# Pedaleira Digital de Efeitos Para Guitarras

Gabriel da Silva Soares
Faculdade do Gama
Universidade de Brasília
Gama - DF, Brasil

Ricardo Vieira Borges
Faculdade do Gama
Universidade de Brasília
Gama - DF, Brasil

## I. RESUMO

O projeto consiste no desenvolvimento de uma pedaleira digital de efeitos para guitarras, utilizando a placa Raspberry-Pi como núcleo de processamento, adquirindo, convertendo e editando as formas de onda a fim de adicionar efeitos ao sinal oriundo da guitarra. A ideia é de ter efeitos clássicos disponíveis, além de um efeito não comum criado pelo grupo, sendo possível misturar os efeitos de acordo com a necessidade do usuário.

## II. INTRODUÇÃO

### 2.1. Pedaleira Digital

A pedaleira de guitarra é um equipamento eletrônico que é usado para alterar o som natural de uma guitarra elétrica. São utilizados durante concertos ao vivo e também em estúdios. Exemplos de efeitos presentes nesses equipamentos são Distortion, Overdrive, Wah-wah, Reverb, Delay etc.

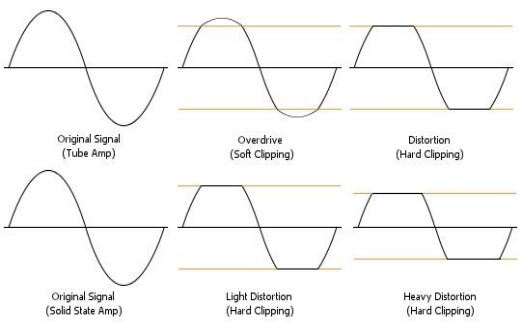


Figura 1 - exemplo de edição de sinal por clipagem (efeito Overdrive).

As pedaleiras geralmente ficam no chão próximo ao músico e são conectados diretamente ao seu instrumento e são acionados pelo pé do músico que pressiona um interruptor. Pedaleiras podem ser compostas por uma série de pedais analógicos organizados em uma caixa com um conjunto de ligações específicas, cada pedal com seu efeito característico, podendo ser realizada a mistura de efeitos acionando mais de um pedal, esse tipo de conjunto recebe o sinal da guitarra, usa um estágio de amplificação e modifica esse sinal analogicamente utilizando circuitos. Outra forma de se implementar uma pedaleira, é por microcontrolador ou microprocessador, onde o sinal da guitarra é amplificado e adquirido por um conversor analógico-digital, esse sinal digital é editado pelo computador, adicionando o efeito desejado e o transformando de volta para analógico, seja por saída de aúdio, geradores de PWM ou conversores digital-analógico.

A grande vantagem da utilização de uma pedaleira digital é a da possibilidade de uma gigantesca gama de efeitos e combinações do mesmo, dependendo da capacidade de processamento, utilizando pouco espaço e com baixo custo.

### 2.2. Conversores Analógico-Digital

O conversor analógico-digital (A/D ou ADC) é um dispositivo capaz de gerar uma representação digital (palavra de bits) a

partir de um sinal analógico, normalmente representado por um nível de tensão. É muito utilizado quando é necessária a comunicação entre um dispositivo digital (microprocessador, microcontrolador, FPGA e etc) e um dispositivo que manda informações em forma de nível de tensão (sensores, comparadores e etc).

Há duas definições cruciais no desempenho de um ADC, a resolução e a taxa de amostragem. Sendo a resolução o que afeta na precisão do ADC (ex: resolução de 12-bits, consegue representar até 4095 valores binários de uma faixa de tensão), quanto maior a resolução, maior a quantidade de representações do sinal e menor o erro. A taxa de amostragem  implica na frequência máxima de sinal que o ADC consegue converter sem perder informação.

## 2.3. Comunicação SPI

Comunicação serial síncrona de 3 fios com conceito de mestre/escravo, ou seja, a comunicação é feita baseada em um sinal de clock gerado pelo dispositivo mestre, que envia ou recebe os dados do escravo, cada um desses dados é enviado de forma independente através de um dos fios, MOSI (master output, slave input) e MISO (master input, slave output), também é adicionado mais um fio para cada escravo presente no sistema, denominado SLAVE SELECT.

