# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | TIM PWM 与蜂鸣器音乐播放器 |
| 姓 名： | 王小龙 |
| 学 号： | 2020211502 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 刘健培、戴志涛 |

2022年 12 月 8 日

# 实验目的

⚫ 学习使用 STM32 定时器 TIM 的基本操作方式。

⚫ 学习使用 STM32 定时器 TIM 输出 PWM 波的方式。

⚫ 学习操作系统环境下编写设备驱动的方式。

⚫ 学习基于 RTOS 的多任务应用程序设计与编程方式。

⚫ 学习在一个综合应用中同时驱动多个外设的方法。

# 实验环境

⚫ FS-STM32F407开发平台

⚫ ST-Link仿真器

⚫ RealView MDK5.23集成开发软件

⚫ PC机Window10 (64bit)

⚫ 串口调试工具

# 实验要求

⚫ 使用 PWM 波驱动蜂鸣器发声，实验一个简单的蜂鸣器音乐播放器。

⚫ 播放器可以通过按键进行控制，支持：显示播放列表、暂停/播放、上一

曲、下一曲、音量调节等功能。

◼ K3——短按，暂停/播放键。

◼ K4——短按作为下一曲的按键，长按为音量加。

◼ K5——短按作为上一曲的按键，长按为音量减。

⚫ 在参考代码的基础上，基于操作系统的I/O子系统框架，编写通用的PWM

驱动，实现音乐的播放。

# 实验原理

**1. 2 种类型的 I/O 设备驱动**

设备驱动是指驱使硬件设备行动的特定软件，是与硬件设备直接打交道的底

层软件。

主要功能是负责应用软件和硬件设备之间的沟通，将应用程序的标准调用映

射为设备特定的命令序列(操作寄存器)。

向下，直接管理底层硬件设备，按照设备的具体工作方式，读写设备控制器

（device controller）的寄存器，完成设备的轮询、中断处理或 DMA 通信

（Device-specific ）

向上，为上层程序提供操作设备的编程接口，隐藏硬件设备控制的具体细节

（OS-specific ）

核心操作有：发送数据给设备、从设备接收数据、控制设备。

根据操作系统是否定义了驱动架构，可把驱动分为 2 类：不基于 OS 的设备

驱动与基于 OS 的设备驱动。

不基于 OS 的设备驱动，使用非标准接口。接口可根据硬件设备的特点自行

定义或和应用程序协商定义应用程序直接调用该接口来使用设备。应用与驱动紧

耦合。例如：

无操作系统、简单 RTOS 中的设备驱动。本课实验 2-3 是这一类。基于 OS 的设备驱动，则使用标准接口。OS 定义了驱动的架构和独立于具体设备的抽象接口，驱动必须实现这些接口才能融入 OS。应用程序通过 OS 的API 间接调用驱动接口。应用、OS 与驱动相对独立。例如：RT-Thread、Linux等操作系统下的设备驱动。本实验可归于这一类。

具体情形下，使用哪种模型实现设备驱动？一般如果要向应用程序提供标准

的 IO 操作接口（open/close/read/write/ioctl）、兼容已有应用程序、提高可移植性时，使用标准接口。如果设备很简单、对速度要求很高、或不满足 IO 模型时，

常选择非标准接口。

**2. 操作系统 I/O 子系统框架下的设备驱动**

本实验的主要目的之一即是学习如何在操作系统环境下编写标准接口的设备驱动，虽然不同操作系统实现的 I/O 设备模型框架细节不同，但主要的层次以及设计方式是很类似的。

以 RT-Thread 为例，它提供了一套简单的 I/O 设备模型框架，如下图所示，

它位于硬件和应用程序之间，共分成三层，从上到下分别是 I/O 设备管理层、

设备驱动框架层、设备驱动层。

应用程序通过 I/O 设备管理接口获得正确的设备驱动，然后通过这个设备

驱动与底层 I/O 硬件设备进行数据（或控制）交互。

I/O 设备管理层实现了对设备驱动程序的封装。应用程序通过 I/O 设备层

提供的标准接口访问底层设备，设备驱动程序的升级、更替不会对上层应用产生

影响。这种方式使得设备的硬件操作相关的代码能够独立于应用程序而存在，双

方只需关注各自的功能实现，从而降低了代码的耦合性、复杂性，提高了系统的

可靠性。

设备驱动框架层是对同类硬件设备驱动的抽象，将不同厂家的同类硬件设备

驱动中相同的部分抽取出来，将不同部分留出接口，由驱动程序实现。

设备驱动层是一组驱使硬件设备工作的程序，实现访问硬件设备的功能。它

负责创建和注册 I/O 设备，对于操作逻辑简单的设备。

**3. 音乐数据的 PWM 编码与解码**

声音的三要素是：响度、音色、音调。利用这三个要素就可以演奏不同的音

乐。一般说来，蜂鸣器演奏音乐只能是单音频率，它不包含相应幅度的谐波频率，

也就是说不能象电子琴那样能奏出多种音色的声音。因此蜂鸣器奏乐只需弄清楚

两个概念即可，也就是“音调”和“节拍”。音调表示一个音符唱多高的频率，节拍

表示一个音符唱多长的时间。十二平均律就规定了每一个音符的标准频率。

十二平均律，是一种音乐定律方法，将一个纯八度平均分成十二等份，每等

分称为半音，是最主要的调音法。十二平均律中各音的频率：

C: 262 Hz、#C: 277

Hz、D: 294 Hz、#D: 311 Hz、E: 330 Hz、F: 349 Hz、#F: 370 Hz、G: 392 Hz、#G:

415 Hz、A: 440 Hz、#A: 466 Hz、B: 494 Hz

**4. 使用定时器的 PWM 输出驱动蜂鸣器**

只有无源蜂鸣器才能够用来播放音乐。因为无源蜂鸣器振动的频率是可调的，

而有源蜂鸣器的振动频率是固定的。只有频率（也就是音高）可调，才能够播放

简单的音乐。由于无源蜂鸣器需要有震荡信号才能发出声音，所以需要使用

PWM 设备来控制蜂鸣器的播放。

通过查《FS\_STM32F4 底板原理图 V1.pdf》，获知蜂鸣器使用 PF6 管脚，并

且是高电平驱动发声。即 PF6=1：发声，=0：灭声。

通过查 《 STM32F4x7-Datasheet.pdf 》， 获 知 PF6 对应的 TIM 通 道 是

TIM10\_CH1，即 TIM10 的通道 1。

所以，实验使用 TIM10 的通道 1。

**5. TIM 的 PWM 输出模式**

PWM 输出就是对外输出脉宽（即占空比）可调的方波信号。

# 实验步骤

1.阅读 project4\_main 下的代码，理解程序结构与实现方式。

2.参考 pin\_driver.c 中 pin 设备驱动的实现方式，完成 pwm\_driver.c 中

pwm 设备驱动需要完成的接口函数。

需要实现的 4 个接口函数是：

int stm32\_pwm\_close(struct device\_t \*dev, void\* file\_data)

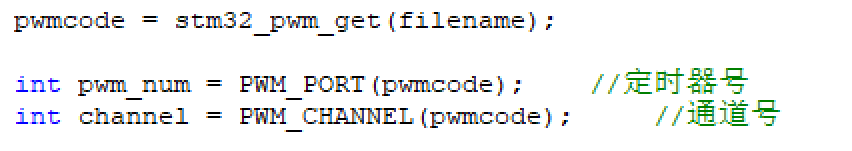
int stm32\_pwm\_open(struct device\_t \*dev, const char \*filename, int flags, int mode)

int stm32\_pwm\_write(struct device\_t \*dev, void\* file\_data, char \*buffer, int size)

