**武汉大学计算机学院**

**本科生实验报告**

**蓝牙/NFC音响**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：嵌入式系统原理与应用

指 导 教 师 ：袁志勇

团 队 成 员 一 ：刘淼(2017301500080)

团 队 成 员 二 ：吴正武(2017301500077)

二〇二〇 年 伍 月

**郑 重 声 明**

本团队呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

团队成员签名： 日期：

摘 要

蓝牙/NFC音响在常见的蓝牙音箱的基础上多了NFC功能，NFC有传输距离近的特点，所以NFC连接成功即可默认目的是放音乐，通过协议互相确认让手机将音乐传输过来，就可做到少交互，少操作。NFC的特性还可以和无线充电器配合，让音响集多种功能于一体。本次实验主要实现了NFC和蓝牙连接播放音乐的功能，其他附加特性可通过后期OTA升级获得。

蓝牙方式：蓝牙连接设备后，进入APP软件即可控制设备播放音乐。

NFC方式：将手机放在设备NFC感应区即可自动播放音乐，NFC模式下会自动屏蔽蓝牙模式。

本文均由作者独立完成。

**关键词：**蓝牙，NFC，音乐播放，音响，自动控制

目 录

[1 设计需求分析 1](#_Toc31336)

[1.1 选题目的和意义 1](#_Toc2974)

[1.2 功能 1](#_Toc4684)

[2 硬件组成 2](#_Toc7938)

[2.1 树莓派4b 2](#_Toc26513)

[2.2 音响/喇叭+DAC0832LCN 2](#_Toc3838)

[2.3 蓝牙 2](#_Toc28117)

[2.4 PN532 2](#_Toc2214)

[2.5 电路图 2](#_Toc11808)

[3 音乐传输控制协议 3](#_Toc18459)

[3.1 协议结构 3](#_Toc22115)

[3.2 控制信号 3](#_Toc25747)

[3.3 音乐播放流程 5](#_Toc26072)

[4 音响软件设计 6](#_Toc6586)

[4.1 总体结构图 6](#_Toc25971)

[4.2 数据结构 6](#_Toc24800)

[4.3 数据流 6](#_Toc21697)

[4.4 数据接收 6](#_Toc210)

[4.5 数据处理 7](#_Toc30449)

[5 代码实现 8](#_Toc12630)

[5.1 目录结构 8](#_Toc18638)

[5.2 关键代码 8](#_Toc6045)

[6 心得体会 20](#_Toc28669)

[参考文献 21](#_Toc19239)

1. 设计需求分析
   1. 选题目的和意义

现在市面上蓝牙音箱非常普遍，但少有NFC音响，因为之前NFC的普及还不够广，不是所有手机都有NFC，但现在有NFC功能的手机已经越来越多了，NFC音响也成了一种可能性。而且，因为NFC距离近的特性，可以确保连接音响的目的是确定的——播放音乐，这样可以增加默认播放的操作，减少了打开手机，打开APP，播放音乐的步骤，拿走手机也会自动停止播放，极大的简便了使用外放音响播放音乐的步骤。

* + 1. 蓝牙模式

设计的音响保留了蓝牙连接方式，可以使用户平滑的过度使用习惯，或者作为用户的第二选择，当音响距离用户过远时蓝牙模式也成了第一选择。蓝牙模式的操作方式和市面上的蓝牙音箱差不多。

* + 1. NFC模式

NFC模式下，最大化的简化操作流程，一放就放，一收就停，非常适合用户只是单纯的想听歌的情况，放下手机马上就能听到音乐，还能使用自己设定的歌单。

* 1. 功能

用户先要安装手机APP，APP内自带了音乐播放器，打开音响后，音响会进入等待连接状态，蓝牙开启被扫描，用户可以直接蓝牙连接音响，然后就可以通过控制内载音乐播放器来播放音乐了，支持暂停、开始、快进、后退、切歌等操作。

1. 硬件组成
   1. 树莓派4b

使用树莓派作为音响的控制中心，树莓派已经安装好了linux系统，音响的控制软件只需要写入linux系统中即可。

* 1. 音响/喇叭+DAC0832LCN

树莓派自带有音频输出口，可以使用线控音响连接音频输出口，使用树莓派系统的声卡驱动来输出音频；也可以使用普通喇叭+数模转化器，通过树莓派的GPIO口输出音频。DAC0832LCN是个8位数模转化器，可以将8位的数字信号转化为模拟信号。

