# User Guide

# For V2IP Development

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

**Aug. 14, 2013.**

Telechips

***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



# 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 |
|---|---|---|
| 2012-03-28 | 1.00 | Initial Release |
| 2013-08-14 | 1.10 | Modified for Jelly Bean |
| | | |
| | | |
| | | |
| | | |

# 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.

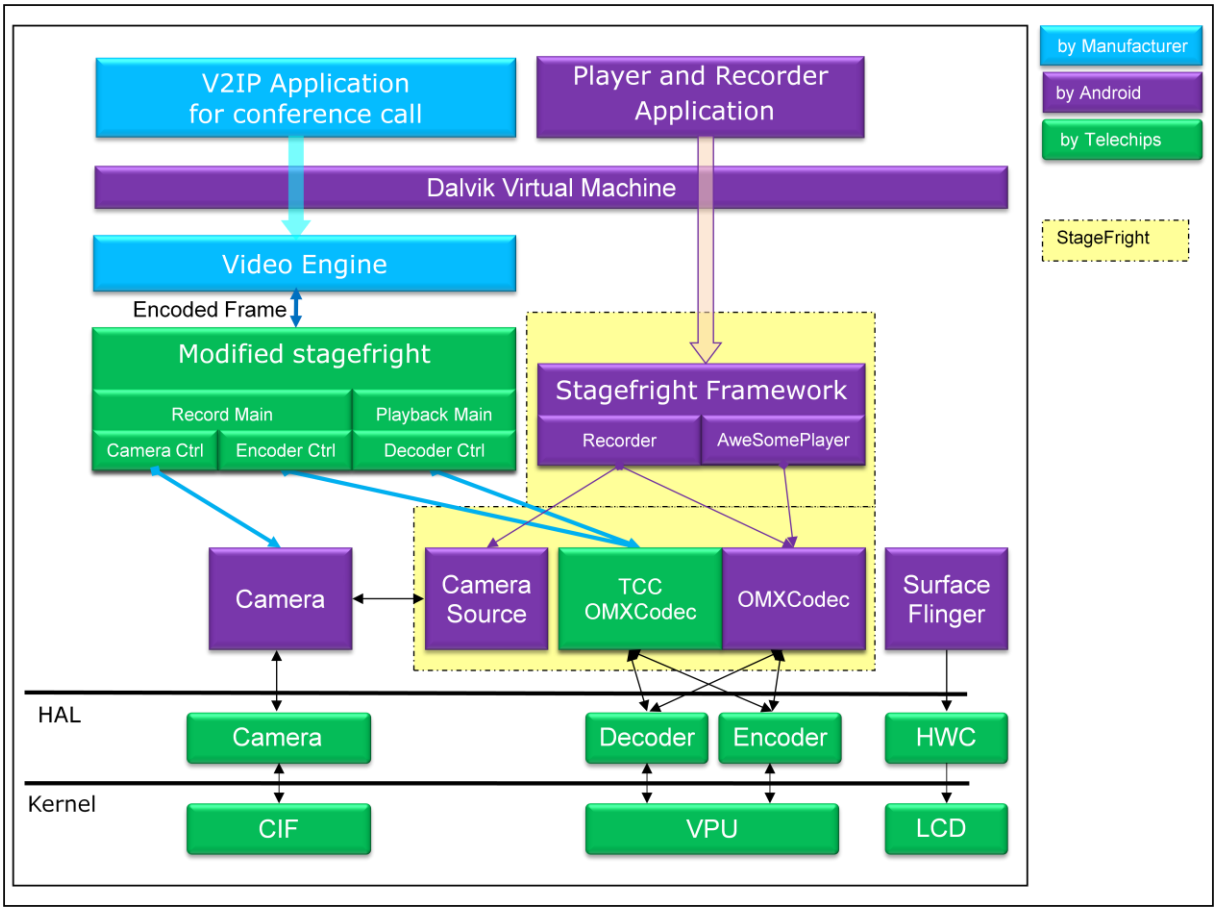
Telechips's Android platform also support V2IP engine itself using Unicoi solution which was developed by using these functions.

# 2 How to implement Video over IP function.

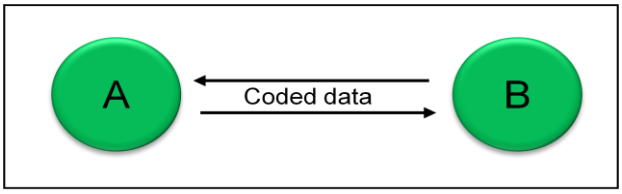
Manufacturer is responsible for V2IP App, JNI and native V2IP Engine.
The V2IP engine is in charge of many things: network communication using RTP, buffer management, data unpack/pack, A/V sync and etc.

Telechips provide the interface using Modified StageFright to implement Video engine.
It is very easy to implement. But, these interfaces only can support 2-way video call.
In other words, it is not suitable for multi-way video call because one of all peers has to be server to gather all those together in one frame.
Please refer another document(TC-TCCxxx-Android-ICS-ALL-V1.00E-User Guide-for V2IP-2way only Development.pdf) if your engine only support 2-way video call.

Below diagram shows simple operation related with 2-way video call using it.



