User Guide

For V2IP Development

TCCxxx-Android-ICS-ALL-V1.00E-User Guide-for V2IP Development

Aug. 14, 2013.

Preliminary

DISCLAIMER

All information and data contained in this material are without any commitment, are not to be considered as an offer for conclusion of a contract, nor shall they be construed as to create any liability. Any new issue of this material invalidates previous issues. Product availability and delivery are exclusively subject to our respective order confirmation form; the same applies to orders based on development samples delivered. By this publication, Telechips, Inc. does not assume responsibility for patent infringements or other rights of third parties that may result from its use.

Further, Telechips, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.
No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of Telechips, Inc.
This product is designed for general purpose, and accordingly customer be responsible for all or any of intellectual property licenses required for actual application. Telechips, Inc. does not provide any indemnification for any intellectual properties owned by third party.
Telechips, Inc. can not ensure that this application is the proper and sufficient one for any other purposes but the one explicitly expressed herein. Telechips, Inc. is not responsible for any special, indirect, incidental or consequential damage or loss whatsoever resulting from the use of this application for other purposes.

COPYRIGHT STATEMENT

Copyright in the material provided by Telechips, Inc. is owned by Telechips unless otherwise noted.
For reproduction or use of Telechips' copyright material, permission should be sought from Telechips. That permission, if given, will be subject to conditions that Telechips' name should be included and interest in the material should be acknowledged when the material is reproduced or quoted, either in whole or in part. You must not copy, adapt, publish, distribute or commercialize any contents contained in the material in any manner without the written permission of Telechips. Trade marks used in Telechips' copyright material are the property of Telechips.

Important Notice

This product may include technology owned by Microsoft Corporation and in this case it cannot be used or distributed without a license from Microsoft Licensing, GP.

For customers who use licensed Codec ICs and/or licensed codec firmware of mp3:
"Supply of this product does not convey a license nor imply any right to distribute content created with this product in revenue-generating broadcast systems (terrestrial. Satellite, cable and/or other distribution channels), streaming applications(via internet, intranets and/or other networks), other content distribution systems(pay-audio or audio-on-demand applications and the like) or on physical media(compact discs, digital versatile discs, semiconductor chips, hard drives, memory cards and the like). An independent license for such use is required. For details, please visit http://mp3licensing.com".

For customers who use other firmware of mp3:
"Supply of this product does not convey a license under the relevant intellectual property of Thomson and/or Fraunhofer Gesellschaft nor imply any right to use this product in any finished end user or ready-to-use final product. An independent license for such use is required. For details, please visit http://mp3licensing.com".

For customers who use Digital Wave DRA solution:
"Supply of this implementation of DRA technology does not convey a license nor imply any right to this implementation in any finished end-user or ready-to-use terminal product. An independent license for such use is required."

For customers who use DTS technology:
"Supply of this implementation of DTS technology does not convey a license, exhaust DTS' rights in the implementation, or imply a right under any patent, or any other industrial or intellectual property right of DTS to use, offer for sale, sell, or import such implementation in any finished end-user or ready-to-use final product. Notice is hereby provided that a license from DTS is required prior to such use."

"This product made under license to U.S. Patents 5,451,942; 5,956,674; 5,974,380; 5,978,762; 6,487,535; 6,226,616 and/or foreign counterparts."

"© 1996 – 2010 DTS, Inc."

Revision History

Date	Version	Description
2012-03-05	1.00	Initial Release
2013-08-14	1.10	Minor update for JB

TABLE OF CONTENTS

Contents

1 Introduction

This document provides guideline for users to implement V2IP(Video Voice over IP) using their own V2IP engine on Telechips Android platform quickly.