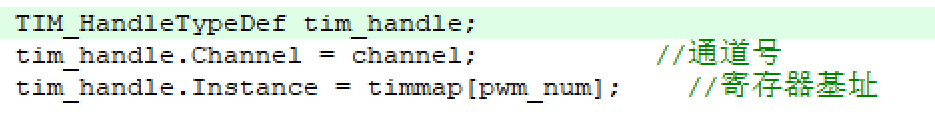
int stm32\_pwm\_ioctl(struct device\_t \*dev, void\* file\_data, int cmd, void \*args)

3.首先实现int stm32\_pwm\_open

从filename提取定时器号和通道号，编码到pwmcode中，



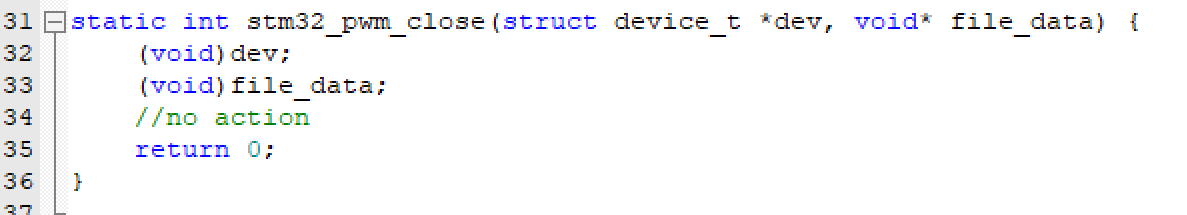
然后实现参数TIM\_HandleTypeDef tim\_handle;为调用HAL\_TIM\_PWM\_Start（）函数做准备



