UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

NILSON CEZAR DE OLIVEIRA

ANALISADOR DE ESPECTRO VIA PC COM AQUISIÇÃO MICROCONTROLADA

Canoas, Julho de 2010

NILSON CEZAR DE OLIVEIRA

Analisador de espectro 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. Miriam N. C. Villamayor – CREA-RS: 06.7231-D

Canoas

2010

FOLHA DE APROVAÇÃO

Nome do Autor: Nilson Cezar de Oliveira

Matrícula: 031009240-0

Título: Analisador de espectro 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. Miriam N. C. Villamayor

CREA-RS: RS067231-D

Banca Avaliadora:

MSc. Eng. Eletr. André Luis Bianchi

CREA-RS: RS089197-D

Conceito Atribuído (A-B-C-D):

MSc Eng. Eletr. Paulo César Cardoso Godoy

CREA-RS: RS0116822-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor Nilson Cezar de Oliveira	Orientador Miriam Noemi Caceres Villamayor
Avaliador André Luis Bianchi	Avaliador Paulo César Cardoso Godoy

Relatório Aprovado em:

DEDICATÓRIA

Dedico a minha mãe e a minha namorada...

AGRADECIMENTOS

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

A Professora Miriam pelo estímulo, dedicação e esforço pessoal proporcionado.

Ao Professor Dalton pelas valiosas contribuições.

A minha mãe Gislaine pelo apoio incondicional durante toda a minha vida e à minha namorada Silvia que sempre me incentivou.

EPÍGRAFE

"Suba o primeiro degrau com fé.

Não é necessário que você veja toda a escada.

Apenas dê o primeiro passo".

Martin Luther King

RESUMO

DE OLIVEIRA, Nilson Cezar **Analisador de espectro Via PC com aquisição microcontrolada**. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2010.

Este trabalho é uma contribuição ao trabalho executado pelo aluno Daniel Alvienes Wentz nessa mesma Instituição de ensino. A base do projeto é um osciloscópio via PC com aquisição microcontrolada, sendo realizadas no mesmo algumas alterações e melhorias. A principal delas é a adição de um visualizador de espectro de amplitudes e frequências através de uma modelagem em C++, utilizando o algoritmo FFT (Fast Fourier Transform). Com isso foi possível verificar a o conteúdo espectral de vários sinais, em especial sinais elétricos que estão presentes em residências e empresas, podendo ser utilizado, inclusive para verificar a qualidade dos mesmos.

Palavras chave: Osciloscópio. Espectro. Microcontrolador. Análise. Programação.

ABSTRACT

DE OLIVEIRA, Nilson Cezar. **Spectrum analyzer through the computer with acquisition microcontrolled.** Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2010.

This work is a contribution to the work performed by student Daniel Alvienes Wentz in that educational institution. The basis of the project is an oscilloscope via PC with microcontrolled acquisition, being held on the same few changes and improvements. The main one is the addition of a viewer spectrum of amplitudes and frequencies through a modeling in C + +, using the FFT algorithm (Fast Fourier Transform). It was then possible to verify the spectral content of various signals, particularly electrical signals that are present in homes and businesses and can be used, including to check the quality.

Keywords: Oscilloscope. Spectrum. Microcontroller. Analysis. Programming.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FFT – Fast Fourier Transform

BCB – Borland C Builder

FT – Fourier Transform

DFT – Discrete Fourier Transform

IBM - International Business Machines

DSP - Digital signal processing

DTE - Data Terminal Equipament

DCE - Data Communication Equipment

ASCII - American Standard Code for Information Interchange

EBCDIC - Extended Binary Coded Decimal Interchange Code

PC – Personal Computer

USB – Universal Serial Bus

PCI - Peripheral Component Interconnect

ISA - Industry Standard Architecture

IDE - Integrated Development Environment

RAD - Rapid Application Development

PIC - Controller Integrated Peripherals

SOIC - Small-Outline Integrated Circuit

DIP - Dual In-Line Package

TQFP - Thin Quad Flat Pack

BNC - British Naval Connector

MIPS - Milhões de Instruções Por Segundo

EIA - Eletronics Industries Association

LISTA DE SÍMBOLOS

V – Volt

Hz – Hertz

k – kilo – 1×10^3

M – Mega – 1×10^6

Hz – Hertz

ms – milisegundos

μs - microsegundos

SUMÁRIO

1. INTRODUÇÃO

Num mundo globalizado, onde qualquer tecnologia esta disponível em praticamente todo lugar, muitas vezes sendo criada em um país e aperfeiçoada em outro, é vital pensar nas possibilidades de melhoria de muitos produtos com os quais se tem contato, principalmente na área de engenharia. Seguindo esta lógica o presente trabalho foi realizado, partindo de um projeto iniciado anteriormente e através da inserção de melhorias, foi construído um produto com mais recursos que o original. O software Analisador espectral (AE), desenvolvido neste trabalho, é formado por um osciloscópio que apresenta o sinal real que foi amostrado e por um espectro que exibe as amplitudes e frequências que o compõe, determinando como este sinal é formado e passando a informação necessária para verificação de interferências para a aplicação de filtros e até mesmo determinando a sua qualidade.

Este trabalho segue a mesma linha do anterior, com a obtenção do sinal, o tratamento microcontrolado e a visualização na tela de um computador comum. Isto é capaz de torná-lo um sistema de fácil utilização com um custo muito baixo, possibilitando sua ampla aplicação, principalmente em instituições de ensino que queiram demonstrar de forma prática o que é visto de forma teórica nas aulas de engenharia.

1.1. *Visão Geral do Problema*

Equipamentos que realizam a tarefa de analisador de espectro possuem um alto custo. Com isso as escolas encontram uma dificuldade de proporcionar a demonstração prática da teoria de análise de sinais para seus alunos deixando de prepará-los de uma forma mais completa para o mercado de trabalho.

1.2. *Formulação do Problema de Engenharia*

Dificuldade na aquisição de equipamentos de análise espectral por parte das escolas de ensino da área elétrica, não disponibilizando da maneira adequada esta ferramenta para seus alunos.

1.3. *Formulação do Problema Comercial*

Do ponto de vista econômico, um sistema de análise espectral possui um alto custo de aquisição, sendo necessária uma solução que corrija este problema

disponibilizando de forma mais fácil e barata esta ferramenta, essencial para qualquer engenheiro eletricista.

1.3.1. Identificação dos Interessados

Os principais interessados no sistema, certamente, serão as instituições de ensino superior que possuam qualquer curso de graduação na área elétrica e que não disponibilizem um orçamento suficiente para aquisição de equipamentos de análise espectral.

1.3.2. Oportunidade de Negócios

O sistema não é uma novidade, pois já foi elaborado em outras instituições, com dados reais ou virtuais, mas não tendo objetivos comerciais ou didáticos, sendo este o diferencial do produto do qual é tratado neste trabalho.

1.4. *Definição do Escopo do Projeto*

Foram adicionadas algumas melhorias no projeto anterior, o que pôde torná-lo um produto mais atraente, sendo mais completo e eficiente. Para isso foram alterados os softwares do microcontrolador e do computador, introduzindo um algoritmo melhor para o protocolo de comunicação e para o tratamento dos dados.

1.4.1. Objetivos Gerais

O principal objetivo é o de elaborar um sistema capaz de mostrar o espectro de amplitudes e frequências através de um microcontrolador e visualização na tela de um computador, melhorando também o protocolo de comunicação para dar mais velocidade na atualização dos gráficos.

1.4.2. Objetivos Específicos

Aumento do limite de tensão máxima de entrada para 320 V, pois o limite de 50 V, para o qual o sistema foi inicialmente projetado, não atende as medições de uma rede elétrica tradicional, além disso, serão acrescidas informações relativas aos gráficos do domínio tempo e frequência.

1.4.3. Metas

O projeto tem como meta principal, que instituições de ensino possam utilizar o projeto durante as suas aulas de demonstração prática possibilitando ao aluno a comprovação visual de toda a teoria da transformada discreta de Fourier que possui grande importância no processamento digital de sinais.

1.4.4. Resultados Previstos

Os resultados esperados são a execução de um analisador espectral, além da implementação de melhorias no osciloscópio atual, como o aumento da escala de entrada, ampliação da velocidade de atualização, indicação de informações dos gráficos, acréscimo do número de amostras coletadas e gravação em arquivo com possibilidade de visualização futura.

2. REFERENCIAL TEÓRICO

2.1. *Comunicação serial RS-232*

A RS-232 (também conhecido por EIA RS-232C ou V.24) é um padrão para troca serial de dados binários entre um DTE (terminal de dados, Data Terminal Equipament) e um DCE (comunicador de dados, de Data Communication Equipament). É comumente usado nas portas seriais dos PCs.

2.1.1. Escopo do padrão

A Eletronics Industries Association (EIA), que padronizou o RS-232-C em 1969, define:

- Características elétricas como níveis de tensão, taxa de sinalização, taxa de rotação dos sinais, nível máximo de tensão, comportamento de curto-circuito e carga máxima da capacitância;
- Características mecânicas da interface, conectores "plugáveis" e identificação dos pinos;
- Funções de cada circuito no conector da interface;
- Subconjuntos padrões de circuitos de interface para aplicações selecionadas de telecomunicação.

O padrão não define elementos como:

- Codificação de caracteres (por exemplo, ASCII, código Baudot ou EBCDIC);
- Enquadramento dos caracteres no fluxo de dados (bits por caractere, bits de início e parada, paridade);
- Protocolos para detecção de erros ou algoritmos para compressão de dados;
- Taxas de bit para transmissão, apesar de o padrão dizer ser destinado para taxas de bits menores que 20.000 bits por segundo.

Muitos dispositivos modernos suportam velocidade de 115.200 bits/s;

- Fornecimento de energia para dispositivos externos.

2.1.2. História

Este padrão foi originalmente usado para conectar um teletipo (equipamento eletromecânico de comunicação assíncrona que usava código ASCII) a um modem. Quando terminais eletrônicos começaram a ser usados, eram projetados para serem intercambiáveis com as "*teletypewriters*", e também suportavam RS-232. A terceira revisão deste padrão (chamada de RS-232C) foi publicada em 1969, em parte para adequar-se às características elétricas destes dispositivos. Deste modo, foi utilizado em diversos tipos de comunicação remota, especialmente por modems. Posteriormente PCs (e outros equipamentos) começaram a utilizar este padrão para comunicação com equipamentos já existentes. Quando a IBM lançou computadores com uma porta RS-232, esta interface tornou-se realmente onipresente. Por muitos anos o padrão para comunicação serial em quase todos os computadores era algum tipo de porta RS-232. Continuou sendo utilizado em grande escala até o fim dos anos 90. Durante este tempo esta foi a maneira padrão para a conexão de modems [4].

A importância de portas seriais começou a decrescer gradualmente quando redes de alta velocidade tornaram-se disponíveis para comunicação PC com PC. Hoje é comum utilizar conexões Ethernet Base 10, 100 ou 1000 [4].

2.1.3. Papel na computação moderna

Hoje, o protocolo de comunicação RS-232 vem sendo, gradualmente, suprimido pelo USB para comunicação local. O protocolo USB é mais rápido, possui conectores mais simples de usar e tem um melhor suporte por software. Por isso muitas placas-mãe, destinadas ao uso em escritórios ditas "livre de legados" (*legacy-free*), são produzidas sem circuitos RS-232. Mesmo assim, esse protocolo continua sendo utilizado em periféricos para pontos de venda (caixas registradoras, leitores de códigos de barra ou fita magnética) e para a área industrial (dispositivos de controle remoto). Por essas razões, computadores para estes fins continuam sendo produzidos com portas RS-232, tanto *on-board* ou em placas para barramentos PCI ou barramento ISA. Como alternativa, existem adaptadores para portas USB, que podem ser utilizados para conectar teclados ou mouses PS/2, uma ou mais portas seriais e uma ou mais portas paralelas [4].

2.1.4. Conectores

O padrão especifica 20 diferentes sinais de conexão, e um conector em forma de D é comumente usado (Figura 1). São utilizados conectores machos e fêmeas - geralmente os conectores dos cabos são machos e os conectores de dispositivos são fêmeas - e estão disponíveis adaptadores m-m e f-f. Há também os chamados "*null modems*" para conectar unidades utilizando-se ambas como terminais de dados (ou modems).

Figura 1 - Conector fêmea RS-232 de nove pinos

Para configuração e diagnóstico de problemas com cabos RS-232 pode-se utilizar uma "*breakout box*". Este dispositivo possui um conector macho e um conector fêmea e deve ser anexado em linha. Além disso, possui luzes para cada pino e meios de interconectar os pinos com diferentes configurações.

Três são os tipos de sinais carregados por esses fios: terra, transmissão/recepção e "*handshake*". Existem códigos para estes sinais, por exemplo, os apresentados na Tabela 1.

Tabela 1 – Códigos dos sinais da interface serial

Sinal	Significado
SG ou **GND**	Terra
TD ou **TX**	Transmissão de dados
RD ou **RX**	Recepção de dados
DTR	Terminal de dados pronto
DSR	Conjunto de dados pronto
RTS	Pronto para enviar (computador)
CTS	Envie os dados (modem)
DCD	Portadora detectada
RI	Indicador de telefone tocando
FG	(Frame Ground)

Os dispositivos RS-232 podem ser classificados em DTE e DCE. Essa classificação permite definir quais fios irão mandar e/ou enviar sinais de dados. De qualquer modo, estas definições nem sempre são seguidas. Normalmente é

necessário consultar a documentação ou testar as conexões com uma "*breakout box*" para determinar os sinais necessários.

2.1.5. Configurações

Há várias configurações de *software* para conexões seriais. As mais comuns são velocidade e bits de paridade e parada. A velocidade é a quantidade de bits por segundo transmitida de um dispositivo para outro. Taxas comuns de transmissão são 300, 1200, 2400, 9600, 19200, etc. Tipicamente ambos os dispositivos devem estar configurados com a mesma velocidade, alguns dispositivos, porém, podem ser configurados para auto-detectar a velocidade [4].

Paridade é um método de verificar a exatidão dos dados. Paridade é normalmente nula (não usada), mas pode ser par ou ímpar. Paridade funciona modificando os dados, em cada byte enviado. Paridade nula é simples, os dados não são modificados. Na paridade par, os dados são acomodados de modo que o número de bits 1 (isto é, sua contagem em um byte), incluindo o bit de paridade, seja um número par; isto é feito definindo este bit (geralmente os bits mais ou menos significativo) como 0 ou 1. Na paridade impar, o número de bits 1 é um número impar. A paridade pode ser usada pelo receptor para detectar a transmissão de erros - se um byte foi recebido com o número errado de bits 1, então ele deve estar corrompido. Se a paridade estiver correta então não deve haver erros, ou então há um número par de erros.

Bits de parada são enviados no fim de cada byte transmitido com o intuito de permitir que o receptor do sinal se sincronize.

Existe uma convenção para a notação de uma configuração de software na conexão serial, esta notação é da forma D/P/S. Sendo que a configuração mais comum é a 8/N/1 que especifica que são transmitidos 8 bits de dados, paridade nula e um bit de parada. O número de bits de dados pode ser 7, 8 ou (às vezes) 9. Paridade pode ser nula (N), impar (O) ou par (E); o bit de paridade é emprestado dos bits de dados, então 8/E/1 significa que um dos oito bits de dados é utilizado como bit de paridade [4].

2.2. *Linguagem C*

C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, de alto nível, e padronizada, criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional UNIX (que foi originalmente escrito em Assembly). A linguagem C é classificada de

alto nível pela própria definição desse tipo de linguagem. A programação em linguagens de alto nível tem como característica não ser necessário conhecer o processador, ao contrário das linguagens de baixo nível. As linguagens de baixo nível estão fortemente ligadas ao processador. A linguagem C permite acesso de baixo nível com a utilização de código Assembly inserido no código fonte. Assim, o baixo nível é realizado por Assembly e não C. Desde então, espalhou-se por muitos outros sistemas e tornou-se uma das linguagens de programação mais usadas influenciando muitas outras linguagens, especialmente C++, que foi originalmente desenvolvida como uma extensão para C [2].

2.2.1. C++

O C++ é uma linguagem de programação de alto nível com facilidades para o uso em baixo nível, multiparadigma e de uso geral. Desde os anos 1990 é uma das linguagens comerciais mais populares, sendo bastante usada também na área acadêmica por seu grande desempenho e base de utilizadores.

