MPP Development Reference

File Status:	Project:	SoFiA-3GR YoCTO Linux
[ ] Draft	Version:	1.1
[ ] Beta	Author:	timkingh.huang
[√] Release	Date:	10/09/2016

Revision	Date	Description	Author
1.0	7/26/2016	This revision is the first draft release.	timkingh.huang
1.1	10/9/2016	The vpu_api interface is replaced by rk_mpi.	timkingh.huang

Table of Content

Chapter 1 Introduction to the MPP

1.1 Overview

Rockchip provides a media processing platform (MPP for short) for rapid application development. For application, the MPP shields the complex processing procedures at the bottom layer related to the chip, and provides MPP programming interfaces (MPIs) for implementing various functions. The functions include:

- encoding
  - H.264: up to High Profile Level 4.0, including Baseline Profile and Main Profile, 1920x1080@30fps
- decoding
  - H.264: up to High Profile Level 4.1, including Baseline Profile and Main Profile, 1920x1080@30fps
  - H.265: Main Profile up to Level 4.1, 1920x1080@30fps
  - MPEG-2: Main Profile up to High Level, 1920x1080@30fps
  - MPEG-4: Simple Profile up to Level 6; Advanced Profile up to Level 5, 1920x1080@30fps
  - VP8: 1920x1080@30fps

This document describes the architecture and modules of the MPP, and the user MPIs related to the MPP. It is intended for but not limited to application development engineers and technical support personnel.

1.2 System Architecture

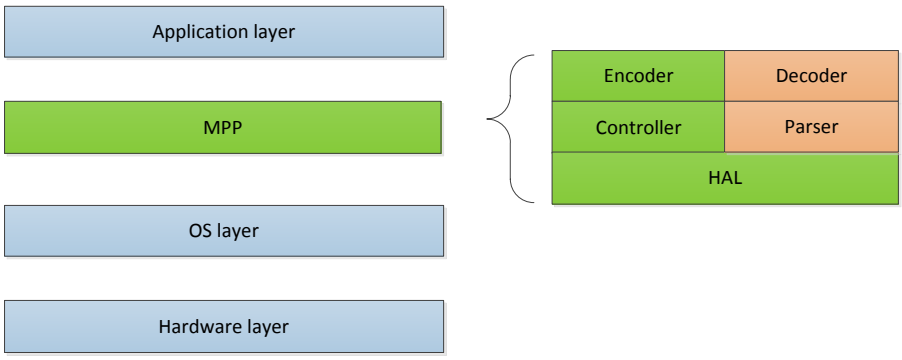

Figure 1 System architecture of the MPP

- Hardware layer

It is comprised of the RK32xx and peripherals, including the flash memory, double-data rate (DDR), video sensor or video analog-to-digital converter(VADC).

- Operating system(OS) layer

Android, Linux or Windows OS

- MPP

Based on the OS layer, the MPP controls the media processing functions. That is, the MPP shields the details about hardware processing, and provides application programming interface (APIs) for the application layer.

- Application layer

It is used for developing application based on the MPP and drivers.

1.3 Architecture of MPP

In MPP, mpp layer creates mpp_dec instance including parser and hal module or mpp_enc instance including control and hal module. For encoder, mpp layer receives the pictures from video input module and encodes the pictures and outputs streams complying with different protocols. For decoder, mpp layer can decode the encoded video stream and transfer the decoded streams to the video output module.