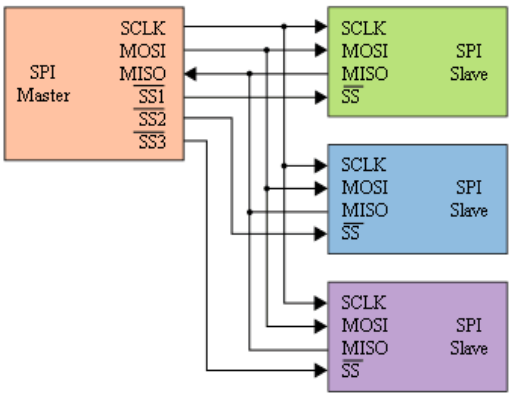


Figura 2 - diagrama de comunicação SPI.

## 2.4. Amplificadores de Tensão

Circuito com amplificador operacional para obter um ganho de tensão de acordo com os valores dos componentes utilizados, esse ganho de tensão é limitado pela tensão de alimentação do AmpOp.

Abaixo um esquema com amplificador operacional com malha de realimentação e ganho de tensão dado por:

$$A = \frac{R1}{R2} + 1$$

portanto:

$$Vout = A * Vin$$

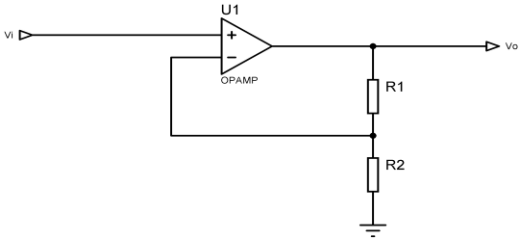


Figura 3 - Exemplo de amplificador de tensão não inversor.

## 2.5. Raspberry Pi 3 Model B

Computador de placa única com as seguintes especificações:

- CPU Quad Core 1.2GHz.
- 1GB de RAM.
- Conexões Wi-fi e Bluetooth.
- Ethernet.
- 4 Portas USB 2.0.
- Saída HDMI.
- **Saída estéreo de 4 polos (p/ áudio ou vídeo).**
- 40 Pinos GPIO
- Entrada p/ cartão Micro SD.
- Entrada micro-USB p/ fonte.
- Entradas e saídas p/ periféricos Pi (tela touch screen, câmera e etc).

## 2.6. Tiva-C Séries TM4C123GH6PM

Launchpad da Texas Instruments com as seguintes especificações:

- ARM Cortex-M4 32-bit 80MHz.

- 256KB Flash, 32KB SRAM, 2KB EEPROM.
- Comunicações seriais disponíveis: 8 UART, 6 I2C, **4 SPI**.
- **2 ADC's 12-bit 1 MSPS**.
- Duas entradas micro USB, p/ debug e alimentação.

## III. DESENVOLVIMENTO

Para esse projeto deve-se seguir algumas etapas básicas:

- Amplificar sinal da guitarra, até obter a mesma escala da faixa de tensão suportada pela entrada do ADC.

- Gerar uma tensão de offset em série com a tensão amplificada da guitarra, p/ elevar a mesma até não haver tensões negativas.

- Realizar conversão analógico-digital.

- Processar sinal digital via software, dependendo do efeito desejado.

- Converter sinal digital em analógico novamente, para reprodução em um alto-falante.

A solução proposta é de utilizar um amplificador operacional com ganho específico (2.4) para amplificar o sinal da guitarra e elevar esse sinal (offset) utilizando um potenciômetro ligado a uma fonte de tensão, assim teremos um offset regulável de acordo com a necessidade. Após isso essa tensão deve ser convertida utilizando o conversor A/D 12-bits da placa Tiva-C (2.6) e a comunicando com a Raspberry-Pi (2.5) via SPI (2.3), observando os tempos de conversão e comunicação para gerar uma taxa de amostragem correta para a aplicação. Após receber os dados, a Raspberry-Pi trabalha esses dados via software com operações matemática e/ou condições de loops para gerar os efeitos desejados, disponibilizando por fim esses dados em sua saída P2, para gerar um sinal de áudio mono.

