**《固件库的GPIO输出》实验报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**： **嵌入式系统设计** | **年级**：23级 | **上机实践成绩**： |
| **指导教师**：郭建 | **姓名**： **张建夫** |  |
| **上机实践名称**： **固件库的GPIO输出** | **学号：10235101477** | **上机实践日期**：2025/03/25 |
| **上机实践编号**： | **组号**： | **上机实践时间**： 14：50~16：30 |

1. **目的与要求**
2. 了解固件库控制LED的方式；
3. 学会使用固件库点亮LED灯；
4. **内容与设计思想**
5. 实验原理：

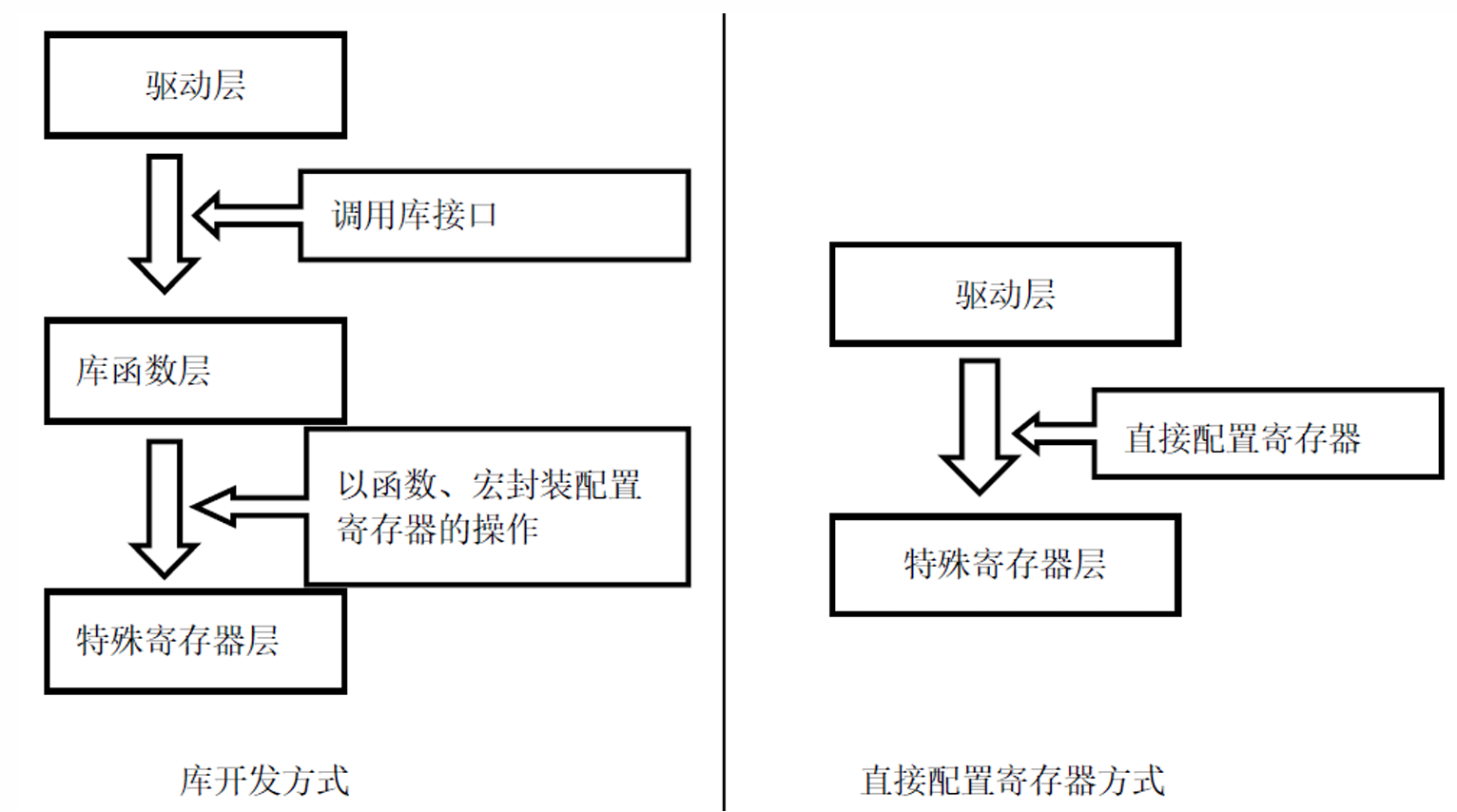
在上一个实验中，我们使用寄存器直接操纵引脚接口，需要照着硬件手册寄存器的说明来编写代码，很不方便，此时由于ST公司对STM32提供了标准函数接口，可以直接调用来实现引脚控制，即使用软件库。