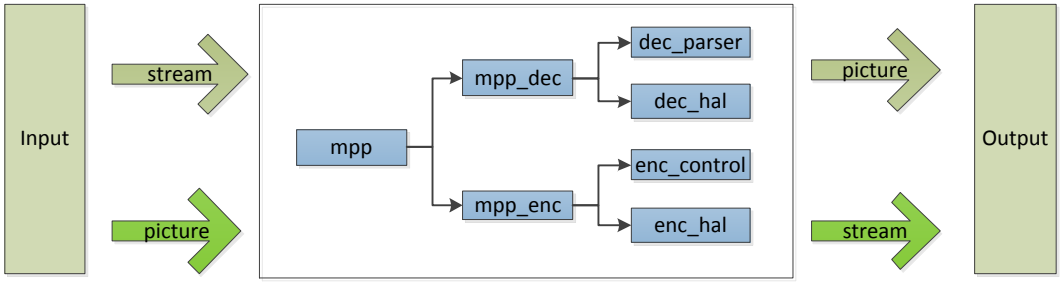

Figure 2 Internal workflow of the MPP

1.4 Function Description

1.4.1 File Structure

(1)
/MPP/build : CMake out-of-source build directory
/MPP/build/cmake : cmake script directory
/MPP/build/android : android build directory
/MPP/build/linux : linux build directory
/MPP/build/vc10-x86_64 : visual studio 2010 on x86_64 build directory
/MPP/build/vc12-x86_64 : visual studio 2013 on x86_64 build directory
(2)
/MPP/inc: header file for external usage, including platform header and mpi header
(3)
/MPP/mpp: Media Process Platform: mpi function private implement and mpp infrastructure (vpu_api private layer)
/MPP/mpp/common: video codec protocol syntax interface for both codec parser and hal
/MPP/mpp/codec: all video codec parser, convert stream to protocol structure
/MPP/mpp/codec/inc: header files provided by codec module for external usage
/MPP/mpp/codec/dec: mpp decoder parser
/MPP/mpp/codec/enc: mpp encoder controllor
(4)
/MPP/mpp/hal: Hardware Abstract Layer (HAL): modules used in mpi
/MPP/mpp/hal/inc: header files provided by hal for external usage
/MPP/mpp/hal/iep: iep user library
/MPP/mpp/hal/pp: post-processor user library
/MPP/mpp/hal/rga: rga user library
/MPP/mpp/hal/deinter: deinterlace function module including pp/iep/rga
/MPP/mpp/hal/rkdec: rockchip hardware decoder register generation library

