Universidade de Caxias do Sul - UCS
Centro de Computação e Tecnologia da Informação
Bacharelado em Ciência da Computação

Rodolfo Stangherlin

Lighted Strings: Tecnologia Aplicada ao Aprendizado de Guitarra

Caxias do Sul
2013

Rodolfo Stangherlin

Lighted Strings: Tecnologia Aplicada ao Aprendizado de Guitarra

Trabalho de Conclusão de Curso para obtenção do Grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Orientador: Elisa Boff

Caxias do Sul
2013

Agradecimentos

Este trabalho não começou nos últimos anos, com o fim do curso de Bacharelado em Ciência da Computação, mas desde o dia que nasci, quando despertou a curiosidade de saber como as coisas funcionam e de aprender sobre tantas coisas interessantes que existem no mundo que nos cerca. Por isso, nem todo espaço do mundo seria suficiente para agradecer a todas as pessoas que, de alguma forma, fizeram e fazem parte desse aprendizado diário onde cresci. E continuo crescendo, como estudante, eterno aprendiz, como pessoa.

Não poderia deixar de agradecer ao *Nonno*, Sr. Altair Dallegrave, meu avô materno, que me ajudou neste e em tantos outros projetos. Todos os ensinamentos e brincadeiras presentes desde criança aguçaram minha curiosidade sobre sua oficina e suas ferramentas. Foi um incentivo muito bom para aguçar minha curiosidade desde criança. E à *Nonna*, Orlanda, "in memorian", que sempre permitia brincar na sua casa, bagunçar tudo e ainda servia um pão quentinho nas tardes depois da aula.

Agradeço ao tio Loi (Luiz Carlos), meu dindo, que é um exemplo para mim. Sua criatividade serve de inspiração para muitos projetos. E ao tio Ade (Ademir), que sempre tem uma habilidade manual na manga para solucionar algum problema ou construir alguma coisa.

Meus pais, Vilsso e Marta, que sempre incentivaram minha curiosidade por desmontar brinquedos, mesmo quando eu terminava por quebrá-los para entender como funcionavam. Também pelos incentivos no tempo da escola, quando o bom aprendizado serviu de base para todo o estudo que se seguiu aos anos iniciais. E pelos puxões de orelha, reclamações para dormir mais cedo, para me cuidar e para não deixar as coisas para última hora. Aprendi muito com o que me ensinaram.

Agradeço a todos os meus tios e tias, que sempre encontramos nos almoços ou jantares de família, trazendo conversas e opiniões diferentes. E principalemente aos tios e tias que me aguentavam nas férias (de verão e de inverno).

À minha namorada, Juliane, agradeço a compreensão nesse período complicado da vida chamado TCC pelas ausências, noites mal-dormidas e olheiras que me deixaram com uma cara abatida. Agradeço a ajuda neste trabalho também, com ideia muito boa para o processo de produção.

Agradeço à minha orientadora, professora Elisa, por me ajudar a manter o foco deste trabalho e não desviá-lo tentando implementar tudo o que se poderia imaginar. Sem esses empurrões e puxões de orelha, este trabalho não teria sido mais do que um amontoado de ideias. E obrigado pela paciência e pelas boas discussões sobre o ensino universitário,

principalmente de algoritmos. Aos demais professores, por todo conhecimento que trouxeram para a sala de aula nesses anos de graduação.

Aos colegas de curso, que me incentivaram a estudar mais do que simplesmente o que os professores colocam em aula e pelas longas discussões após as aulas - de filosofia à computação. Esse conhecimento que me trouxeram é diferente, mas não menos importante, do que o conhecimento formal das disciplinas universitárias.

Muito obrigado a todos que me incentivam, todos os dias da minha vida!

Resumo

Esta monografia analisa aspectos da aprendizagem de um instrumento musical e abordagens tecnológicas para esse objetivo. Algumas dessas tecnologias são clássicas, podendo até ser representadas em papel (como a tablatura), outras envolvem o uso de *software* e mais algumas empregango *hardware* específico para esse fim. Após essa análise, é proposta uma arquitetura baseada em *hardware* e *software* para servir de apoio nesse aprendizado, criando uma arquitetura extensível e aberta, permitindo expansões, melhorias e trabalhos derivados.

A implementação proposta une diversas das soluções estudadas, empregando múltiplos fatores já existentes e buscando aplicá-los ao aprendizado. A solução é composta por três partes principais: *software*, que deve executar em um PC, *software* embarcado, que é executado em um Arduino, e *hardware*, que é o próprio instrumento (a guitarra) com LEDs adicionados. O PC controlará os demais dispositivos conforme aulas pré-programadas por um professor de música ou entusiasta.

A monografia expõe testes realizados de arquitetura para a abordagem que suportam a construção de um protótipo para testes da solução.

Palavras-chaves: aprendizado de música. aprendizagem mediada por computador. computação embarcada.

Abstract

This dissertation examines the aspects of a music instrument learning and technological approaches to this purpose. Some of this technologies are standards ant they can be represented on paper (as the tablature), others envolve software and even others applay an specific hardware to this goal. After this analysis, an architecture based on hardware and software used to support music learning is proposed, building an extensible and open architecture, allowing expansions, improvements and derivated works.

This proposed execution aggregates different studied solutions, using multiple existent factors and trying aplly them to learning. This solution is made by three main parts: software, running into a PC, embedded software, running into an Arduino, and hardware, the instrument itself with LEDs. The PC control all other devices folowing preprogrammed classes by a music teacher or a fan.

The dissertation presents tests of the approach to architecture that will support the construction of a prototype to test the solution.

Key-words: music learning. computer based learning approach. embedded computing.

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application programming interface
ASCII	American Standard Code for Information Interchange
BSD	Berkeley Software Distribution
C	Programming language developed at Bell Labs in 1972
C++	Programming language developed at Bell Labs in 1979
CCS	Code Composer Studio
CI	Circúito Integrado
CLI	Command-line interface
EEPROM	Electrically Erasable Programmable Read-Only Memory
EUA	Estadis Unidos da América
GCC	GNU Compiler Collection
GND	Ground (terra)
GNU	GNU's Not Unix
GPL	GNU General Public License
GST	GStreamer
GTK	The GIMP Toolkit
GUI	Graphical user interface
Hz	Hertz
I/O	In/Out (entrada/saída)
I2C	Inter-Integrated Circuit
IDE	Integrated development environment
LED	Light-emitting diode
MIDI	Musical Instrument Digital Interface

MIT	Massachusetts Institute of Technology
MP3	MPEG-1 or MPEG-2 Audio Layer III
OS	Operating system (sistema operacional)
PC	Personal computer (computador pessoal)
PIL	Python Imaging Library
PWM	Pulse-width modulation
RAD	Rapid application development
RAM	Random-access memory (memória de acesso aleatório)
RGB	red-green-blue (varmelho-verde-azul) - modelo de cor
SCL	clock line (I2C)
SDA	data line (I2C)
SMF	Standard MIDI Files
TCC	Trabalho de Conclusão de Curso
USB	Universal Serial Bus
VCC	IC power supply pin
XML	Extensible Markup Language

Sumário

1 Introdução

A música acompanha o ser humano há muito tempo, mesmo sem evidências concretas de quando a música foi criada. Muitas pessoas que conhecemos tocam instrumentos musicais. Dentre os mais populares está o violão. Entre os mais jovens, a guitarra também se destaca.

Diversas habilidades são desenvolvidas na aprendizgem de música. A coordenação motora é fortalecida pois o aprendizado do instrumento exige habilidade do músico. Por exigir esforço e dedicação, crianças que tem contato com a aprendizagem de música desde cedo tornam-se alunos com menos problemas disciplinares. Além disso, a música pode reduzir o estresse, melhorando a qualidade de vida (HUMMES, 2004).

Apesar de inúmeros benefícios relacionados à aprendizagem de música e de um instrumento musical, muitas pessoas desistem de aprender um instrumento ou simplesmente não tentam aprender. Isso acontece em parte pelo modelo de expectativa e valor, que prediz que o indivíduo tem mais probabilidade de se envolver em que atividades que acredita poder realizar satisfatoriamente, ou seja, em relação às quais tem expectativa de sucesso (HENTSCHKE, 2009). Outro ponto que merece destaque é que a motivação ou aspecto de valor muda durante a vida. Por exemplo, um valor em atividade de lazer está presente para crianças e pessoas interessadas na atividade. Um valor de utilidade pode ser mais saliente na adolescência, quando o indivíduo se prepara para exercer determinada profissão ou, quando adulto, em escolhas que influenciarão a carreira (HENTSCHKE, 2009).

Existem diversos problemas em utilizar apenas um método para o ensino de instrumentos musicais. O ensino da música feito apenas com a escrita convencional, ou seja, a partitura, não garante ao aprendiz uma prática satisfatória no seu instrumento musical (VIEIRA; RAY, 2007). O fato de alguém saber ler partitura, por exemplo, não garante a esta pessoa a capacidade de acompanhar um cantor com instrumento harmônico.

1.1 Aprendizagem musical: alternativas tecnológicas

Diversas alternativas tecnológicas existentes foram pesquisadas por autores de outros trabalhos (GOHN, 2003). Atualmente, algumas das alternativas incluem vídeos (em canais de TV ou vídeo-aulas) e o computador (com ferramentas online, Internet e também vídeos). Na Internet, diversos sites disponibilizam material e aulas on-line. Por exemplo, a Figura 1 representa um *website* com vídeo-aulas de bateria (sua única funcionalidade tecnológica é um *player* de vídeo). Nesse caso, o problema não é encontrar material, mas sim classificá-lo quanto à sua qualidade.

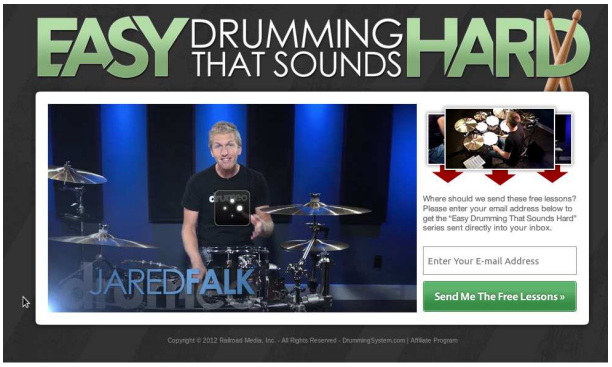

Figura 1 – Captura do website drummingsystem.com

Estão disponíveis, também, materiais específicos para violão e guitarra (SILVA, 2008a) (SILVA, 2008b), bem como revistas disponibilizadas também *online*, como na Figura 2. Essas revistas foram uma das primeiras formas desenvolvidas para o auto-aprendizado musical, mesmo antes da popularização da Internet. Elas são, de certa forma, antecessores das vídeo-aulas e da proposta de solução deste trabalho.

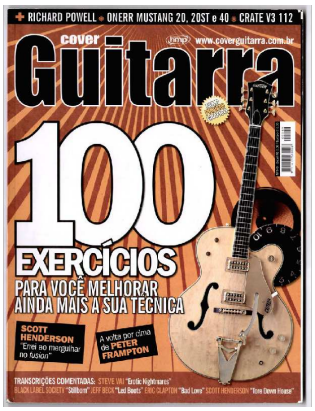

Figura 2 – Capa de uma revista sobre guitarra disponibilizada na web

Seguindo uma abordagem ainda mais tecnológica, o uso de realidade aumentada também mostra-se interessante para o ensino de violão e guitarra (LIAROKAPIS, 2005). Nessa aborgagem, mistura-se o real e o virtual para demonstrar, em um vídeo com o instrumento capturado em tempo real, quais cordas devem ser pressionadas, demonstrado na Figura 3. O uso de realidade aumentada baseia-se em cartões com *tags* impressas que são posicionadas na guitarra em frente a uma câmera em um computador. Ao ler essa *tag*, um *software* identifica a nota a que ela se refere e reproduz a imagem da *webcam* com apontamentos (no caso da Figura 3, pontos vermelhos) das cordas correspondentes.

Uma proposta similar ao que este projeto propõe chama-se gTar (BECK, 2012), que coloca um IPhone em uma guitarra real com LEDs, demonstrando como tocar música no próprio instrumento, como mostrado na Figura 4. O diferencial da implementação aqui proposta está no fato de ser aberta para qualquer dispositivo USB, além de prever não apenas o *hardware*, mas também o *software*.

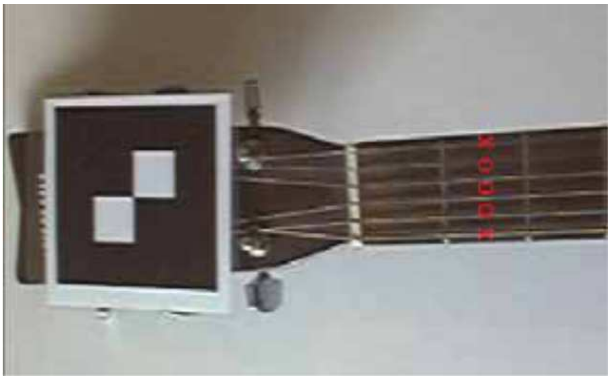

Figura 3 – Implementação teórica de realidade aumentada no ensino de Guitarra

Figura 4 – Implementação do gTar, similar à proposta apresentada

Outra proposta interessante é o *website* SoundSlice. Ele apresenta uma música de diversas formas, incluindo o nome da nota, tablatura, estrutura da música e vídeo (utilizando o serviço do Youtube^{TM}). Uma funcionalidade desse *website* é a possibilidade de reproduzir as músicas em um tempo menor (menor velocidade), facilitando o acompanhamento. A Figura 5 mostra o *website* e suas funcionalidades. Na metade superior, tem-se o controle da música (*play* e *pause*), controle da velocidade (*full speed* e *half speed*), o vídeo produzido para acompanhamento e dados da música, como nome, autor e tempo de duração. Na metade inferior, tem-se a indicação do tempo (cronológico, em minutos e segundos), tablatura, tempo musical e o nome da nota que será reproduzida no instrumento.

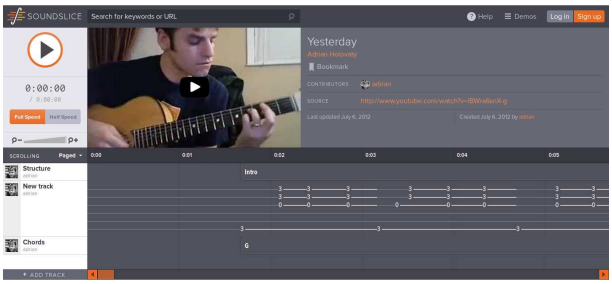

Figura 5 – soundclice.com

Recentemente, diversas universidades passaram a disponibilizar seus cursos online abertos ao público. Através do website coursera.org, Figura 6, o Berklee College of Music (Bostom, Massachusetts, EUA) disponibilizou o curso *Introduction to Guitar*, ministrado por Thaddeus Hogarth. Com duração de 6 semanas, o curso oferece vídeos e propõe atividades onde as gravações devem ser enviadas para avaliação pelo instrutor (HOGARTH, 2013).

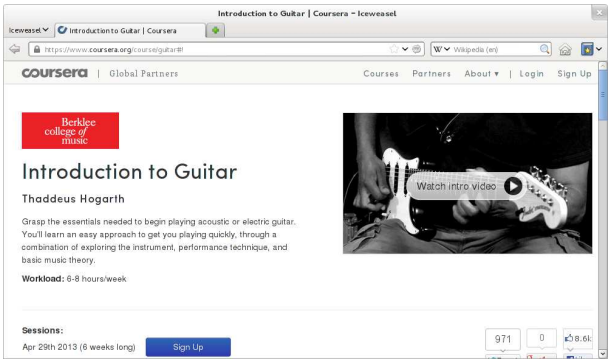

Figura 6 – Página inicial do custo Introduction to Guitar - coursera.org

O uso de tecnologia no ensino de música não fica restrito à guitarra e violão. A Casio e a Yamaha, por exemplo possuem uma série de produtos com uma função denominada "*Lighted Keys*" (YAMAHA, 2012) (CASIO, 2012). Seu funcionamento é simples: o teclado guia o músico indicando as teclas que devem ser pressionadas iluminando cada uma delas no momento correto, como representado na Figura 7. Além disso, alguns teclados também possuem uma tela que indica qual o dedo correto deve ser utilizado em cada tecla. Inspirado nesse produto, o presente trabalho busca criar um protótipo para aplicar essa solução ao aprendizado de violão e guitarra.

Figura 7 – Representação da Casio para o *Lighted Keys*

1.2 Objetivo

O presente trabalho busca uma alternativa tecnológica para o ensino e aprendizagem de música. Essa alternativa tecnológica diferencia-se de outras já disponíveis, apesar de agregar diversas delas. A implementação proposta inclui *hardware* e *software* voltados para

a aprendizagem de guitarra. A proposta inclui, também, a possibilidade de trabalhos futuros basearem-se no protótipo desenvolvido, permitindo a criação de jogos e outras aplicações voltadas a diferentes públicos e atividades.

Neste contexto, este trabalho é norteado pela seguinte questão de pesquisa: É possível desenvolver uma aplicação que controle um *hardware* específico para apoiar a aprendizagem dos acordes de guitarra?

