**试用期间知识点总结**

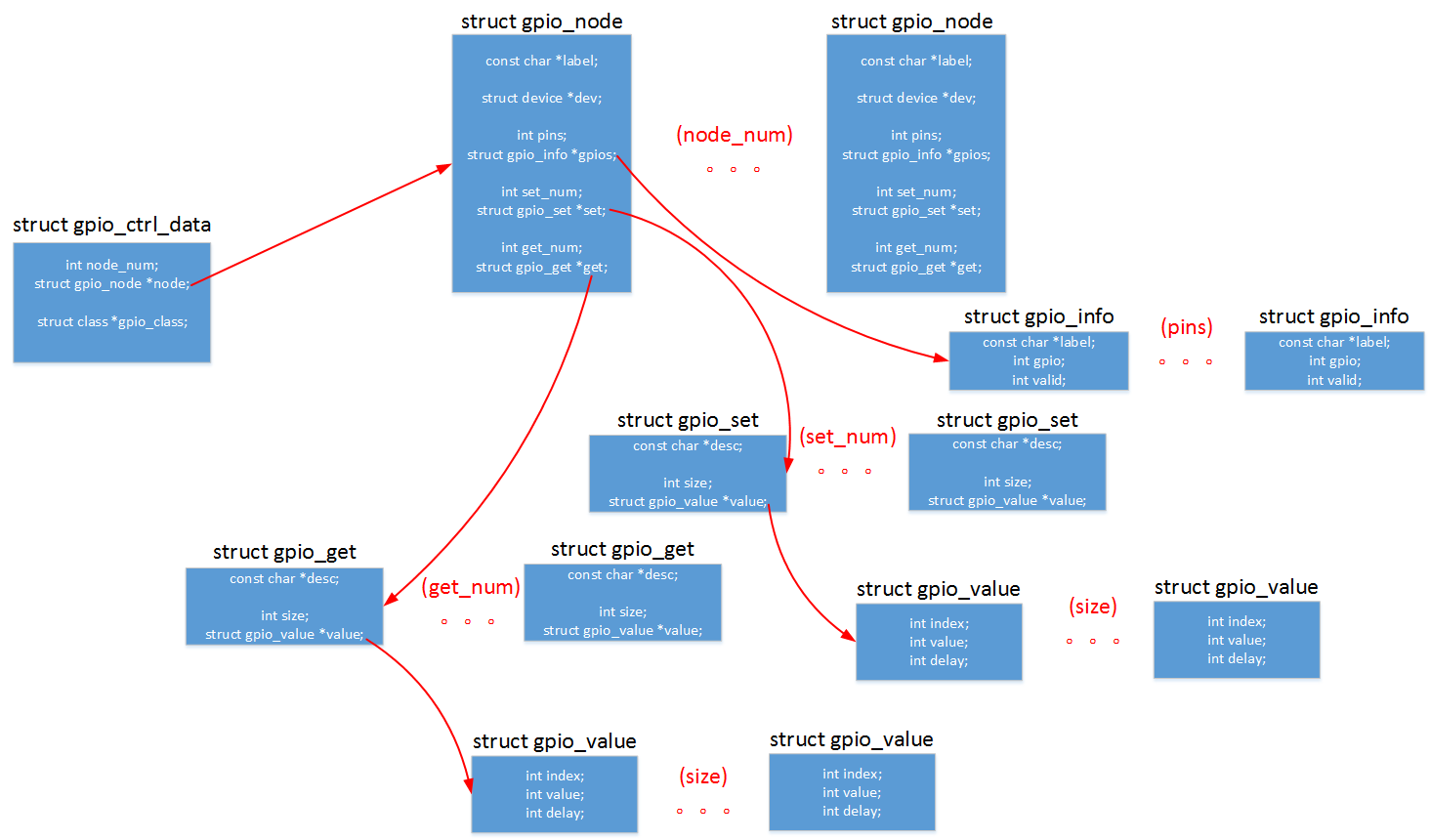
该文章用于记录本人在试用期间的学习过的一些知识总结，岗位为产品研发二部的嵌入式助理软件工程师，主要任务为负责公司产品内的Linux底层驱动代码的编写和调试。