总体代码如下：

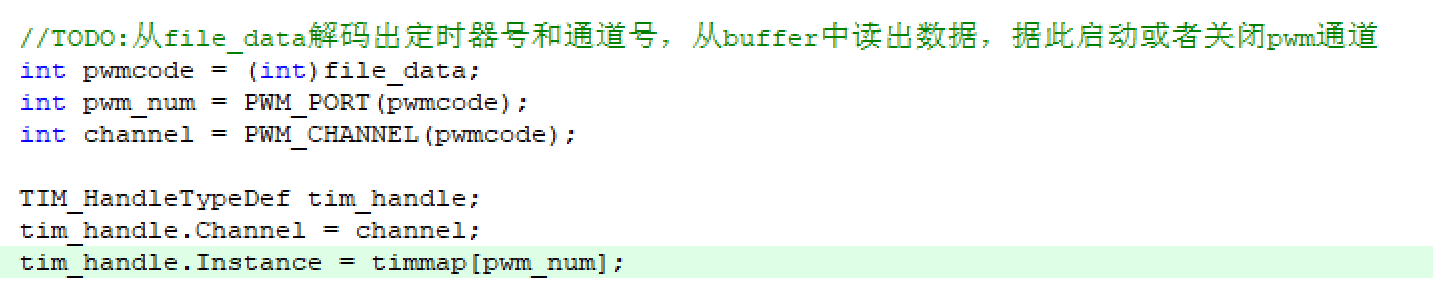


4.int stm32\_pwm\_close函数不用修改：



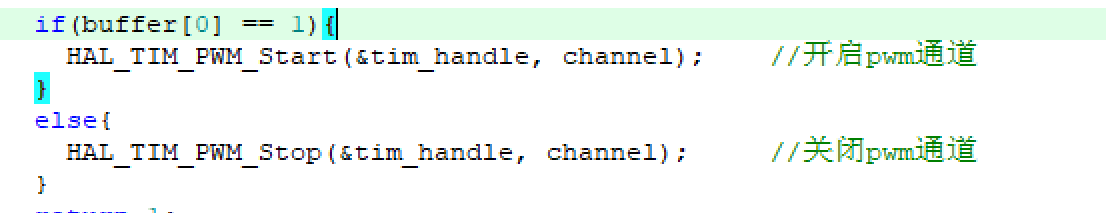
5.int stm32\_pwm\_write函数：

此函数和open函数前面思路类似：

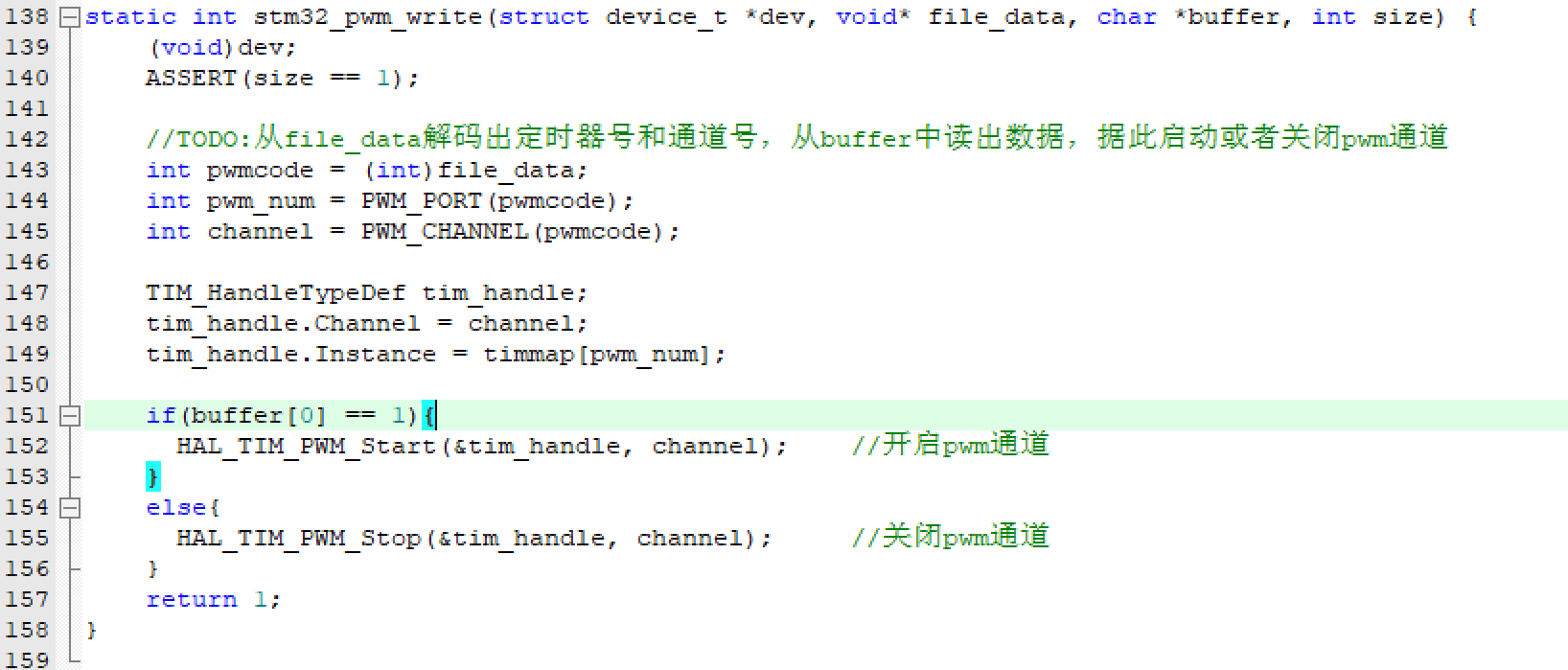


然后通过传入的参数buffer第一位来判断是否需要开启通道，这个buffer是由

pwm\_device.c里面的函数操作的。

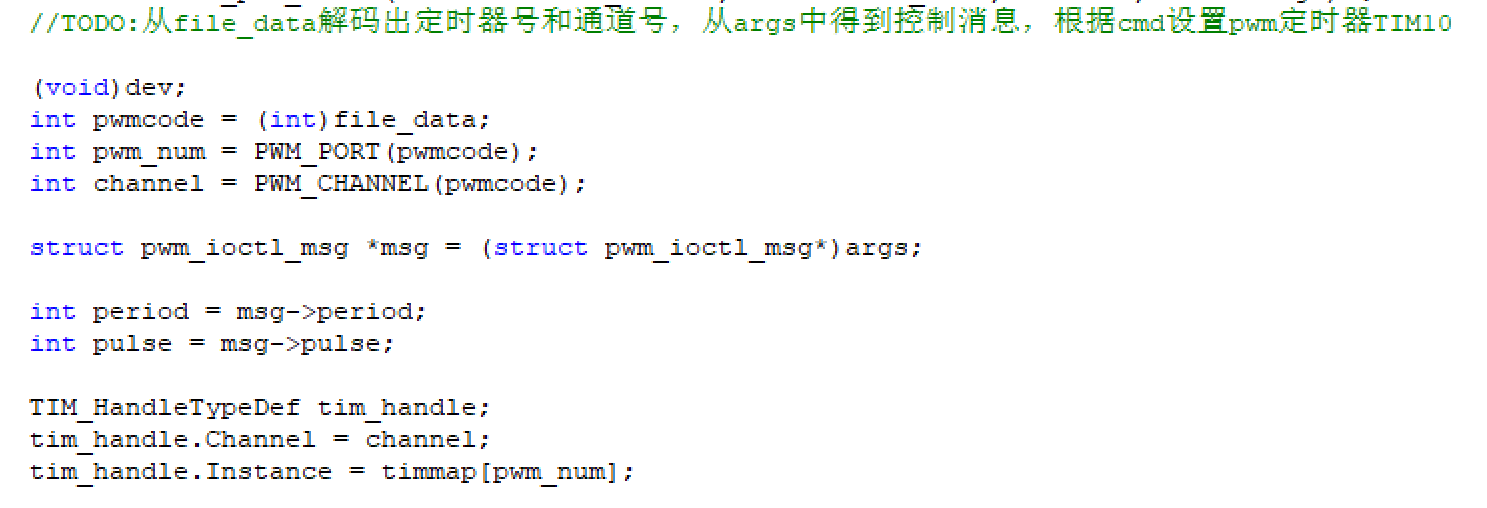