1.3 A Estrutura do Texto

No Capítulo 1 estão descritas alternativas tecnológicas e trabalhos relacionados. Baseado no estudo realizado para produção deste capítulo, foi elaborada a questão de pesquisa e os objetivos do presente projeto.

Seguindo nos estudos sobre computação e música, o Capítulo 2 documenta o estudo realizado em conceiros e representaçcoes musicais para o instrumento alvo da proposta. Também neste capítulo são descritas formas existentes de representação musical para computadores.

O presente projeto utiliza desenvolvimento de *hardware* e de uma forma de comunicação com o *software* desenvolvido. Para isso, foi necessário um estudo sobre computação embarcada, que se encontra no Capítulo 3. Uma comparação entre alternativas para criar um protótipo busca escolher uma forma simples de implementar o *hardware* e o *firmware* necessários para o desenvolvimento da proposta de solução.

Por fim, é detalhado detalhar como o protótipo será construído. No Capítulo 4, uma descrição detalhada da proposta de solução pode ser encontrada, juntamente com passos realizados para construção do protótipo necessário para validá-la. Ao fim do trabalho, no Capítulo 5, são detalhadas dificuldades, contribuições e possíveis trabalhos futuros para o prosseguimento do projeto.

2 Computação e Música

A proposta deste trabalho visa aplicar conhecimentos de computação na música. Para isso, será necessário explorar diversos conceitos, notações e nomeclaturas adotados no meio musical. Este capítulo descreverá os tópicos relevantes para a implementação proposta.

2.1 Partes da Guitarra e Violão

A guitarra elétrica e o violão guardam diversas similaridades. Podemos ver as partes que formam cada um na Figura 8 (LIZ, 2011).

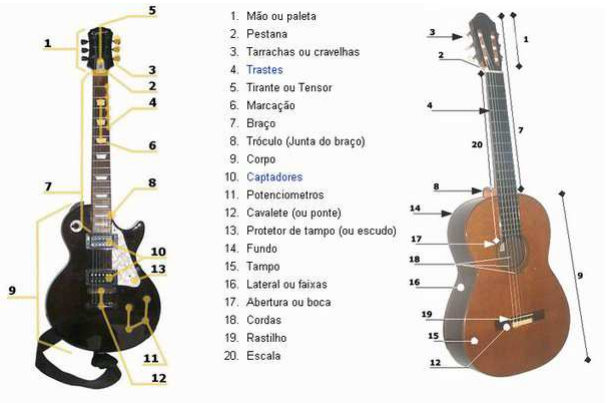

Figura 8 – Partes de uma guitarra e um violão

Para facilitar o aprendizado e a leitura de algumas notações, as cordas foram numeradas no intrumento. A primeira corda é a que tem o som mais agudo e a sexta é a com o som mais grave, como pode ser visto na Figura 9 (BAY, 2005).

2.2 Os Dedos

Também para facilitar o aprendizado, os dedos foram numerados e padronizados. Diversos autores utilizam uma numeração que pode se colocada em diversas notações. A numeração da mão esquerda (que fica no braço do instrumento) inicia em 1, no dedo indicador, e vai até 4, no dedo mínimo, conforme Figura 10 (BAY, 2005).

Figura 9 – Numeração das cordas no braço do instrumento

2.3 Estudo de Guitarra

Diversas habilidades são necessárias para o aprendizado de violão e guitarra. Por serem instrumentos similares, o material de estudo de um instrumento pode ser facilmente adaptado ao outro. O autor de Toque Junto - Guitarra (CASTILHO, 2000), por exemplo, utiliza um violão em algumas melodias.

Alguns tópicos de estudo necessários para tocar guitarra (CAESAR, 2003) são listados:

- Leitura
  - Rítmica – estuda os valores das notas;
  - Melódica – estuda as notas na pauta, já relacionada com a localização no braço;
  - Harmônica – estuda acordes na pauta.
- Intervalos, Escalas e Acordes – entender os desenhos (geométricos) no braço da guitarra, compreendidos visualmente e auditivamente;
- Grooves – aprender os principais padrões rítmicos e harmônicos dos Acompanhamentos.
- Técnica – fundamentos das articulações e movimentos das duas mãos no instrumento, conhecimento prático dos elementos técnicos e técnicas empregadas nos diversos estilos musicais.
- Percepção – desenvolvimento do "ouvido musical".

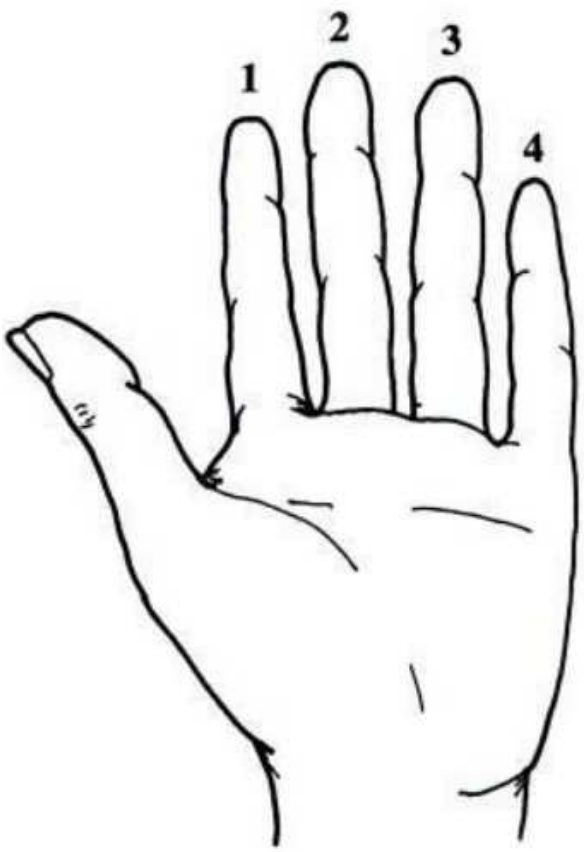

Figura 10 – Numeração dos dedos da mão esquerda

- Harmonia – relação dos acordes com as melodias.
- Improvisação – processo de criatividade espontânea, mas que precisa ser trabalhando com exercícios de padrões melódicos.
- Chord-Melody Style – tocar melodia e harmonia simultaneamente.
- Estudo de estilos – blues, rock, jazz, entre outros.
- Arranjos e Composição

2.4 Palheta

Opcional para o violão e obrigatória para a guitarra, a palheta (*flatpicks*) pode tocar as cordas de diversas maneiras, como de cima para baixo (*down strum*) ou de baixo para cima (*up strum*) (BAY, 2005). A notação musical pode indicar como o movimento deverá ser executado.

A palheta alternada é o método mais eficiente para iniciantes (GILBERT, 2007). Ela consiste em movimentos consecutivos para cima e para baixo. É necessário o cuidado para que o movimento para cima (inerentemente mais fraco) tenha o mesmo som que o movimento para baixo.

2.5 Notas e Acordes

Aprender como as notas são representadas é um pré-requisito importante para compreender e fazer uso eficaz de muitos dos materiais básicos de música, incluindo acordes, escalas, tablaturas e assim por diante (HEWITT, 2008).

2.5.1 Nota

Nota (pitched) é o som cuja altura é definida e identificada (DOURADO, 2004). Também pode ser denominada frequência (HEWITT, 2008) e é medida em Hertz. O ouvido humando capta sons entre 20 Hz e 20 kHz (RUI; STEFFANI, 2007).

As notas possuem frequências estáveis, ou seja, idealmente possuem a mesma frequência independente do instrumento musical. A afinação dos instrumentos devem garantir essa consistência. A nota "lá"("A") tem frequência de 440 Hz.

As características das notas podem ser mapeadas como qualidades sensoriais e a amplitude da nota está relacionada com a intensidade do som (popularmente conhecido como "volume"). Quanto maior a amplitude, maior o deslocamento do ar e, portanto, maior a intensidade percebida. Já a frequência é associada à altura de um som, se ele é grave (baixas frequências) ou agudo (altas) (OGASAWARA et al., 2008).

2.5.2 Acorde

Acorde (ou *chord*, em Inglês) é um grupo de notas, tipicamente três ou mais, emitidas simultaneamente (DOURADO, 2004). Um acorde pode ser representado e tocado de mais de uma forma na guitarra (elétrica ou acústica) (MANUS, 1978). Além disso, existem variações na posição dos dedos, mesmo que usando as mesmas cordas.

2.5.3 Compasso

É a unidade métrica musical formada por grupos de tempos em porções iguais. Popularmente, é conhecido como "batida", "ritmo"ou "tempo". Pode ser binário (dois tempos), ternário (três) ou quaternário (quatro).

2.6 Representação Musical

As notações musicais procuram encontrar o melhor compromisso entre riqueza e legibilidade, quanto mais rica a notação, mais precisa ela pode ser. Porém também menos legível (CABRAL et al., 2001).

Internacionalmente, o alfabeto musical é representado por 7 letras: C, D, E, F, G, A e B, (HEWITT, 2008). Essa notação, no entanto, é comum para a língua inglesa. Na

América Latina, a notação se dá por Do, Re, Mi, Fa, Sol, La e Si (SANCHEZ, 2003), respectivamente.

2.6.1 Cifras

Cifras são símbolos que definem os acordes, representados por letras (A a G, ou Dó, Ré, Mi...), acidentes e algarismos junto às palavras ou notas. (DOURADO, 2004)

2.6.2 Partitura

A partitura (*sheet music*) é, genericamente, música impressa (DOURADO, 2004) e escrita sobre a pauta (ou pentagrama; *staff* em inglês). São 5 linhas paralelas e 4 espaços onde as notas serão representadas (GORDON, 2007). Linhas e espaços são numerados de baixo para cima. As notas são representadas por símbolos ovais e podem estar nos espaços ou nas linhas, como na Figura 11. Se as notas aparecerem no topo da pauta, as notas serão mais altas (agudas). Se aparecem na base, serão mais baixas (graves)[1].

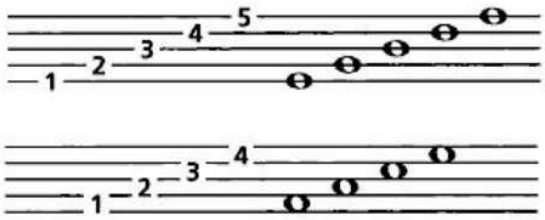

Figura 11 – Pauta (ou pentagrama) com numeração de linhas e espaços

Símbolos são utilizados para identificar notas altas e baixas. A Clave de Sol (em inglês, *"treble staff"*), representada na Figura 12, é usada para identificar as notas mais altas. A partitude pracisa de uma referência e a Clave de Sol indica a posição das linhas e espaços. Assim, pode-se identificar as notas conforme a posição nas linhas ou espaços. O violão é um instrumento transpositor: suas notas escritas em clave de sol soam uma oitava abaixo da partitura (SANCHEZ, 2003). A Clave de Fá (em inglês, "bass clef"), representada na Figura 12 indica que a partitura representa notas mais graves.

As notas podem ser indicadas fora das linhas e espaços da partitura adicionando-se "espaços suplementares" (*ledger lines*). São usadas quando as notas são muito altas ou muito baixas para serem representadas na partitura. Na Figura 13, as notas são as mesmas B, CeD, porém em oitavas diferentes.

Para guitarra e violão, a partitura pode indicar também a direção que se deve tocar as cordas ("baixo para cima" ou "cima para baixo"). Para isso, são utilizados os símbolos da Figura 14. Eles são colocados no topo da partitura, como mostrado na Figura 15 (GILBERT, 2007).

[1] "alta" e "baixa" referem-se à frequência das Notas Musicais

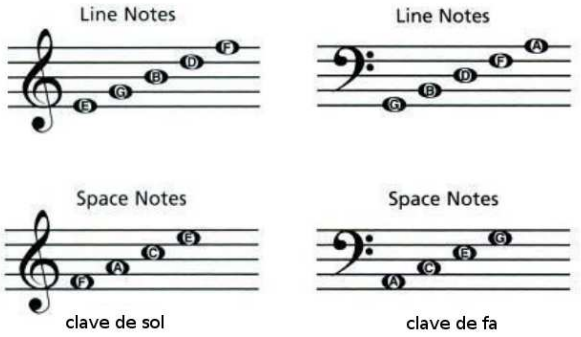

Figura 12 – Indicações das notas na tablatura, utilizando linhas e espaços, na notação em Clave de Sol e Clave de Fá

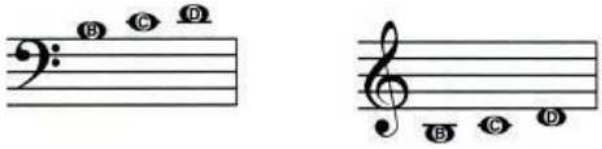

Figura 13 – Espaços suplementares na partitura

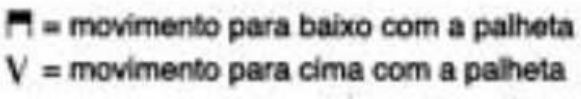

Figura 14 – Símbolos da direção da palhetada na partitura

A duração das notas também é representada na partitura. Para isso, utiliza-se o símbolo oval (chamada "cabeça da nota", ou "notehead") e um linha ligada a ela (chamada "haste", ou "stem"). Uma comparação da duração das notas pode ser medida como na Figura 16 (GORDON, 2007). As figuras das notas recebem diferentes nomes conforme a sua duração: breve (dobro de um tempo), semibreve (um tempo), mínima (1/2 tempo), semínima (1/ tempo), colcheia, semicolcheia, fusa e semifusa. A representação pode ser vista na Figura 17 (CARDOSO; MASCARANHAS,).

A haste deve ser escrita à esquerda e apontando para baixo se a nota aparecer na terceira linha ou acima dela na pauta. Caso contrário (abaixo da terceira linha), a haste fica à direita, apontando para cima.

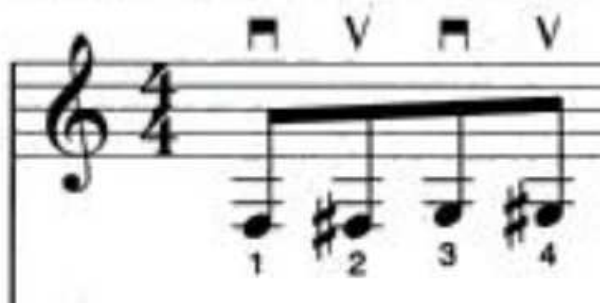

Figura 15 – Indicação de direção da palhetada na partitura

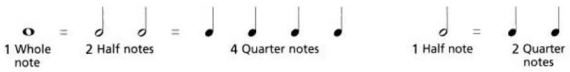

Figura 16 – Símbolos da duração das notas em uma partitura

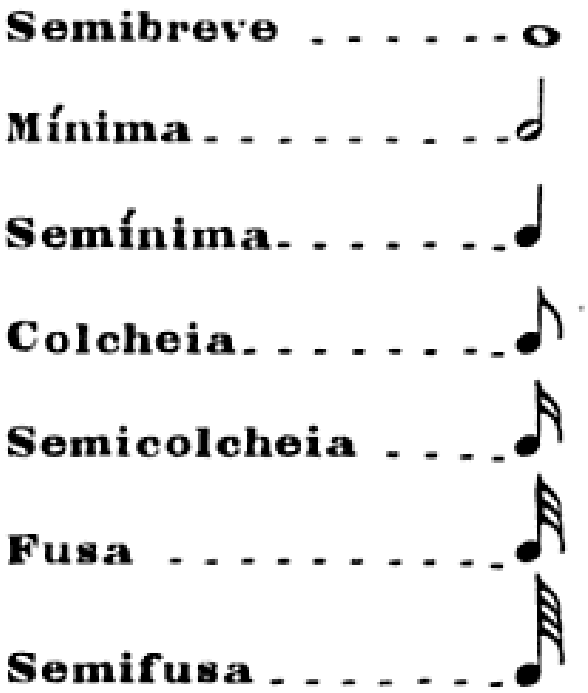

Figura 17 – Nomes dos símbolos conforme sua duação

O tempo (time signature) aparece no início da música após o símbolo da clave. Contém 2 números, um abaixo do outro. O número superior indica quantas "batidas" cada medida da partitura. O número inferior indica qual o tipo da nota recebe 1 batida. No exemplo da Figura 18, com 4/4, são 4 batidas para cada "quarto de nota". Uma comparação pode ser feita nas Figuras 19, 20 e 21. Os traços indicam que os sons são contínuos.

