



**REALTEK**

**AN0301**

**Multimedia Framework v2**

### **Abstract**

Purpose of AN0301 document is to describes Multimedia Framework of AmebaPro SDK. This framework integrates multiple units such as Video, Audio, Network, Storage, and media streams which are passed or stored between these units.

## Table of Contents

<b>1</b>	<b>Multimedia Framework Architecture .....</b>	<b>6</b>
1.1	Architecture.....	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 .....	16
1.2.1	ISP.....	16
1.2.2	H264.....	19
1.2.3	RTSP .....	22
1.2.4	Video frame control .....	23
1.2.5	JPEG.....	25
1.2.6	AAC Encoder.....	25
1.2.7	AAC Decoder .....	26
1.2.8	Audio Codec .....	26
1.2.9	RTP Input.....	26
1.2.10	G711 Codec .....	27
1.2.11	MP4 .....	27
1.2.10	I2S.....	29
1.2.11	httpfs .....	29
1.3	Using the MMF v2 example .....	30
1.3.1	Sample Program.....	30
1.4	Selecting and setting up the sample program.....	30
1.4.1	Choose the proper sample program .....	31
1.4.2	Adjusting the Video/Audio Parameters of MMFv2 Example .....	44
1.4.3	Adjusting LWIP Parameters.....	45
1.4.4	Echo Cancellation .....	46
1.4.5	Capture the first frame and related notes .....	47
1.4.6	VLC media player settings .....	48
<b>2</b>	<b>Video API.....</b>	<b>50</b>
2.1	H264 API.....	50
2.1.1	h264_open .....	50
2.1.2	h264_initial .....	50
2.1.3	h264_encode .....	51
2.1.4	h264_release.....	51
2.2	JPEG API .....	51
2.2.1	jpeg_open .....	51
2.2.2	jpeg_initial.....	52
2.2.3	jpeg_encode.....	52
2.2.4	jpeg_release .....	52

<b>3</b>	<b>ISP API .....</b>	<b>53</b>
3.1	video_subsys_init .....	53
3.2	isp_stream_create .....	53
3.3	isp_stream_destroy .....	53
3.4	isp_stream_set_complete_callback .....	54
3.5	isp_stream_apply .....	54
3.6	isp_stream_start.....	54
3.7	isp_stream_stop .....	54
3.8	isp_stream_poll .....	55
3.9	isp_handle_buffer.....	55
3.10	isp_set_flip .....	55
3.11	isp_get_flip.....	56
3.12	isp_set_brightness .....	56
3.13	isp_get_brightness.....	56
3.14	isp_set_contrast .....	56
3.15	isp_get_contrast .....	57
3.16	isp_set_saturation .....	57
3.17	isp_get_saturation .....	57
3.18	isp_set_sharpness.....	58
3.19	isp_get_sharpness .....	58
3.20	isp_set_gamma.....	58
3.21	isp_get_gamma .....	58
3.22	isp_set_gray_mode.....	59
3.23	isp_get_gray_mode .....	59
3.24	isp_set_exposure_mode .....	59
3.25	isp_get_exposure_mode .....	60
3.26	isp_set_exposure_time .....	60
3.27	isp_get_exposure_time.....	60
3.28	isp_set_zoom.....	60
3.29	isp_get_zoom .....	61

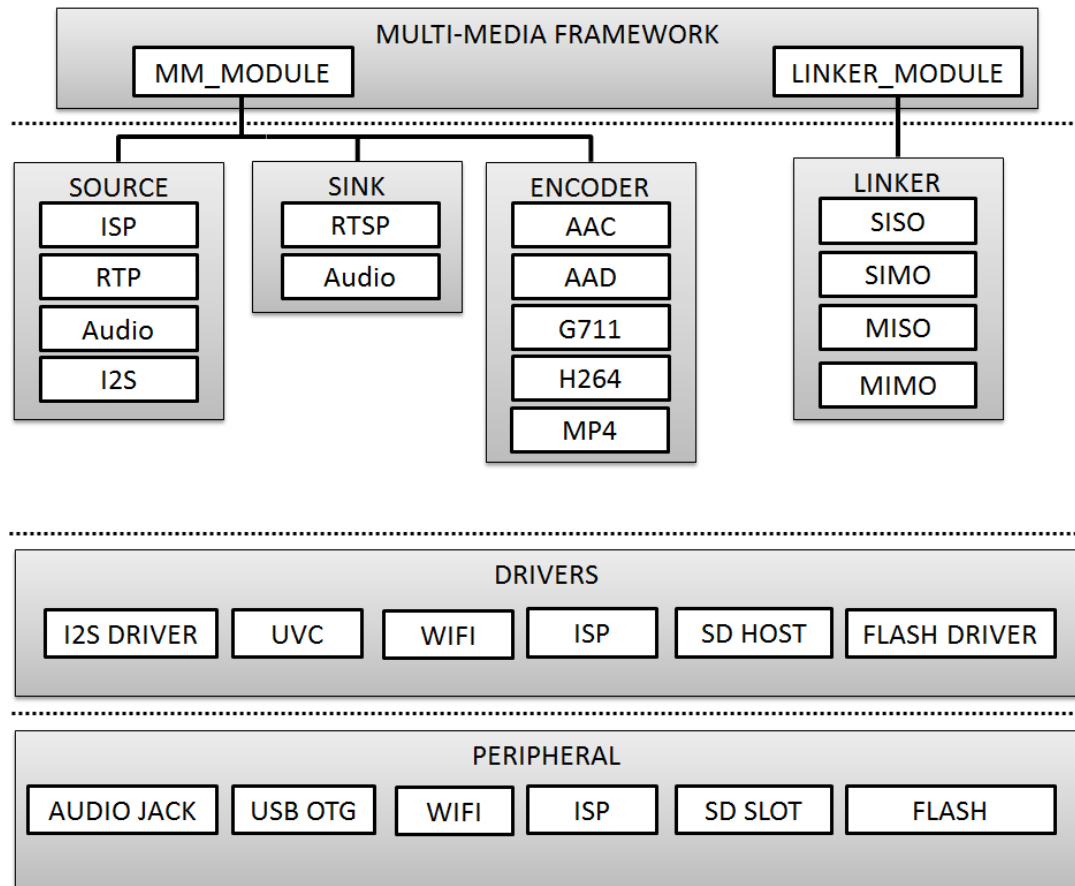
3.30	isp_set_pan_tilt .....	61
3.31	isp_get_pan_tilt.....	61
3.32	isp_set_AWB_ctrl.....	62
3.33	isp_get_AWB_ctrl .....	62
3.34	isp_set_power_line_freq.....	62
3.35	isp_get_power_line_freq .....	62
3.36	isp_set_AE_gain.....	63
3.37	isp_get_AE_gain .....	63
3.38	isp_set_WDR_mode.....	64
3.39	isp_get_WDR_mode .....	64
3.40	isp_set_WDR_level .....	64
3.41	isp_get_WDR_level.....	64
3.42	isp_stream_power_off_config.....	65
3.43	isp_check_boot_status.....	65
<b>4</b>	<b>OSD .....</b>	<b>66</b>
4.1	OSD introduction .....	66
4.2	OSD example .....	66
4.3	OSD Show Time information .....	67
4.4	OSD API .....	67
4.4.1	rts_video_query_osd_attr .....	67
4.4.2	rts_video_set_osd_attr.....	70
4.4.3	rts_video_release_osd_attr .....	70
4.4.4	rts_query_isp_osd_attr.....	71
4.4.5	rts_set_isp_osd_attr .....	72
4.4.6	rts_release_isp_osd_attr .....	72
<b>5</b>	<b>Motion Detect.....</b>	<b>73</b>
5.1	Motion Detect introduction.....	73
5.2	Motion Detect example .....	73
5.3	Motion Detect API .....	74
5.3.1	rts_video_query_md_attr .....	74
5.3.2	rts_video_set_md_attr .....	76
5.3.3	rts_video_release_md_attr .....	77
5.3.4	rts_video_check_md_status .....	77
5.3.5	rts_video_get_md_result.....	77

5.3.6	rts_query_isp_md_attr .....	78
5.3.7	rts_set_isp_md_attr.....	78
5.3.8	rts_check_isp_md_status .....	79
5.3.9	rts_get_isp_md_result .....	79
<b>6</b>	<b>Mask .....</b>	<b>80</b>
<b>6.1</b>	<b>Mask introduction .....</b>	<b>80</b>
<b>6.2</b>	<b>Mask example .....</b>	<b>80</b>
<b>6.3</b>	<b>Mask API .....</b>	<b>81</b>
6.3.1	rts_video_query_mask_attr.....	81
6.3.2	rts_video_set_mask_attr .....	82
6.3.3	rts_video_release_mask_attr .....	82
6.3.4	rts_query_isp_mask_attr .....	82
6.3.5	rts_set_isp_mask_attr .....	84
6.3.6	rts_release_isp_mask_attr.....	85

# 1 Multimedia Framework Architecture

The Multimedia Framework Architecture version 2(MMFv2) is responsible for handling the connection and management of different media resources on AmebaPro.

## 1.1 Architecture



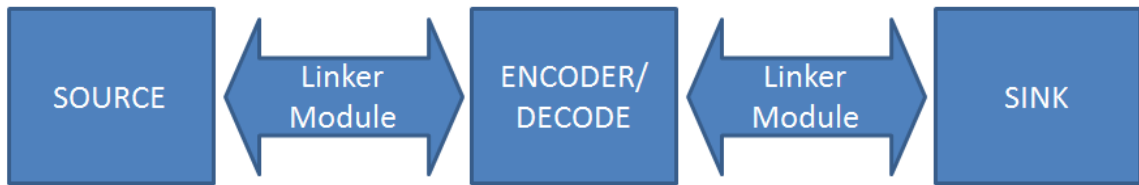
The structure of MMFv2 is as shown in the following chart :

There are two important entities in the MMFv2. One is MM\_MODULE(the source, sink and decode/encode module are one of MM\_MODULE), which source produces resource and the sink consumes the resource produce by source module. The source can be the file input, microphone, camera, and or storage, and the sink can be RTSP or other stream. The other is LINKER\_MODULE which connect different type of module and deal with inter module communication.

In order to use the MMFv2, the following aspects must be followed.

- Define a valid source.
- Define a valid sink.
- Define a valid encoder/decoder if needed.
- Define a valid link module.

The main usage flow is to initialize different MM\_MODULE, and connect different MM\_MODULE through LINKER\_MODULE.



### 1.1.1 Module

MMFv2 allows users to define customized source, sink and encoder/decoder modules depending on the application. Although implementation details may be different, however basic rules of thefor MMF structure should be a little bit similar.

#### MMF module

MMFv2 requires users to predefine both source and sink modules through implement create, destroy, control, handle, new\_item, del\_item and rsz\_item function callbacks. mmf\_module\_t provides the interface for communication between mmf modules. In order to maintain the flexibility and convenience between modules, modules only retain the interface of each type to provide module to access. Function's constant of each module is defined by module itself.

```

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;
}
  
```

#### **Function description**

##### create

Pointer to the function that loads and initializes the module that you wish to add. For example, for ISP source, it points to the function in which the ISP driver is initialized and the corresponding context is returned.

**destroy**

Pointer to the function that de-initializes module instance and releases resource. For example, for ISP source, it points to function in which ISP driver is de-initialized and the corresponding context is released.

**control**

Pointer to function that sends the control command to the MMF module layer or a specific module. For example, for ISP source, it points to function that controls ISP parameters (“frame height”, “frame width”, “framerate”, etc.) and MMFv2 service task on or off.

**handle**

Pointer to the function that manipulates media data (how to produce data in source or how to consume data in sink). Data is transferred from source to sink and vice versa by means of OS message queue. Please note that MMF service task reacts differently based on message exchange buffer status.

**new\_item**

Pointer to the function that create queue item that will be send to input and output queue, only will be used when setting MM\_CMD\_INIT\_QUEUE\_ITEMS to MMQI\_FLAG\_STATIC.

**del\_item**

Pointer to the function that destroy queue item, only will be used when setting MM\_CMD\_INIT\_QUEUE\_ITEMS to MMQI\_FLAG\_STATIC.

**rsz\_item**

Pointer to the function decrease memory pool size, only will be used when h264 and aac module is created.

**output\_type and module\_type**

Output\_type indicates output mode. There are MM\_TYPE\_NONE, MM\_TYPE\_VSRC, MM\_TYPE\_ASRC, MM\_TYPE\_VDSP, MM\_TYPE\_ADSP, MM\_TYPE\_VSINK, MM\_TYPE\_ASINK, and MM\_TYPE\_AVSINK can be used, corresponding to different module usage scenarios, let application know which mode the output is. module\_type represents the identity of the module, and there are three option can be used MM\_MASK\_SRC, MM\_MASK\_DSP and MM\_MASK\_SINK.

**name**

Pointer to the module name.



### 1.1.2 Context

MMFv2 context supply message transfer between different modules. It contains `mm_module_t`, and queue that used to pass data. There are 6 types of status that `mm_context` support(`MM_STAT_INIT`, `MM_STAT_READY`, `MM_STAT_ERROR`, `MM_STAT_ERR_MALLOC`, `MM_STAT_ERR_QUEUE`, `MM_STAT_ERR_NEWITEM`), these status are responsible for maintaining the module state to ensure the program runs smoothly.

```
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` is responsible for maintaining each module entity. MMFv2 default support these module (isp, h264, jpeg, aac\_encoder, aac\_decoder, amebapro\_audio, g711, mp4, rtp, rtsp). Each module is independent and corresponding to the individual input/output queue, state and in the `mm_context` of the module to update parameters and delivery entities.

### 1.1.3 Module Inter Connection

This section introduces `mm_siso_t`, `mm_simo_t`, `mm_miso_t`, `mm_mimo_t` and its corresponding create, delete, ctrl, start, stop, pause, resume function, which is responsible for connection and control between modules in mmfv2.

#### 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;
}
```

The SISO module is a unidirectional interface between modules. Input and output are independent. The state of the SISO module is responsible for determining the correct process. The `stack_size` is used to determine the size of the handler, and `xTaskHandle` task is reserved to control the use of the task.

There are several functions in the SISO module that are responsible for the module inter-connection. These functions are mostly used to update the status of the task and are handed over to the task handler for the main processing:

**siso\_create**

Pointer to the function that siso\_create declares the space of mm\_siso\_t and returns mm\_siso\_t entity after initialization.

**siso\_delete**

Pointer to the function stops SISO execution and free space of mm\_siso\_t entity.

**siso\_ctrl**

Pointer to the function sends the control command to siso module.

There are three types of operations MMIC\_CMD\_ADD\_INPUT, MMIC\_CMD\_ADD\_OUTPUT, MMIC\_CMD\_SET\_STACKSIZE.

MMIC\_CMD\_ADD\_INPUT link the input module to the input of the siso module, MMIC\_CMD\_ADD\_OUTPUT link the output module to the output of the siso module, and MMIC\_CMD\_SET\_STACKSIZE add size to the stack\_size of siso.

**siso\_start**

Pointer to the function checks whether there is anything in the input and output module before siso start. If the answer is yes, siso task will create a task handler to send data from input module to the output module.

**siso\_stop**

Pointer to the function updates status to MMIC\_STAT\_SET\_EXIT and wait for task handler to switch status to MMIC\_STAT\_EXIT.

**siso\_pause**

Pointer to the function updates status to MMIC\_STAT\_SET\_PAUSE and wait for task handler to switch status to MMIC\_STAT\_PAUSE.

**siso\_resume**

Pointer to the function updates status to MMIC\_STAT\_SET\_RUN and wait for the task handler to switch status to 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];  
};
```

The SIMO module is a unidirectional interface between modules. Input and output are independent, and `output_cnt` represents the number of simultaneous output modules. `status[4]` maintains the state of the SIMO module to confer the process is correct in the middle of the transfer, `stack_size` is used to determine the size of the handler task for intermediate transfers, and the `xTaskHandle` task of `xTaskCreate` is reserved to control the use of the task.

There are several functions in the SIMO module that are responsible for the module inter-connection. These functions are mostly used to update the state of the task and are handed over to the task handler for the main processing:

**simo\_create**

Pointer to the function that `simo_create` declares the space of `mm_simo_t` entity and returns `mm_siso_t` after initialization, and `simo_create` crate a queue head and `queue.lock` to protect the results of multiple outputs.

**simo\_delete**

Pointer to the function calls `simo_stop()` to stop SIMO execution and free space.

**simo\_ctrl**

Pointer to the function sends the control command to simo module.

There are six types of operations, `MMIC_CMD_ADD_INPUT` link the input module to the input of the simo module. `MMIC_CMD_ADD_OUTPUT0`, `MMIC_CMD_ADD_OUTPUT1`, `MMIC_CMD_ADD_OUTPUT2`, `MMIC_CMD_ADD_OUTPUT3` link output module to the corresponding output and increase the `output_cnt` to record number of output modules. `MMIC_CMD_SET_STACKSIZE` add size to `simo stack_size`.

**simo\_start**

Pointer to the function that simo\_start will create corresponding number of task handlers based on simo -> output\_cnt, and each task handler will be used to send the received data.

**simo\_stop**

Pointer to the function that simo\_stop sets each simo status to MMIC\_STAT\_SET\_EXIT, and waits for the task handler to switch each status to MMIC\_STAT\_EXIT.

**simo\_pause**

Pointer to the function that simo\_pause will set each simo -> status to MMIC\_STAT\_SET\_PAUSE according to pause\_mask, and wait for the task handler to switch each status to MMIC\_STAT\_PAUSE.

**simo\_resume**

Pointer to the function that simo\_resume will set each simo -> status to MMIC\_STAT\_SET\_RUN, and wait for the task handler to switch each status to 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;  
}
```

The MISO module is a unidirectional interface between modules. Input and output are independent, and input\_cnt represents the number of simultaneous input modules. status[4] maintains the state of the MISO module to confer the process is correct in the middle of the transfer, stack\_size is used to determine the size of the handler task for intermediate transfers, and the xTaskHandle task of xTaskCreate is reserved to control the use of the task.

There are several functions in the MISO module that are responsible for the module inter-connection. Most of these functions update the state of the task and hand over to the task handler for the main processing:

**miso\_create**

Pointer to the function that space of mm\_miso\_t is declared in miso\_create and initialized to return mm\_miso\_t entity.

**miso\_delete**

Pointer to the function that calls miso\_stop() to stop MISO and free space.

**miso\_ctrl**

Pointer to the function sends the control command to miso module. There are six operating can be use. MMIC\_CMD\_ADD\_INPUT0, MMIC\_CMD\_ADD\_INPUT1, MMIC\_CMD\_ADD\_INPUT2, MMIC\_CMD\_ADD\_INPUT3 link input module to the corresponding miso input and increase the value of input\_cnt for number of input module. MMIC\_CMD\_ADD\_OUTPUT links the output module to the output of the miso module.

the input module coupled to the input miso module and to increase the value of the number of recording input\_cnt input module, MMIC\_CMD\_ADD\_OUTPUT the output The module is linked to the output of the miso module, and MMIC\_CMD\_SET\_STACKSIZE adds the size to the stack\_size of the miso. MMIC\_CMD\_SET\_STACKSIZE adds size to the stack\_size of miso.

**miso\_start**

Pointer to the function checks whether there is anything in the input and output module before starting. If the answer is yes, a task handler will be created, and the data of the input module will be sent to the output module.

**miso\_stop**

Pointer to the function sets the miso status to MMIC\_STAT\_SET\_EXIT and wait for the task handler to switch the status to MMIC\_STAT\_EXIT.

**miso\_pause**

Pointer to the function that miso\_pause will set miso -> status to MMIC\_STAT\_SET\_PAUSE according to pause\_mask, waiting for the task handler to switch status to MMIC\_STAT\_PAUSE.

**miso\_resume**

Pointer to the function that miso\_resume will set miso -> status to MMIC\_STAT\_SET\_RUN, waiting for the task handler to switch each status to 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];
}
```

The MIMO module is a unidirectional interface between modules, Input[4] and output[4] represent input and output modules respectively, and input\_cnt represents the number of simultaneous input modules. Input and output support up to 4 outputs at the same time, MIMO module also needs mm\_mimo\_queue\_t queue[4] to maintain the synchronization problem of each input queue. Each mm\_mimo\_queue\_t has a lock and head to record the beginning of each queue and whether a program is already in use. status[4] maintains the state of the MIMO module to determine the correct process in the middle of the transfer, stack\_size is used to determine the size of the handler task for the intermediate transfer, and the xTaskHandle task of xTaskCreate is reserved to control the use of the task.

**mimo\_create**

Pointer to the function mimo\_create declares the space of mm\_mimo\_t entity and returns mm\_mimo\_t after initialization.

**miso\_delete**

Pointer to the function calls mimo\_stop() to stop the mimo module and free space.

**mimo\_ctrl**

Pointer to the function sends the control command to miso module. There are nine available operations (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) in mimo\_ctrl function. MMIC\_CMD\_ADD\_INPUT0, MMIC\_CMD\_ADD\_INPUT1,

MMIC\_CMD\_ADD\_INPUT2, and MMIC\_CMD\_ADD\_INPUT3 link input module to the input corresponding to the mimo module and increase the value of input\_cnt to record the number of input modules. MMIC\_CMD\_ADD\_OUTPUT0, MMIC\_CMD\_ADD\_OUTPUT1, MMIC\_CMD\_ADD\_OUTPUT2, and MMIC\_CMD\_ADD\_OUTPUT3 link the output module to the output of the mimo module and increase the value of output\_cnt to record the number of output modules. MMIC\_CMD\_SET\_STACKSIZE adds size to the stack\_size of mimo.

**mimo\_start**

Pointer to the function that mimo\_start will generate corresponding task handler according to output\_cnt to transfer the received data.

**mimo\_stop**

Pointer to the function that mimo\_stop will set the mimo status to MMIC\_STAT\_SET\_EXIT according to output\_cnt, and waiting for the task handler switch the status to MMIC\_STAT\_EXIT.

**mimo\_pause**

Pointer to the function that mimo\_pause will set each mimo -> status to MMIC\_STAT\_SET\_PAUSE according to pause\_mask, and waiting for the task handler to switch status to MMIC\_STAT\_PAUSE.

**mimo\_resume**

Pointer to the function that mimo\_resume will set mimo -> status in the task of MMIC\_STAT\_PAUSE for each status to MMIC\_STAT\_SET\_RUN, and waiting for the task handler to switch each status to MMIC\_STAT\_RUN.

## 1.2 Module Type and Module Parameter

MMFv2 Example supports many application scenarios, and the module parameter also supports manual adjustment. The reader will be able to understand the meaning and setting of different module parameters through this chapter.

### 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_SEMIPLANAR.
```

- **Resolution :** Supports settings up to 1080P
- **FPS :** DROP FRAME mechanism settings, detailed description as below
- **HW\_SLOT :** Maximum support 4
- **SW\_SLOT :** Which contains the number of HW\_SLOT, the rest will be used as a buffer
- **Format :** Currently only support YUV420
- **Boot mode :** Which divided into ISP\_FAST\_BOOT and ISP\_NORMAL\_BOOT, the former can support the ISP to initialize the speed in BOOT TIME, the latter initializes the ISP settings after the BOOT CODE is completed. If you want FAST BOOT, you need to set the ISP parameters first, but only support one to set one way isp.

```
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  
    .sensor_fps = SENSOR_FPS,
```



- PIN\_IDX : Default set PIN\_IDX to ISP\_PIN\_SEL\_S0
- MODE : If isp\_boot\_stream\_t.mode is ISP\_FAST\_BOOT, the MODE here must also be set to ISP\_FAST\_BOOT. The two parameters must be set the same.
- INTERFACE : Currently preset MIPI, also supports DVP interface
- CLK: Use the SENSOR.H header to choose the sensor.
- SENSOR\_FPS : Select the preset SENSOR FRAME according to SENSOR.H
- Wake mode: In addition to support wake mode, you can choose to wake up from BOOT, WLAN or GPIO situation. If the wakeup does not meet the setting, ISP FAST BOOT will not be performed. The default is WAKE\_FROM\_BOOT, which means you will enter FAST BOOT.
- ISP FW LOCATION: The default is XIP, and DRAM can be selected in special cases.



*Video/ Audio/ ISP parameters are not arbitrarily adjustable. On RTOS systems, resource configuration requires actuarial. Please adjust within the recommended range.*

About file size calculation, current default ISP output format is YUV420. Assuming current resolution is 1080P, FRAME size is  $1920 \times 1080 \times 1.5 = 3110400$  bytes. If HW\_SLOT =1, SW\_SLOT=3, then the memory size that be consumed is  $3110400 \times 3 = 9331200$  bytes. Regarding the SLOT number setting, it is currently recommended that HW\_SLOT be set to 2, and SW\_SLOT will be determined according to the FPS size. If it is set to 4 for 30FPS, it is set to 3 if 15FPS is recommended.

The maximum supported resolution combination is 1080P 15 FPS with 720P 30 FPS. If this limit is exceeded, there is a chance to drop the FRAME problem. The main limitation is the H264 compression speed limit.

There are two types of FRAME RATE settings here. The first one is to set the FRAME RATE of SENSOR itself. The second one is to set the required FRAME RATE through DROP FRAME mode, but it can't exceed the FRAME RATE of SENSOR itself. If the SENSOR FPS is 30, you can lose one at a time, that is, 30, you can lose one for two, it is 15, you can lose one for three, four because you can't divide it, so it can't support to set, and so on. You can get support frame rate is set to 30, 15, 10, 5, 1

Note on the simultaneous operation of ISP CHANNEL:

1. Do not initialize ISP CHANNEL at the same time. You need to wait for one of CHANNEL settings to continue setting other ISP CHANNEL.
2. Do not switch ISP CHANNEL at the same time. You need to wait for one of CHANNEL settings to continue setting other ISP CHANNEL.

Regarding the SENSOR switch problem, the first time ISP Stream is opened, the SENSOR will be initialized, but the second stream will not have this action, saving the switch SENOSR initialization time. If all STREAMs are turned off, the ISP will be restarted when the ISP is turned on. SENSOR initialization process, here also provides API settings to close SENSOR.

To use ISP FAST BOOT, initialize the ISP\_BOOT\_STREAM parameter is needed, set the variable SECTION to CINIT, and change the mode of the two parameters to ISP\_FAST\_BOOT\_MODE. ISP FAST BOOT will set up ISP parameters before BOOT CODE, than enter main program. The main purpose is to mount the ISP IRQ CALLBACK, which can be received before the first frame. Other program initialization steps need to be executed later. For related settings, please refer to the example to enable ISP\_BOOT\_MODE\_ENABLE Flag. We can choose the path from mp4 , rtsp or mux to see the result. The follow is set from mp4.

```
#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 0
#define ISP_BOOT_MUX 1
#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 : video width
- height : video length
- bps : Bit per second(Bit data transmitted per second)
- fps : Frame per second(Number of frames transmitted per second)
- gop : Grout of picture(How many frames are updated per I Frame)
- rc\_mode : Rate control mode(currently available CBR, VBR, FIXQP and Rate Adaptive)
- mem\_total\_size : H264 encoder memory size capacity
- mem\_block\_size : Block size used by Memory pool
- men\_frame\_size : Set a maximum FRAME SIZE capacity

*Rate Control Adjustment.***CBR:**

- Fixed bit rate, bit rate is controlled by V1\_BITRATE.
- QP range default value is [10, 10, 51].
- If there is an adjustment requirement in the [minQp, minIQp, maxQp] control of h264\_rc\_parm\_t in module\_h264.h, use API h264\_control, the example is as follows (minIQp is I frame QP.)

**VBR:**

- The bit rate is changed by V1\_BITRATE. When the screen is still, it will automatically adjust to 1/2 bit rate. When the screen changes, it will exceed V1\_BITRATE. The excess amplitude is controlled by maxQp. The larger the maxQp, the smaller the bit rate is.
- QP range default value is [20, 20, 40]

**ABR:**

- ABR is a long-term average, and the principle is similar to CBR, which allows the bit rate to be significantly exceeded by the preset V1\_BITRATE when the picture is changed. The excess amplitude is controlled by maxQp. The difference with CBR is that it does not force the bit rate immediately when the picture changes, but takes a long time average. The difference with VBR is that the picture is not automatically adjusted to 1/2 bit rate when the picture is still.
- QP range default value is [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: The bit rate control mode [RC\_MODE\_H264CBR/ RC\_MODE\_H264VBR/ RC\_MODE\_H264ABR]
- minQp: Minimum QP recommended value range: [10, 20]
- minIQp: Minimum QP of the I frame, It is recommended that the value of this parameter be equal to minQp in normal scenarios.
- maxQp: Maximum QP recommended value range: [40, 51]

- maxBps: The maximum bit rate per second (for VBR)
- minBps: The minimum bit rate per second (for VBR)
- intraQpDelta: Adjusting the quantization of the intra frames. [-12..12]
- mbQpAdjustment: To reduce coding artifacts on low-detail areas. This is enable by setting a negative MB QP adjustment value. [-8..0]

#### Improving Image Quality:

- Set the maxQp: Setting the maximum QP helps effectively protect the image quality, but bit rate overshooting is prone to occur.
- Set the minQp: This parameter is used to control the highest image quality. The QP is not decreased any longer after being adjusted to this value, which may cause bit rate insufficiency. This parameter is intended to reduce the bit rate in simple still scenarios.
- Set the intraQpDelta: The intra frames in H.264 video can sometimes introduce noticeable flickering because of different prediction method. This problem can be overcome by adjusting the quantization of the intra frames compared to surrounding inter frames.
- Set the mbQpAdjustment: To reduce coding artifacts on low-detail areas. The recommended value range: [-1,-2].

**1.2.3 RTSP**

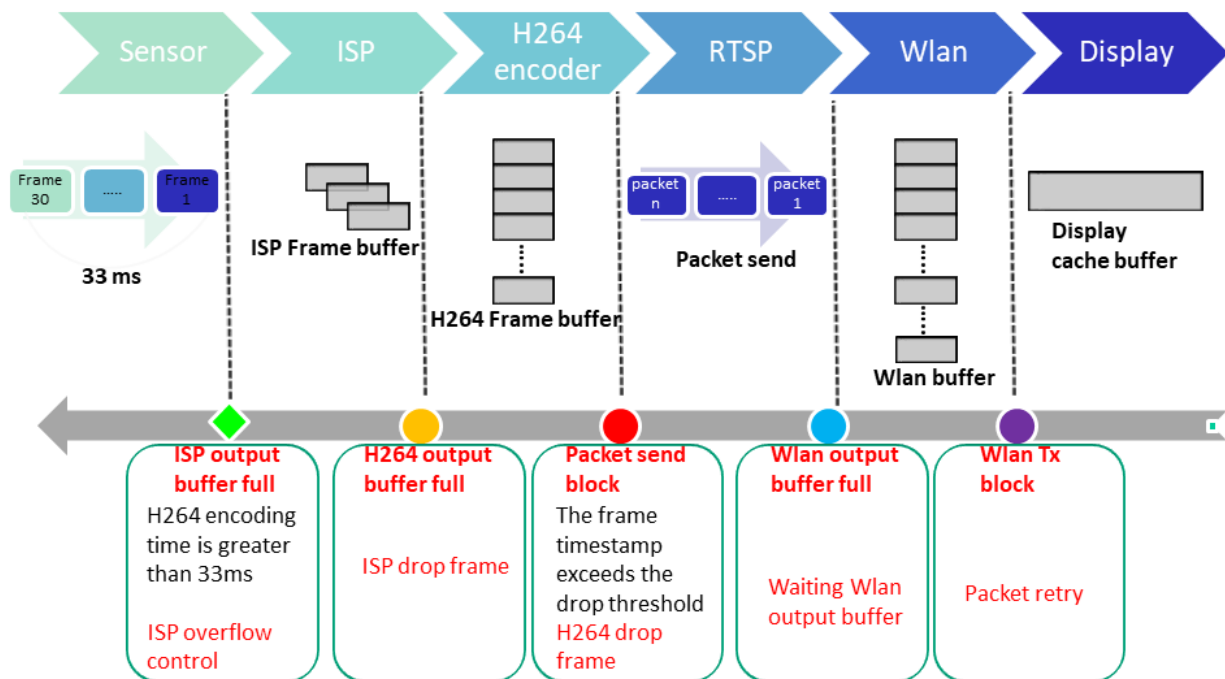
```
typedef struct rtsp2_params_s{
    uint32_t type;
    union{
        struct rtsp_video_param_s{
            uint32_t codec_id;
            uint32_t fps;
            uint32_t bps;
            uint32_t ts_flag;
            char* sps;
            char* pps;
            char* lv;
        }v;
       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

- type: Media type, available Video, Audio
- codec\_id: Codec ID , available 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
- ts\_flag: H264 and AAC rtsp time sync enable switch
- sps,pps,lv: setting sps, pps and profile level of H264
- 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**  
The setting of ISP output buffer is related to the speed of h264 encode (1080P 30ms, 720P 15ms). If it is too small, it will cause buffer full, and then start the ISP overflow control mechanism.
- **ISP output buffer setting**  
It is recommended that FPS 30 be set to 3 and FPS 15 to be set to 2.

**ISP drop frame**

- H264 output buffer full  
When H264 output buffer full, the current ISP frame will be dropped and the program of H264 encode will not be entered.
- H264 output buffer setting  
If the H264 output buffer setting is too large, the ISP drop frame mechanism may be invalid and the latency may be increased.  
The H264 output buffer setting is too small, and the cache effect is insufficient, which may cause the ISP drop frame to increase.  
It is recommended to set to 6 for Streaming and 70 for Storage.

**H264 drop frame**

[CMD\\_RTSP2\\_SET\\_DROP\\_FRAME\\_EN](#)

[CMD\\_RTSP2\\_SET\\_DROP\\_TIME](#)

[CMD\\_RTSP2\\_SET\\_DROP\\_FRAME\\_FORCEI](#)

- Packet send block  
When the H264 drop frame program is enabled, if the delay between the transmission and the ISP frame output timestamp is greater than the drop threshold, the H264 drop frame handler is started.  
When dropping a frame, it will drop to the next I frame to avoid breaking the image.。
- Drop threshold setting  
If the drop threshold setting is too small, the H264 drop frame mechanism will frequently trigger. It is recommended to set the time to 3 times the H264 output buffer cache.。
- Drop frame and forcei  
Improve the problem of dropping a large number of frames at a time, but the I frame size is larger and requires more bandwidth transmission, which may cause the FPS to remain at a low level.



### 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 : Image width
- Height : Image length
- LEVEL : Image quality 0-9, The higher the value, the better the picture quality.
- FPS: How many FRAMEs per second
- Mem\_total\_size : JPEG encoder memory size capacity
- Mem\_block\_size : Block size used by Memory pool
- Mem\_frame\_size : Set a maximum FRAME SIZE capacity

### 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 : Must be the same as the Audio codec setting. For example, when the Audio codec is set to ASR\_8KHZ, it must be set to 8000 here.
- .channel : Mono is set to 1, and stereo is set to 2. This setting is related to Audio codec. Amebapro built-in codec is mono, so set it to 1.
- bit\_length : Must be the same as the word\_length of the audio codec, such as audio codec word\_length = WL\_16BIT, which must be set to 16.
- Mem\_total\_size : AAC encoder output memory size.
- Mem\_block\_size : Block size used by Memory pool.
- Mem\_frame\_size : Set maximum FRAME SIZE capacity

### 1.2.7 AAC Decoder

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

- **sample\_rate** : Need to be the same source to decode correctly.
- **channel** : Need to be the same source to decode correctly.
- **type** : TYPE\_ADTS is used when the source is AAC encoder, TYPE\_RTP\_RAW is used when source is RTP, and TYPE\_TS is not currently supported.

### 1.2.8 Audio Codec

AEC (Acoustic Echo Cancellation) is included in this module.

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

- Sample rate currently supports : 8K, 16K, 32K, 44.1K, 48K, 88.2K, 96K HZ
- word\_length currently supports: 16, 24 bit
- Microphone gain value support : 0, 20, 30, 40 DB
- Channel currently supports mono, set to 1
- If enable\_aec set to 1, echo cancellation will be enabled. If not set or 0, the echo cancellation will be disabled.

### 1.2.9 RTP Input

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

- **valid\_pt** : Processable RTP Payload types, set 0xFFFFFFFF to handle PCMU(0), PCMA(8), DYNAMIC (96)
- **port** : The port that receives the RTP packet
- **frame\_size**: Maximum RTP packet size

- **cache\_depth :** The number of caches for RTP packets. The cache handler will send the RTP packet in the cache to the output of the module when the number of packets in the cache  $\geq 50\%$  cache\_depth

### 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 currently supports PCMU and PCMA codec modes.

**buf\_len :** Determine the length of the encode buffer (byte)

**mode :** Determine whether the G711 codec module is encode or 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 : Video length
- Height : Video height
- FPS : Frame number per second
- GOP : Update I frame cycle
- SAMPLE RATE : Audio sample rate
- CHANNEL: Audio channel number.
- RECORD LENGTH : Video length in seconds
- Record type: Select STORAGE\_ALL (with video), STORAGE\_VIDEO (video only),  
STORAGE\_AUDIO (sound only)
- REOCD FILE NUM: Number of videos
- RECORD FILE NAME: Video name
- FATFS BUF SIZE: FATFS cache 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 currently supports : 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 : currently supported sampling rate is the same as the sample rate, but less than or equal to sample\_rate.
- word\_length currently supports : 16、24、32 bit.
- out\_word\_length currently supported bit depth is the same as the word\_length, but less than or equal to word\_length.
- channel : Currently supports stereo or mono, please set to 2 or 1, and also supports 5.1 channels (but only support tx)

### 1.2.11 httpfs

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

- fileext : File extension
- filedir : Directory where the file is located
- request\_string : The string of http page
- fatfs\_buf\_size : Buffer size of read file

### 1.3 Using the MMF v2 example

Describe how to use the sample program to construct the data stream required by the terminal application.

#### 1.3.1 Sample Program

- The sample program is located at:  
component\common\example\media\_frameworkexample\_media\_framework.c
- Must set platform\_opts.h before use.
- Open project\realtek\_amebapro\_v0\_example\inc\platform\_opts.h

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

- Modify CONFIG\_EXAMPLE\_MEDIA\_FRAMEWORK from 0 to 1, compile and execute

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

### 1.4 Selecting and setting up the sample program

Steps:

- Choose the appropriate sample program
- How to adjust the Audio/Video parameters



*Note: IAR version requires 8.30*

### 1.4.1 Choose the proper sample program

The main sample program name is `example_mmf2_signal_stream_main`, and all examples are comment out in default. Pick the example want to open before using it, remove the comment, and recompile. Opening more than two examples at the same time will result in unpredictable program execution results.

```
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
}
```

For example, to open the first example in the sample program

```
// CH1 Video -> H264 -> RTSP
mmf2_example_v1_init();
// CH2 Video -> H264 -> RTSP
```

#### **Currently supported example**

Example	Description	Result
mmf2_example_v1_init	CH1 Video -> H264 -> RTSP	Transfer AmebaPro's H264 video stream over the network
mmf2_example_v2_init	CH2 Video -> H264 -> RTSP	Transfer AmebaPro's H264 video stream over the network



mmf2_example_v3_init	CH3 Video -> JPEG -> RTSP	Transfer AmebaPro's JPEG video stream over the network
mmf2_example_simo_init	1Video (H264) -> 2 RTSP(V2)	Transmitting two H264 video streams from AmebaPro over the network, the source of the video is the same ISP stream
mmf2_example_a_init	1 Audio (AAC) -> RTSP (A)	AmebaPro's AAC sound stream over the network
mmf2_example_av_init	1 Video (H264) 1 Audio -> RTSP	Transfer AmebaPro's H264 video and AAC sound stream over the network
mmf2_example_av2_init	2 Video (H264) 1 Audio -> 2 RTSP (V1+A, V2+A)	Transmitting two H264 videos and AAC audio streams from AmebaPro over the network. The source of the videos is different ISP streams.
mmf2_example_av21_init	1 Video (H264) 1 Audio -> 2 RTSP (V+A)	Transfer two copies of AmebaPro's H264 video and AAC sound stream through the network, the video source is the same ISP stream
mmf2_example_audioloop_init	PCM audio -> PCM audio , audio loopback	The sound received by AmebaPro can be broadcast from the 3.5 audio channel of AmebaPro , and the PCM transmission is directly used in the procedure .
mmf2_example_g711loop_init	audio -> G711E -> G711D -> audio	The sound received by AmebaPro can be broadcast from the 3.5 audio channel of AmebaPro
mmf2_example_aacloop_init	audio -> AAC -> AAD -> audio	The sound received by AmebaPro can be broadcast from the 3.5 audio channel of AmebaPro
mmf2_example_i2s_audio_init	I2s -> PCM audio, audio loop back	Sound received by i2s can be played from the 3.5 audio channel of AmebaPro , and the PCM transmission is directly used in the procedure .
mmf2_example_rtp_aad_init	RTP -> AAD -> audio	Stream AAC sound over the network to AmebaPro for playback
mmf2_example_2way_audio_init	AUDIO -> AAC -> RTSP RTP -> AAD -> AUDIO	Stream AAC sound to AmebaPro via the network and transmit the sound received by AmebaPro over the network
mmf2_example_joint_test_init	ISP -> H264 -> RTSP (with AUDIO) ISP -> H264 -> RTSP (with AUDIO) AUDIO -> AAC -> RTSP RTP -> AAD -> AUDIO	(1) Transmitting two H264 video and AAC audio streams from AmebaPro over the network. The source of the video is different ISP streams. (2) Stream AAC sounds to AmebaPro for playback over the network
mmf2_example_av_mp4_init	1 Video (H264) 1 Audio -> MP4 (SD card)	AmebaPro will record three videos to the SD card for 30 seconds each The default storage name is : 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 -> AAC -> AUDIO	(1) Transfer AmebaPro's H264 video and AAC sound stream over the network (2) AmebaPro will record three videos to the SD card for 30 seconds each. The default storage name is : AmebaPro_recording_0.mp4 AmebaPro_recording_1.mp4 AmebaPro_recording_2.mp4 (3) Streaming AAC sounds to AmebaPro via the network (1) video source of (2) is different from the ISP stream
mmf2_example _h264_2way_ audio_pcmu_ doorbell_init	ISP -> H264 -> RTSP (V1) AUDIO -> G711E -> RTSP RTP -> G711D -> AUDIO ARRAY (PCMU) -> G711D -> AUDIO (doorbell)	(1) Transmitting AmebaPro's H264 stream and PCMU sound stream over the network (2) AAC sound can be streamed to AmebaPro via the Internet (3) Play PCMU sound array in AmebaPro (default is the doorbell)
mmf2_example _pcmu_array_ rtsp_init	ARRAY (PCMU) -> RTSP (A)	Transmitting PCMU sound arrays within AmebaPro over the network
mmf2_example _aac_array_rtsp _init	ARRAY (AAC) -> RTSP (A)	Transfer AAC sound arrays in AmebaPro over the network
mmf2_example _h264_array_rts p_init	ARRAY (H264) -> RTSP (V)	Transfer H264 stream array in AmebaPro over the network
mmf2_example _v1_param_ change_init	H264/ISP parameter change	Transfer AmebaPro's H264 video over the network and support dynamic adjustment of video parameters
mmf2_example _av_mp4_httpfs _init	1 Video (H264) 1 Audio -> MP4 (SD card) Http File Server	AmebaPro will record a video every 30 seconds and save it to the SD card. The default is to record 30 files, and repeat the recording after the end. The default storage name is : mp4_record_0.mp4~mp4_record_29.mp4 Also open Http File Server for client to do playback.

Please refer to the steps below to learn how to use the MMFv2 example.

#### **Pre-requisites**

- AmebaPro board \* 1

- Camera sensor board\* 1
- USB cable \* 1
- Wifi used to transfer rtsp stream
- MicroSD \* 1

**Hardware setup**

- Connect the Camera sensor board to the AmebaPro CON1 port
- Connect the USB cable to the AmebaPro CON8 port and the other end to the PC
- Connect the MicroSD card to the MicroSD card slot.

**Softer setup**

Please refer to the compilation and execution chapter in AN0300 Realtek AmebaPro user\_manual\_en.

- Open SDK\project\realtek\_amebapro\_v0\_example\inc\platform\_opts.h
- Open example\_media\_framework, set CONFIG\_EXAMPLE\_MEDIA\_FRAMEWORK to 1

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

- Open SDK\common\example\media\_framework\example\_media\_framework.c, and select the sample program which locate near the bottom of example\_mmf2\_signal\_stream\_main. To test the full function sample program, it is recommended to uncommon and compile mmf2\_example\_joint\_test\_rtsp\_mp4\_init();

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

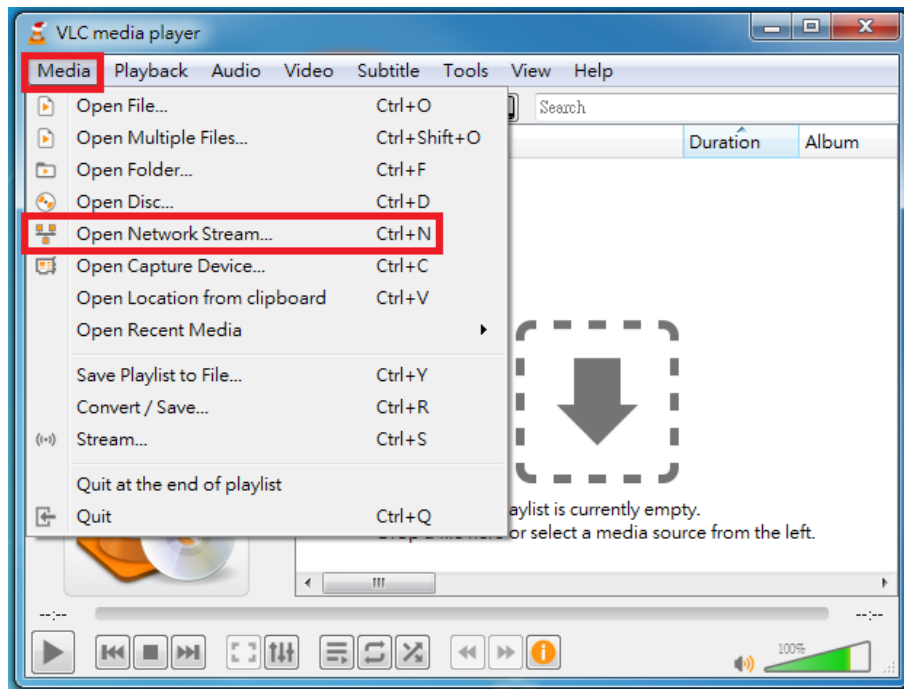
mmf2_example_joint_test_rtsp_mp4_init();
```

- Compile and execute firmware

**Execution and testing**

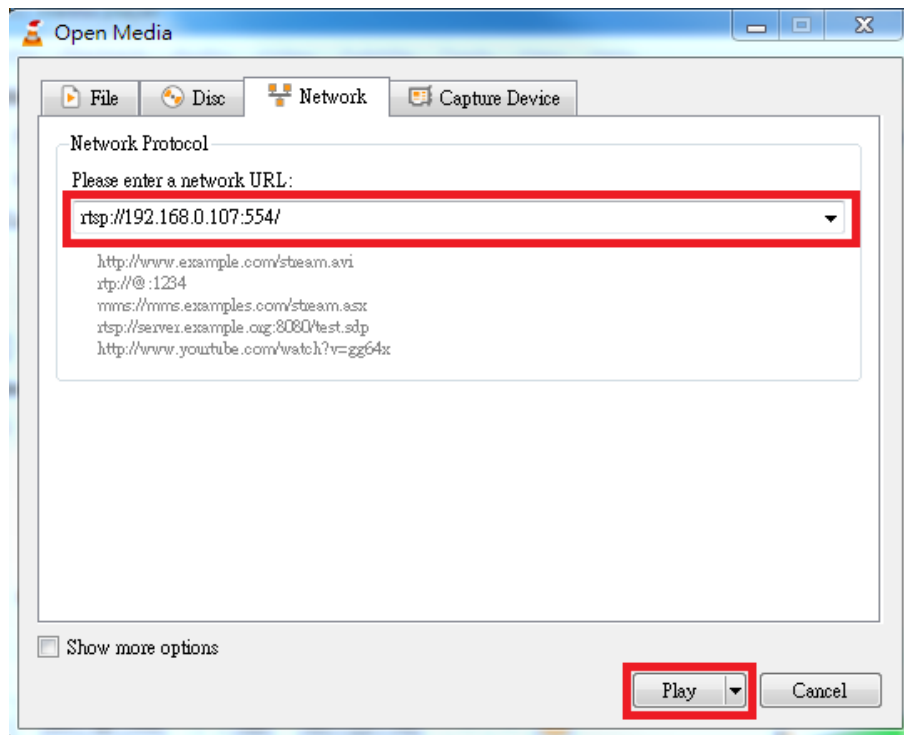
Before executing example, must set Tera Term or try PuTTY first and set the serial port to COMX/115200: Port number. Once the setting is completed, AmebaPro is also connected with the PC and booted to get the Log message output by AmebaPro.

- In order to execute rtsp stream, you must set up AmebaPro first to connect to the network. Please refer to the steps below.  
ATW0 = <Name of WiFi SSID> : Set the WiFi AP to be connected  
ATW1 = <Password> : Set the WiFi AP password  
ATWC : Initiate the connection
- When the “RTSP stream enabled” message shown on console, it indicates that the RSTP server is already running. To stream audio/video from AmebaPro to VLC player, please open the VLC media player




Click “Media” -> “Open Network Stream”.

1. Enter “rtsp://xxx.xxx.xxx.xxx:yyy/”, and click “Play”.



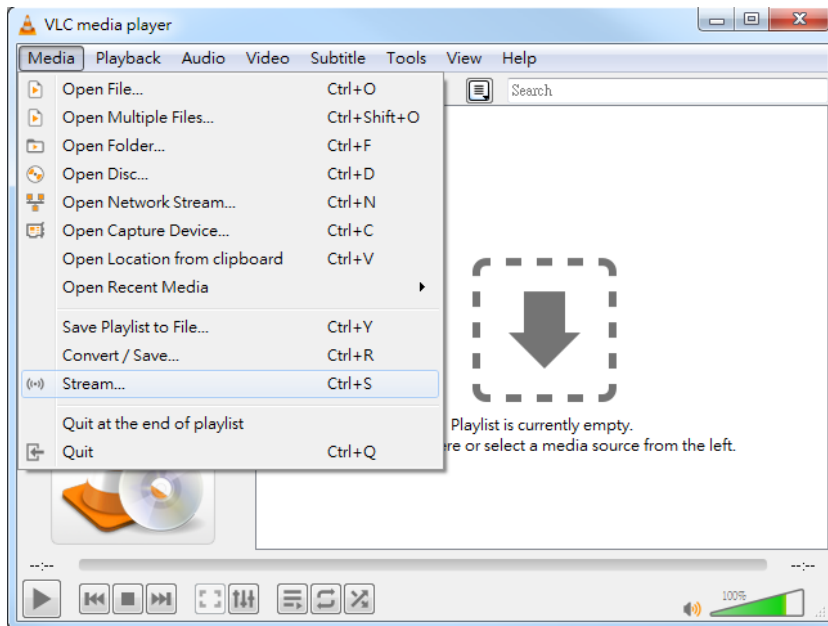
xxx.xxx.xxx.xxx: the Ameba IP address.

yyy: RTSP server port (default is 554).

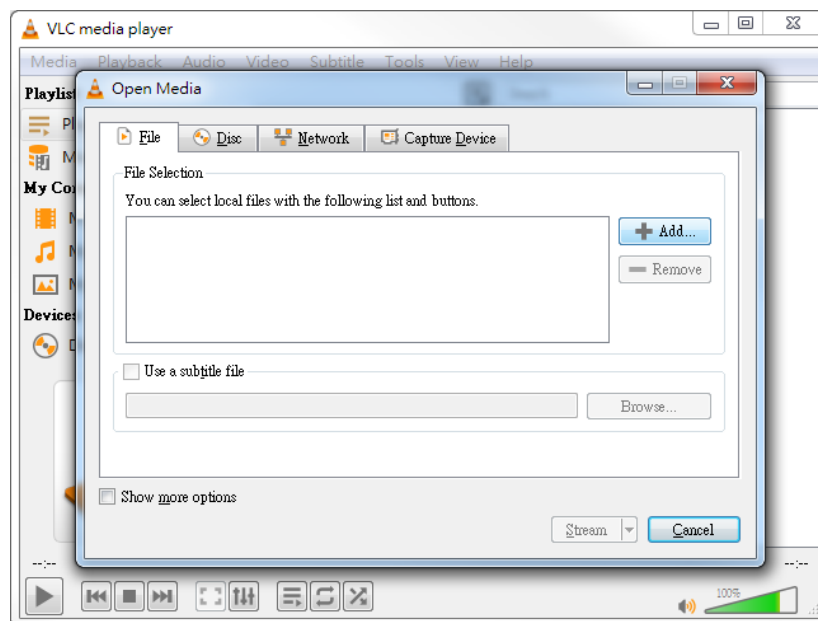
 *Note: For latency related settings, please refer to section 1.4.5*

*VLC media player settings*

- To stream audio from VLC player to AmebaPro, please open the VLC media player.
  1. click “Media” -> “Stream”.



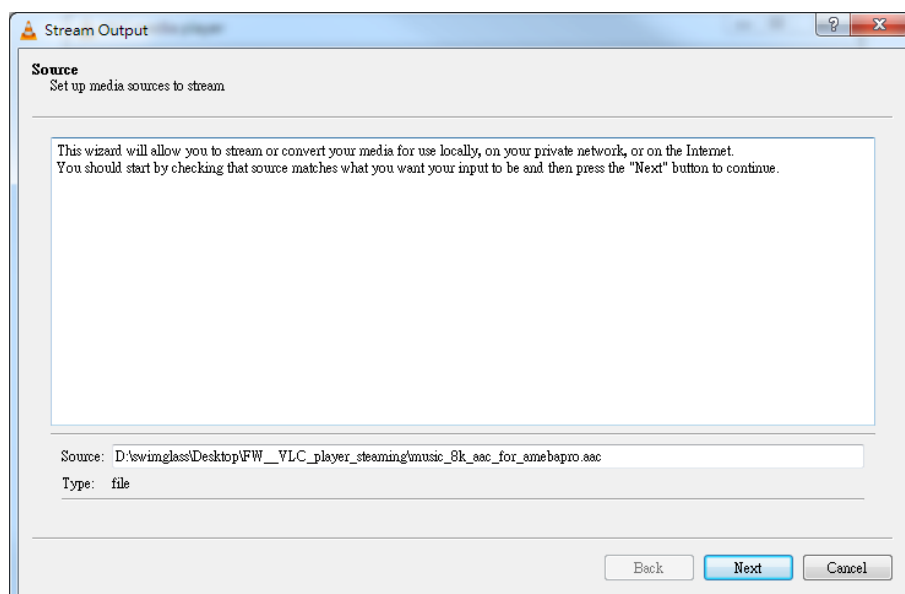
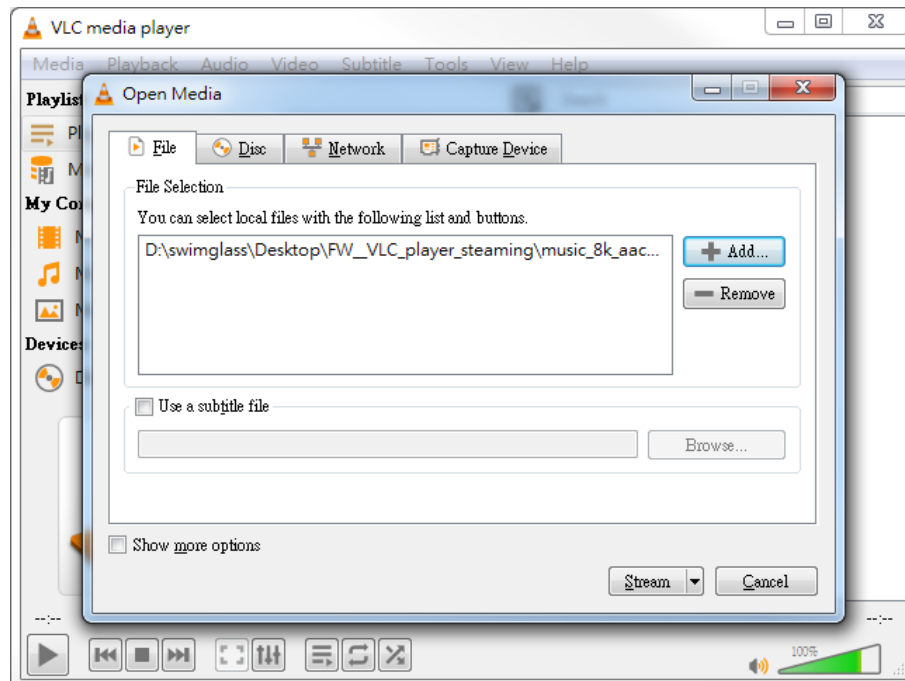
2. Select “File” and click “Add”.



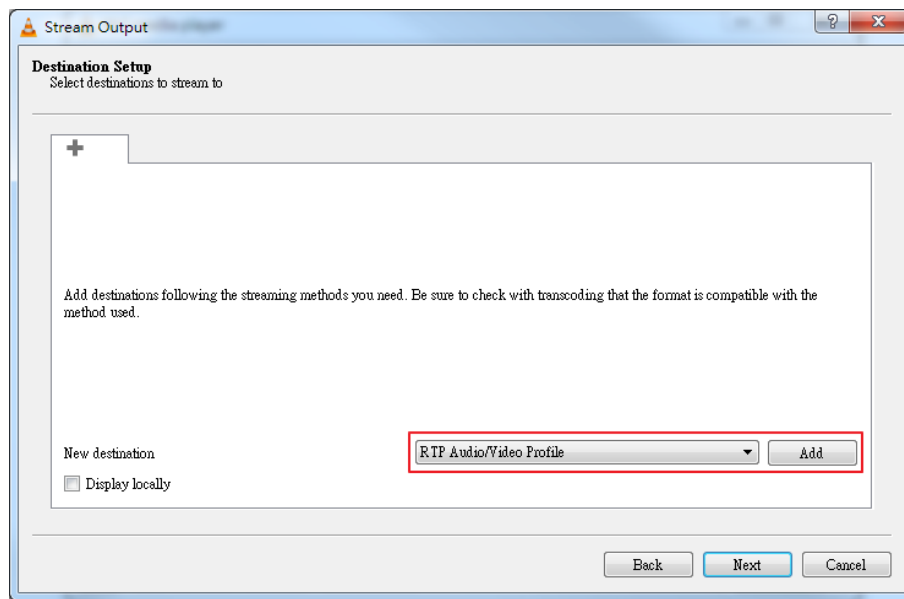
(If the startup example is RTP -> AAD -> AUDIO please select the audio file with the file name .aac. If the startup example is RTP -> G711D -> AUDIO, please select the audio file with the file name .wav)

*Note: Download and use ffmpeg to generate a compatible WAV file with following command:*  
***ffmpeg -i input.wav -acodec pcm\_mulaw -ac 1 -ar 8000 -ab 64k output.wav***

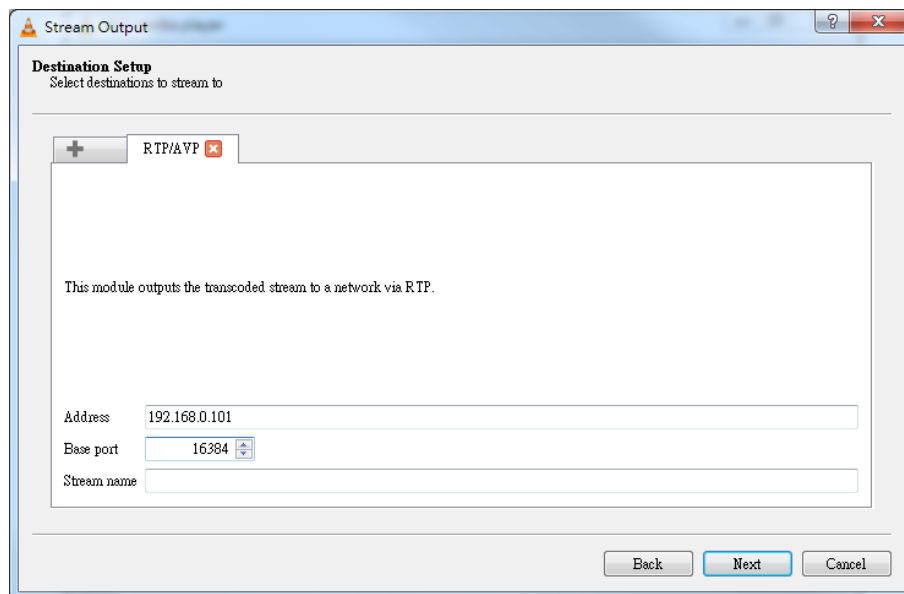
3. Select the audio file to be added, and click “Stream” -> “Next”.



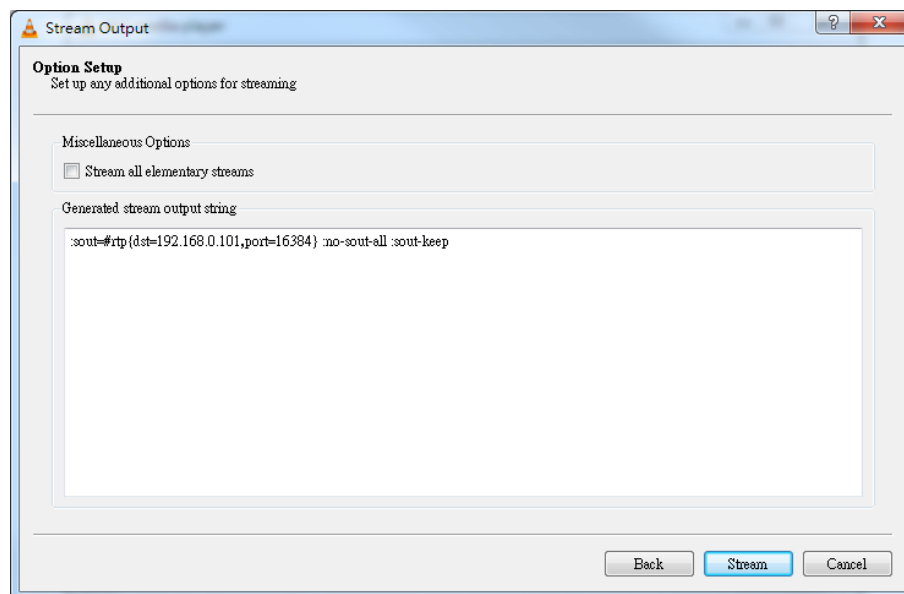
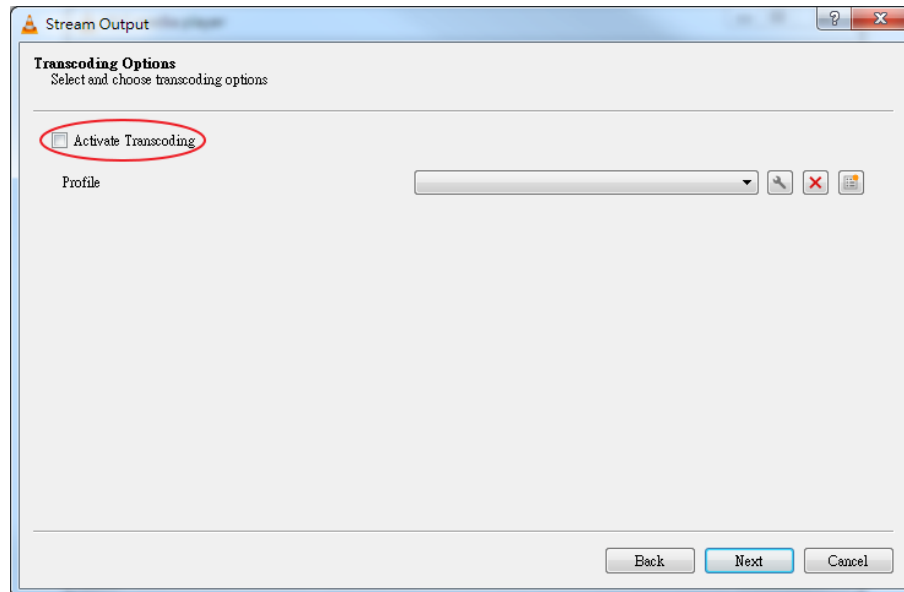
4. Select “RTP Audio/Video Profile”, and click “Add”



5. Enter AmebaPro's IP Address in “Address” field, with “Base port” set to 16384, and click “Next”.



6. Confirm “Activate Transcoding” is unchecked, and click “Next” -> “Stream”.  
Then the sound can be heard on AmebaPro 3.5mm audio jack.



### **Individual instructions and compilation options**

If there are more than the above instructions, add the following

- mmf2\_example\_v1\_init(Source AmebaPro Camera, Sink RTSP Stream) :  
To modify the image quality parameter, please modify the V1 parameter in example\_media\_framework.h. For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.
- mmf2\_example\_v2\_init(Source AmebaPro Camera, Sink RTSP Stream) :



To modify the image quality parameter, please modify the V2 parameter in `example_media_framework.h`. For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2. If you want to enable ISP BOOT MODE, please enable `ISP_BOOT_MODE_ENABLE` and `ISP_BOOT_RTSP`, and VLC will not be an instant image, but the BOOT TIME image at the time.

- `mmf2_example_v3_init`(Source AmebaPro Camera, Sink RTSP Stream) :  
To modify the image quality parameter, please modify the V1 parameter in `example_media_framework.h`.  
Also set `#define ENABLE_V3_JPEG V3_JPEG_STREAMING` in `example_media_framework.h`  
For the ISP related parameter settings, please refer to chapter 1.2.1. For JPEG related parameter, please refer to chapter 1.2.3.
- `mmf2_example_simo_init`(Source AmebaPro Camera, 2 Sink RTSP Stream) :  
Two VLC video players must be open at the same time, the other RTSP port is 555.  
To modify the image quality parameter, you can modify the V2 parameter in `example_media_framework.h`.  
For the ISP related parameter settings, please refer to chapter 1.2.1.  
For H264 related parameter, please refer to chapter 1.2.2.
- `mmf2_example_a_init`(Source AmebaPro Microphone, Sink RTSP Stream) :  
For audio related parameters, please refer to chapter 1.2.4~1.2.6
- `mmf2_example_av_init`(Source AmebaPro Camera/Mic, Sink RTSP Stream) :  
To modify the image quality parameter, please modify the V1 parameter in `example_media_framework.h`.  
For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.  
For audio related parameters, please refer to chapter 1.2.4~1.2.6
- `mmf2_example_av2_init`(Source AmebaPro Camera/Mic, Sink RTSP Stream) :  
Two VLC video players must be open at the same time. The other RTSP port is 555.  
If image quality parameters modification is needed, modify the V1 and V2 parameters in `example_media_framework.h`.  
For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.  
For audio related parameters, please refer to chapter 1.2.4~1.2.6
- `mmf2_example_av21_init`(Source AmebaPro Camera/Mic, Sink RTSP Stream) :  
Two VLC video players must be open at the same time. The other RTSP port is 555.

To modify the image quality parameter, please modify the V1 parameter in `example_media_framework.h`.

For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.

For audio related parameters, please refer to chapter 1.2.4~1.2.6

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

For audio related parameters, please refer to chapter 1.2.4~1.2.6

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

For audio related parameters, please refer to chapter 1.2.4~1.2.6

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

For audio related parameters, please refer to chapter 1.2.4~1.2.6

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

For audio related parameters, please refer to chapter 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) :

Two VLC video players must be open at the same time. The other RTSP port is 555. If image quality parameters modification is needed, please modify the V1 and V2 parameters in `example_media_framework.h`.

For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.

For audio related parameters, please refer to chapter 1.2.4~1.2.6

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

To modify the image quality parameter, please modify the V2 parameter in `example_media_framework.h`.

For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.

For audio related parameters, please refer to chapter 1.2.4~1.2.6.

If you want to turn on ISP BOOT MODE, enable `ISP_BOOT_MODE_ENABLE` and `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) :

If image quality parameters modification is needed, please modify the V1 and V2 parameters in `example_media_framework.h`.

For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.

For audio related parameters, please refer to chapter 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) :

If image quality parameters modification is needed, please modify the V1 and V2 parameters in example\_media\_framework.h.

For the ISP related parameter settings, please refer to chapter 1.2.1. For H264 related parameter, please refer to chapter 1.2.2.

For audio related parameters, please refer to chapter 1.2.4~1.2.6

Please use the .wav audio file for the music extension file played from the PC.

- 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 Adjusting the Video/Audio Parameters of MMFv2 Example

- ISP

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

- Resolution: Supports up to 1080P.
  - FPS: Currently open for 30, 15, 10, 5 and 1.
  - HW\_SLOT: Currently supports up to 4.
  - SW\_SLOT: contains the number of HW\_SLOT, the rest will be used as a cache
- Regarding file size calculation, current default ISP output format is YUV420 Format. Assuming the current resolution is 1080P, a FRAME size will be  $1920 \times 1080 \times 1.5 = 3110400$  bytes. If HW\_SLOT = 2, SW\_SLOT=4, then the memory consumed size is



*Video/Audio/ISP parameters are not arbitrarily adjustable. On RTOS systems, resource configuration requires actuarial. Please adjust within the recommended range.*

$3110400 \times 4 = 12441600$  bytes. Regarding the SLOT number setting, it is currently recommended that HW\_SLOT be set to 2, and SW\_SLOT will be determined according to the FPS size. If it is set to 4 for 30FPS, it is set to 3 if 15FPS is recommended.

### 1.4.3 Adjusting LWIP Parameters

For video streaming application, it requires stable and higher performance for network transmission. To meet this requirement, there are some modifications for LWIP parameters within the standard SDK, by enlarging some LWIP buffers to improve the network transmission efficiency. These modifications reside in component\common\api\network\include\lwipopts.h:

```
#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
```

CONFIG\_VIDEO\_APPLICATION is defined as 1 in project\realtek\_amebapro\_v0\_example\inc\platform\_opts.h as default.

If user has some other requirements for their applications, you can change the LWIP parameters based on current memory usage. Take TCP streaming as example, if the RTT(Round Trip Time) is quite long, e.g. TCP server locates at foreign network, user can try to enlarge the TCP\_SND\_BUF parameter to make it send more data each time, so that the overall transmission performance would not be low due to the long RTT. As standard SDK, user can adjust some LWIP parameters as below to make the TCP\_SND\_BUF set to maximum value:

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

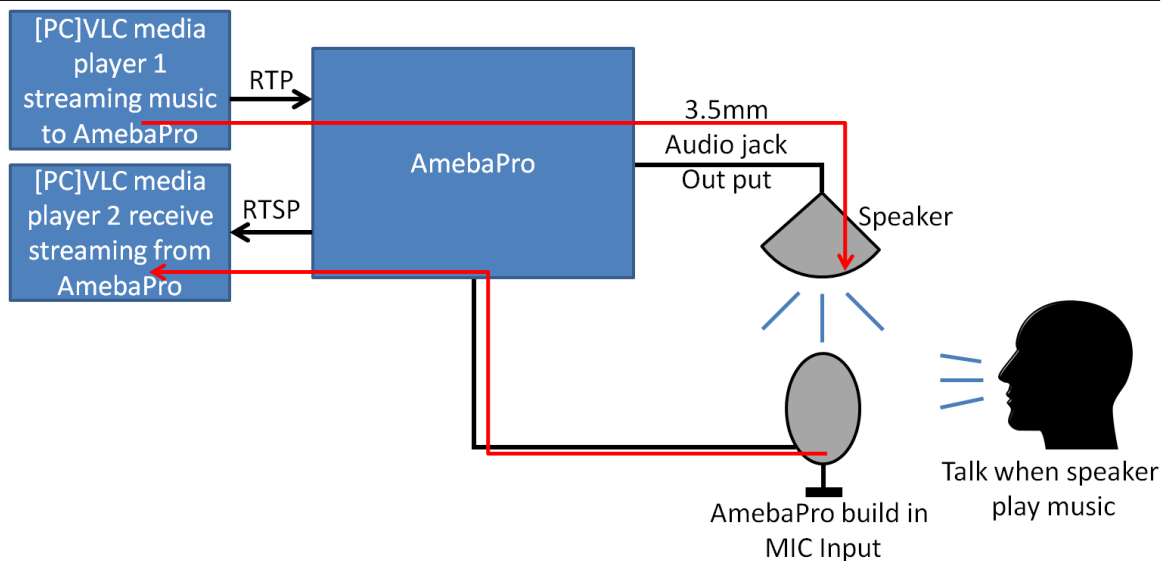
For above configuration, we need to relocate some LWIP buffer to external RAM, by modifying project\realtek\_amebapro\_v0\_example\EWARM-RELEASE\application\_is.icf and move the “*section \*.itcm.bss object memp.o*” from DTCM\_RAM\_region to ERAM\_BSS. For this configuration, the external RAM would get extra cost about 120KB, but the transmission performance should be improved for long RTT case.

#### **1.4.4 Echo Cancellation**

Echo cancellation is default provided in the audio part of MMFv2. To test whether the echo cancellation function is correct, use VLC media player to verify it on the computer.

The verification method is as follows:

1. Use VLC media player on the PC to stream music to AmebaPro.
2. Put AmebaPro speaker next to AmebaPro build in Mic and speak at the same time.
3. Then pass the received sound to the VLC media player on the PC via AmebaPro to see if the sound in step 1 is small or not (the sound of the speech and the sound of the music, but the sound of the music is relatively small)



### Parameter adjustment

Speex echo cancellation provides three parameters that can be adjusted, sampling rate, frame size and filter length. Sampling rate is recommended to use 8kHz, and the corresponding frame size is 20ms. Frame size is the amount of data (in samples) you want to process at once and filter length is the length (in samples) of the echo cancelling filter you want to use (also known as tail length). It is recommended to use a frame size in the order of 20 ms (or equal to the codec frame size) and make sure it is easy to perform an FFT of that size (powers of two are better than prime sizes). The recommended tail length is approximately the third of the room reverberation time. For example, in a small room, reverberation time is in the order of 300 ms, so a tail length of



*Echo cancellation example is included in MMFv2 audio example*

100ms is a good choice (800 samples at 8000 Hz sampling rate).

### 1.4.5 Capture the first frame and related notes

The purpose of this function can speed up the appearance of ISP Frame. The current measurement time is about 188ms from boot to frame done. The main method is to start ISP initialization in the bootloader. At this time, you can wait for the frame to appear in main. For details, please refer to ISP 1.2.1.

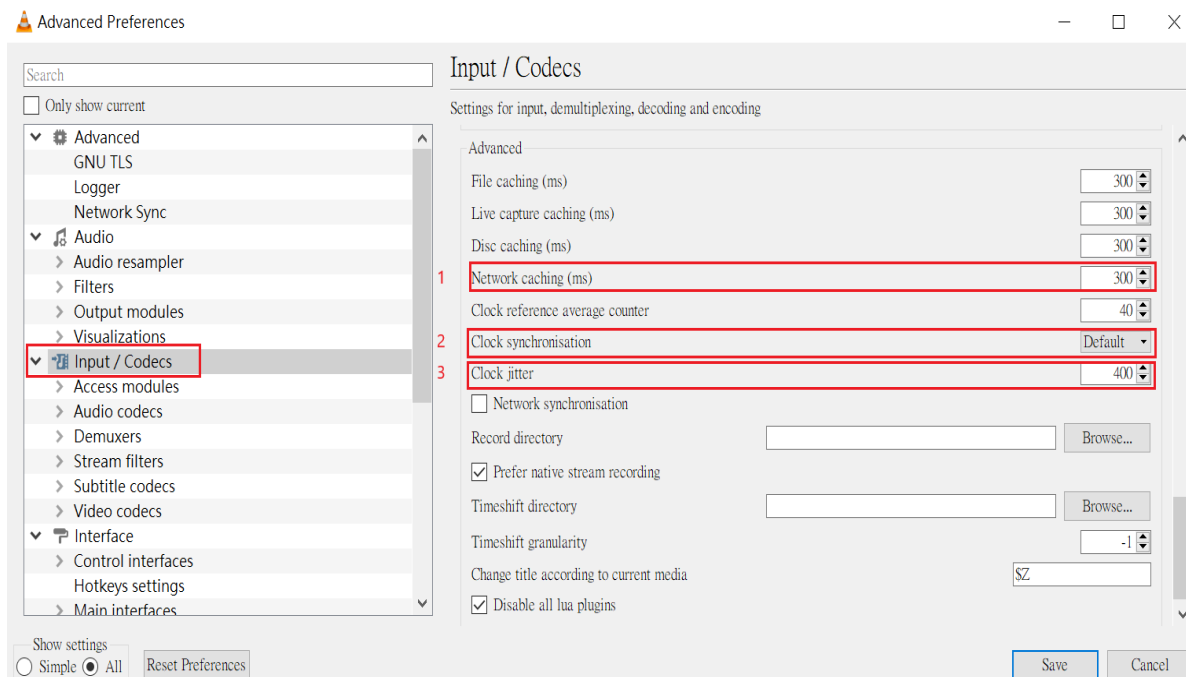
Note: When opening two channels at the same time, you need to execute the First boot all the way. You can refer to the ISP\_BOOT\_MUX example.

In addition, if you need to change the Frame rate later, please change Flag back to Normal boot after the setting is completed.

### 1.4.6 VLC media player settings

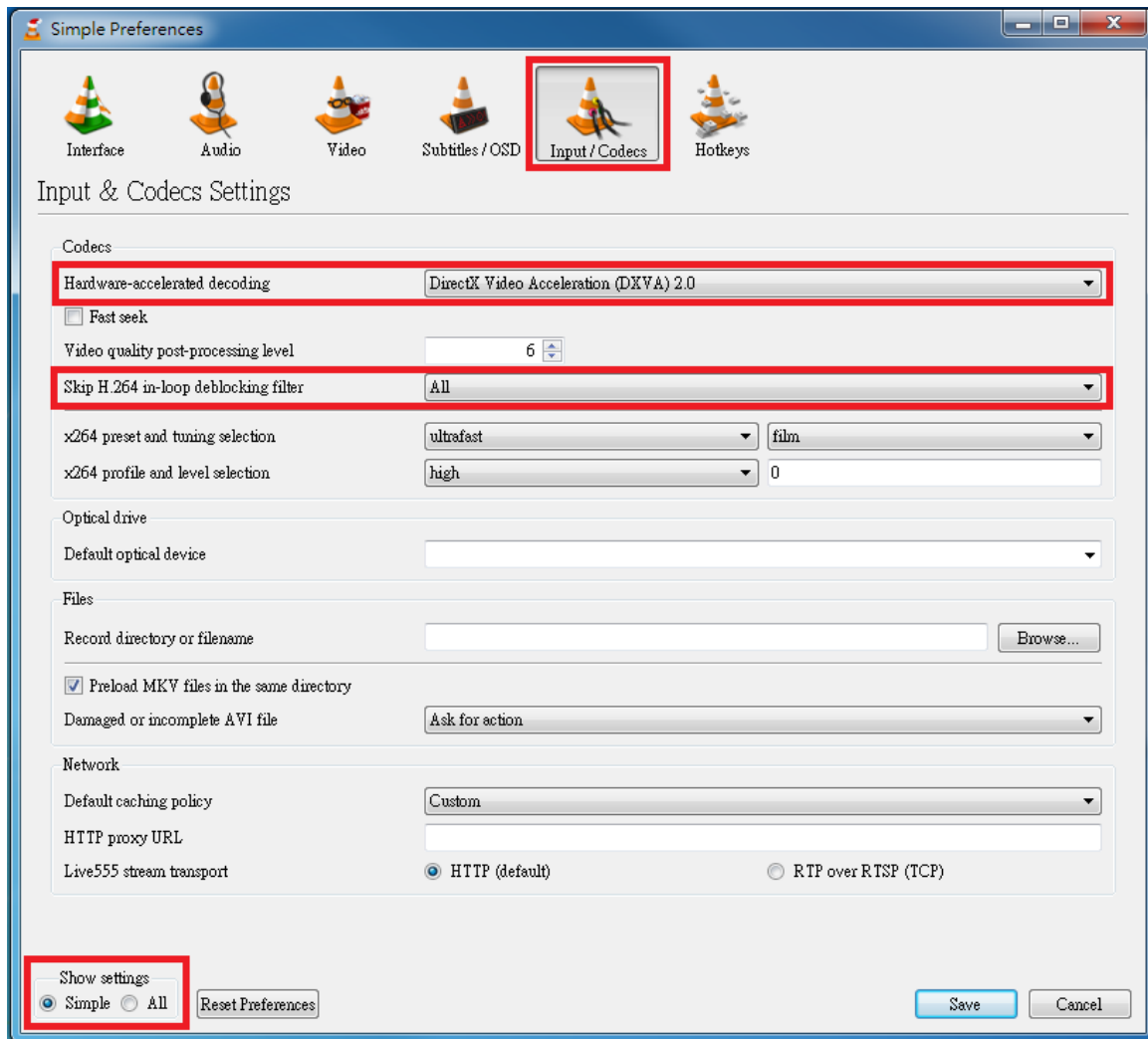
- Download VLC media player from website <https://www.videolan.org/>
- Adjust latency (buffer) related settings
  1. Click “Tools” -> “Preferences” -> “Show settings: All” (lower left corner) -> “Input/ Codecs”, (1) set “Network caching” to 300ms (recommended) , (2)set “Clock synchronisation” to Default, (3) set “Clock jitter” to 400ms (recommended) .

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



2. Click “Tools” -> “Preferences” -> “Show settings: Simple” (lower left corner) -> “Input/ Codecs”. Enable “Hardware-accelerated decoding” if available, and set “Skip H.264 in-loop deblocking filter” to “All”.





3. VLC have a pts\_delay buffer by "network buffer" and "clock jitter". The maximum value of this buffer is equal to "network buffer" plus "clock jitter". The video display on the VLC side will delay due to the increase of pts\_delay buffer. By reducing the "network cache" and "clock jitter" can achieve the effect of shortening the delay.

- Playing live streaming from AmebaPro to VLC player  
Please refer to section "[1.4.1 Choose the proper sample program](#)" -> Execution and testing.
- Streaming Audio from VLC player to AmebaPro  
Please refer to section "[1.4.1 Choose the proper sample program](#)" -> Execution and testing.

## **2     Video API**

Currently includes H264 and JPEG API.

### **2.1   H264 API**

#### **2.1.1   h264\_open**

Purpose	Create H264 encoder instance.
Function	<code>void *h264_open();</code>
Parameter	None
Return	If the return value is not NULL, it means that the correct Encoder pointer is returned.

#### **2.1.2   h264\_initial**

Purpose	Set H264 parameter.
Function	<code>int h264_initial(void *ctx, struct h264_parameter *h264_parm);</code>
Parameter	<code>void *ctx :</code> Encoder pointer. <code>h264_parameter *h264_parm :</code> It is necessary to set the width, length, GOP (Group of picture), BPS (Bit per second), FPS (Frame per second), and Rate Control mode. Rate Control : Currently supports CBR (Constant bit rate), VBR (Variable bit rate) and FIXQP (Fixed QP). The VBR mode can currently support setting the maximum and minimum QP. The FIXQP mode indicates that the minimum and maximum QP values are the same.
Return	0 : Success ; -1 : Failure

### 2.1.3 h264\_encode

Purpose	Compress a Frame
Function	<code>int h264_encode(void *ctx);</code>
Parameter	Need to set the Input buffer address data, including Y and UV from the ISP buffer. In addition, need to set the Output buffer address data, including the destination address and the buffer length.
Return	0 : successful The length of the compression success and the destination address can be obtained. Non-zero : Image compression failed.

### 2.1.4 h264\_release

Purpose	Release compressed resources.
Function	<code>int h264_release(void *ctx);</code>
Parameter	<code>void *ctx</code> points to the Encoder pointer
Return	0 : Success. Non-zero : failed to release resources

## 2.2 JPEG API

### 2.2.1 jpeg\_open

Purpose	Create a JPEG encoder instance.
Function	<code>void *jpeg_open();</code>
Parameter	None
Return	If the return value is not NULL, it means that the correct Encoder pointer is returned.

### 2.2.2 jpeg\_initial

**Purpose**

Set up JPEG parameters.

**Function**

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

**Parameter**

void \*ctx :  
encoder pointer  
struct jpeg\_parameter \*jpeg\_parm:  
Need to set the width, length and level (0~9). The higher the level,  
the better the picture quality.

**Return**

0 : Success  
-1 : Failed

### 2.2.3 jpeg\_encode

**Purpose**

Compress a Frame

**Function**

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

**Parameter**

Need to set the Input buffer address data, including Y and UV from the ISP  
buffer. In addition, need to set the Output buffer address data, including  
destination address and buffer length.

**Return**

0: successful  
The length of the compression success and the destination address can be  
obtained.  
Non-zero: Image compression failed.

### 2.2.4 jpeg\_release

**Purpose**

Free compressed resources.

**Function**

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

**Parameter**

void \*ctx points to the Encoder pointer

**Return**

0 : Success.  
Non-zero : Failed to release resources

## **3     ISP API**

### **3.1   video\_subsys\_init**

Purpose

Initialize the setting of video environment.

Function

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

Parameter

isp\_cfg\_t \*cfg pointer

Return

0 : Success  
-1 : Failed

### **3.2   isp\_stream\_create**

Purpose

Create isp stream

Function

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

Parameter

isp\_cfg\_t \*cfg pointer

Isp\_id specifies stream ID (0~2). Format currently only supports YUV420 SEMI PLANAR, length, width, FPS (30, 15, 10, 5, 1) and HW\_SLOT (hard compression BUFFER)

Return

Null failed, successfully return isp\_stream\_t pointer

### **3.3   isp\_stream\_destroy**

Purpose

Destroy isp stream

Function

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

Parameter

isp\_stream\_t\* stream pointer

Return

Return NULL

### **3.4 isp\_stream\_set\_complete\_callback**

Purpose	CALLBACK FUNCTION when ISP FRAME is registered.
Function	
Parameter	<code>isp_stream_t* isp_stream_destroy(isp_stream_t* stream);</code>
	<code>void (*cb) (void*)</code> User registered function <code>void*</code> arg the parameters required by the user
Return	Return NULL

### **3.5 isp\_stream\_apply**

Purpose	Fill in the settings to the ISP
Function	
Parameter	<code>void isp_stream_apply(isp_stream_t* stream);</code>
	<code>isp_stream_t*</code> stream pointer
Return	None

### **3.6 isp\_stream\_start**

Purpose	Start ISP to get FRAME
Function	
Parameter	<code>void isp_stream_start(isp_stream_t* stream);</code>
	<code>isp_stream_t*</code> stream pointer
Return	None

### **3.7 isp\_stream\_stop**

Purpose	Stop ISP
Function	
Parameter	<code>void isp_stream_stop(isp_stream_t* stream);</code>
	<code>isp_stream_t*</code> stream pointer
Return	None

### 3.8 isp\_stream\_poll

Purpose	Query if FRAME is completed
Function	
Parameter	Int isp_stream_poll(isp_stream_t* stream)
Return	isp_stream_t* stream pointer
	0 : Success
	-1 : Failed

### 3.9 isp\_handle\_buffer

Purpose	Manage ISP BUFFER
Function	
Parameter	void isp_handle_buffer(isp_stream_t* stream, isp_buf_t* buf, int mode);
	isp_stream_t* stream pointer
	isp_buf_t* buf can get ISP BUFFER ADDRESS
	Mode is divided into the following ways :
	<b>MODE_EXCHANGE</b>
	Brings in the next ISP BUFFER data and brings the information obtained by the current ISP
	<b>MODE_SNAPSHOT</b>
	Will raise a FRAME, but will not continue the next FRAME generation
	<b>MODE_SKIP</b>
	This FRAME skip
	<b>MODE_SETUP</b>
	Set ISP HARDWARE BUFFER
Return	None

### 3.10 isp\_set\_flip

Purpose	Set the image to flip left and right, and flip it upside down
Function	
Parameter	void isp_set_flip(int a_dValue); Usage: After video_subsys_init() is successfully initialized.
	int a_dValue: Flip the set value, the range is 0~3, each value is as follows
	0 : Original output image    1 : Flip left and right
	2 : Flip up and down        3 : Left and right and flipped up and down
Return	

None

### 3.11 isp\_get\_flip

Purpose

Get value of the isp flip

Function

Void isp\_get\_flip(int \*a\_pdValue);

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int \*a\_pdValue: Retrieves the set value of the flip, the range is 0 ~ 3, each value is as follows

0: original output image      1: flip left and right

2: flip up and down          3: Left and right and flipped up and down

Return

None

### 3.12 isp\_set\_brightness

Purpose

Set isp brightness

Function

void isp\_set\_brightness(int a\_dValue);

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int va\_dValue: The brightness value of the image

Rang: -64 to 64,

Adjustable precision: +-1

Return

None

### 3.13 isp\_get\_brightness

Purpose

Get the current brightness of the image

Function

void isp\_get\_brightness(int \*a\_pdValue);

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int \*a\_pdValue: Retrieves the current brightness value.

Range: -64 to 64

Return

None

### 3.14 isp\_set\_contrast

Purpose

Set image contrast value



**Function**

Void isp\_set\_contrast(int a\_dValue);  
Usage: After video\_subsys\_init() is successfully initialized.

**Parameter**

Int a\_dValue: image contrast value.  
Range: 0~100.  
Adjustable precision is +-1

**Return**

None

**3.15 isp\_get\_contrast****Purpose**

Get the current contrast value of isp

**Function**

Void isp\_get\_contrast(int \*a\_pdValue);  
Usage: After video\_subsys\_init() is successfully initialized.

**Parameter**

int \*a\_pdValue: Get the current contrast value  
Range: 0~100

**Return**

None

**3.16 isp\_set\_saturation****Purpose**

Set isp saturation

**Function**

void isp\_set\_saturation(int a\_dValue);  
Usage: After video\_subsys\_init() is successfully initialized.

**Parameter**

int a\_dValue: ISP saturation.  
Range: 0 to 100.  
Adjustable accuracy: +-1

**Return**

None

**3.17 isp\_get\_saturation****Purpose**

Get the current saturation of isp

**Function**

void isp\_get\_saturation(int \*a\_pdValue);  
Usage: After video\_subsys\_init() is successfully initialized.

**Parameter**

int \*a\_pdValue: Get the current saturation  
Range: 0 to 100

Return  
None

### 3.18 isp\_set\_sharpness

Purpose  
Set isp sharpness

Function  
`void isp_set_sharpness(int a_dValue);`  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter  
int a\_dValue: Sharpness of isp  
Range: 0~100.  
Adjustable precision: +-1

Return  
None

### 3.19 isp\_get\_sharpness

Purpose  
Get the current sharpness of isp

Function  
`void isp_get_sharpness(int *a_pdValue);`  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter  
int \*a\_pdValue: Retrieve the current sharp value from 0 to 100

Return  
None

### 3.20 isp\_set\_gamma

Purpose  
Set the Gamma coefficient

Function  
`void isp_set_gamma(int a_dValue);`  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter  
int a\_dValue: Gamma coefficient.  
Range: 100~500.  
Adjustable precision: +-1

Return  
None

### 3.21 isp\_get\_gamma

Purpose

	Get the Gamma coefficient
Function	<code>void isp_get_gamma(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	
	<code>int *a_pdValue</code> : Retrieve the current Gamma coefficient from 100~500
Return	None

### 3.22 isp\_set\_gray\_mode

Purpose	Set the gray/color mode
Function	<code>void isp_set_gray_mode(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	
	<code>int a_dValue</code> : The value of gray/color mode. 0: color mode, 1: gray mode
Return	None

### 3.23 isp\_get\_gray\_mode

Purpose	Get the gray/color mode
Function	<code>void isp_get_gray_mode(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	
	<code>int *a_pdValue</code> : Retrieve the value of gray/color mode, 0: color mode , 1: gray mode
Return	None

### 3.24 isp\_set\_exposure\_mode

Purpose	Set the mode of auto/manual exposure
Function	<code>void isp_set_exposure_mode(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	
	<code>int a_dValue</code> : The mode of exposure, value is 1 or 8. (1: manual, 8: Auto)
Return	None

### 3.25 isp\_get\_exposure\_mode

Purpose	Get the mode of auto/manual exposure
Function	<code>void isp_get_exposure_mode(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int *a_pdValue:</code> Retrieve the mode of exposure, value is 1 or 8, (1: manual, 8: auto)
Return	None

### 3.26 isp\_set\_exposure\_time

Purpose	Based on manual exposure mode, set the exposure time.
Function	<code>void isp_set_exposure_time(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValue:</code> The exposure time, unit is us, range is 1~1,000,000, the adjustable precision is +-1
Return	None

### 3.27 isp\_get\_exposure\_time

Purpose	Get the exposure time
Function	<code>void isp_get_exposure_time(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int *a_pdValue:</code> Retrieve the exposure time, unit is us, the range is 1~1,000,000
Return	None

### 3.28 isp\_set\_zoom

Purpose	Set zoom index.
Function	<code>void isp_set_zoom(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValue:</code> The zoom index.

Range: 0~3 (0: 1.0x, 1: 1.28X, 2: 1.6X, 3: 2.0X)

Adjustable precision: +-1

Note: Since width is limited between 64~640 (exclude 640), the resolution larger than 640X480 is not supported.

Return

None

### 3.29 isp\_get\_zoom

Purpose

Get zoom index

Function

`void isp_get_zoom(int *a_pdValue);`

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int \*a\_pdValue: Retrieve the zoom index, range is 0~3

Return

None

### 3.30 isp\_set\_pan\_tilt

Purpose

Set the shift distance of pan-direction and tilt-direction

Function

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

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int a\_dValuePan: The shift distance of pan-direction.

Range: -576000~57600.

Adjustable precision: +-3600

int a\_dValueTilt: (Tilt is not supported)

Note: Only resolution smaller than 640X480 is supported. (include 640x480)

Return

None

### 3.31 isp\_get\_pan\_tilt

Purpose

Get the shift distance of pan-direction and tilt-direction

Function

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

Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int \*a\_pdValuePan: Retrieve shift distance of pan-direction

Range: -576000~57600

int \*a\_pdValueTilt: (Tilt is not supported)

Return

None

### 3.32 isp\_set\_AWB\_ctrl

Purpose

Set AWB mode

Function

void isp\_set\_AWB\_ctrl(int a\_dValue);  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int a\_dValue: Mode of white balance  
0: Manual temperature, 1: Auto  
Note: The API of manual temperature is not supported.

Return

None

### 3.33 isp\_get\_AWB\_ctrl

Purpose

Get AWB mode

Function

void isp\_get\_AWB\_ctrl(int \*a\_pdValue);  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int \*a\_pdValue: Retrieve the mode of white balance  
0: Manual, 1: Auto

Return

None

### 3.34 isp\_set\_power\_line\_freq

Purpose

Set the mode of Anti-flicker.

Function

void isp\_set\_power\_line\_freq(int a\_dValue);  
Usage: After video\_subsys\_init() is successfully initialized.

Parameter

int a\_dValue: Anti-flicker mode.  
Range: 0 ~ 3  
0: Disable, 1: 50Hz, 2: 60Hz, 3: Auto

Return

None

### 3.35 isp\_get\_power\_line\_freq

Purpose

	Get the mode of Anti-flicker.
Function	<code>void isp_get_power_line_freq(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int *a_pdValue</code> : Retrieve the mode of Anti-flicker. Range: 0 ~ 3
Return	None

### 3.36 isp\_set\_AE\_gain

Purpose	Set gain value.
Function	<code>void isp_set_AE_gain(int a_dValueAnalogGain, int a_dValueDigitalGain, int a_dValueISPDigitalGain);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValueAnalogGain</code> : Gain value in sensor. Range: 256~4080. Adjustable precision: +-16 <code>int a_dValueDigitalGain</code> : Default is 256 and cannot be adjustable <code>int a_dValueISPDigitalGain</code> : Gain value in ISP. Range: 0~4095. Adjustable precision: +-1  Note: Analog gain is the gain inside sensor; ISPDigitalGain is the gain inside ISP
Return	None

### 3.37 isp\_get\_AE\_gain

Purpose	Get gain value
Function	<code>void isp_get_AE_gain(int *a_pdValueAnalogGain, int *a_pdValueDigitalGain, int *a_pdValueISPDigitalGain);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValueAnalogGain</code> : Retrieve gain value in sensor. Range: 256~4080 <code>int a_dValueDigitalGain</code> : Default value: 256 <code>int a_dValueISPDigitalGain</code> : Retrieve gain value in ISP: 0~4095
Return	None

### 3.38 isp\_set\_WDR\_mode

Purpose	Set WDR mode
Function	<code>void isp_set_WDR_mode(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValue</code> : WDR mode. Range: 0 ~ 4 0: Disable, 1: Manual, 2: Auto(weak), 3: Auto(medium), 4: Auto(strong)
Return	None

### 3.39 isp\_get\_WDR\_mode

Purpose	Get WDR mode
Function	<code>void isp_get_WDR_mode(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int *a_pdValue</code> : Retrieve the value of WDR mode. Range: 0 ~ 4
Return	None

### 3.40 isp\_set\_WDR\_level

Purpose	Based on WDR manual mode, set WDR level
Function	<code>void isp_set_WDR_level(int a_dValue);</code> Usage: After video_subsys_init() is successfully initialized.
Parameter	<code>int a_dValue</code> : WDR level. Range: 0~100. Adjustable precision: +-1
Return	None

### 3.41 isp\_get\_WDR\_level

Purpose	Get WDR level
Function	<code>void isp_get_WDR_level(int *a_pdValue);</code> Usage: After video_subsys_init() is successfully initialized.



Parameter  
int \*a\_pdValue: Retrieve the value of WDR level.  
Range: 0~100.

Return  
None

### **3.42 isp\_stream\_power\_off\_config**

Purpose  
Set the SENSOR POWER OFF mode

Function  
int isp\_stream\_power\_off\_config(unsigned char enable);

Parameter  
Unsigned char enable :  
0: After all the ISP stream is closed, the SENSOR will not be closed. It will save the ISP switch time. Pay attention to the last time to enter the power saving mode. You need to set it to 1 to turn off the sensor.  
1: SENSOR will be turned off after all the ISP stream is turned off. This is the default value.

Return  
0: Success -1: Failed, ISP is not initialized

### **3.43 isp\_check\_boot\_status**

Purpose  
Get isp boot mode status

Function  
void isp\_get\_WDR\_level(int \*a\_pdValue);

Parameter  
None

Return  
0: normal boot 1: fast boot

## **4**    **OSD**

### **4.1**   **OSD introduction**

rtstream provides a set of API functions to set the OSD configuration of the data flow stream, etc. Note that when calling this set of APIs, you need to create a data flow before calling. The group interface is as follows:

rts\_video\_query\_osd\_attr interface gets the video osd attribute;

rts\_video\_set\_osd\_attr interface sets the video osd attribute;

rts\_video\_release\_osd\_attr interface releases the attributes obtained by

rts\_video\_query\_osd\_attr;

In addition, a set of simple interfaces for setting up a video OSD is provided. This set of interfaces does not require the user to create an additional data flow. The group interface is as follows:

rts\_query\_isp\_osd\_attr interface gets the video osd attribute;

rts\_set\_isp\_osd\_attr interface sets the video osd attribute;

rts\_release\_isp\_osd\_attr interface releases the attributes obtained by

rts\_query\_isp\_osd\_attr;

### **4.2**   **OSD example**

The sample program is located at:

component\common\example\isp\example\_isp\_osd\_multi.c

Must set platform\_opts.h before use.

Open project\realtek\_amebapro\_v0\_example\inc\platform\_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
  
0
```

Modify CONFIG\_EXAMPLE\_MEDIA\_UVCD from 0 to 1.

Modify CONFIG\_EXAMPLE\_ISP\_OSD\_MULTI from 0 to 1, compile and execute.

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
  
1
```

#### **Execution and testing**

- Connect the USB cable to the AmebaPro CON port and the other end to the PC.
- Open potplayer, amebaPro atcmd enter "ATIO" will show result.

### 4.3 OSD Show Time information

The time displayed by the OSD is based on SNTP. The time is obtained by the “sntp\_gen\_system\_time” function. Therefore, the timezone needs to be set by the global variable rtsTimezone.

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

## 4.4 OSD API

### 4.4.1 rts\_video\_query\_osd\_attr

Purpose

Get video stream osd attribute.

Function

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

Parameter

**stream** Input parameters, RtStream pointer.

**attr** The output parameter, which points to the address of the variable storing osd attr, needs to be called rts\_video\_release\_osd\_attr to release.

Retrun

Return 0 means success, returning negative means failure。

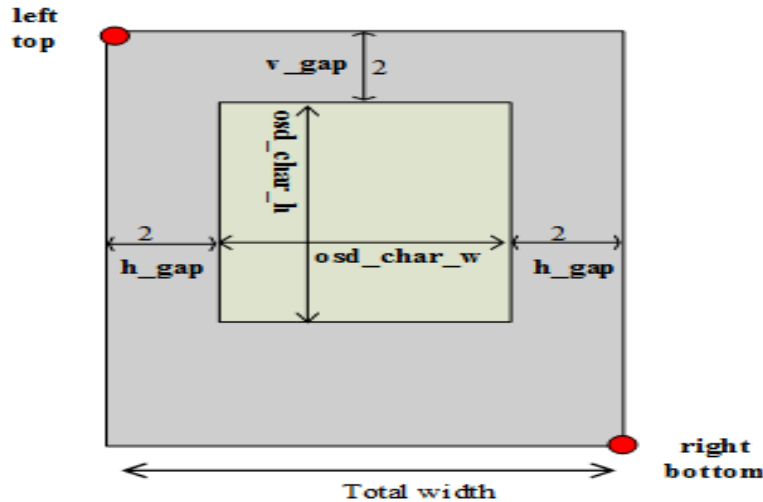
Description

Each video stream has a separate osd module, and each osd module is represented by the structure rts\_video\_osd\_attr. Each osd module supports up to 6 blocks, which is an area in the video for displaying characters or images, represented by the structure rts\_video\_osd\_block. The width of a word is inconsistent between English and Chinese in digital display. The English and array are single-wide, and the font file is stored in the single-wide font file. The Chinese display takes up double width and the font file is stored in the double-width font file.

The single\_lib\_name and double\_lib\_name in rts\_video\_osd\_attr are used to save the file names of each glyph file. The picture in osd is represented by a pbitmap in the block which is a pointer to the BITMAP\_S structure.

```
struct rts_video_osd_attr {
    int number; // the number of blocks in osd
    struct rts_video_osd_block *blocks;
    enum rts_osd_time_fmt time_fmt; // displayed time format
    uint8_t time_blkidx; // displayed time block index
    int time_pos; // Time display position
    enum rts_osd_date_fmt date_fmt; // displayed date format
    uint8_t date_blkidx; // date of the block of the date displayed
    int date_pos; // date display position
    char *single_lib_name; // single font file name
    char *double_lib_name; // double font file name
```

```
struct rts_video_osd_block {
    struct rts_video_rect rect; // block coordinates
    uint8_t bg_enable; // background enable
    uint32_t bg_color; // background color
    uint32_t ch_color; // character color
    uint8_t h_gap:4,v_gap:4; // Interval between characters and characters
    uint8_t flick_enable; // character flick enable
    uint32_t flick_speed; //flick speed , flick every 2^flick_speed
    uint8_t char_color_alpha; // character semi-transparent
    uint8_t stroke_enable; // character stroke switch
    uint8_t stroke_direct; // the direction of the character stroke,
                                0: minus the increment, 1: plus the
                                increment
    uint8_t stroke_delta; // character stroke increment
```



The meaning of the field in the osd structure is shown in the figure. The value of the interval between the OSD character and the character is configured by the user. The minimum value is 2 and the maximum value is 15. Interval between the character level directions is `h_gap`, the vertical interval is `v_gap`.

rts osd time fmt	顯示樣式	例子
osd_time_fmt_no	Not display time	Not display time
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	Not display date	Not display date
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

Purpose

Set the osd property of the video stream.

Function

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

Parameter

*stream*

Input parameters, RtStream pointer.

*attr*

Input parameters, points to the osd attribute, obtained by rts\_video\_query\_osd\_attr. Structure rts\_video\_osd\_attr is defined in rts\_video\_query\_osd\_attr

Retrun

Return 0 indicates success and a negative error code indicates failure.

None

#### 4.4.3 rts\_video\_release\_osd\_attr

Purpose

Release the osd attribute of the video stream.

Function

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

Parameter

*stream*

Input parameters, RtStream pointer.

*attr*

Input parameter which point to the osd attribute is obtained by rts\_video\_query\_osd\_attr. The structure rts\_video\_osd\_attr is defined in rts\_video\_query\_osd\_attr

Retrun

None

**Description**

This function is used to release the osd attr obtained by `rts_video_query_osd_attr`, otherwise a memory leak will occur.

**4.4.4 `rts_query_isp_osd_attr`****Purpose**

Get the video osd attribute.

**Function**

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

**Parameter**

*isp\_id*

Input parameters, isp supports simultaneous output of multiple channels, each channel can create an isp stream, where id is the index of a certain path isp, starting from 0.

*attr*

The output parameter, which points to the address of the variable storing osd attr, needs to be called `rts_release_isp_osd_attr` to release.

**Retrun**

Return 0 indicates success ,return a negative value indicates failure.

**Description**

Each video stream has a separate osd module, and each osd module is represented by the structure `rts_video_osd_attr`. Each osd module supports up to 6 blocks, a block is an area in the image for displaying characters or images, which represented by the structure `rts_video_osd_block`. English and digital width of a word are inconsistent with Chinese in display. English and array use a single, the width and font files are saved in the single font file. The Chinese display takes up double width, and the font file is saved in the double wide font file. If you want to display image information such as logo or QR code, you can save the image in the image file, like a glyph file. The `single_lib_name`, `double_lib_name`, and `picture_lib_name` in `rts_video_osd_attr` are used to save the file names of each glyph file.

#### 4.4.5 **rts\_set\_isp\_osd\_attr**

Purpose

Set the video osd property.

Function

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

Parameter

*attr*

Input parameter which points to the osd attribute is obtained by `rts_query_isp_osd_attr`. The structure `rts_video_osd_attr` is defined in `rts_video_query_osd_attr`

Retrun

Return 0 indicates success and a negative value indicates failure.

Description

None

#### 4.4.6 **rts\_release\_isp\_osd\_attr**

Purpose

Release the video osd attribute.

Function

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

Parameter

*attr*

Input parameter which points to the osd attribute is obtained by `rts_query_isp_osd_attr`. The structure `rts_video_osd_attr` is defined in `rts_video_query_osd_attr`

Retrun

None

Description

This function is used to release the osd attr obtained by `rts_query_isp_osd_attr`, otherwise a memory leak will occur.



## 5 Motion Detect

### 5.1 Motion Detect introduction

rtstream provides a set of API functions to set the configuration of the motion detection of the data flow stream. Note that when calling this set of APIs, you need to create a data flow before calling.

The group interface is as follows:

rts\_video\_query\_md\_attr interface gets the motion detect attribute supported by the isp;

rts\_video\_set\_md\_attr interface setting update motion detect;

rts\_video\_release\_md\_attr interface releases the attributes obtained by

rts\_video\_query\_md\_attr;

rts\_video\_check\_md\_status interface checks if a motion detect is detected.

In addition, a set of simple interfaces for setting motion detection is provided. This set of interfaces does not require the user to additionally create a corresponding data flow.

The group interface is as follows:

rts\_query\_isp\_md\_attr interface gets the motion detect attribute supported by the isp;

rts\_set\_isp\_md\_attr interface setting update motion detect;

rts\_release\_isp\_md\_attr interface releases the attributes obtained by

rts\_query\_isp\_md\_attr;

rts\_check\_isp\_md\_status interface checks if a motion detect is detected.

### 5.2 Motion Detect example

The sample program is located at: component\common\example\isp\example\_md.c

Must set platform\_opts.h before use.

Open project\realtek\_amebapro\_v0\_example\inc\platform\_opts.h

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
  
0
```

Modify CONFIG\_EXAMPLE\_MEDIA\_UVCD from 0 to 1.

Modify CONFIG\_EXAMPLE\_MOTION\_DETECT from 0 to 1, compile and execute.

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
  
1
```

#### **Execution and testing**

- Connect the USB cable to the AmebaPro CON port and the other end to the PC.
- Open Tera Term show log, amebaPro atcmd enter “ATID” will show result.

## 5.3 Motion Detect API

### 5.3.1 rts\_video\_query\_md\_attr

Purpose

Get video stream motion detect attribute.

Function

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

Parameter

**stream** Input parameters, RtStream pointer.

**attr** The output parameter, which points to the address of the variable storing motion detect attr, needs to be called rts\_video\_release\_md\_attr to release.

Retrun

Return 0 means success, returning negative means failure。

Description

```
Struct rts_video_md_attr {  
    Int number; //the number of blocks of motion detect  
    Struct rts_video_md_block *blocks; //index pointing to blocks  
    Uint32_t reserved[4];  
};  
  
Struct rts_video_md_block {  
    Int enable; //Enable switch  
    Struct rts_video_grid area;  
    Uint32_t sensitivity; //sensitivity, 0~100
```

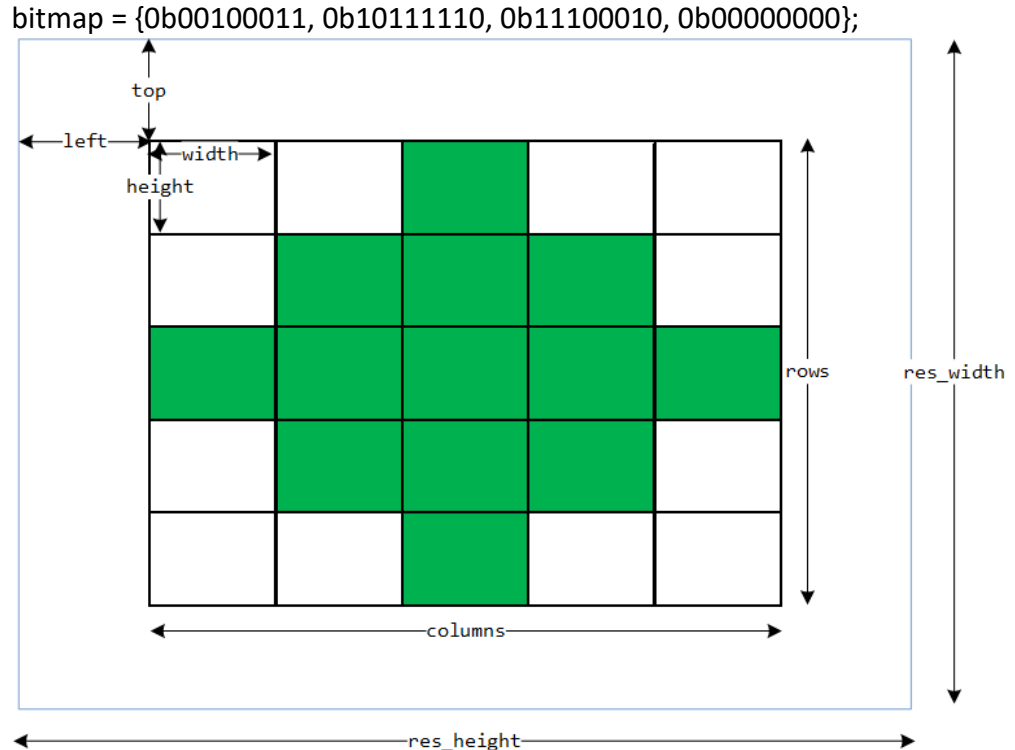
When doing motion detect analysis, it can be analyzed frame by frame or interval analysis. The number of separated frames can be configured by `frame_interval`. If the number of interval frames is small, md is easier to detect high-speed motion, and it is not easy to detect low-speed motion; and if the number of interval frames is large, more differences can be accumulated, which makes it easier to detect slow motion. Depending on the application scenario, you can change the threshold for detecting motion by configuring the sensitivity and percentage. The greater the sensitivity, the more sensitive, the lower the threshold, the easier it is to detect motion. The smaller the percentage, the lower the threshold and the easier it is to detect motion.

```
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];
    // ...
};
```

The variables in the `rts_video_grid` structure are shown in `rts_video_grid`. Each bit of the bitmap represents a cell of the grid, with 0 being disabled and 1 being able to be enabled. In the picture

`rows = 5`  
`columns = 5`  
`length = columns * rows = 5 * 5 = 25`



### 5.3.2 rts\_video\_set\_md\_attr

Purpose

Set the motion detect property of the video stream.

Function

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

Parameter

**stream** Input parameters, RtStream pointer.

**attr** The input parameter, the indicator pointing to md attr, is obtained by rts\_video\_query\_md\_attr and needs to be called by rts\_video\_release\_md\_attr. The structure rts\_video\_md\_attr is defined in rts\_video\_md\_attr.

Retrun

Return 0 means success, returning negative means failure。

Description

None

### 5.3.3 **rts\_video\_release\_md\_attr**

Purpose

Release the motion detect attribute of the video stream.

Function

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

Parameter

**stream** Input parameters, RtStream pointer.

**attr** Input parameters, metrics pointing to md attr, obtained by  
rts\_video\_query\_md\_attr. The structure rts\_video\_md\_attr is  
defined in rts\_video\_md\_attr

Retrun

None

Description

This function is used to release the md attr obtained by  
rts\_video\_query\_md\_attr, otherwise a memory leak will occur.

### 5.3.4 **rts\_video\_check\_md\_status**

Purpose

Check the motion detect status of the video stream to see if motion is  
detected.

Function

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

Parameter

**stream** Input parameter, RtStream pointer.

**mdidx** Input parameter, index of motion detect block

Retrun

Return 1 means motion is detected and a return of 0 means no detected.

Description

None

### 5.3.5 **rts\_video\_get\_md\_result**

Purpose

Obtain a bitmap of motion detection.

Function

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

Parameter

**stream** Input parameters, RtStream pointer.

**mdidx** Input parameter, the index of the motion detect block, currently supports the number of MD blocks for RTS3901 & RTS3902, so set it to 0 for RTS3901&RTS3902.

**result** Output parameter, pointing to the metric of rts\_video\_grid\_bitmap, which contains information about the MD grid bitmap.

Retrun

Return 0 means success, returning negative means failure。

Description

A return of 1 means motion is detected and a return of 0 means no detection.

```
Struct rts_video_grid_bitmap {
    Uint16_t number; //the number of grids uint8_t
    Bitmap[RTS_GRID_BITMAP_SIZE]; // bitmap of all grids
```

### 5.3.6 rts\_query\_isp\_md\_attr

Purpose

Get the motion detect property of the video stream.

Function

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

Parameter

**attr** output parameter, which points to md attr, needs to be called by rts\_video\_release\_md\_attr. See rts\_video\_md\_attr for the definition of the structure rts\_video\_md\_attr.

**Res\_width** Input parameter, the resolution width, and the position in rts\_video\_md\_attr are relative to the resolution.

**Res\_height** Input parameter, the resolution height, and the position in rts\_video\_md\_attr are relative to the resolution.

Retrun

Return 0 means success, returning negative means failure。

Description

The difference with rts\_video\_query\_md\_attr is that rts\_query\_isp\_md\_attr is independent of stream and does not need to provide Rt-Stream metric parameters.

### 5.3.7 rts\_set\_isp\_md\_attr

Purpose

Set the motion detect property of the video stream.

**Function**

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

**Parameter**

**attr** input parameter, the indicator pointing to md attr, is obtained by rts\_query\_isp\_md\_attr and needs to be called by rts\_release\_isp\_md\_attr. The structure rts\_video\_md\_attr is defined in rts\_video\_md\_attr.

**Retrun****Description**

Return 0 means success, returning negative means failure.

The difference with rts\_video\_set\_md\_attr is that rts\_set\_isp\_md\_attr is independent of stream and does not need to provide RtStream indicator parameters.

### 5.3.8 rts\_check\_isp\_md\_status

**Purpose**

Check the motion detect status of the video stream to see if motion is detected.

**Function**

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

**Parameter**

**mdidx** input parameter, the index of the motion detect block, the number of MD blocks supported by RTS3901&RTS3902 is 1, so the fixed setting is 0 for RTS3901&RTS3902.

**Retrun**

Return 1 means motion is detected and a return of 0 means no detected.

**Description**

The difference with rts\_video\_check\_md\_status is that rts\_check\_isp\_md\_status is independent of stream and does not need to provide RtStream metric parameters.

### 5.3.9 rts\_get\_isp\_md\_result

**Purpose**

Obtain a bitmap of motion detection.

**Function**

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

**Parameter**

**mdidx** input parameter, the index of the motion detect block, the number of MD blocks supported by RTS3901&RTS3902 is 1, so the fixed setting is 0 for RTS3901&RTS3902.

**result** output parameter, which points to the rts\_video\_grid\_bitmap

metric, which contains information about the MD grid bitmap.

Retrun

Return 0 means success, returning negative means failure.

Description

The difference with `rts_video_check_md_status` is that `rts_check_isp_md_status` is independent of stream and does not need to provide `RtStream` metric parameters.

## 6 Mask

### 6.1 Mask introduction

`rtstream` provides a set of API functions to set the video mask configuration of the data flow stream. Note that when calling this set of APIs, you need to create a data flow before calling. The group interface is as follows:

The `rts_video_query_mask_attr` interface gets the video mask attribute;

The `rts_video_set_mask_attr` interface sets the video mask attribute;

The `rts_video_release_mask_attr` interface releases the attributes obtained by `rts_video_query_mask_attr`;

In addition, a set of simple interfaces for setting a video mask is provided. This set of interfaces does not require the user to additionally create a corresponding data flow.

The group interface is as follows:

The `rts_query_isp_mask_attr` interface gets the video mask attribute;

The `rts_set_isp_mask_attr` interface sets the video mask attribute;

The `rts_release_isp_mask_attr` interface releases the attributes obtained by `rts_video_query_mask_attr`;

### 6.2 Mask example

The sample program is located at: `component\common\example\isp\example_mask.c`  
Must set `platform_opts.h` before use.

Open `project\realtek_amebapro_v0_example\inc\platform_opts.h`

```
#define CONFIG_EXAMPLE_MEDIA_UVCD  
0
```

Modify `CONFIG_EXAMPLE_MEDIA_UVCD` from 0 to 1.

Modify `CONFIG_EXAMPLE_MASK` from 0 to 1, compile and execute.



```
#define CONFIG_EXAMPLE_MEDIA_UVCD
1
```

### **Execution and testing**

- Connect the USB cable to the AmebaPro CON port and the other end to the PC.
- Open potplayer, amebaPro atcmd enter “ATIM” will show result.

## **6.3 Mask API**

### **6.3.1 rts\_video\_query\_mask\_attr**

#### **Purpose**

Get video stream mask attribute.

#### **Function**

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

#### **Parameter**

**stream** Input parameters, RtStream pointer.

**attr** Output parameter, which points to the address of the variable that holds the private mask attr, needs to be called by rts\_video\_release\_mask\_attr.

#### **Retrun**

Return 0 means success, returning negative means failure。

#### **Description**

The structure rts\_video\_grid see the definition of rts\_video\_grid at rts\_video\_query\_md\_attr.

Rtstream supports a total of 5 mask areas, including 1 grid and 4 rect.

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

### 6.3.2 **rts\_video\_set\_mask\_attr**

Purpose

Set the mask attribute of the video stream.

Function

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

Parameter

***stream***

Input parameters, RtStream pointer.

***attr***

Input parameter, the indicator pointing to mask attr, is obtained by rts\_video\_query\_mask\_attr and needs to be called by rts\_video\_release\_mask\_attr. The structure rts\_video\_mask\_attr is defined in rts\_video\_mask\_attr.

Retrun

Return 0 indicates success and a negative error code indicates failure.

Description

None

### 6.3.3 **rts\_video\_release\_mask\_attr**

Purpose

Release the mask attribute of the video stream.

Function

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

Parameter

***stream***

Input parameters, RtStream pointer.

***attr***

Input parameters, pointers to mask attr, obtained by rts\_video\_query\_mask\_attr. Structure rts\_video\_mask\_attr See rts\_video\_mask\_attr for definitions.

Retrun

None

Description

This function is used to release the mask attr obtained by rts\_video\_query\_mask\_attr, otherwise a memory leak will occur.

### 6.3.4 **rts\_query\_isp\_mask\_attr**

Purpose

Get the video mask attribute.

**Function**

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

**Parameter*****attr***

Output parameter, which points to the address of the variable that holds the private mask attr, needs to be called by rts\_release\_ismask\_attr.

***res\_width***

Input parameter, the resolution width, and the position in rts\_video\_mask\_attr are relative to the resolution.

***res\_height***

Input parameter, the resolution height, and the position in rts\_video\_mask\_attr are relative to the resolution.

**Retrun**

Return 0 indicates success ,return a negative value indicates failure.

**Description**

Structure rts\_video\_mask\_attr See rts\_video\_query\_mask\_attr

```
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_ismask_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\_ismask\_attr**

#### Purpose

Set the mask attribute of the video stream.

#### Function

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

#### Parameter

##### **attr**

Output parameter, which points to the address of the variable that holds the private mask attr, needs to be called by `rts_release_ismask_attr`.

##### **res\_width**

Input parameter, the resolution width, and the position in `rts_video_mask_attr` are relative to the resolution.

##### **res\_height**

Input parameter, the resolution height, and the position in `rts_video_mask_attr` are relative to the resolution.

#### Retrun

Return 0 indicates success and a negative value indicates failure.

**Description**

Structure `rts_video_mask_attr` reference to `rts_video_query_mask_attr`

**6.3.6 rts\_release\_isp\_mask\_attr****Purpose**

Release the mask attribute of the video stream.

**Function**

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

**Parameter*****attr***

Input parameter, which points to the mask attr, is obtained by `rts_query_isp_mask_attr`. Structure `rts_video_mask_attr`  
See `rts_video_mask_attr` for definitions.

**Retrun**

None

**Description**

This function is used to release the mask attr obtained by `rts_query_isp_mask_attr`, otherwise a memory leak will occur.

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.