/MPP/mpp/hal/vpu: vpu register generation library

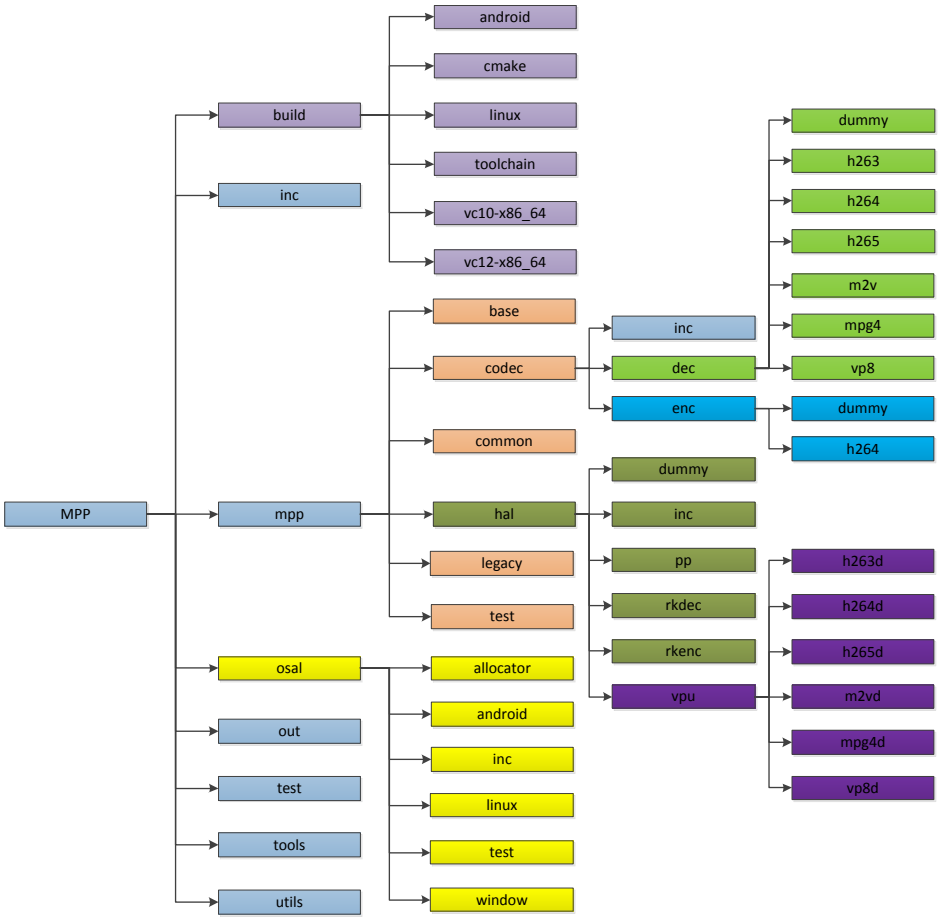

Figure 3 File structure of the MPP source code

(5)

/MPP/mpp/legacy: generate new libvpu to include old vpuapi path and new mpp path

(6)

/MPP/mpp/test: mpp internal video protocol unit test and demo

(7)

/MPP/test: mpp buffer/packet component unit test and mpi demo

(8)

/MPP/out: final release binary output directory

/MPP/out/bin: executable binary file output directory

/MPP/out/inc: header file output directory

/MPP/out/lib: library file output directory

(9)

/MPP/osal: Operation System Abstract Layer: abstract layer for different operation system

/MPP/osal/mem: mpi memory subsystem for hardware

/MPP/osal/android: google's android

/MPP/osal/linux: mainline linux kernel

/MPP/osal/window: microsoft's window

/MPP/osal/test: OASL unit test

(10)

