CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA

CELSO SUCKOW DA FONSECA – CEFET/RJ

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

COORDENADORIA DO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA

DISSERTAÇÃO

SISTEMA ELETRÔNICO DE PROCESSAMENTO DIGITAL PARA A GERAÇÃO DE IMAGENS POR ULTRA-SOM, OPERANDO NO MODO B

Sérgio Luiz Fernandes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM TECNOLOGIA.

CARLOS HENRIQUE F. ALVES, DSc.

Orientador

RIO DE JANEIRO, RJ – BRASIL.

NOVEMBRO / 2007

Sumário

Ficha catalográfica elaborada pela Biblioteca Central do CEFET-RJ

F363 Fernandes , Sérgio Luiz
Sistema eletrônico de processamento digital para a geração de imagens por ultra som , operando no modo B / Sérgio Luiz Fernandes .—2007.
viii , 49f. + Apêndices: il. ; enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca , 2007.
Bibliografia : f.48-49

1.Sistemas eletrônicos 2.Ultra-som 3.Televisão digital I.Título

CDD 621.381

Resumo da dissertação submetida ao PPTEC/CEFET-RJ como parte dos requisitos necessários para a obtenção do grau de mestre em tecnologia (M.T.).

SISTEMA ELETRÔNICO DE PROCESSAMENTO DIGITAL PARA A GERAÇÃO DE IMAGENS POR ULTRA-SOM, OPERANDO NO MODO B

Sérgio Luiz Fernandes

Novembro de 2007

Orientador: Carlos Henrique Figueiredo Alves, D.Sc.

Programa: PPTEC

O foco da dissertação, é apresentar as soluções encontradas para a geração de imagens por ultra-som, operando no modos “B” , utilizando um sistema de interface portátil, produzindo imagens com resoluções satisfatórias e desvinculando o equipamento de uma CPU e de um monitor de vídeo exclusivamente a ele dedicados.

Para a elaboração do sistema, foi realizada uma pesquisa cuidadosa sobre: o processo de aquisição de sinal dos transdutores, as técnicas de varredura dos receptores de TV e dos monitores de vídeo, a formação do sinal composto de vídeo, os circuitos de conversão de sinais analógicos para digitais, as formas e padrões do processamento do sinal digital, para a geração da imagem.

Foram desenvolvidos os processos metodológicos para a criação de um sistema eletrônico, que opera entre a sonda transdutora e o *display*. Sistema esse, capaz de gerar um sinal composto de vídeo com as informações dos ecos e efetuar um processamento digital, a fim de atribuir ao mesmo as características necessárias para a formação da imagem do corpo explorado.

O protótipo produzido, revelou-se versátil e eficiente, pois permite utilizar, como monitor, micro-computadores portáteis (*laptops*) e receptores de TV (analógicos ou digitais), tendo alcançado a definição pretendida na imagem. Logo, pode vir a se tornar uma alternativa atraente para aumentar a versatilidade e a mobilidade dos sistemas atuais.

Palavras chave**: Ultra-som, Imagem, TV digital.**

Abstract of dissertation submitted to PPTEC/CEFET/RJ as partial fulfillment of the requirements for the degree of Master in Technology (M.T.).

ELETRONIC SYSTEM OF DIGITAL PROCESSING FOR GENERATION OF IMAGES IN THE B-MODE

Sérgio Luiz Fernandes

November / 2007

Supervisor: Carlos Henrique Figueiredo Alves (D.Sc.)

Program: PPTEC

The focus of the dissertation, is to present the solutions to the generation of images by ultra-sound, operating in mode "B", using a portable system interface, producing images with satisfactory resolutions and unlinking the equipment of a CPU and of a video monitor exclusively devoted to it.

For the development of the system, a search was carefully conducted about: the process of acquisition of the transducers signal, the scanning techniques of TV recipients and video monitors, the formation of the composite video signal, the converters circuits of analogical signals to digital, the forms and patterns of processing the digital signal to the generation of the image.

It was developed the methodological procedures for the creation of an electronic system, which operates between the transducer probe and the display. This system is capable of generating a composite video signal with information from echoes and make a digital processing, in order to ascribe the same characteristics necessary for the formation of the image of the exploited material.

The prototype produced, has proved versatile and efficient, as it allows use as a monitor, micro-portable computers (laptops) and TV receivers (analogical or digital), and achieved the desired definition in the image. Soon, it may become an attractive alternative to increase the versatility and the mobility of the current systems.

Keywords: **ultra-sound, image, digital TV.**

Lista de ilustrações

Introdução

A crescente demanda na utilização de equipamentos fundamentados na emissão e recepção de ondas de ultra-som é notória nos últimos anos, tanto na área de saúde, como em diversos seguimentos da indústria .

Na indústria, a determinação da espessura, integridade, composição e uniformidade de diversos tipos de materiais são feitas atualmente por meio de testes utilizando o ultra-som. Esses métodos são conhecidos como “Ensaios Não Destrutivos” (END).

Apesar de constituir uma poderosa ferramenta industrial, os equipamentos de ultra-som são considerados indispensáveis no campo da medicina, visto que, apresentam diversas vantagens, quando comparados a outros dispositivos de tratamento e diagnóstico.

- O paciente sofre um mínimo desconforto. Apenas uma leve pressão da sonda na área de aplicação.
- O paciente não necessita ingerir nenhuma substância previamente (contraste).
- O ultra-som é predominantemente não-invasivo, dispensando o uso de agulhas e instrumentos de corte (salvo em procedimentos específicos como a biópsia).
- Por não ser uma energia ionizante o ultra-som pode ser aplicado repetidas vezes sem seqüelas para o paciente.
- O resultado é imediato, não necessitando de técnicas de revelação fotográficas.

A aplicação de ondas de ultra-som em áreas traumatizadas são freqüentes como uso terapêutico e comprovadamente eficazes. Contudo, como instrumento de auxílio em diagnósticos, o equipamento de ultra-som é considerado fundamental, pois permite a visualização de camadas internas do corpo humano, possibilitando o acompanhamento da evolução de doenças orgânicas em diversos segmentos e do processo de gestação de fetos em obstetrícia, onde ganhou maior popularidade.

Com a evolução do sistema, que se traduziu em sondas com alta resolução e equipamento computacional para o processamento da imagem, tornou-se possível a criação de microscópios acústicos, empregados em dermatologia e oftalmologia.

O alto custo dos equipamentos de ultra-som, representam ainda a maior dificuldade de operacionalização nas instituições públicas e nas instituições privadas de pequeno porte, pois a aquisição do equipamento fica comprometida e o treinamento de profissionais por parte das instituições de ensino também se inviabiliza.

A proposta desta dissertação é a criação de um sistema eletrônico para o processamento do sinal proveniente da sonda transdutora (desenvolvida na dissertação de mestrado de Paulo Ernesto Moreira, CEFET-RJ, novembro / 2007), a fim de gerar uma imagem satisfatória em um receptor de TV analógico ou digital, ou em um *laptop.* Tal processamento independe de computadores a ele associados, ou seja são módulos autônomos, de baixo custo e portáteis. As partes integrantes do sistema são:

- Circuito codificador de sinal composto de vídeo.
- Circuito de conversão A/D.
- Circuito de processamento de sinal digital (processador e memórias).
- Circuito de conversão D/A.

Obs.: Os três últimos circuitos citados , foram sintetizados no módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices.***

Os principais fatores que justificam o trabalho, podem ser destacados como:

- Baixo custo, uma vez que não emprega máquinas dedicadas, com monopólios de fabricação.
- Mobilidade, pois os módulos são pequenos e autônomos.
- Uso industrial, o sistema pode ser empregado na formação de imagens de ensaios não destrutivos.
- Didático, visto que pode ser explorado em cursos de treinamento de ultra-som.

A dissertação é composta de quatro partes e um apêndice. Na primeira parte é feita uma revisão bibliográfica onde, se destacam os modos de operação dos transdutores de ultra-som, os padrões de vídeo digital e o processamento do sinal para o *display.*

Na segunda parte é descrita a metodologia empregada no desenvolvimento dos circuitos eletrônicos para a codificação do sinal composto de vídeo, a conversão A/D, o processamento de sinal digital e a conversão D/A, abordando as soluções encontradas objetivando a fidelidade da imagem.

A terceira parte demonstra alguns testes e sinais encontrados.

A quarta parte é uma análise e discussão dos resultados obtidos e uma conclusão geral do sistema.

Os apêndices descrevem : O programa comentado, em linguagem C++, desenvolvido para o processamento digital, as características técnicas do processador empregado e as imagens de testes com técnicas de pré-processamento.

I – Revisão Bibliográfica

I.1 Sinal adquirido pelo transdutor (Modos de Operação)

Os transdutores destinados a aplicações médicas são em geral feitos com materiais piezelétricos. Essas aplicações podem ser divididas como terapêuticas e diagnósticas. No primeiro caso as ondas emitidas são de baixa freqüência e as ondas refletidas não têm aplicabilidade, pois o objetivo principal é a geração de calor em um determinado ponto do corpo. No segundo caso, que é o de interesse nesse trabalho, as ondas refletidas são usadas como base para a geração de imagens. A figura I.1 ilustra um transdutor e as ondas incidente, transmitida e refletida.[1].

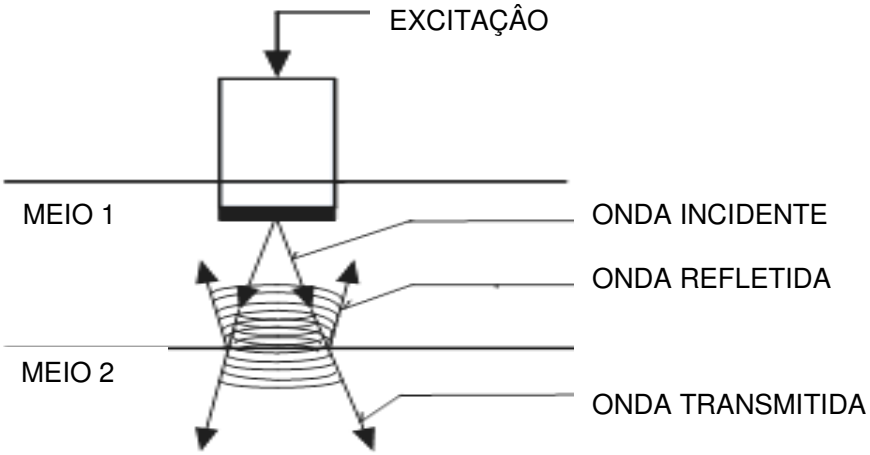


Fig. I.1 – Excitação do transdutor

As ondas refletidas (ecos), podem ser obtidas pela excitação do transdutor em duas formas: A primeira, é através de rajadas (*burst)* de um sinal com a freqüência igual a de ressonância do transdutor. A segunda, é através de um pulso com grande amplitude e curta duração . É possível concluir, empregando-se a “Análise de Fourier”, que quanto menor a largura do pulso maior será a largura da faixa de freqüência de excitação do transdutor, de modo a aumentar a probabilidade de existir uma componente na qual o transdutor entre em ressonância.

O transdutor, juntamente com o cabo e o conector, é chamado de sonda ou *probe*.

No tocante a aquisição e processamento dos dados para a formação de imagens, os transdutores de ultra-som operam nos seguintes modos básicos:

Modo “A”, é a base dos demais modos. Os ecos recebidos pelo transdutor formam uma única coluna, de imagem que corresponde a visão axial (de profundidade) do ponto onde está posicionado em relação ao corpo de prova. As diferentes camadas do corpo, produzem sinais de ecos com fases e amplitudes diferentes. Monitorando esses sinais com um osciloscópio, é possível calcular a distância entre as camadas, com base no intervalo de tempo entre os mesmos. Inicialmente o modo “A” constituiu a principal técnica de medida de espessura em oftalmologia e dermatologia [2]. Uma conversão direta entre o tempo decorrido entre dois ecos recebidos e o espaço percorrido por eles permitiu obter medidas precisas em tecidos vivos, conhecidas, evidentemente, a velocidade de propagação (**v**) nesses tecidos e a base de tempo (**t**) da varredura horizontal do osciloscópio (Figura I.2).

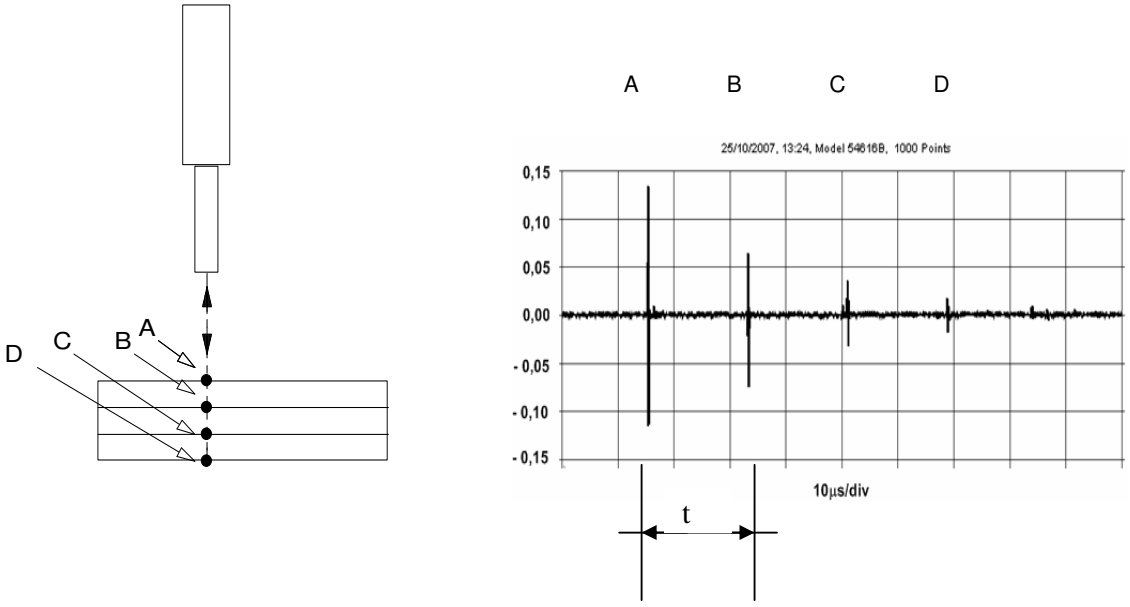


Fig. I.2 – Determinação da espessura das camadas.

As espessuras são dadas pela equação 1.1, onde “d” é a distância entre as camadas, “v” é a velocidade de propagação do ultra som no tecido e “t” é o intervalo de tempo entre os ecos.

$$\mathbf{d = (v . t). 1/2} \qquad (1.1)$$

Em geral, as informações provenientes da sonda transdutora de ultra-som operando em modo “A”, são analisadas por instrumentos de medida, como o osciloscópio por exemplo.

Entretanto, é possível imaginar que, com as envoltórias dos ecos da figura I.2, devidamente submetidas a um processamento (que é o foco desse trabalho) para adequá-los as características do monitor de vídeo, produziria a imagem da figura I.3. É importante ressaltar que a variação de amplitude dos sinais resultaria na mudança de luminância dos pontos.

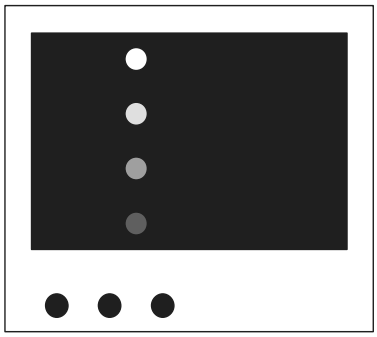


Fig. I.3 - Imagem do sinal da figura I.2.

O diagrama de blocos da Figura I.4 sugere o sistema para a aquisição e processamento dos sinais dos ecos, operando em modo "A". [2]

O transdutor é excitado pelos pulsos do gerador. A onda ultra-sônica é irradiada e os ecos das superfícies do corpo explorado são recebidos pelo próprio transdutor e encaminhados pela chave eletrônica para o amplificador. Para uma suposta geração de imagem, após amplificados, os ecos seriam demodulados em envoltória e aplicados na etapa de processamento de sinal para o display. Tal processamento será discutido em detalhes nos próximos itens.

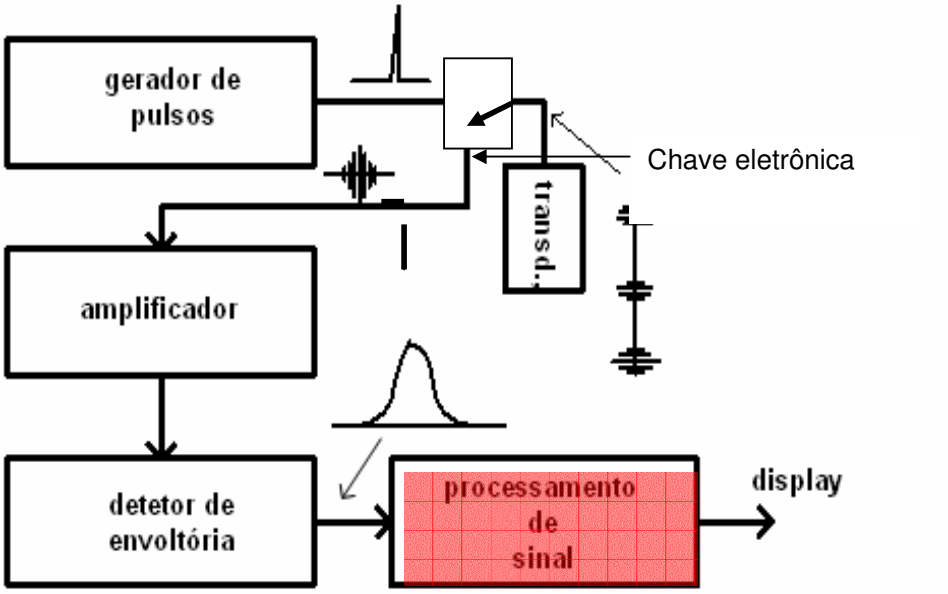


Fig. I.4 - Operação em modo A.

O modo “B”, se analisado pontualmente é equivalente ao modo “A”, a questão é que possui um circuito capaz de processar a imagem em duas dimensões, onde além da exploração axial, existe a exploração lateral, pois um mecanismo promove um deslocamento do transdutor, acontecendo uma varredura (*scanning*) sobre o corpo em análise. Cada posição do transdutor corresponde a uma coluna da imagem. O resultado é um conjunto de colunas em modo “A”, compondo a imagem da operação em modo “B”, como ilustra a figura I.5. [2]

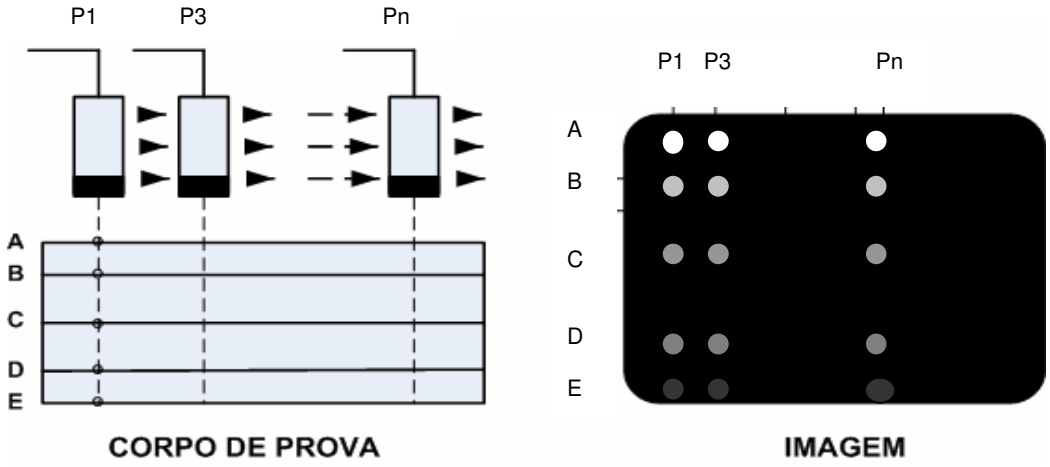


Fig. I.5 – Princípio da formação de imagem no modo B

O sistema de operação em modo “B”, como denota a figura I.6, contempla um circuito gerador de varredura e sincronismo, que comanda o deslocamento do transdutor mantendo-o sincronizado com os pulsos de disparo. Os ecos são encaminhados pela chave eletrônica, a um estágio de amplificação não linear composto por um amplificador com ganho variável no tempo (TGC – *Time Gain Controled*) e um amplificador logarítmico. O TGC usa uma tensão de polarização em forma de rampa, coincidente no tempo compreendido entre os pulsos de disparo. Isto faz com que os ecos obtidos de profundidades maiores, geralmente muito atenuados, sejam mais amplificados que os obtidos de profundidades menores, ou seja, mais próximos em tempo do pulso de disparo. O resultado dessa amplificação e compressão é um aumento na relação sinal-ruído e também a redução do comprimento da palavra binária da codificação digital da amplitude do sinal.

Depois de amplificados, um detector de envoltória elimina as variações de alta freqüência do sinal, transformando em um sinal próprio para ser convertido em um ponto

luminoso em uma tela. O sinal detectado normalmente sofre sofisticado processamento digital antes de ser exibido. Isto torna o sistema apto a colher e armazenar informações assim como eliminar ou realçar uma série de características do sinal. Para que a imagem se mantenha estática na tela e os pontos mostrados nas posições corretas, a etapa de processamento de sinal para o *display* também deve estar sincronizada com o gerador de varredura.

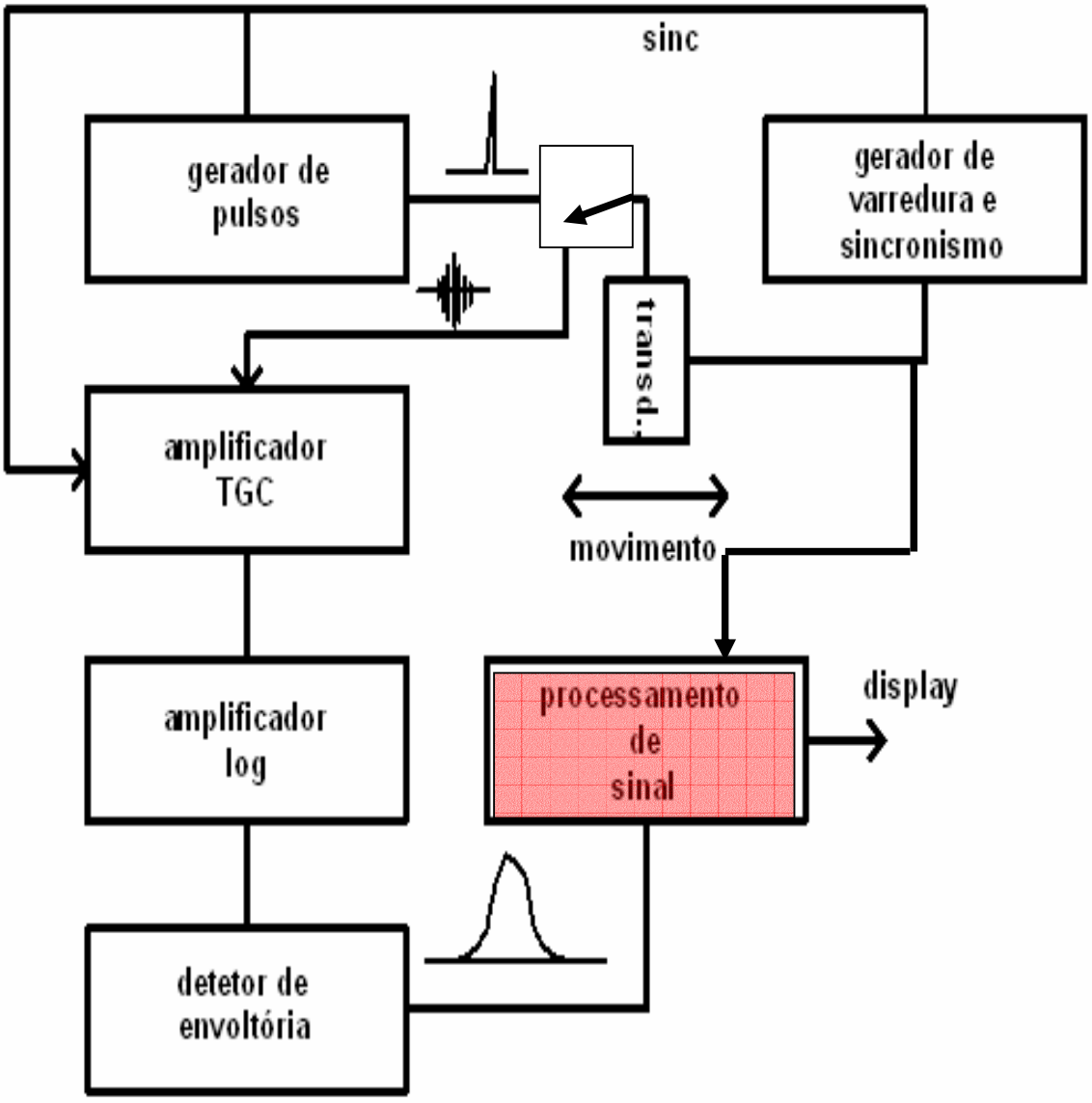


Fig. I.6– Operação em modo B.

O modo “C” é capaz de processar e reproduzir a imagem da varredura em 2 eixos, ou seja, percorrendo um plano. Assim sendo, é possível examinar toda a face do corpo de prova , obtendo visões de camadas externas ou internas. [2] (Figura I.7).

Com as informações obtidas nos diversos planos de exploração e com o auxílio de processamentos com *softwares* apropriados, se consegue até mesmo, a composição da imagem em 3 dimensões.

O Modo “C”, como se pode concluir, é também derivado do Modo “A”, porém com dois eixos de varredura e um sistema de processamento digital mais sofisticado [2].

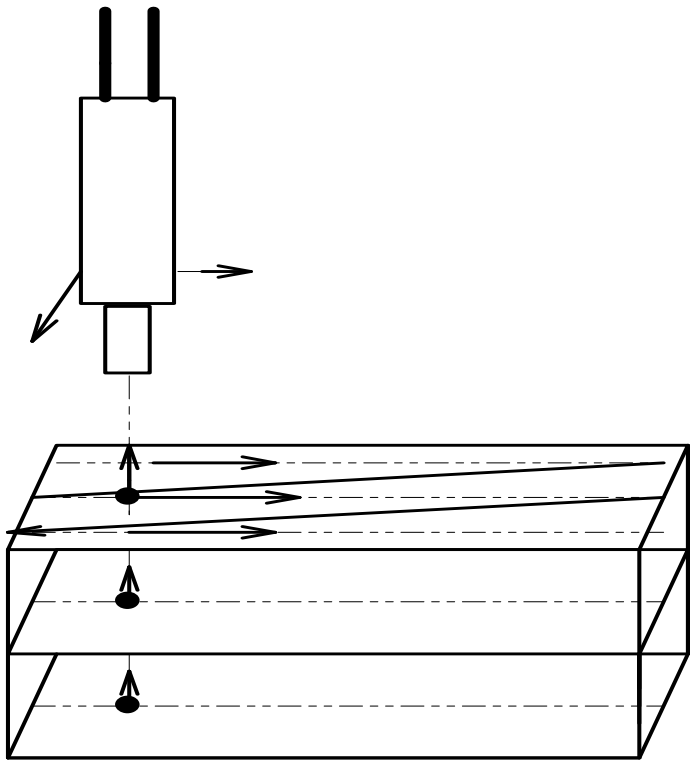


Fig. I.7 – Varredura em Modo “C”

I.2 Processamento de sinal para o *"display"*

Essa etapa tem o compromisso, de efetuar um tratamento com as envoltórias dos ecos, de modo a converte-las em um sinal compatível com o monitor, gerando uma imagem e ressaltando as características que se deseja evidenciar na tela, assim como reduzindo os efeitos das perturbações causadas pelos ruídos.

