# Teclado Musical Digital com Raspberry Pi 3

Trabalho Final de Sistemas Embarcados

Eduardo S. Sales Rodrigues - 140038558
Faculdade Gama – Engenharia Eletrônica
UnB – Universidade de Brasília
Brasília - DF.
eduardo-ssr@hotmail.com

***Resumo* — This electronic document is a "live" template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. Este trabalho é referente ao projeto de um teclado musical digital utilizando uma Raspberry Pi 3 e um Arduino Nano. O teclado é composto por 12 teclas e por 4 potênciometros. Ele toca todas as notas músicas, 8 oitavas que são definidas por um dos potenciômetos, e da a possibilidade de ser tocada de forma continua ou pausadamente de acordo com a necessidade do usário.**

***Palavras-Chaves — teclado musical; raspberry pi; escala musical.***

## I. OBJETIVOS

Implementar um teclado musical digital utilizando Raspberry Pi 3, afim de por em prática parte do conteúdo da matéria de sistemas embarcados.

## II. INTRODUÇÃO

A música é uma forma de arte que se constitui na combinação de vários sons e ritmos, seguindo uma pré-organização ao longo do tempo [1].
Há evidências de que a música é conhecida e praticada desde a pré-história. Provavelmente a observação dos sons da natureza tenha despertado no homem, através do sentido auditivo, a necessidade ou vontade de uma atividade que se baseasse na organização de sons. Embora nenhum critério científico permita estabelecer seu desenvolvimento de forma precisa, a história da música confunde-se, com a própria história do desenvolvimento da inteligência e da cultura humana [2].
Através da necessidade do homem de desenvolver música foram criados instrumentos musicais dos mais diversos tipos. Os vários tipos de instrumentos podem ser classificados de diversas formas, sendo uma das mais comuns, a divisão de acordo com a forma pela qual o som é produzido, como por exemplo sopro, cordas e percussão.
O teclado musical é um "instrumento" moderno, derivado de instrumentos como o piano, o órgão, o cravo, dentre outros. Em 1874, Elisha Gray inventou o primeiro sintetizador o qual chamou de "The Telegraph Musical", ele era constituído por dois teclados com seus sons produzidos por fios telegráficos. [3]
A invenção de Bob Moog, o sintetizador Moog, foi exibido pela primeira vez em 1964. Tecnicamente, no entanto, não era um teclado, uma vez que não têm um teclado. Foi adicionado um teclado ao sintetizador em 1970 e, de lá, a explosão teclado elétrico começou. Outras empresas seguiram o exemplo e como o tempo passou mais avanços e aperfeiçoamentos foram feitos.[3]
Com o avanço dos computadores puderam ser adicionadas mídias a esses sintetizadores e eles passaram a ter muitos usuários, principalmente no Rock Progressivo nos
anos 70 e no pop eletrônico dos anos 80. Hoje com a variedade de timbres que eles possuem, são utilizados em praticamente todos os estilos musicais modernos.[3]
O teclado é reconhecido como um instrumento musical, mas na realidade, por não possuir um timbre natural, ele é um "equipamento" que possui muitos timbres de outros instrumentos ou sintetizados [3].

Tendo isso em vista, propõe-se implementar um teclado utilizado uma Raspberry Pi que terá no mínimo uma oitava para a escala, e as oitavas serão selecionadas a partir de um dos potenciômetros.

## III. REQUISITOS

- Requisitos Funcionais:
  - O teclado deve tocar no minimo 12 teclas com notas diferentes em cada uma;
  - Ter a possibilidade de tocar mais de uma tecla simutaneamente;
  - Interface para utilização do teclado;
  - Deve poder alterar o timbre das teclas;

- Requisitos Não-Funcionais:
  - O sistema deve ter baixa latência;
  - Alimentação para a Raspberry Pi;

## IV. BENEFÍCIOS

Será um teclado pequeno, o que vai facilitar o seu transporte. E mesmo sendo pequeno vai possibilitar tocar todas as notas musicas em diferentes frequências e timbres.

## V. DESENVOLVIMENTO

Neste tópico serão descritos o hardware e o software do teclado musical e também o motivo da escolha de cada um.

### *A. Descrição do hardware*

#### *a) Material Utilizado:*

- Raspberry Pi 3B;
- Arduino Nano;
- Protoboard;
- 12 resistores de 10KΩ
- 4 potenciômetros 10 KΩ;
- Fonte 5V - 3A;
- Caixa de MDF;
- Conector P2(macho)-P2(fêmea);
- Chave tipo gangorra;
- LED vermelho;
- Fones de Ouvido;
- Fios do tipo Jumper;
- 12 chaves tipo *pushbutton.*