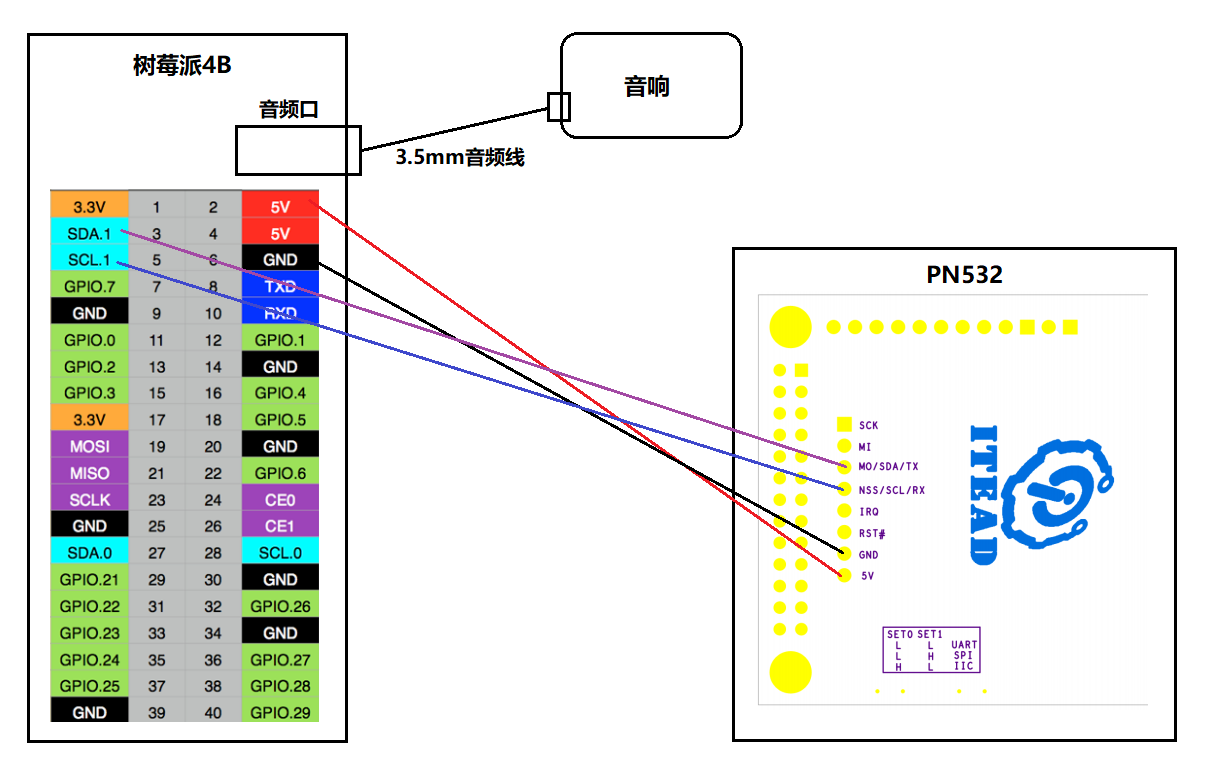
* 1. 蓝牙

使用树莓派自带的蓝牙模块，下载蓝牙驱动即可使用。

* 1. PN532

PN532是一个高度集成的非接触读写芯片，它包含80C51微控制器内核，集成了13.56MHz下的各种主动/被动式非接触通信方法和协议。支持NFC协议，且可以主动传输数据，传输速度也很快。

* 1. 电路图



1. 音乐传输控制协议

本实验通过蓝牙和nfc来传输数据包，要做到音响的效果，至少需要传输两种不同类型的数据包，一是控制信号，用来控制音乐的开始暂停等的操作，二是音乐内容。为了能够区分两种数据包，需要有一个协议。参考了已有的一些协议，比如http协议，ip协议，最后为了简单起见，我选择参考ip协议写一个简单的音乐传输控制协议。

* 1. 协议结构

协议由两部分组成：协议头和正文，协议结构如下：



* + 1. 协议头字段说明

报文类型[0,7]：指出数据部分的类型，类型有控制信号(0x01)和原始音乐二进制数据(0x02)。

音频标识[8,15]：报文类型为0x02时，用来标识同一个音乐数据的。

数据部分长度[16,31]：正文数据部分的长度，单位为字节。

标识[32,47]：和IP协议的标识含义相同，每个报文都有唯一的标识，发送方每发送一个数据包，值加一，如果数据包有分片，所有分片的标识相同。

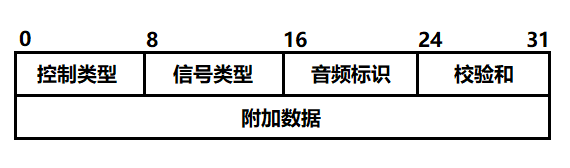
标志[48, 50]：和IP协议的标志含义有所不同，只使用其中的一位[48]，如果这一位为0，表示这是最后一个分片，如果这一位为1，表示后面还有分片。