Como o referido tratamento não tem como ser realizado em tempo real, visto que as informações provenientes das envoltórias devam ser armazenadas com o propósito de serem ajustadas a temporização da varredura do monitor. Os sinais das envoltórias devem então, ser convertidos para a forma digital, onde a memorização e a organização dos dados se tornam viáveis.[3] [4]

O diagrama em blocos da figura I.8, ilustra os estágios do processamento do sinal para a geração da imagem. O 1º estágio é o **conversor A/D**, que converte as envoltórias dos ecos em amostras constituídas por palavras compostas por um certo número de bits.Os bits serão armazenados em uma **memória RAM** .

A seqüência de gravação dos dados na memória é determinada pela ordem de leitura do transdutor , onde apenas se pode sincronizar os pulsos de excitação e de posicionamento com a habilitação de escrita da memória. Já no modo leitura ou reprodução , os dados deverão ser lidos na seqüência compatível com a varredura do monitor , de modo a posicionar os dados no formato desejado na tela. Dessa forma, a sincronização da varredura do monitor deve estar "amarrada" com o controle de leitura da memória.

Um estágio **conversor D/A** pode existir, para devolver a forma analógica aos dados lidos da memória, compondo um sinal de vídeo, para monitores analógicos. Ou uma interface para a comunicação com monitores digitais.

O estágio de **gerenciamento e controle (processador)** coordena todas as habilitações e configurações dos conversores, a escrita/leitura e endereçamento da memória e gera as informações de sincronismo para o monitor.[5]

Uma vez digitalizado o sinal e memorizado, se pode afirmar que a imagem se encontra em uma área de memória, onde cada *pixel* assume um valor discreto , podendo ser de zero a 255 , para 8 bits por exemplo. O processador além de organizar a seqüência de leitura para a formação dos *pixels* da imagem, é possível programa-lo para aplicar funções de transferência nos valores discretos dos *pixels*, afim de promover efeitos ou filtragens que tornem a imagem mais adequada ao uso proposto. Tal processo é chamado de pré processamento [6] [7] [8].

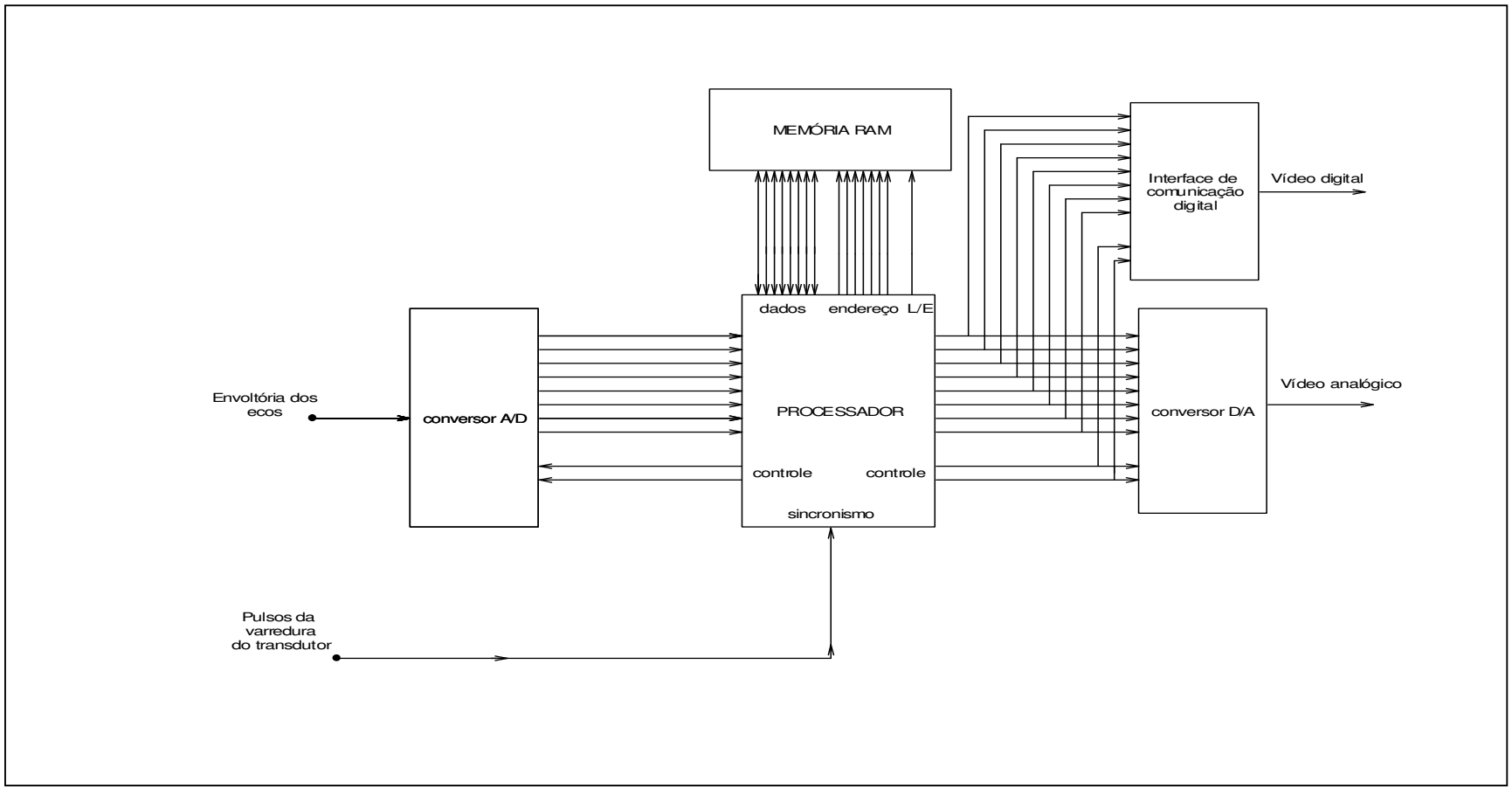


Fig. I.8- Diagrama em blocos do processamento de sinal

I.2.1 Pré-processamento

I.2.1.1 Introdução

Esse ítem é dedicado às técnicas de pré-processamento (aplicadas parcialmente nesse projeto) de imagens digitais . As técnicas de pré-processamento têm como objetivo transformar uma imagem original de forma que a imagem resultante seja mais adequada para obtenção de destaque nas características importantes de serem visualizadas. Em geral, o que se diz tornar uma imagem mais adequada é uma questão de interpretação do usuário, sendo importante a sua experiência. Além disso, as técnicas não são universais, quer dizer, não produzem resultados satisfatórios para todos os tipos de imagens. Desta forma, existe um grande número de técnicas e aqui são apresentadas apenas as que podem contribuir para o realce nos elementos de interesse nas imagens de ultra-som. Os filtros utilizados com o objetivo de pré-processamento de imagens podem ser implementados no domínio espacial ou no domínio de freqüência, fazendo-se a transformada de Fourier da imagem e o uso do Teorema da Convolução.[6] [7] [8]

1.2.1.2 Filtros de Pré-processamento no Domínio Espacial

Nas técnicas de filtragem no domínio espacial trata-se diretamente a matriz de níveis de cinza da imagem digital. Os processamentos no domínio espacial são matematicamente descritos por uma transformação (eq. 1.2)

$$\mathrm{Im_p}(i,j) = T\left[\mathrm{Im}(i,j)\right] \qquad (1.2)$$

onde $\mathrm{Im}(i,j)$ é a imagem original, $\mathrm{Im_p}(i,j)$é a imagem processada e T é um operador em $\mathrm{Im}(i,j)$ definido em uma vizinhança em torno de cada pixel (i,j). Esta vizinhança é definida, em geral como regiões de tamanho 3x3, 5x5, ... nxn, centradas no pixel de referência, veja-se uma ilustração na figura I.9.

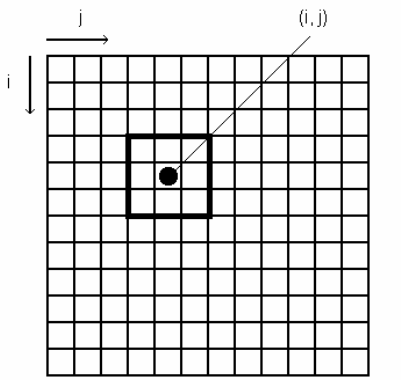


Fig. I.9- Uma vizinhança 3x3 em torno do pixel de coordenadas (i,j) de uma imagem

Quando a vizinhança é 1x1 tem-se a a chamada função de *transformação de níveis de cinza* na forma (eq. 1.3):

$$s = T(r) \tag{1.3}$$

onde r é o nível de cinza na imagem original Im(i,j) e s na imagem transformada $Im_p(i,j)$. As técnicas de transformação de níveis de cinza (ou de transformação de histograma) são denominadas *pixel a pixel*, uma vez que o operador atua em cada pixel independente de sua vizinhança. De outra forma, o valor de nível de cinza, após o processamento, depende apenas de seu valor de nível de cinza original.[6]

I.2.1.3 Expansão de Constraste

Em certas situações têm-se imagens cujos níveis de cinza ficam concentrados em uma pequena faixa da escala 0 – 255. Isto pode ocorrer por um problema de iluminação na aquisição da imagem ou mesmo por um sensor de aquisição com limitada faixa de níveis de cinza. Um exemplo típico de tal situação é apresentado na Figura I.10.

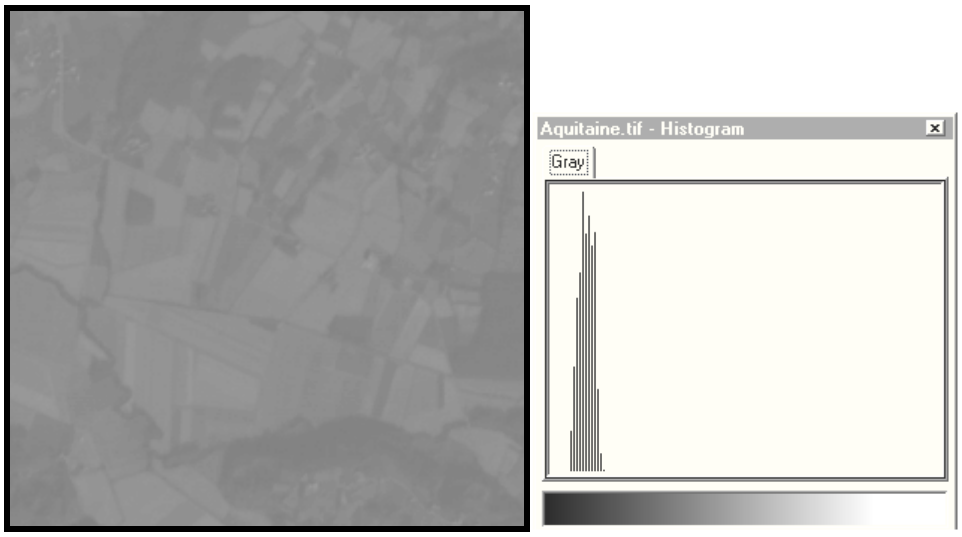


Fig. I.10- Uma imagem de Aquitaine (uma região do sudoeste da França) acompanhada de seu histograma de níveis de cinza. Os níveis estão concentrados na faixa de 10 – 30 na escala 0 – 255.

A idéia então é promover uma expansão de constraste de forma a abranger toda a faixa de níveis de cinza. Aqui se está interessado em transformações do tipo $s = T(r)$, que conduza a um nível de cinza s para cada pixel da imagem original de valor de nível de cinza r.Uma transformação típica é mostrada na figura I.11. [6] [7] [8]

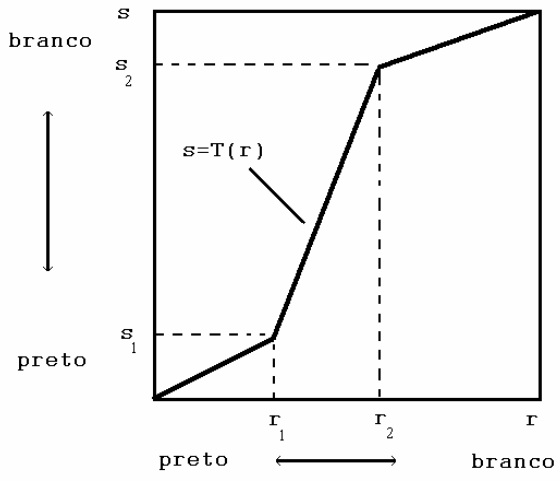


Fig. I.11- Transformação usual para a expansão de contraste.

Em geral, assume-se para as transformações $r_1 \leq r_2$ e $s_1 \leq s_2$. Como casos particulares temos que quando $r_1 = r_2$ e $s_1 = s_2$, nenhuma transformação é operada, isto é, a imagem se mantém inalterada e quando $r_1 = r_2$, $s_1 = 0$ e $s_2 = 255$, tem uma transformação para uma imagem binária.

Assim, diversas transformações $s = T(r)$ podem ser utilizadas. Uma transformação para a expansão de constraste muito usual é dada por (eq. 1.4): [6] [7] [8]

$$s = \begin{cases} 0, \text{se } r \leq r_1 \\ 255x\left(\dfrac{r - r_1}{r_2 - r_1}\right), \text{se } r_1 \leq r \leq r_2 \\ 255, \text{se } r \geq r_2 \end{cases} \qquad (1.4)$$

Na figura I.12 mostra graficamente a transformação dada por esta equação.

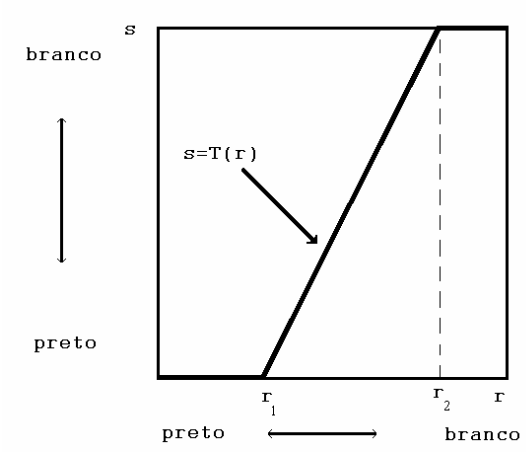


Fig. I.12- A transformação para a expansão de contraste

I.2.1.4 Operações Orientadas a Vizinhança. Máscaras de Convolução

Em operações orientadas a vizinhança, o filtro é geralmente associado a uma mascára que operará em uma vizinhança de cada *pixel* da imagem. Na figura I.13, mostra-se uma máscara 3x3, com coeficientes (pesos) genéricos. Seja um pixel qualquer de uma imagem em

níveis de cinza, onde o centro da mascára é posicionado. Denotando os níveis de cinza do pixel em consideração e da sua vizinhança associada à máscara por $c_1, c_2, c_3, ..., c_9$ a resposta, para o dado *pixel*, dada por um filtro linear será (eq. 1.5): [6] [7] [8]

$$R = p_1c_1 + p_2c_2 + p_3c_3 + ... + p_9c_9 \qquad (1.5)$$

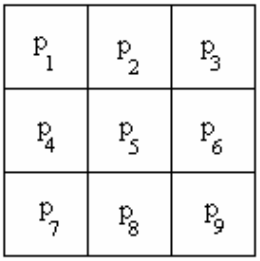


Fig. I.13- Máscara para um filtro 3x3 com coeficientes genéricos. Os níveis de cinza c_1, ..., c_9, estão posicionados seguindo a mesma ordem dos coeficientes.

I.2.1.5 Filtro Passa-Baixa

Os filtros de passa-baixa atenuam as componentes da imagem de alta freqüência (considerando-se o domínio de Fourier) mantendo inalterados os componentes de baixa freqüência. Quer dizer, deixam passar as baixas freqüências. Componentes de alta freqüência ocorrem quando se tem uma variação espacial brusca dos níveis de cinza como é o caso de ruídos e contornos entre fases distintas da imagem. O efeito de aplicação de um filtro de passa-baixa é o de suavizamento (uniformização) dos níveis de cinza de uma imagem.

Um modelo de filtro de passa-baixa consiste na função Gaussiana, contudo o aspecto fundamental de um tal filtro é que todos os seus coeficientes sejam positivos. Ainda, o mais simples é que todos os coeficientes sejam iguais a 1, o denominado filtro da média. Para que a resposta do filtro seja mantida na mesma escala de níveis de cinza que a da imagem original, o resultado de aplicação do filtro deve ser dividido por n^2 para uma máscara de tamanho nxn. [6] [7] [8]

Na figura I.14 são mostrados exemplos de máscaras de tamanhos 3x3 e 5x5.

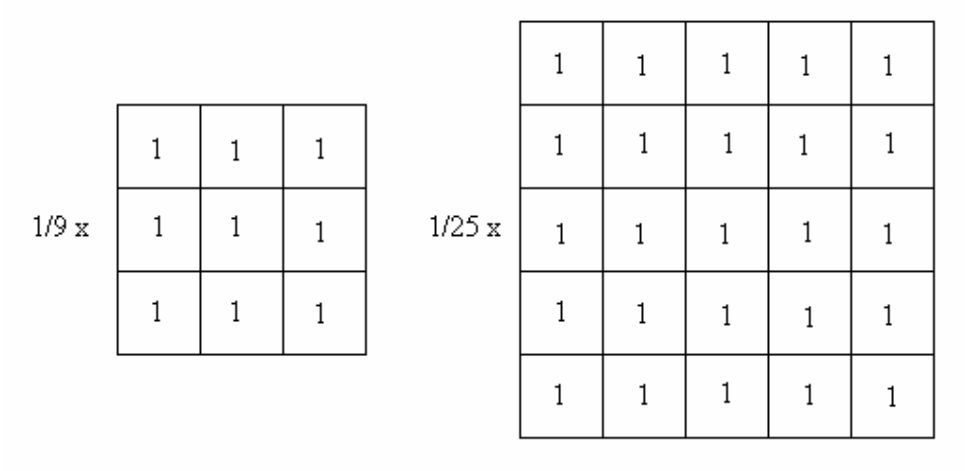


Fig. I.14- Máscaras para filtros passa-baixa 3x3 e 5x5

I.2.1.6 Filtro Mediano

O filtro da média apresenta como limitação, nos casos onde o objetivo é a remoção de ruídos, a sua incapacidade de preservar contornos (bordas) dos objetos bem como detalhes finos. Nestes casos é mais adequada a utilização de um filtro de mediana, um filtro não-linear, onde o valor do pixel central da máscara é transformado pela mediana dos pixels da vizinhança. A mediana *m* de um conjunto de *n* elementos ordenados é o valor tal que metade dos *n* elementos do conjunto apresentem valores inferiores *m* e a outra metade valores superiores a *m*. Assim, se *n* é ímpar a mediana é o valor central do conjunto ordenado. Quando *n* é par a mediana é tomada como a média aritmética dos dois elementos no conjunto ordenado mais próximos ao centro.

Na figura I.15 mostra-se um exemplo esquemático de aplicação do filtro de mediana: o pixel em análise tem valor de nível de cinza 251 sendo modificado para o valor 20, a mediana da vizinhança 3x3.[6] [7] [8]

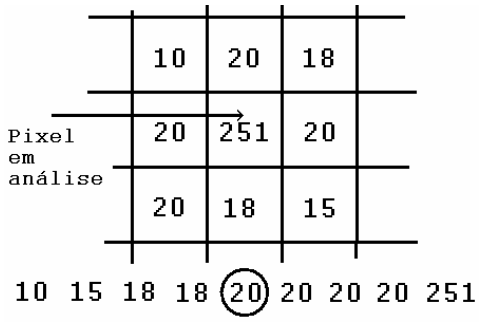


Fig. I.15- Aplicação do filtro de mediana

I.2.1.7 Filtro Passa-Alta

Os filtros passa-alta são usados para o realce de detalhes finos e de contornos dos objetos da imagem, assim sendo, é o mais interessante de ser adotado para as imagens de ultra-som . A resposta de um filtro passa-alta deve ser que a sua máscara associada apresente pesos positivos nas proximidades de seu centro e negativos longe dele. Na figura I.16 mostra-se a máscara de um passa-alta usual. Nota-se que a soma algébrica dos seus coeficientes é nula e desta forma quando aplicado a regiões homogêneas da imagem responderá com valor nulo ou muito baixo, coerente com a filtragem passa-alta. [6] [7] [8]

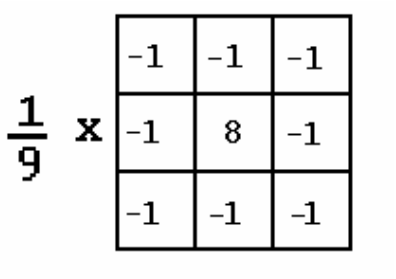


Fig. I.16-Máscara 3x3 de um modelo usual de filtro passa-alta

I.2.1.8 Realce por Diferenciação

A diferenciação tem como efeito enfatizar os componentes de alta freqüência (detalhes finos e contornos) presentes na imagem. Em processamento de imagens um dos operadores mais usuais de diferenciação é o gradiente.

Para o cálculo do gradiente em imagens digitais usam-se várias aproximações sendo o Operador de Sobel um dos mais importantes. Dada uma imagem Im(i,j) a aproximação de Sobel para o gradiente é dada por:

$$\left|\nabla \mathrm{Im}(i,j)\right| = \left|(c_7 + 2c_8 + c_9) - (c_1 + 2c_2 + c_3)\right| + \left|(c_3 + 2c_6 + c_9) - (c_1 + 2c_4 + c_7)\right| \qquad (1.6)$$

onde os níveis de cinza c_1, ..., c_9 estão posicionados como na convenção da figura I.17.a. O pixel central, com nível de cinza c_5 será modificado de acordo com a Equação (1.6). São

usadas duas máscaras para a representação desta Equação, que são mostradas nas figuras I.17, (b) e (c), associadas aos dois termos (módulos) do lado direito da Equação (1.5) [6] [7] [8]

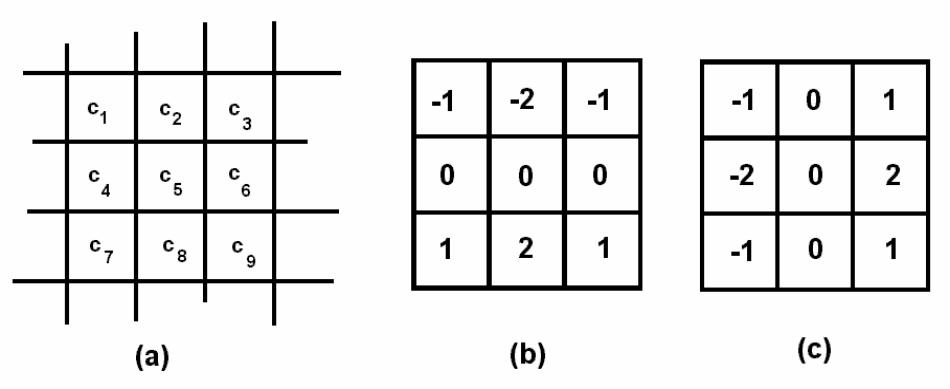


Fig. I.17- Máscaras para o Operador de Sobel

I.3 Descrição parcial do Padrão ITU-656 (*International Telecommunications Union*)

Sempre que se pretende efetuar a digitalização de uma imagem ou a composição de uma imagem digital para ser reproduzida em receptor de TV, se deve seguir as recomendações de algum padrão compatível com o receptor. O padrão de digitalização irá definir a seqüência de amostragem, o número de bits para quantização, os códigos de sincronismo (sinais de controle), os intervalos de apagamento, o entrelaçamento ou não dos campos, o número de linhas digitalizadas e etc.... O ITU-656 é uma definição padrão de formato digital de vídeo de 8 bits "Y""Cb""Cr"(4:2:2) para a transmissão digital. A letra "Y" representa o sinal de luminância, "Cb" a componente B-Y e "Cr" a componente R-Y. A sequência 4:2:2 informa que para cada 4 amostras do sinal "Y" , existirão 2 amostras da componente "Cb" e 2 da componente "Cr". Será feita uma abordagem parcial, não tendo a ambição de explicar todos os detalhes deste padrão. [9] [10]

- **Quadros –** Um quadro completo de vídeo entrelaçado consiste de 2(dois) campos quase idênticos. O primeiro campo contêm as linhas impares e o segundo campo contêm as linhas pares.

- **Campos –** Um campo encerra 3(três) regiões distintas, primeiro período de apagamento vertical, vídeo ativo e segundo período de apagamento vertical. A diferença entre o primeiro e o segundo campo é meramente o número de linhas do primeiro período de apagamento vertical, uma vêz que no primeiro campo este tem uma linha a menos que no segundo campo.

O quadro da figura I.18 ilustra o número de linhas e a codificação dos campos para os sistemas NTSC-M e PAL-N. [9] [10]

Linhas		Campos/Apag.Vertical		Descrição da linha
PAL	NTSC	F	V	
22	19	'0'	'1'	Campo 1 - 1° Apagamento Vertical
288	240	'0'	'0'	Campo 1 – Vídeo ativo
2	3	'0'	'1'	Campo 1 - 2° Apagamento Vertical
23	20	'1'	'1'	Campo 2 - 1° Apagamento Vertical
288	240	'1'	'0'	Campo 2 – Vídeo ativo
2	3	'1'	'1'	Campo 2 - 2° Apagamento Vertical
625	525			

Fig. I.18- Quadro com o número de linhas e códigos dos campos, ITU-656

Códigos de campo - Duas colunas são extremamente importantes de serem observadas: A coluna “F”(bit referente ao número do Campo) e a coluna “V”(bit referente ao apagamento Vertical). Os valores destas colunas indicam a posição vertical de uma linha.

O valor lógico zero na coluna “F”, indica que esta linha pertence ao campo 1, enquanto o valor lógico um indica que esta linha pertence ao campo 2.

O valor lógico zero na coluna “V”, indica que esta linha é parte do vídeo ativo, enquanto o valor lógico um indica que esta linha é de apagamento vertical.

- **Codificação das Linhas –** Conforme descrito no quadro da figura I.19, as linhas são divididas em 4(quatro partes): Código EAV, Apagamento horizontal, Código SAV e Vídeo ativo.

  O código EAV(Fim de Vídeo Ativo) e SAV(Início de Vídeo Ativo) são compostos de 4 bytes. Os 3(três) primeiros bytes indicam o sincronismo horizontal, sendo fixos em “255 (0xFF)”, “0 (0x00)” e “0 (0x00)”, e o último byte indica o código da linha, onde aparecem os bits “F”, “V” e “H”.

  O bit “H” possui valor lógico um durante a ocorrência de Código EAV, passando para zero no Código SAV.

  Os *bytes dos* códigos EAV e SAV são construídos segundo o quadro da figura I.20. [9]

  As funções OU-Exclusivo destes *bytes* de controle podem ser utilizados para detectar e corrigir erros nos valores de F, V e H. Um ou até dois bits de erro podem ser corrigidos. [9] [10]

Código EAV				Apagamento horizontal				Código SAV				Vídeo ativo			
255	0	0	EAV	Cb	Y	Cr	Y	255	0	0	SAV	Cb	Y	Cr	Y
4 bytes				280(268) bytes				4 bytes				1440 bytes			

Fig. I.19- Quadro com o codificação das linhas

Bit	7	6	5	4	3	2	1	0
Valor	‘1’	F	V	H	V xor H	F xor H	F xor V	F xor V xor H

Fig. I.20 - Formação dos *bytes* dos códigos de EAV e SAV

O período de apagamento horizontal consiste de 280 bytes(PAL) ou 268 bytes(NTSC) e o período de Vídeo Ativo de uma linha consiste de 1440 bytes(PAL e NTSC). Esses bytes são sequências de “Cb””Y””Cr””Y”, onde: [9]

Y (Luminância), admite valores entre 16 – 235.
Cb (Componente B-Y) admite valores entre 16 – 240.
Cr (Componente R-Y) admite valores entre 16 – 240.

- **Um quadro completo –** Para exibir o quadro mais simples, que seria uma tela preenchida somente com uma cor, serão usados os valores de Y, Cb e Cr conforme descrito abaixo:

  Pixel azul
  Y = 41
  Cb = 240
  Cr = 110

  É usual utilizar *pixels* pretos no preenchimento da área de apagamento de vídeo, quando dados em anexo, como som e tele-texto, não são transmitidos.

