UNIVERSIDADE FEDERAL DO PARANÁ

ANDRÉ HEIDEMANN IAROZINSKI

BRUNO DE ALMEIDA DE FREITAS BARBOSA

INTERFACE DE COMUNICAÇÃO COM O INSTRUMENTO MUSICAL TEREMIM COM SAÍDA EM FORMATO MIDI

CURITIBA
2017

ANDRÉ HEIDEMANN IAROZINSKI

BRUNO DE ALMEIDA DE FREITAS BARBOSA

INTERFACE DE COMUNICAÇÃO COM O INSTRUMENTO MUSICAL TEREMIM COM SAÍDA EM FORMATO MIDI

Monografia apresentada como requisito parcial à obtenção do título de Bacharel, Curso de Bacharelado em Engenharia Elétrica, Setor de Tenologia, Universidade Federal do Paraná.

Orientador: Prof. Dr. Marlio J. C. Bonfim

CURITIBA
2017

TERMO DE APROVAÇÃO

ANDRÉ HEIDEMANN IAROZINSKI

BRUNO DE ALMEIDA DE FREITAS BARBOSA

INTERFACE DE COMUNICAÇÃO COM O INSTRUMENTO MUSICAL TEREMIM COM SAÍDA EM FORMATO MIDI

TRABALHO APRESENTADO AO CURSO DE ENGENHARIA ELÉTRICA, DA UNIVERSIDADE FEDERAL DO PARANÁ, COMO REQUISITO À OBTENÇÃO DO TÍTULO DE GRADUAÇÃO.

_____
Profº. Dr. Marlio José do Couto Bonfim
Orientador - Departamento de Engenharia Elétrica

_____
Profº. Dr. Eduardo Parente Ribeiro
Avaliador 1 - Departamento de Engenharia Elétrica

_____
Profº. Henri Frederico Eberspacher, PhD
Avaliador 2 - Departamento de Engenharia Elétrica

Dedico este trabalho primeiramente à minha família, pelo seu apoio e incentivo durante todo o período da graduação. E também à minha namorada Jakeline, por toda sua compreensão, apoio e incentivo durante essa longa caminhada e importante etapa da minha vida.

André Heidemann Iarozinski

Dedico não só este trabalho, mas como toda a minha graduação à memória do meu avô, Pedro de Almeida, que até as últimas semanas de conclusão deste trabalho esteve me incentivando a manter o foco e a dedicação.

Bruno de Almeida de Freitas Barbosa

AGRADECIMENTOS

Agradecemos primeiramente ao nosso orientador, professor Marlio, por acreditar neste projeto e por ter estado sempre disponível para nos ajudar a solucionar os problemas encontrados. Agradecimentos especiais para: Bruno Ricobom, Orestes, Vitoria e toda a equipe do NPDEAS.

Agradecemos aos professores participantes da banca examinadora que dividem conosco este momento tão importante e esperado:

Prof. Dr. Eduardo Parente Ribeiro e Prof. Henri Frederico Eberspacher, PhD.

Gostaríamos também de agradecer a todos que acreditam no poder que a música tem em mudar as vidas das pessoas.

RESUMO

Criada na década de 80, a tecnologia MIDI (Interface Digital para instrumentos Musicais) ainda se mantém até hoje como uma referência no setor de produção musical. Esta tecnologia que descreve um protocolo de interfaceamento digital, inovou e transformou a maneira de se fazer música e atualmente está presente em quase todos os estúdios de gravação e palcos ao redor do mundo. Este trabalho descreve o desenvolvimento de uma interface de conversão MIDI para o instrumento musical Teremim, que é na sua essência um instrumento eletrônico analógico. A informação de saída mais relevante do Teremim é a frequência da nota musical emitida, que é monitorada e processada por um microcontrolador (Tiva C - Texas Instruments), convertida em dados do protocolo MIDI e enviada a um instrumento virtual instalado em um computador (VSTi). A interface fornece uma realimentação visual ao músico, através de um visor de LEDs RGB, que informa a nota tocada codificada em cores, facilitando a execução musical do instrumento. Também possui um controle manual que permite definir o volume, através de um resistor sensível à força, e a forma como a nota será tocada, possibilitando além do comportamento tradicional do Teremim, o disparo de notas de maneira controlada. A interface foi projetada para ser utilizada em conjunto com softwares de instrumentos virtuais e interfaces de áudio que possuam comunicação MIDI. Desta forma o músico pode escolher diversos parâmetros sonoros, ampliando as possibilidades musicais do Teremim. O projeto resultou em um protótipo funcional, de acordo com os objetivos propostos e compatível com um modelo de Teremim nacional de baixo custo.

Palavras Chave: Teremim, MIDI, VSTi

ABSTRACT

Created in the 80’s, the MIDI (Musical Instrument Digital Interface) technology still remains a reference in the sector of the music industry. This technology describes a digital interface protocol and has innovated the way of producing music and is currently present in most recording studios and stages around the world. This monograph describes the development of an interface capable of converting the audio signal generated by a Theremin (an analogue electronic musical instrument) to MIDI. The most relevant output information from the Theremin is the frequency of the musical note generated, which is monitored and processed by a microcontroller (Tiva C – Texas Instruments), converted in MIDI data and sent to a virtual instrument installed in a computer (VSTi). The interface provides a visual feedback to the musician through a RGB LED display, which returns the musical note played in colours, facilitating the musical performance of Theremin. There is also a manual controller which allows control to the MIDI parameter velocity through a force sensitive resistor and a button capable of controlling the moment when a musical note is played, in addition to the traditional Theremin behavior. The interface was designed to be used in conjunction with virtual instruments software and audio interfaces that have MIDI communication. Thus the musician can choose several sound parameters, expanding the musical possibilities of a Theremin. The project resulted in a functional prototype, in accordance with the proposed objectives and compatible with a Brazilian low-cost Theremin.

Key words: Theremin, MIDI, VSTi

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AC – *Alternate Current*

ADC – *Analog to Digital Converter*

BJT – *Bipolar Junction Transistor*

CI – Circuito Integrado

DAW – *Digital Audio Workstation*

FSR – *Force Sensitive Resistor*

IDE – *Integrated Development Environment*

LED – *Light Emitting Diode*

MIDI – *Musical Instrument Digital Interface*

PCB – *Printed Circuit Board*

RGB – *Red Green Blue*

SNR – *Signal-to-Noise Ratio*

UART – *Universal Asynchronous Receiver/Transmitter*

USB – *Universal Serial Bus*

V_{IH} – *Input High Voltage*

V_{IL} – *Input Low Voltage*

VST – *Virtual Studio Technology*

VSTi – *Virtual Studio Technology Instrument*

LISTA DE SÍMBOLOS

V – volt

A – ampère

Ω – ohm

Hz – hertz

m – mili – 10^{-3}

M – mega – 10^{6}

μ - micro – 10^{-6}

p – pico – 10^{-12}

SUMÁRIO

1 INTRODUÇÃO

Em diversas aplicações de áudio profissional existem dispositivos eletrônicos que trabalham com sinais analógicos e digitais. Atualmente a maior parte dos sinais utilizados em produção e edição de áudio são digitais, pois a capacidade de processamento computacional atingiu um nível suficientemente elevado em termos de taxa de amostragem, faixa dinâmica de conversão analógico/digital e performance.

Um dos tipos de protocolos digitais responsáveis por uma profunda mudança na maneira em que se realiza a produção e edição de áudio voltado para a música foi o protocolo de comunicação MIDI, *Musical Instrument Digital Interface*. A principal vantagem da sua utilização é que ele transporta dados muito mais compactos que o áudio digitalizado proveniente do áudio analógico original, pelo fato de enviar dados apenas na forma de instruções e comandos.

Considerando o momento atual da tecnologia, este projeto propõe a construção de uma interface de conversão Áudio/MIDI desenvolvida com o microcontrolador Tiva C para ser utilizada com um Teremim. Teremim é um instrumento musical tocado sem qualquer contato físico que não se popularizou devido à grande dificuldade de execução musical decorrente da continuidade espectral e de sua alta sensibilidade.

A interface apresentada neste trabalho objetiva facilitar o uso do instrumento, aumentar sua gama de possibilidades sonoras em termos de timbres e permitir um controle de variação dinâmica para o instrumento. Essa interface é um dispositivo que não emite sons, mas gera comandos MIDI para que um *software* VSTi, *Virtual Studio Technology Instrument,* interprete suas mensagens e sintetize o áudio proveniente do Teremim de maneira digital.

Este projeto tem como justificativa a atual demanda cultural consequente dos avanços tecnológicos surgidos na última década, tornando o protocolo MIDI acessível a músicos independentes e estúdios de pequeno porte. Antes da popularização dos *softwares* VSTi o protocolo MIDI era utilizado majoritariamente em estúdios de grande porte e palcos de grandes artistas.

Aliada à motivação pessoal dos autores, como músicos independentes há mais de 7 anos, existe uma motivação técnica em aplicar os conhecimentos adquiridos na graduação de engenharia elétrica em um projeto com fins musicais. Espera-se que este trabalho venha a incentivar mais pessoas a produzir música ou outras formas de arte com o auxílio da eletrônica e programação, conforme foi realizado no trabalho de conclusão de curso do aluno Rodrigo Uehara Guskuma da Universidade de São Paulo, onde foi desenvolvido um controlador MIDI com linguagem de alto nível para fins didáticos. **[1]**

O presente projeto tem como objetivos específicos:

• Realizar uma revisão teórica sobre o instrumento Teremim, protocolo MIDI, microcontrolador Tiva C e VSTi

• Analisar o sinal de áudio analógico gerado pelo Teremim

• Realizar o desenvolvimento de *software* utilizando o microcontrolador Tiva C, e desenvolver *hardwares* auxiliares ao microcontrolador

• Elaborar um sistema de controle de disparo das notas musicais, sensibilidade dinâmica, ajuste de oitavas e um visor para *feedback* ao usuário

• Testar e validar o projeto tanto no âmbito da engenharia eletrônica como no âmbito musical

Este trabalho escrito foi dividido em três etapas, sendo que na primeira delas foi realizada uma revisão teórica sobre o funcionamento do Teremim, um estudo sobre o protocolo MIDI e os *softwares* VSTi e também uma análise das características do microcontrolador Tiva C. A segunda etapa descreve todo o desenvolvimento do projeto, partindo da aquisição do sinal do Teremim no microcontrolador até o desenvolvimento do *software* que realiza o processamento deste sinal e envia mensagens MIDI. A terceira etapa comenta os resultados obtidos após a utilização da interface e seu potencial de contribuição para o meio musical.

2 REVISÃO TEÓRICA

2.1 TEREMIM

O Teremim foi inventado pelo professor russo Léon Teremim no início do século XX, a partir de pesquisas em sensores de proximidade sensíveis a campo elétrico, sendo então um dos primeiros instrumentos musicais eletrônicos inventados, e o primeiro a funcionar sem que o usuário tenha que encostar no instrumento. O instrumento possui duas antenas que são acopladas a osciladores, sendo que de acordo com a posição de uma das mãos do músico em relação à antena vertical, é controlada a frequência do sinal de saída, e conforme a posição da outra mão em relação à antena horizontal é possível controlar a amplitude do sinal.**[2]**

Figura 1: Moog Etherwave Theremin Standard
Fonte: https://www.bhphotovideo.com/ Acesso em 28/03/2017

2.1.1 PRINCÍPIO DE FUNCIONAMENTO

Um teremim funciona a partir de dois osciladores com um circuito LC em ressonância. O primeiro oscilador opera em uma frequência fixa, tipicamente entre 280 kHz e 285 kHz. **[3]**

O cálculo da frequência de ressonância é obtido em função da capacitância e indutância do circuito conforme ilustra a figura 2.

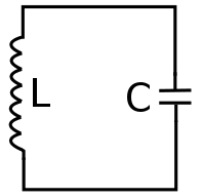


Figura 2: Circuito LC
Fonte: O Autor

A frequência de ressonância é dada pela equação 1, onde L representa a indutância e C a capacitância.

$$2\pi f = \omega = \frac{1}{\sqrt{LC}} \quad \textbf{(1)}$$

O segundo oscilador é controlado por uma antena monopolo omnidirecional e pela mão do usuário do Teremim. Quanto mais próxima a mão do usuário, maior é a capacitância equivalente “vista” pela antena. A variação provocada pela mão na capacitância total do segundo oscilador é da ordem de alguns picofarads. **[3]**

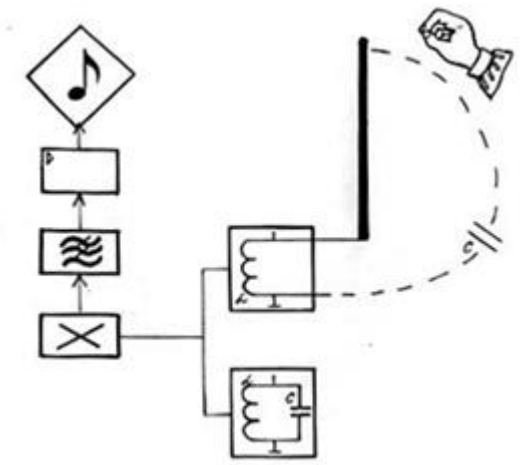


Figura 3: Osciladores do Teremim
Fonte: http://www.lydiakavina.com/therem.html Acesso em 15/04/2017

Para uma primeira análise, aproximando o comportamento a um capacitor de placas paralelas, tem-se a relação entre área e distância como forma de capacitância vista pela antena do Teremim. Em seguida os sinais dos dois osciladores são enviados a um *mixer* de frequências que multiplica os dois sinais. Usando uma identidade trigonométrica, o resultado da multiplicação de dois sinais senoidais de frequências f_1 e f_2 podem ser calculados através da equação 2.

$$\sin(2\pi f_1 t)\sin(2\pi f_2 t) = \frac{1}{2}\cos[2\pi(f_1 - f_2)t] - \frac{1}{2}\cos[2\pi(f_1 + f_2)t] \qquad \textbf{(2)}$$

O que resulta na soma e na diferença dos sinais ($f_1 + f_2$, $f_1 - f_2$). O sinal resultante da diferença fica na ordem de 0-3 kHz. Em seguida se aplica um filtro passa-baixas para eliminar a soma de frequências, e então o sinal resultante é amplificado. As Figuras 4 ilustra o comportamento do circuito com o aumento da capacitância. **[4]**

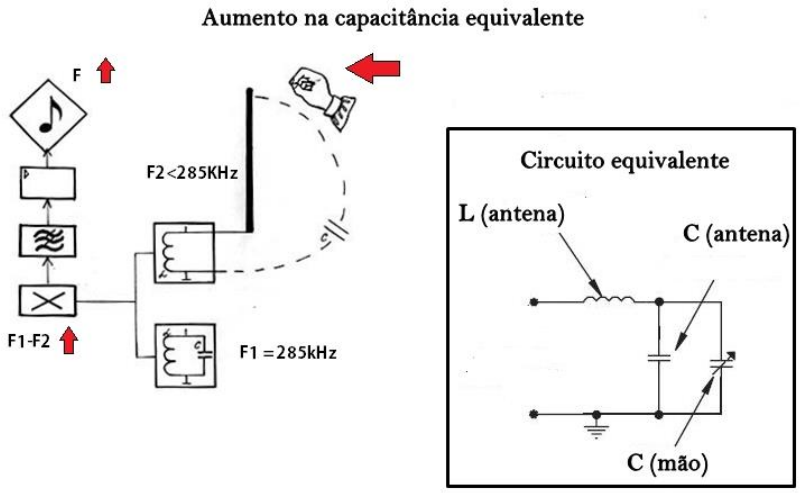


Figura 4: Aumento da capacitância
Fonte: http://www.lydiakavina.com/therem.html Acesso em 15/04/2017 (modificada pelo Autor)

2.2 MICROCONTROLADOR – TIVA C

Para implementar uma interface de comunicação MIDI, suprir as necessidades técnicas de baixa latência e atingir os objetivos, foi utilizado neste projeto o microcontrolador Tiva C TM4C123G da Texas Instruments. Segundo o fabricante ARM, o processador incluso no Tiva C, o ARM® Cortex®-M4 é o processador de 32 bits mais utilizado pela indústria pois foi desenvolvido especialmente para oferecer para as empresas a possibilidade de se obter alta performance, aliado ao baixo custo. Algumas vantagens segundo a ARM em relação ao ARM® Cortex®-M4 estão listadas abaixo: **[5]**

•O ARM® Cortex®-M4 opera com clock de até 80MHz e conta com a arquitetura ARMv7-M que foi desenvolvida para sistemas embarcados eficientes.

•Nested Vectored Interrupt Controller (NVIC) – o ARM® Cortex®-M4 possui um controle de interrupção integrado, flexível e eficiente, com uma latência bem baixa. As funções de tratamento de interrupções (Handlers) podem ser escritas em C.

•Possui várias escolhas de protocolos de comunicação de debug - JTAG e Serial Wire Debug.

