IPCAM SoC

SN98600

Video Codec Programing Guide

Document No.:

Version: v1.02

Revision	Date	Description
1.00	2014-01-18	Formal Release
1.01	2014-07-04	Update example code path
1.02	2014-09-25	Update Md Rate control
Apply to		

Table of Contents

1 Overview

1.1 SDK Architecture

This SDK includes Video codec (H264 & MJPEG), Scaling function and Codec Data Stamp, supports software bit rate control and frame rate control.

1.2 Video Codec driver

Video codec and ISP are based on Video for Linux 2 (V4L2) memory to memory architecture, Video for Linux 2 use videobuf2 memory allocate method, Sonix Codec driver support videobuf2 DMA contiguous memory allocation (vb2-dma-conting) method and Sonix memory allocation method (snx-vb2). The driver stack diagram as follow. Video codec driver includes H264, MJPEG, Scale down and Codec Data Stamp.

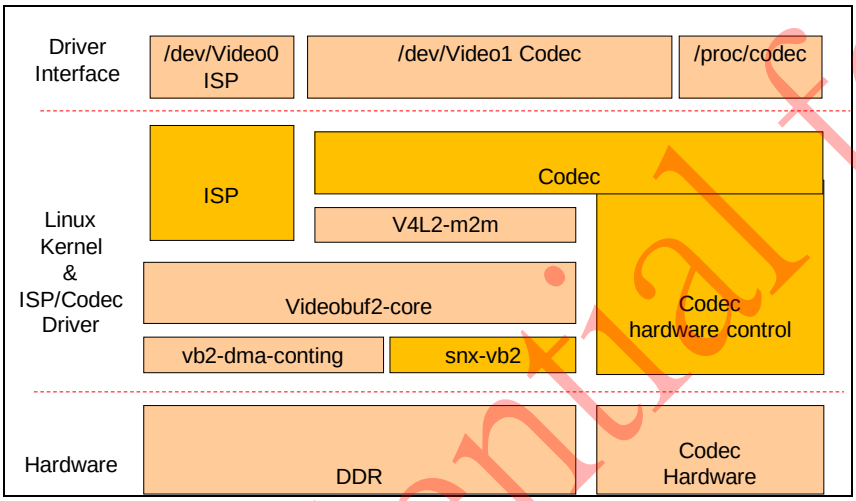

Figure 01. Video Codec SDK Architecture

1.2.1 Codec function

Video codec driver support s H264 and JPEG format output

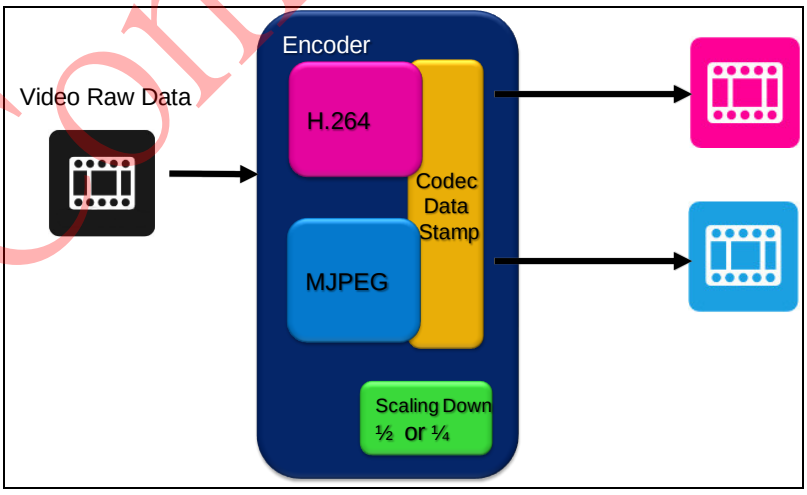

Figure 02. Video Codec Output Format

1.2.2 Scaling function

The scaling function is a build-in feature of the video codec, and it support s1, 1/2 or 1/4 size video scaling at a time. Its output format could be raw, H.264, or JPEG. The figure shows the data path through the scaling function.

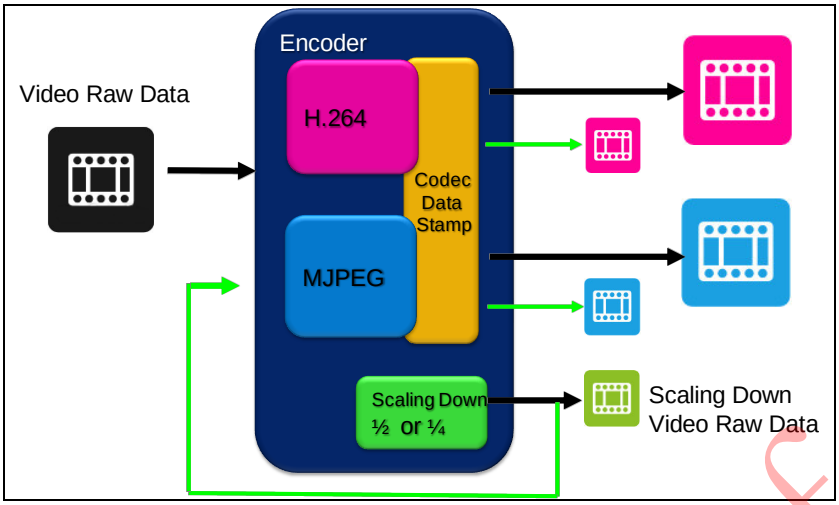

Figure 03. Video Codec Scaling Function

1.2.3 Codec Data Stamp function

The Codec Data Stamp function is a build-in feature of the video codec. Due to the video interleaving, the same stamp would be shown on the main path of H264 and JPEG outputs, but different Codec Data Stamp on the path with the scaling function.

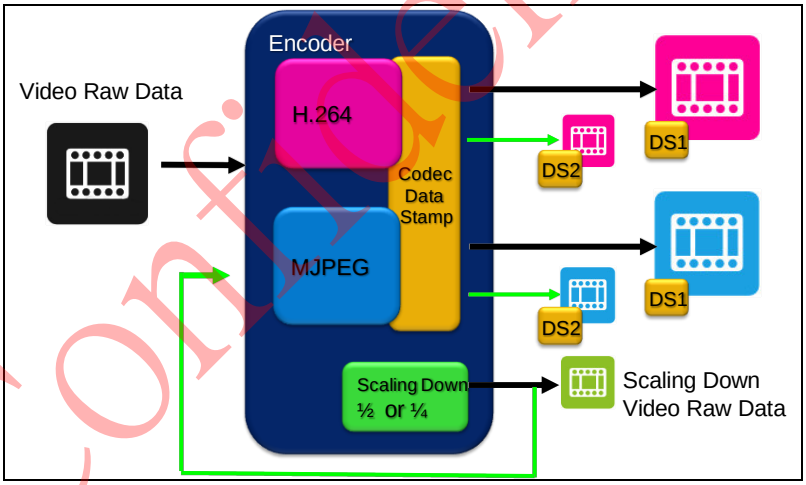

Figure 04. Codec Data Stamp Function

1.2.4 Rate control function

SN986 Series SDK supports the rate control software algorithm for the video codec bit rate and frame rate control.

SN986 Series SDK supports motion drop frame and low lux drop frame function.

Please note that the frame rate encoded from the capture path should not be bigger than the frame rate encoded from the mem2mem path. And the frame rate encoded from the mem2mem path should not be bigger than the frame rate comes from the ISP.