- *Pixel* preto
  Y = 16
  Cb = 128
  Cr = 128

  Então, usando as descrições acima de quadro, campo e linha; poderá ser construída uma tela inteiramente azul. A figura I.21 representa a descrição de uma tela azul para o sistema PAL com valores para NTSC em parênteses. [9]

Campo 1 – Primeiro apagamento vertical – Ocorre por 22(19) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
255	0	0	182	128	16	18	16	255	0	0	171	128	16	128	16	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vêz				Ocorre 360 vêzes				

Campo 1 – Vídeo Ativo – Ocorre por 288(240) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
255	0	0	157	128	16	18	16	255	0	0	128	240	41	110	41	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vez				Ocorre 360 vêzes				

Campo 1 – Segundo apagamento vertical – Ocorre por 2(3) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
255	0	0	182	128	16	18	16	255	0	0	171	128	16	128	16	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vêz				Ocorre 360 vêzes				

Campo 2 – Primeiro apagamento vertical – Ocorre por 23(20) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
236	0	0	241	128	16	18	16	255	0	0	236	128	16	128	16	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vêz				Ocorre 360 vêzes				

Campo 2 – Vídeo Ativo – Ocorre por 288(240) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
255	0	0	218	128	16	18	16	255	0	0	199	240	41	110	41	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vêz				Ocorre 360 vêzes				

Campo 2 – Segundo apagamento vertical – Ocorre por 2(3) linhas																
Código EAV				Apagamento horizontal				Código SAV				Vídeo Ativo				
255	0	0	241	128	16	18	16	255	0	0	236	128	16	128	16	
Ocorre 1(1) vêz				Ocorre 70(67) vêzes				Ocorre 1(1) vêz				Ocorre 360 vêzes				

Fig. I.21- Quadro completo para tela azul

II Metodologia

II.1 Iintrodução

Esse capítulo descreve a análise do sinal proveniente da demodulação dos ecos, disponível na saída do circuito acoplado a sonda transdutora (desenvolvido na dissertação de mestrado de Paulo Ernesto Moreira, CEFET-RJ, novembro / 2007). E partindo desse sinal, quais foram as soluções encontradas para se conseguir gerar a imagem, com a resolução pretendida, compatível com receptores de TV e empregando circuitos / módulos, com operação sem vínculo permanente com um computador , o que torna o sistema portátil.

II.2 Análise do sinal adquirido pelo transdutor

Uma análise rigorosa da envoltória resultante da demodulação (detecção de envoltória) dos ecos , revelou algumas características importantes para a construção do módulo de aquisição A/D . Tais como :

- O transdutor utilizado nos testes produz ecos de 8MHz a 10,8MHz. Faixa essa descoberta pela medição da FFT dos sinais dos ecos da figura II.1.

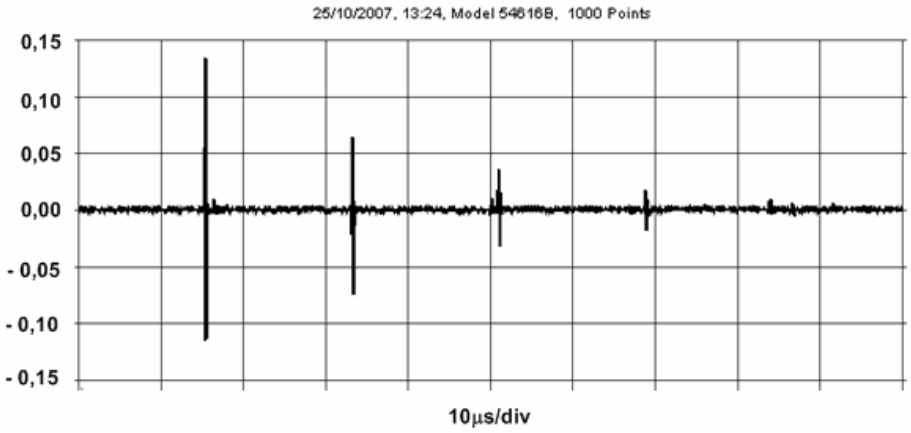


Fig. II.1 – sinal de eco

- Os corpos de prova empregados são de estrutura metálica ou de acrílico , com espessura variando de 1 a 100mm , imersos em água , com a lente do transdutor também imersa e afastada em até 10mm da superfície do corpo . Nessas condições, o

eco de amplitude ainda significativa com maior atraso, se encontra na faixa de 50µs defasado do pulso de excitação do transdutor.

-O deslocamento lateral do transdutor, promovido sobre o corpo de prova é de até 72mm e a resolução pretendida é de 300µm.

- As envoltórias dos ecos podem ser amplificadas, de forma a atingirem a faixa de amplitude para o codificador de sinal composto de vídeo , cujo projeto proposto prevê de 700mV.

Com base nos testes descritos , alguns seguimentos do projeto foram definidos, como :

-A freqüência de repetição dos pulsos de excitação do transdutor adotada no projeto é de 15.734,26Hz , valor esse conveniente por ser a freqüência de varredura das linhas de um receptor de TV convencional padrão M [11] . Dessa forma , os ecos referentes a exploração de profundidade do corpo de prova , constituirão uma linha horizontal da imagem gravada, que será convertida em coluna (linha vertical) na imagem reproduzida. Diante do uso dessa freqüência , o intervalo de tempo entre os pulsos de excitação é de 63,5µs, conseguindo abranger os ecos mais significativos observados .

-A freqüência de amostragem adotada é de 30MHz ou 30MS/s (30 *Mega-Sample por segundo) ,* visto que , a freqüência das envoltórias dos ecos é de, no máximo 10MHz , o teorema de *Nyquist* está sendo respeitado. [5]

-Partindo de observações de imagens adquiridas por outros equipamentos de ultra-som, se determinou que uma imagem com 256 tons de cinza possui definição satisfatória, portanto palavras de 8 bits foram adotadas na codificação das amostras.

- A imagem será composta por 240 colunas, face a resolução lateral pretendida de 300µm , em um percurso total de 72mm a cada ciclo de varredura do transdutor.

-Cada coluna será composta por 360 pontos, logo para uma profundidade máxima de 110mm, a resolução axial também resultará em 300µm.

II.3 Análise do circuito codificador de sinal composto de vídeo.

Os pulsos dos ecos captados pela sonda são detectados em envoltória e aplicados no circuito em questão. Cabe a esse circuito o compromisso de gerar um sinal composto de vídeo, no padrão M, com todas as informações de controle necessárias (sincronismo vertical , sincronismo horizontal, equalização e apagamento), onde as envoltórias dos ecos constituirão as informações de vídeo ou luminância das linhas. Além disso , o circuito deverá fornecer uma amostra dos pulsos de sincronismo horizontal para sincronizar a geração dos pulsos de excitação do transdutor. Sem dúvida que as envoltórias dos ecos serão "*mixadas*" no intervalo de escrita das linhas, com polaridade adequada para que o aumento da amplitude das mesmas configure um aumento de luminância . Como ilustra a figura II.2. [11]

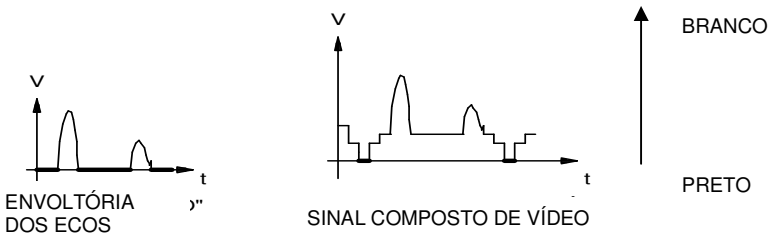


Fig. II.2- Sinal composto de vídeo com as envoltórias dos ecos.

A figura II.3 , apresenta o circuito desenvolvido. Onde *o* ***IC1 (SAA1101*** *da* ***Philips)*** é responsável pela geração dos sinais de controle, estando configurado para operar no padrão M, com sistema NTSC e para tanto necessita de um sinal de *clock de* 5,034963MHz (320 x 15.734,26 Hz), controlado por cristal (X1) [12]. Já o ***IC2 (AD724*** *da* ***Analog Devices*****)** , recebe os sinais de controle (gerados pelo IC1) pelas entradas de sincronismo vertical e horizontal e as envoltórias dos ecos pelas entradas de sinais RGB, executando a montagem do sinal composto de vídeo. Para tanto, o IC2 também está configurado para o padrão M, com sistema NTSC e opera com um sinal de *clock* de 3,579545MHz (freqüência da sub-portadora de cor), também controlado por cristal (X2) [13].

A razão das envoltórias dos ecos entrarem em paralelo nas entradas RGB do IC2, é porque se deseja variações em escala de cinza.

O pino 22 do IC1 fornece uma amostra do sincronismo horizontal para sincronizar a geração dos pulsos de excitação do transdutor.

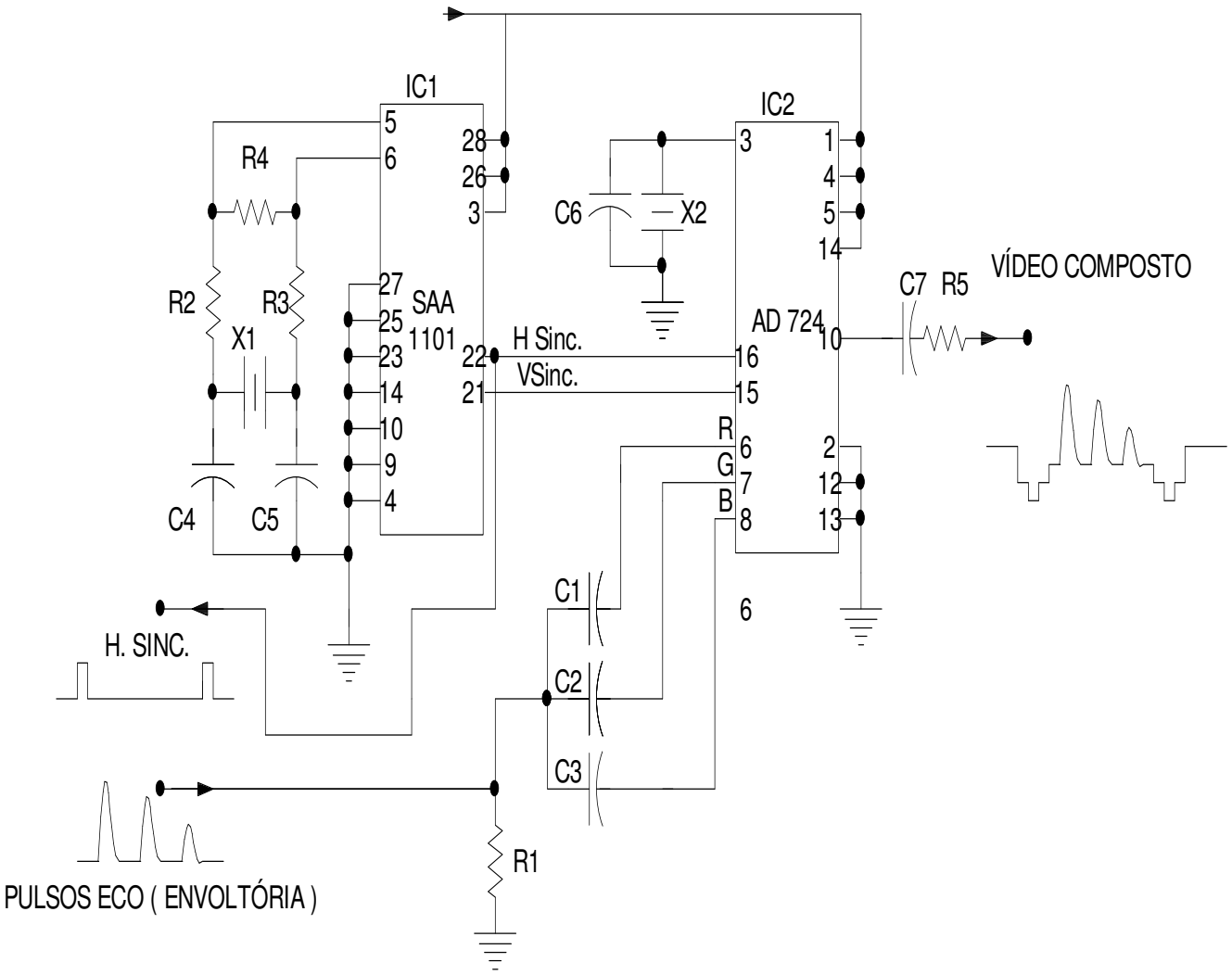
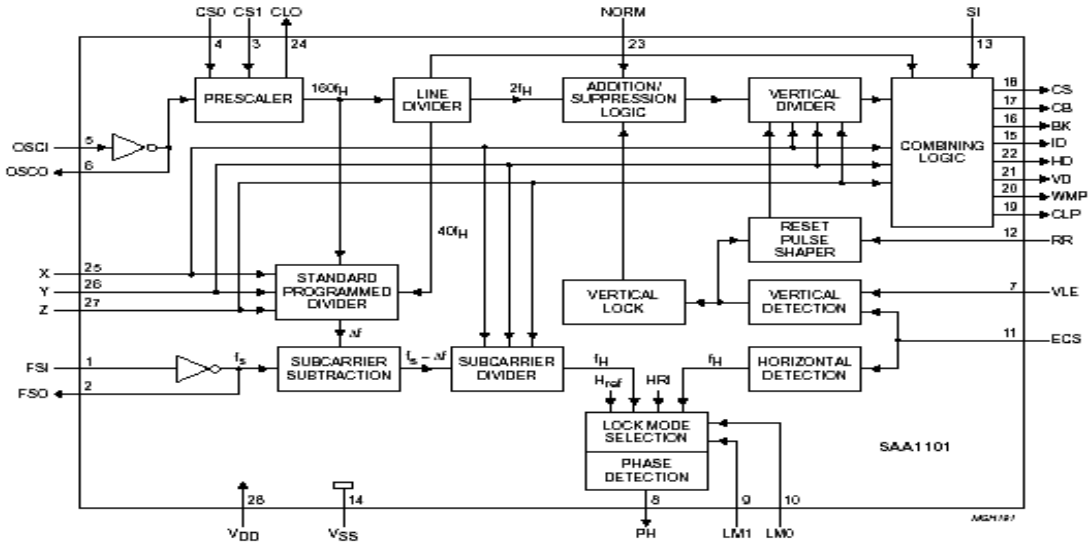
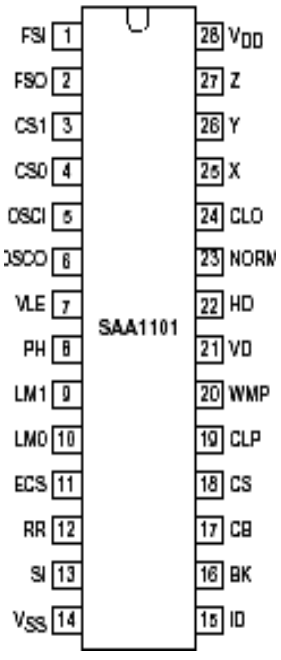


Fig. II.3- Circuito codificador de sinal composto de vídeo

A figura II.4 apresenta o diagrama em blocos e as funções dos pinos do ***IC1 (SAA1101*** *da* ***Philips).*** [12]



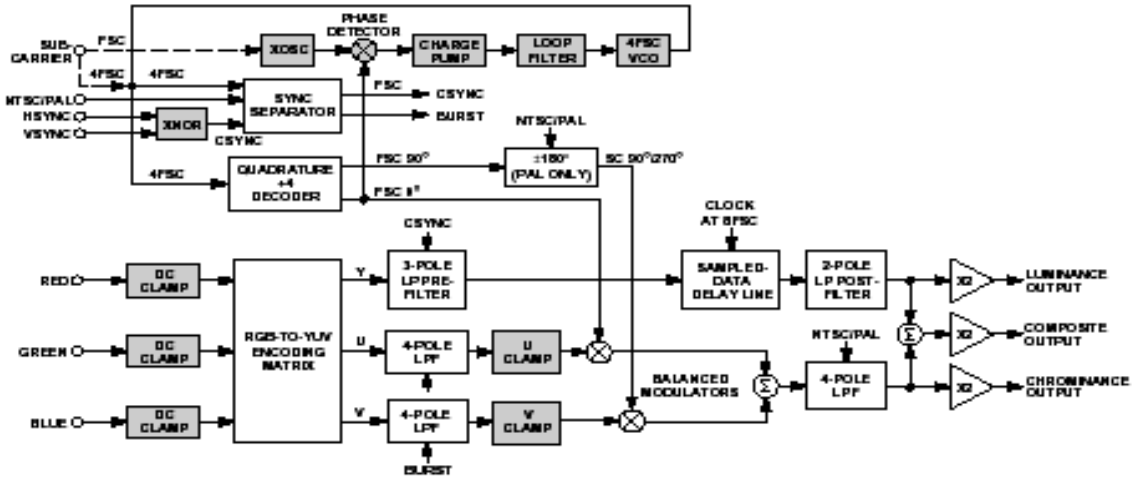


PINNING

SYMBOL	PIN	DESCRIPTION
FSI	1	subcarrier oscillator input, where f_{max} = 5 MHz
FSO	2	subcarrier oscillator output
CS1	3	clock frequency selection - CMOS input
CS0	4	clock frequency selection - CMOS input
OSCI	5	clock oscillator input, where f_{max} = 24 MHz
OSCO	6	clock oscillator output
VLE	7	vertical in-lock enable - CMOS input
PH	8	phase detector output - 3-state output
LM1	9	lock mode selection - CMOS input
LM0	10	lock mode selection - CMOS input
ECS	11	external composite sync. signal - CMOS Schmitt-trigger input
RR	12	frame reset - CMOS Schmitt-trigger input
SI	13	set identification, used to set the correct field sequence in PAL-mode. The correction (inversion of fH2) is done at the left-hand slope of the SI-pulse. Minimum pulse width is 800 ns. CMOS Schmitt-trigger input.
V_{SS}	14	ground
ID	15	identification - push-pull output
BK	16	burst key (PAL/NTSC), chroma-blanking (SECAM) - push-pull output
CB	17	composite blanking - push-pull output
CS	18	composite sync. - push-pull output
CLP	19	clamp pulse - push-pull output
WMP	20	white measurement pulse-3-state output
VD	21	vertical drive pulse - push-pull output
HD	22	horizontal drive pulse - push-pull output
NORM	23	used with X, Y and Z to select TV system; NORM = 0, 625/525 line mode (standard); NORM = 1, 624/524 line mode - CMOS input
CLO	24	clock output - push-pull output
X	25	TV system selection input - CMOS input
Y	26	TV system selection input - CMOS input
Z	27	TV system selection input - CMOS input
V_{DD}	28	voltage supply

Fig. II.4- Diagrama em blocos e as funções dos pinos do ***IC1 (SAA1101** da **Philips).*** [12]

A figura II.5 apresenta o diagrama em blocos e as funções dos pinos do ***IC2 (AD724** da **Analog Devices*)**. [13]



PIN FUNCTION DESCRIPTIONS

Pin	Mnemonic	Description	Equivalent Circuit
1	STND	A Logical HIGH input selects NTSC encoding. A Logical LOW input selects PAL encoding. CMOS/TTL Logic Levels.	Circuit A
2	AGND	Analog Ground Connection.	
3	FIN	FSC clock or parallel-resonant crystal, or 4FSC clock input. For NTSC: 3.579 545 MHz or 14.318 180 MHz. For PAL: 4.433 619 MHz or 17.734 480 MHz. CMOS/TTL Logic Levels for subcarrier clocks.	Circuit B
4	APOS	Analog Positive Supply (+5 V ± 5%).	
5	ENCD	A Logical HIGH input enables the encode function. A Logical LOW input powers down chip when not in use. CMOS/TTL Logic Levels.	Circuit A
6	RIN	Red Component Video Input. 0 to 714 mV AC-Coupled.	Circuit C
7	GIN	Green Component Video Input. 0 to 714 mV AC-Coupled.	Circuit C
8	BIN	Blue Component Video Input. 0 to 714 mV AC-Coupled.	Circuit C
9	CRMA	Chrominance Output.* Approximately 1.8 V peak-to-peak for both NTSC and PAL.	Circuit D
10	COMP	Composite Video Output.* Approximately 2.5 V peak-to-peak for both NTSC and PAL.	Circuit D
11	LUMA	Luminance plus SYNC Output.* Approximately 2 V peak-to-peak for both NTSC and PAL.	Circuit D
12	SELECT	A Logical LOW input selects the FSC operating mode. A Logical HIGH input selects the 4FSC operating mode. CMOS/TTL Logic Levels.	Circuit A
13	DGND	Digital Ground Connections.	
14	DPOS	Digital Positive Supply (+5 V ± 5%).	
15	VSYNC	Vertical Sync Signal (If using external CSYNC set at > +2 V). CMOS/TTL Logic Levels.	Circuit A
16	HSYNC	Horizontal Sync Signal (or CSYNC signal). CMOS/TTL Logic Levels.	Circuit A

Fig. II.5- Diagrama em blocos e as funções dos pinos do ***IC2 (AD724** da **Analog Devices*)**. [13]

Caso o sinal composto de vídeo gerado fosse aplicado diretamente na entrada de vídeo de receptor de TV, a visão axial (de profundidade) do corpo de prova se localizaria ao longo da linha , ou seja , na horizontal da tela , além do que , as linhas seriam repetições do mesmo ponto , a menos que, o transdutor se deslocasse na velocidade da freqüência de varredura horizontal (15.734,26Hz). A figura II.6 ilustra como ficaria a formação da imagem, nessa hipótese.

Face ao exposto, se deve presumir que a geração da imagem não pode ocorrer desta forma, em tempo real. Daí, surge a necessidade de um circuito, que converta o sinal composto de vídeo de analógico para digital, de modo, a ser memorizado e tratado, até que assuma as condições necessárias para ser aplicado no *display.*

O próximo item aborda o módulo empregado para executar a referida conversão A/D, memorização e tratamento digital.

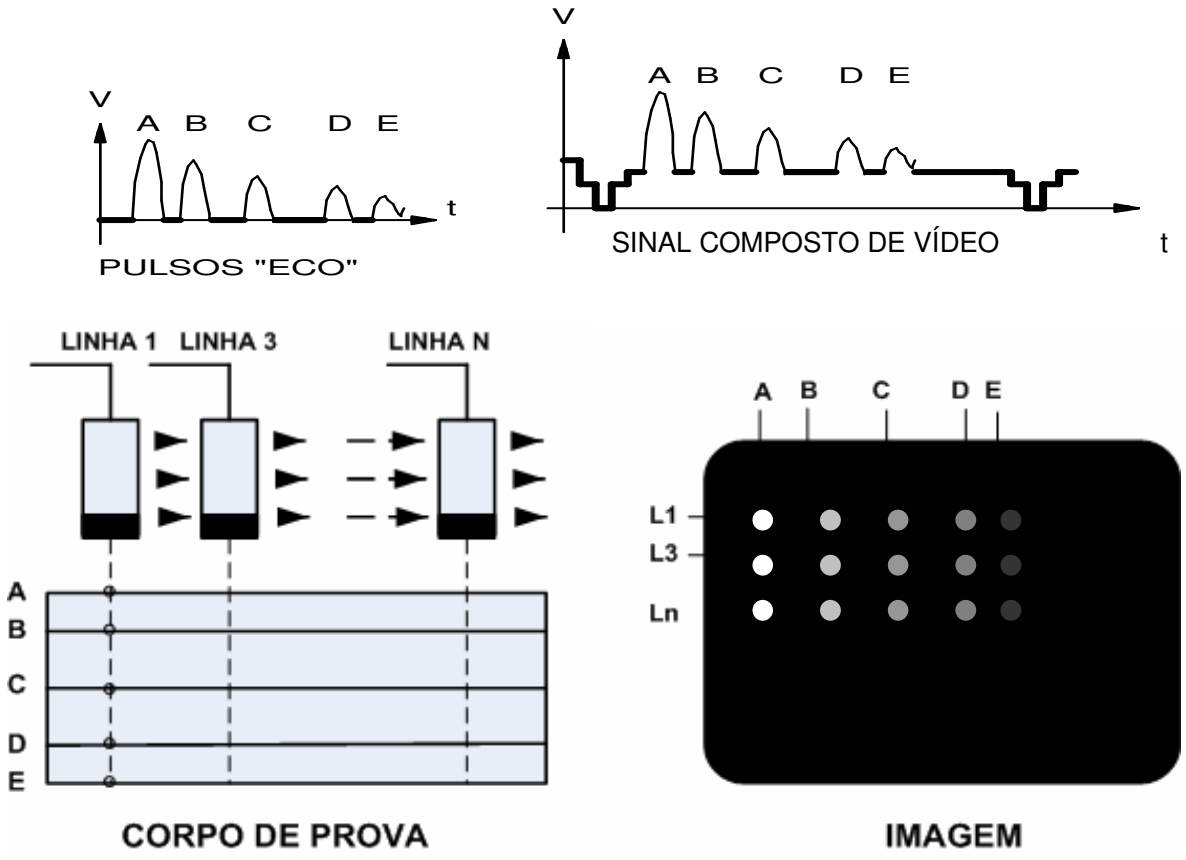


Fig. II.6- Imagem formada em tempo real

II.4 Análise do processamento digital , *hardware e software* envolvidos.

II.4.1- *Hardware*

O módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices*** é uma placa de avaliação do processador ***ADSP-BF533 Blackfin*** (Características técnicas especificadas no apêndice 2) ***,*** utilizada em uma variada gama de aplicações. Este módulo foi o selecionado para o processamento digital do sinal composto de vídeo.

Durante a etapa de desenvolvimento, a ***EZ-KIT Lite*** permite a depuração do ***software*** em construção, via uma conexão USB ou RS-232, em que o ambiente de desenvolvimento ***VisualDSP++*** é executado no PC. Uma vêz superada esta etapa, o módulo funciona em modo ***stand-alone,*** executando o software previamente gravado em sua memória ***FLASH***. [13]

As características do *hardware* do módulo são:

- ***Processador ADSP-BF533 Blackfin®*** (As especificações técnicas se encontram no apêndice 2).
- ***Memória SDRAM de 64 MB (32M x 16-bit)***
- ***Memória FLASH de 2 MB (512K x 16-bit x 2)***
- ***Conversor A/D de vídeo ADV7183 com 3 entradas (tomadas RCA)***
- ***Conversor D/A de vídeo ADV7171 com 3 saídas (tomadas RCA)***
- ***Driver de linha/receptor ADM3202 RS-232***

A figura II.7 ilustra o diagrama em blocos do *hardware* do módulo supra-citado. [13]

II.4.2- Processamento do sinal digital

O sinal composto de vídeo é aplicado no ***IC ADV 7183*** , que faz a conversão de analógico para digital, de acordo com as configurações determinadas pelo *software*. O programa desenvolvido, configura o conversor A/D para operar no padrão “M”, sistema NTSC e padrão de digitalização ITU 656 [13].O sinal digitalizado será entregue ao processador ***IC ADSP-BF533***, que fará a gravação dos bits na memória ***SDRAM de 64 MB***. O processador está configurado pelo programa para gravar os quadros da imagem digitalizada em uma determinada área de memória , seguindo as recomendações do padrão ITU 656. Entretanto, o processador só autoriza a captura e gravação de um novo quadro de imagem, quando recebe um pulso que informa o deslocamento lateral e o conseqüente posicionamento do transdutor. É possível perceber que em cada posição lateral do transdutor, um novo quadro será gravado com todas linhas iguais (repetidas), onde qualquer uma delas corresponde a leitura axial (de profundidade) que o transdutor está realizando do corpo de prova naquele ponto.