/MPP/tools: coding style format tools

(11)
/MPP/utils: small util functions

1.4.2 Interface Process

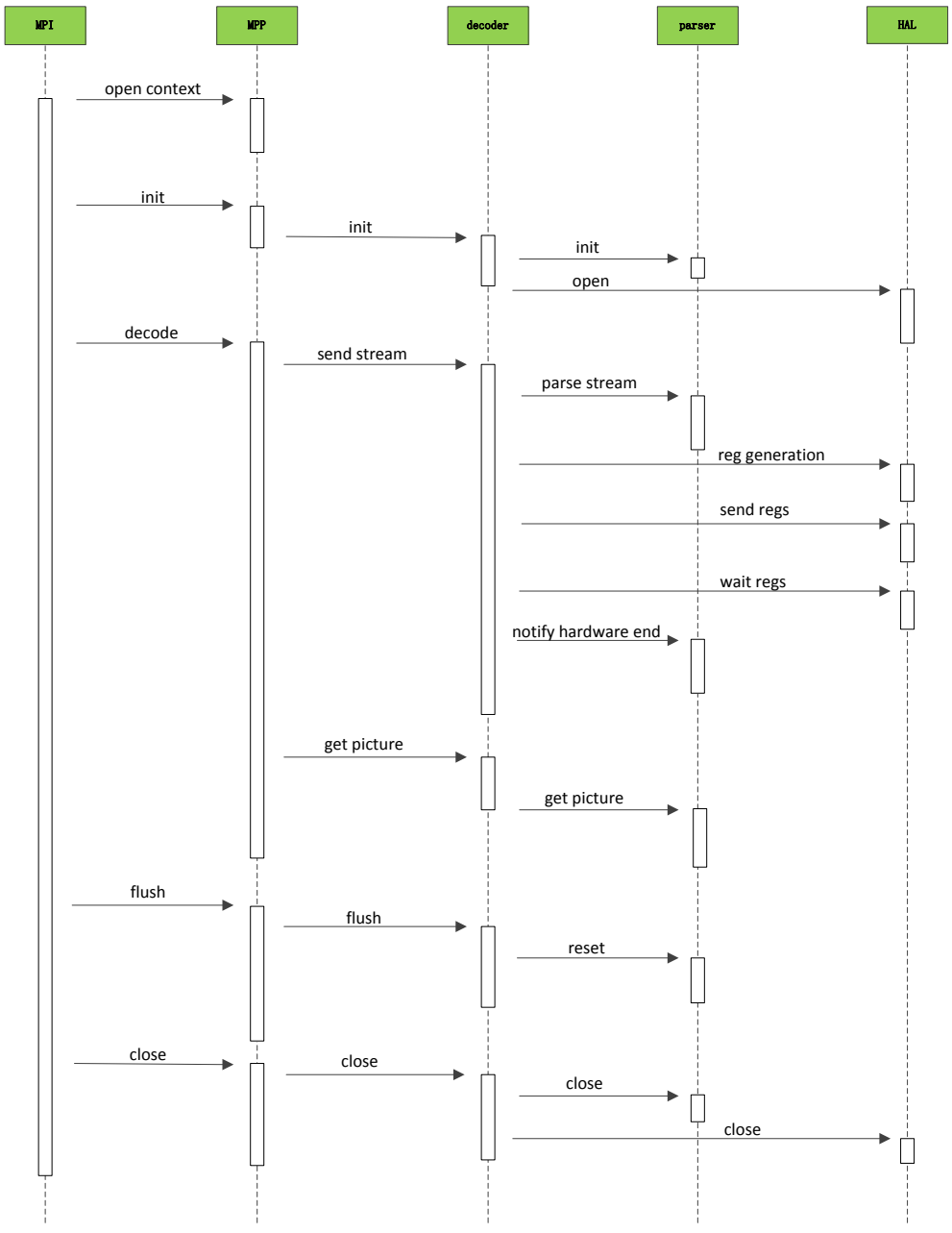

Figure 4 Interface process of the MPP

Take H.264 deocder for example. Video stream will first queued by MPI/MPP layer, MPP will send the stream to codec layer, codec layer parses the stream header and generates a protocol standard output. This output will be send to HAL to generate register file set and communicate with hardware. Hardware will complete the task and resend information back. MPP notify codec by hardware result, codec output decoded frame by display order.

Chatper 2 User Manual

Please see the document named “mpp user manual.pdf”.

Chapter 3 Abbreviations

For the purposes of this documentation, the following abbreviations apply:

MPP Media Process Platform

MPI Media Process Interface

HAL Hardware Abstract Layer

OSAL Operating System Abstract Layer

Chapter 4 API Application Instances

4.1 Encoder Sample

4.1.1 Flow for Encoder

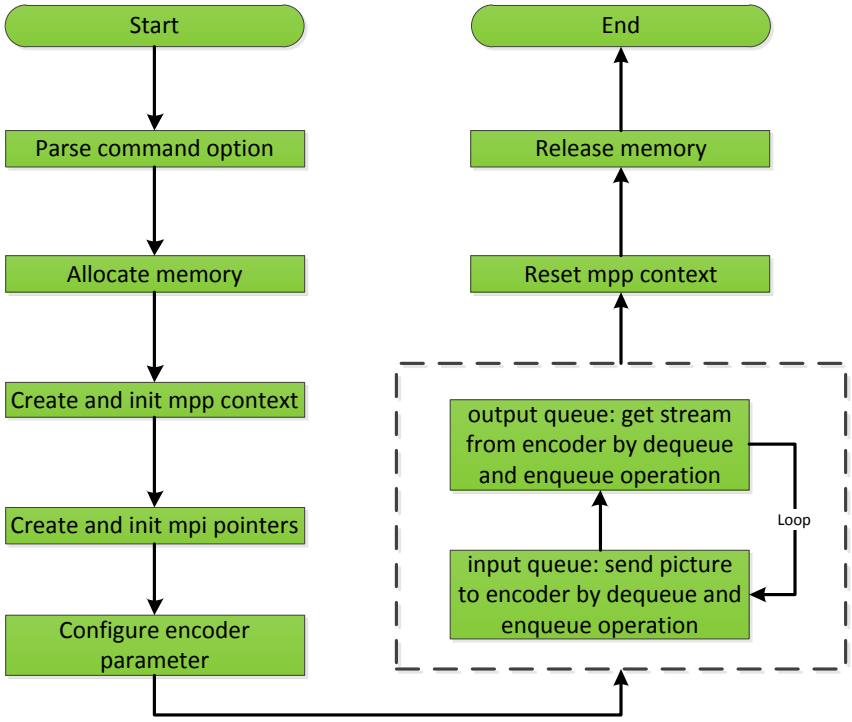

Figure 5 Flow for encoder

Figure 5 is the flow chart of encoder, which details how mpp encoder works. First of all, it parses command option and allocates memory. Then mpp context is created and inited, also mpi pointers. Importantly, encoder parameter should be configured before encoder starts to work. After that, the application sends picture to encoder by dequeue and enqueue operation in input queue while getting stream from encoder by the same operation in output queue. When encoding is done, mpp context must be reset and memory allocated by programmer should be released.

4.1.2 Program Instances

For details about the codes, see mpi_enc_test.c in mpp source code.

4.2 Decoder Sample

4.2.1 Flow for Decoder

In this case for mpp decoder test, as we can see from Figure 6, it parses the command option and allocates memory at first. Then it creates and inits mpp context, also creates and inits mpi pointers. Now there exists two decoder flow called simple decoder and advanced decoder. Only video stream in the format of MJPEG is applied to advanced decoder so far. In simple decoder scene, the application sends video stream packet to decoder by the mpi interface named decode_put_packet and gets video frame from decoder by decode_get_frame function interface. However in advanced case, the application sends stream to decoder

by dequeue and enqueue operation in input queue while getting picture from decoder by dequeue and enqueue operation in output queue. When decoding is done, mpp context must be reset and memory allocated by programmer should be released.

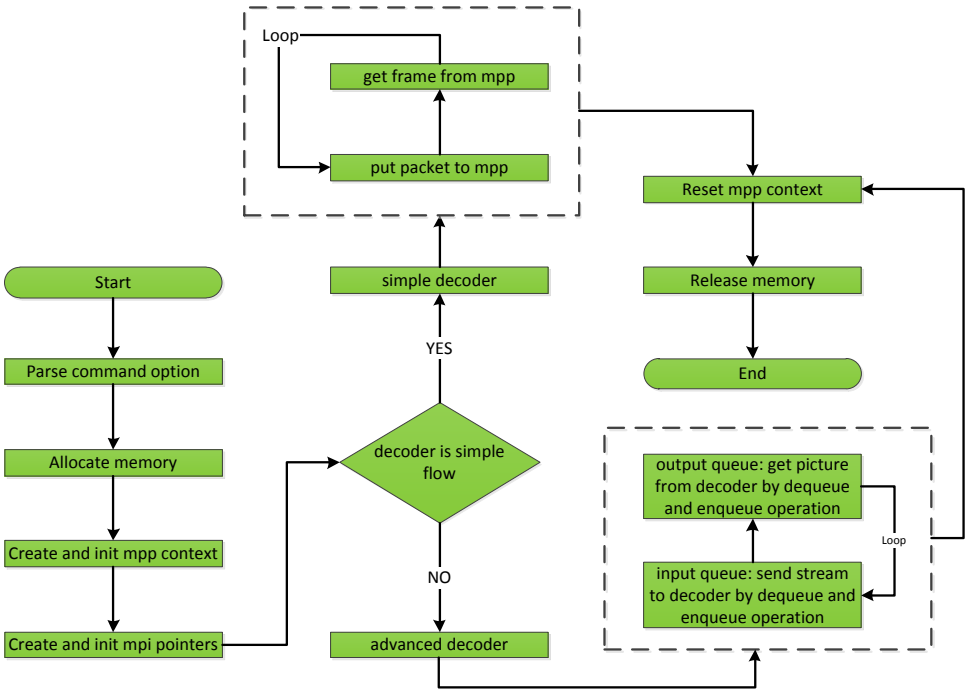

Figure 6 Flow for decoder

4.2.2 Program Instances

For details about the codes, see mpi_dec_test.c in mpp source code.