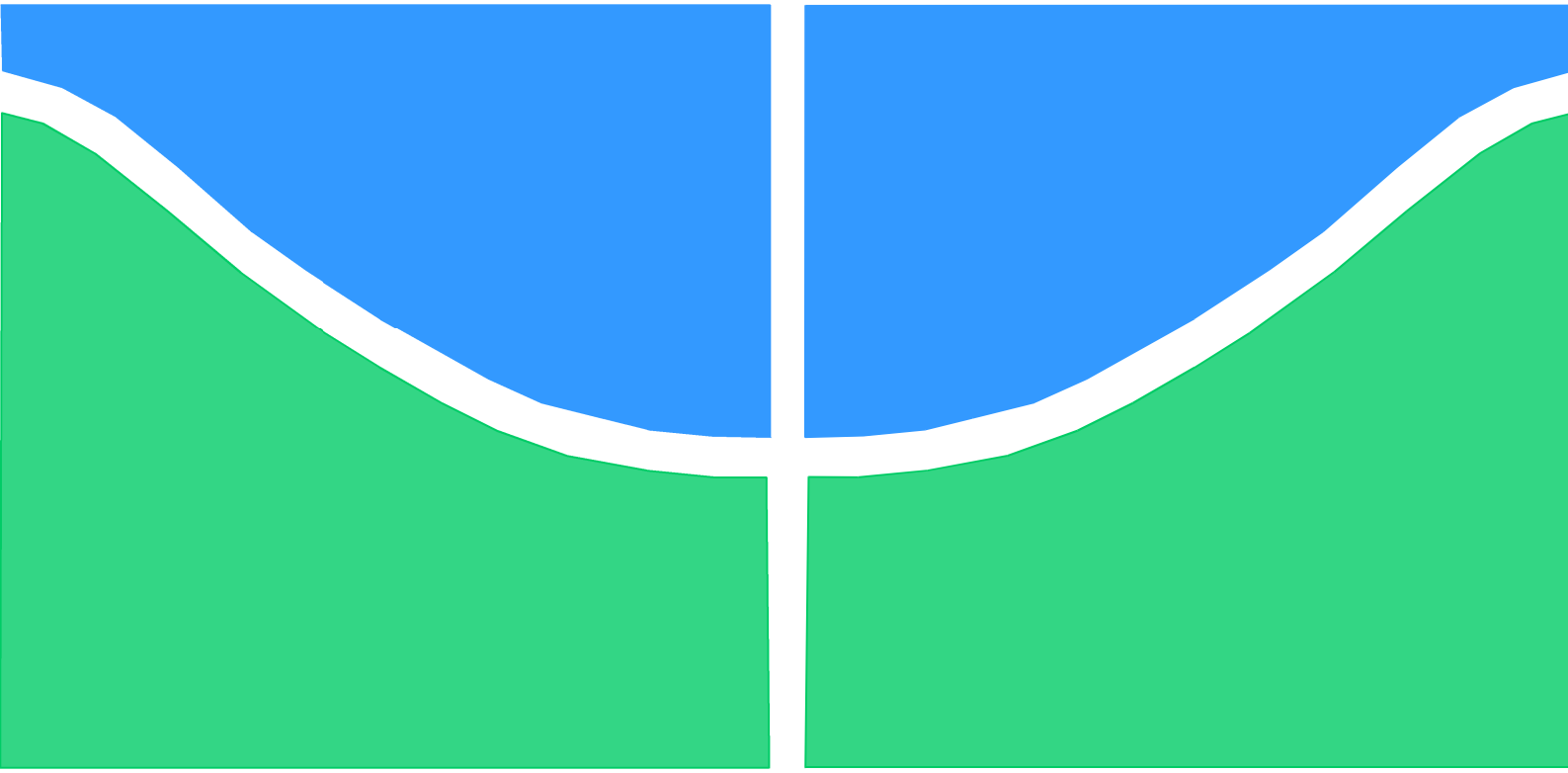


TRABALHO DE GRADUAÇÃO

PROJETO DE UM EQUIPAMENTO DE RECONHECIMENTO DE COMANDOS DE VOZ

André Luis Krueger de Moraes

Brasília, julho de 2007

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

PROJETO DE UM EQUIPAMENTO DE RECONHECIMENTO DE COMANDOS DE VOZ

André Luis Krueger de Moraes

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Lúcio Martins Silva, UnB/ ENE (Orientador) _____

Prof. Ricardo Zelenovsky, UnB/ ENE (Orientador) _____

Eng. Francisco Augusto da Costa Garcia _____

RESUMO

O projeto consiste no projeto de um equipamento de reconhecimento de comandos de voz. Foi decidido que o objetivo era a confecção de um equipamento que fosse capaz de reconhecer 6 comandos de voz. Para se alcançar o objetivo foi utilizado o kit AT91SAM7S-EK fabricado pela empresa ATMEL. Esse kit possui um microcontrolador da família ARM 32 bits AT91SAM7S256. O kit pode ser programado em assembly ou na linguagem C. Escolhemos a linguagem C devido à facilidade de programação em comparação com o assembly. Foi feita a interface entre o conversor analógico-digital do kit e o microfone do tipo eletreto utilizando um amplificador de dois estágios com ganho variável. A técnica escolhida de analise de voz foi a que utiliza quantização vetorial com coeficientes mel-cepstrais. Para o reconhecimento é feito um treinamento do equipamento onde o usuário faz diversas locuções dos comandos a serem reconhecidos e o software faz a analise dos sinais e constrói um dicionário para cada comando. Após a fase de treinamento o equipamento está pronto para realizar o reconhecimento, onde o usuário fala o comando e o equipamento realiza o reconhecimento. Ao capturar o sinal, o programa extrai os coeficientes mel-cepstrais e os compara com o dicionário de cada comando, decidindo qual comando é o mais próximo da locução. Neste trabalho está descrito todo procedimento seguido para o projeto do equipamento, a teoria utilizada e testes realizados.

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Este capítulo apresenta considerações gerais preliminares relacionadas ao projeto de equipamentos de reconhecimento de voz.

1.1 ESCOPO E MOTIVAÇÃO DO TRABALHO

Esse projeto teve como objetivo a confecção de um equipamento de reconhecimento de comandos de voz, com capacidade de reconhecer 6 comandos de voz pré-definidos pelo usuário. Para alcançar esse objetivo, primeiramente procurou-se uma plataforma onde seria construído o sistema. Foi decidido que esse projeto seria feito utilizando um microcontrolador, aproveitando o embasamento teórico obtido no decorrer do curso de graduação na programação desses dispositivos.

Decidiu-se fazer o equipamento na plataforma embarcada no lugar da plataforma PC pois a idéia é que o sistema seja dedicado a automação residencial, um equipamento que futuramente poderá automatizar diversas funções, como acender luzes por exemplo através de comandos de voz.

Pesquisou-se vários tipos de microcontroladores. A família ARM de 32 bits foi a escolhida devido a sua alta velocidade, sua facilidade de programação, sua extensa documentação, integração de quase todo hardware necessário ao projeto e sua disponibilidade em kits de desenvolvimento completos para a aplicação. Foi feito uma pesquisa dos diversos tipos de microcontroladores ARM 32 bits, optamos pelo que apresentava a documentação mais extensa.

A teoria de reconhecimento de voz é uma área ainda em pesquisa e é amplamente documentada, inclusive em livros[1]. Esse foi um dos fatores que determinou a escolha do projeto.

O reconhecimento de voz é o futuro das interfaces homem-máquina, existindo grande pesquisa nessa área. A tendência é que a voz humana substitua os tradicionais dispositivos de entrada de dados, como os teclados, por exemplo[1].

Essa área exige uma razoável capacidade de processamento dos dispositivos, por isso são utilizados microcomputadores ou microcontroladores de ponta para a sua implementação. Esse foi um dos motivos para se utilizar processadores da família ARM 32 bits, pois é a família de microcontroladores mais rápidos que temos acesso.

Existem vários tipos de reconhecimento de voz[1], como exemplos citamos o reconhecimento de locutor, onde se busca identificar quem está falando; o reconhecimento de comandos verbais, onde se busca identificar a palavra dita pelo locutor, normalmente uma palavra isolada, e fazer alguma ação em resposta; o reconhecimento de palavras isoladas, onde se busca identificar palavras ditas isoladamente pelo usuário; e o reconhecimento de fala contínua, onde se busca identificar textos inteiros ditos pelo usuário, contendo várias palavras que são ditas de forma contínua como quando se lê um texto. O último é de longe o mais complexo de todos e é onde se concentra a maioria das pesquisas atuais. O reconhecimento de voz implementado nesse trabalho é o reconhecimento de comandos de voz.

A técnica de reconhecimento utilizado nesse projeto é a que utiliza os coeficientes mel-cepstrais e a quantização vetorial, como técnicas de classificação de padrões.

O equipamento foi projetado e deveria ter sido feitos testes com o mesmo, mas devido a problemas de limitação de memória não foi possível testá-lo, como será visto no decorrer desse trabalho.

2 Hardware

Nesse capítulo é descrito o "hardware" utilizado no projeto, bem como os motivos da escolha do mesmo e suas características.

2.1 KIT AT91SAM7S-EK

Foi decidido se utilizar um kit de desenvolvimento no projeto para minimizar o trabalho com projeto de hardware necessário para o projeto. Existem vários kits de microcontroladores da família ARM disponíveis no mercado, cada um com as suas características particulares. Foi feito uma pesquisa e decidimos utilizar o kit AT91SAM7S-EK da fabricante Atmel, pois era o mais adequado e com documentação mais extensa do mercado[2].

O kit escolhido possibilita o desenvolvimento de aplicações rodando no microcontrolador AT91SAM7S256 da Atmel[2].

A placa AT91SAM7S-EK possui os seguintes componentes[2]:

- Microcontrolador AT91SAM7S256 da Atmel;
- Uma interface USB;
- Quatro LEDS de uso genérico;
- Quatro botões do tipo "pushbuttons" de uso genérico;
- Área de prototipagem;

Na Figura 2.1 se encontra um diagrama da placa do kit AT91SAM7S-EK[2].

O kit AT91SAM7S-EK é alimentado pela porta USB ou por fonte de alimentação de 7 a 14V.