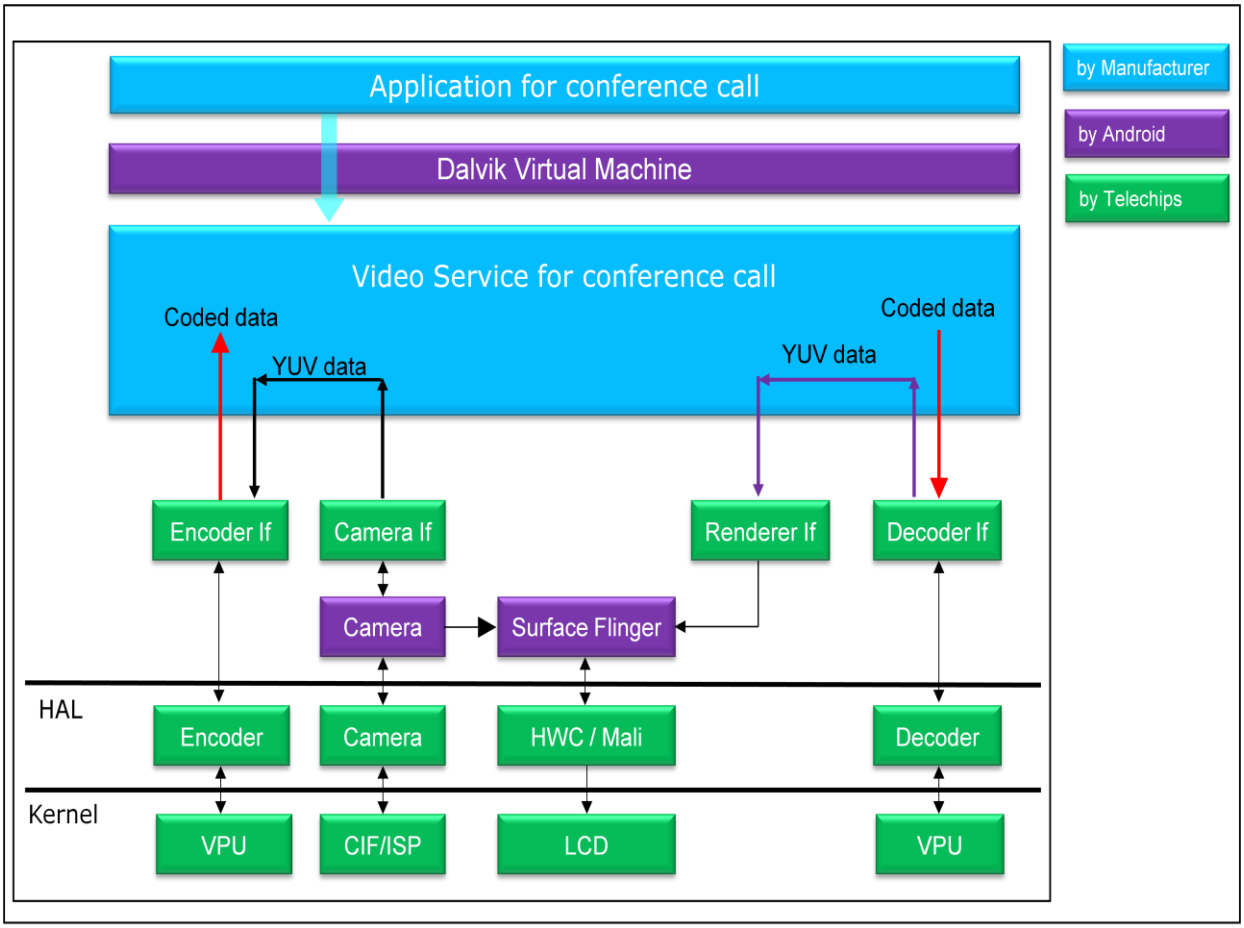
Telechips also provide the interface using direct API related to Camera, Encoder, Decoder, Renderer and Merger to implement multi-way video call.
(Most function related to video will be provided by **libtcc.video.call.direct.interface.so** library.)




Simple 2-way call diagram

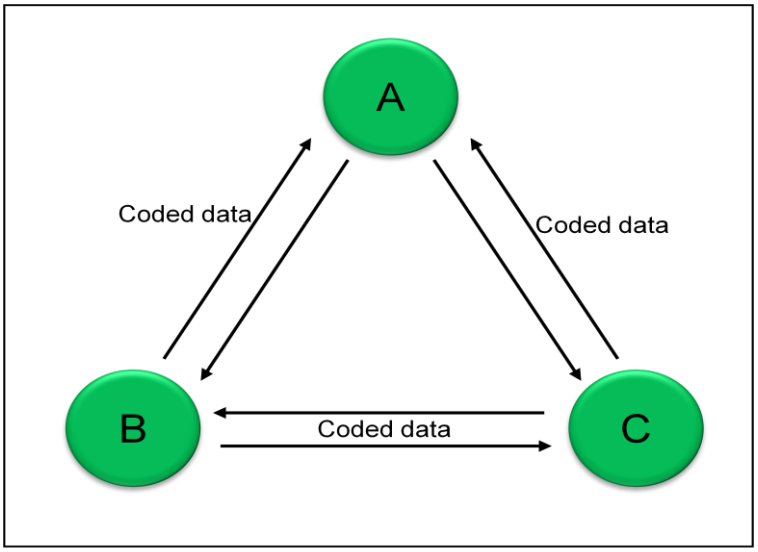
In case of 2-way video call,
- Each peer has to display own frame from camera.
- Each peer also has to display frame received through RTP.

Below diagram shows overall operation related with 2-way interface on Remote and Local peer.