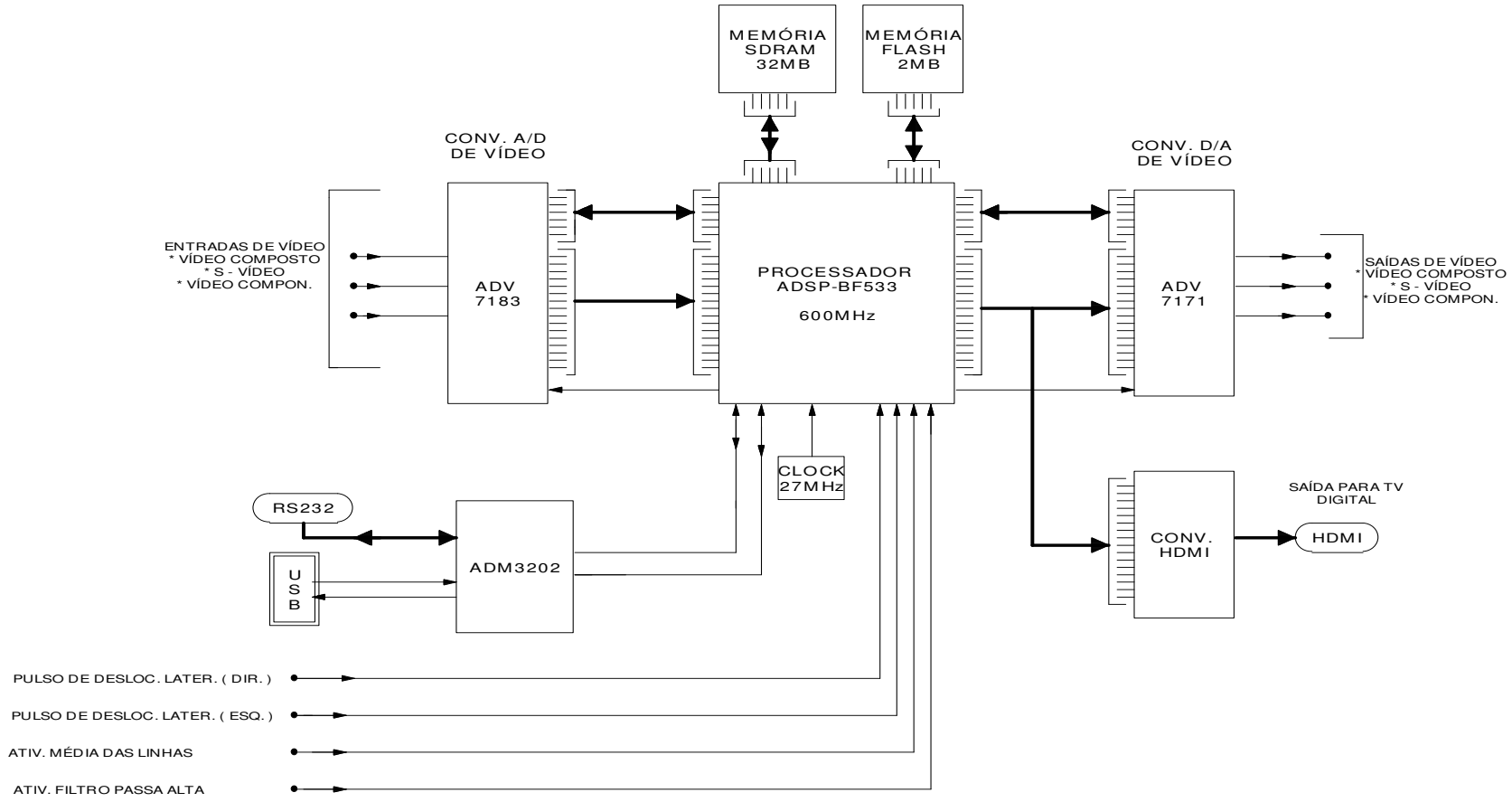


Fig. II.7- Diagrama em Blocos do *Hardware* do módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Device*** [13]

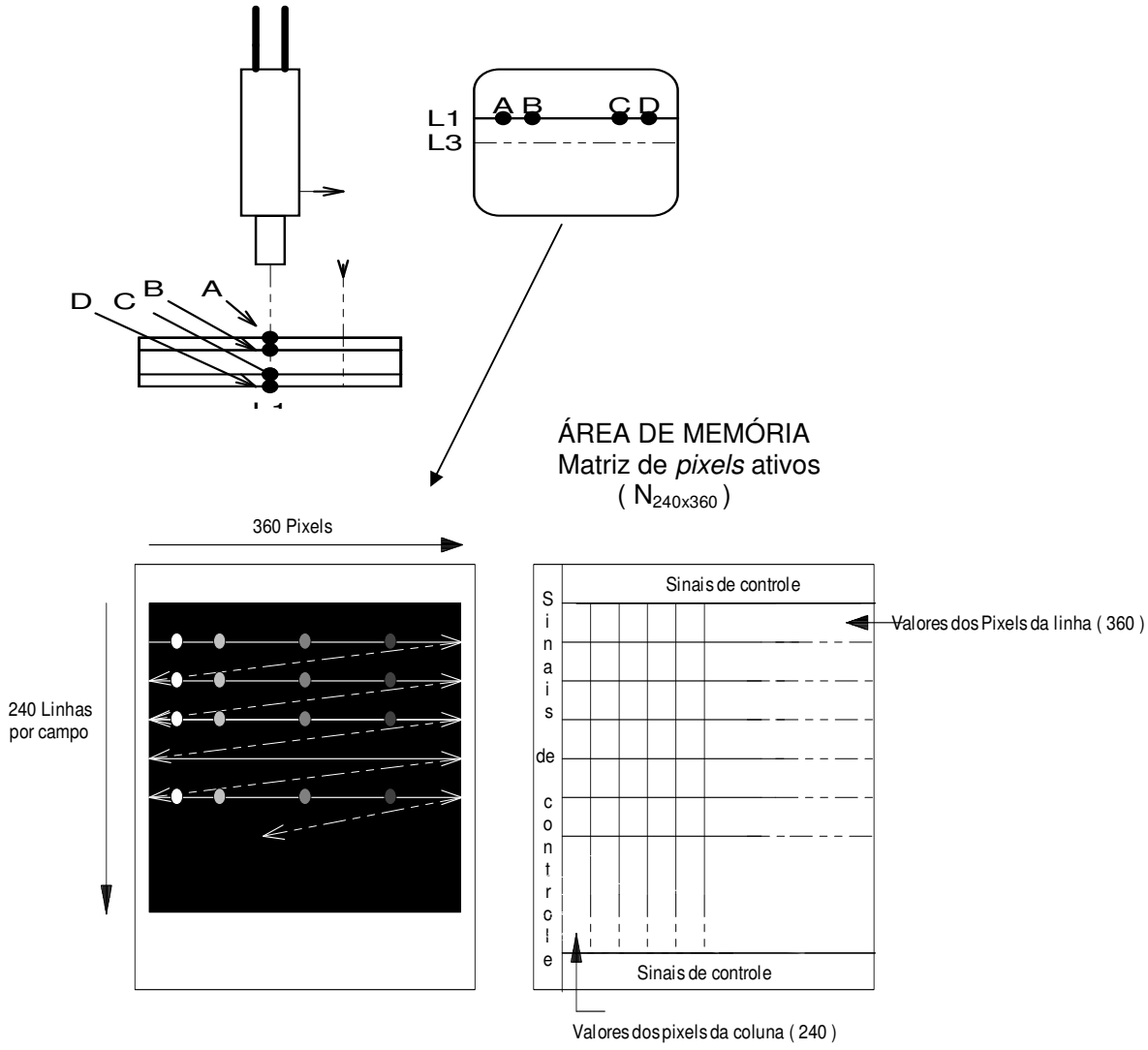


Fig. II.8- Digitalização de um campo de imagem

A figura II.8 ilustra a digitalização de um campo de imagem, denotando as áreas fixas de memória onde serão gravados os sinais de controle e as áreas onde os *pixels* das linhas repetidas se encontram gravados. A configuração determinada pelo padrão ITU 656 é de 240 linhas por campo de vídeo ativo e 360 *pixels* de vídeo ativo por linha. Sendo cada *pixel* composto por 4 *bytes*, na sequência “Cb””Y””Cr””Y”.

Da matriz de *pixels* ativos ($N_{I,J}$) composta pelo campo gravado com linhas repetidas, onde o índice “I” representa o número de linhas e o “J” o número de colunas, o processador deverá compor a linha, para constituir uma coluna da imagem formada no *display.*

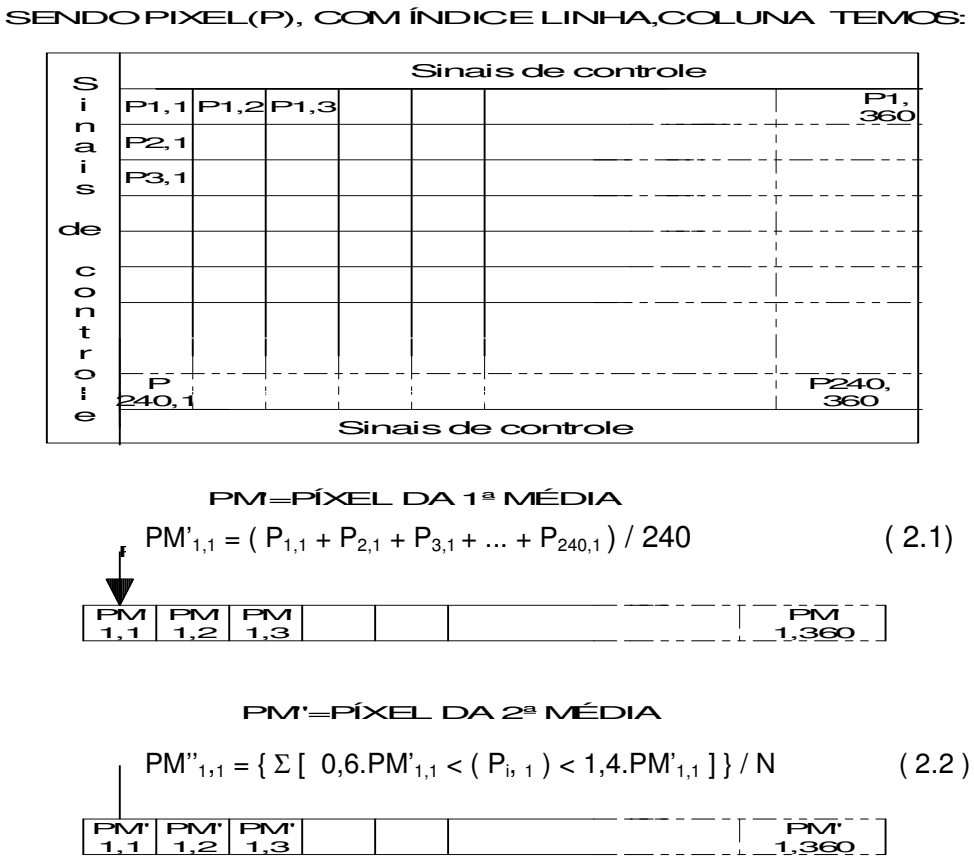
Caso a linha fosse escolhida aleatoriamente, haveria o risco de ser aproveitada uma linha com falhas. Para evitar tal inconveniente é gerada uma linha, cujo valor de cada *pixel* corresponde ao valor médio dos *pixels* de cada coluna ($PM'_{I,J}$ – eq. 2.1).

Contudo, um ou outro *pixel* corrompido possivelmente destorceria a média final. Para tanto , 2 médias são realizadas em cada coluna. [18]

A 1ª é extraída com todos os valores da coluna e é comparada individualmente com o valor de cada *pixel* da mesma coluna, verificando os de maior afastamento. Os *pixels* que se afastarem mais de 40% da média (valor descoberto empiricamente), para mais ou para menos, serão excluídos.

A 2ª média é extraída com os valores dos *pixels* não excluídos de cada coluna e o resultado dessa média, em cada coluna, constituirá a linha a ser apresentada, no sentido vertical, ou seja, como coluna da imagem formada no *display*. O *píxel* resultante da 2ª média de cada coluna é dito $PM''_{I,J}$ (eq. 2.2).

A figura II.9 mostra a sequência para a composição da linha resultante.



$$PM'_{1,1} = (P_{1,1} + P_{2,1} + P_{3,1} + ... + P_{240,1}) / 240 \qquad (2.1)$$

$$PM''_{1,1} = \{ \Sigma [\ 0{,}6.PM'_{1,1} < (P_{i,1}) < 1{,}4.PM'_{1,1}] \} / N \qquad (2.2)$$

Onde:N= Número de *Pixels* da 1ª coluna, cujo valor se encontre no intervalo: **] 0,6.PM'$_{1,1}$, 1,4.PM'$_{1,1}$ [** . Com o índice "i" variando de **1 a 240**.

Fig. II.9- Sequência para a composição da linha resultante

A linha composta, deverá ser gravada em outra área da memória, porém em forma de coluna , para a composição da imagem a ser reproduzida. Como ilustrado na figura II.10.

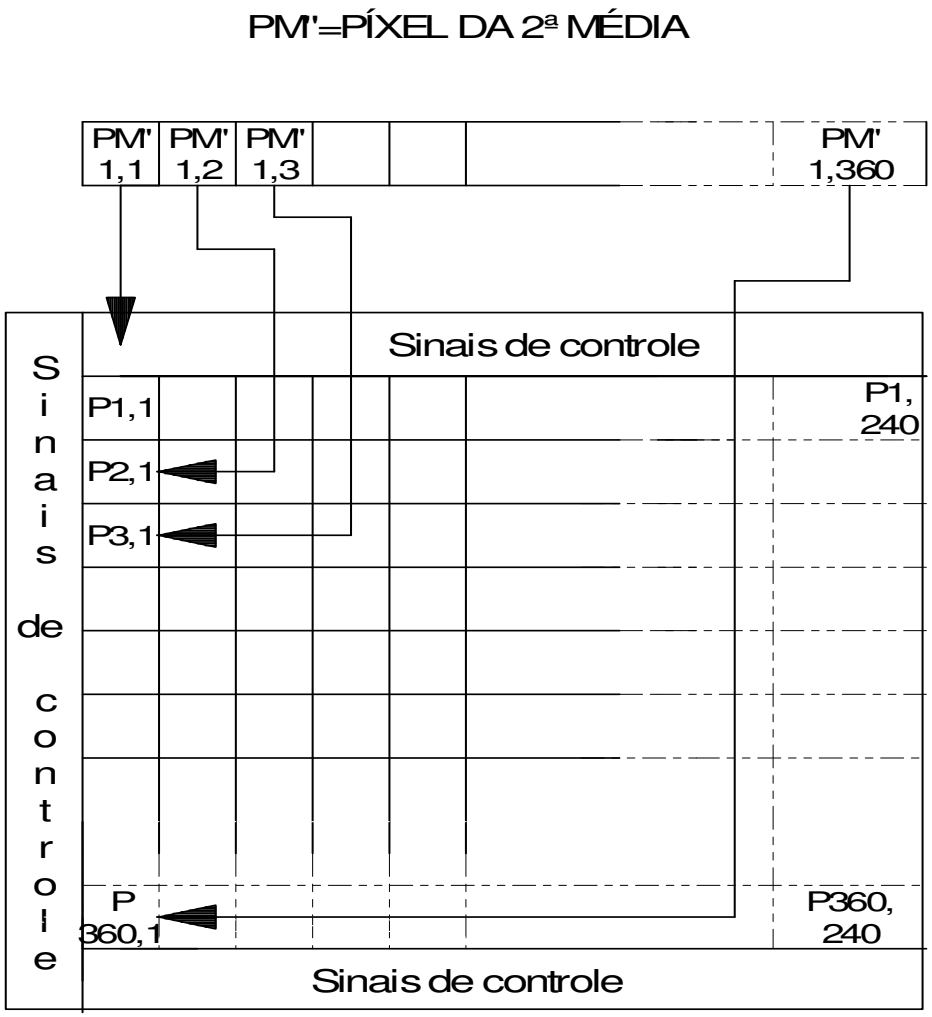


Fig. II.10- Conversão da Linha em coluna

A cada deslocamento lateral do transdutor, o processo se repete e uma nova coluna de *pixels* será montada, com a leitura axial do próximo ponto do corpo de prova. Dessa forma, ao término da exploração do percurso lateral , uma nova matriz de *pixels* será formada, com 360 linhas e 240 colunas, constituindo da imagem a ser reproduzida. Cabe lembrar que, enquanto não ocorrer o deslocamento lateral do transdutor, o processador repete as mesmas colunas, já montadas, em todos os quadros, de modo que, o observador terá a visão dos pontos explorados até então.

Além de todo esse processamento, é possível ativar filtros, onde o processador altera os valores dos *pixels* , mediante uma função de convolução, a fim de realçar aspectos da imagem. Nesse projeto se emprega um filtro passa alta, descrito no próximo item. O bits dos *pixels* das colunas montadas, são repetidos quadro a quadro e aplicados ao ***IC ADV 7171***, que faz a conversão de digital para analógico, compondo o sinal composto de vídeo em sua saída, no padrão “M”, com sistema NTSC e portanto compatível com os receptores de TV analógicos [13].

Uma interface HDMI (*High-Definition Multimedia Interface*) recebe os dados no formato digital, para a aplicação direta em receptores de TV digitais.

As figuras II.11 e II.12 apresentam os quadros com o código comentado em linguagem C++ para o cálculo das médias.

```
// Suponha que a matriz " image1r"  de tamanho Ni x Nj armazene as Ni linhas (de tamanho Nj,
// cada uma) que desejamos "misturar".
// O vetor "xin1", de tamanho Nj, vai armazenar a única linha resultante do processo.

  //Para cada coluna j de image1r:

       for(j=0;j<Nj;j++)
          {

          // Vamos agora, calcular a média inicial (med) para a coluna j:

          med = 0;
          for(i=0;i<Ni;i++)
             {
                med = med + ((float) image1r[i*Nj+j]);
             }
          med = med/((float)Ni);
```

Fig. II.11-. Quadro com o código em C ++ , para o cálculo da 1ª média

```
//Vamos agora calcular uma nova média (nmed), rejeitando os valores muito afastados da média
          //inicial. Estamos considerando um            // precisa ser "calibrado" experimentalmente).

          // A variável "cont" vai armazenar a quantidade de valores válidos na coluna j.
          lim = 0,4*med;
          nmed = 0;
          cont = 0;
          for(i=0;i<Ni;i++)
             {
               if (fabs(((float)image1r[i*Nj+j]) - med) <= lim)
                 {
                   nmed = nmed + ((float) image1r[i*Nj+j]);
                   cont = cont + 1;
                 }
             }
           if (cont != 0)
           {
             nmed = nmed/cont;
           }
           else
           {
           // Nesse caso, os valores estão muito dispersos. Na falta de melhor opção vamos
           // adotar a média inicial. Outra opção, seria adotar, por exemplo, o primeiro valor da série (i=0),
           // ou seja: nmed = (float)image1r[j];
              nmed = med;
           }
           // enfim, armazenando o valor processado na posiçao j do vetor xin1
           xin1[j] = (int)nmed;
        }
    // Provavelmente, agora, o vetor xin1, que guarda a linha processada, vai ser armazenado na matriz
de saída.
```

Fig. II.12- Quadro com o código em C ++ , para o cálculo da 2ª média.

II.4.2.1- Filtro passa-alta

O filtro passa-alta usado neste trabalho, tem o objetivo de realçar detalhes finos e de contornos dos caracteres da imagem, não alterando as regiões homogeneas. O filtro emprega a máscara da figura II.13, que deve ser aplicada em todo o conjunto de *pixels* da matriz geradora da imagem. [8]

A ativação do filtro foi condicionada ao comando de um botão no módulo (SW4), a fim de entrar em operação somente quando for desejado.

$$\frac{1}{8} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & 16 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Fig. II.13- Máscara 3x3 do filtro passa-alta utilizado

As figuras II.14 e II.15 apresentam os quadros com o código em C++, para a aplicação do filtro passa alta.

```
// O trecho a seguir faz a convolução de uma imagem em tons de cinza (0 -255) com um filtro passa-alta
// Kernel do filtro:
//                    -1   -1  -1
//        1/8 x       -1  16   -1
//                    -1   -1  -1
// As variáveis Ni e Nj representam  o número de linhas e o número de colunas da imagem.
// O ponteiro image1r  manipula a imagem de entrada .
// O ponteiro xin1 recebe a imagem já filtrada
// c1,c2,c3,c4,c5,c6,c7,c8 e c9 são variáveis  do tipo float
  for(i=0;i<Ni;i++)
     for(j=0;j<Nj;j++)
       {
        if(i==0 || i==Ni-1 || j==0 || j==Nj-1)
        {
          xin1[i*Nj+j] = image1r[i*Nj+j];
        }
```

Fig.II.14- Quadro com a parte 1 do código em C++ para o filtro passa alta.

```
else
      {
        c1 = (float)image1r[(i-1)*Nj+(j-1)];
        c2 = (float)image1r[(i-1)*Nj+(j)];
        c3 = (float)image1r[(i-1)*Nj+(j+1)];
        c4 = (float)image1r[(i)*Nj+(j-1)];
        c5 = (float)image1r[(i)*Nj+(j)];
        c6 = (float)image1r[(i)*Nj+(j+1)];
        c7 = (float)image1r[(i+1)*Nj+(j-1)];
        c8 = (float)image1r[(i+1)*Nj+(j)];
        c9 = (float)image1r[(i+1)*Nj+(j+1)];

        // a linha a seguir filtra a imagem com o filtro de Sobel  (destaca os contornos)
        //val = fabs(((c7+2*c8+c9)-(c1+2*c2+c3)))+ fabs(((c3+2*c8+c9)-(c1+2*c4+c7)));
        val = ((-(c1+c2+c3+c4+c6+c7+c8+c9)+(16*c5))/8);
        xin1[i*Nj+j] = (int)val;
        if((int) xin1[i*Nj+j] > 255)  xin1[i*Nj+j] = 255;
        if((int) xin1[i*Nj+j] < 0)     xin1[i*Nj+j] = 0;
       }
     } /* do for i e j */
.
```

Fig.II.15- Quadro com a parte 2 do código em C++ para o filtro passa alta.

III- Testes e resultados

III.1 Testes do circuito codificador de sinal composto de vídeo.

A figura III.1 é a impressão da tela do osciloscópio, com a forma de onda dos pulsos de eco demodulados, ou seja, as envoltórias dos mesmos. Este sinal é proveniente do amplificador/ demodulador (desenvolvido na dissertação de mestrado de Paulo Ernesto Moreira, CEFET-RJ, novembro/2007), ligado na sonda transdutora e é aplicado na entrada do circuito codificador de sinal composto de vídeo. A amplitude é ajustada no ganho do amplificador para 700mV, no máximo, compatibilizando com o nível exigido na entrada do codificador.

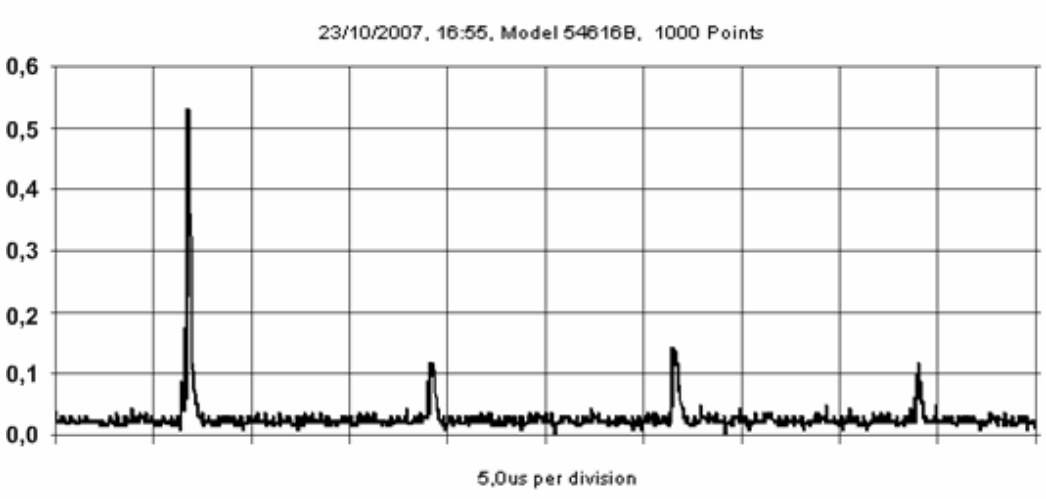


Fig. III.1- Pulso eco demodulado

Na figura III.2, temos a foto da placa, montada artezanalmente, com o circuito codificador do sinal composto de vídeo.

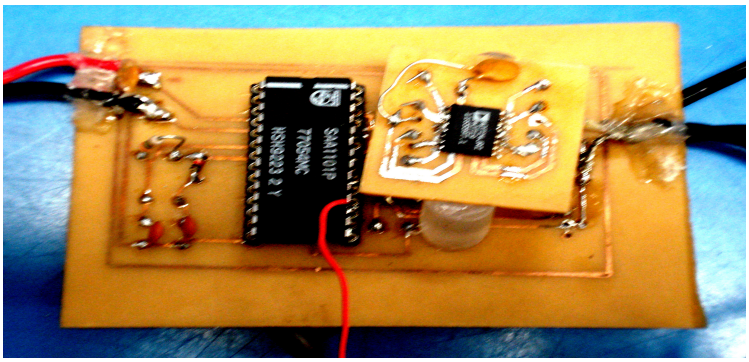


Fig. III.2- Placa com a montagem do circuito codificador de sinal composto de vídeo

Na saída do cicuito codificador, se obtem o sinal composto de vídeo, com as envoltóias do ecos constituindo as informações de luminância das linhas da imagem, como ilustra a figura III.3. A amplitude atinge 900mVpp para o maior eco.

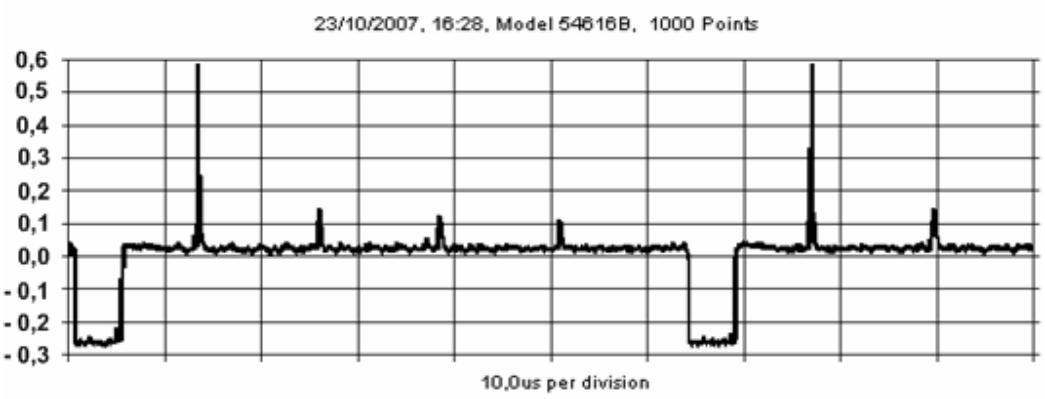


Fig. III.3- Sinal composto de vídeo com as envoltórias dos ecos nas linhas

III.2 Testes com o módulo de processamento digital.

Na figura III.4 é apresentada a foto do módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices***, empregado no processamento digital do sinal composto de vídeo.

.