#### *b) Diagrama de Conexão Completo:*

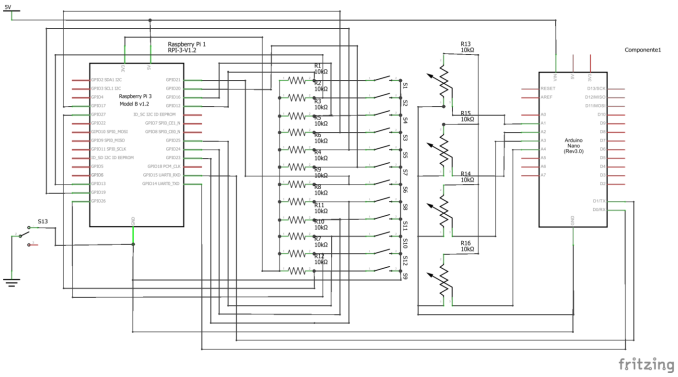


**Figura 1-** Diagrama completo de conexões

A conexão dos botões, aos GPIOs, foi realizada com resistor de *pull up*, ou seja, quando o botão é pressionado o valor que é obtido no pino a que ele está conectado é 0V (GND), e quando não está pressionado é 3.3V (VCC).

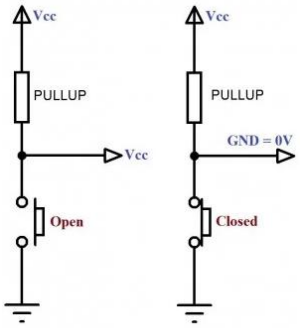


**Figura 2 -** Resistor Pull Up

Foi instalada uma chave do tipo gangorra para poder ligar e desligar a Raspberry Pi, a chave foi colocada entre duas pontas do fio GND, da fonte que foi conectada para liga-la, que foi cortado. Assim quando a chave era colocada na posição 'I' a passagem de corrente elétrica era liberada e quando na posição 'O' a corrente era cortada e a Raspberry de era deligada. Foi ainda colocado um LED vermelho que quando a corrente era cortada ficava acesso e quando era liberada o LED se apagava.

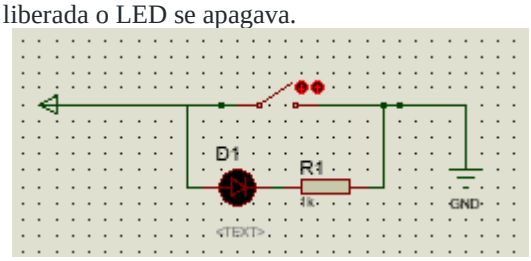


**Figura 3 -** Conexão Chave On-Off

Para leitura da variação de resistência dos potenciômetros foi utilizado o Arduino Nano, pois a Raspberry Pi não possui conversor analógico. Os potenciômetros foram conectados ao conforme a seguinte figura:

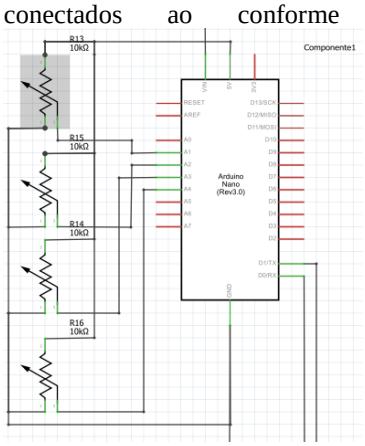


**Figura 4 -** Conexão dos Potenciômetros

Após montado o projeto ficou da seguinte da seguinte forma:

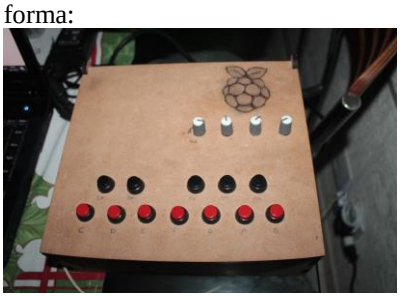

**Figura 5 -** Vista superior do teclado musical fechado

**Figura 6 -**Vista frontal

**Figura 7 -** Conexão da fonte de energia

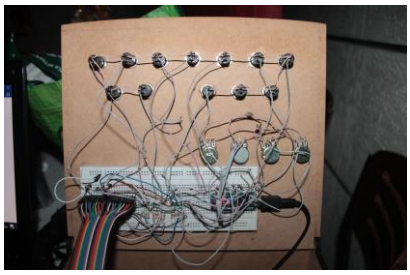

**Figura 8 -** Conexão dos Botões e Potenciômetros

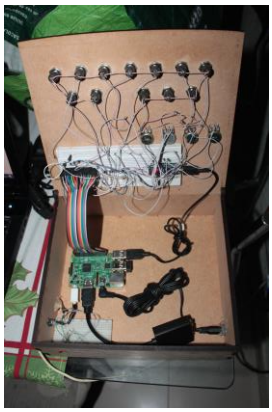

**Figura 9 -** Teclado Internamente

**Figura 10 -** Teclado desligado com LED aceso

*B. Descrição do Software*

O programa que controla o funcionamento do teclado definindo qual será a forma de onda a ser excutada cada vez que um botão ou mais for pressionado é explicado no fluxograma a da figura a seguir (está anexado ao fim do relatório para melhor visualização).
O código foi compilado com o seguinte comando no terminal da Raspberry:

*$ gcc -o Teclado_Musical Teclado_Musical.c -lpthread -lm -lao -lUUGear -lbcm283*

Para rodar o programa e para seu bom funcionamento foram necessárias alguma bibliotecas e funções as seguintes são algumas delas:

*1) Bcm2835:*A biblioteca bcm2835 foi desenvolvida para o Raspberry Pi e dá acesso ao GPIO e outras funções de entrada e saída do chip Broadcom BCM2835 que estão disponíveis no conector P1 da placa de desenvolvimento. Além de dar acesso aos pinos físicos do RPi, a biblioteca permite acesso aos timers do sistema. Interrupções ainda não são suportadas pela biblioteca e, por este motivo, eventos só podem ser detectados através de testes contínuos. Além disso, a biblioteca é compatível com a linguagem C++ e se instala como um arquivo header e uma biblioteca não compartilhada (*non-shared library*).
É com esta biblioteca que será realizada a leitura dos botões que estão configurados com resitores de *pull up*, logo quando o botão é pressionado o nível lógico do pino vai para 0 (LOW) e enquanto não é pressionado se mantém em 1 (HIGH) .

*2) Thread e Mutex: Thread* é a uma forma de um processor dividir a si mesmo em duas ou mais terefas que podem ser executadas concorrencialmente. Ou seja, consegue executar mais de um processo de forma simutânea. No caso deste trabalho serão executadas multiplas threads para que a leitura dos botões ocorram de forma mais rápida e para que a definição de qual nota deve ser tocada seja realiazada sem que precise esperar para que a outra termine de ser execuada, permitindo assim que o som de mais de um botão seja executado. Para que não haja conflito entre as threads de leitura e definição das notas a serem tocadas, foi necessário utilizar o *Mutex* que é uma variável que pode estar em um dos dois estados seguintes: impedido ou desimpedido. Outros termos também podem ser utilizados, tais como bloqueado ou desbloqueado, trancado ou destrancado, etc. Sempre que um recurso global for acessado por mais de uma thread, este recurso deve ter um mutex associado a ele. Pode-se, inclusive, aplicar um mutex a um determinado segmento de memória (uma região crítica, por exemplo) para protegê-la de outras *threads*.

*3) UUGear:* É um projeto que criou uma biblioteca que deve ser executada junto com um código carregado no Arduino. Esse projeto visa solucionar o problema de mudança de porta de comunicção do arduino com a Raspberry Pi, via UART, sempre que é reconectado, o que faz com que o código precise ser alterado. Quando o programa que vem junto com o arquivo .zip da bilbioteca é executado no arduino Nano já conectado a Raspberry, ele torna o Arduino um dispositivo UUGear que pode ser encontrado por um *id* fixo e exclusivo. A biblioteca do projeto também auxilia na comunicação entre os dois, pois

possui funções pré definidas com sintaxe muito parecida com utilizada pela IDE do Arduino.

*4) Ao.h:* A *libao* é uma é uma biblioteca de projetos de softwre livre, que permite a execução de audio em multiplataforma. É a partir dela que o áudio do teclado é executado.

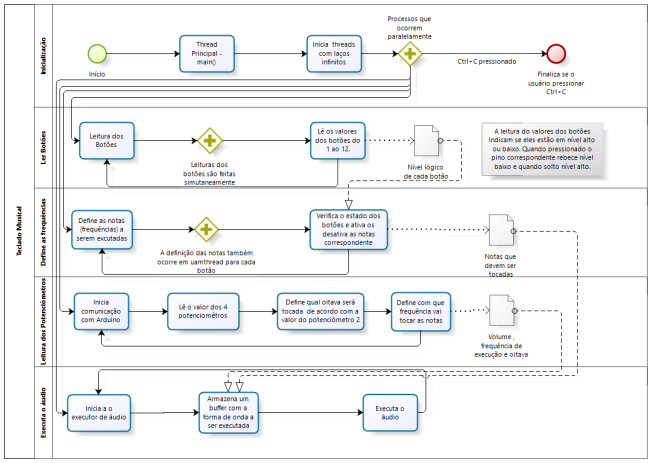


**Figura 11 -** Fluxograma de funcionamento do software

## VI. RESULTADO

Para o teste e validação do resultado do projeto foram pressionaods cada um dos botões separamente e verificado se o som que era executado era equivalente ao som da nota desejada em cada botão enquanto os potenciometros reponsáveis pelo volume, oitava e frequência eram modificados para verificar se estavam cumprindo com seus rewuisitos. O resultado foi satisfatório quando até um certo limite de volume no qual parecia ocorrer uma saturação na forma de onda de saída e o som tornavasse um ruído desagradável, foi possível notar a diferença entre as oito oitavas que o potenciômetro permitia escolher e também ficou evidente o com funcionamento do potenciômetro que controlava a frequência com que o som era executado.
Em seguida foi testado se os quando os botões eram pressionados silmutaneamente, dois a dois, em seguida três a três, era possível identificar o som das notas que estavam sendo pressionadas, ou se eram executadas de forma intermitente. Foi verificado que o som executado eram dos botões pressionados, mas ele não ficavam nítidos quando o volume era muito alto. E quando mais de 7 teclas foram pressionadas ao mesmo tempo, também pareceu haver saturação da forma de onda de saída.

