UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

Curso de Pós-Graduação em Engenharia Elétrica - CPGEE

COMPARAÇÃO ENTRE MÉTODOS DIGITAIS DE LINEARIZAÇÃO DE SENSORES

OSVALDO ANDRE BETAT BASILIO

Dissertação para obtenção do título de Mestre em Engenharia

Porto Alegre

2002

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

Curso de Pós-Graduação em Engenharia Elétrica - CPGEE

COMPARAÇÃO ENTRE MÉTODOS DIGITAIS DE LINEARIZAÇÃO DE SENSORES

OSVALDO ANDRE BETAT BASILIO

Engenheiro Eletricista

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia Elétrica - CPGEE, como parte dos requisitos para a obtenção do título de Mestre em Engenharia.
Área de concentração: Instrumentação Eletro-Eletrônica.
Desenvolvida no Laboratório de Processamento de Sinais e Imagem do Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul.

Porto Alegre

2002

COMPARAÇÃO ENTRE MÉTODOS DIGITAIS DE LINEARIZAÇÃO DE SENSORES

OSVALDO ANDRE BETAT BASILIO

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador:_____

Prof. Luigi Carro, UFRGS

Banca Examinadora:

Prof. Dr. Antonio Petraglia, COPPE-UFRJ.

Prof. Dr. Renato Machado de Brito, UFRGS.

Prof. Dr. Walter Fetter Lages, UFRGS.

Coordenador do CPGEE:_____

Prof. Dr. Carlos E. Pereira

Porto Alegre, fevereiro de 2002.

Dedico esta dissertação aos meus pais, Oswaldo Basilio e Helena Betat, por nunca terem medido esforços para que eu pudesse estudar. E a minha esposa Elisandra pela paciência e apoio em todas as horas.

AGRADECIMENTOS

Agradeço a todos professores do Curso de Pós-Graduação em Engenharia Elétrica – CPGEE, que sempre estiveram prontos para ensinar e orientar.

À CAPES pela provisão da bolsa de mestrado.

Aos Professores da banca examinadora, por terem aceito o convite de participarem deste momento tão importe.

Ao Professor Dr.Luigi Carro que me orientou, ensinou e, principalmente, motivou. Muito obrigado.

Aos colegas de curso e de laboratório, Adriane Parraga, Maria da Glória Cataldi, Leandro Cassol, Marcelo Negreiros, Ronaldo Husemann, Rafael Zeilmann, Adão Junior e Francisco Socal, pela ajuda e amizade de todos as horas. Vocês são nota 10!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

A/D	- *Analog to Digital*	- Analógico para Digital.
D/A	- *Digital to Analog*	- Digital para Analógico.
VHDL	- *Very High Speed Integrated Circuit Hardware Description Language.*	
SFDR	- *Spurios Free Dynamic Range*	- Faixa Dinâmica Livre de Espúrios.
DSP	- *Digital Signal Processor*	- Processador de Sinais Digitais.
LUT	- *Look-up table*	- Tabela.
FFT	- *Fast Fourier Transform*	- Transformada Rápida de Fourier.
MOS	- *Metal-Oxide-Silicon.*	
ROM	- *Read Only Memory*	- Memória somente de leitura.
RAM	- *Random Access Memory*	- Memória de acesso randômico.
LMS	- Least Mean Square	- Mínima média quadrática.
FIR	- Finite Impulse Response	- Resposta finita ao impulso.

RESUMO

Na era de sistemas embarcados complexos, a interface direta de dispositivos e sistemas integrados para o mundo real demanda o uso de sensores e seus circuitos analógicos de suporte. Desde que a maioria das características físicas de um sensor requer algum tipo de calibração, este trabalho compara e discute quatro técnicas digitais de calibração adaptadas para aplicação em sistemas embarcados. Para fins de comparação, estes métodos de calibração foram implementados em Matlab5.3, e em um DSP (*Digital Signal Processor*) .

Através das medidas realizadas durante a operação em regime do DSP, pode-se determinar parâmetros importantes de projeto, como potência dissipada e tempo de processamento. Outros critérios de comparação, como área consumida, tempo de processamento, facilidade de automação e taxa de crescimento do custo área e do custo velocidade com o aumento de resolução também foram analisados. Os resultados das implementações são apresentados e discutidos com o objetivo de descobrir qual o melhor método de calibração para aplicações em sistemas embarcados.

ABSTRACT

In the era of complex embedded systems, the direct interface of System-On-Chip devices to the real world demands the use of sensors and their analog supporting circuits. Since the overwhelming majority of physical sensors requires some sort of calibration, this dissertation compares and discusses four digital calibration techniques well suited for embedded systems applications.

Using a DSP board as test vehicle, important design parameters like power dissipation, area, throughput, design time and rate of cost growth with increased resolution were analysed. The obtained implementation results are discussed, and a sensor linearization scheme best suited to embedded systems is presented.

1 Introdução

Sensores inteligentes e integrados são atualmente encontrados em praticamente todos os campos da vida cotidiana. Engenharia biomédica, circuitos de comunicações e aplicações automotivas são alguns exemplos.

Muitos sensores inteligentes são usados dentro de sistemas embarcados, que geralmente aumentam os requisitos de baixo consumo de área, baixa dissipação de potência e alta performance, junto com o menor custo possível e tempo de desenvolvimento reduzido.

Um sensor inteligente genérico pode ser visto como uma unidade composta por um sensor, um circuito condicionador de sinais e, muito provavelmente, devido à tecnologia atual, de um conversor A/D e associado a um microprocessador, preferencialmente incorporados a um mesmo circuito integrado [HOS97].

A principal tarefa de um sensor é converter uma grandeza física em um sinal elétrico que pode ser lido e processado através de circuitos eletrônicos. Devido à tecnologia envolvida e às características não ideais dos componentes utilizados na fabricação destes sensores e circuitos de condicionamento, características não desejáveis como nível DC, ganho e não linearidade podem ser normalmente encontrados, e devem ser corrigidos interna ou externamente ao sistema sensor.

Alguns métodos desenvolvidos para compensar estas características não ideais são apresentados em [JAH98, HOL98, HOR97, BRI96, MAL94, BOL85], para citar alguns. Entretanto, uma comparação entre estes métodos, a fim de verificar a sua adaptabilidade para um conjunto de soluções embarcadas, poderia determinar qual o melhor método ou estabelecer compromissos para a sua utilização nestes produtos. Neste caso específico, a minimização da potência consumida, do tempo de processamento e da área representam uma regra geral. Mas, não somente estes parâmetros físicos são importantes, também algumas características mais difíceis de mensurar são importantes, como é o caso da facilidade de automação para reduzir o ciclo de projeto, e a taxa de aumento de custo quando resoluções mais altas são necessárias em determinadas aplicações.

A principal contribuição deste trabalho é comparar sistematicamente alguns destes métodos, discutindo suas aplicabilidades no mercado de soluções embarcadas.

Esta dissertação esta organizada como segue: na próxima seção tem-se a revisão bibliográfica e teórica. A metodologia utilizada para implementar e comparar cada método é apresentada no capítulo 3. No capítulo 4 são apresentados os resultados experimentais. No capítulo 5 são analisados outros parâmetros mais subjetivos, porém de grande importância no contexto de sistemas embarcados inteligentes. Finalmente, as conclusões e trabalhos futuros são discutidos no capítulo 6.

2 Revisão Teórica e Bibliográfica

Uma vez que o objetivo deste trabalho é comparar métodos digitais de linearização utilizados em sistemas de medição embarcados, será apresentado nas seções a seguir um resumo dos principais trabalhos da área. Inicialmente, serão abordados assuntos referentes à definição de sensores inteligentes, à importância da micro-eletrônica dentro deste contexto, e aos principais problemas e defeitos associados à tecnologia envolvida na sua construção.

Finalmente, serão apresentados alguns dos principais métodos digitais de linearização, dando-se ênfase aos métodos implementados neste trabalho.

2.1 SENSORES INTELIGENTES INTEGRADOS

O avanço tecnológico das últimas décadas, principalmente no campo da micro-eletrônica, contribuiu de maneira decisiva para o surgimento do conceito de sensor integrado inteligente. Middelhoek e Audet [MID87] analisaram a utilização do silício como material para a construção de sensores, de modo a permitir a sua integração e a de circuitos de processamento de sinais em um mesmo circuito integrado. Porém, para que este atrativo conceito fosse realizável, seria necessário que o silício apresentasse efeitos físicos ou químicos desejáveis, que o sensor pudesse ser projetado baseado nestes efeitos e que o sensor fosse compatível com a tecnologia dos circuitos disponíveis.

O surgimento de novos sensores, com tecnologia desenvolvida para utilização em circuitos integrados foram reportados [MID87]. Entre os sensores citados estão :

- Sensores de silício para sinais irradiados, onde os mais conhecidos são usados para detecção de luz, pois, o silício é especialmente adequado para detecção de luz na faixa de espectro visível. Quatro áreas são citadas como principais: CCDs para utilização em imagem, foto-diodos sensíveis a cor, detetores para radiação nuclear e sensores infra-vermelho baseados em termopares integrados em silício.

- Sensores de silício, utilizados para medir parâmetros mecânicos tais como posição, deslocamento, pressão, fluxo, etc. O dispositivo mais estudado é o chamado diafragma com *strain-gauge* de silício difuso, utilizado para medir pressão. Foto diodos sensíveis à posição e sensores de fluxo, onde o fluxo é convertido em uma diferença de temperatura, são alguns exemplos.
- Sensores de silício para sinais magnéticos, baseados em placas de efeito Hall, encontram aplicação em teclados e sensores de deslocamento. Placas Hall são facilmente integradas em silício. Um novo sensor que não utiliza sensores Hall é o sensor de campo magnético 3D, que mede a diferença de corrente entre coletores em um par de transistores quando expostos a um campo magnético.
- E, finalmente, sensores de silício para sinais químicos.

Muitos destes sensores são baseados em materiais que geram pequenas mudanças de corrente, tensão, resistência ou capacitância em resposta a um ou mais estímulos.

Além disso, o sinal de saída poderá ser fortemente afetado de forma indesejável por outros parâmetros, tais como temperatura ou pressão. Nível DC, sensibilidade cruzada, não linearidade e mudança de sensibilidade com o tempo também são observadas [DOE90].

Em geral, o sensor deve ter uma saída padronizada, onde os desvios indesejáveis são compensados. A calibração e o auto-teste devem ocorrem periodicamente. A forma de realizar este conceito é combinar, em um mesmo circuito integrado, um dispositivo sensor e um número de componentes micro-eletrônicos. Este conjunto é muitas vezes referido como sensor inteligente. Apesar de sensores inteligentes híbridos serem certamente de interesse, uma solução mais elegante é obtida quando o sensor e os circuitos de processamento de sinal são construídos do mesmo material, de tal forma que ambos componentes podem ser integrados em um mesmo substrato [HUI96].

Segundo J. M. Favennec [FAV87], um sensor inteligente é composto essencialmente por um transdutor, circuitos condicionadores, uma fonte de alimentação, capacidade de processamento interno, uma interface de comunicação e um identificador do sensor. Este princípio é representado na figura 2-1.

As características básicas que colocam os sensores inteligentes dentro de uma nova categoria de sensores podem ser resumidas como:

- Computação direta do valor físico medido (com as influências parasitas corrigidas) em nível de sensor (ao invés de correção em nível de computador central);
- Sinal digital ao invés de analógico;
- Identificação do sensor;
- Comunicação com uma rede ou computador central;
- Integração de todas as funções adicionais localmente;
- Possibilidade de auto-diagnósticos para operação apropriada e preparação de dados para manutenção.

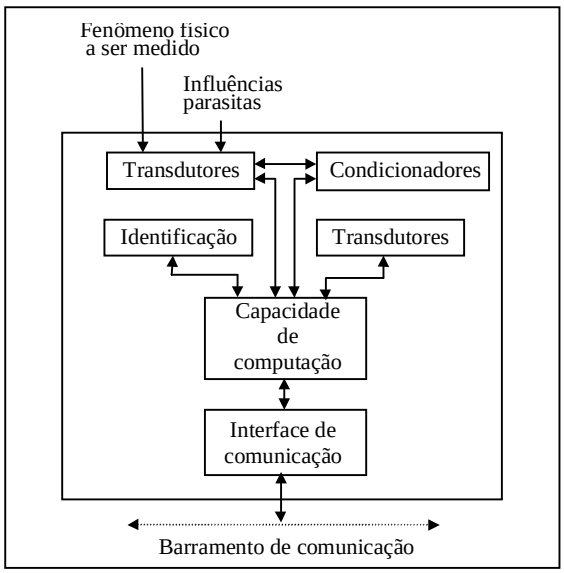

Figura 2-1 Princípio de um sensor inteligente, retirado de [FAV87].

Em [HUI94] o conceito de sensor inteligente integrado é novamente revisado sob o ponto de vista industrial. Com a revolução da automação, seguidas da revolução industrial e a da informação, a necessidade de sensores é muito grande. Os problemas associados a estes sensores também cresceram, tendo cada sensor suas próprias particularidades.

Segundo [HUI94], sensor integrado inteligente é definido como um circuito integrado, sem componentes externos, que inclua as seguintes funções:

- Sensoriamento através de um ou mais sensores;

- Interfaceamento, condicionamento de sinais, conversão analógico para digital, e padronização do formato do barramento de saída;
- Calibração: nível DC, escala, linearidade, sensibilidade cruzada;
- Inteligência: auto-teste, auto-identificação.

Este conceito pode ser visto na figura 2-2.

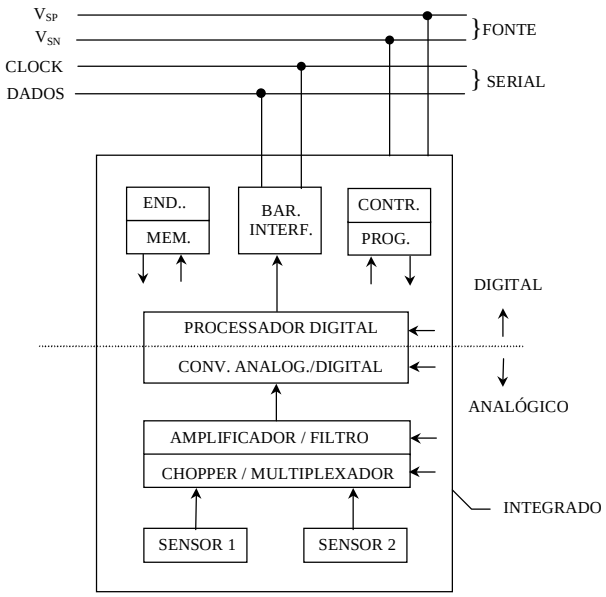

Figura 2-2 Funções de um sensor inteligente integrado, retirado de [HUI94]

A padronização dos sinais de saída do sensor implica em padronizar o ponto de zero, o fator de escala, a não-linearidade e a sensibilidade cruzada. A calibração de sensores contribui consideravelmente no custo do sensor. Muitas vezes, o custo de calibração excede os custos de manufatura primária.

A importância dos micro-sistemas integrados continuou crescendo devido à combinação de dois fatores: o progresso da tecnologia de sensores em silício e a introdução de novas técnicas de circuitos para o projeto de circuitos de interface. Outros exemplos da aplicação deste conceito são mostrados em [HOS97, HAB97, BAL96], onde a integração do condicionador de sinais e de funções de processamento pode melhorar a performance do sistema sensor, reduzir o efeito das não-idealidades e também facilitar a implementação de

algoritmos de processamento de sinais e funções de controle, teste, interfaceamento e monitoração.

Para demonstrar o projeto e as capacidades de modernos sistemas sensores de silício, foram apresentados três exemplos. Um sistema sensor de gás com leitura e cancelamento de desvios integrados, um sistema sensor de pressão baseado em diafragma de silício e quatro piezo-resistores e um sensor linear de imagem fabricado em tecnologia CMOS.

2.2 CARACTERÍSTICAS DA CURVA DE TRANSFERÊNCIA DE SENSORES

Como se viu na seção 2.1, as técnicas envolvidas na fabricação de sensores e circuitos de condicionamento podem, muitas vezes, devido às características não ideais e intrínsecas destes componentes, introduzir imperfeições ou erros na curva de calibração do sistema sensor.

Durante o processo de fabricação, estes sistemas sensores devem ser testados, individualmente ou por amostragem, de forma a garantir uma resposta pré-definida dentro de uma certa precisão [HOR97- DOE90]. Dentre os principais desvios encontrados na curva de calibração de sensores durante o processo de teste estão :

- **Nível DC (*off-set*):** ocorre quando uma grandeza física de entrada nula (ou mínima) é aplicada e o valor de saída medido não é nulo, ou seja, apresenta um nível DC (figura 2-3a).
- **Ganho:** ocorre quando a sensibilidade do sensor não é a desejada, um valor máximo da grandeza física de entrada não corresponde a um valor máximo do sinal elétrico de saída (figura 2-3a).
- **Não linearidade:** ocorre quando valor de saída do sensor não varia linearmente com o valor de entrada da grandeza física medida (figura 2-3a).
- **Sensibilidade cruzada:** a curva de calibração do sensor varia quando medida em diferentes condições ambientais (temperatura, por exemplo), de modo que o sensor não é somente sensível ao sinal de entrada, mas também a outros parâmetros (figura 2-3b).

- **Histerese:** ocorre quando a curva de calibração do sensor apresenta valores diferentes para valores crescentes e decrescentes do sinal físico de entrada (figura 2-3c).
- **Desvio (*Drift*):** ocorre quando a curva de calibração do sensor muda lentamente com o decorrer do tempo (figura 2-3d).
- **Comportamento dinâmico:** devido a co-existência de elementos dissipativos, armazenadores e inerciais, o elemento sensor pode apresentar diferentes comportamentos da sua curva de resposta, dependo da característica em freqüência do sinal de entrada.

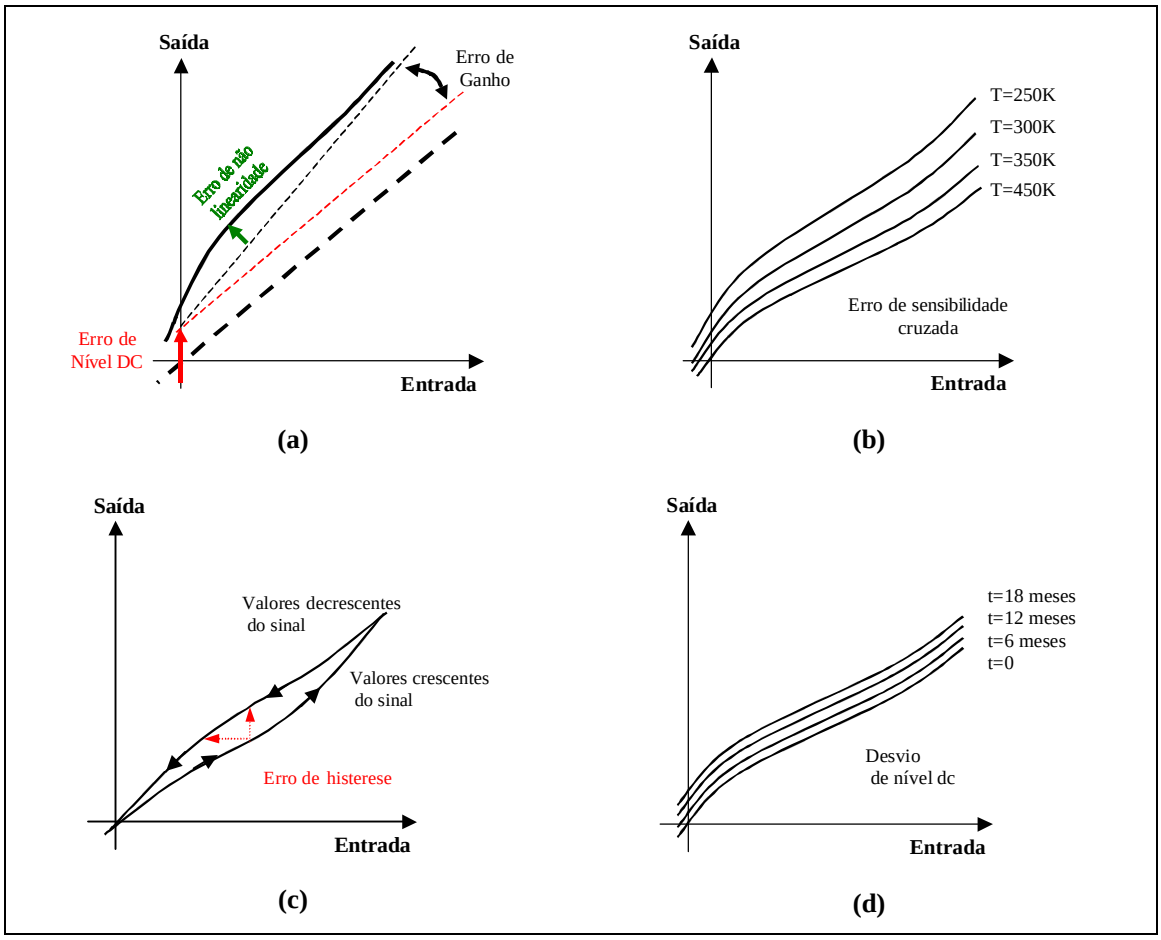

Figura 2-3 Possíveis erros na curva de calibração de um sensor.

Na figura 2-3 os principais erros da curva de calibração são explicados graficamente. Geralmente estes erros são maiores do que permitido pela precisão desejada. Para corrigir estes erros é necessário que haja uma calibração individual.

Os erros de ganho, de nível DC e de não linearidade são observados em quase todos os sensores, e por isso os métodos calibração estudados neste trabalho tem como enfoque a correção destes erros, que serão abordados na próxima seção. Erros de histerese e desvio de

sensibilidade com o tempo são observados em somente alguns sensores e fogem do escopo deste trabalho.

2.3 MÉTODOS DIGITAIS DE LINEARIZAÇÃO

A calibração de sistemas sensores pode ser feita utilizando-se tanto tecnologia analógica, quanto digital. Na figura 2-4 tem-se um exemplo onde a correção de não linearidade de um sensor de temperatura é realizada utilizando-se um circuito analógico. O sinal de saída do sensor possui uma resposta do tipo logarítmica. Após este sinal, vindo do sensor, passar por circuito de analógico que possui uma resposta do tipo exponencial, obtém-se um sinal de saída corrigido.

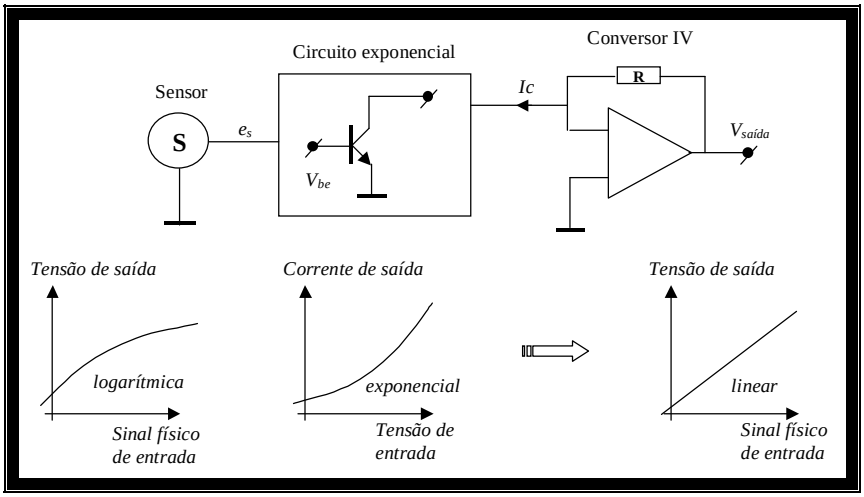

Figura 2-4 Linearização de uma curva de calibração tipicamente logarítmica, retirado de [HOL97]

Ao se utilizarem métodos analógicos, nem sempre será possível encontrar-se um circuito que integre totalmente a função inversa do sensor que se deseja corrigir [HOR97], fazendo que a compensação, quando realizada, não alcance a resolução desejada. Em contrapartida, os métodos digitais de calibração, apesar de consumirem maior área quando implementados, suportam uma grande variação no tipo da função de transferência. Além disso, circuitos digitais estão em constante desenvolvimento e podem oferecer uma gama maior de vantagens no contexto de sensores inteligentes. Tarefas como auto-calibração e auto-teste são geralmente mais facilmente implementadas, diminuindo custos de produção e teste. Facilidades de integração e automação, por outro lado, diminuem os custos de projeto,

fazendo dos métodos digitais uma excelente solução. Por estes motivos, métodos digitais são de grande importância e serão abordados neste trabalho.

Além disso, a calibração e linearização de sensores utilizando tecnologia digital tem sido abordada sistematicamente pelo meio científico e industrial, tendo-se um número significativo de trabalhos publicados. A partir deles, como será visto a seguir, pode-se ter uma idéia geral do que foi feito até os dias de hoje, e das diversas técnicas empregadas na compensação digital.

O método mais simples de linearização é o método Tabela, onde os pontos de calibração são armazenados em uma ROM [BRI96]. A saída do sensor é conectada a um conversor A/D, e o sinal digital é usado para endereçar a memória de calibração. Desta forma, tem-se um sinal de saída corrigido. Este método é simples e rápido, mas a área utilizada cresce exponencialmente com o número de bits requerido pela aplicação, uma vez que para cada bit extra de resolução, dobra-se o espaço requerido para o endereçamento.

O método Tabela, aqui referido como LUT (*Look-Up Table*), tem seu diagrama em blocos, apresentando o processo de linearização, na figura 2-5. O sinal de entrada não calibrado f(x) vindo do sensor é convertido para o domínio digital por um conversor A/D. O sinal quantizado f[n] endereça a memória, a qual fornece um sinal de saída calibrado g[n]. As vantagens do método são simplicidade e velocidade, mas a principal desvantagem é a área usada pela memória. Devido a sua simplicidade, o método Tabela pode ser considerado como o método de referência.

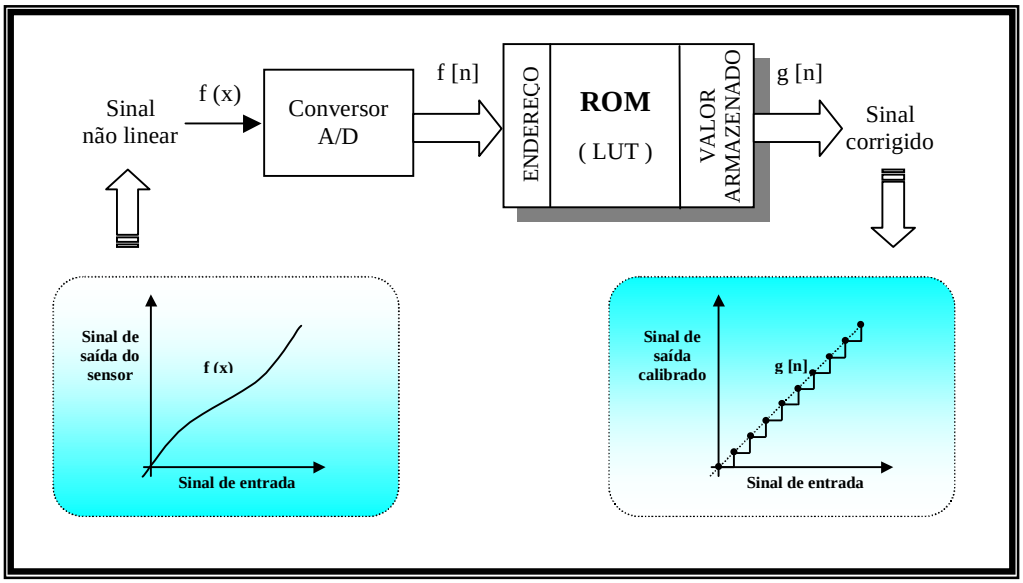

Figura 2-5 Método de linearização Tabela (LUT).

M. P. Kraska [KRA88] utilizou esta técnica na linearização digital de sensores em sistemas de controle automotivos. Além disso, apresentou uma revisão sobre os métodos convencionais de linearização empregados, suas vantagens e desvantagens. A técnica empregada na linearização de um sinal analógico não linear vindo de um sensor utiliza um conversor A/D, uma memória do tipo EPROM e um conversor D/A. A EPROM contém os dados de calibração do sensor a ser corrigido os quais são endereçados pelo conversor A/D. Como resultado, a EPROM fornece um sinal digital corrigido, podendo ser novamente convertido para o domínio analógico pelo conversor D/A utilizado para a visualização do sinal por um indicador analógico. Etapas de filtragem são realizadas antes do sinal ser amostrado para evitar o fenômeno de aliasing, e após a conversão D/A para eliminar as componentes harmônicas introduzidas no sinal devido à quantização.

Quando os requerimentos de velocidade de execução podem ser relaxados, área pode ser economizada em troca de operações processadas digitalmente. Exemplos de tal compromisso podem ser vistos em [MAL94], onde características inerentes a conversores sigma-delta são usadas para interpolar um conjunto limitado de pontos de calibração, ou em [HOR97], que usa calibração polinomial progressiva, e em [JAH98-HOL98], onde filtragem adaptativa é usada para melhorar a resposta dinâmica de um sensor.

Em 1994, Malcovati *et al* [MAL94] apresentaram uma interface para sensor inteligente com conversão A/D e calibração programável. Este método de calibração, aqui chamado de método Inserido no Conversor, é uma variação do método Tabela.

Com o objetivo de reduzir o tamanho da memória usada para armazenar os coeficientes de calibração do sensor, a tabela de calibração foi inserida no meio da etapa de filtragem do conversor, como mostra a figura 2-6. Alguns ciclos extras de processamento são requeridos em troca de área consumida, e a memória deve operar na freqüência de sobre-amostragem do conversor. Obviamente, este método é limitado a conversores do tipo sigma-delta [AZI96].

