UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA

JOÃO RENI LISOT LICO

IMPLEMENTAÇÃO DE UM CONTROLADOR RÍTMICO AUTOMÁTICO PARA APARELHOS DMX

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2018

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA

JOÃO RENI LISOT LICO

IMPLEMENTAÇÃO DE UM CONTROLADOR RÍTIMICO AUTOMÁTICO PARA APARELHOS DMX

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2018

JOÃO RENI LISOT LICO

IMPLEMENTAÇÃO DE UM CONTROLADOR RÍTIMICO AUTOMÁTICO PARA APARELHOS DMX

Trabalho de conclusão de curso, apresentado à banca examinadora, do curso Superior de Engenharia Eletrônica do Departamento Acadêmico de Eletrônica - DAELN - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Me. Osmar Tormena Junior

CAMPO MOURÃO

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Campo Mourão
Coordenação do Curso de Engenharia Eletrônica

TERMO DE APROVAÇÃO
DO TRABALHO DE CONCLUSÃO DE CURSO INTITULADO

Implementação de um Controlador Rítmico Automático Para Aparelhos DMX

Por

João Reni Lisot Lico

Trabalho de Conclusão de Curso apresentado no dia 19 de novembro de 2018 ao Curso Superior de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. O Candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

Prof. Dr. Márcio Rodrigues da Cunha
Universidade Tecnológica Federal do Paraná

_____

Prof. Dr. Roberto Ribeiro Neli
Universidade Tecnológica Federal do Paraná

_____

Prof. Me. Osmar Tormena Júnior
Universidade Tecnológica Federal do Paraná
Orientador

A Folha de Aprovação assinada encontra-se na Coordenação do Curso

AGRADECIMENTOS

A minha mãe, Regina Mara Lisot, pela vida, orações, fé, sustento e pensamentos positivos.

Ao meu pai, Sgt. João Lico, pela vida, sustento, investimento e educação. Por me mostrar como agir, e principalmente, como não agir.

A minha parceira e mulher da minha vida, Milena Alves Correa Quaresma, que me motivou e me compreendeu em todas as etapas do processo de graduação. Que me fez entender que o futuro é feito a partir da constante dedicação no presente.

Ao meu Professor orientador Me. Osmar Tormena Júnior, pela ajuda ao contornar as barreiras do desenvolvimento e as várias correções necessárias.

A todos os professores do departamento acadêmico de Engenharia Eletrônica que dividiram comigo vosso conhecimento.

A todo o corpo docente e técnicos administrativos da Universidade Tecnológica Federal do Paraná, campus Campo Mourão, que possibilitaram o desenvolvimento desta grande instituição de ensino, na qual me orgulho de ter egressado.

Aos autores de todos os livros e artigos que lí, sites e repostas em fóruns. Acredito que, quem compartilha o conhecimento move a engrenagem do desenvolvimento da humanidade.

A todos os funcionários da educação em todos os níveis em que estive, hoje tenho a visão e a ciência de que apenas a educação pode levar um ser humano ao seu máximo potencial intelectual.

Aos meus companheiros e amigos de sala e movimento estudantil, pelo bom convívio, as boas discussões, as boas ideias, os compartilhamentos, e a alegria, que por vezes se instalava.

RESUMO

LICO, João Reni Lisot. IMPLEMENTAÇÃO DE UM CONTROLADOR RÍTMICO AUTOMÁTICO PARA APARELHOS DMX. Trabalho de Conclusão de Curso – Bacharelhado em Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Campo Mourão, 2018.

O presente trabalho de conclusão de curso consiste em implementar um sistema automático, especificamente para o controle de aparelhos de iluminação DMX-512A, analisando os principais sinais sonoros em tempo real. Tal sistema tem como objetivo reproduzir diversas sequências de efeitos pré-programados, de forma que haja uma imersão visual voltada para o público presente em eventos de apresentação musical. O sistema é composto por um bloco de processamento digital de sinais, que amostra, manipula e extrai informações pertinentes provindos do sinal de áudio. O DSP utilizado é o dsPIC33FJ12GP201. O segundo bloco é um microcontrolador que toma decisões e gera o sinal DMX-512A. Para a obtenção dos resultados almejados, é aplicado ao sinal amostrado filtros IIR passa-faixa. O trabalho aborda a geração dos coeficientes dos filtros dentro do software MATLAB®. Assim, baseado nos estudos realizados, percebeu-se a implementação de um código robusto, interagindo de forma eficaz com a maioria dos gêneros musicais testados, considerando que no caso estudado, foram controlados 10 luminárias LED DMX.

Palavras-chave: DMX-512A; Controle; Filtros Digitais; MATLAB®

ABSTRACT

LICO, João Reni Lisot. IMPLEMENTATION OF AN AUTOMATIC RHYTHM CONTROLLER FOR DMX SYSTEMS. Trabalho de Conclusão de Curso – Bacharelhado em Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Campo Mourão, 2018.

This term paper presents a implement an automatic system, specifically for the control of DMX-512A lighting systems, analyzing the sound signals in real time. Such a system aims to reproduce several sequences of preprogrammed effects, so that there is a visual immersion aimed at the audience present in musical presentation events. The system consists of a block of digital signal processing, which samples, manipulates and extracts pertinent information from the audio signal. The DSP used is dsPIC33FJ12GP201. The second block is a microcontroller that makes decisions and generates the signal DMX-512A. In order to obtain the desired results, the sampled signal is applied band-pass IIR filters. The work addresses the generation of filter coefficients within MATLAB® software. Thus, based on the studies carried out, it was possible to implement a robust code, interacting effectively with most of the musical genres tested, considering that in the case studied 10 LED DMX luminaires were controlled

Keywords: DMX-512A; Control; Digital Filters; MATLAB®

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ADC	*Analog to Digital Converter*	Conversor analógico para Digital
AMX	*Analogic Multiplex*	Multiplexador Analógico
ANSI	*American National Standards Institute*	Instituto Nacional Americano de Padrões
Art-Net	*Artistic Network*	Rede Artística
BPM	Batidas Por Minuto	
CI	Circuito Integrado	
CPU	*Central Processing Unit*	Unidade de Processamento Central
DMX	*Digital Multiplexer*	Multiplexador Digital
EIA	*Electronic Industries Alliance*	Aliança das Indústrias Eletrônicas
LED	*Ligth Eimitting Diode*	Diodo Emissor de luz
MIC	Microfone	
MIPS	*Millions of Instructions Per Second*	Milhões de Instruções por Segundo
RGB	*Red, Green and Blue*	Vermelho, Verde e Azul
RGBW	*Red, Green, Blue and White*	Vermelho, Verde, Azul e Branco
Séc.	Século	
TIA	*Telecommunications Industries Association*	Associação das Indústrias de Telecomunicações
ULA	Unidade Lógica Aritmética	
USITT	*United States Institute for Theatre Technology*	Instituto para Tecnologia Teatral dos Estados Unidos

SUMÁRIO

1 INTRODUÇÃO

A iluminação de um ambiente é um dos principais fatores para se criar sensações durante uma peça, concerto ou apresentação artística. Segundo o teórico suíço Adolphe Appia (1954).

> A luz é de uma flexibilidade quase milagrosa. Ela possui todos os graus de claridade, todas as mobilidades, pode criar sombras e irradiar no espaço a harmonia de suas vibrações, como o faria a música. (Apud Pavis, 2008).

E Pavis (2008) complementa.

> A luz intervém decisivamente no espetáculo. Ela não é simplesmente decorativa, mas participa da produção de sentido do espetáculo. Suas funções dramatúrgicas ou semiológicas são infinitas.

Observando a necessidade da indústria do entretenimento, foram desenvolvidas e aprimoradas diversas tecnologias, e recentemente, padrões internacionais para aparelhos de iluminação.

Os contínuos avanços tecnológicos têm como um de seus objetivos a redução de custos. Na indústria de controle para iluminação cênica não é diferente. Seguindo esta tendência, foram criados diversos sistemas para controlar tais aparelhos de iluminação, onde a grande maioria depende de um operador que esteja constantemente controlando os equipamentos. Da minoria restante encontra-se soluções semiautomáticas de custo elevado.

Dentro deste contexto, ao longo desta monografia, será apresentada uma solução para redução de custo operacional, trazendo elementos de amostragem de sinais sonoros, análise destes sinais em tempo real e o controle dos equipamentos de forma automática.

1.1 HISTÓRICO

Diversas técnicas de iluminação foram registradas, a história da iluminação aponta o teatro grego como o ponto de partida dos esforços humanos com esta questão. Estes teatros eram construídos a fim de maximizar o aproveitamento da luz natural para a iluminação do espetáculo. Durante a Idade Média (476 – 1492 d.C.), o

teatro estava quase que exclusivamente voltado para os dramas litúrgicos e a iluminação era favorecida pelos vitrais das igrejas (EDISON ELECTRIC ILLUMINATING COMPANY, 1929).

Já os séculos XVI e XVII, deram início à fabricação de algumas espécies de castiçais que continham ramificações e luminosidade em suas extremidades, passando a serem utilizados com maior frequência ao passar dos anos, e sofrendo diversas adaptações. Tais objetos, por serem enormes, possibilitavam a iluminação do palco e da plateia (CASSOU, 2011).

Em meados do século XVIII, o sebo acabou sendo manipulado para confecção de velas, no entanto, a ideia não prosperou, pois, os espectadores e atores ao exalarem o mau cheiro, apresentavam irritações em seus olhos e vias respiratórias. No ano de 1783, utilizando-se de lampião a óleo, Ami Argand, inventou um mecanismo menos nocivo à saúde, denominando-os de “lampiões Argand”. Adiante, Bernard Carcel, produziu o “lampião Astral Francês”, o qual produzia uma luz mais contínua (EDISON ELECTRIC ILLUMINATING COMPANY, 1929).

No desfecho do século XVIII, a atenção voltou-se à posição das fontes de luz, surgindo, a partir daí as primeiras noções de arandelas, ribalta, luzes laterais e contraluzes (CASSOU, 2011).

A partir do século XIX o gás começa a ser utilizado como combustível para os sistemas de iluminação. Os primeiros registros de sua utilização em teatros datam de 1803, mais especificamente no *Lyceum Theatre* de Londres. Neste mesmo período aparecem as primeiras mesas de controle de luz. Em 1879, Thomas Alva Edson fabrica as primeiras lâmpadas de incandescência com filamento de carbono (WILLIAMS, 1999; CASSOU, 2011).

No início do séc. XX é criado o primeiro *dimmer* fabricado a partir de um balde com água, sal e dois eletrodos, aumentando a intensidade da luz com a aproximação dos eletrodos e sua redução com o afastamento. Em 1911 foi desenvolvido por William D. Coolidge, nos laboratórios da *General Electric*, a primeira lâmpada comercial de tungstênio, com alta durabilidade e resistência a impactos (WHELAN, 2016).

Em 1958 foi anunciado, também pela *General Electric*, a introdução do primeiro SCR, dispositivo de silício que permite interromper a corrente de forma veloz, e permitiu controlar eletronicamente a intensidade das lâmpadas (BURGESS, 2011).

Na década de 60, em mãos da nova tecnologia, diversos fabricantes começaram a criar dispositivos de iluminação dedicados a apresentações. Cada fabricante fazia aparelhos com seus próprios padrões, limitando os consumidores a adquirir uma gama de produtos de um mesmo fabricante e elevando o custo. Vale ressaltar também que estes equipamentos necessitavam de um cabo por canal, tornando-o muito custoso para grandes sistemas (WILLIAMS, 1999).

Com a evolução da eletrônica, em 1975 foi criado o padrão AMX192 (AMX – *Analogic MultipleX*), um multiplexador analógico capaz de controlar 192 canais dimerizáveis com apenas um par trançado de fios (WILLIAMS, 1999; LINS, 2006).

Em 1986 com a eletrônica digital a todo vapor, foi criado o padrão DMX512 (DMX – Digital MultipleX), protocolo que multiplexa digitalmente 512 canais com uma resolução de 8 bits por canal, gerando 256 valores de intensidade para cada um (IRWIN, 2018).

1.2 APARELHOS

No mercado atualmente existem diversos aparelhos dedicados à iluminação cênica, e seu controle é feito de forma mestre-escravo, sendo o mestre uma mesa controladora ou um computador, e os escravos são os dispositivos conectados à rede que serão controlados.

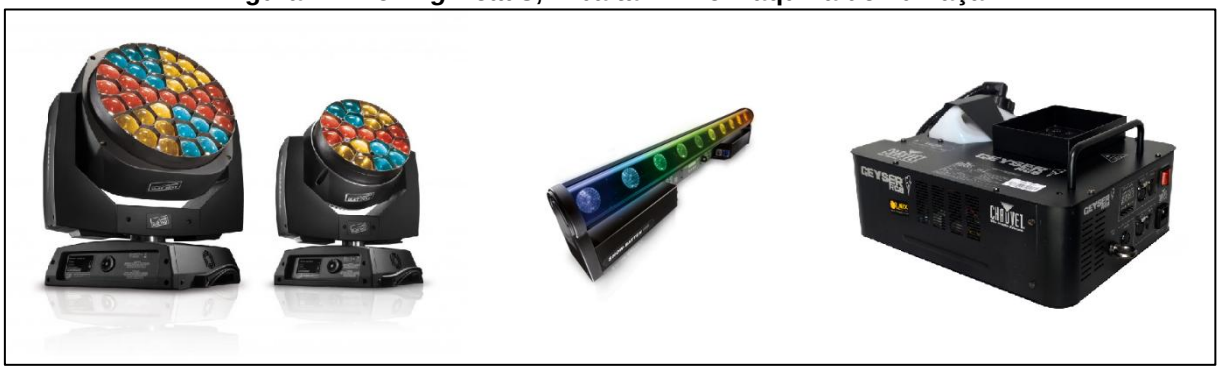
Na Figura 1 é mostrado alguns consoles da Avolites operando a iluminação de um concerto.

Figura 1 – Consoles da *Avolites* realizando o controle de diversos equipamentos.

Fonte: PLSN, 2018.

Alguns aparelhos que trabalham como escravos podem ser vistos na Figura 2. Nesta, à esquerda são mostrados dois Moving Heads modelo B-EYE K20 e B-EYE K10 da marca Clay Parky, ao centro há uma Ribalta de LED modelo Show-Batten 100 também da marca Clay Parky, e finalizando, à direita, uma máquina de fumaça modelo Geyser RGB da fabricante Chauvet.

Figura 2 – *Moving Heads*, Ribalta LED e Máquina de Fumaça.



Fonte: Clay Parky, 2018 e Chauvet, 2018.

Estes aparelhos citados acima são controlados através do padrão DMX-512A, o qual será amplamente especificado na Seção 3.4. Por hora deve-se ter em mente que um dispositivo pode utilizar vários canais para seu controle, por exemplo, um *Moving Head* antigo utiliza dois canais para movimentação vertical, dois canais para a movimentação horizontal, três canais para a intensidade das três cores, vermelho, verde e azul, um canal para configuração e outro para intensidade geral, ou seja, um aparelho apenas consome nove canais dos 512 disponíveis para esta rede.

Já o moderno B-EYE K20 da Clay Parky possui 37 LED's RGBW endereçáveis independentemente, neste modo, um único dispositivo é capaz de utilizar 169 canais, o que indica que uma rede pode ser consumida com apenas três aparelhos deste.

Cada rede desta é denominada como universo. Cada universo contém 512 canais. A tecnologia mais avançada para o controle em grande escala de dispositivos se chama Art-Net (MAGALHÃES, 2014).

1.3 ART-NET

Art-Net é uma abreviatura para *Artistic Network*, este é uma solução desenvolvida para sanar a falta de capacidade do DMX-512A, onde este utiliza como meio de transmissão a rede Ethernet. Para isto existem aparelhos que fazem esta conversão. É necessário um aparelho para cada universo.

A Art-Net em sua mais recente versão é capaz de condensar $2^{15} = 32768$ universos pela rede Ethernet, isto resulta em impressionantes 16.731.136 canais (MAGALHÃES, 2014).

1.4 PROBLEMA

Eliminar a necessidade de um operador que fique alterando manualmente as cenas, fazendo que o fabricante apenas faça uma configuração prévia e que o aparelho, com base nas predefinições e ritmo sonoro, faça o controle dos dispositivos sob seu comando.

1.5 OBJETIVOS

1.5.1 Objetivo Geral

O presente trabalho tem como propósito implementar um controlador de aparelhos de iluminação que utilizam o protocolo DMX-512A, que este controlador obtenha o sinal de áudio e controle os aparelhos supracitados a partir das propriedades do áudio amostrado.

1.5.2 Objetivos Específicos

Nesta subseção apresenta-se os objetivos específicos para o presente trabalho.

- Desenvolver um hardware baseado no microcontrolador dsPIC33FJ12GP201, contendo aparato analógico para suprir o filtro *antialiasing*, bem como o circuito necessário para realizar a comunicação no padrão ANSI/TIA/EIA-485-A com os demais aparelhos da rede.
- Aplicar filtros, transformadas e processos com baixo custo computacional, a fim de detectar o ritmo e sua periodicidade.
- Implementar algoritmos de extração das propriedades do sinal em tempo real.
- Criar tomadas de decisões com base nos dados extraídos do sinal sonoro proveniente de música eletrônica.
- Implementar o protocolo DMX-512A para controlar dispositivos de iluminação comerciais.

2 PROCEDIMENTOS METODOLÓGICOS

Para o êxito no desenvolvimento do trabalho, se faz necessário um método para checar se a resposta de determinada função está de acordo com o planejado, ou não.

Este método pode ser muito volátil devido à variedade de operações exercidas no processo de desenvolvimento. Contudo, a essência do método é a verificação, por meio de instrumentos, dos resultados e respostas às entradas conhecidas do sistema.

Com base nos objetivos traçados, tem-se uma ideia do que seja necessário para validar tais objetivos como concluídos.