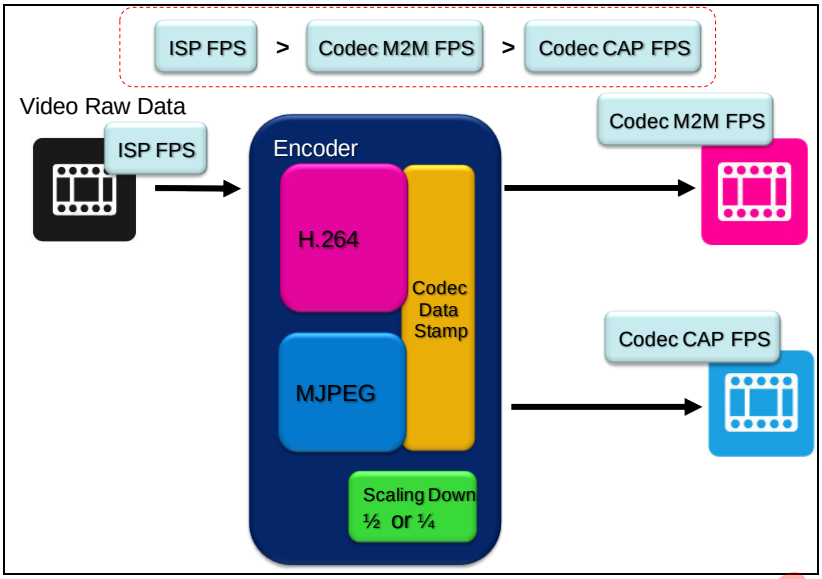

Figure 05. Codec Frame Rate Control Function

1.3 Video Codec driver usage

The video codec driver includes mem2mem path and capture path. M2M path uses Video for Linux 2 memory to memory architecture code flow, which means that the image data from ISP will be copied to the encoder and encoded to the desired format. CAP path uses Video for Linux 2 capture code flow, which means that it would use the data already in the encoder. Before setting a capture path, please make sure one mem2mem path has been made.

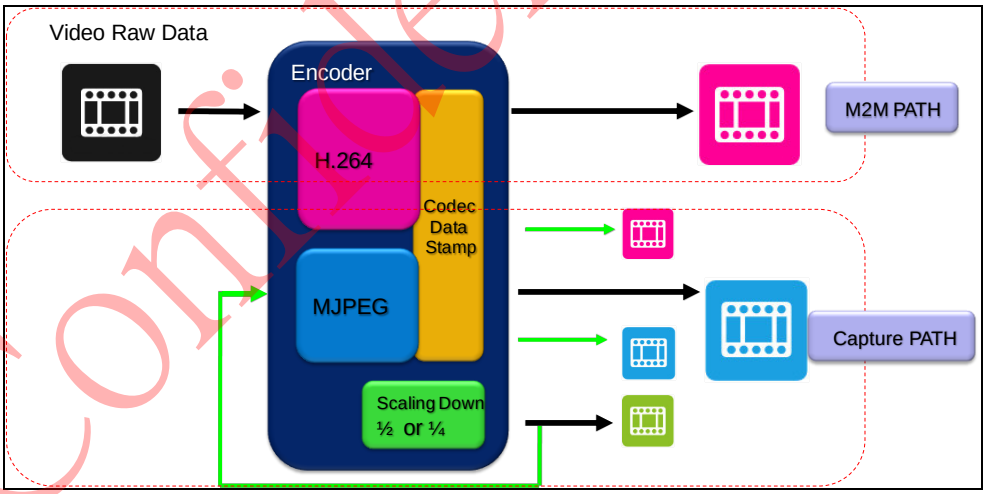

Figure 06. Codec M2M&CAP Path

1.3.1 Video Codec driver initialize usage

ISP and Video codec driver use the pre-allocate memory at boot time. The boot command parameter, “isp”, is ISP driver allocate memory, “vc”, is video codec driver allocate memory.

Example:
bootcmd isp=7M vc=20M

Sonix memory allocation method (snx-vb2) is a memory allocation method and used to reduce the used memory size. It can be set to enable/disable by using the module parameter, “snx_vb2”. When snx_vb2 is enabled, the other parameter, “snx_percent”, can be used to reduce the percentage of the memory size (0% ~100%). Please references

Example:
modprobe snx_vb2;
modprobe snx_vc snx_vb2=1 snx_percent=0;

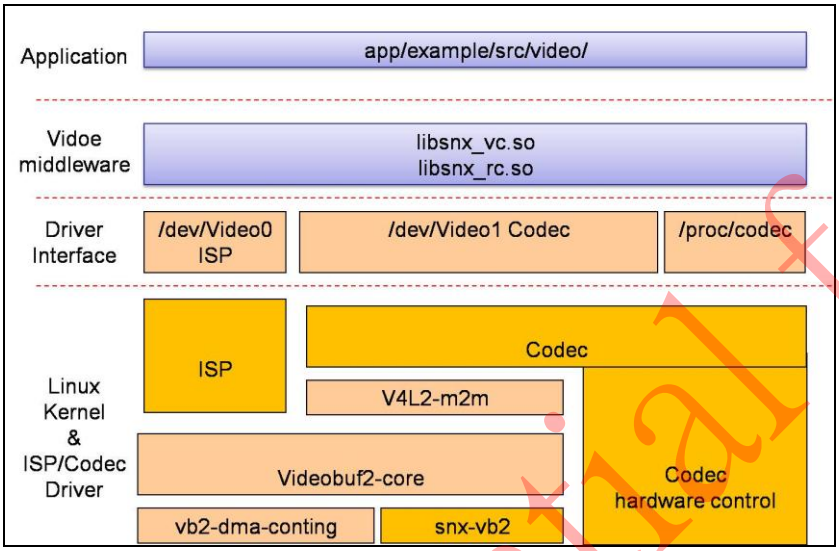

Figure 07. Video Codec SDK Architecture

1.3.2 Video Codec driver memory usage

SN98600_codec_buffer_list_v0.0.xls is used to calculate how much memory the ISP and Codec driver would use in multi-stream paths. Users can fill the streams information and memory size they desired to allocate, then the calculated result would be shown on the table. Warning message would be shown when the desired memory size is not enough.

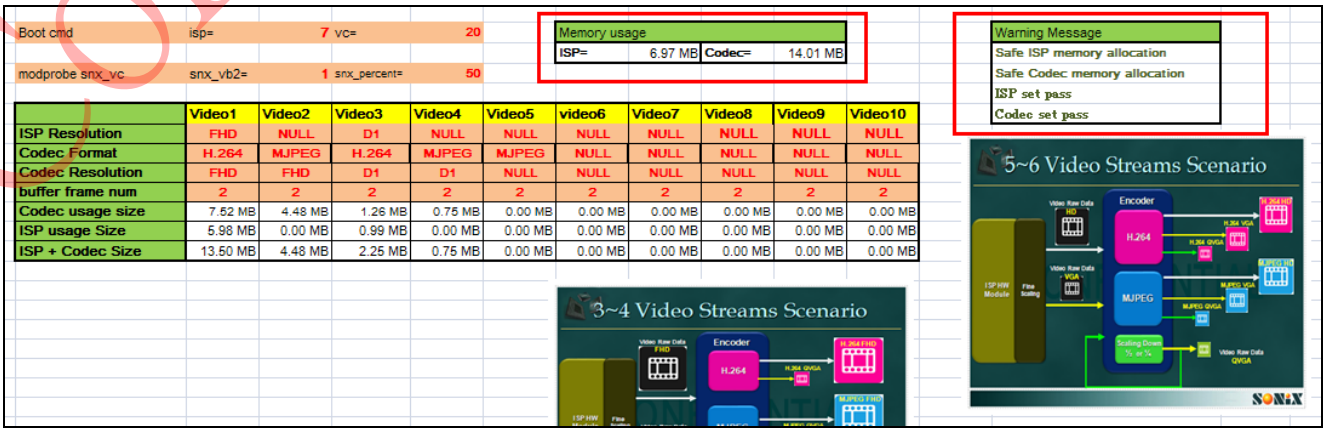

Figure 08. Codec buffer list

2 Middleware Structures and APIs

SN986 Series SDK provides video codec middleware APIs for developers. The codec APIs supported through the Sonix codec library (*libsnx_vc/libsnx_rc)* at *middleware/video/middleware/lib.* SDK also provide some example applications in *app/example/src/video*

