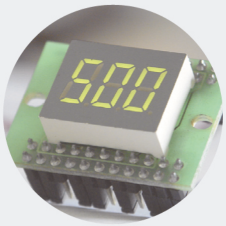
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES





Sequenciador MIDI

RUI MIGUEL TEIXEIRA PINTO
agosto de 2018

POLITÉCNICO
DO PORTO

SEQUENCIADOR MIDI

Rui Miguel Teixeira Pinto

Instituto Superior de Engenharia do Porto

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização de Automação e Sistemas

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2018

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Rui Miguel Teixeira Pinto, Nº 1121249, 1121249@isep.ipp.pt
Orientação científica: Lino Manuel Baptista Figueiredo, lbf@isep.ipp.pt

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
30 de julho de 2018

Agradecimentos

Aos meus pais, aos meus avós e ao meu irmão pelo apoio incondicional durante todo o meu percurso académico. Agradeço a confiança depositada ao longo da minha formação durante o mestrado.

À minha namorada Ana Correia, que esteve sempre presente, mesmo nos momentos de maior dificuldade, e me ajudou a ultrapassar mais uma etapa. Obrigado pela paciência, pela compreensão e pela dedicação.

Aos meus amigos e colegas da faculdade por todo o apoio e incentivo prestados, com um especial agradecimento aos meus amigos Diogo Pereira, Ivo Urbano, Luís Sousa e Nuno Pinto, por toda a ajuda na construção da estrutura para este projeto.

Ao meu orientador, Engenheiro Lino Figueiredo, pela dedicação, auxílio e orientação durante a realização de todo o projeto, bem como na elaboração do relatório.

Resumo

O presente trabalho consiste no desenvolvimento e implementação de um sequenciador físico. Este é composto por 12 linhas e 12 colunas, resultando num total de 144 posições. O utilizador interage com o sistema ao inserir bolas em cada uma das posições, com o objetivo de criar notas musicais de acordo com a transição de um compasso de tempo. Se não pretender a inserção de bolas também tem disponível outro modo de funcionamento, que consiste em premir cada posição.

A reprodução musical do sequenciador é realizada com recurso ao protocolo MIDI, que se encontra presente num *software* instalado no computador. A cada coluna do sequenciador é associado um instrumento e uma nota musical, para que quando o compasso de tempo atinja uma posição ativa, seja enviada a mensagem MIDI relativa à coluna onde esta se encontra.

O sequenciador além de comunicar com o *software* MIDI presente num computador, também o faz com uma aplicação desenvolvida para *smartphones* Android. Esta permite monitorizar todas as posições do sequenciador, assim como visualizar e configurar todos os seus outros parâmetros. Tanto o computador como o *smartphone* Android comunicam com o sequenciador por Bluetooth, o que significa que todo o sistema é interligado sem fios.

O sequenciador é constituído por um microcontrolador, sendo este o cérebro de todo o sistema, dado que determinará quais as posições que se encontram ativas, assim como controlará os LEDs. Este também garante o envio das mensagens MIDI para o *software* do computador, e o conjunto de dados para a aplicação Android, de maneira a que sejam visualizados e configurados todos os parâmetros do sistema.

O sistema desenvolvido funciona tal como esperado. O sequenciador envia mensagens MIDI para o computador, o que permite a sua reprodução musical. A comunicação com a aplicação Android é efetuada corretamente, dado que possibilita a monitorização de todo os parâmetros presentes no sistema, como também a sua configuração.

Palavras-Chave

Sequenciador, MIDI, Android, Bluetooth.

Abstract

The present work consists of the development and implementation of a physical sequencer. This consists of 12 rows and 12 columns, resulting in a total of 144 positions. The user interacts with the system by inserting balls in each of the positions, with the purpose of creating musical notes according to the transition of a time compass. If you don't want to insert balls, you also have another operating mode available, which consists of pressing each position.

Musical playback of the sequencer is performed using the MIDI protocol, which is present in a software installed on the computer. To each column of the sequencer an instrument and musical note is associated so that when the time measure reaches an active position, the MIDI message is sent relative to the column where it is located.

In addition to communicate with the MIDI software present on a computer, the sequencer also does it with an application developed for Android smartphones. This allows you to monitor all the positions of the sequencer, as well as view and configure all your other parameters. Both the Android computer and smartphone communicate with the Bluetooth sequencer, which means that the entire system is interconnected wirelessly.

The sequencer consists of a microcontroller, which is the brain of the entire system, since it will determine which positions are active, as well as control the LEDs. It also ensures that MIDI messages are sent to the computer software and data set for the Android application so that all system parameters are displayed and configured.

The developed system works as expected. The sequencer sends MIDI messages to the computer, which allows for its musical reproduction. Communication with the Android application is performed correctly, since it allows the monitoring of all the parameters present in the system, as well as its configuration.

Keywords

Sequencer, MIDI, Android, Bluetooth.

Índice

AGRADECIMENTOS .... **I**

RESUMO .... **III**

ABSTRACT .... **V**

ÍNDICE .... **VII**

ÍNDICE DE FIGURAS .... **XI**

ÍNDICE DE TABELAS .... **XV**

ACRÓNIMOS .... **XVII**

1. INTRODUÇÃO **1**

1.1. MOTIVAÇÃO 1
1.2. OBJETIVOS 1
1.3. CALENDARIZAÇÃO 2
1.4. ORGANIZAÇÃO DO RELATÓRIO 3

2. CONTEXTUALIZAÇÃO **5**

2.1. MÚSICA POR COMPUTADOR 7
2.2. SINTETIZADORES 8
2.2.1. Origem *9*
2.2.2. Analógico vs Digital *10*
2.3. ALPHASPHERE 10
2.4. EIGENHARP 11
2.5. REACTABLE 12
2.6. HARPA LASER 14
2.6.1. Harpa Framed *15*
2.6.2. Harpa Frameless *16*
2.7. GRIDI 17
2.7.1. Funcionamento *18*

3. TECNOLOGIAS ENVOLVIDAS **21**

3.1. MIDI 21
3.1.1. Origem *21*
3.1.2. Especificação *23*
3.1.3. Mensagens *24*
3.1.4. Controladores *29*
3.1.5. Hardware *31*
3.1.6. Tipos de Software *32*

3.1.7. *General MIDI* ... 33
3.2. ANDROID ... 34
3.2.1. *Android SDK* ... 34
3.2.2. *Componentes de Aplicação* ... 35
3.2.3. *Android Manifest* ... 41

4. ARQUITETURA ... 43

4.1. FUNCIONAMENTO ... 44
4.2. *HARDWARE* ... 44
4.2.1. *Microcontrolador* ... 45
4.2.2. *LED* ... 46
4.2.3. *Interruptor* ... 51
4.2.4. *Shift Register* ... 52
4.2.5. *Módulo Bluetooth* ... 55
4.3. *SOFTWARE* ... 59
4.3.1. *Desenvolvimento* ... 59
4.3.2. *Integração do MIDI* ... 60
4.3.3. *Android* ... 62
4.3.4. *EasyEDA* ... 64

5. IMPLEMENTAÇÃO ... 67

5.1. ESTRUTURA ... 67
5.2. ESQUEMA ELÉTRICO DO SISTEMA ... 70
5.2.1. *Controlo* ... 71
5.2.2. *Leitura* ... 75
5.3. DESENVOLVIMENTO DE PCB ... 77
5.4. ARQUITETURA DO *SOFTWARE* ... 81
5.4.1. *Configurações do Microcontrolador* ... 83
5.4.2. *Inicialização de variáveis* ... 87
5.4.3. *Rotina de Interrupção do Timer/Counter0* ... 90
5.4.4. *Toggle do LED de Funcionamento* ... 92
5.4.5. *Tempo* ... 93
5.4.6. *Leitura dos Botões* ... 96
5.4.7. *Controlo dos LEDs* ... 100
5.4.8. *Controlo dos LEDs do Compasso de Tempo* ... 107
5.4.9. *Envio de Tramas Android* ... 109
5.4.10. *Envio de Mensagens MIDI* ... 114
5.4.11. *Rotina de Interrupção da Receção de Dados na USART0* ... 116
5.4.12. *Execução de Tramas* ... 121

6. APLICAÇÃO ANDROID ... 125

6.1. FICHEIRO *MANIFEST* ... 125
6.2. ATIVIDADES ... 126
6.2.1. *Atividade Inicial* ... 127

6.2.2. Atividade Principal *128*
6.2.3. Atividade Bluetooth *145*
6.2.4. Atividade de Alteração da Dimensão *147*
6.2.5. Atividade de Alteração das Configurações de cada Coluna *148*
6.2.6. Atividade de Alteração da Cor da Linha Ativa do Tempo *154*
6.2.7. Atividade de Importação de Ficheiros de Configuração *155*
6.2.8. Atividade de Inserção de Ícones *157*
6.2.9. Atividade de Visualização das Configurações de cada Coluna *165*
6.3. RESULTADOS 165

7. CONCLUSÕES 171

REFERÊNCIAS DOCUMENTAIS 173

ANEXO A. ESQUEMA ELÉTRICO DA PLACA DE CONTROLO 177

ANEXO B. ESQUEMA ELÉTRICO DE UMA PLACA DE LEITURA 179

ANEXO C. REGISTOS ASSOCIADOS AO *TIMER/COUNTER0* 181

ANEXO D. REGISTOS ASSOCIADOS À USARTN 183

ANEXO E. CÓDIGO RELATIVO À BIBLIOTECA *FUNCS_WS2812B* 185

ANEXO F. FICHEIRO CSV DO ÍCONE DA FIGURA 106 189

Índice de Figuras

Figura 1 Primeiro fonógrafo, inventado por Thomas Edison [2] 6
Figura 2 Minimoog Model D [6].. 9
Figura 3 Alphasphere elite (a); Alphasphere nexus (b); Alphasphere me (c) [7]..................... 11
Figura 4 Modelos Pico da Eigenharp [8] 12
Figura 5 Reactable Experience [10] 12
Figura 6 Aplicação do protocolo TUIO na Reactable [11] 13
Figura 7 Harpa Laser utilizada por Jean Michel Jarre num dos seus concertos [12] 14
Figura 8 Harpa *Framed* [13] 15
Figura 9 Harpa *Frameless* [13] 16
Figura 10 *Spot Paintings*, de Damien Hirst [14] 17
Figura 11 GRIDI [14] 17
Figura 12 Exemplo de configuração dos instrumentos no sistema GRIDI [14] 18
Figura 13 Prophet 600 da Sequential Circuits [17] 22
Figura 14 Roland JP-6 [18] 22
Figura 15 Formato das mensagens MIDI [19] 23
Figura 16 Conetor DIN de cinco pinos macho 31
Figura 17 Rede MIDI [20] 32
Figura 18 Ciclo de vida de uma atividade [22] 36
Figura 19 Ciclo de vida de um serviço [23] 38
Figura 20 Exemplo de utilização de uma *intent* implícita [24] 40
Figura 21 Arquitetura geral do sistema 44
Figura 22 Circuito de proteção do WS2812B (a) e do WS2812 (b) [26] 47
Figura 23 Propriedades mecânicas do WS2812B (a) e do WS2812 (b) [27] 47
Figura 24 Ligação de três pixéis [28] 48
Figura 25 Circuito elétrico utilizado na conexão de três WS2812B [28] 48
Figura 26 Composição dos dados de 24 bits [28] 49
Figura 27 Gráfico de sequência dos valores lógicos e do *reset* [28] 49
Figura 28 Esquema elétrico de ligação de três módulos 50
Figura 29 Interruptor Momentâneo (a), Interruptor Fixo (b) [29] 51
Figura 30 Botões utilizados no sistema 51
Figura 31 Configuração dos pinos do CD4021BE [30] 53
Figura 32 Ligação de dois CD4021BE ao microcontrolador 55
Figura 33 Rede *Piconet* [31] 56
Figura 34 Módulos Bluetooth HC-05 (a) e HC-06 (b) 56

Figura 35 Ambiente gráfico do *Hairless MIDI to Serial Bridge* [32] 60
Figura 36 Ambiente gráfico do *loopMIDI* [33] 61
Figura 37 *SimpleSynth* 62
Figura 38 Janela principal do Android Studio [34] 63
Figura 39 Interface de utilizador do EasyEDA (desenvolvimento de circuitos elétricos) [35] 66
Figura 40 Esboço da parte superior da estrutura 68
Figura 41 Esboço da parte inferior da estrutura 69
Figura 42 Contacto no interruptor sem bola (a) e com bola (b) 69
Figura 43 Diagrama de blocos relativo ao *hardware* 70
Figura 44 Esquema elétrico do circuito de controlo 71
Figura 45 Programador USBasp 72
Figura 46 Esquema elétrico de ligação do microcontrolador aos módulos Bluetooth 73
Figura 47 Divisor de tensão 73
Figura 48 Função de cada pino do LM7805 [36] 74
Figura 49 Esquema do circuito de alimentação 74
Figura 50 Ligação da Placa de Controlo à Placa de Leitura 1 75
Figura 51 Ligação entre duas placas de leitura 76
Figura 52 Esquema elétrico de um *shift register* 76
Figura 53 Circuito de ligação de um botão a um *shift register* 77
Figura 54 PCB relativa ao esquema elétrico do Anexo A 78
Figura 55 Placa de Controlo 79
Figura 56 Face superior da PCB relativa ao esquema elétrico do Anexo B 79
Figura 57 Face inferior da PCB relativa ao esquema elétrico do Anexo B 80
Figura 58 Placa de Leitura 81
Figura 59 Fluxograma do funcionamento geral do sistema 82
Figura 60 Fluxograma relativo à numeração dos LEDs WS2812B 89
Figura 61 Fluxograma relativo à rotina de interrupção 92
Figura 62 Fluxograma relativo à alteração do estado do LED de funcionamento 93
Figura 63 Representação do *Tempo* na horizontal 94
Figura 64 Representação do *Tempo* na diagonal 94
Figura 65 Fluxograma relativo ao *Tempo* 95
Figura 66 Fluxograma relativo à primeira parte do processo de leitura dos botões 96
Figura 67 Fluxograma relativo à função *ShiftIn* 97
Figura 68 Fluxograma relativo à segunda parte do processo de leitura dos botões 98
Figura 69 Fluxograma relativo à função de alteração das variáveis de linha e coluna 99
Figura 70 Fluxograma relativo ao processo de controlo dos LEDs 101
Figura 71 Fluxograma relativo à função *Definir_Pixel_RGB* 102
Figura 72 Gráfico utilizado na transferência de dados para os LEDs WS2812B 103
Figura 73 Fluxograma relativo à função *Enviar_Cor* 104

Figura 74 Fluxograma relativo ao *tempo* no tipo horizontal 108
Figura 75 Cabeçalho geral de uma trama enviada para a aplicação Android 110
Figura 76 Cabeçalho de cada trama enviada para a aplicação Android 111
Figura 77 Fluxograma relativo ao envio de tramas Android 113
Figura 78 Fluxograma relativo à função de envio de comandos MIDI 115
Figura 79 Cabeçalho de cada trama enviada pela a aplicação Android 117
Figura 80 Fluxograma relativo à rotina de interrupção da receção de dados na USART0 119
Figura 81 Fluxograma relativo ao processo de leitura de um *byte* na USART0 120
Figura 82 Diagrama de blocos da aplicação Android 126
Figura 83 Atividade inicial 127
Figura 84 Atividade principal 128
Figura 85 Barra de ações da atividade principal 129
Figura 86 Controlo do *tempo* na atividade principal 133
Figura 87 Tipo horizontal (a); diagonal (b) 133
Figura 88 Parar (a); continuar (b) 134
Figura 89 Bluetooth desligado (a); ligado sem conexão (b); ligado com conexão (c) 134
Figura 90 Caixa de diálogo de ativação do Bluetooth 136
Figura 91 Fluxograma relativo à receção de dados na atividade principal 141
Figura 92 Verificação de erros nas configurações de cada coluna 142
Figura 93 Alteração do modo e controlo da matriz na atividade principal 143
Figura 94 Modo com bolas (a); sem bolas (b) 144
Figura 95 Atividade Bluetooth 145
Figura 96 Atividade de alteração da dimensão do sequenciador 147
Figura 97 *AppBarLayout* da atividade de alteração das configurações de cada coluna 148
Figura 98 Caixa de diálogo relativa à exportação de configurações para um ficheiro 149
Figura 99 Separador Som da atividade de alteração das configurações de cada coluna 151
Figura 100 Separador Cor da atividade de alteração das configurações de cada coluna 153
Figura 101 Atividade de alteração da cor da linha ativa do compasso de tempo 154
Figura 102 Atividade de importação de ficheiros de configuração 155
Figura 103 Atividade de inserção de ícones 157
Figura 104 Ícones de cada tipo 158
Figura 105 Barra de ações da atividade de inserção de ícones 159
Figura 106 Caixa de diálogo relativa à criação de um ícone 159
Figura 107 Caixa de diálogo relativa à indicação da existência de um ícone 160
Figura 108 Caixa de diálogo relativa à inserção de um ícone 162
Figura 109 Rotação do ícone relativo ao dígito 1 163
Figura 110 Fluxograma relativo à criação da variável final do ícone 164
Figura 111 Atividade de visualização das configurações de cada coluna 165
Figura 112 Bola utilizada no sequenciador MIDI 166

Figura 113 Exterior da estrutura 166
Figura 114 Interior da estrutura 167
Figura 115 Sequenciador implementado 168
Figura 116 Reprodução musical do sequenciador.. 168

Índice de Tabelas

Tabela 1 Calendarização relativa ao segundo semestre de 2016/17 ... 3
Tabela 2 Calendarização relativa a todo o ano letivo de 2017/18 ... 3
Tabela 3 Modos de operação do MIDI ... 24
Tabela 4 Caraterísticas das mensagens de voz do canal [20] ... 25
Tabela 5 Exemplo de uma mensagem MIDI [20] ... 26
Tabela 6 Caraterísticas das mensagens de modo do canal [20] ... 27
Tabela 7 Caraterísticas das mensagens comuns e em tempo real do sistema [20] ... 28
Tabela 8 Métodos de retorno do ciclo de vida de uma atividade ... 37
Tabela 9 Caraterísticas do ATmega1284P ... 45
Tabela 10 Função de cada pino do WS2812B [28] ... 48
Tabela 11 Tempos para transferência de dados [28] ... 50
Tabela 12 Função de cada pino do CD4021BE ... 53
Tabela 13 Função de cada pino dos módulos HC-05 e HC-06 ... 57
Tabela 14 Configuração inicial dos módulos HC-05 e HC-06 ... 57
Tabela 15 Comandos AT para o módulo HC-05 ... 58
Tabela 16 Comandos AT para o módulo HC-06 ... 59
Tabela 17 Parâmetros do comando AT+BAUD e da sua resposta ... 59
Tabela 18 Descrição de cada área da janela principal ... 64
Tabela 19 Caraterísticas da estrutura ... 67
Tabela 20 Descrição dos *bits* que definem o modo do *Timer/Counter0* [37] ... 84
Tabela 21 Descrição dos *bits* que selecionam o *prescaler* do *Timer/Counter0* [37] ... 85
Tabela 22 Valores associados a cada *Baud Rate* para a frequência de 16 MHz [37] ... 86
Tabela 23 Equações para o cálculo da *Baud Rate* ou do UBRRn [37] ... 86
Tabela 24 Cálculo do número do LED para linhas pares e ímpares da estrutura ... 90
Tabela 25 Tramas enviadas para a aplicação Android ... 110
Tabela 26 Tempo de envio necessário para cada trama ... 114
Tabela 27 Tempo de envio necessário para uma mensagem MIDI ... 116
Tabela 28 Tramas enviadas pela a aplicação Android ... 118
Tabela 29 Áreas da atividade principal ... 129
Tabela 30 Classes Bluetooth ... 135
Tabela 31 Estado da ligação Bluetooth ... 141
Tabela 32 Valor do argumento na trama do controlo da matriz ... 144
Tabela 33 Configurações implementadas para sistema de quatro colunas ativas ... 150
Tabela 34 Atribuição de linha e coluna consoante rotação do ícone ... 162

Acrónimos

API	–	*Application Programming Interface*
APK	–	*Android Package*
BPM	–	Batimentos por Minuto
CSV	–	*Comma-Separated Values*
DAW	–	*Digital Audio Workstation*
EEPROM	–	*Electronically Erasable Read-Only Memory*
IDE	–	*Integrated Development Environment*
LED	–	*Light Emitting Diode*
MIDI	–	*Musical Instrument Digital Interface*
PCB	–	*Printed Circuit Board*
PWM	–	*Pulse Width Modulation*
RGB	–	*Red, Green and Blue*
SDK	–	*Software Development Kit*
SPI	–	*Serial Peripheral Interface*
USART	–	*Universal Synchronous and Asynchronous Serial Receiver and Transmitter*
XML	–	*eXtensible Markup Language*

1. INTRODUÇÃO

O presente relatório foi realizado no âmbito da unidade curricular de TEDI, Tese/Dissertação, pertencente ao último ano do Mestrado de Engenharia Eletrotécnica e de Computadores (MEEC), do Departamento de Engenharia Eletrotécnica (DEE), lecionada no Instituto Superior de Engenharia do Porto, no qual é explicado o desenvolvimento do projeto.

1.1. MOTIVAÇÃO

Este projeto surgiu da intenção de realizar um trabalho no âmbito do panorama musical. Relacionado com a opção deste projeto está o facto de ser desenvolvido de forma a inovar certos instrumentos musicais, assim como facilitar a aprendizagem de música pelas pessoas, principalmente as crianças.

Com o desenvolvimento do projeto também foi possível adquirir competências na programação de aplicações para *smartphones* com sistema operativo Android, o que se tornou bastante motivador por ser uma das linguagens de programação mais utilizadas nos dias de hoje.

1.2. OBJETIVOS

O objetivo principal deste projeto é a criação de um sequenciador MIDI físico, composto por 144 posições, sendo uma matriz física de 12 linhas por 12 colunas.

O utilizador coloca bolas em cada um dos buracos com o intuito de produzir notas musicais, à medida que um compasso de tempo atinge a posição das bolas. Também tem à disposição um modo sem bolas que, como o próprio nome indica, não necessita de posicionar as bolas nas posições, mas sim pressionar os botões de cada uma.

O sistema será capaz de determinar a posição em que a bola é inserida, sendo iluminada consoante o local onde é colocada. Também possibilitará ao utilizador escolher o instrumento e a nota musical de cada coluna, funcionando como um controlador MIDI, dado que comunicará com um *software* que interpreta o protocolo, presente num computador.

Por último, será desenvolvida uma aplicação Android para *smartphone* com o intuito de comunicar com o sistema de maneira a monitorizar e alterar configurações do mesmo.

Desta forma, o desenvolvimento do projeto foi dividido nas seguintes tarefas:

- Estudo e análise de sistemas similares;
- Desenvolvimento de um sistema capaz de interpretar cada posição da estrutura, e iluminá-la com a cor correta;
- Interligação do sistema geral com um *software* MIDI, que ao receber as mensagens executa-as, tais como mudanças de instrumento ou produção de notas musicais;
- Construção de uma estrutura com o objetivo de interagir o sistema com o utilizador;
- Desenvolvimento de uma aplicação Android capaz de monitorizar e alterar parâmetros do sistema.

1.3. CALENDARIZAÇÃO

Nesta subsecção são demonstradas as etapas que se teve em conta no desenvolvimento do projeto. Como este foi desenvolvido em dois anos letivos, a calendarização visualizada na Tabela 1 é relativa ao segundo semestre de 2016/17, enquanto que a ilustrada na Tabela 2 diz respeito a todo o ano letivo de 2017/18:

Tabela 1 Calendarização relativa ao segundo semestre de 2016/17

Tarefa	2017							
	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out
Estudo de projetos similares	■							
Estudo do microcontrolador e componentes	■	■	■					
Estudo do protocolo MIDI		■	■					
Desenvolvimento *software*			■	■	■			
Desenvolvimento aplicação Android				■	■			
Construção da estrutura						■	■	
Implementação *hardware*								■
Elaboração do relatório		■	■	■			■	■

Tabela 2 Calendarização relativa a todo o ano letivo de 2017/18

Tarefa	2017		2018					
	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun
Implementação *hardware* (correção)	■	■						
Desenvolvimento *software*			■	■	■			
Desenvolvimento aplicação Android			■	■	■			
Elaboração do relatório						■	■	■

1.4. ORGANIZAÇÃO DO RELATÓRIO

O presente relatório é dividido em oito capítulos.

No capítulo atual (Capítulo 1) é realizada uma breve apresentação sobre o projeto desenvolvido, quais os seus objetivos, como também é descrita a organização do relatório.

No Capítulo 2 é abordado a origem da música e dos sintetizadores. Além deste instrumento musical eletrónico, também é feita uma descrição de outros, nomeadamente do sequenciador original denominado GRIDI.

O Capítulo 3 analisa as tecnologias que foram estudadas para o desenvolvimento do sistema, sendo elas o protocolo MIDI e a linguagem de programação Android.

O Capítulo 4 é referente à arquitetura do sistema, isto é, ao estudo e análise que foram realizados antes da sua implementação.

No Capítulo 5 é feita uma análise ao sistema desenvolvido, à exceção do desenvolvimento da aplicação Android, cujo seu funcionamento é descrito no Capítulo 6.

Por último, no Capítulo 7, são apresentadas as conclusões e descritas possíveis melhorias futuras no projeto.

2. CONTEXTUALIZAÇÃO

Atualmente, o ser humano está repleto de música no dia a dia. Porém, pouco pode ser dito das origens da música, dado que as suas raízes apareceram na antiguidade humana. A música deriva de uma palavra grega que tem como significado “A Arte das Musas”, e defini-la não é fácil dado que é difícil encontrar um conceito que englobe todos os seus significados, apesar de ser imediatamente conhecida por qualquer pessoa, variando assim de acordo com a cultura e o contexto social. Resumindo, a música pode ser a combinação de sons e silêncios de uma maneira organizada, sendo composta por três componentes [1]:

- Melodia: voz principal do som.
- Harmonia: sobreposição de notas que servem de base para a melodia.
- Ritmo: marcação do tempo de uma música.

A música também pode ser conhecida como uma forma de arte, tendo diferentes abordagens. Numa abordagem naturalista, a música existe antes de ser ouvida, isto é, a música em si não constitui arte, mas criá-la e expressá-la sim. Outras pessoas já consideram que a música pode não pode funcionar sem que antes seja interpretada e entendida.

Por se tratar de uma arte temporal, isto é, por se desenvolver ao longo do tempo, o ser humano para estudar a música necessitou de gravar, representar e interpretar informação de diversos momentos ao longo do tempo, tendo sido criados para esse mesmo propósito aparelhos de gravação de som.

O primeiro dispositivo de gravação analógica e reprodução de sons, foi inventado em 1877, por Thomas Edison, denominando-se fonógrafo (Figura 1). Consistia num cilindro giratório e um grande bocal em forma de cone, que captava ondas sonoras e reproduzia-as como som.

Figura 1   Primeiro fonógrafo, inventado por Thomas Edison [2]

Após este equipamento ter sido inventado, ocorreu uma transformação relativa ao estudo da música, como também dos métodos pelos quais a música é feita. Isto resultou no desenvolvimento de instrumentos musicais totalmente eletrónicos, tais como o Theremin. Este instrumento, inventado em 1920, pelo russo Lev Sergeyevich Termen, mais conhecido pelo seu nome americano, Léon Theremin, possibilitava controlar a intensidade e a altura de um som, gerado por osciladores, através do movimento das mãos próximo de duas antenas.

De seguida será explicado como nasceu o tema de música por computador, isto é, como se começou a utilizar a área da computação para a produção de músicas. Também serão descritos alguns instrumentos musicais eletrónicos, começando com os sintetizadores. Estes instrumentos produzem música utilizando a eletrónica, e podem incluir uma *interface* de utilizador, que tem como função controlar o som, ajustando alguns parâmetros.

2.1. MÚSICA POR COMPUTADOR

O conceito de música por computador é a aplicação da tecnologia de computação em composição de música, com o objetivo principal de ajudar compositores a criar novas músicas.

O primeiro computador a produzir música foi o CSIRAC (*Council for Scientific and Industrial Research Automatic Computer*), o primeiro computador digital da Austrália, construído por Trevor Pearcey e Maston Beard. Este foi programado por Geoff Hill para produzir melodias musicais populares da década 50 [3].

Em meados de 1960, foi desenvolvido o primeiro programa de música por Max Mathews, nos laboratórios Bell, denominado MUSIC, com as diferentes versões assinaladas em numeração romana. O programa produziu um som pela primeira vez em 1957, e passado um ano, nasceu o MUSIC II, que demorava cerca de uma hora para gerar um minuto de música, no computador mais rápido da época.

A maior parte dos compositores das décadas de 60 e 70 não estavam envolvidos na área dos computadores, devido a várias razões tais como o acesso limitado aos mesmos, especialmente para estudantes, a dificuldade de programar, e a falta de interesse. Assim, em 1968, o criador do programa MUSIC, Max Mathews, desenvolveu um sistema que permitia a um computador controlar um sintetizador analógico em tempo real.

Com o programa GROOVE também era possível gravar as ações de um músico enquanto tocava e alterava controlos num sintetizador. Após efetuada a gravação, a lista de ações poderia ser editada e reproduzida no sintetizador novamente [4].

Devido ao impacto obtido pela música por computador na época, em 1970, cerca de uma dúzia de escolas estavam a estudá-la, e no final da década, existiam uma centena de universidades e centros de pesquisa a pesquisar e compreender a composição de músicas nos computadores. Nestes centros de pesquisa, o trabalho era realizado por *mainframes*, computadores de grande escala dedicados ao processamento de um elevado número de informações.

A criação de uma peça musical exigia bastante esforço dado que o compositor perdia dias a inserir informações de texto, para depois pedir à máquina para compilar os sons, levando horas, onde por último o arquivo seria lido da fita do computador, através de um conversor digital para analógico, para fita de gravação, que por fim o compositor ouviria.

Naquele período de tempo, os artigos científicos eram mais comuns que a música criada por computador. Estes documentos eram na sua maioria publicados no *"The Journal of the Audio Engineering Society"* e no *"Computer Music Journal"*.

O objetivo dos centros de pesquisa era então de desenvolver um sistema capaz de efetuar síntese em tempo real. Isto originou à criação de diversos instrumentos exclusivos, que em alguns casos tornaram-se produtos comerciais. Além deste tipo de *hardware* criado, as instituições também criaram novos *softwares*, tendo como base o MUSIC, de Max Mathews.

O aparecimento dos microprocessadores levou ao desenvolvimento dos primeiros computadores pessoais, como o Apple II. Estes componentes facilitaram a ligação dos computadores aos sintetizadores. A música por computador tornou-se mais acessível a músicos que não estavam relacionados com os centros de pesquisa. À medida que isto ocorria, os sintetizadores eram cada vez menos analógicos, dando origem à popularidade de sequenciadores e teclados digitais.

Na altura, um microprocessador podia ser programado para produzir dezasseis vozes, com algumas limitações, em que algumas placas eram comercializadas para serem inseridas nos computadores Apple II, vendidas por companhias como Mountain Hardware.

Em 1981 foi organizado um encontro entre fabricantes de instrumentos musicais com o objetivo de conectar qualquer computador a um qualquer sintetizador, dando origem à norma MIDI. A partir daí, dispositivos MIDI foram comercializados em lojas de música.

2.2. SINTETIZADORES

Os sintetizadores são instrumentos musicais eletrónicos desenvolvidos com o intuito de imitar ou conter uma grande variedade de sons, como os produzidos por instrumentos, vozes, entre outros. Estes equipamentos também possibilitam a formação de sons que não ocorrem no mundo natural, sendo esta a principal caraterística dos sintetizadores.

Num sintetizador, a geração de um tom é realizada através de um componente denominado oscilador. A maioria destes equipamentos geram formas de ondas, como ondas de dente de serra, triângulo, quadrado e pulso.

O processo de transformar um tom fundamental e seus harmónicos noutro som é executado endereçando o sinal de um componente, também conhecido como módulo, para um outro no sintetizador. Cada um destes módulos realiza um trabalho diferente que afeta o sinal de origem. Na maioria dos sintetizadores modernos, o encaminhamento do sinal entre módulos é internamente ligado, sendo normalmente alterado utilizando interruptores, botões e outro tipo de controlos [5].

2.2.1. ORIGEM

O termo foi introduzido em 1957, pela *Radio Corporation of America* (RCA), através da criação do RCA Mark II Sound Synthetizer, que viria a ser o primeiro sintetizador de som programável. Estes equipamentos como os conhecemos hoje em dia começaram a ser desenvolvidos por volta de 1964, por Robert Moog e o compositor Herbert Deutsch, e combinavam osciladores e módulos amplificadores com teclados.

A partir de 1968 o instrumento ganhou fama devido ao sucesso de vendas do disco *Switched-On Bach*, do músico Walter Carlos, passando a chamar-se sintetizador Moog. Devido à sua complexidade e ao seu custo, o inventor em conjunto com os engenheiros Jim Scott, Bill Hemsath e Chad Hunt, criou um sintetizador acessível, portátil, compacto e fácil de utilizar pelos músicos, tendo nascido em 1970 o Minimoog Model D, visualizado na Figura 2:

Figura 2 Minimoog Model D [6]

Posteriormente, foram projetados sintetizadores menores e polifónicos, como foi o caso do Polymoog em 1976 e, mais tarde no ano 1977, o modelo Prophet 5 criado pela empresa Sequential Circuits, que além de ser polifónico foi o primeiro sintetizador a possibilitar armazenamento, permitindo assim a gravação e utilização de sons modificados pelo utilizador.

2.2.2. ANALÓGICO VS DIGITAL

Um sintetizador analógico combina circuitos controlados por tensão, tais como osciladores, filtros e amplificadores, com o objetivo de gerar e produzir sons. Em contrapartida, um sintetizador digital utiliza técnicas de processamento de sinais digitais (DSP) para produzir sons musicais. Assim, pode-se concluir que a principal diferença entre os dois tipos é que enquanto o analógico utiliza circuitos analógicos, o digital usa processadores digitais.

Devido ao avanço da tecnologia na área dos computadores, é mais vantajoso optar por um sintetizador digital, dado que oferece em alguns casos mais recursos que o analógico.

2.3. ALPHASPHERE

Este instrumento musical eletrónico consiste numa esfera, composta por quarenta e oito *pads* táteis, que podem ser pressionados. Estes *pads* são programados através de um *software* específico, denominado AlphaLive, sendo atribuídas notas musicais a cada uma delas, assim como outras configurações.

O principal objetivo do dispositivo é ampliar o nível de expressão disponível para os músicos, permitindo-lhes tocar um instrumento musical diferente do habitual. É utilizada por músicos, compositores, DJs (*Disc Jockey*), produtores e VJs (*Video Jockey*), e tem aplicações na educação e terapia musical [7].

A Alphasphere foi inventada por Adam Place, que após a criação do protótipo, foi premiado com uma bolsa de estudos na universidade de artes de Nagoya, localizada no Japão. Ao regressar à sua cidade natal Bristol, no Reino Unido, fundou a empresa Nu-Desine, com o intuito de comercializar o projeto.

A Figura 3 ilustra todos os modelos atuais da Alphasphere:

a) b) c)

Figura 3 Alphasphere elite (a); Alphasphere nexus (b); Alphasphere me (c) [7]

A Alphasphere encontra-se disponível para venda a todo o mundo e, como visualizado na Figura 3, conta com três modelos diferentes:

a) Alphasphere elite: projetado para o estúdio, isto é, para os músicos trabalharem na composição de músicas.

b) Alphasphere nexus: criado para os músicos utilizarem em *performances* ao vivo.

c) Alphasphere me: concebido para tornar a música acessível a todos.

2.4. EIGENHARP

Eigenharp é uma criação da companhia Eigenlabs, situada em Devon, no Reino Unido. Foi inventada por John Lambert e lançada em 2009, após ter sido desenvolvida durante oito anos. Em termos físicos parece uma combinação de alguns instrumentos tais como teclado, bateria e saxofone, capaz de imitar diversos sons e efeitos, tornando o instrumento bastante versátil.

O instrumento musical eletrónico é composto por uma matriz de teclado, em que cada uma das teclas pode ser pressionada, possuindo três ajustes diferentes para controlo da intensidade e timbre do som. Além disso, a Eigenharp permite a gravação e reprodução de sons, e conta ainda com um tubo, que funciona como um instrumento de sopro, adicionando diversos efeitos à medida que o instrumento está a ser tocado.

A empresa Eigenlabs comercializa três modelos do instrumento, que por ordem crescente denominam-se Pico (Figura 4), Tau e Alpha, sendo este último o equipamento com um preço mais elevado. Cada um dos especificados contém um tubo, uma matriz de teclado, que tem 18, 72, ou 120 teclas, assim como um controlador. Os modelos Alpha e Tau têm também doze teclas, utilizadas para efeitos de percussão.

Figura 4 Modelos Pico da Eigenharp [8]

2.5. REACTABLE

O instrumento musical eletrónico denominado Reactable consiste numa mesa translúcida, sujeita a vários contactos simultâneos, onde o utilizador controla o sistema manipulando objetos reais que são interpretados como diferentes sinais musicais. Conforme a forma do objeto é possível acrescentar efeitos com o intuito de obter diferentes tipos de som [9]. A Figura 5 demonstra o instrumento:

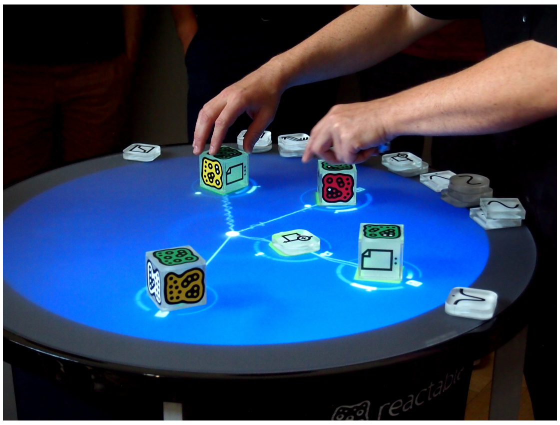


Figura 5 Reactable Experience [10]

Debaixo da mesa está presente uma câmara de vídeo que aponta para a parte inferior da banca, ilustra as imagens e envia os sinais presentes nos objetos colocados sobre a superfície para um computador responsável pela produção do som. Há também um projetor, também ligado ao computador, que projeta vídeo na parte de baixo do tampo da mesa, que pode ser visto pelo o utilizador no lado superior.

O segredo que possibilita ao instrumento produzir sons está nos objetos que o utilizador coloca sob a mesa, dado que cada um deles possui um componente especial denominado *tangible*. Cada *tangible* produz um efeito diferente, sendo visualizado na mesa através de quadrados ou círculos que se formam à volta dos objetos.

Cada objeto tem uma imagem impressa a preto e branco, que consiste em pontos e círculos, formando padrões otimizados para utilização por parte do reacTIVision. Este *software* foi desenvolvido por Martin Kaltenbrunner e Ross Bencina especificamente para este instrumento musical eletrónico para processar as informações recebidas por uma câmara de vídeo no computador.

O reacTIVision deteta o posicionamento cartesiano e rotativo das imagens presentes na superfície inferior de cada objeto posicionado na mesa. O protocolo TUIO funciona como elo de ligação entre a câmara de vídeo e o *software* de visualização, que por sua vez irá projetar os efeitos possíveis para cada objeto na tela através do videoprojector. A Figura 6 demonstra o que foi referido neste parágrafo:

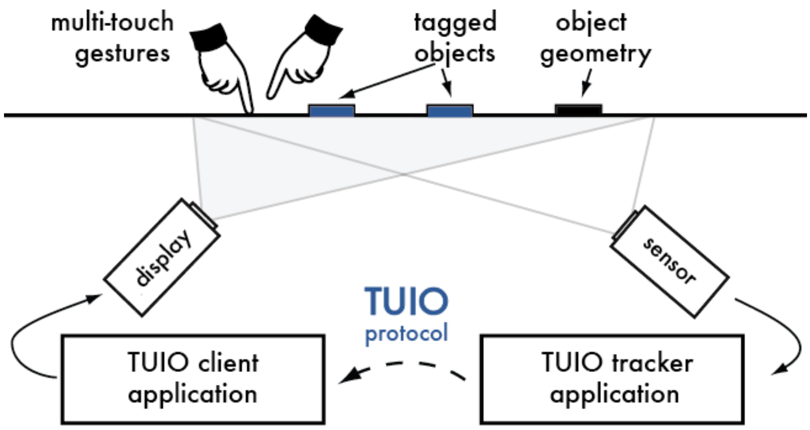


Figura 6 Aplicação do protocolo TUIO na Reactable [11]

A Reactable foi criada em 2003 por uma equipa de investigação da Universidade Pompeu Fabra, de Barcelona. O instrumento foi apresentado pela primeira vez num concerto, na International Computer Music Conference 2005, realizada na cidade onde foi desenvolvido.

Em 2009 foi fundada a empresa Reactable Systems com o intuito de continuar o desenvolvimento e aperfeiçoamento de capacidades da Reactable.

Nos dias de hoje, existem duas versões do instrumento, sendo elas a Reactable Live! e a Reactable Experience. Enquanto que a primeira é uma versão menor e mais portátil, projetada para músicos profissionais, a Reactable Experience é semelhante à primeira a ter sido desenvolvida, e é apropriada para instalações em espaços públicos.

2.6. HARPA LASER

A harpa laser é um instrumento musical eletrónico que projeta raios laser, onde o utilizador bloqueia-os com o objetivo de criar sons musicais. Consoante o feixe de luz interrompido é produzido um diferente som, que corresponde a uma nota musical diferente. Assim, o principio de funcionamento deste instrumento baseia-se na utilização do laser, acrónimo de *Light Amplification by Stimulated Emission of Radiation*.

A primeira harpa laser foi desenvolvida em 1976 por Geoffrey Rose, e foi popularizada pelo cantor e produtor Jean Michel Jarre, dado que era bastante utilizada nos seus concertos. A Figura 7 ilustra a harpa laser utilizada pelo cantor num dos seus concertos:

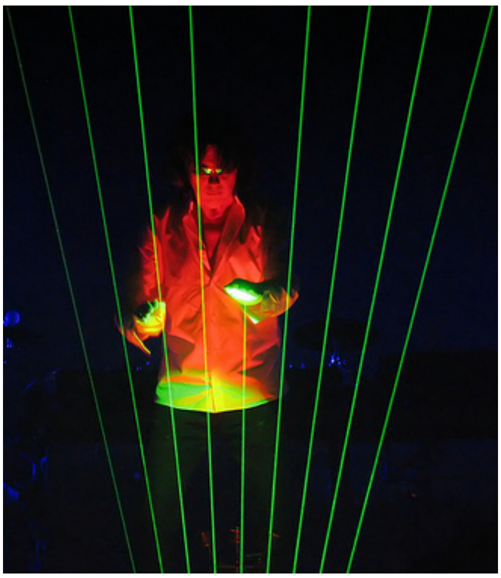


Figura 7 Harpa Laser utilizada por Jean Michel Jarre num dos seus concertos [12]

O *design* da harpa laser pode ser dividido em dois tipos: *framed* e *frameless* [13].

2.6.1. HARPA *FRAMED*

A harpa *framed* (Figura 8) é constituída por uma estrutura física, onde se colocam na base os lasers em paralelo uns com os outros, sendo assim utilizado um laser distinto para retratar cada "corda" do instrumento.

Cada feixe é apontado para um sensor de luz, estando estes inseridos na extremidade oposta, normalmente o topo da estrutura, que determinam quando o feixe laser é bloqueado pelo utilizador, dado que o sensor irá deixar de receber a luz do respetivo feixe. O sinal é enviado para um processador, que por sua vez irá interpretar qual o feixe de luz que foi interrompido, com o intuito de produzir a nota associada.

Em termos de construção é o tipo de harpa mais simples de se criar, sendo também de menor custo dado que é necessário a incorporação de uma estrutura. Este estilo não é bastante utilizado pelos músicos nos seus concertos, pois impossibilita o prolongamento dos feixes laser no espaço.

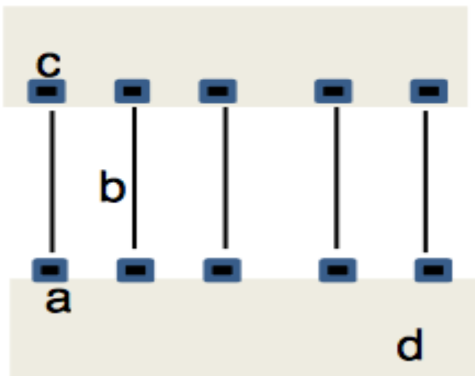


Figura 8 Harpa *Framed* [13]

A Figura 8 representa a constituição de uma harpa laser do tipo *framed*, onde as letras *a* e *c* representam, respetivamente, as posições dos lasers e dos sensores de luz, o caracter *b* identifica o espaço que o utilizador pode interromper o feixe de luz e, por último, a letra *d* mostra o tipo de estrutura mais utilizado nestas harpas, que é um género de moldura.

2.6.2. HARPA *FRAMELESS*

Como o nome indica, este estilo de harpa não possui uma estrutura física como a harpa *framed*, não havendo contenção dos feixes de luz emitidos. A harpa *frameless* tradicional utiliza um motor de passo em conjunto com um espelho para transmitir os feixes, de acordo com o ângulo que o motor coloca no espelho, sendo necessário somente um laser com uma potência elevada, como ilustrado na Figura 9.

Esta mudança do motor constante cria um efeito semelhante a um leque entre os feixes luminosos. Neste tipo de harpa existem diferentes métodos para processar o bloqueio do feixe luminoso, sendo o principal a utilização de um sensor de luz capaz de captar um aumento da intensidade luminosa, dado que o utilizador ao interromper um feixe reflete o laser. Assim, com o processador determina-se a posição que se encontrava o motor de passo, quando o feixe laser foi bloqueado, e é originado o som associado a este mesmo.

Em comparação com o outro estilo de harpa (*framed*), a harpa *frameless* é mais complexa. No entanto, como referido, também permite uma melhor performance, sendo assim o tipo de harpa escolhido para utilização em concertos e festivais ao vivo.

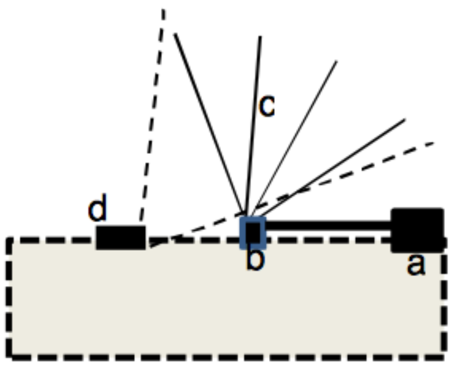


Figura 9 Harpa *Frameless* [13]

Como se pode visualizar na Figura 9, a harpa *frameless* é constituída por um laser, identificado pelo caracter *a*, que aponta o equipamento da letra *b*, que consiste num espelho associado a um motor de passo, que irá produzir os feixes luminosos vistos pela letra *c*. Por último, o caracter *d* ilustra a posição do sensor de luz, que visualiza quando um feixe laser é interrompido.

2.7. GRIDI

GRIDI, junção da palavra *Grid* com o acrónimo MIDI, é um sequenciador MIDI físico em grande escala, desenvolvido pelo produtor musical Yuval Gerstein, com o objetivo de permitir aos utilizadores, sendo músicos ou não, de criar uma composição musical de uma forma acessível e intuitiva. O compositor inspirou-se numa pintura do britânico Damien Hirst (Figura 10), denominada *Spot Paintings*, situada no Museu de Israel, em Jerusalém, onde o projeto GRIDI foi exibido ao público pela primeira vez, em agosto de 2015.

Figura 10 *Spot Paintings*, de Damien Hirst [14]

Foi desenvolvido com um tamanho de 2.80m de comprimento e 1.65m de largura, sendo constituído por quatro placas MDF (*Medium-Density Fiberboard*), cujo material é derivado da madeira, onde cada posição foi furada por uma máquina CNC (*Computer Numerical Control*). No final formam uma matriz física de 16x16, totalizando 256 buracos para serem posicionadas as bolas transparentes.

Na Figura 11 é demonstrado o sistema GRIDI:

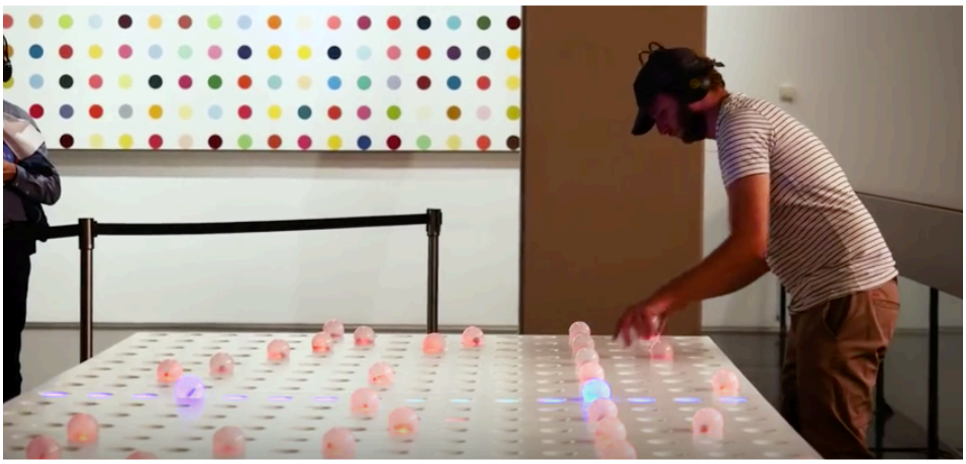


Figura 11 GRIDI [14]

A ideia original é de possibilitar às pessoas compor uma faixa, colocando bolas transparentes no sistema. Ao pôr a bola é acendida a luz naquela respetiva posição. Um cursor verde move-se no tempo e quando alcança a bola, toca o som relativo à sua posição.

2.7.1. FUNCIONAMENTO

O projeto consiste numa matriz física de 16x16, como referido anteriormente. Cada buraco é constituído por um botão e um LED. Este último é programado para acender de acordo com o *tempo* da faixa, isto é, tendo em conta a velocidade do compasso. O cursor percorre a estrutura da esquerda para a direita em dezasseis colunas, que corresponde a um todo de dezasseis notas diferentes. Assim que a batida alcançar a bola transparente, irá produzir o som associado.