Como o sinal de saída de uma guitarra com captadores passivos estão em escala de mV, optou-se por um ganho A = 11(R1 = 10kΩ e R2 = 1kΩ) com offset de 1.6V para que o sinal fique adequado para a entrada do conversor A/D da Tiva-C (0 a 3.3V).

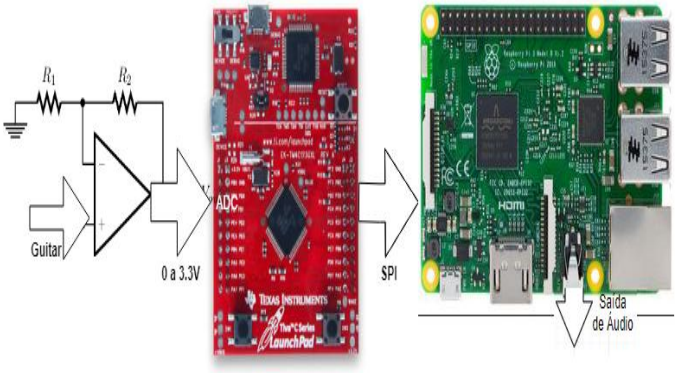


Figura 4 - esquemático de hardware do projeto.

Tabela 1 - materiais utilizados.

| Componente | Preço |
|---|---|
| Placa Raspberry-Pi 3 Model B | R$ 150,00 |
| Tiva-C tm4c123gh6pm | R$ 40,00 |
| Resistores (1kΩ, 10kΩ) | R$ 0,50 |
| Potênciômetro 10kΩ | R$ 0,50 |
| TL072 (ampop) | R$ 1,80 |
| Cabos P10 | R$ 15,00 |
| Cabos P2 | R$ 5,00 |
| Adaptador P2-P10 | R$ 8,00 |

Em primeiro lugar foi trabalhado o programa para que a placa Tiva-C realizasse a conversão A/D e enviasse os valores via SPI como escravo, além de configurar um timer para que o envio de dados seja sincronizado.

No código disponível (ANEXO I), primeiro é configurado o conversor analógico digital, definindo os valores nos registradores de configuração do ADC0, fazendo o mesmo trabalhar com um sample de 1Mpbs, o que resulta em uma velocidade de conversão de 83.33kHz para os 12 bits.

Após isso é configurado o módulo SPI da placa, como escravo e definido para enviar 8-bits por comunicação.

A função de conversão A/D espera o dado ser convertido e o salva em uma variável de 12-bits, que é enviada para a função de separação de bits, que separa a palavra de 12-bits em duas de 8-bits, que serão enviadas uma de cada vez pelo SPI.

O timer foi configurado para entrar nessas funções a cada 50us, o que gera uma frequência de amostragem de 20kHz.

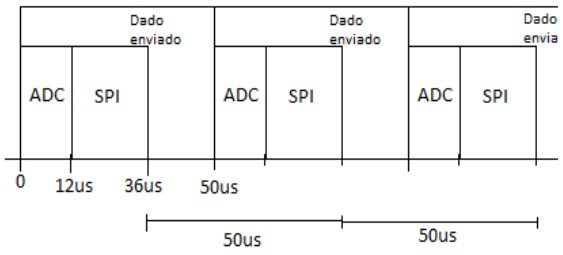


Figura 5 - Temporização dos dados enviados, com clk do SPI a 500kHz e ADC a 83.3kHz.

No código disponível (ANEXO II) são inicializadas as bibliotecas a serem utilizadas, a parte de SPI foi configurada apenas para receber os dados do escravo, sem enviar nada, o SPI recebe os 8-bits de dados da Tiva-C, o salva em uma variável, e recebe outro dado de 8-bits (resto da conversão de 12-bits), desloca esse dado e junta com o dado anterior em um buffer. Esse buffer é colocado para rodar na saída P2 com a biblioteca ALSA.

