



REALTEK

AN0300

AmebaPro application note

Abstract

AmebaPro 是一顆高整合度的 IC，包含了 802.11 Wi-Fi, H.264 影像編碼器，Audio Codec.

這篇使用手冊介紹如何使用開發版，如使用 SDK 編譯範例程序，以及如使下載固件至開發版上運行。

Table of Contents

1	編譯和執行固件.....	5
1.1	SDK Project 說明	5
1.2	編譯程序	5
1.2.1	編譯小核程序.....	5
1.2.2	編譯大核程序.....	5
1.2.3	產生 Bin 檔說明	6
1.3	使用 image tool 下載與合併固件	7
1.3.1	環境設置.....	8
1.3.2	固件下載.....	8
1.3.3	固件合併.....	12
1.4	使用 JTAG/SWD 進行 Debug	13
1.4.1	JTAG/SWD 連線檢查	13
1.5	使用 uart 觀看小核的 log	16
2	開發板.....	17
2.1	選擇 Image Sensor.....	17
2.1.1	OV2735.....	17
2.1.2	SC2232	17
3	Flash Layout.....	18
3.1	Flash Layout 概觀	18
3.2	Flash used by Application	19
4	如何使用範例程式.....	19
4.1	應用範例程式.....	19
4.2	外設範例程式.....	20
4.3	Wi-Fi 範例程式.....	20
4.3.1	使用 AT command 來做連線	20
4.3.2	Wlan scenario example	21
4.4	Video 範例程式.....	21
5	記憶體配置與使用	21
5.1	記憶體種類	21
5.1.1	AmebaPro 大核的記憶體大小與配置	21

5.1.2	AmebaPro 小核的記憶體大小與配置.....	23
5.2	記憶體配置	24
5.2.1	在 IAR 環境裡配置記憶體	24
5.2.2	在 ICF 檔案裡配置記憶體.....	25
5.2.3	記憶體超出範圍.....	25
5.3	Video 使用 DRAM.....	26
5.4	不可移動配置的程式	26
5.4.1	與 XIP 相關的 flash api	26
5.5	計算系統記憶體使用量	26
6	OTA	27
6.1	OTA 執行流程.....	28
6.2	裝置開機流程.....	29
6.3	目標更新區塊.....	30
6.4	生成 OTA 固件.....	31
6.4.1	OTA 固件格式.....	31
6.4.2	OTA 更新切換固件行為.....	32
6.4.3	OTA 固件編譯設定.....	33
6.5	透過 Wi-Fi 實作 OTA 更新.....	35
6.5.1	基於 socket 的 OTA 更新範例	35
6.5.2	基於 HTTP 的 OTA 更新範例	38
6.6	OTA signature	41
7	電源管理.....	42
7.1	AmebaPro Power Save Modes.....	42
7.1.1	DeepSleep	42
7.1.2	Standby	43
7.1.3	Wakeup from WLAN	43
7.1.4	遠端喚醒功能.....	44
7.1.5	使用 Magic Pattern 喚醒.....	46
7.2	Examples.....	46
7.2.1	Example: power_save	46
7.2.2	Example: Custom wake on wlan pattern	50
7.2.3	Example: system suspend with wake on wlan after streaming stop	51
7.3	編程建議	54
7.3.1	Standby 若不需要用到 wowlan, 則需要關掉 wlan	54
7.3.2	當 HS 睡眠時不可以使用 ICC.....	54

7.3.3	進 Standby 前關閉 Video, ISP, SD, USB.....	54
7.3.4	使用 Standby 裡可保留的 LS 記憶體.....	55
7.3.5	連上 AP 與進 wake on wlan mode 中間需要一些延遲.....	55
7.3.6	進 Standby 前對 LS GPIO 狀態進行 Pull Control 設定.....	55
7.4	量測耗電.....	56
8	系統.....	58
8.1	ICC (Inter CPUs Communication).....	58
8.1.1	Example: ICC.....	58
8.2	System reset.....	58
8.2.1	ls_sys_reset.....	58
8.2.2	sys_reset.....	58
9	硬體周邊.....	59
9.1	GTimer.....	59
9.1.1	gtimer_rtc.....	59
9.1.2	SNTP.....	59
9.2	RTC.....	59
9.3	PWM.....	59
9.4	Audio.....	59
9.4.1	DAC Volume.....	60
9.4.2	ADC Volume.....	60
9.5	Efuse.....	61
9.5.1	User OTP.....	61
9.6	Ethernet.....	62
9.7	SD CARD.....	62
10	檔案系統.....	63
10.1	FAT Filesystem on Flash.....	64
10.1.1	Software Setup.....	64
10.2	Dual FAT Filesystem - File system on both SD Card and Flash.....	64
10.2.1	Hardware Setup.....	64
10.2.2	Software Setup.....	65
10.3	產生 FAT File System Image.....	65

1 編譯和執行固件

本章節介紹如何使用開發版。一整套開發版共含一個主板，一個 sensor board, 和一個子板上包含 LED 燈、光感測器，以及 IR-LED。SDK 中包含以上套件的示例程序。SDK 的 toolchain 為 IAR，是一個集編輯、編譯、下載固件、以及除錯於單一環境的整合開發環境。也可以使用 Image tool 對固件做編輯，編譯，以及下載固件至開發版。

1.1 SDK Project 說明

 **注意：** IAR 版本需使用 8.30

AmebaPro 的 project 目前提供 ignore_secureee，project_is(ignore secure)沒使用 ARM 的 TrustZone 技術，程式開發上比較方便，適合只做一次開發的客戶使用。

1.2 編譯程序

AmebaPro 採用最新的 Big-Little 架構，大 CPU 為 300MHz, 高速的單元例如 Wi-Fi, ISP, Encoder, Codec 位於大核，小 CPU 為 4MHz, 需要在低功耗下運作的周邊裝置可以規畫接在小核，大核進入省電模式時小核可以醒著，待喚醒事件觸發時再將大核啟動。由於大核會吃到小核的設定，在編譯階段需先編譯小核再編譯大核。

1.2.1 編譯小核程序

Step1. 開啓 SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/Project_lp.eww


Step2. application_lp 在 WorkSpace 中，右鍵選擇 Rebuild All 編譯 application_lp。

Step3. 確認每一個編譯的結果是否有 error。

1.2.2 編譯大核程序

Step1. 開啓位於 SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/Project_is.eww

Step2. 確認 application_is 在 WorkSpace 中，編譯 application_is 按下右鍵並且選擇

 **大核和小核 CPU 之間的溝通透過 share memory, SDK 裡有範例說明如何使用已經提供好的通訊通道溝通. 請參考 Inter Channel Communication.**

Rebuild All。

Step3. 確認每一個編譯的結果是否有 error。

1.2.3 產生 Bin 檔說明

編譯結束後，在 EWARM-RELEASE\Debug\Exe 的路徑下可以看到 partition.bin，boot.bin，firmware_is.bin 與 flash_is_ota1.bin 幾個 image 檔案。

partition.bin 存放了 partition table，記錄存放 Boot image 與 firmware image 的起始位址，boot.bin 為 bootloader image，firmware_is.bin 為 application image，flash_is_ota1.bin 連結了 partition.bin，boot.bin 與 firmware_is.bin，使用 image tool 燒錄 image 時須選擇 flash_is_ota1.bin。

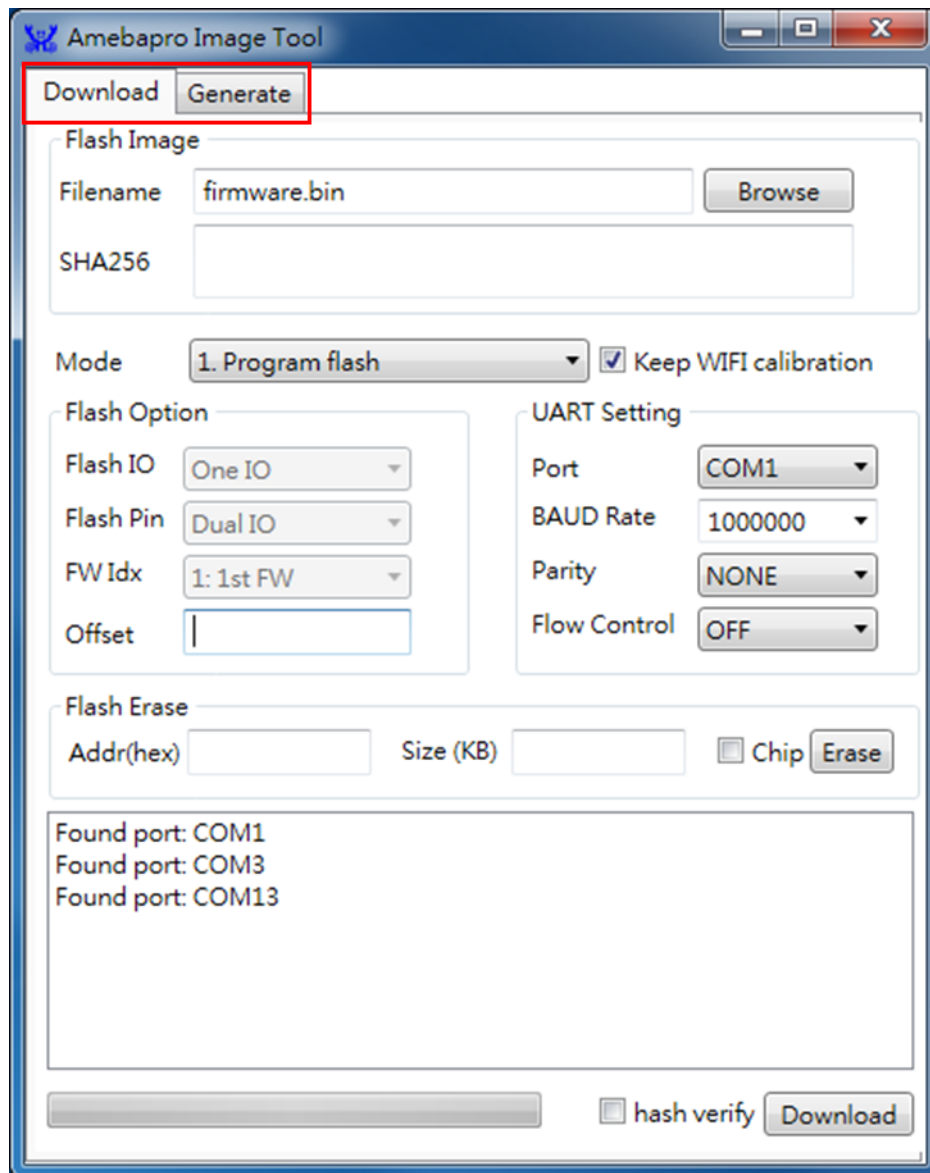
1.3 使用 image tool 下載與合併固件

本章節介紹如何使用 Image Tool 合成與下載固件。

如下圖，Image Tool 有以下兩分頁：

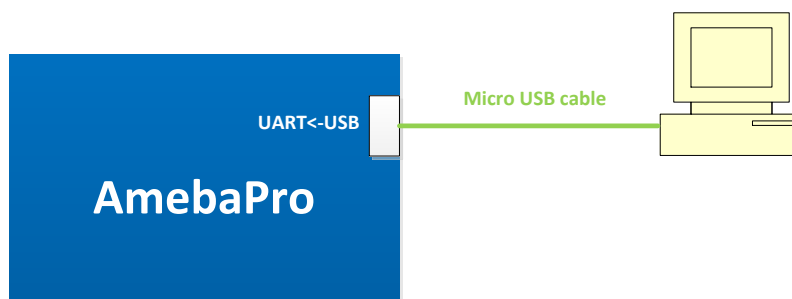
- Download：Image Tool 扮演固件下載伺服器端，透過 UART 將固件下載至 AmebaPro。
- Generate：合併多個固件以生成一最終固件。

 注意：如需透過外接 *uart* 進行燒錄須使用 **FT232** USB To UART dongle


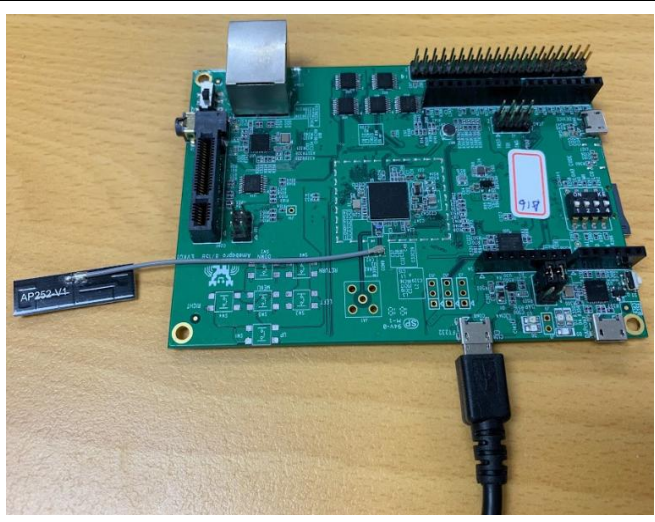


1.3.1 環境設置

1.3.1.1 硬體設置






目前 AmebaPro 提供兩種不同的硬體配置，對應的版號為

2V0、2V1	1V0
	

1.3.1.2 軟體設置

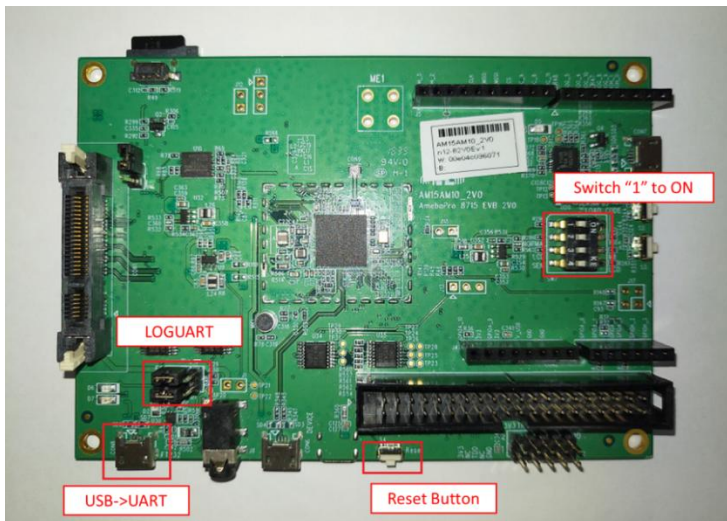
從 project\tools\AmebaPro\Image_Tool\執行 ImageTool.exe

系統需求：Win7 以上，支援 Microsoft .NET Framework 4.5

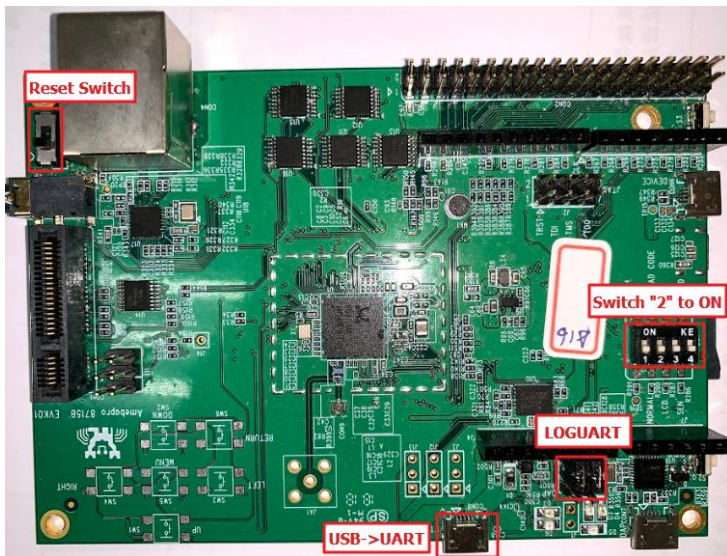
 Install	4/19/2019 2:00 PM	File folder
 ImageTool.exe	4/19/2019 2:00 PM	Application
 Newtonsoft.Json.dll	10/2/2013 12:10 PM	Application extension

1.3.2 固件下載

Image Tool 可透過 UART 將固件傳輸至 AmebaPro。執行此下載功能前，AmebaPro 需先進入 UART_DOWNLOAD 模式。請參考以下步驟使 AmebaPro 以 UART_DOWNLOAD 模式開機：