## VII. CONCLUSÃO

O projeto tinha por objetivo a montagem de uma teclado musical digital pequeno com 12 teclas que permitisse que mais de uma oitava fosse tocada e permitisse que fossem criados timbres diferentes. Com o auxílio de uma Raspiberry Pi 3 e um arduino nano foi possível fabricar esse teclado com resultados satisfatórios com excessão da mudança de timbre, que pode ser verificada em outras fases do trabalho, mas ao fim dele ocorreu um defeito no potenciômetro que permitia a variação de uma das harmônicas da forma de onda de saída.
Assim foi possível também por em prática parte do conteúdo da matéria Sistemas Embarcados, utilizando o sistema Linux, linguagem C, processos (Threads), Mutex e comunicação UART.

## REFERÊNCIAS


[1] Dicionário Aurélio do Século XXI.

[2] Nattiez, Jean-Jacques (1990). Music and discourse: toward a semiology of music. [S.l.]: Princeton University Press. pp. 48, 55. ISBN 0691027145

[3] Júnior, L - Breve História do Teclado – 2013 - http://musicaplena.com/breve-historia-do-teclado. Acesso em 06/09/2017

## ANEXO 1

FLUXOGRAMA E ESQUEMÁTICO DE HARDWARE

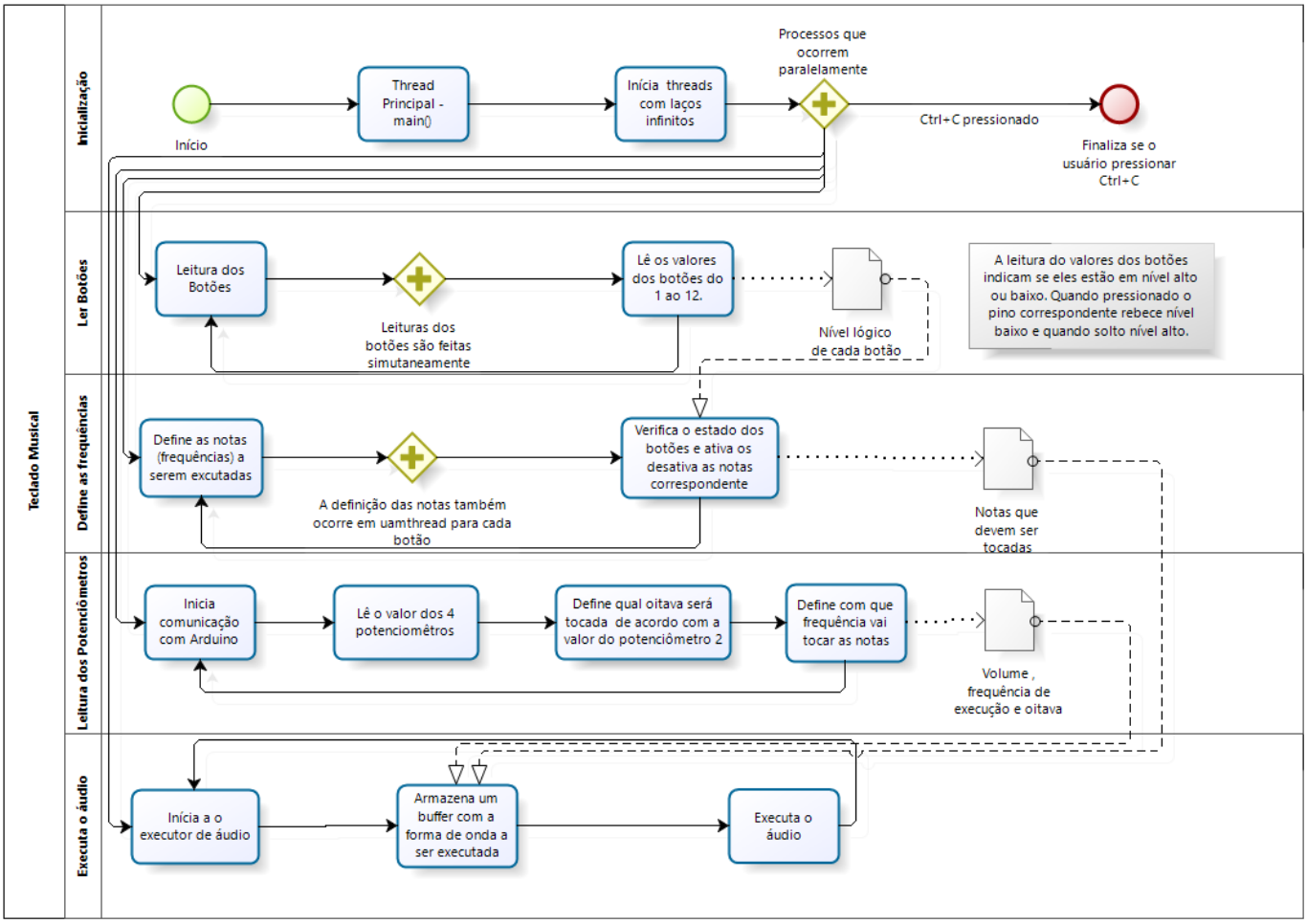


**Figura 12 - Fluxograma do Software**

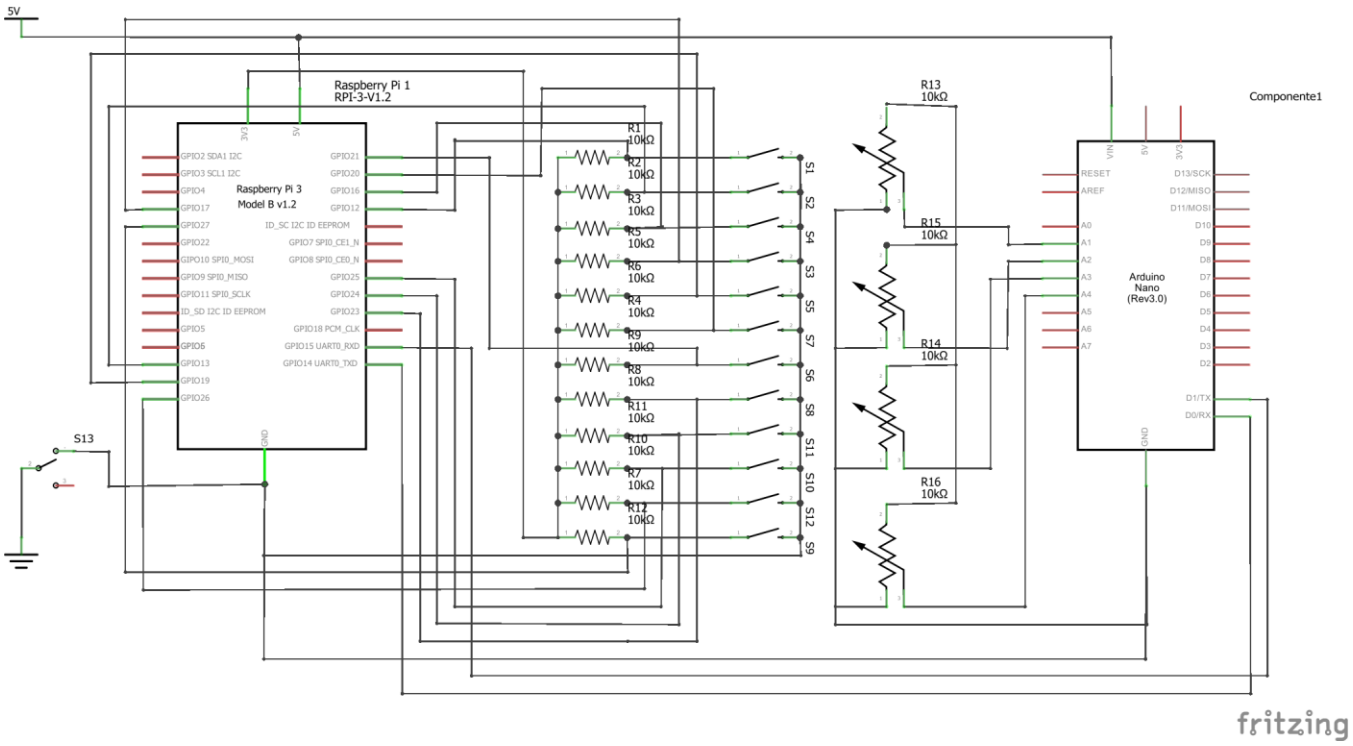


**Figura 13 - Diagrama do Hardware**

ANEXO 2

```
/*********************************************
Trabalho de Sistemas Embarcados
Teclado Musical Digital

Eduardo S. Sales Rodrigues 14/0038558
Data: 29/11/2017

para compilar digite no terminal
gcc -o Teclado_Musical Teclado_Musical.c -lm -lao -lpthread -lbcm2835 -lUUGear
*********************************************/


#include <stdlib.h>                 // biblioteca para usar exit()
#include <stdio.h>                  // biblioteca para usar fprintf()
#include <string.h>                 // biblioteca para usar strerror()
#include <pthread.h>                // biblioteca para usar threads
#include <unistd.h>                 // biblioteca para acessar a API POSIX
#include <bcm2835.h>                // biblioteca para controle do GPIO
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include <ao/ao.h>
#include <math.h>
#include <stdint.h>
#include <mqueue.h>

#include "UUGear.h"


/* Protótipo de Funções */

void *controlBt(void *p);
void *inputRead(void *p);
void *sound(void *p);
void *read_Pots(void *p);

/* Variáveis Globais */

static ao_device *sound_dev=NULL;

pthread_mutex_t lock;                                           //mutex
int value1, value2, value3, value4;             //variável para armazenar valor dos potenciometros
int f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12; //variável que definirá qual nota tocar
float d;                                                                        //variável que definirá a frequência de execução do som

double freq1,freq2,freq3,freq4,freq5,freq6,freq7,freq8,freq9,freq10,freq11,freq12; //frequêcias em Hertz de cada nota

int bt_state[12];                   //variável para armazenar estado dos botões

//*********************************************//
//*********      Pinos Utilizados  *************//
//*********************************************//
