EMBEDDED SYSTEMS · RTOS · EDGE AI

STM32 × FreeRTOS

智慧語音控制

智慧居家語音控制器完整實作課程

從硬體驅動到 Edge AI 產品化

STM32 HAL | FreeRTOS | I2S / DMA | X-CUBE-AI | MQTT | OTA | Matter

本課程以「智慧居家語音控制器」貫穿全程：學員從 STM32 架構、GPIO / PWM 驅動、FreeRTOS 多任務與 IPC，一路做到麥克風擷取、喚醒詞、語音辨識、設備狀態機，最後加入 Watchdog、低功耗與 OTA，完成接近實際產品的嵌入式 AI 系統，並可交給 Siri、Alexa 與 OK Google 遠端控制。

適合對象：具備 C 語言基礎、想進入嵌入式與 Edge AI 的學生與工程師。

建議時數：12 週 / 36 小時（另有 Siri / Alexa / Google 整合選修週），可濃縮為 5 日工作坊。

參考板：NUCLEO-F429ZI（主線） / STM32F407G-DISC1 / STM32H747I-DISCO（AI）。

版本 1.0　編譯日期：2026 年 8 月 30 日　韌體目錄：`firmware/`

前言

語音已經是人機介面的一等公民。手機助理、車載系統、智慧音箱證明了一件事：當「說話」比「按按鈕」更自然時，嵌入式產品就必須同時懂*即時控制*與*感知推論*。本課程不把這兩件事拆開教——我們用一個真實產品把它們接起來。

為什麼是 STM32 + FreeRTOS + 語音。 STM32 是華語圈與業界最常見的 Cortex-M 平台，HAL 與 CubeMX 降低了啟動門檻，卻仍保留暫存器級的能見度。FreeRTOS 是嵌入式即時系統的事實標準：任務、佇列、信號量、事件組足以支撐語音管線這種「週期擷取 + 突發推論 + 非同步控制」的工作型態。語音則是把學員從「會閃 LED」推進到「系統會聽、會判斷、會動作」的最佳載體。

兩條語音路徑，同一套 RTOS 骨架。 現場硬體與算力差異很大。本課程同時支援：

- **路徑 A（建議入門）**：外部語音模組（UART，例如 CI-03T、ASRPRO 或 Gravity 離線語音模組）負責喚醒與指令辨識，STM32 專職即時控制。
- **路徑 B（進階）**：板上 I2S 麥克風 + DMA + X-CUBE-AI，在 STM32H7 ／ F7 上跑關鍵詞偵測（KWS）與小型指令分類。

兩條路徑共用同一套指令協定、設備狀態機與 FreeRTOS 任務拓撲。學完路徑 A 再換路徑 B，不需要重寫產品邏輯。第 13 章另補上**路徑 C（智慧助理）**：Siri、Alexa 與 OK Google 經 HomeKit ／區網模擬／ Matter 或 Home Assistant 開道下指令——喚醒與辨識由手機或音箱代勞，STM32 端只多一個 Gateway 任務，指令佇列與安全策略原封不動。

怎麼使用這本教材。 每一章都對應 `firmware/` 底下的一個實驗專案。建議流程是：先讀觀念與架構圖，再對照程式碼註解，最後在開發板上跑實驗並完成章末練習。完整專案在 `firmware/voice_home_final/`，它把第 3 章到第 12 章的模組組裝成可出貨雛形。

你不會在這本書裡看到的東西。 我們不複製 STM32Cube 產生的數千行初始化碼，也不重貼 FreeRTOS 核心原始檔。教材寫的是*你必須理解並親自維護*的應用層：任務、驅動包裝、協定、狀態機與產品化機制。CubeMX 設定步驟會寫清楚，產生檔請以你的工程為準。

官方延伸。 ST 已公開 FreeRTOS 移植、X-CUBE-FREERTOS、FP-AI-VISION1、X-CUBE-AI 音訊範例與 STM32Cube.AI 開發流程。第 9 章與附錄 B 會對照這些套件，方便學員在課程結束後繼續往官方生態走。

開始之前，請確認你能用 *C* 寫出函式、指標與結構體，並知道什麼是中斷。其餘——從時鐘樹到 *OTA*——我們會一步步做完。

目錄

前言 i

0 課程導論與專案全貌 1

0.1 產品規格：智慧居家語音控制器 1
0.2 為什麼不能用超級迴圈 2
0.3 系統架構 2
0.4 十二週學習路徑 3
0.5 硬體兩條路徑怎麼選 3

1 STM32 嵌入式系統架構 5

1.1 從核心到腳位 5
1.2 記憶體模型：你的變數住哪裡 6
1.3 時鐘樹：萬物的節拍器 6
1.4 三種資料搬運 6
1.5 啟動到 main 的路 7
1.6 中斷優先權與 RTOS 7

2 開發環境、CubeMX 與 HAL 8

2.1 你需要安裝的東西 8
2.2 第一個專案：CubeMX 點選清單 8
2.3 HAL 怎麼看 9
2.4 實驗：1 Hz LED + UART 9
2.5 除錯三板斧 10

3 GPIO 與周邊設備控制 11

3.1 電氣與安全 11
3.2 設備抽象：不要讓任務知道腳位 11
3.3 實驗：四個設備的手動測試 12
3.4 蜂鳴器：有時限的動作 13

4 PWM、定時器與馬達／風扇 14

4.1 ARR、PSC、CCR . . . 14
4.2 直流風扇與 MOSFET . . . 14
4.3 伺服：角度對應脈寬 . . . 15
4.4 實驗：序列埠調光與開門 . . . 15

5 FreeRTOS 核心：任務、優先權與排程 17
5.1 任務是什麼 . . . 17
5.2 本專案的優先權表 . . . 17
5.3 建立任務 . . . 18
5.4 延遲：相對 vs 絕對 . . . 19
5.5 堆疊與 heap . . . 19
5.6 實驗：三任務心跳 . . . 20

6 Queue、Semaphore 與 Mutex 21
6.1 Queue：把命令當成郵件 . . . 21
6.2 Semaphore：事件，不是資料 . . . 22
6.3 Mutex：保護共享匯流排 . . . 23
6.4 優先權反轉（課堂必考） . . . 23
6.5 實驗：按鍵經佇列切設備 . . . 23

7 Event Group、軟體定時器與系統同步 25
7.1 系統事件位元 . . . 25
7.2 軟體定時器 . . . 26
7.3 與 Watchdog 的分工 . . . 26
7.4 實驗：喚醒窗與蜂鳴定時 . . . 27

8 音訊擷取：麥克風、I2S 與 DMA 28
8.1 取樣參數 . . . 28
8.2 麥克風選型 . . . 28
8.3 I2S + DMA 循環模式 . . . 29
8.4 路徑 A：UART 幀當「音訊」 . . . 29
8.5 簡易能量偵測 . . . 30
8.6 實驗 . . . 30

9 喚醒詞與語音辨識 32

9.1 狀態：Idle → Listen → Execute . . . 32
9.2 路徑 A：外部模組 . . . 32
9.3 路徑 B：板上 Edge AI . . . 33
9.4 拒絕與防抖 . . . 33
9.5 實驗 . . . 34

10 指令解析與設備控制狀態機 35
10.1 每個設備一份狀態 . . . 35
10.2 套用命令 . . . 35
10.3 逾時命令 . . . 36
10.4 可單測的邏輯 . . . 36
10.5 實驗：口令驅動全家電 . . . 37

11 通訊協定：UART、JSON 與 MQTT 38
11.1 兩條 UART 的理由 . . . 38
11.2 JSON 最小集 . . . 38
11.3 通訊任務結構 . . . 39
11.4 MQTT（選做） . . . 39
11.5 實驗 . . . 40

12 產品化：異常處理、Watchdog、低功耗與 OTA 41
12.1 分層的失敗 . . . 41
12.2 獨立看門狗 . . . 41
12.3 HardFault 最小處理 . . . 42
12.4 低功耗 . . . 42
12.5 OTA 最小可靠集 . . . 42
12.6 最終專案驗收 . . . 43

13 智慧助理整合：Siri、Alexa 與 OK Google 44
13.1 路徑 C：辨識搬到生態系那一側 . . . 44
13.2 三種整合策略 . . . 45
13.3 閘道協定：STM32 端 . . . 45
13.4 Siri：HomeKit 與 HomeSpan . . . 47
13.5 Alexa：fauxmoESP 本地模擬與正式技能 . . . 48

13.6 OK Google：走 Matter 或 Home Assistant 49
13.7 Matter：一次接好三家 49
13.8 Home Assistant 樞紐：接第 11 章的 MQTT 50
13.9 實驗：lab12 助理閘道 50

A 硬體清單、官方範例與除錯 **52**
A.1 附錄 A 建議 BOM 52
A.2 附錄 A2 腳位建議（NUCLEO-F429ZI） 52
A.3 附錄 B STM32 官方延伸 53
A.4 附錄 C 常見錯誤速查 54
A.5 附錄 D FreeRTOSConfig 教學預設 54
A.6 附錄 E 課程評量建議 55
A.7 附錄 F 韌體目錄索引 55

第 0 章

課程導論與專案全貌

本章學習目標

學完本章你將能夠：

- 用一句話說明本專案要做什麼，以及它為什麼需要 RTOS 而不是超級迴圈。
- 畫出「語音進、設備動、雲端知」三條資料路徑。
- 列出 12 週課程對應的能力，並知道每週該跑哪個韌體專案。
- 依預算與算力選擇路徑 A（外部 ASR 模組）或路徑 B（板上 Edge AI）。

嵌入式課程最常見的失敗模式，是學員會閃燈、會讀 ADC，卻無法把多個週邊組成一個「產品」。本課程反過來設計：先給你產品規格，再把規格拆成硬體、驅動、RTOS、AI 與可靠度五層。每一層都有可執行的實驗，最後疊成同一套韌體。

0.1 產品規格：智慧居家語音控制器

目標裝置放在客廳或實驗室桌面，用口語控制燈光、風扇、門鎖（伺服馬達）與一個插座繼電器，並能回報溫度與系統狀態。

表 1: 最小可行產品（MVP）規格

面向	需求	驗收方式
喚醒	聽到喚醒詞後進入 5 秒聆聽窗	狀態 LED 由呼吸燈改為常亮
指令	至少 10 條中文 / 英文指令	對應設備在 300 ms 內動作
並發	語音、感測、通訊互不阻塞	關燈時風扇仍維持轉速
助理	可由 Siri / Alexa / Google 控制	手機 App 與裝置狀態雙向同步
安全	未知指令不動作；看門狗復位	人為卡死任務後 2 s 內重啟
功耗	閒置可進 Stop 模式	電流計顯示 mA 級下降
維運	UART 可觸發韌體更新流程	模擬 OTA 後版本號加一

指令表（路徑 A / B / C 共用同一枚舉）：

口語	內部命令
開燈 / 關燈 / 亮一點 / 暗一點	`CMD_LIGHT_*`
開風扇 / 關風扇 / 快一點 / 慢一點	`CMD_FAN_*`
開門 / 關門	`CMD_DOOR_*`
開插座 / 關插座	`CMD_RELAY_*`
報警 / 停止報警	`CMD_BUZZER_*`
現在幾度	`CMD_QUERY_TEMP`
全部關閉	`CMD_ALL_OFF`
睡覺	`CMD_ENTER_SLEEP`
離家	`CMD_HOME_LEAVE`

0.2 為什麼不能用超級迴圈

超級迴圈（super-loop）把所有工作塞進 `while(1)`：讀麥克風、跑推論、掃感測器、送 UART。語音幀是硬即時的——16 kHz、16-bit、單聲道時，每 10 ms 就有 320 bytes 必須被取走，否則 DMA 緩衝會覆寫。若此時你正在等 DHT22 的 20 ms 時序，或阻塞在 MQTT 寫出，音訊就丟了。

FreeRTOS 的價值不是「比較潮」，而是把不同時間尺度的工作拆開：

- **硬即時**：I2S DMA 完成、音訊搬運（高優先權）。
- **軟即時**：喚醒 / 辨識、設備 PWM 更新（中優先權）。
- **盡力而為**：感測器、遙測、日誌（低優先權）。

任務之間用 Queue 傳「結果」而不是共用全域變數；用 Semaphore 對齊 DMA 完成；用 Event Group 廣播「已喚醒 / 發生故障 / 準備 OTA」這類系統級旗標。