软件库是寄存器与用户驱动层之间的代码：

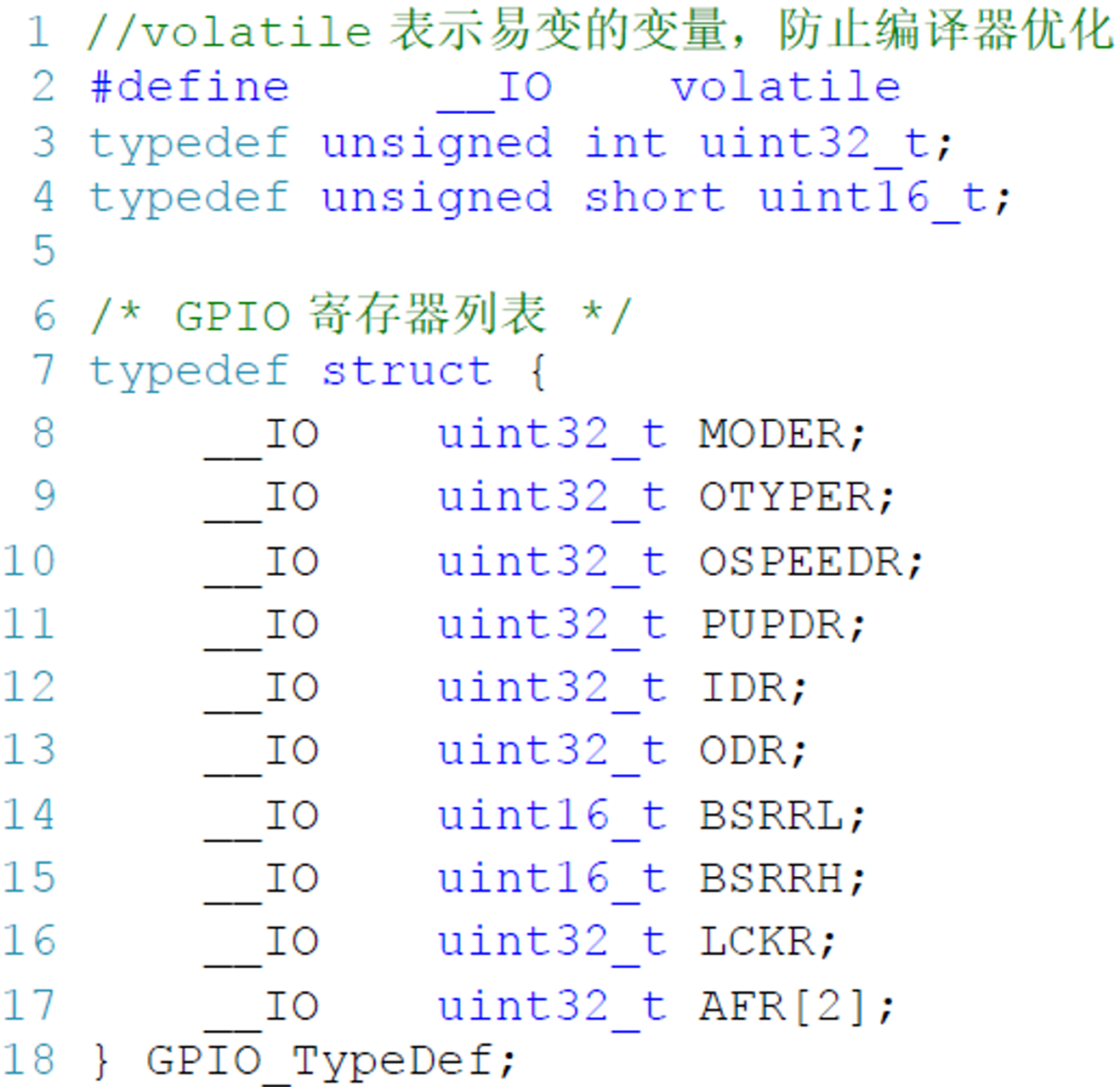
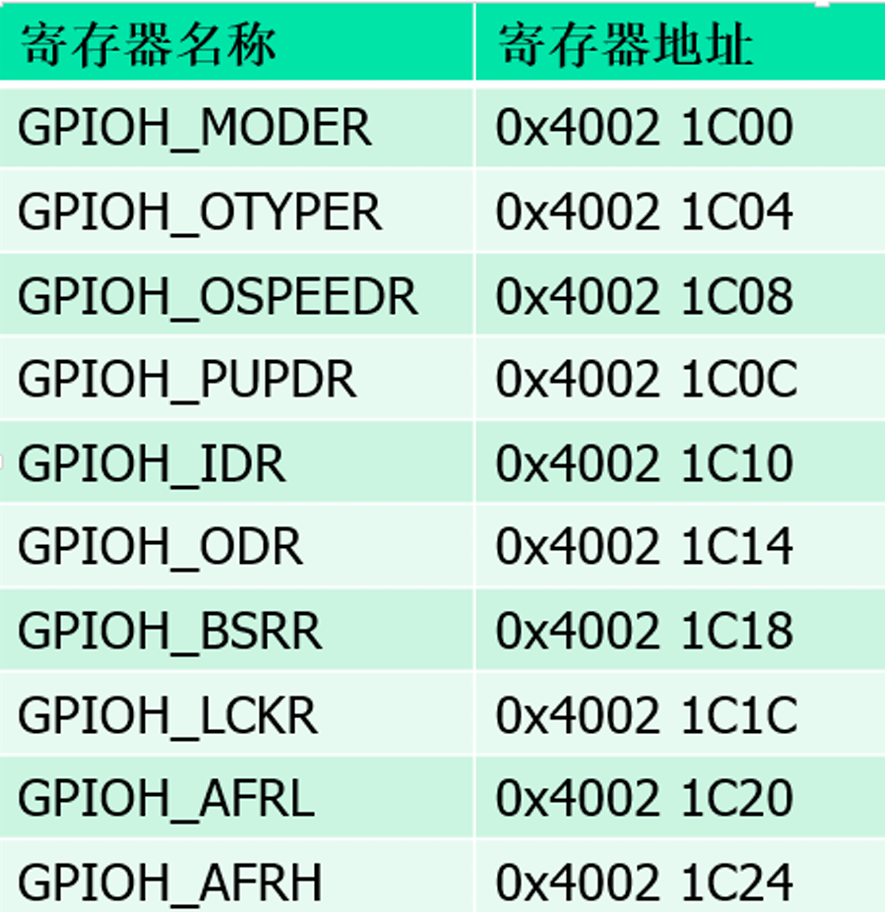
1. 向下处理与寄存器直接相关的配置

