INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA

CAMPUS JOINVILLE
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

ALLAN PEDRO DADAM

ANALISADOR DE FREQUÊNCIA E ESPECTRO SONORO PARA ALINHAMENTO DE SISTEMAS DE ÁUDIO

JOINVILLE
2019

ALLAN PEDRO DADAM

ANALISADOR DE FREQUÊNCIA E ESPECTRO SONORO PARA ALINHAMENTO DE SISTEMAS DE ÁUDIO

JOINVILLE
2019

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA

CAMPUS JOINVILLE
CURSO SUPERIOR DE MECATRÔNICA INDUSTRIAL

ALLAN PEDRO DADAM

ANALISADOR DE FREQUÊNCIA E ESPECTRO SONORO PARA ALINHAMENTO DE SISTEMAS DE ÁUDIO

Submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos de obtenção do título de Tecnólogo em Mecatrônica Industrial

Orientador: Nivaldo T. Schiefler Junior, Dr.

JOINVILLE
2019

Dadam, Allan Pedro

Analisador de Frequência e Espectro Sonoro para Alinhamento de Sistemas de Áudio / Allan Pedro Dadam. - Joinville, SC, 2019.
102 p.

Trabalho de Conclusão de Curso - Instituto Federal de Santa Catarina. Campus Joinville. Graduação. Curso Superior de Tecnologia em Mecatrônica Industrial, Joinville, 2019.
Orientador: Prof. Nivaldo T. Schiefler Junior, Dr.

1. Analisador de Frequência. 2. Transformada Rápida de Fourier. 3. Microcontroladores. I. Schiefler, Nivaldo T. Junior. II. Instituto Federal de Santa Catarina. III. Analisador de Frequência e Espectro Sonoro para Alinhamento de Sistemas de Áudio

ANALISADOR DE FREQUÊNCIA E ESPECTRO SONORO PARA ALINHAMENTO DE SISTEMAS DE ÁUDIO

Este trabalho foi julgado adequado para obtenção do título de Tecnólogo em Mecatrônica Industrial e aprovado na sua forma final pela banca examinadora do Curso Mecatrônica Industrial do Instituto Federal de Educação, ciência e Tecnologia de Santa Catarina.

Aprovado em: _____/_____/_____

Conceito:_____

Banca Examinadora:

_____

Prof. Nivaldo T. Schiefler Junior, Doutor
Orientador

_____

Prof. José Flávio Dums, Doutor
Avaliador

_____

Prof. Rubens Hesse, Doutor
Avaliador

DEDICATÓRIA

“Agradeço a minha família e namorada por todo apoio e suporte no desenvolvimento deste trabalho.”

AGRADECIMENTOS

Agradeço ao próprio Instituto Federal de Santa Catarina, por fornecer toda a infraestrutura necessária ao curso, além dos recursos e investimento necessários para o desenvolvimento deste Trabalho de Conclusão

Ao meu professor e orientador Nivaldo T. Schiefler Junior pelo tempo investido e orientação necessária para a realização deste projeto.

Aos meus pais, Meri Aparecida Dadam e Narcizo Henrique Dadam, por todo apoio e paciência durante a graduação.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema microcontrolado para analisar as frequências e espectro sonoro para alinhamento de sistemas de som. A ideia inicial do projeto se deu a partir da necessidade profissional em alinhar sistemas de som para eventos e concertos musicais. Foram utilizados alguns os conhecimentos obtidos nas aulas tais como o Microcontrolador e Eletrônica Analógica. No desenvolvimento do projeto foi utilizado o PIC 18F4550 da microchip por possuir conversor ADC e velocidade de clock compatíveis com as rotinas FFT em tempo real. Todas as rotinas desenvolvidas neste trabalho foram baseadas em sua maior parte, em código-fonte aberto disponível na rede. Foi projetada e desenvolvida uma placa de circuito impresso para acoplamento do circuito. Testes de detecção de frequências específicas e do espectro geral foram realizados a fim de avaliar os resultados obtidos.

Palavras-Chave: Analisador; Frequência; FFT; Espectro.

ABSTRACT

This paper presents the development of a microcontrolled system for analyzing frequencies and spectrum for alignment of sound systems. The initial idea of project was based on the professional need to align sound systems for music events and concerts. Were some of the knowledge obtained in the classes such as the Microcontroller and Analog Electronics. In the development of Microchip PIC 18F4550 was used because of its ADC converter and clock speed compatible with routines FFT in real time. All routines developed in this mostly based on source code open available on the network. It was designed and developed a printed circuit board for circuit coupling. Tests frequency and general spectrum detection were performed in order to evaluate the results obtained.

Keywords: Analyzer; Frequency; FFT; Spectrum.

LISTA DE FIGURAS

LISTA DE SIMBOLOS

Hz	hertz
kHz	quilo hertz
dB	decibel
Ω	ohm
kΩ	quilo ohm
nF	nano faraday
V	volt
mV	mili volt
μs	micro segundos
λ	comprimento de onda
f	frequência
c	velocidade propagação do som
I	intensidade

LISTA DE ABREVIATURAS E SIGLAS

3D	Three-dimensional space	Três dimensões
AC	Alternating Current	Corrente Alternada
ADC	Analogic Digital converter	Conversor Analógico-Digital
ALU	Arithmetic Logic Unit	Unidade lógica e aritmética
CI	Integrate Circuit	Circuito Integrado
DC	Direct Current	Corrente Contínua
DIP	Dual In-line Package	Tipo de pinagem de CI
DFT	Discrete Fourier Transform	Transformada Discreta de Fourier
FFT	Fast Fourier Transform	Transformada rápida de Fourier
FPS	Frames per second	Quadros por segundo
JFET	Junction Field Efect Transistor	Transistor de junção por efeito de campo
LCD	Liquid crystal display	Display de cristal líquido
PCB	Printed circuit board	Placa de circuito impresso
PIC	Peripheral Interface Controller	Controlador de Interface Periférica
USB	Universal Serial Bus	Barramento Serial Universal
XLR	Cannon X Leatch Ruber	Conector Cannon

SUMÁRIO

1. INTRODUÇÃO

Dentro do campo de estudo do áudio uma das maiores dificuldades encontrada por engenheiros e técnicos é o alinhamento do sistema de som. Vários fatores influenciam o profissional a chegar a um resultado satisfatório, como, posicionamento das caixas, dimensão do espetáculo, qualidade dos equipamentos, reverberação da sala ou do espaço.

De acordo com Page (2015), um sistema linear com maior equilíbrio entre as frequências responde bem a maioria dos arranjamentos musicais, para chegar a essa resultante, além da disposição física dos falantes e caixas, são utilizados analisadores de frequência e equalizadores de banda.

O analisador de frequência processa o sinal captado pela pressão sonora exercida no microfone de referência, neste caso com resposta plana e omnidirecional, e mostra através de um gráfico de banda (Page, 2015).

O processamento do sinal é feito através da função matemática conhecida como Transformada Rápida de Fourier. Basicamente, essa função decompõe um sinal em ondas senoidais de diferentes amplitudes e frequências. Uma vez desconstruído o sinal, pode-se ver e analisar as diferentes frequências presentes no sinal original. Em ondas de áudio, como graves e agudos, pode ser alterado tons ou frequências, atenuando a intensidade de determinadas faixas do espectro para equilibrar o sistema de forma linear (Oppennheim, 2010).

1.1 Organização

Este trabalho está divido em quatro capítulos, além da introdução e conclusão. No Capítulo 2 está a revisão teórica, na

qual se discorre sobre os assuntos que serviram como base para o desenvolvimento do projeto, trazendo para literatura temas que serão abordados nas próximas etapas.

O Capítulo 3 apresenta o funcionamento, as vantagens e as principais utilizações dos componentes utilizados no desenvolvimento do projeto.

No Capítulo 4 mostrou-se a etapa de montagem, onde ocorreu a transformação daquilo que foi pensado e projetado em algo concreto. Esta transformação se deu através da confecção do circuito impresso, solda dos componentes e montagem do projeto. Também é abordada a parte de programação do microprocessador e utilização dos *softwares* MPLAB X IDE v5.15 e ISIS Professional

No Capítulo 5 são apresentados os testes de funcionamento do analisador e os resultados obtidos. No Capitulo 6 a conclusão juntamente com a sugestão de trabalhos futuros.

1.2 Motivação

A motivação para desenvolver esse trabalho surgiu do interesse profissional em compreender como funcionam equipamentos voltados para o áudio disponíveis no mercado. Parte de uma vivencia onde o processo de alinhamento do sistema se torna necessário para apresentar um melhor rendimento e resultado, tanto para o contratante quanto para o público em geral

1.3 Objetivos do trabalho

O trabalho tem por objetivo o desenvolvimento de um protótipo analisador que seja capaz de captar o som emitido por um sistema de áudio, identificar as frequências e mostrar em um

display o espectro sonoro para avaliação das respostas obtidas. O projeto também tem como objetivo o estudo de microcontroladores PIC a nível avançado utilizando técnicas de processamento de sinais.

2. FUNDAMENTAÇÃO TEÓRICA

Para desenvolvimento do projeto, foram utilizados como base, vários assuntos pertinentes ao tema para o entendimento e análise das respostas esperadas.

2.1 Propriedades físicas do som

O som é uma sensação auditiva captada pelo ouvido através da movimentação organizada das moléculas do ar. Basicamente, todo som se caracteriza por três variáveis físicas: frequência, intensidade e timbre. De acordo com Lima (2011) as propriedades físicas do som podem ser exemplificadas em três partes.

2.1.1 Frequência

Frequência (f) em *hertz* é o número de oscilações por segundo. Para uma onda sonora em propagação, é o número de ondas que passam por um determinado referencial em um intervalo de tempo. Chamando de λ o comprimento de onda e c a velocidade de propagação do som, pode-se escrever a equação 1, que fica exemplifica na Figura 1.

$$\lambda = \frac{c}{f} \quad (1)$$

Onde: $c = 340\, m/s$
f em *Hertz*

Figura 1: Representação do comprimento de onda - Fonte: o autor

O nosso ouvido é capaz de captar sons de 20 a 20.000 Hz. Os sons com menos de 20 Hz são chamados de infra-sons e os sons com mais de 20.000 Hz são chamados de ultra-sons. Na Figura 2 observa-se o que se entendo como faixa audível de frequências ou banda audível do ouvido humano.

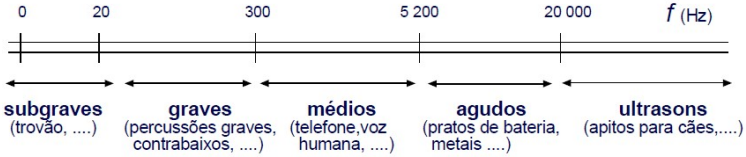

Figura 2: Faixa audível para o ouvido humano – Fonte: o autor.

2.1.2 Intensidade

A intensidade do som é a quantidade de energia contida no movimento vibratório. Essa intensidade se traduz com uma maior ou menor amplitude na vibração ou na onda sonora. É o que permite ao ouvido diferenciar os sons fracos dos sons fortes. Considerando Io como a menor intensidade de som audível ($\mathrm{I_o}= 10^{-12}\ \mathrm{W/m^2}$) e I a intensidade do som que se quer determinar, define-se:

$$\text{Intensidade (dB)} = \log\frac{I}{I_o} \quad (2)$$

Onde: $I_o = 10^{-12}\ W/m^2$
$W = watts$

2.1.3 Timbre

Timbre é a características especifica que torna possível identificar a fonte sonora, e com isso, distinguir entre duas ou mais fontes que estejam emitindo uma mesma nota. Por exemplo, na Figura 3, as mesmas notas tocadas por um piano e por um violino podem possuir a mesma frequência, mas com características sonoras distintas. A caracterização do instrumento é feita pela sua forma de onda e seu envelope sonoro.

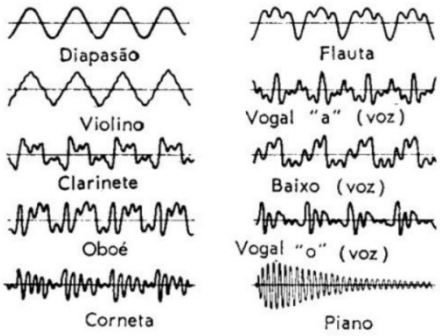

Figura 3: diferentes formas de onda - fonte:
https://raquellima16.wordpress.com/2011/01/27/caracteristicas-do-som-frequencia-amplitude-e-timbre/

2.2 Microcontroladores

O microcontrolador é um circuito integrado responsável por processar dados e executar instruções. É um dispositivo que contém em um único chip, processador, memória e periféricos de entrada/saída. São muito úteis em projetos embarcados, onde existe uma especificidade no processamento, ou seja, as tarefas são dedicadas exclusivamente ao dispositivo ou sistema que ele controla. Geralmente utilizados em projetos com espaço reduzido.

A programação dos periféricos dos Microcontroladores deve ser executada conforme o entendimento do funcionamento do periférico que se deseja utilizar e na configuração correta dos registradores envolvidos. Tanto a forma de funcionamento de um periférico, quanto às informações sobre os registradores envolvidos na sua programação são detalhadamente encontrados no *datasheet* e *application notes* de cada tipo.