片偏移[19,31]：和IP协议的标志含义完全不同，这个值表示这个分片在同标识的所有分片中的顺序，例如：音乐数据流分片传输，分片的片偏移从数据流开头，以0,1,2,3的顺序依次递增。

* 1. 控制信号

协议头部的报文类型为0x01时，表示数据部分为控制信号，控制信号数据很短，所以一般不会分片，报文头部标志值为000b，片偏移值无意义。

控制信号数据部分的格式如下：



信号类型：8位，表示控制类型的分组类型，信号类型有音乐传输信号(0x01)和音乐控制信号(0x02)。

音频标识：8位，对应报文头部的音频标识，标识要控制的音频

校验和：8位，使用和IP协议相同的校验方式，先将这8位全设为0，再用每8位分成一单元，所有单元进行二进制反码求和，将结果放入校验和内。

控制类型的取值根据信号类型分组。

信号类型为音乐传输信号(0x01)，时，控制类型取值有：

0x01：开始传输新的音乐数据流。

此时，附加数据内容为如下：

[31,47]：时间信息，16位，即将传的音乐数据流开始的时间，单位为秒。

接下来的字段是用来设置音频的属性的：

[48,55]：声道，8位，只用第一位，0单声道，1双声道

[56,63]：采样位数，8位，只用前两位，[56]位0表示8bit，1表示16bit， [57]位0表示无符号整形，1表示有符号整形。

[64,95]：采样率，32位，单位为Hz。

信号类型为音乐控制信号(0x02)时，控制信号的类型有：

0x01：开始

0x02：暂停

0x03：前进

0x04：后退

0x05：下一首

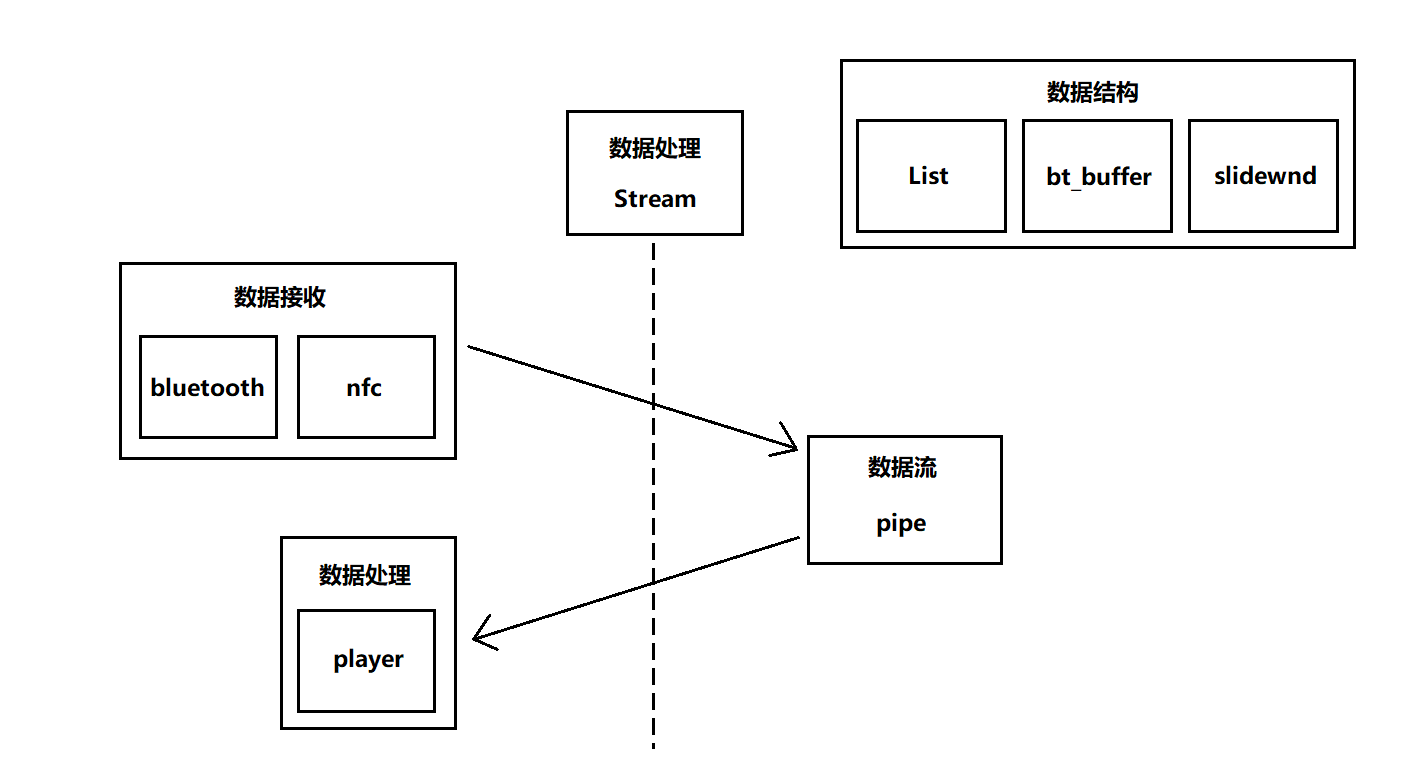
0x06：上一首

此时，附加数据内容如下：

[31,47]：时间信息，每个控制信号都需要携带的参数，表示歌曲需要定 位到的时间轴位置，单位为秒。

* 1. 音乐播放流程

1. App先发送音乐传输信号，表示要传输音乐过来，信号中设置好音乐二进制流的属性，因为二进制流里没有这些属性信息，所以必须要设置，音响拿到属性后设置好准备接收数据。
2. App将解码后的的音乐二进制流分片传输到音响，每个分片大小不能太大，不然后面的控制信号延迟会很高，建议设置为1024个字节。二进制流传输速度要设置得比音乐播放速度稍快，音响拿到数据后可以缓存下来。
3. App发送音乐控制信号，开始播放音乐，音响开始播放音乐。
4. App如果要发送下一首，上一首，前进，后退等信号，需要结束音乐数据流的发送，方式为停止新的分片发送，并设置最后一个分片的标志为0。同时开始传输新的音乐数据流，流的开始是设置的播放位置。
5. 音响软件设计
   1. 总体结构图



* 1. 数据结构
     1. List

存放指针类型数据的单向列表，提供了增删查的基本操作函数。

* + 1. bt\_buffer