Como trata-se de desenvolvimento em sistemas microcontrolados, na maioria dos casos, é comum a utilização de dispositivos que efetuem a programação do arquivo fonte compilado no computador para dentro a memória do dispositivo. Será utilizado o programador PICkit v3. Para compilar os fontes, será utilizado o compilador Microchip XC16 juntamente com a IDE MPLAB X.

Para a visualização e validação da parte analógica do circuito, será utilizado um osciloscópio de 2 canais com máxima frequência de 60 MHz.

Para a visualização e validação da parte digital do circuito, o aparelho utilizado é um analisador lógico de 8 canais, com frequência máxima de aquisição de 24 MHz.

Para a validação dos filtros digitais, é necessário um gerador de sinal analógico e um método para se obter digitalmente o sinal de saída do filtro. Com intuito de solucionar esse item, o analisador lógico possui um decodificador de protocolo, logo, o microcontrolador a ser analisado transmite por serial assíncrona e o analisador decodifica e exporta os dados, que posteriormente serão transformados em gráficos pelo software MATLAB.

Em posse de tais equipamentos, inicia-se uma rotina de pesquisa, teste e erro, até obter-se o resultado almejado.

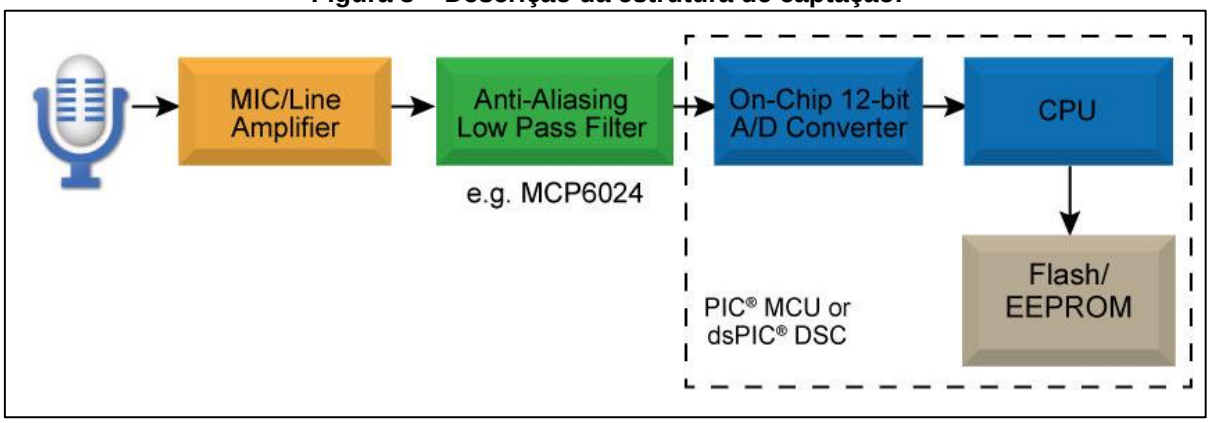
3 FUNDAMENTAÇÃO TEÓRICA

Esta seção traz informações fundamentais para o desenvolvimento do projeto. O enfoque principal se dá em torno do processamento digital do sinal para o reconhecimento de padrões na música eletrônica e da geração do sinal DMX-512A.

3.1 AQUISIÇÃO DO SINAL

O sinal de interesse é o sinal de áudio proveniente do espetáculo, este sinal será retirado da mesa de som e então será condicionado para que seja amostrado e posteriormente processado. Na Figura 3 é exibida uma prévia da estrutura de captação.

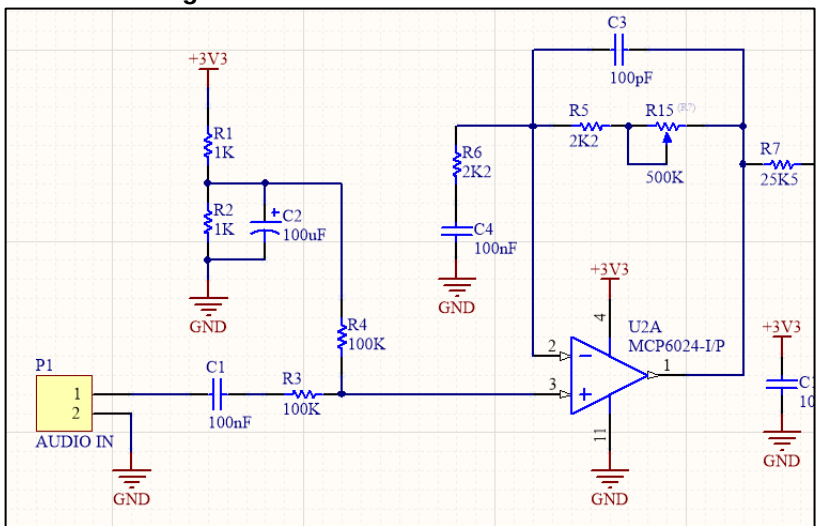
Figura 3 – Descrição da estrutura de captação.



Fonte: MICROCHIP, 2018.

É de amplo conhecimento que o sinal de áudio excursiona entre tensões positivas e negativas, então a primeira parte do circuito presente na Figura 4, composta de R1 a R4, C1 e C2, desacopla o sinal da entrada e soma uma tensão de 1,65 V, fazendo com que o sinal não tenha um valor abaixo de zero. Também deve-se amplificar ou atenuar o sinal até chegar a níveis adequados para que possa ser amostrado corretamente. Para isto será implementado um amplificador operacional com um ajuste de ganho manual, através do potenciômetro R15 que pode ser visto a seguir.

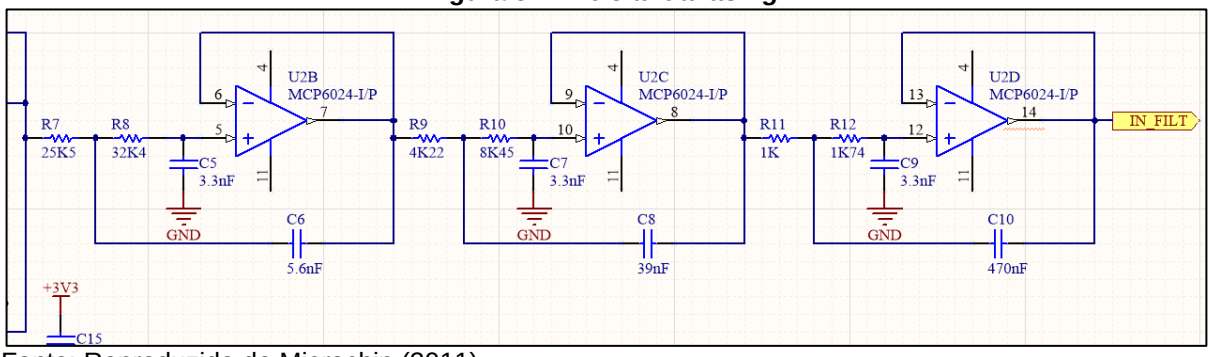
Figura 4 – Circuito de entrada de áudio.



Fonte: Reproduzido de Microchip (2011).

Ainda sobre a aquisição do sinal, é preciso garantir que não esteja presente nenhuma frequência acima da metade da frequência de amostragem. Com isto, será aplicado na saída do controle de ganho um filtro passa-baixa, calculado para uma frequência de corte a 20,5 kHz, conhecido como filtro *antialiasing*, que pode ser visto na Figura 5.

Figura 5 – Filtro *antialiasing*.



Fonte: Reproduzido de Microchip (2011).

3.2 AMOSTRAGEM

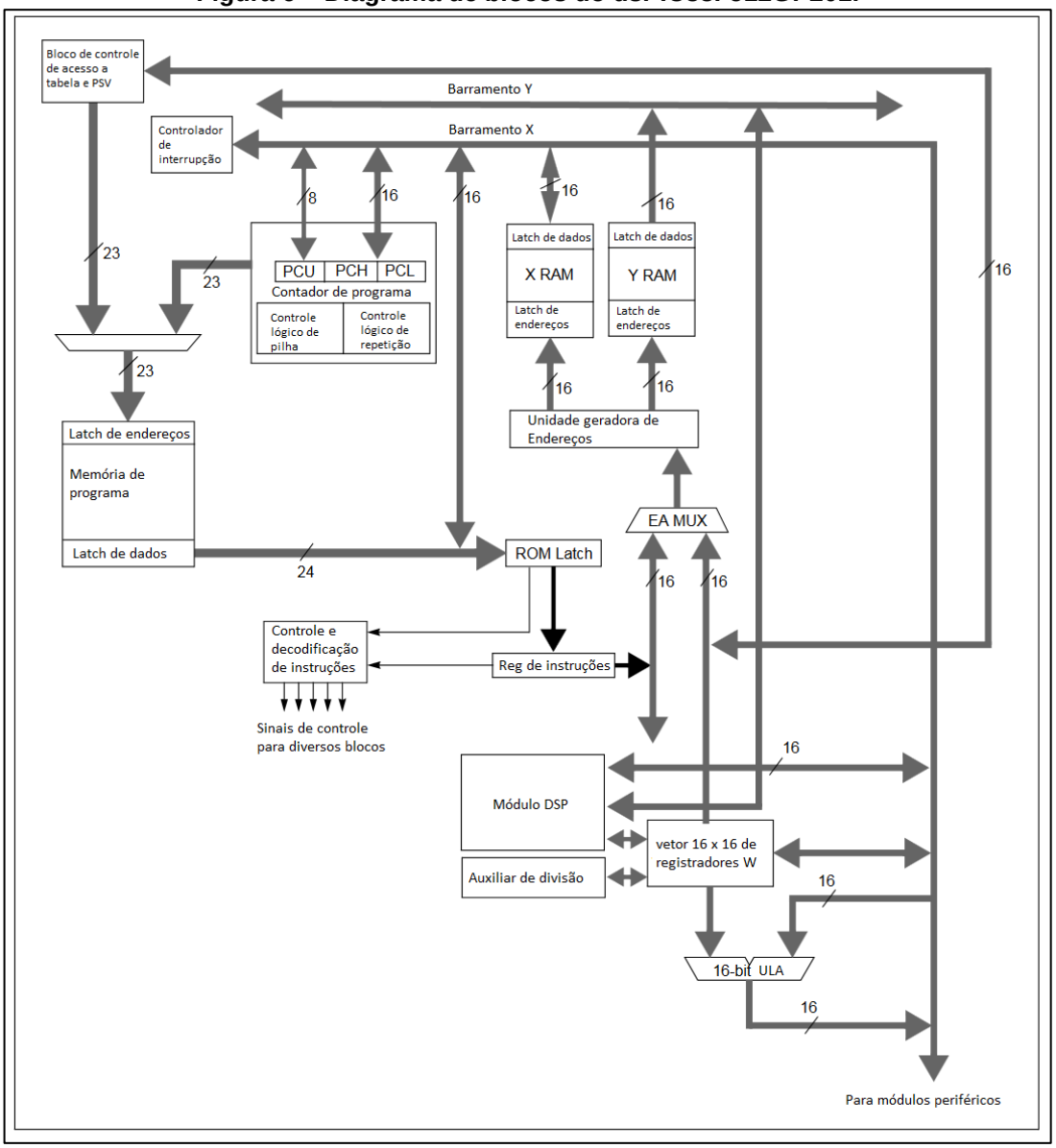
A amostragem será feita pelo conversor ADC interno do microcontrolador, de acordo com a Figura 3. O microcontrolador escolhido para desempenhar tal função foi o dsPIC33FJ12GP201 que pode chegar a frequência de amostragem de até 500 kHz em 12 bits. Entretanto será amostrado o sinal de áudio a uma frequência de 44 kHz (MICROCHIP, 2011).

3.3 PROCESSAMENTO DIGITAL DO SINAL

Para realizar o processamento digital do sinal amostrado, o microcontrolador escolhido tem a capacidade de processar até 40 MIPS (Mega Instruções Por Segundo). Também é relevante informar que este microcontrolador possui uma arquitetura do tipo *Harvard* modificada, trabalhando a 16 bits, aliado a uma ULA preparada para o processamento digital, onde a maioria de suas instruções se executam em apenas um ciclo de máquina (MICROCHIP, 2011).

Internamente o dispositivo possui 1024 bytes de memória RAM e 12 kB de memória de programa (MICROCHIP, 2011).

Figura 6 – Diagrama de blocos do dsPIC33FJ12GP201.



Fonte: Adaptado de Microchip (2011).

3.3.1 Percepção Rítmica

Seguindo o raciocínio de Nunes (2014), o ritmo é responsável pela estrutura temporal da música, o que nos traz a sensação de pulsação, de repetibilidade dentre os sons que compõem a melodia escutada.

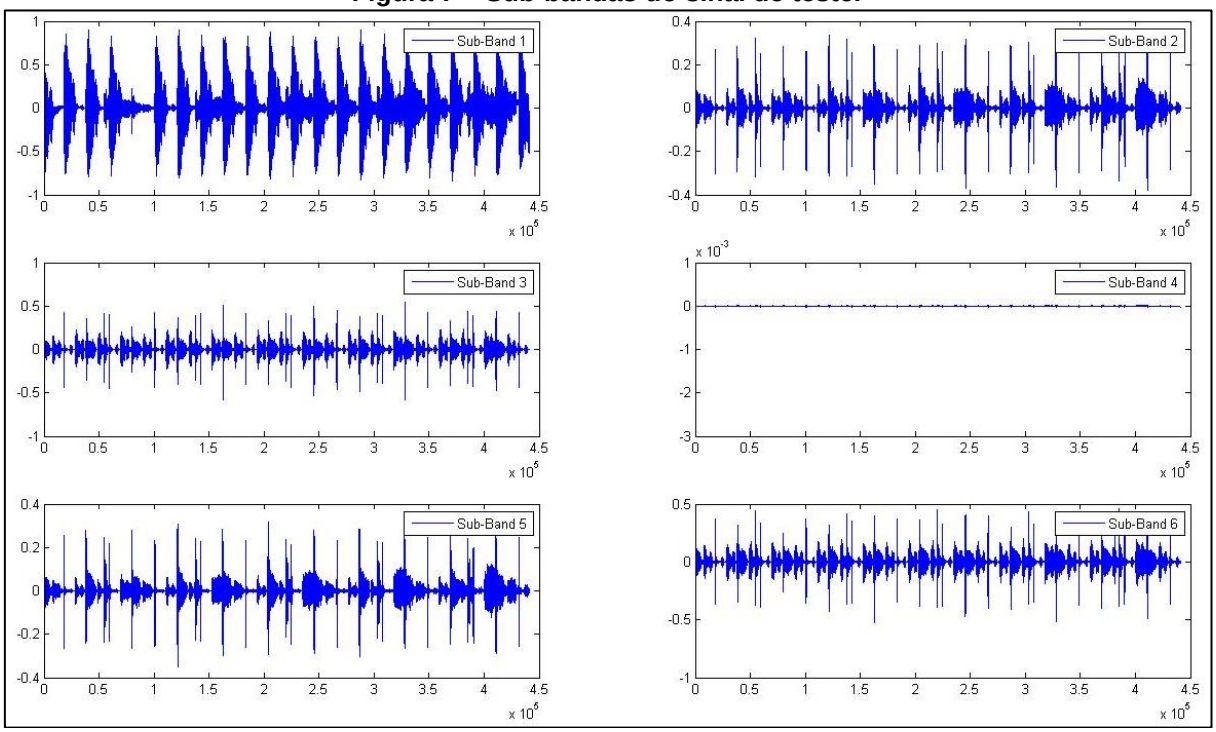
Pode-se classificar as músicas com base na periodicidade destas batidas, o que nos remete ao termo Batidas Por Minuto, ou BPM, que é o número de pulsações que a melodia executa no intervalo de 60 segundos. Este número é de suma importância para a tomada de decisões a ser implementada.

3.3.2 Extração de Atributos

Para conseguir extrair os picos de energia, Souza (2012) descreve que deve-se dividir o sinal em seis sub-bandas, aplicando sobre o sinal original, filtros passa-faixa através da Transformada Discreta de *Wavelet*. As seis frequências escolhidas por Souza foram 200 Hz, 400 Hz, 800 Hz, 1600 Hz, 3200 Hz, 6400 Hz.

Com o áudio de teste, chegou-se ao resultado contido na Figura 7.

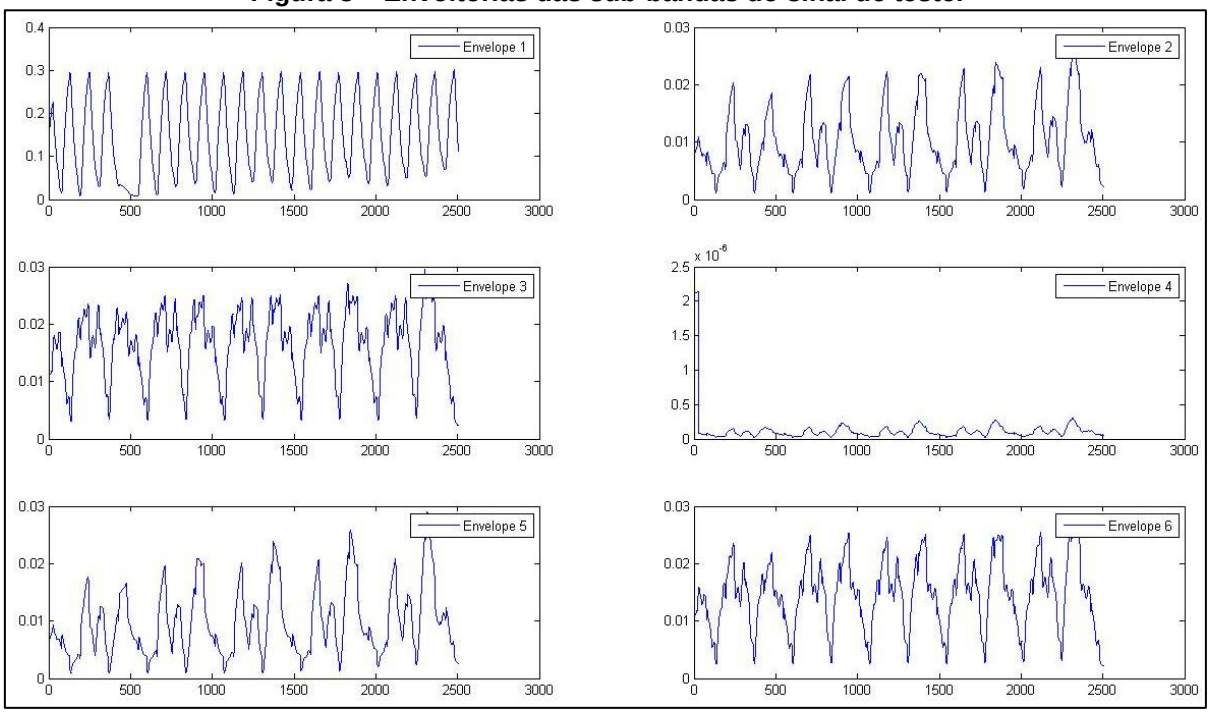
Figura 7 – Sub-bandas do sinal de teste.



