嵌入式实验实验报告

学号 : 201805130168

姓名 : 尹永琪

- 嵌入式实验实验报告
  - 实验要求
    - 具体实现的功能
    - 具体分工
  - 实验思路与逻辑
    - 原理分析
    - 拆包 , JAVA反编译
    - 串口连接与Adb连接
    - 交叉编译
  - 硬件控制逻辑与关键代码
    - http接口设置
    - 通用include文件
    - 多线程
    - beep与Led
    - NFC
    - 步进电机Servo
    - 点阵Badapple
    - 数码管
  - 实验结果
    - 注册
    - 登录
    - 主界面
    - LED测试
    - 步进电机
    - 点阵
    - NFC
    - 数码管
    - 二维码
    - 总览
  - 实现代码

实验要求

> 开发一款基于Android平台的软件，尽可能的利用开发板资源实现尽可能丰富的功能。
>
> 基础要求：
>
> 1. 软件具有用户密码登录界面，软件用户注册界面。
> 2. 实现基于NFC识别，控制 LED灯点亮，蜂鸣器发出提示音。
> 3. 通过识别扫描二维码对应的ID，控制 LED灯点亮，蜂鸣器发出提示音。
> 4. 刷卡和扫描二维码的记录存入本地数据库中。
> 5. App界面中至少包括“扫描”按钮，显示最近五次成功刷卡或二维码识别的系统时间。
>
> 拓展（举例）：
>
> 1. 点阵显示ID号

> 2. 控制舵机动作

具体实现的功能

1. 登录注册

   注册输入用户名和两次密码 , 登录输入用户名和密码

2. LED控制

   对四个LED均可以单独的控制器亮和灭

3. 蜂鸣器

   控制蜂鸣器响一段时间

4. 步进电机

   控制步进电机左转右转暂停

5. 点阵控制

   实现16*16的点阵播放badapple (视频的字符画 , 总时长3分20s , 每126ms一帧)

6. NFC识别卡片UID

   通过刷卡识别卡片UID , 并将UID MD5后的数值给数码管

7. 二维码扫描

   扫描并识别二维码信息后 , 将其信息MD5后的数值给数码管

8. 数码管显示

   显示一个八位数

9. 数据库

   数据库存储用户名和密码 , 刷卡和二维码的记录 (含类型 , 时间 , 内容)

10. 完全自主开发的Server

    在板子上运行的自己通过Cpp编写的Server程序

所有操作在完成时都会调用蜂鸣器响500ms

具体分工

王新宇 : 用java实现安卓的客户端程序 , 查找一些cpp的开源框架

尹永琪 : 用cpp实现控制硬件的后端代码 , 安卓客户端数据库实现

实验思路与逻辑

原理分析

对于硬件来说 (例如 , 步进电机 , 点阵 , NFC , 数码管) 需要通过写内存通过cpld来控制 , 而openmem需要root权限才能执行 .

因此 , 如果在java内写cpp来控制硬件是不现实的 , 因为安卓程序运行时是没有root权限的 .

所以有两条方案 :

方案一 : root本机 , 让程序在root权限下运行 .

结果 : 因为没有代码 , 所以没有root , 且不知道本身有没有改过安卓代码

方案二 : 编写驱动 , 像LED一样.

结果 : 驱动可以写 , 但是需要专用的交叉编译器结合本身代码才能写进去 , 所以也不可行

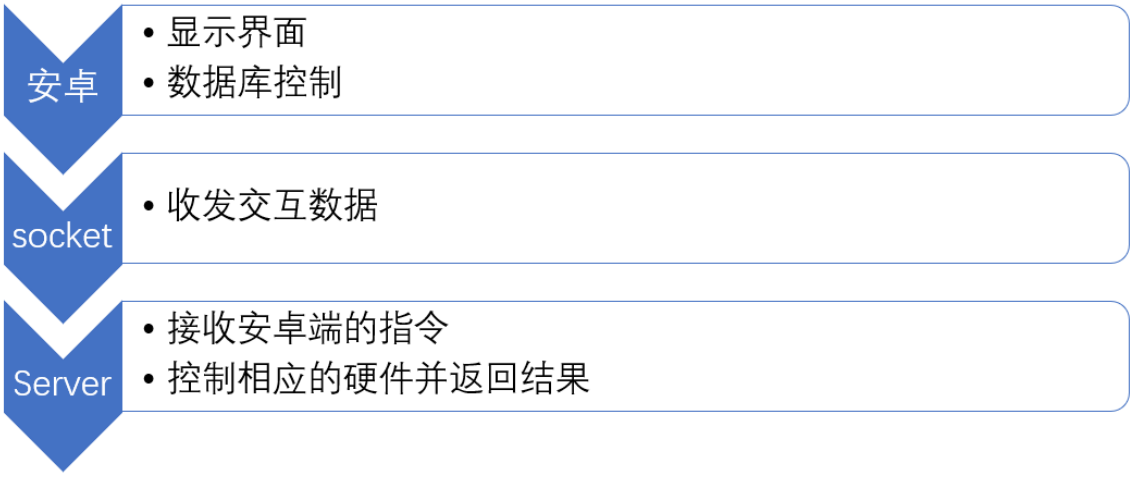
拆包，JAVA反编译

由于板子上存在一个正常运行的，可以控制所有硬件的程序，所以我们通过串口连进去后，找到了该程序的apk包`IMX6Demo.apk`.

我们使用 dex2jar，jd-gui，apktool等工具对该包进行了处理，最后反编译得到了一份java的代码．通过详细阅读该份代码，我们惊奇的发现其内部没有任何关于硬件的逻辑，反而有一个socket类．该程序将所有有关硬件的逻辑都通过socket传输给一个名为Server的可执行程序，该程序在root权限下执行. 所以其可以控制硬件，而安卓的程序不需要任何控制硬件的代码，只需要给Server发包即可

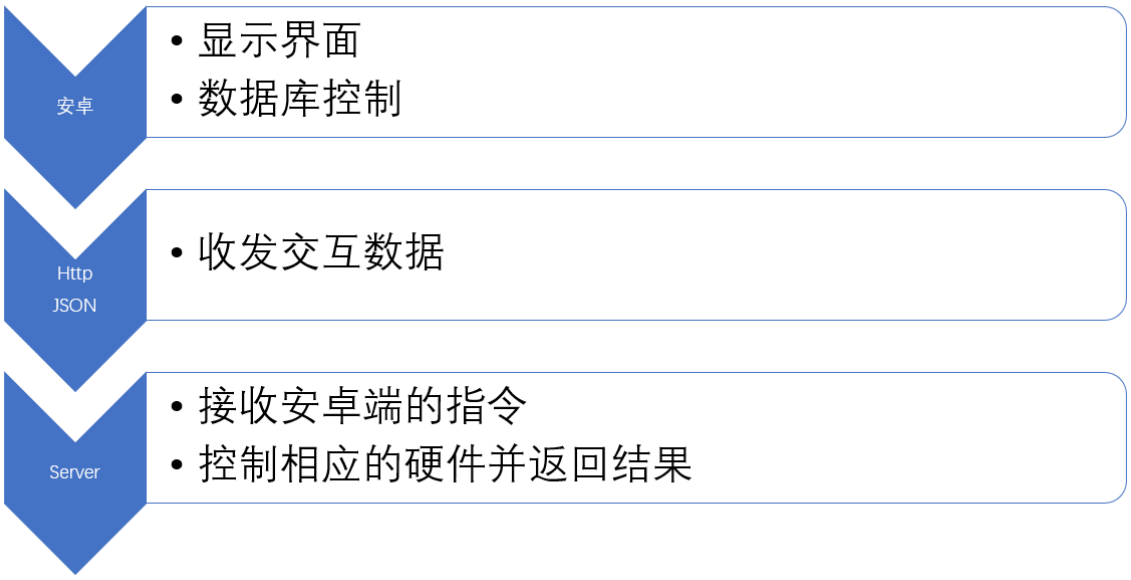
类似于常见网页中的前后端模式，安卓为前端，socket为http，Server为后端程序.

