基于 6818 开发板的嵌入式 Linux 智能车库

1、 项目简介

本项目是在粤嵌学习时做的项目，使用 GEC6818 开发板，搭配触摸屏、USB 摄像头、音响，RFID 卡，模拟实现真实场景中的智能车库常见功能，如:

1、 实时视频监控

2、 自动识别车牌

3、 自动语音播报

4、 停车计费

1、 方案设计

2.1 整体方案

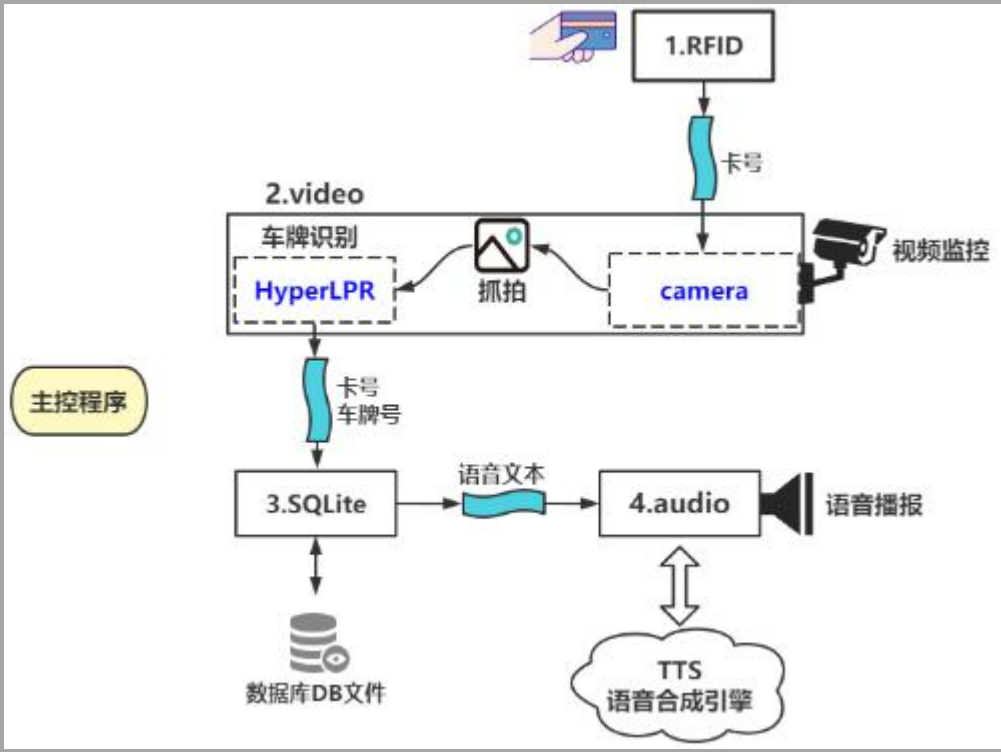
构筑整个项目的代码框架，对视频监控、实时车牌识别、语音播报、数据存储等功能进行分模块设计。整个项目的基本运行流程是：

1. 主控程序依次启动 RFID、Video、SQLite 和 Audio 模块
2. 刷卡得到卡号，将卡号传送给视频模块并触发抓拍

3. 视频模块抓拍后进行车牌识别，然后将卡号和车牌号一并传送给数据库
4. 数据库模块判定卡号的合法性：

- 若合法，增删数据库数据，并将要播报的语音文本传送给音频模块
- 若非法，蜂鸣器鸣响报警
- 数据库可通过按回车键切换出、入库状态

1. 音频模块通过部署在 PC 端的 Ubuntu 虚拟机的 TTS 引擎，板子将文本通过网络传输到虚拟机，虚拟机合成后，通过网络传输给板子，板子获得文本对应的语音后通过喇叭播出。