•Cortex-M Software Interface Standard (Cortex-M Interface Padrão de Software) – CMSIS: Facilita o reaproveitamento e portabilidade de softwares.

Quanto à escolha do microcontrolador Tiva C TM4C123G, pode-se dizer que seu custo benefício é excepcional se comparado a outros microcontroladores de preço similar levando em conta suas principais características: **[6]**

- Clock de 80MHz 32-bit
- 256KB de memória Flash
- 32KB de memória SRAM
- 2KB de memória EEPROM
- Conectividade USB 2.0
- Dois conversores analógico/digital 12-bit 2MSPS
- 8 UART, 6 I2C, 4 SPI
- 40 pinos de expansão disponíveis em conector headers com passo de 2.54 mm
- Circuito de depuração integrado ICDI
- 2 módulos PWM, totalizando 16 saídas PWM
- 7 timers sendo 3 timers de 16 bits – Cada um com até 4 IC/OC/PWM

O launchpad utiliza como regulador de tensão o CI TPS73633DRB, que estabiliza a tensão de entrada em 3.3V para alimentação do circuito. A alimentação da placa é feita através de um conector micro-usb com tensões máximas e mínimas de 5.25V e 4.75V respectivamente. **[7]**

A Figura 5 ilustra o digrama de blocos do microcontrolador.

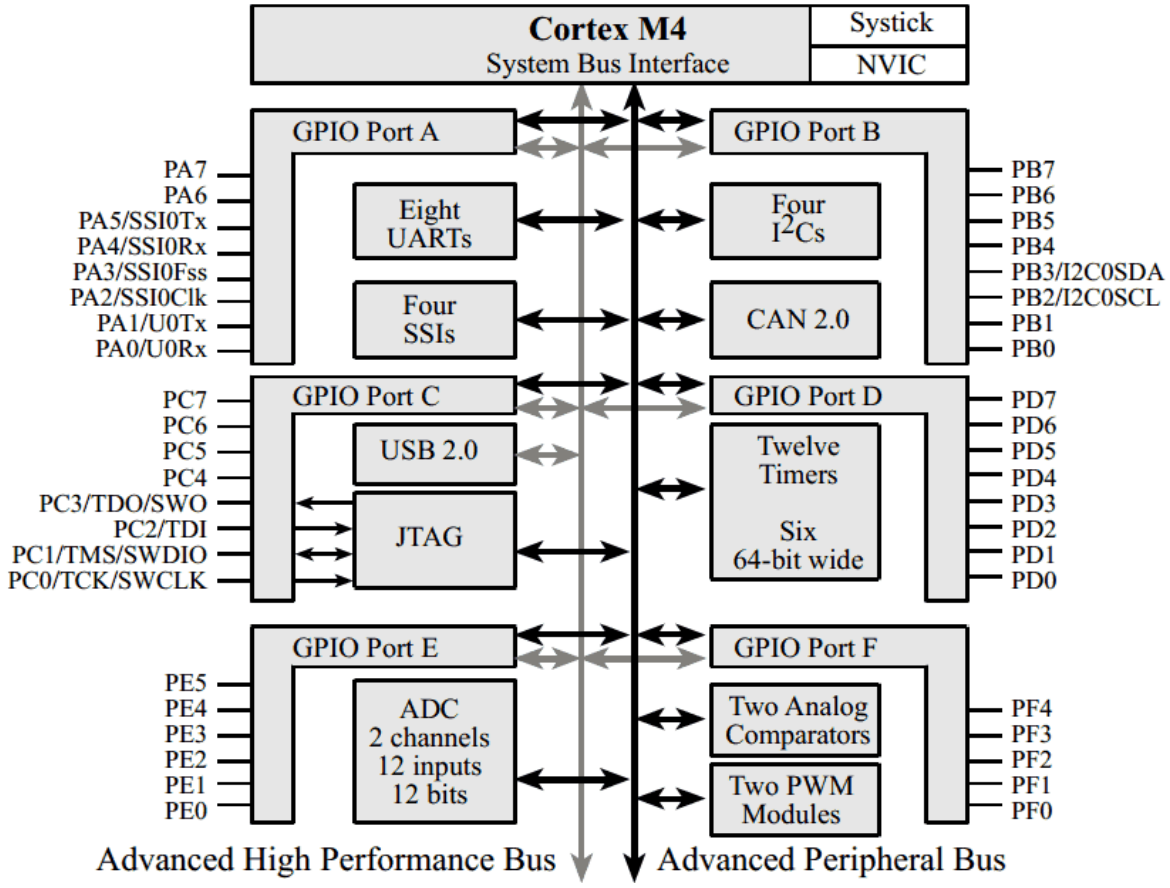


Figura 5: Diagrama de Blocos Cortex M4
Fonte: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm

É possível observar que o microcontrolador escolhido atende e supera os requerimentos para o desenvolvimento do projeto. Operar com baixa latência é um ponto crítico e o número de portas disponíveis também foi considerado devido a possíveis implementações de novos periféricos ou outros sensores futuramente no projeto. Esse microcontrolador foi escolhido também em função de um conhecimento prévio dos autores sobre o seu funcionamento e devido ao seu relativo baixo custo (USD 12.99 na loja da Texas Instruments em 05/03/2017).

2.3 PROTOCOLO MIDI

O protocolo MIDI foi desenvolvido no início dos anos 80 com o intuito realizar a comunicação entre instrumentos musicais digitais, carregando informações de qual nota musical foi tocada, com qual intensidade a nota foi tocada e também informações de variação de *pitch*. Todas essas informações são transportadas de forma serial em um *baud rate* que foi padronizado em 31250 *bauds* por segundo. A comunicação MIDI funciona por pulsos de corrente elétrica, onde um pulso de 5 mA

representa o nível lógico 0 e uma ausência de corrente (0 mA) representa o nível lógico 1. O fato da porta trabalhar com pulsos de corrente e não tensão, como o usual, deve-se ao fato de os dispositivos serem opto-acoplados, ou seja, não há nenhum contato elétrico entre os dispositivos. Então em um pino receptor de um dispositivo MIDI há um circuito integrado onde dentro dele há um LED (*Light Emitting Diode*) que ao receber um pulso de corrente polariza um fototransistor. **[8]**

Foi padronizado que cada nota musical é associada a um número MIDI de 21 a 108, abrangendo todas as 88 notas musicais da escala de A0 até C8 (ou então Lá 0 até Dó 8 na nomenclatura em português brasileiro) e também foram definidos 128 níveis de intensidade com que cada nota é tocada, esse parâmetro é chamado de *velocity*. Outro dado importante é que cada interface MIDI comporta 16 canais de informação independentes entre si, tornando possível cada interface estar conectada a até 16 controladores (dispositivos de interface humana) independentes. No **Anexo 1** é possível visualizar a tabela que mostra as 88 notas musicais associadas ao seu respectivo número MIDI e também à sua frequência de onda sonora fundamental. **[9]**

Os comandos MIDI básicos são comandos que disparam uma nota musical em um determinado canal com um determinado *velocity* e na sequência é enviado um comando que desliga a nota musical previamente tocada, por exemplo: a nota C4 é tocada em um teclado controlador MIDI configurado no canal 1 e o músico segura a tecla por dois segundos e depois solta, a informação transmitida no MIDI é: a tecla C4 foi acionada no canal 1 com um certo valor de *velocity*, e depois de dois segundos será enviada a informação de que a tecla C4 no canal 1 foi solta. Esses comandos são chamados de *Note On* e *Note Off*. A mensagem *Note On* é representada pelo *byte* hexadecimal 0x9* onde * é o canal onde a nota é tocada, podendo variar de 0 a F (16 canais), o comando *Note On* deve ser seguido de mais dois *bytes*, que carregam a informação de qual nota foi tocada e com qual *velocity*. A mensagem *Note Off* é representada pelo *byte 0x8** e também deve ser seguida de dois *bytes* um deles com a informação de qual nota musical deve ser desligada e um *velocity* definido como 0 por convenção. **[10]**

Um exemplo completo foi criado pelo engenheiro de *software* Dominique Vandenneucker disponível em seu site **[11]** (simplificado pelo Autor), onde a

sequencia musical ilustrada na Figura 6 é tocada com um *velocity* de 64 (equivalente a 0x40 em hexadecimal).

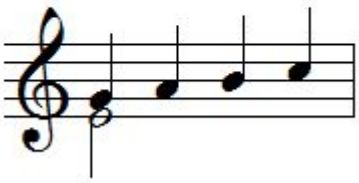


Figura 6: Exemplo de sequência musical
Fonte: http://www.music-software-development.com/midi-tutorial.html (acesso em 23/04/2017, simplificada pelo Autor)

Então a série de comandos enviada é:

- t=0 : **0x90** - **0x40** - **0x40** (Início da nota musical E3, número MIDI = 64)
- t=0 : **0x90** - **0x43** - **0x40** (Início da nota musical G3, número MIDI = 67)
- t=1 : **0x80** - **0x43** - **0x00** (Fim da nota musical G3, número MIDI = 67)
- t=1 : **0x90** - **0x45** - **0x40** (Início da nota musical A3, número MIDI = 69)
- t=2 : **0x80** - **0x45** - **0x00** (Fim da nota musical A3, número MIDI = 69)
- t=2 : **0x80** - **0x40** - **0x00** (Fim da nota musical E3, número MIDI = 64)
- t=2 : **0x90** - **0x47** - **0x40** (Início da nota musical B3, número MIDI = 71)
- t=3 : **0x80** - **0x47** - **0x00** (Fim da nota musical B3, número MIDI = 71)
- t=3 : **0x90** - **0x48** - **0x40** (Início da nota musical C4, número MIDI = 72)
- t=4 : **0x80** - **0x48** - **0x00** (Fim da nota musical C4, número MIDI = 72)

Ao longo dos anos o protocolo MIDI se tornou uma ferramenta muito útil e presente em diversos estúdios de gravação ao redor do mundo, pois através deste protocolo que há a comunicação entre controladores e instrumentos musicais virtuais (VSTi) carregando não só informações musicais, mas como também

informações de controle, como por exemplo *knobs* e *faders* que podem controlar efeitos e volume em uma mesa de som virtual.

2.4 *VIRTUAL STUDIO TECHNOLOGY INSTRUMENT* (VSTi)

Com o amadurecimento do processo de gravação de áudio digital, os racks de efeitos passaram do mundo físico para o virtual. *Virtual Studio Technology* é uma interface desenvolvida pela Steinberg e lançada em 1996 que integra sintetizadores e efeitos de áudio com editores e dispositivos de gravação de som digitais. O VST utiliza processamento digital de sinais para simular o hardware físico de um estúdio de gravação com um software. Existem inúmeros plug-ins desenvolvidos sobre a plataforma VST que são suportados pela grande maioria das aplicações de produção e edição de áudio. A sigla VSTi é utilizada para diferenciar um *software* de efeito de um *plugin* de instrumento virtual. **[12]**

Os *softwares* VSTi possuem a capacidade de receber dados MIDI, podendo atuar como sintetizadores de áudio ou *samplers*. Com essa tecnologia é possível gravar uma pista pela interface MIDI e posteriormente passar para áudio com uma simulação de um instrumento. Existem *plugins* VSTi que simulam um determinado tipo de instrumento como pianos, saxofone, metais, entre outros.

Hoje é possível conectar um controlador MIDI ao computador e utilizar um *software* que simula timbres de instrumentos do alto custo com qualidade profissional, ou mesmo conectar uma guitarra ao computador e simular um *rack* de efeitos. Para isso, é necessário que o usuário utilize um programa hospedeiro, também conhecido como DAW (*Digital Audio Workstation)* que suporte *plugins VST* em tempo real. Uma interface de áudio possui como entrada sinais analógicos de áudio e sinais digitais MIDI, comunicando-se com um sistema operacional que realiza o processamento dos sinais de entrada e em seguida fornecerem sinais analógicos de saída que são enviados a alto falantes ou fones de ouvido. A Figura 7 ilustra um diagrama que mostra as conexões entre um sistema de gravação e reprodução Áudio/MIDI.

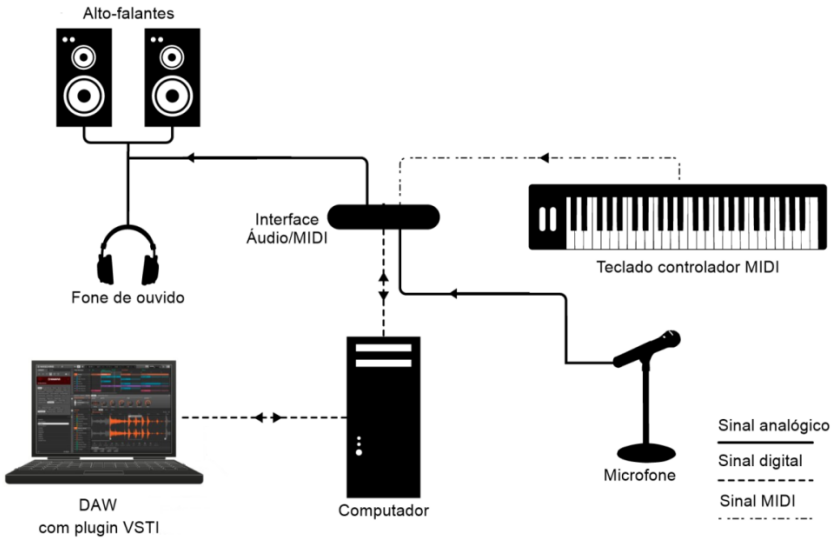


Figura 7: Diagrama Áudio/MIDI
Fonte: O Autor

3 PROCEDIMENTO METODOLÓGICO

O presente projeto consiste no desenvolvimento de um protótipo funcional de uma interface de conversão de áudio para MIDI, para isso foi realizada a revisão teórica do projeto o desenvolvimento do protótipo e os testes finais.

A revisão teórica feita no capítulo 2 foi realizada de forma a compreender o funcionamento do Teremim e verificar os requisitos impostos pelos equipamentos e softwares que trabalham com a comunicação MIDI, foram consultados diversos sites e artigos relacionados ao protocolo e suas informações técnicas. Para projetar o circuito da interface foram consultados alguns livros de eletrônica (como por exemplo o Microelectronics **[13]**), os respectivos *datasheets* e as especificações dos componentes utilizados no circuito.

O desenvolvimento da interface envolve a confecção de todo o *hardware* e *software*, descrito no capítulo 4 e ilustrado em várias etapas o processo para implementação do protótipo final. Para isso foram detalhadas as seguintes etapas: Análise do sinal analógico do Teremim, diagramação do circuito amplificador, projeto do controle de disparo de notas e variação dinâmica, implementação do visor para *feedback*, desenvolvimento do *software* e projeto e confecção da placa de circuito impresso.

Por fim, foram realizados testes para verificar a qualidade de leitura de frequência, funcionamento do display, botão de disparo, sensor de força FSR, frequência de saída do Teremim em função da distância e comunicação MIDI com os *softwares*. Em seguida foram feitos os testes práticos finais afim de validar o funcionamento da interface quando utilizada junto aos softwares VSTi instalados no computador.

3.1 RECURSOS UTILIZADOS

Para o desenvolvimento de todo o projeto desde a parte de análise até o desenvolvimento do produto final foram necessários diversos componentes para a confecção do circuito, assim como equipamentos para a realização de medidas de parâmetros eletrônicos. Então foram montadas três listas, uma de componentes utilizados diretamente na confecção do projeto, outra de equipamentos necessários ao longo de todo desenvolvimento e uma terceira lista com os softwares utilizados.

3.1.1 LISTA DE MATERIAIS

- Teremim RDS Zep
- 1 Microcontrolador Texas Instruments Tiva C – TM4C123G
- Resistores diversos
- Capacitores diversos
- 1 Transistor BC548
- 2 *switchs* de duas posições
- 1 *jack* P10
- 1 Porta USB
- 1 FSR (*Force Sensitive Resistor*)
- 1 Conector DIN 5 Fêmea
- 1 Display Neo Pixel WS2812
- 1 Placa de circuito impresso dupla face
- 1 Caixa plástica
- 1 Adaptador AC/USB 1 A

3.1.2 LISTA DE EQUIPAMENTOS

- Computador
- Interface de áudio com conectividade MIDI
- Osciloscópio digital Agilent DSO2000
- Multímetro digital
- Caixas multimídia

3.1.3 LISTA DE SOFTWARES

- Energia IDE
- MathWorks Matlab
- ISIS Proteus
- Autodesk Eagle - PCB Design & Schematic Software

4 DESENVOLVIMENTO DA INTERFACE

Para o desenvolvimento da interface, foi utilizado como base o Teremim RDS Zep, pois trata-se de um Teremim de fácil acesso, pois ele é relativamente barato e fabricado no Brasil. Esse modelo do instrumento apresenta apenas uma antena (enquanto o Teremim original apresenta duas), sendo que com essa antena o músico é capaz de controlar apenas a frequência gerada pelo instrumento. A amplitude da onda gerada pode ser controlada por um *knob* de volume, mas para todos os cálculos do projeto, foi utilizada a configuração de volume no máximo. Há também um outro *knob* de controle de sensibilidade da antena, no escopo do projeto esse *knob* foi mantido sempre na posição de 105 graus.

