UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Daniel Alvienes Wentz

OSCILOSCÓPIO VIA PC COM AQUISIÇÃO MICROCONTROLADA

Canoas, Julho de 2009

Daniel Alvienes Wentz

OSCILOSCÓPIO VIA PC COM AQUISIÇÃO MICROCONTROLADA

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Departamento:

Engenharia Elétrica

Área de Concentração

Desenvolvimento e Programação

Professor Orientador:

MSc. Eng. Eletr. Dalton Luiz Rech Vidor – CREA-RS: 79.005-D

Canoas

2009

FOLHA DE APROVAÇÃO

Nome do Autor: Daniel Alvienes Wentz

Matrícula: 011102053-0

Título: Osciloscópio Via PC com Aquisição Microcontrolada

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Professor Orientador:

MSc. Eng. Eletr. Dalton Luiz Rech Vidor

CREA-RS: 79005-D

Banca Avaliadora:

Esp. Eng. Eletr. Márcio Gil Faccin

CREA-RS: 122301

Conceito Atribuído (A-B-C-D):

Eng. Eletr. Sílvio Longoni Debaco

CREA-RS: 079566

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor	Orientador
Daniel Alvienes Wentz	Dalton Luiz Rech Vidor
Avaliador	Avaliador
Márcio Gil Faccin	Sílvio Longoni Debaco

Relatório Aprovado em:

AGRADECIMENTOS

A todos que colaboraram direta ou indiretamente na elaboração deste trabalho, o meu reconhecimento.

Ao Professor Dalton Luiz Rech Vidor pela dedicação e pelas valiosas contribuições na orientação deste projeto.

A toda minha família pelo apoio durante os anos de estudos, principalmente aos meus pais que sempre foram um exemplo a seguir.

A minha amada esposa Gabriela por acreditar e me fazer acreditar, estando sempre do meu lado.

EPÍGRAFE

"E não vos conformeis com este mundo, mas transformai-vos pela renovação da vossa mente."

Romanos 12:2

RESUMO

ALVIENES WENTZ, Daniel. **Osciloscópio via PC com aquisição microcontrolada**. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2009.

O presente projeto desenvolve uma ferramenta semelhante a um osciloscópio para análise de tensão utilizando o computador. O trabalho descreve a montagem de um circuito microcontrolado com ganho programável para aquisição dos dados e a elaboração de um *software* para visualização em forma de onda do sinal analisado no tempo.

Palavras chave: Osciloscópio. Tensão. Microcontrolador. Circuito.

ABSTRACT

ALVIENES WENTZ, Daniel. **Oscilloscope via personal computer with micro-controlled acquirement**. Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2009.

This project aims at the creation of a tool similar to an oscilloscope, for analysis of voltage using the computer. The project describe the assembly of a micro-controlled circuit with programmable gain for data acquisition, and the development of software for viewing the analyzed signal in wave form.

Keywords: Oscilloscope. Voltage. Micro-controller. Circuit.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D: Conversor Analógico para Digital

ADC: *Analog to Digital Converter*

ALU: *Arithmetic and Logic Unit*

COM: *Comunication port*

Filtro RC: Filtro Resistivo e Capacitivo

I/O: *input / output*

PC: *Personal Computer*

PIC: *Peripheral Interface Controller*

RAM: *Random Access Memory*

RISC: *Reduced Instruction Set Computer*

RS232: *Recommended Standard* 232

Trigger: Gatilho

UART: *Universal Asynchronous Receiver and Transmitter*

USB: *Universal Serial Bus*

LISTA DE SÍMBOLOS

V – volt (Símbolo de tensão elétrica)

Vpp – volt pico-a-pico

Ω – Ohm (Símbolo de resistência elétrica)

GND – *Ground*

K – kilo, 1×10^{3}

m – mili, 1×10^{-3}

μ – micro, 1×10^{-6}

n – nano, 1×10^{-9}

Hz – hertz (Símbolo de frequência, Hz = 1/segundos)

SUMÁRIO

1 INTRODUÇÃO

Um dos instrumentos mais utilizados durante as aulas práticas do curso de Engenharia Elétrica e também no cotidiano dos profissionais da área de eletrônica é o osciloscópio. Trata-se de uma ferramenta de análise de circuitos, geralmente utilizada em testes onde são necessárias visualizações do formato de onda de um sinal em altas freqüências.

Alguns fatores limitam o uso do osciloscópio. Quando se trata de um osciloscópio analógico tem-se um preço mais acessível do que os digitais, entretanto, são ferramentas de tecnologia antiga, grandes, pesados e não possuem recursos de salvar as imagens geradas.

Já os osciloscópios digitais apresentam um melhor desempenho e a possibilidade de contar com analisador de espectro embutido. Outras vantagens como quantidade de canais e interface com o computador tornam o custo do osciloscópio digital mais elevado, inviabilizando economicamente a sua utilização particular.

Hoje em dia, já podemos encontrar no mercado osciloscópios para PC com comunicação USB, porém sua aquisição ainda requer um grande investimento pois são produtos importados tendo embutido no seu valor final taxas de importação e de cambio. Outra característica deste instrumento é o projeto “fechado” de *hardware* e *software* dos construtores, que são empresas especializadas e que possuem tecnologias próprias.

Este projeto visa criar uma ferramenta de baixo custo para análise de sinais que tenha os recursos de um osciloscópio somado a facilidade de se trabalhar com os dados via *software* em qualquer computador. Com essa ferramenta será possível analisar circuitos fora dos laboratórios da universidade ou empresa devido a sua alta portabilidade, trazendo grandes vantagens aos alunos de Engenharia que frequentemente montam os circuitos propostos em suas casas.

Utilizando os conhecimentos adquiridos durante o curso de engenharia, o projeto será dividido em uma parte de hardware e outra de software.

A aquisição dos dados será feita por um circuito de tratamento do sinal de entrada com ganho controlado por *software* e filtro para frequências acima do limite. Este sinal será convertido para digital através de um conversor AD e armazenado na memória do microcontrolador. Este procedimento será repetido até que o número de amostras definido for alcançado. Após concluídas as amostragens, o conjunto destes dados serão enviados ao computador através do canal serial do microcontrolador e atualizado na tela em forma de onda.

O programa de computador será desenvolvido pela plataforma Borland Builder C++ e tem opções de escalas de amplitude e frequência do sinal a ser analisado, além de opções quanto a apresentação gráfica.

As principais opções gráficas propostas são: escala de tensão (eixo Y), quantidade de amostras (eixo X) e ajuste de *trigger* (gatilho). Esta ultima, é usada para sincronizar o osciloscópio de modo a mostrar a forma de onda estável na tela. Outros recursos como deslocamento da onda por nível de tensão e cálculos de período e freqüência serão implementados via *software*.

Para que qualquer estudante futuramente possa montar seu próprio kit de aquisição, um dos objetivos é construir um protótipo com componentes de fácil acesso no mercado. Logo, o trabalho apresentará limitações quanto à análise de frequências muito elevadas.

Isto se deve porque em sistemas digitalizados a resolução da onda depende da velocidade de amostragem do conversor. Com o uso de um microcontrolador para a aquisição dos dados tem-se um limitador de tempo entre amostragem que é proporcional ao seu *clock*. Da mesma forma, a memória utilizada para a armazenagem dos dados coletados limita a quantidade de amostras para um determinado tempo de análise.

Este documento está dividido em capítulos de forma que o leitor compreenda os procedimentos utilizados durante a implementação do projeto. Primeiramente, está descrito um referencial teórico da fundamentação do projeto de pesquisa. No capítulo seguinte está a descrição dos materiais utilizados e a maneira como foram criados o software de computador e a placa de aquisição. No capítulo 4 pode-se encontrar o relatório dos testes e o levantamento dos resultados obtidos. Algumas sugestões de melhorias para um próximo protótipo estão colocadas junto com as conclusões no final do trabalho.

2 REFERENCIAL TEÓRICO

2.1 Osciloscópios

O Osciloscópio é um instrumento que permite a visualização e/ou medida do valor instantâneo de uma tensão em função do tempo. A leitura do sinal é feita numa tela sob a forma de um gráfico tensão × tempo (vertical × horizontal).

2.1.1 Principais características

Os osciloscópios têm vários ajustes, porém a quantidade varia de um modelo para o outro. Mas os ajustes básicos são os mesmos e para esta explicação pode-se observar um osciloscópio básico representado na figura abaixo:

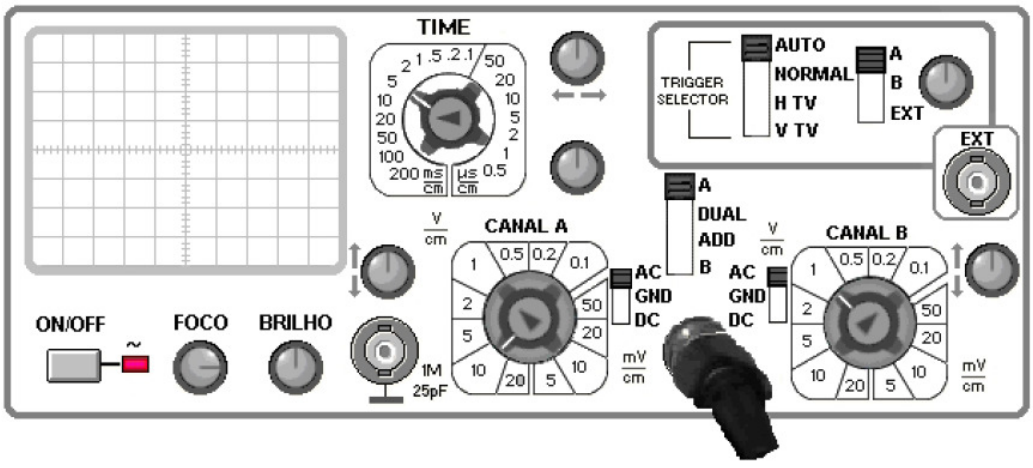


Ilustração 2-1 – Comandos básicos de um osciloscópio

O seletor AC/DC seleciona o tipo de tensão a ser aplicada à entrada do osciloscópio. Na posição GND a entrada do canal fica desligada e aparece apenas um traço no meio da tela. Em AC podemos visualizar uma tensão alternada aplicada e em DC podemos visualizar uma tensão contínua ou um sinal que varie, porém mantenha um eixo de CC como referência, conforme figura 2-2.

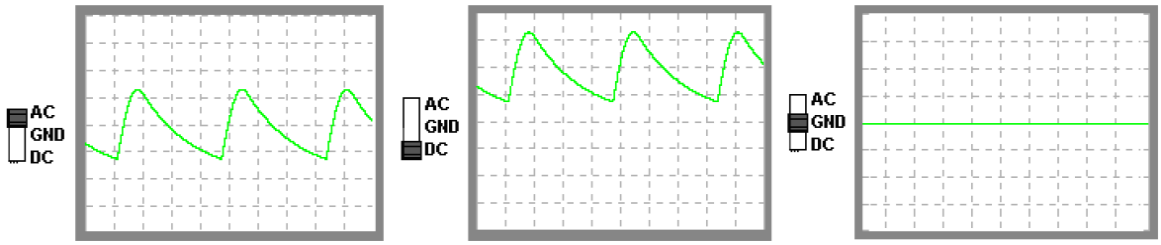


Ilustração 2-2 – Exemplo: seleção AC/DC

Para cada canal do osciloscópio há uma chave seletora de tensão. A tensão é lida no eixo vertical da tela (Y). A figura 2-3 demonstra um exemplo de uma chave de tensão.

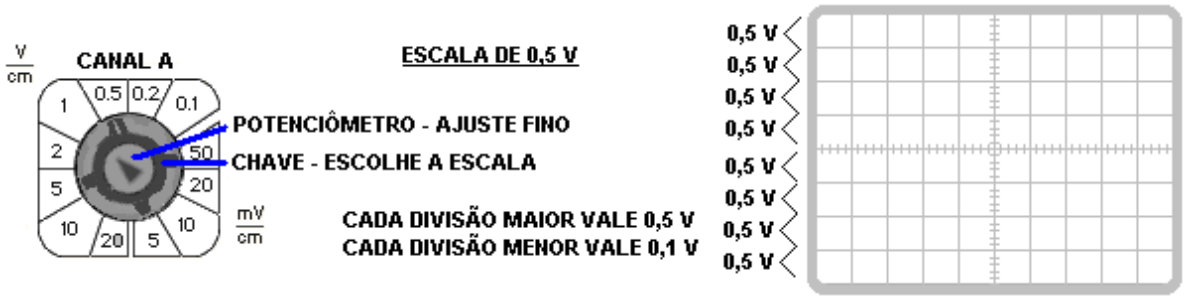


Ilustração 2-3 – Escala de Tensão

No osciloscópio, a frequência é lida no eixo horizontal (x). Porém não é a frequência que vem indicada e sim o período do sinal. Frequência é o número de vezes que um sinal muda de sentido (ciclo completo) por segundo. Período é o tempo necessário para o sinal completar um ciclo. Estas grandezas são inversamente proporcionais: F = 1/T. Por exemplo, a frequência da rede elétrica é 60 Hz e o seu período é: T = 1/60 = 0,016 s ou 16,6 ms. Na figura abaixo observa-se o botão que seleciona o tempo para cada divisão no eixo horizontal:

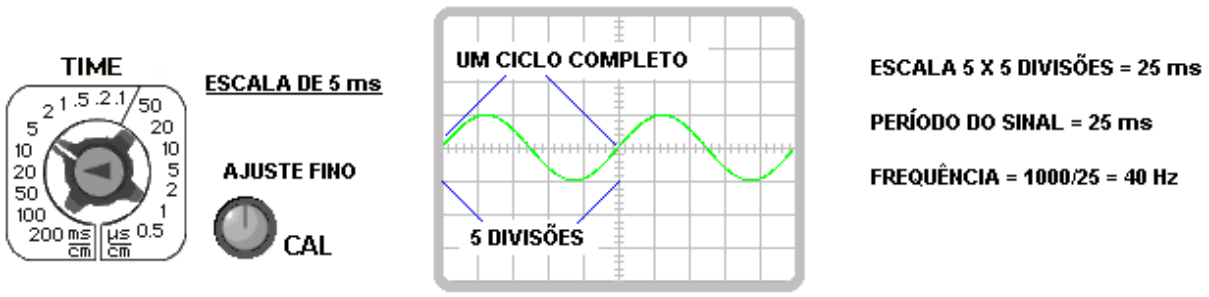


Ilustração 2-4 – Escala de Tempo

O *trigger* ou gatilhamento é um sinal usado para sincronizar o osciloscópio de modo a mostrar a forma de onda estável na tela. O sinal de *trigger* pode ser aplicado por um cabo externo ou pode ser usado o próprio sinal a ser medido para sincronizar o osciloscópio. Normalmente há duas chaves no painel do osciloscópio que escolhe o tipo e a fonte de *trigger* para sincronizar a imagem.

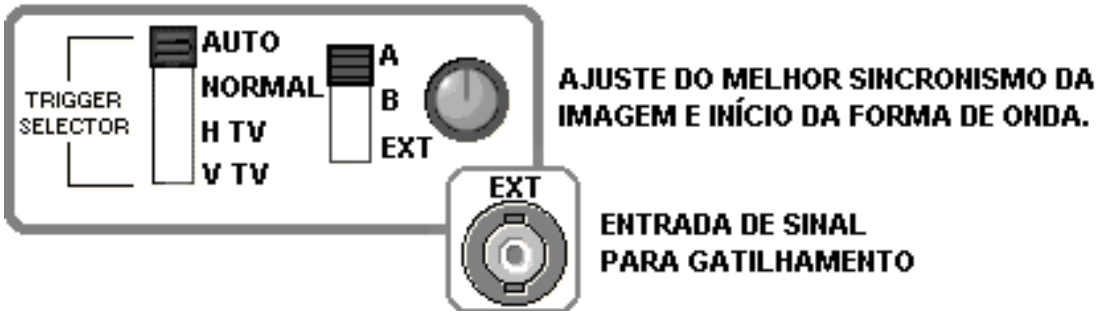


Ilustração 2-5 – Ajuste de *Trigger.*

Para interligar o osciloscópio aos pontos de medida são utilizadas as pontas de prova (ou ponteiras de prova). Uma das extremidades da ponteira de prova é conectada a uma das entradas do osciloscópio através de um conector e a extremidade livre serve para conexão aos pontos de medida.

A extremidade livre possui uma garra jacaré, denominada de terra da ponta de prova, que deve ser conectada ao terra do circuito e uma ponta de entrada de sinal, que deve ser conectada no ponto que se deseja medir. Na figura 2-6 observa-se um desenho representando uma ponteira de prova.

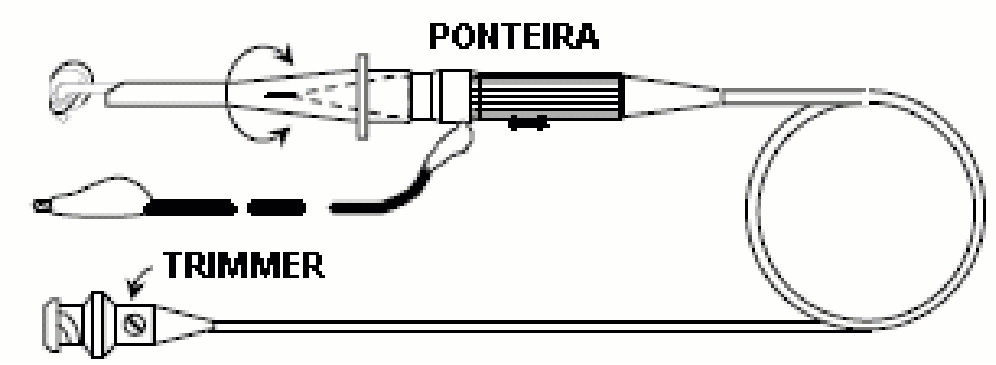


Ilustração 2-6 – Ponteira de prova.

2.1.2 Osciloscópios analógicos

Os osciloscópios analógicos possuem os seguintes elementos básicos:

- Tubo de raios catódicos (TRC): é o componente principal do osciloscópio que permite a visualização de um feixe eletrônico numa tela fosforescente. Um filamento aquecido por uma corrente elétrica emite elétrons que são, em seguida, acelerados e focalizados na direção da tela fosforescente, formando nesta um ponto luminoso. Placas metálicas de deflexão horizontal e vertical são utilizadas para produzir um campo elétrico perpendicular à trajetória do feixe eletrônico, possibilitando a sua deflexão e conseqüente deslocamento do ponto luminoso na tela fosforescente. O ponto luminoso deslocando-se na tela em alta velocidade fornece a impressão visual de uma linha contínua [10].

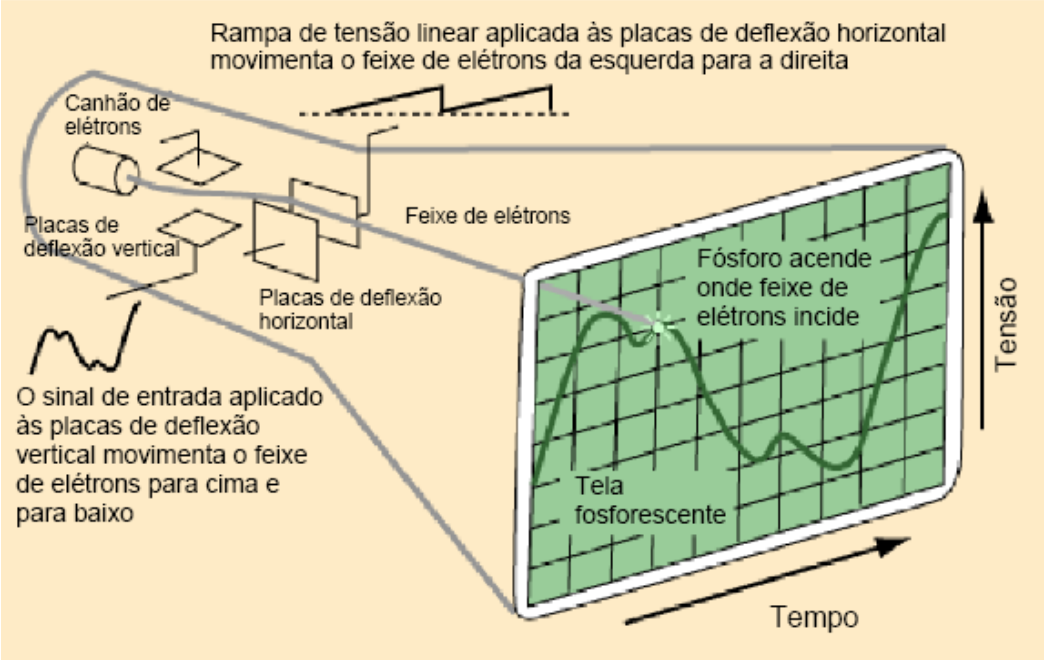


Ilustração 2-7 – Representação do funcionamento do TRC.

- Varredura vertical: o sinal a ser medido é amplificado e aplicado às placas de deflexão vertical que produzem um campo elétrico responsável pelo deslocamento vertical do feixe. Dessa forma a posição vertical do feixe na tela está diretamente relacionada com a amplitude do sinal de entrada. A tela fosforescente possui divisões que permitem uma medida visual do valor instantâneo do sinal. Tensões da ordem de centenas de Volts são necessárias para a

deflexão completa do feixe na tela. Um seletor de escalas baseado em um divisor resistivo é usado para adequar o sinal de entrada aos níveis do amplificador vertical.

- Varredura horizontal: para que o sinal a ser medido (vertical) possa ser visualizado em função do tempo (horizontal), é aplicada uma rampa linear de tensão nas placas de deflexão horizontal, produzindo um deslocamento do feixe da esquerda para a direita na tela, com uma velocidade constante. Esta rampa de tensão é baseada na carga de um capacitor por uma fonte de corrente constante. Uma vez que o feixe atinge a extremidade direita da tela, o mesmo retorna rapidamente à extremidade esquerda e é reiniciado o processo. Um seletor de escalas ajusta a velocidade de varredura do feixe através da carga do capacitor (corrente e/ou capacitância). Obs.: eventualmente pode ser usado um sinal externo qualquer para acionar a deflexão horizontal (modo X×Y).
- Sincronismo horizontal: a visualização "estática" de um sinal periódico na tela do osciloscópio só é possível quando a varredura horizontal do feixe está sincronizada em fase e freqüência com este sinal. Para que isso ocorra é necessário que o início da varredura horizontal seja definido por um sinal de disparo (trigger) proveniente do sinal a ser visualizado. Isso é obtido pela comparação do nível do sinal de entrada com uma tensão de referência interna ajustável.

2.1.3 Osciloscópios digitais

Neste tipo de osciloscópio, o sinal analógico de entrada é inicialmente convertido para o domínio digital através de um conversor A/D rápido, sendo em seguida armazenado em uma memória digital. Após o disparo (sincronismo horizontal) e um processamento matemático opcional, o sinal é apresentado em um *display* digital de modo semelhante aos monitores de vídeo de computadores.

Em um osciloscópio digital, os principais componentes encontrados para o tratamento do sinal são:

- Gerador de *clock* acionado pelo sincronismo horizontal. A freqüência de *clock* é função da base de tempo selecionada e define a taxa de amostragem do conversor A/D;

- O conversor A/D (em geral de 8 bits, paralelo) amostra o sinal de entrada periodicamente e o armazena em uma memória digital rápida. As taxas de amostragem podem variar de alguns Hz até GHz dependendo do conversor utilizado e da base de tempo selecionada;
- Os dados armazenados na memória sob a forma de uma matriz tensão × tempo podem sofrer um processamento matemático (operações algébricas, média, estatística, FFT, etc.) e são transferidos para a memória de vídeo, sendo em seguida apresentados no *display*, que pode ser colorido ou monocromático e do tipo TRC (tubo de raios catódicos), cristal líquido, plasma, etc. O tipo e as características do *display* não interferem na qualidade de aquisição do sinal.