## IV. RESULTADOS

Conversão A/D:

A utilização de uma placa externa para conversão AD exigiu uma demanda maior de estudos, já que foi utilizada uma Tiva-C. Os resultados obtidos na conversão foram muito coerentes e, como o conversor da placa só operava em altas velocidades, sendo a mínima de 125ksps, foi necessário utilizar uma sub-rotina para reduzir a taxa de amostragem para 20ksps. Essa taxa foi reduzida para que houvesse tempo para a comunicação SPI ocorrer.

Comunicação SPI:

A comunicação escolhida foi a SPI justamente pela sua velocidade em vista da comunicação UART e pela sua maior simplicidade, em vista da comunicação I²C. Na *raspberry*, utilizou-se o arquivo SPIDEV para configurar a comunicação, que inicialmente foi ajustada para trabalhar com um *clock* de 500kHz. Tudo ocorreu bem, porém, quando se optou por aumentar a frequência do *clock* para algo em torno de 1MHz, foi como se não houvesse nenhuma alteração na frequência, mantendo-se essa em 500kHz ainda.

Real time:

Foi feito um preenchimento de um buffer com os valores da conversão e depois que este estava preenchido, ele era percorrido para tocar a música em questão. Dessa forma, o projeto não estava em tempo real.

Conversão D/A:

A conversão de digital para analógico *a priori* seria feita por um chip externo. Porém, descobriu-se a biblioteca ALSA, que permite usar o próprio hardware de PCM da raspberry para efetuar a conversão. A configuração da ALSA foi efetuada após algumas referências e não retornou bom resultado. Percebeu-se que 12 bits não geravam variação suficiente para externalizar o som, então deslocou-se os valores da conversão para que estes se tornassem mais significativos. Após isso, percebeu-se que um dos parâmetros da configuração estava incorreto. O parâmetro era o tipo de variável, que estava em FLOAT, e então utilizou-se UNSIGNED de 16 bits. Depois dessa correção, foi possível tocar mesmo com os 12 bits, sem necessidade de se deslocar. Porém, deslocando-se 4 bits o volume foi mais perceptível.

Conjunto:

Quando se colocava um sinal na entrada do conjunto, esperava-se obter o mesmo sinal na saída, porém, havia um grande ruído. Utilizando o osciloscópio foi possível perceber que o ruído era inserido pelo potenciômetro que gerava o offset na entrada. Depois de corrigido esse problema, ainda havia outro: O som parecia ser quebrado em um determinado intervalo. A cada T segundos ouvia-se uma descontinuidade no som, mesmo que esse fosse uma senóide pura. Esse problema não foi corrigido. Além disso, foi setada uma frequência de amostragem de 20kHz no conversor A/D, porém, a frequência de amostragem vista na rasp quando se tocava o som era de aproximadamente 8kHz.

## V. CONCLUSÃO

Dos 5 requisitos de o sistema deveria cumprir, 3 foram cumpridos. Dessa forma, foi impossibilitado o funcionamento do sistema como um todo, porém, houve ainda grande avanço no uso do hardware de PCM da placa. Entretanto, nota-se que o tempo disponível, em face à quantidade de estudos a serem executados, foi um fator limitante. O projeto acabou demandando mais do que o esperado.

## VI. REVISÃO BIBLIOGRÁFICA


[1] Mitch Gallagher. *Guitar Tone: Pursuing the Ultimate Guitar Sound*. Cengage Learning; p. 81.

[2] A. W. FRANÇA. *Uso de Processamento Digital de Áudio na Implementação de Efeitos em Instrumentos Musicais*. Julho de 2015. http://bdm.unb.br/bitstream/10483/13268/1/2015_AndreWagnerFranca.pdf

[3] T. Jeff. *Introduction to sound programming with ALSA*. Setembro de 2004 *https://www.linuxjournal.com/article/6735?page=0,1*

[4] Alsa. *PCM (Digital Audio) Interface*. http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html

[5] Documentação da spidev.

https://www.kernel.org/doc/Documentation/spi/spidev

[6] A. Paterniani. *spidev code example.*

https://elixir.bootlin.com/linux/v3.5/source/include/linux/spi/spidev.h#L114

[7] Jonatham W. Valvano. Design de Sistemas Embarcados. Disponível em: <http://users.ece.utexas.edu/~valvano/>

