UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DE UM FILTRO EQUALIZADOR LMS NUM KIT DSK TMS320C6711

ANDRÉ LEONARDO SOLÉO MIRANDA – 99/16971

ORIENTADOR: FRANCISCO ASSIS DE OLIVEIRA NASCIMENTO

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Brasília/DF: setembro de 02

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DE UM FILTRO EQUALIZADOR LMS NUM KIT DSK TMS320C6711

ANDRÉ LEONARDO SOLÉO MIRANDA – 99/16971

PROJETO FINAL DE GRADUAÇÃO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE EM ENGENHARIA ELÉTRICA

APROVADA POR:

FRANCISCO ASSIS DE OLIVEIRA NASCIMENTO, Doutor, UFRJ
(ORIENTADOR)

ADSON FERREIRA DA ROCHA , Ph. D., Univ. Texas at Austin
(EXAMINADOR INTERNO)

RAIMUNDO GUIMARÃES SARAIVA JÚNIOR, Mestre, UNB
(EXAMINADOR EXTERNO)

DATA: BRASÍLIA/DF, 09 DE SETEMBRO DE 2002.

Índice analítico

Índice de figuras

1 Resumo

O presente trabalho trata da implementação de um equalizador adaptativo LMS no kit DSK 6711/6211 da Texas Instruments através da interface de depuração em tempo real RTDX (Real Time Extended Data – Troca de dados em tempo real), o qual pretende ter a sua implementação empregada na disciplina "Laboratório de Processamento de Sinais Digitais".

Para tanto a escolha do tema foi baseada numa experiência que tratasse de genericamente a utilização do Kit, dos conceitos de Processamento de Sinais, da facilidade de sua implementação e, sobretudo, da capacidade de atender uma demanda acadêmica de integração de uma aplicação real (baseada no kit da Texas Instruments) e a flexibilidade de uma ferramenta teórica (MATLAB Versão 6.1 da Mathworks).

A implementação está baseada, portanto, em duas plataformas de geração de código fonte. A primeira, o MATLAB, tem relativo suporte acadêmico já que integra várias ferramentas computacionais chamadas de Toolboxes em único aplicativo que as pode trata integradamente no desenvolvimento acadêmico. Assim podemos manipular entidades como o cliente OLE (referida mais adiante na introdução teórica) base para a interface RTDX através do Matlab sem que o usuário tenha a ciência do que realmente estão acontecendo na integração do seu aplicativo matlab e a plataforma de desenvolvimento DSK. A segunda plataforma de geração de código-fonte é o aplicativo da Texas chamado Code Composer Studio, como o próprio nome diz, é "estúdio" de edição, ou melhor, um ambiente de desenvolvimento que integra ferramentas de depuração e geração de código, com muitas funções de API's prontas. Além do próprio suporte da Texas que disponibiliza uma quantidade numerosa de códigos típicos de DSP's, além de outros mais.

2 Palavras-Chave

Equalizador Adaptativo; DSK 6711/6211; RTDX – Real Time Data Exchange; TMS320C6711; LMS – Least Mean Square;

3 Introdução Teórica e Conceitual

3.1 Conceitos de Equalização

3.1.1 Introdução

Sistemas de equalização baseados em DSP's ficaram onipresentes em muitas aplicações, inclusive voz, dados e comunicações por vários meios físicos de transmissão. Aplicações típicas variam de canceladores de eco acústicos, para sistemas eliminadores de fantasmas (efeitos de múltiplos percursos) de sinais de vídeo em televisores terrestres de radiodifusão, para modems e para a telefonia sem fio. Além de corrigir o canal em anomalias de resposta em freqüência, o cancelador de eco pode amenizar os efeitos de componentes do sinal em múltiplos percursos, que podem-se manifestar na forma de ecos de voz, fantasmas de sinais de vídeo ou condições de perdas por espaço livre em canais de comunicação móvel. Equalizadores, especificamente projetados para correção de múltiplos percursos, são freqüentemente chamados de canceladores de eco. A literatura é rica com tratamentos práticos e teóricos de vários esquemas de equalizadores e de canceladores de eco. O propósito desta monografia é de tentar familiarizar o leitor com alguns conceitos básicos associados com equalização de canal e comunicação de dados em geral. Há uma esperança de que o uso explícito de gráficos de sinais levará a um entendimento intuitivo de conceitos como interferência intersimbólica. Para um tratamento matemático mais rigoroso, refira-se aos numerosos livros e artigos citados na referência.

Um área de particular interesse atualmente é a área de comunicação digital celular, que tem feito um largo uso de DSP's de ponto-fixo tal como a família TMS320C5x. Esta família de processadores fornece o poder de processamento para realizar o esforço inerente a equalização adaptativa enquanto ao mesmo tempo roda tarefas como codificação de canal, correção de erro, funções de compressão de voz.

3.1.2 O que é interferência intersimbólica

Considere o que acontece quando uma informação pulsada é transmitida ao longo de um canal analógico tal qual um canal telefônico ou ondas aéreas. Mesmo que o sinal original seja uma seqüência discreta no tempo (ou com uma razoável aproximação), o sinal recebido é um sinal continuo no tempo. Didaticamente, alguém pode considerar que o canal atua com um filtro analógico passa-baixas, distorcendo a forma de um trem de pulsos dentro de um sinal continuo, cujos cumes relacionam as

amplitudes dos pulsos originais. Matematicamente, a operação pode ser descrita com uma convolução de uma seqüência de pulsos pela resposta de canal em tempo continuo. A operação começa com uma integral de convolução:

$$r(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \quad (1)$$

Onde r(t) é o sinal recebido, h(t) é a resposta impulsiva do canal e x(t) é o sinal de entrada. A segunda metade da equação 1 é um resultado do fato de que a convolução é uma operação comutativa.

A componente x(t) é o trem de pulsos da entrada, que consiste de pulsos transmitidos periodicamente de amplitudes variadas, então:

$$x(t) = 0 \ \textit{para}\ t \neq kT \quad (2)$$
$$x(t) = X_k \ \textit{para}\ t = kT \quad (3)$$

Onde T representa o *símbolo* período. Isto significa que somente valores significantes da variável de integração na integral (3) são aquelas pelos quais $\tau = kT$. Qualquer outro valor de τ equivale a multiplicação por 0. Então r(t) pode ser reescrito como

$$r(t) = \sum_{k=-\infty}^{\infty} x_k h(t-kT) \quad (4)$$

Esta representação de r(t) mais de perto se assemelha à soma convolucional familiar para engenheiros de DSP. Note, contudo, que ele ainda descreve um sistema de tempo contínuo. Ele mostra que o sinal recebido consiste de uma soma de muitas respostas impulsivas escalonadas e deslocadas de sistemas contínuos no tempo. A resposta impulsiva é escalonada por amplitudes dos pulsos transmitidos de x(t).

Como um exemplo, considere o calculo de para r(t) num índice de tempo não inteiro (t = 1,1):

$$r(1{,}1) = \cdots + x_{-2}h(1{,}1+2T) + x_{-1}h(1{,}1+T) + x_0 h(1{,}1) + x_1 h(1{,}1+T) + x_2 h(1{,}1-2T)\cdots \quad (5)$$

Alguém pode ver como os valores recebidos para qualquer tempo t são computados. Cada valor de pulso de uma seqüência de entrada, x_k, contribui com uma componente do somatório de saída.

Porque se está interessado em processar o sinal recebido no *hardware* digital, deve-se representar o sinal recebido como uma equação de diferença. Fisicamente, amostra-se periodicamente o sinal recebido. Para o caso de modulação por amplitude de pulso, é suficiente amostrar o sinal recebido à taxa transmitida de

símbolos, $\frac{1}{T^2}$. (em algumas instâncias, pode ser vantajoso amostrar a um múltiplo da taxa de símbolos para implementar um sistema de processamento de sinal fracionadamente espaçados). Para representar a amostragem matematicamente, substitua t por nT, onde, outravez, T é a taxa de símbolos transmitida:

$$r(nT) = \sum_{k=-\infty}^{\infty} x_k h(nT - kT) \quad (6)$$

Que também pode ser escrita como

$$r(nT) = x_n h(0) + \sum_{k \neq \infty} x_k h(nT - kT) \quad (7)$$

Um último fato para considerar na fase de amostragem. A menos que a freqüência de amostragem seja perfeitamente sincronizada com a freqüência de transmissão, o deslocamento da fase de amostragem não será nula. Para considerar um deslocamento arbitrário de fase na equação (7), adicione um t_0 deslocado em relação ao índice de tempo.

$$\mathbf{r}(\mathbf{nT} + \mathbf{t}_0) = \mathbf{x_n h}(\mathbf{t}_0) + \sum_{\mathbf{k} \neq \infty}^{\mathbf{N}} \mathbf{x_k h}(\mathbf{t}_0 + \mathbf{nT} - \mathbf{kT}) \quad (8)$$

Na equação 8, o primeiro termo é componente de r(t) devido ao enésimo símbolo. Ele é multiplicado pelo coeficiente central da resposta impulsiva do canal. Os outros termos de produto no somatório são termos da interferência intersimbólica (ISI). Os pulsos de entrada na vizinhança do enésimo símbolo são escalonados por amostras apropriadas nas extremidades da resposta impulsiva de canal. Abaixo temos exemplos numéricos para vários valores de n com $t_0 = 0{,}1$ para valores de k ao longo das cinco amostras na vizinhança de n.

$$\mathbf{r}(0{,}1) = \mathbf{x}_0\mathbf{h}(0{,}1) + \mathbf{x}_{-2}\mathbf{h}(2{,}1) + \mathbf{x}_{-1}\mathbf{h}(1{,}1) + \mathbf{x}_1\mathbf{h}(-0{,}9) + \mathbf{x}_2\mathbf{h}(-1{,}9)\cdots(\mathbf{n} = 0) \quad (9)$$

$$\mathbf{r}(1{,}1) = \mathbf{x}_1\mathbf{h}(0{,}1) + \mathbf{x}_{-1}\mathbf{h}(2{,}1) + \mathbf{x}_0\mathbf{h}(1{,}1) + \mathbf{x}_2\mathbf{h}(-0{,}9) + \mathbf{x}_3\mathbf{h}(-1{,}9)\cdots(\mathbf{n} = 1) \quad (10)$$

$$\mathbf{r}(2{,}1) = \mathbf{x}_2\mathbf{h}(0{,}1) + \mathbf{x}_0\mathbf{h}(2{,}1) + \mathbf{x}_1\mathbf{h}(1{,}1) + \mathbf{x}_3\mathbf{h}(-0{,}9) + \mathbf{x}_4\mathbf{h}(-1{,}9)\cdots(\mathbf{n} = 2) \quad (11)$$

A figura 1 mostra um trem de pulso sendo transmitido. O pulso central é x_0, o pulso em 1 é x_1, o pulso em -1 é x_{-1}, etc. se você assume uma resposta impulsiva arbitraria para o canal de transmissão, você pode construir o canal mais sinal recebido r(t). Este sinal é mostrado sobreposto na forma de onda transmitida x(t). Na realidade, a forma de sinal recebida deverá ser deslocada no tempo por causa do atraso de canal, mas para fins de esclarecimento, r(t) é mostrada com nenhum atraso relativo a x(t). Note que os picos de r(t) a grosso modo relaciona o significado dos pulsos correspondentes transmitidos; Contudo, o valor de r(t) nos instantes de amostragem podem ser

totalmente diferentes dos transmitidos. Isto é a causa dos efeitos da interferência intersimbólica.

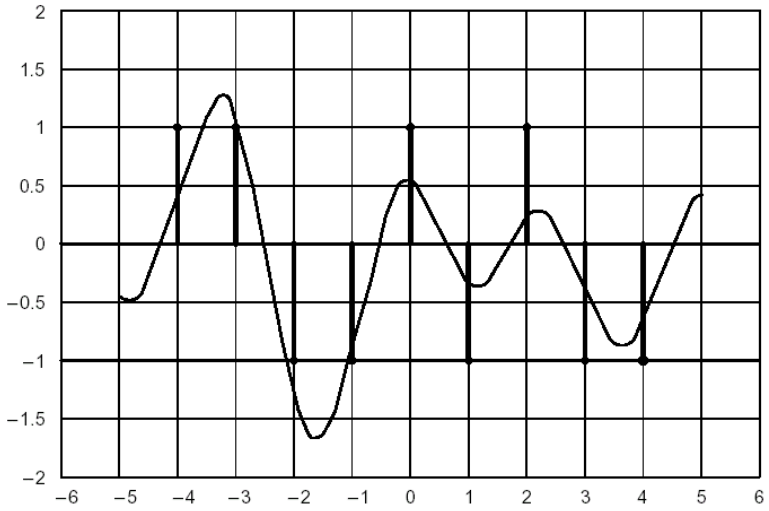

Figura 1 – Um trem de pulso para ser transmitido

A figura 2 mostra a componente de r(t) devido ao único pulso de entrada x_1, que é sobreposto no sinal recebido r(t). Recorde que a forma desta componente é a mesma tal qual da resposta impulsiva do canal transmitido. Os valores dos pulsos individuais nos períodos de amostragem (que são múltiplos de T) são indicados pelos pontos pretos. Note que embora a componente de sinal no exemplo é na forma da *sinc*, os zeros não ocorrem nos intervalos de amostragem. Então, a resposta pulsada, centrada em t = 0, faz contribuições indesejadas para as amostras recebidas próximas de r(t). A contribuição do x_0 símbolo para r(0) é o valor +1.

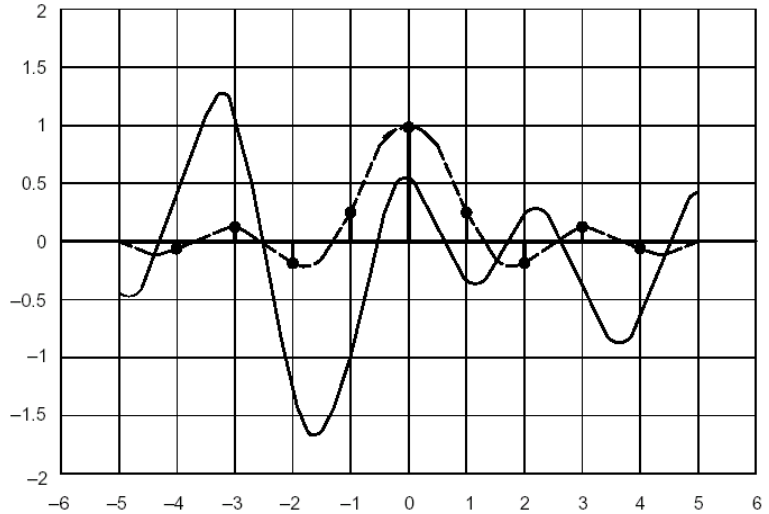

Figura 2 – Componente de r(t)

Para determinar o valor do sinal recebido em t = 0, r(0), some as contribuições da resposta impulsional recebida devido a x_0, x_{-1}, x_1, x_{-2}, x_2... .