2V0、2V1



1V0

Step 1: 將 LOGUART 與 FT pin 以 jumper 連接

Step 2: 將 USB->UART 以 micro-USB 連接至 PC

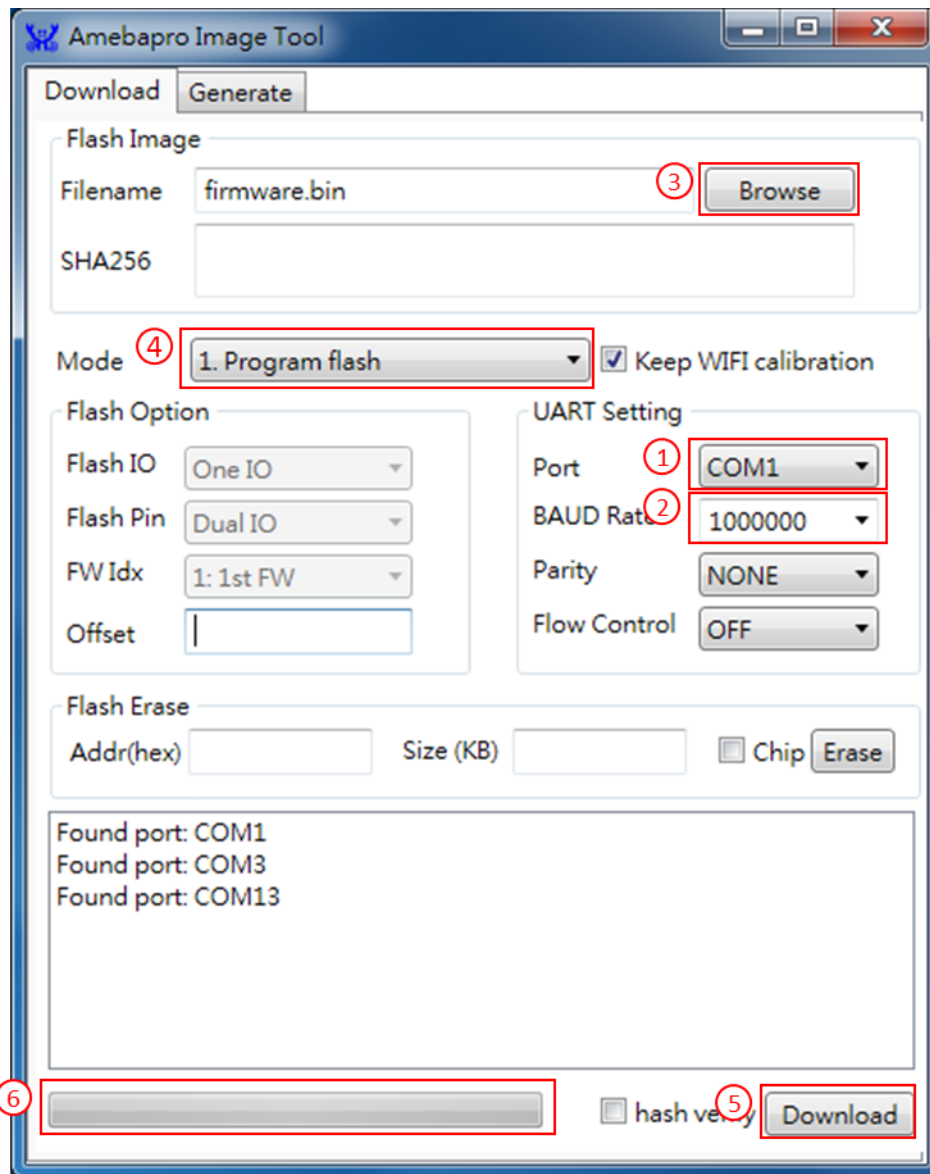
Step 3: 將 SW7 第一個開關撥至 ON 狀態(2V0、2V1)or 第二個開關撥至 ON 狀態(1V0)

Step 4: 按壓 reset 鈕重新開機

執行完以上步驟後，應能從 UART console 看到以下訊息，代表板子目前是以 UART_DOWNLOAD 模式開機：

```
== Rtl8195bh IoT Platform ==
Chip VID: 0, Ver: 2
ROM Version: v3.0
Test Mode: boot_cfg1=0x0
Download Image over UART0_S1
```

1.3.2.1 Flash 固件下載



將固件透過 Image Tool 下載至 AmebaPro，請先確認版子以 UART_DOWNLOAD 模式開機，並參考以下步驟：

Step 1: Image Tool 會自動掃描現在電腦偵測到的 UART ports，請選擇 AmebaPro 所使用之對應 UART port

請確認該 UART port 沒有被其他應用佔用

Step 2: 請選擇 PC 與 AmebaPro 欲使用的 baud rate

Step 3: 選擇目標映像檔 “flash_xx.bin”

Step 4: 確認燒錄模式為 “1. Program flash”

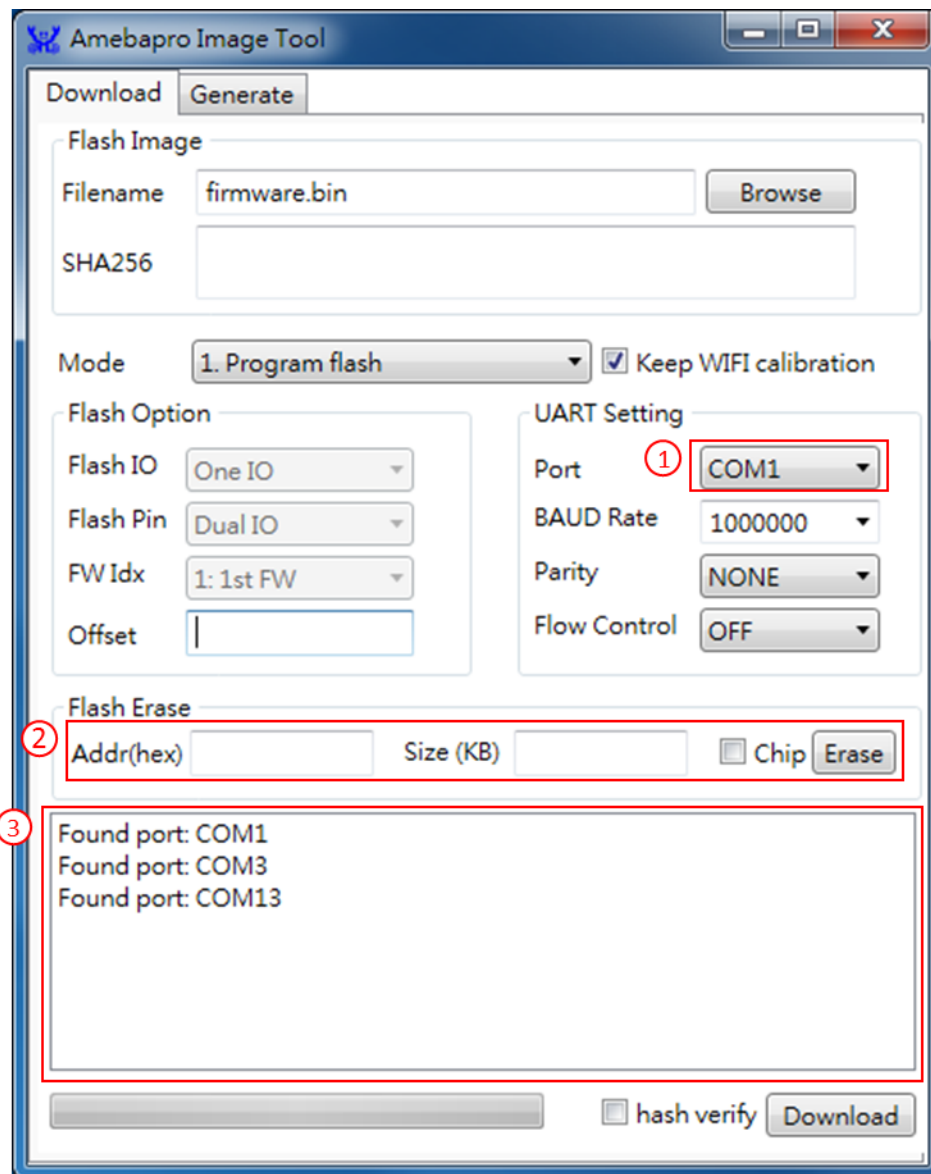
Step 5: 點選 “Download” 以進行燒錄

Step 6: 燒錄之進度會顯示在進度條上，並且會印出燒錄結果

NOTE: 其餘設定保留使用預設值,

Flash IO : One IO/Flash Pin : Dual IO/Parity : NONE/Flow Control : OFF

1.3.2.2 Flash 固件清除



將 AmebaPro Flash 上固件擦除，請先確認版子以 UART_DOWNLOAD 模式開機，並參考以下步驟：

Step 1:請選擇 AmebaPro 所使用之對應 UART port

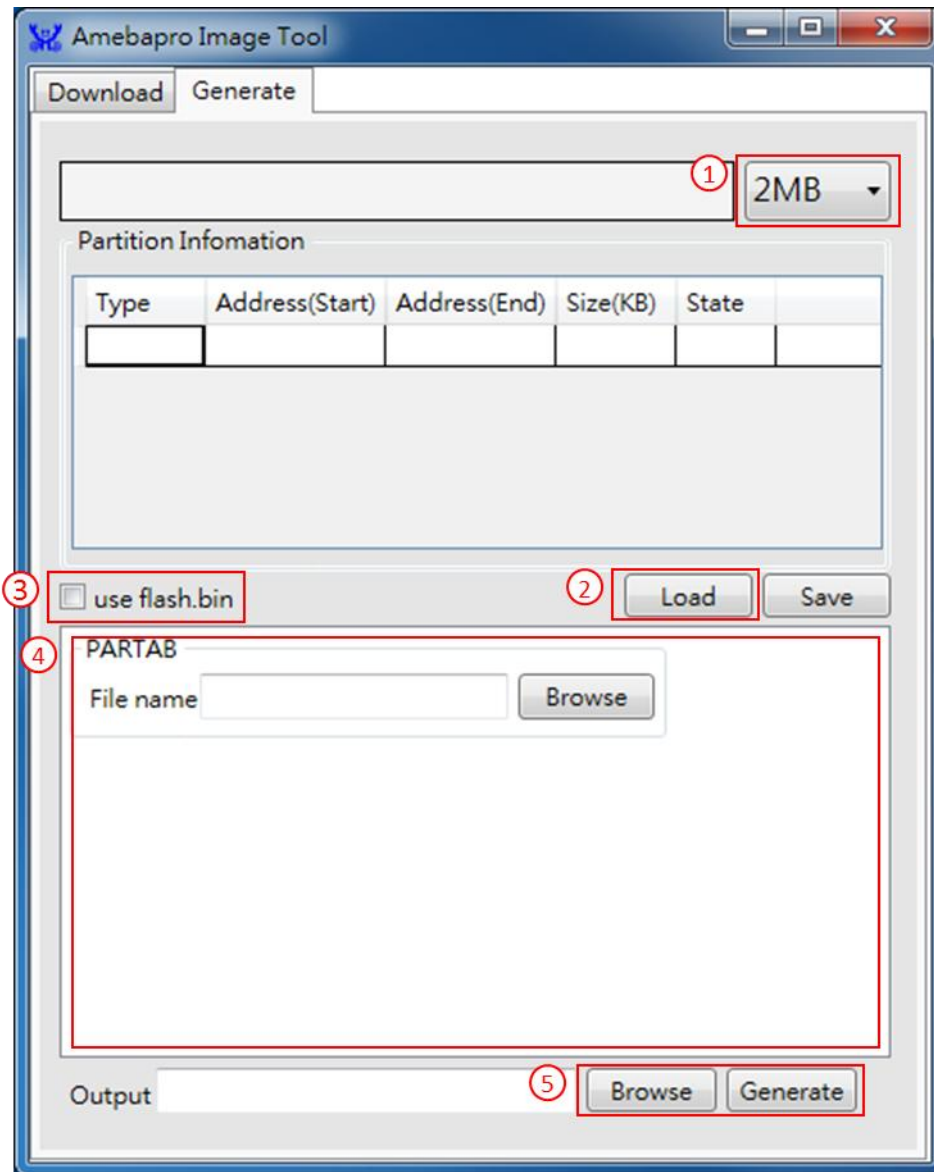
請確認該 UART port 沒有被其他應用佔用

Step 2: 選取清除範圍或直接勾選“Chip”將 Flash 直接做清除。點選“Click”執行。

Step 3: 清除完成後會將結果顯示在 console 欄位

1.3.3 固件合併

Image Tool 內 Generate 分頁能將多個固件合併為一最終固件，例如將 Normal 固件與 MP 固件做合併，後續燒錄時只需燒錄一次。



請參考以下步驟將固件合併：

Step 1: 選取目標 Flash 大小

Step 2: 讀取 partition table 設定，點選“Load”按鈕並由專案目錄選取 partition.json (e.g. EWARM-RELEASE/partition.json)

Step 3: 勾選“use flash.bin”

若不勾選，則需分別選取 partition.bin/boot.bin/firmware_is.bin 路徑

Step 4: 選取目標固件位址，例如在 FLASH.BIN 欄位選取 flash_is.bin，FW2 欄位選取 firmware_is.bin

Step 5: 選取 output 檔案位址並點選“Generate”以產生最終固件

此固件可以再依 1.3.2.1 章結介紹，一次燒錄進 AmebaPro。

1.4 使用 JTAG/SWD 進行 Debug

JTAG/SWD 是一種國際標準測試協議主要用於晶元內部測試。外部 JTAG 接口需要有三個腳位 TCK、TMS、TDI 和 TDO(分別為時鐘、模式選擇、輸入和輸出)，以及一個可選擇的 reset 腳位 nTRST。外部 SWD 接口需要兩個腳位，一個為雙向 SWDIO 訊號另外一個時鐘 SWCLK，可以從裝置輸入或輸出。

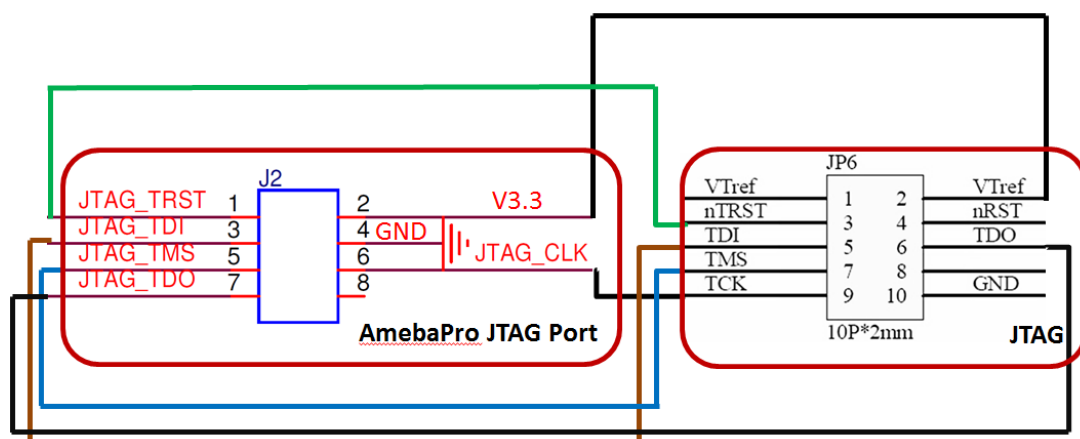
1.4.1 JTAG/SWD 連線檢查



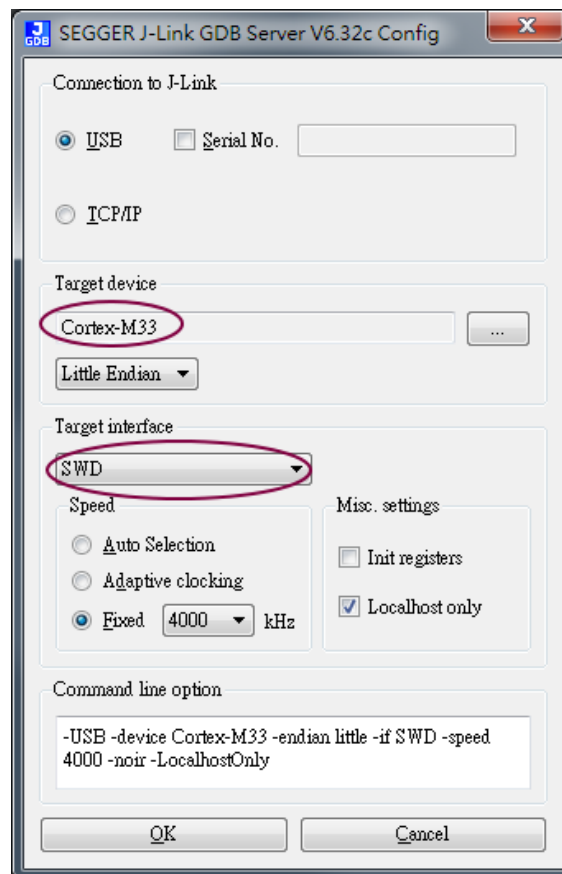
如使用 2V0、2V1 版本的 AmebaPro.

在連線之前請先檢查 SW7 pin 3 在 ON 處.

確保所有六個腳位（VTref、TCK、TMS、TDI、TDO 和 nTRST）都相互連接，請按照電路接線圖將 AmebaPro JTAG 端口連接到 J-Link，如果使用 SWD，請先確認所有四個腳位（VTref[VDD]、TMS[SWDIO]、TCLK[SWCLK] 和 TDO[SWO]）連線正常。



接線連接成功後，打開最新的 J-Link GDB server。然後按照下圖，選擇 target device 為 Cortex-M33(for AmebaPro)，target interface 為 JTAG / SWD，然後選擇 OK。