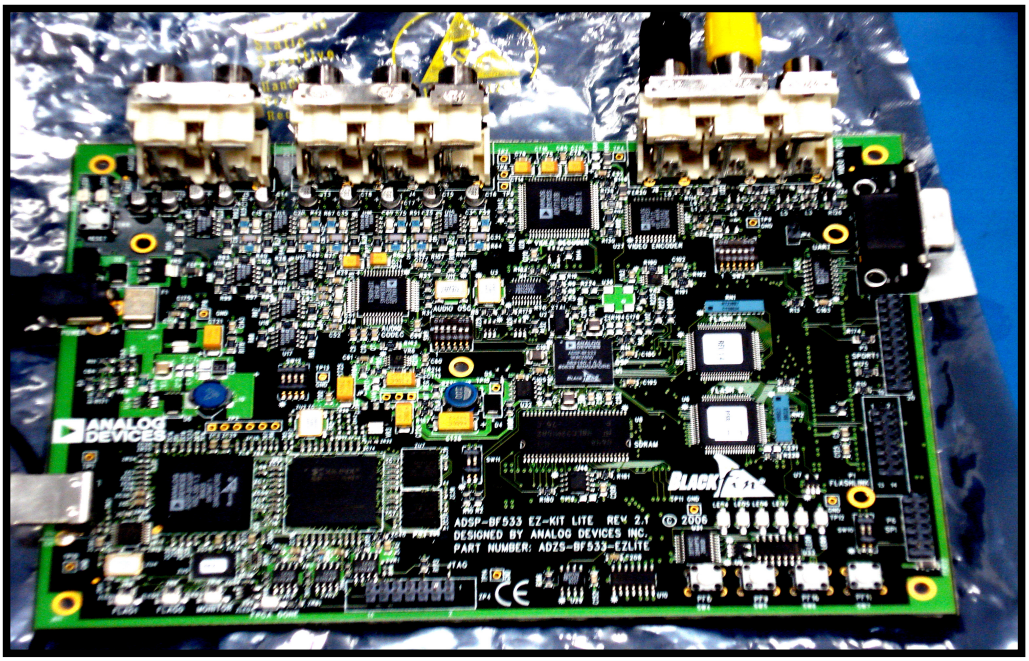


Fig. III.4- Módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices***

O ambiente de programação (***VisualDSP++***) empregado no módulo, é ilustrado na figura III.5. No apêndice, está descrito todo o código comentado, em linguagem C++, desenvolvido com o apoio das bibliotecas dos arquivos fornecidos pela ***Analog Devices*** [14]**.** O programa configura os conversores A/D e D/A para operarem no padrão M, com sistema NTSC e padrão de digitalização ITU-656. A captura de um novo quadro, foi programada para ficar condicionada ao comando do botão SW4, onde foi adaptada a entrada de pulsos da varredura lateral do transdutor, de forma que, a cada passo lateral ocorre uma nova aquisição e a formação de uma nova coluna na imagem. O programa também contempla: o cálculo das médias das linhas, a conversão das linhas em colunas, a repetição das colunas quadro a quadro e o filtro passa alta, sendo esse último condicionado ao comando do botão SW5, que o ativa quantas vezes se desejar, com sucessivos toques.

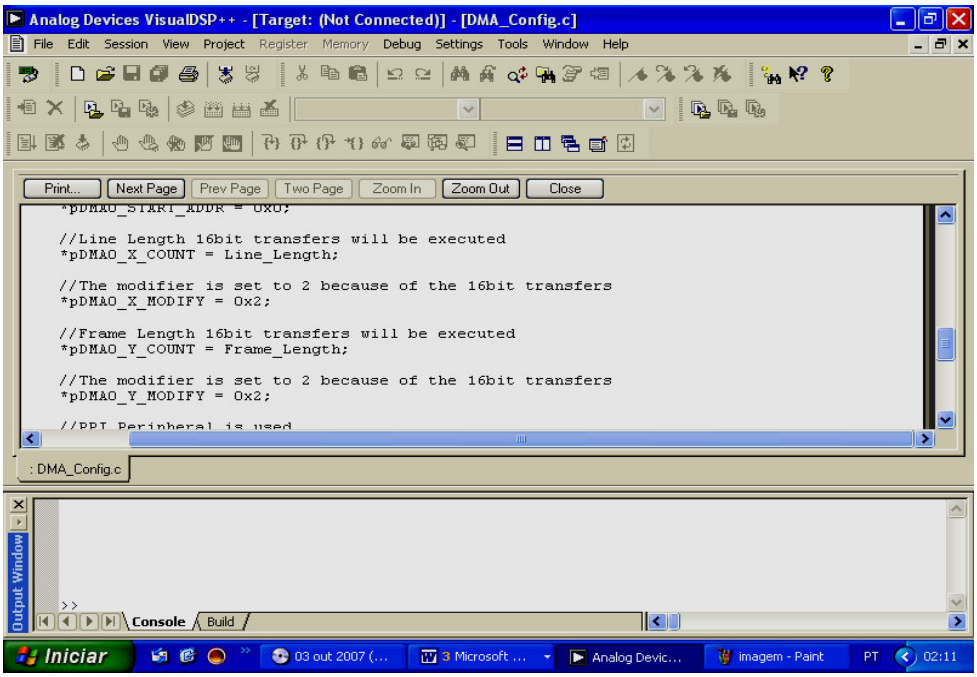


Fig. III.5- Ambiente de desenvolvimento (***VisualDSP++*)** aplicado na programação do módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices*** [14]

A figura III.6 mostra a foto do corpo de prova utilizado para verificar a qualidade da imagem. O experimento foi feito com um cilindro de ferro com 55mm de diâmetro e 60mm de altura. Sobre o cilindro foi colocado um retângulo de acrílico, medindo 30mm X 10mm e 1mm de espessura. Sobre o acrílico, foram apoiados 3 pregos com 15mm de comprimento e 1mm de diâmetro, em disposição transversal, espaçados de 5mm entre si. O conjunto se encontra imerso em água, no interior de uma bacia de vidro com 300mm de diâmetro e 150mm de profundidade. O transdutor desenvolve uma trajetória transversal aos pregos e portanto,

diametral ao cilindro de ferro, percorrendo 72mm e retornando a posição inicial. Em relação ao fundo da bacia, a lente do transdutor se encontra a 90mm de altura.

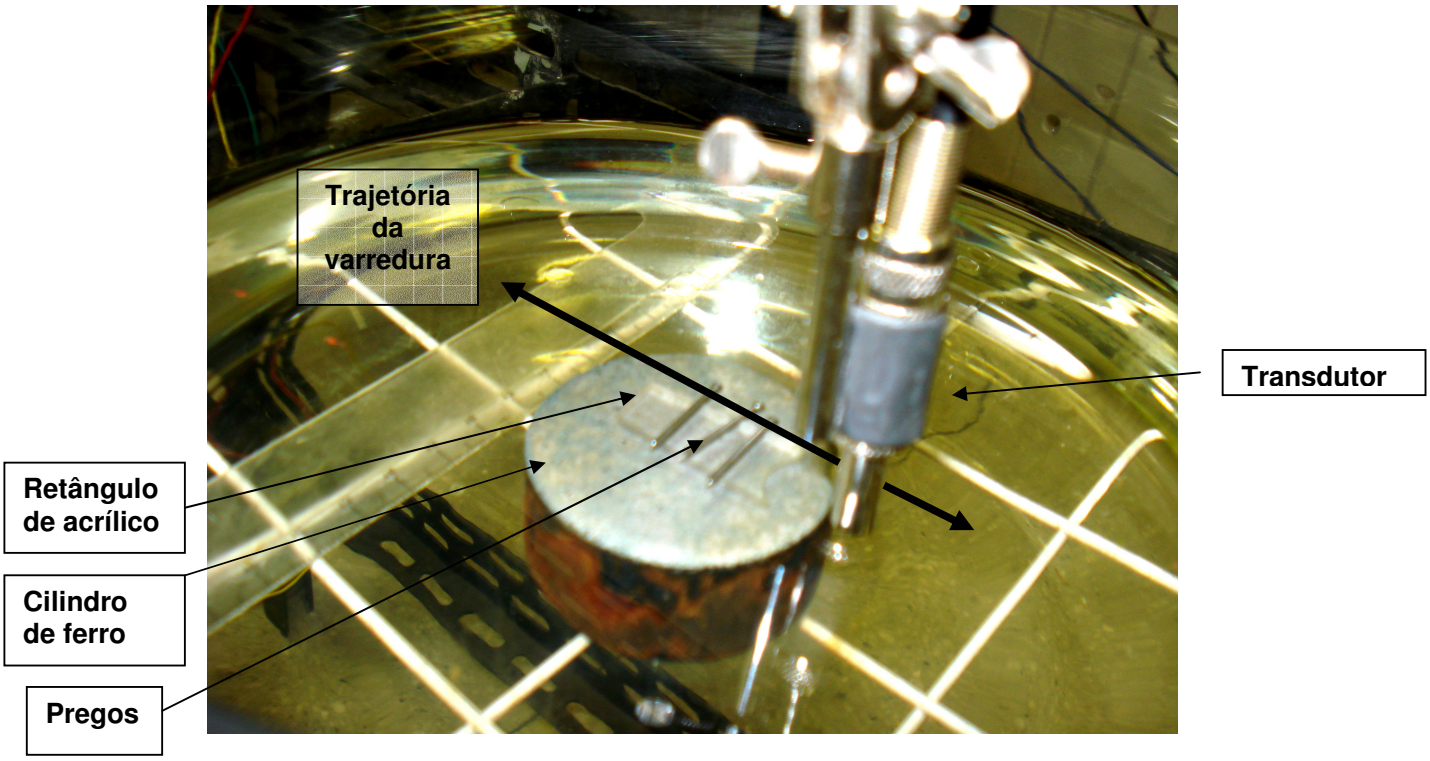


Fig. III.6- Corpo de prova explorado

A figura III.7 apresenta a imagem obtida com a exploração do corpo da figura III.6.

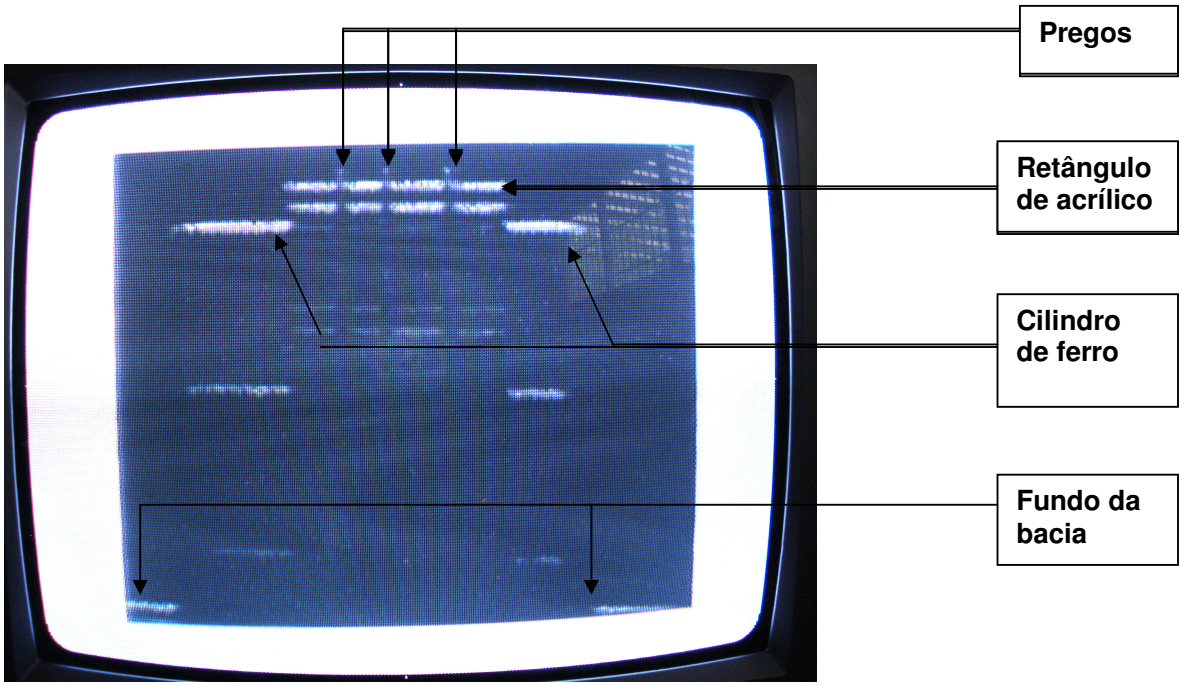


Fig. III.7- Imagem do corpo de prova explorado

A figura III.8 apresenta a imagem da figura III.7, com a ativação do filtro passa alta, onde revela o realce dos detalhes. Os pontos referentes aos 3 pregos se tornam mais nítidos.

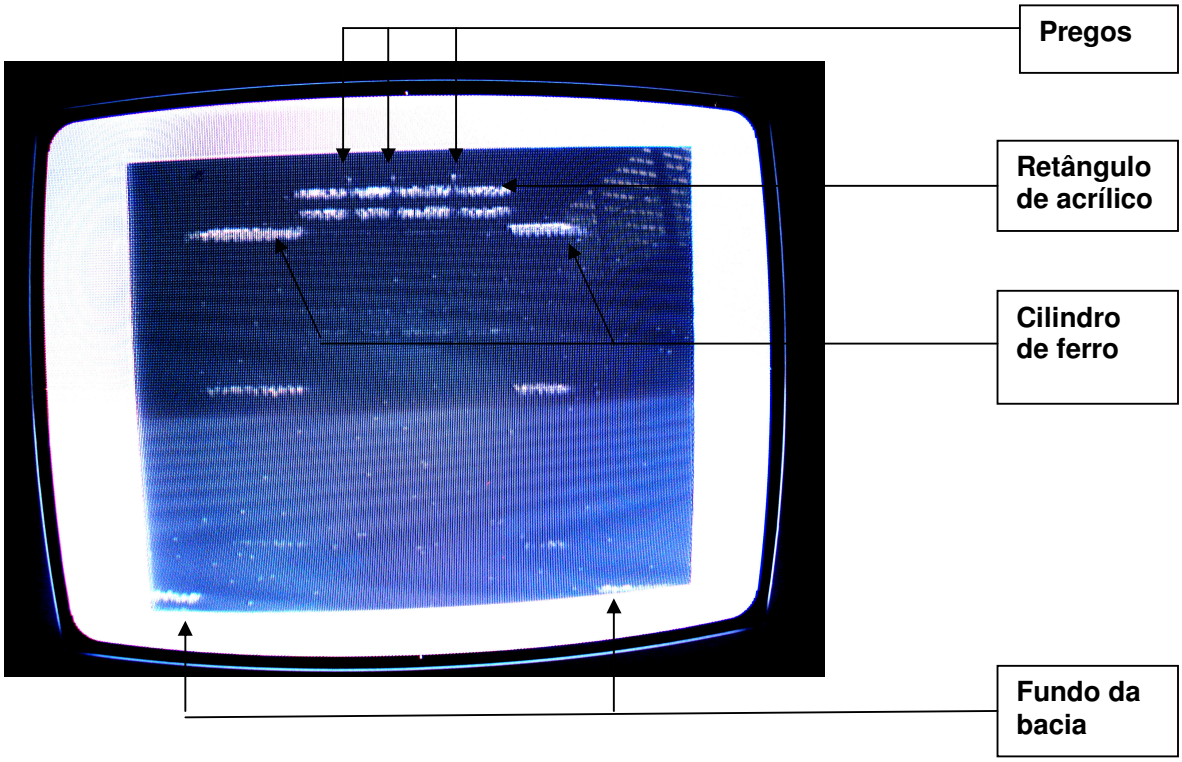


Fig. III.8- Imagem do corpo de prova com a ativação do filtro passa alta

III.3 Programa de varredura do transdutor.

A imagem gerada do corpo de prova do item anterior, foi efetuada com a varredura lateral do transdutor, promovida com o auxílio de uma mesa com 4 eixos (1, 3, 5 e 7) de movimentos programáveis, como mostra a figura III.9. Nesta aplicação, somente o eixo 1 é utilizado.

O programa é desenvolvido no ambiente do *software* ***SiNet Hub Programmer V1.26*** de propriedade do fabricante da mesa [15]. O ambiente de desenvolvimento é apresentado na figura III.10, com o programa elaborado para este experimento. As linhas de programação, traduzem o seguinte:

- O motor do eixo 1 deverá girar 1200 passos em sentido horário, o que corresponde a um deslocamento do transdutor de 0,3mm para a. (Instrução da linha 2 do programa).

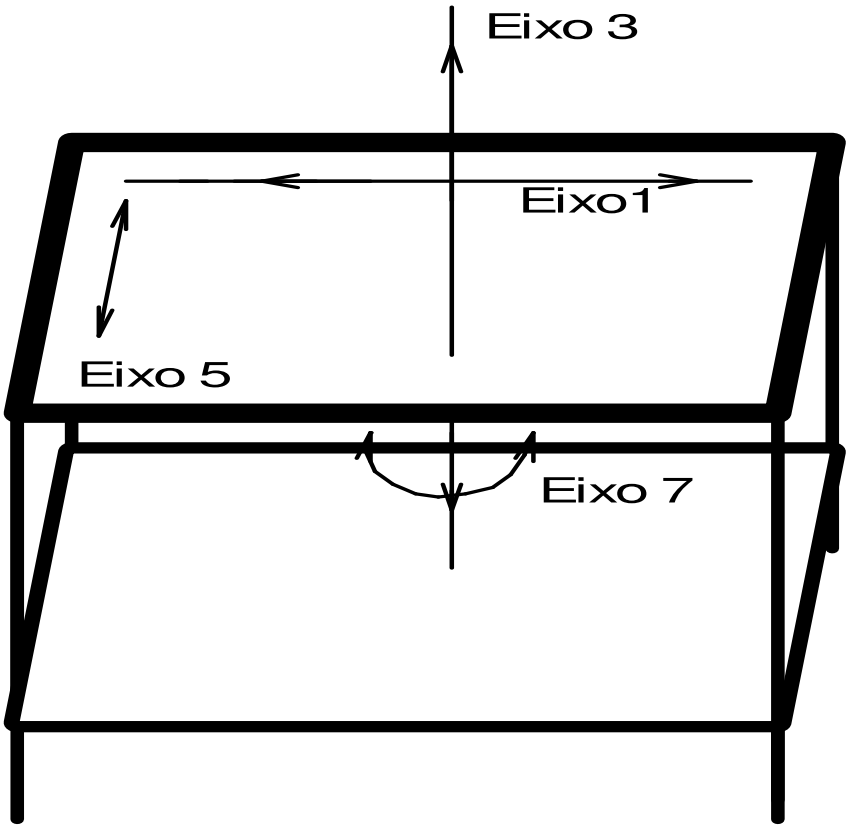


Fig. III.9- Mesa com movimentos programáveis

- Em seguida, o motor deve ficar parado 1 segundo, para o transdutor entrar em repouso. (Instrução da linha 3 do programa).

- Na seqüência, um pulso deverá ser gerado para comandar o módulo de processamento digital, a fim de executar uma nova captura e montar mais uma coluna na imagem, correspondendo a visão axial do transdutor dessa nova posição. (Instrução da linha 4 do programa).

- Esses procedimentos deverão ser repetidos 240 vezes, para que o transdutor faça a leitura desse número de posições, ao longo da varredura lateral. (Instruções das linhas 5 e 1 do programa).

- Após a exploração total , o motor do eixo 1 deverá executar 288.000 passos em sentido anti-horário, de modo a trazer o transdutor de volta a posição inicial. (Instrução da linha 6 do programa).

- Na posição inicial, o transdutor deve ficar parado 30 segundos, para qualquer alteração no corpo de prova. (Instrução da linha 7 do programa).

- Todo o ciclo deverá ser refeito. (Instrução da linha 8 do programa).

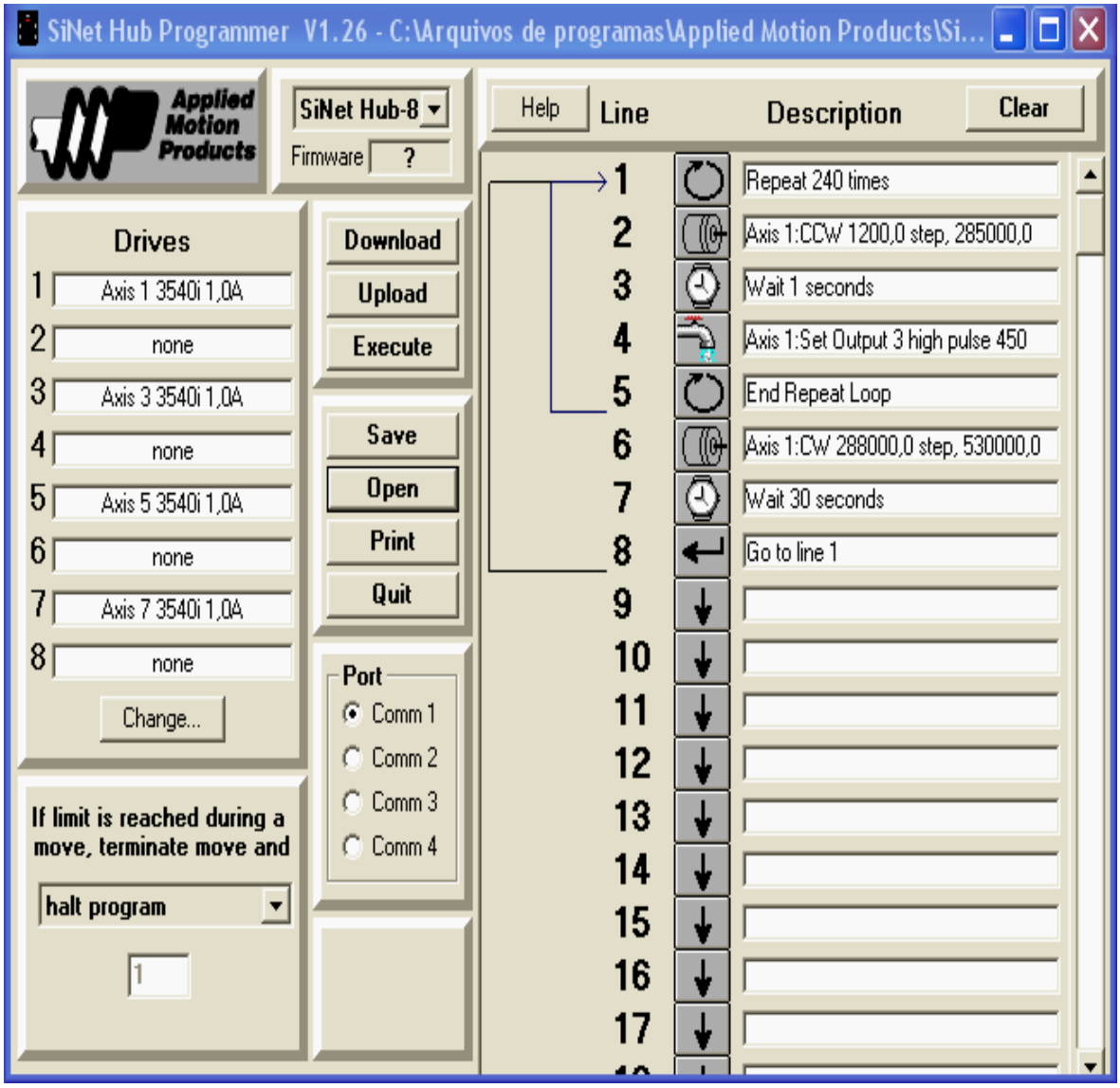


Fig. III.10- Ambiente de desenvolvimento do *software* ***SiNet Hub Programmer V1.26.***

Discussão e conclusão

A idéia de produzir um sistema eletrônico de processamento, para a geração de imagem por ultra-som, teve como parte do objetivo, que o sinal resultante fosse compatível com receptores de TV analógicos e digitais. Isso conduziu a pesquisa para a necessidade de se gerar um sinal composto de vídeo, com os sinais dos ecos na informação de luminância, além dos sinais de controle ou sincronismo necessários. Daí então, circuitos integrados empregados nos sistemas que geram sinais de vídeo para TV (câmeras, geradores de barras, DVDs) , passaram a ser analisados para a elaboração do circuito codificador de sinal composto de vídeo. O protótipo do circuito desenvolvido, utilizou cristais com fabricação dedicada, o que demandou cerca de 4 semanas para a confecção. Os circuitos integrados empregados foram importados, com tempo médio de 8 semanas para a aquisição. Superada essa etapa, os testes iniciais revelaram a presença de um ruído em fase com o pulso de sincronismo horizontal, que após cuidadosa análise, se determinou ser decorrente do pulso de excitação do transdutor, ou seja, uma pequena amostra desse pulso estava sendo amplificada pelo amplificador da sonda (desenvolvido na dissertação de mestrado de Paulo Ernesto Moreira, CEFET-RJ, novembro/2007). Tal inconveniente foi resolvido, utilizando o pulso de apagamento horizontal, fornecido pelo gerador de sincronismo, como comando de *muting* para o amplificador da sonda e uma fonte desacoplada dos demais módulos, assim como, cabos blindados nas inter-conexões. Mediante as correções citadas, o sinal composto de vídeo produzido, apresentou grande estabilidade e alta relação sinal-ruído, que embora não medida, se mostrou eficaz quando o sinal foi testado em um receptor de TV convencional.

O sinal de vídeo gerado, possui a leitura axial do transdutor nas informações das linhas, que seria reproduzida no sentido horizontal da imagem, ou seja, as linhas teriam que ser convertidas em colunas, para ajustar a exploração axial ao sentido vertical da imagem. Além do que, enquanto o transdutor não se deslocasse, as colunas já formadas, das posições lidas, deveriam ser repetidas nos sucessivos quadros. Diante desses fatos, surgiu a necessidade do estudo do processamento digital, com a busca de um módulo que sintetizasse os circuitos para tal fim. Outra característica importante é que o módulo, uma vez programado, operasse de forma autônoma, para atender a outra parte da proposta, que era produzir um sistema portátil, sem vínculo permanente com um computador. Possuindo tais quesitos, o módulo da ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices,*** foi selecionado e se mostrou quase que totalmente eficaz na aplicação dada. Infelizmente o tempo de processamento da captura de um quadro atinge cerca de 120ms, gerando um efeito de cintilação (*flicker)*, na mudança de posição lateral do transdutor. Um dispositivo adicional, para a aceleração do *clock* de processamento, é capaz promover a captura em 16,6ms, evitando esse inconveniente. Outra solução seria a utilização de um módulo similar, porém com processamento *duplex,* onde a entrada e a saída

podem ser tratadas simultaneamente. Dessa forma, o último quadro ficaria sendo repetido na saída, enquanto uma nova captura acontecesse na entrada. Tais dispositivos não foram implementados, mediante ao fato do efeito da cintilação ser razoavelmente tolerável. Nos demais aspectos, o desempenho do módulo foi satisfatório, pois a resolução da imagem alcançou os valores previstos.

A figura IV.1 é a foto do circuito codificador de vídeo, interligado com o módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices.***

Quanto a programação do módulo, foi desenvolvida no ambiente do *software* ***VSUAL DSP++*** e necessitou de muito estudo, da linguagem C++ e das bibliotecas de arquivos da ***Analog Devices*** . Vários meses de dedicação, com sucessivos testes, foram necessários para atingir o programa (descrito no apêndice), que executasse o processamento desejado.

Enfim, os objetivos foram alcançados, os sinais de vídeo analógico e digital, foram gerados, por módulos portáteis e a imagem do corpo explorado pode ser visualizada nos *laptops* e/ou nos receptores de TV , com a definição pretendida, ou seja, 300µm de resolução lateral e axial. O sistema desenvolvido pode vir a se tornar uma opção para aumentar a versatilidade e a mobilidade dos sistemas atuais.

Desenvolvimentos futuros podem tornar os circuitos ainda mais compactos, elimnar totalmente o *flicker*, produzir efeitos coloridos nas diversas camadas de profundidade e introduzir processos de medição dos elementos explorados, como os sugeridos no apêndice 4.

