1. 專案18\_dtsplatform:

* 上一個專案17\_platform，我們知道總線(bus)，驅動(driver)，設備(device)的模型，也知道設備和驅動要先匹配成功才會執行probe()函式，配對方式是透過platform\_device和platform\_driver結構體中name成員變數去做匹配。這個專案我們要以Device Tree來做設備和驅動的匹配。

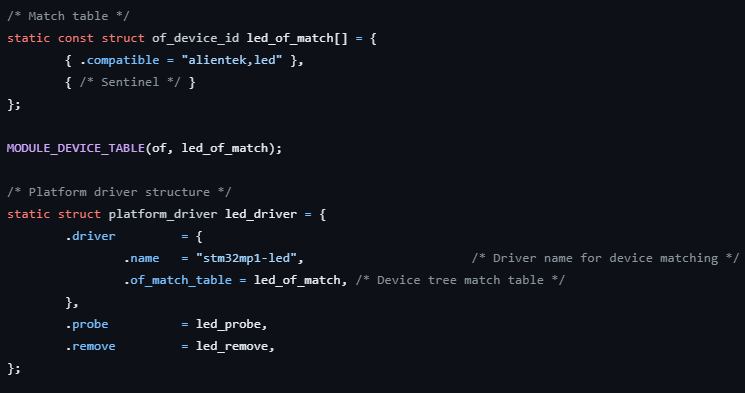
1. Device Tree下platform:

* 上一個專案17\_platform為了匹配驅動(driver)，設備(device)分別向Linux Kernel註冊platform\_driver和platform\_deivce，其中platform\_deivce是用來描述設備信息的，但是當我們引用Device Tree時設備訊息瞜已經在Device Tree中了，就不需要我們實現platform\_device了，沒錯，所以引用Device Tree比較簡潔只需要實現platform\_driver內容即可。
* 先前的專案都沒有使用到platform驅動框架，而實際上應該先配置Pin腳，及其電器屬性才對。所以這裡會引用到Device Tree文件下pinctrl節點來完成相關配置，接著要在Device Tree中創建設備節點來描述設備信息；重點，需要設置compatible屬性，因為platform是仰賴設備節點的compatible屬性來匹配設備和驅動。

1. 以Device Tree匹配platform實作:

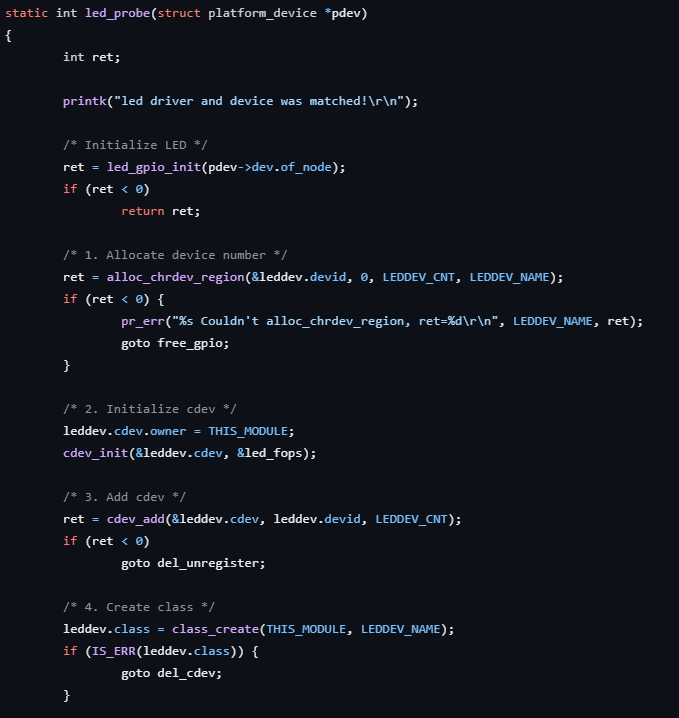
(\*實作內容大多數與17\_platform內容相同，本實作過程指說明差異部分。)

1. 定義”of\_device\_id led\_of\_match[]”這個id\_table是用來匹配Device Tree中的設備的再透過MODULE\_DEVICE\_TABLE()宏定義來實現驅動和設備的匹配，注意，其中compatible屬性中的字串必須和Device Tree中compatible屬性一致才能匹配成功。(如下圖一)



(圖一:id\_table結構體內容)

1. 接著實現platform\_driver結構體中的probe()函數(如下圖四，圖五)及remove()函數。(如下圖六)



(圖四:probe()函式內容)



(圖五:probe()函式內容)



(圖六:remove()函數內容)

1. 最後就是完善驅動出/入口函數，只需要在驅動入口函數透過platform\_driver\_register()函數註冊platform\_driver給Kernel，並在驅動出口函數透過platform\_driver\_unregister()函數來向Linux Kernel註銷platform\_driver，就完成了(如下圖七)。剩餘部分就是一般字元設備驅動內容，逐步完善即可。



(圖七:驅動出入口函數內容)

※結論:

當我們要以platform框架來實現驅動時，就需要匹配驅動和設備可以透過driver name來匹配，也可以透過Device Tree來匹配，就成功地把各個程式碼模塊化，除了增加可讀性也增加了可移植性。

剩餘部分就是已經很熟悉的字元設備驅動框架的完善，以及file\_operations中的操作函式完成，就完成一個驅動專案了。