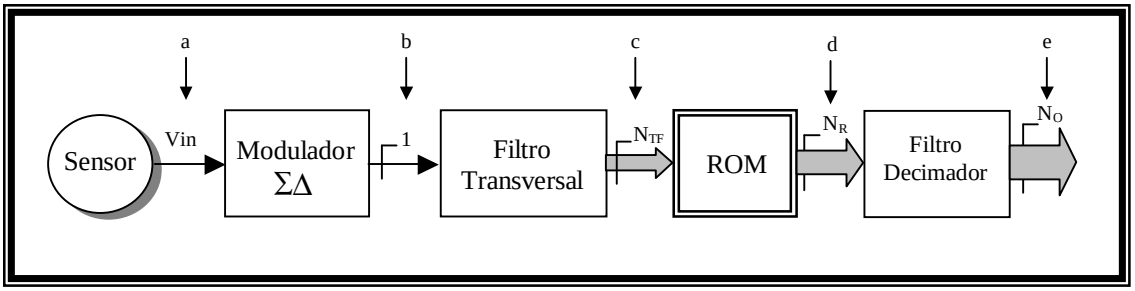

Figura 2-6 Método de linearização Inserido no Conversor.

A topologia de um conversor A/D sigma-delta é composta basicamente por duas etapas principais: Modulação e filtragem. Na primeira etapa o modulador sigma-delta transforma o sinal entrada em um fluxo de bits com freqüência de sobre-amostragem f_s [AZI96]. Ou seja, este fluxo de bits na saída do modulador é uma representação digital do sinal de entrada sob a forma de zeros e uns. A segunda etapa normalmente é um filtro passa-baixas do tipo SINC com decimação, o qual tem por finalidade filtrar o fluxo de bits de forma a eliminar o ruído de quantização inserido pelo modulador, recuperar o sinal de entrada e baixar a freqüência de sobre-amostragem.

O ganho do método Inserido no Conversor está em trazer a ROM para dentro da etapa de filtragem do conversor A/D sigma-delta. Para isso é necessário inserir-se um novo filtro entre o modulador e o filtro decimador. Este novo filtro é um filtro SINC transversal de baixa ordem que trabalha na freqüência de sobre-amostragem, a qual apesar de ser alta, é ainda razoável para circuitos digitais. Desta forma, o sinal de entrada V_{in}, vindo do sensor (figura 2-6 – ponto a) é transformado pelo modulador sigma delta em um fluxo de bits (ponto b). O fluxo de bits passa, então, pelo filtro SINC transversal do tipo média móvel, que devido à fraca filtragem realizada, é transformado em um fluxo de palavras digitais N_{TF} (ponto c) que pode ser quantizado com um número reduzido de bits. O número de bits desta palavra digital é que define o número de endereços de entrada da tabela (ROM), que será utilizada para corrigir a forma de onda do fluxo de palavras. Após a tabela ser aplicada no sinal N_{TF}, tem-se um sinal compensado N_R (ponto d) o qual apresenta, ainda, uma relação sinal-ruído muito baixa devido às componentes espectrais de alta freqüência inseridas pelo processo de quantização do modulador sigma-delta. Finalmente, na última etapa o sinal N_R é levado então a etapa de filtragem com decimação, tendo-se como saída o sinal de entrada corrigido N_O (ponto e) e na taxa de Nyquist. Com a topologia da figura 2-6 conseguiu-se estabelecer um compromisso entre resolução e complexidade de hardware.

Conforme reportado em [MAL94], além de simulações em Matlab5.3 foi implementado um protótipo, os quais demonstraram que na comparação direta com o método convencional, consegue-se reduzir consideravelmente o tamanho da tabela (ROM), mantendo-se a mesma resolução. No protótipo, implementou-se um conversor A/D de 12 bits utilizando um modulador sigma-delta de 2ª ordem seguido de um filtro SINC transversal sem decimação de 6 bits. A ROM utilizada para armazenar os dados de calibração é do tipo 6 bits de endereços por 12 bits de dados (768 bits). O filtro de saída é um filtro SINC de 2ª ordem com decimação.

No método convencional, para manter a mesma resolução, seria necessária uma ROM com 12 bits de endereços por 12 bits de dados perfazendo um total de 49,152 kbits. Portanto, com a topologia Inserido no Conversor consegue-se reduzir em 64 vezes o tamanho da ROM que contém o dados de calibração. Porém, a área utilizada pelo filtro SINC decimador aumenta, uma vez que é filtrada uma palavra digital ao invés de um fluxo de bits, ficando a solução final com uma redução de área de 2 a 5 vezes. Uma discussão mais precisa sobre esta comparação será feita nos capítulos 3 e 4.

G. van der Horn e J. H. Huijsing, em 1997, apresentaram outro método de linearização de sensores utilizado na calibração de sensores inteligentes [HOR97]. Este método foi criado com a finalidade de aumentar a performance e a confiabilidade de sistemas de medição.

Uma vez que o processo de calibração convencional de sensores é um processo demorado e que aumenta consideravelmente os custos de produção, pois requer ajustes manuais e individuais durante a aplicação de um grande número de estímulos de entrada, esta solução integrada é proposta para minimizar estes problemas. Um método de calibração e opções para integração no conceito de sensor inteligente, tanto em hardware bem como em software são propostos. Uma vantagem do método proposto em [HOR97] é que este não necessita de uma grande matriz de dados de calibração para serem armazenados em uma tabela, ou convertida em uma fórmula de conversão.

Este método é executado através de um algoritmo iterativo, trocando área consumida por tempo de execução, quando comparado ao método Tabela. Em uma primeira etapa, os coeficientes de calibração são encontrados, similarmente a um filtro IIR, e na etapa seguinte tem-se a fase de funcionamento em regime. O sinal de referência é um sensor com curva de calibração ideal, e a calibração é feita de forma progressiva.

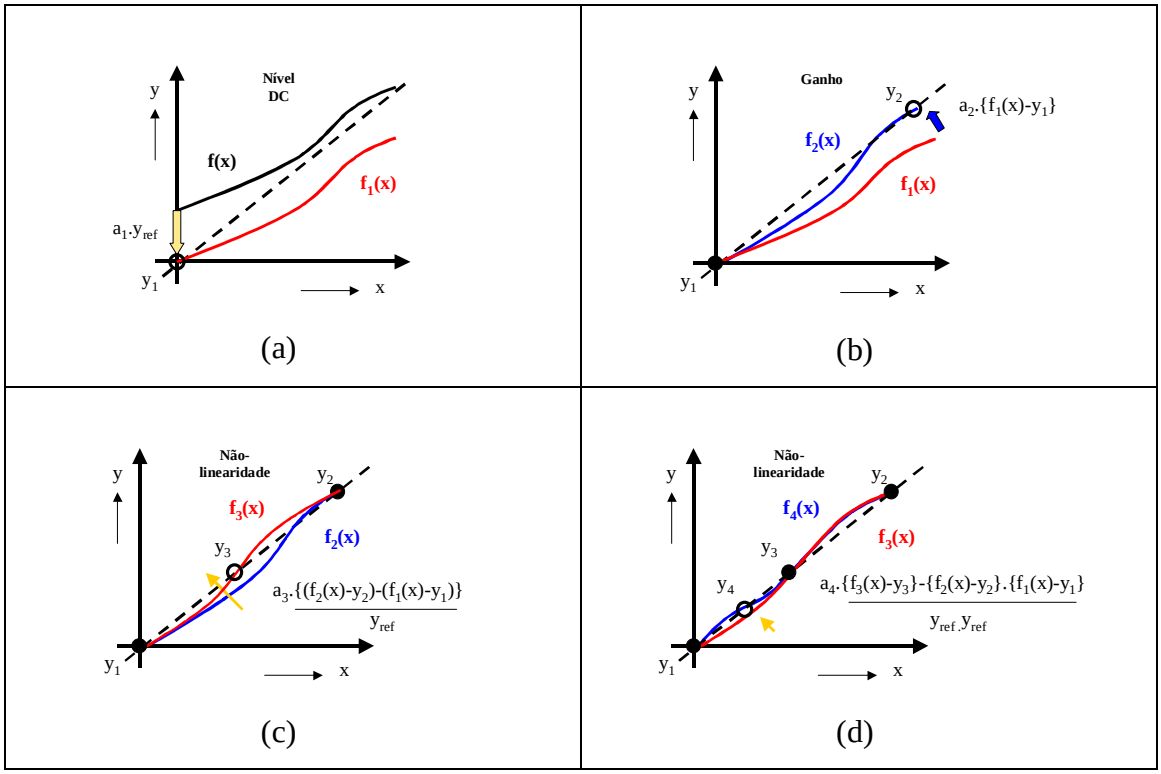

Figura 2-7 Passos de calibração, método Polinomial.

Na figura 2-7 pode-se se ver graficamente como ocorre o processo de calibração. Primeiramente é eliminado o nível DC da curva do sensor através de uma operação de translação (figura 2-7a). Após esta primeira etapa, fixa-se o primeiro ponto de calibração e executa-se uma operação de rotação, de forma a corrigir o erro de ganho da função não linear (figura 2-7b). Com estes dois pontos de calibração fixos, um terceiro e outros possíveis passos de calibração são executados, os quais consistem em deflexionar a curva em direção a reta ideal, ajustando desta forma todos os pontos intermediários (figuras 2-7c e d).

Juntamente com o erro de não-linearidade são corrigidos os erros de ganho e nível DC, podendo também ser implementada calibração bi-dimensional, corrigindo-se dessa forma erros devido à sensibilidade cruzada, como temperatura.

Baseado na predição adaptativa de valores amostrados para transdutores digitais, Hölling, Thaler e Tröster, em 1998 [HOL98], apresentaram um método para diminuir requerimentos dinâmicos em sensores de corrente, utilizando predição adaptativa do próximo valor amostrado. O filtro adaptativo utiliza o algoritmo LMS (*Least Mean Square*). Aplicado a sinais amplamente estacionários e periódicos, o método necessita de um período de aprendizado para conseguir resultados precisos. Utiliza também um conversor D/A, para gerar um sinal de realimentação que corrige o sinal de entrada.

Também em 1998, Jahn e Carro [JAH98] apresentaram uma nova topologia para compensação digital de não-linearidades em circuitos analógicos, baseada em um filtro adaptativo não-linear. Além de linearizar as características VxI do sensor, o método compensa também as etapas de amplificação e filtro anti-alias. Uma das principais vantagens do método é o aumento do desempenho de circuitos analógicos construídos com tecnologia digital CMOS.

O método de JAH[98] utiliza filtros FIR (*Finite Impulse Response*) com coeficientes lineares e não lineares associado ao algoritmo LMS para encontrar os coeficientes que melhor corrigem a função de entrada. Na figura 2-8 tem-se o esquema de linearização. Na etapa de treinamento (área cinza), o sensor é excitado utilizando-se ruído branco. Após a convergência para o menor erro esperado, tem-se o funcionamento em regime permanente. Nesta implementação foram utilizados 10 coeficientes para o filtro linear (x) e 4 coeficientes para os filtros não lineares (ao quadrado e ao cubo).

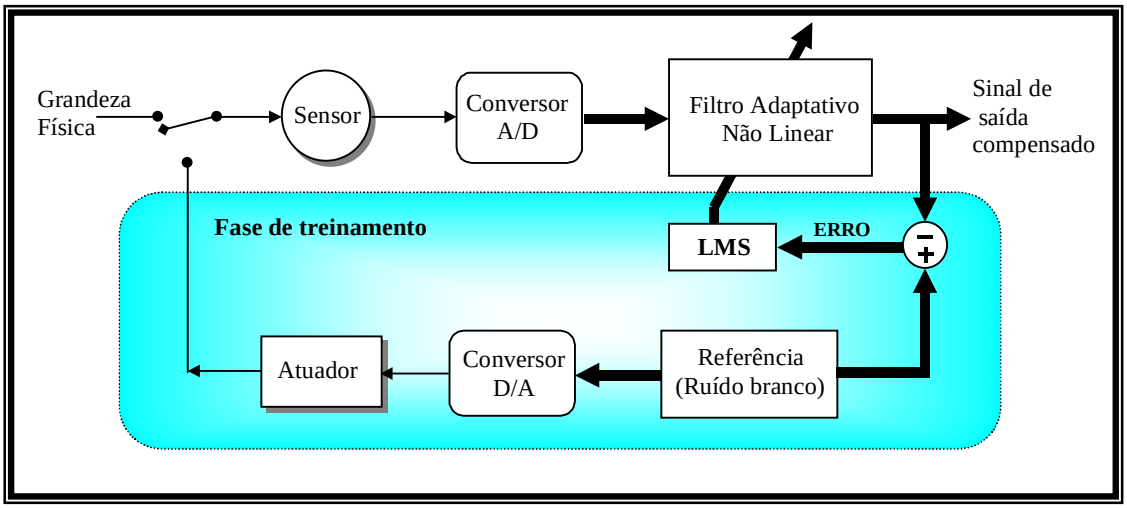

Figura 2-8 Método com Filtragem Adaptativa.

Em 2000, Negreiros e Carro publicaram um artigo sobre medidores de qualidade de energia de alta linearidade no ambiente industrial [NEG00], onde resultados preliminares de projeto foram apresentados. O protótipo de instrumento produzido neste trabalho é capaz de fazer leituras de tensão, corrente e potências trifásicas, bem como medidas de componentes harmônicas do sinal. Os cálculos são realizados por um DSP, o qual é usado também para realizar a correção de não-linearidades de um transdutor, aumentando desta forma a faixa dinâmica e a linearidade do instrumento de medição. O método de linearização é realizado no

domínio freqüência, baseando-se no cálculo da transformada rápida de Fourier (FFT) do sinal. O esquema de medição está apresentado na figura 2-9.

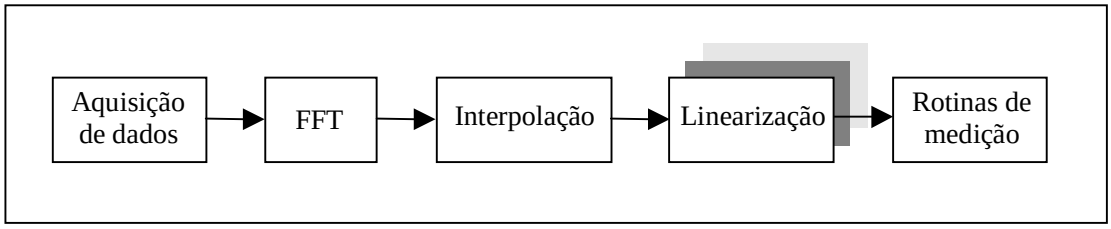

Figura 2-9 Esquema de medição método multi-tabelas.

Em [SCH96], três métodos de linearização de transdutores eletro-dinâmicos foram implementados em um DSP, e comparados em relação a eficiência, complexidade computacional e efetividade. Entretanto, critérios como potência dissipada área consumida e velocidade máxima de execução não foram diretamente discutidos.

Apesar de todos os métodos acima e outros poderem alcançar uma certa linearização requerida por uma aplicação definida, a seleção de um ou outro ainda não é clara, dado o número de diferentes parâmetros a serem analisados, como área, potência e tempo de execução.

Quatro dos métodos de linearização discutidos neste trabalho foram escolhidos para serem implementados e comparados, devido à sua generalidade (sendo aplicáveis a qualquer sistema sensor, independente do tipo de sensor ou interface), e devido à sua ligação com a técnica de processamento digital de sinais. Na era de sistemas embarcados e sistema integrados, grande parte das tarefas de linearização do sensor será realizada no domínio digital, e não mais no domínio analógico.

3 Metodologia de Comparação

Uma vez que se espera que um sensor inteligente seja um sub-sistema em um sistema maior, dentro de um circuito integrado onde possivelmente um microprocessador estará presente, cada algoritmo foi implementado como software a ser executado em um processador de sinais digitais. Para a realização dos experimentos utilizou-se o kit de desenvolvimento ADSP-2181 [ADS95].

A escolha deste kit de desenvolvimento em particular deve-se ao fato dele possuir características adequadas ao processamento de sinais digitais, tais como capacidade de processamento paralelo, alta velocidade de processamento (ciclo de máquina igual a 30ns), área de memória suficiente para armazenar os dados a serem analisados após a execução dos algoritmos e por ser facilmente obtido no mercado nacional.

Quatro dos métodos estudados no capítulo 2 foram escolhidos para serem implementados e posteriormente comparados quanto aos requisitos de área, potência, velocidade de operação e outros. São eles:

- Método Tabela;
- Método Inserido no Conversor;
- Método Polinomial;
- Método Adaptativo.

As interfaces de entrada utilizadas durante os experimentos, e o parâmetros utilizados nas comparações realizadas entre os métodos, serão apresentados nas seções 3.1 e 3.2.

3.1 INTERFACE DE ENTRADA

Dentre os quatro métodos estudados e comparados, o método proposto por Malcovati *et al* [MAL94] requer a utilização de um conversor A/D do tipo sigma-delta [AZI96], enquanto que os outros três métodos podem ser utilizados com qualquer outro tipo de conversor ou interface de entrada.

Esta limitação do método apresentado por [MAL94] conduziu a duas formas de implementação. A fim de desenvolver uma comparação justa, as implementações e comparações foram realizadas em duas etapas. Na primeira, os métodos Tabela [BRI96], Inserido no Conversor [MAL94] e Polinomial [HOR97] são implementados com o mesmo modulador sigma delta, sendo o conversor completo realizado dentro do DSP, que é responsável pela filtragem digital e pela decimação.

Numa segunda etapa, os métodos Polinomial [HOR97] e com Filtro Adaptativo [JAH98] foram implementados utilizando-se um conversor A/D genérico. No caso, utilizou-se o conversor A/D do kit de desenvolvimento. Note-se que nesta nova topologia o DSP executa apenas o algoritmo de linearização, e não mais tarefas referentes ao conversor, como no caso anterior.

A seguir, nas seções 3.1.1 e 3.1.2, são apresentadas as interfaces de entrada utilizadas em cada etapa de implementações e comparações.

3.1.1 Interface de Entrada com Modulador Sigma-delta

A interface de entrada utilizada na primeira etapa de implementações requer um modulador sigma-delta analógico.

A principal restrição encontrada na escolha do modulador a ser utilizado recaiu sobre a sua freqüência de sobre amostragem, que tem de ser baixa o suficiente para permitir que o DSP execute as tarefas de filtragem, decimação e linearização, mas que permita também uma banda de freqüências de entrada larga o suficiente para aplicações em sistemas sensores embarcados.

Como o ciclo de máquina do DSP escolhido é de 30ns, e cada instrução é realizada em um ciclo de máquina, este valor pode ser utilizado para determinar a freqüência de sobre-amostragem do modulador.

O relógio do modulador funciona a uma freqüência de sobre amostragem F_s. A cada período de sobre amostragem $T_s = 1/F_s$ segundos, uma interrupção de leitura é recebida pelo DSP, vinda do modulador. O DSP deve, antes de receber uma nova interrupção, executar todas as tarefas de filtragem e linearização. Ao tempo necessário para executar todos os algoritmos denomina-se tempo de processamento, t_p.

Para que sistema funcione corretamente, o tempo de processamento deve ser menor do que o período de sobre amostragem, ou seja $T_s > t_p$.

O tempo de processamento depende do número de instruções (N) e do tempo de ciclo de máquina do processador (30ns). Como

$$t_p = N*30ns \quad (3\text{-}1)$$

Então,

$$T_s > N*30ns \quad (3\text{-}2)$$

O modulador sigma-delta utilizado neste trabalho foi implementado por [JAH99] e tem as seguintes características:

- Excursão do sinal de entrada: 0 a 2,8V;
- Freqüência de sobre-amostragem: 512 kHz
- Taxa de decimação: 256;
- Banda do sinal de entrada: 2 kHz
- Resolução: 8 bits.

Note-se que para uma freqüência de sobre-amostragem de 512 kHz, conforme a inequação 3-2, tem-se um número de instruções N do DSP igual a:

$$1/512kHz > N*30ns$$

$$N < 65 \text{ instruções} \quad (3\text{-}3)$$

Entretanto, verificou-se que 65 instruções são insuficientes para a implementação dos algoritmos de cada método. Por este motivo optou-se por diminuir a freqüência de sobre-amostragem para 256 kHz.

Para o modulador sigma-delta com freqüência de sobre amostragem de 256kHz, tem-se:

$T_s = 1/F_s = 1/256kHz$

Logo,

$$1/F_s > N*30ns$$
$$1/256kHz > N*30ns \quad (3\text{-}4)$$
$$N < 1/(256kHz*30ns)$$
$$N < 130 \text{ instruções}$$

Com este número de instruções (N=130) consegue-se implementar todos os algoritmos propostos, e a freqüência de sobre amostragem é alta o suficiente para garantir uma relação sinal-ruído (SNR) adequada para um conversor A/D de 8 bits com largura de banda de entrada de 1000Hz.

Uma vez definida a interface de entrada, todos os algoritmos foram simulados em MatLab5.3, posteriormente implementados em DSP e executados em laboratório. Para testar a eficiência de linearização de cada método, utilizou-se um sinal de entrada senoidal (x), que representa o sinal padrão para análise da não linearidade do sensor. Este sinal senoidal é aplicado à interface de entrada, composta pelo circuito de condicionamento e sensor, e que foi modelada como uma função não linear do tipo:

$$f(x) = -0.196 + x + 0.1x^2 \quad (3\text{-}5)$$

Como resultado, obtém-se um sinal não linear que é lido pelo conversor A/D e linearizado. Outras funções não lineares poderiam ter sido utilizadas, uma vez que os métodos são genéricos. Optou-se pela função da equação 3-5 por esta já ter sido utilizada de forma similar em [MAL94], o que facilita comparação entre resultados obtidos.

Como o objetivo deste trabalho é comparar os diferentes métodos e não caracterizá-los, a escolha da função de entrada tem importância secundária e a função escolhida (3-5) preenche os requisitos necessários, por apresentar características facilmente encontradas em sensores, como nível DC, desvio de ganho e não-linearidade.

O hardware utilizado nesta etapa de implementações é visto na figura 3-1, sendo composto de um gerador de sinais HP33120A, um osciloscópio digital HP54645D, um modulador sigma-delta e um kit de desenvolvimento ADSP-2181.

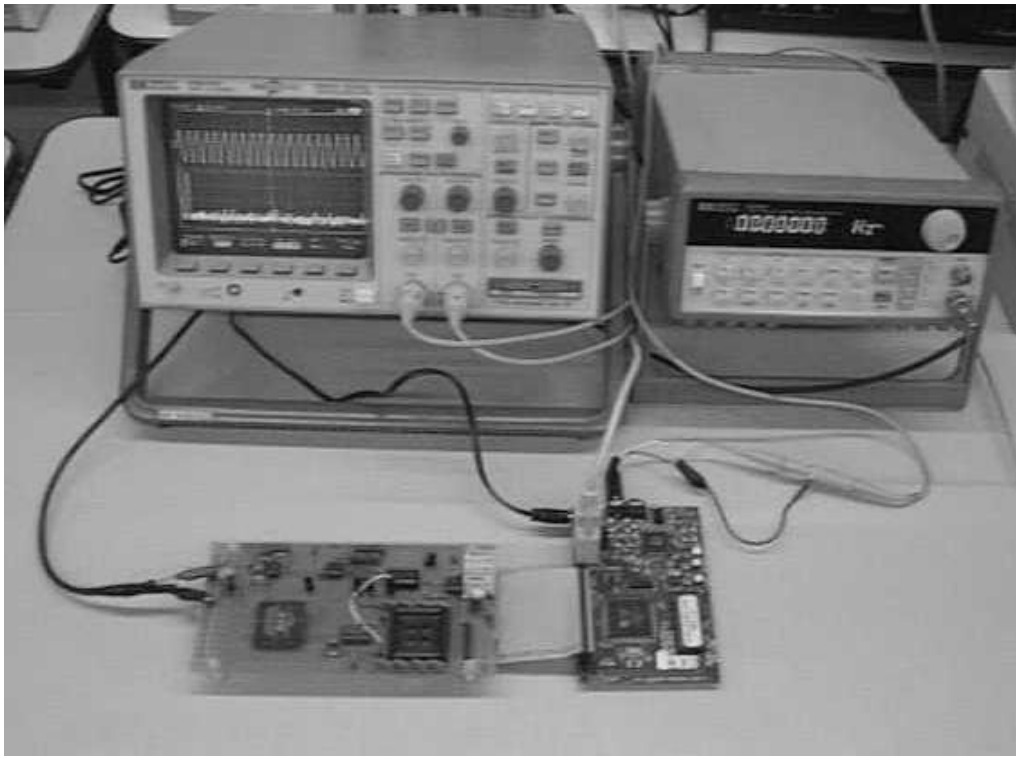

Figura 3-1 Esquema de hardware utilizado para implementação em laboratório na primeira fase de comparações.

Para os experimentos utilizando o DSP, uma vez definida a curva de calibração com característica não linear que representa a resposta do conjunto sensor e circuito de condicionamento, efetuou-se a programação desta curva no gerador de sinais HP33120A. Desta forma, o gerador fornece um sinal com característica não-linear que é aplicado ao conjunto conversor A/D e DSP.

A amplitude do sinal de entrada senoidal foi escolhida de forma a atingir o fundo de escala do conversor A/D, ou seja 1,4 Vp. A freqüência deste sinal de entrada deve estar na banda de passagem do conversor, que é de 0 a 500Hz, seguindo o critério de Nyquist [OPP89]. Para obter-se um número maior de pontos por ciclo de sinal de entrada, utilizou-se um sinal de baixa freqüência de entrada, 17Hz.

As curvas de calibração, com características ideal e não linear, são apresentados na figura 3-2. Note-se que a curva ideal, linha tracejada, é uma linha reta com declividade igual a

1, enquanto a com resposta não ideal, linha cheia, possui um comportamento não linear, com a presença dos erros de linearidade, de nível DC e variação de ganho.

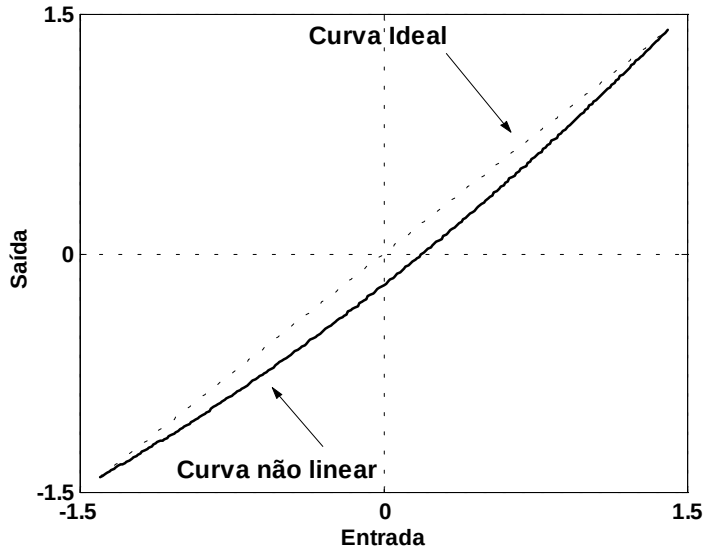

Figura 3-2 Curvas de calibração ideal e não-linear.

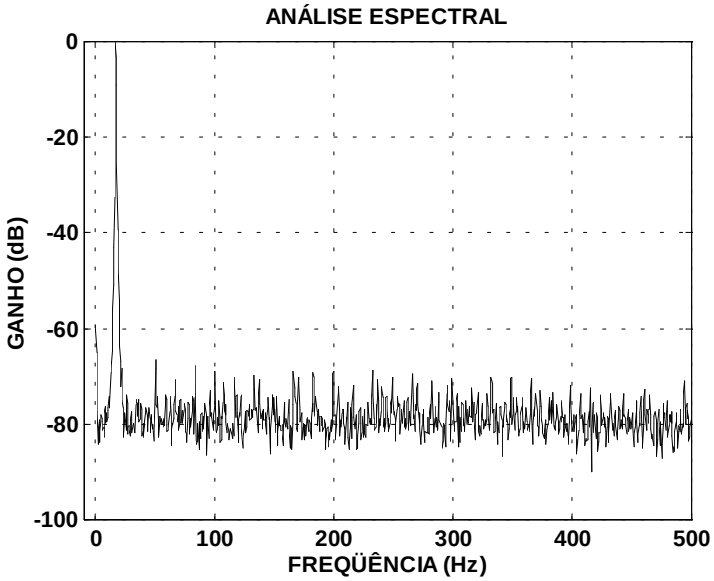

Figura 3-3 FFT do sinal de entrada gerado a partir de uma curva de calibração ideal.

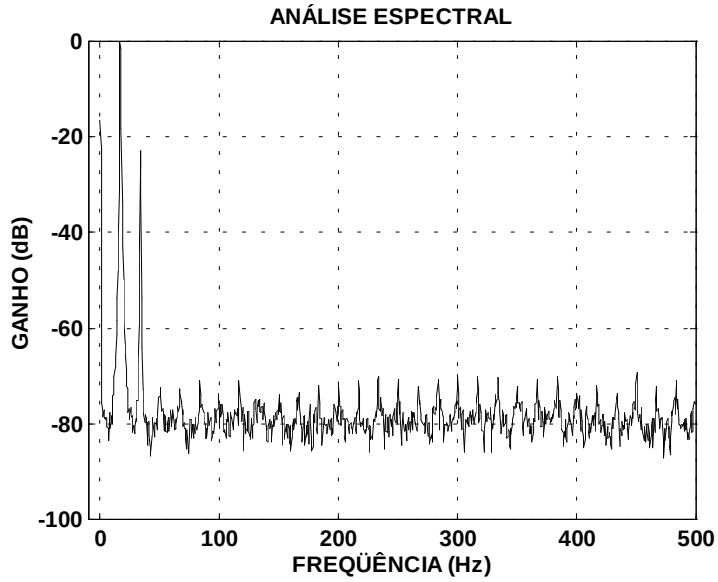

Figura 3-4 FFT do sinal de entrada gerado a partir de uma curva de calibração com característica não linear.

