Luís Carlos Monsanto Lopes

Licenciado em Ciências de Engenharia Electrotécnica e de Computadores

Sintetizador *Midi* com Capacidade de Processamento

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Anikó Katalin Horváth da Costa, Professora Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:
Presidente: Prof. Doutor Luís Filipe dos Santos Gomes
Arguente: Prof. Doutor Luís Filipe Figueira de Brito Palma
Vogais: Prof. Doutora Anikó Katalin Horváth da Costa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Agosto 2013

Luís Carlos Monsanto Lopes

Licenciado em Ciências de Engenharia Electrotécnica e de Computadores

Sintetizador *Midi* com Capacidade de Processamento

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Anikó Katalin Horváth da Costa, Professora Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:
Presidente: Prof. Doutor Luís Filipe dos Santos Gomes
Arguente: Prof. Doutor Luís Filipe Figueira de Brito Palma
Vogais: Prof. Doutora Anikó Katalin Horváth da Costa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Agosto 2013

DIREITOS DE CÓPIA

Sintetizador *Midi* com Capacidade de Processamento

AGRADECIMENTOS

Um especial agradecimento aos meus pais pelos anos que me têm aturado e dado força para continuar esta caminhada.

Ao meu irmão pelo apoio, força e ajuda em todas as fases da minha vida. Obrigado por tudo!

A todos os colegas que partilharam comigo esta fase da vida: Luís Miranda, Micael Simões, Fábio Júlio, Fábio Alves, Flávio Diniz, Pedro Gomes, Tiago Xavier, Ricardo Mendonça, Carlos Carvalho, Carlos Calmeiro, João Santos, Rui Branco, Luís Oliveira, João Gameiro e Marcelo Rodrigues por terem ajudado em todo o percurso académico e pela amizade que fica.

Obrigado Iolanda Travassos pela paciência, compreensão, força e me teres ajudado a descontrair em muitos momentos. Fizeste com que a vida fizesse sentido.

Por fim à orientadora Anikó Costa, pela ajuda e apoio durante todo o processo, pelo conhecimento e motivação.

Resumo

O *Arduino* é uma plataforma *open source hardware*, programável através de linguagem de código aberto, baseada na linguagem *Wiring*, e de ambiente de desenvolvimento integrado.

Com a plataforma *Arduino* é possível programar qualquer tipo de equipamento interativo independente, ou então com um computador.

A implementação que irá ser efetuada através do *Arduino* será um instrumento musical com comunicação com um computador. O controlador *midi (Musical Instrument Digital Interface)* final combinará as funcionalidades de um sintetizador (ver secção 2.1) e um *sampler* (ver secção 2.2).

A plataforma *Arduino* mega 2560 r3, consiste num microcontrolador RISC (*Reduced Instruction Set Computer)* de baixa potência com um CMOS 8-bits baseado na arquitetura Atmel AVR, ATmega2560. Este tem 256Kb de memória para armazenamento de código.

Esta plataforma possui 54 pinos de entradas/saídas digitais, 16 portas de entradas/saídas analógicas, 4 UARTs (*universal asynchronous receiver/transmitter*), um oscilador de cristal de 16Mhz, uma ligação USB, uma entrada de alimentação, ligação ICSP (*in-circuit serial programming)* e um botão de *reset*.

O *Arduino* Mega 2560 tem a capacidade de comunicar com um computador, com outro *Arduino* ou com outros microcontroladores. Esta comunicação é válida devido aos quatro UARTs que ele fornece. Também suporta as comunicações TWI/I2C (I2C – *inter-integrated circuit*) e SPI (*serial peripheral interface)*.

Com o emparelhamento do *Arduino* a um computador, é possível fazer então a comunicação com o programa *Pure Data*. Este é desenvolvido através de uma linguagem gráfica em tempo real, para desenvolvimento de áudio.

O *Pure Data* tem uma grande capacidade de interação na forma como faz o acesso e a interligação de áudio, *midi*, gráfico e vídeo. A linguagem oferece uma grande capacidade de manutenção e desenvolvimento.

Assim como o *Arduino*, o *Pure Data* também é *open source* e também está disponível para todos os sistemas operativos, o que faz com que a ligação destes dois sistemas seja uma mais-valia e com custos de licenciamento nulos.

Palavras-Chave: Controlador *Midi (Musical Instrument Digital Interface)*, Sintetizador, *Sampler, Arduino, Pure Data.*

Abstract

The Arduino is an open source platform that the microcontroller is programmed using the Arduino programming language, based on Wiring and with an integrated development environment.

With this platform is possible to program any type of interactive equipment, independent or with a computer.

The implementation of the midi *(*Musical Instrument Digital Interface*)* controller with a computer communication will be through the Arduino platform. The final midi controller combines a synthesizer (session 2.1) and a sampler or midi fighter (session 2.2).

The Arduino mega 2560 r3 platform, consists of a microcontroller RISC (Reduced Instruction Set Computer) with low-power CMOS 8-bits based on the architecture Atmel AVR, ATmega 2560. This has 256Kb of memory code storage.

This platform has 54 pins of digital inputs/outputs, 16 analog, 4 UARTs (universal asynchronous receiver/transmitter), crystal oscillator with 16Mhz, USB connection, input supply, ICSP (in-circuit serial programming) connection and a reset button.

The Arduino Mega 2560 has the ability to communicate with a computer, Arduino or a microcontroller. This ability is valid due to the four UARTs that it provides. It also supports the TWI/I2C (I2C – inter-integrated circuit) and SPI (serial peripheral interface) communications.

With the pairing of the Arduino to a computer, you can communicate the platform with the Pure Data software. This software is developed through a graphical language in real time, to develop audio.

The Pure Data has a great ability to interact with audio, midi, graphic and video. The language offers a large capacity of maintenance and development.

Like the Arduino, the Pure Data is also open source and is also available for all operating systems. This makes the connection between these two systems an assets and licensing costs zero.

Keywords: Midi (Musical Instrument Digital Interface) Controller, Synthesizer, Midi Fighter, Arduino, Pure Data.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas

Abreviaturas

ADC - *Analogic to Digital Converter*

ADSR - *Attack, Decay, Sustain, Release*

BPM – Beat per Minute

CMOS - *Complementary Metal-Oxide-Semiconductor*

DAC - *Digital to Analog Converter*

dB – Decibel

DJ - *Disk Jockey*

FFT – Fast *Fourier Transform*

FM – *Frequency Modulation*

Hz – Hertz

ICSP - *In-Circuit Serial Programming*

I2C – *Inter-Integrated Circuit*

LFO - *Low-frequency oscillation*

MIDI - *Musical Instrument Digital Interface*

PC – *Personal Computer*

PD - Pure Data

RISC - *Reduced Instruction Set Computer*

SPI – S*erial Peripheral Interface*

UARTs - *Universal Asynchronous Receiver/Transmitter*

USB - U*niversal Serial Bus*

1. Introdução

1.1. Motivação

A teoria de *Darwin* diz que a origem da música esteve "nos sons emitidos pelos progenitores humanos durante a época de acasalamento", uma teoria que parece ser insustentável. A explicação mais plausível encontra-se na teoria de *Theophrastus*, em que a música é atribuída a toda a gama de emoções humanas [1].

Os sons emitidos pelos seres vivos eram considerados música pois expressavam as emoções sentidas na altura, como por exemplo, um animal quando faz um som de choro ou de alegria os tons são completamente diferentes, diferenciando assim o seu estado de espirito [1].

Os Tambores foram os primeiros instrumentos musicais a surgirem. Estes apareceram ainda nas mais remotas eras da humanidade. Todos pensamos que os tambores servem apenas para a reprodução de música, no entanto nas civilizações mais antigas também eram utilizados para meio de comunicação devido à sua forte potência sonora [1].

Mais tarde apareceram os derivados dos tambores, ou seja, flauta, trompete, trompa ou qualquer outro instrumento de tubo. No entanto os instrumentos de lírica são também precedentes do tambor, assim como a harpa Egípcia a guitarra e assim como todos os instrumentos de cordas [1].

Só em 1711 na cidade de Florença, Itália, é que pela primeira vez teve a sua primeira referência o piano, inventado pelo *Bartolomeo Cristofori*. Desde então têm vindo a ter grandes modificações, passando dos pianos de cauda que funcionavam com cordas através das suas vibrações criavam belas melodias, até aos que temos hoje em dia que funcionam já através de sofisticados *software* que fazem com que estes trabalhem digitalmente [1].

Contudo é a partir do piano que apareceram então os famosos sintetizadores. Estes tiveram a sua aparição em 1960 por *Kazu Theremin*. Em 1964, *Robert Moog* e *Herbert Deutsch* desenvolveram um modelo que acabaria por se identificar mais com os sintetizadores que hoje em dia conhecemos mas, apesar de tudo, era muito grande e monofónico [2].

Só em 1970 é que apareceu o primeiro sintetizador minimamente transportável. Foi lançado pela *Moog Music* e foi dos mais usados entre os sintetizadores da altura tanto para as músicas populares como para a música eletrónica [2].

Os controladores *midi* têm como precedente o sintetizador. Foram desenvolvidos já na década de 80 no qual este instrumento podia enviar e receber instruções musicais através de outros instrumentos e computadores [2].

Hoje em dia os controladores *midi* já são bastante utilizados tanto nas rádios como em quase todo o tipo de música existente.

O desafio é desenvolver um sistema eficiente, baseado em plataformas *open source hardware*, de fácil implementação, fácil reconfiguração e de baixo custo que apresente uma rápida resposta às situações apresentadas por alguns artistas.

1.2. Objetivos

O primeiro objetivo deste trabalho é desenvolver um equipamento *midi* capaz de responder de forma eficaz e rápida, aos efeitos musicais que se querem instruir num som a reproduzir.

O segundo objetivo passa por conseguir juntar dois equipamentos, ou seja, ter um sintetizador e um *midi fighter* implementados num equipamento. Esta mesa poderá então incluir sons musicais às listas de músicas já existentes ou então efetuar sons originais com a destreza do utilizador.

De forma a fazer com que o controlador *midi* consiga operar conforme o utilizador pretende, o terceiro objetivo passa por, permitir a reconfiguração do controlador, para que o utilizador consiga alterar certos botões de pressão, isto é possível através do mapeamento do programa de música que o utilizador pretende utilizar.

1.3. Estrutura

No capítulo 2 são apresentados alguns equipamentos que contém a tecnologia *midi*, a criação e método de utilização desta tecnologia e sistemas open source utilizados para o projeto.

No capítulo 3 são introduzidos conceitos tal como, o som e a sua propagação, formas de onda, filtros, tecnologia dos sintetizadores, modulação em frequência e sintetizadores de amostras digitais. Este último é onde se insere este projeto.

No capítulo 4 é apresentado a arquitetura da plataforma e do sistema, no qual é sobre estas arquiteturas que o controlador foi implementado. Também tem as bibliotecas da linguagem *Pure Data* e *Arduino*, referindo qual das bibliotecas que foram utilizadas. Por fim um esboço final do controlador *midi* assim como o circuito implementado e o código implementado no projeto.

No capítulo 5 retiram-se conclusões do trabalho desenvolvido, propondo no final algumas ideias de trabalhos futuros associados a este trabalho de dissertação.

2. Sistemas Midi e Open Source

Existem vários tipos de controladores *midi*, são divididos em vários equipamentos, em função do seu tipo de operação: sintetizadores, *samplers*, baterias eletrónicas e estações de trabalho e sequenciadores de *hardware*.

2.1. Sintetizadores

Os sintetizadores são instrumentos musicais eletrónicos que geram formas de ondas através de circuitos de processamento. Os sintetizadores podem apresentar várias formas, Figura 2.1 é um exemplo de um sintetizador. Estes instrumentos musicais foram os primeiros a serem utilizados com a tecnologia *midi* [3].

Figura 2.1 - Exemplo de um sintetizador recente [web1].

Com o desenvolvimento do *midi* foi possível criar equipamentos de tamanho reduzido para que fossem totalmente transportáveis.

A principal característica dos sintetizadores é fazer uma modelação à música através de, alterações de velocidades, agudos, graves, e mesmo até realizar construções melódicas.

Para este efeito existem dois métodos possíveis, o método analógico e o método digital.

O método analógico e os mais convencionais são gerados por síntese aditiva, subtrativa e *wavetable*.

Este método é mais convencional pois consegue gerir bem o tempo e a capacidade de cálculo e memória, ou seja, quanto menos capacidade de cálculo existir, mais memória será necessário usar no sintetizador, sendo assim serão gravadas mais amostras, reduzindo a velocidade de cálculo para efetuar as modulações. Quanto mais capacidade de cálculo existente, menor será a capacidade de gravação de sons, mas aumenta a velocidade do processo de modulação [3][4].

O método analógico define a envolvente da amplitude de um som em quatro fases: ataque (*attack*), decaimento (*decay*), regime estacionário (*sustain*) e decaimento final (*release*). Isto faz com que a análise de som seja mais rápida [3][4].

O método digital, os sintetizadores são constituídos por, síntese aditiva, modulação em frequência, modulação física e granular [3][4].

Este último é através de uma criação puramente matemática. Normalmente este tipo de sintetizadores são mais usados para fins pedagógicos e de investigação, pois como são muito à base de cálculos matemáticos, apesar de conseguirem ter uma qualidade de som mais semelhante aos instrumentos do que o método analógico, é necessário algum tempo de processamento o que para efeitos musicais não se torna muito viável [3][4].

2.2. Samplers (midi fighter),

Os *Samplers* são uma vertente diferente dos controladores *midi*. Estes em vez de usarem a modulação de sinais como os sintetizadores, usam uma tecnologia chamada DAC (acrónimo para a expressão em língua inglesa *digital to analog converter*), conversor digital para analógico. Mesmo sendo mais fácil a utilização dos dados digitais, é necessário fazer esta transformação para analógico, pois só assim é possível ser reconhecido pelo ouvido humano.

Também usam o ADC (acrónimo para a expressão em língua inglesa *Analog to Digital Converter*), conversor analógico para digital, basicamente é o oposto do DAC. É bastante frequente a transformação de sinais analógicos para digitais pois como referido no parágrafo anterior, os dados digitais são muito mais fáceis de manusear que os analógicos. Esta conversão para efeitos de áudio serve para gravações de áudio através de um microfone, transformando o sinal analógico para digital e passar a usar essa gravação como uma amostra digital, um som de um curto espaço de tempo.

Os equipamentos que utilizam amostras digitais (*Midi Fighter*), Figura 2.2, reproduzem sons de instrumentos reais, e podem ou não ser reproduzidos sobre a música ou som que esteja a ser reproduzido. O sintetizador faz a alteração à onda da música a ser reproduzida. São muitas vezes utilizados como uma adição extra aos sintetizadores [web2].

As principais características dos *Samplers* são:

- Gravação de Áudio
- Repetições de Sons
- Conversão de Digital para Analógico
- Conversão de Analógico para Digital
- Reprodução de Áudio

Figura 2.2 - Exemplo de um *Midi Sampler* [web3].

2.3. Baterias Eletrónicas

O funcionamento das baterias eletrónicas, Figura 2.3, é semelhante ao *sampler*. O *sampler* mais focado em reproduzir vários sons de instrumentos musicais, ou excertos de músicas através de botões de pressão, as baterias eletrónicas servem para reproduzir todo o tipo de som que existe em termos de baterias e tambores.

As baterias eletrónicas funcionam também através da técnica ADC. Têm sensores piezoelétrico que deteta o local da batida enviando um sinal analógico. Depois através da técnica ADC o sinal analógico será convertido para digital de maneira que o som que esteja configurado no *pad* selecionado pela batida seja reproduzido [5].

O pedal interativo que as baterias eletrónicas têm serve para fazer alterações aos graves ou para configurar sons pré configurados. O pedal funciona através de um sensor de pressão [5].

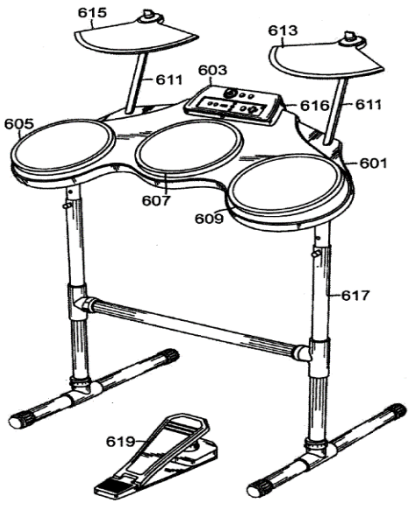


Figura 2.3 - Exemplo de uma Bateria Eletrónica [5]

2.4. Estações de Trabalho e Sequenciadores de *Hardware*

As estações de trabalho e sequenciadores de *hardware*, Figura 2.4, são compostos por vários equipamentos ligados entre si, como sintetizadores, equipamentos de amostras digitais, baterias eletrónicas e os seus derivados dependendo muito do que for necessário para o utilizador [6].

As estações de trabalho e sequenciadores de *hardware* são uma mais-valia para os artistas e editoras que trabalham na área musical. As suas principais características são a capacidade de gravar e manipular os padrões das gravações, podendo reproduzi-los com a ordem desejada, dando ainda a possibilidade de juntar vários sons, modifica-los alterando as suas amplitudes e frequências.

A criação da estação de trabalho digital foi bastante importante pois permite recriar todo o tipo de instrumento, conseguindo fazer até o trabalho inteiro de uma orquestra.

Figura 2.4 - Exemplo de uma estação de trabalho e sequenciador [web4].

2.5. Midi

A tecnologia *midi* veio revolucionar muito os equipamentos musicais eletrónicos. Antes do seu aparecimento a comunicação entre os equipamentos e computadores não era uniformizado. Através deste protocolo que é constituído por especificações de *software* e *hardware*, foi possível uniformizar os sistemas.

Em 1981, *Dave Smith* desenvolveu uma interface universal para os sintetizadores, no entanto foi em 1982 que surgiu o protocolo *midi* através de uma parceria entre duas companhias, uma japonesa e outra

americana, dando assim o nome ao protocolo *Midi 1.0 Detailed Specifications*. Este foi um aperfeiçoamento a interface já desenvolvida por *Dave Smith* [3].

Em 1983 foi apresentado o primeiro equipamento já com a tecnologia *midi*, começando assim a era dos instrumentos *midi*, passou a existir uma compatibilidade com tudo o que era controlador *midi* e computador [7].

Podemos pensar na tecnologia *midi* como uma partitura para os controladores *midi* e computadores, ou seja, é através desta tecnologia que as mensagens são enviadas de forma a reproduzir os sons configurados nos botões ou potenciómetros [6][7].

Hoje em dia todos os equipamentos musicais eletrónicos utilizam o protocolo *midi*. Este protocolo foi aceite pelos músicos e compositores devido ao seu *hardware*, Figura 2.5, de baixo custo e também à capacidade de comunicação como já referido.

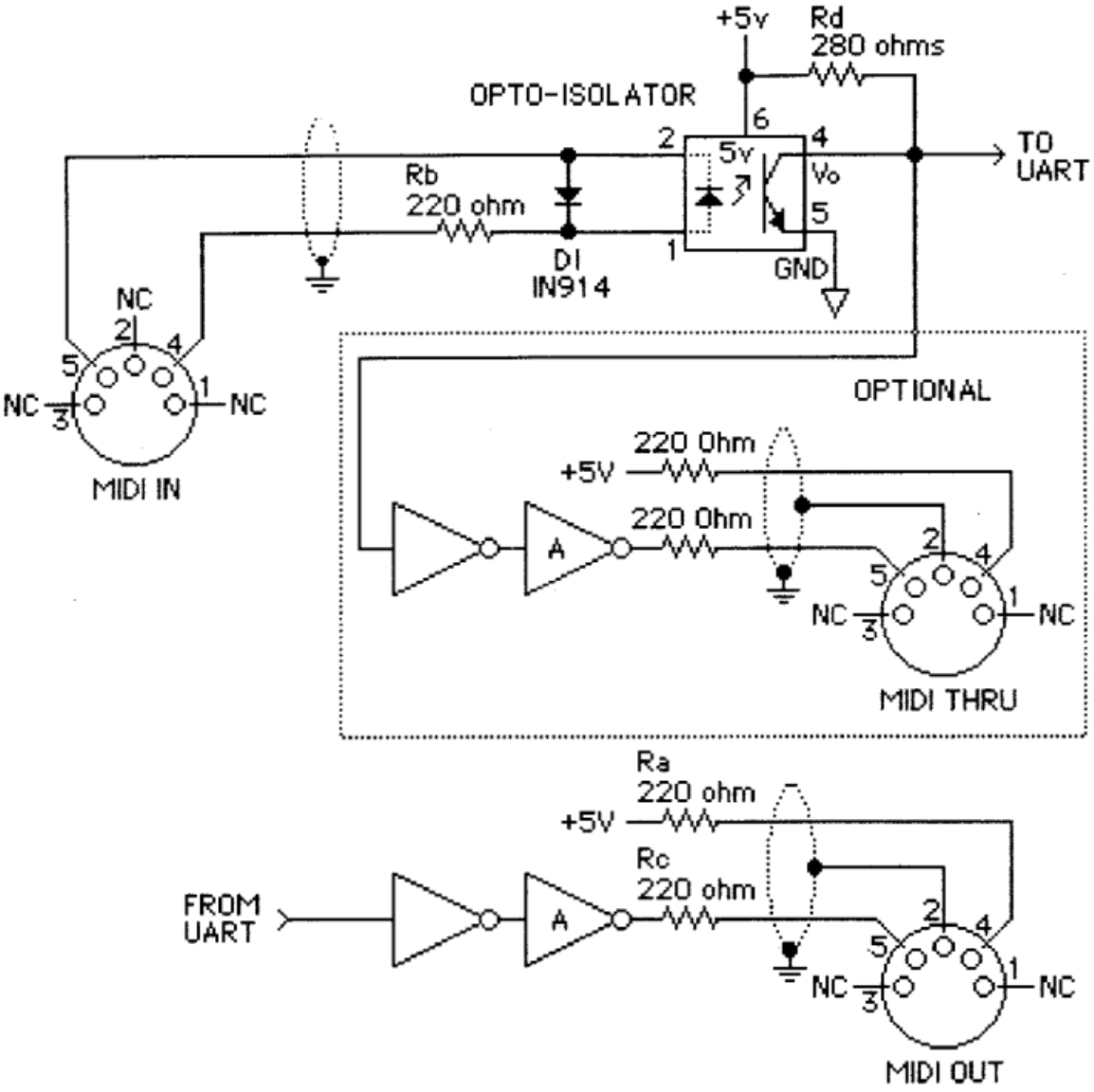


Figura 2.5 – *Hardware* utilizado para equipamentos *midi*. [8]

Através dos cabos *midi* e das ligações *DIN* de 5 pins, Figura 2.6, é possível então fazer a comunicação com o *software* do protocolo, com uma velocidade de transmissão de 31250 bits por segundo (bps) [3].