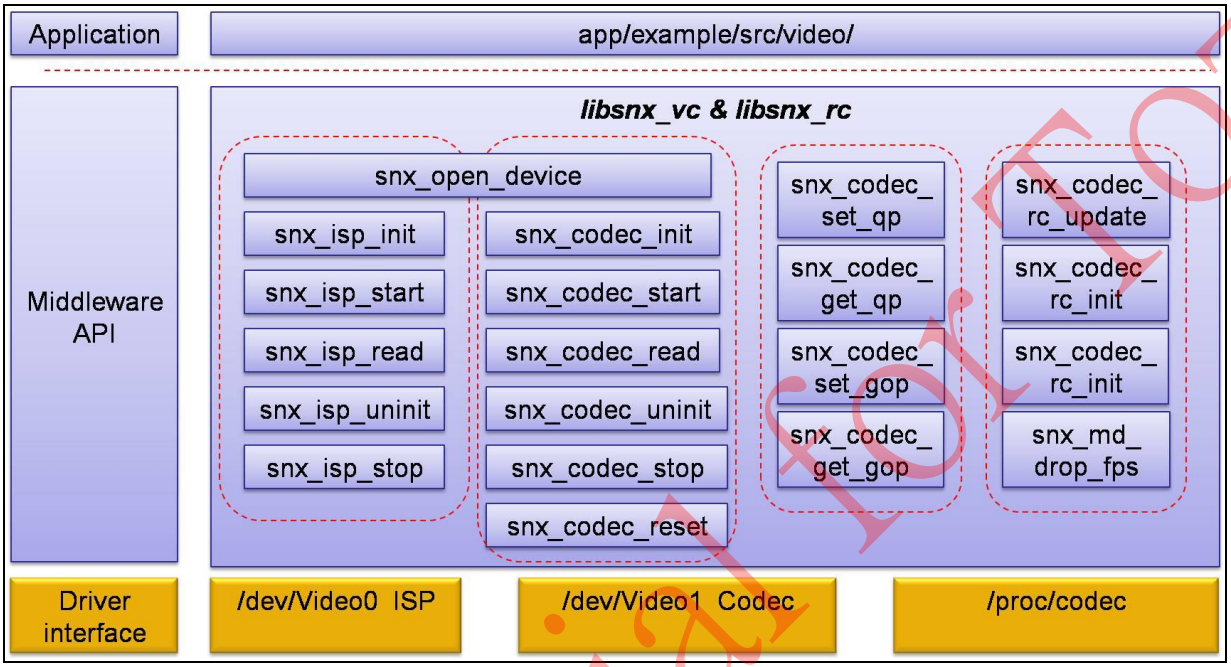

Figure 09. SN986 Series SDK Video codec middleware

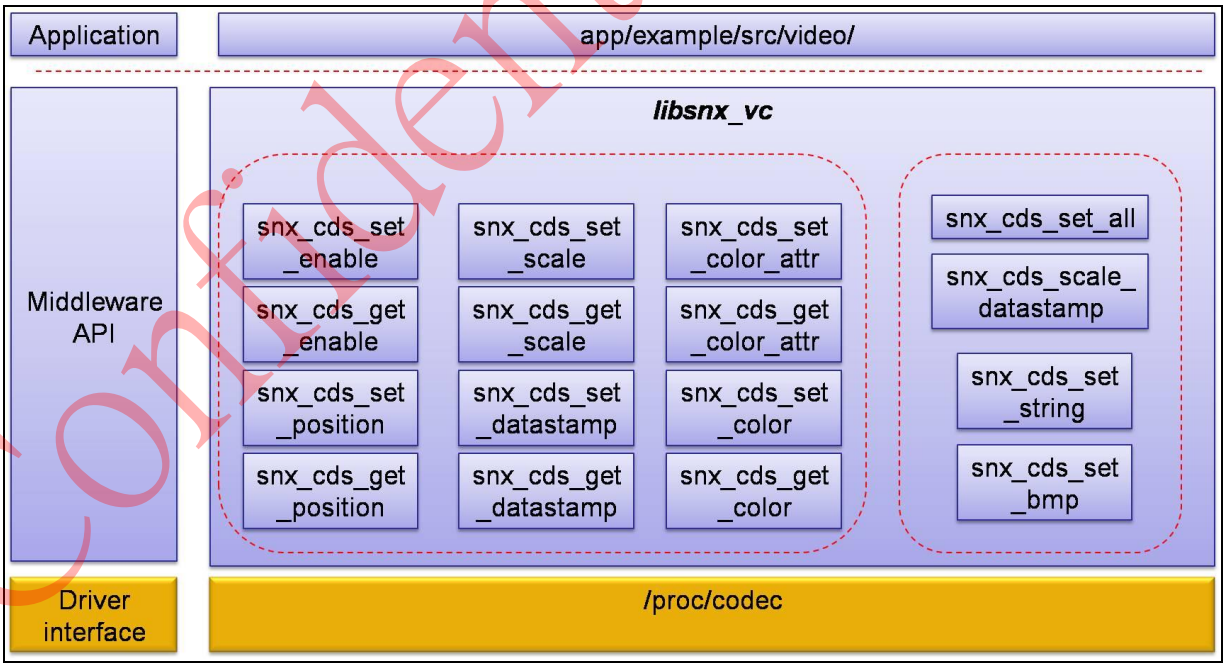

Figure 10. SN986 Series SDK Video codec data stamp middleware

2.1 Codec (H264 & MJPEG) APIs Stucture Definition

The snx_m2m structure is defined for the codec APIs, its detail as follows. The header file is

on *"/middleware/video/middleware/include/snx_vc/snx_vc_lib.h".*

Name	Type	Descriptions
SONiX Set		
m2m	Unsigned int	V4L2 memory to memory mode
socket_fd	Int	sonix proprietary file descript
thread_num	Int	sonix proprietary thread num
i_frame_qp	Int	Set I frame or P frame QP value
scale	unsigned int	scale down set
ds_font_num	unsigned int	Codec data stamp alloction font size number
last	unsigned int	
Basic Set		
isp_dev[12]	char	isp device name
Codec_dev[12]	char	codec device name
ds_dev_name[64]	char	Codec data stamp proc path name
isp_fd	Int	isp file descriptor
codec_fd	Int	codec file descriptor
width	Size_t	input/output frame width
height	Size_t	input/output frame height
isp_fps	Int	Set ISP frame rate (fps)
codec_fps	Int	Set Codec frame rate (fps)
bit_rate	Int	Set H264 output bit rate
qp	Int	H264 and MJPEG quantization parameter (range: H264 10 ~ 52, JPEG 0 ~ 128)
gop	Int	Set group of pictures.
V4L2 Set		
m2m_buffers	Int	v4l2 buffer number
isp_buffers	Struct buffer*	isp alloction buffer pointer
cap_buffers	Struct buffer*	codec capture alloction buffer pointer
isp_index	Int	isp id number of the buffer
cap_index	Int	codec capture id number of the buffer
cap_byteused	Int	number of bytes occupied by data in the plane (payload)
isp_fmt	Unsigned int	Set isp format.
codec_fmt	Unsigned int	Set codec format.
cap_mem	Unsigned int	capture buffer memory type
out_mem	Unsigned int	output buffer memory type
isp_mem	Unsigned int	ISP buffer memory type

Table 01. Snx_m2m Structure Descriptions

The structure, snx_m2m, includes "SONiX set", "Basic set", and "V4l2 set"; "SONiX set" is the parameters for SONiX proprietary setting; "Basic set" is for the codec common parameters;

“V4l2 set” is for V4L2 (Video for Linux 2) standard parameters.