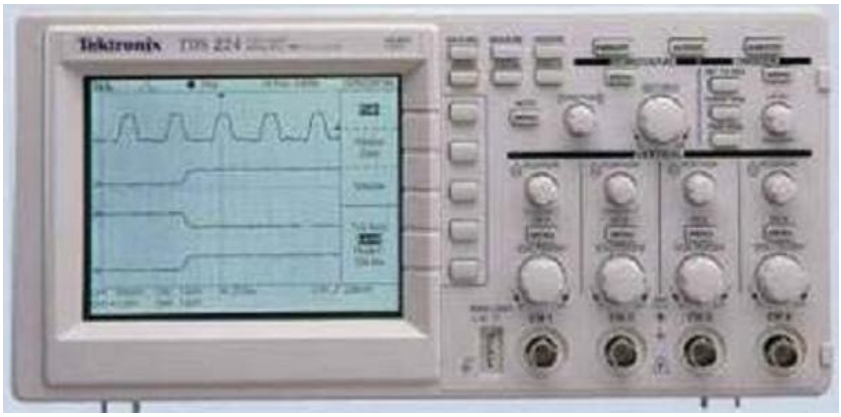


Ilustração 2-8 – Osciloscópio Digital

Pode-se destacar as principais vantagens dos Osciloscópios Digitais como:

- Visualização e armazenagem do sinal por tempo indefinido;
- Captura de sinais não periódicos e eventos únicos no tempo;
- Visualização do sinal antes do disparo (pre-*trigger*);
- Processamento matemático do sinal (processadores rápidos de última geração);
- Possibilita medidas diversas no sinal de forma mais precisa e direta;
- Visualização estática de sinais na tela independente de sua freqüência ou repetibilidade.

2.2 Conversão analógica digital

A necessidade de se coletar dados do mundo real que estão em formato analógico e arquivá-los em formas digitais nos força a utilizar conversores analógico/digital, também conhecidos como ADC (*Analog to Digital Converter* – A/D).

Um sinal analógico é contínuo na amplitude e no tempo, possuindo qualquer valor de amplitude dentro da sua faixa de variação para qualquer instante de tempo. Coletar dados com tal precisão e constância infinitesimal é impossível. O que os ADC's fazem é amostrar repetidas vezes o sinal, sendo que o valor da amplitude de cada amostragem é discretizada, ou seja, só pode medir valores específicos.

Portanto, como não temos uma cópia digital fiel do sinal real, podemos perder qualidade do sinal amostrado. Um método de se precaver contra isso é obedecer ao Teorema de Nyquist[1].

De acordo com o Teorema de Nyquist a quantidade de amostras por unidade de tempo de um sinal, chamada taxa de amostragem, deve ser maior que o dobro da largura de banda contida no sinal a ser amostrado, para que possa ser reproduzido fielmente. A metade da taxa de amostragem é chamada taxa de Nyquist e corresponde ao limite máximo de freqüência do sinal que pode ser reproduzido. Como não é possível garantir que o sinal não contenha sinais acima deste limite (distorções, interferências, ruídos, etc...), é necessário filtrar o sinal com um filtro passa baixo com freqüência de corte igual (ou menor) à taxa de Nyquist, ou filtro anti-aliasing.

Ressalta-se que teoricamente, um sinal contínuo, limitado em banda, se amostrado conforme o teorema, possui erro de reconstrução (retornando do domínio discreto para o contínuo) igual a zero, ou seja, a reconstrução é perfeita. Entretanto, para se considerar um sinal digital, é necessário discretizar também a amplitude de cada amostra. Esta segunda discretização, entretanto, introduz erros que impedem a reconstrução do sinal com erro zero (retornando do domínio digital para o contínuo), uma vez que somos obrigados a associar uma palavra binária de dimensão finita para cada amostra. O erro introduzido por este passo é irrecuperável e dá origem a uma das maiores fontes de ruído em sinais digitais que

[1] Este teorema foi enunciado por Claude Shannon, que homenageou Nyquist, dando seu nome a ele pelos desenvolvimentos realizados anteriormente, dos quais Shannon se valeu. O procedimento de amostragem, mas com caráter mais matemático, sob o ponto de vista de aproximação de funções contínuas, também foi enunciado, independentemente, por Whittaker e Kotel'nikov, [8]

é conhecido como ruído de quantização. Modelos demonstram que, em quantização uniforme, que é bastante empregada em instrumentos de medição, para cada bit extra na palavra binária, a relação sinal/ruído eleva-se em 6dB [7].

Nas Figuras a seguir pode-se observar graficamente o efeito da digitalização em um sinal analógico [1].

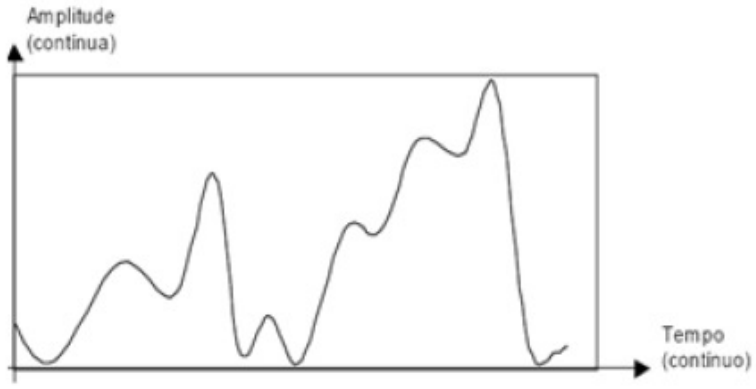


Ilustração 2-9 – Sinal analógico

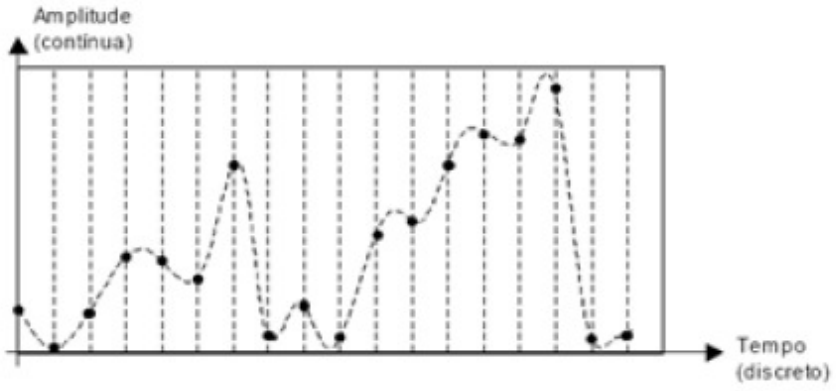


Ilustração 2-10 – Sinal discretizado no tempo

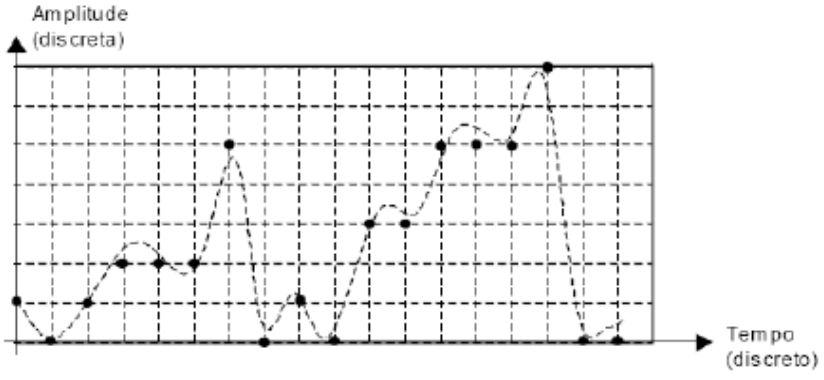


Ilustração 2-11 – Sinal discretizado no tempo e na amplitude

2.2.1 Tipos de conversores AD

Os tipos mais importantes de conversores A/D são:

- Rampa ou Integração (simples e dupla);
- Aproximação Sucessiva;
- Paralelo (ou Flash).

O método de Aproximação Sucessiva é o mais utilizado em microcontroladores por oferecer uma taxa de amostragem satisfatória para a maioria das aplicações e também por ser de baixo custo e baixo consumo quando comparado com os ADC *flash*.

2.2.2 Aproximação Sucessiva

O método da aproximação sucessiva precisa de um número fixo de ciclos de *clock* para chegar a um resultado. Como o próprio nome sugere, opera através de comparações que iniciam por um valor pré-determinado e, através de comparações sucessivas, encontra o dado equivalente ao valor analógico de entrada.

Um diagrama de blocos do funcionamento interno do ADC por Aproximação Sucessiva pode ser visualizado na Ilustração 2-4 [1].

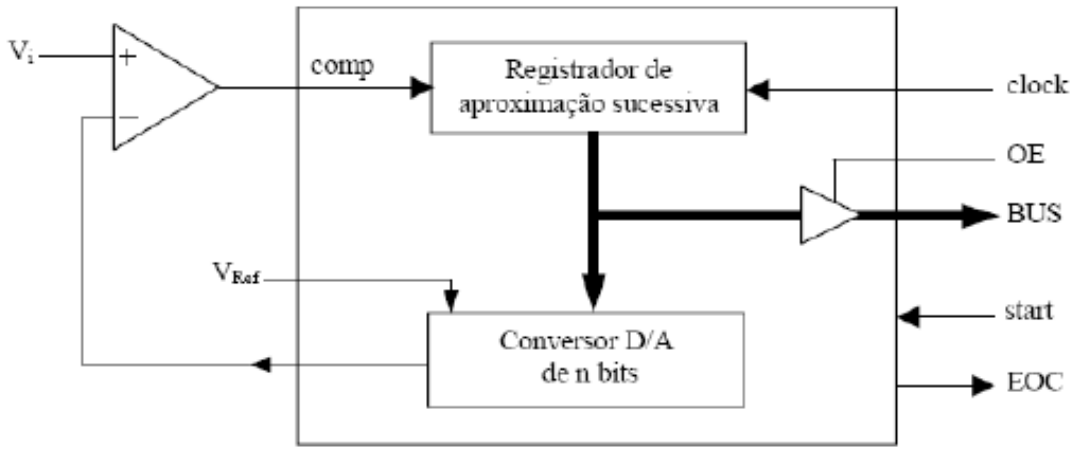


Ilustração 2-12 – Diagrama de blocos de um ADC por Aproximação Sucessiva

Para analisar o funcionamento do conversor A/D por aproximação sucessiva, tomemos um projeto que consiste em um conversor com intervalo de 0V a 15V e um degrau de tensão de 1V. Deste modo, obtêm-se 16 níveis e são necessários 4 bits, que geram a relação da Tabela 2-1.

Tabela 2-1 – Codificação Binária da Tensão

Tensão (V)	bit3	bit2	bit1	bit0	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	3ª
3	0	0	1	1	4ª
4	0	1	0	0	2ª
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	1ª
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

De acordo com a Tabela 2-1, o valor de tensão central é de 8V, sendo o ponto de partida para a comparação com o sinal analógico de entrada. Os próximos níveis de comparação, que são definidos pelo valor de entrada (Vin), têm os bits remanejados de forma a resultar em um dado de saída correspondente ao valor analógico de entrada. Se a entrada Vin for menor que a tensão com a qual está sendo comparada (Vin<Vcomp), então torna-se o bit respectivo igual a zero e o próximo igual a 1. Se Vin>Vcomp, mantém-se o bit em questão e coloca-se o próximo bit em 1.

Por exemplo, se tiver um valor de entrada igual a 3,7 volts, (1ª passo) o conversor compara com 8 volts (na Tabela 2-1 é representado pelo valor binário 1000b), verifica que é menor, torna o bit3=0 e o bit2=1. (2ª passo), ficando desta maneira com 4V (na Tabela 2-1 é representado pelo valor binário 0100b). Uma nova comparação é realizada e, como 4V ainda é maior que 3,7V, deve-se tornar o bit2=0 e o bit1=1 (3ª passo). Em seguida tem-se 2V (na Tabela 2-1 é representado pelo valor binário 0010b), que é menor que 3,7V, e o sistema mantém bit1=1 e coloca bit0=1 (4ª passo). Desta forma, tem-se 3V (na Tabela 2-1 é representado pelo valor binário 0011b). Como o conversor possui um degrau de 1V, o valor de saída correspondente aos 3,7V de entrada, será 3V. Observe que, neste caso, o sistema ainda considera Vin = 3,7V maior que 3V. Como anteriormente já foi verificado que

este valor é menor do que 4V e maior do que 2V, logicamente, para o sistema, a tensão de saída só poderá ser 3V.

2.3 Comunicação serial

Essa transmissão converte informações em paralelo (geralmente oriundas de computadores) para informações seriais, ou seja, bit a bit (geralmente oriundas de modems, linhas telefônicas ou para placa serial de outro micro). Cada bit representa uma parte da mensagem. Os bits individuais são então rearranjados no destino para compor a mensagem original. Em geral, um canal irá passar apenas um bit por vez.

2.3.1 Padrão EIA RS-232

O padrão mais utilizado é o RS-232 e foi definido em 1969 pela *Eletronic Industries Association* (EIA). Especifica as características elétricas dos circuitos e fornece nome e números às linhas necessárias à conexão entre equipamentos.

Os microcomputadores utilizam somente 9 pinos conectados em lugar dos 25 pinos necessários para o circuito completo RS232. Além disso, o padrão define:

- Para que periféricos possam "conversar" através da mesma linha, eles são divididos em 02 tipos: o tipo terminal: que usa o pino 2 para mandar dados e o tipo modem que usa o pino 2 para receber dados;

- periféricos terminais devem ser acoplados de um conector macho e periféricos modem de um conector fêmea.

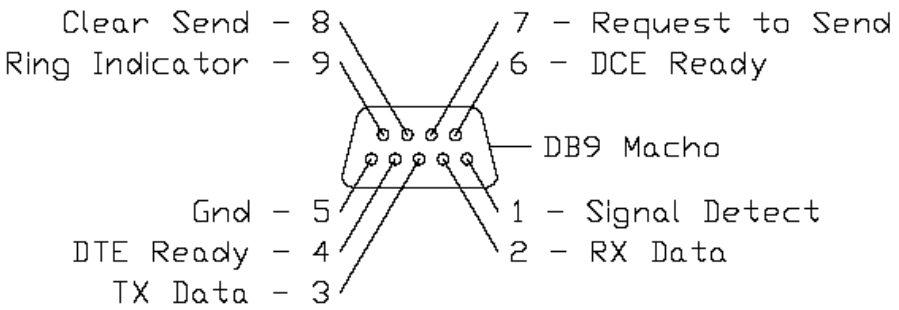


Ilustração 2-13 – Padrão RS232 - conector DB-9

2.3.2 Taxa de transferência

A taxa de transferência refere-se a velocidade com que os dados são enviados através de um canal e é medido em transições elétricas por segundo

2.3.3 Transmissão Assíncrona x Transmissão Síncrona

Duas técnicas básicas são empregadas para garantir a sincronização correta. Em sistemas síncronos, canais separados são usados para transmitir dados e informação de tempo. O canal de temporização transmite pulsos de *clock* para o receptor. Através da recepção de um pulso de *clock*, o receptor lê o canal de dado e armazena o valor do bit encontrado naquele momento. O canal de dados não é lido novamente até que o próximo pulso de *clock* chegue. Como o transmissor é responsável pelos pulsos de dados e de temporização, o receptor irá ler o canal de dados apenas quando comandado pelo transmissor, e portanto a sincronização é garantida.

Em sistemas assíncronos, a informação trafega por um canal único. O transmissor e o receptor devem ser configurados antecipadamente para que a comunicação se estabeleça a contento. Um oscilador preciso no receptor irá gerar um sinal de *clock* interno que é igual (ou muito próximo) ao do transmissor. Para o protocolo serial mais comum, os dados são enviados em pequenos pacotes de 10 ou 11 bits, dos quais 8 constituem a mensagem. Quando o canal está em repouso, o sinal correspondente no canal tem um nível lógico '1'. Um pacote de dados sempre começa com um nível lógico '0' (*start bit*) para sinalizar ao receptor que um transmissão foi iniciada. O "*start bit*" inicializa um temporizador interno no receptor avisando que a transmissão começou e que serão necessários pulsos de *clocks*. Seguido do *start bit*, 8 bits de dados de mensagem são enviados na taxa de transmissão especificada. O pacote é concluído com os bits de paridade e de parada ("*stop bit*").

2.3.4 Paridade

Ruídos e distúrbios elétricos momentâneos podem causar mudanças nos dados quando estão trafegando pelos canais de comunicação. Se o receptor falhar ao detectar isso, a mensagem recebida será incorreta, resultando em conseqüências possivelmente sérias. Como uma primeira linha de defesa contra erros de dados, eles devem ser detectados. Se um erro pode ser sinalizado, pode ser possível pedir que o pacote com erro seja reenviado, ou no mínimo prevenir que os dados sejam tomados como corretos. Se uma redundância na informação for enviada, 1 ou 2 bits

de erros podem ser corrigidos pelo hardware no receptor antes que o dado chegue ao seu destino. O bit de paridade é adicionado ao pacote de dados com o propósito de detecção de erro. Na convenção de paridade-par ("*even-parity*"), o valor do bit de paridade é escolhido de tal forma que o número total de dígitos '1' dos dados adicionado ao bit de paridade do pacote seja sempre um número par. Na recepção do pacote, a paridade do dado precisa ser recomputada pelo *hardware* local e comparada com o bit de paridade recebido com os dados. Se qualquer bit mudar de estado, a paridade não irá coincidir, e um erro será detectado. Se um número para de bits for trocado, a paridade coincidirá e o dado com erro será validado. Contudo, uma análise estatística dos erros de comunicação de dados tem mostrado que um erro com bit simples é muito mais provável que erros em múltiplos bits na presença de ruído randômico. Portanto, a paridade é um método confiável de detecção de erro [9].

3 MATERIAIS E MÉTODOS

O projeto consiste em uma placa de aquisição ligada ao computador através de comunicação serial e um *software* para interface com o usuário.

A placa de aquisição está dividida em um circuito de condicionamento do sinal de entrada que é responsável pelos ajustes de ganho e de *offset* de tensão, e circuito para conversão de analógico para digital através de um microcontrolador.

Na figura 3.1 é apresentado o diagrama de blocos do projeto.

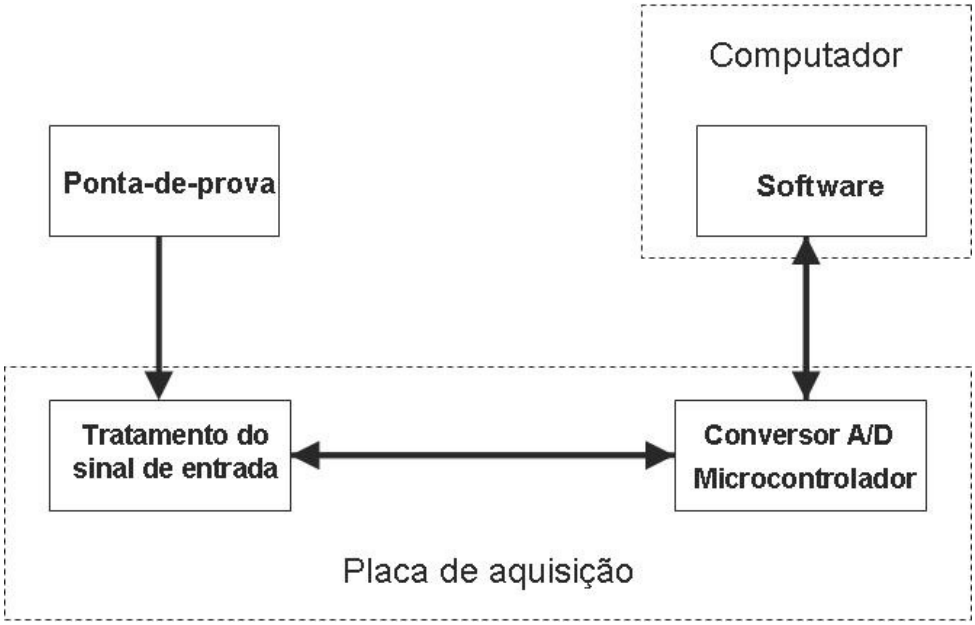


Ilustração 3-1 – Diagrama de Blocos

A aquisição dos dados foi desenvolvida através do microcontrolador PIC16F877A que devido suas características, tais como A/D interno e memória FLASH programável, possibilitou uma simplificação na construção do *hardware.* Outro fator que pesou na escolha pelo microcontrolador da família PIC é o seu baixo custo de aquisição por se tratar de um componente facilmente encontrado no mercado.

A comunicação com o computador é feita serialmente através do protocolo RS-232 e o software foi desenvolvido em linguagem C usando a plataforma Borland Builder C++.

3.1 Microcontrolador PIC16F877A

A família PIC possui um elevado número de microcontroladores, que se diferem pelo número de portas I/O, o tipo e o tamanho da memória, melhoria da performance dos circuitos periféricos e a flexibilidade dos dispositivos que não fazem parte da arquitetura dos microcontroladores, como os comparadores analógicos e os conversores A/D.

O PIC 16F877A contém todos os componentes necessários para o completo desenvolvimento de um sistema digital eletrônico programável. Este possui arquitetura RISC (*Reduced Instruction Set Computer*), 8 bits, até 20 MHz de *clock*, possui 8 Kbytes de memória FLASH programável, 368 bytes de memória RAM, conversor analógico / digital com 8 canais multiplexados, 32 entradas / saídas digitais e 3 *timers* para geração de PWM e contadores.

O PIC16F877A é um objeto mais complexo, com um tratamento completo, o qual foi definido para o escopo deste projeto. Portanto, nos parágrafos seguintes, nos limitaremos a descrever a arquitetura geral e nos aprofundarmos na funcionalidade da utilização para a nossa aplicação e o motivo pelo qual este dispositivo foi escolhido. (Microchip - PIC16F877A Data Sheet, 2003)

3.1.1 Arquitetura

A arquitetura RISC do PIC tem um número relativamente reduzido de instruções. A sua arquitetura mostrada na figura 3-2, com a memória e o barramento de dados separados da memória e do barramento do programa. Particularmente, quando o barramento de dados é de 8 bits, aquele para as instruções é de 14 bits, e é dimensionado de modo a poder capturar cada instrução com um único acesso à memória do programa. Além disso, esta estrutura permite ao mesmo tempo a execução de uma instrução e a leitura da próxima instrução, portanto, são lidos dois níveis de instruções ao mesmo tempo. (Microchip - PIC16F877A Data Sheet, 2003)

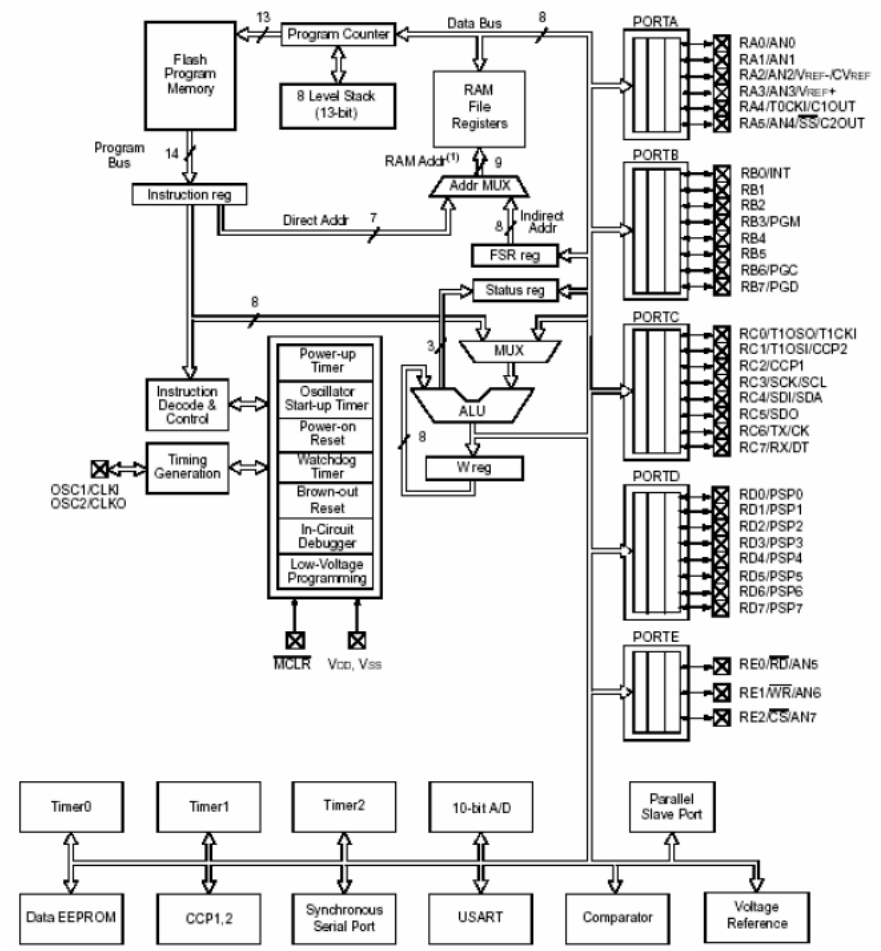


Ilustração 3-2 – Arquitetura do PIC16F877A

Na figura 3-2 pode-se ver o bloco principal do microcontrolador:

- Os pinos externos são organizados em 5 portas de I/O;
- A ALU (*Arithmetic and Logic Unit*), realiza as funções de cálculo e de elaboração dos dados durante a execução do programa;
- O barramento de programa e o barramento de dados;
- A memória de programa (*Flash Program Memory*), a memória de dados e uma terceira memória, a Data EEPROM;
- Alguns registros particulares, como o Program Counter, o Status Register, registros W e a pilha de 8 níveis (*8 level stack*);
- Diversos temporizadores: *Watch Dog Timer Timer0, Timer1 e Timer2*;
- Conversor A/D de 10 bit;
- Outros periféricos, como comparadores analógicos e a porta serial síncrona.