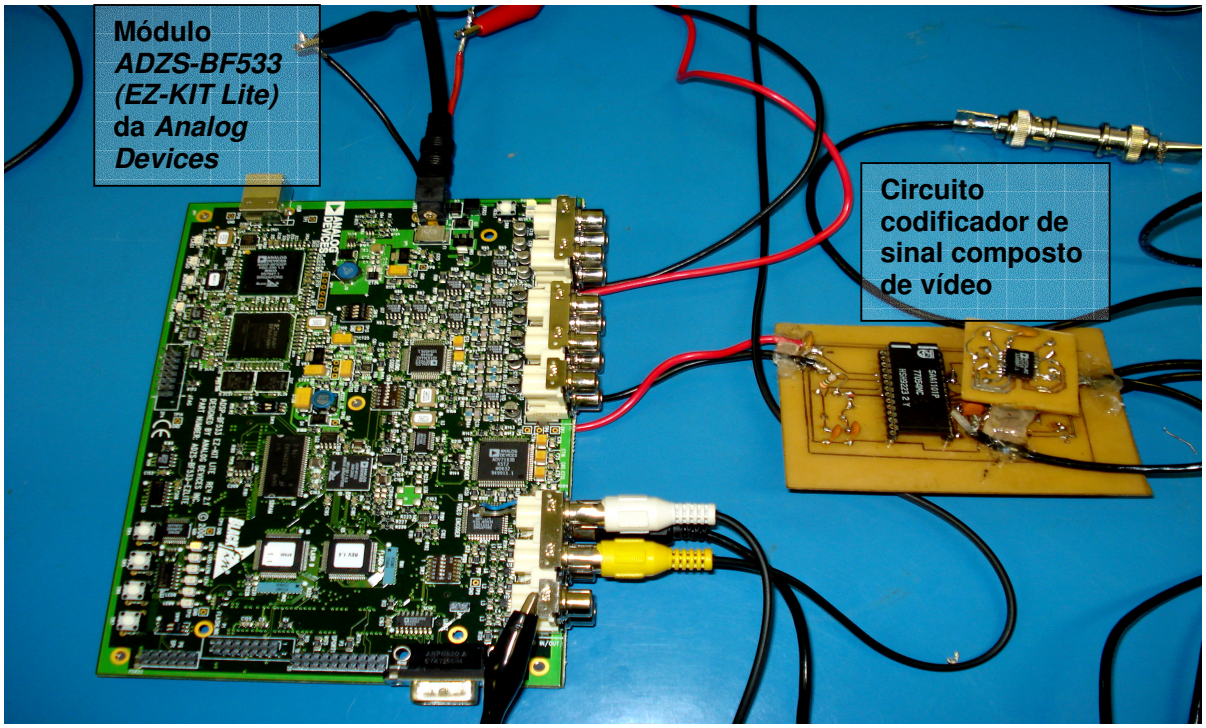


Fig.IV.1- Circuito codificador de vídeo interligado com o módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices.***

Referências Bibliográficas.

[1] Santin, Jorge L. – "Ultra-Som, Técnica e Aplicação" – Artes Gráficas e Editora Unificado, Curitiba – PR – 2003 , 2ª ed.

[2] Lethiecq, Marc et al. – "Principles and Applications of High-Frequency Medical Imaging" - *Advances in Acoustic Microscopy* **–** Volume 2 chapter 2, pp. 39–51;56–69, – ed. by Andrew Briggs and Walter Arnoud - Plenum Press, New York – 1996

[3] Foster, F. Stuart et al. – "A History of Medical and Biological Imaging with Polyvinylidene Fluoride (PVDF) Transducers" – *IEEE Transactions on Ultrasonics, Ferreletrics, and Frequency Control*, vol. 47, N° 6, November 2000.

[4] Hunt, John W. – "Ultrasound Transducers for Pulse-Echo Medical Imaging" – *IEEE Transactions on Biomedical Engineering*, vol. BME-30, N° 8, August 1983.

[5] Yong, Paul H.- " Técnicas de Comunicação Eletrônica "- Pearson Prentice Hall, São Paulo – 2006 – 5ª ed.

[6] Gonzales, Rafael C. & Woods, Richard E. – " Digital Image Processing "- Pearson Prentice Hall, São Paulo – 2002, 2ª ed.

[7] Torreão, J.R.A. & Fernandes, J.L. "Single-image shape from defocus", Proceedings of Sibgrapi 2005, pp. 241-246, *IEEE Computer Society*, October 2005.

[8] Ethan Frome – http://www.materiais.ufsc.br/Diciplinas/EMC5793/textos/PreProc&bin.doc .Acessado em 21 de julho de 2007.

[9] Disponível em: A brief Introduction to Digital Video - http://www.spacewire.co.uk/video_standard.html . Acessado em 20 de julho 2007.

[10] Charles Poynton - Video engineering - http://www.poynton.com/Poynton-video-eng.html Acessado em 20 de julho de 2007

[11] Bastos, Arilson & Fernades, Sergio L. – "Televisão Profissional"- Antenna Edições [Técnicas LTDA – Rio de Janeiro, 2004, 2ª ed.

[12] ***Datasheet*** do **IC SAA 1101** da **Philips**
http://www.datasheetcatalog.net/pt/datasheets_pdf/S/A/A/1/SAA1101.shtml
Acessado em 13 de agosto de 2006.

[13] ***Datasheets*** obtidos no *site* do fabricante ***Analog Devices***
Conversor D/A de vídeo.
http://www.analog.com/UploadedFiles/Data_Sheets/ADV7170_7171.pdf
Conversor A/D de vídeo .
http://www.analog.com/UploadedFiles/Data_Sheets/ADV7183B.pdf
Processador ADSP-BF533 .
http://www.analog.com/UploadedFiles/Data_Sheets/ADSP-BF531_BF532_BF533.pdf
Codificador de vídeo
http://www.analog.com/en/prod/0,2877,AD724,00.html
Acessado em 13 de agosto de 2006.

[14] Manuais obtidos no *site* do fabricante ***Analog Devices***
http://www.analog.com/processors/blackfin/technicalLibrary/manuals/index.html
ADSP-BF53x-BF56x Blackfin® Processor Programming Reference .pdf
ADSP-BF533 Blackfin® Processor Hardware Reference.pdf
Getting StartedWith Blackfin® Processors.pdf
ADSP-BF533 EZ-KIT Lite Manual Rev. 2.1.pdf
Getting Started with ADSP-BF537 EZ-KIT Lite.pdf
ADSP-BF531-BF532-BF533_b.pdf
Acessado em 13 de agosto de 2006.

[15] Valente, Gabriel O. – "Construção e Avaliação Experimental de um Hidrofone Tipo Agulha" Tese de Mestrado – CEFET, Agosto 2006.