Figura 8: Teremim RDS Zep
Fonte: O Autor

Durante o processo de programação do microcontrolador, foi utilizada a IDE (*Integrated Development Environment)* Energia, pois ela é integrável com diversas bibliotecas que foram capazes de auxiliar o desenvolvimento do projeto.

4.1 AQUISIÇÃO DO SINAL DO TEREMIM

4.1.1 ANÁLISE DO SINAL GERADO PELO TEREMIM

Para realizar a aquisição do sinal do Teremim RDS Zep no microcontrolador, foi necessário primeiramente realizar um estudo sobre o sinal gerado por este instumento, para encontrar diversas características tais como forma de onda e amplitude. A Figura 9 mostra a forma de onda de saída do Teremim obtida por um osciloscópio digital Agilent DSO2000.

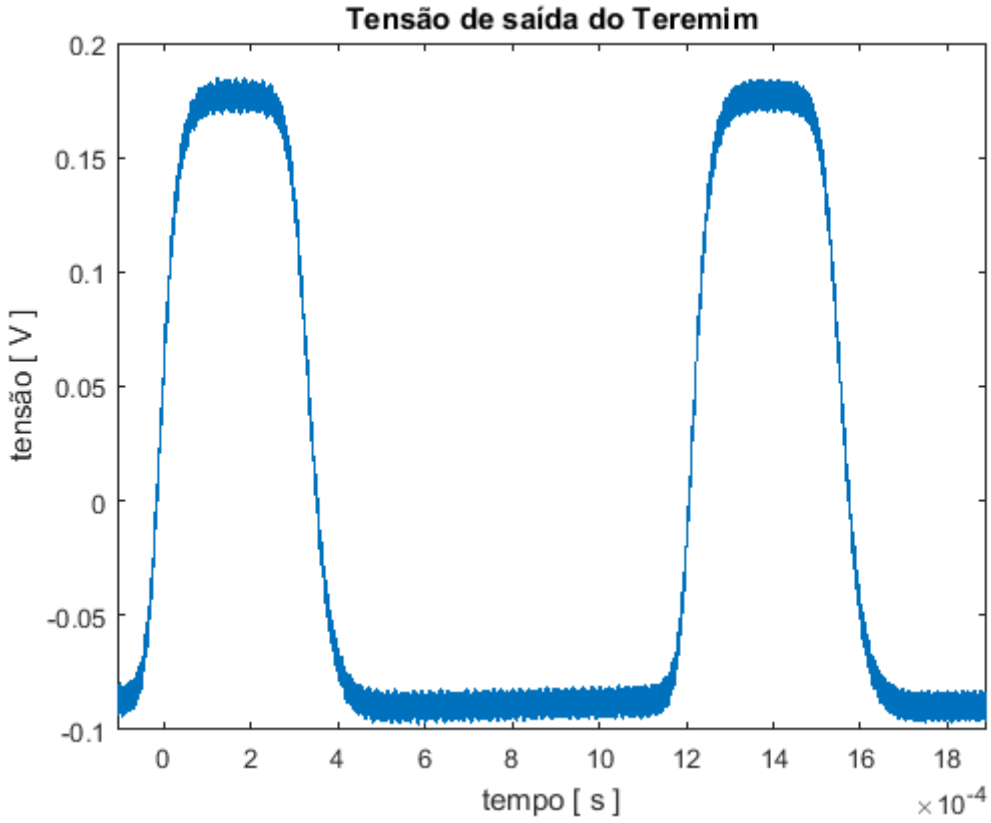


Figura 9: Tensão de saída do Teremim
Fonte: O Autor

Conforme pôde ser observado na Figura 9, o sinal de saída do Teremim apresenta um sinal periódico cuja tensão varia de aproximadamente -90 mV até 180 mV, com frequência de 833 Hz e forma de onda aproximadamente retangular. É importante destacar que essa medida foi feita com o *knob* de volume do Teremim em sua posição de volume máximo, porém para outras configurações do *knob* de volume foi constatada uma forma de onda muito parecida com a apresentada no gráfico, bem como para medidas em outras frequências.

Foi feita também uma análise espectral do sinal de saída do Teremim através de um computador com uma interface de áudio com o *plugin* VST Fab Filter Pro Q^2. Para essa análise a mão do músico foi posicionada de maneira que a componente fundamental do sinal de saída estivesse em torno de 100 Hz. A Figura 10 mostra os resultados dessa análise. Também está disponível no **Anexo 2** imagens com melhor resolução.

Figura 10: Sinal do Teremim no domínio da frequência
Fonte: O Autor

A partir do gráfico mostrado na Figura 10 foi possível constatar que o sinal possui diversas componentes harmônicas tanto pares quanto ímpares, o que corrobora com a forma de onda no domínio do tempo mostrada na Figura 9.

Após a análise do sinal de saída do Teremim foi definida então qual seria a melhor forma do microcontrolador fazer a aquisição do sinal. Primeiramente foi pensado em condicionar o sinal para a faixa dinâmica de um dos conversores analógico digital do microcontrolador (0 V – 3.3 V), para então a partir do sinal digitalizado encontrar a frequência da onda. Porém esta metodologia foi descartada pois o microcontrolador teria que analisar uma janela de tempo do sinal periodicamente para em seguida realizar diversos cálculos para a determinação da frequência. Esta técnica exigiria um processamento considerável do microcontrolador, o que, apesar da alta velocidade de *clock* do *TM4C123G* (80 MHz)

poderia gerar uma latência não desejada no processo. Para o escopo do projeto a única informação necessária a ser capturada do Teremim é a frequência da onda gerada, sendo a sua amplitude uma informação desnecessária dado que o Teremim utilizado gera uma onda de amplitude constante. Optou-se assim por condicionar o sinal de forma que ele fique em uma faixa dinâmica próxima da faixa dinâmica de uma porta de entrada digital do microcontrolador, e assim a partir de uma interrupção por borda de subida seria possível calcular a frequência da onda e ainda economizando processamento que seria utilizado de forma desnecessária pelo microcontrolador.

5.1.2 CONDICIONAMENTO DO SINAL

Para condicionar o sinal para a faixa dinâmica de uma porta digital do microcontrolador utilizou-se um amplificador com tensão de saída limitada entre 0 e 3.3 V. Para isso foi projetado um amplificador utilizando um transistor de junção bipolar do tipo NPN na configuração de emissor comum. É importante lembrar que o sinal de saída de um amplificador na configuração emissor comum tem o seu sinal de saída com a fase invertida em relação ao sinal de entrada, entretanto isso não prejudica de nenhuma maneira o cálculo da frequência.

Para certificar de que o sinal de saída do amplificador estará na faixa dinâmica adequada, foi necessário consultar o *datasheet* do microcontrolador e verificar os parâmetros de V_{IH} (*Voltage input high*) mínimo e V_{IL} (*Voltage input low*) máximo, conforme mostra a Tabela 1.

Tabela 1: Parâmetros Dinâmicos da entrada digital do TM4C123G

Parâmetros da porta digital (V_{DD} = 3.3 V)	
V_{IH} (mín)	2.145 V
V_{IL} (máx)	1.115 v

Fonte: TI TM4C123G – Datasheet (pág. 1360) **[7]**

O transistor escolhido foi o BC548, conforme mostra o esquemático ilustrado na Figura 11.

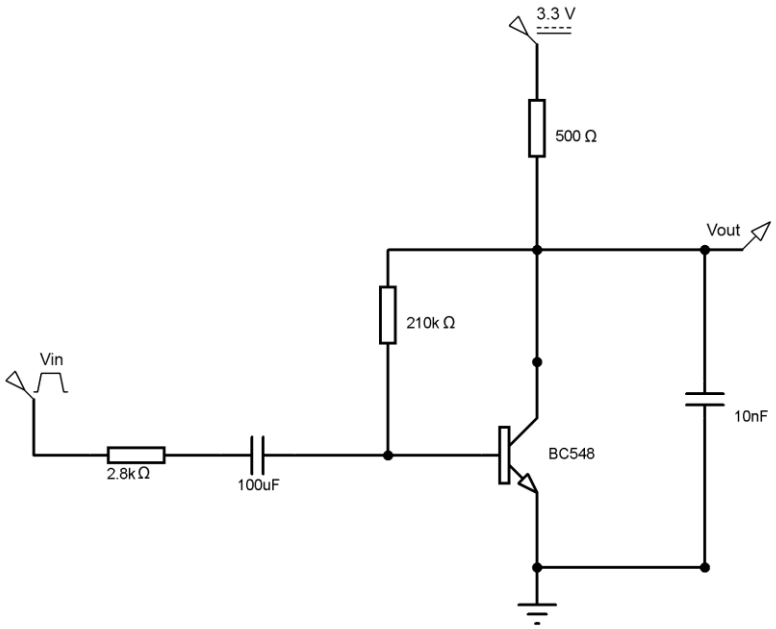


Figura 11: Circuito amplificador
Fonte: O Autor

Foi feita então uma simulação do amplificador utilizando o *software* ISIS Proteus considerando como sinal de entrada um sinal similar ao do Teremim na frequência de 833 Hz.

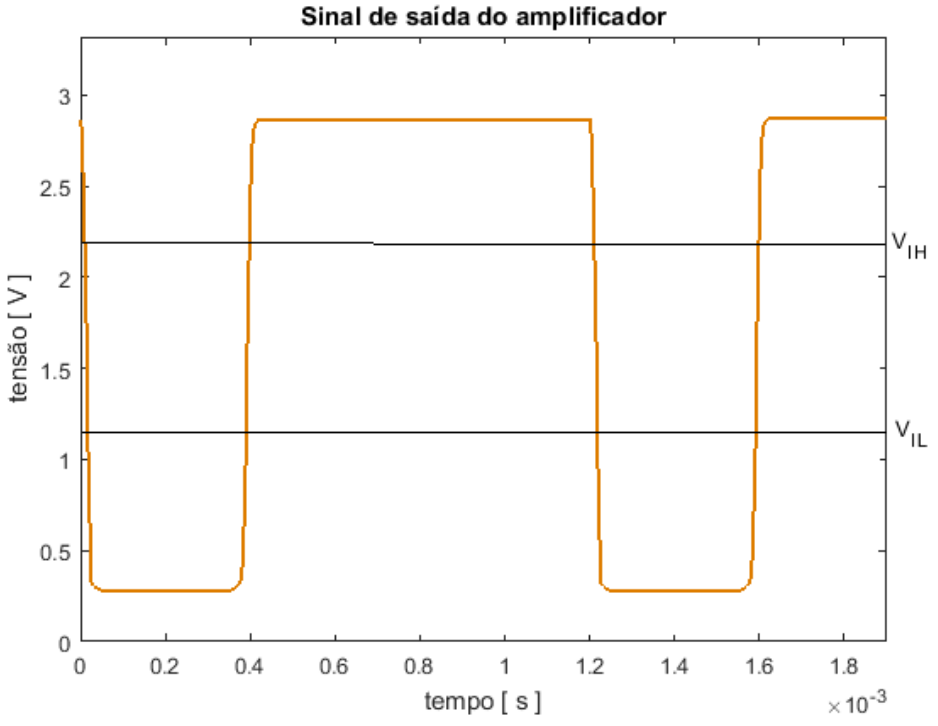


Figura 12: Simulação do sinal de saída do amplificador
Fonte: O Autor

Com o gráfico mostrado na Figura 12, pôde se observar que o sinal amplificado atende aos requisitos necessários para ser realizada a medida da frequência através de uma porta de entrada digital do microcontrolador.

4.1.3 ALGORITIMO DE AQUISIÇÃO

Uma vez condicionado o sinal, foi desenvolvido um algoritimo para a aquisição do mesmo. Esse algoritimo consiste monitorar o pino de entrada, e toda vez em que ocorre uma borda de subida do sinal, uma interrupção é acionada no código, onde através do contador **micros()** (contador que armazena em uma variável do tipo *unsigned long* há quantos microssegundos o microcontrolador foi inicializado) é calculado o intervalo de tempo entre duas bordas de subida, tendo como resultado o período instantâneo do sinal.

A função acionada pela interrupção funciona de forma que assim que acontece uma borda de subida, é definido t_1 como sendo igual a “micros() – t_0”, sendo que no final da interrupção t_0 é atualizado com o valor dado por **micros()**, para então quando houver a próxima borda de subida ser calculado o período anterior e armazenado em t_1. Com a informação do período em microsegundos basta multiplicá-lo por 10^6 para se obter o período em segundos, para então encontrar a frequência em hertz através do inverso do período.

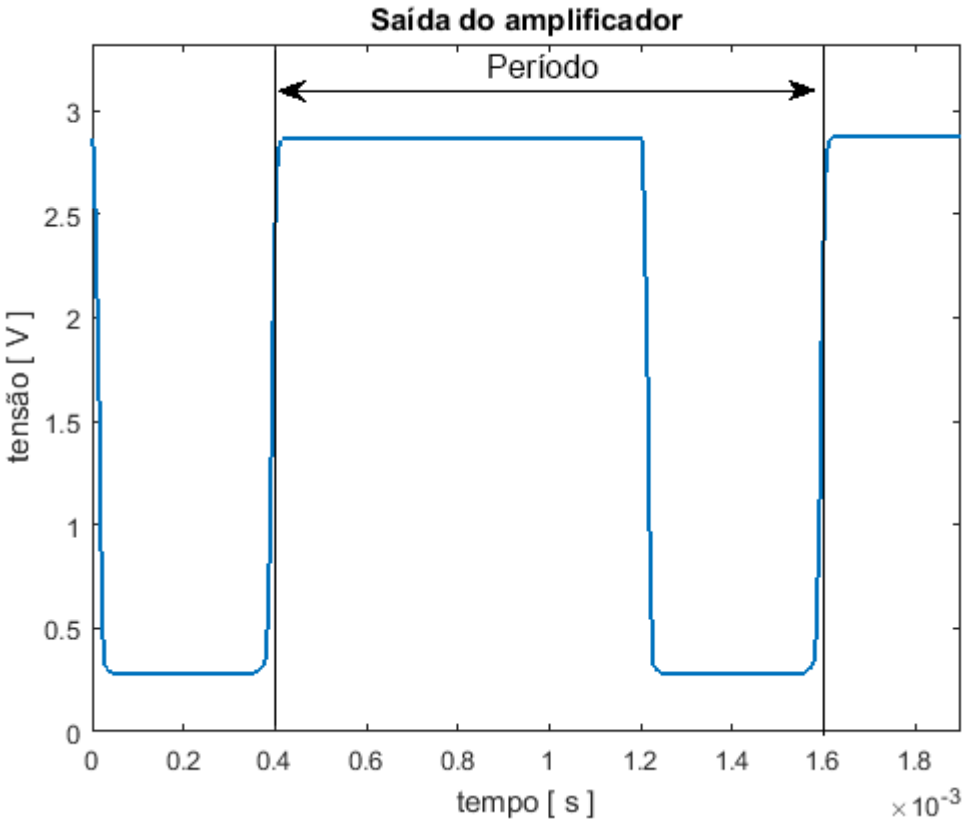


Figura 13: Saída do amplificador
Fonte: O Autor

Foram realizados testes para validar o sistema de cálculo da frequência através de um gerador de funções, fazendo em torno de 1200 medidas com o gerador em uma frequência fixa de 600 Hz. Foi então constatada a ocorrência de algumas medidas errôneas, de modo aleatório, que poderiam comprometer a qualidade do projeto como um todo. A média dessas medidas ficou em 597.66 Hz e o desvio padrão 63.36 Hz, resultando em uma SNR (*signal-to-noise ratio*) de 9.42. A Figura 14 mostra um conjunto de aproximadamente 1200 amostras realizadas, onde pode-se constatar a presença de algumas medidas errôneas, mostradas em pontos vermelhos.