2. 向上为用户提供配置寄存器的接口

如图所示：

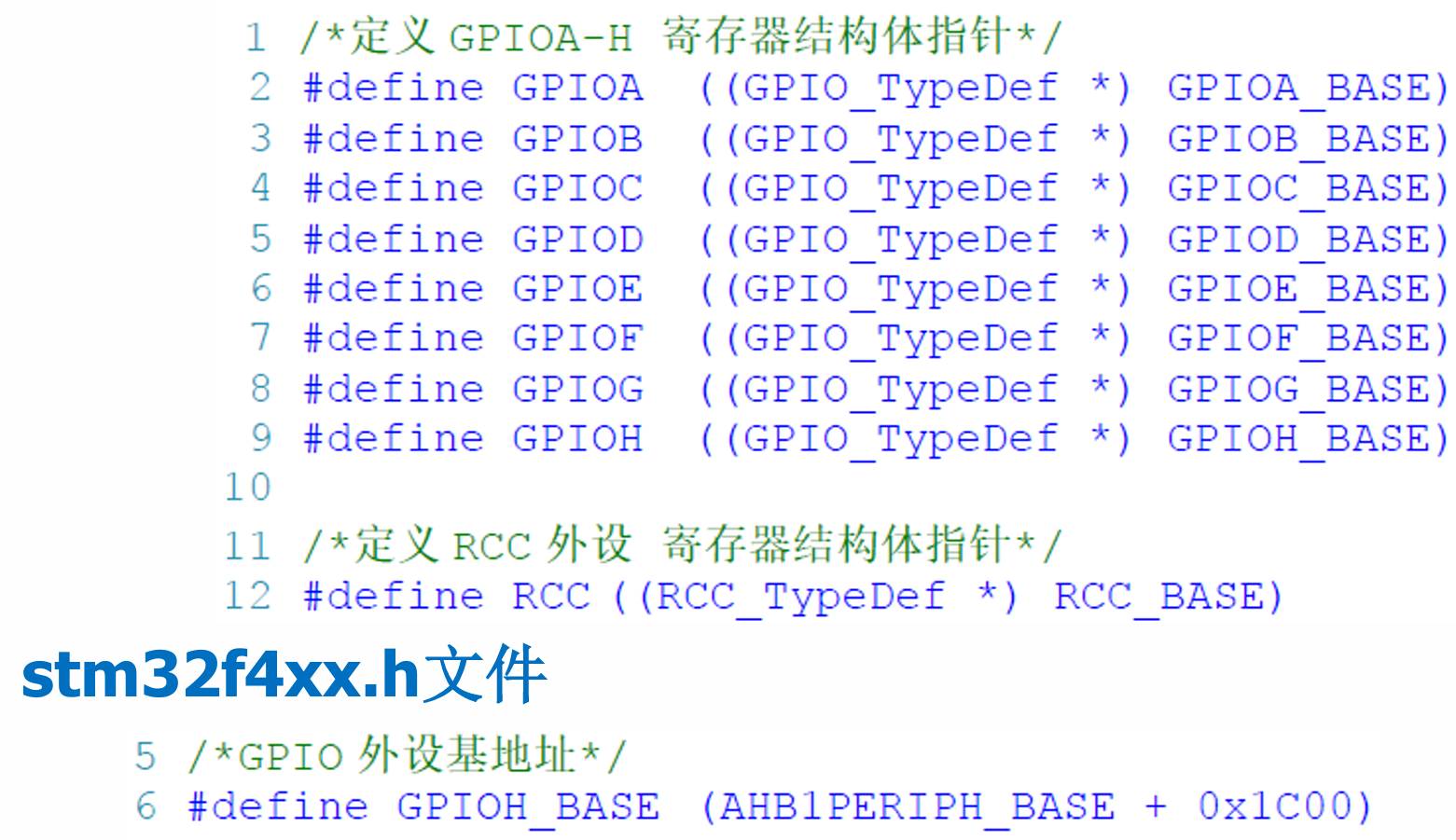


对于构建库函数，首先要写出使用寄存器点亮LED的代码，然后对该代码中的变量逐层封装，以实现库的模型，同时也要对寄存器的地址进行封装，使用寄存器地址的连续递增特性可以使用结构体进行封装，如下图所示：