下图为实例程序整体的流程框架

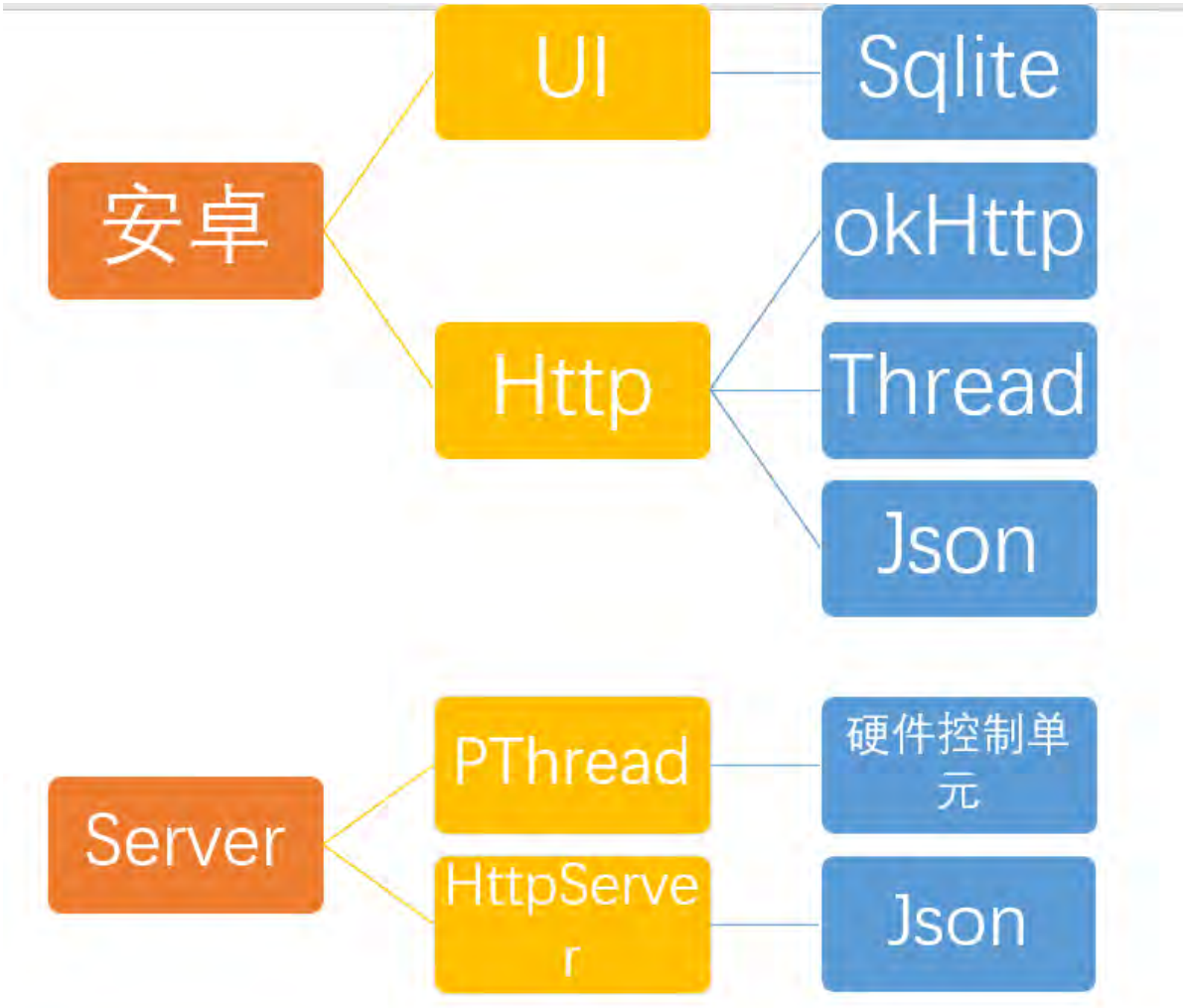


我们最终采用了这种方式进行编程，不过我们将socket换成了http+JSON，Server由Cpp重写，安卓端由Java重写，完全重构可以达到更高的自由度.

下图为我们程序整体的流程框架



最终两端代码的层次框架



串口连接与Adb连接

为了方便进行调试，我们连接板子上的com串口，通过软件Putty来连接，进入后有root权限，可以方便的查找文件，运行程序.

而串口在传输文件以及控制安卓上不如Adb，所以我们又连接otg口进行adb链接，方便传输文件和管理安卓

交叉编译

由于编写cpp代码需要交叉编译，也就是需要在本机编译好以后通过交叉编译器编译出板子对应版本的可执行文件 ，通过adb push传输到板子上，通过串口连接后，chmod 777修改文件权限后可以执行.

最终我们找到的交叉编译器为`gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabi`

4.9为一个比较久的版本，以为板子上的linux内核较为古老，高版本的交叉编译器编译的程序其无法执行. 同时太低的版本没法支持一些库和cpp11的特性，最后选定了4.9，同时板子不支持硬件浮点数运算，所以选择gnueabi版本进行软件模拟浮点运算.

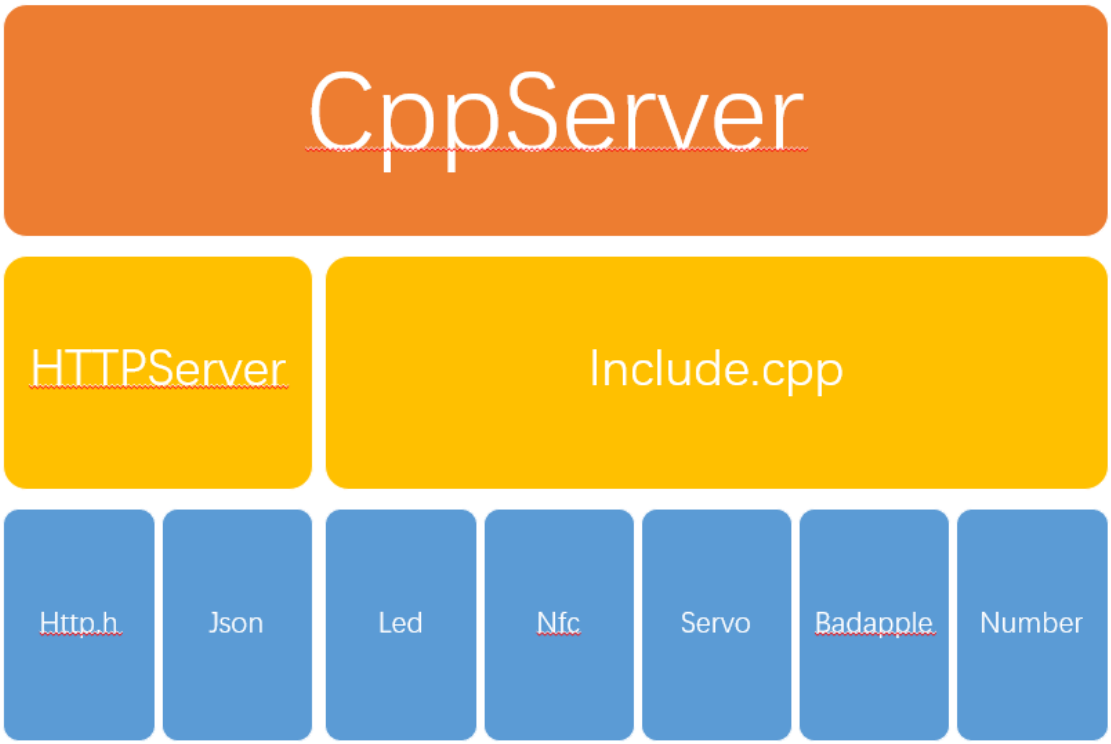
一个可执行的编译方案为`~/gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabi/bin/arm-linux-gnueabi-g++ main.cpp -o test -static -std=c++11 -w -pthread`

注：需要通过static进行静态链接，否则一些动态链接库在板子上找不到.

硬件控制逻辑与关键代码

由于我主要负责Server端，所以主要写Server端的代码 .

代码框架 :



http接口设置

http采用的是开源框架http.h

json采用的是开源框架jsonxx

在代码中按照以下方式写

```
server = http_server_init(8080, handle_request); // 创建httpserver , 绑定
hander

http_server_listen(server); // 监听
```

hander为监听http信息，并根据request执行不同函数

```
void handle_request(http_request_s *request)
{
    http_request_connection(request, HTTP_AUTOMATIC);
    http_response_s *response = http_response_init();
    http_response_status(response, 200);
    http_string_s body = http_request_body(request);
    puts("some http");
    if (request_target_is(request , "/servo/stop")) {
    } else if (request_target_is(request , "/servo/clock")) {
    } else if (request_target_is(request , "/servo/countclock")) {
    } else if (request_target_is(request, "/beep")) {
    } else if (request_target_is(request, "/led")) {
    } else if (request_target_is(request, "/nfc/label")) {
```

```
    } else if (request_target_is(request, "/matrix")) {
    } else if (request_target_is(request, "/tube"))  {
    } else {
        http_response_status(response, 404);
    }
    http_respond(request, response);
}

```

具体内容根据以下接口文档进行编写

> 全部用post传，参数的json----raw传输
>
> 返回的状态码都是200，状态通过http的Body确定

修改LED

```
POST /led
```

请求参数

```
{
   "led" : 0~3 , //表示控制第几个LED
   "op" : "open" / "close" // open是开 close 是关
}
```

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

NFC

NFC Label

> 参考样例中的label函数，读取nfc值并将ID回传

```
POST /nfc/label
```

请求参数

无

返回参数

以json形式返回

```
{
    "state":"OK"/"FAILED",//成功、失败
    "ID":"xxxxxx" // string 如果state为FAILED，可以为空
}
```

示例数据

```
{
    "state":"OK",
    "ID":"12-32-54-12";
}
or
{
   "state":"FAILED" ,
   "ID":""
}
```

蜂鸣器

蜂鸣器响

> 响 500ms （按一个键就会响一次

```
POST /beep
```

请求参数

无

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

点阵

使点阵播放badapple

> 无状态的。

```
POST /matrix
```

请求参数

无

请求参数

无

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

步进电机

左转

```
POST /servo/left
```

请求参数

无

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

右转

```
POST /servo/right
```

请求参数

无

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

停止

```
POST /servo/stop
```

请求参数

无

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

数码管

显示数

```
POST /tube
```

请求参数

json格式

```
{
    "number":1234;// 取这个数的后8位
}
```

返回参数

以raw形式返回（String）

“OK” 操作成功

“FAILED” 操作失败

示例数据

```
“OK”
```

通用include文件

include.cpp为所有代码都会引用的一个通用文件，里面包含了所有需要引用的库.