3.1.2 Organização da Memória

O PIC16F877A tem três blocos de memória: a memória de programa, memória de dados e a EEPROM.

A memória de programa é uma FLASH composta de 8k de palavras de 14 bits. O *program counter* contem 13 bits e durante a execução do programa é automaticamente incrementado. Se ocorrer a chamada de uma sub-rotina ou uma solicitação de interrupção, o *program counter* direciona ao endereço da primeira instrução da sub-rotina, ao invés de informar todas as vezes o endereço da instrução sucessiva.

A memória de dados é uma RAM estática formada por 4 bancos de 128 bytes cada. Os registros localizados na parte inferior da memória são chamados de *Special Function Registers* (SFR), e os localizados na parte superior da memória são chamados de *General Purpose Registers* (GPR). A organização da memória é apresentada na figura 3-3.

Os SFR são registros utilizados pela CPU para o controle dos periféricos e a gestão das operações que o dispositivo exige. Portanto, todas as localizações da memória (1 byte) constituem um registro dedicado de uma função específica. Os registros mais importantes, os quais são replicados em todos os bancos de memória, são mostrados abaixo: (Microchip - PIC16F877A Data Sheet, 2003)

- *STATUS REGISTER* - Contém o bit de seleção dos bancos de memória de dados, os *flag's* são indicados no estado aritmético da ALU;
- O registro *OPTION_REG* – Contém o bit de controle das instruções externas, do *PULL-UP* da Porta B (*PORTB*) e o TMR0 *Prescaler* / WDT *Postscaler*;
- O registro *INTCON* – Contém vários *flag's* para a gestão das instruções;
- O registro *PCLATH* – Permite que o *locus* da memória possa ser selecionado;
- Os registros FSR (*File Select Register*) e INTN – São reservados para o endereçamento direto da memória. Particularmente, algumas operações de leitura ou escrita são colocadas no registro INTN, que na verdade não é um registro físico, é executado na localização de memória cujo endereço está escrito no FSR.

Há também os registros W, cuja particularidade consiste no fato da ALU poder ser acessada diretamente sem a necessidade de fornecer algum endereço de memória.

Os GPR (*General Purpose Register*), por sua vez, compreendem um total de 368 bytes, podendo ser usados para a memorização de dados.

Em todos os casos, quando se seleciona qualquer registro, é necessário avaliar que um bit de seleção do banco do *Status Register* é setado para aquele banco.

Finalmente, no microcontrolador existe também uma Data EEPROM, que não é mapeada no *File Register*, mas no qual é possível o acesso com o endereçamento indireto. (Microchip - PIC16F877A Data Sheet, 2003)

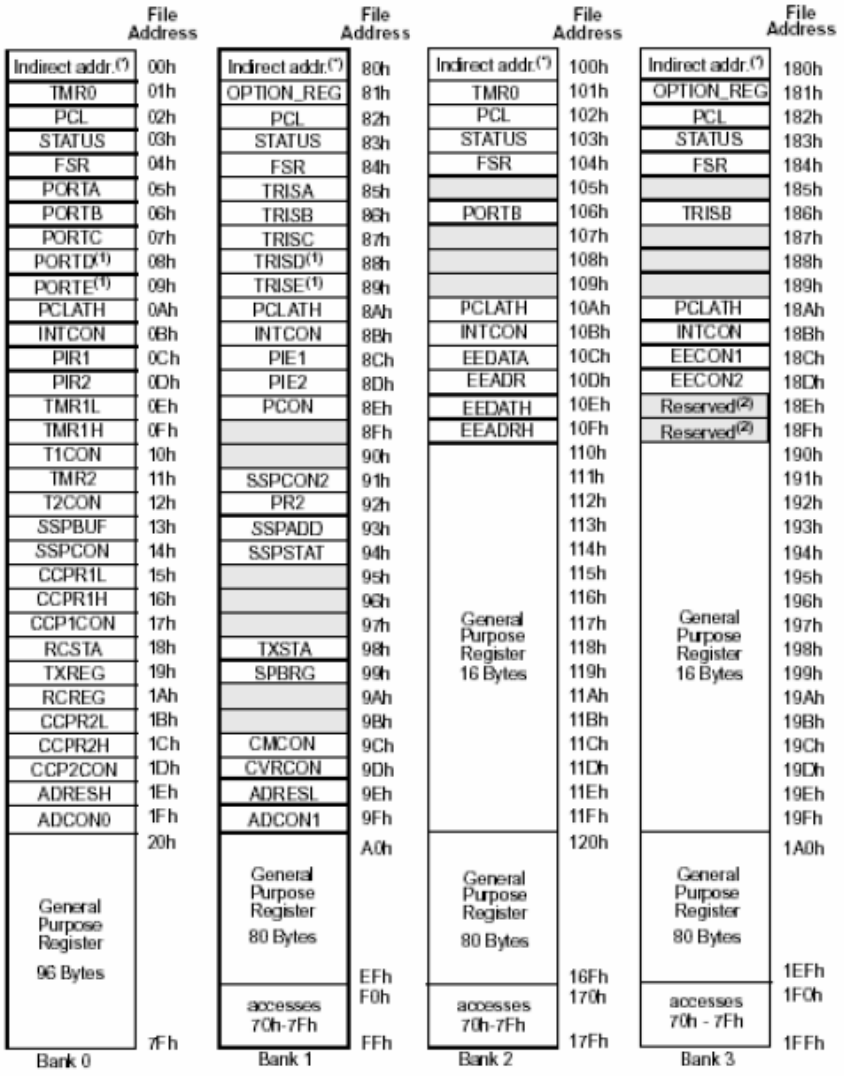


Ilustração 3-3 – Esquema de organização da memória do PIC16F877A.

3.1.3 Portas de I/O

Como pode ser visto na figura 3-4, os pinos externos do microcontrolador são organizados em 5 portas: uma de 6 bits, três de 8 bits e uma de 3 bits. Também encontramos pinos que são dedicados a funções particulares como a alimentação e a conexão do cristal. Pode-se notar que quase todos os pinos estão disponíveis para mais de uma função, que são selecionadas nos registros internos a que são dedicados.

O microcontrolador utiliza dois registros para a administração das portas de I/O: os registros PORTA, PORTB, PORTC, PORTD e PORTE, reportam os valores elétricos da porta, e os registros TRISA, TRISB, TRISC, TRISD e TRISE, todos associados a uma porta e nos quais os valores de cada bit determinam se os pinos estão selecionados para uma porta de entrada ou de saída.

Na figura 3-4 pode-se notar que a maior parte dos pinos é possível associar mais de uma função em cada pino.

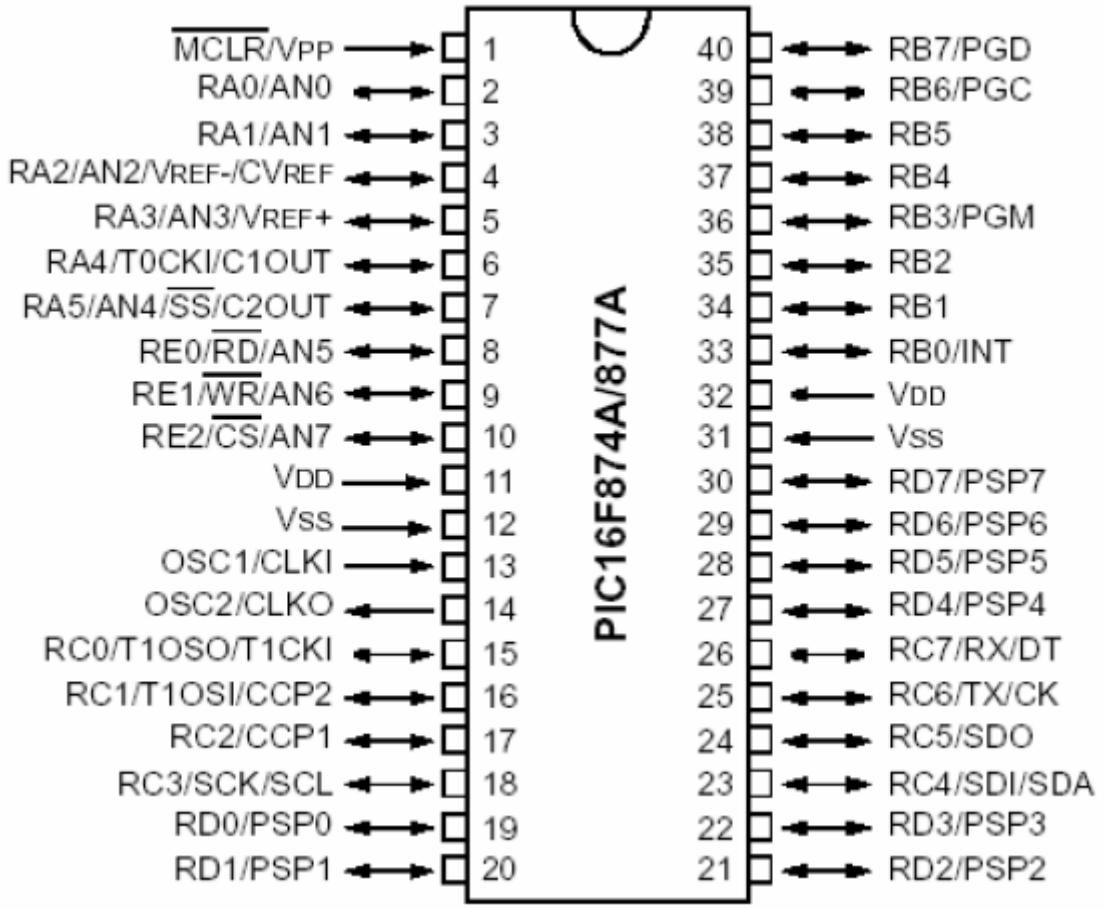


Ilustração 3-4 – Pinagem do PIC16F877A.

3.1.4 Conversores A/D

Interno ao PIC16F877A está disponível um conversor A/D de 10 bits com oito canais de entrada multiplexadas, posicionados em seus pinos correspondentes na porta PORTA (exceto no pino RA4) e PORTE. Os registros dedicados aos conversores A/D são quatro:

- ADRESH é a parte alta ou mais significativa da conversão do A/D;
- ADRESL é a parte baixa ou menos significativa da conversão do A/D. O ADRESH e ADRESL somados formam os 10 bit que constituem o resultado da conversão do A/D;
- ADCON0, controla a habilitação e o *clock* dos conversores e a multiplexação dos canais, e indica se uma conversão está sendo executada. A estrutura do ADCON0 pode ser vista na figura 3-5;

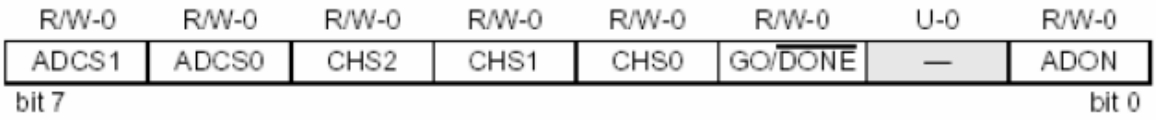


Ilustração 3-5 – Estrutura do ADCON0.

ADCON1, configura a função das portas PORTA e PORTE, determina o alinhamento do resultado da conversão nos registros ADRESH e ADRESL, e ainda administra o *clock* dos conversores. A estrutura do ADCON0 pode ser vista na figura 3-6.

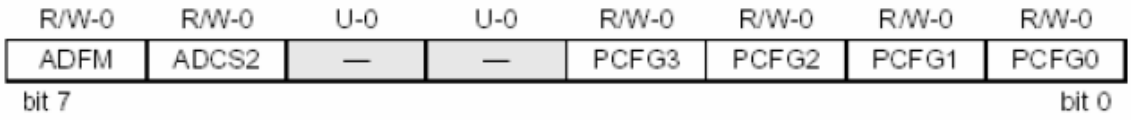


Ilustração 3-6 – Estrutura do ADCON1.

3.2 Ambiente de Programação e BootLoader

3.2.1 MPLAB IDE v7.31

O MpLab é um ambiente integrado de desenvolvimento (I.D.E.: *Integrated Development Environment*). No mesmo ambiente o usuário pode executar todos os procedimentos relativos ao desenvolvimento de um software para o PIC, tornando o

trabalho do projetista mais produtivo. O ambiente de programação do MpLab pode ser visto na figura 3-7.

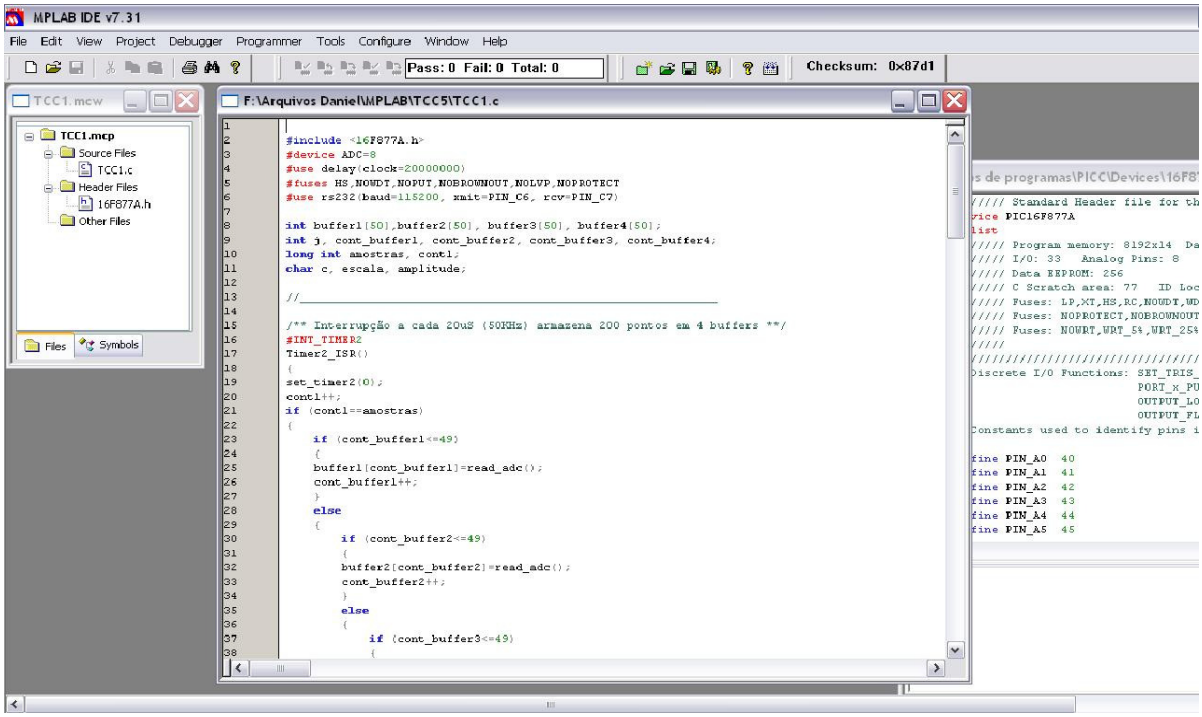


Ilustração 3-7 – Tela MPLAB.

Integrado ao MpLab foi utilizado o software HI-TECH, que possibilita que toda a programação seja feita na linguagem C.

Na compilação gera um arquivo com extensão .hex (hexadecimal) a partir dos arquivos de código fonte (.asm) e de projeto (.pjt). É o conteúdo do arquivo hexadecimal que é gravado na memória de programa do PIC.

3.2.2 BootLoader

A transferência de programas para os microcontroladores da família PIC é normalmente efetuada através de um dispositivo gravador específico. Este possui um inconveniente, que é a necessidade de a cada alteração do software e nova gravação o processador deve ser retirado do circuito e colocado no gravador.

Para facilitar e agilizar o processo de gravação do PIC, inicialmente foi gravado um software BootLoader. Assim que o pino do PIC, previamente configurado no BootLoader, é colocado em nível lógico 1 o BootLoader fica aguardando a chegada de um novo software pelos pinos seriais do PIC. Após o termino do download, o PIC começa a executar novo *software*.

Desta forma, não é necessária a retirada do PIC do circuito a cada nova programação, a necessidade de adquirir um gravador e possibilita o envio do *software* através da porta serial do PC. Na figura 3-8 pode-se visualizar a tela do *software* de download.

Ilustração 3-8 – Tela software BootLoader.

3.3 Circuito de Tratamento do Sinal de Entrada

A entrada do sinal analógico foi projetada para atender a três faixas diferentes de tensão através de ganhos chaveados pelo próprio microcontrolador com seleção via *software*.

Este circuito também é responsável pelo condicionamento do sinal a ser lido pelo A/D de 0 a 5V conforme visto anteriormente na descrição do microcontrolador. Como em sinais periódicos frequentemente temos tensões negativas, nossa referência de zero volt do sinal analisado deve ser 2,5V na entrada do conversor A/D. Esse ajuste de *offset* de tensão é feito através de um somador ligado ao primeiro estágio de ganho.

3.3.1 Simulação do condicionamento do sinal

Antes da implementação física do circuito de tratamento do sinal foi efetuado uma simulação utilizando a ferramenta LTspice.

O circuito inicial representado na figura 3-9 foi simulado de forma que o resistor R1 ligado a fonte V1 acrescenta um nível de tensão para ajuste do *offset,* e o resistor de entrada R2 está ligado ao sinal, definindo o ganho através da relação com o resistor de 1KΩ.

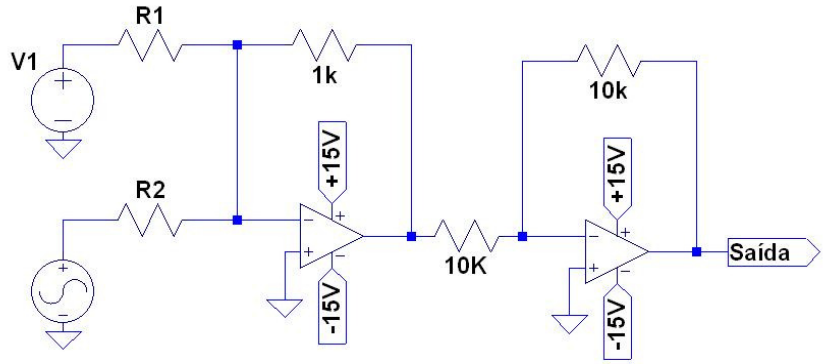


Ilustração 3-9 – Esquemático inicial da simulação.

Sabendo que o ganho do primeiro amplificador inversor é -1K/R2, para um sinal entre -2,5V e +2,5V precisamos de ganho igual a 1, assim R2 = 1KΩ. Na figura 3-10 observa-se a simulação deste caso, onde a entrada está representada na cor vermelha e a saída na cor azul.

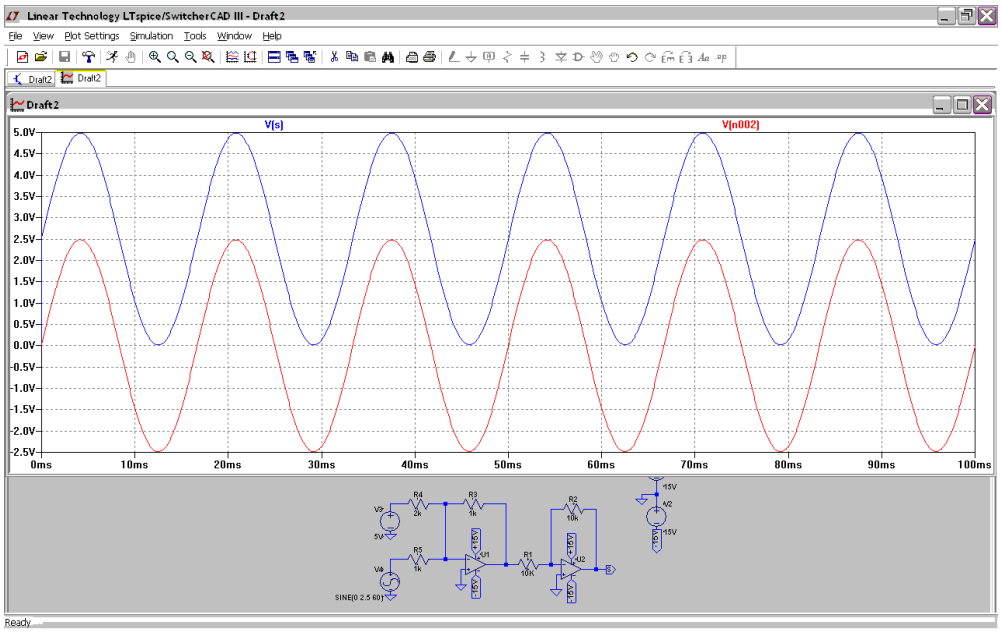


Ilustração 3-10 – Simulação sinal ±2,5V.

Pode-se observar uma reprodução perfeita da onda de entrada na saída, porém com o nível de tensão apropriado para a leitura do conversor A/D.

No segundo caso, foi simulado um sinal de -10V a +10V tendo como valor de R2 4KΩ, obtendo um ganho de 0,25 conforme a figura 3-11.

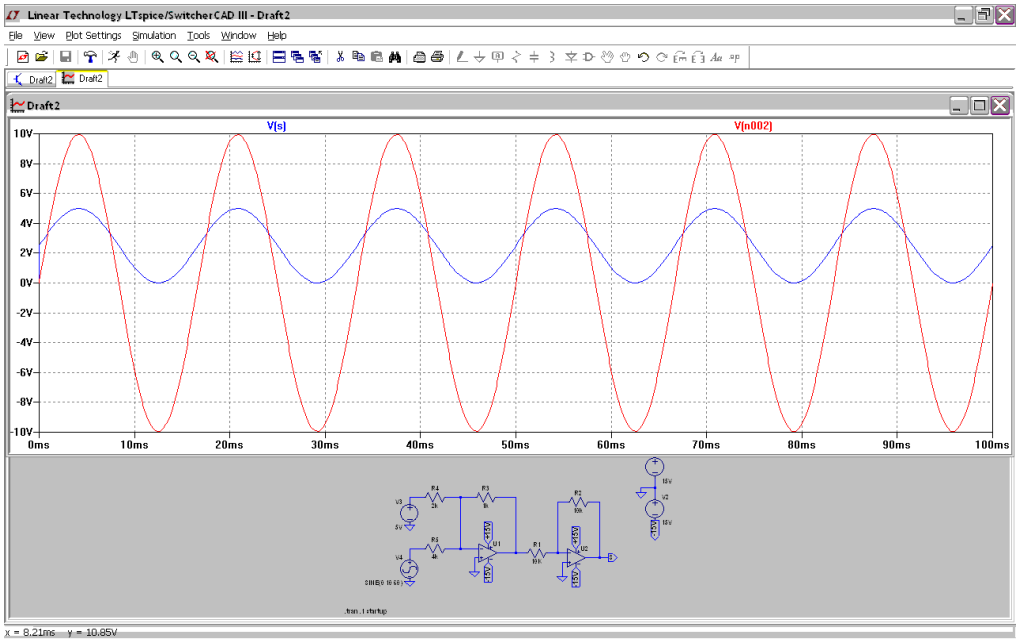


Ilustração 3-11 – Simulação sinal ±10V.