Fonte: Souza (2012).

A partir destas subdivisões, é aplicado um algoritmo para extrair a envoltória destes. Este algoritmo possui algumas etapas, a primeira delas é a retificação por completo da onda, transformando todos os valores negativos em positivos. O segundo passo é aplicar um filtro passa baixa no sinal. Como terceiro passo o sinal passa por um *downsample*, ele é re-amostrado com uma frequência menor. O quarto e último passo é uma convolução com um filtro designado para remover o sinal médio. Portanto, a Figura 8 exibe os resultados.

Figura 8 – Envoltórias das sub-bandas do sinal de teste.



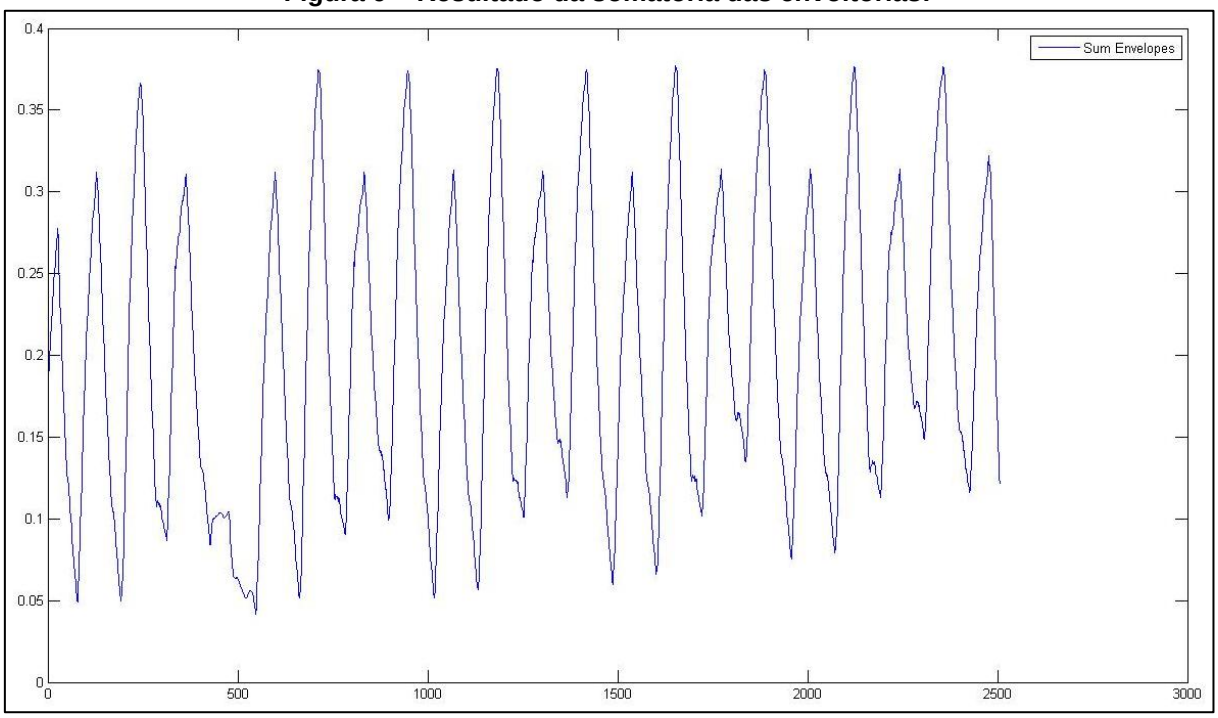
Fonte: Souza (2012).

Em posse da envoltória das seis bandas, é feito uma somatória com estes dados, que nos resulta claramente onde está localizado as batidas no tempo.

A partir deste resultado, aplica-se um algoritmo de auto correlação que resultará na diferença de tempo entre as batidas, e então prever sua periodicidade.

Outra técnica para obter a periodicidade é medir diretamente o tempo entre os picos, armazenar os 10 últimos tempos, e fazer uma média com estes valores. O inverso do resultado da média é uma aproximação da repetição de picos por segundo.

Figura 9 – Resultado da somatória das envoltórias.



Fonte: Souza (2012).

3.3.3 Implementação do Algoritmo em Tempo Real

Os algoritmos acima foram desenvolvidos para trabalhar no software MATLAB, com arquivos de áudio já conhecidos, não em tempo real como pretende-se implementar neste projeto. Estes algoritmos citados servirão como base para a criação de um algoritmo em tempo real.

3.4 GERAÇÃO DO SINAL DMX

O protocolo original DMX-512 foi desenvolvido por uma comissão de engenharia do Instituto para Tecnologia Teatral dos Estados Unidos (*United States Institute for Theatre Technology* - USITT) em 1986, passando por diversas revisões até chegar a sua versão atual, padronizada pela Organização Nacional Americana de Padrões (*American National Standards Institute* - ANSI), denominada como "E1.11 – 2008, USITT DMX512-A", ou apenas "DMX512-A"

Segundo este documento emitido pela ANSI, a comunicação entre os dispositivos se dá pelas camadas física e virtual.

3.4.1 Camada Física

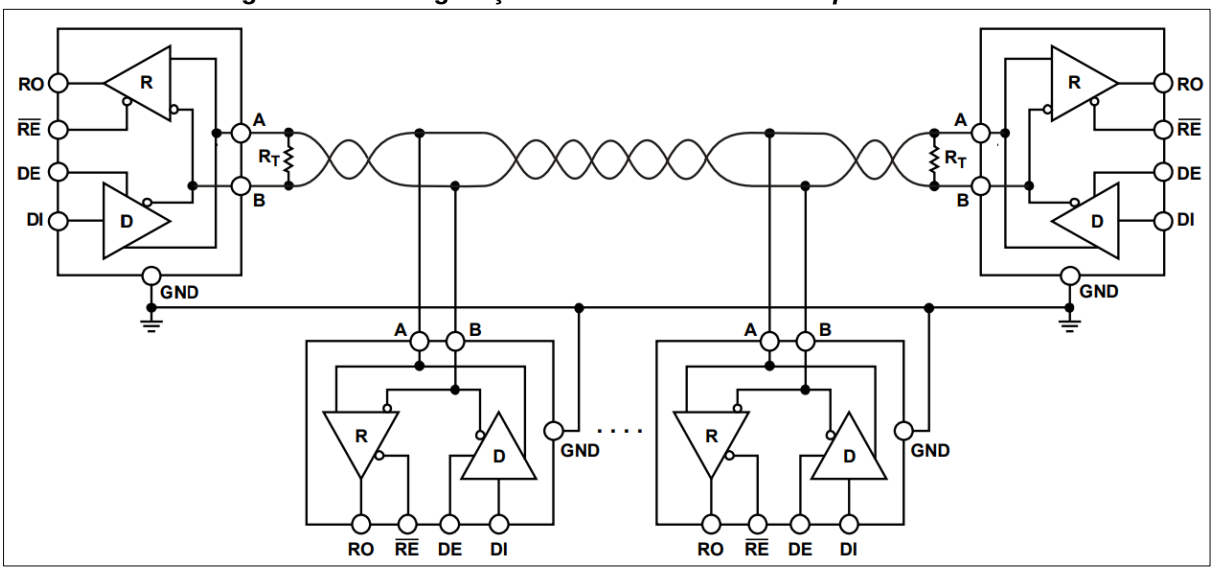
As características elétricas da comunicação DMX-512A são regidas pela norma ANSI/TIA/EIA-485-A-1998, que define um padrão para transmissores e receptores ligados a sistemas multiponto utilizando sinal balanceado digital.

Esta norma define os padrões de tensões, isolamento elétrico, cabeamentos e topologias de ligação entre dispositivos em uma mesma rede.

O DMX-512A utiliza o método *Half-Duplex*, o que permite economizar no cabeamento ao custo de não haver comunicação bidirecional simultânea. Este padrão também utiliza a topologia *Daisy Chain*, pois evita falhas de comunicação.

Assim, pode-se realizar a conexão dos dispositivos conforme demonstra Marais (2008), na Figura 10.

Figura 10 – Configuração do barramento *Half-Duplex* RS-485.

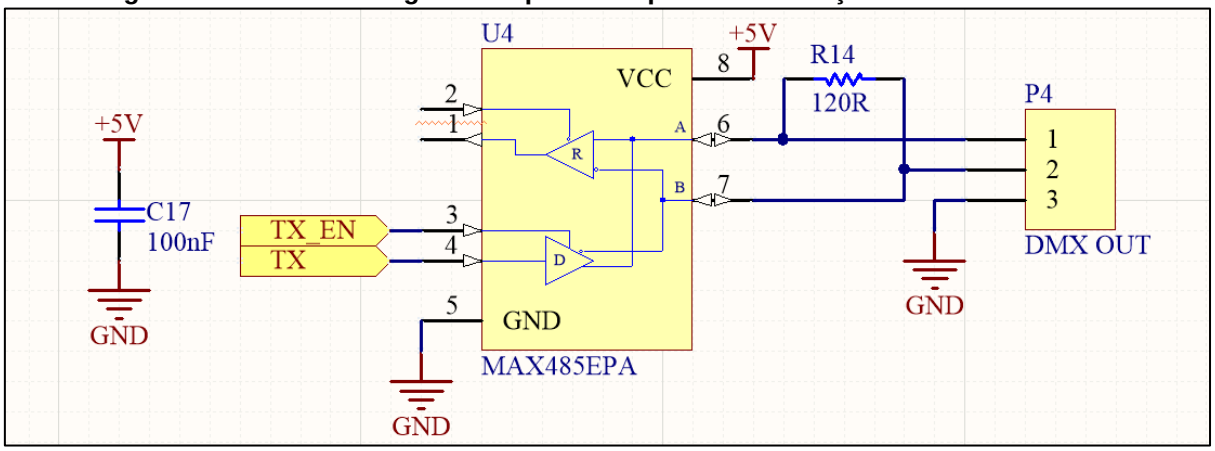


Fonte: Marais, (2008).

Observa-se que existem dois resistores de terminação (R_T), estes possuem valor de 120 ohms, sendo que, o primeiro deve ser posto no transmissor, e o segundo no último equipamento a ser conectado na rede. Este resistor tem a finalidade de casar a impedância da rede e evitar que haja reflexão dos *frames.*

Aplicando os conceitos, no esquema a ser prototipado, utiliza-se o CI MAX485 para a conversão dos níveis lógicos. Pode-se acompanhar isto pela Figura 11.

Figura 11 – Circuito Integrado responsável pela comunicação ANSI/TIA/EIA-485.



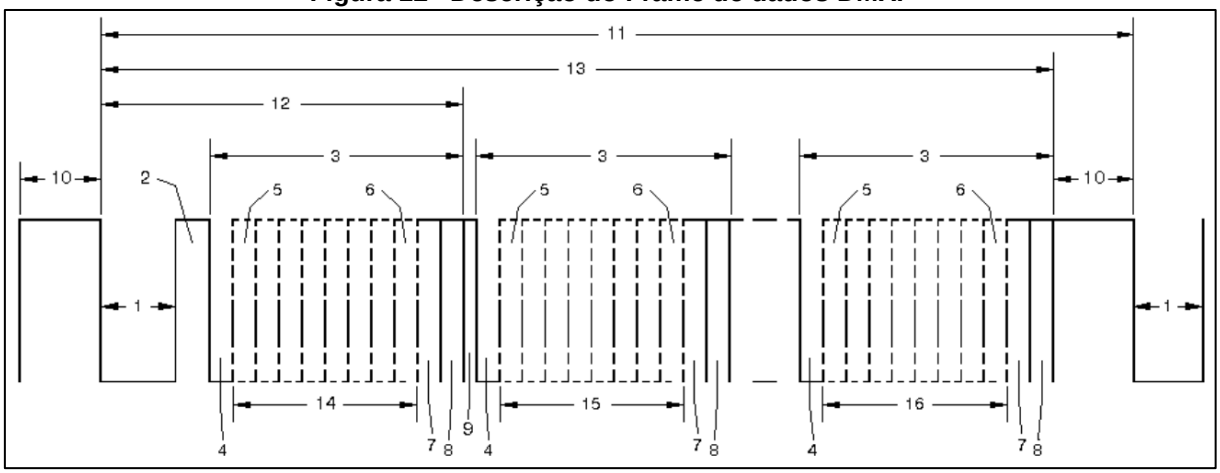
Fonte: Autoria Própria.

Os sinais elétricos providos nas saídas A e B pelo MAX485, irão percorrer toda a corrente de dispositivos conectados a ele. Por questões de estabilidade, o padrão RS-485 limita o uso de no máximo 32 aparelhos por transmissor.

3.4.2 Camada Virtual

Esta camada é responsável por codificar digitalmente os dados a serem transmitidos pela rede RS-485 descrita anteriormente. De acordo com Luna e Torres (2006) a implementação deste protocolo se dá por uma comunicação assíncrona serial de 8 bits, somado com algumas marcações de tempos para informar os receptores qual é o início da comunicação.

Figura 12 –Descrição do Frame de dados DMX.



Fonte: Luna e Torres (2006)

Tabela 1 – Descrição dos itens da Figura 12.

Numero	Descrição	Min	Max	Unidade
1	*“SPACE” for BREAK*	92	180	µs
2	*“MARK” After BREAK (MAB)*	0,012	1000	ms
3	*Slot Time*	43,12	44,88	µs
4	*START Bit*	3,92	4,08	µs
5	*LEAST SIGNIFICANT Data BIT*		4,08	µs
6	*MOST SIGNIFICANT Data BIT*		4,08	µs
7	*STOP Bit*		4,08	µs
8	*STOP Bit*		4,08	µs
9	*“MARK” Time Between slots*	0	1000	ms
10	*“MARK” Before BREAK (MBB)*	0	1000	ms
11	*BREAK to BREAK Time*	1,204	1000	ms
12	*RESET Sequence (BREAK, MAB, START Code)*			
13	*DMX512 Packet*	1,204	1000	ms
14	*START CODE (SLOT 0 Data)*			
15	*SLOT 1 DATA*			
16	*SLOT n DATA (*Máximo *512)*			

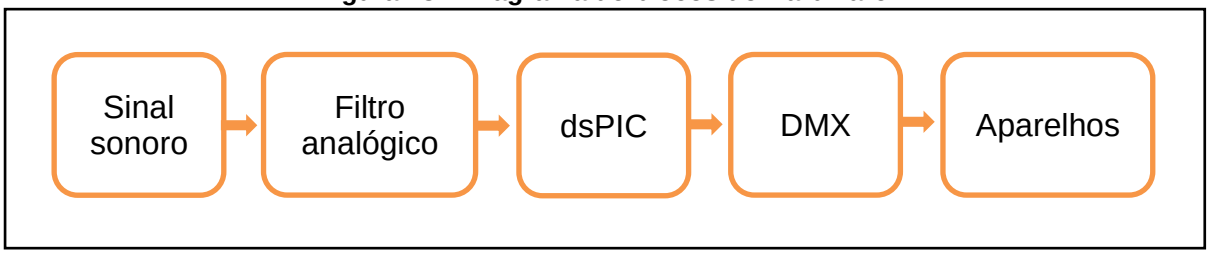
Fonte: ESTA, (2013), Luna e Torres (2006).

Será implementado na programação um código que faça essa transmissão e respeite todos os itens descritos na norma.

4 IMPLEMENTAÇÃO DO *HARDWARE*

A Figura 13 apresenta um diagrama de blocos que sintetiza o fluxo das informações da entrada até a saída.

Figura 13 – Diagrama de blocos do Hardware.



Fonte: Autoria Própria.

4.1 ALIMENTAÇÃO DOS PERIFÉRICOS

Na Figura 14 observa-se a entrada de tensão, que pode variar de 9 V a 25 V em valores positivos. O diodo D1 tem a finalidade de evitar a queima do circuito por polarização invertida na entrada.