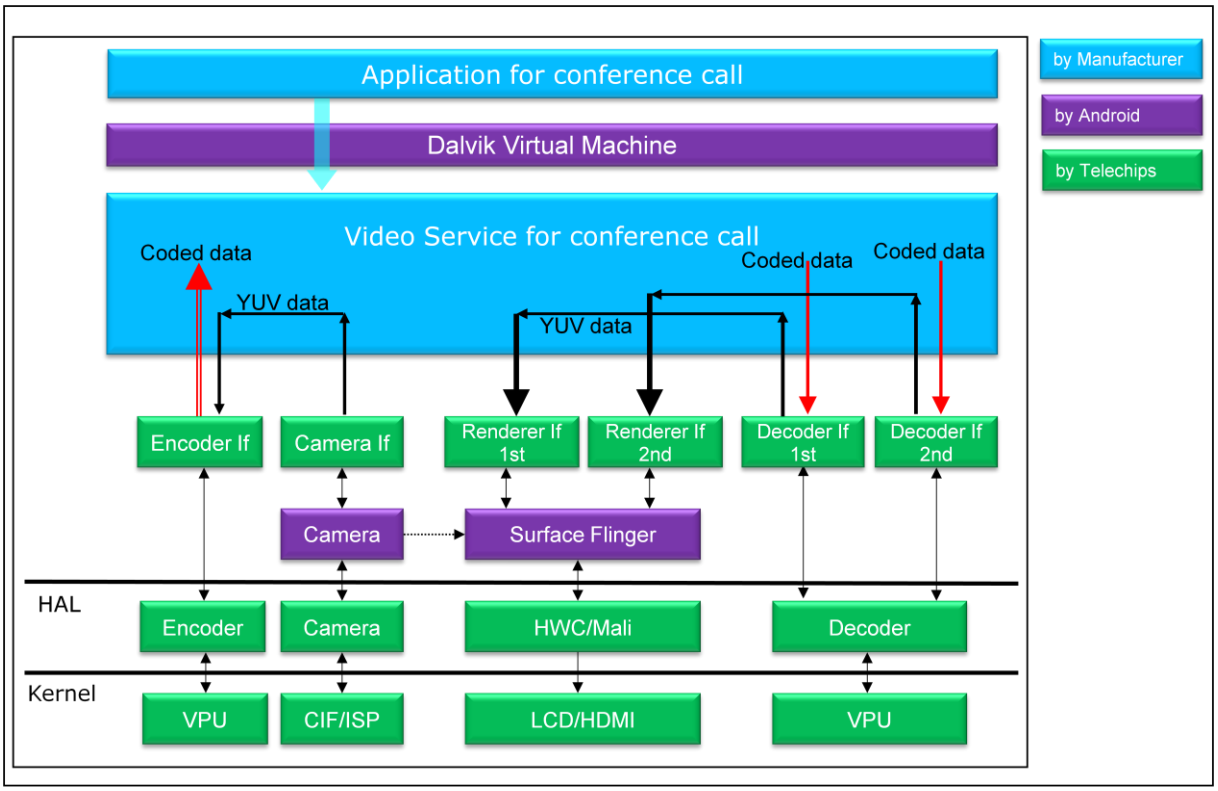

In case of 2way call. (Remote and Local peer)

Generally, It is easy to think that below diagram show multi-way video call.
But, It is not right way to implement multi-way video call.




Not right 3-way call diagram

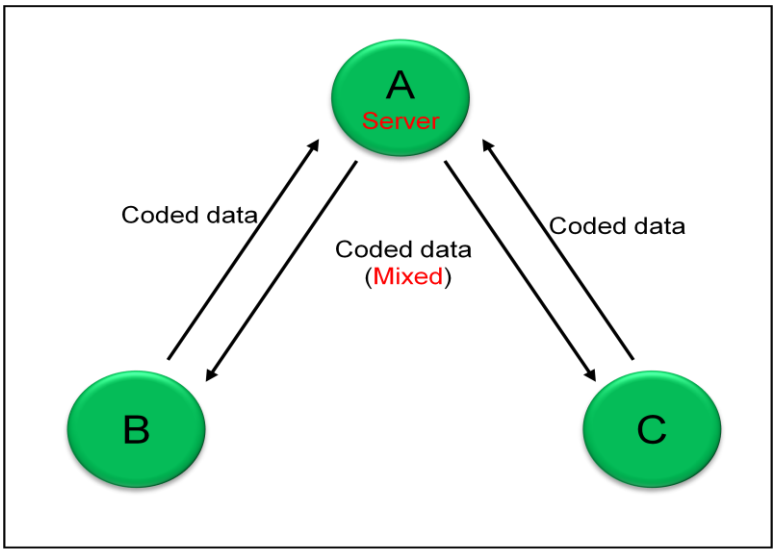
Even though this is not right way, it can be used if all peers's devices has enough performance to encode and decode simultaneously.




In case of multi-way call. (not right way)

If this way is used to implement multi-way call, the displayed region of each peer can be changed.

Below diagram shows right way for multi-way call.