同时定义了一些常见的函数方便调用

http相关

```
// 对比request
int request_target_is(struct http_request_s* request, char const * target) {
    http_string_t url = http_request_target(request);
    int len = strlen(target);
    return (len == url.len) && ( memcmp(url.buf, target, url.len) == 0 );
}
// 返回ok
void set_response_ok(http_response_s* response){
    http_response_header(response,"Content-Type", "text/plain");
    http_response_body(response,"OK",2);
}
// 返回failed
void set_response_fail(http_response_s* response){
    http_response_header(response,"Content-Type", "text/plain");
    http_response_body(response,"FAILED",6);
}

```

绑定Ctrl+C时的退出

```
void handle_sigint(int signum){
    matrixRet = 1;
    tubeRet = 1;
    sleep(2);
    puts("我跑卤辣");
    free(server);
    exit(0);
}
```

由于在退出是需要释放资源，所以在这里将matrixRet，tubeRet设置为1用来控制点阵和数码管的结束(因为这两个需要单独开线程写死循环进行控制) sleep以供这两个线程有足够的时间处理结束 .

数码管和点阵在运行时每0.1s就会检查一遍这个变量的值，如果为1则可以退出 .

这个方案还可以用在唯一性上，即这个变量相当于一个锁，保证数码管和点阵永远只有一个线程在控制，每次调用时将其设置为1然后sleep一段时间后再执行

```
tubeRet = 1;
usleep(100000);
tubeRet = 0;

创建线程 执行
```

```
while (1)
{
    if (tubeRet == 1) {
        puts("程序结束");
        break;
    }
    // do something
}

```

多线程

由于程序中存在一些sleep (例如beep会sleep) 以及死循环 (数码管和点阵) . 如果在主进程阻塞则会影响http的收发 , 导致程序卡死 , 所以需要多线程 . (也可以用fork)

多线程通过c语言的pthread库 (c11的Thread由于缺少相关的库 , 无法执行)

```
pthread_t th; // 创建线程
int ret;
int *thread_ret = NULL;
ret = pthread_create(&th, NULL, badapple/*要执行的函数*/, NULL);
```

beep与Led

由于存在ledtest这个驱动 , 其可以控制led , 所以可以直接通过系统调用ioctl来进行控制led和beep

```
int fd;
fd = open("/dev/ledtest", O_RDWR | O_SYNC); /* 打开设备文件 */
ioctl(fd, status, num); /* 系统调用，设置 LED 的状态 */
close(fd); /* 关闭设备文件 */
```

status为led状态 (0灭1亮 , 蜂鸣器反过来)

num 0~3为led , 4为蜂鸣器

led执行函数为

```
jsonxx::json j = jsonxx::json::parse(get_body_string(body));
int status = std::stoi(std::string(j[std::string("op")]));
int num = std::stoi(std::string(j[std::string("led")]));
led(status, num);
set_response_ok(response);
```

beep函数为

```
pthread_t th;
int ret;
int *thread_ret = NULL;
ret = pthread_create(&th, NULL, beep, NULL);
set_response_ok(response);

void *beep(void *arg) {
    led(0, 4);
    usleep(500000);
    led(1, 4);
    return arg;
}
```

NFC

NFC为参考实例程序封装好的NFC类，主要需要修改的函数为LableNFC ()函数，即读取

下面的代码只有类的框架和LableNFC的代码，详细代码可以去文件中阅读

```
class NFC{
public:
    NFC();
    ~NFC();
    int initNFC();
    int set_port_option(int fd,int nSpeed, int nBits, char nEvent, int
nStop);
    void WakeUpNFC();
    std::string LabelNFC();
    int  handler(int );

private:
    unsigned char buf[100];
    int fd;
    int writebyte,readbyte;
    int i;
    const unsigned char WakeUp[24];
    const unsigned char Label[11];
};

std::string NFC::LabelNFC(){
    writebyte =write(fd, Label, sizeof(Label)); // 写内存 准备读数据
    sleep(1);
    readbyte=read(fd,buf,30); // 读数据
    char temp[100];
    if(readbyte) // 如果读到
    {
        for(i=0;i<readbyte;i++) printf("%x-",buf[i]);
        printf("\n");
        for(i=0;i<readbyte;i++)
        {
            if(buf[i]==0x8&&buf[i+1]==0x04)
            {
                    printf("UID: ");
                    printf("%x-",buf[i+2]);
                    printf("%x-",buf[i+3]);
                    printf("%x-",buf[i+4]);
                    printf("%x",buf[i+5]);
```

```
                    int now = 0;
                    sprintf(temp,"UID: ");
                    // 把读到的数据写到temp
                    sprintf(temp+5,"%2x-",buf[i+2]);
                    sprintf(temp+8,"%2x-",buf[i+3]);
                    sprintf(temp+11,"%2x-",buf[i+4]);
                    sprintf(temp+14,"%2x",buf[i+5]);
                }
            }
            readbyte=0;
            memset(buf,0,30);
        } else {
            // 如果没读到
            return std::string("");
        }
        printf("\n");
        return std::string(temp);
}

```

步进电机Servo

```
unsigned char data;
int mem_fd;
unsigned char *cpld;
mem_fd = open("/dev/mem", O_RDWR);
cpld = (unsigned char*)mmap(NULL,(size_t)0x04,PROT_READ | PROT_WRITE | PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
if(cpld == MAP_FAILED) return;
*(cpld+(0xe2<<1)) = status;   //步进电机地址0xe2<<1
munmap(cpld,0x04);
close(mem_fd);

```

步进电机为的逻辑为，openmem后，在0xe2<<1处写相关的内容以控制步进电机的状态

```
0:停止
1:停止
2:逆时针
3:顺时针
```

点阵Badapple

点阵的逻辑为：

在(0xc0 + x(0~15) 的位置写一个数字 y，x表示你要控制的是第几行，y表示这一行的16个点如何亮和灭。

每隔一段时间，读入当前这一帧的所有内容，即每一位亮不亮，通过二进制修改其0~15位，然后写到相应的内存位置

```
// 打开内容
int mem_fd = open("/dev/mem", O_RDWR);
unsigned short *cpld = (unsigned short *)mmap(NULL, (size_t)0x20, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_SHARED, mem_fd, (off_t)(0x8000000));
if (cpld == MAP_FAILED) return arg;

```

```
// 设置文件流
std::ifstream in("badapple2.txt");
std::string str;
// 每次读一帧
while (std::getline(in, str, 'x')) {
    while (str.front() != '0' && str.front() != '1' )
str.erase(str.begin());
    int look = 0; // 当前要写的是哪一行
    int line = 0; // 这一行的如何显示 (16位二进制 , 0表示不亮1表示亮)
    for (auto & i : str) {
        if (i == '1') {
            look += 1;
            look <<= 1;
        } else if (i == '0') {
            look <<= 1;
        } else {
            look >>= 1;
            *(cpld + ((0xc0 + line) << 1)) = look;
            look = 0;
            line += 1;
        }
    }
    // 每0.126s切换一帧可以比较自然
    usleep(126000);

}
in.close();
munmap(cpld, 0x20);
close(mem_fd);

```

数码管

数码管的逻辑为 :

在 0xe6<<1 处写数码管亮哪一个 , 在 0xe4<<1 处写数码管内容

由于一次只能亮一个 , 所以每0.001s会切换一个数码管亮以欺骗眼睛.

其中八个数码管对应的值(即要在0xe6<<1处写的值)为 `0x80 0x40 0x20 0x10 0x08 0x04 0x02 0x01`

0~9的数码管显示内容(即0xe4<<1写的值)为
`0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x7f,0xff`

实现代码

```
// 80 40 20 10 08 04 02 01
// 处理需要显示的数值
int num = *(int*)arg;
if (num < 0) num = -num;
// 打开内容 , 设置八个数码管的地址以及要显示内容的16进制数
int mem_fd = open("/dev/mem", O_RDWR);
unsigned char* cpld = (unsigned char*)mmap(NULL,(size_t)0x10,PROT_READ |
PROT_WRITE | PROT_EXEC,MAP_SHARED,mem_fd,(off_t)(0x8000000));
unsigned char tube[] =
{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x7f,0xff};
unsigned char addr[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

