BIBLIOTECA DE AQUISIÇÃO DE DADOS

TRABALHO DE GRADUAÇÃO INTERDISCIPLINAR

LEANDRO SANTIAGO RAMIRES
MÁRCIO TETSUYA MURASUGI

UNIVERSIDADE BRAZ CUBAS
MOGI DAS CRUZES – SP
2003

BIBLIOTECA DE AQUISIÇÃO DE DADOS

TRABALHO DE GRADUAÇÃO INTERDISCIPLINAR

ORIENTADOR : PROF. Dr. VALDEMIR CARRARA
PROF. Dr. ANTÔNIO CARLOS DA CUNHA MIGLIANO

UNIVERSIDADE BRAZ CUBAS
ENGENHARIA DE COMPUTAÇÃO

Mogi das Cruzes
2003

Comissão Julgadora	Rúbrica	Conceito
_____	_____	_____
_____	_____	_____

Mogi das Cruzes, 01 de Abril de 2003

Esta publicação foi aceita como Relatório Final de Trabalho de Graduação Interdisciplinar.

Mogi das Cruzes, 01 de Abril de 2003

Alunos:

_____

Leandro Santiago Ramires

_____

Márcio Tetsuya Murasugi

Orientadores:

_____

Prof. Dr. Valdemir Carrara

_____

Prof. Dr. Antônio Carlos da Cunha Migliano (Prof. de TGI)

_____

Diretor do Centro de Exatas e Tecnologia

AGRADECIMENTOS

A DEUS o grande criador do Universo, por ter nos dado a oportunidade de cursar uma Universidade.

Aos nossos pais grandes responsáveis por nosso sucesso, que sempre nos apoiaram e ensinaram como vencer as barreiras encontradas pela vida.

Ao Prof. Migliano por ter nos ajudado e incentivado na realização do projeto nos orientando e abrindo as portas do laboratório do CTA para a execução de testes do projeto.

Ao Prof. Valdemir Carrara que desde o início nos incentivou na realização do projeto, acreditando e nos orientando para o êxito do projeto.

Aos técnicos dos laboratórios de Hardware e Física da Universidade Cláudio Remo Truffa, Pedro Yukio Tsuge e Paulo Ferreira que nos ajudaram bastante com suas experiências, em diversas fases do projeto.

Aos nossos colegas de trabalho Tibor Beles, Jorge Tokuda, Ricardo Hayashi e Rinaldo Bersi pelo auxilio e compreensão durante a fase de execução desse projeto.

RESUMO

Sistemas de Aquisição de Dados é um tema que carrega uma boa base teórica para estudantes de Engenharia de Automação e Computação, pois ele é base para conceitos de transmissão de dados, conversão de dados, sistemas operacionais, sistemas de tempo real e a interação *hardware* / *software*. Para se ter um sistema de aquisição em laboratório é necessário possuir um dispositivo de aquisição e um *software* de aquisição. Mas atualmente não é comum encontrar nas universidades esses componentes para que alunos possam criar seus sistemas de aquisição, isso ocorre por um conjunto de fatores, em primeiro lugar uma placa de aquisição convencional hoje custa em torno de U$ 1.900,00 e os *softwares* de aquisição geralmente são construídos sob plataforma UNIX, que não é uma plataforma simples de operação e também não é encontrada como padrão na maioria das Universidades. Neste sentido foi pensado em montar um sistema de aquisição onde o dispositivo de aquisição fosse a placa de som, hoje muito popular devido ao seu baixo custo que está em torno de R$ 50,00, e também a criação de uma biblioteca de uso genérico através de uma linguagem de programação em ambiente Windows, que realizasse a interface entre placa e aplicação de aquisição. Desta forma os alunos de Engenharia poderiam criar seus próprios projetos de aquisição, utilizando-se de ferramentas de fácil utilização como Visual Basic, Delphi, Matlab, etc, e assim aplicar os conceitos teóricos dos temas citados acima de uma forma prática, simples e barata.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1 - INTRODUÇÃO

Sistemas de Aquisição para fins didáticos são de extrema importância para que conceitos de Sistemas Operacionais, Sistemas de Tempo Real possam ser vivenciados na prática por alunos de Engenharia de Computação e Automação. Hoje os componentes necessários para ser ter um sistema de aquisição não são comuns na maioria das Universidades por causa do alto custo do componente de aquisição. Outra dificuldade é o desenvolvimento do *software* de comunicação entre o *hardware* e o *software*. Esse trabalho visa a criação de uma biblioteca de uso genérico no ambiente Windows, para aquisição de dados utilizando a placa de som do PC como dispositivo de aquisição, que apesar de algumas limitações atende perfeitamente a práticas didáticas em Universidades. O documento a seguir apresenta conceitos de sistemas de aquisição e sistemas de tempo real, no decorrer do trabalho serão apresentados exemplos de sistemas que poderão ser criados com a utilização do componente desenvolvido no projeto.

1.1 MOTIVAÇÃO

O que motivou a confecção do projeto, foi a possibilidade de colocar em prática os conceitos de engenharia adquiridos na universidade e também a possibilidade da criação de um componente de uso genérico para uso dos alunos e professores de engenharia de computação e controle e automação da Universidade Braz Cubas.

Devido ao fato de a universidade possuir apenas algumas placas de aquisição de dados, a utilização da placa de som como componente de aquisição veio a contribuir para o valor do projeto, pois desta forma a universidade possuiria centenas de placas de aquisição, uma vez que quase todos os PC’s da Universidade as possuem.

A facilidade de obtenção do *hardware* de aquisição e a criação do componente, que poderá ser utilizados pelas linguagens de programação comerciais em ambiente Windows, acreditamos que ajudarão não só o desenvolvimento de novos TGI's mas também será uma ferramenta de grande valor para disciplinas lecionadas na Universidade.

1.2 OBJETIVO

O objetivo deste projeto é desenvolver uma biblioteca de aquisição de dados de tempo-real (do tipo Soft devido as limitações do sistema operacional Windows), utilizando a entrada auxiliar da placa de som de um PC, como dispositivo de aquisição, em ambiente Windows. O trabalho visa demonstrar a utilização da placa de som como um dispositivo de aquisição de dados de boa taxa de amostragem, freqüência e principalmente de baixo custo e de fácil disponibilidade para utilizações didáticas por alunos de engenharia.

1.3 CONTEÚDO DO TEXTO

O texto apresenta o desenvolvimento do projeto de aquisição, os resultados e os conceitos teóricos aplicados, desta forma está dividido em capítulos para uma melhor visão do seu conteúdo.

O capitulo 1 apresenta a utilidade e os objetivos do projeto.

O capitulo 2 apresenta os conceitos teóricos utilizados para confecção do projeto, são fornecidas noções de sistemas de aquisição de sistemas de tempo real.

O capitulo 3 apresenta o projeto, como o mesmo está estruturado de que forma foi desenvolvido, etc.

O capitulo 4 apresenta e explica os resultados da fase de testes do projeto.

O capitulo 5 apresenta a conclusão projeto desenvolvido, explicitando os resultados conseguidos.

Nos anexos são apresentados exemplos de aplicações desenvolvidas com a utilização da biblioteca.

CAPÍTULO 2 - CONCEITOS TEÓRICOS

2.1 SISTEMAS DE AQUISIÇÃO DE DADOS

Os sistemas de aquisição encontram-se presente na vida do homem em todos os momentos, pois ele próprio é exemplo de sistemas desse tipo, quando experimenta o mundo real através dos seus sentidos (olfato, paladar, visão, audição e tato), processa essas informações e muitas vezes gera uma reação.

No cotidiano humano podem ser destacados exemplos comuns de sistemas de aquisição tais como: medição de água e luz, pesquisas de opinião, exames e diagnósticos médicos.

Atualmente com o avanço da microinformática e a popularização dos computadores (PC), a utilização de sistemas de aquisição vem crescendo no campo da engenharia, onde podem ser destacados exemplos de pesquisas de laboratórios, testes e medição, automação e controles industriais.

A seguir serão apresentados e explicados os conceitos e os componentes de um sistema de aquisição.

2.1.1 Definição

Os sistemas de aquisição podem ser definidos como medição de grandezas físicas do mundo real, tais como : temperatura; luz; pressão; força; deslocamento. Essas grandezas físicas em sua maioria são de natureza analógicas. Essas grandezas apresentam energia que necessitam ser captadas e convertidas para que possam ser tratadas pelos circuitos eletrônicos, para isso faz-se necessária a utilização de alguns elementos. Esses elementos são os sensores e os transdutores que captam as grandezas analógicas do mundo real a as transformam em quantidades elétricas como tensão, corrente ou impedância.

Além dos sensores e transdutores são necessários elementos que validem que os dados amostrados possuam relação com o que está se medindo.

2.1.2 Arquitetura dos Sistema de Aquisição

Um sistema de aquisição de dados deve captar os sinais através dos sensores e transdutores passar por um condicionador de sinais para que as especificações (corrente, tensão e impedância) necessárias da placa de aquisição sejam cumpridas, na placa de aquisição os dados são convertidos para digitais e processados pelo processador, segue as regras da aplicação e esse processamento resulta em alguma possível correção no sistema onde é enviado um sinal digital que é convertido para analógico na placa de aquisição que através de atuadores atuam no sistema, o importante é que todos esses elementos interajam entre si.

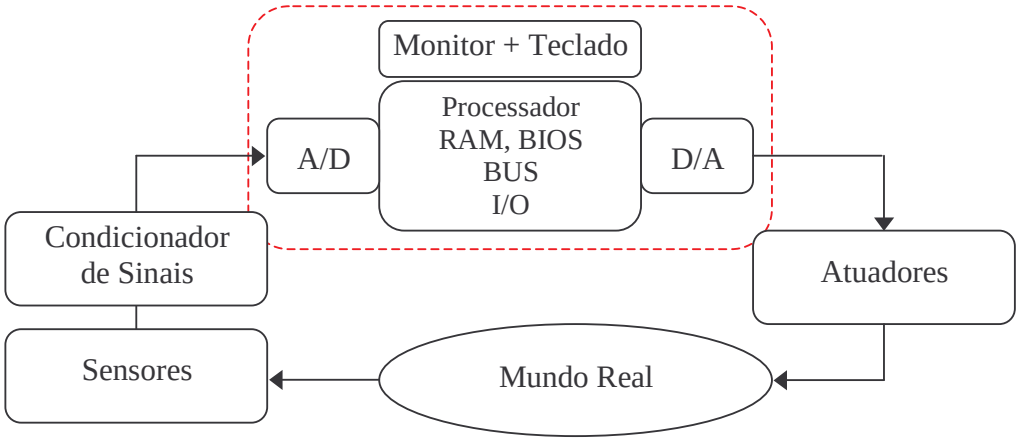


Figura 2.1 – Arquitetura dos Sistemas de Aquisição e Controle

2.1.3 Componentes de um sistema de aquisição

Os sistemas típicos de aquisição de dados em aplicações no ramo da engenharia elétrica, apresentam os seguintes componentes:

- Sensores e Transdutores;
- Condicionador de Sinais;
- Placa de Aquisição;
- Processador;
- Programas.

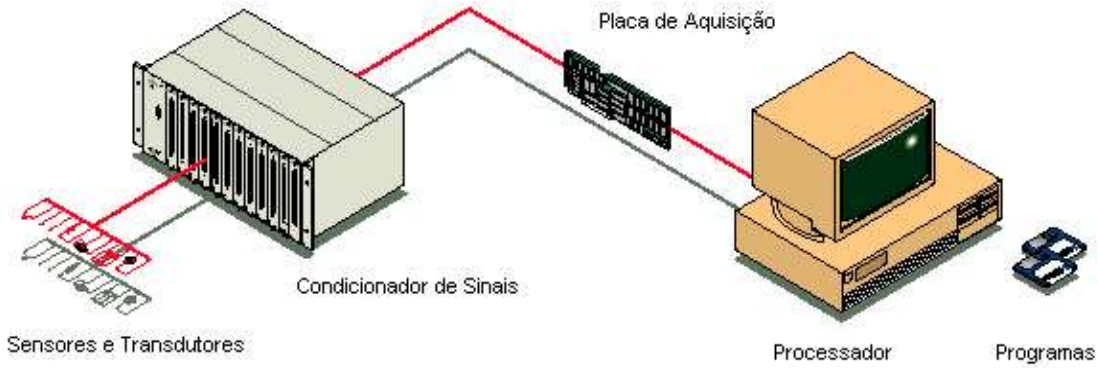


Figura 2.2 – Componentes dos Sistemas de Aquisição (Fig. Extraída da Apostila Fundamentos de Aquisição na National Instruments[1])

2.1.3.1 Sensores e Transdutores

São dispositivos que mudam seus comportamentos sob a ação de uma grandeza física, podendo fornecer diretamente ou indiretamente um sinal que indica esta grandeza. Quando operam diretamente, convertendo uma forma de energia neutra, são chamados transdutores. Os de operação indireta alteram suas propriedades, como a resistência, a capacitância ou a indutância, sob ação de uma grandeza, de forma mais ou menos proporcional. Por exemplo transdutores de fluxo, transdutores de pressão, sensores foto-elétricos, etc. Novamente é importante lembrar que os sinais elétricos gerados por esses componentes devem obedecer uma relação com os sinais monitorados.

2.1.3.2 Condicionamento de Sinais

Os sinais coletados pelos sensores e transdutores nem sempre atendem as imposições da placa de aquisição, pois os sinais podem ter freqüências, voltagens e impedâncias muito altas ou muito baixas, portanto faz-se necessário o condicionamento do sinal. Os métodos de condicionamento de sinais podem:

- Mudança no Nível - o deslocamento de nível é o método mais simples e mais usado em condicionamento de sinais. Por exemplo a necessidade de amplificar ou atenuar um nível de voltagem. Geralmente, em aplicações que resultam em sinais que variam lentamente com o tempo (baixas freqüências), onde amplificadores DC (Digital Code) ou de baixas freqüências podem ser utilizados. Os sinais são sempre representativos de alguma variável do sistema, e qualquer efeito de carregamento afetará a relação entre o sinal medido e o valor da variável.

- Isolar - é um método geralmente utilizado no condicionamento de sinais para proteção dos componentes do sistema, em caso de sinais medidos pelos sensores conterem uma voltagem muito acima do valor máximo tolerável pelo circuito de condicionamento e também para evitar a distorção do sinal por causa da diferença de terra. Nestas situações faz-se necessário o uso de amplificadores isolados para interfaciar este sinal com a aquisição do sinal.

- Multiplexar - é o método que serve para se medir diversos sinais com um único aparelho, para isso ele multiplexa os canais de amostragem e vai trocando esses canais indefinidamente.

- Filtrar - freqüentemente, sinais indesejáveis estão presentes no que está sendo medido. Em muitas situações é necessário a utilização de filtros passa altas, passa baixa ou rejeita faixa para eliminar ou minimizar

estes sinais. Estes filtros podem ser implementados apenas com elementos passivos, como resistores, capacitores ,indutores, ou filtros ativos, como uso de amplificadores realimentados.

- Excitar - é o método que serve para excitar alguns transdutores que requerem tensão externa ou sinais de correntes de excitação.

- Linearizar - é o grau de proporcionalidade entre o sinal gerado e a grandeza física. Quanto maior, mais fiel é a resposta do sensor ao estímulo. Os sensores mais usados são os mais lineares, conferindo mais precisão ao sistema. Os sensores não lineares são usados em faixas limitadas, em que os desvios são aceitáveis, ou com adaptadores especiais, que corrigem o sinal.

- Casamento de Impedância - O casamento de impedância é uma característica importante na interface entre sistemas, pois uma impedância interna do sensor ou a impedância da linha podem causar erro na medida da variável dinâmica. Neste caso, tanto malhas ativas ou passivas podem ser empregadas para realizar tal casamento.

2.1.3.3 Placa de Aquisição Convencional

As placa de aquisição são compostas dos seguintes componentes:

a) Entradas Analógicas

As especificações das entradas analógicas fornecem as informações referente a precisão do sistema. Nas especificações são encontrados dados sobre número de canais, taxa de amostragem, resolução e escala de amostragem.

- Número de Canais – são especificadas pelas entradas single-ended e diferenciais. Entradas single-ended são referenciadas a terras comuns, essas entradas são bastante utilizadas em sinais de entradas de alto nível, aquelas acima de 1V e que tenham distância pequena, menos de 3 m, entre os sensores e transdutores e a placa de aquisição. Outra

característica dessa entrada é que todos os sinais de entrada utilizam o mesmo terra. As entradas diferenciais são utilizadas para sinais de baixo nível, menores que 1V e quando se espera que os erros causados por ruídos sejam reduzidos, pois nessas entradas cada uma possui seu próprio terra.

- Taxa de Amostragem – neste parâmetro é determinada a freqüência em que as conversões são realizadas, quanto maior a taxa de amostragem mais original será a representação do sinal.

- Escala – esse parâmetro é referente aos níveis de tensão máximos e mínimos que o conversor pode quantizar.

b) Conversor Analógico / Digital

Conversores A/D (analógico / digital) convertem os sinais analógicos adquiridos pelos sensores e transdutores para digitais. A precisão dessa conversão é dependente de duas variáveis, resolução e linearidade do conversor . A característica mais importante nos conversores A/D é sua taxa de amostragem, ou seja, seu processamento, os elementos que especificam o processamento de um conversor são : tempo de conversão, tempo de aquisição e tempo de transferência.

- Tempo de conversão – é o tempo de conversão do sinal analógico para o sinal digital.

- Tempo de aquisição – é o tempo em que o sinal leva para ser adquirido.

- Tempo de Transferência – é o tempo em que o sinal leva para ir da interface para o centro de processamento.

Segundo o Teorema de Nyquist, “a quantidade de amostras por unidade de tempo de um sinal, chamada taxa ou freqüência de amostragem, deve ser maior que o dobro da maior freqüência contida no sinal a ser amostrado, para que possa ser reproduzido integralmente”.

c) Saídas Analógicas

Saídas analógicas são necessárias para gerar estímulos para o sistema, a qualidade do sinal analógico produzido depende das especificações do tempo de ajuste, slew rate e resolução de saída.

- Tempo de ajuste – é o tempo que a saída leva para alcançar um modo estável.
- Slew Rate – é a taxa máxima de mudança que o conversor D/A pode produzir para o sinal de saída.
- Resolução de Saída – é o número de bits no código digital que geram o sinal analógico.

d) Triggers

Existem triggers digitais e analógicos ambos servem para parar ou começar a aquisição, baseados em um evento externo.

e) Entradas e Saídas Digitais

Essas interfaces geralmente são utilizadas em sistemas baseados em PC para controlar processos, gerar padrões e comunicar com periféricos. A quantidade de dados utilizados nessas saídas e entradas digitais, dependem do que se quer controlar, caso o que se queira controlar seja equipamentos que não respondam rapidamente não existe a necessidade de uma alta taxa de dados. Também existe a possibilidade de se utilizar módulos de acionamento quando o que está sendo controlado necessita de uma voltagem e corrente alta, pois essas saídas trabalham em torno de 0 a 5 VDC e alguns miliamperes, esses módulos optoacoplados geram o sinal de potência necessário para controlar o dispositivo.

f) Contadores e Temporizadores

Contadores e temporizadores são utilizados geralmente para contagem de eventos digitais, temporização digital de pulsos e geração de ondas quadradas e pulsos. Para se obter essas aplicações três sinais de contadores e temporizadores são utilizados.

- Gate – é a entrada digital utilizada para habilitar ou desabilitar a função do contador.
- Fonte – é a entrada digital que causa o incremento do contador a cada pulso, portanto gera a base de tempo para operações de temporização e contagem.
- Saída – gera ondas quadradas ou pulsos na linha de saída.

As especificações mais importantes para contagem e temporização são a resolução e clock

- Resolução – número de bits que o contador utiliza. Uma alta resolução significa que o contador pode incrementar.
- Clock – determina a velocidade com que se pode ativar a fonte de entrada digital. Quando a freqüência é alta os pulsos de maior freqüência são gerados e as ondas tendem a serem mais quadradas na saída.

2.1.3.4 Placa de Som com Dispositivo de Aquisição

A placa de som é um periférico quase que indispensável nos PC`s sejam eles “domésticos” ou “comerciais”, e além de servir como interface de áudio do micro é geralmente a única interface analógica presente em um microcomputador, podendo ser considerada como um conversor analógico/digital e digital/analógico.

Sendo desta forma, a placa de som pode perfeitamente executar o papel de um dispositivo de aquisição, pois possui os componentes necessários

para esse fim e com a vantagem de ter um custo muito menor do que de uma placa de aquisição.

Uma desvantagem da placa de som em relação a placa de aquisição é que a captação de dados não é tão rápida; porém, para um laboratório didático a precisão é suficiente.

Abaixo são apresentadas algumas características das placas de som que são importantes sob a visão de um dispositivo de aquisição:

- ü Entradas e saídas: As placas de som normalmente possuem duas entradas e duas saídas. As entradas são chamadas line in e mic in. Já as saídas são chamadas line out e spk out, portanto pode-se ter dois canais de entrada e dois canais de saída. Uma observação importante a ser feita é que quando se utiliza a entrada Mic in, devido a mesma possuir uma impedância baixa (600 Ohms) essa entrada fica suscetível a ruídos.

- ü Amplificador: A placa de som geralmente possui duas saídas, uma de linha - chamada line out -, que não é amplificada, e uma saída amplificada, chamada spk out.

- ü Mixer: O mixer da placa de som é controlado por *software* e controla os níveis das entradas e das saídas da placa de som.

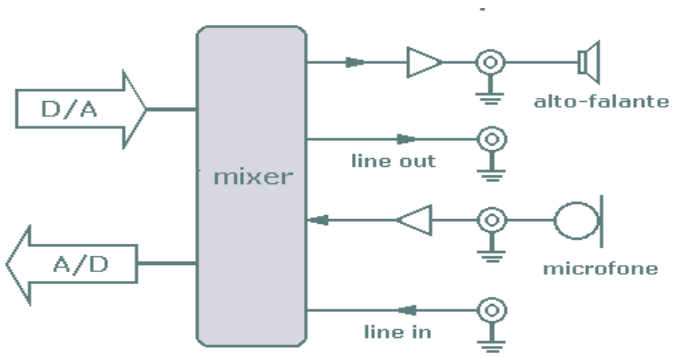


Figura 2.3 – Esquema das Entradas e Saídas de uma Placa de Som

- Resolução: O número de bits define a resolução do áudio ou dado capturado e reproduzido pela placa de som. As placas de som atualmente possuem 16 bits.

- Taxa de amostragem: É a quantidade de pontos por segundo que é capturada ou reproduzida pela placa de som, quanto maior for a taxa de amostragem maior será a originalidade da representação do sinal aquisitado. A maioria das placas de som trabalha com uma taxa máxima de 44.1 KHz. Diversas placas de som conseguem trabalhar com uma taxa de amostragem maior, em geral 48 KHz.

- Resposta de freqüência: É a faixa de freqüência que uma placa de som consegue capturar ou reproduzir. A resposta de freqüência padrão adotada mundialmente é a faixa de 20 Hz a 20 KHz. Placas de som de melhor qualidade possuem uma resposta de freqüência maior do que essa.

- Relação sinal/ruído: Essa característica mede o nível de ruído gerado pela placa de som. A maioria das placas de som possui uma péssima relação sinal/ruído.

- Conversor Analógico Digital / Digital Analógico: Os conversores digital/analógico (DAC) e analógico/digital (ADC) de uma placa de som têm um atraso inerente. A latência[1] típica de um conversor está na faixa de 30 a 50 amostras de áudio (samples), o que representa 1 a 1,5 milisegundos de atraso quando se opera com uma taxa de amostragem de 44.1 kHz. porém como citado anteriormente esse atraso para fins de experimentos didáticos não é impeditivo.

[1] Latência - O tempo necessário para que um sinal viaje de um ponto a outro dentro de um circuito ou rede.

2.1.3.5 Processadores

São os elementos responsáveis pelo processamento dos sinais adquiridos.

2.1.3.6 Programas

São os *softwares* específicos de cada aplicação, os mesmos tratam os sinais digitalizados e de acordo com a aplicação armazenam ou geram sinais na saída para controle de dispositivos.

