



REALTEK

AN0301

Multimedia Framework v2

Abstract

這份文件介紹 AmebaPro SDK 的 Multimedia Framework ,
此架構整合了多個單元如 Video, Audio, Network, Storage,
媒體流在這些單元之間傳遞、或是儲存。

Table of Contents

1	Multimedia Framework Architecture.....	6
1.1	架構	6
1.1.1	Module	7
1.1.2	Context	9
1.1.3	Module Inter Connection	9
1.2	Module Type and Module Parameter	19
1.2.1	ISP	19
1.2.2	H264	23
1.2.3	RTSP	27
1.2.4	Video frame control	28
1.2.5	JPEG	30
1.2.6	AAC Encoder	30
1.2.7	AAC Decoder	31
1.2.8	Audio Codec	31
1.2.9	RTP Input	32
1.2.10	G711 Codec	32
1.2.11	MP4	33
1.2.10	I2S	34
1.2.11	https	34
1.3	使用 MMF v2 範例.....	36
1.3.1	範例程式.....	36
1.4	選擇和設定範例程式.....	36
1.4.1	選擇合適的範例程式.....	36
1.4.2	調整範例程式的 Video/Audio 參數	48
1.4.3	調整 LWIP 參數	49
1.4.4	Echo Cancellation	50
1.4.5	擷取第一張 frame, 相關注意事項	51
1.4.6	VLC media player 設定	51
2	Video API.....	54
2.1	H264 API.....	54
2.1.1	h264_open	54
2.1.2	h264_initial	54
2.1.3	h264_encode	55
2.1.4	h264_release.....	55

2.2	JPEG API	55
2.2.1	jpeg_open	55
2.2.2	jpeg_initial.....	56
2.2.3	jpeg_encode.....	56
2.2.4	jpeg_release.....	57
3	ISP API	57
3.1	video_subsys_init	57
3.2	isp_stream_create	57
3.3	isp_stream_destroy	58
3.4	isp_stream_set_complete_callback	58
3.5	isp_stream_apply	58
3.6	isp_stream_start.....	59
3.7	isp_stream_stop	59
3.8	isp_stream_poll	59
3.9	isp_handle_buffer.....	60
3.10	isp_set_flip	60
3.11	isp_get_flip.....	61
3.12	isp_set_brightness	61
3.13	isp_get_brightness.....	61
3.14	isp_set_contrast	62
3.15	isp_get_contrast	62
3.16	isp_set_saturation	62
3.17	isp_get_saturation	63
3.18	isp_set_sharpness.....	63
3.19	isp_get_sharpness	63
3.20	isp_set_gamma.....	64
3.21	isp_get_gamma	64
3.22	isp_set_gray_mode.....	64
3.23	isp_get_gray_mode	65
3.24	isp_set_exposure_mode	65
3.25	isp_get_exposure_mode	65

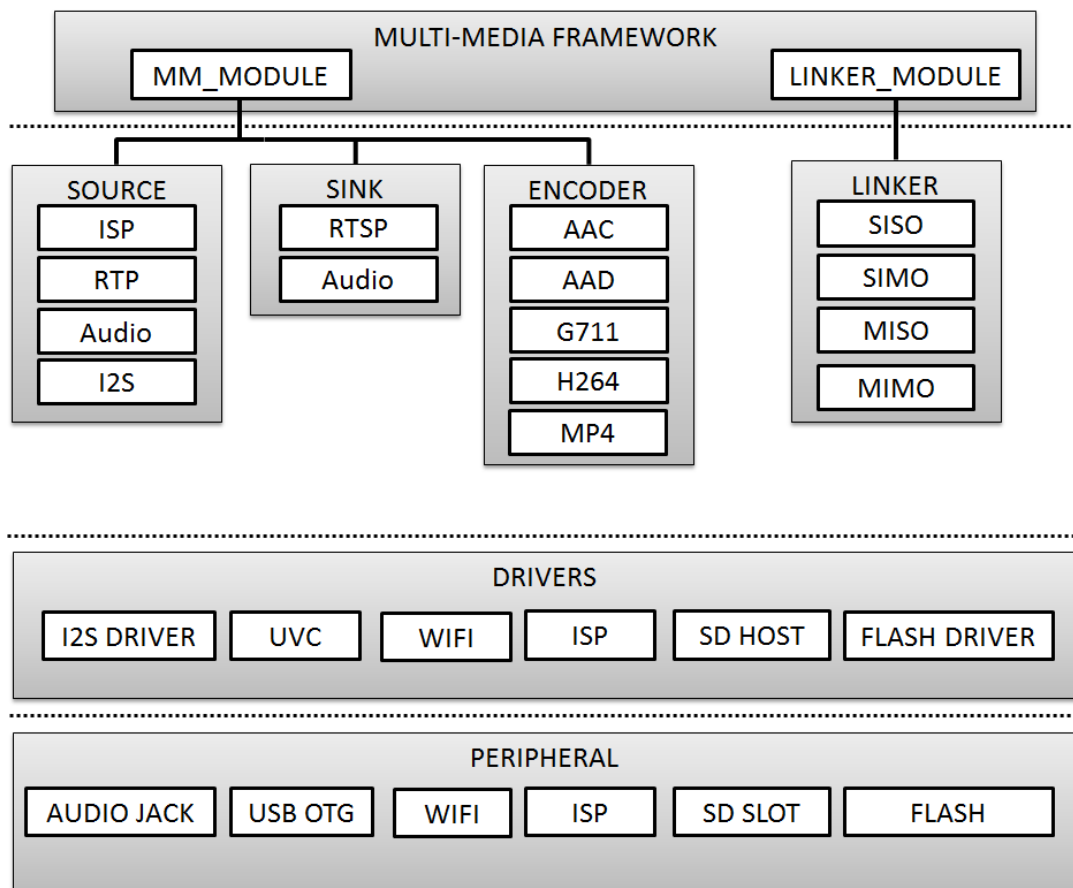
3.26	isp_set_exposure_time	66
3.27	isp_get_exposure_time	66
3.28	isp_set_zoom.....	66
3.29	isp_get_zoom	67
3.30	isp_set_pan_tilt	67
3.31	isp_get_pan_tilt.....	67
3.32	isp_set_AWB_ctrl.....	68
3.33	isp_get_AWB_ctrl	68
3.34	isp_set_power_line_freq.....	69
3.35	isp_get_power_line_freq	69
3.36	isp_set_AE_gain.....	69
3.37	isp_get_AE_gain	70
3.38	isp_set_WDR_mode.....	70
3.39	isp_get_WDR_mode	70
3.40	isp_set_WDR_level	71
3.41	isp_get_WDR_level	71
3.42	isp_stream_power_off_config.....	71
3.43	isp_check_boot_status.....	72
4	OSD	73
4.1	OSD introduction	73
4.2	OSD example	73
4.3	OSD Show Time information	74
4.4	OSD API	74
4.4.1	rts_video_query_osd_attr	74
4.4.2	rts_video_set_osd_attr.....	77
4.4.3	rts_video_release_osd_attr	78
4.4.4	rts_query_isp_osd_attr.....	78
4.4.5	rts_set_isp_osd_attr	79
4.4.6	rts_release_isp_osd_attr	80
5	Motion Detect.....	81
5.1	Motion Detect introduction.....	81
5.2	Motion Detect example	81

5.3	Motion Detect API	82
5.3.1	rts_video_query_md_attr.....	82
5.3.2	rts_video_set_md_attr	85
5.3.3	rts_video_release_md_attr	86
5.3.4	rts_video_check_md_status	86
5.3.5	rts_video_get_md_result.....	87
5.3.6	rts_query_isp_md_attr	87
5.3.7	rts_set_isp_md_attr.....	88
5.3.8	rts_check_isp_md_status	89
5.3.9	rts_get_isp_md_result	89
6	Mask	90
6.1	Mask introduction	90
6.2	Mask example	90
6.3	Mask API	91
6.3.1	rts_video_query_mask_attr.....	91
6.3.2	rts_video_set_mask_attr	93
6.3.3	rts_video_release_mask_attr	94
6.3.4	rts_query_isp_mask_attr	94
6.3.5	rts_set_isp_mask_attr	95
6.3.6	rts_release_isp_mask_attr.....	96

1 Multimedia Framework Architecture

Multimedia Framework Architecture (MMF)主要負責處理 AmebaPro 上連接與管理不同媒體資源。

1.1 架構



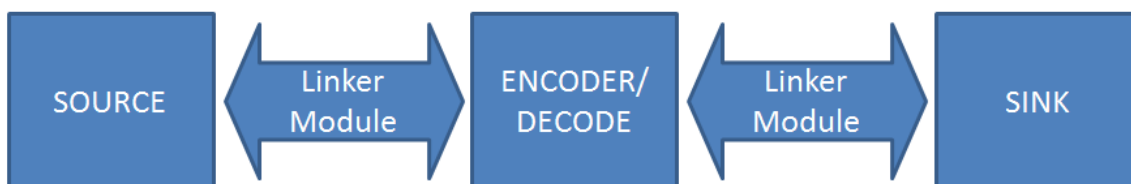
MMF 主要架構圖如下：

MMFv2 中有兩個重要的部分。一個是 **MM_MODULE** (**Source** , **Sink** 和 **Encode/Decode** 模塊是屬於 **MM_MODULE**) , **Source** 產生之資源交由 **Sink** 使用由接 **Source** 產生的資源。 **Source** 可以是檔案輸入、麥克風、攝影機或儲存在記憶體上的資料 , 而 **Sink** 可以是 **RTSP** 或其他串流。另一個是 **LINKER_MODULE** , 它連接不同類型的模組並處理模組間通信。

為順利使用 MMFv2，必須遵循以下方面。

- 定義有效的 Source。
- 定義有效的 Sink
- 如有需要，定義有效的 Encoder/Decoder。
- 定義有效的 LINKER_MODULE。

主要使用流程為初始化不同 MM_MODULE，將不同的 MM_MODULE 透過 LINKER_MODULE 串接起來。



1.1.1 Module

MMFv2 允許用戶根據需求定義客製的 Source、Sink 和 Encode/Decoder 模組。雖然實現細節可能不同，但建構 MMF 模組的基本規則類似。

MMF module

MMFv2 要求使用者根據其需求預先定義好 create、destroy、control、handle、new_item、del_item 和 rsz_item 函數。MMFv2 提供 mmf_module_t 來成為使用模組的介面。為保持模組之間的靈活性和便利性，mmf_module_t 只保留每種類型的介面以提供使用者訪問，每個模組的函數實體由模組本身定義。

```

typedef struct mm_module_s{
    void*      (*create)(void*);
    void*      (*destroy)(void*);
    int        (*control)(void*, int, int);
    int        (*handle)(void*, void*, void*);
    void*      (*new_item)(void*);
    void*      (*del_item)(void*, void*);
    void*      (*rsz_item)(void*, void*, int);
    uint32_t   output_type;
}
  
```

函數說明

create

負責 module_ctx 之記憶體空間配置並初始化與底層硬體之初始化。以 ISP 模組舉例此函數負責分配空間與初始化 ISP 相關參數。

destroy

module_ctx、hw slot 與 module 相關空間在此釋放。以 ISP 模組舉例此函數負責釋放 isp 使模組所使用到之空間。

control

將各 module 之參數在此指派給各個 module 的 context，從 module 中讀取參數也在此實做。其中 case CMD_AUDIO_APPLY 裡實做套用各個參數與 module 之開始運作。以 ISP 舉例此函數控制 ISP 參數（"frame height"，"frame width"，"framerate"等等）和 MMFv2 任務開啟或關閉。

handle

處理數據函數（如何在 Source 中產生數據或如何在 Sink 中使用數據）。通過 OS 訊息佇列將數據從 Source 端傳輸到 Sink 端反之亦然。請注意 MMFv2 handler 會根據訊息交換緩衝區狀態做出不同的反應。

new_item

將 Queue 所需要的資料空間分配出來並且將位置傳到 Queue 中，只有在設定 MM_CMD_INIT_QUEUE_ITEMS 為 MMQI_FLAG_STATIC 時會使用到。

del_item

當 module 使用完畢，將透過 del_item 來實做 queue 各元素之釋放，只有在設定 MM_CMD_INIT_QUEUE_ITEMS 為 MMQI_FLAG_STATIC 時會使用到。

rsz_item

rsz_item 裡實做將 memory pool 縮小，在目前的範例中只有 h264 與 aac 兩支 module 會使用到。

output_type and module_type

定義在 mmf2_module.h 中，output_type 表示輸出的模式，共有 (MM_TYPE_NONE, MM_TYPE_VSRC, MM_TYPE_ASRC, MM_TYPE_VDSP,

MM_TYPE_ADSP, MM_TYPE_VSINK, MM_TYPE_ASINK, MM_TYPE_AVSINK) 可以使用，分別對應不同的 module 使用情境，讓 application 知道輸出為何種模式，module_type 表示 module 的身分，共有 (MM_MASK_SRC, MM_MASK_DSP, MM_MASK_SINK) 三種可供選擇。

name

保存 module 的名字。

1.1.2 Context

MMFv2 context 負責提供不同模塊之間的訊息傳輸介面，包含 mm_module_t 和用於傳遞數據的隊列。mm_context 支援 6 種狀態 (MM_STAT_INIT, MM_STAT_READY, MM_STAT_ERROR, MM_STAT_ERR_MALLOC, MM_STAT_ERR_QUEUE, MM_STAT_ERR_NEWITEM)，負責用來維護 module 狀態以確保程式順利執行。

```
typedef struct mm_context_s{
    xQueueHandle  output_ready;
    xQueueHandle  output_recycle;
    mm_module_t*  module;
    void*         priv;    // private data structure for created instance
```

mm_context 主要負責維護各 module 實體，module 種類有 isp、h264、jpeg、aac_encoder、aac_decoder、amebapro_audio、g711、mp4、rtp 和 rtsp，每個 module 皆各自獨立並且在使用的同時對應了個別的 input、output queue 與狀態，並在該 module 的 mm_context 中更新參數與傳遞實體。

1.1.3 Module Inter Connection

此章節介紹 mm_siso_t、mm_simo_t、mm_miso_t、mm_mimo_t 與其相對應的 create、delete、ctrl、start、stop、pause、resume function，在 mmfv2 中負責模組間連接與控制。

SISO module(Single Input Single Output)

```
typedef struct mm_siso_s{  
    mm_context_t      *input;  
    mm_context_t      *output;  
    uint32_t          status;  
    uint32_t          stack_size;  
}
```

模組間單向串接介面，input 與 output 分別為輸入與輸出 module，status 維持 SISO module 的狀態以確定傳輸中間的過程正確，stack_size 用來決定中間傳輸的 handler task 大小，並保留 xTaskCreate 的 xTaskHandle task 用來控制 task 的使用。

在 SISO module 中有幾個 function 負責 module inter-connection，這些 function 大多用來更新 task 的狀態並且交由 task handler 來進行主要處理：

siso_create

siso_create 宣告 mm_siso_t 的空間並初始化後回傳 mm_siso_t。

siso_delete

停止 SISO 執行並釋放 mm_siso_t 的空間。

siso_ctrl

有三個可使用之操作 MMIC_CMD_ADD_INPUT、MIC_CMD_ADD_OUTPUT、MMIC_CMD_SET_STACKSIZE，MMIC_CMD_ADD_INPUT 將 input module 連結至 siso module 的 input 中，MMIC_CMD_ADD_OUTPUT 將 output module 連結至 siso module 的 output 中，MMIC_CMD_SET_STACKSIZE 將 size 加到 siso 的 stack_size 中。

siso_start

在開始前會檢查 input 與 output module 有沒有東西，如果都有會建立一個 task handler 將 input module 的資料送到 output module 中。

siso_stop

更新狀態為 MMIC_STAT_SET_EXIT 並等待 task handler 將 status 切換到 MMIC_STAT_EXIT。

siso_pause

更新狀態為 MMIC_STAT_SET_PAUSE 並等待 task handler 將 status 切換到 MMIC_STAT_PAUSE。

siso_resume

更新狀態為 MMIC_STAT_SET_RUN 並等待 task handler 將 status 切換到 MMIC_STAT_RUN。

SIMO module(Single Input Multiple Output)

```
typedef struct mm_simo_s{
    mm_context_t      *input;
    int               output_cnt;
    mm_context_t      *output[4];

    // internal queue to handle reference count and usage log
    mm_simo_queue_t    queue;
    uint32_t           status[4];
};
```

模組間單向串接介面，input 與 output 分別為輸入與輸出 module，其中 output_cnt 表示同時 output 的 module 數量，output 最高同時支援 4 種 output，status 維持 SIMO module 的狀態以確定傳輸中間的過程正確，stack_size 用來決定中間傳輸的 handler task 大小，並保留 xTaskCreate 的 xTaskHandle task 用來控制 task 的使用。

