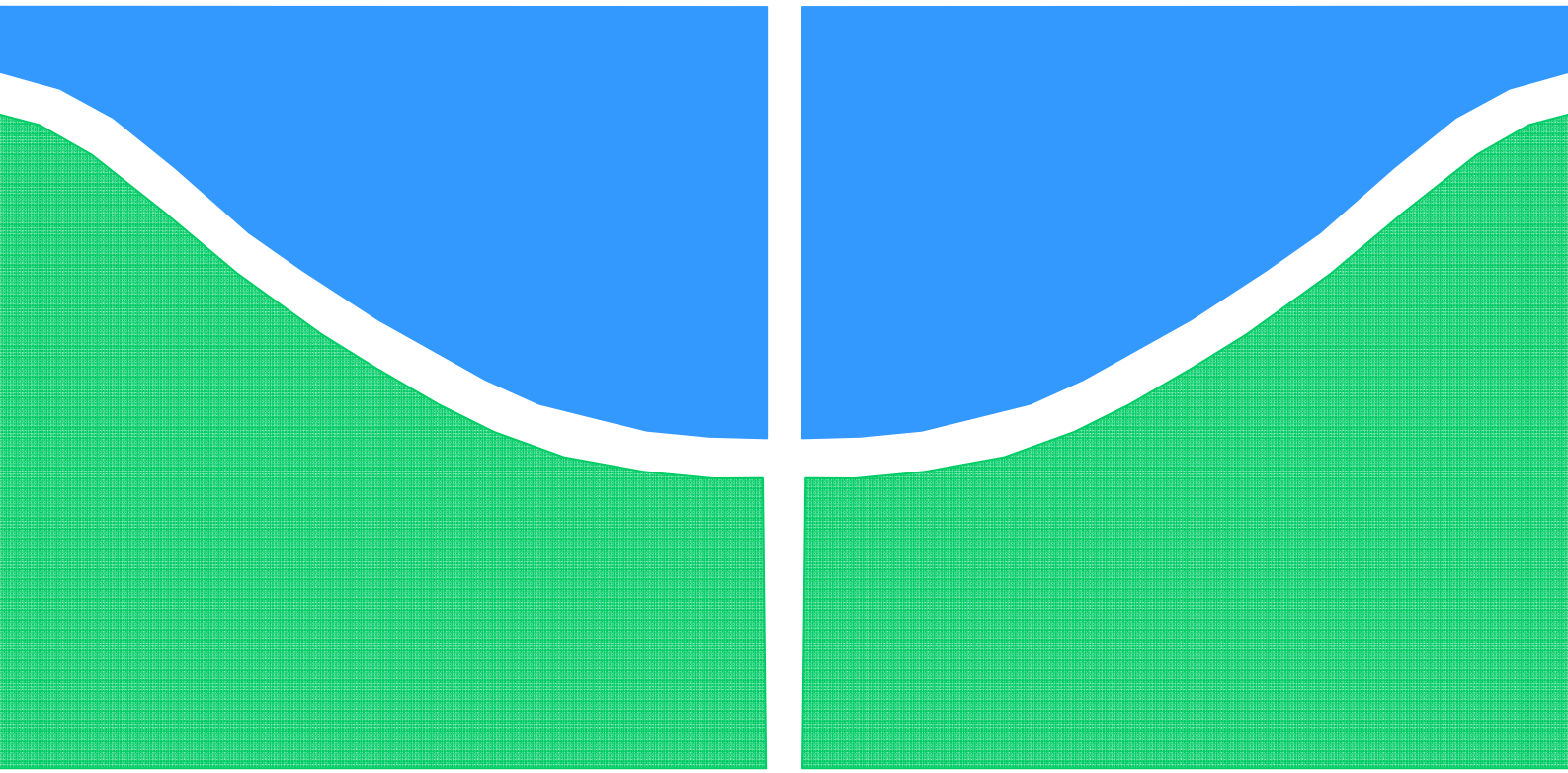

TRABALHO DE GRADUAÇÃO

Análise do desempenho de codecs de voz usando uma plataforma embarcada com processador ARM e software livre.

Bruno Lyra Gollo
Júlio Maurício Pinho Ribeiro Júnior

Brasília,20 dezembro de 2007

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Análise do desempenho de codecs de voz usando uma plataforma embarcada com processador ARM e software livre.

Bruno Lyra Gollo
Júlio Maurício Pinho Ribeiro Júnior

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

Banca Examinadora

Prof. Doutor Francisco Assis de Oliveira Nascimento
UnB/ENE (Orientador)

Prof. Doutor Pedro de Azevedo Berger (Co-Orientador) UnB/CIC

Tiago Alves da Fonseca, UnB/ Ene

Edson Mintsu Hung, UnB/ENE	

Dedicatória(s)

Dedico este trabalho à minha mãe, à minha irmã, por todo o apoio e carinho durante minha vida acadêmica, e aos professores que contribuíram no meu desenvolvimento acadêmico e humano.

Júlio Maurício Pinho Ribeiro Júnior

Dedico este trabalho à minha mãe que nunca mediu esforços para me proporcionar uma educação de boa qualidade. Ao meu pai, e aos meus irmãos por todo o carinho demonstrado durante a trajetória de realização desse trabalho.

Bruno Lyra Gollo

Agradecimentos

Agradeço à minha mãe por todos os esforços para me garantir uma educação de qualidade, ao meu pai por todos os anos de aprendizado. Agradeço aos meus irmãos e ao Tadeu por todos esses anos de convívio e crescimento pessoal, à Nathália, por toda compreensão e companheirismo. Agradeço ao Júlio Maurício, grande parceiro neste trabalho, aos Professores Francisco Assis e Pedro Berger co-orientadores neste trabalho, ao Edson Mintsu e ao Tiago Alves pelos momentos de ajuda no GPDS, ao Rafael Ortis por toda a atenção para proporcionar um laboratório adequado para o nosso trabalho e a todos os colegas da Universidade de Brasília, que compartilharam tantos bons momentos.

Bruno Lyra Gollo

Agradeço a minha mãe por todo esforço feito, mesmo com as dificuldades que a vida nos colocou, em me fornecer uma educação de qualidade, me proporcionando a obtenção desse diploma de Engenharia Elétrica. Aos amigos de curso, que me ajudaram diversas vezes durante todo curso. Aos amigos de fora da faculdade por me proporcionar momentos de diversão. Ao meu parceiro de projeto final Bruno Lyra, por toda amizade, dedicação e bom humor nesse trabalho realizado. E por fim agradeço ao orientador Francisco Assis, e principalmente, ao co-orientador Pedro Berger por toda ajuda e esforço para que esse trabalho se realizasse.

Júlio Maurício Pinho Ribeiro Júnior

RESUMO

Este trabalho teve como objetivo estudar o desempenho de codificadores voz em uma plataforma de desenvolvimento que utiliza arquitetura ARM ® [4] (i.MX21 da Freescale ®) com Linux embarcado. O processo de desenvolvimento utilizou a ferramenta LTIB ® em um computador *host* (com arquitetura x86) para criar uma distribuição Linux que foi utilizada na plataforma de desenvolvimento. Uma vez que o ambiente de desenvolvimento foi configurado, foram realizados testes de diversos codificadores de voz, avaliando as suas reais possibilidades de uso em tempo real para aplicações VoIP [1].

ABSTRACT

This project intents to study the performance of voice codecs in a development platform that uses ARM ™ [4] architecture, Freescale ™ i.MX21, with embedded Linux. The development process used the tool LTIB on a host computer (with architecture x86) to create a Linux distribution used in the development platform. Once the development environment was set up, tests were made on various audio codecs, evaluating their potential for use in real-time for VoIP[1] applications.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Neste capítulo introdutório há uma explanação da nossa motivação para a realização do trabalho, e descrição sucinta dos nossos objetivos do trabalho e do meio de realizá-lo.

1.1 INTRODUÇÃO

A motivação inicial deste trabalho, foi aprender como funciona todo processo de comunicação utilizando a tecnologia VoIP [1]. Isso porque essa tecnologia tem grande potencial para ser implementada em grande escala na comunicação mundial, por diminuir custos de forma significativa, e conseguir manter uma qualidade aceitável para o seu propósito. Nesse contexto, pretendemos realizar um estudo focado em sistemas embarcados, onde há certas condições de contorno que impõe certas restrições, como, por exemplo, o consumo de energia, que deve ser extremamente reduzido.

Neste estudo de caso, pretende-se testar diversos codecs de voz e avaliar quais deles poderá ser utilizado em um sistema de comunicação VoIP. Utilizamos a plataforma de desenvolvimento i.MX21ADS ® [2] da Freescale ® [3], que é dotado de um processador com arquitetura ARM® [4]. Os resultados serão úteis para testar a viabilidade do sistema.

A arquitetura ARM teve êxito no mercado mundial por ser pioneira no oferecimento de um processador de 32-bit com baixa potência, o que são características desejáveis para estruturas portáteis. Atualmente, há uma enorme quantidade de telefones celulares e de PDA´s vendidos a cada ano que utiliza a arquitetura ARM (*Advanced RISC Machines*) [4], como exemplo, podemos citar o grande sucesso comercial iPhone ® da Apple® [5] isso nos trás uma boa perspectiva de estudar essa tecnologia, pois há uma crescente demanda.

Para o estudo de uma aplicação VoIP, é interessante, mas não essencial, que seu desenvolvimento seja baseado em aplicações que estejam disponíveis sob licenças de código aberto. Dessa forma, tem-se disponível os códigos-fonte podendo, inclusive ser alterados para se otimizar ou adaptar a uma determinada aplicação desse código. Os custos para a comercialização são reduzidos, na maioria dos casos, que podemos obter o software aberto de forma gratuita, havendo, portanto, uma diminuição de gastos com software de desenvolvimento e aplicativos para o usuário final.

1.2 OBJETIVO

O objetivo principal deste projeto é realizar testes com codificadores de voz em uma plataforma de desenvolvimento que utiliza arquitetura ARM (i.MX21 da Freescale ®) e sistema operacional Linux. Para atingir este objetivo, pretende-se também configurar um ambiente de desenvolvimento específico para aplicações VoIP e gerar uma distribuição Linux capaz de compilar, testar e executar diversos codificadores de voz.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho é dividido em 7 capítulos. No capítulo 1 apresenta-se uma motivação para o trabalho e mostra o que fora usado no trabalho. No capítulo 2, tem-se uma noção e a apresentação de alguns tipos de licença de código aberto.

O capítulo 3 trata dos codificadores avaliados no trabalho, dando uma breve explicação sobre esses codificadores.

O capítulo 4 apresenta a plataforma de desenvolvimento mostrando alguns detalhes do hardware utilizado.

O capítulo 5 mostra o processo de criação da imagem linux utilizada nesse trabalho.

O capítulo 6 mostra o procedimento adotado para a avaliação dos codecs na plataforma utilizada, e explica os resultados obtidos na avaliação feita neste trabalho.

O capítulo 7 é a conclusão do trabalho.

O trabalho é dotado, também, de dois anexos, sendo que o primeiro trata na forma de um da apresentação do processo de criação da imagem descrito no capítulo 5, e o segundo anexo mostra os pacotes contidos na ferramenta de criação da imagem utilizada (LTIB).

2 TIPOS DE LICENÇA

Capítulo destinado para descrição dos tipos de licenças dos pacotes utilizados no trabalho.

2.1 SOFTWARE LIVRE

A expressão *software* livre se refere à liberdade do usuário executar, copiar, distribuir, estudar, modificar e aperfeiçoar o *software*. Sendo assim, ele se refere aos quatro tipos de liberdade, para os usuários do software, definidas pela *Free Software Foundation* nos anos 80 ainda, que são as seguintes: [6]

1. A liberdade de executar o programa, para qualquer propósito;
2. A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades, sendo o acesso ao código-fonte um pré-requisito essencial para esta liberdade;
3. A liberdade de redistribuir cópias de modo que você possa ajudar outras pessoas;
4. A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie com isso, sendo portando o acesso ao código-fonte é um pré-requisito para esta liberdade.

Um programa é considerado um software livre se os usuários possuem todas essas liberdades, portanto o usuário tem todo direito de modificar o software e distribuir ele privativamente sem que seja necessário pedir nenhum tipo de permissão.

A liberdade de usar o programa significa que qualquer pessoa, pode utilizar o software independentemente do sistema operacional que utilize, em qualquer tipo de computador, para qualquer tipo de trabalho que o software seja apto a realizar, sem que para isso seja necessário qualquer tipo de comunicação feita ao desenvolvedor do software.

A terceira liberdade para redistribuir cópias, junto à segunda liberdade de termos acesso ao código fonte, e, portanto tendo liberdade de modificá-lo possuem juntas, características importantes para que se melhore cada vez mais a qualidade dos softwares livres e para que cada vez mais a comunidade se beneficie em um ciclo virtuoso de melhorias.