Além disto, quando posicionada uma bola transparente, o LED irá iluminar a esfera, dependendo do instrumento associado ao posicionamento da mesma. Tem-se como opção criar padrões de bateria, de melodia e de baixo, em que sete colunas estão atribuídas para os tons de melodia, cinco para os sons de melodia, e os restantes quatro para baixo. A cada um destes instrumentos é atribuída uma cor. A figura seguinte ilustra um exemplo da configuração dos instrumentos no sistema [14]:

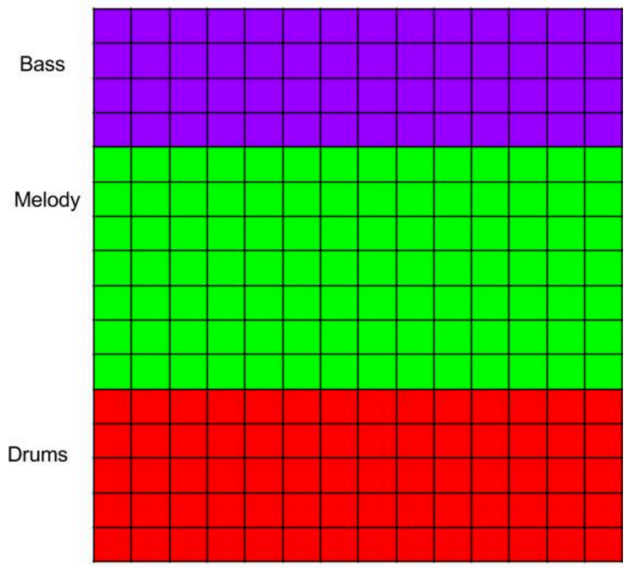


Figura 12 Exemplo de configuração dos instrumentos no sistema GRIDI [14]

Existem dois tipos de pessoa no sistema GRIDI:

- **Utilizador**: são os visitantes que compõem a música, colocando as bolas em sítios diferentes da estrutura;

- **Administrador**: normalmente será uma pessoa que tenha conhecimentos na área, dado que irá fazer efeitos e alterações na velocidade do cursor, sendo utilizado um programa denominado *touchable*, utilizado num iPad, para controlar o *software* que produz as notas no computador portátil.

O visitante ao colocar a bola transparente no sítio pretendido, está a acionar um botão, que se encontra por debaixo do LED. Os botões funcionam como entradas que ao serem premidos, acendem LEDs com certas cores, consoante o instrumento associado, que são as saídas do microcontrolador. À medida que um LED é ativado, é enviado um sinal para o computador que indica a escrita de uma nota MIDI, no local do componente.

No sequenciador está também presente um modo manual que permite aos utilizadores inserir notas, pressionando os botões sem a necessidade de colocar bolas nas posições, utilizando somente os dedos para os acionar.

O controlo do sistema é efetuado por um Arduino Uno, em conjunto com um computador portátil, um Macbook Pro, a correr o *software* Ableton Live, que é um dos DAWs mais utilizados atualmente, dado que é designado para ser um instrumento para *performances* ao vivo, ou uma ferramenta para compor, gravar, entre outras funções. Neste projeto, o Ableton é utilizado para definir os sons associados a cada posição da matriz.

3. TECNOLOGIAS ENVOLVIDAS

Neste capítulo são analisadas as tecnologias importantes no desenvolvimento deste projeto, sendo elas o protocolo MIDI e o sistema operativo Android.

3.1. MIDI

MIDI, acrónimo de *Musical Instrument Digital Interface*, é um protocolo utilizado na indústria da música que interliga instrumentos musicais eletrónicos, computadores, *tablets* e *smartphones*. Músicos, produtores, artistas e DJs utilizam o padrão para aprender, criar, executar e compartilhar música e trabalhos artísticos. Resumindo, a norma permite que vários instrumentos sejam tocados num único controlador, tal como um teclado [15].

3.1.1. ORIGEM

No final da década de 1970, registou-se na América do Norte, Europa e Japão, um enorme desenvolvimento dos instrumentos musicais eletrónicos. Devido aos baixos custos dos microprocessadores e à produção em massa de circuitos integrados, os fabricantes de sintetizadores começaram a implementar circuitos digitais em instrumentos musicais.

Mais tarde foram criadas *interfaces* digitais de controlo que permitiriam a interligação dos seus próprios produtos, tendo como grande desvantagem a incompatibilidade entre sistemas de diferentes fabricantes. Portanto, em junho de 1981, o fundador da Roland Corporation, Ikutaro Kakehashi propôs a ideia de padronização para instrumentos de fabricantes diferentes, tornando possível a comunicação entre eles.

Em outubro de 1981, acompanhado por Tom Oberheim, fundador da empresa norte-americana Oberheim Electronics, e Dave Smith, presidente da Sequential Circuits, discutiu a ideia com representantes da Yamaha, Korg e Kawai, que passado um mês fez com que Dave Smith propusesse o conceito à AES, *Audio Engineers Society*, em Nova Iorque [16].

Num encontro da NAMM (*National Association of Music Merchants)* decorrido em janeiro de 1982, Kakehashi, Oberheim e Smith convocaram uma reunião que teve a participação de representantes de vários fabricantes de sintetizadores. A Sequential Circuits, juntamente com empresas japonesas, tais como Yamaha, Korg e Kawai, uniram-se com o objetivo de manter o projeto e, assim, foi definida a primeira especificação técnica, que ficou denominada por *Musical Instrument Digital Interface*, abreviado MIDI.

Na convenção NAMM de janeiro de 1983, Dave Smith demonstrou uma ligação MIDI entre o sintetizador Prophet 600 da Sequential Circuits (Figura 13), e um Roland JP-6 (Figura 14):

Figura 13 Prophet 600 da Sequential Circuits [17]

Figura 14 Roland JP-6 [18]

Com a experiência das primeiras implementações, foram efetuados ajustes ao padrão, e em agosto de 1983, o MIDI 1.0 foi lançado. Atualmente, a norma MIDI encontra-se ainda na versão 1.0, ou seja, desde 1983 até os dias de hoje, qualquer produto pode ser ligado a outro de um fabricante diferente, estabelecendo uma comunicação entre os dois.

Apesar de manter as suas caraterísticas básicas, ao protocolo têm sido feitas adições ao longo dos anos, tornando-o mais abrangente e capaz de lidar com uma maior diversidade de caraterísticas dos instrumentos que possuam MIDI.

3.1.2. ESPECIFICAÇÃO

O protocolo MIDI especifica uma linguagem de dados em série, sendo estes formados por mensagens MIDI, em que cada uma delas é transmitida a uma velocidade de 31250 bps, tendo sido escolhido este valor por ser a divisão exata de 1 MHz por 32, dado que esta frequência era a velocidade utilizada pela primeira geração de microprocessadores.

As mensagens MIDI são compostas por informações de 8 bits, isto é, 1 byte, sendo adicionado mais dois bits denominados *Start* e *Stop*, com a finalidade de sincronizar os dados, em que o primeiro tem sempre o valor lógico 0, e o segundo o valor lógico 1. Assim, cada byte MIDI necessita de 10 bits para transmissão, como é demonstrado na Figura 15, onde são enviados três bytes:

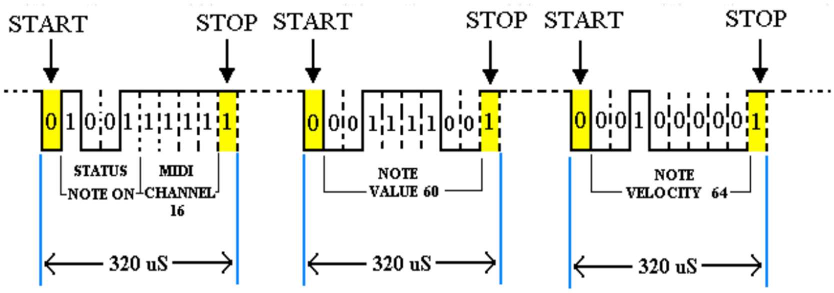


Figura 15 Formato das mensagens MIDI [19]

Uma ligação MIDI pode conter até 16 canais independentes, permitindo que os instrumentos musicais sejam divididos e direcionados por canais diferentes, e que sejam criados sistemas eletrónicos musicais personalizados. Esta informação é transportada através de um único cabo, em que cada canal transporta a informação para um determinado instrumento.

Um instrumento ao receber mensagens MIDI interpreta os dados de acordo com o modo de operação. Cada modo é composto por duas partes:

- ***Omni On/Off***: define se o dispositivo é configurado para ouvir e responder somente a canais específicos, ignorando mensagens enviadas noutros canais – *Omni Off* – ou se pode ouvir e responder em todos os canais – *Omni On*.

- ***Mono/Poly:*** um dispositivo pode ser monofónico (*Mono*), em que somente uma nota é tocada cada vez, ou polifónico (*Poly*), onde várias notas podem ser ouvidas simultaneamente.

O MIDI possui quatro modos de operação, combinando a parte *Omni On/Off* com *Mono/Poly*. A Tabela 3 descreve cada modo de operação:

Tabela 3 Modos de operação do MIDI

Modo	*Omni*	*Poly/Mono*	Descrição
1	*On*	*Poly*	O dispositivo responde aos dados MIDI independentemente do canal, sendo polifónico.
2	*On*	*Mono*	O dispositivo responde aos dados MIDI independentemente do canal, sendo monofónico.
3	*Off*	*Poly*	O dispositivo responde aos dados MIDI apenas no canal especificado, sendo polifónico.
4	*Off*	*Mono*	O dispositivo responde aos dados MIDI apenas no canal especificado, sendo monofónico.

3.1.3. MENSAGENS

MIDI baseia a sua linguagem em mensagens utilizadas por equipamentos para se comunicarem entre eles. Uma mensagem é definida como uma instrução capaz de controlar várias configurações do dispositivo recetor.

A estrutura das mensagens MIDI consiste num byte de estado, que indica o tipo de mensagem, seguido por um ou dois bytes de dados, que contêm os parâmetros. Em relação ao byte de estado: o bit mais significativo é colocado a '1'; os últimos quatro bits identificam o canal; os restantes três bits identificam a mensagem. O byte de dados é definido com o bit mais significativo a '0', e os restantes referem-se ao parâmetro a ser enviado.

As mensagens MIDI podem ser classificadas de duas maneiras, havendo no total cinco formatos [20]:

- **Mensagens do Canal:** mensagens que são transmitidas para canais específicos, e não globalmente para todos os dispositivos MIDI.
    - Mensagens de Voz
    - Mensagens de Modo
- **Mensagens do Sistema:** mensagens que carregam informações que não são específicas a um canal, mas sim a todos.
    - Mensagens Comuns
    - Mensagens em Tempo Real
    - Mensagens Exclusivas

Mensagens de Voz do Canal

Os dispositivos MIDI são equipados para receber mensagens em um ou mais canais. A voz específica de um equipamento responde às mensagens enviadas para o canal configurado, ignorando as restantes. Estas mensagens transmitem informações sobre a ativação ou desativação de uma nota, a quantidade de pressão exercida numa tecla (denominada *aftertouch*), entre outras. A Tabela 4 mostra quais as mensagens MIDI mais comuns, pertencendo a este formato, e as suas caraterísticas:

Tabela 4 Caraterísticas das mensagens de voz do canal [20]

Status Byte	Data Byte 1	Data Byte 2	Message	Legend
1000nnnn	0kkkkkkk	0vvvvvvv	Note Off	n=channel* k=key # 0-127(60=middle C) v=velocity (0-127)
1001nnnn	0kkkkkkk	0vvvvvvv	Note On	n=channel k=key # 0-127(60=middle C) v=velocity (0-127)
1010nnnn	0kkkkkkk	0ppppppp	Poly Key Pressure	n=channel k=key # 0-127(60=middle C) p=pressure (0-127)
1011nnnn	0ccccccc	0vvvvvvv	Controller Change	n=channel c=controller v=controller value(0-127)
1100nnnn	0ppppppp	[none]	Program Change	n=channel p=preset number (0-127)
1101nnnn	0ppppppp	[none]	Channel Pressure	n=channel p=pressure (0-127)
1110nnnn	0fffffff	0ccccccc	Pitch Bend	n=channel c=coarse f=fine (c+f = 14-bit resolution)

Como exemplo, será demonstrado como enviar uma mensagem MIDI para tocar uma nota (*Middle C)*, no canal 5, a um som bastante alto, isto é, define o parâmetro velocidade com o valor 127. Assim, o comando a ser enviado é demonstrado na seguinte tabela [20]:

Tabela 5 Exemplo de uma mensagem MIDI [20]

status byte	data byte	data byte
10010100	00111100	01111111

Os primeiros quatro bits do byte de estado (1001) indicam que a mensagem será um comando *Note On*, enquanto que os outros quatro indicam ao MIDI o canal que a mensagem é para ser enviada, que varia entre os valores 0000 (canal 1) e 1111 (canal 16). O primeiro byte de dados indica qual nota deve ser tocada, que neste caso, como referido anteriormente, será *Middle C*, que corresponde ao valor decimal 60. Já o segundo byte de dados diz ao MIDI quão alto para tocar a nota.

Mensagens de Modo do Canal

As mensagens de modo do canal tem como função especificar aos instrumentos como processarão as mensagens referidas no ponto anterior (mensagens de voz do canal).

Existem quatro tipos de modos de canal:

- *Omni;*
- *Mono/Poly;*
- *All Notes Off;*
- *Local Control.*

A tabela seguinte ilustra as mensagens que podem ser enviadas neste formato:

Tabela 6 Caraterísticas das mensagens de modo do canal [20]

Status Byte	Data Byte 1	Data Byte 2 (& 3)	Description
1011nnnn	01111010 (122)	00000000 (0) = off 01111111 (127) = on	Local Control
1011nnnn	01111011 (123)	00000000 (0)	All Notes OFF
1011nnnn	01111100 (124)	00000000 (0)	Omni Mode OFF (all notes off as well)
1011nnnn	01111101 (125)	00000000 (0)	Omni Mode ON (all notes off as well)
1011nnnn	01111110 (126)	0mmmmmmm (m=number of channels*)	Mono Mode ON (poly mode off, all notes off as well)
1011nnnn	01111111 (127)	00000000 (0)	Poly Mode OFF (Mono Mode OFF, all notes off as well)

A mensagem *All Notes Off* é utilizada em programas de sequenciação, em que as mensagens perdidas deixam uma nota a soar por tempo indefinido, dado que não recebe o comando *Note Off*.

O comando *Local Control* foi designado para separar a função do teclado de um sintetizador, da sua capacidade de produzir som, sendo útil quando o pretendido é obter de um computador uma determinada nota, evitando que sejam tocadas em simultâneo várias notas.

O modo *Omni* se ativado (*Omni Mode On*) habilita que todas as vozes de um dispositivo respondam a todas as mensagens de voz do canal recebidas, não importando o canal onde são enviadas.

Os modos *Mono/Poly* modificam as caraterísticas polifónicas de uma ou mais vozes. Por exemplo, um novo comando *Note On* no formato *Mono* termina a produção da nota anterior, para que seja tocada uma nova nota.

Como referido anteriormente, existem quatro combinações dos modos *Omni* e *Mono/Poly*, que se encontram explicadas na Tabela 3.

Mensagens Comuns e em Tempo Real do Sistema

As mensagens de sistema têm como função endereçar todos os equipamentos, presentes num sistema.

O primeiro tipo de mensagens transmite informações que lidam principalmente com sequenciadores de instrumentos, onde são gravadas sequências MIDI. Os comandos são constituídos por um byte de estado, seguido por um ou dois bytes de dados.

Um exemplo de uma mensagem de sistema é *Song Position Pointer*, que é um registo que guarda o número de batidas MIDI, desde o início de uma sequência.

As mensagens em tempo real do sistema contêm um relógio que envia 24 *clocks* por *quarter note*, que é uma nota tocada por um quarto da duração de uma nota inteira. Também podem enviar mensagens *Start* e *Stop* para um sequenciador, *Active Sensing* que envia um byte de estado em cada 300ms, e por último, foi adicionado o comando *System Reset*, que reinicia o sistema, retornando as condições do recetor para a inicial.

Estas mensagens são compostas por apenas um byte de estado, não contendo nenhum byte de dados. Na Tabela 7 encontra-se a demonstração destes dois formatos de mensagens de sistema:

Tabela 7 Caraterísticas das mensagens comuns e em tempo real do sistema [20]

Status Byte	Data Byte 1	Data Byte 2	Description
system commom			
11110010	0lllllll	0hhhhhhh	Song Position Pointer (l=least significant bit, h=most significant bit)
11110011	0sssssss		Song Select (s=song number)
11110110	none		Tune Request
11110111	none		EOX (end of system exclusive mssage)
system real time			
11111000	none		Timing Clock
11111010	none		Start (song)
11111011	none		Stop
11111110	none		Active Sensing
11111111	none		System Reset

Mensagens Exclusivas do Sistema

Embora haja uma grande variedade de mensagens para os controladores MIDI, foram criadas mensagens exclusivas do sistema que ampliam a funcionalidade da MIDI, sendo uma das principais razões para a flexibilidade e longevidade do protocolo.

As mensagens *SysEx* (*System Exclusive*) foram projetadas de modo a que os fabricantes criem as suas próprias mensagens MIDI, permitindo controlar os seus equipamentos, havendo um melhor controlo que o facultado pelas mensagens do padrão.

Um comando exclusivo de um sistema inicia com o byte 11110000, seguido por um identificador único, atribuído a cada fabricante, por um certo número de bytes de dados e, por último, terminando com 11110111, que indica o término da mensagem *SysEx*. A maioria dos instrumentos também inclui uma configuração *SysEx*, para que mais de um dos mesmos instrumentos possam estar na mesma rede, trabalhando independentemente.

Estas mensagens podem incluir funcionalidades além das que são fornecidas pelo padrão, dado que são direcionadas para um determinado instrumento, sendo por isso ignoradas por todos os outros dispositivos presentes no sistema. Alguns exemplos da sua utilização são:

- Editar parâmetros dos sintetizadores;
- Guardar dados relativos ao som;
- Transmitir *samples*, isto é, pequenos trechos/fragmentos sonoros obtidos de gravações anteriores para posterior reutilização.

3.1.4. CONTROLADORES

No MIDI, o termo controlador refere-se ao equipamento MIDI que gere e transmite dados para outros dispositivos, com o objetivo de controlar parâmetros de uma performance musical.

O controlador é empregado para tocar sons de outros aparelhos, ou os seus próprios sons. Um exemplo deste tipo de controlador é um sintetizador MIDI de teclado, que além de funcionar como módulo de som, controla os sons de outros instrumentos sem que o seu próprio som atrapalhe a execução.

Nos seus primórdios, MIDI foi projetado para os controladores descritos no parágrafo anterior (teclados), considerando outros tipos como sendo alternativos, tendo sido visto como um impedimento para compositores que não estavam interessados na produção de música baseada em teclados. Mais tarde, a interface MIDI viria a ser introduzida a outros tipos de controladores devido à sua flexibilidade, tais como violinos, violoncelos, instrumentos de sopro, guitarras, baterias, entre outros.

Os controladores de instrumentos com cordas, tais como as guitarras, podem ser ajustados para digitalizar a saída do instrumento, possibilitando a produção de sons de um sintetizador, atribuindo um canal MIDI para cada corda. Em contrapartida, uma guitarra MIDI em comparação com um teclado torna-se mais lenta na transmissão dos dados, dado que tem dificuldade em determinar a afinação de cada nota.

Os instrumentos de percussão tais como baterias são os controladores que apresentam maior desempenho, após os teclados. Estes controladores podem assumir a forma de baterias, superfícies de controlo autónomas, ou a aparência de instrumentos de percussão acústicos.

Outro tipo de controladores MIDI são controladores de vento, que permitem tocar MIDI da mesma forma que instrumentos de sopro. Um controlador de vento possui um sensor que converte a pressão de respiração em dados sobre o volume. Exemplos deste tipo de controladores são o *Electronic Wind Instrument* (EWI) e o *Electronic Valve Instrument* (EVI) da empresa Akai.

Apesar do tipo de controlador, há certos parâmetros que todos eles utilizam, dado que cada um deles proporciona sinais de controlo que geram os seguintes aspetos [21]:

- *Pitch* de um evento de uma nota;
- Principio e fim de um evento de uma nota;
- Volume de uma nota;
- Alterações no *pitch*;
- Alterações na modulação;
- Controlos adicionais.

3.1.5. HARDWARE

Para ser possível trabalhar com MIDI é necessário um conjunto de requisitos no que toca diz respeito ao *hardware*. Cada instrumento que tenha o protocolo MIDI é equipado com um transmissor e recetor, embora em alguns dispositivos, tais como processadores de sinal, possam ter somente um ou outro. Como especificado subsecção 3.1.2, a interface opera a uma taxa de transmissão de 31250 bps [20].

A conexão entre dispositivos MIDI é realizada através de cabos, em que as pontas possuam conetores DIN de cinco pinos macho. A Figura 16 descreve cada terminal destes conetores, onde o pino 1 e 3 não são utilizados, o pino 2 corresponde ao *shield*, que evita que a informação seja interferida por outras frequências, o pino 4 corresponde à terra, e o pino 5 é o transmissor da informação MIDI.

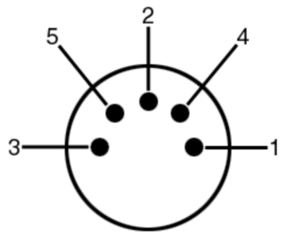


Figura 16 Conetor DIN de cinco pinos macho

Os cabos MIDI conectam os instrumentos através das ligações MIDI IN, MIDI OUT e MIDI THRU. A entrada MIDI IN recebe as mensagens MIDI e envia-as para o instrumento, para serem analisadas e, por sua vez, darem instruções ao mesmo. A porta MIDI OUT transmite ações que estejam a ocorrer no instrumento, e envia essa informação para outros ou para um computador. MIDI THRU duplica a informação recebida na entrada MIDI IN, com o objetivo de servir de elo de uma cadeia.

Qualquer ação num instrumento que corresponda a um comando MIDI específico, tais como pressionar uma tecla ou alterar um botão, normalmente transmite a mensagem para MIDI OUT, mas não para MIDI THRU. Porém, os instrumentos mais recentes possuem um interruptor que pode alterar a função de uma porta MIDI THRU para MIDI OUT.

Na Figura 17 pode-se visualizar uma rede de instrumentos conectada a um computador, através de uma interface MIDI, que tem como principal função corresponder as velocidades entre as taxas especificadas pelos instrumentos MIDI e o computador.

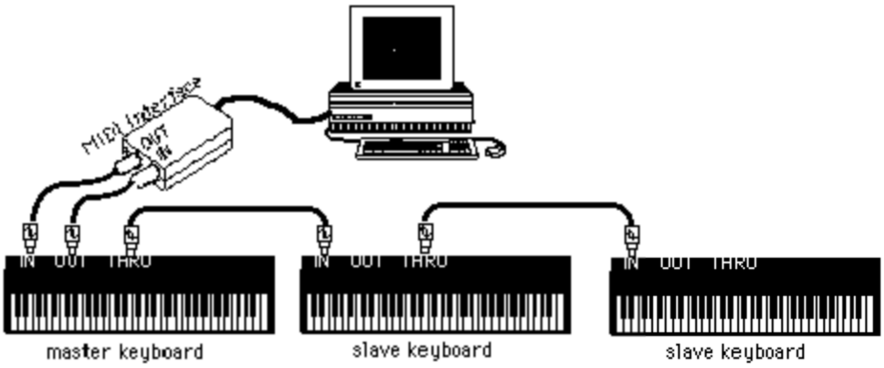


Figura 17 Rede MIDI [20]

3.1.6. TIPOS DE SOFTWARE

Hoje em dia existem diversos tipos de *software* diferentes que utilizam o protoloco MIDI, o que torna difícil a distinção de cada um deles. Os *softwares* mais utilizados na interligação de instrumentos MIDI e de computadores são programas de sequenciação MIDI.

Além dos programas de sequenciação MIDI, os mais famosos *softwares* são instrumentos virtuais, que consistem essencialmente na simulação de instrumentos reais, sendo na sua maioria sintetizadores e *samplers*, em que os sons são produzidos através de um teclado ou controlador MIDI; *workstations* MIDI, também denominados DAWs, que são caraterizados por possibilitarem a gravação, reprodução e edição de música usando MIDI e/ou áudio. Estes últimos em comparação com os sequenciadores MIDI não suportam *plugins* externos [21].

Sequenciadores MIDI

Um sequenciador MIDI não grava som, mas sim mensagens MIDI. Assim, as informações são gravadas e editadas em faixas individuais, para depois serem atribuídas a um ou mais canais MIDI.

Em modo de gravação recebe as mensagens, armazena-as e ordena-as, etiquetando-as tendo em conta a receção de cada uma delas.

Quando em modo de reprodução, vai verificar as mensagens que foram gravadas, tocando-as de acordo com a ordem. As faixas gravadas anteriormente, podem ser reproduzidas enquanto novas gravações estiverem a serem registadas.

Após a gravação das faixas, as mesmas podem ser editadas, recorrendo a diferentes maneiras como selecionar uma área gravada, transpô-la, encurtar durações rítmicas e incrementar velocidade, para obter um aumento gradual do volume.

Nos primórdios, os sequenciadores MIDI vinham na forma de dispositivos físicos, isto é, em *hardware*, oferecendo naquela época maior fiabilidade na sequenciação do que qualquer computador. À medida que os anos foram avançando, a importância destes dispositivos foi diminuindo, devido ao crescimento da tecnologia nos computadores, que fez com que os sequenciadores fossem produzidos em forma de *softwares*.

Atualmente, alguns sintetizadores possuem pequenos sequenciadores, que permitem trabalhar com autonomia. Porém, têm muitas limitações comparando com os programas de sequenciação [21].

3.1.7. General MIDI

O *General MIDI* ou *GM* é uma especificação para sintetizadores, e um complemento à norma MIDI. Foi criado em 1991 pela MMA (*MIDI Manufacturers Association*) e pelo JMSC (*Japan MIDI Standards Comitee*), em que definiu que certos programas, tanto de instrumentos de *General MIDI* como de placas de som, contivessem configurações específicas instrumentais ou de efeitos de som.

Os instrumentos compatíveis com a especificação *GM* têm de possuir, no mínimo, os seguintes requisitos:

- Permitir no mínimo 24 notas de polifonia, incluindo 16 para melodia e acompanhamento e 8 para ritmo;
- Suportar simultaneamente 16 canais MIDI, em que o canal 10 é reservado para o ritmo;

Alguns fabricantes têm desenvolvido as suas próprias extensões de *General MIDI*, como a Roland Corporation, que criou a extensão GS, e a Yamaha, que inventou a extensão XG.

Em 2003, foi criada uma nova versão denominada *GM Level 2*, ou apenas *GM2*, que em relação à anterior adiciona mais controladores, altera a polifonia mínima para 32 notas, entre outras funções.

3.2. ANDROID

Atualmente, a plataforma de desenvolvimento Android é um dos sistemas operacionais para dispositivos móveis mais populares do mundo, sendo utilizado por milhões de pessoas.

O Android é um sistema operativo baseado no *kernel* do Linux, que é desenvolvido nos dias de hoje pela empresa multinacional Google. Inicialmente, foi desenvolvido pela Android Inc., fundada em 2003, tendo sido posteriormente adquirida pela Google em 2005.

Em novembro de 2007, a empresa norte-americana juntamente com a Open Handset Alliance, uma aliança dos maiores fabricantes de dispositivos móveis, como é o caso da Samsung, HTC, entre outras, anunciaram o Android como uma plataforma aberta para dispositivos móveis, que em outubro de 2008 foi implementada no primeiro *smartphone* disponível comercialmente, o HTC Dream.

O sistema destina-se principalmente a dispositivos móveis com *touchscreen*, como *smartphones* e *tablets*, contendo ambientes gráficos que são utilizados em conjunto com televisões – Android TV – e relógios de pulso – Android Wear. Uma das principais vantagens do Android é a sua compatibilidade com diversos dispositivos, com diferentes tamanhos de ecrã, dado que se trata de um sistema *open-source*.

3.2.1. ANDROID SDK

As aplicações Android são geralmente desenvolvidas, isto é, programadas em linguagem Java, utilizando o *kit* de ferramentas de desenvolvimento, o Android *Software Development Kit* (SDK). As ferramentas deste *kit* compilam o código e os arquivos de dados e recursos utilizados num pacote Android com o sufixo .apk. Os arquivos APK contêm todo o conteúdo de uma aplicação e são utilizados pelos dispositivos móveis para a instalação da mesma. O SDK também contém um *debugger*, bibliotecas e um emulador baseado em QEMU,

Oficialmente, o programa utilizado na criação de aplicações é o Android Studio, que é baseado no ambiente de desenvolvimento integrado (IDE) IntelliJ IDEA, criado em 2015. Até o final de 2014, os programadores Android utilizavam o programa Eclipse para desenvolver as suas aplicações. O Android Studio é suportado pelos sistemas operativos Windows, Linux e OS X.

3.2.2. COMPONENTES DE APLICAÇÃO

Nesta secção são descritos os componentes essenciais ao desenvolvimento de aplicações Android, sendo estes: as atividades; os serviços; os fornecedores de conteúdo; as *intents* e os *broadcast receivers*.

Atividades

Uma atividade é um componente de aplicação que representa um ecrã na interface de utilizador, onde são executadas certas ações por parte do utilizador, como por exemplo quando se tira uma foto na aplicação da câmara fotográfica.

Normalmente, associado a uma aplicação estão várias atividades, dependendo do número de ecrãs que esta necessita, que se encontram ligadas entre si, em que a atividade conhecida como sendo a principal é a apresentada ao utilizador na inicialização da aplicação.

Cada atividade pode iniciar uma outra através da passagem do componente de aplicação denominada *intent,* com o objetivo de realizar outro tipo de ações. Ao ser iniciada uma nova, a atividade anterior é interrompida, sendo notificada da alteração de estado através de métodos de retorno que identificam o ciclo de vida da atividade.

A vantagem da utilização destes métodos é possibilitar ao programador atribuir diferentes operações, dependendo do estado da atividade. A Figura 18 ilustra os diferentes métodos de retorno de uma atividade, representados por um retângulo [22]:

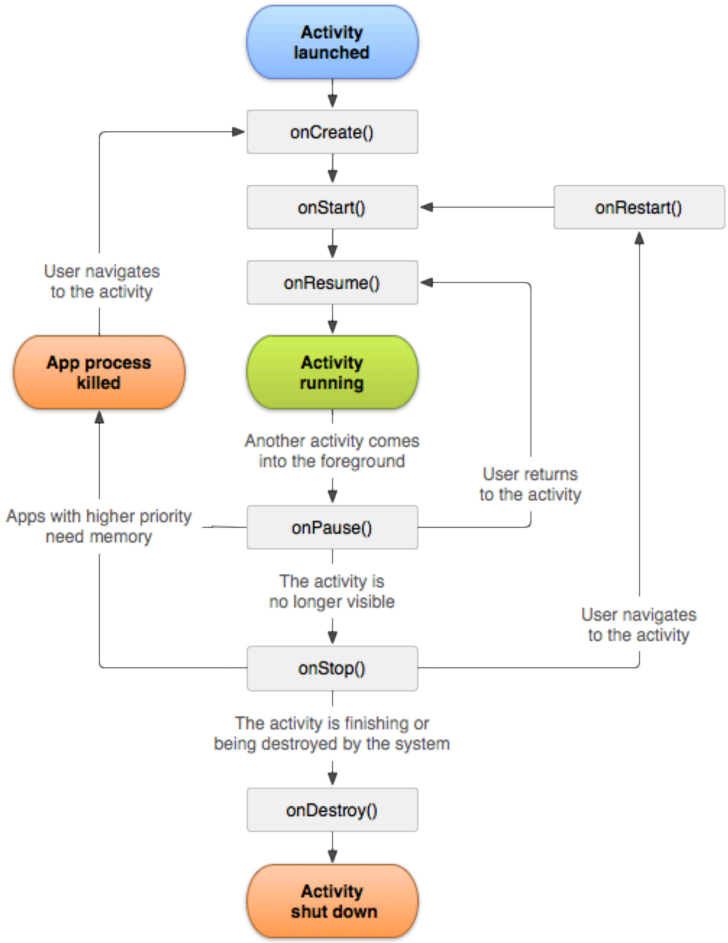


Figura 18 Ciclo de vida de uma atividade [22]

Através da Figura 18 é possível afirmar que uma atividade pode seguir diferentes caminhos entre os estados, existindo assim três *loops*, em que cada método é analisado na Tabela 8, assim como também indicados os próximos dentro da atividade.

- **Todo o tempo de vida** de uma atividade é definido entre o método *onCreate()*, que é o primeiro evento utilizado na atividade quando esta é criada, e o *onDestroy()*, que é recebido quando a atividade é destruída.

- O **tempo de vida visível** de uma atividade é determinado entre os métodos *onStart()* e *onStop()*, que durante este período de tempo o utilizador visualiza na aplicação o ecrã da atividade, interagindo assim com ela. O método *onStop()* é chamado quando é inicializada uma nova atividade.

- O **tempo de vida em primeiro plano** de uma atividade acontece entre a chamada dos métodos *onResume()* e *onPause()*. Como se encontra no interior do tempo de vida visível, o utilizador interage com a atividade dado que a sua interface é visualizada na aplicação. O método *onPause()* é chamado quando o sistema está prestes a retomar outra atividade.

Tabela 8 Métodos de retorno do ciclo de vida de uma atividade

Método	Próximo	Descrição
onCreate()	*onStart()*	Método chamado na criação da atividade pela primeira vez.
onRestart()	*onStart()*	Método chamado depois da atividade ser interrompida, quando esta é reiniciada.
onStart()	*onResume()* *onStop()*	Método chamado antes do utilizador visualizar a atividade. De seguida, ocorre o *onResume()* caso a atividade continue a correr em primeiro plano, ou o *onStop()* se ficar oculta.
onResume()	*onPause()*	Método chamado antes do utilizador interagir com a atividade.
onPause()	*onResume()* *onStop()*	Método chamado quando o sistema está prestes a retomar a outra atividade. É seguido do método *onResume()* caso o utilizador regresse a esta atividade, ou do *onStop()* se esta ficar invisível.
onStop()	*onRestart()* *onDestroy()*	Método chamado quando o utilizador deixa de visualizar a atividade. Caso o mesmo volte a interagir com a atividade, o método é seguido do *onRestart()*, caso contrário a atividade é destruída e é chamado o método *onDestroy()*.
onDestroy()	Não existe	Método chamado antes da atividade ser destruída, sendo esta a última vez que a atividade é chamada.

Serviços

Um serviço é um componente de aplicação utilizado principalmente em aplicações que necessitam de executar uma operação por um longo período de tempo. Este tipo de componente não fornece uma interface de utilizador como uma atividade, e consiste numa tarefa que está a ser realizada em segundo plano. Um exemplo de aplicação de um serviço é a reprodução de música enquanto o utilizador está numa aplicação diferente.

Normalmente, um serviço pode ser de dois tipos:

- Um serviço é **iniciado** quando uma atividade inicia-o através da função *startService()*. Ao ser criado fica em execução em segundo plano por tempo indefinido, mesmo que o componente que o iniciou seja destruído.

- Um serviço diz-se **ligado ou vinculado** quando um componente de aplicação chama a função *bindService()*. Este tipo de serviço é utilizado quando se pretende que atividades ou outros componentes interajam com o serviço, permanecendo em execução enquanto o componente de aplicação que o iniciou estiver ligado a ele.

Tal como as atividades, os serviços também possuem métodos de retorno que definem o ciclo de vida destes. À esquerda da Figura 19 é ilustrado o ciclo de vida quando o serviço é criado com a função *startService()*, e à direita é apresentado quando criado com a função *bindService()*:

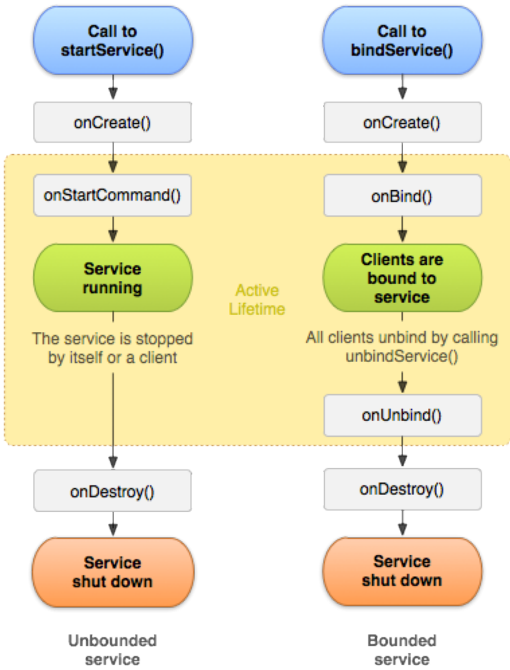


Figura 19 Ciclo de vida de um serviço [23]

Em cada um dos casos, o ciclo de vida do serviço consiste em dois *loops*:

- **Todo o tempo de vida** de um serviço é definido entre os métodos *onCreate()* e *onDestroy()*, que igual a uma atividade realizam as configurações iniciais no primeiro método referido, e enviam informação quando chamado o segundo método. Por exemplo, num serviço de reprodução musical é produzida música quando iniciado o mesmo, através do *onCreate()*, sendo interrompida no *onDestroy()*.

- O **ciclo de vida ativo** de um serviço acontece quando é chamado *onStartCommand()* ou *onBind()*. Caso se refira a um serviço iniciado – *startService()* – o seu ciclo de vida ativo terminará à mesma altura que o ciclo de vida inteiro do serviço, tendo como método de retorno o *onDestroy()*. Se for um serviço ligado, o ciclo de vida ativo termina quando for retornado o método *onUnbind()*.

Fornecedores de conteúdo

Este componente de aplicação, como o nome indica, fornece dados a outras aplicações ou atividades. O funcionamento dos fornecedores de conteúdo é receber solicitações por parte de clientes/aplicações, realizar a ação pedida para depois enviar os resultados a quem inquiriu. Um exemplo de utilização deste tipo de componentes são os contactos, tais como nomes, endereços e números de telefone, que podem ser acedidos por qualquer aplicação.

Intents

As *intents* são objetos que representam "intenções de fazer algo", utilizados no pedido de ações entre componentes de aplicação, funcionando como um mecanismo intermédio que recebe ou envia dados dentro ou fora de uma aplicação. Existem três formas de utilizar estes componentes:

- **Inicialização de uma atividade:** a *intent* é utilizada para alteração de ecrãs dentro da aplicação, ou seja, quando realizada alguma ação por parte do utilizador, este irá abrir uma nova atividade através do método *startActivity()*. Caso seja necessário receber dados da nova atividade, o programador utiliza a função *startActivityForResult()*, para receber em *onActivityResult()* o resultado da atividade quando for terminada esta.

- **Inicialização de um serviço:** a *intent* descreve o serviço que se pretende iniciar, assim como são carregados todos os dados necessários, sendo passada para a função *startService()* para que seja iniciado um serviço, que executa uma operação que ocorre somente uma vez.

- **Fornecimento de uma transmissão:** o sistema fornece várias transmissões, isto é, mensagens enviadas para qualquer aplicação, para eventos do sistema, como por exemplo, a inicialização de um dispositivo ou o seu carregamento.

Existem dois tipos de *intents*: as explícitas e as implícitas. As primeiras especificam qual o componente de aplicação que se pretende iniciar através do nome, e são utilizadas geralmente para abrir componentes dentro da própria aplicação, como é o caso da inicialização de uma nova atividade ou de um serviço.

Já as *intents* implícitas não nomeiam nenhum componente em específico, mas sim anunciam uma ação que irá ser executada no geral, permitindo que componentes de outras aplicações a processem. A Figura 20 ilustra como uma *intent* implícita é utilizada pelo sistema Android para iniciar uma nova atividade:

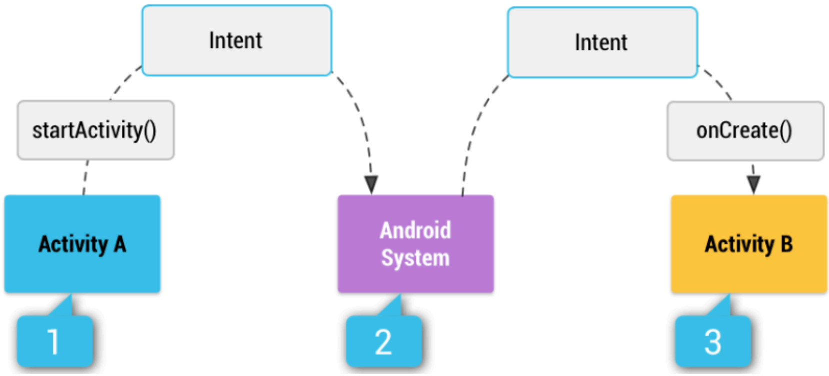


Figura 20 Exemplo de utilização de uma *intent* implícita [24]

Na Figura 20, a atividade A cria uma *intent* e passa-a para *startActivity()*. De seguida, o sistema Android procura em todas as aplicações um filtro de *intents* que corresponda à *intent*, que ao encontrar uma ligação, é iniciada a atividade pretendida (atividade B), através do método *onCreate()*, onde é passada a *intent.*

O filtro de *intents* especifica o tipo de *intents* que o componente de aplicação deseja receber, ou seja, é uma expressão implementada no Android *Manifest*, que irá ser explicado mais adiante, que garante a inicialização de uma atividade com um determinado tipo de *intent* através de outra aplicação.

Broadcast Receivers

Os *broadcast receivers* possibilitam a comunicação de eventos que não estão inseridos no funcionamento de uma aplicação. Assim, uma aplicação pode responder a *broadcasts* enviados por quaisquer componentes do sistema, como por exemplo a receção de mensagens de texto. A maioria dos *broadcasts* são originados pelo sistema Android, como é o caso do aviso de bateria fraca.

3.2.3. ANDROID *MANIFEST*

Todas as aplicações desenvolvidas para o sistema Android contêm um ficheiro denominado *Manifest*, que é caraterizado por expor informações sobre a aplicação ao sistema, sendo estas necessárias antes que o sistema execute o código da aplicação. Trata-se de um ficheiro XML que tem como funções principais as seguintes [25]:

- Atribuição do nome ao pacote Java da aplicação, servindo também como identificação para esta;
- Descrição dos componentes da aplicação tais como as atividades, serviços, fornecedores de conteúdo e *broadcast receivers* que esta é composta;
- Declaração de permissões que a aplicação necessita para aceder a áreas protegidas da API e interagir com outras, como é o caso do acesso à internet;
- Declaração do nível mínimo da API que a aplicação exige.

4. ARQUITETURA

Neste capítulo será descrita a arquitetura geral do sistema desenvolvido, com base no estudo efetuado, e serão fundamentadas todas as tomadas de decisão em relação à seleção do *hardware* do projeto, e descritos todos os *softwares* utilizados, tanto no desenvolvimento do código, como também na interpretação das mensagens MIDI.

A arquitetura do sistema é obrigatória fazer em qualquer projeto, dado que permite compreender o sistema de forma mais simplificada, ter uma noção geral da estrutura do mesmo, sendo um requisito para a implementação do projeto.

Tendo em conta o enunciado do problema, a solução encontrada é a criação de um sistema baseado num microcontrolador, que noutras palavras será o cérebro de todo o projeto, dado que irá realizar a leitura do estado dos botões, acionando os LEDs com as cores pretendidas.

O microcontrolador também é responsável pela a comunicação série, realizada a partir de módulos Bluetooth, tanto com um computador, para enviar os comandos MIDI, que serão interpretados por um *software*, como também com uma aplicação Android presente num *smartphone*, para visualização dos parâmetros do sistema. A Figura 21 representa o que foi dito num diagrama de blocos:

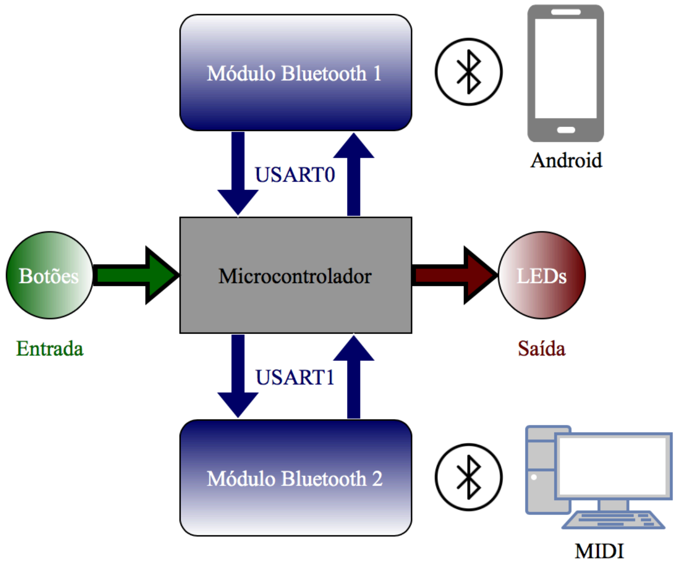


Figura 21 Arquitetura geral do sistema

4.1. FUNCIONAMENTO

O sistema desenvolvido consiste numa matriz física de 12x12, sendo então composto por 144 posições, onde estão presentes um botão e um LED em cada uma. Estes últimos acenderão conforme o compasso de tempo que percorrerá a estrutura de linha a linha com um tempo definido, e também à medida que os botões forem premidos, através da inserção de uma bola em cada local.

Quando a faixa de LEDs correspondente ao *tempo* imposto no sistema coincide com as posições onde as bolas se encontram inseridas, será produzida uma nota musical. Através da comunicação USART presente no microcontrolador, este envia para um módulo Bluetooth a mensagem MIDI *Note On,* que é interpretada por um *software* de reprodução, que interpreta os comandos do protocolo.

Também está conectado ao microcontrolador um segundo módulo Bluetooth, que irá interligar o sistema a uma aplicação desenvolvida para *smartphones* Android, dado que este possui duas USARTs. Esta aplicação tem como objetivo permitir ao utilizador visualizar a cor em todas as posições do sistema, assim como alterar as definições do mesmo.

4.2. *HARDWARE*

Neste subcapítulo é analisado o *hardware* implementado no projeto e explicado o motivo pelo qual foi selecionado, após ter sido feito o planeamento e estruturação do projeto.

4.2.1. MICROCONTROLADOR

Como referido, o responsável pelo controlo de todo sistema é o microcontrolador. Este é caraterizado por ser um circuito integrado que é constituído por periféricos tais como processador, memória e periféricos de entrada e saída, sendo assim um pequeno computador, ou por outras palavras, um *System On Chip*. Os microcontroladores têm aplicações em produtos comerciais, para que controlem as suas ações através do desenvolvimento de *software*.

Após terem sido avaliadas as opções existentes no mercado acerca dos microcontroladores, a escolha incidiu num ATmega1284P, fabricado pela Atmel. A Tabela 9 mostra as caraterísticas que este microcontrolador possui:

Tabela 9 Caraterísticas do ATmega1284P

Caraterísticas ATmega1284P	
Frequência de Operação	20 MHz
Tensões de Operação	1.8 - 5.5 V
Linhas I/O Programáveis	32 linhas
Flash	128 *KBytes*
EEPROM	4 *Kbytes*
SRAM	16 *Kbytes*
Periféricos	2 *Timers/Counters* (8 *bits*)
	2 *Timers/Counters* (16 *bits*)
	6 Canais PWM
	8 Canais ADC
	2 USART
	3 SPI
	I2C

A principal razão por se ter escolhido este microcontrolador, além da familiarização com a sua arquitetura, foi o facto de este possuir duas USARTs. Como descrito no funcionamento do projeto, na subsecção 4.1, estas são necessárias dado que o sistema comunica com:

- Um computador, onde está presente um *software* MIDI, que recebe comandos deste protocolo para produção de notas musicais.

- Um *smartphone* Android, que contém uma aplicação em que o utilizador visualiza e modifica certos aspetos do sistema.

4.2.2. LED

Tal como no projeto criado pelo artista Yuval Gerstein foram utilizados LEDs RGB, de forma a ser possível a reprodução de diversas cores em cada posição.

Devido ao sistema possuir doze linhas e doze colunas, totalizando cento e quarenta e quatro posições diferentes, não foi possível atribuir um LED a cada pino digital, dado que o microcontrolador possui apenas 32 pinos digitais. Para isso foi necessário encontrar uma solução que reduzisse esse mesmo número, tendo-se optado por LEDs encadeados por um único cabo para transmissão de dados. Estes denominam-se WS2812B, que também são conhecidos por NeoPixels, um produto comercial vendido pela Adafruit.

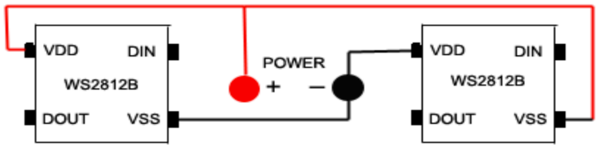
No geral, o componente é um LED de controlo inteligente, desenvolvido pela WorldSemi, que é constituído por um circuito de controlo e um *chip* RGB, que são integrados num pacote de componentes 5050, que indica a dimensão do *chip*, sendo neste caso 5x5mm.

Tem como vantagem principal poder ser interligado a outros, sem ter necessidade de utilizar mais pinos digitais do microcontrolador, sendo somente utilizado um que controlará todos os LEDs do sistema, sem recorrer ao uso de *multiplexers*. Porém, têm como desvantagem o seu protocolo de dados, que indica que a transmissão é executada de pixel em pixel. Isto faz com que caso um LED esteja danificado, os restantes não recebem o conjunto de dados relativo à cor.