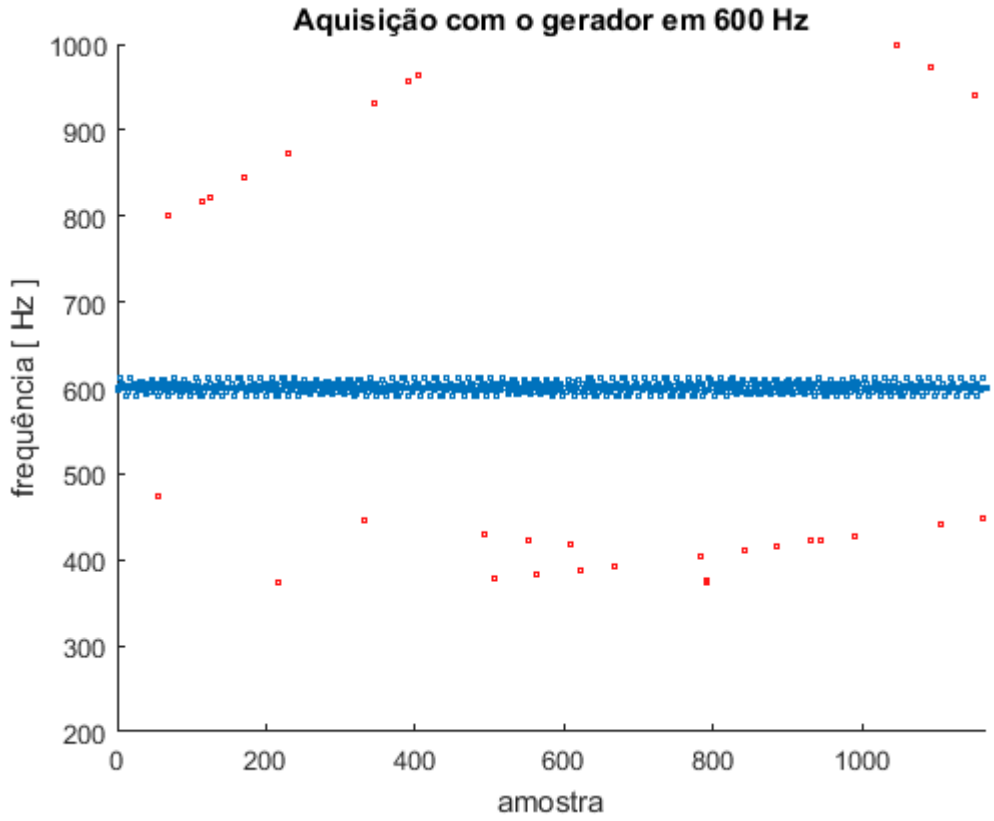


Figura 14: Aquisição com o gerador em 600 Hz
Fonte: O Autor

Como uma SNR tão baixa poderia comprometer severamente a funcionalidade do projeto, foram consideradas algumas hipóteses que poderiam estar gerando esse erro aleatório, sendo a primeira delas: uma instabilidade no gerador de funções, que foi descartada ao realizar o ensaio com um outro gerador de funções de outro fabricante. A segunda hipótese seria o contador **micros()** que poderia estar sofrendo um *overflow* gerando essa variação nas medidas, porém esse contador utiliza uma variável do tipo *unsigned long*, o que faria o estouro acontecer apenas uma vez a cada 2^{32} µs (71 minutos e 35 segundos). A terceira hipótese seria alguma falha na programação do algoritimo de aquisição, entretanto foram feitos testes com códigos diferentes, e o erro persistiu. A quarta hipótese seria que a plataforma Energia estaria acionando algumas interrupções de alta prioridade não escritas no código, que poderiam estar fazendo com que a interrupção de borda de subida fosse ignorada, e então as variáveis t_0 e t_1 estariam sendo atualizadas fora de sincronismo. Esta quarta hipótese é a mais provável, no entanto, por falta de tempo hábil, não foi possível verificá-la, pois seria necessário o desenvolvimento de outro código em uma nova plataforma de desenvolvimento.

Dada a impossibilidade de encontrar a causa e solucionar esse erro, foram estudadas formas de fazê-lo não interferir na qualidade do projeto. Foi considerada a utilização de médias das medidas, mas como quando o erro ocorre a dispersão é muito grande, seria necessário um número muito grande de medidas por média, que acabaria resultando em um atraso no cálculo da frequência. Como foi observado que apesar de o erro acontecer de forma aleatória, em 1200 medidas ele ocorreu em torno de 50 vezes sendo que somente uma vez aconteceram dois erros em amostras consecutivas, optou-se pela utilização de um algoritmo de eliminação dos dados errôneos através do cálculo da mediana.

Foi definido que para cada 5 leituras de frequência seria feita uma mediana e armazenada em uma variável acessível pelo *loop* principal do código toda vez que necessário. O número 5 foi escolhido porque garante uma boa margem de segurança para o sistema, dado que é bastante improvável que ocorram duas medidas erradas consecutivamente, uma mediana de 3 amostras poderia ser suficiente, mas foi definido 5 para aumentar a confiabilidade. Como a taxa de amostragem da leitura varia de acordo com a frequência foi constatado que para a menor frequência da faixa dinâmica do projeto (100 Hz, período de 10 ms) haverá um atraso de leitura de 50 ms, o que é musicalmente aceitável para notas de tão baixa frequência.

Como a biblioteca **math.h** da linguagem C não possui nativamente uma função que seja capaz de calcular a mediana de um vetor, foi necessário implementar uma função que armazene em um *array* 5 leituras de frequência em ordem crescente, e após isso armazene o terceiro valor em uma variável global de frequência. Após implementar esse sistema, o erro de leitura foi solucionado. Esse processo matemático foi aplicado para os mesmos valores mostrados na Figura 14, e então foi observada uma média de 600.74 Hz, um desvio padrão de 3.26 Hz e uma SNR de 183.91.

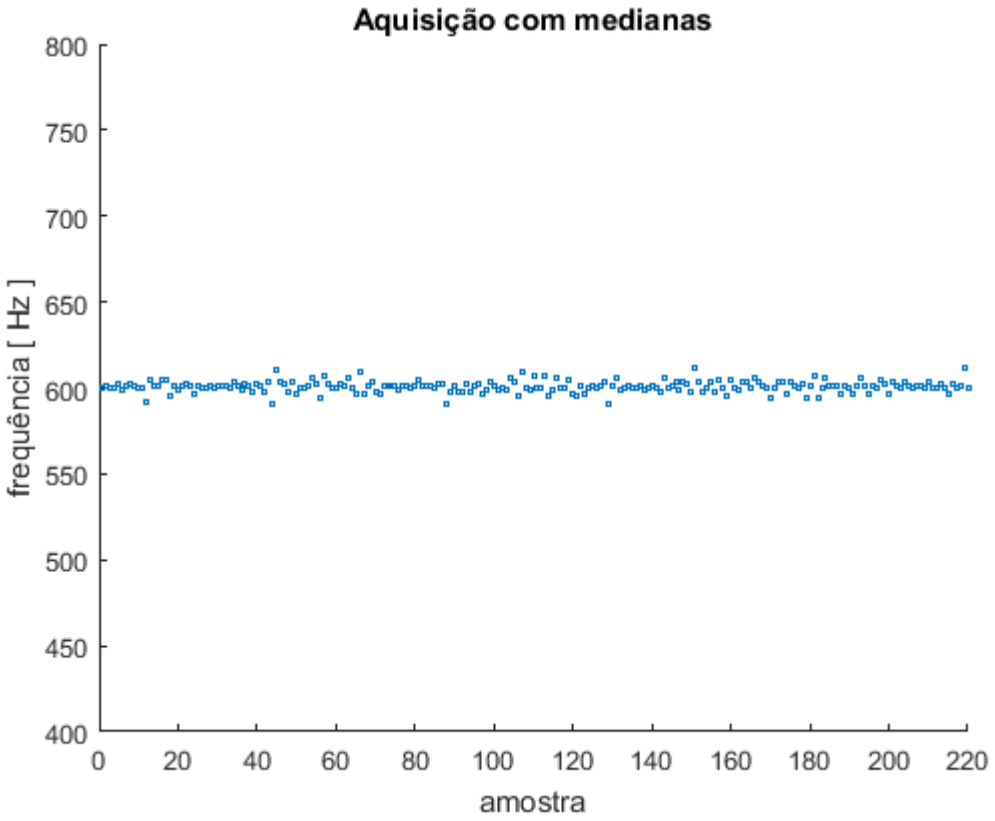


Figura 15: Aquisição com o filtro de medianas
Fonte: O Autor

4.2 IMPLEMENTAÇÃO MIDI

Após implementada a rotina que calcula a frequência do sinal gerado pelo Teremim, foi implementada uma função que relaciona essa frequência com o seu número MIDI correspondente, de acordo com a equação 3, descrita pelo professor Joe Wolf da universidade australiana University New South Wales **[9]**.

$$n_{MIDI} = 69 + 12\ log_2\left(\frac{f}{440}\right) \qquad \textbf{(3)}$$

Como o Teremim gera frequências contínuas no espectro e os intervalos musicais são discretizados, foi necessário utilizar a função **round()** para arredondar o número calculado para a nota musical mais próxima, discretizando a escala do Teremim. Porém foi notado que utilizando este método, a tocabilidade do instrumento continuaria sendo bastante dificultosa, ofendendo a ideia inicial do projeto, que é de facilitar o uso do instrumento.

A dificuldade de execução citada no parágrafo anterior se deve à não linearidade entre a distância da mão do músico até a antena do Teremim, em relação à frequência gerada pelo instrumento, tornando o espaçamento entre as notas cada vez mais estreito conforme é diminuída a distância entre a mão e a antena. Para ilustrar este aspecto, foi feito um ensaio medindo-se a frequência gerada pelo instrumento em função da distância radial entre a mão do músico e a antena, na faixa de 5 a 75 cm. A Figura 16 exibe a relação encontrada.

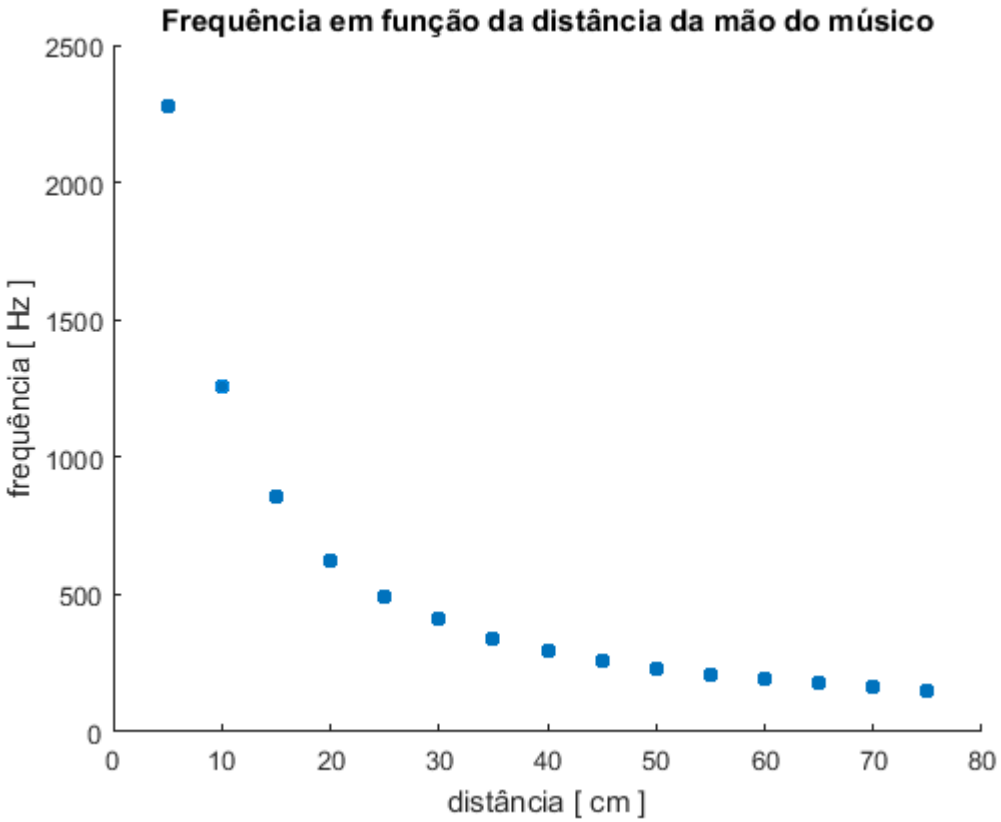


Figura 16: Frequência em função da distância
Fonte: O Autor

4.2.1 LINEARIZAÇÃO DA ESCALA

Para facilitar a execução musical do instrumento, optou-se por utilizar um espaçamento fixo entre as notas, de forma que foi necessário encontrar uma equação que descrevesse a distância radial da mão do músico em função da frequência lida pelo microcontrolador. Então através da função *Curve Fitting* do Matlab, tendo como entrada os pontos exibidos na Figura 16, foi possível encontrar a equação 4.

$$d = 8811\, f^{-0.9483} \quad \textbf{(4)}$$

Onde **f** é a frequência lida em hertz e **d** é a distância radial da mão do músico em centímetros.

Desta forma tornou-se possível gerar o número MIDI a partir da distância calculada. Para isso foram definidos dois modos de escala com a distância variando de 65 a 15 cm, conforme mostra a Tabela 2.

Tabela 2: Modos de oitava

Modo	$n_{inicial}$	n_{final}	Espaçamento
Uma oitava	48 (C3)	60 (C4)	4.16 cm
Duas oitavas	48 (C3)	72 (C5)	2.08 cm

A partir de dois pares ordenados (65, 48) e (15, 60) foi possível obter a equação para o modo de uma oitava e de maineira similar para o modo de duas oitavas, respectivamente representados pelas equações 5 e 6.

$$n_{MIDI\,1} = -0.24\, d + 63.6 \quad \textbf{(5)}$$

$$n_{MIDI\,2} = -0.48\, d + 79.2 \quad \textbf{(6)}$$

Após a implementação da linearização da escala, foi possível notar uma melhora significativa na tocabilidade do instrumento.

4.2.2 CHAVE DE CONFIGURAÇÃO DE OITAVAS

Para o músico poder escolher se quer trabalhar no modo de uma oitava ou modo de duas oitavas, foi instalada uma chave de duas posições similar à mostrada na Figura 17. A chave foi instalada em uma porta de entrada digital do microcontrolador, onde foi habilitado o resistor de *pull-up* interno do Tiva C, de forma com que quando a chave está posicionada de maneira que não é fornecida tensão na porta, ela fica em nível lógico alto. A outra posição da chave faz com que o pino

digital fique aterrado, ocasionando então uma leitura de nível lógico baixo. A cada execução do *loop* principal é feita uma verificação do estado do pino.

Figura 17: Chave de duas posições
Fonte: http://www.steren.com.mx/switch-miniatura-de-palanca-1-polo-2-tiros-2-posiciones-de-3-y-6-amperes.html, Acesso em 07/06/2017

4.2.3 ENVIO DO COMANDO MIDI NA UART

Para realizar o envio dos comandos MIDI, foi utilizada uma biblioteca disponível na plataforma colaborativa GitHub, chamada **midi.h**. Essa biblioteca torna possível o envio de comandos MIDI através da UART (*Universal Asynchronous Receiver/Transmitter*) do microcontrolador. Para a utilização da biblioteca foi necessário configurar o *baud rate* do microcontrolador em 31250 *bauds* por segundo (padrão do protocolo MIDI), e então tornou-se possível enviar as notas musicais através da função **MIDI.sendNoteOn()**, lembrando que sempre que uma nota é enviada, após a sua duração ela deve ser "desligada" através do comando **MIDI.sendNoteOff()**. Como o Teremim é um instrumento monofônico (notas musicais tocas individualmente) foi definido que sempre que uma nova nota musical for enviada, a nota tocada anteriormente será desligada. Caso as notas não fossem desligadas, haveria uma sobreposição de notas no instrumento virtual que recebe os comandos MIDI.

A função **MIDI.sendNoteOn()** requer três argumentos inteiros, sendo o primeiro deles o número MIDI correspondente da nota a ser tocada, o segundo deles o *velocity* da nota, e o terceiro deles o canal no qual a nota será enviada. Então um exemplo da nota C4 sendo tocada com um *velocity* de 100 no canal 1 por 4 segundos seria:

"MIDI.sendNoteOn(60,100,1);"

Após 4 segundos

"MIDI.sendNoteOff(60,0,1);"

Como para a interface projetada há a comunicação com apenas um instrumento musical, foi definido que todos os comandos MIDI serão enviados no canal 1.

Todos esses comandos são enviados no pino de Tx do microcontrolador e ligados diretamente no pino de saída do conector MIDI. Há também uma conexão do conector MIDI com o terra do circuito, e também uma conexão com a tensão de alimentação do Tiva C, de 3.3 V, passando por um resistor de 220 Ω. Essa alimentação tem como função polarizar o fototransistor que existe nos dispositivos MIDI.

4.3 MODOS DE OPERAÇÃO

Após o desenvolvimento do algoritimo de aquisição de frequência e envio das notas musicais em formato MIDI, foi definido que pelo bem da vesatilidade da interface, seriam necessários dois modos de operação, podendo ser selecionados pelo músico através de uma chave de duas posições com as mesmas configurações da chave descrita na seção 4.2.2.

4.3.1 MODO CONTÍNUO

O modo contínuo funciona de forma com que toda vez em que é lida uma frequência que de acordo com a seleção de oitavas seja associada a um número MIDI diferente do número MIDI enviado anteriormente, uma nova nota musical é enviada, tornando então a escala musical contínua, da mesma maneira que ocorre com o Teremim originalmente.