Bjarne Stroustrup desenvolveu o C++ (originalmente com o nome C with Classes) em 1983 no Bell Labs como um adicional à linguagem C. Novas características foram adicionadas com o tempo, como funções virtuais, sobrecarga de operadores, herança múltipla, gabaritos e tratamento de exceções. Após a padronização ISO realizada em 1998 e a posterior revisão realizada em 2003, uma nova versão do padrão da linguagem, conhecida como C++0x, está em desenvolvimento [2].

2.2.2. C++ Builder

C++ Builder (ou abreviado por BCB) é um ambiente de desenvolvimento integrado (IDE) produzido pela CodeGear para a escrita de programas na linguagem C++. Assemelha-se ao Delphi, sendo considerada sua versão em C++, de forma que muitos componentes desenvolvidos para Delphi podem ser utilizados no C++ Builder sem modificação, apesar do inverso não ser verdade. A partir do Borland Developer Studio 2006, ambos as linguagens (Delphi e C++) passaram a compartilhar o mesmo IDE. Hoje eles podem ser adquiridos juntamente no CodeGear RAD Studio [2].

O aplicativo inclui ferramentas que permitem desenvolvimento visual através de "arrastar e soltar", tornando a programação mais simples. Direcionado originalmente somente para a plataforma Windows. Versões mais recentes incorporaram o CLX, uma biblioteca de componentes multi-plataforma baseada em Qt, suportando Windows e Linux.

Em 2003 a Borland introduziu o sucessor do C++ Builder, C++ BuilderX (CBX), que foi escrito utilizando o mesmo framework do JBuilder. Esse produto, que foi direcionado para o desenvolvimento de grandes programas, não obteve sucesso comercial. Como resultado, a Borland anunciou no final de 2004 o retorno do C++ Builder como parte da suíte de desenvolvimento Delphi.

No final de 2005 a Borland lançou o Borland C++ Builder 2006, baseado nas versões antigas do aplicativo e correções de falhas.

Em 2007 a empresa decidiu vender suas ferramentas de desenvolvimento de software e dividiu-se em Borland e CodeGear. A CodeGear lançou o RADStudio 2007 com o Delphi e o C++ Builder incorporado.

Em 2008 a CodeGear foi comprada pela Embarcadero e lançou o RAD Studio 2009 que uniu ferramentas para bancos de dados da Embarcadero com a IDE. (UNICID, 2007).

2.3. *Microcontrolador PIC*

Os PIC's (PICmicro) são uma família de micro controladores fabricados pela Microchip Technology, que processam dados de 8 bits, de 16 bits e mais recentemente 32, com extensa variedade de modelos e periféricos internos, com arquitetura Harvard e conjunto de instruções RISC (conjunto de 35 instruções ou de 76 instruções), com recursos de programação por Memória flash, EEPROM e OTP. O micro controlador PIC tem famílias com núcleos de processamento de 12 bits, 14 bits e 16 bits e trabalham em velocidades de 0kHz (ou DC) a 48MHz, usando ciclo de instrução mínimo de 4 períodos de clock, o que permite uma velocidade máxima de 10 MIPS. Há o reconhecimento de interrupções tanto externas como de periféricos internos. Funcionam com tensões de alimentação de 2 a 6V e os modelos possuem encapsulamento de 6 a 100 pinos em diversos formatos (SOT23, DIP, SOIC, TQFP, etc).

Os PICs podem ser programados em linguagem mnemônica (assembly) ou usando-se compiladores de linguagem de alto nível (Pascal, C, Basic) que geram um código em formato hexadecimal (Intel Hex format ou linguagem de máquina) para ser gravado na memória de programa desses microcontroladores. Para tal procedimento, utiliza-se um hardware especial (gravador) acoplado a um PC (com um PIC é possível rodar pequenos programas de computadores gravados neles). PICs com memória FLASH são altamente flexíveis na fase de desenvolvimento, pois permitem uma rápida alteração do código de programa. Como ferramentas de

desenvolvimento, encontram-se disponíveis: gravadores, depuradores, emuladores, placas de protótipos, etc.

2.4. *Transformada de Fourier*

A Transformada de Fourier (FT) é uma ferramenta largamente empregada em processamento de sinais. Denominada assim em homenagem ao físico francês Jean Baptiste Joseph Fourier (1768-1830), a FT decompõe um sinal em suas componentes elementares seno e cosseno. A FT aplicada a uma imagem no domínio espacial gera uma informação no domínio da frequência, em que cada ponto, definido por um vetor do tipo (k.cosseno, k.seno), representa uma dada frequência contida no domínio espacial da imagem [3].

As aplicações referentes à FT são inúmeras: filtragem, segmentação, reconhecimento de padrões, descrição de imagens, compressão e reconstrução constituem algumas delas.

A transformada de Fourier representa a soma infinita de uma série de formas de onda senoidais com diferentes amplitudes, fases e frequências. Pode ser utilizada em processamento digital de imagens quando se deseja conhecer frequências espaciais de um determinado padrão. No exemplo da Figura 2 é possível visualizar harmônicas de sinais senoidais e seu espectro de amplitudes.

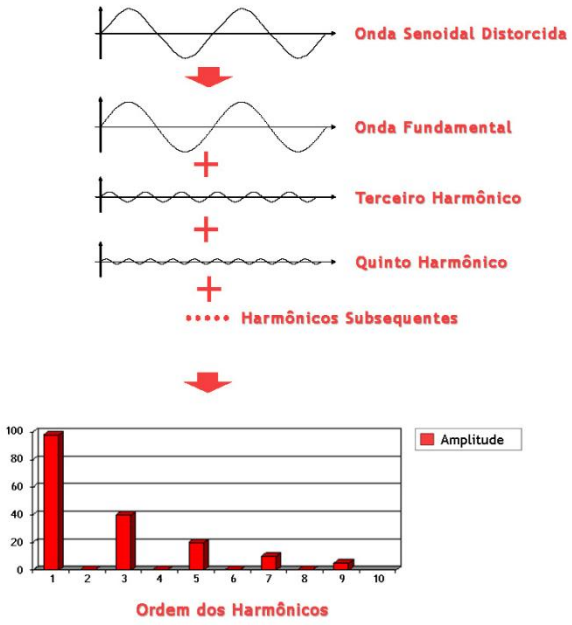

Figura 2 – Exemplo de Transformada de Fourier

Entretanto, a transformada pode ser utilizada também na reconstrução bi-dimensional de imagens em geral, por sua facilidade e rapidez de cálculo,

comparado com a resolução das equações de projeção algébrica, que consistem na montagem de uma matriz e sua resolução.

2.5. *DFT e FFT*

Em casos práticos, normalmente a avaliação da transformada de Fourier não é feita utilizando os procedimentos analíticos.

Muitas vezes não se dispõe de uma expressão analítica para a função que se deseja analisar o espectro.

A Transformada Discreta de Fourier (DFT) é muito usada no estudo do espectro de sinais e é determinada numericamente com o auxílio de computador digital.

J.W. Cooley (IBM) em colaboração com J.W. Tukey (Bell Labs) conseguiram uma revolução maior no tratamento digital de sinais em 1965 com a publicação da transformada rápida de Fourier, a FFT. Trata-se de um método engenhoso e altamente eficiente de reagrupar os cálculos dos coeficientes de uma DFT [1].

Muitos *softwares* dispõem de rotinas para o cálculo da FFT, por exemplo, MATEMATICA™, MATLAB™, MATHCAD™ etc.

Ao invés do cálculo da DFT diretamente pela definição, faz-se uso de um algoritmo conhecido como a FFT, que permite avaliar a DFT com menor esforço computacional. A FFT não é um tipo diferente de transformada e sim uma técnica que possibilita avaliar a DFT de forma mais rápida e eficiente.

As potencialidades da análise espectral, com base na Transformada de Fourier contínua, são bem estabelecidas. Ainda que teoricamente atrativo, o cálculo prático do espectro via a transformada clássica (empregando propriedades ou soluções analíticas) não é comum. Vários sinais de interesse (como voz, vídeo etc.) não possuem expressões analíticas para descrevê-los.

A redução acentuada no custo dos DSPs e o aumento da capacidade de processamento (e.g. milhões de MIPS- Mega Instruções Por Segundo), em conjunto com o aparecimento de novas técnicas eficientes para a avaliação de transformadas discretas (as ditas Transformadas Rápidas), vêm permitindo processar em tempo real muitos sinais [1].

Assim, as transformadas discretas vêm se firmando como ferramentas na análise espectral.

A maneira usual de lidar com sinais físicos é calcular a transformada através de um analisador de espectro (*hardware*).

Este instrumento realiza o cálculo da Transformada de Fourier e exibe o resultado em tela. Entretanto, com o desenvolvimento de técnicas de Processamento de Sinais DSP (e Processadores em *chip*), a DFT aparece como uma solução prática cada vez mais atrativa.

O tamanho dos analisadores de espectro não pode ser comparado ao tamanho reduzido dos DSPs, que adicionalmente permitem operação em grande velocidade, viabilizando numerosas aplicações.

O esforço computacional pode ser definido como o número máximo de operações elementares necessárias para resolver o problema. No caso da DFT, pode-se tratar da complexidade multiplicativa e complexidade aditiva i.e., número de multiplicações de ponto flutuante (respectivamente adições) necessárias para calculá-la. Tradicionalmente utiliza-se apenas a complexidade multiplicativa como o parâmetro mais importante.

2.6. *FFT- Transformada Rápida de Fourier (Cooley-Tukey base 2)*

A FFT foi implementada com o objetivo de diminuir a complexidade (temporal) necessária para calcular uma DFT (Transformada Discreta de Fourier), visando aplicações em tempo real.

O número de operações realizadas no cálculo da DFT através da definição é proporcional à N^2, i.é., para cada N valores de u, a expansão de F(u) requer N multiplicações complexas de x(n) por W^{ux} além de (N-1) adições dos resultados. A Eq. 1 apresenta o cálculo de F(u).

Alguns dos termos podem ser computados uma vez e armazenados para serem usados em operações futuras. Logo tais multiplicações de x(n) não são consideradas na implementação.

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) \cdot W_N^{ux} \qquad \text{Eq. 1}$$

Algumas decomposições apropriadas na Eq.1 podem tornar o número de multiplicações e adições proporcionais a $N.Log_2N$. Este procedimento é denominado de Transformada Rápida de Fourier ou FFT (Fast Fourier Transform).

2.7. *O Algoritmo FFT*

Por conveniência, a Eq. 1 pode ser reescrita sob a seguinte forma apresentada na Eq. 2:

Sendo: $W_N = \exp[-j \cdot 2 \cdot \frac{\pi}{N}]$

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) \cdot \exp[-j \cdot 2 \cdot \frac{\pi}{N}]^{ux} \qquad \text{Eq. 2}$$

Assumindo que N é da forma: $N=2^n$; onde n é inteiro positivo, então N pode ser expresso como: N=2*M; em que M é um inteiro positivo.

Substituindo esta relação na Eq. 1 tem-se a Eq. 3:

$$F(u) = \frac{1}{2 \cdot M}\sum_{x=0}^{2M-1} f(x) \cdot W_{2M}^{ux} \qquad \text{Eq. 3}$$

Dividindo-se o somatório em duas partes, relativa aos índices pares e ímpares chega-se a Eq. 4:

$$F(u) = \frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1} f(2x) \cdot W_{2M}^{u(2x)} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1) \cdot W_{2M}^{u(2x)}\right] \qquad \text{Eq. 4}$$

Como $W_{2M}^{u(2x)} = W_M^{ux}$, é possível reescrever a Eq. 4 na forma:

$$F(u) = \frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1} f(2x) \cdot W_M^{ux} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1) \cdot W_M^{ux}\right] \qquad \text{Eq. 5}$$

Agora considerando as seguintes DFTs:

$$f_{even}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x) \cdot W_M^{ux} \qquad \text{Eq. 6}$$

$$f_{odd}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1) \cdot W_M^{ux} \qquad \text{Eq. 7}$$

Para u=0,1,2,....,M-1.

Chega-se a uma relação que decompõe a DFT F(u) de comprimento 2M em duas DFTs Feven(u) e Fodd(u), ambas de comprimento M:

$$F(u) = \frac{1}{2}[f_{even}(u) + f_{odd}(u) \cdot W_{2M}^{u}] \quad \text{Eq. 8}$$

Observa-se também que: $W_M^{u+M} = W_M^u$ e $W_{2M}^{u+M} = -W_{2M}^u$, de modo que se segue da eq. 8,

$$F(u+M) = \frac{1}{2}[f_{even}(u) - f_{odd}(u) \cdot W_{2M}^{u}] \quad \text{Eq. 9}$$

Uma análise cuidadosa das equações revela algumas propriedades interessantes destas expressões.

Uma transformada de comprimento N pode ser computada dividindo a expressão original em duas partes, como indicado acima.

O cálculo da primeira metade de F(u) requer a avaliação de dois (N/2) pontos da transformada dada pelas equações Feven(u) e Fodd(u). Os resultados de $F_{even}(u)$ e $F_{odd}(u)$ são substituídos em Eq. 8 para obter F(u) para u=0,1,2,...., (N/2-1). Os outros valores seguem diretamente da Eq. 9 sem avaliações de transformações adicionais. Para examinar as implicações computacionais deste procedimento, considere m(n) e a(n) como o número de multiplicações complexas e adições, respectivamente, exigidas para implementação.

Suponha o número de amostras igual a 2n, onde n é um inteiro positivo.

Suponha agora que n=1. A transformada de dois pontos requer a avaliação de F(0), pois F(1) segue da equ. 8. Para obter F(0) precisa-se do cálculo de Feven(0) e Fodd(0).

Neste caso M=1 e Feven(u) e Fodd(u) são a transformada de apenas um ponto, porque a transformada de Fourier de um único ponto é o próprio ponto, por isso nenhuma multiplicação ou adição são efetuadas para obtenção de Feven(0) e Fodd(0).

Apenas uma multiplicação de Fodd(0) por W_2 e uma adição são requeridas para a obtenção de F(0).

Então F(1) segue da eq. 8 com uma adição a mais (considera-se, em termos de complexidade, que subtração e adição são idênticas). Como o produto Fodd(0) por W_2 já foi previamente computado, o número total de operações requeridas para uma transformada de dois pontos consiste em m(1)=1 multiplicação e a(1)=2 adições.

O próximo valor permitido para n é 2. De acordo com o desenvolvimento acima, a transformada de quatro pontos pode ser dividida em duas partes. A primeira metade de F(u) requer avaliação da transformada de dois pontos, como nas equ. Feven(u) e Fodd(u). Para M=2, a transformada de dois pontos requer m(1) multiplicações e a(1) adições, logo a avaliação destas duas equações requer um total de 2m(1) multiplicações e 2a(1) adições. Duas multiplicações e adições adicionais são necessárias para obter F(0) e F(1) a partir da Eq. 8. Isto porque já tinham sido computados Fodd(u).W_{2M} para u={0,1}.

Mais duas adições fornecem F(2) e F(3). A complexidade total é então m(2)=2m(1)+2 e a(2)=2a(1)+4.

Quando n é igual a 3, a transformada de quatro pontos é considerada na avaliação de Feven(u) e Fodd(u). Estas requerem 2m(2) multiplicações e 2a(2) adições. Mais quatro multiplicações e mais oito adições levam à transformada completa. O total é então m(3)=2m(2)+4 e a(3)=2a(2)+8.

Continuando com este argumento para qualquer valor de inteiro positivo de n encontram-se expressões de recursividade para o número de multiplicações e adições exigidas para a implementação de FFT:

$$m(n) = 2m(n-1) + 2^{n-1} \qquad \text{Eq. 10}$$

$$a(n) = 2a(n-1) + 2^{n} \qquad \text{Eq. 11}$$

Neste caso, m(0)=0 e a(0)=0, porque a transformada de um único ponto não requer qualquer adição ou multiplicação.