ANEXO I

```
 1  #include <stdint.h>
 2  #include <stdlib.h>
 3  #include "TM4C123.h"
 4
 5  //PE2 -> entrada analÃ³gica
 6  //PA2 -> SSIClk
 7  //PA3 -> slave select
 8  //PA4 -> rx
 9  //PA5 -> tx MOSI
10
11  void ADC_config()
12  {
13      //configurando ADC
14      SYSCTL->RCGCGPIO |= 0x0010;          //habilitando clk em PE
15      GPIOE->DIR &= ~(0x0004);             //porta PE2(ADCin) como
entrada
16      GPIOE->AFSEL |= 0x0004;              //funÃ§Ã£o alternativa no
pino PE2
17      GPIOE->DEN &= ~(0x0004);             //desabilita funÃ§Ã£o
digital no pino PE2
18      GPIOE->AMSEL |= 0x0004;              //habilita funÃ§Ã£o
analÃ³gica no pino PE2
19
20      SYSCTL->RCGCADC |= 0x0001;           //habilita ADC0
21      SYSCTL->RCGC0 |= 0x10000|(0x03<<8);//habilita clk para ADC0,
mantÃ©m MAXADC0SPD em 00 p/ taxa de amostragem de 125kHz
22
23      ADC0->SSPRI = 0x0123;                //define SS3 como
sequenciador com alta prioridade
24      ADC0->ACTSS &= ~(0x0008);            //desabilitar ASEN3 para
configurar
25      ADC0->EMUX |= 0xF000;                //start do sequenciador
(modo contÃ-nuo)
26      ADC0->SSMUX3 &= 0x000F;              //clear
27      ADC0->SSMUX3 += 1;                   //set Ain1 (PE2)
28      ADC0->SSCTL3 = 0x0006;               //habilitando IE0 e END0
29      ADC0->ACTSS |= 0x0008;               //finalmente habilita
ASEN3 (sequenciador)
30  }
31  void SSI_config()
32  {
33      //configurando SSI
34      SYSCTL->RCGC1 |= (1<<4);
35      SYSCTL->RCGC2 |= (1<<0);
36      SYSCTL->RCGCGPIO |= (1<<0);          //habilitando clk em PA
37      SYSCTL->RCGCSSI |= (1<<0);           //ativando SSI0
38      while((SYSCTL->PRGPIO & 0x0001)==0){};//aguarda PortA ser
ativada
39      GPIOA->AFSEL |= 0x003C;              //funÃ§Ã£o alternativa nos
pinos SSI
40      GPIOA->PCTL &= ~0x222200;            //clear
41      GPIOA->PCTL |= 0x222200;             //especificar funÃ§Ã£o dos
pinos (tabela 23-5)
42      GPIOA->AMSEL &= ~(0x003C);           //desabilita funÃ§Ã£o
```

```
analÃ³gica
 43      GPIOA->DEN |= (1<<2)|(1<<3)|(1<<4)|(1<<5);//habilitando
funÃ§Ã£o digital em todos os pinos SSI
 44
 45      SSI0->CR1 &= ~(0x0002);             //desabilitando SSE p/
configurar
 46      SSI0->CR1 |= 0x0004;                //define como escravo c/
saÃ-da habilitada
 47      //SSI0->CR1 |= 0x0000;              //master
 48      SSI0->CC = 0x0005;                  //16MHz (sÃ³ p/ modo