Nas figuras 3-3 e 3-4 tem-se os espectros de freqüências para um sinal de entrada senoidal. Realizou-se uma FFT de 2048 pontos com janela de Hanning de mesmo tamanho. Na primeira, figura 3-3, tem-se a FFT do sinal senoidal produzido pelo gerador de sinais, cuja curva de calibração tem caracterísca ideal, e que representa neste trabalho um sensor com resposta linear. Note-se que além da componente fundamental de 17 Hz , tem-se o ruído de quantização com uma atenuação de aproximadamente 66 dB e nível DC de 60 dB. Na segunda, figura 3-4, tem-se a FFT do sinal senoidal produzido pelo gerador de sinais , cuja curva de calibração tem característica não-linear dada pela equação 3-5, e que representa neste trabalho um sensor não ideal. Percebe-se, claramente, a inclusão de uma componente harmônica com atenuação de 22 dB, e a degradação do nível DC para −17dB.

3.1.2 Interface de entrada com conversor A/D genérico

Nesta etapa, os métodos comparados são o Polinomial [HOR97] e o com filtro adaptativo não linear [JAH98]. A interface de entrada utilizada é a que está inserida na placa de desenvolvimento do DSP. Composta de um circuito eletrônico condicionador de sinal e um conversor A/D com resolução de 16 bits ou 8 bits, configurada por software. Nas implementações realizadas, utilizou-se o conversor em modo de operação de 8 bits, com uma freqüência de amostragem de 8kHz.

Na figura 3-5 tem-se o esquema de hardware utilizado para executar os algoritmos de linearização, da segunda etapa de comparações, em laboratório. Note-se que em relação à figura 3-1 apenas eliminou-se o modulador sigma-delta, uma vez que utilizou-se o conversor A/D do próprio kit de desenvolvimento.

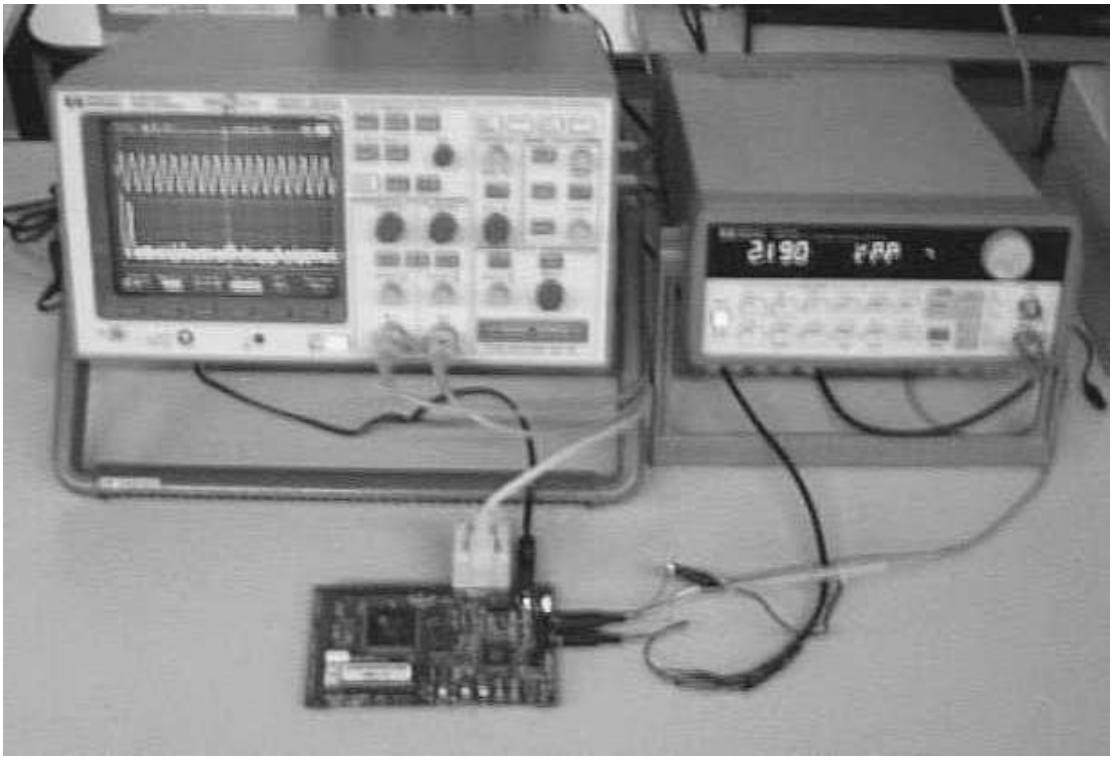

Figura 3-5 Esquema de hardware utilizado em laboratório na segunda etapa de comparação.

Nas implementações Matlab5.3, inicialmente, utilizou-se a mesma função de transferência apresentada no capítulo anterior, equação 3-5, para modelar o sistema sensor. Entretanto, o algoritmo utilizando filtros adaptativos não lineares apresentou dificuldades em convergir para um valor de erro adequado, quando comparado com os outros métodos.

Isto se deveu, basicamente, ao fato de filtros adaptativos serem aplicados a sinais estacionários de média zero, portanto, sem nível DC [WID85].

Uma outra dificuldade encontrada foi o fato de não se ter acesso ao circuito condicionador de sinais da placa de desenvolvimento, já que este circuito elimina o nível DC do sinal de entrada.

Desta forma, optou-se por, nesta segunda etapa de comparação, alterar a função de teste retirando o nível DC da mesma. Assim, a nova função de teste utilizada é:

$$g(x) = x + 0.1x^2 \qquad (3\text{-}6)$$

A banda de interesse do sinal de entrada (x) deve estar dentro da faixa de 0 a 4kHz (critério de Nyquist). Para fins de teste utilizou-se um sinal de entrada senoidal com amplitude 1.2 Vp e freqüência de 100 Hz. Desta forma, a excursão em amplitude do sinal g(x), sinal a ser corrigido, deve ficar próxima ao fundo de escala de leitura do conversor A/D (1,4Vp). Na figura 3-6 tem-se as curvas de calibração para uma sensor com resposta ideal e não linear, considerando-se uma faixa de operação entre ± 1,4Volts do sinal de entrada.

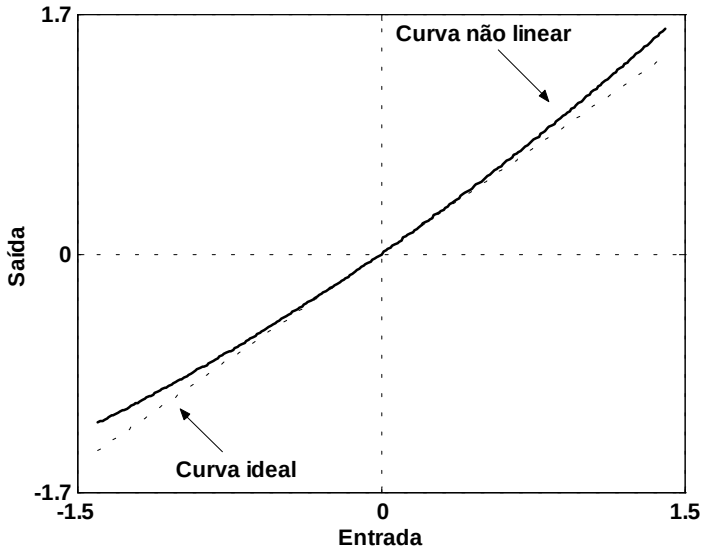

Figura 3-6 Curvas de calibração ideal e não-linear.

Nas figuras 3-7 e 3-8 tem-se a análise espectral para um sinal senoidal (x) obtido a partir do gerador de sinais programado para ter resposta linear, e não linear conforme à equação 3-6. Note-se que na figura 3-7 (FFT do sinal ideal), além do ruído de quantização e do nível DC que possuem atenuação próxima a 60 dB, tem-se somente a componente fundamental do sinal de entrada com freqüência de 100 Hz. Já na figura 3-8, além da componente fundamental do sinal de entrada, a função g(x) introduziu uma distorção harmônica com freqüência de 200 Hz, que possui atenuação de 27 dB e degradação do nível DC para -22 dB.

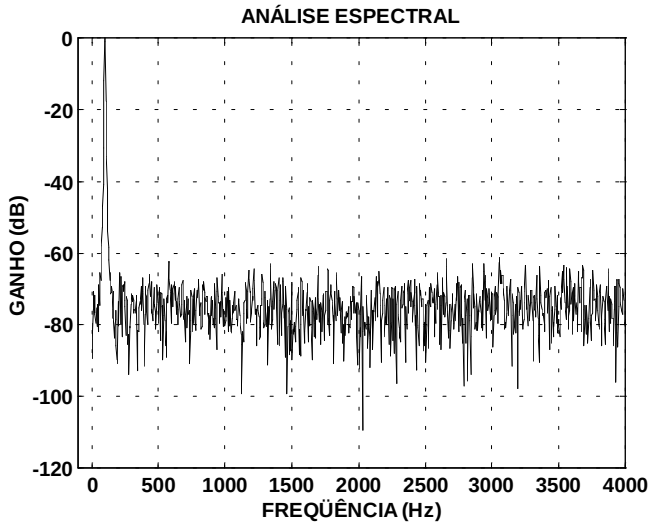

Figura 3-7 FFT do sinal de entrada gerado a partir de uma curva de calibração ideal.

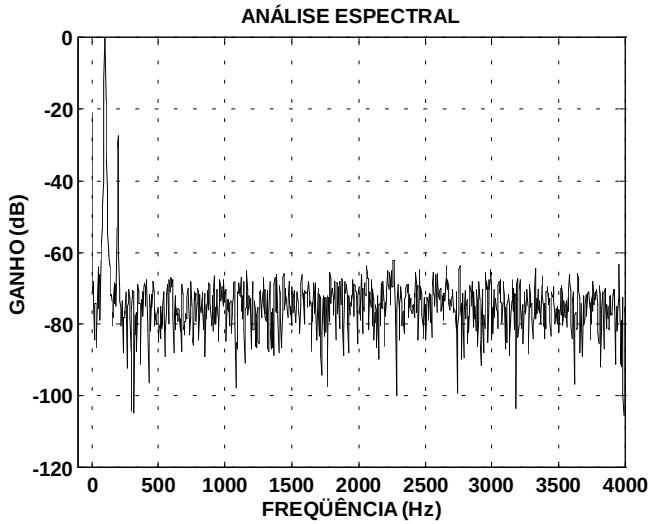

Figura 3-8 FFT do sinal de entrada gerado a partir de uma curva de calibração não linear.

3.2 PARÂMETROS DE COMPARAÇÃO

Uma vez definidas as interfaces de entrada e as funções de estímulo a serem utilizadas, passa-se à definição dos parâmetros a serem utilizados para comparar os diversos métodos de linearização.

Dentro do contexto de sistemas embarcados, uma variedade de fatores pode influir na escolha, por parte do projetista, de qual método de linearização utilizar. Esta escolha deve-se basear na otimização dos requisitos de hardware e software disponíveis no sistema a que estes

métodos serão inseridos, de forma a se obter performance adequada e redução nos diversos custos envolvidos.

Um dos parâmetros usados na comparação foi o tamanho das memórias ROM (programa) e RAM (dados), que podem fornecer uma medida da ocupação de área. Potência consumida e velocidade de execução também foram analisadas, cobrindo a maioria das características físicas de cada método. Como o foco do trabalho é em sistemas embarcados, outros parâmetros importantes como tempo de treinamento, facilidade de automação, comportamento em freqüência e extensão da relação sinal ruído para um maior número de bits também foram estudados.

Nas seções 3.2.1 até 3.2.4, os parâmetros de qualidade de conversão, consumo de área, tempo de execução e potência dissipada são analisados, respectivamente. Parâmetros de comparação mais subjetivos como tempo de treinamento, facilidade de automação e outros, são analisados no capítulo 5.

3.2.1 Qualidade da Conversão

Tratando-se de sinais senoidais, a qualidade da conversão pode ser verificada através da análise espectral dos sinais de saída, uma vez que as não linearidades podem ser vistas, no domínio freqüência, como componentes harmônicas do sinal de entrada ou componentes espúrias de ruído. Para tanto, através da Transformada Rápida de Fourier é encontrado o parâmetro SFDR (*Spurius Free Dynamic Range* – Faixa dinâmica livre de espúrios)que é definido como a diferença, em dBs, entre o coeficiente da freqüência fundamental e próxima componente de maior magnitude dentro da banda de interesse [TIL99].

Por exemplo, nas figuras 3-9 e 3-10 tem-se as FFTs para um sinal senoidal de entrada convertido por um conversor de 8 bits. No primeiro caso, antes de ser convertido, o sinal passa por um sensor com uma resposta linear, e no segundo caso por um sensor que possui uma resposta não linear obtida a partir da função $f(x) = -0.196 + x + 0.1x^2$.

Na conversão com resposta linear tem-se uma SFDR de 60 dB, pois além da freqüência fundamental tem-se apenas o ruído de quantização causado pelo conversor. Já na conversão com resposta não linear a taxa SFDR cai para 27dB.

Visto desta forma, o objetivo do processo de linearização é devolver ao sinal a sua pureza espectral.

No caso da utilização de um sinal não senoidal, como por exemplo uma rampa, o parâmetro SFDR não poderia ser usado, apenas a comparação direta entre os espectros do sinal de entrada e do sinal linearizado.

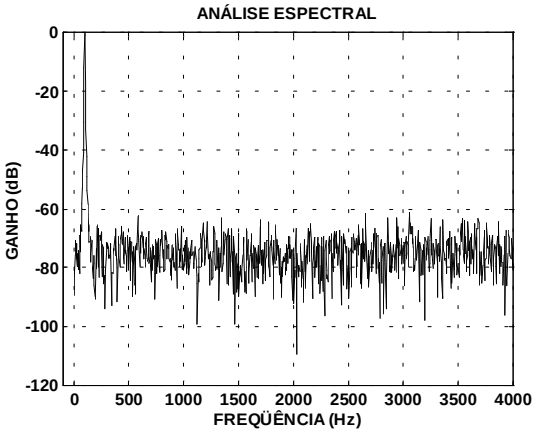

Figura 3-9 FFT sinal ideal .

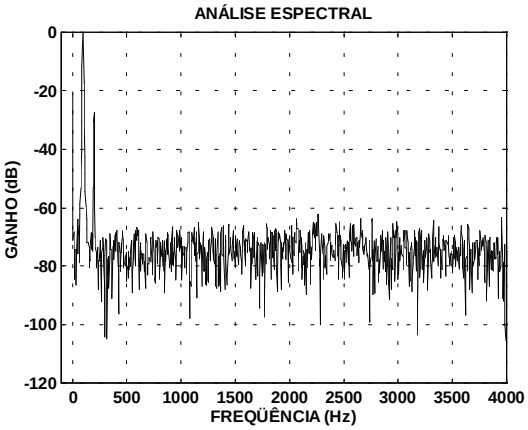

Figura 3-10 FFT sinal com não linearidades inseridas.

O procedimento utilizado para calcular as FFTs dos sinais adquiridos foi de utilizar a média de dez FFTs dos sinais de interesse. Além disso, para o cálculo de todas as FFTs foram utilizados sinais com 2048 pontos e janela de Hanning de mesmo tamanho.

3.2.2 Área Consumida

A área ocupada pelos algoritmos, após sua implementação no DSP, pode ser retirada das informações contidas nos arquivos gerados pelo próprio compilador do processador.

As memórias de dados (RAM) e de programa (ROM) usadas podem ser uma boa métrica para o consumo de área, já que os recursos computacionais são os mesmos para todos os algoritmos.

Uma medida mais eficiente de área poderia ser obtida através de síntese em VHDL. A área ocupada seria dada sob a forma de células lógicas utilizadas na implementação de cada método. Por exemplo, um registrador de 1 bit em tecnologia atual, utiliza 8 transistores MOS. Portanto, um registrador de 16 bits é formado por 128 transistores MOS. Contudo, a descrição de cada método em VHDL e sua validação transcenderia o escopo deste trabalho.

3.2.3 Tempo de Execução

O tempo de execução de um algoritmo tem uma forte correlação com o número total de ciclos de relógio usados durante a execução da calibração. Isto se deve principalmente ao fato de que cada instrução do processador escolhido é executada em apenas um ciclo de máquina.

Na primeira etapa de comparações, foram computadas as seguintes medidas de tempo:

- Total de ciclos utilizados ;
- Ciclos utilizados na conversão / Ciclos usados na calibração.

A soma dos ciclos utilizados na conversão e na calibração corresponde ao total de ciclos utilizados. Esta separação deve-se também ao fato do uso do conversor sigma-delta, pois facilita análises futuras.

Na segunda etapa de comparações, onde se utiliza um conversor A/D genérico, apenas o número de ciclos utilizados na calibração é computado.

3.2.4 Potência Dissipada

Como nos casos anteriores, a energia dissipada também foi medida de forma diferente para cada etapa de comparação.

Na primeira etapa, a medição da energia dissipada é um pouco mais problemática, para assegurar uma comparação justa entre os métodos.

A energia dissipada por um processador executando um programa (E) [GEB99], é o produto do tempo necessário para executar este programa (T), e a potência dissipada (P). P é dado pelo produto da corrente média e a tensão de alimentação (Vcc). O termo T é igual a N x τ, onde N é o número de ciclos de relógio e τ o período do ciclo, e estas relações são apresentadas na equação 3-7.

$$E = P \times T = I \times Vcc \times T = I \times N \times \tau \times Vcc \qquad (3\text{-}7)$$

Na figura 3-11 tem-se o esquema de hardware usado para medir a corrente média. Um osciloscópio digital HP54645D foi usado para realizar a aquisição da tensão sobre o resistor R (shunt), então os dados foram processados em MatLab5.3 seguindo a equação 3-7.

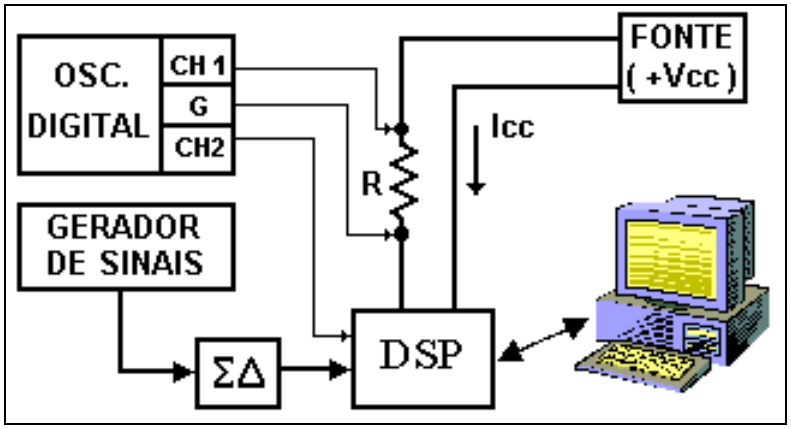

Figura 3-11 Esquema de hardware utilizado para medir corrente consumida, primeira etapa.

Ainda, para medir a potência dissipada por cada método, foi necessário analisar a forma como cada método funciona, a fim de se estabelecer uma comparação justa. Na figura 3-12 tem-se um diagrama seqüencial de como os algoritmos ocorrem temporalmente. Analisando esta figura observa-se que:

- D é a taxa de decimação, ou seja, são necessários D pulsos de relógio do modulador sigma delta para se ter um dado válido, e na freqüência de Nyquist, ao final da filtragem;
- O relógio do modulador trabalha na freqüência de sobre-amostragem de 256kHz e cada pulso de relógio do modulador, o DSP tem tempo para executar 130 instruções, conforme equação 3-4.
- No método Tabela (LUT) e no método Polinomial o processador executa durante D ciclos o algoritmo de filtragem digital do conversor sigma-delta ($Sinc^2$) e somente no último ciclo, após ter-se um dado válido e decimado, portanto na taxa de Nyquist, é que se aplica o algoritmo de linearização. Este é o algoritmo T para o método Tabela e algoritmo Polinômio para o método Polinomial;

- No método Inserido no Conversor em todos os ciclos de relógio do modulador o DSP executa os algoritmos referentes a filtro decimador e de linearização, algoritmos MM, I.C. e $Sinc^{2*}$.

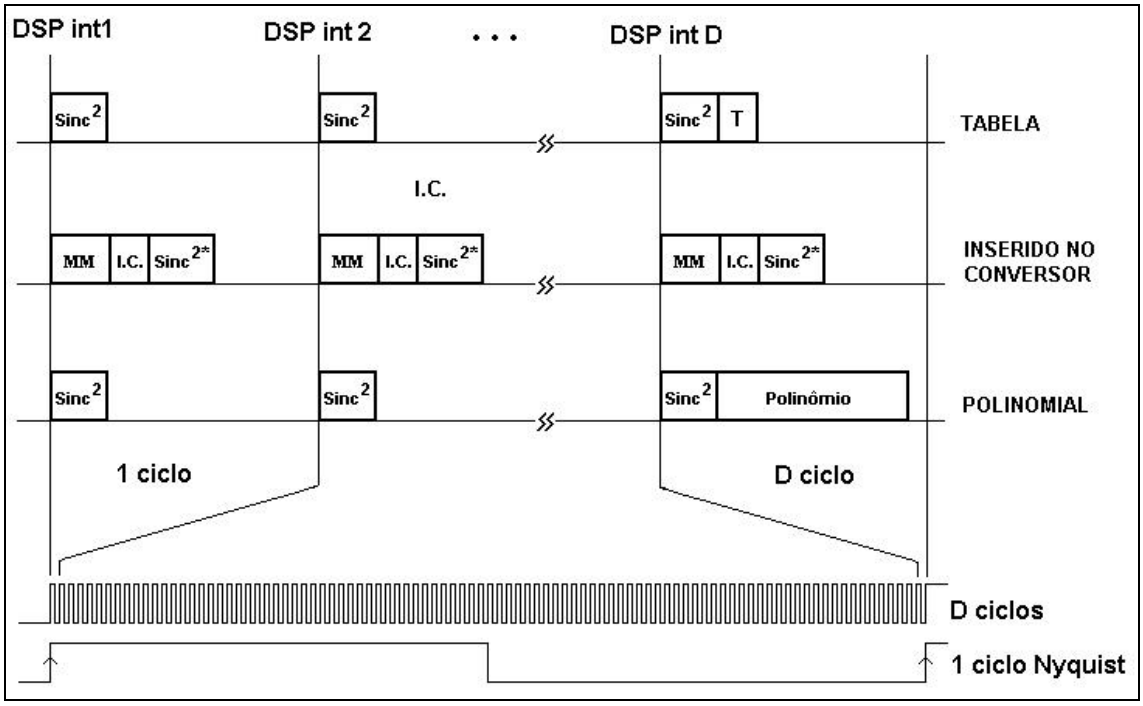

Figura 3-12 Diagrama seqüencial dos métodos de linearização.

Para o cálculo da potência dissipada por cada método, pode-se desprezar a potência dissipada pela parte analógica, que é comum a todos os métodos, porém as contribuições dos filtros digitais SINC devem ser consideradas.

Desta forma, seguindo-se o esquema da figura 3-8, as equações 3-6 descrevem como foi realizado o cálculo da potência dissipada por cada método.

Método LUT : $P_{d\_total} = P_d (Sinc^2) \times D + P_d (T)$

Método Polinomial : $P_{d\_total} = P_d (Sinc^2) \times D + P_d (Polinomial)$ (3-6)

Método Inserido no Conversor : $P_{d\_total} = (P_d (Sinc^{2*}) + P_d (MM) + P_d (I.C.)) \times D$

Onde:

- P_d (x) é a potência dissipada ao se executar o algoritmo (x);
- Sinc2 é um filtro digital transversal de 1 bit do tipo Sinc;
- T é o algoritmo de linearização do método Tabela;

- I.C. é parte do algoritmo de linearização do método Inserido no Conversor;
- MM é um filtro digital transversal de média móvel, que faz parte do método Inserido no Conversor;
- Sinc2* é um filtro digital transversal do tipo Sinc, que compõe o método Inserido no Conversor;
- D é a taxa de decimação do filtro digital.

Note-se que os filtros $Sinc^2$ e $Sinc^{2*}$ são filtros diferentes, já que o primeiro filtra um sinal de um bit vindo do modulador sigma-delta, enquanto o segundo filtra uma palavra digital de n bits vinda da tabela de correção. Portanto, a área e a potência consumidas pelo $Sinc^{2*}$ são maiores do que no $Sinc^2$.

Os resultados finais para os cálculos da potência dissipada, para todos os métodos, foram realizados a partir das médias da corrente medida para um conjunto de dez aquisições.

Na segunda etapa de comparações o esquema de hardware utilizado para medir a energia dissipada pelo DSP muda muito pouco. Conforme a figura 3-13, retirou-se o modulador sigma-delta, e utilizou-se o conversor A/D inserido no kit de desenvolvimento. Portanto, todas as operações referentes à conversão A/D são externas ao DSP.

Nesta nova configuração, a corrente drenada pela placa de desenvolvimento é medida somente durante a execução do respectivo algoritmo de linearização, uma vez que o DSP não realiza mais operações referentes à conversão A/D.

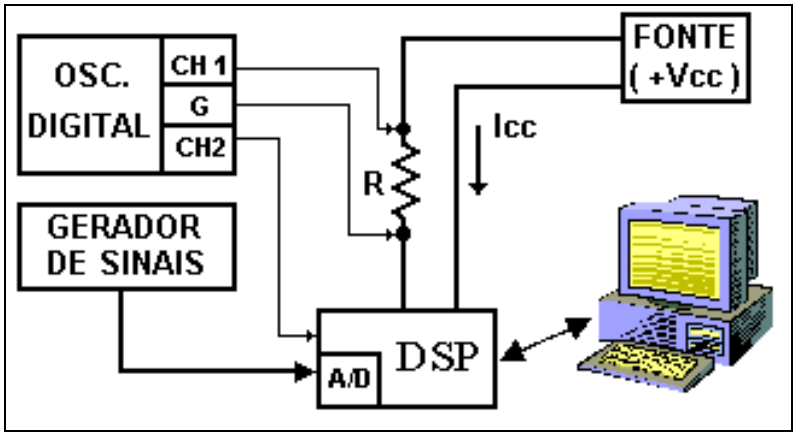

Figura 3-13 Esquema utilizado para medir corrente consumida, segunda etapa.

4 Resultados

O resultados práticos obtidos durante as implementações realizadas serão apresentados a seguir. Na seção 4.1 são apresentados e discutidos os resultados referentes às implementações dos métodos Tabela, Inserido no Conversor e Polinomial [BET02].

O método Polinomial, que apresentou os melhores resultados na primeira etapa de comparação, é comparado novamente ao método com filtragem adaptativa não linear. Para esta última comparação utilizou-se um conversor A/D genérico. Os resultados desta comparação são apresentados e discutidos na seção 4.2.

4.1 PRIMEIRA ETAPA DE COMPARAÇÃO

Na tabela 1 são apresentados os resultados da comparação dos diferentes métodos em relação à eficiência de calibração. Para um sinal de entrada com não-linearidade de 7%, conseguiu-se uma redução para 0,4% para todos os métodos. Desta forma, todas as comparações posteriormente apresentadas foram desenvolvidas levando-se em consideração o mesmo nível de redução de não linearidade.

A SFDR obtida para todos os métodos corresponde aos valores encontrados durante as simulações MatLab5.3. Isto é importante para relacionar as diferenças que cada método apresenta em termos de potência dissipada e tempo máximo de execução.

Primeiramente, a aparente pequena diferença de tamanho de programa, encontrada em todos os métodos, pode ser explicada pelo fato de que todos tem inserido em seu código as etapas de filtragem e decimação referentes ao conversor sigma-delta. A diferença de fato está apresentada na linha 8 da tabela, que mostra a quantidade de instruções considerando-se somente o código que realiza a calibração e a parte de código usada pelo processo de conversão A/D.

A memória de dados cresce exponencialmente com o número de bits a serem calibrados no método LUT, e o mesmo ocorre no método de calibração Inserido no Conversor. Este crescimento não ocorre no método Polinomial, indicando que este método

pode ser favorecido quando a memória usada tem maior importância por razões de área e velocidade.

Desde que todos os métodos foram implementados com o mesmo conversor A/D, é interessante notar que muitos ciclos do microprocessador são usados pelo próprio conversor. Pelo menos 28 ciclos são consumidos na filtragem do *bit-stream*, e isto faz com que as informações da linha 7 (tempo de execução) sejam mascaradas. Mesmo descontando-se as operações de filtragens e decimação requeridas pelo modulador sigma-delta, o número de ciclos de relógio utilizados pelo método Polinomial parece ser muito maior do que o utilizado pelos outros métodos. Entretanto, deve-se notar que muitos passos de calibração executados dentro do conversor são executados em uma alta taxa de amostragem. Como mencionado, na linha 8 apresenta-se o número de ciclos executados na freqüência de sobre amostragem (requerido pelo sigma-delta e pelo método de calibração inserido nele) e o número de ciclos na taxa de Nyquist. Claramente, no caso de integração de hardware ou no caso de um sinal com maior freqüência ser necessário, a velocidade de sobre amostragem requerida pelo método Inserido no Conversor irá necessitar maiores recursos de hardware.

Em relação à potência dissipada, pode-se notar que as operações de filtragem e sub-amostragem executadas para trazer o sinal para a taxa de Nyquist são dominantes. Isto pode ser visto na linha 9, onde os ciclos gastos para trazer o sinal de uma freqüência de sobre amostragem para a taxa de Nyquist causam a maior dissipação de potência em todos os métodos. Portanto, o método Inserido no Conversor apresenta a maior dissipação, desde que ciclos extras de processamento são necessários para endereçar a memória a uma taxa de amostragem alta.

Na linha 10, tem-se a quantidade de energia consumida por operações realizadas na taxa de Nyquist. Claramente, esta é a energia gasta nos passos finais do processo de calibração, e neste caso o método Polinomial drena maior potência nesta fase. Em relação a toda potência dissipada, portanto, o método Inserido no Conversor drena mais potência do que os outros métodos, graças às operações extras executadas na taxa de sobre-amostragem.