Por indução, o número de multiplicações complexas a adições requeridas na implementação do algoritmo FFT é, respectivamente:

$$m(n) = \frac{1}{2} 2^n \log_2 2^n \qquad \text{Eq. 12}$$

, ou seja,

$$m(n) = \frac{1}{2} N \log_2 N \qquad \text{Eq. 13}$$

E

$$a(n) = 2^n \log_2 N \qquad \text{Eq. 14}$$

, ou seja,

$$a(n) = N \log_2 N \qquad \text{Eq. 15}$$

2.8. *A FFT inversa*

Para calcular uma "FFT inversa" usa-se praticamente o mesmo algoritmo "FFT direta", com algumas modificações. As relações entre a DFT e a DFT inversa:

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) exp\left[-j2\pi \frac{ux}{N}\right] \qquad \text{Eq. 16}$$

E

$$f(x) = \sum_{u=0}^{N-1} f(u) exp\left[\frac{j2\pi ux}{N}\right] \qquad \text{Eq. 17}$$

Tomando o complexo conjugado dos dois lados e dividindo ambos os membros por N, obtém-se:

$$\frac{1}{N} f'(x) = \frac{1}{N}\sum_{u=0}^{N-1} F'(u) exp\left[\frac{-j2\pi ux}{N}\right] \qquad \text{Eq. 18}$$

Comparando este resultado com a Eq. 16, observa-se que o lado direito está na forma da transformada de Fourier.

Se na entrada de um algoritmo projetado para computar a transformada FFT aplica-se F*(u), a saída será f*(x)/N. Tomando o complexo conjugado e multiplicando por N, tem-se o sinal f(x) inverso desejado.

2.9. *Implementação da FFT*

O principal ponto das implementações diz respeito ao arranjo dos dados de entrada visando à aplicação da divisão de uma transformada em termos de outras de menor comprimento.

O procedimento de reordenação pode ser ilustrado através de um exemplo simples (N=8).

Entrada {f(0), f(1), f(2), f(3), f(4), f(5), f(6), f(7)}.

Separando as amostras de argumentos pares e ímpares:

Pares {f(0), f(2), f(4), f(6)}.

Ímpares {f(1), f(3), f(5), f(7)}.

Contudo, para reaplicar o procedimento, as componentes pares e ímpares devem ser novamente separadas:

Pares {f(0), f(4)}. {f(1), f(5)}.

Ímpares {f(2), f(6)}. {f(3), f(7)}.

REARRANJO:

Entrada {f(0), f(1), f(2), f(3), f(4), f(5), f(6), f(7)}.

{f(0), f(4), f(2), f(6), f(1), f(5), f(3), f(7)}.

Tabela 2 - Arranjo

Numeração original			Arranjo original	Numeração bits invertidos			Arranjo reordenado
0	0	0	f(0)	0	0	0	f(0)
0	0	1	f(1)	1	0	0	f(4)
0	1	0	f(2)	0	1	0	f(2)
0	1	1	f(3)	1	1	0	f(6)
1	0	0	f(4)	0	0	1	f(1)
1	0	1	f(5)	1	0	1	f(7)
1	1	0	f(6)	0	1	1	f(3)
1	1	1	f(7)	1	1	1	f(5)

A existência de algoritmos rápidos é um fator decisivo nas inúmeras aplicações em tempo real da DFT. Devido ao algoritmo Cooley- Tukey base 2, frequentemente usam-se comprimentos que são potência de 2.

A Tabela 3 mostra uma cota inferior na complexidade para determinar uma DFT de comprimento N, denotada m(DFT(N)).

Tabela 3 – Comprimento da DFT

N	u(DFT(N)	N	u(DFT(N)
4	0	120	120
8	2	240	274
12	4	720	986
24	12	...	...
48	38	65520	108594
60	56	...	...

Existe um grande número de diferentes algoritmos rápidos propostos na literatura, incluindo Cooley-Tukey, Good-Thomas, PFA (Algoritmo de fatores primos) e Algoritmo de Winograd-Fourier (WFTA).

3. MATERIAIS E MÉTODOS

3.1. *Descrição Geral do Sistema*

O sistema é um osciloscópio integrado a um analisador de espectro, podendo executar funções de verificação da composição sinais elétricos, principalmente tensões provenientes de uma rede de energia residencial ou industrial. Essa análise é feita através da coleta de amostras do sinal na entrada do circuito, sendo adequado posteriormente para a leitura da entrada analógica do pic, onde será coletado, armazenado e enviado ao *software* de visualização através da comunicação serial entre o controlador e o PC. Este sinal é visualizado em dois gráficos simultâneos, um descrevendo a curva real e o outro o espectro no domínio da frequência, determinando sua composição através de um algoritmo que implementa uma FFT (Fast Fourier Transform).

Como o projeto é uma sequência de um trabalho anterior, primeiramente foi necessário conhecer todos os métodos utilizados na execução, além dos problemas, limitações e particularidades do sistema como um todo, para então começar a implementar melhorias. A placa utilizada foi a mesma do projeto original, mas com algumas alterações para adequar a nova concepção de projeto, direcionada agora para análise do sinal amostrado. Como existiam escalas fixas, que exigiam que o usuário tivesse conhecimento da forma de onda medida, foi necessário calcular o valor ideal de intervalo entre as amostras para cobrir a faixa de frequência desejada, já que o sinal poderia ter várias componentes de frequência.

A Figura 3 apresenta o diagrama simplificado do sistema:

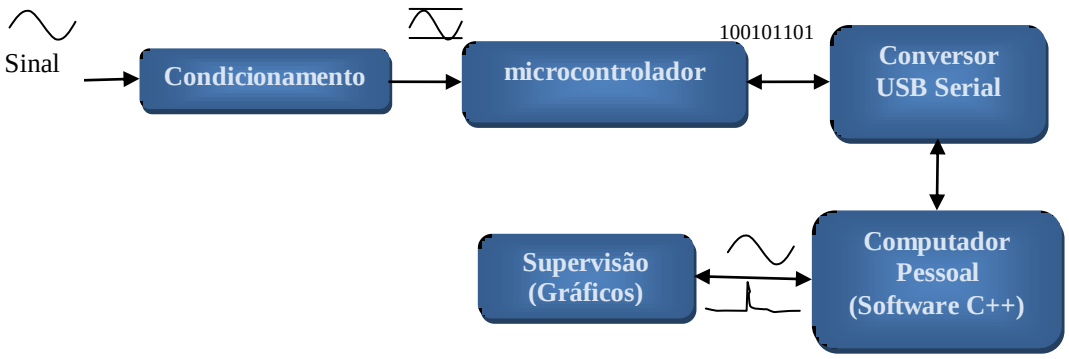

Figura 3 – Diagrama simplificado do Sistema

3.2. *Projeto inicial*

O projeto inicial compreendia um osciloscópio com ajuste de escalas de tempo e tensão de acordo com o sinal de entrada. Havia cinco escalas de tempo e três de tensão que eram gerenciadas pelo software supervisório do PC. Este tinha a função de enviar as escalas através da serial para o controlador e imprimir os dados no domínio tempo no gráfico. Existia, ainda, a possibilidade de ajuste de trigger automático ou manual atuando sobre as amostras recebidas pela serial, ou seja, através da exclusão de algumas delas para posicionamento adequado no gráfico. Outra característica do projeto anterior era a demora para atualização do gráfico com os dados coletados em virtude do protocolo utilizado para troca das informações, havendo tempos de espera, além do envio individual de cada dado coletado.

3.3. *Melhorias implementadas*

3.3.1. Análise Espectral

Uma das principais modificações propostas neste trabalho é a inserção de uma análise espectral, sendo necessário, primeiramente, conhecer a capacidade que o projeto anterior tinha de processar ao mesmo tempo sinais de baixa e alta frequência e, com base nisso, definir o que poderia ser realizado para melhorar este desempenho. Para isso, foram definidas faixas de frequências, através da modificação do intervalo entre as amostras que podem representar, da melhor maneira possível, o sinal que esta sendo analisado. Ainda, foi necessário utilizar o número máximo de amostras que o microcontrolador pode processar e armazenar antes de iniciar o processo de exibição e processamento destes dados. A análise espectral foi realizada através da implementação em linguagem C++ de um

algoritmo que executa uma FFT (Fast Fourier Transform), que será vista com mais detalhes em seguida.

3.3.2. Atualização

Outra modificação implementada foi o aumento da taxa de atualização dos gráficos na tela do computador, através de uma melhoria no desempenho do protocolo de comunicação entre o computador e o microcontrolador por meio da comunicação serial. No protocolo do projeto original havia intervalos fixos de espera, o que impedia o melhor aproveitamento da velocidade da rede de comunicação, principalmente porque cada dado era enviado individualmente ocasionando a multiplicação destes intervalos fixos de envio pela serial. Para acelerar esta troca de informações foi necessário modificar este processo.

No programa atual é realizado o envio da totalidade da informação pelo microcontrolador, utilizando o buffer da serial, com a leitura e filtragem desses dados pelo *software* do PC, o que o torna mais ágil. Os tempos fixos de espera foram substituídos por testes que verificam constantemente se a informação já esta disponível, aproveitando de uma forma melhor a velocidade do canal serial.

3.3.3. Informações dos Espectros

Outra alteração proposta e inserida no sistema foi a exibição de informações referentes ao sinal que está sendo amostrado, tanto do osciloscópio quanto do espectro de frequências.

Informações como valor médio e eficaz, frequências e amplitudes mais significativas, que compõem os sinais, são visualizadas no *software* supervisório ajudando na interpretação do gráfico. Estas informações podem ser visualizadas a cada coleta de amostras executada pelo *software* do PIC. Além disso, ao posicionar o mouse sobre as curvas, é possível obter os valores das coordenadas de cada ponto do gráfico que, no novo programa, possui o eixo do tempo e da frequência em ms e Hz, respectivamente, e não mais em função do número de amostras como no projeto original.

- Valor Eficaz – Foi calculado utilizando um número de iterações relacionadas ao número total de amostras. A cada iteração o valor instantâneo do sinal é elevado ao quadrado e somado ao valor resultante da soma anterior. Ao final do total de iterações, este é divido pelo total de amostras, resultando o valor eficaz do conjunto

de amostras coletado. O cálculo matemático executado por este algoritmo pode ser visualizado na Eq. 19:

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{i=0}^{N} x_i^2} = \sqrt{\frac{x_1^2+x_2^2+\cdots x_N^2}{N}} \quad \text{Eq. 19}$$

- Valor médio – Segue a mesma lógica do cálculo do valor eficaz utilizando a equação Eq. 20:

$$x_{médio} = \frac{1}{N}\sum_{i=0}^{N} x_i = \frac{x_1+x_2+\cdots x_N}{N} \quad \text{Eq. 20}$$

- Valor máximo – É o máximo valor do total de amostras.
- Frequências importantes (harmônicas) – Foram calculadas três frequências principais de acordo com a posição da maior amplitude no espectro, esta sendo a frequência fundamental do total de amostras. Com base nessa posição foram localizados os picos das outras duas frequências que possuem maior amplitude e que são harmônicas da fundamental.
- Amplitudes das harmônicas – A cada valor de frequência selecionado foi armazenado o valor de amplitude respectivo, sendo também possível visualizá-lo a cada leitura das amostras.

3.3.4. Gravação em arquivo

Esta opção possibilita que os dados coletados possam ser gravados no computador para futura análise. Assinalando no *software* a seleção "gravar", todos os dados coletados a partir desse momento estarão sendo guardados em um arquivo de texto, cujo nome será a data selecionada pelo usuário do sistema.

Quando houver a necessidade de verificar novamente estes dados, basta assinalar a opção "Ver histórico" que todos os dados serão carregados e analisados novamente. A visualização do histórico é executada através de uma barra de rolagem horizontal que permite navegar pelas informações do arquivo. Através dela é possível verificar cada grupo de amostras por vez, reproduzindo a mesma análise executada no momento da gravação. Ainda há a possibilidade de rolar automaticamente os dados para que possam ser visualizados da mesma forma que foram gravados assinalando a opção "Play". A Figura 4 exibe estas opções.

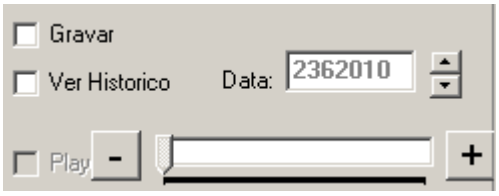

Figura 4 - Sistema de visualização do histórico

Para determinar a escala de frequência do momento da gravação, foram utilizados caracteres extras, para que no momento da visualização do histórico seja possível conhecer o intervalo de tempo entre as amostras e imprimi-las adequadamente.

3.3.5. Intervalo entre as amostras

Como o sinal que será analisado é composto, teoricamente, por várias frequências, foi necessário adequar o intervalo entre as amostras de acordo com a faixa de frequência mínima e máxima que está sendo amostrada para se obter uma melhor exatidão no resultado final da análise. Para isso, foram criadas cinco faixas de frequências que determinam a precisão da análise em função do tipo de sinal.

3.3.6. Aumento de escala

A faixa de entrada foi aumentada para que fosse possível a leitura de sinais de tensão comuns no ambiente residencial e industrial, com um limite de 350 V. Em virtude disso, foi adicionado um transformador na entrada do sistema, que executará a função de adequar o sinal a ser analisado para os limites atuais do hardware. Este transformador foi acoplado por meio de uma placa extra que possui duas conexões, uma na entrada da placa do microcontrolador e outra para conexão em tomadas de energia. Este acoplador tem a função de rebaixar, com eficiência, apenas tensões de sinais senoidais, em virtude de sua constituição física. Uma foto deste dispositivo é vista na Figura 5.

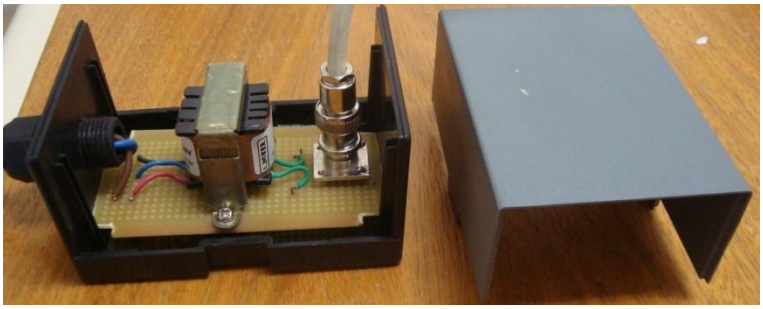

Figura 5 – Placa de interligação com a rede elétrica

3.3.7. *Trigger* na origem da coleta

Uma alteração importante, inserida, foi a retirada da opção de *trigger* automático ou manual do sinal no *software* do supervisório. Isto fazia com que os dados coletados não fossem exibidos na sua totalidade em função da necessidade de iniciar a exibição após o valor determinado. Para sempre exibir todos dados amostrados este "gatilho" foi realizado no momento da coleta das informações possibilitando a utilização de todas as amostras na análise do sinal.

3.3.8. Ajuste automático do gráfico

Devido à limitação da opção de auto ajuste na ferramenta da biblioteca PlotLab, responsável pela impressão dos dados no gráfico, foi necessário criar um algoritmo que, em função dos dados amostrados, altera automaticamente os limites dos gráficos, possibilitando sempre um aproveitamento total da área de visualização.

Em especial para os limites de tempo e frequência ocorre um auto-ajuste de acordo com a escala de frequência selecionada. Para isto foi calculada uma constante para cada uma das faixas de acordo com a Tabela 4:

Tabela 4 - Calculo dos limites dos gráficos

Faixa	Intervalo (us)	Leitura(us)	Total(us)	intervalo (ms)	intervalo (ms)	fmin	fmax	multiplicador	intervalo (Hz)
0	0	42,6	42,6	13,632	14	146,713615	7335,680751	23,47	7511
1	100	42,6	142,6	45,632	46	43,82889201	2191,4446	7,01	2244
2	200	42,6	242,6	77,632	78	25,76257214	1288,128607	4,12	1319
3	300	42,6	342,6	109,632	110	18,2428488	912,1424402	2,92	934
4	400	42,6	442,6	141,632	142	14,12110258	706,0551288	2,26	723

Como os limites dos gráficos configurados só podiam ser números inteiros, os mesmos foram arredondados, mas para o cálculo das informações foi utilizado uma constante multiplicadora calculada utilizando duas casas. Este multiplicador é uma constante, resultado da divisão do número de amostras pelo intervalo calculado, em milissegundos, do gráfico no domínio tempo.