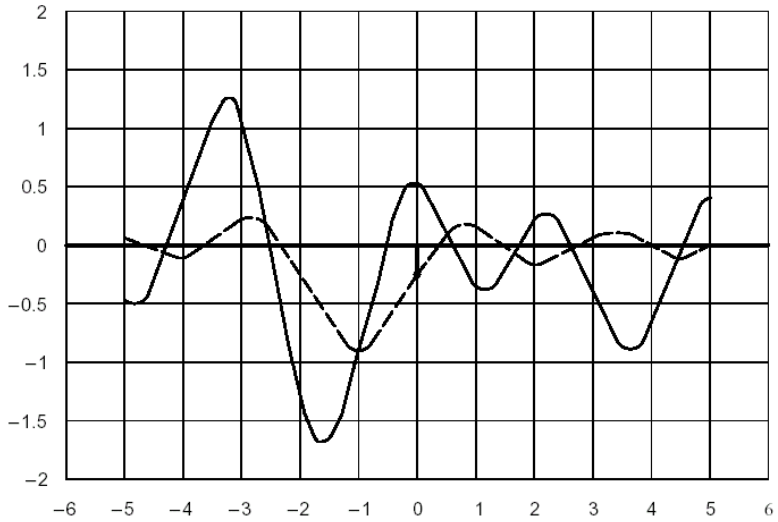

Figura 3 – Contribuição devida a X_1

Como é mostrada na figura 3, a contribuição devida a x_{-1} é o valor em t = 0 da resposta impulsiva deslocada e escalonada correspondendo para pulso transmitido x_{-1}.

Neste caso a resposta impulsiva é escalonada por -1, que é o valor de x_{-1} e é avançado por um período de amostragem porque x_{-1} é transmitido um período antes de x_0. Então, o símbolo x_{-1} resulta numa pequena componente negativa de r(0).

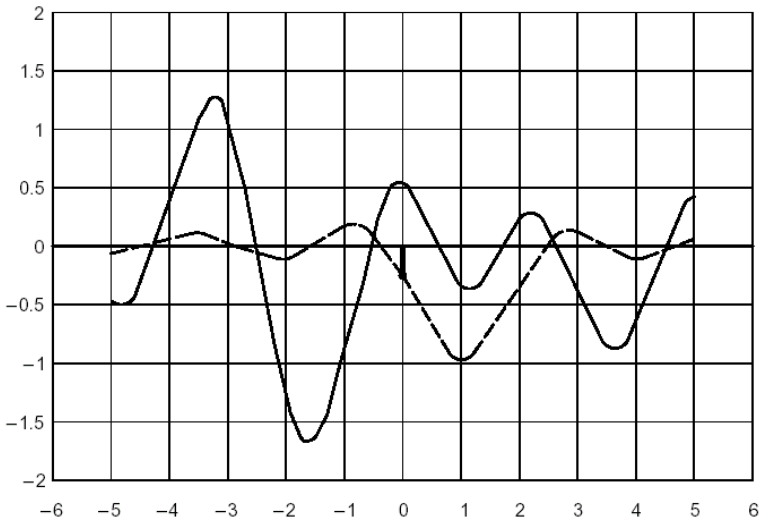

Figura 4 – Contribuição devida a x1 em t=0

Um raciocínio similar explica a contribuição devido à x_1, exceto este tempo que usa o valor da resposta impulsiva deslocada no tempo em t = 0 como ilustrado na figura 4

O valor recebido de r(t=0) é computado somando a contribuição de x_0 mais todos os outros termos ISI; que é, x_{-1}, x_{-2}, x_{-3}, ... Teoricamente, está é uma soma infinita,

mas como é mostrada aqui, a resposta do canal é tipicamente um decaimento exponencial. Então, na pratica, um sistema FIR pode ser usado para modelar um sistema compensado.

Do exemplo acima se pode ver que a enésima amostra recebida é primeiramente influenciada pelo enésimo símbolo transmitido. Contudo, há componentes ISI adicionadas pelo símbolo transmitido anterior e posterior. Os termos devidos aos símbolos anteriores (X_{n-1} e anteriores) são chamados "postcursor" ISI [3] porque o enésimo símbolo transmitido afeta nos símbolos seguintes no enésimo símbolo recebido. A natureza deste ISI pode ser determinada examinando-se a porção direita da resposta impulsiva do sistema. Alternativamente, os termos ISI devidos aos símbolos transmitidos subseqüentes (x_{n+1} e adiante) são chamados precursor ISI [3] porque o enésimo símbolo transmitido influencia os últimos símbolos recebidos até o enésimo. Estes termos ISI são determinados pela forma da porção esquerda da resposta impulsiva do sistema.

3.2 Equalização Adaptativa

Na maioria dos sistemas de comunicação digital ocorre a dispersão temporal do sinal transmitido no canal, fazendo com que dados transmitidos num dado instante venham a interferir com dados transmitidos em outros instantes. Esse fenômeno, chamado de interferência intersimbólica (IIS), provoca a redução da confiabilidade e/ou da taxa com as quais os dados são transmitidos. A fim de minorar a IIS, faz-se uso de equalizadores, normalmente, no receptor. Tais equalizadores são usualmente capazes de corrigir as distorções produzidas pelo canal.

Todavia, tendo em vista a necessidade de equalizar canais desconhecidos ou variantes no tempo, faz-se imperativo o uso de equalizadores adaptativos. Esses equalizadores são usualmente implementados na forma de filtros digitais com resposta finita ao impulso (FIR – *Finite Impulse Response*) e/ou resposta infinita ao impulso (IIR – *Infinite Impulse Response*). Estes são adaptados por meio de uma seqüência de treinamento conhecida no receptor que é tida como resposta desejada do equalizador. A diferença entre a seqüência de treinamento e a saída do equalizador é utilizada para ajustar seus parâmetros. Contudo, não seria necessário fazer a transmissão dos dados caso se soubesse *a priori* seus valores. Desta forma, o equalizador, após o fim da seqüência de treinamento, é chaveado para o modo de decisão direta (DD – *Decision-Directed*), onde a decisão sobre a saída do próprio equalizador é

utilizada como sinal de referência. Existem equalizadores ou, mais propriamente, algoritmos em que não se faz necessária a presença da seqüência de treinamento. Isto tem como vantagem a possibilidade de se aumentar a taxa sem se enviar dados já conhecidos. Neste tópico, iremos ver alguns tipos de estruturas utilizadas na equalização de canais e alguns algoritmos utilizados na adaptação dos parâmetros desses filtros. Na seção 3.2.1, apresentamos os conceitos de sistema de comunicação e o modelo em banda base. A seção 3.2.2 apresenta o filtro de Wiener, o método da descida mais íngreme e os algoritmos LMS (*Least Mean Square*) e RLS (*Recursive Least Squares*).

3.2.1 Sistemas de Comunicação e Modelo em Banda Base

Antes de iniciar a discussão sobre equalização adaptativa, vamos mostrar alguns conceitos básicos de um sistema de comunicação simplificado. A figura 5 mostra o esquema de blocos de um sistema de comunicação. A fonte de informação pode ser um sinal de voz amostrado, algum texto para ser transmitido, por exemplo, via Internet, etc. O codificador de fonte é responsável por eliminar redundância de informação gerada pela fonte, mas desde que seja possível realizar o procedimento inverso e re-obter a seqüência da fonte. Por sua vez, o codificador de canal insere redundância de forma controlada na seqüência resultante do codificador de fonte, visando explorá-la no receptor e, desta forma, corrigir erros que venham a ser inseridos pelo canal.

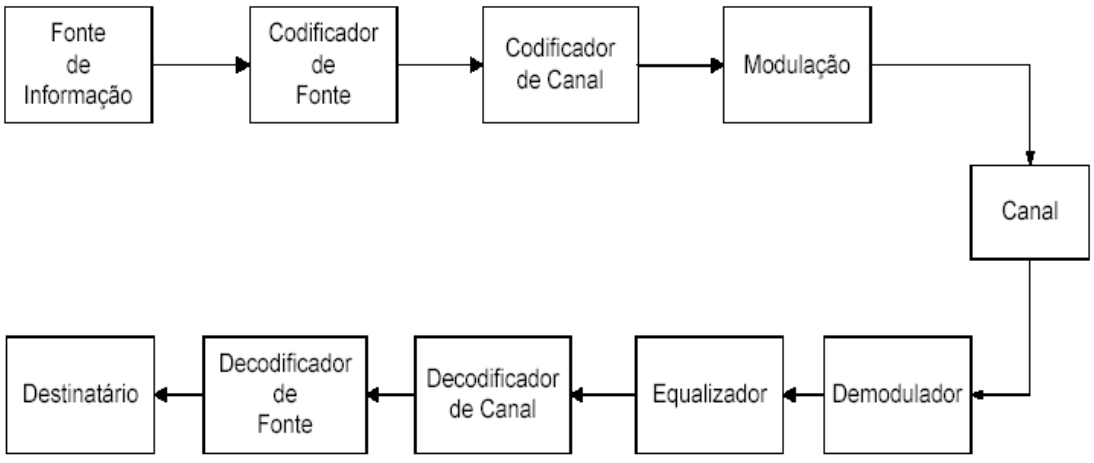

Figura 5 – Sistema de comunicação digital genérico.

O modulador recebe a seqüência do codificador de canal e transforma esses dados em uma forma de onda adequada para ser transmitida no canal. O canal de comunicação é o meio físico pelo qual são transmitidas as informações. Pode ser, por exemplo, o ar ou o espaço, no caso de comunicações do tipo sem fio, um fio, no caso do

telefone fixo, ou uma fibra ótica. Contudo, qualquer que seja o meio, o sinal sempre será corrompido por ruídos que são flutuações aleatórias na amplitude do sinal. Esse ruído pode ser causado por componentes eletrônicos, o caso do ruído térmico, motores a combustão ou por fenômenos como raios etc. Além disso, o canal pode distorcer as formas das ondas transmitidas, atenuando-lhe certas freqüências. No fim do processo de recepção, o demodulador recebe as formas de onda corrompidas pelo canal e as transforma em estimativas da seqüência transmitida. Essa seqüência é tratada pelo equalizador, de modo a compensar as distorções impostas pelo canal. A seqüência tratada é passada ao decodificador de canal que utiliza a redundância do codificador correspondente para corrigir eventuais erros, se possível. Finalmente, a seqüência resultante do processo de decodificação de canal é passada ao decodificador de fonte que faz o processo inverso do codificador de fonte e passa a seqüência ao destinatário.

Agora consideraremos o modelo em banda-base do sistema descrito na figura 5. Nesse modelo, o conjunto modulador-canal-demodulador pode ser considerado como um filtro digital, ao qual referenciaremos, por simplicidade, como sendo apenas o canal. Além disso, usualmente essa combinação pode ser bem modelada por um filtro do tipo FIR (*Finite Impulse Response*). A figura 6 representa o modelo em banda-base.

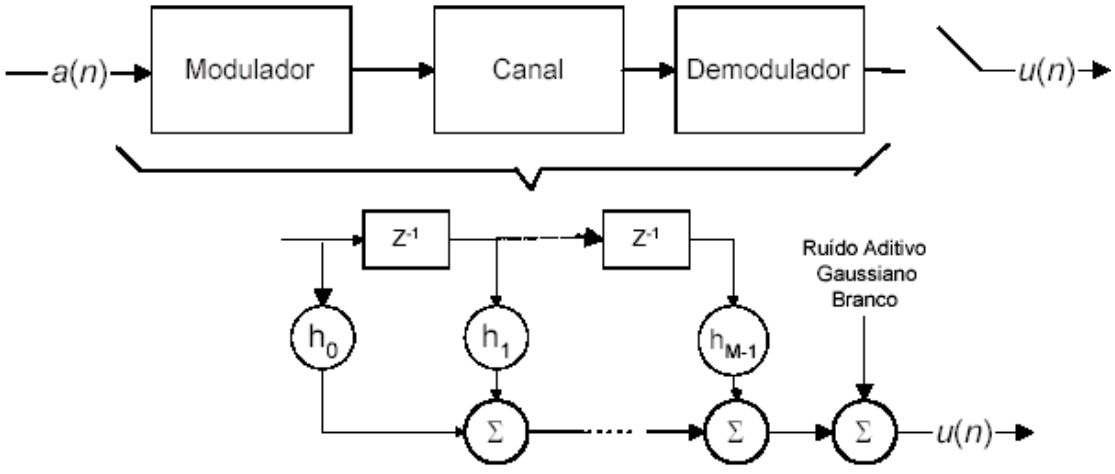

Figura 6 – Modelo em banda-base

Denominaremos por h_k os coeficientes do canal, que podem, ou não, variar com o tempo. Caso não variem, chamamos o canal de invariante no tempo. Caso contrário, ele é denominado de variante no tempo. Ainda, à saída do filtro digital, soma-se um ruído aditivo Gaussiano branco (AWGN). A distribuição Gaussiana do ruído foi escolhida porque representa a forma de ruído que mais reduz a capacidade do canal,

conceito que foi desenvolvido por Shannon. Capacidade do canal significa quantos bits/segundo o canal comporta transmitir, para que seja possível alcançar probabilidade de erro tão pequena quanto desejada no receptor, através de codificação de canal. Veremos na seção 3.2.2 como conceber filtros que minimizem as distorções impostas pelo canal.

3.2.2 Filtro de Wiener, Método da Descida Mais Íngreme e Algoritmos LMS e RLS

Para inverter as distorções impostas pelo canal, geramos um equalizador tal que a transformada Z é igual a:

$$\mathbf{W(z)} = \frac{1}{\mathbf{H(z)}} \quad \textbf{(12)}$$

onde $H(z)$ é a transformada Z do canal e $W(z)$ é a transformada Z do equalizador. Tal método é denominado de *Zero-Forcing* (ZF). Contudo, esse equalizador possui a desvantagem de amplificar demais o ruído, caso o canal possua nulos espectrais. A este fenômeno se dá o nome de *noise-enhancement*, fator este que pode reduzir consideravelmente o desempenho do sistema. Todavia, existem outras formas de se obter um equalizador levando em conta a presença do ruído. Esta técnica é chamada de filtragem de Wiener.

O filtro de Wiener é um método para obtenção dos parâmetros ótimos para um filtro linear discreto onde, ao se levar em conta a potência do ruído, verifica-se uma redução no fenômeno de *noise-enhancement*. O filtro de Wiener faz uso do critério da minimização do erro quadrático médio, cujo valor é obtido a partir da diferença do sinal desejado e da saída do equalizador. Contudo, a solução de Wiener demanda uma inversão matricial que pode ser muito custosa computacionalmente, especialmente quando o equalizador possui muitos coeficientes. Daí surgem métodos iterativos como o da descida mais íngreme, que utilizam o vetor gradiente do critério para obter os coeficientes até convergir para a solução de Wiener. O algoritmo adaptativo conhecido como LMS parte do mesmo princípio utilizando uma aproximação estocástica do verdadeiro vetor gradiente. Deste modo, o LMS se torna excepcionalmente simples, do ponto de vista computacional, destacando-se também por sua robustez e sendo normalmente utilizado como referência de desempenho.

3.2.2.1 Filtro de Wiener

Nesta subseção, vamos definir o filtro de Wiener. Para tanto, definamos os seguintes vetores:

$\mathbf{u}(\mathbf{n}) = [\mathbf{u}(\mathbf{n}) \quad \mathbf{u}(\mathbf{n}-1) \quad \cdots \quad \mathbf{u}(\mathbf{n}-\mathbf{N}+1)]^{\mathbf{T}}$ vetor de entrada do filtro linear de ordem

N-1 e *u*(*n*) é a *n*-ésima saída do canal; e

$\mathbf{w} = [\mathbf{w}_0 \quad \mathbf{w}_1 \quad \cdots \quad \mathbf{w}_{\mathbf{N}-1}]^{\mathbf{T}}$ vetor de coeficientes do filtro linear.