2.2 整体代码设计

将项目各个子模块单独编译成进程，由主控程序负责启动

项目共分成 5 个模块，分别是：

- 主控模块，负责启动各子模块、常见通信管道
- RFID 模块，负责读取刷卡卡号，并触发摄像头抓拍
- video 模块，负责视频监控，及抓拍并车牌识别
- SQLite 模块，负责管理识别出来的车牌，并实现计费
- audio 模块，负责从网络 TTS 中获取合成语音，并在 Linux 本地上播放出来

1、 项目源码分析

3.1 文件说明

1、 HyperLPR-master：HyperLPR 的源码，里面已经包含了可以在 GEC6818 Linux 开发板直接运行的可执行文件：alpr。只要执行下面的代码，就可以直接识别 a.jpg，识别结果将会被存储在新创建的名为 license 的文件中。

```
system("./alpr a.jpg");
```

2、 include：包含各类头文件，如 SQLite 数据库头文件、jpeg 图片处理头文件、本工程自定义的头文件等。

3、 lib：包含各类动态库，如 jpeg 动态库、SQLite3 动态库。

4、 ttsSDK：语音合成的 SDK 包。

5、 main.c：主函数

6、 camera.c：

7、 RFID.c

8、 SQLite.c

9、 Video.c

10、 CMakeLists.txt

11、 Makefile:

3.2 主函数

对应 main.c 文件，主函数 main()的具体步骤：

1、 创建各种管道，用于各模块之间的数据交互；

```
#define RFID2SQLiteIN  "/tmp/fifo1" // RFID 卡读取到卡号，告诉数据库有车进来，对应卡号要入库
#define RFID2SQLiteOUT "/tmp/fifo2" // RFID 卡读取到卡号，告诉数据库有车出库，对应卡号要从库里删除
#define SQLite2Audio   "/tmp/fifo3" // 数据库模块通过该管道，给音频模块发送文本
#define Video2SQLite   "/tmp/fifo4" // 视频模块识别出车牌号后会通过该管道发送到数据库模块
```

2、 创建信号量，用于各模块之间的互相通知；

```
// 子进程启动成功与否的信号量的名称
#define SEM_OK       "sem1"

//
#define SEM_TAKEPHOTO "sem2"
```

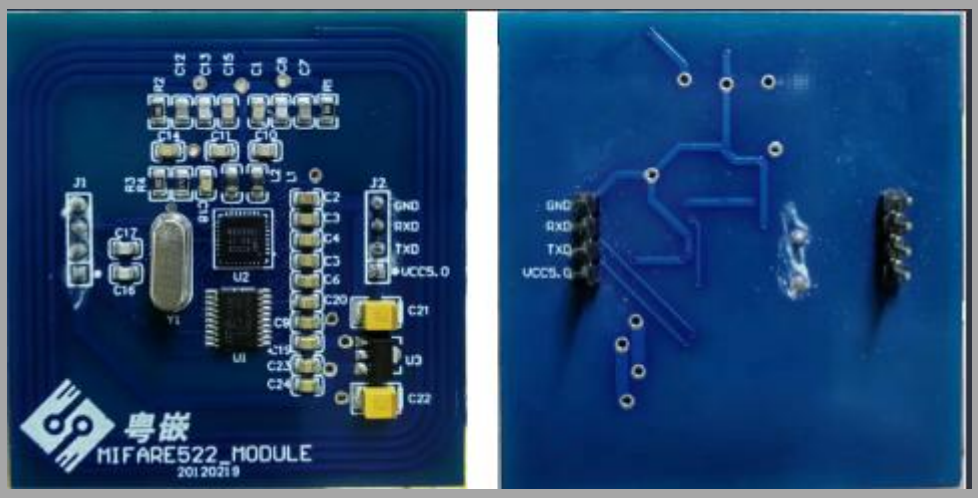
3、 为了体验度更佳，专门创建一个打印…的线程，这样命令行就会一直打印…，可以让用户感受到系统正在运行中；