Figura 18 – Indicação dos tempos musicais em uma partitura

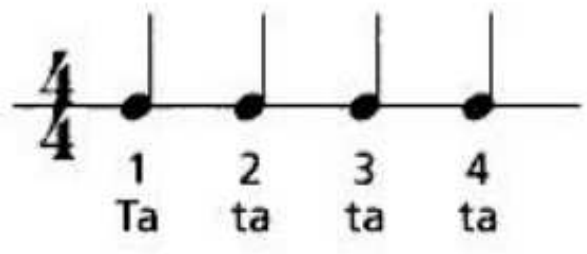

Figura 19 – Representação do tempo de duração da nota (1/4)

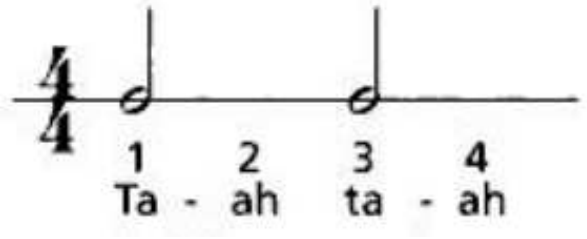

Figura 20 – Representação do tempo de duração da nota (1/2)

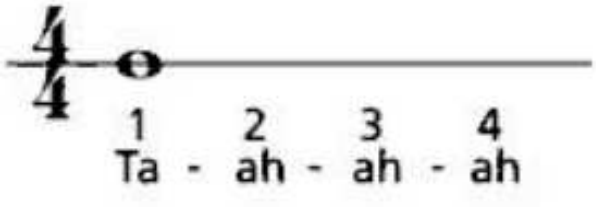

Figura 21 – Representação do tempo de duração da nota (1/1)

Pausa (rest) também é representada na partitura. São usados traços e outros símbolos para indicar a quantidade de tempos sem som. Exemplo na Figura 22.

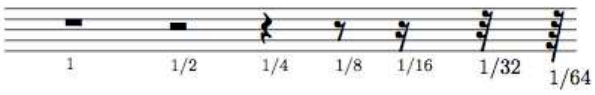

Figura 22 – Indicação dos tempos de pausa (sem som) em uma partitura

2.6.3 Tablatura

A tablatura (*tab* ou *tablature*, em Inglês) é uma notação simplificada, considerada a partitura para a guitarra (CHESNEY, 2004), que indica ao músico o que fazer com os dedos para tocar uma nota (HESELWOOD, 2008). Nessa notação, as linhas representam as cordas do instrumento no mesmo sentido que o braço da guitarra. Os números nas linhas representam as casas que o músico deve tocar. O 0 (zero) indica que a corda deve ser tocada solta e, se não há número na linha, a corda correspondente não deve ser tocada (CAESAR, 2003). Uma visualização dessa representação pode ser vista na Figura 1. A

tablatura é apenas uma ferramenta que ajuda o guitarrista a iniciar a sua longa caminhada como músico (SILVA, 2008a).

```
Mi   e  1 | - - - - - - - - - - - 0 - - - - - - - - - - |
Si   B  2 | - - - - - - - - - - 0 - - - 0 - - - - - - - |
Sol  G  3 | - - - - - - - 0 - - - - - - - 0 - - - - - - |
Ré   D  4 | - - - - - 0 - - - - - - - - - - 0 - - - - - |
Lá   A  5 | - - - 0 - - - - - - - - - - - - - 0 - - - - |
Mi   E  6 | - 0 - - - - - - - - - - - - - - - - - 0 - - |
```

Representação 1: Exemplo de tablatura

Na Figura 1, a primeira coluna contém as notas (ou números) que identificam cada corda da guitarra. No entanto, nem sempre ela é incluída na representação. A ordem das cordas é sempre a indicada.

2.6.3.1 ASCII Tab

ASCII Tab é um arquivo texto que representa uma tablatura. Tem como objetivo ser um formato lido tanto por humanos quanto por máquinas. Como exemplo, temos o acorde "Dó" (C) na Figura 2.

```
| - - - - - 0 - - - - - - |
| - - - - - 1 - - - - - - |
| - - - - - 0 - - - - - - |
| - - - - - 2 - - - - - - |
| - - - - - 3 - - - - - - |
| - - - - - - - - - - - - |
```

Representação 2: Exemplo de tablatura em ASCII, com o acorde "Dó" (C)

As tablaturas em ASCII podem representar muitas outras situações, sendo legível tanto para computadores quanto para pessoas (já que utiliza apenas caracteres padrão ASCII). Ela segue exatamente a mesma notação da tablatura comum, com cada linha indicando uma corda e os números indicando as casas.

Por não indicar o ritmo, normalmente vem acompanhada da notação tradicional (SACRAMENTO, 2007), a partitura. Um exemplo completo pode ser visto na Figura 23

2.6.4 Diagrama de Acordes

As notas podem ser representadas em um "diagrama de acordes" (*"fretboard diagram"*, em Inglês). Eles identificam as cordas da guitarra que devem ser tocados para cada acorde. Algumas vezes indicam também quais dedos devem ser usados para compor o acorde. Na Figura 24 vemos uma representação do acorde "Dó" (C) na vertical. O desenho representa

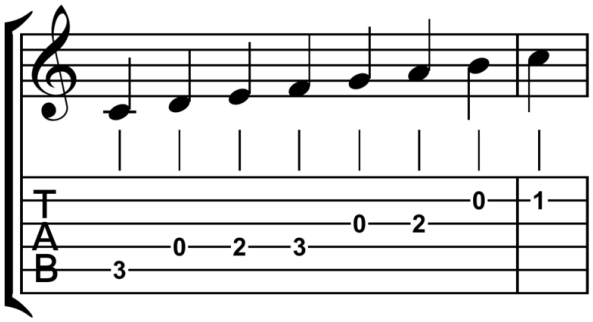

Figura 23 – Tablatura acompanhada de Partitura

o braço do instrumento, com as linhas horizontais representando os trastes (*frets*)[2]. Os espaços entre as linhas são as casas (*fingerboard* ou *fretboard*), onde o músico coloca os dedos. O topo do desenho apresenta uma linha mais escura que representa a pestana[3]. Na parte de baixo da figura, vê-se círculos e a letra "x". Os círculos representam quais cordas serão tocadas, o "x" representa a corda que não deve ser tocada. O círculo escuro indica a primeira corda a tocar, indicando o sentido para tocar as cordas. Os círculos e o "x" podem aparecer também no topo da imagem, mas possuem o mesmo significado.

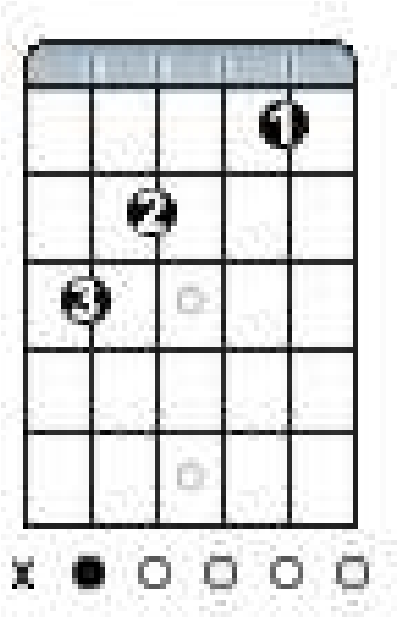

Figura 24 – *Freatboard* na vertical

Por não ser uma notação oficial, existem variações (OVERLY, 2004). As indicações das cordas que devem ser tocadas podem estar na base da imagem. Ou, ainda, o desenho pode ficar no horizontal, como na Figura 25. Nessa notação, a pestana fica na esquerda, com uma linha mais expessa (SILVA, 2008a). Em outros casos, o diagrama mostra a pestana

[2] Partes do Intrumento na Figura 8
[3] Partes do Intrumento na Figura 8

como uma linha vasada, à esquerda, como na Figura 26. Em outras, apenas um fragmento é representado, sem pestana indicada, podendo os espaços serem numerados, indicando a casa que está sendo representada. Alternativamente, a representação é meramente auxiliar, sem representação de casas, como na Figura 27 (OVERLY, 2004).

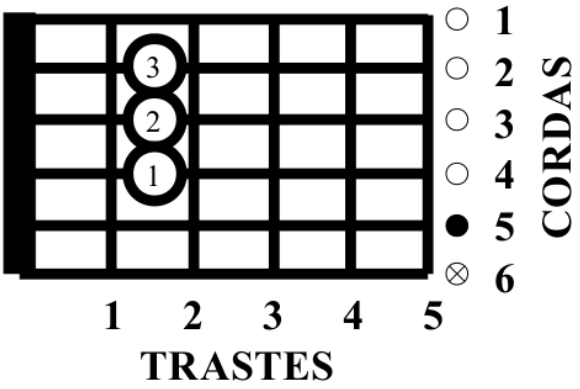

Figura 25 – *Fretboard* na horizontal

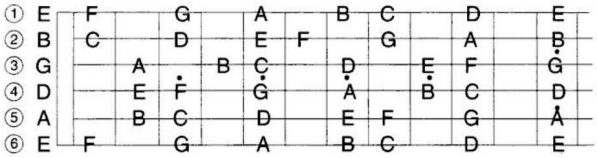

Figura 26 – Diversas notas em um único *fretboard*

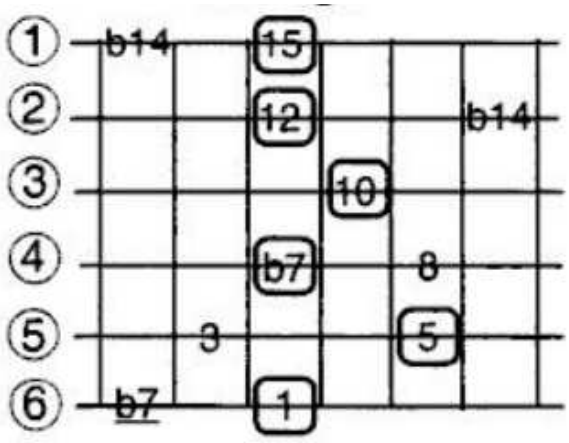

Figura 27 – Fragmento de um *fretboard*

Pode-se observar que alguns diagramas de acordes indicam números nas cordas que devem ser tocadas. Esses números indicam os dedos, numerados conforme visto na Seção 2.2.

2.7 Notação Computacional

Os conceitos vistos até agora são voltados à musica, com o objetivo de representá-la aos do instrumentista. No entanto, para a implementação proposta neste trabalho, será necessário representar computacionalmente essas informações.

2.7.1 MIDI

MIDI é a sigla de Musical Instrument Digital Interface (Interface Digital para Instrumentos Musicais). A especificação desse formato foi publicada pela primeira vez em Agosto de 1983, criado para permitir a comunicação entre diferentes instrumentos de diferentes fabricantes. Originalmente limitado apenas a dispositivos eletrônicos, o padrão MIDI permitiu a rápida expansão desses instrumentos e dos *softwares* computacionais relacionados à música. Permitiu, também, o início da computação no ensino de música (SKILLBOX, 2012).

O MIDI foi concebido, originalmente, para enviar eventos de execução de música de um instrumento para outro. O objetivo é prover facilidade para transmissão de todos os eventos que o músico executa no instrumento "crontrolador" para o instrumento "receptor". Atualmente, diversos equipamentos possuem interface MIDI (como guitarras, baterias e equipamentos de efeitos), mas inicialmente apenas os teclados utilizavam MIDI (RATTON, 1991).

As mensagens básicas do MIDI representam os elementos mais significativos de uma performance musical. Alguns elementos dessas mensagens são "note on", "note off", "velocity", "Pitch bend" e "aftertouch" (NICHOLL, 1993). Eventos como início e fim das notas, como representado na Figura 28, são representados como "Note on" e "Note off". A força com que uma nota é tocada também pode ser representada como velocidade. Comumente, notas tem volume mais alto ou baixo conforme são mais fortes ou fracas, respectivemente (Figura 29).

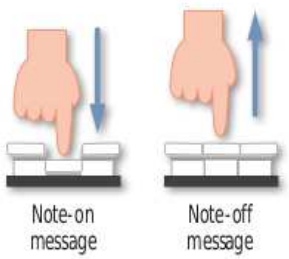

Figura 28 – Evento no protocolo MIDI

"Pitch bend" é um tipo de mensagem geralmente transmitida movendo um regulador de modulação, geralmente à esquerda de um controlador do tipo teclado.

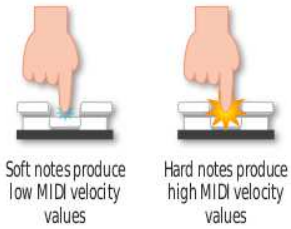

Figura 29 – Velocidade no protocolo MIDI

"Aftertouch" é o comando disparado depois de colocar pressão em uma tecla. Para cada tecla pode-se colocar pressão diferente, gerando, portanto, mensagens diferentes.

Um sequenciador MIDI é um equipamento que permite armazenar os códigos gerados pelo controlador preservando a ordem e o tempo em que ocorrem. Com os códigos armazenados, é possível executar novamente a música. Expressa-se "executar novamente" e não "reproduzir", pois é necessário que o instrumento (que pode ser a placa de som de um computador) toque todos os eventos novamente. O padrão MIDI original era adequado para lidar com tempos musicais (compassos, tempos) e não cronológicos (horas, minutos, segundos). Para tornar possível o uso de sequenciadores, foi desenvolvido o "MIDI Time Code" como uma extensão à especificação do formato (RATTON, 1991).

Arquivos que armazenam os eventos MIDI seguem uma especificação denominada Standard MIDI Files (SMF). Possuem, por padrão (mas não obrigatório), a extensão ".mid". Um arquivo SMF possui, além das informações padrão do protocolo MIDI, dados adicionais como tempo (que permite a execução em ordem e velocidade corretas), instrumento (por canal) e configurações do controlador, assim como informações da composição (copyright ou autor, por exemplo).

É possível encontrar arquivos MIDI na internet e existem *softwares* já disponíveis para criá-los. Na Figura 30, uma tela do *software* Muse (MUSE, 2012) exibindo os controles do canal "Acustic guitar" (guitarra acústica) para a execução de Yesterday (The Beatles). Outros *softwares* representam o canal MIDI da mesma forma, representando um teclado mesmo que o instrumento seja outro. A Figura 31 mostra o *software* Seq24 (SEQ24, 2012) e a Figura 32 mostra o *software* QTractor (QTRACTOR, 2012).

2.7.2 MusicXML

MusicXML é um padrão para descrição de partituras baseado em XML. Seu objetivo é representar para o compartilhamento de partituras o que o MP3 representa para o compartilhamento de gravações musicais. Segundo o seu *website*, mais de 150 aplicações[4] incluem, hoje, suporte ao formato. O seu *website* oficial ainda disponibiliza uma lista com inúmeros outros *websites*[5] que provêem download de partituras nesse formato (MAKEMUSIC,

[4] Lista disponível em http://www.makemusic.com/musicxml/community/software
[5] Lista disponível em http://www.makemusic.com/musicxml/music

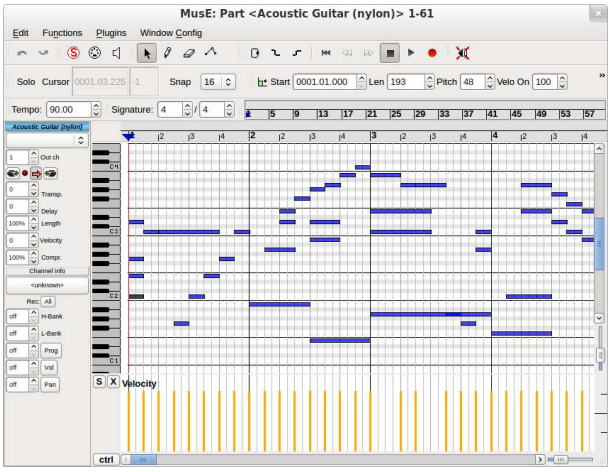

Figura 30 – *Screenshot* do *software* Muse

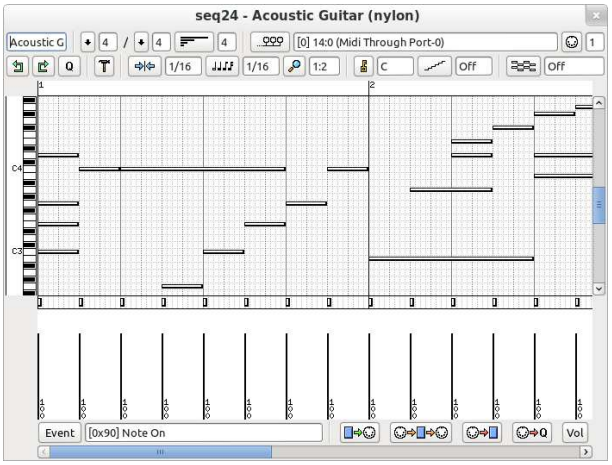

Figura 31 – *Screenshot* do *software* Seq24

2012).

A mesma música testada em aplicativos em formato MIDI (Yesterday - The Beatles) pode ser encontrada em MusicXML. No Noteedit (NOTEEDIT, 2012), que pode ser visto na Figura 33, vê-se que a partitura vem acompanhada da letra.

Outro *software* onde o formato foi testado foi o KGuitar (KDE, 2010), do projeto KDE. Focado diretamente em violão e guitarra, pode ser visto na Figura 34 que o *software* mostra, além da partitura, a tablatura e uma imagem do braço do instrumento, indicando quais cordas devem ser tocadas. Essa representação em *software* é o mais aproximado com a implementação diretamente no braço do instrumento encontrada nesse estudo.

Um exemplo da notação de MusicXML pode ser visto na Figura 3. O resultado desse simples XML pode ser visto na Figura 35.

