1. 專案33\_DAC:

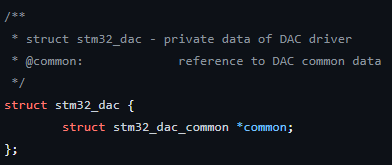
* 在專案32我們已經完成ADC驅動講解，接下來就是DAC解析，ADC 是模數轉換器，負責將外界的模擬信號轉換為數字信號。DAC 剛好相反，是數模轉換器，負責將 SOC 的數字信號轉換為模擬信號。

1. DAC驅動實作:

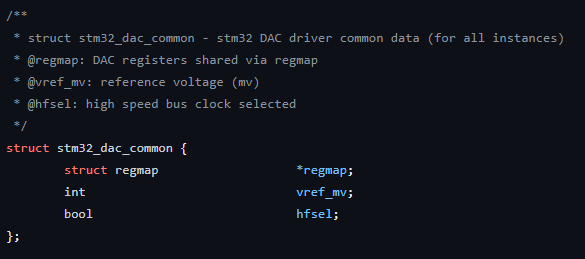
* 一樣ST已經幫我們編寫好DAC驅動程式，驅動文件也有兩個stm32-dac-core.c和stm32-dac.c。stm32-dac-core.c是DAC核心層，主要用於DAC時鐘、電源等初始化。我們需要重點關心的是stm32-dac.c這個文件。stm32-dac.c主體框架是platform，配合IIO驅動框架實現DAC驅動。

1. 首先觀察，stm32\_dac結構體(如下圖一)，而stm32\_dac結構體中有stm32\_dac\_common結構體(如下圖二)，定義在stm32-dac-core.h中。

* 其中可以發現DAC驅動也採用regmap API(如圖二)。



(圖一: stm32\_dac結構體)

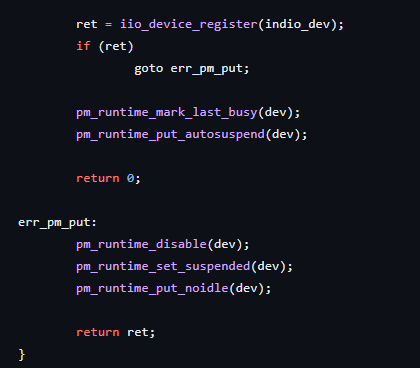


(圖二: stm32\_dac\_common結構體)

1. 接著重點觀察probe()函數(如圖三圖四)，probe()函數功能如下表一整理。



(圖三:probe()函式內容)



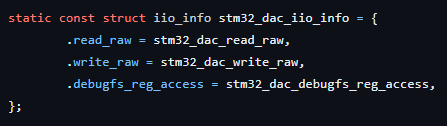
(圖四:probe()函式內容)

(表一:probe()函式功能整理)

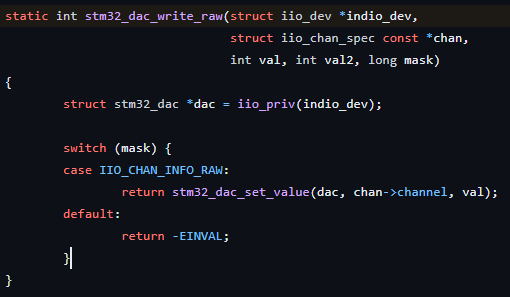
|  |  |
| --- | --- |
| **probe()函式步驟** | **相關程式碼片段** |
| 檢查設備節點 | if (!np) return -ENODEV; |
| 分配並初始化 IIO 設備結構 | indio\_dev = devm\_iio\_device\_alloc(&pdev->dev, sizeof(\*dac));  if (!indio\_dev) return -ENOMEM; |
| 保存設備專用數據 | platform\_set\_drvdata(pdev, indio\_dev); |
| 初始化 DAC 結構體 | dac = iio\_priv(indio\_dev); dac->common = dev\_get\_drvdata(pdev->dev.parent); |
| 初始化 DAC 通道 | ret = stm32\_dac\_chan\_of\_init(indio\_dev); if (ret < 0) return ret; |
| 電源管理設置 | pm\_runtime\_get\_noresume(dev); pm\_runtime\_set\_active(dev); |
| 註冊 IIO 設備 | ret = iio\_device\_register(indio\_dev);  if (ret) goto err\_pm\_put; |
| 標記最後忙碌時間 | pm\_runtime\_mark\_last\_busy(dev);  pm\_runtime\_put\_autosuspend(dev); |
| 錯誤處理 | err\_pm\_put: pm\_runtime\_disable(dev); pm\_runtime\_set\_suspended(dev);  pm\_runtime\_put\_noidle(dev); |

1. stm32\_dac\_iio\_info結構體(如下圖五)

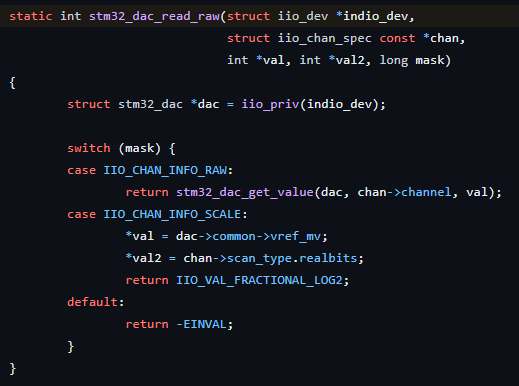
* 其綁定stm32\_dac\_read\_raw和stm32\_dac\_write\_raw函式(如圖六，圖七)。
* 而stm32\_dac\_read\_raw函式功能就是讀取DAC的原始值以及分辨率。stm32\_dac\_write\_raw函式功能就是向DAC寫入原始值，也就是設置DAC



(圖五: stm32\_dac\_iio\_info結構體內容)



(圖六: stm32\_dac\_write\_raw函式內容)

\

(圖六: stm32\_dac\_read\_raw函式內容)

※總結:

由於ST程式碼非常的長，考量篇幅因素重點注意上述這些內容。也是本專案DAC的重點部分。