A quarta liberdade engloba toda filosofia do software livre, de modo que a liberdade de fazer modificações, e de publicar versões aperfeiçoadas, tenha algum significado, deve-se ter acesso ao código-fonte do programa. Para que as liberdades sejam reais, elas têm que ser irrevogáveis se nada de errado for feito, caso o desenvolvedor de certo software tenha o poder de revogar a licença do usuário, com motivo ou não, o software não será considerado livre.

Não se pode confundir conceitos de liberdade com o de ser comercial ou não, um indivíduo pode ter pagado por alguma cópia de um software GNU ou apenas adquirido de forma gratuita o software, sendo que independentemente do modo de ter obtido a sua cópia, você sempre terá liberdade de copiar e modificar o software, podendo até mesmo vende-las se você assim desejar, mas sempre respeitando as liberdades de um software livre.

O desenvolvedor de um software possui algumas possibilidades de impor regras às pessoas que por ventura utilizarem seu código fonte e o redistribuírem, mas nunca poderão limitar sua liberdade, um exemplo disso é quando o desenvolvedor pede na licença do seu software, que caso esse programa seja modificado e redistribuído, ele receba uma cópia do mesmo.

No projeto GNU, há o uso de *copyleft* para proteger estas liberdades legalmente para todos. Um software livre que não possuir o *copyleft* permitirá que um indivíduo não mantenha a política de um software livre, e restrinja o código fonte de seu novo software baseado em um código livre sem *copyleft*. [6]

2.2 LICENÇA GPL

O GPL (GNU *General Public License*) é uma forma de distribuição em que o código fonte está disponível (copyright GPL) nos termos da *Free Software Foundation*. Uma licença serve para proteger o seu código quando ele for lançado para o público. A licença GPL especificamente permite que o autor do código distribua livremente o seu código, sendo que outras pessoas terão acesso a esse código utilizado, e poderão modificar dentro das suas necessidades e usá-lo livremente. O único requerimento que a licença GPL impõe é que os códigos derivados de um primeiro que utilizava a licença GPL também permitam o acesso direto ao seu código fonte, é o chamado *copyleft*. Tal licença foi feita para manter esse espírito de cooperação que existe entre os desenvolvedores de programas em linux, para que haja cada vez mais pessoas envolvidas no mesmo projeto e assim melhorando-o cada vez mais. Sendo assim a licença GPL é aquela tão idealizada no projeto GNU para um software livre, ou seja, aqui haverá necessariamente todas as 4 liberdades já expostas para o software livre, e também haverá sempre a perpetuação desse processo. Tem-se que prestar atenção ao fato de que nem todo software livre terá a licença GPL, mas todos que possuírem a licença GPL será necessariamente um software livre.[7]

É com essa licença que o kernel do Linux é liberado, há contribuições de diversas cabeças que obtêm livremente o código-fonte do kernel e procuram *bugs* e tentam resolvê-los, ou até mesmo desenvolver novas implementações ao kernel, aumentando sempre a praticidade e a confiabilidade do software em questão, dando continuidade a cultura que tanto é idealizada no projeto GNU na comunidade de desenvolvedores.

2.3 A LICENÇA LGPL

A LGPL - GNU *Lesser General Public License* - é uma variação da licença GPL e a sua principal diferença é que ela permite o desenvolvimento de programas de código aberto com módulos proprietários ou de software livre genérico, sem que sejam GPL. [8]

O código de um software que esteja na licença GNU GPL tradicional será sempre aberto, o que atende à diversos projetos colaborativos em que o auxílio mútuo é mais importante do que qualquer outro fator. Mas, como nada é perfeito, existem outros interesses que devem ser respeitados, como os segredos de algum software proprietário que não queira ser revelado, mas que também queira participar do projeto GNU se conectando de diversas formas com outros softwares GPL ou LGPL.

A licença LGPL fornece ao software uma característica de ser semi aberto, o que impede que um usuário qualquer tenha acesso aos códigos fonte da parte que se deseja que não haja acesso, e possibilita haver acesso a outra parte do código fonte. Para realizar esse processo esconde-se parte do código fonte disponibilizando-o em forma binária para que não haja acesso direto ao código, e os binários são recebem um *link* para o programa no momento da compilação.

De acordo com Richard Stallman em 1999 o idealizador do projeto GNU:

"Que licença é melhor para dada biblioteca é uma questão de estratégia, e depende dos detalhes da situação. No momento, a maioria das bibliotecas GNU são cobertas pela LGPL, e isso significa que estão usando apenas uma das duas estratégias (permitindo/não permitindo programas proprietários de usarem uma biblioteca), negligenciando a outra. Então estamos buscando mais bibliotecas a lançar sob a GPL comum.".

A inclusão de uma página de agradecimentos é opcional. Na mesma o autor pode expressar seus agradecimentos a pessoas ou entidades que de alguma forma deram contribuição relevante ao trabalho. Quando tal se aplica, recomenda-se que os agradecimentos sejam feitos de forma sucinta.

2.4 A LICENÇA BSD

A licença BSD impõe pequenas restrições e é considerada uma licença intermediária de restrições, ela é chamada de *copycenter* fazendo alusão ao *copyright* padrão e o *copyleft* da licença GPL. A sua principal característica é que os autores originais devem ser mantidos, mas não estabelece outras limitações para o uso do código. Ao desenvolver uma versão comercial de um programa sob esta licença, não se tem nenhuma obrigação de disponibilizar o código fonte como no caso da licença GPL, o que o torna mais comercial de certa forma. [9]

É uma licença de código aberto criada na Universidade de Berkeley, que possui compatibilidade com diversas licenças proprietárias. Ela permite que o software distribuído sob a

licença, seja incorporado a produtos proprietários. Trabalhos baseados no material podem ser até liberados com licença proprietária.

2.5 LICENÇA MIT

A licença MIT é uma licença de software criada pelo *Massachusetts Institute of Technology*. Ela é uma licença não *copyleft* usada em software livre, ela é considerada um licença *copycenter*, pois ela permite a reutilização do software licenciado em programas livres ou proprietários.[10]

Ela se assemelha muito à licença BSD, a diferença reside no fato de a licença BSD conter um aviso proibindo o uso promocional do nome dos autores do código, quando este for fechado sob copyright para uso comercial.

2.6 MOTIVAÇÂO PARA O USO DE SOFTWARE LIVRE

Ao se utilizar softwares livres no desenvolvimento do projeto, aumenta-se, na maioria dos casos, a sua viabilidade comercial, uma vez que há uma diminuição real de gastos com pagamentos de direitos de uso e comercialização de softwares proprietários.

Por possuir o código fonte aberto há a possibilidade de se começar o desenvolvimento de um projeto novo trabalhando com base em algum outro código fonte desenvolvido por outro programador. Isso é uma característica importante que torna mais acelerado o processo de desenvolvimento de softwares a partir de softwares livre. Um exemplo disso é o uso de diversas bibliotecas já existentes para desenvolvimento de diversos aplicativos novos.

O fato de termos acesso ao código fonte também é um fator que nos possibilita um maior dinamismo, não ficamos tão dependentes de quem construiu o programa, mesmo que não tenhamos o manejo para depurar o código em questão. Deste modo, fica-se mais fácil conseguir suporte com um programador qualquer que se comprometa a ajudar no projeto em questão.

Ao se analisar o uso de software livre em uma visão macro, observando sua importância imediata e futura para o Brasil, chegamos a conclusão que o projeto GNU deveria ser visto com bons olhos, pois é uma ótima saída para se libertar da dependência de softwares proprietários de outros países, deste modo poderíamos ser cada vez mais competitivos não apenas no ramo de desenvolvimento de softwares, mas também em outras áreas que possuem custos associados aos pagamentos de direitos de softwares proprietários.

3 CODECS

Explanações sobre a forma como os codecs processam as informações.

3.1 CODIFICADORES DE VOZ

Os sinais de voz são sinais analógicos, e que para serem processados e transmitidos, utilizam geralmente alguma forma de codificação, havendo uma digitalização do sinal inicial.

Para transmissão de voz via rede IP, é desejável que o codificador de voz utilizado trabalhe com baixas taxas de bits. Além disso, dependendo das características de como é realizada a codificação e a decodificação, a quantidade quadros do sinal de voz em cada pacote transmitido pode variar. Assim, o codificador tem que possuir mecanismos que minimizem os efeitos prejudiciais à transmissão que são inerentes às redes de comunicação de pacotes como: atrasos e perdas de pacotes, ou erros inseridos no fluxo transmitido.

A codificação da voz na sua transmissão esta grande parte relacionada com a necessidade de haver uma compressão do sinal, havendo uma diminuição significativa na taxa de bits a serem transmitidos. Existem 3 tipos genéricos de codificadores: os codificadores de forma de onda, os paramétricos e os híbridos.

Codificadores de forma de onda têm como princípio a codificação do sinal baseando-se na forma de onda do sinal analógico, esse processo de codificação mantém as características temporais e espectrais do sinal original. Como exemplo, temos os modelos de codificação PCM, do G.711, e ADPCM, do G.726, por exemplo.

Os codificadores de fonte ou paramétricos são baseados em um modelo da produção da voz para, a partir desse modelo, determinar padrões que são normalmente repetidos em um sinal de voz.

Esses padrões são estabelecidos com base em algum padrão de parâmetros que, de acordo com algum modelo, estabelece-se que esse padrão se repete. Por exemplo, formas de onda básicas que se repetem com certa freqüência em sinal de voz. Então, cria-se uma base de dados contendo um conjunto de parâmetros como as formas de onda que são padrões do sinal de voz que se repetem. Essa base de dados deve conter uma quantidade de formas de onda básicas suficiente e devidamente escolhidas para que seja possível representar um sinal de voz.

Dessa base de dados, cria-se um dicionário que relaciona cada termo dessa base de dados a um índice simplificado que representa os termos em questão de forma simplificada. Desse dicionário extraem-se do sinal original apenas os índices que se referem a cada forma de onda da base de dados. Esse modelo é dito de análise e síntese, por que primeiro quebra-se o sinal em termos menores que já estão contidos na base de dados, obtendo como resultado da codificação apenas os índices para essa base de dados. Para o processo de decodificação, utilizam-se os índices obtidos no processo de análise

para buscar no dicionário os termos adequados para sintetizar um sinal de voz. Este sinal sintetizado deve ser interpretado como sendo igual ao sinal original acrescido dos ruídos provenientes das aproximações feitas nos processos de codificação e decodificação.

Têm-se como características dos codificadores de forma de onda, taxas de bits por segundo resultantes do processo de codificação relativamente altas, e uma boa qualidade do sinal resultante. Já os codificadores paramétricos têm taxas de bits por segundo relativamente baixas com a penalidade de um sinal resultante de qualidade não tão boa quanto à dos codificadores de forma de onda.

Para resolver este problema, o que é utilizado é o codificador híbrido, que junta características de ambos os codificadores citados acima. Dessa maneira, obtêm-se codificadores com taxas de bits por segundo relativamente baixas e qualidade de áudio relativamente boa. Um exemplo para esse tipo de codificador é o CELP (*Code Excited Linear Prediction*).

3.1.1 G.711

O G.711 é um codificador de áudio com padrão ITU-T que possui uma elevada taxa de transmissão (64 Kbps). Ele é considerado a língua nativa da rede telefônica digital moderna. Ele é um codificador de forma de onda, logo ele trabalha estritamente na codificação e decodificação da forma de onda original do sinal, sendo esse o motivo da alta taxa de transmissão necessária.

O princípio do codificador G.711 é a utilização de uma escala logarítmica na quantização para se obter uma SNR (*Signal-noise ratio*) independente da intensidade. Isso é possível duplicando a quantização cada vez que a intensidade do sinal for duplicada, mantendo constante assim sua SNR. A quantificação mais simples do sinal de voz, seria fazê-lo de forma linear, mas ocorre um problema crítico, a sua SNR varia com a amplitude do sinal de entrada, portanto, para pequenas amplitudes, pequenas SNR. Ao olharmos a qualidade do sinal, e sabendo que o sinal tem uma alta variabilidade no tempo devido aos problemas de transmissão, como perda de pacotes teremos que sua SNR também variará, resultando em um sistema totalmente variável, o que não é desejado.[11]

O codec G.711, embora formalmente normalizada em 1988, é o mais utilizado na telefonia digital. Inventado pela Bell Systems e introduzido no início dos anos 70´s, ele empregava um uso de 8 bits descompactados por modulação de pulsos, o PCM, a uma taxa de amostragem de 8000 amostras por segundo, mantendo a largura de banda entre 300 e 3400 Hz, desse modo cada canal de voz precisa de 64 kbps na sua transmissão.[12]

Para evitar este problema é utilizada a quantização logarítmica que fornece como resultado uma quantificação uniforme de ruído, isto permite a definição de duas maneiras de transmissão utilizando a lei de compressão. Difundiram-se pelo mundo 2 tipos dessa quantização logarítmica, a Lei-μ e a Lei-A.

Lei-μ é utilizada nos EUA e no Japão, e é originária do padrão inicial desenvolvida em 1988.