很感谢各位同事的帮忙，才得以让我快速适应公司的工作，很感谢带领我入门嵌入式Linux驱动的江工和林工，感谢你们的耐心指导。

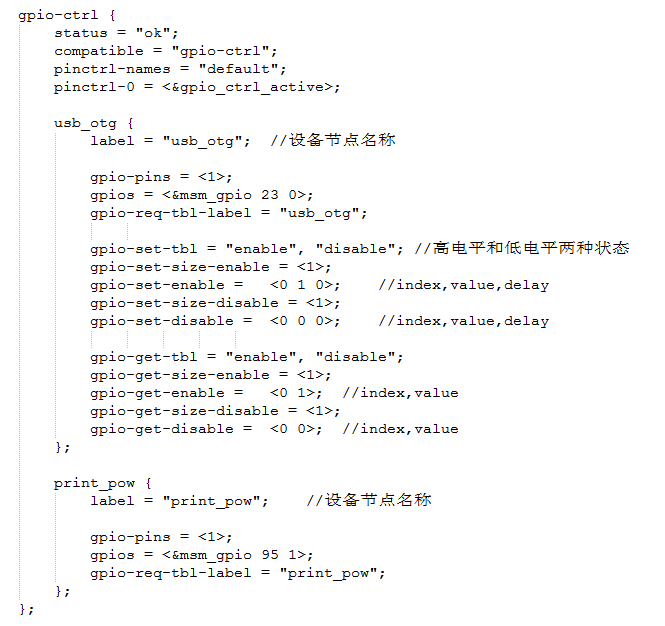
在7月份刚入职的时候，由江工进行指导并进行一些任务分配，在此期间，我学习和熟悉了公司中gitlab的使用和源码编译服务器的使用，Git是一个版本分布控制系统，gitlab用于托管公司源码，使得在产品开发中获得了很大的便利，另外，在此期间，我学习了如何对高通的安卓源码进行镜像编译和开发步骤，另外，还学习了，当镜像编译完成后，如何通过QFile软件或者fastboot命令进行镜像烧写等。

了解和学习了一些基本知识后，开始逐渐进入到Linux驱动的开发，在此学习期间，实验和学习的工具为公司内部产品TPS508，韦宝华出版的《Linux设备驱动开发详解：基于最新的Linux 4.0内核》这本书给予了我很大的帮助，引领了我打开嵌入式Linux的大门，在江工的指导下，我先开始学习Linux内核中的驱动模型框架，以及内核中的驱动模块化编程，Linux内核中的驱动模块编程和平时的应用程序编程不一样，它并没有main函数入口，而是使用了模块化，module\_init()为程序的入口，module\_exit()为程序的出口，当Linux内核加载时，会调用module\_init()函数进行驱动模块加载，当使用rmmod命令卸载模块，则会调用module\_exit()函数进行模块的卸载，在内核中，每个设备至少使用一个struct device结构体进行描述，驱动程序使用struct device\_driver结构体进行描述，此外，在驱动模块编程中需要注意device、bus和class的相关概念，这些内容非常重要，基本遍布在Linux内核中的所有设备驱动程序中。

对Linux内核设备驱动模型具有一定的理解后，我开始分析和学习江工给我发的驱动程序代码，一个gpio控制驱动程序，看完这个驱动程序后，我感触很大，下图是这个驱动的结构体封装：