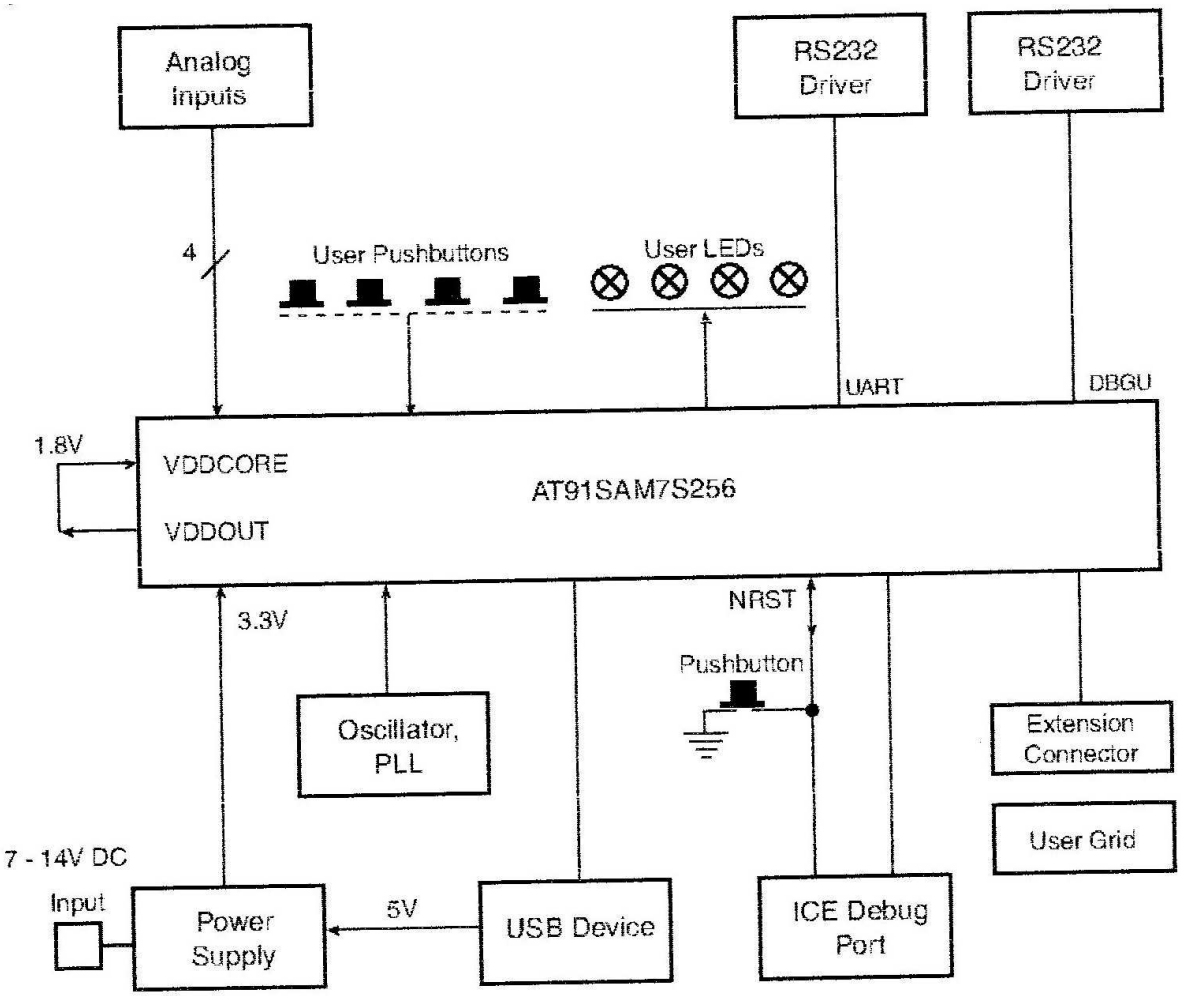


Figura 2.1. Diagrama do kit AT91SAM7S-EK.

Na Figura 2.2 se encontra uma foto do Kit.

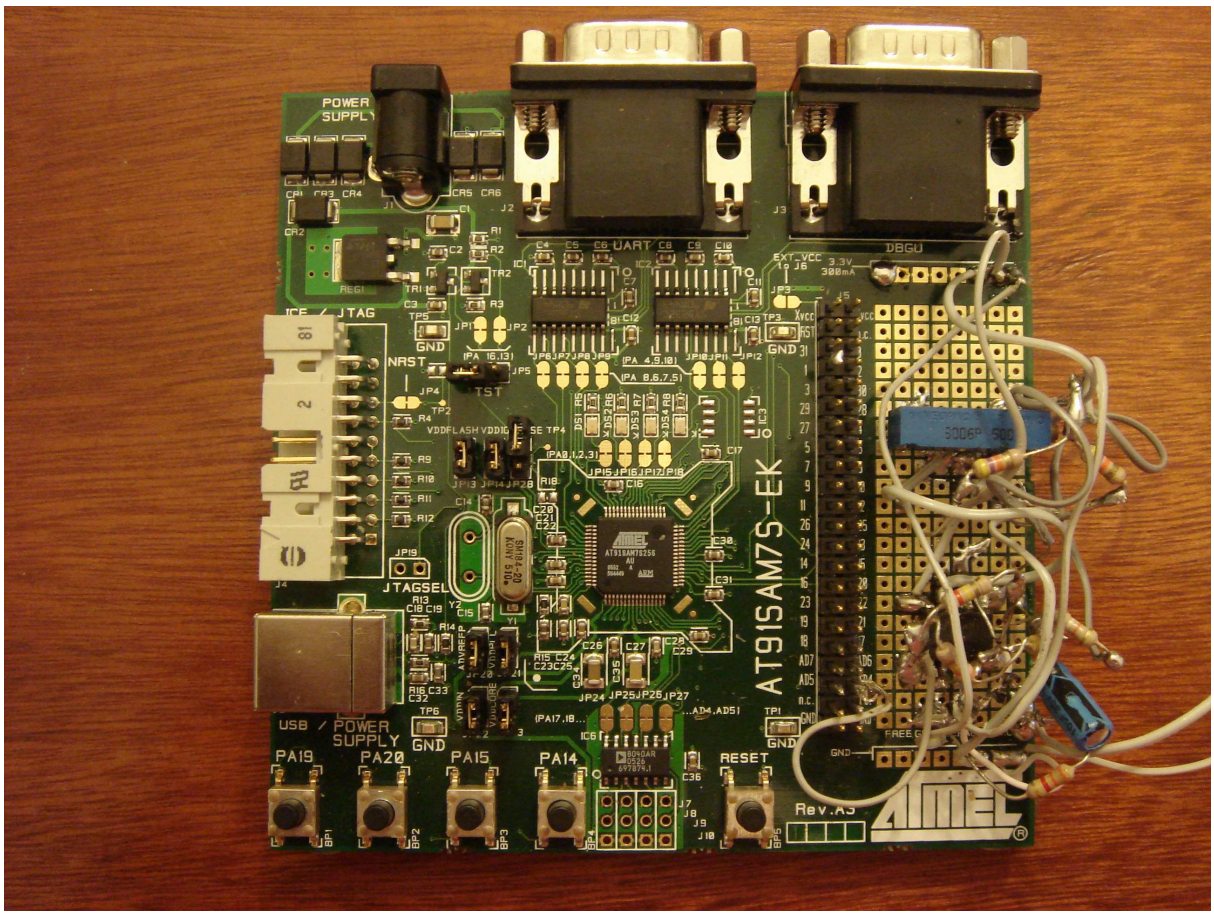


Figura 2.2. Foto do kit AT91SAM7S-EK.

2.2 MICROCONTROLADOR AT91SAM7S256

O microcontrolador escolhido para o projeto foi o AT91SAM7S256. Ele foi escolhido devido a sua alta velocidade e grande quantidade de memória RAM e ROM.

O microcontrolador AT91SAM7S256 possui as seguintes características[3]:

- Possui um processador ARM7TDMI ARM “Thumb” de 32 bits[4];
- Arquitetura RISC[4];
- Conjunto de instruções de alta densidade em 16 bits[4];
- “ICE” integrado;
- 256Kbytes de memória “Flash” interna de alta velocidade;
- Memória “Flash” organizada em 1024 paginas de 256 Bytes;
- Acesso a memória “Flash” em 30MHz;
- Tempo de escrita da “Flash” em página de 4ms

- Tempo de “erase” de “Flash” de 10ms;
- “Flash” de 10.000 ciclos de escrita;
- Capacidade de retenção da memória “Flash” de 10 anos;
- Capacidade de bloqueio de setores da memória “Flash”;
- 64Kbytes de SRAM de alta velocidade interna;
- Controlador de memória “Flash” integrado;
- Controlador de “reset” integrado;
- Detector do tipo “Brown-out”;
- Pino de “reset” externo;
- Gerador de “clock” integrado;
- Oscilador RC de baixo consumo;
- Oscilador integrado de 3 a 20MHz;
- Controlador de energia integrado;
- Modo “Slow Clock”
- Modo ocioso;
- Controlador de Interrupção avançado;
- Fontes internas de interrupção mascarável individualmente, com oito níveis de prioridade;
- Duas fontes externas de interrupção;
- Uma fonte rápida externa de interrupção;
- Unidade de DBGU;
- UART de 2 fios;
- Temporizador/Contador de 20 bits;
- Contador de intervalos de 12 bits ;
- “Watchdog”;
- Temporizador de tempo real de 32 bits com alarme;
- Um controlador de entrada e saída paralela;
- 32 linhas de entrada e saída;
- 11 canais controladores de dados dos periféricos;
- Uma porta USB 2.0 de 12 Mbits por segundo;
- “Tranceiver” USB integrado;
- 328 Bytes de FIFOs integrados para USB;
- Um controlador serial síncrono;

- Suporte a I^2S;
- 2 portas USART;
- Modulação e demodulação infravermelho IrDA;
- Suporte a ISO7816 T0/T1 “Smart Card”;
- Suporte a RS485;
- Decodificador codificador “Manchester”;
- Um interface SPI;
- Um contador temporizador de 16 bits de 3 canais;
- 3 entradas para “clocks” externos para o contador temporizador;
- Um controlador PWM de 16 bits;
- Uma interface “two-wire”;
- Um conversor Analógico Digital de 10 bits com oito canais;
- IEEE 1149.1 JTAG “Boundary Scan” para todos os pinos digitais;
- Regulador de 1.8V integrado;
- Linhas VDDIO de 1.8V ou 3.3V;
- Líder em MIPS/Watt;

Na Figura 2.3 se encontra um diagrama do microcontrolador[3].

Como pode-se ver este microcontrolador é bastante avançado, possuindo muitos recursos, fato que pesou na sua escolha.

Na Figura 2.4 se encontra uma foto do microcontrolador.

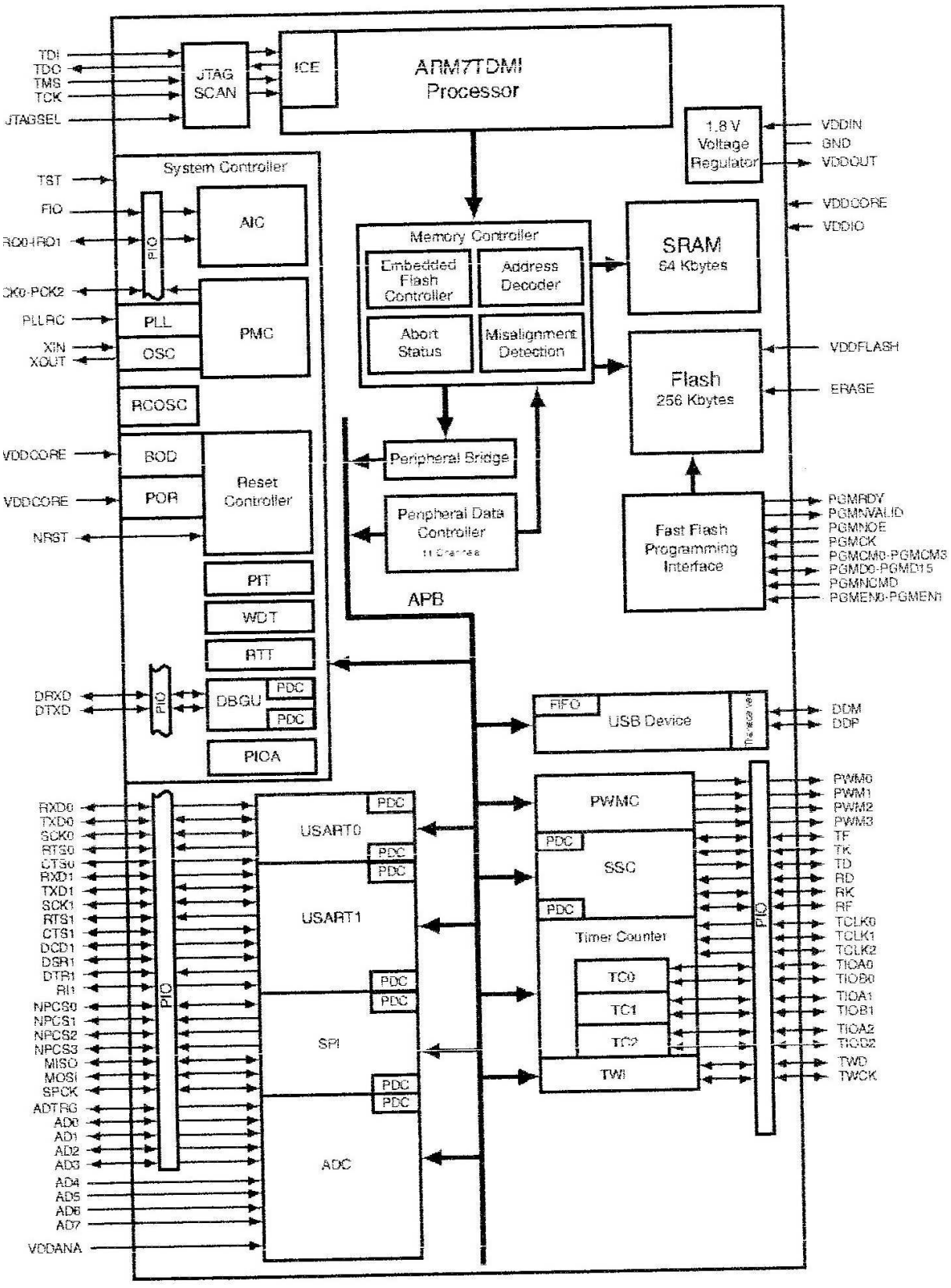


Figura 2.3. Diagrama do microcontrolador AT91SAM7256.