As três últimas linhas da tabela 1 mostram algumas características as quais são mais difíceis de quantificar, mas que são igualmente importantes para aplicações embarcadas, e que serão melhor explicadas no capítulo 5. Na linha 13 discute-se a facilidade de automação, que é claramente relacionada com o tempo de projeto. O método Polinomial pode ser favorecido neste caso, desde que todos os passos são realizados pelo próprio algoritmo, sem a assistência de um operador. Os outros métodos requerem uma rede extra de re-alimentação para prover armazenagem automática dos dados de calibração.

Finalmente, na ultima linha da tabela 1, discute-se a generalidade de cada método. A principal desvantagem do método Inserido no Conversor é justamente o fato dele ter de ser inserido no conversor, o qual pode criar dificuldades no caso de conversores com propriedade intelectual, ou quando outras arquiteturas de conversores devem ser usadas para aumentar a velocidade, por exemplo.

Tabela 1 - Resultados Experimentais Primeira Etapa.

#	Parâmetros medidos	LUT	Inserido no Conversor	Polinomial
1	SFDR do sinal linear de entrada	66,06 dB	66,06 dB	66,06 dB
2	SFDR do sinal não linear	23,14 dB	23,14 dB	23,14 dB
3	SFDR do sinal corrigido	58,3 dB	57,4 dB	60,56 dB
4	Memória de programa (palavras de 16 bits)	109 palavras	118 palavras	179 palavras
5	Memória de dados (palavras de 16 bits)	261 palavras	37 palavras	14 palavras
6	Área total utilizada: (palavras de 16 bits) (dados + memória de programa)	370 palavras	155 palavras	193 palavras
7	Tempo de execução	33 ciclos	52 ciclos	107 ciclos
8	Ciclos conversor/calibração	28/5	51/1	28/78
9	Soma da Energia Média dissipada nos últimos (N-1) ciclos.	175,76e-6J	426,13e-6 J	176,07 e-6 J
10	Energia Média dissipada no último ciclo (N)	1,21e-6J	2,05e-6 J	4,78 e-6 J
11	Energia Média Total dissipada	176,97 e-6 J	428,18 e-6 J	180,85 e-6 J
12	Tempo de treinamento	---	--	+++
13	Facilidade de automação	--	++	+++
14	Generalidade (conversor A/D)	Todos	Sigma-delta	Todos

4.2 SEGUNDA ETAPA DE COMPARAÇÃO

Na tabela 2 os resultados das implementações dos métodos Polinomial e Adaptativo são apresentados.

A qualidade da calibração apresentada nas linhas 1, 2 e 3, foi obtida a partir das FFTs dos sinais de entrada linear, não linear e compensado, sendo expressa em termos da SFDR dos sinais. Este parâmetro foi teoricamente pré-estabelecido como devendo ser igual em ambos os métodos, a fim de se estabelecer uma comparação justa dos demais parâmetros que serão analisados.

Tabela 2 – Resultados Experimentais Segunda Etapa

#	Parâmetros medidos	Polinomial	Adaptativo
1	SFDR do sinal linear de entrada	55dB	55dB
2	SFDR do sinal não linear	27,7dB	27,7dB
3	SFDR do sinal corrigido	49,3 dB	50 dB
4	Memória de programa (palavras de 16 bits)	234 palavras	266 palavras
5	Memória de dados (palavras de 16 bits)	33 palavras	48 palavras
6	Área total utilizada: (palavras de 16 bits) (dados + memória de programa)	267 palavras	314 palavras
7	Tempo de execução	80 ciclos	90 ciclos
8	Energia Média Total dissipada	3,013 e-6 J	3,691 e-6 J
9	Tempo de treinamento	+++	++
10	Facilidade de automação	+++	+++

O método Polinomial ocupa uma menor área de programa (234 palavras digitas) e dados (33 palavras digitais) do que o método Adaptativo, que tem 266 palavras digitais de memória de programa e 48 palavras digitais de dados, conforme se vê nas linhas 4 e 5. Esta diferença deve-se ao fato do método com filtragem adaptativa necessitar de mais operações para realizar o processo de linearização, e por ter um número maior de coeficientes de calibração. Somando-se memória de dados e programa o método Polinomial leva uma vantagem de 47 palavras digitais em relação ao método Adaptativo.

Com relação ao tempo de execução dos algoritmos, novamente o método Polinomial tem melhor performance, sendo 10 ciclos mais rápido do que o método Adaptativo. Com esta diferença o método Adaptativo é aproximadamente 12,5% mais lento do que o Polinomial. Aumentando-se o número de bits requerido na conversão este diferença tende a se manter.

Este número maior de ciclos de execução ocasiona também um aumento na energia dissipada pelo DSP, uma vez que estes parâmetros estão diretamente relacionados.

O método Polinomial, sob o ponto de vista de sistemas embarcados, apresentou a melhor performance para os requisitos de consumo de área, potência dissipada e velocidade de execução quando comparado com o método Adaptativo. Os dois métodos necessitam de uma fase de treinamento, a fim de calibrar o sistema sensor e apresentam características adequadas a automação de sistemas. Estas características serão melhor analisadas no capítulo 5.

5 Outros parâmetros de comparação

Ao optar-se pela utilização de um método em um projeto, é importante considerar outras características de métodos de linearização menos facilmente mensuráveis, mas que podem alterar o custo ou até mesmo inviabilizar seu uso.

Tempo de treinamento, facilidade de automação, comportamento em freqüência e taxa do aumento de custo área e de custo velocidade com o aumento do número de bits requeridos na conversão foram analisados, e são apresentados nas seções a seguir.

5.1 TEMPO DE TREINAMENTO

Produção de equipamentos em larga escala requer custos baixos e rapidez de produção e teste. O tempo de treinamento ou de calibração do sistema sensor tem impacto direto sobre estes custos.

Dentre os métodos implementados, o método Tabela é o que possui o maior tempo de treinamento. Este método é muito dependente de um operador e de equipamentos para realizar a sua calibração. O método consiste em levantar toda a curva de calibração do sensor através de várias medições e armazená-las na memória de dados.

Para sistemas sensores que possuem uma função de transferência bem conhecida e repetitiva entre lotes de produção, a curva de resposta poderia ser adquirida somente uma vez e utilizada para todos os sistemas. O tempo de calibração neste caso seria o tempo de gravação da memória realizada por um operador.

No método de linearização Inserido no Conversor, o tempo de treinamento tende a diminuir consideravelmente. O menor número de pontos de calibração facilita e agiliza o processo de calibração. No método Tabela, para um conversor de 8 bits, são necessárias 256 medições, enquanto que no método Inserido no Conversor somente 16 medições são suficientes, já que os coeficientes de calibração são aplicados na freqüência de sobre-amostragem, onde o sinal está quantizado em um número menor de bits (4 bits neste caso).

No método Polinomial progressivo o número de medições de calibração é menor ainda. Porém, depende do número de passos de calibração que serão utilizados. Para quatro passos de calibração (M=4), são necessárias quatro medições padrão pré-estabelecidas e 25 operações matemáticas, ou seja, 10 subtrações, 6 adições, 6 multiplicações e 3 divisões. Para cada novo passo de linearização implementado são necessárias (2n-2) novas operações.

No método utilizando filtragem adaptativa não linear, o tempo de treinamento depende dos parâmetros estimados para o algoritmo LMS. Parâmetros como número de taps, erro mínimo e passo de convergência determinam este tempo. No exemplo implementado neste trabalho, obteve-se uma solução utilizando um filtro linear de 10 taps, um filtro x^2 com 4 taps, um filtro x^3 com 4 taps. Para obter-se a qualidade de calibração desejada foram necessárias 30000 iterações (escrita e leitura do sinal de referência). A cada iteração o algoritmo LMS realiza aproximadamente 56 operações de multiplicação, 20 operações de soma e 1 subtração, perfazendo um total de 77 operações matemáticas.

Em relação aos outros métodos, este é o que possui o maior tempo de treinamento, pois um número significativamente grande de operações deve ser realizado até a obtenção de erro um mínimo. Entretanto, não se exige a necessidade de um operador, e o sinal de referência deve ser ruído branco, o qual pode ser facilmente produzido.

5.2 FACILIDADE DE AUTOMAÇÃO DA CALIBRAÇÃO

Enquanto o tempo de treinamento ou de calibração tem impacto direto sobre os custos de produção, a facilidade de automação tem maior impacto sobre o tempo de projeto. Estes dois fatores influem significativamente no custo dos sistemas atuais.

Quanto mais compacto for o algoritmo de calibração, e menor o tamanho do circuito a ser implementado, maior será facilidade de integração junto ao sistema maior para o qual foi projetado.

No método Tabela, a necessidade de um grande número de medições de calibração, a grande área ocupada pela memória e a necessidade da intervenção constante de um operador durante o processo de calibração dificultam a integração e a automação deste método. Soluções como auto-calibração e auto-teste são mais problemáticas de serem implementadas, também devido ao grande número de medidas.

Este problema é resolvido em parte no método Inserido do Conversor. Um número menor de coeficientes a serem armazenados, e de medidas de calibração a serem realizadas tornam este método mais adequado para a integração em sistemas embarcados. Auto-teste e auto-calibração podem ser implementadas, dispensando a intervenção de operação manual.

Da mesma forma, o método Polinomial apresenta facilidades de automação e integração. Mesmo para altas resoluções, o número de calibrações é pequeno. A calibração pode ser totalmente automatizada, independentemente do tipo de sensor e da variação de suas características.

O método que utiliza filtragem adaptativa não linear possui boas características para a integração em sistemas embarcados. Apesar do número de operações e coeficientes serem maiores do que no método Polinomial, a facilidade de automação e a implementação de rotinas de auto-calibração e de auto-teste são notadamente importantes. Neste método a intervenção de um operador é bastante reduzida, aumentando a confiabilidade e diminuindo o custo global do sistema.

5.3 COMPORTAMENTO EM FREQÜÊNCIA

Dentre os métodos estudados, os métodos Tabela, Inserido no Conversor e Polinomial, são amplamente aplicados tanto a sinais de entrada do tipo AC (corrente alternada) como do tipo DC (corrente contínua). Ao contrário destes métodos, o método Adaptativo possui melhor performance quando aplicado a sinais estacionários com média zero. Os três primeiros são baseados na resposta estática do sensor, ou seja, nos valores de entrada *versus* saída do sensor, e portanto, apresentam uma melhor performance para sinais de entrada constantes ou que variem lentamente com o tempo, como em sensores de temperatura por exemplo.

Na topologia utilizada no método Adaptativo, o sistema sensor é inversamente modelado de forma a possibilitar uma correção em toda sua faixa dinâmica. Este método, apesar de não ser aplicado a sinais contínuos, tem como ponto forte o bom desempenho para sensores que dependem não somente dos valores de amplitude do sinal de entrada, mas também da sua mudança em freqüência, como em sensores de aceleração, por exemplo [DOE90, ADL98].

5.4 TAXA DE AUMENTO DO CUSTO DE ÁREA POR BIT EXTRA DE RESOLUÇÃO

Tendo em vista que um sistema sensor pode ter diversas aplicações em diferentes sistemas embarcados, é provável que o número de bits necessários na conversão dos sinais, para estes sistemas, varie de aplicação para aplicação. Desta forma, uma projeção do custo de área com o aumento de bits de resolução, pode permitir ao projetista uma escolha adequada a cada tipo de aplicação. Nas seções a seguir tem-se uma aproximação matemática para os quatro métodos de calibração estudados.

5.4.1 Método Tabela

No método Tabela um aumento no número de bits requeridos na conversão claramente ocasiona um aumento na área de memória responsável por armazenar os coeficientes de calibração. Este aumento ocorre de forma exponencial, conforme descreve a equação 5-1.

A área total consumida pelo método de linearização em função do número de bits e dada por:

$$A_{total} = p \times 2^n \tag{5-1}$$

Onde p é o tamanho da palavra digital na saída da tabela e n o número de bits que endereçam esta tabela. Para efeitos de comparação, assume-se que n = p, logo:

$$A_{total} = n \times 2^n \tag{5-2}$$

$$A_{total} = n \times 2^n \times A_{MEM} \tag{5-3}$$

Sendo que a área ocupada por 1bit de memória corresponde a 8 transistores MOS ($A_{MEN} = 8\ transistores\ MOS$), tem-se que área total expressa em número de MOS é:

$$A_{total} = n \times 2^n \times 8 \tag{5-4}$$

O gráfico da figura 5-1 mostra a evolução do consumo de área com o acréscimo do número de bits requerido na conversão, para o método Tabela. Observa-se através da equação 5-4 e de seu gráfico o comportamento exponencial do crescimento de área para uma variação no número de bits de 4 a 16.

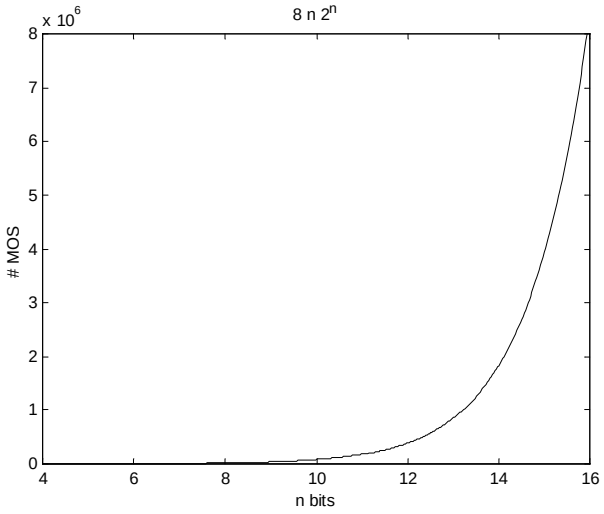

Figura 5-1 Aumento do custo área com o aumento do número de bits para o método Tabela.

5.4.2 Método Inserido no Conversor

Neste método a área consumida depende, basicamente, de três blocos distintos, como mostra a figura 5.2.

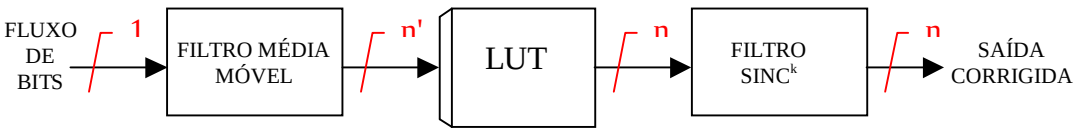

Figura 5-2 Esquema de linearização Inserido no Conversor.

No primeiro bloco tem-se um filtro digital do tipo média móvel. Note-se na figura 5-3 que este filtro é composto de registradores de um bit e um grande somador. Seu consumo de área pode ser dado pela equação 5-5.

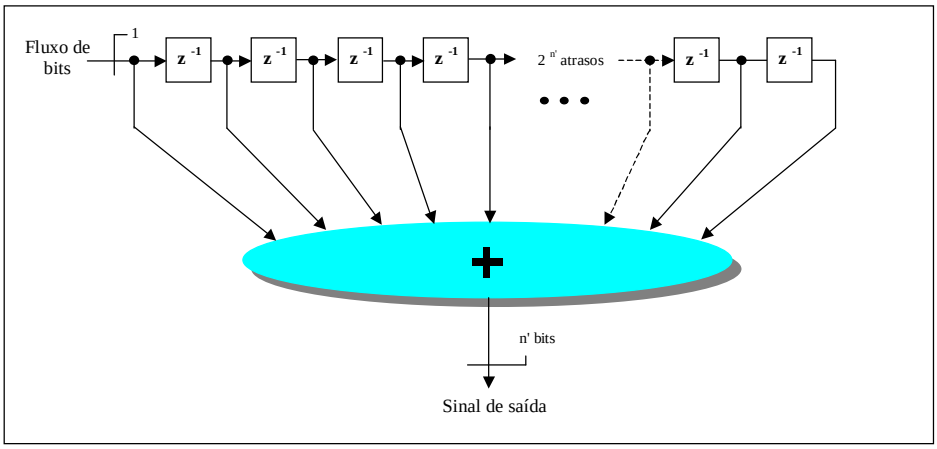

Figura 5-3 Topologia de um filtro média móvel.

$$A_{MM} = 2^{n'} \times A_{REG.1bit} + 2^{n'} \times A_{\Sigma 1bit} \qquad (5\text{-}5)$$

onde, $A_{REG.1bit} = 16\ transistores\ MOS$ (Área de um registrador de um bit) e

$A_{\Sigma 1bit} = 28\ transistores\ MOS$ (Área de um somador de um bit).

Portanto, a área do filtro média móvel em número de transistores MOS é:

$$A_{MM} = 2^{n'} \times 16 + 2^{n'} \times 28 \qquad (5\text{-}6)$$

Como regra de projeto, para efeitos de comparação, a aproximação $n' = \frac{n}{2}$ pode ser aplicada, portanto, substituindo-se na equação 5-6, tem-se a equação 5-7.

$$A_{MM} = 2^{\frac{n}{2}} \times 16 + 2^{\frac{n}{2}} \times 28 \qquad (5\text{-}7)$$

No segundo bloco da figura 5-2, tem-se uma área de memória (LUT) que contém os dados de calibração, e a área consumida por este bloco é dada por:

$$A_{LUT} = n \times 2^{n'} \times A_{MEM} \qquad (5\text{-}8)$$

Seguindo o critério de que $n' = \frac{n}{2}$, tem-se a área da tabela (LUT) em função do número de bits, e expressa em número de transistores MOS:

$$A_{LUT} = n \times 2^{\frac{n}{2}} \times 8 \qquad (5\text{-}9)$$

Para o terceiro bloco da figura 5-2, filtro decimador sinc^k, a solução não é tão trivial de ser encontrada, uma vez que a área consumida não depende mais somente do número de bits necessários na conversão, mas também da ordem do filtro decimador (k) e da taxa de sobre amostragem utilizada. Utilizando-se um filtro decimador de ordem 2 e taxa de decimação de 256 ou maior, consegue-se uma atenuação suficiente para representar 16 bits [CAN86].

Um filtro sinc^2 é construído, de forma clássica, por dois filtros sinc^1 em cascata. Tal filtro pode ser implementado utilizando-se somadores, subtratores e registradores [LEU94]. Desta forma, a área consumida por estes operadores e registradores está relacionada na equação 5-12.

$$Asinc^2 = (2 \times n \times A_{\Sigma.1bit}) + (2 \times n \times A_{\Sigma.1bit}) + (5 \times n \times A_{REG.1bit}) \quad (5\text{-}10)$$

$$Asinc^2 = (2 \times n \times 28) + (2 \times n \times 28) + (5 \times n \times 16) \quad (5\text{-}11)$$

$$Asinc^2 = (192 \times n)\ transistores\ MOS \quad (5\text{-}12)$$

A equação da área total, equação 5-14, é obtida a partir da soma das equações parciais 5-7, 5-9 e 5-12.

$$A_{IC} = (2^{\frac{n}{2}} \times 16 + 2^{\frac{n}{2}} \times 28) + (n \times 2^{\frac{n}{2}} \times 8) + (192 \times n) \quad (5\text{-}13)$$

$$A_{IC} = (2^{\frac{n}{2}} \times 44) + (2^{\frac{n}{2}} \times 8 \times n) + (192 \times n) \quad (5\text{-}14)$$

Na figura 5-4 tem-se a curva do aumento do custo de área (A_{IC}) em função do número de bits, para o intervalo de 4 a 16 bits de resolução. Note-se que à medida que o número de bits aumenta, o número de transistores MOS utilizados aumenta de forma exponencial.

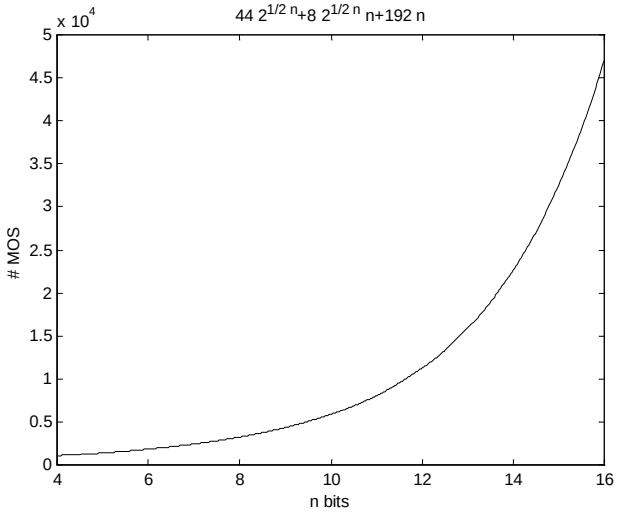

Figura 5-4 Aumento do custo área com o aumento do número de bits para o método Inserido no Conversor.

5.4.3 Método Polinomial

Na figura 5-5 tem-se uma implementação em hardware digital sugerida por [HOR94]. O número de operações depende do número de passos de calibração (*M*). Enquanto que o tamanho dos operadores depende do número de bits requeridos na conversão (*n*). O consumo de área neste método pode ser determinado conforme a equação 5-15.

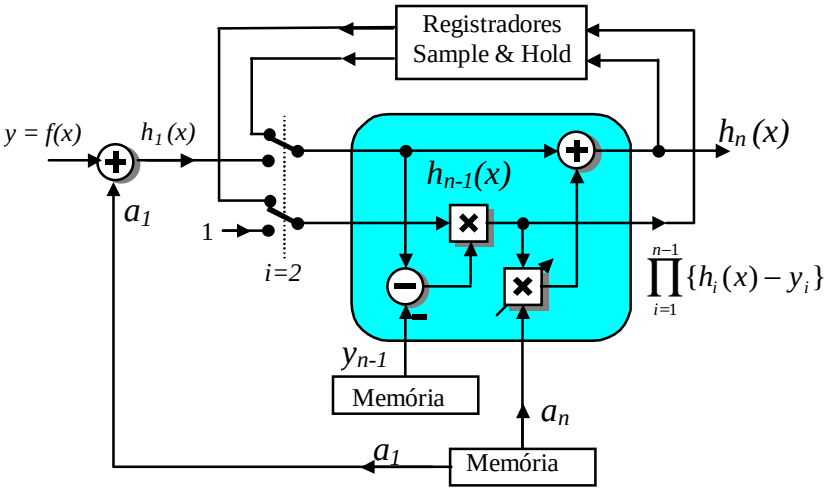

Figura 5-5 Método Polinomial, esquema de hardware.

$$A_{POL} = (2\times n\times A_{\Sigma.1bit}) + (2\times A_{MULT}) + (2\times n\times A_{SUB.1bit}) + (M\times n\times A_{MEM}) + \{(M-1)\times n\times A_{MEM}\} + (2\times n\times A_{REG.1bit}) \quad (5\text{-}15)$$

Onde:

- Área de um somador de 1 bit: $A_{\Sigma.1bit} = 28\ transistores\ MOS$;
- Área de um subtrator de 1 bit: $A_{SUB.1bit} = 28\ transistores\ MOS$;
- Área de um registrador LATCH: $A_{MEM} = 8\ transistores\ MOS$.

Um multiplicador do tipo paralelo de nxn bits é composto por n(n-2) somadores completos (28 MOS), n meio somadores (12 MOS) e n^2 portas AND (2 MOS) [WES85]. Portanto, a área do multiplicador em função do número de bits por ser dado por:

- Área do multiplicador: $A_{MUlT} = (30\times n^2 + 24\times n)\ transistores\ MOS$;

O parâmetro *M* (número de passos de calibração) depende do erro mínimo que se quer obter ao final do processo de linearização e também do comportamento da função não linear de entrada. Para a função não linear utilizada neste trabalho, apenas 4 passos de calibração foram necessários, portanto considerou-se este para fins de avaliação.

Desta forma, substituindo na equação 5-15, *M=4* e os valores de área por operador definidos acima, obtém-se a equação 5-16 somente em função do número de bits (*n*).

$$A_{POL} = (n\times 56) + (30\times n^2 + 24\times n) + (n\times 56) + (n\times 32) + (n\times 24) + (2\times n\times 16) \quad (5\text{-}16)$$

Somando-se os termos, tem-se a área total para o método Polinomial expressa em número de transistores MOS:

$$A_{POL} = (30\times n^2) + (224\times n)\ \text{(transistores MOS)} \quad (5\text{-}17)$$

A partir da equação 5-17, obteve-se a curva da figura 5-6, que representa o aumento da área em função do acréscimo de bits. Considerou-se uma variação no número de bits de 4 a 16. Note-se que a curva tem um comportamento aproximadamente linear, o que demonstra

que tanto para pequenas resoluções, quanto para maiores, o consumo de área é menor que nos métodos Tabela e Inserido do Conversor.

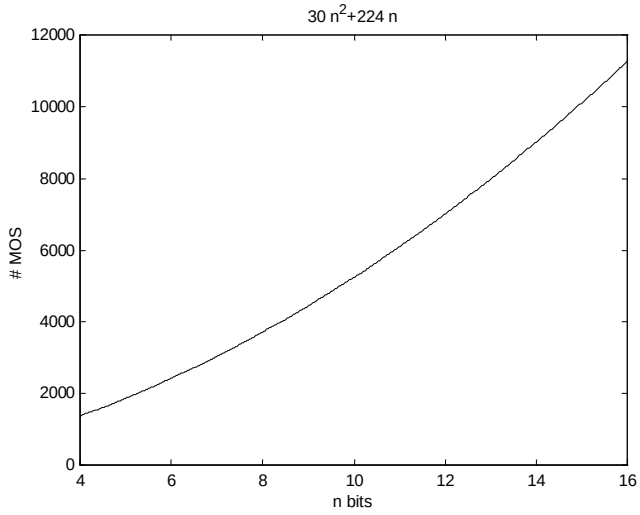

Figura 5-6 Aumento do custo área com o aumento do número de bits para o método Polinomial progressivo.

5.4.4 Método com Filtragem Adaptativa Não Linear

Os filtros utilizados são do tipo FIR, portanto formados basicamente de registradores, multiplicadores e memórias que guardam os coeficientes ou pesos do filtro, conforme a figura 5-7.

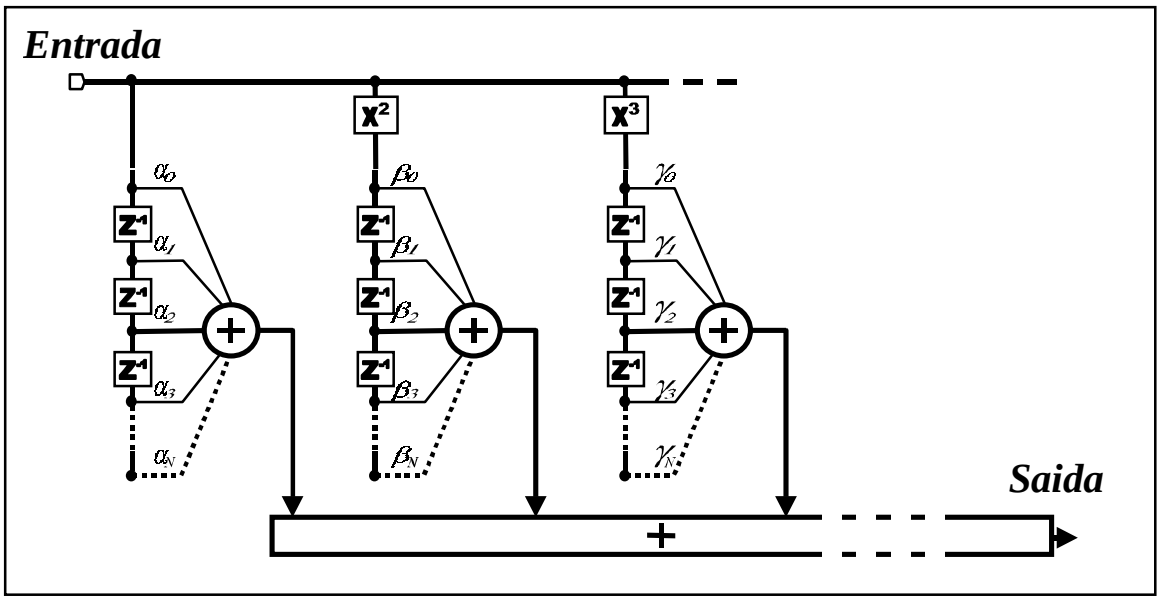

Figura 5-7 Esquema de linearização utilizando filtro adaptativo não linear.

Utilizando-se uma estrutura de hardware do tipo processador, pode-se utilizar um único multiplicador *nxn*, que realiza todos as operações de multiplicação, inclusive os termos x^2 e x^3 do sinal de entrada, um somador de *n* bits, dois registradores de *n* bits e *Txn* memórias. *T* é o número de taps e *n* o número de bits. A área do filtro é expressa conforme a equação 5-18.

$$A_{FA} = A_{MULT} + (2 \times n \times A_{REG.1bit}) + (n \times A_{\Sigma.1bit}) + (T \times n \times A_{MEM}) \qquad (5\text{-}18)$$

O número de taps de cada filtro, da mesma forma que no método Polinomial, depende do erro mínimo que se deseja obter ao final da linearização. Para a função não linear utilizada neste trabalho, foram necessários 10 taps para o filtro linear, 4 taps para o filtro com entrada x^2 e 4 taps para o filtro com entrada x^3, perfazendo um total de 18 taps (*T=18*). Porém, convém lembrar que dependendo dos critérios adotados na fase de treinamento, do erro mínimo que se deseja obter e do tipo de função de entrada que se quer corrigir, pode haver uma grande variação no número taps dos filtros utilizados.

Substituindo-se os valores de área em função do número de transistores MOS, obtém-se uma equação final 5-19.

$$A_{FA} = (30 \times n^2 + 24 \times n) + (2 \times n \times 16) + (n \times 28) + (18 \times n \times 8) \qquad (5\text{-}19)$$

Finalmente, somando-se os termos, tem-se:

$$A_{FA} = (30 \times n^2) + (228 \times n)\ transistores\ MOS \qquad (5\text{-}20)$$

Expressando-se, graficamente, a equação 5-20 em função do número de bits, obteve-se a curva da figura 5-8, que mostra o número de transistores MOS utilizados. Verifica-se que esta curva tem um comportamento aproximadamente linear, e bastante similar ao do método Polinomial. Considerou-se uma variação no número de bits de 4 a 16.