2.2 Codec Data Stamp APIs Stucture Definition

The structure, “snx_cds”, is defined for Codec Data stamp API, and its parameters include, dev_name, bmp_file, string, pos, dim, t_color, b_color, attr, enable, scale.
The header file is on */middleware/video/middleware/include/snx_vc/snx_vc_lib.h.*

Name	Type	Descriptions
dev_name	char [64]	device name
bmp_file	char [64]	Bitmap file name
string	char *	input string
pos	snx_cds_position	postition structure, start_x & start_y
pos->start_x	unsigned int	postition unit is 16 pixel
pos->start_y	unsigned int	postition unit is 16 pixel
dim	snx_cds_dimension	font dimension structure, dim_x & dim_y
dim->dim_x	unsigned int	font dimension horizontal
dim->dim_y	unsigned int	font dimension vertical
t_color	snx_cds_color	foreground color structure, color Y, Cb, Cr
b_color	snx_cds_color	background color structure, color Y, Cb, Cr
t_color->color_Y b_color->color_Y	unsigned int	foregruond color Y & Backgrund color Y
t_color->color_Cb b_color->color_Cb	unsigned int	foregruond color Cb & Backgrund color Cb
t_color->color_Cr b_color->color_Cr	unsigned int	foregruond color Cr & Backgrund color Cr
attr	snx_cds_color_attr	color attrubilt structure
attr->weight	unsigned int	OUT=(DS_WT*TEXT + 8-DS_WT)*IMG)/8
attr->Mode	unsigned int	text (wighting) mode and background (transparent) mode
enable	unsigned int	Enable Codec data stamp
scale	unsigned int	scale down 0: 1x1 1:2x2 2: 4x4

Table 02. snx_cds Descriptions

2.3 Bit Rate Control APIs Stucture definition

The structure, “snx_rc”, is defined for Bit Rate control API. Users can simply control the video bit rate by setting ***targetbitrate*** and ***framerate.***
The header file is on */middleware/video/middleware/include/snx_vc/snx_vc_lib.h.*

Name	Type	Descriptions
codec_fd	int	codec file descriptor
wdth	size_t	frame width
hight	size_t	frame height

gop	unsigned int	Set group of pictures.
frames_gop	int	
Targetbitrate	int	Set the bit rate
rtn_quant	int	
Previous_frame_size	int	
BitRemain	int	
IFrameAve	int	
IntraPreDeviation	int	
FirstPQP	int	
IntraQP	int	
FrameCntRemain	int	
framerate	int	Set the frame rate
total_quant_gop	int	
QpAdjEnable	int	
QpAdjStopCnt	int	
QpDifSum	int	
NFirstGOP	int	
IFrameCnt	int	
CurSecHaveIFrame	int	
TotalBitPerSec	int	
frames	int	
current_frame_size	int	
UpBoundUp	int	
UpBoundDw	int	
ConvergenceRate	int	
QPStopCnt	int	
SumQPBound	int	
UpBoundBytesPerSec	int	
BitExceed	int	
RealBytesPerSec	int	
SumQPCnt	int	
I_frame_size	Int [4]	
bufsize;	int	
Pre_Framerate	int	
reinit	int	
md_flag	int	

Table 03. Snx_rc Descriptions

2.4 Codec (H264 & MJPEG) APIs

Codec APIs integrate V4L2 ioctl functions and v4l2 structures for developers to easily program codec drivers, for example: snx_codec_init() will integrate three V4L2 ioctls, ioctl(VIDIOC_QUERYCAP), ioctl(VIDIOC_S_CTRL) and ioctl(VIDIOC_S_FMT).

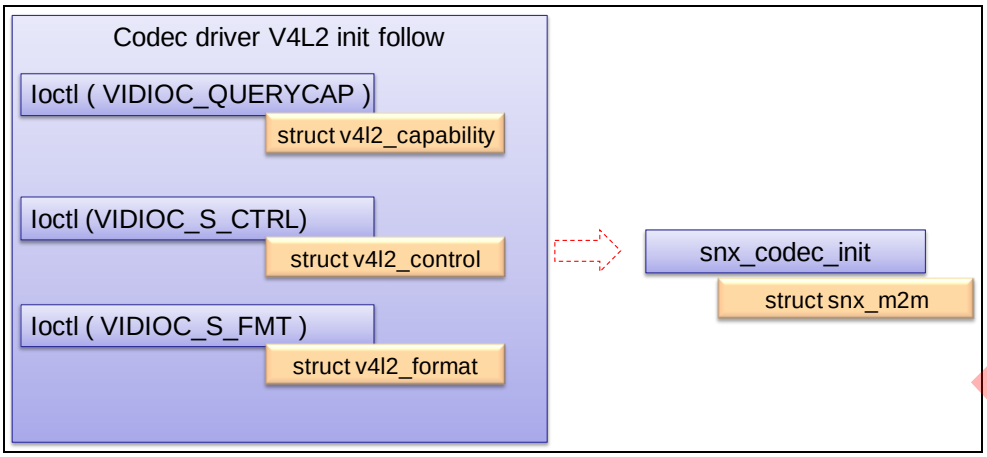

Figure 11. Snx_codec_init Flow

Function Name	Descriptions
snx_open_device(char *dev_name)	Open device name
snx_codec_init(struct snx_m2m * m2m)	Integrate V4L2 I/O control: VIDIOC_QUERYCAP, VIDIOC_S_CTRL and VIDIOC_S_FMT
snx_codec_start(struct snx_m2m * m2m)	Integrate V4L2 I/O control: VIDIOC_REQBUFS, VIDIOC_QUERBUF, VIDIOC_QBUF, VIDIOC_STREAMON
snx_codec_read(struct snx_m2m * m2m)	Integrate V4L2 I/O control: VIDIOC_DQBUF, VIDIOC_QBUF
snx_codec_reset(struct snx_m2m * m2m)	Integrate V4L2 I/O control: VIDIOC_DQBUF, include snx_isp_read() API
snx_codec_stop(struct snx_m2m * m2m)	Integrate V4L2 I/O control: VIDIOC_STREAMOFF
snx_codec_uninit(struct snx_m2m * m2m)	release memory

Table 04. Codec APIs Descriptions

2.5 Codec Data Stamp APIs

Function Name	Descriptions
snx_cds_get_enable(char *dev_name, int enable)	Get codec data stamp enable/disable status
snx_cds_set_enable(char *dev_name, int *enable)	Set codec data stamp enable/disable
snx_cds_get_scale(char *dev_name, SCALING_UP_RATIO *scaling_up_ration)	Get codec data stamp scale ratio status
snx_cds_set_scale(char *dev_name,	Set codec data stamp scale ratio