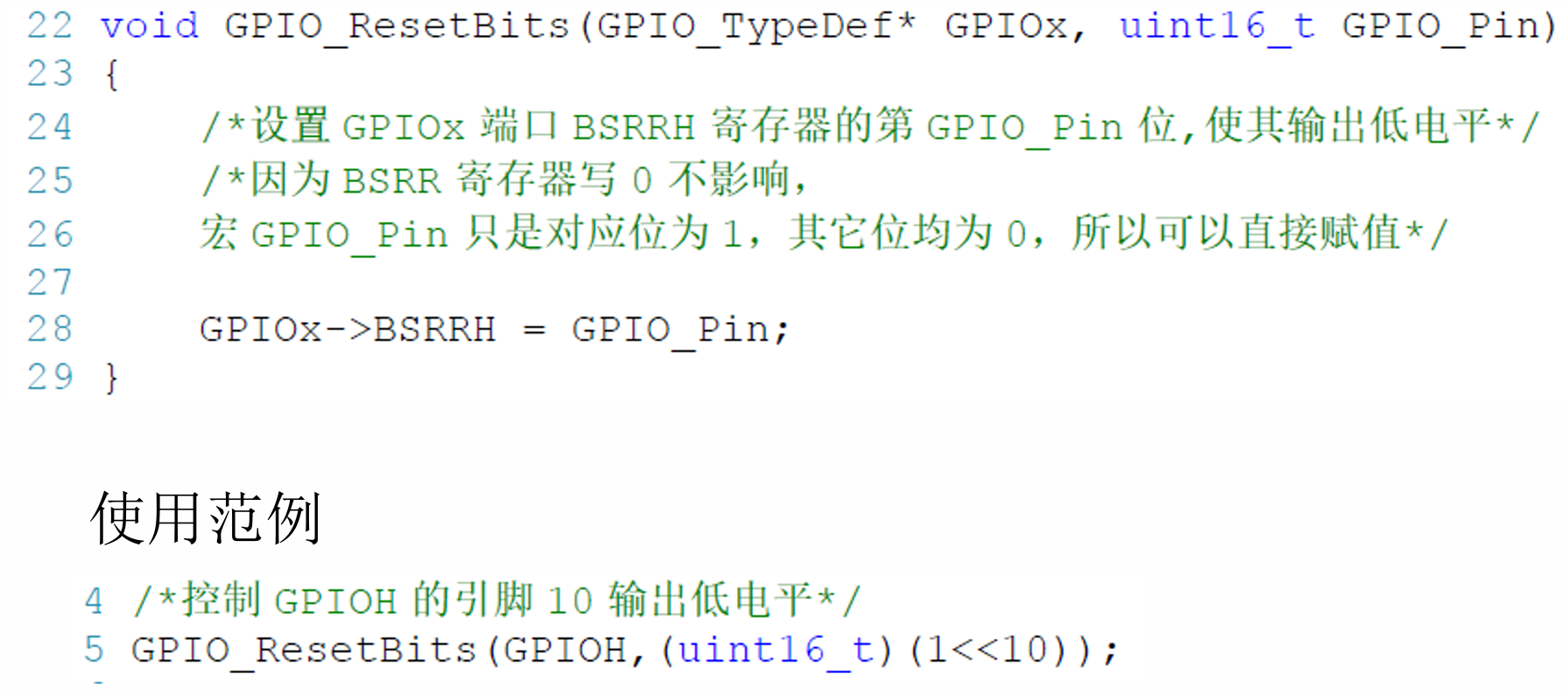
（左图为具体的例子，右图为封装的实现）

接着，需要定义结构体指针进行进一步封装所有寄存器以及外设的基地址：

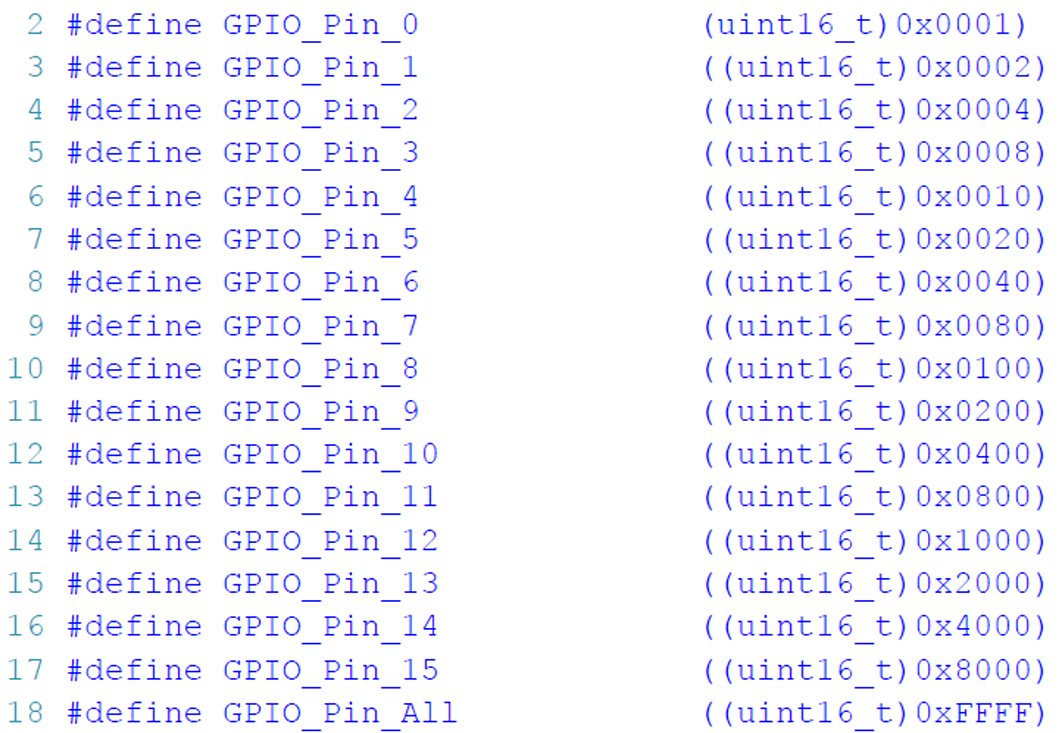


将这些基地址强制转化为GPIO\_TypeDef 类型的地址即可通过结构体指针访问，下面是一个使用结构体封装后便于编码的例子：

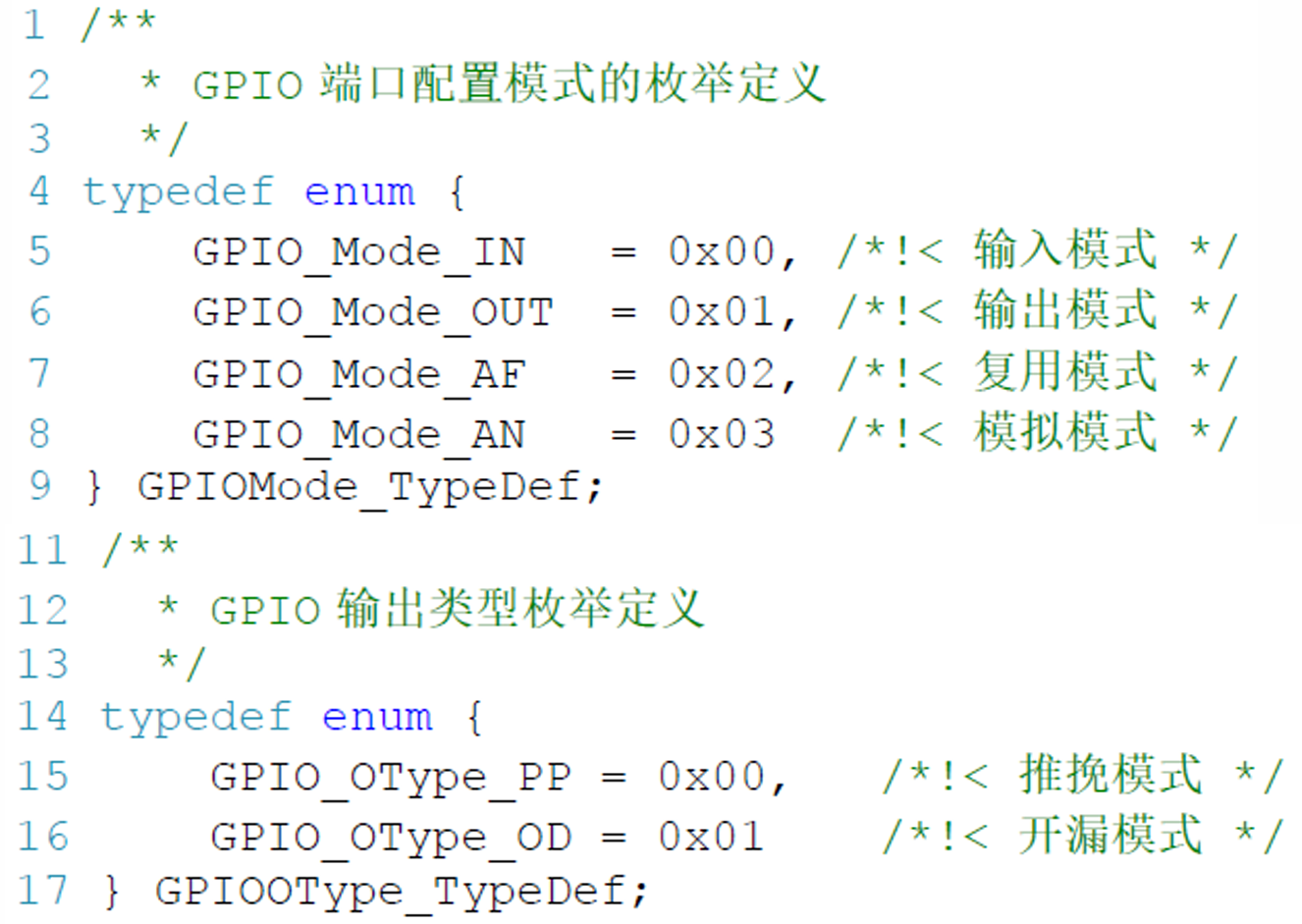
对于设置GPIOx端口的GPIO\_PIN位：



接下来具体封装引脚：



最后封装初始化结构体GPIO\_InitTypeDef以及引脚模式的枚举类型（枚举类型过多，不再全部展示，都是从ppt里面总结的）：



1. CMSIS标准及库层次关系与STM32固件库：
2. 基于Cortex 系列芯片采用的内核都是相同的 ，区别主要为核外的片上外设的差异
3. 这些差异却导致软件在同内核，不同外设的 芯片上移植困难
4. 3 ARM与芯片厂商建立了CMSIS 标准(Cortex MicroController Software Interface Standard)
5. CMSIS 标准，新建了一个软件抽象层
6. **使用环境**

调用dxdiag工具：

Operating System: Windows 11 家庭中文版 64-bit (10.0, Build 22621) (22621.ni\_release.220506-1250)

Language: Chinese (Simplified) (Regional Setting: Chinese (Simplified))

System Manufacturer: HP

System Model: HP Pavilion Aero Laptop 13-be2xxx

BIOS: F.13 (type: UEFI)

Processor: AMD Ryzen 5 7535U with Radeon Graphics (12 CPUs), ~2.9GHz

Memory: 16384MB RAM

Available OS Memory: 15574MB RAM

Page File: 27604MB used, 5685MB available

Windows Dir: C:\WINDOWS

DirectX Version: DirectX 12

DX Setup Parameters: Not found

User DPI Setting: 144 DPI (150 percent)

System DPI Setting: 192 DPI (200 percent)