4.3.2 MODO BOTÃO DE DISPARO

Esse segundo modo funciona de forma com que o músico tem controle sobre quais notas musicais serão disparadas. Esse controle é feito através de um controle remoto com um botão, onde enquanto o músico controla a frequência de acordo com a distância de uma das mãos da antena, com a outra mão ele pode controlar o momento exato em que a nota será disparada. A Figura 18 mostra o protótipo do controle remoto.

Figura 18: Controle de disparo
Fonte: O Autor

4.4 *DISPLAY* NEOPIXEL

Para o músico ter a possibilidade de um retorno visual sobre qual nota está sendo tocada, foi inicialmente considerada a ideia de um *display* de 7 segmentos mostrar a nota musical associada à posição da mão do músico em relação ao Teremim. Contudo essa solução foi descartada pelo fato de músico ter de a todo momento estar olhando diretamente para o *display* e fazer a leitura da nota, o que poderia comprometer a sua performance musical. Tendo em vista este aspecto, foi escolhida a ideia de utilizar um *display* de LED RGB, onde é possível configurar os LEDs de forma a gerar 7 cores fáceis de serem distinguidas. O *display* escolhido para o projeto foi o NeoPixel WS2812, que possui 12 matrizes de LED RGB que podem ser controlados através de apenas um pino digital. Como existem 12 notas musicais, e o *display* é capaz de gerar apenas 7 cores de fácil distinção, foi definido que cada nota musical “cheia” há uma cor associada, onde todos os 12 LEDs acendem, e para as notas sustenidas apenas 6 LEDs acendem em forma de “meia lua”, facilitando para o músico ter uma referência visual sem ter de olhar diretamente para o *display*.

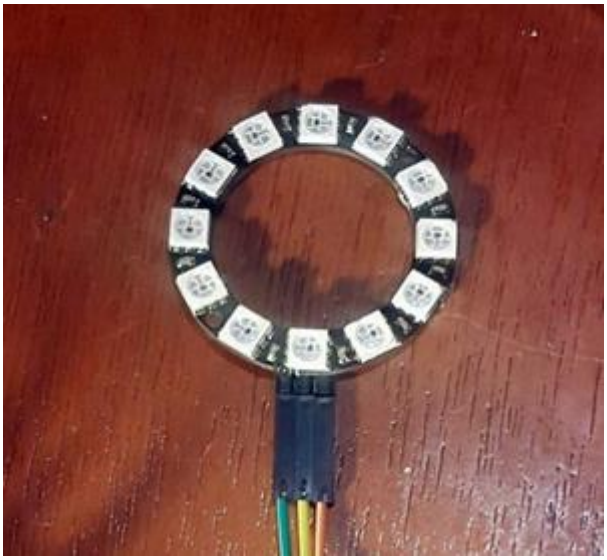


Figura 19: Display NeoPixel WS2812
Fonte: O Autor

A Tabela 3 mostra a associação de cores com notas musicais:

Tabela 3: Associação de cores

Nota	**C**	**D**	**E**	**F**	**G**	**A**	**B**
Cor	Vermelho	Amarelo	Branco	Verde	Azul	Magenta	Ciano

4.4.1 COMUNICAÇÃO COM O *DISPLAY*

A comunicação com o *display* funciona por *one wire communication*, onde existem diversas bibliotecas disponíveis para a linguagem Arduino, compatíveis com a plataforma Energia. No entanto nenhuma das bibliotecas disponível é compatível com o microcontrolador Tiva C, pois todas essas bibliotecas possuem funções dependentes do *clock*, mas nenhuma delas é compatível com o *clock* de 80 MHz do Tiva C. Como a temporização do protocolo de comunicação é bastante crítica, foi necessário implementar uma função capaz de fazer essa comunicação. **[14]**

O protocolo de comunicação com o *display* funciona de forma que são enviados pacotes de 24 *bits* de cor para uma determinada matriz de LEDs, que possui um endereço. Os bits 0 e 1 são representados por pulsos de nível lógico alto e nível lógico baixo com larguras diferentes. A Figura 20 mostra a representação dos bits 0 e 1, onde os valores de temporização podem ser verificados na Tabela 4. **[14]**

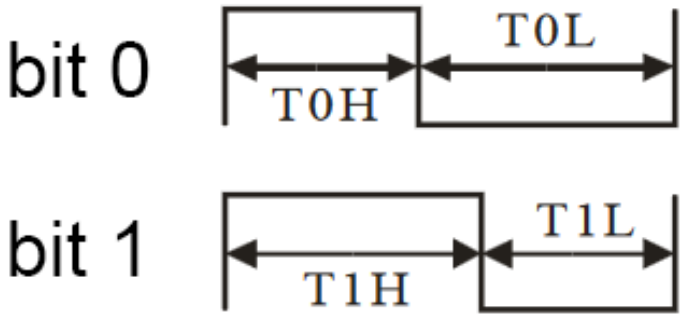


Figura 19: Largura de pulso dos bits
Fonte: *datasheet* NeoPixel WS2812[14] (modificada pelo autor)

Tabela 4: Temporizações do display WS2812

	Tempo (µs ± 150 ns)
T0H	0.35
T0L	0.80
T1H	0.70
T1L	0.60

Fonte: *datasheet* NeoPixel WS2812 **[14]**

Então para definir a cor de um LED são enviados dois *bytes*, sendo o primeiro deles o endereço do primeiro LED e o segundo o endereço do último LED a ter o seu estado alterado, na sequência são enviados 3 *bytes* que definem a cor. A sequência de *bits* enviada é G7, G6, G5, G4, G3, G2, G1, G0, R7, R6, R5, R4, R3, R2, R1, R0, B7, B6, B5, B4, B3, B2, B1 e B0, onde G representa a cor verde, R a cor vermelha e B a cor azul. **[14]**

Para implementar o protocolo de comunicação no código, foi encontrada na plataforma colaborativa GitHub uma função criada pelo usuário Jordan Mosher capaz de controlar o *display* WS2812 com o microcontrolador Stellaris EK-LM4F120XL. **[15]**

Para o funcionamento da comunicação, foram necessárias algumas alterações no código de Jordan Mosher, como configurações de temporização e adaptações de endereçamento dos LEDs, pois o código original foi desenvolvido para um *display* de maior número de LEDs. Após essas adaptações foi implementado através do Timer1A uma interrupção no código principal que é

acionada a cada 75 ms (6 milhões de ciclos de máquina) que envia para o *display* qual cor ele deve acender, e se deve acender por completo ou pela metade (nota sustenida). A cor é definida por uma rotina de "*switch case*", baseada na análise do resto da divisão do número MIDI atual por 12. As cores e suas combinações são definidas através de códigos hexadecimais com as suas componentes armazenadas em variáveis do tipo inteiro chamadas de **r**, **g** e **b**, que são enviadas à função de Jordan Mosher. Nessa rotina também é definido o endereço do último LED a ser aceso (o primeiro LED tem como endereço sempre 0), sendo no caso de uma "nota cheia" o LED de endereço 11, e no caso de uma nota sustenida o LED de endereço 5. A tabela 5 mostra todos os *cases* da rotina.

Tabela 5: Parâmetros a serem enviados pelo display

Nota	Resto	r	g	b	End	Cor
C	0	0x30	0x00	0x00	11	Vermelho
C#	1	0x30	0x00	0x00	5	Vermelho
D	2	0x30	0x30	0x00	11	Amarelo
D#	3	0x30	0x30	0x00	5	Amarelo
E	4	0x15	0x15	0x15	11	Branco
F	5	0x00	0x30	0x00	11	Verde
F#	6	0x00	0x30	0x00	5	Verde
G	7	0x00	0x00	0x30	11	Azul
G#	8	0x00	0x00	0x30	5	Azul
A	9	0x30	0x00	0x30	11	Magenta
A#	10	0x30	0x00	0x30	5	Magenta
B	11	0x00	0x30	0x30	11	Ciano

4.4.2 PARÂMETROS ELÉTRICOS DO *DISPLAY*

De acordo com o *datasheet* do *display* [14] cada LED de cor individual em sua intensidade máxima (cor em 0xFF) consome 20 mA, o que significa que cada matriz RGB consome 60 mA, com as 12 matrizes do *display* consumindo um total de 720 mA.

4.5 PARÂMETRO *VELOCITY*

Para o músico ter controle sobre o parâmetro *velocity* das notas enviadas via MIDI foi instalado no controle de disparo um sensor de força resistivo do tipo FSR (*Force Sensitive Resistor*). Os sensores FSR geralmente têm uma resistência elétrica muito alta (na ordem de alguns mega ohms) quando não estão sendo pressionados, e a sua resistência diminui conforme a intensidade da força aplicada. Foi utilizado um FSR de uma polegada de diâmetro similar ao da imagem:

Figura 20: Sensor de força
Fonte: https://www.sparkfun.com/products/9375 acesso em 31/05/2017

Como para o sensor obtido não foi encontrado nenhum *datasheet* específico, foi elaborado um circuito divisor de tensão com um potenciômetro para definir o melhor valor de resistor para aproveitar ao máximo a faixa dinâmica do ADC do microcontrolador. Foi então definido um resistor de 3.3 kΩ, conforme mostra o esquemático da Figura 22.

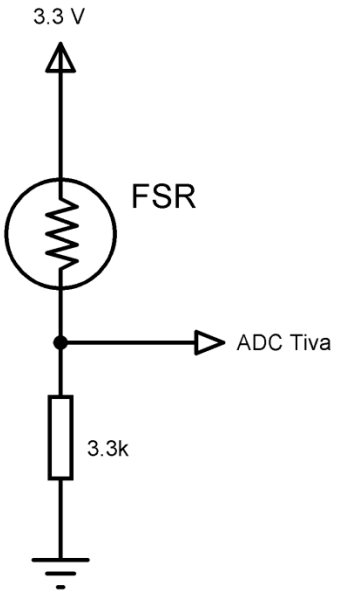


Figura 21: Esquemático do FSR
Fonte: O Autor

Para o processo de calibração do FSR foram analisadas medidas da leitura do valor analógico do divisor de tensão entre o FSR e o resistor, de forma com que de acordo com alguns níveis de força aplicados foram definidos valores de *velocity* desejados, e então foi montada uma tabela que relaciona os níveis de *velocity* com os valores lidos pela entrada analógica.

Tabela 6: Relação ADC – Velocity

Valor lido pelo ADC (12 bits)	***Velocity***
800	40
1300	60
1850	70
2100	80
2500	100
3300	127

Então através do *toolbox Curve Fitting* do Matlab foi feita uma interpolação polinomial para encontrar uma equação que se aproximasse dos valores estipulados, então foi encontrada a equação 7.

$$f(x) = (2.577 * 10^{-6})x^2 + 0.02405x + 20.37 \quad \textbf{(7)}$$

A partir da equação encontrada foi feita uma análise gráfica para verificar a sua coerência, mostrada na Figura 23. A curva definida pela equação é mostrada pela linha contínua azul, enquanto que os valores experimentais são representados pelos pontos vermelhos.

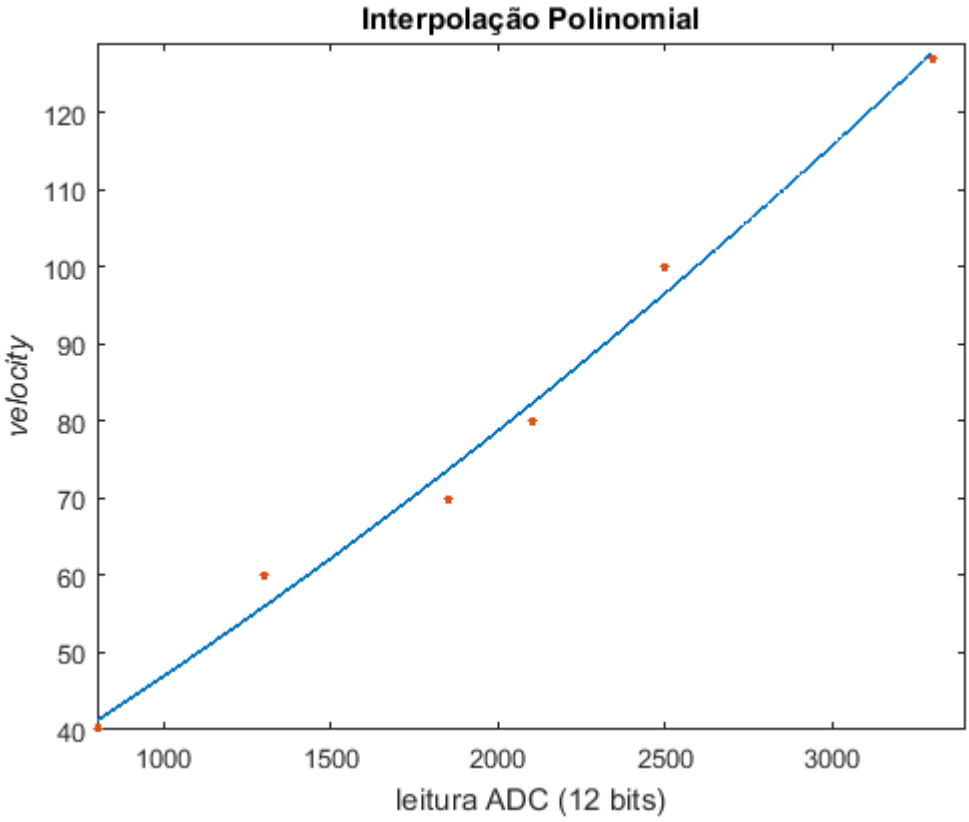


Figura 22: Interpolação polinomial de segunda ordem
Fonte: O Autor

Conforme pôde ser verificado no gráfico, a curva encontrada possuí erros de até 7%, porém dado que os pontos foram pré-definidos empiricamente, a curva foi considerada válida para servir como equação de calibração para o parâmetro de *velocity*. Também foi implementado no código uma rotina que verifica se o valor de *velocity* está entre 40 e 127, caso o valor seja menor que 40, é enviado 40, e caso seja superior a 127, é enviado 127.

4.6 DESENVOLVIMENTO DO SOFTWARE

O desenvolvimento do software foi realizado através de diversas etapas de testes e otimizações ao longo do desenvolvimento do projeto. Nesta seção serão discutidas as principais partes do programa da interface: a configuração inicial, o loop principal e as funções mais importantes para o funcionamento do programa. O código de fonte completo do programa encontra-se no **Anexo 4**.

4.6.1 CONFIGURAÇÃO INICIAL

Ao inicializar o microcontrolador, tem-se no código a definição de todas as variáveis e constantes utilizadas pelo programa e posteriormente a função *setup*, função que configura pinos de entrada e funções importantes para o funcionamento da interface como mostrado no fluxograma da Figura 23.

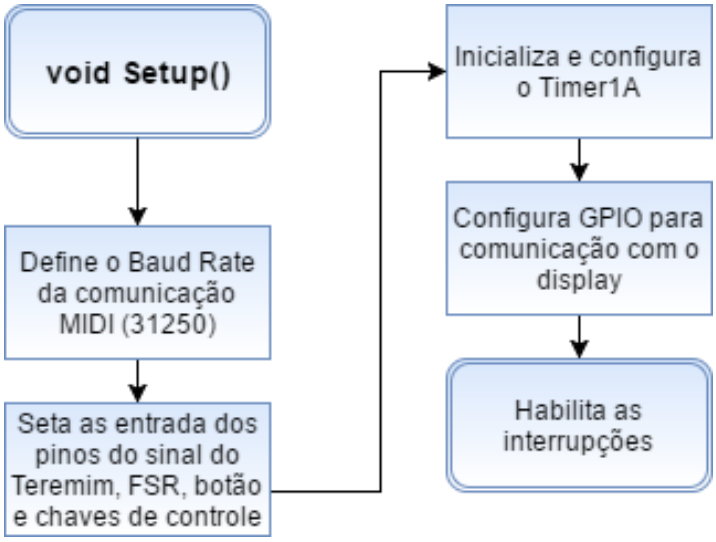


Figura 23: Função de inicialização
Fonte: O Autor

Nessa função, são definidas e habilitadas as funções de interrupção do sinal do Teremim e do botão de disparo das notas. Também é configurado o Timer1A. Uma função chamada **configureTimer1A()** é responsável pela configuração da interrupção que acontece a cada 75 ms para informar ao *display* WS2812 qual nota está sendo obtida através da leitura da frequência do sinal do Teremim. A função que realiza esta interrupção é chamada **Timer1IntHandler()** e será comentada com mais detalhes posteriormente.

4.6.2 *LOOP* PRINCIPAL

O *loop* principal é dividido nos dois modos de operação que podem ser selecionados pelo usuário: o modo disparo e o modo contínuo. É feito o monitoramento do pino digital associado à chave de seleção de modos e a cada execução do *loop* se verifica se o nível lógico atual. Caso o nível lógico seja 1 o *loop* executará o modo disparo e em nível lógico 0 executará o modo contínuo.