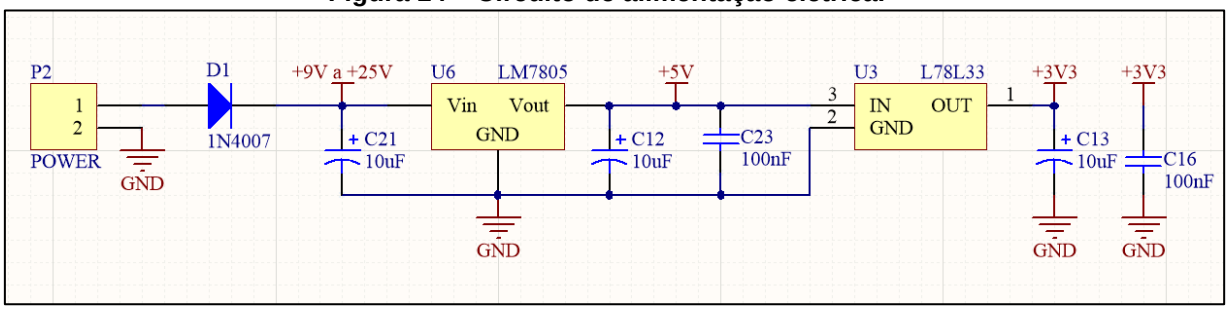
O circuito conversor do sinal para RS485, o microcontrolador que gera este sinal e o filtro analógico utilizam a tensão de 5 V, que é obtida a partir da saída do regulador de tensão LM7805. O microcontrolador que faz o processamento digital necessita da tensão de 3,3 V para seu funcionamento, que é provida pelo regulador L78L33.

Como o sinal de áudio, por padrão, excursiona entre valores positivos e negativos de tensão, é comum utilizar fontes de tensão simétricas em relação ao GND, contudo, no circuito da seção 3.2, é descrito a alternativa escolhida a fim de evitar a necessidade da tensão negativa.

A corrente consumida medida para os periféricos em 5 V foi de 68,3 mA, em 3,3 V o resultado foi 47 mA.

Os valores dos capacitores presentes na Figura 14 foram retirados das folhas de dados disponibilizadas pelos fabricantes dos reguladores de tensão escolhidos.

Figura 14 – Circuito de alimentação elétrica.



Fonte: Autoria Própria.

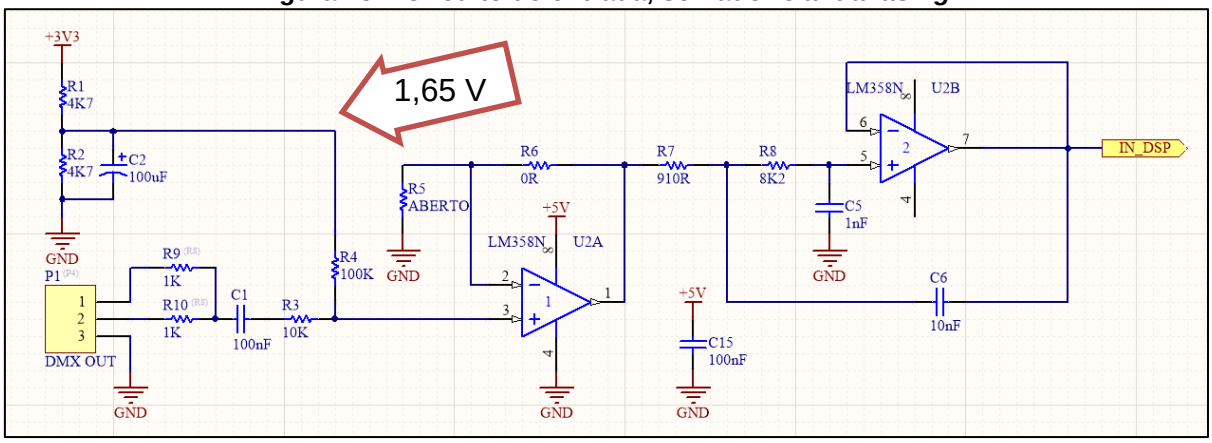
4.2 CIRCUITO SOMADOR

No processamento digital do sinal, pode-se facilmente deslocar o sinal. Logo, desloca-se o sinal analogicamente, somando 1,65 V ao sinal de entrada. Isto pode ser visto na Figura 15, com os componentes R1, R2, R3 e R4.

Outra soma pode ser encontrada na Figura 15. Os resistores R9 e R10 somam os sinais do som estereofônico, e seu resultado será deslocado 1,65 V positivamente.

Este circuito prevê um ganho para o sinal, que por hora está unitário.

Figura 15 – Circuito de entrada, somador e *antialiasing*.



Fonte: Autoria Própria.

4.3 FILTRO *ANTIALIASING*

Ainda na Figura 15, existe um filtro *chebyshev* passa baixa de segunda ordem, com a topologia *Sallen-Key*, destinado a ceifar frequências superiores a 20

kHz. Composto pelos componentes R7, R8, C5, C6 e U2B, este filtro é o último ponto analógico a ser tratado neste trabalho, sua saída tem como destino uma entrada analógica do dsPIC.

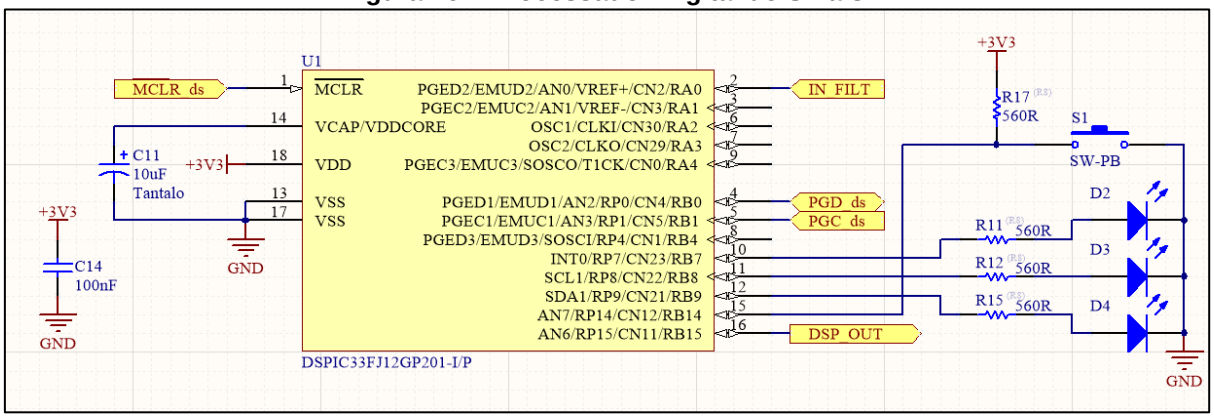
Os valores de R7, R8, C5 e C6 foram obtidos através do software Filter Pro, da fabricante de CIs *Texas Instruments*.

4.4 PROCESSADOR DIGITAL DE SINAIS

O circuito presente na Figura 16 é composto pelo dsPIC33Fj12GP201, que executará os filtros que serão definidos na seção 5.3, três LED's e seus resistores, um botão com seu *pull-up*, o capacitor de tântalo para dar estabilidade ao regulador de 2,5 V interno do dsPIC, e as conexões de alimentação, entrada e saída do microcontrolador.

Os LED's servem para avisar ao usuário se o nível do sinal que está entrando está satisfatório ou não. O botão zera os parâmetros de auto nivelamento digital do sinal.

Figura 16 – Processador Digital de Sinais.



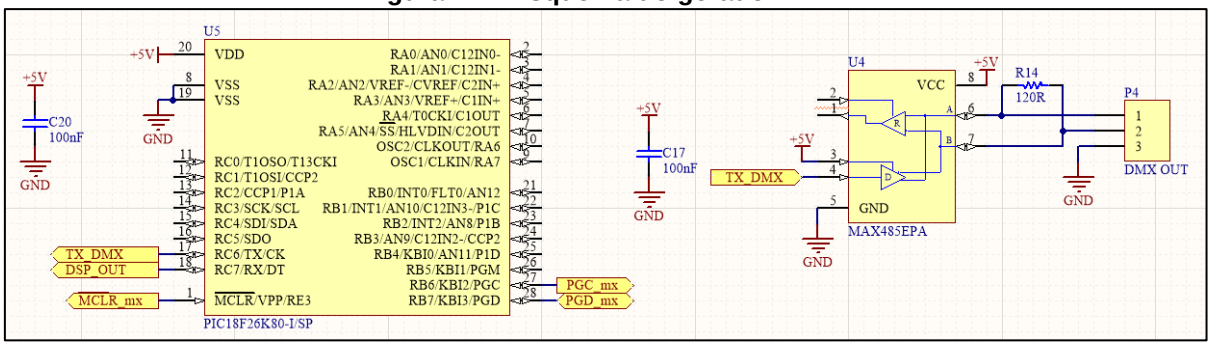
Fonte: Autoria Própria.

4.5 GERADOR DE SINAL DMX

Durante a execução do projeto, evidenciou-se que apenas o dsPIC tem capacidade de processamento suficiente para executar o processamento digital dos sinais em conjunto com a geração do sinal DMX, porém, não tem memória suficiente para os macros, e o fato do processamento dos sinais ocuparem o processador por muito tempo pode gerar gargalo na execução das tarefas.

A saída é adicionar outro microcontrolador, o qual revela-se na Figura 16, PIC18F26K80, este conta com uma memória de 64 kB de ROM e 3,6 kB de RAM, contra 12 kB de ROM e 1 kB de RAM do dsPIC.

Figura 17 – Esquema do gerador DMX.



Fonte: Autoria Própria.

Desta forma, garante-se que há recursos físicos suficientes para atender os objetivos especificados na Seção 1.5.

5 IMPLEMENTAÇÃO DOS FILTROS DIGITAIS

A implementação dos filtros digitais em processadores de sinais normalmente é feita a partir de funções desenvolvidas pelos programadores dos compiladores utilizados.

Para se utilizar destas funções, faz-se necessária a leitura da documentação do compilador. Neste caso, a documentação sugere que o usuário deve utilizar o software dsPIC Filter Designer para gerar coeficientes de filtros, com o atual preço de $29,90 dólares americanos, anteriormente (antes de 2016) era $249,00.

Durante a pesquisa sobre este tema, encontra-se diversas pessoas presenciando o mesmo problema. Logo, foi desenvolvido um método para geração dos coeficientes e posterior aplicação dos filtros.

Dentre os filtros digitais pesquisados, o IIR e o FIR se mostram mais eficientes, pois mesmo tendo ordem de complexidade O(N^2), tem o valor N muito menor frente a FFT e DWT, que compartilham o valor O(N.log(N)).

5.1 FILTRO IIR NO dsPIC

As funções existentes no compilador para a aplicação de filtros digitais são otimizadas para a melhor utilização dos recursos de *hardware* possível. Dentro destas funções, encontra-se 3 maneiras de executar um filtro IIR, que são:

- IIR Transposta.
- IIR Lattice.
- IIR Canonica.

A técnica escolhida com base no maior número de informações encontradas foi a IIR transposta.

A documentação do compilador traz a informação de que a função *IIRTransposed* aplica um filtro IIR utilizando uma cascata de seções de segunda ordem, cada seção é transposta utilizando a forma direta II.

5.2 MATLAB GERANDO FILTROS E COEFICIENTES

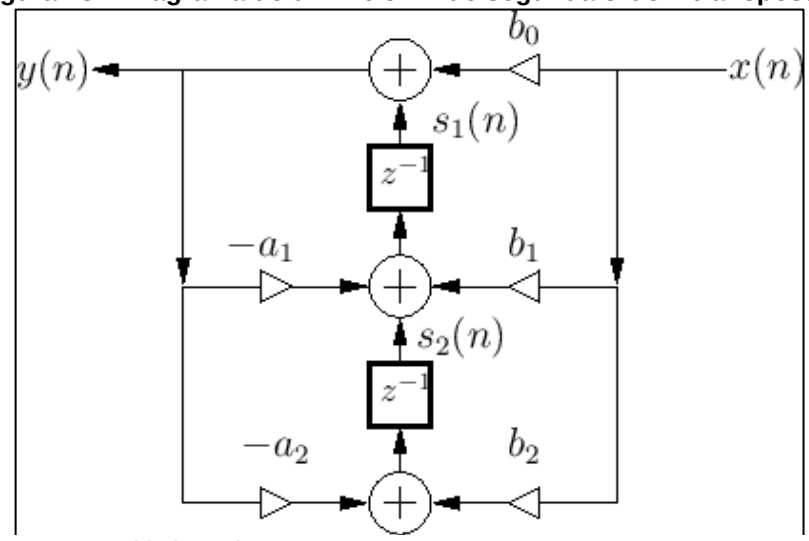
Para o projeto de qualquer filtro, se faz necessário métodos matemáticos para o cálculo dos coeficientes. Nesse seguimento, atualmente conta-se com uma potente ferramenta para manipulações matemáticas, o software MATLAB.

Dentro do MATLAB existem funções destinadas à criação de filtros em diversas respostas conhecidas, como *Butterworth*, *Chebyshev*, *Bessel*, elíptico, entre outros, cada um com seus prós e contras.

Devido à ausência de necessidade de uma resposta linear em fase do filtro, e a prioridade por uma ordem menor com uma queda mais acentuada possível entre a banda passante e de rejeição, é escolhida a aproximação elíptica para a realização desta etapa.

Logo, no código implementado no MATLAB, é utilizado a função *ellipord(...)*, que recebe os parâmetros da banda passante e de rejeição, retornando a ordem do filtro. A etapa seguinte é utilizar a função *ellip(...)* que recebe a ordem especificada pela função *ellipord(...)* e os demais parâmetros de frequência, *ripple* e tipo do filtro, como retorno, obtém-se os vetores de zeros e polos e o ganho k referente ao filtro projetado. Em seguida utiliza-se a função *zp2sos(...)*, inserindo os parâmetros de zeros, polos e ganho obtidos previamente, a saída e uma matriz de coeficientes separadas em seções de segunda ordem transpostas, já na forma direta II. Na Figura 18 observa-se a disposição dos coeficientes gerados pelo MATLAB e sua aplicação no sinal de entrada, gerando o sinal de saída.

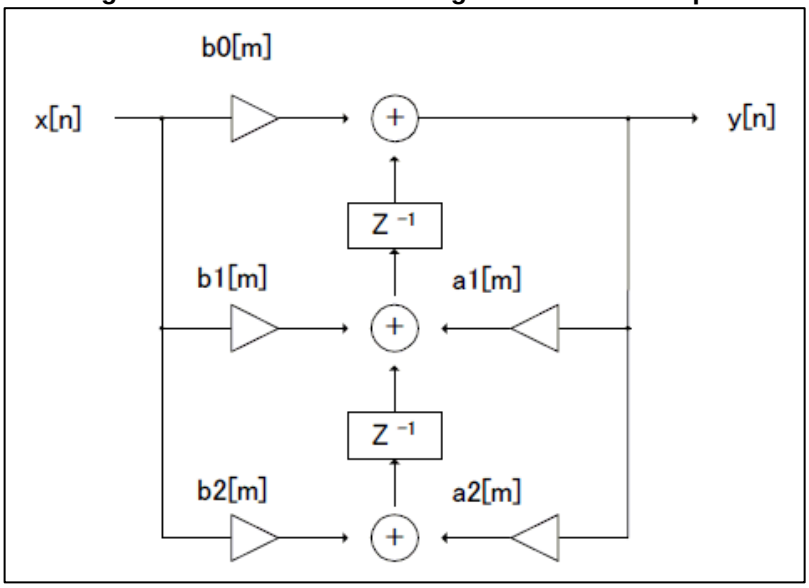
Figura 18 – Diagrama de um filtro IIR de segunda ordem transposto.



Fonte: Smith (2007).

Já na Figura 19 é informado a forma que a função do compilador espera os coeficientes para posterior aplicação do filtro e obtenção do sinal de saída. É notório que os coeficientes a_n tem sinal invertido, logo, para a aplicação dos coeficientes gerados no MATLAB, faz-se necessário a inversão dos mesmos.

Figura 19 – Diagrama de um filtro IIR de segunda ordem transposto no dsPIC.



Fonte: Digital Filter Corporation (2004).

O próximo passo é a quantização dos coeficientes gerados pelo MATLAB, que por padrão estão no formato ponto flutuante, para ponto fixo 16 bits, que é o padrão interpretado pelo compilador e aritmética interna do dsPIC. No MATLAB, a função *fi(...)* executa esta quantização conforme os parâmetros necessários.

Como parte importante das casas decimais dos coeficientes é perdida na quantização dos mesmos, deve-se observar a resposta do filtro antes e depois da quantização, a fim de verificar se a resposta continua dentro dos parâmetros especificados na criação do filtro. Para esta visualização, executa-se a função *fvtool(...)* duas vezes, uma com os coeficientes em ponto flutuante e outra com os coeficientes em ponto fixo, a comparação é visual e cabe ao desenvolvedor julgar se a resposta pós-quantização é adequada.

O restante do código disponível no Apêndice D manipula os coeficientes gerados para formar uma lista e informar o número de seções de segunda ordem. Esta lista deve ser inserida no arquivo de configuração do filtro calculado a ser implementado no dsPIC.

5.3 DIVISÃO DO SINAL EM 6 SUB BANDAS

Seguindo os passos de Nunes (2014), foram escolhidos seis intervalos de frequências para posterior filtragem do sinal de interesse, conforme Tabela 2.

Tabela 2 – Parâmetros escolhidos para os filtros digitais.

Numero	Fr0	Fp0	Fp1	Fr1	Rp	Rr
0	*110Hz*	*120Hz*	*200Hz*	*210Hz*	1dB	20dB
1	*1000Hz*	*1200Hz*	*1500Hz*	*1700Hz*	1dB	31dB
2	*1900Hz*	2000Hz	2500Hz	2600Hz	1dB	34dB
3	*3kHz*	*4kHz*	*6kHz*	*7kHz*	1dB	30dB
4	*6kHz*	*8kHz*	*11kHz*	*13kHz*	1dB	30dB
5	*10kHz*	*12kHz*	*17kHz*	*19kHz*	1dB	30dB

Fonte: Autoria própria.

Os seis filtros são passa-faixa, a banda de passagem é formada pelo intervalo de frequência existente entre Fp0 e Fp1, a banda de transição é formada entre Fr0 e Fp0, Fp1 e Fr1. Rp é o *ripple* máximo permitido dentro da banda de passagem, e, Rr é a mínima e atenuação das frequências 0Hz até Fr0 e Fr1 até o infinito.