Os chamados NeoPixels são baseados no WS2812, em que o circuito integrado foi melhorado a partir de arranjos mecânicos fora da estrutura interna, o que aumenta a estabilidade e eficiência dos componentes. Os seguintes pontos mencionam as melhorias mais significativas [26]:

- Foi adicionado um circuito de proteção inverso, que garante a proteção do produto, caso seja mal alimentado o circuito. A Figura 22 ilustra a representação gráfica do que foi mencionado:

a)



b)

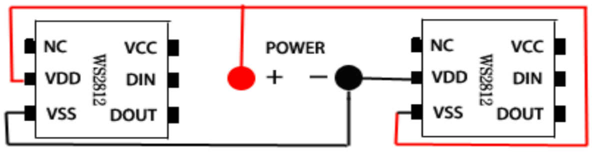


Figura 22 Circuito de proteção do WS2812B (a) e do WS2812 (b) [26]

- Manteve-se o tamanho, tendo sido reduzidos o número de pinos para quatro, aumentando assim o espaçamento entre eles. A Figura 23 compara as propriedades mecânicas do WS2812 e do WS2812B:

a) b)

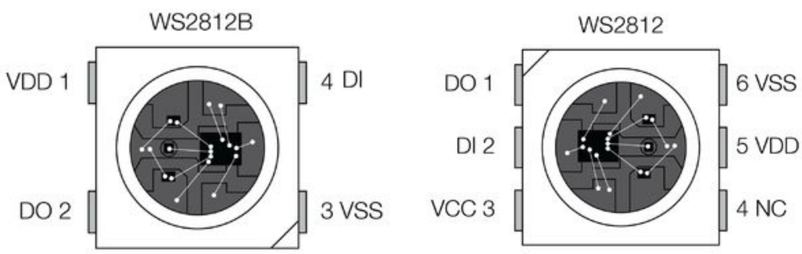


Figura 23 Propriedades mecânicas do WS2812B (a) e do WS2812 (b) [27]

- O *chip* RGB presente no WS2812B possui um maior brilho e uniformidade de cor que o do WS2812.

- Em termos de estrutura, o WS2812B é melhor, em que o circuito de controlo e o *chip* RGB foram separados, obtendo-se um melhor desempenho em relação à dissipação de calor.

O WS2812B é composto por quatro pinos, em que a função de cada um deles é descrita na Tabela 10:

Tabela 10 Função de cada pino do WS2812B [28]

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

A partir da Tabela 10 verifica-se que a alimentação dos WS2812B é efetuada a partir dos pinos VDD e VSS, entre 3.5 e 5.3V. O pino DIN irá recolher a informação relativa à cor do LED, isto é, receberá os dados do microcontrolador caso seja o primeiro emissor de luz do sistema. Caso contrário será do LED anterior através da conexão ao pino DOUT, que é caraterizado por ser a saída do sinal de dados de controlo, como ilustrado na Figura 24:

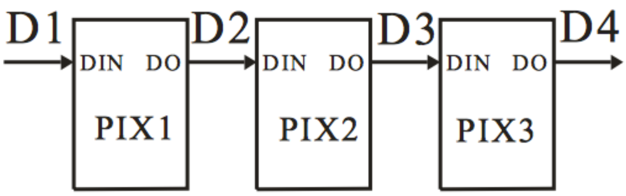


Figura 24 Ligação de três pixéis [28]

Na Figura 25 é demonstrado um circuito de aplicação típico utilizado na ligação de três WS2812B:

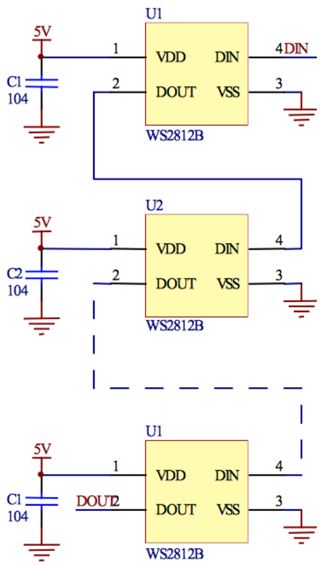


Figura 25 Circuito elétrico utilizado na conexão de três WS2812B [28]

A adição do condensador em cada LED, como visualizado na Figura 25, permite armazenar energia de maneira que a corrente que passa pelos pixéis não os danifique.

Para cada pixel (LED), os dados a serem enviados formam um conjunto de 24 bits, sendo divididos 8 bits para cada cor. Os primeiros oito são relativos à cor verde (G), depois à cor vermelha (R), e por último à cor azul (B), em que é sempre enviado o bit mais significativo em primeiro lugar, como demonstra a Figura 26:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Figura 26 Composição dos dados de 24 bits [28]

O protocolo de transferência de dados é bastante específico em relação ao tempo. Os valores lógicos '0' e '1' são enviados na forma de um pulso quadrado, com tempos definidos pelo fornecedor:

- O valor lógico '0' é enviado se o microcontrolador colocar a um o respetivo pino por 350ns, e de seguida a zero por 900ns.
- O valor lógico '1' é enviado se o microcontrolador colocar a um o respetivo pino por 900ns, e de seguida a zero por 350ns.

Portanto, para ser enviado um valor lógico é necessário aproximadamente 1.1 ou 1.4 µs (TH+TL = 1.25 µs ± 150 ns), o que significa que um byte irá demorar 10µs, e os três bytes de dados relativos à cor do LED irão demorar aproximadamente 30µs. Se durante 50µs não for recebido nenhum bit ocorre um *reset*, isto é, torna-se necessário voltar a transmitir informação desde o primeiro componente. A Figura 27 permite demonstrar o que foi referido em relação ao protocolo de transferência de dados:

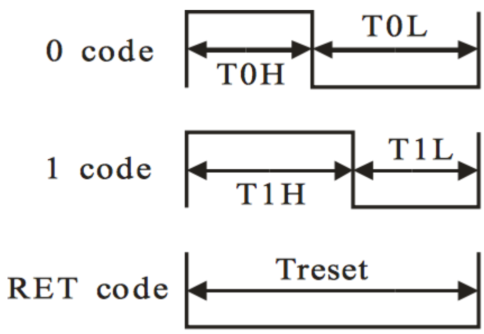


Figura 27 Gráfico de sequência dos valores lógicos e do *reset* [28]

Os tempos para transferência de dados dos parâmetros T0H, T0L, T1H, T1L e Treset foram obtidos do *datasheet* do componente, e são mostrados na Tabela 11:

Tabela 11 Tempos para transferência de dados [28]

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.9us	±150ns
T0L	0 code , low voltage time	0.9us	±150ns
T1L	1 code ,low voltage time	0.35us	±150ns
RES	low voltage time	Above 50µs	

Os LEDs foram adquiridos já soldados uns aos outros, o que possibilitou progredir mais rápido no desenvolvimento do projeto. Cada módulo é constituído por um WS2812B, um condensador e uma resistência, sendo necessário seis pinos para realizar o controlo e a alimentação de todos eles. A Figura 28 demonstra o esquema elétrico de cada módulo, como também a ligação entre três:

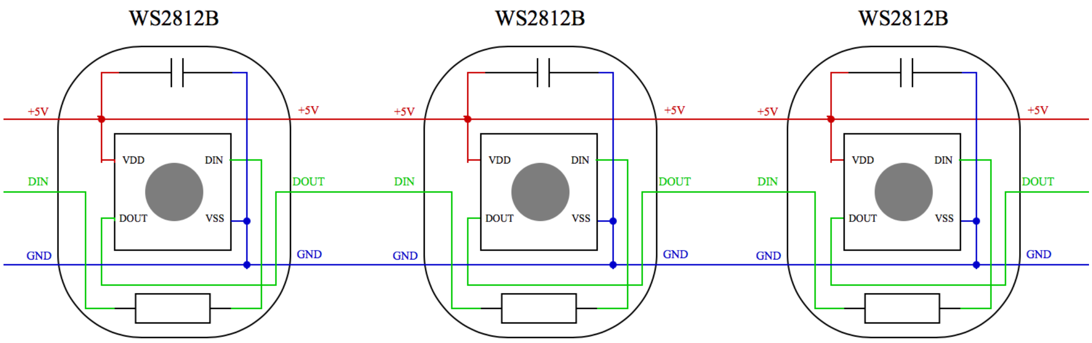


Figura 28 Esquema elétrico de ligação de três módulos

Cada LED consome no máximo 60 mA quando presente a cor branca, que significa que cada componente de cor tem presente o seu valor máximo. Portanto, por questões de segurança, o valor da corrente foi calculado para quatro linhas do sistema (48 LEDs), com o objetivo de todos os pixéis terem cor branca caso necessário:

$$48\ LEDs * 60\ mA = 2.88\ A \cong 3\ A$$

A corrente necessária para alimentar quatro linhas do sistema com a cor branca é calculada multiplicando o número de LEDs (48) pela corrente de cada um (60 mA), no que resulta um total aproximado de 3 A.

4.2.3. INTERRUPTOR

Um interruptor é um componente que controla a abertura ou fecho de um circuito elétrico, controlando o fluxo da corrente sem recorrer à alteração de ligações. Este elemento só pode adquirir um estado de cada vez, sendo aberto ou fechado.

A escolha dos interruptores foi realizada tendo em conta a facilidade de serem acionados quando inseridas as bolas nas posições da estrutura, que irão pressionar os LEDs.

Assim, optou-se pelos interruptores momentâneos (Figura 29 a)), ou por outras palavras botões de pressão, em relação aos interruptores fixos ou ON/OFF (Figura 29 b)), dado que os primeiros permanecem ativos apenas enquanto acionados, enquanto que os outros, utilizados nos interruptores que acendem e desligam a luz dos nossos quartos, mantêm o seu estado, havendo somente alteração quando atuados novamente.

a) b)

Figura 29 Interruptor Momentâneo (a), Interruptor Fixo (b) [29]

No projeto foram utilizados 144 botões devido a consistir numa matriz física de 12x12. A Figura 30 demonstra os botões empregados no projeto, que têm como exemplo de aplicação os ratos para computadores:

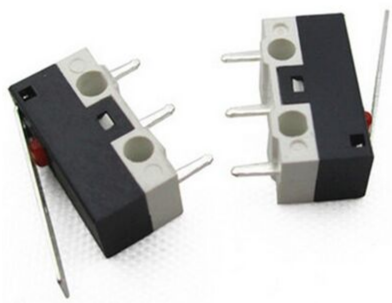


Figura 30 Botões utilizados no sistema

Como representado na Figura 30, estes botões são constituídos por três terminais, em que dois deles são descritos como normalmente aberto (NO) e normalmente fechado (NC), onde nunca podem estar os dois ligados ao mesmo tempo, e o outro por comum (C).

Os dois primeiros pinos caraterizam o interruptor momentâneo pelo seu estado normal, em que quando é utilizado o pino NO, significa que o botão está em aberto até ser acionado, estabelecendo contacto elétrico com o circuito. Por outro lado, quando é ligado o pino NC, indica que o circuito está fechado no seu estado normal, e ao premir o botão ocorre uma abertura no mesmo, não havendo fluxo de corrente. O pino comum é conectado ao outro lado do circuito, que, tipicamente, costuma ser ao VCC ou ao GND.

4.2.4. *SHIFT REGISTER*

A utilização de 144 botões obrigou a que fosse estudada uma solução, para ultrapassar a limitação do número de pinos digitais imposta pelo microcontrolador.

Numa primeira fase, foi posta a hipótese de utilizar um multiplexador, ou *multiplexer*, dado que se trata de um componente capaz de selecionar uma entrada entre várias. Porém, a sua implementação não foi possível dado que ultrapassava na mesma o número de pinos digitais do microcontrolador, apesar de haver uma redução face aos mesmos.

Assim, foi utilizado um *shift register* que permitiu implementar todos os botões, utilizando somente três pinos do microcontrolador.

Como associados aos componentes estão os botões, ou seja, as entradas do sistema, foram utilizados *shift registers* com configuração *Parallel-In Serial-Out* (PISO). Um exemplo deste tipo de elemento é o CD4021BE, tendo sido este o *shift register* utilizado no projeto. Este permite efetuar a leitura de oito entradas digitais ligadas ao componente de uma só vez.

A configuração dos pinos é demonstrada na Figura 31, e a descrição de cada um deles na Tabela 12:

Tabela 12 Função de cada pino do CD4021BE

Pino	Simbologia	Descrição
1, 4-7, 13-15	*P1-P8*	*Parallel Inputs*
2, 3, 12	*Q6, Q8, Q7*	*Serial Outputs*
8	*VSS*	*Ground*
9	*PARALLEL/SERIAL CONTROL*	*Latch*
10	*CLOCK*	*Clock*
11	*SERIAL IN*	*Serial Data Input*
16	*VDD*	*DC Supply Voltage*

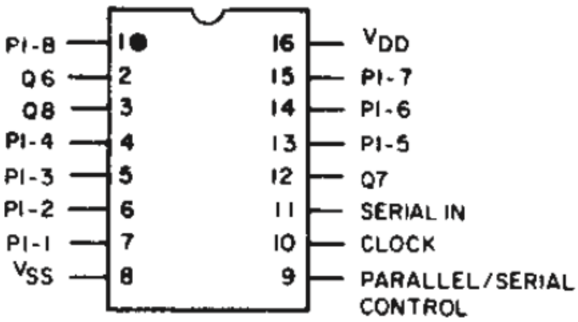


Figura 31 Configuração dos pinos do CD4021BE [30]

Como referido, apenas são utilizados três pinos do microcontrolador ao serem utilizados estes *shift registers*. Conforme o que foi descrito na Tabela 12, as oito entradas (botões) são ligadas aos pinos 1, 4, 5, 6, 7, 13, 14 e 15. O estado de cada uma delas é transmitido através dos valores lógicos '0' e '1', num pino de saída , normalmente Q8, estando este conectado a um pino digital do microcontrolador, configurado como entrada, sendo atribuído o nome de pino *data*.

A transmissão de informação relativa ao estado de cada botão é denominada de *"Synchronous Serial Output"*, devido ao componente enviar os dados sequencialmente para o microcontrolador, sempre que o mesmo os solicitar. Para isso, é necessário que o microcontrolador defina um pino digital como saída (pino *clock*), para ser ligado ao pino 10 do CD4021BE, sincronizando todo o processo entre os dois sistemas. Cada vez que o microcontrolador alterar o valor lógico do pino *clock* de '0' para '1', o *shift register* irá indicar no pino *data* o estado do próximo botão [30].

O terceiro pino associado ao microcontrolador é o pino 9 do CD4021BE, referido como pino *latch*. Quando este assume o valor '1' significa que o *shift register* está a determinar o estado das oito entradas. Caso contrário, isto é, quando o pino é definido com o valor lógico '0', transmite a informação guardada sobre cada botão, à medida que é feita a transição do estado '0' para '1' no pino *clock*.

Resumindo, os seguintes passos indicam como efetuar a leitura de cada uma das entradas conectadas ao CD4021BE:

1. Atualizar o *shift register* com a informação de cada entrada, ou seja, atribuir o valor lógico '1' ao pino *latch*.

2. Após determinação do estado de cada entrada, efetuar a sua leitura, colocando o pino *latch* a '0'.

3. Para cada entrada, alterar o valor lógico do pino *clock* de '0' para '1', com o intuito de verificar o estado de cada uma no pino *data*.

Como são implementados 144 botões no projeto, serão necessários vários *shift registers* para determinar o estado de todas as entradas. A equação seguinte indica o número de *shift registers* que serão utilizados:

$$Total = \frac{144\ botões}{8\ entradas/CD4021BE} = 18$$

Equação 1

Através do pino 11 (*Serial In*) é possível interligar diversos CD4021BE, permitindo que o primeiro receba e transmita para o microcontrolador, o estado de outras entradas conectadas a outros *shift registers*. A Figura 32 ilustra como é feita a conexão entre dois CD4021BE, assim como a ligação ao microcontrolador:

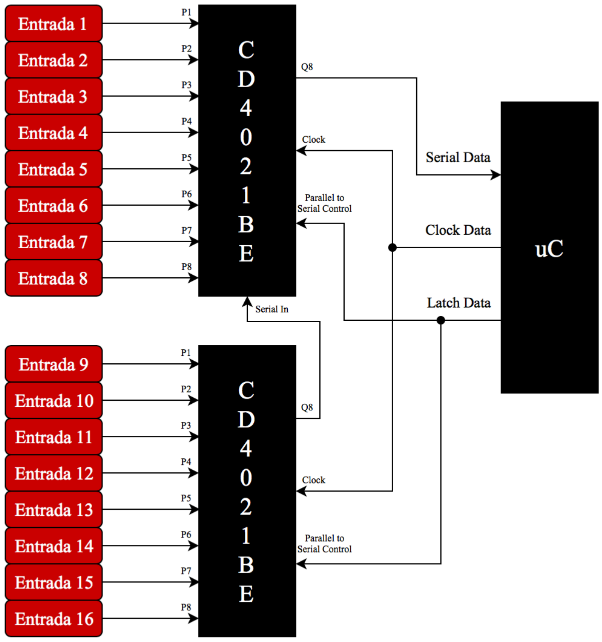


Figura 32 Ligação de dois CD4021BE ao microcontrolador

4.2.5. MÓDULO BLUETOOTH

O Bluetooth é um protocolo no envio e receção de dados num sistema sem fios. Este padrão é considerado uma rede PAN ou WPAN (*Wireless Personal Area Networks*), dado que possibilita a troca de informação entre dispositivos, tais como telemóveis, *notebooks*, computadores, impressoras, entre outros, sendo seguro e utilizado em aplicações de curto alcance, de baixo custo e de baixo consumo.

O protocolo opera a uma frequência de 2.4 GHz, onde o seu alcance máximo pode ser um, dez ou cem metros, dependendo da potência máxima, e a transmissão de dados é efetuada através de radiofrequência, possibilitando a deteção de dispositivos [31].

As redes Bluetooth, denominadas de *piconets* (Figura 33), utilizam o modelo *master/slave* com o intuito de controlar a sincronização entre dispositivos. Estas redes são compostas por um *master* que pode ser conectado até sete dispositivos *slave*, sendo estes conectados somente a um *master*. Como referido, o *master* controla toda a comunicação na rede, e envia informação para os *slaves*, assim como solicita dados dos mesmos, que só podem comunicar com o *master*, não podendo comunicar entre si.

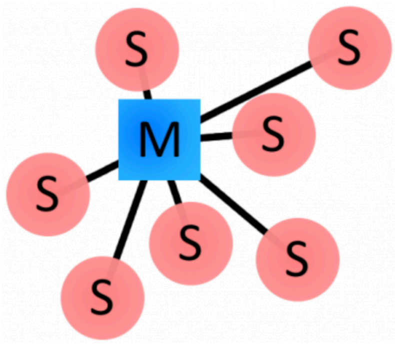


Figura 33 Rede *Piconet* [31]

Atualmente existem diversos módulos Bluetooth que têm a capacidade de realizar ligações ponto-a-ponto ou ponto-multiponto, embora seja difícil encontrar módulos que funcionem como *master* numa *piconet*, efetuando diversas ligações em simultâneo com os *slaves* da rede. Portanto, como solução optou-se por módulos de comunicação Bluetooth ponto-a-ponto, que em relação aos outros têm um custo inferior, sendo estes os módulos HC-05 e HC-06, que podem ser vistos na Figura 34:

a) b)

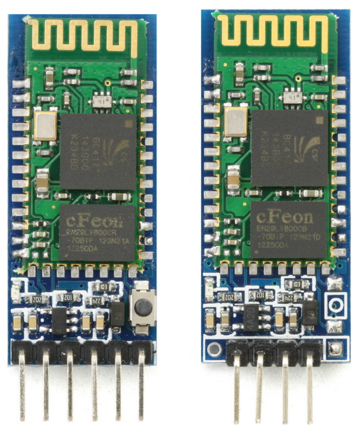


Figura 34 Módulos Bluetooth HC-05 (a) e HC-06 (b)

Tanto um módulo como outro têm caraterísticas semelhantes. A principal diferença entre eles é que o HC-05 permite comunicações tanto no modo *master* como no *slave*, assim como é constituído por seis terminais, enquanto que o HC-06 funciona somente como *slave*, e é composto por quatro pinos (VCC, GND, TXD e RXD). Estes terminais são descritos na Tabela 13:

Tabela 13 Função de cada pino dos módulos HC-05 e HC-06

Pino	Descrição
KEY	Senão estiver conectado, o módulo encontra-se em emparelhamento ou no modo de comunicação, caso contrário entra no modo AT.
VCC	O módulo é alimentado entre 3.3 e 6 V.
GND	*Ground*.
TXD	Transmissão de dados do módulo, conectado ao pino RX do microcontrolador.
RXD	Receção de dados no módulo, conectado ao pino TX do microcontrolador.
STATE	Ligado a um pino digital do microcontrolador, configurado como entrada, determina se foi estabelecida uma ligação ao módulo ou não.

Ambos os módulos efetuam ligações Bluetooth ponto-a-ponto e têm um alcance máximo de dez metros. Estes módulos podem ser configurados através de comandos AT. A Tabela 14 demonstra a configuração inicial para cada módulo:

Tabela 14 Configuração inicial dos módulos HC-05 e HC-06

Módulo	Nome do dispositivo	Palavra-Passe	*Baud Rate*	*Data*	*Stop Bits*
HC-05	HC-05	1234	9600 bps	8 *bits*	1 *bit*
HC-06	linvor				

É importante salientar que caso o módulo HC-05 seja colocado no modo AT, isto é, se o pino KEY estiver ligado ao VCC, o valor da taxa de transmissão (*Baud Rate*) é alterado para 38400 bps. A Tabela 15 descreve os principais comandos AT disponíveis para o módulo HC-05, em que no final terminam com um *enter* ou um “\r\n”:

Tabela 15 Comandos AT para o módulo HC-05

Comando	Descrição	Resposta
AT	Teste de comunicação	OK
AT+RESET	*Reset*	OK
AT+VERSION?	Indica a versão do software	+VERSION:<*version*> OK
AT+ORGL	Restaura configurações padrão	OK
AT+ADDR?	Indica o endereço do módulo	+ADDR:<address> OK
AT+NAME=<*name*>	Altera o nome do módulo	OK
AT+NAME?	Indica o nome do módulo	+NAME:<name> OK
AT+RNAME?<*rname*>	Indica o nome do dispositivo Bluetooth remoto	+RNAME:<*rname*> OK
AT+ROLE=<*mode*>	Altera o modo do módulo	OK
AT+ROLE?	Indica o modo do módulo	+ROLE<*mode*> OK
AT+PSWD=<*nnnn*>	Altera a palavra-passe do módulo	OK
AT+PSWD?	Indica a palavra-passe do módulo	+PSWD:<*nnnn*> OK
AT+UART=<*baud*>, <*stop*>,<*parity*>	Altera os parâmetros da comunicação série	OK
AT+UART?	Indica os parâmetros da comunicação série	+UART:<*baud*>, <*stop*>,<*parity*> OK

Os elementos *version* e *address* indicam ao utilizador qual o número da versão e o endereço Bluetooth do módulo, respetivamente. O parâmetro *mode* pode assumir o valor 0 (modo *slave*), 1 (modo *master*) ou 2 (*Slave-Loop*). Já as variáveis *baud, stop e parity* representam por ordem a taxa de transmissão, o número de *stop bits*, em que o valor 0 significa que é 1 bit e o valor 1 são 2 bits, e o bit de paridade (0 - nenhuma; 1 - ímpar; 2 - par). Os restantes parâmetros são alterados conforme escolha do utilizador.

A Tabela 16 representa os comandos AT utilizados para o HC-06, assim como a resposta a cada um destes comandos, que são caraterizados por serem *case-sensitive*:

Tabela 16 Comandos AT para o módulo HC-06

Comando	Descrição	Resposta
AT	Teste de comunicação	OK
AT+BAUD<*p*>	Alteração da *baud rate*	OK<*r*>
AT+NAME<*name*>	Alteração do nome	OK<*name*>
AT+PIN<*nnnn*>	Alteração da palavra-passe	OK<*nnnn*>

O parâmetro *p* define o valor do *baud rate* desejado e o *r* traduz esse valor em bps. Estes dois elementos são demonstrados na Tabela 17. Em relação aos restantes, o *name* indica o nome do dispositivo para o qual se quer alterar, tendo um total de vinte caracteres, e o *nnnn* define um código de quatro dígitos para ser utilizado no emparelhamento.

Tabela 17 Parâmetros do comando AT+BAUD e da sua resposta

<p>	1	2	3	4	5	6	7	8
<r>	1200	2400	4800	9600	19200	38400	57600	115200

4.3. *SOFTWARE*

Nesta secção serão analisados e explicados os *softwares* utilizados no desenvolvimento do sistema.

4.3.1. DESENVOLVIMENTO

Em microcontroladores AVR a melhor opção para compilação e desenvolvimento da sua programação passa pelo Atmel Studio, sendo um *software* gratuito, que é disponibilizado pela Atmel, a empresa que fabrica este tipo de microcontroladores.

Esta plataforma de desenvolvimento é de fácil utilização e fornece outras ferramentas que auxiliam o utilizador na criação dos seus projetos, tais como o compilador AVR-GCC, o *Assembler* e o programador por linha de comandos AVRDUDE, sendo ambos *open-source*.

Além destas funcionalidades, o *software* permite a simulação do programa desenvolvido através do modo *debug*, onde é possível a descoberta de erros no projeto antes da compilação e envio do código para o microcontrolador.

4.3.2. Integração do MIDI

Foi necessário emparelhar um módulo Bluetooth com o computador, para que as notas musicais correspondentes a cada posição pudessem ser transmitidas, através do envio de mensagens MIDI. Assim, foi utilizado um *software* denominado *Hairless MIDI to Serial Bridge*, que estabelece uma ligação, para envio e receção de dados convertidos para comandos MIDI, entre a porta de comunicação (módulo Bluetooth) e o *software* presente no computador utilizado para a reprodução musical.

Este é *open-source* e multiplataforma, isto é, o programa é disponibilizado gratuitamente para todos os sistemas operativos (Windows, Linux, OS X). A Figura 35 ilustra o programa em questão:

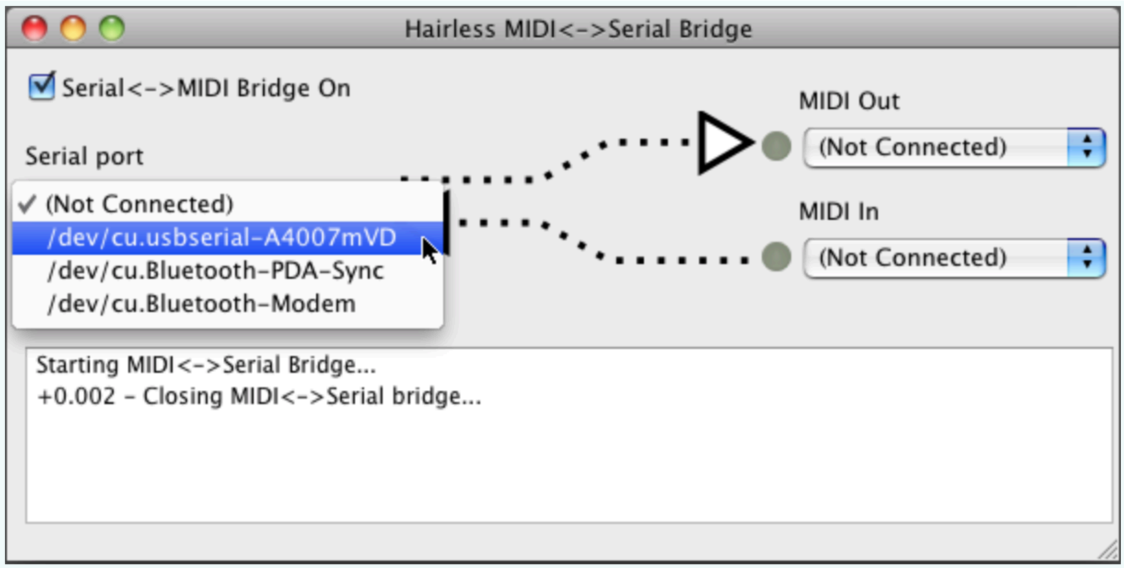


Figura 35 Ambiente gráfico do *Hairless MIDI to Serial Bridge* [32]

No início do programa, o utilizador seleciona a opção *Preferences* com o intuito de alterar a taxa de transmissão, que inicialmente está definida como 115200 bps, e outros parâmetros, conforme as suas definições.

No lado esquerdo do *software* escolhe-se a porta série que se pretende utilizar, referindo-se neste projeto ao módulo Bluetooth depois de emparelhado, enquanto que no lado direito o utilizador escolhe a aplicação que irá receber e/ou enviar dados MIDI.

Quando estabelecida a ligação, as mensagens MIDI são visualizadas no terminal situado abaixo na aplicação, caso a opção *Debug MIDI Messages* seja ativa.

Caso o utilizador trabalhe com o programa no sistema operativo Windows, necessita de criar um *driver* virtual para que os dados MIDI sejam reconhecidos pelo *software* de reprodução musical. Para tal foi utilizado o *loopMIDI* (Figura 36), dado que se trata de uma aplicação *open-source*, que irá criar portas MIDI virtuais com o objetivo de interligar o *software* que realiza a conversão dos comandos MIDI – *Hairless MIDI to Serial Bridge* – e o programa de reprodução musical que irá ser analisado em seguida – *SimpleSynth*.

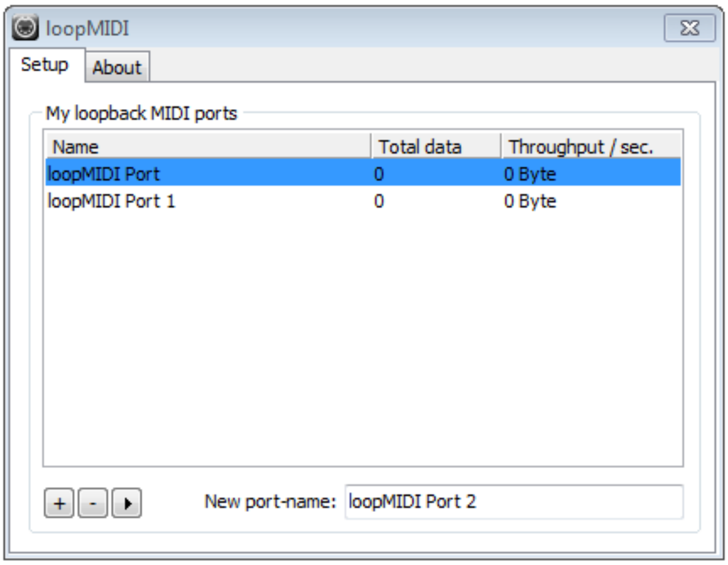


Figura 36 Ambiente gráfico do *loopMIDI* [33]

A criação de uma porta MIDI virtual é feita quando inserido um nome no parâmetro *New port-name*, e clicado em seguida o botão "+". Durante a sua execução a porta está ativa, sendo desativada quando o utilizador clica no botão direito do rato no ícone do *software*, e seleciona a opção *Stop loopMIDI*.

O programa que se utilizou para interpretação e execução dos eventos MIDI denomina-se *SimpleSytnh* (Figura 37), e é disponibilizado gratuitamente para a plataforma OS X.

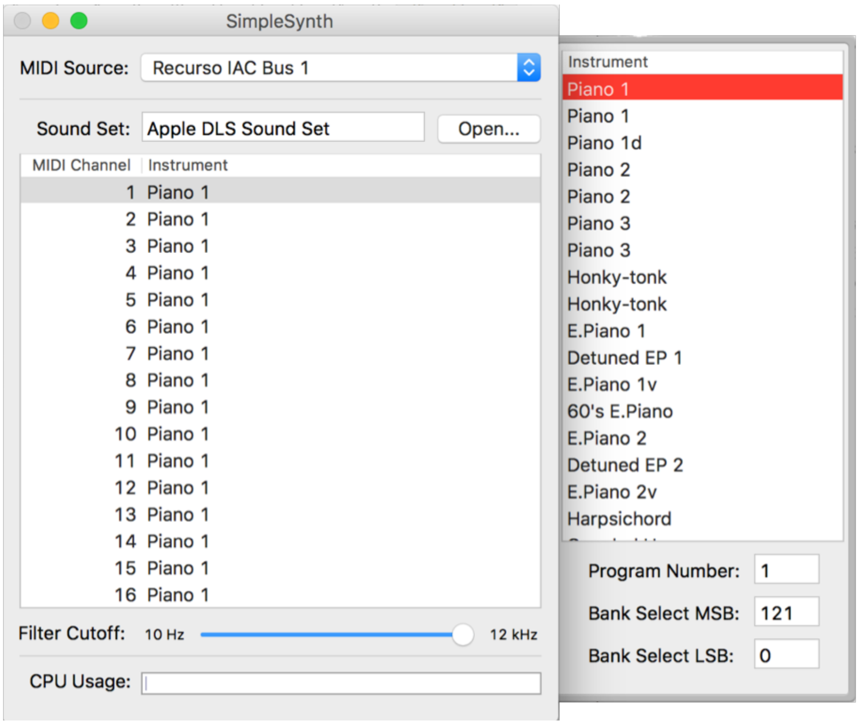


Figura 37 *SimpleSynth*

Em relação às funcionalidades do *software*, além dos comandos MIDI comuns tais como *NoteOn* e *NoteOff*, este permite selecionar o instrumento que irá tocar uma determinada nota musical, através de uma base de dados existente no programa.

4.3.3. ANDROID

A transferência de dados entre o sistema e o *smartphone* Android é realizada através de um módulo Bluetooth, ligado a uma USART do microcontrolador para comunicação série.

A aplicação foi desenvolvida no *Android Studio*, por ser o IDE oficial para o desenvolvimento de aplicações Android, sendo baseado no *IntelliJ IDEA*. Além das ferramentas utilizadas no *IntelliJ*, o *Android Studio* proporciona ainda mais recursos tais como [34]:

- Sistema de compilação flexível baseado no *Gradle*;
- Emulador rápido;
- Ferramenta *Instant Run* para alterar aplicações que estão a ser executadas, sem necessidade de compilar um novo APK;

- Ferramentas de verificação de código para detetar problemas de desempenho e compatibilidade de versões;

- Compatibilidade com C++ e NDK.

A janela principal do *Android Studio* (Figura 38) é dividida em seis áreas, descritas na Tabela 18. Esta pode ser organizada, ocultando ou movendo barras e janelas, de forma a obter mais espaço no programa.

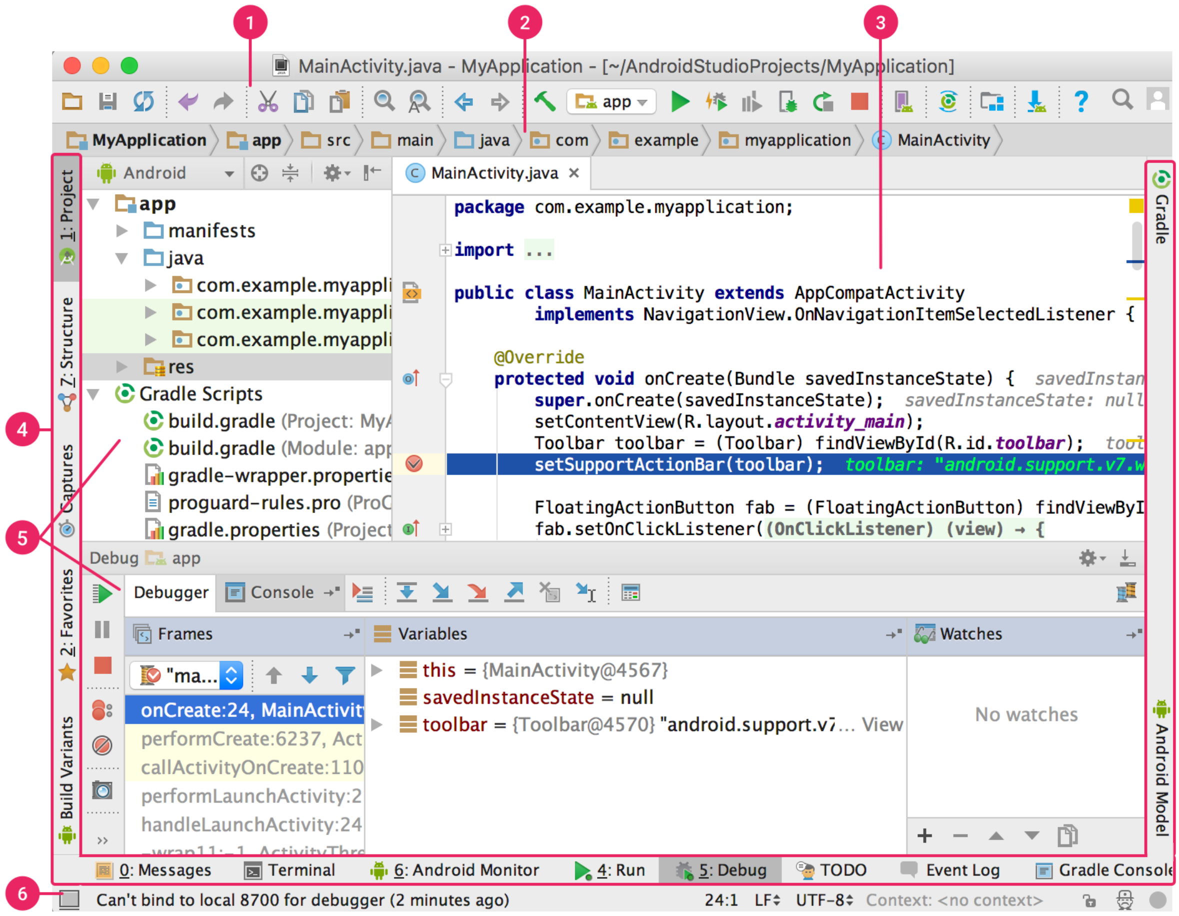


Figura 38 Janela principal do Android Studio [34]

Tabela 18 Descrição de cada área da janela principal

Área		Descrição
No.	**Nome**	
1	**Barra de Ferramentas**	Executa diversas ações, tais como executar aplicações e inicializar ferramentas.
2	**Barra de Navegação**	Auxilia na navegação pelo projeto e na abertura de ficheiros para edição.
3	**Janela do Editor**	Local utilizado para criar e editar, dependendo do ficheiro, que pode ser Java ou XML.
4	**Barra de Janela de Ferramentas**	Contém botões que expandem ou ocultam a janela de cada ferramenta.
5	**Janela de Ferramentas**	Permite aceder a determinadas tarefas, como gestão de projetos.
6	**Barra de Estado**	Demonstra o estado do projeto e do IDE, assim como advertências e mensagens.

4.3.4. EASYEDA

Para a criação dos esquemas elétricos e das placas de circuito impresso foi utilizado um *website* denominado *EasyEDA*. A primeira versão do *software* apareceu *online* em agosto de 2013 e tem como vantagem ser compatível e gratuita com vários *web browsers*, e oferecer ao utilizador várias ferramentas utilizadas em EDAs (*Electronic Design Automation*) famosos, como é o caso do *Altium Designer*, *Eagle* ou *Kicad*, permitindo assim a importação de projetos realizados nestes programas.

Para o projeto utilizou-se as ferramentas de desenvolvimento de circuitos elétricos e de PCB. Para além destas, o *EasyEDA* também permite que o utilizador efetue a simulação dos seus circuitos. A interface de utilizador (Figura 39) é constituída por dez campos [35]:

1. **Filtro**: pesquisa de ficheiros, projetos e componentes utilizados, através da inserção de texto.

2. **Painel de Navegação:** procura de projetos, ficheiros, componentes e seus *footprints*. O painel divide-se em seis áreas:

   a. **Projeto:** área de todos os projetos realizados, tanto privados como públicos.

b. **EELib:** bibliotecas do *EasyEDA* que fornece vários componentes com módulos de simulação, para facilitar a sua utilização em testes.

c. ***Design*:** verifica cada componente e ligação, e permite corrigir erros (DRC – *Design Rule Check*).

d. **Partes:** contém símbolos utilizados nos circuitos elétricos e *footprints* de PCB para diversos componentes acessíveis.

e. **Compartilhado:** aparecem os projetos privados que são partilhados somente com parceiros de trabalho.

f. **LCSC:** empresa pertencente ao *EasyEDA* que vende componentes eletrónicos, tendo já definidos os seus *footprints* para desenvolvimento de PCB, facilitando a implementação dos mesmos nos projetos.

3. **Barra de Ferramentas:** possui ícones que possibilitam ao utilizador realizar uma ação de forma mais rápida, como é o caso de apagar um elemento ou voltar atrás em alterações efetuadas num projeto.

4. **Diálogo de Pré-Visualização:** ajuda na identificação de esquemáticos e *layouts* de PCB.

5. **Ferramentas de Ligação:** dependendo do tipo de ficheiro, mostra diversas ferramentas utilizadas na ligação de componentes.

6. **Menu Gestor de Utilizadores:** permite ao utilizador alterar as suas preferências de conta, como é o caso da palavra-passe.

7. ***Super Menu*:** local onde todos os menus do *EasyEDA* podem ser encontrados, com o objetivo de aceder a cada um deles mais rapidamente, sem necessitar de procurar na barra de ferramentas.

8. **Ferramentas de Desenho:** possibilita adicionar elementos de desenho tais como caixas de texto, imagens, linhas, círculos, entre outros.

9. **Atributos da Área de Trabalho:** mostra as propriedades escolhidas para o local de trabalho, podendo ser configuradas, como é o caso da cor do fundo.

10. Área de Trabalho: local onde será criado e editado os circuitos elétricos, *layouts* PCB, símbolos e *footprints*, como também onde serão feitas simulações.

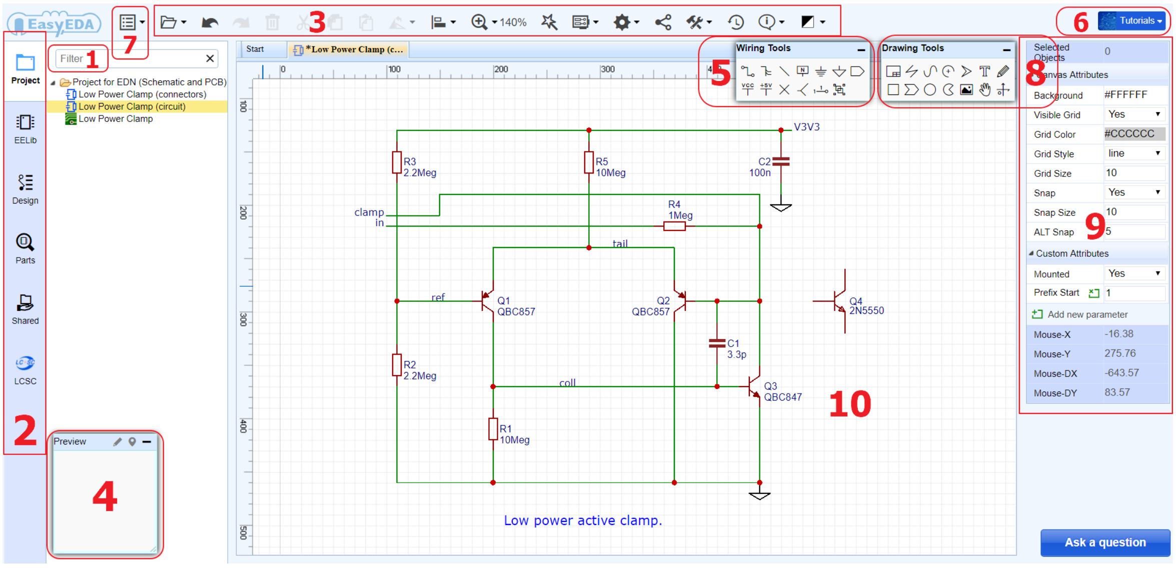


Figura 39 Interface de utilizador do EasyEDA (desenvolvimento de circuitos elétricos) [35]

5. IMPLEMENTAÇÃO

Através do estudo efetuado e após ter sido definida a arquitetura do projeto, começou-se o desenvolvimento do mesmo. Este capítulo começa por apresentar a estrutura, onde é explicada a sua construção. De seguida, é analisado cada esquema elétrico elaborado para o sistema, sendo justificadas todas as ligações, como também o desenvolvimento das placas de circuito impresso e seu resultado final com os componentes soldados. Numa última instância são justificadas as programações relativas ao microcontrolador.

5.1. ESTRUTURA

A estrutura construída consiste essencialmente numa tábua MDF (*Medium-Density Fiberboard*), um material derivado da madeira, onde foram realizados os buracos que servirão para controlar cada posição na parte superior da estrutura, em que as caraterísticas estão demonstradas na Tabela 19:

Tabela 19 Caraterísticas da estrutura

Dimensões			Distância entre posições	
Largura	Espessura	Comprimento	Horizontal	Vertical
1220 mm	19 mm	1495 mm	125 mm	100 mm

O sistema é constituído por doze linhas e doze colunas, o que significa que existem ao todo 144 posições na estrutura que podem ser controladas. Na Figura 40 é demonstrado um esboço da parte superior da estrutura em que as linhas são identificadas por números de 0 a 11 e as colunas pelas letras de A a L:

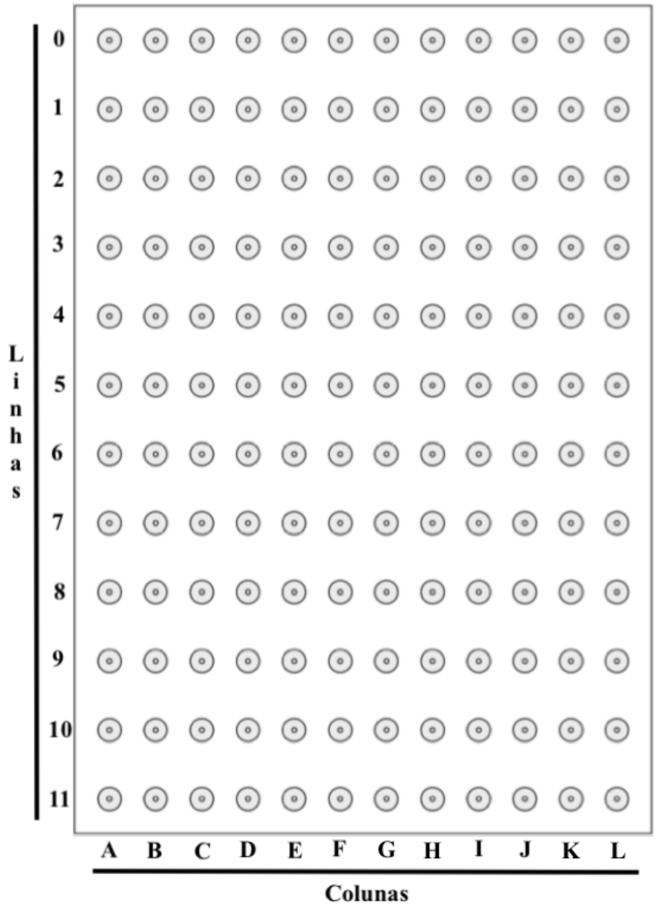


Figura 40 Esboço da parte superior da estrutura

Debaixo da tábua principal estão colocadas 24 placas de madeira. Cada uma tem as dimensões 725x30x19 mm (comprimento, largura, espessura), onde são inseridos seis botões.

Entre a tábua e as placas é introduzida a fita dos LEDs RGB em formato de serpente, de forma a não ser percorrida uma distância muito grande entre dois pixéis de linhas diferentes. Assim, o posicionamento de cada um varia consoante o tipo de linha – par ou ímpar – dado que o primeiro LED de cada uma é sempre diferente.

A parte inferior da estrutura que mostra o posicionamento das placas de madeira e de todos os LEDs é ilustrada na Figura 41:

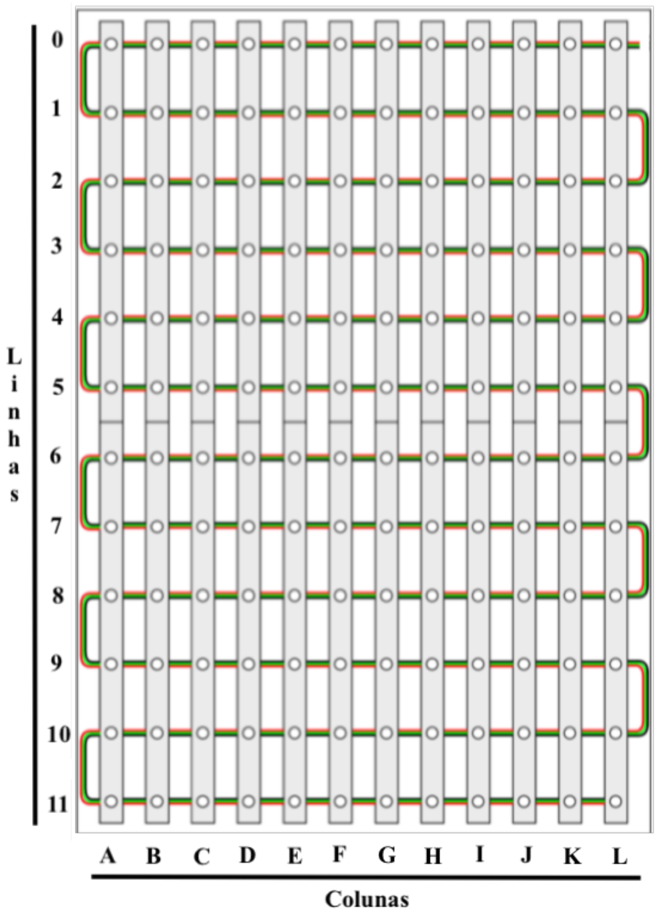


Figura 41 Esboço da parte inferior da estrutura

De seguida é explicado como é realizado o contacto das bolas com os interruptores, com a ajuda da Figura 42:

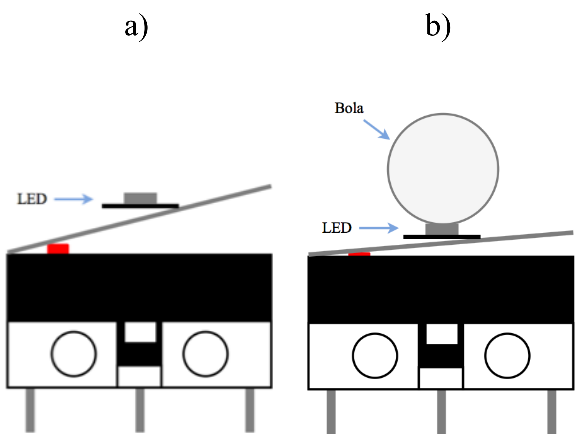


Figura 42 Contacto no interruptor sem bola (a) e com bola (b)

Na Figura 42 a) não é posicionada nenhuma bola, o que faz com que o botão mantenha o seu estado normal com o LED por cima. Quando inserida a bola na posição, o seu peso move o LED, que por sua vez faz com que a chapa metálica do interruptor se desloque para baixo, premindo assim o mesmo como pode ser visualizado em b).

5.2. ESQUEMA ELÉTRICO DO SISTEMA

Após serem explicadas as tomadas de decisão na construção da estrutura, é agora analisado cada esquema elétrico presente no sistema.