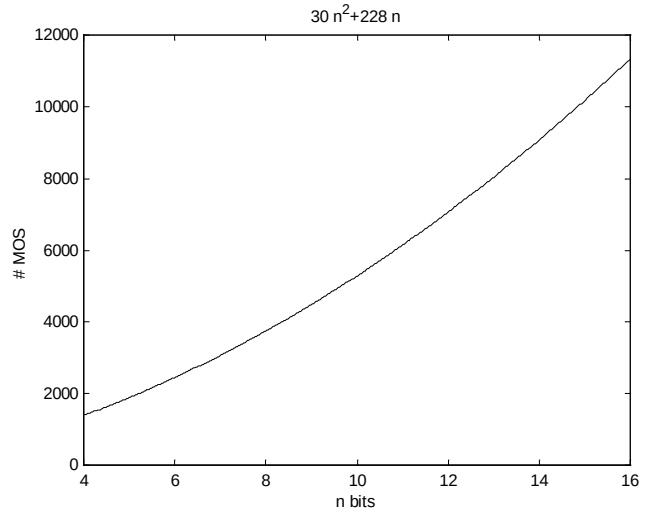

Figura 5-8 Aumento do custo área com o aumento do número de bits para o método com filtragem adaptativa não linear.

A fim de verificar-se a taxa de aumento do custo de área de cada um dos quatro métodos estudados, obteve-se a derivada das equações 5-4, 5-14, 5-17 e 5-20 em função de n, obtendo-se o gráfico da figura 5-9. Pode-se verificar facilmente neste gráfico que o método Tabela (símbolo - □) tem a maior taxa de crescimento de área, seguido do método Inserido no Conversor (símbolo - ∇), ambos com um comportamento exponencial devido à área ocupada pela memória de dados. Com um comportamento de crescimento linear, tem-se o método Polinomial (símbolo - •) e o com filtragem adaptativa (símbolo - o). Nestes dois últimos métodos a taxa de crescimento e a área ocupada são praticamente iguais.

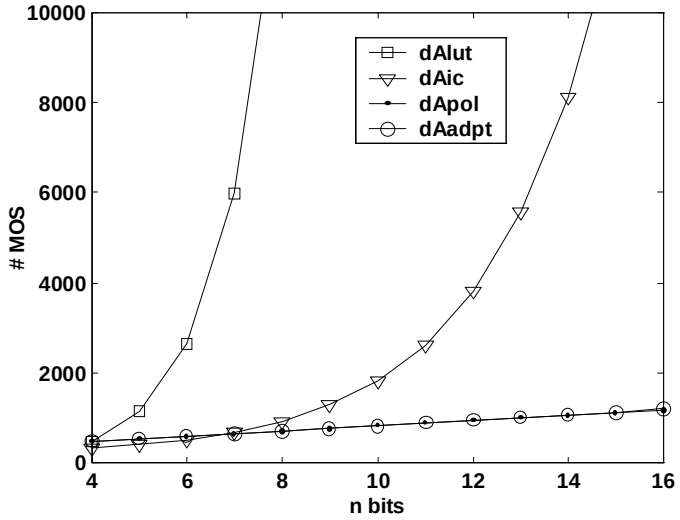

Figura 5-9 Taxa de crescimento do custo área com o aumento do número de bits para todos os métodos.

5.5 TAXA DE AUMENTO DO CUSTO DE VELOCIDADE POR BIT EXTRA DE RESOLUÇÃO

À medida que o número de bits do sistema é expandido, tem-se um acréscimo de área, como visto na seção anterior. Este aumento de área ocorre devido ao aumento do tamanho dos operadores, e também em alguns casos devido à inclusão de novas operações. Esta variação, obviamente, provoca também variações no tempo de resposta do sistema, aumentando ou diminuindo a velocidade de operação do processo de linearização.

No método Tabela, viu-se na seção 5.4 – equação 5-4, que em se aumentando o número de bits, a área da memória aumenta de forma exponencial. O mesmo não ocorre com o tempo de processamento, que apesar de memórias maiores terem um aumento no tempo de endereçamento, este aumento no tempo ocorre de forma sub-linear [WES85]. Desconsiderando-se este tempo de endereçamento, pode-se aproximar que o tempo de processamento do método Tabela é constante em relação ao número de bits n.

$$T_{LUT} = T_{MEM} \tag{5-21}$$

Para o método Inserido no Conversor, o tempo de processamento (T_{IC}) pode ser dado pela equação 5-22.

$$Tic = T_{MM} + T_{LUT} + T_{SINC} \tag{5-22}$$

O tempo de processamento do filtro de média móvel (T_{MM}) depende do número de bits n e do tempo dos somadores, conforme a relação da equação 5-23.

$$T_{MM} = T_{\Sigma} \times \frac{n}{2} \quad (5\text{-}23)$$

O tempo de processamento da memória (T_{LUT}) é praticamente constante. E o tempo de processamento do filtro sinc^2 (T_{SINC}) depende do número de bits dos operadores (somadores e subtratores), que tem um comportamento linear conforme demonstra a equação 5-24.

$$T_{SINC} = 4 \times T_{\Sigma} \times n \quad (5\text{-}24)$$

Substituindo-se as equações 5-22, 5-23 e 5-24 na equação geral 5-20, tem-se a equação final 5-25. Como se pode ver, esta equação apresenta um comportamento linear em n.

$$T_{IC} = T_{\Sigma} \times \frac{n}{2} + T_{LUT} + 4 \times T_{\Sigma} \times n \quad (5\text{-}25)$$

No método Polinomial, o tempo de processamento (T_{POL}), depende não só do número de bits requeridos para a conversão, mas também do número de passos de linearização utilizados. Analisando-se a figura 5-5, pode-se obter a equação 5-26 que mostra esta dependência:

$$T_{POL} = (n \times T_{\Sigma}) + (n \times M \times T_{SUB}) + (n \times 2 \times M \times T_{MULT}) + (n \times M \times T_{\Sigma}) \quad (5\text{-}26)$$

Utilizando-se 4 passos de linearização ($M = 4$), obtém-se a equação 5-27.

$$T_{POL} = (n \times T_{\Sigma}) + (n \times 4 \times T_{SUB}) + (n \times 2 \times 4 \times T_{MULT}) + (n \times 5 \times T_{\Sigma}) \quad (5\text{-}27)$$

Considerando o tempo do somador (T_{Σ}) igual ao tempo do subtrator (T_{SUB}) e ambos constantes, tem-se:

$$T_{POL} = (10 \times n \times T_{\Sigma}) + (n \times 8 \times T_{MULT}) \quad (5\text{-}28)$$

Ainda, o tempo do multiplicador (nT_{MULT}) para um multiplicador paralelo de $n \times n$ bits é igual a $(2n+1)T_{\Sigma}$ [WES85]. Utilizando esta relação tem-se a equação final 5-29.

$$T_{POL} = (10 \times n \times T_{\Sigma}) + \{8 \times [(2 \times n + 1) \times T_{\Sigma}]\} \quad (5\text{-}29)$$

$$T_{POL} = 26 \times n \times T_{\Sigma} + 8 \times T_{\Sigma} \quad (5\text{-}30)$$

O método utilizando filtragem adaptativa tem um comportamento similar ao método Polinomial, pois o tempo de processamento, T_{ADPT}, depende do número de pesos utilizados no processo de adaptação. O filtro adaptativo não linear durante a fase regime permanente processa, basicamente, operações de multiplicação e soma. E esta relação é dada pela equação 5-31.

$$T_{ADPT} = (T \times n \times T_{MULT}) + (T \times n \times T_{\Sigma}) \quad (5\text{-}31)$$

Para um número de pesos igual a 18, ($T=18$), e utilizando a relação $nT_{MULT}=(2n+1)T_{\Sigma}$, tem-se a equação 5-34.

$$T_{ADPT} = (18 \times n \times T_{MULT}) + (18 \times n \times T_{\Sigma}) \quad (5\text{-}32)$$

$$T_{ADPT} = \{18 \times [(2 \times n + 1) \times T_{\Sigma}]\} + (18 \times n \times T_{\Sigma}) \quad (5\text{-}33)$$

$$T_{ADPT} = 54 \times n \times T_{\Sigma} + 18 \times T_{\Sigma} \quad (5\text{-}34)$$

A taxa de aumento de custo velocidade por bit extra de resolução, para cada método estudado, pode ser encontrada a partir da derivada das equações 5-21, 5-25, 5-30 e 5-34. Considerando-se que tempo de execução da tabela ($T_{LUT}=T_{MEM}$) e que o tempo de execução do somador (T_{Σ}) são iguais a um ciclo, obtiveram-se as curvas da figura 5-9, para valores de n de 4 a 16 bits.

Claramente, o método Tabela mantém a mesma velocidade de operação com a variação do número de bits. No método Inserido no Conversor a cada bit requerido 4,5 ciclos a mais são necessários, no método Polinomial 26 e no com filtragem adaptativa 54.

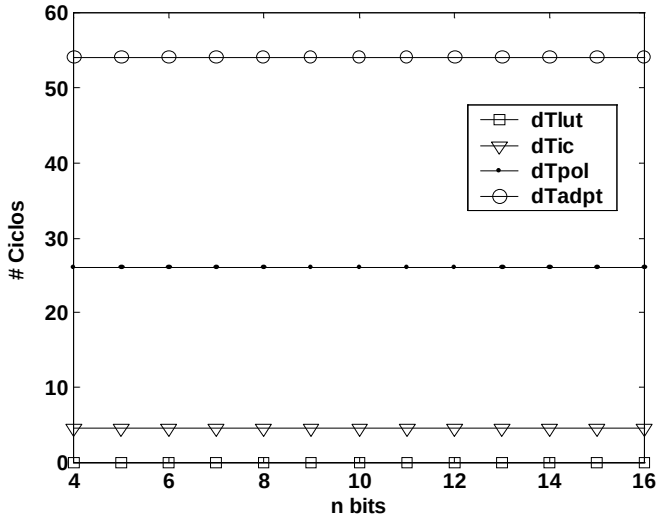

Figura 5-10 Taxa de aumento do custo velocidade com o aumento do número de bits.

Através da análise das figuras 5-9 e 5-10, percebe-se o compromisso existente entre área e tempo de processamento. Nos métodos Tabela e Inserido no Conversor, a taxa de aumento do custo de área é exponencial com o aumento número de bits. Entretanto, a velocidade tem uma característica aproximadamente constante. Nos métodos Polinomial e Adaptativo troca-se área por tempo de processamento, pois à medida que se aumenta o número de bits requeridos na conversão, o número de ciclos de processamento aumenta de forma linear.

6 Conclusões

Quatro métodos digitais usados na linearização de sensores e seus circuitos de condicionamento foram analisados e implementados em um DSP. Estes métodos foram analisados e comparados seguindo os critérios de área consumida, tempo de execução e energia média dissipada.

Além destes parâmetros, outros como taxa de aumento do custo área e custo velocidade com o aumento do número de bits, comportamento em freqüência, facilidade de automação e tempo de treinamento foram analisados.

Em todos os casos, as medidas foram obtidas durante a execução dos algoritmos para as duas configurações de hardware. Na primeira utilizou-se um conversor sigma-delta parcialmente implementado em software no DSP, e na outra se utilizou um conversor A/D totalmente externo ao processador.

Baseando-se nos resultados obtidos e relatados neste trabalho, o método recomendado para aplicações embarcadas, onde as restrições de hardware e software são maiores, é o método Polinomial. Não só a dissipação de potência é claramente melhor, mas também a generalidade e o uso em múltiplos contextos são favorecidos por este método.

Além disso, um eventual aumento da resolução final pode ser realizada acompanhada de menor acréscimo de área (o crescimento é linear, enquanto que em outros dois é exponencial), e possivelmente menor potência.

Para trabalhos futuros sugere-se a analise de outros métodos para serem utilizados em aplicações embarcadas, e suas variações no que se relaciona a relação sinal ruído, potência, área e tempo de execução. Também seria interessante analisar o comportamento de cada método, em relação a estes parâmetros, para diferentes tipos de curvas característicasdo sistema sensor, por exemplo, quando o sistema sensor tem um curva de calibração que não possui monotonicidade, ou que apresenta erros do tipo histerese e zona morta.

Referências Bibliográficas

[ADL98] ADLX202 – Datasheet. "Low Cost 62 g Dual Axis *i*MEMS® Accelerometer with Digital Output". Analog devices Inc. 1998.

[ADS95] ADSP-2100 Family User Manual. Analog Devices Inc. September. 1995.

[AZI96] P. M. Aziz, H. V. Sorensen, e J. V. D. Spiegel. *" An Overview of Sigma Delta Converters"*. IEEE Signal Processing Magazine. January 1996.

[BET02] O. Betat e L. Carro. "Comparison of digital linearization methods for embedded sensor interfaces". Aceito para publicação no ISCAS 2002 Proceedings. Arizona. 2002.

[BOL85] W. T. Bolk. *" A general digital linearising method for transducers"*. J. Phys. E: Sci. Instrument, Vol. 18, 1985.

[BRI96] B. Brignell e N. White. "*Intelligent Sensor Systems*". IOP Publishing Ltd. Revised Edition. 1996.

[CAN86] J. C. Candy. *" Decimation for Sigma Delta Modulation"*. IEEE Transactions on Communications, Vol. COM-34, N° 1, January 1986.

[DOE90] E. O. Doebelin. "Measurement Systems: application and design". McGraw-Hill Book Co. c1990.

[FAV87] J. M. Favennec, "*Smart sensors in industry*". J.Phys. E: Sci. Instrum. 20. 1987.

[GEB99] C. H. Gebotys e R. J. Gebotys. "*Statistically based prediction of power dissipation for complex embedded DSP processors*". Ed. Elsevier. Microprocessors and Microsystems. Vol. 23, n° 3, May 1999.

[HAB97] A. Haberli, F. Mayer, D. Jaeggi e H. Baltes. "*IC Microsensors – Between System and Techonology*", Analog and Mixed IC Design, 1997. Proceedings., 1997 2nd IEEE-CAS Region 8 Workshop on, 1997,pp 36 –40.

[HOL98] M. Hölling, M. Thaler e G. Tröster. "*Adaptive Prediction of Sample Values for Digital Transducers"*. IEEE. Zurich. 1998.

[HOR97] G.V. D. Horn and J. H. Huijsing. "*Integrated Smart Sensor Calibration*". Analog Integrated Circuits and Signal Processing. Vol. 14. pp. 207-222. Boston. 1997.

[HOS97] B. J. Hostica, W. Brockherde, and D. Hammerschmidt. "*Silicon Sensor Systems*". Kluwer Academic Publishers. Analog Integrated Circuits and Signal Processing. Vol. 14. pp. 261-273. Boston. 1997.

[HUI94] J. H. Huijsing, F.R. Riedijk e G. v. d. Horn. "*Developments in Integrated Smart Sensors*". Ed. Elsevier. Sensor and Actuators A, vol. 43, pp. 276-288, 1994.

[JAH98] G. P. Jahn e L. Carro. "*A Non-Linear Adaptive Filter for Sensor and Amplifier Linearization*". Proceedings of the SBCCI. Rio de Janerio.1998.

[JAH99] G. P. Jahn, L. Bettin e L. Carro. "*Conversor Analógico/Digital Sigma-Delta*". Relatório de Pesquisa Número 02 do Departamento de Engenharia Elétrica e da Pós-Graduação em Engenharia Elétrica. Porto Alegre. 1999.

[LEU94] B. Leung, "Theory of Σ-Δ Analog to Digital Converters". IEEE International Symposium on Circuit and Systems. Tutorials.1994.

[MAL94] P. Malcovati, C. Azeredo Leme, P. O`Leary, F. Maloberti, e H. Baltes. "*Smart Sensor Interface with A/D Conversion and Programmable Calibration*" .Special Brief Papers. IEEE Journal of Solid State Circuits. Vol. 29, nº 8, August de 1994.

[MID87] S. Middelhoek e S. A. Audet. "*Silicon sensors: full of promises ad pitfalls*". J.Phys. E: Sci. Instrum. 20. 1987.

[NEG00] M. Negreiros e L. Carro. "*Energy Quality Measurement with High Linearity*". Induscon. Porto Alegre.2000.

[OPP89] A. V. Oppenhaeim e R. W. Scahfer, "*Discrete-Time Signal Processing*", Prentice-Hall, Englewood Cliffs, NJ, 1989.

[SCH96] H. Schurer, C.H. Slump and O.E. Herrmann, "*Comparison of Three Methods for Linearization of Electrodynamic Transducers*", Proceedings of the ProRISC/IEEE BeNeLux workshop on Circuits, Systems and Signal Processing, Mierlo, The Netherlands, November 27 & 28,1996, pp. 285-290.

[TIL99] S. J. Tilden, T. E. Linnenbrink e P. J. Green. "*Overview of IEEE-STD-1241 - Standard for Terminology and Test Methods for Analog-to-Digital Converters*". 1999. Instrumentation and Measurement Technology Conference, 1999. IMTC/99. Proceedings of the 16th IEEE , Volume: 3 , pp 1498 –1503, 1999.

[WES85] N. E. Weste e K. Eshraghian, "*Principles of CMOS VLSI design*". Addison-Wesley Publishing Company, 1985.

[WID85] B. Widrow e S. D. Stearms, "*Adaptive Signal Processing*". Pretince-Hall Signal Processing Series, 1985.

ANEXO A - IMPLEMENTAÇÃO SIMULIN K

Na figura A-1 tem-se a topologia utilizada para implementar um modulador sigma-delta em Simulink. Para um sinal de entrada senoidal analógico (x), encontrou-se a função f(x), que é o sinal com as não-linearidades inseridas. Ambos sinais são convertidos pelo modulador sigma-delta em bit-streams, que são carregados no workspace do Matlab5.3. Posteriormente estas variáveis são salvas em arquivos que serão utilizados nas implementações Matlab5.3 dos métodos de linearização.

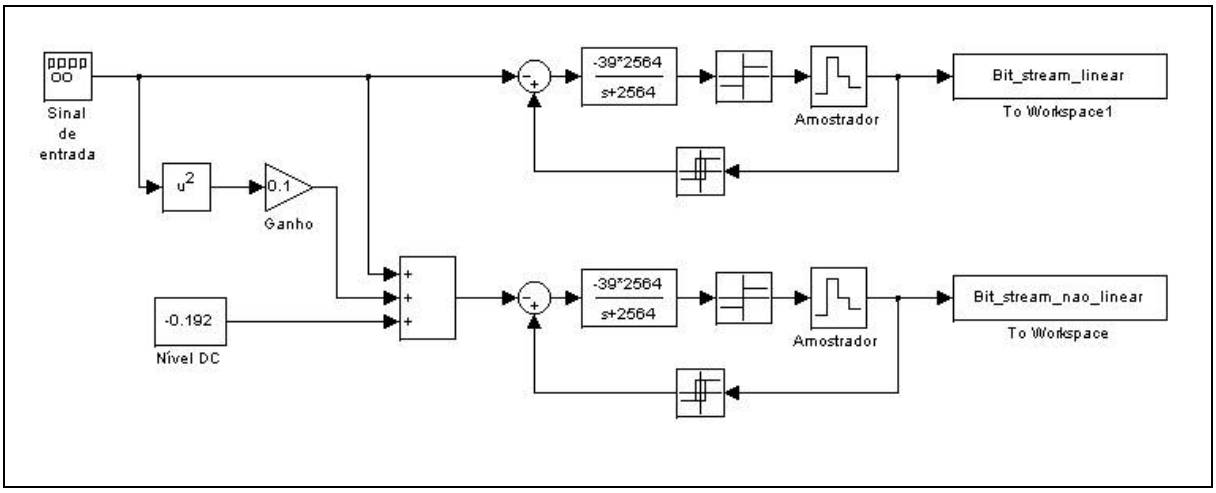

Figura A-1 Topologia utilizada nas simulações Simulink.

ANEXO B - PROGRAMAS IMPLEMENTADOS EM MATLAB5.3

B.1 MÉTODO TABELA

```
%*****************************************************************************
% Algoritmo de Linearização -  Método Tabela                                  *
%                                                                             *
% Modulo: tabelaf1.m                                                         *
% Data   : 25/10/2001                                                         *
% Autor  : Osvaldo Betat                                                      *
%                                                                             *
%*****************************************************************************

clear all;
close all;

% Carregar arquivos com sinais linear e não linear gerados no
% modelo de conversor sigma-delta implementado em Simulink

load 's17hz14vp_d255.mat';
seno_ad_pf  = h(1:2048);

load 'sf1dc_17hz14vp_d255.mat';
senod_ad_pf = h(1:2048);

fs = 256000/255;                    % Freqüência de amostragem após decimação
f = 17;                             % Freqüência do sinal de entrada
npontos = length(senod_ad_pf);

t=0:1/fs:npontos/fs-1/fs;

%****** plota sinais no tempo *************
figure;
stairs(t,seno_ad_pf);
hold
stairs(t,senod_ad_pf,'r');

%****** Gera  FFT's ******************
yh2 = fft(seno_ad_pf'.*hanning(npontos),npontos);
yh3 = fft(senod_ad_pf'.*hanning(npontos),npontos);

%****** Passa para dB *****************
yh_db2 = 20*log10(abs(yh2)/max(abs(yh2)));
yh_db3 = 20*log10(abs(yh3)/max(abs(yh3)));

%****** Plota FFT's ******************
F = (0:fs/npontos:fs-fs/npontos)';

figure;
plot (F,yh_db2,'r')
title('FFT sinal de saida linear - janela hanning');

figure;
plot (F,yh_db3,'b')
title ('FFT Sinal não linear - 2048 pontos - janela hanning');
xlabel('Freqüência [Hz]');
ylabel('Ganho [dB]');

nbits= 8;
```

```
% Chama programa de calculo dos coeficientes
% de correção da tabela

coef_tab_pfixof1;

% Aplica linearização- método tabela

for i = 1:npontos,
     a = senod_ad_pf(i)+1;
if  a > 2^nbits
     a = 2^nbits;
end
if  a < 1
     a = 1;
end
     nc2(i) = saidapf(a);
end

nc2 = nc2';
figure;
stairs(t,nc2);

hold
stairs(t,seno_ad_pf,'r');
stairs(t,senod_ad_pf,'g');

%****** Gera  FFT ******************
yh4 = fft(nc2.*hanning(npontos),npontos);

%****** Passa para dB *****************
yh_db4 = 20*log10(abs(yh4)/max(abs(yh4)));

figure;
plot (F,yh_db4)
title('FFT sinal de saida compensado - janela hanning');

erro1 = (seno_ad_pf - senod_ad_pf)/256 * 100;
erro2 = (seno_ad_pf'- nc2        )/256 * 100;
figure;
plot(erro1,'r');
title('Erro percentual do sinal não linear');

figure;
plot(erro2);
title('Erro percentual do sinal após linearização');



%*************************************************************************
% Algoritmo para cálculo dos coeficientes para o Método Tabela           *
%                                                                        *
% Modulo: coef_tab_pfixof1.m                                             *
% Data  : 25/10/2001                                                     *
% Autor : Osvaldo Betat                                                  *
%                                                                        *
%*************************************************************************


Nbits = 8;            %número de bits de endereços da tabela
Nbits_saida = 8;      %número de bits da palavra digital de saída

Vmax =  1.4;          % excursão do sinal de entrada em amplitude
Vmin = -1.4;
Vpp = Vmax - Vmin;

n = 2^Nbits;
nsaida = 2^Nbits_saida;
LSB = Vpp/n;
x = (Vmin:LSB:Vmax-LSB)';

% a função a ser implementada é inversa a f(x)=(-0.196+x+0.1*x^2)

a=0.1;
b=1;
c=-0.196-x;

for i=1:n,
```

```
   g(i,:)=[a b c(i)];
   j= roots (g(i,:));
   k(i)=j(2);
end

k=k';

saida21=k;
entrada21=x;

figure;
plot(x,entrada21);
hold
plot(x,saida21,'r');

% nova tabela agora convertida para ponto fixo

entradapf = fix  (entrada21/LSB)+128;
saidapf   = round(saida21/LSB)  +128;

for i=1:n,
   if saidapf(i)> 255,
      saidapf(i)= 255;
   end
   if saidapf (i)< 0,
      saidapf (i)= 0;
   end
end

figure
plot(x,entradapf);
hold
plot(x,saidapf,'r');

erro = (-entradapf+saidapf)/256*100;
figure;
plot(erro);
grid on;
title('Erro percentual máximo sem linearização');
```

B.2 MÉTODO INSERIDO NO CONVERSOR

```
%*************************************************************************
% Algoritmo de Linearização - Método Inserido no Conversor                *
%                                                                         *
% Modulo: malo1sinc2.m                                                    *
% Data  : 25/10/2001                                                      *
% Autor : Osvaldo Betat                                                   *
%                                                                         *
%*************************************************************************

clear all;
close all;

% Carregar arquivos com sinais linear e não linear gerados no
% modelo de conversor sigma-delta implementado em Simulink

load 'bit17hz14vp_256k';
y = x(1:600000)';
y = (y+1)*0.5;

load 'bitf1dc_17hz14vp256k';
y_res =x(1:600000)';
y_res=  (y_res+1)*0.5;

j2=length(y);

% Aplicar filtro média móvel de 16 taps

b=ones(1,16);
ja=1;

x1=filter(b,ja,y);
x2=filter(b,ja,y_res);
```

```
x1_1=x1(32:length(x1));
x2_2=x2(32:length(x2));

npontos = length(x2_2);

figure;
stairs(x1_1)
hold
stairs(x2_2,'r')

% aplicar filtragem com tabela linear no  sinais x - sinal linear;
tabela_linear_entrada  = 0:16;
tabela_linear_saida    = tabela_linear_entrada*16;

for i = 1:npontos,
        a = x1_1(i)+1;
        saida_linear(i) = tabela_linear_saida(a);
end

% aplicar filtragem com tabela linear no  sinais x2 - sinal não %linear;
for i = 1:npontos,
        a = x2_2(i)+1;
        saida_nao_linear(i) = tabela_linear_saida(a);
end

% Chamar programa para cálculo dos coeficientes de correção

calc_coef_maloberti;

% aplicar tabela de correção em x2;

for i = 1:npontos,
        a = x2_2(i)+1;
        saida_compensada(i) = saida211_pf(a);
end

% Filtra fluxo de palavras com filtro sinc de segunda ordem

[saida11] = filtro_sinc2_8bits (saida_linear,256);
[saida22] = filtro_sinc2_8bits (saida_nao_linear,256);
[saida33] = filtro_sinc2_8bits (saida_compensada,256);

saida1_1=saida11(1:2048);
saida2_2=saida22(1:2048);
saida3_3=saida33(1:2048);

figure;
stairs(saida1_1)
hold
stairs(saida2_2,'r')
stairs(saida3_3,'g')

erro_max       = (saida2_2 - saida1_1) / 256 * 100;
erro_corrigido = (saida3_3 - saida1_1) / 256 * 100;

figure;
plot (erro_max)
title('Erro percentual sem correção');

figure;
plot (erro_corrigido,'r')
title('Erro percentual após linearização');

yh1 = fft(saida1_1'.*hanning(length(saida1_1)),length(saida1_1));
yh2 = fft(saida2_2'.*hanning(length(saida2_2)),length(saida2_2));
yh3 = fft(saida3_3'.*hanning(length(saida3_3)),length(saida3_3));

%****** Passa para dB *****************
yh_db1 = 20*log10(abs(yh1)/max(abs(yh1)));
yh_db2 = 20*log10(abs(yh2)/max(abs(yh2)));
yh_db3 = 20*log10(abs(yh3)/max(abs(yh3)));

npontos = length(saida1_1);
fs = 256000/255;
F = (0:fs/npontos:fs-fs/npontos)';

figure;
```

```
plot (F,yh_db1)
title('FFT sinal de entrada - janela hanning');

figure;
plot (F,yh_db2,'r')
title('FFT sinal com não linearidades - janela hanning');

figure;
plot (F,yh_db3,'g')
title('FFT sinal linearizado - janela hanning');


%*************************************************************************
% Algoritmo para cálculo dos coeficientes do método Inserido no Conversor    *
%                                                                             *
% Modulo: calc_coef_maloberti.m                                               *
% Data: 25/10/2001                                                            *
% Autor: Osvaldo Betat                                                        *
%                                                                             *
%*************************************************************************


Nbits = 4;                    %numero de bits de endereço da tabela
Nbits_saida = 8;      %numero de bits palavra digital de saida

Vmax = +1.4;
Vmin = -1.4;
Vpp = Vmax - Vmin;

n = 2^Nbits;
nsaida = 2^Nbits_saida;
LSB = Vpp/n;
LSB2=Vpp/nsaida;
x = (Vmin:LSB:Vmax)';

a=0.1;
b=1;
c=-0.196-x;

for i=1:n+1;
   g(i,:)=[a b c(i)];
   j= roots (g(i,:));
   k(i)=j(2);
end

k=k';
saida211=k;
entrada211=x;

figure;
plot(x,entrada211);
hold
plot(x,saida211,'r');

entrada211_pf = round(((entrada211/LSB)+16));
saida211_pf   = round(((saida211/LSB2)+128));
figure;
plot(x,saida211_pf);



%*************************************************************************
% Filtro sinc de segunda ordem com entrada de 8 bits                        *
%                                                                           *
% Modulo: filtro_sinc2_8bits.m                                              *
% Data: 25/10/2001                                                          *
% Autor: Osvaldo Betat                                                      *
%                                                                           *
%*************************************************************************

function [out1] = filtro_sinc2_8bits(y,d);

j2=length(y);
cont = 0;
acc  = 0;
coef = 0;
for i = 1:j2;
```

```
   acc = acc + y(i) * (coef+1);  % reservar 24 bits
   cont = cont + 1;
   if cont < (d/2)
      coef = coef + 1;
   end
   if cont >= (d/2)
      coef = coef - 1;
   end
   if cont == d-1;
      out(i)= acc;
      acc  = 0;
      cont = 0;
      coef =0;
   end
end
out1 = out(d-1:d-1:j2);
out1 = fix(out1./128^2);
for i = 1:length(out1),
   if out1(i)==256,
      out1(i)=255;
   end
end
```