0.3 系統架構

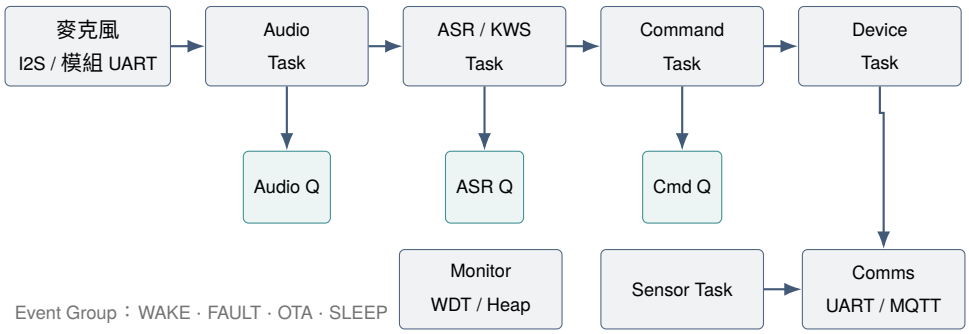


圖 1: 智慧居家語音控制器的任務與資料路徑

圖 1 是全書的地圖。之後每一章只加深其中一塊，不會另起爐灶。第 13 章會在 Comms 旁

再加一個 Gateway 任務，讓 Siri、Alexa 與 Google 的指令也匯入同一條 Cmd Queue——入口變多，骨架不變。

0.4 十二週學習路徑

表 2: 課程進度與韌體對照

週	主題	你會做出來的東西	目錄
1	架構與環境	CubeIDE 專案能下載、LED 閃	`lab01`
2	STM32 架構	看懂時鐘、記憶體、中斷向量	—
3	GPIO 設備	LED、蜂鳴器、繼電器可單獨控	`lab02`
4	PWM 與馬達	風扇轉速、伺服開門角度	`lab03`
5	FreeRTOS 任務	三個任務並行閃不同週期	`lab04`
6	Queue / Sem	按鍵經佇列安全地切設備	`lab05`
7	Event / Timer	喚醒窗、看門狗餵食、軟體定時	`lab06`
8	音訊擷取	I2S+DMA 或模組 UART 收幀	`lab07`
9	語音辨識	喚醒詞 + 10 條指令	`lab08`
10	狀態機	口令驅動完整設備邏輯	`lab09`
11	通訊	JSON 遙測、可選 MQTT	`lab10`
12	產品化	WDT、低功耗、OTA 雛形	`lab11` + `final`
13*	智慧助理	Siri / Alexa / Google 閘道整合	`lab12`

說明 每週建議節奏：90 分鐘講解 + 90 分鐘實驗。若辦成五日工作坊，第 1–2 日對應週 1–5，第 3 日對應週 6–8，第 4 日對應週 9–10，第 5 日對應週 11–12 與成果展示。第 13 週（*）為選修：把裝置接上 Siri、Alexa 與 OK Google，可併入成果週或作為課後專題。

0.5 硬體兩條路徑怎麼選

表 3: 路徑選擇

	路徑 **A**：外部 **ASR**	路徑 **B**：板上 **Edge AI**
開發板	NUCLEO-F429ZI 或 F407 Discovery	STM32H747I-DISCO 或 F746-DISCO
語音硬體	CI-03T / ASRPRO / Gravity 模組	MP34DT01 或 INMP441 + X-CUBE-AI
優點	便宜、中文指令現成、除錯容易	真正理解特徵與推論、無額外模組
代價	辨識能力受模組限制	模型、記憶體、延遲都要自己扛
本課程預設	**主線實驗**	第 9 章進階 + 附錄 B

無論選哪一條，請準備：USB 線、麵包板、LED×3、主動蜂鳴器、5 V 繼電器模組、N-MOSFET + 12 V 風扇（或先用 PWM LED 模擬）、SG90 伺服、DHT22 （可選）、邏輯分析儀或

至少一個 USB 轉 UART。完整 BOM 見附錄 A。

> **對應程式碼**
>
> 本章沒有新的 C 檔。請先把倉庫的 `firmware/` 目錄瀏覽一遍，確認 `common/include/app_types.h` 裡的命令列舉與你在紙上畫的規格一致。下一章開始真正碰到矽晶片。

第 1 章

STM32 嵌入式系統架構

本章學習目標

學完本章你將能夠：

- 指出 Cortex-M4 核心、匯流排矩陣、Flash ／ SRAM、GPIO 與週邊的關係。
- 解釋為什麼時鐘樹錯了，UART 鮑率與 I2S 取樣率都會一起錯。
- 分清輪詢、中斷、DMA 三種資料搬運，並知道語音該用哪一種。
- 讀懂啟動檔、向量表與 `Reset_Handler` 在 `main` 之前做了什麼。

還沒寫應用程式之前，先建立一張「晶片內部地圖」。之後 FreeRTOS 的 Tick、I2S 的 DMA、Watchdog 的獨立時鐘，都會回到這張地圖。

1.1 從核心到腳位

STM32F429 這類 MCU 可以想成四層：

1. **Arm Cortex-M4F**：執行 Thumb-2 指令、巢狀中斷（NVIC）、可選 FPU。FreeRTOS 的 context switch 就是在這裡存／還暫存器。
2. **匯流排矩陣**：I-Code、D-Code、System、AHB1/2、APB1/2。CPU、DMA、USB 可以同時走不同路徑，這是「語音 DMA 不卡 CPU」的硬體前提。
3. **記憶體**：內部 Flash 放程式與常數；SRAM 放堆疊、堆、`.bss` 與 DMA 緩衝。F429 約 2 MB Flash、256 KB SRAM，夠跑 FreeRTOS + 路徑 A；路徑 B 的 Mel 特徵與 NN 權重通常需要 H7 的更大 SRAM 或外部 SDRAM。
4. **週邊**：GPIO、TIM、USART、I2C、SPI、I2S、ADC、RTC、IWDG、WWDG。應用程式幾乎不直接碰核心，而是設定這些區塊。



圖 1.1: STM32 內部層次（教學用簡圖）

1.2 記憶體模型：你的變數住哪裡

連結腳本（`.ld`）決定符號落點。對語音專案特別重要的三件事：

- **堆疊**：每個 FreeRTOS 任務有自己的堆疊，從 FreeRTOS heap 配置，不是啟動檔那個主堆疊（MSP）包辦一切。
- **DMA 緩衝**：必須在 DMA 能看到的 SRAM，且對齊快取（H7 有 D-Cache 時還要做 cache clean/invalidate）。
- **大陣列**：梅爾頻譜或模型張量不要放在任務堆疊上，應設為 `static` 或從 heap 配置。

> **注意**　F4 的 CCM RAM 很快，但 **DMA 通常不能存取 CCM**。若你把 I2S 緩衝放進 CCM，現象是「設定看起來都對，緩衝永遠是 0」。這是課堂上最高頻的陷阱之一。

1.3 時鐘樹：萬物的節拍器

上電後 MCU 跑在內部 HSI（16 MHz 量級）。CubeMX 的 Clock Configuration 會把 HSE → PLL → SYSCLK，再分頻給 AHB / APB。本課程建議：

- F429：SYSCLK 180 MHz（注意過驅動與 Flash wait state）。
- APB1 計時器時鐘要能整除出 1 MHz 或 10 kHz，方便 PWM。
- I2S 需要獨立的精準時鐘（PLLI2S）。取樣率 16 kHz 算錯 1%，語音特徵就會漂，辨識率會無聲地崩掉。
- IWDG 掛在 LSI，即使主時鐘停了（Stop 模式）仍能復位——這是第 12 章低功耗能「睡死也能醒」的原因。

1.4 三種資料搬運

表 1.1: 輪詢、中斷、DMA

方式	CPU 在做什麼	本專案用法
輪詢	一直讀旗標	初始化、極短的暫存器確認
中斷	事件來了才進去 ISR	UART 收 ASR 位元組、按鍵、EXTI
DMA	CPU 零介入搬一長段	I2S 音訊、USART 大量遙測

ISR 裡只做最少的事：清旗標、給 Semaphore、`xQueueSendFromISR`。推論、字串解析、`printf` 全部留給任務。違反這條，輕則優先權反轉，重則在 ISR 裡開銷過大導致丟音。

1.5 啟動到 main 的路

重置後硬體跳到 Flash 開頭的向量表。`Reset_Handler`（在 `startup_stm32f429xx.s`）會：複製 `.data`、清 `.bss`、呼叫 `SystemInit`（設 FPU、重定位向量、預設時鐘），再進 `main`。Cube 產生的 `main` 通常先 `HAL_Init`、`SystemClock_Config`、再初始化週邊，最後才建立 FreeRTOS 任務並 `osKernelStart`——那之後 `main` 就再也不回來了。

程式碼 1.1: 啟動後應用程式應有的心理模型（偽碼）

```
int main(void)
{
    HAL_Init();                 /* SysTick 先給 HAL，稍後交給 FreeRTOS */
    SystemClock_Config();       /* 時鐘樹一次設對 */
    MX_GPIO_Init();
    MX_DMA_Init();              /* 必須在 I2S/USART 之前 */
    MX_I2S2_Init();
    MX_USART3_UART_Init();
    osKernelInitialize();
    app_create_objects();       /* Queue / Sem / EventGroup */
    app_create_tasks();
    osKernelStart();            /* 不再返回 */
    while (1) { }               /* 不應該走到這裡 */
}
```

> **提示**　CubeMX 勾選 FreeRTOS 後，HAL 的時間基準會改成 TIM 而把 SysTick 讓給 RTOS。若你手動改 `FreeRTOSConfig.h` 卻忘了這件事，會出現「延遲函數亂跳、HAL_Delay 卡死」的幽靈 bug。

1.6 中斷優先權與 RTOS

Cortex-M 的優先權數字越小越緊急。FreeRTOS 要求：任何會呼叫 `*FromISR` 的中斷，其邏輯優先權不能高過 `configMAX_SYSCALL_INTERRUPT_PRIORITY`。語音 DMA 完成中斷應設在「可以呼叫 FreeRTOS API」的範圍內；硬體故障（HardFault）則不走這套。

> **動手練習**
>
> 打開你即將使用的 `.ioc`，在 Clock Configuration 截一張圖，標出 SYSCLK、APB1 Timer clock、I2S clock。再打開 startup 檔，找到向量表裡 `DMA1_Stream4_IRQHandler` 的位置（名稱依實際流而變）。把這兩張圖貼進實驗筆記本——第 8 章會用到。

第 2 章

開發環境、CubeMX 與 HAL

本章學習目標

學完本章你將能夠：

- 安裝 STM32CubeIDE、接上 Nucleo、用 ST-LINK 下載第一個程式。
- 用 CubeMX 產生 GPIO + FreeRTOS 骨架，並理解哪些檔不可手改。
- 區分 HAL、LL、暫存器三級 API，知道本課程為何以 HAL 為主。
- 設定 UART 除錯輸出，養成「先看 log 再猜硬體」的習慣。

工具鏈是嵌入式的第一道牆。本章目標只有一個：到下課時，板上的使用者 LED 以 1 Hz 閃爍，且 UART 印出 `hello voice-home`。

2.1 你需要安裝的東西

1. **STM32CubeIDE**（內含 gcc-arm、CubeMX、GDB、ST-LINK）。
2. 對應系列的 **STM32CubeF4 / F7 / H7** 套件（IDE 會自動拉）。
3. **STM32CubeMX** 可獨立使用，但本課程以 IDE 內建為準。
4. 可選：**STM32CubeMonitor**、邏輯分析儀軟體、**STM32Cube.AI**（路徑 B）。

接線：Nucleo 的 ST-LINK USB（通常標 CN1）接到電腦。Windows 需 ST-LINK 驅動；macOS / Linux 多半即插即用。若 IDE 看不到探針，先更新 ST-LINK 韌體（IDE 的 *Help → ST-LINK Upgrade*）。

2.2 第一個專案：CubeMX 點選清單

新建 *STM32 Project*，選 NUCLEO-F429ZI（或你的板）。在 Pinout：

- 讓 IDE 套用板級預設，使用者 LED（F429ZI 上是 PB0 / PB7 / PB14）會自動設為 GPIO_Output。
- Connectivity → USART3（Nucleo 的 VCP）設 Asynchronous，115200 8N1。
- Middleware → FREERTOS → CMSIS_V2。
- 在 Clock 頁把 HSE 打開（Nucleo 有 8 MHz 從 ST-LINK 進來），PLL 拉到 180 MHz。
- Project Manager → Code Generator：勾選 *Generate peripheral initialization as a pair of '.c/.h'*，

以及 *Keep User Code when re-generating*。

產生程式碼後，只在 `/* USER CODE BEGIN */ ... /* USER CODE END */` 區塊寫你的邏輯。被 Cube 管的 `MX_*_Init` 不要手改，否則下次重新產生會被覆寫。

2.3 HAL 怎麼看

HAL 把「設暫存器」包成有傳回值的函式。本課程常用家族：

`HAL_GPIO_WritePin / TogglePin`	數位輸出
`HAL_TIM_PWM_Start / set compare`	風扇與伺服
`HAL_UART_Transmit / Receive_IT / DMA`	ASR 與遙測
`HAL_I2S_Receive_DMA`	麥克風
`HAL_IWDG_Refresh`	餵獨立看門狗
`HAL_PWR_EnterSTOPMode`	低功耗

傳回 `HAL_OK` 才算成功。語音路徑上每一個 DMA 啟動都要檢查，失敗應點亮故障 LED 並寫入 Event Group 的 `FAULT` 位，而不是假裝沒發生。