```

```
uint8_t sw_pins[12]={RPI_GPIO_P1_11,RPI_GPIO_P1_12,RPI_V2_GPIO_P1_13,RPI_GPIO_P1_15,RPI_GPIO_P1_16
,RPI_GPIO_P1_18,RPI_GPIO_P1_22,RPI_V2_GPIO_P1_29, RPI_V2_GPIO_P1_31,RPI_V2_GPIO_P1_33,
RPI_V2_GPIO_P1_35,RPI_V2_GPIO_P1_37/*, RPI_GPIO_P1_22, RPI_V2_GPIO_P1_29*/};

/*************************************************************
 * Função principal (thread principal)
 *************************************************************/
int main(int argc, char **argv){

	pthread_t threads_bt[12], threads_sw[12], thread_pot, thread_sound/*,threads_sound[2]*/;


	int i, ret, num[12]={1,2,3,4,5,6,7,8,9,10,11,12}; //Variáveis do para identificação das Threads
	value1 = 1;
	value2 = 1;
	value3 = 1;
	value4 = 1;

	// Inicializa mutex
	if( pthread_mutex_init(&lock, NULL) != 0 ){
		fprintf(stderr, "Erro ao inicializar mutex.\n");
		exit(EXIT_FAILURE);
	}
	// Iniciliza biblioteca bcm2835
	if( !bcm2835_init() ){
		fprintf(stderr, "Erro ao inicializar biblioteca bcm2835.\n");
		exit(EXIT_FAILURE);
	}

	// Seleciona pinos de entrada para push buttons
	// Desabilita resistores de pull up/down nesses pinos, porque foi feito externamente
	for(i=0; i<12; i++){
		bcm2835_gpio_fsel(sw_pins[i], BCM2835_GPIO_FSEL_INPT);
		bcm2835_gpio_fsel(sw_pins[i], BCM2835_GPIO_PUD_OFF);
	}

	//Cria Thread para leitura dos potenciometros
		ret = pthread_create(&thread_pot, NULL, &read_Pots, NULL);
			if(ret != 0){
			  fprintf(stderr, "Erro thread read_Pots %d.Código %d: %s\n", (i+1), ret, strerror(ret));
			  exit(EXIT_FAILURE);
		}
	//Cria Thread para gerar o som
		ret = pthread_create(&thread_sound, NULL, &sound, NULL);
			if(ret != 0){
			  fprintf(stderr, "Erro thread Sound %d. Código %d: %s\n", (i+1), ret, strerror(ret));
			 exit(EXIT_FAILURE);
		}
	// Laço para criar 24 threads
	for(i=0; i<12; i++){

		// Cria thread para ligar/desligar LEDs
		// Repassa um número correspondente (1 e 2)
		ret = pthread_create(&threads_bt[i], NULL, &controlBt, &num[i]);
		if(ret != 0){
			fprintf(stderr, "Erro thread LED %d. Código %d: %s\n", (i+1), ret, strerror(ret));
			exit(EXIT_FAILURE);
```

```
                }
                // Cria thread para ler estado dos botões
                // Repassa um número correspondente (1 e 2)
                ret = pthread_create(&threads_sw[i], NULL, inputRead, &num[i]);
                if(ret != 0){
                        fprintf(stderr, "Erro thread SW %d. Código %d: %s\n", (i+1), ret, strerror(ret));
                        exit(EXIT_FAILURE);
                }
        }
        // Aguarda as threads terminarem
        for(i=0; i<12; i++){
                pthread_join(thread_pot, NULL);
                pthread_join(thread_sound, NULL);
                pthread_join(threads_bt[i], NULL);
                pthread_join(threads_sw[i], NULL);
        }
        // Finaliza biblioteca bcm2835
        bcm2835_close();
        //Destrói o mutex
        pthread_mutex_destroy(&lock);

        return 0;
}

/*************************************************************************
 * Função responsável por controlar as notas que serão tocadas