一个可以自动变长的内存结构，可以不断添加数据，在分配空间不足的时候会重新分配空间。

* + 1. slidewnd

长度的固定的环形内存结构，有两个指针指向数据头和尾，写到分配内存的最后时会从开头继续写，拿出数据后，头指针也会往后移动。

* 1. 数据流
     1. pipe

数据接收和数据处理之间的数据中转站，使用方式类似socket，由一方创建一个pipe，双方就可以通过pipe id打开这个pipe，然后分别执行读写操作，线程安全。和socket不同的是，这是单向管道，因为第一版程序结构简单，音频控制传输协议也是单方向的，所以只需要数据接收模块写，数据处理模块读。

* 1. 数据接收
     1. bluetooth

通过蓝牙和手机连接获取数据的模块，解析得到的数据，按照协议，如果是数据类型，则使用音频标识作为pipe id往管道内写入数据，如果是控制类型，则写入控制管道内。使用bluez提供的蓝牙库，可以像使用socket一样连接蓝牙，传输数据。

* + 1. nfc

通过NFC和手机连接获取数据的模块，解析得到的数据，按照协议，如果是数据类型，则使用音频标识作为pipe id往管道内写入数据，如果是控制类型，则写入控制管道内。对外表现行为和bluetooth一致，这样数据处理模块就不用理解数据来源问题，只关注于处理数据了。使用libnfc提供的库，这个库支持PN53X系列的芯片。

* 1. 数据处理
     1. player

即音频控制播放模块，控制信号有单独的pipe id，通过pipe不断获取控制信号，然后根据控制信号处理音频。如果有新的数据即将传过来，则创建管道，等待数据接收，有专门的播放线程，不断使用当前播放音乐的数据播放音乐。使用alsa提供的c库，可以自动检测音频设备，播放音频。

1. 代码实现
   1. 目录结构

-include

-nodes

list\_types.h

list.h

memory\_types.h

memory.h

music\_types.h

music.h

slidewnd\_types.h

slidewnd.h

bluetooth.h

log.h

nfc.h

pipe.h

player.h

stream.h

-src

-nodes

list.c

memory.c

music.c

slidewnd.c

bluetooth.c

main.c

nfc.c

pipe.c

player.c

stream.c

makefile

* 1. 关键代码
     1. bluetooth.c

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <bluetooth/bluetooth.h>

#include <bluetooth/rfcomm.h>

#include "log.h"