5.4 ALGORITMO DE DETECÇÃO DE ENVOLTÓRIA E AUTO NIVELAMENTO

Cecília Jarne (2017) descreve em seu artigo um método empírico para a detecção de envoltória, o qual intitula como *peak approach*, “aproximação de pico” em português. Este método possui três etapas:

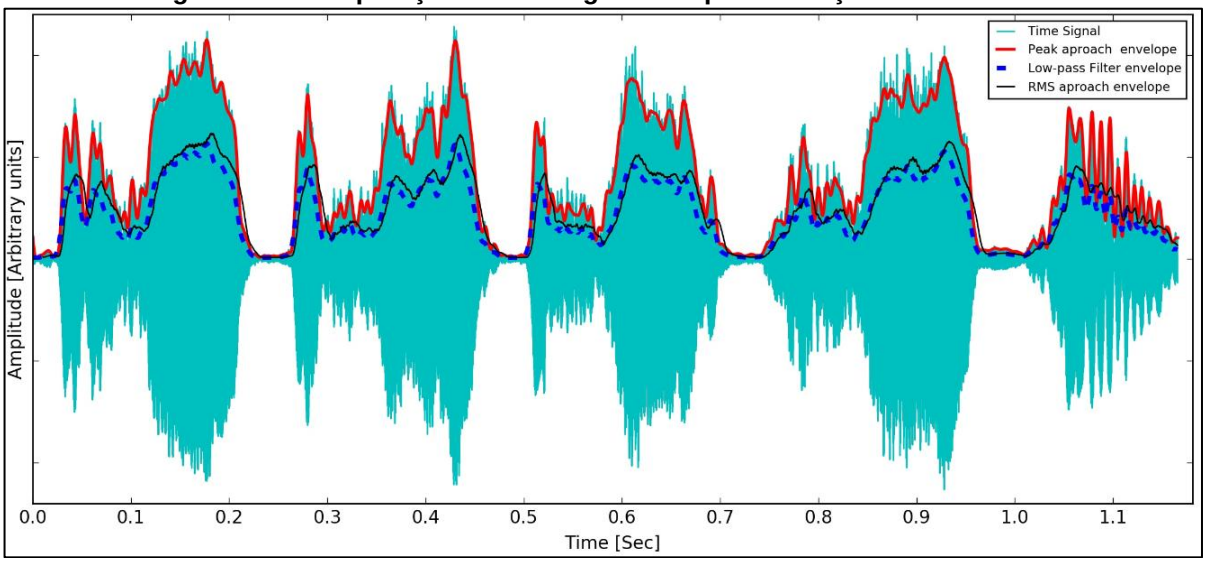
- Retirar o valor absoluto do sinal;
- Manter o maior valor dentro de um determinado número de amostras;
- Aplicar um filtro passa baixa.

A Figura 20 demonstra uma comparação entre o algoritmo descrito por ela e os conhecidos algoritmos para extração de envoltória utilizando filtro passa baixas e utilizando a aproximação pelo valor quadrático médio.

É notório que seu algoritmo possui uma resposta mais aproximada com a real envoltória do sinal utilizado para teste.

Para a implementação neste trabalho, foi seguido as primeiras duas etapas do artigo citado.

Figura 20 – Comparação de três algoritmos para extração de envoltória.



Fonte: Jarne (2017).

Para o auto nivelamento do sinal, utiliza-se uma variável que registra o maior valor processado pelo sistema. Desta variável se extrai o coeficiente de escalamento do sinal a ser transmitido. Os LED's coloridos citados na subseção 4.4 auxiliam o usuário para encontrar o nível aceitável para o sinal de entrada.

Nenhum LED acende caso a entrada de sinal esteja abaixo de 1 V pico a pico e a variável de auto nivelamento estiver vazia. Ao sinal ultrapassar 1 V pico a pico acende-se o LED verde, indicando a presença de algum sinal. O LED amarelo é acionado quando o sinal atinge 2 V pico a pico, indicando que o sinal atingiu o valor mínimo para que o auto nivelamento funcione. O LED vermelho acende uma vez que o sinal tenha excedido 3 V pico a pico, e mantém-se aceso, indicando que o usuário deve reduzir a amplitude do sinal de entrada e apertar o botão para reiniciar o processo de auto nivelamento.

5.5 TRANSMISSÃO DOS RESULTADOS

As seis envoltórias e o período estimado são transmitidos a cada 1 ms para o microcontrolador responsável por controlar os equipamentos DMX. O método de transmissão é uma comunicação assíncrona com velocidade de 250 kbps.

6 IMPLEMENTAÇÃO DO PROTOCOLO DMX E MACROS

6.1 DETECÇÃO DE PICOS

Uma estratégia para detecção de picos é a monitoração constante do sinal em busca de uma rampa de subida e outra de descida dentro de um curto espaço de tempo, isto é feito a partir do armazenamento de valores passados na memória e uma máquina de estado.

Esta detecção localização de picos é parte fundamental para a obtenção do sincronismo e para a estimação de periodicidade.

6.2 ESTIMAÇÃO DE PERIODICIDADE

Conhecer o período entre uma batida e outra, se ele é estável ou não, são informações importantes para a execução de macros, uma vez que são parâmetros de entrada para os mesmos.

Estes dados capturados, normalmente correspondem com o que um indivíduo está escutando no mesmo momento, e uma correta avaliação em tempo real permite que os resultados visuais sejam agradáveis e tornem a experiência mais imersiva para o indivíduo.

6.3 ALGORITMO DE REPRODUÇÃO DE MACROS SINCRONIZADOS

Uma macro pode ser definida como uma regra ou padrão que, conforme a entrada, produz uma saída, de acordo com um procedimento pré-definido.

Neste caso, as macros serão diversas cores e animações programadas previamente, e executadas em sequência ou aleatoriamente, de acordo com as informações obtidas pelo processamento digital do sinal de áudio.

Exemplos de macros implementáveis são efeitos vai-e-vem, centro para fora, fora para o centro, intensidade do som e cores aleatórias. Estes efeitos podem ser combinados entre si.

Outras macros de movimento em aparelhos que possuem esta função podem ser programadas e executadas, todavia, como foi obtido acesso a tais

aparelhos, a criação das macros limitou-se a controlar apenas 10 pontos de iluminação.

O trabalho desenvolvido tem a capacidade de controlar todos os 85 pontos de luzes fixas, de 6 canais cada, ou 30 pontos de luzes móveis, de 18 canais cada, sendo que a limitação para tal são os 512 canais disponíveis no protocolo

6.4 ALGORITMO DE GERAÇÃO DO PACOTE DMX

Como o protocolo DMX-512A já é bem conhecido, com vasta documentação, não é complexo de implementar, basta respeitar os valores de tempo citados na Seção 3.4.2 que os dispositivos escravos irão obedecer aos comandos.

7 RESULTADOS

Neste capítulo apresenta-se os resultados obtidos durante o desenvolvimento deste trabalho.

7.1 *HARDWARE*

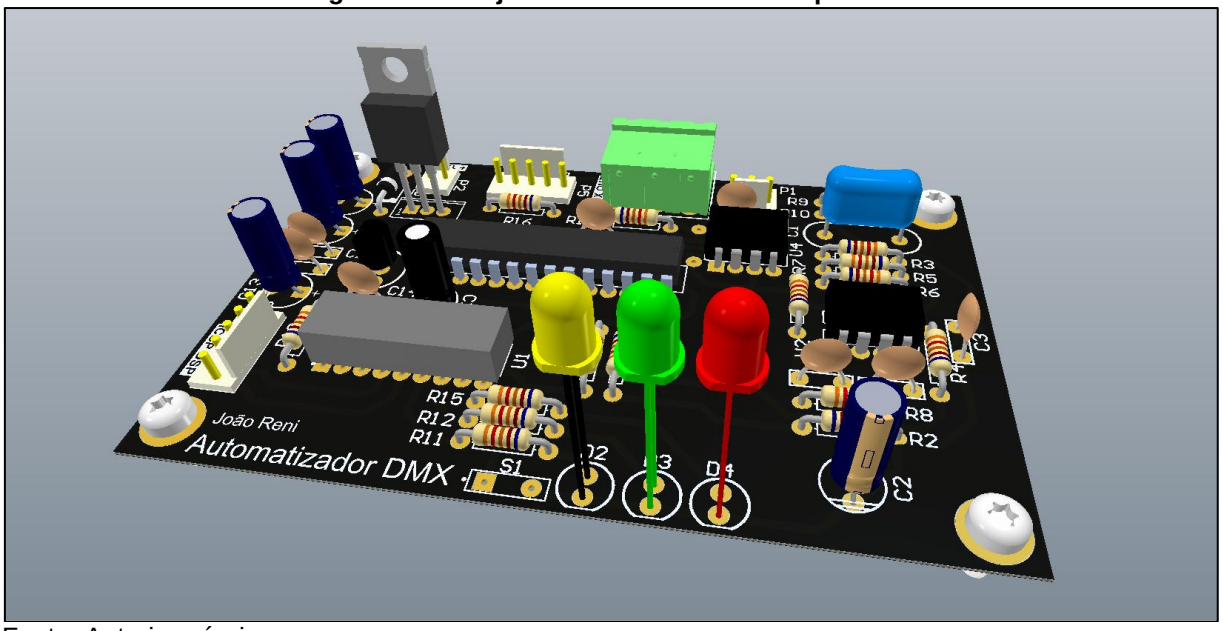
O circuito completo disponível no apêndice A pode ser considerado simples, uma vez que todos os 6 filtros e demais processamentos são executados de forma digital pelo dsPIC em questão.

Seria possível implementar todos os filtros passa banda de forma analógica, o que acarretaria em uma maior complexidade de *hardware,* levando ao desenvolvedor utilizar uma fonte simétrica, um aumento significativo no número de componentes e no tamanho da placa.

Para a execução do trabalho, será confeccionada uma placa de circuito impresso artesanal. A Figura 21 mostra a placa protótipo desenhada em computador.

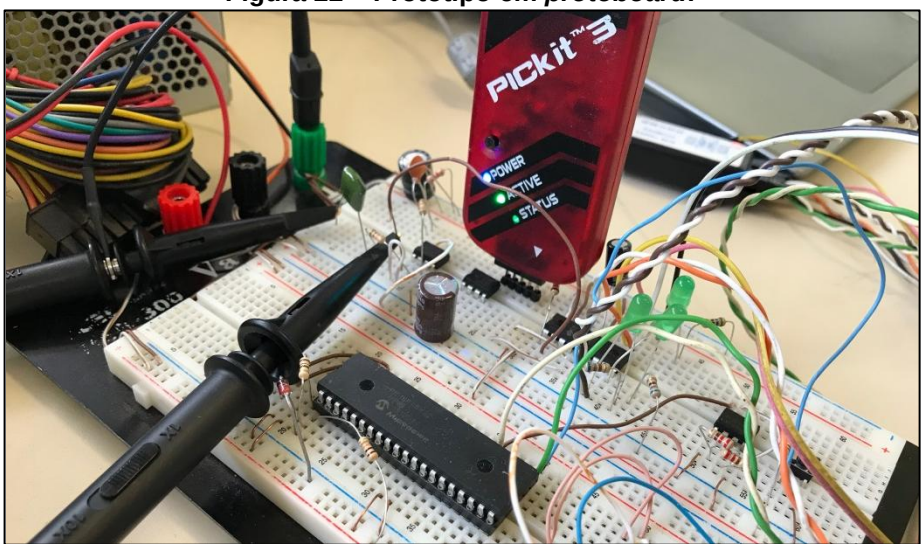
A Figura 22 exibe o circuito montado em uma *protoboard*. A Figura 23 exibe a placa artesanal feita para os testes finais.

Figura 21 – Projeto em 3D do circuito impresso.



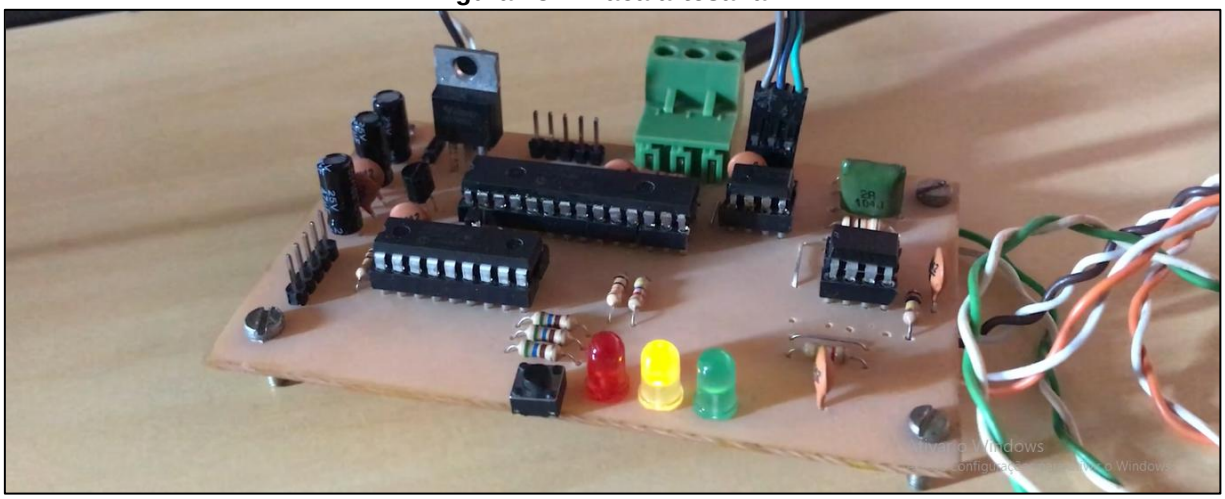
Fonte: Autoria própria.

Figura 22 – Protótipo em *protoboard*.



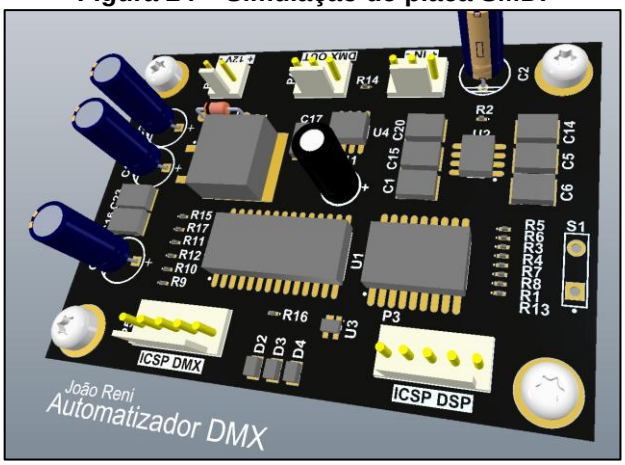
Fonte: Autoria própria.

Figura 23 – Placa artesanal.



Fonte: Autoria própria.

Figura 24 – Simulação de placa SMD.



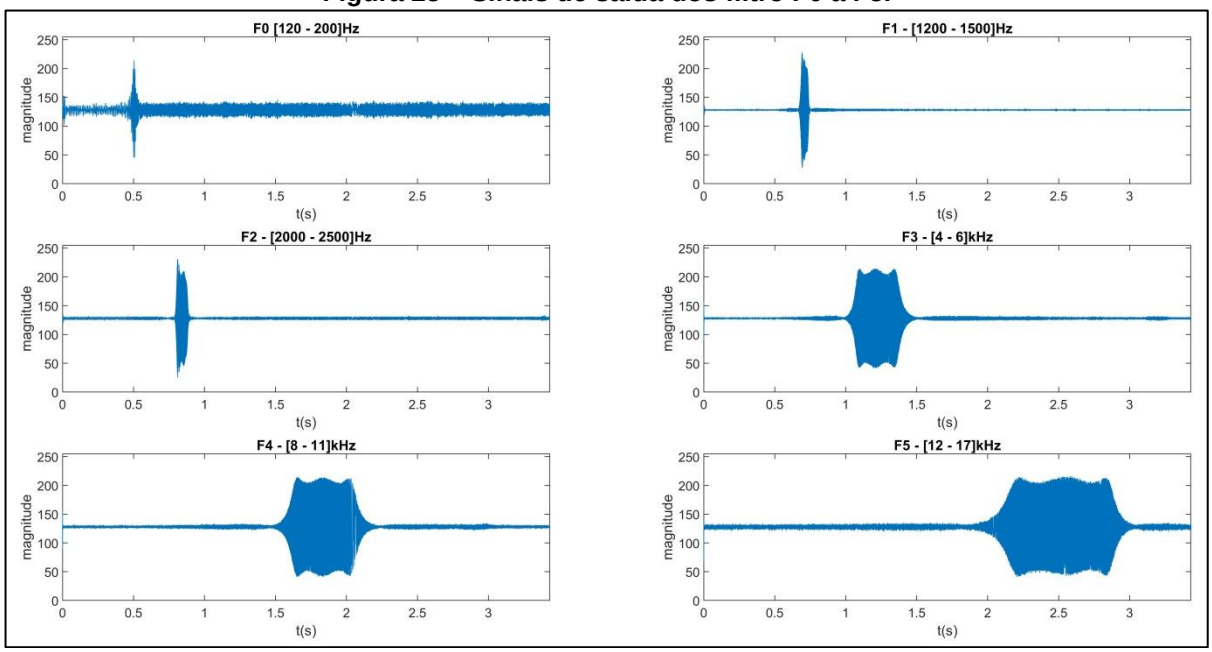
Fonte: Autoria própria.

A Figura 24 transmite uma prévia da utilização de técnicas de montagem de componentes na superfície, a fim de reduzir o tamanho final da placa. A título de comparação, a placa em PTH resultou em um tamanho 9,7 cm de comprimento por 6 centímetros de largura, sobre 6,8 cm de comprimento por 5 cm de largura para a técnica SMD. Uma redução de 41% na área total.

