Rockchip Rkmedia Instructions

File ID: RK-SM-YF-365

Release version: V0.0.1

Date: 2020-06-08

File Confidentiality: □ Top Secret □ Secret □ Internal Information ■ Public

Rockchip Microelectronics Co., Ltd.

Rockchip Electronics Co., Ltd.

Address: No. 18, Area A, Software Park, Tongpan Road, Fuzhou City, Fujian Province

URL: www.rock-chips.com

Customer Service Tel: +86-4007-700-590

Customer Service Fax: +86-591-83951833

Customer Service Email: fae@rock-chips.com

1. Introduction

Overview

This article mainly describes the usage instructions of each module of rkmedia application.

product version

Chip name	Kernel version
RV1109	Linux 4.19
RV1126	Linux 4.19
RK1808	Linux 4.4
RK1806	Linux 4.4

Audience

This document (this guide) is mainly applicable to the following engineers:

- Technical Support Engineer
- Software Development Engineer

Revision record

date	version	Author	Modify the description
2020-06-08	V0.0.1	Fan Lichuang	initial version

2. Table of Contents

3.1 Introduction

rkmedia is the multimedia library of the RockChip Linux platform, which encapsulates the interface of the underlying media module and provides common multimedia solutions.

Currently includes video hardware encoding and decoding interfaces, media format encapsulation and decapsulation interfaces, audio software encoding and decoding interfaces, audio capture, output a
Interface, camera acquisition interface, etc.

4.2 Video hardware encoding

Example: **mpp_enc_test.cc**

Use the command to view the usage method: ./rkmpp_enc_test -? (Maybe there is no executable bin in the firmware generated by default, it needs to be generated on the pc
Manually push the path to the end of the board).

Interface and example process description

easymedia::REFLECTOR(Encoder)::DumpFactories(): List the currently programmed encoding modules (may not be called)

easymedia::REFLECTOR(Encoder)::Create<easymedia::VideoEncoder> ：Create a video encoder instance, parameters
The string corresponding to a module listed in the above DumpFactories, and the corresponding output data type

InitConfig: Initialize the encoder, the parameter is the setting coefficient corresponding to the required encoding algorithm

GetExtraData: Get parameter information data, the pps and sps data of h264 are in the buffer returned here

Process: Perform encoding, the parameters are the original uncompressed image data buffer, the compressed image output buffer, and additional output
buffer (if mv data in h264 is needed, output in this buffer)

Note: Before calling this function, you need to give all buffers SetValidSize to indicate the length space that the buffer can access. Finally output buffer
The data length is reflected by GetValidSize.

5.3 Video hardware decoding

Example: **mpp_dec_test.cc**

Use the command to view the usage: ./rkmpp_dec_test -? (Maybe there is no executable bin in the firmware generated by default, and it needs to be generated on the pc Manually push the path to the end of the board).

Interface and example process description

easymedia::REFLECTOR(Decoder)::DumpFactories(): List the currently-encoded decoding modules (may not be called)

easymedia::REFLECTOR(Decoder)::Create<easymedia::VideoDecoder> ：Create a video encoder instance, parameters
For the string corresponding to a module listed in the above DumpFactories, as well as some other setting parameters, please refer to
Notes in mpp_dec_test.cc
SendInput: Send the compressed image data to the decoder, and also requires SetValidSize to indicate the data length. If the function return value returns
Back to -EAGAIN, it means that this frame of data has not been accepted by the decoder, and you will try to input again after waiting. After the last frame, you need to send a
The empty buffer of EOF is given to the decoder.
FetchOutput: Used in conjunction with SendInput to take out the decoded raw format data from the decoder. Function error takes the value of errno
reflect.

Page 5

Process: Synchronous decoding, currently only jpeg decoding is supported, the parameters are compressed image data buffer, raw format data output
imagebuffer (space must be allocated)

Note: Before calling this function, you need to give all buffers SetValidSize to indicate the length space that the buffer can access. Finally output buffer
The data length is reflected by GetValidSize.