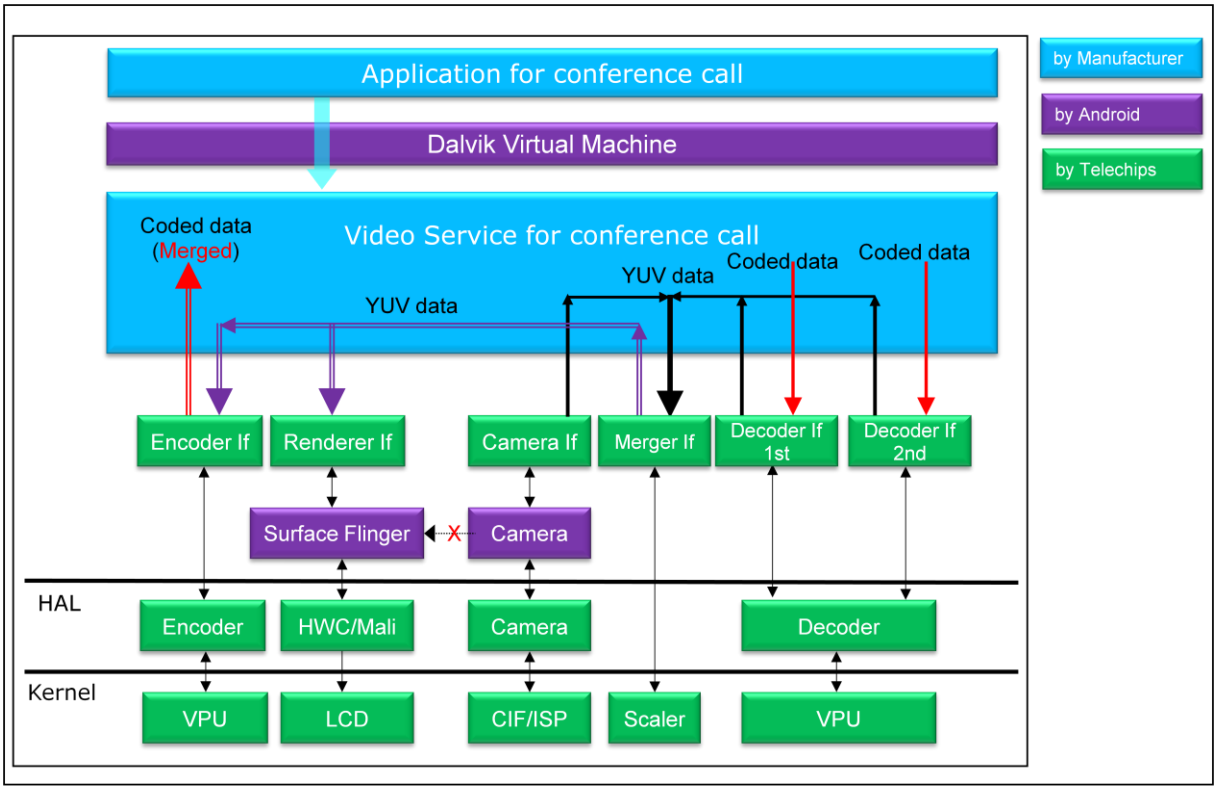

Simple 3-way call diagram

In case of multi-way video call,

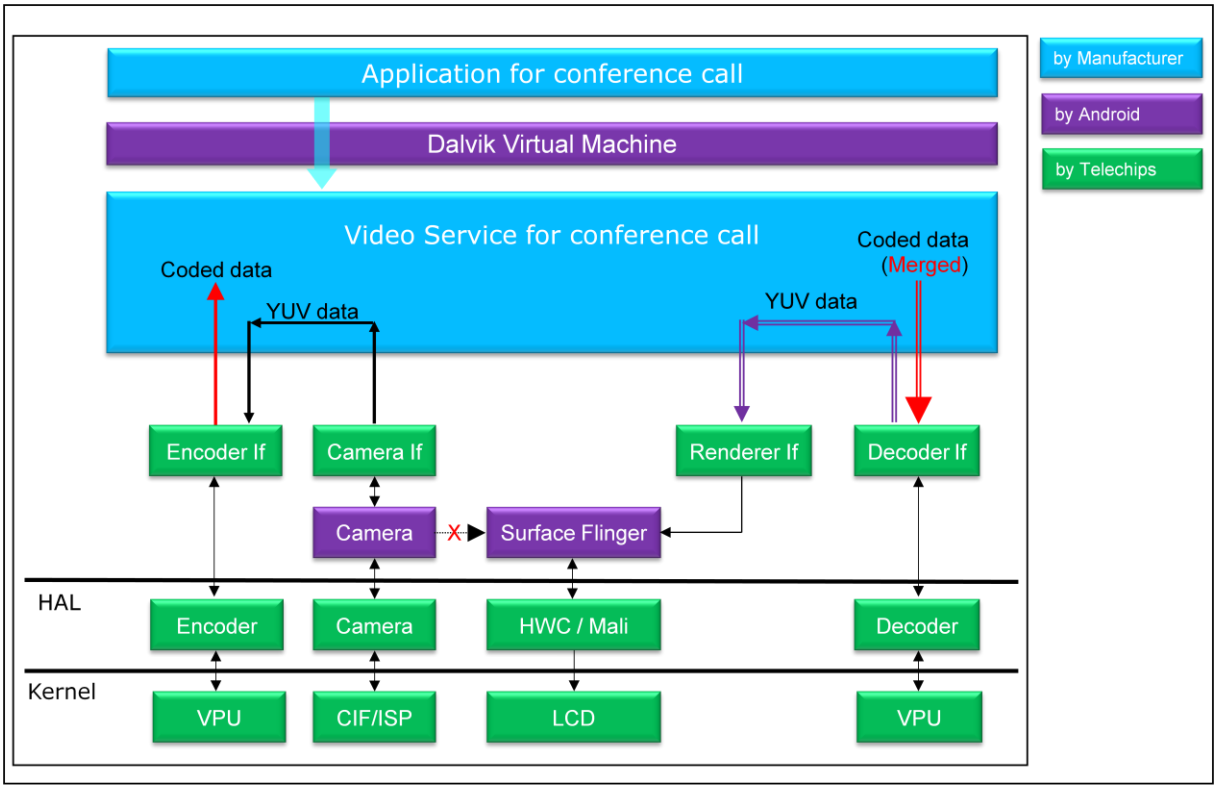
- One of peers has to be server peer.
- Server peer has to provide function to merge frames from clients and own camera.
- All peers don't have to display own frame from camera because of below reason.
  - Server peer only display merged frame which already have own frame and client's frames.
  - Client peers only display frame received through RTP.

- This concept has disadvantage like below:
  - The displayed region of each peer is fixed.
  - The server peer has to have enough performance to merge and decode several frames.
    - ⇨ Telechips provides Merger function using h/w operation.

Next diagram shows overall operation related with 3-way interface on Server and Client sides.