7.2 FILTROS DIGITAIS

Após a implementação dos coeficientes no microcontrolador, para teste dos filtros, foi aplicado uma onda senoidal de frequência crescente iniciando em 0Hz e após um período de 3,43s finaliza em 21kHz. A Figura 25 exibe os resultados obtidos na saída dos 6 filtros digitais projetados.

Figura 25 – Sinais de saída dos filtro F0 a F5.

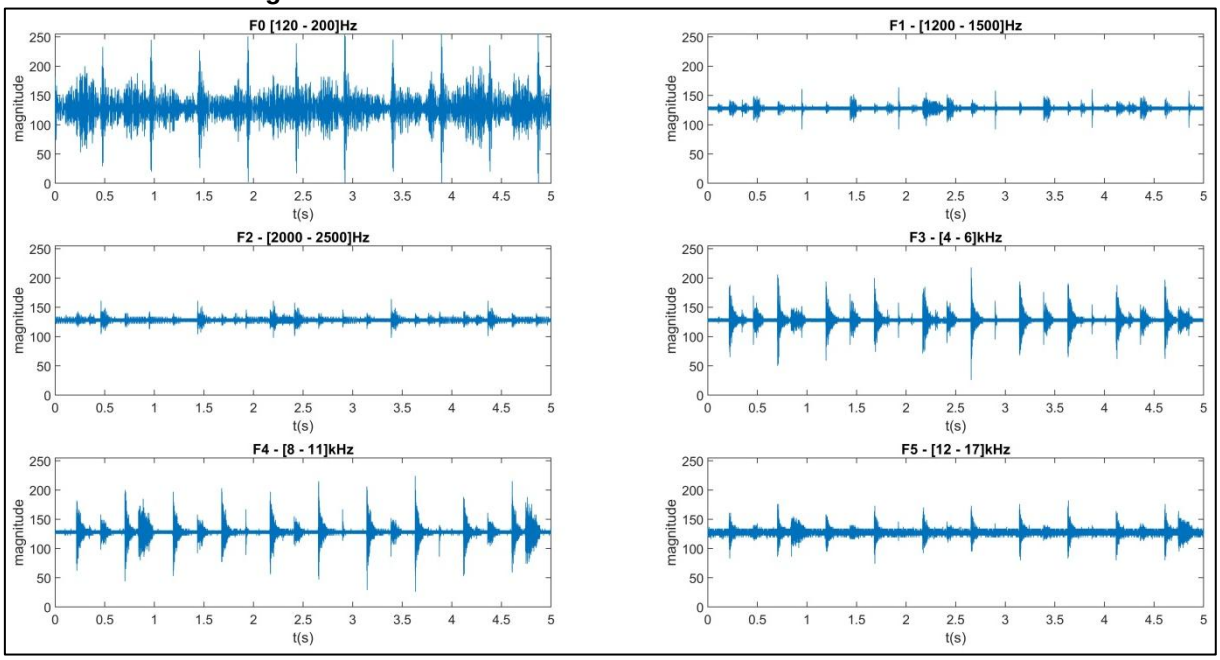


Fonte: Autoria própria.

Para a aquisição dos resultados, o dsPIC transmite por serial assíncrona conforme descrito na Seção 5.5 o sinal resultante. Assim, é posto um analisador de protocolo, e em seguida, os dados são exibidos pelo MATLAB.

O segundo sinal de teste foi um período de 5 segundos de música eletrônica com evidentes pulsações. Os sinais de saída dos filtros F0 a F5 podem ser observados na Figura 26.

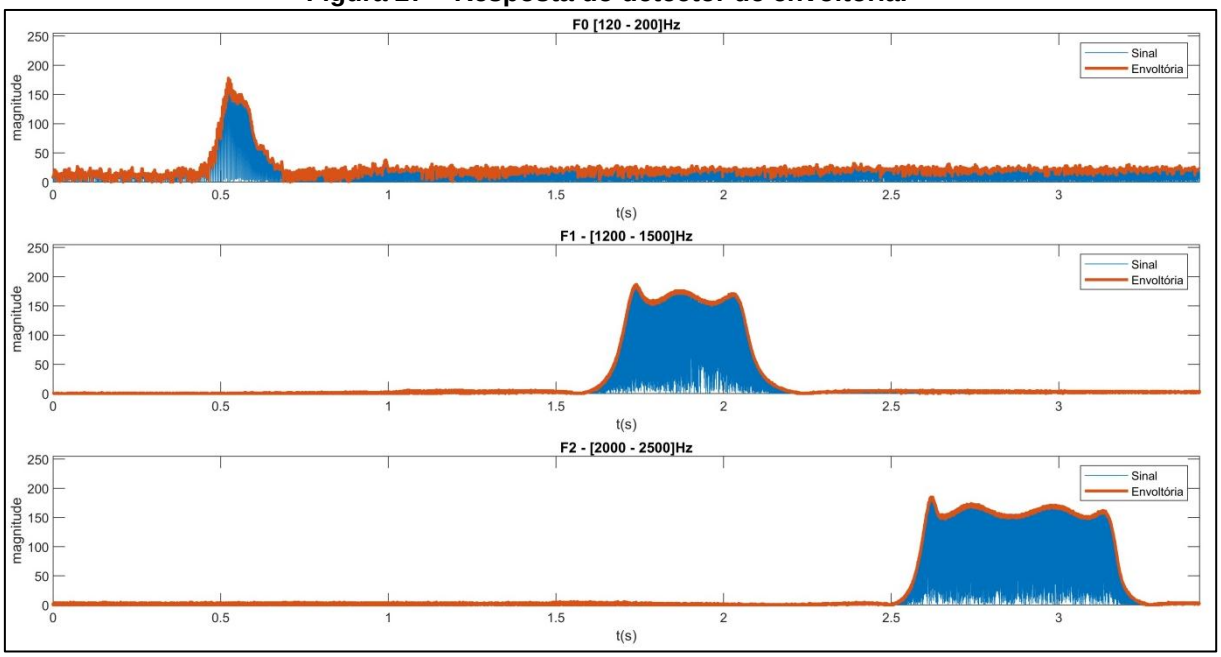
Figura 26 – Sinais de saída dos filtro F0 a F5 com música.



Fonte: Autoria própria.

A Figura 27 demonstra o resultado da aplicação do algoritmo detector de envoltória nos filtros F0 a F2 em tempo real, inserindo um sinal senoidal de frequência crescente, iniciando em 0 Hz e finalizando em 3 kHz, com duração de 3,42 segundos.

Figura 27 – Resposta do detector de envoltória.



Fonte: Autoria própria.

Observa-se nas Figuras 24, 25 e 26 que a saída do filtro F0 possui um ruído na frequência de rejeição que pode ser considerado elevado se comparado aos demais filtros implementados com frequências diferentes. Este fenômeno se dá pelo erro de quantização que deixa o filtro instável.

7.3 DETECTOR DE PICOS E PERIODICIDADE

Para a detecção de picos foi projetado uma máquina de estados que detecta borda de subida e descida do sinal. Observa-se sua implementação neste trecho de código, retirado do Apêndice C.

```
203 void trata_envoltoria(void)
204 {
205     static unsigned int maq_estado = 0;
206     static unsigned int val_anterior = 0;
207 
208     switch(maq_estado)
209     {
210         case 0:
211             if((rx_pack.F0 < val_anterior) &&
(rx_pack.F0 > SEGURALIXO))
212             {
213                 maq_estado ++;
214                 val_anterior = 0;
215             }
216             else if(rx_pack.F0 > val_anterior)
217             {
218                 val_anterior = rx_pack.F0;
219             }
220 
221         break;
222 
223         case 1:
224             if(t_ult_batida > 100)
225             {
226                 tem_batida = 1;
227                 adiciona_t_batida(t_ult_batida);
228                 t_ult_batida = 0;
229             }
230             else
231             {
232                 maq_estado = 0;
233             }
234         break;
235     }
236 }
```

O detector de picos por máquina de estado não obteve o resultado esperado. Para teste do detector, foram aplicados dois intervalos de áudio. O primeiro era composto apenas pelo toque solo de um Bumbo, constantemente

repetido duas vezes por segundo. O resultado no primeiro caso foi ótimo, detectando 98% dos picos desta amostra.

O segundo áudio é composto por uma música com diversos instrumentos. O resultado foi de 81% de acertos dos picos, resultando em problemas para se medir a periodicidade.

O algoritmo para mensurar a periodicidade funcionou muito bem, com o primeiro áudio de teste, o resultado foi de 118,1 BPM, um erro de 1,5% comparado com a real taxa de 120 BPM contida na amostra. Como resultado da segunda amostra de áudio, não foi obtida a mesma precisão frente a primeira amostra, observou-se oscilação em torno da correta medida de 128 BPM, em momentos com 32% de variação, indicando 87 BPM.

Para solucionar esta imprecisão, sugere-se estudar profundamente a composição das batidas de diversos estilos musicais, a fim de localizar melhor a frequência de pico que pode ser considerado como uma batida.

7.4 PROTOCOLO DMX

Nesta seção, apresenta-se o resultado da implementação de uma rotina para enviar os comandos elétricos até os dispositivos escravos.

Figura 28 – Início do frame DMX.

Fonte: Autoria própria.

A Figura 28 exibe um trecho do início do sinal gerado pelo PIC18F26K80 para controlar os aparelhos DMX, percebe-se que os tempos estão dentro das especificações.

A Figura 29 mostra com maior detalhe os dados dos primeiros 8 canais contidos no frame gerado.

Figura 29 – Detalhe dos primeiros canais no frame DMX.

Fonte: Autoria própria.

Os sinais presentes nas Figuras 28 e 29 foram adquiridos através de um analisador lógico.

A implementação do código para gerar o pacote de comunicação DMX pode se considerar simples. O trecho do código exibido é retirado do Apêndice C

```
238 void mandaframe(void)
239 {
240     setup_uart(0);
241     output_low(PIN_C6);
242     delay_us(200);
243     output_high(PIN_C6);
244     delay_us(100);
245     TXSTA = 0xE1;
246     TXREG = 255;
247     setup_uart(1);
248     enable_interrupts(INT_TBE);
249     k=0;
250 }

122 #INT_TBE
123 void envia_byte(void)
124 {
125     if (k <= 69)
126     {
127         TXREG = tx_pack.tx[k];
128         if(k == 69)
129         {
130             disable_interrupts(INT_TBE);
131         }
132         k ++;
133     }
134 }
```

O primeiro trecho, entre as linhas 238 e 250, contém tempos que necessitam atender as definições da norma USITT DMX512-A, conforme a Tabela 1 disponível na Seção 3.4.2. O segundo trecho, entre as linhas 122 e 134, mostra a rotina de

interrupção que envia o próximo *byte* assim que o *hardware* finalizar o envio do *byte* atual.

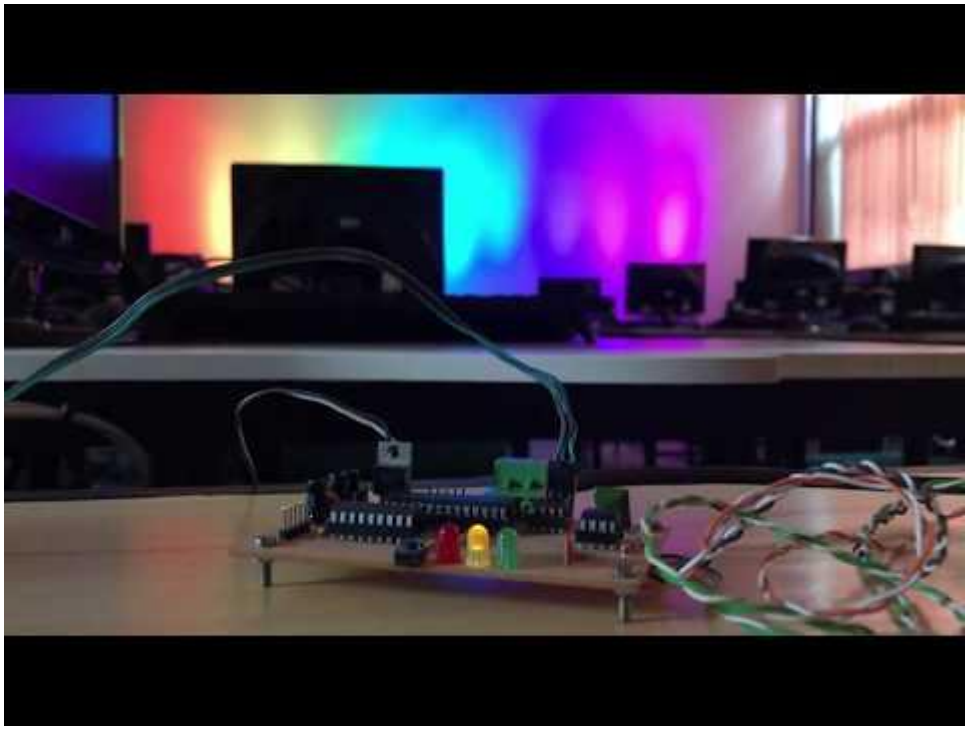
Os dispositivos escravos utilizados para teste foram corretamente comandados pelo frame gerado a partir do PIC18F26K80.

7.5 REPRODUÇÃO DE MACROS SINCRONIZADOS

Para a validação dos resultados, foram utilizados 10 canhões de luz da marca YDTECH, os resultados obtidos são mostrados no vídeo em anexo ao final desta seção.

O macro reproduzido no vídeo é apenas uma das possibilidades que podem ser programadas para posterior reprodução. Este macro reproduz as cores do arco-íris de forma circular, até que se detecte uma batida. Ao detectar uma batida, as cores se imobilizam e inicia-se um flash branco partindo do centro para fora. Caso não se detecte uma nova batida no período de dois segundos, é retomado as cores do arco-íris de forma circular.

Como complemento à explicação, está disponível em domínio público, na plataforma YouTube, o vídeo do sistema operando plenamente.



O vídeo citado está disponível no seguinte link: (https://www.youtube.com/watch?v=5NO-nra_F1k).

7.6 PROJETOS DE *SOFTWARE* E *HARDWARE*

Para fins de reprodução do presente trabalho, são disponibilizados os arquivos dos projetos de *software* e *hardware* que foram gerados. Estes arquivos podem ser vistos a partir do link: (https://drive.google.com/drive/folders/1nCSeMF-N4YLWnf8bLdf0ot8AIfGODoQL?usp=sharing).

8 CONCLUSÃO

Durante a elaboração deste trabalho, tornou-se perceptível a importância das aulas ministradas no curso, visto que, facilitou o entendimento do tema, possibilitando transformar o conhecimento adquirido em conclusões originais e independentes.

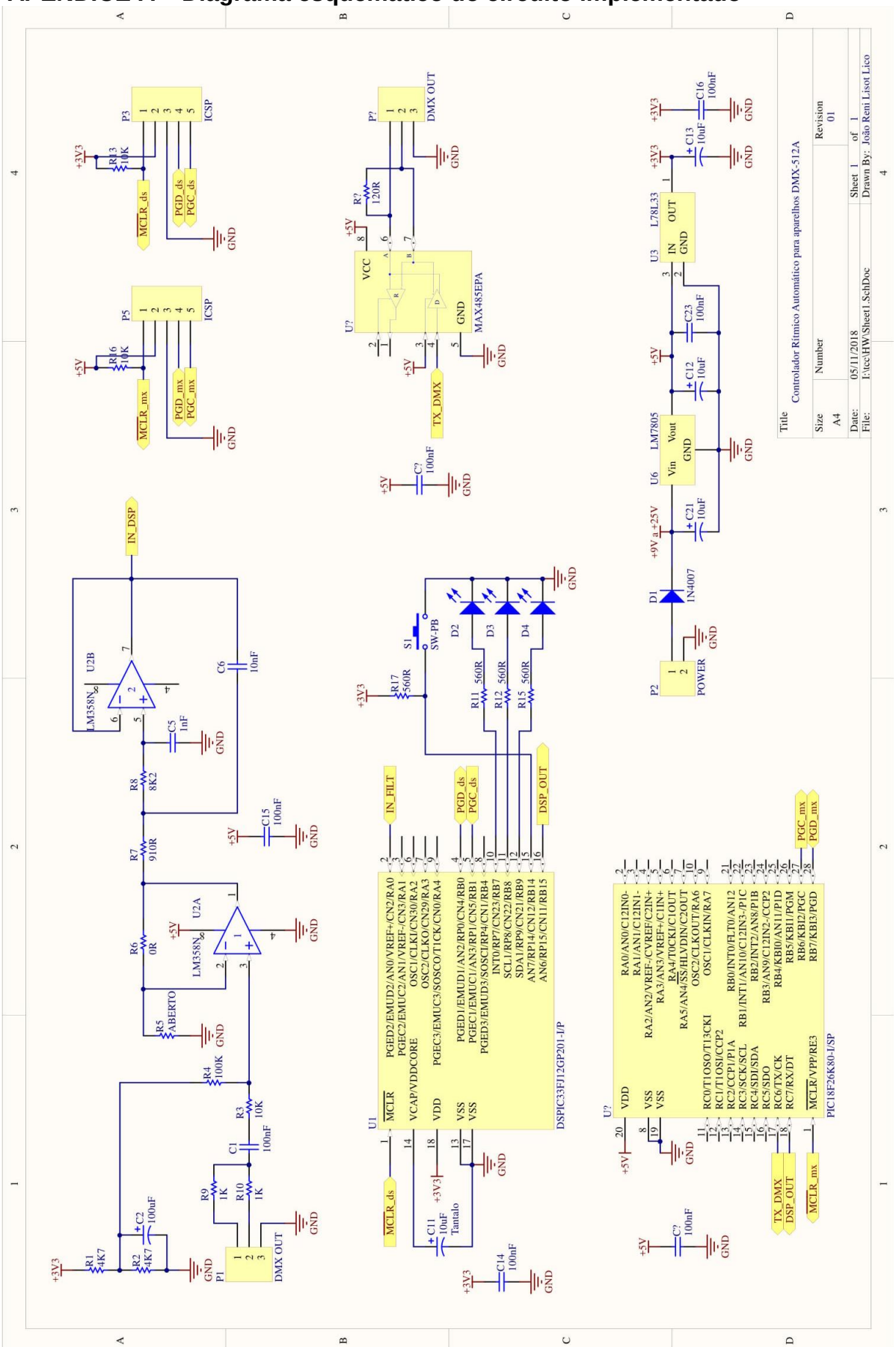
Os objetivos propostos foram concluídos com sucesso, com exceção à eficácia do algoritmo detector de picos, haja visto que os resultados obtidos dele não são condizentes com os esperados.