SCALING_UP_RATIO scaling_up_ration)	
snx_cds_get_color_attr(char *dev_name, COLOR_ATTR *color_attr)	Get codec data stamp color attrubilt status
snx_cds_set_color_attr(char *dev_name, COLOR_ATTR *color_attr)	Set codec data stamp color attrubilt
snx_cds_get_color(char *dev_name, COLOR *forg_color, COLOR *bkg_color)	Get codec data stamp foreground/background color status
snx_cds_set_color(char *dev_name, COLOR *forg_color, COLOR *bkg_color)	Set codec data stamp foreground/background color
snx_cds_get_position(char *dev_name, POSITION *position, DIMEMSION *dimension)	Get codec data stamp position/dimension status
snx_cds_set_position (char *dev_name, POSITION *position, DIMEMSION *dimension)	Set codec data stamp position/dimension
snx_cds_get_datastamp(char *dev_name, char *data, int len)	Get codec data stamp value
snx_cds_set_datastamp(char *dev_name, char *data, int len)	Set codec data stamp value
snx_cds_set_string(char *dev_name, char *data, SCALING_UP_RATIO *scaling_up_ration)	Set string to codec data stamp
snx_cds_set_bmp(struct snx_cds)	Set bitmap file to codec data stamp
snx_cds_set_all(char *dev_name, struct snx_cds *cds)	Set full codec data stamp setting

Table 05. Codec data Stamp API Descriptions

2.6 Rate Control APIs

Function Name	Descriptions
snx_codec_rc_init(struct snx_rc *rc, int reinit)	Rate control function init function, rcinit set SNX_RC_INIT
snx_codec_rc_update(int current_frame_size, struct snx_rc *rc)	software algorithm calculator new QP value
snx_md_drop_fps (struct snx_rc *rc, int *force_i_frame)	Motion detection drop frame function

Table 06. Rate Control API Descriptions

3 Middleware Usage Descriptions

This chapter descripts the usage of video codec middleware APIs, including the structure parameters, code flow, and some sample code for reference. For more detail examples, please refer to files below.

Function	Sample Code Path
H.264 Encode	*/app/example/src/video/snx_m2m_one_stream.c*
MJPEG Encode	*/app/example/src/video/snx_m2m_one_stream.c*
Scaling	*/app/example/src/video/snx_m2m_capture_4stream.c*
Stamp	*/app/example/src/video/snx_vc_ds.c*
Bit Rate Control	*/app/example/src/video/snx_m2m_one_stream_with_rc.c*
Frame Rate Control	*/app/example/src/video/snx_m2m_capture_4stream.c*
MD drop frame control	*/app/example/src/video/snx_m2m_one_stream_with_rc.c*

Table 07. Video Codec Middleware Sample Code Path

3.1 Before Starting

Before starting to use video codec middleware APIs, there are some parameters should be set for initialization.

Parameters	Structure	Value
m2m	snx_m2m	0: Capture. Without memory to memory path. 1: M2M
codec_fmt	snx_m2m	V4L2_PIX_FMT_H264 V4L2_PIX_FMT_MJPEG V4L2_PIX_FMT_ SNX420
isp_dev	snx_m2m	Set isp device name path
codec_dev	snx_m2m	Set codec device name path
isp_mem	snx_m2m	V4L2_MEMORY_MMAP (Only when m2m = 1)
isp_fmt	snx_m2m	V4L2_PIX_FMT_SNX420 (Only when m2m = 1)
scale	snx_m2m	1: by pass 2: 1/2 scale down 4: 1/4 scale down
isp_fps	snx_m2m	Up to 30, according to the sensor input frame rate. (Only when m2m = 1)
codec_fps	snx_m2m	Desired encoded frame rate, up to 30 (30 fps)
width	snx_m2m	1280 (HD resolution). The encoded frame width.
height	snx_m2m	720 (HD resolution). The encoded frame height.
m2m_buffers	snx_m2m	2 (V4L2 buffer number)

Table 08. Snx_m2m Structure Initial Data

3.1.1 Functions Settings

For different encoding scenarios, there are different settings should be set before using Video codec middleware APIs.

Functions	Parameters	Structure	Value
H.264 Encode	codec_fmt	snx_m2m	V4L2_PIX_FMT_H264
MJPEG Encode	codec_fmt	snx_m2m	V4L2_PIX_FMT_MJPEG
Scaling	scale	snx_m2m	set scale down ratio 1, 2 = 1/2, 4 = 1/4 Scaling function only support one scale size output at a time
Bit Rate Control	width	snx_rc	Frame width from codec frame width
	height	snx_rc	Frame height from codec frame height
	codec_fd	snx_rc	File descriptor from Codec file descriptor
	gop	snx_rc	GOP from codec gop
	Targetbitrate	snx_rc	target bit rate
	framerate	snx_rc	Frame rate from codec frame rate
Frame Rate Control	codec_fps	snx_m2m	Desired codec output frame rate
	isp_fps	snx_m2m	ISP output frame rate
Codec Data Stamp	ds_dev_name	snx_m2m	Get codec data stamp proc path
	ds_font_num	snx_m2m	Set codec data stamp font number size
	enable	snx_cds	Enable data stamp
	scale	snx_cds	scale down 0: 1x1 1:2x2 2: 4x4
	dev_name	snx_cds	/proc/codec/[01]_[hj][s] 0: ISP0 1: ISP1 h: H.264 j: MJPEG s: scaling
	string	snx_cds	Input string (string and bitmap select one)
	bmp_file	snx_cds	Input bitmap file (string and bitmap select one)
	attr	snx_cds	0: text(wighting) + background(transparent) 1: text(wighting) + background 2: text + background(transparent) 3: text + background
	pos	snx_cds	Set position
	dim	snx_cds	Set dimension
	t_color	snx_cds	Set text color
	B_color	snx_cds	Set background color
Functions	**File**	**Folder**	**Value**
MD drop frame	Md_ratectl_en	/etc/rc/	motion detection rate contorl enable (default 1)

Table 09. Function Settings

3.2 Video Encoding Code Flow

3.2.1 V4L2 Memory to Memory (M2M) Architecture Functions Code Flow.

After snx_m2m structure initialized, call the functions to open and initialize the isp device, including *snx_open_device(isp)*, *snx_isp_init,* and *snx_isp_start.* Then call the codec initial functions, *snx_open_device(codec)* and *snx_codec_init.* Now, we start to encode by calling *snx_codec_start*.

When all the initializations are done, the M2M architecture starts working, and developers can get the bit stream of one encoded frame by calling *snx_codec_read.* Call *snx_codec_reset* after one frame read to be ready for the next frame encoding.

To end the video encoding, call the functions, *snx_codec_stop, snx_codec_uninit, snx_isp_stop, snx_isp_uninit, close (codec) and close (isp)*, to stop and close the devices.

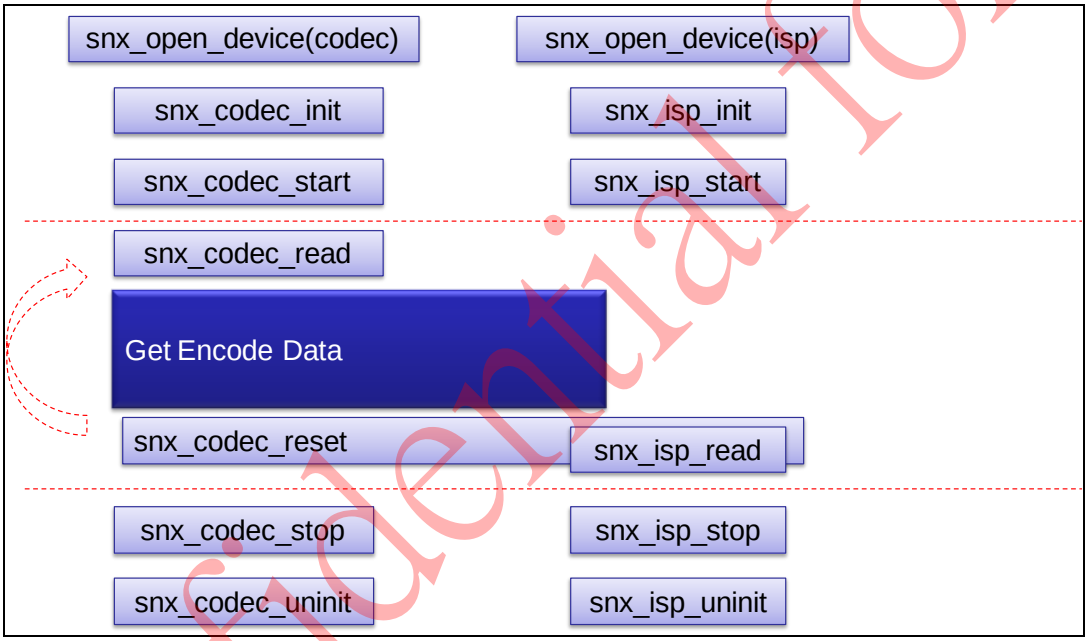

Figure 12.Video Encoding Code Flow (M2M)