 *************************************************************************/
void *controlBt(void *p){
        // Faz o cast do ponteiro para voltar a ser um inteiro
        int *num = (int *)p;

        //Loop infinito
        while(1){
                // Trava o mutex antes de acessar variável led_state
                pthread_mutex_lock(&lock);
                // Atualiza estado dos f's para definir se toca ou não cada nota
                if( bt_state[*num-1] ){//Se o botão 'num-1' não for pressionado mantém sua nota correspondente não sendo
tocada
switch(*num){
        case 1:
                f1=0;
        break;
        case 2:
                f2=0;
        break;
        case 3:
                f3=0;
        break;
        case 4:
                f4=0;
        break;
        case 5:
                f5=0;
        break;
        case 6:
                f6=0;
        break;
```

```
        case 7:
                f7=0;
        break;
        case 8:
                f8=0;
        break;
        case 9:
                f9=0;
        break;
        case 10:
                f10=0;
        break;
        case 11:
                f11=0;
        break;
        case 12:
                f12=0;
        break;
}
                }
                else{ //Se for pressionado, toca a nota correspondente
        switch(*num){
        case 1:
                f1=1;
        break;
        case 2:
                f2=1;
        break;
        case 3:
                f3=1;
        break;
        case 4:
                f4=1;
        break;
        case 5:
                f5=1;
        break;
        case 6:
                f6=1;
        break;
        case 7:
                f7=1;
        break;
        case 8:
                f8=1;
        break;
        case 9:
                f9=1;
        break;
        case 10:
                f10=1;
        break;
        case 11:
                f11=1;
        break;
        case 12:
                f12=1;
```

```
		break;
}
}
			// Libera o mutex
			pthread_mutex_unlock(&lock);
			// Pequeno delay
			bcm2835_delay(1);
		}
		pthread_exit(NULL);
}

/*****************************************************************
 * Função responsável pela leitura dos botões
 *****************************************************************/
void *inputRead(void *p){
		// Faz o cast do ponteiro para voltar a ser um inteiro
		int *num = (int *)p;

		//Loop infinito
		while(1){
			// Trava o mutex antes de acessar variável led_state
			pthread_mutex_lock(&lock);
			// Atualiza estado do botão na variável
			led_state[*num-1] = bcm2835_gpio_lev(sw_pins[*num-1]);
			// Libera o mutex
			pthread_mutex_unlock(&lock);
			// Pequeno delay
			bcm2835_delay(50);
		}
		pthread_exit(NULL);
}
/*****************************************************************
 * Função responsável pelo som
 *****************************************************************/

void* sound(void *p){
  static ao_sample_format format;
  static double sample_period;
while(1){
  double duration = d;

  int VOL=value3;
  int har3=value4;
  int VOL2=value1;

			const double w1=2*M_PI*freq1;