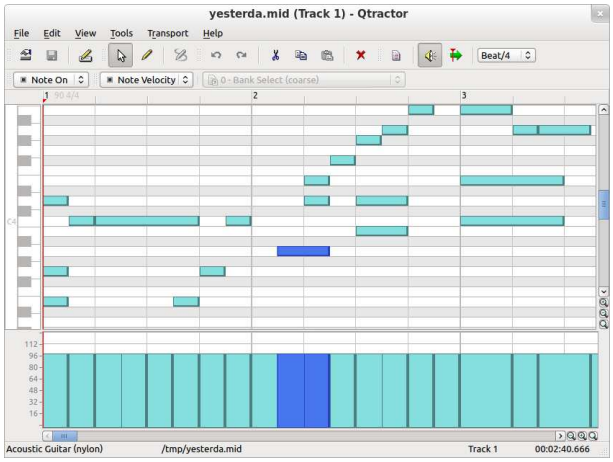

Figura 32 – *Screenshot* do *software* QTractor

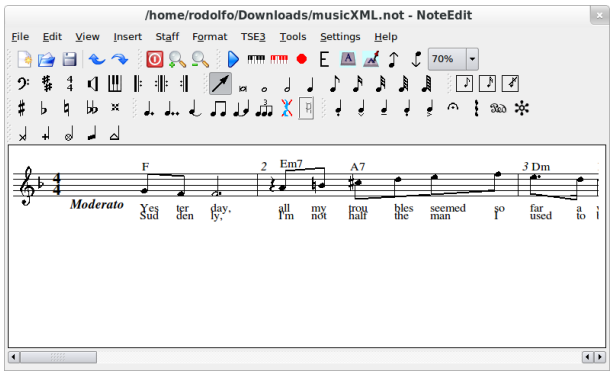

Figura 33 – *Screenshot* do *software* NoteEdit

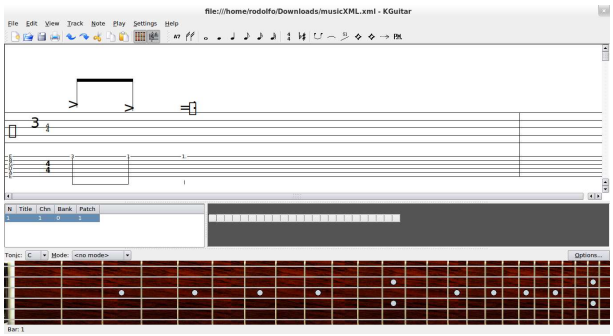

Figura 34 – *Screenshot* do *software* KGuitar

2.7.3 APIs

Existem diversas bibliotecas (libraries) prontas para ler e editar arquivos de representação de música nos diferentes formatos. Uma delas é a music21 (CUTHBERT, 2012b), desenvolvida pelo MIT. Escrita em Python, ela permite converter e editar não apenas entre MIDI e MusicXML, mas entre outros formatos mais incomuns de representação musical. Essa biblioteca está disponível sob Creative Commons Attribution ShareAlike[6] e o código-fonte

[6] Detalhes do licenciamento podem ser encontrados em (CREATIVECOMMONS, 2012)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 2.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="2.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Representação 3: Exemplo de representação de MusicXML

está disponível no repositório Google Code (CUTHBERT, 2012a).

2.8 Considerações Finais

Diversos tópicos foram listados sobre o aprendizado do instrumento musical. Em alguns casos, foram definidos padrões a serem seguidos, como numeração de dedos e cordas. Esse conhecimento será importante no desenvolvimento da solução proposta pois torna possível

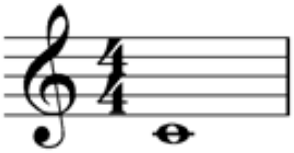

Figura 35 – Partitura gerada a partir do exemplo em MusicXML da Figura 3

compreender o que deverá ser indicado durante o aprendizado.

Diferentes formas de representar músicas foram apresentadas, nem sempre oficialmente padronizadas ou mesmo especificadas. O trabalho a ser desenvolvido pretende colocar no próprio instrumento algumas dessas notações. A que mais se adapta à essa implementação é o Diagrama de Acordes, pois indica exatamente quais casas devem ser pressionadas e quais cordas precisam ser tocadas. Esses diagramas indicam apenas um acorde. A tablatura representa mais de um acorde, também sendo uma fonte interessante para a proposta de solução. Ao contrário do diagrama de acordes, uma tablatura demonstra mais de um acorde na mesma notação, podendo representar trechos ou uma música inteira. Assim, a tablatura se mostra mais adequada para armazenar os dados necessários ao programa, quando a execução for uma música ou um trecho. Além disso, a tablatura tem significado para o músico, sendo uma alternativa a ser exibida durante a execução do diagrama de acordes na guitarra.

Os estudos de formas de representação computacional de música coloca a necessidade de criar uma forma diferente de comunicação. MIDI está voltado quase inteiramente para teclados, enquanto MusicXML apresenta complexidade para ser tratada diretamente pelo *software* embarcado. O *software* poderia incluir funcionalidades para importar esse tipo de arquivo de música, mas a comunicação com o dispositivo será através do protocolo estabelecido, permitindo controles mais precisos, inclusive permitindo indicações que não fazem parte das notas. Outra vantagem dessa abordagem é permitir flexibilidade para as diferentes possibilidades de cada nota na guitarra (que não são representadas na notação MIDI, por exemplo).

3 Computação Embarcada

Parte da implementação do trabalho é a adaptação de uma guitarra, dispondo LEDs no braço para indicar a posição correta das notas no instrumento. Para isso, serão analisadas duas tecnologias que permitem o controle de LEDs através da comunicação USB com um computador: Arduino e Freescale Launchpad.

3.1 Arduino

Segundo seu próprio *website*, o Arduino é uma plataforma open-source para prototipagem baseado em *hardware* e *software* flexível e fácil de usar. Planejado para servir artistas, designers, hobbistas e qualquer interessado em criar objetos ou ambientes interativos (ARDUINO, 2012c).

Em publicação no The Wall Street Journal, o Arduino é descrito como um cérebro eletrônico executando qualquer coisa, de um robô de ensino médio até instalações de arte (LAHART, 2009). O projeto do Arduino pode ser encontrado no *website* oficial do projeto (ARDUINO, 2012c), e qualquer um pode legalmente copiar, construir e vender o *hardware* desenvolvido. É classificado como "hardware-aberto". Diversos projetos podem ser encontrados utilizando a plataforma, como no *website* Instructables (INSTRUCTABLES, 2012).

Foi criado na Itália, no Interaction Design Institute da cidade de Ivrea, em 2005. O professor Massimo Banzi procurava uma forma de tornar mais simples para os estudantes de design trabalharem com tecnologia. Em discussões com David Cuartielles, um engenheiro visitante de Malmö University, Suécia, decidiram juntos construir o *hardware* para ser incorporado em seus trabalhos.

Existem clones do projeto, alguns inclusive brasileiros, que são vendidos a preços menores do que os originais, que incluem fretes e impostos internacionais (DOITKITS, 2012).

3.1.1 Hardware

A última versão do Arduino disponível durante a produção deste trabalho (2013) é chamada Arduino Leonardo. Ele permite que, via *software*, o Arduino seja identificado como mouse, teclado ou *joystick* USB. As versões anteriores (Arduino Uno e Arduino Duemilanove) não permitiam esse controle. No entanto, o comportamento default é a comunicação com uma porta serial emulada (ARDUINO, 2012f).

Existem diversas variações oficiais do Arduino. Como exemplo, podemos citar o Arduino LilyPad (ARDUINO, 2012e), criado para projetos de computação vestível. Ele pode ser costurado a roupas e pode ser lavado junto com os tecidos. Outro exemplo é o Arduino Mega (ARDUINO, 2012b), que possui mais pinos de I/O (entrada e saída), memória e capacidade de processamento.

O Arduino Uno possui um processador ATmel ATMega328 de 16MHz, 32kB memória Flash, 2kB memória RAM e 1kB memória EEPROM. O processador possui 32 registradores de 8 bits e timers diversos. Para comunicação, disponibiliza 14 portas digitais, sendo que 6 podem ser utilizadas com PWM (pulse-width modulation)[1] e 2 são ocupadas com os pinos RX (recepção) e TX (transmissão) da comunicação serial. Além delas, mais 6 portas analógicas estão disponíveis. A alimentação pode ser feita através de uma fonte AC (de 6v a 12v) ou através da porta USB.

A comunicação com computadores pode ser feita através de um controlador USB serial já incluído na placa. No entanto, é possível expandir essa comunicação através de dispositivos chamados Shield (ARDUINO, 2012g). Shields são a forma oficial de expandir as funcionalidades do Arduino, adicionando interface de rede, wireless, bluetooth, zigbee, etc.

3.1.2 Software

Por ter não-programadores como público-alvo, o Arduino provê uma interface simplificada de acesso ao *hardware*. Inspirada em *processing* (FRY; REAS, 2013), a linguagem utilizada é C e o compilador é baseado no conhecido GCC, o gcc-avr (ARDUINO, 2012a). O IDE é escrita em Java com o código-fonte disponível (ARDUINO, 2012d). Todo o código é open-source e multiplataforma, rodando em Microsoft Windows, GNU/Linux e Apple MacOS. O IDE pode ser visto na Figura 36 rodando em um ambiente GNU/Linux.

Um programa para o Arduino possui duas funções obrigatórias: “setup” e“loop”. A função“setup” é executada uma única vez após o início do dispositivo. A função“loop” é executada repetidamente após o“setup”, até o dispositivo ser desligado ou reiniciado. O Código-Fonte 1 mostra a declaração das 2 funções utilizadas, na linguagem C (ou C++).

```
1 void setup () {
2 }
3
4 void loop () {
5 }
```

Código-Fonte 1: Template (mínimo) para programas que rodam no Arduino

[1] é uma técnica de modulação que se conforma a largura do pulso, formalmente duração do pulso, com base numa informação de sinal modulado

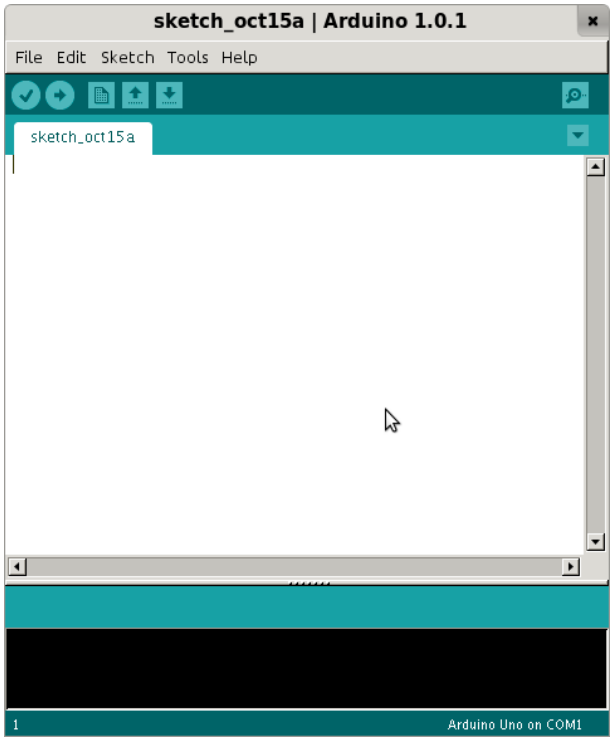

Figura 36 – IDE do Arduino

3.1.3 Piscando um LED

Comumente, cria-se um "Hello World" como primeiro programa em uma linguagem de programação que se está aprendendo. Na computação embarcada, coloca-se um LED piscando em intervalos regulares com o mesmo objetivo. Isso serve para verificar o funcionamento básico do *hardware* e a validação do ambiente de desenvolvimento escolhido.

Para esse programa simples, no Arduino, são necessários:

- Arduino (no teste, utilizado o UNO)
- LED (LEDs padrão - Vazados GREEN DIFFUSED, Chicago Miniature ref 4303F5)
- resistor (de 1k ohm)

Um LED possui um polo positivo, chamado ânodo, e um polo negativo, chamado cátodo. É possível identificar no LED cada um dos polos conforme a Figura 37.

Montar o circuito é simples, demonstrado na Figura 38. O cátodo do LED é ligado ao terra (*ground*, indicado como GND no Arduino). O ânodo é ligado ao resistor, que é ligado ao pino digital 13 do Arduino.

Para desenhar projetos com o Arduino, existe um *software* chamado Fritzing. Ele torna desenhos esquemáticos menos técnicos e mais atraentes (FRITZING, 2012). A Figura 39 é a mesma que a Figura 38, porém utilizando esse *software*.

O *software* a ser enviado para o Arduino para piscar o LED é o Código-Fonte 2. Para enviar o programa, basta plugar o Arduino na USB e clicar no botão "Upload"no IDE.

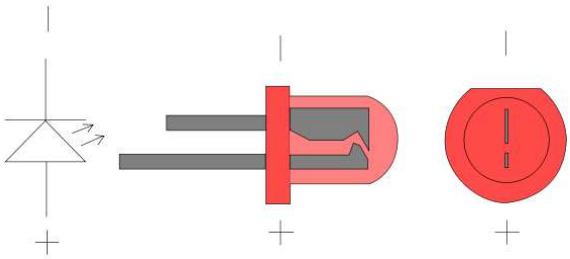

Figura 37 – LED

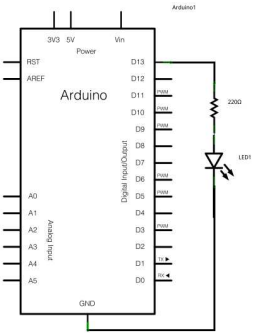

Figura 38 – Circuito para controle de LED

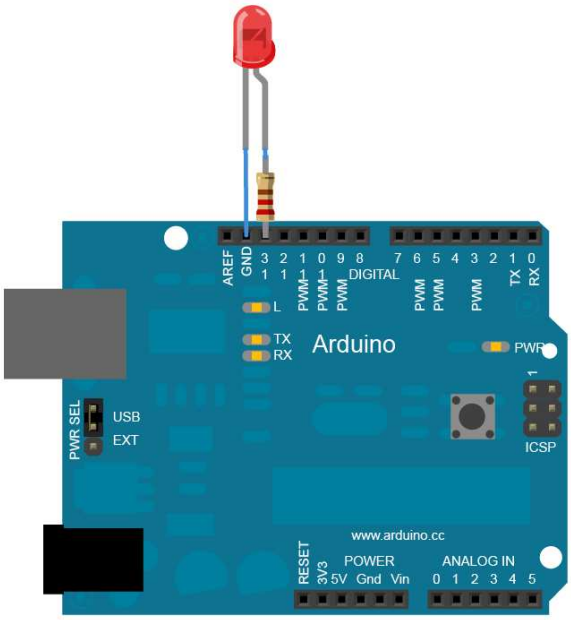

Figura 39 – Circuito para controle de LED com o Arduino

3.1.4 Comunicação Serial

O Arduino é conectado ao PC usando uma porta USB. É possível estabelecer comunicação entre os dispositivos da mesma forma que utilizando-se uma porta serial. O Arduino usa os pinos 0 e 1 como RX e TX, que também estão ligados na porta USB, permitindo essa comunicação. Um exemplo dessa comunicação pode ser visto no Código-Fonte 3.

Basicamente, utilizam-se duas funções para ler e escrever dados na porta serial: “Serial.println"e

```
1  /*
2    Blink
3    Turns on an LED on for one second, then off for one second, repeatedly.
4
5    This example code is in the public domain.
6   */
7
8  // Pin 13 has an LED connected on most Arduino boards.
9  // give it a name:
10 int led = 13;
11
12 // the setup routine runs once when you press reset:
13 void setup() {
14   // initialize the digital pin as an output.
15   pinMode(led, OUTPUT);
16 }
17
18 // the loop routine runs over and over again forever:
19 void loop() {
20   digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
21   delay(1000);               // wait for a second
22   digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
23   delay(1000);               // wait for a second
24 }
```

Código-Fonte 2: Controle de LED usando Arduino

```
1  /*
2    DigitalReadSerial
3   Reads a digital input on pin 2, prints the result to the serial monitor
4
5   This example code is in the public domain.
6   */
7
8  // digital pin 2 has a pushbutton attached to it. Give it a name:
9  int pushButton = 2;
10
11 // the setup routine runs once when you press reset:
12 void setup() {
13   // initialize serial communication at 9600 bits per second:
14   Serial.begin(9600);
15   // make the pushbutton's pin an input:
16   pinMode(pushButton, INPUT);
17 }
18
19 // the loop routine runs over and over again forever:
20 void loop() {
21   // read the input pin:
22   int buttonState = digitalRead(pushButton);
23   // print out the state of the button:
24   Serial.println(buttonState);
25   delay(1);        // delay in between reads for stability
26 }
```

Código-Fonte 3: Comunicação serial (através da porta USB) usando Arduino

"Serial.readln" (linha 24, 3). Para isso, dentro da função de inicialização "setup", o método "begin" indica a velocidade da transferência serial. É a inicialização da comunicação, abstraída pela API. Na função "loop", a cada ciclo é efetuada a leitura do estado do botão (através da função "digitalRead") e enviado via serial (através do método "println"). Todos os métodos acessados da classe "Serial" são métodos estáticos (não é criada uma instância do objeto).

