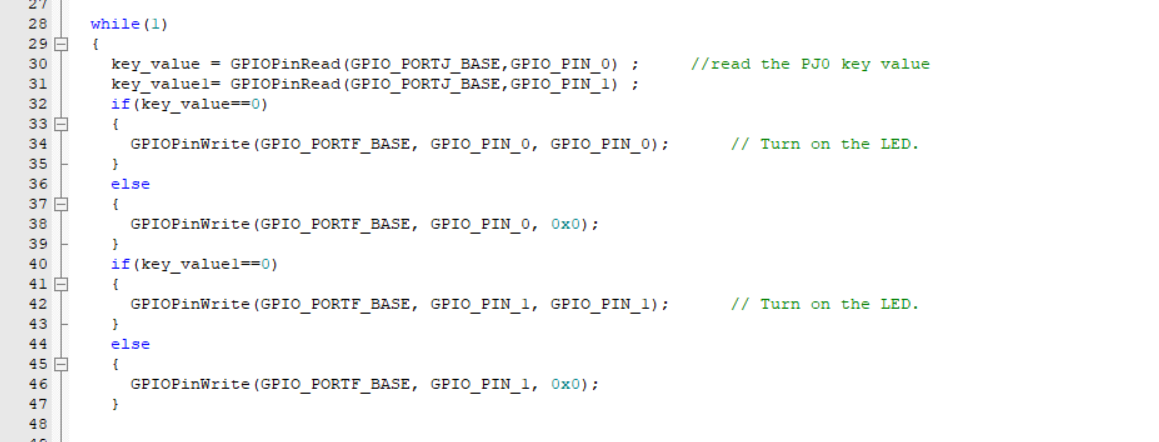
**ARM实验一 时钟选择与GPIO实验**

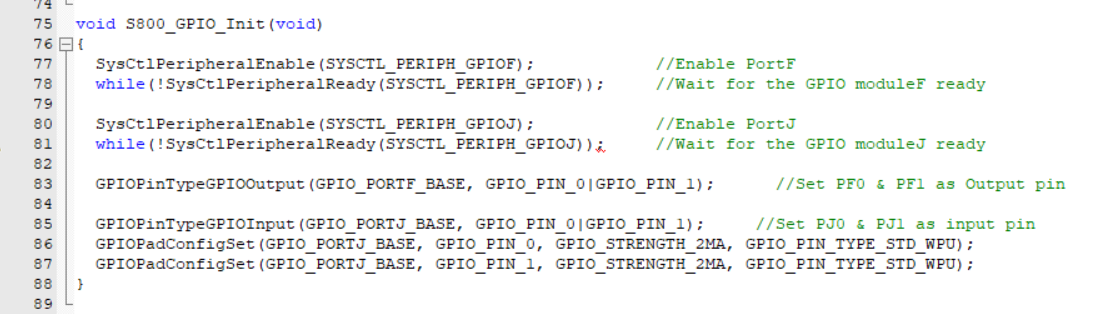
1. **实验主要程序段展示与分析**

以下展示各分项实验的关键代码，具体代码详见附加的c文件压缩包。

实验1.2：

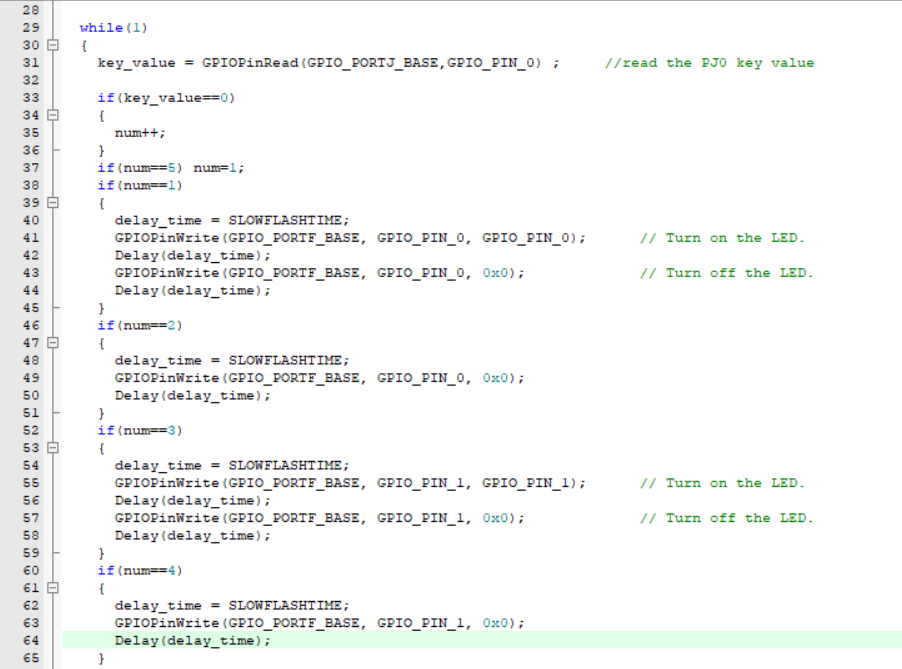


在while循环中，分别读取两个按键的电平信息，分别存入key\_value、key\_value1内。之后分别根据两者的值对相对应的led发出高低电平命令。采用if-else语句实现分类判断。由于while循环中语句简单，执行速度很快，所以可以认为是同时判断了两按键的电平高低并输出了指令信号。