A versão da Lei-A foi desenvolvida na Europa e espalhada para o resto do mundo, com exceção dos países que utilizam a versão da Lei-μ.

A diferença entre ambas está no método de como o sinal analógico é amostrado. Em ambos os regimes, o sinal não é amostrado linearmente, mas sim através de funções logarítmicas. A versão da Lei-A nos fornece mais dinâmica no processo em relação à versão da Lei-μ. O resultado é um som melhor, nas amostras que são suprimidas.

As equações são:

Lei μ:

$$y = \frac{\ln(1+\mu x)}{\ln(1+\mu)}, \text{ sendo } \mu = 255 \qquad (1)$$

Lei A:

$$y = \frac{Ax}{(1+\ln A)}, \text{ para } x \leq 1/A, \text{ sendo A=87,6} \qquad (2)$$

$$y = \frac{(1+\ln Ax)}{(1+\ln A)}, \text{ para } 1/A \leq x \leq 1 \qquad (3)$$

A sua qualidade do som transmitido é tão boa que se assemelha a um telefone analógico convencional.

3.1.2 G.726

G.726 é um codec de conversação ITU-T que opera em taxas de 16, 24,32 e 40 kbits/s, que corresponde utilizarmos 2, 3, 4 e 5 bits/amostra respectivamente. O padrão é a utilização de 32 kbits/s, que seria correspondente ao antigo G.721, e as taxas de 24 e 40 kbits/s referentes ao antigo G.723, sendo, portanto o G.726, apenas uma unificação desses padrões já existentes que utilizavam técnicas similares.

É utilizada na codificação do G.726 a codificação ADPCM, que difere da codificação PCM, pois ele ao invés de quantizar a forma de onda diretamente, ele codifica a diferença entre os sinal de voz em questão e a sua predição feita. Isso é possível porque as amostras de sinal de voz são dependentes, e há usualmente uma considerável redundância nas amostras. [13]

Sendo a predição feita do sinal de entrada feita com um grau de precisão boa, a diferença entre os sinal previsto e o original são pequenas, sendo assim pode-se realizar uma codificação do sinal com menos bits, aproveitando as redundâncias existentes. Podemos ver na Figura 3.1 um esquemático de codificação e decodificação do G.726.

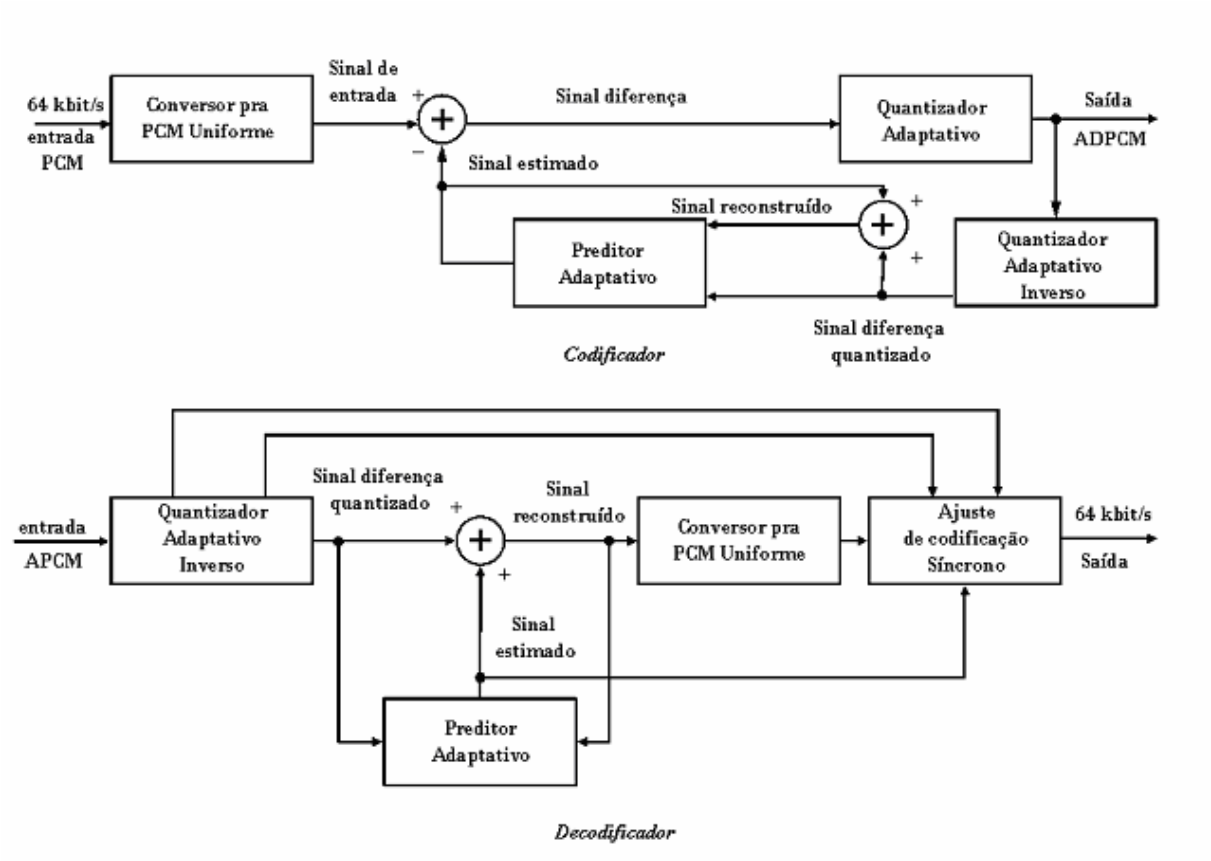

Figura 3.1: Diagramas de blocos do codificador e do decodificador G.726

Quando passa pelo decodificador os sinais de diferenças são somados aos sinais previstos. Na codificação do sinal há uma quantização chamada de adaptativa, ou seja, o preditor e o quantizador de diferenças adaptam-se às características variáveis do sinal que está sendo codificado.

3.1.3 G.729

O codificador G.729a é um codificador baseado no modelo CS-ACELP. Ele divide o sinal de voz em blocos de 10 ms. Cada bloco é representado por 80 bits, distribuídos da forma expressa na Tabela 3.1 abaixo.[14]

Tabela 3.1 Distribuição dos bits de saída do G.729 [15]

Parâmetro	Quantidade de bits			
	Palavra código	Subquadro 1	Subquadro 2	Total por quadro
Parâmetros LFS	L0, L1, L2, L3			**18**
Atraso tonal (dicionário adaptativo)	P1,P2	8	5	**13**
Bit de paridade	P0	1		**1**
Índice do dicionário fixo	C1,C2	13	13	**26**
Sinal dos pulsos do dicionário fixo	S1, S2	4	4	**8**
Ganho dos dicionários (estágio 1)	GA_1, GA_2	3	3	**6**
Ganho dos dicionários (estágio 2)	GB_1, GB_2	4	4	**8**
Total				**80**

O processo de codificação toma quadros de 10 ms de sinal de áudio, divide-os em dois subquadros, para cada subquadro, extrai os parâmetros de predição linear (LP),que são os índices do filtro de predição linear. Converte os parâmetros do filtro de predição em parâmetros LSF (L0, L1, L2 e L3), que representa parte da saída do processo. Verifica o atraso tonal (P1 e P2) para localização das seqüências do dicionário adaptativo. Verifica-se a paridade dos bits mais significativos do atraso tonal (P0). Os índices do dicionário fixo (C1 e C2) representam as posições dos pulsos i_0, i_1, i_2e i_3.[14] E cada amplitude (S1 e S2) dos pulsos é codificada em 1 bit. Os índices obtidos pelos dicionários podem ser quantificados, e o resultado dessa quantificação esta nas palavras GA_1, GA_2, GB_1, GB_2.

A Figura 1 abaixo mostra o processo de codificação para a obtenção das palavras de saída do processo, tal como discutidas acima.

A Figura 2 abaixo ilustra como se dá o processo de decodificação. Os índices dos dicionários buscam o sinal ao qual eles se referem, dá-se o ganho indicado a cada sinal obtém-se um primeiro sinal sintetizado, que é então passado filtros de curto termo que reduzem o erro desse sinal com relação ao sinal original.

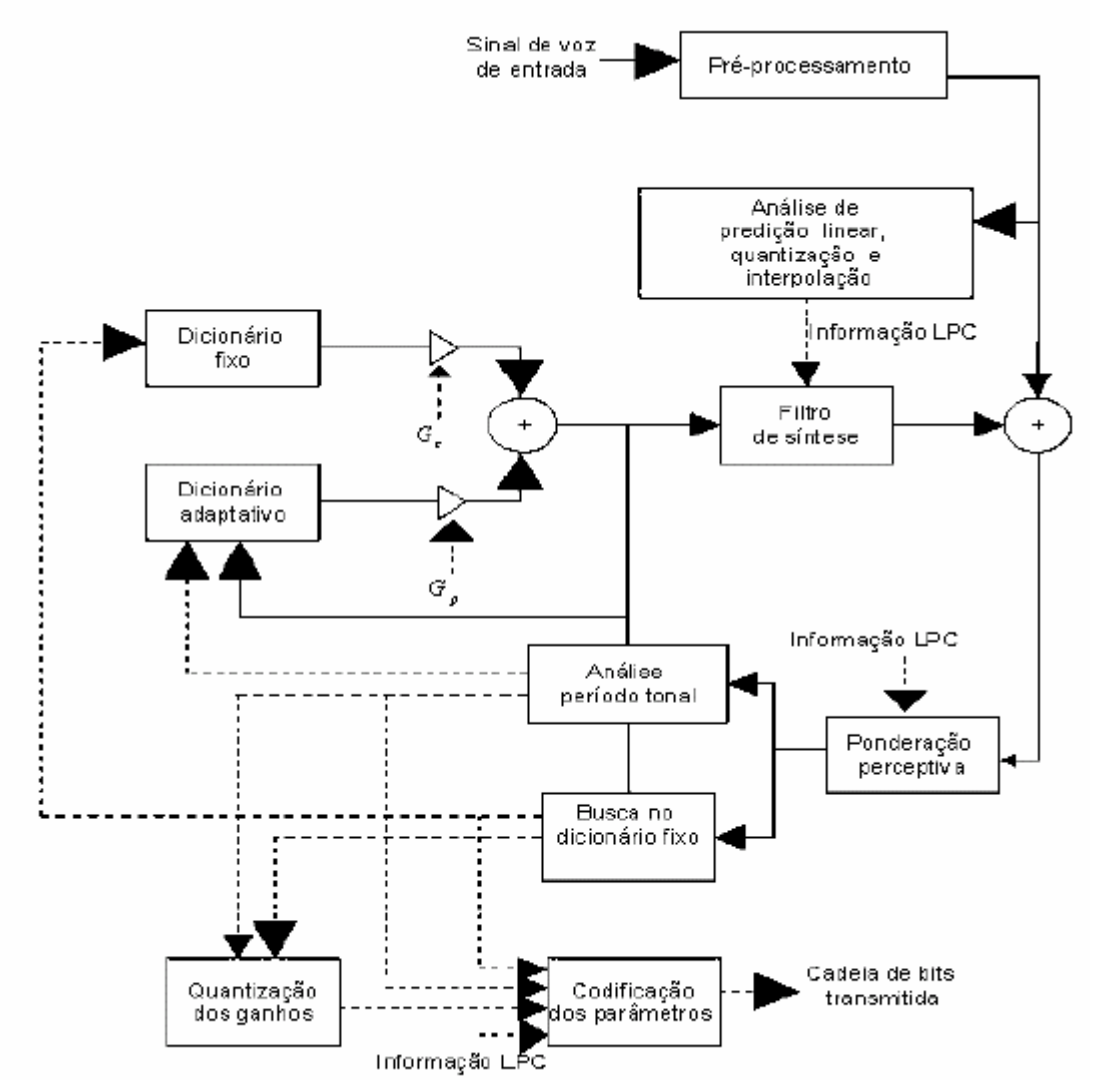

Figura 3.2: Diagrama de blocos codificador G.729 [16]

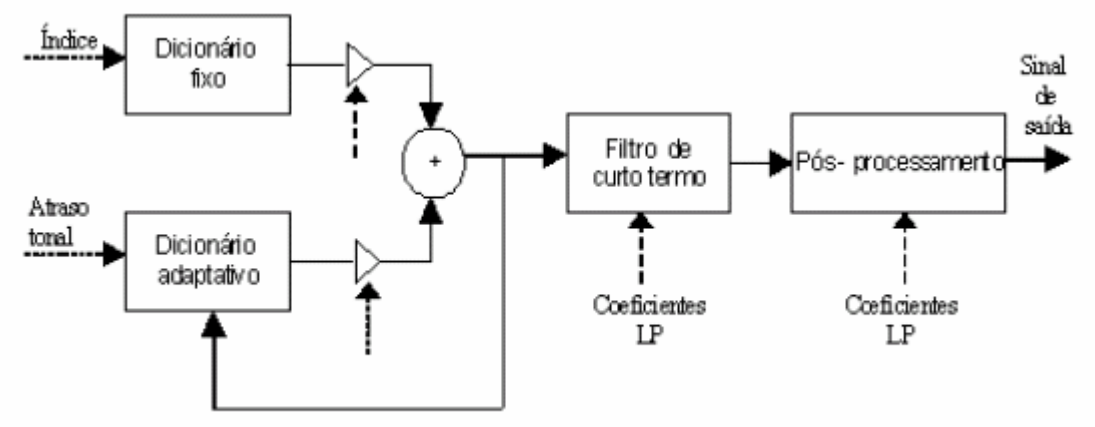

Figura 3.3: Diagrama de blocos do decodificador G.729 [16]

4 PLATAFORMA DE DESENVOLVIMENTO

Especificações e comentário sobre a plataforma de desenvolvimento onde realizamos os testes propostos.

