismo, dentre outras.

Na seção 4.2, é apresentado o subsistema de Pré-ênfase e, na seção 4.3, o subsistema de Divisão em quadros e janelamento. Na seção 4.4, é realizada uma breve descrição das principais ferramentas utilizadas no desenvolvimento deste trabalho.

4.2 Subsistema de Pré-ênfase

A função da pré-ênfase é eliminar uma tendência espectral de aproximadamente -6dB/oitava na fala irradiada dos lábios. Essa distorção espectral não traz informação adicional e pode ser eliminada através da aplicação de um filtro, de resposta aproximadamente +6dB/oitava, que proporciona um nivelamento no espectro. A descrição matemática da pré-ênfase é dada por [Petry 2000]:

$$s_p(n) = s(n) - ap.s(n-1) \qquad \textbf{(4.1)}$$

Sendo:

$s_p(n)$ – amostra pré-enfatizada;

$s(n)$ – amostra original;

ap – fator de pré-ênfase, $0{,}9 \leq ap \leq 1$.

No intuito de simplificar a implementação do filtro de pré-ênfase, na Equação (4.1), o fator *ap* é substituído pelo valor 15/16, correspondente a 0,9375:

$$s_p(n) = s(n) - \frac{15}{16}s(n-1) = s(n) - s(n-1) + \frac{s(n-1)}{16} \qquad \textbf{(4.2)}$$

A modificação efetuada na Equação (4.1) implica na substituição de uma multiplicação por um número fracionário, por uma divisão por um número do tipo 2^m, sendo m inteiro, e que pode ser realizada através de um simples deslocamento de m bits, nesse caso $m = 4$.

A arquitetura utilizada para realização da pré-ênfase (Figura 4.2) consiste de um somador, um subtrator, um acumulador, dois registradores de deslocamento e um circuito de divisão. A pré-ênfase é realizada em um único pulso de *clock*, sendo também necessários dois pulsos adicionais para a inicialização dos registradores.

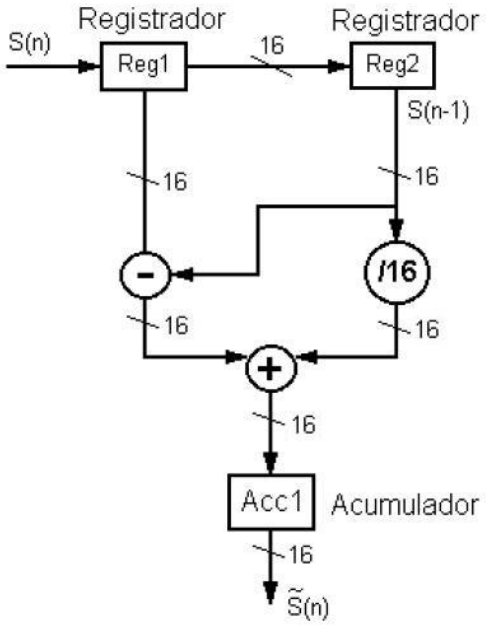


Figura 4.2 – Arquitetura do módulo da função de pré-ênfase.

Na etapa de inicialização, a primeira amostra do sinal de voz digitalizado, *s(1)*, é armazenada no registrador Reg1. Na etapa seguinte, é lido um novo dado, a amostra *s(2)*. Este novo dado é armazenado em Reg1 e o valor anterior de Reg1 é transferido para Reg2. As outras operações são realizadas em paralelo e o resultado final é armazenado no acumulador Acc1. Este procedimento é repetido até o processamento total das amostras do sinal de voz.

A divisão por 16 da Equação 4.2, é obtida fazendo-se um deslocamento de quatro bits para à direita, repetindo quatro vezes o bit mais significativo, à esquerda (Figura 4.3). Essa repetição é necessária para que seja mantido o sinal do valor correspondente da amostra de voz. Tal solução simples permite realizar a divisão em um único pulso de *clock*.

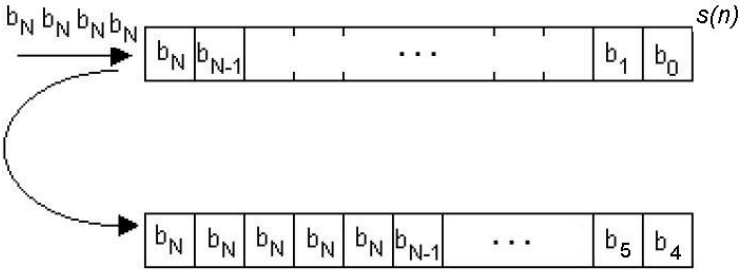


Figura 4.3 – Deslocamento à direita para realizar a divisão por 16 em uma palavra de N+1 bits.

4.3 Subsistema de Separação em Quadros e Janelamento

Um sinal de voz possui características estatísticas que variam fortemente com o tempo, podendo ser considerado estacionário para curtos intervalos de tempo. A separação em quadros consiste em particionar o sinal de voz em segmentos, selecionados por janelas ou "frames" de duração perfeitamente definida . O tamanho desses segmentos é escolhido dentro dos limites de estacionariedade do sinal (duração média de 16 a 32 ms) [Rabiner 1993, Shaughnessy 2000]. No sistema proposto o algoritmo de janelamento utilizado foi o de *Hamming*, que minimiza as descontinuidades no início e final de cada quadro, e cuja equação é dada por:

$$J(n) = \begin{cases} 0,54 - 0,46\cos[2\pi n/(N_A - 1)] & , 0 \le n \le N_A - 1 \\ 0 & , \text{caso contrário} \end{cases}$$

Sendo:
N_A – número de amostras de um segmento de análise.

O janelamento é realizado em conjunto com a separação dos quadros, que contêm 252 (N_A) amostras cada. Cada amostra do quadro é multiplicada pelo valor correspondente da janela, cujo tamanho também é igual a 252. A utilização de 252 amostras corresponde a um quadro com aproximadamente 23 *ms*, valor este que atende ao intervalo especificado por [Rabiner 1993], o que garante a estacionariedade da parte analisada do sinal. Além disso, o número 252 é divisível por 3 e por 9, o que facilita a implementação do algoritmo de janelamento com superposição utilizado (Figura 4.4), uma vez que, cada quadro é formado

por 3 blocos, sendo que cada bloco é escrito em 3 etapas, daí a facilidade oferecida por 252 ser divisível por 9. O algoritmo de escrita das amostras será explicado mais adiante.

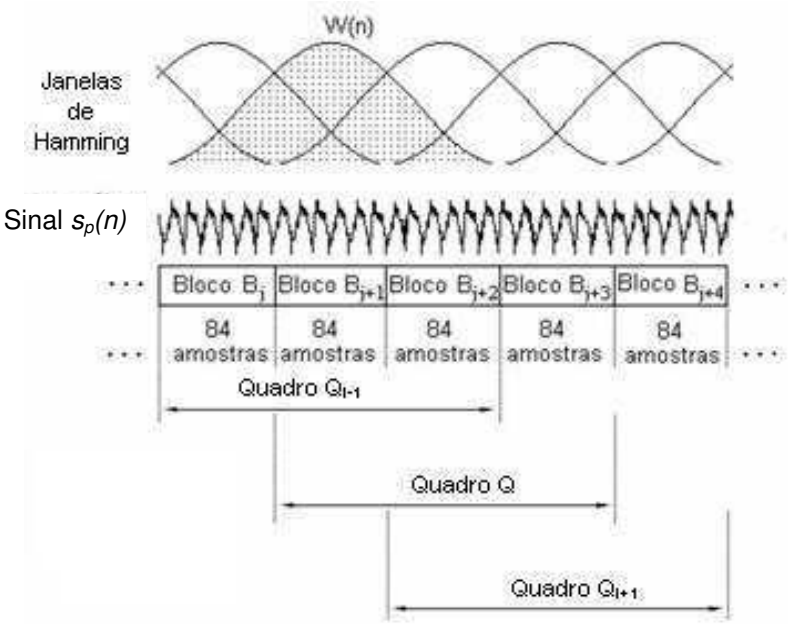


Figura 4.4 – Divisão em Quadros e Janelamento.

Para formar os quadros, as amostras depois de serem processadas pela função de pré-ênfase são inicialmente segmentadas em blocos, em que cada bloco contém 84 amostras, não existindo superposição entre blocos, mas apenas entre quadros. Cada quadro, Q_i, será composto por três blocos sucessivos, totalizando 252 amostras. A divisão dos quadros em 3 blocos de 84 amostras, ou seja, 1/3 do tamanho do quadro, possibilita uma superposição de aproximadamente 66%, como pode ser observado na Figura 4.4. À medida que cada quadro vai sendo formado, suas amostras vão sendo multiplicadas pelo valor da janela de *Hamming* correspondente, como é ilustrado na Figura 4.5.

Para a implementação em *hardware* do algoritmo de divisão em quadros, utiliza-se uma memória *RAM R/W* de dupla porta, que permite a leitura e a escrita de forma paralela. Esta memória é dividida em três segmentos *M(0)*, *M(1)* e *M(2)*, de 84 amostras cada um. Em cada segmento de memória serão armazenadas as amostras dos blocos que formarão um quadro. O resultado da multiplicação de cada amostra do quadro final pelo valor correspondente da janela de *Hamming* é armazenado em um registrador.

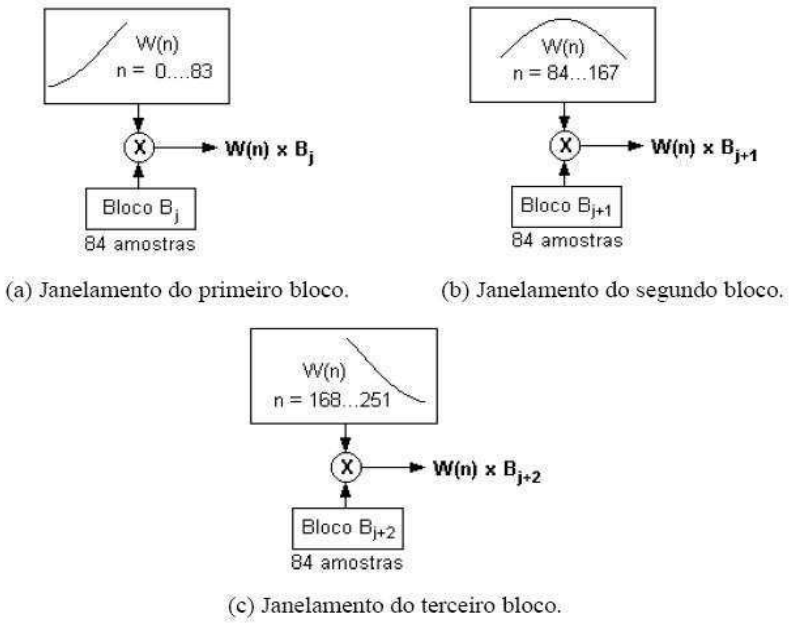


Figura 4.5 – Multiplicação dos blocos pela janela de *Hamming*.

Os quadros são formados pelo agrupamento dos segmentos da memória. Para permitir o trabalho em paralelo das operações de leitura e escrita na memória, a seqüência em que os segmentos de memória são agrupados ocorre ciclicamente, como mostrado na Figura 4.6.

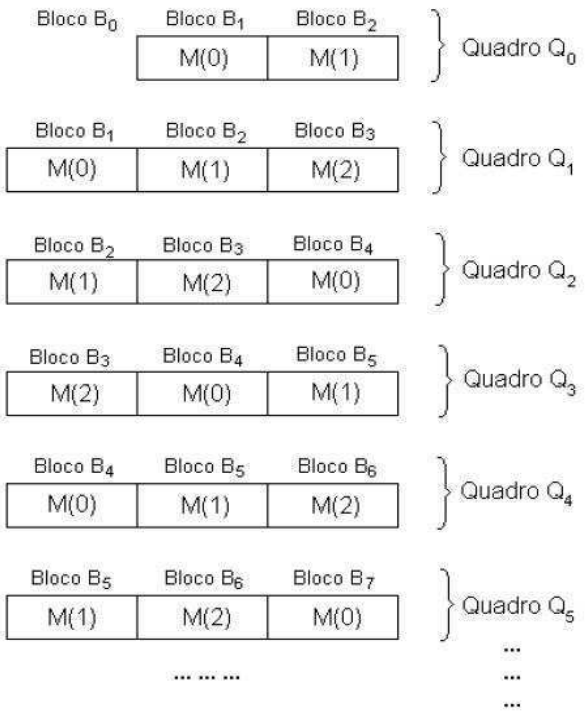


Figura 4.6 - Seqüência de blocos e seu armazenamento nos segmentos da memória.

As amostras do primeiro bloco (o bloco B_0) não precisam ser armazenadas na memória, pois serão processadas diretamente, multiplicando-as pela porção da janela de *Hamming* correspondente. As amostras do segundo bloco (o bloco B_1) são armazenadas no segmento de memória *M(0)* e as amostras do terceiro bloco (o bloco B_2) são armazenadas no segmento de memória *M(1)*. Assim, o primeiro quadro Q_0 será formado pelas amostras do primeiro bloco (que não precisou ser armazenado) e as amostras armazenadas nos segmentos de memória *M(0)* e *M(1)*.

Em um tempo posterior, as amostras do quarto bloco (o bloco B_3) são armazenadas no segmento de memória *M(2)*. Neste momento não é necessário alterar o conteúdo dos segmentos *M(0)* e *M(1)*. Assim, tem-se o segundo quadro, Q_1, que é formado pelas amostras armazenadas nos segmentos de memória *M(0)*, *M(1)* e *M(2)*.

Continuando com o processamento, as amostras do quinto bloco (o bloco B_4) são armazenadas no segmento de memória *M(0)*. O conteúdo anterior deste segmento (o bloco B_1), não será mais necessário para o processamento posterior. Neste momento não é necessário alterar os segmentos *M(1)* e *M(2)*, que contêm os blocos B_2 e B_3. Assim, tem-se o terceiro quadro, Q_2, o qual é formado pelas amostras armazenadas nos segmentos de memória *M(1)*, *M(2)* e *M(0)*, gerando uma nova seqüência de segmentos.

Na continuação, as amostras do sexto bloco (o bloco B_5) são armazenadas no segmento *M(1)*. O conteúdo anterior deste segmento (o bloco B_2), não será mais utilizado no processamento posterior. No entanto, o conteúdo dos segmentos *M(2)* e *M(0)* não é alterado (blocos B_3 e B_4), neste momento. Assim, tem-se o quarto quadro (o quadro Q_3), formado pelas amostras armazenadas nos segmentos de memória *M(2)*, *M(0)* e *M(1)*.

O processamento dos demais blocos é realizado segundo o algoritmo explicado anteriormente, o que implica nas seqüências de escrita e leitura apresentadas na Figura 4.7.

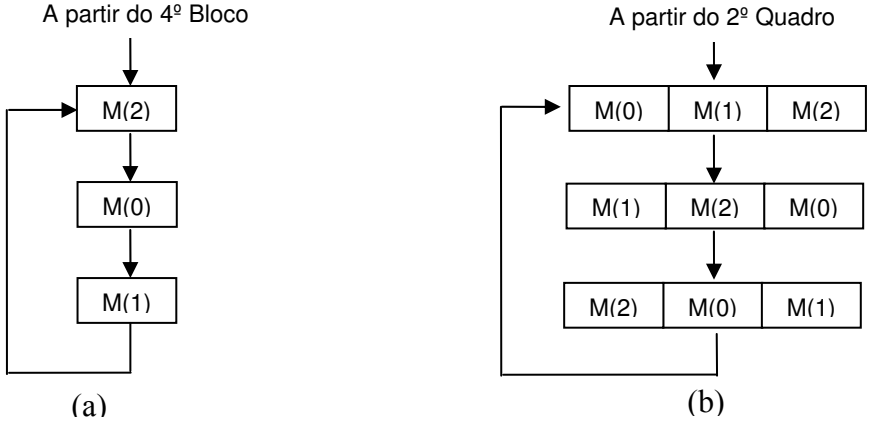


Figura 4.7 – Seqüências de Escrita dos Blocos (a) e Leitura dos Quadros (b).

A escrita de amostras nos segmentos da memória é realizada paralelamente com a operação de leitura dos segmentos. Isto significa que, enquanto um segmento da memória está sendo escrito, outro segmento da memória está sendo lido e multiplicado pela janela de *Hamming*. Para tanto, foi necessária uma velocidade da operação de leitura da memória igual ao triplo da velocidade da operação de escrita. Isto significa que, enquanto são lidas 84 amostras que estão armazenadas em um segmento da memória, são recebidas do sinal de voz e armazenadas em um outro segmento apenas 28 amostras.

Para ilustrar o funcionamento desse algoritmo (Figura 4.8), considera-se o quadro Q_i formado pelos blocos B_j, B_{j+1} e B_{j+2}, armazenados nos segmentos M(0), M(1) e M(2) respectivamente. O quadro anterior Q_{i-1} foi formado pelos blocos B_{j-1}, B_j e B_{j+1}, que ocupavam os segmentos de memória M(2), M(0) e M(1), respectivamente.