2.1.4 Características dos Sistemas de Aquisição de Dados

2.1.4.1 Precisão

A precisão dos sistemas de aquisição sob os dados coletados depende dos dispositivos utilizados. A precisão pode ser avaliada atentando aos dados e as fontes que podem contribuir para o erro.

Um conceito encontrado sobre precisão é que ela descreve a quantidade de incerteza que existe em uma medição em relação ao padrão relevante absoluto.

2.1.4.2 Resolução

Em quantidade relativa, resolução é o grau pelo qual uma mudança pode ser detectada. Em sistemas de aquisição resolução é expressa como um número de bits. Mas não se pode utilizar a máxima resolução, por causa de fatores como por exemplo o ruído. Desta forma torna-se necessário utilizar alguns bits apenas para o ruído.

2.1.4.3 Sensibilidade

Sensibilidade é uma quantidade absoluta que descreve a menor quantidade absoluta de alteração que pode ser detectada pela medição, geralmente na casa de milivolts, ou décimos de grau. Sensibilidade atualmente é

mais uma função do equipamento de medição do que do meio onde se está sendo realizada a medição. A sensibilidade de 1μV nas medições realizadas por um equipamento pode ser perfeitamente alcançada se o cabeamento estiver adequadamente aterrado e não existir tensões geradas termicamente.

2.1.4.4 Ruído

Ruídos são os sinais indesejados que aparecem após a digitalização de um sinal aquisitado. Na utilização de PC, por ser um ambiente ruidoso, é necessário a construção do sistema em várias camadas. A utilização de aterramento metálico sobre o dispositivo ajuda a diminuir o ruído.

2.1.4.5 Calibração

Com a calibração pode-se ter uma maior precisão. Os elementos principais da calibração são os compensadores e os elementos de ganho. A calibracão de *hardware* por meio de programas utiliza conversores D/A para compensação, que cancela tensões e erros de ganho antes da conversão A/D acontecer. A calibração incorreta pode reduzir a precisão dos dados de um modo inesperado.

2.2 SISTEMAS DE TEMPO REAL

A grande importância dos sistemas de tempo real está se dando devido sua utilização em lugares onde o erro e o atraso não são admitidos, pois muitas vezes a segurança de pessoas, ganhos econômicos ou o meio ambiente estão em risco.

Atualmente os sistemas de tempo real estão presentes em áreas estratégicas como por exemplo na aviação em controladores de vôo, na área militar em controles de bombas teleguiadas e também na área médica em cirurgias e aparelhos de acompanhamento de pacientes (marcapasso).

O texto a seguir visa apresentar uma breve visão de conceitos, terminologias e aplicações de sistemas de tempo real.

2.2.1 Definição e Conceitos

O conceito de sistemas de tempo real segundo *Burns, A; Wellings* é :

"Qualquer sistema de processamento de informação que deve responder a estímulos externos em um tempo finito e determinado."

O que deve ser garantido em sistemas de tempo real é que cada função computada deve ser correta e precisa ser realizada em intervalos de tempo definidos. Assim, tanto o comportamento funcional como o comportamento temporal deve ser previsível e bem controlado.

A concepção de sistemas de tempo real está diretamente relacionada ao ambiente e seu comportamento temporal. Na classe dos sistemas de tempo real onde se encontram os sistemas de supervisão e controle esse é composto por sistema a controlar, sistema computacional de controle e o operador. No sistema como um todo o sistema a controlar e o operador são considerados como ambiente do sistema, a interação entre eles se dá através de interface de instrumentação (sensores, transdutores e atuadores) e da interface do operador como se pode observar na figura 2.4.

Figura 2.4 – Esquema de STR de Supervisão e Controle

2.2.2 Previsibilidade

Previsibilidade em sistemas de tempo real no domínio lógico e no domínio temporal é quando se garante, em tempo de projeto, que independente do *hardware*, carga e falhas o comportamento do sistema pode ser previsto antes de sua execução e a garantia de que os tempos especificados para o sistema sejam cumpridos. A maioria dos autores classifica a previsibilidade em dois tipos :

- ü Determinista – garante que todos os *deadlines* [2] a partir das interações com o ambiente, serão respeitados, ou seja que todos os eventos temporais serão cumpridos.

- ü Probabilista - baseadas em estimativas, probabilidades são associadas aos *deadlines* definindo assim as possibilidades que os mesmos sejam cumpridos. Esses são úteis onde a carga computacional não pode ser conhecida a princípio e desta forma não se tem garantias, em tempo de projeto, que os prazos serão atendidos.

A previsibilidade deve afetar todos os níveis dos sistemas de tempo real, linguagens, sistemas operacionais, comunicação e arquitetura do computador.

2.2.3 Classificação dos Sistemas de Tempo Real

Os sistemas de tempo real são classificados quanto a seus tipos de atuação, segurança e tempo de reposta.

2.2.3.1 Classificação quanto a tipo de sistemas de Tempo Real

- ü Sistemas Reativos – são os sistemas que reagem enviando respostas continuamente a estímulos de entradas vindo de seus

---

[2] *Deadline* - Instante máximo desejado para a conclusão de uma tarefa.

ambientes, ou seja, interagem continuamente com o meio ambiente, esses são os tipos mais utilizados em sistemas de tempo real.

- ü Sistemas Dedicados – são esses os sistemas que controlam *hardware* especializado, como por exemplo os sistemas de controle da mistura de ar/combustível no carburador, sistemas que controlam a unidade de medida inercial do ônibus espacial.

- ü Sistemas Semi-Acoplados – são esses os sistemas que operam parcialmente independentes do *hardware* no qual operam, esses tipos podem ser migrados para outros *hardwares* bastando rescrever partes da aplicação.

- ü Sistemas Orgânicos – são esses os sistemas que operam completamente independentes do *hardware*, geralmente esses sistemas possuem interface genérica com o usuário.

2.2.3.2 Classificação quanto a segurança e tempo de resposta

Os sistemas de tempo real podem ser classificados quanto a sua segurança e tempo de resposta, fatores esses primordiais nos diferentes tipos de aplicações de tempo real. Cada aplicação tem sua respectiva classificação, a seguir são apresentadas as classificações:

- ü Hard – são os sistemas onde é imprescindível que a resposta ocorra em tempos bem definidos e onde a não obediência das restrições de tempo levam a algum tipo de falha . Sistema de tempo real que necessitam ser Hard, são por exemplo sistemas embarcados em controladores de vôo.

- ü Soft – são os sistemas onde o tempo é importante mas não imprescindível, pois caso haja atrasos, apesar do sistema não operar em excelência o mesmo ainda atende os objetivos. Sistemas

que podem ser Soft são por exemplo sistemas bancários, onde o mais importante e comprometimento com a execução da tarefa e não com o tempo de execução.

- ü Real – são os sistemas do tipo Hard Real Time mas que necessitam de resposta extremamente curtas. Esses tipos de sistemas são utilizados por exemplo nos direcionamentos de mísseis.

- ü Firm – sistemas altamente críticos mas admiti-se uma pequena probabilidade de perda de *deadline*, são como sistemas soft real time mas não são aceitos atrasos . Esses tipos de sistemas são utilizados em video-conferência.

Na figura 2.5 pode ser observado um gráfico comparativo entre os sistemas do tipo Hard e do tipo Soft, nesse gráfico pode ser observado que os sistemas Hard necessitam de uma alta precisão em um curto espaço de tempo.

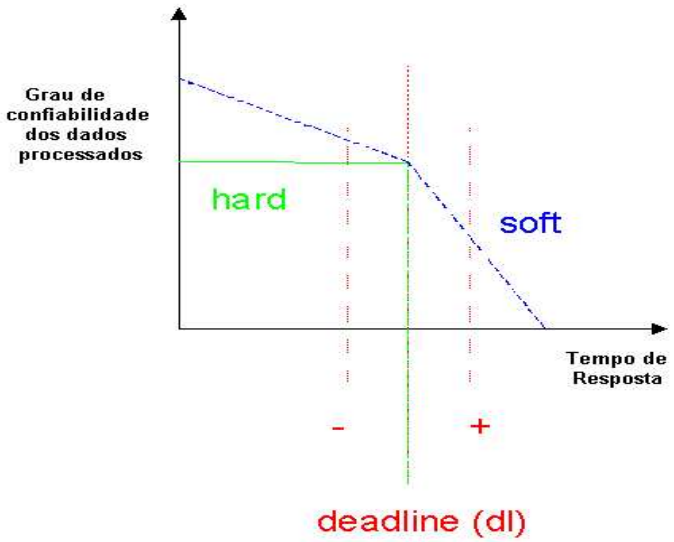


Figura 2.5 – Gráfico comparativo entre Hard e Soft

2.2.4 Sistemas Operativos

Os sistemas operativos funcionam como gerentes, eles que distribuem e executam serviços para as aplicações, os mesmos são responsáveis pelo

controle e coordenação da utilização do *hardware* da melhor forma possível, tendo em vista segurança e eficiência. Na figura 2.6 pode-se ter uma idéia da função dos sistemas operativos, que é justamente de interfaciar entre o *hardware* e as aplicações.

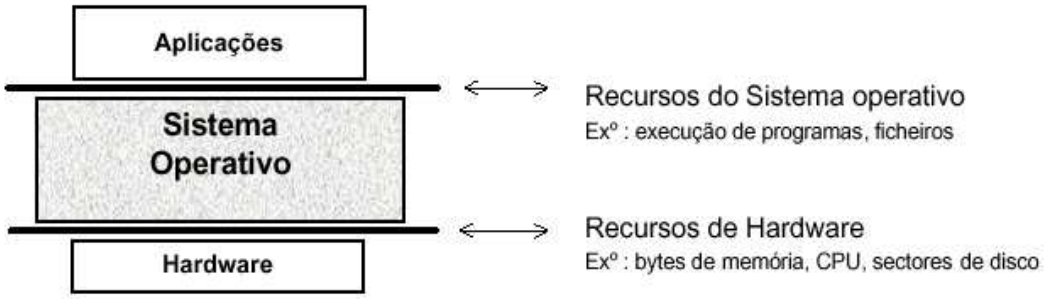


Figura 2.6 – Função dos sistemas operativos

Em sistemas de tempo real é necessário que haja um sistema operativo multitarefa na qual várias tarefas críticas devem ser processadas em simultâneo, e que as tarefas críticas sejam tratadas em tempo útil. Os sistemas operativos devem gerenciar os processos que compõe os sistemas de tempo real, atribuindo espaço para que cada processo seja executado sem comprometer a integridade temporal do sistema.

Alguns conceitos de sistemas operativos serão apresentados, para que se possa compreender a fundamental importância desse para os sistemas de tempo real.

- ü Tarefa (thread) – são abstrações que incluem um espaço de endereçamento próprio, um conjunto de arquivos abertos, um conjunto de direitos de acesso, um contexto de execução formado pelo conjunto de registradores do processador.

- ü Multitarefa – esse é o processo que consiste em escalonar e distribuir o tempo de CPU entre várias diferentes tarefas. O processo de multitarefa permite estruturar uma aplicação em um

conjunto de pequenas tarefas que permitem o compartilhamento da utilização do processador.

- Kernel - essa é a parte do computador responsável pela realização das tarefas e também a comunicação entre elas. O funcionamento do Kernel consiste em organizar as tarefas que estão sendo executadas e garantir a integridade da tarefa anterior, para isso o mesmo realiza uma salva guarda das tarefas correntes em suas *stacks*[3], por fim os endereços dessas *stacks* são guardados numa estrutura de dados que depois é gerida pelo sistema operativo. Kernel's de tempo real permitem que se tornem mais fácil de realizar e manter as aplicações.

- Interrupções – esse é o instante em que o código, para lhe dar com esse pedido começa a ser processado, os sistemas operativos desativam as interrupções quando estão tratando uma tarefa crítica em sistemas de tempo real o pedido de tempo de interrupção e de fundamental importância para o sucesso do sistema.

- Escalonador – essa é a parte do kernel responsável por decidir qual tarefa deve ser processada. O mesmo realiza a escolha das tarefas de acordo com seu grau de prioridade, as tarefas de prioridade altas são as primeiras a serem executadas.

[3] Stacks - são armazenadores de dados temporários que o sistema utiliza quando recebe uma chamada de interrupção de um dispositivo de hardware.

2.2.5 Windows como sistema operativo em tempo real

O sistema operacional Windows[4] é o mais utilizados entre milhões de usuários de PC's, este sistema foi concebido para ser um SO de abrangência geral, mais especificamente para ser um Sistema Operacional de Rede, embora esse possua algumas características de SO de tempo real, como por exemplo classe de processos de Tempo Real, ele não pode ser considerado como uma plataforma para aplicações de tempo real, pois o mesmo não foi concebido para esse fim.

O Windows apresenta falhas de segurança, alocação de memória e de aplicativos, o gerenciamento de prioridade apresenta poucos níveis e também existem problemas de inversão dessas prioridades, problemas esses vitais para aplicações de tempo real Hard.

O Windows pode suportar aplicações de tempo real do tipo Soft, onde o tempo de resposta não é imprescindível mas sim a confiabilidade dos dados operados.

Atualmente existem pacotes que reconfiguram o Kernel do sistema e que possibilitam que o Windows se aproxime de ser um sistema de tempo real, mas ainda sim esse sistema não pode ser considerado como SO de tempo real.

O Windows CE é a plataforma que suporta aplicações de tempo real, mas esse é um sistema para *palm top*, portanto não é possível sua utilização em PC's.

2.2.6 Exemplos de Aplicações de Sistemas de Tempo Real

Muitos são os exemplos de aplicações de tempo real que podem ser citado em diferentes ramos de atividades, como industrias, medicina e militar.

---

[4] Sistema Operacional Multitarefa da Microsoft que é utilizado na maioria dos PC's de usuários de baixa plataforma

Na industria pode-se citar o controle da mistura de substâncias (tintas), onde a falha de qualquer válvula exige uma correção imediata do sistema para alterar a proporção de mistura, para que desta forma não se perca a matéria prima.

Na medicina pode-se citar aparelhos de monitoração inteligentes de pacientes em UTI's[5], onde alterações no quadro do paciente deve resultar em uma intervenção, na dosagem de medicamentos, operação de aparelhos de apoio a respiração e aos batimentos cardíacos por exemplo.

Outros exemplos podem ser citados como Games, Sistemas Bancários, Sistemas de Controle de Tráfego Aéreo, operação de Usinas Nuclear, etc. Os sistemas exemplificados enquadram-se em sistemas hard-real time e sof – real time.

[5] UTI – Unidade de Tratamento Intensivo

CAPÍTULO 3 - PROJETO

Com a utilização de um computador PC é possível gravar e reproduzir sinais de áudio em formato Wave. Para o processo de aquisição e reprodução, a Microsoft fornece uma biblioteca de funções de baixo nível para manipulação de áudio para aplicações que operam sobre a plataforma Windows. Estas funções são extremamente úteis quando existe a necessidade de uma maior flexibilidade nas configurações e controle do dispositivo de entrada e saída de áudio.

3.1 PROCESSO DE AQUISIÇÃO

Em um processo de aquisição convencional utilizando um computador, o microfone funciona como um sensor de pressão que capta as diversas variações e as converte em sinais elétricos que são capturados pela placa de som. Estes sinais são digitalizados, agrupados em pacotes chamados de amostras (sample em inglês) e armazenados em *buffers*[6] que serão transferidos posteriormente para a memória principal. A partir daí, a CPU[7] poderá analisar e processar os dados armazenados.

A proposta deste projeto é utilizar outras fontes de sinais no lugar de um microfone como forma de aquisição de dados e assim ampliar o universo de aplicação da placa de som como uma ferramenta de análise de sinais.

3.1.1 Inicialização do Dispositivo

Antes de iniciar o processo de aquisição é necessário realizar uma série de configurações do dispositivo. Para o projeto optou-se por utilizar o formato

---

[6] Buffers - é um dispositivo de armazenagem temporária usado para compensar as diferenças na taxa de dados e o fluxo de dados entre dois dispositivos.

[7] CPU - Central Processing Unit / Unidade de Processamento Central.

PCM[8] como forma de armazenamento devido ao fato de ser o formato nativo do Windows. O PCM possui três propriedades que determinam a qualidade dos dados armazenados: a taxa de amostragem, a resolução e o número de canais. A taxa de amostragem padrão para a utilização da placa de som é determinada em 8000, 11025, 22050 ou 44100 amostras por segundo. A resolução pode ser configurada para 8 ou 16 bits. Quando a resolução for de 8 bits, o valor digitalizado será um valor entre 0 a 255. Caso a resolução seja de 16 bits, o valor digitalizado deverá variar entre -32768 a 32767. O número de canais pode ser configurado como 1 (mono) ou 2 (estéreo).

A função que suporta a inicialização do dispositivo de entrada é a w*aveInOpen*. Dentre os principais parâmetros pode-se destacar a identificação do dispositivo de entrada, a estrutura de configuração do dispositivo e forma de notificação quando o *buffer* for preenchido.

```
//** Configura o dispositivo de entrada **
WaveInitFormat();

//** Habilita o dispositivo de entrada **
result = waveInOpen(&m_hWaveIn,m_Dispositivo, &m_WaveFormat,
          (DWORD)m_WaveInEvent, NULL, CALLBACK_EVENT);
if(result != MMSYSERR_NOERROR)
{
        if (m_ExibeErro){
                char msg[255];
                waveInGetErrorText(result,msg,256);
                AfxMessageBox("Erro waveInOpen");
                AfxMessageBox(msg);
        }
        return FALSE;
}
```

O dispositivo de entrada que é utilizado pela biblioteca é a placa de som mas dentro das configurações de cada computador podem existir mais dispositivos como modem ou cartões de aquisição. Cada dispositivo recebe uma identificação interna do Windows e é esta identificação que deverá ser configurada

[8] PCM - Pulse Code Modulation - Formato de áudio desenvolvido pela Microsoft/IBM.

para a utilização da biblioteca. A identificação do canal de entrada pode ser adquirida utilizando a função *waveInGetDevCaps* .

A estrutura de configuração do dispositivo define o padrão de armazenamento dos dados de áudio. Dentre os principais atributos podem ser destacados o tipo do formato dos dados, o número de canais utilizados, a taxa de amostragem e a resolução.

O parâmetro da forma de notificação quando o *buffer* for preenchido define o procedimento que o dispositivo irá tomar no momento que um *buffer* for preenchido com os dados capturados. Esta configuração está diretamente ligada à estratégia de processamento das informações coletadas e é definida pelas seguintes alternativas de implementação:

- aviso automático na forma de uma mensagem do Windows;
- execução automática de uma função de *CallBack*;
- aviso automático através da um evento definido pelo usuário;
- execução automática de uma função ou *thread* (tarefa) definida pelo usuário;
- monitoramento de um *flag* de identificação do estado do *buffer*;

3.1.2 Preparação do Buffer

Durante o processo de aquisição, os dados de áudio digitalizados são carregados em *buffers* de memórias organizadas em fila circular. No processo de aquisição de dados, a organização circular possibilita a captura e o processamento dos dados coletados ao mesmo tempo. Assim enquanto um *buffer* é utilizado para o armazenamento dos dados coletados pelo canal de entrada, o outro *buffer* é utilizado para ser processado de forma concorrente. A figura 3.1 exemplifica o funcionamento de um *buffer* organizado em fila circular:

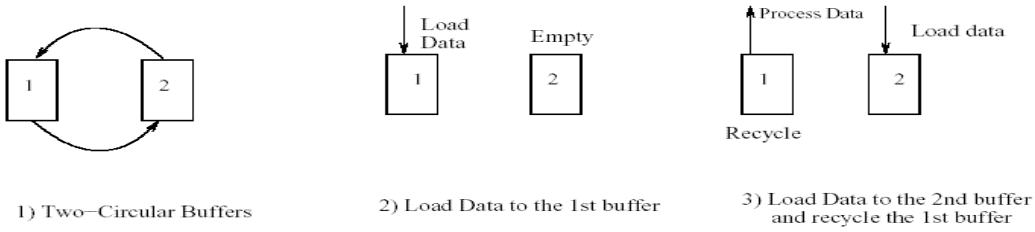


Figura 3.1 – Funcionamento de uma fila circular

Uma vez configurado o dispositivo, é necessário preparar o espaço onde os dados serão armazenados após a sua captura e digitalização. O tamanho estimado para a alocação de memória pode ser calculado utilizando a fórmula:

Tamanho do buffer = Tempo de aquisição(em segundos) x Nº de bytes por segundo

A função para a configuração do *buffer* é a *waveInPrepareHeader* e seu principal parâmetro é a estrutura WAVEHDR. Esta estrutura carrega consigo, dentre outros atributos, o tamanho e o endereço do *buffer* de armazenamento. Após a configuração é necessário disponibilizar o *buffer* ao dispositivo para o recebimento dos dados capturados. Para tal utilizamos a função *waveInAddBuffer*.

```
//** Configura e adiciona o cabeçalho de cada buffer na fila de aquisição **
for ( int i = 0; i < NUM_BUF - 1; i++ )
{

        //** Configura e prepara o cabeçalho para aquisição **
        WaveInitHeader(i);
        result = waveInPrepareHeader(m_hWaveIn, &m_WaveHeader[i],
                  sizeof(WAVEHDR) );
        if(result != MMSYSERR_NOERROR)
        {
                if (m_ExibeErro){
                        char msg[255];
                        waveInGetErrorText(result,msg,256);
                        AfxMessageBox("Erro waveInPrepareHeader");
                        AfxMessageBox(msg);
                }
```

```
                return FALSE;
        }

        //** Adiciona buffer à fila do dispositivo **
        result = waveInAddBuffer(m_hWaveIn, &m_WaveHeader[i], sizeof(WAVEHDR)
);
        if(result != MMSYSERR_NOERROR)
        {
                if (m_ExibeErro){
                        char msg[255];
                        waveInGetErrorText(result,msg,256);
                        AfxMessageBox("Erro waveInAddBuffer");
                        AfxMessageBox(msg);
                }
                return FALSE;
        }
}
```

Como foi descrita anteriormente, a utilização de uma fila circular de armazenamento é altamente recomendada nos processos de aquisição e reprodução e, antes de acionarmos as funções, devemos primeiro calcular a quantidade de *buffers* que serão necessários para o processo. Este cálculo baseia-se na aplicação que se pretende desenvolver e é empírico - no caso deste projeto optamos por utilizar 100 *buffers*. Assim para cada *buffer* da fila circular será necessário acionarmos as funções de configuração e disponibilização do *buffer*.

3.1.3 Início do Processo de Aquisição

Após a habilitação do dispositivo e do *buffer* de armazenamento, o dispositivo estará preparado para iniciar o processo de captura dos dados. Utilizando a função *waveInStart*, o dispositivo iniciará imediatamente a coleta dos dados de forma assíncrona.

```
//** Inicia o processo de aquisição **
result = waveInStart(m_hWaveIn);
if(result != MMSYSERR_NOERROR)
{
        if (m_ExibeErro){
                char msg[255];
                waveInGetErrorText(result,msg,256);
```

```
                AfxMessageBox("Erro waveInStart");
                AfxMessageBox(msg);
        }
        return FALSE;
}
```

Neste projeto optou-se por utilizar o método de chamada automática de uma função de aviso automático de um evento quando o dispositivo preencher um *buffer*. Isto facilita o controle, pois exime a responsabilidade de se monitorar o estado do *buffer* antes do mesmo ser processado.