3.2.2 Video Encoding M2M Sample Code (H.264)

```
m2m->m2m = 1;
m2m->isp_mem = V4L2_MEMORY_MMAP;
m2m->isp_fmt = V4L2_PIX_FMT_SNX420;
m2m->codec_fmt = V4L2_PIX_FMT_H264;
m2m->scale = SCALE;
m2m->isp_fps = FRAME_RATE;
m2m->codec_fps = FRAME_RATE;
m2m->width = WIDTH;
m2m->height = HEIGHT;
```

```
m2m->m2m_buffers = 2;
m2m->out_mem = V4L2_MEMORY_USERPTR;
 m2m->ds_font_num = 128;
strcpy(m2m->isp_dev,IMG_DEV_NAME);
strcpy(m2m->codec_dev,CAP_DEV_NAME);

m2m->isp_fd = snx_open_device(2m->isp_dev);
snx_isp_init(m2m);
snx_isp_start(m2m);

m2m->codec_fd = snx_open_device(m2m->codec_dev);
snx_codec_init(m2m);
snx_codec_start(m2m);

while(1)
{
      snx_codec_read(m2m);
      if(m2m->cap_bytesused != 0) {
            // Read Frame Data
            // Frame len =    m2m->cap_bytesused;
            // Frame point = m2m->cap_buffers[m2m->cap_index].start
            frame_num--;
      }
      snx_codec_reset(m2m);
      if(frame_num == 0)
      break;
}

snx_codec_stop(m2m);
snx_codec_uninit(m2m);
snx_isp_stop(m2m);
snx_isp_uninit(m2m);
close(m2m->codec_fd);
close(m2m->isp_fd);
```

3.2.3 V4L2 Capture Architecture Functions Code Flow.

After snx_m2m structure initialized, call the codec initial functions, *snx_open_device(codec)* and *snx_codec_init.* No isp initialization functions are needed. Now, we start to encode by calling *snx_codec_start*.

When all the initializations are done, the M2M architecture starts working, and developers can get the bit stream of one encoded frame by calling *snx_codec_read.* Call *snx_codec_reset*

after one frame read to be ready for the next frame encoding.

To end the video encoding, call the functions, *snx_codec_stop, snx_codec_uninit, and close (codec)*, to stop and close the devices.

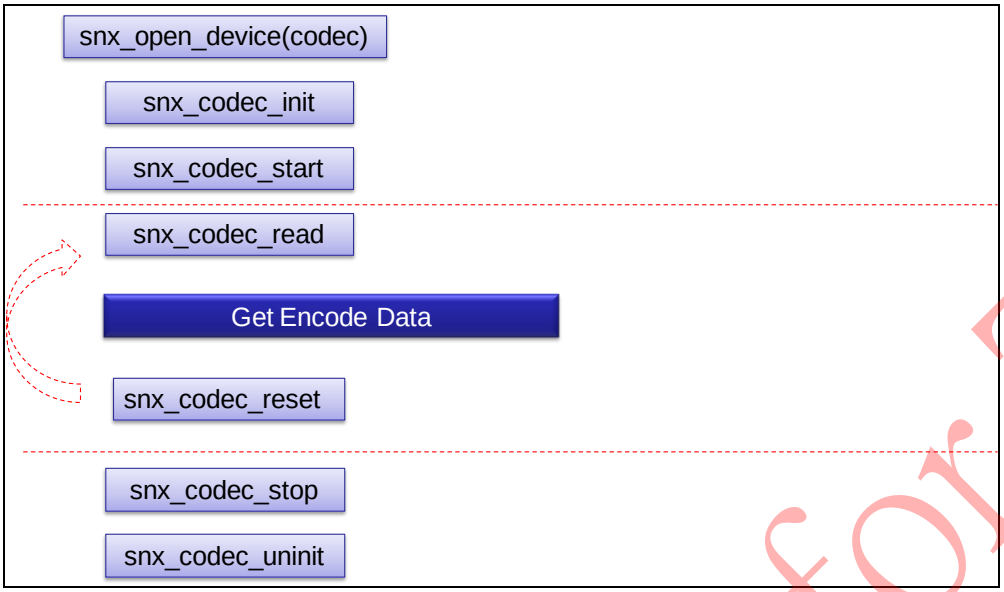

Figure 10. Video Encoding Code Flow (Capture)

3.2.4 Video Encoding Capture Sample Code (H.264)

```
m2m->m2m = 0;
m2m->codec_fmt = V4L2_PIX_FMT_H264;
m2m->scale = SCALE;
m2m->isp_fps = FRAME_RATE;
m2m->codec_fps = FRAME_RATE;
m2m->width = WIDTH;
m2m->height = HEIGHT;
m2m->m2m_buffers = 2;
m2m->out_mem = V4L2_MEMORY_USERPTR;
 m2m->ds_font_num = 128;

strcpy(m2m->isp_dev,IMG_DEV_NAME);
strcpy(m2m->codec_dev,CAP_DEV_NAME);

m2m->codec_fd = snx_open_device(m2m->codec_dev);
snx_codec_init(m2m);
snx_codec_start(m2m);

while(1)
{
      snx_codec_read(m2m);
```

```
        if(m2m->cap_bytesused != 0) {
            // Read Frame Data
            // Frame len =    m2m->cap_bytesused;
            // Frame point = m2m->cap_buffers[m2m->cap_index].start
            frame_num--;
        }
        snx_codec_reset(m2m);
        if(frame_num == 0)
        break;
    }

    snx_codec_stop(m2m);
    snx_codec_uninit(m2m);
    close(m2m->codec_fd);
```

3.2.5 Video Encoding M2M Sample Code (MJPEG with 1/2 scaling)

```
m2m->m2m = 1;
 m2m->isp_mem = V4L2_MEMORY_MMAP;
 m2m->isp_fmt = V4L2_PIX_FMT_SNX420;
 m2m->codec_fmt = V4L2_PIX_FMT_ MJPEG;
 m2m->scale = 2; // Scaling rate need related with capture flow
 m2m->isp_fps = FRAME_RATE;
 m2m->codec_fps = FRAME_RATE;
 m2m->width = WIDTH;
 m2m->height = HEIGHT;
 m2m->m2m_buffers = 2;
 m2m->out_mem = V4L2_MEMORY_USERPTR;
  m2m->ds_font_num = 128;

 strcpy(m2m->isp_dev,IMG_DEV_NAME);
 strcpy(m2m->codec_dev,CAP_DEV_NAME);

 m2m->isp_fd = snx_open_device(2m->isp_dev);
 snx_isp_init(m2m);
 snx_isp_start(m2m);

 m2m->codec_fd = snx_open_device(m2m->codec_dev);
 snx_codec_init(m2m);
 snx_codec_start(m2m);

 while(1)
 {
```

```
        snx_codec_read(m2m);
        if(m2m->cap_bytesused != 0) {
            // Read Frame Data
            // Frame len =    m2m->cap_bytesused;
            // Frame point = m2m->cap_buffers[m2m->cap_index].start
            frame_num--;
        }
        snx_codec_reset(m2m);
        if(frame_num == 0)
        break;
    }
    snx_codec_stop(m2m);
    snx_codec_uninit(m2m);
    snx_isp_stop(m2m);
    snx_isp_uninit(m2m);
    close(m2m->codec_fd);
    close(m2m->isp_fd);
```