Os controladores *midi* normalmente têm uma porta de entrada e de saída, *midi in* e *midi out*, USB, e também alguns deles têm a porta *Thru*, o que permite fazer a ligação em cadeia entre vários equipamentos *midi*.

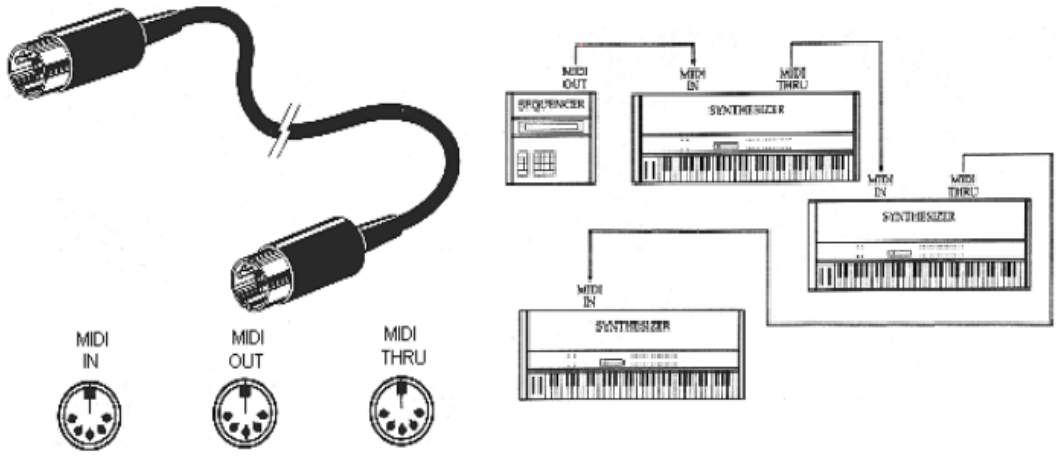


Figura 2.6 – Cabo de ligação às portas *midi* e possíveis ligações [7].

A mensagem transmitida tem sempre o seu tempo de processo, esta tem que ser recebida, processada e ser enviada para o módulo de som. Visto assim podemos dizer que existem duas mensagens muito comuns neste tipo de equipamento, ativar nota e desativar nota [6][7].

Uma grande vantagem dos equipamentos que contenham a tecnologia *midi* é que estes conseguem simular qualquer som de qualquer instrumento musical, e para o fazer basta apenas um controlador *midi*, sintetizador ou sequenciador [7]. A Figura 2.7 representa o tipo de mensagens enviadas e recebidas ao ser pressionado ou acionado um botão de pressão ou um potenciómetro.

TIMESTAMP	IN	PORT	STATUS	DATA1	DATA2	CHAN	NOTE	EVENT
000076EA	9	3	90	3C	53	1	C 4	Note On
000078A6	9	3	80	3C	12	1	C 4	Note Off
00007939	9	3	90	3E	54	1	D 4	Note On
00007A44	9	3	80	3E	30	1	D 4	Note Off
00007B09	9	3	90	3F	21	1	Eb 4	Note On
00007B93	9	3	80	3F	40	1	Eb 4	Note Off
00007D87	9	3	90	42	37	1	F♯ 4	Note On
00007F88	9	3	80	42	01	1	F♯ 4	Note Off
000083DD	9	3	90	3E	3C	1	D 4	Note On
00008550	9	3	80	3E	05	1	D 4	Note Off
00008622	9	3	90	42	44	1	F♯ 4	Note On
000087BB	9	3	80	42	08	1	F♯ 4	Note Off
00008D48	9	3	90	3C	56	1	C 4	Note On
00008D48	9	3	90	3F	32	1	Eb 4	Note On
00008D52	9	3	90	37	2A	1	G 3	Note On
00008D9B	9	3	80	3F	18	1	Eb 4	Note Off
00008DA1	9	3	80	37	12	1	G 3	Note Off
00008DA5	9	3	80	3C	3C	1	C 4	Note Off
00008DF2	9	3	90	3F	43	1	Eb 4	Note On
00008DF2	9	3	90	3C	55	1	C 4	Note On

Figura 2.7 - Exemplo típico das mensagens *midi* [7].

Os controladores *midi* através da ligação USB (*universal serial bus*), estes permitem evitar uma placa de som com a entrada *midi*, que por muita das vezes é dispendiosa. Os controladores por ligação USB precisam sempre de um programa no computador que faça a conversão digital para analógico.

O funcionamento de um controlador via USB é o seguinte:

- Ligar o controlador ao computador via USB;
- O computador corre o programa;
- O controlador *midi* irá controlar esse mesmo programa;
- O programa gere informação digital que irá representar o som;
- A placa de som irá transformar essa informação digital em analógico;
- O sinal analógico irá por fim sair nas colunas de som.

Assim através dos controladores *midi*, quer pelo cabo *midi* ou pelo cabo USB, e de potentes computadores é possível evitar sintetizadores de grande porte, que tornavam quase impraticável a deslocação de um equipamento para qualquer local.

2.6. Sistemas *open source hardware e software*

O sistema *open source hardware* é quando toda a arquitetura física desenvolvida é conhecida. Assim é possível qualquer utilizador copiar ou modificar o equipamento, com o propósito de voltar a partilhar, sem qualquer benefício lucrativo.

Exemplo de sistemas de *open source hardware*:

- *Arduino* – Plataforma com microcontrolador [web5];
- *Openmoko* – Projeto de desenvolvimento de telemóveis [web6];
- *Simputer* – Alternativa de baixo custo a computadores portáteis [web7];
- *Raspberry Pi* - Alternativa de baixo custo a computadores [web8].

O sistema *open source software* são programas desenvolvidos onde todo o seu código fonte e arquitetura são conhecidos.

Desta forma qualquer utilizador poderá copiar, utilizar e alterar o código, com o propósito de voltar a partilhar, sem qualquer benefício lucrativo.

Exemplo de sistemas de *open source software*:

- *Mozilla Firefox* – Navegador de Internet [web9];
- *Linux* – Sistema Operativo [web10];
- *OpenOffice* – Aplicativos de Escritórios [web11];
- *Android* – Sistema Operativo [web12];

- *Pure Data* – Ambiente de programação gráfico para áudio, vídeo e processamento gráfico [web13].

Estes dois sistemas regem-se através de várias particularidades [web7]:

- **Distribuição livre** – a licença não pode restringir ninguém, proibindo que se venda ou doe o *software* a terceiros;
- **Código Fonte** – O programa precisa obrigatoriamente de incluir código-fonte e permitir a distribuição tanto do código-fonte quanto do programa já compilado;
- **Trabalhos Derivados** – A licença deve permitir modificações e obras derivadas que possam ser redistribuídas dentro dos mesmos termos da licença original;
- **Integridade do código do autor** - A licença pode proibir que se distribua o código-fonte original modificado desde que a licença permita a distribuição de atualizações com a finalidade de modificar o programa em tempo de construção;
- **Não discriminação contra pessoas ou grupos** - A licença não pode discriminar contra pessoas ou grupos;
- **Não discriminação contra áreas de utilização** - A licença não pode restringir os usuários de fazer uso do programa em uma área específica;
- **Distribuição da licença** - Os direitos associados ao programa através da licença são automaticamente repassados a todas as pessoas às quais o programa é redistribuído sem a necessidade de definição ou aceitação de uma nova licença;
- **Licença não pode ser específica a um produto** - Os direitos associados a um programa não dependem de qual distribuição em particular aquele programa está inserido. Se o programa é retirado de uma distribuição, os direitos garantidos por sua licença continuam valendo;
- **Licenças não podem restringir outro *software*** - A licença não pode colocar restrições em relação a outros programas que sejam distribuídos junto com o *software* em questão;
- **Licenças devem ser neutras em relação as tecnologias** - Nenhuma exigência da licença pode ser específica a uma determinada tecnologia ou estilo de interface.

O *Arduino* e o *Pure Data* estão inseridos nestes termos que definem o *open source.*

3. Som e Tecnologia

3.1. Propagação do Som

A onda sonora é gera por uma perturbação inicial, que desencadeia o processo da propagação até chegar aos nossos ouvidos [9][web14][web15][3][4].

O meio de propagação e que habitualmente ouvimos é o ar. As ondas que se propagam nesse meio são designadas por ondas de pressão ou ondas de compressão/rarefação, Figura 3.1.

Figura 3.1 - Ilustração da vibração das moléculas [web14].

As ondas quando se propagam geram pressões diferentes no ar, isto faz com que as partículas se movimentem. Este movimento das partículas gera a compressão, quando a pressão entre elas é maior e a rarefação, quando a pressão entre elas é menor [9][web14][web15] [3][4].

A propagação do som é regida pelo princípio de Huygens. Este considera que qualquer ponto de um meio transmissor à distância Δx tende a imitar a origem, com um atraso: $\Delta x = \frac{\Delta x}{c}$.

Sendo assim, a relação entre as amplitudes vibratórias em dois pontos com distância Δx é:

$$y(x + \Delta x\,, t) = y(x, t - \Delta t) \Leftrightarrow$$

$$\Leftrightarrow y(x,t-\frac{\Delta x}{c})$$

Através do princípio de Huygens foi possível verificar os seguintes fenómenos da forma de onda [9][web14][web15] [3][4].

Fenómenos da forma de onda

Reflexão

A reflexão, Figura 3.2, do som segue muito a norma de, o ângulo de incidência é igual ao ângulo de reflexão. Este mesmo comportamento pode ser observado também nos feixes de luz. A onda refletida pode vir a interferir com a onda original (incidente), tanto pode vir a aumentar a sua amplitude como a diminuí-la [9][3][4].

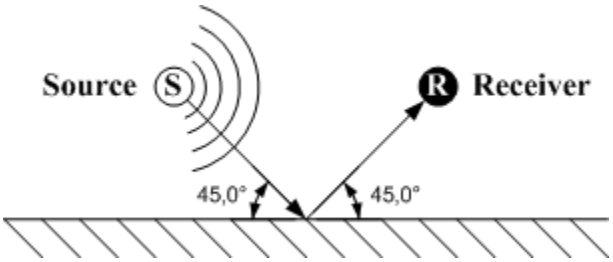


Figura 3.2 - Exemplo de uma onda refletida [10].

Interferência

A interferência, Figura 3.3, acontece quando duas ondas se encontram no mesmo meio. Estas tanto podem ser construtivas ou destrutivas. Quando as ondas estão na mesma fase estas são chamadas de ondas construtivas, quando estão em oposição de fases são ondas destrutivas [10][3][4].

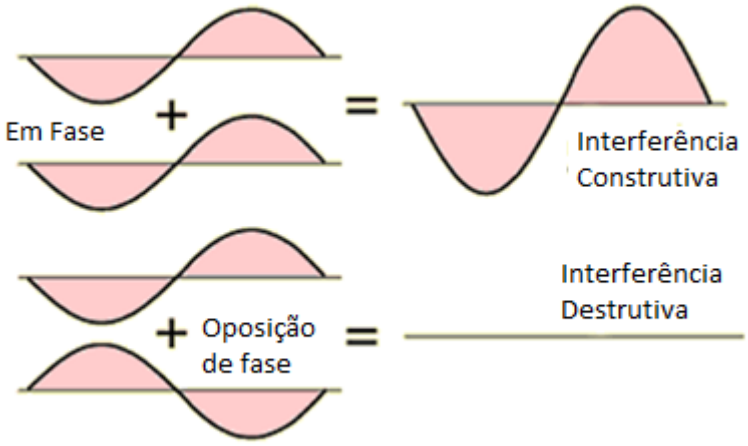


Figura 3.3 - Exemplo de ondas construtivas e destrutivas [10]

Difração

A difração, Figura 3.4, de uma onda é quando a onda consegue passar por locais não tão diretos. Ou seja, quando uma música está a ser reproduzida numa certa sala, ela consegue ser ouvida noutra sala através da difração da onda, pois esta contorna os obstáculos [10][3][4].

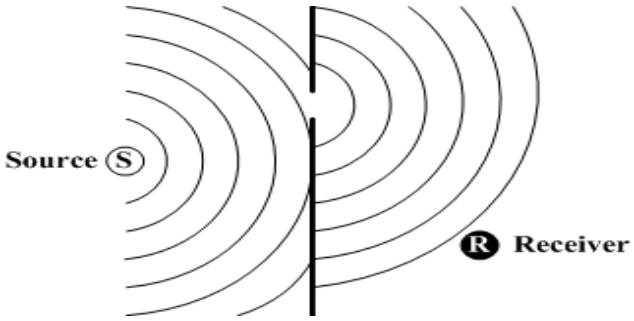


Figura 3.4 - Exemplo de uma difração de uma onda [10].

Ondas Longitudinais

Numa onda longitudinal, Figura 3.5, as partículas do meio deslocam-se paralelamente à propagação da [10].

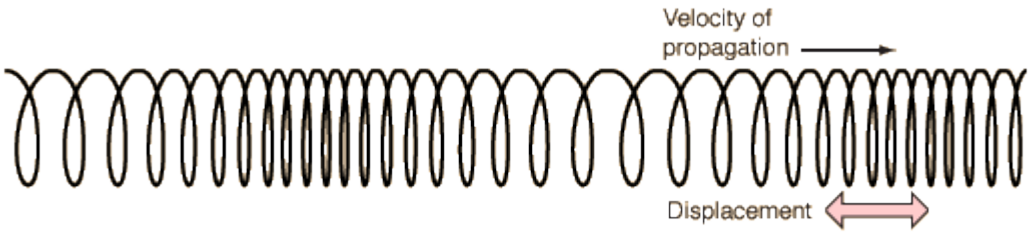


Figura 3.5 - Exemplo de uma onda longitudinal [10].

Ondas Transversais

Numa onda transversal, Figura 3.6, as partículas do meio deslocam-se perpendicularmente à propagação da onda [10].

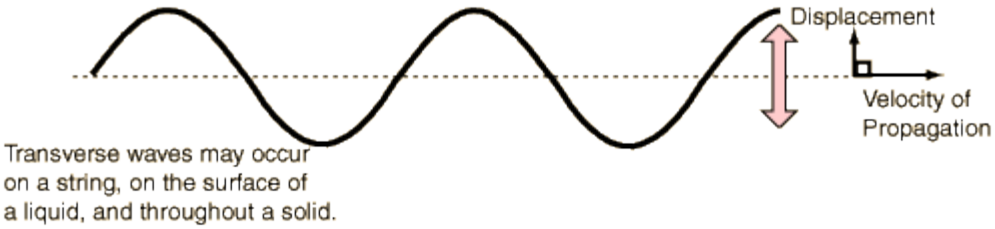


Figura 3.6 - Exemplo de uma onda transversal [10].

3.2. Série de Fourier

O matemático *John Baptiste Fourier*, no século 18 determinou que, tendo uma função periódica $f = x(t)$ de período T, $x(t)$ pode ser representada por uma série, série de *Fourier* [9][3][4].

$$x(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) + \sum_{n=1}^{\infty} B_n \sin\left(\frac{2\pi nt}{T}\right) \quad (1)$$

Os coeficientes da série são dados por:

$$A_0 = \frac{1}{T} \int_{-T/2}^{T/2} x(t)\, dt \quad (2)$$

$$A_n = \frac{2}{T} \int_{-T/2}^{T/2} x(t) \cos\left(\frac{2\pi nt}{T}\right) dt \quad (3)$$

$$B_n = \frac{2}{T} \int_{-T/2}^{T/2} x(t)\, sin\left(\frac{2\pi nt}{T}\right) dt \quad (4)$$

E a frequência é dada por:

$$w_n = \frac{2\pi n}{T} \tag{5}$$

A expressão (1) diz que através das somas das frequências múltiplas da frequência fundamental, ou seja, f,2f,3f…pode compor um sinal periódico de frequência f [3][4].

Outra forma mais compacta da série de *Fourier* é:

$$x(t) = A_0 + \sum_{n=1}^{\infty} A_n \sin\left(\frac{2\pi nt}{T} + \emptyset_n\right) \tag{6}$$

Esta expressão trata-se de uma soma de componentes harmónicos, com amplitudes e fases adequadas.

A série de *Fourier* é utilizada na música para fazer a passagem do sinal de um domínio temporal para o domínio da frequência.

3.3. Transformada de Fourier e Frequência fundamental

O som de uma nota reproduzida por um instrumento musical é chamado de som musical. O som musical é composto por uma série de sinusoides da frequência fundamental e várias frequências harmónicas [11][3][4].

A frequência fundamental é conhecida por ser a sinusoide com a frequência mais baixa. As harmónicas de uma onda é uma frequência componente do sinal, que é um múltiplo inteiro da frequência fundamental. Por exemplo, uma frequência fundamental que seja 100Hz, a 2ª harmónica a 200Hz, a 3ª a 300Hz, etc… Existem certos instrumentos, como o Oboé, que a amplitude da frequência fundamental é ligeiramente mais baixa que as harmónicas [11][3][4].

Através das transformadas de *Fourier,* é possível visualizar-se rapidamente as componentes do sinal. Isso é feito retirando a frequência do sinal num intervalo de tempo, essa técnica é chamada de *Fast Fourier Transform (FFT)*. Esta técnica devolve um conhecimento bastante pormenorizado do sinal.

Na Figura 3.7, mostra-se um exemplo de uma FFT, esta foi feita ao som de um Oboé, podendo observar-se também que a frequência fundamental tem uma amplitude menor que a 2ª harmónica.

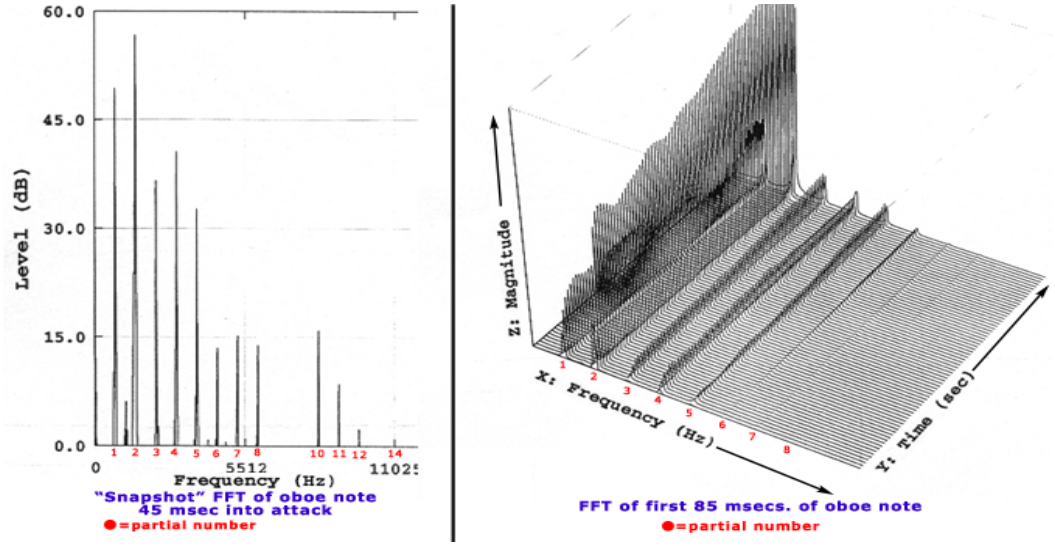


Figura 3.7 - Exemplo de um FFT ao som de um Oboé [9].

Com o desenvolvimento dos computadores e da música digital, a análise de *Fourier* tornou-se no método mais comum para definir as frequências e as amplitudes das ondas sonoras. Assim permite fazer a mistura cuidadosa de vários sinais, de forma a criar timbres complexos através de um sintetizador, o que não seria possível fazer com apenas um instrumento musical [9].

3.4. Filtros

De uma forma mais genérica, os filtros são dispositivos que selecionam certas frequências de um sinal de entrada e as direciona para a saída, suprimindo certas frequências.

Os sintetizadores usam os filtros praticamente desde a sua aparição e vieram com os quatro tipos de filtros mais básicos, Figura 3.8. Estes encontram-se também nos sintetizadores de hoje mas com já algumas variações.

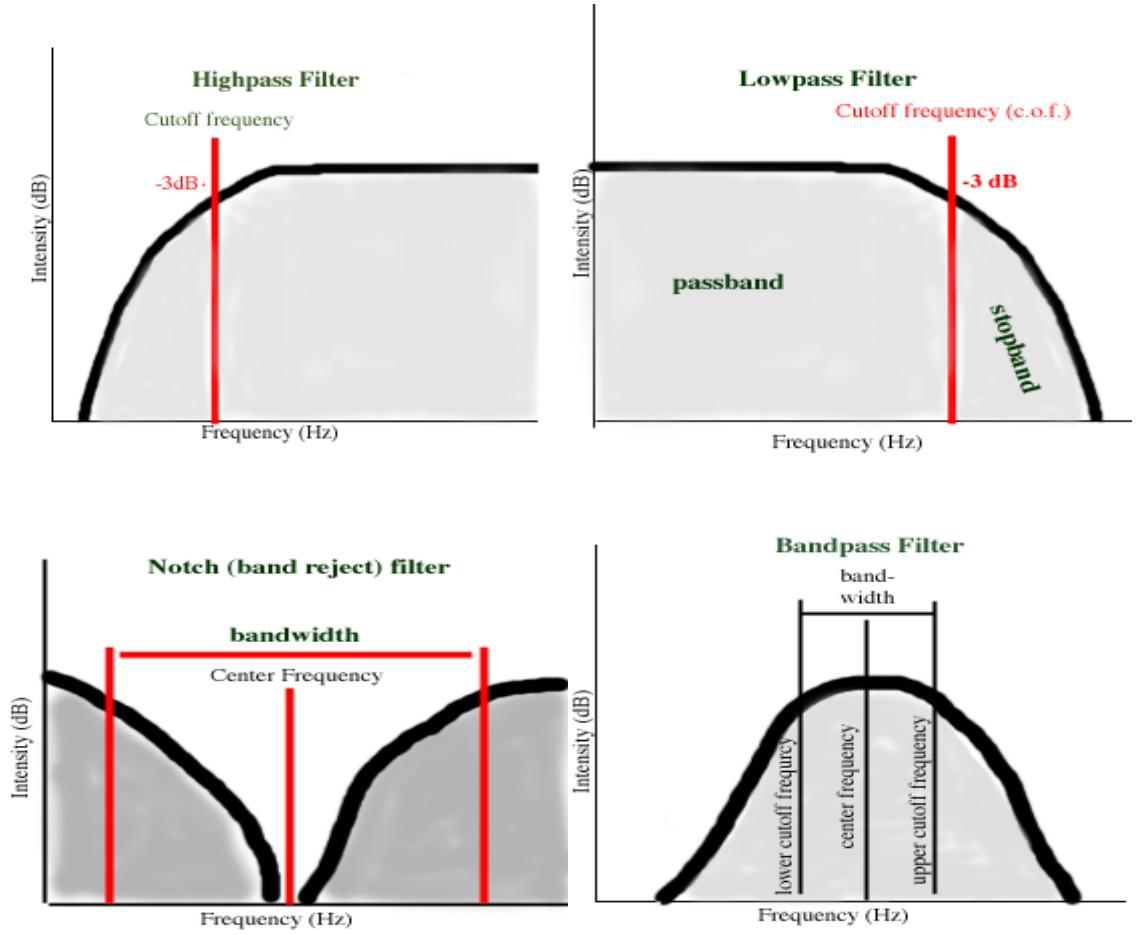


Figura 3.8 - Filtro Passa Alto, Filtro Passa Baixo, Filtro Rejeita Faixa e Filtro Passa Banda [9].

Os filtros podem ser usados para suprimir os ruídos de fundo assim como para a análise de sinais.

Os sintetizadores normalmente usam uma técnica de varrimento da frequência de corte, isto para garantir um som ao longo do tempo. Estas frequências são controladas através do gerador de envelope ou de um oscilador.

Os filtros podem ser combinados de forma a atingir o fim desejado. Tanto podem ser ligados em série, Figura 3.9, como em paralelo, Figura 3.10.

Em série a saída de um dos filtros irá alimentar a entrada do outro filtro.

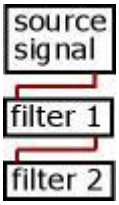


Figura 3.9 - Exemplo de dois filtros em série [9]

Os filtros em paralelo podem ser alimentados por uma única fonte, em que as saídas são depois somadas. Este tipo de ligação é ideal para criar várias áreas de ressonância ou picos num sinal sonoro [9].

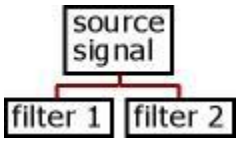


Figura 3.10 - Exemplo de uma ligação em paralelo [9]

Com a junção de vários filtros passa banda, é possível modelar um sinal sonoro de forma a ouvir-se a vogal "A". Para tal as frequências (Hz),amplitude do sinal (dB) e a largura de banda (Hz) seria:

Tabela 3.4.1 - Modelação de um sinal sonoro para se ouvir a vogal "A", método analógico [9].

Freq(Hz)	Amp(dB)	Larg.Band(Hz)
800	0	80
1150	-6	90
2900	-32	120
3900	-20	130
4950	-50	140

3.5. Tecnologia dos sintetizadores

Existem vários tipos de tecnologias usadas nos sintetizadores. No entanto, como já foi referido no capítulo 2.1 existem duas que têm vantagens sobre as outras, estas são, o método analógico e o método digital.

3.5.1. Método Digital

O método digital como já foi referenciado no capítulo 2.1, utiliza a matemática para a modulação do som. Este método utiliza as seguintes técnicas, síntese aditiva, modulação em frequência, modulação física e granular.

Síntese Aditiva

O teorema de *Fourier* é a base da síntese aditiva. Como o próprio nome sugere, a síntese aditiva é uma soma de ondas normalmente sinusoidais, Figura 3.11. O objetivo da síntese aditiva é fazer com que em determinadas situações a somas das sinusoides, criem um único som rico ao ouvido humano [3][4].

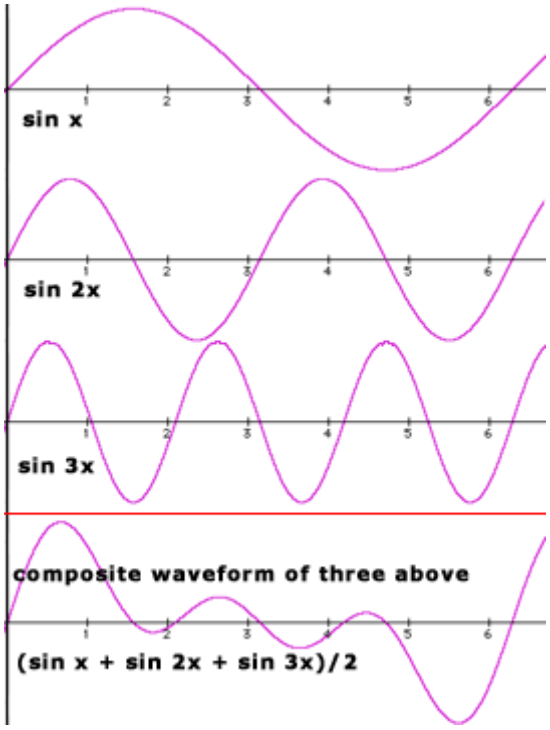


Figura 3.11 – Soma de 3 sinais sinusoidais [9].

Esta ideia já é utilizada há muito tempo através dos órgãos de tubos. Este tipo de órgãos quando tocado cria sons simples mas quando esses sons simples se somam produzem um som muito rico musicalmente [3][4].

Como esta técnica consiste na soma de sinusoides oscilatórias em que a frequência e a amplitude variam no tempo e os parâmetros de controlo são determinados através de análises espectrais dos sons reais, faz com que esta técnica seja adequada para a emulação de sons dos equipamentos musicais [3][4].

Modulação em Frequência

A técnica FM normalmente usa um sinal periódico (modulador) para modelar a frequência de outro sinal (transmissor). Se sinal de modulação estiver na gama de som correta, então irá resultar numa alteração significante no som do sinal transmitido. Cada sinal FM requere no mínimo dois geradores de sinais. Estes geradores são normalmente referidos de operadores e podem ser operados de formas diferentes [3].

Os sistemas mais sofisticados que usem a tecnologia FM podem ter 4 a 6 operadores de sinal, e podem conter envelopes que permitem fazer alterações ao sinal. Este sistema começou por ser implementado nos sintetizadores analógicos. As implementações mais recentes desta tecnologia já têm sido em sintetizadores digitais [3].

As técnicas FM são bastante uteis para criar sons através dos sintetizadores. No entanto é mais fácil fazer a criação de sons através da técnica *Digital Sampling*.

Os sistemas que contenham esta técnica são chamados de sintetizadores *wavetable*.

Modulação Física

Como o próprio nome indica, esta técnica consiste na integração temporal das equações físicas do sistema. Este método é um método puramente matemático e não um método de tratamento de sinais [3][4].

A grande vantagem da modulação física relativamente às modulações convencionais é a melhor nitidez nos sons, isto é, consegue produzir os sons dos instrumentos mais próximo da realidade. No entanto apesar da existência de processadores muito potentes, a grande dificuldade e a não utilização deste método para efeitos de sintetizadores é devido à realização de instrumentos polifónicos que reajam em tempo real [3][4].

Síntese Granular

A síntese granular partilha a mesma ideia da síntese aditiva, pois o objetivo é criar sons mais complexos a partir da soma de sons mais simples. Neste caso a criação dos sons é a partir de amostras muito pequenas, normalmente amostras medidas em milissegundos, o que lhe chamam de grãos [3][4].

O som é dividido em segmentos sobrepostos, este processo é chamado de tempo de granulação, isto é semelhante ao que acontece quando tempos várias imagens fixas e fazendo uma sequência dessas imagens produz a sensação de movimento. Com esta técnica de pequenas amostras é possível fazer vários sons alterando a velocidade e a ordem dos segmentos [3][4].

3.5.2. Método Analógico

O método analógico como já foi referenciado no capítulo 2.1, é o mais convencional pois consegue gerir bem o tempo e a capacidade de cálculo e memória. Este método é constituído pelas técnicas, síntese aditiva, subtrativa e *wavetable*.

Síntese Subtrativa

Esta técnica parte de uma onda complexa que contem múltiplas frequências. Com isto vai efectuar uma filtragem de forma a conseguir eliminar ou atenuar as zonas do espetro [3][4].

Desta forma é possível obter sons muito semelhantes ao dos instrumentos reais assim como a voz humana. Esta técnica assim como a aditiva é mais eficaz quando está articulada a uma etapa de análise [3][4].

A síntese subtrativa é normalmente aplicada quando se trata da síntese de voz [3][4].

WaveTable

Os sintetizadores que usem a técnica, *wavetable*, têm guardado na memória várias amostras de sons digitais e só são reproduzidos quando forem solicitados pelo utilizador. Este sistema normalmente tem inserido várias técnicas especiais, assim como, repetição das amostras, envelopes, *pitch shifting* (técnica de mistura de sons), interpolação matemática e filtros digitais.

Este tipo de sintetizadores é o que tem vindo a ser mais utilizado nos controladores *midi*.

Repetição e Envelope

Uma das principais técnicas utilizadas pelos sintetizadores é a repetição das amostras que estão já inseridas no sintetizador. Ao permitirem a repetição contínua das amostras, fazem com que seja utilizado menos memória para o controlador. Assim também permitem um processador não tão forte pois tem menos amostras para processar.

A modelação do som consiste em duas partes muito importantes, o ataque na secção e na secção de sustentação. Estas duas partes são bastante importantes pois são elas que vão fazer com que haja dinâmica e intrusão entre os sinais sonoros.

A secção de ataque é a parte inicial do som, é importante pois é o que dá a entrada da alteração do sinal sonoro, o que pode acontecer quase de forma instantânea, depende dos batimentos por minuto do som.

A secção de sustentação é a parte em que a dinâmica do som é menos alterada, ou seja, a alteração já foi efetuada, é só manter a característica do sinal.

No exemplo seguinte é possível identificar a altura do ataque e a altura da sustentação do sinal, Figura 3.12. Como se pode observar as características da onda permanecem constantes na zona de sustentação, perdendo a amplitude conforme o tempo.

A zona de ataque é a que tem a amplitude mais elevada, pois é a entrada de um novo sinal na qual a onda é somada com a onda anterior até chegar à zona de sustentação.

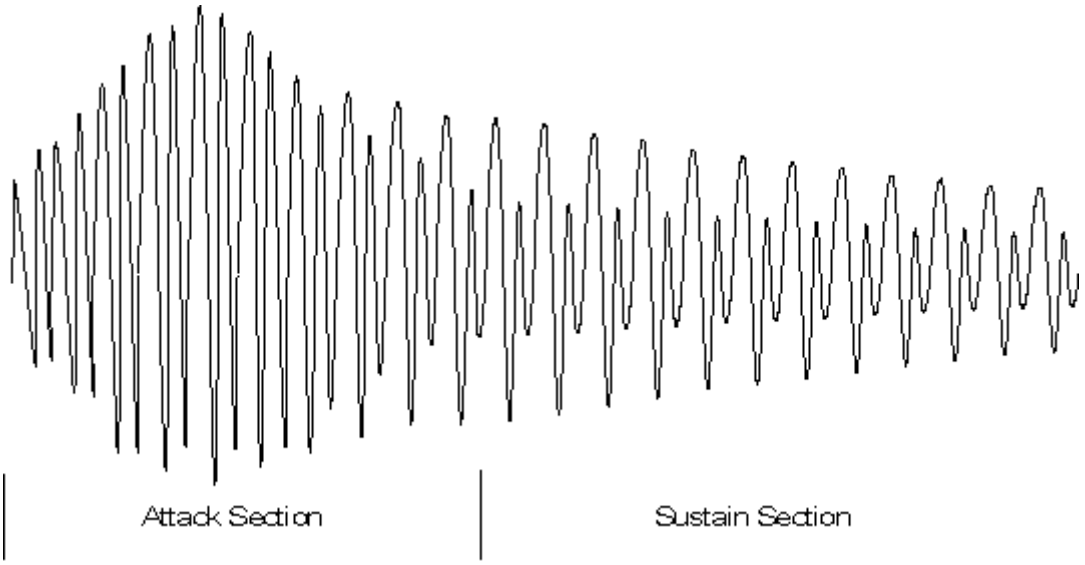


Figura 3.12 - Ataque e sustentação de uma onda [12].

Nos sintetizadores *wavetable*, uma grande parte da memória é poupada pois, o sistema consegue guardar a zona de sustentação do sinal e ficar a repeti-la continuamente até o utilizador desejar. Na Figura 3.13, é possível observar-se como esta técnica é utilizada.

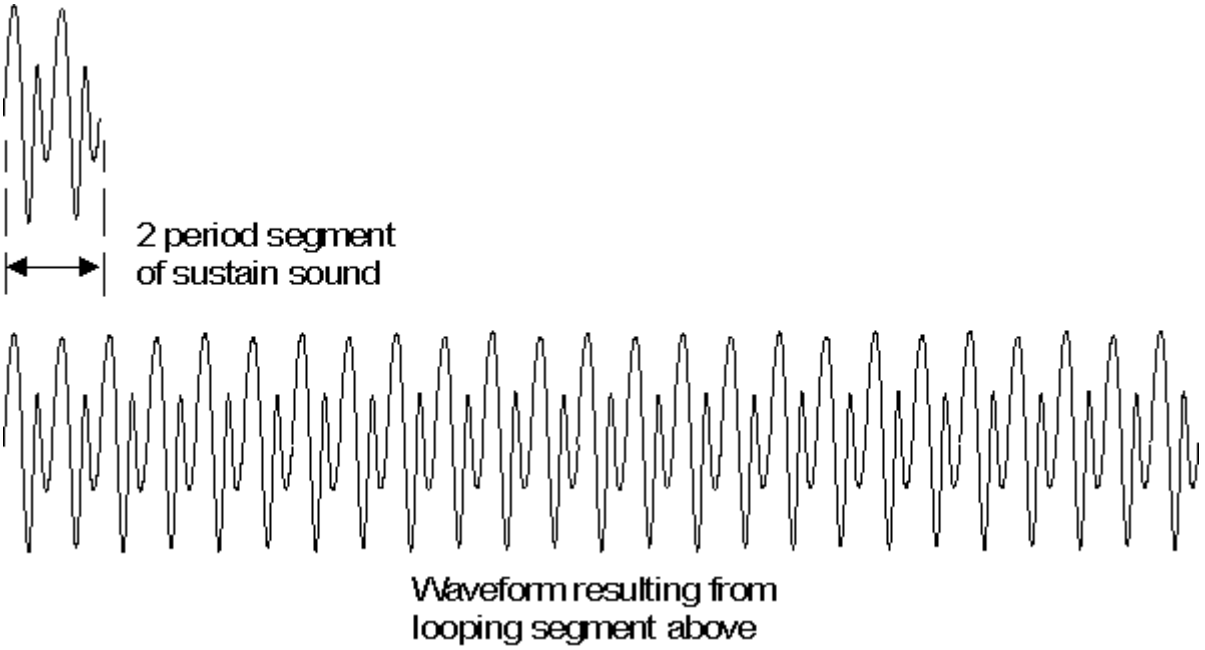


Figura 3.13 - Repetição contínua de uma amostra [12].

Se a amostra de som tiver uma boa zona de sustentação então, a repetição contínua do sinal será uma boa aproximação dessa amostra.

A maioria dos instrumentos musicais de cordas, as características do sinal permanecem quase constante durante a zona de sustentação, enquanto a sua amplitude vai diminuindo. É possível fazer esta simulação num sintetizador, basta multiplicar um segmento de amostras com um fator de ganho decrescente

durante a reprodução para se obter essa diminuição da amplitude (*envelope*).

O *envelope* é normalmente constituído por segmentos lineares. Um exemplo de envelope é o modelo ADSR (*Attack-Decay-Sustain-Release)*, como mostra a Figura 3.14 e Figura 3.15.

Figura 3.14 - Exemplo típico de um ADSR *Envelope* [12].

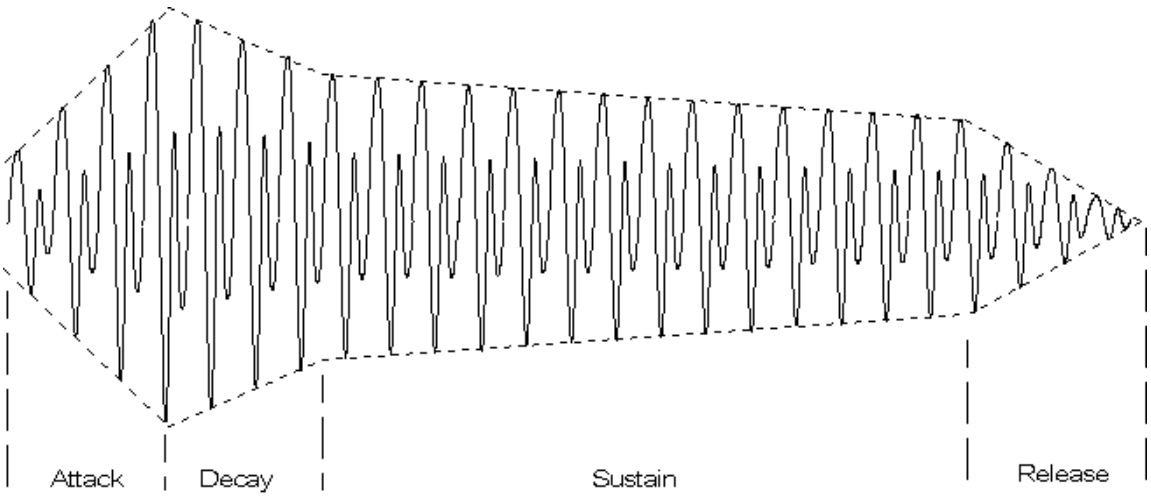


Figura 3.15 - Modelo ADSR *Envelope* aplicado a uma amostra [12].

Um sintetizador *wavetable* armazena tanto a parte de ataque como a parte de sustentação. A parte do ataque é reproduzida uma única vês no inicio do som, enquanto a parte de sustentação fica a repetir-se continuamente até que a amostra acabe, ou que seja finalizada pelo utilizador. A função de *envelope* serve para fazer a modelação de um determinado instrumento e esta pode ser implementado durante a reprodução dos sons.

A reprodução da onda inicial (parte do ataque do *envelope*) começa quando a mensagem de nota ativa é recebida. Os comprimentos de segmento de ataque e de queda são normalmente fixos para um determinado instrumento. A zona de sustentação repete-se continuamente durante as amostras, no entanto para fazer o efeito de instrumentos de cordas, é aplicada uma ligeira queda durante a sustentação até chegar a mensagem de nota desativa.

Amostras curtas e de grande variação

Nos parágrafos anteriores falou-se das repetições das amostras, em que se dividia em zona de ataque e em zona de sustentação. No entanto para amostras curtas ou amostras com grandes variações, não são apropriadas as repetições contínuas do sinal. Estas amostras mais curtas, como por exemplo o bater de um tambor, são amostras de uma só batida, não sendo necessário fazer uma repetição do som.

Pitch Shifting

Pitch shifting é uma técnica utilizada nos controladores *midi* e em instrumentos de cordas. Esta permite fazer notas/sons, aumentando ou diminuindo, meio-tom ou um tom das notas já definidas à partida. Ou seja, se tivermos por exemplo a nota C, é possível através do *pitch shifting* reproduzir a nota C# ou a nota D.

O *pitch shifting* também pode alterar o tempo da nota, aumentando ou diminuindo o tempo em que o som está definido.

Mais uma vez esta técnica é boa para o sintetizador porque permite poupar a memória de várias amostras com tons diferentes.

Oversampling and Interpolation

Tanto o *oversampling* e a *interpolation* (sobre amostragem e interpolação) servem para atenuar ou acentuar a distorção de um som. Apesar de servirem para a mesma situação a sobre amostragem consegue ter uma precisão melhor que a interpolação. Assim a combinação das duas técnicas acaba por ter excelentes resultados, pois para os sons que precisem de melhor precisão é utilizado a sobre amostragem, o que acaba por fazer uma utilização mais forte da memória. A interpolação é usada para sons mais simples.

LFOs for Vibrato and Tremolo

Vibrato e *tremolo* são efeitos normalmente utilizados em instrumentos acústicos. O *vibrato* é uma modulação de baixa frequência de uma nota, enquanto o *tremolo* é a modulação da amplitude do som.

Estes dois efeitos conseguem ser simulados através dos sintetizadores, basta implementar *LFO's* (osciladores de baixa frequência) no sintetizador, que servem para modelar a amplitude ou os filtros de frequência.

Os *vibratos* e *tremolos* têm a tendência para aumentar o poder de sustentação do sinal, fazendo o sinal perdurar durante mais tempo. Estas técnicas são bastante usadas nas guitarras para conseguirem ter um maior tempo de nota e ter-se um tempo maior de sustentação da nota. Pode-se dizer que cada vez que se toca numa corda primeiramente vem o som forte (ataque), o *tremolo* e *vibrato* evitam esse efeito. Para simular este efeito num sintetizador aplica-se um *envelope* ao *LFO*.

Samples and sampling

Os sintetizadores têm também a capacidade de gravar áudio através do conversor analógico para digital (ADC). Como mostra a seguinte imagem, ele grava a amplitude do sinal da forma de onda analógica. Muitas das vezes o ADC já nem se encontra no sintetizador mas sim no computador através da placa de som.

Uma amostra é definida através das leituras instantâneas da amplitude dos sinais reais ou artificiais. As características da onda assim como a forma de onda, fase, frequência, etc, não são guardadas na amostra, todas estas características serão recriadas quando for feita a leitura sequencial das amplitudes.

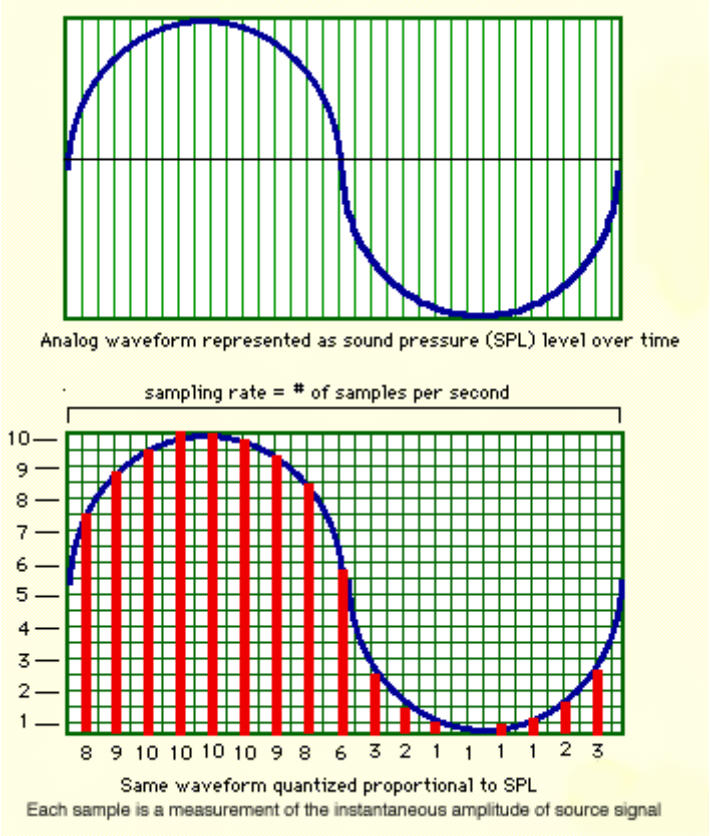


Figura 3.16 - Exemplo de uma medição de amplitudes [9].

As amostras são retiradas com um intervalo de tempo regular. Esta taxa de medição é intitulada de taxa de amostragem ou frequência de amostragem.

4. Implementação do Sistema *Midi*

Neste capítulo é apresentada toda a implementação feita no projeto, desde a sua arquitetura, *hardware*, até ao produto final, o controlador *midi*.

4.1. Arquitetura da plataforma

Esta plataforma foi construída através do *Arduino* Mega 2560 r3, para implementar um controlador de som. Para este tipo de implementação foi necessário abrir todas as portas do *Arduino* a fim de deixar receber e enviar os dados fornecidos pelo *Pure Data* e também pelo equipamento ligado a si.

O *Arduino* está a receber dados de 33 entradas digitais e de 13 portas analógicas.

O funcionamento em tempo real e de rápida resposta do microcontrolador ATMega2560, é uma grande valia para o funcionamento do controlador, pois assim permite não existir qualquer tipo de latência entre o inicio e o fim de cada evento.

A Figura 4.1 e Figura 4.2 apresentam a arquitectura da plataforma e as dependencias do mecanismo, respetivamente, onde está incluído todo o processo necessário para o funcionamento do controlador *midi*, módulo disco rigido, hardware (potênciometros e botões de pressão) e o *Arduino*.

Como mostra a Figura 4.1 a plataforma para além do micro controlador ATMega2560 dispõe ainda dos seguintes módulos:

- Comunicação por USB - garante a comunicação entre o computador e a plataforma, bem como a alimentação energética do sistema;
- Comunicação do tipo UART - através deste módulo que irá ser feita a ligação com o *software* (*Pure Data*) permitindo a comunicação rápida entre os dispositivos a usar;
- Memória interna - para ter alguma liberdade no armanezamento de sons e os programas a querer usar;
- *Hardware* (potênciometros e botões de pressão) - para dar início aos eventos pré-definidos para cada um dos dispositivos.

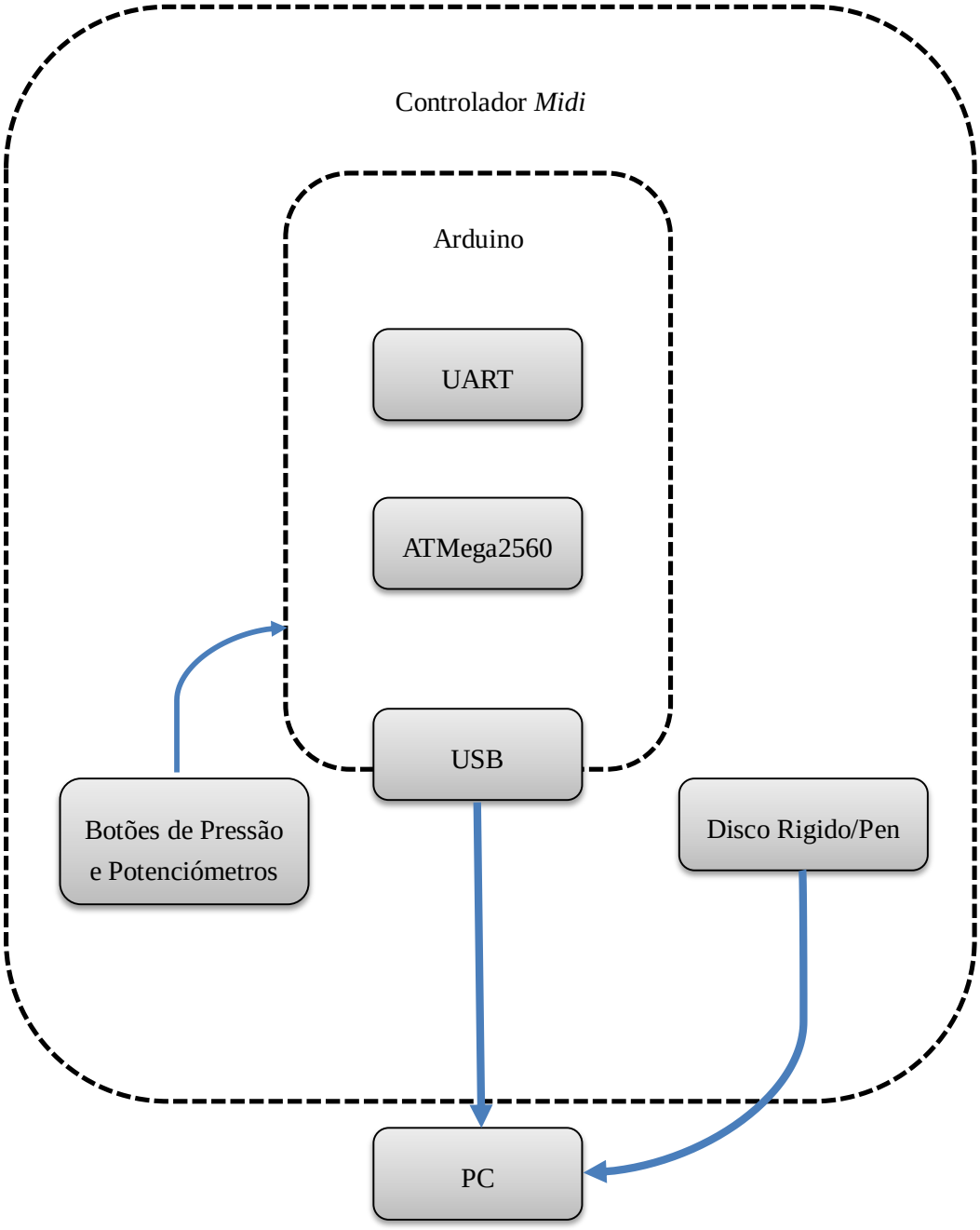


Figura 4.1 - Arquitetura da Plataforma

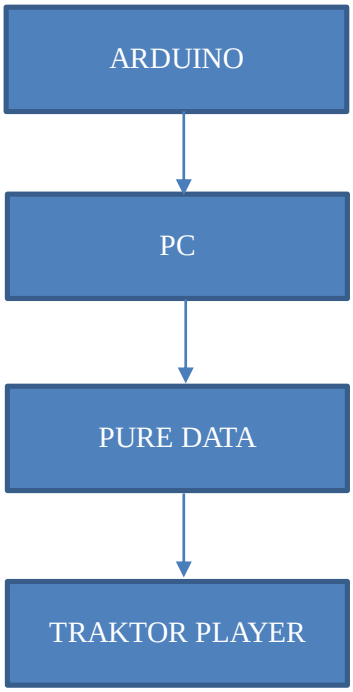


Figura 4.2 – Dependências do Mecanismo

A Figura 4.2 mostra as dependências do mecanismo, ou seja;

- *Arduino* – é no *Arduino* que está implementada toda a parte de *hardware* do controlador *midi*.
- PC – o computador é a base da comunicação entre o *Arduino* e os programas que são necessários para o funcionamento do controlador.
- *Pure Data* – é onde se encontra implementado todo o sistema do controlador *midi*.
- *Traktor Player* – programa de música onde o controlador *midi* opera.

4.2. Arquitetura do sistema

Na fase inicial deste projeto foram definidas algumas funcionalidades que o sistema deveria ter para cumprir os objetivos propostos.

O sistema tem identificado dois tipos de atores, ou seja, o "Arquiteto" e o "Utilizador".

O "Arquiteto" será o projetista do sistema, este poderá configurar a máquina conforme assim o desejar ou até implementar novas funcionalidades ao fazer uma atualização do ficheiro inserido na memória interna (*pen drive*) que se encontra no controlador de som.

A Tabela 4.2.1 tem identificado os módulos utilizados pelo arquiteto assim como as suas funções.

Tabela 4.2.1 - Apresentação dos módulos referentes ao Arquiteto.

Controlador *Midi*	
Arquiteto	
Módulo	**Função**
Configurar	O arquiteto pode configurar todo o sistema, ou seja, pode alterar os sons configurados.
Implementar	O arquiteto pode fazer alterações ao equipamento, implementando novos módulos ao controlador *midi*.
Arquivar	Permite arquivar qualquer tipo de ficheiro para a memória interna (disco rígido) do controlador *midi*.

O "Utilizador" irá tirar partido das funcionalidades que se encontram no controlador de som assim como, modelação sons, gravação, auto, configuração e arquivar ficheiros.

A Tabela 4.2.2 apresenta os módulos que estão ao dispor do utilizador assim como a função de cada um.

Tabela 4.2.2 - Apresentação dos módulos referentes ao Utilizador.

<table>
<tr><th colspan="4">Controlador Midi</th></tr>
<tr><th colspan="4">Utilizador</th></tr>
<tr><th>Módulo</th><th colspan="3">Função</th></tr>
<tr><td rowspan="3">Modelar Sons</td><td rowspan="3">Este módulo está dividido em três funções: volume, velocidade e caixa de som.</td><td>Volume</td><td>Nesta secção é possível controlar o volume do programa que se está a utilizar através dos potenciómetros que estão implementados no controlador midi.</td></tr>
<tr><td>Velocidade</td><td>Nesta secção é possível controlar a velocidade (BPM) das músicas que estão a ser reproduzidas pelo programa de Dj a ser utilizado.</td></tr>
<tr><td>Caixa de Som</td><td>Nesta secção é possível fazer a construção de uma música através dos sons já configurados, nos botões de pressão.

Também é possível utilizar estes sons para fazer acréscimos em tempo real às músicas.</td></tr>
<tr><td>Gravação</td><td colspan="3">Faz a gravação de som através de um microfone para um ficheiro wave.</td></tr>
<tr><td>Auto</td><td colspan="3">Faz a criação de um som de forma aleatória através de dois samplers.</td></tr>
<tr><td>Configurar</td><td colspan="3">Neste módulo é possível fazer o mapeamento dos potenciómetros assim como de alguns botões de pressão.</td></tr>
<tr><td>Arquivar</td><td colspan="3">Permite arquivar qualquer tipo de ficheiro para a memória interna (disco rígido) do controlador midi.</td></tr>
</table>

Na Figura 4.3 está apresentado o diagrama de atividades do controlador *midi*.

Na Figura 4.4 estão apresentados os casos de uso deste projeto. Este projeto está dividido em onze casos de uso em que estes estão divididos por dois atores.

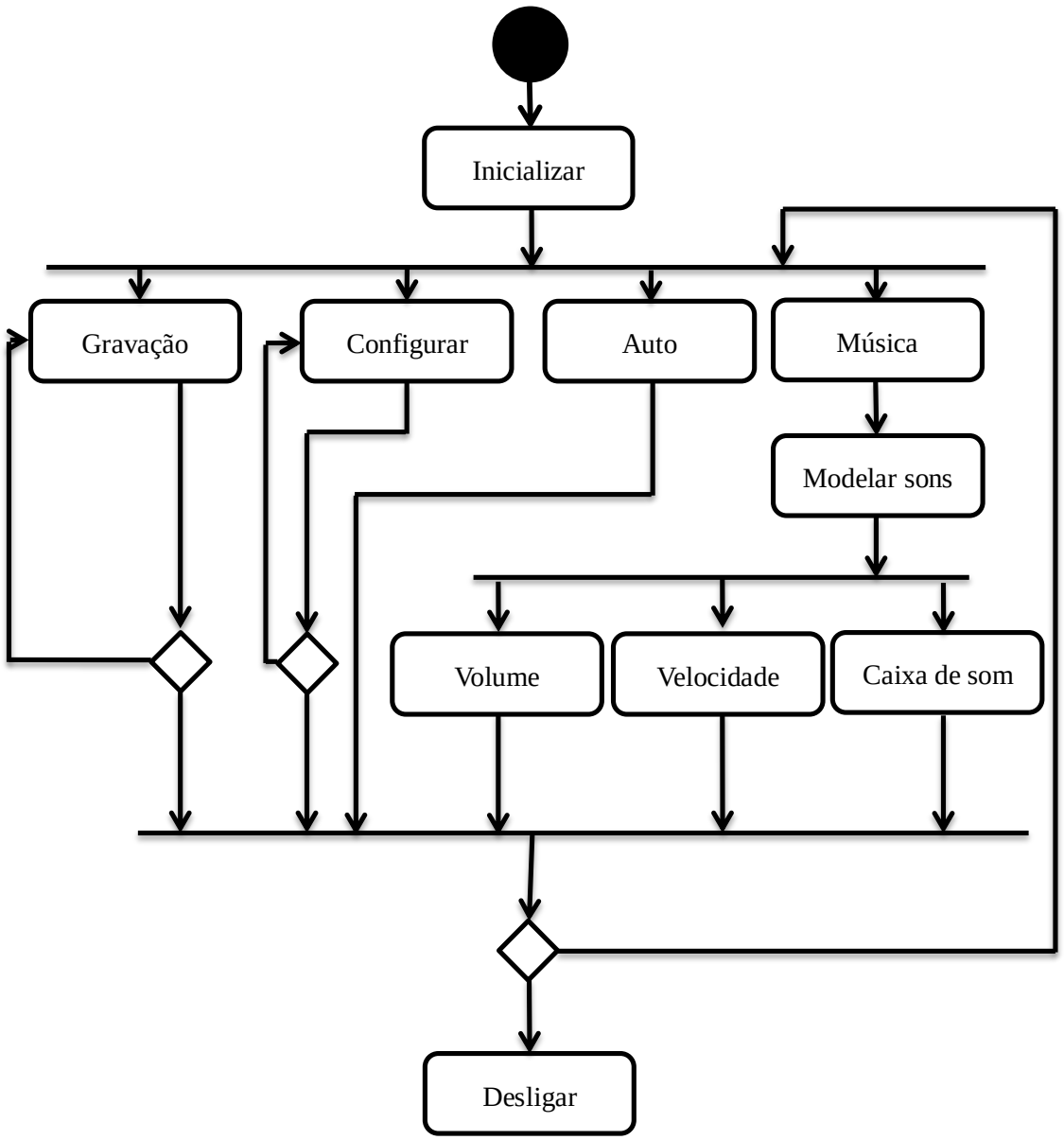


Figura 4.3 - Diagrama de atividade.

Os casos de usos implementados são os seguintes:

Caso 1:

- Nome: Configurar;
- Descrição: Este caso de uso permite ao arquiteto fazer a configuração do sistema;
- Atores envolvidos: Arquiteto;
- Pré-condições: Conhecimento do código implementado e ligação do disco rígido ao

computador;

- Fluxo: O arquiteto pode fazer a alteração dos efeitos sonoros já programados nos botões de pressão.

Caso 2:

- Nome: Implementar;
- Descrição: Este caso de uso permite ao arquiteto fazer a implementação de um sistema novo no controlador *midi*;
- Atores envolvidos: Arquiteto;
- Pré-condições: Conhecimento do código e da implementação do *hardware*;
- Fluxo: O arquiteto pode fazer alterações ao equipamento, implementando novos módulos ao controlador *midi*.

Caso 3:

- Nome: Inicializar;
- Descrição: Este caso de uso permite ao utilizador ligar o controlador *midi*;
- Atores envolvidos: Arquiteto;
- Pré-condições: O equipamento tem que estar ligado por USB a um computador e com o programa que se encontra no disco rígido a funcionar;
- Fluxo: Permite o início da utilização de todo o equipamento e funcionalidades.

Caso 4:

- Nome: Arquivar;
- Descrição: Este caso de uso permite o arquiteto e o utilizador arquivar ficheiros no disco rígido do controlador *midi*;
- Atores envolvidos: Arquiteto e utilizador;
- Pré-condições: Ligar o disco rígido por USB a um computador;
- Fluxo: O arquiteto e o utilizador podem arquivar ficheiros no disco rígido.

Caso 5:

- Nome: Modelar Sons;
- Descrição: Este caso de uso permite ao utilizador fazer alterações às músicas que estão a ser reproduzidas;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. Conhecimento dos efeitos que estão

configurados no controlador;

- Fluxo: Através dos botões de pressão e dos potenciómetros é possível alterar o volume, velocidade e introduzir sons às músicas que estão a ser reproduzidas.

Caso 6:

- Nome: Volume;
- Descrição: Este caso de uso faz a alteração do volume da reprodução;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. Conhecimento dos efeitos sonoros que os vários tipos de volume efetuam, *low*, *mid* e *high*;
- Fluxo: O utilizador pode fazer alterações ao volume da música assim como alterações aos *low's, mid* e *high*. Também pode alterar o volume através do *crossfader*. Este por sua vez alterna o volume dos dois *decks*.

Caso 7:

- Nome: Velocidade;
- Descrição: Este caso de uso altera a velocidade das músicas;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. Conhecimento dos efeitos gerados ao som;
- Fluxo: O utilizador pode acelerar ou desacelerar a velocidade da música (BPM).

Caso 8:

- Nome: Caixa de Som;
- Descrição: Este caso de uso faz com que sejam reproduzidos excertos de sons;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. Saber quais os sons que estão configurados nos botões de pressão;
- Fluxo: Ao pressionar os botões de pressão os sons configurados neles são reproduzidos. Estes podem ser reproduzidos a meio de uma música assim como construir uma música.

Caso 9:

- Nome: Gravação;
- Descrição: Este caso de uso permite ao utilizador fazer a gravação através de um microfone;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. O computador necessita de um microfone;

- Fluxo: Através do microfone é possível fazer a gravação de qualquer som. Após o som estar gravado é possível fazer a regravação ou reproduzir o ficheiro de som.

Caso 10:

- Nome: Auto;
- Descrição: Este caso de uso faz a reprodução de dois *samplers* aleatoriamente;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado;
- Fluxo: Ao carregar no botão de pressão de efeito automático, o controlador *midi* faz a reprodução aleatória de dois *samplers*. Estes ficam em *loop* até o botão seja novamente pressionado.

Caso 11:

- Nome: Configurar;
- Descrição: Este caso de uso permite ao utilizador configurar alguns botões de pressão do controlador *midi*;
- Atores envolvidos: Utilizador;
- Pré-condições: Ter o equipamento inicializado. Conhecimento do programa e o modo de mapeamento;
- Fluxo: O utilizador pode configurar alguns botões de pressão assim como os potenciómetros através do mapeamento dos programas de *Dj's*.

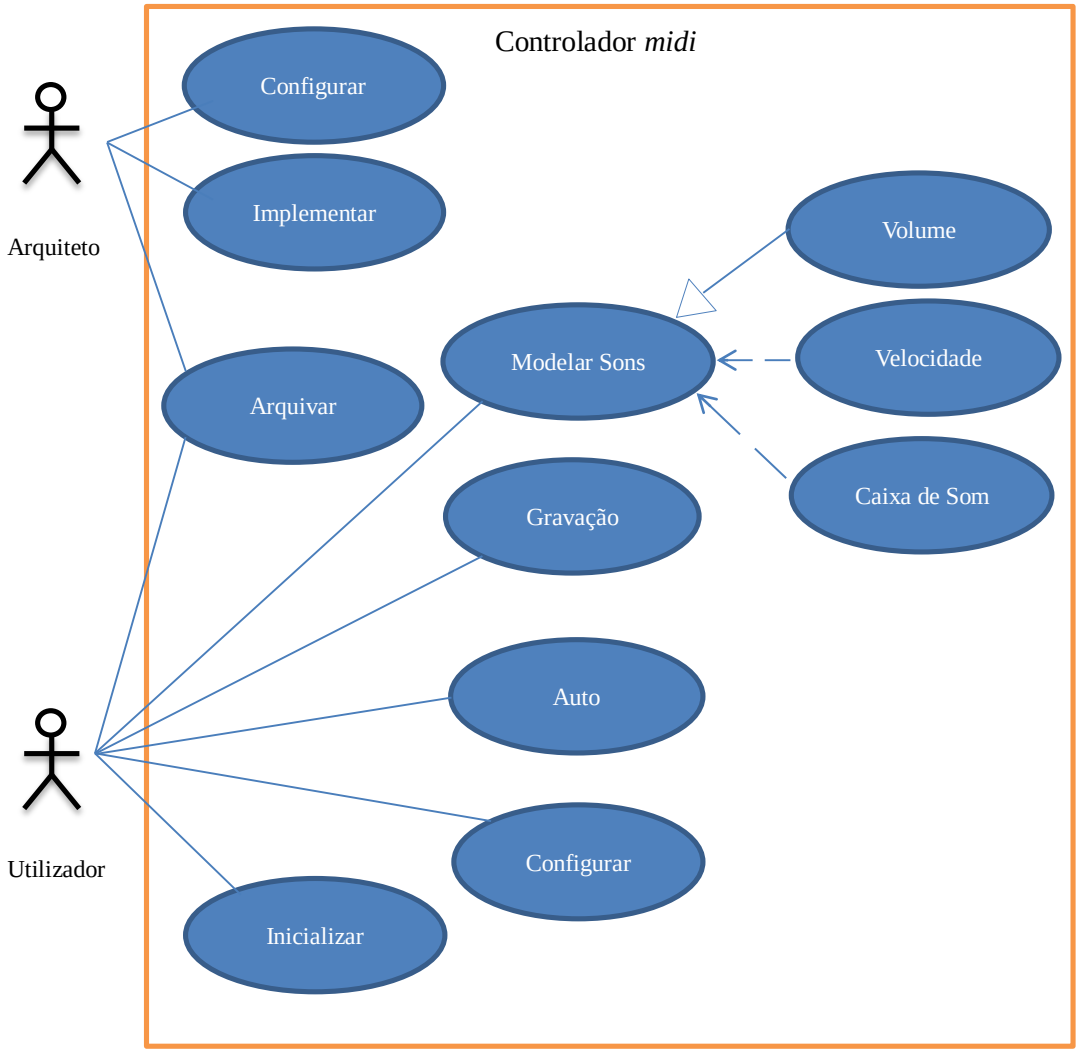


Figura 4.4 – Casos de Usos

Neste subcapítulo apresenta-se o código *Pure Data*. Sendo tratado de uma linguagem gráfica que está em constante desenvolvimento devido a ser *open source*, faz com que o criador da linguagem, *Miller Puckette*, tenha dividido o código em várias secções: *dataflow, audio e patch management*. Tendo cada secção algumas subsecções.

Dataflow

- *Glue - General dataflow control*
- *Math - Mathematical operations*
- *Time - Time-related operations*
- *Midi - Midi Input/Output*
- *Tables - Table and array management*
- *Misc - Objects that don't fit any previous category*

Audio

- *Audio Glue - General audio control*
- *Audio Math - Mathematical operations*
- *Audio Oscillators and Tables- Audio generators and table readers*
- *Audio Filters - Filters and convolvers*
- *Audio Delay- Time-related operations*

Patch Management

- *Subwindows - Patch structuring*
- *Data Templates and Accessing Data - Objects related to data structures*

External libraries

- *GEM - OpenGL graphics and video library*
- *PDP - Video library to provide a way to use data packets as messages*
- *Physical Modelling - Physical modelling library*

Neste projeto foram utilizadas as seguintes subsecções: *glue, math, time, tables, misc, audio glue, audio math, audio oscilators and tables, subwindows* e *data templates and acessing data*.

4.4. Arduino Code

A linguagem *Arduino,* assim como *Pure Data,* é uma linguagem *open source,* encontra-se também em constante desenvolvimento.

Pode-se dizer que o código está dividido em três partes principais, *structure, variables* e *functions*.

A parte de estrutura é a parte que trabalha com os operadores ou seja, aritmética, comparação, apontadores, booleanos, *bitwise* (operações a nível de bits com números inteiros), complexos, sintaxe e controlo

A parte das variáveis trabalha com constantes, datas, conversões, variáveis globais e locais.

Por fim temos as funções, estas são normalmente criadas quando o programador necessita repetir este código várias vezes, assim só necessita de chamar a função cada vez que seja necessária.

Existem também já bibliotecas criadas no qual podem fazer comunicação com vários tipos de *hardware* ou a manipulação de dados.

As bibliotecas existentes são: *Standard Libraries, USB Libraries, Due Only Libraries, Esplora Only Library* e *Contributed Libraries.*

Standard Libraries

- *EEPROM - reading and writing to "permanent" storage*
- *Ethernet - for connecting to the internet using the Arduino Ethernet Shield*
- *Firmata - for communicating with applications on the computer using a standard serial protocol.*
- *LiquidCrystal - for controlling liquid crystal displays (LCDs)*
- *SD - for reading and writing SD cards*
- *Servo - for controlling servo motors*
- *SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus*
- *SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.*
- *Stepper - for controlling stepper motors*
- *WiFi - for connecting to the internet using the Arduino WiFi shield*
- *Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.*

USB Libraries (Leonardo, Micro, Due, and Esplora)

- *Keyboard - Send keystrokes to an attached computer.*
- *Mouse - Control cursor movement on a connected computer.*

Due Only Libraries

- *Audio - Play audio files from a SD card.*
- *Scheduler - Manage multiple non-blocking tasks.*
- *USBHost - Communicate with USB peripherals like mice and keyboards.*

Esplora Only Library

- *Esplora - this library enable you to easily access to various sensors and actuators mounted on the Esplora board.*

Neste projeto foi utilizado a biblioteca *Firmata*, esta biblioteca permite a comunicação do *hardware Arduino* com o programa *Pure Data* através da ligação USB.

O *hardware Arduino* disponibiliza as suas portas analógicas e digitais, que por sua vez é a base do controlador *midi*.

4.5. Controlador *midi*

Este subcapítulo apresenta o desenvolvimento do controlador *midi*, tanto a construção da caixa como a

parte de implementação do material eletrónico.

Para a construção do controlador foi utilizado material reciclado dando uma nova vida ao material não utilizado.

A caixa do controlador foi construída com placas de madeira na parte inferior e nas laterais, a parte de cima é uma placa de alumínio. Assim foi possível obter uma construção segura e resistente.

O material eletrónico que foi implementado no controlador foi:

- 33 Botões de pressão;
- 10 Potenciómetros analógicos de 22k;
- 3 Potenciómetros analógicos (*deslizante*) de 22k;
- *Arduino Mega 2560 r3;*
- 33 Resistências de 10k;
- *Breadboard;*
- Fios unifilares;
- Cabo *USB* macho/fêmea;
- Pen usb.

Na Figura 4.5, Figura 4.6 e Figura 4.7 está apresentado o resultado final do controlador *midi* (ilustração).

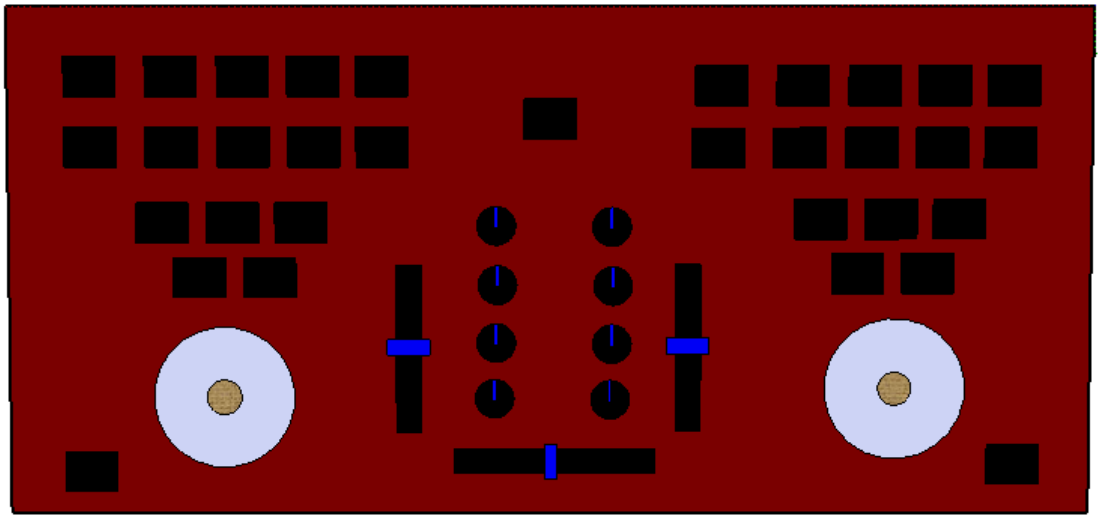


Figura 4.5 - Ilustração da parte superior do controlador *midi*

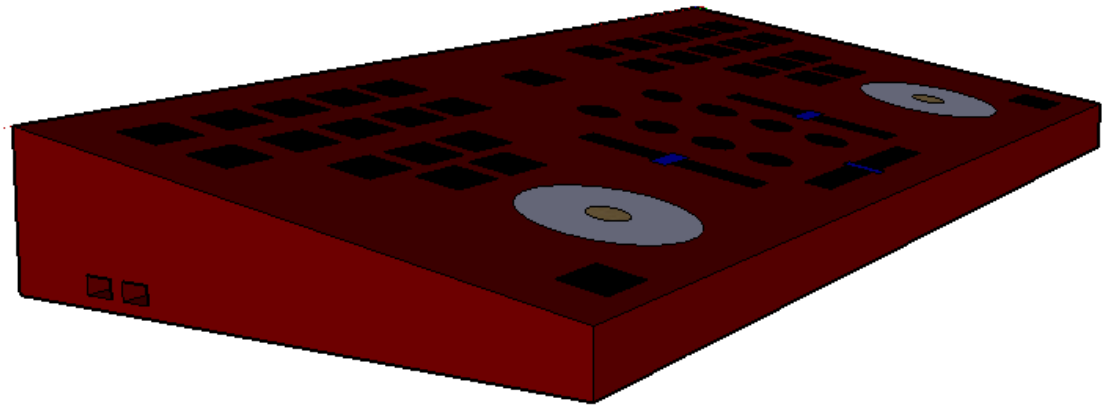


Figura 4.6 - Ilustração com vista da lateral esquerda do controlador

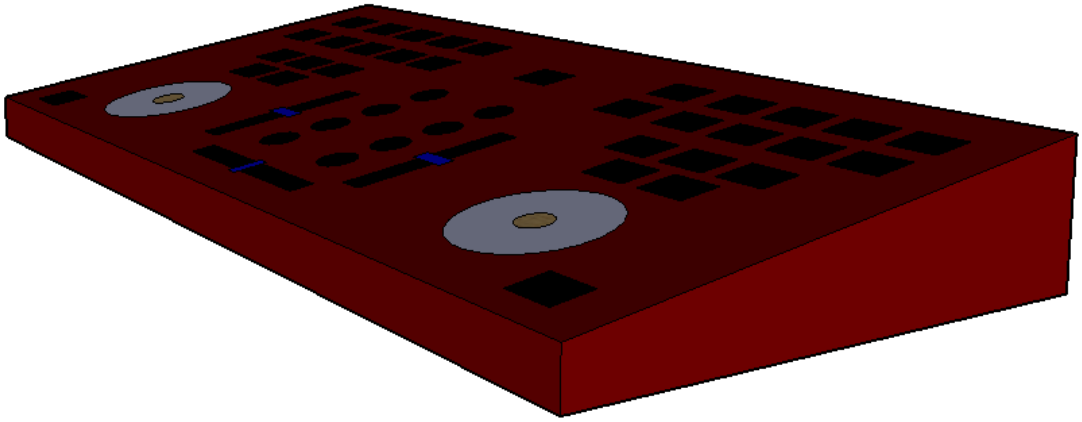


Figura 4.7 - Ilustração com vista da lateral direita do controlador

Na Figura 4.8, Figura 4.9 e Figura 4.10, é apresentado uma ilustração da implementação do circuito do controlador *midi*.

Cada botão de pressão está ligado em série com uma resistência de 10k para fazer o efeito *debounce*, ou seja, ao pressionar um dos botões de pressão não permitir que ocorra múltiplos sinais.

Os botões de pressão estão ligados às portas digitais do *Arduino* de forma a executar o som pré definido no programa *Pure Data*.

Os potenciómetros utilizados estão ligados às portas analógicas do *Arduino*. Estes efetuam alterações do volume e velocidade dos sons.

Legenda da ilustração:

- Botão de pressão;
-  Potenciómetro 22k;
-  Potenciómetro (*deslizante*) 22k;
- Cabo que faz a ligação digital dos botões de pressão ao *Arduino*;
- Cabo que faz a ligação analógica dos potenciómetros ao *Arduino*;
- Cabo de ligação aos 5V;
- Cabo de ligação à massa;
-  *Arduino Mega 2560 r3;*
-  *Breadboard.*

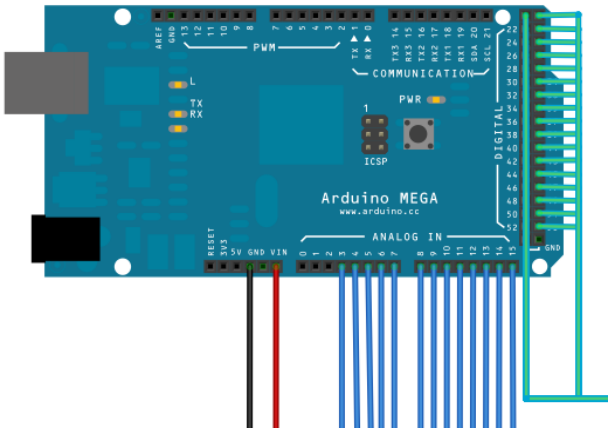


Figura 4.8 – Ilustração das ligações implementadas ao *Arduino*.

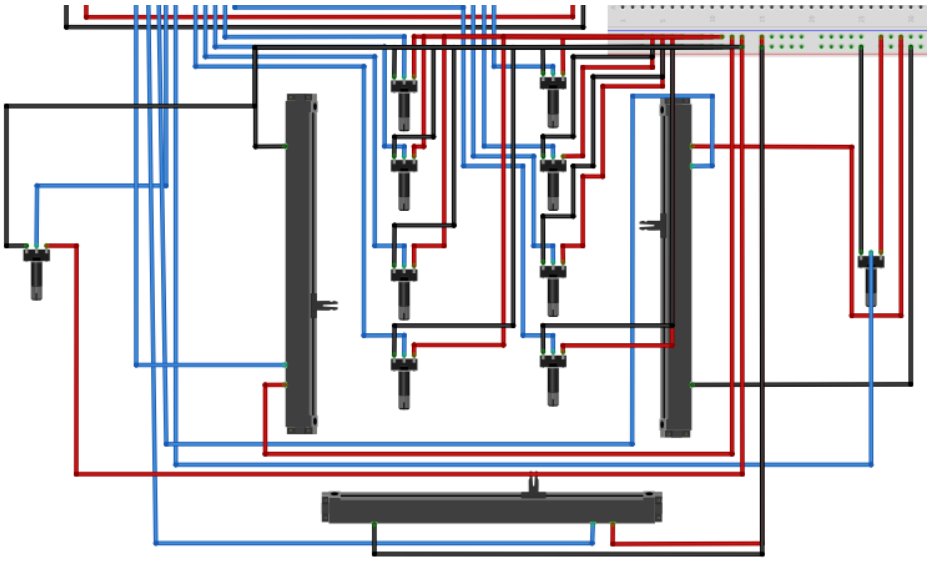


Figura 4.9 – Ilustração do circuito implementado (parte analógica).

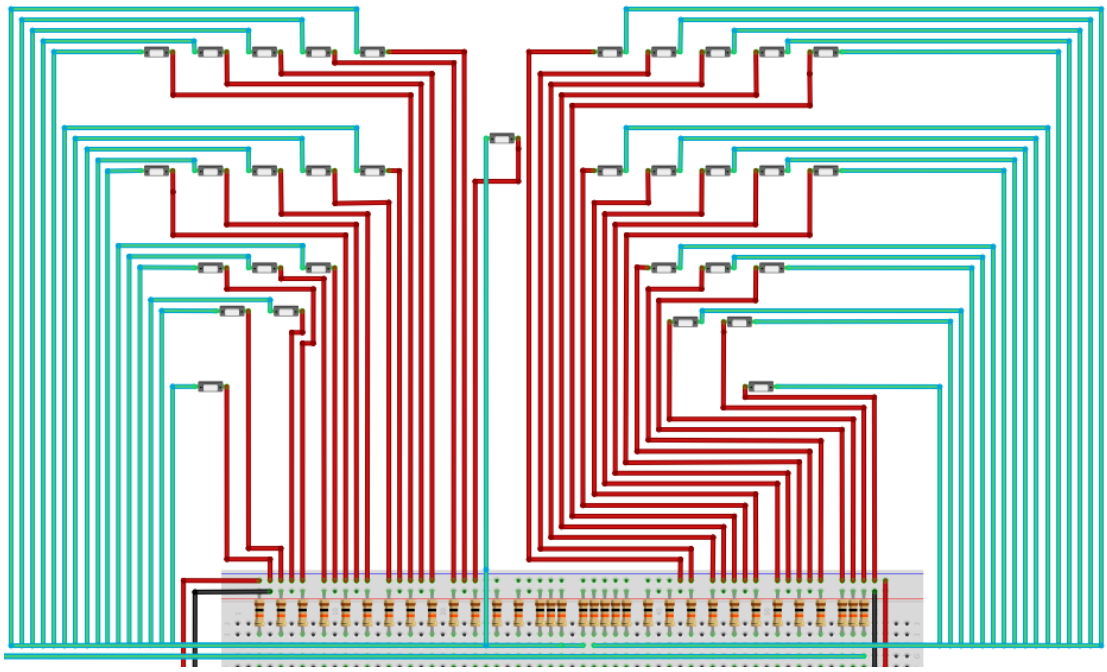


Figura 4.10 – Ilustração do circuito implementado (parte digital).

4.6. Implementação do código *Pure Data*

Neste subcapítulo é apresentado o código implementado, com a explicação de cada módulo e também de todos objetos que permitem o total funcionamento do controlador *midi*.

Como referido no subcapítulo 4.4, no *Arduino* foi utilizada a biblioteca *Firmata*. Esta biblioteca permite a comunicação do *hardware Arduino* com o programa *Pure Data* através da ligação *USB*.

Após a ligação *USB (serial port)*, todo o processo será feito através da comunicação entre os dois programas, *Pure Data* e *Arduino*. O *Arduino* irá enviar a informação de ativação ou desativação dos botões de pressão e da alteração dos potenciómetros para o *Pure Data* que por sua vez irá reproduzir o efeito do botão ou potenciómetro.

Na Figura 4.11, mostra o código usado para fazer a comunicação entre o programa *Pure Data* com o *Arduino*, assim como todas as portas analógicas e digitais do *Arduino*.

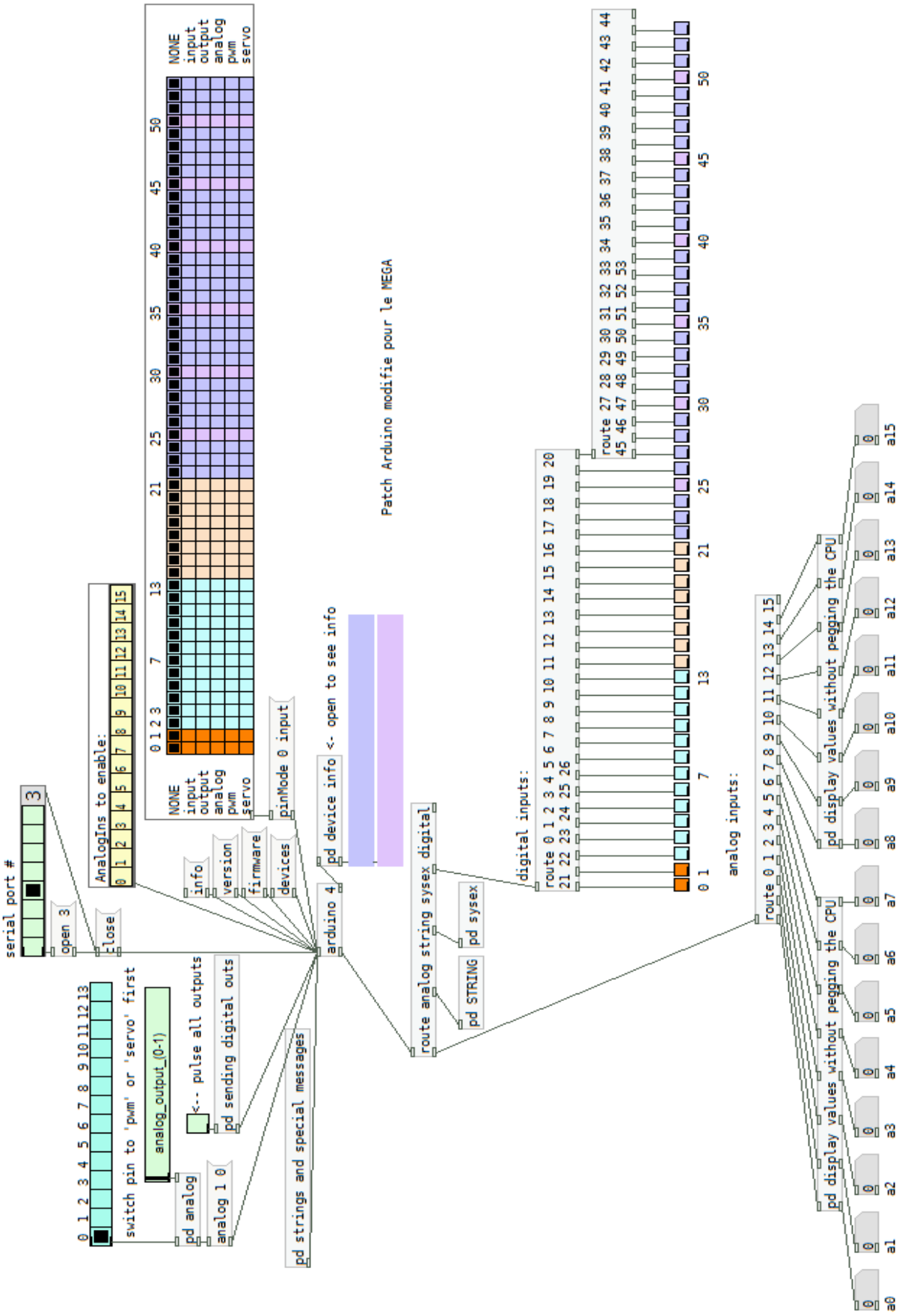


Figura 4.11 – Módulo que simula e faz a comunicação com o *Arduino*

Módulos de ligação analógica

Na Figura 4.12, Figura 4.13 e Figura 4.14, apresenta-se o código da modelação de som.

Neste módulo é onde se faz o controlo do volume da música e também a velocidade da sua batida.

Os potenciómetros que estão ligados nas portas analógicas de a3 a a6 são os controladores de volume do *deck* 1, que representam o volume, *master*, *high*, *mid* e *low*.

Os potenciómetros ligados nas portas a7 a a10 são os controladores de volume do deck 2, volume, *master*, *high*, *mid* e *low*.

Os potenciómetros (*deslizante*) ligados a a11 e a12 são os controladores das BPM dos respetivos *decks* 1 e 2.

A ligação a13 é o que faz o controlo do *crossfader*, ou seja, controla o volume dos dois *decks*.

Para a utilização correta destes controladores de volume é necessário fazer a conversão da leitura de 0 a 1 para 0 a 127 que é a escala usada pela tecnologia *midi*, daí ter sido necessário usar o objeto *autoscale*.

As ligações a14 e a15 são os controladores de velocidade da música mais acentuada, mais conhecidos por *scratch*. Estes serão apresentados na Figura 4.13 e Figura 4.14.

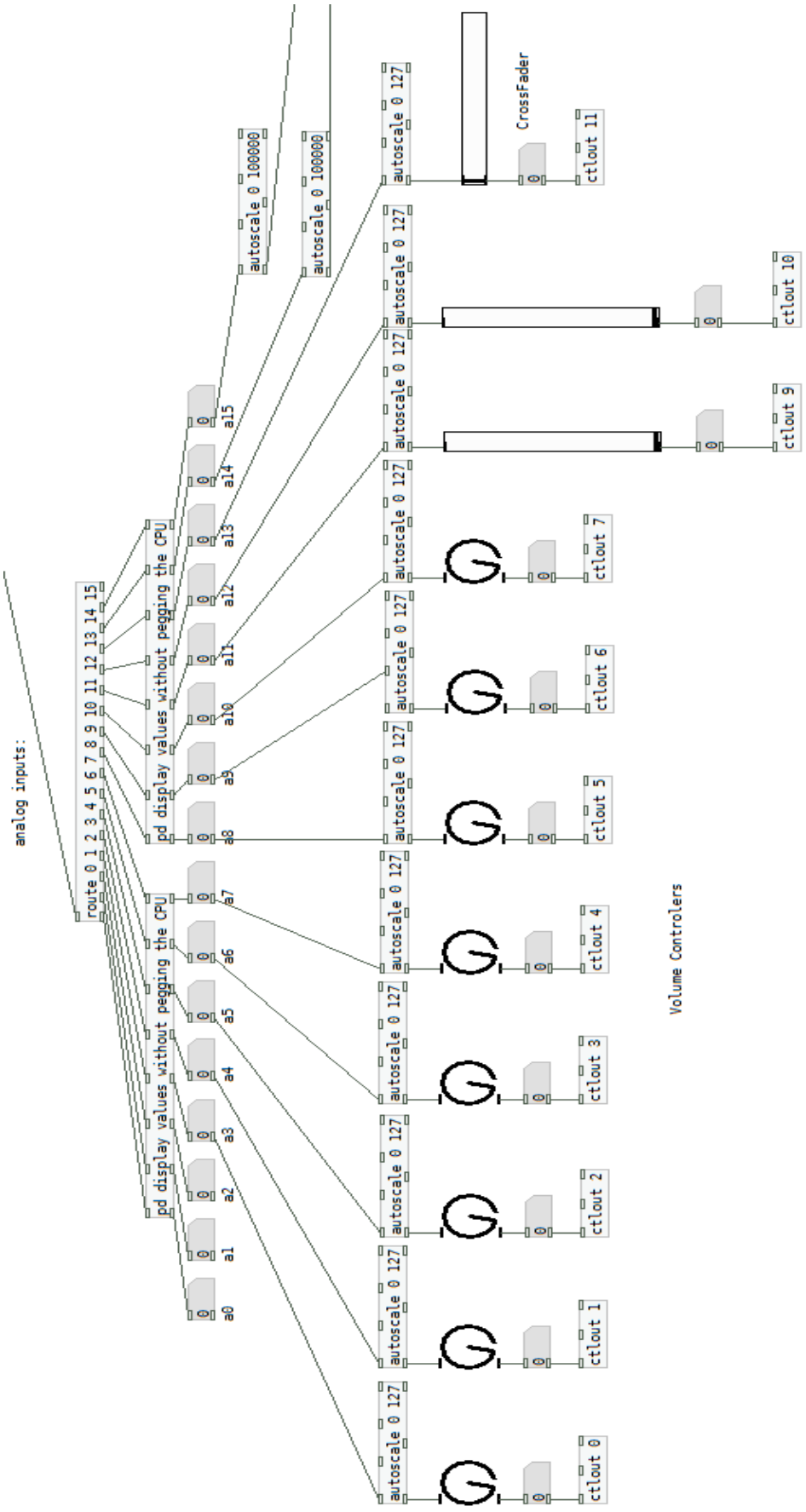


Figura 4.12 – Controladores de volume e batimentos por minuto.

A Figura 4.13 apresenta as funções *cue* e *scratch*.

O *cue* serve para marcar uma posição na música na qual mais tarde poderá voltar a essa posição ao pressionar novamente o botão.

No entanto estes botões de pressão, através do objeto ctlout, é possível serem programados consoante se deseje no programa de *dj*.

Os objetos *pd* são objetos que contêm código dentro deles, como o caso do *pd scratch* 1 e 2.

Os objetos s *scratch* 1 e 2, fazem o envio do sinal para um objeto *receive* com o mesmo nome, neste caso será para o *r scratch* 1 e 2 que se encontram dentro do *pd scratch* 1 e 2.

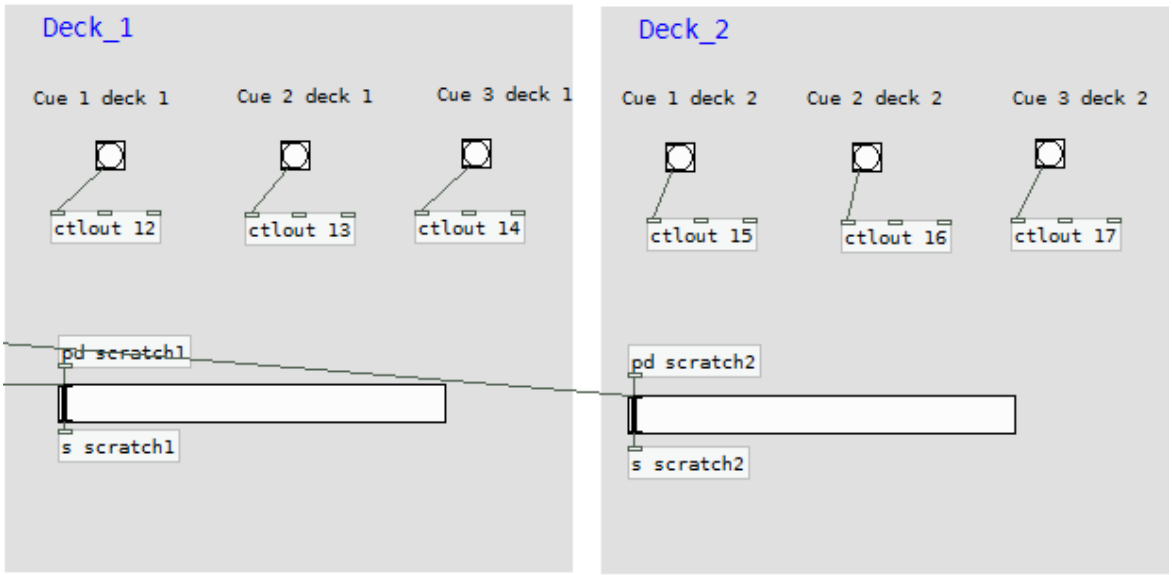


Figura 4.13 – Funções *cue* e *scratch*

Na Figura 4.14, apresenta o código dentro dos objetos *pd scratch* 1(a) e 2 (b).

Estes objetos fazem com que seja possível fazer um varrimento rápido no espetro da música. Com este método é possível adicionar efeitos ao som que está a ser reproduzido.

O objeto *loadbang* envia uma mensagem de ativação, assim permitindo a abertura do som *break.wav*, de seguida vai ser feita a leitura do tamanho do espetro da música. Após essa leitura faz uma redução do seu tamanho.

O objeto *soundfiler* faz uma nova leitura e emite o tamanho que foi escrito pela mensagem *read* no *array Tab*. A mensagem *;ps range 0 $1* faz o envio do tamanho do som para o objeto *s ps*.

O objeto *r scratch* ao receber o sinal do *s ps* e do *s scratch* vai enviar as duas mensagens para o *pack 0 100*. Este por sua vez une as duas mensagens.

De seguida através do *vline~* permite fazer a leitura dos tempos e das amplitudes com grande precisão.

Assim o objeto *tabreader4~ tab* com a informação recebida recria um *sampler*.

Por fim o objeto *dac~* faz com que o sinal sonoro possa ser ouvido.

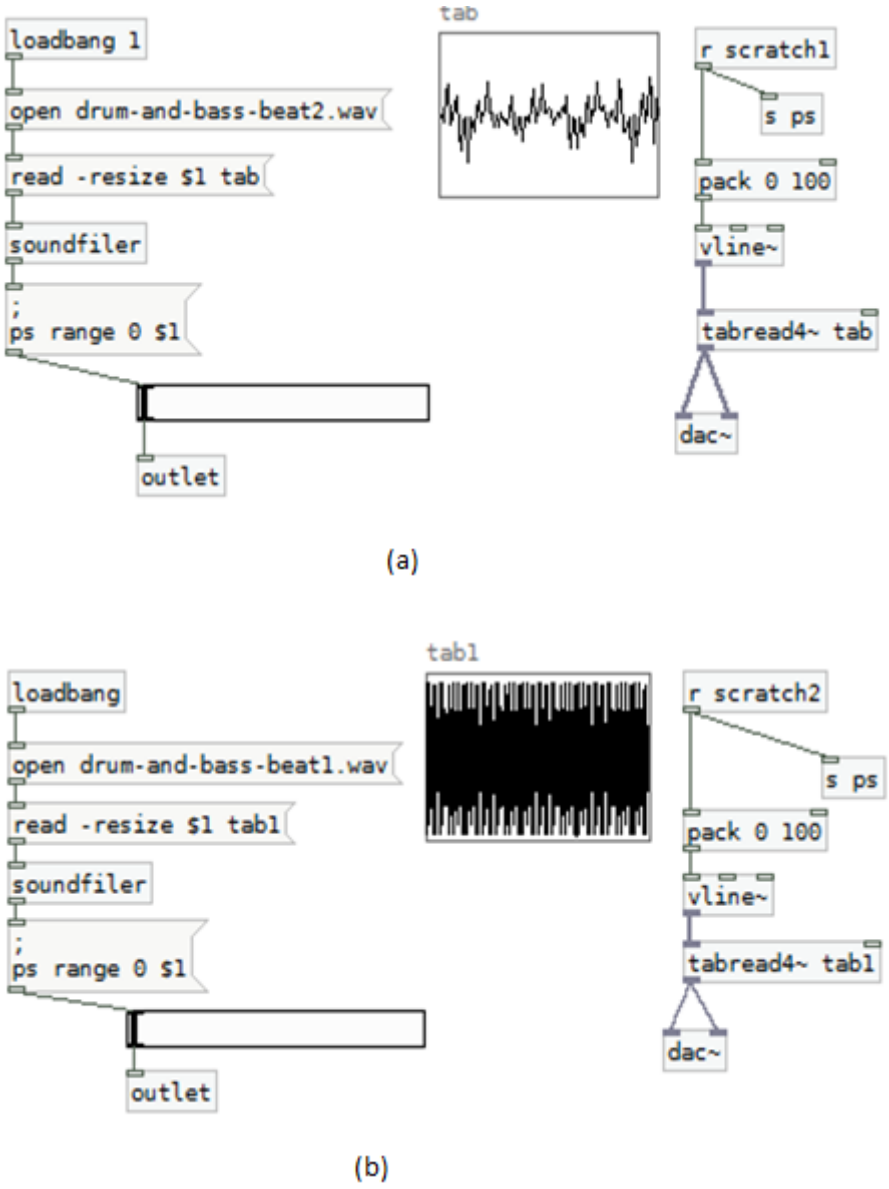


Figura 4.14 - Código *pd scratch* 1 (a) e *scratch* 2 (b).

Módulos de ligações digitais

Na Figura 4.15 e Figura 4.16, estão apresentados os *samplers* de curta duração, o botão de ativação da repetição e os *samplers* de longa duração.

A diferença entre os *samplers* de curta duração para os de longa duração, é que os de curta duração, os botões de pressão fazem sempre o início do *sample* cada vez que é pressionado. Os de longa duração, quando são pressionados ativam o *sample*, quando pressionado durante o som será feita a função de paragem do *sample*.

Os objetos *pd sample* contêm o código que faz o disparo do *sample* e também o código de repetição de som. Esse código é apresentado na Figura 4.17.

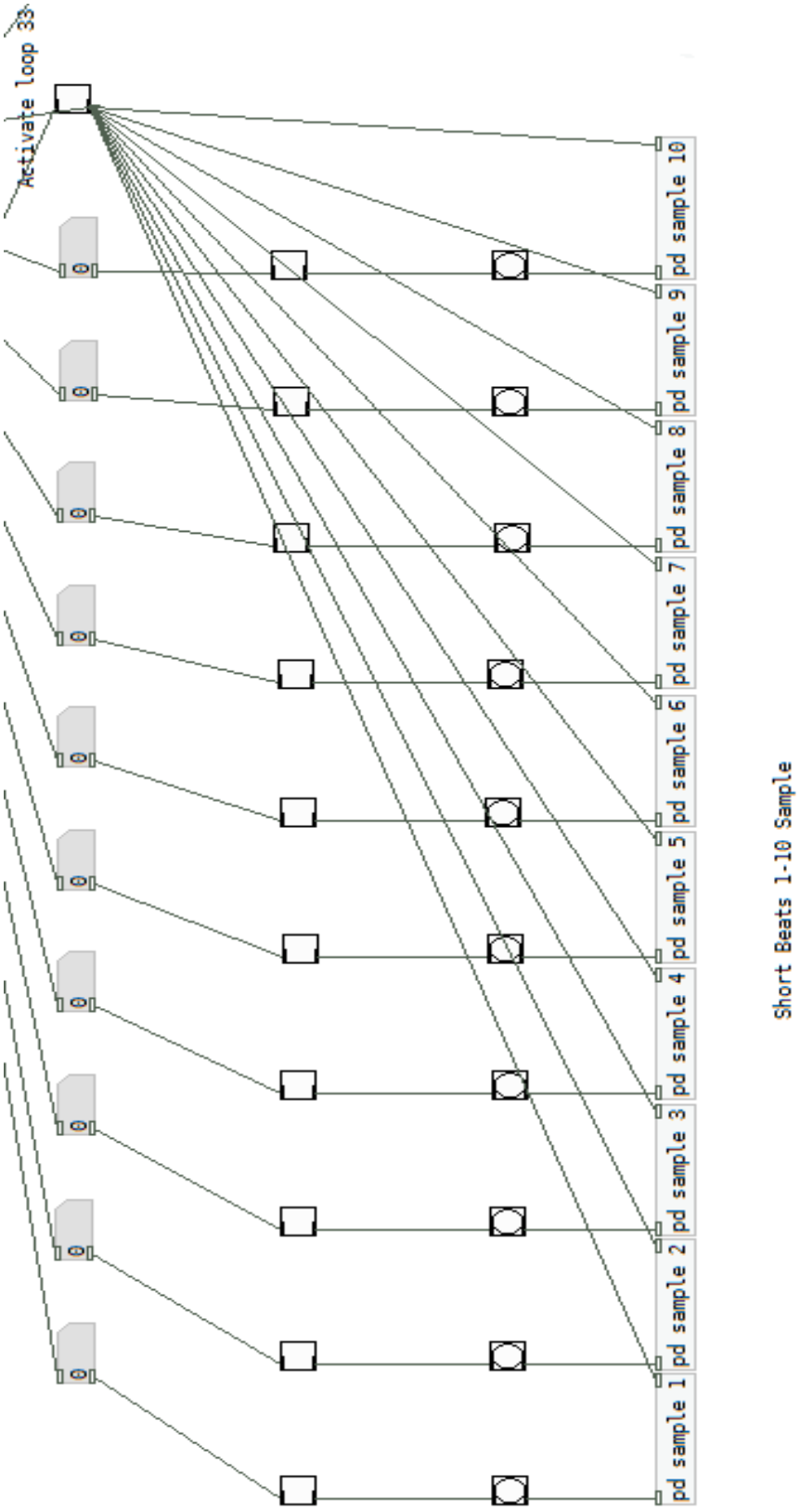


Figura 4.15 – *Samples* de curta duração e ativação de repetição

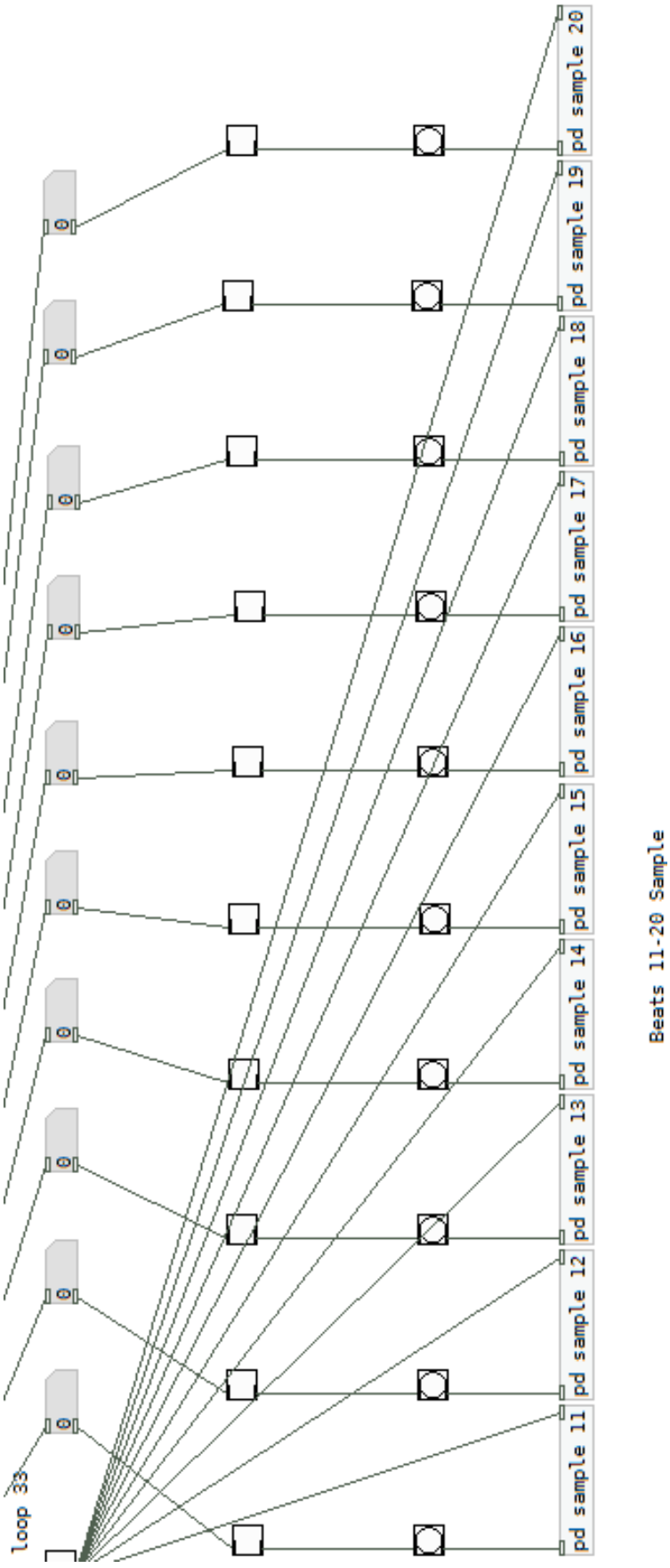


Figura 4.16 – *Samples* de longa duração

Como referido no parágrafo acima, a Figura 4.17, apresenta o código dos objetos *pd sample*.

Este código faz a reprodução de um som *wave* cada vez que o botão de pressão seja acionado.

O objeto *inlet* serve para criar um controlo de entrada para um objeto *pd*.

Quando o botão de pressão *sample* é ativado a informação entrará no primeiro *bang*. O *bang* dá o início da sequência do código. De seguida através da mensagem *open 123bass.wav* o ficheiro *wave* vai ser aberto para ser lido através do objeto readsf~.

Após a leitura do som este será transmitido para o *dac~* que permitirá o ficheiro *wave* ser ouvido.

O último *bang* dá indicação de quando o som foi terminado. Este serve também para dar continuação à repetição da sequência.

A ativação do botão de repetição faz com que o *toggle* fique acionado. Essa informação passa para o objeto *spigot*, que funciona como um *if*, ou seja, quando o *toggle* estiver ativo e o ultimo botão *bang* for acionado, entrará no *spigot*, o que faz com que a sequência seja repetida sem ser necessário premir novamente o botão de *sampler.*

A diferença entre os *samplers* de longa duração (b) para os de curta duração (a) são os objetos *toggle spigot* e a mensagem *stop* que se encontram a mais no código à direita. Esse código a mais faz com que ao pressionar duas vezes o botão de *sampler* suceda a paragem da música.

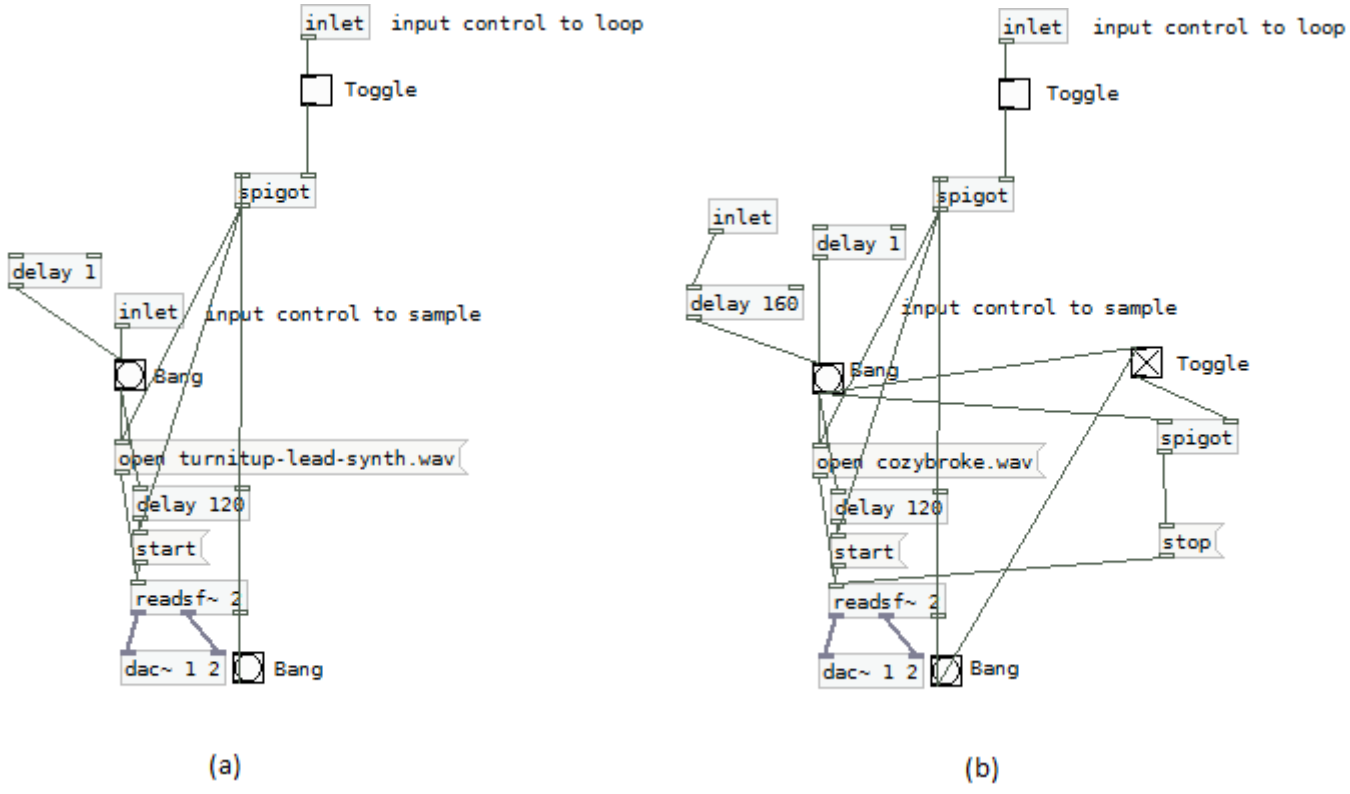


Figura 4.17 - Código dos *samplers* de curta (a) e longa duração (b)

Na Figura 4.18, estão representados os botões de pressão que fazem a ativação do controlador *midi*, a gravação de som, reprodução do som e a mistura automática de dois *samplers*.

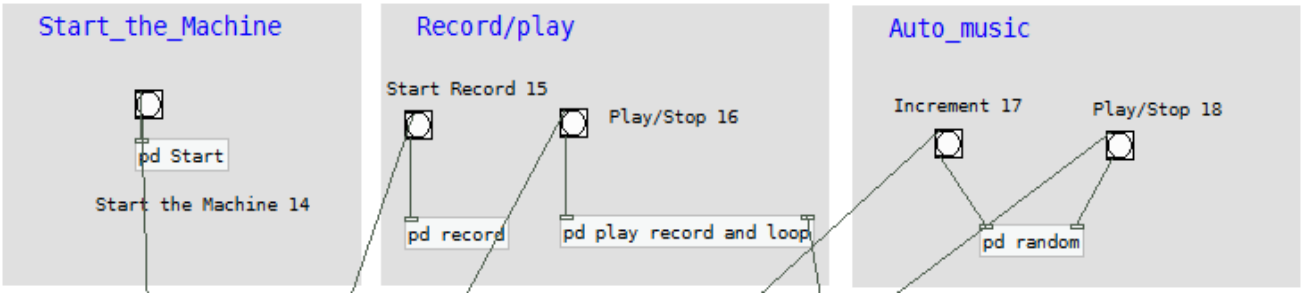


Figura 4.18 – Ativação do controlador, gravação/reprodução e mistura automática.

Os objetos pd Start, pd record e pd play record and loop têm o seu código representados na Figura 4.19.

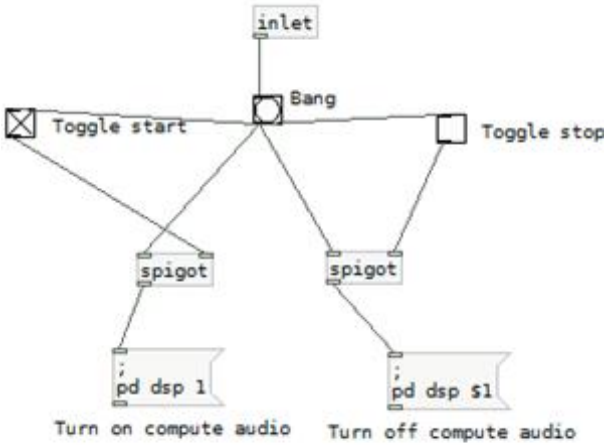
Os objetos do Auto_music estão representados na Figura 4.20.

O código (a) da Figura 4.19 representa a ativação do equipamento, sempre que o bang for a 1, o toggle que estiver ativo irá acionar o referido spigot dando origem à ativação da máquina e desativação da mesma, ;pd dsp 1 e ;pd dsp $1.

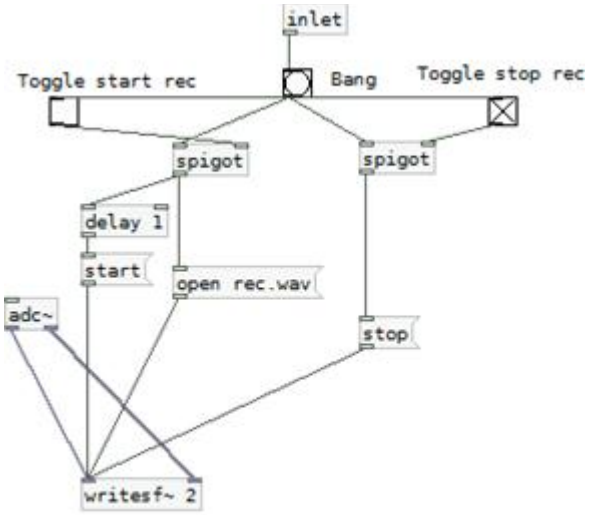
O código (b) da Figura 4.19 representa a gravação de um som através de um microfone.

Sempre que o bang (botão de pressão) for a 1, o toggle start rec é acionado e dá início à gravação. É criado um ficheiro com o nome rec.wav e através do adc~, que permite a entrada de sons externos e o writesf ~, faz a gravação do som no ficheiro wave. Tudo o que for dito ao microfone fica gravado no rec.wav. Este processo só termina quando o bang voltar a 1 e o toggle stop rec ficar acionado. Dando assim por finalizado a gravação.

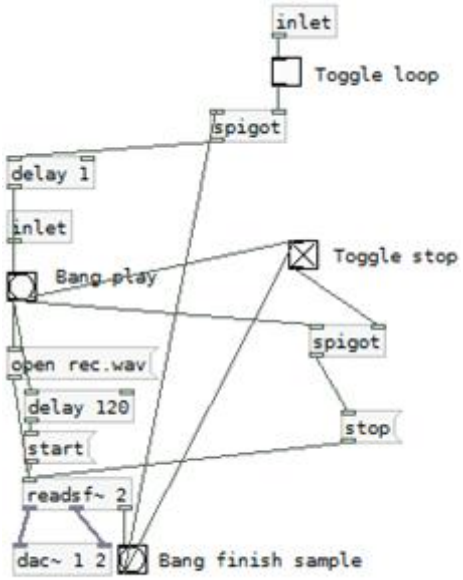
O código (c) da Figura 4.19 é a reprodução do ficheiro rec.wav. Este código é igual ao código apresentado na Figura 4.17 (b).



(a)



(b)



(c)

Figura 4.19 - Código *pd start (a), pd record (b)* e *pd play record and loop (c)*

Na Figura 4.20, tem-se o código que faz início da mistura automática de dois samplers e o incremento de divisões da secção musical.

Ao premir o botão de pressão *play* do *Auto_music* este vai fazer um incremento e lançar a mistura dos dois *samplers*. Quando o botão for pressionado a segunda vez este irá fazer a paragem da música.

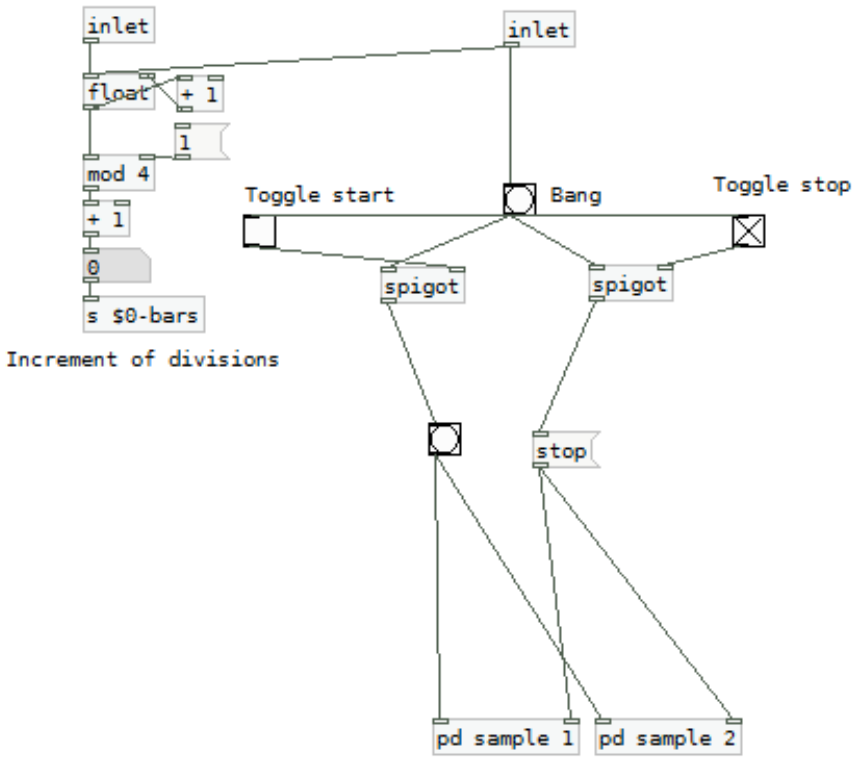


Figura 4.20 - Código de incremento de secções e início da mistura automática

Na Figura 4.21, está apresentado o código *pd sample 1* e *2* da mistura automática.

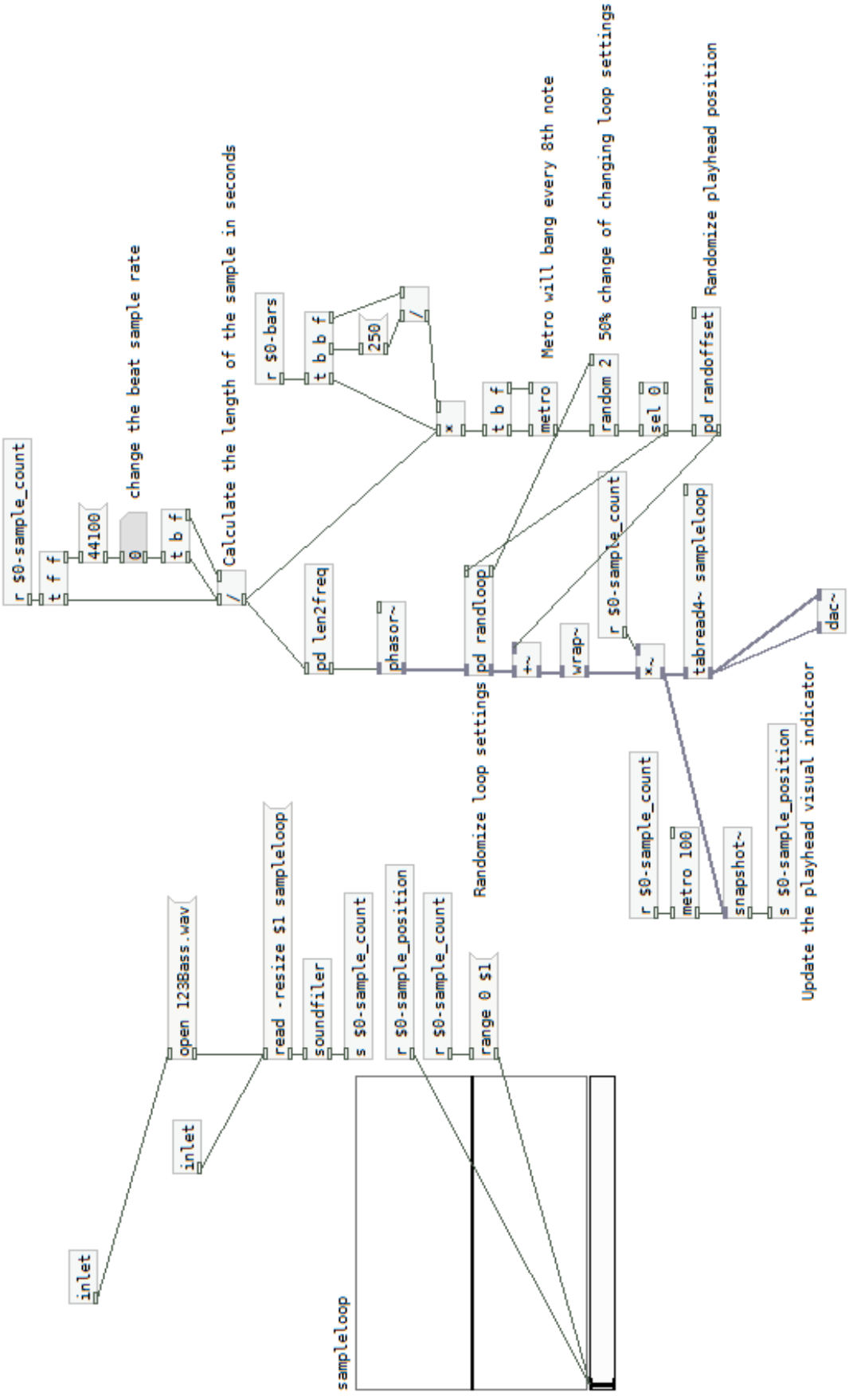


Figura 4.21 - Código que faz com que um *sampler* toque aleatoriamente

Este módulo pode ser representado em 5 partes, *sample loading*, *sample playback*, *loop position marker display*, *randomization of the playback position* e *randomization of the loop*.

Sample Loading

O ficheiro de som inicia, *open 123bass.wav*, quando o botão de pressão de *random loop* é pressionado. O objeto *soundfiler* lê a amostra no qual foi passada para um *array* através da mensagem *read*. O tamanho do ficheiro é emitido pelo objeto *soundfiler* e pode ser sempre utilizado através do objeto *r $0-sample_count*.

Sample Playbacj

O *sample* é reproduzido usando o objeto *phasor~* que fornece um índice para o objeto *tabread4~*. O *phasor~* gera valores que vão de 0,0 a 1,0, este vai ser multiplicado pelo número de amostras do ficheiro, e assim obtém-se um intervalo entre 0,0 e o número de amostras.

Para reproduzir a amostra no seu tempo original, definiu-se no *phasor~* para executar uma frequência, de modo que um único período leve o mesmo tempo que um ciclo do loop. Esta frequência é calculada através do objeto *pd len2freq*.

Loop Position Marker Display

O objeto *hslider* mostra o local onde o ficheiro está a ser reproduzido. O tamanho do *hslider* advém do tamanho enviado do *sample_count*. Ao usar o objeto *snapshot* é possível gerar um valor em resposta ao objeto *metro*, este último tem uma resposta de 10 *frames* por segundo. A posição do *hslider* reflete esse valor.

Randomization of Loop

Graças ao objeto *pd randloop* é possível adicionar rápido staccato para dar um efeito *drum and bass* ao som. Isto é feito de forma aleatória através da multiplicação do *phasor~* por 1, 2, 4, 8, 16, 32, 64 e 128. Passando pelo objeto *wrap~* e dividindo novamente pelo valor que foi anteriormente multiplicado.

Este processo diminui a eficácia do *phasor~*, assim este faz um *loop* de um tempo menor da amostra de som.

Quanto maior for o valor da multiplicação maior vai ser a possibilidade de aumentar a variação dos trechos da música, esta passa a ser entre 50 a 100%, nos outros valores anda entre 0 a 50%.

5. Conclusão

Os sintetizadores são hoje em dia um equipamento comum na música. A modernização dos sintetizadores levou a um sistema cada vez mais eficiente, com diversas funcionalidades incorporadas. O uso de *samplers*, gravação e repetição dos sons são as aplicações mais frequentes deste género de sistemas. No entanto, novas aplicações foram introduzidas neste sistema tais como a mistura de dois sons automaticamente e a possibilidade de arquivar ficheiros no disco rígido interno.

Os sintetizadores *midi* ou controladores *midi*, foram o alvo de estudo nesta dissertação. Em particular, foi implementada a técnica do Método Analógico, capítulo 3.5.2. Todos os objetivos propostos para este projeto foram alcançados, tendo sido possível implementar todas as funcionalidades pretendidas.

O sistema desenvolvido inclui a plataforma *Arduino* e o programa *Pure Data*. A escolha do *Arduino* baseou-se na sua capacidade de permitir um processamento e comunicação rápido para que a eficácia fosse preservada a custos rentáveis. Esta plataforma tem a grande vantagem por ter um grande leque de portas analógicas e digitais que permitiram implementar uma grande diversidade de efeitos sonoros. A escolha da plataforma *Arduino* apresenta os riscos inerentes à escolha de uma plataforma livre. A possibilidade de utilização de bibliotecas com erros de implementação é uma realidade, podendo levar a comportamentos indesejados por parte do sistema final. No entanto, nenhum comportamento não esperado foi observado ao longo do desenvolvimento deste projeto, o que revela a maturidade que esta plataforma apresenta à data. Juntando este facto ao seu baixo custo de aquisição, trata-se definitivamente de uma plataforma a considerar para qualquer projeto que envolva a leitura/escrita de sinais analógicos e digitais.

Relativamente ao programa *Pure Data*, a sua escolha foi devida à sua grande capacidade de criar em tempo real diversos efeitos sonoros. Este também foi escolhido devido à comunicação que é possível ser feita com o *Arduino* e a ligação com o protocolo *midi*. A sua interface gráfica revelou-se adequada para a implementação do controlador *midi*. Os riscos da utilização do *Pure Data* são semelhantes aos da utilização do *Arduino*, dado que o seu desenvolvimento segue a filosofia *open source*. No entanto, à semelhança do *Arduino*, não foi experienciada qualquer instabilidade na sua utilização, sendo claramente uma referência para o desenvolvimento de controladores *midi*. Há que realçar que o *Pure Data* apresenta diversas limitações no desenvolvimento de outras funcionalidades para além das relacionadas com controladores *midi*, pelo que não deverá ser considerado como plataforma central de desenvolvimento para todos os projetos *Arduino*.

Tendo em conta as inúmeras possibilidades de expansão providenciadas pelo *Arduino/Pure Data*, este projeto apresenta considerável potencial de desenvolvimento em futuros trabalhos. Eis alguns dos possíveis desenvolvimentos a considerar:

- Utilização do atual controlador *midi* (sintetizador mais *midi fighter*) como base para a implementação de outro tipo de controladores *midi* como por exemplo uma bateria eletrónica.
- Introdução de uma funcionalidade de leitura dos espectros das músicas e dos *samplers* e aplicar os *samplers* nas músicas consoante os espectros compatíveis;

- Desenvolver uma ligação entre controladores *midi*, podendo assim fazer uma estação de trabalho e sequenciador de *hardware*;

Estes trabalhos futuros em conjunto com esta dissertação fornecerão informação importante acerca dos controladores *midi*, assim como da capacidade de plataformas livres implementarem sistemas tão complexos como sistemas comerciais com custos bastante mais elevados.

Bibliografia

[1] Edward MacDowell, "Critical & Historical Essays Lectures delivered" Columbia University, 2005.

[2] Montanaro, Larisa Katherine "A singer's Guide to Performing Works for Voice and Electronics", University of Texas, 2004.

[3] Curtis Roads, Stephen Travis Pope, Aldo Piccialli e Giovanni de Poli, "Musical Signal Processing", Swets & Zeitlinger B.V., Lisse, Holanda, 1997.

[4] Luís L. Henrique. "Acústica Musical", 2ª edição, Fundação Calouste Gulbenkian, Lisboa, 2007.

[5] Mccautley, Jack J. Bright, Brian. Devecka, John. Vandenberghe, Jason. "System and Method for Playing a Music Video Game With a Drum System Game Ccontroller" Activision Publish Inc, USA, 2013. Available: http://www.freepatentsonline.com/y2013/0012279.html,

[6] Bejamin W. Karpf, B.A. "Sound and Code: The effect of digital technology on music making", Georgetown University, Washington, D.C., 2003.

[7] Roman Kogan, "Brief History of Electronic and Computer Musical Instruments", 2008.

[8] MMA, "Midi 1.0 Detailed Specefication", 1995

[9] Prof. Jeffrey Hass. "Introduction to Computer Music: Volume One". Center for Electronic and Computer Music, D.M., Indiana, 2010
Available: http://www.indiana.edu/~emusic/etext/acoustics/chapter1_intro.shtml,

[10] C.R.Nave "HyperPhisics". Paper presented to AAPT, Guelph, Canada, 2000. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/hframe.html.

[11] Kruvczuk, Michelle. Pusateri, Ernest. Covell, Alison. "Music Transcription for Lazy Musician", 5000 Forbes Avenue Pittsburgh, PA 15213-3890 USA, 2000.

[12] Heckroth Jim. "A Tutorial on MIDI and Wavetable Music Synthesis", MIDI Manufacturers Association Incorporated, 1993.

Referências Web

[web1] http://nuansamusik.com/p/8997/korg-microkorg-mk1-synthesizer-vocoder-keyboards

[web2] http://www.studiomastering.net/e/mastering08e.html, 2008.

[web3] http://www.akaipro.com/mpd18

[web4] http://www.axetopia.com/news/06_10/digital-analog.html

[web5] http://www.arduino.cc.

[web6] http://wiki.openmoko.org/wiki/Main_Page.

[web7] http://opensource.org.

[web8] http://www.raspberrypi.org

[web9] http://www.mozilla.org

[web10] http://www.linux.org/

[web11] http://www.openoffice.org/

[web12] http://www.android.com/

[web13] http://puredata.info/

[web14] http://www.jhu.edu/virtlab/ray/acoustic.htm.

[web15] http://library.thinkquest.org/19537/

A. ANEXO

Esquemáticos

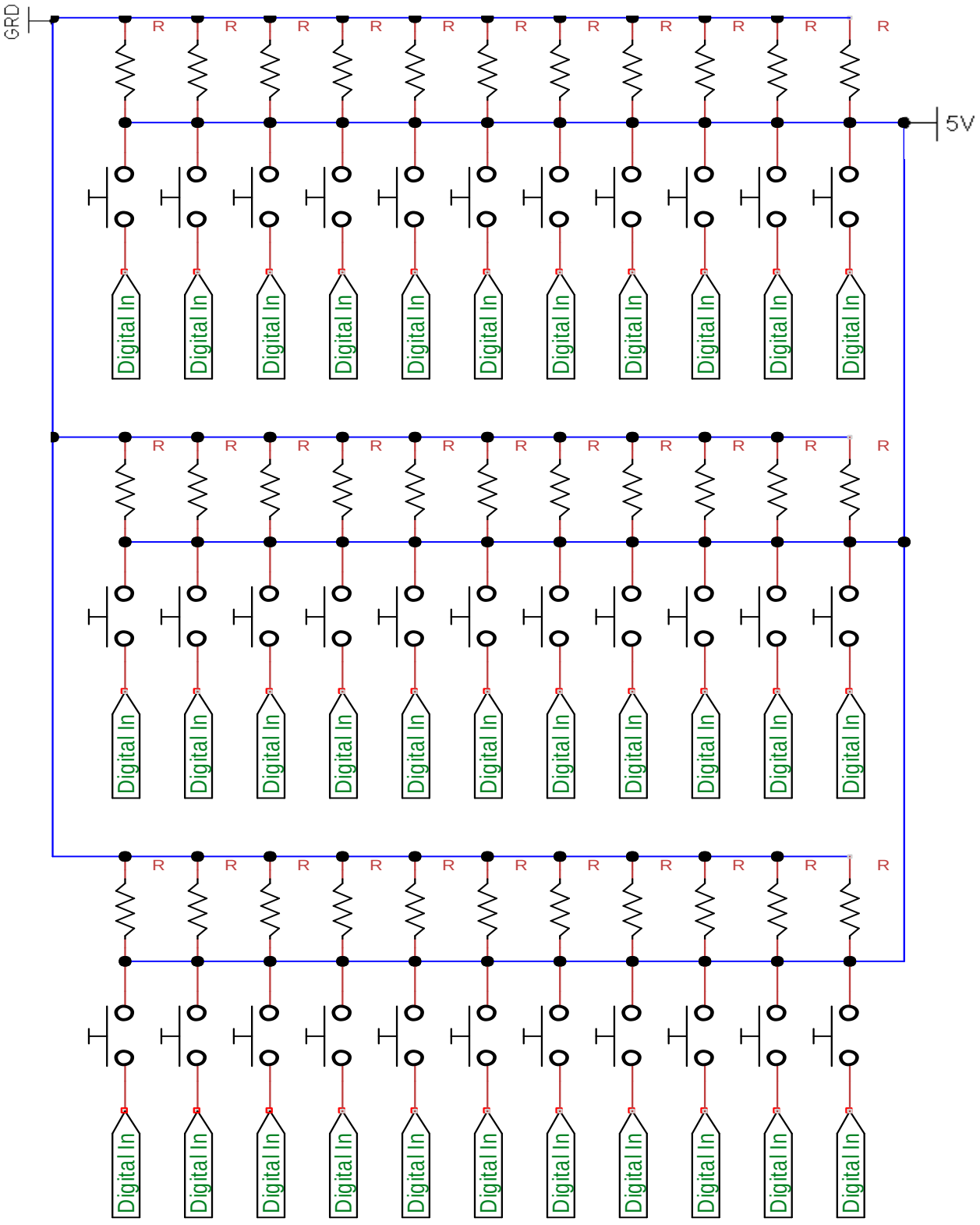


Figura A.1 – Esquemático dos botões de pressão

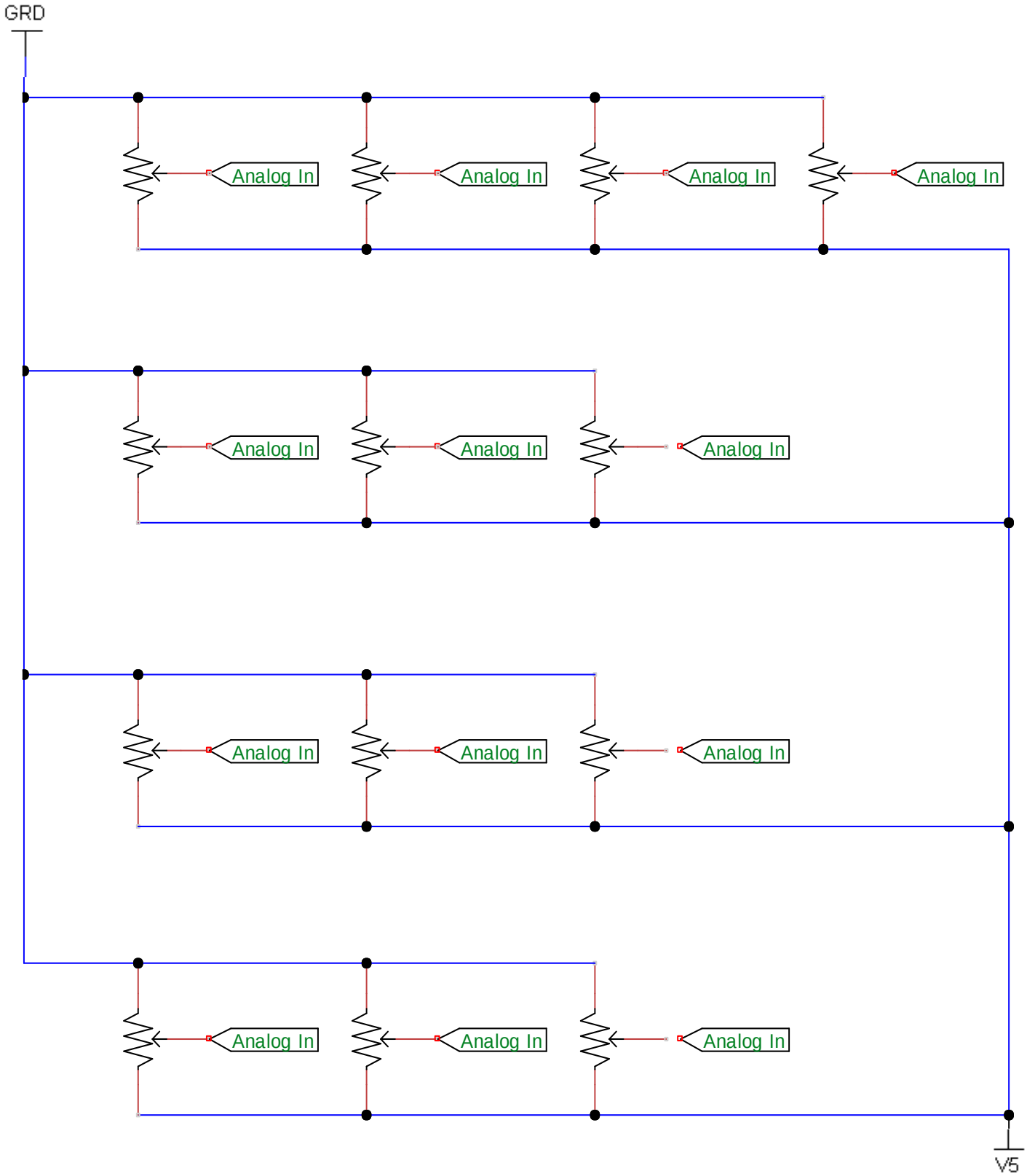


Figura A.2 – Esquemático dos potenciómetros

Arduino Code

```
//*************************************************************************
//* Includes
//*************************************************************************
#include "Firmata.h"
#include "HardwareSerial.h"

extern "C" {
#include <string.h>
#include <stdlib.h>
}
//*************************************************************************
//* Support Functions
//*************************************************************************
void FirmataClass::sendValueAsTwo7bitBytes(int value)
{
  FirmataSerial->write(value & B01111111); // LSB
  FirmataSerial->write(value >> 7 & B01111111); // MSB
}
void FirmataClass::startSysex(void)
{
  FirmataSerial->write(START_SYSEX);
}
void FirmataClass::endSysex(void)
{
  FirmataSerial->write(END_SYSEX);
}
//*************************************************************************
//* Constructors
//*************************************************************************
FirmataClass::FirmataClass()
{
  firmwareVersionCount = 0;
  firmwareVersionVector = 0;
  systemReset();
}
//*************************************************************************
//* Public Methods
//*************************************************************************
/* begin method with default serial bitrate */
void FirmataClass::begin(void)
{
```

```
  begin(57600);
}
/* begin method for overriding default serial bitrate */
void FirmataClass::begin(long speed)
{
  Serial.begin(speed);
  begin(Serial);
  blinkVersion();
}
/* begin method for overriding default stream */
void FirmataClass::begin(Stream &s)
{
  FirmataSerial = &s;
  printVersion();
  printFirmwareVersion();
}
// output the protocol version message to the serial port
void FirmataClass::printVersion(void) {
  FirmataSerial->write(REPORT_VERSION);
  FirmataSerial->write(FIRMATA_MAJOR_VERSION);
  FirmataSerial->write(FIRMATA_MINOR_VERSION);
}
void FirmataClass::blinkVersion(void)
{
  // flash the pin with the protocol version
  pinMode(VERSION_BLINK_PIN,OUTPUT);
  strobeBlinkPin(FIRMATA_MAJOR_VERSION, 40, 210);
  delay(250);
  strobeBlinkPin(FIRMATA_MINOR_VERSION, 40, 210);
  delay(125);
}
void FirmataClass::printFirmwareVersion(void)
{
  byte i;
  if(firmwareVersionCount) { // make sure that the name has been set before reporting
    startSysex();
    FirmataSerial->write(REPORT_FIRMWARE);
    FirmataSerial->write(firmwareVersionVector[0]); // major version number
    FirmataSerial->write(firmwareVersionVector[1]); // minor version number
    for(i=2; i<firmwareVersionCount; ++i) {
      sendValueAsTwo7bitBytes(firmwareVersionVector[i]);
    }
    endSysex();
```

```
  }
}
void FirmataClass::setFirmwareNameAndVersion(const char *name, byte major, byte minor)
{
  const char *filename;
  char *extension;
  // parse out ".cpp" and "applet/" that comes from using __FILE__
  extension = strstr(name, ".cpp");
  filename = strrchr(name, '/') + 1; //points to slash, +1 gets to start of filename
  // add two bytes for version numbers
  if(extension && filename) {
    firmwareVersionCount = extension - filename + 2;
  } else {
    firmwareVersionCount = strlen(name) + 2;
    filename = name;
  }
  free(firmwareVersionVector);
  firmwareVersionVector = (byte *) malloc(firmwareVersionCount);
  firmwareVersionVector[firmwareVersionCount] = 0;
  firmwareVersionVector[0] = major;
  firmwareVersionVector[1] = minor;
  strncpy((char*)firmwareVersionVector + 2, filename, firmwareVersionCount - 2);
  // alas, no snprintf on Arduino
  //    snprintf(firmwareVersionVector, MAX_DATA_BYTES, "%c%c%s",
  //            (char)major, (char)minor, firmwareVersionVector);
}
// this method is only used for unit testing
// void FirmataClass::unsetFirmwareVersion()
// {
//   firmwareVersionCount = 0;
//   free(firmwareVersionVector);
//   firmwareVersionVector = 0;
// }
 //------------------------------------------------------------------------------
// Serial Receive Handling
int FirmataClass::available(void)
{
  return FirmataSerial->available();
}
void FirmataClass::processSysexMessage(void)
{
  switch(storedInputData[0]) { //first byte in buffer is command
  case REPORT_FIRMWARE:
```

```
    printFirmwareVersion();
    break;
  case STRING_DATA:
    if(currentStringCallback) {
      byte bufferLength = (sysexBytesRead - 1) / 2;
      char *buffer = (char*)malloc(bufferLength * sizeof(char));
      byte i = 1;
      byte j = 0;
      while(j < bufferLength) {
        buffer[j] = (char)storedInputData[i];
        i++;
        buffer[j] += (char)(storedInputData[i] << 7);
        i++;
        j++;
      }
      (*currentStringCallback)(buffer);
    }
    break;
  default:
    if(currentSysexCallback)
      (*currentSysexCallback)(storedInputData[0], sysexBytesRead - 1, storedInputData +
1);
  }
}
void FirmataClass::processInput(void)
{
  int inputData = FirmataSerial->read(); // this is 'int' to handle -1 when no data
  int command;
      // TODO make sure it handles -1 properly
    if (parsingSysex) {
    if(inputData == END_SYSEX) {
      //stop sysex byte
      parsingSysex = false;
      //fire off handler function
      processSysexMessage();
    } else {
      //normal data byte - add to buffer
      storedInputData[sysexBytesRead] = inputData;
      sysexBytesRead++;
    }
  } else if( (waitForData > 0) && (inputData < 128) ) {
    waitForData--;
    storedInputData[waitForData] = inputData;
```

```
  if( (waitForData==0) && executeMultiByteCommand ) { // got the whole message
    switch(executeMultiByteCommand) {
    case ANALOG_MESSAGE:
      if(currentAnalogCallback) {
        (*currentAnalogCallback)(multiByteChannel,
                                 (storedInputData[0] << 7)
                                 + storedInputData[1]);
      }
      break;
    case DIGITAL_MESSAGE:
      if(currentDigitalCallback) {
        (*currentDigitalCallback)(multiByteChannel,
                                  (storedInputData[0] << 7)
                                  + storedInputData[1]);
      }
      break;
    case SET_PIN_MODE:
      if(currentPinModeCallback)
        (*currentPinModeCallback)(storedInputData[1], storedInputData[0]);
      break;
    case REPORT_ANALOG:
      if(currentReportAnalogCallback)
        (*currentReportAnalogCallback)(multiByteChannel,storedInputData[0]);
      break;
    case REPORT_DIGITAL:
      if(currentReportDigitalCallback)
        (*currentReportDigitalCallback)(multiByteChannel,storedInputData[0]);
      break;
    }
    executeMultiByteCommand = 0;
  }
} else {
  // remove channel info from command byte if less than 0xF0
  if(inputData < 0xF0) {
    command = inputData & 0xF0;
    multiByteChannel = inputData & 0x0F;
  } else {
    command = inputData;
    // commands in the 0xF* range don't use channel data
  }
  switch (command) {
  case ANALOG_MESSAGE:
  case DIGITAL_MESSAGE:
```

```
    case SET_PIN_MODE:
      waitForData = 2; // two data bytes needed
      executeMultiByteCommand = command;
      break;
    case REPORT_ANALOG:
    case REPORT_DIGITAL:
      waitForData = 1; // two data bytes needed
      executeMultiByteCommand = command;
      break;
    case START_SYSEX:
      parsingSysex = true;
      sysexBytesRead = 0;
      break;
    case SYSTEM_RESET:
      systemReset();
      break;
    case REPORT_VERSION:
      Firmata.printVersion();
      break;
    }
  }
}
//------------------------------------------------------------------------------
// Serial Send Handling
// send an analog message
void FirmataClass::sendAnalog(byte pin, int value)
{
  // pin can only be 0-15, so chop higher bits
  FirmataSerial->write(ANALOG_MESSAGE | (pin & 0xF));
  sendValueAsTwo7bitBytes(value);
}
// send a single digital pin in a digital message
void FirmataClass::sendDigital(byte pin, int value)
{
  /* TODO add single pin digital messages to the protocol, this needs to
   * track the last digital data sent so that it can be sure to change just
   * one bit in the packet.  This is complicated by the fact that the
   * numbering of the pins will probably differ on Arduino, Wiring, and
   * other boards.  The DIGITAL_MESSAGE sends 14 bits at a time, but it is
   * probably easier to send 8 bit ports for any board with more than 14
   * digital pins.
   */
  // TODO: the digital message should not be sent on the serial port every
```

```
  // time sendDigital() is called.  Instead, it should add it to an int
  // which will be sent on a schedule.  If a pin changes more than once
  // before the digital message is sent on the serial port, it should send a
  // digital message for each change.
  //    if(value == 0)
  //        sendDigitalPortPair();
}
// send 14-bits in a single digital message (protocol v1)
// send an 8-bit port in a single digital message (protocol v2)
void FirmataClass::sendDigitalPort(byte portNumber, int portData)
{
  FirmataSerial->write(DIGITAL_MESSAGE | (portNumber & 0xF));
  FirmataSerial->write((byte)portData % 128); // Tx bits 0-6
  FirmataSerial->write(portData >> 7);  // Tx bits 7-13
}
void FirmataClass::sendSysex(byte command, byte bytec, byte* bytev)
{
  byte i;
  startSysex();
  FirmataSerial->write(command);
  for(i=0; i<bytec; i++) {
    sendValueAsTwo7bitBytes(bytev[i]);
  }
  endSysex();
}
void FirmataClass::sendString(byte command, const char* string)
{
  sendSysex(command, strlen(string), (byte *)string);
}
// send a string as the protocol string type
void FirmataClass::sendString(const char* string)
{
  sendString(STRING_DATA, string);
}
// expose the write method
void FirmataClass::write(byte c)
{
  FirmataSerial->write(c);
}
// Internal Actions/////////////////////////////////////////////////////////
// generic callbacks
void FirmataClass::attach(byte command, callbackFunction newFunction)
{
```

```
  switch(command) {
  case ANALOG_MESSAGE: currentAnalogCallback = newFunction; break;
  case DIGITAL_MESSAGE: currentDigitalCallback = newFunction; break;
  case REPORT_ANALOG: currentReportAnalogCallback = newFunction; break;
  case REPORT_DIGITAL: currentReportDigitalCallback = newFunction; break;
  case SET_PIN_MODE: currentPinModeCallback = newFunction; break;
  }
}
void FirmataClass::attach(byte command, systemResetCallbackFunction newFunction)
{
  switch(command) {
  case SYSTEM_RESET: currentSystemResetCallback = newFunction; break;
  }
}
void FirmataClass::attach(byte command, stringCallbackFunction newFunction)
{
  switch(command) {
  case STRING_DATA: currentStringCallback = newFunction; break;
  }
}
void FirmataClass::attach(byte command, sysexCallbackFunction newFunction)
{
  currentSysexCallback = newFunction;
}
void FirmataClass::detach(byte command)
{
  switch(command) {
  case SYSTEM_RESET: currentSystemResetCallback = NULL; break;
  case STRING_DATA: currentStringCallback = NULL; break;
  case START_SYSEX: currentSysexCallback = NULL; break;
  default:
    attach(command, (callbackFunction)NULL);
  }
}
// sysex callbacks
/*
 * this is too complicated for analogReceive, but maybe for Sysex?
 void FirmataClass::attachSysex(sysexFunction newFunction)
 {
 byte i;
 byte tmpCount = analogReceiveFunctionCount;
 analogReceiveFunction* tmpArray = analogReceiveFunctionArray;
 analogReceiveFunctionCount++;
```

```
 analogReceiveFunctionArray = (analogReceiveFunction*)
calloc(analogReceiveFunctionCount, sizeof(analogReceiveFunction));
 for(i = 0; i < tmpCount; i++) {
 analogReceiveFunctionArray[i] = tmpArray[i];
 }
 analogReceiveFunctionArray[tmpCount] = newFunction;
 free(tmpArray);
 }
*/
//*****************************************************************************
//* Private Methods
//*****************************************************************************
// resets the system state upon a SYSTEM_RESET message from the host software
void FirmataClass::systemReset(void)
{
  byte i;
  waitForData = 0; // this flag says the next serial input will be data
  executeMultiByteCommand = 0; // execute this after getting multi-byte data
  multiByteChannel = 0; // channel data for multiByteCommands
  for(i=0; i<MAX_DATA_BYTES; i++) {
    storedInputData[i] = 0;
  }
  parsingSysex = false;
  sysexBytesRead = 0;
  if(currentSystemResetCallback)
    (*currentSystemResetCallback)();
  //flush(); //TODO uncomment when Firmata is a subclass of HardwareSerial
}
// ============================================================================
// used for flashing the pin for the version number
void FirmataClass::strobeBlinkPin(int count, int onInterval, int offInterval)
{
  byte i;
  pinMode(VERSION_BLINK_PIN, OUTPUT);
  for(i=0; i<count; i++) {
    delay(offInterval);
    digitalWrite(VERSION_BLINK_PIN, HIGH);
    delay(onInterval);
    digitalWrite(VERSION_BLINK_PIN, LOW);
  }
}
// make one instance for the user to use
FirmataClass Firmata;
```