Page 6

6.4 Media format decapsulation

Note: Currently only ogg is supported, other formats are to be expanded

Example: **ogg_decode_test.cc**

Use the command to view the usage method: ./ogg_decode_test -? (Maybe there is no executable bin in the firmware generated by default, and it needs to be generated on the PC Manually push the completed path to the board end).

Interface and example process description

easymedia::REFLECTOR(Demuxer)::DumpFactories(): List the currently compiled format decapsulation modules (may not be called)
easymedia::REFLECTOR(Demuxer)::Create<easymedia::Demuxer>: create an example of format decapsulation, the parameters are above
The string corresponding to a module listed in DumpFactories and some other setting items
Init(Stream *input, MediaConfig *out_cfg): Set input stream and get audio parameters
Read: Read the data once

7.5 Audio playback output

Example: **ogg_decode_test.cc**

The same example as the decapsulation example of the reused media format
Use the command: ./ogg_decode_test -i test.ogg -o alsa:default

Interface and example process description

easymedia::REFLECTOR(Stream)::DumpFactories(): List the input and output modules currently compiled into the stream
The parameters are similar to c's FILE (can not be called)
easymedia::REFLECTOR(Stream)::Create<easymedia::Stream>: Create an audio playback output stream instance, the parameter is a word
String "alsa_playback_stream" and set the parameters of opening the device.
Write: Write data once, the parameters are the frame size and frame numbers corresponding to the buffer
Close: Close the output stream

8.6 Audio input collection

Example: **ogg_encode_test.cc**

Example of multiplexed media format packagingThe same example
Use the command: ./ogg_encode_test -f s16le -c 2 -r 48000 -i alsa:default -o output_s16le_c2_r48k.pcm

Interface and example process description

easymedia::REFLECTOR(Stream)::DumpFactories(): List the input and output modules currently compiled into the stream
The parameters are similar to c's FILE (can not be called)
easymedia::REFLECTOR(Stream)::Create<easymedia::Stream>: Create an audio playback collection stream instance, the parameter is a word
String "alsa_capture_stream" and set the parameters of opening the device.
Read: Read the data once, the parameter is buffer and its corresponding frame size and frame numbers

Page 7

Close: Close the collection stream

9.7 Audio coding

Example: **ogg_encode_test.cc**

Use the command to view the usage method: ./ogg_encode_test -? (Maybe there is no executable bin in the firmware generated by default, and it needs to be generated on the PC Manually push the completed path to the board end).

Interface and example process description

easymedia::REFLECTOR(Encoder)::DumpFactories(): List the currently programmed encoder modules (may not be called)
easymedia::REFLECTOR(Encoder)::Create<easymedia::AudioEncoder>: Create an audio encoder instance, parameters
The string corresponding to a module listed in the above DumpFactories, such as libvorbisenc in the example
InitConfig: Initialize the encoder, the parameter is the setting coefficient corresponding to the required encoding algorithm
Process: If the function returns a negative value and errno==ENOSYS, it means that the encoder does not support this interface and you need to call the following SendInput and FetchOutput interfaces
SendInput: incoming raw data SampleBuffer, if nb_samples is 0, it means that the encoder is notified that the incoming data is over
FetchOutput: Get the encoded data. Because some encoders, such as libvorbisenc, input one time and output multiple frames, so
Here you need a while loop to get until there is no data

10. 8 Media format packaging

Example: **ogg_encode_test.cc ffmpeg_enc_mux_test.cc**

Use the command to view the usage method: ./ogg_encode_test -? / ./ffmpeg_enc_mux_test -? (Maybe there is no such option in the default generated firmware
To execute bin, you need to manually push the path generated on the pc to the board end).

Interface and example process description

easymedia::REFLECTOR(Muxer)::DumpFactories(): List the currently compiled format package modules (may not be called)
easymedia::REFLECTOR(Muxer)::Create<easymedia::Muxer>: Create a format package instance, the parameters are the above
A string corresponding to a module listed in DumpFactories, such as liboggmuxer in the example
IncludeEncoder(): Determine whether the decapsulation module already includes the encoding function. If not, you need to follow the chapterAudio codingSaid
Create an encoder instance first do encoding and then pass in for encapsulation
NewMuxerStream: Create a data stream and return the serial number corresponding to the data stream in the parameter stream_no
SetIoStream: Manage the io stream to which data is written after encapsulation. If this function is called, the iostream will be called immediately after encapsulation
The Write method writes data; otherwise, an external program needs to process the output encapsulated data by itself. Because of its internal read and write logic, ffmpeg
This function is not supported
WriteHeader: Get the header information data of the package format
Write: Pass in the encoded data and the serial number of the corresponding data stream, and output the encapsulated data

11. 9 rtsp server (based on **live555**)

Example: **rtsp_server_test.cc**

Page 8

Copy the corresponding h264 single frame data [h264_frames] to the board folder for future use.
Use the command to view the usage: ./rtsp_server_test -? (Maybe there is no executable bin in the firmware generated by default, and it needs to be generated on the pc Manually push the path to the end of the board).
When playing and verifying on the PC side, you need to verify the reliability of the network first to avoid the jam problem caused by network packet loss.

Example process description

Take the SIMPLE macro as an example. This is a pure RTSP server function example.

split_h264_separate: Split multiple slices into separate slices, because live555 only accepts one slice at a time. As in the example, sps
Together with pps, you need to call this function for segmentation
SetUserFlag/SetValidSize/SetTimeStamp (MediaBuff interface): Set the Buffer attribute and send it to the Rtsp Flow
Before, MediaBuff must set these three attributes. Among them, SetUserFlag is used to mark whether the current Buffer is an I frame, etc.
easymedia::REFLECTOR(Flow)::Create<easymedia::Flow>("live555_rtsp_server", param.c_str()): Create
rtsp server, the parameters must include KEY_INPUTDATATYPE/KEY_CHANNEL_NAME,
KEY_PORT_NUM (port number) and KEY_USERNAME/KEY_USERPASSWORD (user name and password) are not
have to.

rtsp_flow->SendInput(buf, 0): Send data to the rtsp server, the second parameter 0, which means that it is sent to the rtsp server data link 0 input
After that, the rtsp server will get data from the 0 entry.

Note: The number of RTSP Flow entrances varies according to actual usage scenarios. If the built RTSP contains audio and video, then
RTSP Flow will have two entrances. When creating RTSP Flow, the audio parameters are before the video parameters, then the 0 entry will correspond to the audio
For audio, 1 entry corresponds to video; otherwise, 0 entry corresponds to video, and 1 entry corresponds to audio.

12. 10 Camera input collection

Only supports V4L2

Example: **camera_capture_test.cc**

Use the command to view the usage method: ./camera_cap_test -?

Interface and example process description

easymedia::REFLECTOR(Stream)::DumpFactories(): List the input and output modules currently programmed (may not be called)
easymedia::REFLECTOR(Stream)::Create<easymedia::Stream>: Create a camera capture stream instance, the parameters are characters
String "v4l2_capture_stream" and set the parameters of opening the device. Parameter reference example.
Read: Read in the data once, the parameter is empty, and return to MediaBuffer

13. 11 2D image processing

Example: **rga_filter_flow_test.cc**

Use the command to view the usage method: ./rga_filter_flow_test -?

Interface and example process description

easymedia::REFLECTOR(Filter)::Create<easymedia::Filter>: Create an instance of rga, the parameter is the string "rkrga" and the
Set the parameters of the image. Parameter reference example.
SetRects: Set the original and target rectangle information.
Process: Convert the input image into the target image according to the configuration.
rga_blit: Convert the input image into the target image according to the configuration. This interface can be used independently by Filter.

14. 12 Media Pipeline Construction

14.1 Introduction to **Flow**

Flow is used as a re-encapsulation of the above-mentioned functional modules to facilitate data communication between modules. For example, there are the following business scenarios:

v4l2 capture video --> rga --> display

If you use Flow, the application code will look like the following form (pseudo code):

```
// Create v4l2_capture_stream flow
std::string v4l2_params = "name=v4l2_capture_stream ...." ;
auto v4l2_flow = easymedia::REFLECTOR(Flow)::Create < easymedia::Flow >
( "source_stream" , flow_param.c_str());
// Create rga flow
std::string rga_params = "name=rkrga ...." ;
auto rga_flow = easymedia::REFLECTOR(Flow)::Create < easymedia::Flow > ( "filter" ,
flow_param.c_str());
// Create display flow
std::string rga_params = "name=drm_output_stream ...." ;
auto display_flow = easymedia::REFLECTOR(Flow)::Create < easymedia::Flow >
( "output_stream" , flow_param.c_str());
// data links
rga_flow -> AddDownFlow(display_flow, 0 , 0 );
v4l2_flow -> AddDownFlow(rga_flow, 0 , 0 );
```

14.2 **Flow** type

Flow can be divided into Source type, IO type, Sink type.

Source type: As a data source, get the data (read file/V4L2 node, etc.), encapsulate it as MediaBuff, and then send it to the lower level Flow. There is no superior flow. Such as: file_read_flow, source_stream, etc.

IO type: Process the input MediaBuff, and then output it to the lower-level Flow. Support multiple entrances and multiple exits. Such as: file_write_flow, output_stream

Sink type: As the last level of flow of the data path, this type of flow has no lower level of Flow.

14.3 Flow connection management

14.3.1 Flow connection

Flow uses the following interface to connect multiple Flows in series into a data pipeline (Pipeline).

```
bool AddDownFlow (std::shared_ptr < Flow > down, int out_slot_index,
int in_slot_index_of_down);
```

Parameter Description:

down: Lower-level Flow pointer

out_slot_index: The ID of the current Flow output interface. For example, if Flow has two entrances, the IDs are 0 and 1 respectively

in_slot_index_of_down: The ID of the inbound and outbound interface of the lower-level Flow. For example, if Flow has 3 entrances, the IDs are 0, 1, and 2, respectively.

Function description:

This interface establishes the binding relationship between the outlet and the inlet of the Flow, and specifies which outlet the current Flow selects through out_slot_index. in_slot_index_of_down selects which entry of the lower-level Flow, and then connects the two. After connection, the data will be transmitted according to the established connection data.

Connection mode:

One-to-one mode

One-to-one mode refers to the one-to-one binding of Flow's data exit and entry. Some flows support one entry/exit, and some flows support multiple entries Ports/exports, but the exit/entrance between the upper and lower flows are bound one by one. such as:

Example 1: FileSourceFlow -> FileSinkFlow

```
// Bind the 0 exit of file_src_flow to the 0 entry of file_sink_flow
file_src_flow -> AddDownFlow(file_sink_flow, 0 , 0 );
```

Example 2: VideoEncoderFlow -> MuxerFlow; AudioEncoderFlow -> MuxerFlow

```
// Bind the 0 exit of video_enc_flow to the 0 entry of muxer_flwo
// The 0 entry of MuxerFlow is the video stream input port.
video_enc_flow -> AddDownFlow(muxer_flwo, 0 , 0 );
// Bind the 0 exit of audio_enc_flow to the 1 entry of muxer_flwo
// The 1 entry of MuxerFlow is the audio stream input port.
video_enc_flow -> AddDownFlow(muxer_flwo, 0 , 1 );
```

One-to-many mode

One-to-many mode refers to the fact that there is one exit bound to multiple entries in the data entry and exit of Flow. This has nothing to do with how many exits/entries Flow has. ratio Such as:

Example 3: One FileSourceFlow connects to two FileSinkFlow

```
// Bind the 0 exit of file_src_flow to the 0 entry of file_sink_flow0
file_src_flow -> AddDownFlow(file_sink_flow0, 0 , 0 );
// Bind the 0 exit of file_src_flow to the 0 entry of file_sink_flow1
file_src_flow -> AddDownFlow(file_sink_flow1, 0 , 0 );
```


Note: In this mode, FileSourceFlow only sends one MediaBuff to the lower-level Flow, and the two lower-level FileSinkFlows get the same
A MediaBuff, the reference of MediaBuff plus one. In the scenario above, the lower-level Flow generates two copies of the same upper-level Flow.

14.3.2 **Flow** disconnect

Flow calls the following interface to disconnect

```
void RemoveDownFlow (std::shared_ptr < Flow > down);
```

Parameter Description:

down: The pointer of the lower-level Flow.

Note: When Flow is connected, in order to avoid data loss, it is best to connect to the data source Flow last; when Flow is disconnected, it is best to disconnect data first
Source Flow. For example, FileSourceFlow -> VideoEncFLow -> FileSinkFlow

connection:

```
VideoEncFLow -> AddDownFlow(FileSinkFlow, 0 , 0 );
// Finally connect to the data source
FileSourceFlow -> AddDownFlow(VideoEncFLow, 0 , 0 );
```

disconnect:

```
// Disconnect the data source first
FileSourceFlow -> RemoveDownFlow(VideoEncFLow);
VideoEncFLow -> RemoveDownFlow(FileSinkFlow);
```

14.4 **Flow** transmission mode

The mode of Flow determines whether or not threads are enabled for Flow data transfer. The following transmission modes are supported:

Synchronization mode (SYNC)

No thread is created inside Flow. When the upper-level Flow submits data to the current Flow, it will wait for the current Flow to process it before returning.

Asynchronous mode (ASYNCCOMMON)

A thread is created inside Flow, and the data submitted by the upper-level Flow is directly placed in the input buffer of the current Flow and returned immediately. Created by current Flow
The thread will get the data from the input buffer and send it to the lower-level Flow after processing.

Fixed-duration asynchronous mode (ASYNCATOMIC)

Similar to "asynchronous mode", but there is only one input buffer, the current thread inside Flow will fetch data from the input buffer at regular intervals according to.

The transmission mode of most flows is fixed, and a few flows can be selected by specifying KEK_THREAD_SYNC_MODEL when creating the flow. Option, such as adding in the string to create Flow:

```
"KEK_THREAD_SYNC_MODEL = asynccommon"
```

14.5 **Flow** Enumeration

audio_enc

Function: audio encoder package, support vorbis/aac/mp2/g711a/g711u/g726

Source code: audio_encoder_flow.cc

Example: audio_encoder_flow_test.cc

Type: IO type, 1 input and 1 output

file_read_flow

Function: read local files

Source code: file_flow.cc

Example: video_encoder_flow_test.cc

Type: Source type, 0 in and 1 out.

file_write_flow

Function: Write local files.

Source code: file_flow.cc

Example: video_encoder_flow_test.cc

Type: Sink type, 1 in 0 out

filter

Function: an IO type of Flow, which processes the input data and sends it to the downstream Flow, supporting rga/rknn, etc.

Source code: filter_flow.cc

Example: rga_filter_flow_test.cc

Type: IO type, support multiple input and multiple output (according to the actual use scenario)

live555_rtsp_server

Function: rtsp server based on live555

Source code: rtsp_server.cc

Example: rtsp_multi_server_test.cc

Type: Sink type, 1 in 0 out

move_detec

Function: do motion detection on the input image, and send the motion area information.

Source code: move_detection_flow.cc

Example: move_detection_flow_test.cc

Type: Sink type, 1 in 0 out

muxer_flow

Function: media packaging, support MP4/AVI/MPEG-PS/MPEG-TS/FLV/MKV

Source code: muxer_flow.cc

Example: muxer_flow_test.cc

Type: IO type, 2 in 1 out

output_stream

Function: encapsulate all output sources, such as drm_output_stream, alsa_playback_stream, etc.

Source code: output_stream_flow.cc

Example: drm_display_test.cc

Type: Sink type, 1 in 0 out.

source_stream

Function: encapsulate all data sources, such as alsa_capture_stream, v4l2_capture_stream, etc.

Source code: source_stream_flow.cc

Example: audio_loop_test.cc

Type: Source type, 0 in 1 out

video_dec

Function: Video decoding, support H265/H264/JPEG

Source code: decoder_flow.cc

Example: video_decoder_flow_test.cc

Type: IO type, 1 input and 1 output

video_enc

Function: video encoding, support H265/H264/JPEG

Source code: video_encoder_flow.cc

Example: video_encoder_flow_test.cc

Type: IO type, 1 input and 1 output