Finalmente, para um sinal de -50V a +50V o ganho necessário é 0,05 utilizando uma resistência de 20K na entrada.

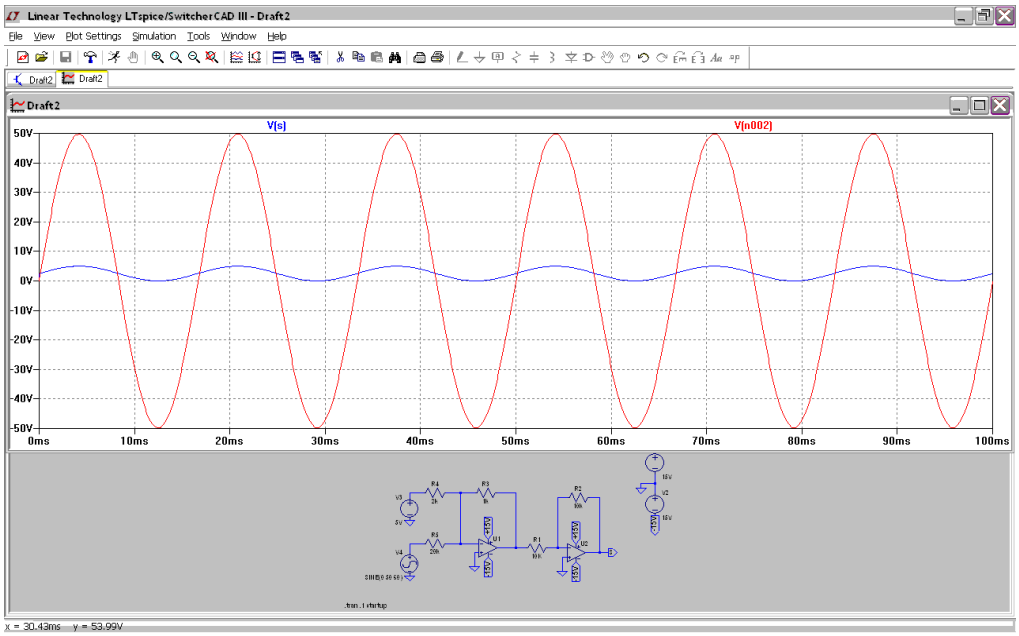


Ilustração 3-12 – Simulação sinal ±50V.

3.3.2 Ganho programável

Para que ocorra a amplificação ou atenuação apropriada do sinal a ser analisado, o circuito está dotado de chaveamentos de ganhos controlados por *software*.

Existem integrados chamados de PGA's (amplificadores de ganhos programáveis) utilizados para casos como este, porém verificou-se que de uma forma geral apresentavam apenas ganhos de 1, 10, 100 ou potências de 2, não se enquadrando nas necessidades do projeto.

Para isso, foi desenvolvido um circuito eletrônico utilizando relés para efetuar o controle do ganho, o que garante também uma maior robustez elétrica conforme figura 3-13.

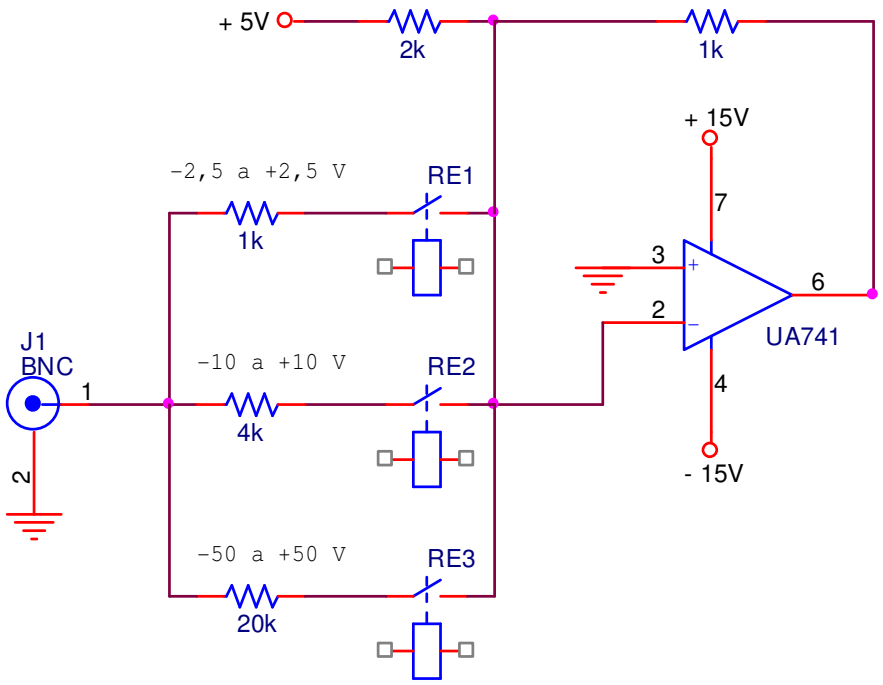


Ilustração 3-13 – Esquemático do ganho programável.

Com este mesmo esquema pode-se analisar sinais de grandezas de até ±250V apenas alterando o resistor de entrada para 100KΩ.

Na figura 3-14 observa-se como é feito o acionamento dos relés.

Os relés são alimentados em 12V e o acionamento é feito através de transistores que recebem o sinal vindos dos pinos do microcontrolador. Foram utilizados opto-acopladores para evitar interferências externas no circuito do PIC.

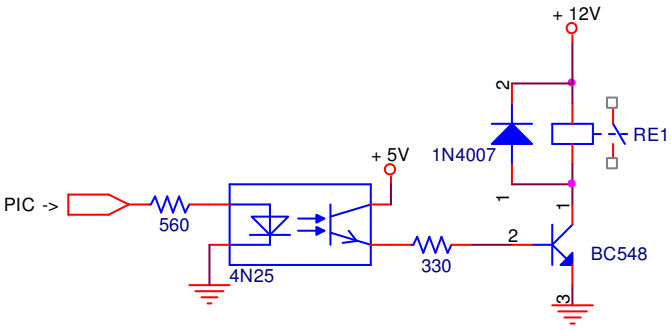


Ilustração 3-14 – Esquemático do acionamento dos relés.

3.3.3 Proteções e filtro RC

Para garantir que uma eventual sobre-tensão não cause danos ao pino do A/D e ao PIC, foram utilizados diodos de proteção para drenar a corrente nestes casos. Se ocorrer tensões acima de 5,7V o diodo ligado a VCC é polarizado (considerando uma queda de tensão no diodo de 0,7V). Caso a tensão for negativa, o diodo ligado no GND que será polarizado, conforme a figura 3-15.

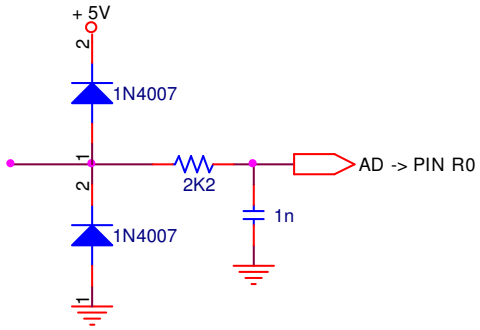


Ilustração 3-15 – Diodos de proteção e filtro RC.

Pode-se observar na figura anterior um filtro RC antes do sinal chegar ao conversor A/D. Este filtro passa-baixa foi projetado afim de impedir que ruídos com frequências muito acima das frequências analisadas interfiram nas medições.

O filtro RC é um filtro de 1ª ordem que atenua 6dB por década após a frequência de corte [2]. O calculo de sua frequência de corte é dado por:

$$f_c = \frac{1}{2\pi RC} \qquad (1)$$

Neste caso, foi utilizado um filtro RC com frequência de corte bem acima da frequência máxima de análise, através de um resistor de 2,2KΩ e um capacitor de 1nF obtém-se uma frequência de corte de aproximadamente 73 KHz.

3.3.4 Circuito completo

A figura 3-16 mostra o esquemático completo do circuito de condicionamento do sinal.

É importante salientar que devido a valores de resistências não comerciais foram necessárias associações de resistores na montagem do protótipo, afim de atender os ganhos simulados.

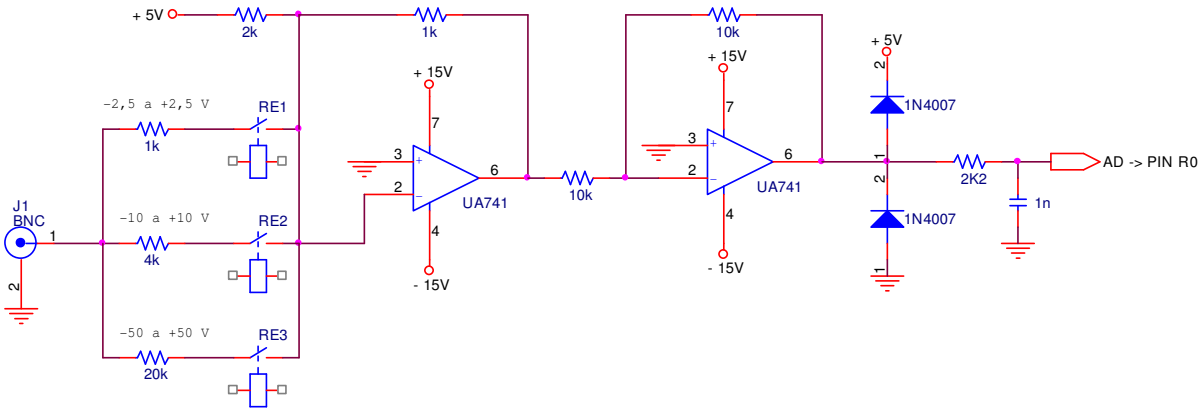


Ilustração 3-16 – Circuito completo de condicionamento.

A aquisição do sinal de entrada até a placa é feita através de uma ponteira-de-prova similar as utilizadas nos osciloscópios. Para isso foi instalado um conector tipo BNC fêmea, como pode-se observar na foto da figura 3-17.

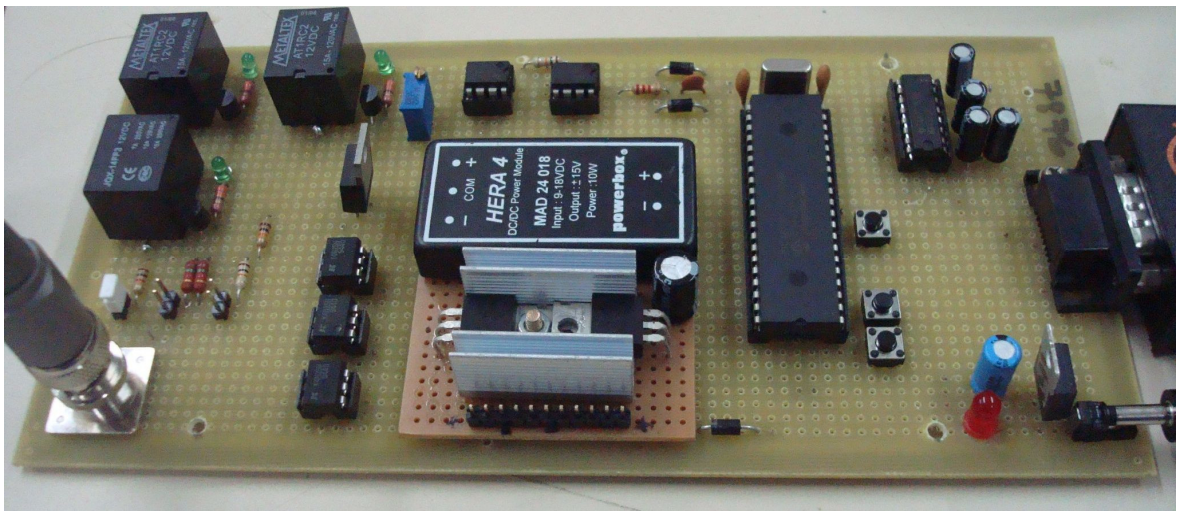


Ilustração 3-17 – Foto da placa de aquisição.

3.4 O programa do microcontrolador

O código fonte do PIC pode ser visualizado no Apêndice D, cujo fluxograma encontra-se na figura 3-18. No *main* do programa são definidas as configurações de trabalho do microcontrolador, são inicializadas as variáveis globais e o programa entra em *loop* infinito esperando que o buffer de leitura do conversor A/D seja preenchido para depois enviá-lo via interface serial ao PC. Ao final de cada envio de dados o microcontrolador recebe um caractere e avalia se houve ou não mudanças nos parâmetros.

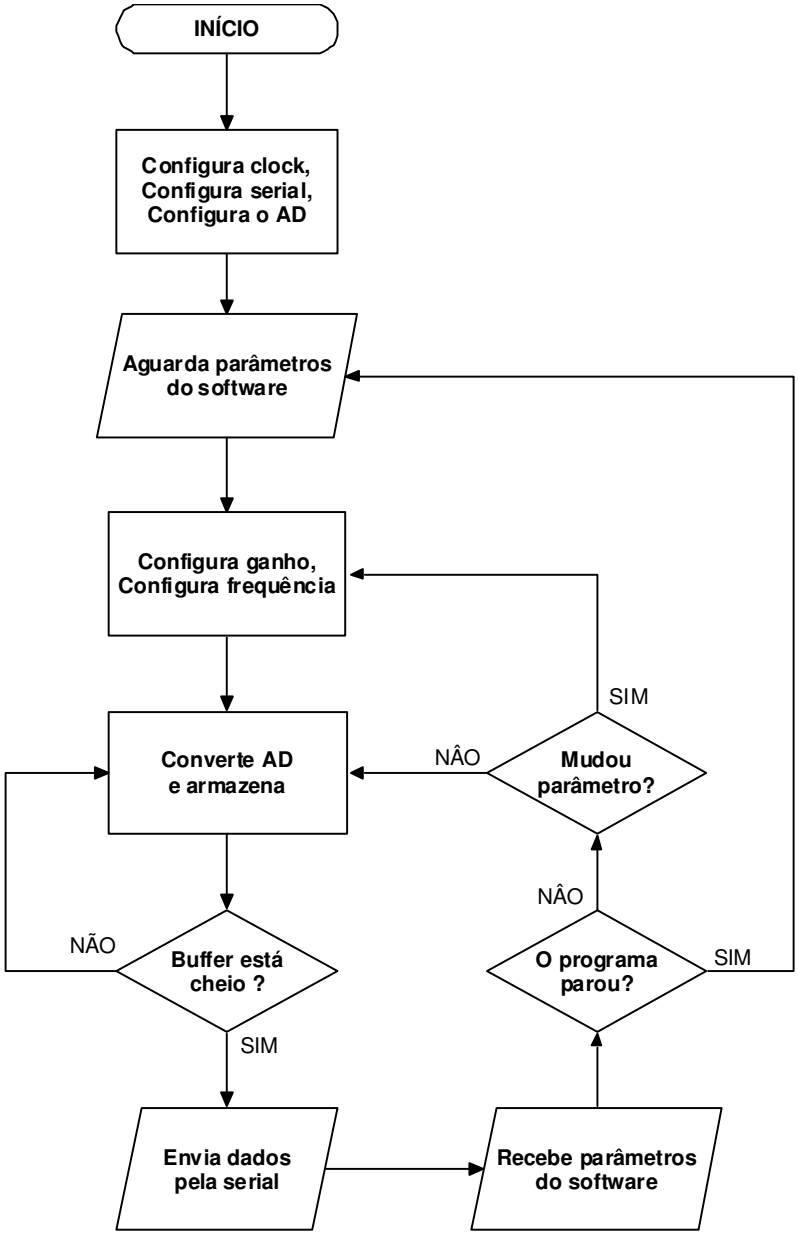


Ilustração 3-18 – Fluxograma do programa do PIC.

O conversor A/D foi configurado para 8 bits com o objetivo de facilitar e agilizar o envio do "pacote" de dados ao computador e também otimizar a quantidade de amostras armazenadas na memória RAM do PIC. O *buffer* de amostragem do A/D foi configurado para 240 amostras por ciclo (240 *bytes* de memória).

Uma dificuldade encontrada no *firmware* foi a manipulação do conversor A/D, pois o *datasheet* do PIC16F877A o descreve como capaz de realizar 50 mil amostras por segundo. Entretanto, na prática não foi possível fazê-lo trabalhar a mais que 28 mil amostras por segundo, devido a problemas de travamento da execução do código quando submetido a rotinas de armazenagem e incremento do buffer.

Um conceito importante que deve ser compreendido é como funciona a escala de leitura do sinal. O interesse de se implementar este artifício foi o de manter um número pequeno de pontos armazenados no buffer independente da frequência do sinal que se deseja visualizar.

O conversor AD do PIC no projeto está trabalhando a cerca de 28.000 amostras por segundo. Caso seja analisada uma onda de 1 kHz, significa que serão feitas aproximadamente 28 amostras por período de onda. Com um *buffer* de 240 pontos obtém-se 8 períodos do sinal por ciclo de programa, o que é o suficiente para tratar e apresentar na tela.

Contudo, caso o sinal em questão seja de baixa frequência, como 10 Hz, é necessário armazenar 2800 pontos para ver um período completo do sinal. O que não é tão simples. Isso exigiria *hardware's* mais potentes e *software's* otimizados.

Este problema é resolvido com o sistema de escalas. Se o usuário do *software* seleciona uma opção de grandeza do sinal de alta frequência, o *firmware* passa a armazenar todos os pontos lidos e transmiti-los ao PC, onde serão desenhados. Caso o usuário selecione opções de baixa frequência o *firmware* passa a armazenar no buffer de transmissão 1 ponto a cada 2 ms ou mais, dependendo da frequência do sinal.

3.5 O programa de computador

Todo software foi desenvolvido em linguagem C++, na plataforma Borland Builder C++. Para fazer a parte gráfica foi utilizada uma biblioteca gratuita chamada PlotLab fornecida em [3]. O código fonte completo e comentado do software do Osciloscópio via PC está no Apêndice E.

A interface final do Software ficou como mostrado na Figura 3-19.

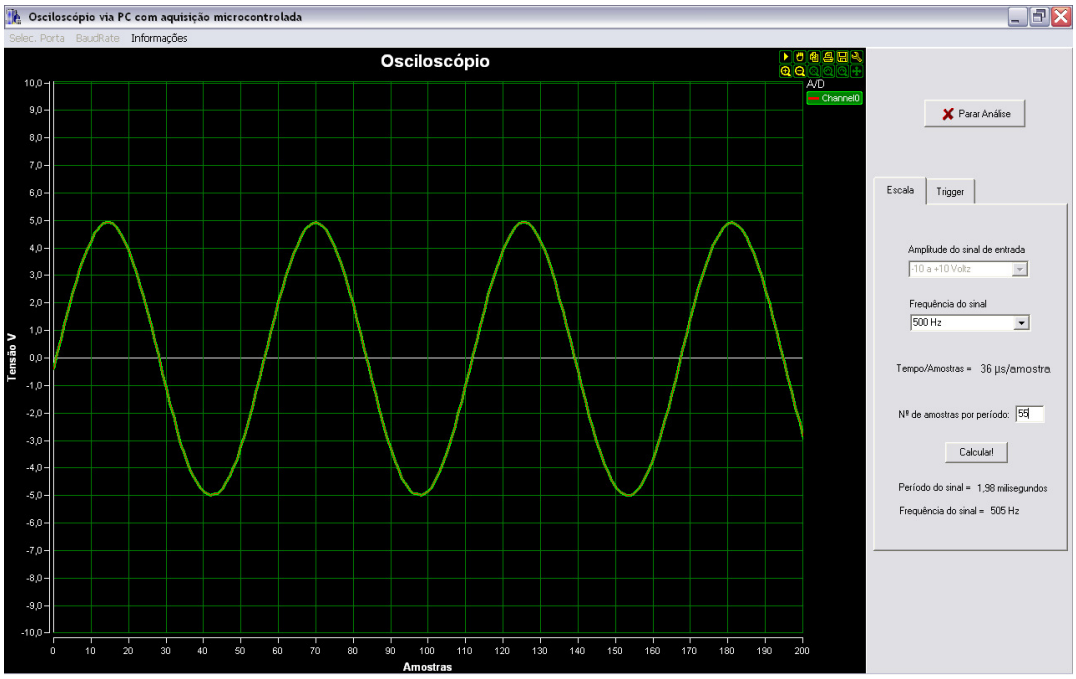


Ilustração 3-19 – Tela do programa de computador.

O fato de o número de pontos desenhados no gráfico ser menor que o número de pontos armazenados e enviados pelo microcontrolador está relacionado com a função de *trigger* via software. Maiores detalhamentos podem ser encontrados no Apêndice E.

A figura 3-20 mostra o fluxograma do programa a partir do *click* no botão "Iniciar Análise" no canto superior direito da tela. Uma das características deste programa é o uso de uma *Thread* (função de paralelismo) o que deixa o programa com uma resposta mais rápida ao comando do usuário. Enquanto a interface está sendo executada em primeiro plano a *Thread* está mantendo a comunicação pela serial com o microcontrolador em segundo plano.

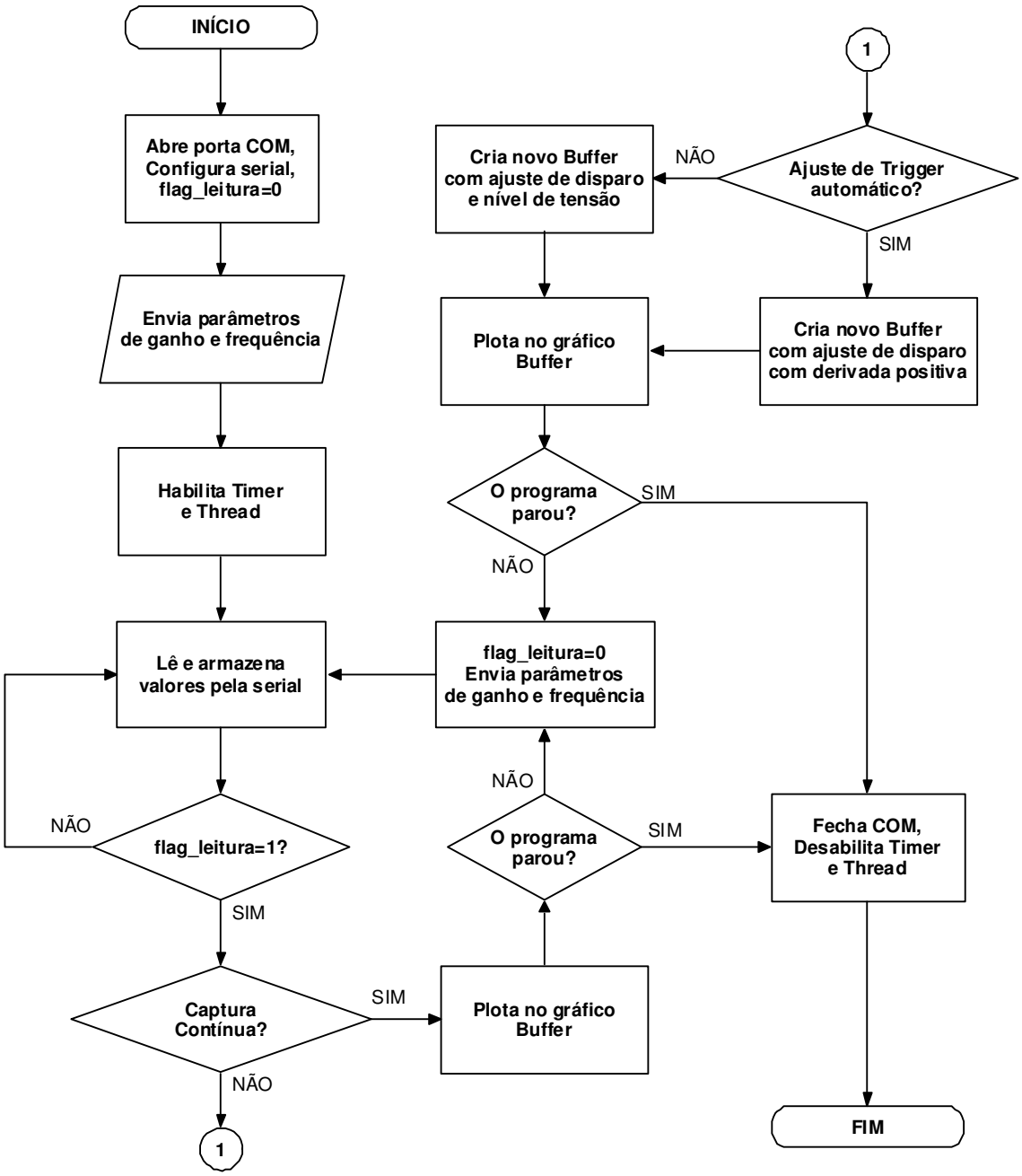


Ilustração 3-20 – Fluxograma do programa de computador.