Mas, para que houvesse progresso desenvolvimento, foi necessário a realização de diversas pesquisas acerca de métodos para implementação de filtros digitais em sistemas embarcados, base estrutural do presente trabalho.

A realização da comunicação no padrão DMX-512A com dispositivos comerciais e a visualização do sistema operando é uma forma eficiente de comprovar que a análise de sinais em tempo real obteve êxito.

Estudar e aplicar filtros digitais na automação de iluminação é um tema pouco recorrente na produção acadêmica, e também não se encontra tecnologia brasileira neste sentido. Para futuros trabalhos, nesta linha de pesquisa, pode-se ampliar a quantidade de equipamentos controlados, através da Art-Net.

APÊNDICE A – Diagrama esquemático do circuito implementado



APÊNDICE B – Código completo implementado no dsPIC33FJ12GP201

```
/*
 * File:   main.c
 * Author: joaoreni
 *
 * Created on 10 de Outubro de 2018, 21:58
 */

#include "p33fj12gp201.h"
#include <uart.h>
#include <dsp.h>

#define    FCY    40000000UL    // Instruction cycle frequency, Hz - required for __delayXXX() to work
#include <libpic30.h>        //   delayXXX() functions macros defined here

_FOSCSEL(FNOSC_FRCPLL & IESO_OFF);
_FOSC(POSCMD_NONE & OSCIOFNC_ON & FCKSM_CSDCMD & IOL1WAY_OFF);



#define        VERDE    PORTAbits.RA4
#define        VERMELHO PORTAbits.RA2
#define        AMARELO  PORTAbits.RA3
#define        COEFautoniv  370

struct
{
    unsigned f1ms : 1;
    unsigned f10ms : 1;
    unsigned f50ms : 1;
    unsigned f100ms : 1;
    unsigned FazDSP : 1;
    unsigned EnviaUART : 1;
    unsigned SW4 : 1;
    unsigned SW6 : 1;
} flag;


unsigned char auto_niv = 1;
unsigned char i = 0;
unsigned char tcounter = 0;
unsigned char countfilt = 0;
unsigned char indexUART = 0;
unsigned int maxvalF0t = 0;
unsigned char maxvalF1t = 0;
unsigned char maxvalF2t = 0;
unsigned char maxvalF3t = 0;
unsigned char maxvalF4t = 0;
unsigned char maxvalF5t = 0;
unsigned int maxvalF0 = 0;
unsigned int maxvalF1 = 0;
unsigned int maxvalF2 = 0;
unsigned int maxvalF3 = 0;
unsigned int maxvalF4 = 0;
unsigned int maxvalF5 = 0;
unsigned int medval = 0;
unsigned int max_nivel = 0;
#define cofF0 500//120
#define cofF1 30
#define cofF2 5
#define cofF3 10
#define cofF4 10
#define cofF5 10

union
{
    struct
    {
        unsigned char header;
        unsigned char F0;
        unsigned char F1;
        unsigned char F2;
        unsigned char F3;
        unsigned char F4;
```

```
74          unsigned char F5;
75          unsigned char Ftot;
76      };
77      char bytes[8];
78  } tx pack;
79
80  #define NUMSAMP 1
81
82  extern IIRTransposedStruct F0Filter;
83  extern IIRTransposedStruct F1Filter;
84  extern IIRTransposedStruct F2Filter;
85  extern IIRTransposedStruct F3Filter;
86  extern IIRTransposedStruct F4Filter;
87  extern IIRTransposedStruct F5Filter;
88  extern IIRTransposedStruct F6Filter;
89
90  fractional inputSignal[NUMSAMP];
91  fractional outputSignalF0[NUMSAMP];
92  fractional outputSignalF1[NUMSAMP];
93  fractional outputSignalF2[NUMSAMP];
94  fractional outputSignalF3[NUMSAMP];
95  fractional outputSignalF4[NUMSAMP];
96  fractional outputSignalF5[NUMSAMP];
97  fractional outputSignalF6[NUMSAMP];
98
99  void __attribute__((__interrupt__, no_auto_psv)) _ADC1Interrupt(void)
100 {
101     IFS0bits.AD1IF = 0; // Limpa a flag de interrupção
102     inputSignal[0] = ADC1BUF0;
103     flag.FazDSP = 1;
104 }
105
106 void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
107 {
108     IFS0bits.T1IF = 0;  // Limpa a flag de interrupção
109     flag.f1ms = 1;
110 }
111
112 void __attribute__((__interrupt__, no_auto_psv)) _T2Interrupt(void)
113 {
114     IFS0bits.T2IF = 0; // Limpa a flag de interrupção
115     AD1CON1bits.SAMP = 1;
116 }
117
118 char UART1_ocupada(void)
119 {
120     if (!IFS0bits.U1TXIF)
121     {
122         return 1;
123     }
124     else
125     {
126         IFS0bits.U1TXIF = 0;
127         return 0;
128     }
129 }
130
131 void UART1_escreveChar(unsigned char Data)
132 {
133     while (UART1 ocupada());
134     U1TXREG = Data;
135 }
136
137 void UART1_escreveCharNB(unsigned char Data)
138 {
139     U1TXREG = Data;
140 }
141
142 void faz_dsp(void)
143 {
144     IIRTransposed(NUMSAMP, &outputSignalF0[0], &inputSignal[0], &F0Filter);
145     if(countfilt >= 10)
146     {
147         countfilt = 0;
148         IIRTransposed(NUMSAMP, &outputSignalF6[0], &outputSignalF0[0], &F6Filter);
149     }
150     countfilt ++;
```

```
151     IIRTransposed(NUMSAMP, &outputSignalF1[0], &inputSignal[0], &F1Filter);
152     //IIRTransposed(NUMSAMP, &outputSignalF2[0], &inputSignal[0], &F2Filter);
153     IIRTransposed(NUMSAMP, &outputSignalF3[0], &inputSignal[0], &F3Filter);
154     IIRTransposed(NUMSAMP, &outputSignalF4[0], &inputSignal[0], &F4Filter);
155     IIRTransposed(NUMSAMP, &outputSignalF5[0], &inputSignal[0], &F5Filter);
156 }
157
158 void trasmite_puro(void)
159 {
160     tx pack.F0 = (outputSignalF0[0] + 32767) / 260;
161     tx pack.F1 = (outputSignalF1[0] + 32767) / 260;
162     tx_pack.F2 = (outputSignalF2[0] + 32767) / 260;
163     tx_pack.F3 = (outputSignalF3[0] + 32767) / 260;
11
165     tx pack.F5 = (outputSignalF5[0] + 32767) / 260;
166 }
167
168 void faz_envolt_autoniv(void)
169 {
170     unsigned int atuvalF0 = abs(outputSignalF6[0]);
171     unsigned int atuvalF1 = abs(outputSignalF1[0]);
172     //unsigned int atuvalF2 = abs(outputSignalF2[0]);
173     unsigned int atuvalF3 = abs(outputSignalF3[0]);
174     unsigned int atuvalF4 = abs(outputSignalF4[0]);
175     unsigned int atuvalF5 = abs(outputSignalF5[0]);
176
177     unsigned int novo nivel = abs(inputSignal[0]);
178
179     if(novo_nivel > max_nivel)
180     {
181         max_nivel = novo_nivel;
182         auto niv = (max nivel / COEFautoniv) + 1;
183     }
184
185     if(novo_nivel > 10000) VERDE = 1;
186     else    VERDE = 0;
187
188     if(auto niv >= 40) AMARELO = 1;
189     else AMARELO = 0;
190
191     if(novo_nivel > 32300) VERMELHO = 1;
192
193     if(PORTBbits.RB4 == 0)
194     {
195         max_nivel = 0;
196         auto_niv = 1;
197         VERMELHO = 0;
198         AMARELO = 0;
199     }
200
201     if (atuvalF0 >= maxvalF0)
202     {
203         maxvalF0 = atuvalF0;
204         maxvalF0t = cofF0;
205     }
206     else
207     {
208         if (maxvalF0t)
209         {
210             maxvalF0t--;
211         }
212         else
213         {
214             if(maxvalF0 >= auto niv) maxvalF0 -= auto niv;
215         }
216     }
217
218     if (atuvalF1 >= maxvalF1)
219     {
220         maxvalF1 = atuvalF1;
221         maxvalF1t = cofF1;
222     }
223     else
224     {
225         if (maxvalF1t)
226         {
227             maxvalF1t--;
```

```
228         }
229         else
230         {
231             if(maxvalF1 >= auto niv) maxvalF1 -= auto niv;
232         }
233     }
234     /*
235     if (atuvalF2 >= maxvalF2)
236     {
237         maxvalF2 = atuvalF2;
238         maxvalF2t = cofF2;
239     }
240     else
241     {
242         if (maxvalF2t)
243         {
244             maxvalF2t--;
245         }
246         else
247         {
248             if(maxvalF2 >= auto niv) maxvalF2 -= auto niv;
249         }
250     }*/
251 
252     if (atuvalF3 >= maxvalF3)
253     {
254         maxvalF3 = atuvalF3;
255         maxvalF3t = cofF3;
256     }
257     else
258     {
259         if (maxvalF3t)
260         {
261             maxvalF3t--;
262         }
263         else
264         {
265             if(maxvalF3 >= auto niv) maxvalF3 -= auto niv;
266         }
267     }
268 
269     if (atuvalF4 >= maxvalF4)
270     {
271         maxvalF4 = atuvalF4;
272         maxvalF4t = cofF4;
273     }
274     else
275     {
276         if (maxvalF4t)
277         {
278             maxvalF4t--;
279         }
280         else
281         {
282             if(maxvalF4 >= auto niv) maxvalF4 -= auto niv;
283         }
284     }
285 
286     if (atuvalF5 >= maxvalF5)
287     {
288         maxvalF5 = atuvalF5;
289         maxvalF5t = cofF5;
290     }
291     else
292     {
293         if (maxvalF5t)
294         {
295             maxvalF5t--;
296         }
297         else
298         {
299             if(maxvalF5 >= auto niv) maxvalF5 -= auto niv;
300         }
301     }
302 }
303 
304 void inicia_uart(void)
```

```
305 {
306     ConfigIntUART1(UART_RX_INT_DIS & UART_RX_INT_PR0 & UART_TX_INT_DIS & UART_TX_INT_PR0);
307     // Aloca U1TX para RP7, Pin 10
308     RPOR3bits.RP7R = 3;
309     // Aloca U1RX para RP9, Pin 12
310     RPINR18bits.U1RXR = 9;
311     U1MODEbits.STSEL = 1;
312     U1MODEbits.PDSEL = 0;
313     U1MODEbits.ABAUD = 0;
314     U1MODEbits.BRGH = 1;
315     U1BRG = 40;
316     U1MODEbits.UARTEN = 1;
317     U1STAbits.UTXEN = 1;
318 }
319
320 void inicia_adc(void)
321 {
322     ADPCFG = 0b0000001111111110; // RB3, RB4, and RB5 are digital
323     AD1CON2bits.VCFG = 0;
324     AD1CON3bits.ADRC = 1;
325     AD1CON2bits.CHPS = 0;
326     AD1CON1bits.SSRC = 0b111;
327     AD1CON3bits.SAMC = 0;
328     AD1CON1bits.ASAM = 0;
329     AD1CON1bits.AD12B = 1;
330     AD1CON1bits.FORM = 3;
331     IFS0bits.AD1IF = 0; // Clear ISR flag
332     AD1CON1bits.ADON = 1; // turn ADC ON
333     IEC0bits.AD1IE = 1;
334     return;
335 }
336
337 void inicia_TMR1(void)
338 {
339     T1CONbits.TGATE = 0;
340     T1CONbits.TCS = 0;
341     T1CONbits.TCKPS = 2;
342     T1CONbits.TSYNC = 0;
343     TMR1 = 0;
344     PR1 = 613; // configura interrupção para 1ms
345     T1CONbits.TON = 1; // liga timer1
346     IEC0bits.T1IE = 1; // liga interrupção do timer1
347 }
348
349 void inicia_TMR2(void)
350 {
351     T2CONbits.TGATE = 0;
352     T2CONbits.TCS = 0;
353     T2CONbits.TCKPS = 0;
354     TMR2 = 0; // zera prescaler
355     PR2 = 891; // configura para 44000hz
356     IPC1bits.T2IP = 7; // prioridade máxima para esta interrupção
357     T2CONbits.TON = 1; // liga timer2
358     IEC0bits.T2IE = 1; // liga interrupção do timer2
359 }
360
361 void inicia_clock(void)
362 {
363     CLKDIVbits.PLLPOST = 0;
364     PLLFBDbits.PLLDIV = 40;
365 }
366
367 /*
368  *
369  */
370 int main(int argc, char** argv)
371 {
372     inicia_clock();
373     inicia_TMR1();
374     inicia_TMR2();
375     inicia uart();
376     inicia adc();
377     TRISB = 0b0000000010000;
378     TRISA = 0b0000000000011;
379     TRISAbits.TRISA4 = 0;
380     IIRTransposedInit(&F0Filter);
381     IIRTransposedInit(&F1Filter);
```

```
382     IIRTransposedInit(&F2Filter);
383     IIRTransposedInit(&F3Filter);
384     IIRTransposedInit(&F4Filter);
385     IIRTransposedInit(&F5Filter);
386     IIRTransposedInit(&F6Filter);
387
388     tx_pack.header = 0xFF;
389
390     while (1)
391     {
392         if (flag.f1ms)
393         {
394             flag.f1ms = 0;
395             flag.EnviaUART = 1;
396         }
397
398         if (flag.FazDSP)
399         {
400             flag.FazDSP = 0;
401             faz dsp();
402             faz envolt autoniv();
403         }
404
405         if (flag.EnviaUART)
406         {
407             if (UART1 ocupada() == 0)
408             {
409                 switch(indexUART)
410                 {
411                     case 1:
412                         tx_pack.F0 = maxvalF0/auto_niv;
413                     break;
414
415                     case 2:
416                         tx_pack.F1 = maxvalF1/auto_niv;
417                     break;
418
419                     case 3:
420                         tx_pack.F2 = maxvalF2/auto_niv;
421                     break;
422
423                     case 4:
424                         tx pack.F3 = maxvalF3/auto niv;
425                     break;
426
427                     case 5:
428                         tx pack.F4 = maxvalF4/auto niv;
429                     break;
430
431                     case 6:
432                         tx_pack.F5 = maxvalF5/auto_niv;
433                     break;
434
435                     case 7:
436                         //medval =
(maxvalF0+maxvalF1+maxvalF2+maxvalF3+maxvalF4+maxvalF5)/6;
437                         tx_pack.Ftot = auto_niv;//medval/auto_niv;
438                     break;
439                 }
440
441                 if((indexUART >= 1) && (tx pack.bytes[indexUART] == 255))
tx_pack.bytes[indexUART] = 254;
442
443                 UART1 escreveCharNB(tx pack.bytes[indexUART]);
444
445                 indexUART++;
446
447                 if (indexUART >= 8)
448                 {
449                     indexUART = 0;
450                     flag.EnviaUART = 0;
451                 }
452             }
453         }
454     }
455 }
456
```