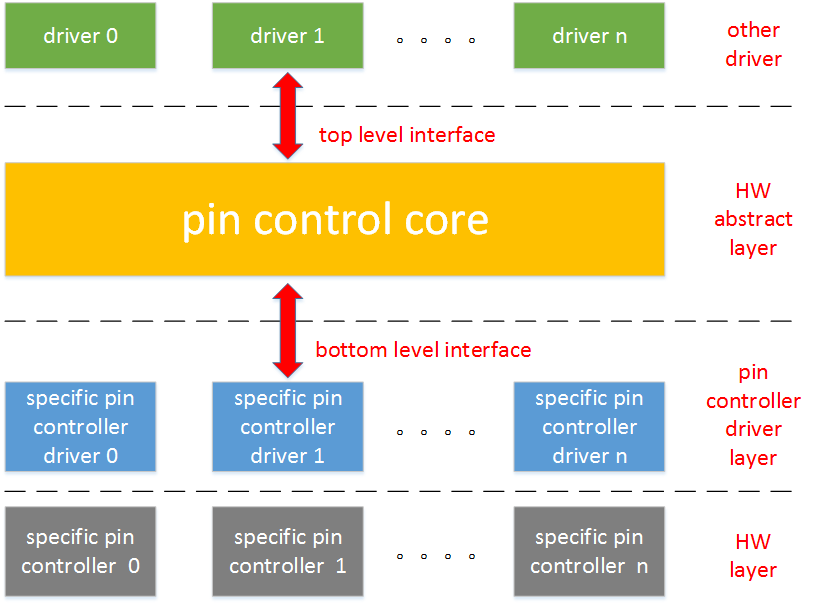


这个模块的驱动程序处理得非常好，兼容性和可移植性非常地好，能够支持多组以及多个gpio的控制，在内核中生成一个gpio-ctrl的类，在类下可生成多个gpio控制的设备节点，在设备节点中导出了ctrl属性文件，在应用层使用cat或echo命令就可以完成设备的操作，很方便，使用这个驱动程序，基本可以完全控制一个产品的所有GPIO（通用输入输出接口），使用的时候，只需要在板子对应得设备树文件中添加设备节点即可，类似下面的gpio-ctrl设备节点，如下图：

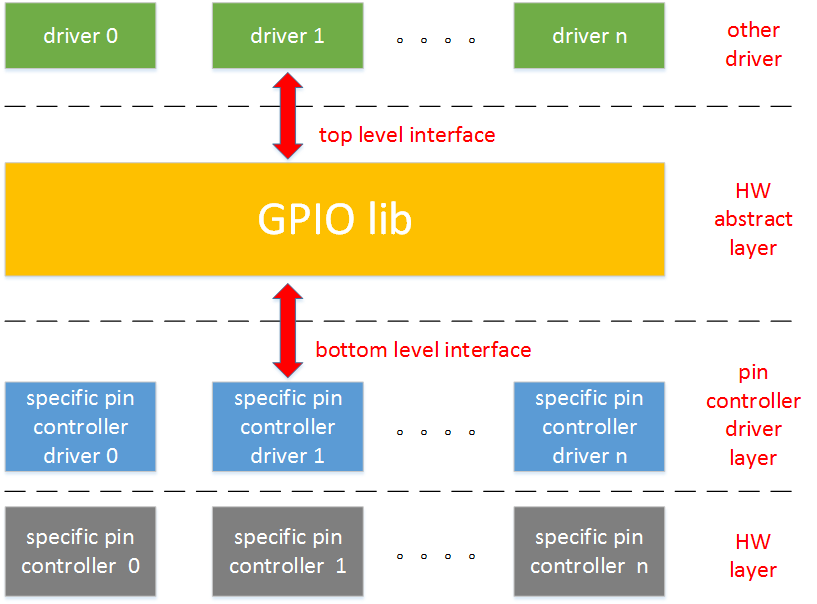


在上面中，定义了一个gpio-ctrl大节点，然后大节点中包含了usb\_otg和print\_pow子节点，子节点对应着一个个需要gpio控制的设备，在sysfs中导出设备对应得属性文件，便可以实现相关IO口的电平控制，看懂这个程序后，明白了内核中驱动程序编写需要考虑的一些东西，包括程序的可移植性、稳定性、兼容性等，另外，还需要让驱动程序发挥尽可能大的作用，甚至支持一类设备，还有就是，驱动程序在内核中运行后，需要对外提供设备的访问接口，对设备如何进行操作，应该留给应用层而不是内核层，内核层只负责对外提供出设备访问接口。

在看这个程序的这段时间，我也了解了Linux内核驱动中pinctrl子系统和GPIO子系统，pinctrl子系统功能非常强大，用于管理板子上的IO口，包括IO引脚的复用功能和电气特性等，在Linux内核中，pinctrl子系统的框架图如下所示：



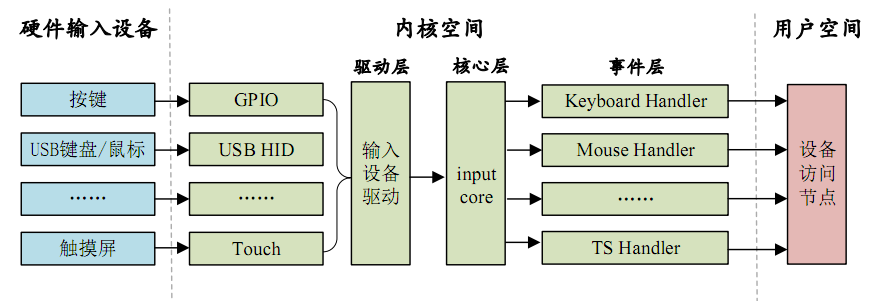
对上面这个框架图的理解为，这是内核驱动中的典型软件分层，bottom level interface接口是给予SoC厂商的bsp工程师使用，该层直接驱动和操作芯片上的硬件层，我们是直接用别的公司的SoC，因此，该层不需要过多关注，pinctrl子系统提供了top level interface接口，供其它驱动使用，当我们编写设备的驱动代码时，更需要关注该层，使用已经提供好的接口，当pin复用为GPIO（通用输入输出接口）便要使用到Linux内核中GPIO子系统，该子系统的软件分层和pinctrl子系统类似，如下：