```c
while (1)
{
    // 判断结束
    if (tubeRet == 1) {
        puts("程序结束");
        break;
    }
    // 不断取最后一位然后再相应的位置写上对应的值
    int temp = num;
    for (int i = 0; i < 8; i++) {
        *(cpld+(0xe6<<1)) = addr[i];   //数码管地址
        *(cpld+(0xe4<<1)) = tube[temp % 10];   //数码管千位
        temp /= 10;
        usleep(1000);
    }
}
munmap(cpld,0x10);
close(mem_fd);
```

实验结果

注册

注册界面，输入用户名和两遍密码

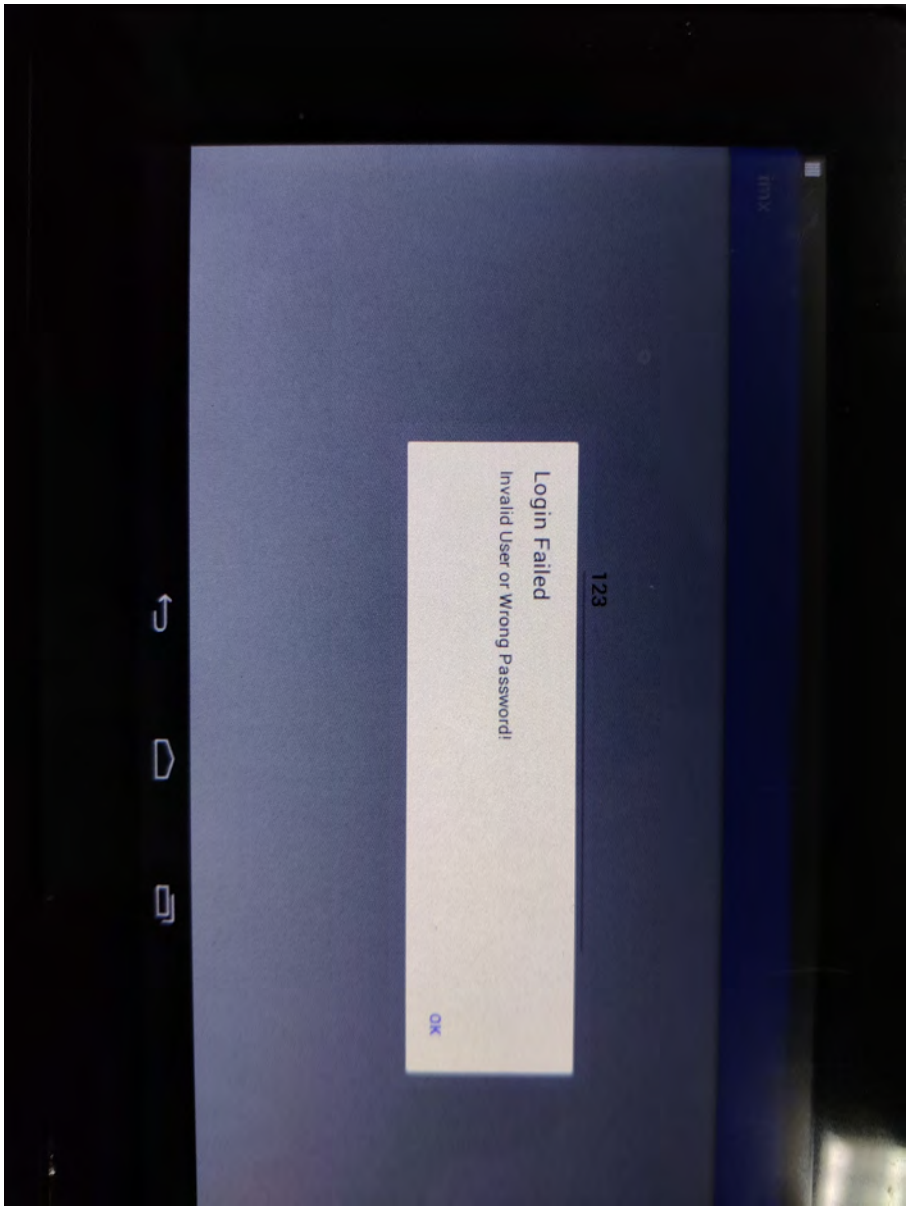
两遍密码出错

注册成功

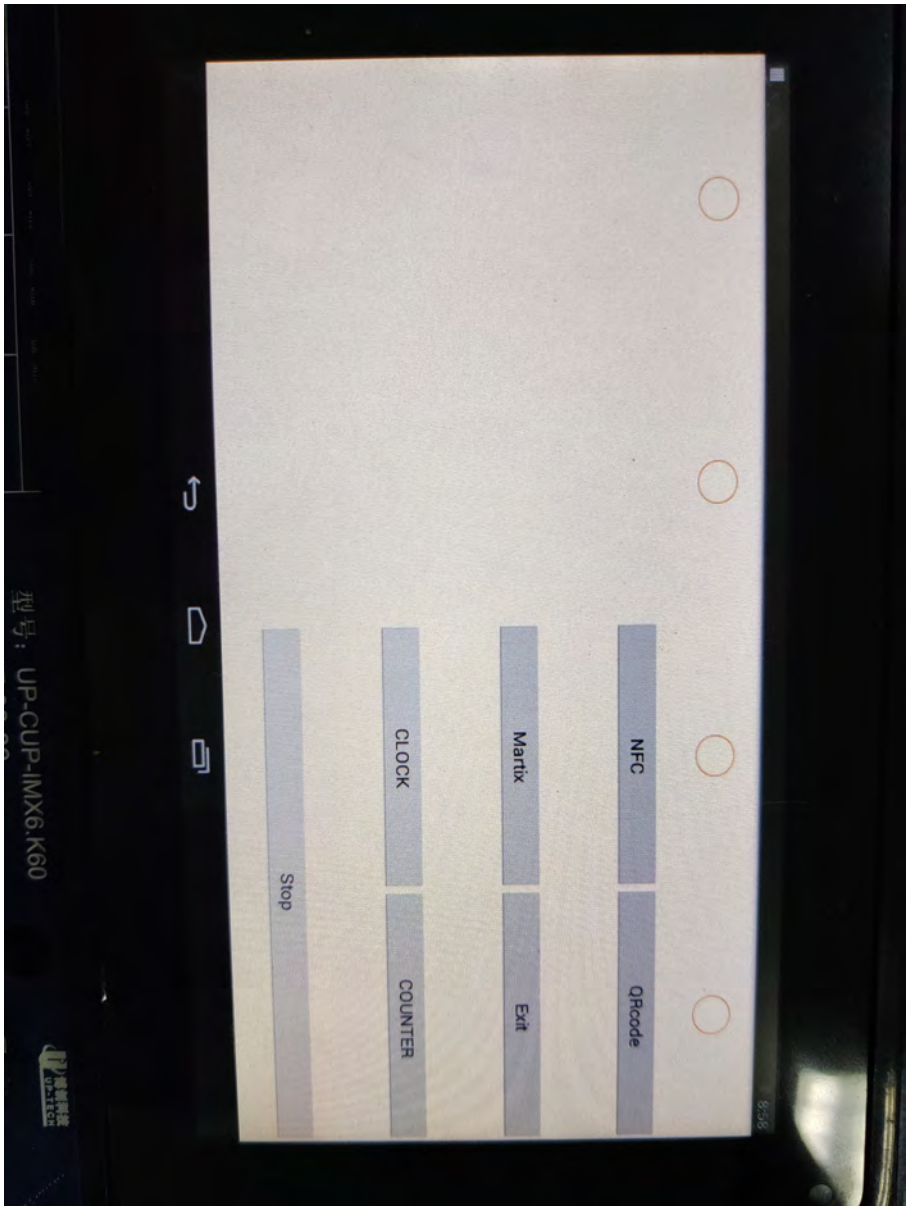
登录

输入用户名密码即可

密码错误 :



主界面



上方四个按钮是用来控制LED的四个按钮

左边的空白区域存储刷卡和扫码记录

NFC按钮为控制nfc硬件读卡

QRcode为调用摄像头扫描识别二维码