2.3 Programação em C

Linguagem C nada mais é que um conjunto de termos e abreviações que substitui os códigos de máquina ou códigos binários (zeros e uns). Imagine que no início a programação dos dispositivos era feita sem o auxílio de comandos, instruções ou funções já predefinidas. O que tornava o trabalho muito extenso, pois o programador teria que desenvolver rotinas e operações, criando um fluxo complexo difícil de ser seguido.

Conforme Fábio Pereira (2008, p. 18) “A maioria dos microcontroladores disponíveis no mercado conta com compiladores de linguagem C para desenvolvimento de *software*“. Desta forma, facilita o desenvolvimento de rotinas complexas e utilização de bibliotecas pré-existentes.

“C utiliza a filosofia de programação estruturada, ou seja, os programas são divididos em módulos ou estruturas (que em C são chamadas funções) independentes entre si e com o objetivo de realizar determinada tarefa” Fábio Pereira (2008).

2.4 Transformada rápida de Fourier

A transformada rápida de Fourier ou *fast Fourier transform* (FFT) é utilizada para decompor frequências do som, pois o microfone capta apenas a pressão do resultante da somatória de todas as frequências geradas, ou seja, ele só vê a soma final.

A FFT é uma otimização da transformada discreta de Fourier (DFT). “O numero de cálculos necessários para executar a DFT foi drasticamente reduzido por um algoritmo desenvolvido por Cooley e Tukey em 1965” Lathi (2007). A FFT é utilizada no processamento digital de sinais, transformado do tempo continuo para o tempo discreto, facilitando assim o processamento do sinal em ambiente digital.

“Um sinal analógico é caracterizado pelo fato de sua amplitude poder assumir qualquer valor em uma faixa continua. Logo, a amplitude de um sinal analógico pode assumir infinitos valores. Por outro lado, a amplitude de um sinal digital só pode assumir um número finito de valores. Um sinal analógico pode ser convertido em um sinal digital através da amostragem e quantização.” Lathi (2007).

2.5 Open source

Para implementação da função de FFT na programação do microcontrolador foi utilizado código-fonte aberto existente na rede e portais. O modelo de código-fonte aberto, ou *open source*, originalmente aplicado a *softwares*, mas posteriormente ampliado

de forma a cobrir diversas outras áreas de conhecimento, como hardware, desenho (como projetos 3D editáveis de uso livre), alimentos e bebidas e até mesmo medicamentos (Munos, B. 2006).

Além do uso livre, o modelo permite a livre colaboração no desenvolvimento, permitindo a entrada de um maior número de colaboradores em projetos com grande demanda e trazendo vantagens para todos os usuários do projeto, em um modelo privado-coletivo (Von Hippel, E.a et al. 2003), que adicionalmente traz aprendizado a todos os envolvidos (Lakhani, K.R. et al, 2003).

3. COMPONENTES UTILIZADOS

Após análise e pesquisa teórica sobre tema abordado no trabalho, segue a etapa de levantamento do material necessário para realização do projeto.

3.1 Microfone de Eletreto

Um microfone muito comum e de fácil aquisição, com uma resposta de frequência compatível com projeto. O microfone de eletreto funciona através da variação da capacitância de um capacitor de placas paralelas. Quando a pressão sonora atinge a placa móvel a capacitância varia dependendo da intensidade da onda sonora. A Figura 4, mostra a parte externa do encapsulamento e as conexões de alimentação. É comumente utilizado em aparelhos eletrônicos em geral. Tem uma resposta típica de frequência entre 50 Hz a 10 kHz, sensibilidade típica de -65 dB e funciona com tensão de alimentação de no máximo 10 V.

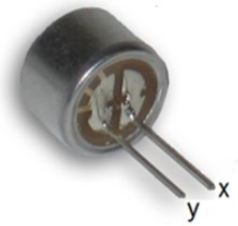

Figura 4: Microfone de Eletreto - Fonte: http://www.openmusiclabs.com/learning/sensors/electret-microphones/ acesso em 14/06/2019.

3.2 Circuito Integrado TL084

O TL084 é um circuito integrado composto por quatro amplificadores operacionais. Como observado na Figura 5,

existem duas entradas para cada saída, uma designada por entrada inversora (-) e outra identificada por entrada não-inversora (+). A tensão de saída é a diferença entre as entradas (+) e (-), multiplicado pelo ganho em malha aberta. Como o ganho é muito alto, utiliza-se uma realimentação negativa para ajustar a tensão de saída. A relação de resistores na alimentação negativa será exemplificada no capítulo de desenvolvimento, assim como a expressão utilizada para calcular os valores dos resistores. O circuito integrado possui alimentação simétrica, apresentado na Figura 5, o que implicou ao projeto a necessidade de desenvolver uma fonte simétrica para alimentação do CI.

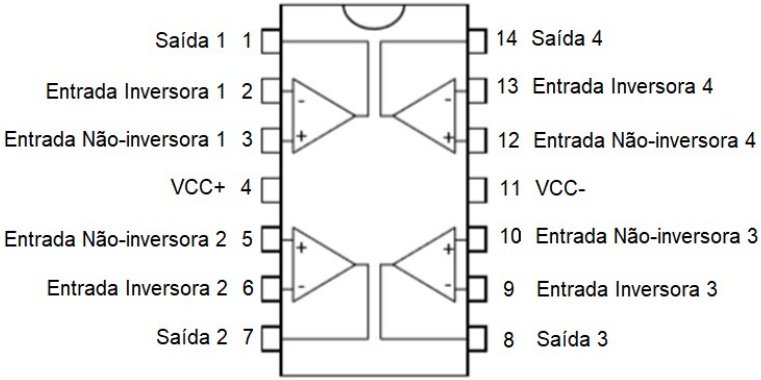

Figura 5: TL084 Pinos de conexão (*Vista Superior*) - Fonte: http://www.ti.com/lit/ds/symlink/tl084.pdf

3.3 Placa de Desenvolvimento

Para desenvolvimento de protótipo foi escolhido a placa de desenvolvimento AFSmart, Figura 6, o modulo já vem equipado com o Microcontrolador PIC 18F4550, alimentação via USB com regulador de tensão, oscilador de cristal de 20 MHz, *Bootloader* interno para gravação do PIC, compatibilidade com Pickit3 da Microchip, além de entradas e saídas disponíveis para futuras utilizações em projetos em *Protoboard*.

Figura 6: AFSmart - Fonte: https://www.afeletronica.com.br

3.4 PIC 18F4550

A escolha do PIC 18F4550 foi uma das etapas mais importantes do projeto. É um PIC de 8 *bits* capaz efetuar as rotinas de FFT em 16 *bits* já que possui uma função de *hardware* 8 x 8 no processador. Possui conversores ADC (Analógico-Digital) internos, o que facilitou na comunicação direta com o amplificador operacional TL084 e funciona com *clock* de frequência de 48 MHz.

40-Pin PDIP

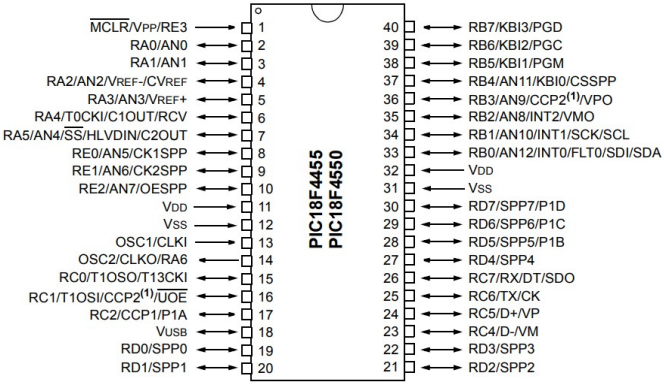

Figura 7: Pinagem PIC 18F4550 - Fonte:
http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf

3.5 Display LCD Gráfico 128×64

O *display* gráfico escolhido para o projeto, Figura 8, foi o LCD com resolução 128x64 pixels com controladora KS0108, o que permite comunicação com o PIC 18F4550.

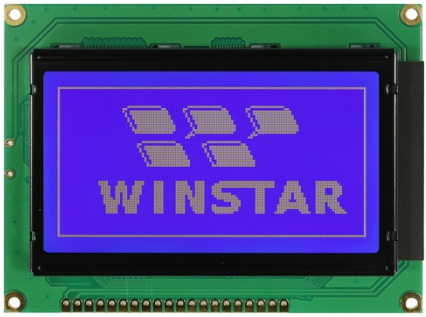

Figura 8: Display LCD 128x64 - Fonte:
https://www.winstar.com.tw/products/graphic-lcd-display-module/monochrome-graphic.html

4. METODOLOGIA

Neste capítulo, disserta-se sobre as etapas de desenvolvimento do projeto, com detalhamento suficiente de forma a torná-lo compreensível. O projeto pode ser divido em três partes, o desenvolvimento do circuito analógico, programação do Microcontrolador PIC e montagem da placa de circuito impresso.

O funcionamento do circuito consiste em captar o som, através de um microfone, transformar o sinal de áudio em sinal analógico, amplificar o sinal adquirido, decompor as frequências em faixas pré-determinadas e mostrar os valores no *display* em modo gráfico. Desta forma o detalhamento de cada etapa será descrito a seguir.

4.1 Circuito analógico

O circuito analógico consiste no circuito de captação do áudio, amplificação e filtragem do sinal, conforme visto no diagrama de blocos apresentado na Figura 9. Assim, o sinal é mandado para a entrada do microcontrolador PIC, onde, no primeiro momento é convertido o sinal analógico em sinal digital através das portas de entrada do microcontrolador. A programação de conversão será descrita conforme o andamento do trabalho.

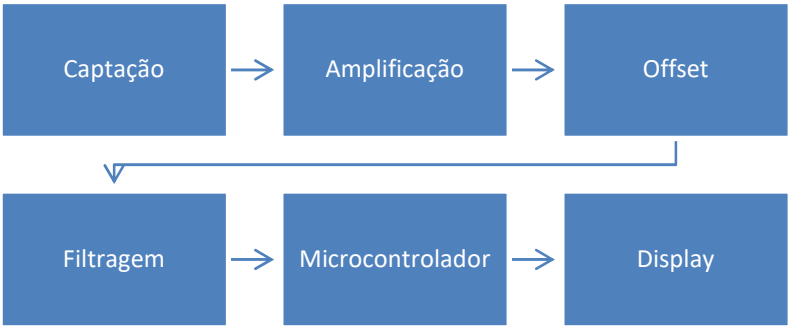

Figura 9: Fluxograma de Processos – Fonte: o autor.

Como a alimentação do microcontrolador é de +5 V DC e o amplificador operacional TL084 necessita de uma alimentação de simétrica +9 V e -9 V DC. Foi projetado, em paralelo ao circuito analógico, um circuito de alimentação com a função de regular a tensão para a alimentação da placa.

4.1.1 Captação do áudio

Para a captação do som foi utilizado um microfone de eletreto apresentado pelo circuito resultando na Figura 10.

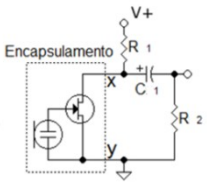

Figura 10: Encapsulamento Microfone Eletreto – Fonte: http://www.openmusiclabs.com/learning/sensors/electret-microphones/

Conforme observado na Figura 10, o correto funcionamento do microfone se dá através da sua polarização e consequentemente da polarização do Transistor (JFET). O capacitor de placas paralelas dentro do encapsulamento está ligado entre o *gate* e o *source* do transistor. Uma vez que a condução do JFET é controlada pela tensão entre *gate* e o *source*, variações na capacitância fazem com que a tensão sobre o capacitor varie, e assim, a condução do transistor também varia. A polarização é feita através do resistor R1. O capacitor C1 e o resistor R2 formam um filtro passa-alta com o intuito de filtrar o nível DC e as frequências abaixo de 300 Hz da saída do microfone.

Para calcular os valores do capacitor e resistor do filtro passa-alta utilizou-se a equação 3, com corte de frequência em 300 Hz, os valores comerciais encontrados para R2 e C1 ficaram em 5 kΩ e 100 nF, respectivamente.

$$f_c = \frac{1}{2\pi R} \quad (3)$$

4.1.2 Amplificação do Sinal

Para realizar um cálculo de FFT em um sinal de áudio, é necessário preparar o áudio para que o PIC18F4550 possa converter o sinal. O microcontrolador fornece vários conversores analógicos para digitais (ADCs) que podem ser usados para medir uma tensão de 0 V a 5 V com precisão de 10 *bits* (0-1023). Um sinal típico de saída de áudio é uma onda analógica com intensidade pico-a-pico de 2 V centrada em torno de 0 V. O sinal gerado pelo microfone de eletreto é muito baixo, o que dificultaria a implementação da FFT. Para resolver essa questão é

necessário amplificar o sinal de saída do microfone, utilizando um bloco de amplificação do sinal.

A fim de controlar o ganho de entrada e ter uma resposta mais linear da saída de tensão, utiliza-se um modo de operação em malha fechada com realimentação negativa ilustrado pela Figura 11. O controle de ganho de acordo com o *datasheet* do TL084 se dá na relação de resistores através da equação 4, onde R3 é o resistor de entrada R_i e R4 o resistor de realimentação R_f, para controle de ganho utiliza-se resistores com valores em $\mathrm{k\Omega}$. Conforme teste em bancada, evitando que a amplitude de tensão saturasse na entrada do PIC, o ajuste de ganho ficou em torno de 37 vezes. Assim os valores comerciais encontrados para os resistores R3 e R4 ficaram em 2,2 $\mathrm{k\Omega}$ e 82 $\mathrm{k\Omega}$, respectivamente.