在Linux内核驱动框架中，软件框架的分层都类似，我们更需要关注的是子系统提供的上层接口，在我们日常程序编写中应用非常多。

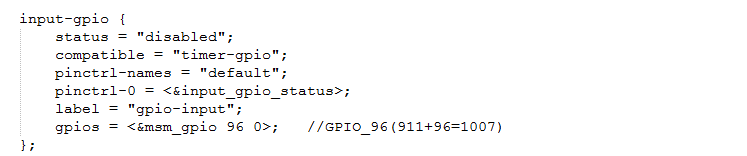
学习和理解了最简单的GPIO子系统后，根据林工的指导，开始接触input输入子系统，并需要完成按键上报事件的任务。

在我们接触的设备中，按键、触摸屏和鼠标等都属于输入设备，因此在编写驱动代码的时候必须要在程序中嵌入input输入子系统，该子系统的框架图如下所示：



在Linux内核中，使用struct input\_dev结构体进行输入子设备的描述，当我们需要使用input输入子系统中，需要在我们封装的设备结构中嵌入该结构体，在模块加载的时候，需要调用相关的API接口进行动态分配该结构体，另外，还需要在驱动程序中设置相应的事件位图，以及要上报给应用层的事件，例如，在任务中，我们需要上传按键的键码到应用层，则在驱动程序加载的时候，我们需要设置按键事件位图，并设置相应的键码位图，当按键发生的时候，便使用相关的API接口将按键事件上报到应用层，应用层则捕获上报的事件，进行下一步处理，在Android系统中的处理则是将键码进行映射，然后转换成相关的业务流，在Android应用层完成对应得按键业务处理，事件处理的流程都是类似的。

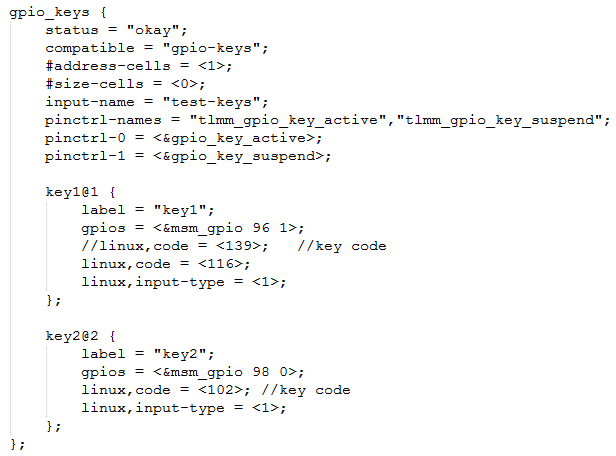
林工安排的任务为：编写一个驱动程序，在Linux内核层中上报一个按键事件到应用层。由于按键是输入子设备，因此必须要使用到input输入子系统，此外，按键还需要用到中断子系统，当按键按下后，产生相应中断触发，进入到中断服务程序后，做进一步的处理，例如上报事件，还有就是，我们需要考虑的是按键消抖问题，要不然，会进行多次触发，造成多次事件上报，在驱动程序中，我的实现方法为，驱动模块加载后，将按键IO口方向设置为输入，将IO号转换成IRQ号，然后向系统中申请中断资源，触发方式为双边沿触发，使用延时工作队列的方式进行按键消抖（也可以使用定时器消抖），延时工作队列初始化，定义延时工作队列的任务，并向系统中完成注册，当按键触发后，产生中断，进入到中断服务程序后，调用延时工作队列（线程）将处理的工作任务加载到队列中，在任务中进行按键的事件上报，使用延时工作队列的方式进行消抖，下面是按键的设备树节点：