3.5.1 Características do Gráfico

A biblioteca utilizada fornece inúmeras vantagens como a facilidade de programação, auto-zoom, impressão, salvar imagem do gráfico (ícone destacado na Figura 3-21), além de permitir o desenho de mais de um canal no mesmo gráfico. Porém, esta última ferramenta não foi utilizada visto que o projeto foi de um osciloscópio de apenas um canal.

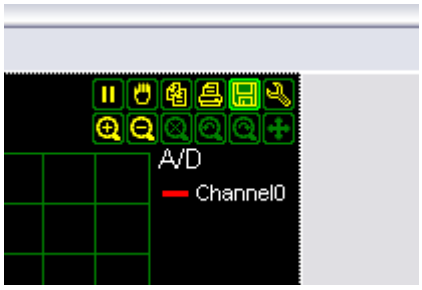


Ilustração 3-21 – Recursos da biblioteca gráfica.

3.5.2 Configurações do Osciloscópio via PC

As configurações necessárias para se fazer uma análise através da ferramenta projetada estão disponíveis no menu localizado na barra superior e na lateral direita.

Como a comunicação é Serial × USB, ao conectar a placa ao PC será criada uma ComPort Virtual para a troca de dados com a placa. Portanto, para utilizar o osciloscópio, deve-se inicialmente selecionar em qual ComPort está conectada a placa, conforme figura 3-22.

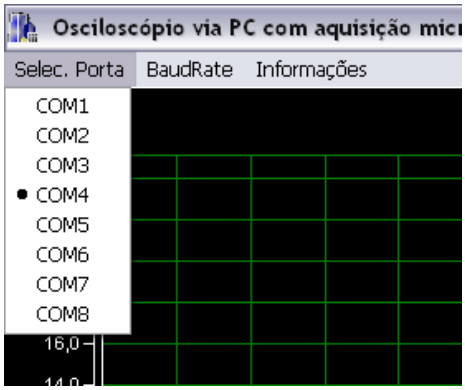


Ilustração 3-22 – Configuração da porta COM.

A velocidade de comunicação pode ser selecionada entre 4.800 e 115.200 bits por segundo, através do menu “BaudRate” conforme figura 3-23.

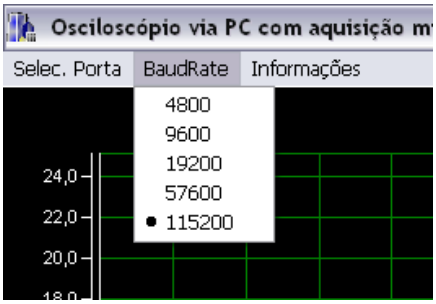


Ilustração 3-23 – Configuração da velocidade de comunicação.

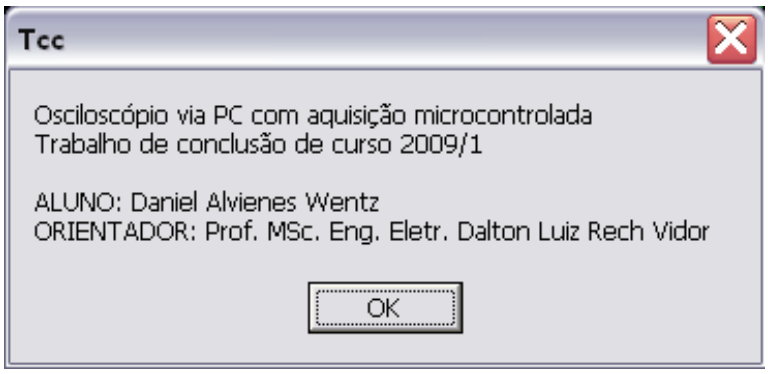


Ilustração 3-24 – Tela de informações do projeto.

As seleções de amplitude e de frequência do sinal de entrada são feitas na aba "Escala" no menu localizado à direita da tela como observar-se na figura 3-25.

Ilustração 3-25 – Aba de seleção de amplitude e frequência.

Para facilitar a leitura do período e da frequência do sinal amostrado existe na parte de baixo da aba "Escala" um botão de calculo, onde o usuário entra com a quantidade de amostras em um período na caixa de texto mostrada na figura 3-26 e o programa calcula levando em consideração o tempo/amostra.

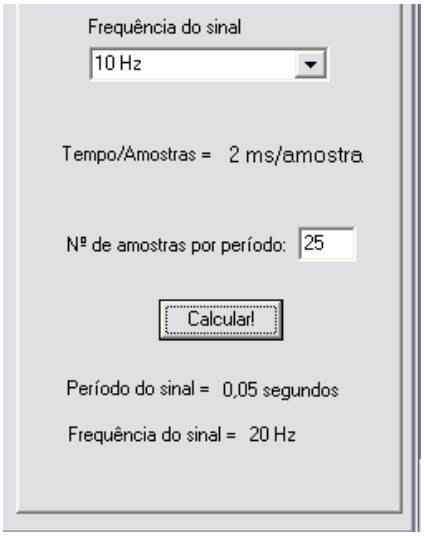


Ilustração 3-26 – Botão de calculo do período e frequência.

Na aba "Trigger", existem três opções de captura passíveis de seleção, como se verifica na figura 3-27:

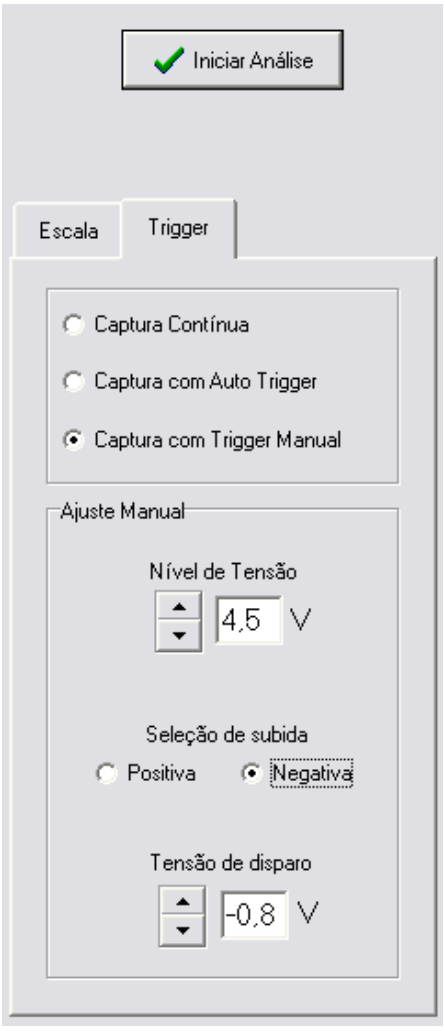


Ilustração 3-27 – Aba de ajuste do *Trigger*.

Os três modos estão descritos a seguir:

- O modo de captura contínua desenha o conteúdo lido do ADC na tela sem passar por nenhum algoritmo para 'gatilhar' a onda em momentos iguais. Portanto, disponibiliza-se um sinal atualizado na tela, porém sem nenhuma sequência.
- A captura com Auto *Trigger* mantém a tela sempre com um sinal atualizado, mas esse sinal passa por um algoritmo que sempre tenta iniciar o desenho da onda a partir do primeiro ponto onde o sinal cruza o seu valor médio com derivada positiva.
- O *Trigger* Manual se assemelha ao Auto *Trigger*, porém com opções de disparo com derivada negativa, nível de tensão e recurso de deslocar a onda no gráfico, iniciando o desenho pelo valor de tensão contido na caixa de texto logo abaixo.

4 RESULTADOS

Para levantar os resultados finais do projeto foram efetuados testes no laboratório da universidade. Os itens avaliados foram : precisão na amplitude, ajuste de *trigger* e desempenho com diferentes frequências.

Foi utilizado para comparar os valores obtidos um osciloscópio de bancada da marca Tektronix modelo “TAS 465” de 100Mhz. Para criar o sinal foi utilizado um gerador de funções também da Tektronix, modelo “CFG 280”. A figura 4-1 mostra uma foto dos testes em bancada.

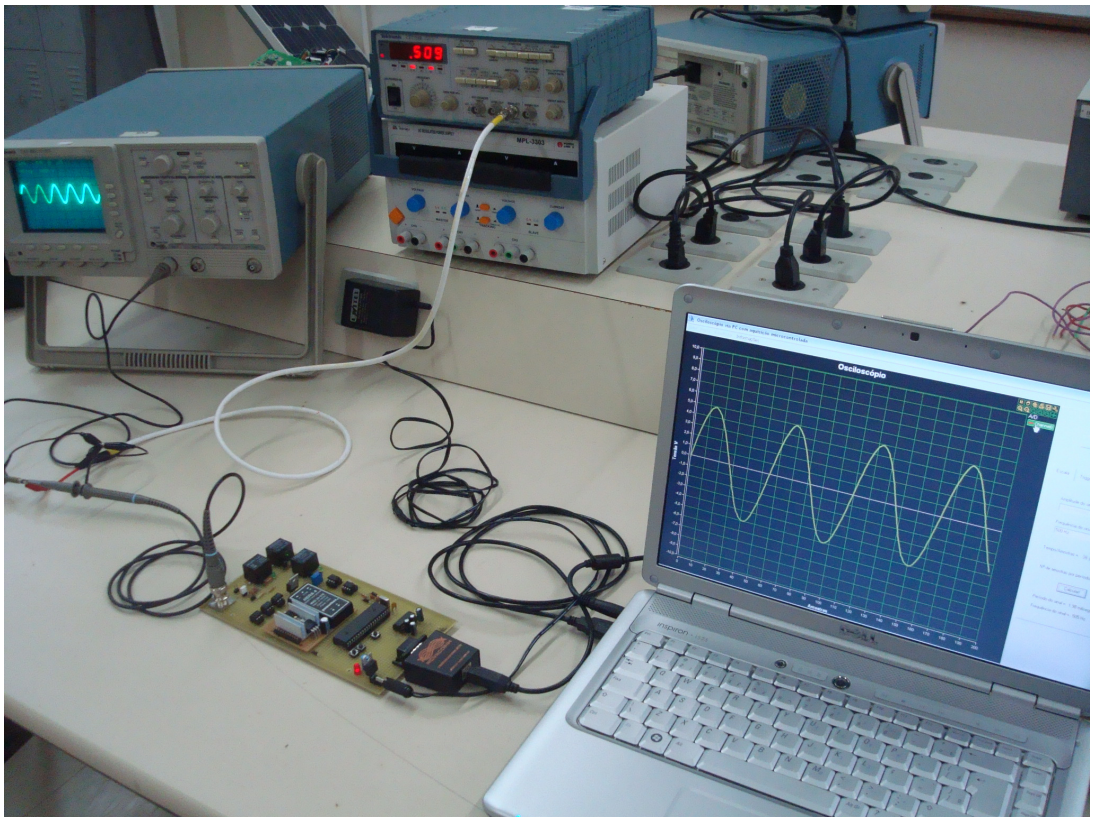


Ilustração 4-1 – Foto: Levantamento dos resultados em bancada.

4.1 Análise de precisão da amplitude

Para analisar a precisão do circuito de entrada para os três diferentes ganhos foi utilizado o gerador de funções no modo senoidal com frequência de 500Hz.

A amplitude máxima do gerador disponível é de 20V pico-á-pico (Vpp), sendo esta amplitude utilizada para os testes nos circuitos de ±50V e ±10V. A figura a seguir mostra o sinal enviado a placa pelo gerador.

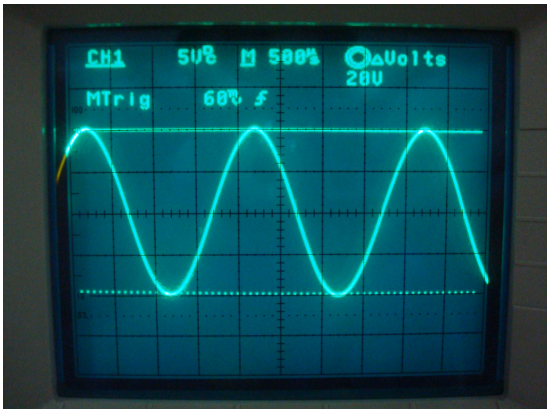


Ilustração 4-2 – Sinal gerado com amplitude de 20 Vpp.

Na figura 4-3, observa-se o sinal já apresentado na tela do computador através do circuito de ±50V. Os gráficos foram negativados em tons de cinza para proporcionar uma melhor visualização.

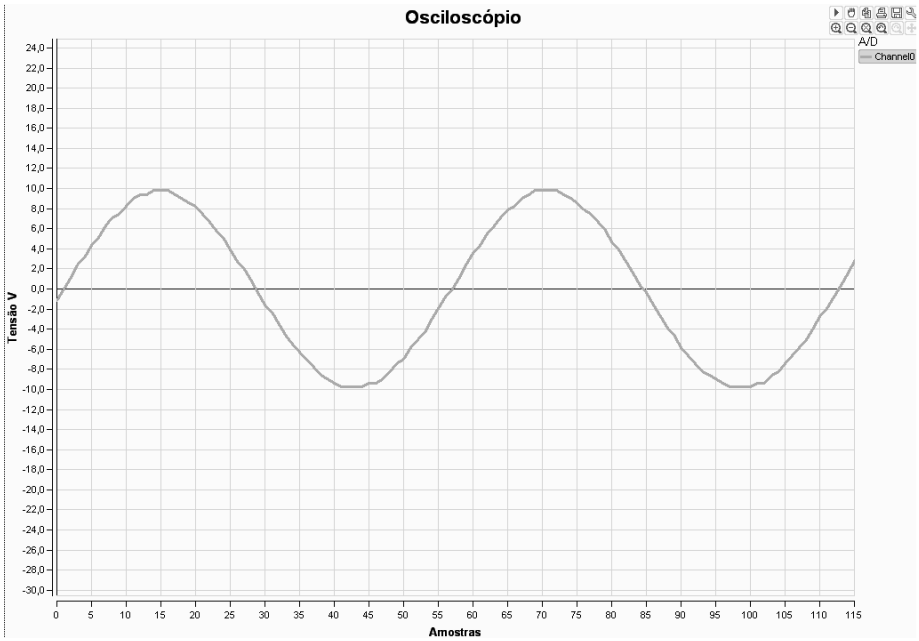


Ilustração 4-3 – Sinal analisado - Entrada ±50V.

A próxima figura mostra o mesmo sinal, porém analisado pelo circuito de ±10V.

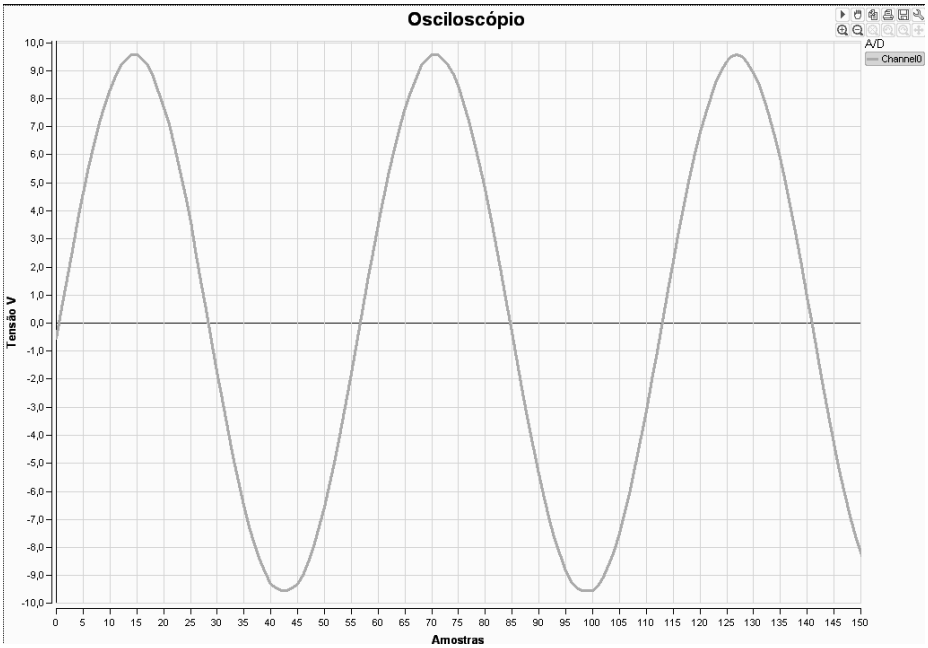


Ilustração 4-4 – Sinal analisado - Entrada ±10V.

Pode-se observar um erro nos finais de escala positiva e negativa de aproximadamente 2,5%. Este erro é gerado porque na criação do circuito de ganho programável não foi levado em consideração a resistência da ponta-de-prova que é de aproximadamente 260Ω.

Essa condição não é percebida no teste anterior devido a grande diferença da resistência de entrada (20K Ω) em relação a resistência da ponteira. Já o menor ganho (±2,5V) irá sofrer maior influência dessa resistência em série devido ao valor do resistor de entrada ser apenas 4 vezes maior que o da ponta-de-prova. Pode-se observar através da figura 4-6 um erro de 8% para um sinal gerado de 5Vpp conforme figura 4-5.

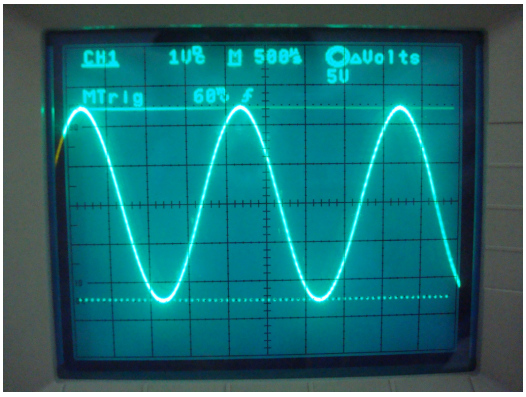


Ilustração 4-5 – Sinal gerado com amplitude de 5 Vpp.

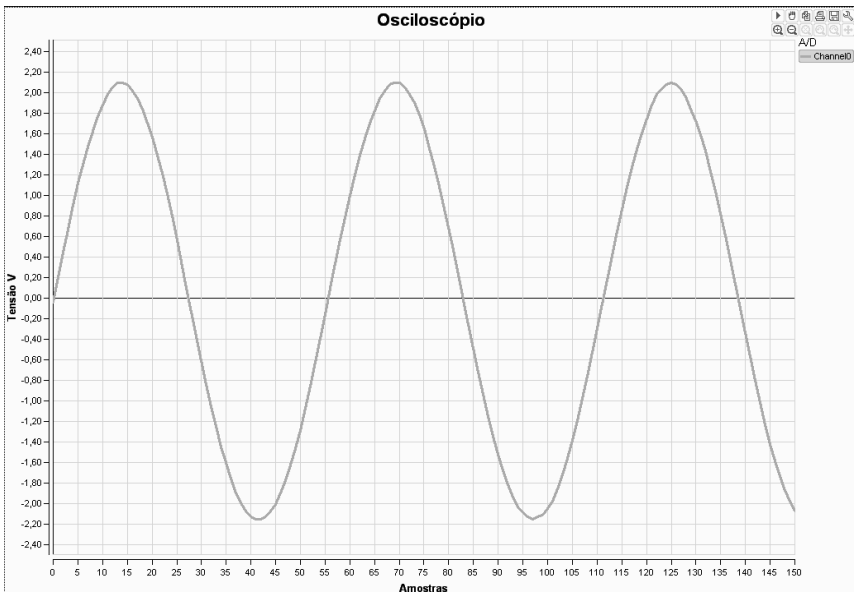


Ilustração 4-6 – Sinal analisado - Entrada ±2,5V.

4.2 Análise dos recursos de Trigger

Através de uma onda senoidal de 500Hz com amplitude de 14Vpp gerada pelo gerador de funções foram analisados os recursos do programa quanto ao ajuste de *Trigger* e nível de tensão.

Primeiramente verificou-se o comportamento do gráfico com a seleção de "Auto *Trigger*", onde o gatilho é feito pelo valor médio do sinal com derivada positiva, representado na figura 4-7.

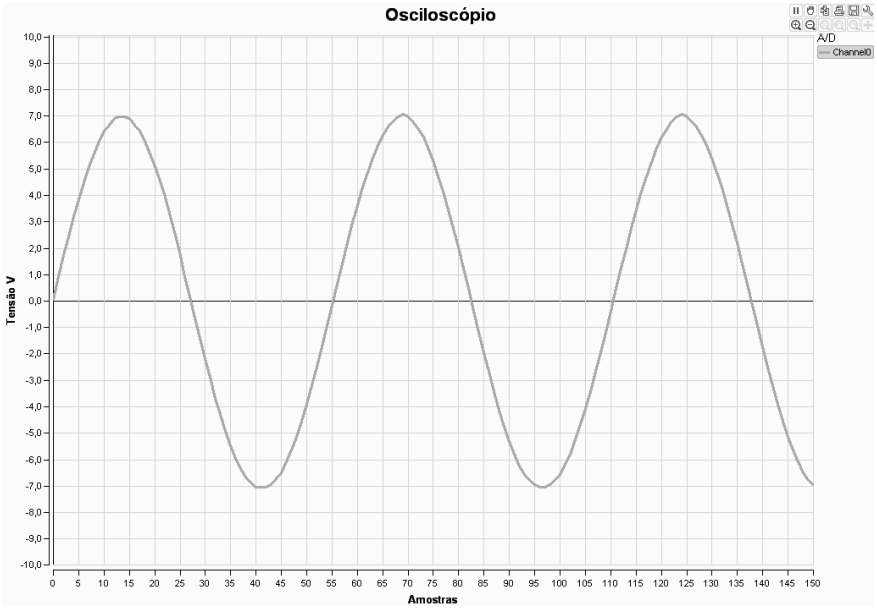


Ilustração 4-7 – Captura com Auto *Trigger.*

Uma das opções de seleção do *Trigger* Manual é o disparo com derivada negativa , apresentado na figura 4-8.

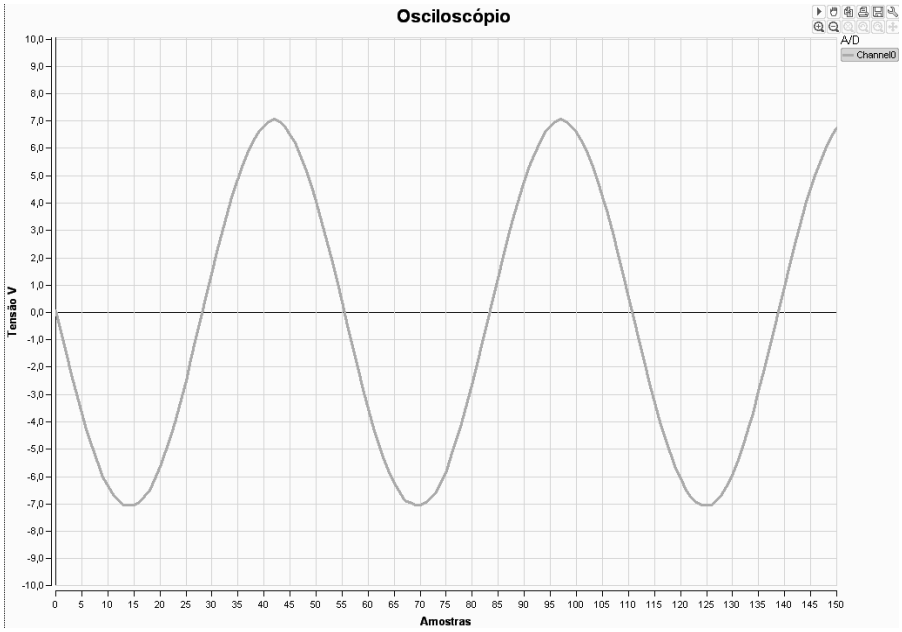


Ilustração 4-8 – Captura com *Trigger* Manual e disparo com derivada negativa.