Qualquer linguagem de programação que permita comunicação serial poderá comunicar com o Arduino, independente do sistema operacional. Obviamente, será necessário respeitar

a forma que cada sistema operacional trata acesso, instalação de drivers e nomeclatura dessas portas.

3.2 Texas Instruments Launchpad

A Launchpad é uma placa criada pela Texas Instruments. Ela provê o que o desenvolvedor precisa para iniciar o desenvolvimento de aplicações a um custo bastante baixo (INSTRUMENTS, 2012b). Atualmente, o preço dessa placa, nos EUA, é de apenas US$4.30[2] (INSTRUMENTS, 2012a).

3.2.1 Hardware

O modelo utilizado nos testes é o MSP430, que utiliza um processador identificado como "msp430g2231". Essa informação é importante no desenvolvimento do *software* desse processador, pois será necessário informar a versão correta durante a compilação e gravação do programa no chip.

O MSP430 utilizado provê 14 pinos, apesar da placa prover suporte para até 20. São 8 pinos de uso geral, numerados de P1.0 a P1.7, sendo que 2 deles estão ligados à interface USB para prover comunicação serial (P1.1 e P1.2). Os pinos P1.0 e P1.6 estão ligados a 2 LEDs já disponíveis na placa.

3.2.2 Software

Para fazer download do Code Composer Studio (CCS) da Texas Instruments para a LauchPad, é necessário fazer um cadastro, informando dados básicos do usuário. A página de download avisa que a versão gratuíta do CCS tem limitações de tempo, tamanho de código compilado, etc. Existem versões para GNU/Linux e Microsoft Windows.

Outra alternativa dada pelo fabricante é utilizar um compilador livre, o GCC, adaptado para a plataforma. Essa versão do GCC é o MSPGCC.

3.2.3 Piscando um LED

O *software* a ser enviado para o Arduino para piscar o LED é o Código-Fonte 4. No exemplo, utilizou-se o LED que está na própria placa, mas é possível ligar um LED externo seguindo o mesmo esquema do utilizado com o Arduino[3].

A compilação utilizando o MSPGCC utiliza o seguinte comando:

```
msp430-gcc -mmcu=msp430g2231 hello.c -o main.elf
```

[2] preço indicado para o ano de 2012
[3] Como mostrado na Figura 38 e na Figura 39

"msp430g2231"é a identificação do chip que está na Lauchpad.

Para enviar o código para o chip, utilizamos o seguinte comando:

```
mspdebug rf2500
```

O programa aberto terá uma interface por linha de comando (CLI - *command line interface*). Os seguintes comandos devem ser informados:

```
prog main.elf
run
```

3.2.4 Comunicação Serial

A comunicação serial utilizando a Launchpad necessita de controle específico de interrupções. Para isso, utilizam-se as interrupções disponíveis nativamente na plataforma chamadas *Timer_A* e *Port_1*. A comunicação precisa ser controlada *byte* a *byte*. A implementação completa está disponível no Anexo 6.1. O código é bastante complexo, necessitando desabilitar e habilitar interrupções durante todo o processo de comunicação (OLSON, 2012).

Outra forma de comunicação é usando o comando *printf*. O comando funciona da mesma forma que o comando que escreve na tela, mas os *bytes* são enviados via porta serial. Essa funcionalidade, no entanto, está disponível apenas em modo de *debug*, com a ferramenta de gravação mspdebug sendo executada (STATUSBITS, 2012).

3.3 Expansão de Portas de I/O

O Arduino possui quatorze (14) portas para I/O (entrada e saída). Para a implementação, serão necessárias mais portas para controlar cada um dos LEDs. Para expandir a quantidade de portas é possível utilizar o Circuito Integrado (CI) PCF8574, que serve como expansão de portas. Com ele, é possível utilizar duas portas e controlar de oito (8) a sessenta e quatro (64) portas, tanto de entrada como de saída.

A pinagem do PCF8574 (PHILIPS, 2002) pode ser vista na Figura 40. Os pinos A0, A1 e A2 servem para endereçamento do CI, os pinos P0 a P7 servem como saídas ou entradas (bits, apenas leituras 0 ou 1), e os pinos SDA e SCL farão a comunicação com o microcontrolador (o Arduino). SCL é *serial clock*, enquanto SDA é *serial data*.

Pode-se ver o endereçamento conforme os pinos A[0-2], representados na Figura 41, onde L representa *LOW* (0v, *ground*, GND ou terra) e H representa *HIGH* (5v ou VCC).

Ao ligar dois CIs no Arduino, como na Figura 42, os CIs recebem os endereços zero (0) e um (1), conforme a pinagem em A2, A1 e A0 (000b = 0 e 001b = 1). Nos pinos P2

```
#include  <msp430g2231.h>

 //Initialize variables. This will keep count of how many cycles
 //between LED toggles
 unsigned int i = 0;

void delay(unsigned int ms)
{
    //http://www.threadabort.com/?p=26
    while (ms--)
    {
        //set for 16Mhz change it to 1000 for 1 Mhz
        __delay_cycles(1000);

        //set for 16Mhz change it to 1000 for 1 Mhz
        //__delay_cycles(16000);
    }
}

void main(void)
{
  //Stop watchdog timer. This line of code is needed at the beginning
  //of most MSP430 projects.
  WDTCTL = WDTPW + WDTHOLD;

  P1DIR |= 0x01;
  //P1DIR is a register that configures the direction (DIR) of a port
  //pin as an output or an input.

  //To set a specific pin as output or input, we write a '1' or '0'
  //on the appropriate bit of the register.

  //P1DIR = <PIN7><PIN6><PIN5><PIN4><PIN3><PIN2><PIN1><PIN0>

  //Since we want to blink the on-board red LED, we want to set the
  //direction of Port 1, Pin 0 (P1.0) as an output

  //We do that by writing a 1 on the PIN0 bit of the P1DIR register
  //P1DIR = <PIN7><PIN6><PIN5><PIN4><PIN3><PIN2><PIN1><PIN0>
  //P1DIR = 0000 0001
  //P1DIR = 0x01     <-- this is the hexadecimal conversion of 0000 0001

  for (;;)
  //This empty for-loop will cause the lines of code within to
  //loop infinitely
  {
    //Toggle P1.0 using exclusive-OR operation (^=)
    P1OUT ^= 0x01;

    //P1OUT is another register which holds the status of the LED.
    //'1' specifies that it's ON or HIGH, while '0' specifies that
    //it's OFF or LOW
    //Since our LED is tied to P1.0, we will toggle the 0 bit of
    //the P1OUT register

    //Delay between LED toggles.
    delay(500);
  }
}
```

Código-Fonte 4: Controle de LED usando LaunchPad

e P3 de cada um dos CIs são ligados aos LEDs, da mesma forma que com pinos comuns do Arduino. O código-fonte para controlar esses LEDs pode ser visto no Código-Fonte 5.

A biblioteca *Wire* do Arduino utiliza apenas os pinos analógicos 4 e 5 para essa implementação. No entanto, não é difícil encontrar uma implementação via *software* usando outros pinos (FAT16LIB, 2008). A desvantagem fica na performance: enquanto a

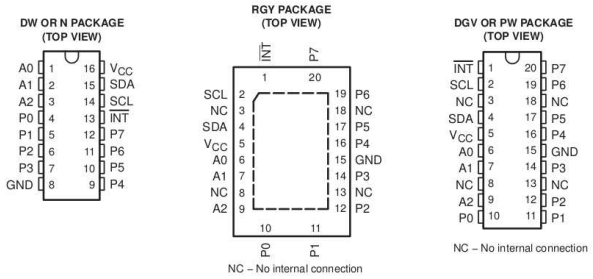

Figura 40 – Pinagem do CI PCF8574

ADDRESS REFERENCE

INPUTS			I²C-BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	32 (decimal), 20 (hexadecimal)
L	L	H	33 (decimal), 21 (hexadecimal)
L	H	L	34 (decimal), 22 (hexadecimal)
L	H	H	35 (decimal), 23 (hexadecimal)
H	L	L	36 (decimal), 24 (hexadecimal)
H	L	H	37 (decimal), 25 (hexadecimal)
H	H	L	38 (decimal), 26 (hexadecimal)
H	H	H	39 (decimal), 27 (hexadecimal)

Figura 41 – Endereçamento do CI PCF8574

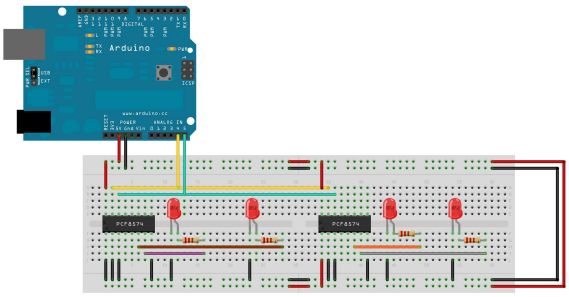

Figura 42 – Conexões entre o Arduino e o PCF8574 para controle de LEDs

biblioteca *Wire* do Arduino consegue utilizar o protocolo I2C a 400kHz, a implementação via *software* chega aproximadamente a 65kHz.

Essa implementação de controle de LED usando apenas duas portas será reproduzida inúmeras vezes para atingir o número necessário de LEDs para a proposta de solução.

3.3.1 Problemas na Implementação

Durante a implementação do *software* embarcado, testes feitos em implementações de I2C implementado via *software* mostraram-se insatisfatórias. Esse teste, não efetuado anteriormente, mostrou-se falho para a solução proposta originalmente[4]. A solução não era estável, deixando de controlar os *chips* após alguns segundos de operação.

[4] Solução proposta por (FAT16LIB, 2008)

```
#include <Wire.h>

#define CHIP0 32 // Se PCF8574A mude 32 para 56
#define CHIP1 33 // Se PCF8574A mude 33 para 57

void setup()
{
    Wire.begin();
}

void loop()
{
    Wire.beginTransmission(CHIP0);      // Begin the transmission to PCF8574
    Wire.write(3);                      // Send the data to PCF8574
    delay(1000);                        // Wait for a second
    Wire.write(2);                      // Send the data to PCF8574
    Wire.endTransmission();             // End the Transmission
    delay(1000);                        // Wait for a second

    Wire.beginTransmission(CHIP1);      // Begin the transmission to PCF8574
    Wire.write(3);                      // Send the data to PCF8574
    delay(1000);                        // Wait for a second
    Wire.write(2);                      // Send the data to PCF8574
    Wire.endTransmission();             // End the Transmission
    delay(1000);                        // Wait for a second
}
```

Código-Fonte 5: Controle dos LEDs para o projeto da Figura 42

Alternativamente, foi reutilizada parte do código utilizados em produtos da ThingM (THINGM, 2010) disponível sob licença GPL v2 (FSF, 1992) em um repositório do Google Code (BLINKM, 2012). Com essa abordagem, qualquer 2 pinos do Arduino podem ser usados para esse tipo de comunicação, permitindo expandir para mais de 500 portas de entrada ou saída digital, muito mais do que o necessário para a construção do protótipo.

Outra vantagem dessa segunda solução é a similaridade com a biblioteca padrão do Arduino, tornando desnecessário grandes alterações no *software* testado anteriormente. Um exemplo de código pode ser visto no Código-Fonte 6. Para utilizar outras portas, basta criar novas instâncias da classe SoftI2CMaster com os pinos correspondentes na inicialização. Note que o código utiliza orientação a objetos da linguagem C++, não apenas C.

3.4 Considerações Finais

Neste capítulo, foram testadas duas alternativas para *hardware* e *software* embarcado, implementando exemplos de 2 funcionalidades principais para a proposta de solução.

No primeiro comparativo, verificou-se que o controle de LED é simples em ambas a plataformas testadas, o Arduino e a LaunchPad. Nessa implementação constatou-se que a IDE para Arduino é aberta e sem restrições. A IDE para LaunchPad possui limitações ou, no caso da tecnologia testada, provê apenas o compilador GCC adaptado.

Um grande diferencial foi encontrado na implementação da comunicação serial (via porta USB). Enquanto Arduino abstrai boa parte da complexidade, tornando a comunicação

```
#include <SoftI2CMaster.h>

const byte sdaPin = 12; // green
const byte sclPin = 11; // white

SoftI2CMaster i2c = SoftI2CMaster( sdaPin,sclPin );

void setup() {
}

void loop() {
  i2c.beginTransmission(0x20);
  i2c.send(0b11111111);
  i2c.endTransmission();

  i2c.beginTransmission(0x21);
  i2c.send(0b11111111);
  i2c.endTransmission();
}
```

Código-Fonte 6: Usando I2C via *software* em portas não-padrão

apenas uma implementação de leitura e escrita, utilizando LaunchPad o desenvolvedor precisa manipular interrupções e outros detalhes diretamente no código.

Tomando como base os testes realizados, verifica-se que a plataforma Arduino possui implementação simplificada das funcionalidades necessárias para o protótipo a ser construído. Como a proposta deste trabalho não necessita de controle avançado de interrupções de *hardware*, optou-se por utilizar o Arduino para implementação do protótipo.

4 Protótipo “Lighted Strings”

Diversas alternativas tecnológicas foram demonstradas na Seção 1.1. Buscando responder à questão de pesquisa formulada na Seção 1.2, o presente trabalho propõe a construção de uma interface de *hardware* e produção de *software* para auxiliar no aprendizado dos acordes de guitarra.

Inspirado no projeto *Lighted Keys* descrito na Seção 1.1, buscou-se construir uma versão adaptada para guitarras. A solução proposta foi incluir LEDs correspondentes a cada corda em cada traste do instrumento. Com isso, pode-se projetar no próprio instrumento a notação de cada acorde na notação conhecida como Diagrama de Acordes, revisada na Seção 2.6.4. A Figura 43 demonstra como a solução foi planejada em um *fretboard*.

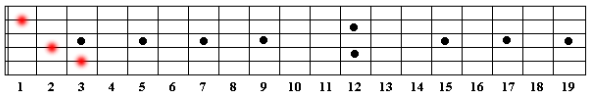

Figura 43 – Planejamento da solução em um *fretboard* - LEDs em vermelho

Além do braço do instrumento, o projeto inclui também LEDs no corpo da guitarra. Relacionados a cada corda, eles indicam quais cordas devem ser tocadas em determinado momento. Com isso, é possível indicar o tempo e o ritmo do instrumento.

Os computadores hoje são dotados de uma interface USB que provê alimentação de energia - 5v, 500-900 mA (USF-IF,) - suficientes para alimentar diversos LEDs. No entanto, é necessário utilizar computação embarcada para controlar independentemente esses LEDs. No Capítulo 3 verificou-se que a plataforma Arduino representa uma solução adequada e suficiente para controle desses LEDs. Dessa forma, apenas um cabo ligando o computador à guitarra será necessário. Para controlar os LEDs, o Arduino precisa de um código embarcado que precisa ser desenvolvido usando a linguagem de programação C.

Controlando o dispositivo embarcado, um *software* executando em um *desktop* envia comandos utilizando a interface USB, orquestrando o conjunto de LEDs instalados na guitarra. Complementando a funcionalidade, a tablatura será exibida (quando disponível) em conjunto com um vídeo, que pode ou não conter áudio. Completando a funcionalidade, o *software* busca os comandos a serem enviados para a guitarra a partir de arquivos-texto em formato pré-definido, seguindo padrões de legendas de vídeos (porém com *bytes*, não palavras).

Além de permitir enviar músicas, o *software* permite indicar diferentes itens na guitarra, como ligar todos os LEDs ou indicar cada uma das cordas. Dessa forma, qualquer combinação

dos LEDs é permitida, garantindo flexibilidade para além das músicas e acordes existentes. Isso pode ser útil para apresentar o instrumento ou nomear/numerar as cordas para o aluno. Nesses casos, a tablatura não está disponível, sendo substituída por outra representação na interface.

4.1 Arquitetura da Solução

O funcionamento do protótipo inicia através da interação do usuário com a interface gráfica gerada pelo *software*. Isso é representado na Figura 44, detalhe 1.

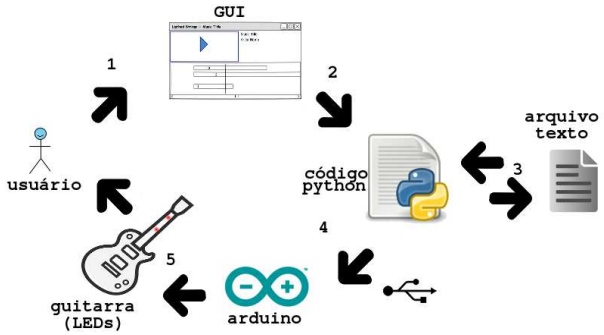

Figura 44 – Esquemático da arquitetura da solução proposta

Partindo das informações recebidas via interface gráfica, representadas no esquemático pelo detalhe 2, o código Python busca informações de arquivos-texto, indicado no detalhe 3. A interface é atualizada com os dados carregados e com o vídeo. Os arquivos com informações referentes aos dados que serão enviados para o *hardware* são carregados e interpretados nesse momento. Utilizando de uma porta de comunicação USB, comumente encontrada em PCs, o código Python faz comunicação com o Arduino, conforme mostrado no detalhe 4. O código embarcado presente interpreta os dados recebidos e ativa ou desativa as saídas digitais do Arduino.