DWM DPI Scaling: UnKnown

Miracast: Available, with HDCP

Microsoft Graphics Hybrid: Not Supported

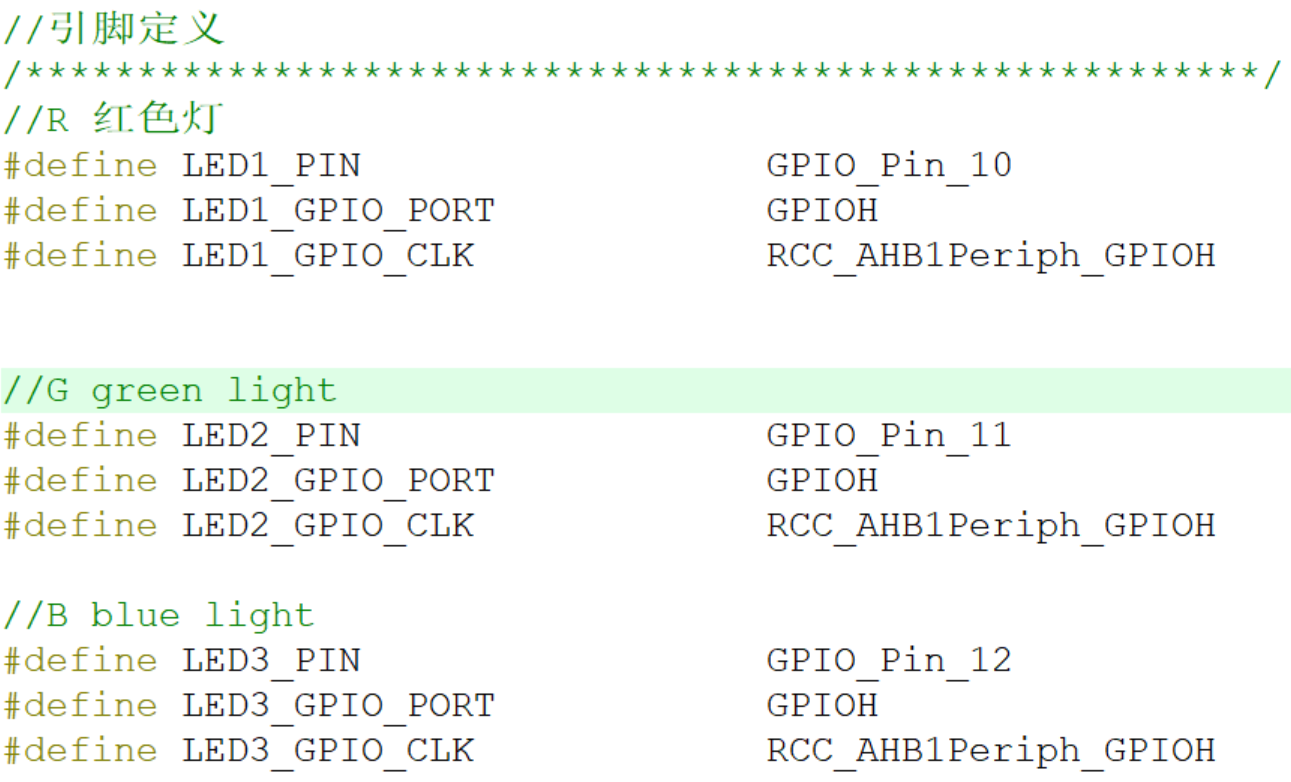
1. **主要实验内容和结果展示**
2. **完成示例实验**

示例实验为红灯闪烁，将提供的模板进行编译，并烧录到开发板上即可，效果如图：

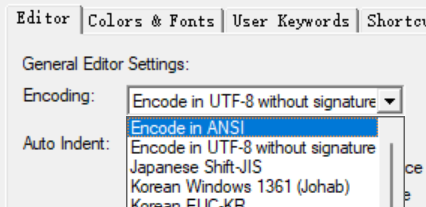


**2.** **完善bsp\_led.h和bsp\_led.c文件的内容，使得与硬件相关的都写 到库文件中，然后重写main.c文件，完成“红-绿-蓝”滚动点亮~**

和两周前的实验一样，我们通过PH10~PH12访问三色灯，根据提供的模板，为了控制三色灯，我们要在bsp\_led.h封装好对三个灯引脚的访问，个人修改代码如下所示:

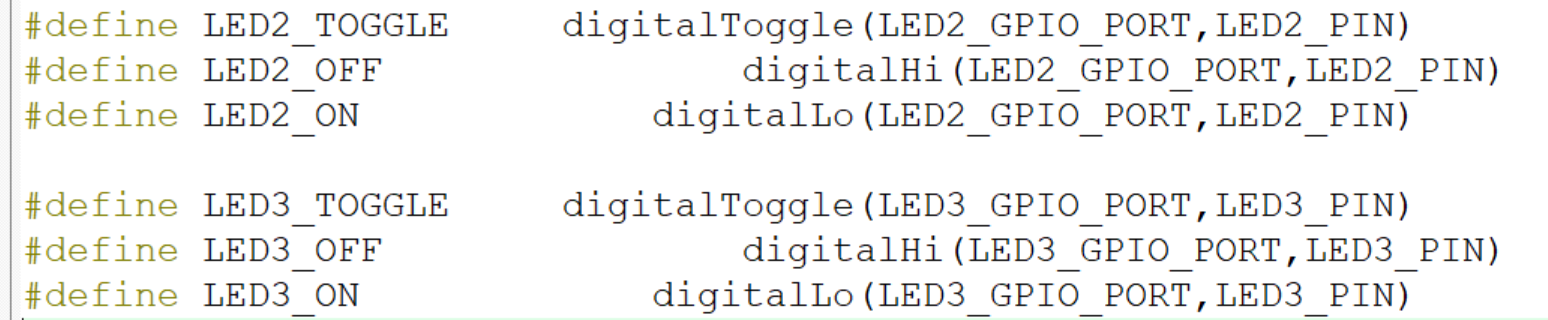