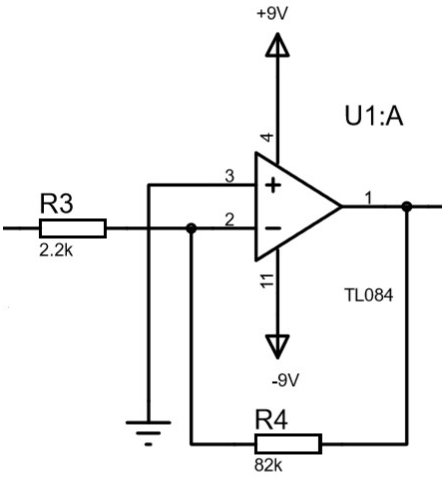

Figura 11: Amplificador Operacional – Fonte: o autor.

$$V_o = -\frac{R_f}{R_i}.V_i \quad (4)$$

Como a realimentação é negativa pela entrada inversora, a saída tem uma defasagem de 180° em relação ao sinal de origem. Para corrigir a fase é feito um cascateamento da saída do primeiro amplificador operacional para a entrada inversora do segundo amplificador operacional, Figura 12. Conseguindo assim uma tensão de saída em fase com a tensão de entrada. O ajuste de ganho e a realimentação do segundo amplificador operacional são semelhantes ao primeiro, porém, é adicionado um resistor variável RV1 no valor de 10 $\mathrm{k\Omega}$ em série com um resistor R6 de 10 $\mathrm{k\Omega}$. Variando o valor de resistência de RV1, controle-se o ganho conforme equação 4.

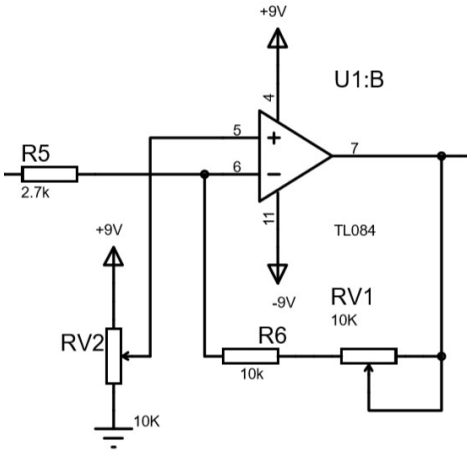

Figura 12: Amplificador Operacional e Offset – Fonte: o autor.

Como comentado anteriormente, o conversor ADC do PIC18F45550 trabalha com tensão de entrada entre 0 V e +5 V.

O TL084 funciona com alimentação simétrica de -9 V e +9 V. Isso faz com que a tensão de saída varie entre esses dois valores, centrada em um referencial comum. Para ajustarmos a saída de acordo com a faixa de entrada do conversor é necessário fazer um *Offset* da saída de tensão. Para tal utiliza-se, conforme Figura 12, um potenciômetro de resistência variável RV2 na entrada não-inversora do segundo amplificador operacional. Com o potenciômetro ligado no comum e na alimentação +9 V, é possível controlar a tensão na entrada não-inversora, movendo a referencia para um comum imaginário em torno de +2,5 V.

4.1.3 Filtro RC passa-baixa

O intuito do projeto é trabalhar na faixa de frequências do campo auditivo do ser humano. Sistemas de som endereçados ao publico também compreendem esta faixa, sendo assim quaisquer frequências abaixo de 20 Hz ou acima dos 20 kHz são irrelevantes. Na Figura 13, apresenta-se o filtro passa-baixa do projeto.

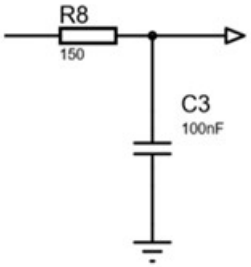

Figura 13: Filtro passa-baixa – Fonte: o autor

Para a filtragem das frequências altas é utilizado um filtro passa-baixa, formado pelo resistor R8 e o capacitor C3. O calculo dos valores comercias é feito através da expressão (3), chegando aos valores de 150 Ω e 100 nF para R8 e C3, respectivamente. O capacitor eletrolítico C2 tem a finalidade de filtrar os níveis de tensão DC provenientes do circuito integrado.

4.1.4 Circuito regulador de tensão e fonte simétrica

Como mencionado anteriormente, para suprir as necessidades de alimentação dos componentes é necessário projetar uma fonte regulada com tensões de saída conforme Figura 14. A fim de manter o equilíbrio de consumo nas cargas foi utilizado um transformador de núcleo de ferro com tomada central possuindo entrada de 220 V AC no primário e saída de 9 ~ 0 ~ 9 V AC no secundário. Em circuitos eletrônicos utiliza-se corrente contínua, como o transformador nos fornece uma tensão alternada proveniente da rede elétrica, é preciso converter essa tensão. Para tal finalidade utiliza-se uma ponte retificadora composta por quatro diodos comuns conforme Figura 14. Os reguladores de tensão LM7805 e LM7809 determinam as tensões positivas +5 V DC e +9 V DC, respectivamente. O LM7909 fica responsável por determinar a tensão negativa de -9 V DC.

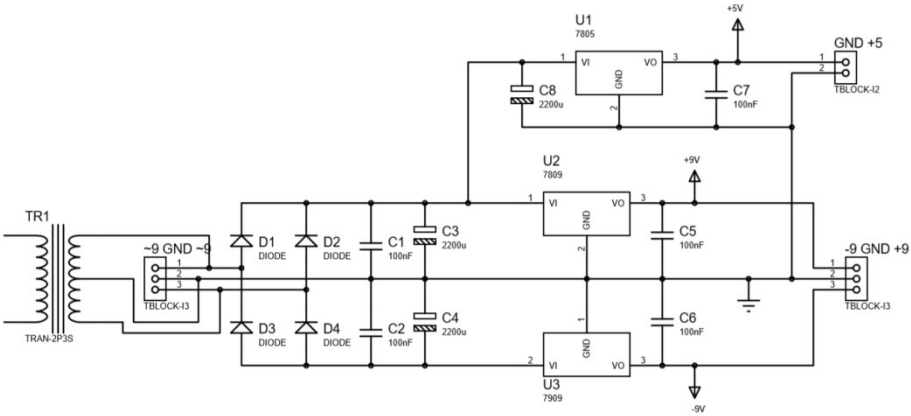

Figura 14: Esquemático regulador de tensão e fonte simétrica – Fonte: o autor

4.2 Programação do Microcontrolador

Com a devida amplificação e filtragem do sinal, a próxima etapa do projeto é a implementação e programação do microcontrolador. Para o desenvolvimento das rotinas necessárias para todo o processamento, foi utilizado o *software* MPLAB X IDE v.5.15 e o compilador de linguagem C da HiTech C18.

As bibliotecas e rotinas utilizadas no projeto seguem o modelo de código-fonte aberto, ou *open source*. Foram originalmente desenvolvidas por Simon Inns e devidamente adaptadas ao projeto aqui proposto. Os anexos 1, 2, 3, 4, 5 e 6 trazem todas as linhas de programação na íntegra.

Os tópicos seguintes são baseados no código-fonte original e explicam de forma simples e compreensível cada etapa do *firmware*. A Figura 15, exemplifica as etapas de programação.

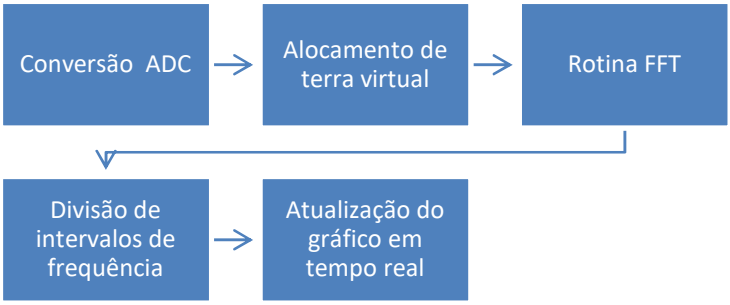

Figura 15: Etapas de programação – Fonte: o autor.

4.2.1 Conversor ADC

O sinal captado pelo microfone é de característica analógica e se altera conforme a pressão exercida no encapsulamento, variando capacitância do capacitor interno e consequentemente variando a tensão de saída. É necessário que esse sinal analógico seja convertido para um sinal digital, pois o microcontrolador trabalha com informações codificadas em código binário. A função que habilita e seta as portas de conversão ADC é:

```
ADCON0 = 0b00000001;          // Start the ADC conversion on AN0
```

De acordo com Simons (2011), a rotina de amostragem ADC mostra o nível de tensão no RA0/AN0 a cada 50 µs. Isso resulta em uma taxa de amostragem de 20 kHz. É importante para a FFT que as amostras sejam colocadas de forma uniforme e precisa.

Para garantir isso, há um pequeno atraso no *loop* de amostragem. O ADC é amostrado na resolução total de 10 *bits* e, em seguida, deslocado para baixo em 512 para simular o terra virtual do sinal de entrada de volta em zero. Isso significa que as amostras resultantes estão no intervalo de -512 a +512 exatamente como o equacionamento teórico da FFT requer. O roteamento de amostragem ADC leva um pouco mais de 64 × 50 us = 32 ms (3200 us) em tempo de execução para cada *loop*.

4.2.2 FFT de 16 *bits*

A rotina da FFT foi baseada de um exemplo encontrado na *web* (referências ao código original podem ser encontradas no

código-fonte). O código foi reduzido aos comandos mínimos necessários e transferido para o PIC. Como o microcontrolador PIC18F4550 possui uma função de multiplicação por *hardware* 8 × 8 na ALU do processador, os cálculos são otimizados para permitir que o compilador use corretamente os recursos do chip. O fato de o microcontrolador escolhido ter um multiplicador de *hardware* 8 × 8 é realmente a chave de como um *chip* de baixa potência pode executar FFT em tempo real. A vantagem da velocidade do ciclo, mesmo com cálculos de 16 *bits*, é enorme (Simon, 2011).

A saída da FFT é 32 números complexos que consistem em uma parte real e imaginária representada por dois *arrays*. Para mostrar o resultado de maneira significativa, é necessário calcular o valor absoluto do número complexo que é feito usando um cálculo de Pitágoras para calcular a distância do número complexo a partir da origem de 0. Isso envolve a realização de operações de raiz quadrada de modo que o software implemente um equivalente da função SQRT( ) muito rápido para os valores inteiro abstraídos da conversão de 123bits, já que qualquer operação de ponto flutuante seria muito lenta. A rotina FFT e os cálculos de valor absolutos combinados levam aproximadamente 70 ms (7000 us) para cada *loop* (Simon, 2011).

A velocidade aproximada da exibição da análise de espectro é de um quadro por 150 ms, resultando em uma taxa de quadros por segundo em geral de cerca de 6,5 quadros por segundo (ou cerca de 10 FPS sem o LCD). Isso pode ser facilmente aprimorado reduzindo o número necessário de intervalos de frequência (o que reduziria os tempos de execução de

amostragem e FFT) ou usando um dispositivo de exibição com um tempo de atualização mais rápido (Simon, 2011).

A frequência de Nyquist da FFT (a maior frequência que pode detectar) é de 10 kHz. Os 31 intervalos de frequência são divididos uniformemente ao longo do intervalo, no entanto, devido à forma como a rotina FFT funciona, não é possível usar um intervalo mais baixo. A Tabela 1 apresenta os intervalos de frequência por barra que são mostradas no *display* gráfico.

Tabela 1 - Faixa de Frequência por barra

Faixa	Mínimo	Máximo
1	312,5	625
2	625	937,5
3	937,5	1250
4	1250	1562.5
5	1562,5	1875
6	1875	2187,5
7	2187,5	2500
8	2500	2812,5
9	2812,5	3125
10	3125	3437,5
11	3437,5	3750
12	3750	4062,5
13	4062,5	4375
14	4375	4687,5
15	4687,5	5000
16	5000	5312,5
17	5312,5	5625
18	5625	5937,5
19	5937,5	6250
20	6250	6562,5
21	6562,5	6875
22	6875	7187,5
23	7187,5	7500
24	7500	7812,5
25	7812,5	8125
26	8125	8437,5
27	8437,5	8750
28	8750	9062,5
29	9062,5	9375
30	9375	9687,5
31	9687,5	10000

Fonte: o autor.

4.2.3 Display Gráfico

O *display* deve ser atualizado o mais rápido possível e para fazer isso é utilizado um algoritmo de desenho de gráfico de barras muito simples que requer o mínimo número possível de comandos para a exibição.

A rotina aproveita a maneira como o monitor LCD é endereçado para fazer a atualização o mais rápido possível. Foi adicionado dois *switch* no projeto para permitir ao usuário alternar entre ampliação x1 e x8 na saída (já que para música as médias de frequência são bem baixas) e também uma saída linear ou uma saída logarítmica (baseada em dB). Essas são maneiras diferentes de exibir a saída, dependendo se você deseja representações precisas de nível de frequência ou mais resultados baseados em efeitos visuais (Simon, 2011).