#include "bluetooth.h"

#include "nodes/music.h"

#include "nodes/slidewnd.h"

#include "stream.h"

#include "pipe.h"

#define LOG\_NAME "bluetooth"

// 所有线程退出标志

static volatile int thread\_end = 0;

void\* bluetoothMainThread() {

    int blue\_socket, client\_socket;

    struct sockaddr\_rc loc\_addr = {0}, rem\_addr = {0};

    char rem\_addr\_str[19];

    int ret;

    int rem\_addr\_len;

    slide\_wnd \*wnd = create\_slide\_wnd(1048576); // 1M

    char buffer[1024];

    int size;

    music\_protocol\* piece = NULL;

    memset(rem\_addr\_str, 0 ,19);

    LOG(LOG\_NAME, "creating socket...");

    blue\_socket = socket(PF\_BLUETOOTH, SOCK\_STREAM, BTPROTO\_RFCOMM);

    if (blue\_socket < 0) {

        ERR(LOG\_NAME, "create socket error");

        return (void \*) -1;

    } else {

        LOG(LOG\_NAME, "success");

    }

    // 本地地址

    loc\_addr.rc\_family=AF\_BLUETOOTH;

    loc\_addr.rc\_bdaddr=\*BDADDR\_ANY;

    loc\_addr.rc\_channel=(uint8\_t)1;

    // 绑定本地地址

    LOG(LOG\_NAME, "binding socket...");

    ret = bind(blue\_socket, (struct sockaddr \*) &loc\_addr, sizeof(loc\_addr));

    if (ret < 0) {

        ERR(LOG\_NAME, "bind socket error");

        close(blue\_socket);

        return (void \*) -1;

    } else {

        LOG(LOG\_NAME, "success");

    }

    // 检查是否有历史配对

    // if (hasHistory) {

    // 有历史记录，主动搜索连接

    // } else {

    // 没有历史记录，等待外部主动连接，listen监听

        LOG(LOG\_NAME, "start listen...");

        ret = listen(blue\_socket, 1);

        if (ret < 0) {

            ERR(LOG\_NAME, "listen error");

            close(blue\_socket);

            return (void \*) -1;

        } else {

            LOG(LOG\_NAME, "listening...");

        }

        while (!thread\_end) {

            rem\_addr\_len = sizeof(rem\_addr);

            client\_socket = accept(blue\_socket, (struct sockaddr \*) &rem\_addr, &rem\_addr\_len);

            if (client\_socket < 0) {

                ERR(LOG\_NAME, "accpet error");

                continue;

            } else {

                LOG(LOG\_NAME, "connect success");

            }

            ba2str(&rem\_addr.rc\_bdaddr, rem\_addr\_str);

            LOG(LOG\_NAME, rem\_addr\_str);

            LOG(LOG\_NAME, "reading message...");

            while(!thread\_end) {

                size = read(client\_socket, buffer, 1024);

                if (size <= 0) {

                    LOG(LOG\_NAME, "out connect");

                    break;

                }

                size = slide\_wnd\_push(wnd, buffer, size);

                // 读出一个协议包

                while(piece = read\_a\_piece(wnd), piece != NULL) {

                    if (MUSIC\_CONTROL == piece->head.type) {

                        // 如果是控制信号

                        ret = write\_control\_pipe(piece, piece->head.len + MUSIC\_HEAD\_LEN);

                    } else if (MUSIC\_DATA == piece->head.type) {

                        // 如果是音乐数据

                        ret = write\_pipe(piece->head.music\_id, piece, piece->head.len + MUSIC\_HEAD\_LEN);

                    }

                    free(piece);

                    piece = NULL;

                }

            }

            close(client\_socket);

        }

    // }

    close(blue\_socket);

    LOG(LOG\_NAME, "bluetooth thread end");

    return (void \*) 0;

}

* + 1. nfc.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <nfc/nfc.h>

#include "log.h"

#include "nfc.h"

#include "nodes/slidewnd.h"

#include "nodes/music.h"

#include "pipe.h"

#include "stream.h"

#define LOG\_NAME "nfc"