总体函数如下：

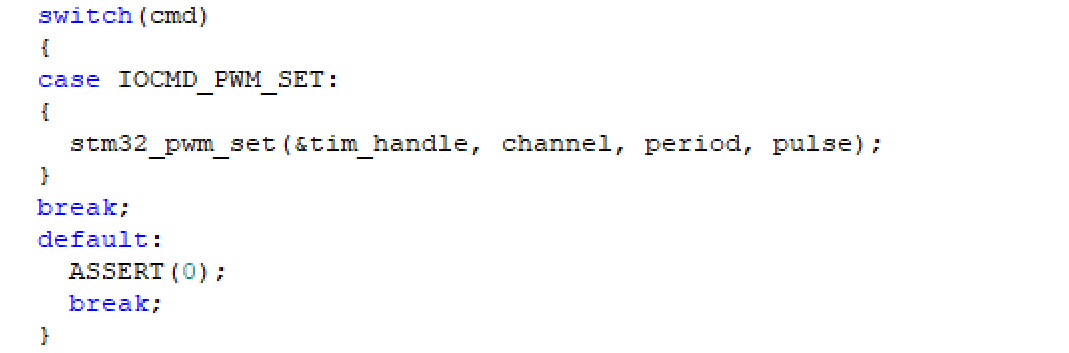


6.int stm32\_pwm\_ioctl函数

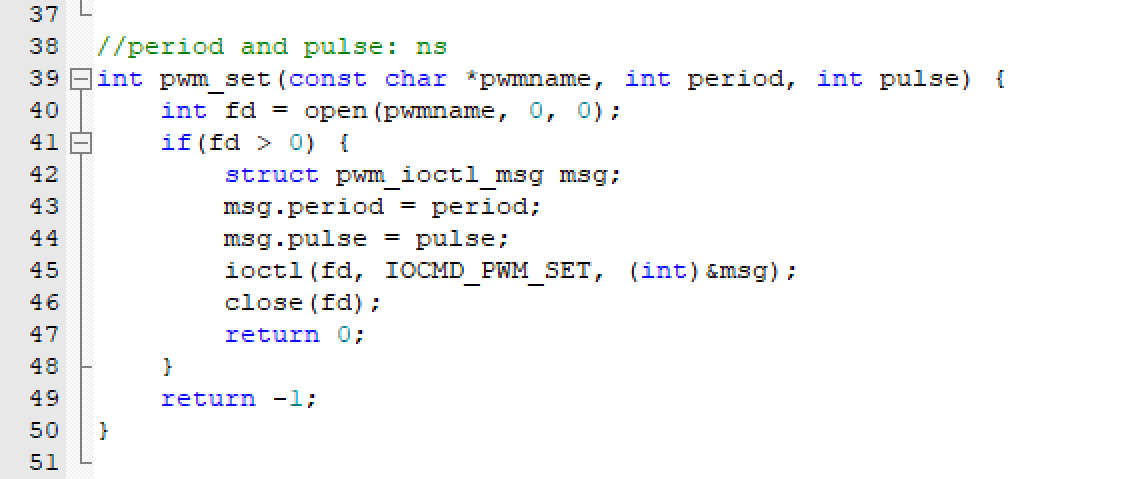
此函数还是类似open函数：

区别就是从msg里面需要获得周期和占空比来传入set函数

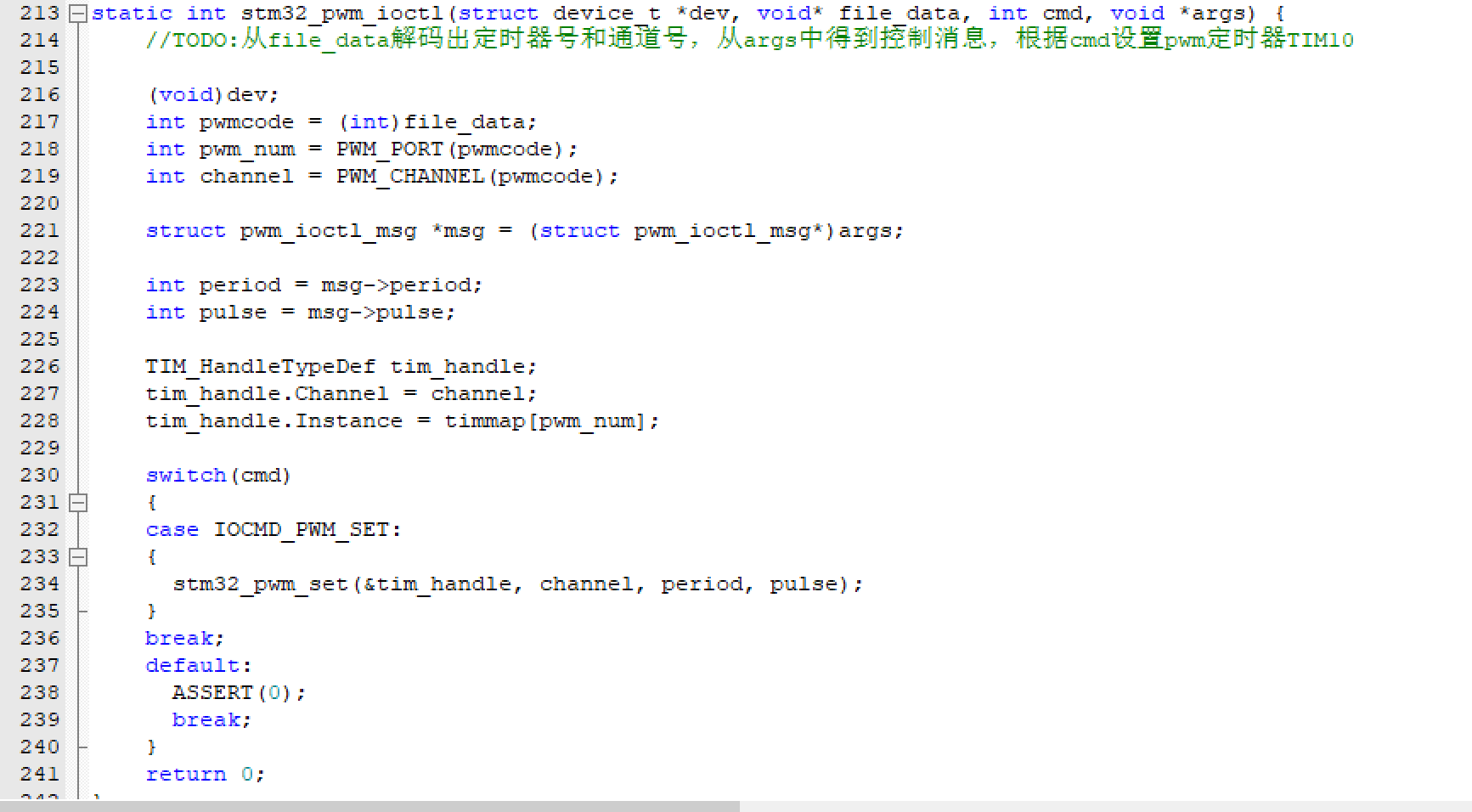
然后还要根据传入的参数cmd判断是不是要调用set函数：



这个cmd参数是从pwm\_device.c里面的如下函数修改的：



总体函数如下：