4.2.4 Simulação ISIS Proteus 7.6

Depois de feita a programação do microcontrolador é possível simular o circuito no *software* ISIS Proteus. A Figura 16, mostra uma visão geral da simulação em andamento.

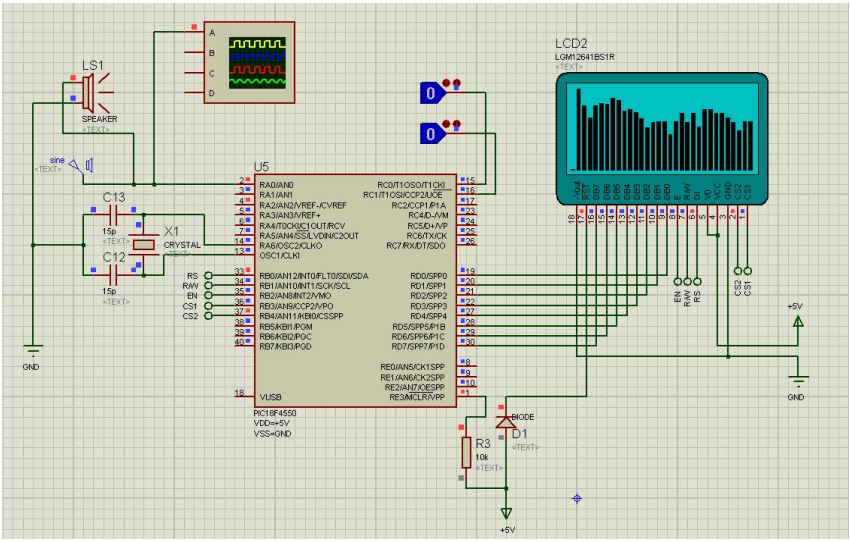

Figura 16: Esquemático desenvolvido no Simulador – Fonte: o autor.

Através de um gerador de áudio proveniente do próprio *software,* pode-se simular o sinal de entrada de um microfone utilizando um arquivo de áudio com som de ruído branco (*white noise*), por exemplo. A Figura 17, mostra as opções que determinam a amplitude e o *offset* do sinal. Através da ferramenta de osciloscópio digital observa-se na Figura 18, as características desse sinal de entrada, muito semelhante ao que se encontra em testes reais.

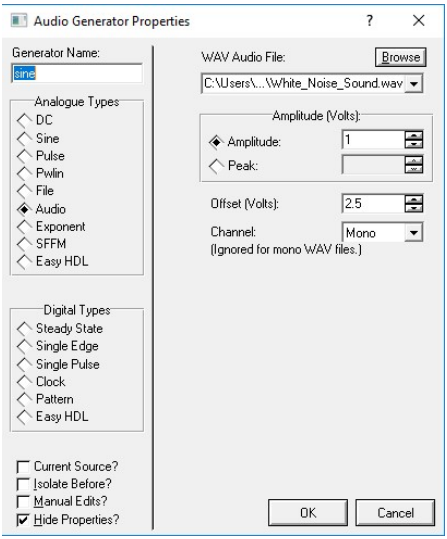

Figura 17: Tela salva do Proteus - Gerador de sinal – Amplitude e *Offset* – Fonte: o autor.

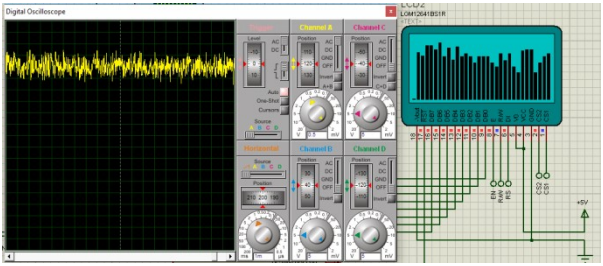

Figura 18: Características do Sinal de entrada – Fonte: o autor.

4.3 Montagem da placa de circuito impresso

O circuito é divido em três módulos com o intuito de aperfeiçoar o processo de montagem e diminuir consideravelmente as possibilidades de erros. A primeira etapa sendo a fonte de alimentação, seguido do circuito de captação, amplificação e adequação do sinal de áudio e terminando com o circuito responsável por fazer as ligações do microcontrolador e *display* gráfico, assim como suas alimentações.

Assim como na etapa de desenvolvimento é utilizado o *software* ISIS Proteus para desenhar o *layout* das placas de circuito impresso através da extensão ARES que faz parte do pacote do *software*.

4.3.1 Esquemáticos ISIS Proteus 7.6

Através dos circuitos projetados e dimensionados anteriormente, obtém-se os esquemáticos necessários para montagem do desenho de *layout* dos PCB. A Figura 14, no tópico 4.1.4 já mostra em detalhes o esquemático da fonte de alimentação. A seguir os esquemáticos ainda não apresentados, que são responsáveis pela captação e amplificação do sinal de áudio e o seguinte pela ligação do microcontrolador e o *display* gráfico. Observe que há conectores de entrada e saída, tanto de alimentação quanto de sinal para a comunicação entre os módulos. A Figura 19, mostra o circuito responsável pela captação e amplificação do sinal de áudio.

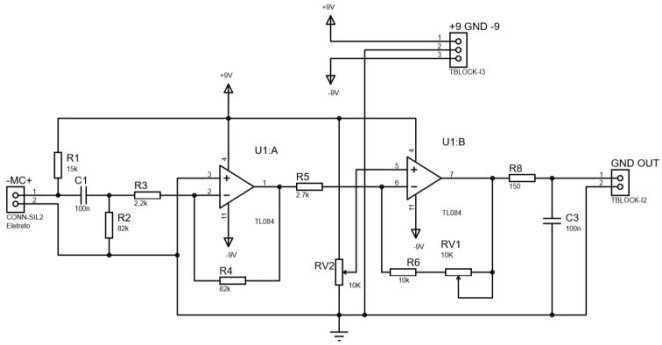

Figura 19: Esquemático Captação e amplificação do sinal de áudio – Fonte: o autor.

Na Figura 20, além das conexões de comunicação e alimentação do microcontrolador e do *display* observa-se também a inclusão de um conector de seis pinos, que facilitará em futuras gravações e atualizações no código-fonte do microcontrolador. Assim como na placa de desenvolvimento utilizada anteriormente é necessário um cristal oscilador externo de 20 MHz para suprir as exigências de processamento do projeto. De acordo com o *datasheet* do PIC 18F45550, para obter um bom funcionamento os cristais precisam de um par de capacitores ligados a eles. Os valores de capacitores sugeridos para utilização com cristal de 20 MHz são de 15 pF.

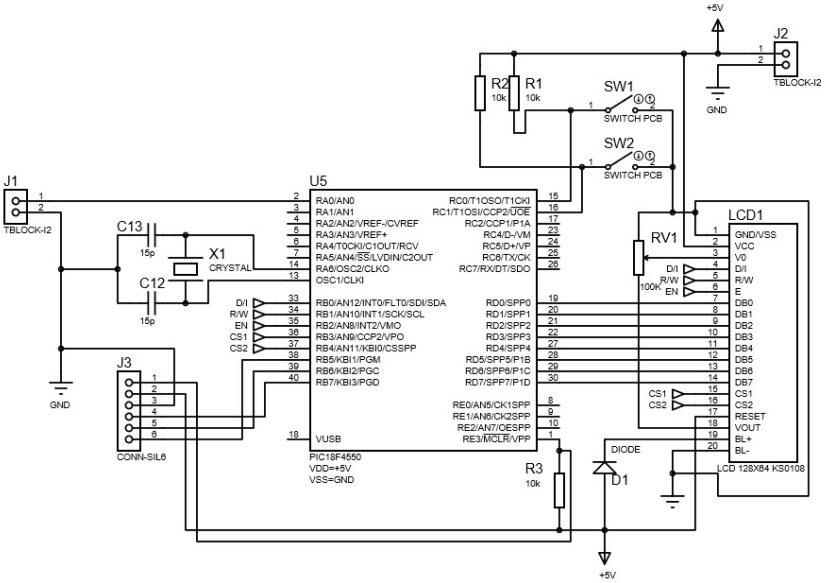

Figura 20: Esquemático PIC e Display – Fonte: o autor.

4.3.2 Layout circuito impresso ARES Proteus

Com os esquemáticos montados e os componentes devidamente dimensionados e com auxílio da extensão ARES do Proteus é possível desenhar o *layout* das placas de circuito impresso. A Figura 21, do *layout* da PCB do circuito de captação e amplificação do sinal de áudio serve como exemplo de *layout*. Os *layouts* restantes estão no Anexo 1. O *software* permite uma visualização 3D virtual da placa com componentes já

posicionados, tendo assim uma contemplação do resultado final que se procura na Figura 22.

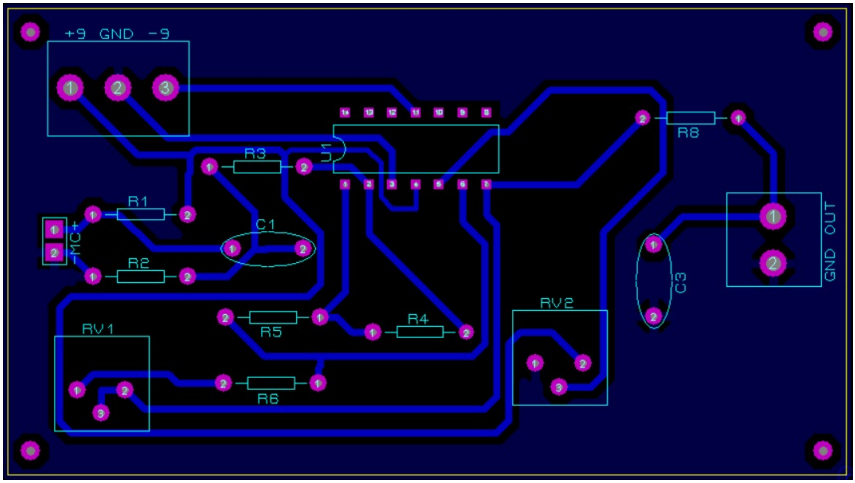

Figura 21: Layout Circuito de Captação e Amplificação – Fonte: o autor.

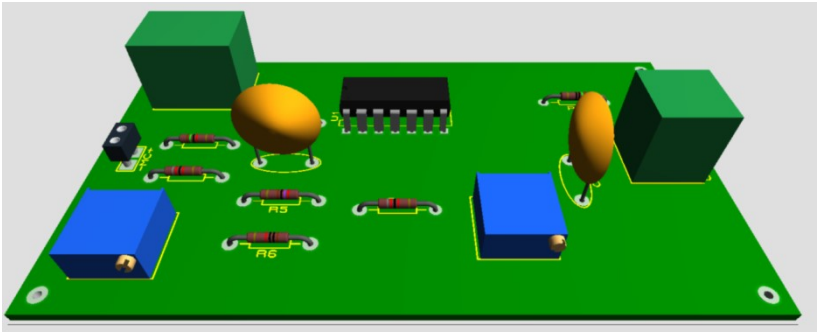

Figura 22: Modelagem 3D – Fonte: o autor.

4.3.3 Confecção das Placas

Para confecção das placas de circuito impresso foi utilizado o método de transferência termina e corrosão com percloreto de ferro. A partir do *layout* obtido no ARES pode-se transferir termicamente o desenho impresso em papel *Glossy* 180g em impressora laser para a placa de cobre. Para o processo foi utilizado uma prensa térmica com temperatura controlada em torno de 170 °C durante 6 minutos. A placa então é mergulhada em uma solução de percloreto de ferro por cerca de 20 minutos. Pode-se observar na Figura 23, ao lado esquerdo o resultado da corrosão ainda com a tinta impressa. Tinta essa que serve como película protetora do cobre, fazendo com que o percloreto de ferro atue somente nas áreas expostas. Na Figura 23, ao lado direito o resultado final do processo de corrosão, restando apenas as trilhas de cobre que configuram o circuito. O processo foi similar em todos os módulos.

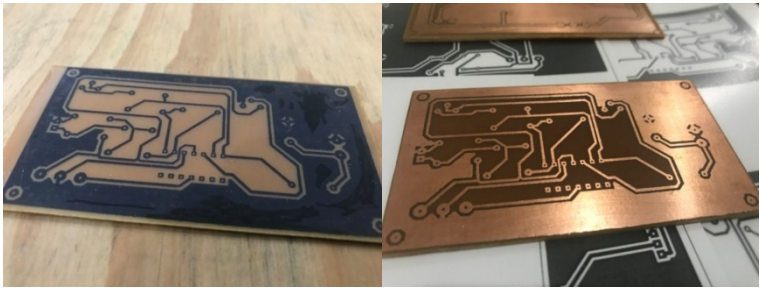

Figura 23: Placa após corrosão – Fonte: o autor.

Tendo o processo de corrosão concluído, com o auxílio de uma furadeira de bancada são feitas as furações nas marcações

onde os componentes foram fixados. Possibilitando assim a solda dos componentes. A Figura 24, mostra os módulos já montados e com os componentes soldados. Os *layouts* de todos os módulos estão no Anexo 1 e Anexo 2.