由于程序比较长，就不完全贴出，下面是按键上报事件的函数：

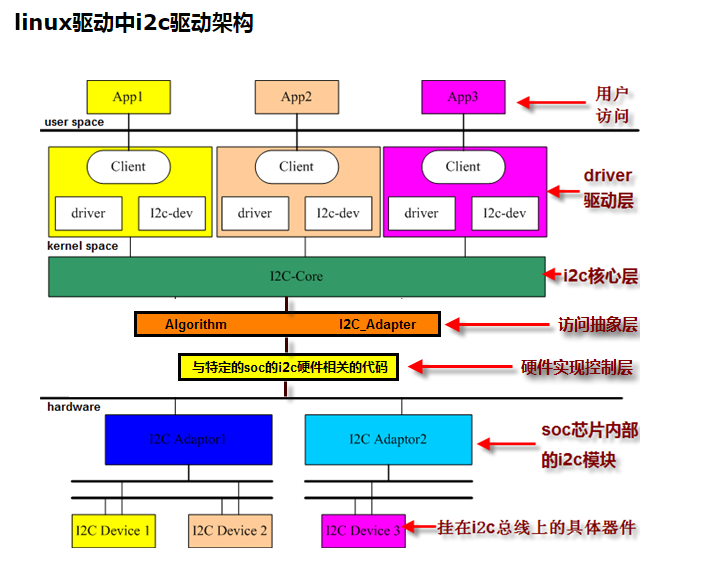


当然，关于按键的驱动程序，也可以直接使用内核里面提供的驱动程序gpio\_keys.c，只需要在设备树中定义相关的节点，节点中高的compatible属性值定义为“gpio-keys”，如下：



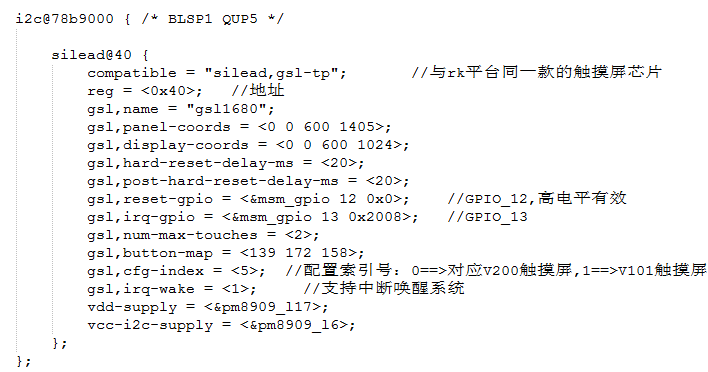
在编写按键的驱动程序时，了解和学习了Linux内核驱动中中断子系统的使用以及工作队列的使用，这些知识在Linux驱动编程中频繁使用到的知识点，也是内核驱动中经常使用到的子系统，所以，对于这方面的理解和学习非常重要，但是关于这方面的原理并非只是像调用API接口那么简单，当熟悉到一定的程度后，需要进一步加深对这方面子系统的学习，对其实现的原理需要有更深刻的理解。

在对input输入子系统、中断子系统以及工作队列等相关的知识具有一定的理解后，听从林工的指导，学习和理解公司产品TPS508的触摸屏驱动，TPS508的触摸芯片为gsl1680，支持多点电容触摸，触摸芯片的接口为I2C接口，在此期间，学习了Linux内核中的I2C子系统的驱动框架，Linux内核中I2C子系统的软件驱动框架图如下所示：



在I2C子系统中，需要注意主设备和从设备，支持一主多从模式，I2C的从设备挂接在I2C总线上，I2C从设备在Linux内核驱动中用struct i2c\_client结构体进行描述，也就是是客户端设备，关于I2C适配器的驱动程序由SoC厂商进行提供，我们做产品的不需要关注那么多，需要关注的是I2C子系统提供给顶层驱动程序的API接口，当我们编写相应的I2C驱动程序时，也是要调用其相关的顶层API接口，另外，Linux内核中提供了I2C驱动程序的驱动框架，在编写和I2C设备的驱动程序时，需要嵌入该驱动程序框架。

公司内很多产品都使用了gsl1680这个触摸芯片，该芯片的兼容性和稳定性都非常好，驱动程序大致处理流程为，当触摸屏按下后，会产生对应的IO口中断，进入到中断程序后，调度相关的工作队列，完成触摸坐标事件的上报，Android系统捕获对应的坐标事件后，做出相应的处理，下面是设备树中gsl1680触摸芯片的设备节点：



当我们在其它设备中使用该gsl1680触摸芯片，需要对设备树进行相应的修改，包括面板的大小以及LCD屏幕的分辨率和中断IO口等。

在试用的这段时间中，我的收获非常大，也让我逐渐感觉到了Linux内核的强大之处，对于Linux内核学习的地方还有很多很多，接下来的我，将会更加投入工作中，认真钻研于Linux内核驱动程序，努力完成领导下发的工作任务，最后，感谢天波同事的热情和耐心指导。

在试用期间，一些学习的知识点记录在了博客，链接如下：

<https://www.cnblogs.com/Cqlismy/>