1. 專案17\_platform:

* 我們在前面幾個端案編寫的字元設備驅動都非常的簡單，都是對 GPIO進行最簡單的讀寫操作。但是像I2C、SPI、LCD等這些複雜的驅動就不能這麼去寫了， Linux系統要考慮到驅動的可重用性，因此提出了驅動的分離與分層這樣的設計思路，因此誕生了我們將來最常使用的 platform設備驅動。

1. 驅動的分離與分層:

* 簡單來說，就是將設備信息從設備驅動中剝離開來，驅動使用標準方法去獲取到設備信息（比如從Device Tree中獲取到設備信息），然後根據獲取到的設備信息來初始化設備。如此一來就做到了，驅動只負責驅動，設備只負責設備，想辦法將兩者進行匹配即可。這個就是Linux中的總線（bus）、驅動（driver）和設備（device）模型，也就是常說的驅動分離。總線就是負責給驅動和設備連接的媒介。

1. platform驅動模型:

* 剛剛提到總線（bus）、驅動（driver）和設備（device）模型，但是在SOC中並不是所有設備都支援總線(bus)，因此Linux系統提出了虛擬總線(platform)的概念，對應的有platform\_driver和platform\_device。
* 而Linux 系統內核使用 bus\_type 結構體表示總線，此結構體定義在文件 include/linux/device.h中
* 其中比較重要的是match()函式及probe()函式，首先match()函數會先匹配設備和驅動，如果匹配成功probe()函數就會執行，而probe()函式就是實際驅動執行的函式。
* 總而言之，platform 驅動還是傳統的字元設備驅動、塊設備驅動或網路設備驅動，只是套上了一張 “platform” 的框架，目的是為了使用總線(bus)、驅動(driver)和設備(deivce)這個驅動模型來實現驅動的分離與分層。

1. platform設備模型:

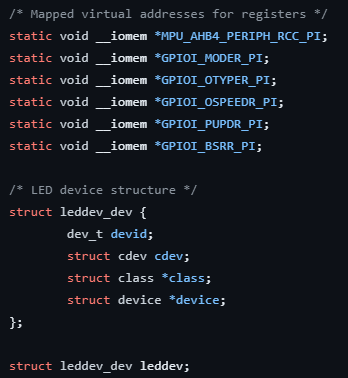
* platform 驅動已經準備好了，我們還需要 platform 設備，否則的話單單一個驅動也做不了什麼。
* platform\_device這個結構體表示platform設備，如果Kernel已經支援Device Tree的話就不必再使用platform\_device來描述設備了，因為這樣方便許多。當然如果想使用platform\_device就需要自行完善。而platform\_device結構體定義在文件include/linux/platform\_device.h 中。

1. platform驅動實作(於專案17\_platform/leddriver.c):

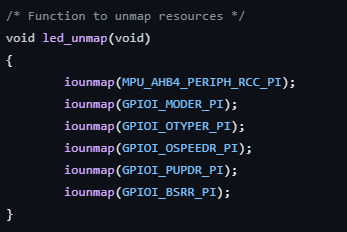
(本專案分兩個區塊驅動(leddriver.c)和設備(leddevice.c)分別編寫在不同的.c檔案中)

* 1. 首先是leddriver.c中，定義設備結構體與#define 暫存器地址映射

(如下圖一)以及iounmap()函式用在最後卸載驅動模塊時取消地址映射(如下圖二)。



(圖一:設備結構體及宏定義)



(圖二:iounmap()函式內容)

* 1. 接著是一般file\_operations中函式操作結構體(如下圖三):

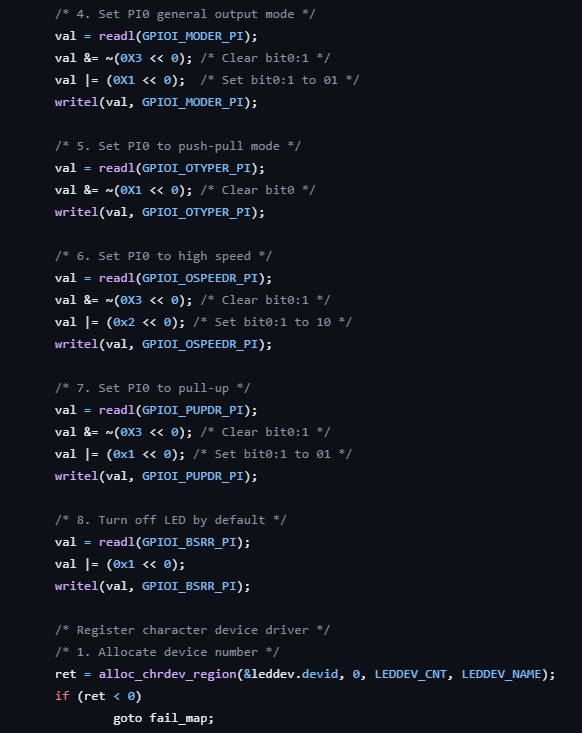


(圖三:file\_operations函式操作結構體內容)