Na figura 4-9 é demonstrado o recurso de deslocamento da onda através de um nível de tensão. No primeiro caso foi deslocado para baixo -3V e no segundo, para cima +3V.

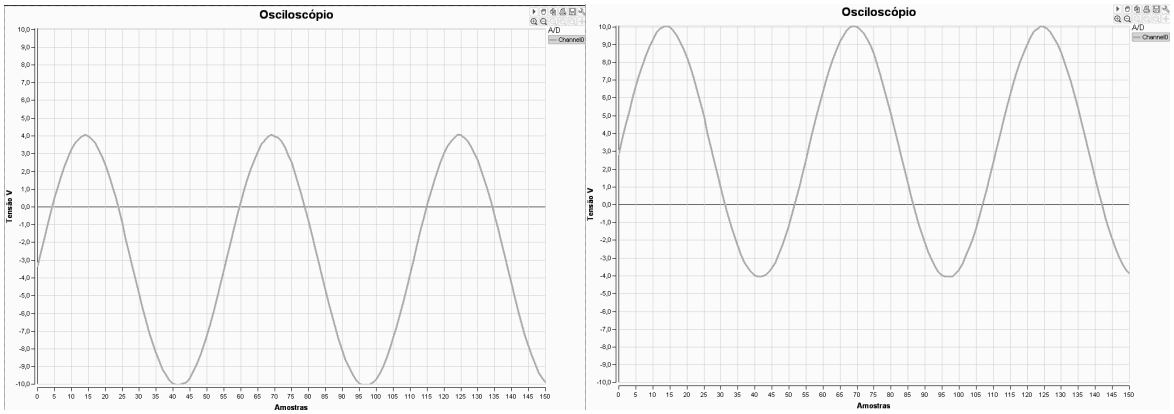


Ilustração 4-9 – Captura com *Trigger* Manual com deslocamento de nível.

Outro teste efetuado foi com o recurso de deslocamento horizontal da onda através de uma tensão de gatilho selecionada pelo usuário. Este teste está demonstrado na figura 4-10 com valor de disparo negativo (-5V) e na figura 4-11 com valor de disparo positivo (+5V). Ambos os casos estão apresentados com derivada positiva e negativa.

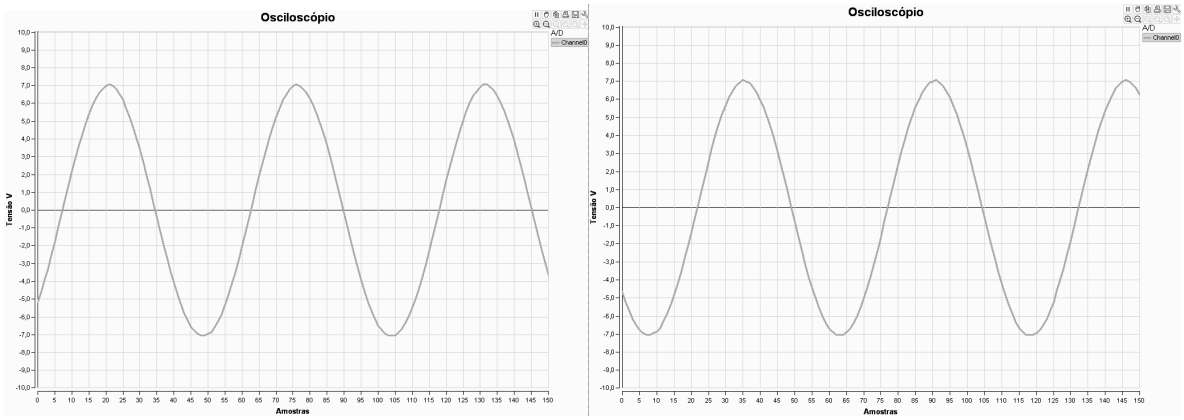


Ilustração 4-10 – Gatilho em -5V com derivada positiva e negativa

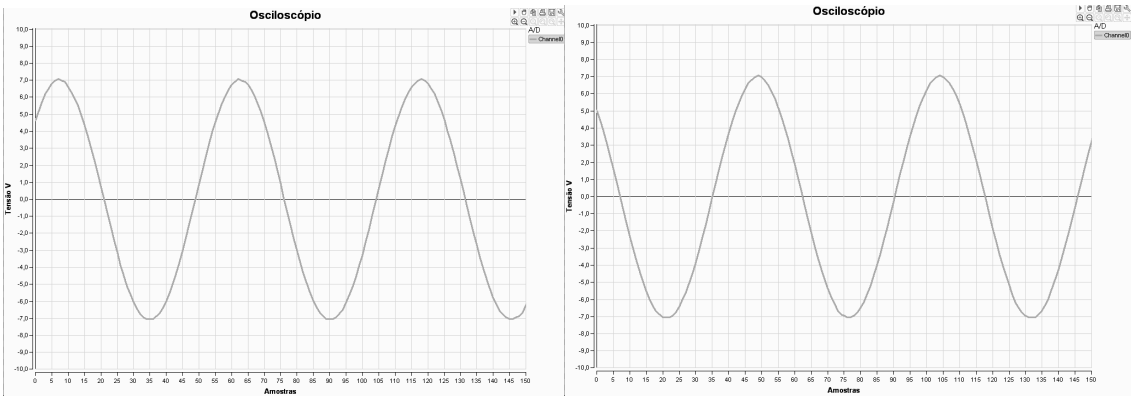


Ilustração 4-11 – Gatilho em +5V com derivada positiva e negativa

4.3 Análise de resposta em diferentes frequências

Para os testes de desempenho com diferentes frequências foram analisados sinais senoidais, triangulares e retangulares com amplitude de 10Vpp.

4.3.1 Onda senoidal

Inicialmente, foi aplicado uma onda de 1Hz para teste com frequências extremamente baixas. O resultado apresentado na tela do computador foi de uma forma de onda bastante simétrica (figura 4-12).

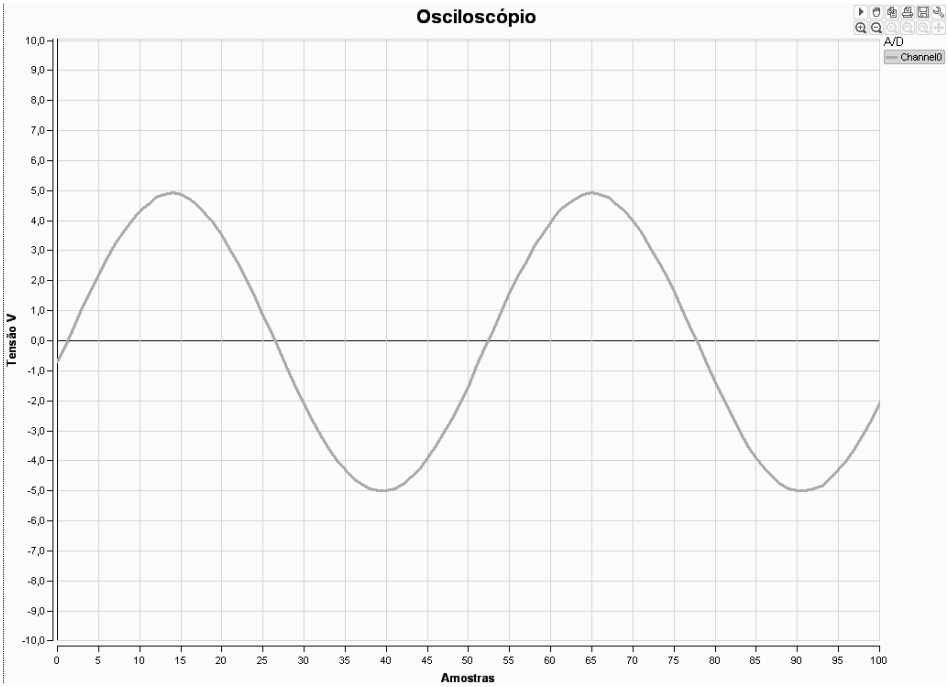


Ilustração 4-12 – Sinal analisado – onda senoidal de 1Hz.

Esta visualização se torna difícil em osciloscópios analógicos por se tratar de um sinal com período de tempo elevado, o que não permite a continuidade no traço da forma de onda na tela. Este efeito é característica dos raios catódicos, visto no capítulo 2. Na figura 3-13 é apresentado o resultado para um sinal de senoidal de 10Hz.

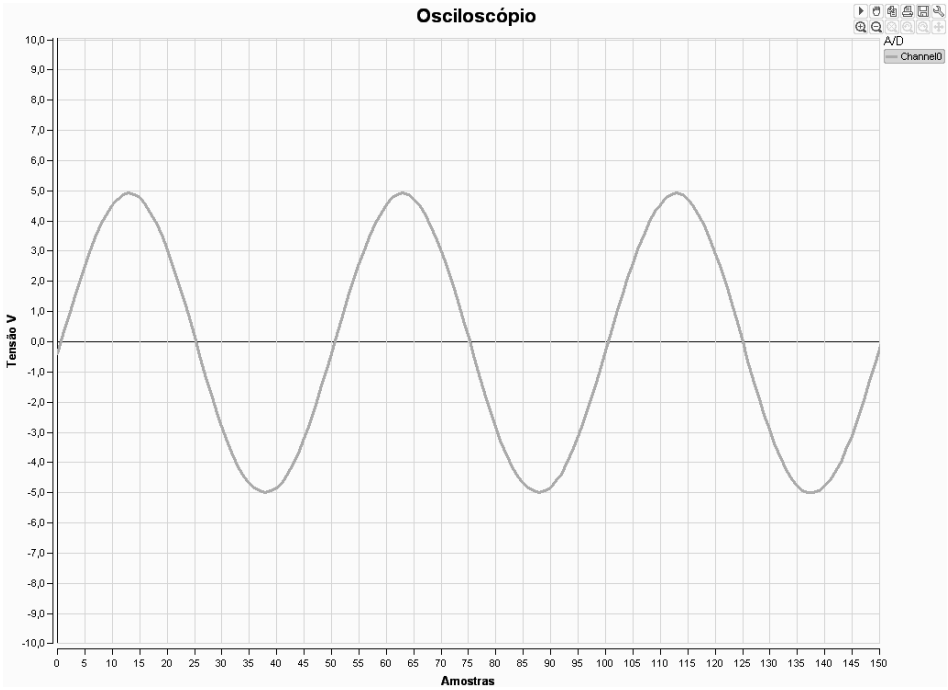


Ilustração 4-13 – Sinal analisado - onda senoidal de 10Hz.

Nas imagens 4-14 e 4-15 pode-se observar a resposta à sinais de 500Hz e 1KHz, apresentando uma boa visualização do senoide gerada.

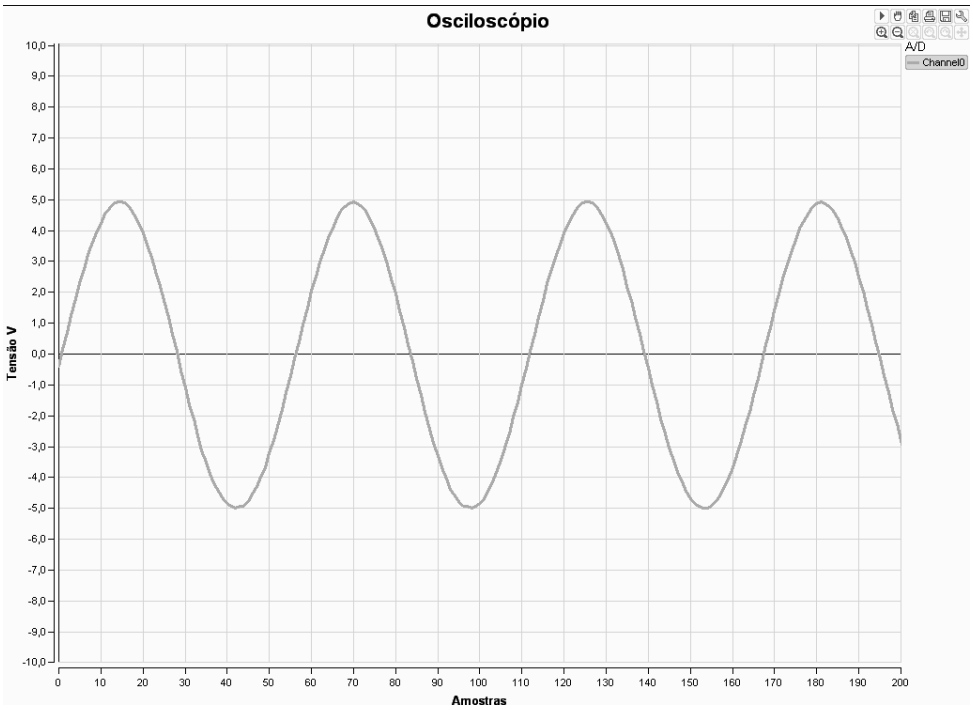


Ilustração 4-14 – Sinal analisado – onda senoidal de 500Hz

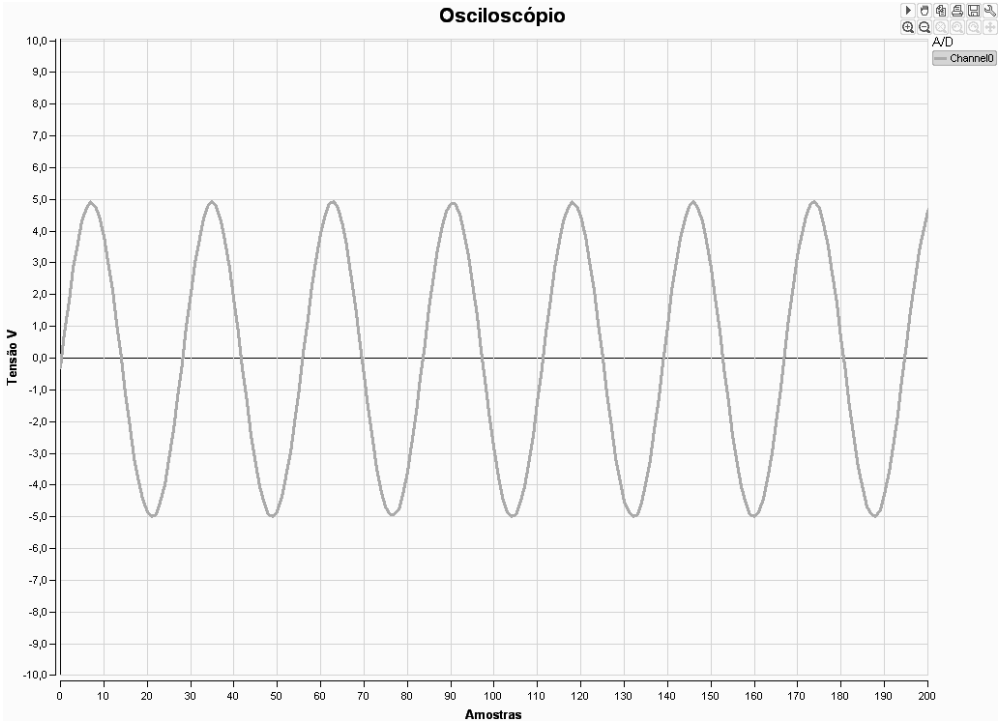


Ilustração 4-15 – Sinal analisado – onda senoidal de 1KHz

A próxima imagem é referente a uma frequência de 2KHz:

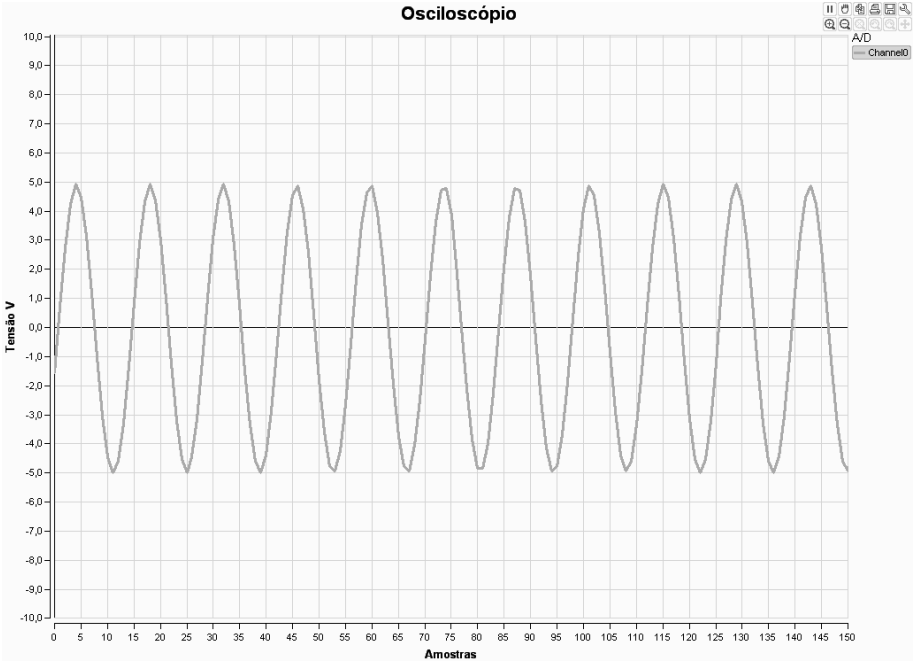


Ilustração 4-16 – Sinal analisado – onda senoidal de 2KHz

Nota-se uma perda de resolução a partir do sinal de 1KHz decorrente de uma diminuição no número de amostras por período, indicando um limite máximo de frequência para uma boa qualidade de visualização. Observa-se que em um sinal de 2KHz (figura 4-16) o número de amostras por período já é de 14, quando o ideal serial no mínimo 20. Esta diminuição de amostras/período se torna mais clara na figura 4-17, onde é muito difícil de identificar uma senoide com frequência de 5KHz.

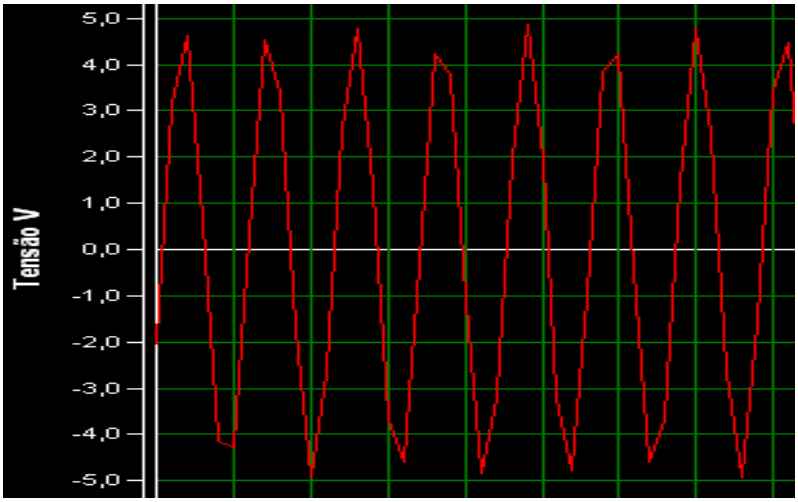


Ilustração 4-17 – Sinal analisado – onda senoidal de 5KHz

4.3.2 Onda triangular

Os testes também foram feitos com ondas triangulares e apresentaram resultados semelhantes às senoidais, onde sinais de até 1KHz apresentaram uma boa qualidade na imagem gráfica.

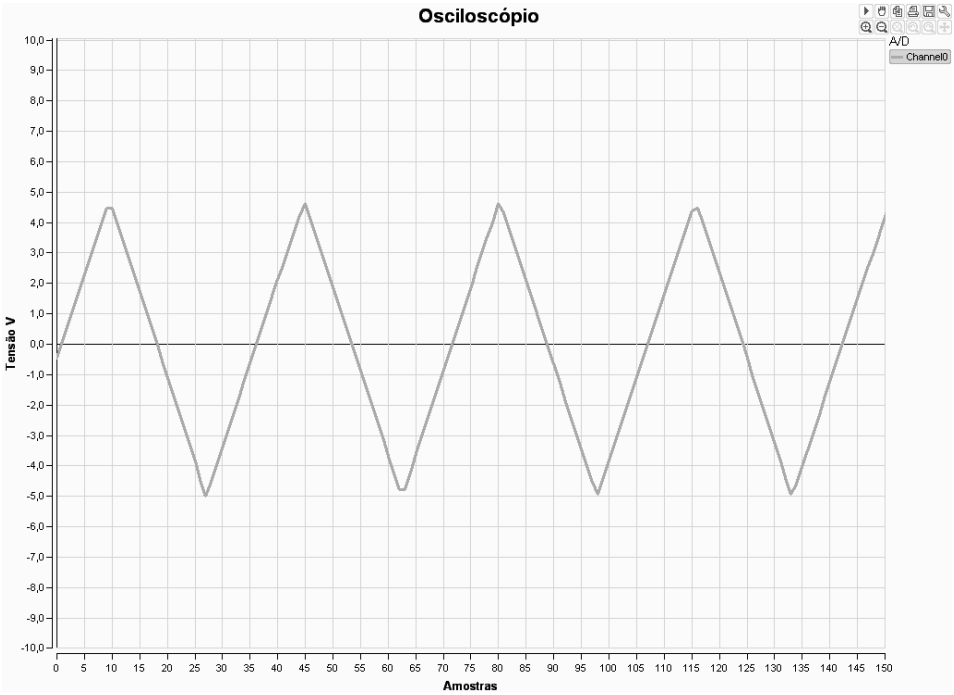


Ilustração 4-18 – Sinal analisado – onda triangular de 100Hz

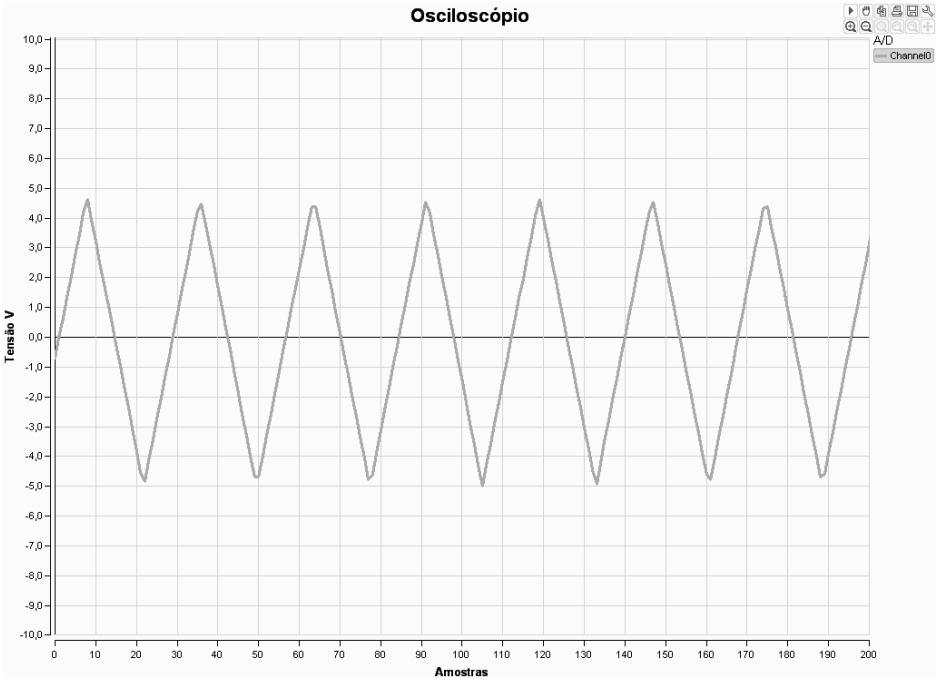


Ilustração 4-19 – Sinal analisado – onda triangular de 1KHz

4.3.3 Onda retangular

Finalmente, em sinais retangulares o Osciloscópio via PC também obteve um bom desempenho para frequências de até 1KHz. Porém apresentou uma pequena deformação na linha de borda superior, como pode-se observar nas figuras a seguir.