A Figura 43 demonstra como são organizados os componentes relativos ao *hardware*:

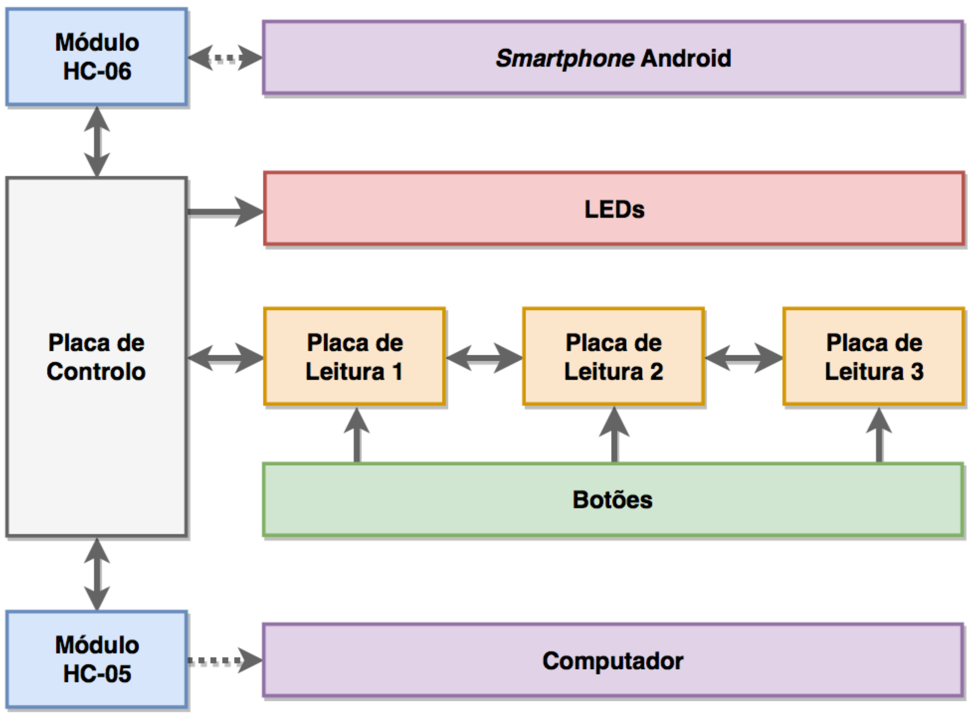


Figura 43 Diagrama de blocos relativo ao *hardware*

O principal elemento do diagrama de blocos da Figura 43 é denominado de Placa de Controlo. Esta é responsável pelo funcionamento de todo o sistema, dado que é onde se encontra o microcontrolador.

Os restantes blocos encontram-se conectados à Placa de Controlo, sendo eles os LEDs, as Placas de Leitura, que garantem a leitura de todos os botões, e os módulos HC-05 e HC-06, que efetuam a ligação por Bluetooth, tanto ao computador como ao *smartphone* Android.

O módulo HC-06 envia dados para uma aplicação instalada num *smartphone* Android, assim como também os recebe da mesma. Em relação ao outro módulo Bluetooth (HC-05), este somente envia informação para o computador.

5.2.1. Controlo

Nesta subsecção são explicadas todas as conexões da placa de controlo, tendo como componente principal o microcontrolador ATmega1284P, que é o cérebro de todo o sistema.

O microcontrolador funciona a uma frequência de 16 MHz devido à adição de um cristal. Inicialmente o ATmega1284P veio com a frequência de oscilação de 1 MHz e com o *clock* dividido por 8, o que obrigou à configuração dos *fuse bits* para que este trabalhasse a 16 MHz. Esta alteração foi efetuada devido aos tempos exigidos pelo protocolo de transferência de dados dos LEDs WS2812B serem demasiado curtos.

Está presente um circuito de *reset* que reinicia o microcontrolador, não sendo necessário desligá-lo da alimentação e voltar a ligá-lo. O circuito é constituído por uma resistência de *pull-up* de 10 kΩ (R7) e um botão (S1), dado que o *reset* só é ativado quando o pino associado a esta função receber o valor 0. Assim, o botão encontra-se ligado à massa, para quando for acionado enviar o respetivo valor de tensão, para efetuar o *reset* do sistema.

A Figura 44 ilustra a ligação do microcontrolador ao circuito de *reset* e ao cristal:

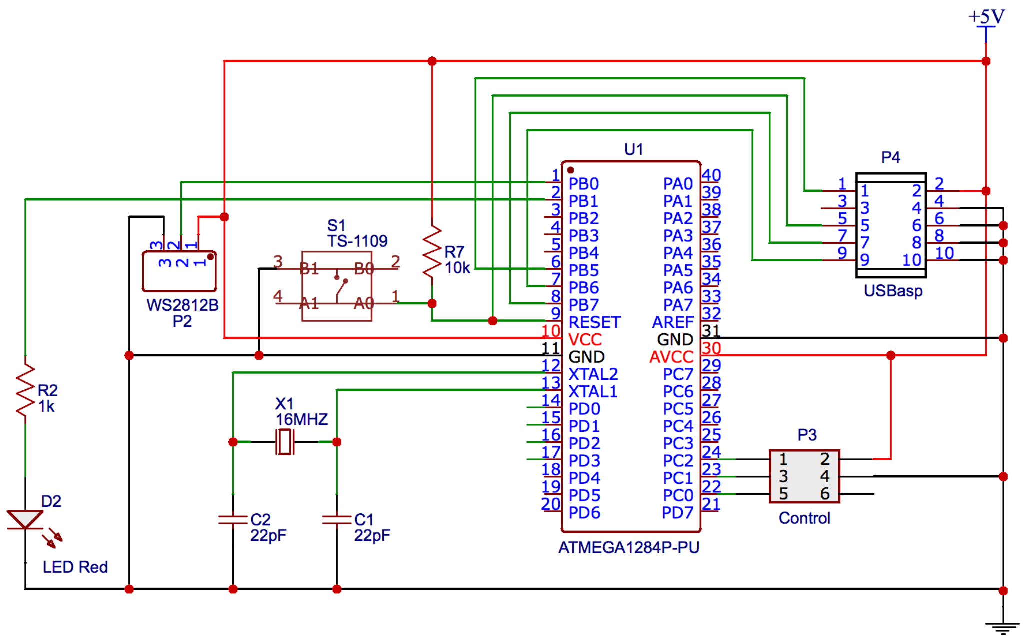


Figura 44 Esquema elétrico do circuito de controlo

Além da conexão do microcontrolador ao cristal (X1) e ao circuito de *reset*, a Figura 44 demonstra a ligação ao LED com prefixo D2, aos WS2812B (conetor P2), à primeira placa de leitura (conetor P3) e ao programador USBasp (conetor P4).

O sistema tem presente um LED de cor vermelha (D2) ligado ao pino PB1, denominado LED de funcionamento, que tem como função demonstrar o bom funcionamento do microcontrolador. A transmissão de cor para os LEDs RGB (WS2812B) é realizada através do pino PB0 do microcontrolador, que se encontra ligado ao pino 2 do conetor P2, correspondendo este ao pino de entrada DIN do primeiro pixel.

O conetor P3 é responsável pela ligação à Placa de Leitura 1. Este encontra-se ligado aos pinos 0, 1 e 2 do porto C do microcontrolador, com o intuito de efetuarem a leitura de todos os botões presentes no sistema. Os pinos 2 e 4 do conetor P3 alimentam as placas de leitura com uma tensão de 5 V.

Também é interligado ao microcontrolador o conetor P4 que tem como objetivo programar o mesmo através do USBasp, que é o programador utilizado neste projeto (Figura 45). Na ligação a este existem quatro pinos conectados à massa e um aos 5 V. Os restantes são de programação e estão devidamente ligados a pinos do microcontrolador, sendo estes MOSI (*Master Output/Slave Input*), RST (*Reset*), SCK (*Serial Clock*) e MISO (*Master Input/Slave Output*).

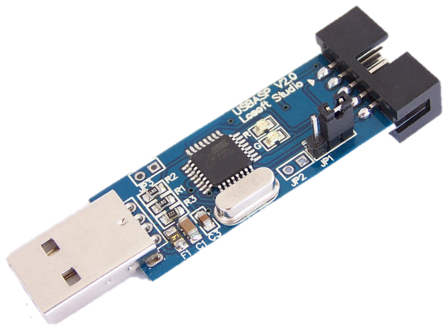


Figura 45 Programador USBasp

Tanto com o computador para produção musical, como com o *smartphone* Android para monitorização e controlo do sistema, a comunicação é efetuada através da ligação de módulos Bluetooth às duas USARTs do microcontrolador, sendo demonstrada na Figura 46:

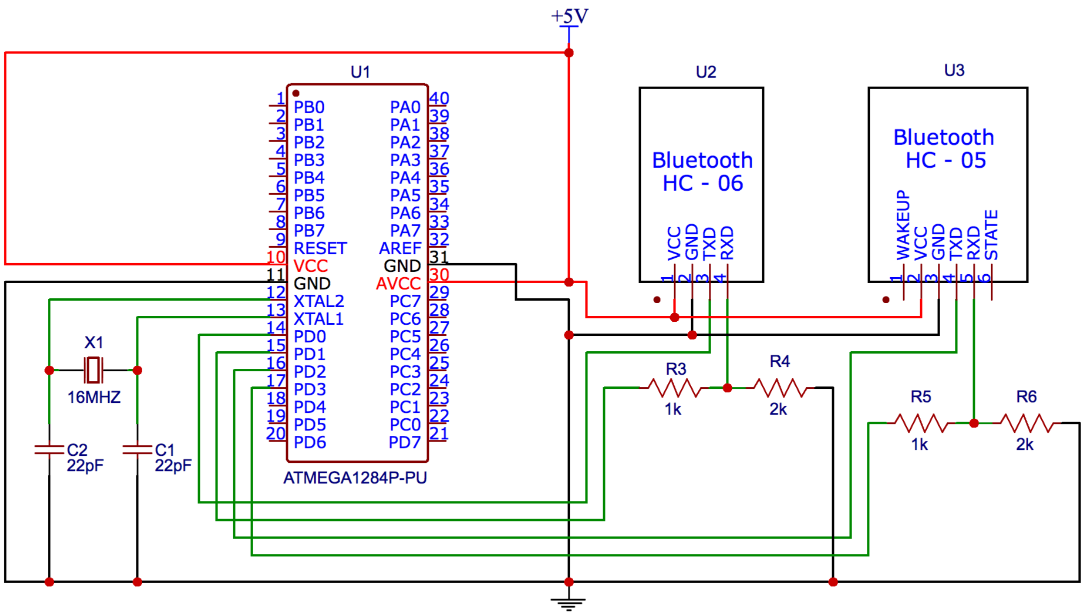


Figura 46 Esquema elétrico de ligação do microcontrolador aos módulos Bluetooth

Como visualizado na Figura 46, no pino RXD de cada módulo Bluetooth encontra-se implementado um divisor de tensão, que efetua a conversão de uma tensão de entrada de 5 V para 3.3 V, como ilustrado na Figura 47. Isto acontece dado que os pinos de comunicação do módulo trabalham entre 2.7 e 3.3 V.

Caso não fosse estabelecido o divisor de tensão, o envio de sinais do microcontrolador para os módulos com valores superiores à gama recomendada poderia danificá-los, dificultando a comunicação com o microcontrolador.

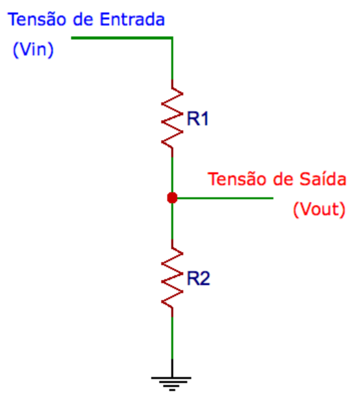


Figura 47 Divisor de tensão

A Equação 2 demonstra como é efetuado o cálculo da tensão de saída:

$$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in}$$

Equação 2

Considerando $V_{in} = 5$ V, $R_1 = 1$ kΩ e $V_{out} = 3.3$ V, obtém-se aproximadamente o valor 2 kΩ para a resistência R_2. Dado que não é um valor tipicamente utilizado nas resistências arredonda-se para 2.2 kΩ.

Juntamente com o circuito de controlo foi implementado um regulador de tensão (LM7805), em que a função de cada um dos seus pinos pode ser visualizada na Figura 48, que tem como função limitar a tensão à entrada para um valor máximo de 5 V.

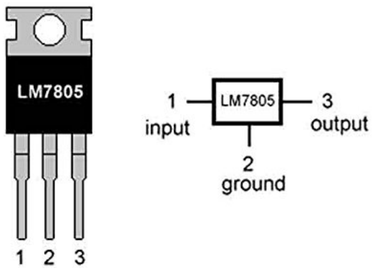


Figura 48 Função de cada pino do LM7805 [36]

Assim, na Figura 49 está representado o circuito de alimentação, que contém um LED sinalizador de alimentação (o componente acende quando o circuito está a ser alimentado) e um *switch* para ligar ou desligar a alimentação.

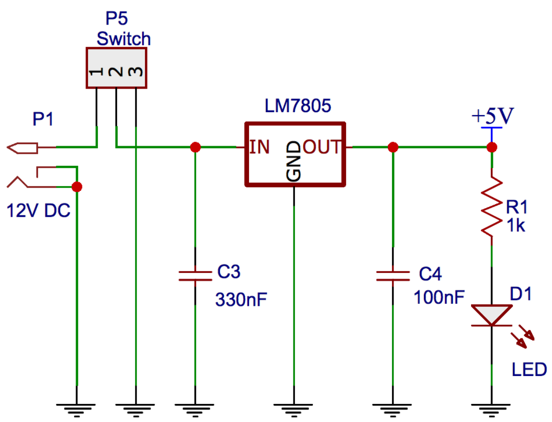


Figura 49 Esquema do circuito de alimentação

O esquema elétrico completo da Placa de Controlo pode ser analisado no Anexo A.

5.2.2. LEITURA

Esta parte do relatório serve para explicar as ligações elétricas presentes em cada Placa de Leitura, referidas na Figura 43. Estas são implementadas para leitura do estado de todos os botões do sistema, através da utilização de *shift registers*. Assim, foram desenvolvidas três destas placas, com o mesmo circuito elétrico, em que cada uma efetua a leitura de quatro linhas do sistema (48 botões).

No circuito de leitura estão presentes conetores de entrada e de saída, que interligam as três placas, assim como realizam a sua conexão com a placa de controlo. Em ambos os conetores, cada pino tem a mesma função que os do conetor P3 do circuito de controlo (Figura 44). Assim, todas as placas de leitura são alimentadas com a tensão de 5 V fornecida pelo circuito de alimentação presente na placa de controlo (Figura 49).

A conexão entre o microcontrolador e todos os *shift registers* é efetuada pela ligação entre a Placa de Controlo e a Placa de Leitura 1, como visualizado na Figura 50:

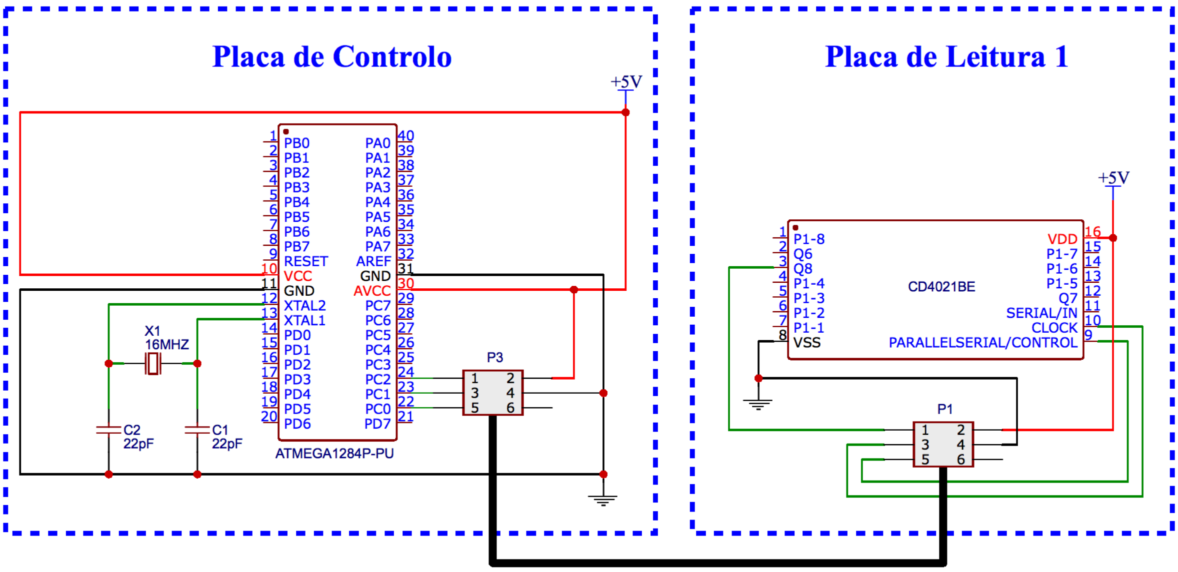


Figura 50 Ligação da Placa de Controlo à Placa de Leitura 1

Cada placa de leitura realiza a leitura de 48 botões. Estas encontram-se interligadas através do conetor P2 de uma e o conetor P1 de outra, que efetuam a conexão entre o último *shift register* da primeira placa de leitura e o primeiro da segunda. Esta é realizada através dos pinos *Serial/In* e *Q8*, como é visualizado na Figura 51:

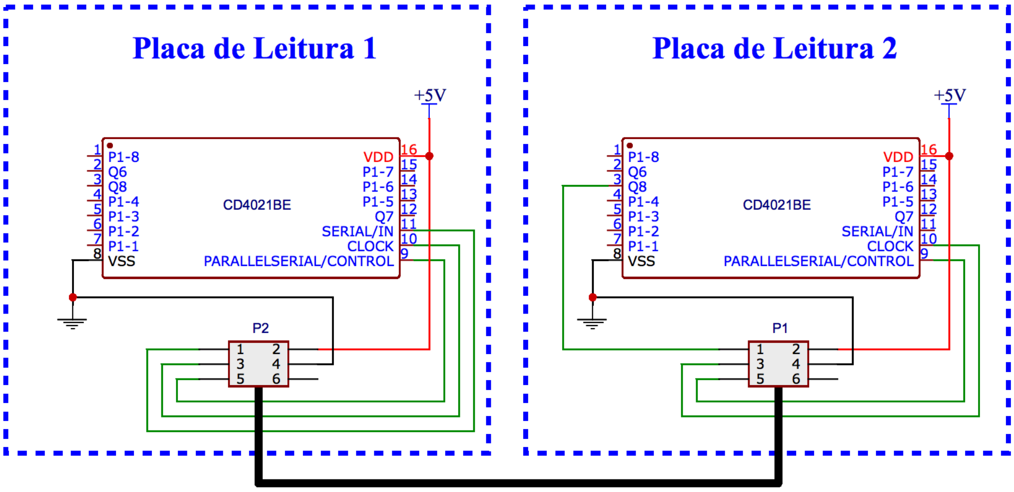


Figura 51 Ligação entre duas placas de leitura

Os pinos 9 e 10 de cada *shift register* encontram-se ligados, respetivamente, aos pinos PC0 e PC1 do microcontrolador, sendo denominados *latch* e *clock*. Em relação ao *latch*, tem como função indicar a cada *shift register* que os dados de todas as entradas podem ser transmitidos em série, ou que cada uma delas está a ser lida. O pino *clock* possibilita a leitura de todas as entradas e a sua transmissão para o microcontrolador pelo pino de dados (PC2).

Justificadas as ligações da Placa de Controlo à Placa de Leitura 1 (Figura 50), assim como entre placas de leitura (Figura 51), é de seguida explicado como é realizada a conexão dos botões com as placas de leitura. O esquema elétrico presente na Figura 52 representa a ligação de oito entradas a um *shift register*:

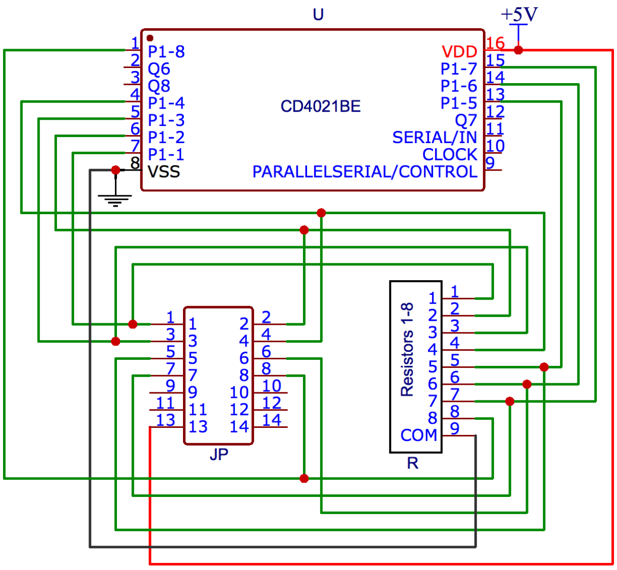


Figura 52 Esquema elétrico de um *shift register*

Na Figura 52, está inserido um conetor de 14 pinos (identificado pelo prefixo JP), que permite ligar doze botões, como a alimentação comum a estes, sendo esta os 5 V. Destes, somente oito são ligados ao *shift register* dado que o componente só permite efetuar a leitura de oito entradas.

Em cada *shift register* temos presente uma rede de resistências, nomeada por R. A utilização desta em vez de individuais residiu-se no facto de haver várias resistências, tendo-se, portanto, optado pela primeira opção que engloba oito resistências. Todas possuem o mesmo valor de 10 kΩ e um terminal comum, sendo este a massa.

Assim, a ligação de um pino de entrada (P1-1 até P1-8) é visualizada na Figura 53, para melhor compreensão do que está a ser feito através da rede de resistências:

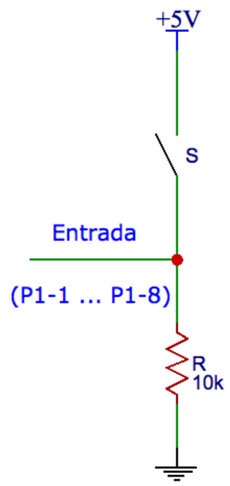


Figura 53 Circuito de ligação de um botão a um *shift register*

Em cada pino de entrada do *shift register* (P1-1 até P1-8) é inserida uma resistência de *pull-down*, para que quando acionado o interruptor, o microcontrolador receba o valor lógico 1 para tomada de decisões.

Todo o esquema elétrico relativo a uma placa de leitura pode ser visualizado no Anexo B.

5.3. Desenvolvimento de PCB

Nesta secção são analisadas as placas de circuito impresso desenvolvidas para o projeto, que foram criadas a partir dos esquemas elétricos demonstrados no Anexo A e no Anexo B.

Na Figura 54 é visualizada a placa de circuito impresso relativa ao esquema elétrico que possui o microcontrolador (Anexo A). Como visualizado na Figura 54, foram empregados componentes SMD (*Surface-Mount Device*) nas resistências, nos LEDs e em alguns condensadores, o que significa que foram soldados na superfície. Em relação aos outros, são posicionados através de terminais inseridos em furos realizados na placa, denominando-se *Through-Hole Devices* (THD).

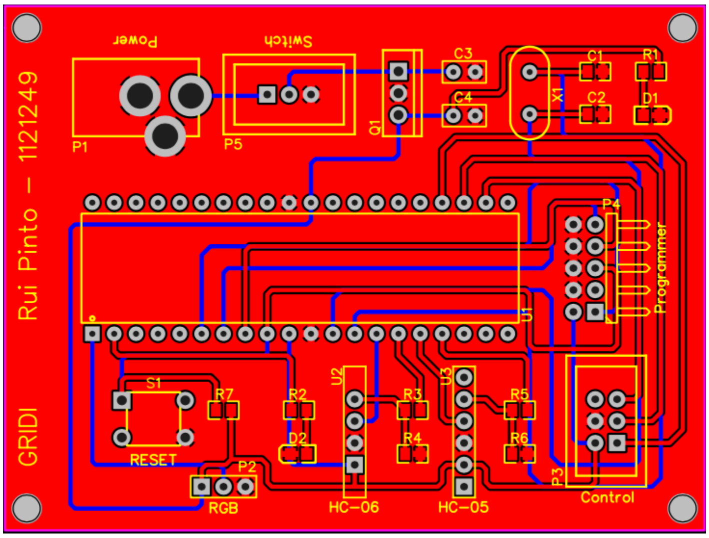


Figura 54 PCB relativa ao esquema elétrico do Anexo A

As duas faces da placa de circuito impresso foram utilizadas para criação das pistas. A cor vermelha corresponde ao desenho de pistas na camada superior, enquanto que a azul se refere às ligações na camada inferior.

Em relação aos *footprints*, estes são definidos pela cor amarela, que indica que os componentes têm de ser posicionados na face superior no processo de soldagem. Foi ainda adicionada uma área de cobre na camada superior que conecta todas as ligações à massa (GND), sendo por isso representada na figura com a cor vermelha.

Cada pista da placa tem uma largura de 0.5 mm e em relação à área de cobre tem um espaçamento de 0.25 mm.

O resultado final da placa de controlo, que é obtido após todos os componentes serem soldados, é demonstrado na Figura 55:

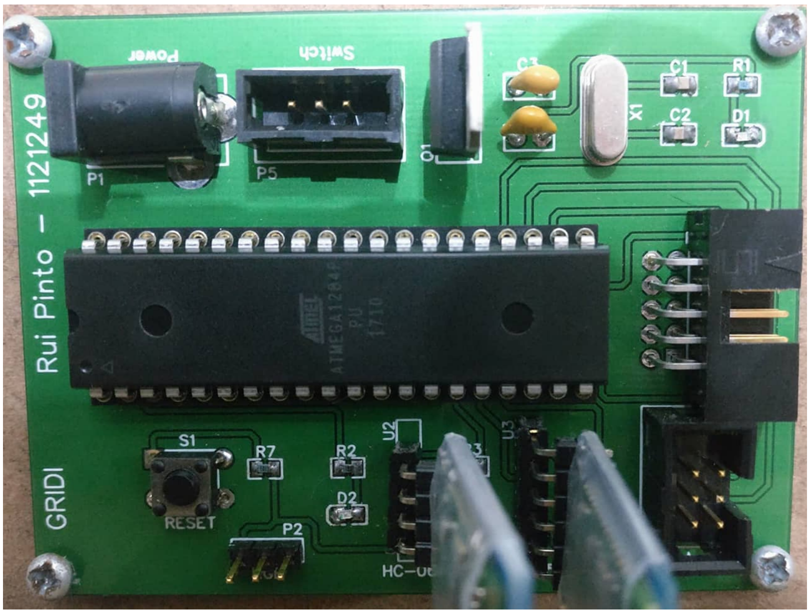


Figura 55 Placa de Controlo

No Anexo B está representado o esquemático completo da ligação dos *shift registers* aos interruptores, com a utilização de resistências *pull-down*. A PCB desenvolvida é dividida em duas faces: superior (Figura 56) e inferior (Figura 57).

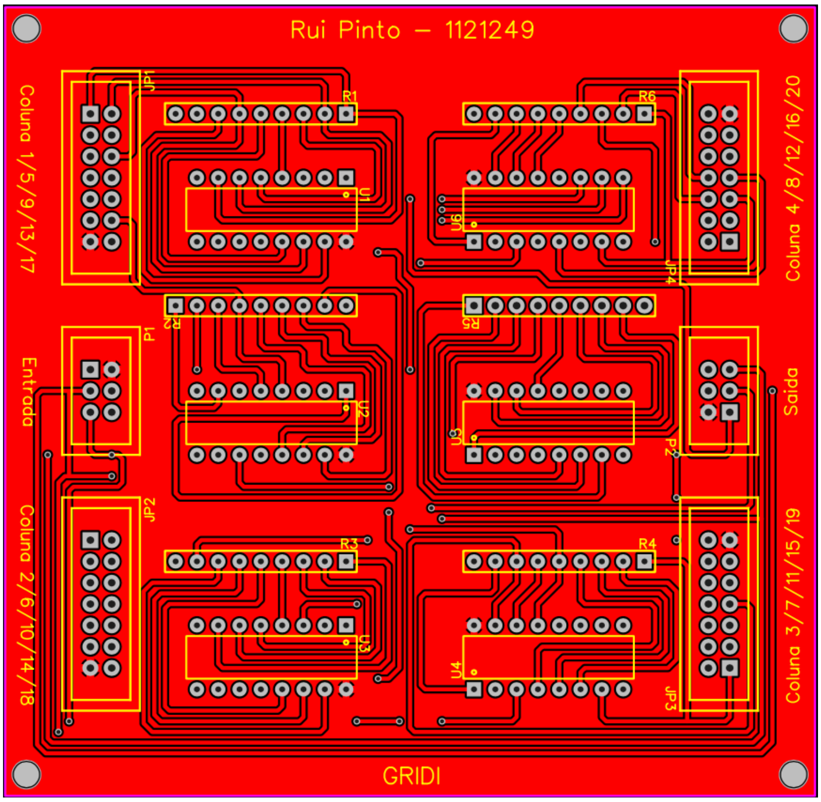


Figura 56 Face superior da PCB relativa ao esquema elétrico do Anexo B

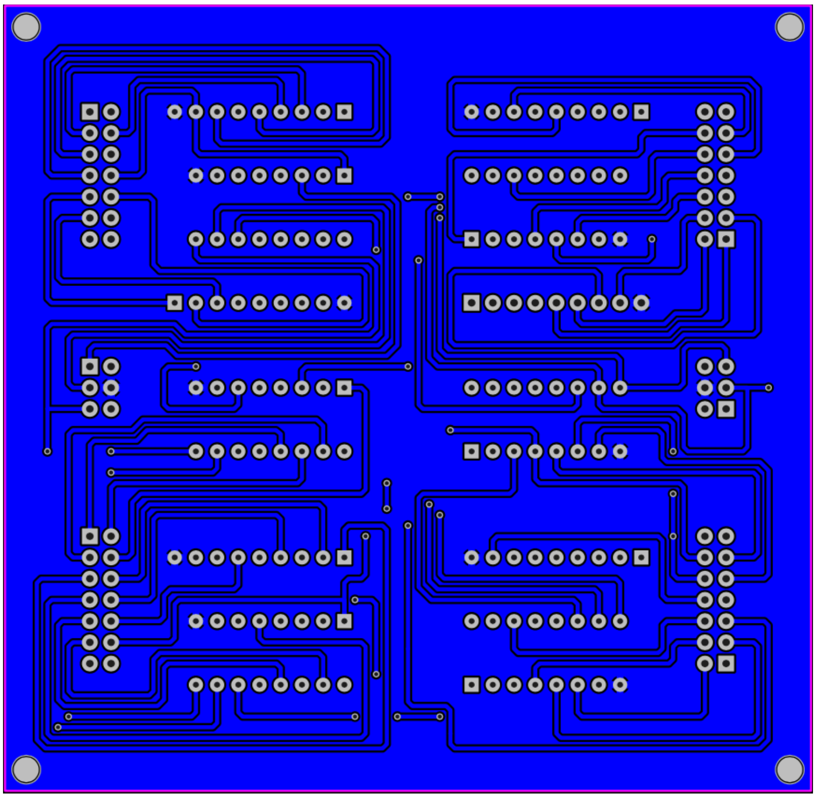


Figura 57 Face inferior da PCB relativa ao esquema elétrico do Anexo B

Devido ao número de ligações ser superior ao da PCB da Placa de Controlo (Figura 54), foi necessário a inserção de vias em algumas ligações, com um diâmetro de 0.5 mm, que permitem ligar pistas entre a camada superior (cor vermelha) e a inferior (cor azul).

Os *footprints* são identificados pela cor amarela, ou seja, todos os elementos são inseridos na face superior da placa, sendo soldados no outro lado da placa, dado que apenas são utilizados componentes THD.

Tanto a camada superior como a inferior são compostas por uma área de cobre, que possibilitou a redução do número de pistas na placa. A área de cobre superior realiza a conexão de todos os pinos ligados à alimentação dos +5V, enquanto que a inferior liga todos os pinos da massa (GND). Ambas as áreas têm um espaçamento de 0.25 mm em relação às pistas, que foram desenhadas com 0.5 mm de largura.

A Figura 58 mostra como ficou a placa de circuito impresso após todos os componentes terem sido soldados:

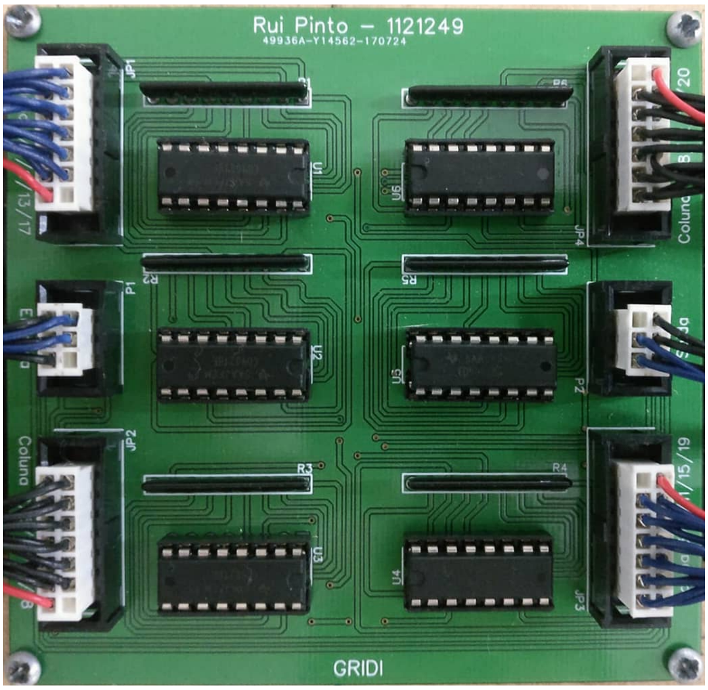


Figura 58 Placa de Leitura

Como referido, cada placa de leitura estabelece a ligação aos botões presentes em quatro linhas do sistema, o que equivale a 48 ao todo. Assim, criou-se três placas da Figura 58, de forma a obter-se o estado de todos os *switches*.

5.4. ARQUITETURA DO *SOFTWARE*

Após ser implementado o *hardware* definido para o projeto e terem sido criadas as placas de circuito impresso, procedeu-se para a programação do sistema, isto é, à escrita do código a ser compilado. Assim, o programa foi escrito na linguagem de programação C, para ser inserido no microcontrolador através do programador USBasp visualizado na Figura 45.

O *software* desenvolvido é demonstrado no fluxograma da Figura 59:

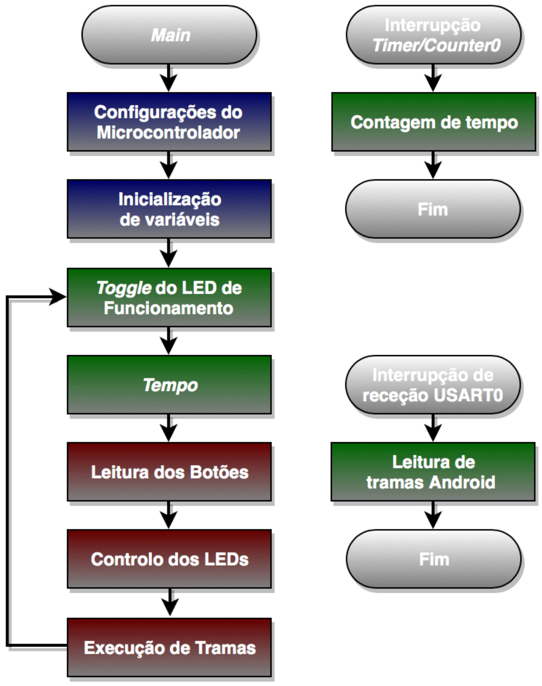


Figura 59 Fluxograma do funcionamento geral do sistema

O *software* inserido no microcontrolador consiste essencialmente no processo principal, designado por *Main*, e pelas interrupções do *Timer/Counter0* e da receção na USART0.

O processo *Main* executa um ciclo infinito após terem sido efetuadas todas as configurações e inicializações de variáveis do microcontrolador. Este ciclo começa por realizar a mudança de estado do LED de funcionamento e a transição do compasso de tempo, que acontecem em conformidade com a interrupção do *Timer/Counter0*. Nesta é feita uma contagem de tempo, com o intuito de verificar quando estes dois processos são executados.

De seguida, o microcontrolador efetua a leitura do estado de todos os botões do sistema, que fazem com que os LEDs sejam ligados ou desligados.

No processo *Tempo*, dependendo do estado de cada posição presente na linha ativa do compasso, são enviados comandos MIDI para o computador, para que sejam produzidas as notas musicais corretas.

O ciclo infinito do processo principal termina com a execução das tramas enviadas pela aplicação Android, que possibilitam a alteração de parâmetros do sistema. A leitura de uma trama é realizada na interrupção de receção na USART0.

Antes de serem analisados e explicados todos os processos é necessário ter-se em conta que o sistema possui dois modos de funcionamento:

- **Modo com bolas** – utilizador interage com a estrutura através do posicionamento de bolas transparentes em cada uma das posições.

- **Modo sem bolas** – as bolas são substituídas pela própria mão do utilizador, que utiliza os dedos para pressionar cada posição com o objetivo de realizar uma mudança de estado.

É importante também salientar que a primeira linha e coluna do sistema são identificadas pelo valor zero na programação e não por um, sendo, portanto, a última linha e coluna o número onze.

5.4.1. CONFIGURAÇÕES DO MICROCONTROLADOR

Normalmente, um *software* quando é desenvolvido para ser implementado num microcontrolador começa por configurar os periféricos do mesmo e definir as variáveis iniciais, para depois entrar num ciclo infinito.

A primeira configuração que foi necessária efetuar no ATmega1284P foi os pinos dos respetivos portos, que se encontram ligados ao LED de funcionamento, aos LEDs WS2812B e aos *shift registers*.

A cada porto estão associados três endereços de memória: o registo de dados (PORT), o registo de direção dos dados (DDR) e os pinos de entrada do porto (PIN), em que os dois primeiros são de leitura e escrita e o último apenas de leitura.

Para a configuração de cada pino é utilizado então o DDR, que é configurado a 1 caso se pretenda uma saída de dados, ou a 0 para receber dados de entrada.

O excerto de código seguinte mostra a configuração dos pinos referidos, onde *DDR_LEDs* corresponde ao DDRB; *DDR_Botoes* a DDRC; *BIT_WS2812B* e *BIT_Latch* a 0; *BIT_LED* e *BIT_Clock* a 1; e *BIT_Data* a 2:

```
DDR_LEDs = (1 << BIT_WS2812B) | (1 << BIT_LED);
DDR_Button = (1 << BIT_Latch) | (1 << BIT_Clock)
& ~(1 << BIT_Data);
```

Enquanto que o pino de dados (*BIT_Data*) é configurado como entrada pois recebe o estado de cada um dos botões, os pinos *latch* e *clock* são definidos como saídas para controlar os *shift registers*, de modo a efetuar as leituras, como é dito na subsecção 4.2.4 do relatório.

Após configuração dos portos configurou-se o *Timer/Counter0* com o modo CTC (*Clear Timer On Compare*) e uma temporização de 10 ms, de forma a obter os tempos pretendidos. Neste modo é incrementado o valor do contador TCNT0 até atingir o máximo definido no registo OCR0A, gerando uma interrupção no final e recomeçando o contador. O valor de comparação (OCR0A) é calculado através da Equação 3:

$$T = \frac{N}{f_{CLK\_I/O}} * (OCR0A + 1) <=> OCR0A = \frac{T * f_{CLK\_I/O}}{N} - 1$$

Equação 3

Com o microcontrolador a funcionar a uma frequência de 16 MHz, escolhendo um valor de *prescaler* de 1024 e sabendo que a base de tempo é 10 ms, obtemos no registo OCR0A o valor 156 aproximadamente.

A configuração do *Timer/Counter0* foi realizada através da manipulação de quatro registos, descritos no Anexo C. Depois de terem sido analisados procedeu-se ao desenvolvimento do código que configurasse o *Timer/Counter0* do microcontrolador. Para isso, primeiramente, teve-se em conta a Tabela 20 para seleção do seu modo, e a Tabela 21 para seleção do seu *prescaler*:

Tabela 20 Descrição dos *bits* que definem o modo do *Timer/Counter0* [37]

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP
0	0	0	0	Normal	0xFF
1	0	0	1	PWM, Phase Correct	0xFF
2	0	1	0	CTC	OCRA
3	0	1	1	Fast PWM	0xFF
4	1	0	0	Reserved	-
5	1	0	1	PWM, Phase Correct	OCRA
6	1	1	0	Reserved	-
7	1	1	1	Fast PWM	OCRA

Tabela 21 Descrição dos *bits* que selecionam o *prescaler* do *Timer/Counter0* [37]

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Assim, as linhas de código seguintes descrevem como se configurou o *Timer/Counter0* através da seleção efetuada com um retângulo vermelho na Tabela 20 e Tabela 21:

```
#define PRESCALER_TC0 1024
#define TEMP_S_TC0 0.010
#define OCR0A_Value (((F_CPU*TEMP_S_TC0)/PRESCALER_TC0)-1)
TCCR0A = (1 << WGM01);
TCCR0B = (1 << CS02) | (1 <<CS00);
OCR0A = OCR0A_Value;
TIMSK0 |= (1 << OCIE0A);
```

De seguida configurou-se a USART. Como mencionado, o microcontrolador é constituído por dois periféricos deste tipo sendo bastante importantes, dado que um realiza a comunicação com o computador que recebe os comandos MIDI, e o outro com a aplicação Android que recebe parâmetros do sistema para monitorização, assim como também transmite tramas com o intuito de modificá-los.

Assim, configurou-se a USART0 e a USART1 de forma igual, isto é, ambas têm o mesmo número de *bits* de dados, o mesmo número de *stop bits*, a mesma paridade, como também a mesma taxa de transmissão. Também, ambos são definidos para trabalhar no modo de funcionamento assíncrono.

Inicialmente, definiu-se uma taxa de transmissão de 115200 bps, que significa que são transmitidos 115200 *bits* em cada segundo. A explicação para a escolha deste valor é que o processo de transmissão de dados necessita de ser executado de forma rápida, para que não haja atrasos consideráveis no *software* desenvolvido.

De acordo com a Tabela 22, assinalado com um retângulo vermelho, o valor selecionado como taxa de transmissão torna-se aceitável, pois para a frequência de 16 MHz apresenta um erro de 2.1% ou -3.5%, consoante o valor do *bit* U2X:

Tabela 22 Valores associados a cada *Baud Rate* para a frequência de 16 MHz [37]

Baud Rate [bps]	f_{osc} = 16.0000MHz			
	U2X = 0		U2X = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%

Como é visível na Tabela 22, com o valor da taxa de transmissão sabe-se o valor do *USART Baud Rate Register* ou UBRRn, sendo a letra *n* 0 ou 1 consoante a USART utilizada. Existem três modos de operação na USART em que o escolhido foi o assíncrono de dupla velocidade. Este modo permite obter uma taxa de erro inferior ao modo normal.

A Tabela 23 indica o modo de operação selecionado, assim como as equações que foram utilizadas para a configuração:

Tabela 23 Equações para o cálculo da *Baud Rate* ou do UBRRn [37]

Operating Mode	Equation for Calculating Baud Rate(1)	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2X = 0)	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{16(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{16\mathrm{BAUD}} - 1$
Asynchronous Double Speed mode (U2X = 1)	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{8(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{8\mathrm{BAUD}} - 1$
Synchronous Master mode	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{2(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{2\mathrm{BAUD}} - 1$

Depois de analisados todos os registos necessários para se configurar as duas USART, explicados no Anexo D, desenvolveu-se o seguinte excerto de código:

```
1 | #define BAUD_USART0 115200
2 | #define UBRR0_Value ((F_CPU/(8UL*BAUD_USART0))-1)
3 | #define BAUD_USART1 115200
4 | #define UBRR1_Value ((F_CPU/(8UL*BAUD_USART1))-1)
  |
5 | UBRR0H = (uint8_t) (UBRR0_Value >> 8);
6 | UBRR0L = (uint8_t) (UBRR0_Value);
7 | UCSR0A = (1 << U2X0);
8 | UCSR0B = (1 << RXCIE0)|(1 << RXEN0)|(1 << TXEN0);
9 | UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
  |
10| UBRR1H = (uint8_t) (UBRR1_Value >> 8);
11| UBRR1L = (uint8_t) (UBRR1_Value);
12| UCSR1A = (1 << U2X1);
13| UCSR1B = (1 << RXCIE1)|(1 << RXEN1)|(1 << TXEN1);
14| UCSR1C = (1 << UCSZ11) | (1 << UCSZ10);
```

Como dito anteriormente, em ambas as USARTs foi escolhido o modo assíncrono de dupla velocidade. Para isso, foi necessário colocar o *bit* U2Xn a 1 no registo UCSRnA, como visualizado nas linhas 7 e 12.

Em relação ao registo UCSRnB foram inseridos com o valor lógico 1 os *bits* RXCIEn, RXENn e TXENn – linhas 8 e 13 – que permitem a receção e transmissão de dados, e a ativação da interrupção de receção.

Por último, no registo UCSRnC é selecionado o número de *bits* de dados, ao serem colocados a 1 os *bits* UCSZn1 e UCSZn0 como representado nas linhas 9 e 14.

5.4.2. Inicialização de variáveis

Após efetuadas todas as configurações inicializou-se as variáveis importantes no sistema tais como as que guardam o número de batimentos por minuto, a cor RGB da linha do compasso de tempo e as definições relativas a cada coluna do sistema, sendo elas o instrumento, a nota e a cor RGB associada. Para isso utilizou-se a memória EEPROM existente no microcontrolador, uma memória não volátil que serve para guardar informação durante períodos sem energia até 4 *Kbytes*.

O acesso à EEPROM é realizado através de registos especiais que controlam o endereço, os dados a serem escritos, como também as *flags* que permitem efetuar as operações de leitura ou escrita. Através da inclusão do módulo <*avr/eeprom.h*>, existente na biblioteca *avr-libc*, foi possível aceder e manipular a EEPROM com a utilização de funções pré-criadas, que simplificam o uso da memória [38].

Existem três tipos de funções: escrita, atualização e leitura. Em relação às duas primeiras, elas são bastante semelhantes dado que tanto uma como a outra efetuam operações de escrita. A única diferença é que a função de atualização realiza uma escrita na EEPROM somente se os novos dados a ser inseridos diferirem dos atuais, enquanto que a função de escrita escreve sempre os dados solicitados independentemente se forem os mesmos, o que origina uma redução do tempo de vida útil da memória. A função de leitura como o próprio nome indica permite obter a informação presente num determinado endereço.

Na inicialização do *software* são chamadas duas funções que possibilitam efetuar a leitura da memória EEPROM, sendo elas:

```
uint8_t eeprom_read_byte(const uint8_t *addr)
```

Esta função é chamada para efetuar a leitura de um *byte* inserido no único argumento da função, que diz respeito ao endereço *addr* da memória EEPROM, e guarda esse valor na variável que chamou a rotina.

```
void eeprom_read_block(void *pointer_ram, const void
*pointer_eeprom, size_t n)
```

Esta função realiza a operação de leitura de um vetor de qualquer tipo de dados presente na memória EEPROM e copia-o para um vetor localizado na memória RAM. Dentro da função encontramos três parâmetros:

- O primeiro parâmetro, o **pointer_ram*, utiliza-se para alterar o vetor pretendido, isto é, um local na memória RAM. Trata-se de um apontador do tipo *void*, o que significa que pode aceitar qualquer tipo de dados.

- O segundo parâmetro, o **pointer_eeprom*, tal como o primeiro, é um apontador do tipo *void* que representa uma localização na memória EEPROM.

- O terceiro e último parâmetro, o *n*, é o número de *bytes* que se pretende ler.

Além das funções, implementou-se também a *macro* EEMEM. Esta instrui o compilador GCC a atribuir a sua variável ao espaço de endereços EEPROM, em vez do espaço de endereços SRAM, como se fosse uma variável normal. Ao definir-se a variável com a *macro* EEMEM pode-se declarar um valor *default* [38]. Estes valores iniciais são guardados num ficheiro com extensão *.eep* sempre que for compilado o programa, sendo enviado para o microcontrolador durante a sua programação.

A última inicialização que necessitou de ser feita foi a atribuição da posição de todos os LEDs WS2812B presentes no sistema a um vetor de duas dimensões, tendo como objetivo controlá-los através da posição que se encontram na estrutura, identificando a linha e a coluna, tornando mais fácil o desenvolvimento do código. O fluxograma da Figura 60 demonstra a programação desenvolvida para esta função do *software*:

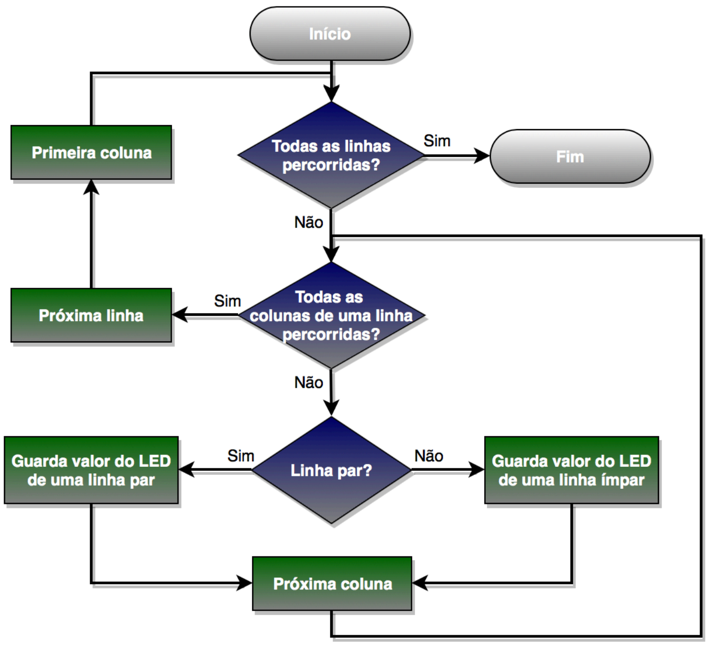


Figura 60 Fluxograma relativo à numeração dos LEDs WS2812B

A função representada na Figura 60 é constituída por dois ciclos que percorrem todas as linhas e colunas do protótipo. Dentro é determinado o tipo de linha (par ou ímpar) em cada fase do ciclo, dado que os pixéis estão posicionados ao longo da estrutura no formato de serpente, como ilustrado na Figura 41.