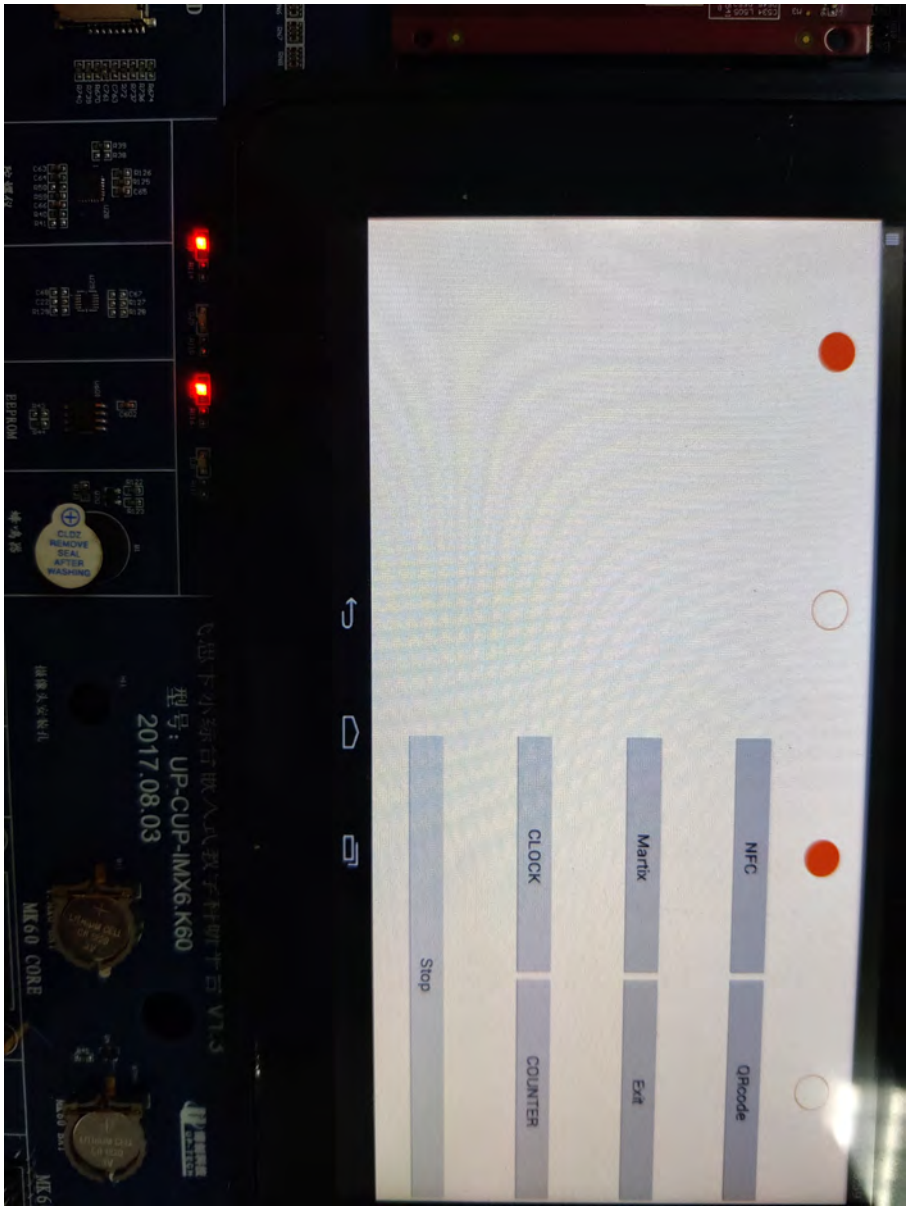
Matrix为调用点阵播放badapple

Clock为顺时针调用步进电机，Counter为逆时针调用步进电机，Stop为停止步进电机

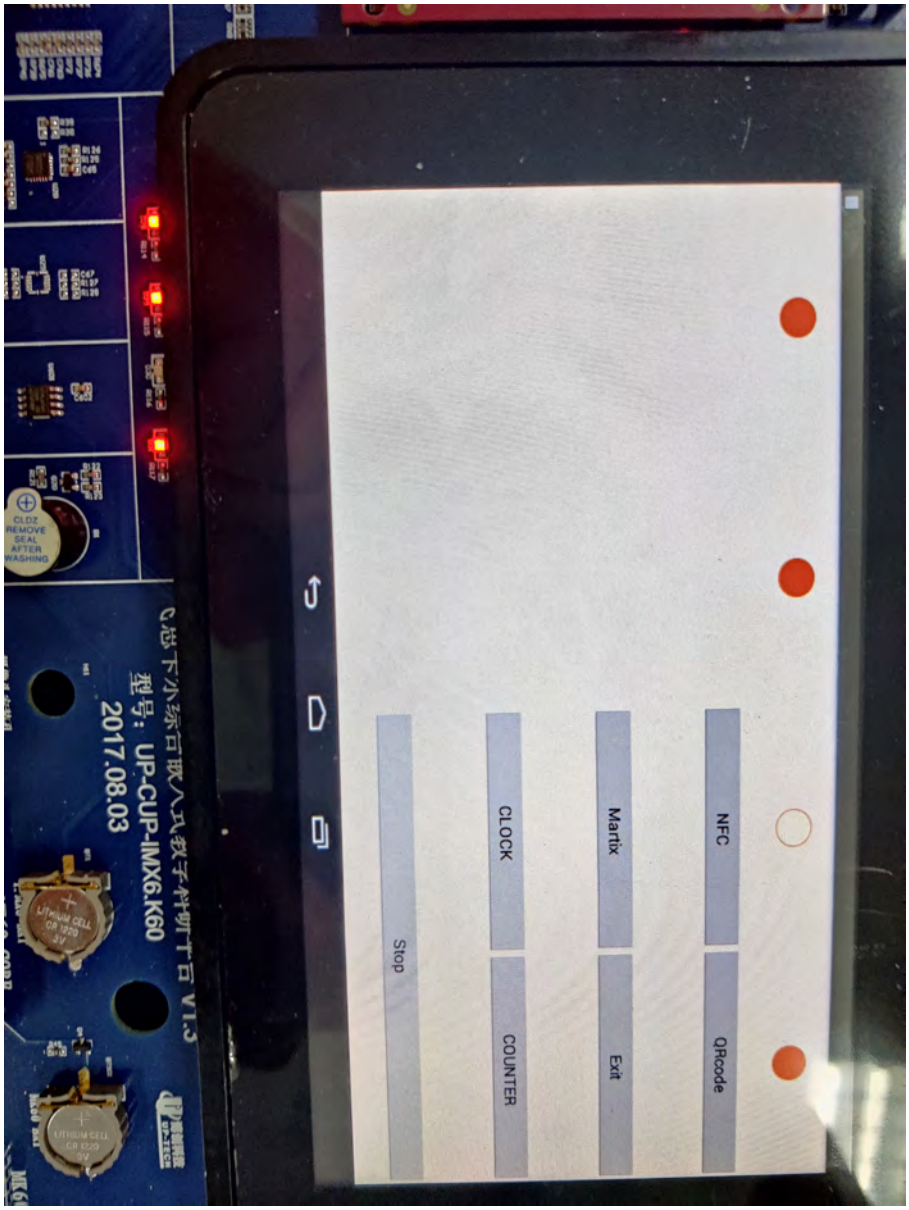
Exit为退出

LED测试

LED1和3亮

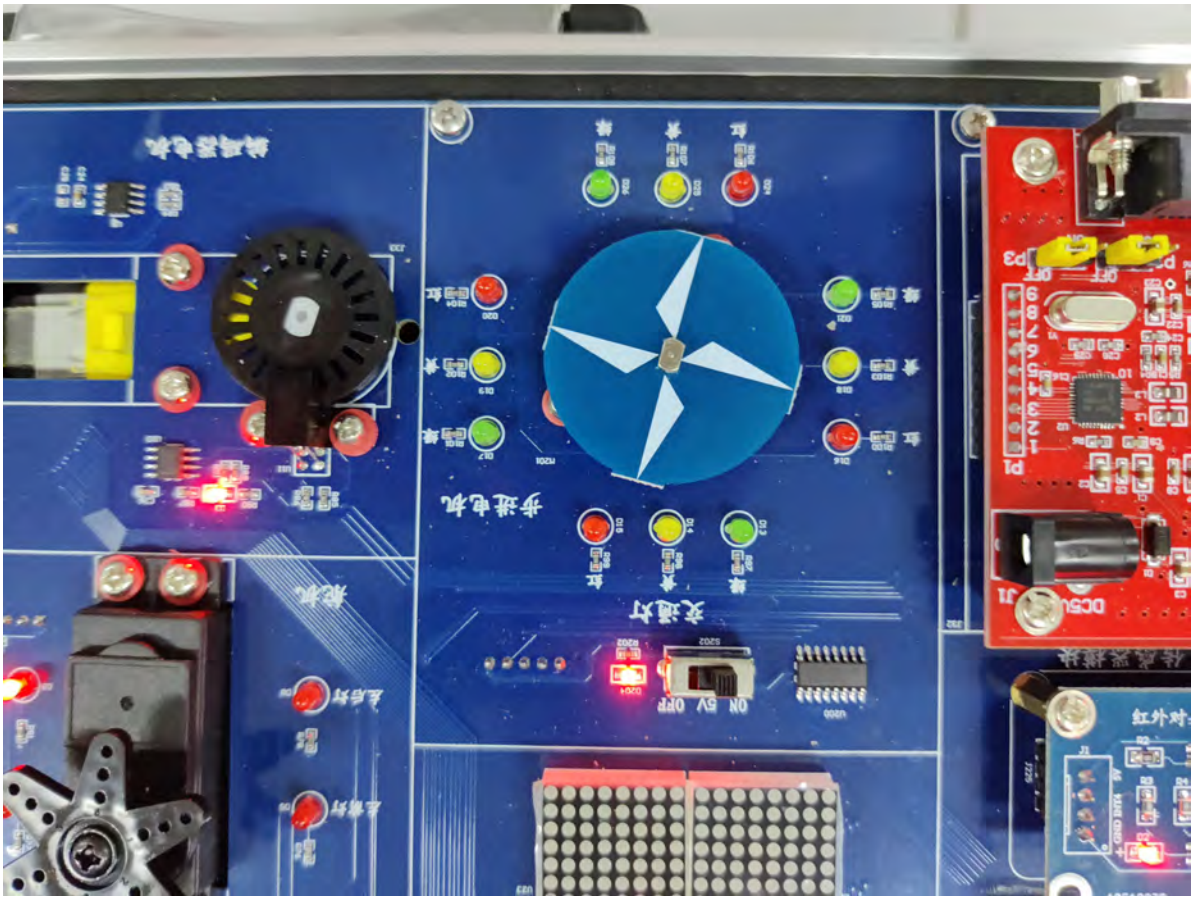

NFC
QRcode
Martix
Exit
CLOCK
COUNTER
Stop
型号：UP-CUP-IMX6.K60
2017.08.03
ME60 CORE