83行添加了对PF1的设置。85行添加了对PJ1的设置。除了端口有变化外其他设置与PF0和PJ0相同。新增加一行87行，用来设置PJ1按键属性。

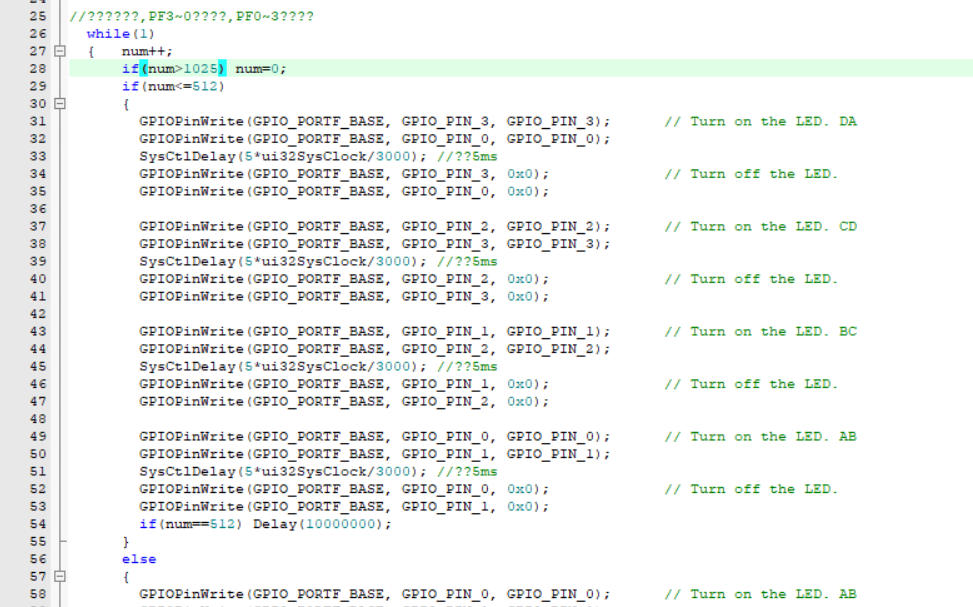
实验1.3：

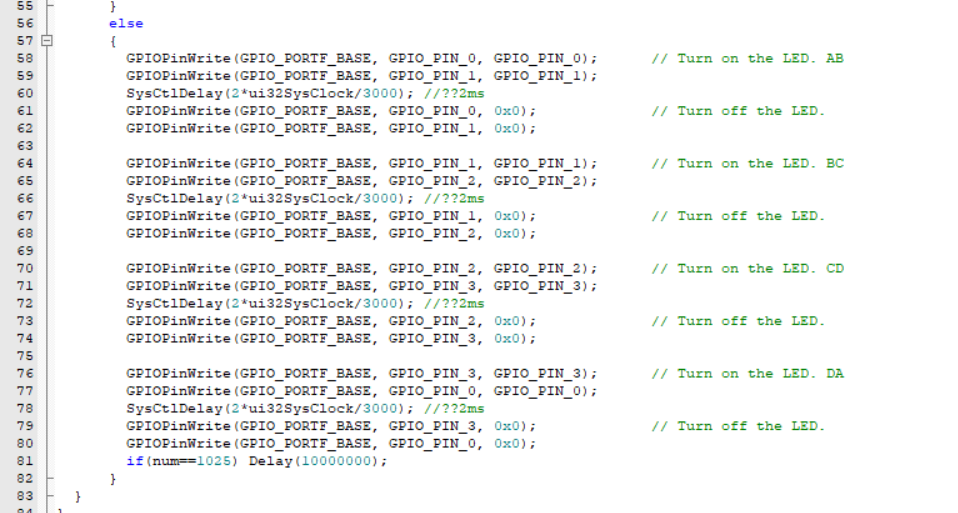


先设置一个全局变量num，记录按键次数。在while循环中，先读取按键电平，若按下则为低电平，num加1。当num为5时回到1，实现同余4的效果。之后分四类情况讨论。50行和64行分别设置了delay，目的是创造延时以避免因为while循环一次执行太快导致一次按键经历了多次while循环，从而产生的多次读取问题。

程序的缺点在于，若程序恰好在delay中，按下按键将无响应。