In case of multi-way call. (Server peer)




In case of multi-way call. (Client peers)

# 3 Camera interface class.

## 3.1 C++ API header and library include

- Camera_if.h
  - API for CameraIf class.
  - Path: /hardware/telechips/common/tcc-interface/tccif_direct/

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

## 3.2 Initialization and Startup

```
CameraIf *_camera = new CameraIf((CameraFrameCallBack*)this);

_camera->Init(pSurface, 1280, 720, 20, 0);

_camera->Start();
```

Below function can be called after calling start ().

```
_camera->GetFrame_Format();

_camera->SetFramerate(10);

_camera->ReleaseFrame(frame);

_camera->GetVirtual_Address(0);
```

## 3.3 Shutdown and Cleanup

```
_camera->Stop();

delete _camera;
```

## 3.4 Reference

**CameraIf ()**

This is the constructor of CameraIf class.
After camera is started, the callback delivered by argument will be called whenever captured frame from camera is ready.

| | |
|---|---|
| Prototype | CameraIf (CameraFrameCallBack *callback) |
| Parameters | Callback |
| Return value | None |

**~ CameraIf ()**

This is the destructor of CameraIf class.

| | |
|---|---|
| Prototype | ~ CameraIf () |
| Parameters | None |
| Return value | None |

**Init ()**

This is the method to initialize Camera.

| | |
|---|---|
| Prototype | int Init(const sp<Surface> pSurface, int frame_width, int frame_height, int default_framerate, int no_preview) |
| Parameters | const sp<Surface> pSurface<br>- This surface pointer has to be provided although the frame from Camera don't want to be displayed by itself.<br>- If the frame from Camera don't want to be displayed by itself, call Start() function with no_display, "1".<br>int frame_width : frame width<br>int frame_height : frame height<br>int default_framerate : framerate<br>int no_preview<br>- If the frame from Camera don't want to be displayed by itself, call Start() function with no_preview, "1". In this case, minimize region of Camera Surface.<br>- Otherwise, set the value into "0". |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Start ()**

This is the method to start Camera.

| | |
|---|---|
| Prototype | int Start() |
| Parameters | None |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Stop ()**

This is the method to stop Camera.

| | |
|---|---|
| Prototype | int Stop() |
| Parameters | None |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**ReleaseFrame ()**

This is the method to release it after frame from Camera is encoded.

| | |
|---|---|
| Prototype | void ReleaseFrame(const sp<IMemory>& frame) |

| | |
|---|---|
| Parameters | const sp<IMemory>& frame<br>- This pointer is same with received one from callback provided when calling constuctor of this class. |
| Return value | None |

**SetFramerate ()**

This is the method to change framerate.

| | |
|---|---|
| Prototype | int SetFramerate(int fps) |
| Parameters | int fps<br>- framerate value to change. |
| Return value | If successful, return fps value applied actually.<br>- (The applied fps can be differed with given one because the Camera can't support all kinds of framerate.)<br>- Use this returned value for encoder because same fps value have to set between camera and encoder. |

**GetVirtual_Address ()**

This is the method to get virtual address that can be matched with physical address given by Callback.

The physical address can be got like below:
- enc_metadata_t* pEncMetadataInfo = (enc_metadata_t*)dataPtr->pointer();
  unsigned int phyAddr = (unsigned int)pEncMetadataInfo->fd_0;
  Above phyAddr can be used for Encoder and Merger.

| | |
|---|---|
| Prototype | unsigned int GetVirtual_Address(unsigned int phyAddr) |
| Parameters | unsigned int phyAddr<br>- This address is what get from given frame's pointer by callback function when calling constructor of this class. |
| Return value | If successful, return virtual address. Otherwise it return NULL. |

**GetFrame_Format ()**

This is the method to get frame format from camera..

| | |
|---|---|
| Prototype | tFRAME_BUF_FORMAT GetFrame_Format(void) |
| Parameters | None |
| Return value | If successful, return format value. |

# 4 Encoder interface class.

## 4.1 C++ API header and library include

- Encoder_if.h
  - API for EncoderIf class.
  - Path: /hardware/telechips/common/tcc-interface/tccif_direct/
- Shared library : libtcc.video.call.direct.interface.so

## 4.2 Initialization and Startup

EncoderIf*_encoder = new EncoderIf();

Call below function after configuring tENC_INIT_PARAMS parameters.

_encoder->Init(pInit);

Below function can be called with configuring tENC_FRAME_INPUT parameters after calling Init().

_encoder->Encode(pInput, pOutput);

## 4.3 Shutdown and Cleanup

_encoder->Close();

delete _encoder;

## 4.4 Reference

**EncoderIf ()**

This is the constructor of EncoderIf class.

| Prototype | EncoderIf() |
|---|---|
| Parameters | None |
| Return value | None |

**~ EncoderIf ()**

This is the destructor of EncoderIf class.

| Prototype | ~ EncoderIf () |
|---|---|
| Parameters | None |
| Return value | None |

**Init ()**

This is the method to initialize Encoder

| | |
|---|---|
| Prototype | int Init(tENC_INIT_PARAMS *pInit) |
| Parameters | tENC_INIT_PARAMS *pInit<br>- pointer of encoder initial parameters<br>● refer Encoder_if.h file in detail. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Encode ()**

This is the method to encode frame.

| | |
|---|---|
| Prototype | int Encode(tENC_FRAME_INPUT *pInput, tENC_FRAME_OUTPUT *pOutput) |
| Parameters | tENC_FRAME_INPUT *pInput<br>- pointer of encoder input parameters<br>● refer Encoder_if.h file in detail.<br>tENC_FRAME_OUTPUT *pOutput<br>- pointer of encoder output parameters<br>● refer Encoder_if.h file in detail. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Close ()**

This is the method to close Encoder.

| | |
|---|---|
| Prototype | int Close(void) |
| Parameters | None |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

# 5 Decoder interface class.

## 5.1 C++ API header and library include

- Decoder_if.h
  - API for DecoderIf class.
  - Path: /hardware/telechips/common/tcc-interface/tccif_direct/
- Shared library : libtcc.video.call.direct.interface.so

## 5.2 Initialization and Startup

DecoderIf*_decoder = new DecoderIf();

Call below function after configuring tDEC_INIT_PARAMS parameters.

_decoder->Init(pInit);

Below function should be called with configuring tDEC_FRAME_INPUT parameters after calling Init().