在 SIMO module 中有幾個 function 負責 module inter-connection，這些 function 大多用來更新 task 的狀態並且交由 task handler 來進行主要處理：

simo_create

simo_create 宣告 mm_simo_t 的空間並初始化，另外維護一個 queue head 與 queue.lock 用來維護多個 output 的結果，最後回傳 mm_simo_t。

simo_delete

呼叫 simo_stop()，停止 SIMO 執行並釋放空間。

simo_ctrl

有六個可使用之操作 MMIC_CMD_ADD_INPUT 將 input module 連結至 simo module 的 input 中，MMIC_CMD_ADD_OUTPUT0、MMIC_CMD_ADD_OUTPUT1、MMIC_CMD_ADD_OUTPUT2、MMIC_CMD_ADD_OUTPUT3 將 output module 連結至 simo module 對應的 output 中並增加 output_cnt 記錄 output module 數量，MMIC_CMD_SET_STACKSIZE 將 size 加到 simo 的 stack_size 中。

simo_start

simo_start 會根據 simo->output_cnt 來建立對應數量之 task handler，每個 task handler 都用來傳出接收到的資料。

simo_stop

simo_stop 將每個 simo status 設定為 MMIC_STAT_SET_EXIT，等待 task handler 將每個 status 切換到 MMIC_STAT_EXIT。

simo_pause

simo_pause 會根據 pause_mask 將每個 simo->status 設定為 MMIC_STAT_SET_PAUSE，等待 task handler 將每個 status 切換到 MMIC_STAT_PAUSE。

simo_resume

simo_resume 會將每個 simo->status 設定為 MMIC_STAT_SET_RUN，等待 task handler 將每個 status 切換到 MMIC_STAT_RUN。

MISO module(Multiple Input Single Output)

```
typedef struct mm_miso_s{
    int          input_cnt;
    mm_context_t *input[4]; // max 4 input
    mm_context_t *output;
    uint32_t     status;
    uint32_t     stack_size;
```

模組間單向串接介面，input 與 output 分別為輸入與輸出 module，其中 input_cnt 表示同時 input 的 module 數量，input 最高同時支援 4 種 output，status 維持 MISO module 的狀態以確定傳輸中間的過程正確，stack_size 用來決定中間傳輸的 handler task 大小，並保留 xTaskCreate 的 xTaskHandle task 用來控制 task 的使用。

在 MISO module 中有幾個 function 負責 module inter-connection，這些 function 大多用來更新 task 的狀態並且交由 task handler 來進行主要處理：

miso_create

miso_create 裡宣告 mm_miso_t 的空間並初始化後回傳 mm_miso_t。

miso_delete

呼叫 miso_stop() 將 MISO 停止並釋放空間。

miso_ctrl

有六個可使用操作 MMIC_CMD_ADD_INPUT0、MMIC_CMD_ADD_INPUT1、MMIC_CMD_ADD_INPUT2、MMIC_CMD_ADD_INPUT3、MMIC_CMD_ADD_OUTPUT、MMIC_CMD_SET_STACKSIZE，MMIC_CMD_ADD_INPUT0、MMIC_CMD_ADD_INPUT1、MMIC_CMD_ADD_INPUT2、MMIC_CMD_ADD_INPUT3 將 input module 連結至 miso module 的 input 中並增加 input_cnt 的值以記錄 input module 個數，MMIC_CMD_ADD_OUTPUT 將 output module 連結至 miso module 的 output 中，MMIC_CMD_SET_STACKSIZE 將 size 加到 miso 的 stack_size 中。

miso_start

在開始前會檢查 input 與 output module 有沒有東西，如果都有會建立一個 task handler 將 input module 的資料送到 output module 中。

miso_stop

將 miso status 設定為 MMIC_STAT_SET_EXIT，等待 task handler 將 status 切換到 MMIC_STAT_EXIT。

miso_pause

miso_pause 會依據 pause_mask 將 miso->status 設定為 MMIC_STAT_SET_PAUSE，等待 task handler 將 status 切換到 MMIC_STAT_PAUSE。

miso_resume

miso_resume 會將 miso->status 設定為 MMIC_STAT_SET_RUN，等待 task handler 將每個 status 切換到 MMIC_STAT_RUN。

MIMO module (Multiple Input Multiple Output)

```
typedef struct mm_mimo_s{
    int          input_cnt;           // depend on input count
    mm_context_t *input[4];
    mm_mimo_queue_t queue[4];
    int          output_cnt;         // depend on output count
    mm_context_t *output[4];         // output module context
    uint32_t     output_dep[4];      // output depend on which input, bit mask
    uint32_t     input_mask[4];      // convert from output_dep, input
                                         referenced by which output, bit mask
    uint32_t     status[4];
}
```

模組間單向串接介面，input 與 output 分別為輸入與輸出 module，其中 input_cnt 與 output_cnt 表示同時間 input 與 output 的 module 數量，input 與 output 最高同時支援 4 種 output，MIMO module 另需要 mm_mimo_queue_t queue[4]來維持各 input queue 的同步問題，mm_mimo_queue_t 都有一個 lock 與 head 分別記錄各個 queue 的開頭與是否有程式正在使用當中，status 維持 MIMO module 的狀態以確定傳輸中間的過程正確，stack_size 用來決定中間傳輸的 handler task 大小，並保留 xTaskCreate 的 xTaskHandle task 用來控制 task 的使用。

mimo_create

mimo_create 裡宣告 mm_mimo_t 的空間並初始化後回傳 mm_mimo_t。

`miso_delete`

呼叫 `mimo_stop()` 將 `mimo module` 停止並釋放空間。

`mimo_ctrl`

有九個可使用操作 `MMIC_CMD_ADD_INPUT0`、`MMIC_CMD_ADD_INPUT1`、`MMIC_CMD_ADD_INPUT2`、`MMIC_CMD_ADD_INPUT3`、`MMIC_CMD_ADD_OUTPUT0`、`MMIC_CMD_ADD_OUTPUT1`、`MMIC_CMD_ADD_OUTPUT2`、`MMIC_CMD_ADD_OUTPUT3`、`MMIC_CMD_SET_STACKSIZE`，`MMIC_CMD_ADD_INPUT0`、`MMIC_CMD_ADD_INPUT1`、`MMIC_CMD_ADD_INPUT2`、`MMIC_CMD_ADD_INPUT3` 將 `input module` 連結至 `mimo module` 對應的 `input` 中並增加 `input_cnt` 的值以記錄 `input module` 的個數，`MMIC_CMD_ADD_OUTPUT0`、`MMIC_CMD_ADD_OUTPUT1`、`MMIC_CMD_ADD_OUTPUT2`、`MMIC_CMD_ADD_OUTPUT3` 將 `output module` 連結至 `mimo module` 的 `output` 中並增加 `output_cnt` 的值以記錄 `output module` 個數，`MMIC_CMD_SET_STACKSIZE` 將 `size` 加到 `mimo` 的 `stack_size` 中。

`mimo_start`

`mimo_start` 根據 `output_cnt` 建立相對應的 `task handler` 來傳輸接收到的數據。

`mimo_stop`

`mimo_stop` 根據 `output_cnt` 將 `mimo status` 設定為 `MMIC_STAT_SET_EXIT`，等待 `task handler` 將 `status` 切換到 `MMIC_STAT_EXIT`。

`mimo_pause`

`miso_pause` 會根據 `pause_mask` 將每個 `mimo->status` 設定為 `MMIC_STAT_SET_PAUSE`，等待 `task handler` 將 `status` 切換到 `MMIC_STAT_PAUSE`。

`mimo_resume`

mimo_resume 會將每個 status 為 MMIC_STAT_PAUSE 的 task 中的 mimo
->status 設定為 MMIC_STAT_SET_RUN，等待 task handler 將每個 status 切
換到 MMIC_STAT_RUN。

1.2 Module Type and Module Parameter

MMFv2 Example 支援許多不同的應用情境，module 參數也支援手動調整，讀者
將可透過本章節了解不同 module 參數的意義與如何設定。


1.2.1 ISP

```
isp_params_t isp_v1_params = {  
    .width   = V1_WIDTH,  
    .height  = V1_HEIGHT,  
    .fps     = V1_FPS,  
    .slot_num = V1_HW_SLOT,  
    .buff_num = V1_SW_SLOT,  
    format   = ISP_FORMAT_YUV420_SFMIPI_ANAR.
```

- 解析度:支援 1080P 含以下的設定。
- FPS: DROP FRAME 機制設定,詳細說明在底下
- HW_SLOT:目前最大支援 4 個
- SW_SLOT:包含 HW_SLOT 數目,剩餘的會作為緩存
- Format: 目前僅支援 YUV420
- Boot mode :分為 ISP_FAST_BOOT 以及 ISP_NORMAL_BOOT,前者 可以支援 ISP
在 BOOT TIME 時間初始化已加快速度,後者在 BOOT CODE 完成之後再由使用
者初始化 ISP 設定,若要 FAST BOOT 需要先設定好 ISP 參數,僅支援一路設定,
- Wake mode:可以選擇從 BOOT,WLAN 或者 GPIO 情境喚醒,若非選擇模式,就不
會進行 ISP FAST BOOT,預設為 WAKE_FROM_BOOT,也就是會進入 FAST BOOT.
- PIN_IDX : 預設選擇 ISP_PIN_SEL_S0
- MODE : 須設定為 ISP_FAST_BOOT,兩個參數必須設定一樣
- INTERFACE : 目前預設 MIPI,另外也可支援 DVP 介面

- CLK : 根據 eva_board.h 選擇不同 sensor
- SENSOR_FPS : 根據 SENSOR.H 選擇預設的 SENSOR FRAM
- ISP_FW_LOCATION : 預設為 XIP,在特殊情況可以選擇 DRAM

```
CINIT_DATA_SECTION isp_boot_stream_t isp_boot_stream = {  
    .width = V1_WIDTH,  
    .height = V1_HEIGHT,  
    .isp_id = 0,  
    .hw_slot_num = V1_HW_SLOT,  
    .fps = V1_FPS,  
    .format = ISP_FORMAT_YUV420_SEMIPLANAR,  
    .pin_idx = ISP_PIN_SEL_S0,  
    .mode = ISP_FAST_BOOT,  
    .interface = ISP_INTERFACE_MIPI  
    .clk = SENSOR_CLK_USE
```

 *Video/Audio/ISP 參數並非任意可調整，在 RTOS 系統上，資源配置需要精算，請按照建議的範圍內調整*

關於檔案大小計算，目前預設 ISP 輸出格式為 YUV420 Format，假設目前設定解析度為 1080P, 那麼一個 FRAME 大小為 $1920 \times 1080 \times 1.5 = 3110400$ bytes, 若 HW_SLOT = 1, SW_SLOT=3, 那麼消耗的記憶體大小為 $3110400 \times 3 = 9331200$ bytes。關於 SLOT 數目設定，目前建議 HW_SLOT 設定為 1，SW_SLOT 會根據 FPS 大小決定,如果為 30FPS 建議設為 3,如果為 15FPS 建議設為 2。

最大支援解析度組合為 1080P 15 FPS 搭配 720P 30 FPS,如果超過此限制有機會掉 FRAME 問題。主要限制在於 H264 壓縮速度限制

關於 FRAME RATE 設定分為兩種,第一個是設定 SENSOR 本身的 FRAME RATE,第二個是透過 DROP FRAME 方式,個別設定需要的 FRAME RATE,但是不能超過 SENSOR

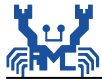
本身的 FRAME RATE.設定方式解說,如果 SENSOR FPS 為 30,可以每次都丟一張就是 30,可以兩次丟一張就是 15,可以三張丟一張就是 10,每次丟四張不行,因為無法整除,所以無法支援,依此類推可以得到支援 FRAME RATE 設定為 30,15,10,5,1

關於同時操作 ISP CHANNEL 注意事項:

- 1.不要同時初始化 ISP CHANNEL,需要等待其中一個 CHANNEL 設定完畢,才能繼續設定其他 ISP CHANNEL.
- 2.不要同時開關 ISP CHANNEL,需要等待其中一個 CHANNEL 設定完畢,才能繼續設定其他 ISP CHANNEL.

關於 SENSOR 開關問題,第一次開 ISP Stream 會對 SENSOR 執行初始化順序,但是第二個串流不會有這動作,節省開關 SENOSR 初始化時間,如果全部 STREAM 都關閉,在開啟 ISP 就會再重新進行 SENSOR 初始化過程,這邊也提供 API 設定是否要關閉 SENSOR.

關於 ISP FAST BOOT, 如果需要使用此功能,需要再初始化 ISP_BOOT_STREAM 參數,需要將變數 SECTION 設定為 CINIT,設定參數跟前一個一樣,需要將兩個參數的 mode 更改為 ISP_FAST_BOOT_MODE,此步驟會先將設定在 BOOT CODE 前設定好 ISP 參數,另外需要在進入程式 MAIN 先執行 ISP 初始化程式,主要目的是要掛載 ISP IRQ CALLBACK,可以在第一個 Frame 來之前可以接收,其他程式初始化步驟,需要擺在後面執行,相關設定請參考範例,開啟 ISP_BOOT_MODE_ENABLE Flag,另外可以選擇從 ISP_BOOT_MP4,ISP_BOOT_RTSP 或者 ISP_BOOT_MUX(錄影 First boot,串流 Normal).



```
#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 1
#define ISP_BOOT_RTSP 0
#define ISP_BOOT_MUX 0
#if (ISP_BOOT_MP4 == 1 && ISP_BOOT_RTSP == 1 && ISP_BOOT_MUX == 1) ||
```

1.2.2 H264

```
h264_params_t h264_v1_params = {  
    .width      = V1_WIDTH,  
    .height     = V1_HEIGHT,  
    .bps        = V1_BITRATE,  
    .fps        = V1_FPS,  
    .gop        = V1_FPS,  
    .rc_mode     = V1_H264_RCMODE,  
    .mem_total_size = V1_BUFFER_SIZE,  
};
```

- width : 影像寬度
- height : 影像長度
- bps : Bit per second 每秒傳輸的位元資料
- fps : Frame per second 每秒傳輸的 Frame 數目
- gop : Group of picture 多少個 Frame 更新一次 I Frame
- rc mode : Rate control mode, 目前有提供 CBR,VBR, ABR, FIXQP
- mem_total_size : H264 encoder memory size 容量
- mem_block_size : Memory pool 所使用的 block size
- mem_frame_size : 設定一個 FRAME SIZE 最大容量

CBR:



固定 bit rate , bit rate 由 V1_BITRATE 控制.

QP 範圍預設值 [10, 10, 51].

若有調整需求在 module_h264.h 裡面 h264_rc_parm_t 的[minQp, minIQp, maxQp]控制, 使用 API *h264_control*, 範例如下 (minIQp 為 I frame QP.)

VBR:

變動 bit rate, bit rate 由 V1_BITRATE 控制, 畫面靜止時會自動調整成 3/4 bit rate, 畫面變動時會超過 V1_BITRATE, 超出幅度由 maxQp 控制. maxQp 越大, bit rate 超過的幅度越小.

QP 範圍預設值[20, 20, 40]

ABR:

ABR 是長時間平均, 原理與 CBR 類似, 允許畫面變動時 bit rate 可以大幅超出預設 V1_BITRATE。超出幅度由 maxQp 控制。與 CBR 的差別是畫面變動時不會立即強制壓下 bit rate, 而是取長時間平均. 與 VBR 的差別是畫面靜止時不會自動調整成 3/4 bit rate.

QP 範圍預設值[20, 20, 40]

Advanced rate control parameters:


```
typedef struct h264_rc_parm_s
{
    unsigned int rcMode;
    unsigned int minQp;           // for CBR/VBR
    unsigned int minIQp;          // for CBR/VBR
    unsigned int maxQp;           // for CBR/VBR
}h264_rc_parm_t;

typedef struct h264_rc_adv_parm_s
{
    unsigned int rc_adv_enable;
    unsigned int maxBps;          // for VBR
    unsigned int minBps;          // for VBR
    int intraQpDelta;
    int mbQpAdjustment;
}h264_rc_adv_parm_t;

h264_ctx = mm_module_open(&h264_module);
mm_module_ctrl (h264_ctx, CMD_H264_SET_RCPARAM, &new_rc_param);
mm_module_ctrl (h264_ctx, CMD_H264_SET_RCADVPARAM, &new_adv_rc_param);
```

- **rcMode:** Rate control mode, 目前有提供 CBR,VBR, ABR, FIXQP
- **minQp:** 最小 QP 值 , 建議值: [10, 20]
- **minIQp:** I frame 最小 QP 值 , 正常場景建議等於 minQp
- **maxQp:** 最大 QP 值 , 建議值: [40, 51]
- **maxBps:** 每秒傳輸的最大位元資料 (for VBR)
- **minBps:** 每秒傳輸的最小位元資料 (for VBR)
- **intraQpDelta:** 調整 I Frame 的 QP 值 [-12..12]
- **mbQpAdjustment:** 減少 low-detail 區域的 artifacts , 負值為啟用 [-8..0]

Improving Image Quality:

- 調整 maxQp: 設置 maxQp 有助於有效保護圖像質量，但設置太小很容易發生 bit rate 過高。
- 調整 minQp: 此參數用於控制最高圖像質量。調整到 minQp 後，QP 不再降低，這可能導致 bit rate 不足。此參數旨在降低簡單靜態方案中的 bit rate。
- 調整 intraQpDelta: 由於預測方法不同，H.264 視頻中的 I Frame 有時會引入明顯的閃爍。通過調整 I Frame 的量化與周圍 P Frame 相比，可以克服該問題。
- 調整 mbQpAdjustment: 減少 low-detail 區域的 artifacts，建議值: [-1,-2].

```
/* Configure QP example */  
....  
h264_rc_parm_t new_param = {  
    .rcMode = RC_MODE_H264CBR,  
    .minQp = 10,  
    .minIQp = 51,  
    .maxQp = 51,  
};  
...
```

1.2.3 RTSP

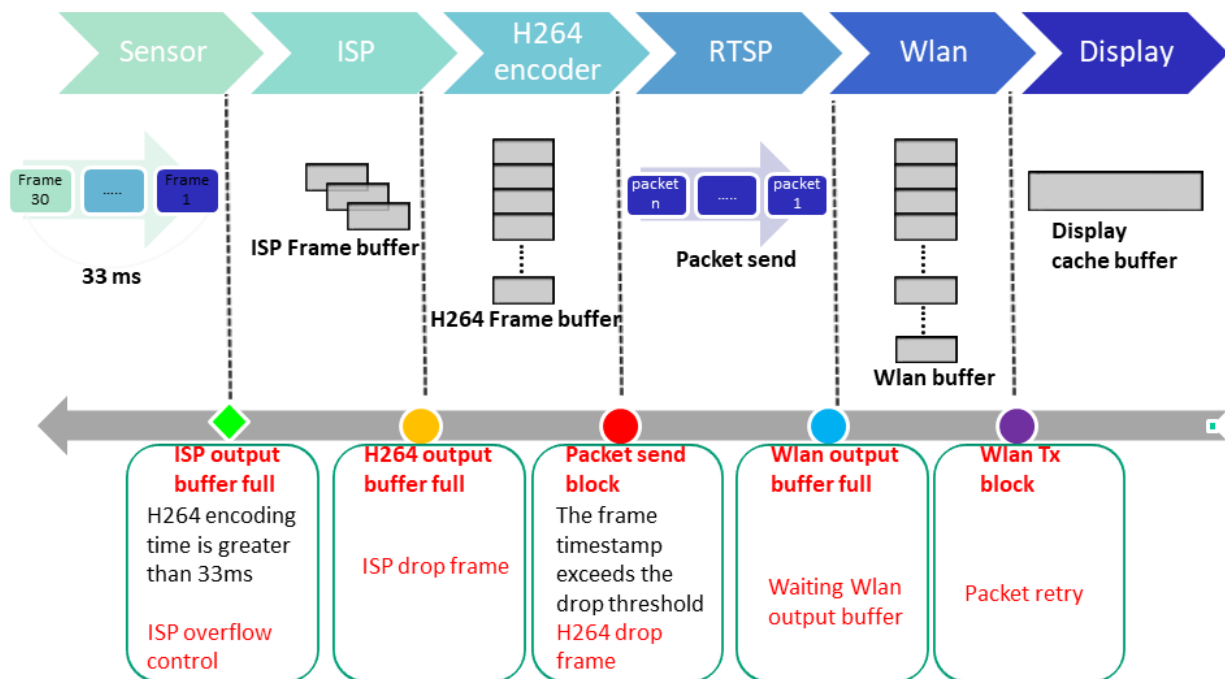
```
typedef struct rtsp2_params_s{
    uint32_t type;
    union{
        struct rtsp_video_param_s{
            uint32_t codec_id;
            uint32_t fps;
            uint32_t bps;
            uint32_t ts_flag;
            char* sps;
            char* pps;
            char* lv;
        }v;
        struct rtsp_audio_param_s{
            uint32_t codec_id;
            uint32_t channel;
            uint32_t samplerate;
        }a;
    }u;
}rtsp2_params_t;
```

- type: 媒體類型 , 目前提供 Video, Audio
- codec_id: Codec ID , 目前提供 AV_CODEC_ID_MJPEG, AV_CODEC_ID_H264, AV_CODEC_ID_PCMU, AV_CODEC_ID_PCMA, AV_CODEC_ID_MP4A_LATM, AV_CODEC_ID_MP4V_ES
- fps: Video frame rate
- bps: Bit per second 每秒傳輸的位元資料
- ts_flag: H264 及 AAC rtsp time sync 的開關
- sps,pps,lv: 設置 H264 的 sps, pps 及 profile level
- channel: audio channel
- samplerate: audio samplerate

Current codec table:

```
static const struct codec_info av_codec_tables[] = {
    {AV_CODEC_ID_MJPEG, "MJPEG", RTP_PT_JPEG, 90000, 0, 0},
    {AV_CODEC_ID_H264, "H264", RTP_PT_DYN_BASE, 90000, 0, 0},
    {AV_CODEC_ID_PCMU, "PCMU", RTP_PT_PCMU, 8000, 1, 0},
    {AV_CODEC_ID_PCMA, "PCMA", RTP_PT_PCMA, 8000, 1, 0},
    {AV_CODEC_ID_MP4A_LATM, "MP4A", RTP_PT_DYN_BASE, 8000, 2, 0},
    {AV_CODEC_ID_MP4V_ES, "MP4V", RTP_PT_DYN_BASE, 90000, 0, 0},
};
```

1.2.4 Video frame control



ISP overflow control

- ISP output buffer full
ISP output buffer 的設置與 h264 encode 的速度有關(1080P 30ms, 720P 15ms) , 設太小會導致 buffer full , 進而啟動 ISP overflow control 機制。
- ISP output buffer setting

建議 FPS 30 設 3，FPS 15 設 2。

ISP drop frame

- H264 output buffer full
H264 output buffer full 時，會 drop 掉當前的 ISP frame，不進入 H264 encode 的程序。
- H264 output buffer setting
H264 output buffer 設置過大，ISP drop frame 機制可能失效且易造成 latency 提高。
H264 output buffer 設置過小，緩存效果不足，可能導致 ISP drop frame 增加。
Streaming 時建議設置為 6，Storage 時建議設為 70。

H264 drop frame

[CMD_RTSP2_SET_DROP_FRAME_EN](#)
[CMD_RTSP2_SET_DROP_TIME](#)
[CMD_RTSP2_SET_DROP_FRAME_FORCEI](#)

- Packet send block
開啟 H264 drop frame 程序時，當傳輸 frame 的時間與 frame 產出的時間大於 drop threshold，則啟動 H264 drop frame 的處理程序。
Drop frame 時，會 drop 至下一張 I frame 為止，避免產生破圖。
- Drop threshold setting
Drop threshold 設置太小容易導致 H264 drop frame 機制頻繁觸發，出現卡頓現象。建議設置為 H264 output buffer 緩存的 3 倍時間。
- Drop frame and forcei
改善一次 drop 大量 frame 的問題，但 I frame size 較大，需要更大的頻寬傳輸，可能導致 FPS 一直維持在較低的狀況。

1.2.5 JPEG

```
jpeg_params_t jpeg_v3_params = {  
    .width      = V3_WIDTH,  
    .height     = V3_HEIGHT,  
    .level      = V3_JPEG_LEVEL,  
    .fps        = V3_FPS,  
    .mem_total_size = V3_BUFFER_SIZE,  
    .mem_block_size = V3_BLOCK_SIZE,  
};
```

- **WIDTH** : 影像寬度
- **HEIGHT** : 影像長度
- **LEVEL** : 影像品質 0 - 9 ,越大數字,畫質越好,但是容量越大
- **FPS**: 每秒多少張 FRAME
- **Mem_total_size** : JPEG encoder memory size 容量
- **Mem_block_size** : Memory pool 所使用的 block size
- **Mem_frame_size** : 設定一個 FRAME SIZE 最大容量

1.2.6 AAC Encoder

```
aac_params_t aac_params = {  
    .sample_rate = 8000,  
    .channel = 1,  
    .bit_length = 16,  
    .mem_total_size = 10*1024,  
    .mem_block_size = 128,  
};
```

- **sample_rate** : 必須與 Audio codec 設定相同 , 例如 Audio codec 設定為 ASR_8KHZ 時 , 這裡必須設定為 8000
- **channel** : mono 設定為 1 , stereo 設定為 2 , 本設定與 Audio codec 相關 , Amebapro 內建的 codec 收音為 mono , 所以這裡設定為 1
- **bit_length** : 必須與 Audio codec 的 word_length 相同 , 例如 Audio codec word_length = WL_16BIT , 這裡必須設定為 16
- **Mem_total_size** : AAC encoder output memory size 容量

- Mem_block_size : Memory pool 所使用的 block size
- Men_frame_size : 設定一個 FRAME SIZE 最大容量

1.2.7 AAC Decoder

```
aad_params_t aad_params = {  
    .sample_rate = 8000,  
    .channel = 1,  
    .type = TYPE_ADTS
```

- sample_rate : 需與來源來同才能正確解碼
- channel : 需與來源來同才能正確解碼
- type : 來源是 AAC encoder 時使用 TYPE_ADTS , 來源是 RTP 時使用 TYPE_RTP_RAW , 目前不支援 TYPE_TS

1.2.8 Audio Codec

AEC (Acoustic Echo Cancellation) 包含在此 module 中。

```
audio_params_t audio_params = {  
    .sample_rate = ASR_8KHZ,  
    .word_length = WL_16BIT,  
    .mic_gain = MIC_40DB,  
    .channel = 1,
```

- 取樣頻率目前支援 : 8K、16K、32K、44.1K、48K、88.2K、96K HZ
- word_length 目前支援 : 16、24 bit
- 麥克風 gain 值支援 : 0、20、30、40 DB
- Channel 目前支援 mono , 設定為 1
- enable_aec 如果為 1 將會開啟 echo cancellation , 反之沒有設定或是為 0 時則會關閉 echo cancellation

1.2.9 RTP Input

```
rtp_params_t rtp_aad_params = {  
    .valid_pt = 0xFFFFFFFF,  
    .port = 16384,  
    .frame_size = 1500,  
    cache_depth = 6  
}
```

- Valid_pt : 可處理的 RTP Payload types , 設 0x FFFFFFFF 則可處理 PCMU(0), PCMA(8), DYNAMIC (96)
- Port : 接收 RTP 封包的 port
- Frame_size: 最大的 RTP 封包大小
- Cache_depth : RTP 封包的 cache 數量。cache handler 在 cache 的封包數 >= 50% cache_depth 時才會將 cache 內的 RTP 封包往 module 的 output 送

1.2.10 G711 Codec

G711 Encode

```
g711_params_t g711e_params = {  
    .codec_id = AV_CODEC_ID_PCMU,  
    .buf_len = 2048,  
    .mode = G711_ENCODE  
}
```

G711 Decode

```
g711_params_t g711d_params = {  
    .codec_id = AV_CODEC_ID_PCMU,  
    .buf_len = 2048,  
    .mode = G711_DECODE  
}
```

codec_id : G711 目前支援 PCMU、PCMA 兩種 codec mode

buf_len : 用來決定 encode buffer 的長度(byte)

mode : 用來決定該 G711 codec module 是 encode 或 decode

1.2.11 MP4

```
mp4_params_t mp4_params = {  
    .width      = V2_WIDTH,  
    .height     = V2_HEIGHT,  
    .fps        = V2_FPS,  
    .gop        = V2_FPS,  
  
    .sample_rate = 8000,  
    .channel     = 1,  
  
    .record_length = 30, //seconds  
    .record_type  = STORAGE_ALL,
```

- Width : 影像長度
- Height : 影像高度
- FPS : 每秒的 FRAME 數目
- GOP : 更新 I FRAME 週期
- SAMPLE RATE : AUDIO SAMPLE RATE
- CHANNEL: Audio channel number.
- RECORD LENGTH : 錄影長度,以秒為單位
- Record type:可選擇 STORAGE_ALL(含影音) , STORAGE_VIDEO(僅影像) ,
STORAGE_AUDIO(僅聲音)
- REOCD FILE NUM:錄影個數
- RECORD FILE NAME:錄影名稱
- FATFS BUF SIZE:FATFS 緩存 BUFFER

1.2.10 I2S

```
typedef struct i2s_param_s{  
    i2s_sr          sample_rate;          // SR_32KHZ  
    i2s_sr          out_sample_rate;      // SR_8KHZ  
    i2s_wl          word_length;         // WL_24b  
    i2s_wl          out_word_length;     // WL_16b  
    int             channel;              // 2  
    int             out_channel;          // 1  
    int             enable_aec;          // 0  
}
```

- sample_rate 目前支援：8K、12K、16K、24K、32K、48K、64K、96K、192K、384K、7.35K、11.025K、14.7K、22.05K、44.1K、58.8K、88.2K、176.4K HZ
- out_sample_rate 目前支援的取樣平率與 sample_rate 相同，但需小於或等於 sample_rate
- word_length 目前支援：16、24、32 bit
- out_word_length 目前支援的長度與 word_length 相同，但需小於或等於 word_length
- Channel 目前支援 stereo 或 mono，設定為 2 或 1，另外也支援 5.1 聲道(但為 Tx only)

1.2.11 httpfs

```
httpfs_params_t httpfs_params = {  
    .fileext = "mp4",  
    .filedir = "VIDEO",  
    .request_string = "/video_get.mp4",  
    .fatfs_buf_size = 1024  
};
```

- fileext：檔案的副檔名
- filedir：檔案所在的目錄
- request_string：http page 字串
- fatfs_buf_size：一次讀檔的 buffer 大小

1.3 使用 MMF v2 範例

介紹如何使用範例程式建構出終端應用需要的資料流。

1.3.1 範例程式

範例程式位於：

component\common\example\media_frameworkexample_media_framework.c

使用前必須設定 platform_opts.h

開啟位於 project\realtek_amebapro_v0_example\inc\platform_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK
0
#if CONFIG_EXAMPLE_MEDIA_FRAMEWORK
```

修改 CONFIG_EXAMPLE_MEDIA_FRAMEWORK 由 0 改為 1，編譯後燒錄

```
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK
1
#if CONFIG_EXAMPLE_MEDIA_FRAMEWORK
```

1.4 選擇和設定範例程式

步驟：

- 選擇適合的範例程式
- 如何調整 Audio/Video 參數

 **注意：** IAR 版本需使用 8.30

1.4.1 選擇合適的範例程式

範例程式主程式名稱為 example_mmf2_signal_stream_main，所有範例在預設的狀態都是被註銷掉。使用前先挑選想要開啟的範例，將注釋符號移除，重新編譯。同時開啟兩個以上的範例將會導致程式執行結果無法預期。

```
void example_mmf2_signal_stream_main(void *param)
{
    //int ret;

    #if ISP_BOOT_MODE_ENABLE == 0
        common_init();
    #endif

    // CH1 Video -> H264 -> RTSP
    //mmf2_example_v1_init();

    // CH2 Video -> H264 -> RTSP
    //mmf2_example_v2_init();

    // CH3 Video -> JPEG -> RTSP
    #if ENABLE_V3_JPEG == V3_JPEG_STREAMING
        mmf2_example_v3_init();
    #endif
}
```

例如要開啟範例程式中的第一個範例

```
// CH1 Video -> H264 -> RTSP
mmf2_example_v1_init();

// CH2 Video -> H264 -> RTSP
```

目前支援的範例程式

範例	範例描述	執行結果
mmf2_example_v1_init	CH1 Video -> H264 -> RTSP	透過網路傳輸 AmebaPro 的 H264 影像串流
mmf2_example_v2_init	CH2 Video -> H264 -> RTSP	透過網路傳輸 AmebaPro 的 H264 影像串流
mmf2_example_v3_init	CH3 Video -> JPEG -> RTSP	透過網路傳輸 AmebaPro 的 JPEG 影像串流
mmf2_example_simo_init	1Video (H264) -> 2 RTSP(V2)	透過網路傳輸 AmebaPro 的兩份 H264 影像串流，影像源頭為同一個 ISP 串流
mmf2_example_a_init	1 Audio (AAC) -> RTSP (A)	透過網路傳輸 AmebaPro 的 AAC 聲音串流



mmf2_example_av_init	1 Video (H264) 1 Audio -> RTSP	透過網路傳輸 AmebaPro 的 H264 影像與 AAC 聲音串流
mmf2_example_av2_init	2 Video (H264) 1 Audio -> 2 RTSP (V1+A, V2+A)	透過網路傳輸 AmebaPro 的兩份 H264 影像與 AAC 聲音串流，影像源頭為不同的 ISP 串流
mmf2_example_av21_init	1 Video (H264) 1 Audio -> 2 RTSP (V+A)	透過網路傳輸 AmebaPro 的兩份 H264 影像與 AAC 聲音串流，影像源頭為同一個 ISP 串流
mmf2_example_audioloop_init	PCM audio -> PCM audio , audio loopback	從 AmebaPro 的 3.5 音訊孔播出 AmebaPro 收到的聲音，中間直接使用 PCM 傳輸
mmf2_example_g711loop_init	audio -> G711E -> G711D -> audio	從 AmebaPro 的 3.5 音訊孔播出 AmebaPro 收到的聲音，將 PCM 經由 G711 編碼後傳輸
mmf2_example_aacloop_init	audio -> AAC -> AAD -> audio	從 AmebaPro 的 3.5 音訊孔播出 AmebaPro 收到的聲音，將 PCM 經由 AAC 編碼後傳輸
mmf2_example_i2s_audio_init	I2s -> PCM audio, audio loop back	從 AmebaPro 的 3.5 音訊孔播出 i2s 收到的聲音，中間直接使用 PCM 傳輸
mmf2_example_rtp_aad_init	RTP -> AAD -> audio	透過網路將 AAC 聲音串流傳輸至 AmebaPro 播放
mmf2_example_2way_audio_init	AUDIO -> AAC -> RTSP RTP -> AAD -> AUDIO	透過網路將 AAC 聲音串流傳輸至 AmebaPro 播放，並將 AmebaPro 收到的聲音透過網路傳出
mmf2_example_joint_test_init	ISP -> H264 -> RTSP (with AUDIO) ISP -> H264 -> RTSP (with AUDIO) AUDIO -> AAC -> RTSP RTP -> AAD -> AUDIO	(1) 透過網路傳輸 AmebaPro 的兩份 H264 影像與 AAC 聲音串流，影像源頭為不同的 ISP 串流 (2) 可透過網路將 AAC 聲音串流傳輸至 AmebaPro 播放
mmf2_example_av_mp4_init	1 Video (H264) 1 Audio -> MP4 (SD card)	AmebaPro 會錄製三段影片存到 SD 卡中，每段 30 秒 預設儲存名稱為 AmebaPro_recording_0.mp4 AmebaPro_recording_1.mp4 AmebaPro_recording_2.mp4
mmf2_example_joint_test_rtsp_mp4_init	ISP -> H264 -> RTSP (V1) ISP -> H264 -> MP4 (V2) AUDIO -> AAC -> RTSP and mp4 RTP -> AAD -> AUDIO	(1)透過網路傳輸 AmebaPro 的 H264 影像與 AAC 聲音串流 (2)錄製三段影片存到 SD 卡中，每段 30 秒 預設儲存名稱為 AmebaPro_recording_0.mp4 AmebaPro_recording_1.mp4 AmebaPro_recording_2.mp4 (3)可透過網路將 AAC 聲音串流傳輸至 AmebaPro 播放 p.s.(1)與(2)的影像源頭為不同的 ISP 串流
mmf2_example_h264_2way_	ISP -> H264 -> RTSP (V1)	(1)透過網路傳輸 AmebaPro 的 H264 影像與 G711 聲音串流

audio_pcmu_doorbell_init	AUDIO -> G711E -> RTSP RTP -> G711D -> AUDIO ARRAY (PCMU) -> G711D -> AUDIO (doorbell)	(2)可透過網路將 G711 聲音串流傳輸至 AmebaPro 播放 可播放 AmebaPro 內的 G711 聲音陣列 (預設為門鈴聲)
mmf2_example_pcmu_array_rtsp_init	ARRAY (PCMU) -> RTSP (A)	透過網路傳 AmebaPro 內的 PCMU 聲音陣列
mmf2_example_aac_array_rtsp_init	ARRAY (AAC) -> RTSP (A)	透過網路傳 AmebaPro 內的 AAC 聲音陣列
mmf2_example_h264_array_rtsp_init	ARRAY (H264) -> RTSP (V)	透過網路傳 AmebaPro 內的 H264 影像陣列
mmf2_example_v1_param_change_init	H264/ISP parameter change	透過網路傳 AmebaPro 的 H264 影像，動態調整影像參數
mmf2_example_av_mp4_https_init	1 Video (H264) 1 Audio -> MP4 (SD card) Http File Server	AmebaPro 會每 30 秒錄製一段影片存到 SD 卡中，預設為錄製 30 個檔案，結束後重頭循環錄製，預設儲存名稱為 mp4_record_0.mp4~mp4_record_29.mp4 同時開啟 Http File Server 供 client 端做 playback

請參考下方步驟了解如何使用 MMFv2 範例

準備項目

- AmebaPro 版子 * 1
- Camera 版子 * 1
- USB 線 * 1
- 用來傳輸 RTSP 串流的 Wifi
- MicroSD * 1

硬體組裝

- 將 Camera 版子接到 AmebaPro CON1 port 上
- 將 USB 線接到 AmebaPro CON8 port 上，另一端接上 PC
- 將 MicroSD 卡接到 MicroSD 卡插槽中

軟體設定

請參考 [AN0300 Realtek AmebaPro application note.cn](#) 編譯和執行固件

- 打開 SDK\project\realtek_amebapro_v0_example\inc\platform_opts.h
- 打開 example_media_framework , 將 CONFIG_EXAMPLE_MEDIA_FRAMEWORK 設為 1

```
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK 1
#if CONFIG_EXAMPLE_MEDIA_FRAMEWORK
    #define FATFS_DISK_SD 1
#endif
```

- 打開 SDK\common\example\media_framework\example_media_framework.c , 在接近最下方 example_mm2_signal_stream_main 中選擇想要使用的範例程式。欲測試 full function 範例程式建議將 mmf2_example_joint_test_rtsp_mp4_init(); 打開並編譯。

```
// Joint test RTSP MP4
// ISP -> H264 -> RTSP (V1)
// ISP -> H264 -> MP4 (V2)
// AUDIO -> AAC -> RTSP and mp4
// RTP -> AAD -> AUDIO

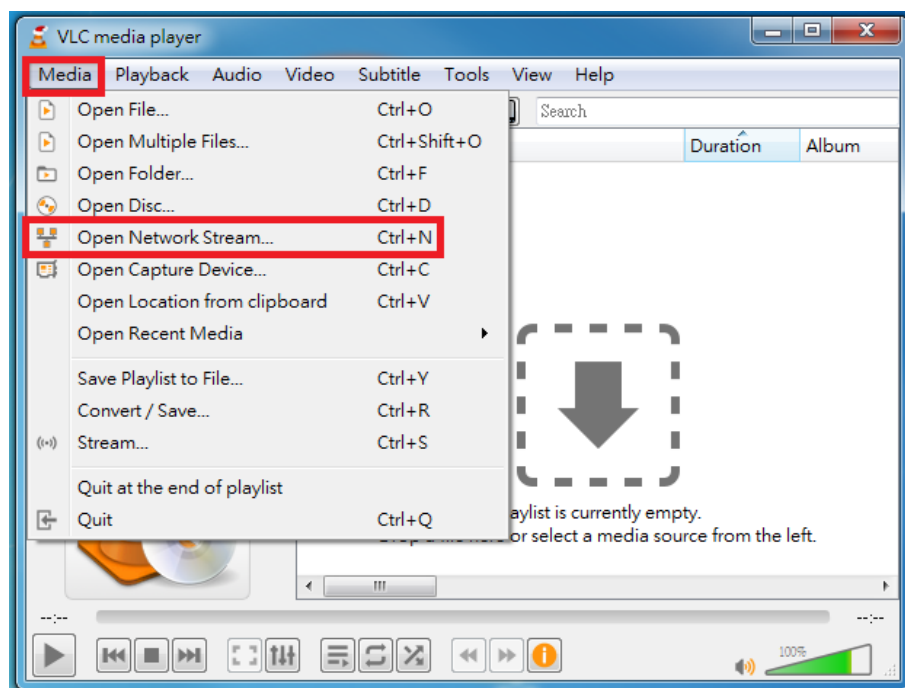
mmf2_example_joint_test_rtsp_mp4_init();
```

- 編譯與執行固件

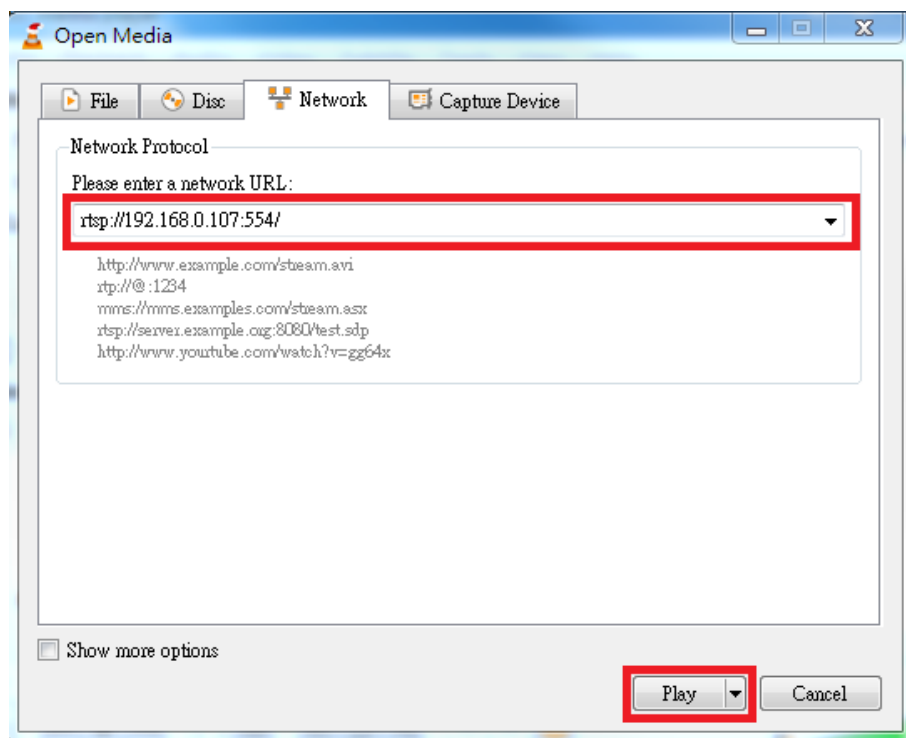
執行與測試

- 開始執行前須先設定好 Tera Term 或試 PuTTY 與設定好 serial port 為 COMX/115200:Port number , 一旦設定完成 AmebaPro 也已經與 PC 端連接好並開機即可拿到 AmebaPro 輸出的 Log 訊息
- 為了執行 rtsp stream 須先設定 AmebaPro 連上網路, 請參照下方步驟
ATW0 = <Name of WiFi SSID> : Set the WiFi AP to be connected
ATW1 = <Password> : Set the WiFi AP password
ATWC : Initiate the connection
- 當 Console 訊息中顯示“RTSP stream enabled” , 表示 RTSP server 已在運作 , 若要播放 AmebaPro 的影音串流, 請開啟 VLC media player

1. 選擇 “Media - > Open Network Stream”



2. 輸入 “rtsp://xxx.xxx.xxx.xxx:yyy/” 後選擇 “Play”



xxx.xxx.xxx.xxx : the Ameba IP address.

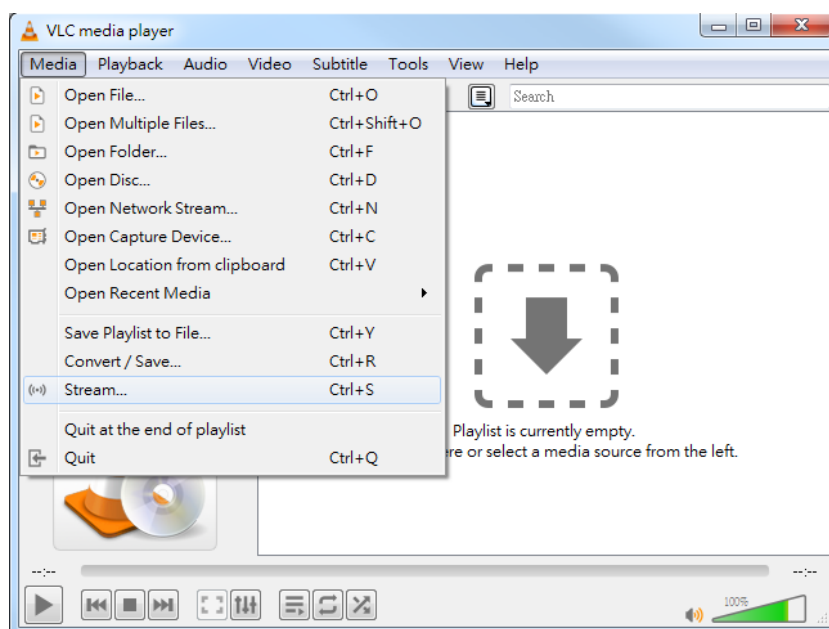
yyy : RTSP server port (default is 554).

 注意：若要調整 VLC 播放延遲等相關設定，請參考章節 1.4.5

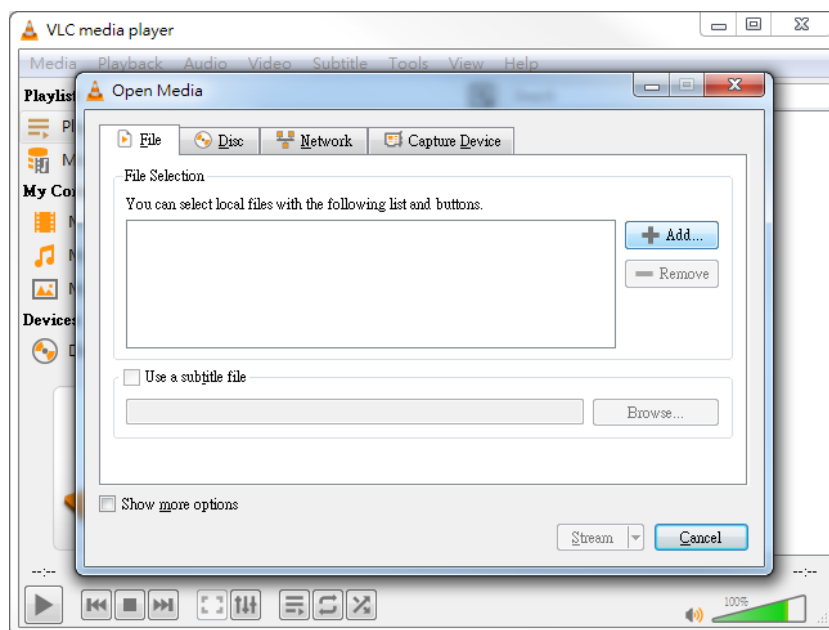
VLC media player 設定

- 若要傳送聲音串流至 AmebaPro 播出，請打開 VLC media player

1. 選擇“Media” -> “Stream”



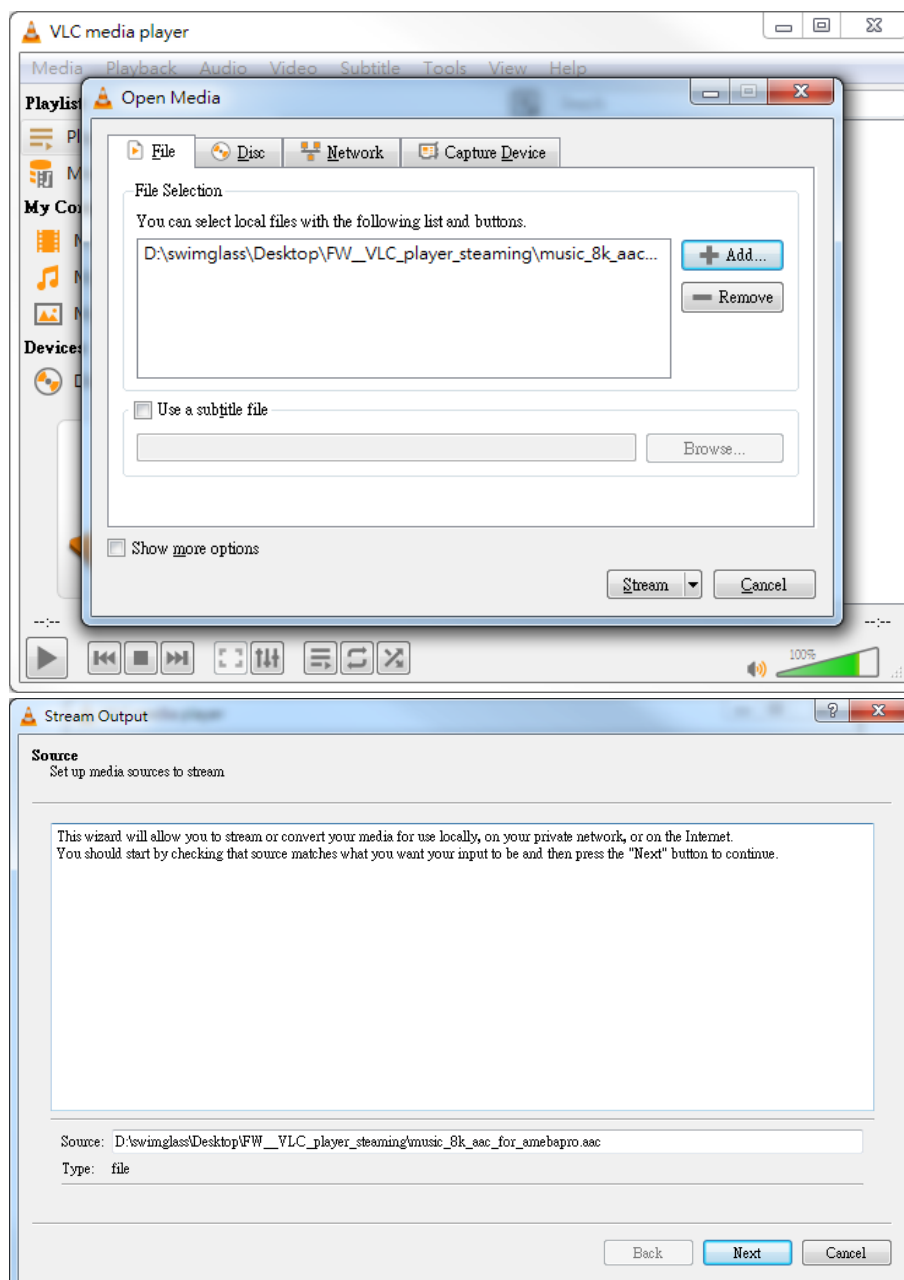
2. 選擇“File” 並選擇“Add”



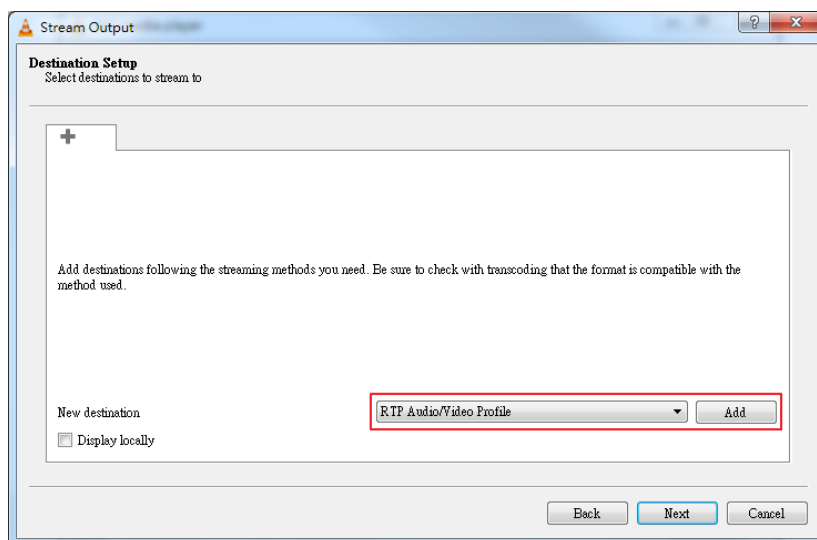
(若啟動的範例為 RTP -> AAD -> AUDIO 請選擇附檔名為.aac 之音訊檔，
若啟動的範例為 RTP -> G711D -> AUDIO 請選擇附檔名為.wav(需為 pcm
mu law)之音訊檔)

 **注意：**請下載並使用 ffmpeg 產生相容的 WAV 檔, *command 如下*
ffmpeg -i input.wav -acodec pcm_mulaw -ac 1 -ar 8000 -ab 64k output.wav

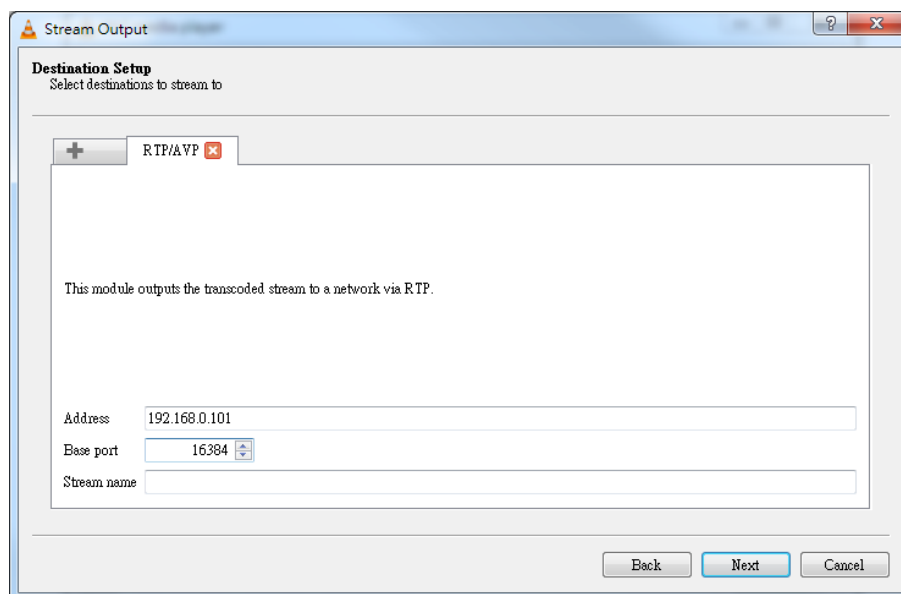
3. 加入音訊檔後選擇“Stream”，再按下“Next”



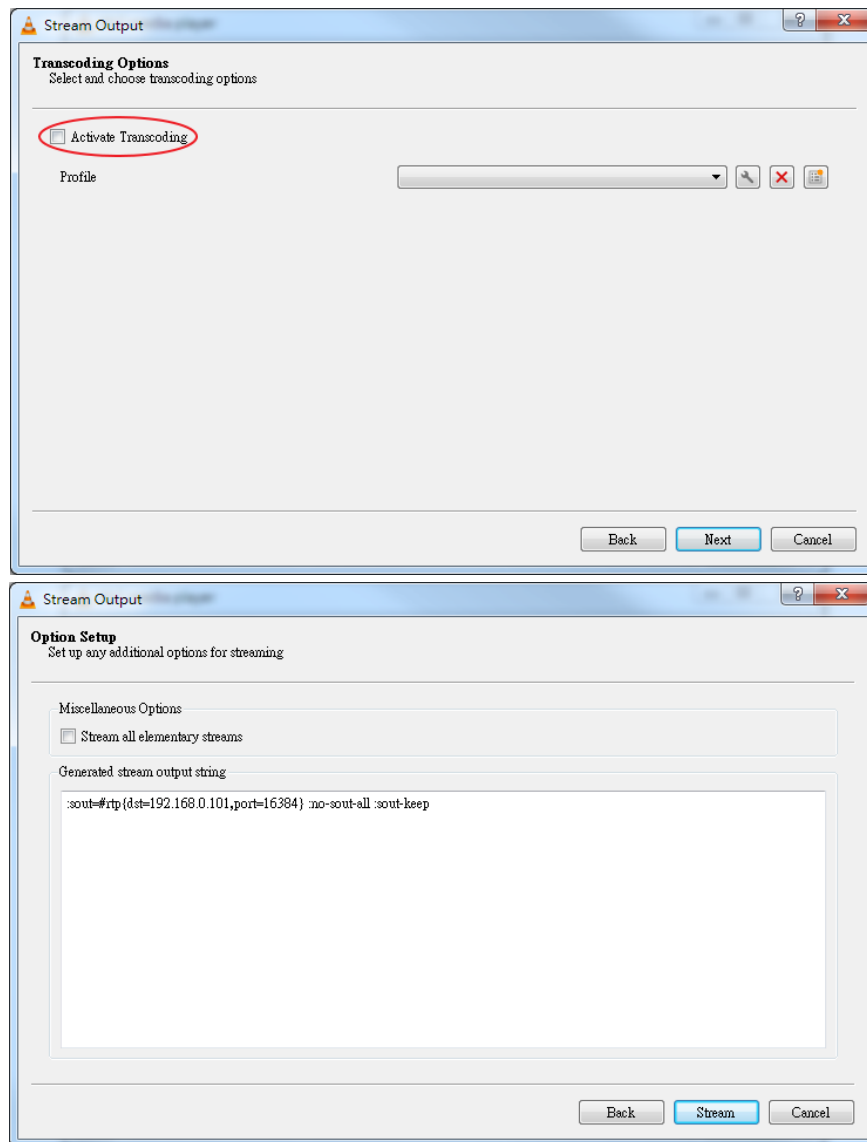
4. 選擇“RTP Audio/Video Profile” 並按下“Add”



5. 在 Address 處輸入 AmebaPro 的 IP Address , 在 Base port 處輸入 16384 , 並按下 Next



6. 確認 Activate Transcoding 是未被勾選的，按下“Next”，最後按下
“Stream”即可在 AmebaPro 上聽到從 VLC player 選擇的檔案播出的聲音



個別說明與編譯選項

如有超過前述說明補充如下

- mmf2_example_v1_init(Source AmebaPro Camera, Sink RTSP Stream) :
如要修改影像參數可以修改 example_media_framework.h 中的 V1 參數
ISP 相關參數設定請參考章節 1.2.1，H264 相關參數請參考章節 1.2.2
- mmf2_example_v2_init(Source AmebaPro Camera, Sink RTSP Stream) :

如要修改影像參數可以修改 `example_media_framework.h` 中的 V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2 , 如果要開啟 ISP BOOT MODE,請 Enable `ISP_BOOT_MODE_ENABLE` 以及 `ISP_BOOT_RTSP`, 另外 VLC 不會是即時的影像,而是當時 BOOT TIME 影像

- `mmf2_example_v3_init(Source AmebaPro Camera, Sink RTSP Stream)` :

如要修改影像參數可以修改 `example_media_framework.h` 中的 V1 參數

另外須在 `example_media_framework.h` 中設定

```
#define ENABLE_V3_JPEG          V3_JPEG_STREAMING
```

ISP 相關參數設定請參考章節 1.2.1 , JPEG 相關參數請參考章節 1.2.3

- `mmf2_example_simo_init(Source AmebaPro Camera, 2 Sink RTSP Stream)` :

須同時打開兩個 VLC video player , 另外一個 RTSP 的 port 為 555

如要修改影像參數可以修改 `example_media_framework.h` 中的 V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

- `mmf2_example_a_init(Source AmebaPro Microphone, Sink RTSP Stream)` :

音訊相關參數請參考章節 1.2.4~1.2.6

- `mmf2_example_av_init(Source AmebaPro Camera/Mic, Sink RTSP Stream)` :

如要修改影像參數可以修改 `example_media_framework.h` 中的 V1 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

音訊相關參數請參考章節 1.2.4~1.2.6

- `mmf2_example_av2_init(Source AmebaPro Camera/Mic, Sink RTSP Stream)` :

須同時打開兩個 VLC video player , 另外一個 RTSP 的 port 為 555

如要修改影像參數可以修改 `example_media_framework.h` 中的 V1,V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

音訊相關參數請參考章節 1.2.4~1.2.6

- `mmf2_example_av21_init(Source AmebaPro Camera/Mic, Sink RTSP Stream)` :

須同時打開兩個 VLC video player , 另外一個 RTSP 的 port 為 555

如要修影像參數可以修改 `example_media_framework.h` 中的 V1 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_audioloop_init(Source AmebaPro Microphone, Sink audio jack) :

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_g711loop_init(Source AmebaPro Microphone, Sink audio jack) :

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_aacloop_init(Source AmebaPro Microphone, Sink audio jack) :

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_2way_audio_init(Source AmebaPro Microphone, Sink audio jack and Source RTP, Sink audio jack) :

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_joint_test_init(Source AmebaPro Camera/Mic, Sink RTSP Stream and and Source RTP, Sink audio jack) :

須同時打開兩個 VLC video player , 另外一個 RTSP 的 port 為 555

如要修改影像參數可以修改 example_media_framework.h 中的 V1,V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_av_mp4_init(Source AmebaPro Camera/Mic, Sink SD card) :

如要修改影像參數可以修改 example_media_framework.h 中的 V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2

音訊相關參數請參考章節 1.2.4~1.2.6 , 如果要開啟 ISP BOOT MODE,請 Enable

ISP_BOOT_MODE_ENABLE 以及 ISP_BOOT_MP4

- mmf2_example_joint_test_rtsp_mp4_init(Source AmebaPro Camera/Mic, Sink RTSP Stream and Source AmebaPro Camera/Mic, Sink SD card and and Source RTP, Sink audio jack) :

如要修改影像參數可以修改 example_media_framework.h 中的 V1,V2 參數

ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2


音訊相關參數請參考章節 1.2.4~1.2.6

- mmf2_example_h264_2way_audio_pcmu_doorbell_init(Source AmebaPro Camera/Mic, Sink RTSP Stream and and Source RTP, Sink audio jack) :
如要修改影像參數可以修改 example_media_framework.h 中的 V1,V2 參數
ISP 相關參數設定請參考章節 1.2.1 , H264 相關參數請參考章節 1.2.2
音訊相關參數請參考章節 1.2.4~1.2.6
從 PC 端播放之音樂附檔名請使用.wav 之音訊檔
- mmf2_example_pcmu_array_rtsp_init(Source Music file in memory, Sink RTSP)
- mmf2_example_aac_array_rtsp_init(Source Music file in memory, Sink RTSP)
- mmf2_example_h264_array_rtsp_init(Source video file in memory, Sink RTSP)

1.4.2 調整範例程式的 Video/Audio 參數

- ISP

```
isp_params_t isp_v1_params = {  
    .width  = V1_WIDTH,  
    .height = V1_HEIGHT,  
    .fps    = V1_FPS,  
    .slot_num = V1_HW_SLOT,  
    .buff_num = V1_SW_SLOT
```

 Video/Audio/ISP 參數並非任意可調整，在 RTOS 系統上，資源配置需要精算，請按照建議的範圍內調整

- 解析度：支援 1080P 含以下的設定。
- FPS：目前開放設定 30，15，10, 5 以及 1。
- HW_SLOT：目前最大支援 4 個
- SW_SLOT：會包含 HW_SLOT 數目,剩餘的會作為緩存

關於檔案大小計算，目前預設 ISP 輸出格式為 YUV420 Format，假設目前設定解析度為 1080P, 那麼一個 FRAME 大小為 $1920 \times 1080 \times 1.5 = 3110400$ bytes, 若

HW_SLOT = 2, SW_SLOT=4, 那麼消耗的記憶體大小為 $3110400 \times 4 = 12441600$ bytes。

關於 SLOT 數目設定，目前建議 HW_SLOT 設定為 2，SW_SLOT 會根據 FPS 大小決定,如果為 30FPS 建議設為 4,如果為 15FPS 建議設為 3。

1.4.3 調整 LWIP 參數

由於在影像傳輸的應用上對網路效能有一定的要求，SDK 內預設有對 LWIP 做參數上的調整，以增加 buffer 的方式提升傳輸效率，此段設定位於 component\common\api\network\include\lwipopts.h 檔案內：

```
#if CONFIG_VIDEO_APPLICATION
#undef MEM_SIZE
#define MEM_SIZE (20*1024)

#undef MEMP_NUM_TCP_SEG
#define MEMP_NUM_TCP_SEG 60

#undef PBUF_POOL_SIZE
#define PBUF_POOL_SIZE 60

#undef MEMP_NUM_NETBUF
#define MEMP_NUM_NETBUF 60

#undef DEFAULT_UDP_RECVMBOX_SIZE
#define DEFAULT_UDP_RECVMBOX_SIZE 60

#undef IP_REASS_MAX_PBUFS
#define IP_REASS_MAX_PBUFS 40

#undef TCP_SND_BUF
#define TCP_SND_BUF (10*TCP_MSS)

#undef TCP_SND_QUEUELEN
#define TCP_SND_QUEUELEN (6*TCP_SND_BUF/TCP_MSS)

#undef TCP_WND
#define TCP_WND (4*TCP_MSS)
#endif
```

其中 CONFIG_VIDEO_APPLICATION 在 project\realtek_amebapro_v0_example\inc\platform_opts.h 內預設開啟為 1。

若使用者在應用上有其他特殊需求，可以考量記憶體剩餘狀況，再基於此設定做進一步調整。以 TCP 發送串流來說，若可能遇到 RTT (Round Trip Time) 較長的情況，例如傳輸數據至國外網路，可以藉由增加 TCP_SND_BUF 讓 TCP 傳輸中一次發送多筆數據，藉此避免 RTT 拉低傳輸效率，以 standard SDK 來說，透過將 LWIP 部分 buffer 改移至 external RAM 可以將 LWIP 部分設定修改為以下讓 TCP_SND_BUF 調整至最大值：

```
#define MEM_SIZE (55*1024)
#define MEMP_NUM_TCP_SEG 300
#define PBUF_POOL_SIZE 220
#define TCP_SND_BUF (44*TCP_MSS)
```

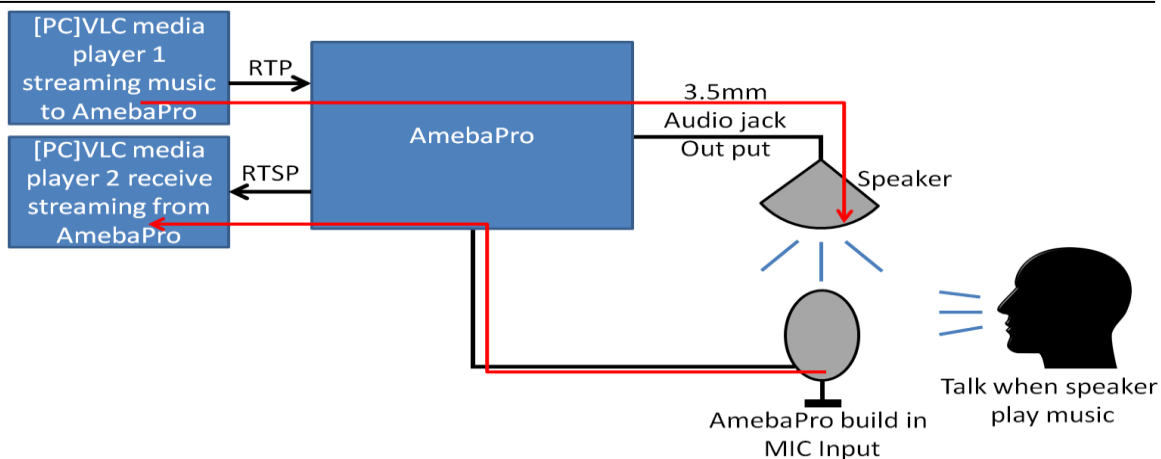
以上設定藉由修改 project\realtek_amebapro_v0_example\EWARM-RELEASE\application_is.icf 將"section *.itcm.bss* object memp.o,"由 DTCM_RAM_region 改放至 ERAM_BSS , external RAM 記憶體約需額外占用 120KB。

1.4.4 Echo Cancellation

MMFv2 的 audio 預設都有提供 echo cancellation，若要測試 echo cancellation 功能是否正確可以透過電腦端使用 VLC media player 來驗證。

其驗證方式如下所述：

1. 在 PC 端使用 VLC media player 將音樂串流至 AmebaPro 撥放
2. 將 AmebaPro 端撥放用的喇叭放到 AmebaPro build in Mic 旁邊並同時講話
3. 再經由 AmebaPro 將收到的聲音傳到 PC 端的 VLC media player 撥放出來看看在步驟 1 的聲音有沒有比較小或沒有(同時有講話的聲音與音樂的聲音，但音樂的聲音比較小聲)



參數調整

Speex echo cancellation 主要提供三種參數可以調整，sampling rate、frame size 與 filter length。Sampling rate 建議使用 8kHz，對應的 frame size 為 20ms。Frame size 是在樣本中一次想要處理的數據量。Filter length 是在樣本中要使用的 echo cancellation filter 的長度（也叫做 tail length）。建議 frame_size 使用大小為 20 ms（或等於 codec frame size 大小）的大小，並確保很容易執行該大小的 FFT。建議的 filter_length 約為周邊回應時間的三分之一。例如，在小房間中回應時間大約為 300 毫秒，因此 100 毫秒的 filter_length 是一個不錯的選擇（8000 採

 *echo cancellation example 包含於 MMFv2 audio example 中*

樣率的 8000 個樣本）。如需更詳細的設定請參考 Speex 文件說明。

1.4.5 擷取第一張 frame,相關注意事項

此功能目的可以加快 ISP Frame 出現時間, 目前量測時間從開機到 frame done 大概約 188ms,做法主要在 bootloader 內先啟動 ISP 初始化,此時就可以在 main 內等待 frame 出現,詳細主要說明可以參考 1.2.1.

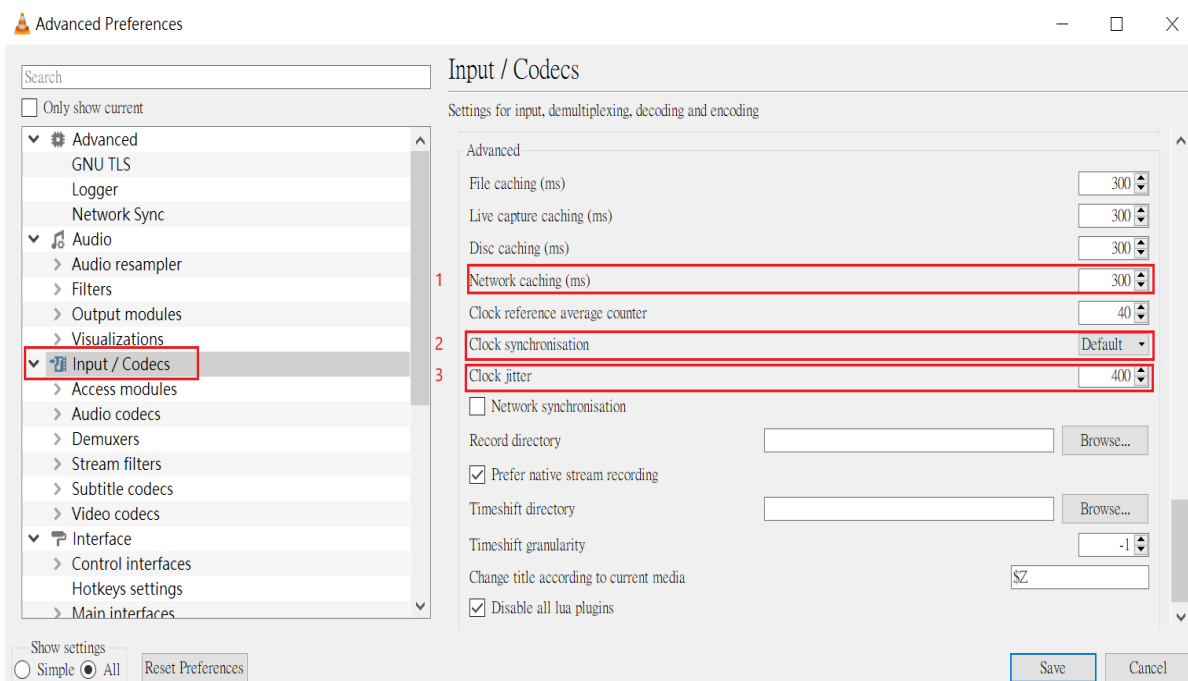
注意事項:同時開兩路,需先執行 First boot 那一路,可以參考 ISP_BOOT_MUX 範例,另外如果之後需要更改 Frame rate 請在設定完成之後將 Flag 更改回 Normal boot.

1.4.6 VLC media player 設定

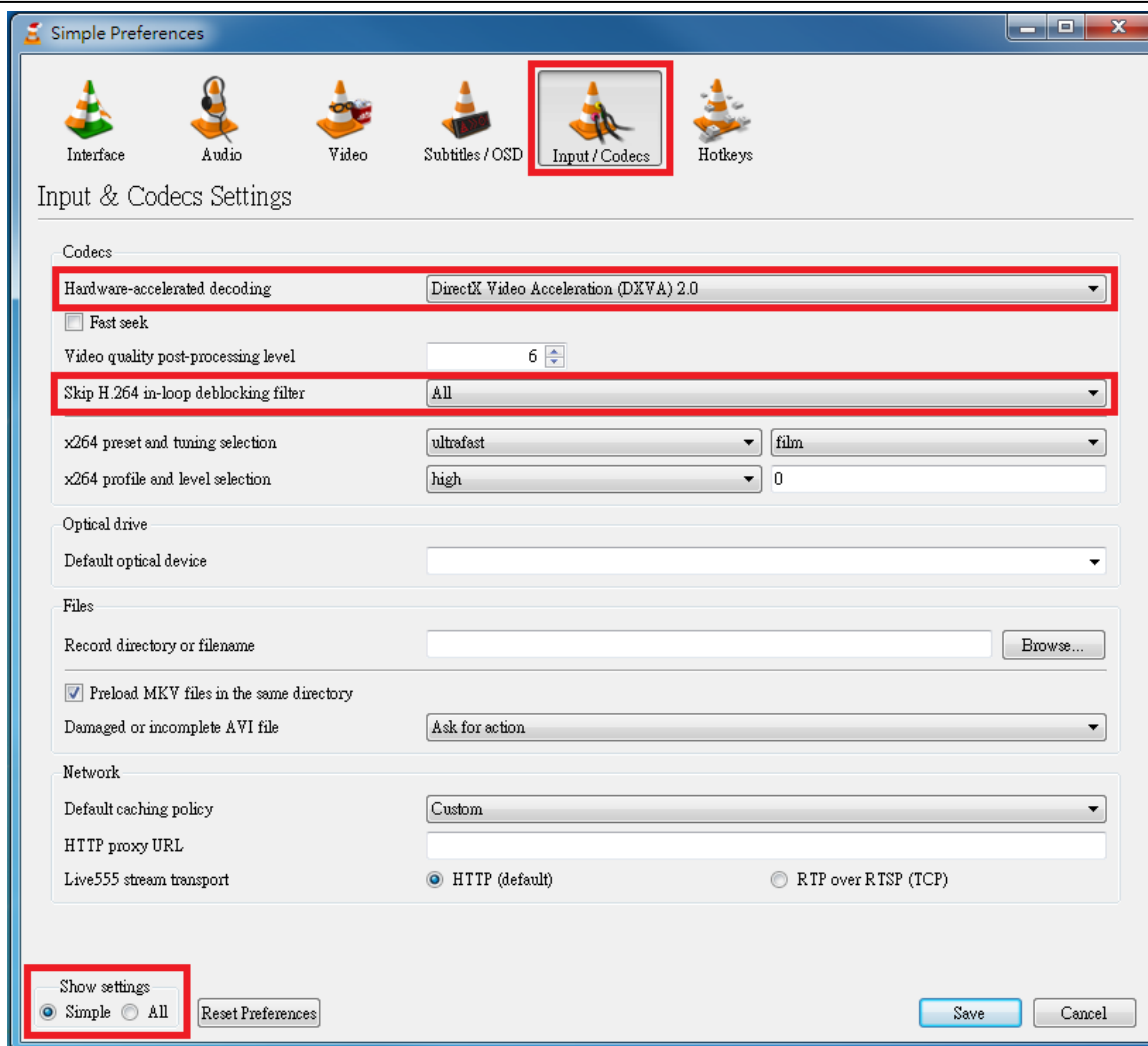
- 至 VLC 官網 <https://www.videolan.org/> 下載 VLC media player
- 調整網路延遲(緩存)相關設定

 *Note : Please use VLC media player version 2.2.4 or greater.*

1. 點擊工具列中“Tools” -> “Preferences” , 左下角 “Show settings” 選擇 All , 並點選左欄 “Input/ Codecs” , (1)將 “Network caching” 設為 300ms (建議值) , (2)將 “Clock synchronisation” 設為 Default , (3)將 “Clock jitter” 設為 400ms (建議值)。



2. 點擊工具列中“Tools” -> “Preferences” , 左下角 “Show settings” 選擇 Simple , 並於上方選擇 “Input/ Codecs” , 使用 “Hardware-accelerated decoding” , 並將 “Skip H.264 in-loop deblocking filter” 設為 “All”。



3. VLC 由“Network caching”及“Clock jitter”形成 pts delay buffer，此 buffer 的最大值等於“Network caching”加上“Clock jitter”，VLC 端的 video display 會因 pts delay buffer 增加而使 latency 增長，透過縮小“Network caching”及“Clock jitter”可達到縮短 latency 的效果。

- 使用 VLC player 播放來自 AmebaPro 的影音串流
請參考章節[“1.4.1 選擇合適的範例程式”](#) ->執行與測試
- 使用 VLC player 傳送聲音串流至 AmebaPro 播放
請參考章節[“1.4.1 選擇合適的範例程式”](#) ->執行與測試

2 Video API

目前包含 H264 以及 JPEG API

2.1 H264 API

2.1.1 h264_open

目的

建立 H264 encoder instance.

概要

```
void *h264_open();
```

參數

無

回傳值

假如回傳值不是 NULL，則表示回傳正確 Encoder 指針

2.1.2 h264_initial

目的

設定 H264 參數.

概要

```
int h264_initial(void *ctx, struct h264_parameter *h264_parm);
```

參數

void *ctx : Encoder 指針。

h264_parameter *h264_parm :

需要設定寬度、長度、GOP(Group of picture)、BPS(Bit per second)、

FPS(Frame per second) 和 Rate Control mode。 .

Rate Control : 目前支援 CBR(Constant bit rate), VBR(Variable bit rate) and

FIXQP(Fixed QP). VBR 模式目前可以支援設定 QP 最大以及最小值. FIXQP

模式表示最小和最大 QP 值是一樣的

回傳值

0 : 成功

-1: 失敗

2.1.3 h264_encode

目的

壓縮一張 Frame

概要

```
int h264_encode(void *ctx);
```

參數

目前需要設定 Input buffer address 資料，包含 Y 以及 UV 從 ISP buffer。
另外需要設定 Output buffer address 資料，包含目的 address 以及 buffer 長度。

回傳值

0：成功

可以取得壓縮成功的長度以及目的 address。

非零：影像壓縮失敗。

2.1.4 h264_release

目的

釋放壓縮資源。

概要

```
int h264_release(void *ctx);
```

參數

void *ctx 指向 Encoder 指針

回傳值

0: 成功.

非零 :釋放資源失敗

2.2 JPEG API

2.2.1 jpeg_open

目的

建立 JPEG encoder instance.

概要

```
void *jpeg_open();
```

參數

無

回傳值

假如回傳值不是 NULL，則表示回傳正確 Encoder 指針

2.2.2 jpeg_initial

目的

設定 JPEG 參數.

概要

```
int jpeg_initial(void *ctx, struct jpeg_parameter *jpeg_parm);
```

參數

void *ctx : Encoder 指標

struct jpeg_parameter *jpeg_parm:

目前需要設定寬度、長度和 Level(0~9), Level 越高畫質越好.

回傳值

0 : 成功

-1: 失敗

2.2.3 jpeg_encode

目的

壓縮一張 Frame

概要

```
int jpeg_encode(void *ctx);
```

參數

目前需要設定 Input buffer address 資料，包含 Y 以及 UV 從 ISP buffer。

另外需要設定 Output buffer address 資料，包含目的 address 以及 buffer 長度。

回傳值

0 : 成功

可以取得壓縮成功的長度以及目的 address。

非零：影像壓縮失敗。

2.2.4 jpeg_release

目的

釋放壓縮資源。

概要

```
int jpeg_release(void *ctx);
```

參數

void *ctx 指向 Encoder 指針

回傳值

0: 成功.

非零 :釋放資源失敗

3 ISP API

3.1 video_subsys_init

目的

初始化 video 環境設定。

概要

```
int video_subsys_init(isp_init_cfg_t *ctx);
```

參數

isp_cfg_t *cfg 指標

回傳值

0 : 成功

-1: 失敗

3.2 isp_stream_create

目的

建立 stream

概要

```
isp_stream_t* isp_stream_create(isp_cfg_t *cfg);
```

參數

isp_cfg_t *cfg 指標

isp_id 指定 Stream ID(0~2), Format 目前僅支援 YUV420 SEMI PLANAR, 長度, 寬度, FPS(30,15,10,5,1)以及 HW_SLOT 數目(硬體壓縮 BUFFER 個數)

回傳值

Null 失敗, 成功回傳 isp_stream_t 指標

3.3 isp_stream_destroy

目的

銷毀 stream

概要

```
isp_stream_t* isp_stream_destroy(isp_stream_t* stream);
```

參數

isp_stream_t* stream 指標

回傳值

回傳 NULL

3.4 isp_stream_set_complete_callback

目的

註冊 ISP FRAME 完成時的 CALLBACK FUNCTION

概要

```
isp_stream_t* isp_stream_destroy(isp_stream_t* stream);
```

參數

void (*cb)(void*) 使用者註冊的函數

void* arg 使用者所需要的參數

回傳值

回傳 NULL

3.5 isp_stream_apply

目的

將設定值填入 ISP

概要

```
void isp_stream_apply(isp_stream_t* stream);
```

參數

isp_stream_t* stream 指標

回傳值

無

3.6 isp_stream_start

目的

啟動 ISP 獲取 FRAME

概要

```
void isp_stream_start(isp_stream_t* stream);
```

參數

isp_stream_t* stream 指標

回傳值

無

3.7 isp_stream_stop

目的

停止 ISP

概要

```
void isp_stream_stop(isp_stream_t* stream);
```

參數

isp_stream_t* stream 指標

回傳值

無

3.8 isp_stream_poll

目的

查詢是否有 FRAME 完成

概要

```
int isp_stream_poll(isp_stream_t* stream)
```

參數

isp_stream_t* stream 指標

回傳值

0:成功 ; -1:失敗

3.9 isp_handle_buffer

目的

管理 ISP BUFFER

概要

```
void isp_handle_buffer(isp_stream_t* stream, isp_buf_t* buf, int mode);
```

參數

isp_stream_t* stream 指標

isp_buf_t* buf 可以取得 ISP BUFFER ADDRESS

mode 分為以下方式

MODE_EXCHANGE 將下一筆 ISP BUFFER 資料帶入,並將目前 ISP 取得的資料帶出

MODE_SNAPSHOT 將一筆 FRAME 提出,但是不繼續下一筆 FRAME 產生

MODE_SKIP 這一筆 FRAME 跳過

MODE_SETUP 設定 ISP HARDWARE BUFFER

回傳值

無

3.10 isp_set_flip

目的

設定影像左右翻轉,以及上下翻轉

概要

```
void isp_set_flip(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue 翻轉設定值, 範圍為 0~3, 各值說明如下

0: 原始輸出影像

1: 左右翻轉

2: 上下翻轉

3: 左右且上下翻轉

回傳值

無

3.11 isp_get_flip

目的

取得影像翻轉的設定值

概要

```
void isp_get_flip(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue 取回翻轉的設定值, 範圍為 0~3, 各值說明如下

0: 原始輸出影像

1: 左右翻轉

2: 上下翻轉

3: 左右且上下翻轉

回傳值

無

3.12 isp_set_brightness

目的

設定影像亮度

概要

```
void isp_set_brightness(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue 影像的亮度值, 範圍為-64~64, 可調整精度為+-1

回傳值

無

3.13 isp_get_brightness

目的

取得影像目前的亮度

概要

```
void isp_get_brightness(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

`int *a_pdValue` 取回目前的亮度值, 範圍為 -64~64

回傳值

無

3.14 `isp_set_contrast`

目的

設定影像對比值

概要

`void isp_set_contrast(int a_dValue);`

使用: 在 `video_subsys_init()`成功之後

參數

`int a_dValue` 影像的對比值, 範圍為 0~100, 可調整精度為+-1

回傳值

無

3.15 `isp_get_contrast`

目的

取得影像目前對比值

概要

`void isp_get_contrast(int *a_pdValue);`

使用: 在 `video_subsys_init()`成功之後

參數

`int *a_pdValue` 取回目前的對比值, 範圍為 0~100

回傳值

無

3.16 `isp_set_saturation`

目的

設定影像飽和度

概要

`void isp_set_saturation(int a_dValue);`

使用: 在 `video_subsys_init()`成功之後

參數

`int a_dValue` 影像的飽和度, 範圍為 0~100, 可調整精度為+-1

回傳值

無

3.17 isp_get_saturation

目的

取得影像目前飽和度

概要

```
void isp_get_saturation(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue 取回目前的飽和度, 範圍為 0~100

回傳值

無

3.18 isp_set_sharpness

目的

設定影像銳利度

概要

```
void isp_set_sharpness(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue 影像的銳利度, 範圍為 0~100, 可調整精度為+-1

回傳值

無

3.19 isp_get_sharpness

目的

取得影像目前銳利度

概要

```
void isp_get_sharpness(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue 取回目前的銳利值, 範圍為 0~100

回傳值

無

3.20 isp_set_gamma

目的

設定 Gamma 係數

概要

`void isp_set_gamma(int a_dValue);`

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 影像的 Gamma 係數, 範圍為 100~500, 可調整精度為+-1

回傳值

無

3.21 isp_get_gamma

目的

取得影像目前 Gamma 係數

概要

`void isp_get_gamma(int *a_pdValue);`

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前的銳利值, 範圍為 100~500

回傳值

無

3.22 isp_set_gray_mode

目的

設定影像灰階/彩色模式

概要

`void isp_set_gray_mode(int a_dValue);`

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 影像的灰階模式設定值, 0: 彩色模式, 1: 灰階模式

回傳值

無

3.23 isp_get_gray_mode

目的

取得影像灰階/彩色模式

概要

```
void isp_get_gray_mode(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前灰階模式設定值, 0: 彩色模式, 1: 灰階模式

回傳值

無

3.24 isp_set_exposure_mode

目的

設定自動/手動曝光

概要

```
void isp_set_exposure_mode(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue:

曝光模式, 數值為 1 或 8, (1: 手動曝光, 8: 自動曝光)

回傳值

無

3.25 isp_get_exposure_mode

目的

取得曝光模式

概要

```
void isp_get_exposure_mode(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue:

取回目前的曝光模式, 數值為 1 或 8, (1: 手動曝光, 8: 自動曝光)

回傳值

無

3.26 isp_set_exposure_time

目的

在手動曝光的模式下設定曝光時間

概要

```
void isp_set_exposure_time(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 曝光時間, 單位 us, 範圍為 1~1,000,000, 可調整精度為+-1

回傳值

無

3.27 isp_get_exposure_time

目的

取得目前曝光時間

概要

```
void isp_get_exposure_time(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue:

取回目前的曝光時間, 單位 us, 範圍為 1~1,000,000

回傳值

無

3.28 isp_set_zoom

目的

設定影像放大係數

概要

```
void isp_set_zoom(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 影像的放大係數

範圍為 0~3 (0: 1.0x, 1: 1.28X, 2: 1.6X, 3: 2.0X), 可調整精度為+-1

註: 由於使用上寬度限制在 64~640 之間 (不包括 640)

因此 640x480 以上不支援

回傳值

無

3.29 isp_get_zoom

目的

取得影像放大係數

概要

`void isp_get_zoom(int *a_pdValue);`

使用: 在 `video_subsys_init()`成功之後

參數

`int *a_pdValue`: 取回目前的放大係數, 範圍為 0~3

回傳值

無

3.30 isp_set_pan_tilt

目的

設定畫面的橫向(pan), 縱向(tilt)的平移距離

概要

`void isp_set_pan_tilt(int a_dValuePan, int a_dValueTilt);`

使用: 在 `video_subsys_init()`成功之後

參數

`int a_dValuePan`:

橫向(pan)的平移距離, 範圍為-576000~57600, 可調整精度為+-3600

`int a_dValueTilt`: (Tilt 目前不支援)

註: 640x480 以下支援. (包括 640x480)

回傳值

無

3.31 isp_get_pan_tilt

目的

取得畫面目前的橫向(pan), 縱向(tilt)平移距離

概要

```
void isp_get_pan_tilt(int *a_pdValuePan, int *a_pdValueTilt);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValuePan:

橫向(pan)的平移距離, 範圍為 -576000~57600

int *a_pdValueTilt: (Tilt 目前不支援)

回傳值

無

3.32 isp_set_AWB_ctrl**目的**

設定白平衡模式

概要

```
void isp_set_AWB_ctrl(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue:

白平衡模式, 可設定為 0, 1

0: 手動色溫, 1: 自動

註: 目前尚未有支援手動色溫的 API

回傳值

無

3.33 isp_get_AWB_ctrl**目的**

取得目前白平衡模式

概要

```
void isp_get_AWB_ctrl(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前白平衡設定, 範圍為 0, 1

回傳值

無

3.34 isp_set_power_line_freq

目的

設定 Anti-flicker 模式

概要

```
void isp_set_power_line_freq(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: Anti-flicker 模式, 可設定為 0, 1, 2, 3

0: 關閉, 1: 50Hz, 2: 60Hz, 3: Auto

回傳值

無

3.35 isp_get_power_line_freq

目的

取得目前 Anti-flicker 模式

概要

```
void isp_get_power_line_freq(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前的 Anti-flicker 模式, 範圍為 0, 1, 2, 3

回傳值

無

3.36 isp_set_AE_gain

目的

設定曝光增益值

概要

```
void isp_set_AE_gain(int a_dValueAnalogGain, int a_dValueDigitalGain, int  
a_dValueSPDigitalGain);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValueAnalogGain: 範圍為 256~4080, 可調整精度為+-16

int a_dValueDigitalGain: 預設為 256, 目前不開放修改

int a_dValueISPDigitalGain:範圍為 0~4095, 可調整精度為+-1

註: AnalogGain 是 Sensor 內部 Gain 值; ISPDigitalGain 是 ISP 內部分的 Gain 值.

回傳值

無

3.37 isp_get_AE_gain

目的

取得目前曝光增益值

概要

```
void isp_get_AE_gain(int *a_pdValueAnalogGain, int *a_pdValueDigitalGain,
int *a_pdValueISPDigitalGain);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValueAnalogGain: 範圍為 256~4080

int a_dValueDigitalGain: 預設為 256

int a_dValueISPDigitalGain:範圍為 0~4095

回傳值

無

3.38 isp_set_WDR_mode

目的

設定寬動態模式

概要

```
void isp_set_WDR_mode(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 寬動態模式, 可設定為 0, 1, 2, 3, 4

0: 關閉, 1: 手動, 2: 自動(弱), 3: 自動(中), 4: 自動(強)

回傳值

無

3.39 isp_get_WDR_mode

目的

取得寬動態模式

概要

```
void isp_get_WDR_mode(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前的寬動態模式, 範圍為 0, 1, 2, 3, 4

回傳值

無

3.40 isp_set_WDR_level

目的

在寬動態的手動模式下設定強度

概要

```
void isp_set_WDR_level(int a_dValue);
```

使用: 在 video_subsys_init()成功之後

參數

int a_dValue: 寬動態強度, 範圍為 0~100, 可調整精度為+-1

回傳值

無

3.41 isp_get_WDR_level

目的

取得目前寬動態強度

概要

```
void isp_get_WDR_level(int *a_pdValue);
```

使用: 在 video_subsys_init()成功之後

參數

int *a_pdValue: 取回目前的寬動態強度, 範圍為 0~100

回傳值

無

3.42 isp_stream_power_off_config

目的

設定 SENSOR POWER OFF 模式

概要

```
int isp_stream_power_off_config(unsigned char enable);
```

參數

unsigned char enable :

0:ISP 串流都關閉後,不會關閉 SENSOR,節省切換 ISP 時間,注意在最後進入省電模式,需要需設定為 1 再進行關閉 ISP 串流動作

1:ISP 串流都關閉後,會關閉 SENSOR,此為預設值

回傳值

0:成功 -1:失敗,ISP 沒初始化

3.43 isp_check_boot_status**目的**

取得 isp boot mode 狀態

概要

```
int isp_check_boot_status(void);
```

參數

無

回傳值

0: normal boot 1: fast boot

4 OSD

4.1 OSD introduction

rtstream 提供了一組 API 函數來設置資料流程 stream 的 OSD 配置等，注意調用這組 API 時需要在調用之前先創建資料流程 stream。該組介面如下：

rts_video_query_osd_attr 介面獲取 video osd 屬性；

rts_video_set_osd_attr 介面設置 video osd 屬性；

rts_video_release_osd_attr 介面釋放掉 rts_video_query_osd_attr 獲取到的屬性；

另外提供了一組設置視頻 OSD 的簡易介面，這組介面不需要使用者額外創建相應的資料流程 stream。該組介面如下：

rts_query_isp_osd_attr 介面獲取 video osd 屬性；

rts_set_isp_osd_attr 介面設置 video osd 屬性；

rts_release_isp_osd_attr 介面釋放掉 rts_query_isp_osd_attr 獲取到的屬性；

4.2 OSD example

範例程式位於： component\common\example\isp\example_isp_osd_multi.c

使用前必須設定 platform_opts.h

開啟位於 project\realtek_amebapro_v0_example\inc\platform_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
0
```

修改 CONFIG_EXAMPLE_MEDIA_UVCD 由 0 改為 1，

修改 CONFIG_EXAMPLE_ISP_OSD_MULTI 由 0 改為 1，編譯後燒錄

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
1
```

執行與測試

3. 將 USB 線接到 AmebaPro CON port 上，另一端接上 PC
4. 透過 potplayer 觀看影像，使用 amebaPro atcmd 鍵入"ATIO"，觀看結果

4.3 OSD Show Time information

OSD 顯示的時間是依據 SNTP，透過"sntp_gen_system_time"函數取得時間，因此需要藉由全域變數 rtsTimezone 設定 timezone

```
extern int rtsTimezone;  
rtsTimezone = 8;
```

4.4 OSD API

4.4.1 rts_video_query_osd_attr

目的

獲取視頻 stream 的 osd 屬性。

概要

```
int rts_video_query_osd_attr(RtStream stream, struct rts_video_osd_attr  
**attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸出參數，指向存放 osd attr 的變數的位址，需調用
rts_video_release_osd_attr 釋放。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

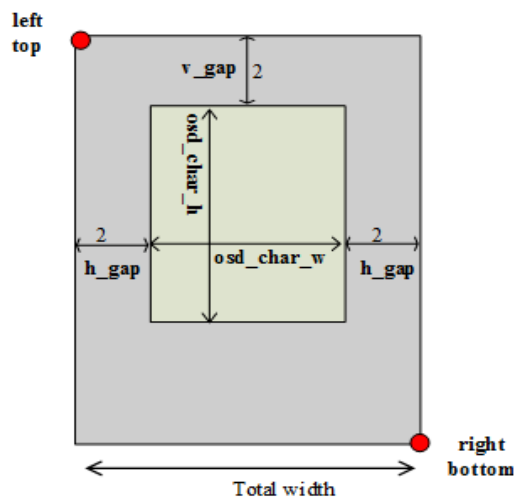
描述

每一路視頻stream 都有一個單獨的osd 模組，每個osd 模組由結構 rts_video_osd_attr 表示。每個osd 模組最多支援6 個block，block 是圖像中用於顯示字元或圖像的一個區域，由結構體rts_video_osd_block 表示。英文和數位與中文顯示時所占字寬不一致，英文和陣列採用單倍寬，字形檔保存在單倍字形檔中，中文顯示佔用雙倍寬，字形檔保存在雙倍寬字形檔中。rts_video_osd_attr 中的single_lib_name，double_lib_name 分別用來保存各個字形檔的檔案名。osd 中的圖片由 block 中的pbitmap 表示，是一個指向BITMAP_S 結構體的指標。

```
struct rts_video_osd_attr {  
    int number; //osd 的block 數量  
    struct rts_video_osd_block *blocks;  
    enum rts_osd_time_fmt time_fmt; //顯示的時間格式  
    uint8_t time_blkidx; //時間顯示的block 的index  
    int time_pos; //時間顯示的位置  
    enum rts_osd_date_fmt date_fmt; //顯示的日期格式  
    uint8_t date_blkidx; //日期顯示的block 的index  
    int date_pos; //日期顯示的位置  
    char *single_lib_name; //單倍字形檔檔案名  
    char *double_lib_name; //雙倍字形檔檔案名  
    uint8_t *font_color; //font 字三原色
```

```

struct rts_video_osd_block {
    struct rts_video_rect rect; //block 座標
    uint8_t bg_enable; //背景使能
    uint32_t bg_color; //背景顏色
    uint32_t ch_color; //字元顏色
    uint8_t h_gap:4, v_gap:4; //字元與字元之間的間隔
    uint8_t flick_enable; //字元閃爍使能
    uint32_t flick_speed; //閃爍速度，每隔 2^flick_speed 閃爍一次
    uint8_t char_color_alpha; //字元半透明
    uint8_t stroke_enable; //字元描邊的開關
    uint8_t stroke_direct; //字元描邊的方向，0: 減去增量，1: 加上增量
    uint8_t stroke_delta; //字元描邊增量
}
    
```



osd 結構體中的欄位意義如圖所示，OSD 字元與字元之間存在間隔的值是由使用者配置的，最小值是2，最大值是15. 字元水準方向間隔為h_gap，垂直方向間隔為v_gap.

rts osd time fmt	顯示樣式	例子
osd_time_fmt_no	不顯示時間	不顯示時間
osd_time_fmt_24	hh:mm:ss	14:32:58
osd_time_fmt_12	hh:mm:ss	02:32:58
osd_time_fmt_12_1	Phh:mm:ss	P02:32:58
osd_time_fmt_12_2	PMhh:mm:ss	PM02:32:58
osd_time_fmt_12_3	PM~hh:mm:ss	PM~02:32:58
osd_time_fmt_12_4	hh:mm:ssPM	02:32:58PM
osd_time_fmt_12_5	hh:mm:ss~PM	02:32:58~PM
osd_time_fmt_12_6	hh:mm:ss~~PM	02:32:58~~PM
osd_time_fmt_12_7	hh:mm:ss~~~PM	02:32:58~~~PM

rts osd date fmt	樣式	例子
osd_date_fmt_no	不顯示日期	不顯示日期
osd_date_fmt_0	dd/MM/yyyy	26/05/2015
osd_date_fmt_1	dd/MM/yy	26/05/15
osd_date_fmt_2	d/M/yy	26/5/15
osd_date_fmt_3	M/d/yyyy	5/26/2015
osd_date_fmt_4	M/d/yy	5/26/15
osd_date_fmt_5	MM/dd/yy	05/26/15
osd_date_fmt_6	MM/dd/yyyy	05/26/2015
osd_date_fmt_7	yyyy/M/d	2015/5/26
osd_date_fmt_8	yyyy-M-d	2015-5-26
osd_date_fmt_9	yyyy-MM-dd	2015-05-26
osd_date_fmt_10	yyyy/MM/dd	2015/05/26
osd_date_fmt_11	yy-MM-dd	15-05-26
osd_date_fmt_12	yy/M/d	15/5/26
osd_date_fmt_13	yy-M-d	15-5-26
osd_date_fmt_14	yy/MM/dd	15/05/26

4.4.2 rts_video_set_osd_attr

目的

設置視頻 stream 的 osd 屬性。

概要

```
int rts_video_set_osd_attr(RtStream stream, struct rts_video_osd_attr
*attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向osd 屬性的指標，由rts_video_query_osd_attr 獲得。結構體rts_video_osd_attr定義參見[rts_video_query_osd_attr](#)

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

無

4.4.3 rts_video_release_osd_attr

目的

釋放視頻 stream 的 osd 屬性。

概要

```
void rts_video_release_osd_attr(RtStream stream, struct rts_video_osd_attr *attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向osd 屬性的指標，由rts_video_query_osd_attr 獲得。結構體rts_video_osd_attr定義參見[rts_video_query_osd_attr](#)

回傳值

無

描述

該函數用來釋放rts_video_query_osd_attr 獲取到的osd attr，否則會產生記憶體洩露。

4.4.4 rts_query_isp_osd_attr

目的

獲取視頻 osd 屬性。

概要

```
int rts_query_isp_osd_attr(int isp_id, struct rts_video_osd_attr **attr);
```

參數

isp_id

attr 輸入參數，isp 支持同時輸出多路，每一路可以創建一個isp stream，此處的id 即某一路isp 的index，從0 開始。

輸出參數，指向存放osd attr 的變數的位址，需調用
`rts_release_isp_osd_attr` 釋放。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

每一路視頻stream 都有一個單獨的osd 模組，每個osd 模組由結構體 `rts_video_osd_attr` 表示。每個osd 模組最多支援6 個block，block 是圖像中用於顯示字元或圖像的一個區域，由結構體 `rts_video_osd_block` 表示。英文和數位與中文顯示時所占字寬不一致，英文和陣列採用單倍寬，字形檔保存在單倍字形檔中，中文顯示佔用雙倍寬，字形檔保存在雙倍寬字形檔中，如果要顯示logo 或二維碼等圖像資訊，可以把圖像保存在圖像字形檔中。`rts_video_osd_attr` 中的 `single_lib_name`，`double_lib_name` 和 `picture_lib_name` 分別用來保存各個字形檔的檔案名。

4.4.5 `rts_set_isp_osd_attr`

目的

設置視頻 osd 屬性。

概要

```
int rts_set_isp_osd_attr(struct rts_video_osd_attr *attr);
```

參數

Attr

輸入參數，指向osd 屬性的指標，由 `rts_query_isp_osd_attr` 獲得。結構體 `rts_video_osd_attr` 定義參見 `rts_video_query_osd_attr`

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

無

4.4.6 rts_release_isp_osd_attr

目的

釋放視頻 osd 屬性。

概要

```
void rts_release_isp_osd_attr(struct rts_video_osd_attr *attr);
```

參數

Attr

輸入參數，指向osd 屬性的指標，由rts_query_isp_osd_attr 獲得。結構體rts_video_osd_attr 定義參見*rts_video_query_osd_attr*

回傳值

無

描述

該函數用來釋放 rts_query_isp_osd_attr 獲取到的 osd attr，否則會產生記憶體洩露。

5 Motion Detect

5.1 Motion Detect introduction

rtstream 提供了一組 API 函數來設置資料流程 stream 的運動檢測的配置，注意調用這組 API 時需要在調用之前先創建資料流程 stream。

該組介面如下：

rts_video_query_md_attr 介面獲取 isp 支援的 motion detect 屬性；

rts_video_set_md_attr 介面設置更新 motion detect；

rts_video_release_md_attr 介面釋放掉 rts_video_query_md_attr 獲取到的屬性；

ts_video_check_md_status 介面檢查是否檢測到 motion detect。

另外提供了一組設置運動檢測的簡易介面，這組介面不需要使用者額外創建相應的資料流程 stream。

該組介面如下：

rts_query_isp_md_attr 介面獲取 isp 支援的 motion detect 屬性；

rts_set_isp_md_attr 介面設置更新 motion detect；

rts_release_isp_md_attr 介面釋放掉 rts_query_isp_md_attr 獲取到的屬性；

rts_check_isp_md_status 介面檢查是否檢測到 motion detect。

5.2 Motion Detect example

範例程式位於： component\common\example\isp\example_md.c

使用前必須設定 platform_opts.h

開啟位於 project\realtek_amebapro_v0_example\inc\platform_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_UVCD
0
```

修改 CONFIG_EXAMPLE_MEDIA_UVCD 由 0 改為 1，

修改 CONFIG_EXAMPLE_MOTION_DETECT 由 0 改為 1，編譯後燒錄

```
#define CONFIG_EXAMPLE_MEDIA_UVCD
1
```

執行與測試

- 將 USB 線接到 AmebaPro CON port 上，另一端接上 PC
- 透過 Tera Term 觀看 Log，使用 amebaPro atcmd 鍵入“ATID”，觀看結果

5.3 Motion Detect API

5.3.1 rts_video_query_md_attr

目的

獲取視頻 stream 的 motion detect 屬性。

概要

```
int rts_video_query_md_attr(RtStream stream, struct rts_video_md_attr
**attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸出參數，指向存放motion detect attr 的變數的位址，需調用
rts_video_release_md_attr 釋放。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

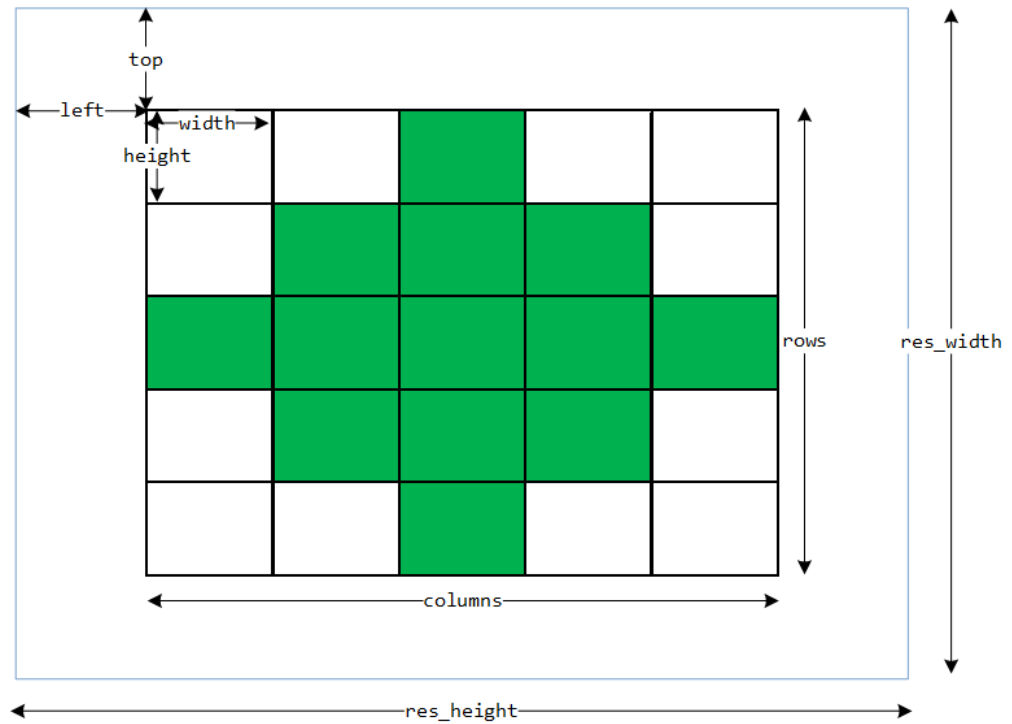
```
struct rts_video_md_attr {  
    int number; //motion detect 的 block 數量  
    struct rts_video_md_block *blocks; //指向 blocks 的指標  
    uint32_t reserved[4];  
};  
  
struct rts_video_md_block {  
    int enable; //使能開關  
    struct rts_video_grid area;  
    uint32_t sensitivity; //敏感度,0~100
```

做motion detect 分析時，既可以逐幀分析，也可以隔幀分析，相隔的幀數可以通過frame_interval 配置。如果間隔幀數較小，md 比較容易檢測到高速運動，而不容易檢測到低速運動；而如果間隔幀數較大，可以累計更多的差異，這樣更容易檢測到緩慢的運動。根據不同的應用場景，可以通過配置sensitivity 和percentage來改變檢測到運動的閾值。sensitivity 越大，越敏感，閾值越低，更容易檢測到運動，percentage 越小，閾值越低，越容易檢測到運動。

```
struct rts_video_grid_unit {
uint32_t width;
uint32_t height;
};
struct rts_video_grid {
int32_t left;
int32_t top;
struct rts_video_grid_unit cell;
uint32_t rows;
uint32_t columns;
int length;
uint8_t bitmap[(RTS_ISP_GRID_MAX_NUM + 7) / 8];
};
```

rts_video_grid 結構體中的變數如圖 rts_video_grid 所示。bitmap 的每個 bit 表示 grid 的一個儲存格，0 表示disable, 1 表示enable。圖中

rows = 5
columns = 5
length = columns * rows = 5 * 5 = 25
bitmap = {0b00100011, 0b10111110, 0b11100010, 0b00000000};



5.3.2 rts_video_set_md_attr

目的

設置視頻 stream 的 motion detect 屬性。

概要

```
int rts_video_set_md_attr(RtStream stream, struct rts_video_md_attr *attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向md attr 的指標，由rts_video_query_md_attr獲得，需調用rts_video_release_md_attr釋放。結構體rts_video_md_attr定義參見rts_video_md_attr。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

無

5.3.3 rts_video_release_md_attr

目的

釋放視頻 stream 的 motion detect 屬性。

概要

```
void rts_video_release_md_attr(RtStream stream, struct rts_video_md_attr *attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向md attr 的指標，由rts_video_query_md_attr 獲得。結構體rts_video_md_attr 定義參見rts_video_md_attr

回傳值

無

描述

該函數用來釋放rts_video_query_md_attr 獲取到的md attr，否則會產生記憶體洩露。

5.3.4 rts_video_check_md_status

目的

檢查視頻 stream 的 motion detect 狀態，查看是否檢測到運動。

概要

```
int rts_video_check_md_status(RtStream stream, int mdidx);
```

參數

stream

輸入參數，RtStream 指針。

mdidx

輸入參數，motion detect block 的index

回傳值

返回 1 表示檢測到運動，返回 0 表示沒有檢測到。

描述

無

5.3.5 rts_video_get_md_result

目的

獲得運動檢測的點陣圖。

概要

```
int rts_video_get_md_result(RtStream stream, int mdidx, struct  
rts_video_grid_bitmap *result);
```

參數

stream

輸入參數，RtStream 指針。

mdidx

輸入參數，motion detect block 的index，目前RTS3901&RTS3902 支持的MD block 數為1，所以對於RTS3901&RTS3902 請固定設置為0。

result

輸出參數，指向rts_video_grid_bitmap 的指標，該結構中包含了MD grid 點陣圖的資訊。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

```
struct rts_video_grid_bitmap {  
    uint16_t number; //網格的數量  
    uint8_t  
    bitmap[RTS_GRID_BITMAP_SIZE]; //所有網格的點陣圖
```

5.3.6 rts_query_isp_md_attr

目的

獲取視頻 stream 的 motion detect 屬性。

概要

```
int rts_query_isp_md_attr(struct rts_video_md_attr **attr, uint32_t  
res_width, uint32_t res_height);
```

參數

attr 輸出參數，指向md attr 的指標，需調用rts_video_release_md_attr 釋放。結構體rts_video_md_attr定義參見rts_video_md_attr。

res_width
h

輸入參數，解析度寬度，rts_video_md_attr 中的位置是相對於該解析度的。

res_height
ht

輸入參數，解析度高度，rts_video_md_attr 中的位置是相對於該解析度的。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

與rts_video_query_md_attr 的區別是，rts_query_isp_md_attr 與stream 無關，不需要提供Rt-Stream 指標參數。

5.3.7 rts_set_isp_md_attr

目的

設置視頻 stream 的 motion detect 屬性。

概要

```
int rts_set_isp_md_attr(struct rts_video_md_attr **attr);
```

參數

attr 輸入參數，指向md attr 的指標，由rts_query_isp_md_attr 獲得，需調用rts_release_isp_md_attr釋放。結構體rts_video_md_attr 定義參見rts_video_md_attr。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

與rts_video_set_md_attr 的區別是，rts_set_isp_md_attr 與stream 無關，不需要提供RtStream指標參數。

5.3.8 rts_check_isp_md_status

目的

檢查視頻 stream 的 motion detect 狀態，查看是否檢測到運動。

概要

```
int rts_check_isp_md_status(int mdidx);
```

參數

mdidx 輸入參數，motion detect block 的index，目前RTS3901&RTS3902 支持的 MD block 數為1，所以對於RTS3901&RTS3902 請固定設置為0。

回傳值

返回 1 表示檢測到運動，返回 0 表示沒有檢測到。

描述

與rts_video_check_md_status 的區別是，rts_check_isp_md_status 與 stream 無關，不需要提供RtStream 指標參數。

5.3.9 rts_get_isp_md_result

目的

獲得運動檢測的點陣圖。

概要

```
int rts_get_isp_md_result(int mdidx, struct rts_video_grid_bitmap *result);
```

參數

mdidx 輸入參數，motion detect block 的index，目前RTS3901&RTS3902 支持的 MD block 數為1，所以對於RTS3901&RTS3902 請固定設置為0。

result 輸出參數，指向rts_video_grid_bitmap 的指標，該結構中包含了MD grid 點陣圖的資訊。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

與rts_video_check_md_status 的區別是，rts_check_isp_md_status 與 stream 無關，不需要提供RtStream 指標參數。

THIS SOFTWARE AND DOCUMENT ARE PROVIDED "AS IS" WITHOUT ANY WARRANTIES OF ANY KIND. REALTEK MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THIS THE SOFTWARE AND DOCUMENT AT ANY TIME AND AT ITS SOLE DISCRETION. WITH RESPECT TO THE SOFTWARE; DOCUMENT; INFORMATION; MATERIALS; SERVICES; AND ANY IMPROVEMENTS AND/OR CHANGES THERETO PROVIDED BY REALTEK, REALTEK DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

6 Mask

6.1 Mask introduction

rtstream 提供了一組 API 函數來設置資料流程 stream 的視頻遮罩配置，注意調用這組 API 時需要在調用之前先創建資料流程 stream。該組介面如下：

rts_video_query_mask_attr 介面獲取 video mask 屬性；

rts_video_set_mask_attr 介面設置 video mask 屬性；

rts_video_release_mask_attr 介面釋放掉 rts_video_query_mask_attr 獲取到的屬性；另外提供了一組設置視頻遮罩的簡易介面，這組介面不需要使用者額外創建相應的資料流程 stream。該組介面如下：

rts_query_isp_mask_attr 介面獲取 video mask 屬性；

rts_set_isp_mask_attr 介面設置 video mask 屬性；

rts_release_isp_mask_attr 介面釋放掉 rts_video_query_mask_attr 獲取到的屬性；

6.2 Mask example

範例程式位於： component\common\example\isp\example_mask.c

使用前必須設定 platform_opts.h

開啟位於 project\realtek_amebapro_v0_example\inc\platform_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
0
```

修改 CONFIG_EXAMPLE_MEDIA_UVCD 由 0 改為 1 ,

修改 CONFIG_EXAMPLE_MASK 由 0 改為 1 , 編譯後燒錄

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
1
```

執行與測試

- 將 USB 線接到 AmebaPro CON port 上 , 另一端接上 PC
- 透過 Tera Term 觀看 Log , 使用 amebaPro atcmd 鍵入"ATIM" , 觀看結果

6.3 Mask API

6.3.1 rts_video_query_mask_attr

目的

獲取視頻 stream 的 private mask 屬性。

概要

```
int rts_video_query_mask_attr(RtStream stream, struct rts_video_mask_attr  
**attr);
```

參數

stream

輸入參數 , RtStream 指針。

attr

輸出參數 , 指向存放private mask attr 的變數的位址 , 需調用
rts_video_release_mask_attr 釋放。

回傳值

返回 0 表示成功 , 返回負的錯誤碼表示失敗。

描述

```
struct rts_video_mask_attr {
    uint32_t color; /*rgb24*/
    int number; //private mask 的 block 數量
    struct rts_video_mask_block *blocks;
    uint32_t reserved[4];
};

struct rts_video_mask_block {
    int type;
    int enable;
    union {
        struct rts_video_grid area;
        struct rts_video_rect rect;
    };
    uint32_t res_width;
    uint32_t res_height;
};

struct rts_video_rect {
```

結構體 `rts_video_grid` 參見 `rts_video_query_md_attr` 處 `rts_video_grid` 的定義。

`rtstream` 一共支援5個mask區域，其中1個grid，4個rect。

示例

```
RtStream stream;

/*1. create stream, refer to rts_create_h264_stream */

/*2. get mask attr */
ret = rts_video_query_mask_attr(stream, &attr);

/*3. release md attr */
rts_video_release_mask_attr(stream, attr);
```

6.3.2 rts_video_set_mask_attr

目的

設置視頻 stream 的 private mask 屬性。

概要

```
int rts_video_set_mask_attr(RtStream stream, struct rts_video_mask_attr
*attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向mask attr 的指標，由rts_video_query_mask_attr 獲得，需調用rts_video_release_mask_attr 釋放。結構體 rts_video_mask_attr 定義參見rts_video_mask_attr。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

無

6.3.3 rts_video_release_mask_attr

目的

釋放視頻 stream 的 private mask 屬性。

概要

```
void rts_video_release_mask_attr(RtStream stream, struct  
rts_video_mask_attr *attr);
```

參數

stream

輸入參數，RtStream 指針。

attr

輸入參數，指向mask attr 的指標，由rts_video_query_mask_attr 獲得。

結構體rts_video_mask_attr

定義參見rts_video_mask_attr。

回傳值

無

描述

該函數用來釋放rts_video_query_mask_attr 獲取到的mask attr，否則會產生記憶體洩露。

6.3.4 rts_query_isp_mask_attr

目的

獲取視頻 stream 的 private mask 屬性。

概要

```
int rts_query_isp_mask_attr(struct rts_video_mask_attr **attr, uint32_t  
res_width, uint32_t res_height);
```

參數

attr

輸出參數，指向存放 private mask attr 的變數的位址，需調用 rts_release_isp_mask_attr 釋放。

res_width *h*

輸入參數，解析度寬度，rts_video_mask_attr 中的位置是相對於該解

析度的。

res_heig
ht

輸入參數，解析度高度，`rts_video_mask_attr` 中的位置是相對於該解析度的。

回傳值

返回 **1** 表示檢測到運動，返回 **0** 表示沒有檢測到。

描述

結構體 `rts_video_mask_attr` 參見 `rts_video_query_mask_attr`

示例

```
1 struct rts_video_md_attr *attr = NULL;
2
3 /*1. init rtstream context */
4 rts_av_init();
5
6 /*2. get mask attribute, */
7 int ret = rts_query_isp_mask_attr(&attr, 1280, 720);
8 if (ret) {
9 rts_av_release();
10 return ret;
11 }
12
13 /*3. release mask attribute*/
```

6.3.5 `rts_set_isp_mask_attr`

目的

設置視頻 stream 的 private mask 屬性。

概要

```
int rts_set_ismask_attr(struct rts_video_mask_attr *attr, uint32_t  
res_width, uint32_t res_height)
```

參數

res_width

輸入參數，解析度寬度，*rts_video_mask_attr* 中的位置是相對於該解析度的。

res_height

輸入參數，解析度高度，*rts_video_mask_attr* 中的位置是相對於該解析度的。

attr

輸入參數，指向mask attr 的指標，由*rts_query_ismask_attr* 獲得，需調用*rts_release_ismask_attr* 釋放。結構體*rts_video_mask_attr* 定義參見*rts_video_mask_attr*。

回傳值

返回 0 表示成功，返回負的錯誤碼表示失敗。

描述

結構體*rts_video_mask_attr* 參見*rts_video_query_mask_attr*

6.3.6 *rts_release_ismask_attr*

目的

釋放視頻 stream 的 private mask 屬性。

概要

```
void rts_release_ismask_attr(struct rts_video_mask_attr *attr);
```

參數

attr

輸入參數，指向mask attr 的指標，由*rts_query_ismask_attr* 獲得。結構體*rts_video_mask_attr* 定義參見*rts_video_mask_attr*。

回傳值

描述

該函數用來釋放*rts_query_ismask_attr* 獲取到的mask attr，否則會產

生記憶體洩露。