3.4. *Hardware*

3.4.1. Entrada de sinal

O sinal é captado através de uma ponteira de osciloscópio comum que pode ser acoplada ao sistema através de um conector BNC apropriado. Este sinal passa por um condicionamento para poder ser lido pela entrada analógica do PIC. Além disso, existem faixas de amplitude de sinal que podem ser selecionadas de acordo com o máximo valor que será amostrado, controladas através do acionamento de relés e configuradas pelo *software* do PC. Em especial, para a maior escala de

tensão, foi confeccionada uma placa que possui um transformador de tensão eficaz 220V/12V, responsável pela adequação de sinais de tensão da rede elétrica com o objetivo de preservar a placa original, não realizando alterações significativas.

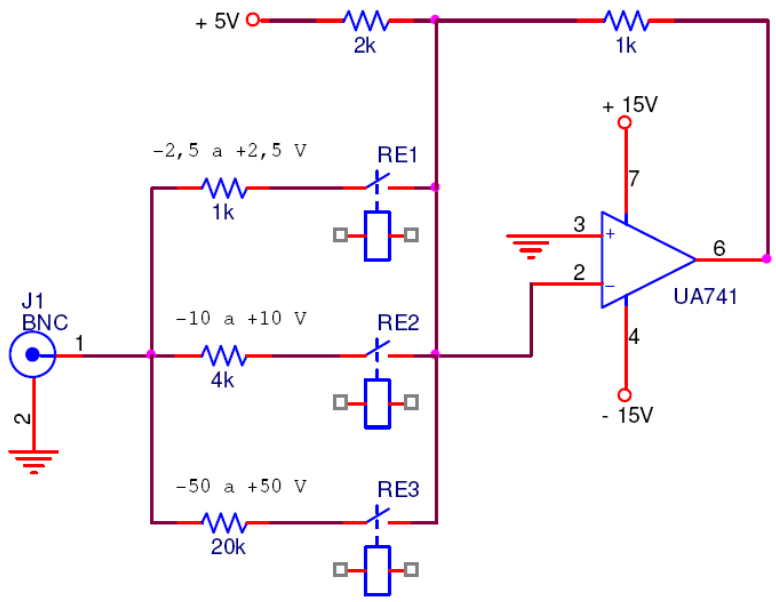

Figura 6 - Entrada do sinal

3.4.2. Microcontrolador

Foi utilizado o PIC 16F877A programado em linguagem C no *software* MPLAB e compilado pelo *software* CCS. Esta é a mesma plataforma do projeto anterior. Na Figura 7 pode ser vista um foto da placa utilizada.

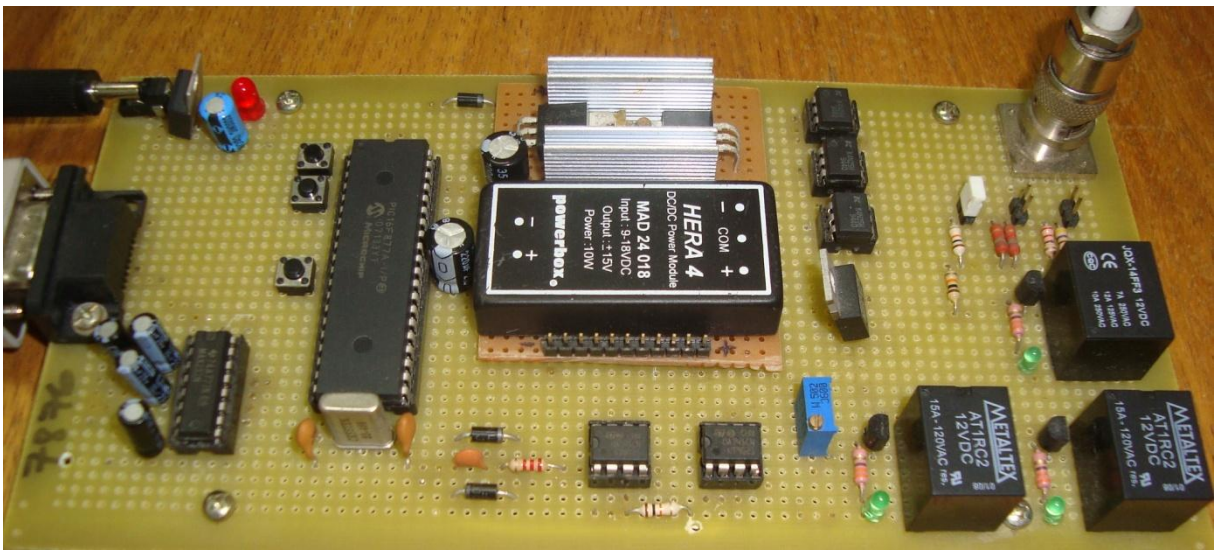

Figura 7 – Foto da placa do PIC

Para gravação do programa no microcontrolador foi utilizado um *software* que envia o algoritmo compilado no formato .hex no software MPLAB, através da serial, para a memória flash do microcontrolador, não necessitando que haja um

dispositivo de gravação especial. Na Figura 8 é visto a tela principal do software responsável pela gravação do firmware no microcontrolador.

Figura 8 – Software de gravação do programa no pic

3.4.3. Comunicação Serial

A comunicação serial é ponto importante para este trabalho, pois ela é responsável por toda troca de informações entre o sistema supervisório localizado no PC e o *firmware* que está gravado na memória do pic. A porta serial desejada pode ser alterada diretamente no *software* do PC, sendo isso essencial para a utilização por qualquer usuário. Para este projeto foi utilizado um conversor serial-usb com o objetivo de utilização da porta usb do notebook utilizado como parte do projeto. Este conversor possibilita que em qualquer computador se consiga facilmente efetuar uma comunicação com o sistema, visto que este é o padrão atual de portas seriais.

3.5. *Softwares*

3.5.1. Microcontrolador

O software foi remodelado para atender as especificações do projeto atual, sendo utilizada a mesma plataforma de programação, apenas com a atualização do software MPLAB, utilizado para programação em linguagem C do programa do controlador, agora utilizado na versão 8.50. As modificações principais foram a compactação do código e a remodelação do tempo entre as amostras, para atender determinadas faixas de frequências e com o objetivo de possibilitar uma analise espectral tanto de baixas como de altas frequências com boa exatidão, além de melhorias no protocolo de comunicação com o *software* supervisório. A Figura 9

mostra a tela de programação do software MPLAB que foi atualizado para a versão 8.5 em relação ao projeto original.

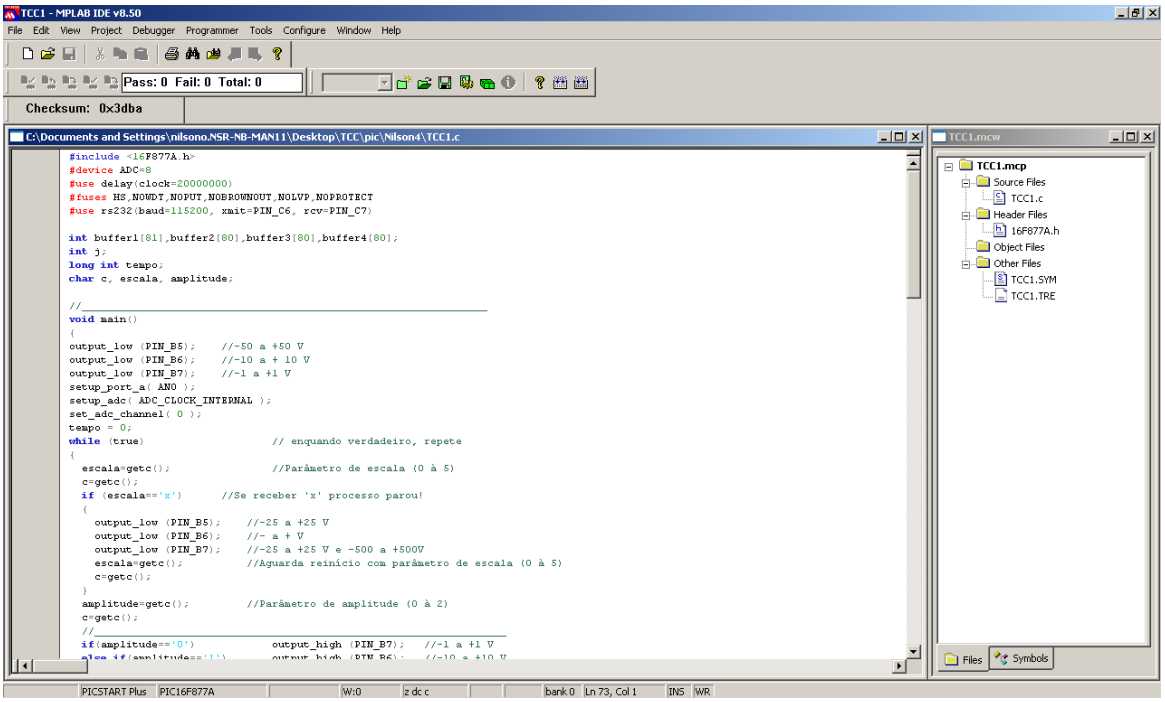

Figura 9 – Software MPLAB 8.5

3.5.1. Fluxograma do software do microcontrolador

Na Figura 10 pode ser visualizado o fluxograma do firmware do microcontrolador. Este programa foi modificado para atender, principalmente, as melhorias no desempenho da comunicação.

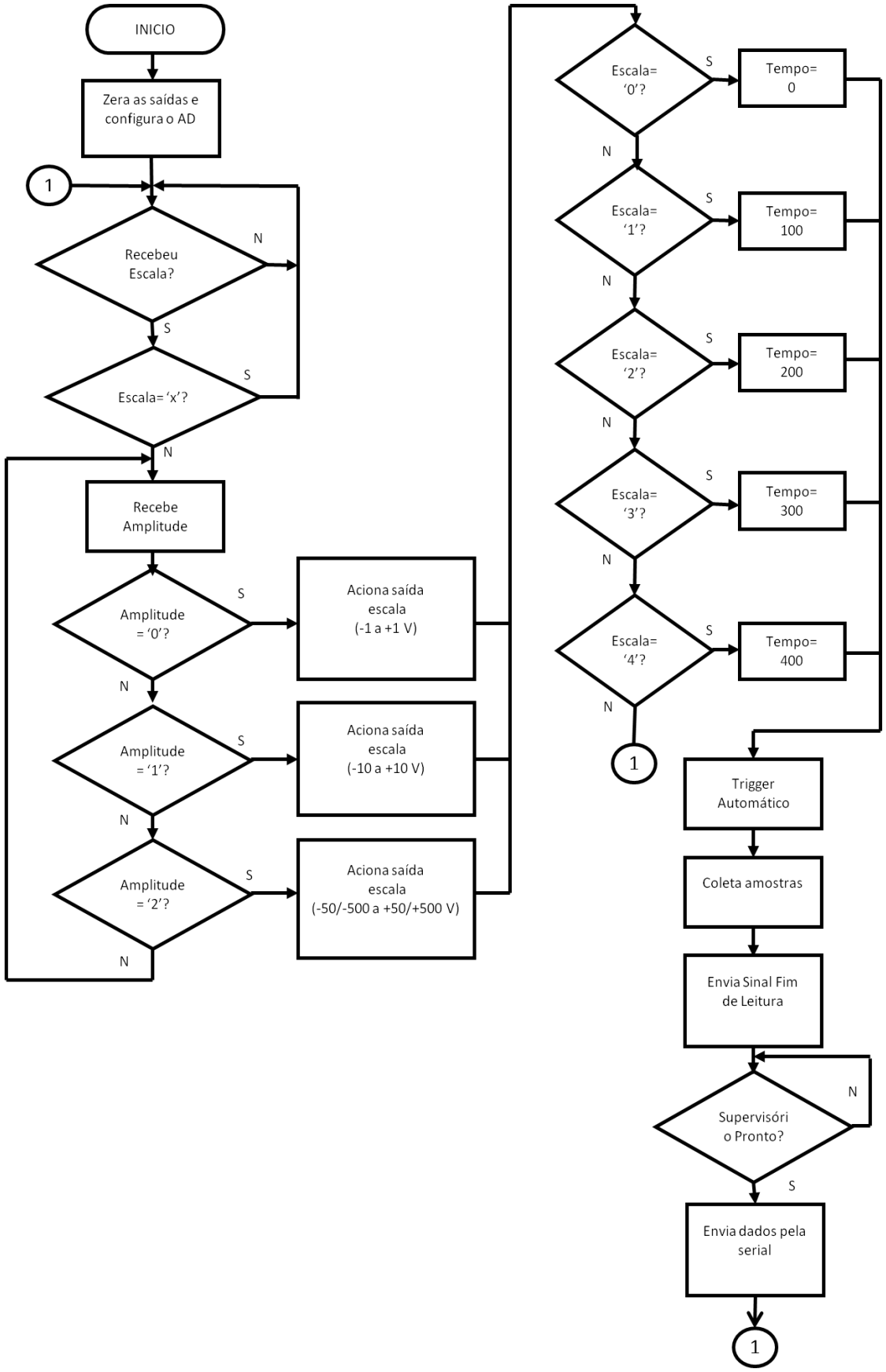

Figura 10 - Fluxograma do firmware

3.5.2. Software Supervisório

Foi desenvolvido na plataforma Builder C++, sendo que é onde foram realizadas as modificações mais significativas. A principal delas foi a introdução de uma analise espectral através de um algoritmo que calcula a transformada discreta de Fourier de forma mais simples, chamada FFT (Fast Fourier Transform) ou Transformada Rápida de Fourier. Este algoritmo realiza algumas iterações, manipulando os dados e desenhando o gráfico em função da quantidade de amostras disponíveis. Foram realizados, também, cálculos para exibição de algumas informações do sinal que foi captado. A execução do software se dá através de um *timer* e uma *thread* (forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas ao mesmo tempo). A função de ler os dados da serial fica com o *thread* e a de imprimir as amostras nos gráficos está dentro do *timer*. As duas ações somente são habilitadas a partir do acionamento do botão "Iniciar Análise" que se encontra na tela principal do software supervisório que é vista na Figura 11.

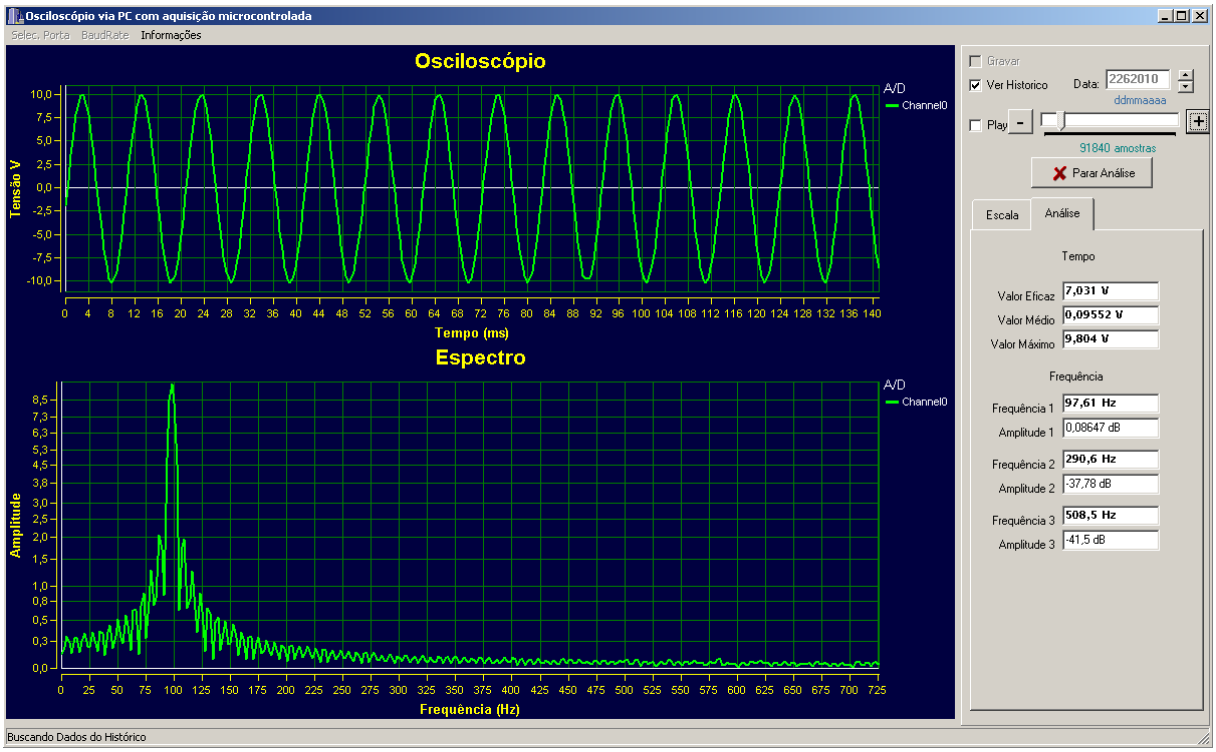

Figura 11 – Software Supervisório

3.5.3. Fluxograma C++

Na Figura 12 é exibido o fluxograma simplificado do software do supervisório. O mesmo apresenta como são as trocas de informações entre este software e o firmware gravado no microcontrolador.