A condição que testa se a linha é par ou ímpar é executada através do sinal de percentagem (%), que calcula o resto da divisão entre a linha pretendida e o valor 2. Caso o resto seja 0 significa que a linha é par, caso contrário é ímpar.

De seguida, o número de cada LED é convertido para a respetiva posição na estrutura consoante o tipo de linha em que se encontra, como demonstrado nos cálculos da Tabela 24. Nesta, a variável *i* corresponde à linha que o primeiro ciclo se encontra a percorrer, enquanto que *j* diz respeito à coluna identificada pelo segundo ciclo *for*.

Tabela 24 Cálculo do número do LED para linhas pares e ímpares da estrutura

Tipo de Linha	Par	Ímpar
Cálculo do número do LED	$j + (i * 12)$	$(i + 1) * 12 - j + 1$

No final é efetuado um *enable* global de todas as interrupções utilizadas no programa. Como representado no fluxograma da Figura 59, foram utilizadas duas interrupções:

- Interrupção do *Timer/Counter0* relativa ao modo CTC (*TIMER0_COMPA_vect*).
- Interrupção da USART0 relativa à receção de dados (*USART0_RX_vect*).

Após realizadas todas as configurações e inicializações é executado um ciclo infinito no *software* (Figura 59). As duas primeiras funções realizadas dentro do ciclo (*Toggle do LED de Funcionamento* e *Tempo*) são de acordo com a interrupção gerada pelo *Timer/Counter0*, que efetua a alteração do estado do LED de funcionamento e a transição do *tempo*.

5.4.3. ROTINA DE INTERRUPÇÃO DO *TIMER/COUNTER0*

Dentro da rotina de interrupção do *Timer/Counter0* são utilizadas duas variáveis que têm como função ser um contador. Estas são incrementadas de dez em dez milissegundos como configurado, com o objetivo de se obter os tempos relativos à alteração do estado do LED de funcionamento e da linha ativa do compasso de tempo.

O primeiro contador (*cnt_led*) é utilizado para piscar o LED de funcionamento de um em segundo, estando assim o LED ligado ou desligado durante 500 ms, com o intuito de demonstrar que o microcontrolador está a operar à velocidade desejada.

Como o contador é incrementado a cada 10 ms, testa-se a variável para ver se é igual a 50, valor que significa que passaram 500 ms. Caso se cumpra a condição, reinicia-se o contador com o valor 0 e ativa-se uma *flag*, denominada *flag_led*, que permitirá trocar o estado do LED de funcionamento.

A segunda variável (*cnt_tempo*) é implementada para ser alterada a linha ativa do compasso de tempo. Assim, o princípio de funcionamento é o mesmo em relação ao contador anterior, sendo apenas alterado o tempo pretendido, que varia consoante o número de BPM do compasso de tempo, e que é calculado pela função denominada *Conversao_BPM*.

A função *Conversao_BPM* recebe como entrada o valor BPM pretendido para o compasso de tempo e converte-o para uma unidade adimensional. Esta é denominada *limite_cnt* e, quando igualada com o contador *cnt_tempo*, origina a alteração da linha ativa do compasso de tempo. O cálculo realizado dentro da função começa por converter o número de batimentos num minuto em milissegundos, como demonstrado na Equação 4:

$$Delay\ Tempo = \frac{1\ min}{Nº\ batimentos} = \frac{60000\ ms}{Nº\ batimentos}$$

Equação 4

Como a rotina de interrupção do *Timer/Counter0* é gerada de 10 em 10 ms, é necessário dividir o valor obtido na Equação 4 por esta base de tempo. Assim, o valor final da variável *limite_cnt* é calculado através da Equação 5:

$$limite\_cnt = \frac{Delay\ Tempo}{10\ ms}$$

Equação 5

Como exemplo de aplicação da função *Conversao_BPM* definimos 60 batimentos por minuto, ou noutro contexto, 1 batimento por segundo. Efetuando os cálculos, o resultado final da variável *limite_cnt* equivale a 100, o que significa que quando o contador for igual a este valor ativa a *flag_tempo*, que realizará a alteração da linha do compasso de tempo de um em um segundo, resultando num batimento por segundo.

A Figura 61 ilustra o fluxograma relativo à rotina de interrupção que colmata o que foi explicado. É de notar que o contador de *tempo* (*cnt_tempo*) só é incrementado quando o compasso de tempo não se encontra parado. Isto é executado através de uma *flag* que é colocada a 1 no início do programa, sendo alterada para 0 quando dada a ordem de paragem do compasso na aplicação Android.

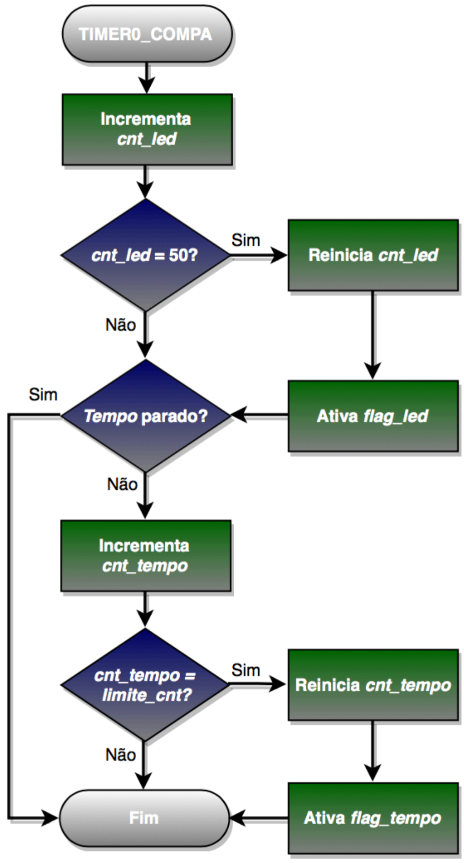


Figura 61 Fluxograma relativo à rotina de interrupção

5.4.4. *TOGGLE* DO LED DE FUNCIONAMENTO

O processo *Toggle do LED de Funcionamento*, como o próprio nome indica, executa uma mudança de estado no LED de funcionamento (ligado ao pino PB1), que tem como função indicar que o microcontrolador está a operar na velocidade correta.

O fluxograma presente na Figura 62 permite demonstrar o funcionamento da alteração do estado do LED de funcionamento:

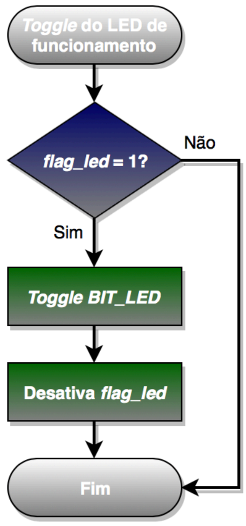


Figura 62 Fluxograma relativo à alteração do estado do LED de funcionamento

A função começa por testar a variável *flag_led* para saber se tem o valor 1. Esta é colocada somente a 1 quando tiverem passado 500 ms, isto é, quando o contador *cnt_led* chegar ao valor 50 na rotina de interrupção do *Timer/Counter0*.

Com o valor da *flag_led* igual a 1 é realizado o código no interior da condição, que efetua um *toggle* ao pino do microcontrolador onde está ligado o LED de funcionamento, configurado com o nome *BIT_LED*. De seguida, desativa *flag_led*, colocando-a a 1.

Caso a *flag* esteja desativa, não executa nenhuma linha de código em relação a este processo, o que significa que ainda não passaram 500 ms desde a última alteração do estado do LED.

5.4.5. *TEMPO*

O segundo processo relacionado com a rotina de interrupção do *Timer/Counter0* é a alteração da linha ativa do compasso de tempo. Antes da sua explicação tem de se ter em conta que existem dois formatos diferentes do *Tempo* no sistema:

- **Horizontal** – é a principal forma em que é representado o compasso de tempo, onde a sua linha ativa transita horizontalmente de linha em linha, como representado na Figura 63:

Figura 63 Representação do *Tempo* na horizontal

- **Diagonal** – formato em que a sua linha ativa representa uma diagonal do sistema (Figura 64). Além de ser um formato que apresenta uma visualização distinta do habitual de um sequenciador, permite também produzir excertos de música mais longos. Isto porque o tipo diagonal apresenta mais linhas no compasso de tempo do que o horizontal, que está limitado a um máximo de doze linhas.

Figura 64 Representação do *Tempo* na diagonal

O processo do *Tempo* é executado caso a *flag* relativa à alteração da linha do compasso de tempo (*flag_tempo*) seja ativa na rotina de interrupção do *Timer/Counter0*. O fluxograma apresentado na Figura 65 demonstra o seu funcionamento:

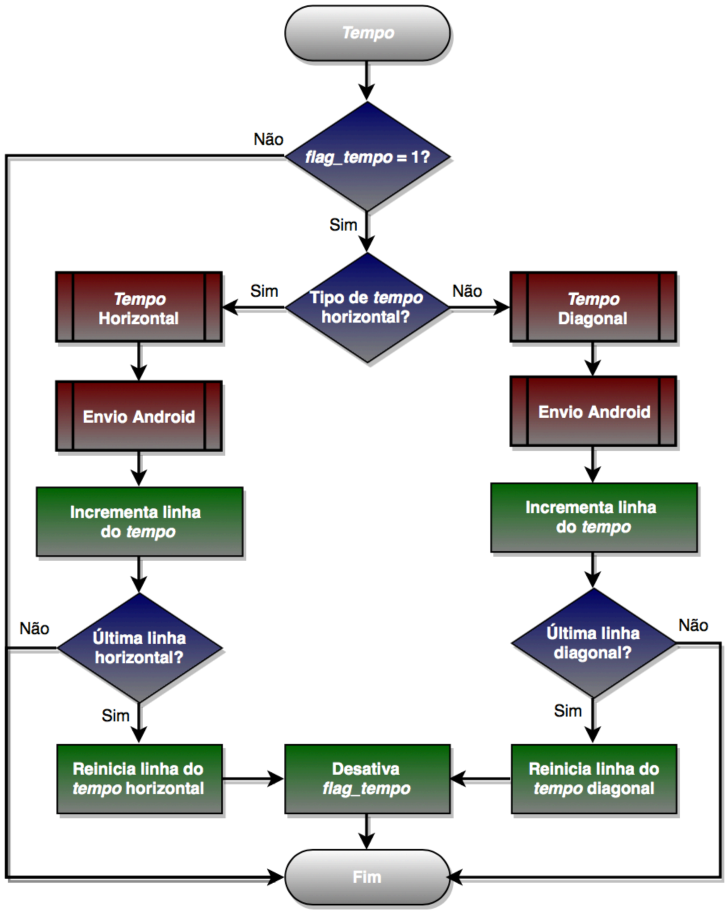


Figura 65 Fluxograma relativo ao *Tempo*

Como visível na Figura 65, o processo *Tempo* inicia com a verificação do estado da *flag* denominada *flag_tempo*. Como dito anteriormente, esta variável é ativa na rotina de interrupção do *Timer/Counter0* quando o respetivo contador (*cnt_tempo*) atinge o valor limite correspondente ao número de batimentos por minuto, calculado pela função *Conversao_BPM*.

Supondo que a *flag* de *tempo* se encontra ativa, primeiramente é determinado qual o tipo do compasso de tempo que está a ser implementado. Caso se trate do formato horizontal é executada a função *Tempo_Horizontal*, caso contrário significa que estamos perante o tipo diagonal, realizando assim *Tempo_Diagonal*. Ambas as funções servem para ligar ou desligar os LEDs correspondentes à linha ativa do compasso, sendo explicadas mais adiante.

De seguida, para cada um dos casos, é enviada uma trama para a aplicação Android com o objetivo de monitorizar a linha do compasso de tempo ativa e a sua cor. Consecutivamente, é incrementada a variável referente à linha ativa de *tempo* e testado o seu valor para saber se corresponde à última linha. Caso se confirme, a variável é colocada a 0, significando que o compasso irá reiniciar na primeira linha.

Em relação ao tipo horizontal, a última linha corresponde ao número máximo de linhas definidas pelo o utilizador na aplicação Android. Para o outro formato, o limite da linha diagonal corresponde ao valor anterior da soma entre o número máximo de linhas e colunas ativas.

Por exemplo, no caso de serem utilizadas todas as linhas e colunas do sistema (dimensão 12x12), no tipo horizontal o valor máximo de linhas percorridas é 12, enquanto que no formato diagonal são percorridas 23 linhas.

5.4.6. Leitura dos Botões

Nesta rotina é efetuada a leitura de todos os botões do sistema, através da ligação destes aos *shift registers*.

Numa primeira parte do processo é lido o estado de cada botão presente no sistema. Depois da atualização do estado dos botões é guardado o estado das oito entradas associadas a cada *shift register* presente no sistema. Para isso, implementa-se um ciclo *for* que percorre todos os *shift registers*, em que é chamada uma função, denominada *ShiftIn*, para gravar o estado de oito botões de cada vez.

O fluxograma da Figura 66 representa a primeira parte do processo de leitura dos botões:

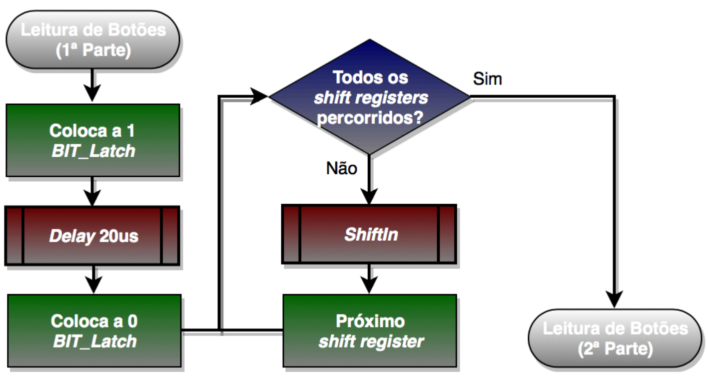


Figura 66 Fluxograma relativo à primeira parte do processo de leitura dos botões

No início do processo é alterado o *bit latch* do *shift register*, sendo colocado a 1 para determinar o estado de todas as entradas, e depois a 0 para transmitir o estado de cada uma delas, à medida que o pino *clock* do *shift register* altera. Entre a alteração do seu valor lógico está inserido um *delay* de 20 µs, implementado para garantir que todas as entradas (botões) sejam lidas, como analisado no *datasheet* do componente (CD4021BE).

De seguida é guardado o estado de oito botões, ligados a cada *shift register*, com a chamada da função *ShiftIn*. O fluxograma apresentado na Figura 67 representa o funcionamento desta:

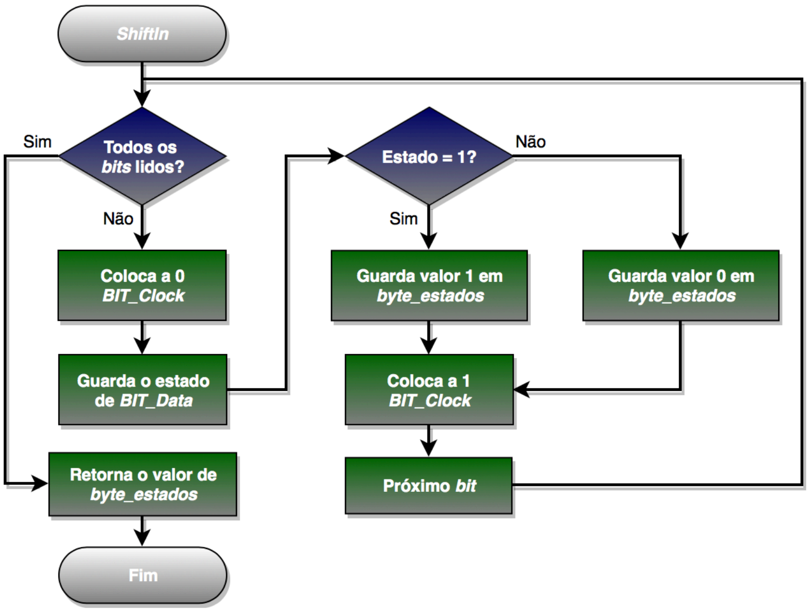


Figura 67 Fluxograma relativo à função *ShiftIn*

Na função *ShiftIn* é realizada a leitura de oito botões de um *shift register*, através da manipulação do seu *bit clock*. Para isso implementou-se um ciclo que percorrerá todos os *bits* do *shift register*, desde o *bit* mais significativo (*bit* 7) até o menos significativo (*bit* 0).

Dentro do ciclo ocorre uma mudança do *BIT_Clock* de 0 para 1, que faz com que o *shift register* altere o pino de saída de dados (*BIT_Data*) ou não, consoante o estado do botão seguinte, e o guarde numa variável de oito *bits*, denominada *byte_estados*.

No final, depois de serem percorridas as oito entradas associadas ao componente, a variável *byte_estados* é retornada para a função *Leitura dos Botões* e guardada, transitando de seguida para o próximo *shift register*, como demonstrado no fluxograma da Figura 66.

Terminada a primeira parte do processo de leitura dos botões, correspondente à gravação do estado de todos os botões, é determinada a posição de cada um.

A segunda parte do processo de leitura dos botões consiste na atribuição do estado de cada botão à posição relativa na estrutura, tal como acontece na função de numeração dos LEDs WS2812B, analisada através do fluxograma da Figura 60. Esta designação é realizada com o intuito de determinar qual o estado de cada posição, para que seja efetuado o controlo dos LEDs, consoante a linha e a coluna onde se situam os respetivos botões.

O fluxograma ilustrado na Figura 68 demonstra como é executada a segunda parte do processo de leitura dos botões.

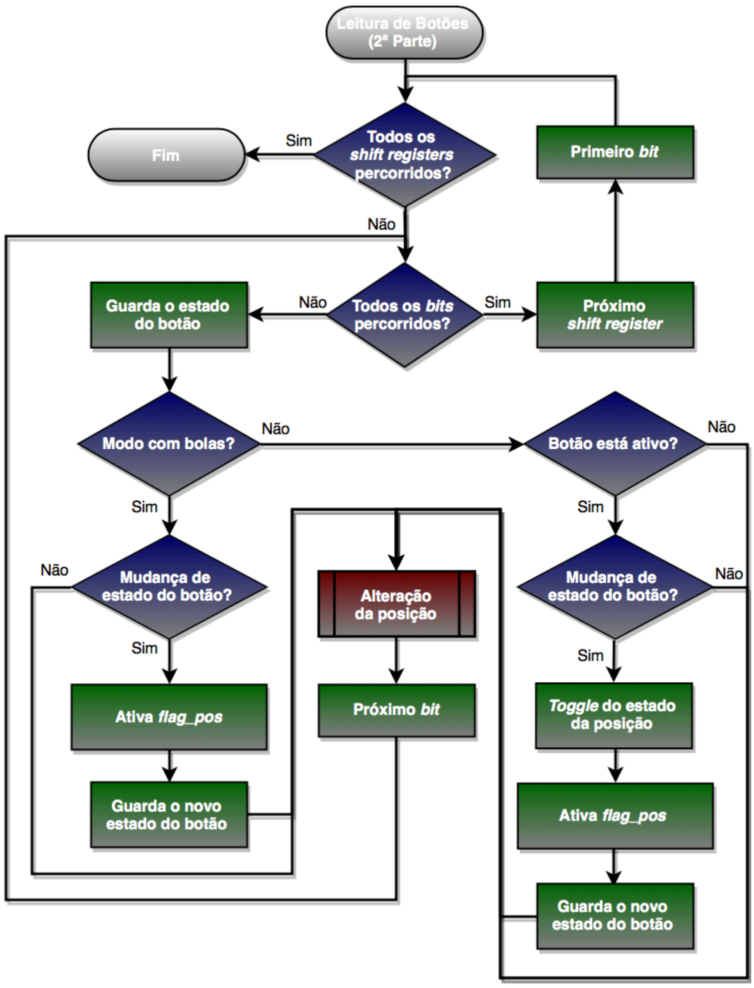


Figura 68 Fluxograma relativo à segunda parte do processo de leitura dos botões

Através do fluxograma da Figura 68 verifica-se que a segunda parte do processo de leitura dos botões é constituída por dois ciclos, em que o primeiro percorre todos os *shift registers*, e o segundo os oito *bits* referentes ao estado de todos as entradas de cada um.

Dentro dos ciclos é feita a atribuição do estado de todos botões a um vetor de duas posições, que identifica a sua posição no sistema, isto é, a linha e a coluna onde se encontram. De seguida verifica-se qual o modo que está a ser utilizado:

- Caso esteja a ser utilizado o modo com bolas, o primeiro passo que realiza é verificar se houve uma mudança de estado no botão. Se se confirmar esta alteração é ativada a *flag* de mudança de estado da posição (*flag_pos*) onde se encontra o botão, que servirá para indicar uma mudança de cor no respetivo LED WS2812B. Por fim, a variável referente ao estado anterior do interruptor é atualizada com o novo estado.

- Caso o modo utilizado pelo utilizador seja o modo sem bolas, o *software* verifica se o botão está a ser pressionado, para que a seguir compare se o botão anteriormente estava premido ou não. Se houver confirmação que o estado anterior seja diferente do atual, isto é, que foi premido, é realizado um *toggle* – uma mudança de valor entre zero e um e vice-versa – na posição onde se encontra presente o botão, sendo também ativada a *flag_pos,* indicando uma alteração do estado desta.

Depois de feita a verificação do modo de funcionamento e da execução dos processos iniciados por cada um, é realizada a alteração das variáveis de linha e coluna, de forma a indicar a posição do próximo botão. O seu fluxograma é representado na Figura 69:

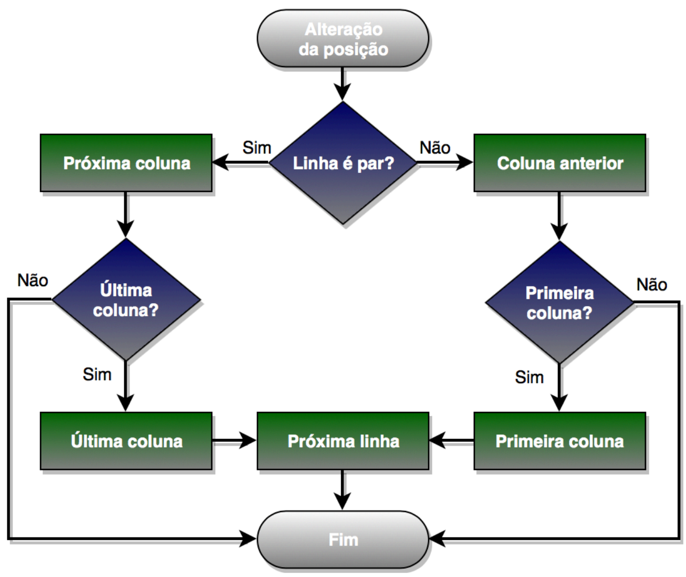


Figura 69 Fluxograma relativo à função de alteração das variáveis de linha e coluna

A função representada no fluxograma da Figura 69 foi desenvolvida com base no esquema de ligação dos botões aos *shift registers*. Tal como os LEDs WS2812B, os botões encontram-se ligados sequencialmente ao longo da estrutura num formato do tipo "cobra". Assim, a identificação de cada botão é realizada dependendo se a linha é par ou ímpar.

Inicialmente é testado o valor da linha. No caso de ser par, a variável que indica o número da coluna do botão é incrementada, enquanto que se for ímpar é decrementada.

Uma linha do sistema aceita valores entre zero e onze, portanto, quando não se encontra nesta gama (abaixo de zero ou acima de onze) significa que a atribuição do estado de todos os botões de uma linha já foi realizada, tendo de se avançar para a linha seguinte.

Sempre que incrementada a variável de linha é testado o seu valor para saber se se encontra entre a gama de valores de zero a onze. Então, caso esta variável seja igual ou superior a doze significa que a leitura de todos os botões do sistema foi concluída, dando por encerrado todo o processo de *Leitura dos Botões*.

5.4.7. CONTROLO DOS LEDS

Terminado o processo de leitura dos botões, procede-se para o controlo dos LEDs WS2812B.

Além de explicada e analisada ao todo a função de controlo dos LEDs RGB, são também justificadas as funções desenvolvidas para a alteração de cor dos WS2812B, de acordo com o seu protocolo de transferência de dados.

A função de *Controlo dos LEDs* atribui a cor a cada LED do sistema, de acordo com o modo de funcionamento e o estado de cada posição. O fluxograma que a representa é visualizado na Figura 70:

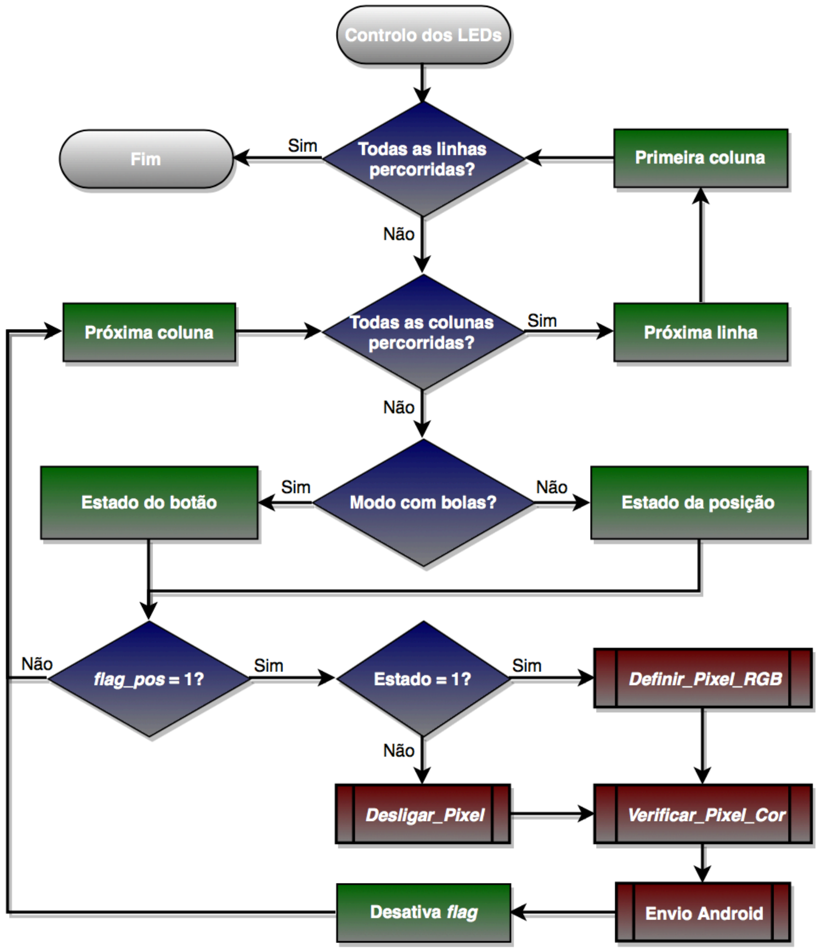


Figura 70 Fluxograma relativo ao processo de controlo dos LEDs

O processo de *Controlo dos LEDs* verifica inicialmente qual o modo de funcionamento atual no sistema, com o intuito de determinar como é lido o estado de cada posição.

No modo com bolas, o estado da posição corresponde ao estado do seu botão, que no caso de corresponder ao valor 1 significa que a bola está colocada na posição do botão, o que faz com que esta e o seu LED estejam ativos. Em relação ao modo sem bolas, o estado da posição é alterado quando pressionado o seu botão.

Em seguida, testa-se o valor da *flag* de mudança de estado da posição (*flag_pos*), que é ativa ou não no processo de leitura dos botões. Caso a *flag* se encontre ativa, significa que a posição sofreu uma mudança de estado, em que é determinado o seu valor. Depois, se o estado de uma posição for 1 é chamada a função de atribuição da cor do LED, denominada *Definir_Pixel_RGB*, caso contrário é desligado o pixel através da função *Desligar_Pixel*.

Por último, é verificada qual a cor atribuída ao LED, através da função *Verificar_Pixel_Cor*, para que seja enviada a informação referente à cor do pixel, para monitorização do sequenciador na aplicação Android.

Depois de explicado o funcionamento de todo o processo de alteração da cor dos LEDs, são analisadas todas as funções implementadas no controlo dos LEDs, que se encontram guardadas numa biblioteca desenvolvida à parte, denominada *funcs_ws2812b*. Todo o código desenvolvido na biblioteca é disponibilizado no Anexo E.

Definir_Pixel_RGB

A primeira função da biblioteca *funcs_ws2812b* chama-se *Definir_Pixel_RGB*, sendo a mais importante no controlo dos LEDs. Esta foi desenvolvida com o objetivo de atribuir uma cor a cada pixel presente no sistema.

O fluxograma da Figura 71 explica o funcionamento da função *Definir_Pixel_RGB*:

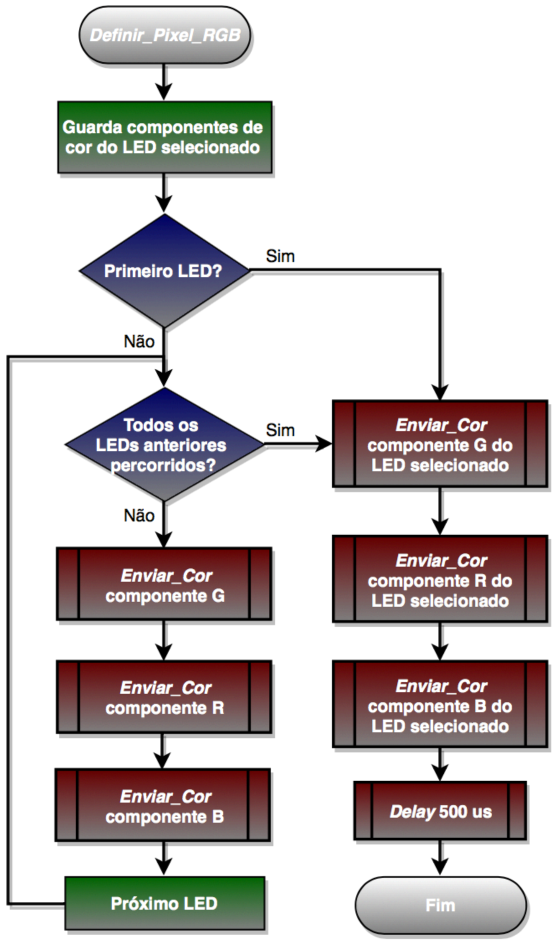


Figura 71 Fluxograma relativo à função *Definir_Pixel_RGB*

A função é composta por quatro argumentos de entrada, em que o primeiro é relativo ao LED que se pretende alterar a cor, e os outros três a cada uma das componentes RGB. Como demonstrado na Figura 71, inicialmente, é guardado num vetor de duas dimensões – de tamanho 144 (número de LEDs RGB) por 3 (número de componentes de cor) – cada componente de cor definida nos argumentos da função.

De seguida é verificado se o LED inserido corresponde ao primeiro, ou seja, se tem valor 0. Se se confirmar é enviada somente a alteração de cor desse pixel dado que se trata do primeiro do sistema. Assim, é chamada a função *Enviar_Cor* três vezes para efetuar o envio de cada componente da cor.

O mesmo processo é executado caso não se verifique a condição, porém, antes da alteração de cor do LED pretendido, são enviadas as componentes de cor dos anteriores. Para isso é chamada a função *Enviar_Cor*, que permite atribuir a cor ao LED pela ordem GRB definida no protocolo (Figura 26).

Por último é realizado um *delay* de 500 µs que indica que não há mais nenhuma atribuição de cor a ser realizada. Isto faz com que da próxima vez que seja definida uma cor para um pixel, a informação comece a ser enviada novamente a partir do primeiro LED do sistema, que é ligado ao pino PB0 do microcontrolador, como demonstrado na Figura 44.

A função *Enviar_Cor* foi desenvolvida na linguagem de programação *Assembly*. Trata-se de uma linguagem de baixo nível que foi escolhida em relação à utilizada no resto do *software*, que foi o C, devido aos requisitos temporais implementados pelo protocolo de transferência de dados dos LEDs WS2812B. O seu desenvolvimento foi baseado no gráfico da Figura 72:

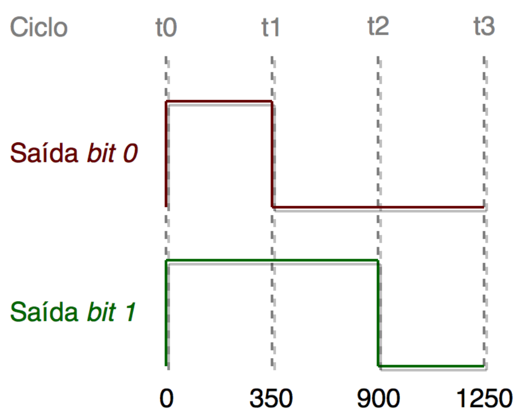


Figura 72 Gráfico utilizado na transferência de dados para os LEDs WS2812B

O gráfico da Figura 72 ilustra como é enviado o *bit* zero ou um para o pino de saída de dados ligado aos LEDs WS2812B (*BIT_WS2812B*), de forma a indicar a cor que se pretende para cada um. Tanto um *bit* como o outro são representados através do envio de pulsos ascendentes e descendentes em tempos definidos.

O ciclo inicia em *t0*, isto é, nos zero segundos, com o envio de um pulso ascendente para o *bit* de saída de dados nos dois casos. De seguida, passados 350 ns até *t1*, a saída transmite um pulso descendente caso seja pretendido enviar o *bit* 0, caso contrário continua inalterada.

Em *t2*, após ocorridos 900 ns, é enviado um pulso descendente independentemente do *bit* pretendido. Por último, em *t3*, mantendo a saída permanente, é transferido o *bit* para os LEDs WS2812B, tendo demorado ao todo 1250 ns, como descrito no protocolo de transferência de dados.

Assim, conclui-se que ao longo de toda a transferência do *bit*, a saída é apenas alterada uma vez consoante o seu valor lógico, permanecendo os restantes ciclos inalterados. Isto facilitou bastante o desenvolvimento do código para o envio de um *byte* correspondente a uma componente de cor vermelha, verde ou azul.

O fluxograma da Figura 73 demonstra o funcionamento da função *Enviar_Cor*:

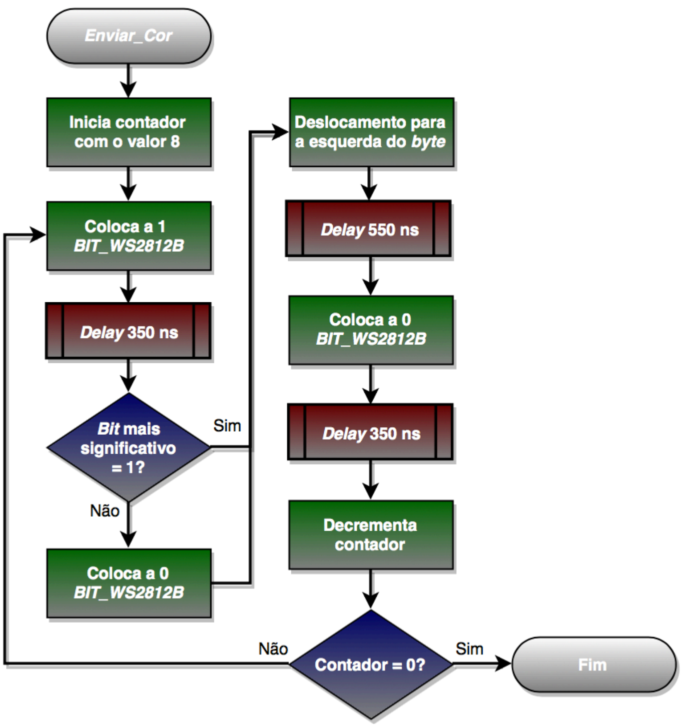


Figura 73 Fluxograma relativo à função *Enviar_Cor*

Dado que a função foi desenvolvida na linguagem de programação *Assembly*, todos os *delays* mencionados no fluxograma da Figura 73 foram executados através da instrução *nop*. Esta corresponde a um ciclo de funcionamento do microcontrolador, que corresponde a 62.5 ns.

Sabendo o tempo de um ciclo no microcontrolador, é possível calcular o número de ciclos para cada intervalo de tempo da Figura 72, através da Equação 6:

$$Nº\ Ciclos = \frac{Tempo\ pretendido\ em\ ns}{62.5}$$

Equação 6

Como exemplo, temos o intervalo de tempo entre *t0* e *t1*, que corresponde a 350 ns. Portanto, realizando o cálculo apresentado na Equação 6, temos um total de aproximadamente seis ciclos, o que significa que a instrução *nop* será executada seis vezes.

Regressando ao fluxograma da Figura 73, primeiramente é definido o contador que percorre os oito *bits* da componente de cor com o valor 8. De seguida é realizado o processo de transferência de dados explicado anteriormente com a ajuda do gráfico da Figura 72.

Em primeiro lugar é colocado a 1 o pino ligado aos LEDs WS2812B (*BIT_WS2812B*) e executado o número de ciclos correspondentes ao intervalo de tempo entre *t0* e *t1* (350 ns). Depois é testado o valor do *bit* mais significativo, para que caso tenha o valor 0 execute um pulso descendente, ou seja, coloque *BIT_WS2812B* a 0. Caso contrário, esta última instrução não é executada.

Testado o *bit*, efetua-se o deslocamento à esquerda do *byte* inteiro para que seja lido o próximo *bit* mais significativo, e é realizado o número de ciclos entre *t1* e *t2*, correspondente a 550 ns.

De seguida, independentemente do valor do *bit*, é enviado um pulso descendente através do envio do valor 0 ao pino de saída, e executado o número de ciclos correspondentes ao intervalo entre *t2* e *t3*. O valor em nanossegundos deste intervalo é 350 ns, dado que o tempo total de envio é 1250 ns e já foram realizados 900 ns.

Finalmente é decrementado o contador e feito o teste a ver se todos os *bits* foram percorridos e enviados, através da comparação do seu valor com zero. Caso já tenham sido transferidos todos os *bits* significa que a função chegou ao fim e que a componente de cor foi enviada.

Desligar_Pixel

A função *Desligar_Pixel*, como o próprio nome indica, permite desligar um pixel presente no sistema, através da alteração da sua cor para preto. Assim, o que esta rotina executa é uma chamada à função *Definir_Pixel_RGB* com o número do pixel que se pretende desligar, e com todas as componentes de cor com o valor 0, que resultam na cor preta.

Verificar_Pixel_Cor

A função permite saber qual a cor atual de um LED WS2812B no sistema, pela ordem das componentes de cor vermelha, verde e azul.

O vetor bidimensional referido anteriormente na função *Definir_Pixel_RGB* possibilita o conhecimento das cores atuais de cada LED e é denominado *pixels_cores*. A primeira dimensão identifica o pixel, enquanto que a segunda (de tamanho 3) diz respeito a todas as componentes de cor, em que na posição 0 é identificado o valor da componente verde, na 1 a componente vermelha e, por último, na posição 2 é localizada a componente azul.

Assim, a função retorna a cor atual de um LED – no formato RGB – através da junção de todas as suas componentes de cor, como exposto no seguinte excerto de código:

```
return ((uint32_t)pixels_cores[num_pixel][0] << 8)  |
       ((uint32_t)pixels_cores[num_pixel][1] << 16) |
       ((uint32_t)pixels_cores[num_pixel][2]);
```

Cada componente de cor corresponde a oito *bits* de informação. Para ser retornada a cor final é executada a ligação das três componentes numa variável de 32 *bits*, através da operação lógica OR. Assim, torna-se necessário organizar as componentes na variável final.

A ordem da variável de 32 *bits* para o formato RGB é efetuada com a instrução de deslocamento à esquerda (<<), em que a componente vermelha é deslocada 16 posições e a verde 8, enquanto que a azul não se desloca dado que é a última componente.

Desligar_Todos_Pixeis

Esta função permite que todos os LEDs presentes no sistema sejam desligados, através da chamada da função *Definir_Pixel_RGB* dentro de um ciclo *for* que percorre todos os pixéis, com as componentes de cor equivalentes a zero, correspondendo à cor preta.

5.4.8. CONTROLO DOS LEDS DO COMPASSO DE TEMPO

O processo de controlo dos LEDs da linha ativa do compasso de tempo atribui as cores definidas para a transição do compasso de tempo, com base na dimensão do número de linhas e colunas ativas. A atribuição da cor do compasso de tempo aos pixéis posicionados na sua linha ativa é realizada caso não estejam ativos.

Como descrito na subseção 5.4.5, existem dois tipos de compasso de tempo, sendo eles o horizontal e o diagonal. Para cada formato do *tempo* foi desenvolvida uma função dado que o controlo dos LEDs é realizado de forma diferente.

Tempo Horizontal

Como apresentado anteriormente na Figura 63, a função relacionada com o *tempo* horizontal percorre todas as colunas da linha ativa do compasso de tempo, com o intuito de ativar os LEDs da mesma com a cor deste, à exceção das posições que estejam ativas. As posições ativas não sofrem nenhuma mudança de cor e enviam o comando MIDI *Note On*, para produção da nota do instrumento musical associado à sua coluna.

O fluxograma da Figura 74 analisa a programação desenvolvida para o compasso de tempo do tipo horizontal:

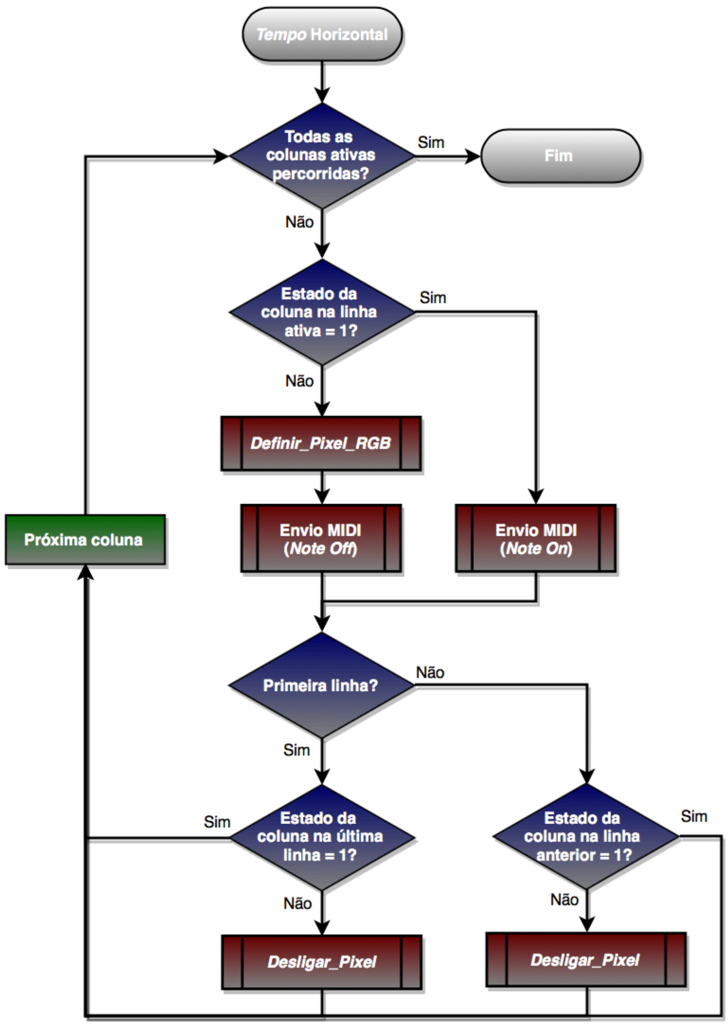


Figura 74 Fluxograma relativo ao *tempo* no tipo horizontal

O processo identificado pelo fluxograma da Figura 74 percorre todas as colunas ativas no sistema. De seguida, é determinado o estado de todos os pixéis posicionados na linha ativa do compasso de tempo:

- Caso a variável esteja desativada significa que o LED não se encontra ligado, sendo então chamada a função *Definir_Pixel_RGB* para ser modificada a cor para a do compasso de tempo. Também é enviado um comando MIDI do tipo *Note Off*, para que não haja produção musical associada à coluna onde se situa o LED.

- Caso a variável tenha o valor um, não há atribuição nenhuma de cor ao LED e é transmitido um comando MIDI *Note On*, que produz a nota musical do instrumento associado à coluna onde está presente o pixel.

De imediato é realizada a transição do compasso de tempo de uma linha para a outra, através da desativação dos pixéis da linha anterior que não se encontram ativos.

Para isso, primeiramente, é averiguado se a linha ativa atual corresponde à primeira. Caso corresponda é determinado o estado da última linha ativa e da coluna relativa ao valor do ciclo, que se tiver o valor 0 efetua a chamada à função *Desligar_Pixel*. Caso contrário, é testado o estado relativo ao posicionamento do LED na linha anterior à linha ativa do compasso, sendo também desligado este se se encontrar desativada a variável.

Tempo Diagonal

A função relativa ao *tempo* no formato diagonal executa o controlo dos LEDs de maneira diferente que o tipo horizontal, dado que a linha ativa do compasso de tempo corresponde a uma diagonal, como visualizado anteriormente na Figura 64. Neste tipo, o compasso é composto por duas fases:

- A primeira fase é ascendente, ou seja, à medida que o compasso avança, o número de pixéis que compõe a diagonal ativa do mesmo aumenta. Isto acontece até à diagonal que engloba o LED da última coluna ativa, significando que se percorreu apenas metade do *tempo*.

- Percorrendo metade do compasso de tempo é iniciada uma fase descendente, em que o número de LEDs começa a diminuir até ao final.

Assim, à exceção do controlo dos LEDs da linha ativa do compasso de tempo, o princípio de funcionamento da função *Tempo Diagonal* é idêntico ao da função *Tempo Horizontal*. Isto significa que em cada transição do compasso são verificadas quais as posições que se encontram ativas na sua linha, para que sejam produzidas as respetivas notas musicais.

5.4.9. ENVIO DE TRAMAS ANDROID

O processo de envio de comandos Android é realizado pela USART0. Neste são enviadas tramas para a aplicação presente no *smartphone*, que permitem a monitorização de todos os parâmetros do sistema.

A Figura 75 apresenta o cabeçalho de cada trama enviada para a aplicação Android:

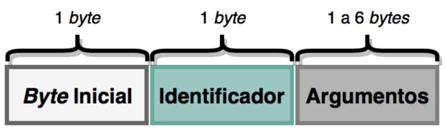


Figura 75 Cabeçalho geral de uma trama enviada para a aplicação Android

Uma trama enviada pelo microcontrolador para a aplicação Android é constituída por três campos:

1. ***Byte* Inicial** – corresponde ao primeiro *byte* que constitui uma trama, sendo representado pelo valor 129.

2. **Identificador** –reconhece qual o tipo de trama através do seu valor, que como descrito na Tabela 25, varia entre 0 e 7.

3. **Argumentos** – consiste em dados sobre o sistema, que a aplicação Android utiliza para efetuar a monitorização deste.

A Tabela 25 descreve todas as tramas que podem ser enviadas pelo microcontrolador para a aplicação Android, que são identificadas pelo seu segundo *byte*:

Tabela 25 Tramas enviadas para a aplicação Android

Identificador	Descrição
0	Informações sobre um pixel como o estado, a linha e a coluna onde se situa, e a sua cor RGB
1	Informações sobre o compasso de tempo como a linha que está ativa e a sua cor RGB
2	Número de batimentos por minuto (BPM) do compasso de tempo
3	Estado do compasso de tempo
4	Tipo de compasso de tempo
5	Definições associadas a cada coluna como o instrumento, a nota e a sua cor RGB
6	Dimensão da matriz através do número de linhas e colunas ativas
7	Modo de funcionamento do sistema

Analisadas todas as tramas que podem ser enviadas para a aplicação Android, assim como cada campo que as compõe, é agora ilustrado o cabeçalho de cada uma na Figura 76. A sua descrição é realizada através do segundo *byte* (identificador), que corresponde à primeira coluna da Tabela 25:

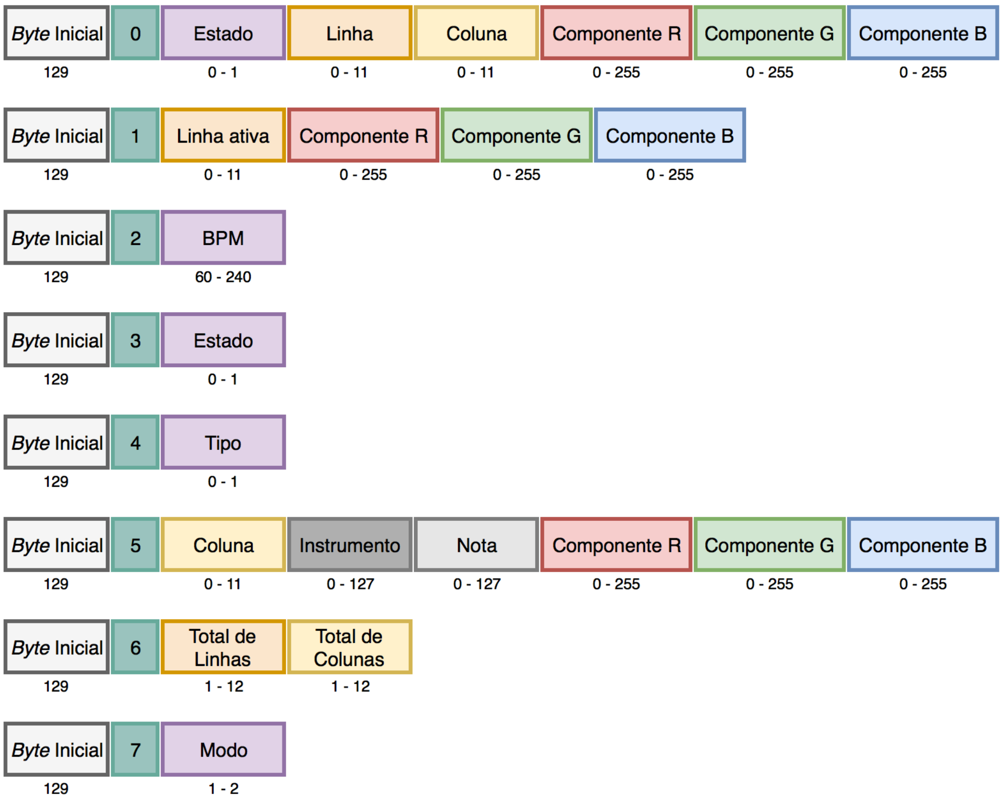


Figura 76 Cabeçalho de cada trama enviada para a aplicação Android

Como visível na Figura 76, cada trama tem tamanho diferente devido ao seu número de argumentos. Assim, tornou-se necessário o desenvolvimento de uma função que aceita um número variável de parâmetros, que na linguagem de programação C se denomina por *variadic functions*.