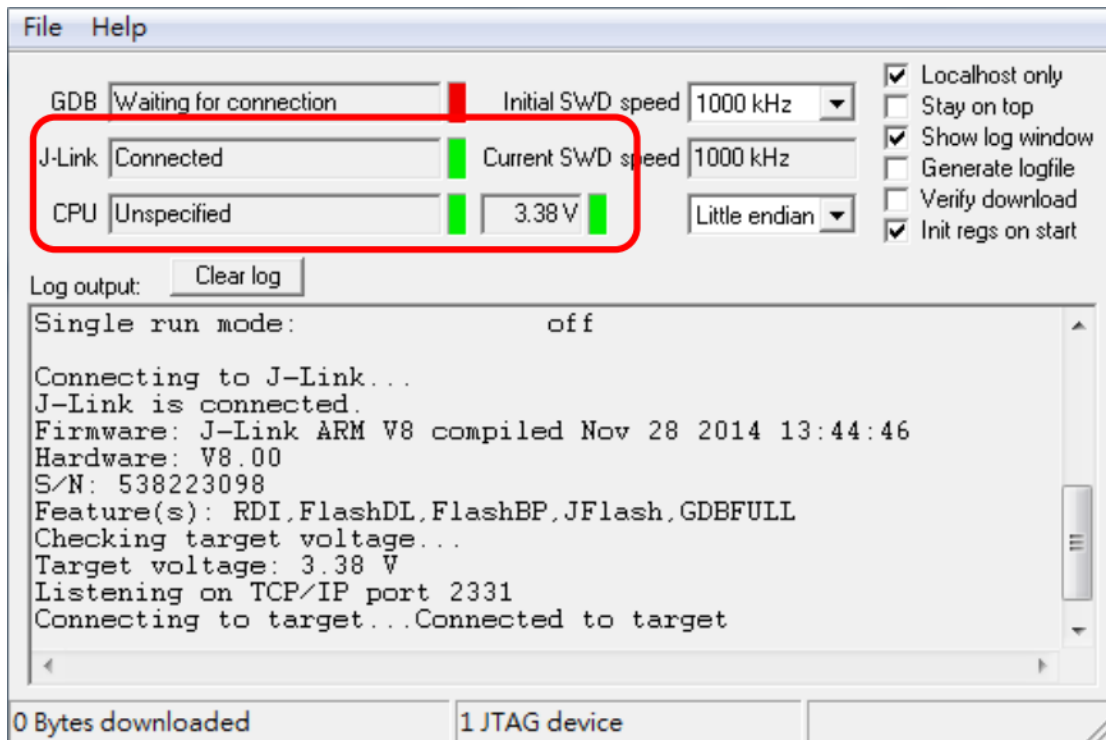
註、CMSIS-DAP 在應用上有兩個限制：

- (1) CMSIS-DAP 傳輸速度太慢。
- (2) System reset 無法完整 reset 系統只能 reset HS core。

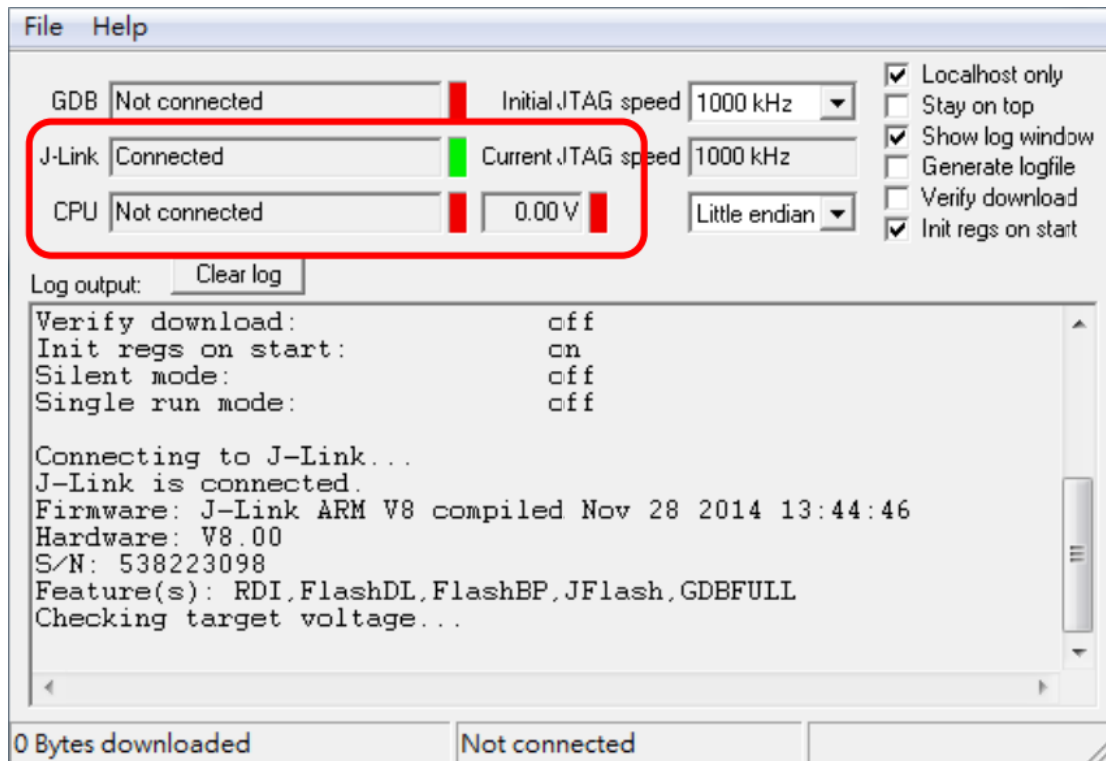
若要使用 WATCHDOG RESET + SYSTEM RESET，只能用於 JLINK 無法用在 CMSIS-DAP。

建議使用 JLINK V9 以上的版本，目前不支援 step in/out/over 的功能，若欲設立中斷位置直接在想看的點設 breakpoint。

如果連接成功，J-Link GDB 服務器將如下圖所示。

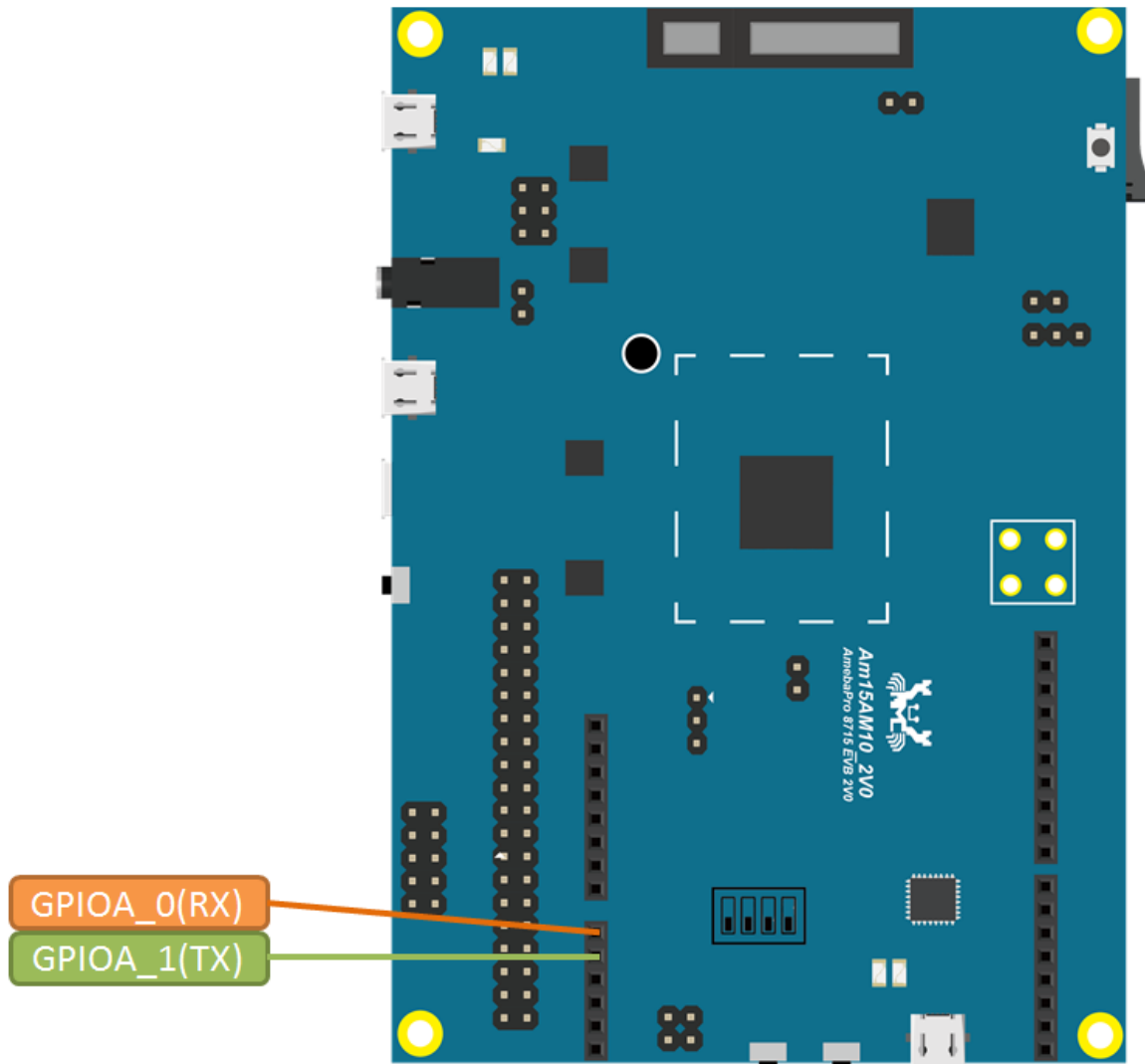


如果連接失敗，J-Link GDB server 將如下圖所示。



1.5 使用 uart 觀看小核的 log

在 AmebaPro 中大核與小核之 log 是分開的，大核 log 可以透過 FT232 或 DAP 觀看，而小核需透過 A0,A1 腳位接到 uart console，來得到小核的 log



2 開發板

這個章節介紹如何使用 AmebaPro 所支持的 image sensor board.。只有在軟體及硬體都正確的設定下，image sensor 才可以正常的操作。在 AmebaPro 的 SDK 中，我們提供了一套防護機制可以避免軟體及硬體的操作失誤。

2.1 選擇 Image Sensor

以下是目前開發版支援的 sensor 列表。

AmebaPro EVAL	Sensor	備註
AmebaPro	OV2735	
AmebaPro	SC2232	

2.1.1 OV2735

硬件設定步驟

- 插入正確的 OV2735 sensor board

軟件設定步驟

- In project\realtek_amebapro_v0_example\inc\sensor.h
- Set the definition: #define SENSOR_USE Sensor_OV2735
- In component\soc\realtek\8195b\misc\bsp\image
Delete original isp.bin, and rename isp_ov2735.bin to isp.bin

2.1.2 SC2232

硬件設定步驟

- 插入正確的 SC2232 sensor board

軟件設定步驟

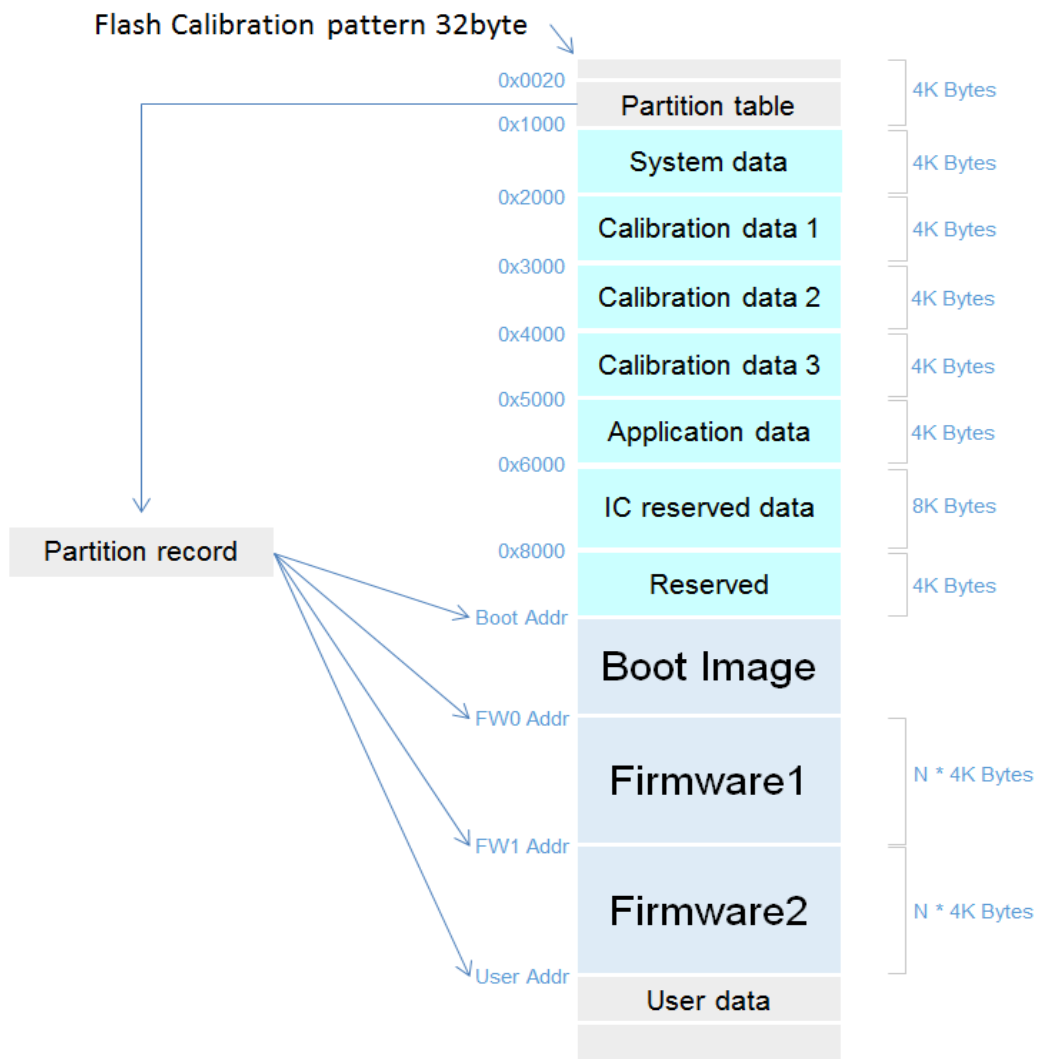
- In project\realtek_amebapro_v0_example\inc\sensor.h
Set the definition: #define SENSOR_USE Sensor_SC2232
- In component\soc\realtek\8195b\misc\bsp\image
Delete original isp.bin, and rename isp_sc2232.bin to isp.bin

 如果使用者使用錯誤的硬體或是軟體，video_sensor_check(SENSOR_USE) 會依據 SENSOR_USE 對硬體發出 i2c 訊號來檢查，如果硬體回傳值錯誤將會返回錯誤碼 -1

3 Flash Layout

AmebaPro 採用了不同於 Ameba-1 的 flash layout, 主要是 AmebaPro 是 Big-Little 大小核架構，且有 sub-mcu 系統設計，因此考慮更有彈性和擴充性的設計。

3.1 Flash Layout 概觀



3.2 Flash used by Application

SDK 中的某些函式或應用可能會使用閃存區塊。如果啓用了以下某些功能或應用程序，請特別注意 flash 區塊的設定是否和應用程序覆蓋或衝突。如果需要，可以 flash 區塊的定義可以被修改。

警告: 這些 flash 區塊的設定基於 SDK 預設固件。如果整體固件尺寸較大，則應用程序，例如 DCT_BEGIN_ADDR (0x200000) 將覆蓋到 flash 中的固件。請根據所選的 flash 尺寸和固件大小移動。

現在，如果啓用該功能，將使用以下 flash 區塊。這些定義在 platform_opts.h 中。

1. UART AT Command. Default disabled.

#define UART_SETTING_SECTOR	0x000FC000
-----------------------------	------------

2. Fast Connect: To store the AP information. Default enabled.

#define FAST_RECONNECT_DATA	0x5000
-----------------------------	--------

3.DCT: device configuration table API usage. Default disabled.

#define DCT_BEGIN_ADDR	0x200000
------------------------	----------

4 [如何使用範例程式](#)

4.1 應用範例程式

AmebaPro 的應用範例程式放在 SDK/common/example 資料夾中，在範例程式目錄中有相關的檔案，.c, .h 與 readme 檔案，在 readme 中說明了如何編譯與各重要參數之說明。

在了解完 readme 之後開啓 project，找到範例程式在 application_is 裡 utilities 的 example 資料夾中，如果沒有可以自行加入(直接拉近來或在 example 資料夾上點選右鍵選擇 Add->Add Files...)。

AmebaPro 執行的過程中 example 不會每一次開機都把所有 example 執行一次，每個 example 是否執行是透過 example_entry 來管理。選擇是否執行哪一個範例程

式是透過 Preprocessor Directives `#if` 來判斷是否執行該 `example`，若要開起範例程式需先知道 Preprocessor Directives 使用哪個參數判斷。

而 Preprocessor Directives 在 `platform_opts.h` 裡被定義，在我們想使用某一支 `example` 時需要先將 `platform_opts.h` 裡的 Preprocessor Directives 設定為 1(以 `example_dct` 為例，若要執行 `example_dct`，須將 `platform_opts.h` 裡的 `#define CONFIG_EXAMPLE_DCT` 設定為 1)，再到 `example_entry` 中確定是否有 `#if` 來判斷是否進入 `example`。

確定上面步驟都完成後重新 `rebuild application_is project` 即可確定執行該 `example`。

4.2 外設範例程式

外設範例程式可以幫我們測試 AmebaPro 周邊功能，外設範例位於 `SDK/project/realtek_amebapro_v0_example/example_sources` 中。

在每個外設範例資料夾裡分別有 `main.c` 與 `readme.txt`，`main.c` 是每一個外設範例的主程式名稱，`readme.txt` 裡面說明了外設範例的功能與參數設定或執行結果也會一併在 `readme.txt` 中說明。

想要執行外設範例只需要將相對應外設範例 `main.c` 附蓋掉原本 `project` 的 `main.c`(位於 `SDK/project/realtek_amebapro_v0_example/src` 資料夾中)，如果有其它檔案位於資料夾中也需要一併複製過去，經過編譯後即可執行。

4.3 Wi-Fi 範例程式

4.3.1 使用 AT command 來做連線

AmebaPro 提供許多 AT command 給使用者來做測試與開發，透過在 `console` 下 `command` 的方式，讓使用者可以用 AT command 來做 `wifi` 的連線，AT command 的指令請參考 `AN0025 Realtek at command.pdf`。

目前 `wi-fi direct GO` 和 `Concurrent mode` 的功能還在驗證中，會在未來的 Release 開始支援。

4.3.2 Wlan scenario example

AmebaPro 提供 wlan scenario 的 example 給使用者測試 wifi 的各種應用，example 的路徑在 \component\common\example\wlan_scenario\example_wlan_scenario.c，這個 example 提供了許多應用如 station mode，AP mode，Concurrent mode，WPS 和 P2P GO，詳細的細節以及 example 的使用方式可以參考路徑下的 readme.txt。目前 wi-fi direct GO 和 Concurrent mode 的功能還在驗證中，會在未來的 Release 開始支援。

4.4 Video 範例程式

SDK 有提供 Multimedia Framework v2，Video 的範例程式基於這個架構呈現。有關 Multimedia Framework v2 的架構以及使用範例請參考 UM0301。

5 記憶體配置與使用

本篇章節說明 AmebaPro 的各種記憶體的大小，包含 ROM, RAM, SRAM, TCM, DRAM，Flash，以及如何將程式配置在這些記憶體當中。有一些程式必須固定不可以挪動位置，也會在本章節中說明。其中 flash 的章節請參考 Flash Layout。