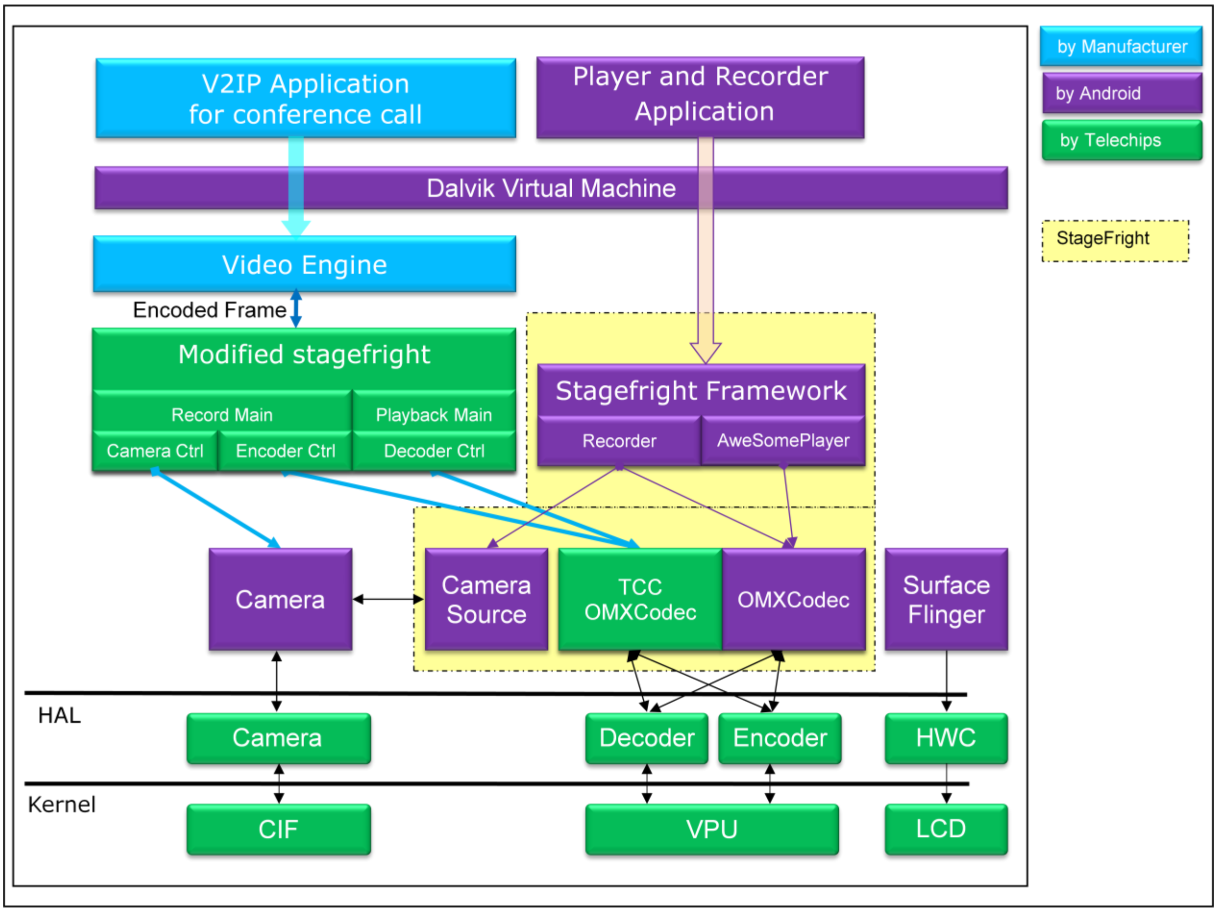
Note> Telechips supports various multimedia function which is required to implement Video/Audio Engine.
Telechips's Android platform also support V2IP engine itself using Unicoi solution which was developed by using these functions.

2 How to implement Video over IP function.

Manufacturer is responsible for V2IP App, JNI and native V2IP Engine.
The V2IP engine is in charge of many things: network communication using RTP, buffer management, data unpack/pack, A/V sync and etc.

Telechips provide the interface using Modified StageFright to implement Video engine.
(Almost function related with video will be provided by **libtcc.video.call.interface.so** library.)

Below diagram shows overall flows related with it.



3 How to get encoded frame.

3.1 C++ API header and library include

- Record_main.h
  - API for Encoder/Camera class.
  - Path: /hardware/telechips/common/tcc-interface/tccif/

- Shared library : libtcc.video.call.interface.so

3.2 Initialization and Startup

```
RecordMain *_record = new RecordMain((EncodedFrameCallBack*) this);

_record->set_Surface(surf);

_record-> configure_record(CODEC_FORMAT_H264, 1280, 720, 2048, 20, 3, 0);

_record-> start_record();
```

Below function can be called after calling start_record().

```
_record-> get_YUVframe(buffer);

_record-> replace_YUVframe(buffer, true);

_record->request_intraFrame();

_record->set_bitrate(1024);

_record->set_framerate(10);
```

3.3 Shutdown and Cleanup

```
_record->stop_record();

delete _record;
```

3.4 Reference

RecordMain()

This is the constructor of RecordMain class.
After record is started, the callback passed by argument will be called whenever encoded frame is ready.

Prototype	RecordMain(EncodedFrameCallBack *callback)
Parameters	Callback
Return value	None

~RecordMain()

This is the destructor of RecordMain class.

Prototype	~RecordMain()
Parameters	None
Return value	None

set_Surface ()

This is the method to set Surface which Camera will use

Prototype	status_t set_Surface(const sp<Surface>& surf)
Parameters	sp<Surface>& surf
Return value	android::OK or others(Error)

configure_record ()

This is the method to configure parameters which Camera and Encoder will use.
And, this create instance related with Camera and Encoder.

Prototype	status_t configure_record(int codec_info, int frame_width, int frame_height, int bitrate_Kbps, int framerate, int keyFrame_Interval_seconds, int no_preview)
Parameters	codec_info : CODEC_FORMAT_H263 / CODEC_FORMAT_MPEG4 / CODEC_FORMAT_H264 frame_width : 1280 ~ 160 frame_height : 720 ~ 120 bitrate_Kbps : 64Kbps ~ 8192Kbps framerate : 5 ~ 30 key_Frame_Interval_seconds : 1 ~ 10000 seconds no_preview: set this if you want not to preview the frame from Camera.
Return value	android::OK or others(Error)

start_record ()

This is the method to start a Camera and Encoder.
Once this is called, callback provided by user will be called whenever there is encoded frame.

Prototype	status_t start_record()
Parameters	None
Return value	android::OK or others(Error)

stop_record ()

This is the method to stop a Camera and Encoder.

Prototype	status_t stop_record()
Parameters	None
Return value	android::OK or others(Error)

get_YUV420p_Frame ()

This is the method to get YUV data form Camera.
Maybe, This function will be used to capture and save frame into image(ex. jpeg).

Prototype	status_t get_YUV420p_Frame (uint8_t *frame, uint32_t len, uint8_t *width, uint8_t *height)
Parameters	*frame : buffer pointer to get frame from Camera. len : buffer length. *width : frame width. *height: frame height.
Return value	android::OK or others(Error)

replace_YUV420p_Frame ()

This is the method to replace Camera frame into specific image(jpeg) provided by user because of security reason.
After user call this function with enable is 'true', user has to call again with enable is 'false' if user want to receive normal frame from Camera.

- Framerate value for Camera and Encoder will be changed into minimum value to reduce needless burden if enable argument is 'true'.
  Framerate value will be restored current value that is set recently after this function is recalled with enable is 'false'.

Prototype	status_t replace_YUV420p_Frame (uint8_t *frame, uint32_t len, bool enable)
Parameters	*frame : buffer pointer which has frame data. len: buffer length enable: true or false
Return value	android::OK or others(Error)

request_intraFrame ()

This is the method to request I-Frame to Encoder.

Prototype	status_t request_intraFrame()
Parameters	None
Return value	android::OK or others(Error)

set_bitrate ()

This is the method to change bitrate of the Camera and Encoder.

Prototype	status_t set_bitrate(int Kbps)
Parameters	bitrate
Return value	android::OK or others(Error)

set_framerate ()

This is the method to change framerate of the Camera and Encoder.

Prototype	int set_framerate(int fps)
Parameters	fps
Return value	applied framerate value. (because the Camera can't support all kinds of framerate.)

4 How to display encoded frame from Remote peer.

4.1 C++ API header and library include

- Playback_main.h
  - API for Decoder/Renderer class.
  - Path: /hardware/telechips/common/tcc-interface/tccif/

- Shared library : libtcc.video.call.interface.so

4.2 Initialization and Startup

PlaybackMain *_playback = new PlaybackMain ();

_ playback -> set_Surface (surf);

_ playback -> configure_playback (CODEC_FORMAT_H264, 1280, 720);

_ playback -> start_playback ();

Below function can be called after calling start_playback().

_ playback -> receivedPacket (data, len, timestamp);

_ playback -> get_YUVframe(buffer);

4.3 Shutdown and Cleanup

_ playback ->stop_playback ();

delete _ playback;

4.4 Reference

PlaybackMain()

This is the constructor of PlaybackMain class.

Prototype	PlaybackMain ()
Parameters	None
Return value	None

~PlaybackMain()

This is the destructor of PlaybackMain class.

Prototype	~PlaybackMain ()
Parameters	None
Return value	None

set_Surface ()

This is the method to set Surface which Renderer will use.

Please refer sample JNI codes to understand how to get ISurfaceTexture pointer.
Path: hardware/telechips/common/jniTccif/jniTccif.cpp

Prototype	void set_Surface(sp<ISurfaceTexture> &surf)
Parameters	sp< ISurfaceTexture >& surf
Return value	android::OK or others(Error)

configure_playback ()

This is the method to configure parameters which Decoder will use.
And, this create instance related with Decoder and Renderer.

Prototype	status_t configure_playback(int codec_info, int frame_width, int frame_height)
Parameters	codec_info : CODEC_FORMAT_H263 / CODEC_FORMAT_MPEG4 / CODEC_FORMAT_H264 frame_width : 1280 ~ 160 frame_height : 720 ~ 120
Return value	android::OK or others(Error)

start_playback()

This is the method to start Decoder and Renderer.

Prototype	status_t start_playback()
Parameters	None
Return value	android::OK or others(Error)

stop_playback()

This is the method to stop Decoder and Renderer.

Prototype	status_t stop_playback()
Parameters	None
Return value	android::OK or others(Error)

get_YUV420p_Frame()

This is the method to get YUV data form Decoder.
Maybe, This function will be used to capture and save frame into image(ex. jpeg).

Prototype	status_t get_YUV420p_Frame (uint8_t *frame, int32_t len, uint8_t *width, uint8_t *height)

Parameters	*frame : buffer pointer to get frame from Camera. len : buffer length. *width : frame width. *height: frame height.
Return value	android::OK or others(Error)

receivedPacket()

This is the method to receive encoded frame(depacketized one) from RTP.

Prototype	status_t receivedPacket(const uint8_t *data, uint32_t len, int64_t timestamp_ms)
Parameters	*data : buffer pointer which has frame. len : buffer length. timestamp_ms: timestamp value provided by Camera.
Return value	android::OK or others(Error)

5 Limitation.

This section describes performance limitation related with displaying through HDMI.

In order to prevent the performance issue

* First issue

TCC Android platform supports DVFS(Dynamic Voltage Frequency Scaling).
However DVFS is based on CPU load and does not consider other factors – for example VPU(Video Process Unit)/Camera h/w block.
For example, the operating clock is decided according to resolution and bit rate during video playback or camera operation, But it is not enough for V2IP because video playback and camera operation are done simultaneously.
To clear this problem, TCC use specific clock table as below.

Path: kernel/arch/arm/mach-tcc892x/tcc_clocktbl_ddr2.h or tcc_clocktbl_ddr3.h
kernel/arch/arm/mach-tcc88xx/tcc_clocktbl_ddr2.h or tcc_clocktbl_ddr3.h

The name of clock table: gtJpegMaxClockLimitTable
Each value means minimum clock of each h/w block including CPU.
So, User can adjust each value if user meet performance problem.

* Second issue

Flickering or tearing can be happened during displaying via HDMI because it needs more memory access than local LCD.

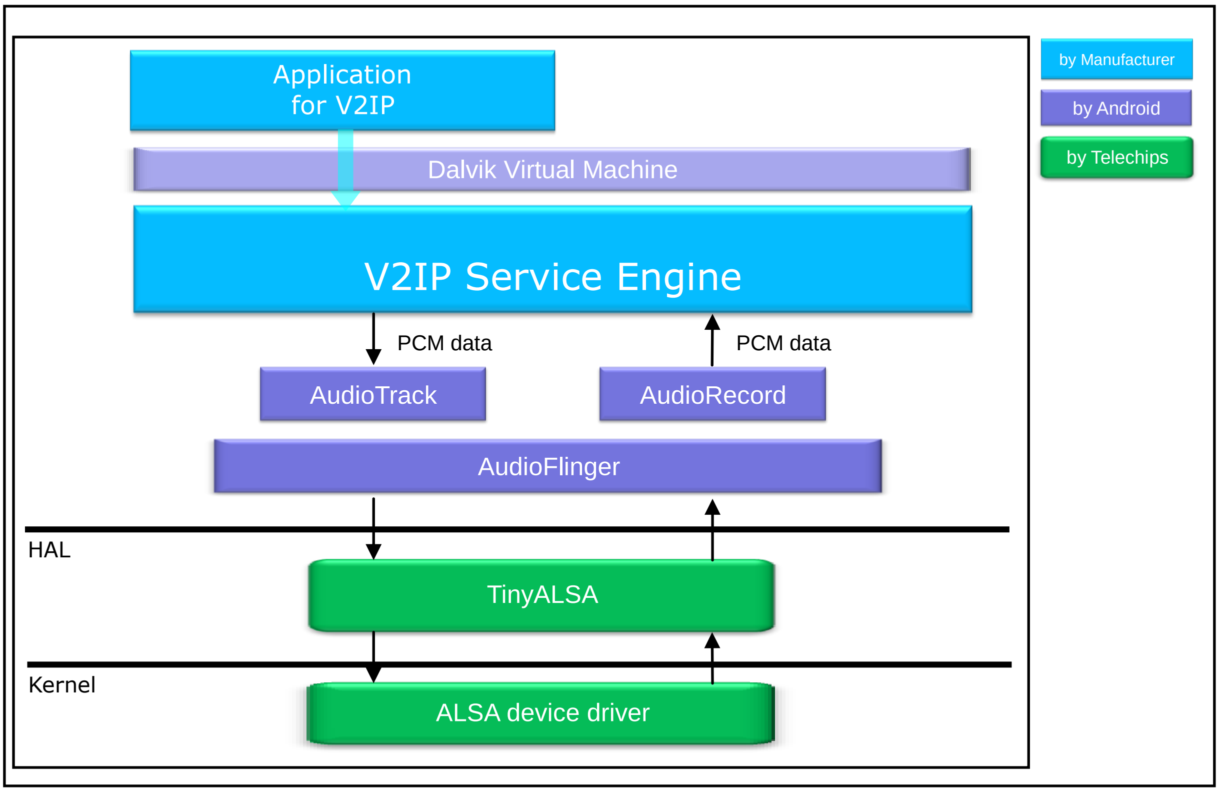
If user meet this kind of problems, HDMI resolution will have to be fixed into 720p to clear it.
Change below property by the own application while V2IP function use.

- "tcc.all.hdmi.720p.fixed" : default value is 0.
  : Use this property to fix HDMI resolution into 720p.

In case of java file, use SystemProperties.set() function to change setting value.
In case of cpp file, use property_set() function to change setting value.

6 How to develop Voice over IP function.

The media framework of android supports AudioTrack and AudioRecord Class for PCM data interface. You can get and send the PCM data by it. The block diagram of this is below.



For more information about AudioTrack and AudioRecord class, visit at

http://developer.android.com/reference/android/media/AudioRecord.html
http://developer.android.com/reference/android/media/AudioTrack.html

6.1 In order to PCM data using AudioRecord.

The AudioRecord class supports many functions for PCM data management. Among those, I explain the main function that you need to get PCM data. See the below sample code.

* Sample code

```
/************************************
* Initialize part
*    This needs just one time.
************************************/
AudioRecord mAudioRecord;
mAudioRecord = new AudioRecord();                                            ← Create the AudioRecord class
mAudioRecord.getMinFrameCount(&frameCount, sampleRate, format, channels);    ← get min frameCount
mAudioRecord.set(inputSource, sampleRate, format, channels, frameCount, …);  ← set parameter
            inputSource: 0 (Mic)
            sampleRate: 8000 ~ 48000
            format        : AUDIO_FORMAT_PCM_16_BIT
            channels      : AUDIO_CHANNEL_IN_MONO or AUDIO_CHANNEL_IN_STEREO
            frameCount : the total size of the buffer.
buffer = malloc(frameCount * 4);                                ← allocate memory at buffer.
mAudioRecord.start();                                           ← Start getting PCM data from inputSource.
```

```
/***********************************
* Get PCM Data from input device.
***********************************/
while (read_stop_flag) {
      ret = mAudioRecord.read(buffer, userSize);                ← You can get PCM data at buffer.
               buffer   : buffer pointer of input PCM data
               userSize: PCM data size
               ret         : actually read PCM data size.
      if(ret < 0) continue;                                     ← Do not get PCM data from input device
                                                                ← retry
      /* Insert the your code */                                ← PCM data copy, etc…
      …
}


/***********************************
* Destroy part
***********************************/
mAudioRecord.stop();                                            ← Stop getting PCM data from inputSource.
```

6.2 In order to play PCM data using AduioTrack.

The AudioTrack class supports many functions for PCM data management. Among those, I explain the main function that you need to write PCM data. See the below sample code.

* Sample code

```
/***********************************
* Initialize part
*    This needs just one time
***********************************/
AudioTrack mAudioTrack;
mAudioTrack = new AudioTrack();                                         ← Create the AudioTrack class
mAudioTrack.getMinFrameCount (&frameCount, streamType, sampleRate);     ← Get min frameCount
mAudioTrack.set(streamType, sampleRate, format, channels, frameCount, …); ← Set parameter
               streamType: the type of audio stream. Set the AUDIO_STREAM_VOICE_CALL
               sampleRate: 8000 ~ 48000
               format        : AUDIO_FORMAT_PCM_16_BIT
               channels     : AUDIO_CHANNEL_OUT_MONO or AUDIO_CHANNEL_OUT_STEREO
               frameCount : the total size of the buffer.
mAudioTrack.start();                                            ← Start PCM data playback
buffer = malloc(frameCount * 4);                                ← allocate memory at buffer.


/***********************************
* write pcm data to output device.
***********************************/
while(write_stop_flag) {
      ret = mAudioTrack.write(buffer, userSize);                ← Write PCM data to output device.
               buffer   : buffer pointer of output PCM data
               userSize: PCM data size
               ret        : actually written PCM data size.
      if(ret < 0) continue;                                     ← Do not write pcm data to output device.
                                                                ← retry
      /* Insert the your code */                                ← PCM data copy, etc…
      …
}


/***********************************
* Destroy part
***********************************/
mAudioTrack.flush();                                   ← Flush PCM buffer.
mAudioTrack.stop();                                    ← Stop PCM data playback.
delete mAudioTrack;                                    ← destroy AudioTrack class
```