3.1.4 Processamento dos Dados: Identificação do evento

Todas as vezes que a função *waveInStart* ou *waveInStop* for acionada, ou quando o *buffer* for preenchido pelo dispositivo, o sistema irá gerar um evento que será tratado por uma rotina interna da biblioteca. Para saber o motivo pelo qual a função foi acionada utiliza-se o parâmetro uMsg que é disponibilizada pela função e que contém uma mensagem de identificação do motivo da chamada. A mensagem WIM_DATA representa o acionamento devido ao preenchimento do *buffer* e é através dela que se dará o prosseguimento ou não ao processo de tratamento. Após a identificação de que um *buffer* foi disponibilizado, é realizada uma cópia de seu conteúdo para que se possa processar em cima da cópia e reciclar o *buffer* original.

```
UINT WaveInThreadProc(void * pParam)
{
        :
        :
while (!((CDALxCtrl*)pParam)->m_TerminateThread)
{
        //** Aguarda uma mensagem de buffer cheio **
        result = WaitForSingleObject(((CDALxCtrl*)pParam)-
>m_WaveInEvent,INFINITE);
        if ((result == WAIT_OBJECT_0)&&(!((CDALxCtrl*)pParam)-
>m_TerminateThread ))
        {
                Indice = ((CDALxCtrl*)pParam)->m_BufferCounter;
                Tamanho = ((CDALxCtrl*)pParam)-> m_WaveHeader[Indice].
                                                        dwBytesRecorded;
```

```
                //** Recicla o buffer**
                ((CDALxCtrl*)pParam)->AdicionaBuffer();

                //** Trata os dados**
                ((CDALxCtrl*)pParam)->TrataFila(Indice, Tamanho);

        }
        else
        {
                return 0;
        }

        return 0;
}
```

Como o intuito deste trabalho é criar uma biblioteca de aquisição de dados, foi adotada a premissa de que o ambiente de trabalho que a utilizará não será sempre o mesmo. Fatores como a velocidade de processamento da máquina, quantidade de memória, quantidade de aplicações ativas, ou mesmo aplicações mal projetadas podem ser cruciais para o bom funcionamento da biblioteca. Na tentativa de minimizar os impactos que tais condições podem causar, optou-se por criar uma fila circular interna que armazena e disponibiliza os dados coletados para as aplicações. Esta fila está estruturada da seguinte maneira:

- Índice : é um número seqüencial para possibilitar um controle rápido da aplicação que poderá realizar consistências se um mesmo valor foi lido repetidamente devido à velocidade de processamento da rotina de aquisição da aplicação. Este varia de 0 a 88000 e ao chegar ao último número retorna novamente a 0.

- Tempo: é um valor cuja unidade é milisegundo. O seu conteúdo é o valor adquirido pela função *GetTickCount* do Windows, que conta o tempo que o sistema operacional está ativo. Este atributo possibilita à aplicação ter noção exata do momento (tempo) da aquisição do valor do sinal.

- Valor: é o valor digitalizado do sinal coletado e caso a resolução configurada seja de 8 bits irá conter valores entre 0 a 255. Caso a resolução seja de 16 bits o valor armazenado será de -32768 a 32767.

```
void CDALxCtrl::TrataFila(int pIndice, int pTamanho)
{
	int	pos;
	int	OffSet;
	BYTE *bBuffer;
	short	*uBuffer;

	pos = pIndice * pTamanho;
	if (m_Resolucao == 8){
		OffSet = 1;
		bBuffer = new BYTE[m_TamanhoBuffer];
		memcpy(bBuffer, m_WaveHeader[pIndice].lpData, pTamanho);
	} else {
		OffSet = 2;
		uBuffer = new short[m_TamanhoBuffer];
		memcpy(uBuffer, m_WaveHeader[pIndice].lpData, pTamanho);
	}

	for ( int i = 0 ; i < pTamanho/OffSet; i++)
	{
		FILA[m_UltimoIndice].Indice = m_UltimoIndice;
		FILA[m_UltimoIndice].Tick = GetTickCounter() - FirstTick;
		if (m_Resolucao == 8){
			FILA[m_UltimoIndice].Valor = *(bBuffer+i);
		}else{
			FILA[m_UltimoIndice].Valor = *(uBuffer+i);
		}

		m_Valor = (short) FILA[m_UltimoIndice].Valor;

		if (m_GravaArquivo) fprintf(fp, "%d\n", m_Valor);

		m_UltimoIndice++;
		if (m_UltimoIndice == TAM_BUFFER) m_UltimoIndice = 0;
	}
}
```

Uma vez armazenados, os dados desta fila ficarão disponíveis para aplicação consultá-los e processá-los da melhor forma.

3.1.5 Finalização do Processo de Aquisição

Apesar das informações fornecidas pela Microsoft quanto à utilização das funções de manipulação de áudio não mencionarem nenhum cuidado especial, a finalização do processo deve seguir alguns passos específicos para evitar que a aplicação usuária destas funções apresente erros de “travamento”[9].

```
//** Termina o processo de aquisição **
result = waveInStop(m_hWaveIn);
if(result != MMSYSERR_NOERROR)
{
        if (m_ExibeErro){
                char msg[255];
                waveInGetErrorText(result,msg,256);
                AfxMessageBox("Erro waveInStop");
                AfxMessageBox(msg);
        }
}

//** Inicializa todas as variáveis internas do dispositivo **
result = waveInReset(m_hWaveIn);
if(result != MMSYSERR_NOERROR)
{
        if (m_ExibeErro){
                char msg[255];
                waveInGetErrorText(result,msg,256);
                AfxMessageBox("Erro waveInReset");
                AfxMessageBox(msg);
        }
}
```

```
//** Termina o processo de aquisição **
waveInStop(m_hWaveIn);

//** Inicializa todas as variáveis internas do dispositivo **
waveInReset(m_hWaveIn);
```

[9] Relatos de sites de discussões técnicas informaram problemas com a função waveInReset quando se utiliza o método de CallBack. Este problema pode ser contornado seguindo o procedimento apresentado e descoberto nós durante o desenvolvimento da biblioteca.

Primeiramente é necessário interromper o processo de aquisição utilizando a função *waveInStop*. Depois é preciso limpar a fila circular de aquisição do dispositivo utilizando a função *waveInUnprepareHeader*. É necessário configurar os atributos da estrutura WAVEHDR, conforme visto anteriormente, e acionar a função para cada *buffer* preparado. Realizado a limpeza da fila, é a vez de limpar as variáveis internas utilizadas pelo dispositivo acionando a função *waveInReset*. O último passo é desabilitar o dispositivo acionando a função *waveInClose*.

3.1.6 Acesso aos Dados pela Aplicação

O acesso aos dados pode ser realizado através de quatro maneiras: acionamento do método *LeValorAtual*, pelo método *LeValorBuffer* disponibilizadas pela biblioteca, pela propriedade Valor ou ainda pelas propriedades *FilaIndice, FilaTick e FilaValor.*

O método *LeValorAtual* devolve a estrutura contendo o índice, tempo e valor do último dado armazenado na fila circular. Este método atende os casos em que a aplicação possui ambiente favorável de processamento ou não necessita de grandes taxas de aquisição.

```
void CDALxCtrl::LeValorAtual(long FAR* Indice, long FAR* Tick, long FAR* Valor)
{
        *Indice = FILA[m_UltimoIndice].Indice;
        *Tick   = FILA[m_UltimoIndice].Tick;
        *Valor  = FILA[m_UltimoIndice].Valor;
}
```

Já o método *LeValorBuffer* lê a fila circular interna e devolve a estrutura contendo o índice, tempo e valor do índice a ser consultado, que é passado como parâmetro na função. Para aplicações que não precisam trabalhar em tempo real é recomendável utilizar este método e obter todos os valores coletados pelo dispositivo. O único cuidado que se deve ter com esta função é que, caso a velocidade de processamento da aplicação seja baixa, existe o risco de se perder valores antigos sobrepostos por valores novos devido a estrutura de armazenamento ser uma fila circular.

```
short CDALxCtrl::LeValorBuffer(long Indice, long FAR* Tick, long FAR* Valor)
{
	//** Consiste índice **
	if (Indice < 0 || Indice > TAM_BUFFER) return -100;

	*Tick   = FILA[Indice].Tick;
	*Valor  = FILA[Indice].Valor;

	if (Indice - m_UltimoIndice > TAM_BUFFER-200) return 2;
	if (Indice > m_UltimoIndice-1) return 1;

	return 0;

}
```

Na hora de escolher a função que será utilizada pela aplicação é necessário notar que a taxa de aquisição do sinal é dado pela configuração do dispositivo e pode variar entre 8000, 11025, 22050 ou 44100 Hz. Porém a taxa com que a aplicação irá acessar os dados na fila circular pode não ser a mesma e vai depender única e exclusivamente das condições de processamento e a maneira como foi construído o algoritmo de aquisição.

A propriedade Valor da biblioteca contém o valor instantâneo da aquisição e é configurada no momento do tratamento dos dados coletados.

Através da propriedade *FilaIndice* é possível configurar o índice dos dados que se deseja consultar da fila circular interna e tem utilização semelhante ao método *LeValorBuffer*. Os dados de retorno serão configurados nas propriedades *FilaTick e FilaValor*.

3.2 DESENVOLVIMENTO DO SOFTWARE

Alguns fatores determinaram as características de desenvolvimento e utilização desta biblioteca:

- Sistema operacional Windows - certamente este sistema operacional não é o mais indicado para aplicações de tempo real devido às diversas

características já apresentadas. Porém é o mais difundido e portanto a biblioteca terá uma maior aplicação por um número maior de usuários.

- Disponibilização em componente *ActiveX* - A opção pela criação de um componente *ActiveX* deve-se ao fato da grande maioria das ferramentas de desenvolvimento suportarem este tipo de biblioteca e sua utilização é bastante simples em relação a *DLL* (Dinamic Linkage Library). Ferramentas como LabView[10], MatLab[11] e Visual Basic[12] são usuárias em potencial da aplicação.

- Linguagem de desenvolvimento da biblioteca – para o desenvolvimento utilizamos a linguagem MS Visual C++ devido às necessidades de rapidez de processamento, maior controle das funções de aquisição e facilidade de manipulação do espaço em memória.

A biblioteca de aquisição, batizada de *DALx* (Data Aquisition Library *ActiveX*), foi construída sob o paradigma da orientação a objetos e inteiramente armazenada sob forma de classe. Os métodos e as propriedades públicas são listados a seguir:

Métodos

- *VerificaDispositivo*;
- *HabilitaDispositivo;*
- *DesabilitaDispositivo;*

---

[10] LabView (laboratory Virtual Instument Engineering Workbench) – Ambiente de programação utilizando recursos gráficos voltada para engenharia e desenvolvida empresa National Instruments Corporation.

[11] MatLab – Ambiente de manipulação de matrizes desenvolvida pela empresa Math Works.

[12] MS Visual Basic – Ambiente de programação que utiliza a linguagem Basic e desenvolvida pela Microsoft Corporation.

- *IniciaAquisição;*
- *TerminaAquisição;*
- *LeValorAtual;*
- *LeValorBuffer;*
- *ObtemFormatoAudio;*
- *ObtemNumeroCanais;*
- *ObtemQuantidadeDispositivo*.

Propriedades

- *Dispositivo;*
- *NoCanais;*
- *TaxaAmostragem;*
- *Resolucao;*
- *TamanhoBuffer;*
- *IndiceAtual;*
- *Valor;*
- *FilaIndice;*
- *FilaTick;*
- *FilaValor;*
- *ExibeErro;*
- *GravaArquivo*.

Os detalhes de cada método e propriedades são descritos a seguir.

3.2.1 Método: VerificaDispositivo

O método *VerificaDispositivo* verifica se existe um dispositivo de aquisição instalado no computador.

boolean VerificaDispositivo();

Valores de Retorno

O método retorna o valor verdadeiro caso exista dispositivo de entrada. Caso contrário retorna o valor falso.

3.2.2 Método: HabilitaDispositivo

O método *HabilitaDispositivo* configura os parâmetros de funcionamento do dispositivo, prepara o *buffer* de armazenamento dos dados coletados e habilita o dispositivo de entrada.

boolean HabilitaDispositivo();

Valores de Retorno

O método retorna o valor verdadeiro caso o componente tenha obtido sucesso no processo de habilitação. Caso contrário retorna o valor falso.

3.2.3 Método: DesabilitaDispositivo

O método *DesabilitaDispositivo* encerra o processo de aquisição e desabilita o dispositivo de entrada.

boolean DesabilitaDispositivo();

Valores de Retorno

O método retorna o valor verdadeiro caso o componente tenha obtido sucesso no processo de desabilitação. Caso contrário retorna o valor falso.

3.2.4 Método: IniciaAquisição

O método *IniciaAquisição* inicia o processo de coleta de dados.

boolean IniciaAquisicao();

Valores de Retorno

O método retorna o valor verdadeiro caso o componente tenha obtido sucesso no processo de inicialização. Caso contrário retorna o valor falso.

3.2.5 Método: TerminaAquisição

O método *TerminaAquisição* finaliza o processo de coleta de dados.

boolean TerminaAquisicao();

Valores de Retorno

O método retorna o valor verdadeiro caso o componente tenha obtido sucesso no processo de finalização. Caso contrário retorna o valor falso.

3.2.6 Método: LeValorAtual

O método *LeValorAtual* fornece os dados da fila circular do instante da chamada do método.

void LeValorAtual(

long* Indice,

long* Tick,

long* Valor

);

Parâmetros

Índice	long (4 bytes)	Número seqüencial de começa em 0 e vai até 88000 para controle simples da aplicação.
Tick	long (4bytes)	É um valor cuja unidade é milisegundo e o seu conteúdo é o valor adquirido pela função GetTickCount do Windows no instante da aquisição.
Valor	long (4 bytes)	É o valor digitalizado do sinal coletado.

3.2.7 Método: LeValorBuffer

O método *LeValorBuffer* fornece os dados da fila circular da posição determinada pelo parâmetro Indice.

short LeValorBuffer(

long Indice,

long* Tick,

long* Valor

);

Parâmetros

Indice – é um valor inteiro de 4 bytes que representa a posição do dado na fila circular a ser consultado.

Valor de Retorno

Tick	long (4bytes)	É um valor cuja unidade é milisegundo e o seu conteúdo é o valor adquirido pela função GetTickCount do Windows no instante da aquisição.
Valor	long (4 bytes)	É o valor digitalizado do sinal coletado.

3.2.8 Método: ObtemFormatoAudio

O método *ObtemFormatoAudio* retorna os formatos suportados pelo dispositivo.

long ObtemFormatoAudio(

short IdDispositivo

);

Parâmetros

IdDispositivo – Identificação do dispositivo de entrada

Valores de Retorno

O método retorna a combinação dos seguintes formatos:

Formato	Descrição
WAVE_FORMAT_1M08	11.025 kHz, mono, 8-bit
WAVE_FORMAT_1M16	11.025 kHz, mono, 16-bit
WAVE_FORMAT_1S08	11.025 kHz, stereo, 8-bit
WAVE_FORMAT_1S16	11.025 kHz, stereo, 16-bit
WAVE_FORMAT_2M08	22.05 kHz, mono, 8-bit
WAVE_FORMAT_2M16	22.05 kHz, mono, 16-bit

WAVE_FORMAT_2S08	22.05 kHz, stereo, 8-bit
WAVE_FORMAT_2S16	22.05 kHz, stereo, 16-bit
WAVE_FORMAT_4M08	44.1 kHz, mono, 8-bit
WAVE_FORMAT_4M16	44.1 kHz, mono, 16-bit
WAVE_FORMAT_4S08	44.1 kHz, stereo, 8-bit
WAVE_FORMAT_4S16	44.1 kHz, stereo, 16-bit

3.2.9 Método: ObtemNumeroCanais

O método *ObtemNumeroCanais* retorna a quantidade de canais disponíveis para aquisição.

short ObtemNumeroCanais();

Valores de Retorno

O método retorna o valor 1 para mono e 2 para estéreo.

3.2.10 Método: ObtemQuantidadeDispositivo

O método *ObtemQuantidadeDispositivo* retorna a quantidade de dispositivos de entrada existentes no computador.

long ObtemQuantidadeDispositivos();

Valores de Retorno

O método retorna a quantidade de dispositivos presentes no computador.

3.2.11 Propriedade: Dispositivo

Valor short (2 bytes) que representa a identificação do dispositivo de entrada. Deve ser configurado antes do acionamento do método *HabilitaDispositivo*.

3.2.12 Propriedade: NoCanais

Valor short (2 bytes) que representa a quantidade de canais de entrada. O valor deve ser 1(mono) ou 2(estéreo). Deve ser configurado antes do acionamento do método *HabilitaDispositivo*.

3.2.13 Propriedade: TaxaAmostragem

Valor long (4 bytes) que representa a taxa de amostragem de operação do dispositivo. O valor deve ser 8100, 11050, 22100 ou 44200. Deve ser configurado antes do acionamento do método *HabilitaDispositivo*.

3.2.14 Propriedade: Resolucao

Valor short (2 bytes) que representa a resolução de digitalização do sinal. O valor deve ser 8 ou 16. Deve ser configurado antes do acionamento do método *HabilitaDispositivo*.

3.2.15 Propriedade: TamanhoBuffer

Valor long (4 bytes) que representa o tamanho do *buffer* de armazenamento do dispositivo. O dispositivo retornará o *buffer* para tratamento somente após o *buffer* estiver cheio. Deve ser configurado antes do acionamento do método *HabilitaDispositivo*.

3.2.16 Propriedade: IndiceAtual (somente para consulta)

Valor long (4 bytes) que representa a posição atual da fila circular interna. Serve para que a aplicação possa controlar o tratamento da fila circular.

3.2.17 Propriedade: Valor (somente para consulta)

Valor long (4 bytes) que representa o último valor lido pelo dispositivo.

3.2.18 Propriedade: FilaIndice

Valor long (4 bytes) que representa o índice que se deseja consultar dos dados da fila circular interna.

3.2.19 Propriedade: FilaTick (somente para consulta)

Valor long (4 bytes) que representa o instante (tempo) da aquisição do dado (em milisegundos) da fila circular na posição especificada pela propriedade FilaIndice.

3.2.20 Propriedade: FilaValor (somente para consulta)

Valor long (4 bytes) que representa o valor digitalizado da fila circular na posição especificada pela propriedade *FilaIndice*.

3.2.21 Propriedade: ExibeErro

Valor boleano (verdadeiro/falso) que especifica se exibe ou não erros das funções *WaveIn* de aquisição em uma janela durante a execução da aplicação.

3.2.22 Propriedade: GravaArquivo

Valor boleano (verdadeiro/falso) que especifica se gera um arquivo texto com os dados coletados durante a execução da aplicação.

3.3 REQUISITOS DE HARDWARE

Os requisitos de *hardware* aqui explanados servirão como uma base para o planejamento do acondicionamento do sinal de entrada na placa de som. Os dados foram baseados nas especificações da empresa Creative Labs[13],

[13] As especificações mencionadas podem ser consultadas no anexo E deste documento.

fabricante das placas da família Sound Blaster, que são tidas como padrão nesta área. As especificações exatas dependerão de cada fabricante da placa utilizada e poderão ser consultadas no manual técnico ou na Internet no próprio site dos fabricantes. Apesar de não existir nenhuma complexidade técnica no acondicionamento, deve-se tomar um cuidado especial no acondicionamento do sinal pois qualquer erro pode danificar a placa de som ou mesmo a placa mãe do computador.

3.3.1 Tensão de entrada

A tensão de entrada é de aproximadamente 1 Volt RMS para entrada auxiliar (line-in) e de aproximadamente 15 mVolts RMS para entrada de microfone.

3.3.2 Impedância de entrada

A impedância do microfone é de 600 ohms e da entrada auxiliar (line-in) é de 47Kohms. Apesar da baixa impedância do microfone, é recomendável a utilização do line-in como canal de entrada pois a alta sensibilidade do microfone o torna sujeito a ruídos, além do fato de algumas placas alimentarem os microfones pela própria entrada. Nestes casos recomenda-se a utilização de um capacitor no circuito de acondicionamento para impedir esta tensão.

3.3.3 Sinais de corrente contínua e de baixa freqüência

Geralmente as placas de som possuem um capacitor no canal de entrada de microfone e no auxiliar de entrada (line in). Assim a aquisição de sinais de corrente contínua não é possível, tampouco os sinais de baixa freqüência devido às características de passa-alta do capacitor. Portanto a biblioteca de aquisição funcionará para sinais de corrente alternada e também de média e alta freqüência.

CAPÍTULO 4 - RESULTADOS OBTIDOS

Para avaliar a biblioteca desenvolvida foram realizados alguns testes quanto aos tipos de sinais que poderão ser coletados, a qualidade dos sinais coletados e aplicações da biblioteca. A metodologia e o ambiente utilizado para os testes serão detalhados em cada item de teste.

4.1 RESULTADOS RELATIVOS AOS TIPOS DE SINAIS QUE PODEM SER COLETADOS

O ambiente utilizado para os testes foi um computador PC com processador Intel Pentium II 700MHz, RAM 128 Mbytes, placa de som Sound Blaster AWE 64, gerador de sinais LG FG7002C, e *software* desenvolvido especialmente para a avaliação, sob o sistema operacional MS Windows 95.

Foram avaliados 4 tipos de sinais: senoidal, serrado, quadrado e contínuo. Devido ao fato da taxa de amostragem da placa ser especificada pelo fabricante como sendo de 20Hz à 44kHz, a faixa de freqüência avaliada foi de 10Hz à 22kHz, conforme o teorema de Nyquist.

Para a realização dos testes foi desenvolvido um *software* que utiliza a biblioteca de aquisição e, a partir dos parâmetros de configuração do dispositivo e o intervalo, coleta os dados oriundos da entrada line-in da placa. Para evitar que a aplicação influa na avaliação, foi utilizada a função da biblioteca de criação interna do arquivo dos valores coletados. Assim a aplicação funciona simplesmente como um acionador da biblioteca, finalizando a aquisição após o intervalo especificado.

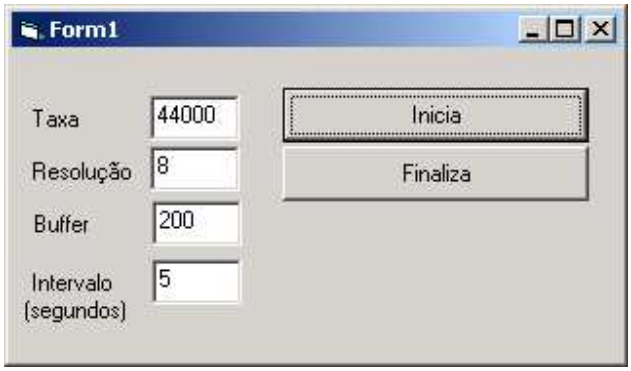


Figura 4.1 – IHM da aplicação para os testes

As configurações utilizadas para os testes e as faixas de freqüência avaliadas podem ser consultadas na tabela a seguir:

Taxa de amostragem (Hz)	Resolução (bits)	Buffer (bytes)	Intervalo de aquisição (segundos)	Freqüência avaliada (Hz)
8000	8	200	5	10, 100, 1000
8000	16	200	5	10, 100, 1000
11000	8	400	5	10, 100, 1000
11000	16	400	5	10, 100, 1000
22000	8	1024	5	10, 100, 1000
22000	16	1024	5	10, 100, 1000
44000	8	2048	5	10, 100, 1000
44000	16	2048	5	10, 100, 1000

Tabela 4.1 – Configurações utilizadas para os testes dos tipos de sinais

Conforme se pode observar nas figuras seguintes, os resultados obtidos foram bastante satisfatórios para o sinal senoidal, serra e quadrado. Para facilitar a visualização, inseriu-se uma curva calculada de referência para comparação com o sinal coletado.