Apêndice 1

Código comentado, em linguagem C++, referente a todo o processamento do sinal no módulo ***ADZS-BF533 (EZ-KIT Lite)*** da ***Analog Devices,*** desenvolvido no ambiente do *software* ***VSUAL DSP++,*** com o auxílio das bibliotecas de arquivos da ***Analog Devices.*** [14]

```
MACROS ************************************************************************/

#define u64         unsigned long long

#define USE_CIRCULAR_BUFFER
//#define PAL_FRAME

//#define MYTEST
#define BLACK               false
#define COLOUR              true

#define EZ_BUTTON_SW5 1
#define EZ_BUTTON_SW6 2
#define FILTER_FACTOR   16

/*********************************************************************

Include files

*********************************************************************/
#include <drivers/adi_dev.h>                                        // Device manager
includes
#include <drivers/decoder/adi_adv7183.h>            // AD7183 device driver includes
#include <drivers/encoder/adi_adv717x.h>            // ADV7171 device driver includes
#include <drivers/twi/adi_twi.h>                            // TWI device driver includes
#include <SDK-ezkitutilities.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include <cdefBF533.h>

/*****************************************************************************

Static data

*****************************************************************************/
#define   PAL               1
#define NTSC      0

#if defined(PAL_FRAME)
#define FRAME_DATA_LEN          1728            // Data length per line for PAL video
format
```

```
#define NUM_LINES                   625                         // Number of lines per frame
#define ACTIVE_FIELD1   22                          // Line number of first active field1
#define ACTIVE_FIELD2   335                         // Line number of first active field2
#define HORIZONTAL_BLANKING 72                      // Width (32bits) of horizontal blanking
#define NUM_LINES_ACTIVE        288                 // Number of active lines per frame

// more defines
#define BOTTOM_VERTICAL_BLANKING 2

#else   // its NTSC Frame
#define FRAME_DATA_LEN          1716                // Data length per line for NTSC format
#define NUM_LINES                   525                         // Number of lines per frame
#define ACTIVE_FIELD1   19                          // Line number of first active field1
#define ACTIVE_FIELD2   282                         // Line number of first active field2
#define HORIZONTAL_BLANKING 69                      // Width (32bits) of horizontal blanking
#define NUM_LINES_ACTIVE 240                // Number of active lines per frame

// more defines
#define BOTTOM_VERTICAL_BLANKING 3

#endif

#define FRAME_SIZE        FRAME_DATA_LEN*NUM_LINES
#define FRAME_DATA_LEN_32BIT FRAME_DATA_LEN/4

#define NUM_COLS_ACTIVE 360

// Define the DMA buffer for input video frame.
section("sdram0") static char sFrame0[FRAME_SIZE];

// Define the DMA buffer for output video frame.
section("sdram0") static char sFrame1[FRAME_SIZE];

// Define the image buffer for input video frame
// to apply brightness filter
section("sdram0") static u32 sFilterIn[NUM_LINES_ACTIVE*2*NUM_COLS_ACTIVE];

// Define the image buffer for output video frame.
// to apply brightness filter
section("sdram0") static u32 sFilterOut[NUM_LINES_ACTIVE*2*NUM_COLS_ACTIVE];

#ifdef USE_CIRCULAR_BUFFER
// buffer for input/output.
static ADI_DEV_CIRCULAR_BUFFER CircBuffer;
static ADI_DEV_CIRCULAR_BUFFER CircBufferOut;
#else
        ADI_DEV_2D_BUFFER Buffer2D;
        ADI_DEV_2D_BUFFER Buffer2DOut;
#endif

/*****************************************************************************

typedefs

*****************************************************************************/
```

```
typedef struct lf {
        u32 eav;
        u32 blankVideo[
#if defined(PAL_FRAME)
        70
#else
                67
#endif
        ];
        u32 sav;
        u32 activeVideo[360]; } lframe;

/*********************************************************************

memory for initialization of system services and device manager

*********************************************************************/
// storage for interrupt manager
static u8 IntMgrData[(ADI_INT_SECONDARY_MEMORY * 0)];

// DMA Manager data (base memory + memory for 1 DMA channel (PPI0)
static u8 DMAMgrData[ADI_DMA_BASE_MEMORY + (ADI_DMA_CHANNEL_MEMORY * 1)];
static ADI_DMA_MANAGER_HANDLE DMAManagerHandle;                 // DMAMgr handle

// Device Manager data (base memory + memory for 3 devices (1xPPI,TWI,AD7183 or
AD7171))
static u8 DevMgrData[ADI_DEV_BASE_MEMORY + (ADI_DEV_DEVICE_MEMORY * 5)];
static ADI_DEV_MANAGER_HANDLE DeviceManagerHandle;// DevManager handle

// storage for critical region
static ADI_INT_CRITICAL_REGION_DATA     CriticalRegionData;

/*********************************************************************

handles to device drivers

*********************************************************************/
ADI_DEV_DEVICE_HANDLE AD7183DriverHandle;       // handle to the ad7183 driver
ADI_DEV_DEVICE_HANDLE EncoderDriverHandle;      // handle to the ad7171 driver

/*********************************************************************

protótipos de funções

*********************************************************************/
static void InitSystemServices(void);

static void SetEncoderMode(ADI_DEV_DEVICE_HANDLE EncoderDevHandle, u32 mode);
static void Read7183StatusReg(ADI_DEV_DEVICE_HANDLE AD7183DevHandler);
static void ReadEncoderRegs(ADI_DEV_DEVICE_HANDLE EncoderDevHandler);

static void DisplayVideoFrame(
        ADI_DEV_MANAGER_HANDLE      DeviceManagerHandle,
        ADI_DMA_MANAGER_HANDLE      DMAManagerHandle );

static void CaptureVideoFrame(
```

```
            ADI_DEV_MANAGER_HANDLE      DeviceManagerHandle,
            ADI_DMA_MANAGER_HANDLE      DMAManagerHandle );

static void PrepareBuffer(void);

static ADI_INT_HANDLER(ExceptionHandler); // exception handler
static ADI_INT_HANDLER(HWErrorHandler);               // hardware error handler
static void ClientCallback(void *AppHandle,u32  Event,void *pArg);


static void CopyField2ToField1(void);

volatile bool FrameReady = FALSE;

/*****************************/
// Variáveis Novas
/*****************************/

int         currentCol;
bool        flgDirection = true;

/*****************************/
// Novas definições de funções
/*****************************/

static void InitiatesBuffer(char *, bool);

static void RotateFrame_90G(void);

static void RotateLine_90G(void);

static void Init_PLL_DIV(u32 value);

static void ApplyFilterGrayScale(void);

static void FilterGrayScale(void);

static void PopulateImageInBuffer(void);

static void FillsDMABufferOut(void);

static void Interpolation(void);

void        my_ezEnableVideoDecoder             (void);                 // my function to enable the
7171 video decoder

// Pseudo TWI Configuration
#if defined(__ADSP_TETON__)
// Pseudo TWI will be used to access ADV7183 and ADV7179 registers.
// BF561 Ports (PF0=SCL,PF1=SDA) & Timer(Timer 3) used for Pseudo TWI
adi_twi_pseudo_port TWIPseudo=
{ADI_FLAG_PF0,ADI_FLAG_PF1,ADI_TMR_GP_TIMER_3,(ADI_INT_PERIPHERAL_ID)NULL
};
#else
// Pseudo TWI will be used to access ADV7183 and ADV7171 registers.
// BF533 Ports (PF0=SCL,PF1=SDA) & Timer(Timer 1) used for Pseudo TWI
```

```
adi_twi_pseudo_port TWIPseudo=
{ADI_FLAG_PF0,ADI_FLAG_PF1,ADI_TMR_GP_TIMER_1,(ADI_INT_PERIPHERAL_ID)NULL
};
#endif
        ADI_DEV_CMD_VALUE_PAIR PseudoTWIConfig[]={
        {ADI_TWI_CMD_SET_PSEUDO,(void *)(&TWIPseudo)},
        {ADI_DEV_CMD_SET_DATAFLOW_METHOD,(void
*)ADI_DEV_MODE_SEQ_CHAINED},
        {ADI_DEV_CMD_SET_DATAFLOW,(void *)TRUE},
        {ADI_DEV_CMD_END,NULL}
        };



/*************************************************************************
*
*       Function:main
*       Description:    Atraves do uso de devices drives do AD7183 e do AD7179,
                        este programa demonstra como é feita a captura e a posterior
                        exibição de video.
                        Um frame de video é capturado para a SDRAM pelo decoder
AD7183 e,
                        então, é enviado para a saída de vídeo pelo encoder AD7179.
                        Quando o primeiro botão é pressionado, um novo frame é
capturado.

*************************************************************************/
void main(void)
{


        unsigned int i;


        // Acelerates SCLCK to 90 Mhz setting SSEL to 0x0011
        Init_PLL_DIV(0x000011);


        // initialize the system services
        InitSystemServices();


        // enable all LED's
        for(i=0;i<EZ_NUM_LEDS;i++){
                ezInitLED(i);
        }

        // enable the first push button(SW4)
        ezInitButton(EZ_FIRST_BUTTON);

        // enable the last push button(SW5)
        ezInitButton(EZ_BUTTON_SW5);

        // enable the last push button(SW6)
        ezInitButton(EZ_BUTTON_SW6);
```

```
        // enable the last push button(SW7)
        ezInitButton(EZ_LAST_BUTTON);


        ezTurnOffAllLEDs();

        // prepare input/output buffers
        PrepareBuffer();

   currentCol = 60;

        // fills Vertical and Horizontal Blanking areas of output buffer
        InitiatesBuffer(sFrame1, BLACK);

#ifdef MYTEST

        // fills Vertical and Horizontal Blanking areas of input buffer
        InitiatesBuffer(sFrame0, COLOUR);

#endif

        // and display the first video frame continuosly
        DisplayVideoFrame(DeviceManagerHandle,DMAManagerHandle);

        // loops forever until reset button pressed
   while (1)          {

      // capture a new frame of video if first button(SW4)
      // or SW6 button is push
      if(ezIsButtonPushed(EZ_FIRST_BUTTON) == TRUE ||
           ezIsButtonPushed(EZ_LAST_BUTTON) == TRUE ||
           ezIsButtonPushed(EZ_BUTTON_SW5) == TRUE  ||
           ezIsButtonPushed(EZ_BUTTON_SW6) == TRUE )       {

           // SW6 button pressed, changes the direction of
           // displayed column
           flgDirection = !ezIsButtonPushed(EZ_LAST_BUTTON);

           // stop displaying video
                              ezErrorCheck(adi_dev_Close(EncoderDriverHandle));
                              ezTurnOffLED(0);

                              // first button or SW6 button was pressed
                              if(ezIsButtonPushed(EZ_BUTTON_SW6) == FALSE)    {

                                 // capture a new video frame
                                       CaptureVideoFrame( DeviceManagerHandle,
DMAManagerHandle);

                              }
                              else       {
                                 // SW6 button pressed, apllies filter
                                 // over output video frame
                                       ApplyFilterGrayScale();

                              }
```

```
                        // display the video frame.
                        DisplayVideoFrame( DeviceManagerHandle,DMAManagerHandle);

      }
   }

}



/*********************************************************************

        Function:       InitSystemServices

        Description:    Inicializa os serviços do sistema

*********************************************************************/

static void InitSystemServices(void) {

   u32 i;
   u32 Result;


        ezInit(1);

   // initialize the interrupt manager
   ezErrorCheck( adi_int_Init(NULL,  // pointer to memory for interrupt manager to use
      0,                                    // memory size (in bytes)
      &i,                                   // location where the number of secondary
handlers that
                                            // can be supported will be stored
      NULL));                       // parameter for adi_int_EnterCriticalRegion
                                            // (always NULL for VDK and standalone
systems)

   // hook the exception and hardware error interrupts
        ezErrorCheck( adi_int_CECHook(3, ExceptionHandler, NULL, FALSE));

        ezErrorCheck( adi_int_CECHook(5, HWErrorHandler, NULL, FALSE));


        // initialize the dma manager
        ezErrorCheck( adi_dma_Init(DMAMgrData,      // pointer to memory for the DMA
manager

                        sizeof(DMAMgrData),   // memory size (in bytes)
                        &i,             // location where # of DMA channels is
                                                                    // stored
                        &DMAManagerHandle,    // location where DMA manager handle is
                                                                    //stored
                        NULL));         // parameter for adi_int_EnterCriticalRegion
                                                                    // (always NULL for
VDK and standalone
```

```
                                                                    // systems)


        // initialize device manager
        ezErrorCheck( adi_dev_Init(DevMgrData,      // pointer to memory for the Device
manager to use
                                   sizeof(DevMgrData),                  // memory
size (in bytes)
                                   &i,                          // returns number of
devices DevMgr can support
                                   &DeviceManagerHandle,        // pointer to DevMgr
handle
                                   &CriticalRegionData));       // pointer to critical
region info


}


/*************************************************************************

   Function:      PrepareBuffer
   Description:   Prepara o buffer de entrada de Video para o ADV7183 e
                  o buffer de saída de Video para o ADV7171.
                  Utilizamos a configuração de buffer circular.

*************************************************************************/
static void PrepareBuffer(void){

#if defined(__ADSP_EDINBURGH__)      // the Edinburgh EZ-Kit uses a flash for ports

        // populate the buffers that we'll use for the PPI input and output
#ifdef  USE_CIRCULAR_BUFFER

        CircBuffer.Data = (void*)sFrame0;
        CircBuffer.SubBufferCount = NUM_LINES;
   CircBuffer.SubBufferElementCount = (FRAME_DATA_LEN/2);
        CircBuffer.ElementWidth = 2;
        CircBuffer.CallbackType = ADI_DEV_CIRC_FULL_BUFFER;
        CircBuffer.pAdditionalInfo    = NULL;

        CircBufferOut.Data = (void*)sFrame1;
        CircBufferOut.SubBufferCount = NUM_LINES;
   CircBufferOut.SubBufferElementCount = (FRAME_DATA_LEN/2);
        CircBufferOut.ElementWidth = 2;
        CircBufferOut.CallbackType = ADI_DEV_CIRC_FULL_BUFFER;
        CircBufferOut.pAdditionalInfo          = NULL;

#else

        Buffer2D.Data = (void*)sFrame0;
        Buffer2D.ElementWidth = 2;
        Buffer2D.XCount = (FRAME_DATA_LEN/2);
        Buffer2D.XModify = 2;
        Buffer2D.YCount = NUM_LINES;
        Buffer2D.YModify = 2;
```

```
            Buffer2D.CallbackParameter = &Buffer2D;
            Buffer2D.pNext = NULL;

            Buffer2DOut.Data = (void*)sFrame1;
            Buffer2DOut.ElementWidth = 2;
            Buffer2DOut.XCount = (FRAME_DATA_LEN/2);
            Buffer2DOut.XModify = 2;
            Buffer2DOut.YCount = NUM_LINES;
            Buffer2DOut.YModify = 2;
            Buffer2DOut.CallbackParameter = &Buffer2D;
            Buffer2DOut.pNext = NULL;

#endif

#endif

#if defined(__ADSP_TETON__)

#ifdef      USE_CIRCULAR_BUFFER

            CircBuffer.Data = (void*)sFrame0;
            CircBuffer.SubBufferCount = NUM_LINES;
    CircBuffer.SubBufferElementCount = (FRAME_DATA_LEN/4);
            CircBuffer.ElementWidth = 4;
            CircBuffer.CallbackType = ADI_DEV_CIRC_FULL_BUFFER;
            CircBuffer.pAdditionalInfo    = NULL;

            CircBufferOut.Data = (void*)sFrame1;
            CircBufferOut.SubBufferCount = NUM_LINES;
    CircBufferOut.SubBufferElementCount = (FRAME_DATA_LEN/4);
            CircBufferOut.ElementWidth = 4;
            CircBufferOut.CallbackType = ADI_DEV_CIRC_FULL_BUFFER;
            CircBufferOut.pAdditionalInfo           = NULL;

#else

            Buffer2D.Data = (void*)sFrame0;
            Buffer2D.ElementWidth = 4;
            Buffer2D.XCount = (FRAME_DATA_LEN/4);
            Buffer2D.XModify = 4;
            Buffer2D.YCount = NUM_LINES;
            Buffer2D.YModify = 4;
            Buffer2D.CallbackParameter = &Buffer2D;
            Buffer2D.pNext = NULL;

            Buffer2DOut.Data = (void*)sFrame1;
            Buffer2DOut.ElementWidth = 4;
            Buffer2DOut.XCount = (FRAME_DATA_LEN/4);
            Buffer2DOut.XModify = 4;
            Buffer2DOut.YCount = NUM_LINES;
            Buffer2DOut.YModify = 4;
            Buffer2DOut.CallbackParameter = &Buffer2D;
            Buffer2DOut.pNext = NULL;

#endif
```

```
#endif

}

/*********************************************************************

        Function:       ExceptionHandler
                                HWErrorHandler

        Description:    Não deverá acontecer nunca uma exceção ou um erro
                                de hardware, mas caso ocorra, estes serão
capturados,
                                tratados e, então, todos os LEDS serão acesos.

*********************************************************************/

static ADI_INT_HANDLER(ExceptionHandler)  // exception handler
{
                // turn on all LEDs and wait for help
        ezTurnOnAllLEDs();

          return(ADI_INT_RESULT_PROCESSED);
}

static ADI_INT_HANDLER(HWErrorHandler)          // hardware error handler
{
                // turn on all LEDs and wait for help
        ezTurnOnAllLEDs();

          return(ADI_INT_RESULT_PROCESSED);
}

/*********************************************************************

        Function:       ClientCallback

        Description:    Cada evento de callback tem sua ID única, então,
                                podemos usar a mesma função de callback para
procesar
                                todos os eventos. Sabemos qual o evento que
deu origem
                                à chamada da função através do comando
switch.

                                Observe que pelo modelo de device drivers,
uma chamada de
                                callback ocorre a cada preenchimento do buffer,
                                para isso, ao parâmetro CallbackParameter do
buffer
                                é atribuído um valor não nulo.

*********************************************************************/

static void ClientCallback(
```

```
        void *AppHandle,
        u32  Event,
        void *pArg)
{

        switch (Event)
        {
                // CASE (DMA buffer processed)
        case ADI_DEV_EVENT_BUFFER_PROCESSED:

           if (AppHandle == (void *)0x7183) {
                FrameReady = TRUE;
           }
      break;


        }

}




/*************************************************************************

        Function:       CaptureVideoFrame

        Description:    Esta função é chamada para capturar um frame de vídeo
                        usando o device driver ADV7183.
                        Esta função habilita primeiro o decoder de vídeo AD7183 e
                        aguarda um tempo para sua estabilzação.
                        Depois do processamento do buffer, ou seja, a captura do frame
                        para SDRAM, o driver ADV7183 é fechado e a função retorna para
a
                        rotina chamadora.

*************************************************************************/


static void CaptureVideoFrame(
        ADI_DEV_MANAGER_HANDLE      DeviceManagerHandle,
        ADI_DMA_MANAGER_HANDLE      DMAManagerHandle

){

#ifndef MYTEST

        FrameReady = FALSE;

// turn on last LED
        ezTurnOnLED(5);

        my_ezEnableVideoDecoder();                      // enable the video decoder
(7183)
```

```
#if defined(__ADSP_EDINBURGH__)     // the Edinburgh EZ-Kit uses a flash for ports
//          ezDelay(500);                                       // wait for 7183 to come out of
reset
#endif

/********************** AD7183 driver ************************************************************/
          // open the ad7183 driver
          ezErrorCheck( adi_dev_Open(          DeviceManagerHandle,      // DevMgr handle
                                        &ADIADV7183EntryPoint,          // pdd entry point
                                        0,
                                        (void*)0x7183,              // client handle callback
identifier
                                        &AD7183DriverHandle,                    // DevMgr
handle for this device
                                        ADI_DEV_DIRECTION_INBOUND,              // data
direction for this device
                                        DMAManagerHandle,               // handle to DmaMgr
for this device
                                        NULL,                           // handle to deferred
callback service
                                        ClientCallback));                       // client's
callback function


          // Send Pseudo TWI Configuration table to AD7183 if register configuratian is needed
          ezErrorCheck(adi_dev_Control(AD7183DriverHandle,
                    ADI_AD7183_CMD_SET_TWI_CONFIG_TABLE,(void*)PseudoTWIConfig));

          // do the register read/write here if needed.

          //Read7183StatusReg(AD7183DriverHandle);

          // open the AD7183-PPI device 0 (see Schematic)
          ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                    ADI_AD7183_CMD_OPEN_PPI, (void *)0));

          // command PPI to work in NTSC or PAL mode
#if defined(PAL_FRAME)
                    ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                              ADI_AD7183_CMD_SET_VIDEO_FORMAT, (void *)PAL));
#else
                    ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                              ADI_AD7183_CMD_SET_VIDEO_FORMAT, (void *)NTSC));
#endif

#ifdef    USE_CIRCULAR_BUFFER
          // configure the ad7183 dataflow method to circular
          ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                    ADI_DEV_CMD_SET_DATAFLOW_METHOD,
(void*)ADI_DEV_MODE_CIRCULAR));

                    // give the PPI driver the buffer to process
          ezErrorCheck( adi_dev_Read(AD7183DriverHandle,
                    ADI_DEV_CIRC, (ADI_DEV_BUFFER *)&CircBuffer));

#else
```

```
            // configure the ad7183 dataflow method to chained
            ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                        ADI_DEV_CMD_SET_DATAFLOW_METHOD, (void
*)ADI_DEV_MODE_CHAINED));

                        // give the PPI driver the buffer to process
            ezErrorCheck( adi_dev_Read(AD7183DriverHandle,
                        ADI_DEV_2D, (ADI_DEV_BUFFER *)&Buffer2D));
#endif


   // open PF2 and PF13
   adi_flag_Open(ADI_FLAG_PF2);
   adi_flag_Open(ADI_FLAG_PF13);

   //Blackfin PF2 pin must be set as output
   adi_flag_SetDirection(ADI_FLAG_PF2, ADI_FLAG_DIRECTION_OUTPUT);

   //Set the Blackfin pin PF2 to output enable the ADV7183 data bus
   adi_flag_Clear(ADI_FLAG_PF2);
//   ezDelay(120);
   ezDelay(420);

            /*****************************************************************************/
            // start capturing the input data
            ezErrorCheck( adi_dev_Control(AD7183DriverHandle,
                        ADI_DEV_CMD_SET_DATAFLOW, (void*)TRUE));

            // wait till the buffer has been processed
            while (FrameReady == FALSE) ;

   // set bit PF2 to disable ADV7183 outputs
   adi_flag_Set(ADI_FLAG_PF2);

            // close the PPI driver
            ezErrorCheck(adi_dev_Close(AD7183DriverHandle));

#endif

            // turn off LED
            ezTurnOffLED(5);


            if(ezIsButtonPushed(EZ_BUTTON_SW5) == TRUE)      {
                        Interpolation();
            }

            RotateLine_90G();

            // copies field2 buffer output over field1 buffer
            // output
            CopyField2ToField1();

}
```

```
/*************************************************************************

        Function:       DisplayVideoFrame

        Description: Esta função instancia o driver do encoder para exibir os
                        dados do vídeo armazenado no buffer.
                        O encoder utilizado é o ADV7171

                        Primeiro o driver do encoder é aberto, então o encoder é registrado
                        usando-se o registro para os formatos NTSC ou PAL do serviço de
                        acesso TWI, dependendo se a macro PAL_FRAME foi escolhida
ou não.
                        Quando o fluxo de dados é habilitado, o driver do encoder envia
                        continuamente os dados do buffer de vídeo para a saída.

*************************************************************************/

static void DisplayVideoFrame(
        ADI_DEV_MANAGER_HANDLE      DeviceManagerHandle,
        ADI_DMA_MANAGER_HANDLE      DMAManagerHandle

){


        // turn on first LED
        ezTurnOnLED(0);

        ezEnableVideoEncoder();// enable the video encoder (7171)

        // open the ad7171 driver
        ezErrorCheck( adi_dev_Open(  DeviceManagerHandle,     // Dev manager Handle
#if defined(__ADSP_TETON__)
                                                &ADIADV7179EntryPoint,       // Device Entry
point
#else
                                                &ADIADV7171EntryPoint,       // Device Entry
point
#endif
                        0,                      // Device number
                      NULL,                       // No client handle
                              &EncoderDriverHandle,        // Device manager handle address
                          ADI_DEV_DIRECTION_OUTBOUND,   // Data Direction

                        DMAManagerHandle,           // Handle to DMA Manager
                                NULL,                               // Handle to callback manager
                             ClientCallback));
        /**************** PPI Configuration Settings ****************/
        // Set PPI Device number.
#if defined(__ADSP_TETON__)
        ezErrorCheck(adi_dev_Control( EncoderDriverHandle,
                    ADI_ADV717x_CMD_SET_PPI_DEVICE_NUMBER, (void*)1 ));
#else
        ezErrorCheck(adi_dev_Control( EncoderDriverHandle,
                    ADI_ADV717x_CMD_SET_PPI_DEVICE_NUMBER, (void*)0 ));
```

```
#endif

        // Open PPI Device
        ezErrorCheck(adi_dev_Control( EncoderDriverHandle,
                ADI_ADV717x_CMD_SET_PPI_STATUS, (void*)ADI_ADV717x_PPI_OPEN
));

        // Send Pseudo TWI Configuration table to AD7171 driver
        ezErrorCheck(adi_dev_Control(EncoderDriverHandle,

        ADI_ADV717x_CMD_SET_TWI_CONFIG_TABLE,(void*)PseudoTWIConfig));

        /*************** Read AD7171 registers before configuration **********************/
        //ReadEncoderRegs(EncoderDriverHandle);

   /*****************************************************************************/

#ifdef PAL_FRAME

        SetEncoderMode(EncoderDriverHandle,PAL);

#else
// NTSC video out configuration

   SetEncoderMode(EncoderDriverHandle,NTSC);

#endif

// Read AD7171 register value after configuration

#ifdef  USE_CIRCULAR_BUFFER
        // configure the ad7171 dataflow method
        ezErrorCheck( adi_dev_Control(EncoderDriverHandle,
                ADI_DEV_CMD_SET_DATAFLOW_METHOD,
(void*)ADI_DEV_MODE_CIRCULAR));

        // give the PPI driver the buffer output to process
        ezErrorCheck( adi_dev_Write(EncoderDriverHandle,
                ADI_DEV_CIRC, (ADI_DEV_BUFFER *)&CircBufferOut));

#else
        // configure the ad7171 dataflow method to "chained with loopback"
        ezErrorCheck( adi_dev_Control(EncoderDriverHandle,
                ADI_DEV_CMD_SET_DATAFLOW_METHOD, (void
*)ADI_DEV_MODE_CHAINED_LOOPBACK));

        // give the PPI driver the buffer output to process
        ezErrorCheck( adi_dev_Write(EncoderDriverHandle,
                ADI_DEV_2D, (ADI_DEV_BUFFER *)&Buffer2DOut));

#endif

        // start outputting the output data
        ezErrorCheck( adi_dev_Control(EncoderDriverHandle,
                ADI_DEV_CMD_SET_DATAFLOW, (void*)TRUE));
```

```
}


/*************************************************************************
* Configura o modo do Encoder do EZ-Kit para PAL or NTSC
*************************************************************************/
static void SetEncoderMode(ADI_DEV_DEVICE_HANDLE EncoderDevHandle, u32 mode)
{
   // ADV7171 register configuration array for NTSC mode
   ADI_DEV_ACCESS_REGISTER NTSCCfg[]=
                                                {{ADV717x_MR0,  0x00},    // register
address, configuration data
//                                          {ADV717x_MR1,           0x58},
                                            {ADV717x_MR1,           0x59},
                                            {ADV717x_MR2,           0x00},
//                                                  {ADV717x_MR3,   0x00},
                                            {ADV717x_MR3,           0x04},
                                            {ADV717x_MR4,           0x10},
                                            {ADV717x_TMR0,          0x00},
                                            {ADV717x_TMR1,          0x00},
                                            {ADI_DEV_REGEND,0    }};        // End of
register access

    // ADV7171 register configuration array for PAL mode
   ADI_DEV_ACCESS_REGISTER PALCfg[]=
   {{ADV717x_MR0,          0x05},    // register address, configuration data
                                            {ADV717x_MR1,           0x10},
                                            {ADV717x_MR2,           0x00},
                                            {ADV717x_MR3,           0x00},
                                            {ADV717x_MR4,           0x00},
                                            {ADV717x_TMR0,          0x08},
                                            {ADV717x_TMR1,          0x00},
                                            {ADI_DEV_REGEND,0    }};        // End of
register access


          if (mode == NTSC){
          ezErrorCheck(adi_dev_Control(EncoderDevHandle,
ADI_ADV717x_CMD_SET_SCF_REG, (void *) ADV717x_SCF_VALUE_NTSC));

                    // configure encoder(ADV7171 or ADV7179) in selected mode
          ezErrorCheck(adi_dev_Control(EncoderDevHandle,
ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *)NTSCCfg));
          }
          else {
             ezErrorCheck(adi_dev_Control(EncoderDevHandle,
ADI_ADV717x_CMD_SET_SCF_REG, (void *) ADV717x_SCF_VALUE_PAL_BI));

                    // configure encoder(ADV7171 or ADV7179) in selected mode
          ezErrorCheck(adi_dev_Control(EncoderDevHandle,
ADI_DEV_CMD_REGISTER_TABLE_WRITE, (void *)PALCfg));

          }

}
```

```
/*********************************************************************
* Leitura dos registros de status do ADV7183
*********************************************************************/
static void Read7183StatusReg(ADI_DEV_DEVICE_HANDLE AD7183DevHandler)
{
        u32 Result = 0, i;

   // array to hold the read AD7183 subaddress register value
   u16 Read_data[4] ={0};


   ADI_DEV_ACCESS_REGISTER Regs[] =
        {{ ADV7183_STATUS1_RO,          0 },            // Register address to access,
corresponding register data
      { ADV7183_IDENT_RO,       0 },
      { ADV7183_STATUS2_RO,     0 },
      { ADV7183_STATUS3_RO,     0 },
      { ADI_DEV_REGEND,         0 }};     // Register access delimiter (indicates end of
register access)

//To read list of registers in DevRegs
   Result =  adi_dev_Control(AD7183DevHandler,
ADI_DEV_CMD_REGISTER_TABLE_READ, (void *)&Regs);
   if (Result != 0) printf("CMD_SELECTIVE_REGISTER_READ failed(error:%x)\n",Result);
   else {
        // print the values
                printf("AD7183: STATUS1 IDENT STATUS2 STATUS3\n ");
                for (i=0; i<4; i++){
                printf("0x%02X ",Regs[i].Data);
                }
                printf("\n");

   }

}


/*********************************************************************
* Leitura do modo(PAL or NTSC) do encoder do EZ-Kit(ADV7171 or ADV7179)
*********************************************************************/
static void ReadEncoderRegs(ADI_DEV_DEVICE_HANDLE EncoderDevHandler)
{
        u32 i,EncoderSCFValue,Result;

  // ADV7171 register configuration array for NTSC mode
   ADI_DEV_ACCESS_REGISTER read[]={{ADV717x_MR0,     0},             // register
address, configuration data

        {ADV717x_MR1,   0},

        {ADV717x_MR2,   0},

        {ADV717x_MR3,   0},
```

```
        {ADV717x_MR4,   0},

        {ADV717x_TMR0,          0},

        {ADV717x_TMR1,          0},

        {ADI_DEV_REGEND,0       }       };      // End of register access

        Result = adi_dev_Control(EncoderDevHandler,
ADI_ADV717x_CMD_GET_SCF_REG, (void *)&EncoderSCFValue);
        if ( Result != ADI_DEV_RESULT_SUCCESS) printf("Get SCF-Reg failed\n");
        else
        printf("\nEncoder SCF=0x%x\n",EncoderSCFValue);


           // Read ADV7171 in selected mode
   Result = adi_dev_Control(EncoderDevHandler, ADI_DEV_CMD_REGISTER_TABLE_READ,
(void *)read);
        if ( Result != ADI_DEV_RESULT_SUCCESS) printf("Selective Read failed\n");
        else {
                printf("Encoder Regs: ");

                for(i=0;i<7;i++){
                printf("0x%x ",read[i].Data);
                }
                printf("\n");
        }

}


/*********************************************************************

        Function:       Copy Odd to Even Frame

        Description:    Esta função copia o campo 2 sobre o campo 1
                                no buffer de saída do 7183.

                                Isto é necessário para reduzir o flicker causado
                                quando o campo de vídeo par e impar
armazenam frames
                                diferentes.

*********************************************************************/

static void CopyField2ToField1(void)

{

        u32 i,j;
        u32 *ipF, // pointer to field2
                *opF; // pointer to field1

        // Field1, 1st active line number * line length(in 32 bit) + horizontal blanking length
        // pointer to field1
```

```
        ipF =((u32 *)sFrame1) + ACTIVE_FIELD2*FRAME_DATA_LEN_32BIT +
HORIZONTAL_BLANKING;

        // Field2, 1st active line number * line length(in 32 bit) + horizontal blanking length
        // pointer to field2
        opF= ((u32 *)sFrame1)+ ACTIVE_FIELD1*FRAME_DATA_LEN_32BIT +
HORIZONTAL_BLANKING;

        for(j=0;j<NUM_LINES_ACTIVE;ipF+=HORIZONTAL_BLANKING,opF+=HORIZONTAL
_BLANKING,j++)
                for(i=0;i<NUM_COLS_ACTIVE;ipF+=1,opF+=1,i++)
                        *opF=*ipF;          //copy field1 to field 2
}


/*********************************************************************

        Function:       InitiatesBuffer

        Description:    Esta função inicializa o buffer que armazena o frame
                                de vídeo.

                                Ela é necessária para o preechimento do buffer.

*********************************************************************/

static void InitiatesBuffer(char *sFrame, bool fColour)    {

   #define GREEN3

   u32 colours[] = {0xEB80EB80,     // White
                                0xD292D210,             // Yellow
                                0xAA10AAA6,             // Cyan
                                0x91229136,             // Green
                                0x6BDE6BCA,             // Magenta
                                0x52F0525A,             // Red
                                0x296E29F0,             // Blue
                                0x10801080};    // Black
        u32 blanking = 0x10801080;
        u32 blankingActive = 0xD2F0D210;
        int i,w;
        lframe *lineFrame = (lframe*)sFrame;

        memset(sFrame, 0x00000000, FRAME_SIZE);

        for( i=0; i< NUM_LINES; i++)        {
           // fill black on all horizontal blanking areas
           for(w=0; w<(HORIZONTAL_BLANKING-2);w++)
                lineFrame->blankVideo[w] = blanking;

           // fill black on all active video areas
           for(w=0;w<NUM_COLS_ACTIVE;w++)
        lineFrame->activeVideo[w] = blanking;

           // fill with colour bars on all active video areas
           if(i<ACTIVE_FIELD1)      {
```

```
                lineFrame->eav = 0xB60000ff;   // or 182
                lineFrame->sav = 0xAB0000ff;   // or 171
            }
            // field1 active video
            else if(i<ACTIVE_FIELD1+NUM_LINES_ACTIVE)     {
                lineFrame->eav = 0x9D0000ff;   // or 157
                lineFrame->sav = 0x800000ff;   // or 128
                    for(w=0;w<NUM_COLS_ACTIVE;w++)       {
                        if(fColour)          {
                            if(i<(ACTIVE_FIELD1+25) && i>ACTIVE_FIELD1+20)   {
                                lineFrame->activeVideo[w] = colours[GREEN];
                            }
                            else      {
                            lineFrame->activeVideo[w] = colours[(int)w/45];
                            }
                        }
                    else      {
                        if((w>59 && w<299) &&
                            (i>ACTIVE_FIELD1+29 &&
i<ACTIVE_FIELD1+NUM_LINES_ACTIVE-34))
                            lineFrame->activeVideo[w] = blanking;
                        else
                            lineFrame->activeVideo[w] = blankingActive;
                    }
                        }
            }
            // field1 second vertical blanking
            else if(i<ACTIVE_FIELD1+NUM_LINES_ACTIVE+
                        BOTTOM_VERTICAL_BLANKING)   {
                lineFrame->eav = 0xB60000ff;   // or 182
                lineFrame->sav = 0xAB0000ff;   // or 171
            }
            // field2 first vertical blanking
            else if(i<ACTIVE_FIELD2)          {
                lineFrame->eav = 0xF10000ff;   // or 241
                lineFrame->sav = 0xEC0000ff;   // or 236
            }
            // field2 active video
            else if(i<ACTIVE_FIELD2+NUM_LINES_ACTIVE)     {
                lineFrame->eav = 0xDA0000ff;   // or 218
                lineFrame->sav = 0xC70000ff;   // or 199
                    for(w=0;w<NUM_COLS_ACTIVE;w++)       {
                        if(fColour)
                            if(i<(ACTIVE_FIELD2+25) && i>ACTIVE_FIELD2+20)   {
                                lineFrame->activeVideo[w] = colours[GREEN];
                            }
                            else      {
                            lineFrame->activeVideo[w] = colours[(int)w/45];
                            }
                    else      {
                        if((w>59 && w<299) &&
                            (i>ACTIVE_FIELD2+29 &&
i<ACTIVE_FIELD2+NUM_LINES_ACTIVE-34))
                            lineFrame->activeVideo[w] = blanking;
                        else
                            lineFrame->activeVideo[w] = blankingActive;
```

```
                    }
                        }
                }
                // field1 second vertical blanking
                else if(i<ACTIVE_FIELD2+NUM_LINES_ACTIVE+
                                BOTTOM_VERTICAL_BLANKING)   {
                    lineFrame->eav = 0xF10000ff;   // or 241
                    lineFrame->sav = 0xEC0000ff;   // or 236
                }

                lineFrame++;
            }
}

/*********************************************************************

        Function:       Rotate Line in Field1 and Field2 90 degrees

        Description:    Esta função lê a primeira linha do campo 1 do
                                        buffer de entrada armazenando-a defasada de
90 graus
                                        nos campos 1 e 2 do buffer de saída.
                                        .
                                        Ela é necessária para transformar linhas em
colunas.

*********************************************************************/

static void RotateLine_90G(void)

{

        int w;
        lframe *iFrame;
        lframe *oFrame1,*oFrame2;

        if(flgDirection)    {
                if(currentCol == NUM_LINES_ACTIVE+60)
                        currentCol = 60;
        }
        else    {
                if(currentCol <= 60)
                        currentCol = 60;
        }

        // Points to 23th line of ACTIVE_FIELD1 of input buffer
        //
   iFrame = (lframe*)sFrame0+ACTIVE_FIELD1+23;

   // Points to 1st line of both Active Fields of output buffer
   //
   oFrame1 = (lframe*)sFrame1+ACTIVE_FIELD1+30;
   oFrame2 = (lframe*)sFrame1+ACTIVE_FIELD2+30;

   for(w=0;w<351;w+=2, oFrame1++,oFrame2++)        {
        oFrame1->activeVideo[currentCol] = iFrame->activeVideo[w];
```

```
            oFrame2->activeVideo[currentCol] = iFrame->activeVideo[w+1];
    }

    // Set next col to process on output buffer
    //
    if(flgDirection)
            currentCol++;
    else
            currentCol--;

    flgDirection = true;

}


//-------------------------------------------------------------------------//
// Function:        Init_SCLCK
//
// Parameters:      Valores em hexa
//                                  CSEL; 0x00, 0x01, 0x10, 0x11
//                                  SSEL; 0x0000 up to 0x00ff
// Return:          Sem retorno
//
// Description:     Esta função configura SCLCK                                    //
//-------------------------------------------------------------------------//
void Init_PLL_DIV(u32 value)        {
            *pPLL_DIV = value;
}

/*************************************************************************

            Function:           FilterGrayScale

            Description:        O trecho a seguir faz a convolução de uma imagem
                                    em tons de cinza (0 - 255) com um filtro
                                    passa-alta

                                    Kernel do filtro:
                                                    -1    -1   -1
                                            1/8x    -1     16   -1
                                                    -1    -1   -1

                                    As variáveis Ni e Nj representam  o número de
                                    linhas e o número de colunas da imagem..
                                    O ponteiro image1r  manipula a imagem de
entrada.
                                    O ponteiro xin1 recebe a imagem já filtrada.
                                    c1,c2,c3,c4,c5,c6,c7,c8 e c9 são variáveis do
                                    tipo float

*************************************************************************/

static void FilterGrayScale(void)
{

    int Ni = NUM_LINES_ACTIVE*2;
```

```
    int Nj = NUM_COLS_ACTIVE, i, j;
    float c1, c2, c3, c4, c5, c6, c7, c8, c9;
    u32 lumen;

            for(i=0;i<Ni;i++)     {
                        for(j=0;j<Nj;j++)
                        {

                                    sFilterOut[i*Nj+j] = sFilterIn[i*Nj+j];

                                    if(!(i==0 || i==Ni-1 || j==0 || j==Nj-1))
                                    {
                                                // extraindo a luminancia dos pixels YCbCr
                                                // que estão na memória como: CrYCbY
                                                //
                                                c1 = (float)((sFilterIn[(i-1)*Nj+(j-1)] & 0xFF00) >> 8);
                                                c2 = (float)((sFilterIn[(i-1)*Nj+(j)] & 0xFF00) >> 8);
                                                c3 = (float)((sFilterIn[(i-1)*Nj+(j+1)] & 0xFF00) >> 8);
                                                c4 = (float)((sFilterIn[(i)*Nj+(j-1)] & 0xFF00) >> 8);
                                                c5 = (float)((sFilterIn[(i)*Nj+(j)] & 0xFF00) >> 8);
                                                c6 = (float)((sFilterIn[(i)*Nj+(j+1)] & 0xFF00) >> 8);
                                                c7 = (float)((sFilterIn[(i+1)*Nj+(j-1)] & 0xFF00) >> 8);
                                                c8 = (float)((sFilterIn[(i+1)*Nj+(j)] & 0xFF00) >> 8);
                                                c9 = (float)((sFilterIn[(i+1)*Nj+(j+1)] & 0xFF00) >> 8);

                                                // a linha a seguir filtra a imagem com o filtro de Sobel
                                                // (destaca os contornos)
                                                //          lumen = fabs(((c7+2*c8+c9)-(c1+2*c2+c3))) +
                                                //                      fabs(((c3+2*c8+c9)-(c1+2*c4+c7)));

lumen = (u32)fabs((-(c1+c2+c3+c4+c6+c7+c8+c9)+(FILTER_FACTOR*c5))/8);

                                                // como a luminancia varia entre 16-235, faz o ajuste
                                                // necessario
                                                //
                                                if(lumen > 235) lumen = 235;
                                                else if(lumen < 16) lumen = 16;

                                                // prepara a nova luminancia para inserir no pixel
                                                //
                                                lumen <<= 8;
                                                lumen = ((lumen << 16) | lumen);       //0xYY00YY00

                                                // atribuo o novo valor da luminancia ao pixel de saida
                                                //
                                                //          extrai o Cr e o Cb do pixel
                                                //
                                                //
                        sFilterOut[i*Nj+j] = ((sFilterOut[i*Nj+j] & 0x00FF00FF) | lumen);

                                    }
                        } // do for i e j
            }
}

/*********************************************************************
```

```
            Function:           Populates image in buffer

            Description:        Esta função preenche o buffer de entrada para
                                        a aplicação do filtro passa-alta.

*****************************************************************/

static void PopulateImageInBuffer(void)

{

            int lout,col;
            Iframe *iFrame1,*iFrame2;

            iFrame1 = ((Iframe*)sFrame1)+ACTIVE_FIELD1;          // pointer to 1st line Active
Field1
   iFrame2 = ((Iframe*)sFrame1)+ACTIVE_FIELD2;       // pointer to 1st line Active Field2

            for(lout=0; lout<NUM_LINES_ACTIVE*2; lout+=2, iFrame1++, iFrame2++) {
            memcpy(&sFilterIn[lout*NUM_COLS_ACTIVE], &iFrame1->activeVideo[0],1440);
            memcpy(&sFilterIn[(lout+1)*NUM_COLS_ACTIVE], &iFrame2->activeVideo[0], 1440);
            }

}

/*****************************************************************

            Function:           Fills DMA buffer input

            Description:        Esta função move a imagem do buffer onde foi
                                        aplicado o filtro passa-alta para o buffer de
                                        saída.

*****************************************************************/

static void FillsDMABufferOut(void)

{

            int lin,col;
            Iframe *oFrame1,*oFrame2;

            oFrame1 = (Iframe*)sFrame1+ACTIVE_FIELD1;          // pointer to 1st line Active
Field1
   oFrame2 = (Iframe*)sFrame1+ACTIVE_FIELD2;       // pointer to 1st line Active Field2

            for(lin=0; lin<NUM_LINES_ACTIVE*2;oFrame1++, oFrame2++,lin+=2)      {
            memcpy(&oFrame1->activeVideo[0], &sFilterOut[lin*NUM_COLS_ACTIVE], 1440);
            memcpy(&oFrame2->activeVideo[0], &sFilterOut[(lin+1)*NUM_COLS_ACTIVE],
1440);
            }
}

/*****************************************************************
```

```
            Function:           Applly Filter Gray Scale

            Description:        Esta função aplica o filtro passa-alta.

*********************************************************************/

static void ApplyFilterGrayScale(void)

{

   PopulateImageInBuffer();

            FilterGrayScale();

            FillsDMABufferOut();

            CopyField2ToField1();

}

/*********************************************************************

            Function:           Interpolation

            Description:        Esta função calcula os valores médios dos pixels
                                                das colunas.

*********************************************************************/

static void Interpolation(void)

{
   #define TOTAL_LINES    200
   #define MAX_MEDIA      1.4
   #define MIN_MEDIA      0.6

   int lin, col, count, x=0;
   int valColb1,valColb2,valColb3,valColb4;
   int mediaMaxb1, mediaMaxb2, mediaMaxb3, mediaMaxb4;
   int mediaMinb1, mediaMinb2, mediaMinb3, mediaMinb4;
   u32 mediaMax, mediaMin;
   Iframe *iFrame, *i1stLineActive, lineMedia;

            // pointer to 23th line Active Field1
            i1stLineActive = iFrame = (Iframe*)sFrame0+ACTIVE_FIELD1+23;
   memset(&lineMedia.activeVideo[0], 0x00, NUM_COLS_ACTIVE*4);

            for(col=0;col<NUM_COLS_ACTIVE;col++)      {

                        valColb1 = valColb2 = valColb3 = valColb4 = 0;

                for(lin=0;lin<TOTAL_LINES;lin++,iFrame++) {
          valColb1 += ((iFrame->activeVideo[col] & 0xFF000000) >> 24);
          valColb2 += ((iFrame->activeVideo[col] & 0x00FF0000) >> 16);
          valColb3 += ((iFrame->activeVideo[col] & 0x0000FF00) >>  8);
          valColb4 += ( iFrame->activeVideo[col] & 0x000000FF);
```

```
      }
      lineMedia.activeVideo[col] = (((valColb1/TOTAL_LINES) << 24) |
                                                                  ((valColb2/TOTAL_LINES)
<< 16) |
                                                                  ((valColb3/TOTAL_LINES)
<<  8) |
                                                                   (valColb4/TOTAL_LINES));
      iFrame = i1stLineActive;
   }

            for(col=0;col<NUM_COLS_ACTIVE;col++)      {

                  mediaMaxb1 = ((lineMedia.activeVideo[col] & 0xFF000000) >> 24) *
MAX_MEDIA;
                  if(mediaMaxb1 > 235) mediaMaxb1 = 235;

                  mediaMaxb2 = ((lineMedia.activeVideo[col] & 0x00FF0000) >> 16) *
MAX_MEDIA;
                  if(mediaMaxb2 > 240) mediaMaxb2 = 240;

                  mediaMaxb3 = ((lineMedia.activeVideo[col] & 0x0000FF00) >> 8)  *
MAX_MEDIA;
                  if(mediaMaxb3 > 235) mediaMaxb3 = 235;

                  mediaMaxb4 =  (lineMedia.activeVideo[col] & 0x000000FF)       *
MAX_MEDIA;
                  if(mediaMaxb4 > 240) mediaMaxb4 = 240;

      mediaMax   = ((mediaMaxb1 << 24) | (mediaMaxb2 << 16) |
                              (mediaMaxb3 <<  8) | mediaMaxb4);

            mediaMinb1 = ((lineMedia.activeVideo[col] & 0xFF000000) >> 24)  * MIN_MEDIA;
                  if(mediaMinb1 < 16) mediaMinb1 = 16;

            mediaMinb2 = ((lineMedia.activeVideo[col] & 0x00FF0000) >> 16)  * MIN_MEDIA;
                  if(mediaMinb2 < 16) mediaMinb2 = 16;

            mediaMinb3 = ((lineMedia.activeVideo[col] & 0x0000FF00) >>  8)  * MIN_MEDIA;
                  if(mediaMinb3 < 16) mediaMinb3 = 16;

            mediaMinb4 =  (lineMedia.activeVideo[col] & 0x000000FF)        * MIN_MEDIA;
                  if(mediaMinb4 < 16) mediaMinb4 = 16;

      mediaMin   = ((mediaMinb1 << 24) | (mediaMinb2 << 16) |
                              (mediaMinb3 << 8)  | mediaMinb4);

                  valColb1 = valColb2 = valColb3 = valColb4 = 0;
                  count = 0;

            for(lin=0;lin<TOTAL_LINES;lin++,iFrame++) {
               if(iFrame->activeVideo[col] > mediaMin &&
                  iFrame->activeVideo[col] <= mediaMax)          {
                 valColb1 += ((iFrame->activeVideo[col] & 0xFF000000) >> 24);
                 valColb2 += ((iFrame->activeVideo[col] & 0x00FF0000) >> 16);
                 valColb3 += ((iFrame->activeVideo[col] & 0x0000FF00) >>  8);
                 valColb4 +=  (iFrame->activeVideo[col] & 0x000000FF);
```

```
                    count++;
                }
        }

        if(count > 0)   {
                i1stLineActive->activeVideo[col] = (((valColb1/count) << 24) |

            ((valColb2/count) << 16) |

            ((valColb3/count) <<  8) |

(valColb4/count));
        }

        iFrame = i1stLineActive;
    }

    return;

}
```

Apêndice 2

Características técnicas do processador *Blackfin® ADSP-BF531/ADSP-BF532/ADSP-BF533.* [13] [14]

a) Funcionalidades.