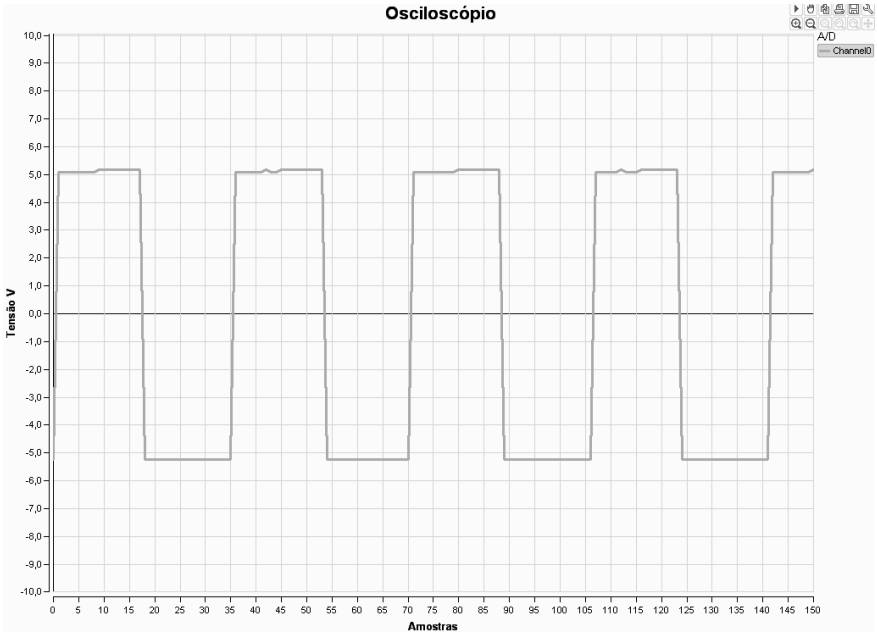


Ilustração 4-20 – Sinal analisado – onda retangular de 100Hz

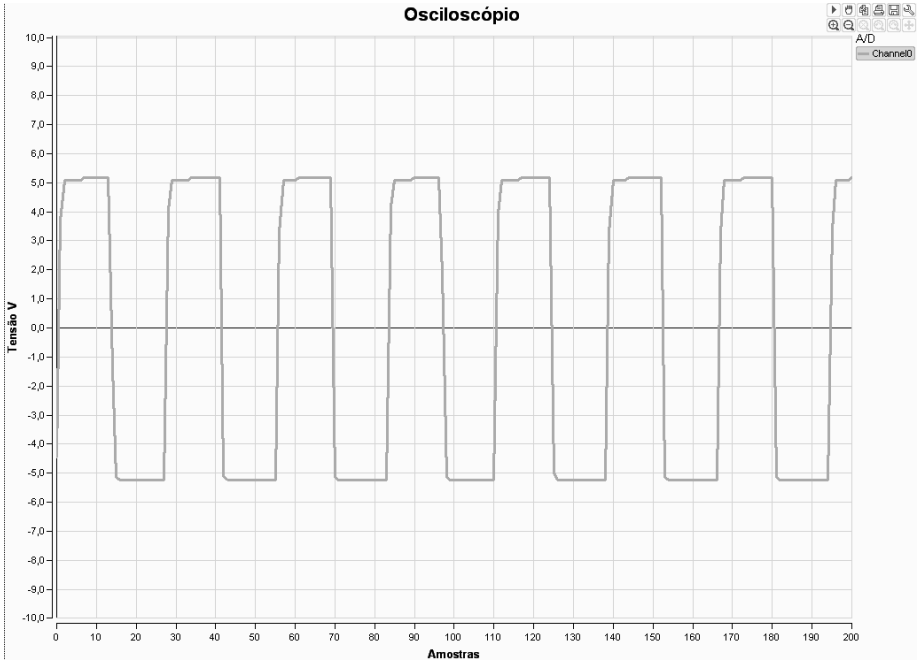


Ilustração 4-21 – Sinal analisado – onda retangular de 1KHz

5 CONCLUSÕES FINAIS

O objetivo inicial de se projetar uma ferramenta de análise como um osciloscópio para aquisição e apresentação de sinais no tempo foi alcançado. Através dos testes com o protótipo implementado, verificou-se que todos os recursos propostos inicialmente foram concluídos, tais como: ganho variável no sinal de entrada com seleção por *software*; funções de auto *trigger* e de ajuste manual de disparo com derivada positiva ou negativa; e recursos de deslocamento da onda na tela do computador tanto na horizontal como na vertical.

As simulações por computador mostraram que com a mesma configuração de *hardware* é possível alterar as escalas de amplitude apenas trocando o resistor de entrada, podendo facilmente configurar o Osciloscópio via PC para amplitudes como ±1V ou ±200V.

Os resultados finais indicaram uma frequência máxima de 1KHz para se obter uma boa visualização da forma de onda de um sinal senoidal utilizado como parâmetro. Este desempenho ficou abaixo do esperado que era de se analisar frequências de até 5KHz. Alguns fatores colaboraram para isso, como o retardo entre as amostragens sucessivas do conversor A/D gerado pelas instruções do compilador.

Por outro lado, se pensarmos em um sistema de aquisição como este sendo utilizado para análise de harmônicas em redes elétricas de 60Hz, seria possível observar até a vigésima harmônica com um bom grau de precisão. Na qüinquagésima harmônica ainda seriam lidos aproximadamente 10 pulsos em um período, o que é razoável para uma análise como esta.

Para uma futura versão do programa de computador, alguns recursos podem ser implementados a partir do *software* atual. Cálculos automáticos de grandezas como frequência, período, tensões de pico e valor eficaz do sinal estão entre as melhorias possíveis, além da incorporação de um analisador de espectro.

Em relação a melhorias no *hardware*, é imprescindível a utilização de um conversor AD de maior velocidade para análises de frequências altas. Alguns microcontroladores já possuem opção de comunicação serial via USB, trabalhando com taxas elevadas de velocidade na troca de informações com o PC proporcionando uma possível aquisição em tempo real.

Por fim, a criação de mais canais de entrada e a opção de seleção para acoplamentos AC/DC estão disponíveis como sugestões para um próximo protótipo.

6 REFERÊNCIAS

[1] ALMEIDA, AILSON ROSETTI DE. Conversores Digital/Analógicos (DAC) e Analógico/Digitais (ADC). [on line]. Disponível: http://www2.ele.ufes.br/~ailson/digital2/adda.pdf

[2] CARVALHO, ROGÉRIO MUNIZ. Princípios de Comunicações - 3ª Ed.- Vitória, ES. 2003.

[3] MITOV SOFTWARE. PlotLab 4.0. [on line]. Disponível: http://www.mitov.com/html/download_plotlab.html [capturado em 5 mar. 2009]

[4] COCIAN, Luis Fernando Espinosa. Modelo de Monografia de TCC em EE, Revisão G.

[5] PEREIRA, Fábio. Microcontroladores Pic – Programação em C - 7ª Ed. - São Paulo: Editora Érica Ltda, 2007.

[6] SCHILDT, Herbert. Borland C++ - Completo e Total - 5ª Ed. - São Paulo: Editora Makron Books, 1996.

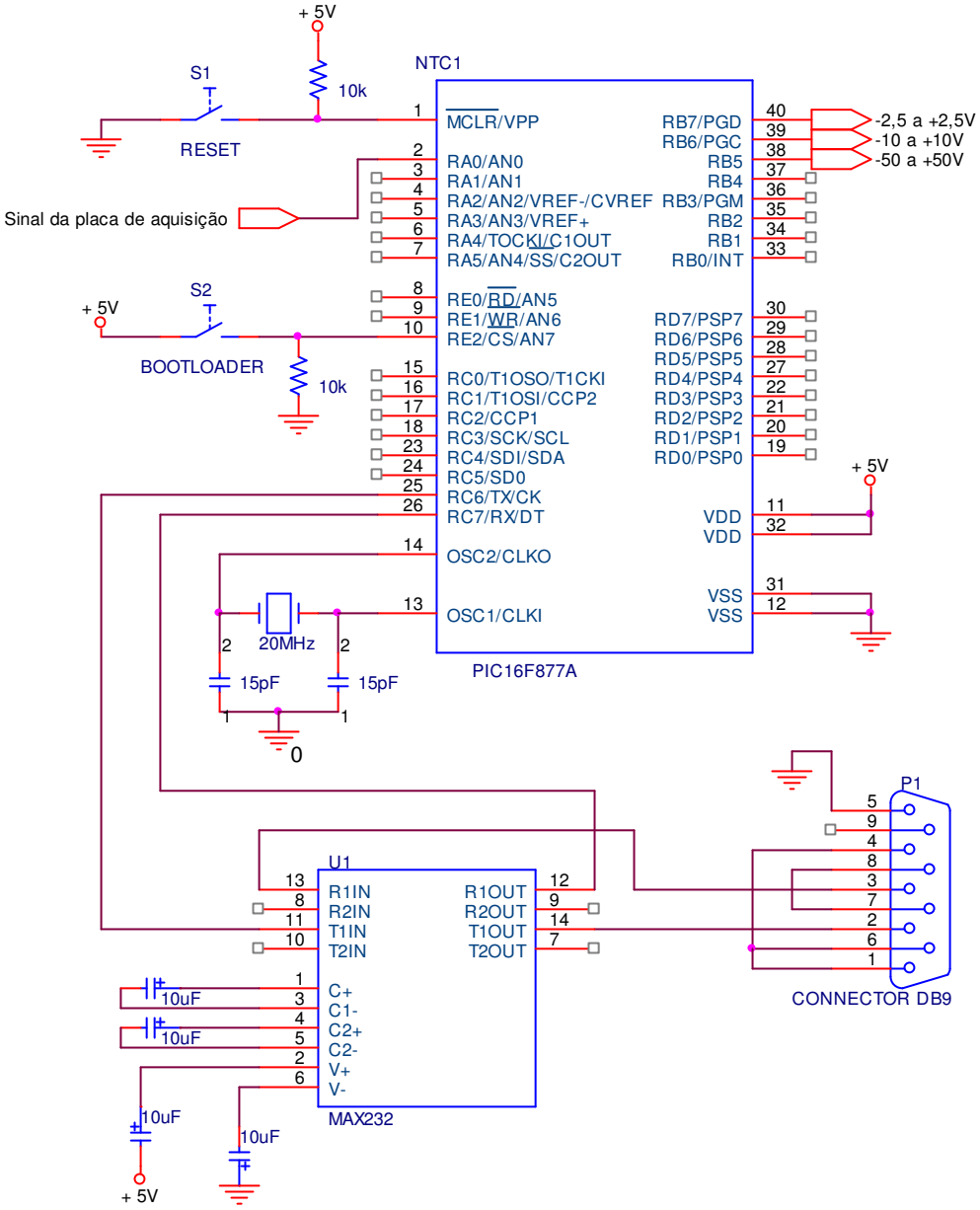
[7] MICROSHIP - PIC16F877A, Data Sheet. Disponível: http://www.datasheetcatalog.com

[8] UNSER, M. Sampling-50 years after Shannon, Proceedings of the IEEE, Vol. 88, n. 4, Abril 2000, pp. 569-587.

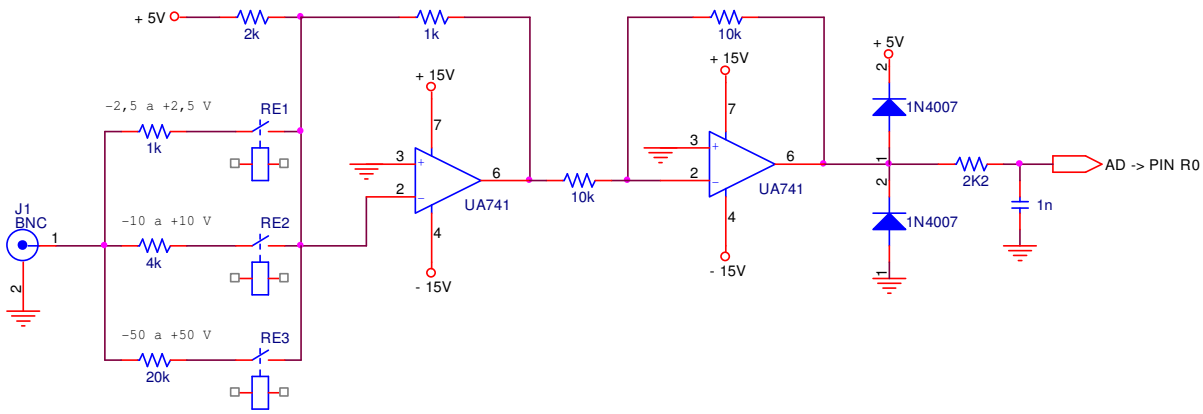
[9] CAZIAN, Edmur. Minicurso – Comunicação Serial RS-232. Disponível: http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf

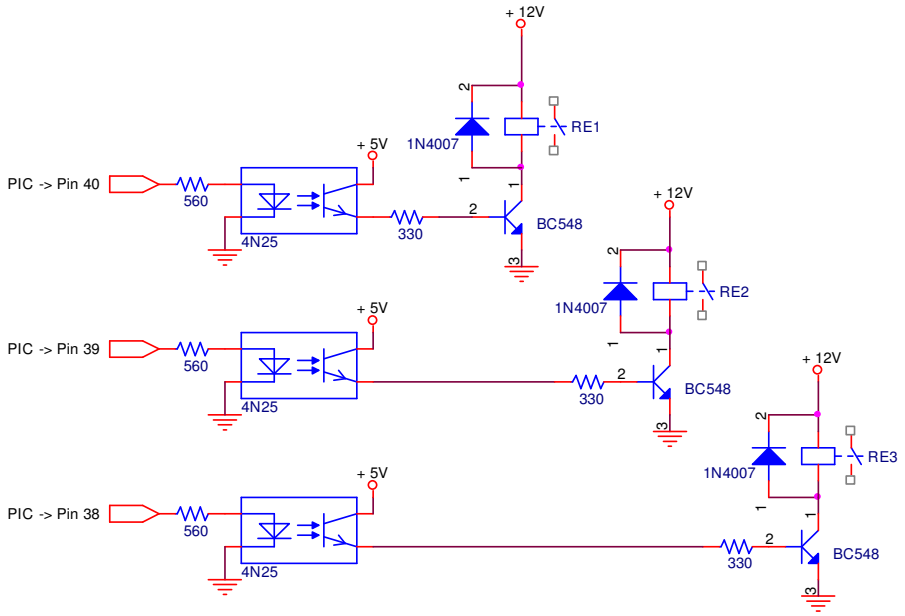
[10] BONFIM, Marlio. Introdução ao Osciloscópio. Disponível: http://www.eletrica.ufpr.br/marcelo/TE042/experimentos/exp9.pdf

APÊNDICE A – ESQUEMÁTICO - CIRCUITO DO MICROCONTROLADOR



APÊNDICE B – ESQUEMÁTICO DO CONDICIONAMENTO DO SINAL





APÊNDICE C – TABELA DE CUSTOS

Componente	Especificação	Quant.	Preço unid.	Sub Total
Microcontrolador	PIC16F877A	1	R$ 24,00	R$ 24,00
RS232 buffer de linha	MAX232	1	R$ 3,50	R$ 3,50
Cristal	20 MHz	1	R$ 2,00	R$ 2,00
Amp. Operacional	LM741	2	R$ 1,20	R$ 2,40
Opto-acoplador	4N25	3	R$ 1,90	R$ 5,70
Transistor	BC-548	3	R$ 0,20	R$ 0,60
Relé	120V - acionamento 12V	3	R$ 3,00	R$ 9,00
Conector	BNC p/ placa	1	R$ 7,20	R$ 7,20
Conector	DB-9 fêmea	1	R$ 2,50	R$ 2,50
Diodo	1N4007	5	R$ 0,40	R$ 2,00
Reg. Tensão	7805	1	R$ 1,00	R$ 1,00
Reg. Tensão	7812	1	R$ 1,00	R$ 1,00
Reg. Tensão	7815	2	R$ 1,00	R$ 2,00
Cap. Eletrolítico	10uF	4	R$ 0,35	R$ 1,40
Cap. Cerâmica	1nF, 15pF	3	R$ 0,30	R$ 0,90
Resistores	330, 560, 1K, 2K2, 10K	15	R$ 0,10	R$ 1,50
Potenciômetro	5K - multivoltas	1	R$ 3,00	R$ 3,00
Botão de pressão	Normal aberto	2	R$ 1,50	R$ 3,00
Placa	Padrão	1	R$ 10,00	R$ 10,00
			TOTAL	**R$ 82,70**

APÊNDICE D – CÓDIGO FONTE DO MICROCONTROLADOR

```
#include <16F877A.h>
#device ADC=8
#use delay(clock=20000000)
#fuses HS,NOWDT,NOPUT,NOBROWNOUT,NOLVP,NOPROTECT
#use rs232(baud=115200, xmit=PIN_C6, rcv=PIN_C7)

int buffer1[50],buffer2[50], buffer3[50], buffer4[50], buffer5[40];
int j;
long int tempo;
char c, escala, amplitude;

main()
{

output_low (PIN_B5);          //-50 a +50 V
output_low (PIN_B6);          //-10 a + 10V
output_low (PIN_B7);          //-2,5 a +2,5 V

  setup_port_a( AN0 );
  setup_adc( ADC_CLOCK_INTERNAL );
  set_adc_channel( 0 );
  tempo = 0;

while (true)
{

        escala=getc();                  //Parâmetro de escala (0 à 5)
        c=getc();

                if (escala=='x')        //Se receber 'x' processo parou!
                {
                output_low (PIN_B5);
                output_low (PIN_B6);
                output_low (PIN_B7);

                //Aguarda reinício com parâmetro de escala (0 à 5)
                escala=getc();
                c=getc();
                }

        amplitude=getc();               //Parâmetro de amplitude (0 à 2)
        c=getc();
```

```
//_____________________________________________________________
//Configura ganho
if(amplitude=='0')          output_high (PIN_B7);//-2,5 a +2,5 V

else if(amplitude=='1')     output_high (PIN_B6);//-10 a +10 V

else if(amplitude=='2')     output_high (PIN_B5);//-25 a +25 V

//_____________________________________________________________
//Configura tempo entre amostras

if(escala=='0')             tempo=20;          //1Hz - 10ms/amostra

else if(escala=='1')        tempo=2;           //10Hz - 2ms/amostra

else if(escala=='2')        tempo=250;         //100Hz - 285us/amostra

else if(escala=='3')        tempo=0;           //500Hz - 36us/amostra

else if(escala=='4')        tempo=0;           //1KHz - 36us/amostra

//_____________________________________________________________
//Se a escala for de 500Hz ou 1Kz entra na rotina de 36us

if (escala=='3' || escala=='4')
{
        for(j=0; j<60; j++)
        {
        buffer1[j]=read_adc();
        }

        for(j=0; j<60; j++)
        {
        buffer2[j]=read_adc();
        }

        for(j=0; j<60; j++)
        {
        buffer3[j]=read_adc();
        }

        for(j=0; j<60; j++)
        {
        buffer4[j]=read_adc();
        }
}

//Se a escala for de 100Hz entra na rotina com delay em us
else if (escala=='2')
        {
                for(j=0; j<60; j++)
                {
                delay_us(tempo);
                buffer1[j]=read_adc();
                }

                for(j=0; j<60; j++)
                {
                delay_us(tempo);
                buffer2[j]=read_adc();
```

```
                }

                for(j=0; j<60; j++)
                {
                delay_us(tempo);
                buffer3[j]=read_adc();
                }

                for(j=0; j<60; j++)
                {
                delay_us(tempo);
                buffer4[j]=read_adc();
                }
        }

        //Finalmente, se escala for de 1Hz ou 10Hz entra na rotina de ms
        else
        {
        for(j=0; j<60; j++)
                {
                delay_ms(tempo);
                buffer1[j]=read_adc();
                }

                for(j=0; j<60; j++)
                {
                delay_ms(tempo);
                buffer2[j]=read_adc();
                }

                for(j=0; j<60; j++)
                {
                delay_ms(tempo);
                buffer3[j]=read_adc();
                }

                for(j=0; j<60; j++)
                {
                delay_ms(tempo);
                buffer4[j]=read_adc();
                }
        }

 //Sinaliza ao computador que o buffer está cheio
 printf("blz");

//_____________________________________________________________________
//Aguarda uma caractere e inicia o envia pela serial

        for(j=0; j<60; j++)
        {
        c=getc();
        printf ("%3u",buffer1[j]);
        }

        for(j=0; j<60; j++)
        {
        c=getc();
        printf ("%3u",buffer2[j]);
        }
```

```
        for(j=0; j<60; j++)
        {
        c=getc();
        printf ("%3u",buffer3[j]);
        }

        for(j=0; j<60; j++)
        {
        c=getc();
        printf ("%3u",buffer4[j]);
        }
}
}
```

APÊNDICE E – *SOFTWARE* DO BUILDER C++

Unit1.cpp

```
//---------------------------------------------------------------------
#include <vcl\vcl.h>
#include <time.h>
#include <stdio.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit3.h"
#include "Serial.h"

//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SLScope"
#pragma link "CSPIN"
#pragma resource "*.dfm"

TForm1 *Form1;
TLeitura *LeituraThread;

AnsiString sPorta;
String amostras, amplitude;
int parar=0;
int temp_amost=36;

//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
leitura=1;
sPorta = "COM4";                                    //Porta inicial
amostras="4";                                       //Frequencia inicial 1KHz
amplitude="2";                                      //Amplitude inicial de -50 a +50 V
fator = 20;                                         //Amplitude inicial de -50 a +50 V
Label2->Caption=(AnsiString(temp_amost) + (" µs/amostra"));
ComboBox3->ItemIndex=2;
ComboBox4->ItemIndex=4;
RadioButton3->Checked=true;
GroupBox1->Enabled=false;
Edit2->Enabled=false;
Edit3->Enabled=false;
RadioButton2->Enabled=false;
LeituraThread = new TLeitura(false);    //Ativa a Thread
LeituraThread->Suspend();               //Pausa a Thread}
//---------------------------------------------------------------------
```

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
   bEncerraSerial();
   LeituraThread->Terminate();
}

//---------------------------------------------------------------------------
//Seleção da frequência,
//Indica na tela o tempo/amostra e
//Configura o nº de pontos do gráfico (eixo X)

void __fastcall TForm1::ComboBox4Change(TObject *Sender)
{
if (ComboBox4->ItemIndex==0)    //1Hz
{
amostras="0";
temp_amost=20;
Label2->Caption=(AnsiString(temp_amost) + (" ms/amostra"));
SLScope1->XAxis->Max->Value=100;
}

if (ComboBox4->ItemIndex==1)    //10Hz
{
amostras="1";
temp_amost=2;
Label2->Caption=(AnsiString(temp_amost) + (" ms/amostra"));
SLScope1->XAxis->Max->Value=150;
}

if (ComboBox4->ItemIndex==2)    //100Hz
{
amostras="2";
temp_amost=285;
Label2->Caption=(AnsiString(temp_amost) + (" µs/amostra"));
 SLScope1->XAxis->Max->Value=150;
}

if (ComboBox4->ItemIndex==3)    //500Hz
{
amostras="3";
temp_amost=36;
Label2->Caption=(AnsiString(temp_amost) + (" µs/amostra"));
SLScope1->XAxis->Max->Value=150;
}

if (ComboBox4->ItemIndex==4)    //1000Hz
{
amostras="4";
temp_amost=36;
Label2->Caption=(AnsiString(temp_amost) + (" µs/amostra"));
SLScope1->XAxis->Max->Value=200;
}
}
//---------------------------------------------------------------------------
//Seleção da amplitude
//Configura o fator de conversão de tensão
//Configura os limites do gráfico (eixo Y)

void __fastcall TForm1::ComboBox3Change(TObject *Sender)
{
```

```
if (ComboBox3->ItemIndex==0)
{
amplitude="0";                  //De -2,5 a +2,5 V
fator = 1;
SLScope1->YAxis->Min->Value=-2.5;
SLScope1->YAxis->Max->Value=+2.5;
}
if (ComboBox3->ItemIndex==1)
{
amplitude="1";                  //De -10 a +10 V
fator = 4;
SLScope1->YAxis->Min->Value=-10;
SLScope1->YAxis->Max->Value=+10;
}
if (ComboBox3->ItemIndex==2)
{
amplitude="2";                  //De -50 a +50 V
fator = 20;
SLScope1->YAxis->Min->Value=-50;
SLScope1->YAxis->Max->Value=+50;
}
}
//---------------------------------------------------------------------------
// Botão iniciar análise

void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
static char buffer[TAM_MAX_BUFFER];
String c;
BYTE aux;

if(BitBtn1->Caption=="Iniciar Análise")
        {
        parar=0;
        bInicializaSerial(sPorta.c_str());      //inicializa serial