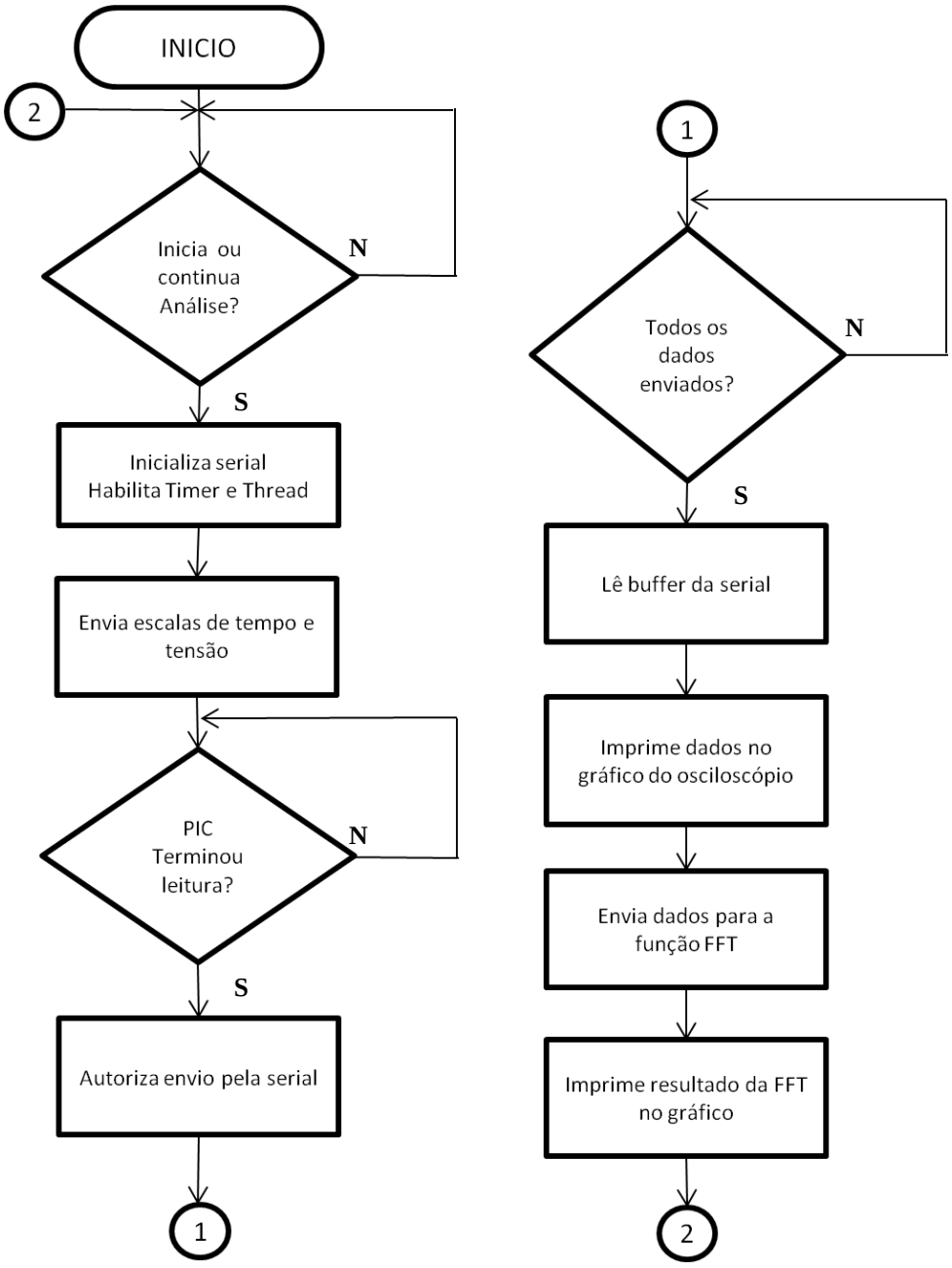

Figura 12 – Fluxograma do sistema supervisório

3.5.4. Algoritmo FFT

A geração de um gráfico com o espectro amplitudes através da implementação de uma FFT pode ser realizada com a aplicação de uma quantidade muito grande de algoritmos que conseguem realizar de forma rápida uma Transformada Discreta de Fourier. O método escolhido para este trabalho é baseado

no método mais utilizado e antigo, o Cooley-Tukey. Depois de realizado estudo sobre o método, foi realizada uma adaptação do sistema conforme o número de amostras e outras particularidades do analisador espectral do projeto.

4. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Os resultados encontrados, durantes os testes efetuados no sistema, foram considerados satisfatórios para os limites fisicos do *hardware* utilizado.

4.1. *Modelos*

Foram realizadas simulações através da utilização de modelos matemáticos para gerar sinais senoidais puros no interior do programa. Estes modelos simulam de maneira fiel, por exemplo, o comportamento de uma onda puramente senoidal, sem qualquer interferência. Dessa forma foi possível obter a resposta do sistema como um todo, definindo seu comportamento para sinais ideais.

A Figura 13 mostra o gráfico de um sinal senoidal puro de 383,6 Hz simulado matematicamente dentro do algoritmo. Esta frequência foi calculada baseada na expressão da Eq. 21, inserida no programa:

$$a = sen(30 \cdot 2 \cdot \pi \cdot \frac{i}{320}) \qquad \text{Eq. 21}$$

Onde,

a é a amostra atual;

i é a iteração atual.

A expressão simula 30 períodos incompletos, sendo que, com a faixa 30 – 1200 Hz selecionada, o intervalo no domínio tempo para 29 períodos, verificado no gráfico, é de 75,6 ms. Dividindo-se este valor por 29 obtêm-se o período do sinal, que equivale a uma frequência de 383,6 Hz.

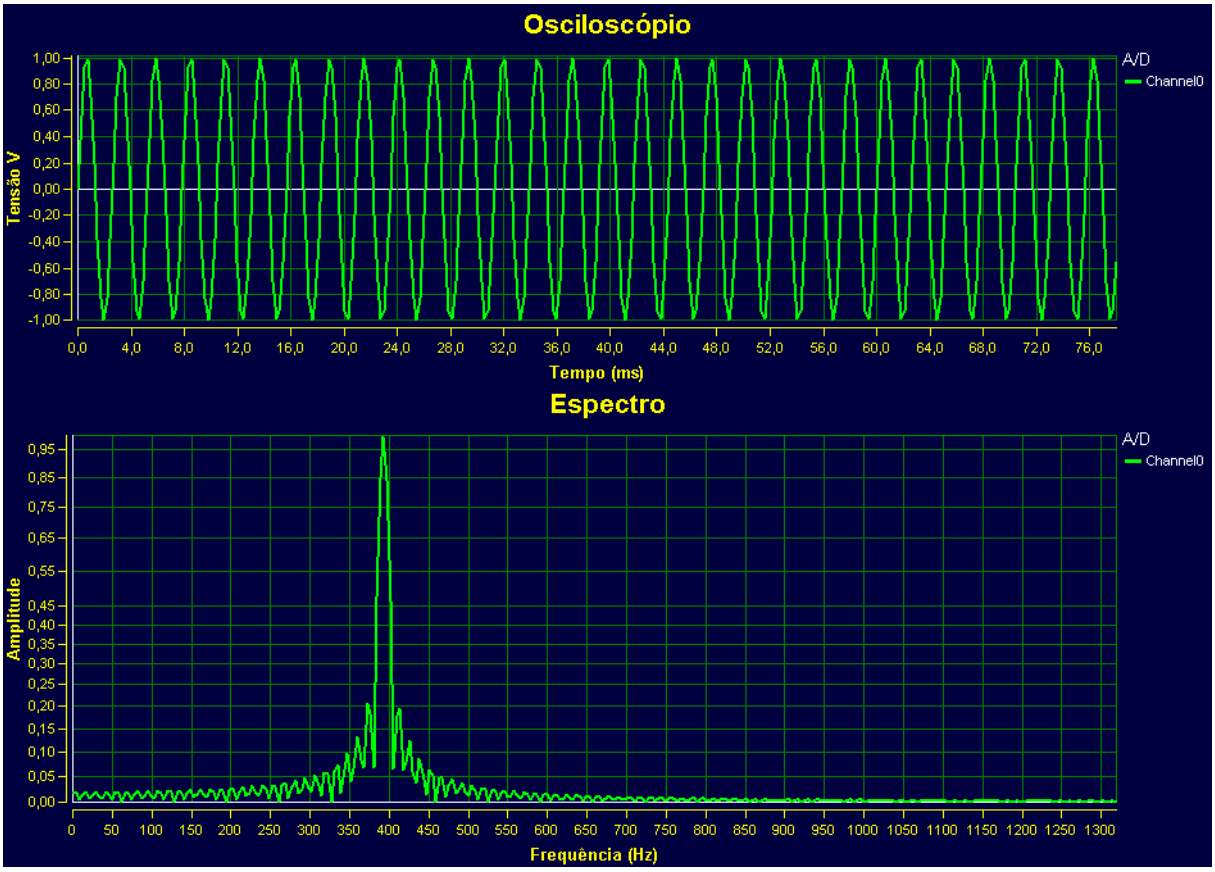

Figura 13 – Espectro senoidal simulado

4.2. *Metodologia dos Testes Integrados no Sistema Físico*

Os testes finais efetuados para ajuste do sistema incluíam um gerador de sinais e um osciloscópio calibrado da marca Fluke modelo 123. Dessa forma foi possível injetar na entrada da placa um sinal conhecido que deveria ser reproduzido da mesma maneira no gráfico do osciloscópio e mostrando a frequência correta no espectro, resultado do algoritmo FFT. Foi realizado ajuste no intervalo entre as amostras, em todas as faixas, para compensar o tempo de execução das instruções durante a coleta dos dados. Com o sistema de gravação de dados foi possível guardar vários sinais conhecidos, e analisá-los novamente após algum ajuste no sistema.

4.2.1. Testes na rede elétrica

Foram realizados testes acoplando o sistema a uma rede elétrica cuja frequência fundamental é 60 hertz. Foi possível analisar algumas situações em uma residência e em uma indústria, sendo que em ambos os casos foram detectadas distorções na forma de onda analisada. No primeiro caso notou-se uma distorção que aproxima a senóide a uma onda triangular como na Figura 14:

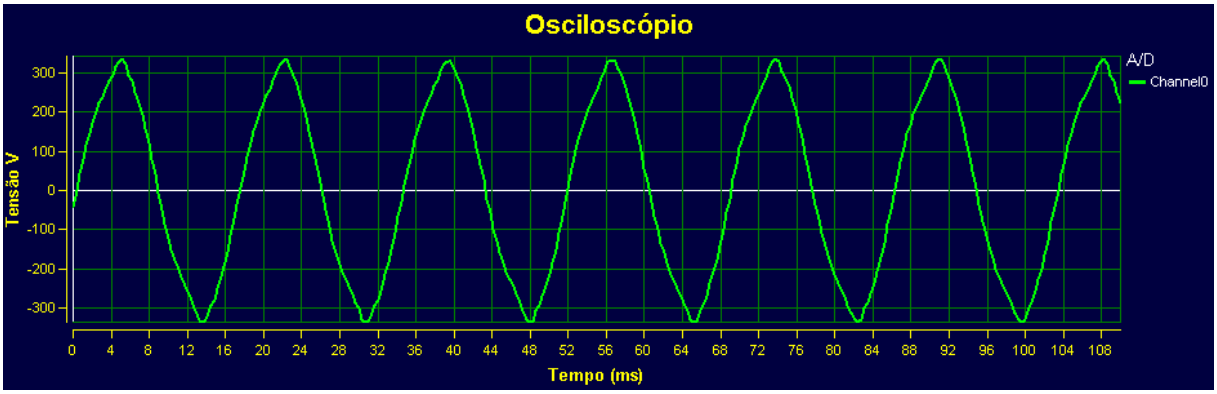

Figura 14 – Sinal de uma rede elétrica residencial

O espectro de frequência identifica a componente principal e ainda mostra que há uma amplitude considerável na terceira e quinta harmônicas, como visualizado na Figura 14:

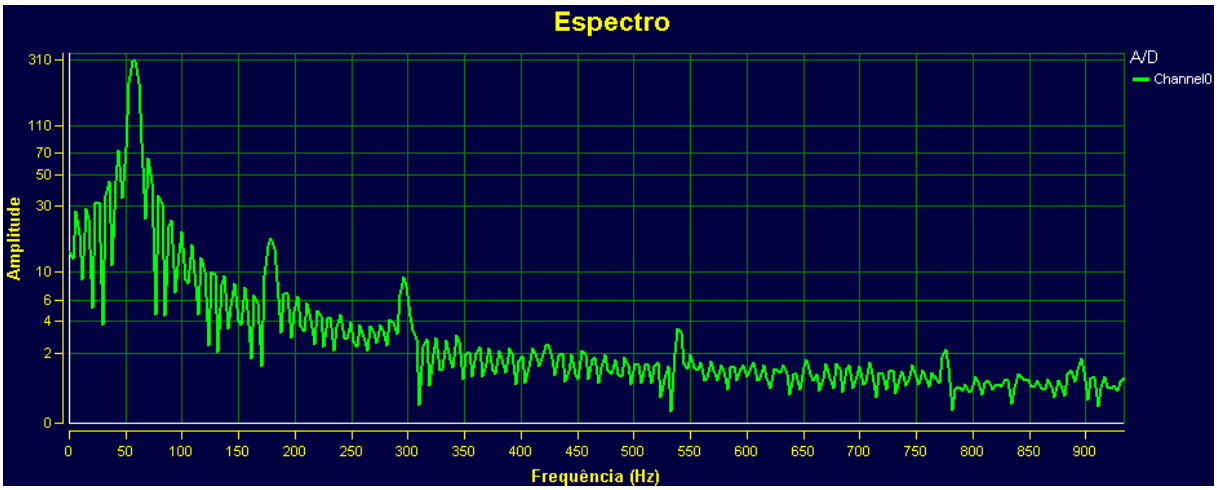

Figura 15 – Análise de uma rede elétrica residencial

As informações mostradas na Figura 16 quantificam estes dados:

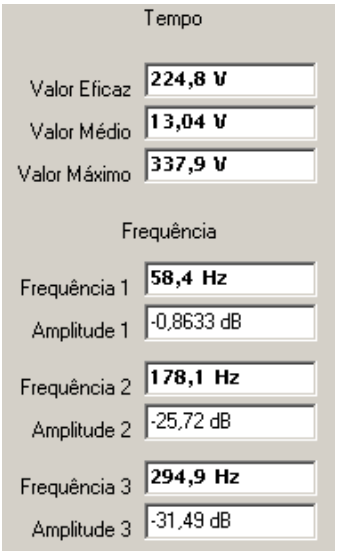

Figura 16 – Cálculos do conjunto de amostras

Outra situação, ainda em uma residência, foi a verificação das interferências que um chuveiro eletrônico introduz em uma rede de energia. Os gráficos da Figura 17 mostram o sinal afetado e o espectro de frequências, evidenciando o nível de influência das harmônicas introduzidas no sistema.