* 1. 接著是本專案重點部分，probe()函式定義內容，其實就是把先前驅動入口函數的工作移到probe()函式執行而已。(如下圖四，圖五)



(圖四: probe()函式內容)



(圖五: probe()函式內容)



(圖六: probe()函式內容)

* 1. 然後再定義好platform\_driver結構體，用來匹配設備，加載/卸載驅動模塊函式。(如下圖七)



(圖七:platform\_driver結構體內容)

* 1. 最終就是驅動加載/卸載函數(如下圖八)，如此一來，相較於先前字元設備的編寫方式就簡潔許多。



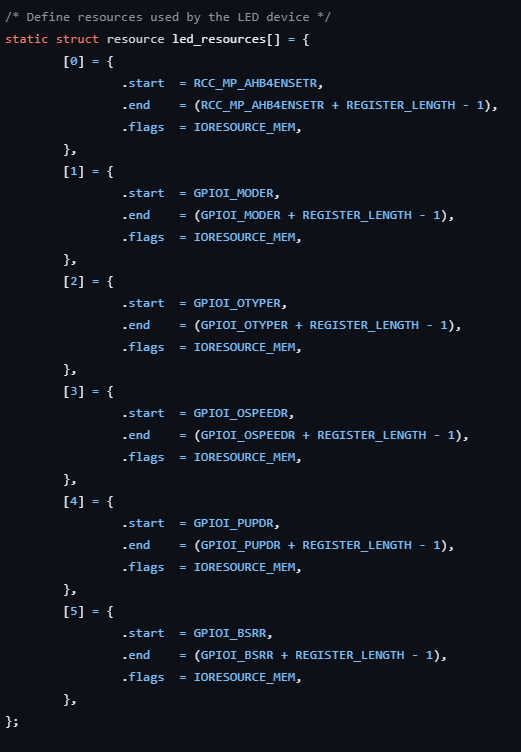
(圖八: 驅動加載/卸載函數內容)

※總結:

單純將先前字元設備驅動出入口函式內容，轉變成驅動加載/卸載函數來執行，也就是platform\_driver結構體中的’probe()函式’及’remove()函式’，如此一來就完成了驅動的部分。接下來是設備的部分。

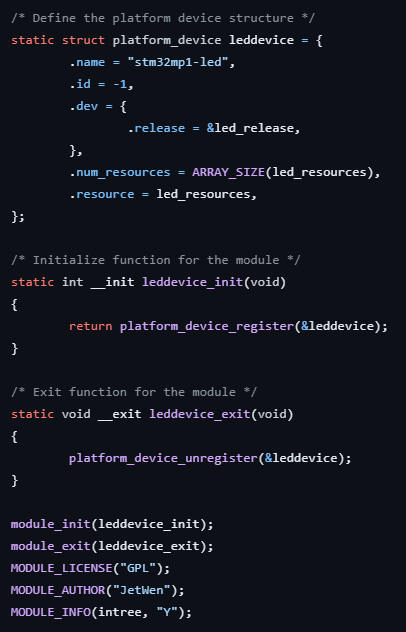
1. platform設備實作(於專案17\_platform/leddevice.c):
2. 首先要定義好設備的I/O資源，每個資源都有起始地址和終止地址。

(如下圖九)



(圖九:leddrvice設備資源)

1. 緊接著就是完善platform\_device結構體，並在初始化函式中呼叫platform\_device\_register()向Kernel註冊device，並且在結束設備時呼叫platform\_device\_unregister()向Kernel註銷device。(如下圖十)



(圖十: platform\_device結構體內容)

※總結:

如此一來就完善了platform\_device及platform\_driver之間的連結。值得注意的是，他們之間的匹配是透過結構體中”.name”來實現所以這個直必須一樣，才能匹配成功進一步觸發驅動函式的執行。