Este tipo de funções são implementadas através da inserção de reticências como último parâmetro na sua lista de argumentos. Estas são chamadas com a definição dos argumentos fixos (sendo necessário no mínimo um), seguidos dos argumentos variáveis. Um exemplo de uma *variadic function* é a função *printf*, que permite imprimir num terminal o número de elementos que se pretende.

Para a rotina de envio de tramas Android, a função variável foi implementada com apenas um argumento fixo (*id_byte*), que corresponde ao *byte* identificador da trama a ser enviada. Seguido do argumento fixo são inseridos os argumentos variáveis, que variam conforme o tipo de trama. O seguinte excerto de código demonstra o cabeçalho da função de envio de tramas Android, assim como a sua implementação:

```
  | // Cabeçalho da função Envio_Android
1 | void Envio_Android(uint8_t id_byte, ...)
2 | {
3 |   ...
4 | }
  |
  | // Implementação da função Envio_Android
5 | Envio_Android(2, 120);
```

Na linha 5 do código, a função *Envio_Android* é implementada para envio da trama relativa ao número BPM do compasso de tempo. Assim, são definidos dois argumentos em que o primeiro corresponde ao identificador da trama (valor 2), e o segundo aos batimentos por minuto do *tempo* (valor 120).

Antes da análise da função *Envio_Android* é necessário ter em conta que a função que realiza a transmissão de um *byte* pela USART0 é denominada *Envio_Android_Byte*. Esta recebe como argumento o valor do *byte* que se pretende enviar.

Dentro da função é testado o valor da *flag* UDRE0, pertencente ao registo UCSR0A. Esta indica se o *buffer* de transmissão está preparado para receber novos dados para enviar. Assim, o envio do *byte* pretendido só é realizado quando a *flag* UDRE0 está ativa. Caso contrário é necessário esperar que o *buffer* de transmissão esteja vazio.

O conteúdo presente no interior da função de envio de tramas Android é representado pelo fluxograma da Figura 77:

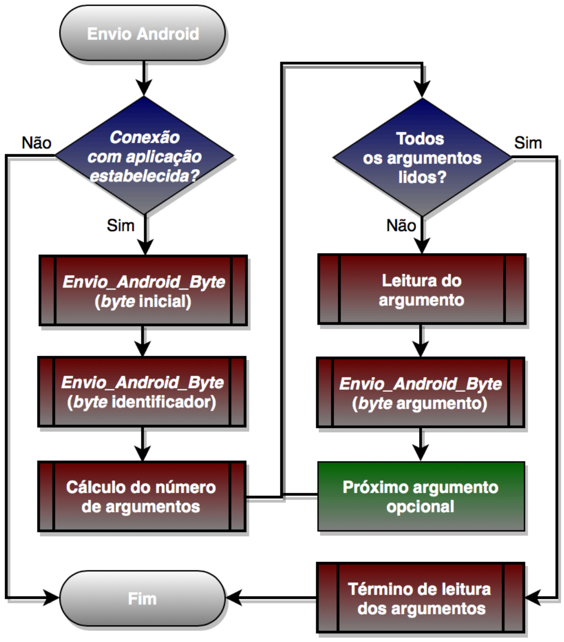


Figura 77 Fluxograma relativo ao envio de tramas Android

O fluxograma da Figura 77 começa por determinar se a conexão à aplicação Android já foi estabelecida. No caso da ligação à aplicação Android estar estabelecida, é transmitida a trama pretendida para a aplicação Android, caso contrário não é enviada.

O primeiro *byte* enviado na trama é sempre o *byte* inicial que é definido com o valor 129, dado que corresponde a um valor que é raramente utilizado como argumento. Este é enviado pela USART0 através da chamada da função *Envio_Android_Byte*. De seguida, é transmitido o *byte* relativo ao argumento fixo (*id_byte*), que identifica o tipo de trama que é enviado.

Após o envio dos *bytes* inicial e identificador são calculados o número de argumentos, que varia consoante o tipo de trama que se pretende enviar, para que sejam enviados para a aplicação Android através da função *Envio_Android_Byte*. Por fim, percorridos todos os argumentos opcionais, é terminado o processo de leitura destes, sendo concluída a função de envio de tramas Android.

Como referido anteriormente, a USART0 foi configurada com uma taxa de transmissão de 115200 *bits* por segundo. Assim, o tempo de envio de um by*te* é calculado através da Equação 7:

$$Tempo\ de\ envio\ de\ um\ byte = \frac{Start\ Bit + 8\ Bits + Stop\ Bit}{Número\ total\ de\ bits} = \frac{10}{115200} \cong 87\mu s$$

Equação 7

A Tabela 26 identifica o tempo de envio necessário para cada trama transmitida, com a inclusão do *byte* inicial e do identificador do tipo de trama:

Tabela 26 Tempo de envio necessário para cada trama

Identificador	Trama	Tempo de envio necessário
0	Informação sobre um pixel	$8\ bytes * 87\ \mu s = 696\ \mu s$
1	Informação sobre o compasso de tempo	$6\ bytes * 87\ \mu s = 522\ \mu s$
2	Número de BPM do compasso de tempo	$3\ bytes * 87\ \mu s = 261\ \mu s$
3	Estado do compasso de tempo	$3\ bytes * 87\ \mu s = 261\ \mu s$
4	Tipo do compasso de tempo	$3\ bytes * 87\ \mu s = 261\ \mu s$
5	Definições associadas a cada coluna	$8\ bytes * 87\ \mu s = 696\ \mu s$
6	Dimensão da matriz	$4\ bytes * 87\ \mu s = 348\ \mu s$
7	Modo de funcionamento do sistema	$3\ bytes * 87\ \mu s = 261\ \mu s$

5.4.10. ENVIO DE MENSAGENS MIDI

As mensagens MIDI são enviadas para o computador através da USART1. No sistema são implementados no máximo três comandos sendo eles *Note On*, *Note Off* e *Program Change*. Assim, à semelhança do processo de envio de tramas Android, foi desenvolvida uma *variadic function* que aceita dois argumentos fixos que compõem o *byte* de estado, sendo eles o tipo de comando e o canal MIDI destinado. Os restantes argumentos da função são variáveis dado que dependem do tipo de mensagem MIDI que se pretende enviar.

O seguinte excerto de código demonstra o cabeçalho da função de envio de comandos MIDI, assim como a sua implementação:

```
  | // Cabeçalho da função Envio_MIDI
1 | void Envio_MIDI(uint8_t comando, int canal, ...)
2 | {
3 |   ...
4 | }
  |
  | // Implementação da função Envio_MIDI
5 | Envio_MIDI(0b10010000, 5, 60, 127);
```

Na linha 5 do código, a função *Envio_MIDI* é implementada para envio da nota musical C4 (valor 60) para o canal MIDI 5. Assim, o primeiro argumento da função mostra que é para ser enviada uma mensagem *Note On*, para o canal mencionado no segundo argumento, cuja nota corresponde ao valor 60. Por último, no quarto argumento é definido o valor 127 que indica que a nota é tocada com a intensidade máxima.

O fluxograma da Figura 78 analisa a programação relativa à função de envio de comandos MIDI, denominada *Envio_MIDI*:

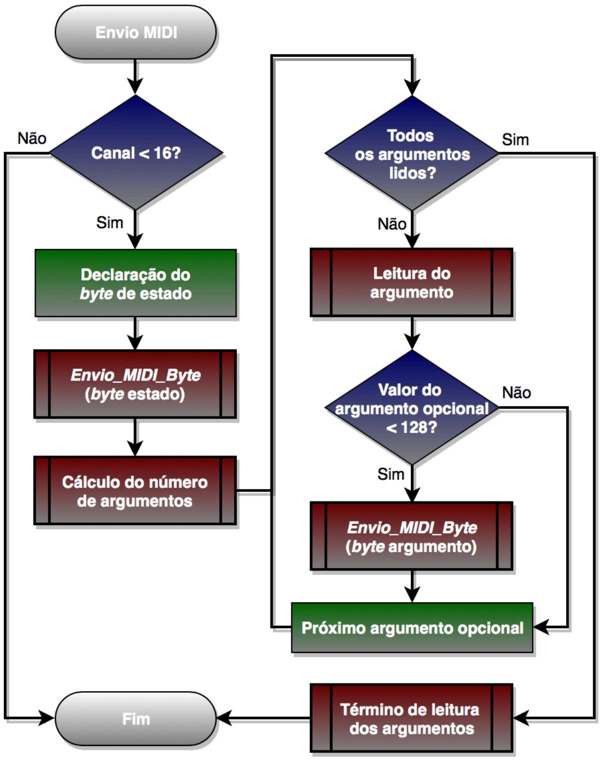


Figura 78 Fluxograma relativo à função de envio de comandos MIDI

Tal como na função de envio de tramas para a aplicação Android (*Envio_Android*), neste processo também é chamada uma função idêntica à *Envio_Android_Byte*, sendo somente alterados os registos para corresponderem à USART1. A esta função é atribuída o nome *Envio_MIDI_Byte*.

Inicialmente, relativamente à Figura 78, é realizada uma verificação ao argumento do canal MIDI que se pretende enviar o comando. Para o canal ser válido, o seu valor tem de estar compreendido entre zero e quinze. Estando dentro deste intervalo, começa por determinar o valor do *byte* de estado.

O *byte* de estado, explicado anteriormente na subsecção 3.1.3, é constituído por oito *bits* em que o mais significativo é sempre colocado a 1, os três *bits* seguintes identificam o comando pretendido, e os restantes quatro o canal MIDI. Assim, o valor do *byte* de estado é calculado realizando a operação lógica OR entre os dois argumentos fixos da função – *comando* e *canal*.

Depois de ser calculado o *byte* de estado e transmitido pela USART1 através da chamada da função *Envio_MIDI_Byte*, é iniciada a leitura dos argumentos não fixos, que correspondem aos *bytes* de dados, e que variam consoante o tipo de mensagem.

Sabendo o tipo de mensagem MIDI que se pretende enviar é possível determinar por quantos *bytes* de dados esta é composta. Como descrito na Tabela 4, as mensagens *Note On* e *Note Off* são constituídas por dois *bytes* de dados, enquanto que *Program Change* apenas necessita de um.

Determinado o número de *bytes* de dados procede-se à sua leitura e envio para o computador, sendo testados para confirmar que o seu valor não ultrapassa 127. Por último, lidos todos os argumentos opcionais, é terminada a leitura destes parâmetros.

Através da Equação 7 criou-se a Tabela 27, que indica o tempo necessário para a transmissão de cada mensagem MIDI:

Tabela 27 Tempo de envio necessário para uma mensagem MIDI

Tipo de Mensagem MIDI	Tempo de envio necessário
Note On	$3\ bytes * 87\ \mu s = 261\ \mu s$
Note Off	$3\ bytes * 87\ \mu s = 261\ \mu s$
Program Change	$2\ bytes * 87\ \mu s = 174\ \mu s$

5.4.11. Rotina de Interrupção da Receção de Dados na USART0

Após explicadas as funções de transmissão de dados pela USART0, para envio de comandos Android, e pela USART1, para envio de mensagens MIDI, é analisada a forma como o microcontrolador recebe dados.

Os únicos conjuntos de dados que o microcontrolador recebe são provenientes da aplicação Android, dado que esta além de efetuar a monitorização do sistema também possibilita a alteração de parâmetros do mesmo.

A Figura 79 disponibiliza todas as tramas enviadas pela aplicação Android para o microcontrolador:

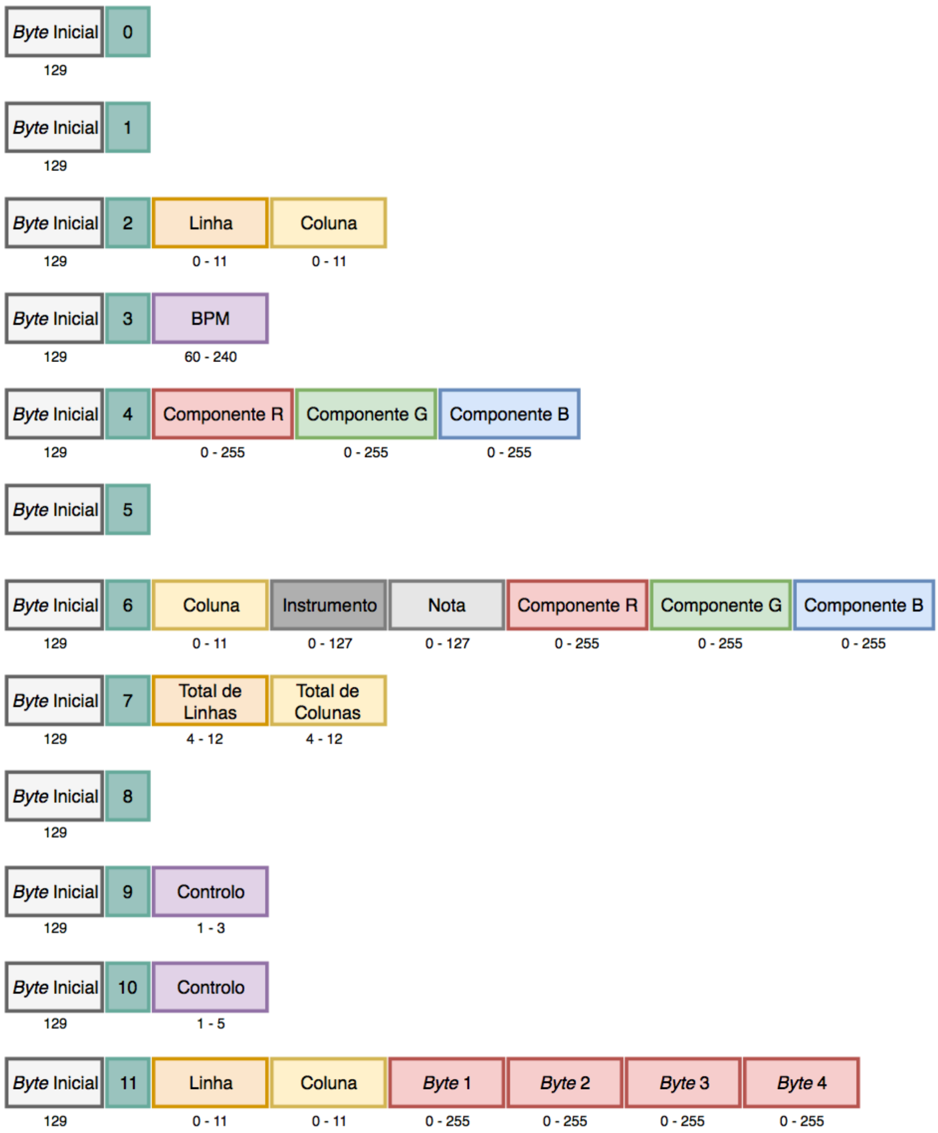


Figura 79 Cabeçalho de cada trama enviada pela a aplicação Android

Cada trama apresentada na Figura 79 é identificada pelo seu segundo *byte*, que corresponde à primeira coluna da Tabela 28:

Tabela 28 Tramas enviadas pela a aplicação Android

Identificador	Descrição
0	Pedido geral de todas as informações do sistema
1	Pedido de configurações associadas a cada coluna
2	Alteração de um pixel no modo de funcionamento sem bolas
3	Alteração do número de batimentos por minuto do compasso de tempo
4	Alteração da cor da linha ativa do compasso de tempo
5	Alteração do tipo do compasso de tempo
6	Alteração das configurações associadas a uma coluna
7	Alteração da dimensão do sequenciador
8	Alteração do modo de funcionamento
9	Execução de um controlo relacionado com o compasso de tempo
10	Execução de um controlo relacionado com a matriz do sequenciador
11	Inserção de um ícone no modo de funcionamento sem bolas

Os dados enviados pela aplicação são recebidos na rotina de interrupção da receção de dados da USART0, definida pelo vetor de interrupção *USART0_RX_vect*. Esta rotina é executada quando a *flag Receive Complete* (RXC) encontra-se ativa, significando que existem novos dados por ler no *buffer* de receção.

Para a leitura dos dados recebidos na USART0 foi definida uma estrutura. As linhas de código seguinte ilustram os parâmetros deste tipo de dados:

```
typedef struct USARTRX
{
  unsigned char init_byte;
  unsigned char type_byte;
  unsigned char arg_byte[20];
  unsigned char cnt_bytes;
  unsigned char flag_limit: 1;
  unsigned char status;
  unsigned char flag_error:   1;
} USARTRX_st;
```

A estrutura é composta por sete variáveis: *init_byte*, *type_byte* e *arg_byte* referem-se, respetivamente, a cada campo de uma trama (Figura 75); *cnt_bytes* efetua a contagem do número de *bytes* recebidos com o objetivo de construir a trama; *flag_limit* indica que todos os *bytes* da trama foram recebidos; *status* identifica se são ativadas *flags* de erro na receção de dados, ativando também a variável *flag_error* caso estes se verifiquem.

Com base no que foi explicado em relação à receção de dados na USART0 desenvolveu-se a programação da rotina de interrupção, que se encontra representada no fluxograma da Figura 80:

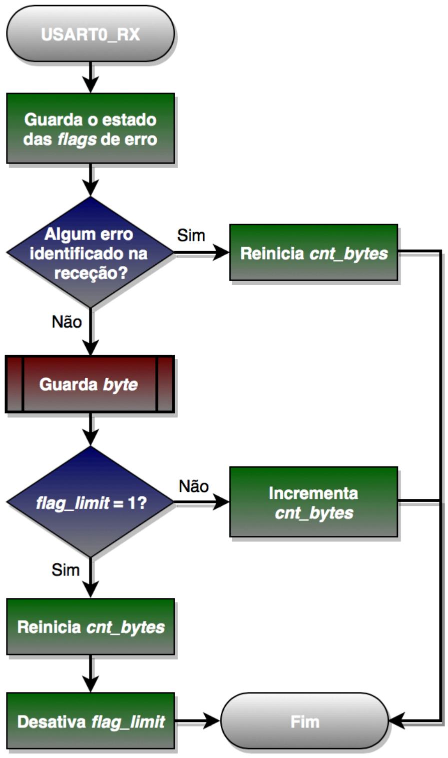


Figura 80 Fluxograma relativo à rotina de interrupção da receção de dados na USART0

Inicialmente, a rotina começa por guardar as *flags* de erro presentes no registo UCSR0A, e determinar se alguma delas se encontra ativa. Caso isto se verifique não é lido o valor presente no *buffer* de receção, sendo reinicializado o contador do número de *bytes* a zero (*cnt_bytes*), e terminada a interrupção.

A receção de dados é realizada no processo *Guarda byte* quando todas as *flags* de erro se encontram desativas. É determinado o valor do contador *cnt_bytes* para que a leitura do *buffer* de receção (UDR0) seja efetuada para uma variável de leitura (*init_byte*, *type_byte* ou *arg_byte*). Com o intuito de se perceber melhor este processo foi criado o fluxograma da Figura 81, que corresponde ao processo *Guarda Byte*:

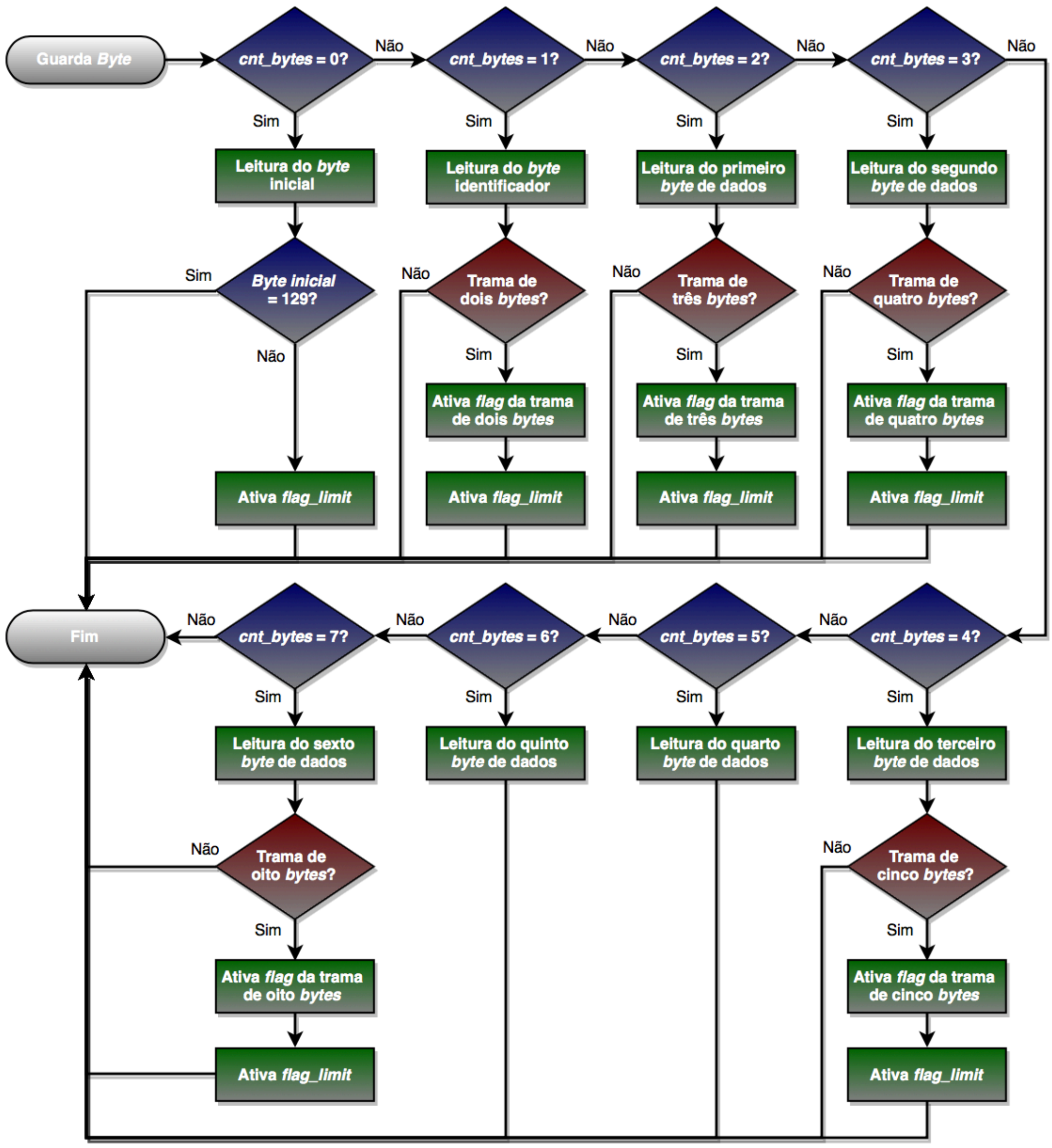


Figura 81 Fluxograma relativo ao processo de leitura de um *byte* na USART0

Como referido, a leitura de um *byte* na USART0 é efetuada conforme o valor do contador *cnt_bytes*, que é incrementado como analisado na Figura 80, com o intuito de construir a trama enviada pela aplicação Android. O tamanho desta varia consoante o número de *bytes* de dados, dado que os primeiros dois *bytes* correspondem, respetivamente, ao *byte* inicial e ao *byte* identificador da trama, e são comuns a todos os comandos.

O primeiro *byte* recebido no microcontrolador corresponde ao *byte* inicial da trama. O valor deste é guardado e comparado com o valor constante correto, que é igual a 129. Caso não esteja presente este valor significa que a trama não está a ser bem recebida, o que faz com que o microcontrolador continue à espera do *byte* inicial.

Assim, estando identificadas pelo segundo *byte* e sabendo o tamanho de todas as tramas enviadas pela aplicação para o microcontrolador, é possível ativar cada uma das *flags* referentes a cada trama quando efetuado a leitura do seu último *byte* de dados. A *flag* de limite da trama (*flag_limit*) também é ativada, indicando que todos os *bytes* foram lidos e reiniciando, portanto, o contador do número de *bytes* a zero.

A ativação das *flags* correspondentes a cada trama recebida permite realizar alterações de parâmetros do sistema. Este processo denomina-se *Execução de Tramas*, como ilustrado no fluxograma geral do *software* (Figura 59), e será analisado na subsecção seguinte.

5.4.12. EXECUÇÃO DE TRAMAS

O processo *Execução de Tramas* executa as tramas enviadas pela a aplicação Android para a rotina de interrupção da receção de dados na USART0. Quando é recebido o último *byte* da trama, a *flag* correspondente a esta é ativa, para que o microcontrolador execute o processo indicado pela a aplicação, sendo depois colocada com o valor 0, estando pronta para ser recebida novamente.

Conforme a Tabela 28, que descreve as tramas enviadas pela aplicação, existem dois tipos de comandos, sendo eles de pedido de informações e de alteração de parâmetros do sistema.

Em relação ao primeiro tipo, as *flags* de pedido de informações indicam ao microcontrolador que a aplicação Android pretende saber certos parâmetros do sistema, com o intuito de efetuar uma monitorização completa deste. As tramas de pedido recebidas na USART0 são:

- **Pedido geral** – quando recebida a trama significa que a aplicação estabeleceu ligação com o módulo Bluetooth associado à USART0. De seguida, é enviado para a aplicação todas as informações presentes no sistema, tais como as configurações associadas a cada coluna; a dimensão da matriz; o número de batimentos por minuto do *tempo*, assim como o seu estado e tipo; o modo de funcionamento; e o estado de cada LED.

- **Pedido de configurações associadas a cada coluna** – esta trama indica ao microcontrolador para transmitir para a aplicação Android as definições associadas a cada coluna do sistema, sendo elas o instrumento, a nota musical e a cor atribuída a cada LED que se encontre ativo na coluna.

As *flags* de alteração de parâmetros do sistema fazem com que o microcontrolador modifique os parâmetros do sistema, tais como o estado dos LED, as configurações de uma coluna, as caraterísticas do compasso de tempo, a dimensão e controlo da matriz, como também o modo de funcionamento do sistema.

No primeiro caso, quando recebida a trama completa da alteração de um pixel, é determinado a posição deste e modificado o seu estado. Só é possível efetuar este controlo quando o sistema se encontra no modo de funcionamento sem bolas.

As caraterísticas do compasso de tempo podem ser alteradas quando ativadas as *flags* consoante o comando recebido, sendo elas:

- **Número de batimentos por minuto** – altera o valor BPM do *tempo*, realizando também a sua conversão para ser implementado na rotina de interrupção do *Timer/Counter0*, com a utilização da função *Conversao_BPM*.

- **Cor da linha ativa** – modifica a cor da linha ativa do compasso de tempo, sendo enviado na trama três *bytes* correspondentes a cada componente da cor**.**

- **Tipo –** efetua uma alteração no tipo do compasso de tempo, do formato horizontal para diagonal, e vice-versa.

Além de enviadas tramas de alteração das caraterísticas do compasso, também pode ser enviado um controlo deste. Este controlo pode ser de três formas, descritas de seguida:

- **Parar** – quando recebido este controlo significa que o utilizador, através da aplicação Android, parou o *tempo*. No final do processo também é enviado uma trama do microcontrolador para a aplicação para efetuar a monitorização desta paragem.

- **Tocar** – inversamente ao primeiro controlo, este indica para o compasso de tempo continuar.

- **Iniciar** – o controlo de iniciar é enviado com o intuito de iniciar o *tempo* na primeira linha do sistema, consoante o tipo.

O microcontrolador aceita outra trama de controlo que diz respeito à matriz do sistema. Este comando é somente enviado no modo sem bolas, e permite manipular os LEDs em todas as direções, assim como desligá-los todos.

A última trama apresentada na Figura 79 (identificador 11) é recebida na USART0 do microcontrolador quando se pretende inserir um ícone no sistema. Um ícone representa uma imagem que pode ser visualizada através do controlo dos LEDs.

Cada ícone ocupa cinco linhas e cinco colunas, em que todas as posições são representadas numa variável de 25 *bits*. Assim, a trama é composta por 4 *bytes* que revelam o estado de cada posição do ícone, como também por argumentos que identificam qual a linha e coluna inicial que este é introduzido.

Durante a execução de algumas tramas recebidas, depois de efetuadas as alterações são guardadas na memória EEPROM. Este processo é executado através da implementação das funções *eeprom_update_byte*, para gravação de um único *byte*, e *eeprom_update_block*, que guarda dados de todas as posições de um vetor para a memória não volátil.

6. APLICAÇÃO ANDROID

Este capítulo é dedicado ao desenvolvimento da aplicação Android. Esta é responsável pela monitorização e pela transmissão de alterações a parâmetros no sistema para o microcontrolador. De forma a estruturar a programação, será descrito em primeiro lugar o ficheiro *Manifest*, onde são apresentadas as propriedades da aplicação e declaradas permissões, seguindo-se a descrição de todas atividades, que englobam todas as funcionalidades, através da implementação de ficheiros *Java* e *XML*.

Numa última secção é demonstrado o sistema implementado, como também são apresentados e discutidos os seus resultados em conjunto com a aplicação Android e o programa MIDI instalado no computador.

6.1. FICHEIRO *MANIFEST*

Neste ficheiro são definidas as propriedades da aplicação, sendo elas o seu nome, o seu ícone, as suas atividades e restantes funcionalidades. Além disto é determinado o nome da pasta onde são inseridos os ficheiros da aplicação.

É também no ficheiro *AndroidManifest.xml* que são declaradas as permissões que a aplicação necessita para aceder a áreas protegidas e utilizar seus recursos. Nesta foram definidas cinco permissões:

- ***BLUETOOTH*** – permitir a comunicação Bluetooth, isto é, solicitar ou aceitar uma ligação, assim como a transferência de dados;

- ***BLUETOOTH_ADMIN*** – iniciar a pesquisa de dispositivos ou alterar definições Bluetooth;

- ***ACCESS_COARSE_LOCATION*** – localizar dispositivos Bluetooth;

- ***READ_EXTERNAL_STORAGE*** – aceder a um ficheiro guardado no armazenamento externo;

- ***WRITE_EXTERNAL_STORAGE*** – criar um ficheiro no armazenamento externo.

6.2. ATIVIDADES

O diagrama de blocos ilustrado na Figura 82 demonstra o conjunto de atividades da aplicação Android. Esta começa através da atividade inicial onde é demonstrado um vídeo de funcionamento do sistema, que depois possibilita ao utilizador transitar para a atividade principal. Nesta, após ser efetuada a comunicação Bluetooth com o microcontrolador, é permitido visualizar e alterar parâmetros do sistema.

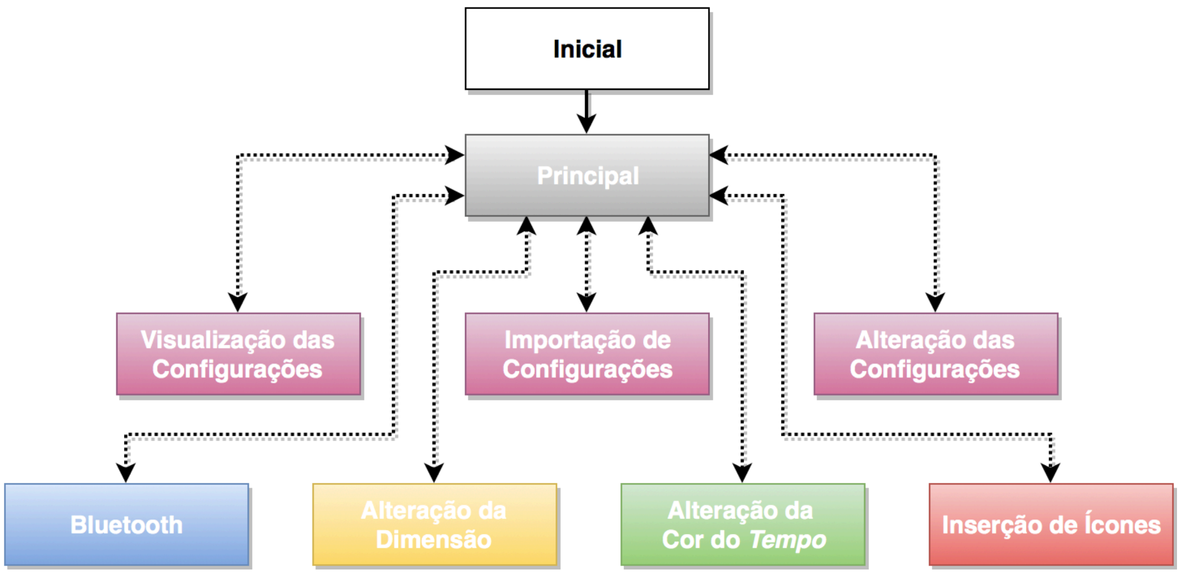


Figura 82 Diagrama de blocos da aplicação Android

6.2.1. ATIVIDADE INICIAL

A atividade inicial é a primeira a ser inicializada pela aplicação Android. Nesta é apresentado um vídeo de demonstração do sistema desenvolvido, assim como um texto de boas-vindas e um botão de início, através de uma animação.

A Figura 83 ilustra a atividade inicial no ecrã do *smartphone*:

Figura 83 Atividade inicial

O ficheiro XML para a atividade inicial define todos os componentes apresentados no ecrã na inicialização da aplicação, sendo eles: os elementos de texto (*TextView*), utilizados no texto de boas-vindas; o elemento de vídeo (*VideoView*), inserido na demonstração do vídeo de demonstração do sistema; o elemento de botão (*Button*), empregado pelo utilizador na passagem da atividade inicial para a principal.

O utilizador ao pressionar o botão *Iniciar* presente na atividade inicial (Figura 83) transita para a atividade principal. Esta ação é especificada pelo evento *onClick* no objeto do botão, que após efetuar o desaparecimento do mesmo e do objeto do texto de boas-vindas, cria uma *Intent*. Neste caso, a *Intent* é implementada para iniciar outra atividade.

O código seguinte demonstra como é realizada a transição da atividade inicial para a principal, através da *Intent startMain*:

```
Intent startMain = new Intent(this, MainActivity.class);
startActivity(startMain);
```

O construtor da *Intent startMain* é composto por dois parâmetros, em que o primeiro se refere à atividade presente no ecrã do *smartphone*, e o segundo à atividade que se pretende iniciar. Logo, através do método *startActivity*, é executada a mudança da atividade inicial para a principal, denominada *MainActivity*, sendo alterada a visualização da aplicação no ecrã.

6.2.2. ATIVIDADE PRINCIPAL

A atividade principal na aplicação Android é responsável por realizar a comunicação Bluetooth, assim como controlar todos os parâmetros do sistema, como visualizado no diagrama de blocos da Figura 82. Assim, esta efetua a monitorização e alteração de configurações no microcontrolador.

Igual às outras atividades, ela é composta por dois ficheiros: uma classe Java denominada *MainActivity*, onde são executados os métodos referentes à atividade, e um ficheiro *activity_main*, que trata da representação da mesma no ecrã do *smartphone*.

A Figura 84 ilustra como é constituída a atividade principal:

Figura 84 Atividade principal

Como visualizado na Figura 84, a atividade principal da aplicação é dividida em cinco áreas, descritas na Tabela 29:

Tabela 29 Áreas da atividade principal

Id	Área	Descrição
1	**Barra de Ações**	Área que permite visualizar e alterar as configurações de cada coluna, alterar a dimensão da matriz, importar ficheiros de configuração e informar o utilizador sobre como usufruir da aplicação.
2	**Controlo das Posições**	Área que permite monitorizar a cor de cada LED, como também controlá-los no modo de funcionamento sem bolas.
3	**Controlo do *Tempo***	Área que permite continuar, parar ou inicializar o compasso de tempo, como também visualizar as suas caraterísticas, tais como a cor e o número de batimentos por minuto.
4	**Comunicação Bluetooth**	Área que permite realizar a comunicação com o módulo Bluetooth associado ao microcontrolador, para transmissão e receção de tramas.
5	**Alteração do Modo e Controlo da Matriz**	Área que permite variar entre o modo com e sem bolas, e controlar o sequenciador.

Barra de Ações

Na barra de ações, o utilizador tem à disposição três botões de ação visíveis, mais dois escondidos que representam a importação de ficheiros de configuração e a secção *Sobre* da aplicação. Esta barra é definida num recurso XML, em que cada botão é criado com a *tag* *<item>*.

A barra de ações implementada na atividade principal é ilustrada e legendada na Figura 85:

Figura 85 Barra de ações da atividade principal

Como apresentado na Figura 85, a barra de ações é composta por cinco itens:

- **Retroceder** – Volta atrás na aplicação, isto é, retorna à atividade inicial onde é demonstrado o vídeo de funcionamento do sistema.

- **Info** – Visualização de uma tabela que mostra as configurações associadas a cada coluna, sendo elas a cor dos seus LEDs, o instrumento e a nota musical.

- **Dimensão** – Permite ao utilizador da aplicação alterar a dimensão da matriz do sistema, ou seja, modificar quantas linhas e colunas estão ativas.

- **Configurações** – Área onde são alteradas as configurações associadas a cada coluna.

- **Mais** – Oculta os itens de importação de ficheiros de configuração e *Sobre*, em que este último é selecionado quando o utilizador necessita de ajuda na utilização da aplicação.

O utilizador ao premir um botão de ação invoca o evento *onOptionsItemSelected*, que irá executar a programação desenvolvida para o botão, através do método denominado *getItemId*, que identifica qual foi o item selecionado com base no seu identificador. O seguinte excerto de código apresenta a programação do evento:

```
1 | public boolean onOptionsItemSelected(MenuItem item) {
2 |   switch(item.getItemId()) {
3 |     case android.R.id.home:
4 |       finish();
5 |       break;
  |
6 |     case R.id.action_info:
7 |       (...)
8 |       break;
  |
9 |     case R.id.action_settings:
10|       (...)
11|       break;
  |
12|     case R.id.action_import:
13|       (...)
14|       break;
  |
15|     case R.id.action_about:
16|       (...)
17|       break;
18|   }
19| }
```

Como demonstrado, é associado a cada botão de ação um identificador, que faz com que o evento, através do método *getItemId* (linha 2), determine qual foi premido. Assim, implementa-se um *switch case* que permite efetuar o pretendido para cada botão de ação.

Em relação aos itens *Dimensão*, *Configurações* e *Importação de Ficheiros* (oculto na secção *Mais*), é implementada uma *Intent* com o objetivo de iniciar novas atividades, que irão retornar resultados. Para explicar melhor este processo, são apresentadas as seguintes linhas de código que demonstram a programação relativa ao botão de ação *Dimensão*:

```
1 | Intent dimension_intent = new Intent(MainActivity.this,
  | DimensionActivity.class);
2 | startActivityForResult(dimension_intent, REQUEST_DIMENSION);
  |
3 | protected void onActivityResult(int requestCode, int
  | resultCode, Intent data) {
4 |   switch(requestCode) {
5 |     case REQUEST_DIMENSION:
6 |       if(resultCode == RESULT_OK) {
7 |         (...)
8 |       }
9 |       break;
10|   }
11| }
```

Em relação às linhas 1 e 2, estas são implementadas quando o botão de ação *Dimensão* é premido. Na linha 2, é implementado o método *startActivityForResult*, que recebe dois parâmetros de entrada, em que o primeiro (*dimension_intent*) diz respeito à atividade que irá ser iniciada (*DimensionActivity*), e o segundo a um valor inteiro que identifica o pedido (*REQUEST_DIMENSION*), para que possa ser analisado o resultado obtido dessa atividade.

Neste caso, o evento *onActivityResult* é invocado quando *DimensionActivity* termina. A variável *requestCode* permite descobrir a origem do resultado, correspondendo a *REQUEST_DIMENSION*, enquanto que *resultCode* indica uma alteração na dimensão (*RESULT_OK*), ou que o utilizador não realizou mudanças (*RESULT_CANCELED*). Por último, a *Intent data* é implementada para transmitir os dados do resultado.

Controlo das Posições

A segunda área da atividade principal diz respeito à disposição das posições no sistema. Cada posição do sequenciador é implementada no ecrã do *smartphone* pelo elemento *ImageButton*, que exibe um botão com uma imagem. A imagem implementada é um recurso XML que define um círculo como formato geométrico, e que pode assumir qualquer cor do sistema.

A mudança de cor de cada círculo é executada através da implementação de uma lista de níveis no recurso XML, que gere vários desenháveis alternativos aos quais são atribuídos valores máximos. Assim, foram criados dentro da lista, definida pela *tag* *<level-list>*, 36 círculos, cada um com uma cor do sistema. Na classe Java, é denominado *setImageLevel* o método que permite designar para cada botão de imagem o nível pretendido, isto é, a cor desejada.

Esta área além de ser utilizada para monitorização de cada posição do sistema, também permite que o utilizador ative ou não estas no modo de funcionamento sem bolas, através do *click* nos elementos *ImageButton*. Logo, quando presente este modo no sistema, é enviada uma trama de envio de alteração de um pixel, de acordo com o elemento pressionado.

Os botões de imagem relativos à posição de cada LED são inseridos no método *setOnClickListener*, para que quando for selecionado um deles seja invocado o evento *onClick*, cujo seu funcionamento é apresentado no seguinte extrato de código:

```
1 | public void onClick(View v) {
2 |   for (int i = 0; i < 12; i++) {
3 |     for (int j = 0; j < 12; j++) {
4 |       int id = getResources().getIdentifier("circle_" + i +
  |       "_" + j, "id", getPackageName());
5 |       if (v.getId() == id) {
6 |         if(mode) {
7 |           // Envio da trama de alteração de um pixel
8 |         }
9 |         else {
10|           // Alterar para modo sem bolas
11|         }
12|       }
13|     }
14|   }
15| }
```

No interior do evento, são executados dois ciclos *for* (linhas 2 e 3) que percorrem todos os botões de imagem, de forma a determinar qual deles foi premido. Na linha 5 é verificado o identificador do botão, com o intuito de descobrir a posição que se encontra.

Ao ser encontrada a posição do botão, é verificado se o modo de funcionamento presente no sistema é o modo sem bolas (linha 6). Caso seja confirmado este modo, é enviada para o microcontrolador uma trama de alteração do LED referente à posição do elemento pressionado. A trama efetua um *toggle* na posição enviada, como visualizado no seu cabeçalho na Figura 79, cujo seu identificador tem valor 2.

Caso esteja presente no sistema o modo com bolas, é apresentado um *popup* através da classe *Toast*, que informa o utilizador que tem de realizar a alteração do modo para o sem bolas.

Controlo do Tempo

Nesta área da atividade principal, o utilizador controla todas as caraterísticas relacionadas com o compasso de tempo, como visualizado na Figura 86:

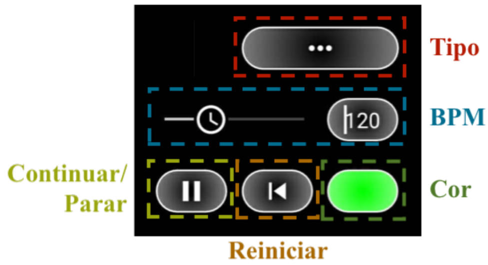


Figura 86 Controlo do *tempo* na atividade principal

No topo da área de controlo do *tempo* está presente o botão *Tipo* (Figura 86). Este permite escolher qual o tipo do compasso de tempo pretendido no sistema, podendo optar pelo tipo horizontal ou diagonal.

A Figura 87 ilustra a imagem presente no botão *Tipo* para cada formato do *tempo*:

Figura 87 Tipo horizontal (a); diagonal (b)

A Figura 87 a) indica que o formato do compasso de tempo presente no sistema é o horizontal, enquanto que o outro, visualizado na alínea b), indica o tipo diagonal.

A área BPM apresentada na Figura 86 representa o número de batimentos por minuto do *tempo*. Esta zona é composta por um elemento *SeekBar* que possibilita ao utilizador variar esta caraterística entre 60 e 240 BPM. Quando a barra deixa de ser movida é invocado o evento *onStopTrackingTouch*, onde é transmitida a trama de alteração do número BPM (identificador 3 – Figura 79).

A alteração de cor da linha ativa é definida a partir de um botão de imagem denominado *Cor*, representado na Figura 86. Este botão assume a cor atual no *tempo* e inicia a *Atividade de Alteração da Linha Ativa do Tempo*, que é analisada mais adiante na secção 6.2.6.

Além das caraterísticas descritas anteriormente (*Tipo*, *BPM*, *Cor*), a área de controlo do *tempo* é também composta pelos botões *Continuar/Parar* e *Reiniciar*. Estes são utilizados para continuar, parar ou reinicializar o compasso de tempo, em que as duas primeiras ações são realizadas pelo botão *Continuar/Parar*, e a última por *Reiniciar*.

A Figura 88 indica ao utilizador qual o estado que se pretende impor no compasso de tempo, através do botão de imagem *Continuar/Parar*:

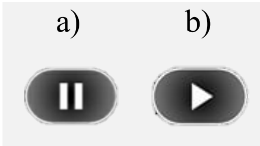


Figura 88 Parar (a); continuar (b)

A Figura 88 a) indica que *tempo* está em andamento, o que significa que quando premido o botão *Continuar/Parar* é dada uma ordem de paragem ao compasso. Assim, a imagem deste é alterada para a apresentada na alínea b), que mostra que o compasso de tempo está parado.

Quando o botão *Continuar/Parar* ou *Reiniciar* é premido, é transmitida uma trama de controlo para o microcontrolador, cujo seu identificador tem valor 9 (Tabela 28). Esta, além do seu *byte* inicial e identificador, é composta somente por um argumento, em que a sua gama de valores está compreendida entre 1 e 3. O valor 1 corresponde à paragem do *tempo*, o 2 continua-o e o 3 reinicia-o na primeira linha.

Comunicação Bluetooth

É na atividade principal que é realizada a comunicação Bluetooth com o microcontrolador, sendo também gerida a mesma para obtenção de informações do sistema, através da receção de tramas. Esta é executada através de um objeto *ImageButton*, que assume três estados inseridos numa lista de níveis definida num recurso XML. Na Figura 89 são ilustradas todas as situações que o botão Bluetooth aceita:

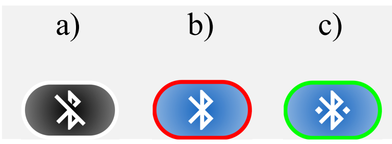


Figura 89 Bluetooth desligado (a); ligado sem conexão (b); ligado com conexão (c)

O primeiro estado (Figura 89 a)) indica que o Bluetooth se encontra desligado, enquanto que os restantes revelam que o Bluetooth está ligado. Porém, na alínea b) é demonstrado que o adaptador Bluetooth do *smartphone* encontra-se ligado, mas sem ligação a um dispositivo remoto. Já o último estado, representado na alínea c), mostra que o adaptador Bluetooth está ligado e estabelece uma ligação com um dispositivo.

Para a realização de ligações Bluetooth é necessário utilizar a Android Bluetooth API, que permite executar as operações com maior consumo de energia, tais como *streaming* e a comunicação entre dispositivos Android. Estas são [39]:

- Procurar dispositivos Bluetooth;
- Consultar o adaptador Bluetooth para verificação da existência de dispositivos Bluetooth emparelhados;
- Conexão a outros dispositivos através da descoberta de serviços;
- Transferência de dados para outros dispositivos;

As Bluetooth APIs são disponibilizadas no pacote *android.bluetooth*, sendo implementadas na programação da aplicação, as classes descritas na Tabela 30:

Tabela 30 Classes Bluetooth

Classe	Descrição
BluetoothAdapter	Representa o adaptador local, sendo utilizado para descoberta de outros dispositivos Bluetooth, consulta de dispositivos emparelhados, assim como iniciar um *BluetoothDevice* utilizando um endereço MAC conhecido.
BluetoothDevice	Representa um dispositivo remoto Bluetooth, sendo utilizado para solicitar uma ligação a um dispositivo remoto através de um *BluetoothSocket*, ou consultar informações sobre o dispositivo, tais como o nome, o endereço, a classe e o estado da ligação.
BluetoothSocket	Representa a interface de um *socket* Bluetooth, sendo o ponto de ligação que permite à aplicação trocar dados com outro dispositivo Bluetooth.

Inicialmente, a atividade principal determina se o *smartphone* suporta Bluetooth através da chamada do método *getDefaultAdapter* da classe *BluetoothAdapter*, como demonstrado em seguida:

```
BluetoothAdapter mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter == null) {
  // o smartphone não suporta Bluetooth
}
```

De seguida, é demonstrado no excerto de código o que acontece quando o botão *Bluetooth* é premido:

```
1 | if (!mBluetoothAdapter.isEnabled()) {
2 |   Intent enableBT = new
  |   Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
3 |   startActivityForResult(enableBT, REQUEST_ENABLE_BT);
4 | }
5 | else {
6 |   mBluetoothAdapter.disable();
7 | }
```

Como evidenciado na linha 1, caso o método *isEnabled* retorne o valor *true*, significa que o Bluetooth está ativo, sendo assim desligado com a implementação do método *disable* no objeto *mBluetoothAdapter*. Caso contrário, o Bluetooth está desativo e é solicitada a sua ativação com a chamada do método *startActivityForResult*. Neste é implementada uma *Intent* de ação, denominada *ACTION_REQUEST_ENABLE*, que apresenta uma caixa de diálogo a perguntar ao utilizador se permite a ativação do Bluetooth (Figura 90).

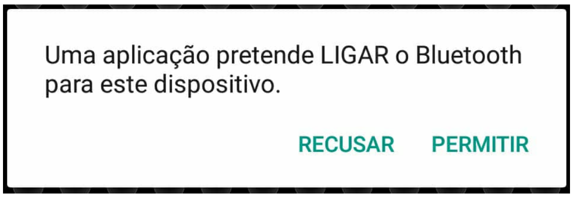


Figura 90 Caixa de diálogo de ativação do Bluetooth

Além de ser inserida a *Intent enableBT* no método *startActivityForResult* é também introduzida a constante *REQUEST_ENABLE_BT* (linha 3). Esta indica o valor no método *onActivityResult*, para onde é enviado o resultado da escolha do utilizador na caixa de diálogo da Figura 90.

Se a ativação do Bluetooth for bem-sucedida, isto é, caso seja premido o botão *Permitir* representado na Figura 90, é iniciada a *Atividade Bluetooth* (secção 6.2.3), onde será escolhido o dispositivo que se pretende estabelecer a ligação, sendo ele o módulo Bluetooth HC-06. Sendo este selecionado, é finalizada a *Atividade Bluetooth* e obtido o endereço físico do dispositivo para que seja estabelecida a ligação entre a aplicação e o sequenciador.

O estabelecimento de ligação da aplicação ao módulo Bluetooth HC-06 é realizado pela classe Java denominada *BluetoothConnection*, que é analisada e explicada de seguida.