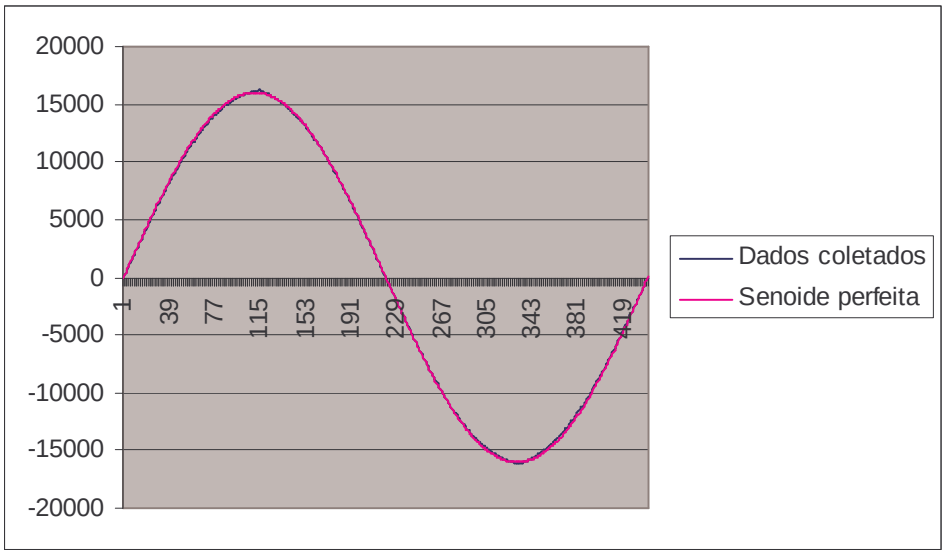


Figura 4.2 – Sinal senoide (gráfico Dados coletados x Senoide)

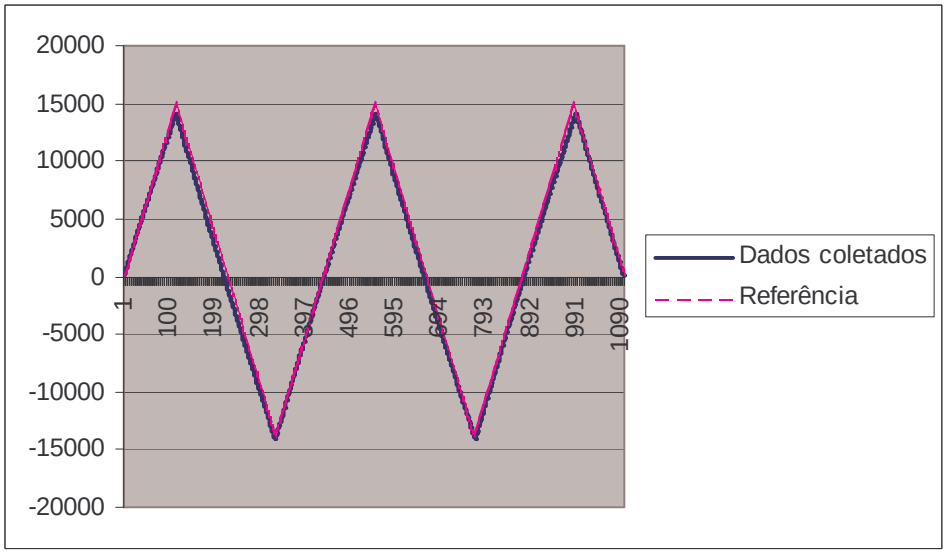


Figura 4.3 – Sinal serra (gráfico Dados coletados x Serra)

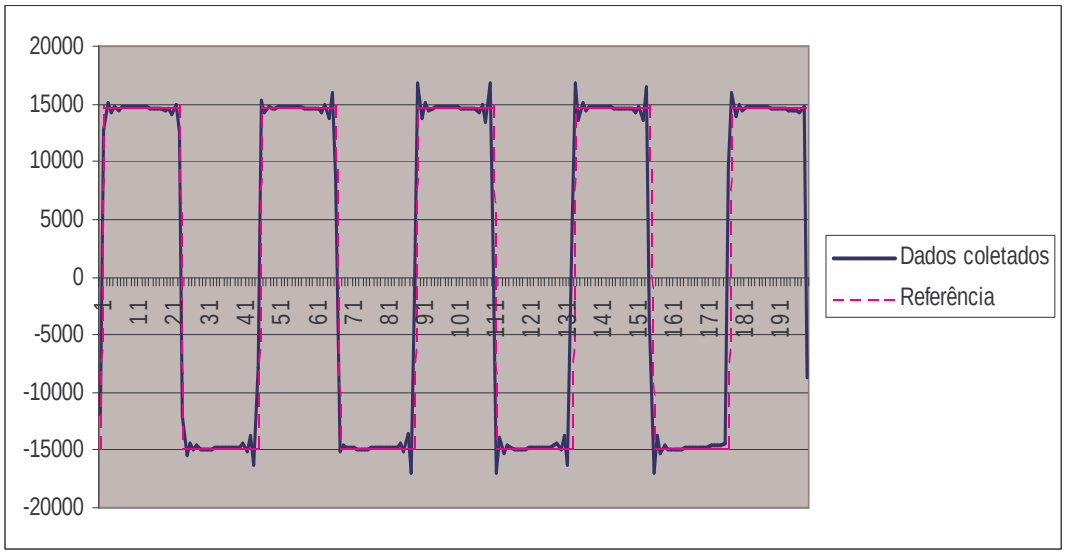


Figura 4.4 – Sinal quadrado(gráfico Dados coletados x Quadrado)

Como já era de se esperar, o sinal senoidal foi o que apresentou a melhor curva em relação aos sinais inseridos, devido à característica física da placa de som utilizada. O Sound Blaster, como a maioria das placas de som, possui um capacitor na entrada do microfone e line-in, interferindo nos sinais contínuos. Sendo assim os sinais que possuem maior confiabilidade ficaram na seguinte ordem decrescente: senoidal, serra, quadrado. Já o sinal contínuo foi totalmente ignorado pela placa devido à característica acima, conforme se pode ver na figura seguinte.

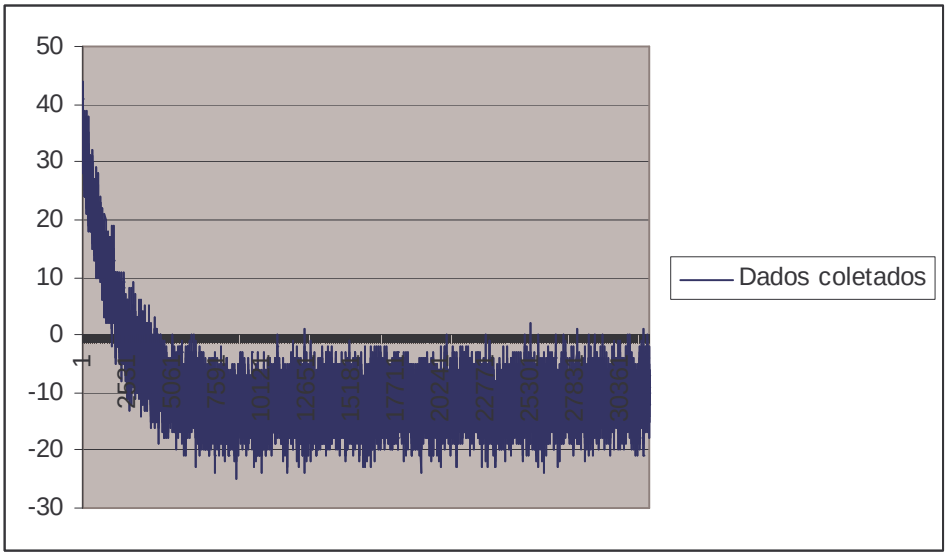


Figura 4.5 – Sinal Continuo

Os estudos referentes à qualidade dos sinais são melhor detalhados no item seguinte.

4.2 RESULTADOS RELATIVOS À QUALIDADE DOS SINAIS COLETADOS

A partir dos dados da avaliação dos tipos de sinais foi realizado um estudo quanto à qualidade dos sinais coletados. Utilizando a ferramenta MatLab foi realizado um estudo comparativo entre a forma da onda teórica e a forma da onda do dado coletado, analisando a correlação entre as transformadas de Fourier de ambos os sinais e verificando o coeficiente de correlação.

Antes do início dos testes, o resultado esperado era de que o sinal senoidal fosse o que apresentasse melhor qualidade devido ao capacitor que se encontra na entrada do microfone e line-in, interferindo em sinais contínuos. Sendo assim, também era esperado que o sinal quadrado fosse o de pior qualidade.

As avaliações comprovaram que os sinais senoidais apresentaram o maior índice de correlação, que em todos os casos apresentaram o valor 0,9999 em uma escala de 0 a 1, onde 1 representa a correlação perfeita. Em todas as

freqüências analisadas e taxas de amostragem foram observadas que a qualidade eram quase perfeitas, como é apresentado o quadro de resultados a seguir:

Taxa de amostragem (Hz)	Resolução (bits)	Freqüência (Hz)	Índice de correlação
8000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9999
11000	8	10	0,9999
		100	1,0000
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9999
22000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9999
44000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9999

Tabela 4.2 – Resultados da avaliação da qualidade dos sinais senoidais

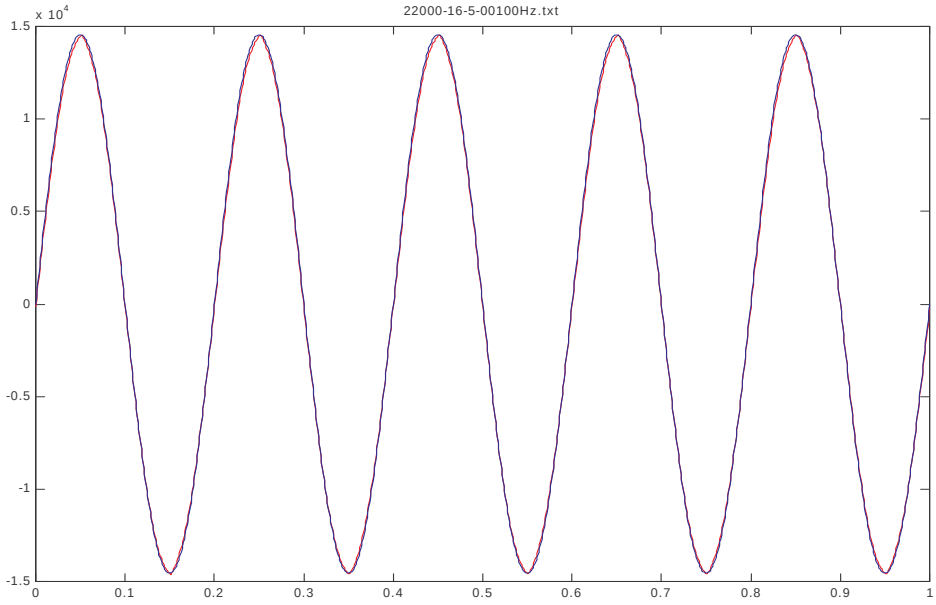


Figura 4.6 – Amostra de um sinal senoidal coletado e um sinal perfeito

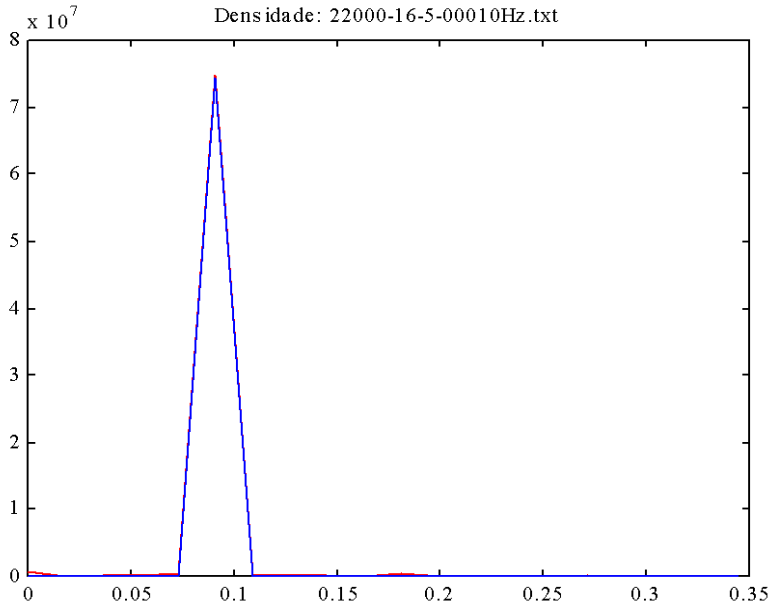


Figura 4.7 – Comparativo do espectro da freqüência do sinal coletado e do sinal perfeito

No caso dos sinais do tipo serra, nota-se novamente uma grande qualidade para as faixas 10 a 100 Hertz. Para os sinais com freqüência de 1000Hz observa-se uma pequena distorção na forma da onda.

Taxa de amostragem (Hz)	Resolução (bits)	Freqüência (Hz)	Índice de correlação
8000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9998
		100	0,9998
		1000	0,9959
11000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9998
		100	0,9998
		1000	0,9957
22000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9998
44000	8	10	0,9999
		100	0,9999
		1000	0,9999
	16	10	0,9999
		100	0,9999
		1000	0,9998

Tabela 4.3 – Resultados da avaliação da qualidade dos sinais do tipo serra

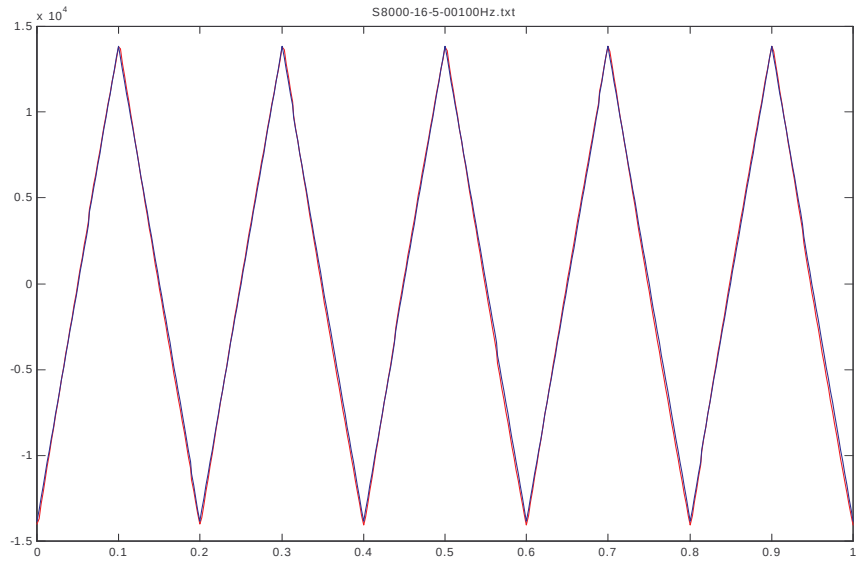


Figura 4.8 – Amostra de um sinal serra coletado e um sinal perfeito

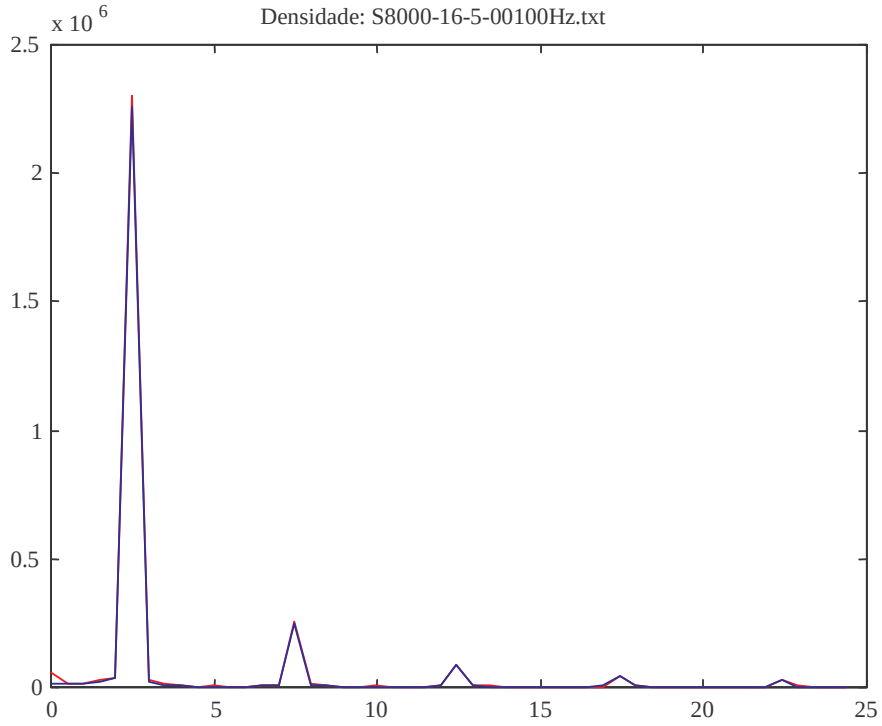


Figura 4.9 – Comparativo do espectro da freqüência do sinal coletado e do sinal perfeito

A onda de sinal quadrada foi o que apresentou pior qualidade da forma de onda. Novamente para sinais na freqüência acima de 1000Hz ficou nítido que existe uma degradação da forma da onda. Para as demais freqüências, dependendo da aplicação dos sinais, podem-se considerar satisfatórios os resultados obtidos, tendo em vista que o coeficiente médio ficou em 0,9968 aproximadamente.

Taxa de amostragem (Hz)	Resolução (bits)	Freqüência (Hz)	Índice de correlação
8000	8	10	0,9853
		100	0,9995
		1000	0,9853
	16	10	0,9946
		100	0,9961
		1000	0,9721
11000	8	10	0,9852
		100	0,9997
		1000	0,9984
	16	10	0,9944
		100	0,9971
		1000	0,9702
22000	8	10	0,9850
		100	0,9997
		1000	0,9998
	16	10	0,9943
		100	0,9985
		1000	0,9888
44000	8	10	0,9841
		100	0,9995
		1000	0,9998

	16	10	0,994
		100	0,997
		1000	0,9843

Tabela 4.4 – Resultados da avaliação da qualidade dos sinais do tipo quadrado

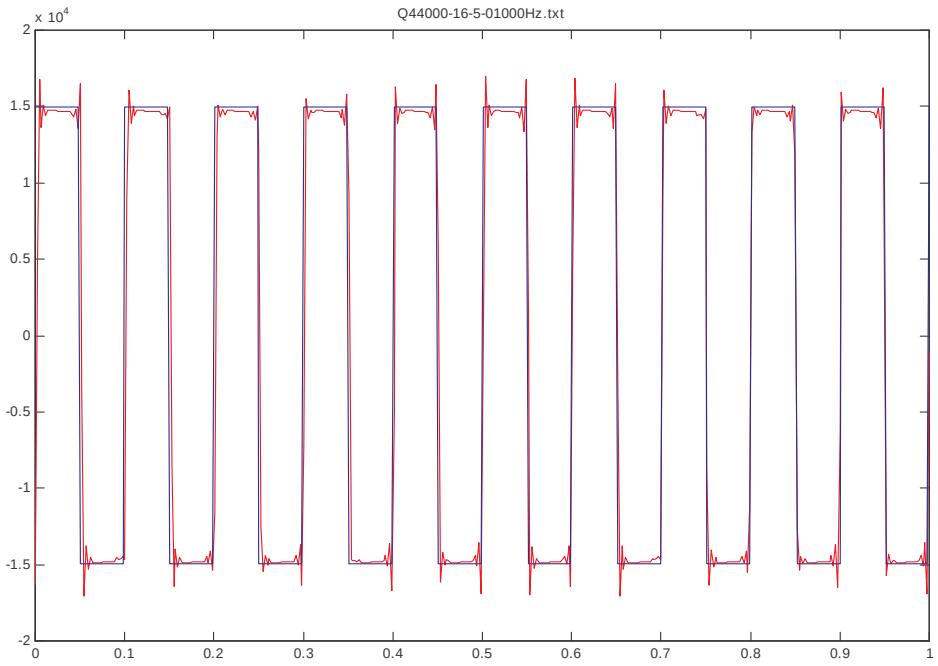


Figura 4.10 – Amostra de um sinal quadrado coletado e um sinal perfeito

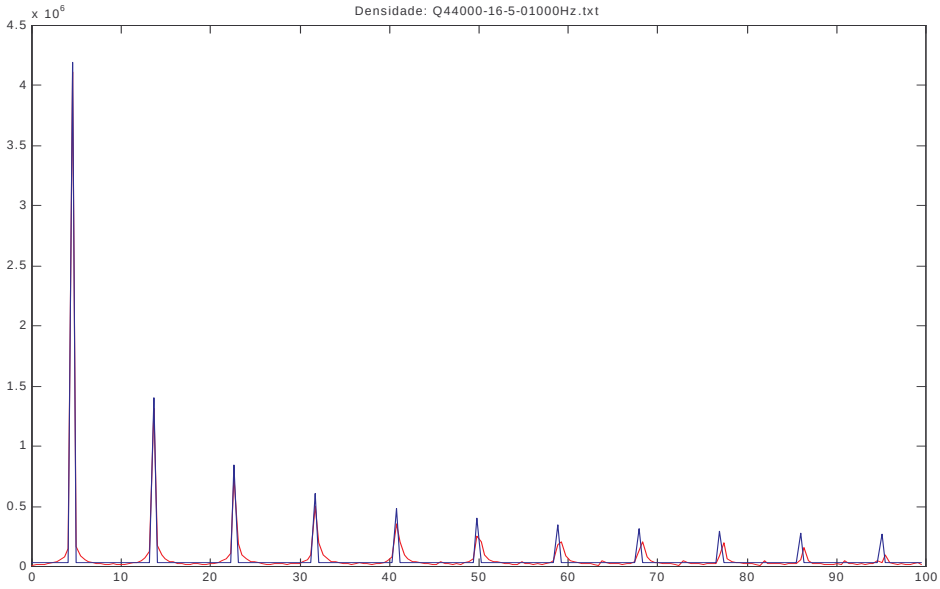


Figura 4.11 – Comparativo do espectro da freqüência do sinal coletado e do sinal perfeito

4.3 RESULTADOS RELATIVOS ÀS APLICAÇÕES DA BIBLIOTECA

Para a avaliação da utilização da biblioteca foram desenvolvidas três aplicações[14] semelhantes com os ambientes de programação MS Visual Basic, LabView e Matlab. A proposta das aplicações era desenvolver uma interface de onde fosse possível configurar os parâmetros do dispositivo de aquisição (placa de som) e visualizar a forma de onda da tensão da rede elétrica residencial de 220V.

O ambiente de desenvolvimento e testes foram instalados em um computador PC com processador Intel Pentium MMX 333MHz, 96 Mbytes de RAM e placa de som Sound Blaster AWE 64, sob o sistema operacional MS Windows

[14] Os códigos fontes das aplicações desenvolvidas para este experimento estão disponíveis no anexo deste documento.

2000. Também foi construído um pequeno circuito de acondicionamento[15] e casamento de impedância que transforma o sinal alternado de 220V para 1V.

Foram utilizados para esta avaliação as versões dos produtos MS Visual Basic 6.0 da Microsoft, LabView Evaluation 6i da National Instruments, e o MatLab 6.0 R13 MathWorks.

O fluxograma básico utilizado pelas aplicações é apresentado a seguir:

[15] O esquema do circuito de acondicionamento do sinal utilizado para este experimento está disponível no anexo deste documento.

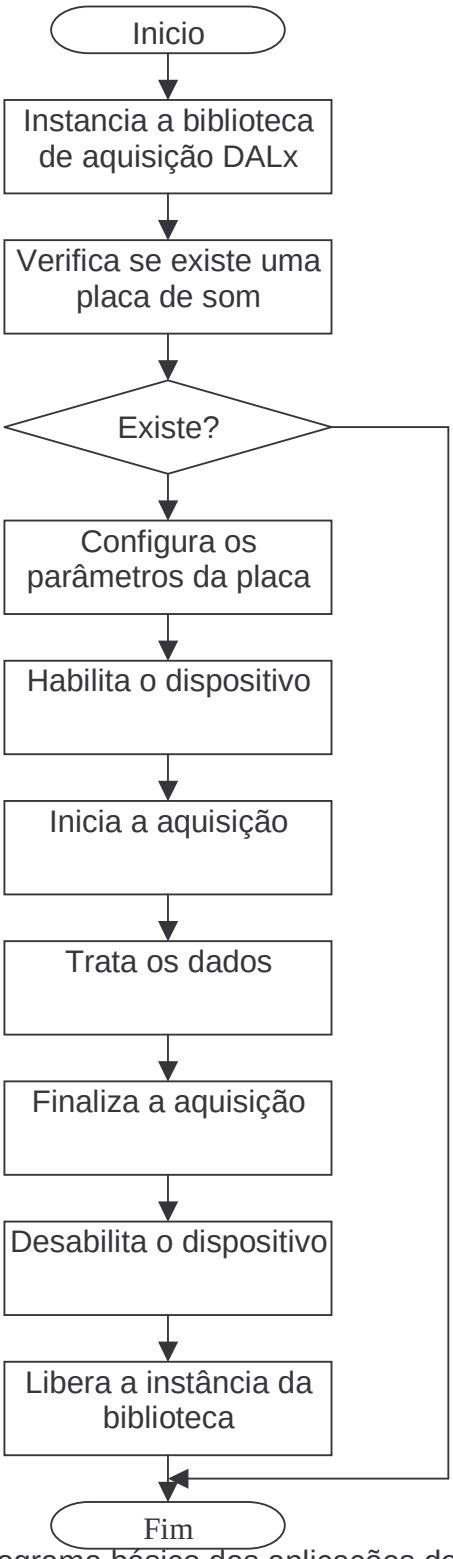


Figura 4.12 – Fluxograma básico das aplicações de teste da biblioteca de aquisição

Apesar dos três ambientes possuírem paradigmas distintos de programação[16], a forma de utilização do componente ActiveX é bastante semelhante entre si. Sendo assim foi verificado que é extremamente simples utilizar a biblioteca e montar aplicações de aquisição, mesmo sem um profundo conhecimento da linguagem de programação.