4.1 PLATAFORMA DE DESENVOLVIMENTO

A plataforma de desenvolvimento utilizada é a mostrada na Figura 4.1 abaixo. Esta plataforma contém 3 conectores DB-9 que funcionam com protocolo RS-232, sendo dois deles DCE (P1 e P3) e um deles DTE (P2). Sendo, ainda, 2 desses conectores com interface ao processador (P1 e P2) e um com interface externa ao processador (P3)[17]. Contém 3 conectores de áudio, para saída estéreo, uma entrada de nível de microfone e uma entrada de nível de linha. Também é dotada de conectores para teclado de 36 botões, para sensor CMOS de imagem e para um monitor LCD. Uma porta para extensão de memória, para memórias SD ou MMC.[18]

Na placa principal, é conectada a placa do processador. Nesta placa, estão contidos o processador i.MX.21®, memórias SDRAM (U6 e U8) e memórias NOR (U7 e U9), além dos conectores PM1 e PM2 para conectar uma memória NAND. O kit tem uma memória NAND de 512 MB para este conector.[18]

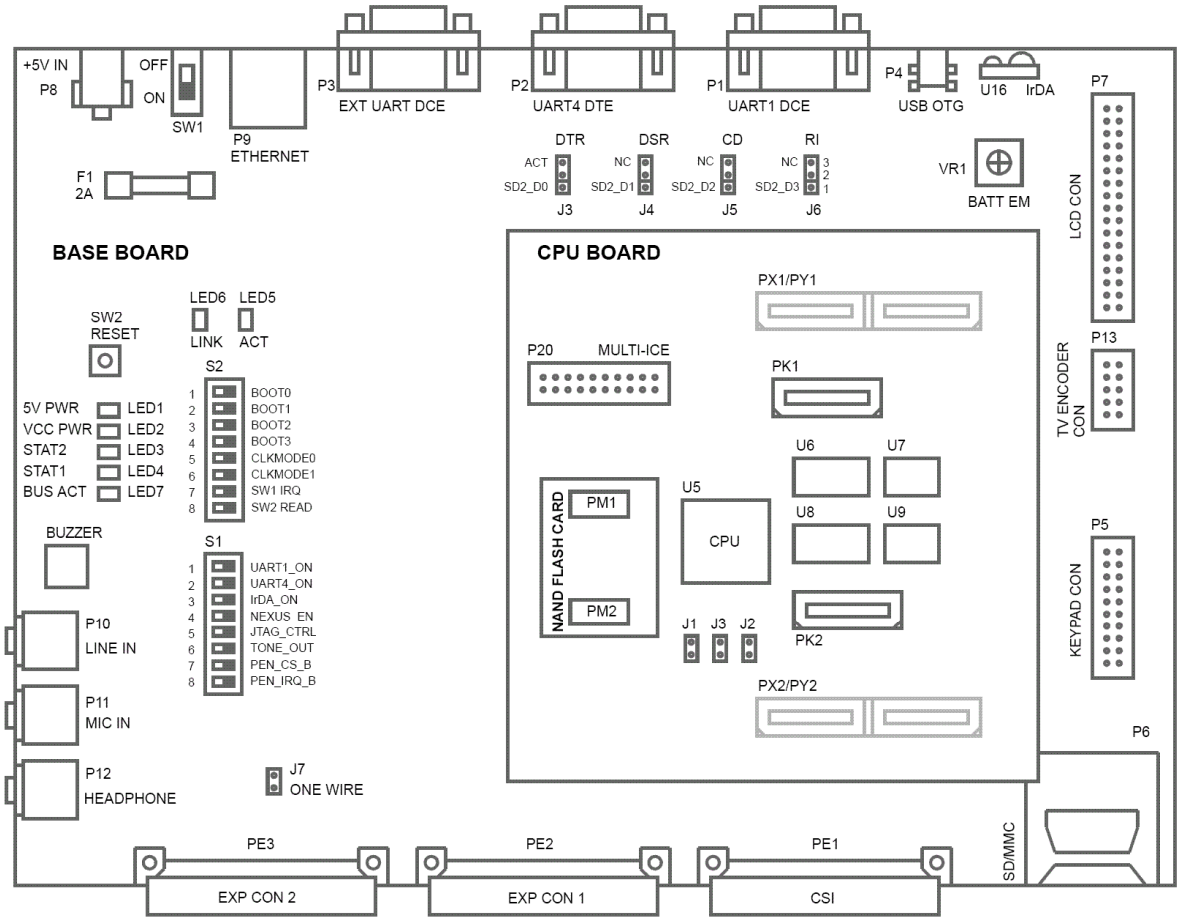

Figura 4.1 A placa de desenvolvimento da Freescale® [17]

4.2 PROCESSADOR

O processador da plataforma de desenvolvimento utilizada é o i.MX21 ® da Freescale®, com um núcleo ARM926EJ-S ® [19] (Figura 4.2), cuja freqüência do relógio interno é de 266MHz, e a freqüência do relógio do sistema, e também da memória externa, é de até 133MHz, esta obtida por divisão inteira da freqüência interna do relógio do processador [18]. Este microprocessador tem barramento de 32 bits, com instruções ARM de 32 bits ou instruções Thumb® [20] de 16 bits.

A extensão Thumb permite ao processador maior rendimento de desempenho para uma determinada densidade de código. Thumb usa, em um sistema de 32 bits, instruções de 16 bits. Armazena instruções de forma comprimida, em 16 bits. Permite performance de 32 bits usando dispositivos de memória de 8 ou 16 bits com barramentos de 8 ou 16 bits.

O microprocessador desse kit é dotado de memórias cache de instruções e de dados de 16KB cada. O chip do processador contém também uma unidade de processamento de vídeo MPEG4 e H.263, que pode ser usado em aplicações multimídia como videoconferência, por exemplo.

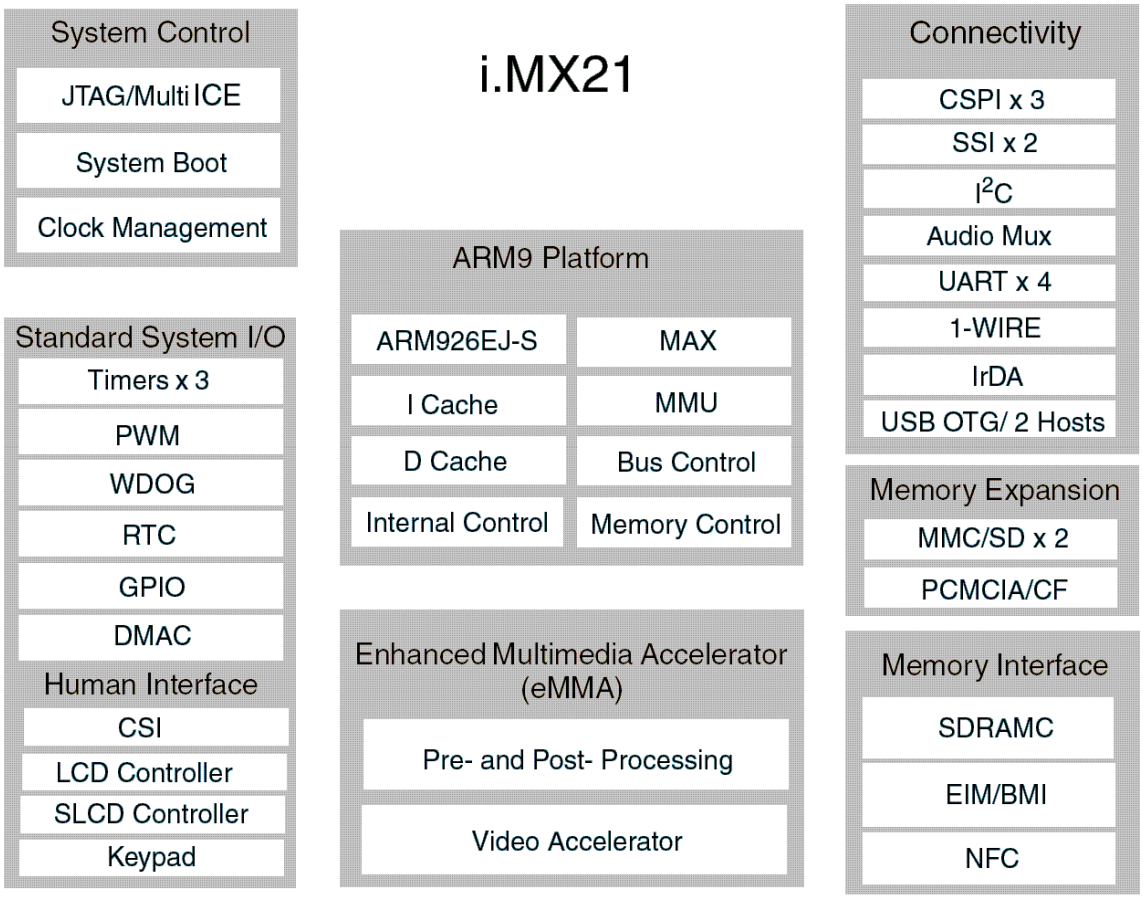

Figura 4.2: Esquema do processador i.MX21 [18]

4.3 MEMÓRIAS EXTERNAS

O kit contém uma memória *flash* NOR de 32 MB (são duas memórias de 8MB, 16 bits, configuradas de tal forma que seja usada como uma memória de 32 MB, e 64 MB [17]), *on-board*, um *slot* de memória *flash* NAND e uma memória deste tipo de 512 MB. São essas memórias *flash* os dispositivos de armazenamento do kit.

Pode-se traçar uma comparação entre memória NAND e memória NOR: [21]

- Ambas são memórias *flash*, não voláteis;

- As células de memória da NOR são maiores, por conseguinte, o chip de memória NOR é maior que um chip de memória NAND de mesma capacidade;

- A memória NOR é mais cara que a NAND;

- A leitura da memória NOR é mais rápida que leitura da NAND.

- A memória NOR não requer correção de dados, ao contrário da NAND, que para ser confiável necessita de um código de correção de erro;

- É possível ter o boot pela NOR e XIP (*execute in place* - execução de códigos armazenados nessa memória), ao contrário da NAND, que é uma memória de acesso em blocos.

- O core ARM não tem controlador de memória NAND, no entanto, o i.MX21 tem fora do core um controlador para este tipo de memória, é o NFC (NAND *Flash Controller*)

- Comparando com a NAND, a memória NOR é lenta para escrita.

A memória RAM do sistema é uma SDRAM de 64 MB, com freqüência de 133MHz, barramento de 16 ou de 32 bits. Sendo formada por duas memórias separadas de 16MB, 16 bits configuradas de tal forma a obter os 64 MB, 32bits. Com o controlador dessa memória, também no i.MX21, fora do core ARM, é o SDRAMC.

Outro controlador de memória contido no processador é o EIM (*External Module Interface*), que dá suporte a boot das memórias ROM, flash NAND e flash NOR. Também o BMI (*Bus Master Interface*), que possibilita interface entre o processador e os barramentos de acesso a memórias, usando o *clock* das memórias no processador.

4.4 PORTAS DE ACESSO

Para acessar a placa remotamente, podemos usar o USB-OTG (*Universal Serial Bus - On The Go*), rede IP, ou a interface serial RS-232. Esta interface serial é um conector DB-9 (P1), que usa o protocolo RS-232, UART com as seguintes configurações: [23]

- *baud rate* de 115200;

- dados de 8 bits;

- sem bit de paridade;

- um bit de parada e;

- sem nenhum bit de controle.

Interface de Rede RJ-45, com controlador CS8900A-CQ3Z ® da Cirrus Logic ®, que usa 10BASE-T, do protocolo 802.3, ou seja, padrão *Ethernet* a 10Mbps, em cabo de par trançado.

USB-OTG permite que os dispositivos que se relacionam possam ser tratados como periféricos ou como *host*, com isso, podemos ligar a plataforma de desenvolvimento a um PC (*Personal Computer)* e tratar a placa de desenvolvimento como um periférico, ou podemos ligar a um outro dispositivo que será o periférico.