> **說明** LL（Low Layer）更接近暫存器、較省週期。教學階段用 HAL 較易讀；若你在 I2S ISR 裡真的量到超時，再針對熱路徑改 LL。不要一開始就全專案 LL，可讀性會崩。

2.4 實驗：1 Hz LED + UART

> **對應程式碼**
>
> 完整可編譯應用層見 `firmware/lab01_blinky/`。把該目錄的 `app_main.c` 內容貼進 Cube 專案的 USER CODE，或直接把檔案加入建置。

程式碼 **2.1:** lab01：尚未上 RTOS 的超級迴圈心跳

```c
#include "app_board.h"

void app_main(void)
{
    uint32_t tick = 0U;
    char line[48];

    app_log("hello voice-home\r\n");

    for (;;) {
        HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
```

```
        int n = snprintf(line, sizeof line, "tick %lu\r\n",
                         (unsigned long)tick++);
        if (n > 0) {
            HAL_UART_Transmit(&huart3, (uint8_t *)line,
                              (uint16_t)n, 50);
        }
        HAL_Delay(500);
    }
}
```

用終端機（115200）應看到持續增加的 `tick`。若 LED 閃但沒有字，十之八九是 USART 腳位不是 VCP 那一組，或鮑率與時鐘不一致。

2.5 除錯三板斧

1. **SWD 斷點**：看變數與週邊暫存器。不要在 I2S 熱路徑上長時停住，DMA 會繼續跑。
2. **UART log**：模組化成 `app_log()`，之後 RTOS 裡改成經佇列輸出，避免多任務打架。
3. **GPIO 翻轉**：在 ISR 進出各翻一次腳，用邏輯分析儀量真實週期。這比任何 printf 都準。

> **注意**　`printf` 經 semihosting 極慢，且在未接 debugger 時可能卡死。本課程禁止 semihosting，一律走 UART。

> **動手練習**
>
> 把閃爍週期改成 200 ms，並讓 LD2 與 LD1 相位相反。量測 UART 兩行 `tick` 的時間差，確認 `HAL_Delay` 與你的 SYSCLK 設定吻合。若差超過 5%，回頭查時鐘樹。

第 3 章

GPIO 與周邊設備控制

本章學習目標

學完本章你將能夠：

- 設定推挽、開漏、上拉 / 下拉，並為繼電器與蜂鳴器選對模式。
- 寫出設備抽象層，讓上層只認 `dev_light_set()` 而不是腳位號碼。
- 處理繼電器的低電平觸發與續流二極體，避免 MCU 腳直接扛電感負載。
- 用有限狀態避免「蜂鳴器卡在響」這種產品級醜態。

語音指令最後一定落到「改某根腳的電位」。先把這層做對、做乾淨，後面的 RTOS 與 AI 才有地方接。

3.1 電氣與安全

- **LED**：串 220 Ω–1 kΩ，MCU 腳推挽輸出。
- **主動蜂鳴器**：自帶振盪，GPIO 高 / 低即可；被動蜂鳴器要 PWM（下一章）。教學用主動較單純。
- **繼電器模組**：幾乎都是「光耦 + 電晶體」，標 `IN` 的腳給 3.3 V GPIO。注意有的模組低電平觸發。
- **絕對不要用 MCU 腳直接驅繼電器線圈或風扇。電感踢回會毀掉腳位。**



圖 3.1: 邏輯層與功率層必須分開

3.2 設備抽象：不要讓任務知道腳位

應用層出現 `HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, ...)` 是壞味道。腳位會因改板而變；語音任務不該重新編譯只因為 LED 換腳。本課程的 `dev_*` API 定義在 `firmware/common/`：

程式碼 3.1: 設備抽象（節錄）

```
typedef enum {
    DEV_LIGHT = 0,
```

```
    DEV_FAN,
    DEV_DOOR,
    DEV_RELAY,
    DEV_BUZZER,
    DEV_COUNT
} dev_id_t;

void     dev_init(void);
void     dev_light_set(uint8_t on);
void     dev_light_set_level(uint8_t percent); /* 0--100，PWM 章實作 */
void     dev_relay_set(uint8_t on);
void     dev_buzzer_beep(uint16_t ms);
void     dev_all_off(void);
```

路徑 A / B 與第 10 章狀態機都只呼叫這些函式。換板時只改 `pin_map.h`。

3.3 實驗：四個設備的手動測試

> **對應程式碼**
>
> `firmware/lab02_gpio_devices/`。序列埠下指令：`L1/L0` 燈、`R1/R0` 繼電器、`B` 蜂鳴 200 ms、`A` 全關。尚未使用 FreeRTOS。

程式碼 3.2: 以單字元協定驗證硬體（節錄）

```
void app_main(void)
{
    uint8_t ch;
    dev_init();
    app_log("lab02: L1/L0 R1/R0 B A\r\n");

    for (;;) {
        if (HAL_UART_Receive(&huart3, &ch, 1, 10) != HAL_OK)
            continue;
        switch (ch) {
        case 'L': /* 下一個位元為 1 或 0，見完整檔 */ break;
        case 'B': dev_buzzer_beep(200); break;
        case 'A': dev_all_off(); break;
        default:  break;
        }
    }
}
```

建議用麵包板先驗證每一路，再焊或接杜邦線到最終佈局。繼電器吸合時用電表量負載側，不要用耳朵當唯一驗收。

3.4 蜂鳴器：有時限的動作

產品不能「指令來了就 `WritePin(1)` 然後忘記關」。`dev_buzzer_beep` 在本章用 `HAL_Delay` 阻塞實作；進 FreeRTOS 後改成軟體定時器（第 7 章），避免卡死語音任務。

程式碼 3.3: 阻塞版蜂鳴——只允許在還沒有 RTOS 時使用

```c
void dev_buzzer_beep(uint16_t ms)
{
    HAL_GPIO_WritePin(BUZZ_PORT, BUZZ_PIN, GPIO_PIN_SET);
    HAL_Delay(ms);
    HAL_GPIO_WritePin(BUZZ_PORT, BUZZ_PIN, GPIO_PIN_RESET);
}
```

> **動手練習**
>
> 為繼電器加上「最短吸合 100 ms」保護，避免 UART 雜訊造成繼電器顫動。再增加一條 `T` 指令：連續切換 LED 10 次，每次 80 ms，模擬「找到裝置」的視覺回饋——喚醒成功時會用到同樣的模式。

第 4 章

PWM、定時器與馬達 / 風扇

本章學習目標

學完本章你將能夠：

- 用 TIM PWM 產生可變佔空比，驅動 LED 亮度與直流風扇。
- 產生 50 Hz、1–2 ms 脈寬控制 SG90 伺服（開門 / 關門）。
- 解釋 ARR、PSC、CCR 三個暫存器如何決定頻率與佔空比。
- 把亮度與轉速限制在安全範圍，避免「口令說 1000%」寫爆比較值。

GPIO 只能全開全關。燈光調光、風扇轉速、舵機角度都需要**脈衝寬度調變**（PWM）：固定週期、改變高電平時間。STM32 的通用 / 進階計時器就是做這件事的硬體。

4.1 ARR、PSC、CCR

計時器時鐘 f_{tim} 先經預分頻器 PSC，再數到自動重載 ARR 就更新一次：

$$f_{\text{pwm}} = \frac{f_{\text{tim}}}{(PSC+1)(ARR+1)}$$

佔空比 $\approx \text{CCR}/(ARR+1)$。實務設定：

- **LED ／風扇**：20 kHz 左右，避開人耳可聽範圍；ARR 取 999，CCR 0–999 對應 0–100%。
- **SG90**：週期 20 ms（50 Hz）。0° 約 0.5–1 ms，180° 約 2–2.5 ms（以你手上的舵機規格為準）。

CubeMX：把某根支援 TIM 的腳設為 `TIM_CH1`，參數填 PSC / ARR，產生後 `HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1)`，再用 `__HAL_TIM_SET_COMPARE` 改 CCR。

4.2 直流風扇與 MOSFET

12 V 風扇由邏輯電平 N-MOSFET（例如 IRLZ44N 對 5 V 友善；3.3 V 請選邏輯閘極型號）低端開關。PWM 打在閘極，源極接地，汲極接風扇負端，風扇正端接 12 V。務必加飛輪二極體。

> **注意**　有些風扇在低佔空比時會 stalled（轉不動但耗電流）。軟體應設最低轉速（例如 25%），或先全開 300 ms 再降到目標值（soft-start）。

4.3 伺服：角度對應脈寬

程式碼 **4.1:** 脈寬微秒轉 CCR（50 Hz、1 MHz 計數為例）

```c
/* 1 MHz 計時 => 1 tick = 1 us，ARR = 19999 => 20 ms */
static uint16_t servo_us_to_ccr(uint16_t us)
{
    if (us < 500)  us = 500;
    if (us > 2500) us = 2500;
    return us; /* 因為 1 tick = 1 us */
}

void dev_door_set_angle(uint8_t deg)
{
    if (deg > 180) deg = 180;
    uint16_t us = (uint16_t)(500U + ((uint32_t)deg * 2000U) / 180U);
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, servo_us_to_ccr(us));
}
```

開門 / 關門不要瞬間對打：在任務裡用 20–40 ms 步進，或限制指令頻率。機構卡住時電流會飆，課程用 SG90 小負載即可，不要接真正的大門電機。

4.4 實驗：序列埠調光與開門

> **對應程式碼**
>
> `firmware/lab03_pwm_fan_motor/`。協定：`l 40` 亮度 40%、`f 70` 風扇 70%、`d 0/d 90` 門角度、`a` 全關。

上層仍走 `dev_*`。PWM 細節關在 `dev_pwm.c`。若沒有 12 V 電源，把風扇腳改接 LED，用亮度模擬轉速，邏輯完全相同。

程式碼 **4.2:** 限制百分比，避免 CCR 溢出

```c
void dev_set_percent(TIM_HandleTypeDef *htim, uint32_t ch,
                     uint16_t arr, uint8_t percent)
{
    if (percent > 100) percent = 100;
    uint32_t ccr = ((uint32_t)percent * arr) / 100U;
    __HAL_TIM_SET_COMPARE(htim, ch, ccr);
}
```

動手練習

實作「亮一點 / 暗一點」：每次加減 15%，飽和在 0 與 100。這正好對應語音指令 `CMD_LIGHT_UP` / `DOWN`。再量一次 PWM 腳的頻率，確認是 20 kHz 而不是 20 Hz（PSC 填錯一個零是經典事故）。

第 5 章

FreeRTOS 核心：任務、優先權與排程

本章學習目標

學完本章你將能夠：

- 說明搶佔式排程如何依優先權切換，以及時間片在同優先權下的角色。
- 建立週期任務與事件驅動任務，並為每個任務估算堆疊。
- 正確交接 SysTick，使用 `vTaskDelay` 而不是 `HAL_Delay`。
- 用 `uxTaskGetStackHighWaterMark` 證明堆疊沒有默默溢位。

從本章起，超級迴圈退役。系統變成一群「看起來同時在跑」的函式——任務（Task）。語音控制器至少需要音訊、辨識、設備、感測、通訊、監控六類任務，先用三個 LED 任務把肌肉練好。

5.1 任務是什麼

任務就是一個永不返回的 C 函式：

程式碼 5.1: 任務函式原型

```c
void vTaskDevice(void *arg)
{
    (void)arg;
    for (;;) {
        /* 等待命令、驅動 GPIO/PWM */
    }
}
```

每個任務有：優先權、堆疊、狀態（Running / Ready / Blocked / Suspended）。高優先權就緒時，低優先權立刻被搶佔。同優先權則依 `configTICK_RATE_HZ` 做時間片輪轉（需開啟 `configUSE_TIME_SLICING`）。

5.2 本專案的優先權表

數字越大越優先（FreeRTOS 慣例，與 NVIC 相反——這點務必記牢）。

表 5.1: 建議優先權（app_types.h）

任務	優先權	理由
Audio / ASR RX	5	避免 DMA / UART 環形緩衝溢位
Wake / ASR parse	4	推論或模組句柄要盡快消化
Device control	3	使用者可感知的動作
Comms	2	允許被控制打斷
Gateway（第 13 章）	2	助理指令與狀態回推，與 Comms 同級
Sensor	1	數秒級週期
Monitor	1	統計與餵狗，可與 Sensor 同級
Idle	0	核心自帶，可掛低功耗

注意 不要把所有任務都設成同一高優先權「以求保險」。那樣只是把超迴圈拆成幾個互相搶時間片的迴圈，失去搶佔的意義，而且堆疊浪費。

5.3 建立任務

CMSIS-RTOS v2 與原生 FreeRTOS API 都可以。本課程應用層用原生 API，語意較清楚；Cube 產生的 `MX_FREERTOS_Init` 可只保留 kernel 啟動。