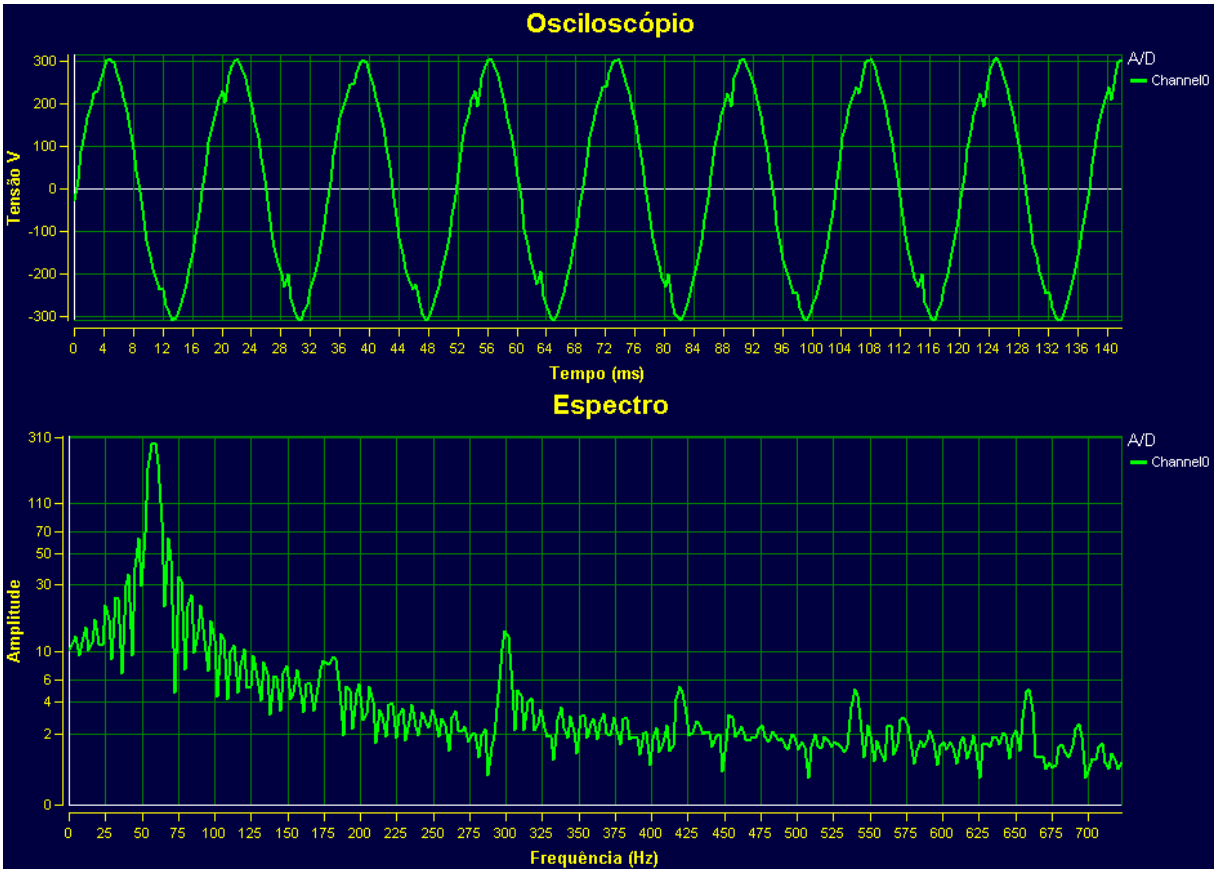

Figura 17 – Interferência causada por chuveiro eletrônico

As informações coletadas foram as mostradas na Figura 18:

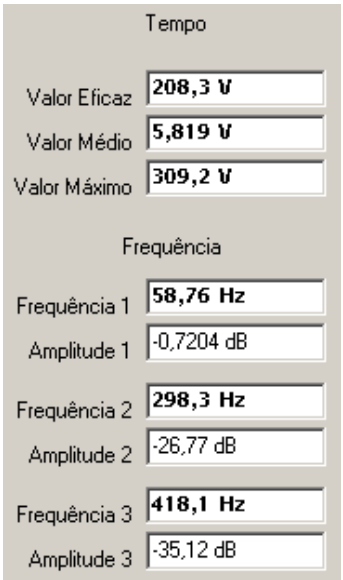

Figura 18 – Informações dos espectros

Já em uma indústria, foram coletadas amostras em uma sala de escritório onde também foram localizadas harmônicas, mas a forma de onda apresentou-se bem mais limpa que na análise residencial, como visualizado na Figura 19:

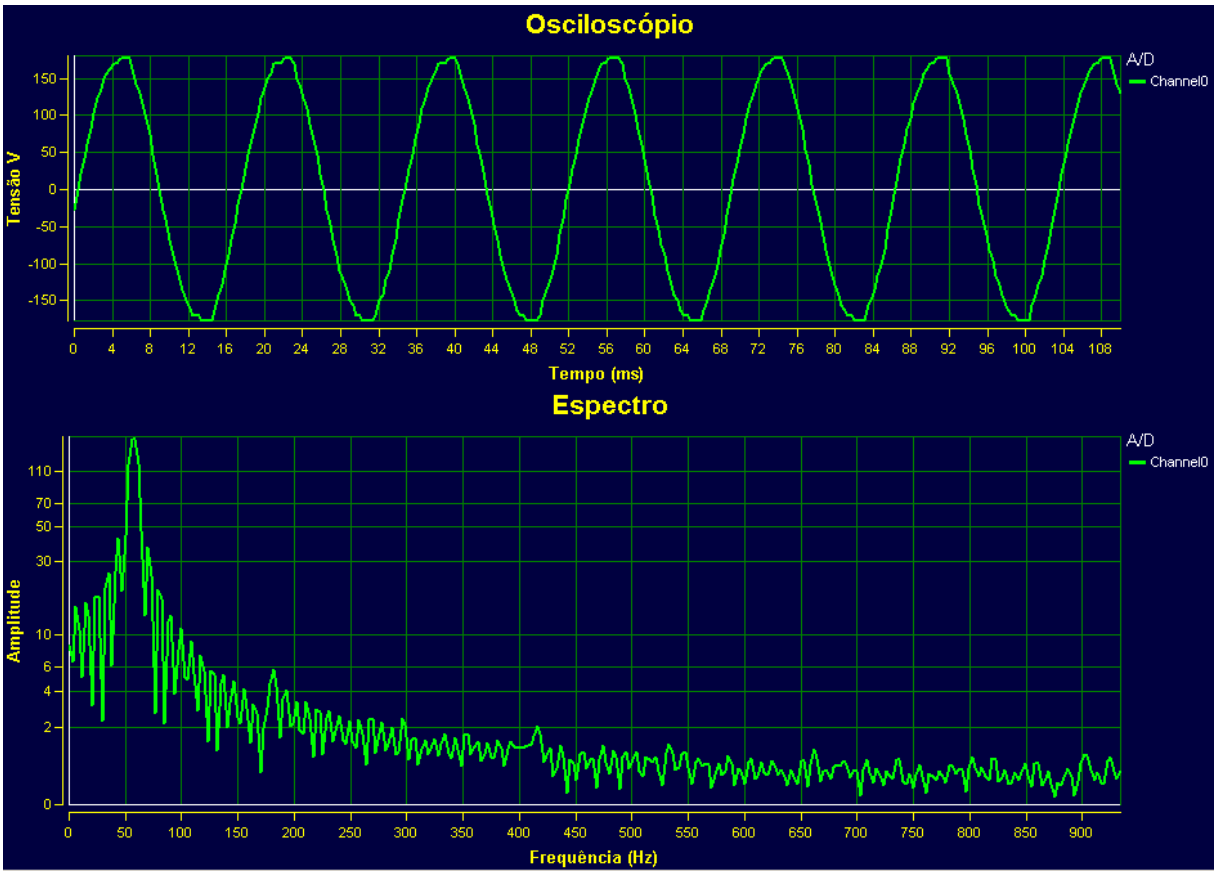

Figura 19 – Sinal coletado em um escritório

As informações calculadas, para este conjunto de amostras, são as exibidas na Figura 20 e uma foto do projeto montado no momento dos testes, na Figura 21:

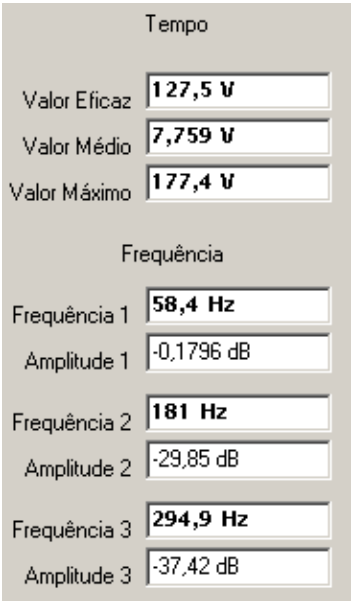

Figura 20 – Informações calculadas

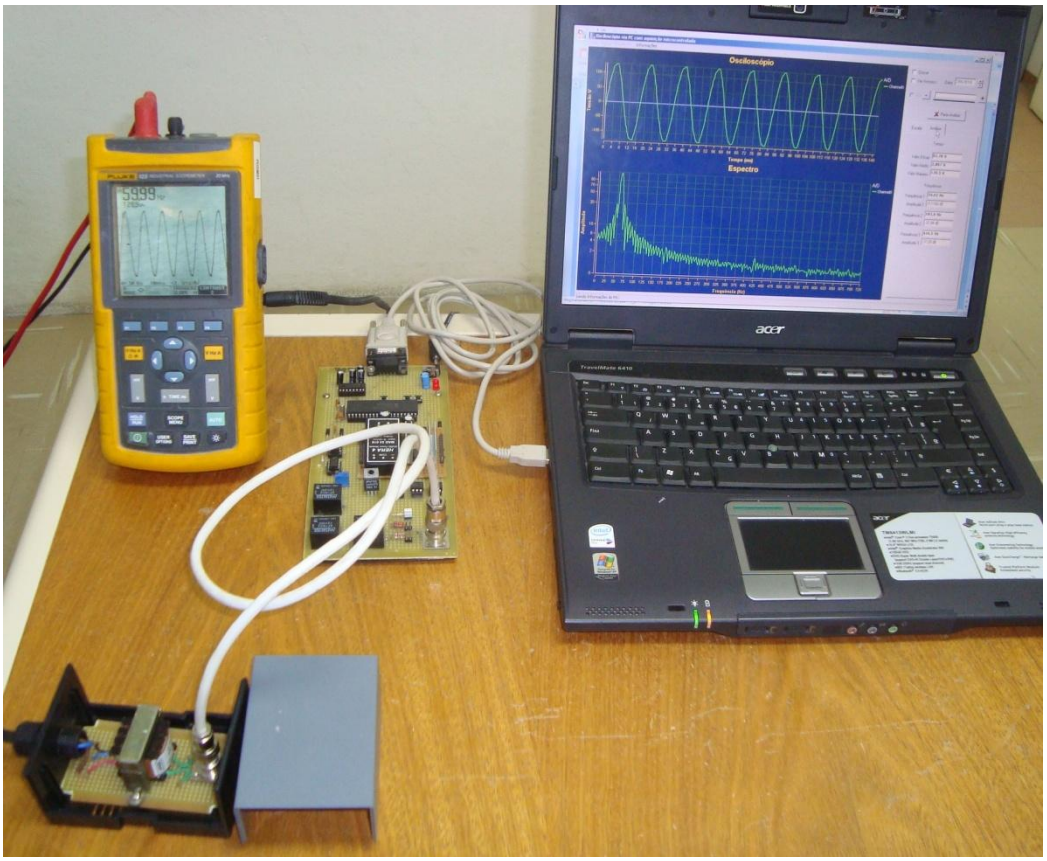

Figura 21 – Testes executados em um escritório

4.2.2. Calibração das faixas de frequências

Para ajuste do osciloscópio e do analisador de espectro do projeto foi utilizado um gerador de frequência e um osciloscópio calibrado. O procedimento adotado foi a utilização de uma tabela de escala de padrões onde foram sendo ajustadas as frequências e anotando-se os valores de ambos os instrumentos. Após o preenchimento completo da tabela foram ajustados os parâmetros para aproximação dos valores coletados. Ao mesmo tempo em que os dados foram anotados, foram também gravados em arquivo, possibilitando a execução dos mesmos testes após o ajuste do sistema para verificação do desvio final. A Tabela 5 mostra a primeira calibração, antes do ajuste:

Tabela 5 – Dados antes do ajuste

	Faixa 15 - 700			Faixa 20 - 900			Faixa 30 - 1200			Faixa 45 - 2000			Faixa 150 - 7000		
	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro
Antes do Ajuste	15,42	13,56	-12,06%	20,1	17,52	-12,84%	29,97	24,72	-17,52%	45	35	-22,22%	150,2	117,3	-21,90%
	30,01	27,12	-9,63%	59,98	55,48	-7,50%	60,4	56,7	-6,13%	61,3	56	-8,65%	791,2	774,5	-2,11%
	61,46	58,76	-4,39%	99,56	96,36	-3,21%	99,61	94,76	-4,87%	200,4	189	-5,69%	1667	1643	-1,44%
	99,6	97,18	-2,43%	200	195,6	-2,20%	303,8	296,6	-2,37%	594,5	574	-3,45%	2563	2535	-1,09%
	199,3	196,6	-1,35%	299,2	294,9	-1,44%	593,8	585	-1,48%	893,4	868	-2,84%	3575	3544	-0,87%
	300,8	298,3	-0,83%	499,6	496,4	-0,64%	798,3	786,9	-1,43%	1317	1288	-2,20%	4960	4905	-1,11%
	499,6	497,2	-0,48%	700,3	695	-0,76%	998,8	984,7	-1,41%	1611	1575	-2,23%	5933	5891	-0,71%
	697	693,8	-0,46%	900	896,4	-0,40%	1203	1187	-1,33%	2012	1967	-2,24%	6950	6900	-0,72%
		correção	-0,46%		correção	-0,40%		correção	-1,33%		correção	-2,24%		correção	-0,72%

Em seguida, as constantes e limites foram ajustados de acordo com o erro encontrado na última medição de cada faixa de acordo com a Tabela 6.

Tabela 6 – Ajustes dos parâmetros

Constante frequência	2,26	2,27		2,92	2,93		4,12	4,18		7,01	7,17		23,47	23,64
Limite Frequência	723	726		934	938		1319	1337		2244	2294		7511	7565
Limite Tempo	141,632	141		109,632	109		77,632	77		45,632	45		13,632	14

Após alteração dos parâmetros no *software*, foi verificado o histórico gravado em arquivo no momento da coleta das varias faixas e gerada a Tabela 7 com as novas informações, demonstrando o efeito da correção.

Tabela 7 – Dados após ajuste

	Faixa 15 - 700			Faixa 20 - 900			Faixa 30 - 1200			Faixa 45 - 2000			Faixa 150 - 7000		
	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro	Padrão	Projeto	Erro
Após do Ajuste	15,42	13,62	-11,67%	20,1	17,58	-12,54%	29,97	25,08	-16,32%	45	35,85	-20,33%	150,2	118,2	-21,30%
	30,01	27,24	-9,23%	59,98	55,67	-7,19%	60,4	57,8	-4,30%	61,3	57,36	-6,43%	791,2	780,1	-1,40%
	61,46	59,02	-3,97%	99,56	96,69	-2,88%	99,61	96,14	-3,48%	200,4	193,6	-3,39%	1667	1655	-0,72%
	99,6	97,61	-2,00%	200	196,3	-1,85%	303,8	301	-0,92%	594,5	587,9	-1,11%	2563	2553	-0,39%
	199,3	197,5	-0,90%	299,2	295,9	-1,10%	593,8	593,6	-0,03%	893,4	889,1	-0,48%	3575	3570	-0,14%
	300,8	299,9	-0,30%	499,6	498,1	-0,30%	798,3	798,4	0,01%	1317	1315	-0,15%	4960	4941	-0,38%
	499,6	499,4	-0,04%	700,3	697,3	-0,43%	998,8	999	0,02%	1611	1613	0,12%	5933	5934	0,02%
	697	696,9	-0,01%	900	899,5	-0,06%	1203	1204	0,08%	2012	2015	0,15%	6950	6950	0,00%

Verificou-se que o final de escala de cada faixa ficou com uma ótima exatidão em função de a correção ser baseada no seu erro. Em contrapartida ainda existem erros grandes em baixas frequências, em função do pequeno número de pontos no início da escala do espectro de frequências. Foram pintadas na cor verde, na tabela, todas as faixas com erro inferior a 10 %, identificando as faixas de melhor exatidão. Deve-se levar em consideração esta exatidão quando se tem idéia do sinal que esta sendo amostrado. Um exemplo representativo ocorre quando se está analisando um sinal de uma rede elétrica cuja frequência fundamental é 60 Hz. Caso seja utilizada a faixa 15 a 700 Hz, o analisador terá a condição de identificar as componentes até a décima primeira harmônica (660 Hz) e com erro de 3,97 % na fundamental, enquanto que na faixa de 20 a 900 Hz o erro na fundamental seria de 7,19%, mas, nesse caso, analisando até a décima quinta harmônica.