APÊNDICE C – Código completo implementado no PIC18F26K80

```
#include <18f26k80.h>
#include <math.h>
#fuses INTRC IO, NOWDT, NOPUT, NOPROTECT, NOBROWNOUT, NOCPD, PLLEN
#use delay(clock = 64M)
#use rs232(BAUD=250000, BITS=8, STOP=2, UART1, stream=a)


#define BRANCO      0xFFFFFF
#define VERDE       0x00FF00
#define AZUL        0x0000FF
#define VERMELHO    0xFF0000

#define SEGURALIXO  100

#BYTE TXREG = 0xFAD
#BYTE TXSTA = 0xFAC
#BYTE RCSTA = 0xFAB
#BYTE SPBRG = 0xFAF

typedef struct
{
    unsigned int red;
    unsigned int green;
    unsigned int blue;
}cor24b;



typedef struct
{
    unsigned int main dimmer;
    unsigned int main strobe;
    unsigned int func_op;
    unsigned int func_speed;
    cor24b cor;
} typeparled;

union
{
    struct
    {
        unsigned char header;
        unsigned char F0;
        unsigned char F1;
        unsigned char F2;
        unsigned char F3;
        unsigned char F4;
        unsigned char F5;
        unsigned char Ftot;
    };
    char bytes[8];
} rx pack;

union
{
    typeparled parled[10];
    unsigned int tx[70];
} tx pack;

const unsigned int teste0[50] =
    {BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,
     BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,
     BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,
     BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,
     BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE,AZUL,VERMELHO,BRANCO,VERDE};
const unsigned int teste1[50] =
{0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,

0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,

0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,

0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
```

```
 69
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1};
 70 const unsigned int teste2[50] =
{0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 71
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 72
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 73
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 74
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1};
 75 const unsigned int teste3[50] =
{0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 76
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 77
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 78
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 79
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1};
 80 const unsigned int teste4[50] =
{0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 81
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 82
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 83
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,
 84
0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1,0xF2F2F2,0xF3F3F3,0xF0F0F0,0xF1F1F1};
 85 unsigned int temp = 0;
 86 unsigned int cont10ms = 0;
 87 unsigned int cont100ms = 0;
 88 unsigned int batida = 0;
 89 unsigned long t ult batida = 2300;
 90 unsigned long tempos batida[30];
 91 unsigned int i = 0;
 92 unsigned int k = 0;
 93 short tem_batida = 0;
 94 short dado_disp = 0;
 95 short flag 1ms = 0;
 96 short flag 10ms = 0;
 97 short flag_100ms = 0;
 98 short flag_1000ms = 0;
 99
100 #INT RDA
101 void dado_disponivel (void)
102 {
103     static unsigned int index;
104     unsigned char d = 0;
105     d = getc();
106     if(d == 0xFF)
107     {
108         index = 0;
109     }
110     if(index <= 7)
111     {
112         rx pack.bytes[index] = d;
113         index ++;
114     }
115     if(index == 8)
116     {
117         dado disp = 1;
118         index ++;
119     }
120 }
121
122 #INT TBE
123 void envia_byte(void)
124 {
125     if (k <= 69)
126     {
127         TXREG = tx_pack.tx[k];
128         if(k == 69)
129         {
```

```
130             disable_interrupts(INT_TBE);
131         }
132         k ++;
133     }
134 }
135
136
137 // interrupção de base de tempo
138 #INT TIMER2
139 void timer2(void)
140 {
141     flag_1ms = 1;
142     if(t_ult_batida < 2500) t_ult_batida ++;
143
144     cont10ms++;
145     if (cont10ms >= 10)
146     {
147         cont10ms = 0;
148         flag_10ms = 1;
149         cont100ms ++;
150     }
151     if(cont100ms >= 10 )
152     {
153         cont100ms = 0;
154         flag_100ms = 1;
155     }
156 }
157
158 void adiciona_t_batida(unsigned long t_batida)
159 {
160     static unsigned int posicao = 0;
161     tempos batida[posicao] = t batida;
162     posicao ++;
163     if (posicao >= 30) posicao = 0;
164 }
165
166 cor24b Hue2Rgb(unsigned int hue)
167 {
168     cor24b rgb;
169     unsigned char region, remainder, p, q, t;
170
171     region = hue / 43;
172     remainder = (hue - (region * 43)) * 6;
173
174     p = (252 - remainder);
175     q = (255 * (255 - ((255 * remainder) >> 8))) >> 8;
176     t = remainder;
177
178     switch (region)
179     {
180         case 0:
181             rgb.red = 255; rgb.green = t; rgb.blue = 0;
182             break;
183         case 1:
184             rgb.red = p; rgb.green = 255; rgb.blue = 0;
185             break;
186         case 2:
187             rgb.red = 0; rgb.green = 255; rgb.blue = t;
188             break;
189         case 3:
190             rgb.red = 0; rgb.green = p; rgb.blue = 255;
191             break;
192         case 4:
193             rgb.red = t; rgb.green = 0; rgb.blue = 255;
194             break;
195         default:
196             rgb.red = 255; rgb.green = 0; rgb.blue = p;
197             break;
198     }
199
200     return rgb;
201 }
202
203 void trata_envoltoria(void)
204 {
205     static unsigned int maq estado = 0;
206     static unsigned int val_anterior = 0;
```

```
207
208     switch(maq_estado)
209     {
210         case 0:
211             if((rx pack.F0 < val anterior) && (rx pack.F0 > SEGURALIXO))
212             {
213                 maq_estado ++;
214                 val_anterior = 0;
215             }
216             else if(rx pack.F0 > val anterior)
217             {
218                 val_anterior = rx_pack.F0;
219             }
220
221         break;
222
223         case 1:
224             if(t_ult_batida > 100)
225             {
226                 tem batida = 1;
227                 adiciona t batida(t ult batida);
228                 t ult batida = 0;
229             }
230             else
231             {
232                 maq estado = 0;
233             }
234         break;
235     }
236 }
237
238 void mandaframe(void)
239 {
240     setup_uart(0);
241     output_low(PIN_C6);
242     delay_us(200);
243     output high(PIN C6);
244     delay us(100);
245     TXSTA = 0xE1;
246     TXREG = 255;
247     setup_uart(1);
248     enable_interrupts(INT_TBE);
249     k=0;
250 }
251
252 void zera_tudo(void)
253 {
254     unsigned int k = 0;
255     for (k = 0; k <= 69; k++)
256     {
257         tx_pack.tx[k] = 0;
258     }
259 }
260
261 void seta_strobo (unsigned char nivel)
262 {
263     if(nivel <= 10)
264     {
265         for(i=0;i<10;i++)
266         {
267             tx pack.parled[i].main strobe = nivel*25;
268         }
269     }
270 }
271
272 void seta_main_dimmer (unsigned int nivel)
273 {
274     for(i=0;i<10;i++)
275     {
276         tx pack.parled[i].main dimmer = nivel;
277     }
278 }
279
280 void cor_geral (cor24b cor)
281 {
282     for(i=0;i<10;i++)
283     {
```

```
284         tx_pack.parled[i].cor = cor;
285     }
286 }
287
288 void cor_unico (unsigned int par, cor24b cor)
289 {
290     tx_pack.parled[par].cor = cor;
291 }
292
293 void reproduz_macro_a (void)
294 {
295     static unsigned int macro_a = 0;
296     tx_pack.parled[0].cor = Hue2Rgb(teste0[macro_a]);
297     macro a ++;
298     if(macro a >= 50) macro a = 0;
299 }
300
301 void reproduz_macro_b (void)
302 {
303     static unsigned int macro b = 0;
304     tx pack.parled[1].cor = Hue2Rgb(teste1[macro b]);
305     macro b ++;
306     if(macro_b >= 50) macro_b = 0;
307 }
308
309 void reproduz_macro_c (void)
310 {
311     static unsigned int macro_c = 0;
312     tx_pack.parled[2].cor = Hue2Rgb(teste2[macro_c]);
313     macro_c ++;
314     if(macro_c >= 50) macro_c = 0;
315 }
316
317 void reproduz_macro_d (void)
318 {
319     static unsigned int macro_d = 0;
320     tx pack.parled[3].cor = Hue2Rgb(teste3[macro d]);
321     macro d ++;
322     if(macro_d >= 50) macro_d = 0;
323 }
324
325 void reproduz_macro_e (void)
326 {
327     static unsigned int macro e = 0;
328     tx_pack.parled[4].cor = Hue2Rgb(teste4[macro_e]);
329     macro_e ++;
330     if(macro e >= 50) macro e = 0;
331 }
332
333 void trata_batida(void)
334 {
335
336     tx pack.parled[0].cor = Hue2Rgb(batida);
337     tx pack.parled[1].cor = Hue2Rgb(batida + 20);
338     tx pack.parled[2].cor = Hue2Rgb(batida + 40);
339     tx pack.parled[3].cor = Hue2Rgb(batida + 60);
340     tx_pack.parled[4].cor = Hue2Rgb(batida + 80);
341     tx_pack.parled[5].cor = Hue2Rgb(batida + 100);
342     tx pack.parled[6].cor = Hue2Rgb(batida + 120);
343     tx pack.parled[7].cor = Hue2Rgb(batida + 140);
344     tx pack.parled[8].cor = Hue2Rgb(batida + 160);
345     tx_pack.parled[9].cor = Hue2Rgb(batida + 180);
346
347     if((t ult batida >= 0)&&(t ult batida < 50))
348     {
349         tx pack.parled[4].cor = 0xFFFFFF;
350         tx_pack.parled[5].cor = 0xFFFFFF;
351     }
352     if((t_ult_batida >= 50)&&(t_ult_batida < 100))
353     {
354         tx pack.parled[3].cor = 0xFFFFFF;
355         tx pack.parled[6].cor = 0xFFFFFF;
356     }
357     if((t_ult_batida >= 100)&&(t_ult_batida < 150))
358     {
359         tx pack.parled[2].cor = 0xFFFFFF;
360         tx_pack.parled[7].cor = 0xFFFFFF;
```

```
361     }
362     if((t_ult_batida >= 150)&&(t_ult_batida < 200))
363     {
364         tx pack.parled[1].cor = 0xFFFFFF;
365         tx pack.parled[8].cor = 0xFFFFFF;
366     }
367     if((t_ult_batida >= 200)&&(t_ult_batida < 250))
368     {
369         tx pack.parled[0].cor = 0xFFFFFF;
370         tx pack.parled[9].cor = 0xFFFFFF;
371     }
372 }
373
374 void main(void)
375 {
376     setup timer 2(T2 DIV BY 16, 249, 4);
377     enable_interrupts(INT_TIMER2);
378     enable_interrupts(INT_RDA);
379     enable_interrupts(GLOBAL);
380     set tris c(0b10111111);
381
382     zera tudo();
383
384     seta_main_dimmer(0xFF);
385
386     while (1)
387     {
388         if(dado_disp)
389         {
390             dado_disp = 0;
391             trata_envoltoria();
392         }
393
394         if (tem_batida)
395         {
396             tem_batida = 0;
397             batida = temp;
398         }
399
400         if (flag_10ms)
401         {
402             flag_10ms = 0;
403
404             temp ++;
405
406             if(t_ult_batida > 2000)
407             {
408                 tx pack.parled[0].cor = Hue2Rgb(temp);
409                 tx pack.parled[1].cor = Hue2Rgb(temp + 25);
410                 tx pack.parled[2].cor = Hue2Rgb(temp + 50);
411                 tx_pack.parled[3].cor = Hue2Rgb(temp + 75);
412                 tx_pack.parled[4].cor = Hue2Rgb(temp + 100);
413                 tx pack.parled[5].cor = Hue2Rgb(temp + 125);
414                 tx pack.parled[6].cor = Hue2Rgb(temp + 150);
415                 tx pack.parled[7].cor = Hue2Rgb(temp + 175);
416                 tx pack.parled[8].cor = Hue2Rgb(temp + 200);
417                 tx_pack.parled[9].cor = Hue2Rgb(temp + 225);
418             }
419             else trata batida();
420             mandaframe();
421
422         }
423         if (flag_100ms)
424         {
425             flag 100ms = 0;
426             //reproduz macro a();
427             //reproduz_macro_b();
428             //reproduz_macro_c();
429             //reproduz_macro_d();
430             //reproduz macro e();
431         }
432     }
433 }
```

APÊNDICE D – Código completo implementado no MATLAB com função de gerar os coeficientes para filtros IIR digitais.

```
% Função desenvolvida para gerar coeficientes para filtros IIR a serem
% utilizados no microchip dsPIC
% Testado e funcionando no dsPIC33FJ12GP201, utilizando o compilador XC16
% versão 1.35
% Estes coeficientes são válidos para serem incluídos na estrutura de um
% IIR transposto, e esta estrutura é parâmetro para a função IIRTransposed

clc
clear

Fs = 44000;
Ftrans = 10;
Fpass = [50 80];
Fstop = [20 110];
Rp = 1;
Rs = 30;

Wp = Fpass/(Fs/2);
Ws = Fstop/(Fs/2);

[n,Wp] = ellipord(Wp,Ws,Rp,Rs);

[z,p,k] = ellip(n,Rp,Rs,Wp,'bandpass');

sos = zp2sos(z,p,k)

fvtool(sos)
fvtool(fi(sos,1,16))

for u = 1:size(sos,1)
    sos(u,5) = -sos(u,5);
    sos(u,6) = -sos(u,6);
end

for u = 1:size(sos,1)
    saida(((u-1)*5)+1) = sos(u,1);
    saida(((u-1)*5)+2) = sos(u,2);
    saida(((u-1)*5)+3) = sos(u,5);
    saida(((u-1)*5)+4) = sos(u,3);
    saida(((u-1)*5)+5) = sos(u,6);
end

saida = fi(saida,1,16);
saida = transpose(saida);
said = char(saida.hex);

asaida = strcat('numero de secoes:  ',int2str(size(sos,1)));
for u = 1:size(saida)
    asaida(u+1,1:14) = strcat('.hword 0x',said(u,1:4),';');
end
```

REFERÊNCIAS

BURGESS, Mark P D. **General Electric History:** Semiconductor Research and Development at General Electric. 2011. Disponível em: <https://sites.google.com/site/transistorhistory/Home/us-semiconductor-manufacturers/general-electric-history>. Acesso em: 13 nov. 2018.

CASSOU, Raphael. **A HISTÓRIA DA ILUMINAÇÃO CÊNICA.** 2011. Disponível em: <http://aureliodesimoni.blogspot.com.br/2011/08/historia-da-iluminacao-cenica.html>. Acesso em: 14 maio 2018.

CLAY PARKY. ***LED-based moving lights*.** 2018. Disponível em: <https://www.claypaky.it/en/products/entertainment/led-based-moving-lights>. Acesso em: 17 maio 2018.

DIGITAL FILTER CORPORATION (Japão) (Org.). **Tutorial25: How to implement IIR coefficients into dsPIC.** 2004. Disponível em: <http://digitalfilter.com/tutorials/dspiciir/endspiciir3.html>. Acesso em: 19 set. 2018

EDISON ELECTRIC ILLUMINATING COMPANY. **The history of stage and theatre lighting.** 1929. Disponível em: <http://www.iar.unicamp.br/lab/luz/ld/C%EAnica/Hist%F3ria/the_history_of_stage_and_theatre_lighting.pdf>. Acesso em: 01 nov. 2018.

ENTERTAINMENT SERVICES AND TECHNOLOGY ASSOCIATION. ***Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories*.** Nova York: ANSI, 2013. 50 p. Disponível em: <http://tsp.esta.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2013.pdf>. Acesso em: 02 maio 2018.

IRWIN, Steve. **What Is DMX?** 2018. Disponível em: <https://www.lightsoundjournal.com/2018/01/30/what-is-dmx/>. Acesso em: 06 nov. 2018.

JARNE, Cecília. **Simple empirical algorithm to obtain signal envelope in three steps.** 2017. Disponível em: <https://pdfs.semanticscholar.org/401b/10f76b1a9b668a5df0a829873aaf356ac27f.pdf>. Acesso em: 29 out. 2018.

LINS, Karyne. A evolução das mesas de iluminação. **Backstage**, Rio de Janeiro, v. 1, n. 151, p.84-88, jun. 2006. Mensal. Disponível em: <http://luztecnologiaearte.weebly.com/uploads/1/3/5/6/13567015/sobre_mesas_de_luz.pdf>. Acesso em: 29 maio 2018.

LUNA, Oscar; TORRES, Daniel. ***DMX512 PROTOCOL IMPLEMENTATION*.** 2006. Disponível em: <https://www.nxp.com/docs/en/application-note/AN3315.pdf>. Acesso em: 20 maio 2018.

MAGALHÃES, Pedro Emanuel Fernandes. ***Ethernet Stage Lightning Protocol*.** 2014. 31 f. Dissertação (Mestrado) - Curso de Engenharia Eletrotécnica e de Computadores, Universidade do Porto, Porto, 2014. Disponível em: <https://paginas.fe.up.pt/~ee08288/docs/PDI_Relatorio_Final_200804909_Pedro_Magalhaes.pdf>. Acesso em: 19 maio 2018.

MARAIS, Hein. ***RS-485/RS-422 Circuit Implementation Guide.*** *Norwood: Analog Devices*, 2008. 12 p. Disponível em: <http://www.analog.com/media/en/technical-documentation/application-notes/AN-960.pdf>. Acesso em: 14 maio 2018.

MICROCHIP. ***Audio and Speech Recording / Encoding / Compression Application Solutions*.** Disponível em: <http://www.microchip.com/design-centers/audio-and-speech/technology/recording>. Acesso em: 13 maio 2018.

MICROCHIP. Datasheet: **DsPIC33FJ12GP201/202.** Publicação eletrônica, 2011. 260 p. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70264E.pdf>. Acesso em: 26 abr. 2018.

NUNES, Leonardo de Oliveira. **ALGORITMOS PARA ANÁLISE RÍTMICA COMPUTACIONAL.** 2014. 201 f. Tese (Doutorado) - Curso de Pós-graduação em Engenharia Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014. Disponível em: <http://pee.ufrj.br/teses/textocompleto/2014092401.pdf>. Acesso em: 14 maio 2018.

OLIVEIRA, Fábio Roberto; BARAUCE, Marcelo; PINHEIRO, Marcos Menino. **COMUNICAÇÃO SEM FIO USANDO DMX 512.** 2012. 115 f. TCC (Graduação) - Curso de Tecnologia em Sistemas de Telecomunicações, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná, Curitiba, 2012. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/884/1/CT_COTEL_2012_1_01.pdf>. Acesso em: 07 maio 2018.

ON SEMICONDUCTOR. Datasheet: **NCP1117.** Publicação eletrônica, 2017. 17 p. Disponível em: <https://www.onsemi.com/pub/Collateral/NCP1117-D.PDF>. Acesso em: 17 maio 2018.

PLSN, *Projection Lights & Staging News*. ***Avolites Ai Infinity RX8 servers feed spectacular Calibash Las Vegas visuals*.** 2018. Disponível em: <http://plsn.com/featured/featured-slider/avolites-ai-infinity-rx8-servers-feed-spectacular-calibash-las-vegas-visuals/>. Acesso em: 13 maio 2018.

SMITH, Julius Orion. **Transposed Direct-Forms.** 2007. Disponível em: <https://ccrma.stanford.edu/~jos/fp/Transposed_Direct_Forms.html>. Acesso em: 30 out. 2018

SOUZA, Eder de. ***Audio Beat Track*.** 2012. Disponível em: <https://ederwander.wordpress.com/2012/10/21/audio-beat-track/>. Acesso em: 21 maio 2018.

WHELAN, M.. **The First Form of Electric Light.** 2016. Disponível em: <http://edisontechcenter.org/ArcLamps.html>. Acesso em: 26 out. 2018.

WILLIAMS, Bill. ***A History of Light and Lighting*.** 1999. Disponível em: <https://web.archive.org/web/20150702103538/http://www.mts.net:80/~william5/history/hol.htm>. Acesso em: 10 maio 2018.