4、 按顺序开启子模块，分别有如下子模块：RFID 模块、数据库模块、语音模块、视频模块；

5、 最后，开启了 4 个子模块后，主程序就挂起，由 4 个模块之间继续完成智能车库相关功能。

3.3 RFID 模块

本项目使用粤嵌上课发的 RFID 读写模块：



如果你是网上自己买的，和商家拿一下案例代码，只要代码可以在你刷卡，比如我就是用公交卡，公交卡靠近这个 RFID 模块后，代码就会返回一串字符串代表当前公交卡的标识。

相关代码在 RFID.c 文件，里面有详细注释，在此介绍主要有以下大步骤：

- 初始化串口
- 告诉主程序，RFID 子模块启动完成
- 通过串口往 RFID 模块中写入请求命令，判断是否探测到卡片

探测到卡片就读取卡片序列号，并写入数据库

3.4 数据库模块

对应 SQLite.c 文件，main.c 主要流程是：

1，创建、打开一个数据库文件*.db

2，创建表 Table，表头是：卡号、车牌、时间

3.1，准备好与 RFID 模块沟通的通信管道

3.2，准备好与视频模块沟通的通信管道

4.1 ，准备好向主控程序通知本模块启动成功的信号量

4.2， 准备好与视频模块沟通的信号量

5，创建入库、出库线程：

- carIn():用户刷身份卡入车库时，将用户身份卡卡号和用户对应车牌号入库的线程
- carOut():用户刷身份卡出车库时，将用户身份卡卡号入库的线程

6，延时一小会，等本进程一切就绪，向主控程序汇报子模块启动成功

3.5 语音模块

对应 Audio.c 文件，main.c 主要流程是：

1，创建 UDP 通信端点

2，准备好虚拟机的地址结构体和相应的地址信息

3，准备好管道，数据库模块通过该管道，给本音频模块发送文本

3.6 视频模块

对应 Video.c 文件，main.c 主要流程是：

1、 准备 LCD 显示屏、YUV-RGB 映射表

2、 打开摄像头设备、配置摄像头的采集格式、启动摄像头数据采集

3、 准备好和数据库通讯管道，视频模块识别出车牌号后会通过该管道发送到数据库模块

4、 创建一个专门用于抓拍的线程

3.7 整体流程

整个项目的基本运行流程是：

1. 主控程序依次启动 RFID、Video、SQLite 和 Audio 模块
2. 刷卡得到卡号，将卡号传送给视频模块并触发抓拍
3. 视频模块抓拍后进行车牌识别，然后将卡号和车牌号一并传送给数据库
4. 数据库模块判定卡号的合法性：

- 若合法，增删数据库数据，并将要播报的语音文本传送给音频模块
- 若非法，蜂鸣器鸣响报警
- 数据库可通过按回车键切换出、入库状态

1. 音频模块通过部署在 PC 端的 Ubuntu 虚拟机的 TTS 引擎，板子将文本通过网络传输到虚拟机，虚拟机合成后，通过网络传输给板子，板子获得文本对应的语音后通过喇叭播出。

Main 函数传入三个参数 ， 否则退出打印

```
./main /dev/ttySAC2 /dev/video7
采用命名管道，创建 mkfifo
```

调用 open()打开命名管道的进程可能会被阻塞，但如果同时用读写方式（ O_RDWR）打开，则一定不会导致阻塞；这里利用的是读写方式，

无名信号量用于父子进程之间， **命名信号量用于无血缘关系**

```
sem_open（） 创建一个信号量并打开
signal(SIGCHLD, quit);子进程启动失败调用 quit 杀死所有进程

signal(SIGINT/*ctrl+c*/, cleanup); ctrl+c 也是杀死
```

创建 rfid 通过

```
execl("./RFID", "RFID", argv[1], NULL); 传入串口文件的位置
    int fifoIN  = open(RFID2SQLiteIN,  O_RDWR);       //可读可写 成功返回
fd ,失败返回-1
    int fifoOUT = open(RFID2SQLiteOUT, O_RDWR);
```

```
RFID 卡读取到卡号，告诉数据库有车进来，对应卡号要入库
RFID 卡读取到卡号，告诉数据库有车出库，对应卡号要从库里删除
```