master)
 49      SSI0->CPSR &= ~(0x00FF);
 50      SSI0->CPSR |= 0x0001;               //prÃ©-escala divisora do
clk, nÂº de 2 a 254 (16MHz/160 = 100kHz)
 51      SSI0->CR0 |= 0x0007;                //definir SCR=0, SPH/SPO =
0/0, freescale SPI, 8bit data.
 52    SSI0->CR1 |= 0x0002;                  //ativar SSI
 53
 54  }
 55
 56  void send_data(int data)    //envia 8-bits de dados
 57  {
 58
 59          SSI0->DR = data;
 60          while((SSI0->SR & (1<<0))==0){};  //aguarda transmissÃ£o
SPI
 61
 62  }
 63  void send_byte(int bits12)    //separa o dado de 12-bits em 2 de
8-bits, manda um de cada vez
 64  {
 65      int bits8;
 66
 67        bits8 = bits12 & 0xFF;     //primeiro byte
 68          send_data(bits8);          //envia primeiro byte p/
funÃ§Ã£o SPI
 69
 70          bits8 = (bits12>>8)&0x0F;  //segundo byte
 71          send_data(bits8);          //envia segundo " " " "
 72  }
 73  int convert_write(void)
 74  {
 75    int ADC;                                //p/ salvar conversÃ£o de
12-bits
 76
 77      ADC0->PSSI = 0x0008;                //inicia sequenciador SS3
 78
 79      while((ADC0->RIS & 0x0008)==0){};   //aguada conversÃ£o
 80
 81      ADC = ADC0->SSFIFO3;                //salva conversÃ£o em
variÃ¡vel
 82      ADC0->ISC = 0x0008;
 83
 84    send_byte(ADC);                       //envia dado p/
separaÃ§Ã£o
 85      return ADC;
 86  }
 87  int main()
 88  {
 89
 90      ADC_config();     //configuraÃ§Ã£o do ADC
```

```
 91      SSI_config();     //configuraÃ§Ã£o do SPI
 92
 93
 94      SYSCTL->RCGCTIMER |= (1<<0);          //ativando TIMER A0
 95      TIMER0->CTL &= ~(1<<0);               //desabilita timer p/
configurar
 96      TIMER0->CFG = 0x00000000;
 97      TIMER0->TAMR |= 0x02;                 //timer em modo
periÃ³dico
 98      TIMER0->TAMR |= (1<<4);               //count up
 99      TIMER0->TAILR = 0x4E2;                //conta atÃ© 1250 com
clk de 25MHz, perÃ-odo de 50us, 20kHz
100     TIMER0->CTL |= (1<<0);                //ativando timer
101
102     while(1)
103     {
104             if((TIMER0->RIS & 0x00000001) == 1){//condiÃ§Ã£o com
a flag do timer p/ entrar na funÃ§Ã£o
105                                           //entra na funÃ§Ã£o de
enviar/converter a cada 50us
106           TIMER0->ICR |= (1<<0);
107             convert_write();            //entra nas funÃ§Ãµes de
converter, separar bytes e enviar SPI
108             }
109     }
110  }
111
```