Quando se trata em desenvolver uma aplicação, observou-se que o usuário da biblioteca precisa levar em consideração alguns itens primordiais para o funcionamento de maneira esperada:

- velocidade do processador e quantidade de memória disponível;
- quantidade de aplicações em funcionamento no momento da aquisição ou exibição dos dados coletados – aplicativos como antivírus podem afetar o processamento dos dados;
- velocidade de processamento do ambiente de programação – ambientes que podem operar interpretando as linhas de comando como o Visual Basic e o LabView são lentos e podem influenciar no resultado esperado;

No desenvolvimento da aplicação em VB foi utilizado um exemplo disponibilizado no site da Microsoft MSDN e adaptado para a utilização da biblioteca. Para o processamento gráfico da onda utilizaram-se funções disponibilizadas pelo Windows e que se mostraram lentas no ambiente do VB. Este fato influenciou o processamento dos dados coletados em tempo real gerando falhas esporádicas na exibição da forma da onda. Para se verificar a influência do processamento gráfico na aplicação, foi desenvolvido uma função de criação de arquivo a partir dos dados coletados e observou-se que os todos os

[16] O Visual Basic é orientado a eventos, o LabView é orientado a componentes e o MatLab é estruturado.

dados foram registrados corretamente, comprovando assim que a visualização gráfica estava afetando o tratamento dos dados coletados.

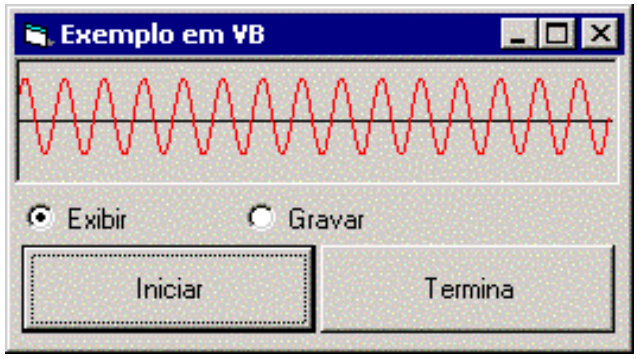


Figura 4.13 –Aplicação da biblioteca em VB

Para desenvolver a aplicação em LabView enfrentou-se um problema inesperado. Apesar do ambiente reconhecer o componente ActiveX e exibir os métodos e as propriedades da biblioteca, não funcionou apresentando um erro não identificado. Após pesquisa no site da National Instruments foi constado que outros usuários da versão Evaluation 6i também estavam tendo o mesmo problema e a resposta dada pela própria National era de que estava faltando, no programa de instalação, uma dll (biblioteca) que tratava exatamente do gerenciamento dos controles Activex. Assim a solução sugerida por ela era de realizar o *download* de uma versão run-time[17] posterior e sobrepor a instalação inicial. Este procedimento corrige o problema da instalação inicial.

Executado o procedimento, a aplicação finalmente funcionou e apresentou os resultados semelhantes ao da aplicação em Visual Basic. O processamento gráfico foi novamente um fator determinante para o funcionamento da aplicação. O tamanho da área de exibição do gráfico influenciava diretamente

[17] LabVIEW 6.0.2 Run-Time Engine

a velocidade do processamento e a solução encontrada foi diminuir a largura e altura do gráfico.

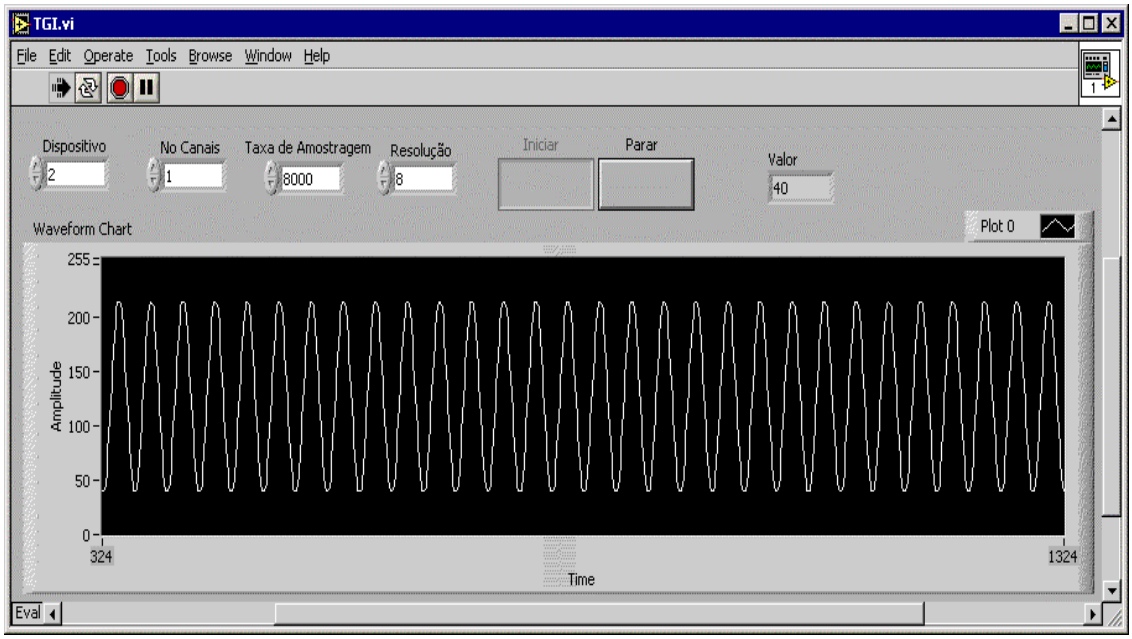


Figura 4.14 – Aplicação da biblioteca em LabView

No caso da aplicação MatLab, não foi utilizado o tratamento em tempo real dos dados coletados e portanto não apresentou os problemas citados nas aplicações anteriores. Utilizando o suporte aos controles ActiveX que o ambiente possui, a programação foi fácil e a utilização dos comandos muito semelhante ao do VB.

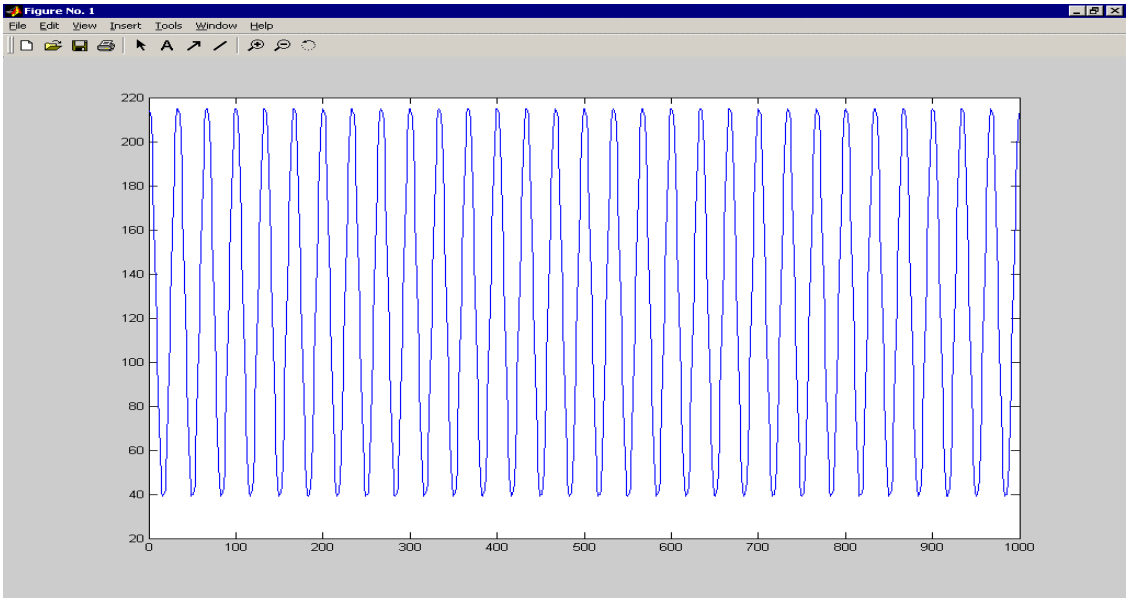


Figura 4.15 – Aplicação da biblioteca em MatLab

Os problemas apresentados pelas aplicações no tratamento gráfico dos dados em tempo real podem ser conseqüências de alguns fatores como a velocidade de processamento da máquina (Pentium MMX 333 MHz), falta de memória (96 Mbytes), a lentidão do ambiente de desenvolvimento (Visual Basic e LabView), e por fim, a própria arquitetura de *software* da aplicação. Também é sabido que o sistema operacional Windows não é um sistema de tempo real e que o tempo de latência de execução de comandos é lento em relação a outros sistemas operacionais.

Apesar da influência destes fatores, as aplicações apresentaram os resultados esperados dentro das limitações de *hardware* e *software*, mostrando que, em um ambiente controlado, a biblioteca de aquisição pode facilitar o trabalho de programação e tornar um microcomputador em uma ferramenta de estudo e análise de sinais, utilizando simplesmente um dispositivo padrão nas configurações atuais de PC: a placa de som.

CAPÍTULO 5 - CONCLUSÃO

O objetivo deste projeto era o de criar uma biblioteca de simples utilização para que os alunos das áreas de computação, mecânica e automação, que são os usuários em potencial do componente, pudessem realizar simulações de sistemas de controle e análise de sinais, em conjunto com os estudos teóricos sem a necessidade de um laboratório específico para auxiliar na carga de estudos práticos. Apesar de existirem limitações quanto aos tipos e as freqüências de sinais que podem ser coletadas, os resultados comprovam que o objetivo foi alcançado com sucesso tendo em vista a facilidade de operação que biblioteca oferece.

A qualidade dos sinais se mostrou bastante satisfatória e, dentro de um ambiente controlado, possibilita implementações de aplicações de relativa complexidade, por exemplo sistemas de análise de qualidade de energia cuja taxa de amostragem (44100Hz) e resolução (16 bits) atendem perfeitamente às pré-requisitos deste tipo de sistema.

Outros estudos como o do editor da revista Test&Measurement World, Brad Thompson, descrevem os seguintes parágrafos no artigo Sound Cards Work in Some Data-Acquisition Applications[18]:

"If your signal-analysis problems require high accuracy, long-term calibration, flexible configuration, and high-performance measurements, then a conventional data-acquisition card and software provides the better all-around choice.

A sound card and third-party data-analysis software, however, can provide an inexpensive alternative that's good for relative measurement and complex display of audio-frequency periodic signals and their spectra. My test

[18] Sound Cards Work in Some Data-Acquisition Applications

showed that a sound card's THD-measurement capabilities compare to those of the data-acquisition card."

O artigo ainda apresenta quadro comparativo entre uma placa de aquisição comercial e uma placa de som, apresentado suas vantagens e desvantagens.

Table 1. Comparison of a Data-Acquisition Card to a Sound Card for Analog-to-Digital Conversion

Parameter	National Instruments MIO-16XE-10PCI Data-Acquisition Card	Creative Labs AWE-32 PCI Sound Card
Number of analog inputs	16 single-ended AC/DC; eight differential AC/DC	two single-ended AC
Number of analog outputs	two AC/DC	two AC
Max. input sampling rate	100 ksamples/s	44.1 ksamples/s
Resolution and ADC technology	16-bit successive approximation	16-bit delta-sigma
Stated relative accuracy	±1 LSB max.; ± 3 μV (unity gain, calibrated)	not specified
External auxiliary inputs and Outputs	digital I/O; trigger; counter	two-channel microphone, loudspeaker, game port
Calibration source	built-in	user-supplied external
Frequency response	DC–255 kHz	10 Hz–22 kHz
Software suppliers	single source	multiple sources
Hardware documentation	very good	minimal from vendor
Tech support	excellent	uncertain

Tabela 5.1 – Comparativo de uma placa de aquisição comercial e uma placa de som

Porém a grande contribuição deste projeto não é o estudo da complexidade do algoritmo da biblioteca ou mesmo de um *hardware* especializado, mas é exatamente o oposto. De uma forma simples e com recursos que a grande maioria dos computadores atuais possuem, demonstrou-se que é perfeitamente viável desenvolver uma solução simples, prática e de grande aplicação em diversas áreas. A mudança de paradigma quanto à visão da utilização da placa de som é a principal contribuição deste trabalho desenvolvido. Um inocente dispositivo de multimídia torna-se uma ferramenta de aquisição, capaz de analisar sinais com certa precisão e possibilitar, a um custo baixíssimo,

construir aplicações simples ou média complexidade para estudos acadêmicos ou mesmo experimentos didáticos.

BIBLIOGRAFIA

[1] National Instruments Corporation, **Data Acquisition** (**DAQ**) **Fundamentals Introduction Today**

www.chipcenter.com/benchtop/tn004.html

[2] TONETO Carlos, DEUSDARÁ Romerito, SOUSA Rodrigo, **Introdução à Sistemas de Aquisição de Dados,** Junho 2002

mira.dee.bauru.unesp.br/~aguiar/ Cursos/Controle/P2/DAQ.pdf

[3] FARINES Jean, FRAGA Joni, OLIVEIRA Rômulo, **Sistemas de Tempo Real,** Julho 2000, Editora Escola de Computação 2000

[on-line] www.lcmi.ufsc.br/gtr/livro/sumario.htm

[4] MELO Amanda, BORIN Edson, **Tempo Real (RT CORBA),**

www.ic.unicamp.br/~ra007250/mo641/seminarioTR.ppt

[5] TIMMERMAN Martin, MONFRET Jean, **Windows NT as Real-Time OS?,**

http://www.realtime-info.be/magazine/97q2/winntasrtos.htm

[6] AITKEN Peter, JONES Bradley, **Guia do Programador C**, Berkeley Brasil Editora, 1994

[7] SHAMMAS Namir, **Programando em Visual C++ para Windows**, Berkeley Brasil Editora, 1993

[8] NATIONAL INSTRUMENTS, **Hands-On Course: LabView Basic I**, Course Software Version 5.1, February 1999 Edition

[9] NATIONAL INSTRUMENTS, **Hands-On Course: LabView Basic II**, Course Software Version 5.1, February 1999 Edition

[10] MATH WORKS, **Application Program Interface Guide,** Online Manuals (in PDF), University of California

http://www-ccs.ucsd.edu/matlab/pdf_doc/matlab/api/apiguide.pdf

[11] RÓJ Michał Konrad, **Real-Time Protocol-based Audio Engine**, University of Technology of Warsaw

http://home.elka.pw.edu.pl/~mroj/homepage/works/mroj/html/audio/audio-book.htm

[12] TSUI Jing, **Wave Input & Output**, Center for Advanced Computation & Telecommunications- Dept. of ECE

http://morse.uml.edu/~jtsui/WAVE.pdf

[13] MINDFIRE SOLUTIONS, **White Paper: Starting with VoIP**, March 2002

http://www.mindfiresolutions.com/download/Technical=Voice%20over%20IP.pdf

[14] PERKINS Travell, **Audio Capture Under Win32 NT/95**, Massachusetts Institute of Technology - MIT Media Lab

http://www.white.media.mit.edu/vismod/demos/speechcom/AudioCapture.txt

[15] INES TEST AND MEASUREMENT, **Sound card driver with FreeVIEW software**, FreeVIEW sound

http://www.inesinc.com/sound.htm

[16] LLOYD Alan G, **Tidal Flows - Using WaveIn Functions to Record Sound**, The Unofficial Newsletter of Delphi Users

http://www.undu.com/Articles/010323d.html

[17] NATIONAL INSTRUMENTS, **ActiveX and LabVIEW**, NI Developer Zone,

http://zone.ni.com/devzone/conceptd.nsf/webmain/5401BE584FBAEECE862567C2006D36C7?opendocument#

[18] NATIONAL INSTRUMENTS, **ActiveX Container Does Not Work in LabVIEW 6i Evaluation Version**, NI TroubleShooting KnowledgeBase

http://zone.ni.com/devzone/conceptd.nsf/webmain/5401BE584FBAEECE862567C2006D36C7?opendocument#

[19] DELL COMPUTERS CORPORATION, **Specifications: Sound Blaster® AWE64 Value Sound Card User's Guide**,

http://support.ap.dell.com/docs/acc/4110d/specs.htm

[20] MITSUBISHI ELECTRIC, **Sound Blaster Cards**

http://www.osemidlands.co.uk/support/insight/insight/en/common/addin/soundbla.htm

ANEXO A - EXEMPLO DE APLICAÇÃO EM VB

Esta aplicação criada em ambiente Visual Basic 6.0 demonstra como a biblioteca pode ser utilizada facilmente. Consiste basicamente em iniciar o processo de aquisição, coletar os valores e terminar o processo.

O exemplo abaixo visa criar uma janela onde pode-se visualizar ou registrar a forma da onda da corrente elétrica residencial devidamente acondicionada para a captação pela placa de som.

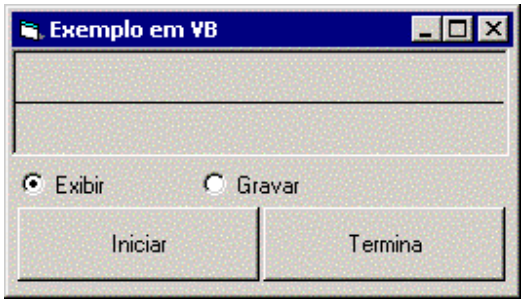


Figura A1 – IHM Aplicação VB

A IHM (Interface Homem-Máquina) possui:

- um controle PictureBox onde é exibido o gráfico dos dados coletados;
- um botão para iniciar o processo de aquisição;
- um botão para terminar o processo de aquisição;
- dois controles Option para selecionar entre exibir e registrar os dados.
- um componente de aquisição DALx.ocx.

Os resultados obtidos podem ser vistos a seguir:

a) Visualização da forma de onda

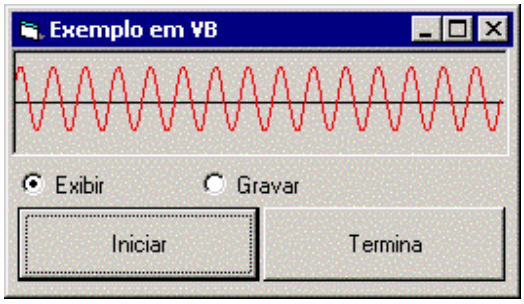


Figura A2 - IHM Aplicação VB apresentando senoide aquistada

b) Arquivo dos dados gerados

As colunas representam respectivamente o índice, instante de aquisição em milisegundos, e por último o valor coletado.

" 2"	" 241"	" 99"
" 10"	" 241"	" 99"
" 18"	" 241"	" 72"
" 26"	" 241"	" 47"
" 34"	" 241"	" 39"
" 42"	" 241"	" 40"
" 50"	" 241"	" 50"
" 58"	" 241"	" 82"
" 66"	" 241"	" 110"
" 74"	" 251"	" 145"
" 82"	" 251"	" 174"
" 90"	" 251"	" 199"
" 98"	" 251"	" 216"
" 106"	" 251"	" 215"
" 114"	" 251"	" 212"
" 122"	" 251"	" 184"
" 130"	" 251"	" 154"

" 138"	" 251"	" 122"
" 146"	" 251"	" 89"
" 154"	" 251"	" 64"
" 162"	" 251"	" 42"
" 170"	" 251"	" 39"
" 178"	" 251"	" 41"
" 186"	" 251"	" 59"
" 194"	" 261"	" 91"
" 202"	" 261"	" 120"
" 210"	" 261"	" 155"
" 218"	" 261"	" 182"
" 226"	" 261"	" 207"
" 234"	" 261"	" 215"
" 242"	" 261"	" 214"
" 250"	" 261"	" 205"
" 258"	" 271"	" 173"
" 266"	" 271"	" 145"
" 274"	" 271"	" 111"
" 282"	" 271"	" 80"
" 290"	" 271"	" 56"
" 298"	" 271"	" 39"
" 306"	" 271"	" 40"
" 314"	" 271"	" 42"
" 322"	" 281"	" 70"
" 330"	" 281"	" 100"
" 338"	" 281"	" 132"
" 346"	" 281"	" 165"
" 354"	" 281"	" 189"

Tabela A1 - Conteúdo do arquivo "ValoresVB.txt"

O código fonte foi extraído do exemplo "HOWTO: Create a Dynamically Scrolling Graph" do site do MSDN da Microsoft e adaptado para a utilização da biblioteca DALx. Utiliza basicamente a função BitBlt (disponibilizada pela Microsoft) de manipulação de imagem em memória para permitir o efeito de "rolagem" do sinal aquisitado. A fonte do gráfico é o sinal aquisitado pela biblioteca DALx e é exibido na janela conforme pode ser observado no exemplo.