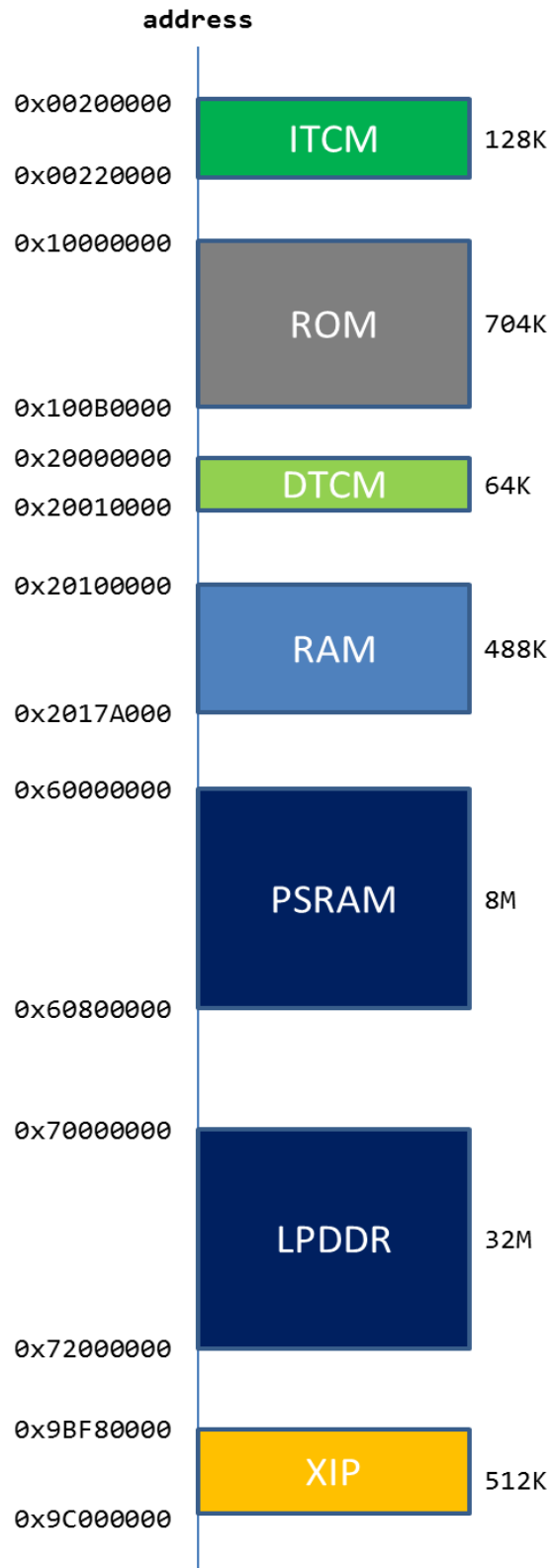
5.1 記憶體種類

5.1.1 AmebaPro 大核的記憶體大小與配置

AmebaPro 大核的記憶體大小與配置如下

	Size(bytes)	Description
ITCM	128K	可放置指令
ROM	704K	
DTCM	64K	可放置讀寫的資料
RAM	488K	即 SRAM。使用者可以使用 128K
PSRAM	8M	PSRAM 與 LPDDR 只能擇一使用。目前公板包裝使用 LPDDR。
LPDDR	32M	PSRAM 與 LPDDR 只能擇一使用。目前公板包裝使用 LPDDR。LPDDR 又稱之為 DRAM
XIP	512K	Execute In Place, Flash 的 text section 可放在 XIP

AmebaPro 大核的記憶體位址如下圖：

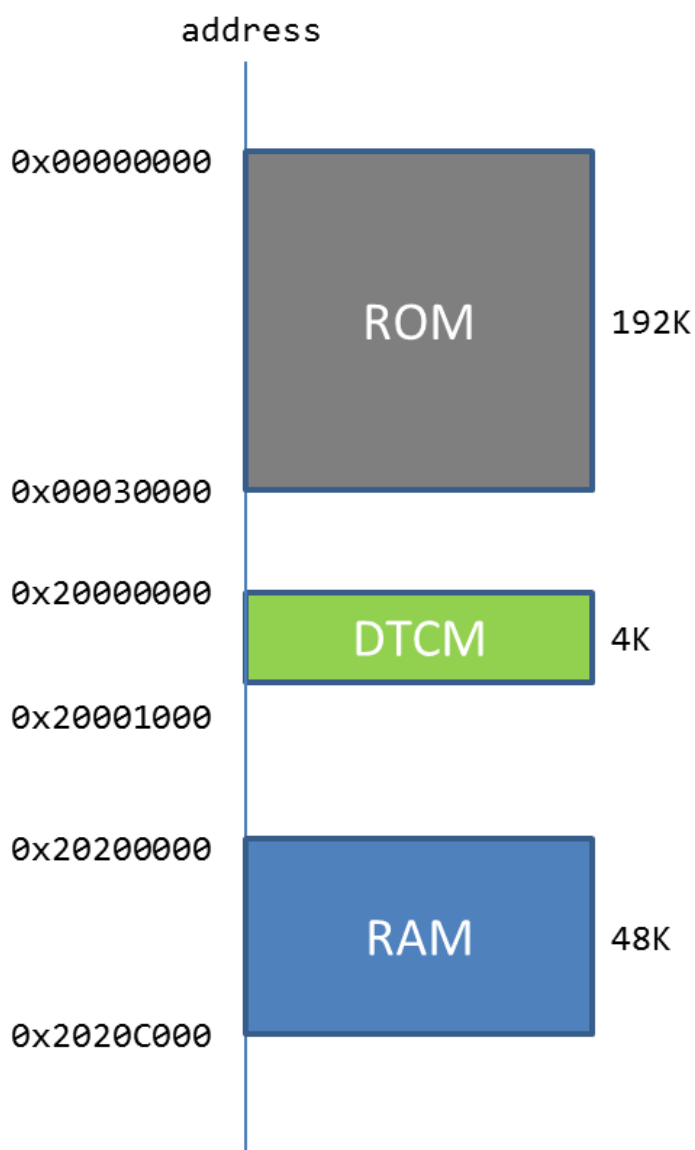


5.1.2 AmebaPro 小核的記憶體大小與配置

AmebaPro 小核的記憶體大小與配置如下

	Size(bytes)	Description
ROM	192K	
DTCM	4K	可放置讀寫的資料
RAM	48K	即 SRAM

AmebaPro 小核的記憶體位址如下圖：

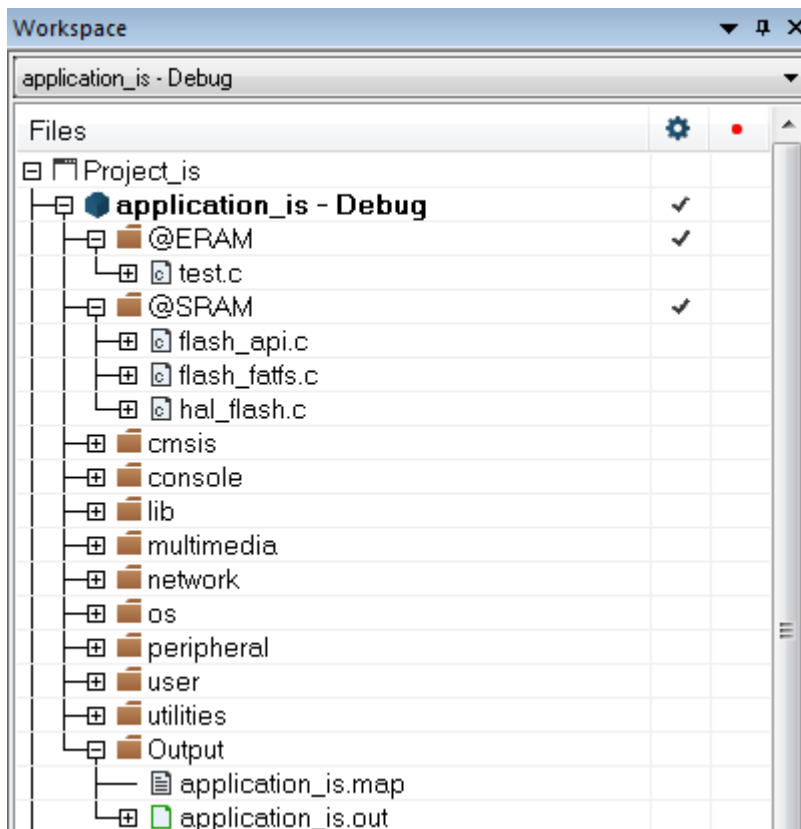


5.2 記憶體配置

5.2.1 在 IAR 環境裡配置記憶體

在 Workspace 裡, 裡面有 “@ERAM” 與 “@SRAM” 的 group, 其中 “@ERAM” 代表 External RAM, 即 PSRAM 或 LPDDR, “@SRAM” 代表 SRAM, 不在這兩個 group 的 text section 預設會被放置於 XIP 裡。

如果想將特定的 source file 的 text section 放置於 PSRAM/LPDDR, 將檔案拖曳至



“@ERAM” 即可。如圖中的 “test.c” 它的 text section 會被放至 PSRAM/LPDDR 裡。

如果想將特定的 source file 的 text section 放置於 SRAM, 要將檔案拖曳至 “@SRAM” 即可。如圖中的 “flash_api.c”, “flash_fatfs.c”, “hal_flash.c” 會被放至 SRAM 裡。

5.2.2 在 ICF 檔案裡配置記憶體

IAR 使用 ICF (IAR Configuration File) 檔案來描述與配置記憶體。AmebaPro 大核的 ICF 檔位在:

```
"SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/  
application_is.icf"
```

使用文字編輯器打開 "application_is.icf", 在檔案前面已經定義好一些 memory regions。其中可供使用者擺放的包括:

- ITCM_RAM_region
- DTCM_RAM_region
- RAM_region
- RAM_NC_region
- ERAM_region
- XIP_FLASH_region

至於 ICF 的格式與撰寫規定，請參考 IAR 的說明文件

5.2.3 記憶體超出範圍

由於目前 text section 預設擺放至 XIP 區塊裡，如果遇到 XIP 空間不足, 在 linking 時將會出現底下的錯誤

```
Error[Lp011]: section placement failed  
    unable to allocate space for sections/blocks with a total estimated minimum  
    size of 0xa0051 bytes (max align 0x8) in <[0x9bf80140-0x9bfffff]> (total uncommitted  
    space 0x7fec0).
```

可以看見錯誤的位置發生在 XIP 的位置，解決方式是將部份程式移至其它 region

5.3 Video 使用 DRAM

Video 相關的範例會使用到大量的記憶體, 以底下的情況為例：

- Stream 1: H264 1080p, 15fps, bitrate 2M + 8K AAC
- Stream 2: 720p, 30fps, bitrate 1M + 8K AAC
- Snapshot mode: 720p, JPEG Level 5

舉例: Video 總共使用的 DRAM 為 **30.58MB**:

- 1080p H264: Encoder: 6.54MB, ISP buffer: $2.97 * 3 = 8.91\text{MB}$; Encoder buffer pool: 2.97MB , **total 18.42MB**
- 720p H264: Encoder: 2.92MB, ISP buffer: $1.32 * 4 = 5.28\text{MB}$; Encoder buffer pool: 1.32MB , **total 9.52MB**
- 720p JPEG: Encoder: 3.21KB, ISP buffer: 1.32 MB; Snapshot buffer: 1.32MB , **total 2.64MB**

5.4 不可移動配置的程式

5.4.1 與 XIP 相關的 flash api

由於 XIP 使用 flash 介面, 因此 flash 相關的 api 目前已經被放置於 SRAM, 這些檔案不可移動或更改記憶體配置。這些檔案包括:

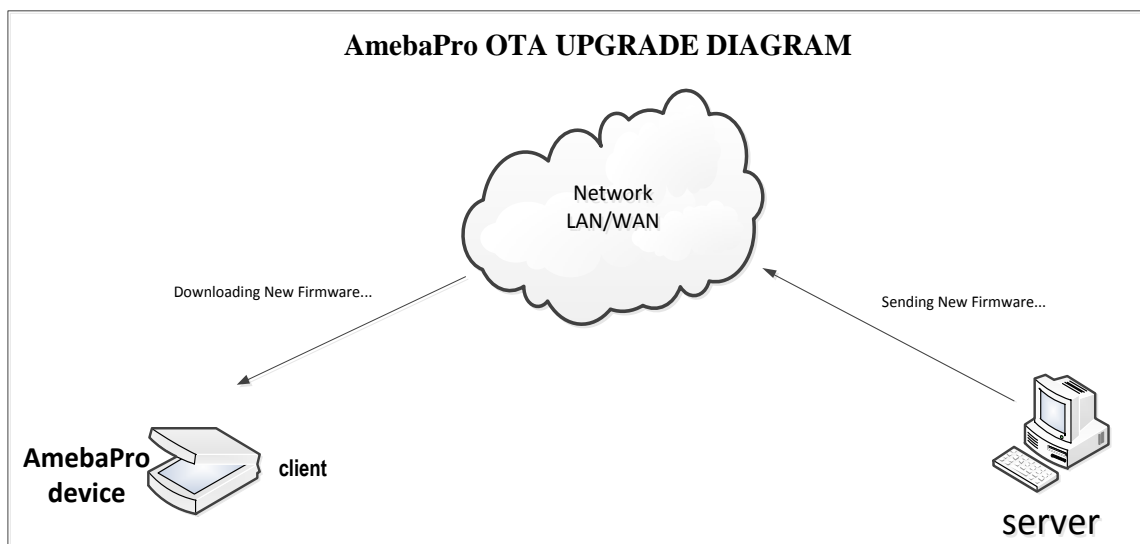
- flash_api.c
- flash_fatfs.c
- hal_flash.c

5.5 計算系統記憶體使用量

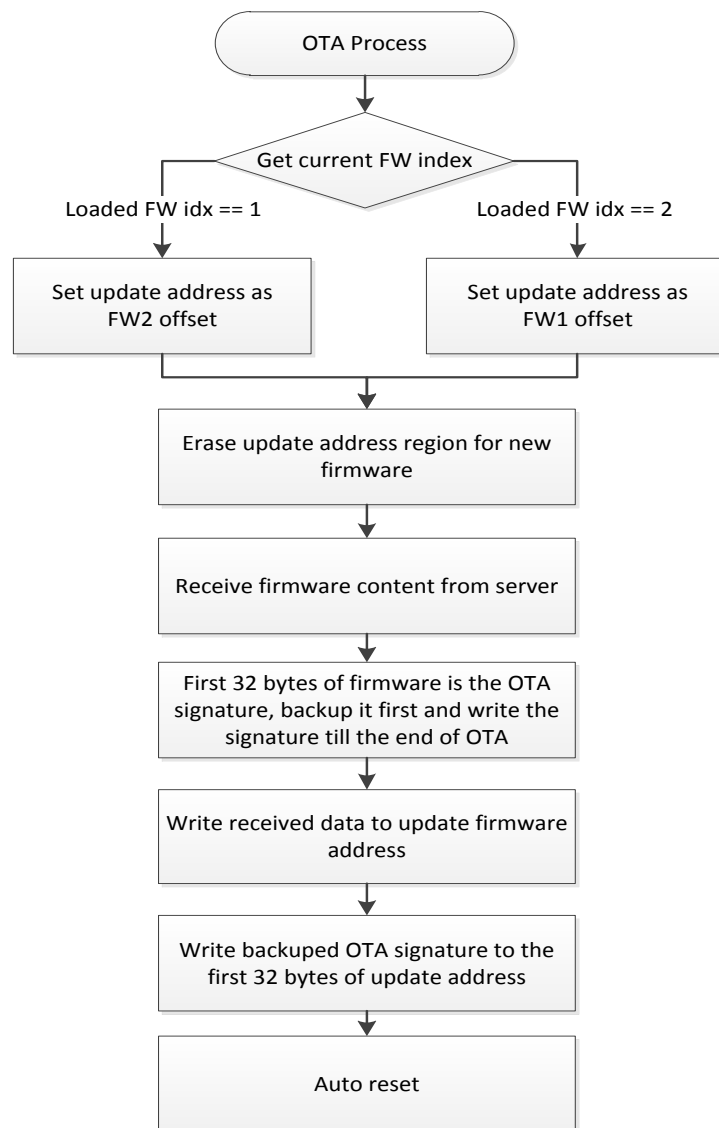
請參考 UM0070.