A *n*-ésima saída do filtro, *y(n)*, é dada pela convolução do sinal da saída do canal pelos coeficientes do filtro:

$$\mathbf{y}(\mathbf{n}) = \mathbf{w}^{\mathbf{H}}\mathbf{u}(\mathbf{n}) \quad \textbf{(13)}$$

Seja *d*(*n*) o *n*-ésimo sinal desejado. Desta forma, o erro de estimação de *d*(*n*) é dado por:

$$\mathbf{e}(\mathbf{n}) = \mathbf{d}(\mathbf{n}) - \mathbf{y}(\mathbf{n}) \quad \textbf{(14)}$$

O erro *e*(*n*) é uma variável aleatória dada à natureza estocástica de *y(n)*. Então, como critério de otimização do filtro, escolhemos minimizar o erro quadrático médio, ou seja:

$$\mathbf{J} = \mathbf{E}[\mathbf{e}(\mathbf{n})\mathbf{e}^{*}(\mathbf{n})] = \mathbf{E}\left[|\mathbf{e}(\mathbf{n})|^{2}\right] \quad \textbf{(15)}$$

onde $\mathbf{E}[\cdot]$ é o operador espectância. Como desejamos obter o menor valor de *J*, devemos calcular o gradiente do critério em relação a **w**:

$$\nabla_{\mathbf{w}}\mathbf{J} = -2\mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{e}^{*}(\mathbf{n})] \quad \textbf{(16)}$$

e igualando este gradiente a zero, obtemos as equações de Wiener-Hopf:

$$\mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{u}^{\mathbf{H}}(\mathbf{n})]\mathbf{w}_{\mathbf{opt}} = \mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{d}^{*}(\mathbf{n})] \quad \textbf{(17)}$$

Mas,

$\mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{u}^{\mathbf{H}}(\mathbf{n})]$ é a matriz auto-correlação **R**;

$\mathbf{E}[\mathbf{u}(\mathbf{n})\mathbf{d}^{*}(\mathbf{n})]$ é o vetor de correlação cruzada **p**; e

$\mathbf{W}_{\mathbf{opt}}$ é vetor ótimo dos coeficientes.

Então, reescrevendo (17) na forma matricial, temos:

$$\mathbf{R}\mathbf{w}_{\mathbf{opt}} = \mathbf{p}$$

Assumindo que a matriz de autocorrelação admite inversa, obtemos $\mathbf{w}_{opt}$:

$$\mathbf{w}_{\mathbf{opt}} = \mathbf{R}^{-1}\mathbf{p}$$

A função custo J resulta num parabolóide, possuindo, por conseqüência, somente um único mínimo.

3.2.2.2 Método da Descida Mais Íngreme

Como já dissemos, a solução de Wiener pode ser muito custosa computacionalmente, uma vez que ela exige a inversão de uma matriz de dimensão igual à ordem do filtro. Desta forma, usa-se um método iterativo baseado no gradiente para se chegar a solução de Wiener, sem que seja necessária a inversão da matriz de auto-correlação.

O nome método de descida mais íngreme provém de que o gradiente $\nabla_{\mathbf{w}}\mathbf{J}$ é um vetor que aponta para a direção de maior crescimento da função J. Assim, como desejamos minimizar J, estabelecemos que o vetor de pesos caminhe na direção oposta à do gradiente $-\nabla_{\mathbf{w}}\mathbf{J}$, ou seja, na direção onde a função J decresce mais rapidamente. Desta forma, temos:

$$\mathbf{w}(\mathbf{n}+1) = \mathbf{w}(\mathbf{n}) + \frac{1}{2}\mu[-\nabla_{\mathbf{w}}\mathbf{J}(\mathbf{n})], \textbf{ (18)}$$

onde μ é o passo de adaptação. Como

$$\nabla_{\mathbf{w}}\mathbf{j}(\mathbf{n}) = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(\mathbf{n}), \textbf{ (19)}$$

obtém-se a fórmula de atualização do vetor de coeficientes:

$$\mathbf{w}(\mathbf{n}+1) = \mathbf{w}(\mathbf{n}) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(\mathbf{n})] \textbf{ (20)}$$

Este método apresenta um único ponto de mínimo, dado pela solução de Wiener. Contudo, esse ponto só é atingido se respeitarmos certas condições sobre o passo de adaptação. Por meio da análise de autovalores da matriz de auto-correlação, é possível obter a condição sobre μ para a qual se garante a convergência do algoritmo:

$$0 < \mu < \frac{2}{\lambda_{\mathbf{R}_{\text{máx}}}}, \textbf{ (21)}$$

onde $\lambda_{R_{máx}}$ é o maior autovalor da matriz de auto-correlação **R**.

3.2.2.3 Algoritmo LMS

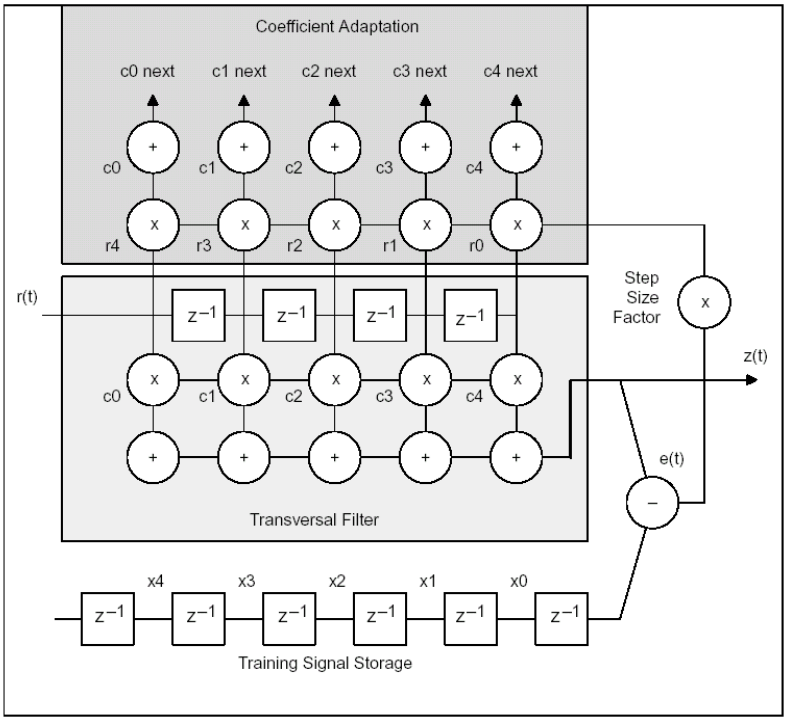

Figura 7 – Resposta computacional do filtro

O algoritmo LMS nada mais é que uma versão estocástica do método da descida mais íngreme, pois não tem sentido prático obter exatamente o vetor gradiente, uma vez que seria preciso um conhecimento *a priori* da matriz auto-correlação **R** e do vetor de correlação cruzada **p**. Assim, o gradiente deve ser estimado a partir dos dados recebidos. A forma mais fácil e imediata de fazê-lo é utilizando uma estimativa instantânea de **R** e de **p** que são dadas respectivamente por:

$$\hat{\mathbf{R}}(\mathbf{n}) = \mathbf{u}(\mathbf{n})\mathbf{u}^{\mathbf{H}}(\mathbf{n}) \quad \textbf{(22)}$$

e

$$\hat{\mathbf{p}}(\mathbf{n}) = \mathbf{u}(\mathbf{n})\mathbf{d}^{*}(\mathbf{n}) \quad \textbf{(23)}$$

Desta maneira, substituindo (22) e (23) em (19), obtemos a estimativa instantânea do vetor gradiente:

$$\hat{\nabla}_{\mathbf{w}}\mathbf{j}(\mathbf{n}) = -2\mathbf{u}(\mathbf{n})\mathbf{d}^{*}(\mathbf{n}) + 2\mathbf{u}(\mathbf{n})\mathbf{u}^{\mathbf{H}}(\mathbf{n})\mathbf{w}(\mathbf{n}) \quad \textbf{(24)}$$

Note-se que este vetor gradiente pode ser obtido derivando-se a estimativa instantânea do erro quadrático $|\mathbf{e}(\mathbf{n})|^2$.

Substituindo o vetor gradiente, obtido em (24), no método da descida mais íngreme (18), obtemos a relação de inovação dos pesos do algoritmo LMS:

$$\mathbf{w}(\mathbf{n}+1) = \mathbf{w}(\mathbf{n}) + \mu\mathbf{u}(\mathbf{n})\left[\mathbf{d}^{*}(\mathbf{n}) - \mathbf{u}^{\mathbf{H}}(\mathbf{n})\mathbf{w}(\mathbf{n})\right] \textbf{ (25)}$$

Desta forma, cada iteração do algoritmo LMS é feita seguindo os seguintes passos:

1. Calcular a saída do filtro:

$$\mathbf{y}(\mathbf{n}) = \mathbf{w}^{\mathbf{H}}(\mathbf{n})\mathbf{u}(\mathbf{n}) \textbf{ (26)}$$

2. Calcular o erro:

$$\mathbf{e}(\mathbf{n}) = \mathbf{d}(\mathbf{n}) - \mathbf{y}(\mathbf{n}) \textbf{ (27)}$$

3. Atualização dos coeficientes:

$$\mathbf{w}(\mathbf{n}+1) = \mathbf{w}(\mathbf{n}) + \mu\mathbf{u}(\mathbf{n})\mathbf{e}^{*}(\mathbf{n}) \textbf{ (28)}$$

Do mesmo modo que o método da descida mais íngreme, o algoritmo LMS também possui condições sobre o passo de adaptação para que o algoritmo seja estável.

Para tornar o problema matematicamente tratável, formulam-se, normalmente, várias hipóteses que na prática são pouco realistas. Contudo a análise pode dar uma idéia de qual valor máximo de passo o algoritmo comporta. Usualmente, adota-se um cálculo mais conservador, onde o passo fica restrito a:

$$0 < \mu < \mathbf{tr}\left[\mathbf{R}\right] \textbf{ (29)}$$

onde tr[**R**] é a soma dos elementos da diagonal principal da matriz autocorrelação. Esse valor equivale à soma dos autovalores da matriz, ou à potência do vetor de entrada $\left(\mathbf{u}^{\mathbf{H}}\mathbf{u}\right)$. O algoritmo LMS oscila em torno e converge em média para a solução de Wiener. A oscilação do algoritmo está diretamente ligada ao valor do passo de adaptação. Quanto maior o passo, maior é o ruído do vetor estocástico.

3.2.2.4 Algoritmo RLS

O algoritmo RLS (*Recursive Least Squares*) baseia-se na minimização de uma função custo decorrente da soma ponderada do valor absoluto do erro quadrático. O RLS faz uso da função custo sem o operador esperança, como no filtro de Wiener. Contudo, as equações a que chegamos são semelhantes às de Wiener-Hopf, onde, ao invés de usarmos as funções autocorrelação e correlação cruzada, utilizam-se estimativas temporais das mesmas, que são atualizadas iterativamente.

O RLS demanda a inversão da estimativa da matriz de autocorrelação, o que faz com que a complexidade computacional fique proporcional a N^2, onde N é o número de coeficientes do filtro. Portanto, é extremamente custoso, em termos computacionais. Todavia, é possível evitar a inversão dessa matriz aplicando-se o *lema de inversão de matrizes* ou *identidade de Woodbury*. Este lema permite a estimação recursivamente da inversa da matriz, sem que seja necessário realizar a operação de inversão. Com este artifício, a complexidade computacional do RLS fica proporcional a N^2. Note-se que esta complexidade ainda é relativamente alta quando comparada à complexidade do LMS que é da ordem de N.

Através da saída do equalizador (26) e do erro em relação ao sinal desejado (27), as equações de atualização do vetor de pesos $\mathbf{w}$ e da estimativa da inversa da matriz autocorrelação $\mathbf{R}_{\mathbf{D}}^{-1}$ são dadas por:

$$\begin{aligned}
\mathbf{g}(\mathbf{n}) &= \frac{\lambda^{-1}\,\mathrm{R}_{\mathrm{D}}^{-1}(\mathbf{n}-1)\mathbf{u}(\mathbf{n})}{1+\lambda^{-1}\mathbf{u}^{\mathbf{H}}(\mathbf{n})\mathrm{R}_{\mathrm{D}}^{-1}(\mathbf{n}-1)\mathbf{u}(\mathbf{n})} \\
\mathbf{w}(\mathbf{n}+1) &= \mathbf{w}(\mathbf{n})+\mathbf{g}(\mathbf{n})\mathbf{e}^{*}(\mathbf{n}) \qquad \textbf{(30)} \\
\mathbf{R}_{\mathbf{D}}^{-1}(\mathbf{n}) &= \lambda^{-1}\left[1-\mathbf{g}(\mathbf{n})\mathbf{u}^{\mathbf{H}}(\mathbf{n})\right]\mathbf{R}_{\mathbf{D}}^{-1}(\mathbf{n}-1)
\end{aligned}$$

onde $\mathbf{g}$ é o vetor ganho, λ é o fator de esquecimento e $\mathbf{R}_{\mathbf{D}}^{-1}(0)$ é igual a $\delta^{-1}\mathbf{I}$, sendo que δ é uma constante positiva pequena.

O fator de esquecimento, como o próprio nome diz, é usado para que o RLS esqueça os dados mais antigos como, por exemplo, a inicialização de $\mathbf{R}_{\mathbf{D}}^{-1}$ .Normalmente, valores em torno ou maiores que 0,98 são usados para canais invariantes ou pouco variantes. Quando o canal é variante, este valor tende a cair, podendo chegar a valores próximos a 0,9. Se utilizados em canais invariantes, valores baixos de λ podem gerar um erro residual considerável, visto que $\mathbf{R}_{\mathbf{D}}^{-1}$ não é uma boa estimativa da verdadeira matriz autocorrelação, ou ainda, pode resultar em instabilidade do algoritmo.

O algoritmo RLS proporciona a solução ótima para a matriz $\mathbf{R}_{\mathbf{D}}^{-1}(\mathbf{n})$ a cada iteração. O vetor ganho sempre aponta para a solução ótima, o que torna o RLS independente das estatísticas do canal. Tal fato faz com que o RLS difira do LMS no sentido de que este tende a ser mais lento, quanto mais correlacionado estiver o sinal à saída do canal.

Existem ainda versões do RLS, chamadas de FLS (*Fast Least Squares*), com custo diretamente proporcional a *N*. Não serão tratadas aqui por envolverem dificuldades no que se refere à estabilidade numérica.

3.3 O Kit DSK C6211/6711

3.3.1 Introdução

O DSK C6711 representa uma linha de baixo custo que resgata o conceito de facilidade na implementação de placas de desenvolvimento, com uma características de alta-performance do DSP de ponto flutuante TMS320C6711, capaz de aferir 900 milhões de operações em ponto-flutuante por segundo (MFLOPS).

O DSK é uma interface de porta paralela que permite a Texas, seus clientes e suas terceiras partes eficientemente desenvolverem e testarem aplicações para o DSP C6711. Este DSK consiste de uma placa de circuito-impresso baseada no C6711 que poderá servir como projeto de referencia de hardware para os produtos dos clientes Texas. Com um extenso suporte a software tanto no PC como no DSP que incluem ferramentas Texas incorporadas.