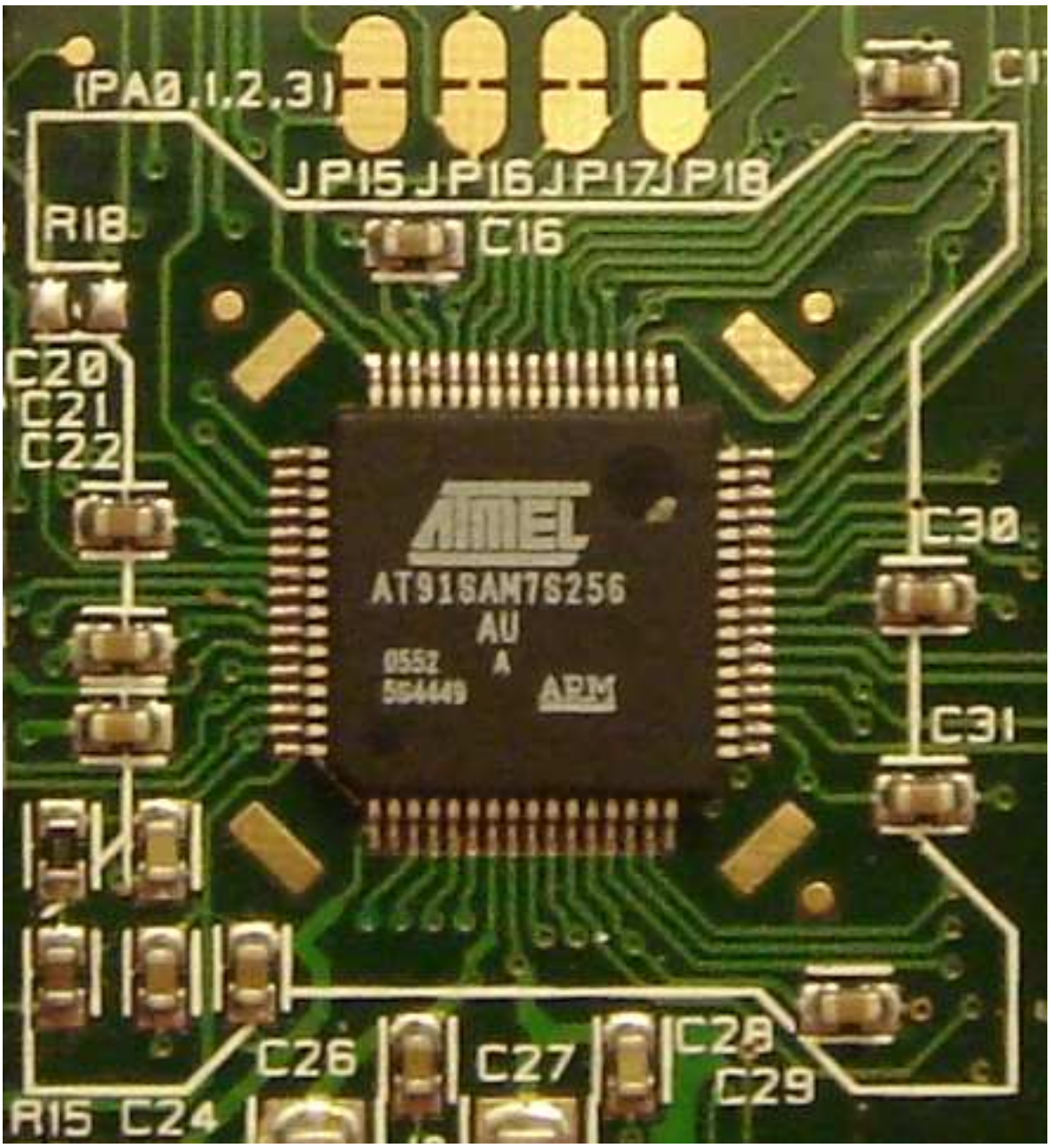


Figura 2.4. Foto do microcontrolador AT91SAM7S-256.

2.3 INTERFACE MICROFONE CONVERSOR ANALÓGICO DIGITAL

O kit AT91SAM7S-EK possui um conversor analógico digital de 10 bits de resolução[3]. Esse conversor pode ser utilizado para aquisição de áudio se for feita a devida interface entre o microfone e o conversor.

O kit converte sinais entre 0V e 3.3V. Valores acima e abaixo desse intervalo geram a saturação do conversor, acarretando erro na conversão[3]. A interface deve adaptar a tensão gerada pelo microfone para um valor dentro do intervalo de conversão, com amplitude suficiente para se obter uma conversão com resolução adequada.

O microfone escolhido foi do tipo eletreto, pois é o mais fácil de ser encontrado, é barato e adequado a aplicação. O microfone converte a pressão sonora em tensão. Como o valor dessa tensão é muito pequena para o conversor, torna-se necessário a utilização de um amplificador de tensão entre o eletreto e o conversor.

Existem várias topologias de amplificadores possíveis de se utilizar nessa situação, soluções via transistores bipolares, transistores CMOS e amplificadores operacionais, por exemplo. Foi escolhido topologia com amplificadores operacionais, pois é a de projeto mais simples e apresenta resultados satisfatórios para a aplicação.

O amplificador de tensão utilizado amplifica a tensão na saída do microfone para a faixa de conversão do conversor analógico digital. Para maior flexibilidade, adotamos uma solução de amplificação em dois estágios com ganho resultante variável.

O primeiro estágio de amplificação é um amplificador operacional de ganho fixo. O segundo estágio é um segundo amplificador operacional de ganho variável, de forma que, com o auxílio de um osciloscópio pudéssemos observar o ganho obtido e regular o ganho, através de um potenciômetro, de forma a onda de tensão gerada pelo microfone de eletreto esteja dentro dos limites do conversor analógico digital.

O circuito de interface entre o microfone de eletreto e o conversor analógico digital se encontra na figura (2.5)[7].

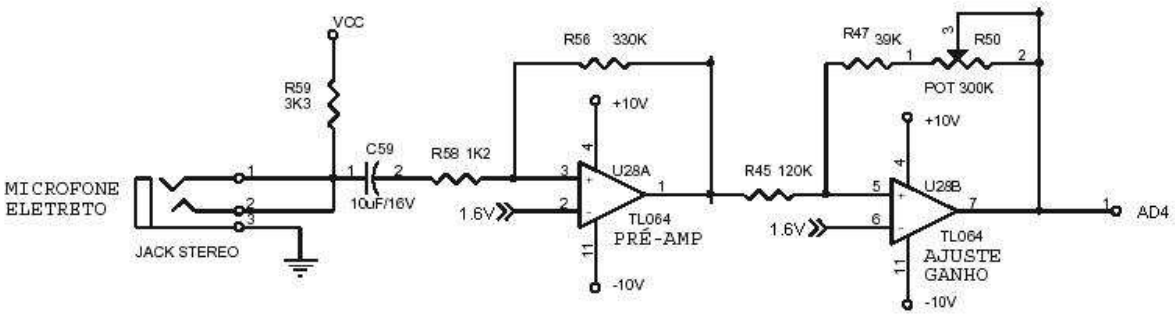


Figura 2.5. Interface entre o microfone e o conversor analógico digital.

Com o auxílio de um osciloscópio conseguimos ajustar com sucesso o ganho do amplificador da interface entre o conversor analógico digital e o microfone eletreto para valores satisfatórios de amplitude e de valor médio. O sinal obtido quando se fala a uma altura razoável é de valor médio 1.65V e amplitude média de 1.60V.

3 TEORIA DE RECONHECIMENTO

Esse capítulo apresenta as técnicas de reconhecimento de comandos de voz utilizada no software do equipamento projetado.

3.1 AQUISIÇÃO DE SINAIS DE VOZ

A voz humana possui um espectro de freqüências bastante amplo[1], sendo que é necessário uma solução de compromisso para o problema espectro da voz humana e a taxa de amostragem de aquisição desse sinal. Quando se utiliza uma taxa de amostragem baixa perde-se informação da voz amostrada, quando se utiliza uma taxa de amostragem alta tem-se mais esforço computacional no processamento desse sinal de voz. Na maioria das aplicações são utilizadas freqüências de no mínimo 8kHz para a amostragem da voz humana[1]. Essa é a freqüência de amostragem utilizada no sistema desenvolvido.

A fala humana é um sinal que mantêm sua composição espectral aproximadamente constante durante curtos intervalos de tempo, mas apresenta grandes variações espectrais ao longo de intervalos de tempo longos. Torna-se conveniente, então, a analise da voz humana em janelas de curta duração, onde a composição espectral se mantenha aproximadamente constante[1], como ilustra a Figura 3.1.

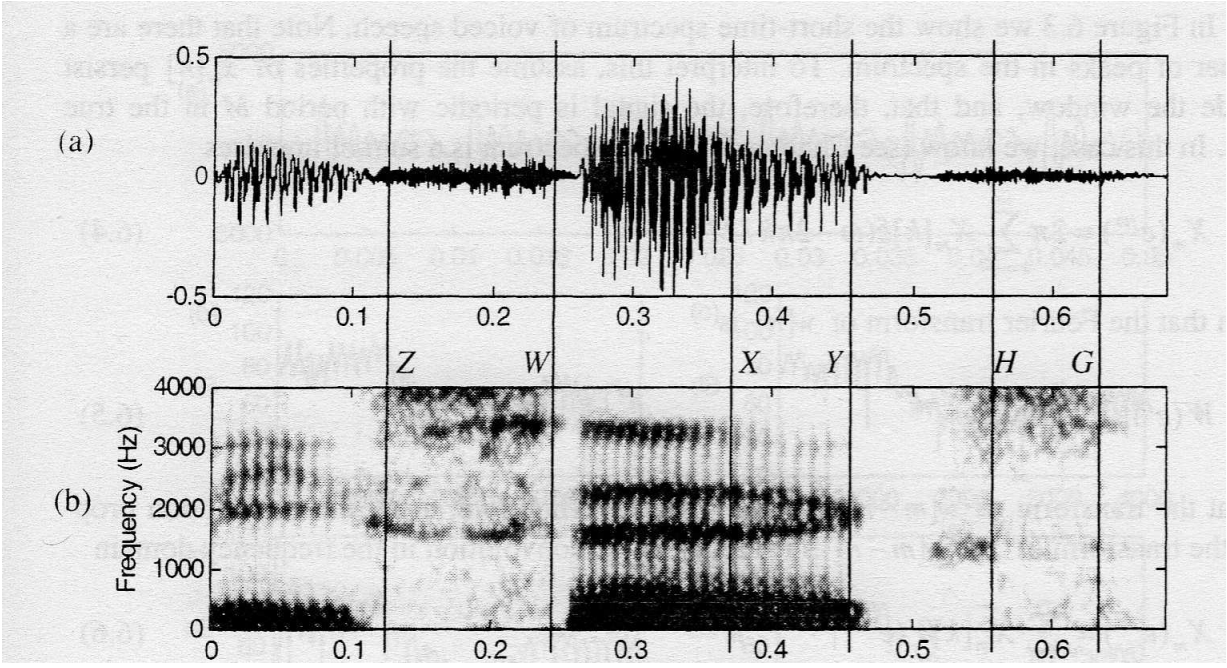


Figura 3.1. Sinal de voz e sua composição espectral.