LED124亮



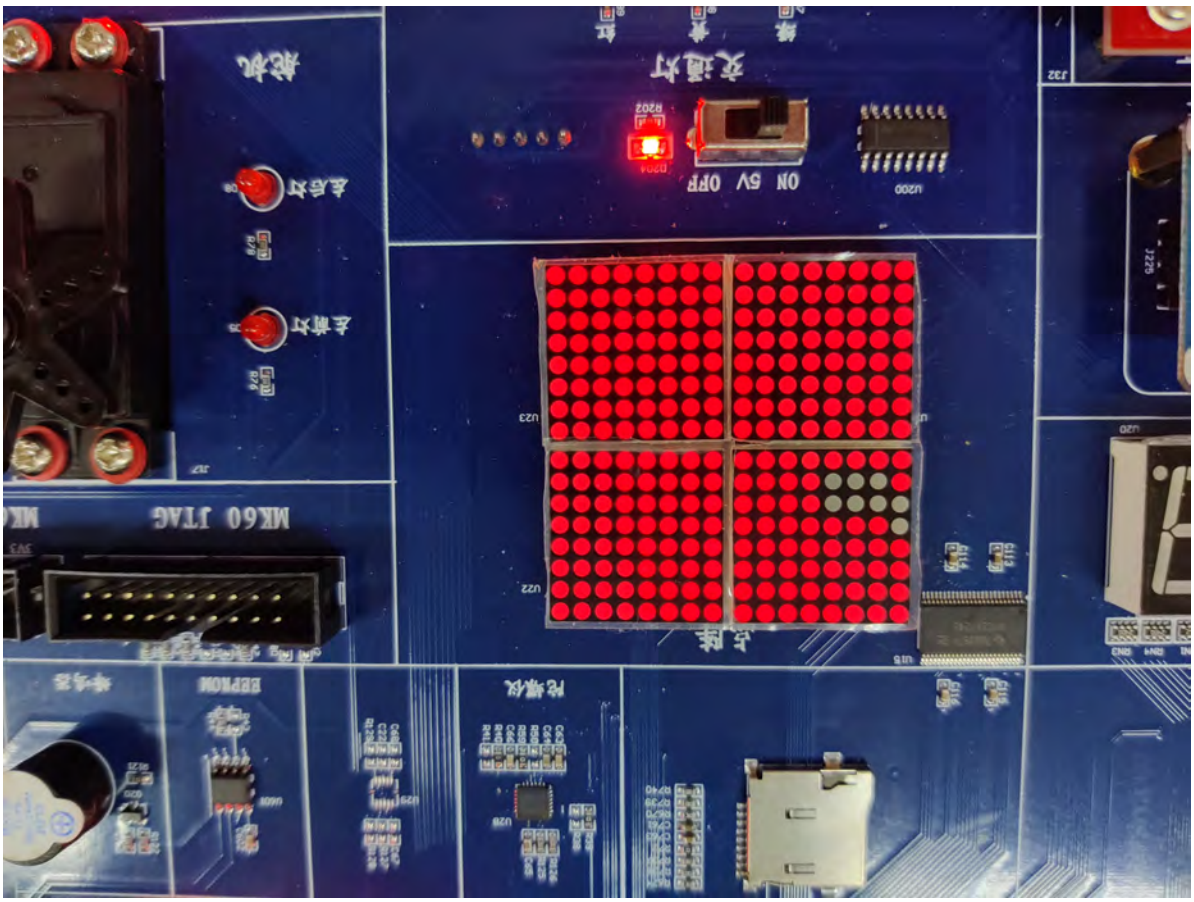
步进电机

(静态看不出来转没转)



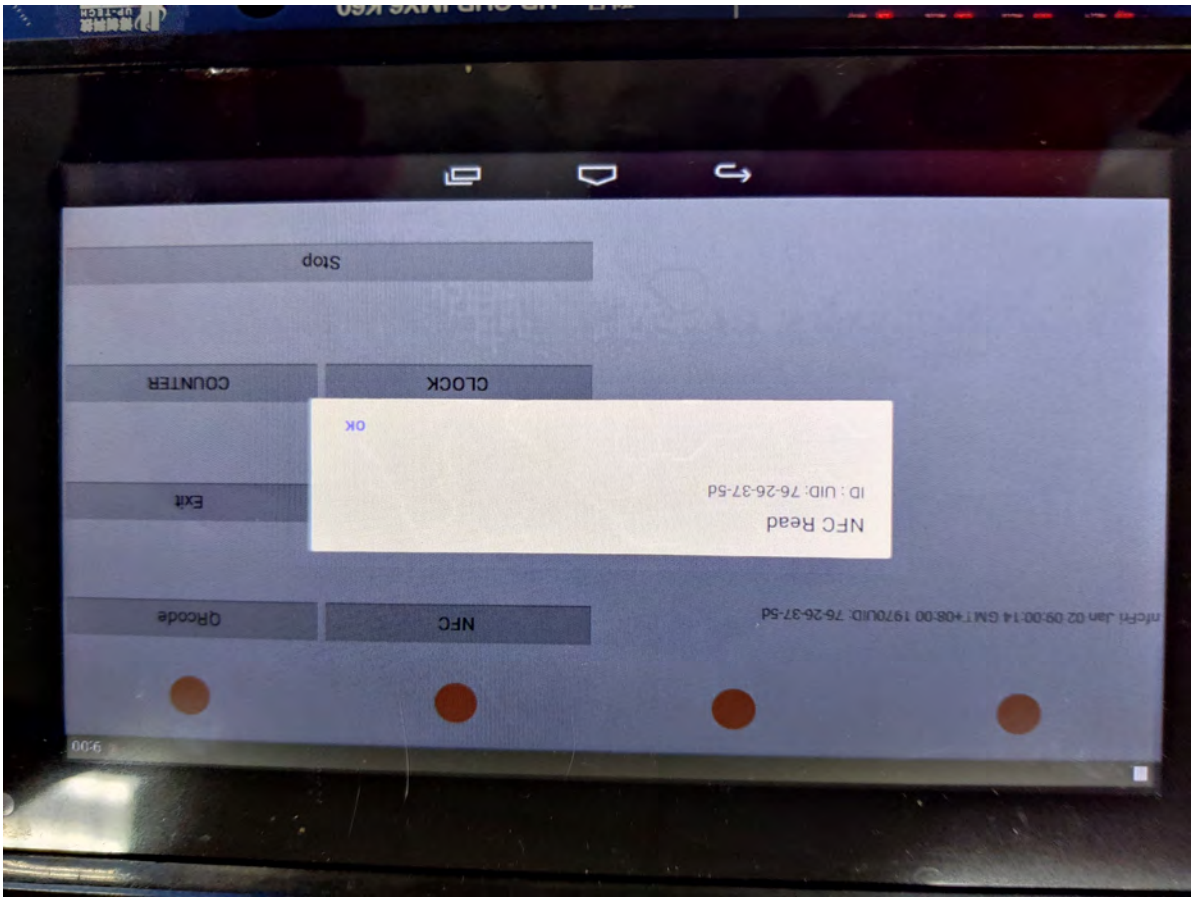
点阵

播放badapple的某一帧，具体视频可以查看 https://www.bilibili.com/video/BV11U4y157sg/

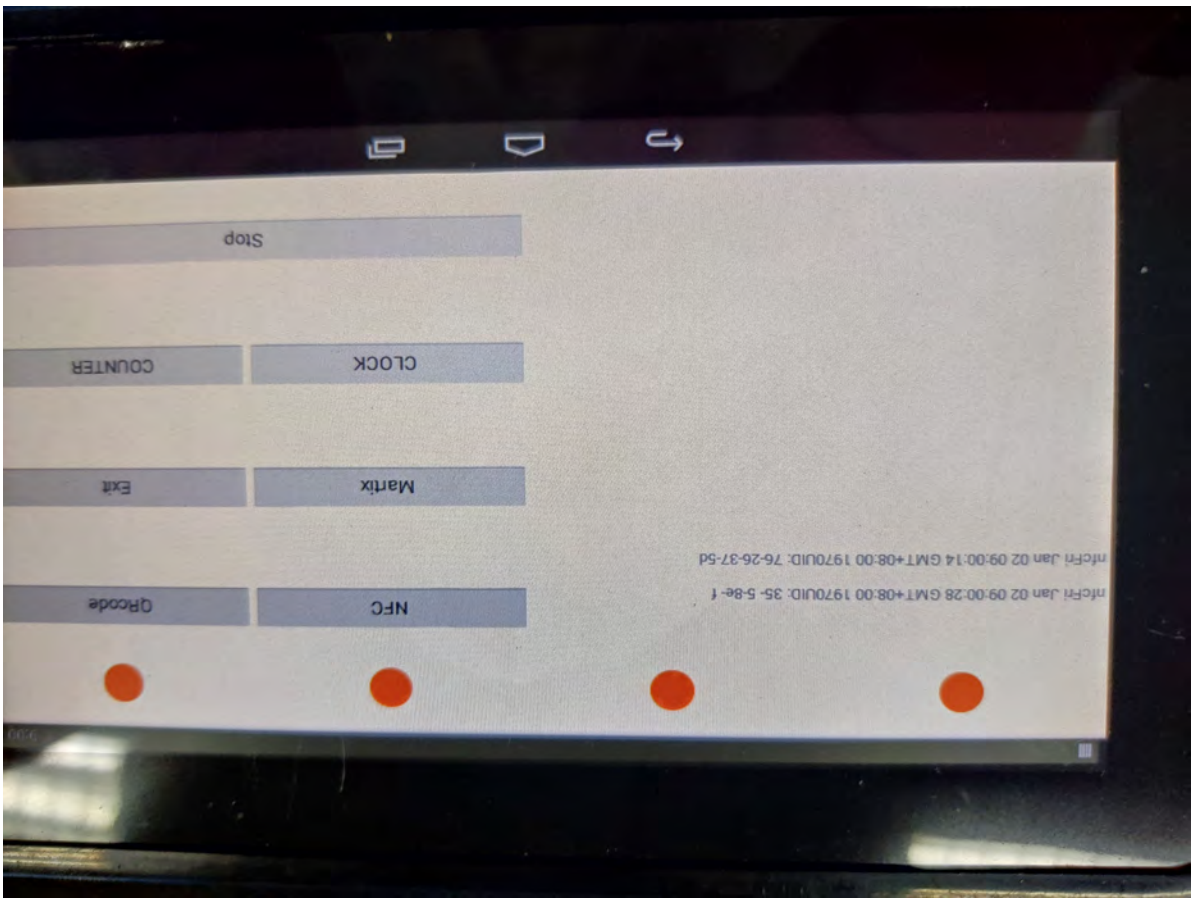


NFC

读取一个nfc卡片的UID

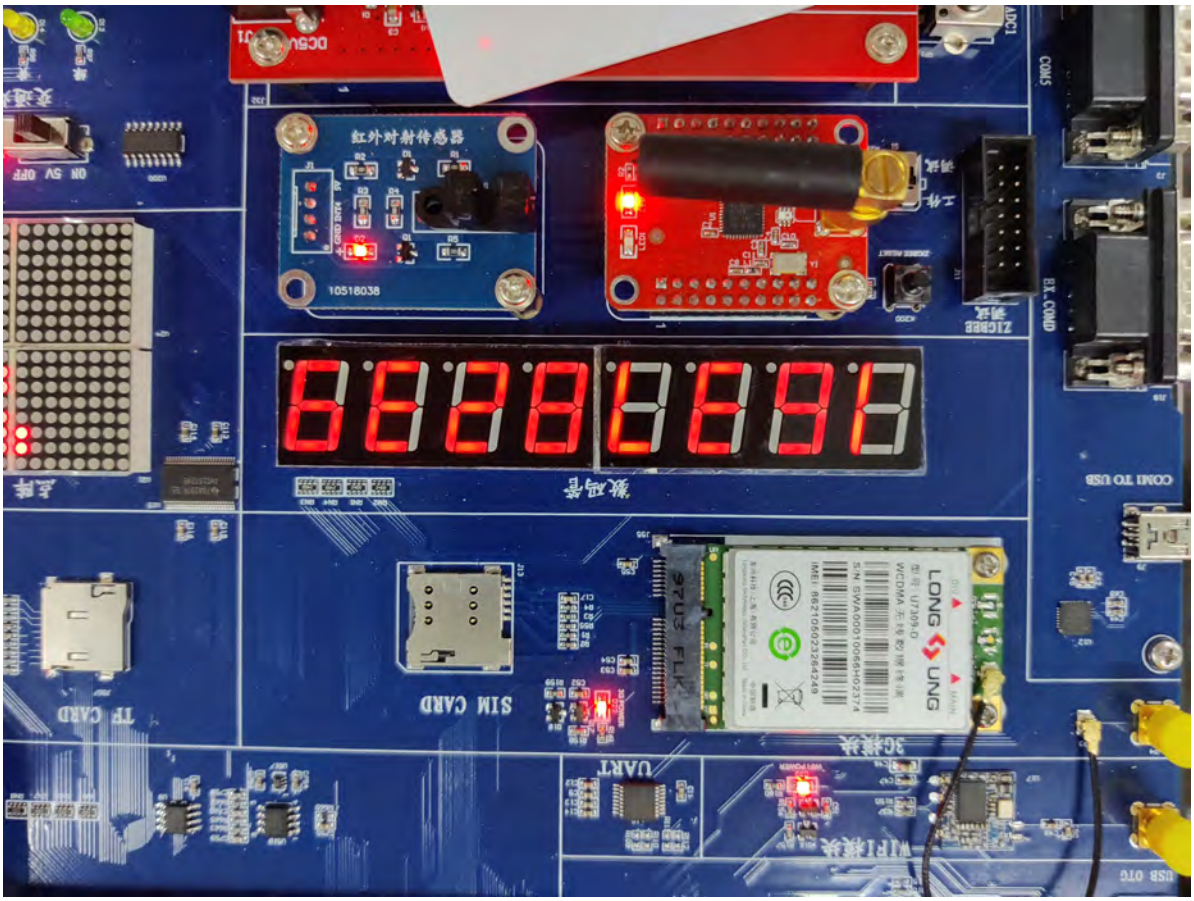