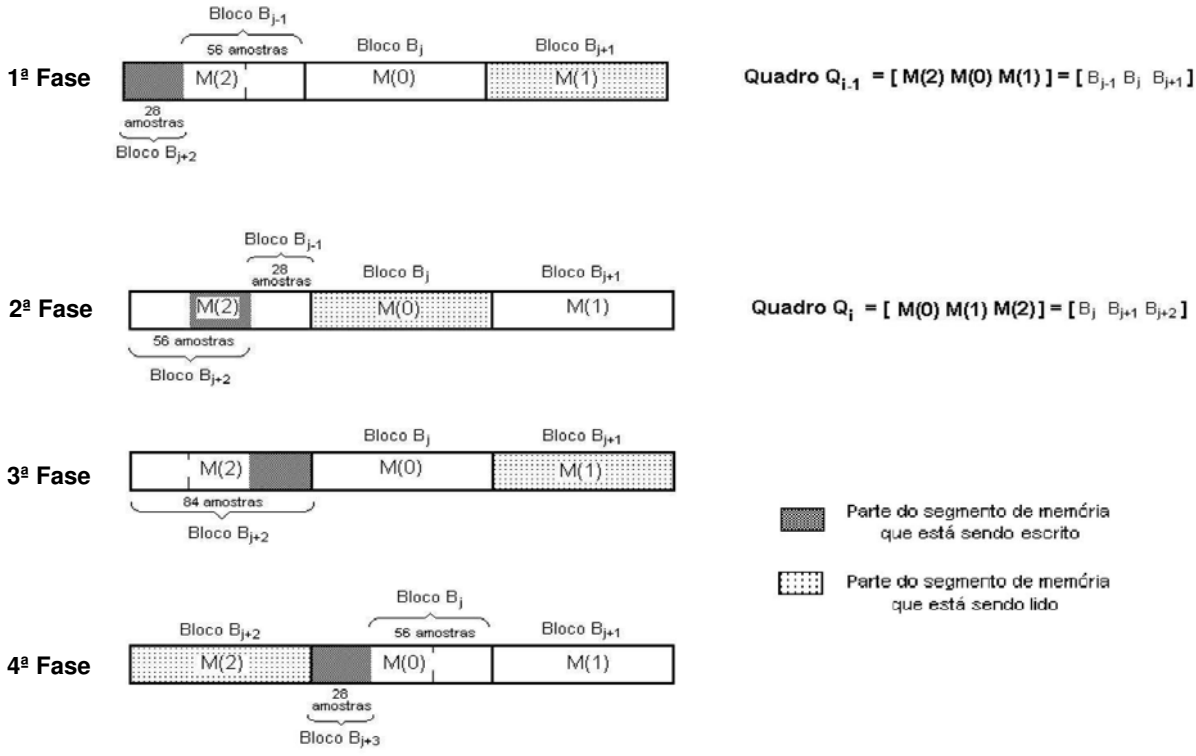


Figura 4.8 – Operações de escrita e leitura nos segmentos de memória.

Quando o último bloco de $Q_{i-1}(B_{j+1})$, armazenado em M(1) está sendo processado, as primeiras 28 amostras de B_{j+2} de Q_i já estão sendo armazenadas em M(2), em seguida, ao iniciar o processamento de Q_i, com a leitura de B_j, armazenado em M(0), paralelamente a segunda parte de B_{j+2} é escrita em M(2). Novamente é lido o conteúdo de M(1), que ainda

contém B_{j+1} (segunda parte de Q_i), enquanto finalmente é escrita a última parte de B_{j+2} em M(2), referente a último bloco de Q_i. O processo continua com a escrita da primeira parte do bloco seguinte (B_{j+3}) em M(0), que será o último segmento lido do próximo quadro Q_{i+1}.

4.3.1 Função de Janelamento

A implementação da janela de *Hamming* foi realizada com uma memória *ROM* que armazena os primeiros 126 valores da janela *W(n)* (*n=0, ..., 125*). A partir destes valores são obtidos os 126 valores restantes de *W(n)* (*n= 126, ..., 251*), utilizando a propriedade de simetria da janela (Figura 4.9).

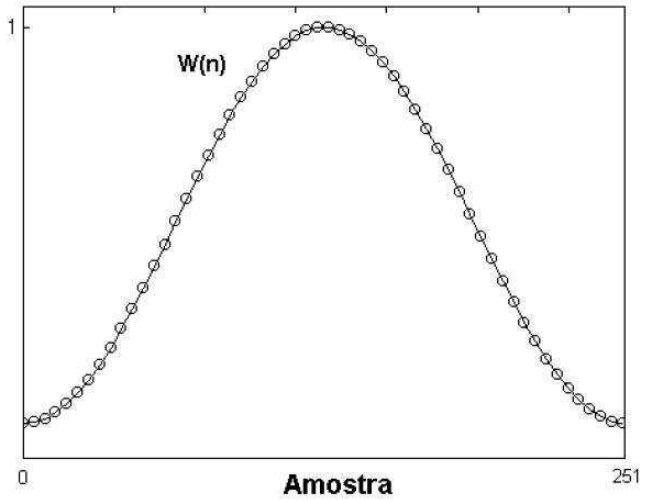


Figura 4.9 – Janela de *Hamming*.

O registrador de endereçamento da memória que contêm os valores da janela de *Hamming* é implementado com um contador binário de 0...125, que opera nos modos crescente/decrescente.

Na Figura 4.10, é apresentada parcialmente a arquitetura do módulo da função de divisão de quadros e janelamento. Os valores da janela de *Hamming* encontram-se armazenados na memória *rom_memory*, de 12 bits, de onde são lidos e multiplicados pelos valores do quadro do sinal de voz pré-enfatizado.

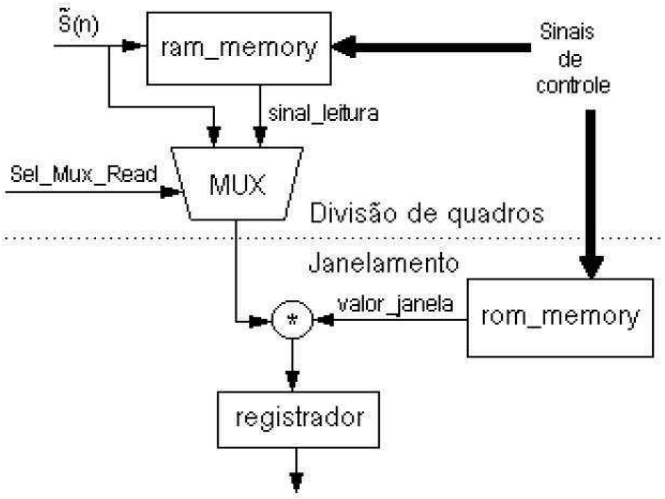


Figura 4.10 – Arquitetura do módulo da função de Divisão em Quadros e Janelamento.

Os valores de um quadro do sinal de voz pré-enfatizado se encontram armazenados na memória *ram_memory*. A memória *ram_memory* é uma memória de duplo acesso síncrona, de 16 bits, que permite a leitura e a escrita de valores em paralelo. Essa memória foi dividida em três segmentos, para a implementação do algoritmo de separação em quadros explicado anteriormente. Foi incluído um multiplexador que seleciona, através do sinal *Sel_Mux_Read*, entre realizar a leitura de um quadro Q_i a partir da memória *RAM* ou a partir do sinal pré-enfatizado. Isso permite o processamento do primeiro bloco de 84 amostras que é multiplicado diretamente pela janela, sem necessidade de lê-lo da memória *RAM*. Para multiplicar a janela de *Hamming* por cada quadro formado, a partir do sinal de voz, utilizou-se um multiplicador de 12 bits x 16 bits e o resultado de 28 bits foi reduzido para 16 bits e armazenado em um registrador de saída.

Na Figura 4.11, é apresentada a descrição de como é realizado o endereçamento das memórias utilizadas. A leitura da memória *ROM* é realizada à medida que vai sendo calculado o janelamento das amostras e é controlada pelo componente *up_down counter* com duas freqüências de *clock* diferentes f_{ck2} e f_{ck1}, sendo $f_{ck2}=3f_{ck1}$. A freqüência f_{ck1} é a freqüência com que os dados do sinal de voz são recebidos da pré-ênfase. Como foi mencionado anteriormente, o primeiro bloco não é lido da memória, e sim processado diretamente do sinal de voz pré-enfatizado, o que torna necessária a leitura dos valores da janela de *Hamming,* nesse instante, com a freqüência f_{ck1}. No entanto, a partir do segundo bloco todos os valores são lidos da memória *RAM*. Como a escrita na memória *RAM* é realizada conforme as amostras do sinal de voz pré-enfatizado são recebidas, ou seja, com freqüência f_{ck1}, e devem

ser lidos três blocos da memória *RAM*, enquanto é escrito apenas um, a freqüência de leitura da *RAM* deve ser três vezes a freqüência de escrita, ou seja, f_{ck2}.

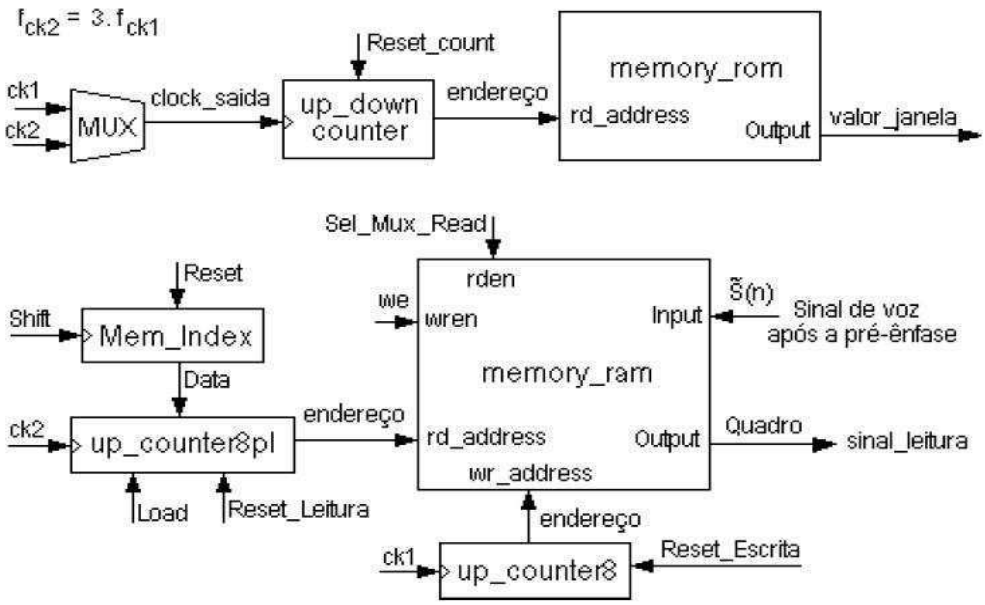


Figura 4.11 – Endereçamento das memórias RAM e ROM nas funções de divisão em quadros e janelamento.

A memória *RAM* utiliza o contador *up_counter8*, de 8 bits, para o cálculo dos endereços de escrita. Na memória são armazenadas 252 amostras, portanto, uma vez que o contador atinge o valor 251 ele é reinicializado e a contagem recomeça. O cálculo dos endereços de leitura da memória *RAM* é feito através de um contador *up_counter8pl*, de 8 bits, com uma entrada *Load* para carregar o valor a partir do qual o contador inicia a contagem. Isto permite selecionar o endereço da *RAM* a partir do qual as amostras serão lidas. Este endereço pode ser 0, 84 ou 168, que são os endereços onde começam os três segmentos em que a *RAM* foi dividida. Este valor é fornecido pelo circuito *Mem_Index* apresentado na Figura 4.12. O circuito *Mem_index* foi implementado com três registradores em cascata e um contador *Counter7*, de 7 bits, que faz a contagem de 0, ..., 83, contando o número de valores lidos de um segmento de memória (até 84 valores). Em cada registrador é armazenado um dos endereços que cada segmento de memória inicia e o sinal de controle *Shift* servirá como *clock* indicando a circulação dos valores entre cada registrador. Na saída, *Data*, ter-se-á o endereço inicial para ser carregado no contador up_*counter8pl*.

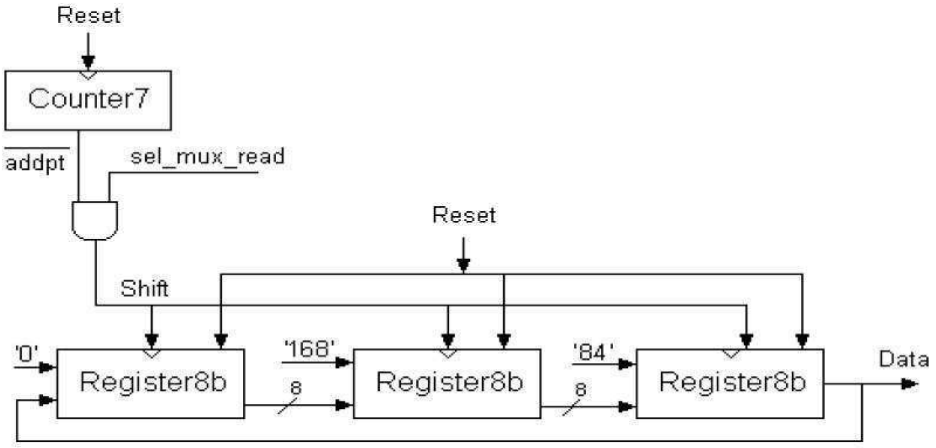


Figura 4.12 – Componente Mem_Index.

4.4 Considerações Gerais

Neste capítulo, foi realizada a descrição do sistema desenvolvido, apresentando seus módulos e funções, bem como a comunicação entre eles.

O módulo de pré-enfase inclui um somador, um subtrator, um acumulador, dois registradores de deslocamento e um circuito de divisão que realiza a operação através de um simples deslocamento de 4 bits à direita.

No módulo de divisão em quadros e janelamento, foi apresentado o algoritmo de leitura e escrita paralela na memória RAM e o armazenamento e leitura dos valores da janela de *Hamming* na memória ROM.

Capítulo 5

Resultados Obtidos

5.1 Apresentação

O trabalho desenvolvido consiste de um módulo sintetizável para pré-processamento digital de sinais de voz, que inclui as funções de pré-ênfase, divisão em quadros e janelamento. Este módulo foi projetado e implementado visando a sua reutilização como um componente, em sistemas embutidos para reconhecimento de fala ou de locutor.

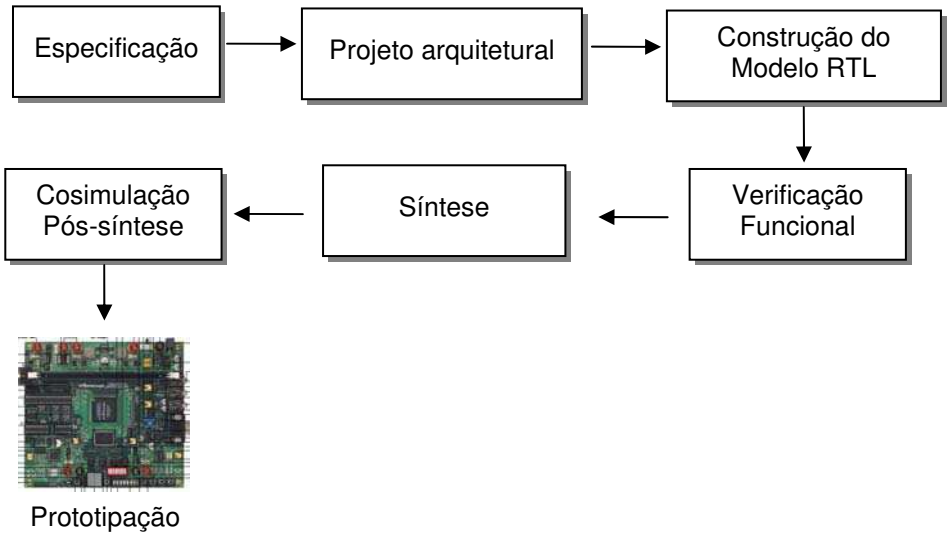


Figura 5.1 – Fluxo de desenvolvimento do IP Core de pré-processamento digital de sinais de voz.

Na Figura 5.1 são apresentadas as etapas necessárias para obtenção de um IP *Core*, as quais foram descritas no Capítulo 3 e executadas até a etapa de prototipação ao longo deste trabalho.

Neste capítulo, serão descritos os resultados obtidos em cada etapa de desenvolvimento.

5.2 Especificação

O sistema desenvolvido consiste de uma biblioteca de pré-processamento digital de sinais de voz adaptada para sistemas embutidos, incluindo as funções de pré-ênfase, divisão em quadros e janelamento.

5.3 Projeto Arquitetural

Na Figura 5.2, é apresentado o diagrama do sistema. Para função de janelamento pode ser utilizado qualquer tipo de janela (*Hamming, Hanning,* Retangular, etc), desde que o tamanho da mesma seja igual a 252. Os valores da janela são armazenados na memória ROM e lidos durante o processamento das amostras.