3.3.2 Características chaves para o kit DSK TMS320C6711

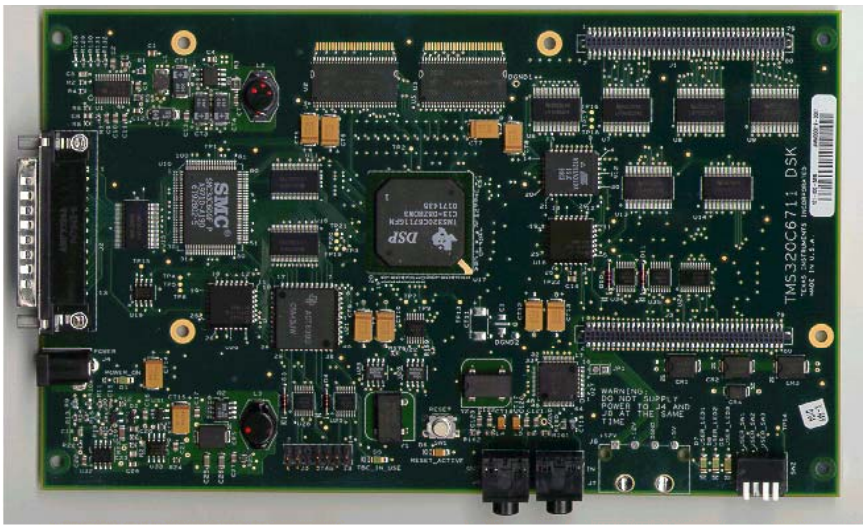

Figura 8 – Vista Real da placa do Kit DSK

O C6711DSK tem as seguintes características:

- DSP C6711, trabalhando a 150-MHz, capaz de executar 900 milhões de operações de ponto-flutuante por segundo (MFLOPS);
- Suporte a duplo oscilador; CPU a 150MHz e interface de memória externa (EMIF) a 100MHz;

- Controlador de porta paralela (PPC) com interface para porta padrão presente em PC's (EEP ou suporte bi-direcional SPP);
- 16M Bytes de memória SDRAM de 100 MHz;
- 128K Bytes de memória não volátil programável e memória ROM apagável;
- Porta E/S de 8-bit mapeado em memória;
- Emulação embutida JTAG através da porta paralela e suporte ao XDS510;
- Acesso através da porta acesso pelo PC para toda a memória do DSP através da porta paralela;
- Codec de áudio de 16-bit;
- Chave de regulagem de tensão na placa para 1,8 e 3,3 volts de corrente continua;
- Suporte a placa-filha através da expansão de memória e conectores periféricos.

3.3.3 Diagrama de blocos do DSK C6711

O diagrama de blocos funcional básico e as interfaces do DSK são mostrados na figura 9.

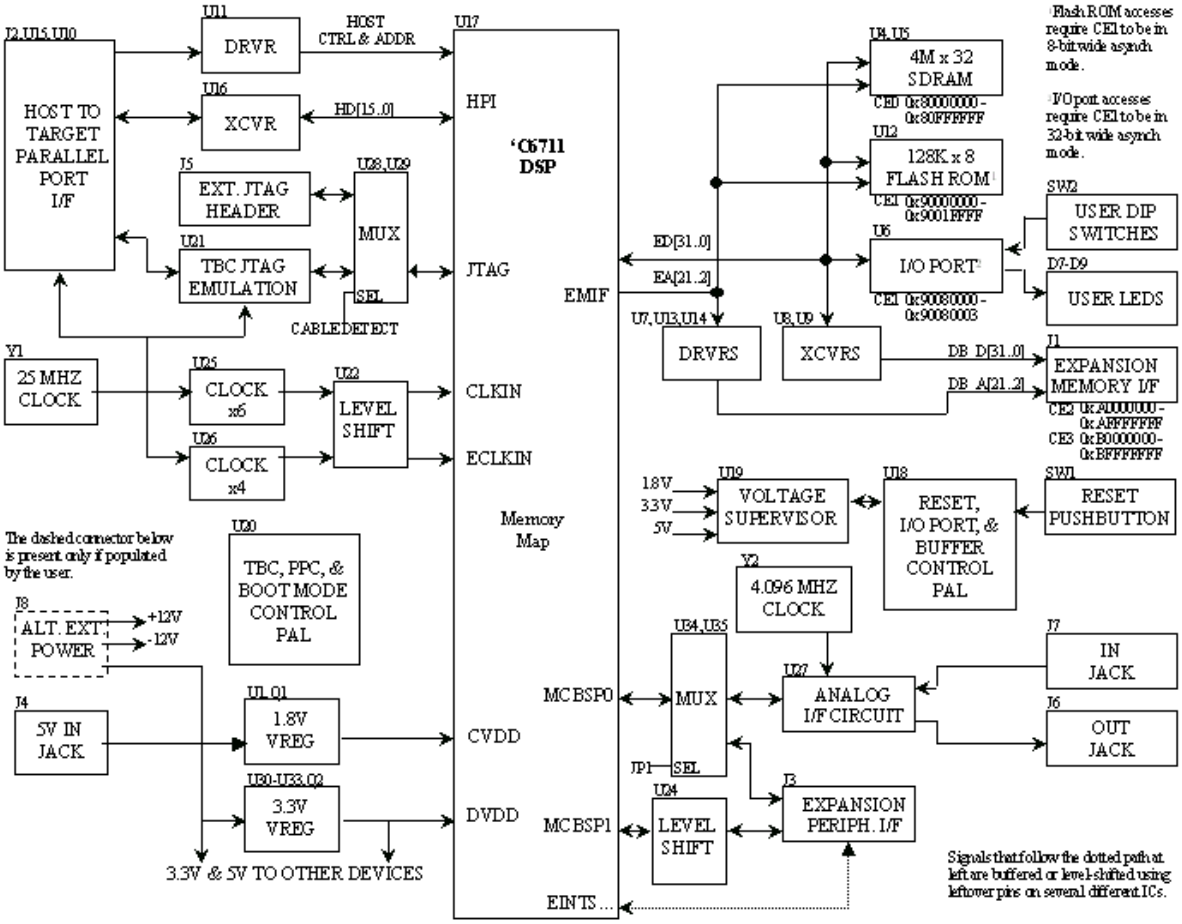

Figura 9 – Diagrama em Blocos das funcionalidades do DSK

3.3.4 Controles de usuário e indicadores

O C6711DSK tem os seguintes controles de usuário e indicadores:

- Emulação JTAG (embutida ou externa);
- Endereçamento de memória Little- or big-endian;
- LED indicador de força;
- LED indicador de reset ativo;
- LED indicador de TBC-in-use
- Três indicadores LED definidos por usuário

3.3.5 Interfaces externas

O C6711DSK tem os seguintes conectores:

- Interface de porta paralela de 25 pinos (macho DB-25) padrão IEEE Standard 1284;
- Conector de 5 Volts externo de 2,5 mm;
- Pente de 14 pinos externo JTAG;

- Jack de 3,5 mm de entrada de áudio;
- Jack de 3,5 mm de saída de áudio;
- Dois conectores a placa-filha de 80 pinos (.050 polegadas) .

3.4 Introdução ao RTDX

3.4.1 Motivação

No passado, projetistas de aplicações para DSP's tinham garantido os seus dados parando a aplicação da placa em pontos de paradas designadas por eles para ler registradores e outros locais de armazenamento de dados. Esta pratica não é somente incomoda, ela pode trazer dados equivocados porque rende apenas um instante "fotográfico" obtido pela obstrução repentina de uma aplicação rodando em alta velocidade – uma leitura que pode não representar uma visão acurada da contínua operação do sistema. Esta seção apresenta a tecnologia de analise instrumental da Texas aplicada a DSP's, chamada RTDX. O RTDX oferece aos desenvolvedores uma continua troca de dados bidirecional em tempo real com um mínimo de perturbação na aplicação. Porque o RTDX usa o caminho de dados JTAG, ele pode ser suportado em todos os processadores da TI sem necessitar de nenhum periférico particular E/S e, pode também usar outras interfaces de dados ao invés de ou em adição ao JTAG. O RTDX mostra os dados usando qualquer pacote habilitado de visualização OLE.

3.4.2 Usos para o RTDX

O RTDX capacita analises fáceis para muitos sistemas conhecidos ou em fase de reconhecimento. Projetistas em telecomunicação sem fios podem capturar a saída de seus algoritmos de vocoder's para checar as implementações de suas aplicações.

Sistemas de controle integrados também são beneficiados. Por exemplo, Aplicações de dispositivos de controle de discos rígidos podem ser testadas sem sinais impróprios que serviriam para a quebra do servo-motor. Projetistas de mecanismos de controle poderiam analisar a mudança de condições tal qual o aquecimento e condições ambientais enquanto a aplicação de controle está rodando.

3.4.3 Caminho de dados para o RTDX

As duas últimas letras do acrônimo RTDX para "troca de dados – Data Exchange" – precisamente é o que esta tecnologia dispõe. Dados podem ser mandados da placa para o PC e do PC para a placa.

O suporte básico para a troca de dados é fornecido pela emulação lógica inclusa em cada DSP Texas, portanto os dados podem ser transferidos usando a mesma conexão simples de depuração JTAG usada para todos as outras operações de depuração e controle. Este fato significa que o RTDX pode ser suportado em todo processador sem assumir nenhum periférico particular E/S. Os usuários estão livres para usar o caminho de dados que melhor lhe provenha altas taxas de comunicação ou uma comunicação mais seguras em seus produtos finais.

Em adição para fornecer uma conexão universal, a interface JTAG e a emulação lógica também servem para minimizar a capacidade de perturbação do RTDX. Emulações avançadas permitem que os dados sejam transmitidos para o PC como uma atividade em segundo plano através de uma perturbação mínima para o processador. Em processadores com emulação avançada, cada palavra de dados é transferida diretamente da memória. O *hardware* pode apropriar-se de um ciclo para cada palavra ou fazer um acesso direto à memória dependendo da implementação do processador.

3.4.4 DSP para Computador

O depurador da Texas controla o fluxo de dados entre o PC e a placa (acompanhe pela figura 10). Os dados fluem da aplicação na placa até a biblioteca de interface do usuário e, depois, para a interface de comunicação indo diretamente ao depurador Texas disponível no PC. As aplicações da placa chamam rotinas na biblioteca de interface de usuário que armazenam os dados e passam eles para a interface de comunicação. Este conjunto de dados é então mandado para o depurador pelo caminho da interface JTAG. O depurador grava os dados em um arquivo de "log" no PC

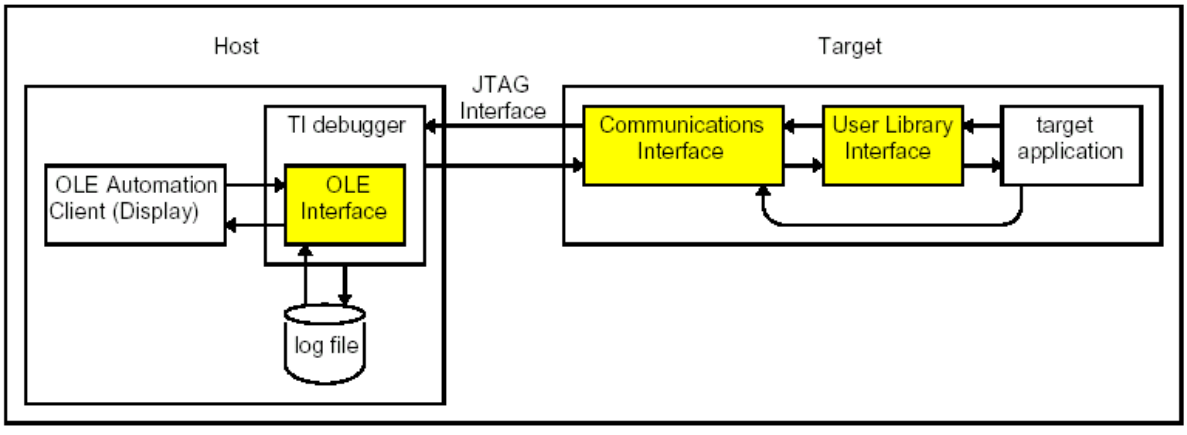

Figura 10 – Fluxo de dados do RTDX

No PC, o depurador suporta a automatização do OLE. Qualquer aplicação do PC que é um Cliente de automatização OLE (estes escritos em Visual Basic ou Visual C++, por exemplo) podem acessar os dados no arquivo de "log" RTDX ou mandar os dados para a placa através da interface OLE.

3.4.5 Computador para DSP

O caminho de dados do cliente automatizado OLE para o depurador Texas. Os *buffers* de depuração de dados no PC. Quando o depurador recebe uma requisição de leitura da aplicação da placa, o dados são enviados para a interface de comunicação na placa através de uma interface JTAG. A interface de comunicação passa os dados para a biblioteca de interface de usuário que manda então os mesmos para a aplicação da placa.

3.4.6 Sumário

A habilidade de continuamente monitorar a performance das aplicações para do DSP TMS320 – como a analogia de se ver através de um pára-brisa de um carro – economiza tempo de maneira significativa para os desenvolvedores da chamada "terceira parte" e usuário finais. Para rever os benefícios chaves do RTDX:

- Dispõem continuamente da troca de dados bidirecional sem parar a aplicação;
- Funções em tempo real com o mínimo de perturbação nas aplicações;
- Mostra dados usando o pacote habilitado de visualização – OLE preferido pelo usuário
- É fácil para programar em ambos os ambiente no DSP e no PC
- É disponibilizado sem custo adicional nos DSP's agregando um valor substancial às soluções para os DSP's da Texas

3.5 Code Composer Versão 1.2

O code composer é uma ferramenta de desenvolvimento em ambiente Windows que integra vários mini-aplicativos que são usados em diversos momentos do desenvolvimento de forma transparente ao usuário. Assim no momento da compilação, há um caminho bem definido entre a passagem do projeto escrito em linguagem C/C++ para o produto final que é o chamado arquivo COFF – "Comand Output File Format" de extensão .out.

Neste ambiente de desenvolvimento podemos definir um variado numero de parâmetros que serão usados pelo compilador. Aqui citaremos, dentre outros, diretivas que montarão o código em seções previamente alocadas em memória que poderão corresponder a locais de simples armazenamento de dados até posições que guardaram código executável. Existem também diretivas de inicialização em carga, auto-inicialização, ou simplesmente, de não inicialização de porções de memória reservadas a variáveis globais. Diretivas também que definem tamanho máximo de pilha, de "overhead" de funções, ou melhor, de espaço reservado a mudança e salvamento de contexto na ida para sub-rotinas do código.

Pode-se citar também opções simples conhecidas como diretivas de compilação do pré-processador que podem ser usadas para a montagem de códigos baseados em checagens do compilador da arquitetura usada.

Antigamente o "Code Composer Studio" não existia como uma ferramenta única, o desenvolvedor deveria seguir a utilização de um conjunto de aplicativos do MS-DOS para que o código executável fosse finalmente montado. Inclusive o processo de depuração seguia um caminho tortuoso, pois a depuração era feita neste mesmo ambiente MS-DOS sem muitas facilidades inerentes a interface visual hoje largamente usada.

Com o advento do "Code Composer Studio" a integração aconteceu, e todas as facilidades decorrentes do ambiente criado foram disponibilizadas a um grupo maior de interessados que puderam utilizá-las sem perdas extensivas de tempo em assuntos operacionais simples como carregamento ou geração de arquivos objetos. Este aplicativo forneceu também uma interface "invisível" chamado cliente OLE, uma proposta já existente dentro do sistema operacional Windows. Que não é nada mais do que um modo de aplicativos diferentes poderem trocar informações ou até mesmo dados, além de outras opções que não interessam ao contexto do trabalho, de forma simplificada e até, certo ponto, de forma protegida. Assim um aplicativo gerado, por exemplo, em c++ builder pode agregar no seu código bibliotecas e cabeçalhos (os chamados "headers") para que ele possam interfacear com o cliente disponibilizado em primeira instancia pelo "Code Composer" acessando a placa através do canal RTDX.