经过尝试，这个问题可以通过改写delay函数获得部分改善，即delay循环到一定次数后开始检测按键的电平，若低电平则用break跳出delay。当然，由于delay的for循环内加入了语句，使循环一次的时间变长，因此要适当调小输入delay的数值来获得相近的延时时间。但该法不能彻底解决问题。我认为该问题的完全解决需要用到后面实验的中断。

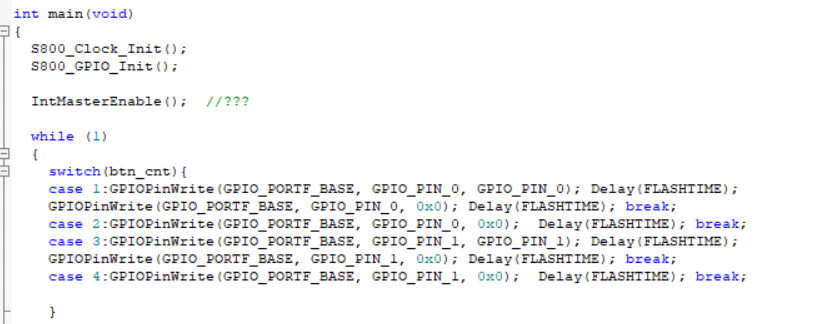
实验1.4：





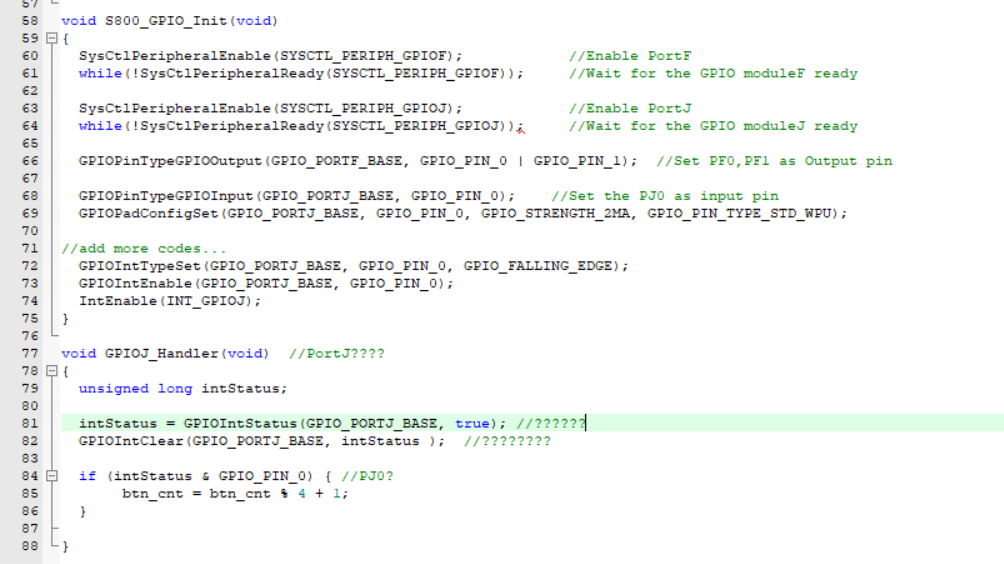
这里采用了双四拍的方法。已经知道电机需要512个循环周期转动一周，因此对num做以下分段：0-511顺时针转动；512为停止（约三秒）；513-1024逆时针转动；1025为停止（约3秒）；之后num回归0。这里通过调整逆时针的延时时长加快了逆时针转速。顺时针和逆时针是通过双四拍顺序的颠倒实现的。