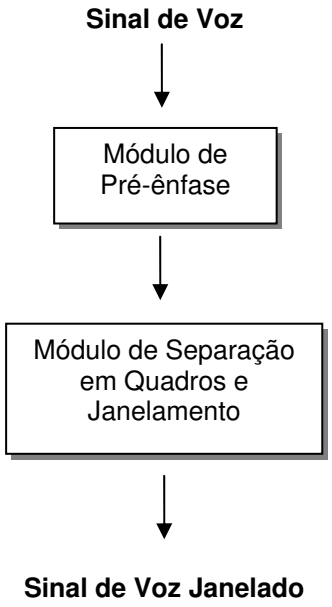


Figura 5.2 – Diagrama do Sistema Desenvolvido.

5.4 Modelo RTL

O modelo RTL do sistema é baseado nos modelos de pré-ênfase, divisão em quadros e janelamento descritos e implementados em VHDL por [Cipriano 2001], os quais foram apresentados no Capítulo 4. Foram realizadas algumas adaptações nesses modelos, de modo a atender às especificações pretendidas por este trabalho.

Para gerar os valores da janela foi criado um programa na linguagem C (Figura 5.3), que cria um arquivo do tipo *.mif*, o qual é utilizado na inicialização da memória ROM. A linha de código destacada na Figura 5.3 refere-se à equação da janela de *Hamming* de 252 amostras, a qual pode ser substituída pela equação da janela desejada.

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159265358979323846264338327795

main(int argc, char **argv) {

  FILE  *arqD;
  int i;
  float w;
  char valor;

  if ((arqD = fopen("teste.mif","wt")) != NULL) {

     fputs("WIDTH = 16;\nDEPTH = 126;\n\nADDRESS_RADIX =
     HEX;\nDATA_RADIX = DEC;\n\nCONTENT BEGIN\n",arqD);

      for (i = 0; i < 126; i++) {
        w = 0.54 - (0.46*cos((2*PI*i)/251));
        fprintf(arqD,"%x \t : \t %0.4f;\n",i,w);
      }

      fputs("END;",arqD);

      fclose(arqD);

 } else {
   printf("\n Arquivo nao pode ser aberto");
 }

}
```

Figura 5.3 – Programa *geramif.c*.

5.5 Verificação Funcional

Para realização da verificação funcional, foi utilizada a metodologia VeriSC descrita no Capítulo 3. Embora o modelo RTL tenha sido descrito em VHDL, o ambiente de simulação (*testbench*), gerado através da ferramenta VeriSC, foi escrito na linguagem SystemC. A utilização de SystemC para descrever o *testbench* oferece as vantagens citadas anteriormente, como por exemplo, a inclusão de nível de transação, cobertura dirigida, checagem automática e construção de verificação funcional randômica.

A ferramenta *CoCentric SystemC HDL CoSim* [Synopsys 2006] foi utilizada para permitir a cosimulação dos módulos descritos em VHDL com o *testbench* descrito em SystemC.

Na primeira etapa, foi realizada a verificação funcional isolada de cada módulo do sistema. Como entrada para teste dos módulos foram utilizados 10 sinais de voz, os quais são apresentados na Tabela 5.1.

Os sinais de voz foram capturados utilizando:

- Microfone Dynamic LeSon MK60 de 600Ω de impedância;
- Placa de som Sis 7018 PCI Audio.

Tabela 5.1 – Amostras utilizadas nos testes.

Fonemas	Sílabas	Palavras	Frases
/a/	/ba/	/pet/	/Quero usar a máquina./
	/za/	/aplausos/	/O maior São João do mundo ocorre em Campina Grande./
		/bola/	
		/zero/	
		/fui/	

Na Figura 5.4, é exibida uma parte das amostras da sílaba /ba/ utilizada como entrada das simulações apresentadas nas seções seguintes.

Nas seções 6.5.1 e 6.5.2, são apresentados, respectivamente, os resultados da verificação funcional do módulo de pré-ênfase e do módulo de divisão em quadros e janelamento.

```
>more ba.txt
-145  -159  -175  -188  -203  -218  -233  -249  -265  -277  -291  -305  -317  -328
-338  -351  -359  -368  -379  -388  -397  -408  -417  -426  -436  -444  -454  -462
-468  -474  -479  -486  -492  -497  -501  -507  -511  -515  -520  -526  -533  -539
-545  -551  -555  -561  -568  -574  -580  -587  -591  -596  -600  -603  -605  -608
-612  -615  -616  -615  -616  -616  -615  -613  -610  -608  -604  -603  -603  -603
-605  -606  -609  -611  -614  -620  -625  -628  -631  -634  -637  -640  -643  -642
-643  -645  -646  -647  -651  -655  -660  -664  -670  -676  -685  -691  -697  -701
-706  -705  -708  -711  -713  -716  -717  -723  -730  -734  -741  -747  -755  -762
-772  -780  -788  -801  -811  -822  -834  -845  -858  -873  -886  -899  -915  -929
-941  -953  -965  -974  -983  -991  -998  -1006  -1015  -1024  -1031  -1042  -1054
-1065  -1078  -1091  -1102  -1117  -1130  -1145  -1159  -1173  -1184  -1195  -1207
-1217  -1231  -1242  -1252  -1264  -1275  -1289  -1302  -1317  -1332  -1345  -1357
-1364  -1373  -1381  -1385  -1391  -1394  -1400  -1400  -1400  -1397  -1396  -1394
-1385  -1381  -1373  -1364  -1355  -1342  -1331  -1316  -1301  -1287  -1268  -1251
-1232  -1215  -1199  -1182  -1167  -1150  -1133  -1115  -1096  -1077  -1053  -1029
-1006  -979  -957  -935  -914  -889  -868  -851  -832  -815  -801  -787  -775  -
760  -748  -732  -720  -709  -699  -688  -675  -664  -654  -644  -637  -630  -623
-615  -609  -607  -603  -598  -592  -587  -582  -578  -572  -568  -561  -557  -554
-551  -553  -550  -545  -545  -540  -534  -530  -525  -518  -513  -511  -506  -505
-504  -505  -510  -517  -520  -527  -536  -546  -556  -566  -576  -588  -600  -616
-637  -657  -682  -712  -743  -777  -814  -854  -895  -937  -979  -1023  -1069  -
1116  -1168  -1220  -1271  -1323  -1375  -1429  -1486  -1544  -1600  -1656  -1709
-1765  -1820  -1871  -1919  -1963  -2002  -2039  -2074  -2108  -2135  -2159  -2181
  -2202  -2218  -2233  -2241  -2243  -2243  -2234  -2222  -2206  -2185  -2160  -
2130  -2094  -2062  -2024  -1983  -1939  -1895  -1844  -1795  -1744  -1688  -1637
-1579  -1521  -1462  -1399  -1338  -1276  -1215  -1154  -1095  -1038  -982  -931
-880  -832  -789  -742  -699  -657  -621  -587  -555  -526  -500  -477  -458  -439
-423  -406  -388  -371  -360  -347  -333  -318  -307  -290  -270  -254  -235  -220
-201  -183  -166  -151  -132  -115  -98  -84  -69  -54  -40  -28  -15  -3  8  20
29  31  33  39  36  24  3  -20  -48  -83  -118  -156  -196  -236  -277  -319  -363
-410  -458  -506  -557  -615  -672  -732  -792  -851  -916  -979  -1038  -1096  -
1151  -1200  -1249  -1294  -1333  -1370  -1402  -1430  -1456  -1476  -1498  -1520
-1540  -1556  -1570  -1582  -1593  -1601  -1609  -1614  -1621  -1627  -1635  -1641
-1645  -1647  -1648  -1648  -1641  -1635  -1630  -1630  -1629  -1629  -1636
-1643  -1652  -1662  -1674  -1688  -1699  -1708  -1712  -1716  -1714  -1712  -1704
-1696  -1680  -1660  -1641  -1620  -1600  -1578  -1552  -1528  -1504  -1481  -1452
-1425  -1396  -1362  -1326  -1285  -1239  -1192  -1143  -1091  -1038  -983  -928
-876  -824  -775  -723  -674  -621  -572  -524  -475  -426  -379  -333  -286  -243
-203  -166  -128  -94  -62  -33  -9  12  33  56  77  98  116  131  147  161  168
175  178  185  191  195  200  208  213  224  232  240  253  267  281  296  310
324  338  347  347  333  311  287  260  234  205  174  146  114  84
--More--(8%)
```

Figura 5.4 – Amostras da sílaba /ba/.

6.5.1 Pré-ênfase

A etapa de verificação funcional isolada do módulo de pré-ênfase incluiu 5000 transações[2] para cada sinal testado.

[2] Uma transação consiste de uma operação que inicia em um determinado instante de tempo, a partir de um estímulo, e termina em outro instante de tempo gerando uma saída. É caracterizada por um conjunto de instruções e dados necessários para realização de tal operação.

Na Figura 5.5, são apresentadas as formas de onda da simulação da pré-ênfase. De cima para baixo na figura, são exibidos os seguintes sinais: *clk*, *reset*, *Fim_Sinal*, *Sinal_Entrada*, *Done_Preenfase* e *Sinal_Saida*, que representam as seguintes informações:

- *clk* – *clock* do sistema;
- *reset* – quando igual a "0" reinicia o módulo;
- *Fim_Sinal* – quando igual a "1" indica o fim do sinal de voz de entrada;
- *Sinal_Entrada* – amostras do sinal de voz a ser processado;
- *Done_Preenfase* – para inicialização dos registradores que compõem o módulo são necessários dois ciclos de *clock,* e antes disso os valores de saída não devem ser considerados válidos. Diante disso, *Done_Preenfase* é utilizado para indicar que no próximo ciclo de *clock* os valores de *Sinal_Saida* correspondem ao sinal pré-enfatizado do sinal original e podem ser considerados válidos.
- *Sinal_Saida* – amostras do sinal pré-enfatizado.

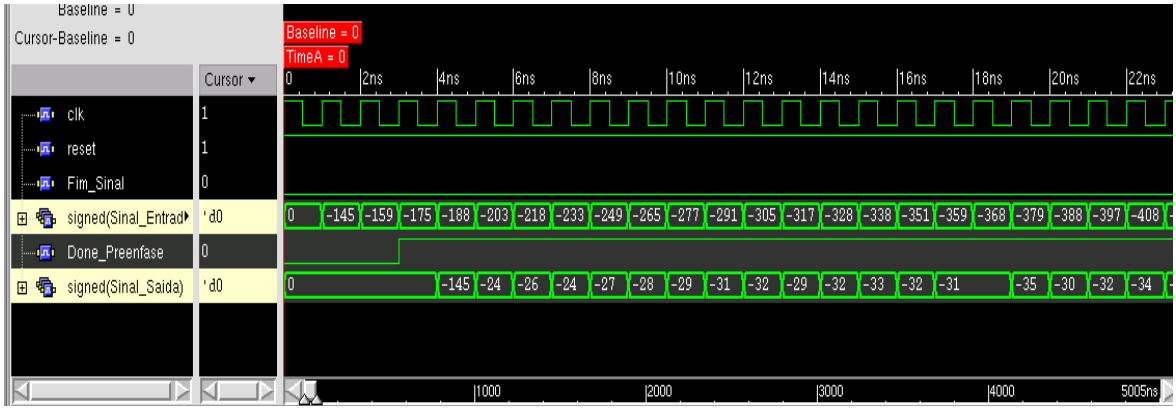


Figura 5.5 – Simulação da função de pré-ênfase.

Na Tabela 5.2, é apresentado o cálculo da pré-ênfase para as amostras de entrada (*Sinal_Entrada)* da simulação apresentada na Figura 5.5, com base nas equações apresentadas no Capítulo 4. São apresentados os resultados, tanto para a Equação 4.1, que corresponde à descrição matemática original da pré-ênfase, quanto para a Equação 4.2, que consiste da adaptação efetuada para o sistema implementado. O valor utilizado para o fator de pré-ênfase (*ap*) foi igual a 0,9375 correspondendo a 15/16, cuja utilização foi justificada no Capítulo 4 (Seção 4.2).

Tabela 5.2 – Cálculo da Pré-ênfase.

Clock	Entrada s(n)	Pré-ênfase (Equação 4.1) $s_p(n) = s(n) - ap.s(n-1)$	Pré-ênfase (Equação 4.2) $s_p(n) = s(n) - s(n-1) + s(n-1)/16$	Resultado da Simulação
1	0	0	0	0
2	-145	-145	-145	-145
3	-159	-23.0625	-23.0625	-24
4	-175	-25.9375	-25.9375	-26
5	-188	-23.9375	-23.9375	-24
6	-203	-26.75	-26.75	-27
7	-218	-27.6875	-27.6875	-28
8	-233	-28.625	-28.625	-29
9	-249	-30.5625	-30.5625	-31
10	-265	-31.5625	-31.5625	-32
11	-277	-28.5625	-28.5625	-29
12	-291	-31.3125	-31.3125	-32
13	-305	-32.1875	-32.1875	-33
14	-317	-31.0625	-31.0625	-32
15	-328	-30.8125	-30.8125	-31

Observando-se *Sinal_Saida* (Figura 5.5) e comparando-se com os resultados da Tabela 5.2, é possível verificar o funcionamento correto do módulo de pré-ênfase desenvolvido. Devido à realização da divisão através de um deslocamento, é obtido um erro de truncamento no cálculo final da pré-ênfase. No entanto, esse erro é considerado insignificante diante da simplicidade e conseqüente economia de recursos obtidas no módulo desenvolvido.

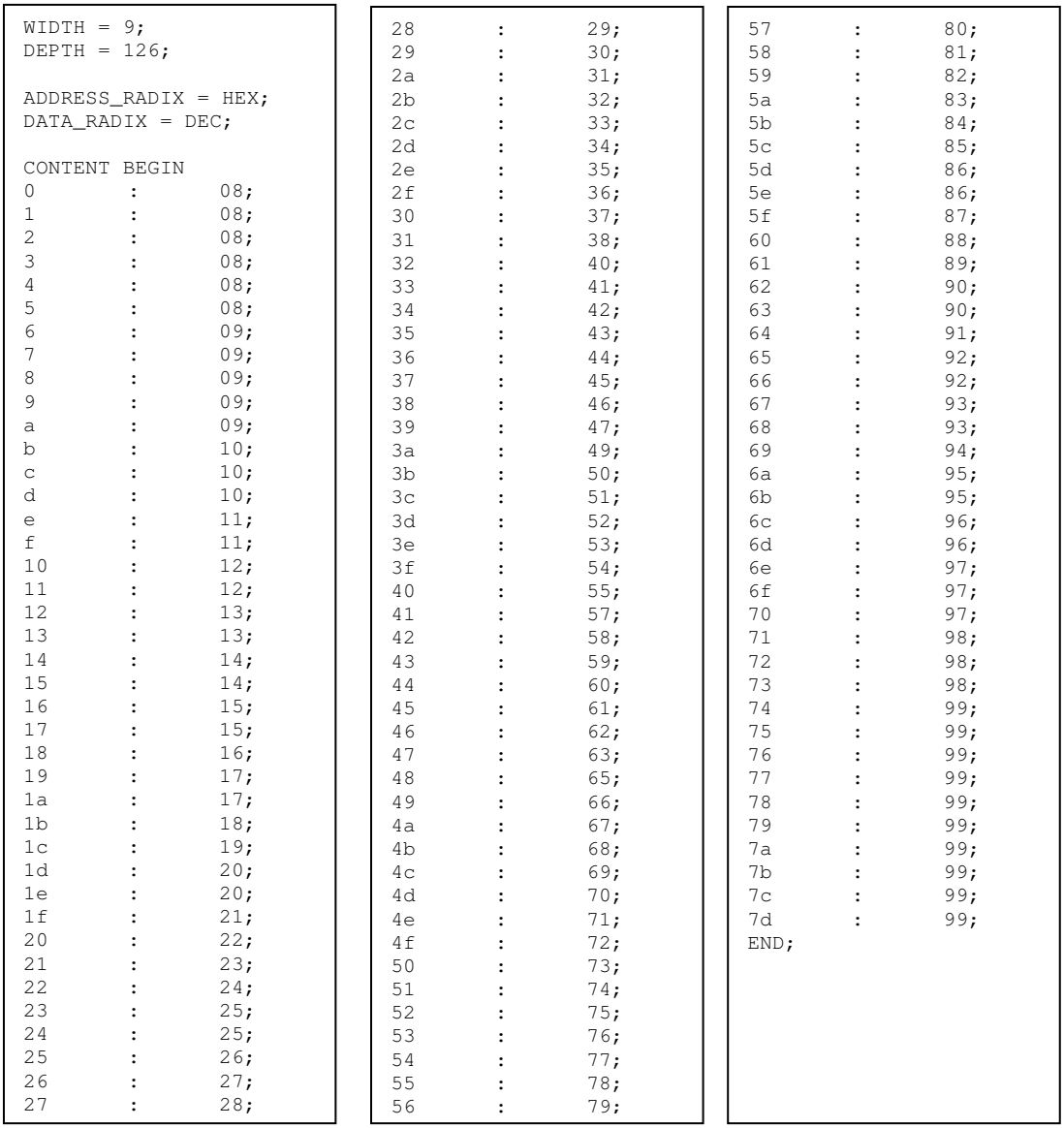
O total de linhas de código VHDL para este módulo foi 227 e, para o seu respectivo *testbench,* 370 linhas de código em SystemC, incluindo o modelo de referência.

6.5.2 Divisão em Quadros e Janelamento

A verificação funcional da divisão em quadros e janelamento também foi realizada isoladamente, ou seja, as amostras de voz foram processadas diretamente, sem a execução da pré-ênfase. Nessa etapa também foram realizadas 5000 transações para cada sinal testado.

O módulo desenvolvido pode ser utilizado com qualquer tipo de janela desejado desde que seu tamanho seja igual a 252. Nos testes apresentados, foi empregada a janela de *Hamming.* Uma vez que os valores da janela estão na faixa $0 < w(n) < 1$, ou seja, a parte inteira é sempre igual a 0, para facilitar o uso de ponto-fixo na ROM são gravados apenas os

dois algarismos (7 bits) mais significativos referentes à parte fracionária. Na Figura 5.6, é apresentado o arquivo *.mif* gerado para a janela de *Hamming* com 7 bits.