3.6 Integração MATLAB/RTDX

3.6.1 Usando chamadas para o RTDX

O que as chamadas podem fazer pela aplicação:

- Enviar e receber dados da memória do processador;
- Mudar as características operacionais do programa;
- Fazer trocas de algoritmos necessárias sem parar o programa ou configurar pontos de parada no código

3.6.2 Introdução ao uso do RTDX no MATLAB

Os esforços no desenvolvimento do processamento digital de sinais começaram com a idéia de processamento de dados; uma área de aplicação, tal como áudio ou comunicação sem fio ou computação multimídia; e uma plataforma ou hardware para processar o sinal. Usualmente estes esforços de processamento envolvem estratégias como filtragem de sinal, compressão e transformações de troca do conteúdo de dados.

Em todos os casos, os desenvolvedores criaram algoritmos de que eles necessitavam para atingir os resultados desejados. Uma vez tendo os algoritmos, os desenvolvedores usam modelos e ferramentas de desenvolvimento para DSP para testar os seus algoritmos, para determinar se o processamento alcança os seus objetivos e se o processamento trabalha na plataforma proposta. O kit de desenvolvimento e os "links" para o RTDX e para o CCS IDE facilitam o trabalho de abstrair os algoritmos do mundo teórico para o mundo real do processador digital no qual o algoritmo irá rodar.

RTDX e os "links" para o CCS IDE dispõem um caminho de comunicação para manipular dados e processar programas no seu DSP alvo. O RTDX oferece troca de dados em tempo real em duas direções entre o MATLAB e o seu processo alvo. Os dados que são mandados para a placa têm poucos efeitos no processo em andamento e a exibição dos dados retidos da placa mostra ao usuário como os algoritmos estão sendo executados em tempo real.

Para introduzir as técnicas e ferramentas avaliadas no kit do desenvolvedor ("Developer's kit") para o uso com RTDX, os procedimentos acima explicitados usam muitos dos métodos no software de ligação para configurar a placa, o DSP, abrir e habilitar os canais, mandar dados para o alvo e limpá-los depois do seu uso. Entre as funções descritas estão:

Da ligação para o CCS IDE

ccsdsp – Cria ligação para o CCS IDE e o RTDX.

cd – Troca o diretório de trabalho do CCS IDE do MATLAB.

open – Carrega os arquivos de programa no CCS IDE.

run – Roda processos no processador alvo.

Da classe do RTDX

close – Fecha a ligação RTDX entre o MATLAB e o seu alvo.

configure – Determina quantos canais de buffer para serem usados e configura o tamanho de cada buffer.

disable – Desabilita a ligação RTDX antes de ser fechada.

display e **disp** – retorna os resultados das funções get e set.

enable – habilita os canais abertos para que se possa usá-los para mandar e receber dados da placa.

isenabled – Determina se os canais estão habilitados para a comunicação RTDX.

isreadable – Determina se o MATLAB pode ler o local de memória especificado.

iswritable – Determina se o MATLAB pode escrever para o alvo.

msgcount – Acha quantas mensagens estão esperando no canal paradas.

open – Abre os canais no RTDX.

readmat – Lê as matrizes de dados da placa para o MATLAB como matriz.

readmsg – Lê uma ou mais mensagens do canal.

writemsg – Escreve mensagens para o alvo do canal.

4 Formulação do Problema

O atual laboratório de processamento de sinais está na sua fase inicial de implantação. Portanto, como é de praxe, estão sendo elaborados novos experimentos a fim de se adequarem no conjunto de roteiros da disciplina, onde substituem roteiros falhos e/ou problemáticos ou apenas se adicionam ao conjunto. Daí o mesmo busca trazer ferramentas que auxiliem o intento e minimizem o esforço tanto do professor como do aluno, procurando ao mesmo tempo maximizar o aprendizado, explorando mais afundo a teoria aplicada a prática, e mantendo a fluidez da experimentação.

Um problema recorrente a este laboratório é instabilidade inerente às novas ferramentas, aqui o kit DSK, que por sua vez representa uma tentativa de baixo custo e flexível da empresa Texas Instruments em deixar acessíveis as suas novas tecnologias aos desenvolvedores e interessados. Esta instabilidade citada pode ser vista como erros na interface de controle e monitoramento do hardware presentes no já citado ambiente de desenvolvimento "Code Composer Studio" e na própria Placa DSK. Também temos um outro grande problema, que é o de que o usuário que está implementado os roteiros de laboratório está sendo obrigado a trabalhar sobre um processador de arquitetura considerada complexa dentro de um ambiente profissional de desenvolvimento sem que este indivíduo tenha tido uma preparação conveniente para tal fim. Dando a impressão de que quem faz o laboratório tem uma preocupação maior em apenas aprender o conteúdo introdutório de como fazer, do que o que realmente é procurado, que é o fazer "Processamento de Sinais" através de uma ferramenta que facilite e não atrapalhe o desenvolvimento que, de preferência, ajude o desenvolvedor a atingir os seus objetivos economizando tempo na geração e depuração de código.

Assim o problema foi equacionado para atingir uma maximização dos resultados em laboratório através de experimentos mais abrangentes e complexos e da minimização do tempo de desenvolvimento e de depuração do código gerado, fornecendo tempo para novas implementações por parte do usuário.

O problema aqui apresentado a respeito da dificuldade do desenvolvedor de código para DSP's já é uma preocupação presente nos planos de grandes empresas como a Texas Instruments e MathWorks que trabalharam conjuntamente na formulação do "ToolBox" especifico para o nosso kit de desenvolvimento parte integrante do aplicativo Matlab (versão 6.1 e 7), o qual possui, concomitantemente, um conjunto bem

abrangente de ferramentas de desenvolvimento para os DSP's da Texas. Podemos citar dentre estas ferramentas as seguintes:

Simulink – com as bibliotecas DSP BlockSet, Developers's kit for TI DSP, Fixed-Point BlockSet entre outras;

Link RTDX – onde não há a necessidade da parada do processamento para a depuração ou para a escrita/leitura de dados da memória ou de registradores, por exemplo;

Link CCS – Code Composer Studio – Comunicação direta com o compilador com geração de código Ansi C proveniente de diagramas de blocos montados no simulink com os blocos pertinentes à aplicação. Além de um conjunto grande de instruções que comandam a operação deste compilador pelo MATLAB. O que é conseguido através do encapsulamento do "Code Composer" pela abstração de objeto "Code Composer" no seu ambiente de trabalho (WorkSpace).

FDATool – Ferramenta de projeto de filtros FIR, IIR quantizados ou não, que exporta arquivos cabeçalhos .h em C com os coeficientes montados em vários formatos para o "Code Composer Studio".

5 Metodologia

A metodologia empregada no desenvolvimento do trabalho seguiu as seguintes diretivas:

- Facilidade de implementação e depuração;
- Integração de ferramentas conhecidas e de grande suporte no desenvolvimento do aplicativo cliente e servidor;
- O tempo necessário de implementação para o estudante;
- Integração entre aplicação e teoria;
- Aproveitamento de material já disponível (kit DSK).

Assim foi dado como primeiro passo a escolha mais sucinta das ferramentas que seriam usadas no experimento, partindo do pressuposto de que já havia um laboratório com kits em uso e de que seria necessário aproveitá-los na nova proposta. Desta forma limitou-se o conjunto de possibilidades de uso de ferramental.

Como próximo passo, buscou-se junto a Texas dos chamados "third-part" (que são a "terceira parte" da Texas) as quais são empresas que se agregam a Texas no desenvolvimento de produtos (software e/ou hardware) para os produtos da primeira. A MathWorks apresentou um suporte muito grande e profissional aos produtos de desenvolvimento da Texas, além de sinalizar uma aliança em longo prazo garantindo suporte ao ferramental utilizado. Além disso, é de conhecimento notório de que as universidades já utilizam o matlab como ferramenta cotidiana na resolução de muitos problemas acadêmicos e até mesmo de mercado pela parceria destas com empresas e/ou incubadoras.

Em seguida detalhou-se como o suporte da MathWorks poderia ser utilizado no roteiro. Daí houve uma abertura das possibilidades disponíveis que são as já apresentadas na caracterização ou formulação do problema. Dentre a que poderia ser um primeiro passo dentro deste conjunto de possibilidades, vislumbrou-se a do uso do link RTDX que apresenta um grau de complicação pequeno no acerto de parâmetros inerente à placa e de obstrução do processamento do kit, já que como citado, o mesmo é um canal de depuração em tempo real que não interfere necessariamente no desenrolar do processamento, ou seja, ele pode atuar em "background".

Com este detalhamento maior da aplicação partiu-se para a execução da implementação. O algoritmo de processamento de sinais foi escolhido como uma aplicação muito abrangente e bem trabalhada pela comunidade acadêmica e pelas

empresas, que é o citado filtro adaptativo ou equalizador de canal. Algoritmo este que despende um esforço computacional razoável no conjunto de aplicações onde é inserido comumente e, portanto, é tomado como critico no aprendizado de qualquer pessoa que queira se interar sobre aplicações de processamento digital.

A proposta então foi mais bem formulada, partindo da premissa que quanto maior fosse a proximidade da implementação com o que se pode encontrar em meios extraclasse correlatos, como por exemplo: as aplicações demonstrativas da Mathworks em seus ToolBoxes e afins ou nos "Application Notes" (Notas de aplicação, refere-se a exemplos de aplicações detalhadas disponibilizadas aos desenvolvedores) da Texas Instruments, maior seria o material de pesquisa para o alicerçamento necessário do experimentador na produção e retenção dos conhecimentos de seus experimentos, bem como o ganho de experiência numa aplicação, muito provavelmente, presente no seu cotidiano futuro como Engenheiro de processamento de sinais. Daí chegamos ao seguinte nível de detalhamento baseados nestes meios já citados e nas premissas apresentadas no inicio desta discussão de metodologia. Primeiro seria gerado no Matlab um sinal senoidal puro de 440 Hertz e somado a um sinal de ruído uniforme disperso na freqüência aplicado a um filtro passa-baixas os quais seriam armazenados num buffer temporário. O Buffer em questão seria levado através do canal RTDX ao DSP que efetuaria o algoritmo de equalização do mesmo junto com a conseqüente filtragem do sinal ruidoso por este mesmo filtro. Daí os coeficientes do filtro de compensação de canal e do sinal filtrado seriam devolvidos ao matlab através do canal RTDX que demonstraria graficamente o filtro desenvolvido no tempo e na freqüência e do tom senoidal, do tom mais ruído limitado em banda e do sinal filtrado pelos coeficientes do equalizador. Aqui teríamos o modelo de canal a ser equalizado caracterizado como o filtro passa-baixas limitador de banda do ruído uniforme gerado pelo Matlab.

Pode-se notar que não foram utilizados, em nenhum momento, os conversores analógico-digital ou digital-analógico no experimento, pois o controle de ganho e ruído nestes conversores é de essencial importância no sucesso do experimento e, justamente são eles os que causariam maiores problemas, sem que estes problemas fossem justificáveis numa aplicação mais acadêmica, já que demandam tempo no seu acerto e não contribuem significativamente na fixação do conteúdo de processamento de sinais digitais. Os mesmos são de maior interesse de quem realmente quer desenvolver uma aplicação comercial e/ou robusta e, têm tempo, dinheiro e material humano para o intento.

Portanto isolamos o processamento digital de sinais num ambiente mais "protegido" de condições dependentes de interfaces de aquisição, que são conhecidamente causadoras de descaracterização dos modelos teóricos, pois estas adicionam erros referentes às condições de canal, como exemplo: variações da função de transferência do canal durante a adaptação, ou até mesmo ruídos impulsivos ou "Jitters" que interfeririam em muito na convergência do equalizador LMS. Desta maneira demos a quem experimenta um ambiente de geração de sinais poderoso, conhecidamente o matlab, e integramos a interface com o DSK de maneira mais intuitiva pela transferência de sinais através de uma "ponte digital" já que contornamos os conversores e repassamos os buffer's diretamente ao processador.

Daqui a diante temos a analise metodológica divida em duas grandes vertentes, a saber: o caminho funcional pelo ambiente Matlab e do caminho pelo DSP através da interface RTDX.

No Matlab o caminho funcional é descrito diagramática e conceitualmente de forma mais simples do que o próximo caminho, já que neste o processamento despendido se limita a gerar o tom senoidal e adicioná-lo ao ruído, convenientemente, limitado em banda segundo uma função de transferência de filtro passa-baixas previamente escolhida a fim de torna a equalização mais rápida e livre de eventualidades inerentes ao algoritmo interativo de convergência do filtro, aqui houve o ajuste do numero de zeros da função de transferência do filtro passa-baixas limitador de banda e o número de coeficientes do modelo de função de transferência do algoritmo de equalização. Em seguida é repassado o buffer formado ao Servidor OLE que é embasado em bibliotecas presentes na ferramenta "Code Composer Studio" que por sua vez fazem todo o interfaceamento com o kit transferindo a informação digital ao DSP.

No caminho pelo DSP temos o efetivo processamento do sinal, onde, numa primeira instancia, o mesmo começa a receber os buffer's que são repassados a ele e, a montar através de sucessivos quadros, o seu buffer interno a ser usado no processo de equalização. Na seqüência, depois de cessada a transferência inicial dos dados, o buffer de erros que representa a diferença entre o sinal adaptado e o sinal recebido é forçado a zero. Numa serie de interações o algoritmo de equalização descreve um caminho convergente à minimização do erro já citado. Produzindo conjuntamente, a cada interação, a resposta do filtro equalizador que compensaria a fonte de distorção presente no sinal ruidoso limitado em banda. Portanto, a cada interação já temos uma resposta do SLID (sistema linear invariante ao deslocamento). Contudo o algoritmo foi

propositalmente desenvolvido para considerar como modelo final o resultado de n interações definidas pelo usuário no momento da chamada do Script Matlab, evitando problemas com decisores de parada que se perdessem caso a convergência não fosse atingida. Assim a resposta repassada pelo DSP ao Matlab mostra a sucessão de interações até o momento em que a enésima interação é atingida pelo equalizador. Como último passo neste caminho funcional o DSP devolve através do mesmo canal de RTDX configurado para o sentido inverso de transmissão o sinal equalizado e os coeficientes do filtro que ele utilizou na equalização.

6 Resultados e Discussões

Na seqüência de ilustrações de 11 a 14 vemos os resultados obtidos pelos programas desenvolvidos pelo presente trabalho. O filtro de equalização, na medida em que as interações se seguem, tem uma melhora na sua resposta, pois o sinal filtrado vai aos poucos se aproximando do tom senoidal de 440 Hertz que carrega a informação do nosso sistema. A resposta tende a melhorar porque os quadros, que se sucedem, tem como estimativas de coeficientes iniciais os coeficientes dos filtros anteriores. Assim, teoricamente, as respostas paulatinamente vão convergindo ao filtro que melhor minimiza a função de erro, aqui a função dos mínimos quadrados.

É interessante notar que os resultados foram mostrados como resultados de quadros inteiros. Porém, os resultados plotados pelo Matlab são demonstrados, em cada quadro, como uma seqüência de blocos que são exatamente o tamanho de nosso filtro LMS equalizador. Assim, no exemplo acima, os quadros de 128 amostras foram subdivididos em dois blocos de 64 cada que derivam, cada um, um filtro equalizador e uma saída filtrada a cada atualização das telas de apresentação. Daí o que se vê ser, em cada figura referida, o produto final da sucessão de blocos em cada quadro. A figura 25 é o resultado de todos os blocos anteriores inseridos em cada quadro transmitido à placa.