3.2.6 Video Encoding Capture Sample Code (MJPEG with Frame Rate Control)

```
    m2m->m2m = 0;
    m2m->codec_fmt = V4L2_PIX_FMT_ MJPEG;
    m2m->scale = SCALE; // Scaling rate need related with M2M flow
    m2m->isp_fps = ISP_FRAME_RATE;
    m2m->codec_fps = CODEC_FRAME_RATE;
    m2m->width = WIDTH;
    m2m->height = HEIGHT;
    m2m->m2m_buffers = 2;
    m2m->out_mem = V4L2_MEMORY_USERPTR;
     m2m->ds_font_num = 128;
    strcpy(m2m->isp_dev,IMG_DEV_NAME);
    strcpy(m2m->codec_dev,CAP_DEV_NAME);

    m2m->codec_fd = snx_open_device(m2m->codec_dev);
    snx_codec_init(m2m);

    rc->width = m2m->width;
    rc->height = m2m->height;
    rc->codec_fd = m2m->codec_fd;
    rc->Targetbitrate = m2m->bit_rate;
    rc->framerate = m2m->isp_fps;
    rc->gop = m2m->codec_fps;
```

```
if(m2m->codec_fmt == V4L2_PIX_FMT_H264) {
        m2m->qp = snx_codec_rc_init(rc , SNX_RC_INIT);
}
snx_codec_start(m2m);

while(1)
{
        snx_codec_read(m2m);
        if(m2m->cap_bytesused != 0) {
                if(m2m->codec_fmt == V4L2_PIX_FMT_H264) {
                        m2m->qp = snx_codec_rc_update(m2m->cap_bytesused, rc);
                        snx_md_drop_fps(rc, &m2m->force_i_frame);
                }
                // Read Frame Data
                // Frame len =    m2m->cap_bytesused;
                // Frame point = m2m->cap_buffers[m2m->cap_index].start
                frame_num--;
        }
        snx_codec_reset(m2m);

        if(frame_num == 0)
        break;
}

snx_codec_stop(m2m);
snx_codec_uninit(m2m);
close(m2m->codec_fd);
```

3.3 Bit Rate Control Code Flow

The bit rate control implements the software algorithm of constant bit rate (CBR).
Motion detection drop frame function, that used drop frame method to get good image quality.

3.3.1 V4L2 Memory to Memory (M2M) Architecture Functions Code Flow.

The bit rate control functions code flow is almost the same as the video encoding but pluses the rate control functions.

After snx_m2m structure initialized, call the functions to open and initialize the isp device, including *snx_open_device(isp)*, *snx_isp_init, snx_isp_start.* Then call the codec initial functions, *snx_open_device(codec), snx_codec_init.*. Call the rate control initial functions, snx_codec_rc_init, and snx_codec_set_qp. Now, we start to encode by calling *snx_codec_start*.

When all the initializations are done, the M2M architecture starts working, and developers can get the bit stream of one encoded frame by calling *snx_codec_read.* According to the current encoded frames, the rate control would change the compression quality dynamically by calling *snx_codec_rc_update* and *snx_codec_set_qp*. *Use snx_md_drop_fps to drop frame(option), snx_md_drop_fps config file on /etc/rc/.* Call *snx_codec_reset* after one frame read to be ready for the next frame encoding.

To end the video encoding, call the functions, *snx_codec_stop, snx_codec_uninit, snx_isp_stop, snx_isp_uninit, close (codec) and close (isp)*, to stop and close the devices.

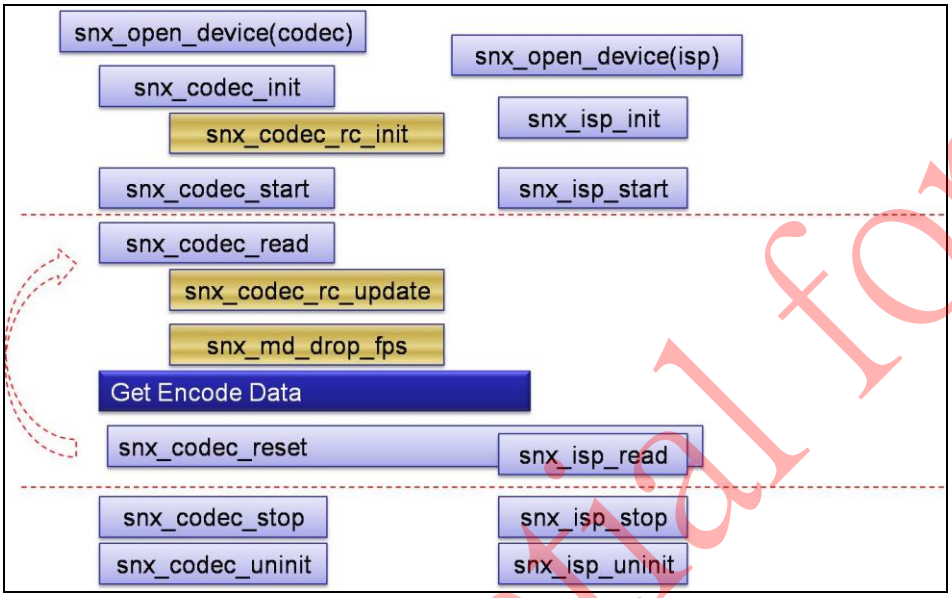

Figure 13. Video Encoding with Bit Rate Control Code Flow (M2M)

3.3.2 V4L2 Capture Architecture Functions Code Flow.

The bit rate control functions code flow is almost the same as the video encoding but pluses the rate control functions.

After snx_m2m structure initialized, call the codec initial functions, *snx_open_device(codec), snx_codec_init.*. Call the rate control initial functions snx_codec_rc_init. No isp initialization functions are needed. Now, we start to encode by calling *snx_codec_start*.

When all the initializations are done, the M2M architecture starts working, and developers can get the bit stream of one encoded frame by calling *snx_codec_read.* According to the current encoded frames, the rate control would change the compression quality dynamically by calling snx_codec_rc_update. *Use snx_md_drop_fps to drop frame(option), snx_md_drop_fps config file on /etc/rc/.* Call *snx_codec_reset* after one frame read to be ready for the next frame encoding.

To end the video encoding, call the functions, *snx_codec_stop, snx_codec_uninit, and close*

(codec), to stop and close the devices.