6 OTA

Over-the-air (OTA) 提供了一種經由 TCP/IP 網路連線更新裝置固件的方式。

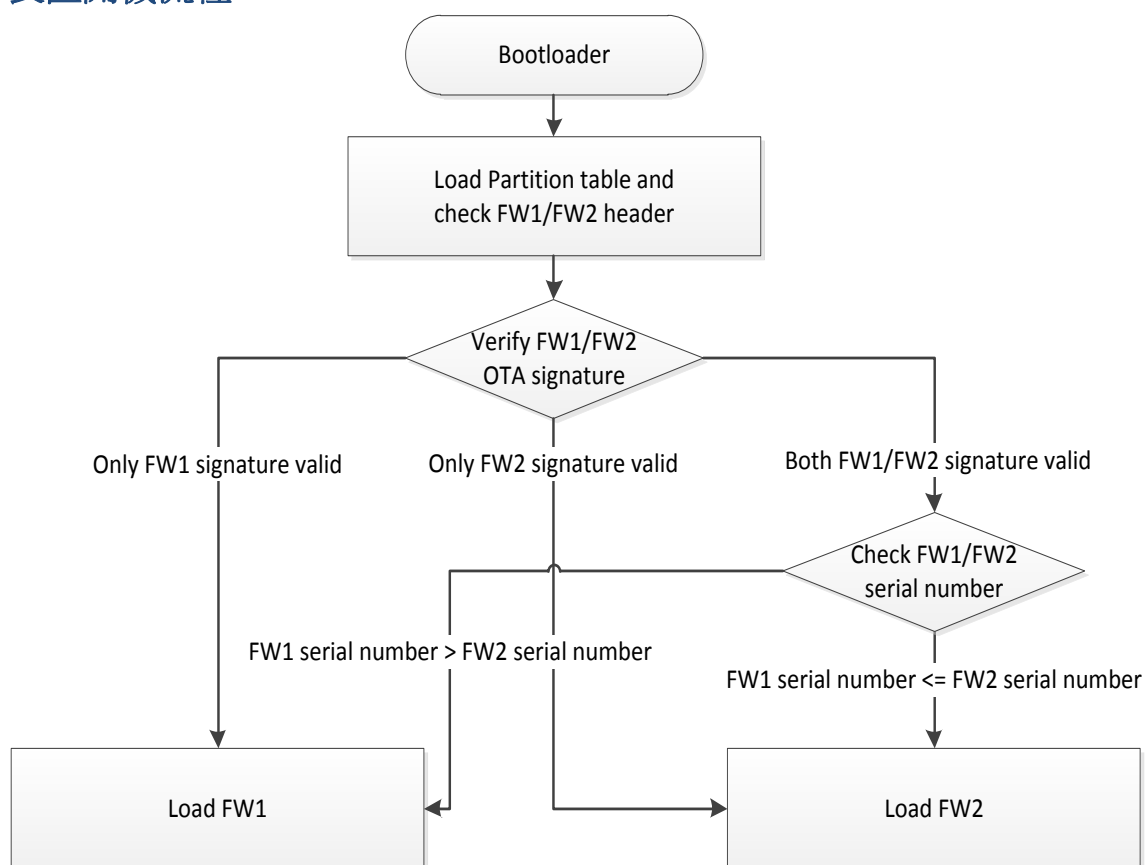


6.1 OTA 執行流程



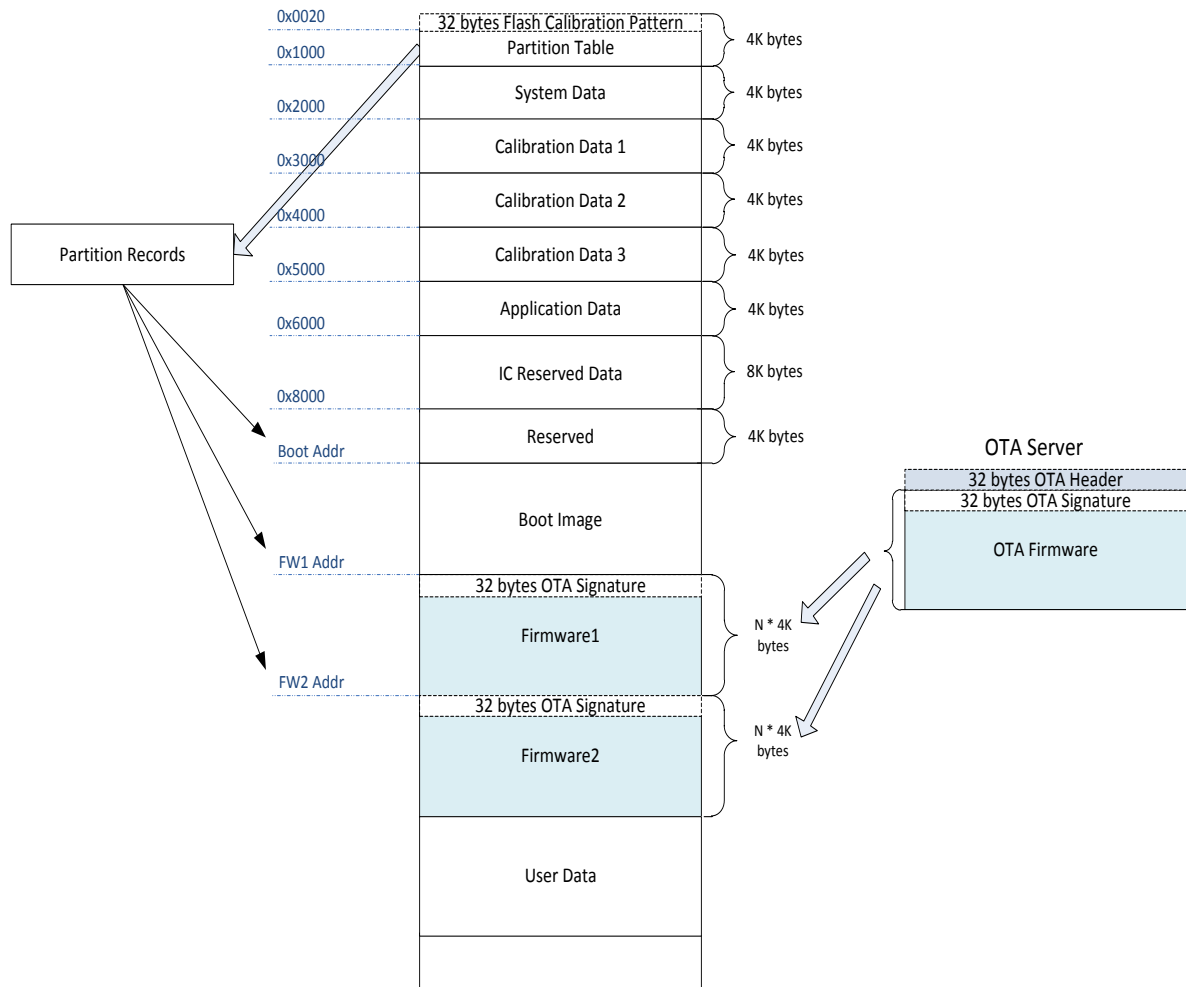
 在步驟” Write received data to update firmware address ”中，32 bytes 長的 OTA signature 需先設為非法值 0xff，正確的 OTA signature 需要在整個 OTA 流程結束時才做寫入，以避免裝置由不完整的固件開機而造成錯誤。

6.2 裝置開機流程



Bootloader 會依據 serial number 選擇最新的固件做開機。

6.3 目標更新區塊



在 AmebaPro OTA 更新流程中，Firmware1 與 Firmware2 會做交替更新，此兩固件的起始位址記錄在 partition records 內，可在 `project\realtek_amebapro_v0_example\EWARM-RELEASE\partition.json` 內找到其定義，並依據您的固件大小做調整，確保兩固件在 flash 上不會互相重疊。另外由於 AmebaPro 使用 XIP remapping 機制，請確保兩固件起始位址為 256KB aligned 位址，以避免出現 XIP 錯誤，例如 0x40000 或 0x240000 皆為 256KB aligned。

```

"fw1":{
  "start_addr" : "0x40000",
  "length" : "0x200000",
  "type": "FW1",
  "dbg_skip": false,
  "hash_key": "000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
},

```

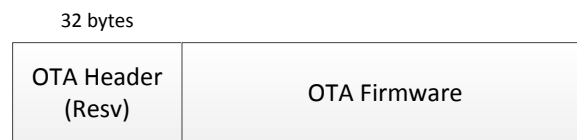
```
"fw2":{
  "start_addr" : "0x240000",
  "length" : "0x200000",
  "type": "FW2",
  "dbg_skip": false,
  "hash_key": "000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
}
```

6.4 生成 OTA 固件

在 AmebaPro 專案編譯成功後，會生成 `ota_is.bin`，此固件即為上述的 OTA Firmware，在 OTA 流程中遠端 OTA 伺服器所傳送的資料即為該固件的檔案內容，裝置在執行 OTA 時即會判斷欲更新的區塊而燒錄對應的 OTA Firmware 至 flash。

6.4.1 OTA 固件格式

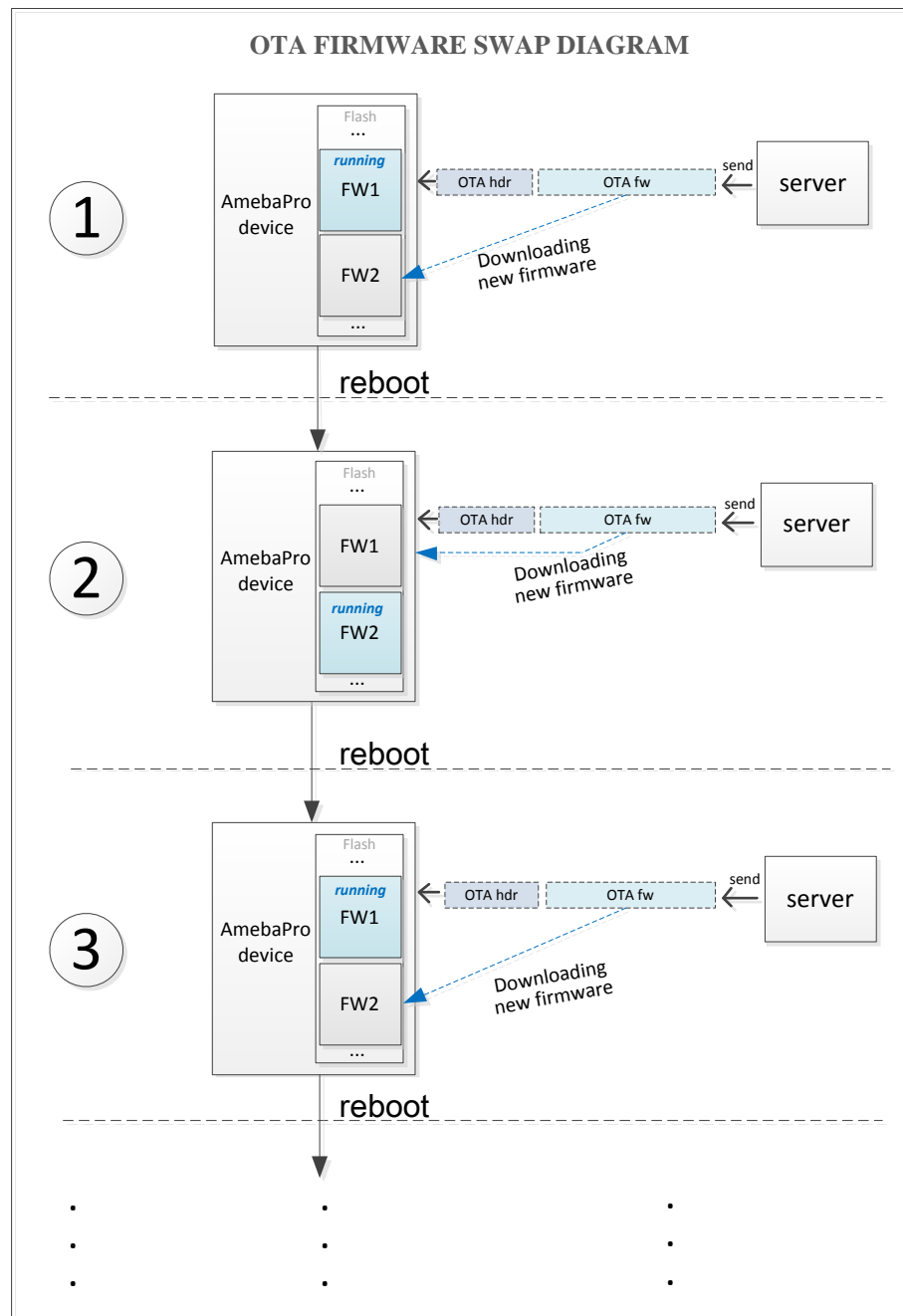
OTA 伺服器使用的 `ota_is.bin` 格式如下：



`ota_is.bin` 包含了 OTA 升級的固件 OTA Firmware，並在前頭帶上了 32 bytes 長度的 OTA Header，目前 OTA Header 皆為 reserved 區域，可作為未來擴充使用。

6.4.2 OTA 更新切換固件行為

裝置執行 OTA 更新時，固定會更新至非目前執行的 OTA 區塊，若更新後的固件皆使用較新的 serial number 值，重覆執行 OTA 更新應會如下圖行為。



6.4.3 OTA 固件編譯設定

裝置開機時會依據 OTA Firmware 的 serial number 決定開機順序，在編譯專案生成 OTA 固件時需對此做設定，確保裝置更新後會以新的固件開機。

6.4.3.1 Serial number

AmebaPro 在開機時會依據固件的 serial number 大小決定以哪塊固件做開機，因此在編譯專案時，請先確認 serial number 有做正確設定。設定 serial number 方式如下：

Step 1: 固件的 serial number 配置會以固件內第一塊 image 的 serial number 設定值為依據，固件內的 image 排放順序可參考

project\realtek_amebapro_v0_example\EWARM-RELEASE\amebapro_firmware_is.json 。

```
"FIRMWARE":{
  "images":[
    {"img": "XIP", "offset":"0x00"},
    {"img": "ISP", "offset":"0x00"},
    {"img": "CINIT", "offset":"0x00"},
    {"img": "FWLS", "offset":"0x00"},
    {"img": "FWHS", "offset":"0x00"},
    {"img": "WLAN", "offset":"0x00"},
    {"img": "WOWLAN", "offset":"0x00"}
  ]
}
```

以上圖為例，XIP image 在 images 序列最上方，因此此固件的 serial number 設置與 XIP image 內的 serial number 配置相同。

Step 2: 修改第一塊 image 的 serial number，如上例需將正確的 serial number 值修改至 XIP image 設定內。

```
"XIP": {
  "source": "Debug/Exe/application_is.out",
  "header": {
    "next": null,
    "__comment_type": "Support
Type:PARTAB,BOOT,FWHS_S,FWHS_NS,FWLS,ISP,VOE,WLN,DTCM,ITCM,SRAM,ERAM,XIP,MO,CPFW",
    "type": "XIP",
    "enc": false,
    "__comment_pkey_idx": "assign by program, no need to configure",
    "serial": 100
  },
}
```

Serial number 為 4 byte 大小且由數值 0 開始代表有效值，請將此值依據您預期的版號自行做修改。

Step 3: 編譯專案後，資料夾下應自動生成 OTA Firmware 固件

SDK_folder/project/project_name/EWARM-RELEASE/Debug/Exe/ota_is.bin，serial number 設定也包含在此固件中。

6.5 透過 Wi-Fi 實作 OTA 更新

6.5.1 基於 socket 的 OTA 更新範例

此範例示範裝置如何透過網路 socket 執行 OTA 更新。
在此範例中，請確保裝置與電腦皆連上相同區域網路。

6.5.1.1 生成 OTA 範例固件

開啓 OTA 指令

指令開關定義在 `\project\realtek_amebapro_v0_example\inc\platform_opts.h`。

```
#define CONFIG_OTA_UPDATE 1
```

確認當前固件 index 並取得預計更新位址

```
/* ota_8195b.c */
uint32_t update_ota_prepare_addr(void){
    uint32_t NewFWAddr;
    fw_img_export_info_type_t *pfw_image_info;

    pfw_image_info = get_fw_img_info_tbl();

    printf("WnWr [%s] Get loaded_fw_idx %dWnWr", __FUNCTION__, pfw_image_info->loaded_fw_idx);
    if(pfw_image_info->loaded_fw_idx == 1)
        NewFWAddr = pfw_image_info->fw2_start_offset;
    else if(pfw_image_info->loaded_fw_idx == 2)
        NewFWAddr = pfw_image_info->fw1_start_offset;
    else {
        printf("WnWr [%s] Unexpected index %d", __FUNCTION__, pfw_image_info->loaded_fw_idx);
        return -1;
    }

    printf("WnWr [%s] NewFWAddr %08XWnWr", __FUNCTION__, NewFWAddr);
    return NewFWAddr;
}
```

6.5.1.2 設置 local download server

Step 1: 生成目標更新文件 ota_is.bin 並放在 tools\DownloadServer 資料夾下。

Step 2: 修改 start.bat 檔：Port = 8082, file = ota_is.bin

```
@echo off
DownloadServer 8082 ota_is.bin
set /p DUMMY=Press Enter to Continue ...
```

Step 3: 執行 start.bat。

```
c():checksum 0x84dd63b
Listening on port (8082) to send ota_is.bin (1372256 bytes)

waiting for client ...
```

6.5.1.3 執行 OTA 更新流程

裝置連線至 AP 後，輸入指令：ATWO=IP[PORT]

```
#ATWO=192.168.0.107[8082]
[ATWO]: _AT_WLAN_OTA_UPDATE_

[MEM] After do cmd, available heap 33446656

#
[update_ota_local_task] Update task start
[update_ota_prepare_addr] Get loaded_fw_idx 1
[update_ota_prepare_addr] NewFWAddr 00240000

[update_ota_local_task] Read info first
[update_ota_local_task] info 12 bytes
[update_ota_local_task] tx file size 0x14f060
[update_ota_erase_upg_region] NewFWLen 1372224
[update_ota_erase_upg_region] NewFWBlkSize 336 0x150
[update_ota_local_task] Start to read data 1372224 bytes

[update_ota_local_task] sig_backup for 32 bytes from index 0
.....
.....
.....
.....
Read data finished

[update_ota_signature] Append OTA signature
[update_ota_signature] signature:
 3C 9D D9 9F E9 30 C1 17 D5 97 E1 9B 35 46 09 77
 5B D4 FD B0 96 7A 81 31 1E 69 B3 F4 BD FE D2 A5
[update_ota_local_task] Update task exit
[update_ota_local_task] Ready to reboot
== Rtl8195bh IoT Platform ==
```

Local download server 成功訊息視窗：

```
c():checksum 0x84dd63b
Listening on port (8082) to send ota_is.bin (1372256 bytes)
```

```
waiting for client ...
Accept client connection from 192.168.0.110
Send checksum and file size first
Send checksum byte 12
Sending file...
```

[illegible]

```
Total send 1372256 bytes
Client Disconnected.
Waiting for client ...
```

在正確更新固件後，裝置會自動重新開機，此時 **bootloader** 會以新版固件開機。

6.5.2 基於 HTTP 的 OTA 更新範例

此範例示範裝置如何透過 HTTP 協議執行 OTA 更新，在此範例中 HTTP server 會在 HTTP response 內將固件內容作為 data 區塊傳至裝置。

在此範例中，請確保裝置與電腦皆連上相同區域網路。

6.5.2.1 生成 OTA 範例固件

開啓 OTA 範例

此範例開關定義在 `\project\realtek_amebapro_v0_example\inc\platform_opts.h` 與 `\component\soc\realtek\8195b\misc\platform\ota_8195b.h`。

```
/* platform_opts.h */
#define CONFIG_OTA_UPDATE      1
#define CONFIG_EXAMPLE_OTA_HTTP 1
```

```
/* ota_8195b.h */
#define HTTP_OTA_UPDATE
```

定義 Server IP 與 Port

(In `\component\common\example\ota_http\example_ota_http.c`)

```
#define PORT      8082
#define IP        "192.168.0.107"
#define RESOURCE  "ota_is.bin"
```

Example:

SERVER: <http://m-apps.oss-cn-shenzhen.aliyuncs.com/051103061600.bin>

Setting: `#define PORT 80`
 `#define HOST "m-apps.oss-cn-shenzhen.aliyuncs.com"`
 `#define RESOURCE "051103061600.bin"`