Apesar dos resultados não parecerem muito promissores pelo fato de não terem uma resposta equalizada suficientemente precisa, cabe salientar de que foram realizados em apenas 8 interações pelo nosso algoritmo na adaptação do filtro, sem considerar o fato de que o filtro foi desenvolvido com o sinal de informação sendo transmitido simultaneamente. Não havendo período de treino e um sucessivo período de transmissão de informação individualizado. Em simulações realizadas pelo simulink do matlab, usando um ambiente muito próximo ao experimentado, foi tido como valor aproximado de interações o número de 1000 interações até que o filtro obtive-se um erro médio quadrático menor do que 0.005, o que tornou o sinal senoidal bem definido na simulação. Na figura 15, temos o resultado da simulação para 511 iterações usando o mesmo algoritmo LMS utilizado no kit com a diferença de ter sido usado o simulink do MATLAB.

O leitor pode estar se perguntando porque transmitimos ao mesmo tempo sinal e ruído no período de equalização, pois assim estaríamos dando uma informação errada de como é o canal ao algoritmo de equalização. Porém essa estranheza não é tão

bem fudamentada já que quando geramos, por exemplo, num modem um sinal de informação a ser transmitido, inicialmente passa através de um "scrambler" ou embaralhador que gera uma pseudo-seqüência randômica de bits e, em conseqüência, de símbolos que tendem num período longo de transmissão terem o comportamento de um ruído uniforme. Daí o sinal de informação que vai, num período de tempo mínimo, se confundir com o ruído uniforme limitado em banda dando pouca interferência na adaptação do canal, já que em numerosos casos o modem também faz correções do seu filtro de equalização enquanto transmite informação. Em segundo lugar, a relação sinal ruído do buffer de entrada do kit DSK é de quase de 0 dB, porque sinal e ruído tem amplitudes equivalentes, fazendo ainda mais a interferência ser diminuída na equalização. Ao mesmo tempo, o caso de 0 dB de relação sinal-ruído foi escolhido porque temos que dar ao nosso equalizador informação o suficiente para que o mesmo convirja baseado no canal e pouco no sinal de informação.

Os resultados obtidos através da experimentação, descrita no corpo deste trabalho, têm resultados promissores, pois se mostraram resultados reproduzíveis a medida em que não tivemos problemas inesperados de convergência dos filtros já que estes foram escolhidos meticulosamente a fim de tornar a convergência ordenada e sem sobressalto, pois foi escolhido um SLID passa-baixa com a ordem de seus coeficientes equivalente ao do filtro de equalização. Além do adicional de que isolamos informações ruidosas de toda sorte dos buffer's de alimentação de nosso algoritmo pela "ponte digital" criada entre o gerador de informação, aqui o Matlab, e o equalizador propriamente dito, aqui o kit DSK mais apropriadamente o DSP 6711.

O toolbox que liga o matlab ao kit DSK se mostrou robusto, pois forneceu uma base sólida de desenvolvimento de código e de depuração que permitisse uma convivência harmoniosa entre o seu ambiente de trabalho e o do "Code Composer Studio" e a sua integração com o kit.

7 Conclusões

Concluiu-se que o presente trabalho foi bastante proveitoso porque conseguiu unir tanto o desenvolvimento teórico presente no Matlab, pelo fato de que podemos mexer com muita facilidade e desenvoltura com o SLID limitador de banda do ruído bem como o sinal de informação dando a ele a forma desejada, como também com o ramo profissional de desenvolvimento de aplicações de processamento de sinais representado pelo "Code Composer Studio". Onde o mesmo forneceu flexibilidade, robustez e alcance teórico na implementação desenvolvida, fazendo mais perto o aluno de graduação ou de pós aos conhecimentos necessários ao desenvolvimento de aplicações de tempo real em situações criticas do uso do esforço computacional do processador de sinais e, tornando familiar aos mesmos, procedimentos de tratamento do sinal analógico no mundo digital como, por exemplo: efeitos de quantização, operações de multiplicação e soma, operações de normalização, estruturas de filtros e linhas de retardo, entre outras. E conhecimentos práticos no desenvolvimento e depuração de código, já que temos parte do código gerado em Ansi C presente no "Code Composer" que é visto como uma ferramenta adicional ao presente trabalho.

Temos como sugestão a um possível prosseguimento deste trabalho a utilização de outras ferramentas que o matlab dispõe e estão disponíveis com maior integração e amplitude na versão posterior a versão aqui trabalhada, que é a versão 7, que promete entre outras coisas, gerar o projeto inteiro do "Code Composer" a partir do "Simulink", de um diagrama de blocos que pode, por exemplo, configurar os "codecs", ou conversores ADC e DAC do kit, de forma intuitiva com código bem mais maduro, para trabalhar com o mesmo filtro equalizador presente neste trabalho. O canal de equalização poderia ser modelado por um canal de função de transferência variante no tempo junto com ruídos de fase e de amplitude. Existem blocos nesta versão que modelam canais através de inúmeros modelos teóricos que poderiam ser explorados, além de blocos que gerariam um sinal analógico de um trem de bits baseados numa determinada modulação escolhida pelo usuário, como por exemplo 256 QAM e assim, transmitida através do canal já equalizado com a posterior demodulação e conseqüente contagem de erros ou BER – "Bit Error Rate", taxa de erros de bit. As possibilidades são inúmeras já que como defendida aqui a integração entre estes aplicativos pode gerar uma infinidade de aplicações robustas e com tempo de desenvolvimento reduzido.

8 Referências

1. Qureshi, S.,"Adaptive Equalization", *IEEE Communications Magazine*, Março de 1992, pp. 9–16.
2. Peebles, P.*Z., Communication System Principles,* Addison-Wesley, 1976.
3. Samueli, H., Daneshrad, B., Joshi, R., Wong, B., and Nicholas, H., "A 64-Tap CMOS Echo Canceller/Decision Feedback Equalizer for 2B1Q HDSL Transceivers", *IEEE Journal on Selected Areas in Communications*, Vol. 9, Iss: 6 , Agosto de 1991, pp. 839–847.
4. Ziemer, R.E., and Peterson, R.L., *Introduction to Digital Communication,* Macmillan, 1992.
5. Lovrich, A. and Simar, R., "Implementation of FIR/IIR Filters with the TMS32010/TMS32020", *Digital Signal Processing Applications with the TMS320 Family,* Volume 1, Texas Instruments, 1989.
6. *TMS320C5x User's Guide,* Texas Instruments, 1993.
7. Bune, P., "A Low-Effort DSP Equalization Algorithm for Wideband Digital TDMA Mobile Radio Receivers", *International Conference on Communications Conference Record*, Juneho de 1991, pp. 763–767.
8. Svensson, L., "Channel Equalizer for a Digital Mobile Telephone Using Narrow-Band TDMA Transmission", *39th IEEE Vehicular Technology Conference,* Volume 1, 1989, pp. 155–158.
9. Oppenheim, A.V,. and Schafer, R.W., *Discrete Time Signal Processing,* Prentice-Hall, 1989.
10. Proakis, J.G., "Adaptive Equalization for TDMA Digital Mobile Radio", *IEEE Transactions on Vehicular Technology*, Volume 40, No. 2, Maio de 1991.
11. Narasimhan, A., Chennnakeshu, S., and Anderson, J.B., "An Adaptive Lattice Decision Feedback Equalizer for Digital Cellular Radio", *40th IEEE Vehicular Technology Conference*, 1990, pp. 662–667.
12. Shimizaki, Y., Nakai, T., Ono, S., and Kondoh, N., "A Decision Feedback Equalizer With a Frequency Offset Compensating Circuit

for Digital Cellular Radio", *Vehicular Technology Society 42nd VTS Conference*, Volume 2, 1992, pp. 596–599.

13. Liu, Yow-Jong, "Bidirectional Equalization Technique for TDMA Communication Systems Over Land Mobile Radio Channels", *Proceedings, GLOBECOM,* 1991.

14. Jung P. and Baier, P.W., "VLSI Implementation of Soft Output Viterbi Equalizers for Mobile Radio Applications", *Vehicular Technology Society 42nd VTS Conference*, Volume 2, 1992, pp. 577–85.