CÓDIGO FONTE DE FORM1.FRM

```
'************************************************************
'*
'* Código original extraído do artigo "HOWTO: Create a
'* Dynamically Scrolling Graph" (MSDN) e adaptado para a
'* utilização da biblioteca de aquisição de dados.
'*
'* O gráfico é primeiramente criado em memória e então copiado
'* para o PictureBox usando a função BitBlt. Esta função per-
'* mite a transferência de blocos de bits de uma fonte para
'* para o destinatário.
'*
'* Para maiores informações do código e das funções utilizadas
'* estão acessíveis no MSDN no site:
'* http:\\www.microsoft.com\msdn\index.htm
'*
'************************************************************
'

Option Explicit

Private Const SRCCOPY = &HCC0020
Private Const PS_SOLID = 0

Private Declare Function CreateCompatibleDC Lib "gdi32" _
        (ByVal hdc As Long) As Long

Private Declare Function CreateCompatibleBitmap Lib "gdi32" _
        (ByVal hdc As Long, _
        ByVal nWidth As Long, _
        ByVal nHeight As Long) As Long

Private Declare Function SelectObject Lib "gdi32" _
        (ByVal hdc As Long, _
        ByVal hObject As Long) As Long
```

```
Private Declare Function CreatePen Lib "gdi32" _
            (ByVal nPenStyle As Long, _
            ByVal nWidth As Long, _
            ByVal crColor As Long) As Long

Private Declare Function LineTo Lib "gdi32" _
            (ByVal hdc As Long, _
            ByVal x As Long, _
            ByVal y As Long) As Long

Private Declare Function MoveToEx Lib "gdi32" _
            (ByVal hdc As Long, _
            ByVal x As Long, _
            ByVal y As Long, _
            ByVal lpPoint As Long) As Long

Private Declare Function BitBlt Lib "gdi32" _
            (ByVal hDestDC As Long, _
            ByVal x As Long, _
            ByVal y As Long, _
            ByVal nWidth As Long, _
            ByVal nHeight As Long, _
            ByVal hSrcDC As Long, _
            ByVal xSrc As Long, _
            ByVal ySrc As Long, _
            ByVal dwRop As Long) As Long

Private Const pWidth = 250        ' Width of picture box in pixels.
Private Const pHeight = 50              ' Height of picture box in pixels.
Private Const pGrid = 25          ' Distance between grid lines.
Private Const tInterval = 1       ' Interval between timer samplings
                                  ' in milliseconds.

Private Const pHeightHalf = pHeight \ 2 ' sync grid.
Dim oldY As Long                        ' Contains the previous y coordinate.
Dim hDCh As Long, hPenB As Long, hPenC As Long

Dim pIndice As Long
Dim Parar As Boolean

Dim fnum As Integer


Private Sub Command1_Click()

'   Inicia o processo de aquisição
    DALx1.Dispositivo = 2
```

```
    DALx1.NoCanais = 1
    DALx1.TaxaAmostragem = 8000
    DALx1.Resolucao = 8
    If (Not DALx1.HabilitaDispositivo) Then
        MsgBox "Erro ao tentar habilitar o dispositivo"
        Exit Sub
    End If

'   Inicia a aquisição
    DALx1.IniciaAquisicao

    If Option2.Value Then
        fnum = FreeFile()
        Open "ValoresVB.txt" For Output As #1
    End If

    pIndice = 2
    Parar = False
    While (Not Parar)
        If Option1.Value Then
'           Exibe o gráfico na janela
            Call ExibeValor
        Else
'           Gera arquivo com os valores
            Call GravaValor
        End If
        DoEvents
    Wend

    If Option2.Value Then Close #1

End Sub


Private Sub Command2_Click()
    Dim rs As Long

    Parar = True

'   Finaliza o processo de aquisição
    rs = DesabilitaDispositivo(H)

End Sub


Private Sub Form_Load()
    Dim hBmp As Long
```

```
    Dim i As Integer

    Me.Show

    Picture1.ScaleMode = 3
    Picture1.Left = 0
    Picture1.Top = 0
    Form1.ScaleMode = 3
    Picture1.Height = pHeight + 5
    Picture1.Width = pWidth + 5

    hDCh = CreateCompatibleDC(Picture1.hdc)
    hBmp = CreateCompatibleBitmap(Picture1.hdc, pWidth, pHeight)
    Call SelectObject(hDCh, hBmp)
    hPenB = CreatePen(PS_SOLID, 0, vbBlack)
    hPenC = CreatePen(PS_SOLID, 0, vbRed)
    Call SelectObject(hDCh, hPenB)

'   Plot horizontal grid lines.
    For i = pGrid To pHeight - 1 Step pGrid
       Picture1.Line (0, i)-(pWidth, i)
    Next

    Call BitBlt(hDCh, 0, 0, pWidth, pHeight, Picture1.hdc, 0, 0, SRCCOPY)
    oldY = pHeightHalf

End Sub


Private Sub ExibeValor()
    Dim rs          As Integer
    Dim i           As Integer
    Dim Tick        As Long
    Dim Valor       As Long

'   Aquisita um valor da fila circular
    rs = DALx1.LeValorBuffer(pIndice, Tick, Valor)
    Select Case rs
       Case 0          'Sucesso

       Case 1          'O índice a consultar é maior que o
          Exit Sub     '  o último índice da fila.

       Case 2          'O índice a consultar é maior que o
                       '  o último índice da fila devido
                       '  ao final da fila.
    End Select
```

```
    pIndice = pIndice + 4
    If pIndice >= 44000 Then pIndice = 2
    i = pValor / 5.5


    Call BitBlt(hDCh, 0, 0, pWidth - 1, pHeight, hDCh, 1, 0, SRCCOPY)
    Call SelectObject(hDCh, hPenC)
    Call MoveToEx(hDCh, pWidth - 3, oldY, 0)
    Call LineTo(hDCh, pWidth - 2, i)
    Call SelectObject(hDCh, hPenB)
    Call BitBlt(Picture1.hdc, 0, 0, pWidth, pHeight, hDCh, 0, 0, SRCCOPY)

    oldY = i
End Sub


Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    Call Command2_Click
End Sub


Private Sub GravaValor()
    Dim rs              As Integer
    Dim pIndice         As Long
    Dim Tick            As Long
    Dim Valor           As Long

'   Aquisita um valor da fila circular
    rs = DALx1.LeValorBuffer(pIndice, Tick, Valor)
    Select Case rs
        Case 0
        Case 1
            Exit Sub
        Case 2
    End Select

    Write #1, Str(pIndice), Str(Tick), Str(Valor)

    pIndice = pIndice + 4
    If pIndice >= 44000 Then pIndice = 2

End Sub


Private Sub Option1_Click()
    Option2.Value = Not Option1.Value
```

```
End Sub


Private Sub Option2_Click()
    Option1.Value = Not Option2.Value
End Sub
```

ANEXO B - EXEMPLO DE APLICAÇÃO EM LABVIEW

Este exemplo demonstra como é fácil criar uma aplicação no ambiente de programação LabView 6i Evaluation utilizando a biblioteca de aquisição de dados DALx.

A aplicação visa criar uma janela onde se pode visualizar a forma da onda da corrente elétrica residencial devidamente acondicionada para a captação pela placa de som.

Para esta aplicação utilizamos:

- Ambiente de LabView 6i Evaluation;
- Biblioteca de aquisição de dados (DALx.ocx);
- Circuito de acondicionamento de sinal (220Vac -> 1Vac, impedância 47k ohms);

É importante salientar que a versão Evaluation apresenta uma falha e não contém uma biblioteca para a utilização de componentes ActiveX (nicont.dll), impossibilitando a utilização da biblioteca de aquisição DALx. Para solucionar o problema é necessário acessar o site da National Instruments no endereço http://digital.ni.com/softlib.nsf/websearch/D5F11A64A905DB27862569EC005E31E2?opendocument&node=132070_US e instalar o pacote LabVIEW 6.0.2 Run-Time Engine sobrepondo a instalação da versão Evaluation.

INTERFACE

A interface do usuário possui os seguintes componentes:

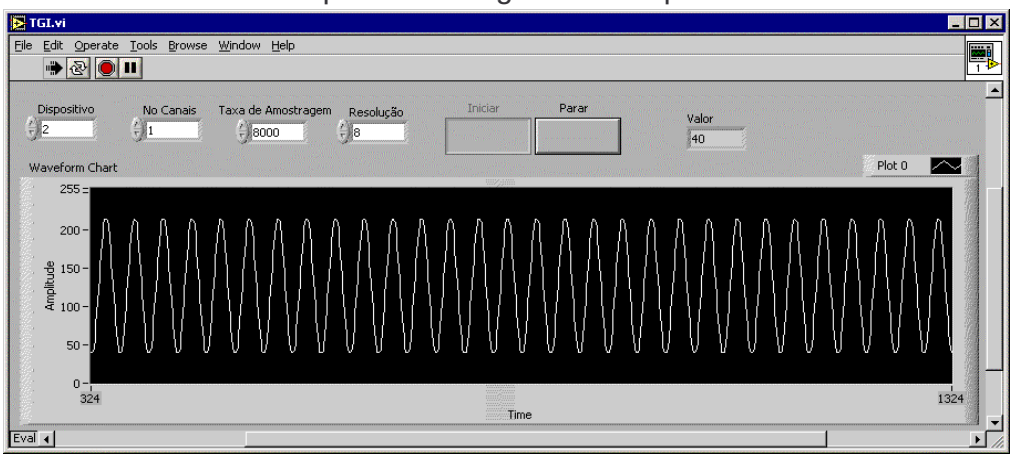


Figura B1 – IHM LabView

- 4 controles numéricos (digital control) para a configuração das propriedades "dispositivo", "no. de canais", "taxa de amostragem" e "resolução".
- 2 botões para iniciar e finalizar o processo de aquisição;
- 1 indicador numérico (digital indicator) para o acompanhamento dos valores coletados;
- 1 gráfico da forma da onda (Waveform chart) para a visualização da onda;

A utilização da IHM é bastante simples e consiste em configurar os parâmetros de entrada do dispositivo, iniciar e finalizar a aquisição.

DIAGRAMAS DO PROCESSO

Para tornarmos a aplicação didática, optamos por explicitar passo-a-passo as etapas que envolvem o processo de aquisição utilizando uma estrutura de seqüência (sequence structure).

O primeiro item da seqüência configura as propriedades de exibição dos botões "iniciar" e "parar". Quando a aplicação for iniciada, o botão parar deverá estar desabilitada. Na mesma seqüência também é feita a verificação se existe uma placa de som instalada no computador utilizando a função ExisteDispositivo do DALx. A resposta é exibida em uma janela de mensagem com o texto “Existe dispositivo” ou “Não existe dispositivo”.

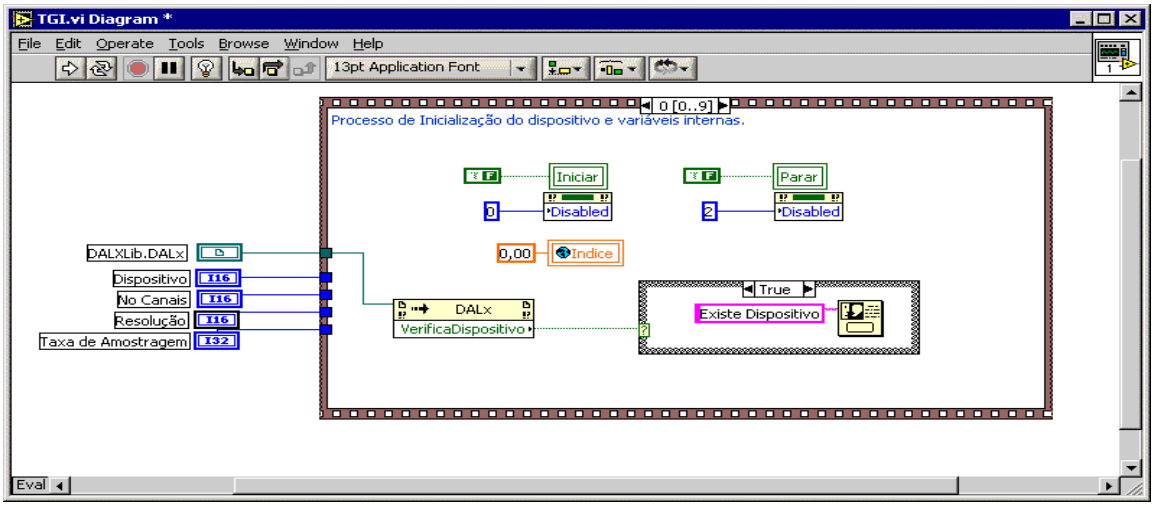


Figura B2 – Algoritmo LabView Parte I

O próximo item da seqüência fica aguardando o usuário clicar o botão "iniciar". Um espera de 150 milisegundos foi colocado dentro do loop para evitar que ele tome todo o tempo de processamento e consequentemente “trave” a aplicação.

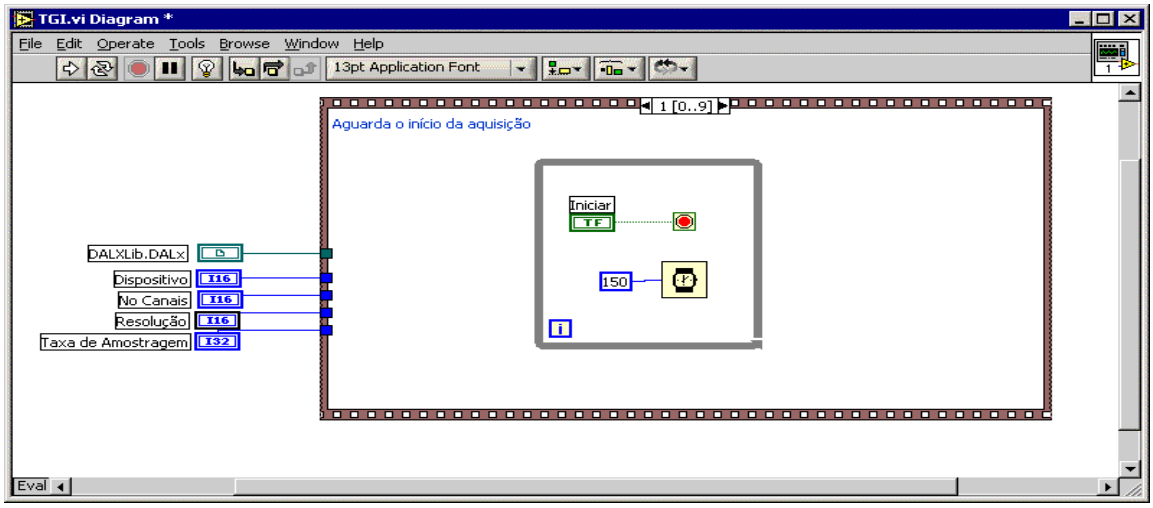


Figura B3 – Algoritmo LabView Parte II

Quando o usuário clicar o botão, a próxima seqüência é inicializada. Neste momento, os valores dos controles numéricos "dispositivo", "no. de canais", "resolução" e "taxa de amostragem" são armazenados na biblioteca e posteriormente é acionado a função HabilitaDispositivo para preparar o dispositivo para o processo de aquisição. Caso algum erro ocorra, é exibido uma mensagem "Erro HabilitaDispositivo". Ao mesmo tempo o botão "iniciar" é desabilitado para que somente o botão "parar" possa ser acionado.

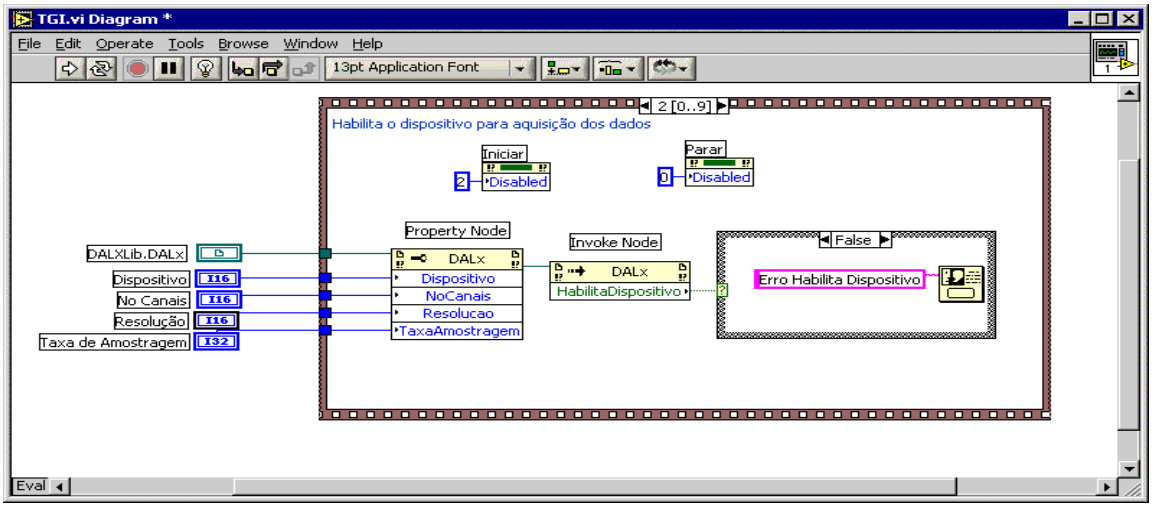


Figura B4 – Algoritmo LabView Parte III

Caso não tenha ocorrido nenhum erro no processo de habilitação do dispositivo, a aplicação aguarda 150 milisegundos para acomodação das funções de aquisição. No caso de aplicações criadas em LabView, este tempo de acomodação é de extrema importância pois sem ela o LabView gera um erro interno e finaliza a aplicação.

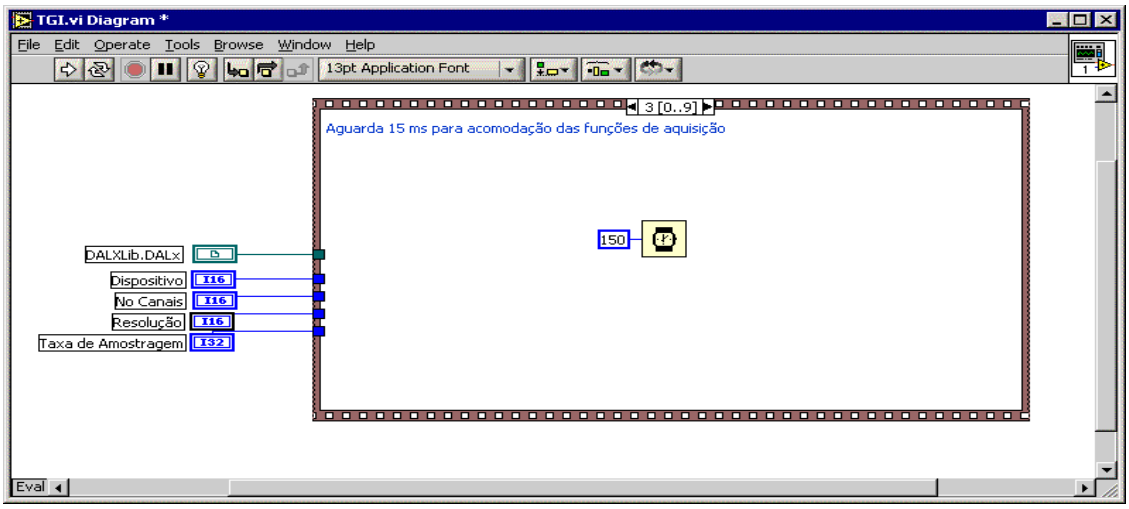


Figura B5 – Algoritmo LabView Parte IV

Após o tempo de acomodação, a aplicação inicia o processo de aquisição chamando a função IniciaAquisição.

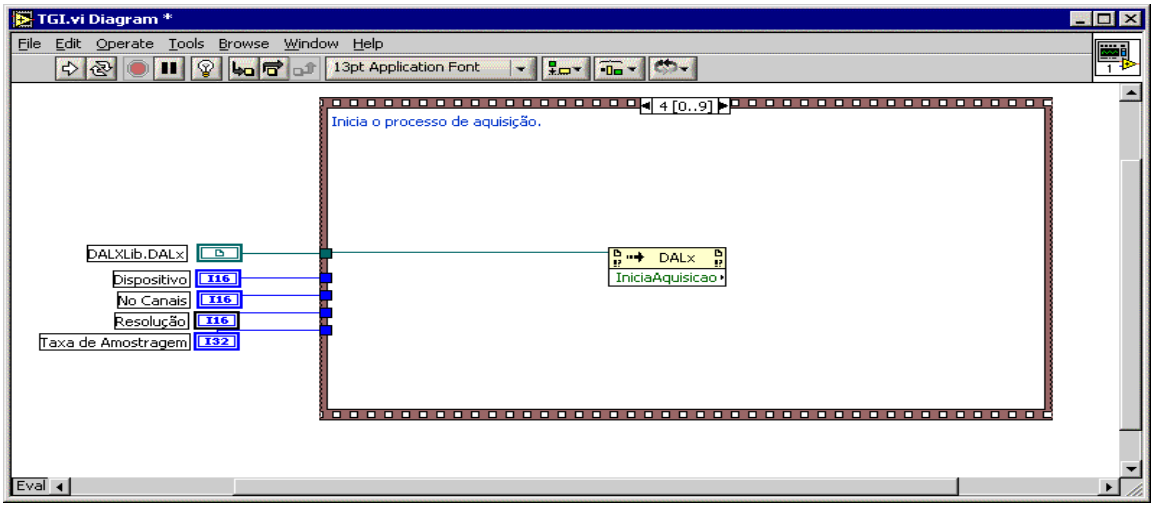


Figura B6 – Algoritmo LabView Parte V

Novamente a aplicação aguarda um tempo de acomodação das funções da biblioteca.

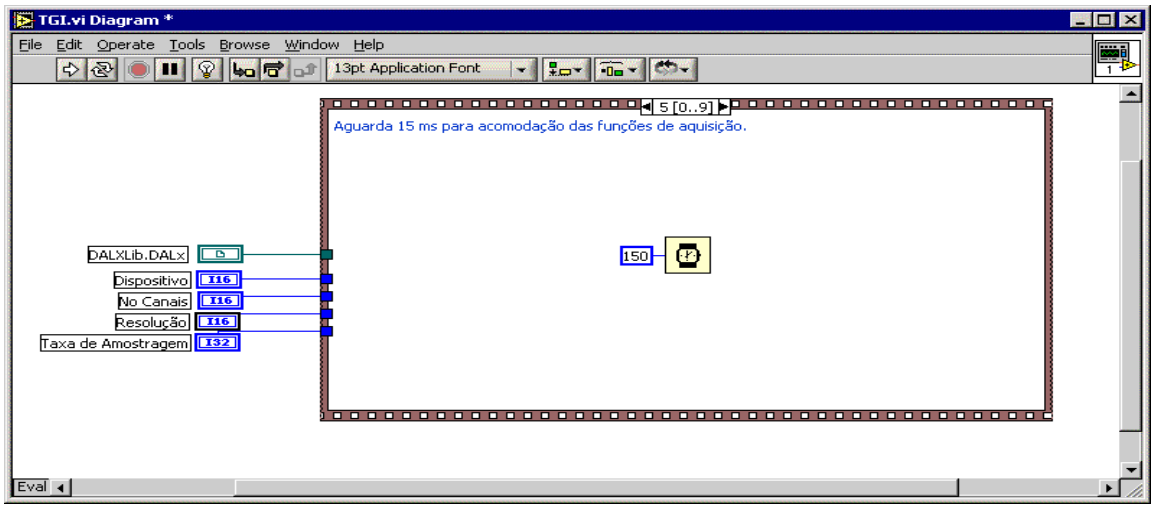


Figura B7 – Algoritmo LabView Parte VI

Após o tempo de acomodação, a aplicação utiliza uma variável global "indice" para controlar as amostras que serão coletadas da fila circular. Para efeito didático chamaremos de "IndiceFC" o último índice da fila circular e "Indice" o índice do dado que se deseja coletar.

Inicialmente o algoritmo obtêm o *IndiceFC* e compara ao *Indice*. Esta verificação é necessária para evitar que dados não coletados sejam consultados nos casos em que o processamento da aplicação é mais rápido do que a taxa de aquisição. Caso o *IndiceFC* seja maior que o *Indice* então o índice que se deseja consultar é passado à biblioteca e o valor armazenado na posição na fila circular é devolvida à aplicação. Este valor é repassado para o gráfico da forma da onda (Waveform chart) para a visualização gráfica do dado e é também exibido no indicador numérico (Digital indicator) . Então o próximo *Indice* é calculado e o algoritmo retorna ao início para a coleta do próximo dado. Este algoritmo e processado até que o usuário clique o botão "parar".

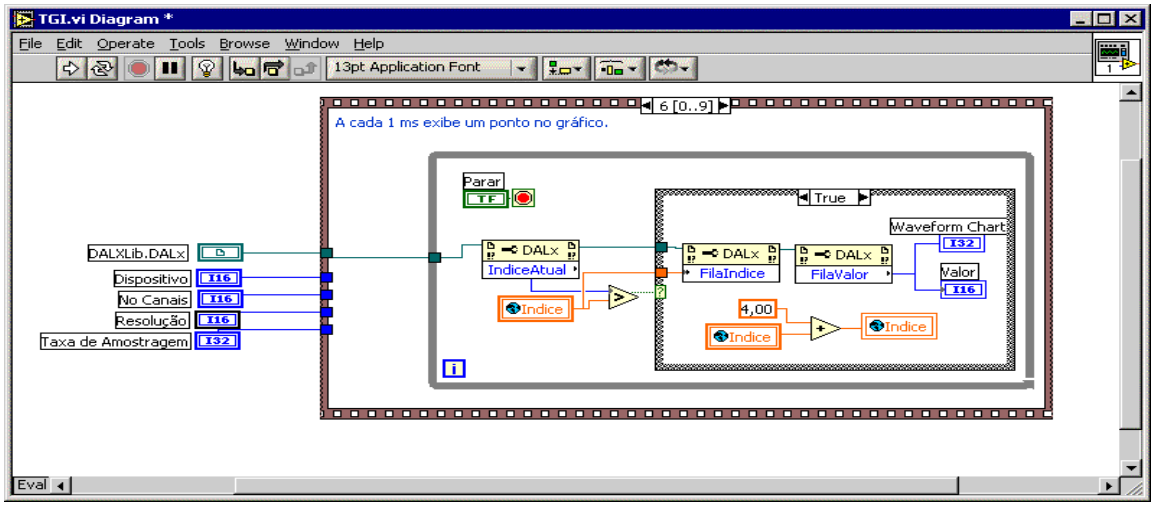


Figura B8 – Algoritmo LabView Parte VII

Quando o botão "parar" for acionado, a aplicação aciona a função TerminaAquisição para finalizar o processo de aquisição.