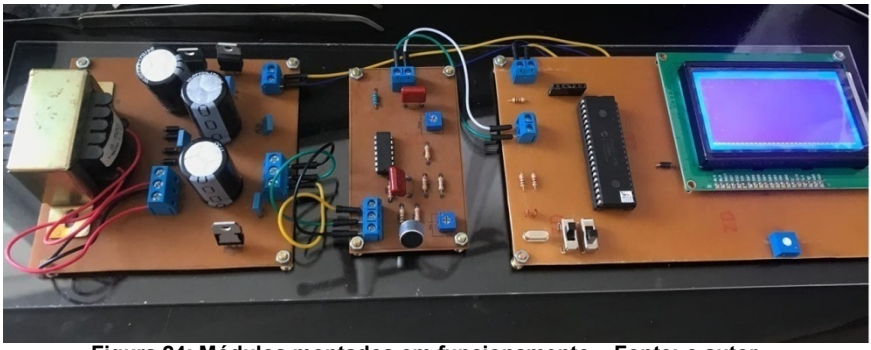

Figura 24: Módulos montados em funcionamento – Fonte: o autor

5. RESULTADOS

Com a parte de desenvolvimento e produção do projeto já concluída, pode-se observar a funcionalidade e realizar os devidos testes a fim de comprovar sua eficiência no problema proposto.

Os procedimentos de testes utilizados para averiguar o funcionamento do analisador de frequência foram divididos em três etapas. A primeira foi um teste simples apenas observando a resposta de captação em tempo real com sons emitidos através de fontes sonoras variadas, como ruído do ambiente, voz humana e palmas. Na segunda etapa utilizou-se um aplicativo de celular com função de gerador frequência, que consegue gerar ondas senoidais puras em uma frequência específica. Terceiro e último teste realizado foi comparar com um *software* de gravação equipado com um analisador de frequências próprio. A seguir a descrição dos processos de teste.

5.1 Teste com frequência específica

Para o teste de frequência especifica foi utilizado o aplicativo gratuito de celular *Tone Gen* (Figura 25), encontrado na *App Store*. Como são infinitas as possibilidades de frequências selecionáveis utilizou-se como padrão as frequências de 1 kHz, 4.5 kHz e 9.5 kHz, que compreendem todo o espectro do analisador, como visto na Tabela 1.

O teste com frequência específica além de validar a funcionalidade das FFT na detecção isolada de ondas sonoras mostra outra funcionalidade do analisador. O analisador pode ser utilizado na correção e equalização de monitores de palco quando ocorre a realimentação de microfones nas caixas de retorno. Este fenômeno ocorre devido às características de construção dos alto-falantes e das cápsulas de captação dos microfones, até mesmo o ambiente pode interferir. Determinadas frequências acabam possuindo maior ganho em relação às outras causando realimentação do sinal de forma a criar uma microfonia. Essas frequências de realimentação podem ser detectadas pelo analisador e posteriormente corrigidas em um equalizador de bandas.

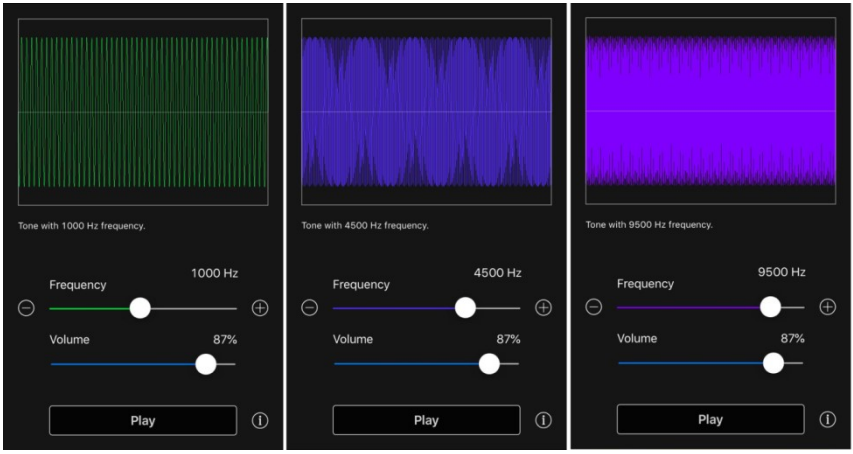

Figura 25: Tela do Aplicativo Tone Gen – Fonte: o autor

Na Figura 26 e na Figura 27, observa-se o analisador detectando as frequências de 1 kHz e 4,5 kHz. Na Figura 28, a frequência de 9,5 kHz.

Figura 26: Detectando 1 kHz – Fonte: o autor

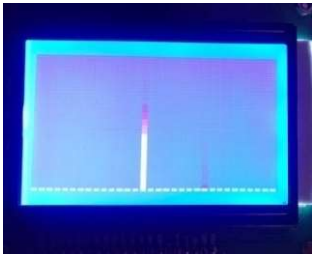

Figura 27: Detectando 4,5 kHz – Fonte: o autor

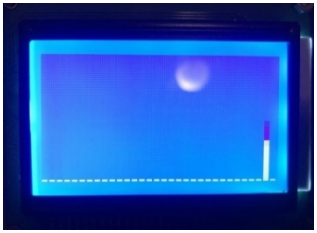

Figura 28: Detectando 9,5 kHz – Fonte: o autor

5.2 Teste de espectro com *software* de gravação

Para comparação com analisador de espectro profissional foi utilizado o *software* de gravação *Cubase 5* que possui *plug-ins* nativos específicos para tratamento de áudio. A Figura 29, mostra o *plug-in* nativo *TestGenerator* que possui recurso gerador de ruído rosa, assim pode-se observar as características do espectro no *plug-in*, que funciona como um analisador de espectro, *MultiScope*.

Figura 29: Software Cubase 5 e seus Plug-ins nativos – Fonte: o autor.

O analisador, devido as limitações do microcontrolador, compreende as frequências de 315 Hz a 10 kHz. O *software* por sua vez compreende uma faixa maior do espectro, mas se observado a faixa de resposta do analisador as características do

ruído rosa mostrada no *software* se assemelham as detectadas pelo analisador na Figura 30. Devido às próprias características do ruído, os picos de cada frequência tendem a oscilar entre altos e baixos, a fim de criar uma potência igual entre todas as oitavas. Por esta razão, na imagem não vemos o mesmo formato no gráfico de barras, mas vemos um padrão semelhante às características encontradas na literatura do que se espera do ruído rosa.

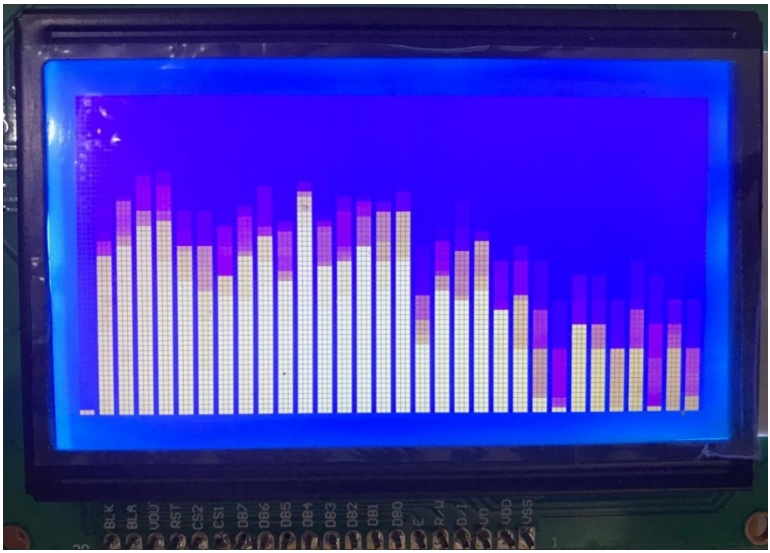

Figura 30: Gráfico ruído rosa no Analisador – Fonte: o autor.

6. CONCLUSÃO

A proposta do trabalho de desenvolver um analisador de frequência e espectro sonoro para alinhamento de sistemas de áudio pode ser cumprida com sucesso. Os testes realizados foram satisfatórios e chegaram aos resultados esperados, tanto nas análises de frequências específicas como na análise do espectro geral. Sendo assim, seria possível utilizá-lo com a finalidade de alinhar sistemas de áudio, partindo logicamente do conhecimento previu do operador em analisar e interpretar os dados obtidos. Ainda assim algumas considerações encontradas durante o desenvolvimento do projeto devem ser apontadas com a finalidade de melhorar o desempenho e funcionamento do analisador, os parágrafos seguintes apontam sugestões para melhoria do projeto.

A implementação de um gerador de sinal com ruído branco e ruído rosa, com o intuito de tornar a ferramenta mais completa e autônoma na questão de alinhamento de sistemas de som. Possibilitando ao técnico de som uma saída de sinal que seria ligada ao sistema e posteriormente captada pelo analisador.

Utilizar microfone condensador e omnidirecional. Neste caso, seria necessária uma revisão no circuito de aquisição e amplificação do sinal. O circuito seria muito semelhante ao já descrito nesse projeto, apenas observando as características do sinal captado pelo microfone e redimensionando componentes conforme necessidade. Assim, no caso de utilizar microfone condensador uma melhoria necessária ao projeto seria adicionar conectores XLR, padrão utilizado em equipamentos de áudio profissional, tanto na entrada do sinal quanto na saída do sinal

de ruído rosa. Facilitando comunicação com todos os equipamentos disponíveis no mercado.

Outras melhorias no campo da estética e interface com usuário. A confecção de circuito em uma única placa acoplada a uma caixa de montagem. Uma revisão no código adicionando um menu interativo através da implementação de um teclado para acesso as funções do dispositivo.

7. REFERÊNCIAS BIBLIOGRÁFICAS

Pereira, Fábio. Microcontroladores PIC: programação em C. 7. Ed. 2ª. Reimpressão. São Paulo: Érica, 2008.

Lakhani, K. R., & Von Hippel, E. (2003). How open source software works: "free" user-to-user assistance. Research Policy, 32(6), 923-943.

Munos, B. (2006). Can open-source R&D reinvigorate drug research? Nature Reviews Drug Discovery, 5(9), 723-729.

Lathi, B. P. Sinais e Sistemas Lineares; tradução Gustavo Guimarães Parma. 2ª Edição. Porto Alegre: Bookman, 2007.

Oppenheim, Alan V.; WILLSKY, Alan S. Sinais e sistemas. 2. Ed. São Paulo, SP: Pearson, 2010. 568 p.

Inns, Simon. Real-Time Audio Spectrum Analyser. Publicado em 08/01/2011 <https://www.waitingforfriday.com/?p=325> acesso em: 04/06/2019.

Page, Howard. Tuning and Optimising Large Scale Concert Sound Systems. Publicado em 11/12/2015 <https://www.audiotechnology.com/tutorials/the-howard-page-method-tuning-optimising-large-scale-concert-sound-systems> acesso em: 10/06/2019.

Lima, Raquel. Características do som: frequência, amplitude e timbre. Publicado em 27/01/2011 https://raquellima16.wordpress.com/2011/01/27/caracteristicas-do-som-frequencia-amplitude-e-timbre/ acesso em: 04/07/2019.

Datasheet TL084 < http://www.ti.com/lit/ds/symlink/tl084.pdf > acesso em: 04/07/2019.

Datasheet PIC18F4550 <https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf > acesso em: 04/07/2019.

PIC18 e seu sistema de clock <https://www.embarcados.com.br/pic18-e-seu-sistema-de-clock/> acessado em: 14/04/2019.

Análise de Sinais Discretizados <http://www.dsce.fee.unicamp.br/~antenor/pdffiles/qualidade/b4.pdf> acesso em: 12/06/2019.

Teorema da amostragem de Nyquist–Shannon <https://pt.wikipedia.org/wiki/Teorema_da_amostragem_de_Nyquist%E2%80%93Shannon> acesso em: 17/06/2019.

8. GLOSSÁRIO

Verbetes mencionados em itálico na descrição de um verbete possuem sua própria entrada no glossário:

AFSmart: placa de desenvolvimento equipada com o microcontrolador PIC 18F4550 para projetos em *protoboard.*

App Store: é um serviço de distribuição digital de aplicativos desenvolvido e operado pela Apple Inc. Ela é a loja oficial de aplicativos para o sistema operacional iOS, da Apple.

Bits: O bit é a menor unidade de informação que pode ser armazenada ou transmitida, usada na Computação e na Teoria da Informação. Um bit pode assumir somente 2 valores: 0 ou 1, corte ou passagem de energia, respectivamente.

Bootloader: um programa que serve para carregar o arquivo executável no microcontrolador. Ele carrega o programa *.hex* na memória do microcontrolador através da interface USB.

Capacitor: é um componente que armazena cargas elétricas num campo elétrico, acumulando um desequilíbrio interno de carga elétrica.

Chip: é um circuito eletrônico miniaturizado (composto principalmente por dispositivos semicondutores) sobre um substrato fino de material semicondutor.

Cubase 5: é um software de gravação e produção, sistema também chamado de DAW (Digital Audio Workstation), lançado

pela empresa alemã Steinberg, pertencente à Yamaha, em abril de 1989.

Datasheet: é um documento que resume o desempenho e outras características técnicas de um produto, máquina, componente, material, subsistema ou software em detalhe suficiente para que possa ser usado por um engenheiro de projeto para integrar o componente em um sistema.