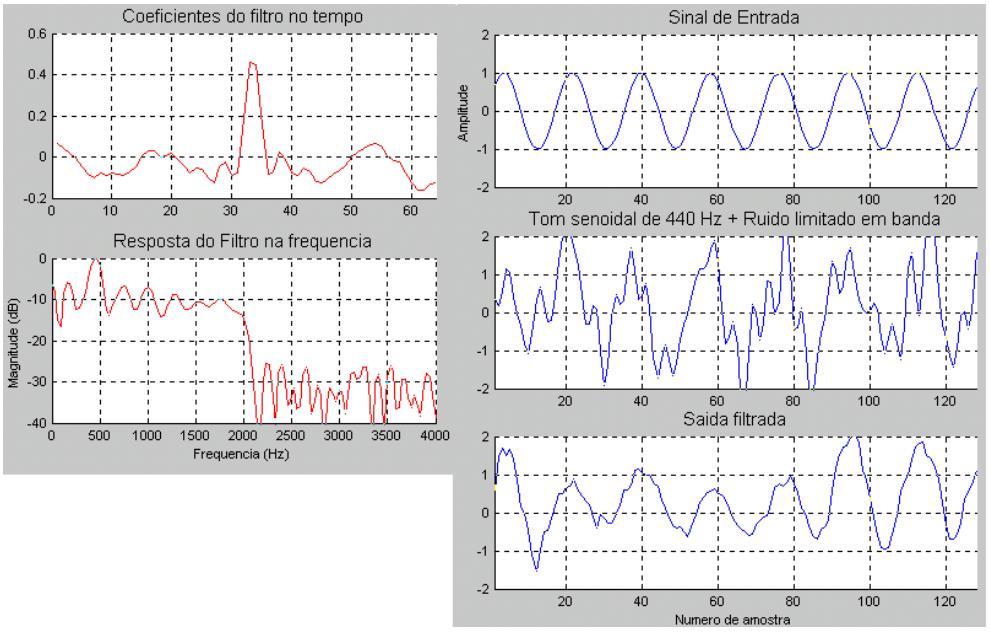

Figura 11 – Resultado do Primeiro Quadro de processamento

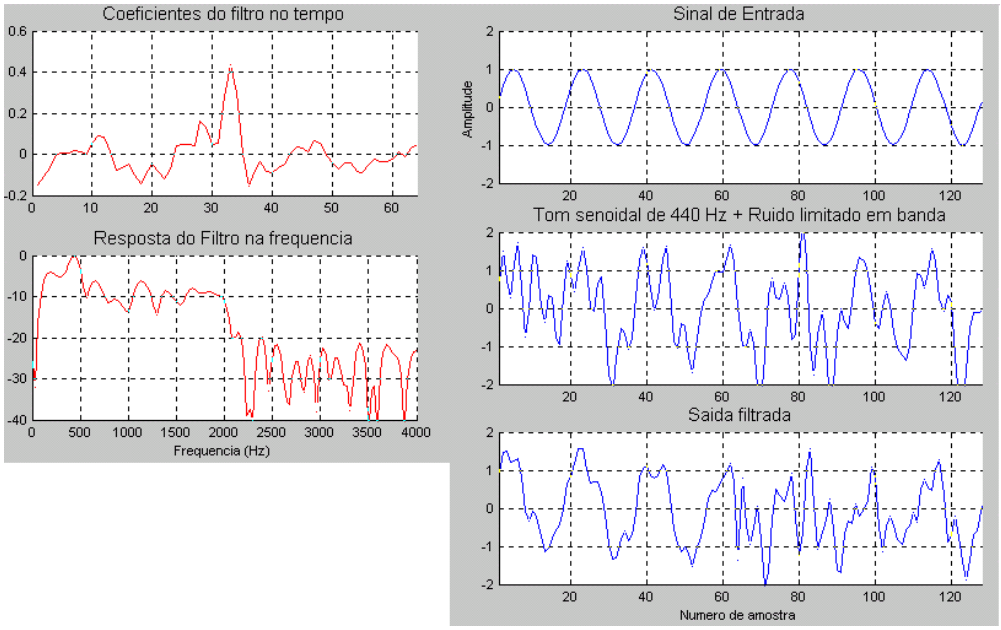

Figura 12 – Resultado do Segundo Quadro de Processamento

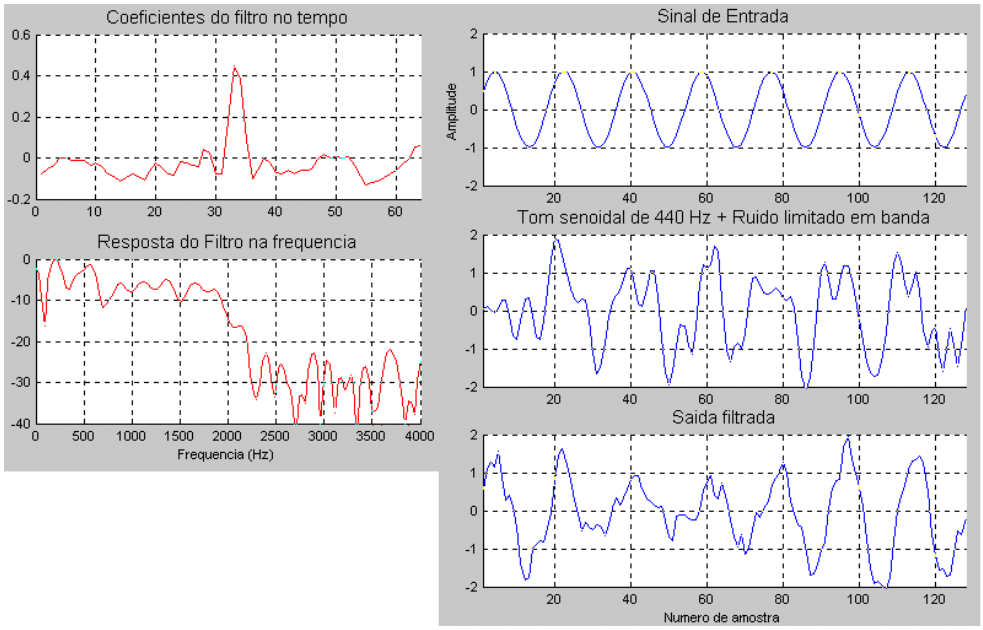

Figura 13 – Resultado do Terceiro quadro de processamento

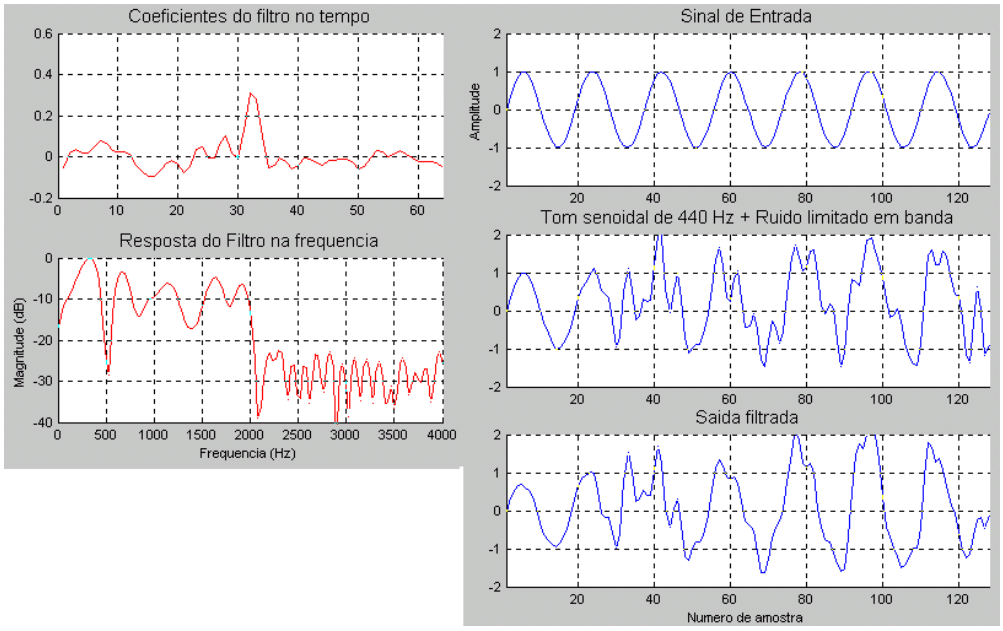

Figura 14 – Resultado do Quarto quadro de processamento

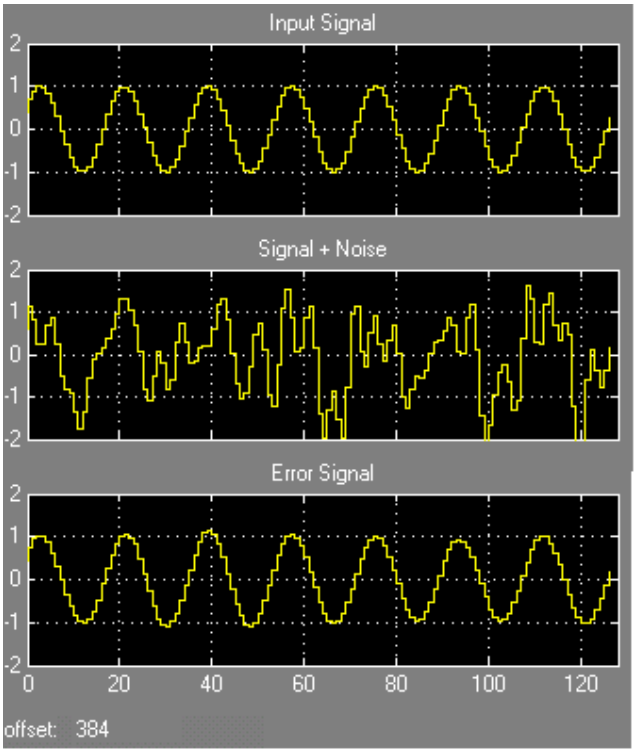

Figura 15 – Resultados de 511 iterações para o algoritmo LMS nas mesmas condições do kit

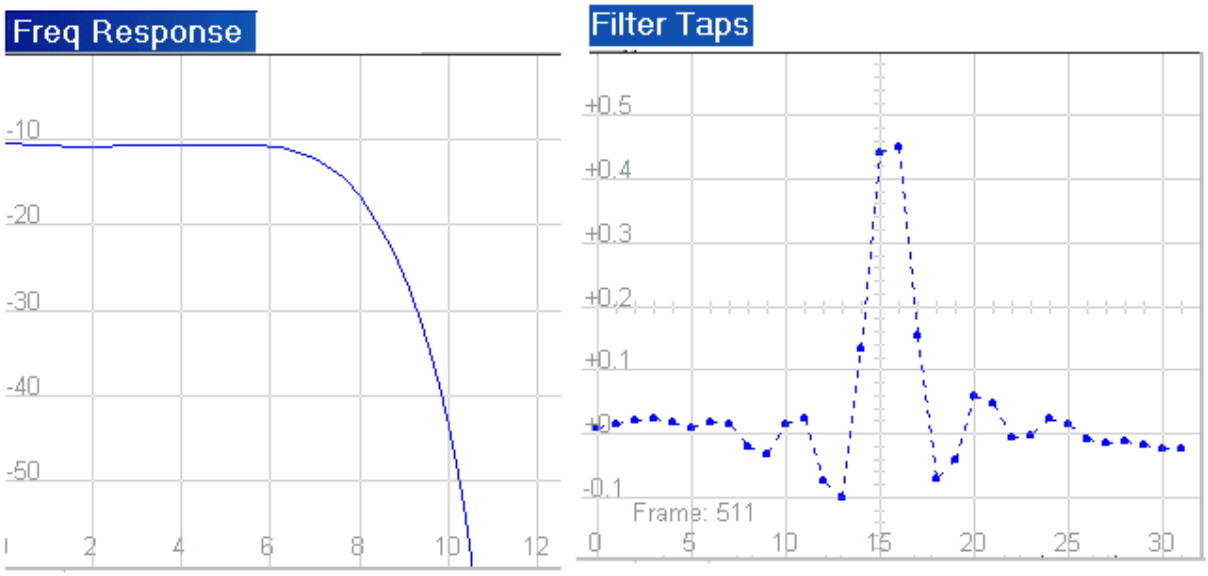

Figura 16 – Respostas na freqüência e no tempo do filtro equalizador LMS da figura 15

9 Anexos

9.1 Códigos Fontes

9.1.1 Script Matlab

```
function dummy = rtdxlms_script(boardNum, procNum, varargin)
%   DUMMY = RTDXLMS_SCRIPT(BOARD,PROC,FILTERORDER,FRAMESIZE,NUMFRAMES)
% Repassa e valida as entradas
[errMsg, filterOrder, frameSize, numFrames] = ...
    parse_args(boardNum, procNum, varargin{:});
error(errMsg);

%O validador acima usa como valores padrao os seguintes valores
%filterOrder = 32;
%frameSize = 64;
%numFrames = 2;

% Cria sinais e vetores de ruido
startN(1) = 1;
startD(1) = 1;
stopN(1) = frameSize;
stopD(1) = frameSize+filterOrder-1;

for i=2:numFrames,
    startN(i) = stopN(i-1)+1;
    stopN(i) = startN(i)+frameSize-1;
    startD(i) = stopD(i-1)+1;
    stopD(i) = startD(i)+frameSize-1;
end

outBuf = frameSize;  % Tamanho do coeficiente de buffer
noise = randn(1,frameSize*numFrames);
maxVal = max([max(noise) abs(min(noise))]);
shiftBits = 15-nextpow2(maxVal)-2;
scale = 2^(shiftBits);

noise_int16 = double2int16(noise, scale);

% Configura a janela de resposta do filtro
[filtWin, hFig] = SetupFilterPlot(filterOrder);

% Filtra ruido espalhado
cutOffFreq = 0.5;
filteredNoise = demo_fir(filterOrder, cutOffFreq, noise);
filteredNoise_int16 = double2int16(filteredNoise, scale);

% Gera um tom puro de 440 Hz
f=440; fs=8000; t=0:(frameSize*numFrames)+filterOrder-2;
CW = sin(2*pi*f*t/fs);
CW_int16 = double2int16(CW, scale);

% Sinal + ruido filtrado
S_N = CW + filteredNoise;
S_N_int16 = double2int16(S_N, scale);

% Constroi o objeto Code Composer/RTDX
cc=ccsdsp('boardnum',boardNum,'procnum',procNum);
pause(2.0);

% Faz a verificaçao para ter certeza de que a aplicaçao alvo realmente e´ a prevista
% ============================================================================
if cc.isrtdxcapable,
   if ((cc.info.subfamily  >= hex2dec('60')) & (cc.info.subfamily  <= hex2dec('6f')) &
(cc.info.revfamily > 10)),
   else
      uiwait(msgbox({'A aplicaçao nao suporta o simulador ou placa selecionado,  ';
      'Por favor selecione outra placa.'},'Selection Error','modal'));
      clear cc;
      close(hFig);
   end
end
%============================================================================
```

```
% Muda o diretorio para o diretorio alvo e abre o arquivo alvo
cc.cd('C:\Lab_DSP\LMS')

fprintf('Carregando arquivo COFF para o DSP alvo...\n');
try
    cc.load('rtdx_6x1x.out')
catch
    % Fecha figura e mostra erro
    clear cc;
    close(hFig);
    errordlg({'Houve um problema carregando o arquivo COFF para o processador
selecionado.';...
              'Voce pode necessitar recarregar o projeto com o respectivo comando de
arquivo'},...
              'Load Error' , 'modal');
    return
end

% Configura os buffers de canal
cc.rtdx.configure(10000,4);   % Define quatro canais de buffers de 1024 bytes cada

fprintf('Abrindo Canais RTDX...\n');
% Abre canal de escrita
cc.rtdx.open('ichan0','w','ichan1','w');

% Abre canal de leitura
cc.rtdx.open('ochan0','r');

% Habilita todos os canais abertos
cc.rtdx.enable('all');

% Habilita RTDX
cc.rtdx.enable;

% Sobreescreve o valor de timeout ou tempo de espera global
cc.rtdx.set('timeout', 30)            % 30 segundos

% Mostra informaçao dos canais
cc.rtdx

% Escreve para os parametros de filtro para o DSP alvo
fprintf('Escrevendo parametros de filtro para o alvo...\n\n');
filtParms=[filterOrder frameSize numFrames shiftBits];
cc.rtdx.writemsg('ichan0', int16(filtParms));

for ct = 1:numFrames,
    % Escreve para o DSP alvo
    fprintf('Escrevendo ruido + sinal para o alvo - Quadro %d...\n',ct);
    cc.rtdx.writemsg('ichan0', noise_int16(startN(ct):stopN(ct)));
    cc.rtdx.writemsg('ichan1', S_N_int16(startD(ct):stopD(ct)));
end
fprintf('\n');

% Roda Alvo
fprintf('Rodando aplicaçao alvo...\n\n');
cc.run;

outError=zeros(1,frameSize);
s=PlotSignalIO(outError,CW,S_N);

yLines = [s.FiltOut s.SigPlusNoise s.Input];
allNaN = NaN*ones(1,frameSize);
set(yLines, 'EraseMode', 'xor', 'xdata',1:frameSize, 'ydata',allNaN);

for ct = 1:numFrames,
    % Limpa Janela de grafico para valores nulos no inicio de cada quadro:
    set(yLines, 'ydata', allNaN);

    fprintf(['Lendo coeficientes atualizados e resultados filtrados de alvo - Quadro
%d...\n'],ct);

    numPerBlock = outBuf/filterOrder;
    numBlocks   = frameSize/numPerBlock;

    for i=1:numBlocks,
        outCoeff_int16 = cc.rtdx.readmsg('ochan0', 'int16');
```

```
        outCoeff = int16todouble(outCoeff_int16,scale);

        for j=1:numPerBlock,
            startIdx = (j-1)*filterOrder+1;
            endIdx = startIdx + filterOrder-1;
            set(filtWin.hTaps, 'ydata',fliplr(outCoeff(startIdx:endIdx)));

            ff = fft(fliplr(double(outCoeff(startIdx:endIdx))), 256);
            ff = 20*log10(abs(ff(1:128)));
            fmax = max(ff);
            set(filtWin.hFreq, 'ydata', ff-fmax);
        end

            % Atualiza graficos de sinal
            %
            % Pega indices de sinal
            x=(i-1)*numPerBlock+1;
            y=(i-1)*numPerBlock+numPerBlock;
            xx = x + (ct-1)*frameSize;
            yy = y + (ct-1)*frameSize;
            %
            % Atualiza grafico de dados
            yAll=get(yLines,'ydata');
            yAll{1}(x:y) = int16todouble(cc.rtdx.readmsg('ochan0', 'int16'),scale);
            yAll{2}(x:y) = S_N(xx:yy);
            yAll{3}(x:y) = CW(xx:yy);
            set(yLines,{'ydata'},yAll);
            drawnow;
    end
end

% Desabilita todos os canais abertos
cc.rtdx.disable('all');

% Desabilita RTDX
cc.rtdx.disable;

% Fecha canais
cc.rtdx.close('all');

clear cc;   % Chama destruidor de objeto

fprintf('\n*************** Experimento Completado ***************\n\n');


%--------------------------------------------------------------------------------
function [errMsg, filterOrder, frameSize, numFrames] = ...
    parse_args(boardNum, procNum, varargin)
% Repassa e valida as entradas
% 'msg' e limpo se nenhum erro ocorre.

filterOrder  = [];
frameSize    = [];
numFrames    = [];

errMsg = nargchk(2,5,nargin);
if ~isempty(errMsg), return; end

if ~(isnumeric(boardNum)&isreal(boardNum)&(boardNum>=0)),
    errMsg = 'O numero da placa deve ser zero ou um intero positivo';
    return
end

if ~(isnumeric(procNum)&isreal(procNum)&(procNum>=0)),
    errMsg = 'O numero da placa deve ser zero ou um intero positivo';
    return
end

if nargin >= 3,
    if (varargin{1} == 16)|(varargin{1} == 32)|(varargin{1} == 64),
        filterOrder = varargin{1};
    else
        errMsg = 'A ordem do filtro deve ser de 16, 32, 64';
    end
    if nargin >= 4,
        if (varargin{2} == 128)|(varargin{2} == 256)|(varargin{2} == 512),
            frameSize = varargin{2};
```

```
        else
            errMsg = 'O Tamanho do quadro deve ser de 128, 256, or 512';
        end
        if nargin == 5,
            if (varargin{3} == 1)|(varargin{3} == 2)|(varargin{3} == 4),
                numFrames = varargin{3};
            else
                errMsg = 'O numero de frames deve ser de 1, 2, or 4';
            end
        else
            numFrames = 2;
        end
    else
        numFrames = 2;
        frameSize = 256;
    end
else
    numFrames = 2;
    frameSize = 256;
    filterOrder = 32;
end

%--------------------------------------------------------------------------
function out = double2int16(data,scaleFactor);

out = int16(round(data*scaleFactor));


%--------------------------------------------------------------------------
function out = int16todouble(data,scaleFactor);

out = double(data)/scaleFactor;


%--------------------------------------------------------------------------
function out = demo_fir(order,cutoff,data);

coeff = fir1(order-1,cutoff);
out = conv(coeff,data);


%--------------------------------------------------------------------------
function [s, hFig] = SetupFilterPlot(filterOrder)
% Configura a resposta do filtro
% Retorna uma estrutura como:
%   .hTaps
%   .hFreq

% Checa por uma janela de figura existente
filtTag = 'rtdxlms_filters';
signalTag = 'rtdxlms_signal';

s=[];

hFig = findobj('tag',filtTag);
if ~isempty(hFig),
    close(hFig);
end

hFig1 = findobj('tag',signalTag);
if ~isempty(hFig1),
    close(hFig1);
end

hFig = figure('numbertitle','off', 'name','rtdxlms - Resultados para o Filtro
equalizador', ...
    'IntegerHandle','off', 'units','normalized', 'tag', filtTag, 'pos',[0.01 0.47 0.35
0.45]);

% Configura eixo de coeficientes no tempo
subplot(2,1,1); grid on;
title('Coeficientes do filtro no tempo');
s.hTaps=line;
set(s.hTaps, 'erase','xor', 'color','r');
a = gca;
set(a,'ylim',[-0.2 0.6]);
set(a,'xlim',[0 filterOrder]);
```

```
set(get(a,'title'),'fontsize',11);
set(a,'fontsize',8);
set(a,'fontweight','light') ;

% Configura eixo de frequencia
subplot(2,1,2); grid on;
title('Resposta do Filtro na frequencia');
xlabel('Frequencia (Hz)');
ylabel('Magnitude (dB)');
s.hFreq=line;
set(s.hFreq, 'erase','xor', 'color','r');
b = gca;
set(b,'ylim',[-40 0]);
set(b,'xlim',[0 4000]);

set(get(b,'title'),'fontsize',11);
set(b,'fontsize',8);
set(b,'fontweight','light') ;
x = get(b,'xlabel');
set(x,'fontsize',8);
set(x,'fontweight','light');
y = get(b,'ylabel');
set(y,'fontsize',8);
set(y,'fontweight','light');

set(hFig,'userdata',s);  % Retem estrutura no campo de dados do usuario da figura

% Inicializa linhas
set(s.hTaps, 'xdata',1:filterOrder,    'ydata',zeros(1,filterOrder));
set(s.hFreq, 'xdata',(0:127)/127*4000, 'ydata',zeros(1,128));

figure(hFig);
drawnow;


%--------------------------------------------------------------------------
function s = PlotSignalIO(outError, CW, S_N)
% Plota Sinal IO

% Checa para janela de figura exsitente
signalTag = 'rtdxlms_signal';
s = [];


hFig = figure( 'numbertitle','off', 'name','rtdxlms - Signals', 'IntegerHandle','off',
...
    'tag', signalTag, 'units','normalized', 'menubar','Figure', 'pos', [0.37 0.28 0.44
0.64]);

hax = axes('parent',hFig, 'pos', [0.13 0.701222343126587 0.775 0.223777656873413], ...
    'xlim', [1 length(outError)], 'ylim', [-2 2], 'drawmode','fast', 'xgrid', 'on',
'ygrid', 'on');

set(get(hax,'title'),'fontsize',11);
set(hax,'fontsize',8);
set(hax,'fontweight','light') ;
x = get(hax,'xlabel');
set(x,'fontsize',8);
set(x,'fontweight','light');
y = get(hax,'ylabel');
set(y,'fontsize',8);
set(y,'fontweight','light');

set(get(hax,'title'),'string','Sinal de Entrada');
set(get(hax,'ylabel'),'string','Amplitude');
s.Input = line('parent',hax, 'color','b');

hax = axes('parent', hFig, 'pos', [0.13 0.405611171563293 0.775 0.223777656873413], ...
    'xlim', [1 length(outError)], 'ylim', [-2 2], 'drawmode','fast', 'xgrid', 'on',
'ygrid', 'on');

set(get(hax,'title'),'string','Tom senoidal de 440 Hz + Ruido limitado em banda');
s.SigPlusNoise = line('parent',hax, 'color','b');

set(get(hax,'title'),'fontsize',11);
set(hax,'fontsize',8);
```

```
set(hax,'fontweight','light') ;
x = get(hax,'xlabel');
set(x,'fontsize',8);
set(x,'fontweight','light');
y = get(hax,'ylabel');
set(y,'fontsize',8);
set(y,'fontweight','light');

hax = axes('parent', hFig, 'pos', [0.13 0.11 0.775 0.223777656873413], 'xlim', [1
length(outError)], ...
    'ylim', [-2 2], 'drawmode','fast', 'xgrid', 'on', 'ygrid', 'on');
set(get(hax,'title'),'string','Saida filtrada');
set(get(hax,'xlabel'),'string','Numero de amostra');
s.FiltOut = line('parent',hax, 'color','b');

set(get(hax,'title'),'fontsize',11);
set(hax,'fontsize',8);
set(hax,'fontweight','light') ;
x = get(hax,'xlabel');
set(x,'fontsize',8);
set(x,'fontweight','light');
y = get(hax,'ylabel');
set(y,'fontsize',8);
set(y,'fontweight','light');

set(hFig,'userdata',s);  % Retem estrutura no campo de dados do usuario da figura

figure(hFig);
drawnow;

% [EOF] rtdxlms_script.m
```