程式碼 5.2: 建立三個週期任務

```
#define STK_SMALL  256

void app_os_start(void)
{
    xTaskCreate(vTaskBlink, "blink1", STK_SMALL, (void *)0, 1, NULL);
    xTaskCreate(vTaskBlink, "blink2", STK_SMALL, (void *)1, 1, NULL);
    xTaskCreate(vTaskBlink, "blink3", STK_SMALL, (void *)2, 2, NULL);
    vTaskStartScheduler();
}

static void vTaskBlink(void *arg)
{
    const uint32_t id = (uint32_t)arg;
    const TickType_t period[] = { pdMS_TO_TICKS(200),
                                  pdMS_TO_TICKS(400),
                                  pdMS_TO_TICKS(1000) };
    for (;;) {
        app_led_toggle((uint8_t)id);
        vTaskDelay(period[id]);
```

```
    }
}
```

第三個任務優先權較高，它就緒時另外兩個會被打斷——用邏輯分析儀看三根 LED，高優先權的邊緣會「插入」另外兩條波形。

5.4 延遲：相對 vs 絕對

`vTaskDelay(T)` 從「現在」再等 T。若任務本身執行超過預期，週期會漂。音訊統計、感測器取樣應使用 `vTaskDelayUntil`，鎖在絕對時間表上。

程式碼 5.3: 固定 2 秒採樣

```
void vTaskSensor(void *arg)
{
    TickType_t last = xTaskGetTickCount();
    (void)arg;
    for (;;) {
        sensor_sample();
        vTaskDelayUntil(&last, pdMS_TO_TICKS(2000));
    }
}
```

5.5 堆疊與 heap

`xTaskCreate` 的堆疊單位是 *Word*（Cortex-M 上 4 bytes）。256 words = 1 KB，對閃燈夠用；含 `snprintf` 的通訊任務建議 512–1024。路徑 B 的推論任務視模型而定，常要數 KB，且張量本身應在獨立緩衝。

Cube 的 `FreeRTOSConfig.h` 裡 `configTOTAL_HEAP_SIZE` 必須大於「所有任務堆疊 + 佇列 + 對齊損耗」。不夠時 `xTaskCreate` 回傳 `pdFAIL`——一定要檢查，否則系統少一個任務卻假裝正常。

> **提示** 在 Monitor 任務每 5 秒印 `uxTaskGetStackHighWaterMark` 與 `xPortGetFreeHeapSize()`。高水位接近 0 就是溢位前兆。這比等 HardFault 再查便宜得多。

5.6 實驗：三任務心跳

對應程式碼

`firmware/lab04_freertos_blinky/`。三顆 LED 不同週期；UART 每秒印一次任務名稱與堆疊高水位。確認 `HAL_Delay` 已從任務裡消失。

動手練習

把 Blink3 的優先權降到與另外兩個相同，再拉高。觀察 UART 時間戳與 LED 相位變化。另：故意把某任務堆疊改成 64 words 並在裡頭呼叫 `snprintf`，記錄 HardFault 或卡死現象，然後改回安全值——用身體記住堆疊溢位長什麼樣子。

第 6 章

Queue、Semaphore 與 Mutex

本章學習目標

學完本章你將能夠：

- 用 Queue 傳遞命令結構，避免任務之間共享可變全域狀態。
- 區分二進位信號量、計數信號量與 Mutex，並選對「DMA 完成」同步。
- 在 ISR 裡只呼叫 `*FromISR`，並正確處理 `pxHigherPriorityTaskWoken`。
- 解釋優先權反轉，以及 Mutex 的優先權繼承如何緩解它。

多任務一旦開始「順便改同一個 `g_light_on`」，bug 就會變成時序競賽。正確做法：**誰擁有資料，誰才改它；其他人傳訊息。**

6.1 Queue：把命令當成郵件

本專案的核心型別：

程式碼 6.1: 跨任務命令（與 `common/include/app_types.h` 一致）

```
typedef enum {
    CMD_NONE = 0,
    CMD_LIGHT_ON, CMD_LIGHT_OFF, CMD_LIGHT_UP, CMD_LIGHT_DOWN,
    CMD_FAN_ON,   CMD_FAN_OFF,   CMD_FAN_UP,   CMD_FAN_DOWN,
    CMD_DOOR_OPEN, CMD_DOOR_CLOSE,
    CMD_RELAY_ON,  CMD_RELAY_OFF,
    CMD_BUZZER_BEEP, CMD_BUZZER_OFF,
    CMD_QUERY_TEMP, CMD_ALL_OFF, CMD_ENTER_SLEEP,
    CMD_HOME_LEAVE,              /* 離家模式（第 10 章練習） */
    CMD_LIGHT_SET, CMD_FAN_SET   /* 絕對值，智慧助理用（第 13 章） */
} cmd_id_t;

typedef enum {                   /* 指令來源：稽核與安全政策用 */
    CMD_SRC_LOCAL = 0,           /* 路徑 A/B 本地語音 */
    CMD_SRC_GATEWAY,             /* 路徑 C 智慧助理（第 13 章） */
    CMD_SRC_DEBUG
} cmd_src_t;

typedef struct {
    cmd_id_t id;
```

```
    uint8_t  arg;      /* 可選參數，例如亮度絕對值 */
    uint8_t  src;      /* cmd_src_t；省略初始化時即為本地 */
    uint32_t tick;     /* 產生時間，便於逾時丟棄 */
} cmd_msg_t;
```

ASR 任務（或 UART 除錯任務）`xQueueSend`；Device 任務 `xQueueReceive`。`src` 標記指令從哪裡來——本地語音、智慧助理閘道或除錯口——第 13 章的安全政策與稽核 log 都靠它。佇列深度 8 通常夠用：語音指令是人類速度，不是每毫秒一筆。音訊幀則用另一條更深、元素更大的佇列，或改 ring buffer + 信號量（第 8 章）。

程式碼 6.2: Device 任務只認佇列

```
void vTaskDevice(void *arg)
{
    cmd_msg_t msg;
    (void)arg;
    for (;;) {
        if (xQueueReceive(g_cmd_q, &msg, portMAX_DELAY) == pdTRUE) {
            device_apply(&msg);
        }
    }
}
```

6.2 Semaphore：事件，不是資料

I2S DMA 半滿 / 全滿時，ISR 給二進位信號量，Audio 任務被喚醒去處理 *已經在靜態緩衝裡的*資料——不必把 320 bytes 複製進佇列。

程式碼 6.3: ISR 給信號量

```
void HAL_I2S_RxCpltCallback(I2S_HandleTypeDef *hi2s)
{
    BaseType_t woken = pdFALSE;
    (void)hi2s;
    xSemaphoreGiveFromISR(g_audio_sem, &woken);
    portYIELD_FROM_ISR(woken);
}
```

計數信號量適合「生產比消費快一點」的短突發；若計數一直漲，代表消費者太慢，應降取樣或升優先權，而不是把最大值設成 1000 假裝沒事。

6.3 Mutex：保護共享匯流排

I2C 上若掛了溫濕度與 OLED，兩個任務不能交錯發出起始條件。Mutex 與二進位信號量的差別：Mutex 有持有者、可優先權繼承，且*不可在 ISR 裡 Give/Take*。

程式碼 6.4: I2C 交易必須成對

```
void sensor_read_temp(float *out)
{
    xSemaphoreTake(g_i2c_mtx, portMAX_DELAY);
    dht_or_sht_read(out);
    xSemaphoreGive(g_i2c_mtx);
}
```

UART 輸出同樣需要鎖，或（較佳）讓唯一的 Log 任務從佇列取字串再送，其他任務只 `xQueueSend`。後者沒有「持鎖時被搶佔導致亂碼」的問題。

6.4 優先權反轉（課堂必考）

低優先權任務持有 I2C Mutex，高優先權任務也要 I2C，卻被一個*不需要 I2C* 的中優先權任務一直搶 CPU——高優先權等於被中優先權卡住。Mutex 的優先權繼承會把持鎖者暫時提升，縮短這個窗口。信號量沒有繼承，因此 I2C / SPI 這種臨界區請用 Mutex。

6.5 實驗：按鍵經佇列切設備

> **對應程式碼**
>
> `firmware/lab05_queue_semaphore/`。EXTI 按鍵 ISR 送 `CMD_LIGHT_ON/OFF` 進佇列；Device 任務執行。另用一個週期任務嘗試印 log，證明沒有 mutex / log 佇列時會交錯，加上之後就整齊。

程式碼 6.5: 按鍵 ISR 只丟訊息

```
void HAL_GPIO_EXTI_Callback(uint16_t pin)
{
    cmd_msg_t msg = { .id = CMD_LIGHT_ON, .arg = 0,
                      .tick = xTaskGetTickCountFromISR() };
    BaseType_t woken = pdFALSE;
    if (pin == USER_Btn_Pin) {
        xQueueSendFromISR(g_cmd_q, &msg, &woken);
    }
    portYIELD_FROM_ISR(woken);
}
```

動手練習

把佇列深度改成 1，在 ISR 裡連續模擬 5 次送出，觀察哪些命令被丟棄（檢查 `xQueueSendFromISR` 回傳值並點亮 LD3）。然後改回 8，並在 Device 任務人為 `vTaskDelay(500)`，再感受「深度用來吸收突發」。

第 7 章

Event Group、軟體定時器與系統同步

本章學習目標

學完本章你將能夠：

- 用 Event Group 廣播喚醒、故障、OTA、睡眠等系統級旗標。
- 設定單次與週期軟體定時器，取代任務裡的阻塞 `HAL_Delay`。
- 實作「喚醒後 5 秒聆聽窗，逾時回 idle」的產品行為。
- 讓 Monitor 任務依旗標決定是否餵看門狗。

Queue 適合「這筆資料給那一個消費者」。有些事情卻是*很多任務都要知道*：已經喚醒、匯流排故障、準備升級。Event Group 用一組位元表達這些事實。

7.1 系統事件位元

程式碼 **7.1:** event_bits.h

```
#define EVT_WAKE        (1U << 0)
#define EVT_LISTEN      (1U << 1)
#define EVT_FAULT       (1U << 2)
#define EVT_OTA         (1U << 3)
#define EVT_SLEEP_REQ   (1U << 4)
#define EVT_AUDIO_OK    (1U << 5)
#define EVT_HEALTHY     (1U << 6)  /* Monitor 判定健康（第 12 章） */
#define EVT_GW_DIRTY    (1U << 7)  /* 狀態已變，閘道需回推（第 13 章） */
```

ASR 任務辨識到喚醒詞：`xEventGroupSetBits(g_evt, EVT_WAKE | EVT_LISTEN)`。Device 任務可點亮「聆聽」LED；Monitor 可暫停進低功耗；5 秒後 Timer 回呼清掉 `EVT_LISTEN`。

等待時可指定「任一」或「全部」：

程式碼 **7.2:** 通訊任務等到喚醒或故障

```
EventBits_t b = xEventGroupWaitBits(
    g_evt, EVT_WAKE | EVT_FAULT,
    pdFALSE,   /* 不自動清除，讓其他人也能讀到 */
    pdFALSE,   /* 任一 bit 即可 */
    portMAX_DELAY);
if (b & EVT_FAULT) { comms_send_fault(); }
```

> **說明** Event Group 的位元數與 `TickType_t` 有關（24 或 8 個使用者位）。本專案用不到那麼多。清位元用 `xEventGroupClearBits`；從 ISR 設定要用 `xEventGroupSetBitsFromISR`（內部可能開 timer daemon 任務代勞）。

7.2 軟體定時器

FreeRTOS Timer 在 `Tmr Svc` 任務裡執行回呼——回呼裡不可阻塞。適合：關蜂鳴器、關聆聽窗、LED 呼吸燈下一步、週期餵狗。

程式碼 7.3: 聆聽窗 5 秒

```
static TimerHandle_t s_listen_tmr;

static void listen_timeout(TimerHandle_t t)
{
    (void)t;
    xEventGroupClearBits(g_evt, EVT_LISTEN);
    dev_light_listen_led(0);
}

void voice_on_wake(void)
{
    xEventGroupSetBits(g_evt, EVT_WAKE | EVT_LISTEN);
    dev_light_listen_led(1);
    xTimerReset(s_listen_tmr, 0);
}
```

建立時：

```
s_listen_tmr = xTimerCreate("listen", pdMS_TO_TICKS(5000),
                            pdFALSE, NULL, listen_timeout);
```

`pdFALSE` 表示單次。蜂鳴器 200 ms 同理：ON 之後 reset 一個 200 ms 單次 timer，回呼裡 OFF。Device 任務不再 `vTaskDelay(200)` 擋住後續命令。

7.3 與 Watchdog 的分工

第 12 章才接上真正的 IWDG，但協議現在就要定：Monitor 任務只有在 *沒有* `EVT_FAULT` 且各關鍵任務最近有「心跳」時才餵狗。心跳可用另一個 Event Group，或每任務更新 `g_heartbeat[id] = now`。Timer 週期餵狗是危險的——任務都死了 timer daemon 若還活著，狗會被騙。