Classe BluetoothConnection

A classe *BluetoothConnection* é responsável pelo estabelecimento de ligação entre a aplicação e o sistema, como também pela gestão da conexão entre os dois após serem conectados com sucesso.

A comunicação Bluetooth é estabelecida entre dois dispositivos através de uma *Thread* denominada *ConnectThread*, que pode ser visualizada nas seguintes linhas de código:

```
1 | private class ConnectThread extends Thread {
2 |   private final BluetoothSocket mmSocket;
3 |   private final BluetoothDevice mmDevice;
4 |   private final UUID mmUUID;
  |
5 |   public ConnectThread(BluetoothDevice device, UUID
  |     device_uuid)
6 |   {
7 |     BluetoothSocket tmp = null;
8 |     mmDevice = device;
9 |     mmUUID = device_uuid;
  |
  |     // Obtenção de um BluetoothSocket para ser estabelecida
  |     uma ligação ao dispositivo Bluetooth
10|     try {
11|       tmp =
  |         device.createRfcommSocketToServiceRecord(device_uuid);
12|     } catch (IOException e) { }
  |
13|     mmSocket = tmp;
14|   }
```

Na *ConnectThread* é obtido e inicializado um *BluetoothSocket* com a chamada do método *createRfcommSocketToServiceRecord* (linha 11). Este requer como parâmetro um *Universally Unique IDentifier* (UUID), representado por 128 *bits*, que é utilizado para identificar um serviço particular e um perfil correspondente fornecido por um dispositivo Bluetooth.

De seguida é iniciada a conexão com o dispositivo, como apresentado no excerto de código:

```
1 | public void run() {
  |   // Cancela descoberta de outros dispositivos
2 |    mBluetoothAdapter.cancelDiscovery();
  |
3 |    try {
  |    // Ligação ao dispositivo
4 |      mmSocket.connect();
5 |    } catch (IOException connectException) {
  |      // Impossível efetuar a ligação ao dispositivo
6 |      try {
7 |        mmSocket.close();
8 |      } catch (IOException closeException) { }
  |
9 |      connectionFailed(); // Falha na conexão ao dispositivo
10|      return;
11|    }
  |
  |    // Gestão da ligação criada
12|    manageConnectedSocket(mmSocket);
13| }
```

A conexão com o dispositivo é iniciada com o método *connect* (linha 4), sendo cancelada a descoberta de outros, dado que atrasa a ligação (linha 2). Caso ocorra uma falha na ligação ou o método esgote o tempo limite (12 segundos), é fechado o *BluetoothSocket*, como mostrado na linha 7, e enviada uma mensagem para o *Handler* da atividade principal, a indicar que houve uma falha na conexão ao dispositivo (linha 9).

Estabelecida a ligação com sucesso, com a chamada de *manageConnectedSocket* (linha 12) é iniciada uma segunda *Thread* designada *ConnectedThread*, que é definida inicialmente da seguinte forma:

```
private class ConnectedThread extends Thread {
  private final BluetoothSocket mmSocket;
  private final InputStream mmInStream;
  private final OutputStream mmOutStream;

  public ConnectedThread(BluetoothSocket socket) {
    mmSocket = socket;
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    // Obtenção do InputStream e OutputStream
    try {
      tmpIn = socket.getInputStream();
      tmpOut = socket.getOutputStream();
    } catch (IOException e) { }
    mmInStream = tmpIn;
    mmOutStream = tmpOut;
  }
```

É na *ConnectedThread* que são partilhados os dados entre a aplicação e o sistema pela obtenção de objetos *InputStream* e *OutputStream*, através dos métodos *getInputStream* e *getOutputStream*, respetivamente, como descritos nas linhas 10 e 11.

De seguida, são explicados os métodos de leitura e escrita da comunicação Bluetooth. O código seguinte apresenta o funcionamento em relação ao primeiro:

```
1  | public void run() {
2  |   byte[] buffer = new byte[1]; // buffer de receção
3  |   int bytes = 0; // número de bytes obtidos pelo método read
4  |   int dataReceived = 0; // conversão do byte do buffer
   |
5  |   while (true) {
6  |     try {
7  |       // Leitura de dados do InputStream
8  |       bytes = mmInStream.read(buffer);
9  |       dataReceived = buffer[0] & 0xFF;
   |       // Envio do byte recebido para a atividade principal
10 |       mHandler.obtainMessage(Constants.READ, bytes, -1,
   |         dataReceived).sendToTarget();
   |
11 |     } catch (IOException e) {
   |       // Conexão perdida ao dispositivo
12 |       connectionLost();
13 |       break;
14 |     }
15 |   }
16 | }
```

A leitura de dados é realizada chamando o método *read* no objeto *InputStream* (linha 8). Como o microcontrolador envia um *byte* de cada vez, é implementado um *buffer* de receção de tamanho 1, do tipo de variável *byte*.

Em programação Android, os valores do tipo *byte* variam entre -128 e 127. Portanto, como visualizado na linha 9, é necessário transformar o valor recebido num inteiro que varie entre 0 e 255, dado que são os valores enviados pelo microcontrolador.

Por último, é enviada uma mensagem do tipo *READ* para o *Handler* da atividade principal com o valor do *byte* recebido, através da implementação do método *obtainMessage* na linha 10.

O método *connectionLost* (linha 12) é invocado quando a ligação ao dispositivo é perdida, sendo enviado uma mensagem do tipo *STATE_CHANGED* para o objeto *Handler* da atividade principal, com o objetivo de informar o utilizador.

Em relação à transmissão de dados, o excerto de código que demonstra o seu funcionamento é o seguinte:

```
1 | public void write(int byteSent) {
2 |   try {
3 |     mmOutStream.write(byteSent);
  |     // Informa a atividade principal que foi enviado um byte
4 |     mHandler.obtainMessage(Constants.WRITE, -1, -1,
        byteSent).sendToTarget();
5 |   } catch (IOException e) { }
6 | }
```

No início do método de envio de dados é chamado o método *write* (linha 3) no objeto *OutputStream*, em que é enviado somente um *byte*. O envio de um *byte* para a USART0 do microcontrolador é informado na atividade principal através de uma mensagem do tipo *WRITE*, que é transferida para o seu objeto *Handler* através do método *obtainMessage* (linha 4).

Ao longo da classe *BluetoothConnection* fala-se de um objeto *Handler* implementado na atividade principal, que se carateriza por ser um elemento manipulador que permite enviar ou receber objetos *Message* numa *thread* de execução. Estes definem uma mensagem contendo um pacote de dados, e são processados pelo evento *handleMessage* do objeto criado a partir da classe *Handler*.

O evento *handleMessage* recebe como argumento de entrada um objeto *Message*, que é constituído por um campo que serve como identificador da mensagem (*what*), um objeto que envia o pretendido (*obj*), e por dois argumentos que são utilizados para guardar valores inteiros (*arg1* e *arg2*). Assim, o método admite três tipos de mensagens, identificadas por *what*, sendo elas:

- ***STATE_CHANGED*** – identifica o estado da ligação Bluetooth através do argumento *arg1*.
- ***READ*** – reconhece que foi recebido um *byte*, pertencente a uma trama enviada pelo microcontrolador, que é guardado no parâmetro *obj* da mensagem.
- ***WRITE*** – identifica o envio de informação para o microcontrolador, através do argumento *obj* da mensagem. Este tipo de mensagem foi implementado somente para testes na aplicação, para verificar se os dados estavam a ser corretamente enviados.

Na Tabela 31 estão apresentados os valores que permitem identificar o estado da ligação Bluetooth numa mensagem do tipo *STATE_CHANGED*:

Tabela 31 Estado da ligação Bluetooth

Constante	Valor	Descrição
STATE_DISCONNECTED	0	Identifica uma ligação perdida a um dispositivo ou uma falha a tentar conectar-se com este.
STATE_CONNECTING	1	Identifica uma tentativa de estabelecimento de uma ligação com um dispositivo.
STATE_CONNECTED	2	Identifica uma ligação bem-sucedida a um dispositivo.

Cada *byte* recebido na aplicação é inserido numa mensagem do tipo *READ*, que é transmitida para o evento *handleMessage* do *Handler*. Dentro deste é realizada a construção da trama para que sejam efetuadas as modificações na aplicação Android.

O fluxograma da Figura 91 demonstra como é realizada a leitura de um *byte*, isto é, como é recebida uma mensagem do tipo *READ* no evento *handleMessage*:

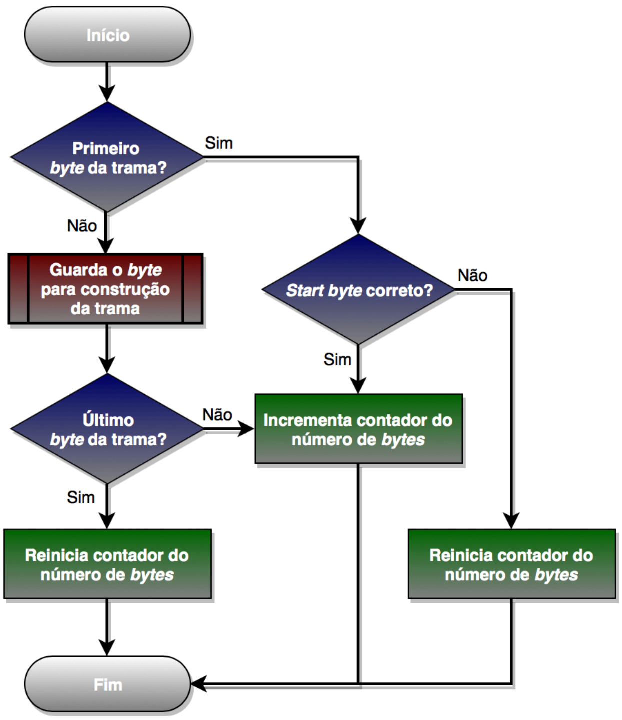


Figura 91 Fluxograma relativo à receção de dados na atividade principal

Como referido, o microcontrolador só envia um *byte* de cada vez, o que faz com que a atividade principal tenha de construir as tramas a partir do *byte* inicial, agrupando assim todos os *bytes*. Existem ao todo oito tramas que são enviadas para a aplicação (Tabela 25), sendo as principais a informação sobre um LED e as configurações associadas a cada coluna.

Quando recebido na aplicação a trama relativa à informação sobre um pixel, é executada uma mudança de cor num elemento *ImageButton* localizado na área *Controlo dos LEDs*. Este é modificado através do método *setImageLevel*, que implementa a cor definida nos últimos três *bytes* da trama, que tem como identificador valor 0 – Figura 76.

A trama que envia as configurações associadas a uma coluna determina a cor que os LEDs assumem quando a respetiva posição se encontra ativa, assim como o instrumento e a nota musical MIDI que produzem quando ativos.

Quando recebida a trama das configurações associadas à última coluna do sequenciador, é efetuado um processo de verificação de erros, em que o seu funcionamento é apresentado no fluxograma da Figura 92. Neste é determinado se as configurações de todas as colunas foram enviadas com sucesso do microcontrolador para a aplicação Android.

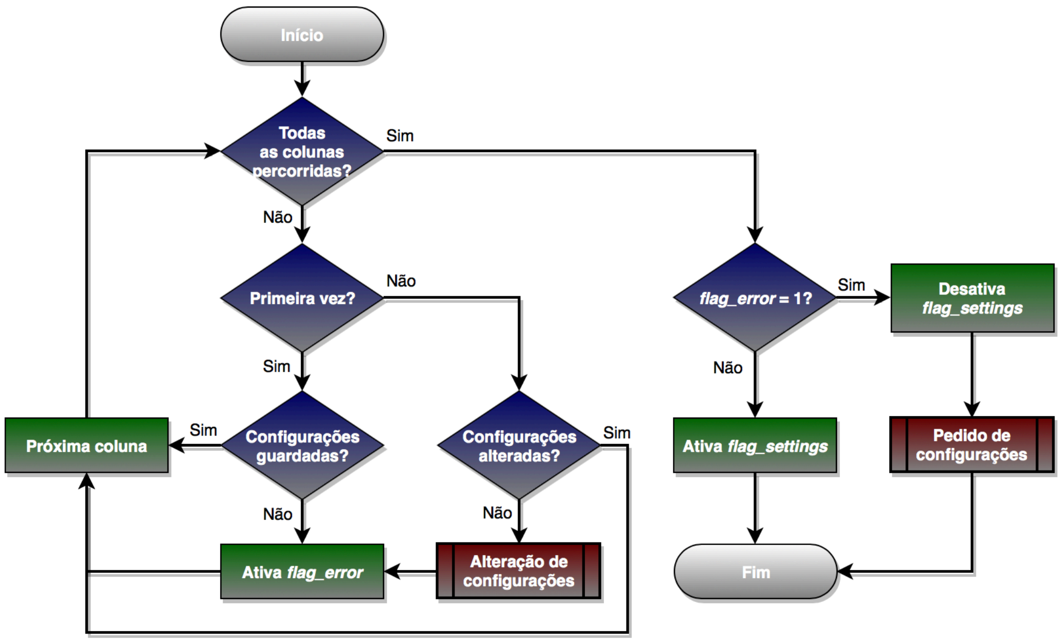


Figura 92 Verificação de erros nas configurações de cada coluna

O processo de verificação de erros começa por determinar se é a primeira vez que a aplicação Android está a receber as configurações de cada coluna. Assim, pode-se dizer que o processo é chamado em duas ocasiões:

- Após ser efetuada com sucesso a ligação Bluetooth;
- Alteração de uma configuração numa coluna.

No primeiro caso são verificadas se todas as configurações foram guardadas com sucesso na aplicação. Em relação ao segundo, é comprovado se os valores pretendidos pelo utilizador foram devidamente alterados.

Em ambos os contextos, quando uma configuração não está correta é ativada uma *flag* de erro, denominada *flag_error*. Neste estado, é enviada para o microcontrolador uma trama de pedido das configurações associadas a cada coluna (identificador 1 – Figura 79), com o objetivo de detetar se o erro foi corrigido, ou seja, se as configurações que estavam em falta foram recebidas com sucesso. Também é desativada a *flag_settings*, que impede o utilizador de aceder às áreas *Info* e *Configurações* da barra de ações da atividade principal (Figura 85).

No caso de todas as configurações serem recebidas com sucesso significa que a *flag_error* se encontra desativada, o que origina à ativação de *flag_settings* e, assim, o acesso às áreas *Info* e *Configurações*.

Alteração do Modo e Controlo da Matriz

A última área da atividade principal diz respeito à alteração do modo e ao controlo do sequenciador. Nesta, o utilizador altera o modo de funcionamento, assim como controla a matriz do sistema, quando presente no modo sem bolas. Na Figura 93 é visualizada a zona na atividade principal relativa a esta funcionalidade:

Figura 93 Alteração do modo e controlo da matriz na atividade principal

O botão *Modo* permite alterar o modo de funcionamento, em que a imagem no seu interior demonstra o modo presente no sistema. Assim, é implementado um elemento *ImageButton*, em que é modificada a imagem com recurso ao seu método *setImageResource*, que define o conteúdo do botão para os ícones ilustrados na Figura 94:

Figura 94 Modo com bolas (a); sem bolas (b)

O primeiro ícone representado na Figura 94 a) indica que o modo de funcionamento presente no sistema é o modo com bolas, enquanto que o outro, visualizado na alínea b), indica a utilização do modo sem bolas.

Quando premido o botão, é enviada uma trama de alteração do modo para o microcontrolador – *byte* identificador com valor 8 (Figura 79). Este ao receber o comando realiza a alteração, e envia outra trama com o identificador 7 (Figura 76), a indicar à aplicação Android que o modo foi alterado com sucesso.

Estando presente o modo sem bolas no sistema é permitido ao utilizador controlar cada posição do sequenciador a partir da aplicação. Este controlo diz respeito à movimentação do estado das posições para todas as direções, manipulando o sequenciador conforme demonstrado na área *Controlo*. O botão *Desativação* também possibilita o controlo do sequenciador no modo sem bolas, no qual desativa o estado de todas as posições.

Assim, a trama de envio do controlo da matriz é composta somente por um argumento, sem contar com o *byte* inicial e o identificador (valor 10), como apresentado na Figura 79. A Tabela 32 descreve cada valor que o argumento da trama pode obter:

Tabela 32 Valor do argumento na trama do controlo da matriz

Argumento	Descrição
1	Desloca o estado das posições uma linha acima.
2	Desloca o estado das posições uma linha abaixo.
3	Desloca o estado das posições uma coluna à esquerda.
4	Desloca o estado das posições uma coluna à direita.
5	Desativa o estado de todas as posições.

No modo sem bolas, também é permitido ao utilizador inserir ícones a partir do botão *Inserir*. Este inicia uma nova atividade na aplicação, denominada *Inserção de Ícones*, que será descrita na secção 6.2.8.

6.2.3. ATIVIDADE BLUETOOTH

A atividade Bluetooth é iniciada quando é ligado o adaptador ou é pressionado o botão de *Bluetooth* da atividade principal (Figura 84) por mais de dois segundos. A Figura 95 demonstra a representação desta atividade no ecrã do *smartphone*:

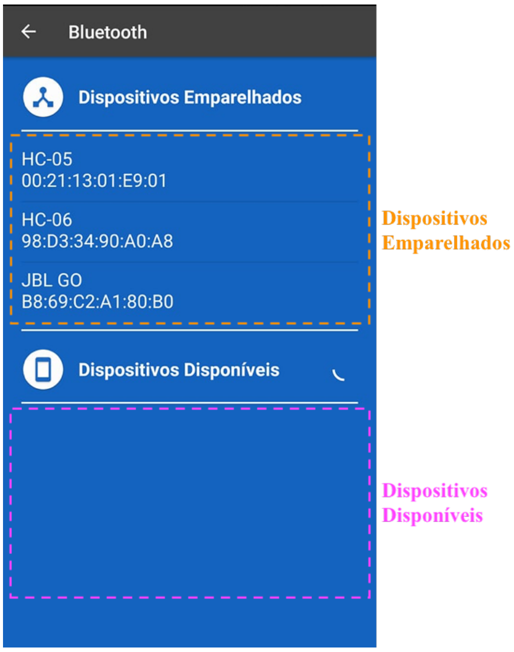


Figura 95 Atividade Bluetooth

O ficheiro XML relacionado com a atividade Bluetooth é responsável pela exibição de todos os elementos da Figura 95. Neste *layout* estão representados dois elementos do tipo *ListView*, que são implementados com o intuito de mostrar os dispositivos que já foram emparelhados com o adaptador Bluetooth, ou os equipamentos disponíveis nas redondezas para estabelecimento de ligação.

O excerto de código seguinte ilustra como é realizado todo o processo de consulta e exibição dos dispositivos emparelhados com o adaptador Bluetooth do *smartphone*. O *ArrayAdapter* utilizado para guardar o nome e o endereço do dispositivo é denominado *mPairedDevicesArrayAdapter*, sendo implementado no elemento *ListView* através do método *setAdapter*.

```
1 | Set<BluetoothDevice> paired_devices =
  | mBluetoothAdapter.getBondedDevices();
  |
  | // verifica se existem dispositivos emparelhados
2 | if(paired_devices.size() > 0) {
  |   // percorre todos os dispositivos emparelhados
3 |   for(BluetoothDevice device : paired_devices) {
4 |     String BTName = device.getName();
5 |     String BTAddress = device.getAddress();
  |     // adiciona o nome e endereço a um ArrayAdapter
6 |     mPairedDevicesArrayAdapter.add(BTName + "\n" + BTAddress);
7 |   }
8 | }
9 | else {
10|   mPairedDevicesArrayAdapter.add("No paired bluetooth devices
  |     found");
11|   Toast.makeText(getApplicationContext(), "No paired bluetooth
  |     devices found", Toast.LENGTH_SHORT).show();
12| }
```

Antes de ser realizada a descoberta de dispositivos nos arredores com o Bluetooth ligado, são consultados os dispositivos emparelhados, como representado na linha 1, para que seja determinado se o equipamento a que se pretende comunicar já é conhecido pelo adaptador Bluetooth. Para isso é chamado o método *getBondedDevices* do objeto *mBluetoothAdapter*, que retorna um conjunto de dispositivos emparelhados.

De seguida, é consultado o nome e endereço de cada dispositivo, nas linhas 4 e 5, para que sejam inseridos no objeto *mPairedDevicesArrayAdapter*, a partir do método *add* (linha 6).

Após consultados todos os dispositivos emparelhados, é realizado o processo de descoberta de outros dispositivos, num tempo aproximado de 12 segundos. Isto é implementado pelo método *startDiscovery* do adaptador Bluetooth (objeto *mBluetoothAdapter*), que retorna um valor booleano que indica se a descoberta foi iniciada corretamente.

A atividade Bluetooth termina quando o utilizador seleciona um dispositivo, quer esteja emparelhado ou não, ou quando volta à atividade principal. Ao ser selecionado um item, é determinado qual o endereço do dispositivo escolhido, sendo o valor deste retornado para a atividade principal.

6.2.4. ATIVIDADE DE ALTERAÇÃO DA DIMENSÃO

Quando premido o botão *Dimensão* presente na barra de ações da atividade principal (Figura 85), é chamada a *Atividade de Alteração da Dimensão*. Nesta, o utilizador define a dimensão do sequenciador, isto é, indica o número de linhas e colunas ativas no sistema. A Figura 96 ilustra o *layout* desta atividade:

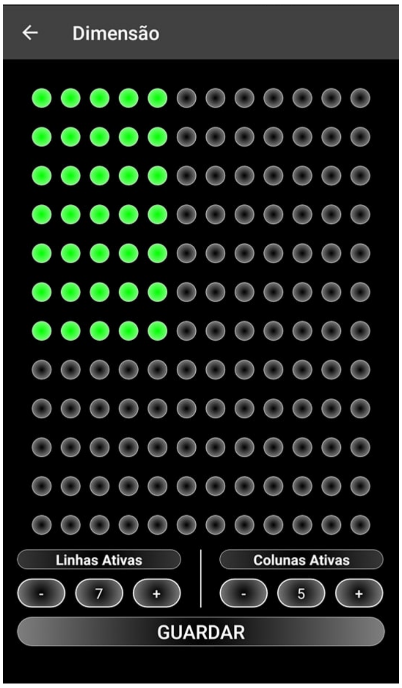


Figura 96 Atividade de alteração da dimensão do sequenciador

O utilizador pressiona um botão de imagem, que através da sua posição, define a dimensão do sequenciador, isto é, o número de linhas e colunas ativas no sistema. Por exemplo, e como demonstrado na Figura 96, caso seja pressionado o botão presente na linha 7 e na coluna 5, a dimensão será de um total de sete linhas e cinco colunas.

A dimensão do sequenciador também pode ser alterada premindo os botões '+' ou '-', que tanto aumentam como diminuem, respetivamente, o número de linhas ou de colunas ativas.

O botão *Guardar* quando clicado retorna a nova dimensão para a atividade principal, que é responsável por informar o sistema desta alteração. Assim, é realizado o envio da trama de alteração da dimensão do sequenciador para o microcontrolador, que tem como *byte* identificador o valor 7. Como indicado no seu cabeçalho (Figura 79), esta só aceita os valores de linha e coluna acima ou igual a 4 (dimensão mínima tem tamanho 4x4).

6.2.5. Atividade de Alteração das Configurações de cada Coluna

Uma das funcionalidades mais importantes da aplicação é a alteração das configurações associadas a cada coluna, sendo elas o instrumento, a nota musical e a cor dos LEDs quando ativos. A atividade relativa a esta é invocada através de um *click* no item *Configurações*, presente na barra de ações da atividade principal (Figura 85), após ter sido estabelecida a ligação Bluetooth com o sistema, e executada a leitura das configurações de todas as colunas (ativação de *flag_settings*).

Para implementação da atividade de alteração das configurações de cada coluna foi selecionada a *Tabbed Activity*, presente na galeria de atividades do Android Studio. Este tipo de atividade proporciona ao utilizador uma navegação lateral entre ecrãs distintos, através do gesto de um dedo na horizontal.

A transição entre separadores é efetuada com a implementação do elemento *ViewPager* no *layout* da atividade. Neste elemento, cada página é representada através de um objeto da classe *FragmentPagerAdapter*. A este tipo de adaptador estão relacionados três métodos:

- ***getItem*** – retorna o *Fragment* relativo à posição do separador presente na atividade.
- ***getCount*** – retorna o número de separadores presente na atividade.
- ***getPageTitle*** – retorna o texto utilizado para descrever cada separador presente na atividade.

Também é inserido no ficheiro XML um elemento denominado *AppBarLayout*, que junta na vertical a barra de ações com a barra de separadores, como visualizado na Figura 97:

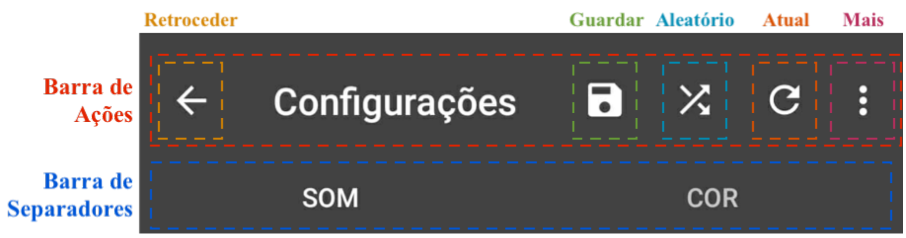


Figura 97 *AppBarLayout* da atividade de alteração das configurações de cada coluna

Como legendado na Figura 97, a barra de ações desta atividade é composta pelas funcionalidades de guardar as configurações escolhidas; atribuir configurações aleatoriamente; restabelecer as configurações iniciais; exportar para um ficheiro as configurações escolhidas (item ocultado na secção *Mais*). Também está presente o botão de regresso à atividade principal, sem mudança de nenhuma configuração.

O procedimento de guardar as configurações escolhidas é iniciado quando ocorre um *click* no botão de ação *Guardar*, através da comparação dos parâmetros atuais do sistema com os parâmetros escolhidos pelo o utilizador. Caso algum tenha sido alterado, é retornado o valor *RESULT_OK* para a atividade principal, que enviará as tramas de alteração de configurações de todas as colunas. Se nenhum parâmetro for diferente dos atuais, é retornado o valor *RETURN_CANCELED*, não sendo efetuada nenhuma modificação no sistema.

O botão de ação *Aleatório* atribui aleatoriamente configurações a cada coluna. Isto é realizado através do método *nextInt* de um objeto *Random*, que indica a posição num *string-array*, obtendo assim o valor do parâmetro. Por último, são chamados os métodos *setInstrument* e *setNote* do *Fragment* do primeiro separador, e *setColor* do segundo. Cada um destes será descrito mais à frente na análise ao respetivo separador.

O botão de ação *Atual* restabelece as configurações atuais de cada coluna, sendo semelhante ao de atribuição aleatória de parâmetros, dado que são implementados na mesma os métodos *setInstrument*, *setNote* e *setColor*. Porém, os valores passados como argumentos para estes métodos correspondem às configurações presentes no sistema.

O utilizador tem ainda disponível na secção *Mais* o botão de ação *Exportar Configurações*. Este permite exportar as configurações inseridas na atividade para um ficheiro. Ao ser selecionado, surge uma caixa de diálogo no ecrã do *smartphone*, ilustrada na Figura 98:

Figura 98 Caixa de diálogo relativa à exportação de configurações para um ficheiro

As configurações de todas as colunas são exportadas para um ficheiro do tipo CSV. Trata-se de um arquivo que apresenta valores de uma tabela na forma de linhas de texto simples, onde os valores são separados por vírgulas que representam cada coluna. A Tabela 33 representa as configurações implementadas em quatro colunas ativas:

Tabela 33 Configurações implementadas para sistema de quatro colunas ativas

Coluna	Cor	Instrumento	Nota
1	*#008080*	*Clarinet*	*C# (Octave 9)*
2	*#F5F5DC*	*Accordion*	*F# (Octave 2)*
3	*#0000FF*	*Whistle*	*C (Octave 7)*
4	*#4682B4*	*English Horn*	*C (Octave 8)*

Assim, o ficheiro CSV relativo à Tabela 33 é constituído pelos seguintes dados:

```
Coluna,Cor,Instrumento,Nota
1,#008080,Clarinet,C# (Octave 9)
2,#F5F5DC,Accordion,F# (Octave 2)
3,#0000FF,Whistle,C (Octave 7)
4,#4682B4,English Horn,C(Octave 8)
```

Todos os ficheiros de configuração são guardados numa pasta denominada *SettingsFiles*, que se encontra localizada no interior da pasta relativa à aplicação. Portanto, o arquivo CSV, com o nome inserido na caixa de diálogo da Figura 98, é criado através da inicialização de um objeto *File*, que recebe como argumentos a localização e o nome do ficheiro.

Após a criação do ficheiro é definido um objeto a partir da classe *FileOutputStream*, que é implementado na escrita de dados para o ficheiro CSV, referido pelo objeto *File*. As configurações são inseridas no arquivo através do método *write*, que envia um conjunto de *bytes* relativos a cada linha, e que termina com o caracter '\n' (indicação de uma nova linha). No final da inserção de todos os parâmetros no ficheiro é fechado o *FileOutputStream* com o método *close*, sendo guardadas todas as configurações.

Explicadas todas as funcionalidades executadas pelos botões presentes na barra de ações, são agora analisados os separadores (*Fragments*) desta atividade, denominados *Som* e *Cor*.

Separador SOM

No primeiro *Fragment* da atividade, o utilizador altera as configurações de cada coluna relativas ao som, sendo elas o instrumento e a nota musical, como visualizado na Figura 99:

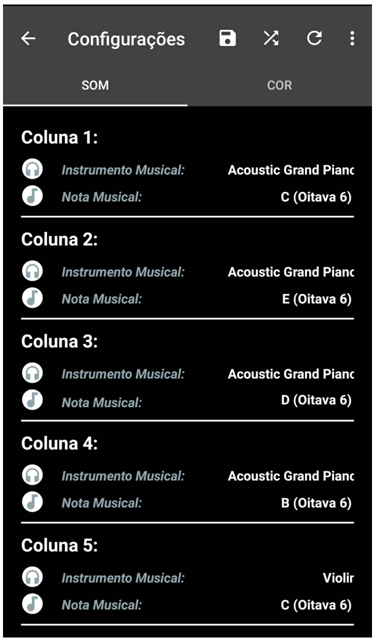


Figura 99 Separador Som da atividade de alteração das configurações de cada coluna

A modificação dos parâmetros do separador é realizada através de elementos do tipo *Spinner*. Estes são caraterizados por possibilitarem a seleção de um valor num conjunto. O objeto mostra a configuração presente, porém, quando premido ilustra um menu com todos os outros valores disponíveis.

Sempre que o utilizador escolhe um item no menu, o objeto *Spinner* invoca o evento *onItemSelected*. Dentro deste são percorridos todos os *Spinners* presentes no *Fragment*, determinando em qual deles foi alterado o parâmetro. Após ser descoberto o elemento selecionado, o *Fragment* envia para a sua atividade o instrumento ou a nota associada à coluna do *Spinner*.

A transmissão de informação entre o *Fragment* e a sua atividade é executada com a definição de uma *interface* na classe do fragmento e a sua implementação dentro da atividade. A definição da *interface* na classe relativa ao separador *Som* é realizada da seguinte maneira:

```
1 | interface SendData {
2 |   void SendInstrument(int column, String instrument);
3 |   void SendNote(int column, String note);
4 | }
```

No interior da *interface* são declarados os métodos *SendInstrument* e *SendNote* nas linhas 2 e 3. Estes são invocados dependendo se o utilizador seleciona um *Spinner* relativo ao instrumento ou nota de uma coluna, e têm como função informar a atividade quais as configurações a ser alteradas.

Assim, para serem recebidos do *Fragment* os parâmetros selecionados em cada elemento *Spinner*, a atividade que o invoca (*Alteração das Configurações de cada Coluna*) deve implementar a *interface SendData*.

Na classe do separador são definidos os métodos *setInstrument* e *setNote*, que são utilizados pelos botões *Aleatório* e *Atual* da barra de ações da atividade *Alteração das Configurações de cada Coluna*. Ambos os métodos, como demonstrado no código seguinte, recebem dois argumentos de entrada, sendo eles o instrumento ou a nota modificada, e a coluna que se pretende configurar o parâmetro.

```
1 | protected void setInstrument(String instrument, int column) {
2 |   SpinnerInstrument[column].setSelection(
  |   getIndex(SpinnerInstrument[column], instrument));
3 | }
  |
4 | protected void setNote(String note, int column) {
5 |   SpinnerNote[column].setSelection(
  |   getIndex(SpinnerNote[column], note));
6 | }
```

A alteração do item selecionado no *Spinner* é efetuada programaticamente através do método *setSelection* deste objeto (linhas 2 e 5), que obtém a posição do item. Como o valor passado pelos métodos da *interface SendData* é do tipo *string*, tornou-se necessário converter esse valor na posição que este se encontra no menu do *Spinner*, tendo sido desenvolvido o método *getIndex*.

```
1 | public int getIndex(Spinner mSpinner, String mString) {
2 |   int index = 0;
3 |   for(int i = 0; i < mSpinner.getCount(); i++) {
4 |     if(mSpinner.getItemAtPosition(i).toString().equals
  |     (mString))
5 |     {
6 |       index = i;
7 |       break;
8 |     }
9 |   }
10|   return index;
11| }
```

O método *getIndex* executa um ciclo *for* (linha 3) que percorre todas as posições do menu de um *Spinner* – método *getCount* – com o objetivo de encontrar a posição da *string* desejada. Assim, como visualizado na linha 4, a posição é determinada através da comparação de um valor *string* no *Spinner* inserido como primeiro argumento (método *getItemAtPosition*), com a variável *mString*.

Verificada a condição da linha 4, é retornado o valor da variável *índex* como demonstrado na linha 10, que corresponde ao valor da posição que *mString* se encontra no objeto *mSpinner*.

Separador COR

Neste separador são escolhidas as cores pretendidas na ativação dos LEDs de cada coluna. É apresentado no *Fragment* todas as cores possíveis, em que o utilizador visualiza a cor configurada para cada coluna através do ícone *check*. A mudança de coluna é realizada através dos botões '+' e '-', sendo representado o seu valor num elemento de texto (*TextView*) inserido entre estes. A Figura 100 ilustra o separador *Cor* presente na atividade de alteração das configurações associadas a cada coluna:

Figura 100 Separador Cor da atividade de alteração das configurações de cada coluna

No *layout* deste separador, a cada cor estão associados dois elementos: um *ImageButton*, que representa a cor, e uma *ImageView*, que tem como imagem o ícone *check*, e que identifica se a cor está selecionada ou não para uma dada coluna.

Em relação aos objetos *ImageButton*, estes são inseridos no evento *onClick*, que especifica a ação a ser executada quando premida uma cor. Neste é determinado qual a cor escolhida pelo utilizador para uma dada coluna, sendo visualizado o respetivo ícone *check*, e enviada a informação sobre a cor configurada para a atividade do *Fragment*.

Tal como o primeiro separador, para ser indicada a modificação do parâmetro da cor de uma coluna à sua atividade, é necessário a definição de uma *interface*, assim como a sua implementação na atividade. Dentro desta é declarado o método *SendColor*, que é utilizado para informar a atividade que uma dada coluna tem uma nova configuração da cor.

```
interface SendData {
  void SendColor(int column, String color);
}
```

Como descrito anteriormente, quando premido o botão de ação *Aleatório* ou *Atual*, a atividade executa o método *setColor*. Este recebe dois argumentos de entrada, em que o primeiro identifica a cor e o segundo a coluna onde é implementada a mudança.

6.2.6. Atividade de Alteração da Cor da Linha Ativa do *Tempo*

Esta atividade, como o próprio nome indica, permite que o utilizador escolha a cor para cada LED presente na linha ativa do compasso de tempo. A atividade é iniciada quando premido o botão *Cor*, visualizado na Figura 86. O *layout* desta atividade é visualizado na Figura 101:

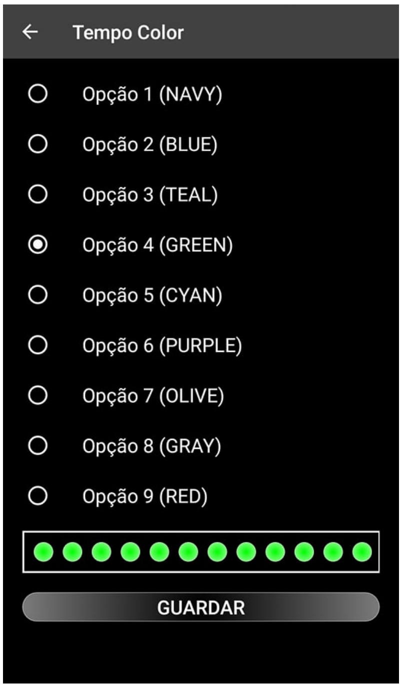


Figura 101 Atividade de alteração da cor da linha ativa do compasso de tempo

No ficheiro XML foram implementados objetos a partir da classe *RadioButton*, que possibilitam ao utilizador a escolha de uma opção de um conjunto. Estes objetos são inseridos dentro de um *RadioGroup*, que assegura que apenas um *RadioButton* é selecionado de cada vez. Também se encontram representados elementos *ImageView* que realizam a visualização da cor selecionada.

Quando um *RadioButton* é selecionado, é invocado o evento *onCheckedChanged*. Dentro do evento é determinado qual o botão que foi premido e ilustrado nos elementos *ImageView* a cor correspondente à seleção.

Modificada a cor, o utilizador realiza um *click* no botão *Guardar*, que tem como função terminar a atividade e retornar para a atividade principal o valor da cor da linha ativa do *tempo* alterada.

6.2.7. Atividade de Importação de Ficheiros de Configuração

Esta atividade é chamada quando premido o botão de ação *Importar Configurações*, que se encontra oculto na secção *Mais* da barra de ações da atividade principal. Nesta atividade é implementado um objeto *ListView* que apresenta ao utilizador todos os ficheiros de configuração, no formato CSV, que foram anteriormente criados. Como explicado, os arquivos CSV são guardados na pasta do aplicativo, que é denominada *SettingsFiles*.

A Figura 102 demonstra o *layout* da atividade de importação de ficheiros de configuração no ecrã do *smartphone*:

Figura 102 Atividade de importação de ficheiros de configuração

O aparecimento dos ficheiros de configuração na atividade é efetuado a partir da pasta onde se encontram guardados. Sabendo a localização desta, é possível obter o nome de todos os seus ficheiros, sendo ordenados por ordem alfabética para serem inseridos na *ListView*.

A escolha do ficheiro de configuração é executada com um *click* num item da *ListView*, que identifica o nome do arquivo através da posição que se encontra nesta, sendo este retornado para a atividade principal.

Na atividade principal são lidas as configurações presentes no ficheiro CSV de configuração escolhido, através da inicialização de um *File* que cria uma nova instância do objeto, a partir da localização da pasta de configurações e do nome do ficheiro. O código seguinte demonstra como é feita a leitura de todas as linhas de um ficheiro CSV:

```
1 | File mFile =
  |   new File(getExternalFilesDir("SettingsFiles"), CSVFile);
  |
2 | FileInputStream mFileInputStream = new FileInputStream(mFile);
3 | InputStreamReader mInputStreamReader =
  |   new InputStreamReader(mFileInputStream);
4 | BufferedReader mBufferedReader =
  |   new BufferedReader(mInputStreamReader);
5 | int line = 0;
6 | String fileLine;
7 | String[][] fileInfo = new String[Constants.max_columns+1][4];
  |
  | // Leitura de todas as linhas do ficheiro CSV
8 | while((fileLine = mBufferedReader.readLine()) != null) {
9 |   String[] rowInfo = fileLine.split(",");
10|   for(int i = 0; i < rowInfo.length; i++) {
11|     fileInfo[line][i] = rowInfo[i];
12|   }
13|   line++;
14| }
15| mFileInputStream.close();
```

Como visualizado no código anterior, a leitura de todas as linhas do ficheiro CSV é executada com um ciclo *while* (linha 8). Dentro deste, é determinado e testado o valor de uma linha para verificar se já foi efetuada a leitura de todas.

Definido o valor da linha, é separado o texto, em que o caracter que identifica cada coluna num ficheiro CSV é a vírgula. Portanto, é implementado o método *split* (linha 9), e guardado o valor de cada coluna na variável auxiliar *rowInfo*. De seguida, na linha 10, são percorridos todos os valores desta, para que cada um seja inserido num vetor de duas dimensões (*fileInfo*), consoante o número da linha e da coluna do ficheiro CSV – linha 11.

Não havendo mais linhas a serem lidas no arquivo é chamado o método *close* do objeto *FileInputStream* na linha 15.

Para terminar o processo de importação são enviadas todas as tramas de configurações associadas a cada coluna, para o microcontrolador efetuar a mudança de parâmetros.

6.2.8. ATIVIDADE DE INSERÇÃO DE ÍCONES

A atividade de inserção de ícones é iniciada quando o utilizador pressiona o botão *Inserir* da área de *Alteração do Modo e Controlo da Matriz* da atividade principal, no modo de funcionamento sem bolas. Nesta atividade são visualizados todos os ícones prontos a serem inseridos no sistema, que foram anteriormente criados por um utilizador da aplicação.

A Figura 103 ilustra o *layout* da atividade de inserção de ícones:

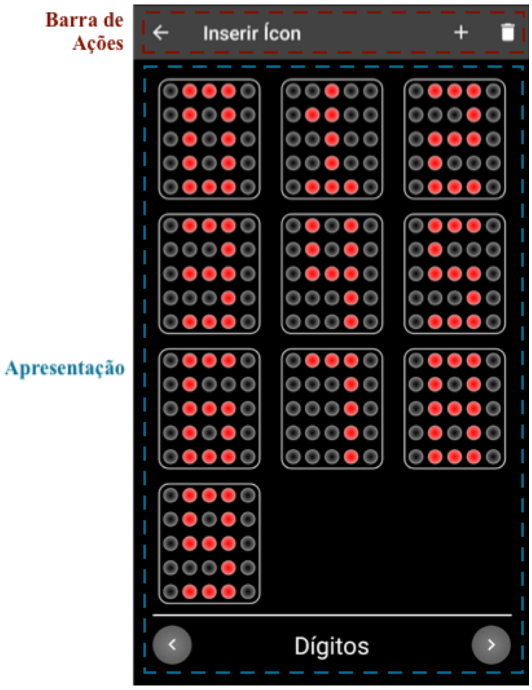


Figura 103 Atividade de inserção de ícones

Como legendado na Figura 103, a atividade de inserção de ícones é composta por duas áreas:

- **Barra de Ações** – permite ao utilizador criar novos ícones, ou eliminá-los.
- **Apresentação** – exibe todos os ícones de acordo com o tipo selecionado (Dígitos, Letras, Sinais, Outros).

Na Figura 104 são apresentados ícones já criados para cada tipo:

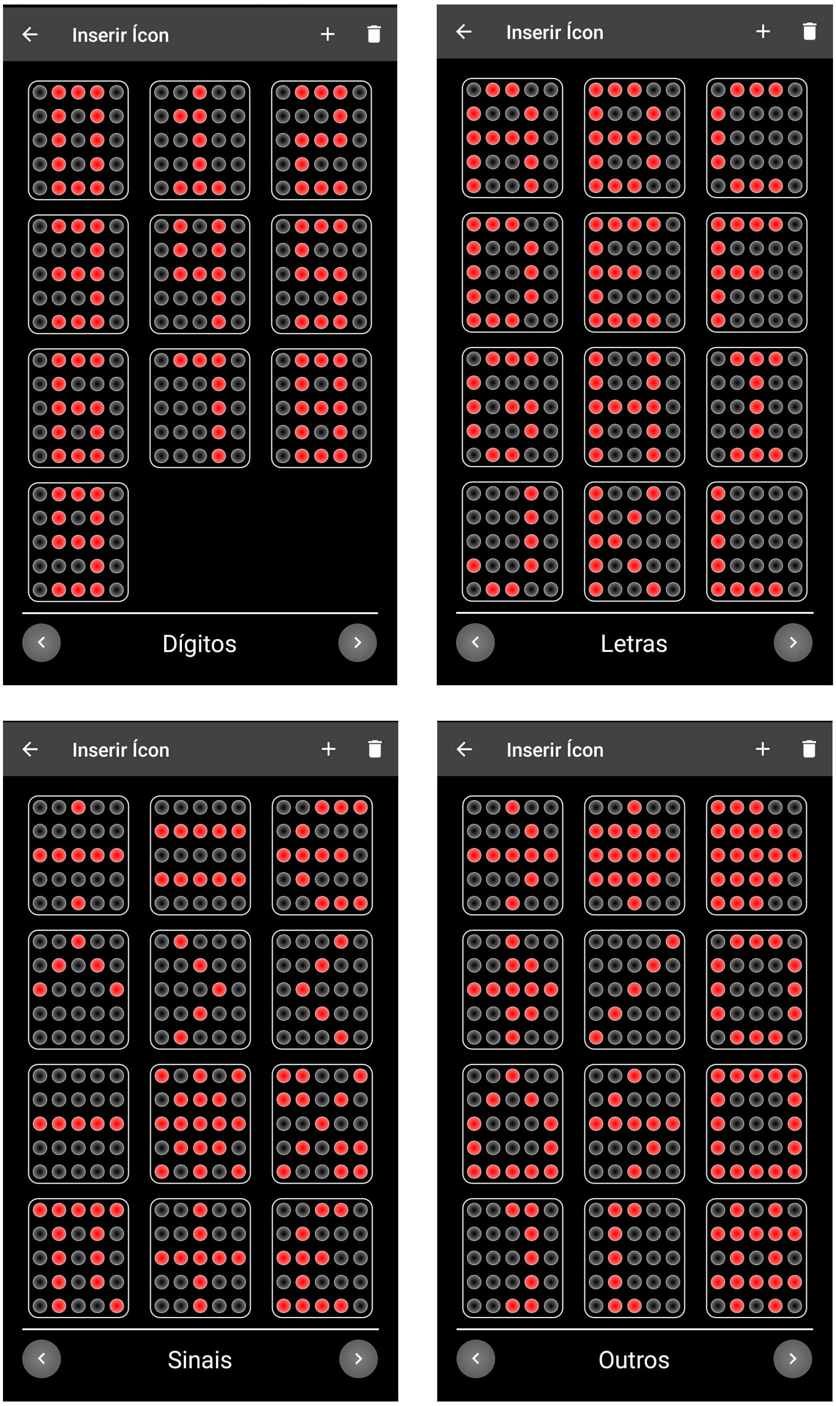


Figura 104 Ícones de cada tipo

Barra de Ações

Na atividade de inserção de ícones está presente uma barra de ações que é visualizada e legendada na Figura 105:

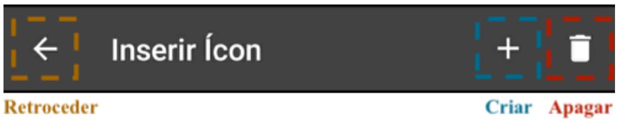


Figura 105 Barra de ações da atividade de inserção de ícones

Assim, através do controlo da barra de ações ilustrada na Figura 105, o utilizador da aplicação pode criar novos ícones (*Criar*), como também eliminá-los (*Apagar*).

A criação de um ícone é realizada através da classe *Dialog*, que foi implementada para o desenvolvimento de um *layout* personalizado na linguagem de programação XML. A caixa de diálogo apresentada quando premido o botão de ação *Criar* é ilustrada na Figura 106:

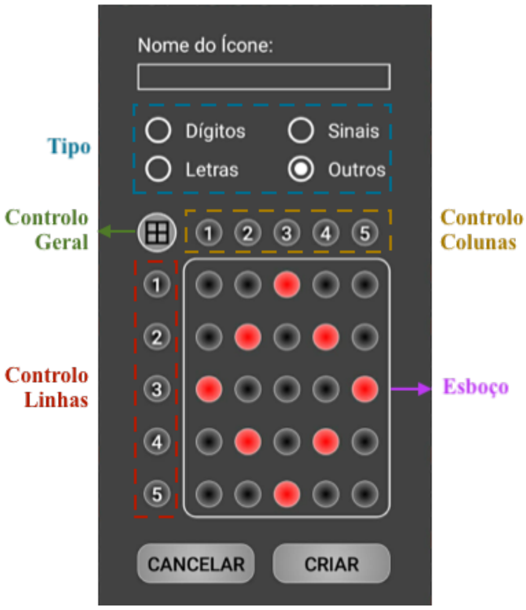


Figura 106 Caixa de diálogo relativa à criação de um ícone

Na caixa de diálogo da Figura 106, o utilizador escolhe o nome que descreve o ícone, assim como o seu tipo, que é selecionado através de elementos *RadioButton*. De seguida, realiza o desenho do ícone numa matriz de 5x5 (*Esboço*), podendo ser efetuado de três maneiras:

- Botões das posições, que o utilizador pressiona para alterar a cor de cada uma;
- Botões numerados, que ativam ou desativam todos as posições relativas a uma linha ou coluna (*Controlo Linhas* / *Controlo Colunas*).
- Botão *Controlo Geral* que permite ativar ou desativar todas as posições do esquema.

Quando presente numa dada posição do desenho a cor preta, significa que esta se encontra desativada, caso contrário é visualizada a cor vermelha que indica a sua ativação. Em cada posição é implementado um objeto da classe *GradientDrawable*, que é caraterizada por ser um desenhável com gradientes de cor, definido como um recurso XML.

Como todas as posições da área *Esboço* correspondem ao mesmo recurso XML, tornou-se necessário a implementação do método *mutate* na declaração de cada objeto *GradientDrawable*. O método garante que o estado numa dada posição não é compartilhado com os restantes. Assim, caso não fosse executado este método, quando o utilizador pressionasse uma posição do desenho, todas as restantes seriam modificadas de igual forma.

Por último, a caixa de diálogo apresenta os botões *Cancelar* e *Criar*. O primeiro fecha a caixa de diálogo, enquanto que o outro realiza o processo de criação do ícone.

Inicialmente, após premido o botão *Criar*, é verificado se o nome do ficheiro inserido já existe no local indicado pelo elemento do tipo *RadioButton*. Caso se comprove esta afirmação, surge uma nova caixa de diálogo (Figura 107) que informa o utilizador que o ícone já existe e lhe pergunta se deseja substituí-lo.