As formas de acesso são via `minicom` (Linux) ou `hyperterminal` (Windows) pela porta serial (P1), que dá acesso ao kit, permitindo, inclusive usar o `blob` para carregar imagem e kernel na

memória *flash* NOR da placa. Via HAB, um programa disponibilizado pela freescale, para permitir descarregar nas memórias *flash* (NOR e NAND) do kit imagens ou kernel, via USB ou via RS-232.

Outra forma de acesso depende do sistema já estar carregado e executando, que é baseado em rede, com isso, podemos ter acesso ao sistema a partir de uma máquina remota usando programas como `telnet` ou `ssh`.

5 PROCESSO DE CRIAÇÃO DA IMAGEM

Este trabalho foi baseado em uma série de testes realizados numa plataforma de desenvolvimento, usando uma imagem Linux criada com auxílio da ferramenta LTIB. Nesse capítulo, descrevemos as características que necessitamos do host e da imagem criada. Os detalhes do processo de criação aparecem no Anexo 1 " Passo a passo da contrução da imagem – Tutorial".

5.1 PRÉ-REQUISITOS

Para a criação da imagem a ser usada na plataforma de desenvolvimento, é necessário cumprir alguns pré-requisitos do sistema *host*:

-*Interface* de acesso *Ethernet* ;

-Porta Serial com controladores para RS-232;

-1GB de espaço em disco;

-servidor `NFS` instalado;

-servidor `TFTP` instalado;

-`perl`;

-`rsync`;

-`ncurses-devel`;

-`zlib-devel`;

-`rpm-build`;

-`gettext`;

-`bison`.

Para facilitar a instalação da ferramenta `LTIB` e a compilação dos pacotes na criação da imagem Linux a ser usada no kit de desenvolvimento, uma boa opção é usar a distribuição Red-Hat 7.3® modificada e disponibilizada no site da Freescale® na forma de sistema a ser executado em uma máquina virtual VM-Ware ®.Essa distribuição já tem instalados a maioria dos pacotes necessários para o uso da ferramenta LTIB, ou seja, já vem pronta para a instalação da ferramenta e criação da imagem desejada.

Com a utilização da máquina virtual, temos uma segmentação das necessidades do *host*, assim divididas:

Requisitos do construtor da imagem, `LTIB`, que fora executado na máquina virtual.

-`perl`

```
-rsync
-ncurses-devel
-zlib-devel
-rpm-build
-gettext
-bison
```

Requisitos do *host* (a máquina que será o servidor do sistema de arquivos e servidor para mandar o *kernel* e *boot loader* para a placa da Freescale®).

-Interface de acesso Ethernet ;
-Porta Serial com controladores para RS-232;
-1GB de espaço em disco;
-servidor NFS instalado;
-servidor TFTP instalado;

5.2 A INSTALAÇÃO DO LTIB

O sistema a ser usado na VM-Ware, que usa uma distribuição Red-Hat 7.3 pede um *login*, que por padrão é:

```
login:vm_user
password:vm_user
```

para entrar com usuário comum , sem privilégios de super-usuário. Ou :

```
login: root
password:vm_root
```

para ter acesso como super usuário.

Para instalar ferramenta LTIB, precisamos acessar como usuário comum, deve-se copiar o LTIB para uma determinada pasta, e instalar esta ferramenta, executando o script de instalação.

5.3 A CRIAÇÃO DA IMAGEM

Para a criação da imagem, é necessário planejar previamente a imagem que se deseja obter. Para o estudo em questão, onde apenas desejamos realizar testes de viabilidade da plataforma de desenvolvimento para o uso de codecs de voz para comunicação em tempo real.

Para esses testes, é necessário ter um compilador que nos testes realizados, utilizou-se GCC. Para a compilação de maneira simplificada, utilizam-se os makefiles, para utilizá-los é necessário ter também o pacote make.

Para futuras aplicações, em tempo real, deverão ser adicionados à imagem também os pacotes do ALSA (*Advanced Linux Sound Architecture*) e deverá também ser instalado na placa o JACK (*Jack Audio Connection Kit*) .

Para melhorar o desempenho do sistema, uma sugestão para trabalhos futuros é habilitar também a memória NAND do sistema. Esta memória tem uma grande vantagem com relação a memória NOR no que diz respeito à velocidade de gravação de arquivos. O que tem um grande peso na análise deste trabalho, uma vez que, se o processo de gravação representar um grande atraso no processo, para uso da memória NOR, então o uso da memória NAND pode acarretar em melhorias significativas nos tempos de processamento de cada codificador.

No processo de criação da imagem, podemos também ajustar alguns parâmetros que serão gravados na memória. Com isso, todo *boot* terá já acertadas, por exemplo, as configurações de rede.

Para esse trabalho, foram implementadas duas imagens distintas, uma para ser usada com sistema de arquivos remoto, acessado via NFS [22] e outra com sistema de arquivos salvo na própria memória *flash* do sistema.

Houve a necessidade de criar essas duas imagens distintas por que o tamanho da imagem completa superou a capacidade de armazenamento da memória *flash* NOR do sistema. Sendo assim, no sistema de arquivos salvo na memória não havia, por exemplo, o GCC (*Gnu Compiler Collection*) ferramenta essencial para a compilação dos codecs utilizados neste trabalho.

Portanto, nesse trabalho, compilaram-se os códigos em C dos codecs quando a partida fora dada com sistema de arquivos acessado via NFS. Posteriormente, com partida feita com sistema de arquivos armazenado na memória *flash* NOR, os binários resultantes da compilação foram gravados nesta memória do sistema, usando TFTP para a transferência de arquivos entre o PC e a plataforma de desenvolvimento.

5.4 TRANSFERÊNCIA DA IMAGEM DA MÁQUINA VIRTUAL PARA O *HOST* (PC)

Usamos uma máquina virtual para a criação da imagem, mas esta máquina virtual não tem acesso a todos os dispositivos físicos da máquina, um desses dispositivos que a máquina virtual não tem acesso é a interface serial RS-232, que é acessada pelo conector DB-9 da máquina. Como não tem acesso a este dispositivo, deve-se transferir a imagem criada para o PC. O método escolhido para essa tarefa foi o uso do SFTP (*secure file transfer program*).

Para que o PC tenha permissão de acesso à maquina virtual via SFTP, é necessário que esse acesso seja permitido, por exemplo, pelo arquivo `/etc/hosts.allow` da máquina virtual, ou vice-versa.

O `SFTP` é um programa que nos permite visualizar as pastas do servidor bem como transferir arquivos contidos na pasta de trabalho do servidor para o cliente, dependendo das permissões que tem o usuário remoto.

Como discutido na secção anterior, temos que o sistema de arquivos pode ser armazenado na memória flash do sistema ou remotamente, no disco rígido do *host*. Para o primeiro caso, devemos disponibilizar o arquivo `rootfs.jffs2` para que seja armazenado na memória *flash*, para tal, devemos passar da máquina virtual para o PC e, então, do PC para a plataforma de desenvolvimento. Dessa forma, o sistema de arquivos poderá ser acessado diretamente pela memória *flash*.

De maneira alternativa, podemos transferir o sistema de arquivos da máquina virtual para o PC para que seja acessado remotamente pela placa. Para tal, devemos disponibilizar o diretório que contém o sistema de arquivos para que se possa ter acesso remoto, via `NFS`. Fazemos isso permitindo acesso remoto dos diretórios que devem ser disponibilizados no PC.

O procedimento para transferência desse sistema de arquivos a ser acessada remotamente passa pela compressão do tipo **tar**, ou qualquer outra forma de unir os arquivos de um determinado diretório em um único arquivo. Essa compressão é feita para facilitar o processo de transferência dos arquivos da máquina virtual para o PC.

5.5 CONFIGURAÇÃO DO MINICOM E INICIALIZAÇÃO DO TARGET

Para utilizar a placa, antes de tudo é necessário usar um *software* de acesso remoto que utilize a porta serial (dispositivo UART), esse software pode ser o `minicom` (para linux) ou o `hyperterminal` (para Windows®).

Neste trabalho, utilizou-se o linux no PC, portanto, utilizou-se o `minicom`.

É necessário configurar o `minicom` da mesma forma nas duas pontas da comunicação, portanto, utilizamos as configurações recomendadas pelo fabricante, com *baud rate* de 115200 por segundo, 8 bits de dados, nenhuma paridade, 1 bit de parada e sem nenhum controle de fluxo (“*no hardware flow control*” e “*no software flow control*”). Com o `minicom` já configurado, e configurações salvadas como *default*, para obter acesso às saídas do `shell` ou do `blob`, basta usar o `minicom`.

Com o `minicom` configurado, e executando, com a plataforma conectada ao PC pelo cabo serial, liga-se a placa de desenvolvimento, interrompe-se o *boot* nos primeiros segundos da placa ligada, pressionando a tecla <ENTER>. Com isso, apenas tem-se carregado o *boot loader*, que pode

ter algumas configurações alteradas, carregar novos arquivos para *kernel*, sistema de arquivos e até mesmo outro *boot loader*.

Com o *boot loader,* carregam-se para a plataforma de desenvolvimento os arquivos desejados. Então, teremos o *boot* como desejado, seja com acesso ao sistema de arquivos remotamente, seja com sistema de arquivos armazenado na memória *flash* do sistema.

5.6 O ARQUIVO PARAM

O arquivo `param` [23] configura parâmetros de rede, da interface serial do ponto de vista do *host,* e de `boot`,. `Param` é um arquivo binário gerado pela ferramenta `LTIB` usando o arquivo de configuração `blob.config.param.`

Criado o arquivo `param`, o copiamos para a pasta `/tftpboot/imx21ads`. No *boot loader* da plataforma de desenvolvimento da Freescale® (`blob`), via `minicom`, ajustam-se o IP da máquina, o IP do *host,* o diretório onde se encontra o sistema de arquivo, o *kernel* e o `blob` e então descarregamos o arquivo `param` necessário e o salvamos para a memória *flash*.

5.7 TFTP E FLASH

A cada vez que se deseja mudar algum dos arquivos (*kernel*, sistema de arquivos, *boot loader* ou arquivo de parâmetros), devemos apagar o setor de memória que ocupa no momento, para então, descarregar o arquivo e armazená-lo na memória *flash*.

Cada um desses arquivos ocupa endereços específicos de memória. Quando somente se sobrescreve pode-se deixar algum vestígio do arquivo anterior, se algum setor da memória não for sobrescrito. Isso pode gerar problemas quando a versão sobrescrita for utilizada.

Os setores da memória NOR onde se iniciam cada um dos arquivos são:

`ramdisk:0xc8300000`

`kernel: 0xc8100000`

`param: 0xc8010000`

`blob: 0xc8000000`

USB-OTG permite que os dispositivos que se relacionam possam ser tratados como periféricos ou como *host*, com isso, podemos ligar o kit de desenvolvimento a um PC e tratar o kit como um periférico, ou podemos ligar a outro dispositivo que será o periférico.

6 RESULTADOS EXPERIMENTAIS

Capítulo com objetivo de mostrar como foram realizados os testes dos codecs e especular a viabilidade do uso desses codecs na plataforma de desenvolviemento em estudo, quando aplicados em tempo real.

6.1 PROCEDIMENTOS

Realizaram-se testes utilizando a plataforma de desenvolvimento apresentada no capítulo 4, executando uma imagem Linux compilada na máquina virtual com distribuição RedHat 7.3, com auxílio da ferramenta `LTIB` (Linux *Target Image Builder*).

Para a realização desses testes, foram criadas duas imagens distintas, conforme discutido no Capítulo 5 uma com intuito de ter, no *boot*, o sistema de arquivos acessado via `NFS` (*Network File System*), outra para ter no *boot*, o sistema de arquivos armazenado na memória *flash* NOR.

Houve a necessidade de criar duas imagens distintas devido à escassez de memória *flash* para implementar o sistema de arquivos. Essa escassez se fez evidente quando percebeu-se a necessidade de criação da imagem com GCC para compilar os códigos já de forma adequada à nossa plataforma de desenvolvimento e que seria necessário, ainda, para um teste em tempo real, implementar as ferramentas ALSA e JACK. Portanto, antevendo essa necessidade, criou-se uma imagem maior que a capacidade da memória *flash*, usando boot pelo sistema de arquivos armazenados no HD do *host*, e acessados pela plataforma de desenvolvimento via `NFS`, sistema de arquivos este armazenado no diretório `/tftpboot/imx21ads`, como descrito no **Anexo 1.** A motivação para o uso do sistema `NFS` foi solucionar com facilidade o problema de falta de memória, no entanto, para uma aplicação comercial, que seria para um sistema portátil, não faz sentido o uso de `NFS`, portanto, sugerimos para trabalhos futuros o uso do cartão de memória SD (do inglês, *Secure Digital*) ou o uso da memória NAND que faz parte do kit de desenvolvimento. Neste trabalho, não configuramos a memória NAND.