打开串口，初始化串口。。。。 将串口设置为独立终端，非阻塞模式 ，信号量加一告知主线程成功

创建 in_out 线程，通过 getchar（）阻塞实现手动切换 state 入库出库，触发发给数据库的条件，

创建打印线程，区分入库出库；

开始配置 rfid 卡寄存器，通过串口像卡写入命令，等待 10ms 读取判断是否有卡，若有卡则打印换一个打印方式，如果有卡就读取卡号，并输出。 如果 flag=true 卡片刚放上去，判断是出库还是入库，入库：通过管道 fifoin 写入数据库，出库：通过 fifoout 发给数据库删除，完成后 flag 为 false

识别卡片为 ture

第二个进程为 sqlite3，轻量级，将其制作成动态库，链接动态库的方式去编译

```
  // 1，创建、打开一个数据库文件*.db
    int ret = sqlite3_open_v2("parking.db", &db,
                              SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE,
NULL);




   // 2，创建表 Table，表头是：卡号、车牌、时间 不存在则创建
    sqlite3_exec(db, "CREATE TABLE IF NOT EXISTS info"
             "(卡号 TEXT PRIMARY KEY, 车牌 TEXT, 时间 TEXT);",
                 NULL, NULL, &err);  //定义卡号为主键， 类型都为 text  //
```

3 open 与 RFID 的管道　open 与视频模块的管道　open 与语言模块的管道

4　准备与主程序的信号量， 准备与视频模块的信号量

5　 创建出库入库线程　in　out

6　主线程延迟一会，报告给主程序，sem_post

7　主线程退出

Car in

```
// A. 静静地等待 RFID 发来的入库卡号并读取，对应写入身份卡卡号的代码在 RFID.c 文件里
        read(fifoIN, &id, sizeof(id));

        // B. 检测该入库卡号是否合法
        bzero(SQL, 100);
        snprintf(SQL, 100, "SELECT * FROM info WHERE 卡号='%x';", id); //功能：检查卡号 id 是否已存在于数据库中。 写入　、、查看卡号 id 是否已存在于名为 info 的数据库表中
        int n = 0;
        sqlite3_exec(db, SQL, callback, &n/*用户自定义参数*/, &err);  //因为 sqlite3 把数据查出来，得通过回调告诉你查出了什么数据
```

每查询到一次 n++ 如果 n>0,则报警，

如果 n=0， 信号量 takephoto 加一，通知视频模快抓拍

读取

```
、、读取 fifoFromVideo　管道对应抓拍图片识别出来的车牌号
```

```
通过 fifoToAudio 管道　　给音频模块发送欢迎文本欢迎。。车牌入场
```

将当前车牌号入数据库

```
// 将当前车牌号也入库
          snprintf(SQL, 100, "INSERT INTO info VALUES"
                             "('%x', '%s', '%lu');", id, license_plate,
time(NULL)); //记录当前时间
          sqlite3_exec(db, SQL, NULL, NULL, NULL);
全显示数据库中的数据一次
```

Car out 同样查询

如果 n==0　报警

如果存在 n=1　删除记录，

打开

```
fifoToAudio 管道，给语音模块发送收费文本
 write(fifoToAudio, msg, strlen(msg));
```

显示所有数据库

Execl 语音模块，创建子进程，利用科大讯飞平台的 sdk，离线语音合成模块 api，利用 alsa-utils 工具 使用 **aplay** 播放 **wav** 文件。

ubuntu 编译主机中运行改造后的 **SDK**，令其绑定 **IP** 并等待开发板的语音模块发来语音文本，**SDK** 接收到文本并合成语音文件。接着，由于每次合成的语音都是不同的，因此要先将语音数据的长度发给开发板语音程序，再发送实际的语音数据，这样才能让开发板语音程序正常接收语音文件数据。