B.3 MÉTODO POLINOMIAL

```
%*************************************************************************
% Algoritmo de Linearização - Método Polinomial                          *
%                                                                        *
% Modulo: dehornf1.m                                                     *
% Data: 25/10/2001                                                       *
% Autor: Osvaldo Betat                                                   *
%                                                                        *
%*************************************************************************

clear all;
close all;

LSB = 2.8 / 2^8;
fundo_escala = 2^8;

a = -1.4:LSB:(1.4-LSB);

b= - 0.196 + a + 0.1*a.^2;

ad_pf = round(a /(2.8/2^8));
bd_pf = round(b /(2.8/2^8));

figure;
plot(ad_pf);
hold
plot(bd_pf,'r');

erro30=((ad_pf+128)-(bd_pf+128))/255*100;
figure;
plot(erro30)

% Escolha dos pontos de calibração com base na observação das curvas
% futuramente deve-se melhorar o algoritmo de forma as calcular os pontos
% automaticamente.

x1 = 1;
x2 = 256;
x3 = 180;
x4 = 95;

y1 = ad_pf(x1);              % primeiro ponto de calibração, sinal desejado
y2 = ad_pf(x2);              % segundo ponto de calibração, sinal desejado
y3 = ad_pf(x3);              % terceiro ponto de calibração, sinal desejado
y4 = ad_pf(x4);

a1 = y1 - bd_pf(x1);
```

```
h1 = bd_pf + a1;

figure;
stairs(h1)
hold
stairs(ad_pf,'m');
stairs(bd_pf,'y')

a2 = round(y2-h1(x2))/(h1(x2)-y1);
h2 = round(h1 + a2 * (h1 - y1));
stairs(h2,'r')

a3 = round(y3-h2(x3))/((h1(x3)-y1)*(h2(x3)-y2));
h3 = round(h2 + a3*(((h1-y1).*(h2-y2))));
stairs(h3,'g');

a4 = round(y4 - h3(x4))/((h1(x4)-y1)*(h2(x4)-y2)*(h3(x4)-y3));
h4 = round(h3 + a4*((h1-y1).*(h2-y2).*(h3-y3)));
stairs (h4,'k')

erro2 = (ad_pf-h3)/fundo_escala*100;
erro3 = (ad_pf-h4)/fundo_escala*100;

figure;
plot(erro2);
hold
plot(erro3,'r')

% Até aqui encontramos os coeficientes,
% a seguir vamos aplica-los em regime a função senoidal.

load 's17hz14vp_d255.mat';
pf1  = h(1:2048);

load 'sf1dc_17hz14vp_d255.mat';
pf2 = h(1:2048);

fs = 256000/255;
f = 17;
amostras = length(pf2);

t=0:1/fs:amostras/fs-1/fs;

figure;
stairs(t,pf1);
hold
stairs(t,pf2,'r');

npontos = amostras;
pf1 = pf1-128;
pf2 = pf2-128;

h1  = pf2 + a1;
xx1 = (a2 * (h1 - y1));
h2  = h1 + xx1 ;

xx2 = round(a3*(((h1-y1).*(h2-y2))));
h3  = h2 + xx2;

xx3 = (a4*((h1-y1).*(h2-y2).*(h3-y3)));
h4  = round(h3 + xx3) ;

pf1 = pf1 ;
pf2 = pf2 ;
h3  = h3  ;
h4  = h4  ;

erro_max=(pf1-pf2)/fundo_escala*100;
erro2 = (seno_ad_pf-h3)/fundo_escala*100;
erro3 = (pf1-h4)/fundo_escala*100;

figure;
plot(erro_max,'r');
title('Erro percentual antes da linearização');

figure;
plot(erro2);
```

```
hold
plot(erro3);
title('Erro percentual após a linearização');

npontos = length(pf1);
F = (0:fs/npontos:fs-fs/npontos)';

figure;
stairs(t,pf1)
hold
stairs(t,h4,'r');
stairs(t,pf2,'g')
title('Sinais no domínio tempo');

yh1 = fft(pf1'.*hanning(length(pf1)),npontos);
yh2 = fft(pf2'.*hanning(length(pf1)),npontos);
yh3 = fft(h4' .*hanning(length(pf1)),npontos);

%****** Passa para dB *****************
yh_db1 = 20*log10(abs(yh1)/max(abs(yh1)));
yh_db2 = 20*log10(abs(yh2)/max(abs(yh2)));
yh_db3 = 20*log10(abs(yh3)/max(abs(yh3)));

figure;
plot (F,yh_db1,'r')
title('FFT sinal de saida linear - janela hanning');

figure;
plot (F,yh_db2,'g')
title('FFT sinal de saida não linear - janela hanning');

figure;
plot (F,yh_db3)
title('FFT sinal de saida corrigido - janela hanning');
```

B.4 MÉTODO COM FILTRO ADAPTATIVO NÃO LINEAR

```
%*************************************************************************
% Algoritmo de Linearização - Método Adaptativo não linear               *
%                                                                        *
% Modulo: adapt1.m                                                       *
% Data: 25/10/2001                                                       *
% Autor: Osvaldo Betat                                                   *
%                                                                        *
%*************************************************************************

clear all;
close all;

% Fase de treinamento com ruído branco
t=30000;
vi =2.8*(rand(1, t)-0.5);

% função não linear
f1 =vi + 0.1* vi.^2;

vref=vi(1:length(vi)-1);
fin= f1(2:length(vi));

h1=zeros(1,10);
h2=zeros(1,4);
h3=zeros(1,4);
u1=0.04;
u2=0.00125;
u3=0.005;

% Executa algoritmo LMS

[y,e,h1,h2,h3] = LMSNL123(fin,h1,h2,h3,vref,u1,u2,u3);%

% gráficos dos sinais de interesse
figure
plot(e)
```

```
title('sinal de erro');
figure
plot(vref)
title('sinal de entrada(blue) - sinal de saida (red)');
hold
plot(y,'r')
figure
plot(h1)
title('coeficientes')
figure
plot((vref-y)/1.4*100)


%************************************************************************
% Algoritmo LMS para cálculo dos coeficientes não lineares               *
%                                                                        *
% Módulo: lmsnl123.m                                                     *
% Data: 25/10/2001                                                       *
% Autor: Osvaldo Betat                                                   *
% Baseado na implementação realizada por G. Jahn em 03/11/97             *
%                                                                        *
%************************************************************************

function [y,e,h1,h2,h3] = LMSNL123(x,h1,h2,h3,r,u1,u2,u3)
%  LMS  LMS Non-linear Adaptative Digital filter.
%       [Y,E,Hf1,Hf2] = LMS(X,H,R,u) runs the adaptative filter starting with
%       the coefficient vector H. Vector X is the data applied
%       to the filter input. Vector R contains the reference
%       data. The filter is a "Direct Form I" implementation
%       of the standard difference equation:
%
%       u is the step size factor. E is the error vector defined by
%       e(n) = r(n) - y(n)
%
%       Last updated 03/11/97 by G. Jahn
%
%        comentários internos
%       x e' o vetor de entrada do filtro
%       h e' o vetor de coeficientes
%       y e a saida do filtro
%       d e' o vetor de linha de atraso
%       i e' o iterador
%       n e' o numero de coeficientes
%       r e' o vetor de referencia
%       e e' o vetor de erro

% inicializações
y =[]; y1=[]; y2=[]; y3=[]; e =[];

if length(x)~=length(r)
   error('Vectors X and R must have the same size.');
end

if length(x)>=length(h1) | length(x)>=length(h2)
% se o vetor de dados for maior ou igual que o de coef.
   n1 = length(h1);          % comprimento da linha de atraso
   n2 = length(h2);          % comprimento da linha de atraso
   n3 = length(h3);   % comprimento da linha de atraso

   d1 = zeros(1,n1);  % valor inicial da linha de atraso
   d2 = zeros(1,n2);    % valor inicial da linha de atraso
   d3 = zeros(1,n3);    % valor inicial da linha de atraso

 for i=1:length(x)
     d1(1)  = x(i);    % nova amostra na linha de atraso
     d2(1)  = x(i)^2;  % nova amostra na linha de atraso
     d3(1)  = x(i)^3;  % nova amostra na linha de atraso
     y1(i)  = h1*d1'; % calcula saida
     y2(i)  = h2*d2';
     y3(i)  = h3*d3';
     y(i)   = y1(i)+y2(i)+y3(i);
     e(i)   = r(i) - y(i);      % calcula erro atual
     h1     = h1 + u1*e(i)*d1;  % atualiza coeficiente h via LMS
     h2     = h2 + u2*e(i)*d2;  % atualiza coef. h2 via LMS quadrático
     h3     = h3 + u3*e(i)*d3;  % atualiza coef. h2 via LMS quadrático
     d1(2:n1) = d1(1:n1-1);     % desloca amostras na linha de atraso
     d2(2:n2) = d2(1:n2-1);     % desloca amostras na linha de atraso
```

```
     d3(2:n3) = d3(1:n3-1);     % desloca amostras na linha de atraso
 end
else
   error('Vectors H cannot be greater than X.');
end
```

ANEXO C - PROGRAMAS IMPLEMENTADOS PARA ADSP2181

C.1 MÉTODO TABELA COM SIGMA-DELTA

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Elétrica
%
% Modulo     : tabsd2.dsp
%
% Objetivos  : Implementar aquisição de um bit stream com kit analog devices (ADSP2181).
%
% Comentários:
%
% Autor      : Osvaldo Betat
%
% Histórico  : 14.03.2001 - Criação do modulo.
%              22.05.2001 - modificado para aquisição sincronizada com interrupção do
%                           clock do sigma-delta.
%             02.10.2001 - modificado para filtrar o bitstream com um filtro sinc2
%             03.10.2001 - modificado para método de linearização tabela convencional.
%
*****************************************************************************/

.module/RAM/ABS=0 tabsd2;
.include <coef2.h>;

/*****************************************************************************
*                    DECLARACAO DE CONSTANTES                                *
*  mapeamento em memória dos registradores de controle do adsp-2181          *
*****************************************************************************/
.const  IDMA=                   0x3fe0;
.const  BDMA_BIAD=              0x3fe1;
.const  BDMA_BEAD=              0x3fe2;
.const  BDMA_BDMA_Ctrl=         0x3fe3;
.const  BDMA_BWCOUNT=           0x3fe4;
.const  PFDATA=                 0x3fe5;
.const  PFTYPE=                 0x3fe6;
.const  SPORT1_Autobuf=         0x3fef;
.const  SPORT1_RFSDIV=          0x3ff0;
.const  SPORT1_SCLKDIV=         0x3ff1;
.const  SPORT1_Control_Reg=     0x3ff2;
.const  SPORT0_Autobuf=         0x3ff3;
.const  SPORT0_RFSDIV=          0x3ff4;
.const  SPORT0_SCLKDIV=         0x3ff5;
.const  SPORT0_Control_Reg=     0x3ff6;
.const  SPORT0_TX_Channels0=    0x3ff7;
.const  SPORT0_TX_Channels1=    0x3ff8;
.const  SPORT0_RX_Channels0=    0x3ff9;
.const  SPORT0_RX_Channels1=    0x3ffa;
.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  DM_Wait_Reg=            0x3ffe;
.const  System_Control_Reg=     0x3fff;

/*****************************************************************************
*             DECLARACAO DE VARIAVEIS e BUFFERS                              *
*****************************************************************************/
.VAR/DM/RAM/CIRC/ABS=0x1000   samples[2000];
.VAR/DM/RAM/ABS=0x17d0        index;
.VAR/DM/RAM                   VoltaAoMonitor;
```

```
.VAR/DM/RAM                  soma;
.VAR/DM/RAM                  acumulador;
.VAR/DM/RAM                  contador;
.VAR/DM/RAM                  coeficientes[256];

/*****************************************************************************
*             INICIALIZACAO DE VARIAVEIS e BUFFERS                           *
*****************************************************************************/
.INIT                   index:0;
.INIT                   VoltaAoMonitor:  0;
.INIT                   soma:0;
.INIT                   acumulador:0;
.INIT                   contador:0;

/*****************************************************************************
*             TABELA DE VETORES DE INTERRUPCAO                               *
*****************************************************************************/
        jump start;     rti; rti; rti;      {00: reset }
        jump leitura;   rti; rti; rti;      {04: IRQ2 }
        rti;            rti; rti; rti;      {08: IRQL1 }
        rti;            rti; rti; rti;      {0c: IRQL0 }
        rti;            rti; rti; rti;      {10: SPORT0 tx }
        rti;            rti; rti; rti;      {14: SPORT0 rx }
        jump retorno;   rti; rti; rti;      {18: IRQE }
        rti;            rti; rti; rti;      {1c: BDMA }
        rti;            rti; rti; rti;      {20: SPORT1 tx or IRQ1 }
        rti;            rti; rti; rti;      {24: SPORT1 rx or IRQ0 }
        rti;            rti; rti; rti;      {28: timer }
        rti;            rti; rti; rti;      {2c: power down }


/*****************************************************************************
%                           START
*****************************************************************************
% Objetivos  :
%
% Comentários:
%
% Entradas   :
%
% Saidas     :
%
% Altera     :
%
*****************************************************************************/
start:
   dis ints;
   ena ints;
   imask = b#1000010000;
       {     |||||||||+ | timer                   imask    - cada bit do imask habilita
             ||||||||+- | SPORT1 rec or IRQ0                (1) ou desabilita (0) cada
             |||||||+-- | SPORT1 trx or IRQ1               interrupção individualmente;
             ||||||+--- | BDMA                             Após o reset permanece em
             |||||+---- | IRQE                             estado 0.
             ||||+----- | SPORT0 rec
             |||+------ | SPORT0 trx
             ||+------- | IRQL0
             |+-------- | IRQL1
             +--------- | IRQ2
       }
   icntl = b#00100;
       {     ||||+- | IRQ0: 0=level, 1=edge
             |||+-- | IRQ1: 0=level, 1=edge
             ||+--- | IRQ2: 0=level, 1=edge
             |+---- 0
             +----- | IRQ nesting: 0=disabled, 1=enabled
       }
   ax0 = b#0000000000000000;   dm (System_Control_Reg) = ax0;
       {   +-/!||+-----/+-/- | program memory wait states
           |  !|||           | 0
           |  !|||           |
           |  !||+---------- 0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
```

```
           |  !||           0
           |  !|+---------- SPORT1 1=serial, 0=FI, FO, IRQ0, IRQ1, SCLK
           |  !+----------- SPORT1 1=enabled, 0=disabled
           |  +============ SPORT0 1=enabled, 0=disabled
           +--------------- 0
                            0
                            0
       }

/*****************************************************************************
*                       SPORT1 - FLAGS                                       *
*****************************************************************************/
   ax0 = b#00000000;
   dm (PFTYPE) = ax0;        {determina PF7-0 pins como entradas (0=input e 1=output) }
   mstat = b#1000000;
       {     ||||||+- | Data register bank select
             |||||+-- | FFT bit reverse mode (DAG1)
             ||||+--- | ALU overflow latch mode, 1=sticky
             |||+---- | AR saturation mode, 1=saturate, 0=wrap
             ||+----- | MAC result, 0=fractional, 1=integer
             |+------ | timer enable
             +------- | GO MODE
       }
       i1 = ^samples;
       l1 = %samples;
       m1 = 1;
espera:
       idle;

       /* Testa se continua processando ou retorna ao Monitor */

testa_retorno:
       ax0 = dm(VoltaAoMonitor);
        ar = PASS ax0;
       if EQ jump espera;

       /* retorna ao programa monitor  */

fim:
       /* Volta ao monitor ...*/
       dis ints;
       rts;

/* Leitura, filtragem e armazenagem do sinal em área de memória */

leitura:
       ax0 = dm(PFDATA);
       af = tstbit 0 of ax0;     {testa bit0 de PFDATA - sinal de entrada(bitstream) }
       if eq jump contagem;      {se for 0 pula para contagem se for 1 incrementa
                                    acumulador e depois conta}
soma1:
       ay0 = dm(acumulador);
       ax0 = dm(soma);
       af = ax0 + ay0;
       ar = af + 1;
       dm(acumulador) = ar;
contagem:
       ar = dm(contador);
       ar = ar + 1;
       dm(contador) = ar;
       ay0 = 128;
       ar = ar - ay0;
       if ge jump soma2;
       ar = dm(soma);
       ar = ar + 1;
       dm(soma) = ar;
       rti;
soma2:
       ar = dm(soma);
       ar = ar - 1;
       dm(soma) = ar;

       ar = dm(contador);
       ay0 = 254;
       ar = ar - ay0;
       if ne rti;
```

```
        sr1 = 0;
        sr0 = dm(acumulador);
        sr = ashift sr0 by -6 (lo);   /* divide por 64 >> saida de 8 bits em sr0 */

inic_contador:
        ax0 = 0;
        dm (soma)       = ax0;
        dm (contador)   = ax0;
        dm (acumulador) = ax0;

/*****************************************************************************
Tabela de conversão
*****************************************************************************/
        i0 = ^coeficientes;
        m0 = sr0;
        l0 = 0;
        modify(i0,m0);
        ax0 = dm(i0,m0);

/* Salva dados no buffer circular de tamanho 2000 */

        dm(i1,m1) = ax0;
        dm(index) = i1;

/*   colocar aqui rotina de escrita no D/A se necessario*/
        rti;

/*****************************************************************************
*     Atende interrupção de retorno ao monitor - IRQE
*****************************************************************************/
retorno:
        ax0 = 1;
        dm(VoltaAoMonitor) = ax0;
        rti;

.endmod;
```

Arquivo: COEF2.H

```
/*****************************************************************************
*     Coeficientes de calibração encontrados usando rotina Matlab           *
*****************************************************************************/

.init          coeficientes:
  0,     0,     0,     0,     0,     0,     0,     0,
  0,     0,     0,     0,     0,     0,     0,     0,
  0,     0,     0,     2,     3,     4,     6,     7,
  8,    10,    11,    13,    14,    15,    17,    18,
 19,    20,    22,    23,    24,    26,    27,    28,
 29,    31,    32,    33,    34,    36,    37,    38,
 39,    41,    42,    43,    44,    46,    47,    48,
 49,    50,    52,    53,    54,    55,    57,    58,
 59,    60,    61,    62,    64,    65,    66,    67,
 68,    69,    70,    72,    73,    74,    75,    76,
 77,    78,    79,    81,    82,    83,    84,    85,
 86,    87,    88,    89,    90,    92,    93,    94,
 95,    96,    97,    98,    99,   100,   102,   102,
103,   104,   105,   107,   108,   109,   110,   111,
112,   113,   114,   115,   116,   117,   118,   119,
120,   121,   122,   123,   124,   125,   126,   127,
128,   129,   130,   131,   132,   133,   134,   135,
136,   137,   138,   139,   140,   141,   142,   143,
144,   144,   146,   147,   147,   148,   149,   150,
151,   152,   153,   154,   155,   156,   157,   158,
159,   160,   161,   162,   163,   163,   164,   165,
166,   167,   168,   169,   170,   171,   172,   173,
173,   174,   176,   176,   177,   178,   179,   180,
181,   182,   183,   183,   184,   185,   186,   187,
188,   189,   190,   191,   192,   192,   193,   194,
195,   196,   197,   198,   198,   199,   200,   201,
202,   203,   204,   204,   205,   206,   207,   208,
209,   209,   211,   211,   212,   213,   214,   214,
216,   216,   217,   218,   219,   219,   221,   221,
222,   223,   224,   224,   226,   226,   227,   228,
229,   229,   231,   231,   232,   233,   234,   234,
```

```
235,   236,   237,   238,   238,   239,   240,   241;
```

C.2 MÉTODO INSERIDO NO CONVERSOR COM SIGMA-DELTA

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Elétrica
%
% Modulo     : malo6.dsp
%
% Objetivos  : Implementar aquisição de um bit stream com kit analog devices (ADSP2181).
%
% Comentários:
%
% Autor      : Osvaldo Betat
%
% Histórico  : 14.03.2001 - Criação do modulo.
%              22.05.2001 - modificado para aquisição sincronizada com
%                           interrupção do clock do sigma-delta.
%              02.10.2001 - modificado para filtrar o bitstream com um filtro sinc2
%              08.10.2001 - inserido linearização por Maloberti
*****************************************************************************/

.module/RAM/ABS=0 malo6;

.include <t_comp14.h>;

/*****************************************************************************
*                     DECLARACAO DE CONSTANTES                               *
*  mapeamento em memória dos registradores de controle do adsp-2181          *
*****************************************************************************/
.const  IDMA=                   0x3fe0;
.const  BDMA_BIAD=              0x3fe1;
.const  BDMA_BEAD=              0x3fe2;
.const  BDMA_BDMA_Ctrl=         0x3fe3;
.const  BDMA_BWCOUNT=           0x3fe4;
.const  PFDATA=                 0x3fe5;
.const  PFTYPE=                 0x3fe6;
.const  SPORT1_Autobuf=         0x3fef;
.const  SPORT1_RFSDIV=          0x3ff0;
.const  SPORT1_SCLKDIV=         0x3ff1;
.const  SPORT1_Control_Reg=     0x3ff2;
.const  SPORT0_Autobuf=         0x3ff3;
.const  SPORT0_RFSDIV=          0x3ff4;
.const  SPORT0_SCLKDIV=         0x3ff5;
.const  SPORT0_Control_Reg=     0x3ff6;
.const  SPORT0_TX_Channels0=    0x3ff7;
.const  SPORT0_TX_Channels1=    0x3ff8;
.const  SPORT0_RX_Channels0=    0x3ff9;
.const  SPORT0_RX_Channels1=    0x3ffa;
.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  DM_Wait_Reg=            0x3ffe;
.const  System_Control_Reg=     0x3fff;

/*****************************************************************************
*              DECLARACAO DE VARIAVEIS e BUFFERS                             *
*****************************************************************************/
.VAR/DM/RAM/CIRC/ABS=0x1000   samples[2000];
.VAR/DM/RAM/ABS=0x17d0               index;
.VAR/DM/RAM                  VoltaAoMonitor;
.VAR/DM/RAM                  janela;
.VAR/DM/RAM                  contador;
.VAR/DM/RAM/CIRC             buffer_dados[16];
.VAR/DM/RAM                  coeficientes[17];

/*****************************************************************************
*              INICIALIZACAO DE VARIAVEIS e BUFFERS                          *
*****************************************************************************/
.INIT                 index          :0;
.INIT                 VoltaAoMonitor :0;
.INIT                 janela         :1;
.INIT                 contador       : 0;
```

```
.INIT                buffer_dados   : 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;

/*****************************************************************************
*             TABELA DE VETORES DE INTERRUPCAO                               *
*****************************************************************************/
jump start;          rti; rti; rti;     {00: reset }
jump leitura;        rti; rti; rti;     {04: IRQ2 }
rti;                 rti; rti; rti;     {08: IRQL1 }
rti;                 rti; rti; rti;     {0c: IRQL0 }
rti;                 rti; rti; rti;     {10: SPORT0 tx }
rti;                 rti; rti; rti;     {14: SPORT0 rx }
jump retorno;        rti; rti; rti;     {18: IRQE}
rti;                 rti; rti; rti;     {1c: BDMA }
rti;                 rti; rti; rti;     {20: SPORT1 tx or IRQ1 }
rti;                 rti; rti; rti;     {24: SPORT1 rx or IRQ0 }
rti;                 rti; rti; rti;     {28: timer }
rti;                 rti; rti; rti;     {2c: power down }

/*****************************************************************************
%                           START                                            *
*****************************************************************************/
start:
   dis ints;
   ena ints;
   imask = b#1000010000;
       {     |||||||||+ | timer                    imask    - cada bit do imask habilita
             ||||||||+- | SPORT1 rec or IRQ0                 (1) ou desabilita (0) cada
             |||||||+-- | SPORT1 trx or IRQ1                interrupção individualmente;
             ||||||+--- | BDMA                              Após o reset permanece em
             |||||+---- | IRQE                              estado 0.
             ||||+----- | SPORT0 rec
             |||+------ | SPORT0 trx
             ||+------- | IRQL0
             |+-------- | IRQL1
             +--------- | IRQ2
       }
   icntl = b#00100;
       {     ||||+- | IRQ0: 0=level, 1=edge
             |||+-- | IRQ1: 0=level, 1=edge
             ||+--- | IRQ2: 0=level, 1=edge
             |+---- 0
             +----- | IRQ nesting: 0=disabled, 1=enabled
       }
   ax0 = b#0000000000000000;   dm (System_Control_Reg) = ax0;
       {   +-/!||+-----/+-/- | program memory wait states
           |  !|||           | 0
           |  !|||           |
           |  !||+---------- 0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !|+----------- SPORT1 1=serial, 0=FI, FO, IRQ0, IRQ1, SCLK
           |  !+------------ SPORT1 1=enabled, 0=disabled
           |  +============= SPORT0 1=enabled, 0=disabled
          +---------------- 0
                            0
                            0
       }

/*****************************************************************************
*                       SPORT1 - FLAGS                                       *
*****************************************************************************/

   ax0 = b#00000000;
   dm (PFTYPE) = ax0;        {determina PF7-0 pins como entradas (0=input e 1=output) }
   mstat = b#1000000;
       {     ||||||+- | Data register bank select
             |||||+-- | FFT bit reverse mode (DAG1)
             ||||+--- | ALU overflow latch mode, 1=sticky
             |||+---- | AR saturation mode, 1=saturate, 0=wrap
             ||+----- | MAC result, 0=fractional, 1=integer
             |+------ | timer enable
             +------- | GO MODE
        }
```

```
        i1 = ^samples;
        l1 = %samples;
        m1 = 1;
        i5 = ^buffer_dados;
        l5 = %buffer_dados;
        m5 = 1;
        l4 = 0;
        mr = 0;
espera:
        idle;

/* Testa se continua processando ou retorna ao Monitor */
testa_retorno:
        ax0 = dm(VoltaAoMonitor);
        ar = PASS ax0;
        if EQ jump espera;

/* retorna ao programa monitor  */
fim:
/* Volta ao monitor ...*/
        dis ints;
        rts;
/* Leitura, filtragem e armazenagem do sinal em área de memória */

leitura:
        ax0 = dm(PFDATA);
        ar  = ax0 and 1;

/* Aplica filtro sem decimação do tipo média móvel de 4 bits */
        dm(i5,m5) = ar;
        ar=0;
        cntr = 16;
        ay0 = 0;
        i4 =^buffer_dados;
        ay0 = dm(i4,m5);
        do soma until ce;
soma:   ar = ar + ay0 , ay0 = dm(i4,m5);

/* Aplica correção utilizando uma tabela de 4x8bits */
        i4 = ^coeficientes;
        m4 = ar;
        modify (i4,m4);
        ar = dm(i4,m4);  {ax0 recebe valor corrigido vindo da tabela}

/* Aplica filtro de 8 bits com decimação de 256  */
        my0 = dm(janela);
        mr  = mr + ar * my0 (uu);
        ar = dm(contador);
        ar = ar + 1;
        dm (contador) = ar;
        ay0 = 128;
        ar = ar - ay0;
        if ge jump direcao;
        ay0 = dm(janela);
        ar  = ay0 + 1;
        dm(janela) = ar;
        rti;
direcao:
        ay0 = dm(janela);
        ar  = ay0 - 1;
        dm(janela) = ar;
        ar = dm(contador);
        ay0 = 255;
        ar = ar - ay0;
        if ne rti;
inic_contador:
        ax0 = 1;
        dm (janela)     = ax0;
        ax0 = 0;
        dm (contador)   = ax0;
        se = -15;
        sr = ashift mr1 (hi);
        sr = sr or lshift mr0 (lo);   {resultado em sr0 = Q16.0*2^7}
        mr = 0;

/* Salva dados no buffer circular de tamanho 2000 */
```

```
       dm(i1,m1) = sr0;
       dm(index) = i1;

/*   colocar aqui rotina de escrita no D/A se necessario*/
       rti;

/*****************************************************************************
*     atende interrupção de retorno ao monitor - IRQE
*****************************************************************************/
retorno:
       ax0 = 1;
       dm(VoltaAoMonitor) = ax0;
       rti;

.endmod;
```

Arquivo: t_comp14.h

```
/*****************************************************************************
*     Coeficientes de calibração encontrados usando rotina Matlab            *
*****************************************************************************/


.INIT          coeficientes:    0,    0,   19,   39,   59,   77,   95,  112,  128,
                              144,  159,  174,  188,  202,  216,  229,  242;
```

C.3 MÉTODO POLINOMIAL COM SIGMA-DELTA

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Elétrica
%
% Modulo     : hornsd2.dsp
%
% Objetivos  : Implementar aquisição de um bit stream com kit analog devices (ADSP2181).
%
% Comentários:
%
% Autor      : Osvaldo Betat
%
% Histórico  : 14.03.2001 - Criação do modulo.
%              22.05.2001 - modificado para aquisição sincronizada com
%                           interrupção do clock do sigma-delta.
%              02.10.2001 - modificado para filtrar o bitstream com um filtro sinc2
%              02.10.2001 - modificado para executar algoritmo de linearização polinomial
*****************************************************************************/

.module/RAM/ABS=0 hornsd2;