Com as imagens do *host,* e de acesso remoto via `NFS`, foram compilados os códigos necessários, e, depois de compilados, os programas binários resultantes foram devidamente testados nesse sistema (tabelas 3 e 4). Posteriormente, com *boot* via memória *flash*, buscamos do *host* os arquivos binários anteriormente compilados com o sistema de arquivos da placa armazenados no HD do *host* e acessados via `NFS`, com o comando `TFTP`. Dessa forma, foram testados os seguintes codificadores discutidos no capítulo 3:

-iLBC com quadros de 20 ms (ponto flutuante)
-iLBC com quadros de 30 ms (ponto flutuante)

-G.729

-G726

-G.711 usando a lei μ

-G.711 usando a lei A

-Celp (ponto flutuante)

-G.723.1 a 6,3kbps

-G.723.1 a 5,3kbps

Como o processador utilizado não é dotado de unidade de ponto flutuante, os codificadores com implementação em ponto flutuante foram inviabilizados, uma vez que para executar os codigos desses codificadores, era necessário ao processador realizar emulação de ponto flutuante. Nos testes realizados com esses códigos, obtivemos tempos de processamento da ordem de 30 minutos tanto para codificação quanto para decodificação, sendo assim excluiremos da análise os resultados desses codecs.

Foram feitos os testes com esses codificadores, e, para a avaliação do desempenho, os códigos foram escritos para trabalhar com um arquivo de entrada com 8.000 amostras por segundo, com 16 bits por amostra, o arquivo de teste usado chama-se `TII`.raw, e tem duração de aproximadamente 59,98 segundos, já que esse arquivo tem 960kB. E esses códigos tem como arquivo de saída um novo arquivo que é nomeado pelo usuário, no desenvolver deste trabalho, adotamos um padão para os nomes dos arquivos de saída dos processos, `TIIxx.cod`, quando se trata de um arquivo resultado de codificação, onde xx diz algo sobre a que codec esse arquivo refere-se. Então, usando o decodificador respectivo, processa-se, então esse arquivo `TIIxx.cod`, e nomeia-se um novo arquivo, nesse padrão, de `TIIxx.dec`.

Do processo descrito acima, os seguintes dados foram avaliados: o tempo de processamento, o percentual de uso do processador, e o tamanho do arquivo de saída.

Tempo de processamento: o fator mais importante desta avaliação. O arquivo de voz utilizado nos testes é um arquivo de 59,98 segundos, assim se o tempo de processamento do codificador somado ao tempo de processamento do decodificador for superior aos 59,98 segundos do arquivo fonte, então, a implementação do codec em questão está condenada para o sistema no qual o teste é realizado.

Esse elemento de análise é obtido executando o comando `time` na mesma linha que se executa o codificador ou decodificador, da seguinte maneira:

```
time <codificador> <arquivo de entrada> <arquivo de saída>
```

Analisando o tempo de processamento de cada codificador e decodificador, é possível ter uma idéia se estes funcionariam em tempo real ou não. Se, por exemplo, um desses codificadores demorar

mais de sessenta segundos para realizar a codificação completa, tem-se a certeza de que não terá um funcionamento satisfatório se implementado em tempo real. Se, por exemplo, executando isoladamente, a soma dos tempos de decodificação e codificação for superior a 59,98 segundos, também é muito provável que o codificador em teste não funcionaria em tempo real quando, para ambos os processos, o uso do processador for de aproximadamente 100% na quase totalidade do tempo de execução. Portanto, buscamos tempos de codificação e decodificação de um codec tal que a soma do tempo de processamento de ambos seja menor que 60s.

Percentual de uso do processador: um estudo de codecs executando isoladamente o codificador e, depois, o decodificador, é um teste apenas preliminar do ponto de vista da previsão do comportamento do sistema embarcado para uso em tempo real. Nesse caso, seria ncessário analisar e especular os resultados que seriam obtidos em uma implementação em tempo real.

Essa variável de estudo pode ser obtida usando-se o comando `top` da s*hell*, esse comando deve ser executado em um outro terminal, no caso de nossa placa, como o terminal dela é remoto, e acessávamos via `minicom`, não tínhamos acesso a um outro terminal via `minicom`, a solução encontrada foi usar o comando `telnet` em uma outra `shell` do *host* e assim, ter acesso a um segundo terminal remoto da plataforma de desenvolvimento, tendo, então, acesso ao comando `top`.

Tamanho do arquivo de saída: com esse dado, e, sabendo que o arquivo é de 59.98 segundos, pode-se fazer uma conta simples de T/t=r, onde T é tamanho do arquivo em KB multiplicado por 8 (obtendo assim o tamanho do arquivo em kb), t é o tempo que o arquivo representa, 59,98 segundos, e r é a taxa final em kbps. O tamanho do arquivo pode ser verificado com o comando `ls –la <nome-do-arquivo>`. A grande relevância do tamanho do arquivo de saída é estimar a taxa de transmissão média do codificador, em kbps.

6.2 RESULTADOS

Utilizando os comandos acima descritos, `top`, `time` e `ls –la`, devidamente associados à execução dos codecs, coletaram-se os dados das Tabelas 6.1 a 6.4 abaixo, e a título de ilustração e comparação, os dados do trabalho desenvolvido anteriormente por Guilherme Pacheco [16], em uma placa com processador Geode da AMD, um processador plataforma x86.

Os codificadores testados são codificadores para testes preliminares, são processos que tem como entrada um sinal de áudio amostrado em 8000 amostras por segundo, e 16 bits por amostra., o arquivo padrão de teste é um arquivo de 59.98 segundos. Os procedimentos de codificação tem como entrada esse arquivo de teste e a saída do processo é gravada em um arquivo de saída.

Os decodificadores, têm como entrada aquele arquivo resultado do processo de codificação e tem como saida um arquivo contendo o sinal decodificado.

Tabela 6.1: Comparação do desempenho dos codificadores de voz.

			boot nativo	boot NFS	x86	
Codificador	Taxa (kbps)	Tamanho do arquivo gerado (kB)	Tempo de codificação	Tempo de codificação	Tempo de codificação	% de uso da CPU
G.729a	8	59980	46,902s	44,322s	25,230s	100,0%
G.726 - 3 bits	24	179958	16,551s	13,767s	5,061s	60,0%
G.726 - 4 bits	32	239943	17,641s	13,522s	5,159s	59,0%
G.726 - 5 bits	40	299930	19,617s	13,906s	5,298s	45,0%
G.711 u-law	64	479885	8,878s	1,732s	0,194s	10,0%
G.711 A-law	64	479885	10,665s	0,498s		
G.723.1 - 6,3	6,4	47976	114,900s	110,728s	73,264s	100,0%

Da Tabela 6.1 e da Tabela 6.2, verifica-se que o tempo de execução dos codificadores sendo executados com sistema de arquivos armazenado na memória *flash* foi pouco maior que o tempo de execução para o sistema de arquivos acessado remotamente, via NFS. Se fizermos a mesma comparação com os decodificadores, observamos que essa diferença de tempo entre o processo executado no sistema de arquivos armazenado na memória NOR do sistema e o processo executado no sistema de arquivos acessado remotamente é ainda maior. Essa diferença se deve ao fato de que quando o sistema de arquivo se encontra na memória NOR, ponto de maior morosidade é a gravação do arquivo de saída, e como o arquivo de saida dos decodificadores é maior que o arquivo de saída dos codificadores, então, no processo de decodificação, o atraso para gravação da saída do processo se torna mais notável.

Tabela 6.2: Comparação do desempenho dos decodificadores de voz.

			boot nativo	boot NFS	x86	
decodificador	Taxa kbps	Tamanho do arquivo gerado (kB)	Tempo de decodificação	Tempo de decodificação	Tempo de decodificação	% de uso da CPU
G.729a	8	959680	26,487s	9,962s	5,607s	82,0%
G.726 - 3 bits	24	959776	27,748s	12,604s	4,761s	59,0%
G.726 - 4 bits	32	959772	28,143s	12,999s	4,761s	87,0%
G.726 - 5 bits	40	959776	28,339s	13,206s	4,961s	70,0%
G.711 u-law	64	959770	12,838s	1,384s	0,250s	10,0%
G.711 A-law	64	959770	12,536s	0,930s		
G.723.1 - 6,3	6,4	959520	25,698s	8,671s	5,780s	32,0%
G.723.1 - 5,3	5,33	959520	25,203s	8,688s	5,787s	51,0%

Um bom modelo de uso dessa plataforma dedesenvolvimento seria usar o sistema de arquivos armazenado na memória NAND do sistema. No entanto, não conseguimos inplementar essa solução.

A memória NAND tem uma vantagem sobre a memória NOR que é o fato de ter a velocidade de gravação muito mais rápida do que a velocidade de gravação na memória NOR. Não conseguimos implementar uma partida para a plataforma de desenvolvimento com o sistema de arquivos armazenado na memória NAND. No entanto, para simular uma situação de gravação mais rápida do arquivo de saída, realizamos pequenas modificações nos códigos dos codificadores G711 e G729 para que não gravem em um arquivo a saída, apenas execute o processo de codificação. Espera-se com essa etapa comprovar o efeito da morosidade de gravação na memória NOR sobre o atraso total gerado

pelo processo de codificação do arquivo. E, conseguir quantificar o efeito desse tempo de gravação sobre o processo.

Para o procedimento de codificação e decodificação sem a gravação do sinal de saída dos codificadores G.729 e G.711, temos os dados contidos nas tabelas 3 e 4 abaixo.

Tabela 6.3: Comparação do desempenho dos codificadores de voz sem arquivos de saída gravados.

			boot nativo	boot NFS
Codificador	Taxa kbps	Tamanho do arquivo gerado	Tempo de codificação	Tempo de codificação
G.729a	8	59980	44,490s	44,322s
G.711 u-law	64	479885	0,520s	1,732s
G.711 A-law	64	479885	0,300s	0,498s

Tabela 6.4: Comparação do desempenho dos decodificadores de voz sem arquivos de saída gravados.

			boot nativo	boot NFS
decodificador	Taxa kbps	Tamanho do arquivo gerado	Tempo de decodificação	Tempo de decodificação
G.729a	8	959680	9,58s	0m9.962s
G.711 u-law	64	959770	0,117s	0m1.384s
G.711 A-law	64	959770	0,142s	0m0.930s

Das tabelas de comparação mostradas nesse capítulo, podemos especular que em uma aplicação em tempo real, os codificadores G.729, G.711 e G.726 funcionarão bem, pelo quesito atraso de codificação. Por outro lado, o G.723 não será satisfatório.

O codificador G.729 parece viável, ao analisar as Tabelas 3 e 4, numa operação em tempo real, temos que o arquivo não será salvo mas sim será encaminhado via rede. Sendo assim, observando os tempos de codificação e de decodificação para o sistema de arquivos acessado via NFS, temos que o tempo de codificação é de 44,3 s, e o tempo de decodificação é de 9,6 s. Pode-se especular que o tempo deste processo será satisfatório em tempo real.

Os codificadores G723.1 para 6,3 kbps, e G.723.1 para 5,6 kbps têm seus tempos de codificação pouco superior a 60 segundos, que é o tempo que o arquivo que contém o sinal de teste representa em uma aplicação de áudio, o que já os torna inviáveis para implementação em tempo real, uma vez que o atraso de codificação é maior que o tempo de cada bloco de sinal a ser codificado.

O G.711 tem um código mais simples e, nesse caso, uma implementação viável para tempo real, com o tempo de codificação de 9,1 s para a lei u e de 9,5 s para a lei A e de decodificação de 12,8 s para a lei μ e de 12,6 s para a lei A, na implementação com boot e sistema de arquivos na memória flash. Para a implementação com sistema de arquivos via NFS, os tempos de codificação foram de 1,7 s para a lei u e de 0,5 s para a lei A. Os tempos de decodificação foram de 1,4 s para a lei u e de 0,9 s para a lei A. Tudo indica que em uma implementação em tempo real esses codificadores G.711 (lei μ e lei A) funcionarão adequadamente.

O codificador G.726 parece ter tempos razoáveis de codificação e de decodificação, o que também nos leva a crer que funcionará satisfatoriamente. Quando analisamos os tempos de

codificação para o sistema de arquivos acessado remotamente, temos tempos de processamento muito menores. Essa situação é mais próxima da implementação em tempo real, na qual os sinais entram no sistema para serem codificados por uma interface de captura de áudio para ser processado e então lançado para uma interface de transmissao de rede e o processo inverso ocorre na outra ponta, que irá decodificar o sinal transmitido, os tempos de processamto do sinal se tornam menores,

Observa-se claramente o grande atraso gerado pela gravação das saídas dos codificadores quando analisadas as linhas relativas ao codificador G.711 das tabelas 6.1 a 6.4. As diferenças entre os tempos de decodificação desse codificador com e sem gravação quando do sistema de arquivos na memória *flash* mostra que praticamente todo o tempo desse processo é voltado para a gravação do arquivo na memória NOR.

7 CONCLUSÃO

Neste trabalho, construiu-se uma imagem linux para ser usada em aplicações de comunicação de voz em tempo real. Essa imagem fora criada, e os testes em tempo real poderão ser realizados em trabalhos futuros utilizando essa imagem, sendo necessário, no entanto o uso do JACK, ferramenta ainda não instalada na imagem de trabalho.