7.4 實驗：喚醒窗與蜂鳴定時

對應程式碼

`firmware/lab06_event_group/`。序列埠送 `W` 模擬喚醒：聆聽 LED 亮 5 秒後自動熄。送 `B` 蜂鳴 200 ms（timer，非 delay）。送 `F` 置位 FAULT，Monitor 印警告且不再印「healthy」。

動手練習

為「全部關閉」加上 300 ms 的視覺確認：三顆 LED 同時閃一下。用一個單次 timer 實作，禁止在 Command 任務裡 delay。再讓 FAULT 發生時自動 `dev_all_off()`——安全策略應在同步層就存在，不要等 AI 章才補。

第 8 章

音訊擷取：麥克風、I2S 與 DMA

本章學習目標

學完本章你將能夠：

- 說明 PDM 與 I2S PCM 的差別，並為你的麥克風選對週邊。
- 設定 DMA 循環雙緩衝，在半滿 / 全滿回呼裡與 Audio 任務同步。
- 計算 16 kHz / 16-bit / mono 的頻寬與 10 ms 幀大小。
- 在路徑 A 用 UART 接收外部模組的「已辨識結果」，仍維持同樣的任務邊界。

語音系統的物理起點是空氣壓力變化。STM32 不會「聽懂」，它只會把電壓或數位串流變成記憶體裡的整數陣列。本章把這條管線做穩——穩了，後面的 AI 才有乾淨的輸入。

8.1 取樣參數

教學預設：**16 kHz、16-bit、單聲道**。一幀 10 ms 含

$$16000 \times 0.010 \times 2 = 320 \text{ bytes} = 160 \text{ samples}.$$

雙緩衝就是兩個 320 bytes（或更大，例如 20 ms）。CPU 必須在下一半被覆寫前處理完當前半——這就是 Audio 任務高優先權的理由。

8.2 麥克風選型

表 8.1: 課堂常見麥克風

型號	介面	注意
INMP441 / ICS-43434	I2S，直接出 PCM	3.3 V、WS/SCK/SD，對 F4 友善
MP34DT01（Discovery）	PDM	需 SAI/I2S PDM 濾波或軟體 CIC
MAX4466 等類比	ADC + 定時器觸發 DMA	簡單但雜訊與偏壓要處理
路徑 A 模組自帶麥	模組內部完成	STM32 只收 UART 文字 / 代碼

路徑 B 建議 INMP441：少一層 PDM 轉換，課堂時間夠用。接線時 WS（LRCLK）決定左右聲道，單顆麥通常把 L/R 腳拉到固定電位。

8.3 I2S + DMA 循環模式

CubeMX：

1. 開啟 I2S2（或 SAI），Master Receive，Philips 標準，16-bit，16 kHz。確認 PLLI2S 精準。
2. DMA 設 Circular、Half Word、中等優先權。先開 **DMA** 再開 **I2S**。
3. NVIC：DMA 中斷優先權必須可呼叫 FreeRTOS API。

程式碼 **8.1:** 雙緩衝與 DMA 啟動

```
#define FRAME_SAMP  160
static int16_t s_pcm[2][FRAME_SAMP];

void audio_start(void)
{
    if (HAL_I2S_Receive_DMA(&hi2s2, (uint16_t *)s_pcm[0],
                            FRAME_SAMP * 2U) != HAL_OK) {
        xEventGroupSetBits(g_evt, EVT_FAULT);
    }
}
```

半滿對應 `s_pcm[0]`，全滿對應 `s_pcm[1]`。回呼裡只記錄「哪一銀行效」並 `xSemaphoreGiveFromISR`，不要在 ISR 做 FFT。

程式碼 **8.2:** Audio 任務

```
void vTaskAudio(void *arg)
{
    (void)arg;
    audio_start();
    for (;;) {
        xSemaphoreTake(g_audio_sem, portMAX_DELAY);
        const int16_t *src = audio_bank_ptr(); /* 0 或 1 */
        xQueueSend(g_audio_q, src, 0);         /* 滿則丟最舊策略見完整檔 */
        heartbeat_touch(TASK_AUDIO);
    }
}
```

佇列元素若是整幀，複製 320 bytes 尚可接受。更省的做法是佇列只傳 bank 索引，消費端必須在下一幀到來前用完——延遲約束更硬。

8.4 路徑 A：UART 幀當「音訊」

外部模組通常輸出像 `OK wakeup` 或二進位指令碼。STM32 側：

- USART RX DMA 環形緩衝，或 byte-IT + 組包狀態機。
- 以換行或固定長度切幀，組成 `asr_raw_t` 丟給 ASR 任務。
- 不要在 UART ISR 裡 `strcmp`。

任務圖與路徑 B 相同：只是 Audio 任務的輸入從 I2S 換成 UART。這讓第 9 章可以共用 Command 以後的所有程式。

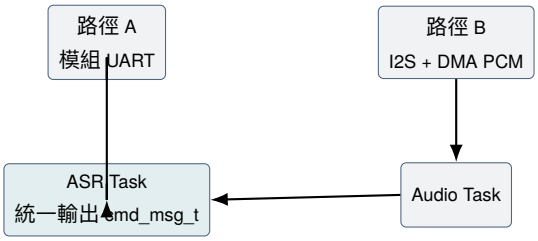


圖 8.1: 兩條擷取路徑在 ASR 任務匯合

8.5 簡易能量偵測

即使走路徑 B，在跑 NN 前先算幀能量，可過濾靜音、省算力：

程式碼 8.3: RMS 能量

```c
uint32_t pcm_energy(const int16_t *x, size_t n)
{
    uint64_t acc = 0;
    for (size_t i = 0; i < n; i++) {
        int32_t v = x[i];
        acc += (uint64_t)(v * v);
    }
    return (uint32_t)(acc / n);
}
```

能量低於門檻就不必喚醒後續 KWS。門檻需依麥克風與房間校正——印出能量值 10 秒是值得的課堂活動。

8.6 實驗

> **對應程式碼**
>
> `firmware/lab07_audio_capture/` 含兩份實作：`audio_i2s.c`（路徑 B）與 `audio_uart_mod.c`（路徑 A）。預設宏 `APP_AUDIO_PATH` 切換。路徑 A 實驗：模組或 PC 終端送一行文字，板子回顯能量或字串長度。路徑 B：UART 印每幀 RMS，對麥說話應看到數字跳升。

注意　H7 開啟 D-Cache 時，DMA 寫入的 PCM 對 CPU 可能是舊資料。必須把緩衝放進 non-cacheable 區，或在使用前 `SCB_InvalidateDCache_by_Addr`。F4 無 D-Cache，較少踩這坑。

動手練習

把幀長改為 20 ms，重算陣列大小與 I2S DMA 長度（長度單位是取樣數不是 byte，HAL 文件要看清楚）。量 DMA 中斷間隔是否變成 20 ms。再故意把 Audio 任務優先權降到 0，對麥吹氣，觀察能量是否出現「一段正常、一段卡住」——那就是即時性被破壞的樣子。

第 9 章

喚醒詞與語音辨識

本章學習目標

學完本章你將能夠：

- 區分 KWS（關鍵詞偵測）與 ASR（指令 / 連續語音）的責任。
- 完成路徑 A：解析模組輸出並對照指令表。
- 描述路徑 B：特徵（MFCC / Log-Mel）→ 量化模型 → X-CUBE-AI。
- 設計拒絕機制：低信心、未知詞、聆聽窗外的指令一律丟棄。

「語音控制」在工程上是兩段：先判斷有沒有人在叫我（喚醒），再判斷叫我做什麼（指令）。兩段都允許誤拒（沒反應），但要極力避免誤接受（自己開燈）。

9.1 狀態：Idle → Listen → Execute



圖 9.1: 語音對話的最小狀態機

Idle 只跑 KWS（或等模組的喚醒碼）。Listen 才允許指令表。Execute 把 `cmd_msg_t` 送進第 6 章的佇列，然後回 Idle。Event Group 的 `EVT_LISTEN` 與這個狀態對齊。

9.2 路徑 A：外部模組

以常見離線模組為例（指令碼請換成你模組的手冊值）：

程式碼 9.1: 字串表對應

```
typedef struct {
    const char *phrase;
    cmd_id_t    cmd;
} asr_map_t;

static const asr_map_t k_map[] = {
    { "xiao-ke",       CMD_NONE },      /* 喚醒，不進設備佇列 */
```

```
    { "kai-deng",      CMD_LIGHT_ON },
    { "guan-deng",     CMD_LIGHT_OFF },
    { "liang-yi-dian", CMD_LIGHT_UP },
    { "kai-feng-shan", CMD_FAN_ON },
    { "kai-men",       CMD_DOOR_OPEN },
    { "quan-bu-guan",  CMD_ALL_OFF },
    { "shui-jiao",     CMD_ENTER_SLEEP },
};
```

模組 UART 一行進來後，去掉 `\r\n`，線性搜尋。喚醒詞只呼叫 `voice_on_wake()`。其他詞若未處於 `EVT_LISTEN`，記錄一次 log 並丟棄——這能大幅降低電視聲音誤觸。

> **提示**　很多模組可自訂 3–5 字中文詞。課堂請避開單音節（「開」「關」），改用「打開電燈」「關閉風扇」。詞愈長，假陽性愈低。

9.3 路徑 B：板上 Edge AI

典型管線：

1. 每 10–20 ms 取一幀 PCM，重疊 50%。
2. 預強調、分幀、加窗、FFT，得到 Log-Mel 或 MFCC（約 40 維 × 49 幀構成一張「圖」）。
3. 量化成 int8，送入 X-CUBE-AI 產生的 `ai_run()`。
4. Softmax / argmax 得到類別與分數；低於門檻視為 silence/unknown。

Cube.AI 流程摘要：在 STM32CubeMX 安裝 X-CUBE-AI 套件 → 匯入 Keras / ONNX / TFLite → 看 Flash / RAM 佔用與 MACC → 產生 `network.c` 與分析報告。官方音訊範例（如 FP-AI-NANOEDG1、X-CUBE-AI 的 KWS 示範，以及 STM32H747 發現板的音訊 AI 應用）可直接對照特徵參數，避免自己從零調 Mel 濾波器。

> **說明**　F429 的 SRAM 能跑極小的 DS-CNN，但餘裕不大。課程把路徑 B 的完整模型部署放在 H7 發現板；F4 上可用「能量 + 極簡單的模板比對」做概念驗證，或只跑官方縮小模型。不要為了在 F4 塞進大模型而關掉所有安全檢查。

推論必須在 ASR 任務中執行，且該任務堆疊與模型活化緩衝分開。`ai_run` 期間 Device 任務仍可跑——這是 RTOS 相對超迴圈的具體好處。

9.4 拒絕與防抖

- **信心門檻**：例如 0.70。印出分數一週，再決定數字。
- **時間防抖**：同一指令 400 ms 內不重複執行。

- **未知類**：模型應含 `unknown` / `silence` 類，不要強迫 argmax 總是對應一個家電。
- **聆聽窗**：窗外只接受喚醒。

程式碼 9.2: 送出前的守門員

```
BaseType_t asr_try_dispatch(cmd_id_t cmd, uint8_t score)
{
    EventBits_t e = xEventGroupGetBits(g_evt);
    if (cmd == CMD_NONE) {
        voice_on_wake();
        return pdTRUE;
    }
    if ((e & EVT_LISTEN) == 0) return pdFALSE;
    if (score < APP_MIN_SCORE) return pdFALSE;
    cmd_msg_t m = { .id = cmd, .arg = score,
                    .tick = xTaskGetTickCount() };
    return xQueueSend(g_cmd_q, &m, 0);
}
```

9.5 實驗

> **對應程式碼**
>
> `firmware/lab08_asr_module/` 實作路徑 A 完整對照表與聆聽窗。`firmware/lab08_asr_edgeai/` 提供特徵 + 假模型（argmax 模板）與 X-CUBE-AI 接線註解：你把 Cube.AI 產生檔丟進專案後，把 `kws_infer()` 換成 `ai_run()` 即可。

用 PC 終端模擬模組，依序輸入：

▶ 執行結果

```
xiao-ke
kai-deng
guan-deng
```

第一行應只亮聆聽 LED；第二行燈才亮；若省略喚醒直接 `kai-deng`，燈不應動作。

> **動手練習**
>
> 加入「連續兩次未知詞就退出聆聽窗」。再為喚醒加上語音回饋：成功喚醒時 `dev_buzzer_beep(80)` 一次，讓使用者不必看 LED。最後讀一篇 ST 官方 KWS 應用筆記，把你的 Mel 參數（窗長、hop、濾波器數）抄進筆記，和下週狀態機對接。

第 10 章

指令解析與設備控制狀態機

本章學習目標

學完本章你將能夠：

- 把「命令列舉」轉成每一個設備的內部狀態（亮度、轉速、門角）。
- 處理相對指令（亮一點）與絕對指令（亮度 40）而不互相打架。
- 實作 ALL_OFF 與故障時的安全降級。
- 讓查詢類指令走通訊佇列，而不是在 Device 任務裡直接印溫度。