# 实验方案与实现

## 软件结构

播放器由 player.c 控制，初始化时启动一个任务 player\_entry 控制播放

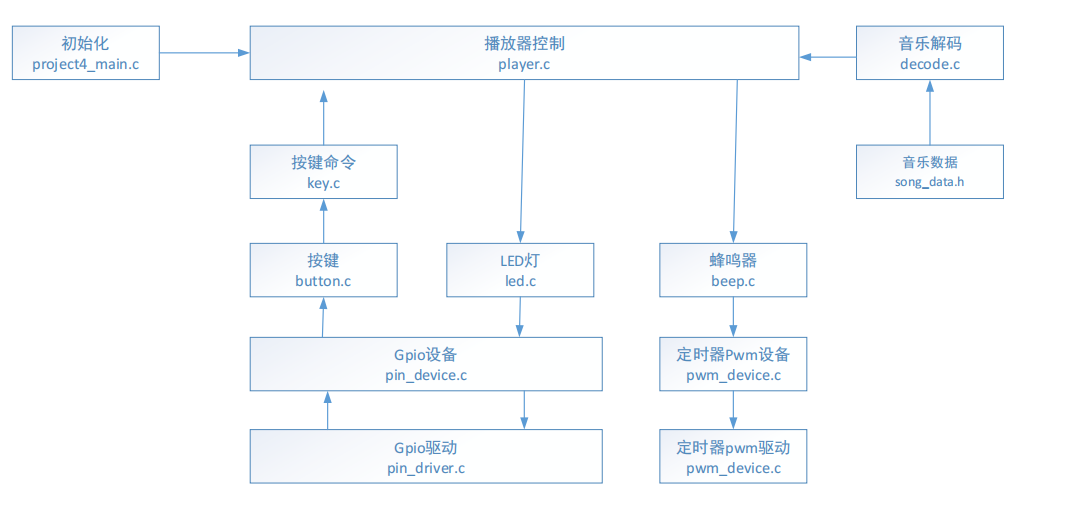
器的暂停与播放。

控制命令来自 key.c ，初始化是启动了一个 20ms 周 期 的 定 时 器

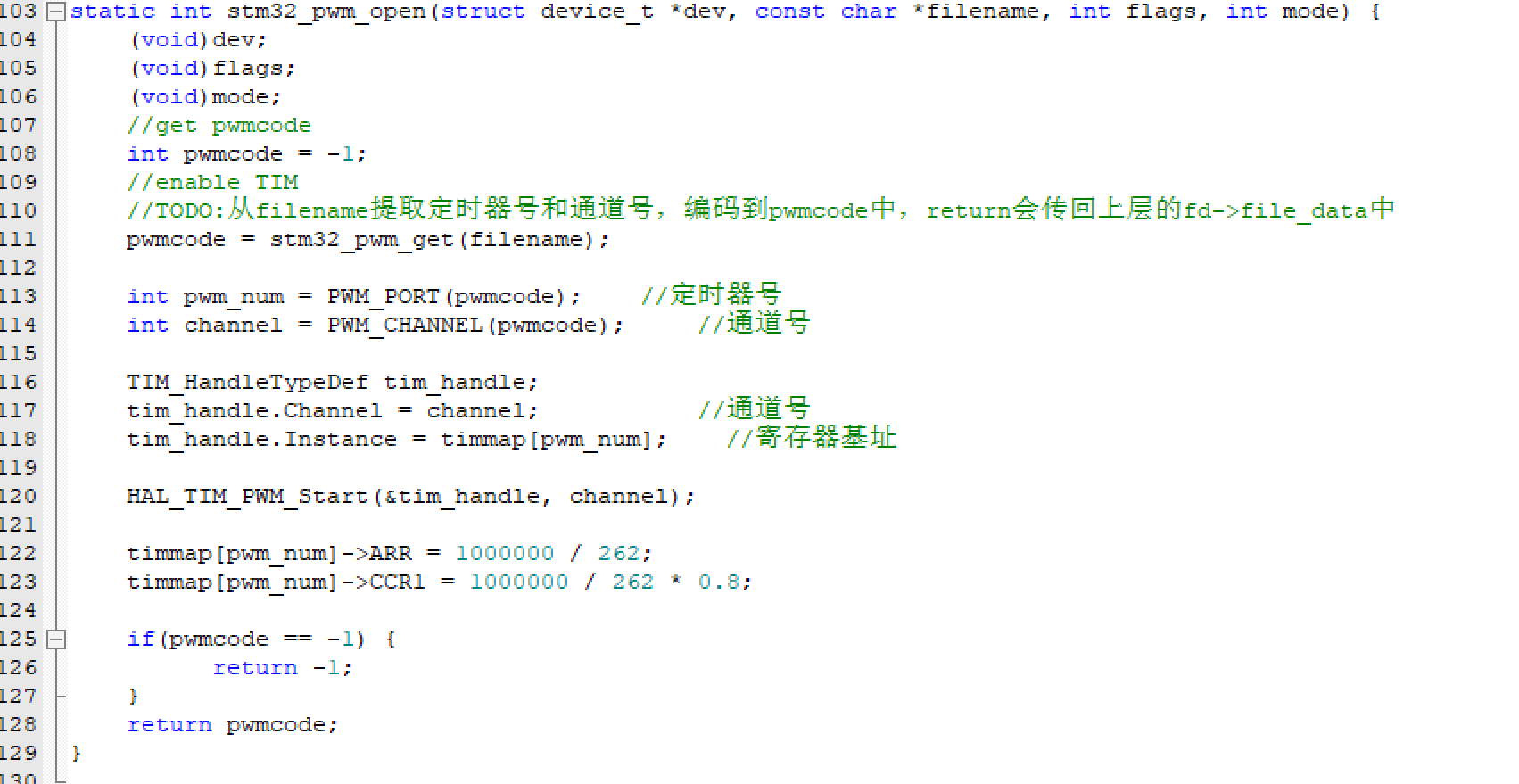
button\_manage.timer，定期扫描按键的值。

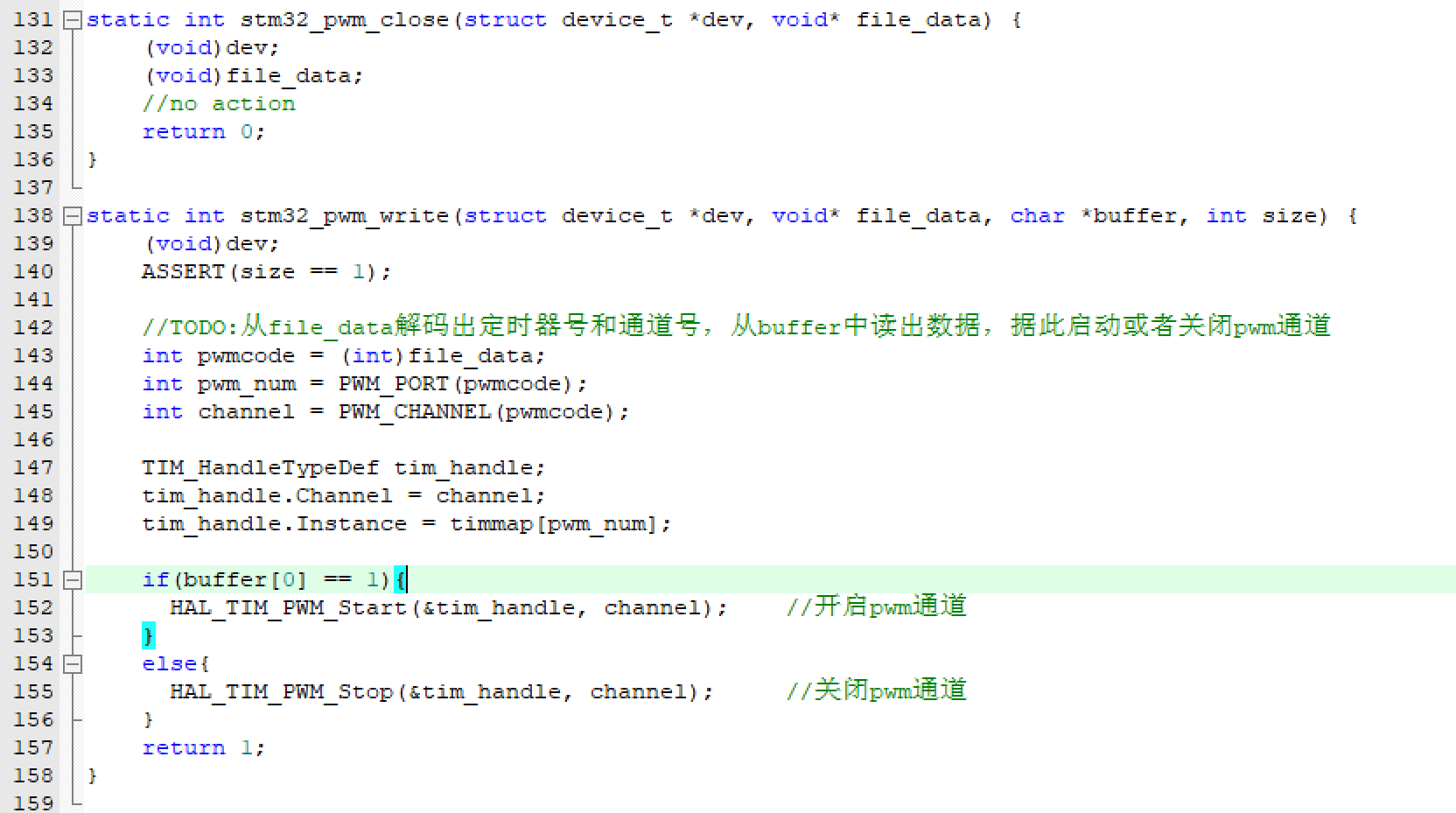
音乐数据来自 song\_data.h，调用 decode.c 解码后，调用 beep.c 改变