Normalmente utilizam-se janelas com certa superposição para que não se perca as informações entre as transições entre uma janela e outra[1]. Nesse projeto escolheu-se não utilizar superposição de janelas devido a restrições de memória disponível no hardware utilizado. Utilizou-se 63 janelas sem superposição de 16ms cada, com um total de 1,008s, tempo suficiente para se dizer cada comando. Ou seja, para cada locução de um comando, foi feita uma gravação com duração de 1,008s.

3.2 TRANSFORMADA RÁPIDA DE FOURIER (FFT)

FFT é a sigla em inglês para transformada rápida de Fourier. É uma família de algoritmos computacionalmente mais eficientes de se calcular a transformada discreta de Fourier (DFT) [1].

A DFT é definida pela equação 3.1:

$$X_N[k] = \sum_{n=0}^{N-1} x_N[n] e^{-j2\pi nk/N} \qquad 0 \le k \le N \tag{3.1}$$

onde $x_N[k]$ é o dado sobre o qual se fará o calculo da DFT e N é o número de amostras contidas no dado.

DFT é a forma digital da transformada de Fourier do mundo analógico, sendo aplicado para dados discretos no tempo.

A computação direta da DFT requer N^2 operações, supondo que as exponenciais já tenham sido computadas[1]. Esse grande número de operações necessárias torna a DFT inviável de ser utilizada na maioria das aplicações práticas. Os algoritmos FFT requerem somente $N\log_2 N$ operações, o que os tornam muito mais rápidos que a DFT[1]. Por essa razão normalmente se usa a FFT nas aplicações em geral, inclusive para o processamento de voz.

Exemplos de algoritmos de FFT são o radix-2, o radix-4, radix-8, o split-radix, o prime-factor, etc[1]. O mais simples de todos é o radix-2, razão pela qual optamos utilizá-lo na nossa aplicação.

O algoritmo radix-2 possui duas variantes, a dizimação no tempo e dizimação na freqüência. O algoritmo do tipo dizimação no tempo está exemplificado no diagrama da Figura 3.2, onde é computado a FFT de um dado com 8 amostras.

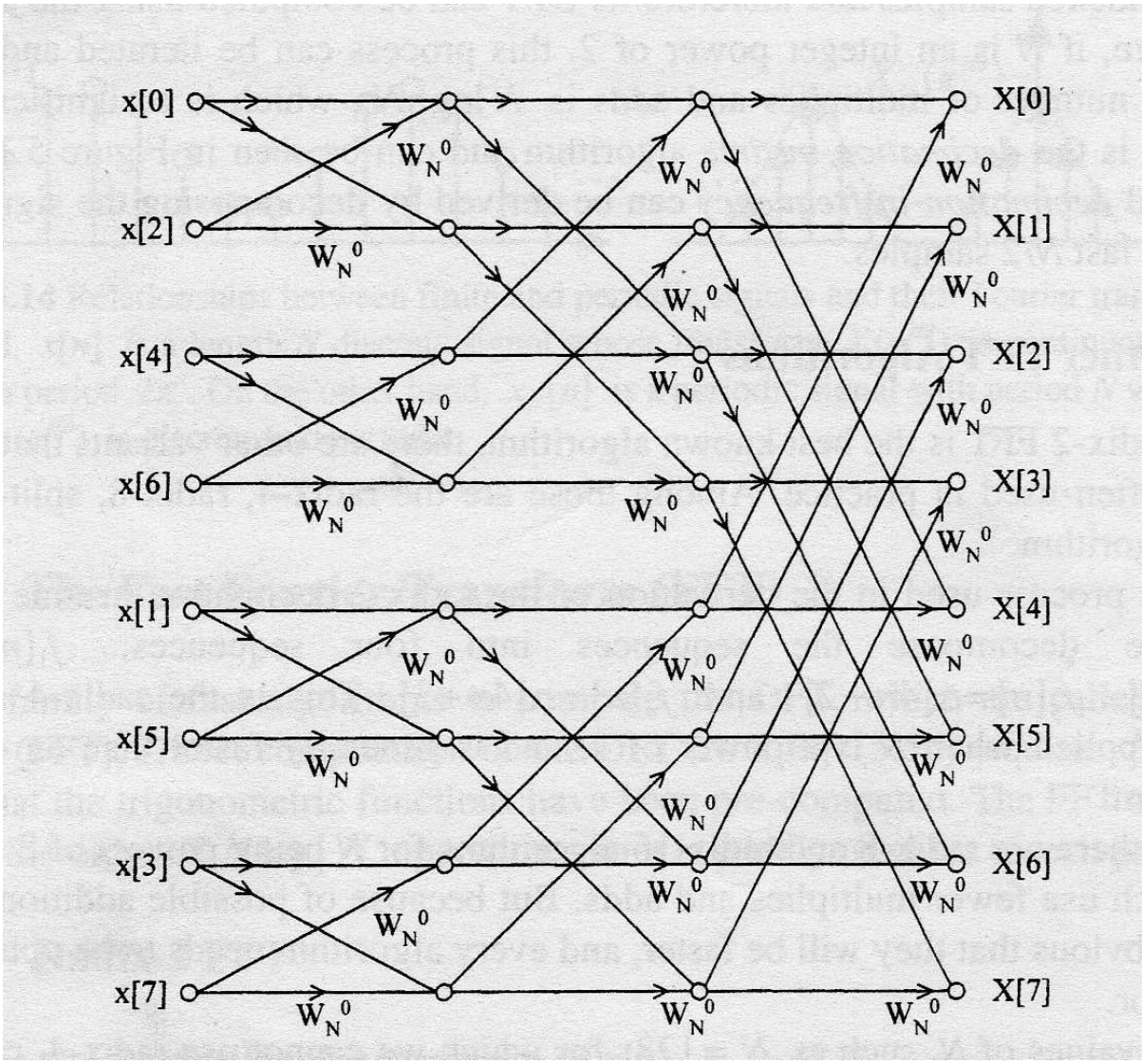


Figura 3.2. Diagrama de uma FFT de um dado com 8 elementos.

onde $W_N = e^{-j2\pi/N}$.

O elemento básico da radix-2 FFT é a operação borboleta, que está mostrada na seguinte figura:

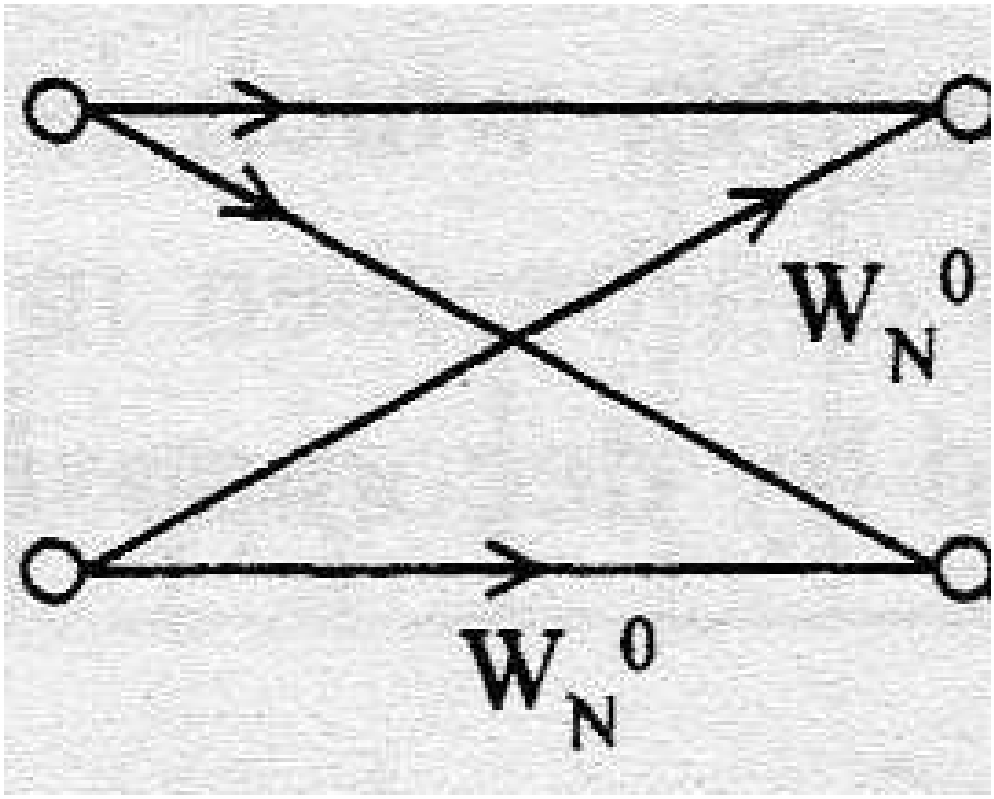


Figura 3.3. Diagrama de uma operação borboleta de uma FFT.

O algoritmo radix-2 é feito se aplicando sucessivas operações borboletas até que estas tenham o tamanho da metade das amostras, como exemplificado na Figura 3.2.

3.3 ANÁLISE MEL-CEPSTRAL

Para fazer a análise de sinais de voz é preciso segmentá-lo em janelas ou quadros como foi dito anteriormente. A análise da voz baseia-se na caracterização do sinal contido em cada janela. Para sua caracterização são extraídos alguns parâmetros desse sinal, a fim de diferenciá-lo.

Os coeficientes mel-cepstrais[1] são os parâmetros mais utilizados em aplicações de reconhecimento de voz. Esses coeficientes visam aproximar a forma de percepção de sons do sistema auditivo humano.

Os coeficientes mel-cepstrais são obtidos utilizando um banco de filtros que são aplicados sobre a FFT do sinal, onde cada coeficiente é obtido através de um dos filtros. A largura e a posição dos filtros são obtidos utilizando a escala de freqüências mel, que é dada pela seguinte equação:

$$B(f) = 1125\ln(1 + \frac{f}{700}) \tag{3.2}$$

onde f é a freqüência em Hertz e B a sua correspondente na escala mel.

O número de filtros é definido pela quantidade de coeficientes mel-cepstrais que se deseja obter de cada janela de sinal de voz. Cada filtro dá origem a um coeficiente. Os filtros utilizados são triangulares e espaçados utilizando a escala mel como mostrado na Figura 3.4.