        Sleep(10);
        bTransmiteMensagem(amostras.c_str());
        Sleep(10);
        bTransmiteMensagem(amplitude.c_str());

        leitura=0;
        BitBtn1->Kind=bkCancel;
        BitBtn1->Caption="Parar Análise";
        ComboBox3->Enabled=false;
        SelecPorta1->Enabled=false;             //Desabilita seleção de porta
        Baud1->Enabled=false;                   //Desabilita seleção de BaudRate

        LeituraThread->Resume();                //Reinicia a Thread
        Timer1 -> Enabled = true;
        }
else
        {
        parar=1;
        BitBtn1->Kind=bkOK;
        BitBtn1->Caption="Iniciar Análise";
        ComboBox3->Enabled=true;
        SelecPorta1->Enabled=true;              //habilita seleção de porta
        Baud1->Enabled=true;
        }
```

```
}
//---------------------------------------------------------------------------
//Seleções da porta COM


void __fastcall TForm1::COM11Click(TObject *Sender)
{
COM11->Checked=true;
sPorta = "COM1";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM21Click(TObject *Sender)
{
COM21->Checked=true;
sPorta = "COM2";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM31Click(TObject *Sender)
{
COM31->Checked=true;
sPorta = "COM3";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM41Click(TObject *Sender)
{
COM41->Checked=true;
sPorta = "COM4";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM51Click(TObject *Sender)
{
COM51->Checked=true;
sPorta = "COM5";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM61Click(TObject *Sender)
{
COM61->Checked=true;
sPorta = "COM6";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM71Click(TObject *Sender)
{
COM71->Checked=true;
sPorta = "COM7";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::COM81Click(TObject *Sender)
{
COM81->Checked=true;
sPorta = "COM8";
}
```

```
//---------------------------------------------------------------------------
//Botão de decremento do nível de tensão
void __fastcall TForm1::CSpinButton1DownClick(TObject *Sender)
{
Edit2->Text = Edit2->Text.ToDouble()-0.2;
}
//---------------------------------------------------------------------------
//Botão de incrementa nível de tensão
void __fastcall TForm1::CSpinButton1UpClick(TObject *Sender)
{
Edit2->Text = Edit2->Text.ToDouble()+0.2;
}
//---------------------------------------------------------------------------
//Decrementa tensão de disparo
void __fastcall TForm1::CSpinButton2DownClick(TObject *Sender)
{
Edit3->Text = Edit3->Text.ToDouble()-0.1;
}
//---------------------------------------------------------------------------
//Incrementa tensão de disparo
void __fastcall TForm1::CSpinButton2UpClick(TObject *Sender)
{
Edit3->Text = Edit3->Text.ToDouble()+0.1;
}
//---------------------------------------------------------------------------
// Timer1

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
int i, j, trigger_point;
double nivel;

//Se a Thread de leitura sinalizou que o buffer está cheio
if (leitura==1)
{
      //Se trigger manual
      if (RadioButton5->Checked==true)
      {
            //O valor de disparo é o digitado no EditBox
            valor_medio = Edit3->Text.ToDouble();

            //Se derivada positiva
            if (RadioButton1->Checked==true)
            {
             for( i = 0; i < 240; i ++ )
             {

                 //Procura a primeira amostra que passa pelo valor_medio
              if ((Buffer[i]<=valor_medio)&&(Buffer[i+1]>=valor_medio))
                        {
                        trigger_point=i;
                        i=240;
                        }
              }
         }

            //Se derivada negativa
            else if (RadioButton2->Checked==true)
            {
             for( i = 0; i < 240; i ++ )
```

```
                {

                  //Procura a primeira amostra que passa pelo valor_medio
                if ((Buffer[i]>=valor_medio)&&(Buffer[i+1]<=valor_medio))
                        {
                        trigger_point=i;
                        i=240;
                        }
                 }
            }

            j=0;
            nivel=Edit2->Text.ToDouble();

            //Forma o novo Buffer com "plot" amostras e soma nível de tensão
            for ( i=trigger_point; i < 240; i++ )
            {
            Buffer2[j]=Buffer[i]+nivel;
            j++;
            }

            //Define o número de pontos no gráfico e imprime
            plot=j;

  // Desenha no gráfico o novo buffer
  SLScope1->Channels->Channels[ 0 ]->Data->SetYData( Buffer2, plot );
 }
//_____________________________________________________________________
//Se trigger automático
      if (RadioButton4->Checked==true)
      {

            for( i = 0; i < 240; i ++ )
            {

              //Procura a primeira amostra que passa pelo valor_medio
             if ((Buffer[i]<=valor_medio)&&(Buffer[i+1]>=valor_medio))
                        {
                        trigger_point=i;
                        i=240;
                        }
             }

                j=0;
                 nivel=Edit2->Text.ToDouble();

            //Forma o novo Buffer com "plot" amostras e soma nível de tensão
            for ( i=trigger_point; i < 240; i++ )
            {
            Buffer2[j]=Buffer[i]+nivel;
            j++;
            }

            //Define o número de pontos no gráfico e imprime
            plot=j;
  // Desenha no gráfico o novo buffer
  SLScope1->Channels->Channels[ 0 ]->Data->SetYData( Buffer2, plot );
   }

//_____________________________________________________________________
```

```
//Se captura contínua imprime os 240 pontos
if (RadioButton3->Checked==true)
   {
   SLScope1->Channels->Channels[ 0 ]->Data->SetYData( Buffer, 240 );
   }

//Se botão p/ parar análise for pressionado envia caractere 'x'
if (parar==1)
{
Edit1->Text = "x";
bTransmiteMensagem(Form1->Edit1->Text.c_str());
bEncerraSerial();                                    // Encerra comunicação serial
LeituraThread->Suspend();                            //Pausa a Thread
Timer1 -> Enabled = false;                           //Para o Timer1
leitura=1;
}

//Se o programa continua envia os parâmetros de freq e amplitude
else
{
bTransmiteMensagem(amostras.c_str());
Sleep(10);
bTransmiteMensagem(amplitude.c_str());

leitura=0;                  //libera a leitura dos dados
}
}
}
//---------------------------------------------------------------------------
// Botão Calcular!

void __fastcall TForm1::Button1Click(TObject *Sender)
{
double freq, period;

if (ComboBox4->ItemIndex>=2)   //500Hz para cima
{
period=temp_amost*Edit4->Text.ToDouble()/1000;
freq=(1/(period/1000));
Label14->Caption=(AnsiString(period) + (" milisegundos"));
//rotina para arredondar a frequência para 0 dígitos depois da vírgula:
Label15->Caption=(AnsiString(FloatToStrF(freq,ffNumber,7,0)) + (" Hz"));
}

else if (ComboBox4->ItemIndex<2)    //1Hz e 10hz
{
period=temp_amost*Edit4->Text.ToDouble()/1000;
freq=(1/period);
Label14->Caption=(AnsiString(period) + (" segundos"));
Label15->Caption=(AnsiString(FloatToStrF(freq,ffNumber,7,0)) + (" Hz"));
}
}
//---------------------------------------------------------------------------
//Informações
void __fastcall TForm1::Informaes1Click(TObject *Sender)
{
ShowMessage(AnsiString( "Osciloscópio via PC com aquisição microcontrolada\n")+
("Trabalho de conclusão de curso 2009/1\n") + ("\nALUNO: Daniel Alvienes Wentz\n")+
("ORIENTADOR: Prof. MSc. Eng. Eletr. Dalton Luiz Rech Vidor" ));
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton3Click(TObject *Sender)
{
GroupBox1->Enabled=false;
Edit2->Enabled=false;
Edit3->Enabled=false;
RadioButton1->Enabled=false;
RadioButton2->Enabled=false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton4Click(TObject *Sender)
{
GroupBox1->Enabled=false;
Edit2->Enabled=false;
Edit3->Enabled=false;
RadioButton1->Enabled=false;
RadioButton2->Enabled=false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton5Click(TObject *Sender)
{
GroupBox1->Enabled=true;
Edit2->Enabled=true;
Edit3->Enabled=true;
RadioButton1->Enabled=true;
RadioButton2->Enabled=true;
}
//---------------------------------------------------------------------------
```

Unit1.h

```
#ifndef Unit1H
#define Unit1H
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "LPDrawLayers.h"
#include "SLComponentCollection.h"
#include "SLScope.h"
#include "SLStreamTypes.h"
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>
#include <Buttons.hpp>
#include "CSPIN.h"
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TSLScope *SLScope1;
        TTimer *Timer1;
        TEdit *Edit1;
        TPanel *Panel1;
        TMainMenu *MainMenu1;
        TPageControl *PageControl1;
        TTabSheet *TabSheet1;
        TTabSheet *TabSheet2;
        TLabel *Label3;
        TComboBox *ComboBox3;
        TLabel *Label4;
        TComboBox *ComboBox4;
        TLabel *Label1;
        TLabel *Label2;
        TGroupBox *GroupBox1;
        TBitBtn *BitBtn1;
        TMenuItem *SelecPorta1;
        TMenuItem *COM11;
        TMenuItem *COM21;
        TMenuItem *COM31;
        TMenuItem *COM41;
        TMenuItem *COM51;
        TMenuItem *COM61;
        TMenuItem *COM71;
        TMenuItem *COM81;
        TCSpinButton *CSpinButton1;
        TEdit *Edit2;
        TLabel *Label5;
        TRadioButton *RadioButton1;
        TRadioButton *RadioButton2;
        TCSpinButton *CSpinButton2;
        TLabel *Label7;
        TLabel *Label6;
        TEdit *Edit3;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
```

```
        TLabel *Label11;
        TMenuItem *Baud1;
        TMenuItem *N48001;
        TMenuItem *N96001;
        TMenuItem *N192001;
        TMenuItem *N571;
        TMenuItem *N1152001;
        TLabel *Label13;
        TButton *Button1;
        TLabel *Label14;
        TLabel *Label15;
        TMenuItem *Informaes1;
        TRadioGroup *RadioGroup1;
        TRadioButton *RadioButton3;
        TRadioButton *RadioButton4;
        TRadioButton *RadioButton5;
        TEdit *Edit4;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall ComboBox4Change(TObject *Sender);
        void __fastcall ComboBox3Change(TObject *Sender);
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall COM11Click(TObject *Sender);
        void __fastcall COM21Click(TObject *Sender);
        void __fastcall COM31Click(TObject *Sender);
        void __fastcall COM41Click(TObject *Sender);
        void __fastcall COM51Click(TObject *Sender);
        void __fastcall COM61Click(TObject *Sender);
        void __fastcall COM71Click(TObject *Sender);
        void __fastcall COM81Click(TObject *Sender);
        void __fastcall CSpinButton1DownClick(TObject *Sender);
        void __fastcall CSpinButton1UpClick(TObject *Sender);
        void __fastcall CSpinButton2DownClick(TObject *Sender);
        void __fastcall CSpinButton2UpClick(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Informaes1Click(TObject *Sender);
        void __fastcall RadioButton3Click(TObject *Sender);
        void __fastcall RadioButton4Click(TObject *Sender);
        void __fastcall RadioButton5Click(TObject *Sender);
private:        // User declarations
public:         // User declarations

        float Buffer[ 250 ], Buffer2[ 250 ], fator, valor_medio;
        int leitura, plot;

        __fastcall TForm1(TComponent* Owner);
};
extern PACKAGE TForm1 *Form1;
#endif
```

Unit3.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit3.h"
#include "Serial.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

__fastcall TLeitura::TLeitura(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TLeitura::Execute()
{
int i, valor;
static char buffer[TAM_MAX_BUFFER];
String c;
BYTE aux;

FreeOnTerminate=true;

while(1)
{

//Se o flag de leitura estiver '0' libera para receber dados
if (Form1->leitura==0)
{
        aux = bRecebeMensagem(buffer);
        c=((String)buffer);

        // Aguarda o PIC terminar as amostragens
        if (c=="blz")
        {
                Form1->valor_medio=0;

                 //Recebe os 240 valores pela serial
                for( i = 0; i < 240; i ++ )
                {
                Form1->Edit1->Text = "";
                bTransmiteMensagem(Form1->Edit1->Text.c_str());
Sleep(20);
                aux = bRecebeMensagem(buffer);
                valor = StrToInt((String)buffer);
                Form1->Buffer[ i ] = valor;

                // se menor que 128 converte p/ 0/-2,5 x fator
                if (!(valor & 0x80))
                Form1->Buffer[ i ] = (((Form1->Buffer[ i ]/128)*2.5)-2.5)*Form1->fator;

                // se maior que 128 converte p/ 0/+2,5 x fator
                else
                        {
```

```
                Form1->Buffer[ i ] = (valor & 0x7f);
                Form1->Buffer[ i ] = (((Form1->Buffer[ i ])/255)*5)*Form1->fator;
                }

            Form1->valor_medio = Form1->valor_medio + Form1->Buffer[ i ];
            }

        Form1->valor_medio = Form1->valor_medio / 240;

        //Desabilita Leitura e habilita escrita
        Form1->leitura=1;
        }
}
}
}
```

Unit3.h

```
#ifndef Unit3H
#define Unit3H
//---------------------------------------------------------------------------
#include <Classes.hpp>
//---------------------------------------------------------------------------
class TLeitura : public TThread
{
private:
protected:
        void __fastcall Execute();
public:
        __fastcall TLeitura(bool CreateSuspended);
};
//---------------------------------------------------------------------------
#endif
```

ANEXO A – PROTOCOLO SERIAL

Serial.cpp

```
#include <vcl.h>
#pragma hdrstop

#include <mmsystem.h>
#include <mem.h>
#include "Serial.h"

#pragma package(smart_init)

/*****************************************************************************
              VARIÁVEIS ESTÁTICAS
*****************************************************************************/
static HANDLE      hComm;
static COMMTIMEOUTS CommTimeouts, DefaultCommTimeouts;
static DCB         dcb;
static OVERLAPPED   osWrite = {0};
static OVERLAPPED   osRead  = {0};
static DWORD       dwErroRecepcaoSerial;
static WORD        wIndexBuffer;

//---------------------------------------------------------------------------
BYTE bInicializaSerial(char* porta)
{
  // Carrega a estrutura de configuração da comunicação com os valores defaults.
  if (porta == NULL || porta == '\0')
    return ERRO_ABERTURA_PORTA_COMUNICACAO;

  // Abre a porta de comunicacao COM1, recebendo um handle para ela.
  hComm =
CreateFile(porta,GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,FILE_FLAG_OVERLAPP
ED,NULL);
  if (hComm == INVALID_HANDLE_VALUE)
    return ERRO_ABERTURA_PORTA_COMUNICACAO;

  // Define o tamanho e limpa os buffers de transmissao e recepcao da porta de comunicacao.
  if (!SetupComm(hComm, (DWORD)TAM_MAX_BUFFER, (DWORD)TAM_MAX_BUFFER))
    return ERRO_CONFIGURACAO_PORTA_COMUNICACAO;

  // Preenche a estrutura "dcb" com os parametros de comunicacao utilizados pela porta.
  if (!GetCommState(hComm, &dcb))
    return ERRO_CONFIGURACAO_PORTA_COMUNICACAO;

  // Modifica os valores da estrutura "dcb" com os parametros a serem utilizados na comunicacao.
  //dcb.BaudRate       = DEFAULT_BAUD_RATE;
  dcb.BaudRate       = 115200;
  dcb.fBinary        = TRUE;
  dcb.fParity        = TRUE;
  dcb.fOutxCtsFlow    = FALSE;
  dcb.fOutxDsrFlow    = FALSE;
  dcb.fDtrControl     = DTR_CONTROL_DISABLE;
  dcb.fDsrSensitivity = FALSE;
  dcb.fOutX          = FALSE;
  dcb.fInX           = FALSE;
  dcb.fErrorChar      = FALSE;
  dcb.fNull          = FALSE;
  dcb.fRtsControl     = RTS_CONTROL_DISABLE;
  dcb.fAbortOnError   = FALSE;
```

```
  //dcb.wReserved      = 0;
  dcb.ByteSize       = DEFAULT_NR_BITS_DADOS;
  dcb.Parity         = DEFAULT_PARIDADE;
  dcb.StopBits       = DEFAULT_NR_STOP_BITS;

  // Programa a porta serial com os novos parâmetros de comunicação.
  if (!SetCommState(hComm, &dcb))
    return ERRO_CONFIGURACAO_PORTA_COMUNICACAO;

  if (!GetCommTimeouts(hComm, &DefaultCommTimeouts))
    return ERRO_CONFIGURACAO_PORTA_COMUNICACAO;

  // Altera os valores dos timeouts para possibilitar o funcionamento do comunicação baseada em eventos.
  // Programa o time-out para ocorrer em 40 ms caso não seja recebido nenhum byte.
  CommTimeouts.ReadIntervalTimeout         = MAXDWORD;
  CommTimeouts.ReadTotalTimeoutMultiplier  = 0;
  CommTimeouts.ReadTotalTimeoutConstant    = 0;
  CommTimeouts.WriteTotalTimeoutMultiplier = 0;
  CommTimeouts.WriteTotalTimeoutConstant   = 0;
  // Programa os novos valores de time-outs
  if (!SetCommTimeouts(hComm, &CommTimeouts))
    return ERRO_CONFIGURACAO_PORTA_COMUNICACAO;

  // Cria um evento para o modo de transmissão (inicializa o handle hEvent da estrutura OVERLAPPED).
  osWrite.hEvent = CreateEvent(NULL, true, false, NULL);
  if (osWrite.hEvent==NULL)
    // Erro ao criar um handle de evento
    return ERRO_CRIACAO_EVENTO;

  // Cria um evento para o modo de transmissão (inicializa o handle hEvent da estrutura OVERLAPPED).
  osRead.hEvent = CreateEvent(NULL, true, false, NULL);
  if (osRead.hEvent==NULL)
    // Erro ao criar um handle de evento
    return ERRO_CRIACAO_EVENTO;

  // Prepara o modo de recepção.
  // Inicializa parâmetros para recepção
  dwErroRecepcaoSerial = 0;
  wIndexBuffer         = 0;

  return ERRO_INEXISTENTE;
}
//---------------------------------------------------------------------------

BYTE bEncerraSerial(void)
{
  if (hComm != INVALID_HANDLE_VALUE)
    {
    // Reprograma os valores defaults de time-outs.
    if (!SetCommTimeouts(hComm, &DefaultCommTimeouts))
      return ERRO_FECHAMENTO_PORTA_COMUNICACAO;

    if (CloseHandle(hComm))
      {
      if (CloseHandle(osWrite.hEvent))
        {
        if (CloseHandle(osRead.hEvent))
          return ERRO_INEXISTENTE;
        else
          return ERRO_FECHAMENTO_PORTA_COMUNICACAO;
        }
```

```
        else
          return ERRO_FECHAMENTO_PORTA_COMUNICACAO;
        }
      else
        return ERRO_FECHAMENTO_PORTA_COMUNICACAO;
      }
    else
      return ERRO_INEXISTENTE;
}
//---------------------------------------------------------------------------

BYTE bTransmiteMensagem(char* mensagem)
{
DWORD dwNrBytesTransmitidos, dwNrBytesTransmitir;

  // Limpa o buffer de transmissão e de recepção da porta de comunicação.
  if (!PurgeComm(hComm, PURGE_RXCLEAR | PURGE_TXCLEAR))
    return ERRO_TRANSMISSAO_SERIAL;

  // Prepara o modo de recepção.
  // Inicializa parâmetros para recepção
  dwErroRecepcaoSerial = 0;
  wIndexBuffer         = 0;

  dwNrBytesTransmitir = strlen(mensagem) + 1;

  // Envia a mensagem para o driver de comunicação da API do WIN32.

  if (!WriteFile(hComm,LPCVOID(mensagem),dwNrBytesTransmitir,&dwNrBytesTransmitidos,&osWrite))
    {
    if (GetLastError() != ERROR_IO_PENDING)
      // Erro no envio da mensagem.
      return ERRO_TRANSMISSAO_SERIAL;
    // Transmissão pendente.
    }
  else
    // Transmissão completada com sucesso.
    {
    // Limpa a flag que sinaliza o evento
    ResetEvent(osWrite.hEvent);
    }
  return ERRO_INEXISTENTE;
}
//---------------------------------------------------------------------------

BYTE bRecebeMensagem(char* mensagem)
{
DWORD dwNrBytesRecebidos;
BYTE  status;

  if (ReadFile(hComm,&(mensagem[wIndexBuffer]),TAM_MAX_BUFFER,&dwNrBytesRecebidos,&osRead))
    {
    if (dwNrBytesRecebidos)
      // Um ou mais bytes foram recebidos.
      {
      wIndexBuffer += (BYTE)dwNrBytesRecebidos;
      status = SS_SERVICO_EM_ANDAMENTO;
      }
    else
      {
      // Não há mais bytes a receber, encerra a recepção.
```

```
      // Sinaliza recepção do frame encerrada com resposta (com ou sem erros de serial)
      ClearCommError(hComm,&dwErroRecepcaoSerial,NULL);
      if (dwErroRecepcaoSerial & CE_FRAME || dwErroRecepcaoSerial & CE_RXOVER ||
         dwErroRecepcaoSerial & CE_OVERRUN || dwErroRecepcaoSerial & CE_RXPARITY)
        status = SS_RESPOSTA_COM_ERRO;
      else
        if (wIndexBuffer > 0)
          status = SS_RESPOSTA_SEM_ERRO;
        else
          status = SS_SEM_RESPOSTA;

      // Prepara o modo de recepção.
      // Inicializa parâmetros para recepção
      dwErroRecepcaoSerial = 0;
      wIndexBuffer         = 0;
      }
    }
  else
    status = SS_RESPOSTA_COM_ERRO;

  return status;
}
```

Serial.h

```
/*****************************************************************************
            DEFINIÇÕES
*****************************************************************************/
#define TAM_MAX_BUFFER              10000
#define DEFAULT_BAUD_RATE           CBR_19200
#define DEFAULT_NR_BITS_DADOS       8
#define DEFAULT_PARIDADE            NOPARITY
#define DEFAULT_NR_STOP_BITS        ONESTOPBIT

#define SetBit(a,b)       (a |= (0x01<<(b)))
#define ClearBit(a,b)     (a &= ~(0x01<<(b)))
#define CheckBit(a,b)     ((a>>b) & 0x01)
#define MSB(a)            (BYTE)((a) >> 8)
#define LSB(a)            (BYTE)(a)
#define MSW(a)            (WORD)((a) >> 16)
#define LSW(a)            (WORD)(a)
#define MSN(a)            (BYTE)(((a) >> 4) & 0x0F)
#define LSN(a)            (BYTE)((a) & 0x0F)

/*****************************************************************************
            ENUMERAÇÕES
*****************************************************************************/

enum MENSAGENS_ERRO
  {
  ERRO_INEXISTENTE,
  ERRO_ABERTURA_PORTA_COMUNICACAO,
  ERRO_FECHAMENTO_PORTA_COMUNICACAO,
  ERRO_CONFIGURACAO_PORTA_COMUNICACAO,
  ERRO_CRIACAO_EVENTO,
  ERRO_RECEPCAO_SERIAL,
  ERRO_TRANSMISSAO_SERIAL
  };

enum STATUS_SERVICO
  {
  SS_SEM_SERVICO,
  SS_SERVICO_EM_ANDAMENTO,
  SS_SEM_RESPOSTA,
  SS_RESPOSTA_SEM_ERRO,
  SS_RESPOSTA_COM_ERRO
  };

/*****************************************************************************
            PROTÓTIPOS
*****************************************************************************/
extern BYTE bInicializaSerial(char* porta);
extern BYTE bEncerraSerial(void);
extern BYTE bTransmiteMensagem(char* mensagem);
extern BYTE bRecebeMensagem(char* mensagem);
```