_decoder->Decode(pInput, pOutput, pResult);

## 5.3 Shutdown and Cleanup

_decoder->Close();

delete _decoder;

## 5.4 Reference

**DecoderIf ()**

This is the constructor of DecoderIf class.

| | |
|---|---|
| Prototype | DecoderIf () |
| Parameters | None |
| Return value | None |

**~ DecoderIf ()**

This is the destructor of DecoderIf class.

| | |
|---|---|
| Prototype | ~ DecoderIf () |
| Parameters | None |
| Return value | None |

**Init ()**

This is the method to initialize Decoder

| | |
|---|---|
| Prototype | int Init(tDEC_INIT_PARAMS *pInit) |
| Parameters | tDEC_INIT_PARAMS *pInit<br>- pointer of decoder initial parameters<br>● refer Decoder_if.h file in detail. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Decode ()**

This is the method to decode frame.

| | |
|---|---|
| Prototype | int Decode(tDEC _FRAME_INPUT *pInput, tDEC _FRAME_OUTPUT *pOutput , tDEC_RESULT *pResult) |
| Parameters | tDEC _FRAME_INPUT *pInput<br>- pointer of decoder input parameters<br>● refer Decoder_if.h file in detail.<br>tDEC _FRAME_OUTPUT *pOutput<br>- pointer of decoder output parameters<br>● refer Decoder_if.h file in detail.<br>tDEC_RESULT *pResult<br>- pointer of decode result patameters<br>● refer Decoder_if.h file in detail. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value.<br>If successful, check *pResult. |

**Close ()**

This is the method to close Decoder.

| | |
|---|---|
| Prototype | int Close(void) |
| Parameters | None |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

# 6 Renderer interface class.

## 6.1 C++ API header and library include

- Renderer_if.h
  - API for RendererIf class.
  - Path: /hardware/telechips/common/tcc-interface/tccif_direct/

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

## 6.2 Initialization and Startup

```
RendererIf *_renderer = new RendererIf(surfaceTexture, NULL, width, height, format);
```

Check if the RendererIf class is created properly or not.

```
If(false == _renderer->IsInit_OK())
    return error;
```

Call the Render() function whenever the frame is ready.

```
_renderer->Render(Y, U, V, 1, 233,private_data);
```

## 6.3 Shutdown and Cleanup

```
delete _renderer;
```

## 6.4 Reference

**RendererIf ()**

This is the constructor of RendererIf class.

| | |
|---|---|
| Prototype | RendererIf (const sp<ISurfaceTexture> &surfaceTexture, void *pNativeWindow, unsigned int inFrame_width, unsigned int inFrame_height, tFRAME_BUF_FORMAT inBuffer_format, int outbuffer_count = DEF_BUFFER_CNT) |
| Parameters | const sp<ISurfaceTexture> &surfaceTexture<br>- pointer of ISurfaceTexture<br>void *pNativeWindow<br>- pointer of NativeWindow<br>● The two argument(surfaceTexture, pNativeWindow) are mutually exclusive. So, one of those arguments should be NULL.<br>unsigned int inFrame_width<br>- width of input frame<br>unsigned int inFrame_height<br>- height of input frame<br>tFRAME_BUF_FORMAT inBuffer_format<br>- format of input frame (raw data) |

| | |
|---|---|
| | int outbuffer_count<br>- buffer count allocated |
| Return value | None |

**~ RendererIf ()**

This is the destructor of RendererIf class.

| | |
|---|---|
| Prototype | ~ RendererIf () |
| Parameters | None |
| Return value | None |

**IsInit_OK ()**

This is the method to check if Renderer instance is created properly..

| | |
|---|---|
| Prototype | bool IsInit_OK() |
| Parameters | NONE |
| Return value | If successful, it returns true. Otherwise, it returns false. |

**Render ()**

This is the method to display frame.

| | |
|---|---|
| Prototype | status_t Render(unsigned int Yaddr, unsigned int Uaddr, unsigned int Vaddr, char bAddrPhy, int64_t timestamp_ms, TCC_PLATFORM_PRIVATE_PMEM_INFO *plat_priv = NULL) |
| Parameters | unsigned int Yaddr<br>- Y(Luminance) address of frame that will be displayed.<br>unsigned int Uaddr<br>- U/UV(Chrominance) address of frame that will be displayed.<br>unsigned int Vaddr<br>- V(Chrominance) address of frame that will be displayed.<br>- This can be NULL.<br>char bAddrPhy<br>- It indicates if Yaddr/Uaddr/Vaddr is physical address or not.<br>int64_t timestamp_ms<br>- Timestamd of current frame (unit is milliseconds.)<br>TCC_PLATFORM_PRIVATE_PMEM_INFO *plat_priv<br>- If the frame is from h/w decoder, it should be delivered. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

# 7 Merger interface class.

## 7.1 C++ API header and library include

- Merger_if.h
  - API for MergerIf class.
  - Path: /hardware/telechips/common/tcc-interface/tccif/

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

## 7.2 Initialization and Startup

```
MergerIf*_merger = new MergerIf();
If(_merger == NULL)
	return error;
```

Allocate target buffer for the Merger.

```
int ret = _merger->Allocate_TargetBuffer(width, height, format);
If(_ret < 0)
	return error;
```

Call the Merge() function whenever there is a frame to be merged.

```
_merger->Merge(320,240, input_physical_address, input_format, 0,0,320,240);
```

Use below function if there is need to get the buffer's address mixed.

```
_merger->Get_BufferAddress(1, 0);
```

## 7.3 Shutdown and Cleanup

```
_merger->Close();

delete _merger;
```

## 7.4 Reference

**MergerIf ()**

This is the constructor of MergerIf class.

| | |
|---|---|
| Prototype | MergerIf() |
| Parameters | None |
| Return value | None |

**~ MergerIf ()**

This is the destructor of MergerIf class.