#define MAX\_DEVICE\_COUNT 16

// 所有线程退出标志

static volatile int thread\_end = 0;

void\* nfcMainThread() {

    nfc\_context \*context;

    nfc\_target target;

    nfc\_device \*pnd = NULL;

    char buffer[1024];

    int buffer\_size = 1024;

    int ret = 0;

    slide\_wnd \*wnd = create\_slide\_wnd(1048576);

    music\_protocol\* piece = NULL;

    LOG(LOG\_NAME, "initing nfc context...");

    nfc\_init(&context);

    if (context == NULL) {

        ERR(LOG\_NAME, "Unable to init libnfc (malloc)");

        return (void \*) -1;

    }

    // Open using the first available NFC device

    LOG(LOG\_NAME, "open nfc device...");

    pnd = nfc\_open(context, NULL);

    if (pnd == NULL) {

        ERR(LOG\_NAME, "Unable to open NFC device.");

        nfc\_exit(context);

        return (void \*) -1;

    }

    LOG(LOG\_NAME, "NFC device opened.");

    LOG(LOG\_NAME, nfc\_device\_get\_name(pnd));

    LOG(LOG\_NAME, "init nfc device...");

    if (nfc\_initiator\_init(pnd) < 0) {

        ERR(LOG\_NAME, "Unable to init as reader.");

        nfc\_exit(context);

        return (void \*) -1;

    }

    // 设置目标

    nfc\_dep\_mode remote\_dep\_mode = NDM\_UNDEFINED;

    nfc\_baud\_rate remote\_dep\_rate = NBR\_UNDEFINED;

    LOG(LOG\_NAME, "start listennig...");

    while (!thread\_end) {

        if (nfc\_initiator\_poll\_dep\_target(pnd, remote\_dep\_mode, remote\_dep\_rate, NULL, &target, 1000) < 0) {

            sleep(500);

            continue;

        }

        LOG(LOG\_NAME, "recevie a connect");

        LOG(LOG\_NAME, "reading message...");

        while (!thread\_end) {

            ret = nfc\_initiator\_transceive\_bytes(pnd, NULL, 0, buffer, buffer\_size, 0);

            if (ret < 0) {

                LOG(LOG\_NAME, "out connect");

                break;

            }

            ret = slide\_wnd\_push(wnd, buffer, ret);

            // 读出一个协议包

            while(piece = read\_a\_piece(wnd), piece != NULL) {

                if (MUSIC\_CONTROL == piece->head.type) {

                    // 如果是控制信号

                    ret = write\_control\_pipe(piece, piece->head.len + MUSIC\_HEAD\_LEN);

                } else if (MUSIC\_DATA == piece->head.type) {

                    // 如果是音乐数据

                    ret = write\_pipe(piece->head.music\_id, piece, piece->head.len + MUSIC\_HEAD\_LEN);

                }

                free(piece);

                piece = NULL;

            }

        }

    }

    nfc\_close(pnd);

    nfc\_exit(context);

    LOG(LOG\_NAME, "nfc thread end");

    return (void \*) 0;

}

* + 1. player.c

void\* playerMainThread() {

    // 初始化

    pthread\_rwlock\_init(&now\_lock, NULL);

    pthread\_rwlock\_init(&next\_lock, NULL);

    pthread\_create(&playThread, NULL, play\_thread, NULL);

    pthread\_mutex\_init(&control\_cond\_lock, NULL);

    pthread\_cond\_init(&control\_cond, NULL);

    pthread\_create(&controlThread, NULL, control\_thread, NULL);

    music\_signal \*ms = (music\_signal \*) malloc(MS\_MAX\_SIZE);

    int ms\_size = MS\_MAX\_SIZE;

    while (!thread\_end) {

        // 读控制信号

        if (read\_control\_pipe(ms, &ms\_size) != -1) {

            LOG(LOG\_NAME, "recieve a signal");

            if (MUSIC\_SIGNAL\_TRANSPORT == ms->signal\_type) {

                // Assert(ms\_size == sizeof(music\_trans\_data) + 2)

                if (MUSIC\_TRANS\_NEWDATA == ms->control\_type) {

                    new\_music((int) ms->music\_id, (music\_trans\_data \*) ms->data);

                } else {

                    // 错误

                }

            } else if (MUSIC\_SIGNAL\_CONTROL == ms->signal\_type) {

                // Assert(ms\_size == sizeof(music\_control\_data) + 2)

                switch (ms->control\_type)