Em ambos os modos, o *loop* principal sempre inicia realizando o cálculo da distância baseado na frequência lida do Teremim. Essa distância é calculada utilizando os parâmetros obtidos da curva de resposta em frequência em função da distância obtida anteriormente. Em seguida o resultado é enviado como parâmetro de entrada para a função que é responsável pela linearização do número MIDI associado. O valor final obtido da linearização é armazenado na variável do tipo inteiro chamada **nota_new**. Esta variável é uma das mais importantes do programa, pois representa a nota musical equivalente obtida do Teremim que será enviada na saída MIDI do circuito.

Com a variável **nota_new** obtida, o *loop* principal segue para a escolha dos modos e caso esteja ajustado o modo de disparo, são efetuadas as etapas ilustradas na Figura 24.

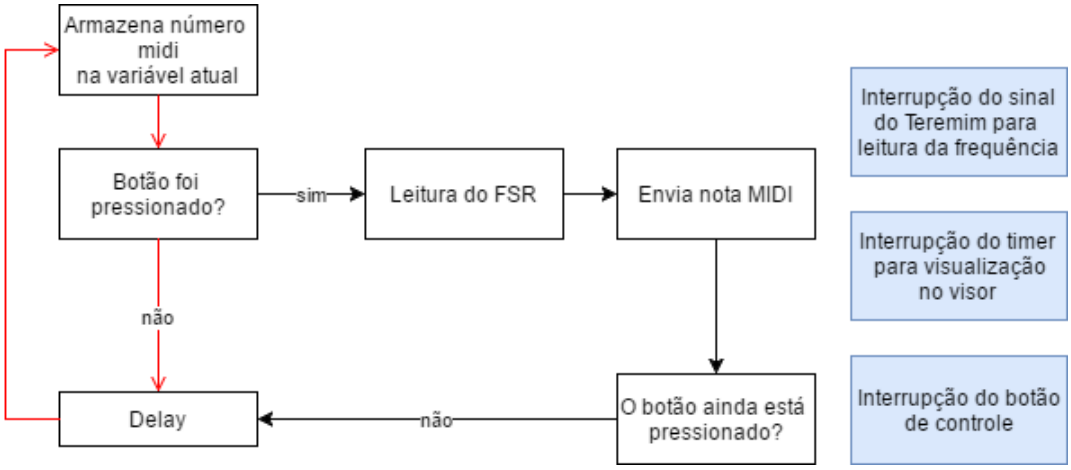


Figura 24: Modo de disparo
Fonte: O Autor

O processo inicia com a verificação da variável **flagb**, ativada na interrupção do botão de controle para indicar se o botão foi pressionado. Caso o botão não seja

pressionado, o *loop* segue armazenando o número MIDI na variável **nota_new** de acordo com a movimentação próxima à antena do Teremim. Para fins de estabilidade e *debouncing* foi inserido um pequeno *delay* de 20 ms.

Caso seja verificado que o botão foi pressionado, é chamada a função responsável pela leitura do sensor FSR para se obter a pressão exercida pelo usuário. Essa informação é armazenada na variável *velocity* e sem seguida é enviada na mensagem MIDI. A mensagem possui como parâmetros a nota musical contida em **nota_new**, a intensidade contida em *velocity* e o canal MIDI a ser enviado. Após esse envio é verificado se o botão ainda está pressionado. No momento em que o botão deixa de ser pressionado, é enviada a mensagem que desliga a nota, ou seja, que informa ao VSTi que é o momento que encerrar a continuidade daquela nota musical.

É importante destacar que ao mesmo tempo que isso ocorre, as interrupções para leitura da frequência e do **Timer1A** estão ocorrendo e enviando a cor correspondente à variável **nota_new** para o *display,* independentemente do botão de controle estar sendo pressionado para enviar as notas ou não.

O segundo modo de operação ocorre quando é garantido nível lógico 0 na porta digital associada. O comportamento é similar ao primeiro modo, porém com alguns detalhes específicos. Esse comportamento pode ser verificado no fluxograma ilustrado na Figura 26.

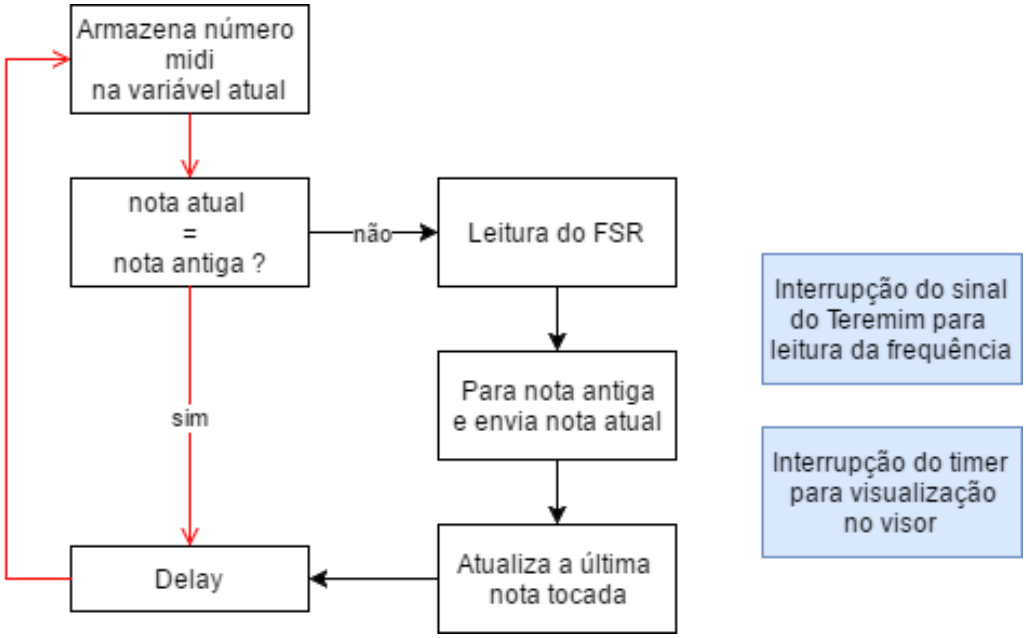


Figura 25: Modo contínuo
Fonte: O Autor

Este modo é o que mais se aproxima do comportamento do Teremim analógico pois envia continuamente as notas musicais associadas e ainda possibilita a alteração do volume sem a necessidade de haver movimento em relação à antena.

Ocorre a verificação se a variável atual **nota_new** é igual a uma variável chamada **nota_old**. Caso essa condição seja falsa, o programa para de enviar a nota antiga e envia a nota atual. Em seguida atualiza-se a última nota tocada armazenando o valor de **nota_new** em **nota_old**.

Ao contrário do modo de disparo, nesse modo tem-se a informação obtida pelo sensor FSR sendo utilizada como parâmetro para enviar o *after-touch* a todo momento. Independentemente se a nota atual possuir valor igual ou diferente da nota antiga, o programa envia continuamente a informação da pressão aplicada no FSR, como um parâmetro de *velocity* no canal MIDI número um. O processo completo do *loop* é ilustrado na Figura 27.

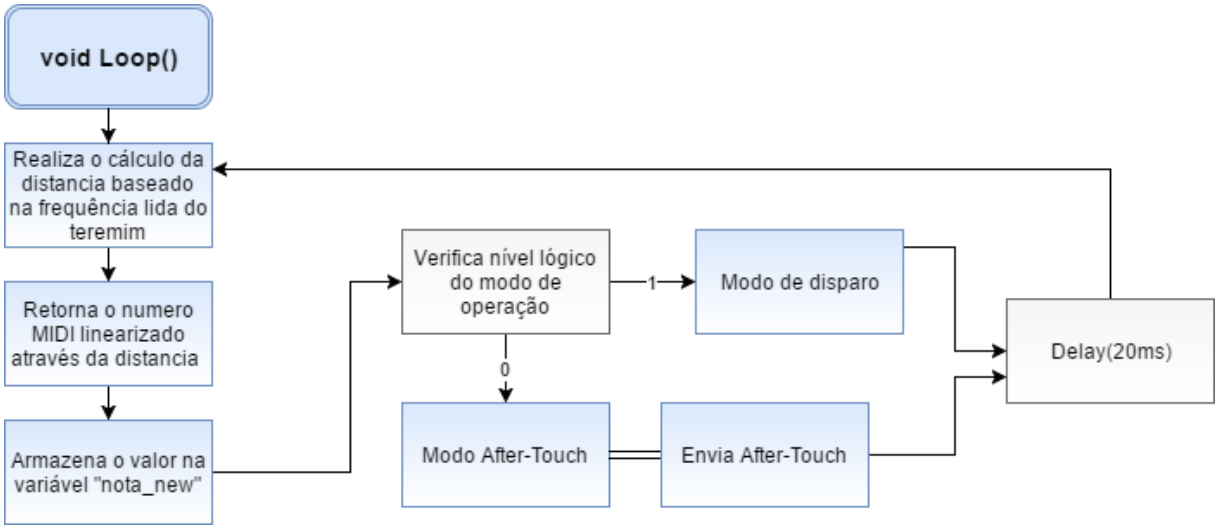


Figura 26: Loop completo
Fonte: O Autor

O envio do *after-touch* pode ser aproveitado de várias maneiras diferentes. No *software* VSTi a ser utilizado, o usuário pode associar este parâmetro a qualquer parâmetro do software. O algoritmo foi desenvolvido com a intenção de se utilizar a informação como controle de volume, mas nada impede que o usuário defina o *after-touch* para variar uma modulação ou qualquer outro efeito específico, como por exemplo um vibrato.

4.6.3 FUNÇÕES E INTERRUPÇÕES

Serão apresentadas nesta seção, duas das principais funções de interrupção executadas pelo programa. A primeira função e talvez mais crítica para o bom funcionamento da interface é a função que realiza a leitura da frequência do sinal proveniente do Teremim. Ocorre pela interrupção por borda de subida na entrada digital do microcontrolador e está ilustrada na Figura 28.

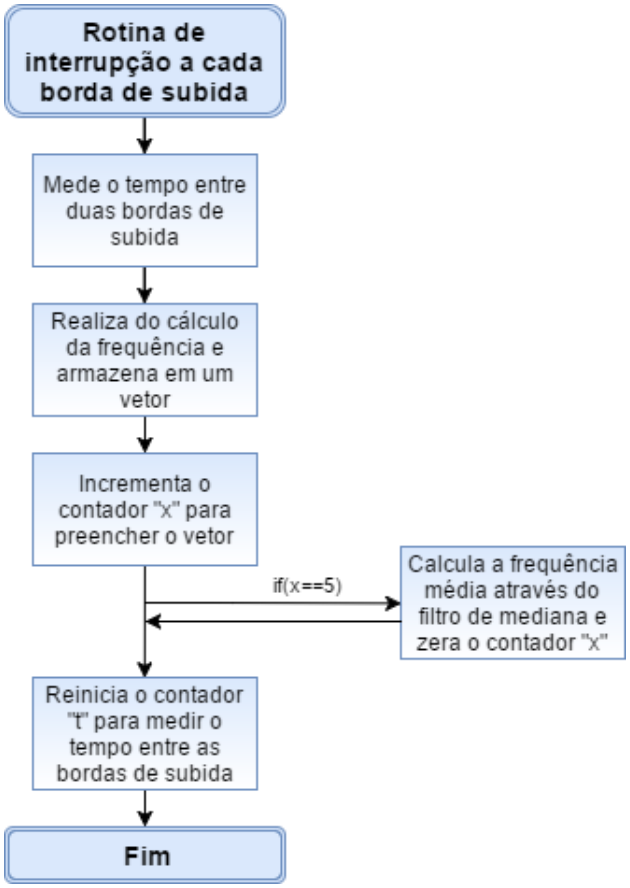


Figura 27: Interrupção por borda de subida
Fonte: O Autor

Essa interrupção é acionada através do pino **PC_5** do microcontrolador. A contagem do tempo entre as bordas de subida é realizada pela função **micros()** e armazenadas em variáveis do tipo *unsigned long*. A frequência é armazenada em um vetor tipo *float* que segue para uma função específica que realiza a filtragem de mediana dos cinco valores encontrados no vetor.

Outra função de interrupção é a interrupção do **Timer1A**. Essa função é configurada pela função **configureTimer1A()** no *setup* e foi definida para ocorrer a cada 100 ms. Sua rotina está ilustrada na Figura 29.

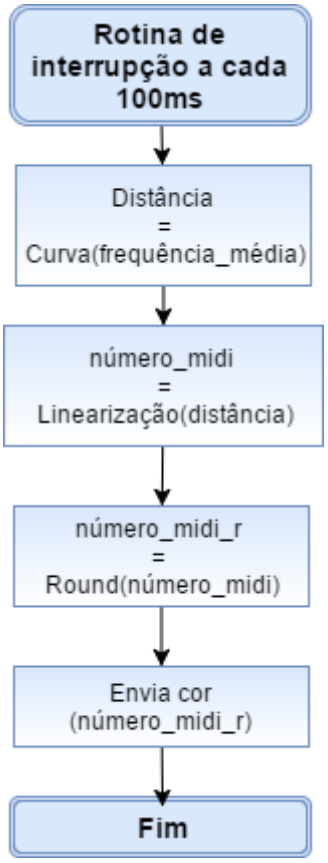


Figura 29: Interrupção do timer
Fonte: O Autor

4.7 APRESENTAÇÃO DO PROTÓTIPO

Após todo o processo de desenvolvimento de *software* e *hardware* foi desenhado o *layout* de uma placa de circuito impresso com o auxílio da ferramenta Autodesk Eagle. Essa placa foi desenvolvida com o intuito de agregar todos os componentes eletrônicos utilizados no projeto dentro de uma caixa que pode ser instalada próxima ao Teremim. A Figura 30 mostra o *layout* da PCB desenhada.

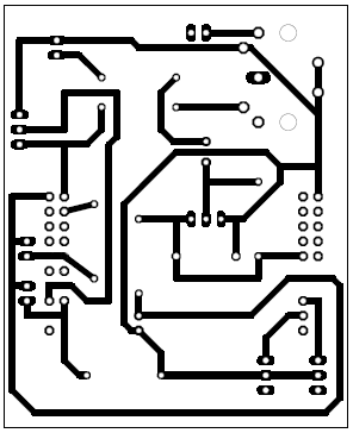


Figura 28: Placa de circuito impresso
Fonte: O Autor

A placa foi desenhada de modo que funcione como um *shield* para o *launchpad* do Tiva C, sendo fácil de encaixar o microcontrolador. As conexões de cada pino do *launchpad* pode ser verificada no **Anexo 3**.

Então a placa encaixada no Tiva C foi instalada em um *case* plástico com alguns furos para as chaves seletoras, controle (utilizando um plug USB), *jack* P10 (comunicação com o Teremim), cabo de alimentação e conector MIDI DB9. O visor também foi instalado na caixa. As Figuras 31 e 32 mostram a aparência do *case*.

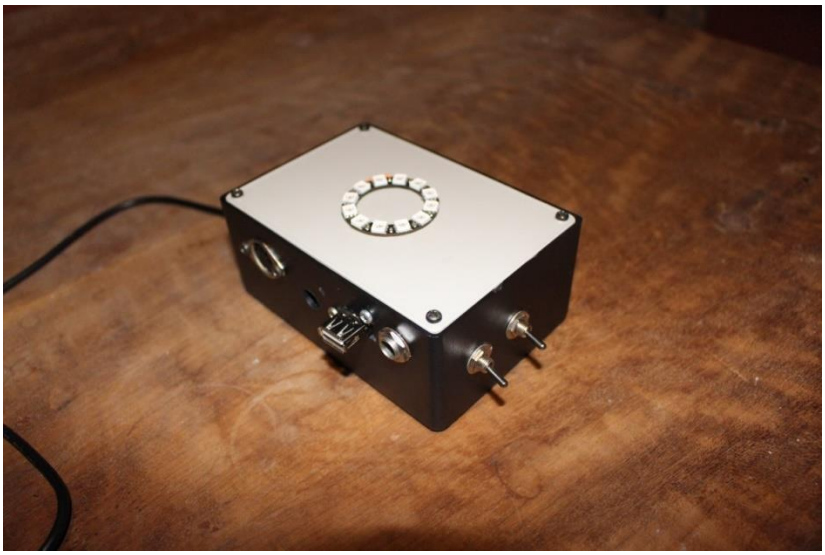


Figura 29: Vista A do case
Fonte: O Autor

Figura 30: Vista B do case
Fonte: O Autor

Para a alimentação recomenda-se um adaptador AC/USB de no mínimo 1 A para garantir toda a alimentação do circuito com uma boa margem de segurança, dado que o item de maior consumo do projeto é o visor NeoPixel WS2812, que em seu limite de operação consome 750 mA.

5 CONSIDERAÇÕES FINAIS

A interface MIDI abre espaço para novos trabalhos futuros que podem facilitar ou aprimorar seu funcionamento atual. Uma possível evolução seria a implementação de uma comunicação sem fio para enviar os dados do FSR e do botão de disparo diretamente para o microcontrolador. Durante os momentos iniciais do trabalho foram levantadas várias possibilidades de hardware para o controle de *velocity* antes da decisão pelo FSR. Não foram implementadas formas de controle com potenciômetros nem com sensores de ultrassom devido a dificuldades mecânicas e baixa relação sinal ruído.