| | |
|---|---|
| Prototype | ~ MergerIf () |
| Parameters | None |
| Return value | None |

**Alloc_TargetBuffer ()**

This is the method to initialize Merger.

| | |
|---|---|
| Prototype | int Alloc_TargetBuffer (int width, int height, tFRAME_BUF_FORMAT format) |
| Parameters | - These are the function to allocate final target buffer that input images are merged in. Based on these parameters, target buffer can be allocated.<br><br>int width : width of target buffer.<br>int height : height of target buffer.<br>tFRAME_BUF_FORMAT format : format of target buffer. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Get_BufferAddress ()**

This is the method to get buffer address.

| | |
|---|---|
| Prototype | unsigned int Get_BufferAddress(bool isPhyAddr, bool previous_buffer) |
| Parameters | bool isPhyAddr<br>- Set this value to get physical address, otherwise virtual address will be returned.<br>bool previous_buffer<br>- Set this value to get previous buffer's address, otherwise current buffer's address will be returned. |
| Return value | Target buffer's address (virtual or physical address) |

This is the example how to set arguments to get proper address.

1. In order to get physical address of current buffer..
   _merger->Get_BufferAddress(1, 0).
2. In order to get virtual address of current buffer.
   _merger->Get_ BufferAddress (0, 0).
3. In order to get physical address of previous buffer.
   _merger->Get_ BufferAddress (1, 1).
4. In order to get virtual address of previous buffer.
   _merger->Get_ BufferAddress (0, 1).

**Close ()**

This is the method to close Merger.

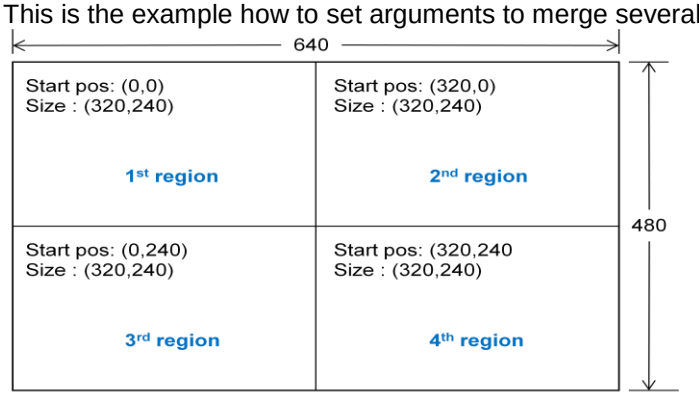
| | |
|---|---|
| Prototype | int Close(void) |
| Parameters | None |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

**Merge ()**

This is the method to merge frames.

| | |
|---|---|
| Prototype | int Merge(int in_width, int in_height, tFRAME_BUF_FORMAT in_format, unsigned int in_address, int out_sx, int out_sy, int out_width, int out_height, int new_buffer) |
| Parameters | ● These are the information related to input frames from Camera and Decoders.<br><br>int in_width : width of input frame<br>int in_height : height of input frame<br>tFRAME_BUF_FORMAT in_format : format of input frame<br>unsigned int in_address : physical address of buffer having raw data.<br><br>● These are the information related to region in target buffer where input frames are merged in.<br><br>int out_sx : start x-position<br>int out_sy : start y-position<br>int out_width : width<br>int out_height : height<br><br>int new_buffer<br>- to replace the target buffer into new one. |
| Return value | If successful, it returns 0 or plus. Otherwise, it returns a minus value. |

This is the example how to set arguments to merge several frames in one frame.




- In case target resolution is VGA(640x480). -

1. if the QVGA frame with YUV420P want to be displayed in 2nd region, set like below
   _merger->Merge( 320, 240, FRAME_BUF_FORMAT_YUV420P, in_address, 320, 0, 320, 240, 0);

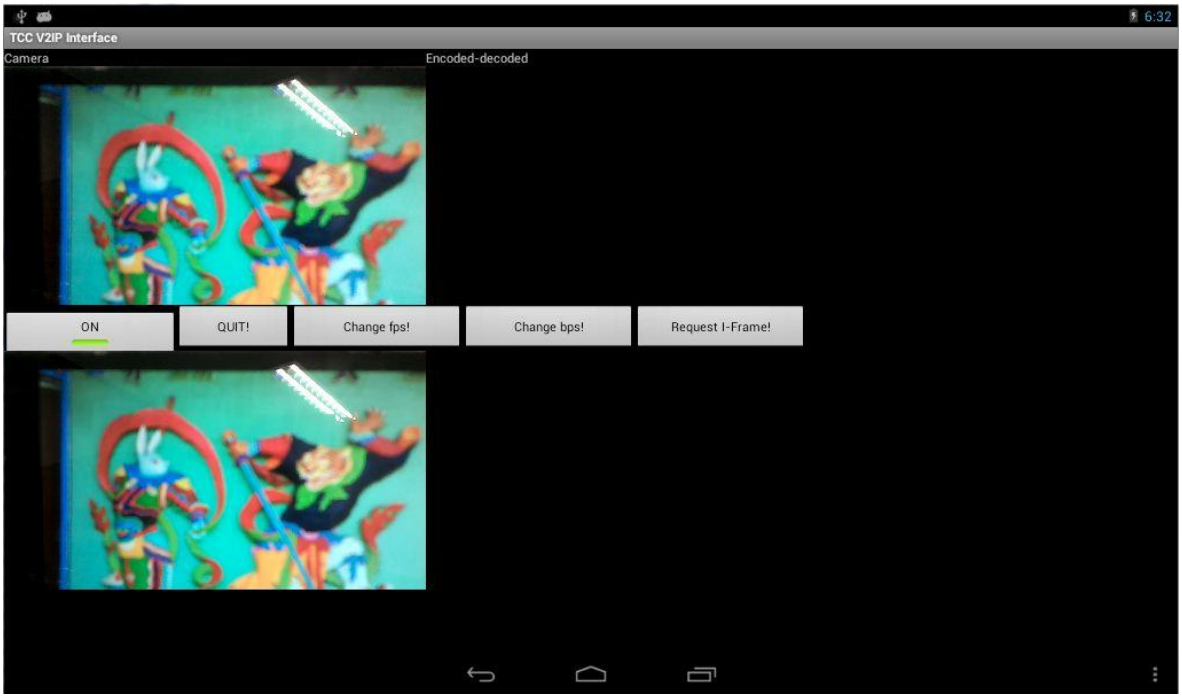
2. if the VGA frame with YUV420I want to be displayed in 3rd region, set like below
   _merger->Merge( 640, 480, FRAME_BUF_FORMAT_YUV420I, in_address, 0, 240, 320, 240, 0);
   In case of this, input VGA frame will be scaled down into QVGA.