Display: é um dispositivo para a apresentação de informação, de modo visual ou táctil, adquirida, armazenada ou transmitida sob várias formas. Quando a informação de entrada é fornecida como um sinal elétrico, o *display* é chamado de "*display* eletrônico".

Firmware: software de baixo nível embarcado em um hardware, com a função de comandar diretamente esse hardware e permitir comunicação do mesmo com o mundo externo.

Gate: Um FET para uso geral apresenta três terminais: porta (gate), fonte (source) e dreno (drain),que permitem seis formas de polarização, sendo três as mais usadas: fonte comum (fonte ligado à entrada e saída simultaneamente), porta comum (porta ligada à entrada e saída simultaneamente) e dreno comum (dreno ligado à entrada e saída simultaneamente).

Glossy: Papel de fotografia (papel convencional ou Inkjet) com superfície brilhante.

Hardware: Conjunto de componentes físicos, tipicamente eletrônicos, capaz de executar software em um sistema computacional.

Hertz: é a unidade de medida derivada do SI (Sistema Internacional) para frequência, a qual expressa, em termos de ciclos por segundo, a frequência de um evento periódico, oscilações (vibrações) ou rotações por segundo. Um dos seus principais usos é descrever ondas senoidais, como as de rádio ou sonoras. O seu nome foi em homenagem ao físico alemão Heinrich Hertz.

Interface: é o nome dado para o modo como ocorre a "comunicação" entre duas partes distintas e que não podem se conectar diretamente. Um software ou sistema operacional, por exemplo, pode ser controlado através de uma pessoa usando um computador

Layout: é uma palavra inglesa, muitas vezes usada na forma portuguesa "leiaute", que significa plano, arranjo, esquema, design, projeto.

Loop: na programação estruturada, são usados vários recursos para executar determinadas ações até que a condição seja satisfatória. Um outro tipo de estrutura usada é o que chamamos de rótulo para alguns programadores conhecidos como *loop*. Lembrando que *loop* em tecnologia da informação também é conhecido como um laço. No caso do comando *loop* ou laço seletivo que o torna possível o local e o momento que será usado em um programa para a verificação da condição de encerramento do mesmo.

MultiScope: extensão do software de gravação Cubase 5.

Offset: ajuste do referencial de tensão DC.

Percloreto de ferro: é um sal que em solução aquosa é usado para corrosão das placas de circuito impresso, tanto de fenolite como de outros plásticos.

PICkit3: é uma família de programadores para microcontroladores PIC da Microchip Technology. Eles são usados para programar e depurar microcontroladores, assim como programar a EEPROM. Alguns modelos também contam com analisador lógico e ferramenta de comunicação serial.

Plug-in: é um programa de computador usado para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica. Geralmente pequeno e leve, é usado somente sob demanda.

Protoboard: é uma placa com furos (ou orifícios) e conexões condutoras para montagem de circuitos elétricos experimentais.

Resistor: é um dispositivo elétrico muito utilizado em eletrônica, ora com a finalidade de transformar energia elétrica em energia térmica por meio do efeito joule, ora com a finalidade de limitar a corrente elétrica em um circuito.

Ruído branco: ou *White Noise* é um sinal aleatório com igual intensidade em diferentes frequências, o que lhe dá uma densidade espectral de potência constante. Com este significado e outros semelhantes, o termo é usado em muitas disciplinas científicas e técnicas, incluindo física, engenharia acústica,

telecomunicações, previsão estatística e muitas outras. O ruído branco se refere a um modelo estatístico para sinais e fontes de sinal, em vez de qualquer sinal específico.

Ruído rosa: é um sinal ou um processo onde o espectro de frequências como a densidade espectral de potência é inversamente proporcional à frequência do sinal. O termo originou-se pelas características desse ruído serem intermediárias entre o ruído branco (1/f0) e o ruído vermelho (1/f2), mais conhecido como ruído browniano.

Software: Conjunto de informações capaz de controlar um hardware a fim de executar uma tarefa em um sistema computacional.

Source: Um FET para uso geral apresenta três terminais: porta (gate), fonte (*source*) e dreno (*drain*),que permitem seis formas de polarização, sendo três as mais usadas: fonte comum (fonte ligado à entrada e saída simultaneamente), porta comum (porta ligada à entrada e saída simultaneamente) e dreno comum (dreno ligado à entrada e saída simultaneamente).

TesteGenerator: extensão do software de gravação Cubase 5.

Tone Gen: aplicativo de celular gerador de frequência senoidal especifica para IOS disponível no App Store**.**

Transistor: é um dispositivo semicondutor usado para amplificar ou trocar sinais eletrônicos e potência elétrica. É composto de material semicondutor com pelo menos três terminais para conexão a um circuito externo.

Web: *World Wide Web* (WWW) é um termo técnico (e anglicismo de tecnologia da informação) que foi traduzido para a língua portuguesa como rede mundial de computadores, também conhecido como *Web*, que designa um sistema de documentos em hipermídia que são interligados e executados na Internet.

9. Apêndice 1

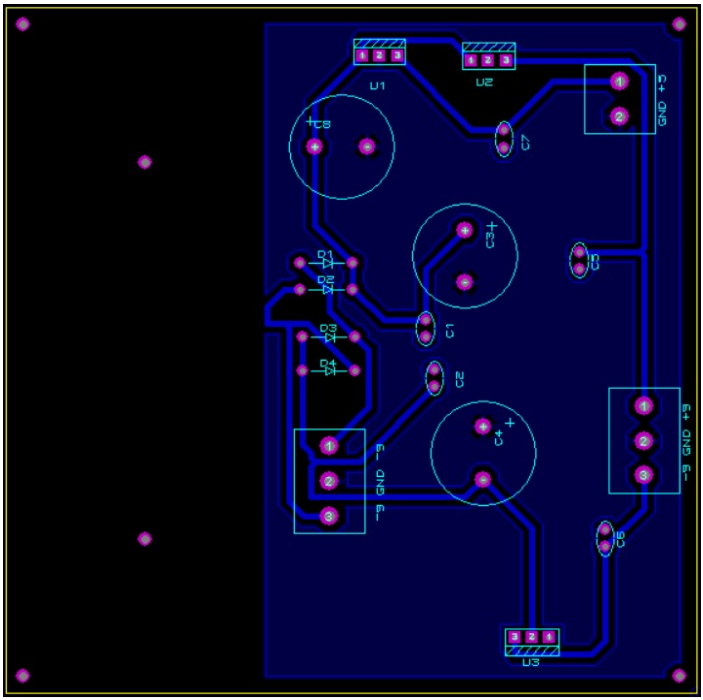

Figura 31: Layout Fonte de Alimentação – Fonte: o autor

10. Apêndice 2

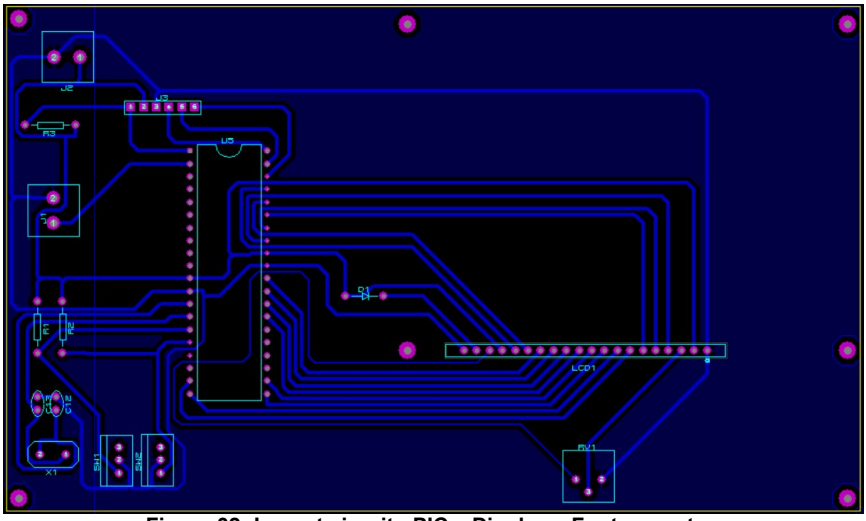

Figura 32: Layout circuito PIC e Display – Fonte: o autor.

11. Anexo 1 – Código Principal

```
/*****************************************************************************
        main.c

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

*****************************************************************************/
#ifndef MAIN_C
#define MAIN_C

// Global includes
#include <htc.h>
#include <stdio.h>

// Local includes
#include "hardware.h"
#include "fft.h"
#include "graph.h"

// PIC18F4550 fuse configuration:
// Config word 1 (Oscillator configuration)
// 20Mhz crystal input scaled to 48Mhz and configured for USB operation
//#pragma config CONFIG1L = 0x24
__CONFIG(1, PLLDIV_5 & CPUDIV_OSC1_PLL2 & USBDIV_2);
//#pragma config CONFIG1H = 0xE
```

```
__CONFIG (2, FOSC_HSPLL_HS & FCMEN_OFF & IESO_OFF);
//#pragma config CONFIG2L = 0x39
__CONFIG(3, PWRT_OFF & BOR_OFF & BORV_3 & VREGEN_ON);
//#pragma config CONFIG2H = 0x1E
__CONFIG(4, WDT_OFF & WDTPS_32768);
//#pragma config CONFIG3H = 0x81
__CONFIG(5, CCP2MX_ON & PBADEN_OFF & LPT1OSC_OFF & MCLRE_ON);
//#pragma config CONFIG4L = 0x81
__CONFIG(6, STVREN_ON & LVP_OFF & ICPRT_OFF & XINST_OFF);
//#pragma config CONFIG5L = 0xF
__CONFIG(7, CP0_OFF & CP1_OFF & CP2_OFF & CP3_OFF);
//#pragma config CONFIG5H = 0xC0
__CONFIG(8, CPB_OFF & CPD_OFF);
//#pragma config CONFIG6L = 0xF
__CONFIG(9, WRT0_OFF & WRT1_OFF & WRT2_OFF & WRT3_OFF);
//#pragma config CONFIG6H = 0xE0
__CONFIG(10, WRTC_OFF & WRTB_OFF & WRTD_OFF);
//#pragma config CONFIG7L = 0xF
__CONFIG(11, EBTR0_OFF & EBTR1_OFF & EBTR2_OFF & EBTR3_OFF);
//#pragma config CONFIG7H = 0x40
__CONFIG(12, EBTRB_OFF);

// Globals
short imaginaryNumbers[64];
short realNumbers[64];

void main(void)
{
	// PIC port set up -----------------------------------------------------

	// Configure on-board ADC
	// Vss and Vdd as voltage references
	ADCON1 = 0b00001110;

	// Configure the ADC acquisition time according to the datasheet
	ADCON2 = 0b10110110; // Note: output is right justified

	// Configure ports as inputs (1) or outputs(0)
	//       76543210
	TRISA = 0b00000001;
```

```
TRISB = 0b00000000;
TRISC = 0b00000011;
TRISD = 0b00000000;
TRISE = 0b00000000;

// Clear all ports
//       76543210
PORTA = 0b00000000;
PORTB = 0b00000000;
PORTC = 0b00000000;
PORTD = 0b00000000;
PORTE = 0b00000000;

RE0 = 0;
RE1 = 0;
RE2 = 0;

// Initialise the gLCD
gLcdInit();
gLcdClear();


while(1)
{
        // Perform the FFT

        // Get 64 samples at 50uS intervals
        // 50uS means our sampling rate is 20KHz which gives us
        // Nyquist limit of 10Khz
        short i = 0;
        unsigned short result;
        for (i = 0; i < 64; i++)
        {
                // Perform the ADC conversion
                // Select the desired ADC and start the conversion
                ADCON0 = 0b00000011;       //       Start       the       ADC
conversion on AN0

                // Wait for the ADC conversion to complete
```

```
                    TESTPIN_W4 = 1; // Don't remove this... it will affect the
sample timing
                    while(GODONE);
                    TESTPIN_W4 = 0; // Don't remove this... it will affect the
sample timing

                    // Get the 10-bit ADC result and adjust the virtual ground
of 2.5V
                    // back to 0Vs to make the input wave centered correctly
around 0
                    // (i.e. -512 to +512)
                    realNumbers[i] = ((short)(ADRESH << 8) +
(short)ADRESL) - 512;

                    // Set the imaginary number to zero
                    imaginaryNumbers[i] = 0;

                    // This delay is calibrated using an oscilloscope
according to the
                    // output on RA1 to ensure that the sampling periods are
correct
                    // given the overhead of the rest of the code and the
ADC sampling
                    // time.
                    //
                    // If you change anything in this loop or use the
professional
                    // (optimised) version of Hi-Tech PICC18, you will need
to re-
                    // calibrate this to achieve an accurate sampling rate.
                    __delay_us(7);
                }

                // Perform the (forward) FFT calculation

                // Note: the FFT result length is half of the input data length
                // so if we put 64 samples in we get 32 buckets out.  The first
bucket
                // cannot be used so in reality our result is 31 buckets.
                //
```

```
// The maximum frequency we can measure is half of the sampling
rate
// so if we sample at 20Khz our maximum is 10Khz (this is called
the
// Nyquist frequency).  So if we have 32 buckets divided over
10Khz,
// each bucket represents 312.5Khz of range, so our lowest bucket
is
// (bucket 1) 312.5Hz - 625Hz and so on up to our 32nd bucket
which
// is 9687.5Hz - 10,000Hz

//  1 : 312.5 - 625
//  2 : 625 - 937.5
//  3 : 937.5 - 1250
//  4 : 1250 - 1562.5
//  5 : 1562.5 - 1875
//  6 : 1875 - 2187.5
//  7 : 2187.5 - 2500
//  8 : 2500 - 2812.5
//  9 : 2812.5 - 3125
// 10 : 3125 - 3437.5
// 11 : 3437.5 - 3750
// 12 : 3750 - 4062.5
// 13 : 4062.5 - 4375
// 14 : 4375 - 4687.5
// 15 : 4687.5 - 5000
// 16 : 5000 - 5312.5
// 17 : 5312.5 - 5625
// 18 : 5625 - 5937.5
// 19 : 5937.5 - 6250
// 20 : 6250 - 6562.5
// 21 : 6562.5 - 6875
// 22 : 6875 - 7187.5
// 23 : 7187.5 - 7500
// 24 : 7500 - 7812.5
// 25 : 7812.5 - 8125
// 26 : 8125 - 8437.5
// 27 : 8437.5 - 8750
// 28 : 8750 - 9062.5
```

```
                    // 29 : 9062.5 - 9375
                    // 30 : 9375 - 9687.5
                    // 31 : 9687.5 - 10000

                    // Note: the '6' is the size of the input data (2 to the power of 6 =
64)
                    TESTPIN_W5 = 1;
                    fix_fft(realNumbers, imaginaryNumbers, 6);

                    // Take the absolute value of the FFT results

                    // Note: The FFT routine returns 'complex' numbers which consist
of
                    // both a real and imaginary part.  To find the 'absolute' value
                    // of the returned data we have to calculate the complex number's
                    // distance from zero which requires a little pythagoras and
therefore
                    // a square-root calculation too.  Since the PIC has multiplication
                    // hardware, only the square-root needs to be optimised.
                    long place, root;
      for (int k=0; k < 32; k++)
      {
                realNumbers[k] = (realNumbers[k] * realNumbers[k] +
            imaginaryNumbers[k] * imaginaryNumbers[k]);

          // Now we find the square root of realNumbers[k] using a very
          // fast (but compiler dependent) integer approximation:
          //                                  (adapted                                  from:
http://www.codecodex.com/wiki/Calculate_an_integer_square_root)
          place = 0x40000000;
                                  root = 0;

                                  if (realNumbers[k] >= 0) // Ensure we don't have a
negative number
                                  {
                                          while (place > realNumbers[k]) place = place
>> 2;

                                          while (place)
                                          {
```

```
                                                    if (realNumbers[k] >= root + place)
                                                    {
                                                            realNumbers[k]  -=  root  +
place;
                                                            root += place * 2;
                                                    }
                                                    root = root >> 1;
                                                    place = place >> 2;
                                            }
                                    }
                                    realNumbers[k] = root;
                            }
                            TESTPIN_W5 = 0;

                            // Now we have 32 buckets of audio frequency data represented as
                            // linear intensity in realNumbers[]
                            //
                            // Since the maximum input value (in theory) to the SQRT function is
                            // 32767, the peak output at this stage is SQRT(32767) = 181.

                            // Draw a bar graph of the FFT output data
                            TESTPIN_W6 = 1;
                            drawFftGraph(realNumbers);
                            TESTPIN_W6 = 0;
                    }
} #endif
```