开发板上

```
// 1，创建 UDP 通信端点
   int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
// 2，准备好虚拟机的地址结构体和相应的地址信息
   struct sockaddr_in addr;
   socklen_t len = sizeof(addr);
   bzero(&addr, len);
   addr.sin_family      = AF_INET;
   addr.sin_addr.s_addr = inet_addr("192.168.9.100");
   addr.sin_port        = htons(50001); // 转换成网络字节序

3 准备好 SQLite2Audio 管道
4 向 main 发送成功 sem_post(s)
5 随时准备将 SQLite 发来的文本转成语音并播放出来 while（1）
 { A. 静静地等待 SQLite 发来的语音文本 read(fifo, text, 500);

 B. 发送给 SDK 去帮忙合成语音
 C. int n = sendto(sockfd, text, strlen(text), 0,
```

```
                        (const struct sockaddr *)&addr, len);
// C. 静静地等待对方发来对应语音文件的大小
        uint32_t size;
        recvfrom(sockfd, &size, sizeof(size), 0, NULL, NULL);

// D. 创建一个专门用来存储对方语音数据的 wav 文件
        int fd = open("a.wav", O_RDWR|O_CREAT|O_TRUNC, 0777);
 开辟内存（10*1024） 每次读取 sdk 语音数据不超过 10kb
// 返回值 n 代表真正读取到的数据字节数
    int n = recvfrom(sockfd, wav, 10*1024, 0, NULL, NULL);

// 将这 n 个字节的数据，妥善地保存到文件 a.wav 中
            write(fd, wav, n);

            size -= n;



}
system("aplay a.wav"); 播放
```

Ubuntu 上调用示例

1 创建 udp 通信

2 准备 ip 和端口号

3 绑定地址

```
  if(bind(sockfd, (struct sockaddr *)&addr, len) != 0)
    {
        perror("绑定地址失败");
        exit(0);
    }
    else
        printf("绑定地址成功\n");
```

4 等待开发板的文本信息

While（1）{

```
1 准备好开发板的地址结构体和相应的地址信息
struct sockaddr_in armAddr;
        len = sizeof(armAddr);
        bzero(&armAddr, len);
```

```
/ B. 接收开发板发来的信息，并且得到对方的地址
 recvfrom(sockfd, text, 500, 0, (struct sockaddr *)&armAddr, &len);


C 使用 TTS 语音合成引擎转换成 wav 文件
    D. 取得语音文件的大小 光标移到末尾获取大小后移到开头
        int fd = open("a.wav", O_RDWR);
 E. 将文件大小发给开发板

 F. 取得语音文件的内容并将之发给开发板
char *buf = malloc(10*1024); // 申请 10k 的内存缓冲区  利用 while（1）每次发
哦送 10k
释放资源
```

}

视频模块进程创建

在应用程序中，操作/dev/fbX 的一般步骤如下：
①、首先打开/dev/fbX 设备文件。
②、使用 ioctl()函数获取到当前显示设备的参数信息，譬如屏幕的分辨率大小、像素格式，根据屏幕参数计算显示缓冲区的大小。
③、通过存储映射 I/O 方式将屏幕的显示缓冲区映射到用户空间（mmap）。
④、映射成功后就可以直接读写屏幕的显示缓冲区，进行绘图或图片显示等操作了。
⑤、完成显示后，调用 munmap()取消映射、并调用 close()关闭设备文件。

1 打开 lcd 显示屏设备　　framebuffer 帧缓冲　LCD 800*480 ，像素深度 16，一个像素点两个字节

2 获取 lcd 显示器的设备参数，使用 ioctl，存储行列像素

3 存储映射 I/O mmap 显存映射到进程的地址空间中

随后获取 lcd RGB 的像素偏移量 offset

4 打开摄像头文件

5 设置摄像头格式参数（视频设备 type 字段 V4L2_CAP_VIDEO_CAPTURE）

```
  // 试图设置摄像头的分辨率
   tmp->fmt.pix.width  = 640;
   tmp->fmt.pix.height = 480;
tmp->fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
```

```
    // 设置摄像头捕获图像的方式
    tmp->fmt.pix.field = V4L2_FIELD_INTERLACED; //采用隔行扫描方 交错式
```