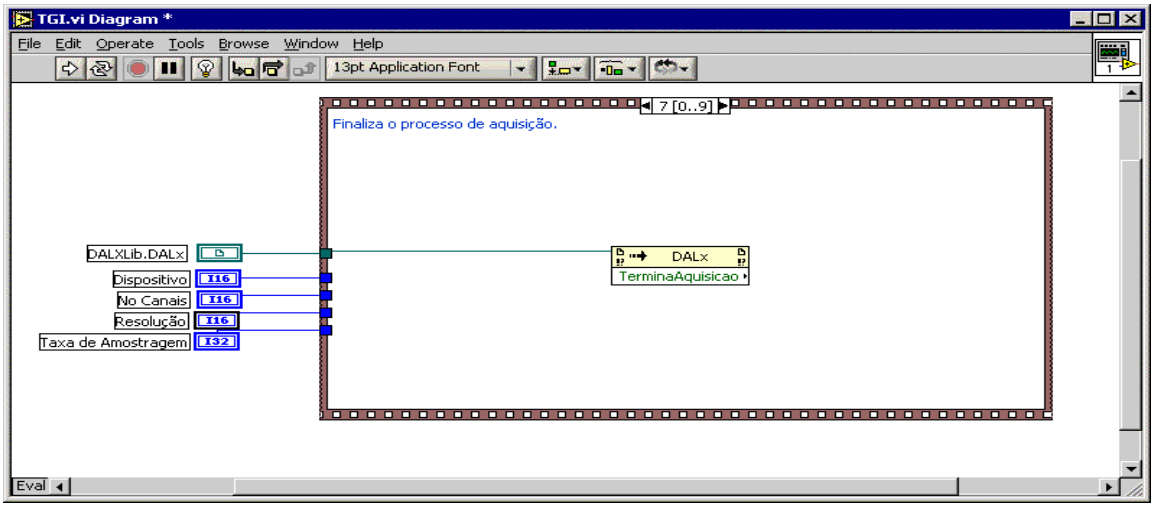


Figura B9 – Algoritmo LabView Parte VIII

Após um tempo de 150 milisegundos de acomodação, a aplicação aciona a função DesabilitaDispositivo encerrando a aplicação.

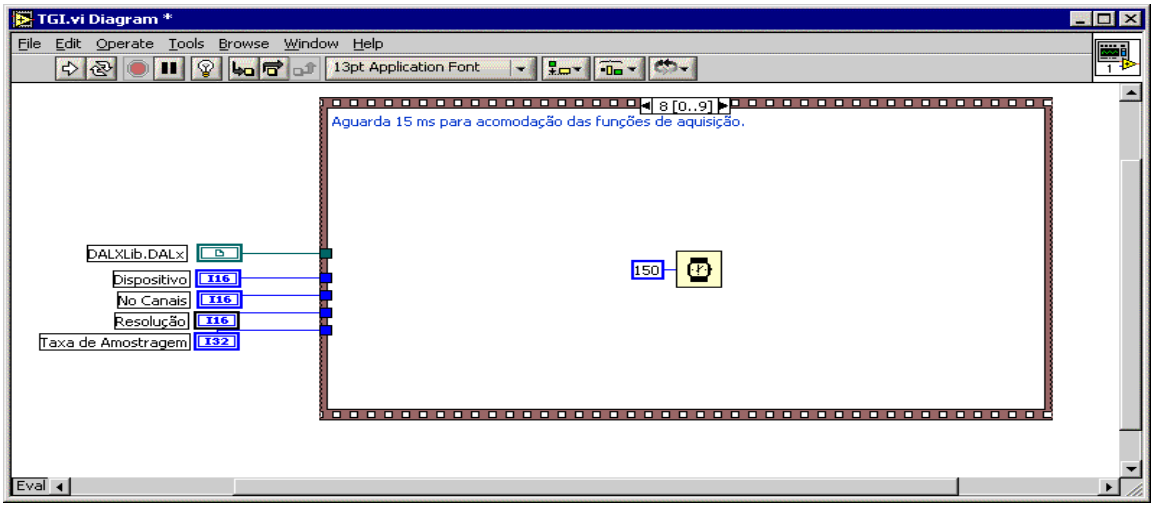


Figura B10 – Algoritmo LabView Parte IX

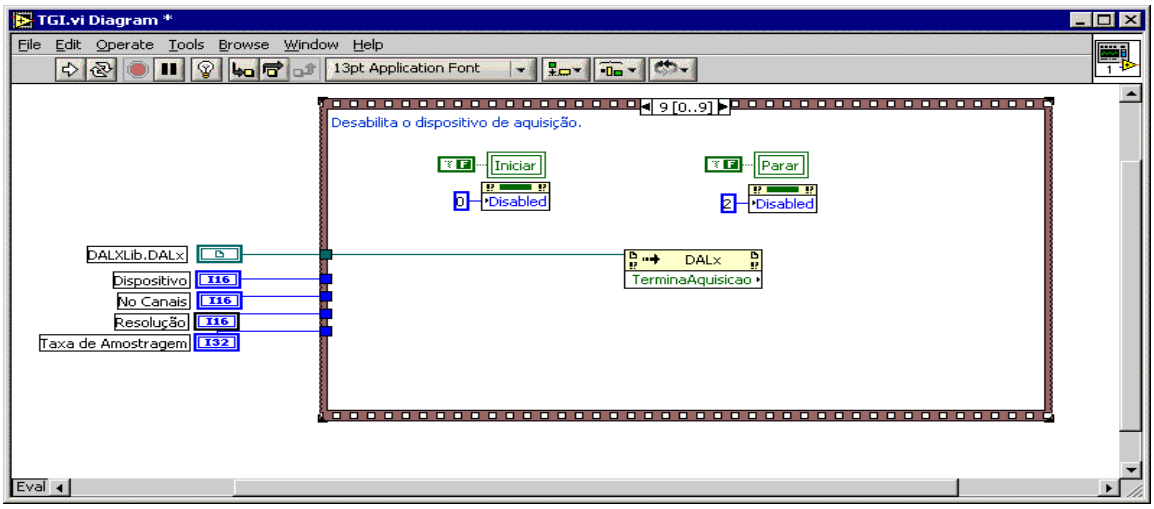


Figura B11 – Algoritmo LabView Parte X

ANEXO C - EXEMPLO DE APLICAÇÃO EM MATLAB

Esta aplicação criada em ambiente MatLab 6.0 R12 tem como objetivo demonstrar a utilização da biblioteca de aquisição DALx.

A função tgi.m recebe como parâmetros a quantidade de amostras, a identificação do dispositivo, no. de canais, taxa de amostragem, resolução e o tamanho do *buffer* de aquisição e retorna um vetor com os valores coletados. A chamada da função é:

vetor = tgi (*quantidade amostras*, *identificação dispositivo*, *no. canais*, *taxa amostragem*, *resolução*, *tamanho buffer*)

Um exemplo da chamada da função e o seu resultado podem ser observados a seguir:

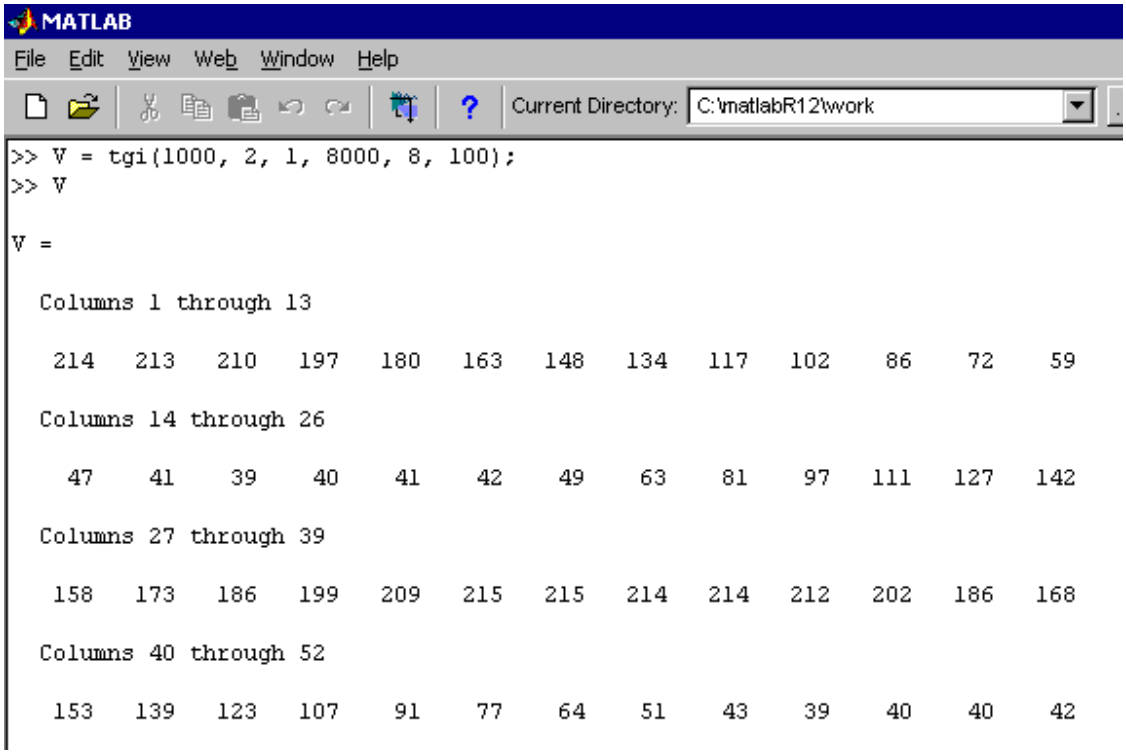


Figura C1 – MatLab Valores Coletados

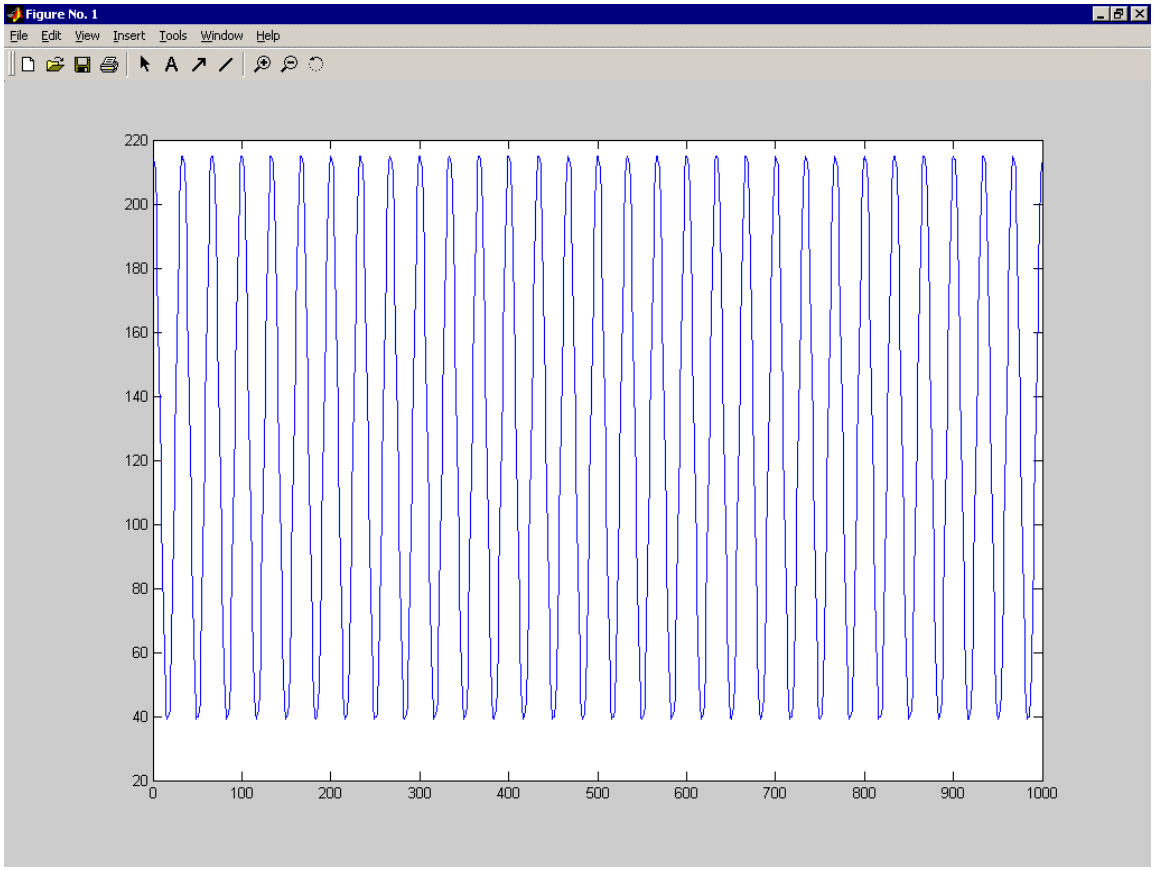


Figura C2 – Gráfico MatLab

FUNÇÃO TGI.M

O algoritmo que gera o resultado acima é extremamente simples e pode ser resumido em algumas etapas:

- Instanciação do objeto DALx.
- Verificação da existência da placa de som no computador.
- Configuração dos parâmetros do dispositivo.
- Preparação do dispositivo.
- Início do processo de aquisição.

- Tratamento dos dados coletados.
- Término do processo de aquisição.
- Liberação do dispositivo.
- Liberação do objeto DALx.

Visualização gráfica dos valores coletados.

O código fonte traz o detalhamento das etapas descritas acima.

Código fonte tgi.m

```
%*============================================================
==
%*
%*     Universidade Braz Cubas
%*     Engenharia da Computação
%*     Marcio Murasugi
%*
%*     Exemplo de aplicaçao em MatLab 6.0
%*     TGI.m
%*     Version 1.00
%*     Fevereiro/2003 .
%*
%*============================================================
=


function [V] =  TGI(pAmostras, pDispositivo, pNoCanais, pTaxaAmostragem,
pResolucao, pTamanhoBuffer)

    % Instancia o objeto DALx
    DALx = actxcontrol('DALx.DALxCtrl.1');

    % Verifica a existencia da placa de som
    if DALx.VerificaDispositivo ~=  -1 end

    % Configura os parametros do dispostivo
    DALx.Dispositivo = pDispositivo;
    DALx.NoCanais = pNoCanais;
    DALx.TaxaAmostragem = pTaxaAmostragem;
    DALx.Resolucao = pResolucao;
    DALx.TamanhoBuffer = pTamanhoBuffer;
```

```
% Habilita a placa de som para aquisicao
if DALx.HabilitaDispositivo ~=  -1
   quit
end

pause(1);

% Incia o processo de aquisicao
DALx.IniciaAquisicao;

% Coleta os dados
i = 0;
j = 1;
while j < pAmostras
   if DALx.IndiceAtual > i
      DALx.FilaIndice = i;
      V(j)  = DALx.FilaValor;
      i = i + 4;
      j = j + 1;
   end
end

% Finaliza o processo de aquisicao
DALx.TerminaAquisicao;

% Desabilita a placa de som para aquisicao
DALx.DesabilitaDispositivo;

delete(DALx);

plot(V);
```

ANEXO D - FUNÇÕES DE CONTROLE DE ÁUDIO DO WINDOWS

As especificações das funções e estruturas de manipulação de áudio do Windows utilizadas no desenvolvimento do projeto foram extraídas integralmente do MSDN (Microsoft Development Network) e podem ser acessadas através do site http://www.msdn.microsoft.com/

ABOUT WAVEFORM AUDIO

Adding sound to your application can make it more efficient and more fun to use. You can improve your users' efficiency by using sounds to get their attention at critical points, to help them avoid mistakes, or to let them know that a time-consuming operation has finished. You can help them have more fun by adding music or sound effects.

There are several methods for adding sound to your application using waveform audio. The simplest method documented here is that of using the PlaySound function. Most of the other waveform-audio API elements are relatively low-level. The MCI overview documents an interface to multimedia programming that offers method of adding sound to your application that is simpler and faster than using the waveform-audio sound API.

For more information, see the following topics:

- The PlaySound Function
- Waveform-Audio Interface
- Auxiliary Audio Interface
- Audio Data Blocks
- Handling Errors with Audio Functions

The PlaySound Function

You can use the PlaySound function to play waveform audio, as long as the sound fits into available memory. (The sndPlaySound function offers a subset of the capabilities of PlaySound. To maximize the portability of your Win32-based application, use PlaySound, not sndPlaySound.)

The PlaySound function allows you to specify a sound in one of three ways:

- As a system alert, using the alias stored in the WIN.INI file or the registry
- As a filename
- As a resource identifier

The PlaySound function allows you to play a sound in a continuous loop, ending only when you call PlaySound again, specifying either NULL or the sound identifier of another sound for the pszSound parameter.

You can use PlaySound to play the sound synchronously or asynchronously, and to control the behavior of the function in other ways when it must share system resources.

For examples of how to use PlaySound in your Win32-based applications, see Playing WAVE Resources.

Auxiliary Audio Interface

Auxiliary audio devices are audio devices whose output is mixed with the MIDI and waveform-audio output devices in a multimedia computer. An example of an auxiliary audio device is the CD audio output from a CD-ROM drive.

- Querying Auxiliary Audio Devices
- Changing the Volume of Auxiliary Audio-Devices

Querying Auxiliary Audio Devices

Not all multimedia systems have auxiliary audio support. You can use the auxGetNumDevs function to determine the number of controllable auxiliary devices present in a system.

To get information about a particular auxiliary audio device, use the auxGetDevCaps function. This function fills an AUXCAPS structure with information about the capabilities of a specified device. This information includes the manufacturer and product identifiers, a product name for the device, and the device-driver version number.

Changing the Volume of Auxiliary Audio-Devices

Windows provides the following functions to query and set the volume for auxiliary audio devices.

Function	Description
auxGetVolume	Retrieves the current volume setting of the specified auxiliary output device.
auxSetVolume	Sets the volume of the specified auxiliary output device.

Not all auxiliary audio devices support volume changes. Some devices can support individual volume changes on both the left and the right channels.

Volume is specified in a doubleword value, as with the waveform-audio and MIDI volume-control functions. When the audio format is stereo, the upper 16 bits specify the relative volume of the right channel and the lower 16 bits specify the relative volume of the left channel. For devices that do not support left- and right-channel volume control, the lower 16 bits specify the volume level, and the upper 16 bits are ignored.

Volume-level values range from 0x0 (silence) to 0xFFFF (maximum volume) and are interpreted logarithmically. The perceived volume increase is the same when increasing the volume level from 0x5000 to 0x6000 as it is from 0x4000 to 0x5000.

Audio Data Blocks

The waveInAddBuffer and waveOutWrite functions require applications to allocate data blocks to pass to the device drivers for recording or playback purposes. Both of these functions use the WAVEHDR structure to describe its data block.

Before using one of these functions to pass a data block to a device driver, you must allocate memory for the data block and the header structure that describes the data block. The headers can be prepared and unprepared by using the following functions.

Function	Description
waveInPrepareHeader	Prepares a waveform-audio input data block.
waveInUnprepareHeader	Cleans up the preparation on a waveform-audio input data block.
waveOutPrepareHeader	Prepares a waveform-audio output data block.
waveOutUnprepareHeader	Cleans up the preparation on a waveform-audio output data block.

Before you pass an audio data block to a device driver, you must prepare the data block by passing it to either waveInPrepareHeader or waveOutPrepareHeader. When the device driver is finished with the data block and returns it, you must clean up this preparation by passing the data block to either waveInUnprepareHeader or waveOutUnprepareHeader before any allocated memory can be freed.

Unless the waveform-audio input and output data is small enough to be contained in a single data block, applications must continually supply the device driver with data blocks until playback or recording is complete.

Even if a single data block is used, an application must be able to determine when a device driver is finished with the data block so the application can free the memory associated with the data block and header structure. There are several ways to determine when a device driver is finished with a data block:

- By specifying a callback function to receive a message sent by the driver when it is finished with a data block
- By using an event callback
- By specifying a window or thread to receive a message sent by the driver when it is finished with a data block
- By polling the WHDR_DONE bit in the dwFlags member of the WAVEHDR structure sent with each data block

If an application does not get a data block to the device driver when needed, there can be an audible gap in playback or a loss of incoming recorded information. This requires at least a double-buffering scheme — staying at least one data block ahead of the device driver.

The following topics describe ways to determine when a device driver is finished with a data block:

- Using a callback function to process driver messages
- Using an event callback to process driver messages
- Using a window or thread to process driver messages
- Managing data blocks by polling

Using a Callback Function to Process Driver Messages

You can write your own callback function to process messages sent by the device driver. To use a callback function, specify the CALLBACK_FUNCTION flag in the fdwOpen parameter and the address of the callback in the dwCallback parameter of the waveInOpen or waveOutOpen function.

Messages sent to a callback function are similar to messages sent to a window, except they have two DWORD parameters instead of a UINT and a DWORD parameter. For details on these messages, see Playing Waveform-Audio Files.

To pass instance data from an application to a callback function, use one of the following techniques:

- Pass the instance data using the dwInstance parameter of the function that opens the device driver.
- Pass the instance data using the dwUser member of the WAVEHDR structure that identifies an audio data block being sent to a device driver.

If you need more than 32 bits of instance data, pass a pointer to a structure containing the additional information.

Using an Event Callback to Process Driver Messages

To use an event callback, use the CreateEvent function to retrieve the handle of an event. In the call to the waveOutOpen function, specify CALLBACK_EVENT for the

fdwOpen parameter. After calling the waveOutPrepareHeader function but before sending waveform-audio data to the device, create a nonsignaled event by calling the ResetEvent function, specifying the event handle retrieved by CreateEvent. Then, inside a loop that checks whether the WHDR_DONE bit is set in the dwFlags member of the WAVEHDR structure, call the WaitForSingleObject function, specifying as parameters the event handle and a time-out value of INFINITE.

Because event callbacks do not receive specific close, done, or open notifications, an application might have to check the status of the process it is waiting for after the event occurs. It is possible that a number of tasks could have been completed by the time WaitForSingleObject returns.

Using a Window or Thread to Process Driver Messages

To use a window callback function, specify the CALLBACK_WINDOW flag in the fdwOpen parameter and a window handle in the low-order word of the dwCallback parameter of the waveInOpen or waveOutOpen function. Driver messages will be sent to the window procedure for the window identified by the handle in dwCallback.

Similarly, to use a thread callback, specify CALLBACK_THREAD and a thread handle in the call to waveInOpen or waveOutOpen. In this case, messages are posted to the specified thread instead of to a window.

Messages sent to the window or thread callback are specific to the audio device type used. For more information about these messages, see Playing Waveform-Audio Files.

Managing Data Blocks by Polling

In addition to using a callback function, you can poll the dwFlags member of a WAVEHDR structure to determine when an audio device is finished with a data block. Sometimes it is better to poll dwFlags than to wait for another mechanism to receive messages from the drivers. For example, after you call the waveOutReset function to release pending data blocks, you can immediately poll to be sure that the data blocks have been released before calling waveOutUnprepareHeader and freeing the memory for the data block.

You can use the WHDR_DONE flag to test the dwFlags member. As soon as the WHDR_DONE flag is set in the dwFlags member of the WAVEHDR structure, the driver is finished with the data block.

Handling Errors with Audio Functions

The waveform-audio and auxiliary-audio functions return a nonzero value when an error occurs. Windows provides functions that convert these error values into textual descriptions of the errors. The application must still examine the error values to determine how to proceed, but textual descriptions of errors can be used in dialog boxes that describe errors to users.

You can use the following functions to retrieve textual descriptions of audio error values:

Function	Description
waveInGetErrorText	Retrieves a textual description of a specified waveform-audio input error.
waveOutGetErrorText	Retrieves a textual description of a specified waveform-audio output error.

The only audio functions that do not return error values are auxGetNumDevs, waveInGetNumDevs, and waveOutGetNumDevs. These functions return zero if no devices are present in a system or if they encounter any errors.

waveInOpen

The **waveInOpen** function opens the given waveform-audio input device for recording.

MMRESULT waveInOpen(

LPHWAVEIN	phwi,
UINT_PTR	uDeviceID,
LPWAVEFORMATEX	pwfx,
DWORD_PTR	dwCallback,

DWORD_PTR dwCallbackInstance,

DWORD fdwOpen

);

Parameters

phwi

Pointer to a buffer that receives a handle identifying the open waveform-audio input device. Use this handle to identify the device when calling other waveform-audio input functions. This parameter can be NULL if WAVE_FORMAT_QUERY is specified for fdwOpen.

uDeviceID

Identifier of the waveform-audio input device to open. It can be either a device identifier or a handle of an open waveform-audio input device. You can use the following flag instead of a device identifier.