Foram analisados os codificadores G.711, G729a, G.723, G.726 implementados em ponto fixo, e iLBC e celp implementados em ponto flutuante. Os códigos implementados em ponto flutuante se mostraram muito lentos para aplicações em tempo real. Isso porque a plataforma de desenvolvimento não é dotada de unidade de processamento de ponto flutuante,

Dos codificadores implementados em ponto fixo, o G.723 foi o que apresentou pior rendimento, com tempo de codificação de 1 minuto e 50 segundos, se mostrou inviável para uso em tempo real.

Os demais codificadores não foram condenados para aplicação em tempo real pelo quesito de tempo de processamento. O G.711 e o G.726 funcionarão em tempo real nessa plataforma de desenvolvimento. Provavelmente, funcionará também o G.729, mas este necessita testes mais detalhados para a aplicação em tempo real.

Um estudo mais detalhado sobre a viabilidade da comunicação em tempo real, bem como a implementação em tempo real é um importante tema para trabalhos futuros.

ANEXOS

A TUTORIAL: CRIAÇÃO DA IMAGEM

Passo 1- Estabelecer os pré-requisitos

Estabelecer os pré-requisitos do *host*:

PC:

-Interface de acesso *Ethernet* ;

-Porta Serial com controladores para RS-232;

-1GB de espaço em disco;

-Servidor `NFS` instalado;

-Servidor `TFTP` instalado;

Máquina Virtual:

-`perl`;

-`rsync`;

-`ncurses-devel`;

-`zlib-devel`;

-`rpm-build`;

-`gettext`;

-`bison`.

Passo 2 - A instalação do LTIB

Acesso na máquina virtual como usuário comum:

```
login:vm_user
password:vm_user
```

e, quando necessário, como *root:*

```
login: root
password:vm_root
```

Para instalar o `LTIB`, que já se encontra na maquina virtual, precisamos logar como usuário comum, ir até o diretório onde se encontra o `LTIB`, e dar o comando de instalação:

```
cd /pastaLTIB
./install <durante a instalação, determinar o diretório onde será instalada a ferramenta LTIB>
cd /DiretórioDeInstalaçãoDoLTIB
```

E, para terminar a instalação, é necessário executar o `LTIB` pela primeira vez.

```
./ltib
```

Passo 3 - A criação da Imagem

```
./ltib -configure
```

Aparecerá uma azul no *shell*, nesta tela, deve-se selecionar a forma de criação da imagem desejada. Pode-se selecionar suporte a memória NAND, os pacotes que farão parte da imagem, criação ou não de um arquivo compactado para ser o sistema de arquivos armazenado na memória flash da plataforma de desenvolvimento e outras configurações.

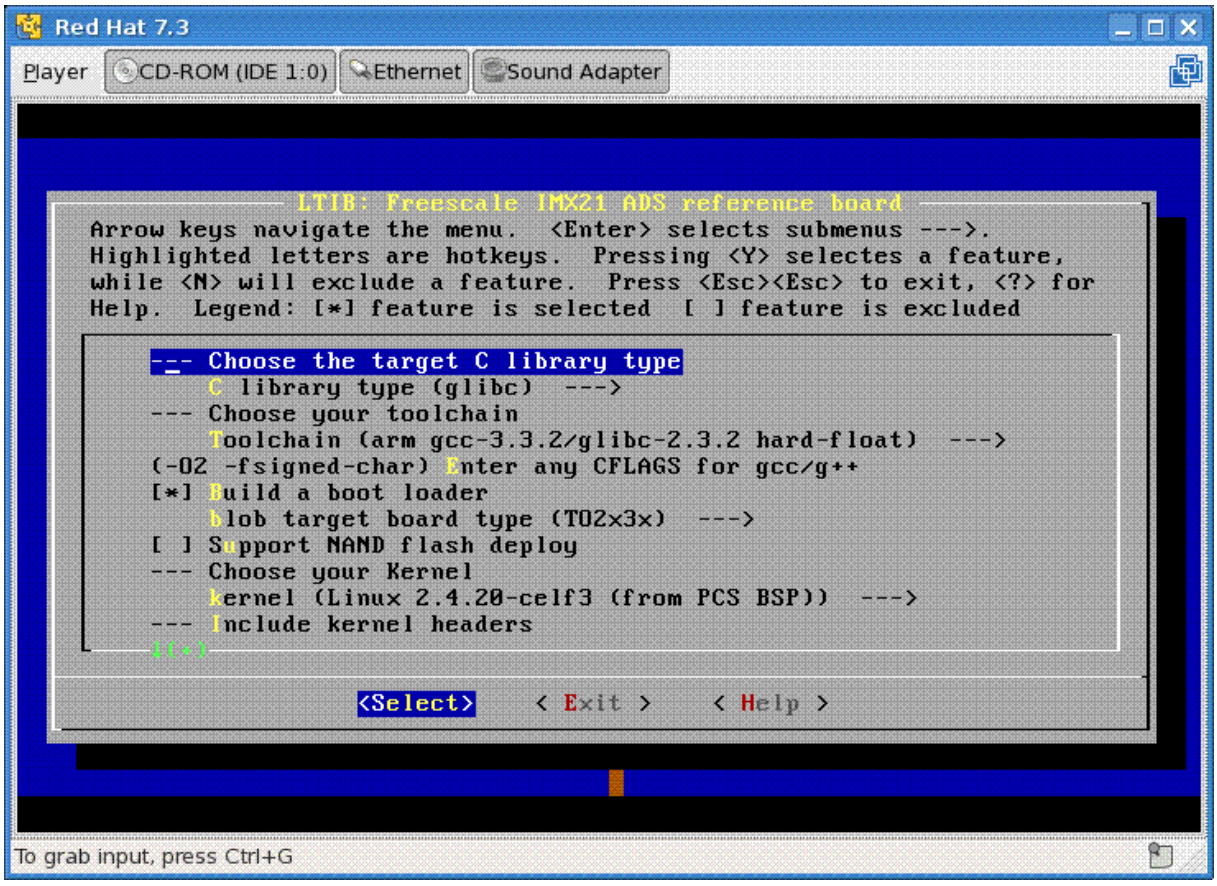

Figura 8.1: Página de configuração do LTIB®.

Essa imagem é criada na máquina virtual, e então deve ser transferida para o PC, essa transferência pode ser feita via SFTP como descrito no passo 4, abaixo.

Passo 4 – Transferindo a imagem da máquina virtual para o *host* (PC)

Para a transferência dos arquivos da máquina virtual para o PC, utiliza-se o seguinte procedimento:

`usuário@MV> cd` <DiretórioDeInstalçãoLTIB>

`usuário@MV> tar cvf  rootfs.tar rootfs` (essas duas ultimas linhas são necessárias para o caso de uso do sistema de arquivos acesado remotamente)

`usuário@PC> mkdir /home/usuário/QualquerDiretórioDeTrabalho` (esse será o diretório onde serão descarregados os arquivos gerados pelo LTIB)

`usuário@PC> sftp vm_user@xxx.xxx.xxx.xxx` (ip da máquina virtual)

`sftp> cd /home/ltib` (ou o diretório onde fora criada a imagem do host na máquina virtual)

`sftp> get arquivo1` (blob, o bootloader, por exemplo)

`sftp> get arquivo2` (zImage, o kernel, por exemplo)

`sftp> get arquivo3` (param, o arquivo que seta parâmetros na flash, especificada a sua criação no passo 6, abaixo - por exemplo)

`sftp> get arquivo4.jffs2` (esse arquivo deve conter o sistema de arquivos compactado no sistema de arquivos para memória flash, jffs, por exemplo. Uma alternativa é usar).

`sftp> quit` (sai do sftp)

`usuário@PC> ls` (devem aparecer aqui os arquivos 1, 2, 3 e 4)

```
usuário@PC> cp arquivo1 /tftpboot/imx21ads
usuário@PC> cp arquivo2 /tftpboot/imx21ads
usuário@PC> cp arquivo3 /tftpboot/imx21ads
```

`usuário@PC> cp arquivo4.jffs2 /tftpboot/imx21ads/` (para sistema de arquivos a ser armazenado na memória flash)

`usuário@PC> tar xvf arquivo4.tar`

`usuário@PC> ln rootfs /tftpboot/imx21ads/ltib` (este é o diretório de trabalho remoto no qual será dado o *boot* com o sistema de arquivos remoto).

Passo 5 - Configurando o Minicom e inicializando o target

```
usuário@PC> minicom -s
|-------------------------------|
|-Filenames and paths           |
|-File transfer protocols       |
|-Serial port setup             |
|-Modem and dialing             |
|-Screen and keyboard           |
|-Save setup as df1             |
|-Save setup as..               |
|-Exit                          |
|-Exit from Minicom             |
|-------------------------------|
```

o "*serial port setup*" deve ficar da seguinte maneira:

```
---------------------------------------------------------
A - Serial Device       : /dev/ttyS0
B - Lockfile Location   : /var/lock
C - Callin Program      :
D - Callout Program     :
E - Bps/Par/Bits        : 115200 8N1
F - Hardware F Control  : No
G - Software F Control  : No
---------------------------------------------------------
```

Feitas as configurações do `minicom` conforme as necessidades da plataforma de desenvolvimento, liga-se a placa de desenvolvimento, interrompe-se a partida nos primeiros segundos da placa ligada, pressionando a tecla <ENTER>.

`blob> help` (imprime todos os comandos disponíveis no blob)

`blob> status` (imprime o status atual, incluindo a situação do blob, kernel, sistema de arquivos da memória *flash*, configurações de rede no *boot*, e endereço de onde a placa busca os arquivos a serem baixados no *host*.)

`blob> ip xxx.xxx.xxx.xxx` (ip do target - a placa)

`blob> server xxx.xxx.xxx.xxx` (ip do host - o PC)

`blob> Tftpfile /tftpboot/imx21ads/zImage` (esse comando indica para o `blob` em qual diretório do *host* buscar os arquivos a serem carregados na RAM, para posteriormente ser armazenda na memória *flash*)

`blob> tftp arquivo` (descarrega o arquivo armazenado no PC, no diretório indicado na linha de comando anterior, para a RAM)

`blob> flash arquivo`.(armazena o arquivo na memória *flash*, conforme a configuração do `blob`, pode ser a memória NOR ou NAND).

`blob> boot` (para ter o *boot* com sistema de arquivos armazenado na memória *flash* do sistema)

```
 blob> boot noinitrd root=/dev/nfs
nfsroot=<IPDoHost>:/tftpboot/imx21ads/ltib
ip=<IPDoTarget>:<IPDoServidor>
```

Passo 6 – O arquivo "Param"

Para criar o arquivo binário contendo os parâmetros para a configuração adequada da plataforma de desenvolvimento seguem-se os seguintes passos:

```
>./ltib -p blob-imx21ads.spec -m prep
>pushd rpm/BUILD/blob-2.0.5-pre2/utils/mkparamblock
>make
>cp mkparamblock <install_path>/ltib/bin/
>popd
```

Então, criamos e configura-se o arquivo de texto `blob.config.param` com os parâmetros desejados:

```
# Config file for Freescale i.MX21
ip 192.168.10.38
server 192.168.10.7
tftpfile /tftpboot/imx21ads/zImage
ramdisk no # o default é "yes"
bootdelay 3 # o default é 10
cmdline root=/dev/mtdblock2 noinitrd ip=none
baud 115200, 115200 # best for serial deployment
```

```
autoboot ram # "flash" para XIP, "ram" para kernel normal

> <DiretórioDeInstalaçãoDoLTIB>/ltib/bin/mkparamblock
blob.config.param param
> cp param /tftpboot/imx21ads
```

Assim, cria-se o arquivo param que deve, então ser copiado para a pasta `/tftpboot/imx21ads.` No *boot loader* do kit de desenvolvimento da Freescale® (`blob`), via `minicom`, setamos o IP da máquina, IP do *host*, o diretório onde se encontra o sistema de arquivo, o *kernel* e o `blob` e então baixamos o arquivo `param` necessário e então o salvamos para a memória *flash* conforme os comandos abaixo:

```
blob> ip 192.168.10.38
blob> sever 192.168.10.7
blob> Tftpfile /tftpboot/imx21ads/zImage
#blob> status
blob> tftp param
blob> flash param
```

Antes do comando `TFTP` verificar com o comando `status` o `IP`, o `server` e o diretório onde se encontra o sistema de arquivo.

Ao realizar o comando `flash param`, os parametros do arquivo `blob.config.param` serão passados para os endereços da memória NOR onde se buscam esses parâmetros durante o *boot*.