5. CONSIDERAÇÕES FINAIS

5.1. *Problemas Encontrados*

Os maiores problemas encontrados estavam relacionados com a adequação do projeto anterior em outro computador e com outras versões de *software*. O *software* Borland C++ Builder, utilizado na versão 5 no projeto anterior, quando utilizado na versão 6, mostrou incompatibilidades com a biblioteca empregada na aplicação inicial. Com isso foi necessário manter a versão original para implementação das melhorias. Outro problema encontrado foi com a comunicação serial, pois a mesma apresentava problemas quando usada com a porta serial do computador disponível para programação, sendo necessária a utilização de um conversor usb-serial para interligação com a placa do controlador para dar prosseguimento no trabalho.

5.2. *Avaliação dos Objetivos Propostos*

Os objetivos propostos foram concluídos na sua totalidade, além disso, foram incluídas outras melhorias que não constavam no escopo inicial. Foi executada satisfatoriamente a transformada rápida de Fourier para geração de um espectro no domínio frequência, isto evidenciado pelos testes executados e comparação com modelos matemáticos. O protocolo de comunicação criado atendeu as expectativas quanto ao aumento da velocidade, além do acréscimo de confiabilidade já que o *software* não corre o risco de ficar travado em alguma condição em função da demora no envio de alguma informação.

5.3. *Conclusões*

O projeto executado atendeu todas as expectativas quanto aos objetivos iniciais, tornando-o um produto melhor em relação a sua situação original e deixando-o pronto para ser utilizado em instituições de ensino que queiram demonstrar de forma prática a teoria de análise de sinais, obtendo também as diferenças entre as curvas resultantes do cálculo teórico e o obtido em um ambiente real.

5.4. *Sugestões para Trabalhos Futuros*

Em trabalhos futuros poderá ser implementada uma análise em tempo real através da utilização de um controlador com um maior poder de processamento e através de uma rede de maior velocidade, além disso, poderão ser adicionados novos canais com mais bits de resolução.

6. REFERÊNCIAS

[1] UFPE. DFT e FFT. Pernambuco, 2004. Disponível em: <www2.ee.ufpe.br/codec/FFT.pdf>. Acesso em: 05/05/2010.

[2]UNICID. Linguagens de Programação. São Paulo, 2007. Disponível em: <www.dee.feis.unesp.br/.../c_04_linguagem_de_programacao.pdf >. Acesso em: 16/04/2010.

[3]UFEPEL. Transformada de Fourier. Rio Grande do Sul, 2003. Disponível em: <atlas.ucpel.tche.br/~tst/fourier.html>. Acesso em: 16/04/2010.

[4]WIKIPÉDIA, RS-232. Disponível em:

< http://pt.wikipedia.org/wiki/RS-232>. Acesso em 10/04/2010.

OBRAS CONSULTADAS

EMBREE, Paul M. - C language algorithms for digital signal processing - Englewood Cliffs: Prentice Hall, 1991. 456 p.

COCIAN, Luis Fernando Espinosa – Manual da Linguagem C - 1ª. Ed. - Local: Editora da Ulbra, 2004. 500 p.

PEREIRA, Fábio – PIC Programação em C – 1ª Ed. – São Paulo: Érica, 2003

SOUZA, David José de – Conectando o PIC 16F877A – 3ª. Ed. - Local:Érica, 2007

CORMEN, Thomas H. – Introduction to Algorithms – 1ª Ed. – New York, McGraw-Hill, 1998

CORREIA, Juarez Corrêa – Programas aplicativos ao processamento de sinais em tempo discreto – 1ª Ed. – Porto Alegre, EDIPUCRS, 1996

HAYES, Monson H. - Schaum's outline of theory and problems of digital signal processing - New York: McGraw-Hill, 1999

WENTZ, Daniel Alvienes – Osciloscópio via PC com aquisição microcontrolada – Canoas: Universidade Luterana do Brasil, 2009

GLOSSÁRIO

Trigger – Trigger ou gatilho é um recurso de programação executado sempre que o evento associado ocorrer. É muito utilizada para ajudar a manter a consistência dos dados ou para propagar alterações em um determinado dado de uma tabela para outras.

APÊNDICE A – PROGRAMA DO PIC

```
#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[81],buffer2[80],buffer3[80],buffer4[80];
int j;
long int tempo;
char c, escala, amplitude;

//_____________________________________________________________
void main()
{
output_low (PIN_B5);  //-50 a +50 V
output_low (PIN_B6);  //-10 a + 10 V
output_low (PIN_B7);  //-1 a +1 V
setup_port_a( AN0 );
setup_adc( ADC_CLOCK_INTERNAL );
set_adc_channel( 0 );
tempo = 0;
while (true)                                    // enquando verdadeiro, repete
{
  escala=getc();                        //Parâmetro de escala (0 à 5)
  c=getc();
  if (escala=='x')                  //Se receber 'x' processo parou!
  {
    output_low (PIN_B5);        //-25 a +25 V
        output_low (PIN_B6);  //- a + V
        output_low (PIN_B7);  //-25 a +25 V e -500 a +500V
        escala=getc();                  //Aguarda reinício com parâmetro de escala (0 à 5)
        c=getc();
  }
  amplitude=getc();                     //Parâmetro de amplitude (0 à 2)
  c=getc();
  //_____________________________________________________________
  if(amplitude=='0')                    output_high (PIN_B7);//-1 a +1 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 e -500 a +500V
  //_____________________________________________________________
  if(escala=='0')           tempo=0;            //escala 150 - 7000 Hz
  else if(escala=='1')          tempo=100;              //escala 45 - 2000 Hz
  else if(escala=='2')          tempo=200;              //escala 30 - 1200 Hz
  else if(escala=='3')          tempo=300;              //escala 20 - 900 Hz
  else if(escala=='4')          tempo=400;              //escala 15 - 700 Hz
  //_____________________________________________________________
  buffer1[1]=128;
  buffer1[2]=128;
```

```
  while (buffer1[1]>=128 || buffer1[2]<128) // Trigger
  {
     buffer1[1]=buffer1[2];
           buffer1[2]=read_adc();
           delay_us(tempo);
  }
  buffer1[0]=escala;
  ///////Leitura do AD///////
  for(j=3; j<81; j++)
  {
     buffer1[j]=read_adc();
           delay_us(tempo);
  }
  for(j=0; j<80; j++)
  {
     buffer2[j]=read_adc();
           delay_us(tempo);
  }
  for(j=0; j<80; j++)
  {
     buffer3[j]=read_adc();
           delay_us(tempo);
  }
  for(j=0; j<80; j++)
  {
           buffer4[j]=read_adc();
     delay_us(tempo);
  }
  printf("blz");   //Envia sinal de fim de leitura
  //_____________________________________________________________
  c=getc();
  ///////Envia dados para a serial/////
  for(j=0; j<81; j++)
  {
     printf ("%3u",buffer1[j]);
  }
  for(j=0; j<80; j++)
  {
     printf ("%3u",buffer2[j]);
  }
  for(j=0; j<80; j++)
  {
     printf ("%3u",buffer3[j]);
  }
  for(j=0; j<80; j++)
  {
     printf ("%3u",buffer4[j]);
  }
}
}
```