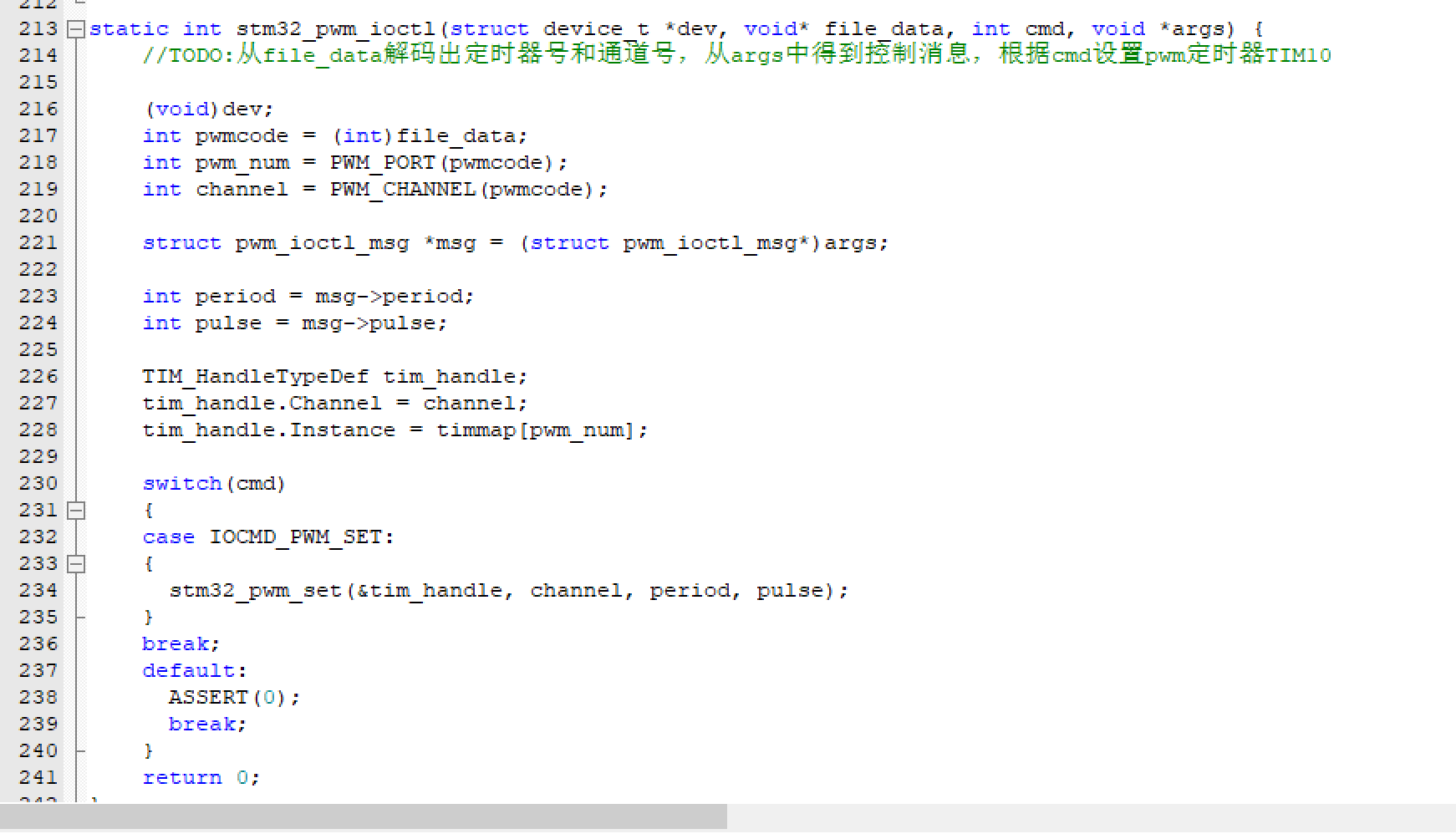
TIM10 的 PWM 输出，控制蜂鸣器的声音。



## 源代码

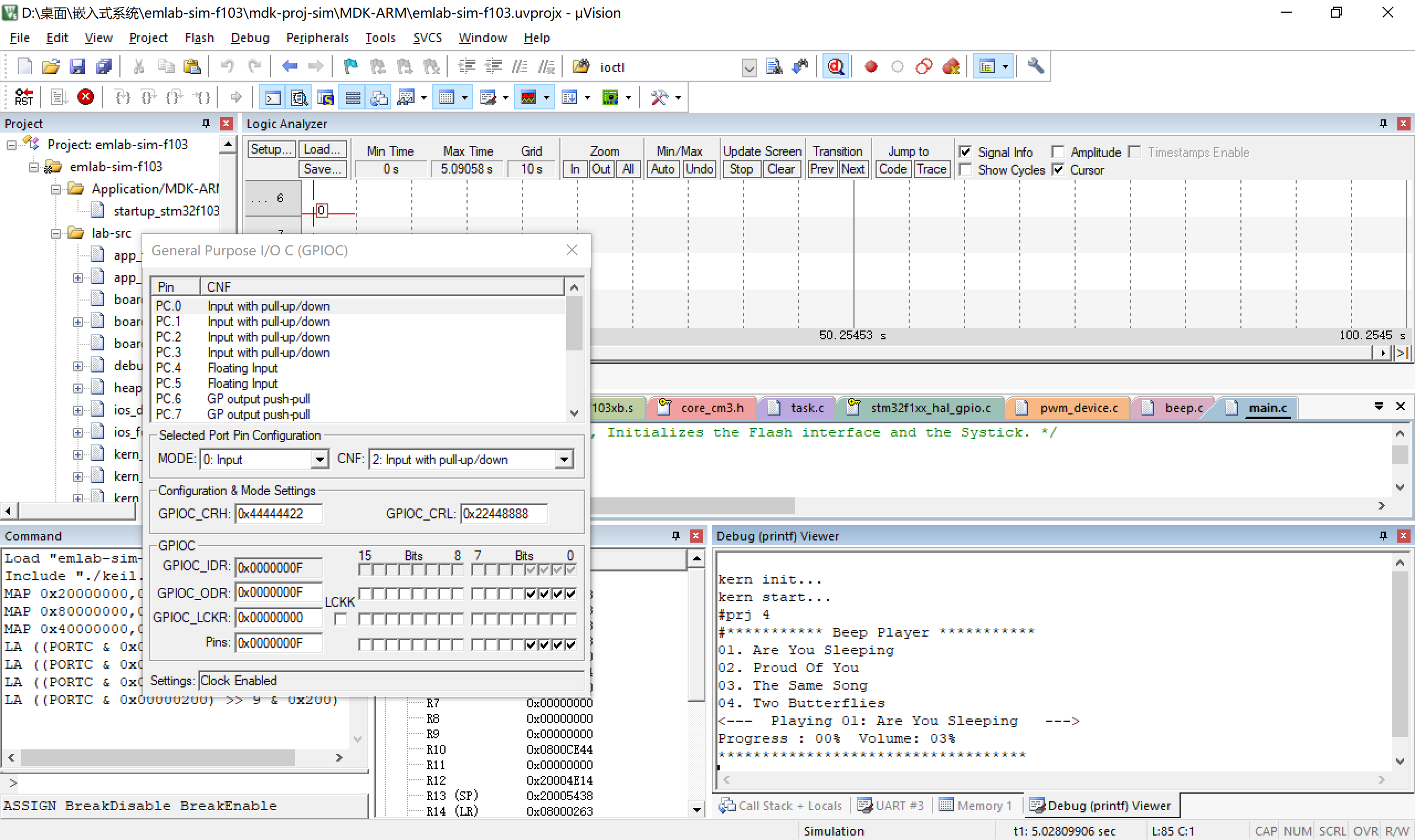






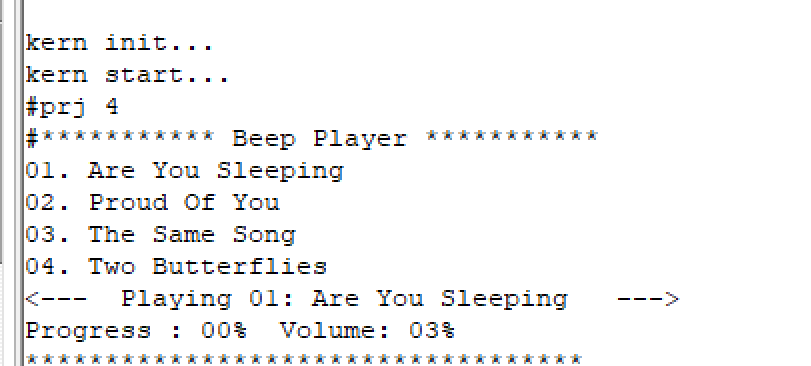
# 实验结果与分析

在调试模式下，打开debugview，输入prj 4，运行：

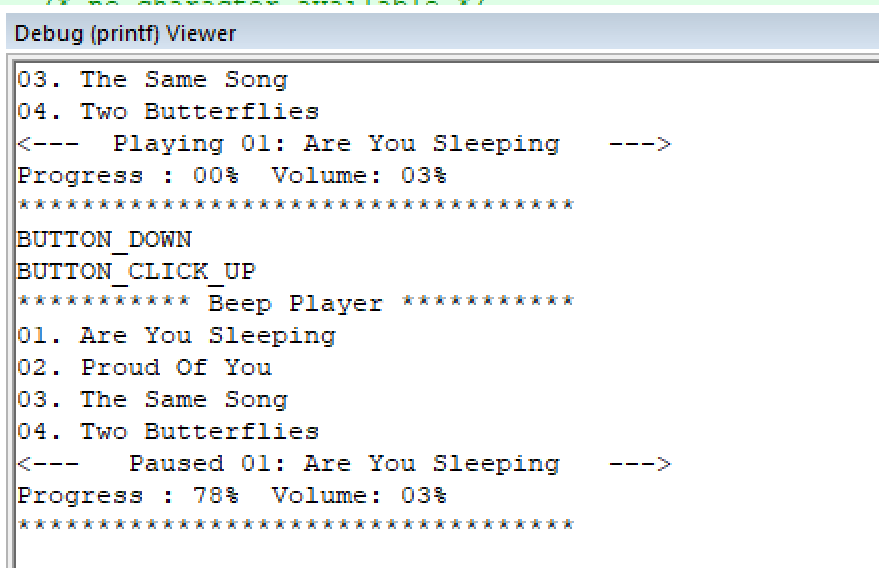


首先验证k3短按的暂停播放效果：

起初进度如下：

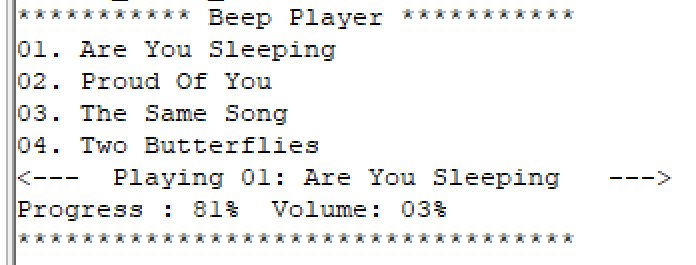


K3短按一次后：

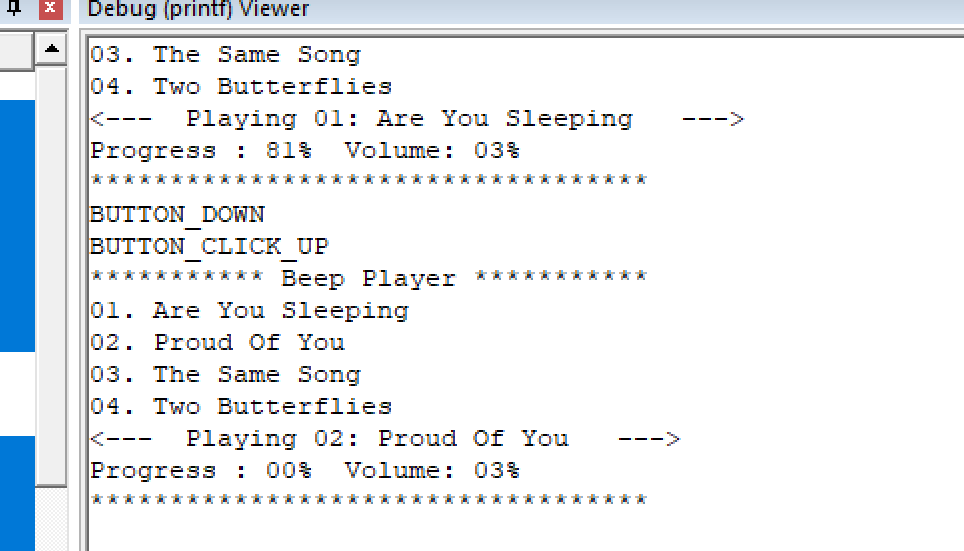


然后看K4短按的下一曲功能：

当前为第一首：

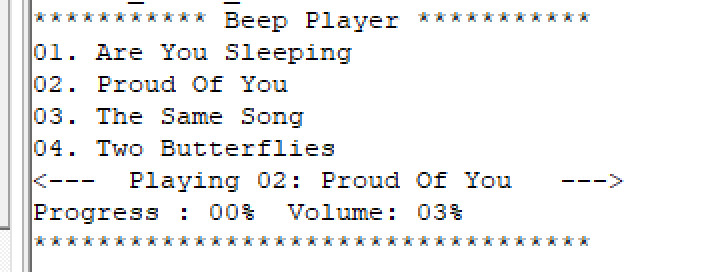


可以看到一次短按后实现了下一曲的功能：

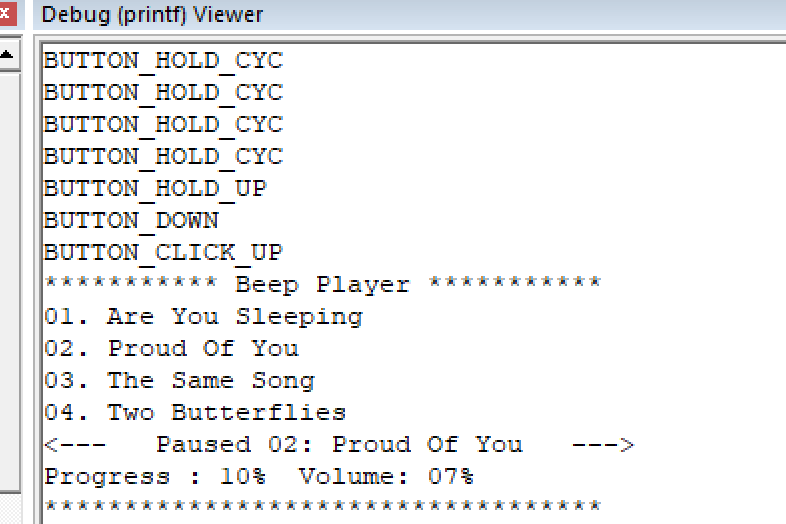


再看长按K4的加音量功能：

当前为%3：

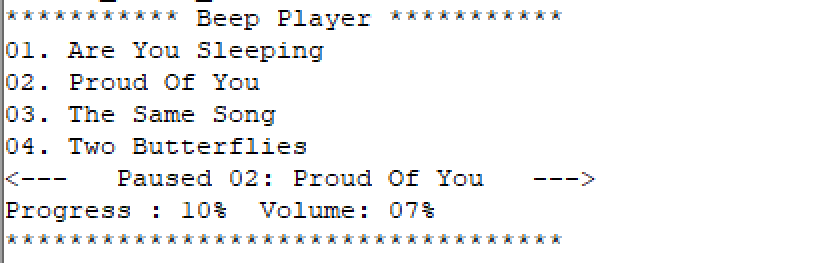


长按一次K4后，可以看到音量增加：

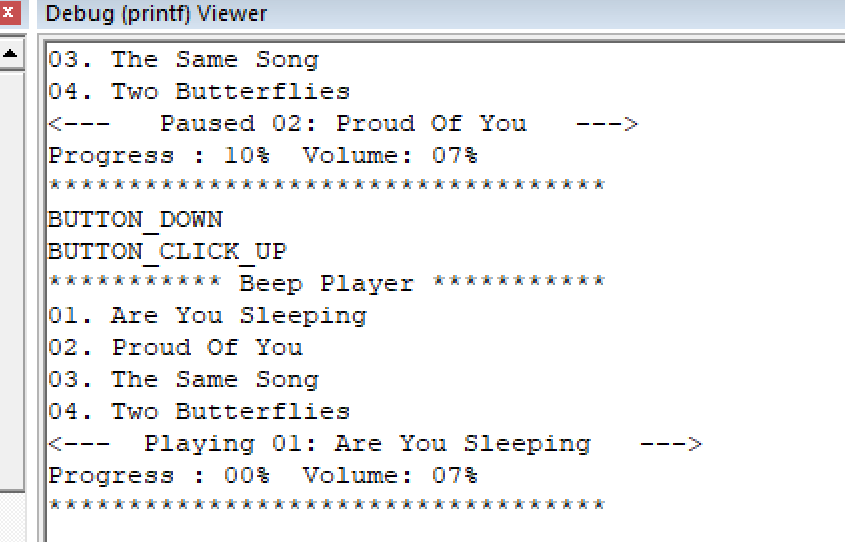


再看短按K5上一曲功能：

当前为第二首：

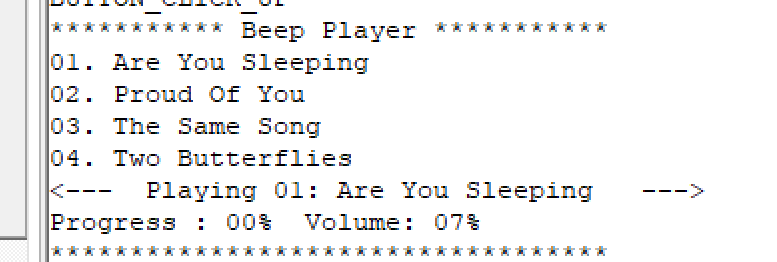


短按K5后，成功跳到了上一曲：

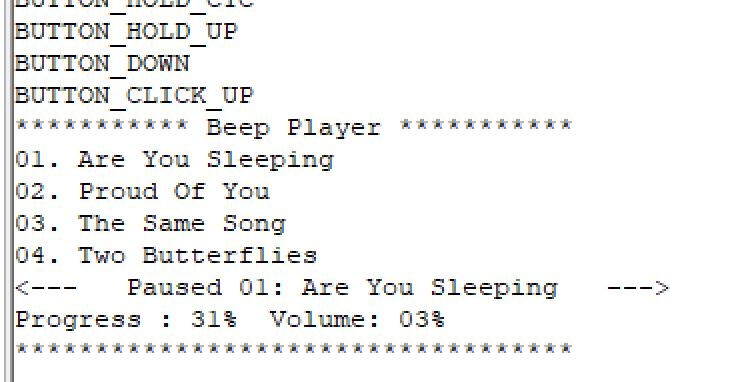