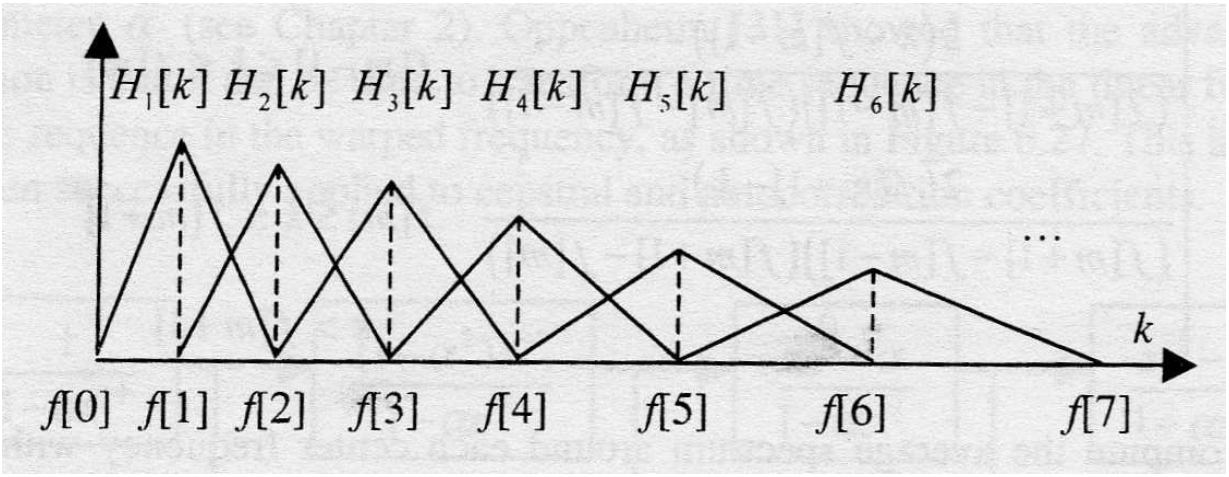


Figura 3.4. Filtros para obtenção dos coeficientes mel-cepstrais.

Cada filtro H_m é obtido utilizando as seguintes equações:

$$
\begin{aligned}
H_m[k] &= 0 && k < f[m-1] \\
H_m[k] &= \frac{2(k - f[m-1])}{(f[m+1] - f[m-1])(f[m] - f[m-1])} && f[m-1] \le k < f[m] \\
H_m[k] &= \frac{2(f[m+1] - k)}{(f[m+1] - f[m-1])(f[m+1] - f[m])} && f[m] \le k < f[m+1] \\
H_m[k] &= 0 && k > f[m+1]
\end{aligned}
\tag{3.3}
$$

onde H_m é a amplitude do filtro em cada ponto, *m* é o número do filtro que se está calculando.

O termo *f[m]* é dado por:

$$f[m] = \left(\frac{N}{F_s}\right) B^{-1}\left(B(f_l) + m\frac{B(f_h) - B(f_l)}{M+1}\right) \tag{3.4}$$

onde *N* é o tamanho da FFT, F_s é a freqüência de amostragem do sinal de voz, *B* é dada pela equação 3.2 e B^{-1} é a inversa de *B*, dada pela equação 3.5, *M* é o número de filtros utilizados, f_h é a freqüência máxima dos filtros e f_l é a menor freqüência do banco de filtros.

$$B^{-1}(b) = 700(\exp(b/1125) - 1) \tag{3.5}$$

Para calcular os coeficientes mel-cepstrais deve-se computar o logaritmo da energia da saída de cada filtro do banco de filtros, da seguinte forma:

$$S[m] = \ln\left[\sum_{k=0}^{N-1} \left|Xa[k]\right|^2 H_m[k]\right] \qquad 0 < m \leq M \tag{3.6}$$

Os coeficientes mel-cepstrais são dados por:

$$c[n] = \sum_{m=0}^{M-1} S[m]\cos(\pi n(m-1/2)/M) \qquad 0 \leq n < M \tag{3.7}$$

Na aplicação aqui relatada, tinha-se uma limitação de processamento dada pelo microcontrolador, por isso decidiu-se não utilizar os componentes mel-cepstrais diretamente pois envolveria grande quantidade de operações. Optou-se assim por utilizar como coeficientes de caracterização dos sinais de voz a energia de cada filtro de acordo com a equação:

$$c[m] = \sum_{k=0}^{N-1} \left|X_N[k]\right|^2 H_m[k] \qquad 0 < m \leq M \tag{3.8}$$

Esses coeficientes são suficientes para o reconhecimento de comandos verbais e serão denominados coeficientes mel-cepstrais no decorrer desse texto. Na aplicação utilizou-se 10 coeficientes mel-cepstrais.

3.4 MODELAGEM DE COMANDOS VERBAIS USANDO DICIONÁRIOS

Uma das técnicas utilizadas na modelagem de comandos verbais se chama quantização vetorial. Essa técnica constrói um conjunto de vetores que melhor caracterizam todo o conjunto de informação obtida na fase de treinamento. O conjunto desses vetores é chamado de dicionário[1], onde cada comando possui o seu. Esse dicionário é composto pelo conjunto de coeficientes mel-cepstrais que melhor caracterizam o comando. Cada elemento desse dicionário é chamado de centróide[1]. O dicionário é um vetor que possui a menor distorção acumulada em relação a toda informação obtida na fase de treinamento, ou seja, os centróides são os coeficientes mel-cepstrais que apresentam a menor distorção em relação aos demais coeficientes obtidos no treinamento.

O Dicionário possui um tamanho que corresponde ao número de centróides utilizados. Quanto maior o seu tamanho, mais detalhadamente ele descreve o comando de voz. Na

aplicação aqui relatada utilizou-se seis dicionários de dez centróides cada um. Os dicionários são obtidos na fase de treinamento (seção 3.6), onde são ditos os comandos e o software extrai os centróides, compondo o dicionário, como será visto.

Para fins de reconhecimento, o sinal que se deseja reconhecer é comparado com os diversos dicionários de forma a se encontrar qual se parece mais com o sinal. Isso é feito encontrando-se o dicionário que possui a menor distorção acumulada em relação ao comando que se deseja reconhecer, como será visto adiante.

3.5 TREINAMENTO

A fase de treinamento é onde se concentra a maior parte das técnicas de reconhecimento de comandos de voz. É no treinamento que se obtém os dicionários dos comandos de voz.

No treinamento é falado certo número de locuções de cada comando, esse número é arbitrário, mas quanto maior a quantidade maior é a precisão do reconhecimento. Em seguida é computado a FFT de cada janela de cada locução de cada comando de voz. Retiram-se os coeficientes mel-cepstrais de cada locução e se armazena todos os coeficientes de todas as locuções de cada comando em um único vetor.

De posse de todos os coeficientes mel-cepstrais precisa-se encontrar os centróides que melhor representam todo esse conjunto de informação. Os centróides que satisfazem essa condição são os que possuem a menor distorção em relação ao conjunto de coeficientes mel-cepstrais do comando[1]. A distorção é calculada como sendo a distância acumulada entre os coeficientes mel-cepstrais e o centróide mais próximo.

Um dos algoritmos utilizados para se encontrar os centróides a partir do conjunto de coeficientes mel-cepstrais é chamado de K-Means na literatura[1]. O algoritmo K-Means é:

Passo 1: Inicialização: Escolha um conjunto inicial de centróides.

Passo 2: Classificação dos coeficientes: Divida o espaço amostral de coeficientes mel-cepstrais em células, cada uma centralizada em um centróide, como mostrado na figura (3.5). Cada coeficiente pertencerá a célula do centróide mais próximo.

Passo 3: Cálculo dos novos centróides: Calcule os novos centróides como sendo a média de todos o coeficientes mel-cepstrais contidos em cada célula.

Passo 4: Iteração: Repita os passos 2 e 3 até que a razão da distorção com os centróides novos com a com os centróides antigos seja menor que um certo valor de parada.

O valor de parada utilizado na nossa aplicação é de 0,001.

Esse algoritmo supõe que já se tenha um conjunto de centróides iniciais. Esses centróides podem ser encontradas de várias maneiras, uma dela é o algoritmo LBG[1]:

Passo 1: Escolha o número de células como sendo um. Encontre o centróide de todo o conjunto de dados.

Passo 2: Divida o espaço amostral em duas células divididas pelo centróide anterior. Escolha os novos centróides de forma aleatória.

Passo 3: Use o algoritmo K-Means para otimizar os novos centróides.

Passo 4: Continue dividindo o espaço amostral em células até o numero de centróides ser igual ao desejado.

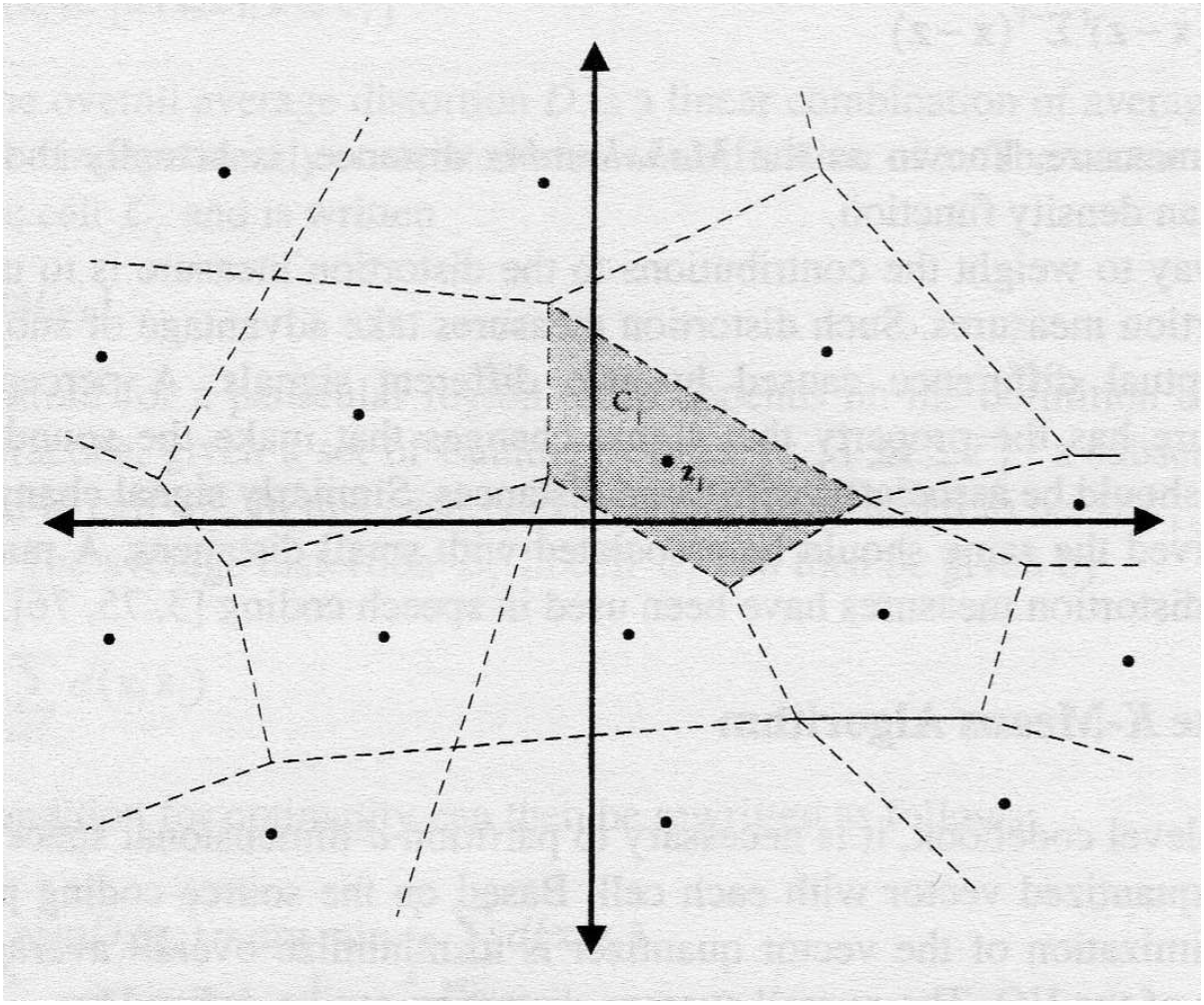


Figura 3.5. Centróides de um espaço amostral.