確認當前固件 index 並取得預計更新位址

```
/* ota_8195b.c */
uint32_t update_ota_prepare_addr(void){
    uint32_t NewFWAddr;
    fw_img_export_info_type_t *pfw_image_info;

    pfw_image_info = get_fw_img_info_tbl();

    printf("WnWr [%s] Get loaded_fw_idx %dWnWr ", __FUNCTION__, pfw_image_info->loaded_fw_idx);
    if(pfw_image_info->loaded_fw_idx == 1)
        NewFWAddr = pfw_image_info->fw2_start_offset;
    else if(pfw_image_info->loaded_fw_idx == 2)
        NewFWAddr = pfw_image_info->fw1_start_offset;
    else {
        printf("WnWr [%s] Unexpected index %d", __FUNCTION__, pfw_image_info->loaded_fw_idx);
        return -1;
    }

    printf("WnWr [%s] NewFWAddr %08XWnWr ", __FUNCTION__, NewFWAddr);
    return NewFWAddr;
}
```

與 HTTP download server 溝通

1. 在 http_update_ota_task() 函式中，裝置會傳送 HTTP request “GET /RESOURCE HTTP/1.1\r\nHost: host\r\n\r\n” 至 server。
2. Local download server 在收到此 request 後，會傳回 response 並在此 response 中將固件大小作為 “Content-Length”，而固件內容則是放在 data 區塊。
3. 裝置收到 HTTP response 後，會依據 content length 決定當前固件傳輸是否完成。

6.5.2.2 設置 local HTTP download server

Step 1: 生成目標更新文件 firmware_is.bin 並放在 tools\DownloadServer(HTTP)資料夾下。

Step 2: 修改 start.bat 檔：Port = 8082, file = ota_is.bin。

```
@echo off
DownloadServer 8082 ota_is.bin
set /p DUMMY=Press Enter to Continue ...
```

Step 3: 執行 start.bat。

```
<Local HTTP Download Server>
Listening on port (8082) to send ota_is.bin (1365984 bytes)

waiting for client ...
```

6.5.2.3 執行 OTA 更新流程

將裝置重新開機並連上 AP，一分鐘過後裝置應會自動執行 HTTP OTA 更新流程。

```
#
[update_ota_prepare_addr] Get loaded_fw_idx 1
[update_ota_prepare_addr] NewFWAddr 00240000

[http_update_ota] Download new firmware begin, total size : 1365984
[update_ota_erase_upg_region] NewFWLen 1365952
[update_ota_erase_upg_region] NewFWBlkSize 334 0x14e.
[http_update_ota] sig_backup for 32 bytes from 0 index
.....
[http_update_ota] Download new firmware 1365952 bytes completed

[update_ota_signature] Append OTA signature
[update_ota_signature] signature:
47 8C 7E E0 31 01 F1 FA 5E 81 58 88 B9 70 20 65
D5 91 B1 CA 3A 2E F1 9B D5 BD 13 54 09 F1 11 5A
[http_update_ota_task] Update task exit
[http_update_ota_task] Ready to reboot
== Rt18195bh IoT Platform ==
```


Local download server 成功訊息視窗：

[illegible]

在正確更新固件後，裝置會自動重新開機，此時 **bootloader** 會以新版固件開機。

6.6 OTA signature

透過 UART AT 指令清除或回復 OTA signature 部分請參考 AN0025 文件。

7 電源管理

7.1 AmebaPro Power Save Modes

AmebaPro 提供多種電源模式，表 7.1 描述這些電源模式之情形 DeepSleep 和 Standby：

Mode	DeepSleep	Standby
Initiator/domain	LS	LS
WLAN wakeup	No	Yes
STimer wakeup	Yes	Yes
GTimer wakeup	No	Yes
HSTimer wakeup	No	Yes
GPIO wakeup	Yes (GPIOA_13)	Yes (GPIOA group)
ADP wakeup	Yes	No
RTC wakeup	Yes	No
PWM wakeup	No	Yes
UART wakeup	No	Yes
MII wakeup	No	No
I2C wakeup	No	Yes
ADC wakeup	No	Yes
COMP wakeup	No	Yes
USB wakeup	No	No
SDIO wakeup	No	No
SGPIO wakeup	No	Yes

Table 7.1 AmebaPro Power mode and wakeup source

有一些功能需要注意：

- DeepSleep 可在所有電源模式中節省最多電量
- 醒來後 DeepSleep 和 Standby 從初始運行代碼，而不是從之前呼叫睡眠程序的位置運行代碼。

7.1.1 DeepSleep

DeepSleep 可在所有電源模式中節省最多電量。HS 已關閉，LS 僅保留喚醒源所需的資源。Deepsleep 需要從 LS 調用 DeepSleep。如果用戶想要從 HS 調用 DeepSleep，他需要設計自己的 ICC 命令/消息並實做相關代碼。

7.1.2 Standby

在待機模式下 HS 關閉，LS 僅保留喚醒源所需的資源。

除 WLAN 之外的大多數喚醒源，功耗是穩定的，並且可以從具有低取樣率的電流表測量。

如果用戶選擇從 WLAN 喚醒，用戶需要在呼叫待機前配置 WLAN。有關更多信息，請參閱 [“Wakeup from WLAN”](#)。

Standby 需要從 LS 調用。如果用戶想要從 LS 調用 Standby，他需要設計自己的 ICC 命令/消息並實現相關代碼。

7.1.3 Wakeup from WLAN

如果使用者選擇在待機模式或 SleepPG 模式下從 WLAN 喚醒，則需要在調用 Standby 之前配置 wlan。

用戶配置 wlan 並調用 Standby 後，系統將進入睡眠狀態。在這種狀態下，LS MCU 會定期喚醒/睡眠以檢查 wlan 狀態並決定是否需要喚醒 HS MCU。在 wlan 接收到與喚醒模式匹配的喚醒數據包之後，LS MCU 將喚醒 HS

完整流程如下圖所示：

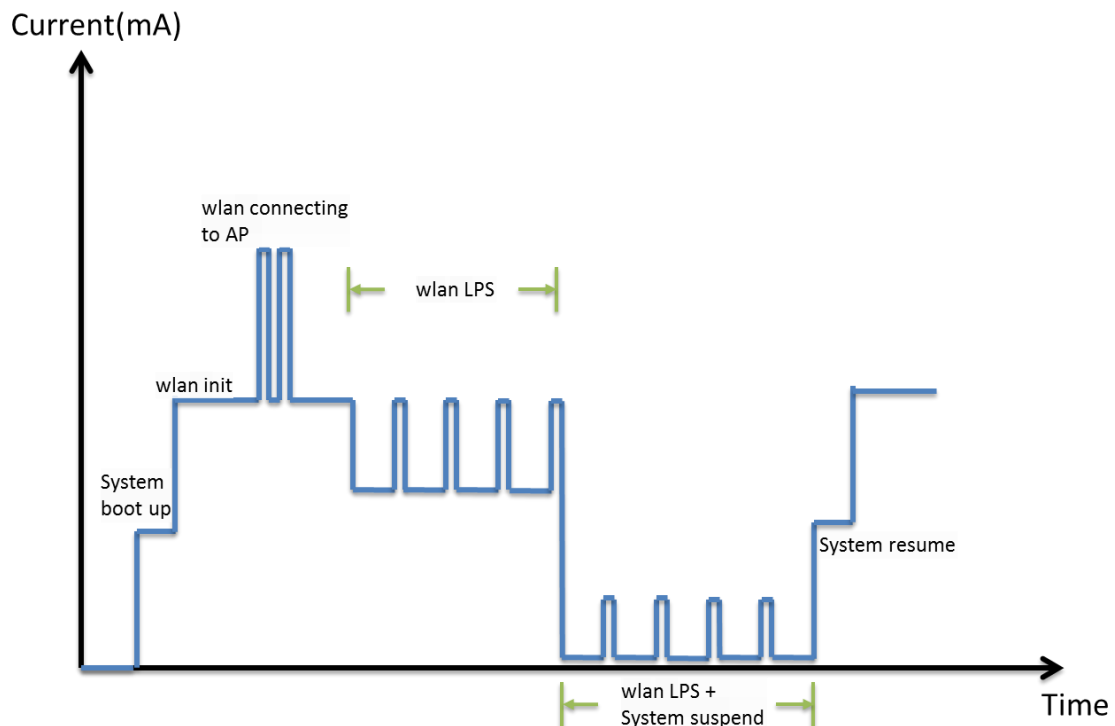


Diagram 7.1 power consumption of wake on wlan

1. 首先啓動系統，初始化 wlan，連接到 AP。
2. 連接到 AP 後，如果沒有繁重的數據流量，wlan 將進入 LPS 狀態。在 LPS 狀態下，wlan 將每隔 100ms 監聽訊號（如果 DTIM 爲 1）。所以你可以看到每 100 毫秒的功耗上升。在 wlan 接收到信標並且 TIM 段沒有該設備的數據包之後功耗將下降，然後 wlan 將關閉 RF 並嘗試保持低功率狀態。
3. 如果用戶試圖透過 wlan associate idle 使系統節省更多電量，可以調用 Standby。可以看到系統睡眠的 wlan LPS 中的功耗下降更多。

請注意，如果想測量系統在使用 wlan LPS 時暫停時的功耗，您必須確保電源或電流表的穩壓器能夠處理電壓降並在數百微安和幾十毫安之間變化。

7.1.4 遠端喚醒功能

當用戶需要使用遠端喚醒功能，讓系統能在 wlan 接收到匹配的數據包時脫離 SleepPG 模式，SDK 預設已配置 ICMP Pattern 喚醒，並支援用戶設置自訂 Pattern。Wakeup pattern 比對的部分包含 MAC Header 中的 Destination、BSSID、Source 及從 LLC Header 的 Protocol Type 到 IP Header 的 Destination IP Address 這串數據。若要設置自訂 Pattern，用戶需要設置(1)Pattern 內容和(2)Mask: Pattern 內容中要比對的 Byte，以上兩項需設置在 wowlan_pattern_t 結構中:

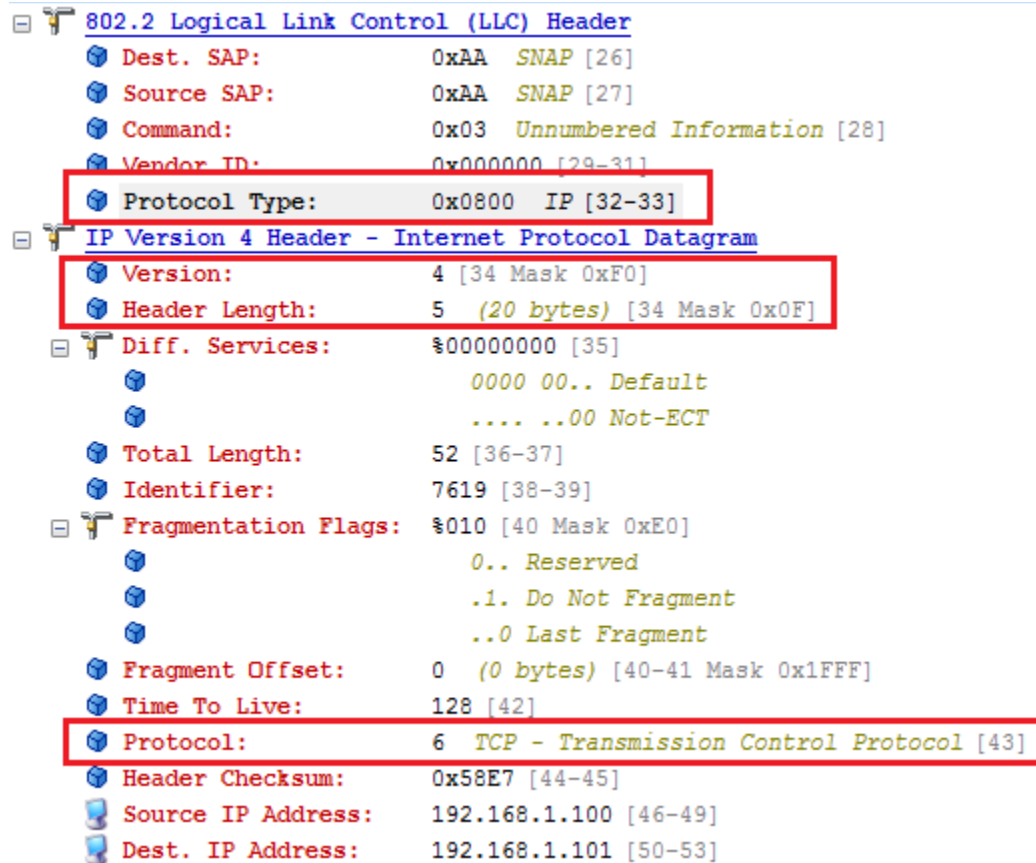
```
typedef struct wowlan_pattern {
    unsigned char eth_da[6];
    unsigned char eth_sa[6];
    unsigned char eth_proto_type[2];
    unsigned char header_len[1];
    unsigned char ip_proto[1];
    unsigned char ip_sa[4];
    unsigned char ip_da[4];
    unsigned char mask[5];
} wowlan_pattern_t;
```

(1) Pattern

(2) Mask

設置完 pattern 及 mask 後，需要使用 wifi_wowlan_set_pattern API 設定到 wlan。以 TCP 數據包為例，假設 Ameba MAC address 爲 00:E0:4C:87:00:00，以下說明設定接收者爲 Ameba 的 TCP Unicast 數據包做爲喚醒包的方式(完整 Example 可參考 7.2.2 Example: Custom wake on wlan pattern)。首先，需要比對數據包 MAC Destination 爲 Ameba，因此把 wowlan_pattern_t 的 eth_da 欄位設置爲 00:E0:4C:87:00:00。接下來設定 LLC Header 中的 Protocol type 爲 IP Protocol: {0x08,

0x00}，Version+Length: {0x45}，IP Header 的 Protocol type 為 TCP: {0x06}，並把 Destination IP 設置為 Ameba 的 IP。這些欄位可參考 TCP 封包格式：



以上欄位設置後會被轉換為 HW 比對的格式，如下

eth_da (6)	eth_sa (6)	eth_proto_type (2)	header_len (1)	Rsvd (8)	ip_proto (1)	Rsvd (2)	ip_sa (4)	ip_da (4)
---------------	---------------	-----------------------	-------------------	-------------	-----------------	-------------	--------------	--------------

因此此例 TCP Pattern 轉換後的 HW Pattern 為：

00 e0 4c	00 00 00	08 00	45	00 00 00 00	06	00 00	00 00 00 00	c0 a8 01 65
87 00 00	00 00 00			00 00 00 00				

接下來用戶需設定 Mask，Mask 中的 1 bit 對應 HW pattern 的 1 byte，Mask 為 1 的 bit 對應到的 byte 才會加入比對。以下說明 Mask 的組成方式：

- 先用 bit sequence mask 出需要比對的 Byte:

111111	000000	11	1	00000000	1	00	0000	1111
--------	--------	----	---	----------	---	----	------	------

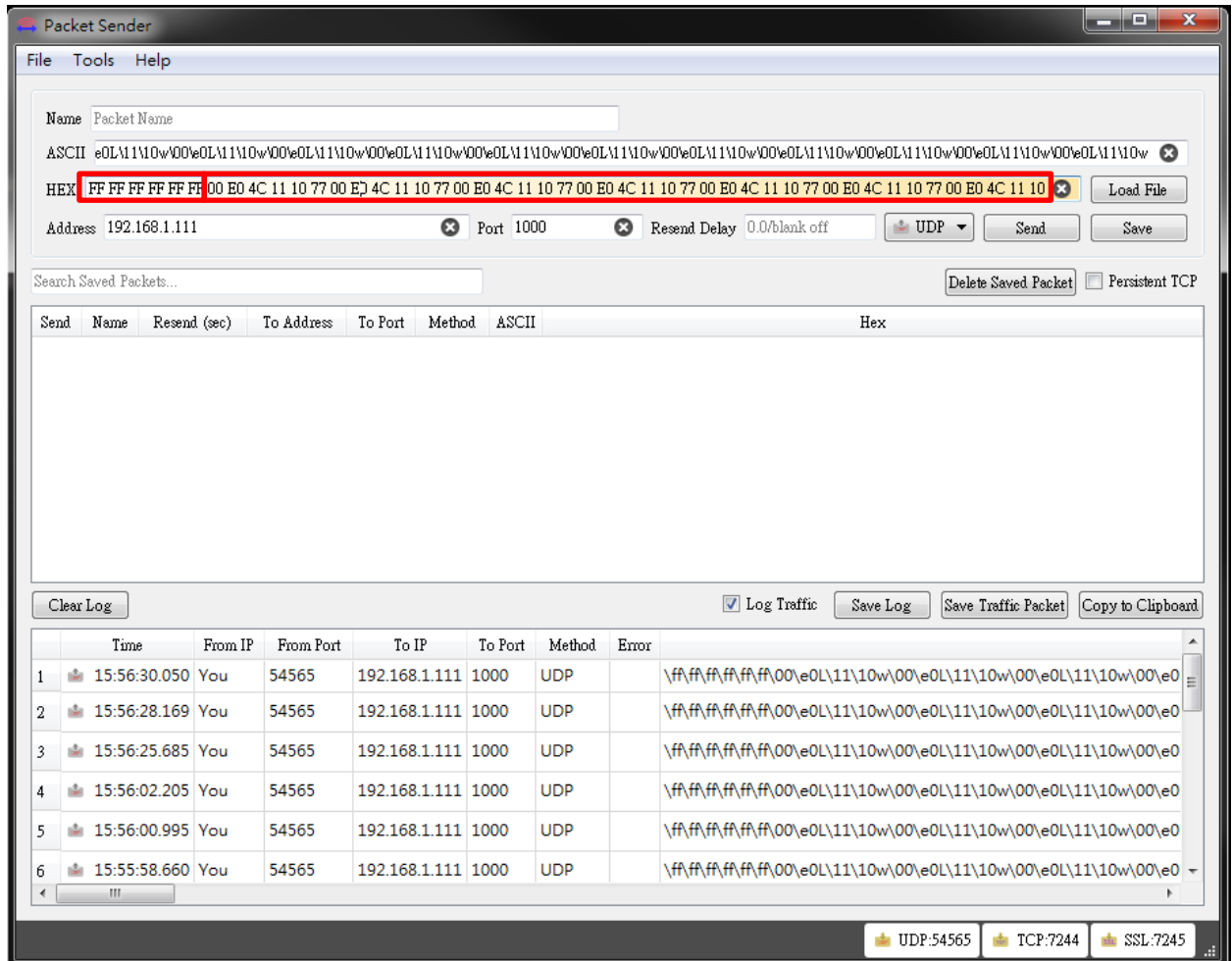
- 將這個 bit sequence 每 8 bit 組成 1byte:

11111100	00001110	00000001	00000011	11
----------	----------	----------	----------	----

- HW 是從每個 byte 的 bit0 開始比對，因此每個 byte 的 bit 順序要反過來

00111111	01110000	10000000	11000000	00000011
----------	----------	----------	----------	----------

- 由第 3 步轉換為 Hex: {0x3f, 0x70, 0x80, 0xc0, 0x03}，得到最後的 Mask



"\project\realtek amebapro v0 example\example sources\power save\"

將 HS 和 LS 文件夾複製到項目文件夾，編譯 LS 和 HS 代碼並下載程序。

7.2.1.3 Deepsleep

在 HS 輸入底下的 command

PS=deepsleep

在 LS 的 log 裡會出現

deepsleep wake event:0x0001

此時如果有將開發板接上電流計，將會看到電流變化，在 60 秒鐘後系統會醒來，HS 與 LS 都像是被重啓般從頭開始執行。LS 的 log 裡，wake event 代表此次設置的喚醒源，請參考“power_mode_api.h”參考對應的喚醒源。

在 HS 輸入底下的 command

PS=deepsleep,stimer=10

此為設置 Deepsleep 由 stimer 喚醒，時間是 10 秒

在 HS 輸入底下的 command

PS=deepsleep,gpio

此為設置 Deepsleep 由 GPIOA_13 喚醒

7.2.1.4 Standby

在 HS 輸入底下的 command

PS=standby

在 LS 的 log 裡會出現

Standby wake event:0x0001

此時如果有將開發板接上電流計，將會看到電流變化，在 60 秒鐘後系統會醒來，HS 與 LS 都像是被重啓般從頭開始執行。LS 的 log 裡，wake event 代表此次設置的喚醒源，請參考“power_mode_api.h”參考對應的喚醒源。

系統醒來後，LS 的 log 裡會顯示醒來的原因

wake from AON_TIMER

在 HS 輸入底下的 command

PS=standby,stimer=10

此為設置 Standby 由 stimer 喚醒，時間是 10 秒

在 HS 輸入底下的 command

PS=standby,gpio=2

此為設置 Standby 由 GPIOA_2 喚醒

在 HS 輸入底下的 command

PS=standby,stimer=10,stimer_wake=0

此為設置 Standby 由 stimer 喚醒, 時間是 10 秒。當 10 秒到了, LS 並不會喚醒 HS, LS 會調用 Standby 並進入睡眠。這個 command 展示如何在系統睡眠時, 週期性地執行一些工作

在 HS 輸入底下的 command

PS=standby,stimer=10,stimer_wake=5

此為設置 Standby 由 stimer 喚醒, 時間是 10 秒。當 10 秒到了, LS 並不會喚醒 HS, LS 會調用 Standby 並進入睡眠。重覆此動作 5 次之後, LS 才會喚醒 HS。也就是說, LS 總共睡眠了 5 次, 每次 10 秒, 而 HS 總共睡眠了 1 次, 時間為 50 秒。

在 HS 輸入底下的 command

PS=standby,gpio=2,gpio_wake=0

此為設置 Standby 由 GPIOA_2 喚醒, 當我們觸發 GPIOA_2 的中斷並喚醒 LS, LS 並不會喚醒 HS, LS 會調用 Standby 並進入睡眠。這個 command 展示當系統睡眠時, 如果 PIR 或按鈕喚醒 LS, LS 可以檢查這些狀態而不用喚醒 HS

在 HS 輸入底下的 command

PS=standby,gpio=2,gpio_wake=5

此為設置 Standby 由 GPIOA_2 喚醒, 當我們觸發 GPIOA_2 的中斷並喚醒 LS, LS 並不會喚醒 HS, LS 會調用 Standby 並進入睡眠。重覆此動作 5 次之後, LS 才會喚醒 HS。也就是說, LS 總共被喚醒了 5 次, 而 HS 總共被喚醒了 1 次

7.2.1.5LS wake reason

當系統從 Standby 睡下去之後, 醒來時, HS 也許需要知道是被什麼喚醒源喚醒, 可以在 HS 輸入底下的 command

PS=ls_wake_reason

在 HS 的 log 會顯示

```
wake from AON_TIMER
```

7.2.1.6Wake on wlan

在 HS 輸入底下的 command


```
PS=wowlan,your_ssid,your_pw
```

系統會嘗試連線至指定的 ssid, 連線上後約 16 秒會進入 wake on wlan 模式, 目前預設使用 Standby 並且喚醒源是 WLAN。

進入 wake on wlan 模式後, LS 會不斷被重啓, 這是對應系統需要聽 AP 的 beacon。當 wlan 收到喚醒的封包後(預設是 PING), 會將系統喚醒。

在 HS 輸入底下的 command

```
PS=standby,wowlan,stimer=60
```

```
PS=wowlan,your_ssid,your_pw
```

第一個 command 與之前的 standby command 相似, 除了它在 standby 的參數裡增加了 wowlan。如此可以設置 Standby 相關的參數但不會進入睡眠。

第二個 command 與前面的 command 相同。

輸入這兩個 command 之後, LS 可以被兩種喚醒源喚醒: stimer 與 wlan。如果 LS 在 60 秒內並會收到 wlan 收到喚醒封包的中斷, 那麼 LS 將會被 stimer 喚醒, 並且 LS 會喚醒 HS

在 HS 輸入底下的 command

```
PS=standby,wowlan,stimer=5,stimer_wake=0
```

```
PS=wowlan,your_ssid,your_pw
```

這些 command 與前面的相似, 除了 stimer 每 5 秒會喚醒 LS, 並且 LS 會直接調用 Standby。這些 command 展示在 wake on wlan mode, LS 每 5 秒醒來檢查狀態。

在 HS 輸入底下的 command

```
PS=standby,wowlan,stimer=5,stimer_wake=0,gpio=2
```

```
PS=wowlan,your_ssid,your_pw
```

這些 command 與前面的相似, 它可以被 3 個喚醒源喚醒: stimer, gpio, 與 wlan。在 wake on wlan mode 裡, LS 每 5 秒被 stimer 喚醒, 但不會喚醒 HS。但如果 GPIO 中斷被觸發, 那麼 LS 將會喚醒 HS。

這些 command 展示了在 wake on wlan mode 底下, LS 定期檢查狀態, 並且如果使用按按鈕將會喚醒系統。

7.2.1.7 Wake on wlan with wakeup pattern

在 HS 輸入底下的 command

```
PS=wowlan_pattern
```

```
PS=wowlan,your_ssid,your_pw
```

第一個 command 將會設置一組 wake on wlan pattern, 這組 pattern 將會比對 TCP 封包, 檢查它的 destination port 是否為 80。也就是說, 在同個 router 底下, 如果使用者打開瀏覽器, 並且輸入 STA 的 IP, 那麼這個 HTTP 請求(也就是 TCP SYN 封包)將會喚醒 STA

7.2.1.8 TCP keep alive

在 HS 輸入底下的 command 可以設置 TCP keep alive

```
PS=tcp_keep_alive,192.168.1.100,5566
```

```
PS=wowlan,your_ssid,your_pw
```

第一個 command 裡, 後面兩個參數是 tcp server 的 IP 與 port, 請依據實際情況填寫。設置完後, 它會在系統進 wake on wlan 之後才會生效。預設是每 30 秒送一個 TCP 封包, 如果 STA 沒有收到 TCP ACK, 那麼 STA 會在 10 秒後重送。你可以在代碼裡修改這個間隔:

```
static uint32_t interval_ms = 30000;  
static uint32_t resend_ms = 10000;
```

在代碼裡, 它還加了一組 wake on wlan pattern, 這組 pattern 將會比對 TCP 封包, 檢查它的 source port 是否與 tcp keep alive 設置的 port 相同, 並檢查 TCP flag 是否為 PSH+ACK。這組 pattern 可以讓 TCP server 端主動送封包時將 STA 喚醒。

7.2.2 Example: Custom wake on wlan pattern

7.2.2.1 Abstract

在此範例中 HS 啟動一個線程以連接到 AP。連接到 AP 後 HS 將先設置自訂喚醒 pattern(TCP packets to destination port 80), 等待 4 秒後將 default pattern(ICMP)及前面設置的自訂 pattern(TCP, port 80)設定到 wlan, 並在 Standby 中進入系統睡眠。

7.2.2.2 Setup example

範例位於

project\realtek_amebapro_v0_example\example_sources\pm_wake_on_wlan_pattern\
將 HS 文件夾複製到項目文件夾。

燒錄及上傳程式至 AmebaPro 之後, 你將可以在 console 看到底下的 log:

```
wait for wifi connection...  
please use AT commands (ATW0, ATWC) to make wifi connection
```

請用 AT command 做連線, 完成連線後 4 秒系統便會進入 wake on wlan mode, 並看到 log 停在

`RTL871X: -rtw_hal_set_fw_wow_related_cmd()-`

此時可以發 TCP 測試封包給 Ameba，應該可以看到 Ameba 被喚醒。

7.2.2.3 測試方式

以下提供幾種發測試封包的方式。

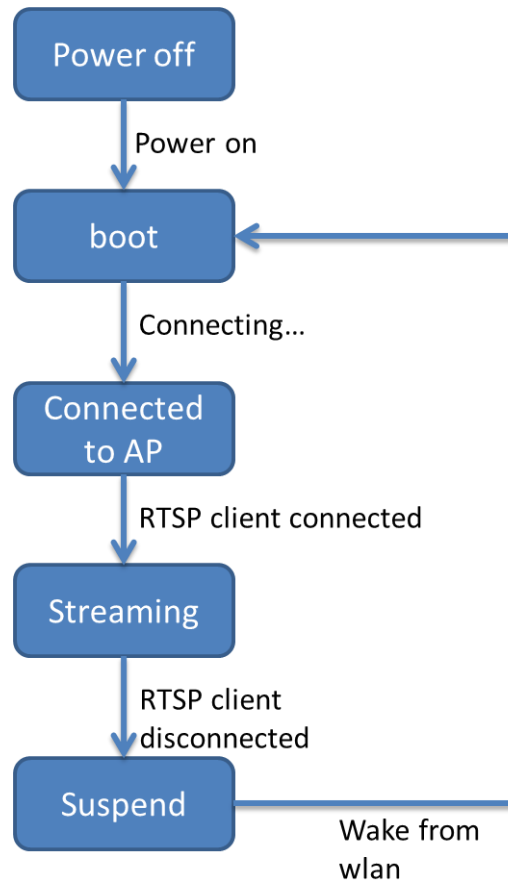
- 使用瀏覽器
此範例設定的 pattern 是 TCP, destination port 80 的封包，也就是 HTTP 封包，因此可以在 PC 瀏覽器網址列輸入 Ameba 的 IP address，即可發送 HTTP 封包到 Ameba。
- 使用 Packet Sender 工具
當設定比較複雜的 pattern 時，可下載 Packet Sender 工具:
<https://packetsender.com/>，發送自訂內容的封包。

7.2.3 Example: system suspend with wake on wlan after streaming stop

7.2.3.1 Abstract

這個範例表示了(1)如何在啟動/恢復後獲得第一個 video frame，以及(2)如何使系統暫停並能夠從 wlan 喚醒。

Life cycle 如下圖所示：



- 首先，系統從上電啟動或從 wlan 喚醒（稍後描述）
- AmebaPro 嘗試通過快速連接功能或 AT 命令連接到 AP。
- AmebaPro 連接到 AP 後，AmebaPro 將啟動 RTSP 服務器並等待 RTSP 客戶端。
- 用戶可以使用 VLC 或其他 RTSP 客戶端連接到 AmebaPro。用戶應該看到啟動後 buffer 的串流內容。因此，用戶可以評估從啟動到獲取第一幀的時間。
- 用戶停止 VLC 客戶端后，AmebaPro 進入系統睡眠 wlan 相關空閒。
- 用戶可以通過向 AmebaPro 發送 wlan unicast 來喚醒 Ameba。（例如 Ping 可以完成這個工作）

7.2.3.2 Setup example

需要搭佩使用 example_sources 裡面的 power_save 範例

修改文件 “platform_opts.h”（放在

“project\realtek_amebapro_v0_example\inc\platform_opts.h” 中）

並啟用這些選項：

可以參考範例 “media_framework”（在

“component\common\example\media_framework” 中）

```
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK 1
#if CONFIG_EXAMPLE_MEDIA_FRAMEWORK
#define FATFS_DISK_SD 1
#define ISP_BOOT_MODE_ENABLE 1
#if ISP_BOOT_MODE_ENABLE
#define ISP_BOOT_MP4 0
#define ISP_BOOT_RTSP 1
#if (ISP_BOOT_MP4 == 1 && ISP_BOOT_RTSP == 1) || (ISP_BOOT_MP4 == 0 && ISP_BOOT_RTSP == 0)
#error "It only can select the mp4 or rtsp"
#endif
#endif
#endif
#endif
```

這將啟用 media framework 範例。您可以引用 “example_media_framework.c”
（放在 “\ component \ common \ example \ media_framework \
example_media_framework.c” 中）。

在編譯和載入映像檔後，啟動 AmebaPro 後可以看到如下訊息：

```
WIFI initialized

init_thread(53), Available heap 0x9b0f60
use ATW0, ATW1, ATWC to make wifi connection
wait for wifi connection...
```

然後使用 AT 命令連接到 AP。

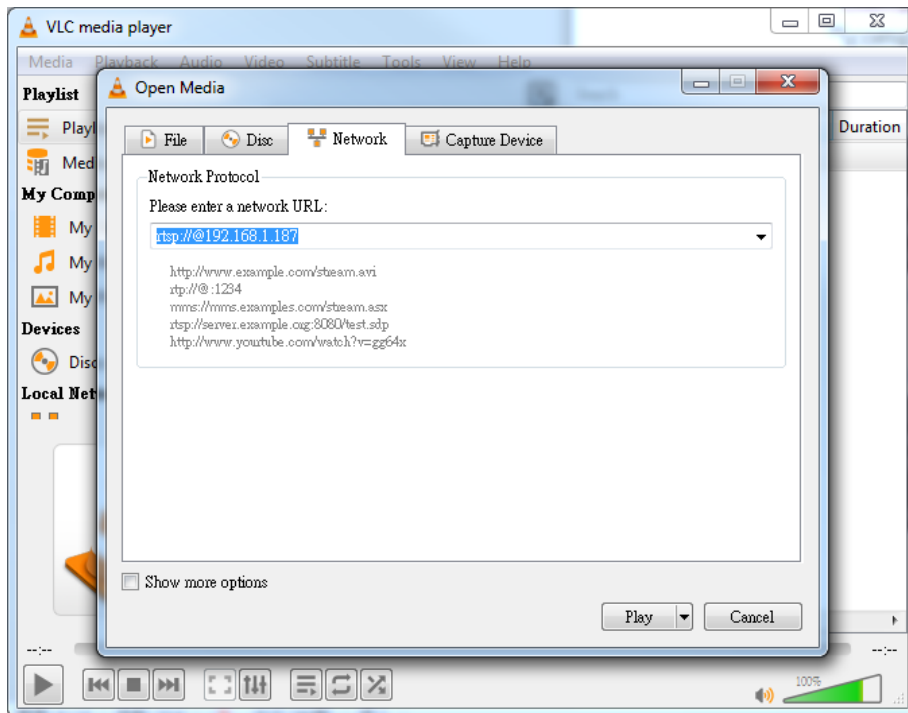
連接到 AP 後 AmebaPro 將啟動 RTSP Server。

```
RTSP[0] port: 554
time_sync_enable
connect successful sta mode

rtsp stream enabled
```

您可以使用 ping 命令 (1) 測試 AmebaPro 的連接，(2) 更新 PC 或智能手機的 ARP 表

然後打開 VLC 作為 RTSP 客戶端並連接到 AmebaPro：



在連接 RTSP 客戶端后，應該看到串流傳輸，當串流停止傳輸後系統將進入省電狀態。預設情況下 AmebaPro 是由 wlan unicast 喚醒的，所以可以透過再次 ping AmebaPro 來恢復它。恢復後，AmebaPro 就像開機啟動一樣，所以你可以再次檢查這個範例。

7.3 編程建議

7.3.1 Standby 若不需要用到 wowlan, 則需要關掉 wlan

當我們使用 standby 卻又不需要 wake on wlan mode, 那麼我們必須在進 standby 前將 wifi 關掉。這個情況通常發生在無法連線上 AP, 卻又希望可以省電。如果我們在進 standby 未將 wifi 關掉，那麼進 standby 將會使系統當機。

7.3.2 當 HS 睡眠時不可以使用 ICC

當 HS 睡眠時，無法使用 ICC。所以在 LS 的編程裡應該分成兩種情況: HS 醒著、以及 HS 睡眠。如果 HS 睡眠時並且嘗試使用 ICC，將會導致無法預期的結果。

7.3.3 進 Standby 前關閉 Video, ISP, SD, USB

如果編程裡使用了 Video, ISP, SD, USB, 那麼在進 Standby 前要將他們關閉。若未將他們關閉，將會導致醒來時系統無法正常開機。

7.3.4 使用 Standby 裡可保留的 LS 記憶體

進 Standby 時, LS 的 SRAM 的資料可以保留, 然而當 LS 醒來時, BSS 的資料都會被清空。如果我們想放一些資料讓 LS 醒來時仍可以使用, 那我們需要將這些資料放在 DATA section 裡。

7.3.5 連上 AP 與進 wake on wlan mode 中間需要一些延遲

如果我們希望連上 AP 之後馬上進 wake on wlan mode, 中間需要加一些延遲。當 STA 連上 AP 時, 通常 AP 端會有一些封包會送至 STA, 如果此時我們進 wake on wlan mode, 會使得 STA 脫離 LPS, 使得進 wake on wlan mode 花費比平常更多的時間。所以建議加一些延遲, 讓 STA 先收完這些封包再進 wake on wlan mode 會比較省電。