/*****************************************************************************
*                     DECLARACAO DE CONSTANTES                               *
*  mapeamento em memória dos registradores de controle do adsp-2181          *
*****************************************************************************/
.const  IDMA=                   0x3fe0;
.const  BDMA_BIAD=              0x3fe1;
.const  BDMA_BEAD=              0x3fe2;
.const  BDMA_BDMA_Ctrl=         0x3fe3;
.const  BDMA_BWCOUNT=           0x3fe4;
.const  PFDATA=                 0x3fe5;
.const  PFTYPE=                 0x3fe6;
.const  SPORT1_Autobuf=         0x3fef;
.const  SPORT1_RFSDIV=          0x3ff0;
.const  SPORT1_SCLKDIV=         0x3ff1;
.const  SPORT1_Control_Reg=     0x3ff2;
.const  SPORT0_Autobuf=         0x3ff3;
.const  SPORT0_RFSDIV=          0x3ff4;
.const  SPORT0_SCLKDIV=         0x3ff5;
.const  SPORT0_Control_Reg=     0x3ff6;
.const  SPORT0_TX_Channels0=    0x3ff7;
.const  SPORT0_TX_Channels1=    0x3ff8;
.const  SPORT0_RX_Channels0=    0x3ff9;
```

```
.const  SPORT0_RX_Channels1=    0x3ffa;
.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  DM_Wait_Reg=            0x3ffe;
.const  System_Control_Reg=     0x3fff;

/*****************************************************************************
*             DECLARACAO DE VARIAVEIS e BUFFERS                              *
*****************************************************************************/
.VAR/DM/RAM/CIRC/ABS=0x1000  samples[2000];
.VAR/DM/RAM/ABS=0x17d0              index;
.VAR/DM/RAM                  VoltaAoMonitor;
.VAR/DM/RAM                  soma;
.VAR/DM/RAM                  acumulador;
.VAR/DM/RAM                  contador;
.var/dm/ram/circ             coef[4];
.var/dm/ram/circ             y_pad [3];
.VAR/DM/RAM                  auxiliar;
.var/dm/ram                  aux2;

/*****************************************************************************
*             INICIALIZACAO DE VARIAVEIS e BUFFERS                           *
*****************************************************************************/
.INIT                  index:0;
.INIT                  VoltaAoMonitor:  0;
.INIT                 soma:0;
.INIT                  acumulador:0;
.INIT                  contador:0;
.INIT                  coef :-18, 0, -17201, 11377; {a1, a2*2^16, a3*2^24, a4*2^32}
.INIT                  y_pad:-128, 32256, 10240;

/*****************************************************************************
*             TABELA DE VETORES DE INTERRUPCAO                               *
*****************************************************************************/
jump start;          rti; rti; rti;     {00: reset }
jump leitura;        rti; rti; rti;     {04: IRQ2 }
rti;                 rti; rti; rti;     {08: IRQL1 }
rti;                 rti; rti; rti;     {0c: IRQL0 }
rti;                 rti; rti; rti;     {10: SPORT0 tx }
rti;                 rti; rti; rti;     {14: SPORT0 rx }
jump retorno;        rti; rti; rti;     {18: IRQE}
rti;                 rti; rti; rti;     {1c: BDMA }
rti;                 rti; rti; rti;     {20: SPORT1 tx or IRQ1 }
rti;                 rti; rti; rti;     {24: SPORT1 rx or IRQ0 }
rti;                 rti; rti; rti;     {28: timer }
rti;                 rti; rti; rti;     {2c: power down }

/*****************************************************************************
%                          START                                              *
*****************************************************************************/
start:
dis ints;
   ena ints;
   imask = b#1000010000;
       {     |||||||||+ | timer                    imask    - cada bit do imask habilita
             ||||||||+- | SPORT1 rec or IRQ0                 (1) ou desabilita (0) cada
             |||||||+-- | SPORT1 trx or IRQ1                interrupção individualmente;
             ||||||+--- | BDMA                              Após o reset permanece em
             |||||+---- | IRQE                              estado 0.
             ||||+----- | SPORT0 rec
             |||+------ | SPORT0 trx
             ||+------- | IRQL0
             |+-------- | IRQL1
             +--------- | IRQ2
       }
   icntl = b#00100;
       {     ||||+- | IRQ0: 0=level, 1=edge
             |||+-- | IRQ1: 0=level, 1=edge
             ||+--- | IRQ2: 0=level, 1=edge
             |+---- 0
             +----- | IRQ nesting: 0=disabled, 1=enabled
       }


   ax0 = b#0000000000000000;   dm (System_Control_Reg) = ax0;
       {   +-/!||+-----/+-/- | program memory wait states
```

```
           |  !|||           | 0
           |  !|||           |
           |  !||+---------- 0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !|+----------- SPORT1 1=serial, 0=FI, FO, IRQ0, IRQ1, SCLK
           |  !+------------ SPORT1 1=enabled, 0=disabled
           |  +============= SPORT0 1=enabled, 0=disabled
          +---------------- 0
                            0
                            0
      }

/*****************************************************************************
*                        SPORT1 - FLAGS                                      *
*****************************************************************************/
   ax0 = b#00000000;
   dm (PFTYPE) = ax0;        {determina PF7-0 pins como entradas (0=input e 1=output) }
   mstat = b#1000000;
       {     ||||||+- | Data register bank select
             |||||+-- | FFT bit reverse mode (DAG1)
             ||||+--- | ALU overflow latch mode, 1=sticky
             |||+---- | AR saturation mode, 1=saturate, 0=wrap
             ||+----- | MAC result, 0=fractional, 1=integer
             |+------ | timer enable
             +------- | GO MODE
       }

i1 = ^samples;
       l1 = %samples;
       m1 = 1;
espera:
       idle;

/* Testa se continua processando ou retorna ao Monitor */
testa_retorno:
       ax0 = dm(VoltaAoMonitor);
       ar = PASS ax0;
       if EQ jump espera;

/* retorna ao programa monitor  */
fim:
/* Volta ao monitor ...*/
       dis ints;
       rts;

/* Leitura, filtragem e armazenagem do sinal em área de memória */

leitura:
       ax0 = dm(PFDATA);
       af = tstbit 0 of ax0; {testa bit0 de PFDATA - sinal de entrada (bitstream) }
       if eq jump contagem;  {se for 0 pula para contagem se for 1 incrementa acumulador
                                  e depois conta}
soma1:
       ay0 = dm(acumulador);
       ax0 = dm(soma);
       af = ax0 + ay0;
       ar = af + 1;
       dm(acumulador) = ar;
contagem:
       ar = dm(contador);
       ar = ar + 1;
       dm(contador) = ar;
       ay0 = 128;
       ar = ar - ay0;
       if ge jump soma2;
       ar = dm(soma);
       ar = ar + 1;
       dm(soma) = ar;
       rti;

soma2:
       ar = dm(soma);
```

```
       ar = ar - 1;
       dm(soma) = ar;
       ar = dm(contador);
       ay0 = 254;
       ar = ar - ay0;
       if ne rti;
       sr1 = 0;
       sr0 = dm(acumulador);
       sr = ashift sr0 by -6 (lo);   /* divide por 64 >> saida de 8 bits em sr0 */

inic_contador:
       ax0 = 0;
       dm (soma)       = ax0;
       dm (contador)   = ax0;
       dm (acumulador) = ax0;

/*****************************************************************************
*      Polinômio de correção                                                 *
*****************************************************************************/
       ay0 = 128;
       ar = sr0 - ay0;              {torna sinal de entrada diferencial de -128 a 127}
       ax0 = ar;
       i4 = ^coef;
       l4 = %coef;
       i5 = ^y_pad;
       l5 = %y_pad;
       m4 = 1;
       m5 = 1;
       ay0 = dm(i4,m4);             {carrega a1 = Q16.0                              }
       ar  = ax0 + ay0;             {h1= entrada + a1 =>correção de off-set
 =>Q16.0+Q16.0=Q16.0                                  }
       sr = ashift ar by 8 (lo);    {h1 = Q16.0*E0                                   }
       dm(auxiliar)= sr0;           {salva h1    => Q16.0*E0                         }
       ay0 = dm(i5,m5);             {carrega y1  => Q16.0                            }
       ar = ar - ay0;               {(h1-y1)     => Q16.0 - Q16.0 = Q16.0            }
       dm(aux2) = ar;               {salva h1-y1 => Q16.0                            }
       my0 = dm(i4,m4);             {carrega a2   => Q16.0*E1                        }
       mr = ar * my0 (ss);          {a2 * (h1-y1) => Q16.0*E1* Q16.0 = Q32.0*E1x2    }
       se = 7;
       sr = ashift mr1 (hi);
       sr = sr or lshift mr0 (lo);  {resultado em sr1 = Q16.0*E0                     }
       ay0 = dm(auxiliar);          {carrega h1 => Q16.0*E0                          }
       ar = sr1 + ay0;              {h2 = h1 + a2 * (h1-y1)= Q16.0*E0+Q16.0*E0 =
                                     Q16.0*E0                                       }
       dm(auxiliar) = ar;           {salva h2 => Q16.0*E0                            }

       sr = ashift ar by -16 (hi);
       ay0 = dm(i5,m5);             {carrega y2 => Q16.0*E0}
       ax0 = sr0;
       ax1 = sr1;
       ay1=0;
       call dps;                    {(h2-y2) = Q16.0*E0 - Q16.0*E0 = Q32.0*E0}
       mr0 = sr0;
       mr1 = sr1;
       se = -1;
       sr = ashift mr1 (hi);
       sr = sr or lshift mr0 (lo);  { divide por dois e resultado em sr0 = Q16.0*2^7  }
       my0 = dm(aux2);              {carrega h1-y1 = Q16.0                           }
       mr = sr0 * my0 (ss);         {(h2-y2)*(h1-y1) = Q16.0 * Q16.0x2^7 = Q32.0x2^7x2}
       se = -9;
       sr = ashift mr1 (hi);
       sr = sr or lshift mr0 (lo);  {resultado em sr0 = Q16.0x2^-1}
       dm(aux2) = sr0;              {salva (h2-y2)*(h1-y1) em formato Q16.0x2^-1     }
       my0 = dm(i4,m4);             {carrega a3 = Q16.0xE2                           }
       mr = sr0 * my0 (ss);         {a3 * (h2-y2)*(h1-y1) = Q16.0xE2 * Q16.0x2^-1 =
                                     Q32.0xE2 = resultado em mr1 = Q16.0xE0}
       ay0 = dm(auxiliar);          {carrega h2 = Q16.0xE0                           }
       ar = mr1 + ay0;              {h3= h2 + a3 * (h2-y2)*(h1*y1) = Q16.0xE0 +
                                     Q16.0xE0 = Q16.0xE0                          }
       ar = ar + 128;               {arredondamento}
       dm(auxiliar) = ar;           {salva h3 em formato Q16.0xE0 }
       sr = ashift ar by -16 (hi);
       ay0 = dm(i5,m5);             {carrega y3 => Q16.0xE0      }
       ax0 = sr0;
       ax1 = sr1;
       ay1=0;
       call dps;                    {h3-y3 = Q16.0xE0 - Q16.0xE0 = Q32.0xE0}
```

```
        mr0 = sr0;
        mr1 = sr1;
        se = -1;
        sr = ashift mr1 (hi);
        sr = sr or lshift mr0 (lo);   {resultado em sr0 = Q16.0*2^7}
        my0 = dm(aux2);               {carrega (h2-y2)*(h1-y1) => Q16.0x2^-1}
        mr = sr0 * my0 (ss);          {(h3-y3)*(h2-y2)*(h1-y1)= Q16.0x2^7 * Q16.0x2^-1 =
                                       Q32.0x2^7 em mr1 = Q16.0x2^-9}
        my0 = dm(i4,m4);              {carrega a4 =>Q16.0xE3}
        mr = mr1 * my0 (ss);          {a4*(h3-y3)*(h2-y2)*(h1-y1) = Q16.0xE3*Q16.0x2^-x2=
                                       Q32.0xE2 => resultado em mr1 = Q16.0xE0}
        ay0 = dm(auxiliar);           {carrega h3 = Q16.0xE0}
        ar = mr1 + ay0;               {h4= h3+ a4*(h3-y3)*(h2-y2)*(h1*y1) = Q16.0xE0 +
                                       Q16.0xE0 = Q16.0xE0}
        ar = ar + 128;                {arredondamento}
        sr1 = 0;
        sr0 = ar;
        sr = ashift sr0 by -8 (lo);
        ay0 = 128;
        ar = sr0 + ay0;               {soma 128 para tornar numero sem sinal de 8 bits}
        sr0 = ar;

/* Salva dados no buffer circular de tamanho 2000 e coloca ponteiro em index */

        dm(i1,m1) = sr0;
        dm(index) = i1;

/*   colocar aqui rotina de escrita no D/A se necessario*/

        rti;

/*****************************************************************************
%                          DPS
%*****************************************************************************
% Objetivos  : Rotina de subtração de dupla precisão (Double-Precision Subtraction).
%
% Comentários: Implementa Z=X-Y. Esta função foi baixada do site da AD.
%
% Entradas   : AX0 = LSW of X
%              AX1 = MSW of X
%              AY0 = LSW of Y
%              AY1 = MSW of Y
%
% Saidas     : SR0 = LSW of Z
%              SR1 = MSW of Z
%
% Altera     : AR, SR
%
*****************************************************************************/
dps:    AR=AX0-AY0;                          {Subtract LSWs}
        SR0=AR, AR=AX1-AY1+C-1;              {Subtract MSWs}
        SR1=AR;
        RTS;

/*****************************************************************************
*     atende interrupção de retorno ao monitor - IRQE
*****************************************************************************/
retorno:
        ax0 = 1;
        dm(VoltaAoMonitor) = ax0;
        rti;

.endmod;
```

C.4 MÉTODO POLINOMIAL COM CONVERSOR A/D DO KIT DSP

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Elétrica
%
% Modulo     : poly3.dsp
%
% Objetivos  : Implementar o método de linearização Polinomial no kit analog
%              devices (ADSP2181).
%
% Comentários: Baseado no Modulo comp.dsp (Marcelo Negreiros)
%
% Autor      : Osvaldo Betat
%
% Histórico  : 28.06.2001 - Criação do modulo.
%
*****************************************************************************/
.module/RAM/ABS=0       poly3;
.include <const.h>;
/*****************************************************************************
                        DECLARACOES DE VARIAVEIS
*****************************************************************************/
.VAR/DM/RAM             VoltaAoMonitor;
.VAR/DM/RAM/CIRC        samples_in  [7000];
.VAR/DM/RAM/CIRC        samples_out [7000];
.VAR/DM/RAM/ABS=0x1b58  index;
.VAR/DM/RAM             IS_x;
.var/dm/ram/circ        coef[4];
.var/dm/ram/circ        y_pad [3];
.VAR/DM/RAM             auxiliar;
.var/dm/ram             aux2;
/*****************************************************************************
                        INICIALIZACOES
*****************************************************************************/
.INIT                   coef :-18, 0, -17201, 11377; /*a1, a2*2^16, a3*2^24, a4*2^32*/
.INIT                   y_pad:-128, 32256, 10240;
.INIT                   VoltaAoMonitor:  0;
.INIT                   index:0;

/*****************************************************************************
                         VARIVEIS E FUNCOES GLOBAIS
*****************************************************************************/
.GLOBAL                 VoltaAoMonitor;
.ENTRY                  start;

/*****************************************************************************
                         VARIAVEIS E FUNCOES EXTERNAS
*****************************************************************************/
.EXTERNAL               codec_start,irqe,next_cmd;
.EXTERNAL               stat_flag,rx_buf,tx_buf;

/*****************************************************************************
                         TABELA DE VETORES DE INTERRUPCAO
*****************************************************************************/
        jump start;           rti; rti; rti;     /*00: reset       */
        rti;                  rti; rti; rti;     /*04: IRQ2        */
        rti;                  rti; rti; rti;     /*08: IRQL1       */
        rti;                  rti; rti; rti;     /*0c: IRQL0       */

        ar = dm(stat_flag);                     /*10: SPORT0 tx    */
        ar = pass ar;                           /*... SPORT0 tx    */
        if eq rti;                              /*... SPORT0 tx    */
        jump next_cmd;                          /*... SPORT0 tx    */

        jump input_samples;   rti; rti; rti;     /*14: SPORT0 rx   */
        jump irqe;            rti; rti; rti;     /*18: IRQE        */
        rti;                  rti; rti; rti;     /*1c: BDMA        */
        rti;                  rti; rti; rti;     /*20: SPORT1 tx or IRQ1 */
        rti;                  rti; rti; rti;     /*24: SPORT1 rx or IRQ0 */
        rti;                  rti; rti; rti;     /*28: timer       */
        rti;                  rti; rti; rti;     /*2c: power down  */

/*****************************************************************************
%                           START
%*****************************************************************************/
```

```
start:
   dis ints;
   ena ints;
   imask = b#0000000001;
       {     |||||||||+ | timer                    imask    - cada bit do imask habilita
             ||||||||+- | SPORT1 rec or IRQ0                 (1) ou desabilita (0) cada
             |||||||+-- | SPORT1 trx or IRQ1                interrupção individualmente;
             ||||||+--- | BDMA                               Após o reset permanece em
             |||||+---- | IRQE                              estado 0.
             ||||+----- | SPORT0 rec
             |||+------ | SPORT0 trx
             ||+------- | IRQL0
             |+-------- | IRQL1
             +--------- | IRQ2

   icntl = b#00000;
       /*    ||||+- | IRQ0: 0=level, 1=edge
             |||+-- | IRQ1: 0=level, 1=edge
             ||+--- | IRQ2: 0=level, 1=edge
             |+---- 0
             +----- | IRQ nesting: 0=disabled, 1=enabled
       */

   mstat = b#1110000;
       /*    ||||||+- | Data register bank select
             |||||+-- | FFT bit reverse mode (DAG1)
             ||||+--- | ALU overflow latch mode, 1=sticky
             |||+---- | AR saturation mode, 1=saturate, 0=wrap
             ||+----- | MAC result, 0=fractional, 1=integer
             |+------ | timer enable
             +------- | GO MODE
       */

   ax0 = b#0000000000000000;   dm (System_Control_Reg) = ax0;
       /*  +-/!||+-----/+-/- | program memory wait states
           |  !|||           | 0
           |  !|||           |
           |  !||+---------- 0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !|+----------- SPORT1 1=serial, 0=FI, FO, IRQ0, IRQ1, SCLK
           |  !+------------ SPORT1 1=enabled, 0=disabled
           |  +============= SPORT0 1=enabled, 0=disabled
          +---------------- 0
                            0
                            0
       */
  call codec_start;

  i2 = ^samples_in;   /* buffer dos valores de entrada */
  l2 = %samples_in;
  m2 = 1;
  i0 = ^samples_out;  /* buffer dos valores de saída   */
  l0 = %samples_out;
  m0 = 1;
  reset fl1;   /* apaga LED FL1 */

espera:
       idle;

/* Testa se continua processando ou retorna ao Monitor */
testa_retorno:
       ax0 = dm(VoltaAoMonitor);
       ar  = PASS ax0;
       if EQ jump espera;

/* Volta ao monitor ...*/
       dis ints;
       rts;

/*****************************************************************************
%                          INPUT_SAMPLES
%*****************************************************************************/
```

```
% Objetivos  :
%
% Comentários: Serial port data is being accessed using autobuffering. When using
% autobuffering the DSP uses the data address generators to automatically transfer
% serial %port received data to data memory and transfer transmit data from data memory
% to the %serial port. The serial port autobuffering is configured to access received
% data in the %rx_buf buffer and transmit data in the tx_buf buffer.
%
*******************************************************************************/

input_samples:

   ena sec_reg;
   ax0 = dm(IS_x);
   dm (tx_buf + 1) = ax0;            /* send left channel data       */
   ax0 = dm(rx_buf + 1);             /* get left channel data        */

/*******************************************************************************
   Efetua o processamento agora que chegaram as amostras.
*******************************************************************************/

salva_amostras:

   dm(i2,m2) = ax0;                  /*armazena valores de entrada original*/
   dm(index) = i2;

/*******************************************************************************
  Polinômio de correção
*******************************************************************************

        sr1 = 0;
        sr0 = ax0;
        sr = lshift sr0 by -8 (lo);

       ay0 = 128;
       ar = sr0 - ay0;       /*torna sinal de entrada diferencial de -128 a 127*/
       ax0 = ar;

       i4 = ^coef;
       l4 = %coef;
       i5 = ^y_pad;
       l5 = %y_pad;
       m4 = 1;
       m5 = 1;

       ay0 = dm(i4,m4);              /*carrega a1 = Q16.0                        */
       ar  = ax0 + ay0;              /*h1 = valor de entrada + a1 =>correção de off-
      set=> Q16.0+Q16.0=Q16.0*/
       sr = ashift ar by 8 (lo);     /*h1 = Q16.0*E0                             */
       dm(auxiliar)= sr0;            /*salva h1    => Q16.0*E0                   */
       ay0 = dm(i5,m5);              /*carrega y1  => Q16.0                      */
       ar = ar - ay0;                /*(h1-y1)     => Q16.0 - Q16.0 = Q16.0      */
       dm(aux2) = ar;                /*salva h1-y1 => Q16.0                      */
       my0 = dm(i4,m4);              /*carrega a2   => Q16.0*E1*/
       mr = ar * my0 (ss);           /*a2 * (h1-y1) => Q16.0*E1* Q16.0 = Q32.0*E1x2*/
       se = 7;
       sr = ashift mr1 (hi);
       sr = sr or lshift mr0 (lo);   /*resultado em sr1 = Q16.0*E0*/
       ay0 = dm(auxiliar);           /*carrega h1 => Q16.0*E0*/
       ar = sr1 + ay0;               /*h2 = h1 + a2 * (h1-y1) = Q16.0*E0 + Q16.0*E0 =
                                           Q16.0*E0*/
       dm(auxiliar) = ar;            /*salva h2 => Q16.0*E0*/
       sr = ashift ar by -16 (hi);
       ay0 = dm(i5,m5);              /*carrega y2 => Q16.0*E0*/
       ax0 = sr0;
       ax1 = sr1;
       ay1=0;

       call dps;                     /*(h2-y2) = Q16.0*E0 - Q16.0*E0 = Q32.0*E0*/
       mr0 = sr0;
       mr1 = sr1;
       se = -1;
       sr = ashift mr1 (hi);
       sr = sr or lshift mr0 (lo);   /* divide por dois e resultado em sr0 = Q16.0*2^7*/

       my0 = dm(aux2);               /*carrega h1-y1 = Q16.0                     */
       mr = sr0 * my0 (ss);          /*(h2-y2)*(h1-y1) = Q16.0 * Q16.0x2^7 =
```

```
                                        Q32.0x2^7x2*/

        se = -9;
        sr = ashift mr1 (hi);
        sr = sr or lshift mr0 (lo);   /*resultado em sr0 = Q16.0x2^-1*/

        dm(aux2) = sr0;               /*salva (h2-y2)*(h1-y1) em formato Q16.0x2^-1*/
        my0 = dm(i4,m4);              /*carrega a3 = Q16.0xE2*/
        mr = sr0 * my0 (ss);          /*a3 * (h2-y2)*(h1-y1) = Q16.0xE2 * Q16.0x2^-1 =
                                        Q32.0xE2 = resultado em mr1 = Q16.0xE0*/
        ay0 = dm(auxiliar);           /*carrega h2 = Q16.0xE0*/
        ar = mr1 + ay0;               /*h3= h2 + a3 * (h2-y2)*(h1*y1) = Q16.0xE0 +
                                        Q16.0xE0 = Q16.0xE0*/
        dm(auxiliar) = ar;            /*salva h3 em formato Q16.0xE0*/
        sr = ashift ar by -16 (hi);
        ay0 = dm(i5,m5);              /*carrega y3 => Q16.0xE0*/
        ax0 = sr0;
        ax1 = sr1;
        ay1=0;
        call dps;                     /*h3-y3 = Q16.0xE0 - Q16.0xE0 = Q32.0xE0*/
        mr0 = sr0;
        mr1 = sr1;
        se = -1;
        sr = ashift mr1 (hi);
        sr = sr or lshift mr0 (lo);   /*resultado em sr0 = Q16.0*2^7*/
        my0 = dm(aux2);               /*carrega (h2-y2)*(h1-y1) => Q16.0x2^-1*/
        mr = sr0 * my0 (ss);          /*(h3-y3)*(h2-y2)*(h1-y1)= Q16.0x2^7 * Q16.0x2^-1 =
                                        Q32.0x2^7 em mr1 = Q16.0x2^-9*/
        my0 = dm(i4,m4);              /*carrega a4 =>Q16.0xE3*/

        mr = mr1 * my0 (ss);          /*a4*(h3-y3)*(h2-y2)*(h1-y1) = Q16.0xE3*Q16.0x2^-
                                        9x2=Q32.0xE2 => resultado em mr1 = Q16.0xE0*/
        ay0 = dm(auxiliar);           /*carrega h3 = Q16.0xE0*/
        ar = mr1 + ay0;               /*h4= h3+ a4*(h3-y3)*(h2-y2)*(h1*y1) = Q16.0xE0 +
                                       Q16.0xE0 = Q16.0xE0*/
        ar = ar + 128;                /*arredondamento*/
        ay0 = 0xff00;
        ar  = ar and ay0;             /*zera lsw*/
        ay0 = 0x8000;
        ar = ar + ay0;                /*soma 128 para tornar numero sem sinal de 8 bits*/

        dm(IS_x)  = ar;               /*entrada de valores corrigidos para saida x*/

/* Salva dados no buffer circular de tamanho 7000 */
        dm(i0,m0) = ar;
        rti;
                /* Return from interrupt, will pop the status register which consists
                 * MSTAT, ASTAT and IMASK ( returns to primary registers because of
                 * pop of MSTAT                                                      */

/*****************************************************************************
%                           DPS
%*****************************************************************************
% Objetivos  : Rotina de subtração de dupla precisão (Double-Precision Subtraction).
%
% Comentários: Implementa Z=X-Y. Esta função foi baixada do site da AD.
%
% Entradas   : AX0 = LSW of X
%              AX1 = MSW of X
%              AY0 = LSW of Y
%              AY1 = MSW of Y
%
% Saidas     : SR0 = LSW of Z
%              SR1 = MSW of Z
%
% Altera     : AR, SR
%
*****************************************************************************/
dps:    AR=AX0-AY0;                         /*Subtract LSWs*/
        SR0=AR, AR=AX1-AY1+C-1;             /*Subtract MSWs*/
        SR1=AR;
        RTS;


.endmod;
```

CODEC2.DSP

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Elétrica
%
% Modulo     : CODEC2.DSP
%
% Objetivos  : Inicializar o codec do kit analog devices ADSP2181.
%
% Comentários: Baseado no exemplo do Marcelo Negreiros codec.dsp.
%
% Autor      : Osvaldo Betat
%
% Histórico  : 28.06.2001 - Criação do modulo.
%
*****************************************************************************/
.module/RAM  codec2;
.include <const.h>;

/*****************************************************************************
                        DECLARACOES DE VARIVEIS
*****************************************************************************/
.var/dm/ram/circ    rx_buf[3];   /* declare receive buffer, Status + L data + R data  */
.var/dm/ram/circ    tx_buf[3];   /* declare transmit buffer, Cmd + L data + R data    */
.var/dm/ram/circ    init_cmds[13]; /* declare buffer with AD1847 initialization codes */
.var/dm             stat_flag; /* declare a variable to be used as a status flag      */
.var/dm/ram         irqe_var;  /* variavel usada para salvar contexto em irqe         */

/*****************************************************************************
                        VARIVEIS E FUNCOES GLOBAIS
*****************************************************************************/
.GLOBAL                 rx_buf,tx_buf,stat_flag;
.GLOBAL                 codec_start,irqe,next_cmd;

/*****************************************************************************
                        VARIAVEIS E FUNCOES EXTERNAS
*****************************************************************************/
.EXTERNAL               VoltaAoMonitor;

/*****************************************************************************
                        INICIALIZACOES
*****************************************************************************/
/* The AD1847 CODEC communicates with the DSP serially. The AD1847 transmits and
* receives three 16 bit words per sample. In this program we declare two, 3 location
* circular buffers, tx_buf for the transmit data buffer and rx_buf for the receive data
* buffer. The AD1847 receives from the DSP's tx_buf, a command word, left channel data,
* and right channel data each sample. The AD1847 transmits to the DSP rx_buf a status
* word, left channel data and right channel data each sample. The DSP's serial port
* communicating with the AD1847 is configured to use autobuffering and multi-channel
* mode. This allows the DSP's serial port to automatically access memory and
* automatically transmit/receive three words at a time. Once the DSP's serial port is
* configured and running, it will run forever interrupting the DSP when new data has
* been received. This means the DSP reads the AD1847's data by reading the rx_buf buffer
* and writes to the AD1847 by writing to the tx_buf buffer. The DSP knows when it can
* read/write the AD1847 via the serial port interrupts.                */
*****************************************************************************/

.init tx_buf:   0xc000, 0x0000, 0x0000;     /* Initialize the tx_buff with these values.
                                             * Command word = 0xc0000
                                             * Left channel data = 0x0000
                                             * Right channel data = 0x0000.
                                             * This command word will place AD1847 in
 * command mode by setting the MCE bit                                    */

/* Initialize the init_cmds buffer with the following values. These are the initial
*  configuration values for the AD1847 CODEC. Bits 0-7 are the 8 bits of data for  the
*  index registers ( data0-7 ). Bits 8-11 are the address for the index registers
*  ( IA0-3 ). Bit 13 sets the part into readback  mode ( RREQ ). Bit 14 is the
*  Mode Change Enable bit ( MCE ). This must be set when changing the index registers.
*  Bit 15 is the Clear Overrange bit ( CLOR ). This bit determines the overrange status
*  mode. The program is set up to automatically load this buffer into the AD1847.*/

.init init_cmds:
       0xc000,     /*  CLOR set, MCE set, Index reg address=0, data=0x2
                    *  Left input control reg
```

```
            *  b7-6: 0=left line 1
            *        1=left aux 1
            *        2=left line 2
            *        3=left line 1 post-mixed loopback
            *  b5-4: res
            *  b3-0: left input gain x 1.5 dB               */