9.1.2 Projeto Code Composer

9.1.2.1 Estrutura organizacional do projeto no Code Composer Studio

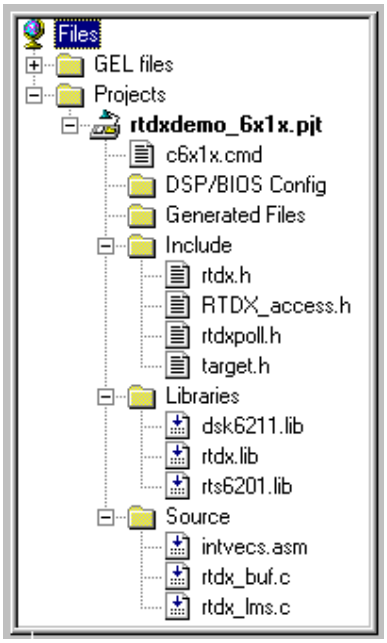

Figura 17 – Estrutura organizacional do projeto

9.1.2.2 Arquivo RTDX_LMS.c

```
/*--------------------------------------------------------------------*/
/* Algoritmo para adaptativamente cancelar ruido espalhado em banda. */
/* O Erro em cada interação é um ponto de saída em tempo continuo    */
/*   do sinal filtrado (sinal + ruido cancelado                       */
/*                                                                    */
/* Canais de leitura do RTDX:                                         */
```

```
/* Nome do Canal :     Dados:                                          */
/*   ichan0            filtParms                                       */
/*                     filtParms[0] = filter length                    */
/*                     filtParms[1] = frame size                       */
/*                     filtParms[2] = number of frames                 */
/*                     filtParms[3] = number of shift bits (scaling)   */
/*                                                                     */
/*   ichan0         x (entrada do filtro)                              */
/*   ichan1         y (saída do filtro desejada)                       */
/*                                                                     */
/* Canais de escrita para o RTDX:                                      */
/* Nome do Canal:    Dados:                                            */
/*   ochan0          hPrime (Atualização dos coeficientes)             */
/*                   error  (Desejado - filtro de saída estimado       */
/*                                                                     */
/*---------------------------------------------------------------------*/

#include <rtdx.h>   /* RTDX_Read      */
#include "target.h"   /* TARGET_INITIALIZE  */
#include <stdio.h>    /* printf                        */

#define MAXFRAME 1024
#define MAXTAPS  64

short filtParms[4];
short m, k, numFrames;
short shiftBits;
short x[MAXFRAME], y[MAXFRAME+MAXTAPS-1], xTaps[MAXTAPS+1];
short hPrime[MAXTAPS], yPrime[MAXFRAME];
short mu = 1;             /* mu = 0.5, LSB = 2^-1                        */
short normFactor;         /* Tamanho de passo e fator de autocorrelação  */
short normError;          /* Erro Interativo e erro normalizado          */
short outBuf[MAXFRAME];   /* buffer de saida para coeficientes           */

long  error[MAXFRAME];

RTDX_CreateInputChannel(ichan0);  /* Canal onde se receberá a entrada do filtro  */
RTDX_CreateInputChannel(ichan1);  /* Canal on se receberá a saida do filtro      */
RTDX_CreateOutputChannel(ochan0); /* Canal de saida de coeficientes atualizados e sinal
filtrado */

void main(void)
 {
  int h, i,j,n;
  int wptr = 0;              /* Ponteiro para a localização corrente no buffer de
escrita */
  short outError[MAXFRAME];  /* Buffer de saída para sinal filtrado */

  TARGET_INITIALIZE();       /* Inicialização do DSK*/

  RTDX_enableInput(&ichan0); /* Habilita Canais */
  RTDX_enableInput(&ichan1);
  RTDX_enableOutput(&ochan0);

  /* Le os parametros do filtro */
  RTDX_read( &ichan0, filtParms, sizeof(filtParms) );

           m = filtParms[0]; /* Comprimento do filtro             */
           k = filtParms[1]; /* filter input framesize  - Tamanho do quadro do filtro de
entrada    */
   numFrames = filtParms[2]; /* number of frames to process - Numero de quadros para
processar*/
  shiftBits  = filtParms[3]; /* LSB: 2^(-shiftBits)         */

  RTDX_read( &ichan0, x, k*2 );
  RTDX_read( &ichan1, y, (k+m-1)*2 );

  /*Inicializa as estimativas de coeficientes*/
  for (i=0; i<m; i++)
   {
   hPrime[i] = 0;
    xTaps[i] = 0;
   }

  for (h=0; h<numFrames; h++)
   {
   for (n=0; n<k; n++)
```

```
    {
    /*Atualiza o coeficiente da linha de retardo e o erro */
      error[n] = y[n] << shiftBits;
      xTaps[m] = x[n];
    normFactor = 0;

    for (i=0; i<m; i++)
     {
       xTaps[i]  = xTaps[i+1];
       error[n] -= (hPrime[i] * xTaps[i]);
     normFactor += (xTaps[i] * xTaps[i]) >> shiftBits + 4;
     }

    outError[n] = error[n] >> shiftBits;
      yPrime[n] = 0;
      normError = mu * error[n]/normFactor >> 5;

    for (j=0; j<m; j++)
     {
     /*Atualiza os coeficientes*/
          hPrime[j] += (normError * xTaps[j]) >> shiftBits;
     outBuf[wptr++]  = hPrime[j];

     /* Calcula o filtro de saida*/
     yPrime[n] += (hPrime[j] * xTaps[j]) >> shiftBits;
     }

    /* Buffer de saída cheio ? */
    if (wptr == k)
     {
     wptr = 0;

     /* Espera escrita anterior se completar */
     while ( RTDX_writing != NULL )
      {
      }

     RTDX_write( &ochan0, outBuf, k*2 );

     /* Espera escrita anterior se completar */
     while ( RTDX_writing != NULL )
      {
      }

     RTDX_write( &ochan0, &outError[n-(k/m)+1], (k/m)*2 );
     }
    }

   /* Sobrescreve com frame anterior */
   for (i=0; i<m; i++)
    {
    y[i] = y[k + i];
    }

   if (h<numFrames-1)
    {
    RTDX_read( &ichan0, x, k*2 );
    RTDX_read( &ichan1, &y[m], k*2 );
    }
   }

  /* Espera escrita anterior se completar */
  while ( RTDX_writing != NULL )
   {
   }

   RTDX_disableInput(&ichan0);
   RTDX_disableInput(&ichan1);
   RTDX_disableOutput(&ochan0);
  }
```

9.1.2.3 RTDX_BUF.c

```
/************************************************************************
*
* Declara os Buffers usados na camada ou seção dos buffer's
```

```
*
*************************************************************************/
#ifndef BUFRSZ
#define BUFRSZ 600
#endif

/*
 * Os buffers usados pelo RTDX são definidos por Dois Simbolos: RTDX_Buffer_Start
 * e RTDX_Buffer_End.  É usada as seguintes declarações em ordem para exportar estes
nomes
*/
#if _LARGE_MODEL
#pragma DATA_SECTION(RTDX_Buffer_Start,".rtdx_data")
#pragma DATA_SECTION(RTDX_Buffer_End,".rtdx_data")
int RTDX_Buffer_Start[BUFRSZ];
int RTDX_Buffer_End;
#else
int far RTDX_Buffer_Start[BUFRSZ];
int far RTDX_Buffer_End;
#endif
```

9.1.2.4 Target.h

```
#ifndef __TARGET_H
#define __TARGET_H

/*************************************************************************
* - Target specific initialization details
*
*************************************************************************/

extern cregister volatile unsigned int IER;
extern cregister volatile unsigned int ISR;
extern cregister volatile unsigned int CSR;

#define NMIE_BIT  0x00000002
#define TARGET_INITIALIZE() \
  IER |= 0x00000001 | NMIE_BIT; \
  CSR |= 0x00000001;

#endif // __TARGET_H
```

9.1.2.5 C6x1x.cmd

```
/*************************************************************************
*   $Revision: $  $Date: 2000/12/14 19:52:43 $
*
* Copyright (c) 1997 Texas Instruments Incorporated
*
*************************************************************************/

_HWI_Cache_Control = 0;

/* RTDX Interrupt Mask
   - This symbol defines those interrupts which are clients of RTDX
     (ie - interrupts which call RTDX functions.
   - RTDX will apply this mask to the IER before excuting code in
     RTDX critical sections temporarily disabling those interrupts for
     a few cycles.
   - Any interrupt handlers which call RTDX_read/write functions should
     be added to the mask to prevent corruption of the RTDX global state
     variables by simulataneous access from multiple RTDX clients.
*/
_RTDX_interrupt_mask = ~0x000000008;

MEMORY
 {
 PMEM:  o=00000000h l=00010000h   /* Internal RAM (L2) mem  */
 BMEM:  o=80000000h l=01000000h   /* CE0, SDRAM, 16 MBytes  */
 }

SECTIONS
 {
 .intvecs > 0h
 .text    > PMEM
 .far   > PMEM
```

```
 .stack   > PMEM
 .bss   > PMEM
 .cinit   > PMEM
 .pinit   > PMEM
 .cio   > PMEM
 .const   > PMEM
 .data    > PMEM
 .switch    > PMEM
 .sysmem    > PMEM
 }
```

9.1.2.6 IntVecs.asm

```
;***********************************************************************
;*            Inicialização do vetor de interrupção para o 'C6x                *
;***********************************************************************
       .title "Interrupt Vectors w/ RTDX for C6X"
       .tab   4

SP     .set   B15                                ; Redefine for convenience

;***********************************************************************
;*        Interrupt Service Table (IST) ou Tabela de serviço de interrupção
;***********************************************************************
       .sect  ".intvecs"
       .align 32*8*4                       ; must be aligned on 256 word boundary
RESET_V                        ; Reset ISFP
       .ref   _c_int00                     ; program reset address
       MVKL   _c_int00, B3
       MVKH   _c_int00, B3
       B              B3
       MVC    PCE1, B0                     ; address of interrupt vectors
       MVC    B0, ISTP                     ; set table to point here
       NOP    3

       .align 32
NMI_V                          ; Non-maskable interrupt Vector
       B              NRP
       NOP    5

       .align 32
AINT_V                         ; Analysis Interrupt Vector (reserved)
       B              $
       NOP    5

       .align 32
MSGINT_V                       ; Message Interrupt Vector (reserved)
       .ref   RTEMU_msg
       STW    B0, *SP--[2]
||     MVKL   RTEMU_msg, B0
       MVKH   RTEMU_msg, B0
       B              B0
       LDW    *++SP[2], B0
       NOP    4

       .align 32
INT4_V                         ; Maskable Interrupt #4
       B              $
       NOP    5

       .align 32
INT5_V                         ; Maskable Interrupt #5
       B              $
       NOP 5

       .align 32
INT6_V                         ; Maskable Interrupt #6
       B              $
       NOP    5

       .align 32
INT7_V                         ; Maskable Interrupt #7
       B              $
       NOP    5

       .align 32
```

```
INT8_V                          ; Maskable Interrupt #8
       B               $
       NOP     5

       .align  32
INT9_V                          ; Maskable Interrupt #9
       B               $
       NOP     5

       .align  32
INT10_V                         ; Maskable Interrupt #10
       B               $
       NOP     5

       .align  32
INT11_V                         ; Maskable Interrupt #11
       B               $
       NOP     5

       .align  32
INT12_V                         ; Maskable Interrupt #12
       B               $
       NOP     5

       .align  32
INT13_V                         ; Maskable Interrupt #13
       B               $
       NOP     5

       .align  32
INT14_V                         ; Maskable Interrupt #14
       B               $
       NOP     5

       .align  32
INT15_V                         ; Maskable Interrupt #15
       B               $
       NOP     5

; the remainder of the vector table is reserved

       .align  32*8*4                          ; reserve full 256 words

       .end
```