7.3.6 進 Standby 前對 LS GPIO 狀態進行 Pull Control 設定

先確認 LS GPIO 外部電路狀態, 並設定 GPIO pull control 和外部匹配, 以防止漏電。

- 外部 pull high→內部 pull high
- 外部 pull low or floating→內部 pull low

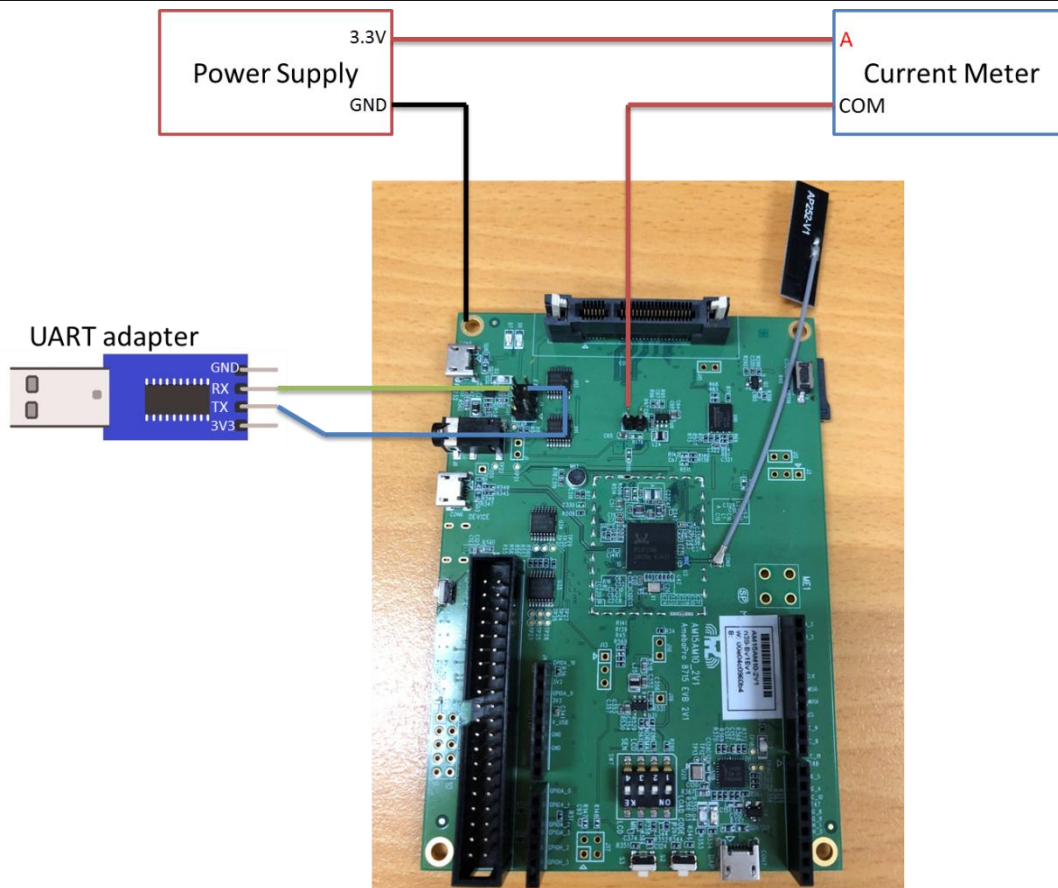
```
union {
    __IOM uint32_t gpioa_pull_ctrl; /*!< (@ 0x00000144) GPIOA Pull Control Register*/

    struct {
        __IOM uint32_t gpioa0_pull_ctrl : 2; /*!< [1..0] 2b'00: high impedance; 2b'01:
pull low; 2b'10: pull high; 2b'11: reserved */
        __IOM uint32_t gpioa1_pull_ctrl : 2; /*!< [3..2] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa2_pull_ctrl : 2; /*!< [5..4] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa3_pull_ctrl : 2; /*!< [7..6] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa4_pull_ctrl : 2; /*!< [9..8] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa5_pull_ctrl : 2; /*!< [11..10] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa6_pull_ctrl : 2; /*!< [13..12] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa7_pull_ctrl : 2; /*!< [15..14] 2b'00: high impedance;
```

```
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    __IOM uint32_t gpioa8_pull_ctrl : 2;    /*!< [17..16] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    __IOM uint32_t gpioa9_pull_ctrl : 2;    /*!< [19..18] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    __IOM uint32_t gpioa10_pull_ctrl : 2;   /*!< [21..20] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    __IOM uint32_t gpioa11_pull_ctrl : 2;   /*!< [23..22] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    __IOM uint32_t gpioa12_pull_ctrl : 2;   /*!< [25..24] 2b'00: high impedance;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    } gpioa_pull_ctrl_b;
};
```

7.4 量測耗電

我們可以在 EVB 2V1 的板上量測耗電, 在下載 image 之後, 移除 J9 的兩個 jumper, 以及 J21 的一個 jumper, 並按照下圖接線



在量測時有些需要注意的地方：

- 電源供應器
我們用 3.3V 量測耗電。在量測耗電時，有時電壓瞬間掉的太低，這是因為電源供應器無法及時反應電流的變化，這種情況下你可以加一些電容避免這種狀況。
- 電流計
當量測 wake on wlan 的耗電時，你需要將電流計的 range 設定至 1A。這是因為 wake on wlan 的電流變化會在幾百微安培至幾百毫安培，若電流計的 range 調成 auto，此時會因為電流計反應不及造成電壓下降太多並且裝置當機。另外也需要將電流計的 sample rate 設至 1K 或更高。當裝置接收 wifi AP 的 beacon 封包時，wifi 會打開 RF 幾個微秒，如果電流計的 sample rate 不夠高，就無法捕捉到此時的耗電。
- UART adapter
UART adapter 可以让你輸入命令，然而它的 TX/RX 線路可能會提供額外的電源至 AmebaPro 上，當你輸入完命令並且要量測耗電時，一定要將 TX/RX 線路移除。

8 系統

8.1 **ICC (Inter CPUs Communication)**

RTL8195B 平台上有 2 個 CPU，HP（高功率）CPU 和 LP（低功耗）CPU。ICC (Inter CPUs Communication) 是爲了使 2 個 CPU 可以相互溝通而設計。

8.1.1 **Example: ICC**

8.1.1.1 *Abstract*

此範例顯示：

1. 如何在 HS 和 LS 兩平台間相互收送 ICC cmd。
2. ICC message 循環回來。ICCs message 數據從 HS 發送到 LS，然後 LS 再將 ICC message 循環發送到 HS。

8.1.1.2 *Setup example*

範例位於：

“\project\realtek_amebapro_v0_example\example_sources\icc\”

將 HS 和 LS 文件夾複製到項目文件夾，編譯 LS 和 HS 代碼並下載程序。

8.2 **System reset**

8.2.1 **ls_sys_reset**

```
/* Reset LS platform via ICC command*/  
void ls_sys_reset( void )
```

8.2.2 **sys_reset**

```
/* Reset some register status and do HS watchdog reset */  
void sys_reset( void )
```

9 硬體周邊

9.1 GTimer

- HS 支持 16 個 Gtimer，LP 支持 8 個 Gtimer
- Source clock 20MHz, moderate accuracy 32KHz

9.1.1 gtimer_rtc

在此示範例中述如何使用 gtimer API 來實現軟體 RTC

gtimer_rtc 範例位於

project\realtek_amebapro_v0_example\example_sources\gtimer_rtc 中

9.1.2 SNTP

在此示範例中述如何從 NTP 伺服器抓取系統時間

sntp_showtime 範例位於 component\common\example\sntp_showtime 中

9.1.2.1 Software Setup

在(project)\inc\platform_opts.h 中配 SNTP 示例設置：

```
/* For sntp show time example */  
#define CONFIG_EXAMPLE_SNTP_SHOWTIME 1
```

9.2 RTC

在此範例中 LS 註冊一個 RTC，設定起始值，讀取時間。

RTC 範例位於 project\realtek_amebapro_v0_example\example_sources\rtc 中

9.3 PWM

- HS 和 LP 分別支持最多 8 個 PWM
- Duty 可配置 0~100%
- 使用選定的 Gtimer 中斷作為 counter source

9.4 Audio

- Default DAC value 等於 0xAF: 0dB

- Default ADC value 等於 0x2F: 0dB

9.4.1 DAC Volume

```
/**
 * @brief Control the digital gain of DAC.
 * @param obj: Audio object define in application software.
 * @param step: The digital volume. Every Step is 0.375dB.
 *             @arg 0xAF: 0dB.
 *             @arg 0xAE: -0.375dB.
 *             @arg ...
 *             @arg 0x00: -65.625dB.
 * @retval none
 */
void audio_dac_digital_vol (audio_t *obj, u8 step);

/**
 * @brief Control the digital mute of DAC.
 * @param obj: Audio object define in application software.
 * @param mute_en: To enable or disable mute.
 * @retval none
 */
void audio_dac_digital_mute (audio_t *obj, BOOL mute_en);
```

9.4.2 ADC Volume

```
/**
 * @brief Control the digital gain of ADC.
 * @param obj: Audio object define in application software.
 * @param step: The digital volume. Every Step is 0.375dB.
 *             @arg 0x7F: 30dB.
 *             @arg ...
 *             @arg 0x2F: 0dB.
 *             @arg ...
 *             @arg 0x00: -17.625dB.
 * @retval none
 */
void audio_adc_digital_vol (audio_t *obj, u8 step);

/**
 * @brief Control the digital mute of ADC.
 * @param obj: Audio object define in application software.
 * @param mute_en: To enable or disable mute.
 * @retval none
 */
void audio_adc_digital_mute (audio_t *obj, BOOL mute_en);
```

9.5 Efuse

- LS efuse
 - Logical area : 0x0 ~ 0x1CF , size: 464 bytes
 - Reserved area : 0x1D0 ~ 0x1FF , size: 48 bytesTotal size : 512 bytes

- HS efuse
 - Logical area : 0x0 ~ 0x10F , size: 272 bytes
 - OTP : 0x110~0x12F = 32 bytes
 - security area 0x130~0x1AF = 128 bytes
 - ◆ security key: 0x130~0x14F
 - ◆ writable security zone : 0x150~0x16F
 - ◆ super security key: 0x170~0x18F
 - ◆ reserved: 0x190~0x1AF
 - wlan hidden area : 0x1B0 ~ 0x1FF = 80 bytesTotal size : 512 bytes

9.5.1 User OTP

User OTP : 0x110~0x12F = 32 bytes

efuse_api

```
/**
 * @brief Read efuse OTP content
 * @param address: Specifies the offset of the OTP.
 * @param len: Specifies the length of readback data.
 * @param buf: Specified the address to save the readback data.
 * @retval 0: Success
 * @retval -1: Failure
 */
int efuse_otp_read(u8 address, u8 len, u8 *buf);

/**
 * @brief Write user's content to OTP efuse
 * @param address: Specifies the offset of the programmed OTP.
 * @param len: Specifies the data length of programmed data.
 * @param buf: Specified the data to be programmed.
 * @retval 0: Success
 * @retval -1: Failure
 */
```

```
int efuse_otp_write(u8 address, u8 len, u8 *buf);
```

9.6 Ethernet

在(project)\component\common\mbed\hal_ext\ethernet_ex_api.h 中配 Ethernet 介面設置：

```
/// the selection of Ethernet pinmux
#define ETH_PIN_SEL      (Eth_Pin_Sel1)//
/// the selection of the interface between MAC and PHY
#define ETH_IF_SEL      (Eth_If_Mii)
```

EVB 指撥開關 SW7 的 KEY3 要切到 MII

9.7 SD CARD

該章節主要介紹如何使用 SD CARD API

- sdio_driver_init() :指定 function pointer 給底層 API 使用
- SD_Init() 初始化 SD CARD
- SD_DeInit() 去始化 SD CARD
- SD_Wait_Ready() 目前只有針對 SPI 有等待,SDCARD 沒有此功能
- SD_SetCLK(enum sdioh_access_mode_e clk) 設定 sd card clock speed,使用者無需設定,內部驅動會根據卡訊息註冊
- SD_Status() 取得目前 SD CARD 狀態,回傳 0 為成功
- SD_GetCID() 目前無功能
- SD_GetCSD() 目前無功能
- SD_GetCapacity() 取得目前 Sector 數目
- SD_ReadBlock(u32 sector,u8 *data,u32 count) SD CARD 讀功能,參數 SECTOR 位置,data 處理的數據緩存,count 多少個 sector count
- SD_WriteBlock(u32 sector,u8 *data,u32 count) SD CARD 寫功能,參數 SECTOR 位置,data 處理的數據緩存,count 多少個 sector count

目前必須先呼叫 sdio_driver_init,之後再執行 SD_Init().做完此初始化之後,就可以開始做讀寫作功能.假如已經不需要使用 SD CARD,則需要呼叫 SD_Deinit 去關閉 SD card 功能

以下是卡的速度設定

SdiohSpeedDS	= 0, // 3.3V Function 0 //Default Speed
SdiohSpeedHS	= 1, // 3.3V Function 1 //High Speed
SdiohSpeedSDR12	= 2, // 1.8V Function 0
SdiohSpeedSDR25	= 3, // 1.8V Function 1
SdiohSpeedSDR50	= 4, // 1.8V Function 2
SdiohSpeedSDR104	= 5, // 1.8V Function 3
SdiohSpeedDDR50	= 6, // 1.8V Function 4
SdiohKeepCurSpd	= 15

下面是必需的注意事項

- SD CARD 有 POWER PIN 開關.DEMO BOARD 和 QFN128 使用不同的 PIN 腳位.DEMO BOARD 使用 GPIOH_0 和 QFN128 使用 GPIOE_12.這隻 PIN 腳位字須拉低給 SD CARD 電源.假使我們需要對 SD CARD RESET,必須把電關閉.流程該 PIN 由低到高執行 RESET 動作.
- 如果是多個 Thread 執行,必須確保 SD CARD 已經初始化完畢,才能執行相關操作,否則操作做會是錯誤的結果
- 假使我們必須要對多個檔案同使存取,必須要打開 REENTRANT FLAG,才能確保檔案操作沒問題
- 請不要在寫卡過程中,進行拔卡動作,這會導致檔案系統毀損
- 爲了加速檔案讀寫的速度,需要建立一塊緩存記憶體,如果該記憶體填滿,才寫回 SD CARD,目前建議爲 32KB 或更高緩存空間,他必須爲 512 BYTE 的倍數
- 我們提供 FATFS 操作 SD CARD

10 檔案系統

AmebaPro 額外支援以 flash 和以 SD card 爲儲存媒介的檔案系統，且 AmebaPro 允許兩個同時存在。其中 SD 卡目前支援的是 FAT32，若使用 exFAT 或其他種檔案系統，則要先對 SD 卡進行格式化。使用者可以使用 FATFS 提供的 `f_mkfs()` 對 SD card 進行格式化。

10.1 FAT Filesystem on Flash

Flash 文件系統示例位於 `component \ common \ example \ fatfs` 中

10.1.1 Software Setup

首先在 `(project)\ inc \ platform_opts.h` 中配置 FATFS 示例設置：

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS          1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN              1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R10C
// fatfs disk interface
#define FATFS_DISK_USB              0 // Not supported on AmebaPro
#define FATFS_DISK_SD              0
#define FATFS_DISK_FLASH          1
#endif
#endif
```

注意 SDIO 在 C-cut QFN128 中可以使用 GPIOE_12 來控制電源，為 low active。預設 GPIO_12 為低電位，如果使用者沒有要控制電源，請注意不要控制 GPIOE_12。

再來修改 `component \ common \ file_system \ fatfs \ r0.10c \ include \ ffconf.h` 中的參數：

```
...
#define _USE_MKFS          1//0    /* 0:Disable or 1:Enable */
...
#define _MAX_SS            4096//512
...
```

請注意在 Flash FATFS 範例中使用的 flash filesystem 的 flash memory base 在文件 `component\common\file_system\fatfs\disk_if\src\flash_fatfs.c` 中定義：

```
...
#define FLASH_APP_BASE    0x180000
...
```

最後重新編譯專案並將應用程序載入到 DEV 板。

10.2 Dual FAT Filesystem - File system on both SD Card and Flash

Dual file system 範例位於 `component\common\example\fatfs`

10.2.1 Hardware Setup

請將 Ameba DEV 板連接上 SD/MMC 卡，如有需要請參考 UM0073

10.2.2 Software Setup

首先在(project)\inc\platform_opts.h 中配置 FATFS 範例的設置：

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS          1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN              1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
// fatfs disk interface
#define FATFS_DISK_USB              0 // Not supported on AmebaPro
#define FATFS_DISK_SD              1
#define FATFS_DISK_FLASH          1
#endif
#endif
```

接下來修改 ffconf.h 中的參數：

```
...
#define _USE_MKFS          1
...
#define _VOLUMES          2
...
#define _MAX_SS          4096
...
```

請注意在 Flash FATFS 範例中使用的 flash filesystem 的 flash memory base 在文件 component\common\file_system\fatfs\disk_if\src\flash_fatfs.c 中定義：

```
...
#define FLASH_APP_BASE 0x180000
...
```

最後重新編譯專案並將應用程序載入到 DEV 板。

10.3 產生 FAT File System Image

AmebaPro 可支援在外部產生的檔案系統映像。以下說明如何在 Ubuntu 使用工具產生 FAT File System Image。

- 使用 *dd* 生成 image file

```
root@ubuntu # dd if=/dev/zero of=test.bin count=64 bs=1KB
```

- 使用 *fdisk* 在 image 上建立分區

```
root@ubuntu # fdisk test.bin
```

- 使用 *mkfs* 在 image 上建立 FATFS

```
root@ubuntu # mkfs.msdos -S 4096 -F 12 test.bin
```

- 使用 *mcop*y 把檔案複製到 FATFS

```
root@ubuntu # mcopy -i test.bin ./fs/hello.txt ::hello.txt
```

- 使用 *mount* 將 image 掛載在 Ubuntu 系統，檢查產生的 FATFS 內檔案是否正確

```
root@ubuntu # sudo mount test.bin ./fs
```