到這裡，語音管線已經會吐出 cmd_msg_t。本章是產品的「手」：所有副作用都發生在 Device 任務，而且是可預測、可測試的。

10.1 每個設備一份狀態

程式碼 **10.1:** device_state.h

```
typedef struct {
    uint8_t light_on;
    uint8_t light_pct;   /* 0--100，on=0 時仍記住上次亮度 */
    uint8_t fan_on;
    uint8_t fan_pct;
    uint8_t door_open;   /* 0 關 1 開 */
    uint8_t relay_on;
    uint8_t buzzer_on;
} home_state_t;
```

相對指令改的是這份結構，再呼叫 dev_* 反映到硬體。重啟後的預設：全關、亮度 60%、風扇 40%、門關閉——寫在 device_reset()，OTA 或看門狗復位後行為一致。

10.2 套用命令

程式碼 **10.2:** device_apply（純邏輯，與 common/src/device_fsm.c 一致）

```
void device_apply(home_state_t *s, const cmd_msg_t *m)
{
    switch (m->id) {
```

```
    case CMD_LIGHT_ON:
        s->light_on = 1;
        if (s->light_pct < 10) s->light_pct = 60;
        break;
    case CMD_LIGHT_UP:                  /* 相對：「亮一點」 */
        s->light_on = 1;
        s->light_pct = (uint8_t)MIN(100, s->light_pct + 15);
        break;
    case CMD_LIGHT_SET:                 /* 絕對：「亮度 40」、助理滑桿 */
        s->light_pct = (m->arg > 100) ? 100 : m->arg;
        s->light_on = (s->light_pct > 0);
        break;
    case CMD_ALL_OFF:
    case CMD_ENTER_SLEEP:
        device_reset_outputs(s);        /* 保留 pct 記憶，只關輸出 */
        break;
    case CMD_QUERY_TEMP:                /* 不改 actuator */
    default:
        break;
    }
}
```

注意這個函式沒有副作用：不碰 HAL、不發事件。副作用留在任務層——vTaskDevice 取出訊息後先驗逾時，呼叫 device_apply 改狀態，再 device_commit 寫入 PWM / GPIO；QUERY_TEMP 由任務轉發遙測請求、ENTER_SLEEP 由任務置位 EVT_SLEEP_REQ。相對（UP / DOWN）與絕對（SET）指令都收斂到同一份 home_state_t，自然不會互相打架。

門的開與關寫成角度常數（例如 90 與 0），不要讓語音直接傳任意角度，以免機構撞到底。

10.3 逾時命令

若 ASR 佇列短暫堵塞，一則 2 秒前的「開門」不應突然執行。cmd_msg_t.tick 由 vTaskDevice 取出訊息後、呼叫 apply 之前檢查：

```
if ((xTaskGetTickCount() - m->tick) > pdMS_TO_TICKS(1500)) {
    app_log("drop stale cmd\r\n");
    return;
}
```

10.4 可單測的邏輯

device_apply 不要呼叫 HAL：它只改傳入的 home_state_t，device_commit 才碰硬體。這樣可以在 PC 上寫一組單元測試（本倉庫 firmware/lab09_voice_controller/host_test/）驗證

「亮一點三次不會超過 100」「風扇絕對值 10% 會被抬到最低轉速 25%」。嵌入式課程很少做這件事，但狀態機是最值得測的一層——第 13 章的智慧助理指令能直接沿用，也是因為這層早就被測過。

10.5 實驗：口令驅動全家電

> **對應程式碼**
>
> `firmware/lab09_voice_controller/` 把 lab02–08 接起來：UART 模擬 ASR（或接真模組），Device 任務維持 `g_home`，每次變更印一行 JSON 方便你對照。

建議驗收腳本（口說或終端）：

1. 喚醒 → 開燈 → 亮一點 → 暗一點。
2. 開風扇 → 開門 → 開插座。
3. 全部關閉：燈、風扇、繼電器、蜂鳴必須全停，門回關閉角。
4. 不喚醒直接「開燈」：無動作。
5. 喚醒後發呆 5 秒再「開燈」：無動作。

> **動手練習**
>
> 增加「離家模式」：一個命令依序關燈、關風扇、關繼電器、關門，每步間隔 150 ms（用 timer 或 Device 任務內的短狀態機，不要一次 blocking 600 ms 若不需要）。思考：若中途收到「開燈」，應中斷離家還是排隊？把你的選擇寫進註解——這就是產品規格。

第 11 章

通訊協定：UART、JSON 與 MQTT

本章學習目標

學完本章你將能夠：

- 設計一組穩定的除錯 / 遙測 UART 協定，與 ASR 模組口分開。
- 產出最小化 JSON，讓 PC 或樹莓派輕鬆接。
- 描述 MQTT 主題規劃，以及 STM32 經 ESP32 AT 或 enticed 乙太網路的接法。
- 避免在通訊任務裡阻塞到讓看門狗復位。

產品不能只活在麵包板上。工廠測試、App 顯示、雲端紀錄都需要一條 機器可讀的通道。語音指令走內部佇列；對外只報「現在狀態」與「事件」。

11.1 兩條 UART 的理由

口	用途	Nucleo-F429ZI 建議
USART3	人類除錯 + JSON 遙測（VCP）	ST-LINK 虛擬 COM
USART6	路徑 A 語音模組	獨立 TX/RX、3.3 V

混在同一條線的後果是：你 `printf` 的 log 被模組當成指令，或模組的 `OK` 被你的解析器吃掉。若腳位不夠，至少用不同的幀頭（例如遙測固定以 `{` 開頭，模組用純文字詞）。

11.2 JSON 最小集

不要引完整 JSON 函式庫除非必要。手寫幾個 `snprintf` 即可：

程式碼 11.1: 狀態上報

```
int telemetry_format(char *buf, size_t n, const home_state_t *s,
                     float temp_c)
{
    return snprintf(buf, n,
        "{\"light\":%u,\"l_pct\":%u,\"fan\":%u,\"f_pct\":%u,"
        "\"door\":%u,\"relay\":%u,\"temp\":%.1f,\"tick\":%lu}\r\n",
        s->light_on, s->light_pct, s->fan_on, s->fan_pct,
        s->door_open, s->relay_on, (double)temp_c,
        (unsigned long)xTaskGetTickCount());
```

```
}
```

事件另用小物件：`{"evt":"wake"}`、`{"evt":"fault","code":3}`。PC 端用 Python `json.loads` 即可畫即時儀表。

11.3 通訊任務結構

程式碼 **11.2:** vTaskComms 骨架

```
void vTaskComms(void *arg)
{
    comms_req_t req;
    (void)arg;
    for (;;) {
        if (xQueueReceive(g_comms_q, &req, pdMS_TO_TICKS(1000)) == pdTRUE) {
            comms_handle(&req);
        } else {
            comms_send_telemetry(); /* 週期心跳 */
        }
        heartbeat_touch(TASK_COMMS);
    }
}
```

`CMD_QUERY_TEMP` 只是往 `g_comms_q` 丟一筆 `REQ_TELEMETRY`。Device 任務不讀 DHT22——感測屬於 Sensor 任務，溫度放在有 mutex 保護的 `g_sensor` 快取。

11.4 MQTT（選做）

課堂若要連家用 MQTT：

- 主題：`home/living/voice/state`（上報）、`home/living/voice/cmd`（下行，用第 13 章的閘道文字行，例如 `CMD LIGHT ON`）。
- 硬體：ESP32 跑 AT 或一個極小 MQTT client；STM32 只出 UART 字串。現成的 MQTT–UART 透明橋在 `lab12_assistant_gateway/esp32/mqtt_uart_bridge/`。或 F429 的 ETH + lwIP + coreMQTT（工作量明顯較大）。
- **QoS**：狀態用 QoS0 即可；OTA 觸發用 QoS1。
- 下行命令必須走同一個 `g_cmd_q` 與安全策略（遠端指令不套本地聆聽窗，但門鎖等高風險輸出受政策管制，見第 13 章），不能讓雲端繞過狀態機直接 `dev_relay_set(1)`。

ST 的 X-CUBE-CELLULAR / Azure 套件與 Mosquitto 教學很多，本課程只要求主題與安全邊界正確，不綁定某一家雲。

11.5 實驗

對應程式碼

`firmware/lab10_comms/`：每秒一筆 JSON；按鍵或口令變更立即多一筆 `evt`。附 `tools/serial_plot.py` 讀溫度與亮度。選做檔 `mqtt_notes.md` 寫 ESP32 接線。

動手練習

從 PC 送一行 `{"cmd":"CMD_LIGHT_ON"}` 進 VCP，通訊任務解析後送入 `g_cmd_q`（僅當 `APP_ALLOW_UART_CMD=1`）。思考實驗室環境為什麼預設應關掉這個後門。

第 12 章

產品化：異常處理、Watchdog、低功耗與 OTA

本章學習目標

學完本章你將能夠：

- 替關鍵任務做心跳，用獨立看門狗復位「沉默的死亡」。
- 分清 HardFault、assert、業務 FAULT，並留下復位原因。
- 在閒置時進入 Stop 模式，用 RTC 或 EXTI（含 UART ／麥檢測）喚醒。
- 說明雙槽（A/B）OTA 的最小可靠流程，並完成 UART 模擬升級。

實驗室 demo 與產品的差距，通常不在辨識率，而在「半夜死機沒人按重置」「電池兩天就沒電」「升級磚機」。本章把這三件事做到最小可用。

12.1 分層的失敗

表 12.1: 失敗要上報到哪一層

層級	例子與動作
可恢復	I2C NACK 一次：重試，三次才 `EVT_FAULT`
降級	麥克風 DMA 失敗：關語音，設備仍可用按鍵／ UART
安全態	未知嚴重錯誤：`dev_all_off` + 故障 LED
復位	心跳停止、IWDG 逾時：整系統重來，並記錄原因

`configASSERT` 在開發期應開啟。進產線可改成記錄後復位，但不要變成空巨集——靜默略過不變量會讓現場無法除錯。

12.2 獨立看門狗

IWDG 掛 LSI，逾時公式見參考手冊。建議 2 s。Monitor 任務每 200 ms 檢查：

程式碼 12.1: 只有大家都活著才餵狗

```
void vTaskMonitor(void *arg)
{
    (void)arg;
    for (;;) {
```

```
        if (heartbeat_all_fresh(pdMS_TO_TICKS(1500)) &&
            (xEventGroupGetBits(g_evt) & EVT_FAULT) == 0) {
            HAL_IWDG_Refresh(&hiwdg);
        }
        vTaskDelay(pdMS_TO_TICKS(200));
    }
}
```

CubeMX 打開 IWDG 後，若你停在斷點超過逾時，板子會自己復位——除錯時可先不啟動 IWDG，或調長逾時。**不要用軟體定時器無條件 Refresh**。

復位後讀 `RCC->CSR` 的 IWDG RST 旗標，UART 印 `boot reason: iwdg`，再清除旗標。這是現場維修的第一句話。

12.3 HardFault 最小處理

在 `HardFault_Handler` 裡不要 `printf`（可能已壞）。可點亮紅 LED、寫 RTC backup register 一個魔術數，然後 `NVIC_SystemReset()`。開發期用 debugger 看堆疊裡的 PC，對照 `.map`。

12.4 低功耗

語音產品多數時間在 Idle。策略：

1. 無 `EVT_LISTEN`、無通訊排程、無 OTA 時，允許睡眠。
2. Idle Hook 或專用任務呼叫 `HAL_PWR_EnterSTOPMode`。
3. 喚醒源：使用者按鍵 EXTI、RTC 週期（更新感測）、UART RX（模組喚醒碼）、或 I2S DMA（路徑 B 通常不能深睡，可改 Standby 只留模組供電）。
4. 離開 Stop 後必須重新設定時鐘（Cube 的 `HAL_PWR_EnterSTOPMode` 範例有 `SystemClock_Config`）。

路徑 A 更適合低功耗：外部模組可自行做低功耗聆聽，STM32 睡死，模組拉一根 WAKE 腳。這是選模組時的重要規格。

12.5 OTA 最小可靠集

磚機通常發生在「只擦了自己還在跑的那塊 Flash」。雙槽規則：

1. Slot A 跑目前版本；映像寫入 Slot B。
2. 寫完算 CRC32 / SHA-256，與標頭核對。
3. 標記 B 為 pending，復位。Bootloader 跳 B。
4. 新韌體在 30 s 內回報 healthy（餵狗 + 特定旗標），再把 B 標為 valid。否則下次開機回 A。

課堂用 UART YMODEM 或自訂 1 KB 封包把測試映像寫進 Bank B（F429 雙 Bank Flash 正好適合）。完整安全啟動（簽章）超出時數，但標頭應預留簽名欄位，避免日後格式不相容。