```
WIDTH = 9;
DEPTH = 126;

ADDRESS_RADIX = HEX;
DATA_RADIX = DEC;

CONTENT BEGIN
0       :       08;
1       :       08;
2       :       08;
3       :       08;
4       :       08;
5       :       08;
6       :       09;
7       :       09;
8       :       09;
9       :       09;
a       :       09;
b       :       10;
c       :       10;
d       :       10;
e       :       11;
f       :       11;
10      :       12;
11      :       12;
12      :       13;
13      :       13;
14      :       14;
15      :       14;
16      :       15;
17      :       15;
18      :       16;
19      :       17;
1a      :       17;
1b      :       18;
1c      :       19;
1d      :       20;
1e      :       20;
1f      :       21;
20      :       22;
21      :       23;
22      :       24;
23      :       25;
24      :       25;
25      :       26;
26      :       27;
27      :       28;
28      :       29;
29      :       30;
2a      :       31;
2b      :       32;
2c      :       33;
2d      :       34;
2e      :       35;
2f      :       36;
30      :       37;
31      :       38;
32      :       40;
33      :       41;
34      :       42;
35      :       43;
36      :       44;
37      :       45;
38      :       46;
39      :       47;
3a      :       49;
3b      :       50;
3c      :       51;
3d      :       52;
3e      :       53;
3f      :       54;
40      :       55;
41      :       57;
42      :       58;
43      :       59;
44      :       60;
45      :       61;
46      :       62;
47      :       63;
48      :       65;
49      :       66;
4a      :       67;
4b      :       68;
4c      :       69;
4d      :       70;
4e      :       71;
4f      :       72;
50      :       73;
51      :       74;
52      :       75;
53      :       76;
54      :       77;
55      :       78;
56      :       79;
57      :       80;
58      :       81;
59      :       82;
5a      :       83;
5b      :       84;
5c      :       85;
5d      :       86;
5e      :       86;
5f      :       87;
60      :       88;
61      :       89;
62      :       90;
63      :       90;
64      :       91;
65      :       92;
66      :       92;
67      :       93;
68      :       93;
69      :       94;
6a      :       95;
6b      :       95;
6c      :       96;
6d      :       96;
6e      :       97;
6f      :       97;
70      :       97;
71      :       98;
72      :       98;
73      :       98;
74      :       99;
75      :       99;
76      :       99;
77      :       99;
78      :       99;
79      :       99;
7a      :       99;
7b      :       99;
7c      :       99;
7d      :       99;
END;
```

Figura 5.6 – Arquivo *.mif* contendo valores da janela de *Hamming* com 7 bits.

Na Figura 5.7, são apresentadas as formas de onda da simulação da divisão em quadros e janelamento, identificadas da seguinte forma:

- *clk* e *clk_Read* - referem-se, respectivamente, às freqüências f_{ck1} e f_{ck2}, explicadas no Capítulo 4. Sendo f_{ck1} (*clk*) a freqüência com a qual as amostras são recebidas da pré-ênfase e escritas na memória RAM, enquanto que, f_{ck2} (*clk_Read*) é a freqüência com a qual as amostras são lidas da RAM.
- *reset e Fim_Sinal* - têm o mesmo significado explicados anteriormente para a pré-ênfase*;*

- *input_signal* – amostras do sinal de voz a ser processado;
- *Resultado* – amostras do sinal janelado. Devido ao uso de ponto-fixo, o resultado do janelamento deve ser lido da seguinte forma: os últimos dois dígitos são considerados como a parte fracionária e os demais são a parte inteira.

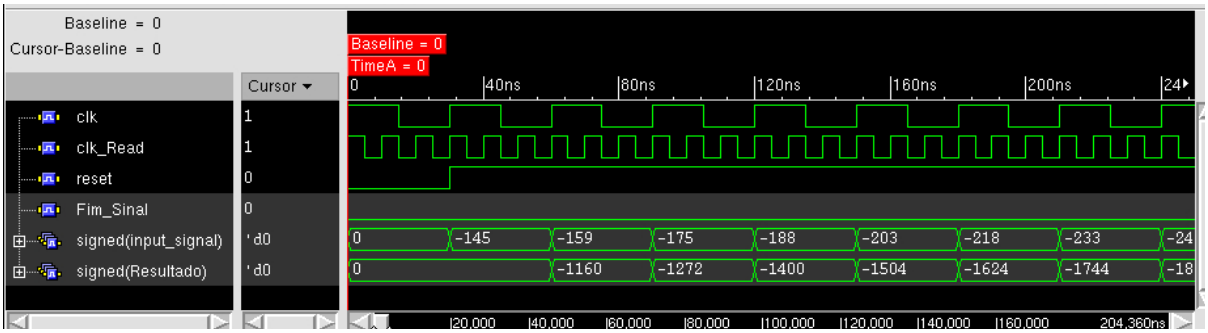


Figura 5.7 – Simulação da função de divisão em quadros e janelamento.

Na Tabela 5.3, é apresentado o cálculo do janelamento para as amostras de entrada (*input_signal*) da simulação apresentada na Figura 5.7, com base nas equações apresentadas no Capítulo 5 e nos valores da janela de *Hamming* apresentados na Figura 5.6.

Tabela 5.3 – Cálculo do Janelamento.

Clock	Entrada s(n)	Valor Teórico da Janela de Hamming $W_t(n)$	Valor da Janela de Hamming utilizado W(n)	Janelamento Teórico $J(n) = s(n).W_t(n)$	Resultado da Simulação $J(n) = s(n).W(n)$
1	0	0	0	0	0
2	-145	0.0800	08	-11.6000	-1160
3	-159	0.0801	08	-12.7359	-1272
4	-175	0.0806	08	-14.1050	-1400
5	-188	0.0813	08	-15.2844	-1504
6	-203	0.0823	08	-16.7069	-1624
7	-218	0.0836	08	-18.2248	-1744
8	-233	0.0852	08	-19.8516	-1864

A partir da Tabela 5.3, é possível verificar uma aproximação entre os valores obtidos pelo módulo de divisão em quadros e janelamento desenvolvido e o janelamento teórico. Devido à utilização de ponto-fixo, o resultado da simulação deve ser lido da seguinte maneira: os dois últimos algarismos referem-se à parte fracionária do resultado e os demais à parte

inteira. Para aumentar a precisão do módulo desenvolvido, em uma etapa seguinte o número de algarismos referentes à janela de *Hamming* armazenado na memória ROM foi aumentado de 2 para 4.

O total de linhas de código VHDL para este módulo foi 616, e para o seu respectivo *testbench* 376 linhas de código em SystemC, incluindo o modelo de referência.

5.6 Síntese

Para realização da síntese do sistema, foi utilizada a ferramenta Quartus 5.1 da Altera [Altera 2006]. Nesta etapa foi definido que o dispositivo de *hardware* a ser utilizado seria a placa Stratix II EP2S60F672C5ES também da Altera. Detalhes sobre esta placa são apresentados no Apêndice B.

Nesta etapa foram necessárias algumas adaptações no modelo RTL do sistema. A principal modificação refere-se à inclusão de um terceiro *clock*, o qual foi chamado de *clk_ram.* A necessidade deste terceiro *clock* deve-se ao fato de o dispositivo utilizado para prototipação suportar apenas memórias RAM R/W e ROM síncronas. Isto implica que são necessários dois ciclos de *clock* entre o fornecimento do endereço e o recebimento do dado. Como o sistema foi projetado considerando as memórias assíncronas, tornou-se necessário gerar um terceiro *clock*, duas vezes mais rápido do que *clk_read* (explicado anteriormente), para permitir que no primeiro ciclo, do *clock* mais rápido, seja fornecido o endereço e no segundo o dado seja recebido. Esta estratégia possibilitou o funcionamento correto do sistema, em função do dispositivo utilizado. Na Figura 5.8, são apresentadas as formas de onda de *clk, clk_Read* e *clk_ram,* quando é possível vizualisar melhor a relação entre os mesmos.

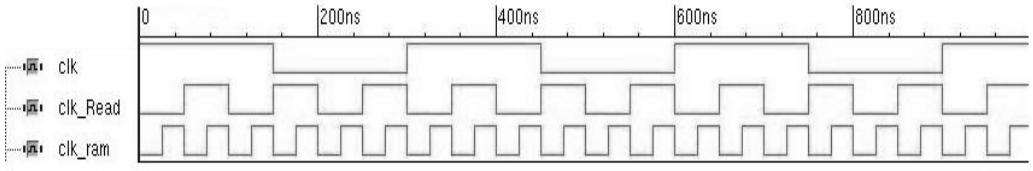
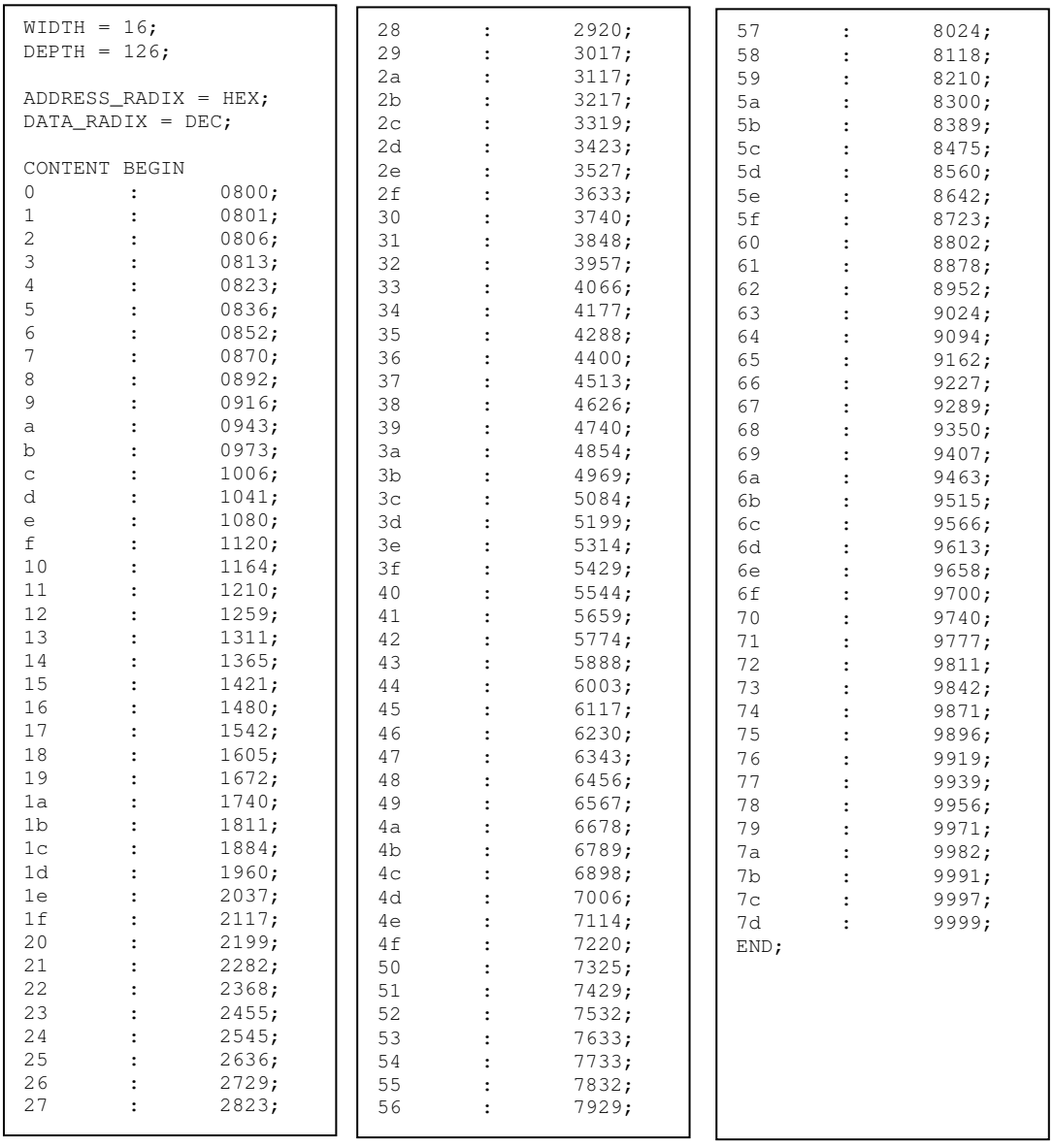


Figura 5.8 – Forma de onda de *clk*, *clk_Read* e *clk_ram*.

Ao final desta etapa foi obtido um módulo sintetizável para a placa Stratix II EP2S60F672C5ES.

5.7 Cosimulação pós-síntese do sistema completo

Após cada módulo ter sido verificado isoladamente, foi realizada a síntese dos módulos trabalhando conjuntamente e, em seguida, a cosimulação pós-síntese. Nesta etapa, com o objetivo de aumentar a precisão do processo de janelamento, foi ampliada de 7 para 14 a quantidade de bits utilizada para os valores da janela armazenados na ROM. Diante disso, ao invés de apenas dois algarismos, passam a ser gravados os quatro algarismos (14 bits) mais significativos referentes à parte fracionária. Na Figura 5.9, é apresentado o arquivo *.mif* gerado para a janela de *Hamming* com 14 bits. A verificação funcional de todo o módulo incluiu 5000 transações.