Com esses algoritmos podemos obter os dicionários.

Na aplicação utilizam-se 10 locuções de cada comando e dicionários com 10 centróides. Por motivos de limitação de hardware não se utilizou o algoritmo LBG, utilizou-se no lugar coeficientes mel-cepstrais aleatórios como sendo os centróides iniciais.

3.6 RECONHECIMENTO

Nesse ponto já se tem os dicionários, obtidos através da fase de treinamento. Agora só falta reconhecer os comandos de voz.

Para se reconhecer os comandos, deve-se amostrar o sinal utilizando-se janelas, em seguida calcular a FFT das diversas janelas, extrair os coeficientes mel-cepstrais e compara-los com os dicionários. O dicionário que for mais próximo do sinal é suposto como sendo o comando.

O sinal é comparado com os dicionários através da distorção acumulada dos coeficientes mel-cepstrais em relação aos centróides de cada dicionário, aquele que resultar na menor distorção acumulada é o comando procurado[1]. A distorção é calculada como sendo a soma das distâncias de cada coeficiente mel-cepstral em relação ao centróide mais próximo.

4 SOFTWARE

Este capitulo apresenta todo o software desenvolvido para a aplicação de reconhecimento de voz. Aqui é implementada toda a teoria de reconhecimento de voz exposta anteriormente.

4.1 LINGUAGEM UTILIZADA

O microcontrolador que se utilizou pode ser programado utilizando a linguagem de baixo nível assembly para ARM ou utilizando a linguagem C.

A linguagem assembly para ARM é mais precisa e poderosa por ser uma linguagem de baixo nível, mas por outro lado é uma linguagem muito difícil de programar coisas complexas, pois tem que se ter controle de todo o hardware, memória, etc.

A linguagem C é uma linguagem de mais alto nível que o assembly para ARM. Utilizando a linguagem C não é necessário se preocupar tanto com a memória e com o hardware que se está utilizando, o compilador trata disso por você. Com esta linguagem temos todas as vantagens de uma linguagem de alto nível, como definições de tipos de variáveis, operações aritméticas, operações de controle de fluxo como “loops” e comandos condicionais (“if” e “else” por exemplo).

O compilador C para ARM converte a linguagem de alto nível para as instruções de baixo nível da família ARM, tratando de distribuir e calcular todas as posições de memória de todas as variáveis, e de transformar todas as operações aritméticas mais complexas em operações de baixo nível do ARM. Por exemplo, em C podemos fazer operações em ponto flutuante, mas o processador possui aritmética em ponto fixo, então o compilador converte as operações em ponto flutuante em operações em ponto fixo correspondentes.

Devido a todas as facilidades que a linguagem C apresenta em relação ao assembly para ARM optou-se por desenvolver o software utilizando a linguagem C. Embora um programa feito em assembly para ARM seja mais rápido e preciso que um feito em C, fazer um software da complexidade que se está fazendo em assembly ficaria inviável.

4.2 COMPILADOR UTILIZADO

Existem vários compiladores C para ARM disponíveis no mercado. Em geral são todos softwares proprietários, e as licenças são caras, o que dificulta o acesso de estudantes a estes softwares. Alguns desses softwares tem versões de demonstração que não se permite compilar o arquivo ou que só permite compilar códigos pequenos. Como o programa que foi elaborado é bastante grande, aproximadamente 45kb, essas ferramentas ficam inviáveis de serem utilizadas.

Felizmente, existe um compilador baseado em software livre que é mais simples, porém mais difícil de se utilizar. Esse compilador é o GCC, que é utilizado através do programa “programers notepad”[5]. Esse programa e o compilador vem no pacote “WinARM”, que possui todas as ferramentas em software livre necessárias para se programar os dispositivos da família ARM utilizando a linguagem C.

Na figura abaixo se encontra a tela do programa “programers notepad”:

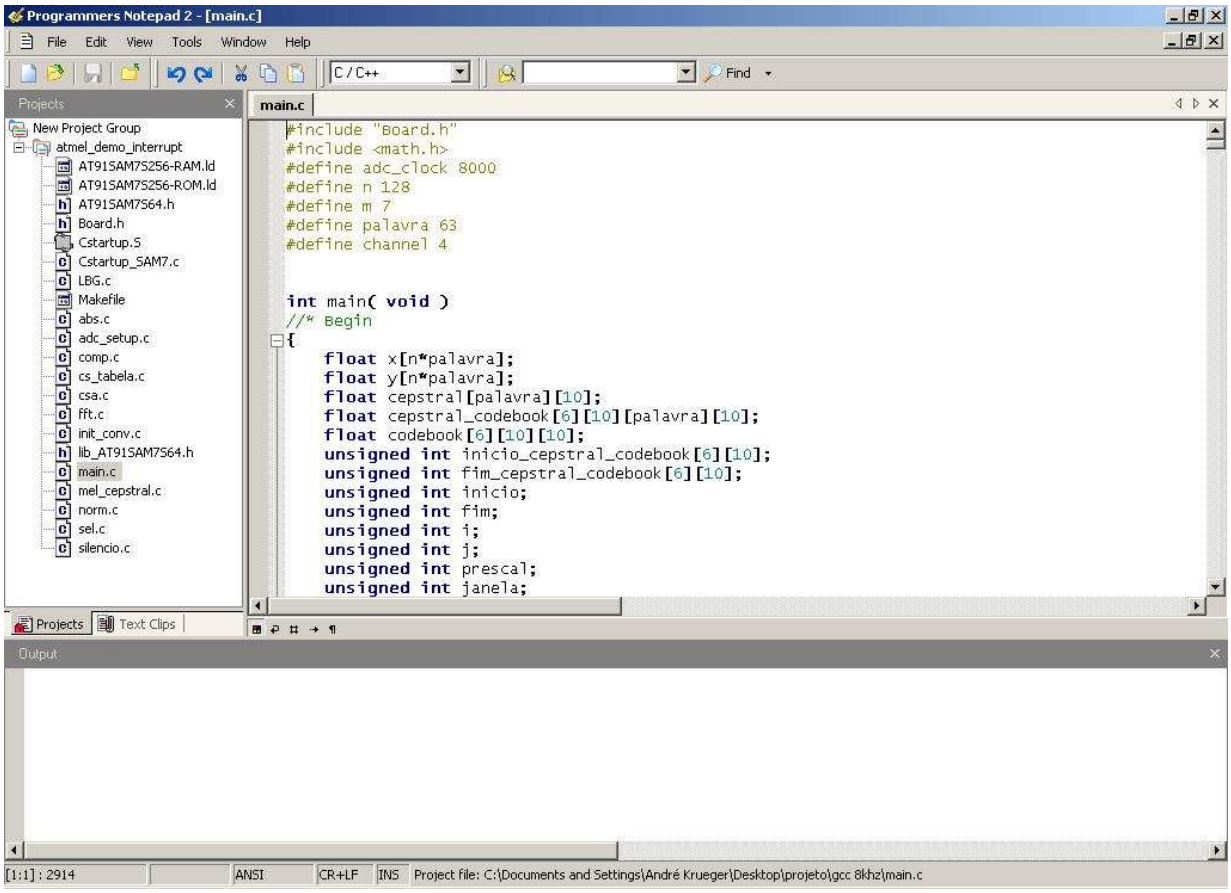


Figura 4.1. Tela do programa programers notepad.

Esse programa utiliza arquivos “MakeFile” para orienta-lo como compilar o arquivo[5], grande parte da dificuldade da utilização do “programers notepad” está na elaboração correta desse arquivo. A facilidade dos programas de código fechado é que eles elaboram automaticamente esse arquivo.

O arquivo “MakeFile” utilizado no nosso software se encontra no anexo A contido no CD.

4.3 CABEÇALHOS

O programa que foi desenvolvido utiliza uma série de cabeçalhos necessários para o seu funcionamento.

O cabeçalho *AT91SAM7S256-RAM.ld*[5] está no anexo B contido no CD. É o cabeçalho que diz ao compilador as características e o tamanho da memória RAM do microcontrolador AT91SAM7S256.

O cabeçalho *AT91SAM7S256-ROM.ld*[5] está no anexo C contido no CD. É o cabeçalho que diz ao compilados as características e o tamanho da memória ROM do microcontrolador AT91SAM7S256.

O cabeçalho *AT91SAM7S256.h*[5] está no anexo D contido no CD. É o cabeçalho mais importante do microcontrolador AT91SAM7S256, nele estão definidos todos os registradores do compilador, definidos com nomes amigáveis. É através desses nomes que se acessam os registradores. Além disso, esse cabeçalho define várias variáveis que contém informações úteis para o microcontrolador, como a freqüência do "clock" do processador por exemplo.

O cabeçalho *board.h*[5] está no anexo E contido no CD. É o cabeçalho onde são definidas as características do kit de desenvolvimento AT91SAM7S256-EK. Nele estão as variáveis de acesso aos leds e os botões do kit, por exemplo.

O cabeçalho *Cstartup.S*[5] está no anexo F contido no CD. O cabeçalho *Cstartup_SAM7.c* está no anexo G. São os cabeçalhos onde é feita a inicialização do microcontrolador AT91SAM7S256. Quando a placa é energizada esses códigos são os primeiros a serem executados.

O cabeçalho *lib_AT91SAM7S256.h*[5] está no anexo H contido no CD. Neste cabeçalho estão definidas diversas funções úteis que automatizam procedimentos no microcontrolador.

4.4 VISÃO GERAL DO SOFTWARE

O programa que foi desenvolvido é composto de várias funções, cada uma realizando uma tarefa necessária para se alcançar os objetivos traçados. As funções que compõe o programa são as funções *main, adc_setup, init_conv, fft, modulo, mel_cepstral, silencio, compara e LBG*.

A função *main* é a principal função do programa, ela é a função que está o corpo do programa, onde o programa começa a rodar. Quando o programa é iniciado essa função é chamada. Nela estão as principais variáveis do programa como o áudio convertido pelo conversor analógico digital, os dicionários, o resultado da FFT, etc. A função *main* coordena a chamada de todas as outras funções.

A função *adc_setup* é a função que faz a preparação do conversor analógico digital para que este possa iniciar a conversão. Essa função determina a freqüência de amostragem do conversor analógico digital, o número de bits da conversão, aciona o sinal de "clock", configura sua entrada, determina toda sua temporização e determina o seu canal de entrada.

A função *init_conv* é a função que realiza a conversão analógica digital do sinal de áudio da entrada do conversor. Essa função deve ser chamada em seguida da *adc_setup*.

A função *fft*, como o próprio nome indica, realiza a transformada rápida de Fourier do sinal de entrada. Essa função possui diversos parâmetros como o tamanho da FFT, a posição da janela onde se deve calcular a FFT no vetor de áudio, etc. A saída da função são dois vetores, um com a parte real da FFT e outra com a parte imaginária.

A função *modulo* calcula o módulo elevado ao quadrado do sinal de entrada, representado por sua parte real e imaginária.