Passo 7 – tftp, flash e boot

Para cada arquivo a ser passado para a memória *flash*, deve-se realizar os passos abaixo descritos.

```
blob> erase X
blob> tftp X
blob> flash X
```

`blob> status` (confere se está tudo correto, caso contrário repetir, corrigindo os erros)
`blob> boot`

Com o comando `boot` será iniciado o *boot* já com os novos arquivos, e com o sistema de arquivos que está na memória *flash*, alternativemente, podemos usar o *boot* com sistema de arquivos acessado via `NFS`, com o seguinte comando (em uma única linha):

```
blob> boot noinitrd
root=xxx.xxx.xxx.xxx:/tftpboot/imx21ads/ltib/rootfs
nfsroot=xxx.xxx.xxx.xxx:/tftpboot/imx21ads/ltib
ip=yyy.yyy.yyy.yyy:xxx.xxx.xxx.xxx:192.168.10.1:255.255.255.0:mx21:eth0:off
```

(sendo `xxx.xxx.xxx.xxx` o IP do *host*, yyy.yyy.yyy.yyy o IP da plataforma de desenvolvimento, 192.168.10.1 o *gateway* e 255.255.255.0 a máscara de sub-rede. Assim teremos o *boot* com o sistema de arquivos acessado remotamente, estando armazenado o no diretório ao qual existe um *link* o `/tftpboot/imx21ads/ltib` do *host*.)
Será pedido o *runleve*" deve-se entrar com <s>.

B LISTA DOS PACOTES UTILIZÁVEIS NO LTIB E SUAS RESPECTIVAS LICENÇAS

Pacote	Licença [24]
DirectFB-0.9.24-1	LGPL
alsa-lib-1.0.10-0	LGPL
alsa-utils-1.0.10-0	GPL
autoconf-2.57-1	GPL
automake-1.7.6-1	GPL
base_libs-1.0-1	LGPL
bash-2.05b-1	GPL
bind-9.3.2-1	Internet Systems Consortium (distributable)
binutils-2.15-1	GPL
bison-1.875-1	GPL
blob-2.0.5_pre2-1	GPL
boa-0.94.14rc21-1	GPL
bonnie++-1.93c-1	GPL
bridge-utils-1.1-1	GPL
busybox-1.1.3-1	GPL
bzip2-1.0.2-1	BSD
can4linux-3.3.3-1	GPL
clamav-0.88-1	GPL
coreutils-6.3-1	GPL
cpio-2.6-1	GPL/LGPL
cracklib-2.8-9	GPL
cramfs-1.1-1	GPL
daemonizer-1.0-0	GPL
db1-1.85-8	BSD
demo_launcher-1.0-1	GPL
dev-1.1-1	GPL
devfsd-1.3.25-1	GPL
devmem2-1.0-1	GPL
dhcp-3.0.3b1-1	Internet Systems Consortium (distributable)
diffutils-2.8.1-1	GPL
distcc-2.18.3-1	GPL
dosfstools-2.11-1	GPL
dpm-utils-imx-1.0-1	GPL
dropbear-0.45-1	MIT
dtc-07jul06-1	GPL
e2fsprogs-1.34-1	GPL
ed-0.2-1	GPL
ethtool-3-1	GPL
expat-1.95.7-3	MIT/X Consortium License
fake-provides-1.0-5	GPL
fakeroot-1.5.10-1	GPL
findutils-4.2.28-1	GPL
flex-2.5.4-1	BSD
freetype-2.1.7-1	GPL or FTL
gawk-3.1.3-1	GPL
gcc-3.3.2-1	GPL
gdb-6.3.50.20051117-0	GPL
genext2fs-1.3-1	GPL
genromfs-0.5.1-1	GPL
gettext-0.15-1	GPL
gmp-4.1.4-1	LGPL
grep-2.5.1-1	GPL
groff-1.18.1-1	GPL
hantro-drivers-mx21-1.0-	GPL
hardwaretest-imx-1.2-1	GPL
hdparm-5.9-1	BSD

helloworld-1.0-1	Public Domain, not copyrighted
hesiod-3.0.2-1	Internet Systems Consortium (distributable)
hotplug-2004_03_29-1	GPL
httpd-2.0.54-0	Apache (distributable)
i2c-tools-2.8.1-1	GPL
inetutils-1.4.2-1	GPL
iperf-1.7.0-1	Distributable/GPL
iproute-2.6.11-050330	GPL
ipsec-tools-0.6.4-1	BSD
ipsecadm-0.9-pre1	GPL
iptables-1.3.6-1	GPL
iputils-0.0.4-1	GPL
irattach-0.9.17-1	GPL
kbd-1.08-1	GPL
kernel-2.4.20_celf3-1	GPL
less-381-1	GPL
lfs-utils-0.3-1	BSD/UCB style license (distributable)
libelf-0.8.5-1	LGPL
libid3tag-0.15.1b-1	GPL
libjpeg-6b-1	Distributable
libmad-0.15.1b-1	GPL
libpcap-0.8.3-1	BSD
libpng-1.2.8-1	distributable, OSI approved
libtermcap-2.0.8-31_1	LGPL
libtool-1.5-1	GPL
libusb-0.1.8-1	LGPL
libxml2-2.6.19-1	MIT
linux-atm-2.4.1-1	GPL/LGPL
linux-wlan-ng-0.1.12-1	MPL
lkc-1.4-2	GPL
lmbench-3.0-a4-1	GPL + restrictions
lrzsz-0.12.21-1	GPL
ltp-full-20050608-1	GPL
m4-1.4-1	GPL
madplay-0.15.2b-1	GPL
make-3.80-1	GPL
man-1.5m2-1	GPL
mdadm-2.3.1-1	GPL
merge-0.1-1	GPL
microwindows-0.90-0	MPL/GPL
module-init-tools-3.1-0.	GPL
modutils-2.4.25-1	GPL
mtd-utils-20060302-1	GPL
mysql-4.1.12-0	GPL
ncurses-5.3-1	Distributable
net-tools-1.60-1	GPL
netperf-2.4.0-1	Distributable
ntpclient-2003_194-1	GPL
openssh-4.3p2-1	BSD
openssl-0.9.8a-1	BSD-style
openswan-2.3.0-1	GPL
oprofile-0.9.1-1	GPL
patch-2.5.4-1	GPL
pciutils-2.1.11-1	GPL
pcmcia-cs-3.2.4-1	Mozilla Public License Version 1.1
pcre-6.3-1	BSD License (revised)
perl-5.8.8-1	Artistic or GPL
php-5.0.4-0	PHP (distributable)
pidentd-3.0.19-1	Freely Distributable
popt-1.7-1	X11 style
portmap-5beta-1	BSD
ppp-2.4.1-1	BSD
procinfo-18-1	GPL
procps-3.1.11-1	GPL

psmisc-22.3-1	GPL
ptpd-1b4-1	BSD License
python-2.2.1-1	OSI Approved Python License
qtopia-free-2.2.0-1	GPL
rng-tools-2-1	GPL
rpm-4.0.4-1	GPL
rsync-2.6.5-1	GPL
samba-3.0.2a-1	GPL
sash-1.1-1	GPL/Distributable
schedutils-0.7.2-1	BSD License
sed-4.0.7-1	GPL
seq-6.3-1	GPL
setserial-2.17-1	GPL
skell-1.13-1	GPL
strace-4.4.98-1	BSD
sysconfig-1.0-1	GPL
sysklogd-1.4.1-1	GPL/BSD
sysvinit-2.85-1	GPL
tar-1.15-1	GPL
tcp_wrappers-7.6-1	Distributable
tcpdump-3.8.3-1	BSD
texinfo-4.8-1	GPL
time-1.7-1	GPL
timezone-2006n-1	Public Domain/BSD
tinylogin-1.4-1	GPL
u-boot-tools-1.1.2-1	GPL
cksum-19990607-1	BSD
udev-056-1	GPL
unzip-5.52-1	BSD
usbutils-0.70-1	GPL
util-linux-2.12-1	GPL
vim-6.2-1	Vim (Distributable)
vsftpd-2.0.3-1	GPL
wget-1.9.1-1	GPL
which-2.14-1	GPL
wireless-tools-28-1	GPL
yaffs_utils-20060418-1	GPL
zlib-1.1.4-2	zlib (Distributable)

LISTA DE SÍMBOLOS

Siglas

ACELP - *Algebraic Code Excited Linear Prediction*
ALSA - *Advanced Linux Sound Architecture*
ARM- *Advanced RISC Machine*
BLOB - *Boot Loader*
BSD - *Berkeley Software Distribution*
CELP - *Code-Excited Linear Prediction*
CODEC - Codificador / Decodificador
CPU - *Central Processing Unit*
CS - *ACELP Complementary Symmetry-Algebraic Code Excited Linear Prediction*
EIM - *External Module Interface*
FTP - *File Transfer Protocol*
GCC - *GNU Compiler Collection*
GNU - é um acrônimo recursivo para "GNU Não é Unix"
GPL - *General Public License*
HD - *Hard Disk*
HTTP - *HyperText Transfer Protocol*
iLBC - *Internet Low Bit Rate codec*
IP – *Internet Protocol*
ITU-T - *International Telecommunication Union-Telecommunication*
JACK - *Jack Audio Connection Kit*
kbps - *Kilobits* por segundo
LAN - *Local Area Network*
LGPL - *Lesser General Public License*
LP - *Linear Prediction*
LSF - *Line Spectral Frequency*
LTIB - *Linux Target Image Builder*
MIT - *Massachusetts Institute of Technology*
MP - MLQ- *Multipulse Maximum Likelihood Quantization*
MMU - *Memory Management Unit*
NAND - *Not And*
NOR- Not Or
NFC - NAND *Flash Controller*
NFS - *Network File System*
PC - *Personal Computer*
PCI - *Peripheral Component Interconnect*
PCM - *Pulse Code Modulation*
PSVQ - *Predictive split vector quantizer*
RAM - *Random Access Memory*
RSH - *Remote Shell*
SD - card- *Secure Digital Memory Card*
SNR - *Signal-to-Noise Ratio*
SFTP -*Secure Shell File Transfer Protocol*
SSH - *Secure Shell*
TCP -*Transmission Control Protocol*
TFTP - *Trivial File Transfer Protocol*
UART – *Universal asynchronous receiver/transmitter*
USB-OTG – *Universal Serial Bus – On The Go*
VoIP - *Voice over Internet Protocol*

REFERÊNCIAS BIBLIOGRÁFICAS

[1] GOODE,B. *Voice Over Internet Protocol (VoIP)*. IEEE, SEPTEMBER, 2002.
[2] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX21&fsrch=1
[3] www.freescale.com.
[4] http://infocenter.arm.com/help/index.jsp.
[5] http://www.arm.com/markets/mobile_solutions/armpp/18072.html.
[6] http://www.gnu.org/philosophy/free-sw.pt.html.
[7] http://www.gnu.org/copyleft/gpl.html.
[8] http://www.gnu.org/licenses/lgpl.html.
[9] http://www.linfo.org/bsdlicense.html.
[10]http://www.opensource.org/licenses/mit-license.php
[11] ITU-T Rec. G.711 - Pulse Code Modulation (PCM) of Voice Frequencies. Disponível em: <http://www.itu.int/rec/T-REC-G.711-198811-I/en>. Acesso em 19 dez. 2007.
[12] www.seprorj.org.br/RioInf2007_material/Tec_Emergentes/claudio_mauricio.pdf.
[13] ITU- T Rec. G.726 – 40, 32, 24, 16 kbit/s ADAPTIVE DIFFERENTIAL PULSE CODE MODULATION (ADPCM). Disponível em: <http://www.itu.int/rec/T-REC-G.726-199012-I/en>. Acesso em 19 dez. 2007.
[14] ITU-T Rec. G.729 - Coding of Speech at 8kbit/s using conjugate-structure algebraic-codeexcited linear prediction (CS-ACELP). Disponível em: <http://www.itu.int/rec/T-REC-G.729/en>. Acesso em 19 dez. 2007.
[15] OBANDO, M. S. B. *Comparação do desempenho dos Codificadores de Voz G.711, G729, G723.1 e iLBC em Transmissão com Perda de Pacotes*. Dissertação (Mestrado) – Universidade de Brasília – Faculdade de Tecnologia – Departamento de Engenharia Elétrica, 2005..
[16] TAVARES,G. P. *Implementação de Sistema VoIP em Plataforma Embarcada Utilizando Código Aberto*. Trabalho de Graduação — Universidade de Brasília - Faculdade de Tecnologia - Departamento de Engenharia Elétrica, 2007.
[17] Freescale. *M9328MX21ADSE Application Development System - User's Manual*. Rev. A, July/2006, Document Number UMS-21100.
[18] Freescale MC9328MX21 Application Processor. Ver 2.1, July 2007, document number MC9328MX21P.
[19] http://www.arm.com/products/CPUs/families/ARM9EFamily.html.
[20] http://www.arm.com/products/CPUs/archi-thumb.html.
[21] Freescale, *Flash Architectures for i.MX Applications Processors*, Rev. 1, October/2006, Document Number AN2711.
[22] RCF1813 - NFS Version 3 Protocol Specification.
[23] Freescale. Linux BSP for the freescale i.MX21ADS User's Guide. Rev.1.2, June/2007.
[24] Freescale EULA.