```
WIDTH = 16;
DEPTH = 126;

ADDRESS_RADIX = HEX;
DATA_RADIX = DEC;

CONTENT BEGIN
0       :       0800;
1       :       0801;
2       :       0806;
3       :       0813;
4       :       0823;
5       :       0836;
6       :       0852;
7       :       0870;
8       :       0892;
9       :       0916;
a       :       0943;
b       :       0973;
c       :       1006;
d       :       1041;
e       :       1080;
f       :       1120;
10      :       1164;
11      :       1210;
12      :       1259;
13      :       1311;
14      :       1365;
15      :       1421;
16      :       1480;
17      :       1542;
18      :       1605;
19      :       1672;
1a      :       1740;
1b      :       1811;
1c      :       1884;
1d      :       1960;
1e      :       2037;
1f      :       2117;
20      :       2199;
21      :       2282;
22      :       2368;
23      :       2455;
24      :       2545;
25      :       2636;
26      :       2729;
27      :       2823;
28      :       2920;
29      :       3017;
2a      :       3117;
2b      :       3217;
2c      :       3319;
2d      :       3423;
2e      :       3527;
2f      :       3633;
30      :       3740;
31      :       3848;
32      :       3957;
33      :       4066;
34      :       4177;
35      :       4288;
36      :       4400;
37      :       4513;
38      :       4626;
39      :       4740;
3a      :       4854;
3b      :       4969;
3c      :       5084;
3d      :       5199;
3e      :       5314;
3f      :       5429;
40      :       5544;
41      :       5659;
42      :       5774;
43      :       5888;
44      :       6003;
45      :       6117;
46      :       6230;
47      :       6343;
48      :       6456;
49      :       6567;
4a      :       6678;
4b      :       6789;
4c      :       6898;
4d      :       7006;
4e      :       7114;
4f      :       7220;
50      :       7325;
51      :       7429;
52      :       7532;
53      :       7633;
54      :       7733;
55      :       7832;
56      :       7929;
57      :       8024;
58      :       8118;
59      :       8210;
5a      :       8300;
5b      :       8389;
5c      :       8475;
5d      :       8560;
5e      :       8642;
5f      :       8723;
60      :       8802;
61      :       8878;
62      :       8952;
63      :       9024;
64      :       9094;
65      :       9162;
66      :       9227;
67      :       9289;
68      :       9350;
69      :       9407;
6a      :       9463;
6b      :       9515;
6c      :       9566;
6d      :       9613;
6e      :       9658;
6f      :       9700;
70      :       9740;
71      :       9777;
72      :       9811;
73      :       9842;
74      :       9871;
75      :       9896;
76      :       9919;
77      :       9939;
78      :       9956;
79      :       9971;
7a      :       9982;
7b      :       9991;
7c      :       9997;
7d      :       9999;
END;
```

Figura 5.9 – Arquivo *.mif* contendo valores da janela de *Hamming* com 14 bits.

Na Figura 5.10, são apresentadas as formas de onda do pré-processamento completo após a síntese, identificadas da seguinte forma:

- *clk* e *clk_Read* – têm o mesmo significado explicado anteriormente para a Divisão em Quadros e Janelamento;
- *clk_ram* – O dispositivo utilizado para prototipação suporta apenas memórias RAM e ROM síncronas, o que implica, que são necessários dois ciclos de *clock* entre o fornecimento do endereço e o recebimento do dado. Como os módulos foram projetados considerando as memórias assíncronas, foi gerado o terceiro *clock*, duas vezes mais rápido que *clk_Read*, para permitir a continuidade do correto funcionamento do sistema. No primeiro ciclo mais rápido é fornecido o endereço e no segundo o dado é recebido.
- *reset e Fim_Sinal* - têm os mesmos significados explicados anteriormente para a pré-ênfase;
- *Sinal_Entrada* – amostras do sinal de voz a ser processado;
- *Resultado* – amostras do sinal janelado. Devido ao uso de ponto-fixo, o resultado do janelamento deve ser lido da seguinte forma: os últimos quatro dígitos são considerados como a parte fracionária e os demais são a parte inteira.

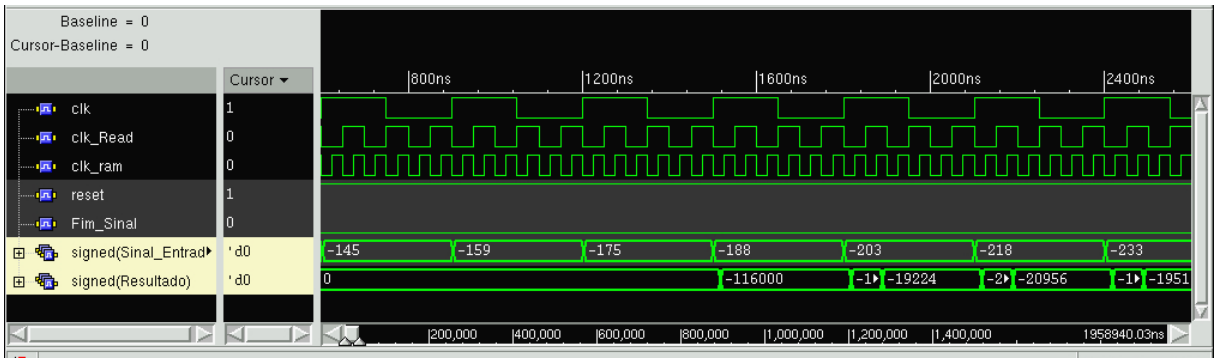


Figura 5.10 – Simulação pós-síntese do Pré-processamento completo.

Na Figura 5.11, é apresentada a saída da verificação funcional do pré-processamento completo, quando são comparados os valores gerados pelo DUV com os valores calculados pelo modelo de referência. Na Figura 5.11, os valores referentes ao DUV são identificados pelo rótulo *MONITOR*, enquanto que os referentes ao modelo de referência, pelo rótulo

MODREF. Os valores do modelo de referência foram arredondados, ou ainda convertidos para inteiros, para permitir sua comparação com os valores produzidos pelo DUV.

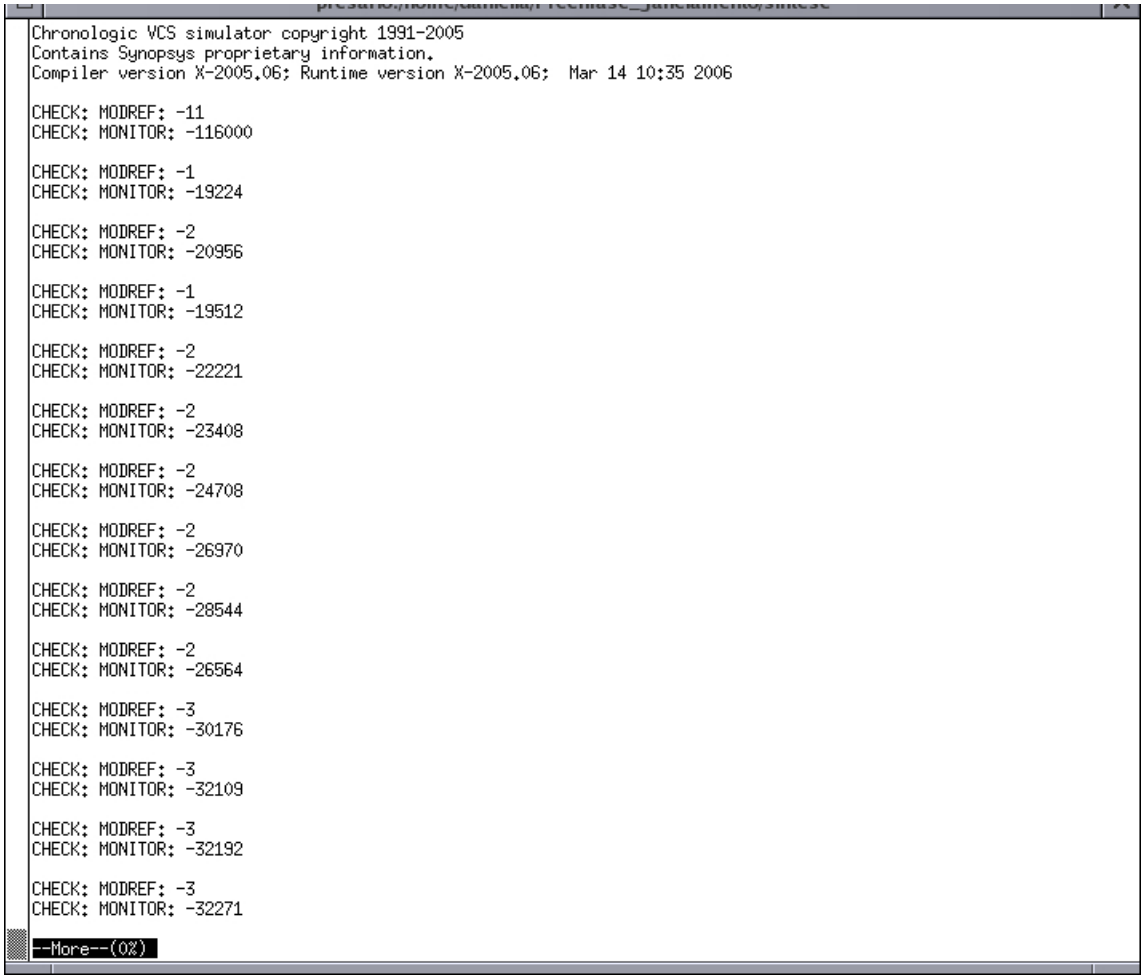


Figura 5.11 – Cosimulação do Pré-processamento completo.

Na Tabela 5.4, é apresentado o cálculo do pré-processamento completo para as amostras de entrada (*Sinal_Entrada)* da simulação apresentada na Figura 5.10, com base nas equações apresentadas no Capítulo 5 e nos valores da janela de *Hamming* apresentados na Figura 5.9.

Comparando-se o cálculo do janelamento teórico com o resultado da simulação apresentados na Tabela 5.4 é possível verificar o funcionamento correto do módulo de pré-processamento desenvolvido.

Tabela 5.4 – Cálculo do Pré-processamento.

Clock	Entrada s(n)	Resultado da Pré-ênfase $S_p(n)$	Valor Teórico da Janela de Hamming $W_t(n)$	Valor da Janela de *Hamming* utilizado W(n)	Janelamento Teórico $J_t(n) = S_p(n).W_t(n)$	Resultado da Simulação $J(n) = S_p(n).W(n)$
1	0	0	0	0	0	0
2	-145	-145	0.0800	0800	-11.6000	-116000
3	-159	-24	0.0801	0801	-1.9224	-19224
4	-175	-26	0.0806	0806	-2.0956	-20956
5	-188	-24	0.0813	0813	-1.9512	-19512
6	-203	-27	0.0823	0823	-2.2221	-22221
7	-218	-28	0.0836	0836	-2.3408	-23408
8	-233	-29	0.0852	0852	-2.4708	-24708

Assim como explicado anteriormente para a Divisão em Quadros e Janelamento, devido à utilização de ponto-fixo, o resultado da simulação deve ser lido da seguinte maneira: os quatro últimos algarismos referem-se à parte fracionária do resultado e os demais à parte inteira. Também é possível observar que o aumento do número de casas decimais consideradas para a janela de *Hamming*, proporcionou um aumento na precisão do resultado final do janelamento.

O código escrito em VHDL do módulo de pré-processamento completo totalizou 935 linhas e o ambiente de simulação 407 linhas.

5.8 Protótipo

Após a etapa de cosimulação pós-síntese do sistema completo, deu-se início a etapa de prototipação, para a qual foi utilizada a placa Stratix II EP2S60F672C5ES da Altera [Altera 2006]. Detalhes sobre a placa Stratix II EP2S60F672C5ES são apresentados no Apêndice B.

Considerando o módulo de pré-processamento como um único bloco, na Figura 5.12 é apresentado o diagrama do protótipo desenvolvido, que foi chamado de *Top_level*. O sistema inclui ainda, os blocos *Divisor_Clock*, *Gerador_Clock* e *FLASH*.

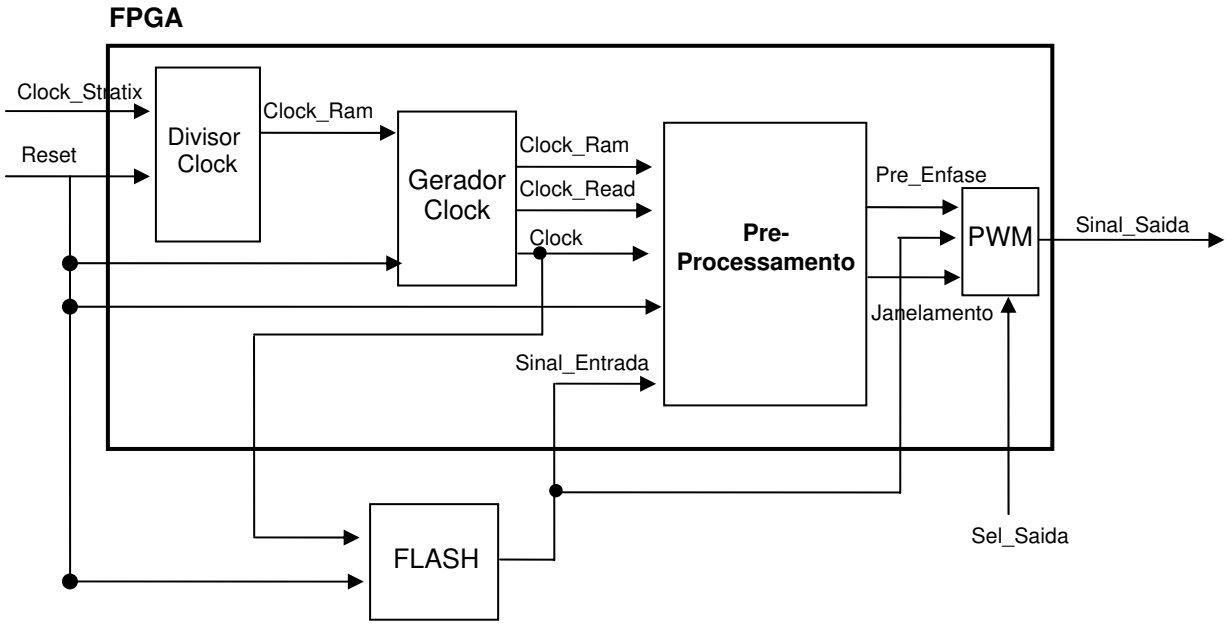


Figura 5.12 – Diagrama do protótipo desenvolvido.

O bloco *Divisor_clock* consiste em um divisor de freqüência que recebe o *clock* fornecido pela Stratix II (50 MHz) e gera *Clock_ram* que é a maior freqüência utilizada pelo sistema. O módulo *Gerador_Clock* recebe *Clock_ram* e com base neste gera os outros dois *clocks* necessários para funcionamento do sistema. Sendo assim, os três *clocks* utilizados são:

- *Clock* – freqüência com que as amostras são lidas da memória Flash e repassadas para o sistema. Essa freqüência refere-se a $f_{ck1,}$ apresentada na Figura 4.11, que corresponde a 11.025Hz;

- *Clock_read* – freqüência de leitura da ROM, três vezes mais rápida que C*lock.* Refere-se a f_{ck2} também apresentada na Figura 4.11;

- *Clock_ram* – O dispositivo utilizado para prototipação suporta apenas memórias RAM e ROM síncronas, o que implica, que são necessários dois ciclos de *clock* entre o fornecimento do endereço e o recebimento do dado. Como o sistema foi projetado considerando as memórias assíncronas, foi gerado o terceiro *clock*, duas vezes mais rápido que *Clock_read*, para permitir a continuidade do correto funcionamento do sistema. No primeiro ciclo mais rápido é fornecido o endereço e no segundo o dado é recebido.

As amostras do sinal de voz, previamente capturadas, foram armazenadas na memória *flash* da placa Stratix. O módulo *FLASH* tem a função de fazer a leitura dessas amostras e fornecê-las para o sistema de pré-processamento. É importante observar que a leitura da Flash é realizada com uma freqüência de 11.025Hz, que corresponde à taxa de amostragem utilizada na captura e posterior reprodução do sinal de voz.

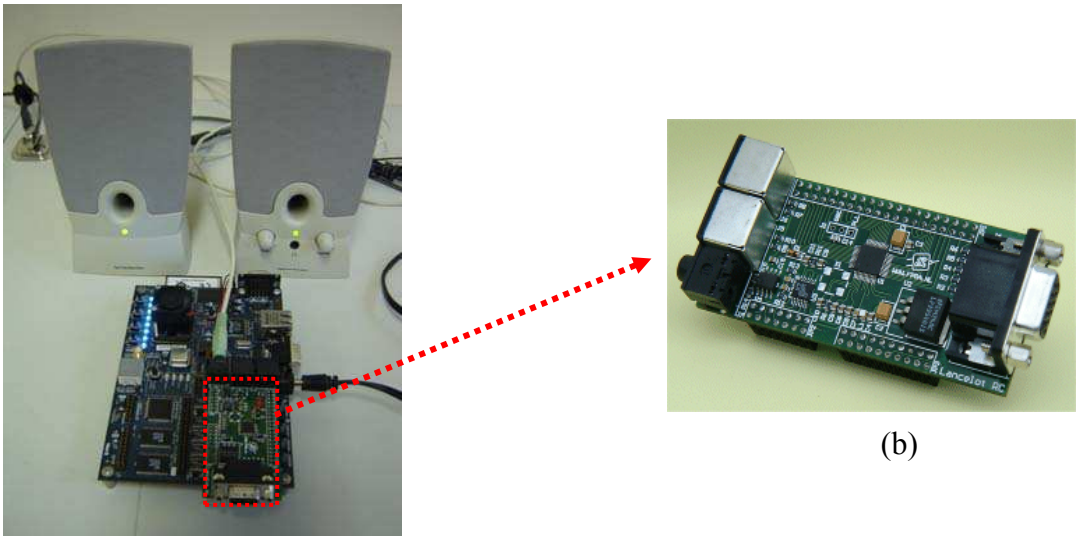
Uma vez que, o sinal de entrada do sistema é de 16 bits, as amostras do sinal de voz foram gravados na memória Flash com palavras de também 16 bits. Foi escrito na linguagem C, um programa (Figura 5.13), que com base nos valores das amostras do sinal de voz, gravadas em um arquivo *.txt*, gera um arquivo binário com cada amostra ocupando 2 bytes (16 bits). O arquivo binário gerado é utilizado no processo de gravação da memória Flash.

```
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[]) {

  FILE *inf, *outf;

  if ( (inf = fopen(argv[1], "r"))==NULL ) {
    printf("Erro no fopen read\n");
    exit(1);
  }
  if ( (outf = fopen(argv[2], "wb"))==NULL ) {
    printf("Erro no fopen write\n");
    exit(1);
  }

  long i;
  int amostra;
  int a;

  while (!feof(inf)) {
      fscanf(inf,"%d", &amostra);
      fwrite(&amostra,2,1,outf);
  }

  fclose(inf);
  fclose(outf);

  return 0;

};
```

Figura 5.13 – Programa *convert.c*.

A saída da pré-ênfase foi conectada a duas caixas de som (Figura 5.14), o que permite verificar seu efeito sobre o sinal original. A placa Stratix II que foi utilizada para prototipação não possui saída de áudio. Diante disso, para possibilitar a conexão das caixas de som à placa,

foi conectada a placa Lancelot [FPGA 2006] ao conector PROTO1 da Stratix II. Também foi necessária a implementação de um conversor D/A baseado em PWM (*Pulse Width Modulation*) [Kwasinski 2003], que efetua a conversão digital-analógica do sinal pré-enfatizado.



(a) (b)

Figura 5.14 – (a) Placa Stratix II conectada às caixas de som através da (b) placa Lancelot.

A saída do janelamento (32 bits) foi conectada ao analisador lógico, a partir do qual as formas de onda do sinal janelado podem ser visualizadas, como mostra a Figura 5.15.

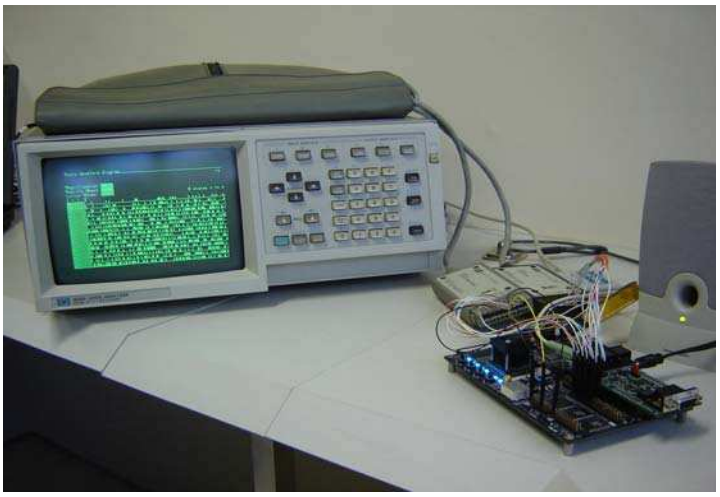


Figura 5.15 – Resultado da divisão em quadros e Janelamento conectado ao analisador lógico.

Após a etapa de síntese, a partir da ferramenta Quartus 5.1, foi obtida a quantidade de células lógicas e segmentos necessários à implementação do IP Core. Esses valores são apresentados na Tabela 5.5. Os elementos de DSP (*Digital Signal Processing*) responsáveis pelas operações de multiplicação e divisão, foram necessários em virtude das operações de multiplicação. Esse tipo de operação é realizado apenas no módulo de Divisão em Quadros e Janelamento. No módulo de Pré-ênfase o uso de DSP não se tornou necessário, uma vez que a operação de divisão foi realizada a partir de deslocamento de bits.

Com base nas informações disponíveis no Apêndice B (Tabela B.2), na Tabela 5.5 é apresentada a quantidade de recursos utilizados pelo sistema desenvolvido. Essa informação permite observar que esse sistema poderia ser prototipado em uma placa com menos recursos do que a que foi utilizada. Para permitir uma melhor visualização (ou comprovação) da utilização de poucos recursos, na Tabela 5.5 também são apresentados os recursos disponíveis no Cyclone II EP2C70 [CycloneII 2006]. Este é um dispositivo de baixo custo da Altera, sendo um dos menores e mais baratos que permitem processamento digital de sinais.

Tabela 5.5 – Quantidade de recursos para implementação do IP Core.

Recurso	Quantidade Utilizada	Disponível em Cyclone II EP2C70
Pinos de entrada	43	622[3]
Pinos de saída	51	
Pinos bidirecionais	24	
Células lógicas	1.600	88.418
Bits de Memória	808.896	1.152.000
Elementos de DSP	2	150

Na Figura 5.16, é apresentado o gráfico com o tempo despendido em cada etapa da construção do protótipo. Conforme previsto na Seção 3.3.3 (Capítulo 3), a verificação funcional do sistema, que inclui a verificação funcional de cada bloco (47%), a cosimulação do sistema completo (15%) e a cosimulação pós-síntese (15%), consumiu cerca de 77% do tempo total de desenvolvimento.

[3] Total disponível no dispositivo para ser utilizado como Entrada, Saída ou Bidirecional.

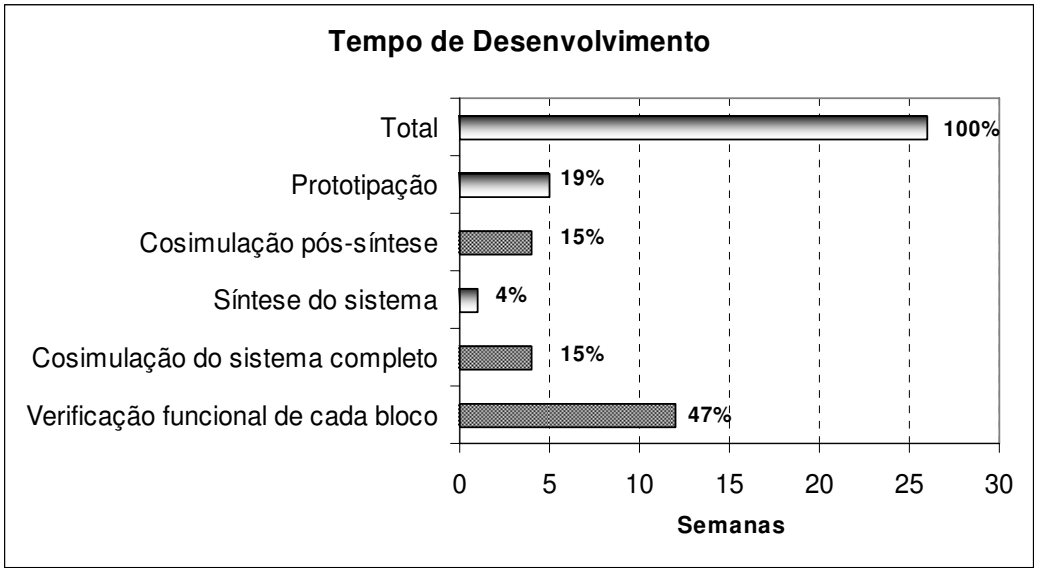


Figura 5.16 – Tempo das etapas de desenvolvimento.

5.9 Considerações Gerais

O objetivo deste trabalho foi o desenvolvimento de uma biblioteca de *hardware* para pré-processamento digital de sinais de voz, incluindo os módulos de pré-ênfase, divisão em quadros e janelamento, para aplicação em sistemas embutidos.

Para atingir tal objetivo, foram realizadas adaptações dos modelos necessários, de modo a permitir a sua implementação em um dispositivo de *hardware* com poucos recursos computacionais.

Neste capítulo, foram apresentados os resultados obtidos em cada etapa do processo de desenvolvimento da biblioteca, os quais possibilitaram a verificação da eficiência do IP Core desenvolvido.

Capítulo 6

Considerações Finais e

Sugestões para Trabalhos Futuros

6.1 Apresentação

Por ser a voz a forma mais natural de comunicação humana, o interesse pela interação homem-máquina através da fala, torna-se cada vez maior, dando origem a uma demanda muito grande por sistemas capazes de reconhecer o que está sendo dito, bem como quem está falando ou responder ao que está sendo solicitado [Rabiner 1978, Doddington 1985, Vieira 1989, Fagundes 1993, Rabiner 1993, Vassali 2000, Minker 2004]. Devido à existência de várias questões que ainda não foram solucionadas, muitas pesquisas ainda vêm sendo realizadas na área de Processamento Digital de Sinais de Voz [Lin 2003, Krishna 2004, Ibnkahla 2005, Nedevschi 2005].

Algoritmos com bom desempenho, em termos de taxa de reconhecimento, demandam um certo poder computacional, o que já se tem disponível em máquinas tipo PC (*Personal Computer*) em tempo real. No entanto, em sistemas embutidos, como um telefone celular, o desafio não representa apenas o desenvolvimento de *software*, mas também de *hardware* que se adapte às características inerentes de tais dispositivos (baixo consumo de energia, baixo

poder de processamento e armazenamento, dimensões reduzidas, portabilidade etc.) [Lin 2003, Krishna 2003, Krishna 2004, Ibnkahla 2005, Nedevschi 2005].

Para permitir a implementação de sistemas de PDSV nesse contexto, alguns trabalhos sacrificam a eficiência no processo de reconhecimento em nome da redução da área e exigências computacionais. Contudo, algumas técnicas de programação permitem a geração de um código mais eficiente no que se refere à quantidade de operações a serem executadas pelo processador (p.ex.: substituição de uma multiplicação por deslocamento de bits), o que conseqüentemente proporciona menor consumo energia [Hon 1991].

Dentro deste contexto, este capítulo apresenta as conclusões do trabalho desenvolvido, destacando as suas contribuições e indicando sugestões para trabalhos futuros.

6.2 Contribuições da Pesquisa

O trabalho desenvolvido consiste de uma biblioteca de *hardware* para pré-processamento digital de sinais de voz, incluindo as funções de pré-ênfase, divisão em quadros e janelamento, adaptada para aplicação em sistemas embutidos com poucos recursos computacionais, como capacidade de processamento e armazenamento.

A validação da biblioteca desenvolvida através de uma metodologia de verificação funcional e sua prototipação em um dispositivo de *hardware*, comprovam a eficiência das adaptações efetuadas, a destacar:

- Simplificações das funções matemáticas;
- Utilização de ponto-fixo nas operações com números fracionários;
- Substituição de algumas divisões e multiplicações por deslocamentos;
- Realização de operações em paralelo;
- Definição de um algoritmo para leitura e escrita da memória, de modo a viabilizar o uso de uma memória de menor capacidade;
- Cálculo *off-line* dos valores obtidos com a técnica de janelamento, possibilitando o uso de uma memória apenas de leitura para a manipulação desses valores.

A reutilização de uma biblioteca contendo as funções de pré-processamento já implementadas otimizará a etapa de projeto e implantação de aplicações embutidas de

reconhecimento de fala ou locutor, reduzindo consideravelmente o custo final do produto. Uma vez que a biblioteca foi projetada de modo a utilizar, de maneira eficiente, o mínimo de recursos possível, esta poderá ser utilizada em dispositivos com poucos recursos computacionais, como equipamentos eletrônicos, celulares ou mesmo máquinas em geral, como elevadores e portas automáticas.

Como o código desenvolvido é um módulo sintetizável, este poderá ser utilizado ainda, na fabricação de um IP *Core* de pré-processamento de voz.

Além da adaptação das funções de pré-processamento para implementação em um dispositivo de *hardware* com poucos recursos computacionais, também foi desenvolvido um conversor D/A baseado em PWM, que possibilitou a audição do sinal de entrada (original) e do correspondente após cada etapa do pré-proceesamento (pré-ênfase e janelamento), de forma a avaliar melhor a eficiência do sistema desenvolvido.

6.3 Sugestões para trabalhos futuros

Conforme descrito anteriormente, em uma aplicação de comunicação vocal homem-máquina, o objetivo da etapa de pré-processamento é, fundamentalmente, tratar o sinal de voz emitido pelo homem, de modo a reduzir efeitos indesejados incorporados ou presentes no sinal, além de prepará-lo para as etapas seguintes do processo de reconhecimento.

As principais operações realizadas nessa etapa são as de pré-ênfase, divisão em quadros e janelamento. No entanto, outros tratamentos (*e.g.*, aquisição e digitalização do sinal, tratamento do ruído ambiente e detecção de início e fim de elocuções) podem ser efetuados no sinal antes que este seja repassado para as etapas efetivas de reconhecimento, os quais poderiam ser incluídos em uma versão futura da biblioteca.

Vale destacar também, que apesar da função de janelamento oferecida pela biblioteca possibilitar o uso de diferentes tipos de janela (*Hamming, Hanning*, Retangular, Triangular etc), o tamanho desta janela deve ser fixo e igual a 252. Essa característica pode tornar a biblioteca inadequada para aplicações nas quais se deseje utilizar janelas de diferentes tamanhos.

Diante do exposto, como sugestões para trabalhos futuros, pode-se citar:

1. Desenvolvimento de um módulo de aquisição e digitalização do sinal de voz capturado através de um microfone. Isso facilitaria a implantação de aplicações em equipamentos que não possuem esta funcionalidade.

2. Tratamento ou eliminação do ruído ambiente no momento da captura do sinal. Diferentes níveis de ruído entre as etapas de treinamento e teste de um sistema de reconhecimento podem comprometer seriamente a eficiência desse processo. Já existem algoritmos que visam resolver essa situação, no entanto, projetados, basicamente, para arquitetura PC.

3. Detecção de início e fim de palavras, eliminando o silêncio inicial e final das elocuções. Isso permite reduzir a quantidade de informação desnecessária, diminuindo o volume de dados a ser processado, podendo tornar o processo de reconhecimento mais eficiente.

4. Biblioteca com possibilidade de utilização de janelas de diferentes tamanhos.

As sugestões para trabalhos futuros supracitadas visam o melhoramento deste trabalho. Porém, vale destacar que o mesmo atende ao seu objetivo principal, o desenvolvimento de um IP Core de pré-processamento digital de sinais de voz para aplicação em sistemas embutidos, incluindo os módulos de pré-ênfase, divisão em quadros e janelamento. A biblioteca desenvolvida foi projetada visando à utilização da menor quantidade de recursos computacionais possível, de modo a permitir sua reutilização em futuras aplicações de reconhecimento de fala e/ou locutor em dispositivos com baixo poder de processamento e armazenamento.

Referências Bibliográficas

[Aguiar 2000] AGUIAR NETO, B. G. *Transmissão Digital da Informação.* Universidade Federal de Campina Grande, 2000.

[Altera 2006] Altera Corporation, *http://www.altera.com*

[Andreao 2001] ANDREAO, R. V. *Implementação em Tempo Real de um Sistema de Reconhecimento de Dígitos Conectados.* Dissertação de Mestrado – Universidade Estadual de Campinas, Campinas, 2001.

[Atal 2006] ATAL, B. S. *The History of Linear Prediction.* IEEE Signal Processing Magazine, Março 2006.

[Barr 1999] BARR, M. *Programming Embedded Systems in C and C++. Sebastopol*. CA, O'Reilly & Associates.

[Beizer 1995] BEIZER, B. *The Pentium bug, an industry watershed*. Testing Techniques Newsletter, On-Line Edition, Setembro, 1995.

[Benzeghiba 2002] BENZEGHIBA, M. F. and BOULARD, H., *User-customized password speaker verification based on HMM/ANN and GMM models*. Proceedings of the Seventh International Conference on Spoken Language Processing (ICSLP 2002), p. 1325-1328, 2002.

[Benzeghiba 2003] BENZEGHIBA, M. F. and BOULARD, H., *On the Combination of Speech and Speaker Recognition.* IDIAP-RR 19, IDIAP – Dalle Molle Institute for Perceptual Artificial Intelligence, ftp://ftp.idiap.ch/pub/reports/2003/rr03-19.pdf, 2003.

[Bergeron 2003] BERGERON, J. *Writing Testbenches: Functional Verification of HDL Model*. 2nd ed, Kluwer Academic Publishers, Boston, 2003.

[Bhasker 2002] BHASKER, J. *A SystemC Primer.* Allentown: Star Galaxy Publishing, 2002.

[Bloch 2004] BLOCH, S.C. *Ferramentas de Análise: Análise de Fourier*. Em Excel para Engenheiros e Cientistas. Rio de Janeiro, LTC, 2004.

[Campbell 1997] CAMPBELL, J. P., *Speaker Recognition: A Tutorial*. Proceeedings of the IEEE, v. 85, 1997.

[Carro 2003] CARRO, L.; WAGNER, F. R. *Sistemas Computacionais Embarcados*. In: JAI'03 – XXII Jornadas de Atualização em Informática, 2003, Campinas.

[Chou 2003] CHOU, W., JUANG, B. H. *Pattern Recognition in Speech Language Processing*. Boca Raton: Taylor & Francis Books, 2003.

[Cipriano 2001] CIPRIANO, J. L. G., *Desenvolvimento de Arquitetura Para Sistemas de Reconhecimento Automático de Voz Baseados em Modelos Ocultos de Markov*. 123 f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

[Costa 1994] COSTA, W. C. de A. *Reconhecimento de Fala Utilizando Modelos de Markov Escondidos (HMM's) de Densidades Contínuas*. Universidade Federal da Paraíba – Dissertação de Mestrado, Junho 1994.

[Cyclone 2006] *Cyclone II Device Handbook, Volume 1*. Copyright© 2006 Altera Corporation. Disponível em: http://www.altera.com

[Da Cunha 2003] DA CUNHA, A. M.; VELHO L. *Métodos Probabilísticos para Reconhecimento de Voz*. Laboratório VISGRAF – Instituto de Matemática Pura e Aplicada, 2003, 62p. Relatório Técnico.

[Da Silva 2004] DA SILVA, K. R. G.; MELCHER, E. U. K.; ARAUJO, G. *An Automaic Testbench Generation Tool for a SystemC Functional Verification Methodology*. In: SBCCI2004 – 17th Symposium on Integrated Circuits and System Design, Porto de Galinhas – PE, 2004.

[Da Silva 2004B] DA SILVA, K. R. G.; MELCHER, E. U. K. *Construção de uma Ferramenta para Verificação Funcional em SystemC (VerificaSystemC)*. Relatório de Projeto - Programa de Pós-Graduação em Engenharia Elétrica, UFCG, 2004.

[Dalpasso 1999] DALPASSO, M.; BOGLIOLO, A.; BENINI. L. *Virtual Simulation of Distributed IP-Based Designs*. DAC'99 – Design Automation Conference, New Orleans, EUA, Jun. 1999. Proceedings, ACM Press, 1999.

[Davis 1952] DAVIS *et al*, K. H. *Automatic Recognition of Spoken Digitis*. Journal of the Acoustical Society of America, v. 24, n. 6, p. 637-642, 1952.

[Davis II 2001] DAVIS II, J. *Overview of the Ptolemy Project*. Technical Memorandum UCB/ERL M01/11, University of California at Berkeley, Department of Electrical Engineering and Computer Science, Mar. 2001. Disponível em:

http://ptolemy.eecs.berkeley.edu

[De Lima 2000] DE LIMA, A. A.; FRANCISCO, M. S.; NETTO, S.L.; F. RESENDE JR., G. V. *Análise Comparativa de Parâmetros em Sistemas de Reconhecimento de Voz*. In: XVIII SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2000.

[De Micheli 2002] DE MICHELI, G. e BENINI, L. *Networks-on-Chip: A New Paradigm for Systems-on-Chip Design*. DATE'02 – Design, Automation and Test in Europe, Paris, França, Mar. 2002. Proceedings, IEEE Computer Society Press, 2002.

[Deller 1993] DELLER Jr. R., PROAKIS, J. G., and HANSEN, J. H. L., *Discrete-time Processing of Speech Signals*, Macmillan Publishing Co., 1993.

[Dhananjai 2000] DHANANJAI, R.; CHERNYAKHOVSKY, V.; WILSEY, P.A. *WESE: A Web-based Environment for Systems Engineering*. WEBSIM'2000 - International Conference on Web-Based Modelling & Simulation. Jan. 2000. Proceedings, SCS, 2000.

[Dias 2000] DIAS, R. de S. F. *Normalização de Locutor em Sistemas de Reconhecimento de Fala.* Dissertação de Mestrado – Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, 2000.

[Diniz 1997] DINIZ, S. S. *Uso de Técnicas Neurais para o Reconhecimento de Comandos à Voz.* Dissertação de Mestrado, Instituto Militar de Engenharia, Rio de Janeiro, 1997.

[Doddington 1985] DODDINGTON, G. R. *Speaker Recognition – Identifying People by their Voices*. Proceedings of the IEEE, v. 73, n. 11, p. 1651-1664, nov. 1985.

[Edwards 1997] EDWARDS, S.; LAVAGNO, L.; LEE, E.A.; SANGIOVANNI-VINCENTELLI, A. *Design of Embedded Systems: Formal Models, Validation, and Synthesis*. Proceedings of the IEEE, Vol. 85, No. 3, Mar. 1997. pp 366-390

[Fagundes 1993] FAGUNDES, R. D. R.; ALENS, N. *Reconhecimento de Voz, Linguagem Contínua, Usando Modelos de Markov*. In: 11º SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 1993.

[Fechine 2000] FECHINE, J. M *Reconhecimento automático de identidade vocal utilizando modelagem híbrida: Paramétrica e*

Estatística. 2000. 212 f. Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Campina Grande, 2000.

[Fin 2000] FIN, A.; FUMMI, F. *A Web-CAD Methodology for IP-Core Analysis and Simulation*. DAC'2000 – Design Automation Conference, Los Angeles, EUA, Jun. 2000. Proceedings, ACM Press, 2000.

[Fine 2004] FINE, S., UR, S., ZIV, A. *Probabilistic Regression Suites for Functional Verification.* In Proceeding of the Design Automation Conference (DAC), 2004.

[Flanagan 1978] FLANAGAN, L. J. *Speech Analysis Synthesis and Perception.* Murray Hill Second Edition, New Jersey, 1978.

[FPGA 2006] *https://www.fpga.nl*

[Francia 2001] FRANCIA, G. A. *Embedded Systems Programming.* The Journal of Computing in Small Colleges, v. 17, n. 2, p. 204-210, dez. 2001. Proceedings of the 15th Annual CCSC Southeastern Conference.

[Furui 1981] FURUI, S. *Cepstral Analysis Technique for Automatic Speaker Verification*. IEEE Transactions on Acoustics, Speech and Signal Processing Magazine, v. 29, 1981.

[Haykin 1994] HAYKIN, S., *Neural Networks – A Compreehensive Foundation*, IEEE Press, Englewood Cliffs, 1994.

[Hessel 1999] HESSEL, F. *MCI: Multilanguage Distributed Cosimulation Tool*. In: F.J.Rammig (ed.), Distributed and Parallel Embedded Systems. Kluwer Academic Publishers, 1999.

[HLA 2000] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - IEEE Standard No. 1516-2000*. IEEE, 2000.

[Hon 1991] HON, Hsiao-Wuen. *A Survey of Hardware Architectures Designed for Speech Recognition.* Technical Report CMU-CS-91-169, 1991.

[Huang 2001] HUANG, X., ACERO, A. and HON, H-W., *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, Prentice Hall PTR, 2001.

[Ibnkahla 2005] IBNKAHLA, M. *Signal Processing for Mobile Communication Handbook.* Florida: CRC Press, 2005.

[Kleijn 1998] KLEIJN, W. B. and PALIWAL, K. K., *Speech Coding and*

Synthesis, Elsevier Science B. V., 1998.

[Koenig 1946] KOENIG, W. *The Sound Spectrograph*. Journal of the Acoustical Society of America, v. 17, p. 19-49, 1946.

[Krishna 2003] KRISHNA, R., MAHLKE, S., AUSTIN, T., *Architectural Optimizations for Low-Power, Real-Time Speech Recognition.* CASES'03, San Jose, California, USA, 2003.

[Krishna 2004] KRISHNA, R., MAHLKE, S., AUSTIN, T., *Memory system design space exploration for low-power, real-time speech recognition.* Proceedings of the 2nd IEEE/ACM/IFIP international conference on *Hardware/software* codesign and system synthesis, p. 140 – 145, 2004.

[Kwasinski 2003] KWASINSKI, A., KREIN, P. T., CHAPMAN, P. L. *Time Domain Comparison of Pulse-Width Modulation Schemes*. IEEE Power Electronics Letters, Vol. 1, No. 3, Setembro, 2003.

[Lathi 1983] LATHI, B. P. *Sistemas de Comunicação*. Rio de Janeiro: Guanabara Dois, 1983. 401 p.

[Lathi 1998] LATHI, B. P. *Modern Digital and Analog Communication Systems*. New York: Oxford University Press, 1998.

[Lee 2003] LEE, C., HYUN, D., CHOI, E., GO, J., LEE, C., *Optimizing Feature Extraction for Speech Recognition.* IEEE Transactions on Speech and Audio Processing, vol. 11, no. 1, Janeiro 2003.

[Lee 2005] LEE, T. *Speech Analysis.* Department of Electronic Engineering, The Chinese University of Hong Kong, 2005.

[Lemmetty 2004] LEMMETTY, S. *Review of Speech Synthesis Technology*. Disponível em www.acoustics.hut.fi/~slemmett/dippa/chap1.html. Acessado em dezembro de 2004.

[Li 1999] LI, Q.; TSAI, A. *A Matched Filter Approach to Endpoint Detection for Robust Speaker Verification.* In Proceedings of IEEE Workshop on Automatic Identification Advanced Technologies (AutoID'99), p. 35-38, Outubro 1999.

[Li 2003] LI, X. *High-Speed, Low-Resource Automatic Speech Recognition System. Masterthesis* – University of Washington, 2003.

[Lin 2003] LIN, E. *A First Generation Hardware Reference Model for a Speech Recognition Engine. Masterthesis* – School of

Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, 2003.

[Madeiro 1999] MADEIRO, F., VILAR, R. M., FECHINE, J. M., and AGUIAR NETO, B. G., *A Self Organizing Algorithm for Vector Quantizer Design Applied to Signal Processing*, International Journal of Neural Systems, v. 9, 1999.

[Mammone 1996] MAMMONE, R. J., ZHANG, X., and RAMACHANDRAN, R. P., *Robust Speaker Recognition* – A Feature-Based Approach. IEEE Signal Processing Magazine, v. 13, 1996.

[Martins 1997] MARTINS, J. A. *Avaliação de diferentes técnicas para reconhecimento de fala.* Tese de Doutorado, Universidade Estadual de Campinas, Campinas, 1997.

[Martins 2003] MARTINS, C. A. P. S., ORDONEZ, E. D. M., CORRÊA, J. B. T., CARVALHO, M. B. *Computação Reconfigurável: conceitos, tendências e aplicações.* In: Ciência, Tecnologia e Inovação: Atalhos para o Futuro: anais. Campinas, 2003, v. 2, p. 339-388.

[Mello 2002] MELLO, B.A.; WAGNER, F.R. *A Distributed Co-Simulation Backbone*. In: M.Robert et al. (eds), SOC Design Methodologies. Kluwer Academic Publishers, 2002.

[Mentor 2003] *Seamless CVE.* Disponível em: http://www.mentor.com/seamless

[Minker 2004] MINKER, W., BENNACEF, S. *Speech and Human-machine Dialog.* Boston: Kluwer Academic Publishers, 2004.

[Moraes 2001] MORAES, F., MESQUITA, D. *Tendências em Reconfiguração dinâmica de FPGAs.* SCR'2001 - Seminário de Computação Reconfigurável. Belo Horizonte, MG, Setembro, 2001.

[Moraes 2004] MORAES, F., CALAZANS, N. MOLLER, L. BRIAO, E., CARVALHO, E. *Dynamic and Partial Reconfiguration in FPGA SoCs:* Requirements Tools and a Case Study. In: ROSENSTIEL, Wolfgang. Reconfigurable Computing. New York, USA. 2004.

[Nedevschi 2005] NEDEVSCHI, S., PATRA, R. K., BREWER, E. A. *Hardware Speech Recognition for User Interfaces in Low Cost, Low Power Devices.* Proceedings of the 42nd annual conference on Design automation, p. 684 – 689, San Diego, California, USA. 2005.

[OCP 2001] *Open Core Protocol Reference* - OCP-IP Association.

[Oyamada 2000] OYAMADA, M.; WAGNER, F.R. *Co-simulation of Embedded Electronic Systems*. 12nd European Simulation Symposium. Hamburgo, Alemanha, Out. 2000. Proceedings, SCS, 2000.

[Paula 2000] PAULA FILHO, W. P., *Multimídia: Conceitos e Aplicações*, JC Editora, 2000.

[Petry 2000] PETRY, A., ZANUZ, A. e BARONE, D. A. C., *Reconhecimento Automático de Pessoas Pela Voz Através de Técnicas de Processamento Digital de Sinais.* SEMAC 2000, 2000.

[Picone 1993] PICONE, J. W., *Signal Modeling Techniques in Speech Recognition.* Proceedings of the IEEE, v. 8, n09, September, p. 1215-1247, 1993.

[Rabiner 1978] RABINER, L. R.; SCHAFER, R. W. *Digital processing of speech signals.* New Jersey: Prentice Hall, 1978.

[Rabiner 1989] RABINER, L. R., *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, v. 77, 1989.

[Rabiner 1993] RABINER, L.; JUANG, B. - H. *Fundamentals of Speech Recognition*. New Jersey: Prentice Hall, 1993. 507p.

[Romero 2005] ROMERO, E. L., STRUM, M., CHAU, W. J. *Comparing Two Testbench Methods for Hierarchical Functional Verification of a Bluetooth Baseband Adaptor.* CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA, 2005.

[Sanderson 2004] SANDERSON, C., PALIWAL, K. K. *Identity Verification Using Speech and Face Information.* Digital Signal Processing, Vol. 14, No. 5, 2004.

[Shandle 2002] SHANDLE, J.; MARTIN, G. *Making Embedded Software Reusable for SoCs*. EEDesign, Março 2002.

[Shaughnessy 1995] SHAUGHNESSY, D. O., VERGIN, R. *Pre-emphasis and Speech Recognition.* Canadian Conference on Electrical and Computer Engineering, CCECE/CCGEI, 1995.

[Shaughnessy 2000] SHAUGHNESSY, D. O. *Speech Communications, Human and machine*, New York: IEEE Press, 2000.

[Silburt 1995] SILBURT, A., PERRYMAN, I., BERGERON, J., NICHOLS, S., DUFRESNE, M., WARD, G. *Accelerating Concurrent Hardware Design with Behavioural Modelling and System Simulation*. In Proceeding of the Design Automation

Conference (DAC), 1995.

[StratixII 2005] *Nios Development Board – Reference Manual, Stratix II Edition.* Copyright© 2005 Altera Corporation. Disponível em: http://www.altera.com

[Synopsys 2006] *CoCentric System Studio*. Disponível em: http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html

[SystemC 2006] Open SystemC initiative, *http://www.systemc.org*, Acesso em março de 2006.

[Tatham 2005] TATHAM, M., MORTON, K. *Developments in Speech Synthesis.* West Sussex: John Wiley & Sons Ltd, 2005.

[Valderrama 1998] VALDERRAMA, C. *Automatic VHDL-C Interface Generation for Distributed Cosimulation: Application to Large Design Examples*. Journal on Design Automation for Embedded Systems, Vol. 3, No. 2/3, Mar. 1998.

[Vassali 2000] VASSALI, M. R.; DE SEIXAS, J. M.; ESPAIN, C. *Reconhecimento de Voz em Tempo Real Baseado na Tecnologia dos Processadores Digitais de Sinais*. In: XVIII Simpósio Brasileiro de Telecomunicações, 2000.

[Vieira 1989] VIEIRA, M. N. *Módulo frontal para um sistema de reconhecimento automático de voz.* 1989 Dissertação – Universidade de Campinas, Campinas, 1989.

[Wolf 2001] WOLF, W. *Computers as Components*. McGraw-Hill, 2001.

[Zheng 2004] ZHENG, N., CHING, P. C. *Using Haar Transformed Vocal Source Information for Automatic Speaker Recognition.* Acoustics, Speech, and Signal Processing, Proceedings. (ICASSP '04). IEEE International Conference, 2004.

Apêndice A. SystemC

SystemC é uma linguagem de descrição tanto de *hardware* (HDL – *Hardware Description Language*) quanto de *software*. Esta característica é obtida através da extensão da linguagem C++, adicionando a esta uma biblioteca de funções, escrita na própria linguagem, que provê mecanismos para modelagem da arquitetura do sistema com elementos de *hardware*, além dos conceitos de concorrência, eventos e tipos de dados [Bhasker 2002].

A primeira versão da linguagem, SystemC 0.9, foi disponibilizada em setembro de 1999, pelo *Open SystemC Initiative* (OSCI), gratuitamente e com código aberto. Em março de 2000 foi liberada a versão 1.0, no entanto esta era limitada à modelagem comportamental da transferência entre registros. Finalmente, em outubro de 2001 foi lançada a versão SystemC 2.0, que introduziu construções de comunicação e sincronização de mais alto nível, como canais, interfaces e eventos, permitindo a modelagem de mecanismos mais abstratos de comunicação. O objetivo do OSCI era transformar SystemC em um padrão IEEE [Carro 2003, Bhasker 2002], o que foi alcançado em Dezembro de 2005 [SystemC 2006] quando o IEEE definiu o padrão 1666 referente à linguagem.

Ao utilizar SystemC, é possível realizar as seguintes atividades [Bhasker 2002]:

- Projeto de sistema;
- Descrição da arquitetura de *hardware*;
- Descrição de algoritmos de *software*;
- Verificação;
- IP *core Exchange*

Como SystemC é baseada em C++, é possível utilizar ferramentas de desenvolvimento já disponíveis para criar, simular, depurar características tanto arquiteturais quanto algorítmicas do projeto. Na Figura A.1, tem-se a ilustração do uso de SystemC em um ambiente de desenvolvimento C++.

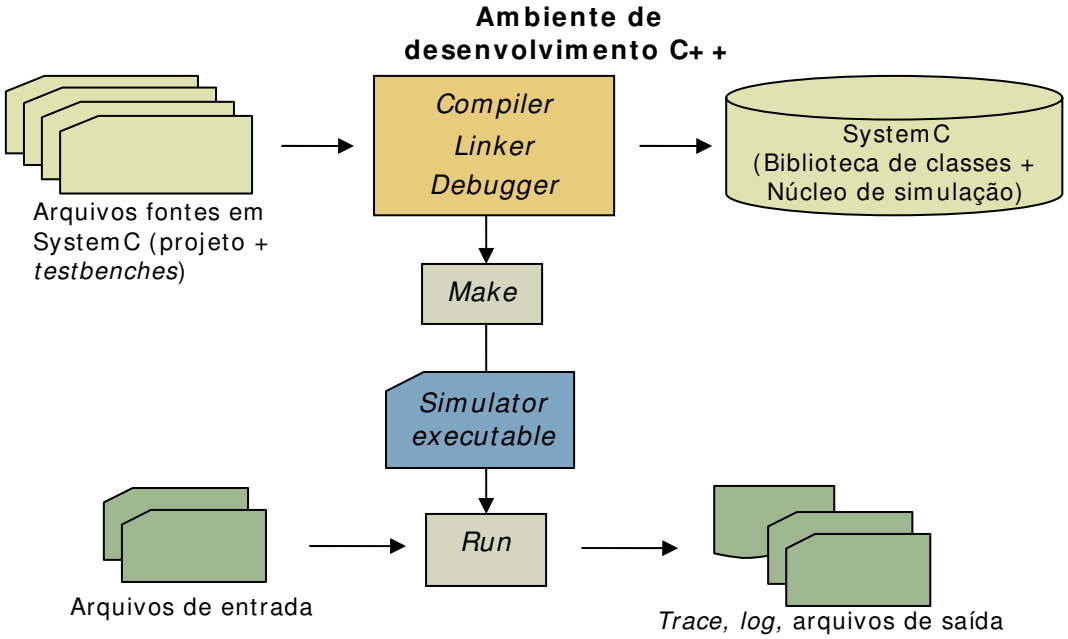


Figura A.1 – SystemC no ambiente de desenvolvimento C++.

Pode-se observar, a partir da Figura A.1, que SystemC consiste de uma biblioteca de classes mais um núcleo de simulação para modelagem de sistemas. Tal biblioteca dá suporte à concorrência, operações seqüenciais em função do tempo, tipos de dados para descrição de *hardware*, estrutura hierárquica e suporte à simulação [SystemC 2006]. Outra vantagem oferecida por SystemC reside no fato de que esta também pode ser utilizada como uma linguagem de verificação (HVL – *High-level Verification Language*), eliminando a necessidade de aprendizado de uma outra linguagem por parte dos projetistas e engenheiros de verificação [Bhasker 2002].

Methodology-Specific Libraries Master/Slave library, etc	**Layered Libraries** Verification library TLM library, etc
Primitive Channels Signal, Fifo, Mutex, Semaphore, etc	
Structural Elements Modules Ports Interfaces Channels	**Data Types** 4-valued logic Bits and Bit Vectors Arbitrary Precision Integers Fixed-point types
Event-driven Simulation Events Processes	
C++ Language Standard	

Figura A.2 – Arquitetura da linguagem SystemC [SystemC 2006].

Na Figura A.2, tem-se a ilustração da estrutura da biblioteca SystemC. A parte preenchida em cinza representa o núcleo da linguagem.

As camadas acima do núcleo consistem de bibliotecas de projeto e padrões, que como não fazem parte do mesmo, o usuário pode optar por usar ou não tais recursos [SystemC 2006].

O núcleo é composto de:

- Simulador orientado a eventos, que trata também processos;
- Módulos e portas para representação da estrutura, interfaces e canais;
- Conjunto de tipos de dados que implementam várias representações de dados necessárias para modelagem de *hardware*, como também para programação de *software*;
- Canais primitivos para tratamento de sinais, FIFO, semáforos, etc.

Um sistema em SystemC consiste de um ou mais módulos, em que cada módulo pode conter processos, portas, dados internos, canais e instâncias de outros módulos. Todos os processos são conceitualmente concorrentes e podem ser utilizados para modelar a funcionalidade do módulo. Portas são objetos através das quais os módulos se comunicam [SystemC 2006].

Quanto aos eventos, esses são basicamente objetos de sincronização entre processos e implementam bloqueio de canais. Processos são disparados ou rodados baseados na ocorrência de eventos, que podem ser dinâmicos ou estáticos.

Na Figura A.3, são apresentadas as etapas do processo de desenvolvimento utilizando SystemC. Inicialmente, é escrito um modelo do sistema, que tipicamente não é sintetizável, o qual utiliza tipos abstratos de dados (classes) e de comunicação (semáforos), e é atemporal. Aqui também podem ser "explorados", ou ainda, avaliados diversos algoritmos pré-definidos. Após o primeiro refinamento, é inserida a noção de tempo, através da associação de tempo de execução de processos ou ainda da introdução do sinal de *clock*. Diante disso, diferentes arquiteturas de *hardware* e *software* podem ser exploradas, análises de desempenho realizadas e, assim, o melhor cenário é definido. Cumprida essa etapa, o próximo passo no que se refere à parte de *software* é a escolha de um sistema operacional de tempo real (RTOS – *Real Time Operating System*) e desenvolvimento do código alvo para o mesmo. Quanto ao *hardware*, ainda é necessário mais um refinamento, que visa a conversão do modelo que ainda está descrito em SystemC, para um modelo sintetizável. Para isso, é utilizado um subconjunto de SystemC, o SystemC RTL, que permite descrever modelos ao nível de transferência de

registro, chamados de modelos RTL, que são sintetizáveis e descrevem a máquina de estados finitos do projeto (FSM – *Finite State Machine*). Com base nos modelos RTL, podem ser realizadas a síntese e a implementação.

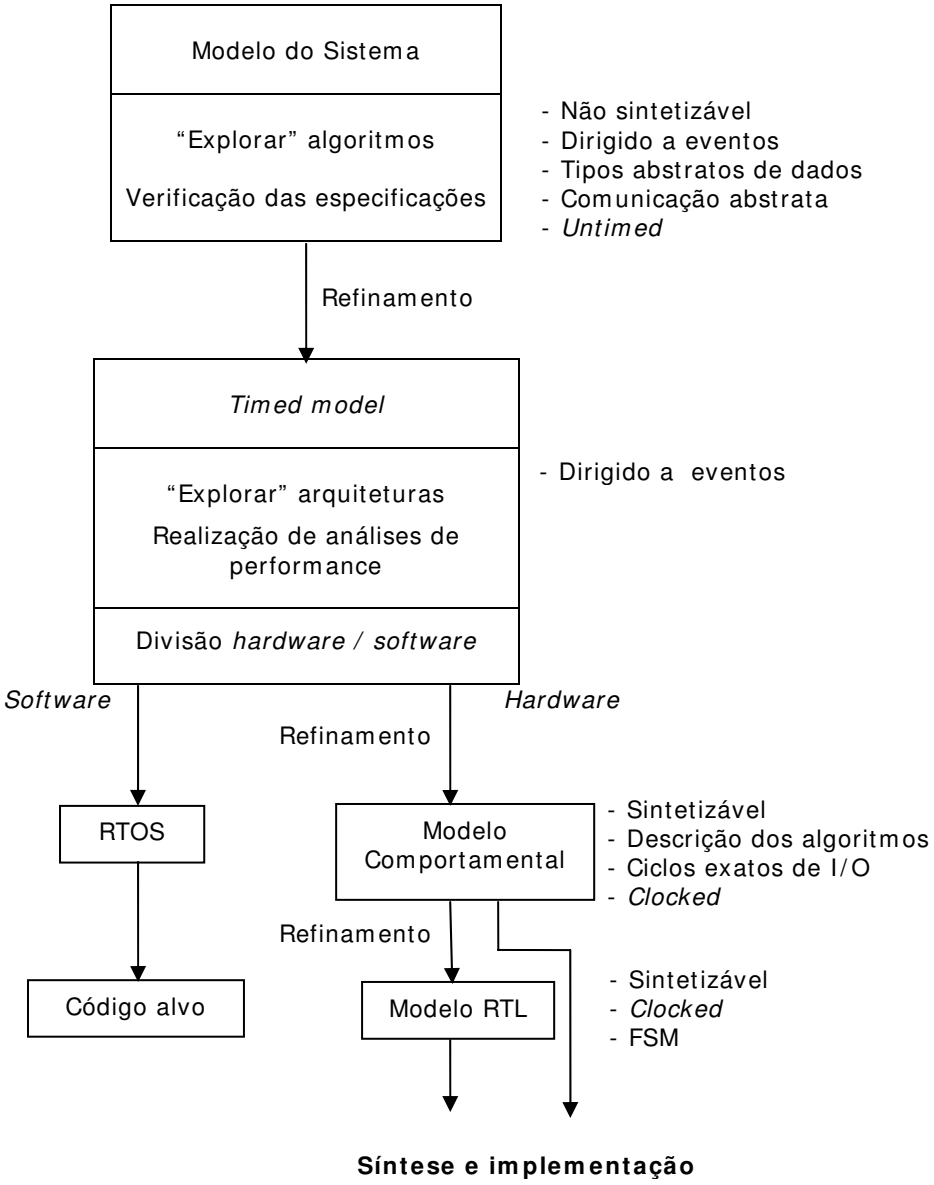


Figura A.3 – Etapas do processo de desenvolvimento [Bhasker 2002].

Diante do que foi exposto, é possível observar que SystemC é uma linguagem que permite desenvolver desde o modelo do sistema até o código executável na arquitetura especificada.

Apêndice B. Stratix II

A placa de desenvolvimento Nios Stratix II Edition oferece uma plataforma de *hardware* para desenvolvimento de sistemas embutidos baseados em dispositivos Stratix II da Altera, e oferece as seguintes características [StratixII 2005]:

- Um FPGA Stratix II EP2S60F672C5ES com mais de 13.500 módulos lógicos adaptativos (ALM - *adaptive logic modules*) e 1.3 milhões de bits de memória *on-chip*.
- 16 Mbytes de memória *flash*
- 1 Mbyte de RAM estática
- 16 Mbytes de SDRAM
- Dispositivo Ethernet MAC/PHY
- Conector CompactFlashTM para cartões CompactFlash Type I
- Conector mictor para *hardware* e *software* debug
- Duas portas serias RS-232 DB9
- Quatro botões de escolha (*push-button*) conectados aos pinos de I/O do Stratix II
- Oito LEDs conectados aos pinos de I/O do Stratix II
- Dual 7-segment LED display
- Conectores JTAG para dispositivos Altera® via Altera *download cables*
- Oscilador de 50 MHz e circuito de distribuição de *clock zero-skew*
- Botão de reset

Na Figura B.1, é apresentada a placa de desenvolvimento Nios Stratix II Edition com a identificação de seus componentes.

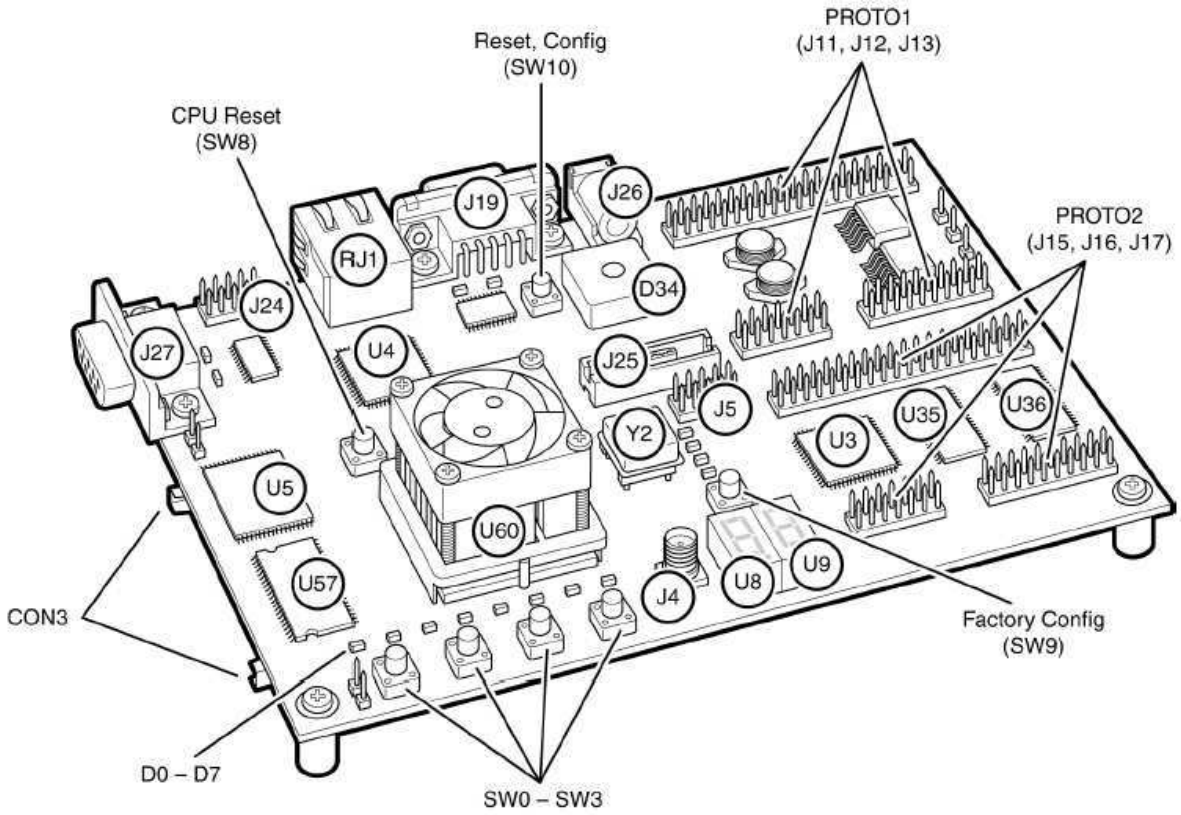


Figura B.1 – Placa de desenvolvimento Nios Stratix II Edition [StratixII 2005].

As descrições dos componentes apresentados na Figura B.1 são apresentadas na Tabela B.1.

Tabela B.1 – Componentes da placa de desenvolvimento Nios Stratix II Edition [StratixII 2005].

	Nome	**Descrição**
Dispositivo		
U60	Stratix II FPGA	EP2S60F672C5 ou EP2S30F672C5 com um *cooler* para evitar excesso de aquecimento.
User Interface		
SW0 – SW3	Botões de escolha	Quatro botões que proporcionam uma interface entre o usuário e o FPGA.
D0 – D7	LEDs individuais	Oito LEDs individuais controlados pelo FPGA.

U8, U9	LEDs de 7 segmentos	Dois LEDs de sete segmentos para exibição de valores numéricos vindos do FPGA.
Memória		
U35, U36	Memória SRAM	Dois chips SRAM combinados compondo 1 Mbyte.
U5	Memória Flash	16 Mbytes de memória não volatil para uso tanto do FPGA como para controle de configuração.
U57	Memória SDRAM	16 Mbytes de SDRAM
Conectores & Interfaces		
U4, RJ1	Ethernet MAC/PHY	Chip 10/100 Ethernet MAC/PHY chip conectado a um conector RJ-45 Ethernet.
J19, J27	Conectores Seriais	Dois conectores seriais com 5 V-tolerant buffers. Suporta todos os sinais RS-232.
PROTO1 (J11, J12, J13)	Expansion prototype connector	Expansion headers connecting to 41 I/O pins on the FPGA. Supplies 3.3V and 5.0V for use by a daughter card.
PROTO2 (J15, J16, J17)	Expansion prototype connector	Expansion headers connecting to 41 I/O pins on the FPGA. Supplies 3.3V and 5.0V for use by a daughter card.
CON3	Conector CompactFlash	Conector CompactFlashTM para cartões CompactFlash Type I.
J25	Mictor connector	Conector Mictor para depuração de sistemas Nios II utilizando pontas de prova *First Silicon Solutions* (FS2).
J24	Conector JTAG	Conecta-se ao FPGA permitindo configuração do *hardware* através do Quartus II e depuração do *software* através do Nios II IDE.
J5	Conector JTAG	Conecta-se ao controlador de configuração.
Configuração & Reset		
U3	Controlador de configuração	Dispositivo Altera EPM7128AE utilizado para configurar o FPGA a partir da memória

		flash.
SW8	Botão Reset CPU	Reinicia o processador Nios II configurado no FPGA.
SW9	Botão Configuração de Fábrica	Reconfigura o FPGA com o projeto de referência programado de fábrica.
SW10	Reset, Config	Reinicia a placa.
LED0 – LED3	Configuration status LEDs	LEDs que exibem a configuração atual do FPGA.
Clock Circuitry		
Y2	Oscilador	Sinal de clock de 50 MHz dirigidos ao FPGA
J4	Entrada para clock externo	Conector para o pino de clock do FPGA.
Power Supply		
J26	**DC power jack**	**Fonte de alimentação 17 V DC.**
D34	Bridge rectifier	Power rectifier allows for center-negative or center-positive power supplies

Ainda com relação à Figura B.1, U60 é um FPGA Stratix II. Algumas placas incluem um dispositivo EP2S60F672C5 e outras um EP2S30F672C5. Na tabela B.2, são apresentadas as características de cada dispositivo. Neste trabalho foi utilizado o EP2S60.

Tabela B.2 – Características dos dispositivos EP2S60F672C5 e EP2S30F672C5 [StratixII 2005].

Característica	EP2S30	EP2S60
ALMs	13.552	24.176
Adaptive look-up tables (ALUTs)	27.104	48.352
Equivalent Les	33.880	60.440
Blocos M512 RAM	202	329
Blocos M4K RAM	144	255
Blocos M-RAM	1	2
Total de bits de RAM	1.369.728	2.544.192
Blocos DSP	16	36

18-bit x 18-bit multipliers	64	144
Enhanced PLLS	2	4
Fast PLLs	4	8
User I/O pins	500	492