			const double w2=2*M_PI*freq2;
			const double w3=2*M_PI*freq3;
			const double w4=2*M_PI*freq4;
			const double w5=2*M_PI*freq5;
			const double w6=2*M_PI*freq6;
			const double w7=2*M_PI*freq7;
			const double w8=2*M_PI*freq8;
			const double w9=2*M_PI*freq9;
			const double w10=2*M_PI*freq10;
			const double w11=2*M_PI*freq11;
			const double w12=2*M_PI*freq12;
```

```
 if(!sound_dev){
   int default_driver;

   ao_initialize();
   default_driver = ao_default_driver_id();

   memset(&format, 0, sizeof(format));
   format.bits = 16;
   format.channels = 2;
   format.rate = 44100;
   format.byte_format = AO_FMT_BIG;

   sound_dev = ao_open_live(default_driver, &format, NULL);
   if(!sound_dev){
     fprintf(stderr, "Error opening sound device.\n");
     exit(1);
   }

   sample_period=1./(double)format.rate;
 }

 size_t n_frames=(size_t)(duration*format.rate);
 uint16_t *wave= (malloc(format.channels*n_frames*sizeof *wave));
 for(size_t i=0; i<n_frames; i++){

          double wt1=f1*w1*i*sample_period;
          double wt2=f2*w2*i*sample_period;
          double wt3=f3*w3*i*sample_period;
          double wt4=f4*w4*i*sample_period;
          double wt5=f5*w5*i*sample_period;
          double wt6=f6*w6*i*sample_period;
          double wt7=f7*w7*i*sample_period;
          double wt8=f8*w8*i*sample_period;
          double wt9=f9*w9*i*sample_period;
          double wt10=f10*w10*i*sample_period;
          double wt11=f11*w11*i*sample_period;
          double wt12=f12*w12*i*sample_period;

          int16_t frequencia1 =((10*VOL+500)*(sin(wt1)+sin(3*wt1)*0.3333333+cos(5*wt1)*.2));
          int16_t frequencia2 =((10*VOL+500)*(sin(wt2)+sin(3*wt2)*0.3333333+cos(5*wt2)*.2));
          int16_t frequencia3 =((10*VOL+500)*(sin(wt3)+sin(3*wt3)*0.3333333+cos(5*wt3)*.2));
          int16_t frequencia4 =((10*VOL+500)*(sin(wt4)+sin(3*wt4)*0.3333333+cos(5*wt4)*.2));
          int16_t frequencia5 =((10*VOL+500)*(sin(wt5)+sin(3*wt5)*0.3333333+cos(5*wt5)*.2));
          int16_t frequencia6 =((10*VOL+500)*(sin(wt6)+sin(3*wt6)*0.3333333+cos(5*wt6)*.2));
          int16_t frequencia7 =((10*VOL+500)*(sin(wt7)+sin(3*wt7)*0.3333333+cos(5*wt7)*.2));
          int16_t frequencia8 =((10*VOL+500)*(sin(wt8)+sin(3*wt8)*0.3333333+cos(5*wt8)*.2));
          int16_t frequencia9 =((10*VOL+500)*(sin(wt9)+sin(3*wt9)*0.3333333+cos(5*wt9)*.2));
          int16_t frequencia10 =((10*VOL+500)*(sin(wt10)+sin(3*wt10)*0.3333333+cos(5*wt10)*.2));
          int16_t frequencia11 =((10*VOL+500)*(sin(wt11)+sin(3*wt11)*0.3333333+cos(5*wt11)*.2));
          int16_t frequencia12 =((10*VOL+500)*(sin(wt12)+sin(3*wt12)*0.3333333+cos(5*wt12)*.2));

          int16_t
sample=(int16_t)(frequencia1+frequencia2+frequencia3+frequencia4+frequencia5+frequencia6+frequencia7+frequencia8+freque
ncia9+frequencia10+frequencia11+frequencia12);
          for(int channel=0; channel<format.channels; channel++)
    wave[i*format.channels+channel]=sample;
```

```
  }

  ao_play(sound_dev, (char *)wave, format.channels*n_frames*sizeof *wave);

  free(wave);
        }
        pthread_exit(NULL);
}