Figura 107 Caixa de diálogo relativa à indicação da existência de um ícone

No caso de o utilizador pretender substituir um ícone ou o nome deste não existir, é executado o seu processo de criação. Tal como os ficheiros de configuração, os ícones também são definidos em arquivos CSV. A informação relativa a estes é disponibilizada em três colunas:

- **Linha** – indica o estado do ícone numa linha do desenho (valor entre 1 e 5);
- **Coluna** – indica o estado do ícone numa coluna do desenho (valor entre 1 e 5);
- **Estado** – indica o estado do ícone na linha e coluna definidas nos parâmetros *Linha* e *Coluna*, respetivamente. O seu valor é booleano e corresponde à cor da posição: cor vermelha, valor *true*; cor preta, valor *false*.

Para melhor compreensão, encontra-se apresentado no Anexo F o conjunto de dados relativos ao ícone desenhado na caixa de diálogo da Figura 106.

Apresentação

É na área de apresentação da atividade de inserção de ícones que são demonstrados todos os ícones já criados pelo utilizador.

Inicialmente, antes de serem apresentados os ícones, é necessário efetuar a leitura de cada um que já tenha sido criado, de forma a que posteriormente possam ser exibidos. O processo de leitura de dados de cada um é semelhante ao código apresentado na atividade de importação de ficheiros de configuração (secção 6.2.7). Porém, a interpretação destes dados é alterada, dado que os valores obtidos servirão para visualizar o ícone.

Depois de identificados todos os ícones criados pelo utilizador, estes são apresentados na atividade de inserção de ícones. Para isso, é implementado um objeto da classe *GridView*, com o objetivo de organizar todos os ícones de um tipo numa matriz de visualização.

A exibição de todos os ícones de um tipo no elemento *GridView* é realizada pelo método *setAdapter*. Este define um adaptador personalizado cujo objetivo é a ilustração de todos os ícones de um certo tipo.

O adaptador implementado no objeto da classe *GridView* denomina-se *IconsAdapter*, tendo sido desenvolvido numa classe Java à parte. Este estende a classe *BaseAdapter*, que implementa os seguintes métodos:

- ***getCount*** – indica quanto itens estão no conjunto de dados representado pelo adaptador;
- ***getItem*** – retorna o objeto relacionado com a posição especificada no conjunto de dados;
- ***getView*** – retorna uma *View* relacionada com a posição especificada no conjunto de dados.

O método mais importante do adaptador *IconsAdapter* é sem dúvida o *getView*. Este cria uma *View*, isto é, uma nova visualização do ecrã, para cada ícone referenciado pelo adaptador. Assim, o método *getView* é invocado consoante o número de ícones de cada tipo.

Quando escolhido um ícone na área *Apresentação* é verificado se o botão de ação *Apagar* se encontra ativo. Este botão, quando apresentado com cor vermelha, permite ao utilizador apagar o ficheiro CSV relativo ao ícone selecionado.

Caso o botão de ação *Apagar* esteja desativado, significa que, quando selecionado um item do objeto da classe *GridView*, é apresentada a caixa de diálogo relativa à inserção de um ícone no sistema, visualizada na Figura 108:

Figura 108 Caixa de diálogo relativa à inserção de um ícone

A caixa de diálogo (Figura 108) é representada por um objeto da classe *Dialog*, sendo este personalizado dado que associa o seu conteúdo a um *layout* definido num recurso XML. Esta janela ilustra o ícone selecionado no elemento *GridView* da atividade.

Também estão disponibilizados na caixa de diálogo dois botões de imagem que permitem rodar o ícone selecionado no esquema para a esquerda ou para a direita. Sempre que premido um destes são alteradas as posições do ícone selecionado, através da passagem de valores de umas posições para as outras. A Tabela 34 representa os cálculos efetuados para a alteração de cada posição, consoante a rotação efetuada:

Tabela 34 Atribuição de linha e coluna consoante rotação do ícone

	Atribuição de Linha	**Atribuição de Coluna**
Rotação para a esquerda	$4 - Valor\_Coluna$	$Valor\_Linha$
Rotação para a direita	$Valor\_Coluna$	$4 - Valor\_Linha$

Por último, o utilizador define a posição que deseja colocar o ícone no sistema através da definição dos parâmetros de linha e coluna inicial.

A Figura 109 ilustra todas as representações do ícone relativo ao dígito 1, dependendo do seu valor de rotação:

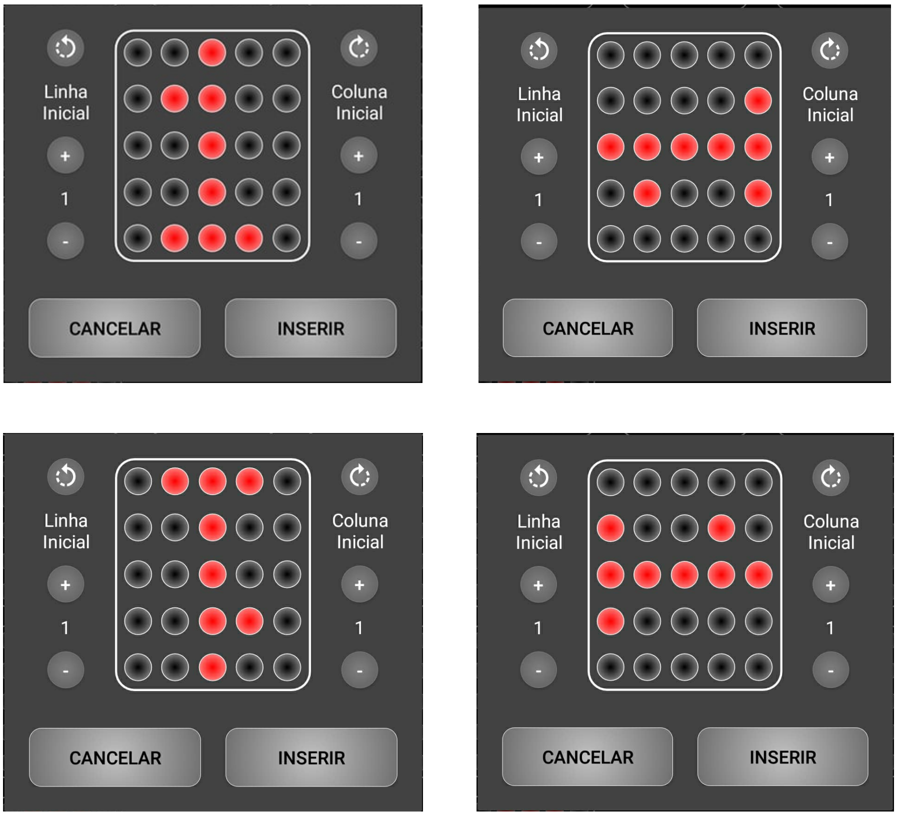


Figura 109 Rotação do ícone relativo ao dígito 1

Definidos todos os critérios relativos à inserção do ícone, é premido o botão *Inserir* da caixa de diálogo da Figura 108, que retorna para a atividade principal os valores de construção da trama de envio de um ícone para o microcontrolador. Nesta, como demonstrado pelo identificador 11 na Figura 79, o ícone é representado por 4 *bytes*, dado que este ocupa 25 posições no sistema, em que cada uma equivale a um *bit*.

O valor lógico do *bit* é 1 caso a posição esteja ativa, isto é, tenha a cor vermelha presente. Caso a cor definida na posição seja preta, o *bit* obtém o valor lógico 0. Todos os valores são guardados numa variável denominada *temp_icon*, que diz respeito aos quatro *bytes* enviados na trama para o microcontrolador.

O processo de criação da variável referente a um ícone (*temp_icon*) é demonstrado no fluxograma da Figura 110:

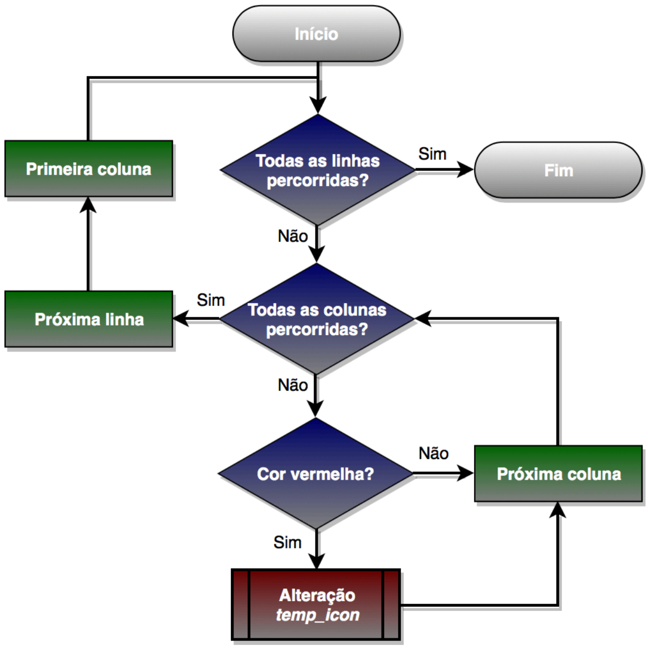


Figura 110 Fluxograma relativo à criação da variável final do ícone

Como visualizado no fluxograma da Figura 110, são percorridas todas as linhas e colunas do esquema do ícone, onde é verificada qual a cor presente em cada posição. Caso se confirme a cor vermelha, é alterado o valor da variável *temp_icon* consoante a linha e a coluna (*Alteração temp_icon*).

No processo *Alteração temp_icon* é realizado o cálculo da variável *temp_icon* quando uma posição se encontra ativa, isto é, tem presente a cor vermelha. Assim, é utilizado o método *pow*, para que o valor lógico de cada *bit* da variável corresponda ao estado de uma posição do esquema do ícone. A Equação 8 demonstra o cálculo da variável *temp_icon* para cada posição ativa, em que as letras *i* e *j* identificam, respetivamente, o valor da linha e coluna:

$$temp\_icon = temp\_icon + 2^{(i*5)+j}$$

Equação 8

Nota: a primeira linha e coluna do esboço do ícone correspondem ao valor zero, sendo, portanto, a última linha e coluna o número quatro.

Como exemplo de aplicação do cálculo da variável *temp_icon* no processo representado pelo fluxograma da Figura 110, temos o ícone desenhado na Figura 106. No final da leitura de todas as posições do ícone, o valor de *temp_icon* é o seguinte:

$$temp_{icon} = 2^2 + 2^6 + 2^8 + 2^{10} + 2^{14} + 2^{16} + 2^{18} + 2^{22} = 4539716$$

6.2.9. Atividade de Visualização das Configurações de cada Coluna

Esta atividade permite que o utilizador verifique quais as configurações associadas a cada coluna ativa no sistema, sendo elas o instrumento, a nota musical e a cor de ativação dos LEDs. Nesta é definida uma tabela através de um elemento *TableLayout*, onde são inseridos os elementos que identificam a cor, e os elementos de texto, que demonstram o instrumento e a nota. A atividade é ilustrada na Figura 111, sendo visualizadas as configurações presentes no sistema, com todas as colunas ativas:

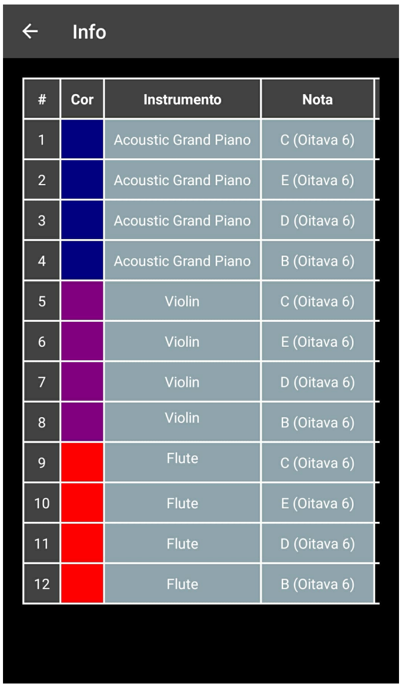


Figura 111 Atividade de visualização das configurações de cada coluna

6.3. Resultados

O principal resultado do projeto é a construção e implementação de um sequenciador MIDI físico, isto é, de uma estrutura totalmente funcional. Esta é composta por doze linhas e doze colunas, resultando ao todo 144 posições, em que cada uma tem um relevo de forma a que o utilizador coloque as bolas transparentes.

No interior de cada bola foi inserido silicone com o objetivo de aumentar o peso a esta, para que quando posicionada faça pressão sobre o interruptor da posição onde se encontra. A Figura 112 demonstra a bola que pode ser inserida em cada posição do sequenciador MIDI:

Figura 112 Bola utilizada no sequenciador MIDI

Na Figura 113 pode ser visualizado o exterior da estrutura construída para o projeto, mais especificamente a parte superior onde podem ser posicionadas as bolas transparentes:

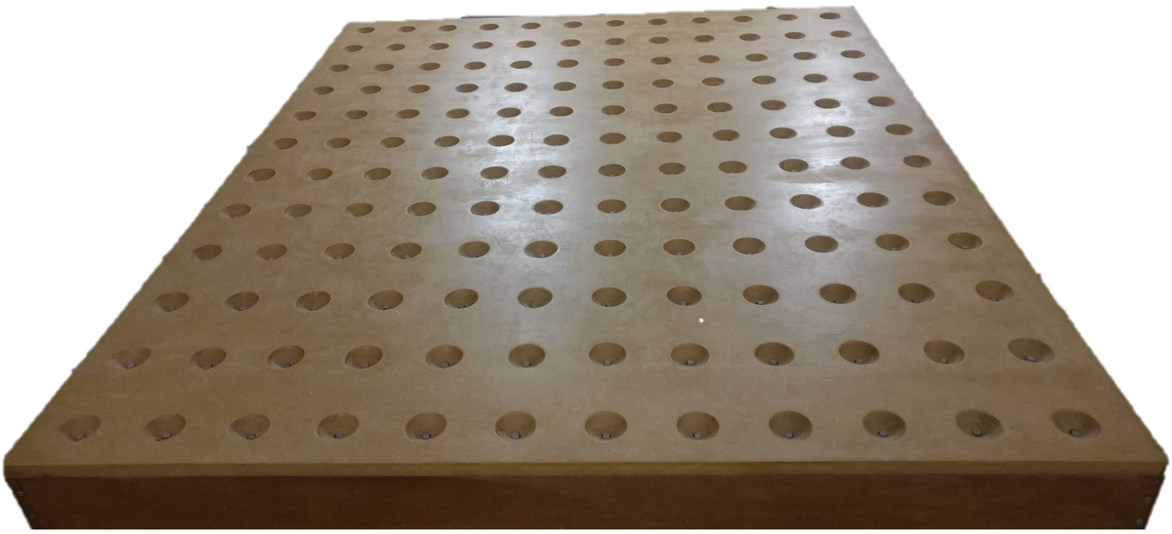


Figura 113 Exterior da estrutura

De seguida, é apresentado na Figura 114 o interior da estrutura. Dentro do sequenciador encontra-se a fita dos LEDs WS2812B, disposta no formato de serpente. A esconder estes componentes estão tábuas de madeira, onde em cada uma estão inseridos seis botões. Estas foram concebidas com o intuito de possibilitar a pressão nos botões quando premidas as respetivas posições.

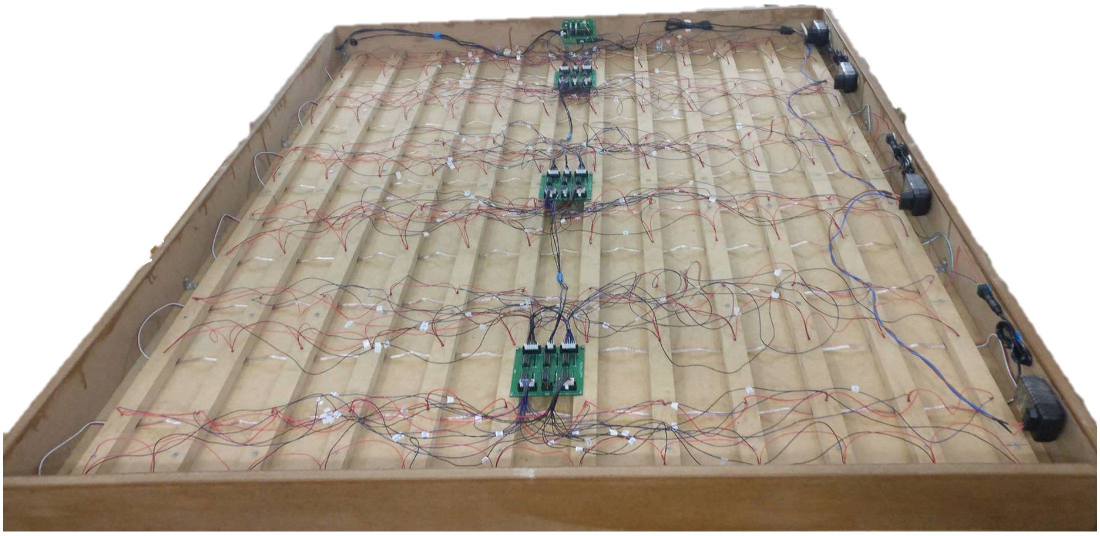


Figura 114 Interior da estrutura

No interior da estrutura encontram-se as placas de circuito impresso desenvolvidas, isto é, a Placa de Controlo (Figura 55) e as três Placas de Leitura (Figura 58). Cada placa de leitura permite determinar o estado de botões de quatro linhas do sistema, estando por isso posicionadas como demonstrado na Figura 114.

Dentro do sequenciador estão ainda colocados transformadores numa das paredes da estrutura. Ao todo são quatro, em que três deles realizam a alimentação dos LEDs WS2812B, e o outro alimenta a Placa de Controlo, assim como as Placas de Leitura e seus componentes (botões), com uma tensão de 12 V.

Em relação aos LEDs tornou-se necessário a utilização de três transformadores devido ao consumo de corrente ser elevado, como já foi descrito na secção 4.2.2. Assim, cada um fornece à saída uma tensão de 5 V e uma corrente de 3 A, o que permite alimentar quatro linha do sistema.

Além da estrutura, isto é, do sequenciador em si, o sistema implementado incide-se na interligação com o programa MIDI, utilizado para a produção do som musical, e com a aplicação Android, que possibilita a monitorização e alteração de parâmetros do sequenciador.

Quando o sequenciador é ligado ao *software* presente no computador, são recebidas mensagens MIDI que permitem emitir o som consoante a linha ativa do compasso de tempo e o estado de cada posição do sistema. Também é possível a escolha do instrumento musical no *software*.

Como exemplo de demonstração do sistema temos a Figura 115, em que a sua monitorização na aplicação Android foi anteriormente ilustrada na Figura 84:

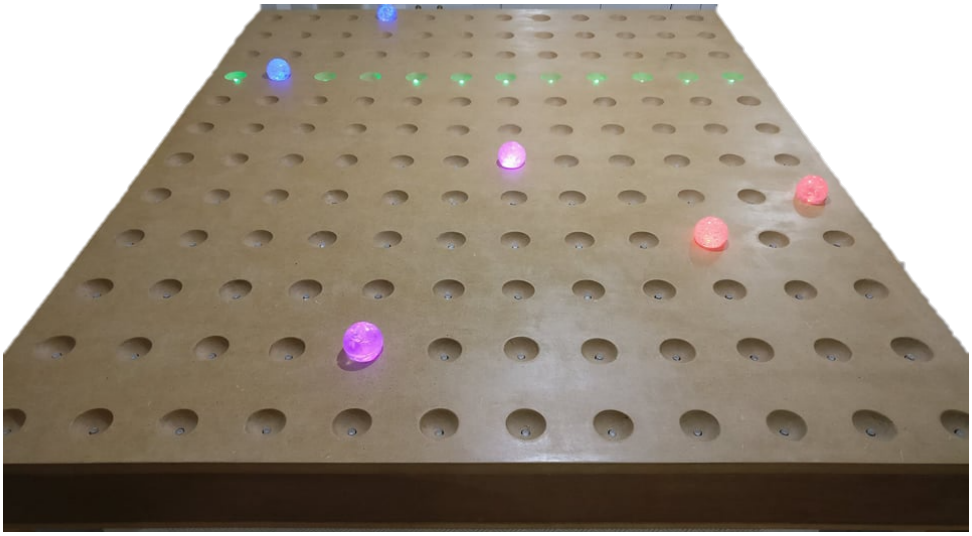


Figura 115 Sequenciador implementado

No que diz respeito à integração do MIDI no sistema, como referido anteriormente na secção 4.3.2, esta é realizada entre o sequenciador e o computador pelo *software Hairless MIDI to Serial Bridge*. Este recebe os dados transmitidos pelo sequenciador, convertendo-os em mensagens MIDI para serem enviadas para o programa *SimpleSynth*.

A Figura 116 ilustra o ambiente gráfico de ambos os *softwares* utilizados na reprodução musical do sequenciador:

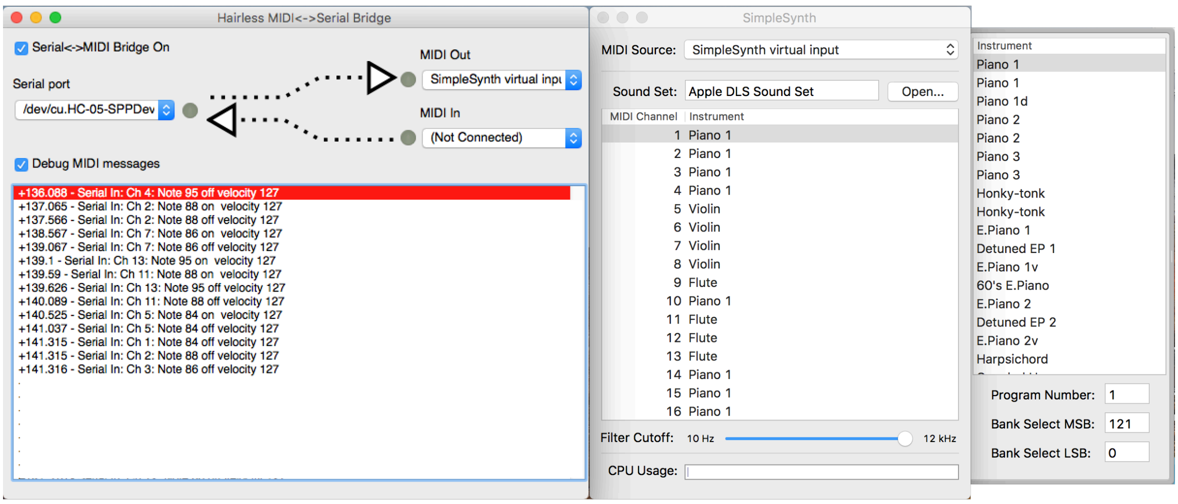


Figura 116 Reprodução musical do sequenciador

Como visível no *software SimpleSynth* na Figura 116, a cada canal MIDI corresponde um instrumento musical, que é associado a todas as posições de uma coluna no sequenciador. Assim, o número do canal indica a coluna respetiva, à exceção das últimas três colunas (10, 11 e 12), em que se teve de avançar um canal, sendo elas, respetivamente, o canal 11, 12 e 13. Isto deve-se ao facto do canal MIDI com o número 10 permitir somente instrumentos de percussão, como especificado na norma.

7. CONCLUSÕES

Ao longo deste relatório foram sendo apresentadas conclusões que permitiram justificar as tomadas de decisão do sistema Sequenciador MIDI. Portanto, nesta última secção são referidas as principais conclusões relativas ao projeto desenvolvido, assim como algumas propostas de trabalho futuro para melhoria do mesmo.

O projeto passou por várias fases, sendo elas a sua esquematização, o estudo de sistemas semelhantes, a construção da estrutura, a implementação do *hardware* e o desenvolvimento do *software*, que no final culminaram na elaboração deste relatório.

O objetivo principal definido inicialmente foi a criação de um instrumento musical eletrónico, que envia informações para um *software* MIDI instalado num computador, com o objetivo de realizar a produção musical do sistema. Também foi desenvolvida uma aplicação Android com o intuito de efetuar a monitorização e alteração de parâmetros.

Conclui-se que o projeto Sequenciador MIDI é um sistema totalmente funcional, que possibilita a criação de faixas musicais por várias pessoas num único instrumento. Assim, todos os objetivos previstos foram atingidos e, consequentemente, pode-se dizer que o resultado final corresponde às expectativas inicialmente criadas.

Durante o desenvolvimento do projeto foi necessário ultrapassar vários obstáculos, sendo os principais a limitação imposta pelo microcontrolador no número de I/O, e a programação da função para atribuição da cor aos LEDs. No primeiro caso encontrou-se como solução a implementação de *shift registers*, com o intuito de ler o estado dos 144 botões do sistema. Em relação ao segundo, a função que permite atribuir a cor aos LEDs foi desenvolvida na linguagem de programação *Assembly*, dado que os tempos impostos pelo protocolo de transferência de dados dos LEDs eram bastante curtos.

Além dos obstáculos referidos no último parágrafo, existe também a criação da aplicação Android, dado que esta implicou a aprendizagem de uma nova linguagem de programação, sendo ela o *Java*.

Após conclusão deste trabalho é importante referir que beneficiei bastante com a sua realização, dado que além de ter colocado em prática conhecimentos aprendidos ao longo do curso, também aprendi novos conceitos, tecnologias, técnicas e métodos de trabalho.

Em relação a futuras melhorias, o projeto Sequenciador MIDI seria capaz de guardar faixas musicais através da aplicação Android, de forma a que fosse possível ao utilizador reproduzi-las mais tarde.

Um trabalho futuro interessante seria a implementação do conceito do projeto de uma forma diferente. Tem-se como exemplo a utilização de um grupo de pessoas, que dependentemente do posicionamento que ocupem numa sala, permitem a ativação de uma das posições do sequenciador, tocando notas musicais à medida que um compasso de tempo transite pela mesma.

Referências Documentais

[1] *O que é Música?*. Disponível em: http://www.descomplicandoamusica.com/o-que-e-musica/. Acedido a 9 de março de 2017.

[2] *Invenção do Fonógrafo.* (2012). Disponível em: http://estoriasdahistoria12.blogspot.pt/2012/11/21-de-novembro-de-1877-invencao-do.html. Acedido a 31 de março de 2017.

[3] MCCAAN, DOUG. THORNE, Peter. *The Music of CSIRAC*. (2015). Disponível em: http://www.cis.unimelb.edu.au/about/csirac/music/introduction.html. Acedido a 31 de março de 2017.

[4] University of California, Santa Cruz. *Computer Music (So Far).* (2010). Disponível em: http://artsites.ucsc.edu/ems/music/equipment/computers/history/history.html. Acedido a 31 de março de 2017.

[5] Apple Inc. *Synthesizer Basics.* (2009). Disponível em: https://documentation.apple.com/en/logicexpress/instruments/index.html#chapter=A%26section=0. Acedido a 31 de março de 2017.

[6] *Minimoog Model D 1970 Modern Remake.* (2016). Disponível em: https://synth.market/en/news/minimoog_is_back/. Acedido a 20 de junho de 2017.

[7] PLACE, Adam. *AlphaSphere*. (2016). Disponível em: http://www.alphasphere.com/about/. Acedido a 2 de abril de 2017.

[8] LAMBERT, John. *Eigenlabs*. (2010). Disponível em: http://www.eigenlabs.com. Acedido a 2 de abril de 2017.

[9] Reactable Systems. *About*. (2009). Disponível em: http://reactable.com. Acedido a 2 de abril de 2017.

[10] *The reacTable experience with Martin Kaltenbrunner*. (2013). Disponível em: https://interactiondesignlmu.wordpress.com/2013/07/13/the-reactable-experience-with-martin-kaltenbrunner/. Acedido a 2 de abril de 2017.

[11] KALTENBRUNNER, Martin. *TUIO – A Protocol for Table-Top Tangible User-Interfaces*. (2009). Disponível em: http://www.tuio.org. Acedido a 2 de abril de 2017.

[12] DARLING, David. *Laser Harp*. (2016). Disponível em: http://www.songsofthecosmos.com/encyclopedia_of_modern_music/L/laser_harp.html. Acedido a 27 de julho de 2017.

[13] LELLO, André. ALMEIDA, António. SOUTO, Catarina. *Harpa Laser ARSLUX – A Musicalidade da Luz*. Disponível em: http://www.optica.pt/IYL2015/components/com_chronoforms5/chronoforms/inscricao/20150313114331_ArsLux_LaserHarp_IYL2015_CLF.pdf. Acedido a 27 de julho de 2017.

[14] GERSTEIN, Yuval. *GRIDI – Large Scale MIDI Sequencer By Yuvi Gerstein.* (2015). Disponível em: http://www.yuvalgerstein.com/gridi-large-scale-midi-sequencer-by-yuvi-gerstein/. Acedido a 23 de março de 2017.

[15] The MIDI Association. *About*. (2016). Disponível em: https://www.midi.org. Acedido a 9 de março de 2017.

[16] CHADABE, Joel. *Part IV: The Seeds of the Future*. (2000). Disponível em: http://www.emusician.com/gear/1332/the-electronic-century-part-iv-the-seeds-of-the-future/32109. Acedido a 9 de março de 2017.

[17] Main Drag Music. *Sequential Circuits Prophet 600*. (2016). Disponível em http://www.maindragmusic.com/ca-1983-sequential-circuits-prophet-600.html. Acedido a 10 de março de 2017.

[18] Vintage Synth Explorer. *Roland Jupiter-6.* (2015). Disponível em: http://www.vintagesynth.com/roland/jup6.php. Acedido a 10 de março de 2017.

[19] *MIDI Messages.* Disponível em: http://www.electronics.dit.ie/staff/tscarff/Music_technology/midi/midi_messages.htm. Acedido a 30 de março de 2017.

[20] HASS, Jeffrey. *Chapter Three: How MIDI Works*. (2010). Disponível em: http://www.indiana.edu/%7Eemusic/etext/MIDI/chapter3_MIDI.shtml. Acedido a 14 de março de 2017.

[21] ABELHA, Tiago. *Sistemas de produção e criação musical*. (2003). Disponível em: http://www.dei.isep.ipp.pt/~paf/proj/Julho2003/Criacao_Producao_Musical.pdf. Acedido a 17 de março de 2017.

[22] Google Developers. *Android – Atividades*. (2016). Disponível em: https://developer.android.com/guide/components/activities.html. Acedido a 9 de agosto de 2017.

[23] Google Developers. *Android – Serviços*. (2016). Disponível em: https://developer.android.com/guide/components/services.html. Acedido a 9 de agosto de 2017.

[24] Google Developers. *Android – Intents e Filtros de Intents*. (2016). Disponível em: https://developer.android.com/guide/components/intents-filters.html. Acedido a 9 de agosto de 2017.

[25] Google Developers. *Android – Manifesto do Aplicativo*. (2016). Disponível em: https://developer.android.com/guide/topics/manifest/manifest-intro.html. Acedido a 9 de agosto de 2017.

[26] WorldSemi. *WS2812 vs WS2812B*. (2012). Disponível em: https://cdn.sparkfun.com/assets/learn_tutorials/1/0/5/WS2812B_VS_WS2812.pdf. Acedido a 1 de junho de 2017.

[27] Acrobotic Industries. *Upgrading Smart RGB LEDs: WS2812B Vs. WS2812*. (2014). Disponível em: http://www.instructables.com/id/Upgrading-Smart-RGB-LEDs-WS2812B-vs-WS2812/. Acedido a 1 de junho de 2017.

[28] WorldSemi. *WS2812B Intelligent Control LED Integrated Light Source*. (2012). Disponível em: http://www.seeedstudio.com/document/pdf/WS2812B%20Datasheet.pdf. Acedido a 27 de março de 2017.

[29] SparkFun. *Switch Basics*. (2010). Disponível em: https://learn.sparkfun.com/tutorials/switch-basics. Acedido a 31 de julho de 2017.

[30] MAW, Carlyn. IGOE, Tom. *Parallel to Serial Shifting-In with a CD4021BE*. (2007). Disponível em: https://www.arduino.cc/en/tutorial/ShiftIn. Acedido a 2 de junho de 2017.

[31] SparkFun. *Bluetooth Basics*. (2010). Disponível em: https://learn.sparkfun.com/tutorials/bluetooth-basics. Acedido a 3 de agosto de 2017.

[32] GRATTON, Angus. *The Hairless MIDI to Serial Bridge*. Disponível em: http://projectgus.github.io/hairless-midiserial/. Acedido a 5 de agosto de 2017.

[33] Infusion Systems Ltd. *Installing and Using LoopMIDI to create a virtual MIDI port*. (2013). Disponível em: https://infusionsystems.com/catalog/info_pages.php/pages_id/245. Acedido a 5 de agosto de 2017.

[34] Google Developers. *Android – Conheça o Android Studio*. (2016). Disponível em: https://developer.android.com/studio/intro/index.html. Acedido a 9 de agosto de 2017.

[35] EasyEDA. *Introduction to EasyEDA*. (2015). Disponível em: https://easyeda.com/Doc/Tutorial/Introduction.htm#Introduction-to-EasyEDA. Acedido a 9 de setembro de 2017.

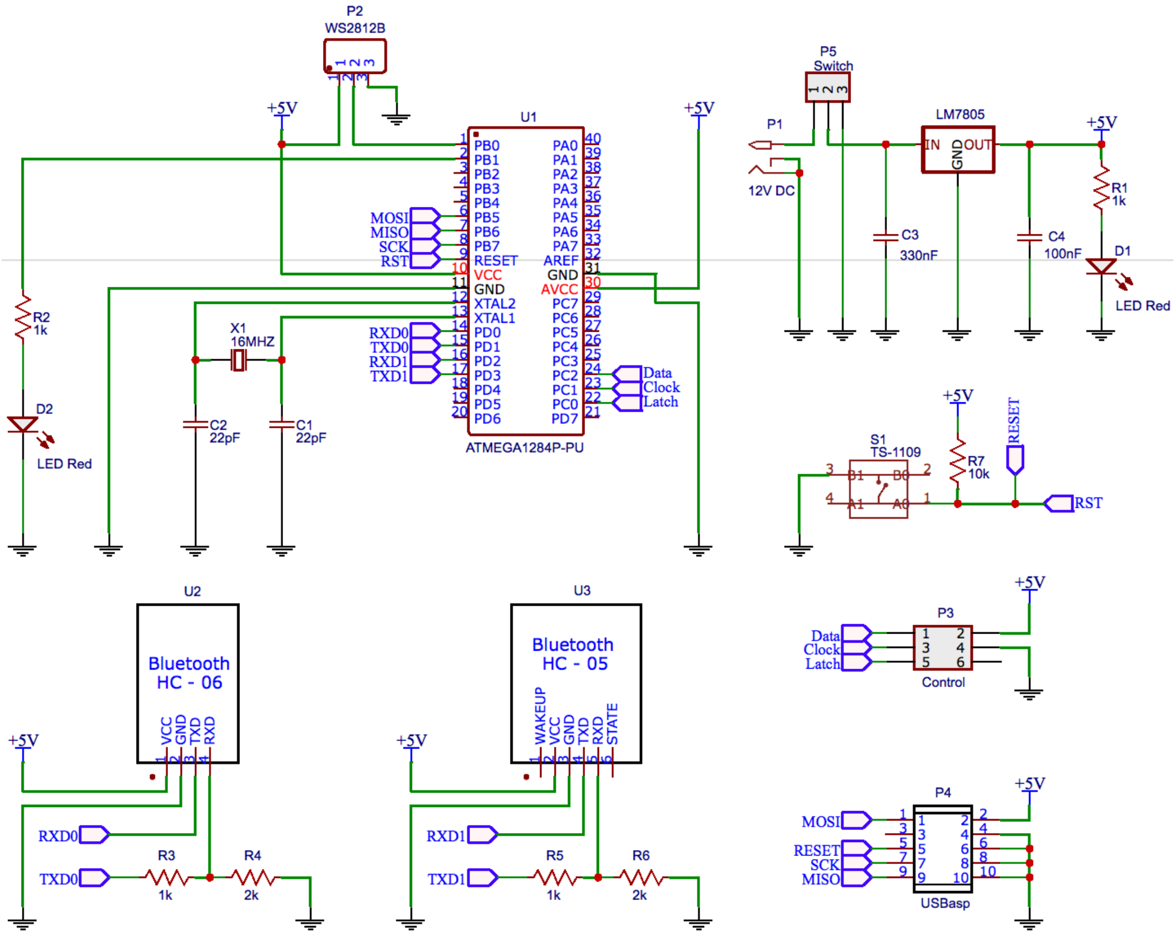
[36] Robomart.com. *LM7805 Voltage Regulators*. (2013). Disponível em: https://www.robomart.com/l7805-voltage-regulator. Acedido a 5 de setembro de 2017.

[37] Atmel. *ATmega1284P – Datasheet Complete*. (2016). Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42719-ATmega1284P_Datasheet.pdf. Acedido a 15 de setembro de 2017.

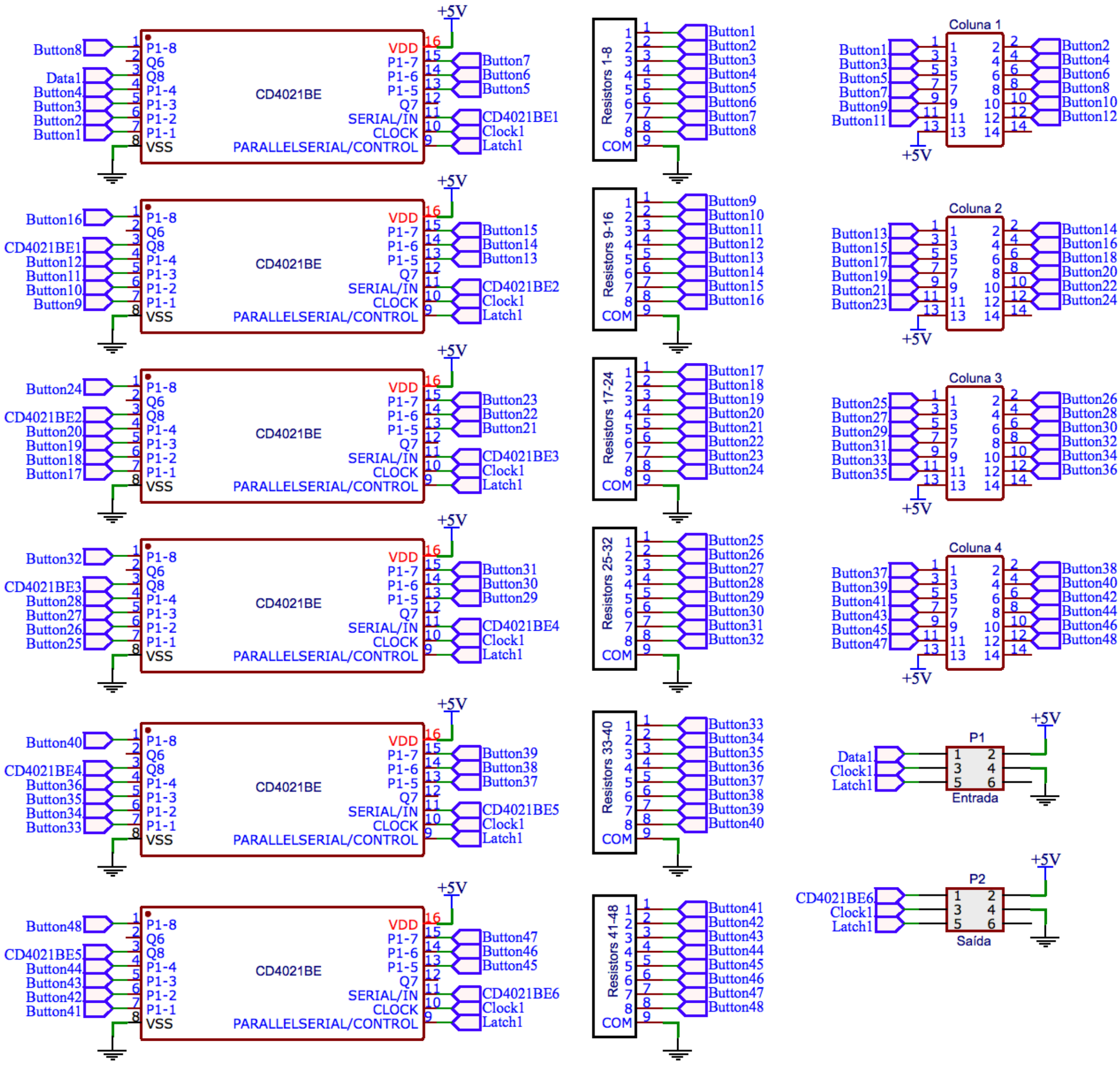
[38] CAMERA, Dean. *Using the EEPROM memory in AVR-GCC*. (2016). Disponível em: https://teslabs.com/openplayer/docs/docs/prognotes/EEPROM%20Tutorial.pdf. Acedido a 20 de novembro de 2017.

[39] Google Developers. *Android – Bluetooth*. (2016). Disponível em: https://developer.android.com/guide/topics/connectivity/bluetooth.html. Acedido a 9 de agosto de 2017.

Anexo A. Esquema Elétrico da Placa de Controlo



Anexo B. Esquema Elétrico de uma Placa de Leitura



Anexo C. Registos associados ao *Timer/Counter0*

***TC0 Control Register* (TCCR0A)** – Este registo permite controlar o pino de comparação de saída tanto do canal A (OC0A) como do B (OC0B), assim como ajudar a definir o modo através dos *bits* WGM00 e WGM01.

Bit	7	6	5	4	3	2	1	0
	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

***TC0 Control Register B* (TCCR0B)** – Neste registo está presente o *bit* WGM02 que permite definir o modo em conjuntos com os *bits* WGM00 e WGM01 do registo TCCR0A, assim como também os *bits* CS00, CS01 e CS02, que selecionam a fonte de relógio a ser utilizada, dependendo do *prescaler*.

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS0[2:0]		
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

***TC0 Interrupt Mask Register* (TIMSK0)** – Neste registo são ativados os *bits* que possibilitam a ativação de certas interrupções relacionadas com o *Timer/Counter0* no microcontrolador.

Bit	7	6	5	4	3	2	1	0
						OCIEB	OCIEA	TOIE
Access						R/W	R/W	R/W
Reset						0	0	0

***TC0 Output Compare Register A* (OCR0A)** – Como já falado, este registo contém um valor de 8 *bits*, isto é, pode variar entre 0 e 255, que é constantemente comparado com o valor do contador TCNT0. Quando ambos coincidirem pode ser gerada uma interrupção ou uma forma de onda no pino OC0A.

Bit	7	6	5	4	3	2	1	0
	OCR0A[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Anexo D. Registos associados à USARTn

***USART Baud Rate Register n* (UBRRn)** – Este registo representa um valor de 16 *bits* que contém a taxa de transmissão da USART, sendo constituído pelos registos de 8 *bits* UBRRnL e UBRRnH.

Bit	15	14	13	12	11	10	9	8
					UBRR[11:8]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	UBRR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

***USART Controlo and Status Register n A* (UCSRnA)** – Neste registo estão presentes as *flags* de receção e transmissão, através dos *bits* RXC e TXC respetivamente, assim como *bits* que são utilizados para a deteção de erros na transmissão de dados. Além destes também se encontra o *bit* U2X que é colocado a 1 para duplicar a taxa de transmissão para o modo de comunicação assíncrono.

Bit	7	6	5	4	3	2	1	0
	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

***USART Controlo and Status Register n B* (UCSRnB)** – Estão presentes neste registo os *bits* RXCIE e TXCIE, que ativam as interrupções de receção e transmissão de dados quando colocados a 1. Também é caraterizado por ser o registo que define a permissão da receção e/ou transmissão de dados através dos *bits* RXEN e TXEN. Em conjunto com o registo UCSR1C define o número de *bits* de dados (UCSZ2).

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

***USART Controlo and Status Register n C* (UCSRnC)** – Este registo é implementado para seleção do modo de operação (*bits* UMSEL), da paridade (*bits* UPM), do número de *stop bits* (*bit* USBS) e do número de *bits* de dados (*bits* UCSZ1 e UCSZ0).

Bit	7	6	5	4	3	2	1	0
	UMSEL[1:0]		UPM[1:0]		USBS	UCSZ1 / UDORD	UCSZ0 / UCPHA	UCPOL
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

USART Data Register n (UDRn) – Neste os registos do *buffer* de transmissão de dados (TXB) e do *buffer* de receção de dados (RXB) partilham o mesmo espaço de endereçamento. O TXB é o destino para serem enviados dados pelo registo UDRn enquanto que a leitura do registo retorna os dados contidos em RXB.

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Anexo E. Código relativo à biblioteca *funcs_ws2812b*

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// total de nanosegundos por ciclo
#define ns_cycle (1000000000L/F_CPU)
// conversão de nanosegundos para número de ciclos
#define ns_to_cycles(time) ((time)/ns_cycle)
// Definição dos tempos para transferência de dados
// tempo do estado a 1 em nanosegundos para o bit 0
// tempo do estado a 0 em nanosegundos para o bit 1
#define T0H_T1L 350
// tempo do estado a 1 em nanosegundos para o bit 1
// tempo do estado a 0 em nanosegundos para o bit 0
#define T0L_T1H 900
// tempo total que demora a ser enviado um bit
#define total_period (T0H_T1L+T0L_T1H)
// tempo de reset (voltar a transmitir para o 1º LED)
#define reset_time 500

uint8_t pixels_cores[total_pixels][3];

// Função que permite efetuar a transferência de dados para os
// LEDs WS2812B
void Enviar_Cor(uint8_t cor)
{
  uint8_t i = 0;
  uint8_t bit_1, bit_0;
  bit_1 = (1 << BIT_WS2812B);
  bit_0 = PORT_LEDs & ~(bit_1);
  cli();
  asm volatile
  (
    // inicia contador a 8 (leitura do byte da cor)
    "ldi %[cnt], 8 \n\t"
    "loop%=: \n\t"
      // coloca a 1 pino onde estão ligados os LEDs (PORTB0)
      "out %[port], %[pin1] \n\t"
      // número de ciclos entre t0 e t1 (350ns)
      ".rept %[cycles1] \n\t"
      // a instrução nop equivale a um ciclo
        "nop \n\t"
      ".endr \n\t"
      // testa o bit mais significativo
      // e se for 1 salta a instrução seguinte
      "sbrs %[byte], 7 \n\t"
      // coloca a 0 pino onde estão ligados os LEDs (PORTB0)
      "out %[port], %[pin0] \n\t"
      // deslocamento para a esquerda do byte
      "lsl %[byte] \n\t"
      ".rept %[cycles2] \n\t"
```

```
          // número de ciclos entre t1 e t2 (900 - 350 = 550 ns)
          "nop \n\t"
        ".endr \n\t"
        // coloca a 0 pino onde estão ligados os LEDs (PORTB0)
        "out %[port], %[pin0] \n\t"
        // número de ciclos entre t2 e t3 (1250 - 900 = 350 ns)
        ".rept %[cycles3] \n\t"
          "nop \n\t"
        ".endr \n\t"
        // decrementa contador
        "dec %[cnt] \n\t"
        // se não for 0, efetua a leitura do próximo bit
        "brne loop%= \n\t"
      ::
      [cnt] "r" (i),
      [port] "I" (_SFR_IO_ADDR(PORT_LEDs)),
      [pin1] "r" (bit_1),
      [pin0] "r" (bit_0),
      [cycles1] "I" (ns_to_cycles(T0H_T1L)),
      [cycles2] "I" (ns_to_cycles(T0L_T1H)-
      ns_to_cycles(T0H_T1L)),
      [cycles3] "I" (ns_to_cycles(total_period)-
      ns_to_cycles(T0L_T1H)),
      [byte] "r" (cor)
  );
}

// Função que permite escolher as componentes R, G e B para
// cada LED
void Definir_Pixel_RGB(uint16_t num_pixel, uint8_t r, uint8_t
g, uint8_t b)
{
  cli();
  if(num_pixel < total_pixels)
  {
    pixels_cores[num_pixel][0] = g;
    pixels_cores[num_pixel][1] = r;
    pixels_cores[num_pixel][2] = b;
    // caso o LED escolhido não seja o primeiro
    if(num_pixel > 0)
    {
      uint16_t i = 0;
      // ciclo que percorre todos os LEDs anteriores
      for(i = 0; i < num_pixel; i++)
      {
        // envia primeiro a componente G do LED i
        Enviar_Cor(pixels_cores[i][0]);
        // depois a componente R do LED i
        Enviar_Cor(pixels_cores[i][1]);
        // em último a componente B do LED i
        Enviar_Cor(pixels_cores[i][2]);
      }
    }
    // envia primeiro a componente G do LED escolhido
    Enviar_Cor(g);
    // depois a componente R do LED escolhido
```

```
    Enviar_Cor(r);
    // em último a componente B do LED escolhido
    Enviar_Cor(b);
  }
  _delay_us(reset_time);
  sei();
}

// Função que permite escolher a cor exata (RGB) para cada LED
void Definir_Pixel_Cor(uint16_t num_pixel, uint32_t cor)
{
  uint8_t R = (cor >> 16) & 0b11111111;
  uint8_t G = (cor >> 8) & 0b11111111;
  uint8_t B = (cor) & 0b11111111;
  Definir_Pixel_RGB(num_pixel, R, G, B);
}

// Função que permite desligar um LED
void Desligar_Pixel(uint16_t num_pixel)
{
  Definir_Pixel_RGB(num_pixel, 0, 0, 0);
}

// Função que permite desligar todos os LEDs
void Desligar_Todos_Pixels(void)
{
  uint8_t i = 0;
  for(i = 0; i < total_pixels; i++)
    Definir_Pixel_RGB(i, 0, 0, 0);
}

// Função que permite determinar a cor atual de cada LED, no
// formato RGB
uint32_t Verificar_Pixel_Cor(uint16_t num_pixel)
{
  if(num_pixel < total_pixels)
    return  ((uint32_t)pixels_cores[num_pixel][0] << 8)  |
            ((uint32_t)pixels_cores[num_pixel][1] << 16) |
            ((uint32_t)pixels_cores[num_pixel][2]);
  else
    return 0;
}
```

Anexo F. Ficheiro CSV do ícone da Figura 106

Linha	**Coluna**	**Estado**
1	1	*False*
1	2	*False*
1	3	*True*
1	4	*False*
1	5	*False*
2	1	*False*
2	2	*True*
2	3	*False*
2	4	*True*
2	5	*False*
3	1	*True*
3	2	*False*
3	3	*False*
3	4	*False*
3	5	*True*
4	1	*False*
4	2	*True*
4	3	*False*
4	4	*True*
4	5	*False*
5	1	*False*
5	2	*False*
5	3	*True*
5	4	*False*
5	5	*False*