程式碼 12.2: 映像標頭

```
typedef struct {
    uint32_t magic;     /* 'VHOM' */
    uint32_t version;
    uint32_t size;
    uint32_t crc32;
    uint32_t flags;     /* pending / valid */
} ota_hdr_t;
```

> **注意**　OTA 期間關閉語音與繼電器輸出。升級到一半掉電靠雙槽與 CRC，不靠祈禱。課堂禁止對 ST-LINK 所在 Bank 做晶片全面抹除的「假 OTA」。

12.6 最終專案驗收

> **對應程式碼**
>
> `firmware/lab11_watchdog_lpm_ota/` 示範 IWDG、睡眠與 UART 假升級（寫 backup register 當版本號）。`firmware/voice_home_final/` 是課程總成：任務表、指令表、遙測、心跳、安全關斷全部打開。

展示當天請準備 3 分鐘口頭說明：架構圖、一條指令的資料路徑、一次你修過的 bug。評分建議：硬體穩定 20%、RTOS 正確性 25%、語音體驗 25%、產品化機制 20%、程式可讀與抽象 10%。

> **動手練習**
>
> 故意在 Device 任務寫空的忙碌迴圈卡死，證實 IWDG 復位且 boot reason 正確。再做一次「升級成功但 CRC 錯」應回退的演練（可只改 flag）。最後量 Idle 與 Stop 的電流，把數字寫進報告——沒有數字的低功耗等於沒做。

你走到哪了

從時鐘樹到看門狗，學員已經走完一條接近量產的嵌入式 AI 路徑。接下來可以換更好的麥克風陣列、加聲源定位、把模型換成 ST 最新的 NPU 系列，或把 MQTT 接到真正的家用 Hub。骨架不必重寫——這正是本課程把 `cmd_msg_t` 與 `dev_*` 守得那麼死的原因。下一章立刻驗證這件事：同一台裝置，原封不動交給 Siri、Alexa 與 OK Google。

第 13 章

智慧助理整合：Siri、Alexa 與 OK Google

本章學習目標

學完本章你將能夠：

- 說明「路徑 C」與路徑 A / B 的差別：喚醒與辨識由手機或音箱代勞，STM32 只收已驗證的指令。
- 在 STM32 實作閘道協定（文字行下行、JSON 狀態上行），且所有外部指令仍走同一條 `g_cmd_q` 與安全策略。
- 用 HomeSpan 讓 Siri 控制燈、風扇與插座，並讓家庭 App 與裝置狀態雙向同步。
- 用 fauxmoESP 讓 Alexa 在區網內直接發現裝置；說明 OK Google 為何建議走 Matter 或 Home Assistant。
- 比較 ESP32 直橋、Home Assistant 樞紐、Matter 三種整合策略的取捨。

前十二章做出來的產品有自己的耳朵。本章給它第二種入口：使用者對手機說「嘿 Siri，打開客廳燈」，或對音箱說「Alexa, turn on the light」、「OK Google，關風扇」，指令穿過各家生態系，最後仍落進我們在第 6 章建好的那條指令佇列。

13.1 路徑 C：辨識搬到生態系那一側

路徑 A / B 的喚醒詞與 ASR 都在裝置端；接上 Siri、Alexa、Google 後，這兩件事完全由手機或智慧音箱完成，STM32 收到的是「生態系已驗證過的結構化指令」。差異整理如下：

表 13.1: 三條語音路徑

	路徑 A（模組）	路徑 B（Edge AI）	路徑 C（助理）
喚醒 + ASR	外部模組	STM32 上的 KWS	手機 / 音箱 / 雲端
需要網路	否	否	是（配網一次）
聆聽窗策略	有（5 秒）	有	不適用（生態系已驗證）
STM32 收到	UART 詞句	類別 + 分數	閘道文字行指令
新增程式	ASR 任務	特徵 + 推論	Gateway 任務

三條路徑在 `g_cmd_q` 匯合。為了讓 Device 任務知道指令從哪來，`cmd_msg_t` 新增了 `src` 欄位（`CMD_SRC_LOCAL` / `CMD_SRC_GATEWAY`），既有程式不受影響——省略欄位的舊初始化預設就是本地來源。

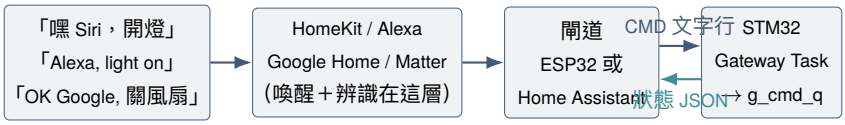


圖 13.1: 路徑 C：辨識在生態系，控制回到同一條指令佇列

13.2 三種整合策略

表 13.2: 把裝置接上三家助理的三種做法

策略	怎麼接	適合
ESP32 直橋	HomeSpan（Siri）＋ fauxmoESP（Alexa）各一份草稿，UART 接 STM32	課堂 demo 最快；無需伺服器
Home Assistant 樞紐	STM32 經 MQTT（第 11 章）→ HA → HomeKit Bridge / Alexa / Google 整合	一台樹莓派同時涵蓋三家
Matter 統一標準	ESP32 跑 esp-matter，三家 App 直接配對同一裝置	產品級；一次認證三家通吃

課堂主線用 **ESP32 直橋**（不需要額外伺服器），並示範 Home Assistant 樞紐；Matter 以導讀方式帶過，作為商品化方向。無論哪一條，STM32 端的程式 完全相同——這正是第 3 章開始堅持設備抽象與指令佇列的回報。

13.3 閘道協定：STM32 端

閘道與 STM32 之間走 115200 UART（預設 USART6，見 `pin_map.h` 的 `GW_UART`；若同時使用路徑 A 模組，改接 USART2 等閒置埠）。協定刻意簡單：下行是空白分隔的文字行，上行是單行 JSON。

表 13.3: 閘道協定

方向	訊息	意義
下行	`CMD LIGHT ON / OFF`	開關燈
下行	`CMD LIGHT PCT 70`	絕對亮度（`CMD_LIGHT_SET`）
下行	`CMD FAN ON / OFF / PCT n`	風扇
下行	`CMD DOOR OPEN / CLOSE`	門（受策略管制）
下行	`CMD RELAY ON / OFF`	插座
下行	`CMD ALL OFF`、`CMD BEEP`	全關、蜂鳴
下行	`GET STATE`	請求立即回報狀態
下行	`PING`	連線心跳（回 `PONG`）
上行	`{"light":1,"l_pct":70,...}`	狀態改變、被查詢或每 5 秒

選文字行而不是 JSON 下行，是為了讓三種閘道（HomeSpan、fauxmo、MQTT 橋）都能一行 `printf` 出指令，STM32 也只需 `sscanf`。解析器獨立成 `gw_proto.c`，不含任何 HAL，可在 PC 上單元測試：

程式碼 13.1: gw_proto.c 解析核心（節錄）

```
gw_line_t gw_parse_line(const char *line, cmd_msg_t *out)
{
    char dev[12] = {0}, act[12] = {0};
    int val = -1;

    out->id  = CMD_NONE;
    out->src = CMD_SRC_GATEWAY;

    if (strcmp(line, "PING") == 0)      return GW_LINE_PING;
    if (strcmp(line, "GET STATE") == 0) return GW_LINE_GET_STATE;

    if (sscanf(line, "CMD %11s PCT %d", dev, &val) == 2) {
        out->arg = (uint8_t)((val < 0) ? 0 : (val > 100) ? 100 : val);
        if (strcmp(dev, "LIGHT") == 0) { out->id = CMD_LIGHT_SET; return GW_LINE_CMD; }
        if (strcmp(dev, "FAN")   == 0) { out->id = CMD_FAN_SET;   return GW_LINE_CMD; }
        return GW_LINE_UNKNOWN;
    }
    if (sscanf(line, "CMD %11s %11s", dev, act) == 2) {
        /* 查 dev×act 對照表，見完整檔 */
    }
    return GW_LINE_UNKNOWN;
}
```

Gateway 任務的工作只有三件：收行、過策略、回狀態。狀態回推靠 Event Group 新增的 `EVT_GW_DIRTY` 位——Device 任務每次 commit 後置位，Gateway 任務看到就推一行 JSON，另外每 5 秒也推一次當保險：

程式碼 13.2: Gateway 任務（節錄）

```
static void vTaskGateway(void *arg)
{
    char line[GW_LINE_MAX];
    int n = 0;
    uint8_t ch;
    (void)arg;
    gw_push_state();

    for (;;) {
        if (xEventGroupGetBits(g_evt) & EVT_GW_DIRTY) {
            xEventGroupClearBits(g_evt, EVT_GW_DIRTY);
            gw_push_state();          /* 本地口令改了狀態，App 也要知道 */
```

```
        }
        if (HAL_UART_Receive(&GW_UART, &ch, 1, 20) != HAL_OK)
            continue;
        /* ... 組行，略 ... */
        cmd_msg_t m;
        switch (gw_parse_line(line, &m)) {
        case GW_LINE_PING:      gw_send_line("PONG");  break;
        case GW_LINE_GET_STATE: gw_push_state();       break;
        case GW_LINE_CMD:
            if (!gw_policy_ok(&m)) break;   /* 例：門未開放遠端 */
            m.tick = xTaskGetTickCount();
            (void)xQueueSend(g_cmd_q, &m, 0);
            break;
        default: break;
        }
    }
}
```

注意　遠端指令不套用本地聆聽窗（生態系已做過喚醒與帳號驗證），但門、鎖、插座這類高風險輸出要另立政策：教材預設 `APP_GW_ALLOW_DOOR=1` 方便 demo，產品請改為 0 並要求 App 端二次確認。HomeKit 對 Lock 類別原生要求解鎖確認、Alexa 例程可設 PIN——善用生態系的保護，而不是繞過它。無論來源為何，指令仍經過過期檢查與去抖，繞過 `g_cmd_q` 直接呼叫 `dev_*` 依然是禁手。

13.4　Siri：HomeKit 與 HomeSpan

HomeKit 的裝置模型是三層：**Accessory**（一台裝置）擁有若干 **Service**（功能，例如燈泡），Service 擁有若干 **Characteristic**（可讀寫的屬性，例如開關、亮度）。我們的映射：

設備	Service	Characteristic
燈	LightBulb	On、Brightness（0–100）
風扇	Fan	Active、RotationSpeed
插座	Outlet	On、OutletInUse
門	教學用 Switch	On（正式品用 LockMechanism）

HomeSpan 是 ESP32 上的開源 HAP 實作：繼承 Service 類別、覆寫 `update()` 收 Siri 指令、用 `setVal()` 回推狀態即可。

程式碼 **13.3:** homespan_siri.ino（節錄）

```
struct GwLight : Service::LightBulb {
    SpanCharacteristic *on, *lvl;
    GwLight() : Service::LightBulb() {
```

```
        on  = new Characteristic::On(false);
        lvl = new Characteristic::Brightness(60);
        lvl->setRange(0, 100, 1);
    }
    boolean update() override {          /* Siri / Home App 下指令 */
        if (!on->getNewVal()) gwSend("CMD LIGHT OFF");
        else                  gwSendf("CMD LIGHT PCT %d", lvl->getNewVal());
        return true;
    }
};

void setup() {
    Serial2.begin(115200, SERIAL_8N1, 16, 17);   /* 接 STM32 */
    homeSpan.setPairingCode("11122333");         /* 課堂務必改掉 */
    homeSpan.begin(Category::Bridges, "VoiceHome Bridge");
    /* Accessory + AccessoryInformation + GwLight/GwFan/... 見完整檔 */
    gwSend("GET STATE");
}
```

STM32 每次狀態變更推回的 JSON，由 `loop()` 裡的 `gwPoll()` 解析後 `setVal()`——於是你對板子說「guan-deng」（路徑 A），手機的家庭 App 一秒內同步變暗。這個雙向同步是驗收重點，不是選配。

> **說明**　自製配件在 iOS 會顯示「未認證配件」，課堂與自用可直接忽略；要上架販售才需要 Works with Apple Home 認證。配對用 8 位數設定碼即可，不需要 MFi 晶片。

13.5　Alexa：fauxmoESP 本地模擬與正式技能

最快的課堂做法是 fauxmoESP：ESP32 在區網內模擬 Philips Hue 裝置，Echo 音箱用「Alexa, discover devices」即可找到，開關與亮度（0–255 映射到 0–100%）都能控，完全不碰雲端開發者帳號。

程式碼 13.4: fauxmo_alexa.ino（節錄）

```
fauxmo.addDevice("living room light");
fauxmo.addDevice("living room fan");
fauxmo.addDevice("smart plug");

fauxmo.onSetState([](unsigned char id, const char *name,
                     bool state, unsigned char value) {
    int pct = (int)value * 100 / 255;
    if (strcmp(name, "living room light") == 0)
        state ? gwSendf("CMD LIGHT PCT %d", pct ? pct : 60)
              : gwSend("CMD LIGHT OFF");
    else if (strcmp(name, "living room fan") == 0)
```

```
        state ? gwSendf("CMD FAN PCT %d", pct ? pct : 40)
              : gwSend("CMD FAN OFF");
    else
        gwSend(state ? "CMD RELAY ON" : "CMD RELAY OFF");
});
```