O Arduino teve suas portas expandidas utilizando-se de chips PCF8574, conforme descrito na Seção 3.3. Ligadas aos LEDs incluídos na guitarra, indicam para o usuário o que o autor dos arquivos planejou, seja uma aula ou uma música, em sincronia com o vídeo e demais recursos presentes na GUI. Essa interface com o instrumento real é o diferencial desse protótipo e é indicada na Figura 44 pelo detalhe 5.

4.2 A Guitarra

Para construção do protótipo da solução sugerida foi necessário incluir 6 LEDs (um para cada corda) em cada casa (*fret*) do braço da guitarra. Existem LEDs de diferentes tamanhos, conforme pode ser visto na Figura 45. Foi utilizado o LED de menor tamanho dos comparados (LED 1 na Figura 45). Os identificados com os números 2 e 3 (na

Figura 45) são grandes demais para a solução proposta. O LED número 3 (na Figura 45) é um LED RGB, que permitiria o uso de diversas cores em cada corda. O LED utilizado pode ser visto em escala na Figura 46, juntamente com o braço da guitarra que será adaptado.

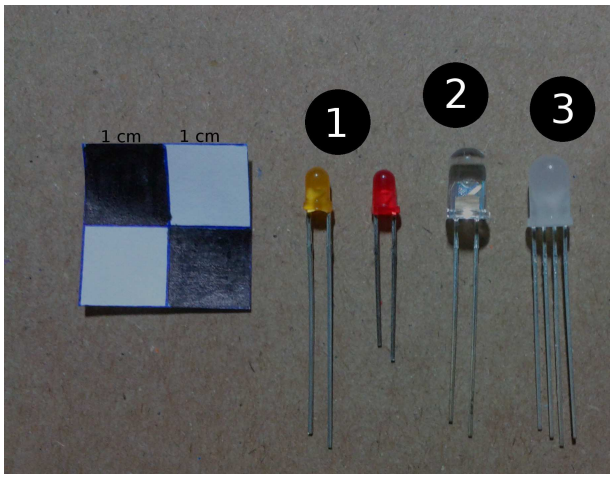

Figura 45 – Comparação entre tamanhos de LEDs disponíveis, incluindo um LED RGB (3)

Os LEDs são encaixados ao longo do braço. Para 12 casas[1], foram necessários 72 LEDs. Para todo o braço da guitarra (21 casas), seriam necessários 126 LEDs. Eles foram encaixados conforme a Figura 46. As linhas verticais são os trastes.

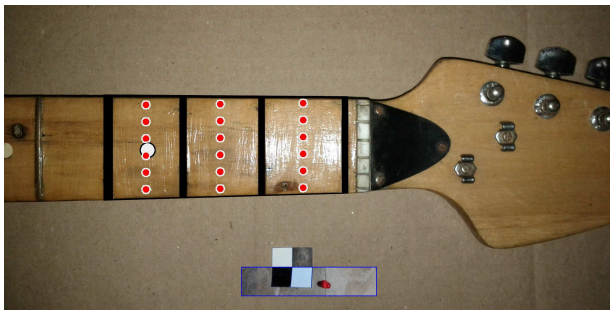

Figura 46 – Representação, no braço da guitarra, da posição onde os LEDs estão instalados

Além do braço, também existem LEDs no corpo da guitarra. Um para cada corda, indicam quais cordas devem ser tocadas no momento. A forma que eles são ligados pode ser vista na Figura 47. Para o corpo são necessários mais 6 LEDs.

Para a instalação dos LEDs, foi retirada parte do braço da guitarra, mas não o suficiente para atravessá-lo. Esse pedaço foi substituído por outro com os LEDs. O braço não foi atravessado para evitar que perca a resistência, conforme a Figura 48. Os cabos para ligar os LEDs saem por cima do braço (a mão para tocar o instrumento fica no lado de baixo). O LED não deve modificar a estrutura da guitarra, uma vez que o tato

[1] Nas 12 primeiras casas permitem todas as notas. A partir da 12ª casa, as notas se repetem uma oitava acima

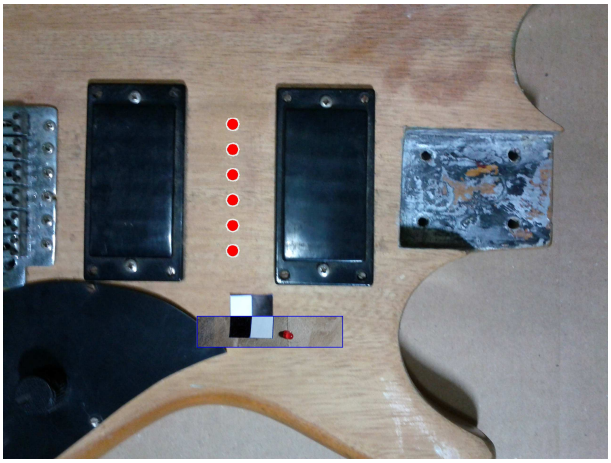

Figura 47 – Representação, no corpo da guitarra, de posição onde os LEDs serão instalados

é importante para o aprendizado do instrumento. Assim, LED foi envolvido em resina transparente, permitindo a visualização do mesmo sem alteração na superfície.

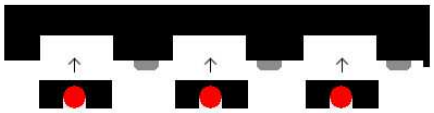

Figura 48 – Representação da lateral do braço da guitarra, indicando a forma dos LEDs instalados

Para a produção do protótipo, foi necessária uma fase de experimentação para obtenção das modificações necessárias para a guitarra. O trabalho não se propõe a construir inteiramente o instrumento, mas adaptar um já em construção. Diversos materiais estão disponíveis com diferentes características para inclusão dos LEDs no protótipo.

A inclusão de sulcos no braço da guitarra precisa de um ponto inicial como guia, como mostrado na Figura 49. A partir dele, utilizando uma furadeira na posição indicada na Figura 50, foi aberto o primeiro sulco, na primeira casa, como demonstrado na Figura 51. Para manter a guitarra firme durante o corte, foi utilizada uma morsa. O primeiro corte foi efetuado sem problemas. No entanto, tentativas posteriores de modificar o tamanho (em largura e profundidade) danificaram parte da guitarra, mostrado na Figura 52, que necessitou de reparos.

Após o dano causado na guitarra, o método para alterar a estrutura foi modificado. Utilizando-se uma serra pequena e fina (comumente utilizada para serrar estruturas metálicas) fez-se cortes verticais no braço na profundidade necessária, distanciados pelo espaço a ser reservado para os LEDs. Após, foram feitos outros cortes entre eles, quebrando os pedaços que não seriam mais utilizados na guitarra. O resultado, que pode ser visto na Figura 53, ficou com melhor qualidade.

A primeira tentativa de preencher o sulco foi com um preenchimento usando compensado de madeira (uma espécia de imitação feita a partir da serragem da madeira). No entanto,

Figura 49 – Ponto usado como guia para primeiro corte

Figura 50 – Posição da furadeira no primeiro corte, com guitarra presa à morsa

essa execução não se mostrou satisfatória, uma vez que, ao perfurá-la para inclusão dos LEDs, o material se mostrou menos resistente que o necessário. A produção pode ser vista na Figura 54.

A segunda tentativa foi em madeira, que é mais resistente e menos maleável do que o compensado. A solução também não mostrou-se fácil, pois ao torná-la pequena o suficiente para incluí-la no braço da guitarra, parte dela se partiu. Novamente, a solução testada não atendeu às expectativas para a produção.

A última tentativa testada foi utilizando-se uma base feita de pedaços de uma matriz de contatos. Essa matriz foi cortada de tamanho compatível com o espaço disponível para os LEDs. Com ela, torna-se mais fácil também posicionar os LEDs, uma vez que ela já tem furos apropriados para encaixar os componentes. A forma como o LED se encaixa pode ser vista na Figura 55. Com o tamanho adotado, o espaço para os fios que precisam ser ligados aos LEDs também ganham espaço de forma mais organizada, facilitando a solda com a existência de uma base fixa.

Dessa forma, o espaço disponível na guitarra não fica completamente preenchido.

Figura 51 – Primeiro corte realizado no braço da guitarra

Figura 52 – Dano causado na guitarra durante produção

Figura 53 – Espaços abertos para os LEDs no braço da guitarra

Para torná-lo uniforme, foi utilizada resina cristal (que possui aspecto transparente). Essa resina é líquida, mas ao adicionar um catalizador, inicia uma reação química que a torna gelatinosa e, após algum tempo, sólida. No processo, o volume da resina diminui levemente, tornando necessário acrescentar mais resina durante o processo. Os espaços abertos no braço da guitarra foram limitados com fita adesiva, evitando que a resina escape pelas laterais. A implementação, com a resina ainda em estado gelatinoso, pode ser vista na Figura 56, onde percebe-se a presença pelo reflexo da luz e pelas bolhas de ar.

Figura 54 – Produção falha do suporte dos LEDs em compensado

Figura 55 – Base recortada que serve de apoio aos LEDs

Figura 56 – LEDs na guitarra, já cobertos com a resina

Tendo a solução com resina se mostrado satisfatória, ela foi adotada. Não foram realizados testes com outros materiais.

4.3 Protocolo de Comunicação

A comunicação entre o computador e o dispositivo é realizada atravéd de uma porta USB. O Arduino já possui uma biblioteca para comunicação serial, conforme descrito na Seção 3.1.4, então foi necessário definir como representar os estados dos LEDs, sem

necessidade de detalhes da comunicação USB.

As mensagens enviadas são decodificadas via *software*, detalhado na Seção 4.4. O formato da comunicação está definido independentemente pois, sendo aberto, será possível criar outros *softwares* que farão uso do mesmo *hardware*. Qualquer programador ou entusiasta poderá criar outros *softwares* utilizando-se do trabalho desenvolvido, independente de linguagem de programação ou sistema operacional.

Na proposta de solução atual, apenas 1 bit é suficiente para indicar cada LED, sendo 0 para desligado e 1 para ligado. Na guitarra demonstrada neste capítulo, são 72 LEDs para o braço e 6 para o corpo da guitarra. Logo, 10 bytes seriam suficientes para toda a informação ser enviada. No entanto, existem guitarras com até 24 casas, incluindo mais 18 LEDs, totalizando 19 bytes.

Buscando simplificar a implementação, são enviados 13 bytes, sendo o primeiro para o corpo da guitarra e os 12 bytes seguintes para cada casa. Dessa forma, apenas 6 bits são utilizados em cada byte, mas a compreensão das mensagens fica simplificada.

Dessa forma, o protocolo pode enviar os 13 bytes e o *software* embarcado deve ligar ou desligar cada um dos LEDs conforme os valores recebidos. Do segundo byte, o primeiro bit indica a primeira casa do braço da guitarra, na primeira corda, conforme a numeração da Figura 9. Os bits seguintes indicam as próximas cordas, na mesma casa. Os bytes seguintes representam da mesma forma a casa seguinte. Assim sucessivamente até a 12ª casa.

4.4 Software Embarcado

O *software* que controla a guitarra mantém o estado dos LEDs e responde às mensagens definidas na Seção 4.3. Recebe os 13 bytes necessários (são 6 bits para cada traste, mas a unidade mínima para transferência via serial na API utilizada é 1 byte). São mantidos os estados de cada LED em memória e o Arduino é capaz de mapear a localização de cada um deles através das diversas placas de expansão utilizadas, conforme descrito na Seção 3.3, já que as mensagens definidas preveem abstração desse aspecto da implementação. É esse *software* que fará a tradução das mensagens recebidas para os comandos que recebem os LEDs. Os dados são recebidos via serial em blocos de 13 bytes, utilizando a função *readBytes* da biblioteca *built in* do Arduino. Os dados desses bytes são repassados para um *array* com 2 objetos do tipo *SoftI2CMaster* - os 8 primeiros são repassados para o primeiro objeto, os demais para o segundo.

O *array* com os objetos *SoftI2CMaster* é inicializado com os pinos correspondentes a cada uma das interfaces conectadas no Arduino. Dessa forma, utilizando 4 pinos, controlamos todos os 13 conjuntos de 6 LEDs, fornecendo uma interface serial para comunicação com o PC (computador *desktop*).

4.5 Expansão de Portas de Comunicação

Diversos testes foram executados para a construção do código que controla os chips para expansão das portas. O primeiro desafio foi testar o endereçamento dos chips, utilizando os 3 bits disponíveis para isso. A implementação dos endereços zero (0), indicado pela letra A na Figura 57 e um (1), e pela letra B na mesma figura.

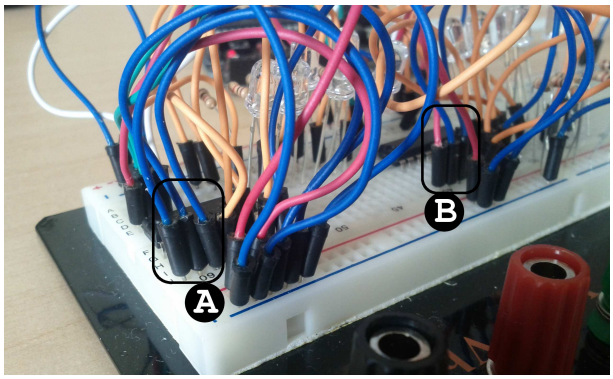

Figura 57 – Endereçamento do PCF8574: "A" indica o endereço zero (0) e "B" indica o um (1)

O segundo teste foi no código criado para o *software* embarcado. Como o *chip* utilizado possui apenas 3 portas para endereço, apenas 8 chips podem ser controlados pelo mesmo par de portas ($2^3 = 8$). Dessa forma, o 9º chip recebe novamente o primeiro endereço (zero), mas em outros pinos de controle. A diferenciação entre cada um dos chip com o mesmo endereço ficou à cargo da implementação do código do *software* embarcado. A Figura 58 mostra um Arduino controlando 3 chips PCF8574, utilizando 2 pinos. O primeiro e o último *chip* possuem o mesmo endereço, mas estão sendo controlados por pinos diferentes.

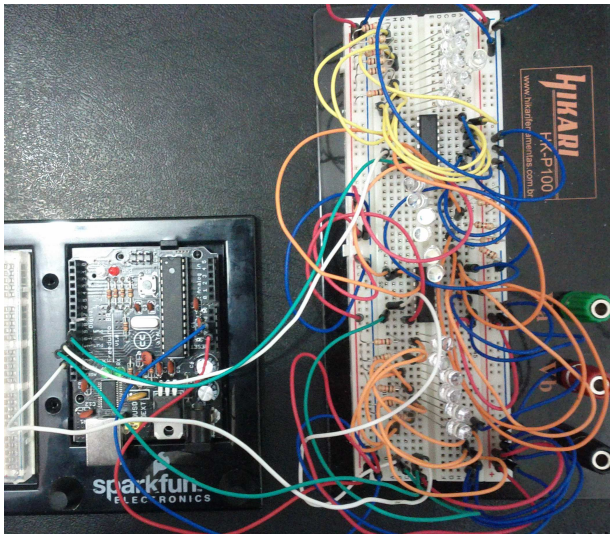

Figura 58 – Arduino controlando 3 chips PCF8574, utilizando 4 pinos

4.6 Software

A interface provê diferentes possibilidades para o aprendizado, como vídeo e partitura, além da interface com o *hardware*. Em trabalhos citados no Capítulo 1, vê-se que mais de uma forma de representar uma nota em uma música ou um tópico em uma aula de música não é incomum. Este trabalho segue essa abordagem.

A interface é dividida em 3 partes principais, conforme Figura 59. A primeira delas exibe um vídeo, que pode ser o videoclipe da música ou uma vídeo-aula. A sincronização das mensagens enviadas para a guitarra é realizada através desse vídeo, no detalhe 1. As informações e controles do vídeo estarão disponíveis ao lado do vídeo, conforme demonstrado na Figura 59, detalhe 2. Os controles permitirão selecionar/abrir o arquivo com vídeo e demais informações e os comandos para navegar e iniciar/pausar o vídeo. As informações exibidas serão extraídas de arquivos-texto disponíveis juntamente com o vídeo. O detalhe 3 na Figura 59 é o espaço reservado que permite incluir a partitura, quando disponível, ou a representação do instrumento. Essa área poderá, em trabalhos futuros, exibir diversar representações musicais, como diagramas de acordes ou a partitura das canções.

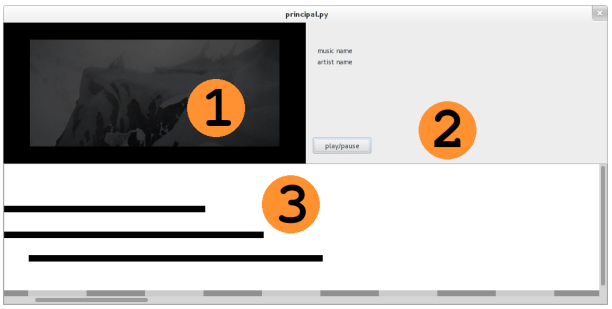

Figura 59 – *Screenshot* do *software*, mostrando áreas de vídeo (1), meta-informações (2) e representação musical (3)

Para indicar quais informações serão mostradas, o protótipo utiliza um arquivo compactado ou um diretório contendo todos os demais arquivos. Os arquivos obrigatórios são um arquivo texto contendo o protocolo a ser enviado para o *hardware*, seguindo um padrão conforme utilizado em legendas de filmes, onde o tempo é indicado juntamente com uma sequência de caracteres (*String*) a ser exibida (MATROSKA, 2011). Outro arquivo a ser utilizado é a imagem da partitura(ou a partitura representada em ASCII 2.6.3.1) e o vídeo a ser exibido.