12. Anexo 2 – Código Setup de Hardware

```
/*****************************************************************************
        hardware.h

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

*****************************************************************************/
#ifndef _HARDWARE_H_
#define _HARDWARE_H_

// Fosc frequency (48 Mhz)
#define _XTAL_FREQ 48000000

#define OFF                     0
#define ON                      1
#define TOGGLE  2

#define TRUE    1
#define FALSE   0

#define READ    1
#define WRITE   0

// Hardware mapping definitions
```

```
// gLCD character display hardware
#define    GLCD_DB0                    RD0
#define GLCD_DB1                  RD1
#define GLCD_DB2                  RD2
#define GLCD_DB3                  RD3
#define    GLCD_DB4                    RD4
#define GLCD_DB5                  RD5
#define GLCD_DB6                  RD6
#define GLCD_DB7                  RD7

#define GLCD_DATABUS              PORTD
#define GLCD_DATADIRECTION        TRISD

#define GLCD_RS                   RB0
#define GLCD_RW                   RB1
#define GLCD_EN                   RB2

#define GLCD_CS1                  RB3
#define GLCD_CS2                  RB4

#define LED0                      RE0
#define LED1                      RE1
#define LED2                      RE2

#define TESTPIN_W4                RA1
#define TESTPIN_W5                RA2
#define TESTPIN_W6                RA3

#define SWITCH0                   RC0
#define SWITCH1                   RC1

#endif
```

13. Anexo 3 – FFT source

```
/************************************************************************
        fft.c

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

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

#ifndef FFT_C
#define FFT_C

#include <htc.h>
#include "fft.h"

// fix_fft.c - Fixed-point in-place Fast Fourier Transform

// All data are fixed-point short integers, in which -32768
// to +32768 represent -1.0 to +1.0 respectively. Integer
// arithmetic is used for speed, instead of the more natural
// floating-point.
//
// For the forward FFT (time -> freq), fixed scaling is
// performed to prevent arithmetic overflow, and to map a 0dB
// sine/cosine wave (i.e. amplitude = 32767) to two -6dB freq
// coefficients.
```

```
//
// Written by:  Tom Roberts  11/8/89
// Made portable:  Malcolm Slaney 12/15/94 malcolm@interval.com
// Enhanced:  Dimitrios P. Bouras  14 Jun 2006 dbouras@ieee.org
// Ported to PIC18F:  Simon Inns 20110104

/*
  fix_fft() - perform forward fast Fourier transform.
  fr[n],fi[n] are real and imaginary arrays, both INPUT AND
  RESULT (in-place FFT), with 0 <= n < 2**m
*/
void fix_fft(short fr[], short fi[], short m)
{
        long int mr = 0, nn, i, j, l, k, istep, n, shift;
        short qr, qi, tr, ti, wr, wi;

        n = 1 << m;
        nn = n - 1;

        /* max FFT size = N_WAVE */
        //if (n > N_WAVE) return -1;

        /* decimation in time - re-order data */
        for (m=1; m<=nn; ++m)
        {
                l = n;
                do
                {
                        l >>= 1;
                } while (mr+l > nn);

                mr = (mr & (l-1)) + l;
                if (mr <= m) continue;

                tr = fr[m];
                fr[m] = fr[mr];
                fr[mr] = tr;
                ti = fi[m];
                fi[m] = fi[mr];
                fi[mr] = ti;
```

```
}

l = 1;
k = LOG2_N_WAVE-1;

while (l < n)
{
        /*
          fixed scaling, for proper normalization --
          there will be log2(n) passes, so this results
          in an overall factor of 1/n, distributed to
          maximize arithmetic accuracy.

          It may not be obvious, but the shift will be
          performed on each data point exactly once,
          during this pass.
        */

        // Variables for multiplication code
        long int c;
        short b;

        istep = l << 1;
        for (m=0; m<l; ++m)
        {
                j = m << k;
                /* 0 <= j < N_WAVE/2 */
                wr =  Sinewave[j+N_WAVE/4];
                wi = -Sinewave[j];

                wr >>= 1;
                wi >>= 1;

                for (i=m; i<n; i+=istep)
                {
                        j = i + l;

                        // Here I unrolled the multiplications to prevent
overhead
```

```
                                        // for procedural calls (we don't need to be
clever about
                                        // the actual multiplications since the pic has
an onboard
                                        // 8x8 multiplier in the ALU):

                                        // tr = FIX_MPY(wr,fr[j]) - FIX_MPY(wi,fi[j]);
                                        c = ((long int)wr * (long int)fr[j]);
                                        c = c >> 14;
                                        b = c & 0x01;
                                        tr = (c >> 1) + b;

                                        c = ((long int)wi * (long int)fi[j]);
                                        c = c >> 14;
                                        b = c & 0x01;
                                        tr = tr - ((c >> 1) + b);

                                        // ti = FIX_MPY(wr,fi[j]) + FIX_MPY(wi,fr[j]);
                                        c = ((long int)wr * (long int)fi[j]);
                                        c = c >> 14;
                                        b = c & 0x01;
                                        ti = (c >> 1) + b;

                                        c = ((long int)wi * (long int)fr[j]);
                                        c = c >> 14;
                                        b = c & 0x01;
                                        ti = ti + ((c >> 1) + b);

                                        qr = fr[i];
                                        qi = fi[i];
                                        qr >>= 1;
                                        qi >>= 1;

                                        fr[j] = qr - tr;
                                        fi[j] = qi - ti;
                                        fr[i] = qr + tr;
                                        fi[i] = qi + ti;
                                }
                        }
```

```
                        --k;
                        l = istep;
                }
}
#endif
```

14. Anexo 4 – Biblioteca FFT

```
/************************************************************************
        fft.h

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

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

#ifndef _FFT_H
#define _FFT_H

// Definitions
#define N_WAVE      1024    // full length of Sinewave[]
#define LOG2_N_WAVE 10      // log2(N_WAVE)

// Since we only use 3/4 of N_WAVE, we define only
// this many samples, in order to conserve data space.
const short Sinewave[N_WAVE-N_WAVE/4] = {
     0,   201,   402,   603,   804,  1005,  1206,  1406,
  1607,  1808,  2009,  2209,  2410,  2610,  2811,  3011,
  3211,  3411,  3611,  3811,  4011,  4210,  4409,  4608,
  4807,  5006,  5205,  5403,  5601,  5799,  5997,  6195,
  6392,  6589,  6786,  6982,  7179,  7375,  7571,  7766,
  7961,  8156,  8351,  8545,  8739,  8932,  9126,  9319,
```

9511, 9703, 9895, 10087, 10278, 10469, 10659, 10849,
11038, 11227, 11416, 11604, 11792, 11980, 12166, 12353,
12539, 12724, 12909, 13094, 13278, 13462, 13645, 13827,
14009, 14191, 14372, 14552, 14732, 14911, 15090, 15268,
15446, 15623, 15799, 15975, 16150, 16325, 16499, 16672,
16845, 17017, 17189, 17360, 17530, 17699, 17868, 18036,
18204, 18371, 18537, 18702, 18867, 19031, 19194, 19357,
19519, 19680, 19840, 20000, 20159, 20317, 20474, 20631,
20787, 20942, 21096, 21249, 21402, 21554, 21705, 21855,
22004, 22153, 22301, 22448, 22594, 22739, 22883, 23027,
23169, 23311, 23452, 23592, 23731, 23869, 24006, 24143,
24278, 24413, 24546, 24679, 24811, 24942, 25072, 25201,
25329, 25456, 25582, 25707, 25831, 25954, 26077, 26198,
26318, 26437, 26556, 26673, 26789, 26905, 27019, 27132,
27244, 27355, 27466, 27575, 27683, 27790, 27896, 28001,
28105, 28208, 28309, 28410, 28510, 28608, 28706, 28802,
28897, 28992, 29085, 29177, 29268, 29358, 29446, 29534,
29621, 29706, 29790, 29873, 29955, 30036, 30116, 30195,
30272, 30349, 30424, 30498, 30571, 30643, 30713, 30783,
30851, 30918, 30984, 31049, 31113, 31175, 31236, 31297,
31356, 31413, 31470, 31525, 31580, 31633, 31684, 31735,
31785, 31833, 31880, 31926, 31970, 32014, 32056, 32097,
32137, 32176, 32213, 32249, 32284, 32318, 32350, 32382,
32412, 32441, 32468, 32495, 32520, 32544, 32567, 32588,
32609, 32628, 32646, 32662, 32678, 32692, 32705, 32717,
32727, 32736, 32744, 32751, 32757, 32761, 32764, 32766,
32767, 32766, 32764, 32761, 32757, 32751, 32744, 32736,
32727, 32717, 32705, 32692, 32678, 32662, 32646, 32628,
32609, 32588, 32567, 32544, 32520, 32495, 32468, 32441,
32412, 32382, 32350, 32318, 32284, 32249, 32213, 32176,
32137, 32097, 32056, 32014, 31970, 31926, 31880, 31833,
31785, 31735, 31684, 31633, 31580, 31525, 31470, 31413,
31356, 31297, 31236, 31175, 31113, 31049, 30984, 30918,
30851, 30783, 30713, 30643, 30571, 30498, 30424, 30349,
30272, 30195, 30116, 30036, 29955, 29873, 29790, 29706,
29621, 29534, 29446, 29358, 29268, 29177, 29085, 28992,
28897, 28802, 28706, 28608, 28510, 28410, 28309, 28208,
28105, 28001, 27896, 27790, 27683, 27575, 27466, 27355,
27244, 27132, 27019, 26905, 26789, 26673, 26556, 26437,
26318, 26198, 26077, 25954, 25831, 25707, 25582, 25456,