                {

                case MUSIC\_CONTROL\_START:

                    start\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                case MUSIC\_CONTROL\_PAUSE:

                    pause\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                case MUSIC\_CONTROL\_FORWARD:

                    forward\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                case MUSIC\_CONTROL\_BACKWARD:

                    backword\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                case MUSIC\_CONTROL\_NEXT:

                    next\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                case MUSIC\_CONTROL\_PREV:

                    prev\_music((int) ms->music\_id, (music\_control\_data \*) ms->data);

                    break;

                default:

                    break;

                }

                pthread\_mutex\_lock(&control\_cond\_lock);

                pthread\_cond\_signal(&control\_cond);

                pthread\_mutex\_unlock(&control\_cond\_lock);

            }

        } else {

            // 协议问题

        }

    }

    pthread\_join(playThread, NULL);

    pthread\_join(controlThread, NULL);

    pthread\_cond\_destroy(&control\_cond);

    pthread\_mutex\_destroy(&control\_cond\_lock);

    pthread\_rwlock\_destroy(&next\_lock);

    pthread\_rwlock\_destroy(&now\_lock);

    LOG(LOG\_NAME, "player thread end");

    return (void \*) 0;

}

static void\* play\_thread() {

    int size, ret;

    char\* buffer = NULL;

    // 打开设备

    LOG(LOG\_NAME, "openning pcm...");

    ret = snd\_pcm\_open(&pcm\_handle, "default", SND\_PCM\_STREAM\_PLAYBACK, 0);

    if (ret < 0) {

        // error

        ERR(LOG\_NAME, "snd pcm open fail");

        return (void \*) -1;

    }

    LOG(LOG\_NAME, "success");

    // 设置阻塞

    snd\_pcm\_nonblock(pcm\_handle, 0);

    // 分配params空间

    snd\_pcm\_hw\_params\_alloca(&pcm\_params);

    while(!thread\_end) {

        if (nowMusic == NULL) {

            change\_music();

            pthread\_mutex\_lock(&control\_cond\_lock);

            pthread\_cond\_signal(&control\_cond);

            pthread\_mutex\_unlock(&control\_cond\_lock);

            continue;

        }

        LOG(LOG\_NAME, "setting up parameters...");

        size = set\_params(pcm\_handle, pcm\_params, nowMusic);

        if (size < 0) {

            // error

            ERR(LOG\_NAME, "snd pcm open fail");

            return (void \*) -1;

        }

        LOG(LOG\_NAME, "success");

        buffer = (char \*) malloc(size);

        while(1) {

            ret = read\_pipe(nowMusic->music\_id, buffer, size);

            if (ret < 0) {

                break;

            }

            ret = snd\_pcm\_writei(pcm\_handle, buffer, ret);

            if (ret == -EPIPE) {

                // 饥饿，数据传输过慢，TODO:通知App加快传输速度

            } else if (ret < 0) {

                break;

            }

        }

        free(buffer);

        buffer = NULL;

        snd\_pcm\_drain(pcm\_handle);

        snd\_pcm\_hw\_free(pcm\_handle);

    }

    snd\_pcm\_close(pcm\_handle);

    return (void \*) 0;

}

// 设置参数

static int set\_params(snd\_pcm\_t \*pcm, snd\_pcm\_hw\_params\_t \*params, music\_t \*music) {

    int ret;

    // 初始化pcm参数，设置默认值

    snd\_pcm\_hw\_params\_any(pcm, params);

    // 交错模式

    snd\_pcm\_hw\_params\_set\_access(pcm, params, SND\_PCM\_ACCESS\_RW\_INTERLEAVED);

    // 设置音频格式

    snd\_pcm\_hw\_params\_set\_format(pcm, params, (snd\_pcm\_format\_t) music->format);

    // 设置通道数

    snd\_pcm\_hw\_params\_set\_channels(pcm, params, music->channel);

    // 设置采样率

    snd\_pcm\_hw\_params\_set\_rate\_near(pcm, params, &music->rate, 0);

    // 设置采样周期，系统自动设置

    int frames;

    // 设置可以暂停

    snd\_pcm\_hw\_params\_can\_pause(params);

    // 设置好的参数回写设备

    ret = snd\_pcm\_hw\_params(pcm, params);

        if (ret < 0) {

        // error

        perror("snd pcm param fail");

        return ret;

    }

    // 获取一个周期有多少帧数据，一个周期一个周期方式处理音频数据。

    snd\_pcm\_hw\_params\_get\_period\_size(params, (snd\_pcm\_uframes\_t \*) &frames, 0);