Processador de alta performance *Blackfin* de até 600 MHz.

- Duas MACs de 16-bits, duas ULAs de 40-bit, quatro ULAs de vídeo de 8-bits, *shifter register* de 40-bit de tecnologia semelhante a RISC-like e modêlo de instrução de fácil programação e suporte amigável para compilação
- Debug avançado, passo-a-passo, e monitoramento de performance tensão de processador de 0.85 V até 1.30 V VDD com regulagem de tensão on-chip de 1.8 V, 2.5V, and 3.3 V compatíveis com I/O
- 160-ball CSP_BGA, 169-ball PBGA, and 176-lead LQFP packages

b) Memória.

Até 148K bytes de memória on-chip:

- 16K bytes de SRAM/Cache para instrução
- Até 64K bytes SRAM para instrução
- Até 32K bytes de SRAM/Cache para dados
- Até 32K bytes de SRAM para dados
- 4K bytes de SRAM *scratchpad*

Unidade de gerenciamento de memória permitindo proteção de memória.
Controlador de memória externa com suport *glueless* para SDRAM, SRAM, flash, e opções de *booting* através da SPI® e memória externa.

c) Periféricos.

Interface paralela para periféricos PPI/GPIO, suportando o formato vídeo ITU-R 656.
Dois canais, portas seriais síncronas *full duplex*, suportando oito canais I2S estéreo Quatro DMAs de memória para memória.
Oito DMAs para periféricos.
Porta compatível com SPI.

Três contadores/timer de 32-bits com suport PWM.

Relógio de tempo real e *watchdog timer .*

32-bit core timer .

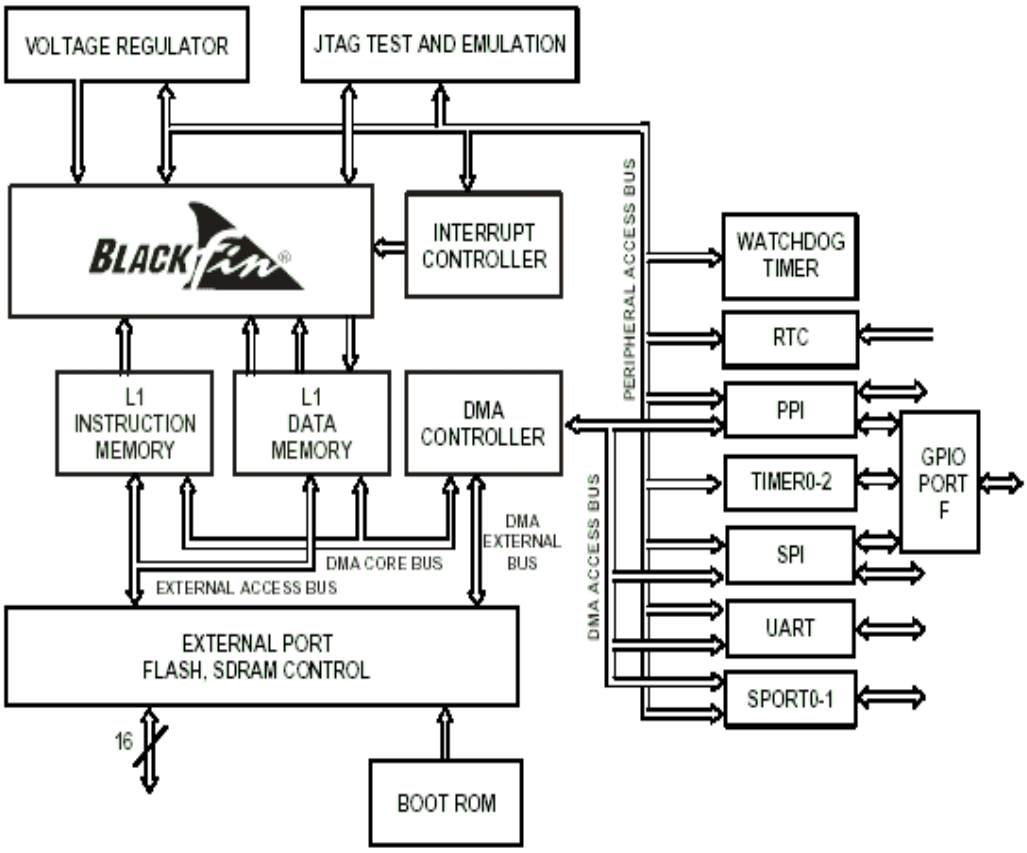
Até 16 pinos de E/S de propósito geral (GPIO) .

UART com suporte IrDA® .

Event handler .

Interface debug/JTAG *On-chip* PLL com capacidade de multiplicação de frequência de 0.5x até 64x.

Diagrama em blocos interno do processador *Blackfin® ADSP-BF531/ADSP-BF532/ADSP-BF533:*



d) *Core* do proessador *blackfin.*

O *core* do processador Blackfin contem 2 multiplicadores de 16-bits, dois acumuladores de 40-bits, duas ULAs de 40-bits, quatro ULAs para vídeo, e um *shifter* de 40-bits. A unidade de computação processa registradores dados de 8-bit, 16-bit, or 32-bit.

A unidade de computação contem oito registradores de 32-bits. Durante a realização de operações sobre operandos de dados de 16-bits, os registradores atuam como 16

registradores independentes de 16-bits. Todos os operandos para as operações de computação provêm dos registradores de múltiplas funções e dos registradores de instrução.
Cada MAC pode executar multiplicações de 16-bits por 16-bits a cada ciclo, acumulando os resultados em acumuladores de 40-bits. Suporte a formatos com sinal e sem sinal, arredondamento e saturação.

As ULAs realizam o conjunto tradicional de operações lógicas e aritméticas em dados de 16-bits ou de 32-bit. Além disso, são incluídas várias instruções especiais que aceleram inúmeras tarefas de processamento de sinais. Dentre elas incluem-se operações de bit tais como: extração de campo e contagem de população, módulo de multiplicação 232, divisão primitivas, saturação e arredondamento e deteção de sinal/expoente. O conjunto de instruções de vídeo incluem alinhamento de byte e operações de *packing*, adições em 16-bits e 8-bits com *clipping*, operaçẽos de média em 8-bit, e operações de subtração/valor absoluto/acumulação(SAA) de 8-bits. Dispõe ainda de instruções de seleção/comparação e pesquisa em vetores.
Para certas instruções, duas operações na ULA podem ser realizadas simultâneamente sobre pares de regitradores(uma metade de 16-bit *high* e uma metade de 16-bit *low* de um registrador de computação). São possíveis quatro operações de 16-bits usando a segunda ULA.
O shifter de 40-bits pode *shifitar* e rotacionar sendo usado para suportar normalização, extração de campos e instruções de *field deposit*.

O *program sequencer* controla o fluxo de execução de instruções, incluindo a decodificação e o alinhamento de instruções. Para controlar o fluxo do programa, o *sequencer* suporta o PC relativo e os *jumps* condicionais indiretos(com predição estática de *branching*), e chamdas a subrotinas. É disponibilizado o suporte à *zero-overhead looping* no hardware. A arquitetura e totalmente inter-travada, desta forma, o programador não precisa gerenciar o *pipeline* ao executar instruções que tenham dependências de dados.

A unidade de endereçamento aritmético disponibiliza dois endereços para dois *fetches* simultâneos da memória. Ela contem um registrador de múltipla função que consiste de quatro conjuntos de registradores de índice de 32-bits, registradores de *modify, length*, e registradores de base(para *bufferização* circular), e oito registradores de ponteiros adicionais de 32-bits (para manipulação de índice de pilha *C-style*).

Os processadores Blackfin suportam a arquitetura Harvard modificada em combinação com uma estrutura hierárquica de memória. As memórias nível(L1) são aquelas que operam tipicamente à velocidade plena do processador com pouca ou nenhuma latência. No nível L1, a memória de instrução retêm somente instruções. As duas memóias de dados retêm dados, e uma memória de dados *scratchpad* armazena a pilha e informações de variáveis locais. Além disso, múltiplos blocos de memória L1 estão disponíveis, oferecendo múltiplas configurações de SRAM e cache. A unidade de gerenciamento de memória (MMU) fornece proteção de

memória para tarefas individuais que devem executar no *core* e pode proteger o sistema de registradores de acesso não intencionados.

A arquitetura disponibiniliza três modos de operação: modo do usuário, modo do supervisor, e modo emulação. O modo usuário tem acesso restrito a certos recursos do sistema, dispondo assim de um ambiente de software protegido, enquanto em modo supervisor o acesso ao sistema é irrestrito e aos recursos do *core* também.

O conjunto de instruções do processador Blackfin foi otimizado para que os *opcodes* de 16-bit representem as instruções mais utilizadas, resultando em uma excelente densidade do código compilado. As intruções DSP complexas são codificadas em opcodes de 32-bit, representando instruções multifuncionais totalmente funcionais. Os processadores Blackfin processors suportam uma capacidade limitada de multi-execução, onde uma instrução de 32-bit pode ser executada em paralelo com duas intruções de 16-bits, permitindo que o programador use vários recursos do *core* em um único ciclo de instrução.

A linguagem assembly do processador Blackfin usa uma sintax algébrica para uma fácil codificação e legibilidade. A arquitetura foi otimizada para o uso em conjunção com o compilador C/C++, resultando em um rápida e eficiente implementação do software.

e) Controladores DMA.

O processador ADSP-BF531/ADSP-BF532/ADSP-BF533 tem múltiplos independentes canais DMA que suportam transferência automática de dados com o mínimo de *overhead* para o *core* do processador. As transferências DMA podem correr entre as memórias internas do processador e qualquer de seus periféricos mapeados pelo DMA. E ainda, as transferências DMA podem ocorrer entre qualquer periférico mapeado pelo DMA e dispositivos externos conectados às interfaces de memórias externas, incluindo o controlador SDRAM e o controlados de memória assíncrona. Os periféricos atendidos pelo DMA incluem os SPORTs, SPI port, UART, e a PPI. Cada periférico mapeado pelo DMA tem no mínimo um canal DMA dedicado.

O controlador DMA do processador ADSP-BF531/ADSP-BF532/ADSP-BF533 suporta ambas as transferências DMA *1-dimensional* (1-D) e *2-dimensional* (2-D). A inicialização da transferência DMA pode ser implementada a partir dos registradores ou a partir de um conjunto de parâmetros chamados blocos descritores.

A transferência 2-D DMA suporta linhas e colunas com até 64K elementos por 64K elementos, e passos de linhas e colunas de até ±32K elementos. E mais, o passo de coluna pode ser menor que o passo de linha, permitindo a implementação de cadeia de daods sejam

intrelaçadas. Esta característica é especialmente interessante em aplicações de vídeo onde os dados podem ser intrelaçados *on the fly*.

Os exemplos de tipos de transferências DMA suportados pelo controlador de DMA do processador ADSP-BF531/ ADSP-BF532/ADSP-BF533 incluem:

- Única, buffer linear que pára após a transferência completa
- Circular, *buffer* reciclável que interrompe a cada transferência completa ou parcial
- 1-D ou 2-D DMA usando descritores em lista encadeada
- 2-D DMA usando *array* de descritores, especificacdo somente o endereço base DMA dentro de uma página comum

Além dos canais dedicados aos periféricos, existem dois pares de canais DMA de memória disponíveis para transferências entre as várias memórias do sistema do processador ADSP-BF531/ ADSP-BF532/ADSP-BF533. Isto permite transferências de blocos de dados entre qualquer das memórias incluindo a SDRAM externa, ROM, SRAM, e memória flash memory com o mínimo de intervenção do processador. As transferências DMA podem ser controladas por metodologia bastante flexível baseadas em descritores ou pelo padrão mecanismo de autobuffer baseado em registradores.

f) Especificaçôes de temporizaçôes

Desde a tabela 12 até a tabela15 descrevem os requisitos de temporizações dos *clocks do* processador ADSP-BF531/ADSP-BF532/ADSP-BF533. Observe as seleções das taxas de MSEL, SSEL, and CSEL para que não excedam o *clock* máximo do *core* e o *clock* do sistema que estão descritos no ítem Taxas Maximas Absolutas, e as frequências descritas na tabela 14 *oscilador controlado de tensão* (VCO). A tabela 14 descreve as condições de operações do *phase-locked loop*(PLL).

Table 12. Core Clock (CCLK) Requirements—400 MHz Models[1]

Parameter		Internal Regulator Setting	$T_{JUNCTION}$ = 125°C Max	All[2] Other $T_{JUNCTION}$ Max	Unit
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.14 V Minimum)	1.20 V	400	400	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.045 V Minimum)	1.10 V	333	364	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.95 V Minimum)	1.00 V	295	333	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.85 V Minimum)	0.90 V		280	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.8 V Minimum)	0.85 V		250	MHz

[1] See Ordering Guide on Page 59.
[2] See Operating Conditions on Page 21.

Table 13. Core Clock (CCLK) Requirements—500 MHz, 533 MHz, and 600 MHz Models

Parameter		Internal Regulator Setting	Max	Unit
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.3 V Minimum)[1]	1.30 V	600	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.2 V Minimum)[2]	1.25 V	533	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.14 V Minimum)[3]	1.20 V	500	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 1.045 V Minimum)	1.10 V	444	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.95 V Minimum)	1.00 V	400	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.85 V Minimum)	0.90 V	333	MHz
f_{CCLK}	CCLK Frequency (V_{DDINT} = 0.8 V Minimum)	0.85 V	250	MHz

[1] Applies to 600 MHz models only. See Ordering Guide on Page 59.
[2] Applies to 533 MHz and 600 MHz models only. See Ordering Guide on Page 59. 533 MHz models cannot support internal regulator levels above 1.25 V.
[3] Applies to 500 MHz, 533 MHz, and 600 MHz models. See Ordering Guide on Page 59. 500 MHz models cannot support internal regulator levels above 1.20 V.

Table 14. Phase-Locked Loop Operating Conditions

Parameter		Min	Max	Unit
f_{VCO}	Voltage Controlled Oscillator (VCO) Frequency	50	Maximum f_{CCLK}	MHz

Table 15. System Clock (SCLK) Requirements

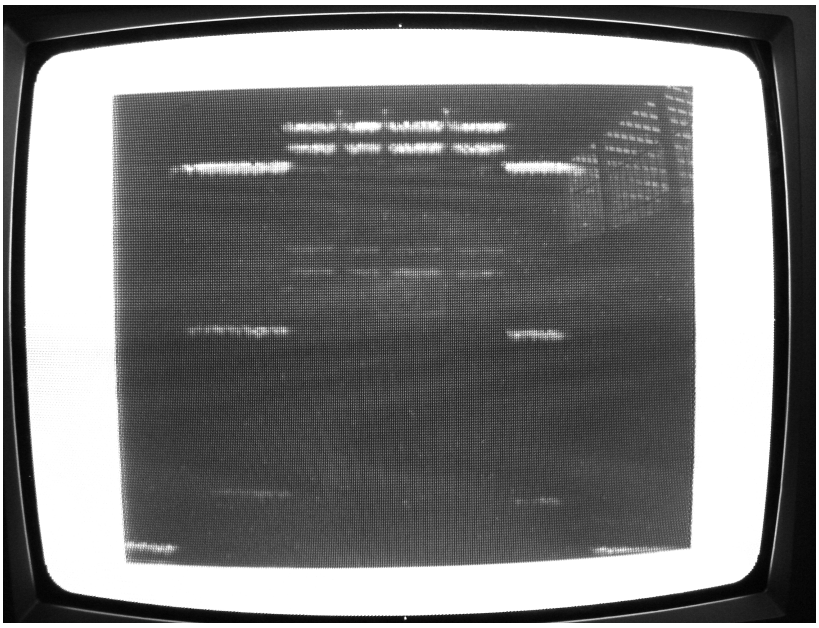
Parameter[1]		V_{DDEXT} = 1.8 V Max	V_{DDEXT} = 2.5 V/3.3 V Max	Unit
MBGA/PBGA				
f_{SCLK}	CLKOUT/SCLK Frequency ($V_{DDINT} \geq 1.14$ V)	100	133	MHz
f_{SCLK}	CLKOUT/SCLK Frequency ($V_{DDINT} < 1.14$ V)	100	100	MHz
LQFP				
f_{SCLK}	CLKOUT/SCLK Frequency ($V_{DDINT} \geq 1.14$ V)	100	133	MHz
f_{SCLK}	CLKOUT/SCLK Frequency ($V_{DDINT} < 1.14$ V)	83	83	MHz

[1] t_{SCLK} (= $1/f_{SCLK}$) must be greater than or equal to t_{CCLK}.

Apêndice 3

Testes com alguns tipos de filtro e suas máscaras, empregadas no pré-processamento da imagem digital.

a) Imagem original:



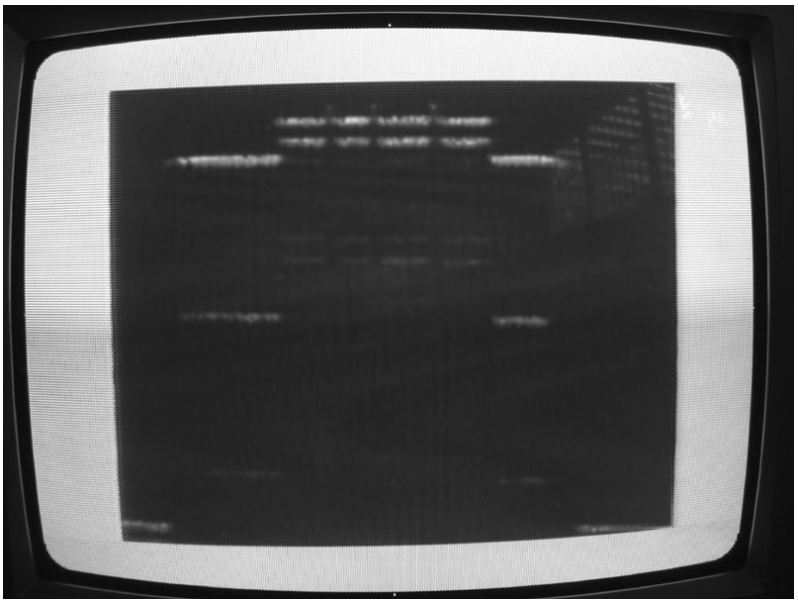
b) Testes com filtros.

b.1) Média:

1/9 x

1	1	1
1	1	1
1	1	1

Primeira iteração:



Segunda iteração:

b.1.1) Mudando a máscara:

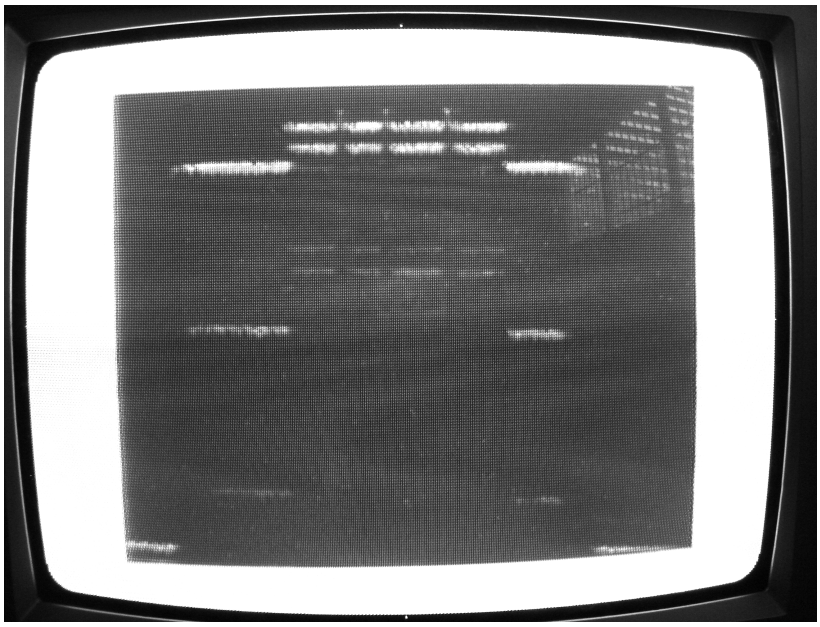
1/18 x

2	2	2
2	2	2
2	2	2

b.2) Passa-alta:

1/9 x

-1	-1	-1
-1	8	-1
-1	-1	-1



b.2.1) Mudando a máscara:

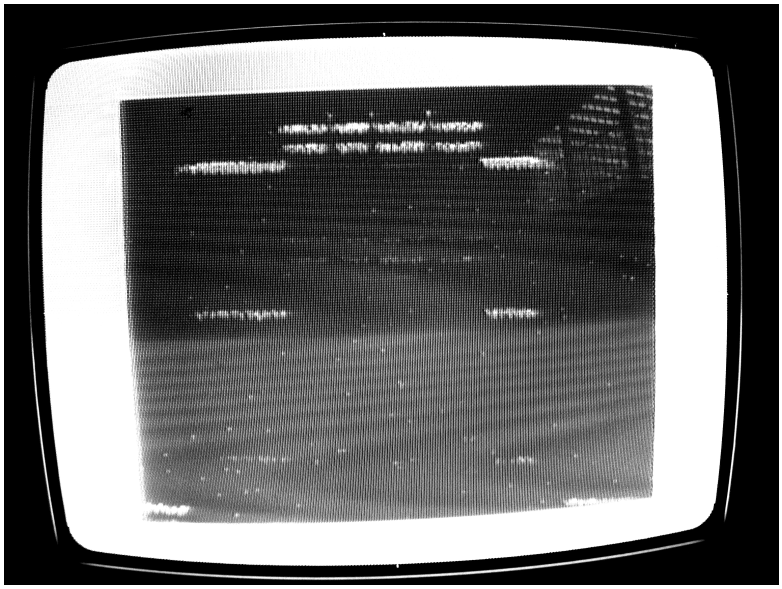
1/16 x

-1	-1	-1
-1	24	-1
-1	-1	-1

b.2.2) Mudando para a máscara adotada no trabalho:

1/8 X

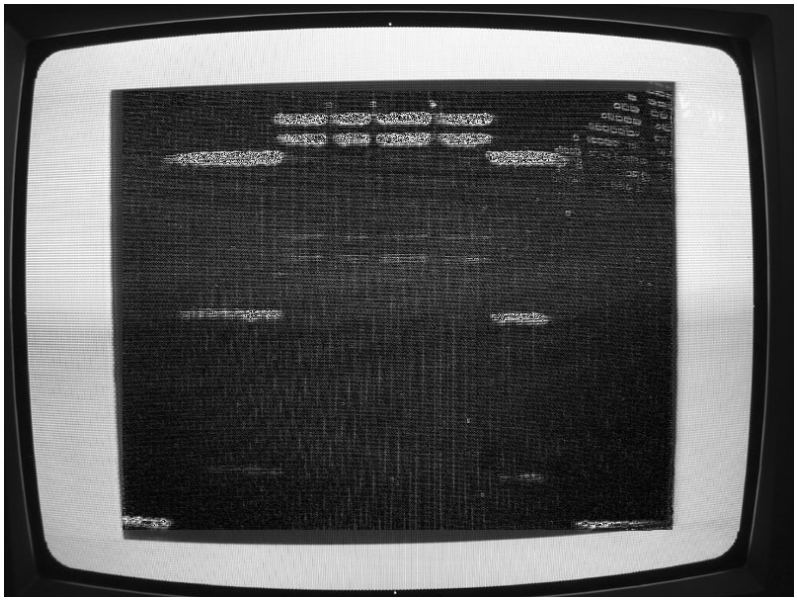
-1	-1	-1
-1	16	-1
-1	-1	-1



b.3) Operador de sobel (realce por diferenciação):

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1



Apêndice 4

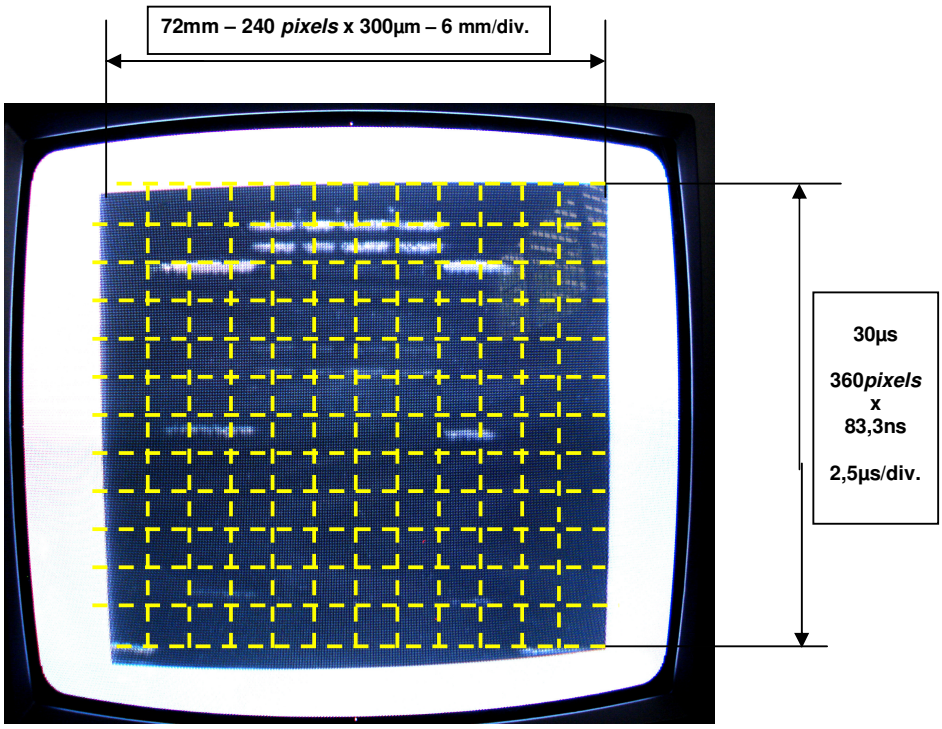
Processos de medição sugeridos.

a) Retícula criada na tela.

Alguns *pixels* podem ser realçados, a fim de gerar uma retícula (como a utilizada nas telas dos osciloscópios) que viabilize o cálculo das distâncias envolvidas nos elementos explorados.

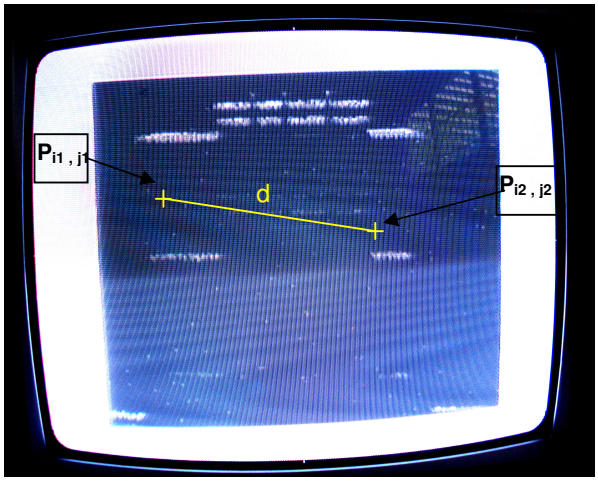
No sentido horizontal, cada retângulo corresponde a 6mm na distância real. Já no sentido vertical, cada retângulo corresponde a 2,5µs multiplicado pela velocidade do ultra-som no meio do corpo explorado.

-



b) Cursores na tela.

Dois cursores móveis na tela, demarcam nos pontos e processador calcula a distância entre eles.



$$d^2 = |(i_2 - i_1).83,3.10^{-9}.v|^2 + |(j_2 - j_1).3.10^{-4}|^2$$

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA-CEFET/RJ

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENADORIA DO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA

DISSERTAÇÃO

SISTEMA ELETRÔNICO DE PROCESSAMENTO DIGITAL PARA A GERAÇÃO DE IMAGENS POR ULTRA-SOM, OPERANDO NO MODO "B"

Sérgio Luiz Fernandes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM TECNOLOGIA.

Data da defesa: 30/11/2007.

Aprovação:

Carlos Henrique Figueiredo Alves, D.Sc.

Aline da Rocha Gesualdi, D.Sc.

Maurício Saldanha Motta, D.Sc.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo