UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RODOLFO PEDÓ PIROTTI

Arquitetura e implementação aberta de um sintetizador subtrativo e aditivo para plataforma de baixo custo

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Pimenta
Coorientador: Prof. Dr. Marcelo Johann

Porto Alegre
2017

CIP - Catalogação na Publicação

Pirotti, Rodolfo Pedó
Arquitetura e implementação aberta de um sintetizador subtrativo e aditivo para plataforma de baixo custo / Rodolfo Pedó Pirotti. -- 2017.
99 f.

Orientador: Marcelo Soares Pimenta.
Coorientador: Marcelo de Oliveira Johann.

Dissertação (Mestrado) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2017.

1. Computação e música. 2. Sintetizadores. 3. Processamento digital de áudio. 4. Arduino Due. 5. DIY. I. Pimenta, Marcelo Soares, orient. II. Johann, Marcelo de Oliveira, coorient. III. Título.

AGRADECIMENTOS

Dedico este trabalho à minha família e em especial à minha esposa Tatiane.

Agradeço também aos meus orientadores, Marcelo Pimenta e Marcelo Johann, por todo auxílio e disponibilidade sempre que foi preciso, com dicas, sugestões, discussões técnicas e musicais, e ao meu amigo Sérgio Bordini, que sempre me inspirou e ajudou quando o assunto era o projeto e desenvolvimento de instrumentos musicais eletrônicos.

RESUMO

Existem inúmeras técnicas de síntese de áudio utilizadas atualmente em instrumentos musicais profissionais, dentre as quais as mais fundamentais são a síntese aditiva e a síntese subtrativa. A síntese subtrativa se tornou popular e foi muito explorada entre as décadas de 60 e 70 com a criação de módulos analógicos de *hardware* que podiam ser interconectados, criando o conceito de sintetizador analógico modular. Apesar do uso deste tipo de sintetizador ter diminuído durante as décadas subsequentes, nos últimos anos sua utilização voltou a crescer e diversos modelos deste tipo de instrumento são vendidos atualmente, porém em geral a preços elevados. Sintetizadores digitais também disponibilizam a técnica de síntese subtrativa utilizando componentes eletrônicos customizados e desenvolvidos pelos fabricantes de sintetizadores com o intuito de utilizar avançadas técnicas de processamento de sinais, o que ainda mantém seus preços elevados.

Neste trabalho investigamos a hipótese de que é possível desenvolver um instrumento musical funcional e de qualidade com recursos limitados de processamento, e exploramos essa hipótese implementando síntese subtrativa em uma plataforma acessível e de baixo custo. O desenvolvimento é baseado em linguagem orientada a objetos para criação de módulos de *software* replicando as características dos módulos encontrados em sintetizadores analógicos modulares. Com esta abordagem, obtemos um software modular que pode ser facilmente modificado baseado nas preferências do programador.

A implementação foi testada na plataforma Arduino Due, que é uma plataforma de baixo custo e contém um processador 32-bits ARM 84 MHz. Foi possível adicionar osciladores com algoritmo *anti-aliasing*, filtros, geradores de envelope, módulo de efeito, uma interface MIDI e um teclado externo, obtendo assim um sintetizador subtrativo completo. Além disto, incluímos no desenvolvimento a implementação de um órgão baseado em síntese aditiva, com polifonia completa e inspirado na arquitetura de órgãos clássicos, mostrando a possibilidade de possuir dois importantes e poderosos métodos de síntese em uma plataforma acessível e de baixo custo.

Com esta implementação aberta e pública, buscamos contribuir com o movimento *maker* e faça-você-mesmo, incentivando novos desenvolvimentos nesta área, em especial na computação e engenharia, aumentando o uso e acesso a instrumentos musicais eletrônicos e a criatividade musical.

Palavras-chave: Computação e música. Sintetizadores. Processamento digital de áudio. Arduino Due. DIY.

An open design and implementation of a subtractive and additive synthesizer for low cost platforms

ABSTRACT

Subtractive and additive synthesis are two powerful sound synthesis techniques that caused a revolution when the first electronic and electro mechanic music instruments started to appear some decades ago. Subtractive synthesis became very popular during the 60s and 70s after the creation of analog hardware modules that could be interconnected, creating the concept of the modular synthesizers. After the initial impact, for some years these instruments faced a slow-down in its usage, a tendency that was reverted on the past decade. Nevertheless, the prices of these instruments are often high. Digital synthesizers also offer the subtractive synthesis technique, by using customized electronic components designed and developed by the synthesizers vendors in order to use the most up-to-date technologies and signal processing techniques, which also leads to high prices.

In this project, we investigate the hypothesis that it is possible to design and develop a good quality music instrument with low budget electronic components and limited processing capabilities, by implementing this on a low budget and easy to use platform. The development is based on object oriented design, creating software modules that replicates the functionalities of analog synthesizer hardware modules. With this approach, we have a modular software that can be easily changed based on programmers' preferences.

The implementation was tested on the Arduino Due board, which is a cheap, easy to use and widely available platform and powered by a 32-bits ARM 84Mhz processor. We were able to add oscillators with anti-aliasing algorithms, filters, envelope generators, delay effects, a MIDI interface and a keybed, making a complete synthesizer. In addition to this, we included an additive synthesis organ design with full polyphony based on classic organs design, demonstrating the possibility of having two powerful synthesis methods on a cheap and widely available platform.

With this design, suitable for low cost platforms, we intend to contribute to the maker movement and encourage new implementations in this area, especially in the computing and engineering fields, increasing the usage and access to (electronic) musical instruments and musical creativity.

Keywords: Music computing. Synthesizers. Digital audio processing. Arduino Due. DIY.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD – Analógico-digital

ADSR – *Amplitude Decay Sustain Release*

ASIC – *Application Specific Integrated Circuits*

BLIT – *Band Limited Impulse Train*

BLEP – *Band Limited Step Function*

DA – Digital-analógico

DCF – *Digital Controlled Filter*

DCO – *Digital Controlled Oscillator*

DIY – *Do It Yourself*

DSP – *Digital Signal Processor*

EG – *Envelope Generator*

FLOP – *Floating point operations*

FM – *Frequency Modulation*

LFO – *Low Frequency Oscillator*

NCO – *Numerically Controlled Oscillator*

ND – Não disponível

PolyBLEP – *Polynomial Bandlimited Step Function*

PWM – *Pulse Width Modulation*

VA – *Virtual Analog*

VCA – *Voltage Controlled Amplifier*

VCF – *Voltage Controller Filter*

VCO – *Voltage Controlled Oscillator*

SUMÁRIO

1 INTRODUÇÃO

Os instrumentos musicais elétricos ou eletrônicos possuem uma história recente se comparados a outros instrumentos musicais como flauta, percussão ou piano. Enquanto é possível encontrar registros destes (e outros) com datas estimadas de séculos ou milênios atrás, os primeiros instrumentos que utilizam um sinal elétrico para gerar um sinal sonoro começaram a aparecer apenas no final do século XIX e começo do século XX, após as primeiras descobertas no campo da eletricidade. Alguns exemplos são o *Electric Telegraph for Transmitting Musical Tones*, patenteado em 1875 por Gray (USPTO, 1875), o *Telharmonium*, patenteado em 1897 por Cahill (USPTO, 1897), o *Theremin*, patenteado por Theremin (USPTO, 1928) em 1928, e o órgão *Hammond*, patenteado por Hammond (USPTO, 1934) em 1934.

Nas décadas seguintes, diversas tentativas foram realizadas com o objetivo de criar outros instrumentos musicais eletrônicos, muitas delas a partir de equipamentos eletrônicos de teste, geradores de sinais ou componentes de rádio. Em 1964, Robert Moog apresentou um sintetizador eletrônico analógico modular que utilizava síntese subtrativa e possuía um teclado para execução das notas musicais similar ao de um piano. A sonoridade do sintetizador apresentado por Moog, aliada à facilidade de controle da geração do som e a arquitetura modular flexível causaram grande impacto na música. (HARLEY, 2005) (JENKINS, 2007).

Além dos sintetizadores "Moog", outros fabricantes começaram a produzir e comercializar, em especial nas décadas de 60 e 70, sintetizadores modulares com este tipo de síntese. Isto abriu uma nova possibilidade sonora para a época, e estes instrumentos passaram a ser utilizados por diversos músicos como Wendy Carlos, Rick Wakeman e Keith Emerson, que exploraram amplamente os timbres característicos que este tipo de instrumento possui. Wendy Carlos, por exemplo, lançou em 1968 o álbum "*Switched on Bach*", contendo músicas compostas por J. S. Bach porém interpretadas e gravadas utilizando sintetizadores Moog. Foi o primeiro álbum de música erudita a receber o certificado de platina por vendas nos Estados Unidos (atingiu a marca de 1 milhão de cópias vendidas). (CARLOS, 2016).

Com o advento dos instrumentos eletrônicos digitais a partir dos anos 80, estes instrumentos puramente analógicos foram perdendo espaço, em especial devido à ampla possibilidade de programações que um instrumento digital oferecia. Poucos modelos de sintetizadores analógicos estavam em produção no final dos anos 80. Nos últimos anos, porém, têm-se observado um renascimento da sonoridade característica destes sintetizadores na música, fazendo com que alguns fabricantes retomassem o projeto e comercialização de sintetizadores analógicos, e também não é raro encontrar instrumentos antigos puramente

analógicos com um preço superior ao da venda na época de fabricação. (WIFFEN, 1997) (JENKINS, 2007).

Em sintetizadores digitais modernos existe o conceito de VA (*Virtual Analog*), que simula os princípios de síntese e arquitetura dos sintetizadores analógicos originais visando a reprodução do mesmo tipo de sonoridade, utilizando componentes eletrônicos customizados e desenvolvidos pelos fabricantes de sintetizadores com o intuito de utilizar avançadas técnicas de processamento de sinais (o que torna o preço destes instrumentos, assim como os analógicos, elevados). Além disto, muitos instrumentos (sintetizadores) digitais disponíveis hoje em dia oferecem pouca liberdade para edição da síntese sonora, sendo chamados muitas vezes de *rompler* (abreviação de *ROM Memory Player*, que seriam instrumentos que reproduzem uma forma de onda previamente gravada em memória e com pouca possibilidade de edição da mesma). Assim, um instrumento pode reproduzir diversos tipos de sons, simulando instrumentos como piano, violão, saxofone, xilofone e outros, porém dando pouca liberdade ao músico para criar sua própria sonoridade de forma original.

O avanço da eletrônica e computação têm permitido a miniaturização e popularização de plataformas de desenvolvimento, processadores e kits de aprendizado com capacidade de processamento e armazenamento superiores às encontradas poucos anos atrás em computadores *desktop*. Atualmente é possível comprar, por um preço acessível, uma placa eletrônica (em geral do mesmo tamanho ou menor do que um aparelho de telefone celular) contendo processador e periféricos "prontos para serem usados". A popularização destas plataformas de desenvolvimento está diretamente ligada ao crescimento do que está sendo chamado de "*DIY culture*" (*do-it-yourself culture*, ou cultura faça você mesmo, em português) ou "*Maker Movement*" (movimento de fazedores, numa tradução livre). Estes termos são utilizados para descrever pessoas ou comunidades que buscam projetar, desenvolver, construir, montar ou consertar itens diversos utilizando suas próprias habilidades, conhecimentos e ferramentas, ou então buscando isto (aprendizado, ferramentas) através de outras pessoas e sem motivação comercial. (KUZNETSOV; PAULOS, 2010).

Os autores Chen (2009) e Gibb (2014) apontam que os processos modernos de fabricação e produção mundial, muito focados em redução de custos e facilidade de produção, acabam por tornar os produtos cada vez mais similares entre si, e a internacionalização faz com que haja uma redução da identidade regional presente em cada item e produto fabricado. Gibb (2014) enfatiza ainda que a cultura da facilidade de produção e redução de custo dos produtos fez a sociedade se acostumar a consumir mais e realizar menos. Porém, o mesmo autor cita que "Nos últimos 10 anos, o pêndulo começou a mudar de lado novamente em favor da criação e

conserto de coisas em vez de apenas aquisição". Assim, neste contexto, o movimento (ou cultura) *DIY*, ou *Maker Movement*, surge e cresce cada vez mais como resposta a esta falta de identidade regional e pessoal nos produtos; como forma das pessoas exercerem sua criatividade e demonstrá-la através de criações das mais diversas formas; tentando democratizar e facilitar o acesso à tecnologia; aumentando e agilizando o compartilhamento de conhecimento e facilitando o aprendizado. (JORDAN; LANDE, 2013) (LINDTNER; BARDZELL; BARDZELL, 2016). Kuznetsov e Paulos (2010) afirmam que as comunidades de *software* livre e código aberto são exemplos deste movimento.

Para demonstrar o crescimento e importância deste movimento, apresentamos alguns números relacionados à cultura DIY:

- até maio de 2014, em torno de 1,2 milhões de placas Arduino já haviam sido vendidas (e estima-se que exista um número similar ou até maior de cópias não-oficiais, não inclusas nesta conta) (ORSINI, 2014);
- em um período de 3 meses, entre 4 a 5 milhões de usuários acessaram a página oficial da Arduino (ORSINI, 2014);
- a plataforma Raspberry Pi já vendeu 10 milhões de unidades nos últimos 4 anos e meio (UPTON, 2016);
- O mercado voltado a hobbies pessoais vale mais de U$ 25 bilhões nos Estados Unidos (HATCH, 2014);
- A revista *Make Magazine*, criada em 2005 nos Estados Unidos voltado ao público *Maker*, teve circulação de 70 mil unidades no primeiro ano. Em 2009, estava em 125 mil, sendo que em 2013 subiu para 315 mil unidades impressas, um número considerado alto para este tipo de mídia (STONE, 2015);
- A feira *Maker Faire*, voltada ao público *maker*, iniciada em 2006 na cidade de San Mateo, CA, EUA, iniciou com 22 mil visitantes. Oito anos depois, em 2014, teve 134 feiras associadas espalhadas ao redor do mundo, e um público visitante total de 781 mil pessoas (STONE, 2015) (MAKER, sem data);
- A empresa norte-americana TechShop, que oferece cursos e ferramentas para a comunidade DIY, foi fundada em 2006, e em meados de 2015 já possuía 6.500 associados, oito filiais nos Estados Unidos e um faturamento anual de 15 milhões de dólares (FELDMAN, 2015).

Considerando a importância da síntese subtrativa no cenário musical e o crescente movimento faça-você-mesmo, propomos neste trabalho investigar a hipótese de que seria

possível desenvolver um sintetizador musical funcional de qualidade com recursos limitados de processamento, contrapondo a premissa de que instrumentos digitais de qualidade necessitam de componentes caros e complexos. Investigamos essa hipótese desenvolvendo uma arquitetura e adaptando algoritmos para uso na plataforma Arduino Due em função do seu baixo custo e facilidade de acesso. O desenvolvimento levou em consideração o uso de boas práticas de engenharia de *software*, de forma a facilitar a modificação de características do sintetizador conforme desejo do programador ou do músico. O objetivo final é obter um sistema modular (do ponto de vista de *software*), flexível e com alta qualidade de áudio.

Com este desenvolvimento, será possível obter funcionalidades e controles da síntese sonora similares aos sintetizadores analógicos que marcaram época, além de permitir a adição de recursos de acordo com a preferência de cada pessoa em função de uma engenharia de *software* que permita facilmente a modificação do código. Isto abre possibilidades para que novos tipos de sons sejam criados, testados e utilizados, contrapondo instrumentos musicais com poucas possibilidades de criação sonora. Também buscamos a possibilidade de acesso a este tipo de instrumento musical a um baixo custo, facilitando o seu uso e ensino. Acreditamos que com isso possamos contribuir para o fortalecimento da cultura DIY e *Maker Movement* dentro da computação e engenharia, auxiliando e incentivando pessoas a desenvolver e construir seus próprios equipamentos e sintetizadores, e incentivando também a cultura e criatividade musical.

Os resultados obtidos com esta pesquisa, bem como os detalhes do desenvolvimento, código fonte e esquemáticos, serão divulgados e publicados com o intuito de contribuir com o *Maker Movement*.

1.1 Sintetizador aditivo

Johann (2015) apresentou, utilizando o Arduino, o desenvolvimento de um sintetizador baseado em síntese aditiva e com inspiração nos órgãos Hammond, também de grande importância no cenário musical e especialmente do rock, desde os anos 60 até hoje (embora os órgãos Hammond tenham surgido na década de 40). Buscando demonstrar outras possibilidades sonoras, propomos a adição do desenvolvimento de Johann (2015) ao desenvolvimento do sintetizador subtrativo. Assim, em uma única plataforma Arduino, dois métodos importantes de síntese estarão disponíveis, podendo ser selecionados em tempo real pelo usuário.

Ressaltamos que, devido ao fato do desenvolvimento original de Johann (2015) ser independente do desenvolvimento aqui descrito, não serão fornecidos detalhes técnicos desta

implementação, mas apenas a indicação de como foi possível incluir este outro método de síntese na mesma plataforma Arduino e o resultado final de uso.

1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Desenvolver um sintetizador utilizando síntese subtrativa tendo como referência características típicas de sintetizadores analógicos conhecidos;
- Utilizar como base para o desenvolvimento uma plataforma acessível e de baixo custo – neste caso, foi escolhido o Arduino Due;
- Identificar métodos e algoritmos apropriados para a implementação em uma plataforma com recursos limitados de processamento e memória.
- Buscar um desenvolvimento modular (através de boas práticas de engenharia de *software*) que facilite a modificação e adição de novas características, permitindo a experimentação de novos tipos de sonoridade;
- Obter uma alta qualidade no resultado final do áudio e do desenvolvimento, para que o sintetizador possa ser utilizado como um instrumento em condições reais de uso;
- Integrar à síntese subtrativa outra forma de síntese já desenvolvida, a síntese aditiva (conforme já realizado por Johann (2015)), mostrando que é possível implementar mais de um método de síntese utilizando a mesma plataforma;
- Contribuir com o movimento DIY e *Maker Movement*, incentivando o desenvolvimento de equipamentos próprios, de trabalhos inovadores com estas plataformas, em especial para as áreas de computação e engenharia, e o uso das mesmas para o aumento da criatividade musical e acessibilidade ao estudo da música.

1.3 Organização desta dissertação

O segundo capítulo apresenta alguns trabalhos relacionados e similares com a implementação aqui proposta. É apresentada uma breve descrição destes trabalhos bem como uma tabela comparativa, posicionando cada um em comparação com nossa solução e apresentando as principais diferenças.

No terceiro capítulo apresentamos uma revisão sobre os métodos de síntese aditiva e subtrativa. Em seguida, apresentamos os principais conceitos envolvendo um sintetizador

subtrativo e seus módulos, para facilitar a compreensão da implementação e arquitetura proposta mais adiante. Por fim, apresentamos uma descrição dos principais componentes do órgão desenvolvido por Laures Hammond e Johann (2015), que serviram como base para implementação da síntese aditiva deste trabalho.

O capítulo quatro descreve algumas plataformas de desenvolvimento muito utilizadas atualmente, e serve para posicionar o leitor da principal diferença entre a plataforma Arduino e outras plataformas populares e muito utilizadas para projetos eletrônicos DIY.

O capítulo cinco é voltado ao processamento digital de áudio, apresentando os principais conceitos e alguns dos desafios enfrentados para manipulação de sinais de áudio em componentes digitais.

O capítulo seis é voltado à implementação aqui proposta. São apresentados os diagramas dos módulos desenvolvidos acompanhados de uma descrição do funcionamento e interfaces de uso. Ainda neste capítulo, descrevemos também como adicionamos a possibilidade de teste e simulação da implementação fora de um ambiente embarcado, alguns exemplos de como os diferentes módulos podem ser interligados e a forma que encontramos para compartilhar este desenvolvimento para contribuir com o movimento *maker*.

No capítulo sete apresentamos os resultados da implementação, dentre eles a medição de sinais de saída e tempos de processamento de alguns algoritmos. Apresentamos ainda o diagrama esquemático utilizado no protótipo de teste.

O capítulo oito apresenta por fim as conclusões deste desenvolvimento, contribuições e propostas de desenvolvimentos futuros.

2 TRABALHOS RELACIONADOS

A tentativa de projeto e montagem de um sintetizador utilizando síntese subtrativa e buscando recursos semelhantes a sintetizadores analógicos clássicos não é novidade. Uma rápida busca em ferramentas como *Google* e em bases de dados de artigos e publicações científicas, tais como *IEEE Xplore Digital Library* e *ACM Digital Library*, nos apresentam diversos projetos de sintetizadores e análises de algoritmos, filtros e outros componentes destes equipamentos.

É possível encontrar projetos utilizando circuitos analógicos, DSPs (*Digital Signal Processing*, processadores voltados ao processamento digital de sinais), modelagem matemática de filtros, ou mesmo alguns projetos utilizando placas de desenvolvimento como Arduino. Alguns são completamente abertos, possuindo diagrama de circuitos e código fonte, enquanto outros são fechados e não indicam detalhes do seu desenvolvimento. Apesar disto, não encontramos um desenvolvimento que possua os requisitos e objetivos deste trabalho, que são, como já mencionado anteriormente, o desenvolvimento de um sintetizador utilizando uma plataforma acessível e de baixo custo, com alta qualidade no resultado final do áudio e apresentando de forma didática como desenvolver isto. Para efeito de comparação entre alguns dos trabalhos similares encontrados, apresentamos uma classificação dos mesmos de acordo com os seguintes itens:

- Complexidade de montagem: indica o quão difícil é a montagem ou criação do sintetizador, assumindo pessoas com conhecimento básico, mas pequeno, de eletrônica e informática;
- Custo total: custo necessário para montagem ou utilização do sintetizador;
- Objetivo principal de uso: neste item consideramos se o objetivo é o aprendizado de noções básicas de áudio e música, sem preocupação com a qualidade final – chamamos de APRENDIZADO; se o objetivo principal é obter um instrumento para uso real (execuções musicais) – chamamos de INSTRUMENTO;
- Qualidade de saída: neste item consideramos como qualidade alta se o sinal de saída apresenta uma resolução mínima de 16 bits, taxa de amostragem de 44 kHz e baixo nível de ruído. Para resolução inferior a 16 bits ou taxa de amostragem inferior a 44 kHz, consideramos como qualidade baixa.

Tabela 2-1 – Comparativo com trabalhos relacionados

Autor	Barrass (2012)	Huovilainen (2010)	Biddulph e Ziembicki (2000)	Este trabalho
Montagem	Médio	ND	Fácil	Médio
Custo	Custo de um Arduino Uno (U$ 22,00 fora do Brasil), mais componentes para montagem externa.	ND	O produto montado é vendido por aproximadamente U$ 266,00 fora do Brasil	Custo de um Arduino Due (U$ 30,00 fora do Brasil), mais componentes para montagem externa.
Objetivo	Aprendizado	ND	Aprendizado	Instrumento
Saída	Baixa	ND	Baixa	Alta

Fonte: elaborada pelo autor

Tendo em vista as referências supracitadas, a seguir descrevemos brevemente os principais trabalhos relacionados encontrados.

2.1 MOZZI Sound Synthesis Library for Arduino

Barrass (2012) fornece uma biblioteca de funções para processamento de áudio no Arduino. Os modelos suportados e testados são apenas os modelos baseados em processador Atmel AVR 8 bits, rodando a 8 MHz ou 16 MHz, dependendo do modelo. A geração de áudio de saída é realizada em 16.384 Hz. Apesar de fornecer uma extensa base de código, os algoritmos utilizados são simples; o filtro passa-baixa possui um erro no código-fonte da aplicação da ressonância (encontrado pelo autor deste trabalho); a biblioteca não resolve o problema de *aliasing.* Além disto, a qualidade do sinal de saída é baixa, pois utiliza uma saída PWM com integrador, que não oferece boa resolução nem baixo índice de ruído.

2.2 Scalable Polyphonic MIDI-Synthesizer for a Low-Cost DSP

Huovilainen (2010) apresenta um projeto de sintetizador, que utiliza principalmente síntese subtrativa (porém em conjunto com outras técnicas de síntese), para implementação utilizando um DSP (VLSI Solution VS1003). São apresentados algoritmos para geração de osciladores, filtros, geradores de envelope e outros componentes, com análise matemática dos mesmos e considerações sobre a implementação utilizando o DSP. Nenhuma implementação foi realizada de fato, e o autor enfatiza que alguns testes foram realizados com MATLAB apenas.

2.3 AVRSynth

Biddulph e Ziembicki (2000) projetaram um sintetizador subtrativo monofônico em um microcontrolador da Atmel. O áudio de saída é gerado com taxa de amostragem de 31.250 Hz. Os autores vendem os componentes para quem quiser montar um, fornecendo diagramas e instruções. Três opções de venda são oferecidas: compra apenas dos componentes eletrônicos; componentes eletrônicos já montados na placa de circuito impresso; projeto completo, incluindo um painel frontal com botões. Os autores disponibilizam apenas três pequenas amostras do áudio de saída (cada amostra em torno de 3 segundos de áudio).

2.4 Outros projetos

Como mencionado no início deste capítulo, existem diversas iniciativas de construção de sintetizadores analógicos ou digitais usando síntese subtrativa. Algumas não possuem comprometimento com a qualidade ou fidelidade, sendo apenas para divertimento ou aprendizado. Outras buscam qualidade, mas atingem um alto custo ou complexidade para desenvolvimento. Dentre todas estas, existem ainda as que possuem o seu desenvolvimento aberto ou fechado (ou vendidas como um pacote fechado).

Além dos citados anteriormente, acrescentamos trabalhos como o *blog* mantido por Finke (2013) que, apesar de expor o código-fonte de um sintetizador subtrativo para rodar como um *plug-in* em um computador, possui exemplos e algoritmos que, com certas adaptações, podem ser utilizados como referência para o desenvolvimento de um sintetizador para plataformas de baixo custo. Citamos também projetos totalmente analógicos como o apresentado por Usson (2016), grupos de discussão como *modular.br*[1] e sites como *www.sdiy.org* que fornecem links para outros sites com conteúdo relacionado a montagem de sintetizadores ou instrumentos musicais, sejam digitais ou analógicos. Estes últimos servem como uma boa referência para a modelagem dos componentes (se software) e um melhor entendimento das características de um sintetizador subtrativo.

2.5 Síntese FM com Arduino Due

Folle (2015) apresentou a implementação de outro tipo de síntese muito utilizada, a síntese FM, com o Arduino Due. A síntese FM foi proposta por John Chowning em 1967 e

[1] "modular.br" é um grupo fechado no Facebook, que no dia 25/09/2016 possuía 1.108 membros, em sua maioria brasileiros, com o objetivo de trocar projetos, experiências e análises de sintetizadores modulares puramente analógicos.

utilizada no sintetizador Yamaha DX7. A implementação deste sintetizador era em ponto fixo, pois não existiam na época DSPs ou processadores (viáveis de serem utilizados comercialmente) com capacidade para efetuar o número necessário de multiplicações. Os desenvolvedores utilizaram então ASICs dedicados e utilizaram somas e tabelas para realizar todo o processamento apenas com ponto fixo.

O autor mostrou ser possível implementar 6 operadores e todos os recursos de geração correspondentes à síntese FM conforme implementada no sintetizador Yamaha DX7, obtendo uma polifonia de até 6 notas. Apesar de ser outro tipo de síntese da que propomos aqui, Folle (2015) demonstrou como implementar síntese FM no Arduino Due, sendo que podemos futuramente utilizar isto como referência para adicionar à nossa implementação, adicionando outros métodos de síntese em uma mesma plataforma e em um mesmo instrumento.

3 SINTETIZADORES MUSICAIS

A definição do que é um sintetizador musical varia um pouco de acordo com alguns autores. Devahari (1982) apresenta um sintetizador musical como sendo um instrumento eletrônico capaz de produzir uma grande variedade de sons através da combinação de frequências. Já Crombie (1982) define que um sintetizador é um dispositivo que gera som através da determinação dos elementos fundamentais de frequência, timbre e intensidade.

A síntese sonora é o modelo, processo ou algoritmo utilizado para geração de som. Diferentes técnicas de síntese foram desenvolvidas e são utilizadas até hoje. A seguir serão revisados os conceitos dos principais métodos de síntese.

3.1 Síntese Aditiva

Roads (1996) define síntese aditiva como uma técnica de síntese sonora baseada na soma de formas de onda básicas (ou elementares) para criação de uma forma de onda mais complexa. De acordo com Roads (1996), a síntese aditiva é uma das mais antigas e mais pesquisadas técnicas de síntese. O poder desta técnica deriva do fato de que, teoricamente, é possível obter qualquer forma de onda periódica a partir da soma de formas de onda elementares.

A Série de Fourier demonstra este princípio no qual através da adição de infinitas ondas senoidais de diferentes frequências, amplitudes e fases, é possível obter qualquer sinal periódico (para aplicações concretas a série é truncada). No caso da síntese aditiva, o sinal de saída, resultado da soma de todos os senos, pode ser calculado de acordo com a equação 1.

$$s(t) = \sum_{i=1}^{N} A_i . \sin(2\pi f_i t + \emptyset_i) \qquad (1)$$

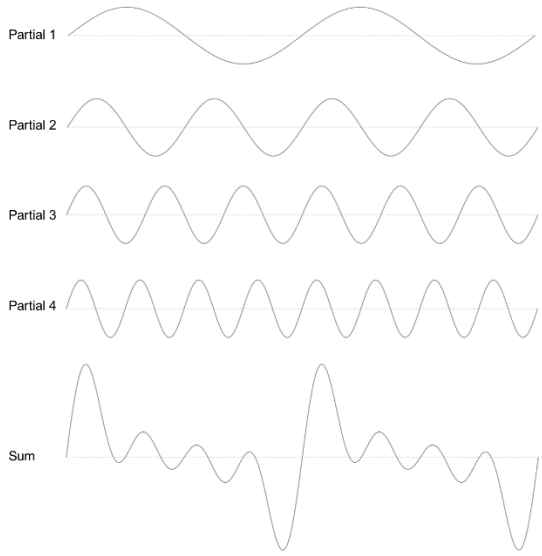
Na equação 1, A_i é a amplitude do sinal, f_i a frequência, $\emptyset_i$ a defasagem e N o número de harmônicas a serem utilizadas. A variedade de sons que podem ser gerados (sintetizados) com esta técnica é enorme, desde que um grande número de senos seja utilizado. Por outro lado, a complexidade computacional deste método é proporcional ao número de ondas necessárias para gerar o sinal de saída. Gerar a onda senoidal para cada frequência existente no sinal pode se tornar complexo do ponto de vista computacional, bem como para implementação digital em *hardware* ou *software*. Para obter sons mais naturais, ainda seria necessário variar de forma diferente a amplitude de cada frequência ao longo do tempo. (ROADS, 1996). Por ser baseado na equação da Série de Fourier, este método só pode gerar sons representados por formas de

onda periódicas – sinais como ruídos não podem ser sintetizados utilizando esta técnica (URBAN, 2002).

Esta técnica é utilizada, por exemplo, em órgãos de tubo. Com a passagem de ar pelos tubos, cada um produz uma forma de onda similar a uma senóide. Através de registros de controle, seleciona-se quais tubos serão utilizados para gerar o som e a "soma" é realizada no próprio ar. Laurens Hammond também utilizou a síntese aditiva para criação dos órgãos eletromecânicos Hammond, que influenciou a música a partir das décadas de 40 e 50. (PEKONEN; PIHLAJAMAKI; VALIMAKI, 2011).

A Figura 3-1 apresenta o sinal resultante de quatro ondas senoidais de frequências diferentes (porém com a mesma fase). Naturalmente, o espectro de frequências do sinal resultante apresentará as quatro componentes que foram somadas.

Figura 3-1 – Soma de ondas senoidais



Fonte: Cabrera, Heintz e Houdorf (2015)

O resultado de um sinal sintetizado com esta técnica tipicamente é um som mais suave, e pode ser utilizado para sintetizar alguns timbres encontrados em instrumentos naturais tais como flauta, oboé, voz humana (coro) e alguns tipos de cordas. Se por um lado esta técnica é de fácil compreensão, sua aplicação pode se tornar complexa dependendo do sinal a ser produzido. Devido a isto, este método de síntese, embora tenha se tornado muito popular especialmente devido aos órgãos Hammond, não é, geralmente, utilizada como única forma de síntese de som – geralmente outras técnicas são aplicadas em conjunto com a síntese aditiva. (URBAN, 2002).

3.2 Síntese Subtrativa

A síntese subtrativa baseia-se no uso de um sinal com um maior conteúdo harmônico do que uma senóide, como por exemplo, uma onda dente de serra, onda quadrada, onda triangular, dentre outras. A partir deste sinal de entrada, aplica-se um filtro que retira conteúdo harmônico do mesmo – daí o nome de síntese subtrativa. (URBAN, 2002).

Embora ainda hoje este método seja amplamente utilizado, nas décadas de 60 e 70 ocorreu o maior crescimento de uso do mesmo com a utilização dos sintetizadores analógicos e modulares na música. A Figura 3-2 apresenta o conceito básico da síntese subtrativa.

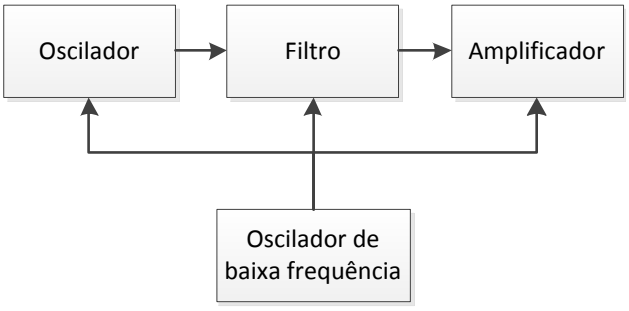
Figura 3-2 – Esquema básico da síntese subtrativa

Fonte: elaborada pelo autor

Na síntese subtrativa, tem-se um oscilador que gera uma forma de onda na frequência desejada. Em instrumentos analógicos, este oscilador é chamado de VCO (*voltage controlled oscilattor*, ou oscilador controlado por tensão). Após este oscilador, o sinal passa por um ou mais filtros (que pode ser um filtro passa-baixa, passa-faixa ou passa-alta, embora tipicamente seja aplicado o filtro passa-baixa) que irão retirar conteúdo harmônico do sinal original. Este filtro é chamado de VCF (*voltage controlled filter*, ou filtro controlado por tensão). Para implementações digitais, estes módulos são chamados de DCO (*digitally controlled oscilattor*) e DCF (*digitally controlled filter*) respectivamente. (URBAN, 2002).

Outro componente é o LFO (*low frequency oscilator*, oscilador de baixa frequência), que pode ser conectado ao VCO, VCF ou ao amplificador de saída (também controlado por tensão), para gerar efeitos em tempo real no sinal, como modulação em frequência ou em amplitude, conforme mostra a Figura 3-3.

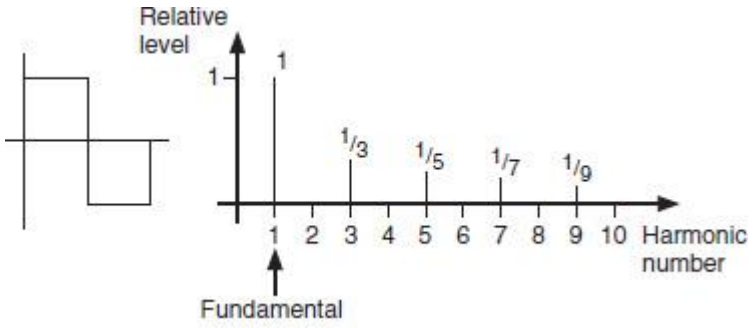
Figura 3-3 – Síntese subtrativa com LFO



Fonte: elaborada pelo autor

A Figura 3-4 apresenta a forma e o conteúdo harmônico (gráfico do espectro de frequências) de uma onda quadrada. Percebe-se que a onda quadrada possui apenas harmônicas ímpares em seu espectro, cuja amplitude decai com o aumento do número da harmônica.

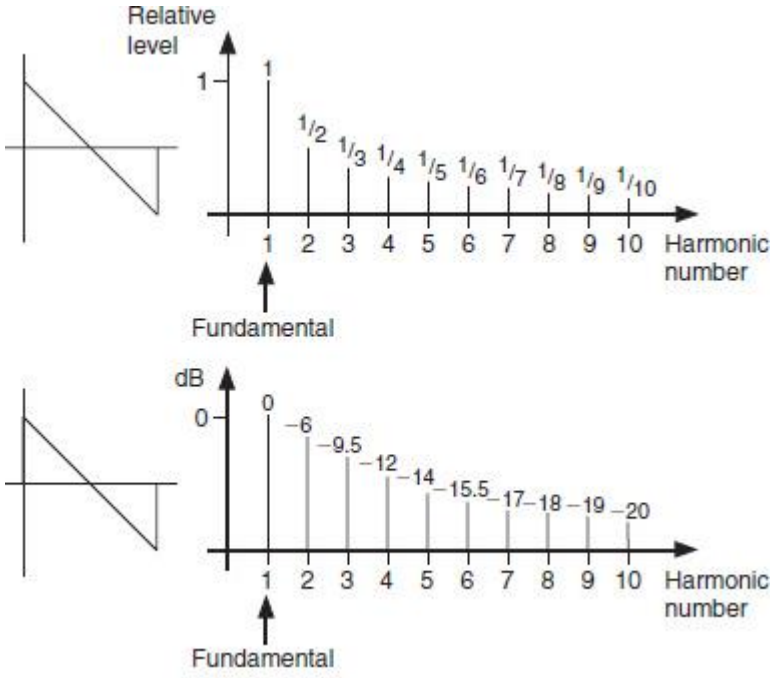
Figura 3-4 – Onda quadrada e seu espectro de frequências



Fonte: Russ (2004)

A Figura 3-5 apresenta uma onda dente-de-serra e seu conteúdo harmônico. A onda dente-de-serra possui todas as harmônicas em seu espectro, cujas amplitudes decaem com o aumento do número da harmônica.

Figura 3-5 – Onda dente-de-serra e seu conteúdo harmônico

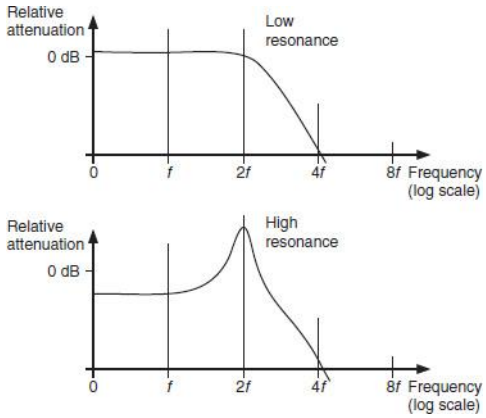


Fonte: Russ (2004)

O filtro utilizado em geral possui parâmetros ajustáveis que podem modificar o timbre final. Os dois principais parâmetros são a frequência de corte, que é a frequência a partir da qual o filtro gera uma atenuação de 3dB no nível do sinal, e a frequência de ressonância, que é a frequência na qual o filtro amplifica o sinal ao invés de atenuar. A Figura 3-6 apresenta a

resposta em frequência de um filtro passa-baixa. Na imagem superior, o filtro praticamente não possui ressonância em nenhuma frequência. Na imagem inferior, há uma determinada frequência na qual o filtro amplifica o sinal ao invés de reduzi-lo – esta é a frequência de ressonância.

Figura 3-6 – Resposta em frequência de um filtro passa-baixa



Fonte: Russ (2004)

3.3 Sintetizadores analógicos subtrativos

Entende-se por sintetizador analógico subtrativo um sintetizador que aplica a síntese subtrativa para geração de som e possui em sua construção interna circuitos eletrônicos analógicos. Devahari (1982) define como sendo uma série de módulos que geram sinais elétricos, modificados ou controlados por outros sinais e circuitos elétricos, para produzir um som desejado.

Apesar de terem havido outras tentativas (como já mencionado nesta dissertação) de construção de um sintetizador analógico, quem em geral recebe o crédito de ter sido um dos pioneiros a desenvolver um é Robert Moog. Moog auxiliou a criar e apresentou um conceito modular para o sintetizador analógico subtrativo – cada módulo do sintetizador é responsável por uma “tarefa”, e diferentes módulos são conectados em série ou paralelo para a criação completa do som. (JENKINS, 2007).

Os autores Crombie (1982) e Devahari (1982) apresentam uma classificação para os sintetizadores. Apresentamos algumas na Tabela 3-1.

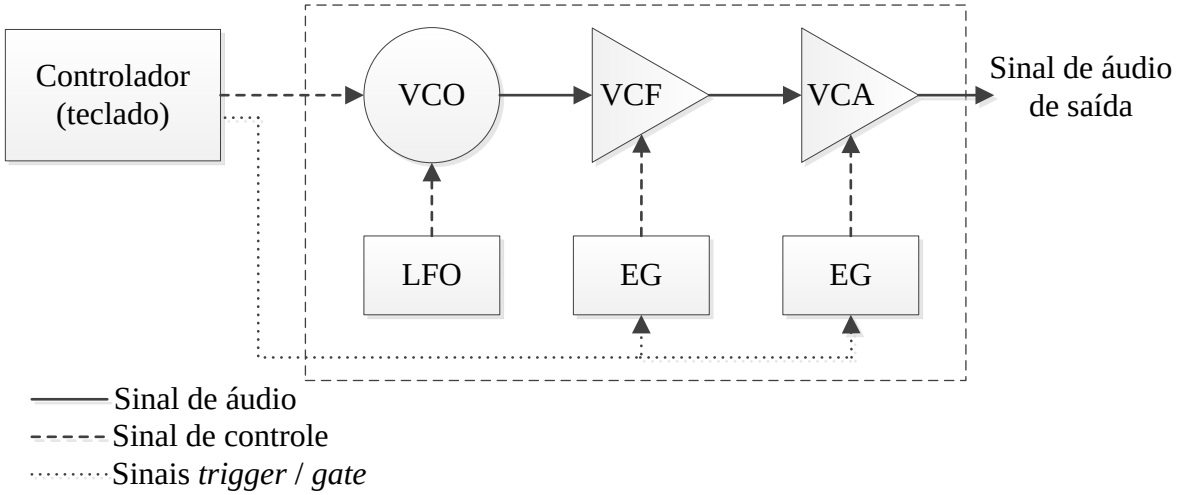
Tabela 3-1 – Tipos de sintetizadores

Analógicos	Construídos principalmente com módulos elétricos e eletrônicos, contendo apenas componentes analógicos na geração principal do som. Podem possuir componentes digitais para tarefas auxiliares, como varredura de teclado ou salvamento de informações.
Digitais	Construídos com módulos elétricos e eletrônicos, porém contendo processadores ou lógicas digitais para implementação de síntese sonora.
Monofônicos	Sintetizadores que são capazes de gerar apenas uma nota por vez.
Polifônicos	Sintetizadores capazes de gerar mais de uma nota por vez.
Hard-Wired	Este tipo de sintetizador é um subtipo da categoria Analógicos. São os sintetizadores que possuem os blocos de geração e controle de som com interligação já pré-definidas pelos fabricantes, podendo os usuários apenas ativarem ou desativarem algum destes caminhos. É o tipo mais comum de sintetizador analógico.
Quasi-modular	Também um subtipo da categoria Analógicos. Este grupo de sintetizadores possui alguns módulos com interligação pré-definida de fábrica, como os *Hard-Wired*, porém oferece entradas e saídas para módulos de geração e controle de som externos. Com o auxílio de cabos externos, é possível realizar algumas ligações entre os módulos conforme desejo do usuário.
Modular	Outro subtipo da categoria Analógicos. Sintetizadores modulares são projetados de forma que o usuário interliga os módulos de geração e controle de som conforme desejar.
Teclado/sintetizador	Um sintetizador com teclado é um sintetizador que possui um teclado, similar ao de um piano, que serve como controle para o gerador de som, e com o qual executa-se a(s) nota(s) desejada(s).
Sintetizador/módulo	Um sintetizador pode não possuir um teclado para execução das notas. Neste caso, outro dispositivo externo deve ser utilizado para executar as notas, como um módulo de teclado separado a ser conectado no sintetizador, ou um controlador MIDI.

Fonte: Devahari (1982) e Crombie (1982)

Os principais módulos, ou blocos, existentes nos sintetizadores analógicos subtrativos estão demonstrados na Figura 3-7. A função de cada módulo é representada por uma sigla em inglês. Utilizaremos estas siglas nesta dissertação devido à ampla utilização das mesmas ainda hoje, independentemente do idioma em uso.

Figura 3-7 – Blocos principais de um sintetizador analógico



Fonte: Crombie (1982, p. 16), adaptado pelo autor

3.3.1 Controlador ou teclado

Como dispositivo de controle temos neste caso um teclado, que é utilizado para controlar os blocos do sintetizador. Em um sintetizador analógico, a saída de controle do teclado é uma saída analógica com um valor de tensão. Esta tensão serve como tensão de controle do VCO, determinando assim a frequência de saída (e consequentemente a nota musical). Além da saída de controle, temos tipicamente dois sinais adicionais: *gate* e *trigger*. O sinal de *gate* funciona como um sinal de habilitação, e se mantém ativo enquanto uma tecla está pressionada no teclado. O sinal de *trigger* é um sinal de disparo, e é um pulso que é gerado sempre que uma nova tecla é pressionada. Estes dois sinais são utilizados pelos geradores de envelope (EG). As formas de uso destes sinais serão descritas mais adiante. (CROMBIE, 1982).

3.3.2 Oscilador controlado por tensão (VCO)

Crombie (1982) e Devahari (1982) apresentam o VCO subdividido em três blocos: um somador de tensão, um oscilador e geradores de formas de onda. O VCO recebe sinais das seguintes fontes:

- Controlador, indicando qual frequência deve ser gerada (nota musical);
- Controles auxiliares, como ajuste fino da frequência e tipo de forma de onda;
- Saída do oscilador de baixa frequência (LFO), que pode ser utilizado para modular a frequência principal.

Os sinais de tensão destas fontes são somados (com os devidos ajustes de ganho) com o objetivo de produzir um único sinal de tensão. Este sinal é então utilizado para alimentar o

oscilador propriamente dito, gerando a frequência de saída desejada. Por final, um circuito específico para cada forma de onda é utilizado, gerando o formato de onda desejado.

3.3.3 Filtro controlado por tensão (VCF)

O timbre característico que cada instrumento possui é dado, em grande parte, pela variação e remoção das frequências harmônicas ao longo do tempo. Crombie (1982) e Devahari (1982) afirmam que o filtro controlado por tensão, em um sintetizador analógico subtrativo, é o principal responsável pelo timbre final característico do instrumento. Os principais fabricantes de sintetizadores patentearam os filtros utilizados nos seus instrumentos na época de lançamento, para garantir uma sonoridade única (citamos como exemplo a patente USPTO (1969), de Robert Moog, e USPTO (1977), de Pearlman e Gillette). Isto porque a resposta em frequência do filtro, a forma como ele atenua ou acentua determinadas frequências pode variar conforme o projeto do mesmo, e isto causa uma sonoridade diferente ao resultado final, diferenciando cada instrumento.

Assim como no VCO, em geral o VCF possui mais de uma entrada de controle. Além dos controles de frequência de corte e frequência de ressonância, geralmente presentes no painel frontal, entradas auxiliares de tensão podem ser usadas para alterar dinamicamente estes parâmetros. Isto permite, por exemplo, utilizar um EG para modificar a frequência de corte do filtro ao longo do tempo, fazendo com que o timbre produzido pelo instrumento varie de acordo com a duração do som.

Uma característica importante do módulo VCF é o *keyboard tracking*. De acordo com Devahari (1982), a maior parte dos módulos VCF possuem esta característica com o intuito de manter o timbre característico independente da nota sendo executada. O *keyboard tracking* é uma entrada do VCF que é conectada ao teclado controlador – ou o mesmo componente conectado no VCO que determina a frequência do oscilador principal. O objetivo é, de forma proporcional, modificar a frequência de corte do filtro em função do aumento ou redução da frequência do oscilador principal. Podemos citar como um exemplo a execução da nota dó 2, que possui a frequência de 65 Hz. Se esta nota estiver sendo executada e a frequência de corte do filtro estiver ajustada em 500 Hz, a frequência fundamental da nota será naturalmente gerada. Caso a execução seja alterada para a nota dó 5, que possui a frequência fundamental de 523 Hz, a frequência fundamental estará sendo atenuada. Assim, com a característica de *keyboard tracking*, a frequência de corte do filtro neste caso estará ajustada para um valor acima de 523 Hz.

Alguns sintetizadores ou módulos permitem também selecionar o tipo de filtro, como passa-baixa (o mais comum), passa-alta, passa-faixa ou rejeita-faixa, e o nível de atenuação do mesmo, em geral 12 dB/oitava ou 24 dB/oitava.

3.3.4 Amplificador controlado por tensão (VCA)

O amplificador controlado por tensão, como o nome já diz, é um amplificador (ou atenuador) cujo valor do sinal de saída é dado pelo sinal de entrada multiplicado por um ganho, que é determinado pelo nível de tensão aplicado na sua entrada de controle. O bloco VCA geralmente possui um ganho máximo de 1, ou seja, o valor máximo de saída é igual ao sinal de entrada. Devido a isto, na maioria dos casos em que o VCA é utilizado, o ganho do amplificador é menor do que 1, e é utilizado em conjunto com um EG conectado na sua entrada de controle, para inserir uma dinâmica de intensidade nas notas musicais geradas. (CROMBIE, 1982).

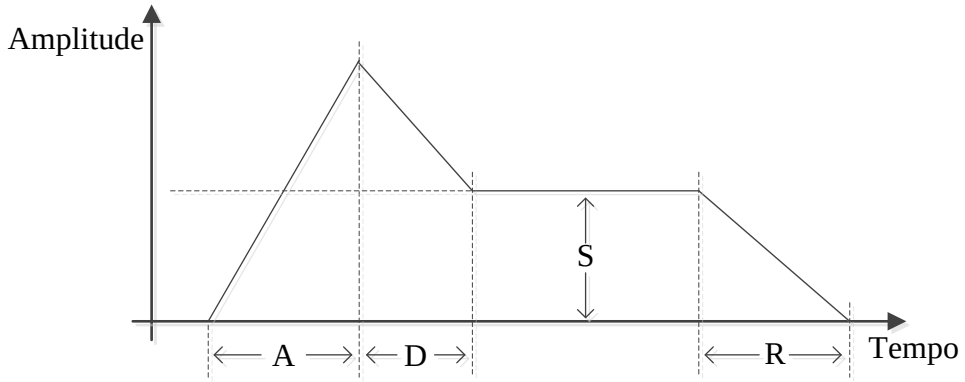
3.3.5 Oscilador de baixa frequência (LFO)

O oscilador de baixa frequência é um módulo que produz um sinal em uma frequência abaixo do espectro de áudio, até um pouco acima do início do espectro de áudio. De acordo com Crombie (1982), a maioria dos LFOs possuem frequência ajustável entre 0,2 Hz e 35 Hz. O LFO, ao contrário do VCO, não é regulado pelo mesmo controle (ou teclado) que indica a nota musical que deve ser executada. Em geral, a frequência do LFO é regulada através de um ajuste no painel do instrumento. A saída do LFO em geral possui as formas de onda senoidal, triangular e quadrada, e essa saída é conectada em outros módulos para servir como sinal modulante. Exemplo de alguns parâmetros que este módulo pode modular são a frequência do oscilador principal e a frequência de corte do VCF.

3.3.6 Gerador de envelope (EG)

O som gerado por diferentes instrumentos musicais possui uma importante característica que os diferencia, que é o formato da amplitude (intensidade) que a onda sonora possui ao longo do tempo, logo após que o som começa a ser gerado. Esse formato é chamado de contorno, ou envelope. O módulo gerador de envelope gera este contorno em seu sinal de saída, que por sua vez pode ser usado para modular algum outro módulo controlado por tensão, como por exemplo o amplificador controlado por tensão (VCA). O EG geralmente possui 4 estágios: ataque (*attack*), caimento (*decay*), sustentação (*sustain*) e liberação (*release*). A Figura 3-8 apresenta os estágios do gerador de envelope.

Figura 3-8 – O contorno do gerador de envelope



Fonte: Devahari (1982, p.81), adaptado pelo autor

Os quatro estágios apresentados na Figura 3-8 são descritos abaixo:

- Estágio A (*attack*): este estágio é o tempo em que o sinal leva para ir do nível mínimo até o nível máximo. O parâmetro de *attack* é um parâmetro de tempo;
- Estágio D (*decay*): este estágio, também um parâmetro de tempo, é o tempo que o sinal leva para cair do nível máximo até o nível definido pelo parâmetro de *sustain*;
- Estágio S (*sustain*): o estágio de *sustain* é um parâmetro de nível, e não de tempo como os anteriores. O nível de *sustain* refere-se ao nível de amplitude que o gerador de envelope manterá no seu sinal de saída, após os tempos de *attack* e *decay*, e enquanto o gerador estiver recebendo o sinal de *gate* (que indica que alguma nota musical está sendo executada ou pressionada);
- Estágio R (*release*): este estágio é um parâmetro de tempo, e refere-se ao tempo em que o gerador de envelope levará para levar o sinal do nível de *sustain* até o nível mínimo, quando não estiver mais recebendo o sinal de *gate*. Mais detalhes sobre o sinal de *gate* serão descritos a seguir.

De acordo com Crombie (1982), existem variações do módulo EG que suprimem alguns estágios, como o módulo AR (*attack and release*), que produz um nível de tensão que cresce até o máximo no tempo de *attack*, ficando no nível máximo até a nota ser liberada, caindo então até o nível zero no tempo de *release*; o módulo AD (*attack and decay*), que produz um nível de tensão crescente e ao atingir o máximo, retorna ao ponto inicial no tempo de *decay*; e o ADS (*attack decay sustain*), que suprime o parâmetro de tempo de *release*, mas utiliza o valor configurado como *decay* em substituição ao tempo de *release*.

3.3.6.1 Sinal de gate e sinal de trigger

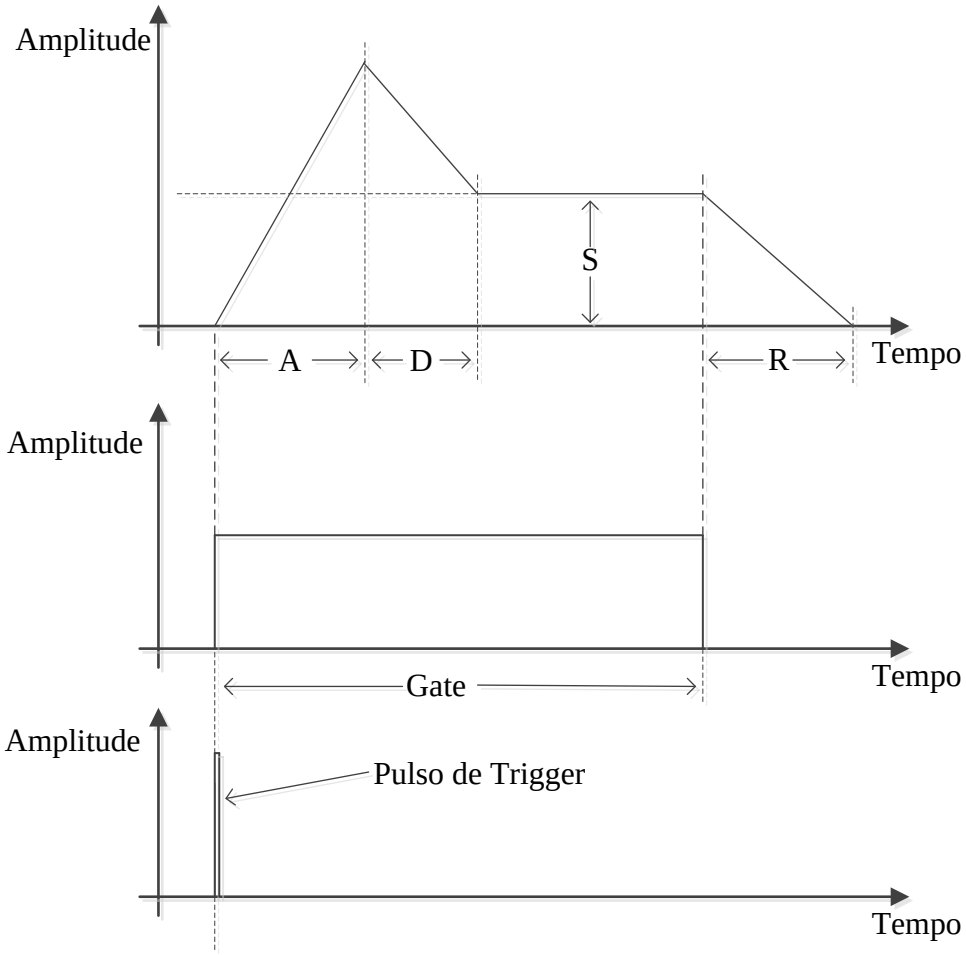
Os sinais de *gate* e *trigger* são dois sinais independentes que servem como entrada para o módulo gerador de envelope. O sinal de *gate* é como um sinal liga-desliga. Quando o sinal de *gate* muda para nível lógico 1, o gerador de envelope inicia a geração do sinal na saída, começando pelo estágio de *attack*. Quando o sinal de *gate* volta para nível lógico 0, o gerador de envelope muda para o estágio *release*, independente de qual estágio estiver. É importante ressaltar que o sinal de *gate* fica sempre em nível lógico 1 enquanto houver alguma nota musical em execução. Ou seja, se ao executar uma música, o músico executa uma nota antes de encerrar a anterior, o sinal de *gate* se mantém em 1, sem voltar para o estado 0.

Já o sinal de *trigger* é um pulso, de curta duração, que acontece a cada nova nota musical executada, independentemente de haver outra em execução. Este sinal pode ser usado para reiniciar o ciclo do gerador de envelope a cada nova nota executada, por exemplo.

A existência destes dois sinais permite que diferentes características sejam utilizadas no módulo gerador de envelope. Devahari (1982) comenta que os geradores de envelope da Moog não possuem o sinal de *trigger*, apenas o de *gate*. Logo, quando um instrumentista toca um teclado em um sintetizador Moog, se as notas forem executadas ligadas umas às outras, sem interrupção, o gerador de envelope receberá apenas um sinal *gate*, fazendo com que o gerador de envelope passe pelos estágios de *attack* e *decay*, e se mantendo no estágio de *sustain* até que o instrumentista pare de tocar completamente, ou insira um espaço entre duas notas, de forma que o sinal de *gate* volte a nível lógico 0. Por outro lado, instrumentos de marcas como ARP e Oberheim possuem o sinal de *trigger* a cada nova nota pressionada, fazendo com que o gerador de envelope volte ao estágio de *attack* a cada nova nota. Os instrumentistas, neste caso, precisam aprender estilos diferentes de execução e criação musical para cada uma destas marcas devido a estas diferenças.

A Figura 3-9 mostra os estágios do gerador de envelope em conjunto com os sinais de *gate* e *trigger*.

Figura 3-9 – Sinal de *gate* e pulso de *trigger*



Fonte: Devahari (1982, p.81), adaptado pelo autor

3.4 Órgão Hammond

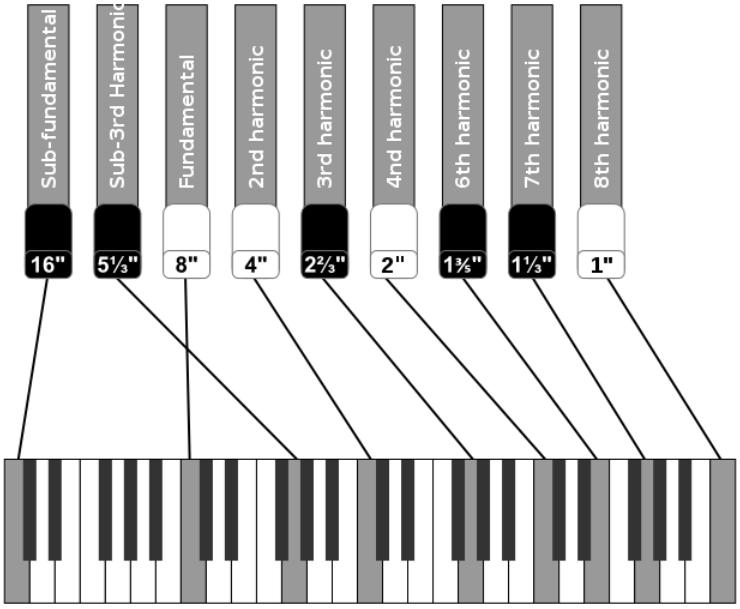
O órgão Hammond, criado por Laures Hammond, foi concebido na década de 30 para ser um substituto móvel e de baixo custo para órgãos de tubos em igrejas, e a partir das décadas de 60 e 70 se tornou também um instrumento muito utilizado na música popular, em especial rock e blues, devido ao seu timbre característico. (PEKONEN; PIHLAJAMAKI, VALIMAKI, 2011).

A síntese utilizada era síntese aditiva, da mesma forma que um órgão de tubos. A geração de formas de onda era realizada através de discos metálicos (*tone wheels*) contendo uma superfície ondular posicionados próximo a um sensor magnético. Com a rotação do disco metálico, o sinal captado pelo sensor magnético variava em amplitude de forma similar a uma onda senoidal, e a frequência deste sinal era dado pela rotação do disco magnético. Ao total, um órgão Hammond possuía 91 discos magnéticos, gerando assim o equivalente a 91

osciladores em 91 frequências diferentes. (PEKONEN; PIHLAJAMAKI, VALIMAKI, 2011) (JOHANN, 2015).

Para cada nota pressionada no teclado, nove desses osciladores eram selecionados e o som dos mesmos enviados à saída. Os osciladores selecionados equivaliam à frequência fundamental da nota, uma oitava abaixo (-12 semitons), o intervalo musical de quinta (+7 semitons), uma oitava acima (+12 semitons), uma oitava e meia acima (+19 semitons), duas oitavas acima (+24 semitons), duas oitavas e uma terça acima (+28 semitons), duas oitavas e uma quinta (+31 semitons) e, por fim, três oitavas acima (+36 semitons). A amplitude com que cada um destes osciladores era adicionado ao sinal de saída era configurado com chaves seletoras chamadas de *drawbars.* (PEKONEN; PIHLAJAMAKI, VALIMAKI, 2011). A Figura 3-10 mostra a relação de cada *drawbar* com a nota fundamental no teclado.

Figura 3-10 -Diagrama de drawbars do Hammond



Fonte: Hammond-drawbars.svg (2007)

O órgão Hammond ainda possuía um módulo de efeito de tremolo, que é uma modulação na amplitude do sinal de saída. Outro sistema de efeitos que se tornou muito popular foi o efeito causado pelo uso das caixas de som Leslie (desenvolvida e vendida em separado por outro inventor, Donald Leslie). Esta caixa de som possuía duas unidades internas (uma para reprodução de graves e outra para médios e agudos) que giravam em rotações diferentes e selecionáveis, causando uma sensação de variação de amplitude e frequência no som. (PEKONEN; PIHLAJAMAKI, VALIMAKI, 2011).

4 PLATAFORMAS DE DESENVOLVIMENTO

Descrevemos neste capítulo as características das principais plataformas de desenvolvimento às quais temos conhecimento e fácil acesso. As plataformas Arduino (ARDUINO, sem data), Teensy (TEENSY, sem data), Raspberry Pi (RASPBERRY, sem data) e Beagle Board (BEAGLEBOARD.ORG, sem data) estão dentre as plataformas mais utilizadas para projetos DIY envolvendo eletrônica e computação, bem como em escolas e universidades destas áreas (Jamieson e Herdtner, 2015).

4.1 Arduino

Arduino é uma plataforma aberta de desenvolvimento e prototipação, com um *hardware* e *software* de fácil uso, focado em engenheiros, hobistas, artistas e qualquer pessoa interessada em criar objetos ou ambientes interativos (“Arduino”, 2016). Existem diversos modelos de placas de desenvolvimento Arduino, contendo processadores de 8 bits até 32 bits, com diversos periféricos e placas auxiliares que podem ser compradas para agregar funcionalidades. O ambiente de desenvolvimento é escrito em linguagem Java e possui uma interface simplificada para o desenvolvimento do código (em C ou C++) a ser executado no Arduino. As placas também incluem um gravador USB, permitindo programar o processador utilizando um cabo USB conectado no computador.

O uso do Arduino em geral não utiliza sistema operacional para executar tarefas como escalonador, gerenciador de memória, semáforos e outros recursos típicos de sistemas operacionais, sendo a maioria dos projetos escritos diretamente em linguagem C ou C++ e utilizando apenas os recursos nativos da linguagem, além de algumas funções adicionais fornecidas pelo ambiente do Arduino para auxiliar na abstração de acesso a periféricos do processador ou da placa.

Há uma grande gama de projetos com esta plataforma disponíveis na internet, incluindo implementações envolvendo áudio, dispositivos MIDI e processamento de sinais (como módulos de efeito). Apesar disso, pelo fato de utilizar processadores simples e sem recursos avançados de processamento matemático, muitas vezes estas implementações possuem resolução ou taxas de amostragem limitadas.

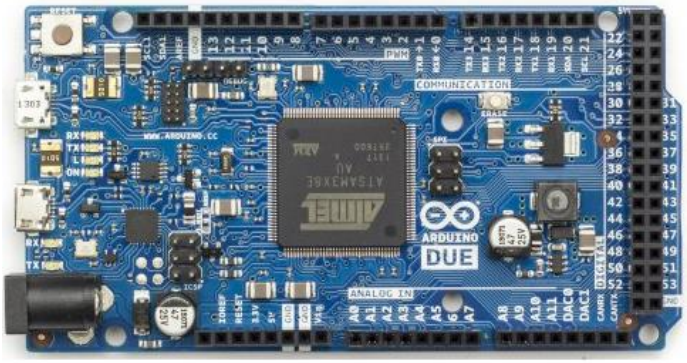
4.1.1 Arduino Due

Dentre os modelos de Arduino disponíveis, o Arduino Due está entre os que possuem o processador mais rápido. Seu processador é o Atmel SAM3X8E, um processador com *core*

ARM Cortex-M3, *clock* de 84 MHz, 512kB de memória flash para programa e 96 kB de memória RAM. A placa possui 54 pinos digitais de entrada e saída, 12 entradas com conversor analógico de 12 bits, 2 saídas com conversor analógico de 12 bits, 4 interfaces seriais, além de outras interfaces incluindo uma interface SSC (*synchronous serial controller*), que pode ser utilizada para comunicação com conversores DA (digital-analógico) externos de alta qualidade. Apesar de não possuir unidade aritmética de ponto flutuante em *hardware*, o que torna os cálculos de pontos flutuantes lentos pois são realizados através de bibliotecas de *software*, a arquitetura 32 bits e a velocidade do *clock* tornam esta plataforma atrativa para implementação de cálculos matemáticos utilizando ponto fixo e o uso para aplicações de síntese musical. Este trabalho explora e demonstra esta capacidade.

A Figura 4-1 mostra a placa de desenvolvimento Arduino Due.

Figura 4-1 – Placa Arduino Due



Fonte: Arduinodue.jpg (2016)

A Tabela 4-1 apresenta um comparativo da capacidade de executar operações com operandos ponto flutuante entre o Arduino Due, o Raspberry Pi e outros processadores utilizados em computadores pessoais e *smartphones*. Destacamos a diferença entre os valores do Arduino Due e do restante, demonstrando a necessidade de não utilizar operandos de ponto flutuante.

Tabela 4-1 – Comparativo de capacidade de operações com ponto flutuante por segundo

Plataforma / Processador	Operações	Fonte
Intel Core i7	Aprox. 3,5 GFLOPs/core	http://www.roylongbottom.org.uk/linpack%20results.htm
Qualcomm Snapdragon 810	Aprox. 1,45 GFLOPs/core	http://www.roylongbottom.org.uk/linpack%20results.htm
Raspberry Pi 3	Aprox. 180 MFLOPs/core	http://www.roylongbottom.org.uk/linpack%20results.htm
Arduino Due	1,11 MFLOPs	https://forum.arduino.cc/index.php?topic=431169.0

Fonte: elaborada pelo autor

4.2 Teensy

A plataforma Teensy possui configurações e aplicações similares à plataforma Arduino. Possui modelos com processadores de 8 bits até 32 bits, porta de programação USB e diversos periféricos como entrada analógica, saída analógica, portas seriais, etc. Também é programada utilizando linguagem C e C++ e, em geral, sua utilização não envolve sistemas operacionais mais complexos como Linux embarcado. Atualmente é uma das principais alternativas à plataforma Arduino para aplicações mais simples.

4.3 Raspberry Pi

Raspberry Pi é um computador de baixo custo, do tamanho de um cartão de crédito que pode ser conectado em uma TV e funciona com teclado e *mouse* padrão de um computador comum. É um dispositivo pequeno que busca facilitar para que pessoas de qualquer idade possam aprender computação e programação (“Raspberry Pi”, 2016). A versão mais recente do Raspberry Pi (Raspberry Pi 3) possui um processador ARMv8 64 bits com quatro núcleos de processamento e 1,2 GHz. Possui ainda 1 GB de memória RAM, quatro portas USB, 40 pinos digitais de entrada e saída, porta HDMI, dentre outros periféricos.

Como mencionado anteriormente, o objetivo de uso desta placa é a programação em um ambiente Linux, possuindo uma vasta comunidade de suporte para desenvolvimento. Apesar de ser possível desenvolver aplicações em C e C++ assim como no Arduino, o uso desta plataforma tende a exigir alguns conhecimentos extras pois o *software* é executado no ambiente do sistema operacional Linux.

4.4 Beagle Board

A placa Beagle Board também possui um desenvolvimento voltado para aplicações em ambiente Linux. Um de seus modelos mais populares, a Beagle Bone Black, possui um processador ARM Cortex-A8 de 1 Ghz, com 512 MB RAM. Possui também pinos digitais de entrada e saída, interfaces seriais e diversas outras. Em relação ao ambiente de desenvolvimento, é muito similar à Raspberry Pi, objetivando o desenvolvimento de aplicações para ambiente Linux.

4.5 A opção pelo Arduino Due

Em comparação com a plataforma Teensy, a escolha pelo Arduino se deu devido à sua popularidade, facilidade de compra e grande quantidade de usuários e projetos realizados com o mesmo, que é superior ao Teensy. Já em comparação com a plataforma Raspberry Pi e Beagle Board, o Arduino se mostra mais atrativo para aplicações mais baixo nível por não necessitar o uso de um sistema operacional como o Linux e por possuir mais periféricos de entrada e saída propícios para a implementação de um sintetizador, como uma grande quantidade de pinos de entrada e saída digitais, analógicos, interfaces para uso de conversor DA (interface SSC) e MIDI (interface UART). Plataformas como Raspberry Pi e Beagle Board necessitariam placas externas adicionais para obter algumas destas capacidades, elevando o custo, tamanho e complexidade do desenvolvimento. Além disto, considerando o poder de processamento de placas como Raspberry Pi e Beagle Board (que possuem processadores com velocidade de *clock* próxima à 1 GHz e unidade de ponto flutuante em *hardware*), é possível implementar sintetizadores com estas placas utilizando outros recursos e linguagens, como Linux e CSOUND, não sendo necessário o uso apenas de linguagem C ou C++. (BATCHELOR; WIGNALL, 2013).

5 PROCESSAMENTO DIGITAL DE ÁUDIO

Os sinais de áudio que se propagam através de um meio físico e são captados pelo ouvido humano são sinais de grandeza analógica. Para que um sinal de áudio seja processado por um microprocessador, microcomputador ou outro dispositivo digital, este sinal deve ser convertido de um sinal no tempo contínuo para um sinal no tempo discreto. Um sinal analógico é um sinal no tempo contínuo, que varia continuamente e é representado por uma quantidade física que também varia continuamente. Quando um sinal analógico que varia no tempo é convertido para digital, temos uma sequência de números que representam o sinal analógico em determinados momentos do tempo. Normalmente, esta sequência de números é chamada de sinal digital ou sinal no tempo discreto. (DINIZ; SILVA; NETO, 2004).

5.1 Sinal no tempo discreto

Os autores Diniz, Silva e Neto (2004) definem um sinal no tempo discreto como sendo aquele capaz de ser representado por uma sequência de números $x[n]$, que pode ser obtida através de uma amostragem periódica de um sinal no tempo contínuo, conforme a equação 2:

$$x[n] = x_a(nT) \quad (2)$$

onde x_a é o valor do sinal no tempo contínuo, T é o período de amostragem e $x[n]$ é a *n*-ésima amostra do valor convertido para o tempo discreto. Uma vez feita esta conversão e o sinal tendo sido processado, é necessário então converter o resultado no tempo discreto de volta ao domínio do tempo contínuo. Portanto, para que essa operação seja efetiva, é necessário que tenhamos a capacidade de restaurar um sinal no tempo contínuo a partir de suas amostras no tempo discreto.

5.1.1 Teorema da amostragem

Definimos um sinal $p(t)$ que consiste em um trem de impulsos conforme a equação 3:

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \quad (3)$$

O sinal no tempo contínuo $x_a(T)$ é então multiplicado pelo sinal $p(t)$, obtendo assim o sinal $x_i(t)$ no tempo contínuo definido pela equação 4:

$$x_i(t) = \sum_{n=-\infty}^{\infty} x_a(nT)\delta(t - nT) \quad (4)$$

A equação 4 apresenta o sinal amostrado no tempo contínuo, que é então convertido para o tempo discreto obtendo-se uma sequência $x[n]$ com valores separados entre si pelo

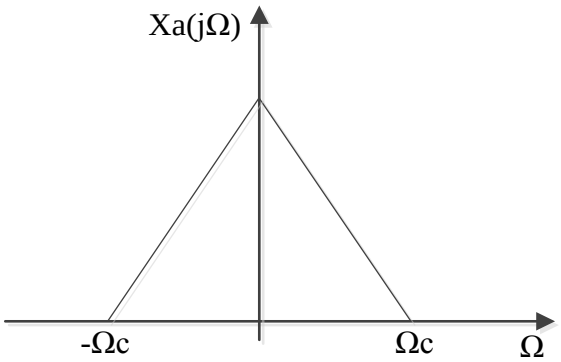
intervalo de amostragem T. Diniz, Silva e Neto (2004) aplicam algumas operações matemáticas nesta equação, como obter a transformada de Fourier do sinal de trem de impulsos e converter o resultado para o domínio da frequência. Após estas operações, a equação resultante que define o sinal amostrado no tempo discreto é definida pela equação 5:

$$X_i(j\Omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_a\left(j\Omega - j\frac{2\pi}{T}k\right) \quad (5)$$

onde $X_i(j\Omega)$ é o sinal amostrado no domínio da frequência, X_a é o sinal a ser amostrado e Ω é a frequência, medida em radianos por segundo (rad/s).

A equação 5 mostra que o espectro do sinal amostrado é composto de infinitas cópias deslocadas do espectro do sinal X_a, sendo os deslocamentos na frequência múltiplos da frequência de amostragem dada por $\Omega s = \frac{2\pi}{T}$. Assumindo que o sinal X_a é limitado em frequência, a Figura 5-1 apresenta o espectro de frequência do mesmo, sendo Ωc a largura de banda.

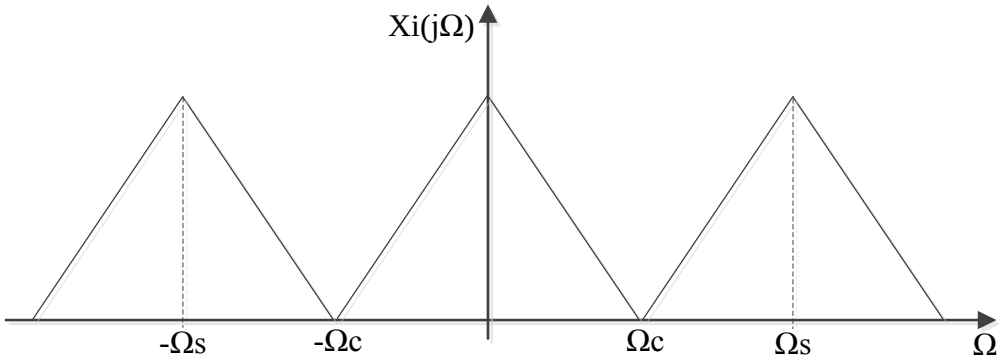
Figura 5-1 – Espectro de frequência do sinal no tempo contínuo



Fonte: Diniz, Silva e Netto (2004, p.43), adaptado pelo autor

A Figura 5-2 mostra o sinal amostrado Xi (jΩ) e seu espectro de frequência. Lembrando que o espectro deste sinal é composto de infinitas cópias deslocadas múltiplas da frequência de amostragem Ωs, vemos que a largura de banda do sinal amostrado (Ωc) deve ser limitado, para evitar que as cópias deslocadas em frequência interfiram umas nas outras.

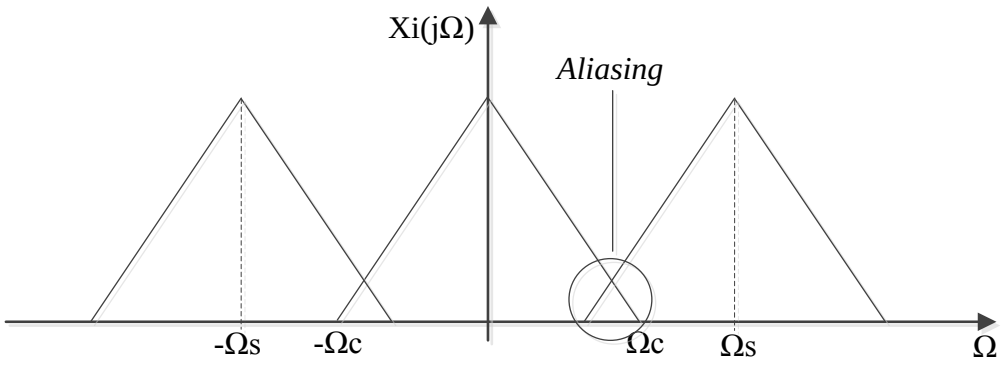
Figura 5-2 – Espectro de frequência do sinal amostrado Xi(jΩ)



Fonte: Diniz, Silva e Netto (2004, p.43), adaptado pelo autor

A largura de banda Ωc deve ser tal que a extremidade superior do espectro centrado em zero se situe abaixo da extremidade inferior do espectro centrado em Ωs, isto é, $\Omega c < \Omega s - \Omega c$, implicando então que $\Omega s > 2\Omega c$, ou seja, que a frequência de amostragem deve ser maior que o dobro da largura de faixa unilateral do sinal no tempo contínuo. A frequência Ω = 2Ωc é chamada de frequência de Nyquist do sinal no tempo contínuo. Se esta condição não for satisfeita, as repetições do espectro interferem uma com a outra, e o sinal no tempo contínuo não pode ser recuperado a partir de suas amostras. A sobreposição destas frequências no espectro, e que corrompe a informação de frequências do sinal, é chamada de *aliasing* (sem termo equivalente em português de acordo com Diniz, Silva e Neto (2004)). A Figura 5-3 demonstra este efeito no espectro de frequência.

Figura 5-3 – Efeito de *aliasing*

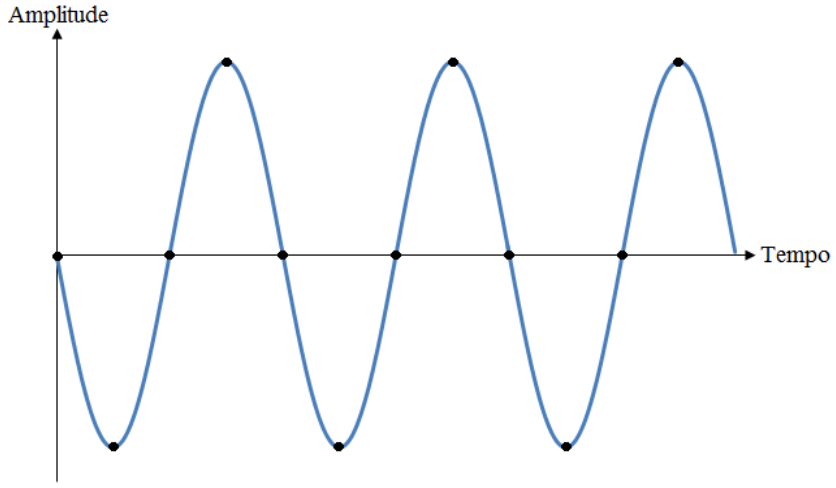


Fonte: Diniz, Silva e Netto (2004, p.43), adaptado pelo autor

Baseados nestas análises, Diniz, Silva e Neto (2004) apresentam o teorema da amostragem: se um sinal periódico no tempo contínuo tem largura de faixa limitada, então o mesmo pode ser completamente recuperado a partir do sinal no tempo discreto se e somente se a frequência de amostragem Ωs satisfaz a condição Ωs > 2Ωc.

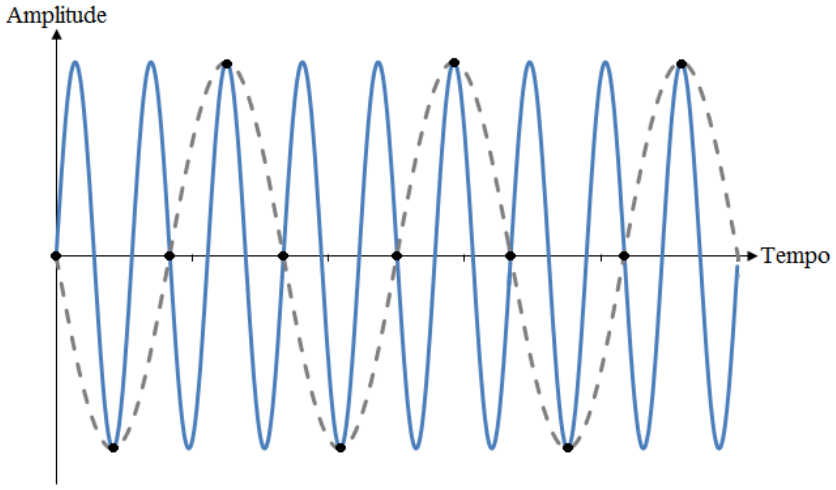
Os autores Dodge e Jerse (1997) apresentam o teorema da amostragem afirmando que a frequência de amostragem ou conversão de um sinal do tempo contínuo para o tempo discreto deve ser o dobro da maior frequência existente no sinal analógico do tempo contínuo. Para melhor entender o efeito causado pela amostragem de um sinal sem respeitar esta definição, no tempo contínuo, Dodge e Jerse (1997) apresentam dois exemplos de conversão de sinal conforme apresentado na Figura 5-4 e Figura 5-5.

Figura 5-4 – Amostragem de um sinal de 10kHz à uma taxa de amostragem de 40kHz



Fonte: Dodge e Jerse (1997, p.64), adaptado pelo autor

Figura 5-5 – Amostragem de um sinal de 30kHz à uma taxa de amostragem de 40kHz



Fonte: Dodge e Jerse (1997, p.65), adaptado pelo autor

A Figura 5-4 apresenta a conversão de um sinal de 10kHz com uma frequência de amostragem de 40kHz. Baseado no teorema da amostragem, a frequência de 40kHz permite a amostragem de um sinal de até 20kHz. Já a Figura 5-5 apresenta a amostragem de um sinal de frequência de 30kHz, acima da frequência máxima de acordo com o teorema da amostragem. Observando as duas imagens, observa-se que os pontos amostrados e convertidos para o tempo

discreto são exatamente os mesmos para os dois sinais. Isto faz com que seja impossível distinguir o sinal de 30kHz do sinal de 10kHz, uma vez tendo sido amostrado e convertido para o tempo discreto. Este é o efeito de *aliasing*, pois a amostragem da frequência de 30kHz gerou um *alias* (termo em inglês que significa pseudônimo ou nome falso) de 10kHz, e quando os pontos amostrados forem convertidos novamente para o tempo contínuo, o sinal gerado será um sinal de 10kHz e não 30kHz (o sinal de 30kHz gerou um *alias* ou *foldover* em 10kHz). (DODGE; JERSE, 1997).

5.1.2 Quantização

O autor Coulter (2000) define uma grandeza contínua como uma grandeza que pode assumir qualquer valor, dentro de uma faixa definida ou não, enquanto que uma grandeza discreta somente pode assumir determinados valores dentro de uma faixa definida. Em se tratando da amplitude de um sinal de áudio, significa que em vez de possuir uma resolução infinita dentro de uma determinada faixa, o sinal deve assumir valores conhecidos dentro de uma faixa e, ao ser convertido de grandeza contínua para grandeza discreta, o sinal contínuo geralmente assume o valor discreto mais próximo.

Esta conversão para o nível mais próximo gera um erro conhecido como ruído de quantização, que mede a relação entre o nível do sinal e o erro causado pela conversão. Um conversor de 16 bits possui uma faixa de conversão equivalente a 96 dB (para efeito de comparação, um concerto de uma orquestra ao vivo possui uma faixa aproximada que varia desde o “silêncio” até um alcance de 110 dB com a orquestra completa executando uma música). (ROADS, 1996).

5.2 Osciladores

Conforme apresentado nas seções 3.2 e 3.3.2, um sintetizador subtrativo possui um ou mais osciladores que são responsáveis tanto pela geração do áudio na frequência desejada (em função da nota musical) como pela modulação de alguns parâmetros de outros módulos (oscilador de baixa frequência).

Para o desenvolvimento de osciladores utilizando circuitos digitais ou microprocessadores, como no caso deste trabalho, alguns cuidados devem ser tomados para evitar a geração de *aliasing*, pois algumas das formas de onda possuem componentes no espectro de frequência que ultrapassam a largura de banda da geração do áudio, não satisfazendo assim a condição $\Omega s > 2\Omega c$. Supondo que a geração dos sinais de saída seja realizada a uma taxa de 48 kHz, de acordo com a frequência de Nyquist a maior componente

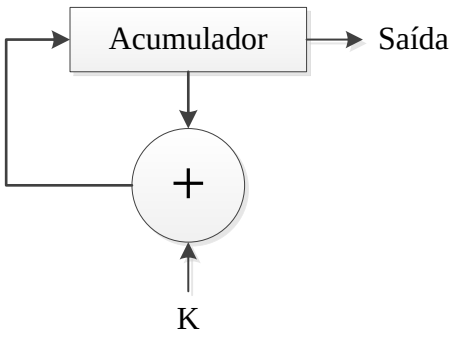
de frequência possível de ser gerada seria 24 kHz, porém ao gerar um sinal de onda quadrada, componentes de frequência maiores que este são necessários e estão implícitos em uma quadrada perfeita (a ser detalhado no item 5.2.3).

No decorrer deste capítulo analisaremos as opções existentes para reduzir o efeito de *aliasing* na geração das formas de onda nos osciladores.

5.2.1 Oscilador controlado numericamente

Um oscilador controlado numericamente (NCO – *numerically controlled oscillator*) é um algoritmo para geração de um oscilador digital onde somas sucessivas de um valor pré-definido são realizadas, o valor resultante acumulado é salvo em um acumulador, e a saída deste acumulador é utilizada para geração do sinal desejado. Este acumulador é chamado de acumulador de fase, que consiste em um contador binário, geralmente sem sinal, incrementado ciclicamente somando-se um valor K. Quando o acumulador chega ao seu limite, acontece um estouro e a contagem continua. O valor residual da soma, quando acontece o estouro, continua no acumulador para a próxima contagem, evitando assim erros de arredondamento. A Figura 5-6 mostra o diagrama do acumulador de fase. (KISENWETHER; TROXELL, 1986).

Figura 5-6 – Acumulador de fase



Fonte: Kisenwether e Troxell (1986), adaptado pelo autor

A saída do acumulador de fase pode ser utilizada de várias formas. Uma delas é utilizar o valor do acumulador como índice para uma acessar tabela de valores previamente preenchida (*lookup table*). A tabela pode conter, por exemplo, valores para uma forma de onda senoidal. Outra forma é utilizar alguns bits mais significativos (MSB – *most significant bits*) para gerar uma onda quadrada, ou então utilizar o próprio valor do acumulador para gerar uma onda rampa ou dente-de-serra (ressaltamos que no caso da onda quadrada ou dente-de-serra o sinal não é limitado em banda e em consequência teremos a geração de *aliasing*). Os autores Pirkle (2014) e Valimake e Huovilainen (2007) também chamam estes dois últimos métodos de Oscilador Trivial (*Trivial Oscillator*).

A frequência de operação do NCO é dada pelo valor da constante K e da frequência com a qual o acumulador é incrementado, conforme mostrado na equação 6.

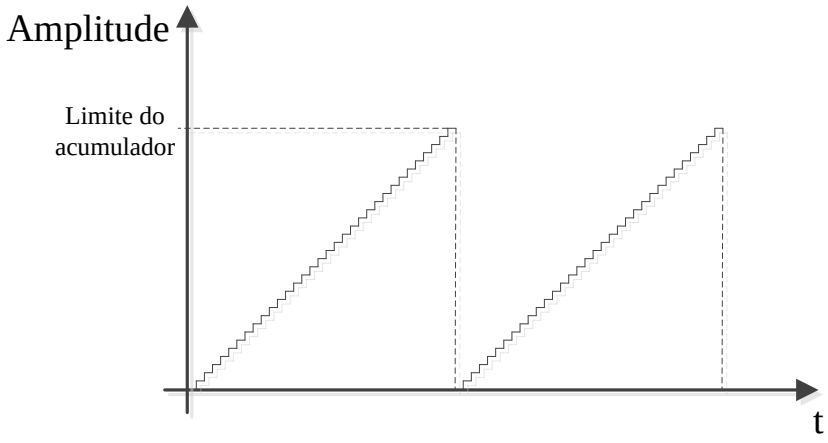
$$f_{NCO} = \frac{K \cdot f_{REF}}{2^N} \quad (6)$$

onde f_{REF} é a frequência na qual o acumulador é incrementado (frequência ou taxa de amostragem) e N é o número de bits do acumulador.

5.2.1.1 Oscilador dente-de-serra

Para a geração de uma onda dente de serra, o próprio valor do acumulador já possui o formato de uma onda dente de serra. A Figura 5-7 apresenta um exemplo de forma de onda de saída utilizando o valor do acumulador. Este sinal não é limitado em banda e ocorrerá o efeito de *aliasing* na saída.

Figura 5-7 – Geração trivial de uma onda dente de serra

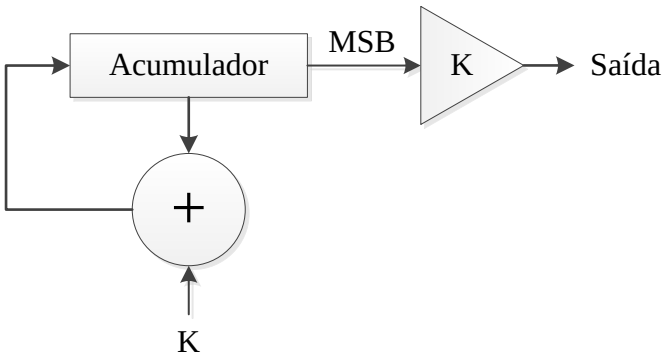


Fonte: Pirkle (2014), adaptador pelo autor

5.2.1.2 Oscilador onda quadrada

O oscilador de onda quadrada pode ser obtido utilizando apenas o bit mais significativo (MSB) do acumulador, multiplicado por um ganho, para gerar o sinal desejado na saída.

Figura 5-8 – Oscilador onda quadrada



Fonte: elaborada pelo autor

A Figura 5-8 mostra um diagrama da implementação de um oscilador de onda quadrada utilizando um NCO. A constante K é o valor a ser multiplicado pelo bit mais significativo para gerar o valor desejado na saída. (PIRKLE, 2014). Assim como o oscilador dente-de-serra apresentado em 5.2.1.1, este sinal não é limitado em banda e ocorrerá o efeito de *aliasing* na saída.

5.2.2 Oscilador por tabela de onda

Um oscilador por tabela de onda (*wavetable oscillator*) é um sistema baseado em uma tabela de valores (*lookup table*) para a criação de sinais periódicos. Uma tabela de valores é pré-carregada com a quantidade de valores equivalente a um período de uma forma de onda. A ideia é que os valores desta tabela sejam lidos sequencialmente, e o valor enviado para a saída. Neste caso, o valor do acumulador do NCO é o valor utilizado como referência para indexar a tabela e acessá-la em determinado momento de tempo.

A equação 7 apresenta como calcular o valor da constante K em função da frequência de saída desejada e da frequência de amostragem.

$$K = \frac{2\pi f_{REF}}{f_{SR}} \quad (7)$$

onde f_{REF} é a frequência de saída desejada e f_{SR} é a frequência de amostragem (frequência na qual a constante K será incrementada no acumulador). A equação 6 apresenta a constante 2π, que significa neste caso um período completo do sinal. Em um oscilador por tabela de onda, a constante 2π equivale ao tamanho da tabela a ser varrida, uma vez que a tabela possui os valores equivalentes a um período completo da forma de onda. Substituímos então na equação 8 a constante 2π pelo tamanho de amostras de valores existentes na tabela.

$$K = \frac{N f_{REF}}{f_{SR}} \quad (8)$$

onde N é o número de amostras da tabela.

5.2.3 *Aliasing*

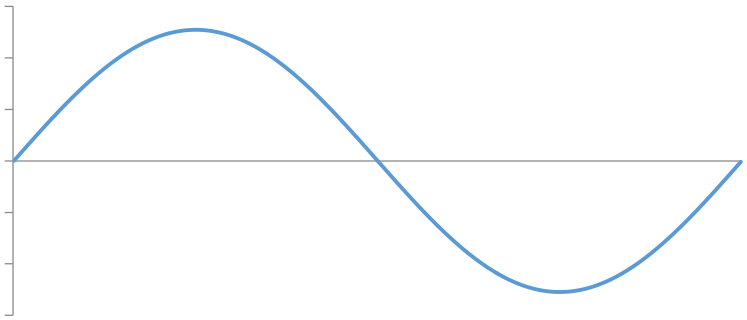
De acordo com Stilson e Smith (1996a), todo sinal analógico que possua uma descontinuidade em sua forma de onda, como uma onda dente-de-serra ou onda quadrada, precisa ter sua largura de banda limitada para menos da metade da taxa de amostragem para aquisição ou geração deste sinal no tempo discreto (frequência de Nyquist). Métodos simples de geração destes sinais, como o método trivial ou por tabela de onda (assumindo uma tabela de onda simples, contendo descontinuidades), resultarão em *aliasing* no sinal de saída.

Utilizaremos a onda quadrada como exemplo para demonstrar a diferença entre a onda quadrada perfeita (e sua descontinuidade) e uma onda quadrada com largura de banda limitada. A equação 9 é a equação utilizada para gerar uma onda quadrada.

$$y(t) = \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{sen(2\pi(2k-1)ft)}{2k-1} \qquad (9)$$

Onde *k* é o número de harmônicas, *f* é a frequência e *t* o tempo. Analisando a equação podemos observar que a onda quadrada é gerada através do somatório de componentes harmônicas ímpares (devido ao termo $(2k-1)$). Também analisando a equação, observamos que com apenas uma harmônica, teríamos uma onda puramente senoidal. Assim, quanto mais harmônicas adicionarmos, mais "perfeita" será a onda quadrada (lembrando que a onda quadrada possui uma descontinuidade na qual o sinal muda do seu ponto máximo para o ponto mínimo instantaneamente). Para demonstrar o efeito da adição de harmônicas na composição da onda quadrada, a Figura 5-9 apresenta o sinal resultante da equação 8 com apenas a frequência fundamental (*k*=1).

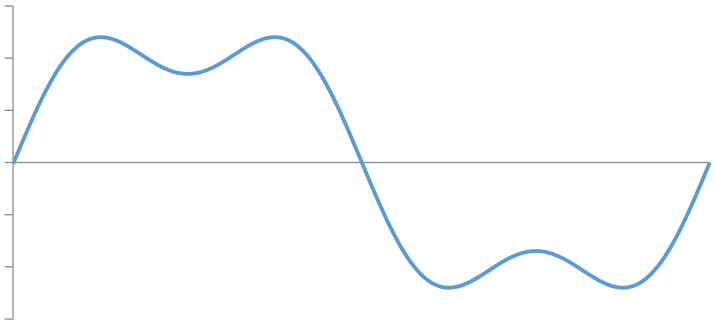
Figura 5-9 – Equação da onda quadrada com apenas a frequência fundamental



Fonte: elaborada pelo autor

Como esperado, sendo a onda quadrada a soma de diversas componentes senoidais, utilizando apenas a componente fundamental, teremos uma onda senoidal pura. Ao adicionarmos mais harmônicas (ímpares), a onda começa a tomar a forma de uma onda quadrada. A Figura 5-10 mostra o sinal resultante utilizando-se duas harmônicas.

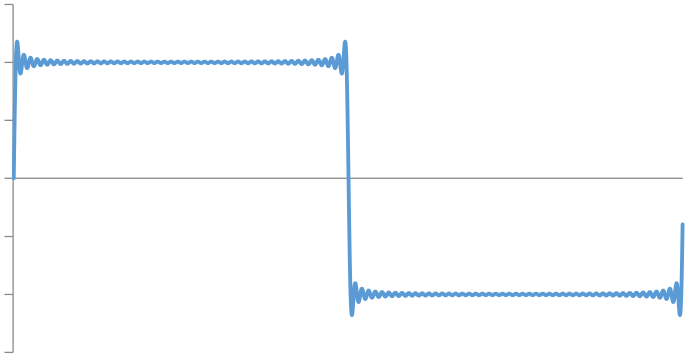
Figura 5-10 – Onda quadrada composta por duas harmônicas



Fonte: elaborada pelo autor

Adicionando mais harmônicas, a Figura 5-11 mostra o sinal resultante utilizando-se 50 harmônicas.

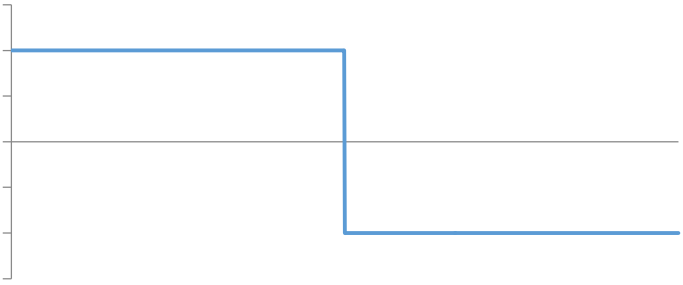
Figura 5-11 – Onda quadrada composta por cinquenta harmônicas



Fonte: elaborada pelo autor

A Figura 5-12 mostra como seria a onda quadrada ideal, a qual teoricamente é gerada utilizando infinitas componentes harmônicas, e não possui limitação de largura de banda.

Figura 5-12 – Onda quadrada ideal

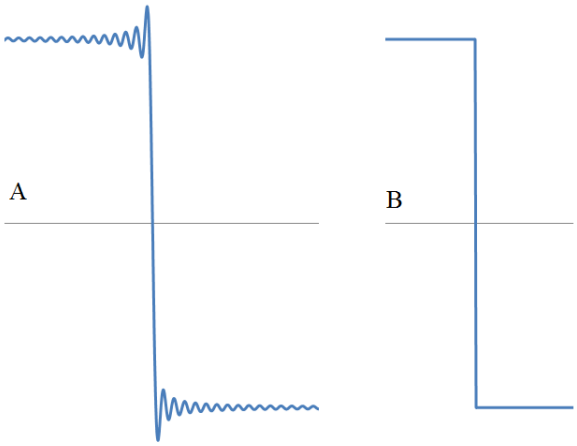


Fonte: elaborada pelo autor

Analisando a descontinuidade da onda quadrada, comparando a Figura 5-11 com a Figura 5-12, é possível observar o efeito que a limitação na largura de banda da onda causa. Em

vez de uma transição instantânea e sem valores intermediários, pode-se observar uma pequena oscilação (*ripple*) que começa a acontecer um pouco antes do ponto de descontinuidade, para então observarmos a transição da onda, que é novamente seguida por uma oscilação, e em seguida esta oscilação reduz (mas nunca é totalmente eliminada no caso da largura de banda limitada). Este efeito é chamado de fenômeno de Gibbs (HEWITT; HEWITT, 1979) e pode ser visto em pontos de descontinuidade em sinais limitados em banda. A Figura 5-13 apresenta uma comparação entre os pontos de transição – o gráfico A mostra a transição do sinal contendo 50 harmônicas, enquanto que o gráfico B apresenta a transição do sinal ideal, teórico, contendo infinitas harmônicas.

Figura 5-13 – Comparação entre o ponto de transição de uma onda com largura de banda limitada (A) e uma com largura de banda infinita (B)



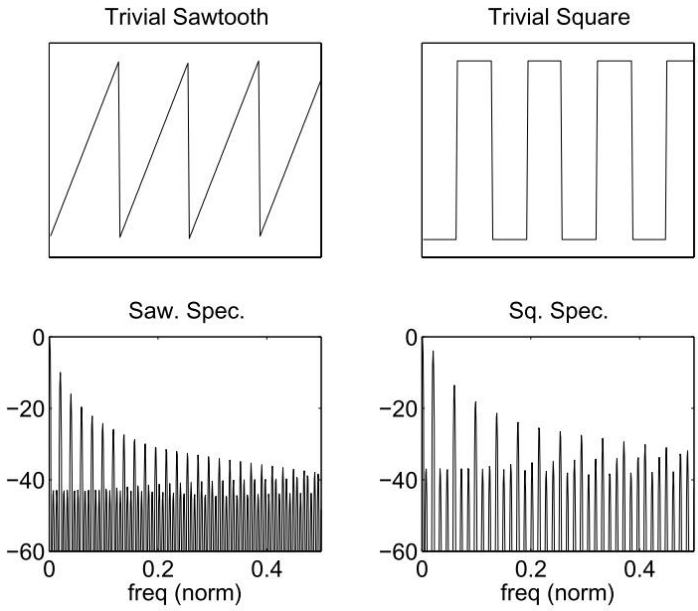
Fonte: elaborada pelo autor

Utilizando-se o método trivial ou uma tabela para geração da forma de onda quadrada contendo o sinal ideal, o instante de tempo da transição do ponto máximo para o ponto mínimo será arredondado para o ponto de amostragem mais próximo, causando um escorregamento no momento da transição, e ainda assim não gerando a oscilação característica do sinal de largura de banda limitada que antecede e sucede a transição. (STILSON; SMITH, 1996a).

A Figura 5-14 apresenta os sinais de onda dente-de-serra e onda quadrada com seus respectivos espectros de frequência equivalentes, assumindo a geração dos mesmos com um oscilador trivial. No gráfico do espectro de frequências, percebe-se harmônicas de amplitudes reduzidas (no gráfico, aproximadamente a partir de -40 dB para baixo) que não fazem parte de um sinal de onda dente-de-serra e onda quadrada ideal. Estas componentes harmônicas são claramente audíveis em um sinal de áudio e alteram o timbre e característica sonora em

comparação a um sinal de onda quadrada ou dente-de-serra perfeita (ou gerados com algoritmos que não geram *aliasing*).

Figura 5-14 – Espectro de frequências de um oscilador trivial



Fonte: Stilson e Smith (1996a)

Para remover o *aliasing* existente no sinal, é necessário utilizar um algoritmo de síntese de forma de onda com largura de banda limitada. Os autores Valimaki e Huovilainen (2007) classificam estes algoritmos em três categorias:

- Métodos de geração com limitação de banda, nos quais apenas as harmônicas até metade da frequência de amostragem são geradas;
- Métodos *quasi-bandlimited*, nos quais é aceitável possuir um pouco de *aliasing*, especialmente em altas frequências;
- Métodos de remoção de *aliasing* baseado em modificação do espectro, nos quais é aceitável possuir *aliasing* em toda largura de banda, mas de baixa amplitude especialmente em frequências graves e médias.

Valimaki e Huovilainen (2007) também apresentam uma tabela (reproduzia e traduzida na Tabela 5-1) resumo de algoritmos para geração de formas de onda sem *aliasing*.

Tabela 5-1 – Comparativo de algoritmos *anti-aliasing*

Algoritmo	**Complexidade computacional**	**Consumo de memória**	**Qualidade de som final**	**Observações**
Trivial	Muito baixa	NA	Ruim	Praticamente inutilizável, exceto em baixas frequências
Lane	Média	Muito baixo	Bom	Requer um oscilador senoidal
DPW	Muito baixa	Muito baixo	Bom	Simples e útil
DPW2X	Baixa	Muito baixo	Muito bom	Levemente melhor que DPW
PolyBLEP	Baixa	Muito baixo	Muito bom	Divisão a cada descontinuidade
Tabela	Média	Muito alto	Muito bom	Requer interpolação
BLIT-SWS	Média-Alta	Baixo	Muito bom	Divisão a cada descontinuidade
BLEP	Média	Baixo	Excelente	Divisão a cada descontinuidade
Síntese aditiva	Muito alta	Muito baixo	Excelente	Alta exigência computacional

Fonte: Valimaki e Huovilainen (2007, p.24), traduzida pelo autor

Os principais métodos indicados nesta tabela são analisados a seguir.

5.2.4 Geração de forma de onda com limitação de banda

Neste método, apenas as harmônicas até metade da frequência de amostragem são geradas. Isto pode ser obtido através da síntese aditiva, síntese por tabela de onda ou equações que descrevam a onda com largura de banda limitada.

A síntese aditiva pode ser utilizada com uma tabela de valores, evitando o custo computacional de cálculo de seno. Porém, isto aumenta a exigência de memória. A quantidade de harmônicas a ser utilizada depende do formato da onda, e todas as harmônicas, desde a fundamental até a metade de frequência de amostragem, podem ser obtidas e somadas para gerar um sinal sem *aliasing*. O custo computacional deste método é bastante elevado, e aumenta de forma inversamente proporcional à frequência da harmônica fundamental.

O uso de síntese por tabela de onda também pode ser considerado, porém o consumo de memória se torna extremamente alto pois, para evitar a geração de *aliasing*, diversas tabelas para diferentes frequências são necessárias. Uma combinação mista destas técnicas pode apresentar um bom resultado final, mas pode apresentar problemas com a troca de frequência da harmônica fundamental, pois exige a troca das tabelas a serem usadas, o que pode gerar uma descontinuidade indesejada no sinal.

O uso destes métodos exige um balanço entre o alto custo computacional ou alto consumo de memória. Enquanto a síntese aditiva pura possui um custo computacional muito elevado, o uso de tabela de memória exige um alto custo de memória para atingir bons resultados finais. Devido a estes pontos, estes métodos são pouco utilizados em implementações atuais, porém servem como referência para o uso e avaliação em conjunto com outros métodos.

5.2.5 Geração de forma de onda com métodos *quasi-bandlimited*

Os métodos classificados como *quasi-bandlimited* permitem um pouco de *aliasing* especialmente em altas frequências, nas quais o ouvido humano é menos sensível. O princípio utilizado nestes métodos é filtrar a forma de onda para atenuar as frequências acima da metade da frequência de amostragem.

5.2.5.1 BLIT – Band Limited Impulse Train

Formas de onda clássicas de sintetizadores analógicos, como onda quadrada, triangular e dente de serra podem ser obtidas através da integração de um trem de impulsos com banda limitada. Um trem de impulsos com banda limitada (*BLIT – band limited impulse train*), pode ser obtido aplicando-se um filtro passa-baixa em um trem de impulsos unitários (PEKONEN, 2007).

A equação 10 define um trem de impulso com banda limitada.

$$y[n] = \sum_{k=-\infty}^{\infty} sinc\left(2\frac{fc}{fs}(n - kP\right) \quad (10)$$

onde *P* é o período entre amostras, *fc* é a frequência de corte e *fs* a frequência de amostragem.

Analisando a equação 9, é possível perceber que o cálculo do BLIT a cada instante de tempo envolve um somatório com a função *sinc*, que se torna impossível de implementar na prática. Uma alternativa é o uso da técnica de janelamento para a função *sinc* (técnica utilizada para truncar e otimizar a resposta de um filtro ou função no tempo discreto), e a partir daí efetuar uma integração para obter a forma de onda desejada. Este método é também chamado de BLIT-SWS (*Band Limited Impulse Train – Summing Windowed Sinc-funtions*).

A técnica BLIT-SWS permite a implementação da geração de formas de onda com um custo computacional menor do que as técnicas de geração por limitação de banda, porém também exige um custo de processamento alto devido ao somatório de funções *sinc*, pois quanto menor for o número de componentes utilizadas para o somatório, maior será o erro de *aliasing.* (PEKONEN, 2007) (STILSON; SMITH, 1996a).

5.2.5.2 BLEP - Bandlimited Step Function

Os métodos chamados de BLEP podem ser considerados como uma extensão do método BLIT. No método BLIT, as formas de onda são geradas através da integração de trem de impulsos, enquanto que no método BLEP as mudanças na amplitude do sinal nos pontos de descontinuidade da onda são geradas com a função degrau unitário com largura de banda limitada (em vez da função impulso unitário com largura de banda limitada). A função degrau unitário é obtida através da integração da função impulso unitário. Tendo o sinal degrau unitário limitado em banda, calcula-se o sinal residual (subtrai-se a função degrau unitário limitada em banda da função degrau unitário ideal) nos instantes próximos às descontinuidades na forma de onda. Soma-se então este valor residual à forma de onda sem limitação de banda (PEKONEN, 2007) (VALIMAKI, 2007).

Assim como no método BLIT, este método resulta em um somatório de funções *sinc*, sendo possível também evitar isto utilizando tabelas de valores previamente calculadas, exigindo assim um uso maior de memória.

5.2.5.3 PolyBLEP – Polynomial Bandlimited Step Function approximation

Apesar da boa redução de *aliasing* que os métodos BLIT, BLIT-SWS, BLEP e MinBLEP apresentam, a implementação dos mesmos requer um custo computacional alto, ou então um alto consumo de memória para armazenamento dos sinais com largura de banda limitada para diferentes faixas de frequência.

Os autores Valimaki e Huovilainen (2007) e Pekonen (2007) apresentam uma alternativa a estes métodos, derivado do método BLEP, chamado de PolyBLEP, que baseia-se no fato de que a principal diferença entre o sinal com largura de banda limitada e o sinal com largura de banda tendendo ao infinito está nas descontinuidades. Para evitar o uso de tabelas de memória ou somatórios com custo computacional alto, este método utiliza um polinômio previamente calculado, que é aplicado na geração da forma de onda nos instantes que antecedem e sucedem a descontinuidade, para dar à mesma o formato de uma onda com largura de banda limitada (lembramos aqui da Figura 5-13, que apresenta a diferença entre as duas formas de onda no momento da descontinuidade). O polinômio proposto pelos autores pode ser visto na equação 11.

$$p(t) = \begin{cases} \frac{t^2}{2} + t + \frac{1}{2}, \ para \ -N \leq t \leq 0 \\ t - \frac{t^2}{2} - \frac{1}{2}, \ para \ 0 < t \leq N \\ 0, \ para \ N > t \ e \ t < -N \end{cases} \quad (11)$$

onde *t = 0* é o instante no qual ocorre a descontinuidade na onda e *N* é dado pela equação $N = 1/fs$, sendo *fs* a frequência de amostragem.

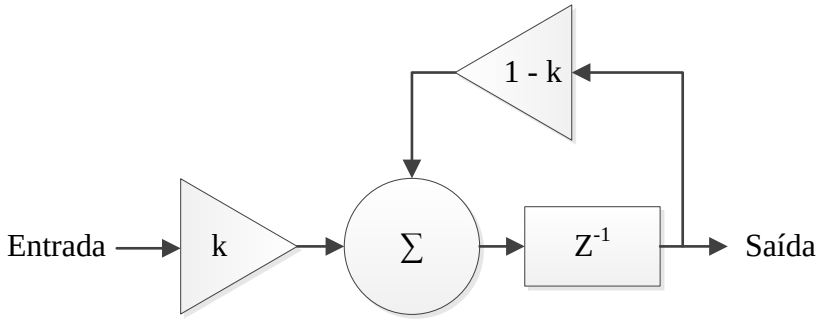
5.3 Filtros Digitais

De acordo com Coulter (2000), a técnica mais utilizada no processamento de áudio digital é o uso de filtros lineares (note que inclusive algumas das técnicas anti-*aliasing* apresentadas na seção 5.2.5 baseiam-se na aplicação de filtros na resposta de algumas funções matemáticas). O autor ainda complementa que os filtros mais utilizados são os filtros lineares e não-variantes no tempo, e são classificados em dois tipos básicos: IIR (*infinite impulse response*) e FIR (*finite impulse response*). Filtros FIR não possuem realimentação de saída, não podem oscilar e seu valor de saída tende a zero se a entrada for removida. Já os filtros IIR possuem realimentação e podem oscilar dependendo da sua estrutura e coeficientes utilizados.

Filtros analógicos em geral são do tipo IIR, porém a implementação digital pode ser de qualquer um dos dois tipos. Filtros do tipo FIR tendem a ser mais fáceis de desenvolver e calcular, porém exigem mais processamento, enquanto filtros IIR tendem a ser mais difíceis de calcular, porém são computacionalmente mais eficientes. Para nosso desenvolvimento, um filtro IIR deve ser utilizado para tentar reproduzir as características e sonoridade de um filtro analógico utilizado nos sintetizadores reais.

A Figura 5-15 apresenta um exemplo de topologia de um filtro digital de primeira ordem que é uma aproximação do que seria equivalente a um filtro RC analógico. A resposta deste filtro é dada por $Saída[n] = k.Entrada + (1 - k).Saída[n - 1]$, sendo $0 < k < 1$.

Figura 5-15 – Filtro “RC” digital



Fonte: Coulter (2000, p.114), adaptado pelo autor

5.4 Efeito de reverberação

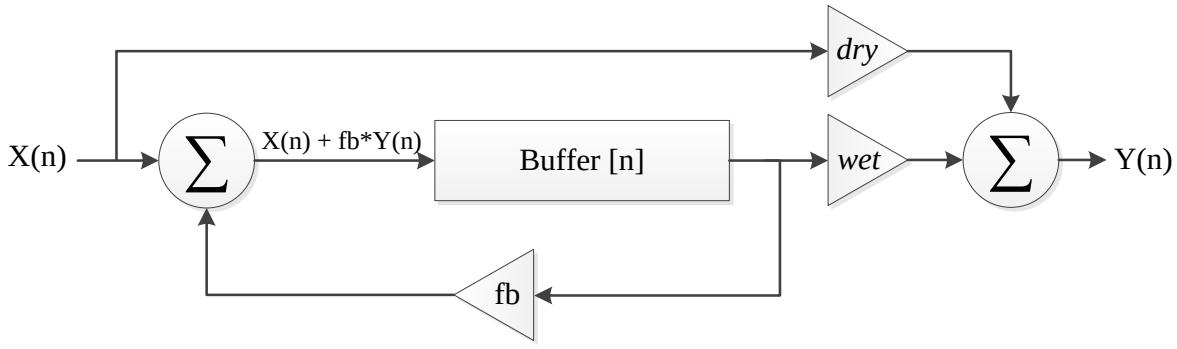
A reverberação é um efeito acústico natural do som. Escutamos este tipo de efeito em igrejas, salas de concerto e outros espaços com superfícies refletoras. Os sons emitidos nestes

espaços são reforçados diversas vezes pelas ondas sonoras que ecoam e refletem em paredes e objetos. Por outro lado, sons sintetizados possuem muito pouco ou nenhuma característica de reverberação. Devido a isso, estes sons podem ser modificados para adicionar essa sensação espacial, assimilando-se à experiência de ouvir ou escutar um instrumento sendo executado em um ambiente amplo. (ROADS, 1996). Quando lançados originalmente, os sintetizadores analógicos subtrativos não possuíam nenhum tipo de processador de efeito, porém atualmente praticamente todos os sintetizadores (digitais e analógicos) possuem algum processador de efeitos internamente, e quando não o possuem, em geral este efeito é adicionado com algum módulo externo. (PIRKLE, 2014).

A implementação digital ou em *software* de um efeito de reverberação utiliza linhas de atraso (*delay*), filtros e mixagem para simular o resultado de um sinal sonoro se espalhando por um ambiente. Do ponto de vista de processamento de sinais, a implementação consiste em um filtro com uma resposta ao impulso que simule a resposta ao impulso de uma sala. O tipo de filtro que optamos por implementar simula um eco que vai reduzindo sua amplitude com o tempo. A topologia utilizada para esta implementação é conhecida como filtro pente (*comb filter*), pois cria uma série regular de picos no espectro do sinal de entrada, lembrando o formato de um pente. (ROADS, 1996).

A implementação digital para este tipo de efeito utiliza uma linha de atraso, que explora o conceito de *buffer* circular para criar um atraso entre o sinal de entrada e o sinal de saída, e adicionando uma realimentação para gerar a repetição do sinal sonoro. A Figura 5-16 mostra a estrutura de um efeito de *delay* com realimentação, utilizada na implementação deste trabalho.

Figura 5-16 – Estrutura de um delay com realimentação utilizada neste trabalho



Fonte: Pirkle (2014), adaptado pelo autor

O sinal de entrada é enviado a dois diferentes caminhos. O caminho chamado *dry* não aplica nenhum efeito no sinal. Neste caminho, o sinal é multiplicado por um ganho *dry* e então somado e enviado para a saída. Já o outro caminho, chamado *wet*, é o caminho no qual o efeito é aplicado. O sinal de entrada é somado com o valor de saída de um *buffer* circular após a

multiplicação por um ganho *fb* (menor que 1). Esta soma então retorna ao *buffer* circular. Ao valor de saída do buffer circular também é aplicado ganho *wet*, e então somado com o sinal do caminho *dry* e enviado para a saída. A existência do buffer circular faz com que o sinal de entrada seja enviado à saída após um certo tempo, causando a repetição do sinal de entrada na saída com um ganho cada vez menor (PIRKLE, 2014).

Os parâmetros *dry* e *wet* são parâmetros de ajuste e controlam a intensidade na qual o efeito é aplicado ao sinal de entrada. Em geral estes parâmetros são ajustados de forma percentual, sendo que a soma dos dois é 100%. Um ajuste de 40% para *dry* e 60% para *wet*, por exemplo, faz com que 40% da intensidade do sinal de entrada seja enviado diretamente à saída, enquanto que os outros 60% do sinal de saída é composto do valor do caminho *wet*, no qual foi aplicado o efeito.

6 DESENVOLVIMENTO DO SINTETIZADOR

Este trabalho tem como um dos objetivos o desenvolvimento de um software que execute as funções de um sintetizador subtrativo. Este software será executado em um sistema embarcado, portanto alguns cuidados e opções de desenvolvimento foram tomadas visando facilitar o desenvolvimento e atingir todos os objetivos citados na seção 1.2. Abaixo algumas destas opções são descritas.

- *Design* e modelo do *software*: para tornar o mesmo modular, adaptável e apto a modificação e criação de novas características, utilizamos o conceito de orientação a objetos para criação de módulos (classes e objetos) com as mesmas responsabilidades que os módulos e interconexões que existiam nos sintetizadores analógicos modulares;
- Linguagem de programação: para facilitar o desenvolvimento de classes e objetos, a linguagem C++ foi utilizada. Porém, evitou-se o uso intensivo de características específicas da linguagem C++ devido ao custo computacional (tamanho de código e tempo de processamento);
- Simulação e teste: tratando-se de um software a ser executado em ambiente embarcado, ideias e conceitos relacionados ao teste e simulação de sistemas embarcados, apresentadas em (Gomes, 2010) foram utilizadas – o objetivo é facilitar a simulação e teste de módulos do software em um computador convencional;
- Utilização de recursos: evitamos o uso intensivo de vetores e tabelas em memória (reduzindo requerimento de memória RAM do sistema) e o uso intensivo de operações matemáticas, em especial operações de divisão (reduzindo requerimento de velocidade de processamento).

Para o desenvolvimento do código, as principais ferramentas utilizadas foram as seguintes:

- IDE (*Integrated Development Environment*) do Arduino: possui incluso o compilador e gravador para carregar o código executável na placa de desenvolvimento;
- Eclipse: para edição do código-fonte em geral;
- Microsoft Visual Express 2015: para edição, compilação e depuração do código em ambiente PC;

- Microsoft Visio 2010: para elaboração de diagrama de classes e digrama de blocos;
- Microsoft Excel 2010: para visualização de gráficos gerados através de simulação com o Microsoft Visual Express 2015.

6.1 Comparação entre ponto fixo e ponto flutuante

O processador existente na plataforma de desenvolvimento não possui unidade para realização de cálculos matemáticos de ponto flutuante em *hardware*. Desta forma, toda implementação é realizada através de bibliotecas de software, o que torna os cálculos de ponto flutuante mais lentos. Para contornar esta limitação, utilizamos o conceito de ponto fixo para todos os cálculos (com o uso de ponto fixo, apenas cálculos com valores inteiros são necessários). De acordo com Yates (2013), o uso de ponto fixo é um conceito no qual a interpretação dada a um valor representado em valor binário pode ser relativo e depende de como decidimos utilizar o valor representado. Um número binário contendo 8 bits pode representar até 256 valores diferentes ($2^8 = 256$). Em geral, este número é utilizado para representar um valor inteiro, que pode ir do valor 0 até 255, representando assim 256 possibilidades.

Na representação por ponto fixo, utiliza-se um "ponto virtual" (*virtual binary point*), ou ponto implícito (*implied binary point*), para separar a representação do valor inteiro e do valor fracionário. Para exemplificar, consideramos um número binário de 8 bits, sendo 6 bits utilizados para representar a parte inteira, e 2 bits para representar a parte fracionária. O valor equivalente a esta representação é dado por $X = 2^A - 2^{-B}$, onde A representa o valor dos 6 bits da parte inteira, e B representa o valor dos 2 bits da parte fracionária. Cada bit da parte fracionária equivale a uma resolução de $\frac{1}{2^N}$, ou seja, neste caso, o equivalente ao número decimal 0,25. A Tabela 6-1 apresenta alguns exemplos de representação de números com ponto fixo e o equivalente em formato decimal.

Tabela 6-1 – Demonstração de representação de números binários com ponto fixo

Número binário	Parte inteira	Parte fracionária	Equivalente em decimal
00000000	000000	00	0
00000001	000000	01	0,25
00000100	000001	00	1,00
00000101	000001	01	1,25
00001011	000010	11	2,75
00001010	000010	10	2,50

Fonte: elaborada pelo autor

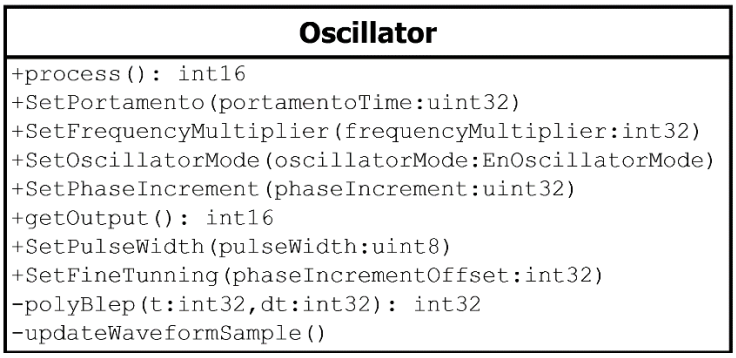
O uso de sinal também se aplica nesta representação, porém aplicado apenas à parte inteira da representação, respeitando o formato complemento de 2. O uso de ponto fixo para cálculos exige também que algumas regras sejam seguidas – estas regras não serão descritas aqui, e podem ser verificadas em Yates (2013).

6.2 Módulo oscilador

De acordo com Devahari (1982) e Crombie (1982), o módulo oscilador é responsável por converter um valor de tensão em uma forma de onda com uma determinada frequência. A frequência é determinada pelo valor da tensão de entrada. Além disto, o módulo oscilador possui entradas de controle (uma das mais comuns atua na modulação da frequência) e ajustes, como seleção da forma de onda de saída. Baseado nestas características, definimos o modelo da nossa classe *Oscillator*.

A Figura 6-1 mostra a definição da classe *Oscillator* com seus métodos públicos. Métodos privados e atributos internos não são exibidos.

Figura 6-1 – Diagrama da classe *Oscillator*



Fonte: elaborada pelo autor

Breve descrição de cada método:

- *process*: método que deve ser chamado sempre com intervalo igual ao período de amostragem. Realiza os cálculos para geração de forma de onda, e retorna o valor da forma de onda do instante da chamada;
- *SetPortamento*: habilita e configura o portamento, que significa quanto tempo o oscilador levará para trocar de uma frequência à outra, quando houver troca;
- *SetFrequencyMultiplier*: configura um multiplicador da frequência original do oscilador. Pode ser utilizado em conjunto com um LFO para gerar um efeito de vibrato no oscilador;
- *SetOscillatorMode*: seleciona a forma de onda a ser gerada;

- *SetPhaseIncrement*: configura o valor do incremento de fase do oscilador, que é proporcional à frequência de saída desejada (este método, em outras palavras, é que define a frequência de saída principal do oscilador);
- *SetPulseWidth*: configura a largura de pulso, quando o oscilador estiver em modo pulso. Quando selecionado em 50%, equivale à geração de uma forma de onda quadrada.

O algoritmo para geração da forma de onda utilizado foi o PolyBLEP (exceto para onda senoidal), descrito na seção 5.2.5.3. Após estudo e análise dos algoritmos citados nas seções 5.2.4 e 5.2.5, e do estudo comparativo apresentado por Valimaki e Huovilainen (2007), optamos por utilizar o PolyBLEP em função da baixa necessidade de memória e reduzido custo computacional. A implementação do oscilador utiliza o método trivial para geração das formas de onda quadrada e dente-de-serra em conjunto com o polinômio PolyBLEP. Para onda senoidal é utilizado apenas o algoritmo de busca por tabela de onda (descrito em 5.2.2).

A Figura 6-2 apresenta o código utilizado para cálculo do polinômio PolyBLEP. O código foi otimizado para trabalhar com números inteiros 32 bits.

Figura 6-2 – Código da função para cálculo do PolyBLEP

```
int32 polyBlep(int32 t, int32 dt)
{
   //  t = [12.20] - sample phase / 1024
   // dt = [12.20] - phase increment / 1024

   // For 0 < t <= 1
   if (t <= dt)
   {
      // Get normalized t
      t = (t << 10) / dt;     // [12.20]/[12.20]
                              // --> [22.10]=[2.30]/[12.20]
      // PolyBLEP
      return ((t + t) << 10) // [12.20]
             - t * t         // [12.20]=[22.10]*[22.10]
             - (1 << 20);    // [1.20]
             // return 12.20
   }
   // -1 <= t <= 0
   else if (t >= ((1 << 20) - dt))
   {
      // Get normalized t
      t = ( (t << 10) - (1 << 30) ) / dt;  // [12.20]/[12.20] -->
                                           // [22.10]=[2.30]/[12.20]
      // PolyBLEP
      return t * t  // [12.20]=[22.10]*[22.10]
             +  ((t + t + (1 << 10)) << 10);  // 12.20
   }
   // 0 otherwise
   return 0;
}
```

Fonte: elaborado pelo autor

O método *process* da classe *Oscillator* deve ser chamado sempre no período de amostragem, e o valor configurado para o incremento de fase (através do método *SetPhaseIncrement*) deve levar em conta a frequência de amostragem. Para a seleção da

frequência do oscilador, o método *SetPhaseIncrement* deve ser utilizado (em conjunto com o método *getTwoPI*, que retorna o número de pontos que o oscilador considera como um período completo da onda - no caso desta implementação, 1024), lembrando que o valor do incremento de fase é definido pela equação 7, e o valor a ser passado como parâmetro deve ser em ponto fixo (16 bits parte inteira com sinal e 16 bits parte decimal). As equações 12, 13, 14 e 15 apresentam os cálculos para selecionar a frequência de 440 Hz, à uma taxa de amostragem de 48 kHz. Também é apresentada a conversão para ponto fixo.

$$IncFase = \frac{2\pi f_{REF}}{f_{SR}} = \frac{1024.440}{48000} = 9{,}3866666667 \quad (12)$$

$$INT(9{,}386666667) = 9 \quad (13)$$

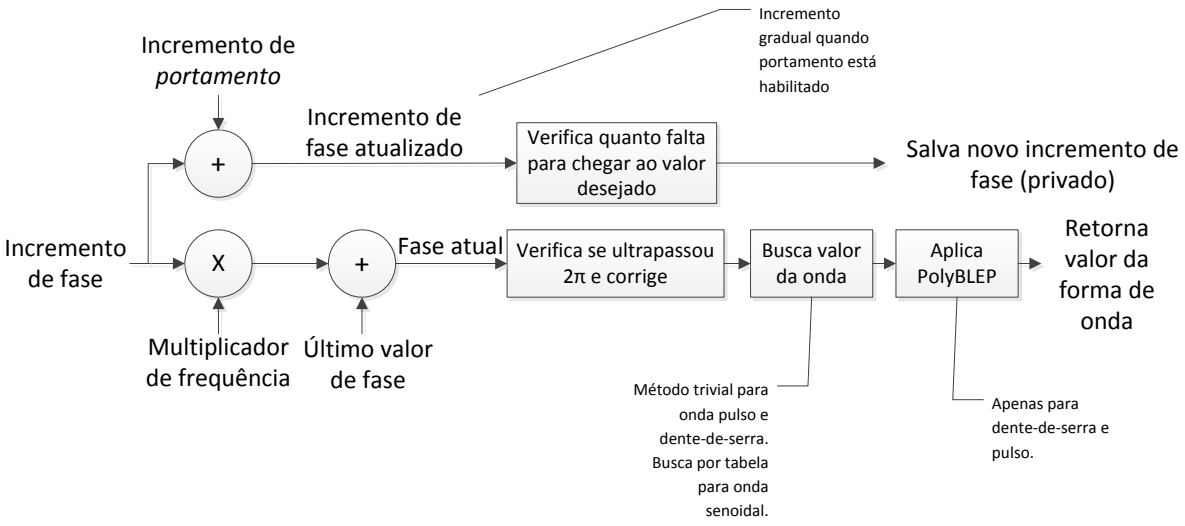
$$\frac{9 - INT(9{,}386666667)}{\frac{1}{2^{16}}} = 25340 \quad (14)$$

$$Valor\ Ponto\ Fixo = (9 \ll 16) + 25340 = 615164 \quad (15)$$

O valor a ser configurado no oscilador é 615164, que equivale a 9,386657715 em ponto fixo, apresentando um erro de 0,00089% em comparação com o valor teórico, apresentado na equação 10. Para evitar o cálculo destes valores em tempo de execução, uma tabela foi previamente carregada em memória contendo os valores necessários para as notas musicais. No ANEXO A é apresentada a tabela com o cálculo do valor de incremento de fase para cada nota musical.

A Figura 6-3 apresenta a sequência de cálculo realizado pelo método *process*. Uma das ramificações é responsável pelo cálculo do portamento, que na nossa implementação é vinculado ao oscilador, diferentemente dos sintetizadores analógicos, nos quais esta característica está vinculada ao teclado. A outra ramificação é o cálculo principal do oscilador, atualizando a fase da onda e verificando a necessidade do algoritmo PolyBLEP em função da forma de onda selecionada.

Figura 6-3 – Diagrama de cálculo realizado pelo método *process* da classe *Oscillator*



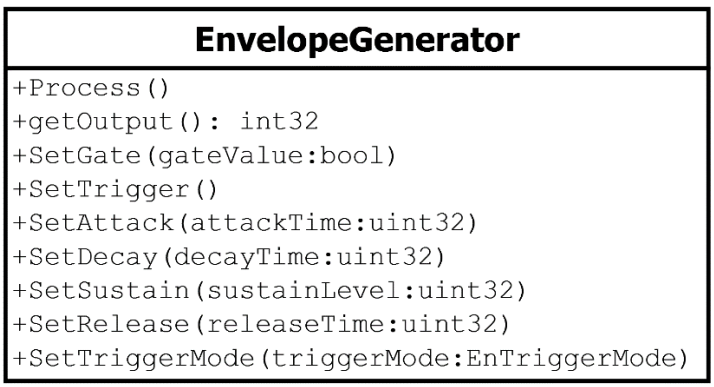
Fonte: elaborada pelo autor

6.3 Módulo gerador de envelope

De acordo com Devahari (1982) e Crombie (1982), o módulo gerador de envelope não produz um sinal sonoro propriamente dito, mas sim gera um sinal que serve para dar um contorno à alguma característica do sinal de áudio – como a amplitude de saída ou a frequência de corte de um filtro. A saída deste módulo é utilizada como entrada de controle para algum outro módulo, como o amplificador ou filtro.

O módulo gerador de envelope possui geralmente quatro controles (tempo de *attack*, tempo de *decay*, nível de *sustain* e tempo de *release*) e duas entradas (sinal de *gate* e pulso de *trigger*). Baseado nestas características, definimos o modelo da classe *EnvelopeGenerator*, apresentado na Figura 6-4 (apenas os métodos públicos são exibidos).

Figura 6-4 – Diagrama da classe *EnvelopeGenerator*



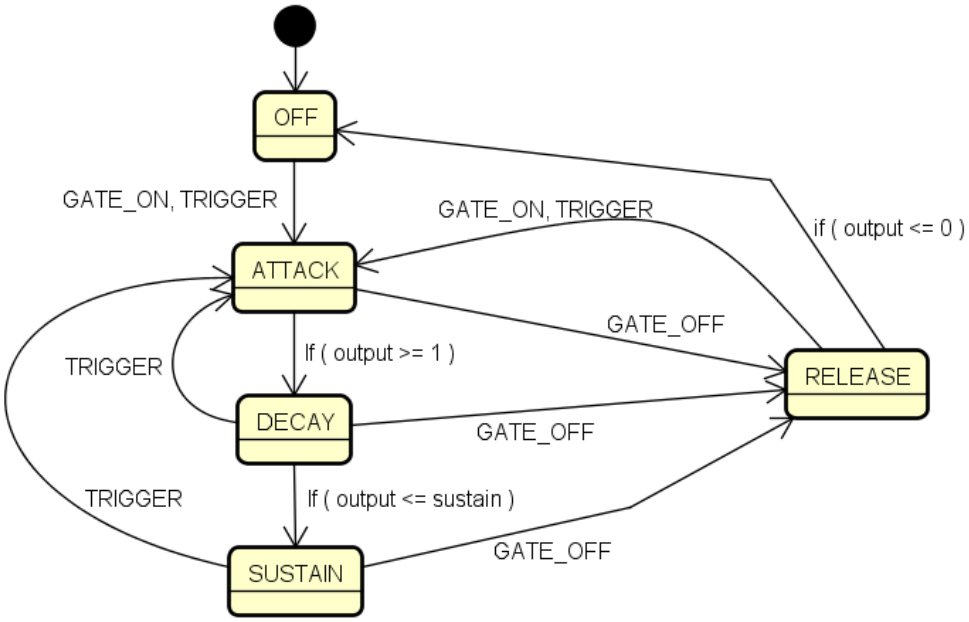
Fonte: elaborada pelo autor

O funcionamento desta classe se dá principalmente através do método *Process*. Este método deve ser chamado periodicamente, e o intervalo da chamada é configurado na instanciação da classe passada como um parâmetro. As unidades utilizadas nas interfaces são sempre em unidades de milissegundos para os parâmetros de tempo, e valores multiplicadores utilizando ponto fixo (16 bits de parte inteira e 16 bits de parte fracionária) para unidades de nível (como nos métodos *SetSustain* e *getOutput*) – o valor de saída, por exemplo, é retornado no método *getOutput*, e varia entre 0 e 65536, que é equivalente a 1,0. Este valor pode ser utilizado como multiplicador para o sinal de algum outro módulo.

Para simular o efeito dos sinais de *trigger* e *gate* existente nos geradores de envelope, foram criados dois métodos para manipular cada um dos sinais. O momento no qual estes sinais são gerados deve ser controlado externamente à classe. Com a inclusão destes, buscamos poder simular o comportamento dos geradores de envelope de sintetizadores como Moog e ARP, que possuíam características de *trigger* e *gate* diferentes um do outro.

A Figura 6-5 apresenta o diagrama de estados de funcionamento da classe *EnvelopeGenerator*.

Figura 6-5 - Diagrama de estados principal da classe *EnvelopeGenerator*

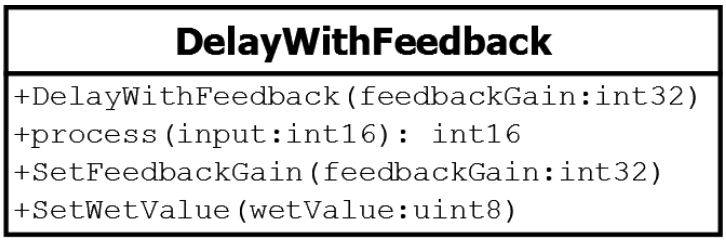


Fonte: elaborada pelo autor

6.4 Módulo efeito de atraso com realimentação

O módulo de efeito atraso com realimentação busca adicionar uma sensação de profundidade ao som, conforme descrito na seção 5.4. O diagrama da classe *DelayWithFeedback* é apresentado na Figura 6-6.

Figura 6-6 – Diagrama da classe *DelayWithFeedback*



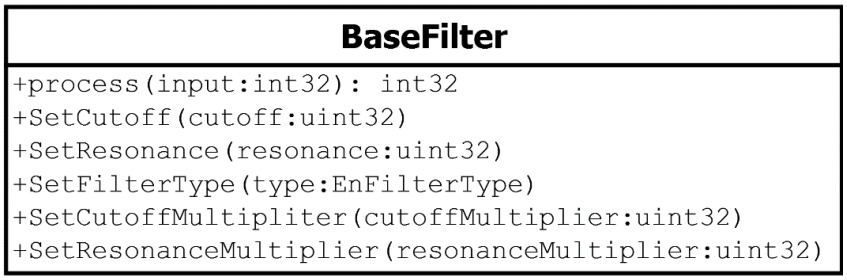
Fonte: elaborada pelo autor

Assim como nas classes anteriores, o método principal e que deve ser chamado periodicamente no período de amostragem é o método *process*. O mesmo realiza o processamento do efeito e retorna o valor de áudio atualizado. Outros dois métodos são implementados para configurar os parâmetros do módulo de efeitos: *SetFeedbackGain*, que configura o valor do ganho de realimentação (constante *fb* na Figura 5-16), e *SetWetValue*, que configura o valor do parâmetro *Wet* (e indiretamente o parâmetro *Dry*), também conforme a Figura 5-16.

6.5 Módulo filtro

Diferentemente dos outros módulos, para o módulo filtro criamos uma classe base, a ser especializada, para que possamos implementar diferentes tipos de filtros. Na classe base definimos os métodos a serem utilizados como interface com o restante dos módulos – estes métodos são mostrados na Figura 6-7.

Figura 6-7 – Diagrama da classe base para implementação de filtros



Fonte: elaborada pelo autor

De forma similar aos outros módulos, o método *process* deve ser chamado na frequência da taxa de amostragem para geração do áudio. Os métodos *SetCutoff* e *SetResonance* são utilizados para ajustar a frequência de corte e ressonância do filtro, enquanto o método *SetFilterType* foi adicionado para selecionar o tipo de filtro, quando necessário. Os métodos *SetCutoffMultiplier* e *SetResonanceMultipler* definem um multiplicador, menor ou igual a 1,0, para ser aplicado no parâmetro de frequência de corte e ressonância. O objetivo é que estes

métodos sejam utilizados em conjunto com um gerador de envelope, permitindo modular estes parâmetros.

Para este trabalho, implementamos dois filtros. O primeiro, baseado na implementação disponibilizada por Finke (2013), é um filtro passa-baixa de quarta ordem, com atenuação de 24 dB por oitava e ressonância.

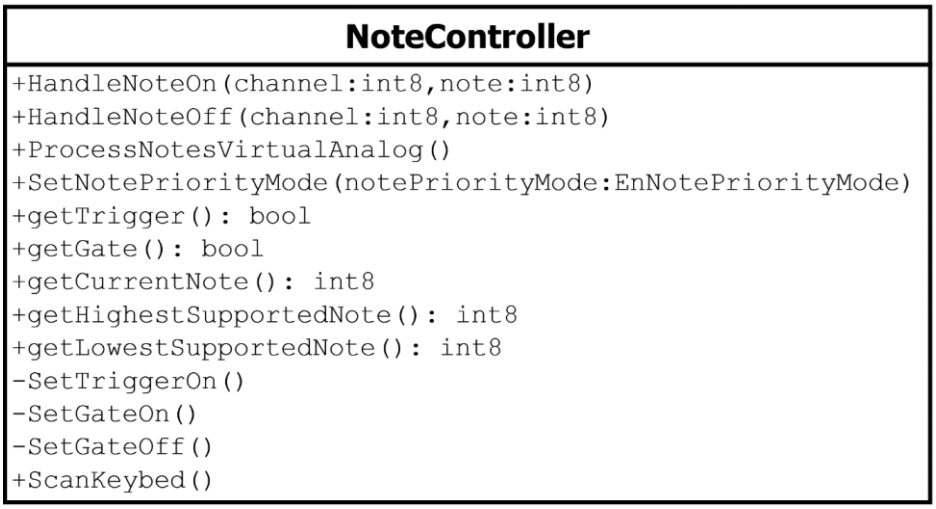
O segundo é um filtro de quarta ordem, com ressonância e atenuação de 24 dB por oitava, baseado na implementação de Kellett (sem data), que utilizou como referência a análise feita por Stilson e Smith (1996b) sobre o filtro projetado por Robert Moog para os sintetizadores Moog.

6.6 Módulo controlador de notas

Conforme a descrito no capítulo 3.3.1 e demonstrado na Figura 3-7, um módulo controlador é responsável por identificar as notas sendo executadas, enviar esta informação aos outros módulos do sintetizador (num sintetizador analógico monofônico, através de um nível de tensão para o oscilador), e gerar alguns outros sinais como *gate* e *trigger*, utilizados por módulos como o gerador de envelope. O controlador geralmente é associado a um teclado, embora um sintetizador com entrada MIDI possa ter esta mesma função realizada pelo módulo MIDI.

Na nossa implementação, a Figura 6-8 mostra o diagrama da classe responsável por executar estas funções.

Figura 6-8 – Diagrama da classe *NoteController*



Fonte: elaborada pelo autor

Esta classe provê métodos que podem servir de *callback* para bibliotecas MIDI (*HandleNoteOn* e *HandleNoteOff*), assim como serem utilizadas por funções de varredura de

teclado. Os métodos *getGate* e *getTrigger* são utilizados em conjunto com os módulos externos que pretendem utilizar este sinal, como por exemplo um gerador de envelopes. O sinal de trigger é limpo na leitura, ou seja, uma vez acionado internamente, assim que for invocado o método *getTrigger*, o método retorna *true*, e logo em seguida o sinal de trigger retorna para *false* internamente, simulando assim um pulso.

Para simular fielmente o uso dos sinais *gate* e *trigger*, pensamos em utilizar o *design pattern Observer*, que seria uma forma de notificar outros módulos sempre que estes sinais são alterados. Porém, para usar este *design pattern* seria necessário algum tipo de lista encadeada ou alguma estrutura um pouco mais complexa para que o módulo controlador de notas pudesse realizar a notificação de mudança destes sinais. Considerando o fato de estarmos desenvolvendo para uma plataforma com processamento limitado, optamos então por não utilizar este *design* neste momento, ficando como uma possibilidade de melhoria futura o estudo e uso do *Observer*.

6.7 Leitura do teclado externo

Para uso como teclado externo, havia à nossa disposição um antigo teclado Casio CTK-240 fora de funcionamento, o qual desmontamos e reaproveitamos as teclas. Possui 49 teclas não-sensitivas, e a leitura das mesmas é realizada através de uma matriz de 8 colunas e 7 linhas. A leitura deste teclado precisa alimentar individualmente cada coluna e ler as linhas, para detectar quais notas estão pressionadas.

Figura 6-9 – Teclado Casio desmontado para investigação de como ler as teclas pressionadas

Fonte: registrada pelo autor

O código para leitura deste teclado foi inserido diretamente na classe *NoteController* dentro do método *ScanKeybed*. Isto torna a classe *NoteController*, da forma como escrevemos o código desta função, específica para esta nossa implementação, sendo necessário modificar o código desta função para outras implementações de leitura de outros teclados caso necessário. Como desacoplar o código de leitura deste teclado da classe *NoteController* ou permitir outras implementações específicas dentro da mesma fica como sugestão para trabalhos futuros.

6.8 Entrada MIDI

Uma entrada MIDI também foi disponibilizada no protótipo como uma opção ao teclado externo. Para tratamento do protocolo MIDI utilizamos uma biblioteca de código-aberto disponibilizada para o Arduino Due no endereço https://github.com/FortySevenEffects/arduino_midi_library/.

6.9 Conversor DA externo

O processador Atmel SAM3X8E presente no Arduino Due possui dois conversores digital-analógico (DA) integrados, com resolução de 12 bits e taxa máxima de atualização de 1,68 MHz. (ATMEL, 2015). Estes conversores são apropriados para aplicações genéricas ou mesmo algumas aplicações envolvendo áudio, com a ressalva de que podem apresentar um nível de ruído elevado se comparado com conversores externos de alta qualidade. O início do desenvolvimento deste trabalho foi realizado utilizando estes conversores, e os mesmos apresentaram desempenho satisfatório e se mostraram uma ótima alternativa para desenvolver algum tipo de aplicação mais simples ou mesmo para validação de ideias e conceitos (a validação inicial dos algoritmos de geração de ondas foi inteiramente realizada com estes conversores integrados).

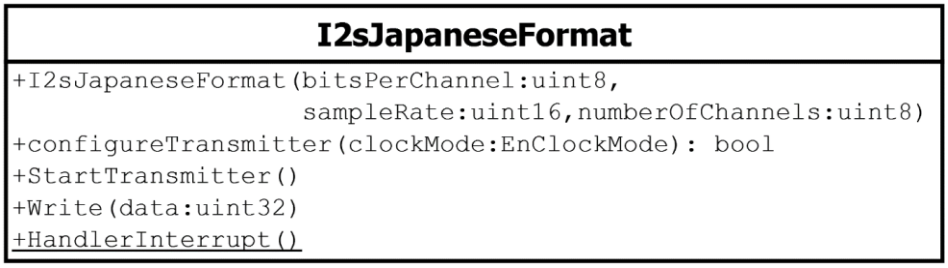
Para melhorar a qualidade no sinal de saída foi utilizado um conversor externo de 16 bits que apresenta um nível de ruído menor, é simples de utilizar e é bastante utilizado para aplicações de alta fidelidade, o TDA1543, fabricado pela Philips/NXP. O TDA1543 é um conversor digital-analógico de 16 bits e dois canais e utiliza interface de comunicação I2S. Existe o modelo TDA1543A (que é o que temos à disposição e utilizamos neste trabalho), que utiliza uma variação da interface I2S, conhecida como *Japanese Format* ou *Sony/Burr-Brown Format*. Este formato utiliza 3 sinais para comunicação serial: BCK (*bit clock input*), WS (*word select input*) e DATA (*data input*). O sinal DATA é utilizado para enviar o dado individual de cada canal a ser convertido. O tamanho do dado é de 32 bits, embora apenas 16 bits sejam de fato utilizados. Os 16 bits ainda devem ser enviados no formato chamado *24-bits left justified*, que significa realizar um deslocamento à esquerda em 8 bits antes de enviar. O sinal de *clock* é uma entrada que serve como referência para amostragem do sinal DATA. A frequência deste sinal deve respeitar a taxa de amostragem desejada, pois o conversor realiza a conversão e atualiza a saída após receber o dado completo. Para o nosso caso, utilizando uma taxa de amostragem de 48 kHz, 32 bits para o tamanho do dado (utilizado pelo conversor) e 2 canais (estéreo), a frequência do *clock* é mostrado na equação 16.

$$f_{clock} = 2\ X\ 32\ X\ 48000 = 3.072.000\ Hz \quad (16)$$

E, por fim, o sinal WS seleciona qual canal está sendo enviado, sendo nível alto para o canal esquerdo e nível baixo para o canal direito.

Para esta comunicação com o conversor, o processador utilizado no Arduino Due possui um módulo de comunicação serial I2S configurável, que permite ajustar os parâmetros necessários para funcionar com o conversar TDA1543A. Utilizamos então como referência o código fonte fornecido por Delsauce (2013?) e o manual do processador encontrado em Atmel (2015) para desenvolver uma classe responsável por configurar o módulo I2S do processador e lidar com a escrita dos dados e tratamento de interrupção. A Figura 6-10 apresenta o diagrama da classe desenvolvida.

Figura 6-10 – Diagrama da classe de comunicação com o TDA1543A



Fonte: elaborada pelo autor

6.10 Órgão baseado em síntese aditiva

Johann (2015) desenvolveu um órgão utilizando síntese aditiva inspirado nos órgãos Hammond, com capacidade para 61 notas, polifonia completa, configuração através de *drawbars* e efeitos para simular uma caixa Leslie. A implementação também foi realizada na plataforma Arduino Due, com uma taxa de atualização de saída de 24 kHz, que no caso desta implementação é um valor aceitável, uma vez que a maior frequência gerada através desta síntese é de 8362 Hz (frequência do oscilador mais agudo), e baseando-se na frequência de Nyquist, uma taxa de atualização de 16724 Hz já é suficiente. Inspirado em Johann (2015), reimplementamos a síntese aditiva neste trabalho em conjunto com a síntese subtrativa, com o objetivo de apresentar a possibilidade de duas formas de síntese fundamentais estarem disponíveis em uma plataforma simples e de baixo custo, podendo ser selecionado em tempo de execução o tipo de síntese a ser utilizado.

Na implementação de Laures Hammond para os órgãos Hammond, cada nota executada no teclado gera até 9 harmônicas, sendo cada uma controlada por uma *drawbar*. Ao total, os Hammond possuíam 91 osciladores, equivalentes às 61 notas existentes, mais os semitons mais

graves e mais agudos para completar as 9 harmônicas de todas as notas. Lembrando que, se na implementação da síntese subtrativa, na qual possuímos um ou mais osciladores e cada um é controlado para oscilar na frequência da nota pressionada, foi possível adicionar três osciladores (dois principais e um LFO), parece inviável implementar 91 osciladores para este tipo de síntese utilizando a mesma arquitetura proposta para o sintetizador subtrativo.

Porém, dois conceitos importantes utilizados por Johann (2015) tornaram possível esta implementação, os quais vamos explicar brevemente aqui:

6.10.1 Tarefas de tempo crítico, periódicas e eventuais

Originalmente propostas como *time-critical* (tempo-crítico), *time-accurate* (periódicas) e *housekeeping* (eventuais), esta classificação de tipos de tarefas permite balancear o tempo de processamento das funções do sintetizador, reduzindo o tempo de processamento total dentro da função de atualização da saída, que possui um curto espaço de tempo para processamento.

As funções de tempo-crítico são aquelas que precisam ser executadas num tempo exato, sem serem postergadas ou atrasadas, sob risco de prejudicar o funcionamento do sintetizador. Estas funções são as funções de processamento e geração do áudio principal, e que precisam ser executadas com a frequência da taxa de atualização de saída. As funções periódicas são aquelas que precisam de uma base de tempo para serem executadas, mas sua execução pode ser levemente atrasada sem comprometer a geração de áudio principal ou o funcionamento do sintetizador. Um exemplo é o processamento da máquina de estados do gerador de envelopes, que possui uma base de tempo, porém um leve atraso na sua execução não comprometerá a experiência e execução musical no sintetizador. E, por fim, funções eventuais são aquelas que podem ser chamadas sem compromisso de tempo, desde que o tempo de chamada não prejudique o funcionamento do sintetizador – como exemplo citamos a leitura de chaves de seleção e potenciômetros para controle de parâmetros.

6.10.2 Contabilização dos osciladores

Contabilizar as notas pressionadas e o valor de cada *drawbar* (contabilizada individualmente para cada nota), obtendo então a intensidade necessária para cada um dos 91 osciladores, além é claro do processamento destes 91 osciladores na frequência da taxa de amostragem parece inviável para o processador do Arduino Due. Para balancear o processamento, dividimos estas tarefas em tarefas eventuais e de tempo crítico.

A leitura do valor de cada *drawbar*, que na nossa implementação é a leitura de uma entrada analógica controlada por um potenciômetro, e a contabilização disto junto com as notas pressionadas e a intensidade total de cada um dos osciladores é realizada no laço principal do

código, sem compromisso de tempo. Assumindo que a modificação das *drawbars* e das notas pressionadas ocorram numa taxa muito menor que a frequência da taxa de amostragem, e que mesmo estando no laço principal, consigamos garantir que esta atualização seja realizada em um intervalo de tempo que não prejudique a execução musical, podemos remover essa parte do processamento de dentro da função de atualização do áudio de saída.

Desta forma, como tarefa de tempo crítico chamada na frequência da taxa de amostragem fica a contabilização dos osciladores, sendo todos de onda senoidal, utilizando o conceito de tabela de onda (seção 5.2.2) e com cálculo do incremento de fase similar ao utilizado no sintetizador subtrativo.

6.10.3 Efeitos adicionais

Para tornar mais próximo do som característico de um órgão Hammond, adicionamos à implementação um efeito vibrato e trêmolo. O efeito de vibrato é uma leve modulação na frequência do áudio final, enquanto que o trêmolo é uma leve modulação na amplitude. Os dois efeitos combinados geram um resultado similar ao efeito da caixa de som Leslie, muito utilizada em conjunto com os Hammods. Duas frequências de modulação foram implementadas, bem como um algoritmo para simular a aceleração e desaceleração da caixa Leslie, de forma independente para graves e agudos.

6.11 Simulação e teste

A implementação deste trabalho consiste em um sistema embarcado que é executado em uma plataforma com poucas interfaces de depuração. Assim, a possibilidade de simulação e teste de partes do software em um ambiente PC facilita e agiliza o desenvolvimento bem como a validação de algoritmos. Inspirados pelos conceitos apresentados por Gomes (2010), decidimos desde o início que o nosso software deveria estar apto a ser compilado em ambiente PC e que deveria ser possível de executar cada módulo independentemente para validação do mesmo.

Para tornar isto possível, alguns trechos de código foram protegidos por diretivas de compilação, evitando a compilação de acessos específicos ao hardware quando em ambiente PC. A Figura 6-11 mostra um trecho de código para configuração dos pinos de entrada e saída do Arduino, protegido pela diretiva de compilação.

Figura 6-11 – Configuração dos pinos do Arduino protegidos por diretivas de compilação

```
#if ( !defined(WIN32) && !defined(linux) )
   pinMode ( A1,  INPUT );
   pinMode ( A2,  INPUT );
   ...
   pinMode ( 51, INPUT_PULLUP );
#endif
```

Fonte: elaborada pelo autor

Em alguns casos, nos quais a adição de uma diretiva de compilação poderia dificultar a leitura do código, foram criadas macros para declaração de funções específicas do Arduino, como demonstrado na Figura 6-12. O objetivo é apenas permitir a compilação, sem que essas macros interfiram no código.

Figura 6-12 – Definições de funções de acesso ao hardware

```
#if defined(WIN32) || defined(linux)
   #define noInterrupts()
   #define interrupts()
   #define digitalRead( x ) false
   #define analogRead( x )  0
#endif
```

Fonte: elaborada pelo autor

Para a validação de módulos mais complexos e do sinal de áudio gerado, o driver do conversor DA foi implementado de tal forma que, quando rodando em ambiente PC, utiliza um arquivo texto para salvar todos os valores escritos. Assim, é possível executar o software e posteriormente analisar o arquivo para verificar os valores. Para isso, a classe recebe em seu construtor um parâmetro que identifica o nome do módulo, e este nome é então utilizado para criar o arquivo de valores de saída. A Figura 6-13 apresenta um exemplo disto.

Figura 6-13 – Construtor do driver do conversor digital-analógico

```
I2sJapaneseFormat ( const char* moduleName)
: m_pModuleName   ( moduleName )
, m_pDataOut      ( NULL )
{
#if !defined(WIN32) && !defined(linux)
   /* Initialize the SSC module and work in loop mode. */
   pmc_enable_periph_clk(ID_SSC);
   ssc_reset(SSC);
#endif
}
```

Fonte: elaborada pelo autor

No método de escrita de valores de saída, a classe salva os valores em arquivo quando executado em ambiente PC. A Figura 6-14 demonstra a escrita em arquivo.

Figura 6-14 – Método de escrita do driver do conversor digital-analógico

```
void Write ( int32 data )
{
#if !defined(WIN32) && !defined(linux)
   if ( m_pDataOut != NULL )
   {
      // TDA1543A uses 24-bit left justified format.
      *m_pDataOut = data << 8;
   }
#else
   std::ofstream fout;
   fout.open (m_pModuleName, std::ofstream::app);
   fout << data << std::endl;
   fout.close();
#endif
}
```

Fonte: elaborada pelo autor

A plataforma Arduino possui uma estrutura de execução com duas funções, chamadas *setup* e *loop*, que são chamadas automaticamente através de um código gerado pela interface de programação do Arduino. Como não existe uma função *main*, necessária para a compilação em PC, é necessário criar um trecho de código em separado que simule os passos do processador na chamada dos módulos a serem testados. A Figura 6-15 demonstra uma parte da função *main*, tendo a chamada da função *setup* do Arduino, a simulação de uma nota pressionada através de um evento MIDI, e em seguida a chamada das classes *NoteController* e *EnvelopeGenerator* para validação das mesmas.

Figura 6-15 – Função main utilizada para simulação em ambiente PC

```
#ifdef WIN32
int main()
{
   using namespace std;
   // Call Arduino setup function
   setup();
   // Simulate note pressed
   MidiHandleNoteOn(1, 60, 127);

   pNoteController->ProcessNotesVirtualAnalog();
   pAmplifierEnvelopeGenerator->SetGate( pNoteController->getGate() );
```

Fonte: elaborada pelo autor

O trecho de código acima foi compilado com o MS Visual Studio 2015, possibilitando executar a depuração passo a passo de cada uma das linhas e das funções chamadas.

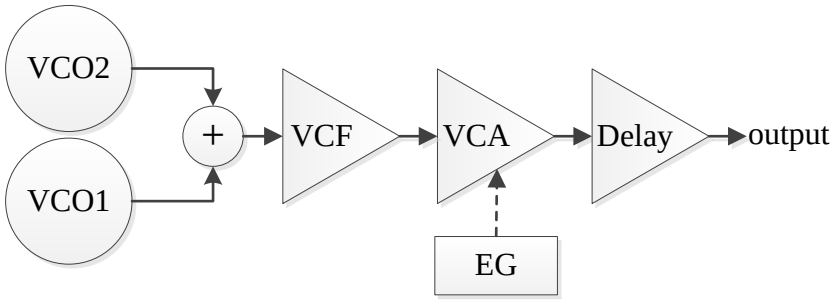
6.12 Integração dos módulos

Para atingir um dos objetivos citados na seção 1.2, que é o desenvolvimento de um software modular e flexível, separamos cada componente do *software* em classes diferentes,

evitando acoplamento entre as mesmas para que seja possível adicionar ou remover características facilmente. As funções principais de cada classe responsável por processamento de áudio possuem o mesmo tipo de dado como entrada e saída, facilitando a passagem de valores de uma classe para outra. Já para as funções principais de controle de cada classe, buscamos sempre utilizar, onde possível e apropriado, o formato de ponto fixo utilizando 16 bits de parte inteira e 16 bits como parte fracionária.

Para demonstrar como realizar a interligação entre os módulos, na Figura 6-16 apresentamos um diagrama de conexão de módulos típico de um sintetizador analógico modular.

Figura 6-16 – Conexão típica de um sintetizador analógico modular



Fonte: elaborada pelo autor

A Figura 6-17 apresenta as linhas de código necessárias para simular a mesma arquitetura no *software* desenvolvido.

Figura 6-17 – Conexão entre módulos do sintetizador no software

```
audioOut = pOscillator01->process() + pOscillator02->process();
audioOut = pLowPassFilter->process( audioOut );
audioOut = ( audioOut * pAmplifierEnvelopeGenerator->getOutput() ) >> EG_FRACTIONAL_WIDTH;
audioOut = pDelayWithFeedback->process( audioOut );
pTda1543a->Write( audioOut );
```

Fonte: elaborada pelo autor

Cada um dos módulos, representados por um objeto, é instanciado, configurado e parametrizado na inicialização e no laço principal do código. O trecho acima é o código executado periodicamente na taxa de atualização de saída (no caso do sintetizador subtrativo, 48 kHz). A variável *audioOut* se comporta como uma linha em um *mixer* de áudio, transportando o valor entre cada um dos módulos. Na primeira linha, os dois osciladores executam sua atualização (chamada do método *process*), retornam o valor atualizado para aquele instante de tempo, e já é efetuada a soma dos dois osciladores, guardando o resultado na variável *audioOut*.

Em seguida, o filtro passa-baixa recebe a mesma variável, processa o sinal, e devolve o resultado, para a seguir o mesmo ser multiplicado pelo valor do gerador de envelope (após a multiplicação o valor é deslocado à direita para normalização devido ao valor em ponto fixo do gerador de envelope). Na penúltima linha o efeito *delay* é processado, e no final a variável *audioOut* é enviada ao driver de comunicação com o conversor DA externo.

A instanciação dos módulos do exemplo acima é demonstrada na Figura 6-18.

Figura 6-18 – Inicialização dos módulos

```
pAmplifierEnvelopeGenerator = new EnvelopeGenerator( );
pOscillator01               = new Oscillator( );
pOscillator02               = new Oscillator( );
pDelayWithFeedback          = new DelayWithFeedback( );
pLowPassFilter              = new LowPassFilter( );
pTda1543a                   = new I2sInterface( "Tda1543A" );
pTda1543a->configureTransmitter( 32, 48000, eAUDIO_STEREO,
                                 eCLOCK_MODE_INTERNAL_CLOCK,
                                    &ProcessI2sInterrupt );
```

Fonte: elaborada pelo autor

Outro exemplo de código é o uso de um oscilador LFO para modular a frequência de um oscilador principal. Neste caso, o método de modulação de frequência do oscilador principal recebe um valor multiplicador em ponto fixo, sendo 65536 o equivalente a 1,0 (16 bits parte inteira e 16 bits parte fracionária). Assim, passando o valor 65536 não haverá alteração na frequência. A classe oscilador utiliza valores de 12 bits para geração de sinal, logo, o valor de saída do LFO oscilará entre -2048 e +2047. Para compor isto e transformar em um valor multiplicador, subtraímos o valor 65536 da saída do LFO, criando assim um valor multiplicador que oscila ao redor de 1,0.

Após testes práticos, achamos que o ideal para a sonoridade era utilizar apenas 10 bits de saída do LFO. Para isso, realizamos um deslocamento à direita com o valor de saída do LFO. Para calcular o percentual de oscilação que isto causará no oscilador principal, apresentamos os cálculos 17, 18, 19, 20 e 21, utilizando como exemplo a frequência de 440 Hz (nota Lá), que possui o valor de incremento de fase ideal de 9,38667.

$$f = \frac{K.IncFase.48k}{2\pi} = \frac{K.IncFase.48k}{1024} = K.IncFase.46{,}875 \quad (17)$$

$$f = 1.9{,}38667.46{,}875 = 440{,}0001 \quad (18)$$

$$\frac{1}{2^{16}}.2^{10-1} = 0{,}0078125 \quad (19)$$

$$f = (1 + 0{,}0078125).9{,}38667.46{,}875 = 443{,}40 \quad (20)$$

$$f = (1 - 0{,}0078125).9{,}38667.46{,}875 = 436{,}56 \quad (21)$$

Os cálculos mostram que a modulação da frequência, considerando a frequência de 440 Hz, fica em aproximadamente 3,5 Hz para mais e para menos, totalizando 7 Hz de modulação total. Para aplicar a modulação, o código abaixo é utilizado na função da taxa de atualização de saída, e faz com que a cada instante de tempo relativo à taxa de amostragem, o oscilador LFO é atualizado e seu valor é aplicado no oscilador principal para gerar uma alteração na frequência do mesmo. A Figura 6-19 demonstra um exemplo disto.

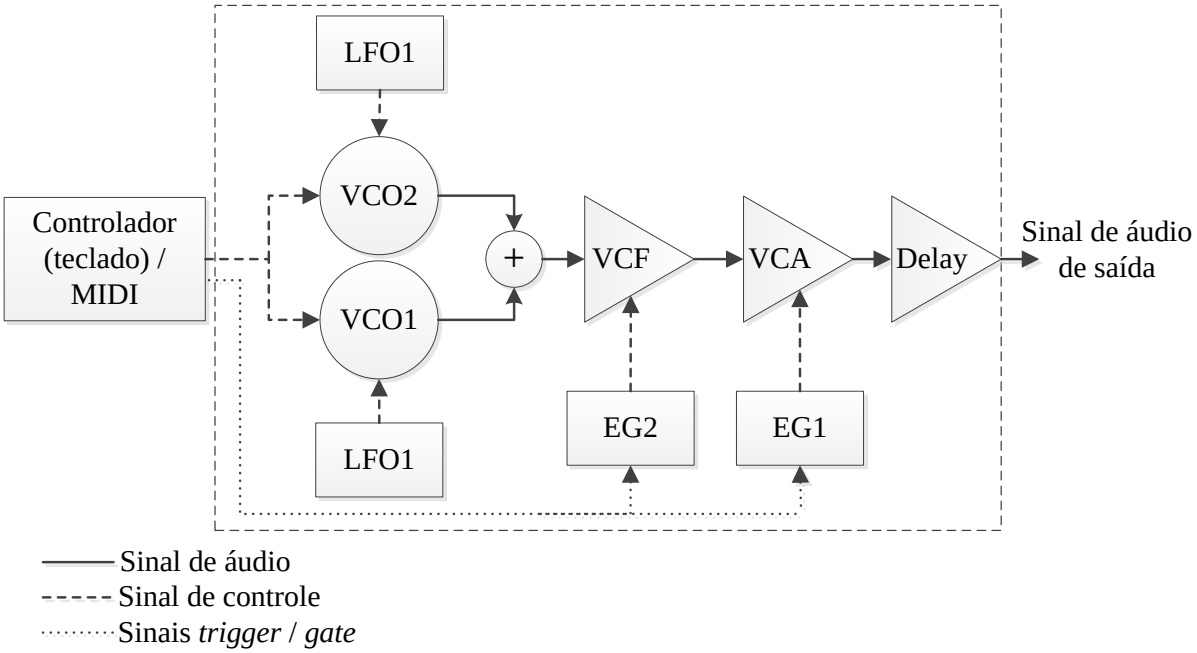
Figura 6-19 – Código para aplicação de um LFO em um oscilador

```
pOscillator01->SetFrequencyMultiplier( 1 << 16 - pOscillatorLFO2->Process() >> 2 );
```

Fonte: elaborada pelo autor

Para finalização deste trabalho e realização dos testes finais, a arquitetura demonstrada na Figura 6-20 foi definida e implementada. Ela possui dois osciladores principais (VCO1 e VCO2), dois osciladores de baixa frequência (LFO1 e LFO2), cada um capaz de modular um dos osciladores, um filtro passa-baixa (VCF), que também pode ter um de seus parâmetros controlados por um gerador de envelopes (EG2), um amplificador de saída que possui o ganho controlador por um gerador de envelopes (EG1), e por fim um módulo de efeito *delay* na saída. O controle de nota a ser executada é realizada pelo módulo controlador, que funciona tanto via comunicação MIDI como pela leitura de um teclado externo.

Figura 6-20 – Arquitetura implementada no sintetizador subtrativo



Fonte: elaborada pelo autor

6.12.1 Chaveamento entre síntese subtrativa e aditiva

Os dois métodos de síntese implementados possuem processamento completamente independente. Embora alguns módulos do software sejam reutilizados, como o *driver* de comunicação com o conversor DA externo, o módulo controlador de notas e o módulo de *delay*, a configuração e chamada dos mesmos é realizada de forma independente em função do tipo de síntese em uso.

Isto nos traz uma possibilidade interessante, que é o uso deste sintetizador para qualquer um dos tipos de síntese, realizando a troca do modo de síntese através de uma chave seletora externa (lida através de um pino de entrada digital) ou de uma diretiva de configuração no código-fonte (neste caso é necessário recompilar o código para trocar o modo de síntese). Uma condição simples de teste contendo a chamada para os métodos de inicialização de cada tipo de síntese realiza esta troca, conforme mostrado na Figura 6-21.

Figura 6-21 – Código de troca entre tipos de síntese

```
if ( true == isHammond )
{
   InitializeHammondMode( );
}
else
{
   InitializeVirtualAnalog( );
}
```

Fonte: elaborada pelo autor

A chamada deste código é realizada no laço principal, sendo a variável booleana *isHammond* atribuída também no laço principal baseado no valor de uma chave seletora. A chamada dos métodos de inicialização é protegida de interrupções e inicializa todos os módulos necessários, interrompendo imediatamente o áudio em execução e chaveando para o novo modo de síntese.

6.13 Compartilhamento do desenvolvimento

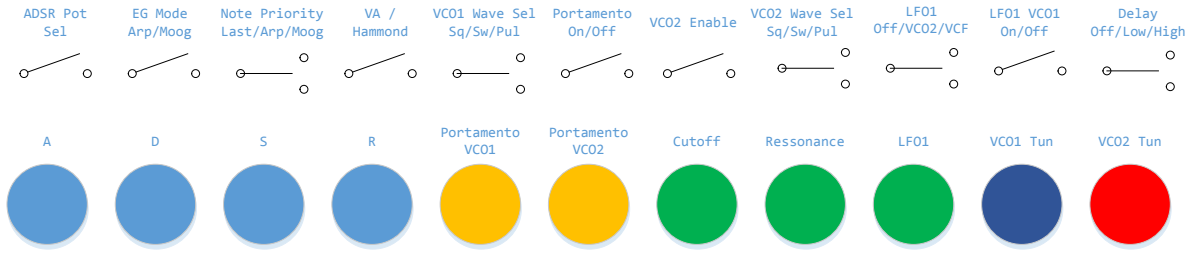
Um dos objetivos deste trabalho é contribuir com o movimento *maker* propondo uma arquitetura e implementação para ser utilizada em plataformas de baixo custo e incentivar o desenvolvimento e inovação na área de engenharia, computação e música (conforme descrito na seção 1.2). A implementação do código-fonte foi toda realizada utilizando as mesmas regras de codificação e cabeçalhos foram adicionados a todas classes e funções para melhor documentação e entendimento das mesmas.

O código-fonte foi disponibilizado no serviço GitHub como um repositório público no endereço *https://github.com/rppirotti/rgsynth*. Neste mesmo endereço foram adicionados arquivos contendo o esquemático utilizado e algumas fotos do protótipo desenvolvido.

7 TESTES E RESULTADOS

Para teste e validação da implementação, realizamos a montagem de alguns componentes eletrônicos externos ao Arduino Due, necessários para o controle de parâmetros analógicos, seleção de características através de chaves seletoras digitais, entrada serial MIDI, conversor DA externo e leitura do teclado externo. A Figura 7-1 mostra a disposição planejada para os potenciômetros e chaves do protótipo, que são utilizados durante a execução musical para alterar parâmetros sonoros.

Figura 7-1 – Disposição de chaves e potenciômetros do protótipo de teste



Fonte: elaborada pelo autor

A função de cada potenciômetro e chave exibido na Figura 7-1 foi planejada para o sintetizador quando utilizando a síntese subtrativa. Após a implementação da síntese aditiva, os mesmos controles foram utilizados para selecionar outros parâmetros. A Tabela 7-1 mostra a função de cada componente de controle em cada um dos modos de síntese.

Tabela 7-1 – Tabela de controles externos do protótipo

Chave / Potenciômetro	**Função em modo sintetizador subtrativo**	**Função em modo sintetizador aditivo**
A	Ajuste do tempo de *attack*	Drawbar 16''
D	Ajuste do tempo de *decay*	Drawbar 5 1/3''
S	Ajuste do nível de *sustain*	Drawbar 8''
R	Ajuste do tempo de *release*	Drawbar 4''
Portamento VCO1	Ajuste do tempo de portamento aplicado ao VCO1	Drawbar 2 2/3''
Portamento VCO2	Ajuste do tempo de portamento aplicado ao VCO2	Drawbar 2''
Cutoff	Ajuste da frequência de corte do filtro	Drawbar 1 3/5''
Resonance	Ajuste da frequência de ressonância do filtro	Drawbar 1 1/3''
LFO1	Ajuste da frequência do LFO1	Drawbar 1''
VCO1 Tun	Ajuste fino da frequência do VCO1	Não utilizado.
VCO2 Tun	Ajuste fino da frequência do VCO2	Não utilizado.
ADSR Pot Sel	Seleciona se o ajuste dos potenciômetros ADSR afetarão o módulo EG aplicado ao VCA ou ao VCF	Habilita a percussão.
EG Mode ARP/Moog	Seleciona o modo do EG do VCA	Não utilizado.
Note Priority	Seleciona o modo de prioridade de nota: modo ARP (mais grave), modo MOOG (mais aguda) ou última nota	Não utilizado.
VA / Hammond	Seleciona o modo de síntese	Não utilizado.
VCO1 Wave Sel	Seleciona o formato de onda do VCO1: quadrada, dente-de-serra ou senoidal	Não utilizado.
Portamento On/Off	Habilita o portamento	Não utilizado.
VCO2 Enable	Habilita o VCO2.	Não utilizado.
VCO2 Wave Sel	Seleciona o formato de onda do VCO2: quadrada, dente-de-serra ou senoidal	Não utilizado.
LFO1 Off/VCO2/VCF	Seleciona a aplicação do LFO1 na modulação do VCO2, VCF (frequência de corte) ou desligado.	Seleciona efeito de *delay* desabilitado, intensidade baixa ou alta
LFO1 VCO1 On/Off	Habilita o LFO1 para modular o VCO1	Não utilizado.
Delay Off/Low/High	Seleciona efeito de *delay* desabilitado, intensidade baixa ou alta	Seleciona efeito *rotary speaker* em velocidade alta, baixa ou desligado.

Fonte: elaborada pelo autor

A Figura 7-2 mostra a montagem do protótipo utilizado para os testes musicais durante o desenvolvimento deste trabalho. Por se tratar de um protótipo, não foram inseridas

identificações em cada componente, até porque durante o desenvolvimento a função de cada componente foi alterada algumas vezes para realização de alguns testes.

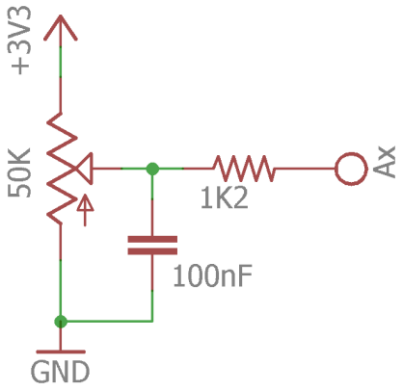
Figura 7-2 – Controles externos do protótipo

Fonte: registrada pelo autor

7.1 Diagramas esquemáticos

Os circuitos montados para interface com o Arduino são simples. Optamos por utilizar resistores em série com todas as portas de entrada e saída para evitar que um erro de configuração ou mesmo de montagem pudesse queimar alguma porta do processador. As entradas analógicas são utilizadas para leitura do valor de cada potenciômetro, e o circuito utilizado em cada uma é mostrado na Figura 7-3.

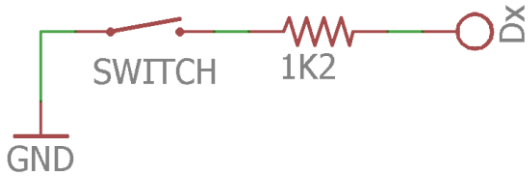
Figura 7-3 – Circuito de entrada analógica para leitura dos potenciômetros



Fonte: elaborada pelo autor

Para as entradas digitais, optamos por habilitar os resistores de *pull-up* internos do processador e utilizar chaves digitais que, quando fechadas, chaveiam o sinal para nível lógico 0. A Figura 7-4 apresenta os componentes utilizados em cada entrada digital.

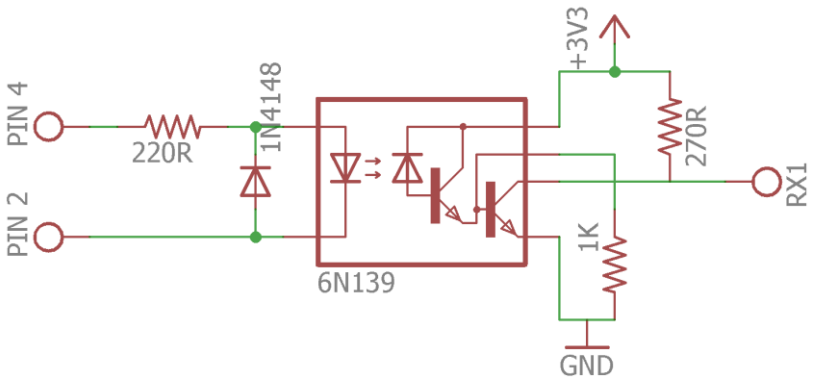
Figura 7-4 – Circuito de entrada digital para leitura das chaves de seleção



Fonte: elaborada pelo autor

Outro circuito necessário foi a entrada MIDI. Como alternativa ao teclado, incluímos uma entrada MIDI no protótipo, assim podemos conectar outro teclado e utilizá-lo para execução das notas. A entrada MIDI possui um opto-acoplador para isolar eletricamente os equipamentos que estão sendo interligados. A Figura 7-5 apresenta o circuito utilizado no nosso protótipo.

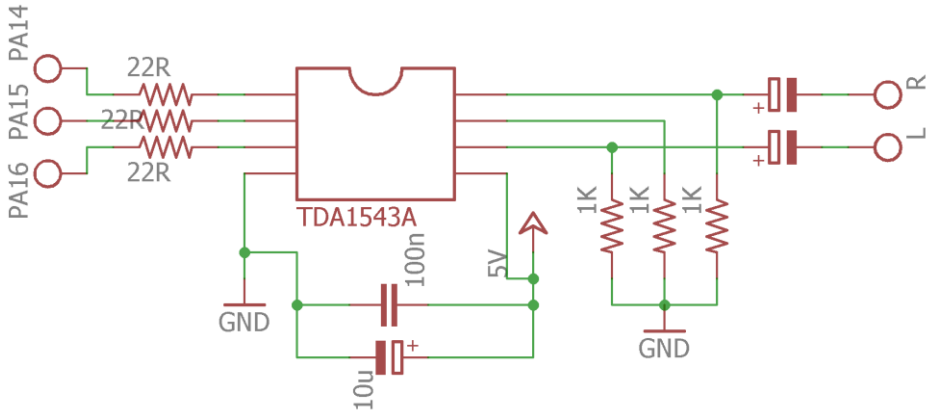
Figura 7-5 – Circuito utilizado na entrada MIDI



Fonte: elaborada pelo autor

O conversor escolhido para realizar a conversão digital-analógico requer poucos componentes externos para o seu funcionamento. A Figura 7-6 apresenta o circuito utilizado. As conexões PA16, PA15 e PA14 são pinos de saída do Arduino Due utilizados para comunicação I2S, enquanto que as saídas L e R são os sinais de saída de áudio..

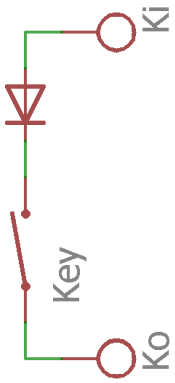
Figura 7-6 – Circuito utilizado para o conversor digital-analógico



Fonte: elaborada pelo autor

O teclado externo utilizado no protótipo possui 49 teclas não-sensitivas e foi desmontado de um antigo teclado eletrônico. O mesmo é composto de uma matriz contendo 8 entradas (colunas) e 7 saídas (linhas), e cada tecla equivale a uma chave liga/desliga. O circuito equivalente de uma entrada e uma saída é mostrado na Figura 7-7.

Figura 7-7 – Circuito interno do teclado externo



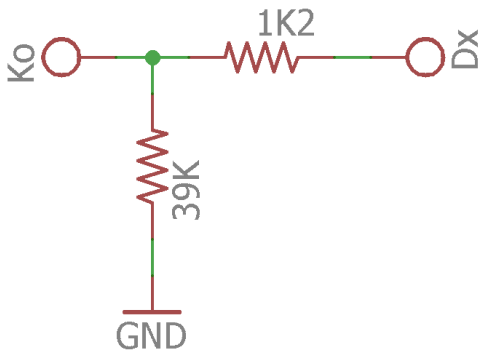
Fonte: elaborada pelo autor

Para realizar a leitura do teclado externo, precisamos utilizar uma saída digital do Arduino Due para alimentar individualmente cada entrada do teclado (representada como Ki na Figura 7-7) e então ler o estado das linhas (uma linha é representada pela saída Ko na Figura 7-7). A Figura 7-8 apresenta a conexão na saída do Arduino, enquanto que a Figura 7-9 apresenta o divisor de tensão utilizado na porta de entrada, para ler cada uma das linhas.

Figura 7-8 – Circuito individual de saída para o teclado externo

Fonte: elaborada pelo autor

Figura 7-9 – Circuito individual de entrada para leitura do teclado externo



Fonte: elaborada pelo autor

7.2 Cálculos com ponto fixo

O erro das operações matemáticas realizadas com números inteiros e ponto fixo foi verificado utilizando o algoritmo PolyBLEP como referência, verificando a diferença entre o valor calculado utilizando ponto flutuante e ponto fixo. Para isto, executamos o algoritmo PolyBLEP implementado com ponto flutuante de dupla precisão (64 bits) e outro com ponto fixo (32 bits) em um computador pessoal, e analisamos o valor equivalente de saída para cada uma das implementações. A Tabela 7-2 apresenta o valor de saída da função em determinados momentos da geração de uma onda quadrada.

Tabela 7-2 – Comparação entre ponto flutuante e ponto fixo

Ponto flutuante	Ponto fixo (normalizado)
0,285877	0,286392
-0,216528	-0,216989
0,004809	0,004807
-0,86611	-0,867954
0,364845	0,365411
-0,156797	-0,157200

Fonte: elaborada pelo autor

7.3 Medição de tempos de execução

A frequência de amostragem utilizada no sintetizador subtrativo é 48 kHz, o que significa um intervalo entre amostras de 20,8us. Para dimensionar e planejar quais módulos e características são possíveis de adicionar considerando esta limitação de tempo, fizemos uma medição do tempo de processamento da função de processamento dos módulos. A medição foi realizada utilizando a função *micros()*, disponível na biblioteca de código do Arduino, que retorna um contador de tempo absoluto com resolução de 1 microsegundo. A Tabela 7-3

apresenta as medições de tempo realizadas para cada chamada de função. O erro de leitura é de ± 1us.

Tabela 7-3 – Medição de tempo de execução

Módulo	Tempo (mín/máx)
Oscillator::process, para onda quadrada e dente-de-serra	3us/5us
Oscillator::process, para onda senoidal	3us/3us
DelayWithFeedback::process	2us/3us
MoogFilter::process	3us/4us
MFFilter::process	2us/3us
EnvelopeGenerator::Process	1us/3us

Fonte: elaborada pelo autor

Com base na medição de tempos realizada, adicionamos dois osciladores, um LFO, um módulo de efeito *delay* e um filtro. Dois módulos geradores de envelope também foram adicionados, porém seu processamento principal é considerada uma tarefa periódica (e não de tempo-crítico), não contabilizando então este tempo para o grupo de tarefas de tempo-crítico.

7.4 Sinais de saída

Para medição dos sinais de saída, optamos por medir o sinal equivalente à nota Dó 6, que é um sinal de 2093 Hz, utilizando as formas de onda quadrada e dente-de-serra. Além disto, optamos por realizar a medição utilizando diferentes métodos, descritos na Tabela 7-4.

Tabela 7-4 – Configurações utilizadas para medição do sinal de saída

	PolyBLEP	Instrumento de medição	Filtro passa-baixa com frequência de corte em 100kHz
Medição A	Desabilitado	Osciloscópio digital com largura de banda de medição de 350 MHz	Não
Medição B	Habilitado	Osciloscópio digital com largura de banda de medição de 350 MHz	Não
Medição C	Desabilitado	Osciloscópio digital com largura de banda de medição de 350 MHz	Sim
Medição D	Habilitado	Osciloscópio digital com largura de banda de medição de 350 MHz	Sim
Medição E	Desabilitado	Placa de som configurada para 16 bits/48kHz	Não
Medição F	Habilitado	Placa de som configurada para 16 bits/48kHz	Não

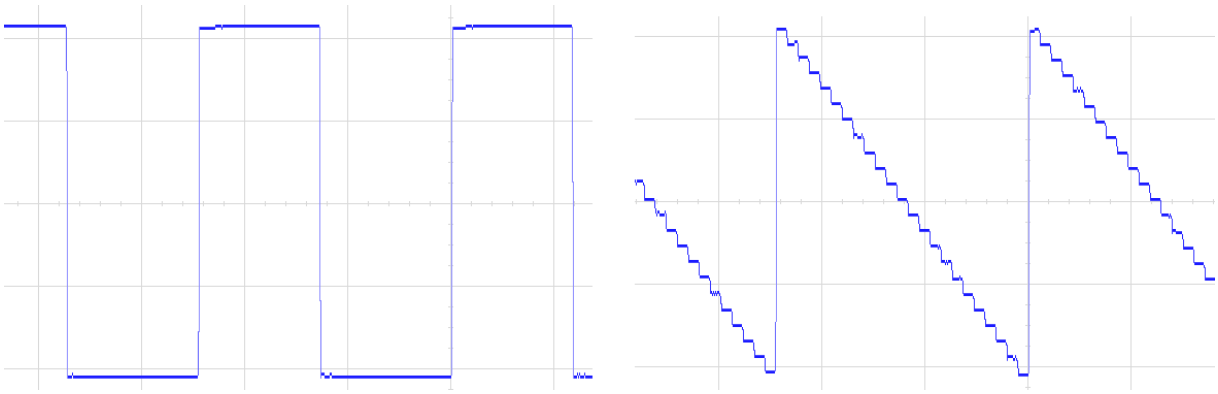
Fonte: elaborada pelo autor

Com a medição realizada apenas através de um osciloscópio, captamos sinais de frequências muito acima das frequências sonoras e reproduzíveis por um sistema de som. Ao

inserir um filtro ou realizar a medição utilizando uma placa de som, simulamos o sinal aproximado que seria reproduzido por um sistema de som.

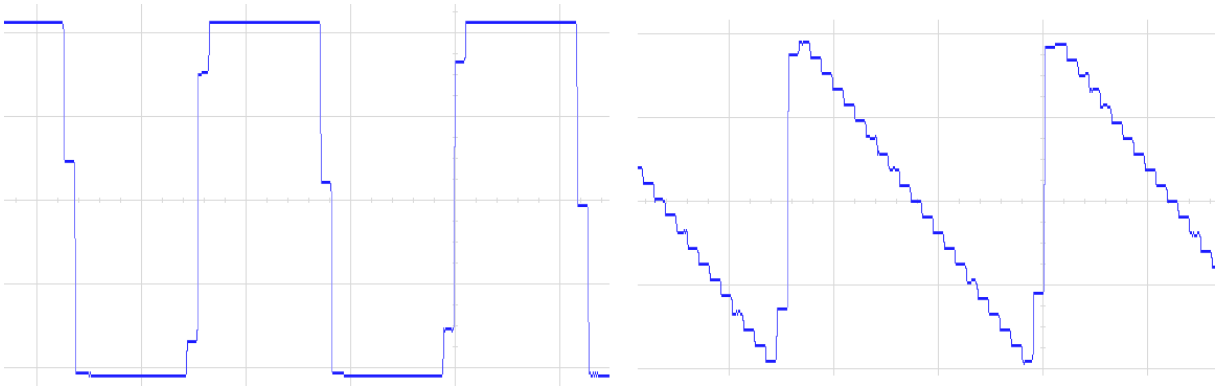
A Figura 7-10, Figura 7-11, Figura 7-12, Figura 7-13, Figura 7-14, e Figura 7-15 apresentam as medições A, B, C, D, E e F, demonstrando uma onda quadrada ao lado esquerdo e uma onda dente-de-serra no lado direito.

Figura 7-10 – Medição A



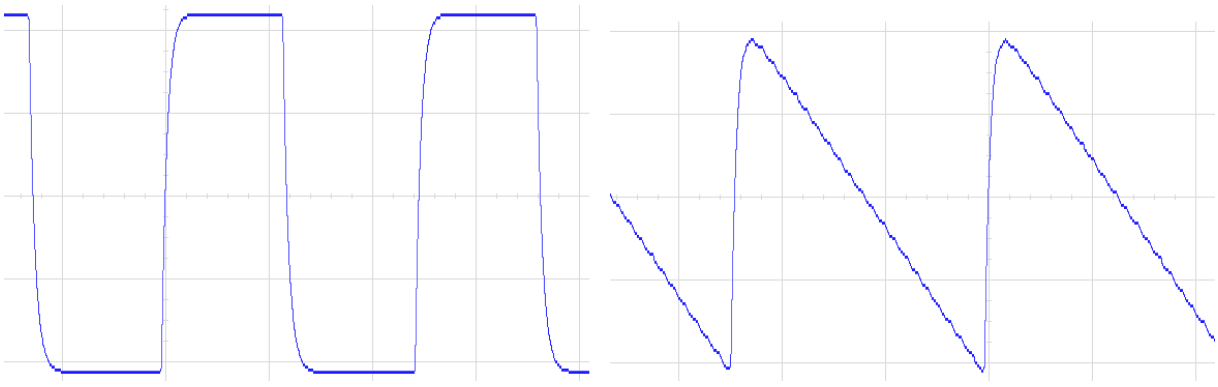
Fonte: elaborada pelo autor

Figura 7-11 – Medição B



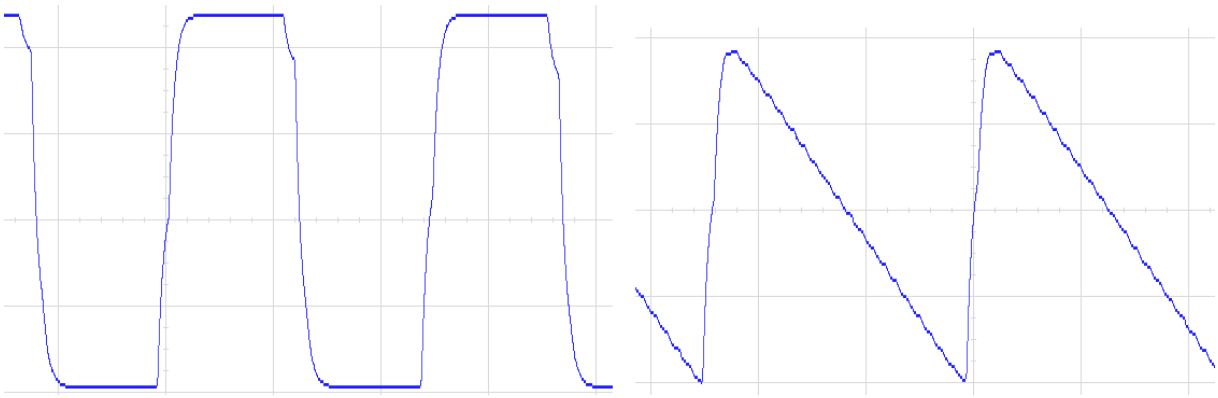
Fonte: elaborada pelo autor

Figura 7-12 – Medição C



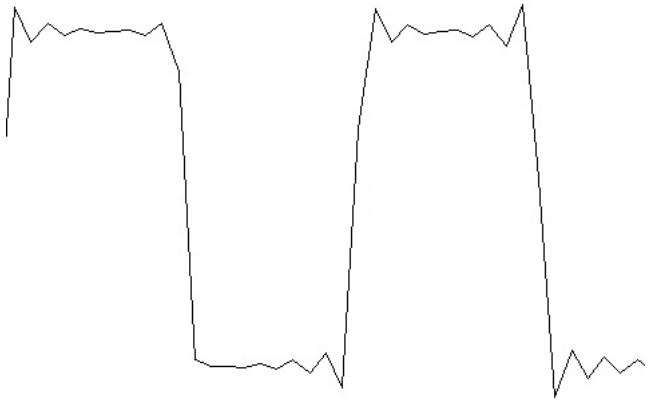
Fonte: elaborada pelo autor

Figura 7-13 – Medição D



Fonte: elaborada pelo autor

Figura 7-14 – Medição E



Fonte: elaborada pelo autor

Figura 7-15 – Medição F

Fonte: elaborada pelo autor

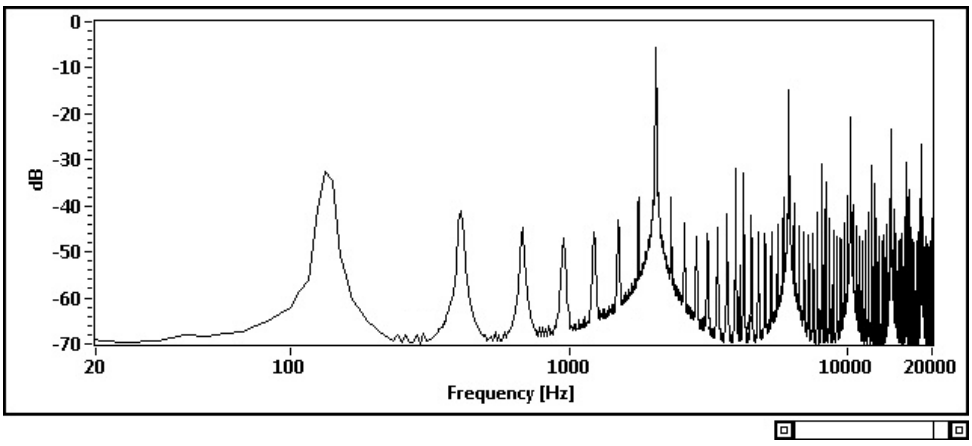
7.5 Verificação de *aliasing*

O algoritmo de geração de onda com limitação de banda PolyBLEP foi escolhido baseado nas análises apresentadas por Valimaki e Huovilainen (2007). Este algoritmo nos pareceu apropriado para a implementação em uma plataforma de processamento limitado

devido ao reduzido número de cálculos matemáticos necessários, porém bom resultado final. Após a implementação, concluímos que o mesmo é, de fato, uma alternativa apropriada para esta proposta. O tempo necessário para processamento do algoritmo ficou em torno de 1us, o que é rápido suficiente.

Para demonstrar o resultado da aplicação do PolyBLEP, apresentamos abaixo a análise do espectro de frequências de duas medições da mesma nota musical sendo executada, em um dos casos sem o algoritmo PolyBLEP (apenas o método trivial, explicado em 5.2.1), e no outro caso com a adição do PolyBLEP. A Figura 7-16 apresenta a análise de espectro da nota Dó, frequência 2093 Hz, formato de onda quadrada, gerado apenas com o oscilador trivial.

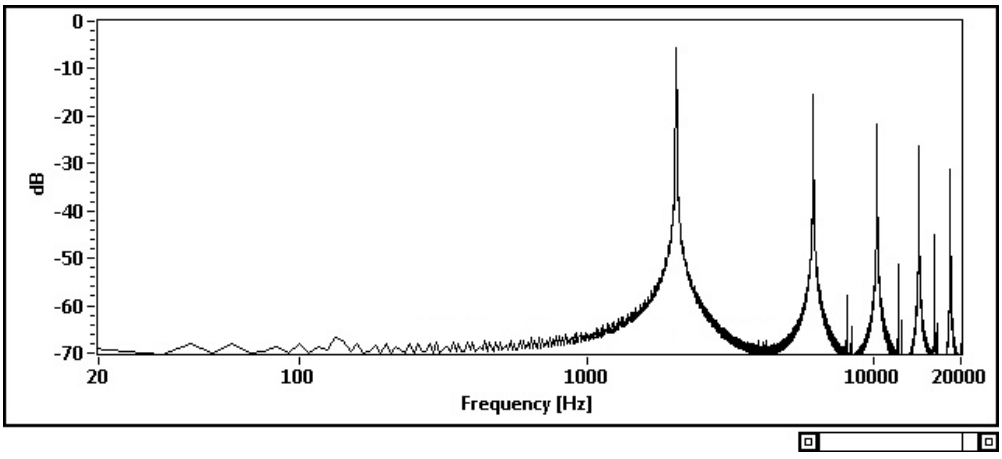
Figura 7-16 – Espectro de frequências com oscilador trivial, sem PolyBLEP



Fonte: elaborada pelo autor

A Figura 7-17 apresenta a análise de espectro da nota Dó, frequência 2093 Hz, formato de onda quadrada, gerado com oscilador trivial e incluindo o algoritmo PolyBLEP.

Figura 7-17 - Espectro de frequências com oscilador trivial e incluindo o algoritmo PolyBLEP



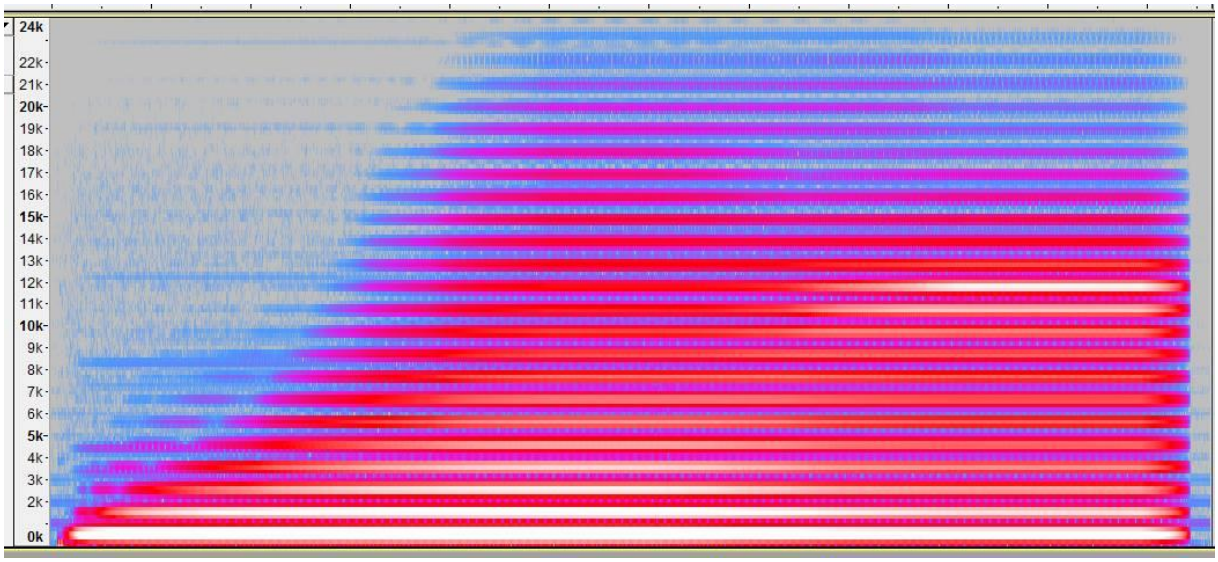
Fonte: elaborada pelo autor

Comparando a Figura 7-16 e a Figura 7-17, é possível observar a remoção do *aliasing* quase que por completo através do algoritmo PolyBLEP. Apenas três componentes de frequência “indevidos” foram observados após a aplicação do algoritmo, sendo que estes três componentes possuem uma amplitude pequena e em uma faixa alta do espectro de frequências, o que reduz a influência das mesmas na sonoridade principal desejada.

7.6 Espectrograma do filtro

Um dos filtros implementados foi baseado na implementação de Kellet (sem data), que é um filtro passa-baixa de quarta ordem, com ressonância e atenuação de 24 dB por oitava. Este filtro é baseado na análise feita por Stilson e Smith (1996b) do filtro projetado por Robert Moog para os sintetizadores Moog. A Figura 7-18 demonstra o espectrograma deste filtro adaptado para funcionar com ponto fixo.

Figura 7-18 – Espectrograma do filtro passa-baixa



Fonte: elaborada pelo autor

Analisando o espectrograma, percebe-se que o maior aumento da ressonância (linhas brancas ao lado direito do gráfico) se dá em uma largura de faixa limitada, em torno de 9 kHz até 12 kHz. Esta largura de faixa é menor do que a observada em outros instrumentos digitais profissionais utilizados como comparação, embora já acrescente ao filtro uma sonoridade característica e a possibilidade de uso deste parâmetro para criação de sons novos.

8 CONCLUSÃO

Este trabalho apresentou uma arquitetura e implementação de um sintetizador subtrativo digital de alta qualidade implementado na plataforma de baixo custo Arduino Due em conjunto com a implementação de um órgão utilizando síntese aditiva. A implementação completa é aberta, pública e voltada para o uso em plataformas de baixo custo. Apesar da implementação ter sido testada em um Arduino Due, acreditamos que ela também possa ser aplicada em qualquer outra plataforma similar contendo um processador ARM Cortex-M3, ou mesmo processadores similares e até mais simples, como por exemplo um ARM Cortex-M0 (se uma velocidade de *clock* inferior à utilizada na nossa implementação for necessária, é possível remover algumas características ou reduzir a taxa de amostragem). O tamanho final de código utilizou 8% do espaço total disponível no Arduino Due.

Os algoritmos implementados com ponto fixo atingiram nossas expectativas, apesar de tornar o código-fonte um pouco mais difícil de entender e tornar a escrita de código mais suscetível a erros. A arquitetura de 32 bits do processador contribuiu para isto, pois nos permitiu utilizar na maior parte do código um padrão de 16 bits para parte inteira e 16 bits para parte fracionária, o que nos garantiu uma boa resolução e operações matemáticas rápidas.

O tempo necessário para processamento dos módulos principais nos permitiu adicionar dois osciladores principais, um LFO, dois geradores de envelope, um filtro, um módulo de efeito, além de permitir o uso de interface MIDI ou teclado externo, o que consideramos muito bom. Reduzindo levemente a taxa de amostragem para 44,1 kHz, ainda seria possível adicionar mais algum módulo. Em relação à geração das formas de onda sem *aliasing*, que era um dos desafios a serem vencidos, o algoritmo PolyBLEP implementado com ponto fixo apresentou resultados satisfatórios, removendo a maior parte do *aliasing* audível conforme esperado com base no algoritmo original.

Uma possibilidade de implementação e teste futuro é o uso do algoritmo de busca por tabela de onda (utilizado para forma de onda seno) em conjunto com o PolyBLEP e armazenar na tabela uma forma de onda obtida diretamente de um sintetizador analógico, para tentar reproduzir detalhes característicos de cada equipamento. Além disto, sugerimos adicionar um filtro passa-baixa na saída de áudio do instrumento (com frequência de corte em torno de 20 kHz), para atenuar frequências agudas e adicionar um caráter mais suave ao som gerado.

O uso de orientação a objetos para desenvolvimento do código permitiu a inclusão ou modificação de características de acordo com a necessidade. Citando um exemplo, para uma determinada execução musical, adicionamos portamento a apenas um dos osciladores,

mantendo o outro sem portamento. Isto trouxe uma sonoridade que achamos interessante, porém não seria possível fazer isto em um sintetizador analógico convencional, no qual o portamento está associado ao teclado. Isto mostra que é possível, com esta abordagem de desenvolvimento, buscar novas possibilidades sonoras e timbres característicos.

Em relação à implementação do filtro, nosso objetivo não era o desenvolvimento de um novo filtro ou estudo detalhado de técnicas de projeto de filtros digitais. Assim, buscamos reutilizar o código de filtros disponíveis na comunidade de *software*, e encontramos dois filtros que servem como ponto de partida para o uso do sintetizador e trazem grandes possibilidades para a criação sonora (um deles sendo uma simplificação do modelo de um filtro utilizado no sintetizador Moog). Porém acreditamos que este é um ponto a ser melhorado neste trabalho, e propomos para projetos futuros o desenvolvimento e teste de outros tipos e implementações de filtros que possam replicar melhor a sonoridade de sintetizadores analógicos clássicos.

Em relação à síntese aditiva, utilizamos como referência a implementação de Johann (2015), conseguindo adicionar o suporte à polifonia completa para 61 notas, configuração do timbre através do conceito de *drawbars* e efeitos simples que tentam reproduzir a sonoridade das caixas Leslie. A troca entre os métodos de síntese pode ser feita em tempo real, através de uma chave de seleção externa ao microprocessador, sendo possível assim ter dois métodos clássicos de síntese na mesma plataforma. Para trabalhos futuros relacionados à implementação da síntese aditiva, sugerimos a implementação de um módulo de efeito que reproduza mais fielmente o efeito causado pelas caixas Leslie, também chamado de *Rotary Speaker effect*. A adição de um efeito mais fiel ao real certamente melhorará bastante a sonoridade neste método de síntese.

Folle (2015) implementou outro tipo de síntese musical muito utilizada, a síntese FM (*frequency modulation*), também utilizando o Arduino Due. Embora não tenha sido testado neste trabalho, acreditamos que este método também possa ser adicionado à esta implementação, tendo então três tipos de síntese possíveis de serem selecionados em tempo real (subtrativa, aditiva e FM).

Um protótipo foi montado contendo potenciômetros e chaves de seleção para validar a variação de parâmetros durante o uso do sintetizador. Em mais de uma oportunidade foi necessário substituir componentes deste protótipo, em especial potenciômetros, pois começaram a apresentar falhas após certo tempo de uso. Com base nestes acontecimentos, recomendamos um certo cuidado na compra de componentes para este uso, buscando sempre componentes de qualidade pois são componentes essenciais para que a experiência musical seja completa.

O desenvolvimento voltado à possibilidade de simulação em um computador foi de extrema importância para o desenvolvimento do *software*, facilitando e reduzindo o tempo de depuração necessário para verificação de erros no código, tornando possível a simulação dos módulos de forma independente e verificação da saída. Além disto, facilitou a verificação de possíveis erros de cálculo e precisão ao utilizar ponto fixo.

Por outro lado, é necessário também a verificação dos resultados utilizando a plataforma final, e para isto recomendamos o uso de alguns *softwares* para computador que, através do sinal captado pela placa de som, exibe o formato de onda e espectro de frequência do sinal, sendo uma boa ferramenta para verificação de problemas.

Como propostas de desenvolvimento futuro, além de algumas já citadas aqui nesta conclusão, sugerimos a adição de mais algumas características existentes em sintetizadores analógicos, tais como:

- *Keyboard tracking*, conforme explicado na seção 3.3.3;
- Outras formas de onda, como onda triangular, *sample-and-hold* e ruído;
- Sequenciador;
- Polifonia de duas vozes;
- Outros módulos de efeito, como *chorus*;

E por fim, para contribuir com o movimento *maker* e incentivar o desenvolvimento de instrumentos musicais eletrônicos em geral, todo o desenvolvimento realizado neste trabalho está disponível em um repositório público, contendo o código-fonte completo, incluindo documentação gerada através de ferramentas automáticas, o esquemático utilizado e algumas fotos do protótipo utilizado nos testes.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. Getting Started with Arduino and Genuino products. [s.d.]. Disponível em: < https://www.arduino.cc/en/Guide/HomePage>. Acesso em: setembro de 2015.

ARDUINODUE.jpg. Formato: JPG. Arduino, [2012?]. Disponível em: <https://www.arduino.cc/en/uploads/Main/ArduinoDUE.jpg>. Acesso em: outubro de 2016.

ATMEL Corporation. **SAM3X / SAM3A Series datasheet**. 2015. Disponível em: <http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf>. Acesso em: setembro de 2015.

BATCHELOR, P., WIGNALL, TREV. Beagle Pi: An Introductory Guide to Csound on the BeagleBone and the Raspberry Pi, as well other Linux-powered tinyware. 2013. **CSOUND Journal**. Issue 18. Disponível em: <http://www.csounds.com/journal/issue18/beagle_pi.html>. Acesso em: março de 2017.

BARRASS, T. **Mozzi Sound Synthesis Library for Arduino**. [2012?]. Disponível em: <http://sensorium.github.io/Mozzi/>. Acesso em: setembro de 2015.

BEAGLEBOARD.ORG Foundation. **Getting Started with Beagle Bone**. [s.d.]. Disponível em: < https://beagleboard.org/getting-started>. Acesso em: setembro de 2015.

BERNARDINIS, F. De et al. An Efficient VLSI Architecture for Real-Time Additive Synthesis of Musical Signals. **IEEE Transactions on very large scale integration (VLSI) systems**, Durham, USA, Vol. 7, N. 1, March 1999, p. 105–110. 1999.

BIDDULPH, L., ZIEMBICKI, J. **AVRSynth**. [2000?]. Disponível em: <http://www.elby-designs.com/contents/en-us/d5.html>. Acesso em: julho de 2016.

CABRERA, A.; HEINTZ, J.; HOUDORF, B. **Csound: Additive Synthesis**. [2016?]. Disponível em: <http://write.flossmanuals.net/csound/a-additive-synthesis/>. Acesso em: agosto de 2016.

CARLOS, W. **Official Wendy Carlos Online Information Source**. Disponível em: <http://www.wendycarlos.com/+sobox.html>. Acesso em: julho de 2016.

CHEN, J. Return to the tradition: DIY design. **2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design**, Wenzhou, China, p. 1774–1777, Nov. 2009. doi: 10.1109/CAIDCD.2009.5375278. Disponível em: < http://ieeexplore.ieee.org/document/5375278/>.

COULTER, D. **Digital Audio Processing**. Lawrence, KS: R&D Books, Group West, 2000.

CROMBIE, D. **The Complete Synthesizer: A Comprehensive Guide**. [s.l.] Omnibus Press, 1982.

DALTON, M. A.; DESJARDINS, A.; WAKKARY, R. From DIY tutorials to DIY recipes. **Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems**, Toronto, Canada, p. 1405–1410, Abril 2014. doi:

10.1145/2559206.2581238. Disponível em: < http://dl.acm.org/citation.cfm?id=2581238&dl=ACM&coll=DL&CFID=869503526&CFTOKEN=64089854>

DELSAUCE. **ArduinoDueHiFi**. [2013?]. Disponível em: <https://github.com/delsauce/>. Acesso em: abril de 2006.

DEVAHARI. **The Complete Guide To Synthesizers**. [s.l.] Prentice-Hall, Inc., 1982.

DINIZ, P. S. R.; SILVA, E. A. B. DA; NETTO, S. L. **Processamento Digital de Sinais**. Porto Alegre. Bookman, 2004.

DODGE, C.; JERSE, T. A. **Computer Music: synthesis, composition and performance.** 2nd edition, New York, Schirmer Books. 1997.

FELDMAN, L. Here's How TechShop Is Manufacturing Startups And Entrepreneurs. **Forbes**. 2015. Disponível em: <http://www.forbes.com/sites/forbestreptalks/2015/07/30/how-mark-hatch-and-techshop-are-energizing-american-manufacturing/#3d36c1393e93>. Acesso em: junho de 2016.

FOLLE, L. **Implementação de Síntese FM na Plataforma Arduino Due**. 2015. 37f. Monografia (Bacharelado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, 2015.

FINKE, M. **Martin Finke's Blog**: Music & Programming. [2013?]. Disponível em: <http://www.martin-finke.de/blog/articles/>. Acesso em: junho de 2015.

GIBB, A. **Building Open Source Hardware**: DIY Manufacturing for Hackers and Makers. Crawfordsville, Indiana. Addison-Wesley Professional, 2014. Disponível em: < https://www.safaribooksonline.com/library/view/building-open-source/9780133373912/>.

GOMES, H. V. **Metodologia de Projeto de Software Embarcado Voltada ao Teste**. 2010. 104 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2010.

HAMMOND-drawbars.svg. Formato: SVG. Wikimedia Commons, [2007?]. Disponível em: <https://commons.wikimedia.org/wiki/File:Hammond-drawbars.svg>. Acesso em: setembro de 2016.

HATCH, M. **The Maker Movement Manifesto**. [s.l.], p. 1–31, 2014. Disponível em : < http://www.techshop.ws/images/0071821139%20Maker%20Movement%20Manifesto%20Sample%20Chapter.pdf>. Acesso em: maio de 2016.

HEWITT, E.; HEWITT, R. E. The Gibbs-Wilbraham Phenomenon: An Episode in Fourier Analysis. **Archive for History of Exact Sciences**. v. 21, n. 2, p. 129-160, 1979. doi:10.1007/BF00330404

HUOVILAINEN, A. **Design of a Scalable Polyphony-MIDI Synthesizer for a Low Cost DSP**. 2010. 66 f. Dissertação (Master of Science in Technology). Faculty of Electronics,

Communication and Automation, Aalto University School of Science And Technology, Espoo, Finland, 2010.

JAMIESON, P.; HERDTNER, J. More missing the Boat - Arduino, Raspberry Pi, and small prototyping boards and engineering education needs them. **2015 IEEE Frontiers in Education Conference (FIE)**, El Paso, TX, USA, p. 1-6, Oct. 2015. doi: 10.1109/FIE.2015.7344259

JENKINS, M. **Analog Synthesizers**. 1a ed. Oxford, UK. Focal Press, 2007. Disponível em: < https://www.safaribooksonline.com/library/view/analog-synthesizers/9780240520728/>.

JOHANN, M. An additive synthesis organ with full polyphony on Arduino Due. **VI Ubimus**. Växjö, Sweden. 2015. Disponível em: <http://lammax.lnu.se/ubimus/wp-content/uploads/sites/5/2015/03/ubimus6_submission_1.pdf>. Acesso em: setembro de 2015.

JORDAN, S.; LANDE, M. Should makers be the engineers of the future? **2013 IEEE Frontiers in Education Conference (FIE)**, Oklahoma, USA, p. 815–817, Oct. 2013. doi: 10.1109/FIE.2013.6684939

KELLETT, P. **Moog VCF, variation 1, 24dB resonant low pass**. [s.d.]. Disponível em: <http://www.musicdsp.org/archive.php?classid=3#25>. Acesso em: março de 2016.

KISENWETHER, E. C.; TROXELL, W. C. Performance Analysis of the Numerically Controlled Oscillator. **40th Annual Symposium on Frequency Control**. Philadelphia, USA, p. 373-378, May 1986. doi: 10.1109/FREQ.1986.200971

KUZNETSOV, S.; PAULOS, E. Rise of the expert amateur: DIY projects, communities, and cultures. **6th Nordic Conference on Human-Computer Interaction: Extending Boundaries**. Reykjavik, Iceland, p. 295–304, Oct. 2010. doi: 10.1145/1868914.1868950

LINDTNER, S.; BARDZELL, S.; BARDZELL, J. Reconstituting the Utopian Vision of Making. **2016 CHI Conference on Human Factors in Computing Systems**. Santa Clara, USA, p. 1390–1402, May 2016. doi: 10.1145/2858036.2858506

LIU, L.; JEREMIA, B.; FRIEDRICH, F. A Low Power Configurable SoC for Simulating Delay-based audio effects. **2012 International Conference on Reconfigurable Computing and FPGAs**. Cancun, Mexico, Dec. 2012. doi: 10.1109/ReConFig.2012.6416759

MAKER Faire: A Bit of History. [s.d.]. Disponível em: <http://makerfaire.com/makerfairehistory/>. Acesso em: setembro de 2016.

MCKAY, G. **DiY Culture: Party & Protest in Nineties Britain**. [s.l.] Verso, 1998. Disponível em: < https://books.google.com.br/books/about/DiY_Culture.html?id=KaP-bAMAYlYC>. Acesso em: maio de 2016.

ORSINI, L. **Arduino's Massimo Banzi**: How We Helped Make The Maker Movement. 2014. Disponível em: <http://readwrite.com/2014/05/12/arduino-massimo-banzi-diy-electronics-hardware-hacking-builders/>. Acesso em: setembro de 2016.

PEKONEN, J. **Computationally Efficient Music Synthesis – Methods and Sound Design.** 2007. 82f. Dissertação (Master of Science in Technology). Department of Electrical and Communications Engineering, Helsinki University of Technology, Espoo, Finland, 2007.

PEKONEN, J.; PIHLAJAMÄKI, T.; VÄLIMÄKI, V. Computationally Efficient Hammond Organ Synthesis. **Proc. of the 14th Int. Conference on Digital Audio Effects (DAFx-11)**, Paris, France, September 19-23, 2011, p. 19–22, 2011.

PIRKLE, W. **Designing Software Synthesizer Plug-Ins in C++.** Burlington, MA. Focal Press, 2014. Disponível em: < https://www.safaribooksonline.com/library/view/designing-software-synthesizer/9781138787070/>

RASPBERRY Pi. [s.d.]. Disponível em <https://www.raspberrypi.org/>. Acesso em: dezembro de 2015.

ROADS, C. **The Computer Music Tutorial**. Cambridge, MIT Press. 1996.

RUSS, M. **Sound Synthesis and Sampling**. 2. ed. Oxford, UK. Focal Press, 2004. Disponível em: < https://www.safaribooksonline.com/library/view/sound-synthesis-and/9780240516929/>.

STILSON, T.; SMITH, J. Alias-free digital synthesis of classic analog waveforms. **1996 International Computer Music Conference**, Hong Kong, China, 1996. Disponível em: <https://ccrma.stanford.edu/~stilti/papers/blit.pdf>

STILSON, T.; SMITH, J. Analyzing the Moog VCF with Considerations for Digital Implementation. **1996 International Computer Music Conference**, Hong Kong, China, 1996. Disponível em: < https://ccrma.stanford.edu/~stilti/papers/moogvcf.pdf>

STONE, Z. **The Maker Movement Is Taking Over America**: Here's How. 2015. Disponível em: <http://thehustle.co/the-diy-maker-movement-survives-by-doing-the-opposite-of-whats-smart>. Acesso em: setembro de 2016.

TEENSY. [s.d.]. Teensy USB Development Board. Disponível em < https://www.pjrc.com/teensy/index.html>. Acesso em: dezembro de 2015.

USPTO. Thaddeus Cahill. **Art of and Apparatus for generating and distributing music electrically**. EUA n. US000580035, 04 fev. 1896, 6 abr. 1897.

USPTO. Elisha Gray. **Electric Telegraph for Transmitting Musical Tones.** EUA n. US166095, 27 jul. 1875.

USPTO. Laurens Hammond. **Electrical Musical Instrument.** EUA n. US1956350 A, 19 jan. 1934, 24 abr. 1934.

USPTO. Robert Moog. **Electronic high-pass and low pass filters employing the base to emitter diode resistance of bi-polar transistors**. EUA n. US3475623, 10 oct. 1966, 28 oct. 1969.

USPTO. Alan Pearlman, Timothy Gillette. **Dynamic filter**. EUA n. US4011466 A, 10 may 1976, 8 mar. 1977.

USPTO. Leo Theremin. **Method of and apparatus for the generation of sounds**. EUA n. US1661058 A, 8 dec. 1924, 28 feb. 1928.

UPTON, E. **Ten Millionth Raspberry Pi, And a New Kit**. 2016. Disponível em: <https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/>. Acesso em: setembro de 2016.

URBAN, O. Short Overview of Methods of Sound Synthesis. **The 32nd International Acoustical Conference - EAA SYMPOSIUM "ACOUSTICS BANSKÁ ŠTIAVNICA 2002"**. Slovakia, p. 237–240, September 2002.

USSON, Y. **Yusynth**. Disponível em: <http://home.yusynth.net/>. Acesso em: fevereiro de 2016.

VÄLIMÄKI, V.; HUOVILAINEN, A. Antialiasing oscillators in subtractive synthesis. **IEEE Signal Processing Magazine**, Maryland, USA, v. 24, n. 2, p. 116–125, 2007. doi: 10.1109/MSP.2007.323276

YATES, R. **Fixed-Point Arithmetic : An Introduction**. 2009. Disponível em: < http://cs.smith.edu/dftwiki/images/1/16/FixedPointAnIntroduction.pdf>

ANEXO A <CÁLCULO DE INCREMENTO DE FASE>

Nota	Freq. (Hz)	Incremento de Fase	Parte inteira	Parte fracionária	Conversão para ponto fixo	Parte fracionária após conversão
C1	32,70	0,69760000	0,0	0,69760000	45717,9	0,697586060
C#	34,65	0,73920000	0,0	0,73920000	48444,2	0,739196777
D1	36,71	0,78314667	0,0	0,78314667	51324,3	0,783142090
D#	38,89	0,82965333	0,0	0,82965333	54372,2	0,829650879
E1	41,20	0,87893333	0,0	0,87893333	57601,8	0,878921509
F1	43,65	0,93120000	0,0	0,93120000	61027,1	0,931198120
F#	46,25	0,98666667	0,0	0,98666667	64662,2	0,986663818
G1	49,00	1,04533333	1,0	0,04533333	2971,0	0,045318604
G#	51,91	1,10741333	1,0	0,10741333	7039,4	0,107406616
A1	55,00	1,17333333	1,0	0,17333333	11359,6	0,173324585
A#	58,27	1,24309333	1,0	0,24309333	15931,4	0,243087769
B1	61,74	1,31712000	1,0	0,31712000	20782,8	0,317108154
C2	65,41	1,39541333	1,0	0,39541333	25913,8	0,395401001
C#	69,30	1,47840000	1,0	0,47840000	31352,4	0,478393555
D2	73,42	1,56629333	1,0	0,56629333	37112,6	0,566284180
D#	77,78	1,65930667	1,0	0,65930667	43208,3	0,659301758
E2	82,41	1,75808000	1,0	0,75808000	49681,5	0,758071899
F2	87,31	1,86261333	1,0	0,86261333	56532,2	0,862609863
F#	92,50	1,97333333	1,0	0,97333333	63788,4	0,973327637
G2	98,00	2,09066667	2,0	0,09066667	5941,9	0,090652466
G#	103,83	2,21504000	2,0	0,21504000	14092,9	0,215026855
A2	110,00	2,34666667	2,0	0,34666667	22719,1	0,346664429
A#	116,54	2,48618667	2,0	0,48618667	31862,7	0,486175537
B2	123,47	2,63402667	2,0	0,63402667	41551,6	0,634017944
C3	130,81	2,79061333	2,0	0,79061333	51813,6	0,790603638
C#	138,59	2,95658667	2,0	0,95658667	62690,9	0,956573486
D3	146,83	3,13237333	3,0	0,13237333	8675,2	0,132369995
D#	155,56	3,31861333	3,0	0,31861333	20880,6	0,318603516
E3	164,81	3,51594667	3,0	0,51594667	33813,1	0,515945435
F3	174,61	3,72501333	3,0	0,72501333	47514,5	0,725006104
F#	185,00	3,94666667	3,0	0,94666667	62040,7	0,946655273
G3	196,00	4,18133333	4,0	0,18133333	11883,9	0,181320190
G#	207,65	4,42986667	4,0	0,42986667	28171,7	0,429855347
A3	220,00	4,69333333	4,0	0,69333333	45438,3	0,693328857
A#	233,08	4,97237333	4,0	0,97237333	63725,5	0,972366333
B3	246,94	5,26805333	5,0	0,26805333	17567,1	0,268051147
C4	261,63	5,58144000	5,0	0,58144000	38105,3	0,581436157
C#	277,18	5,91317333	5,0	0,91317333	59845,7	0,913162231

D4	293,66	6,26474667	6,0	0,26474667	17350,4	0,264739990
D#	311,13	6,63744000	6,0	0,63744000	41775,3	0,637435913
E4	329,63	7,03210667	7,0	0,03210667	2104,1	0,032104492
F4	349,23	7,45024000	7,0	0,45024000	29506,9	0,450225830
F#	369,99	7,89312000	7,0	0,89312000	58531,5	0,893112183
G4	391,99	8,36245333	8,0	0,36245333	23753,7	0,362442017
G#	415,30	8,85973333	8,0	0,85973333	56343,5	0,859725952
A4	440,00	9,38666667	9,0	0,38666667	25340,6	0,386657715
A#	466,16	9,94474667	9,0	0,94474667	61914,9	0,944732666
B4	493,88	10,53610667	10,0	0,53610667	35134,3	0,536102295
C5	523,25	11,16266667	11,0	0,16266667	10660,5	0,162658691
C#	554,37	11,82656000	11,0	0,82656000	54169,4	0,826553345
D5	587,33	12,52970667	12,0	0,52970667	34714,9	0,529693604
D#	622,25	13,27466667	13,0	0,27466667	18000,6	0,274658203
E5	659,26	14,06421333	14,0	0,06421333	4208,3	0,064208984
F5	698,46	14,90048000	14,0	0,90048000	59013,9	0,900466919
F#	739,99	15,78645333	15,0	0,78645333	51541,0	0,786453247
G5	783,99	16,72512000	16,0	0,72512000	47521,5	0,725112915
G#	830,61	17,71968000	17,0	0,71968000	47164,9	0,719665527
A5	880,00	18,77333333	18,0	0,77333333	50681,2	0,773330688
A#	932,33	19,88970667	19,0	0,88970667	58307,8	0,889694214
B5	987,77	21,07242667	21,0	0,07242667	4746,6	0,072418213
C6	1046,50	22,32533333	22,0	0,32533333	21321,0	0,325332642
C#	1108,73	23,65290667	23,0	0,65290667	42788,9	0,652893066
D6	1174,66	25,05941333	25,0	0,05941333	3893,7	0,059402466
D#	1244,51	26,54954667	26,0	0,54954667	36015,1	0,549545288
E6	1318,51	28,12821333	28,0	0,12821333	8402,6	0,128204346
F6	1396,91	29,80074667	29,0	0,80074667	52477,7	0,800735474
F#	1479,98	31,57290667	31,0	0,57290667	37546,0	0,572906494
G6	1567,98	33,45024000	33,0	0,45024000	29506,9	0,450225830
G#	1661,22	35,43936000	35,0	0,43936000	28793,9	0,439346313
A6	1760,00	37,54666667	37,0	0,54666667	35826,3	0,546661377
A#	1864,66	39,77941333	39,0	0,77941333	51079,6	0,779403687
B6	1975,53	42,14464000	42,0	0,14464000	9479,1	0,144638062
C7	2093,00	44,65066667	44,0	0,65066667	42642,1	0,650665283