No período intermediário do trabalho também optou-se por utilizar os dois modos de operação. O botão de disparo evita que sejam enviadas notas indesejadas no momento da execução do usuário, porém o tempo de reação do músico é mais lento por não se ter uma realimentação auditiva. Através do modo contínuo, que simula o comportamento de um Teremim tradicional, é possível variar o volume de uma nota musical fixa no espaço, porém este modo exige que o *software* VSTi seja condizente com sua dinâmica em termos de *sustain*, *decay* e *attack*. Outra possível evolução no *software* da interface seria o processamento do espectro do sinal de entrada com o objetivo de se enviar no modo contínuo as frequências em uma maior resolução através de mensagens de *pitch-bend*. Desta maneira se eliminaria a descontinuidade entre as notas musicais, entretanto seria necessário desenvolver no algoritmo uma maneira de pausar o envio do parâmetro *pitch-bend* em momentos onde a mão do músico estivesse em repouso ou em uma velocidade muito baixa.

O visor WS2812 foi programado de modo a fornecer uma realimentação visual de maneira quantizada em função do número MIDI. O músico durante sua performance não possui a informação de onde é o limite da próxima nota ou da nota anterior nas luzes emitidas pelos LEDs do visor. Seria desejável um controle de brilho ou intensidade dos LEDs de acordo com a proximidade à frequência da nota

musical correspondente. Com essa variação de brilho a possibilidade de uma nota ser tocada errada por estar muito próxima a outra diminuiria consideravelmente.

Como observado nos itens detalhados anteriormente, o objetivo geral do trabalho foi atingido: facilitar o uso do instrumento, aumentar sua gama de possibilidades sonoras em termos de timbres e permitir um controle de variação dinâmica para o instrumento. Assim como o objetivo geral, todos os objetivos específicos foram previamente atingidos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Guskuma, Rodrigo Uehara, *Desenvolvimento de controlador* MIDI empregando linguagem de alto nível com fins didáticos para o ensino de engenharia, Universidade de São Paulo, 2016

[2] Glinsky, Albert (2000). Theremin: Ether Music and Espionage. Urbana, Illinois: University of Illinois Press. p. 25

[3] Aaron Koga, Physics 475, dezembro de 2006:

http://www.thekoga.com/research-and-academics/study-of-a-theremin/, acesso em 23/04/2017

[4] John D. Polstra, A Vacuum Tube Theremin, 2015:

http://www.channelroadamps.com/articles/theremin/, acesso em 23/04/2017

[5] Cortex-M4 - Technical Reference Manual, ARM DDI 0439B, março de 2010:

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

[6] Catálogo Texas Instruments, ARM® Cortex®-M4F Based MCU TM4C123G LaunchPad™ Evaluation Kit:

http://www.ti.com/tool/ek-tm4c123gxl/, acesso em 28/04/2017

[7] Datasheet Tiva C, TM4C123G, 12 de junho de 2014:

http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf/

[8] Dr. Sebastian Anthony Birch, Kent State University, The MIDI Physical Layer:

http://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/midi_physical_layer.htm/, acesso em 22/04/2017

[9] Professor Joe Wolfe, The University of New South Wales, Australia:

http://newt.phys.unsw.edu.au/jw/notes.html, acesso em 22/04/2017

[10] Professor Craig Stuart Sapp, Stanford, California, Essentials of the MIDI Protocol:

https://ccrma.stanford.edu/~craig/articles/linuxmidi/misc/essenmidi.html, acesso em 15/03/2017

[11] Dominique Vandenneucker, Arpege Music, MIDI Tutorial:

http://www.music-software-development.com/midi-tutorial.html, acesso em 15/03/2017

[12] TENG-HE, A. O.; HUI, X. I. E.; HONG-WEI, Ruan. Design and Implementation of Network Sound Module System Based on VST/VSTi Architecture. **Journal of Inner Mongolia University**, v. 2, p. 021, 2009.

[13] Microelectronic Circuits, Sixth Edition, by Adel S. Sedra and Kenneth C. Smith

[14] *Datasheet* do *display* NeoPixel WS2812:

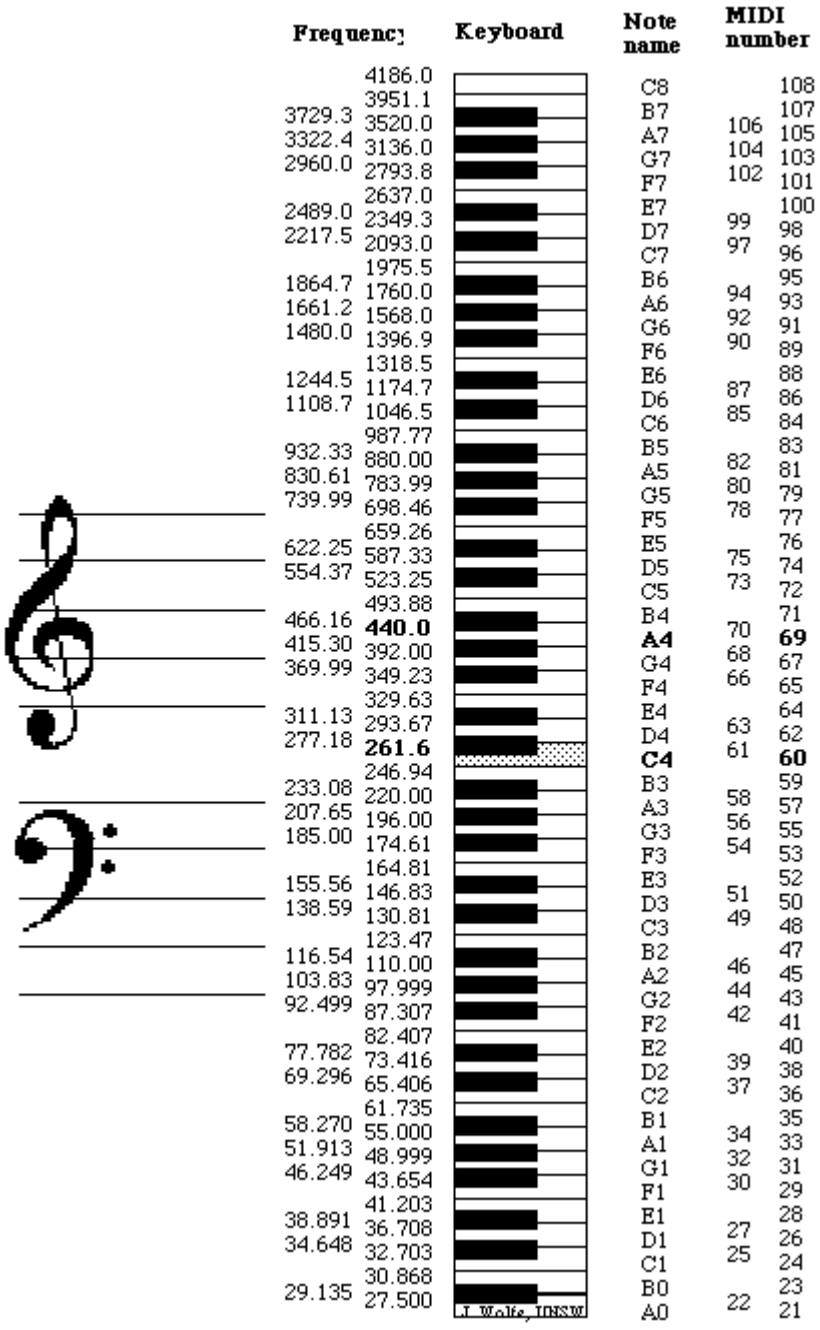
https://cdn-shop.adafruit.com/datasheets/WS2812.pdf, acesso em 01/06/2017

[15] Função de controle do *display* NeoPixel WS2812:

https://github.com/JMoosh/NeoPixelRingGauge/blob/master/Gauge.c, acesso em 01/06/2017

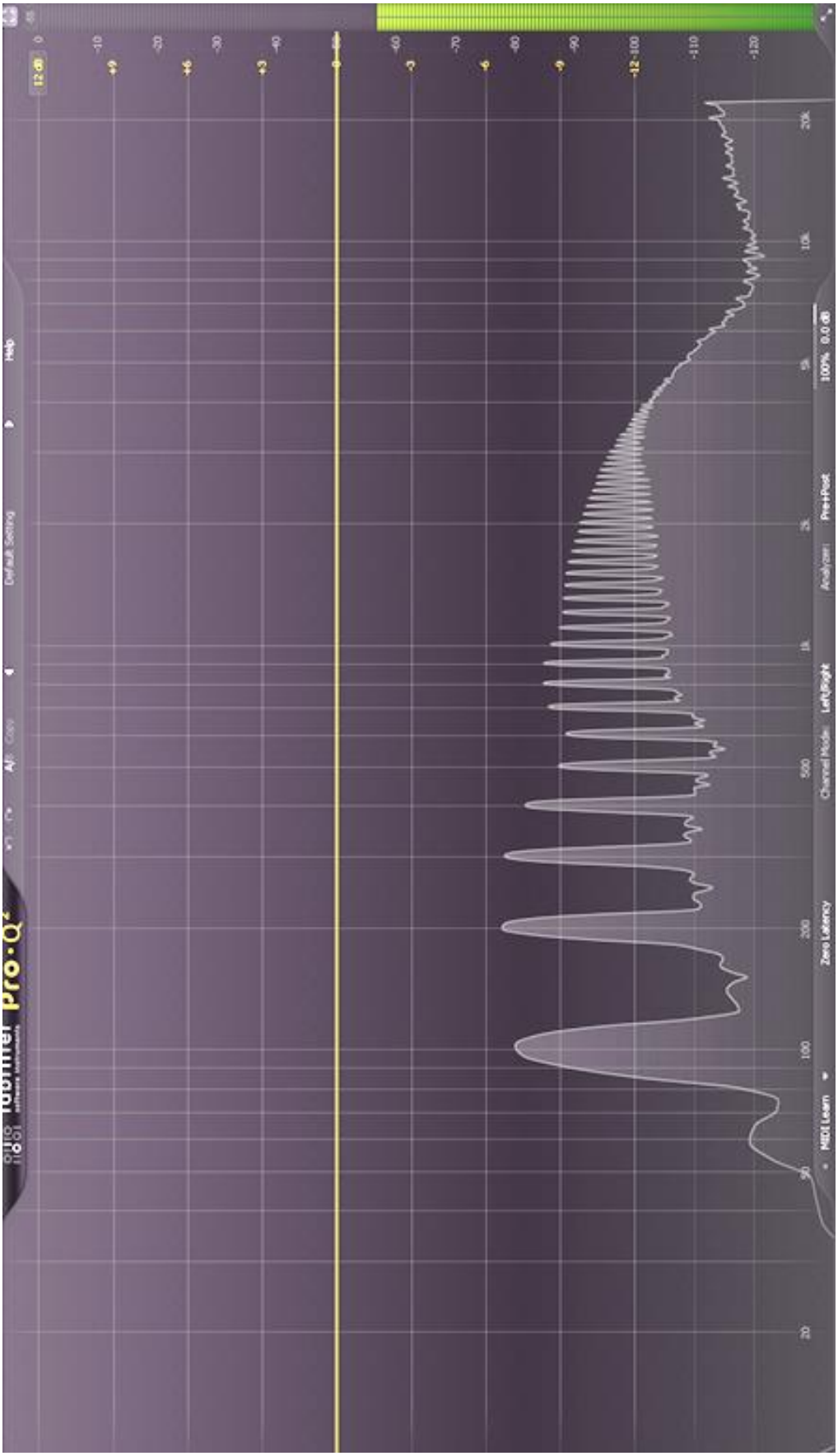
ANEXOS

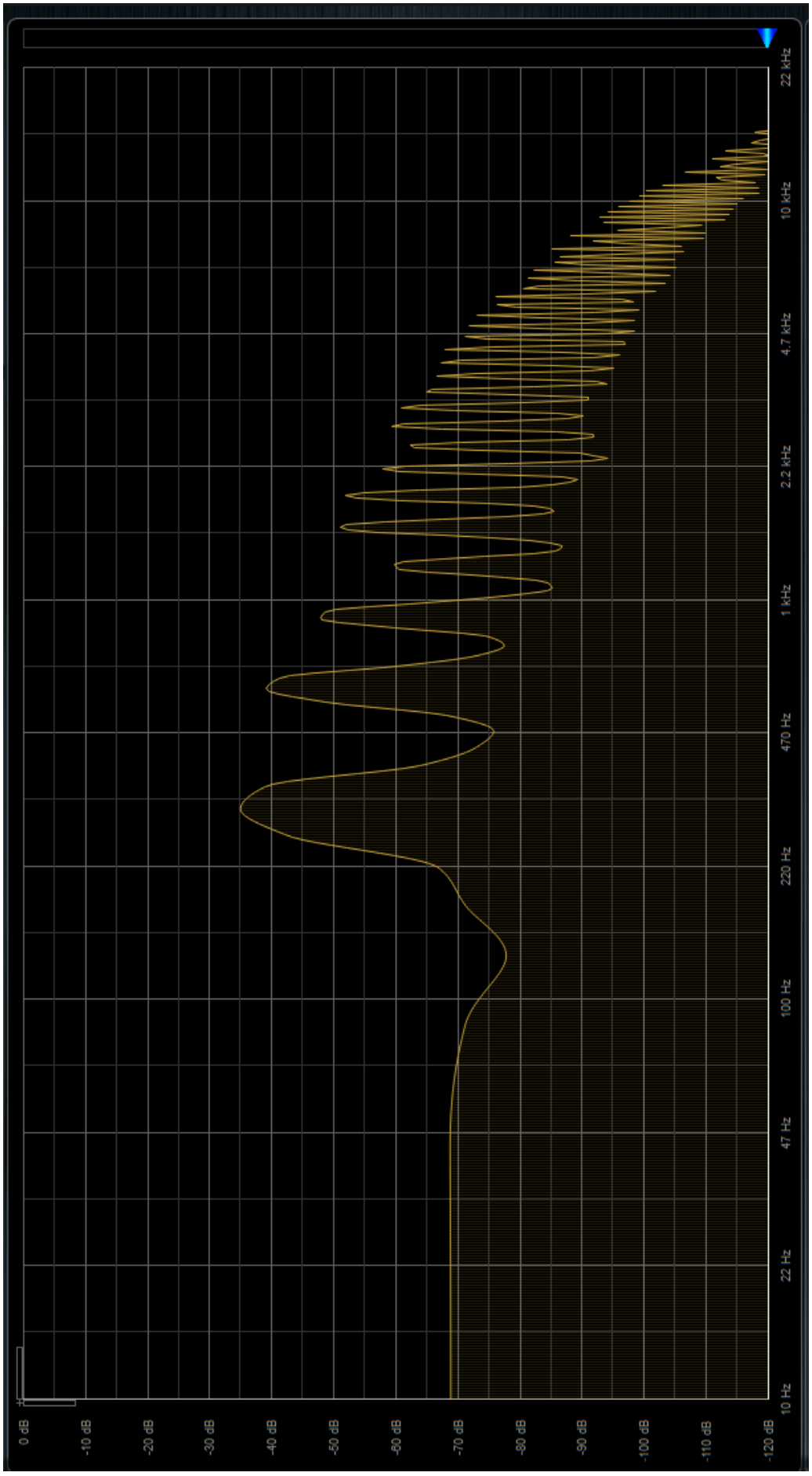
Anexo 1: Relação entre nota musical, frequência fundamental e MIDI:



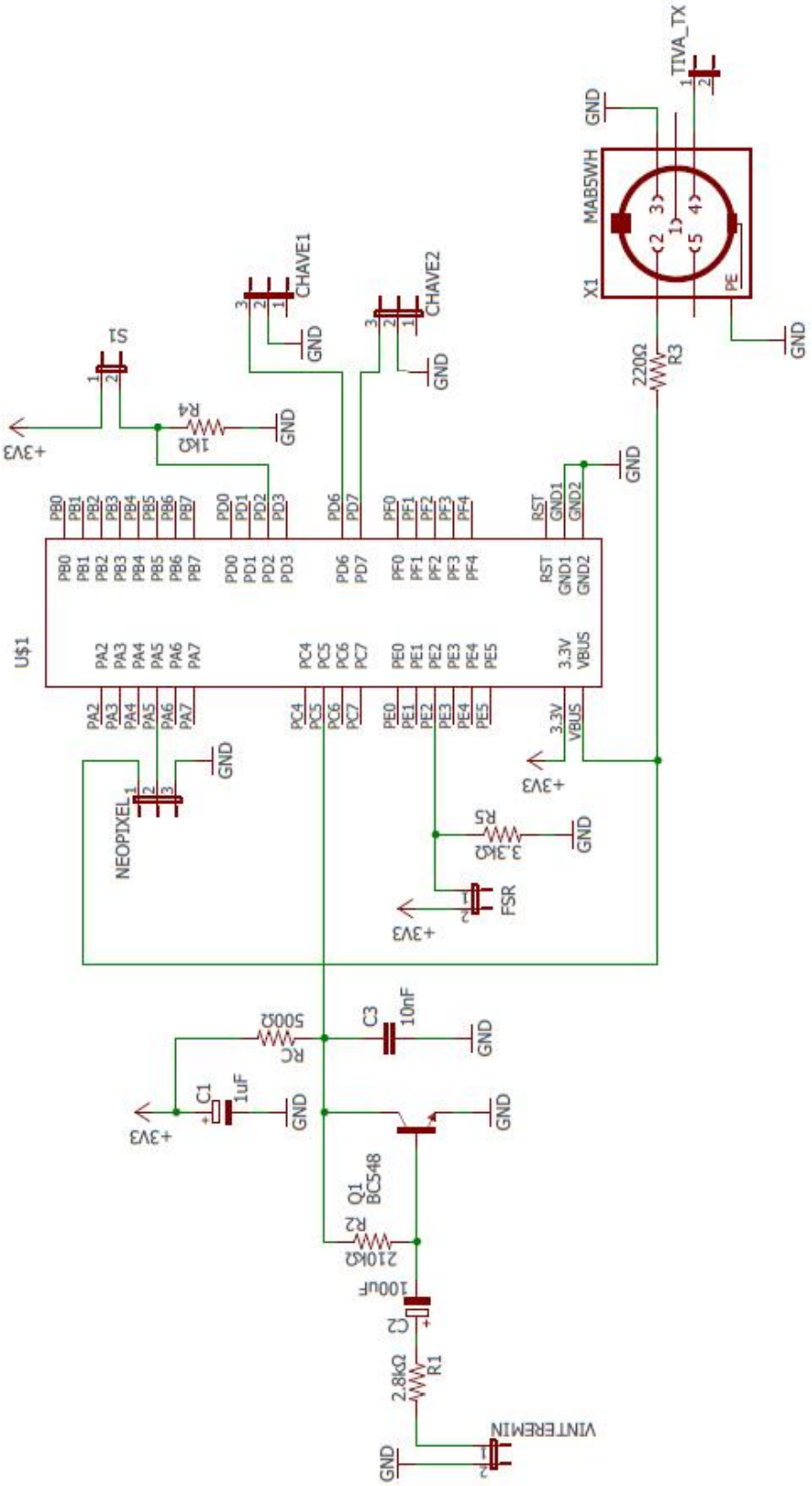
Fonte: http://newt.phys.unsw.edu.au/jw/notes.html/ acesso em 22/04/2017

Anexo 2: Análise de espectro do Teremim




22 kHz
10 kHz
4.7 kHz
2.2 kHz
1 kHz
470 Hz
220 Hz
100 Hz
47 Hz
22 Hz
10 Hz
0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB
-90 dB
-100 dB
-110 dB
-120 dB

Anexo 3: Pinos e conexões Tiva C



Anexo 4: Código de fonte do programa

```
#include <MIDI.h>
#include <math.h>
#include "stdint.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ssi.h"
#include "inc/hw_types.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "wiring_private.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/timer.h"
#include "driverlib/sysctl.h"
MIDI_CREATE_DEFAULT_INSTANCE();

// pinos de entrada
        int s_teremim = 36;             //teremim    --> PC_5
        int botao=25;                   //pino botao  -->PD_2
        int FSR_PIN = 28;               //pino FSR   --> PE_2
        int kmodo = 33;                //chave modo  --> PD6
        int koitava = 32;              //chave oitava -->PD7
//variaveis e constantes
        int velocity;
        int nota_old = 48;
        int nota_new = 48;
        double t1=1;
        double t0=1;
        int x=1;
        int y=0;
        float freq_[5];
        float frequencia_med;
//parametros da curva de distancia em funcao da frequencia (dist)
        int   a=8811;
        float b=-0.9483;
        float dist=0;
//variaveis do botao
        int   flagb=0;
//variaveis/constantes do sensor FSR
                int   readfsr;
                float afsr = 2.557*pow(10,-6);
                float bfsr = 0.02405;
                float cfsr = 20.37;
                float fsr;
//variaveis display
        uint8_t Start =0;
        uint8_t End = 5;
        uint32_t LEDnum = 12; //number of LED's
        uint32_t Green = 0x00;
        uint32_t Red = 0x55;
        uint32_t Blue = 0X00;
        uint32_t BG_Green = 0x00;
        uint32_t BG_Red = 0x00;
        uint32_t BG_Blue = 0x00;
//variavel de modos de operacao
        int modo=1;    //define modos// modo 0 teremim, modo 1 botao



void setup(){

  Serial.begin(31250);                          //definindo serial para comunicacao
MIDI
  pinMode(FSR_PIN, INPUT);                      //setando entrada do sensor
```

```
  pinMode(s_teremim, INPUT_PULLUP);           //setando entrada do teremim com
Pullup
  pinMode(botao, INPUT);                      //sentando entrada do botao
  pinMode(kmodo, INPUT_PULLUP);               //definindo chave de selecao de modos
  pinMode(koitava, INPUT_PULLUP);             //definindo chave de seleção de
oitavas
  attachInterrupt(s_teremim,subida,RISING);   //interrupcao de borda de subida no
sinal do teremim
  attachInterrupt(botao,botaoapertado,RISING);//interrupcao de borda de subida no
botao
  configureTimer1A();                         //configuracao do timer para o
display
  configureDisplay();                         //configuracao da comunicacao do
display
}

//funcao linearizacao
int numeromidilin(float dist){
  float n;
  if(digitalRead(koitava)==1){
  n = -0.48*dist+79.2;  //eq para 2 oitavas de 48 até72 com as distancias de 65 a
15cm
  }
  if(digitalRead(koitava)==0){
  n=-0.24*dist+63.6;  //eq para 1 oitava de 48 até 60 com as distancias de 65 a
15cm
  }
  n=round(n);
  return n;
}


//funcao mediana
float mediana(float vetor[5]){
  float tmp;
  int i;
  int j;
  int n = 5;

        for(i=0;i<n;i++)
     {
           for(j=0;j<n-i;j++)
           {
                 if(vetor[j]>vetor[j+1])
                 {
                       tmp=vetor[j];
                       vetor[j]=vetor[j+1];
                       vetor[j+1]=tmp;
                 }
           }
     }

    return vetor[3];
  }

//interrupcao para calcular freq
void subida(){

             t1 = micros()-t0;
             freq_[x] = 1000000/t1;
             t0 = micros();
             x++;
             if(x == 6){

                frequencia_med = mediana(freq_);
                x = 1;
              }
```

```
}

//interrupcao setando flag do botao a cada borda de subida
void botaoapertado(){
  flagb = 1;
}

//funcao para calcular velocity em função da força aplicada ao FSR
void leitura_FSR(){

  readfsr = analogRead(FSR_PIN);
  fsr = afsr*(pow(readfsr,2)) + bfsr*readfsr + cfsr;
  fsr = round(fsr);
  velocity = fsr;

}

//config comunicação do display
void configureDisplay(){
  SysCtlPeripheralEnable( SYSCTL_PERIPH_SSI0 );
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  SSIDisable( SSI0_BASE );
  GPIOPinConfigure(GPIO_PA2_SSI0CLK);
  GPIOPinConfigure(GPIO_PA3_SSI0FSS);
  GPIOPinConfigure(GPIO_PA5_SSI0TX);
  GPIOPinTypeSSI(GPIO_PORTA_BASE,GPIO_PIN_5|GPIO_PIN_3|GPIO_PIN_2);
  SSIConfigSetExpClk( SSI0_BASE,
                                SysCtlClockGet(),
                                SSI_FRF_MOTO_MODE_1,
                          SSI_MODE_MASTER,
                                2150000,
                                8 );
  SSIEnable( SSI0_BASE );
}

//config interrup timer para display
void configureTimer1A(){
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1); // habilitando Timer 1 Clock
  ROM_IntMasterEnable(); // habilitando interrupcoes
  ROM_TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC); // Configurando operacao do
timer como periodica

  ROM_TimerLoadSet(TIMER1_BASE, TIMER_A,8000000); //80MHz/8MHz --> 10Hz --> 100ms

  TimerIntRegister(TIMER1_BASE, TIMER_A, &Timer1IntHandler); //registra a
interrupçao para o endereço da funcao Timer1IntHandler
  ROM_IntEnable(INT_TIMER1A);  // Habilitando interrupcao do Timer 1A
  ROM_TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // Timer 1A Interrupcao
quando Timeout
  ROM_TimerEnable(TIMER1_BASE, TIMER_A); // inicia Timer 1A

}

//funcao de ajuste de taxa de comunicacao do display
uint32_t Byte2Baud(uint8_t input){
  int8_t rgbBitIdx;
  uint32_t BaseBaud = 9586980;
  for( rgbBitIdx=7; rgbBitIdx>=0; rgbBitIdx-- ) {
    if (input & (0x1<<rgbBitIdx)){
      BaseBaud = BaseBaud | 0x1<<((rgbBitIdx*3)+1);
    }
  }
  return BaseBaud;
}

//funcao display para ascender leds definidos
```

```
void LED_RING_Percent(uint32_t LEDnum, uint8_t Start, uint8_t End, uint32_t Red,
uint32_t Green, uint32_t Blue, uint32_t BG_Red, uint32_t BG_Green, uint32_t
BG_Blue){
  int8_t LEDIdx;
  uint8_t LEDtxIdx;
  int8_t buffer[144]; // = (LEDnum*9)
  uint32_t RedBaud = Byte2Baud(Red);
  uint32_t GreenBaud = Byte2Baud(Green);
  uint32_t BlueBaud = Byte2Baud(Blue);
  uint32_t BG_RedBaud = Byte2Baud(BG_Red);
  uint32_t BG_GreenBaud = Byte2Baud(BG_Green);
  uint32_t BG_BlueBaud = Byte2Baud(BG_Blue);
  uint8_t StartLED = Start;
  uint8_t EndLED = End;

  for( LEDIdx=0; LEDIdx < (StartLED-1); LEDIdx++ ) {
    buffer[(LEDIdx*9)]     = (BG_GreenBaud>>16);
    buffer[((LEDIdx*9)+1)] = (BG_GreenBaud>>8);
    buffer[((LEDIdx*9)+2)] = (BG_GreenBaud);
    buffer[((LEDIdx*9)+3)] = (BG_RedBaud>>16);
    buffer[((LEDIdx*9)+4)] = (BG_RedBaud>>8);
    buffer[((LEDIdx*9)+5)] = (BG_RedBaud);
    buffer[((LEDIdx*9)+6)] = (BG_BlueBaud>>16);
    buffer[((LEDIdx*9)+7)] = (BG_BlueBaud>>8);
    buffer[((LEDIdx*9)+8)] = (BG_BlueBaud);
  }

  for( LEDIdx=StartLED; LEDIdx < (EndLED+1); LEDIdx++ ) {
    buffer[(LEDIdx*9)]     = (GreenBaud>>16);
    buffer[((LEDIdx*9)+1)] = (GreenBaud>>8);
    buffer[((LEDIdx*9)+2)] = (GreenBaud);
    buffer[((LEDIdx*9)+3)] = (RedBaud>>16);
    buffer[((LEDIdx*9)+4)] = (RedBaud>>8);
    buffer[((LEDIdx*9)+5)] = (RedBaud);
    buffer[((LEDIdx*9)+6)] = (BlueBaud>>16);
    buffer[((LEDIdx*9)+7)] = (BlueBaud>>8);
    buffer[((LEDIdx*9)+8)] = (BlueBaud);
  }

  for( LEDIdx=(EndLED+1); LEDIdx < (LEDnum+1); LEDIdx++ ) {
    buffer[(LEDIdx*9)]     = (BG_GreenBaud>>16);
    buffer[((LEDIdx*9)+1)] = (BG_GreenBaud>>8);
    buffer[((LEDIdx*9)+2)] = (BG_GreenBaud);
    buffer[((LEDIdx*9)+3)] = (BG_RedBaud>>16);
    buffer[((LEDIdx*9)+4)] = (BG_RedBaud>>8);
    buffer[((LEDIdx*9)+5)] = (BG_RedBaud);
    buffer[((LEDIdx*9)+6)] = (BG_BlueBaud>>16);
    buffer[((LEDIdx*9)+7)] = (BG_BlueBaud>>8);
    buffer[((LEDIdx*9)+8)] = (BG_BlueBaud);
  }

  while(SSIBusy( SSI0_BASE ));

  for(LEDtxIdx = 0; LEDtxIdx < (LEDnum*9); LEDtxIdx++) {
    uint8_t data = buffer[LEDtxIdx];
    SSIDataPut( SSI0_BASE, data);
  }
}

//funcao para enviar cores no display
void enviacor(int n){
    int r, g, b, Start, metade;
    Start = 0;

    LED_RING_Percent(LEDnum, Start, 11, 0, 0, 0, BG_Red, BG_Green, BG_Blue);
    delay(1);
    switch(n%12){
      case 0:   // do verm
```

```
  r = 0x30;
  g = 0x00;
  b = 0x00;
  metade = 11;
  break;

case 1:  // do sust verm pela met
  r = 0x30;
  g = 0x00;
  b = 0x00;
  metade = 5;
  break;

case 2:    // re amarelo
  r = 0x30;
  g = 0x30;
  b = 0x00;
  metade = 11;
  break;

case 3:    // re sust amarelo metade
  r = 0x15;
  g = 0x15;
  b = 0x00;
  metade = 5;
  break;

    case 4:  // mi branco
  r = 0x15;
  g = 0x15;
  b = 0x15;
  metade = 11;
  break;

case 5:    // fa verde
  r = 0x00;
  g = 0x30;
  b = 0x00;
  metade = 11;
  break;

case 6:    // fa verde metade
  r = 0x00;
  g = 0x30;
  b = 0x00;
  metade = 5;
  break;

  case 7:    // sol azul
  r = 0x00;
  g = 0x00;
  b = 0x30;
  metade = 11;
  break;

case 8:    // sol azul metade
  r = 0x00;
  g = 0x00;
  b = 0x30;
  metade = 5;
  break;

  case 9:    // la magenta
  r = 0x30;
  g = 0x00;
  b = 0x30;
  metade = 11;
  break;
```

```
      case 10:    // la sust magenta metade
        r = 0x30;
        g = 0x00;
        b = 0x30;
        metade = 5;
        break;

        case 11:    // si ciano
        r = 0x00;
        g = 0x30;
        b = 0x30;
        metade = 11;
        break;

    }

  if(n<48)LED_RING_Percent(LEDnum, Start, 11, 0, 0, 0, BG_Red, BG_Green, BG_Blue);

    else LED_RING_Percent(LEDnum, Start, metade, r, g, b, BG_Red, BG_Green,
BG_Blue);

}

//interrupcao timer do visor a cada 100ms
void Timer1IntHandler(void){

  ROM_TimerIntClear(TIMER1_BASE, TIMER_A);
  dist = a*(pow(frequencia_med,b));
  nota_new = numeromidilin(dist);
  enviacor(nota_new);

}



void loop(){
dist = a*(pow(frequencia_med,b));     // realiza o cálculo da distancia baseado na
frequencia lida do teremim
nota_new = numeromidilin(dist);       // retorna o numero midi em funcao da
frequencia lida do teremim linearizada através da distancia

if(digitalRead(kmodo)==1){
           // verifica se está no modo botao
             if(flagb ==
1){                                                                    // verifica se
o botao foi pressionado
                              if(nota_new>=48){
                         leitura_FSR();
                                                  if (velocity <= 65){ velocity =
65;}          // ignorando valores menores que 75
                                                  if (velocity >= 127){ velocity =
127;}        // ignorando valores maiores que 127
                                                  MIDI.sendNoteOn(nota_new,velocity,1
);         // enviando nota MIDI
                                                  int nota_aux =
nota_new;                       // variavel auxiliar para controle de duracao da
nota
                              while(flagb == 1)
                              {
                                  delayMicroseconds(10);
                                if(digitalRead(botao)==LOW)
                                {
                                MIDI.sendNoteOff(nota_aux,0,1);
                                flagb=0;
                                }
                                                        }
                                                  }
```

```
                        }
                                delay(20);
                    }

if(digitalRead(kmodo)==0){
            if(nota_new>=48){
               if (nota_new != nota_old){
                            MIDI.sendNoteOff(nota_old,5,1);
                            MIDI.sendNoteOn(nota_new,60,1);
                            nota_old = nota_new;
                                                    }
                        delay(20);
                    leitura_FSR();
                    if (velocity <= 25){ velocity = 25;}
                    if (velocity >= 127){ velocity = 127;}
                    MIDI.sendAfterTouch(velocity,1);
                         }
                    }
}
```