APÊNDICE B – PROGRAMA BUILDER C++

Edit1.cpp

```
//---------------------------------------------------------------------
#include <vcl\vcl.h>
#include <time.h>
#include <stdio.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit3.h"
#include "Serial.h"
#include <math.h>
#include<stdlib.h>
#include<dos.h>
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SLScope"
#pragma link "CSPIN"
#pragma resource "*.dfm"
float constante=7;
typedef struct { // estrutura que guarda números complexos
float real, imag;
}COMPLEX;
void fft(COMPLEX *x, int m); // protótipo da função fft
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 = "COM6";                    //Porta inicial
amostras="1";                      //Frequencia inicial 1KHz
amplitude="2";                      //Amplitude inicial de -50 a 50 V
fator = 254;                       //Amplitude inicial de -50 a 50 V
Amplitude->ItemIndex=3;
Frequencia->ItemIndex=0;
LeituraThread = new TLeitura(false);    //Ativa a Thread
LeituraThread->Suspend();              //Pausa a Thread
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
   Edit1->Text = "x";
   bTransmiteMensagem(Form1->Edit1->Text.c_str());
```

```
    bEncerraSerial();
    LeituraThread->Suspend();        //Pausa a Thread
    Timer1 -> Enabled = false;
    leitura=1;
    bEncerraSerial();
    LeituraThread->Terminate();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FrequenciaChange(TObject *Sender)
{
if (Frequencia->ItemIndex==0)    //1Hz
{
amostras="0";
temp_amost=20;
constante=23.47;
Osciloscopio->XAxis->Max->Value=14;
Osciloscopio->XAxis->Max->Tick->Value=14;
Espectro->XAxis->Max->Value=7511;
Espectro->XAxis->Max->Tick->Value=7511;
}
if (Frequencia->ItemIndex==1)    //10Hz
{
amostras="1";
temp_amost=2;
constante=7,01;
Osciloscopio->XAxis->Max->Value=46;
Osciloscopio->XAxis->Max->Tick->Value=46;
Espectro->XAxis->Max->Value=2244;
Espectro->XAxis->Max->Tick->Value=2244;
}

if (Frequencia->ItemIndex==2)    //100Hz
{
amostras="2";
temp_amost=285;
constante=4.12;
Osciloscopio->XAxis->Max->Value=78;
Osciloscopio->XAxis->Max->Tick->Value=78;
Espectro->XAxis->Max->Value=1319;
Espectro->XAxis->Max->Tick->Value=1319;
}

if (Frequencia->ItemIndex==3)    //500Hz
{
amostras="3";
temp_amost=36;
constante=2.92;
Osciloscopio->XAxis->Max->Value=110;
Osciloscopio->XAxis->Max->Tick->Value=110;
Espectro->XAxis->Max->Value=934;
Espectro->XAxis->Max->Tick->Value=934;
}

if (Frequencia->ItemIndex==4)    //1000Hz
{
amostras="4";
temp_amost=36;
constante=2.26;
Osciloscopio->XAxis->Max->Value=142;
Osciloscopio->XAxis->Max->Tick->Value=142;
```

```
Espectro->XAxis->Max->Value=723;
Espectro->XAxis->Max->Tick->Value=723;
}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AmplitudeChange(TObject *Sender)
{
if (Amplitude->ItemIndex==0)
{
amplitude="0";                  //De -1 a +1 V
fator = 0.45;
Osciloscopio->YAxis->Min->Value=-1;
Osciloscopio->YAxis->Max->Value=+1;
}
if (Amplitude->ItemIndex==1)
{
amplitude="1";                  //De -10 a +10 V
fator = 4;
Osciloscopio->YAxis->Min->Value=-10;
Osciloscopio->YAxis->Max->Value=+10;
}
if (Amplitude->ItemIndex==2)
{
amplitude="2";                  //De -50 a +50 V
fator = 20;
Osciloscopio->YAxis->Min->Value=-50;
Osciloscopio->YAxis->Max->Value=+50;
}
if (Amplitude->ItemIndex==3)
{
amplitude="2";                  //De -50 a +50 V
fator = 348;// correto =366.66666;
Osciloscopio->YAxis->Min->Value=-50;
Osciloscopio->YAxis->Max->Value=+50;
}
}
//---------------------------------------------------------------------------

// Botão iniciar

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

if(StartStop->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;
        StartStop->Kind=bkCancel;
        StartStop->Caption="Parar Análise";
        Amplitude->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;
        StartStop->Kind=bkOK;
        StartStop->Caption="Iniciar Análise";
        Amplitude->Enabled=true;
        SelecPorta1->Enabled=true;              //habilita seleção de porta
        Baud1->Enabled=true;
}
}
//---------------------------------------------------------------------------
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";
```

```
}
//---------------------------------------------------------------------------
// Timer1
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
int i, j, trigger_point,aa;
double nivel;
if (VerHistorico->Checked==true && Vazio->Text=="não")leitura=1;
if (leitura==1)
{
   float Buffer5[ 320 ],Buffer6[ 320 ]; // buffer que guardará os valores do módulo do sinal
   COMPLEX *samp; // buffer que guarda os valores para a realização da fft
   int m = 10, fft_length;
   fft_length = 1 << m; // obtenção do tamanho da fft
   samp = (COMPLEX *) calloc(fft_length, sizeof(COMPLEX)); // alocação de memória para o
buffer samp
   unsigned char RxChar[ 1024 ]; // buffer que receberá os dados da serial
   float b,amax=0,amin=0,amed=0,f1=0,f2=0,f3=0,a1=0,a2=0,a3=0;
   int iamax=0,iamin=0,iamed=0;
   struct  time t;
   float ValorE=0,ValorM=0,vmax=-200,vmin=200,v1max;
   AnsiString dado="1";
   // transferência do módulo do sinal da fft
   if (Gravar->Checked==true && Gravado->Text!="ok" && VerHistorico->Checked==false)
   {
     for ( j=0; j < 321; j++ )
     {
       Dado->Text=Buffer[j];
       if(Dado->Text!="")
       {
         ListaDados->Items-> Add(Dado->Text);
         Registros->Caption=IntToStr(ListaDados->Items->Count-(ListaDados->Items-
>Count/321));
         Dado->Clear();
       }
     }
     ListaDados->Items->SaveToFile(DataGravada->Text + ".Dot");
     Gravado->Text="ok";
   }
   BarraHistorico->Max=ListaDados->Items->Count-320;
   if (VerHistorico->Checked==true)
   {
      Form1->Frequencia->ItemIndex=StrToInt(ListaDados->Items->Strings[BarraHistorico-
>Position].c_str())-48;
      if (Frequencia->ItemIndex==0)    //1Hz
      {
        amostras="0";
        temp_amost=20;
        constante=23.47;
        Osciloscopio->XAxis->Max->Value=14;
        Osciloscopio->XAxis->Max->Tick->Value=14;
        Espectro->XAxis->Max->Value=7511;
        Espectro->XAxis->Max->Tick->Value=7511;
      }
      if (Frequencia->ItemIndex==1)    //10Hz
      {
        amostras="1";
        temp_amost=2;
        constante=7,01;
        Osciloscopio->XAxis->Max->Value=46;
```

```
        Osciloscopio->XAxis->Max->Tick->Value=46;
        Espectro->XAxis->Max->Value=2244;
        Espectro->XAxis->Max->Tick->Value=2244;
      }
      if (Frequencia->ItemIndex==2)    //100Hz
      {
        amostras="2";
        temp_amost=285;
        constante=4.12;
        Osciloscopio->XAxis->Max->Value=78;
        Osciloscopio->XAxis->Max->Tick->Value=78;
        Espectro->XAxis->Max->Value=1319;
        Espectro->XAxis->Max->Tick->Value=1319;
      }
      if (Frequencia->ItemIndex==3)    //500Hz
      {
        amostras="3";
        temp_amost=36;
        constante=2.92;
        Osciloscopio->XAxis->Max->Value=110;
        Osciloscopio->XAxis->Max->Tick->Value=110;
        Espectro->XAxis->Max->Value=934;
        Espectro->XAxis->Max->Tick->Value=934;
      }
      if (Frequencia->ItemIndex==4)    //1000Hz
      {
        amostras="4";
        temp_amost=36;
        constante=2.26;
        Osciloscopio->XAxis->Max->Value=142;
        Osciloscopio->XAxis->Max->Tick->Value=142;
        Espectro->XAxis->Max->Value=723;
        Espectro->XAxis->Max->Tick->Value=723;
      }
      for (int i = 0; i < 320; i++)
      {
        Buffer6[ i ] =StrToFloat(ListaDados->Items->Strings[i+1+BarraHistorico-
>Position].c_str());
      }
      Osciloscopio->Channels->Channels[ 0 ]->Data->SetYData( Buffer6, 320 ); //
transferência dos dados para o gráfico
      for(int i = 0; i < 320; i ++)
      {
        samp[ i ].real = Buffer6[i];
        if (Buffer6[ i ]>vmax) vmax=Buffer6[ i ];
        if (Buffer6[ i ]<vmin) vmin=Buffer6[ i ];
        ValorE=ValorE+pow(Buffer6[ i ],2);
        ValorM=ValorM+Buffer6[ i ];
      }
      if (Play->Checked==true)
      {
        if (BarraHistorico->Position<=BarraHistorico->Max-320)
        BarraHistorico->Position=BarraHistorico->Position+321;
      }
    }
    else
    {
      for (int i = 0; i < 320; i++) Buffer5[i]=Buffer[i+1];
      Osciloscopio->Channels->Channels[ 0 ]->Data->SetYData( Buffer5, 320 ); //
transferência dos dados para o gráfico
```

```
    for(int i = 1; i < 321; i ++)
    {
      samp[ i ].real = Buffer[i];
      if (Form1->Buffer[ i ]>vmax) vmax=Form1->Buffer[ i ];
      if (Form1->Buffer[ i ]<vmin) vmin=Form1->Buffer[ i ];
      ValorE=ValorE+pow(Form1->Buffer[ i ],2);
      ValorM=ValorM+Form1->Buffer[ i ];
    }
  }
  ValorE=sqrt(ValorE/320);//Cálculo do Valor Eficaz
  ValorM=ValorM/320;//Cálculo do Valor Médio
  Form1->Edit7->Text=FloatToStrF(ValorE,.00,4,4) + " V";
  Form1->Edit8->Text=FloatToStrF(ValorM,.00,4,4) + " V";
  Form1->Edit10->Text=FloatToStrF(vmax,.00,4,4) + " V";
  Form1->Osciloscopio->YAxis->Min->Value=(vmin-1);
  Form1->Osciloscopio->YAxis->Max->Value=(vmax+1);
  Form1->Espectro->YAxis->Max->Value=abs(vmax)+1;
  v1max=vmax;
  vmax=-200;
  vmin=+200;
  fft(samp, m); // chama função fft
  float ult=0,pen=0,atu=0;
  for(int i = 1; i < 321; i ++)//Verifica frequência 1
  {
    Buffer5[ i-1] = pow(pow(samp[ i ].real, 2) + pow(samp[ i ].imag, 2), 0.5) / 160;
    if (amax<Buffer5[ i-1 ])
    {
      amax=Buffer5[ i-1 ];
      iamax=i-1;
    }
  }
   for(int i = iamax + 2*iamax ; i < 321; i ++)//Verifica frequência 2
  {
    if (amed<Buffer5[ i-1 ])
    {
      amed=Buffer5[ i-1 ];
      iamed=i-1;
    }
  }
  for(int i = iamed + 2*iamax; i < 321; i ++)//Verifica frequência 3
  {
    if (amin<Buffer5[ i-1 ])
    {
      amin=Buffer5[ i-1 ];
      iamin=i-1;
    }
  }
  f1=iamax*constante; a1=Buffer5[iamax];//Calcula aplitude e frequência 1
  f2=iamed*constante; a2=Buffer5[iamed];//Calcula aplitude e frequência 2
  f3=iamin*constante; a3=Buffer5[iamin];//Calcula aplitude e frequência 3
  Form1->Edit9->Text=FloatToStrF(f1,.00,4,1) + " Hz";
  Form1->Edit11->Text=FloatToStrF(f2,.00,4,1) + " Hz";
  Form1->Edit12->Text=FloatToStrF(f3,.00,4,1) + " Hz";
  Form1->Amp1->Text=FloatToStrF(20*log10(a1/v1max),.00,4,1) + " dB";
  Form1->Amp2->Text=FloatToStrF(20*log10(a2/v1max),.00,4,1) + " dB";
  Form1->Amp3->Text=FloatToStrF(20*log10(a3/v1max),.00,4,1) + " dB";
  // transferência do módulo do sinal da fft
  Espectro->Channels->Channels[ 0 ]->Data->SetYData( Buffer5, 320 ); // transferência
dos dados para o gráfico
```

```
////////////////////////////////////////////////////////////////////////////////
////
  if (parar==1)
  {
    Edit1->Text = "x";
    bTransmiteMensagem(Form1->Edit1->Text.c_str());
    bEncerraSerial();
    LeituraThread->Suspend();        //Pausa a Thread
    Timer1 -> Enabled = false;
    leitura=1;
  }
  else
  {
    bTransmiteMensagem(amostras.c_str());
    Sleep(10);
    bTransmiteMensagem(amplitude.c_str());
    leitura=0;
  }
}
}
//---------------------------------------------------------------------------
// Botão Calcular!

//Informações
void __fastcall TForm1::Informaes1Click(TObject *Sender)
{
ShowMessage(AnsiString( "Analisador de Espectro via PC com aquisição
microcontrolada\n")+ ("Trabalho de conclusão de curso 2010/1\n") + ("\nALUNO: Nilson
Cezar de Oliveira\n")+ ("ORIENTADOR: Prof. MSc. Eng. Eletr. Miriam Noemi Cáceres
Villamayor" ));
}
//---------------------------------------------------------------------------
void fft(COMPLEX *x, int m)   // função fft
{
static COMPLEX *w;
static int mstore = 0;
static int n = 1;
COMPLEX u, temp, tm;
COMPLEX *xi, *xip, *xj, *wptr;
int i, j, k, l, le, windex;
double arg, w_real, w_imag, wrecur_real, wrecur_imag, wtemp_real;
if(m != mstore)
{ // libera memória alocada previamente e aloca novo m
  if(mstore != 0) free(w);
  mstore = m;
  if(m == 0) return;
  n = 1 << m; // n = 2**m = tamanho da fft
  le = n/2;
  w = (COMPLEX *) calloc(le-1, sizeof(COMPLEX)); // aloca a memória para o w
  if(!w)
  {
    exit(1);
  }
  arg = 4.0 * atan(1.0)/le;
  wrecur_real = w_real = cos(arg);
  wrecur_imag = w_imag = -sin(arg);
  xj = w;
  for (j=1; j<le; j++)
  {
```

```
    xj->real = (float)wrecur_real;
    xj->imag = (float)wrecur_imag;
    xj++;
    wtemp_real = wrecur_real * w_real - wrecur_imag * w_imag;
    wrecur_imag = wrecur_real * w_imag + wrecur_imag * w_real;
    wrecur_real = wtemp_real;
  }
}
le = n; // começa a fft (butterfly)
windex = 1;
for ( l=0; l<m; l++)
{
  le = le / 2;
  for(i=0; i<n; i = i + 2*le)
  {
    xi = x + i;
    xip = xi + le;
    temp.real = xi->real + xip->real;
    temp.imag = xi->imag + xip->imag;
    xip->real = xi->real - xip->real;
    xip->imag = xi->imag - xip->imag;
    *xi = temp;
  }
  wptr = w + windex - 1;
  for(j = 1; j<le; j++)
  {
    u = *wptr;
    for(i=j; i<n; i = i + 2*le)
    {
      xi = x + i;
      xip = xi + le;
      temp.real = xi->real + xip->real;
      temp.imag = xi->imag + xip->imag;
      tm.real = xi->real - xip->real;
      tm.imag = xi->imag - xip->imag;
      xip->real = tm.real * u.real - tm.imag * u.imag;
      xip->imag = tm.real * u.imag + tm.imag * u.real;
      *xi = temp;
    }
    wptr = wptr + windex;
  }
  windex = 2 * windex;
}
//Rearranjo dos bits
j = 0;
for (i=1; i < (n-1); i++)
{
  k = n/2;
  while(k <= j)
  {
    j = j - k;
    k = k/2;
  }
  j = j + k;
  if (i<j)
  {
    xi = x + i;
    xj = x + j;
    temp = *xj;
    *xj = *xi;
```

```
    *xi = temp;
  }
}
}
void __fastcall TForm1::GravarClick(TObject *Sender)
{

if (Gravar->Checked==true)
{
 Aguarde->Visible=true;
 VerHistorico->Enabled=false;
 Barra->SimpleText="Buscando Dados do Dia";
 Form1->Repaint();

if (FileExists(DataGravada->Text + ".Dot"))
{
ListaDados->Items->LoadFromFile(DataGravada->Text + ".Dot");
}
else
{
ListaDados->Items->SaveToFile(DataGravada->Text + ".Dot");
}
Registros->Visible=true;
Label29->Visible=true;
 Aguarde->Visible=false;
  Barra->SimpleText="Gravando";
}
else
{
 VerHistorico->Enabled=true;
Barra->SimpleText="";
Registros->Visible=false;
Label29->Visible=false;
}
}
//---------------------------------------------------------------------------

void __fastcall TForm1::VerHistoricoClick(TObject *Sender)
{
if (VerHistorico->Checked==true)
{
Gravar->Enabled=false;
Play->Enabled=true;
Aguarde->Visible=true;
Barra->SimpleText="Buscando Dados do Histórico";
Form1->Repaint();
if (FileExists(DataGravada->Text + ".Dot"))
{
ListaDados->Items->LoadFromFile(DataGravada->Text + ".Dot");
Registros->Caption=IntToStr(ListaDados->Items->Count-(ListaDados->Items->Count/321));
Registros->Visible=true;
Label29->Visible=true;
Vazio->Text="não";
}
else
{
Barra->SimpleText="Arquivo não existe";
Vazio->Text="sim";
}
Aguarde->Visible=false;
```

```
//Barra->SimpleText="";
}
else
{
Gravar->Enabled=true;
Play->Enabled=false;
Registros->Visible=false;
Label29->Visible=false;
}
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormDeactivate(TObject *Sender)
{
Edit1->Text = "x";
    bTransmiteMensagem(Form1->Edit1->Text.c_str());
    bEncerraSerial();
    LeituraThread->Suspend();        //Pausa a Thread
    Timer1 -> Enabled = false;
    leitura=1;
   bEncerraSerial();
   LeituraThread->Terminate();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  Word Year, Month, Day;
  TDateTime dtPresent = Now()+ UpDown1->Position;
  DecodeDate(dtPresent, Year, Month, Day);
  DataGravada->Text = IntToStr(Day) + IntToStr(Month) + IntToStr(Year);
  if (VerHistorico->Checked==true)
{
Gravar->Enabled=false;
Play->Enabled=true;
Aguarde->Visible=true;
Barra->SimpleText="Buscando Dados do Histórico";
Form1->Repaint();
if (FileExists(DataGravada->Text + ".Dot"))
{
ListaDados->Items->LoadFromFile(DataGravada->Text + ".Dot");
Registros->Caption=IntToStr(ListaDados->Items->Count-(ListaDados->Items->Count/321));
Registros->Visible=true;
Label29->Visible=true;
Vazio->Text="não";
}
else
{
Barra->SimpleText="Arquivo desse dia não existe";
Vazio->Text="sim";
}
Aguarde->Visible=false;
//Barra->SimpleText="";
}
else
{
Gravar->Enabled=true;
Play->Enabled=false;
Registros->Visible=false;
```

```
Label29->Visible=false;
}
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormShow(TObject *Sender)
{
Word Year, Month, Day;
  TDateTime dtPresent = Now();
  DecodeDate(dtPresent, Year, Month, Day);
  DataGravada->Text = IntToStr(Day) + IntToStr(Month) + IntToStr(Year);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
if (BarraHistorico->Position>=321)
BarraHistorico->Position=BarraHistorico->Position-321;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (BarraHistorico->Position<=BarraHistorico->Max-321)
BarraHistorico->Position=BarraHistorico->Position+321;
}
//---------------------------------------------------------------------------
```

Edit3.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <stdlib.h>
#include "math.h"
#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,j,k=0, valor;
static char buffer[TAM_MAX_BUFFER];
String c,t;
AnsiString VHexa;
BYTE aux;
FreeOnTerminate=true;
while(1)
{
  if (Form1->VerHistorico->Checked==true) Form1->leitura=1; //Verifica se histórico
habilitado
  Form1->Barra->SimpleText="";//Limpa barra de status
  if (Form1->leitura==0)//Testa se leitura esta habilitada
  {
    Form1->Barra->SimpleText="Lendo Informações do PIC";
```

```
    Form1->Repaint();
    c="";
    while (c!="blz")  // verifica se terminou a leitura do PIC
    {
      aux = bRecebeMensagem(buffer);   //Testa se mensagem "blz"
      c=((String)buffer);
      Sleep(10);
    }
    k=0;
    Form1->valor_medio=0;
    Form1->Edit1->Text = "";
    bTransmiteMensagem(Form1->Edit1->Text.c_str()); // transmite uma char
    Sleep(100);
    aux = bRecebeMensagem(buffer);
    for( j = 0; j < 962; j++ ) // Lê todo buffer da serial a cada três caracteres
    {
      valor =  StrToInt((String)buffer[j]+(String)buffer[j+1]+(String)buffer[j+2]);
      j=j+2;
      Form1->Buffer[ k ] = valor;
      if (k!=0)
        {
        if (!(valor & 0x80)) // se menor que 128 converte p/ 0/-2,5 x fator
        Form1->Buffer[ k] = (((Form1->Buffer[ k ]/128)*2.5)-2.5)*Form1->fator;
        else // se maior que 128 converte p/ 0/+2,5 x fator
        {
          Form1->Buffer[ k ] = (valor & 0x7f);
          Form1->Buffer[ k ] = (((Form1->Buffer[ k ])/255)*5)*Form1->fator;
        }
      }
      Form1->valor_medio = Form1->valor_medio + Form1->Buffer[ k ];
      k++;
    }
    Form1->valor_medio = Form1->valor_medio / 320;
    memset( buffer, 0, TAM_MAX_BUFFER);  //Limpa o buffer.
    Sleep(10);
    //Desabilita Leitura e habilita escrita
    Form1->Gravado->Text="não";
    Form1->leitura=1;
  }
}
}
//---------------------------------------------------------------------------
```

ANEXO A – CERTIFICADO DE CALIBRAÇÃO DO OSCILOSCÓPIO

LABELO/PUCRS

Página 1 de 3

Pontifícia Universidade Católica do Rio Grande do Sul
LABELO - Laboratórios Especializados em Eletroeletrônica
Calibração e Ensaios
REDE BRASILEIRA DE CALIBRAÇÃO
Laboratório de Calibração Acreditado pela CGCRE/INMETRO de acordo com a ABNT NBR ISO/IEC 17025, sob o número 0024.

Calibração NBR ISO/IEC 17025
CAL 0024

Certificado de Calibração Nº E1532/2008

Data: 10/11/2008

Cliente: Cia. de Cimentos do Brasil
Av. Getulio Vargas, 9499 - Morretes - Nova Santa Rita - RS

Características da Unidade Sob Teste:

Nome: Scopmeter
Fabricante: Fluke
Modelo: 123
Protocolo Nº: 51714
Nº de Série: DM7310265

Nome: Alicate de corrente
Fabricante: Fluke
Modelo: i1000s
Nº de Série: 7761 54309

Procedimento(s) de Calibração Utilizado(s):

- 1.06.01 - Rev. 5
- 1.02.02 - Rev. 2

Método(s) Utilizado(s):

- Comparação direta com o padrão.
- Determinação da grandeza sob calibração através da aplicação da Lei de Ampère.

Padrão(ões) Utilizado(s):

- Fluke 5520A - Certificado de Calibração nº E1353/20088 do LABELO - Válido até 10/2009
  Obs.: Padrões rastreados aos padrões primários nacionais e internacionais.

Observação:

- Os resultados da calibração estão contidos em tabelas anexas, que relacionam os valores indicados pelo instrumento sob teste, com valores obtidos através da comparação com os padrões e as incertezas estimadas da medição (IM).
- A incerteza expandida de medição relatada é declarada como a incerteza padrão de medição multiplicada pelo fator de abrangência "k", com graus de liberdade efetivos (v_{eff}) correspondentes a um nível de confiança de aproximadamente 95%. A incerteza padrão da medição foi determinada de acordo com o "Guia para Expressão da Incerteza de Medição", Terceira Edição Brasileira.

Av. Ipiranga nº 6681, Prédio 30 Bloco 3, Sala 200 – Partenon – CEP 90619-900 – Porto Alegre-RS – Brasil
Telefone: (51) 3320 3551 – Fax: (51) 3320 3901 – CP 1429 – labelo@pucrs.br – www.pucrs.br/labelo