最后看一下长按K5的减音量功能：

当前为%7：



长按K5后，可以看到音量减小：



问题回答：

1.请描述整个应用程序的结构。包括哪些模块，彼此之间的关系如何。

播放器由 player.c 控制，初始化时启动一个任务 player\_entry 控制播放

器的暂停与播放。

控制命令来自 key.c ，初始化是启动了一个 20ms 周 期 的 定 时 器

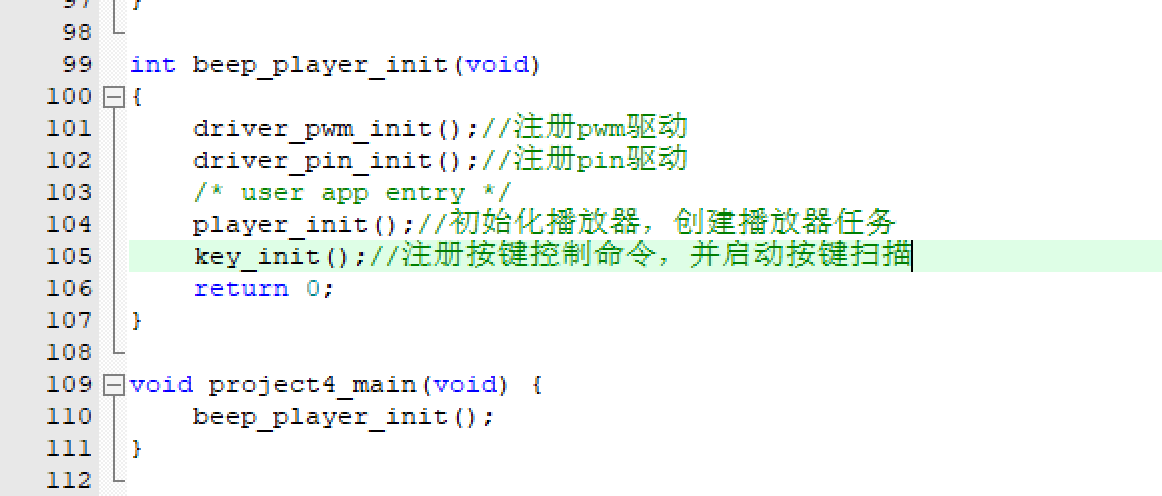
button\_manage.timer，定期扫描按键的值。

音乐数据来自 song\_data.h，调用 decode.c 解码后，调用 beep.c 改变

TIM10 的 PWM 输出，控制蜂鸣器的声音。

2.整个应用程序创建了几个任务，都完成什么功能？

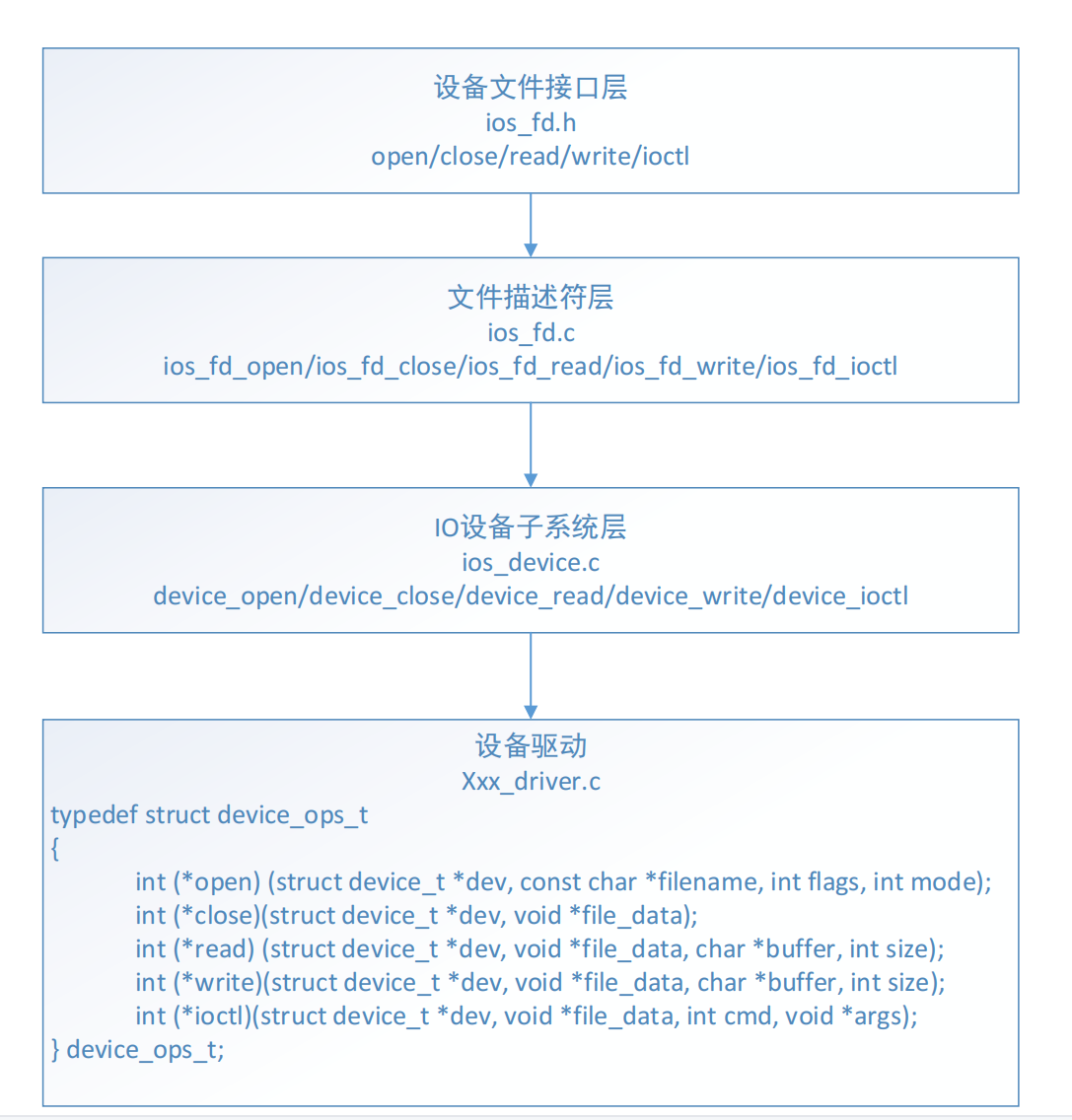
在project4\_main函数中可以看到本程序主要创建了player和key两大任务，分别用于管理播放器和扫描获取按键状态两大任务



3.应用层的 open/read/write 文件操作是如何映射到底层的 GPIO 和 PWM 接

口寄存器操作的？

基于如下模型：



1) 最顶层是设备文件接口层，提供类 unix 系统的 open/close/read/write/ioctl

文件操作函数（posix 标准）。但不支持挂载文件系统等操作。

2) 往下是文件描述符层。Posix文件接口使用文件描述符的概念表示一个

打开的文件。顶层打开一个文件时，是这样的使用方式：int fd = open(filename\_string, flag, mode)，使用文件字符串名字调用 open，返回