> **提示**　Alexa 智慧家庭對中文裝置名支援有限，裝置名請用英文；Siri 則可直接叫「客廳燈」。Google Home 的房間 + 裝置命名在 App 內設定，與韌體無關。

fauxmo 的限制：只有「開關 + 亮度」語意、狀態回查支援有限、依賴區網發現（部分新款 Echo 韌體會挑剔）。正式產品走 **Smart Home Skill**（Lambda + OAuth 帳號連結 + AWS IoT）或直接上 Matter。

13.6　OK Google：走 Matter 或 Home Assistant

Google 的自建雲端路線（Smart Home Action + fulfillment 伺服器 + HomeGraph）需要 OAuth 伺服器與 GCP 專案，超出課堂時數。實務上二選一：

1. **Matter**：Google Nest 音箱 / 顯示器是 Matter 控制器，「OK Google，打開電燈」直接可用（見下節）。
2. **Home Assistant**：HA 的 Google Assistant 整合（自行設定 GCP 或訂閱 Nabu Casa 雲端）把 HA 實體暴露給 Google Home。

13.7　Matter：一次接好三家

Matter 是 CSA（連接標準聯盟）主導的智慧家庭標準，Apple Home、Alexa、Google Home 都是控制器。裝置用標準 **device type**（On/Off Light、Dimmable Light、Fan、On/Off Plug-in Unit……）描述自己，配對一次 QR code，三家 App 都能加入同一顆裝置（multi-admin）。

對本專案，最務實的做法是 ESP32-C3 / S3 跑 **esp-matter** SDK：把官方 light 範例的 attribute 回呼改成送 `CMD` 行給 STM32，狀態回推則更新 Matter attribute。Wi-Fi 版最簡單；Thread 版需要邊界路由器（HomePod mini、Nest Hub 等）。細節與骨架見 `firmware/lab12_assistant_gateway/esp32/matter_bridge.md`。

> **說明**　2026 年的現實：DIY 裝置可用測試憑證在自家配對三家生態系；要出貨才需要 CSA 認證與正式 DAC 憑證。教學階段 Matter 的價值在於理解「device type = 標準化的 `home_state_t`」——你會發現它的 OnOff/Level cluster 與我們的狀態機幾乎一一對應。

13.8 Home Assistant 樞紐：接第 11 章的 MQTT

若實驗室有樹莓派，Home Assistant 是一次涵蓋三家的樞紐：STM32 的 JSON 遙測經 ESP32 的 MQTT–UART 橋（`esp32/mqtt_uart_bridge/`，約 80 行）上到 broker，HA 用 MQTT 實體接進來，再由 HomeKit Bridge 整合暴露給 Siri、Alexa / Google 整合暴露給另外兩家。下行命令直接發布閘道文字行，橋接器原樣轉進 UART——閘道零解析。

程式碼 13.5: configuration.yaml（燈的部分）

```
mqtt:
  light:
    - name: "客廳燈"
      state_topic: "home/living/voice/state"
      state_value_template: "{{ 'ON' if value_json.light == 1 else 'OFF' }}"
      command_topic: "home/living/voice/cmd"
      payload_on: "CMD LIGHT ON"
      payload_off: "CMD LIGHT OFF"
      brightness_state_topic: "home/living/voice/state"
      brightness_value_template: "{{ value_json.l_pct }}"
      brightness_command_topic: "home/living/voice/cmd"
      brightness_command_template: "CMD LIGHT PCT {{ value }}"
      brightness_scale: 100

homekit:            # 暴露給 Siri / 家庭 App
  filter:
    include_domains: [light, fan, switch]
```

13.9 實驗：lab12 助理閘道

> **對應程式碼**
>
> `firmware/lab12_assistant_gateway/`：STM32 端 `app_main.c` + `gw_proto.c`（含 PC 單元測試）；ESP32 端三份草稿 `homespan_siri`、`fauxmo_alexa`、`mqtt_uart_bridge` 與 `homeassistant/configuration.yaml`。接線：ESP32 GPIO16/17 ↔ STM32 USART6 TX/RX、共地。

驗收腳本：

1. 「嘿 Siri，打開客廳燈」→ 燈亮，VCP 印出 `src=gateway`。
2. 在 VCP 打 `xiao-ke`、`guan-deng`（本地路徑）→ 燈滅，且家庭 App 內的燈同步變為關閉。
3. 「Alexa, set living room fan to fifty percent」→ 風扇 50%。
4. 拔掉 ESP32 → 本地口令照常運作；插回後 App 於 5 秒內恢復同步。

5. `APP_GW_ALLOW_DOOR=0` 重編 → Siri 開門被拒，log 顯示 `gw: door blocked`。

> **動手練習**
>
> 三個進階練習：(1) 在 ESP32 每 3 秒送 `PING`，逾時 10 秒未見 `PONG` 就把配件標為離線（HomeSpan 可回報 not-responding）。(2) 為遠端指令加上每來源 400 ms 去抖，防止 App 拖曳亮度滑桿灌爆佇列。(3) 把 Gateway 任務併入 `voice_home_final`：加入 `gw_proto.c` 與 `vTaskGateway`，優先權 `PRIO_GATEWAY`，再跑一次第 12 章的看門狗與睡眠驗收——睡眠期間 UART 喚醒是否正常？

全書終點

回頭看：同一顆 STM32，耳朵可以是離線模組、板上神經網路，或全球最大的三個語音生態系；但設備狀態機、指令佇列、看門狗與 OTA 從第 6 章之後就沒改過。這就是架構的意義——入口會一直換，骨架不必重寫。

第 A 章

硬體清單、官方範例與除錯

A.1 附錄 A　建議 BOM

項目	規格	數量
開發板	NUCLEO-F429ZI（主線）或 F407G-DISC1	1
AI 進階板	STM32H747I-DISCO（路徑 B）	可選
語音模組	CI-03T / ASRPRO / Gravity 離線語音	1（路徑 A）
麥克風	INMP441 I2S 模組	1（路徑 B）
助理閘道	ESP32 DevKit（HomeSpan ／ fauxmo ／ Matter）	1（第 13 章）
LED + 電阻	5 mm，220 Ω–1 kΩ	3 + 3
蜂鳴器	主動、3.3 V 可驅	1
繼電器	5 V 光耦模組，低 / 高電平看標示	1
MOSFET	邏輯閘極 N 溝，加飛輪二極體	1
風扇	12 V 或改用 LED 模擬	1
舵機	SG90 + 5 V 獨立供电（共地）	1
感測	DHT22 或 SHT31	可選
線材	杜邦線、麵包板、USB	—

供電：舵機與繼電器不要跟 MCU 搶 Nucleo 的 5 V 若電流接近上限；外接 5 V 時**共地**。12 V 風扇電源與 MCU 隔離地線相連即可。

A.2 附錄 A2　腳位建議（NUCLEO-F429ZI）

實際以你的 `.ioc` 為準，下表為教材韌體 `pin_map.h` 預設。

功能	腳	週邊
LD1 / LD2 / LD3	PB0 / PB7 / PB14	板上 LED
使用者按鍵	PC13	EXTI
VCP UART	PD8 / PD9	USART3
ASR 模組 UART	PG14 / PG9	USART6
助理閘道 UART	預設共用 USART6，或改 USART2	第 13 章
蜂鳴器	PF7	GPIO
繼電器 IN	PF8	GPIO
燈 PWM	PA6	TIM3_CH1
風扇 PWM	PA7	TIM3_CH2
舵機 PWM	PB8	TIM4_CH3
I2S 麥	依模組（SPI2/I2S2）	見 lab07
DHT22	PG1	GPIO 時序

A.3 附錄 B　STM32 官方延伸

學員做完本課程後，建議依序閱讀（名稱以 ST 網站當年度為準）：

- **X-CUBE-FREERTOS** / CubeMX Middleware FREERTOS：官方移植、CMSIS-RTOS v2、低功耗 tickless 範例。
- **X-CUBE-AI** 與 **STM32Cube.AI Developer Cloud**：模型分析、量化、RAM/Flash 報告。
- **FP-AI-NANOEDG1**、音訊 KWS 應用筆記：特徵參數與 DMA 管線。
- **STM32H747I-DISCO** 音訊與 AI 示範：雙核時可把推論放 M7、控制放 M4（進階選修）。
- **X-CUBE-IAP-USART**、各系列 AN（IAP / 雙 Bank）：OTA 實作細節與 Flash 擦寫時序。
- **UM1722**（F4 HAL）、對應系列 Reference Manual：時鐘與 IWDG 公式的唯一權威。
- **Matter（CSA）與 esp-matter**：跨 Siri、Alexa、Google 的統一配網與裝置模型（第 13 章）。
- **HomeSpan**（ESP32 上的 HomeKit HAP）、**fauxmoESP**（Alexa 區網模擬）、**Home Assistant**（HomeKit Bridge、Alexa、Google Assistant 整合）。

本教材的任務切分刻意對齊這些範例：你把 Audio 任務換成官方 PDM 濾波、把 `kws_infer` 換成官方 `ai_run`，產品狀態機可以不動。

A.4 附錄 C　常見錯誤速查

現象	先查什麼
下載成功但不跑	時鐘、`Reset_Handler`、是否停在 HardFault
HAL_Delay 卡死	SysTick 是否已被 FreeRTOS 占用卻仍呼叫 HAL_Delay
任務建立失敗	`configTOTAL_HEAP_SIZE`、堆疊單位是 word
DMA 永遠 0	緩衝在 CCM、沒開 DMA 時鐘、H7 cache
鮑率亂碼	APB 時鐘與 USART 設定、HSE 沒開卻當 8 MHz
I2S 雜訊 / 辨識差	PLLI2S 不準、WS 左右聲道、地迴路
繼電器狂跳	觸發電平反了、腳浮空、沒去抖
舵機抖動	5 V 不足、地未共、PWM 不是 50 Hz
餵狗仍復位	逾時太短、某任務心跳停、FAULT 位置 1
Stop 後醒來時鐘錯	沒有重新 `SystemClock_Config`
OTA 變磚	擦錯 Bank、沒 CRC、bootloader 跳到空區
助理能控但 App 不同步	`EVT_GW_DIRTY` 沒置位、狀態行欄位順序被改動
Alexa 找不到裝置	fauxmo 需 80 埠、與 Echo 同一區網、裝置名用英文

A.5 附錄 D　FreeRTOSConfig 教學預設

程式碼 **A.1:** 與本課程相依的關鍵宏（節錄）

```
#define configCPU_CLOCK_HZ              180000000
#define configTICK_RATE_HZ              1000
#define configMAX_PRIORITIES            7
#define configMINIMAL_STACK_SIZE        128
#define configTOTAL_HEAP_SIZE           (40 * 1024)
#define configUSE_PREEMPTION            1
#define configUSE_TIME_SLICING          1
#define configUSE_MUTEXES               1
#define configUSE_COUNTING_SEMAPHORES   1
#define configUSE_TIMERS                1
#define configTIMER_TASK_PRIORITY       3
#define configTIMER_QUEUE_LENGTH        8
#define configCHECK_FOR_STACK_OVERFLOW  2
#define configUSE_MALLOC_FAILED_HOOK    1
#define configASSERT(x) if((x)==0){taskDISABLE_INTERRUPTS();for(;;);}
```

`vApplicationStackOverflowHook` 與 `vApplicationMallocFailedHook` 在 `firmware/common/src/app_hooks` 點燈 + 置位 FAULT。

A.6 附錄 E　課程評量建議

- 週測驗（選擇 + 短答）：NVIC vs 任務優先權、Queue vs Event Group、為何 ISR 禁用 `printf`。
- 實驗簽核：每週 lab 的 UART 截圖或助教現場看一眼。
- 期末：final 專案 3 分鐘 demo + 程式走讀 10 分鐘。

A.7 附錄 F　韌體目錄索引

`firmware/common/`	型別、腳位、log、心跳、設備 API
`lab01_blinky/`	第 2 章
`lab02_gpio_devices/`	第 3 章
`lab03_pwm_fan_motor/`	第 4 章
`lab04_freertos_blinky/`	第 5 章
`lab05_queue_semaphore/`	第 6 章
`lab06_event_group/`	第 7 章
`lab07_audio_capture/`	第 8 章
`lab08_asr_module/`	第 9 章路徑 A
`lab08_asr_edgeai/`	第 9 章路徑 B 接線
`lab09_voice_controller/`	第 10 章
`lab10_comms/`	第 11 章
`lab11_watchdog_lpm_ota/`	第 12 章
`lab12_assistant_gateway/`	第 13 章（含 ESP32 橋接與 HA 設定）
`voice_home_final/`	總成
`tools/`	序列埠腳本

編譯本教材：於 `latex/` 目錄執行 `latexmk -xelatex main.tex`。