实验1.5：



主函数部分与实验1.3大同小异。区别仅仅在于用switch替换了if。按键次数的增加依赖于中断。这里还是靠delay函数计时。



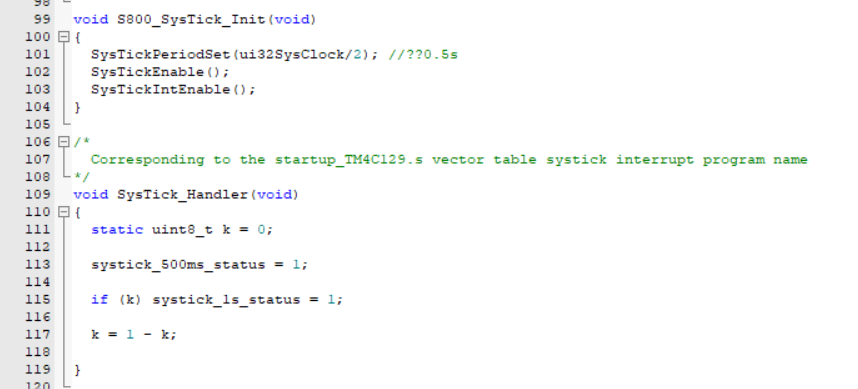


中断方面。首先，需要在主循环前对内核级进行使能操作，即第30行。之后，需要在常规的端口使能函数后添加对按键的中断类型和触发方式设置和使能函数，即72-74行。这里是对源级和NVIC级进行使能操作。

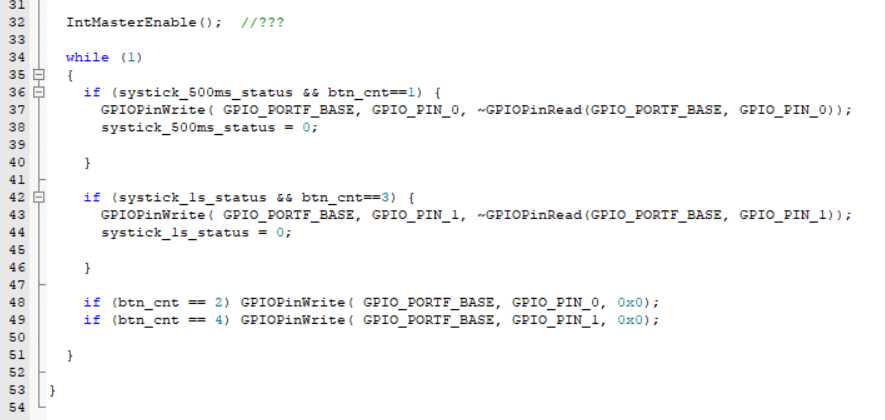
然后是重写中断服务程序。首先读取中断状态确定中断源，接着清除中断请求。最后对按键次数进行操作。

程序的缺点是没有对按键做去抖操作。按下按键有时会发生抖动，这样会导致一次按键多次中断。但是可以通过“短按”避免大部分抖动的出现。在软件上我尝试了一个方法：借助下一个实验采用的systick计时中断，即：每次按键后立即取消GPIOJ中断的使能，并打开systick计时中断使能。当systick中断时打开GPIOJ中断使能并关闭systick使能。然而该法没有起到作用。我认为原因是取消GPIOJ中断使能的语句位于GPIOJ中断服务程序的后半部分，时序上落后于抖动的下降沿。