void *read_Pots(void *p)
{
        int O; //Variável que define a oitava
        float C,C_s,D,D_s,E,F,F_s,G,G_s,A,A_s,B; //Variável que a armazena as frequencias bases de cada nota

        O=1;

        C= 32.703196;
        C_s=34.647829;
        D=36.708096;
        D_s=38.890873;
        E=41.203445;
        F=43.653529;
        F_s=46.249303;
        G=48.999429;
        G_s=51.913087;
        A=55;
        A_s=58.27047;
        B=61.735413;


        setupUUGear();

        setShowLogs(1);

        UUGearDevice dev = attachUUGearDevice ("UUGear-Arduino-3348-6249");

        if (dev.fd != -1)
        {
                int pin1 = 1;       //analog input pin 1
                int pin2 = 2;       //analog input pin 2
                int pin3 = 3;       //analog input pin 3
                int pin4 = 4;       //analog input pin 4

                for(;;)
                {
                        value1 = analogRead(&dev, pin1);
                        usleep(10);
                        value2 = analogRead(&dev, pin2);
                        usleep(10);
                        value3 = analogRead(&dev, pin3);
                        usleep(10);
                        value4 = analogRead(&dev, pin4);
                        usleep(10);

                        if(value2<=128){
                                O=256;}
                        if(value2>128 && value2<=256){
```

```
                        O=128;}
                    if(value2>256 && value2<=384){
                        O=64;}
                    if(value2>384 && value2<=512){
                        O=32;}
                    if(value2>512 && value2<=640){
                        O=16;}
                    if(value2>640 && value2<=768){
                        O=8;}
                    if(value2>768 && value2<=896){
                        O=4;}
                    if(value2>896){
                        O=2;}

                    if(value4<=128){
                        d=0.05;}
                    if(value4>128 && value4<=256){
                        d=0.07;}
                    if(value4>256 && value4<=384){
                        d=0.1;}
                    if(value4>384 && value4<=512){
                        d=0.13;}
                    if(value4>512 && value4<=640){
                        d=0.15;}
                    if(value4>640 && value4<=768){
                        d=0.17;}
                    if(value4>768 && value4<=896){
                        d=0.2;}
                    if(value4>896){
                        d=0.23;}

                freq1=O*C;
                freq2=O*D;
                freq3=O*E;
                freq4=O*F;
                freq5=O*G;
                freq6=O*A;
                freq7=O*B;
                freq8=O*C_s;
                freq9=O*D_s;
                freq10=O*F_s;
                freq11=O*G_s;
                freq12=O*A_s;

}
                detachUUGearDevice (&dev);
        }
        else
        {
                printf("Can not open UUGear device.\n");
        }

        cleanupUUGear();
        pthread_exit(NULL);
        return NULL;

}
```