可以发现读取之后屏幕有记录nfc信息



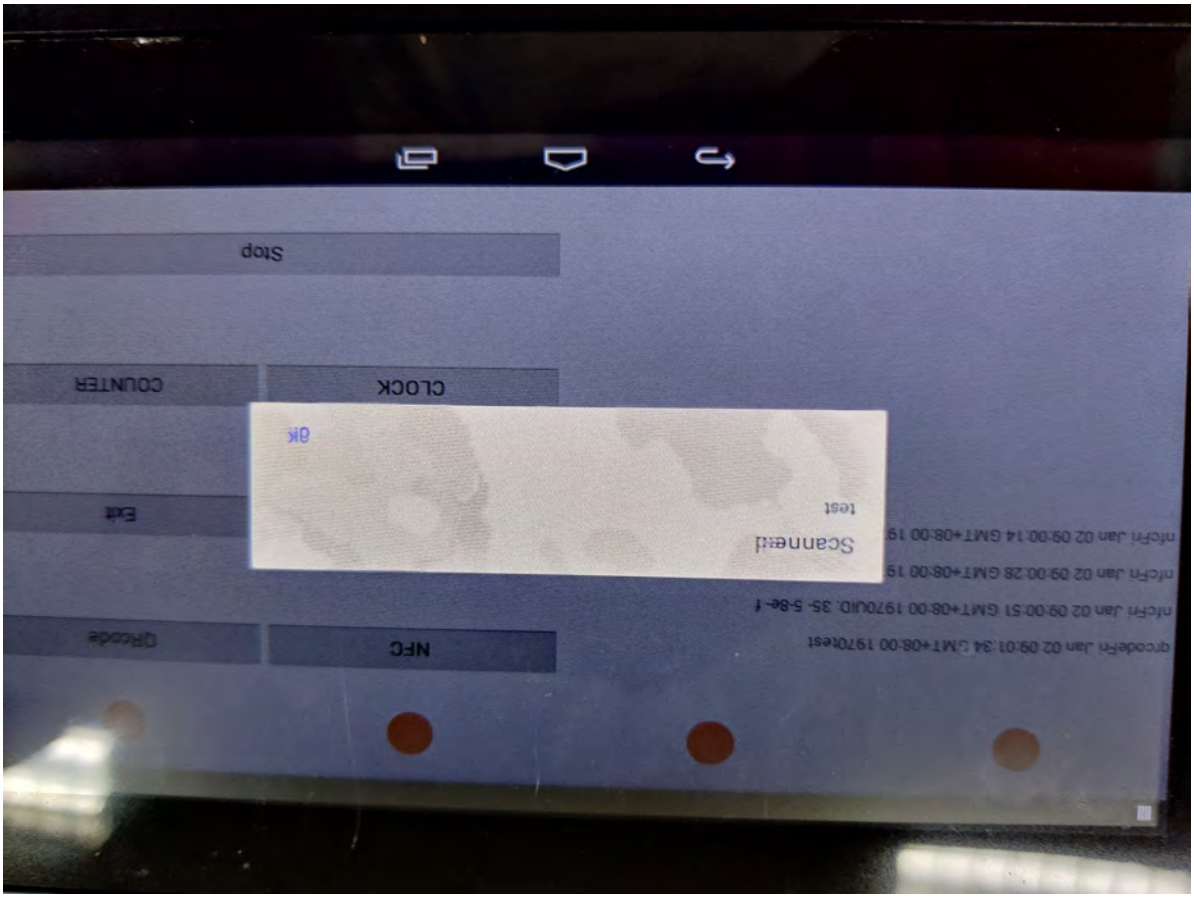
数码管

读取nfc/二维码后可以显示具体的数字

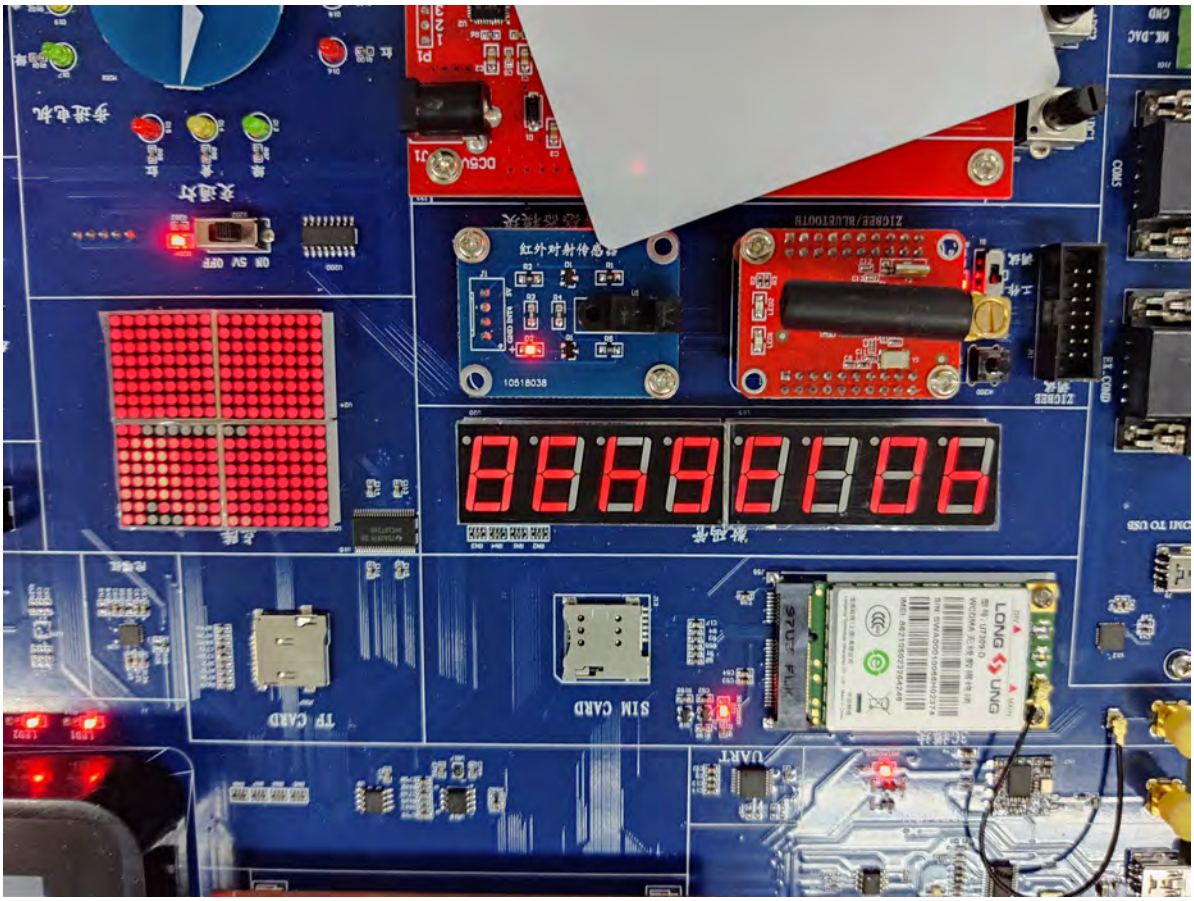


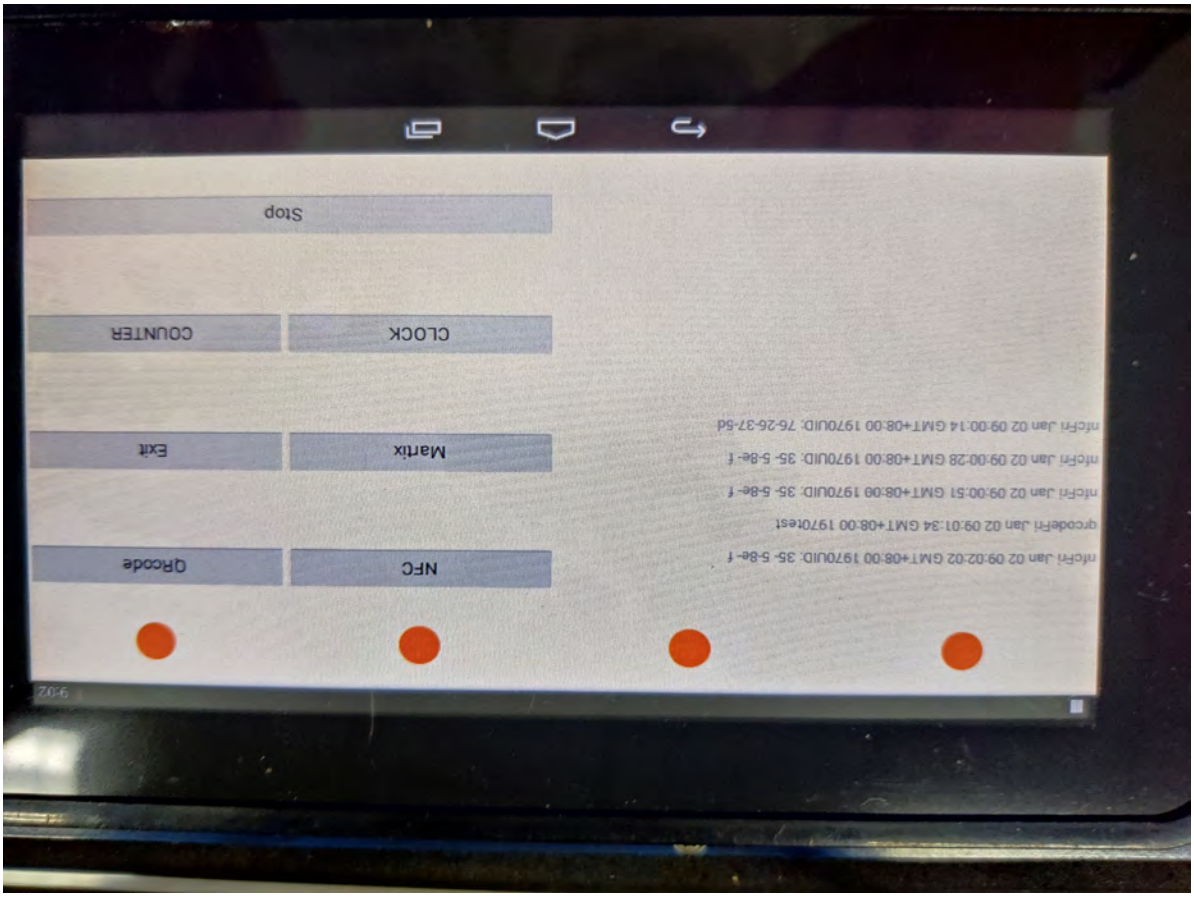
二维码

读取二维码后显示其具体的数值

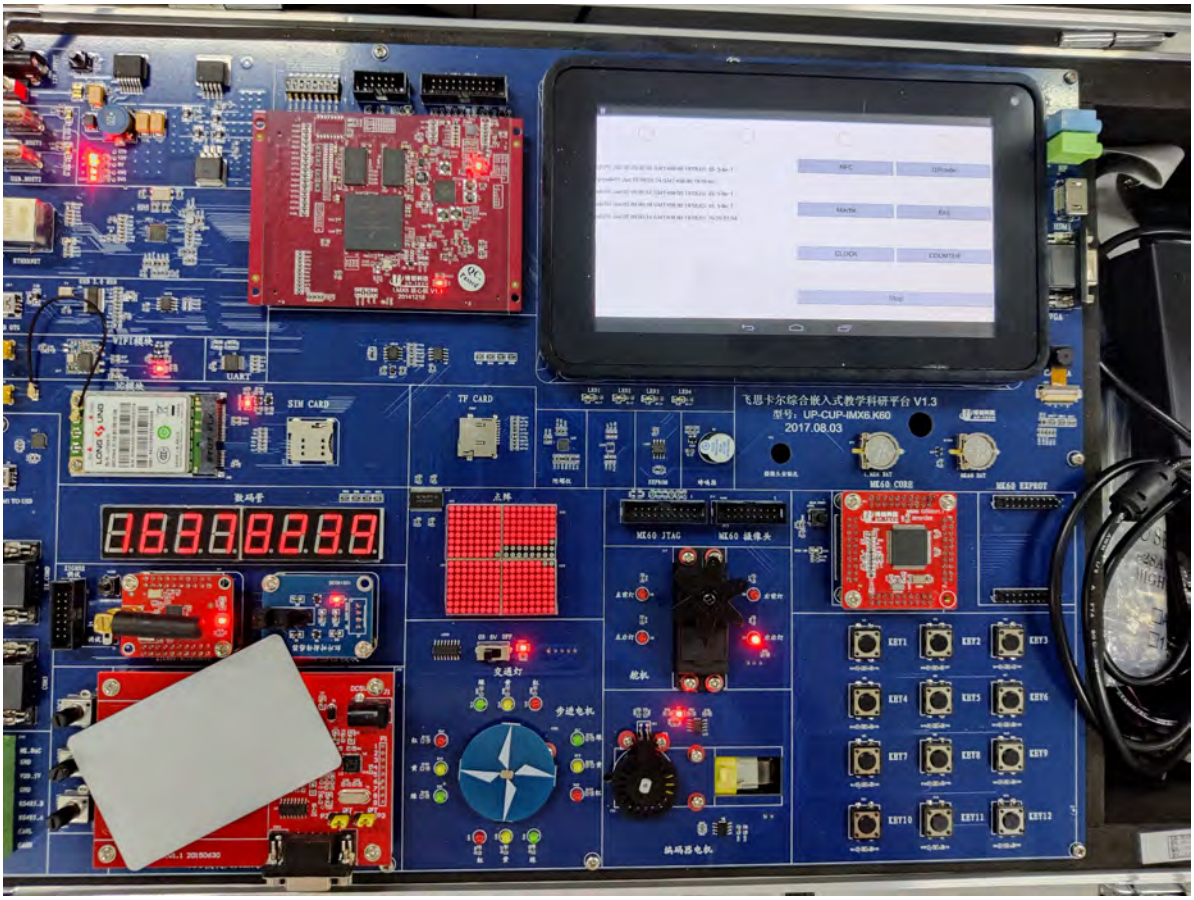


可以发现屏幕上的记录和数码管都有变化





总览



实现代码

请查看压缩包内的代码

分为cppback和安卓两个部分