Essa proposta de *software* indica diferentes formas para representar uma música ou uma aula de música sem tornar obrigatório o uso de todos os recursos disponíveis. Assim, fica a cargo do autor de cada aula utilizar-se dos recursos que achar necessário, permitindo flexibilidade ao código.

O *software* que controla o *hardware* desenvolvido não inclui importação de arquivos MIDI e/ou MusicXML. Essa possibilidade fica aberta para futura implementação pois,

poder-se-ia utilizar arquivos disponíveis na Internet e exportados de outros *softwares*, como os demonstrados na Seção 2.7.

4.6.1 Codificação

O código foi organizado em diferentes classes para separar as responsabilidades e permitir futuras melhorias em cada uma delas. O *design pattern* (ou "padrão de projeto") *observer* foi utilizado para a comunicação entre as classes mantendo baixo acoplamento (GAMMA et al., 1994), permitindo que vários objetos "escutem" esses eventos. Essa abordagem foi adotada após a primeira implementação do *software*, observando o comportamento dos componentes implementados e o seu acoplamento.

A forma como o usuário se relaciona com o *software* é através de uma interface gráfica (*graphical user interface* ou GUI). As relações com as diversas classes do sistema pode ser vista no Diagrama de Atividades da Figura 60.

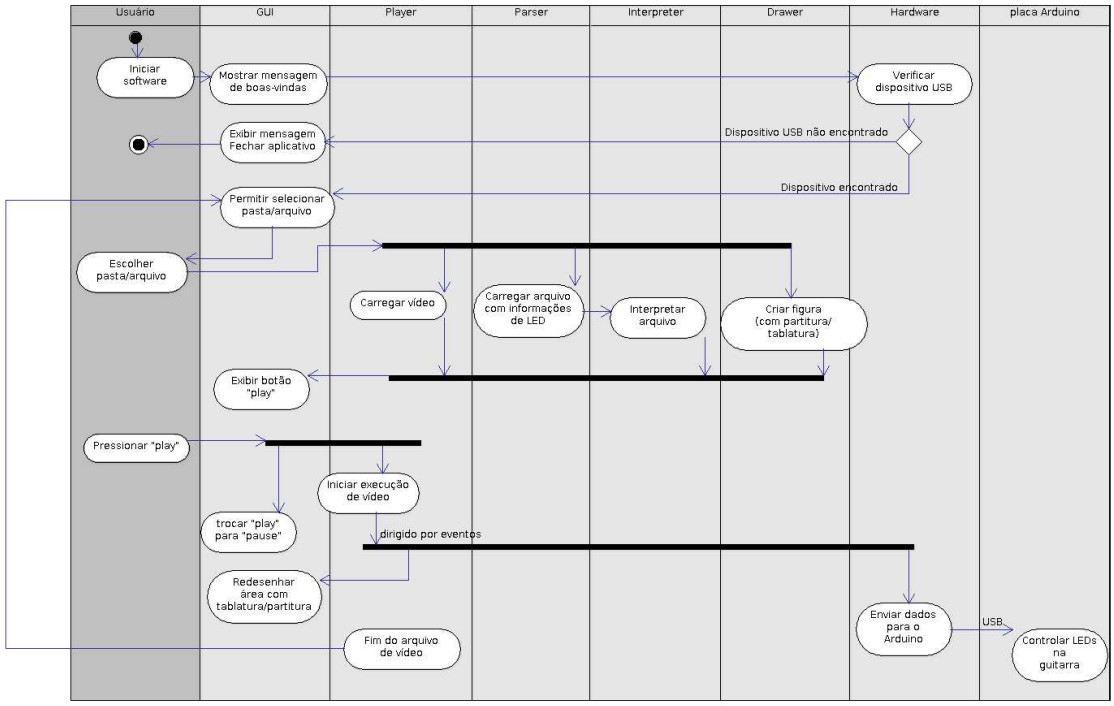

Figura 60 – Diagrama de Atividades

No diagrama, verifica-se que o usuário interage com a interface, que é responsável por executar todos os demais métodos. O uso de eventos permite que a comunicação com o *hardware* (no caso, a placa Arduino) e a atualização da área com a tablatura sejam desacoplados, permitindo que novas funcionalidades sejam incluídas sem necessidade de grande alteração no código dos demais componentes.

A linguagem adotada na implementação foi Python, uma linguagem de alto nível, fortemente e dinamicamente tipada. Multiplataforma, está disponível para Microsoft Windows, GNU/Linux e MacOS (PSF, 2013a), além de existirem versões adaptadas para o Android (FERRILL, 2013). A linguagem pode ser expandida usando C ou C++ e inclui diversos módulos *built-in*. Além disso, um vasto repositório dá acesso à diversas implementações de bibliotecas e código aberto (PSF, 2013b).

A interface foi desenhada em uma ferramenta RAD (*Rapid Application Development*) chamda Glade (GNOME.ORG, 2013a) e pode ser vista na Figura 61. Essa ferramenta permite criar, utilizando a biblioteca GTK, versão 3, disponível para Microsoft Windows, GNU/Linux e Apple Max OS X (GNOME.ORG, 2013b). A ferramenta gera um XML que é carregado em tempo de execução pela aplicação. O Código-Fonte 7 mostra como o XML da interface é carregado e mostrado utilizando a linguagem Python.

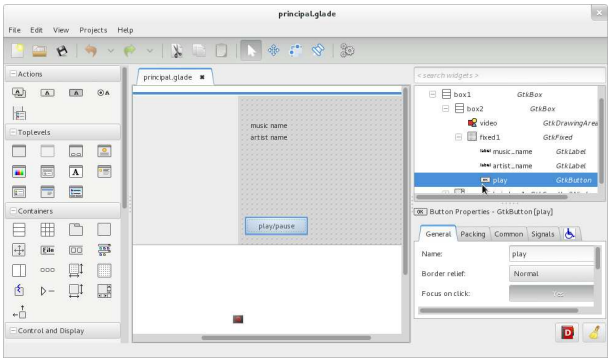

Figura 61 – Ferramenta RAD utilizada: Glade

```
1 from gi.repository import Gtk
2
3 filename = "principal.glade"
4
5 builder = Gtk.Builder()
6 builder.add_from_file(filename)
7 window = builder.get_object("window_top")
8 window.show_all()
```

Código-Fonte 7: Python montando uma interface com Glade

Para exibição do vídeo, foi utilizada uma implementação também da bibliteca GTK chamada GSTreamer (também chamdamada GST). GST é uma biblioteca (*library*) para construção de gráficos e manipulação de arquvio de mídia, como áudio e vídeo (FREEDESKTOP, 2013). Existem diversas incompatibilidades entre a versão atual do GSTreamer (1.0), a atual da GTK (3.6) e o Python. Essas dificuldades foram enfrentadas pelos desenvolvedores do Novacut, um *software* de edição de vídeos para o Ubuntu Linux (DEROSE, 2012). Essas dificuldades foram sanadas atraveś de soluções alternativas, que incluem, por exemplo, importar módulos que não serão utilizados pelo *software*, mas que precisam ser inicializados com a aplicação.

A comunicação serial via porta USB utiliza a biblioteca PySerial, disponível para Microsoft Windows, GNU/Linux e a variantes BSD (LIECHTI, 2010). Apesar da biblioteca ser multiplataforma, os parâmetros necessários para a comunicação mudam. No Microsoft Windows, por exemplo, as portas recebem nomes (COM1, COM2, etc.). No GNU/Linux, são arquivos (/dev/ttyUSB0, /dev/ttyUSB1, etc.). Essa abstração deve ser tratada pelo *software* que usa a biblioteca. Além da comunicação, a biblioteca também permite listar portas disponíveis. Em Código-Fonte 8, o código Python envia 1 byte para a primeira porta serial encontrada no sistema. O códio envia o caracter "A", então o valor enviado é 65 (decimal).

```
1 from serial import Serial
2 from serial.tools import list_ports
3
4 ports = [i[0] for i in list_ports.grep("")]
5 assert ports, u"Nenhuma porta serial encontrada"
6 serial = Serial(ports[0])
7 serial.write("A")
```

Código-Fonte 8: Python enviando dados via porta serial

A tablatura é desenhada utilizando uma biblioteca chamada *Python Imaging Library* (também conhecida como PIL). É uma biblioteca multiplataforma, suportando as mesmas plataformas do Python. Com ela, é gerada uma imagem da tablatura, que é sincronizada com o vídeo em tempo de execução, rolando a imagem à medida que o vídeo é exibido.

O arquivo com os dados a serem enviados para o Arduino é interpretado utilizando expressões regulares. Cada tipo de informação expressa possui uma notação diferente, identificada por uma expressão regular própria (tempo, informação dos bytes e/ou comentários, por exemplo). O tempo é indicado com precisão de décimos de segundo, onde indica o início e o fim de cada exibição dos LEDs, exatamente como uma legenda em um vídeo. O parser implementado permite utilizar, para a informação do LEDs, notação decimal (63), hexadecimal (0x3F) e binário (0b111111). Cada bit do valor informado é o controle de um LED no corpo e braço da guitarra. São 13 bytes ao todo, com o primeiro indicando o corpo e os demais cada uma das casas do braço da guitarra.

A comunicação entre todos esses componentes e a sua sincronia é mantida através de eventos e observadores, com o *design pattern observer*. O uso desses eventos permite a expansão futura do *software* sem necessidade de modificação na implementação já existente.

5 Conclusões

No presente trabalho, foram abordados diversos tópicos relacionados à música, iniciando pelo instrumento e padronizações adotadas para o aprendizado de música. Passando para a representação musical, foram descritas diferentes formas de representar acordes e músicas - tanto as utilizadas de forma comum quanto computacional. Esses estudos mostraram-se necessários para o conhecimento e elaboração da proposta de solução apresentada, selecionando os que foram utilizados na produção do protótipo, os não-adequados e quais poderiam resultar em trabalhos futuros.

Seguindo, foram testadas 2 alternativas para construção de aplicações utilizando computação embarcada. Optou-se pelo Arduino, pois o mesmo é simplificado para a construção de protótipos. Dessa forma, foi necessário também estudar formas de ampliar a quantidade de portas de entrada e saídas disponíveis, já que a proposta de solução inclui mais do que as 20 portas disponíveis em um único Arduino. Essa solução, que permite expandir 2 portas para 64, também reduziu a quantidade de cabos necessários percorrendo o braço inteiro da guitarra no protótipo. Os estudos realizados da Launchpad representaram testes interessantes, mesmo que a solução final adotada não utilize-se desse hardware.

No capítulo de apresentação do protótipo *Lighted Strings*, mostrou-se como o protótipo foi planejado e construído. A partir dessa visão ampla, foi explicada a adaptação da guitarra, com a construção de pequenos circuitos embutidos a serem ligados no Arduino. A descrição do protocolo de comunicação que se seguiu inclui alterações da proposta inicial (explicada no TCC 1), aumentando a quantidade de *bytes*, mas tornando-o mais simples de entender. Descreveu-se também o software embarcado presente no Arduino, utilizando a linguagem C.

A descrição do *sofware*, em linguagem Python, inclui o uso de *design patterns* conhecidos, com destaque para o *observer*, que auxiliou na organização e independência de responsabilidade das classes utilizadas. Esses padrões não são exclusivos da linguagem utilizada, sendo empregados largamente na indústria de software. Também para a construção da aplicação, foi necessário estudar implementações existentes das bibliotecas utilizadas, pois as mesmas estão em processo de transição, com alguns *bugs* e soluções não intuitivas.

5.1 Contribuições

Comparados com os trabalhos existentes estudados, o presente protótipo se diferencia pelo uso da tecnologia no ensino dos acordes da guitarra. Protótipos de ensino usando LEDs são utilizados para ensino de teclado. Protótipos para guitarras se destinam a

demonstrar apenas músicas ou apenas acordes, resumindo-se à apresentação *online* no computador ou apenas no instrumento. A solução apresentada busca agregar todas essas tecnologias para oferecer um novo apoio para o aprendizado dos acordes e desenvolvimento de habilidades para tocar guitarra.

Os materiais utilizados para construção do protótipo são facilmente encontrados em lojas físicas ou na *Internet*. O presente trabalho demonstrou alguns problemas encontrados na produção do protótipo e as soluções adotadas. Dessa forma, busca permitir aos interessados a reconstrução do projeto, com possíveis melhorias identificadas em trabalhos futuros.

5.2 Limitações

O *software* e *hardware* implementados permitem apenas o uso de uma cor no LED. Essa limitação foi imposta pela dificuldade de encontrar LEDs RGB no tamanho necessário e pela quantidade de cabos extras necessários para controlar cada um dos LEDs.

A importação de arquivos MIDI e MusicXML não foi implementada nessa versão do *software*. Assim, o acervo de músicas disponíveis nesses formatos não pode ser utilizada com a solução proposta atualmente. Isso pode ser solucionado com a evolução do *software* desenvolvido.

O uso da guitarra na solução proposta, devido aos inúmeros cabos externos para controle dos LEDs, fica prejudicado. Em uma versão final como produto, uma solução menor deverá ser implementada.

5.3 Dificuldades

A computação embarcada não faz parte do currículo regular do curso em Bacharelado em Ciência da Computação da Universidade de Caxias do Sul. Dessa forma, foi necessário dispender esforço explorando conceitos e aplicações dessa área do conhecimento. No entanto, disciplinas como Organização e Arquitetura de Computadores e Introdução à Informática (quando aplicada especificamente à computação) ofereceram subsídios para o estudo dessa área da computação.

A necessidade de construir pequenos circuitos apresentou mais dificuldades no andamento do trabalho. Devido à inesperiência com soldas de pequenos componentes, a produção teve ritmo lento e diversos problemas foram encontrados devido à contatos mal-feitos ou soldas grandes demais para o tamanho do *chip* utilizado. Esses problemas muitas vezes passaram despercebidos, causando desde mal funcionamento da solução até queima de *chips*.

Para criar aplicações de computação aplicada à música, foi necessário estudar representações musicais e computacionais. Muitas fontes, porém, tem origem em conhecimento popular, já que a música faz parte da história humana. A existência de diversas técnicas e habilidades

necessárias para cada uma delas foi causa de dificuldade para restringir o escopo da solução apresentada.

Alterar um instrumento musical, no caso da guitarra trabalhando com madeira, representou outro desafio. Apesar de não diretamente ligada à computação, foi necessário desenvolver habilidades e soluções de marcenaria para construir o protótipo.

5.4 Trabalhos Futuros

A implementação deste protótipo se utilizou de *software* e *hardware* livres. Com as definições propostas e bem separadas, torna-se possível substituir parte da implementação sem comprometer o restante da solução. O objetivo foi de tornar o trabalho aberto para futuras implementações.

Na campo do *software*, as implementações mais claras de próximos trabalhos são a importação de arquivos MIDI e MusicXML. A biblioteca Music21, já estudada neste trabalho, forneceria subsídios para essa implementação em Python, mesma linguagem utilizada no programa principal.

Seguindo no *software*, outra implementação futura é a inclusão de partituras (também descrito neste trabalho) juntamente com a tablatura na interface gráfica. A biblioteca Music21 também poderia auxiliar na construção dessa melhoria na interface.

Outros softwares poderiam utilizar a guitarra com os LEDs em outras aplicações, como jogos ou desafios. O uso de redes sociais também poderia ser implementado, aumentado desafios e fomentando o aprendizado do instrumento. Outra implementação poderia utilizar reconhecimento de padrões para identificar se o aluno (ou jogador, no caso de um jogo) está tocando corretamente o que foi proposto.

No *hardware*, a evolução seria complementar todas as casas da guitarra com LEDs, permitindo notas com oitava abaixo, permitindo outros tópicos de estudos e músicas mais elaboradas. A não inclusão desses LEDs no protótipo atual deve-se à possibilidade de testar a solução proposta mesmo com apenas alguns LEDs.

O *hardware* pode ser ampliado também incluindo LEDs coloridos (RGB). Dessa forma, poderia-se indicar quais dedos deveriam tocar determinada corda, fazendo um mapeamento das cores nos dedos. Isso abriria outras possibilidades para estudo do instrumento.

O material desenvolvido neste trabalho será disponibilizado sob licença *open source*, acessível a desenvolvedores que se interessarem pelo projeto. Assim, projetos derivados poderão ser desenvolvidos, incluindo melhorias na solução proposta. O objetivo é disseminar o conhecimento descrito neste projeto, permitindo a colaboração nesta e em outras soluções propostas utilizando computação embarcada e/ou aplicada ao ensino de música.