0xc100,     /*  CLOR set, MCE set, Index reg address=1, data=0x2
            *  Right input control reg
            *  b7-6:  0=right line 1
            *         1=right aux 1
            *         2=right line 2
            *         3=right line 1 post-mixed loopback
            *  b5-4:  res
            *  b3-0:  right input gain x 1.5 db              */

0xc288,     /*  CLOR set, MCE set, Index reg address=2, data=0x88
            *   Left aux 1 control reg
            *   b7  :    1=left aux 1 mute
            *   b6-5:    res
            *   b4-0:    gain/atten x 1.5, 08= 0dB, 00= 12dB    */

0xc388,     /* CLOR set, MCE set, Index reg address=3, data=0x88
            *   Right aux 1 control reg
            *   b7  :   1=right aux 1 mute
            *   b6-5:   res
            *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

0xc488,     /* CLOR set, MCE set, Index reg address=4, data=0x88
            *   left aux 2 control reg
            *   b7  :   1=left aux 2 mute
            *   b6-5:   res
            *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

0xc588,     /* CLOR set, MCE set, Index reg address=5, data=0x88
            *   right aux 2 control reg
            *   b7  :   1=right aux 2 mute
            *   b6-5:   res
            *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

0xc680,     /* CLOR set, MCE set, Index reg address=6, data=0x80
            *   left DAC control reg
            *   b7  :   1=left DAC mute
            *   b6  :   res
            *   b5-0:   attenuation x 1.5 dB    */

0xc780,     /* CLOR set, MCE set, Index reg address=7, data=0x80
            *   right DAC control reg
            *   b7  :   1 = right DAC mute
            *   b6  :   res
            *   b5-0:   attenuation x 1.5 dB    */

0xc810,     /* CLOR set, MCE set, Index reg address=8, data=0x5c
            *   data format register
            *   b7  :   res
            *   b5-6:   0 = 8-bit unsigned linear PCM
            *           1 = 8-bit u-law companded
            *           2 = 16-bit signed linear PCM
            *           3 = 8-bit A-law companded
            *   b4  :   0 = mono, 1 =  stereo
            *   b0-3:   0 =   8.00000 Khz
            *           1 =   5.51250 Khz
            *           2 = 16.00000 Khz
            *           3 = 11.02500 Khz
            *           4 = 27.42857 Khz
            *           5 = 18.90000 Khz
            *           6 = 32.00000 Khz
            *           7 = 22.05000 Khz
            *           8 =   .
            *           9 = 37.80000 Khz
            *           a =   .
            *           b = 44.10000 Khz
            *           c = 48.00000 Khz
            *           d = 33.07500 Khz
            *           e =   9.60000 Khz
            *           f =   6.61500 Khz
            *  (b0) :   0 = XTAL1 24.576 Mhz; 1 = XTAL2 16.9344 Mhz     */
```

```
       0xc909,     /* CLOR set, MCE set, Index reg address=9, data=0x09
                   *   interface configuration reg
                   *   b7-4:   res
                   *   b3  :   1 = autocalibrate
                   *   b2-1:   res
                   *   b0  :   1 = playback enabled    */

       0xca00,     /* CLOR set, MCE set, Index reg address=0xa, data=0
                   *   pin control reg
                   *   b7  :   logic state of pin XCTL1
                   *   b6  :   logic state of pin XCTL0
                   *   b5  :   master - 1 = tri-state CLKOUT
                   *           slave  - x = tri-state CLKOUT
                   *   b4-0:   res                          */


       0xcc40,     /* CLOR set, MCE set, Index reg address=0xc, data=0x40
                   *   miscellaneous information reg
                   *   b7  :   1 = 16 slots per frame
                   *           0 = 32 slots per frame
                   *   b6  :   1 = 2-wire system
                   *           0 = 1-wire system
                   *   b5-0:   res                  */


       0xcd00;     /* CLOR set, MCE set, Index reg address=0xd, data=0
                   *   digital mix control reg
                   *   b7-2:   attenuation x 1.5 dB
                   *   b1  :   res
                   *   b0  :   1 = digital mix enabled */



/*====================  ADSP 2181 INITIALIZATION ========================*/

/*---------------- Data Address Generator Initialization  ----------------*/

/*****************************************************************************
%                           CODEC_START
%*****************************************************************************
% Objetivos  : Inicializar o codec do kit analog devices ADSP2181.
%
% Comentarios: Baseado no exemplo do kit (EZ-KIT LITE TALKTHRU DEMO).
%              ATENCAO: i0,i1 e m1 sao usados permanentemente pela serial.
%
% Entradas   :  rx_buf, tx_buf
%
% Saidas     :  rx_buf, tx_buf
%
% Altera     :  i6,l6,i7,l7,i3,l3,m7,ax0,ay0,ar
%
*****************************************************************************/
codec_start:
       i6 = ^rx_buf;           /* set i0 = the starting address of  rx_buf     */
       l6 = %rx_buf;           /* set l0 = the length of rx_buf                */
       i7 = ^tx_buf;           /* set i1 = the starting address of  tx_buf     */
       l7 = %tx_buf;           /* set l1 = the length of tx_buf                */
       i3 = ^init_cmds;        /* set i3 = the starting address of  init_cmds  */
       l3 = %init_cmds;        /* set l3 = the length of  init_cmds            */
       m7 = 1;
        m1 = 1;

/*------------------------- S E R I A L   P O R T   #0   S T U F F -----------------*/

/* This code configures the  serial ports on the DSP. The serial ports are configured by
 * writing to memory mapped control registers. SPORT0 is configured to use autobuffering
 * and  multi-channe  mode  with  an  external  serial clock  and external frame sync.
 * Autobuffering  can be thought of as serial port DMA. The serial port uses the address
 * generators to access data  memory automatically.     In this program autobuffering is
 * configured to use i0, i1, and m1.      These are data address generator registers. i0
 * points to the rx_buf buffer, i1 points to the tx_buf buffer and m1 = 1.
 *      Multi-channel mode is enabled and configured for 32 channels. The AD1847 is also
 * configured  for 32 channels.      In Multi-channel mode  each channel  is a  word and
 * associated with a time slot. In each of the 32 time slot the serial port can transmit
 * and/or receive data or do nothing, tristating the data for that time slot.  One frame
```

```
 * consists of all 32 channels.    One frame sync  will start a frame transfer  ( all 32
 * words ).   This program configures the serial port 0 to transmit on channels 0, 1, 2,
 * 16, 17 and 18 for transmit and channels 0, 1, 2, 16, 17, and 18 for receive. It
 * will ignore all other time slots.
 *  External serial clock and external frame-syncs mean they are inputs to the DSP.   */

       ax0 =0xfef;
       dm (SPORT0_Autobuf) = ax0;    /*  enable receive and transmit autobuffering,
                                      *  RMREG=m7, RIREG=i6, TMREG=m7, TIREG=i7,
                                      *  CLKOUT enabled, BIASRND disabled           */
       ax0 = 0;
       dm (SPORT0_RFSDIV) = ax0;     /*  RFSDIV = SCLK Hz/RFS Hz - 1
                                      *  using external      receive frame sync
                                      * ( RFS=input )        */
       ax0 = 0;
       dm (SPORT0_SCLKDIV) = ax0;    /*  SCLK = CLKOUT / (2  (SCLKDIV + 1)
                                      *  using external serial clock ( SCLK=input )  */
       ax0 =0x860f;
       dm (SPORT0_Control_Reg) = ax0;      /*  SLEN= 16 bits, right justify, zero-fill
                                           *  unused MSBs,
                                           *  INVRFS=0, INVTFS=0, IRFS=0, ITFS=1,
                                           *  MFD=1, ISCLK=0, MCE=1               */
       ax0 =0x7;
       dm (SPORT0_TX_Channels0) = ax0;  /* enable channels 0, 1 and 2 for transmit  */
       ax0 =0x7;
       dm (SPORT0_TX_Channels1) = ax0; /* enable channels 16, 17 and 18 for transmit */
       ax0 =0x7;
       dm (SPORT0_RX_Channels0) = ax0; /* enable channels 0, 1 and 2 for receive     */
       ax0 =0x7;
       dm (SPORT0_RX_Channels1) = ax0; /* enable channels 16, 17 and 18 for receive  */


/*------------------ S Y S T E M   A N D   M E M O R Y   S T U F F ------------------*/
       ax0 = 0xfff;
       dm (DM_Wait_Reg) = ax0;         /*  Set all IOWAIT ranges to 7 wait states,
                                        *  set DWAIT to 0 wait states               */

       ax0 = 0x1000;                   /* PWAIT = 0, enable SPORT0                   */
       dm (System_Control_Reg) = ax0;

       ifc = 0xff;                     /* clear pending interrupt                    */
       nop;
       icntl = 0;                      /* external interrupts set to level sensitivity,
                                        * disable nested interrupts                  */
       mstat = 0x40;                   /* enable go mode                             */


/*==========================  AD1847 CODEC INITIALIZATION ===========================*/
       ax0 = 1;
       dm(stat_flag) = ax0;            /* initialize stat_flag to 1                 */
       imask = b#0001000000;           /* unmask SPORT0s transmit interrupt         */
       ax0 = dm (i7, m7);              /* set ax0 = the first value of the tx_buf   */
       tx0 = ax0;                      /* begin autobuffer transmit by writing first
                                        * value to tx0 reg(when autobuffer transmit
                                        * completes a transmit interrupt
                                        * will be generated)                        */

/*  Test for entire init_cmds  buffer to be sent to the codec   */
check_init:
       ax0 = dm (stat_flag); /* set ax0 = the value at stat_flag. Flag set in SPORT0
                              * tx interrupt routine                               */
       af = pass ax0;        /* pass the value of ax0 thru the ALU setting
                              * status flags                                       */
       if ne jump check_init;  /* if result of pass not equal to 0 jump check_init   */

/*   Once initialized, wait for codec to come out of autocalibration by testing the ACI
 *   bit of the AD1847s status reg. This is done twice, first to ensure the codec is in
 *   autocalibration mode, then to determine when autocalibration is complete.       */

       ay0 = 2;
check_aci1:                     /* loop to test ACI bit = 1 ( in autocalibration )   */
       ax0 = dm (rx_buf);      /* read status word of AD1847                        */
       ar = ax0 and ay0;       /* AND 2 with status word                            */
       if eq jump check_aci1;  /* if result of AND is not = 0 leave loop            */

check_aci2:                     /* loop to test ACI bit =0 (autocalibration complete) */
       ax0 = dm (rx_buf);      /* wait for bit clear                                */
```

```
       ar = ax0 and ay0;
       if ne jump check_aci2;
       idle;

/* Once autocalibration is complete unmute left and right DAC channels by writing the
 * appropriate index register                                                        */

       ay0 = 0xbf3f;                  /* unmute left DAC                      */
       ax0 = dm (init_cmds + 6);
       ar = ax0 AND ay0;
       dm (tx_buf) = ar;
       idle;                          /* wait for transmit to complete        */
       ax0 = dm (init_cmds + 7);      /* unmute right DAC                     */
       ar = ax0 AND ay0;
       dm (tx_buf) = ar;
       idle;                          /* wait for transmit to complete        */

       ifc = 0xff;                    /* clear any pending interrupt          */
       nop;
       imask = 0x30;                  /* unmask rx0 and  IRQE interrupts      */

   rts;                               /* retorna                              */

/*-------------------- Transmit Interrupt Service Routine--------------------------*/

/*     The transmit interrupt service routine is used to initializes the CODEC (AD1847).
 * I3 points to the init_cmds buffer. The AD1847 is expecting a control word and two
 * data words from the DSP. Here we load the next command word into the first location
 * of the tx_buf buffer, the two data words have been initialized to zero in the
 * beginning of the program. This routine also checks to see if the last data
 * word has been transmitted. If the last data word has been transmitted take the AD1847
 * out of command
 * mode and return from interrupt.                                                   */

next_cmd:
       ena sec_reg;
       ax0 = dm (i3, m1);       /* get command word from init_cmds buffer        */
       dm (tx_buf) = ax0;       /* place command word in first location of tx_buf
                                 * ( transmit slot 0)    */
       ax0 = i3;
       ay0 = ^init_cmds;
       ar = ax0 - ay0;          /* test for additional command words             */
       if gt rti;               /* rti if more control words still waiting       */
       ax0 = 0xaf00;            /* else set done flag and remove MCE if done
                                  * initialization                               */
       dm (tx_buf) = ax0;
       ax0 = 0;
       dm (stat_flag) = ax0;    /* reset status flag */
       rti;

/*--------------------- IRQE Interrupt Service Routine ----------------------------*/

/* The IRQE interrupt service routine toggles flag-out which is connected to
 * the FL1 LED on EZ-KIT LITE.                                    */

irqe:   toggle fl1;             /* Toggle flag-out 1            */
        dm(irqe_var) = ar;      /* salva contexto */
        ar = PASS 1;
        dm(VoltaAoMonitor)=ar;  /* Ativa flag de retorno ao monitor do KIT */
        ar = dm(irqe_var);      /* restaura contexto */
        rti;                    /* Return from interrupt, will pop the status register
                                 * which consists MSTAT, ASTAT and IMASK ( returns to
                                 * primary registers because of pop of MSTAT        */
.endmod;
```

CONST.H

```
.const  IDMA=                   0x3fe0;
.const  BDMA_BIAD=              0x3fe1;
.const  BDMA_BEAD=              0x3fe2;
.const  BDMA_BDMA_Ctrl=         0x3fe3;
.const  BDMA_BWCOUNT=           0x3fe4;
.const  PFDATA=                 0x3fe5;
.const  PFTYPE=                 0x3fe6;
```

```
.const  SPORT1_Autobuf=         0x3fef;
.const  SPORT1_RFSDIV=          0x3ff0;
.const  SPORT1_SCLKDIV=         0x3ff1;
.const  SPORT1_Control_Reg=     0x3ff2;
.const  SPORT0_Autobuf=         0x3ff3;
.const  SPORT0_RFSDIV=          0x3ff4;
.const  SPORT0_SCLKDIV=         0x3ff5;
.const  SPORT0_Control_Reg=     0x3ff6;
.const  SPORT0_TX_Channels0=    0x3ff7;
.const  SPORT0_TX_Channels1=    0x3ff8;
.const  SPORT0_RX_Channels0=    0x3ff9;
.const  SPORT0_RX_Channels1=    0x3ffa;
.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  DM_Wait_Reg=            0x3ffe;
.const  System_Control_Reg=     0x3fff;
```

C.5 MÉTODO ADAPTATIVO COM CONVERSOR A/D DO KIT DSP

```
/*****************************************************************************
% Universidade Federal do Rio Grande do Sul
% Escola de Engenharia - Departamento de Engenharia Eletrica
%
% Modulo     :  Adapt6.dsp
%
% Objetivos  :  Implementar o método de linearização com filtro Adaptativo não linear
%               no kit analog devices (ADSP2181).
%
% Comentarios:  Baseado no Modulo comp.dsp (Marcelo Negreiros)
%
% Autor      :  Osvaldo Betat
%
% Historico  : 09.11.2001 - Criacao do modulo.
%
*****************************************************************************/
.module/RAM/ABS=0       Adapt6;

.include <constant.h>;        /* constantes dos filtros   */
.include <const.h>;           /* constantes do ad         */
.include <coef.h>;            /* coeficientes dos filtros */

/*****************************************************************************
                        DECLARACOES DE VARIAVEIS
*****************************************************************************/
.VAR/DM/RAM             VoltaAoMonitor;
.VAR/DM/RAM/CIRC        samples_in  [7000];
.VAR/DM/RAM/CIRC        samples_out [7000];
.VAR/DM/RAM/ABS=0x1b58  index;
.VAR/DM/RAM             IS_x;
.VAR/DM/RAM             IS_f_x1;
.VAR/DM/RAM             IS_f_x2;
.VAR/DM/RAM             IS_f_x3;

.VAR/PM/RAM/CIRC        coefs[N];
.VAR/PM/RAM/CIRC        coefs_x2[N_x2];
.VAR/PM/RAM/CIRC        coefs_x3[N_x3];
.VAR/DM/RAM/CIRC        delay_line[N];
.VAR/DM/RAM/CIRC        delay_line_x2[N_x2];
.VAR/DM/RAM/CIRC        delay_line_x3[N_x3];

.VAR/DM/RAM             delay_line_index;
.VAR/DM/RAM             delay_line_index_x2;
.VAR/DM/RAM             delay_line_index_x3;
/*****************************************************************************
                        INICIALIZACOES
*****************************************************************************/
.INIT                   VoltaAoMonitor:    0;
.INIT                   index:             0;

/*****************************************************************************
                         VARIVEIS E FUNCOES GLOBAIS
*****************************************************************************/
```

```
.GLOBAL                 VoltaAoMonitor;
.ENTRY                  start;


/*****************************************************************************
                         VARIAVEIS E FUNCOES EXTERNAS
*****************************************************************************/
.EXTERNAL               codec_start,irqe,next_cmd;
.EXTERNAL               stat_flag,rx_buf,tx_buf;

/*****************************************************************************
                         TABELA DE VETORES DE INTERRUPCAO
*****************************************************************************/
       jump start;           rti; rti; rti;     /*00: reset */
       rti;                  rti; rti; rti;     /*04: IRQ2  */
       rti;                  rti; rti; rti;     /*08: IRQL1 */
       rti;                  rti; rti; rti;     /*0c: IRQL0 */
       ar = dm(stat_flag);                      /*10: SPORT0 tx */
       ar = pass ar;                            /*... SPORT0 tx */
       if eq rti;                               /*... SPORT0 tx */
       jump next_cmd;                           /*... SPORT0 tx */
       jump input_samples;   rti; rti; rti;     /*14: SPORT0 rx */
       jump irqe;            rti; rti; rti;     /*18: IRQE  */
       rti;                  rti; rti; rti;     /*1c: BDMA */
       rti;                  rti; rti; rti;     /*20: SPORT1 tx or IRQ1 */
       rti;                  rti; rti; rti;     /*24: SPORT1 rx or IRQ0 */
       rti;                  rti; rti; rti;     /*28: timer */
       rti;                  rti; rti; rti;     /*2c: power down */

/*****************************************************************************
%                           START
%*****************************************************************************/
%
% Altera     :  i7,l7,m7,i6,l6,
%
*****************************************************************************/
start:
   dis ints;
   ena ints;

   /*inicializa linha de atraso do filtro linear*/
   ar = 0;
   i0 = ^delay_line;
   m0 = 1;
   l0   = %delay_line;
   CNTR = %delay_line;
   do init_delay until ce;
init_delay:
   dm(i0,m0)=ar;
   dm(delay_line_index)=i0;

   /*inicializa linha de atraso x2*/
   ar = 0;
   i0 = ^delay_line_x2;
   m0 = 1;
   l0   = %delay_line_x2;
   CNTR = %delay_line_x2;
   do init_delay_x2 until ce;
init_delay_x2:
   dm(i0,m0)=ar;
   dm(delay_line_index_x2)=i0;

   /*inicializa linha de atraso x3*/
   ar = 0;
   i0 = ^delay_line_x3;
   m0 = 1;
   l0   = %delay_line_x3;
   CNTR = %delay_line_x3;
   do init_delay_x3 until ce;
init_delay_x3:
   dm(i0,m0)=ar;
   dm(delay_line_index_x3)=i0;

   imask = b#0000000000;
       {     |||||||||+ | timer                    imask    - cada bit do imask habilita
             ||||||||+- | SPORT1 rec or IRQ0                 (1) ou desabilita (0) cada
             |||||||+-- | SPORT1 trx or IRQ1                interrupção individualmente;
```

```
            ||||||+--- | BDMA                          Após o reset permanece em
            |||||+---- | IRQE                          estado 0.
            ||||+----- | SPORT0 rec
            |||+------ | SPORT0 trx
            ||+------- | IRQL0
            |+-------- | IRQL1
            +--------- | IRQ2

   icntl = b#00000;
       /*    ||||+- | IRQ0: 0=level, 1=edge
             |||+-- | IRQ1: 0=level, 1=edge
             ||+--- | IRQ2: 0=level, 1=edge
             |+---- | 0
             +----- | IRQ nesting: 0=disabled, 1=enabled
       */

   mstat = b#1000000;
       /*    ||||||+- | Data register bank select
             |||||+-- | FFT bit reverse mode (DAG1)
             ||||+--- | ALU overflow latch mode, 1=sticky
             |||+---- | AR saturation mode, 1=saturate, 0=wrap
             ||+----- | MAC result, 0=fractional, 1=integer
             |+------ | timer enable
             +------- | GO MODE
       */

   ax0 = b#0000000000000000;   dm (System_Control_Reg) = ax0;
       /*  +-/!||+-----/+-/- | program memory wait states
           |  !|||           | 0
           |  !|||           |
           |  !||+---------- 0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !||            0
           |  !|+----------- SPORT1 1=serial, 0=FI, FO, IRQ0, IRQ1, SCLK
           |  !+------------ SPORT1 1=enabled, 0=disabled
           |  +============= SPORT0 1=enabled, 0=disabled
           +---------------- 0
                             0
                             0
       */

  call codec_start;

  i2 = ^samples_in;   /* buffer dos valores de entrada */
  l2 = %samples_in;
  m2 = 1;

  i5 = ^samples_out;  /* buffer dos valores de saída   */
  l5 = %samples_out;
  m5 = 1;

  reset fl1;          /*Apaga LED FL1 */

espera:
       idle;

/* Testa se continua processando ou retorna ao Monitor */

testa_retorno:
       ax0 = dm(VoltaAoMonitor);
       ar  = PASS ax0;
       if EQ jump espera;

/* Volta ao monitor ...*/
       dis ints;
       rts;


/*****************************************************************************
%                          INPUT_SAMPLES
%*****************************************************************************/
input_samples:
   ena sec_reg;
```

```
   ax0 = dm(IS_x);
   dm(tx_buf + 1) = ax0;        /* send left channel data       */
   ax0 = dm(rx_buf + 1);        /* get left channel data        */

/*******************************************************************************
   Efetua o processamento agora que chegaram as amostras.
   As novas amostras de entrada estao em ax0(left).
   Na proxima interrupcao serao enviados IS_x(left).
*******************************************************************************/
   dm(i2,m2) = ax0;          /*salva dado de entrada em buffer de tamanho 7000*/
   dm(index) = i2;
   ay0 = 0x80FF;
   ar  = ax0 - ay0;          /*torna sinal de entrada diferencial de -128 a 127*/
   dm(IS_x)  = ar;           /*entrada de valores para saida x (left)- formato Q1.15 */

/*******************************************************************************
   Algoritmo de Linerização - Filtro não linear (x -> x^2 -> x^3)
*******************************************************************************/
/*******************************************************************************
   Filtro linear
*******************************************************************************
% filtro_fir
% Entradas   :      I0 -> Oldest input data value in delay line
%                   L0 -> Filter length (N)
%                   I4 -> Begginin of filter coefficient table
%                   L4 -> Filter length (N)
%                   M0,M4 = 1;
%                   CNTR -> Filter length - 1 (N-1)
%
% Saidas     :      MR1 -> Sum of products (rounded and saturated)
%                   I0  -> Oldest input data value in delay line
%                   I4  -> Beggining of filter coefficient table
*******************************************************************************/
   i0 = dm(delay_line_index);
   l0 = %delay_line;
   m0 = 1;
   ax0 = dm(IS_x);

   dm(i0,m0) = ax0;          /*sinal de entrada canal esquerdo - formato Q1.15*/
   dm(delay_line_index) = i0;
   i4 = ^coefs;
   l4 = %coefs;
   m4 = 1;
   CNTR = Nmenos1;
   /* chama filtro ->>  sinal em formato Q1.15 e coeficientes em formato Q2.14 */
   call filtro_fir;
   /* saída em formato Q3.29 -> parametro ponto flutuante ativado -> Q2.30 */
   /* saida em mr1 com formato Q2.14 */
   dm(IS_f_x1) = mr1;

/*******************************************************************************
   Filtro x2
*******************************************************************************
/*
% filtro_fir
% Entradas   :      I0 -> Oldest input data value in delay line
%                   L0 -> Filter length (N)
%                   I4 -> Begginin of filter coefficient table
%                   L4 -> Filter length (N)
%                   M0,M4 = 1;
%                   CNTR -> Filter length - 1 (N-1)
%
% Saidas     :      MR1 -> Sum of products (rounded and saturated)
%                   I0  -> Oldest input data value in delay line
%                   I4  -> Beggining of filter coefficient table
*******************************************************************************/
   i0 = dm(delay_line_index_x2);
   l0 = %delay_line_x2;
   m0 = 1;
   ar = dm(IS_x);
   mr = ar * ar (ss);             /* Q1.15 * Q1.15 = Q2.31 -> Q1.31 */
   if mv sat mr;
   dm(i0,m0)= mr1;                /*sinal de entrada ao quadrado Q1.15 */
   dm(delay_line_index_x2) = i0;
   i4 = ^coefs_x2;
   l4 = %coefs_x2;
```

```
   m4 = 1;
   CNTR = Nmenos1_x2;

   /* chama filtro ->>  sinal em formato Q1.15 e coeficientes em formato Q1.15 */
   call filtro_fir;
   /* saida em mr1 com formato Q1.15 * Q1.15 = Q2.30 -> Q1.15 */

   sr = ashift mr1 by -1 (hi);
   dm(IS_f_x2) = sr1; /* sinal em formato Q2.14 */


/*****************************************************************************
   Filtro x3
*****************************************************************************
%filtro_fir
% Entradas   :      I0 -> Oldest input data value in delay line
%                   L0 -> Filter length (N)
%                   I4 -> Begginin of filter coefficient table
%                   L4 -> Filter length (N)
%                   M0,M4 = 1;
%                   CNTR -> Filter length - 1 (N-1)
%
% Saidas     :      MR1 -> Sum of products (rounded and saturated)
%                   I0  -> Oldest input data value in delay line
%                   I4  -> Beggining of filter coefficient table
*****************************************************************************/
   i0 = dm(delay_line_index_x3);
   l0 = %delay_line_x3;
   m0 = 1;

   ar  = dm(IS_x);
   my0 = dm(IS_x);
   mr = ar * ar (ss); /* Q1.15 * Q1.15 = Q1.15 */
   if mv sat mr;

   /*quadrado em sr0 formato Q16.0 , agora faz ao cubo:*/
   ar = mr1;
   mr = ar * my0 (ss); /* Q1.15 * Q1.15 = Q1.15*/
   if mv sat mr;

   /*cubo em mr1, final Q1.15*/

   dm(i0,m0)=mr1;            /*sinal de entrada ao cubo Q1.15*/

   dm(delay_line_index_x3) = i0;
   i4 = ^coefs_x3;
   l4 = %coefs_x3;
   m4 = 1;
   CNTR = Nmenos1_x3;

   /* chama filtro ->>  sinal em formato Q1.15 e coeficientes em formato Q1.15 */
   call filtro_fir;
   /* saida em mr1 com formato Q1.15 * Q1.15 = Q1.15 */

   sr = ashift mr1 by -1 (hi);
   dm(IS_f_x3) = sr1;        /*sinal F ao cubo em formato  Q2.14*/

/*****************************************************************************
   Calcula a saida total dos filtros
*****************************************************************************/
   ar  = dm(IS_f_x1);
   ay0 = dm(IS_f_x2);
   ar  = ar + ay0;    /* Q2.14 + Q2.14 */
   ay0 = dm(IS_f_x3);
   ar  = ar + ay0;    /* Q2.14 + Q2.14 */

   /* sinal de saida total em ar formato Q2.14 */
   sr0 = 0;
   sr = ashift ar by 1 (hi);
   ar = sr1;
  /* sinal em formato Q1.15 */

   ay0 = 0x8000;
   ar  = ar + ay0;           /* passa sinal em numero sem sinal  */
   dm(IS_x) = ar;            /* Salva dado para ser enviado para o D/A */

/* Salva dados no buffer circular de tamanho 7000 */
```

```
   dm(i5,m5) = ar;
   rti;

/*****************************************************************************
%                           FILTRO_FIR
%*****************************************************************************
% Objetivos  :
%
% Comentarios:      FIR Transversal Filter Subroutine.
%
% Entradas   :      I0 -> Oldest input data value in delay line
%                   L0 -> Filter length (N)
%                   I4 -> Begginin of filter coefficient table
%                   L4 -> Filter length (N)
%                   M0,M4 = 1;
%                   CNTR -> Filter length - 1 (N-1)
%
% Saidas     :      MR1 -> Sum of products (rounded and saturated)
%                   I0  -> Oldest input data value in delay line
%                   I4 -> Begginin of filter coefficient table
%
%
% Altera     :      MX0,MY0,MR
%
*****************************************************************************/
filtro_fir:
   MR = 0, MX0=DM(I0,M0), MY0=PM(I4,M4);
   DO sop UNTIL CE;
sop:
   MR=MR+MX0*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M4);
   MR=MR+MX0*MY0(RND);
   IF MV SAT MR;
   RTS;
.endmod;
```

COEF.H

```
{ coeficientes  ( h1  ) * 2^14 * 256
               (h2,h3) * 2^15 * 256
}
.init  coefs:
        -512,  256,    0,  0, 256, -256, 768,   512,  256, 4200704;

.init  coefs_x2:
        6656, 6400, 5888, -1262336;

.init  coefs_x3:
       -3072,-3328,-2304,   319488;
```

CONSTANT.H

```
.CONST  N               = 10;     {AJUSTE MANUAL}
.CONST  N_x2            = 4;     {AJUSTE MANUAL}
.CONST  N_x3            = 4;     {AJUSTE MANUAL}
.CONST  PILHA           = 0x2000; {AJUSTE MANUAL}
.CONST  N_SAMPLES       = 1024;   {AJUSTE MANUAL}

.CONST  N_div_2         = N/2;    {AJUSTE AUTOMATICO DAQUI PARA BAIXO}
.CONST  N_div_2_mais_1  = N/2+1;
.CONST  Ntimes2         = N*2;

.CONST  Nmenos1         = N-1;
.CONST  Nmenos1_x2      = N_x2-1;
.CONST  Nmenos1_x3      = N_x3-1;
```