A função *mel_cepstral* calcula os coeficientes mel-cepstrais do sinal de entrada. A entrada da função é o módulo ao quadrado da FFT. A saída um vetor com 10 coeficientes mel-cepstrais de cada janela do sinal.

A função *silencio* calcula a posição do início e do termino da locução de voz no vetor de entrada, retirando a parte que representa o silêncio do sinal, restando somente a parte que contém voz. A entrada dessa função é o vetor contendo todos os coeficientes mel-cepstrais do sinal e a saída são duas variáveis uma indicando o início da locução e outra indicando o fim da locução.

A função *compara* realiza a comparação do sinal de entrada com os dicionários, tendo como saída uma variável que indica a qual dicionário possui a menor distorção em relação ao sinal de entrada.

A função *LBG* realiza a construção dos dicionários. A entrada dessa função são os coeficientes mel-cepstrais dos sinais de treinamento e a saída são os dicionários correspondentes a esses sinais de treinamento.

Basicamente, essa é a composição do programa. A seguir serão descritas cada função.

4.5 FUNÇÃO ADC_SETUP

A função *adc_setup* está no anexo I contido no CD.

Essa função faz a preparação do conversor analógico digital do microcontrolador, definindo os seus diversos parâmetros. Os parâmetros da função *adc_setup* são dois. A

primeira se chama *adc_clock*, que é a freqüência de amostragem do sinal que o conversor deve utilizar. A segunda é *channel*, que é qual canal que o conversor deve converter. O microcontrolador pode converter 8 canais diferentes.

A primeira coisa que a função faz é ativar o "clock" do conversor, pois o padrão é que o mesmo esteja desabilitado por economia de energia. Em seguida a função configura o controlador PIO[3] do microcontrolador. Esse configuração é feita através da função *AT92F_ADC_CfgPIO* que a automatiza, contida no cabeçalho *lib_AT91SAM7S256.h*. PIO é o controlador que controla os sinais de entrada e saída do microcontrolador, é necessário que se informe a esse controlador que desejamos utilizar o conversor analógico digital, isso é feito dentro da função *AT92F_ADC_CfgPIO*.

Depois a função calcula a variável *prescal*[3]. Essa variável é a forma de se informar o microcontrolador qual a taxa de amostragem desejada do conversor analógica digital. A variável *prescal* se relaciona da forma da equação (4.1)[3] com a freqüência de amostragem em hertz e o "clock" central do processador em Hertz.

$$prescal = \frac{MCK}{2.adc\_clock} - 1 \qquad (4.1)$$

onde *MCK* é o "clock" central do processador e *adc_clock* é a freqüência de amostragem desejada do conversor.

Em seguida é configurado o registrador *ADC_MR*[3], onde é feita a configuração do conversor analógico digital. O primeiro parâmetro está desabilitando o "trigger"[3] de hardware, dessa forma o conversor só pode iniciar a conversão via comando por software. O segundo parâmetro seleciona qual entrada do microcontrolador será o sinal de "trigger", esse valor não importa, pois foi desabilitado o "trigger" via hardware. No terceiro parâmetro é definida a resolução máxima para o conversor, que é de dez bits. No quarto parâmetro é desabilitada a função "sleep", onde visa a economia de energia do microcontrolador. No quinto parâmetro é passado para o registrador o valor previamente calculado do *prescal*. No quarto e quinto parâmetros foram utilizados os valores padrões de temporização do conversor. Depois de configurado o registrador *ADC_MR* é configurado o regitrador *ADC_CHER* onde se define qual canal se deseja converter. Escolheu-se o canal 4 para a aplicação, pois o mesmo já se encontra pronto no kit.

4.6 FUNÇÃO INIT_CONV

A função *init_conv* está no anexo J contido no CD.

Essa função faz a conversão propriamente dita do sinal na entrada do conversor analógico digital.

A função *init_conv* possui 3 parâmetros. O primeiro é um ponteiro para o vetor onde será armazenada a parte real do sinal convertido pelo conversor. O segundo é um ponteiro para o vetor onde será armazenada a parte imaginária (que será igual a zero) do sinal convertido pelo conversor. O terceiro é a variável que indica a quantidade de conversões que serão feitas.

Inicialmente a função define como zero todo o vetor de entrada do sinal imaginário. Em seguida se inicia o "loop" onde é convertido todas as amostras do sinal de entrada. Primeiramente é dado o sinal *AT91C_ADC_START*, que está na biblioteca *AT91SAM7S256.h*, esse sinal inicia a conversão do conversor. Em seguida, se espera para que o dado convertido esteja pronto. Depois é atribuído ao elemento correspondente do vetor de entrada o valor convertido através da função *AT91F_ADC_GetLastConvertedData*, que está no cabeçalho *lib_AT91SAM7S256.h*.

4.7 FUNÇÃO FFT

A função *fft* está no anexo K contido no CD.

Essa função faz o cálculo da FFT do sinal.

A função *fft* possui 5 parâmetros. O primeiro é o vetor contendo a componente real do sinal a ser feita a FFT e onde será salva a parte real da FFT. O segundo é o vetor contendo a componente imaginária do sinal a ser feita a FFT e onde será salva a parte imaginária da FFT. O terceiro é o tamanho da FFT, que deve ser potência de dois. O quarto é o número de estágios da FFT, pois não era possível se calcular isso de forma simples. O quinto é a posição da janela que se deseja calcular a FFT no vetor de entrada.

Inicialmente, a função inicia o "loop" por todos os *m* estágios da FFT. O número de estágios é definido pela equação 4.2.

$$n = 2^m \tag{4.2}$$

onde *n* é o tamanho da FFT, *m* é o número de estágios.

Em seguida, a função calcula os “twiddle factors”. Depois a função faz as borboletas do presente estágio. Em seguida a função desembaralha os dados da FFT. O algoritmo da FFT utilizado inicia com os dados de entrada na ordem correta e termina, ao fim de todos os estágios com os dados embaralhados, necessitando que se desembaralhe.

4.8 FUNÇÃO MODULO

A função *modulo* está no anexo L contido no CD.

A função é bastante simples, calculando o módulo elevado ao quadrado do sinal complexo de entrada.

A função *modulo* possui quatro parâmetros. O primeiro é a parte real do dado de entrada e onde será salva o resultado do calculo do módulo. O segundo é a parte imaginária do dado de entrada. O terceiro é o tamanho da janela onde será calculada o módulo. O quarto é a posição da janela dentro da totalidade do vetor de entrada.

Essa função é constituída de um único “loop” onde é calculado o módulo ao quadrado do dado de entrada, da forma da equação (4.3).

$$M = R.R + I.I \tag{4.3}$$

Onde *M* é o modulo ao quadrado, *R* é a parte real do dado e *I* é a parte imaginária do dado.

4.9 FUNÇÃO MEL_CEPSTRAL

A função *mel_cepstral* está no anexo M contido no CD.

A função calcula os 10 coeficientes mel-cepstrais de cada janela do sinal de entrada.

A função *mel_cepstral* possui 2 parâmetros. O primeiro é o dado de entrada que é a saída da função modulo. O segundo é o vetor onde será salvo os coeficientes mel-cepstrais.

Essa função é composta de um único “loop” que percorre todas as janelas do sinal de entrada. Para cada janela é calculado o coeficiente mel-cepstral. Cada coeficiente mel-cepstral é obtido de um dos filtros triangulares como explicado na seção 3.4.

4.10 FUNÇÃO SILENCIO

A função *silencio* está no anexo N contido no CD.

A função detecta o início e o fim do sinal de voz no vetor de entrada, excluindo do sinal as partes correspondentes ao silêncio.

A função *silencio* possui 3 parâmetros. O primeiro é a variável que vai armazenar a posição de início do sinal de voz. O segundo é a variável que vai armazenar a posição do fim do sinal de voz. O terceiro é o dado de entrada, que é composto pelos coeficientes mel-cepstrais do sinal.

Essa função começa com um loop percorrendo todas as 63 janelas do sinal de entrada. É calculado a energia contida em cada janela. Em seguida procura-se as janelas com a menor e a maior energia. Em seguida é calculada a energia que será a fronteira entre o silêncio e a voz, como indicado na equação (4.4). Depois são varridas as janelas no sentido do início ao fim procurando-se a primeira janela com energia superior a energia de referência. Essa é a janela de início da voz. Em seguida são varridas as janelas no sentido do fim para o início procurando-se a primeira janela com energia superior a energia de referência. Essa é a janela do fim da voz.

$$ref = energia\_\min + 0.2(energia\_\max - energia\_\min) \tag{4.4}$$

Onde *energia_min* é a energia mínima, a *energia_max* é a energia máxima e *ref* é o sinal de referência.

4.11 FUNÇÃO COMPARA

A função *compara* está no anexo O contido no CD.

A função faz a comparação do sinal de entrada com os diversos dicionários, retornando uma variável que representa o dicionário que apresenta a menor distorção em relação ao dado de entrada.

A função compara possui 4 parâmetros. O primeiro é um vetor contendo os dicionários das 6 palavras. O segundo são os coeficientes mel-cepstrais do sinal de entrada. O terceiro é a variável que armazena a posição de inicio do sinal de voz. O quarto é a variável que armazena a posição do fim do sinal de voz.

Essa função começa com um “loop” pelos 6 dicionários. Depois tem outro “loop” pelas 63 janelas onde é associado a cada janela uma célula correspondente ao centróide mais próximo (com a menor distorção). Em seguida é calculada a soma da distorção de todas as janelas em relação aos centróides correspondentes. Depois é verificado se essa distorção é a menor calculada até então, se é ele armazena na variável correspondente o valor do dicionário que resultou nessa distorção. O “loop” continua pelas outras palavras. A função retorna após o termino do “loop” a variável que contem a referência ao dicionário que apresentou a menor distorção acumulada.

4.12 FUNÇÃO LBG

A função *LBG* está no anexo P contido no CD.

Inicialmente essa função faria o algoritmo LBG descrito anteriormente, mas por motivos de limitação de capacidade computacional decidiu-se não fazer exatamente o algoritmo LBG. A função *LBG* constrói os dicionários a partir dos dados de treinamento, implementando o algoritmo K-Means, partindo de centróides escolhidos de forma aleatória dentro dos dados de treinamento.

Essa função possui quatro parâmetros. O primeiro é um ponteiro para o vetor onde será salvo o dicionário. O segundo é um ponteiro para o vetor que contém os dados de treinamento. O terceiro é um ponteiro para o vetor que contém as posições de inicio da voz dos dados de treinamento. O quarto é um ponteiro para o vetor que contém as posições de termino da voz dos dados de treinamento.

A função inicia com um “loop” pelas seis palavras do treinamento. São encontrados os centróides iniciais de cada palavra. Em seguida é calculado a qual célula, correspondente a um centróide, pertence cada dado de treinamento, como sendo o centróide que possui a menor distorção em relação ao dado. Depois é calculado a distorção acumulada dos dados de treinamento em relação aos centróides iniciais. Em seguida a função entra no “loop” da função K-Means propriamente dita. Inicialmente, esse “loop” calcula a que célula cada dado pertence, como sendo a correspondente ao centróide mais próximo desse. Em seguida, é calculado os novo centróides, que é a média dos dados dentro de cada célula. Depois, é calculado a distorção em relação aos novos centróides. Em seguida a condição de finalização do “loop” é avaliada. Se a razão entre a distorção acumulada antiga e a distorção acumulada nova for menor que 0,001 o “loop” é interrompido e estes centróides são o dicionário da palavra correspondente.

4.13 FUNÇÃO MAIN

A função *main* está no anexo Q contido no CD.

Essa função é a função central do programa, é nela onde está o corpo do software. Quando o programa é inicializado a função que é chamada é a *main*. Essa função dita para o microcontrolador o que ele deve fazer, chamando todas as outras funções nos momentos oportunos.

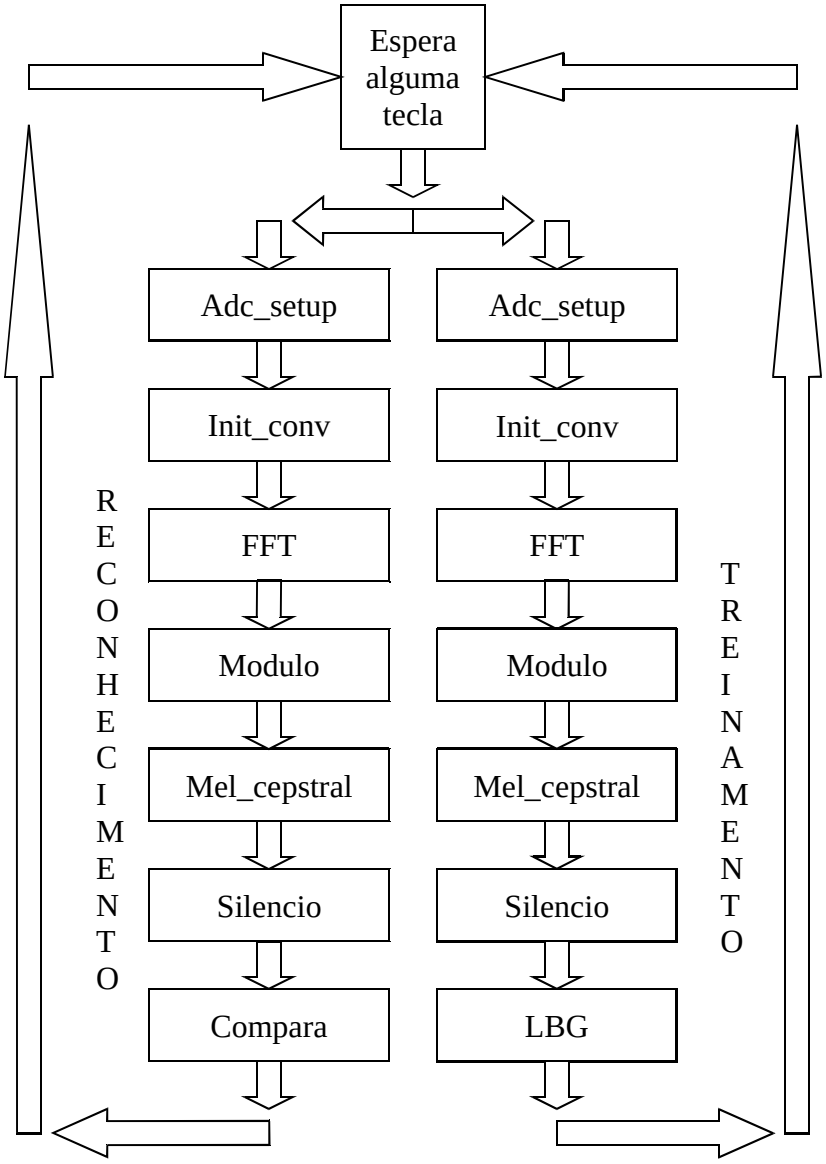
Inicialmente, a função *main* cria todas as variáveis a serem utilizadas pelo programa. Em seguida, é feito a inicialização do microcontrolador e do kit. Depois, o programa entra no seu "loop" principal. Inicialmente, o kit acende o led 1, de forma a avisar que esta dentro do "loop" principal. Em seguida, o microcontrolador fica esperando que alguma tecla seja apertada.

Se a tecla 1 é apertada ele entre no bloco correspondente a fase de reconhecimento. Dentro desse bloco de reconhecimento é aceso o led 2 que indica que o microcontrolador está convertendo os dados do conversor analógico digital. Em seguida, o programa chama a função *init_conv*, que inicia a conversão. Quando a conversão está terminada o kit apaga o led 2. Em seguida, o programa faz as FFTs e calcula os módulos das janelas do sinal que foi convertido, chamando as funções *fft* e *modulo* respectivamente. Depois, ele chama a função mel-cepstral para calcular os coeficientes mel-cepstrais das janelas do sinal obtido. Em seguida, o programa chama a função compara para comparar o sinal obtido com os dicionários, atribuindo o resultado a uma variável. Depois, o programa mostra o código binário do dicionário que foi atribuído ao sinal através dos leds. Termina o bloco de reconhecimento, voltando ao "loop" principal.

Se a tecla 4 é apertada, ele entra no bloco correspondente ao treinamento. Inicialmente o led 4 é aceso para avisar que estamos no treinamento. O programa espera o botão 2 ser apertado para iniciar a conversão da primeira locução da primeira palavra, acendendo o led 2 para avisar que esta convertendo. Concluída a conversão, o led 2 é apagado e o led 3 é aceso para avisar que esta processando o sinal convertido. Em seguida, calcula-se a FFT e o modulo do sinal e os coeficientes mel-cepstrais. Em seguida, ele apaga o led 3 e espera que o botão dois seja apertado. Quando o botão 2 é apertado, inicia-se a conversão da segunda locução da primeira palavra e assim sucessivamente até terminar a aquisição de todas as seis palavras do treinamento. Depois disso, é chamada a função *LBG* para que se calculem os dicionários

correspondentes aos dados de treinamento. O bloco acaba acendendo todos os leds e esperando que se aperte o botão 2, voltando ao “loop” principal.

No diagrama abaixo se encontra resumido a funcionalidade da função main:



5 TESTES

Este capítulo apresenta os testes realizados com o equipamento e os da metodologia utilizada.

5.1 METODOLOGIA UTILIZADA

Decidiu-se realizar os testes em duas etapas. A primeira etapa é o teste do algoritmo utilizando um computador comum. A segunda etapa é o teste do equipamento de reconhecimento de voz propriamente dito.

O teste utilizando computador foi feito adaptando-se o código já feito para o kit AT91SAM7S-EK para a plataforma PC. Foram necessárias poucas mudanças pois o código estava escrito em C, uma linguagem relativamente independente de plataforma. O código adaptado está no anexo R contido no CD.

O teste na plataforma PC foi feito utilizando-se arquivos de áudio gravados com o programa “GoldWave”. Os arquivos possuem 1 segundo de duração. Utilizou-se 6 comandos diferentes: abre, fecha, direita, esquerda, cima e baixo. Cada comando possui 10 locuções, gerando 60 arquivos de áudio para o treinamento do software. Em seguida foram gravados os comandos a serem reconhecidos. Foram utilizados 5 arquivos para cada comando gerando um total de 30 arquivos. Os arquivos foram gravados utilizando a extensão *.snd. Esse formato de arquivo não possui cabeçalho, facilitando a sua manipulação. Todos os comandos possuem taxa de amostragem de 8kHz e quantização de 16 bits, similarmente ao que foi feito no kit.

O programa para PC realiza o treinamento da mesma forma que o para o ARM faz e em seguida faz o reconhecimento de todos os comandos gravados, mostrando os resultados na tela. Foram utilizados as palavras abre, fecha, direita, esquerda, cima e baixo como comandos para o teste.

Os testes utilizando o kit ARM seriam feitos simulando a sua utilização normal. Inicialmente seria feito o treinamento com as palavras abre, fecha, direita, esquerda, cima e baixo. Em seguida seria passado para a etapa de reconhecimento, onde serão ditas 10 locuções de cada palavra e colheremos os resultados.

5.2 RESULTADOS OBTIDOS

Para fazer o teste utilizando o kit ARM foi compilado o código fonte do programa e o foi passado para o kit via porta USB utilizando o programa SAM-BA, fornecido pelo

fabricante Atmel. O programa foi compilado com sucesso e transmitido para o kit corretamente. Quando o programa foi executado verificou-se um problema. O programa funcionava até o momento de converter os dados do conversor analógico digital na fase de treinamento, onde travava.

Foi investigado o problema e foi descoberto que o programa estava exigindo mais memória RAM que o kit dispunha, gerando o travamento, pois o processador tentava acessar uma região fora da memória disponível no kit. Infelizmente foi possível executar o programa corretamente, inviabilizando os testes com o kit. Seria necessária a utilização de um kit que possuísse mais memória RAM que o que foi utilizado, mas não se teve acesso a tal kit, o que prejudicou os testes.

Inicialmente, os testes na plataforma PC reconheceu com sucesso as palavras abre, fecha, cima e baixo. O programa confundiu as palavras direita e esquerda. Foram gravados novamente as locuções de treinamento e as locuções a serem reconhecidas das palavras direita e esquerda, de forma bem mais articulada e mais devagar que feito anteriormente. Com as novas locuções o programa na plataforma PC reconheceu de forma correta todos os comandos, funcionando perfeitamente. Pode-se concluir que o algoritmo funcionou perfeitamente, reconhecendo todos os comandos, sem erros. Pode-se ver então que o algoritmo de reconhecimento está correto, só não foi possível testar no kit por problemas de memória.

6 CONCLUSÃO

Este capítulo apresenta as conclusões obtidas com o presente trabalho.

O objetivo do projeto era a confecção de um equipamento capaz de reconhecer comandos de voz.

Foi aprendido a utilização do microcontrolador AT91SAM7S256[3] através do kit de desenvolvimento AT91SAM7S-EK[2]. Foram feitos estudos sobre as características desse kit de desenvolvimento e foi desenvolvida a forma como esse kit podia ser utilizado para alcançar o objetivo. Foi projetado e construído a interface necessária entre o microfone e o kit. Foi pesquisado a teoria de reconhecimento de voz. Foi desenvolvido o algoritmo a ser utilizado para o reconhecimento de voz. O algoritmo na plataforma ARM foi implementado utilizando a linguagem C. Foram feitos testes do software na plataforma PC com sucesso.

O objetivo do projeto foi alcançado em parte, visto que foi desenvolvido um software de reconhecimento e foi testado o seu funcionamento na plataforma PC, funcionando corretamente. Infelizmente não foi possível fazer o equipamento de reconhecimento de voz funcionar, devido ao fato de o software exigir mais memória que o kit possui, sendo necessário um hardware superior para implementar o software. Esse problema não era detectável no inicio do projeto.

Como o software foi escrito em C, o mesmo pode ser utilizado em qualquer microcontrolador da família ARM, ou seja, basta que se tenha acesso a um microcontolador com mais memória RAM que o software deve funcionar como funcionou na plataforma PC. Infelizmente não se teve acesso a um kit com um microcontrolador com mais memória, devido ao alto custo desses kits.

Futuramente deve-se testar o programa em um kit com mais memória, fazendo-se as devidas adaptações. Como trabalho futuro pode-se citar também o trabalho no código do elaborado neste trabalho, de forma a otimizar a utilização de memória, para que o mesmo funcione no kit utilizado nesse projeto. Em seguida deve-se implementar a técnica e o kit utilizado em aplicações de automação residencial.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]Huang, Xuedong; Acero, Alex; Hon, Hsiao-Wuen. **SPOKEN LANGUAGE PROCESSING**. Prentice-Hall, 2001.

[2]**AT91SAM7S-EK Evaluation Board User Guide**. Atmel. Disponível em: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3784

[3]**AT91SAM7S256 Datasheet**. Atmel. Disponível em: http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

[4]**ARM7TDMI Technical Reference Manual**. Disponível em: http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

[5]Site do pacote **WinArm**. Disponível em: http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/

[6]Site da empresa **Farnell-Newark**. Disponível em: http://www.farnell.com.br

[7]Zelenovsky, Ricardo; Mendonça, Alexandre. **Microcontroladores**. Editora MZ, 2005.