Referências

ARDUINO. *Arduino Build Process*. 2012. Disponível em: <http://www.arduino.cc/en/Hacking/BuildProcess>. 40

ARDUINO. *Arduino Mega*. 2012. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardMega>. 40

ARDUINO. *Arduino, Official Website*. 2012. Disponível em: <http://arduino.cc>. 39

ARDUINO. *Arduino Software*. 2012. Disponível em: <http://arduino.cc/en/Main/Software>. 40

ARDUINO. *Lilypad Arduino Board*. 2012. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardLilyPad>. 39

ARDUINO. *Serial*. 2012. Disponível em: <http://arduino.cc/en/Reference/Serial>. 39

ARDUINO. *Shields*. 2012. Disponível em: <http://www.arduino.cc/en/Main/arduinoShields>. 40

BAY, W. *First Lessons: Beginning Guitar*. Mel Bay Publications, 2005. Disponível em: <http://books.google.com.br/books?id=G_nJT9rzg0UC>. 21, 23

BECK, I. *gTar: The First Guitar That Anybody Can Play*. 2012. Disponível em: <http://www.kickstarter.com/projects/incident/gtar-the-first-guitar-that-anybody-can-play>. 16

BLINKM. *BlinkMSoftI2CDemo*. 2012. Disponível em: <https://code.google.com/p/blinkm-projects/source/browse/trunk/blinkm_examples/arduino/BlinkMSoftI2CDemo/>. 47

CABRAL, G. et al. Cifra para o braço: Estudo dos problemas de execução musical em violão e guitarra. In: *Proceedings of VIII Brazilian Symposium on Computer Music. Fortaleza*. [S.l.: s.n.], 2001. 24

CAESAR, W. *Guitarra-Noções Elementares*. [S.l.]: Irmãos Vitale, 2003. 22, 28

CARDOSO, B.; MASCARANHAS, M. *CURSO COMPLETO DE TEORIA MUSICAL E SOLFEJO - 1ô VOL.* Irmãos Vitale. ISBN 9788585188177. Disponível em: <http://books.google.com.br/books?id=_zQCnNWc3vMC>. 26

CASIO. *Lighted Keys Digital Pianos*. 2012. Disponível em: <http://www.casiomusicgear.com/products/menu_lighted keys>. 18

CASTILHO, J. *Toque Junto-Guitarra*. [S.l.]: Irmãos Vitale, 2000. 22

CHESNEY, T. "other people benefit. i benefit from their work." sharing guitar tabs online. *Journal of Computer-Mediated Communication*, 2004. Wiley Online Library, v. 10, n. 1, p. 00–00, 2004. 28

CREATIVECOMMONS. *Attribution-ShareAlike 3.0 United States (CC BY-SA 3.0)*. 2012. Disponível em: <https://creativecommons.org/licenses/by-sa/3.0/us/>. 35

CUTHBERT, M. S. *Music21, Google Code*. 2012. Disponível em: <http://code.google.com/p/music21/>. 35

CUTHBERT, M. S. *Music21, Website Oficial*. 2012. Disponível em: <http://music21.org/>. 34

DEROSE, J. G. *Port your Python app to PyGI, Gtk3, and GStreamer 1.0*. 2012. Disponível em: <https://wiki.ubuntu.com/Novacut/GStreamer1.0>. 62

DOITKITS. *Freeduino BR*. 2012. Disponível em: <http://freeduino.blogspot.com.br/>. 39

DOURADO, H. *Dicionário de termos e expressões da música*. [S.l.]: Editora 34 Ltda, 2004. 24, 25

FAT16LIB. *New i2c libraries with softi2c*. 2008. Disponível em: <http://forums.adafruit.com/viewtopic.php?f=25&t=13722>. 47

FERRILL, P. *Android applications using Python and SL4A*. 2013. Disponível em: <http://www.ibm.com/developerworks/library/mo-python-sl4a-1/>. 62

FREEDESKTOP. *GStreamer, open source multimedia frameword*. 2013. Disponível em: <http://gstreamer.freedesktop.org/>. 62

FRITZING. *Fritzing, Official Website*. 2012. Disponível em: <http://fritzing.org/>. 42

FRY, B.; REAS, C. *Processing.org*. 2013. Disponível em: <http://processing.org/>. 40

FSF, F. S. F. *GNU General Public License, version 2*. 1992. Disponível em: <https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>. 47

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. ISBN 9780321700698. Disponível em: <http://books.google.com.br/books?id=6oHuKQe3TjQC>. 61

GILBERT, D. *Guitarra Solo*. [S.l.]: Irmãos Vitale, 2007. 23, 26

GNOME.ORG. *Glade*. 2013. Disponível em: <http://glade.gnome.org/>. 62

GNOME.ORG. *GTK*. 2013. Disponível em: <http://www.gtk.org/>. 62

GOHN, D. *Auto-aprendizagem musical: alternativas tecnológicas*. [S.l.]: Annablume, 2003. 15

GORDON, E. *Learning sequences in music: A contemporary music learning theory*. [S.l.]: GIA Publications, 2007. 25, 26

HENTSCHKE, L. Motivação para aprender música em espaços escolares e não-escolares. 2009. ETD - Educação Temática Digita - Unicamp, 2009. Disponível em: <http://www.fae.unicamp.br/revista/index.php/etd/article/view/2056/1861>. 15

HESELWOOD, B. Features of tablature notation in the current international phonetic alphabet chart. 2008. 2008. Disponível em: <http://www.bretton.ac.uk/linguistics/WPL/WP2008/5.pdf>. 28

HEWITT, M. *Music Theory for Computer Musicians*. Course Technology Ptr, 2008. ISBN 9781598635034. Disponível em: <http://books.google.com.br/books?id=YGXwJwAACAAJ>. 24

HOGARTH, T. *Introduction to Guitar*. 2013. Disponível em: <https://www.coursera.org/course/guitar>. 18

HUMMES, J. M. Por que é importante o ensino de música? considerações sobre as funções da música na sociedade e na escola. *Revista da Associação Brasileira de Educação Musical*, 2004. v. 11, p. 17–25, 2004. 15

INSTRUCTABLES. *Instructables Arduino*. 2012. Disponível em: <http://www.instructables.com/technology/arduino/>. 39

INSTRUMENTS, T. *MSP430 LaunchPad Value Line Development kit*. 2012. Disponível em: <https://estore.ti.com/MSP-EXP430G2-MSP430-LaunchPad-Value-Line-Development-kit-P2031.aspx>. 44

INSTRUMENTS, T. *TI Launchpad, Official Website*. 2012. Disponível em: <http://www.ti.com/launchpad>. 44

KDE. *KGuitar, Website Oficial*. 2010. Disponível em: <http://kguitar.sourceforge.net/>. 34

LAHART, J. *Taking an Open-Source Approach to Hardware*. The Wall Street Journal, 2009. Disponível em: <http://online.wsj.com/article/SB10001424052748703499404574559960271468066.html>. 39

LIAROKAPIS, F. Augmented reality scenarios for guitar learning. In: CITESEER. *Third International Conference on Eurographics UK Theory and Practice of Computer Graphics, Canterbury, UK*. [S.l.], 2005. p. 163–170. 16

LIECHTI, C. *Python Serial Port Extension*. 2010. Disponível em: <http://pyserial.sf.net>. 63

LIZ, C. de. *Os nomes dados às partes do instrumento*. 2011. Disponível em: <http://cristiandeliz.blogspot.com.br/2011/06/os-nomes-dados-as-partes-do-instrumento.html>. 21

MAKEMUSIC. *MusicXML, Official Website*. 2012. Disponível em: <http://www.makemusic.com/musicxml>. 33

MANUS, M. *Guitar Chord Dictionary: Handy Guide*. Alfred, 1978. (Alfred handy guide). ISBN 9780739014844. Disponível em: <http://books.google.com.br/books?id=xSuNXGXLthIC>. 24

MATROSKA. *SRT Subtitles*. 2011. Disponível em: <http://www.matroska.org/technical/specs/subtitles/srt.html>. 60

MUSE. *Muse, Website Oficial*. 2012. Disponível em: <http://www.muse-sequencer.org/>. 33

NICHOLL, M. *Introduction to MIDI/Synthesis*. Alfred Publishing Co., Inc., 1993. Disponível em: <http://books.google.com.br/books?id=oQBMAAAAYAAJ>. 32

NOTEEDIT. *Noteedit, Website Oficial*. 2012. Disponível em: <noteedit.berlios.de>. 33

OGASAWARA, A. et al. Reconhecedor de notas musicais em sons polifônicos. *Rio de Janeiro, Rio de Janeiro*, 2008. 2008. Disponível em: <https://www.lps.ufrj.br/profs/sergioln/theses/bsc18angelica.pdf>. 24

OLSON, D. *Finishing the UART Transceiver*. 2012. Disponível em: <http://mspsci.blogspot.com.br/2012/05/tutorial-18-finishing-uart-transceiver.html>. 45

OVERLY, M. *Guitar fretboard facts: how to see the guitar with clarity and avoid confusion*. [S.l.]: Twelve Tone Music Pub, 2004. 30, 31

PHILIPS. *PCF8574 - Product specification*. 2002. Disponível em: <http://datasheet.octopart.com/PCF8574T/3,518-NXP-datasheet-31958.pdf>. 45

PSF, P. S. F. *About Python*. 2013. Disponível em: <http://www.python.org/about/>. 62

PSF, P. S. F. *PyPI - the Python Package Index*. 2013. Disponível em: <https://pypi.python.org/pypi>. 62

QTRACTOR. *QTractor, Website Oficial*. 2012. Disponível em: <http://qtractor.sourceforge.net/>. 33

RATTON, M. B. *MIDI: guia básico de referência*. [S.l.]: H. Sheldon, 1991. ISBN 8570017391. 32, 33

RUI, L. R.; STEFFANI, M. H. Física: som e audição humana. 2007. Simpósio Nacional de Ensino de Física, 2007. Disponível em: <http://www.lume.ufrgs.br/handle/10183/30450>. 24

SACRAMENTO, J. *Notação Musical*. josé sacramento, 2007. ISBN 9789899528802. Disponível em: <http://books.google.com.br/books?id=N5W0O5GmV68C>. 29

SANCHEZ, N. S. *Curso de Violão*. [S.l.]: Irmãos Vitale, 2003. 24, 25

SEQ24. *seq24, Website Oficial*. 2012. Disponível em: <http://filter24.org/seq24/about.html>. 33

SILVA, A. *Manual de Guitarra - Nível 1 - Grau Básico*. [S.l.], 2008. Disponível em: <http://artursilva.com.sapo.pt/manual_guitarra_1.pdf>. 16, 29, 30

SILVA, A. *Manual de Guitarra - Nível 2 - Grau Básico*. [S.l.], 2008. Disponível em: <http://artursilva.com.sapo.pt/manual_guitarra_2.pdf>. 16

SKILLBOX. *Online Midi eLearning*. 2012. Disponível em: <http://www.training-classes.com/learn/_k/m/i/d/midi/_t/online/>. 32

STATUSBITS. *Using printf with MSP430 LaunchPad*. 2012. Disponível em: <https://statusbits.wordpress.com/2012/08/27/using-printf-with-msp430-laundpad-part-1/>. 45

THINGM. *thingm - an electronic product studio*. 2010. Disponível em: <http://thingm.com/>. 47

USF-IF, U. I. F. I. Specification-rev 1.0, universal serial bus. *Sec.* v. 9, n. 2, p. 184–185. 51

VIEIRA, G.; RAY, S. Ensino coletivo de violão: Técnicas de arranjo para o desenvolvimento pedagógico. In: *XVI Encontro Anual da ABEM e Congresso Regional da ISME na América Latina*. [S.l.: s.n.], 2007. 15

YAMAHA. *Lighted Key/Lighted Fret Instruments*. 2012. Disponível em: <http://usa.yamaha.com/products/musical-instruments/keyboards/digitalkeyboards/lighted_key_lighted_fret_instruments/?mode=series>. 18

6 Anexos

6.1 Lauchpad - Comunicação Serial

```
/************************************************************************
 *                 Half Duplex Software UART on the LaunchPad
 *
 * Description: This code provides a simple Bi-Directional Half Duplex
 *         Software UART. The timing is dependant on SMCLK, which
 *         is set to 1MHz. The transmit function is based off of
 *         the example code provided by TI with the LaunchPad.
 *         This code was originally created for "NJC's MSP430
 *         LaunchPad Blog".
 *
 * Author: Nicholas J. Conn - http://msp430launchpad.com
 * Email: webmaster at msp430launchpad.com
 * Date: 08-17-10
 ************************************************************************/

#include    "msp430g2231.h"
#include    "stdbool.h"

#define         TXD             BIT1    // TXD on P1.1
#define         RXD         BIT2    // RXD on P1.2

// 9600 Baud, SMCLK=1MHz (1MHz/9600)=104
#define          Bit_time        104

// Time for half a bit.
#define         Bit_time_5    52

// Bit count, used when transmitting byte
unsigned char BitCnt;
// Value sent over UART when Transmit() is called
unsigned int TXByte;
// Value recieved once hasRecieved is set
unsigned int RXByte;
```

```
// Status for when the device is receiving
bool isReceiving;
// Lets the program know when a byte is received
bool hasReceived;

// Function Definitions
void Transmit(void);

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;        // Stop WDT

    BCSCTL1 = CALBC1_1MHZ;            // Set range
    DCOCTL = CALDCO_1MHZ;            // SMCLK = DCO = 1MHz

    P1SEL |= TXD;
    P1DIR |= TXD;

    P1IES |= RXD;                // RXD Hi/lo edge interrupt
    P1IFG &= ~RXD;                // Clear RXD (flag) before enabling interrupt
    P1IE |= RXD;                // Enable RXD interrupt


    P1DIR |= 0x01; //LED


    isReceiving = false;            // Set initial values
    hasReceived = false;

    __bis_SR_register(GIE);            // interrupts enabled\

    while(1)
    {
        if (hasReceived)        // If the device has recieved a value
        {
            hasReceived = false;    // Clear the flag
            if (RXByte == 'a') {
                P1OUT = 0x01;
                TXByte = '1';
            } else {
                P1OUT = 0x00;
                TXByte = '0';
```

```
            }
            Transmit();
        }
        if (~hasReceived)        // Loop again if another value has been received
                __bis_SR_register(CPUOFF + GIE);
            // LPM0, the ADC interrupt will wake the processor up. This is so that it
            //    endlessly loop when no value has been Received.
    }
}

// Function Transmits Character from TXByte
void Transmit()
{
    while(isReceiving);            // Wait for RX completion
      CCTL0 = OUT;                // TXD Idle as Mark
      TACTL = TASSEL_2 + MC_2;        // SMCLK, continuous mode

      BitCnt = 0xA;                // Load Bit counter, 8 bits + ST/SP
      CCR0 = TAR;                // Initialize compare register

      CCR0 += Bit_time;            // Set time till first bit
      TXByte |= 0x100;            // Add stop bit to TXByte (which is logical 1)
      TXByte = TXByte << 1;            // Add start bit (which is logical 0)

      CCTL0 =  CCIS0 + OUTMOD0 + CCIE;    // Set signal, intial value, enable interrup
      while ( CCTL0 & CCIE );            // Wait for previous TX completion
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    isReceiving = true;

    P1IE &= ~RXD;            // Disable RXD interrupt
    P1IFG &= ~RXD;            // Clear RXD IFG (interrupt flag)

      TACTL = TASSEL_2 + MC_2;    // SMCLK, continuous mode
      CCR0 = TAR;            // Initialize compare register
      CCR0 += Bit_time_5;        // Set time till first bit
    CCTL0 = OUTMOD1 + CCIE;        // Dissable TX and enable interrupts
```

```
    RXByte = 0;            // Initialize RXByte
    BitCnt = 0x9;            // Load Bit counter, 8 bits + ST
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
    if(!isReceiving)
    {
        CCR0 += Bit_time;            // Add Offset to CCR0
        if ( BitCnt == 0)            // If all bits TXed
        {
              TACTL = TASSEL_2;        // SMCLK, timer off (for power consumption)
            CCTL0 &= ~ CCIE ;        // Disable interrupt
        }
        else
        {
            CCTL0 |=  OUTMOD2;        // Set TX bit to 0
            if (TXByte & 0x01)
                CCTL0 &= ~ OUTMOD2;    // If it should be 1, set it to 1
            TXByte = TXByte >> 1;
            BitCnt --;
        }
    }
    else
    {
        CCR0 += Bit_time;                // Add Offset to CCR0
        if ( BitCnt == 0)
        {
              TACTL = TASSEL_2;            // SMCLK, timer off (for power consumption)
            CCTL0 &= ~ CCIE ;            // Disable interrupt

            isReceiving = false;

            P1IFG &= ~RXD;                // clear RXD IFG (interrupt flag)
            P1IE |= RXD;                // enabled RXD interrupt

            if ( (RXByte & 0x201) == 0x200)        // Validate the start and stop bits are
            {
                RXByte = RXByte >> 1;        // Remove start bit
                RXByte &= 0xFF;            // Remove stop bit
```

```
                hasReceived = true;
            }
              __bic_SR_register_on_exit(CPUOFF);    // Enable CPU so the main while lo
        }
        else
        {
            if ( (P1IN & RXD) == RXD)        // If bit is set?
                RXByte |= 0x400;        // Set the value in the RXByte
            RXByte = RXByte >> 1;            // Shift the bits down
            BitCnt --;
        }
    }
}
```