3. if the QCIF frame with YVU420P want to be displayed in 4th region, set like below
   _merger->Merge( 176, 144, FRAME_BUF_FORMAT_YVU420P, in_address, 320, 240, 320, 240, 0);
   In case of this, input QCIF frame will be scaled up into QVGA.

# 8 Sample codes.

This section describes the sample codes to test if the V2IP interface works well or not.

This apk is very simple loopback application.
The application records from camera -> sends to encoder -> sends to decoder -> displays with renderer.
Changing bitrate and framerate and requesting the I-Frame can be also tested.




To start, press left “OFF” button to make “ON”.

## In order to build simple application

You can find the source codes related to sample application.

- Path: hardware/telechips/common/tcc-interface/jniTccif
  Apk source: TCC_Interface_sample_apk_src.zip

You should choose the interface type between direct-if and modified-stagefright-if.

- Path: hardware/telechips/common/tcc-interface/jniTccif/jniTccif.cpp
- Enable TEST_TCCIF_DIRECT_INTERFACE Feature to use the direct-if.

1). Put the 3 so library into proper folder after building Android SDK.

- From out/target/product/tcc893x/obj/lib
  : libjniTccif.so, libtcc.video.call.interface.so, libtcc.video.call.direct.interface.so
- To TCC_Interface/jni/jniTccif/libjniTccif.so
  TCC_Interface/jni/libtccif/libtcc.video.call.interface.so
  TCC_Interface/jni/libtccifdirect/libtcc.video.call.direct.interface.so

2). Run ndk-build on TCC_Interface.

3). Build using Eclipse and run.

# 9 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 builtin LCD.

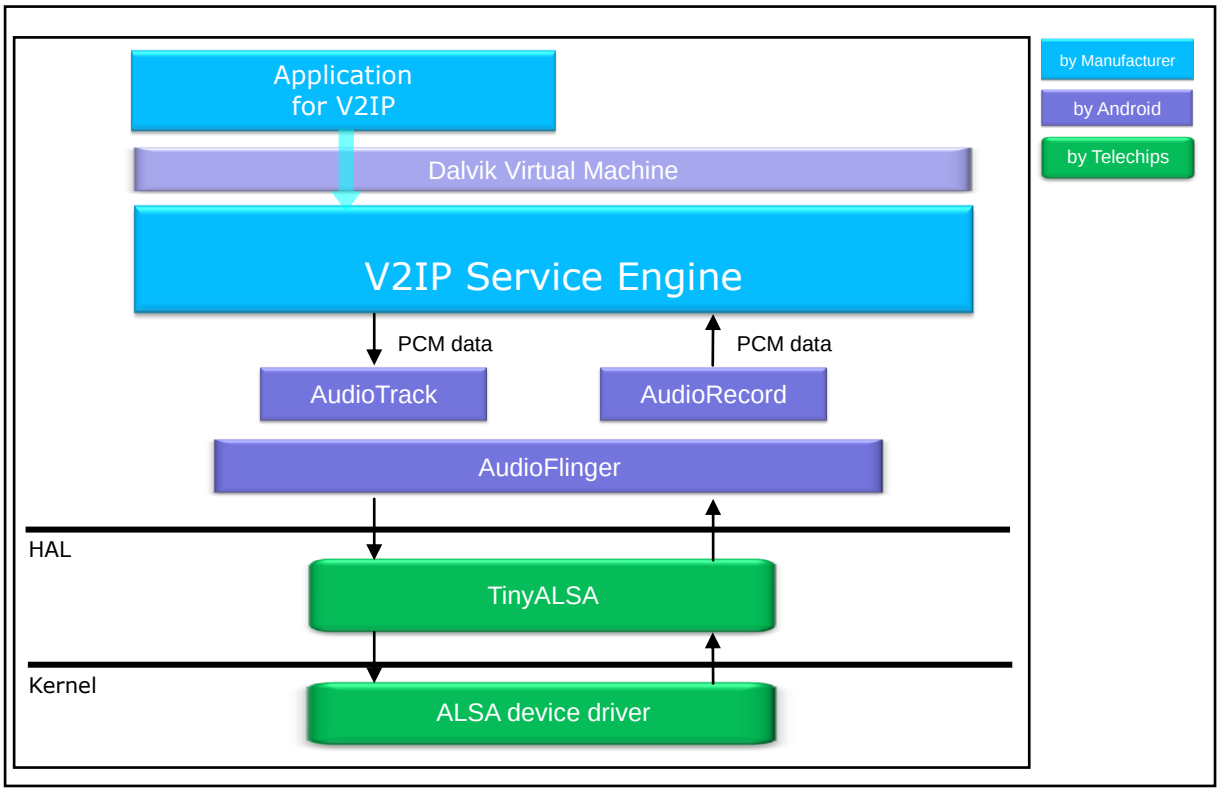
If user meet this kind of problems, HDMI resolution will have to be fixed into 720p to clear it.
Change below property by the own application while V2IP function use.

- “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.

# 10 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

## 10.1 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.
```

## 10.2 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
```