6 查看已调像素格式 （查看摄像头支持的像素格式，获取摄像头基本参数）

```
// 将摄像头的图像宽度、高度存储在变量中
    CAMERA_W = fmt.fmt.pix.width;
    CAMERA_H = fmt.fmt.pix.height;
```

7 申请帧缓存 建立内存映射

```
ioctl(camfd, VIDIOC_QUERYBUF, &buffer[i]);
```

查询 buffer 帧缓冲的长度偏移量信息 内存映射

```
start[i] = mmap(NULL, buffer[i].length,   PROT_READ | PROT_WRITE,
                        MAP_SHARED, camfd, buffer[i].m.offset);
```

8 入队

9 开启摄像头采集

10 读取帧缓存中的映射的数据，出队后，读取一帧数据，显示在 lcd 上，再入队，再读取

```
// 准备好和数据库通讯管道
    fifo = open(Video2SQLite, O_RDWR);
  // 准备好向主控程序通知本模块启动成功的信号量
    s_ok        = sem_open(SEM_OK, 0666);

    //准备好与视频模块沟通的信号量
    s_takePhoto = sem_open(SEM_TAKEPHOTO, 0666);
```

11 创建一个抓拍线程 ，像主控汇报 当数据库发送信号量

```
 sem_wait(s_takePhoto);
```

抓拍一张图

12 播放视频内容

音频模块

硬件：通过 SAI 接口外接了一个 WM8960 音频与 DAC 芯片

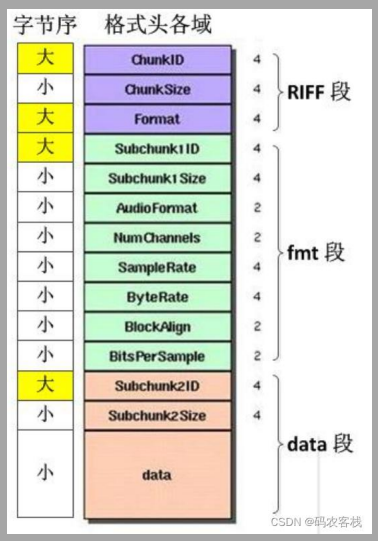
接口：ADC 输入，帧时钟 ， DAC 输入 帧时钟，BCLK 同步时钟， 主时钟，控制接口 iic，配置作用

I2s 协议
sck：串行时钟信号位时钟 ws：选择信号帧时钟 sd:串行数据信号

主要配置 IIc（codec 芯片驱动） SAI（SOC 接口驱动） machine （sound 部分结合）修改设备树

1. 打开 pcm 设备，实例化 hwparams 对象，配置参数
2. 获取 PCM 设备当前硬件配置,对 hwparams 进行初始化
3. 设置访问类型: 交错模式 //一个音频的样本是由两个单声道的样本交错地进行存储得到的，即 A1-B1-A2-B2-A3-B3 样式存储。
4. 设置数据格式: 有符号 16 位、小端模式 ,声道数位 2 立体声，采样率大小 44100 。将周期大小设置为 1024 帧（一个周期其实就是两次硬件中断之间的帧数,一段音频数据就是由若干帧组成的），设置 驱动层 buffer 大小为 16 个周期，
5. 加载配置
6. 播放：从应用层 buffer 向驱动缓冲区写入数据，每次一个周期的数据

Wav 格式
其实也就是对它的头部数据进行校验、解析，获取音频格式信息以及音频数据的位置偏移量，阻塞的写入一个周期，写满等待播放完再写入，


字节序
格式头各域
大
小
大
大
小
小
小
小
小
小
小
大
小
小
ChunkID
ChunkSize
Format
Subchunk1ID
Subchunk1Size
AudioFormat
NumChannels
SampleRate
ByteRate
BlockAlign
BitsPerSample
Subchunk2ID
Subchunk2Size
data
4
4
4
4
4
2
2
4
4
2
2
4
4
RIFF 段
fmt 段
data 段
CSDN @码农客栈