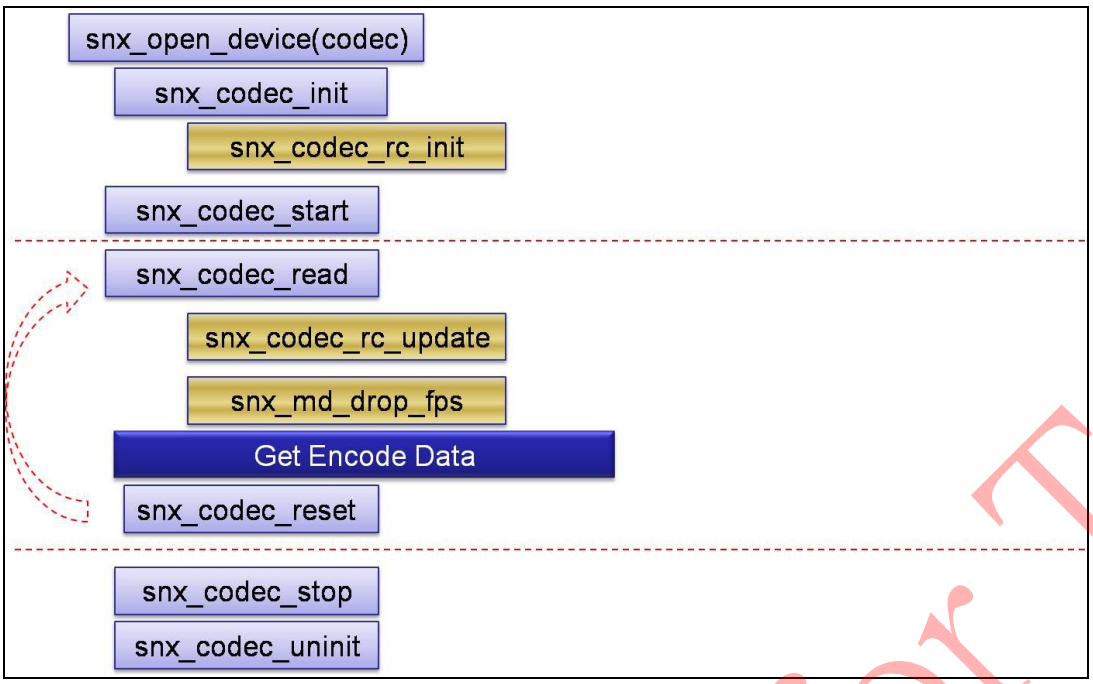

Figure 14.Video Encoding with Bit Rate Control Code Flow (Capture)

3.3.3 Video Encoding Capture Sample Code with Bitrate Control

```
m2m->m2m = 0;
m2m->codec_fmt = V4L2_PIX_FMT_ MJPEG;
m2m->scale = SCALE; // Scaling rate need related with M2M flow
m2m->isp_fps = FRAME_RATE;
m2m->codec_fps = FRAME_RATE;
m2m->width = WIDTH;
m2m->height = HEIGHT;
m2m->m2m_buffers = 2;
m2m->ds_font_num = 128;
m2m->out_mem = V4L2_MEMORY_USERPTR;
strcpy(m2m->isp_dev,IMG_DEV_NAME);
strcpy(m2m->codec_dev,CAP_DEV_NAME);

m2m->codec_fd = snx_open_device(m2m->codec_dev);
snx_codec_init(m2m);

rc->width = m2m->width;
rc->height = m2m->height;
rc->codec_fd = m2m->codec_fd;
rc->Targetbitrate = m2m->bit_rate;
rc->framerate = m2m->isp_fps;
rc->gop = m2m->codec_fps;
```

```
if(m2m->codec_fmt == V4L2_PIX_FMT_H264) {
        m2m->qp = snx_codec_rc_init(rc , SNX_RC_INIT);
}
snx_codec_start(m2m);

while(1)
{
        snx_codec_read(m2m);
        if(m2m->cap_bytesused != 0) {
                if(m2m->codec_fmt == V4L2_PIX_FMT_H264) {
                        m2m->qp = snx_codec_rc_update(m2m->cap_bytesused, rc);
                        snx_md_drop_fps(rc, &m2m->force_i_frame);
                }
                frame_num--;
        }
        snx_codec_reset(m2m);
        if(frame_num == 0)
        break;
}

snx_codec_stop(m2m);
snx_codec_uninit(m2m);
close(m2m->codec_fd);
```

3.4 Stamp Functions Code Flow

After snx_ds structure initialized, developers can set the format of stamps they want. Then call the functions to enable the settings. The video data stamp support 16x16 pixels per unit. Developers not only can use the function, “snx_cds_set_bmp()”, to set the bmp picture (bitmap file width and height must be in multiples of 16 pixel), but also use the function, “snx_cds_set_string()”, to set the string data on the screen.

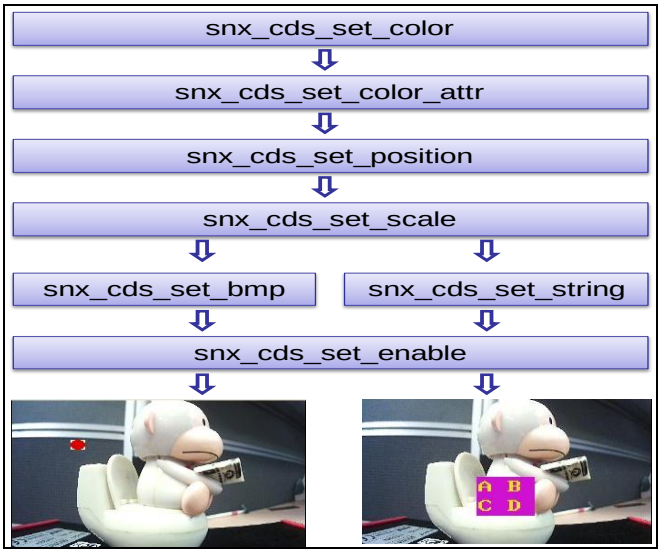

Figure 15. Stamp Functions Code Flow

3.4.1 Stamp Functions Sample Code

```
sprintf(cds->dev_name, "/proc/codec/1_h");
cds->enable=1;
cds->t_color->color_Y = 106;
cds->t_color->color_Cb = 202;
cds->t_color->color_Cr = 221;
cds->b_color->color_Y = 40;
cds->b_color->color_Cb = 240;
cds->b_color->color_Cr = 109;
cds->attr.mode=3;
cds->attr.weight=1;
cds->scale = 1;
sprintf(cds->bmp_file, "bitmap.bmp");
strcpy(cds->string, "ABCD");
cds->pos.start_x=1;
cds->pos.start_y=1;
cds->dim.dim_x=2;
cds->dim.dim_y=2;

snx_cds_set_color(cds->dev_name, &cds->t_color, &cds->b_color);
snx_cds_set_color_attr(cds->dev_name, &cds->attr);
snx_cds_set_position(cds->dev_name, &cds->pos, &cds->dim);
snx_cds_set_scale(cds->dev_name, cds->scale);
if(bmp==1)
    snx_cds_set_bmp(cds);
else
    snx_cds_set_string(cds->dev_name, str, (SCALING_UP_RATIO *)cds->scale);
snx_cds_set_scale(cds->dev_name, cds->scale);
```

```
snx_cds_set_enable(cds->dev_name, cds->enable);
```

3.4.2 Stamp Proc Flow Sample Code

Setting data stamp through the proc interface of Linux is also available. Please refer to the following sample code.

```
echo 0X00000000 0X007C0000 0X00E600C6 0X00DE00F6 0X00C600CE 0X007C00C6
0X00000000 0X00000000 > /proc/codec/0_h/ds_datastamp
echo 0 > /proc/codec/0_h/ds_scale
echo 0x2 0x3 > /proc/codec/0_h/ds_position
echo 0x5 0x2 > /proc/codec/0_h/ds_dimension
echo 1 > /proc/codec/0_h/ds_enable
echo 144 53 34 40 240 109 > /proc/codec/0_h/ds_color
echo 6 2 > /proc/codec/0_h/ds_color_attr
```