25329, 25201, 25072, 24942, 24811, 24679, 24546, 24413,
24278, 24143, 24006, 23869, 23731, 23592, 23452, 23311,
23169, 23027, 22883, 22739, 22594, 22448, 22301, 22153,
22004, 21855, 21705, 21554, 21402, 21249, 21096, 20942,
20787, 20631, 20474, 20317, 20159, 20000, 19840, 19680,
19519, 19357, 19194, 19031, 18867, 18702, 18537, 18371,
18204, 18036, 17868, 17699, 17530, 17360, 17189, 17017,
16845, 16672, 16499, 16325, 16150, 15975, 15799, 15623,
15446, 15268, 15090, 14911, 14732, 14552, 14372, 14191,
14009, 13827, 13645, 13462, 13278, 13094, 12909, 12724,
12539, 12353, 12166, 11980, 11792, 11604, 11416, 11227,
11038, 10849, 10659, 10469, 10278, 10087, 9895, 9703,
9511, 9319, 9126, 8932, 8739, 8545, 8351, 8156,
7961, 7766, 7571, 7375, 7179, 6982, 6786, 6589,
6392, 6195, 5997, 5799, 5601, 5403, 5205, 5006,
4807, 4608, 4409, 4210, 4011, 3811, 3611, 3411,
3211, 3011, 2811, 2610, 2410, 2209, 2009, 1808,
1607, 1406, 1206, 1005, 804, 603, 402, 201,
0, -201, -402, -603, -804, -1005, -1206, -1406,
-1607, -1808, -2009, -2209, -2410, -2610, -2811, -3011,
-3211, -3411, -3611, -3811, -4011, -4210, -4409, -4608,
-4807, -5006, -5205, -5403, -5601, -5799, -5997, -6195,
-6392, -6589, -6786, -6982, -7179, -7375, -7571, -7766,
-7961, -8156, -8351, -8545, -8739, -8932, -9126, -9319,
-9511, -9703, -9895, -10087, -10278, -10469, -10659, -10849,
-11038, -11227, -11416, -11604, -11792, -11980, -12166, -12353,
-12539, -12724, -12909, -13094, -13278, -13462, -13645, -13827,
-14009, -14191, -14372, -14552, -14732, -14911, -15090, -15268,
-15446, -15623, -15799, -15975, -16150, -16325, -16499, -16672,
-16845, -17017, -17189, -17360, -17530, -17699, -17868, -18036,
-18204, -18371, -18537, -18702, -18867, -19031, -19194, -19357,
-19519, -19680, -19840, -20000, -20159, -20317, -20474, -20631,
-20787, -20942, -21096, -21249, -21402, -21554, -21705, -21855,
-22004, -22153, -22301, -22448, -22594, -22739, -22883, -23027,
-23169, -23311, -23452, -23592, -23731, -23869, -24006, -24143,
-24278, -24413, -24546, -24679, -24811, -24942, -25072, -25201,
-25329, -25456, -25582, -25707, -25831, -25954, -26077, -26198,
-26318, -26437, -26556, -26673, -26789, -26905, -27019, -27132,
-27244, -27355, -27466, -27575, -27683, -27790, -27896, -28001,
-28105, -28208, -28309, -28410, -28510, -28608, -28706, -28802,

```
 -28897, -28992, -29085, -29177, -29268, -29358, -29446, -29534,
 -29621, -29706, -29790, -29873, -29955, -30036, -30116, -30195,
 -30272, -30349, -30424, -30498, -30571, -30643, -30713, -30783,
 -30851, -30918, -30984, -31049, -31113, -31175, -31236, -31297,
 -31356, -31413, -31470, -31525, -31580, -31633, -31684, -31735,
 -31785, -31833, -31880, -31926, -31970, -32014, -32056, -32097,
 -32137, -32176, -32213, -32249, -32284, -32318, -32350, -32382,
 -32412, -32441, -32468, -32495, -32520, -32544, -32567, -32588,
 -32609, -32628, -32646, -32662, -32678, -32692, -32705, -32717,
 -32727, -32736, -32744, -32751, -32757, -32761, -32764, -32766,
};
// Function prototypes
void fix_fft(short fr[], short fi[], short m);
#endif
```

15. Anexo 5 – Graphic Source

```
/*****************************************************************************
        graph.c

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

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

#ifndef GRAPH_C
#define GRAPH_C

// Global includes
#include <htc.h>

// Local includes
#include "hardware.h"
#include "graph.h"

// Write a byte directly to the screen hardware (quick version)
void fLcdWrite(unsigned char cmdType, unsigned char bank, unsigned char byte)
{
        // Wait for the busy flag to clear
        GLCD_DATADIRECTION = GLCD_READ;
        GLCD_RW = READ;
        GLCD_RS = GLCD_CMD;
```

```
	do
	{
		// Strobe the EN line
		GLCD_EN = 1;
		__delay_us(ENSTROBEDELAY);
		GLCD_EN = 0;
	} while (GLCD_DB7 == 1);

	// Select the command type
	GLCD_RS = cmdType;

	// Select the screen bank
	if (bank == 0)
	{
		GLCD_CS1 = 1;
		GLCD_CS2 = 0;
	}
	else
	{
		GLCD_CS1 = 0;
		GLCD_CS2 = 1;
	}

	// Place the byte on the databus
	GLCD_DATADIRECTION = GLCD_WRITE;
	GLCD_RW = WRITE;
	GLCD_DATABUS = byte;

	// Strobe the EN line
	GLCD_EN = 1;
	__delay_us(ENSTROBEDELAY);
	GLCD_EN = 0;
}

// Global to hold the display data (required for output damping)
short displayData[32];

// Plot the output graph
void drawFftGraph(short inputData[])
```

```
{
	short inputValue;

	// Scale the input data to 0-63 and perform the dampening of the display
	for (unsigned char counter = 1; counter < 32; counter++)
	{
		// Scale the input data for the display (linear) x1 or x8
		if (SWITCH0 == 0) inputValue = inputData[counter] * 8;
		else inputValue = inputData[counter];
		if (inputValue > 181) inputValue = 181;

		// Apply a linear or logarithmic conversion on the data
		if (SWITCH1 == 0) inputValue = (short)logTable[inputValue];
		else inputValue = (short)linTable[inputValue];

		// Perform damping on the displayed output
		if (inputValue > displayData[counter]) displayData[counter] =
inputValue;
		else displayData[counter] -= 10;
		if (displayData[counter] < 0) displayData[counter] = 0;
	}

	// Our FFT animation speed is dependent on how fast the LCD can
	// be updated, so here we use a bargraph drawing routine which
	// is highly optimised to the manner in which the LCD is updated.
	unsigned char xByte, requiredY, y, pointer;
	for (y = 0; y < 8; y++)
	{
		// Move to the correct screen page

		// Left bank
		fLcdWrite(GLCD_CMD, 0, y | 0b10111000);
		fLcdWrite(GLCD_CMD, 0, 0b01000000);

		// Right bank
		fLcdWrite(GLCD_CMD, 1, y | 0b10111000);
		fLcdWrite(GLCD_CMD, 1, 0b01000000);

		unsigned char xPos = 0;
```

```
                        // We only draw buckets 1 to 31 (bucket 0 is invalid)
                        for (pointer = 0; pointer < 32; pointer++)
                        {
                                xByte = 0;
                                requiredY = 63 - displayData[pointer];

                                // Either fill the whole byte or
                                // left shift according to the remainder of
                                // the division to get the right number of pixels
                                if (requiredY <= y * 8) xByte = 0b11111111;
                                else if (requiredY / 8 <= y) xByte = 0b11111111 <<
(requiredY % 8);

                                if (xPos < 64) fLcdWrite(GLCD_DATA, 0, xByte); // 1/3
of bar
                                else fLcdWrite(GLCD_DATA, 1, xByte);
                                xPos++;

                                if (xPos < 64) fLcdWrite(GLCD_DATA, 0, xByte); // 1/3
of bar
                                else fLcdWrite(GLCD_DATA, 1, xByte);
                                xPos++;

                                if (xPos < 64) fLcdWrite(GLCD_DATA, 0, xByte); // 1/3
of bar
                                else fLcdWrite(GLCD_DATA, 1, xByte);
                                xPos++;

                                if (xPos < 64) fLcdWrite(GLCD_DATA, 0, 0x00); // gap
                                else fLcdWrite(GLCD_DATA, 1, 0x00);
                                xPos++;
                        }
        }
}

// Initialise the gLCD
void gLcdInit(void)
{
        // Bring the display out of reset
        //GLCD_RES = 0; // Screen in reset
```

```
	__delay_us(250);
	//GLCD_RES = 1; // Screen out of reset
	__delay_us(250);
	// Set Y Address = 0
	fLcdWrite(GLCD_CMD, 0, 0b01000000);
	fLcdWrite(GLCD_CMD, 1, 0b01000000);
	// Set X Address = 0
	fLcdWrite(GLCD_CMD, 0, 0b10111000);
	fLcdWrite(GLCD_CMD, 1, 0b10111000);
	// Set Display start line = 0
	fLcdWrite(GLCD_CMD, 0, 0b11000000);
	fLcdWrite(GLCD_CMD, 1, 0b11000000);
	// Turn the display ON
	fLcdWrite(GLCD_CMD, 0, 0b00111111);
	fLcdWrite(GLCD_CMD, 1, 0b00111111);
}

// Clear the gLCD to black (zero)
void gLcdClear(void)
{
	unsigned char x, y;

	for (y = 0; y < 8; y++)
	{
		// Move to the correct screen page
		fLcdWrite(GLCD_CMD, 0, y | 0b10111000);
		fLcdWrite(GLCD_CMD, 0, 0b01000000);

		fLcdWrite(GLCD_CMD, 1, y | 0b10111000);
		fLcdWrite(GLCD_CMD, 1, 0b01000000);

		for (x = 0; x < 64; x++)
		{
			fLcdWrite(GLCD_DATA, 0, 0x00);
			fLcdWrite(GLCD_DATA, 1, 0x00);
		}
	}
}

#endif
```

16. Anexo 6 – Biblioteca Display Gráfico

```
/************************************************************************
        graph.h

FFT Audio Analysis
Copyright (C) 2011 Simon Inns
Este programa é software livre: você pode redistribuí-lo e / ou modificar
sob os termos da Licença Pública Geral GNU, conforme publicada pela
a Free Software Foundation, versão 3 da Licença, ou
(a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que seja útil,
mas SEM QUALQUER GARANTIA; sem mesmo a garantia implícita de
COMERCIABILIDADE OU ADEQUAÇÃO A UM DETERMINADO FIM. Veja o
GNU General Public License para mais detalhes.

Você deveria ter recebido uma cópia da Licença Pública Geral GNU
Junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.

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

#ifndef _GRAPH_H_
#define _GRAPH_H_

// This table is used to convert the FFT output range of 0-181 to the
// display's Y resolution of 64 pixels using a linear scale.
const unsigned char linTable[182] =
{
        0, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8,
        8, 9, 9, 9, 10, 10, 10, 11, 11, 11, 12, 12, 12, 13, 13, 13, 14, 14,
        14, 15, 15, 15, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 20, 20,
        20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 25, 25, 25, 26, 26,
        26, 27, 27, 27, 28, 28, 28, 29, 29, 29, 30, 30, 30, 31, 31, 31, 32,
        32, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 36, 36, 37, 37, 37, 38,
        38, 38, 39, 39, 39, 40, 40, 41, 41, 41, 42, 42, 42, 43, 43, 43, 44,
        44, 44, 45, 45, 45, 46, 46, 46, 47, 47, 47, 48, 48, 49, 49, 49, 50,
        50, 50, 51, 51, 51, 52, 52, 52, 53, 53, 53, 54, 54, 54, 55, 55, 55,
        56, 56, 57, 57, 57, 58, 58, 58, 59, 59, 59, 60, 60, 60, 61, 61, 61,
```

```
        62, 62, 62, 63, 63, 63
};

// This table is used to convert the FFT output range of 0-181 to the
// display's Y resolution of 64 pixels using a logarithmic scale.
// Note: this table is based on a 'dB' conversion of a = 18.1 * log(x/181)
// with some shifting to prevent the noise-floor being displayed.  It's
// basically useless for audio analysis, but makes for a nice looking
// output :)
const unsigned char logTable[182] =
{
        0, 0, 22, 27, 30, 33, 34, 36, 37, 38, 39, 40, 40, 41, 42, 42, 43, 43,
        44, 44, 45, 45, 46, 46, 46, 47, 47, 47, 48, 48, 48, 48, 49, 49, 49,
        49, 50, 50, 50, 50, 50, 51, 51, 51, 51, 51, 52, 52, 52, 52, 52, 52,
        53, 53, 53, 53, 53, 53, 53, 54, 54, 54, 54, 54, 54, 54, 54, 55, 55,
        55, 55, 55, 55, 55, 55, 55, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56,
        57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 58, 58, 58, 58, 58, 58,
        58, 58, 58, 58, 58, 58, 58, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59,
        59, 59, 59, 59, 59, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
        60, 60, 60, 60, 60, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61,
        61, 61, 61, 61, 61, 61, 61, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
        62, 62, 62, 62, 62, 62, 62, 62, 62, 63, 63
};

// This define states the delay when strobing the EN
// line in uS
#define ENSTROBEDELAY 10

// gLCD definitions
#define GLCD_READ                       0b11111111
#define GLCD_WRITE                      0b00000000
#define GLCD_CMD                        0
#define GLCD_DATA                       1

// Function prototypes
void drawFftGraph(short inputData[]);
void gLcdInit(void);
void gLcdClear(void);

#endif
```