实验1.6：



先看systick的初始化和中断服务程序的重写。初始化中，先设置好定时周期为0.5秒，这样一亮一灭周期恰好是1秒，然后使能中断。在中断服务程序中，可以看到每隔0.5秒系统就会输出systick\_500ms\_status置1的命令。接下来使用“软件定时”，利用局部静态变量k实现“分频”，将两个0.5秒计时周期叠加形成一个1秒计时，每隔1秒输出systick\_1s\_status置1的命令，这样就可以实现LED\_M1的2秒亮暗周期。



在while循环前照例写上内核级中断初始化。GPIOJ中断服务程序与实验1.5相同。按键数为2和4时写低电平是必然。当按键数是1时，结合systick中断服务程序分析时序可知，程序进入第一个if（36行），切换led的亮暗状态，之后迅速清除systick\_500ms\_status，之后的while循环虽然btn\_cnt为1，但由于systick\_500ms\_status是0，while循环将不做任何操作，直到systick在一个定时周期（0.5s）后重新把systick\_500ms\_status置1，while循环才会重新执行第一个if语句。btn\_cnt为3时同理。由此可见，程序巧妙地实现了每隔一个周期出现切换led亮暗状态的信号。

程序的缺点是，在从btn\_cnt为4回到1，以及从btn\_cnt为2到3时存在衔接问题。以btn\_cnt是4时为例，由于与btn\_cnt为1之间已经隔了三次按键的时差，systick\_500ms\_status一般已经是1，这时按下按键将进入第一个if，灯亮并将systick\_500ms\_status置0。然而，由于systick一直在后台运行，此时systick计时往往还剩不足0.5秒，导致不足0.5秒后systick\_500ms\_status又将变1导致灯灭，从而第一次灯亮时常会短于0.5秒。当然，之后一切正常。为了弥补该缺陷，我尝试在btn\_cnt为2和4时添加SysTickDisable()取消systick使能，在1和3时添加SysTickEnable()为systick使能，结果有了改善，但是实验现象偶尔有不稳定，原因有待研究。由于这项改进尚不成熟，故暂时没有采用该代码。

1. **讨论题**

1、人为修改内部时钟或外部时钟，如将内部时钟改为8M，或将外部时钟改为30M，会有什么结果？

答：程序运行会变慢；会变快。

2、在使用PIOSC及MOSC时，能否生成非晶振频率？如1M 或10M？

答：可以利用分频生成。

3、能否将PLL时钟调整到外部时钟的频率以下？如将25M外部时钟用PLL后调整为20M？

答：单纯地只使用PLL不能，只能配置为320M或480M。但是可以借助系统分频器，使用PLL分频降低频率。利用公式fsyclk = fvco / (PSYSDIV + 1)。SysCtlClockFreqSet()函数封装了上述过程。

4、在使用PLL时，系统频率最小值及最大值分别为多少？

答：最小为320M/1024=0.3125M；系统最大频率120M。

5、将PLL后的时钟调整为最大值120M，LED闪烁会有什么变化？为什么？

答：闪烁明显变快，若适当调小delay的循环次数，则肉眼难以分清闪烁，似乎在持续发光。

6、GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0)此函数中，每个函数项的意义。第三个函数项为GPIO\_PIN\_0，如果改为1或改为2，或其他值，分别有什么现象？

答：意义分别是端口基地址、引脚屏蔽位、写入的数据。改为1时，正常写入，灯发光；改为2或其他值时，写入被屏蔽，灯不发光。

7、结合硬件说明GPIOPadConfigSet行的作用。如果此行注释，在WATCH窗口中观察key\_value值会有什么变化。

答：配置引脚的驱动强度和类型。设置驱动强度和引脚类型。驱动强度是指电流强度，有些外设需要较强电流才能驱动。引脚类型如弱上拉，与电路设计有关。注释后由于缺少配置，按键输入无效，key\_value始终为0。