这三个灯红，绿，蓝，在固件库中表示为GPIO\_PIN\_10， GPIO\_PIN\_11，GPIO\_PIN\_12（之前第二部分封装引脚部分封装过的），此处本人在写注释的过程中发现正常打字输出是乱码，便使用英文写注释，后面查了相关原因，发现keil5的默认编码为ANSI，而打字出来的编码为utf-8，因而会出现乱码：



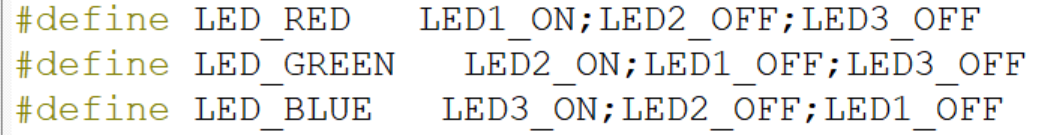
此时改成utf-8即可使打字正常，不过原本的注释就变成乱码了😓，所以还是写英文注释吧。

另外需要改的两个地方为控制io的宏，需要增加绿灯和蓝灯的控制宏io（点亮，关闭和翻转）：



该处控制io的宏的编写完全仿照LED1（红灯）的即可（即将所有LED1改为对应的LED编码）

最后编写各自点亮的函数宏：



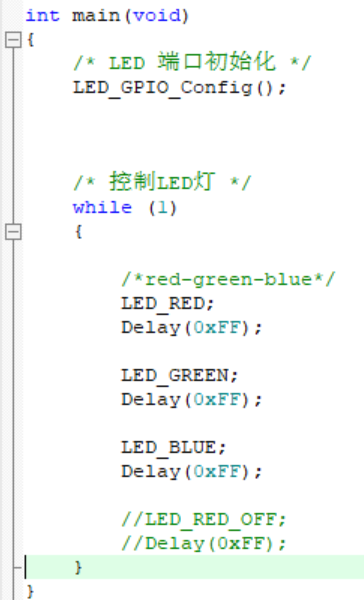
Led+颜色表示此时只有对应的颜色点亮（其他的灯应该灭掉，即宏转化后的LED\_ON和LED\_OFF）