一个 int 类型的文件描述符 fd，后续的 read/write/ioctl/close 操作都是基

于这个 fd 操作。这个 fd 就是本层的文件描述符表 fd\_table[] 的索引号，

表中每一项都是 fd\_t 类型的文件描述符结构体，存储了文件的设备、名

称、私有数据等信息。

3) 再往下是实际完成工作的 IO 设备子系统层。该层最重要的功能是绑定

设备与驱动。

IO 设备子系统层管理者一张设备链表，device\_list，其中每一项的类型是

device\_t，对应一个特定的设备驱动。一个设备驱动对应一类设备（例如，gpio

的驱动是一个设备驱动，每一个 gpio 管脚可以看做一个设备；pwm 驱动是一个 设备驱动，每一个具体的定时器 pwm 输出可以看做一个设备）。一个设备驱动驱动管理所有同类型的的多个设备，这些设备可能只有参数不同，例如端口号、IO地址、中断号等。

一般在系统初始化时注册设备驱动，在应用程序使用时，打开具体的设备，

调用设备驱动进行操作。

4) 设备驱动根据设备模型定义，创建出具备硬件访问能力的设备实例，将

该设备通过 device\_register() 接口注册到 I/O 设备子系统中。

应用程序通过 open-> ios\_fd\_open ->device\_open->device\_find() 接口查找到

设备（使用文件名），然后使用 I/O 设备子系统接口来访问硬件。

# 实验总结

通过这最后一个实验，我学习了编写设备驱动的方式。此次实验编写代码难度较于前几次较大，尤其要理解整个流程是什么样的，需要调用哪些函数，总体而言，收获较大。