Value	*Meaning*
WAVE_MAPPER	The function selects a waveform-audio input device capable of recording in the specified format.

pwfx

Pointer to a WAVEFORMATEX structure that identifies the desired format for recording waveform-audio data. You can free this structure immediately after waveInOpen returns.

dwCallback

Pointer to a fixed callback function, an event handle, a handle to a window, or the identifier of a thread to be called during waveform-audio recording to process messages related to the progress of recording. If no callback function is required, this value can be zero. For more information on the callback function, see waveInProc.

dwCallbackInstance

User-instance data passed to the callback mechanism. This parameter is not used with the window callback mechanism.

fdwOpen

Flags for opening the device. The following values are defined.

Value	Meaning
CALLBACK_EVENT	The dwCallback parameter is an event handle.
CALLBACK_FUNCTION	The dwCallback parameter is a callback procedure address.
CALLBACK_NULL	No callback mechanism. This is the default setting.
CALLBACK_THREAD	The dwCallback parameter is a thread identifier.
CALLBACK_WINDOW	The dwCallback parameter is a window handle.
WAVE_FORMAT_DIRECT	If this flag is specified, the ACM driver does not perform conversions on the audio data.
WAVE_FORMAT_QUERY	The function queries the device to determine whether it supports the given format, but it does not open the device.
WAVE_MAPPED	The uDeviceID parameter specifies a waveform-audio device to be mapped to by the wave mapper.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_ALLOCATED	Specified resource is already allocated.
MMSYSERR_BADDEVICEID	Specified device identifier is out of range.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

WAVERR_BADFORMAT	Attempted to open with an unsupported waveform-audio format.

Remarks

Use the waveInGetNumDevs function to determine the number of waveform-audio input devices present on the system. The device identifier specified by uDeviceID varies from zero to one less than the number of devices present. The WAVE_MAPPER constant can also be used as a device identifier.

If you choose to have a window or thread receive callback information, the following messages are sent to the window procedure or thread to indicate the progress of waveform-audio input: MM_WIM_OPEN, MM_WIM_CLOSE, and MM_WIM_DATA.

If you choose to have a function receive callback information, the following messages are sent to the function to indicate the progress of waveform-audio input: WIM_OPEN, WIM_CLOSE, and WIM_DATA.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions, WAVEFORMATEX, waveInGetNumDevs, waveInProc, MM_WIM_OPEN, MM_WIM_CLOSE, MM_WIM_DATA

waveInClose

The waveInClose function closes the given waveform-audio input device.

MMRESULT waveInClose(

HWAVEIN hwi

);

Parameters

hwi

Handle to the waveform-audio input device. If the function succeeds, the handle is no longer valid after this call.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.
WAVERR_STILLPLAYING	There are still buffers in the queue.

Remarks

If there are input buffers that have been sent with the waveInAddBuffer function and that haven't been returned to the application, the close operation will fail. Call the waveInReset function to mark all pending buffers as done.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions, waveInAddBuffer, waveInReset

waveInReset

The waveInReset function stops input on the given waveform-audio input device and resets the current position to zero. All pending buffers are marked as done and returned to the application.

MMRESULT waveInReset(

HWAVEIN hwi

);

Parameters

hwi

Handle to the waveform-audio input device.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions

waveInPrepareHeader

The waveInPrepareHeader function prepares a buffer for waveform-audio input.

MMRESULT waveInPrepareHeader(

HWAVEIN	hwi,
LPWAVEHDR	pwh,
UINT	cbwh

);

Parameters

hwi

Handle to the waveform-audio input device.

pwh

Pointer to a WAVEHDR structure that identifies the buffer to be prepared.

cbwh

Size, in bytes, of the WAVEHDR structure.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

Remarks

The lpData, dwBufferLength, and dwFlags members of the WAVEHDR structure must be set before calling this function (dwFlags must be zero).

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions

WAVEINUNPREPAREHEADER

The waveInUnprepareHeader function cleans up the preparation performed by the waveInPrepareHeader function. This function must be called after the device driver fills a buffer and returns it to the application. You must call this function before freeing the buffer.

MMRESULT waveInUnprepareHeader(

HWAVEIN hwi,

LPWAVEHDR pwh,

UINT cbwh

);

Parameters

hwi

Handle to the waveform-audio input device.

pwh

Pointer to a WAVEHDR structure identifying the buffer to be cleaned up.

cbwh

Size, in bytes, of the WAVEHDR structure.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.
WAVERR_STILLPLAYING	The buffer pointed to by the pwh parameter is still in the queue.

Remarks

This function complements the waveInPrepareHeader function.

You must call this function before freeing the buffer. After passing a buffer to the device driver with the waveInAddBuffer function, you must wait until the driver is finished with the buffer before calling waveInUnprepareHeader. Unpreparing a buffer that has not been prepared has no effect, and the function returns zero.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions, waveInPrepareHeader, WAVEHDR, waveInAddBuffer

WAVEINADDBUFFER

The waveInAddBuffer function sends an input buffer to the given waveform-audio input device. When the buffer is filled, the application is notified.

MMRESULT waveInAddBuffer(

HWAVEIN	hwi,
LPWAVEHDR	pwh,
UINT	cbwh

);

Parameters

hwi

Handle to the waveform-audio input device.

pwh

Pointer to a WAVEHDR structure that identifies the buffer.

cbwh

Size, in bytes, of the WAVEHDR structure.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.
WAVERR_UNPREPARED	The buffer pointed to by the pwh parameter hasn't been prepared.

Remarks

When the buffer is filled, the WHDR_DONE bit is set in the dwFlags member of the WAVEHDR structure.

The buffer must be prepared with the waveInPrepareHeader function before it is passed to this function.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions, WAVEHDR, waveInPrepareHeader

WAVEINSTART

The waveInStart function starts input on the given waveform-audio input device.

MMRESULT waveInStart(

HWAVEIN hwi

);

Parameters

hwi

Handle to the waveform-audio input device.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

Remarks

Buffers are returned to the application when full or when the waveInReset function is called (the dwBytesRecorded member in the header will contain the length of data). If there are no buffers in the queue, the data is thrown away without notifying the application, and input continues.

Calling this function when input is already started has no effect, and the function returns zero.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions, waveInReset

WAVEINSTOP

The waveInStop function stops waveform-audio input.

MMRESULT waveInStop(

HWAVEIN hwi

);

Parameters

hwi

Handle to the waveform-audio input device.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_INVALHANDLE	Specified device handle is invalid.
MMSYSERR_NODRIVER	No device driver is present.

MMSYSERR_NOMEM	Unable to allocate or lock memory.

Remarks

If there are any buffers in the queue, the current buffer will be marked as done (the dwBytesRecorded member in the header will contain the length of data), but any empty buffers in the queue will remain there.

Calling this function when input is not started has no effect, and the function returns zero.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions

WAVEINGETERRORTEXT

The waveInGetErrorText function retrieves a textual description of the error identified by the given error number.

MMRESULT waveInGetErrorText(

MMRESULT mmrError,

LPSTR pszText,

UINT cchText

);

Parameters

mmrError

Error number.

pszText

Pointer to the buffer to be filled with the textual error description.

cchText

Size, in characters, of the buffer pointed to by pszText.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_BADERRNUM	Specified error number is out of range.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

Remarks

If the textual error description is longer than the specified buffer, the description is truncated. The returned error string is always null-terminated. If cchText is zero, nothing is copied and the function returns zero. All error descriptions are less than MAXERRORLENGTH characters long.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000/XP.

See Also

Waveform Audio Overview, Waveform Functions

WAVEINGETDEVCAPS

The waveInGetDevCaps function retrieves the capabilities of a given waveform-audio input device.

MMRESULT waveInGetDevCaps(

UINT_PTR	uDeviceID,
LPWAVEINCAPS	pwic,
UINT	cbwic

);

Parameters

uDeviceID

Identifier of the waveform-audio output device. It can be either a device identifier or a handle of an open waveform-audio input device.

pwic

Pointer to a WAVEINCAPS structure to be filled with information about the capabilities of the device.

cbwic

Size, in bytes, of the WAVEINCAPS structure.

Return Values

Returns MMSYSERR_NOERROR if successful or an error otherwise. Possible error values include the following.

Value	Description
MMSYSERR_BADDEVICEID	Specified device identifier is out of range.
MMSYSERR_NODRIVER	No device driver is present.
MMSYSERR_NOMEM	Unable to allocate or lock memory.

Remarks

Use this function to determine the number of waveform-audio input devices present in the system. If the value specified by the uDeviceID parameter is a device identifier, it can vary from zero to one less than the number of devices present. The WAVE_MAPPER constant can also be used as a device identifier. Only cbwic bytes (or less) of information is copied to the location pointed to by pwic. If cbwic is zero, nothing is copied and the function returns zero.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000/XP.

See Also

Waveform Audio Overview, Waveform Functions, WAVEINCAPS

WAVEINGETNUMDEVS

The waveInGetNumDevs function returns the number of waveform-audio input devices present in the system.

UINT waveInGetNumDevs(VOID);

Parameters

This function takes no parameters.

Return Values

Returns the number of devices. A return value of zero means that no devices are present or that an error occurred.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Library: Use Winmm.lib.

See Also

Waveform Audio Overview, Waveform Functions

WAVEFORMATEX

The WAVEFORMATEX structure defines the format of waveform-audio data. Only format information common to all waveform-audio data formats is included in this structure. For formats that require additional information, this structure is included as the first member in another structure, along with the additional information.

typedef struct {

    WORD wFormatTag;

```
    WORD  nChannels;
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD  nBlockAlign;
    WORD  wBitsPerSample;
    WORD  cbSize;
} WAVEFORMATEX;
```

Members

wFormatTag

Waveform-audio format type. Format tags are registered with Microsoft Corporation for many compression algorithms. A complete list of format tags can be found in the MMREG.H header file.

nChannels

Number of channels in the waveform-audio data. Monaural data uses one channel and stereo data uses two channels.

nSamplesPerSec

Sample rate, in samples per second (hertz), that each channel should be played or recorded. If wFormatTag is WAVE_FORMAT_PCM, then common values for nSamplesPerSec are 8.0 kHz, 11.025 kHz, 22.05 kHz, and 44.1 kHz. For non-PCM formats, this member must be computed according to the manufacturer's specification of the format tag.

nAvgBytesPerSec

Required average data-transfer rate, in bytes per second, for the format tag. If wFormatTag is WAVE_FORMAT_PCM, nAvgBytesPerSec should be equal to the product of nSamplesPerSec and nBlockAlign. For non-PCM formats, this member must be computed according to the manufacturer's specification of the format tag.

Playback and record software can estimate buffer sizes by using the nAvgBytesPerSec member.

nBlockAlign

Block alignment, in bytes. The block alignment is the minimum atomic unit of data for the wFormatTag format type. If wFormatTag is WAVE_FORMAT_PCM, nBlockAlign should be equal to the product of nChannels and wBitsPerSample divided by 8 (bits per byte). For non-PCM formats, this member must be computed according to the manufacturer's specification of the format tag.

Playback and record software must process a multiple of nBlockAlign bytes of data at a time. Data written and read from a device must always start at the beginning of a block. For example, it is illegal to start playback of PCM data in the middle of a sample (that is, on a non-block-aligned boundary).

wBitsPerSample

Bits per sample for the wFormatTag format type. If wFormatTag is WAVE_FORMAT_PCM, then wBitsPerSample should be equal to 8 or 16. For non-PCM formats, this member must be set according to the manufacturer's specification of the format tag. Note that some compression schemes cannot define a value for wBitsPerSample, so this member can be zero.

cbSize

Size, in bytes, of extra format information appended to the end of the WAVEFORMATEX structure. This information can be used by non-PCM formats to store extra attributes for the wFormatTag. If no extra information is required by the wFormatTag, this member must be set to zero. Note that for WAVE_FORMAT_PCM formats (and only WAVE_FORMAT_PCM formats), this member is ignored.

Remarks

An example of a format that uses extra information is the Microsoft Adaptive Delta Pulse Code Modulation (MS-ADPCM) format. The wFormatTag for MS-ADPCM is WAVE_FORMAT_ADPCM. The cbSize member will typically be set to 32. The extra information stored for WAVE_FORMAT_ADPCM is coefficient pairs required for encoding and decoding the waveform-audio data.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmreg.h.

See Also

Waveform Audio Overview, Waveform Structures

WAVEHDR

The WAVEHDR structure defines the header used to identify a waveform-audio buffer.

```
typedef struct {
    LPSTR       lpData;
    DWORD       dwBufferLength;
    DWORD       dwBytesRecorded;
    DWORD_PTR   dwUser;
    DWORD       dwFlags;
    DWORD       dwLoops;
    struct wavehdr_tag * lpNext;
    DWORD_PTR   reserved;
} WAVEHDR;
```

Members

lpData

Pointer to the waveform buffer.

dwBufferLength

Length, in bytes, of the buffer.

dwBytesRecorded

When the header is used in input, this member specifies how much data is in the buffer.

dwUser

User data.

dwFlags

Flags supplying information about the buffer. The following values are defined:

WHDR_BEGINLOOP

This buffer is the first buffer in a loop. This flag is used only with output buffers.

WHDR_DONE

Set by the device driver to indicate that it is finished with the buffer and is returning it to the application.

WHDR_ENDLOOP

This buffer is the last buffer in a loop. This flag is used only with output buffers.

WHDR_INQUEUE

Set by Windows to indicate that the buffer is queued for playback.

WHDR_PREPARED

Set by Windows to indicate that the buffer has been prepared with the waveInPrepareHeader or waveOutPrepareHeader function.

dwLoops

Number of times to play the loop. This member is used only with output buffers.

wavehdr_tag

Reserved.

reserved

Reserved.

Remarks

Use the WHDR_BEGINLOOP and WHDR_ENDLOOP flags in the dwFlags member to specify the beginning and ending data blocks for looping. To loop on a single block, specify both flags for the same block. Use the dwLoops member in the WAVEHDR structure for the first block in the loop to specify the number of times to play the loop.

The lpData, dwBufferLength, and dwFlags members must be set before calling the waveInPrepareHeader or waveOutPrepareHeader function. (For either function, the dwFlags member must be set to zero.)

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

See Also

Waveform Audio Overview, Waveform Structures, waveInPrepareHeader, waveOutPrepareHeader

WAVEINCAPS

The WAVEINCAPS structure describes the capabilities of a waveform-audio input device.

typedef struct {

```
    WORD      wMid;
    WORD      wPid;
    MMVERSION vDriverVersion;
    TCHAR     szPname[MAXPNAMELEN];
    DWORD     dwFormats;
    WORD      wChannels;
    WORD      wReserved1;
} WAVEINCAPS;
```

Members

wMid

Manufacturer identifier for the device driver for the waveform-audio input device. Manufacturer identifiers are defined in Manufacturer and Product Identifiers.

wPid

Product identifier for the waveform-audio input device. Product identifiers are defined in Manufacturer and Product Identifiers.

vDriverVersion

Version number of the device driver for the waveform-audio input device. The high-order byte is the major version number, and the low-order byte is the minor version number.

szPname

Product name in a null-terminated string.

dwFormats

Standard formats that are supported. Can be a combination of the following:

Format	Description
WAVE_FORMAT_1M08	11.025 kHz, mono, 8-bit
WAVE_FORMAT_1M16	11.025 kHz, mono, 16-bit
WAVE_FORMAT_1S08	11.025 kHz, stereo, 8-bit
WAVE_FORMAT_1S16	11.025 kHz, stereo, 16-bit
WAVE_FORMAT_2M08	22.05 kHz, mono, 8-bit
WAVE_FORMAT_2M16	22.05 kHz, mono, 16-bit
WAVE_FORMAT_2S08	22.05 kHz, stereo, 8-bit
WAVE_FORMAT_2S16	22.05 kHz, stereo, 16-bit
WAVE_FORMAT_4M08	44.1 kHz, mono, 8-bit
WAVE_FORMAT_4M16	44.1 kHz, mono, 16-bit
WAVE_FORMAT_4S08	44.1 kHz, stereo, 8-bit
WAVE_FORMAT_4S16	44.1 kHz, stereo, 16-bit

wChannels

Number specifying whether the device supports mono (1) or stereo (2) input.

wReserved1

Padding.

Requirements

Windows NT/2000/XP: Included in Windows NT 3.1 and later.

Windows 95/98/Me: Included in Windows 95 and later.

Header: Declared in Mmsystem.h; include Windows.h.

Unicode: Declared as Unicode and ANSI structures.

See Also

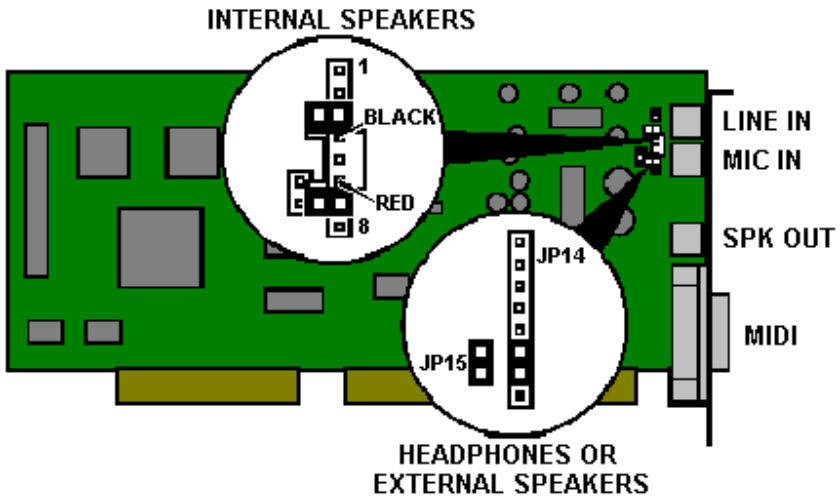
Waveform Audio Overview, Waveform Structures

ANEXO E - ESPECIFICAÇÕES DA PLACA SOUND BLASTER

As especificações foram extraídas integralmente do site da Mitsubishi Electric e podem ser acessadas através no endereço http://www.osemidlands.co.uk/support/insight/insight/en/common/addin/soundbla.htm

Sound Blaster Cards

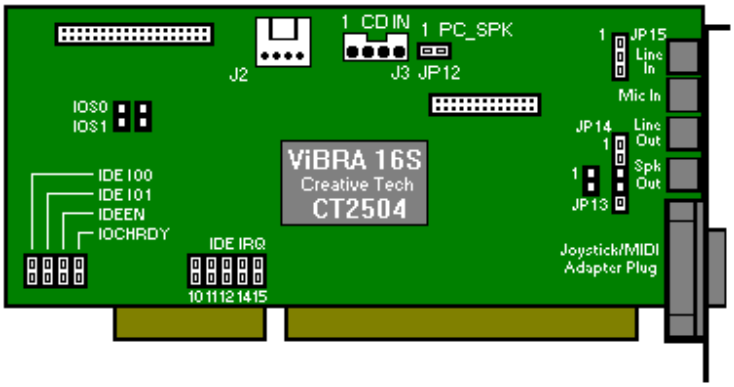
Sound Blaster Pro



Reconfiguring for headphones or external speakers

1. Power the system down, take suitable antistatic precautions and remove the system unit cover.
2. Identify jumpers JP14 and JP15 and the cable, from the illustration above.
3. Disconnect the cable from pins 4, 5 and 6 of JP14.
4. Remove the jumper links from pins 3 and 7 of JP14 and replace on across pins 6 and 7 of JP14, and the other across both pins of JP15.

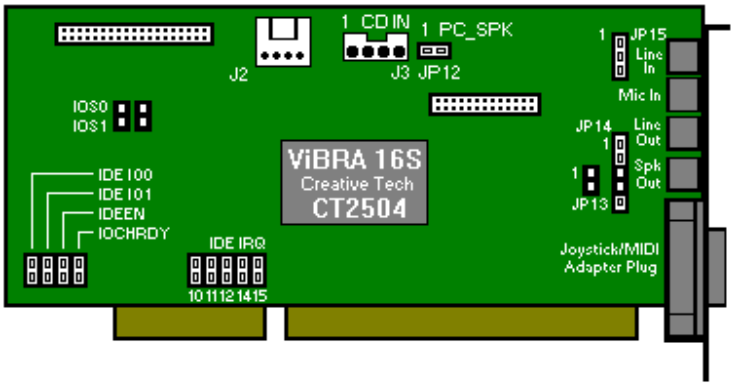
Sound Blaster 16 (Vibra) CT2260



Reconfiguring for headphones or external speakers

1. Power down the system, take suitable antistatic precautions and remove the system unit cover.
2. Identify jumpers JP16 and JP17 and the cable, from the above illustration.
3. Disconnect the cable from pins 1, 2 and 3 of JP17.
4. Remove the jumper links from pins 4 and 5 of JP17 and replace one across pins 3 and 4 of JP17, and the other across both pins of JP16.

Vibra 16 CT2801



Specification
Audio Specification

Frequency Response	20 Hz to 20 KHz (Line out)
Signal to Noise Ratio	80 dB (Line out)

THD + Noise	0.02% (Line out) 0.2% (Speaker out - typ.)
Output Power (Max.)	4 Watts PMPO per channel (4 W load minimum)
Maximum Output Signal	8.8 Vpp
AGC Amplifier Gain	20 dB to 46 dB (20 dB fixed gain when AGC is off)
Microphone Impedance	600 W
Microphone Input Range	30 mVpp to mVpp
Line-In Impedance	15 KW
Line-In Input Range	0 to 2 Vpp
CD Audio-In Impedance	15 KW
CD Audio-In Sensitivity	0 to 2 Vpp

Power Consumption

+5 V Current Consumption	250 mA (typ)
+12 V Current Consumption	300 mA (typ)
-12 V Current Consumption	50 mA (typ)

I/O MAPPING - Default settings in bold face

Audio	220H - 22FH or 240H - 24FH 260H - 26FH or 280H - 28FH
MPU-401 UART	300H - 301H or 330H - 331H
Joystick	200H - 207H
Music Synthesis	388H - 38BH
IDE interface	170 - 177 & 376 - 377 Secondary 1E8 - 1EF Tertiary 168 - 16F Quaternary

INTERRUPTS and DMA Channels - Default settings in bold

DMA setting	
8-bit	1 and 3
16-bit	5 and 7
IRQ setting	2, 5, 7, 10
IDE setting	10, 11, 12, and 15

TEMPERATURE RANGE

Operation	10 to 50 ºC
Non-operating	-40 to 70 ºC

Audio In Pin Assignments

CD_IN CONNECTOR (J3)

PIN	SIGNAL	I/O
1	Ground	IN
2	CD Left Channel	IN
3	Ground	IN
4	CD Right Channel	IN

Internal PC Speaker

PC SPK CONNECTOR (JP12)

PIN	SIGNAL	I/O
1	+5 V	IN
2	SPK	IN

Audio Extension Pin Assignments

CONNECTOR (JP15)

PIN	DESCRIPTION
1	MICGND - Mic input ground.
2	MICGND - Mic input ground.
3	MICL - Mic input left channel.

CONNECTOR (JP14)

PIN	DESCRIPTION
1	SPKGND - Speaker output ground.
2	SPKR - Speaker output right channel. Max. output voltage is 3Vrms at 4W.
3	SPKL - Speaker output left channel. Max. output voltage is 3Vrms at 4W.
4	SPKRL - Speaker output return signal for left channel.
5	SPKRR - Speaker output return signal for right channel.

CONNECTOR (JP13)

PIN	DESCRIPTION
1	SPKR - Speaker output right channel. Max. output voltage is 3Vrms at 4W.
2	SPKRR - Speaker output return signal for right channel.

I/O Address Settings

ADDRESS	JUMPER - IOS0	JUMPER - IOS1
220H (default)	ON	ON
240H	OFF	ON
260H	ON	OFF
280H	OFF	OFF

IDE Interface Settings

STATUS	JUMPER - IDEEN	JUMPER - IDECHRDY
Enabled	ON (default)	ON
Disabled	OFF	OFF (default)

IDE PORT	I/O Address	JUMPER - IDEIO0	JUMPER - IDEIO1
Secondary (default)	170-177 & 376-377	OFF	ON
Tertiary	1E8-1EF	ON	OFF
Quaternary 1	68-16F	OFF	OFF

IDE PORT	IRQ Lines
Secondary (default)	15 (default)
Tertiary	11 or 12
Quaternary	10 or 11