    // buffer\_size = frames \* bit/8 \* channel

    int bytes;

    switch (music->format) {

        case SND\_PCM\_FORMAT\_U8:

        case SND\_PCM\_FORMAT\_S8:

            bytes = 1;

            break;

        case SND\_PCM\_FORMAT\_U16\_LE:

        case SND\_PCM\_FORMAT\_S16\_LE:

            bytes = 2;

            break;

    }

    return bytes \* music->channel \* frames;

}

static void\* control\_thread() {

    while (!thread\_end) {

        pthread\_mutex\_lock(&control\_cond\_lock);

        pthread\_cond\_wait(&control\_cond, &control\_cond\_lock);

        // 快速操作

        pthread\_rwlock\_rdlock(&now\_lock);

        if (nowMusic != NULL) {

            snd\_pcm\_state\_t state = snd\_pcm\_state(pcm\_handle);

            // 获得当前播放的位置

            snd\_pcm\_uframes\_t avail;

            snd\_htimestamp\_t timestamp;

            snd\_pcm\_htimestamp(pcm\_handle, &avail, &timestamp);

            // 计算需要移动的帧数，时间差乘以每秒帧数，每秒帧数=采样率

            long loc = timestamp.tv\_sec - abs(nowMusic->now\_time - nowMusic->start\_time);

            snd\_pcm\_uframes\_t uframes = abs(loc) \* nowMusic->rate;

            if (nowMusic->state == STATE\_PAUSED) {

                snd\_pcm\_pause(pcm\_handle, 1);

            } else if (nowMusic->state == STATE\_RUNNING) {

                snd\_pcm\_pause(pcm\_handle, 0);

            } else {

                goto control\_thread\_end;

            }

            snd\_pcm\_uframes\_t widthable;

            if (loc >= 0) {

                widthable = snd\_pcm\_forwardable(pcm\_handle);

                if (widthable < uframes) {

                    // error

                } else {

                    snd\_pcm\_forward(pcm\_handle, uframes);

                }

            } else if (loc < 0) {

                widthable = snd\_pcm\_rewindable(pcm\_handle);

                if (widthable < uframes) {

                    // error

                } else {

                    snd\_pcm\_rewind(pcm\_handle, uframes);

                }

            }

        }

    control\_thread\_end:

        pthread\_rwlock\_unlock(&now\_lock);

        pthread\_mutex\_unlock(&control\_cond\_lock);

    }

}

* + 1. stream.c

music\_protocol\* read\_a\_piece(slide\_wnd \*wnd) {

    music\_protocol\* result = NULL;

    int result\_len;

    music\_protocol\_head head;

    int piece\_len;

    if (wnd->buffer\_len <= MUSIC\_HEAD\_LEN) {

        return NULL;

    }

    head = music\_fill\_head(wnd->buffer);

    result\_len = head.len + MUSIC\_HEAD\_LEN;

    if (result\_len > wnd->buffer\_len) {

        // 数据不完整，等待

        return NULL;

    }

    switch (head.type) {

    case MUSIC\_CONTROL:  // 控制

        result = (music\_protocol \*) malloc(result\_len);

        slide\_wnd\_pop(wnd, (char \*) result, result\_len);

        if (0 != music\_signal\_check((music\_signal \*) result->data, head.len)) {

            free(result);

            result = NULL;

        }

        break;

    case MUSIC\_DATA:  // 数据

        result = malloc(result\_len);

        slide\_wnd\_pop(wnd, (char \*) result, result\_len);

        break;

    default:

        // error

        break;

    }

    return result;

}

1. 心得体会

通过本次嵌入式课程实验，让我对嵌入式有了更深的理解，对课程内课程外学习的嵌入式知识有了清楚的认知。不过也稍有遗憾，因为本次试验同时考察了多方向上的知识，有软件也有硬件还有软硬件交互的知识。在硬件上，我不是很得心应手，设计一个能够播放的音响不难，但要设计好不是很容易，要考虑到电路安全性问题，电阻电容的使用也是个难题。在这方面我在网上查阅了许多文献，也只能学习他人的设计，从零设计出来对我来说仍有难度。但这正是嵌入式学习的乐趣所在，既动手又动脑，还要不断地思考，相信这次实验对我来说会成为一个宝贵的经验。

参考文献

1. 嵌入式音频编程——alsa库使用, https://blog.csdn.net/wghkemo123/article/details/82152473[2018-08-28]
2. DAC0832中文资料 DAC0832引脚图与应用电路程序,http://www.elecfans.com/dianyuan/436935.html[2016-09-26]
3. 蓝牙学习笔记之HCI协议（一）,https://blog.csdn.net/ylangeia/article/details/87102183[2019-02-12]