ANEXO II

```
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <linux/spi/spidev.h>
#include <signal.h>
#include <wiringPi.h>
#include <alsa/asoundlib.h>
#include <alsa/pcm.h>
#define BUFFER_LEN 50000

/*SPI0: MOSI > GPIO10
 *         MISO > GPIO9
 *         SCLK > GPIO11
 *         CE0  > GPIO8
 *         CE1  > GPIO7
*/

/* driver do SPI suporta as seguintes velocidades:
 * 3.9 MHz
 * 1953 KHz
 * 976 KHz
 * 488 KHz
*/

//variÃƒÂ¡veis globais
int spi_fd;     //descritor de arquivo do SPI
int SL = 4; //slave select
int rc;
unsigned int val;
int dir;


static char *device = "default";     //variÃ¡veis da ALSA
snd_output_t *output = NULL;
unsigned long buffer [BUFFER_LEN];


void ctrl_c(int sig){
        close(spi_fd);
        exit(-1);
}

void SPI_Config(void){ //funÃ§Ã£o para configurar o SPI

        int mode;
        mode = 8;
        //ioctl(spi_fd, SPI_IOC_WR_BITS_PER_WORD, &mode_bit);
        //ioctl(spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &mode_speed);
        ioctl(spi_fd, SPI_IOC_RD_MODE, &mode);
```

```
 53          printf("/dev/spi/spidev0.0 aberto!\n");
 54          ioctl(spi_fd, SPI_IOC_RD_BITS_PER_WORD, &mode);
 55          mode = 976000;
 56          ioctl(spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &mode);
 57
 58  }
 59
 60  void SPI_Read_Write(int fd, unsigned long *data, int length){
//funÃ§ao para transferÃªncia SPI
 61
 62          int ret;
 63          struct spi_ioc_transfer spi;
 64          memset(&spi, 0, sizeof(spi));
 65          spi.tx_buf = (unsigned long)data;
 66          spi.rx_buf = (unsigned long)data;
 67          spi.len           = length;
 68          spi.delay_usecs = 0;
 69          spi.speed_hz = 976000;
 70          spi.bits_per_word = 8;
 71          //Transferencia full duplex
 72          ret = ioctl(fd, SPI_IOC_MESSAGE(1), &spi);
 73
 74  }
 75
 76  //fazer um for pra preencher um vetor e um for pra tocar
 77
 78  int main()
 79  {
 80          wiringPiSetup();
 81          pinMode(SL, OUTPUT);
 82
 83          int err;
 84          snd_pcm_t *handle;
 85          snd_pcm_sframes_t frames;
 86
 87          unsigned long receive1=0, receive2=0;
 88          unsigned int c=0, j=0, rc=0;
 89
 90
 91          digitalWrite(SL, HIGH); //inicia SL em 1
 92          //configurando SPI
 93          spi_fd = open("/dev/spidev0.0", O_RDWR);    //abre
arquivo SPI para leitura e escrita
 94          SPI_Config();
 95
 96          // ERROR HANDLING
 97
 98          if ((err = snd_pcm_open(&handle, device,
SND_PCM_STREAM_PLAYBACK, 0)) < 0) {    //abre PCM em modo playback
 99              printf("Playback open error: %s\n",
snd_strerror(err));
100              exit(EXIT_FAILURE);
101          }
102
103          if ((err = snd_pcm_set_params(handle,
//configura dispositivo PCM
104                                        SND_PCM_FORMAT_U16_LE,
//usa formato 16 bits little endian, pois o buffer Ã© de 12 bits
105                                        SND_PCM_ACCESS_RW_INTERLEAVED,
106                                        1,
107                                        8000,
```

```
//usa-se fs de 8kHz, nÃ£o era a calculada, porÃ©m, empiricamente foi a
Ã°nica que funcionou.
108                                     1,
109                                     500000)) < 0) {
110             printf("Playback open error: %s\n",
snd_strerror(err));
111             exit(EXIT_FAILURE);
112
113         }
114
115
116         for(c=0; c < BUFFER_LEN; c++){
117
118                 digitalWrite(SL, HIGH);
119                 usleep(1);
120                 digitalWrite(SL, LOW); // SL=0, inicia a 1Âª
leitura SPI
121                 SPI_Read_Write(spi_fd, &receive1, 1);
122                 digitalWrite(SL, HIGH); // acaba a 1Âª leitura
SPI
123                 usleep(1);
124                 digitalWrite(SL, LOW); // inicia a 2Âª leitura
SPI
125                 SPI_Read_Write(spi_fd, &receive2, 1);
126                 digitalWrite(SL, HIGH); //Finaliza a 2Âª leitura
SPI
127
128                 buffer[c] = ((receive1<<4) + (receive2<<12));
//monta um buffer com os valores das conversÃµes (deslocados em 4
bits),
129
//que posteriormente vai ser enviado ao PCM
130
131                 printf("Loop em %d, valor do buffer:%d\n", c,
buffer[c]);
132                 //printf("receive1:%d, receive2:%d\n", receive1,
receive2);
133         }
134
135         for(j=0; j<1; j++){
//Reproduz 1 vez o que foi gravado no buffer
136             rc = snd_pcm_writei(handle, buffer, BUFFER_LEN);
//funÃ§Ã£o que escreve no PCM no modo interleaved
137         }
138
139         //fechando todos os arqvuidos
140         snd_pcm_drain(handle);
141         snd_pcm_close(handle);
142         //free(buffer); -> se liberar o buffer da erro na
compilaÃ§Ã£o.
143         close(spi_fd);
144
145         return 0;
146
```