此外，还需要在bsp\_led.c中对新增加的两个灯进行初始化（即设置结构体GPIO\_InitTypeDef的各个字段）：



在这个LED\_GPIO\_Config函数中，除了端口，其他的设置直接复用红灯的设置即可，因此在调用库函数GPIO\_Init时只有第一个端口形参改变，其他沿用红灯设置。（此处的改变使用的ppt的改变，我认为这样改变并不好，虽然说这三个灯的初始化共用了一个结构体，减小了内存开销，对于代码的阅读却是糟糕的，这三个灯的初始化设置并不明确，应将这三个部分分开初始化（即调用三次初始化函数），这样虽然时间久了，单个的初始化逻辑会更清晰）

最后，更改main.c使代码主逻辑变为三个灯轮流亮：



此处没有完全根据ppt的进行改动，ppt的delay时间太长，便将等待时间从0xffffff改为0xff。

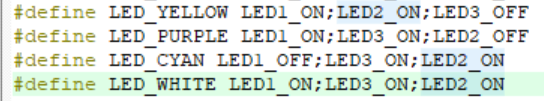
效果展示：



3. 修改代码使得实验结果的现象为：“红-绿-蓝-黄-紫-青-白 ”滚动点亮~

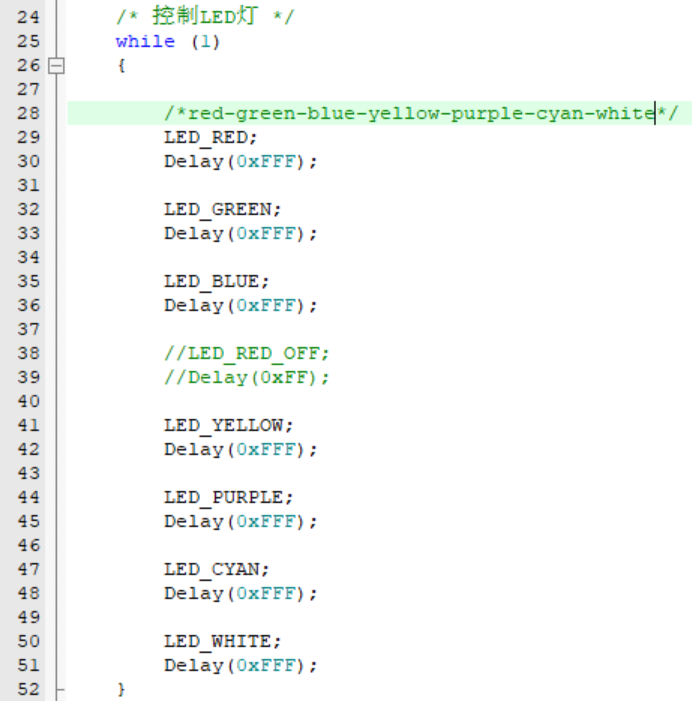
在第二个实验的红绿蓝交替点亮中，我们已经掌握了如何点亮一个led灯，而由于所有颜色都可以由三原色（红绿蓝）根据不同比例得到，因此，对于一个任意颜色的灯，都可以调节不同配色灯的程度得到，但是，由于我们目前只有纯色的红绿蓝，无法调节三个灯的呈现比例（不然就可以做呼吸灯了），只能得到如下的颜色：黄色（红+绿，1：1比例），紫色（红+蓝，1：1比例），青色（蓝+绿，1：1比例），白色（红+绿+蓝，1：1：1比例）

我们发现这些颜色正好是等比例可以得到的，因此，对于每种颜色，我们只要控制让特定的灯（如红和绿同时亮起即可实现黄灯）亮起来即可，且不用再添加其他的灯，也就意味着我们不需要初始化新的灯，因此，我们只需要在bsp\_led.h中添加新的颜色的灯：



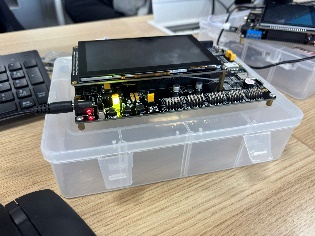
从上到下依次为黄，紫，青，白。

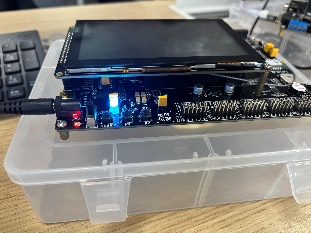
同时修改main.c,使主逻辑变为七种颜色不断循环展现：



此处将延时稍微延长了一些，不然不好拍照，前面调的0xff有点快了。

效果展示：





这里白色在照片里显示的像蓝色，找了很多角度没有能完全体现其白色的，只能找个尽可能像白色的角度，和教室上方的灯对比一下。

**五、实验总结**

该实验的根本原理和上个实验并无不同，唯一改变的是编程方式和对固件库的使用，此外实验模板以红灯为例子，打通了整个通过固件库操纵引脚的方式，对于其他引脚（本实验中只有绿灯和蓝灯引脚），也可以通过相同的方式操纵，而且不需要像上一个实验那样要了解具体寄存器的地址，直接通过名字就可以判断，这为代码的编写（实现了软编码）带来了极大的便利。

除去该实验相比于上次实验的带来的访问引脚的便利性，我还学到了keil自身的一些特性（例如默认使用ANSI编码，不用utf编码，真是个怪胎。。。），让我对keil的使用进一步熟练。