Multi-Channel Video Input Port (MCVIP)

Document Revision 1.01

TVP5158 Software Driver User Guide

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards ought to be provided by the customer so as to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is neither responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Revision History

Version	Date	Revision History
1.00	18 Jan 2010	First Draft
1.01	19 Jan 2010	Updated based on review comments

TABLE OF CONTENTS

1 Introduction

1.1 Overview

TI TVP5158 video decoder is a multi-channel video decoder with capability to send multiple channels of video streams over a single 8-bit/16-bit BT656/BT1120 embedded sync interface.

For TVP5158 device specific information refer to TVP5158 data sheet. (http://focus.ti.com/docs/prod/folders/print/tvp5158.html)

This document explains the details of the TVP5158 software device driver that is used by applications, running on a host processor like DM365, DM6467, to capture and process multiple channels of video streams.

This document explains the multi-channel driver interface (MCVIP) that is used by applications to get the video data. This document also provides information about top-level design, host platform specific details like performance, system data-flows, source code information etc.

1.2 Software Driver : Features Supported

- All line-multiplexed video capture modes like the below are supported by the driver,
    - 2CH D1 (BT656)
    - 4CH D1 (BT656, BT1120)
    - 4CH Half-D1 (BT656, BT1120)
    - 4CH CIF (BT656)
    - 4CH CIF + 1CH D1 (BT656)
    - 4CH Half-D1 + 1CH D1 (BT656)
    - 8CH CIF (BT656) – cascaded mode
    - 8CH Half-D1 (BT656) – cascaded mode
    - 8CH CIF + 1CH D1 (BT656) – cascaded mode
- Supported host platforms
    - DM365 ARM9 linux based software driver
    - DM6467 ARM9 linux based software driver
- Crop modes for line-multiplexed modes (only in DM6467)
- NTSC/PAL video source
- Integrated, low CPU overhead software demuxing logic
- Multi-channel Video Input Port (MCVIP) interface for low overhead exchange of buffers between drivers and application
- Same MCVIP interface is used for multiple platforms, making applications portable across platforms

- MCVIP interface is similar in behavior to V4L2 interface, making it easy to migrate existing single-CH V4L2 applications to make use of this multi-channel interface.
- Multi-instance driver support for DM6467 dual video capture ports
- User Configurable I2C device address for portability across customer hardware boards
- Audio capture support

1.3 Software Driver : Features Not Supported

- Non multiplexed capture, pixel multiplexed capture support not present in software driver
- V4L2 interface not supported. The driver application interface is a special MCVIP interface optimized for multi-channel operation.

2 Top Level Design

2.1 TVP5158 and Host processor interface

Figure below shows connection between TVP5158 and DMxxx host video port.

- As shown below a single video port is used to send data from multiple video sources.
- This is achieved by TVP5158 multiplexing data on a line-by-line basis and then sending the multiplexed "super"-frame to the DMxxx host.
- In the super-frame for every line the TVP5158 tags each line with CH ID, line number and other information. This meta-data is part of the super-frame data.
- To the DMxxx host this super-frame stream appears as a normal BT656/BT1120 data source. And the video port of DMxxx captures the super-frame along with the per line meta data to its DDR external memory.
- The TVP5158 software driver then interprets this meta-data for every line and then DMAs the actual line active data to channel specific frame buffers.
- Once a channel frame buffer is complete i.e all lines of the frame are received, the driver marks the buffers as "filled".
- When the application issues a API call to get the captured frames, the driver returns the captured frame to the application
- Once the application has processed this captured frame, it returns the buffer back to the driver, so that the driver can reuse this buffer for subsequent frame captures.

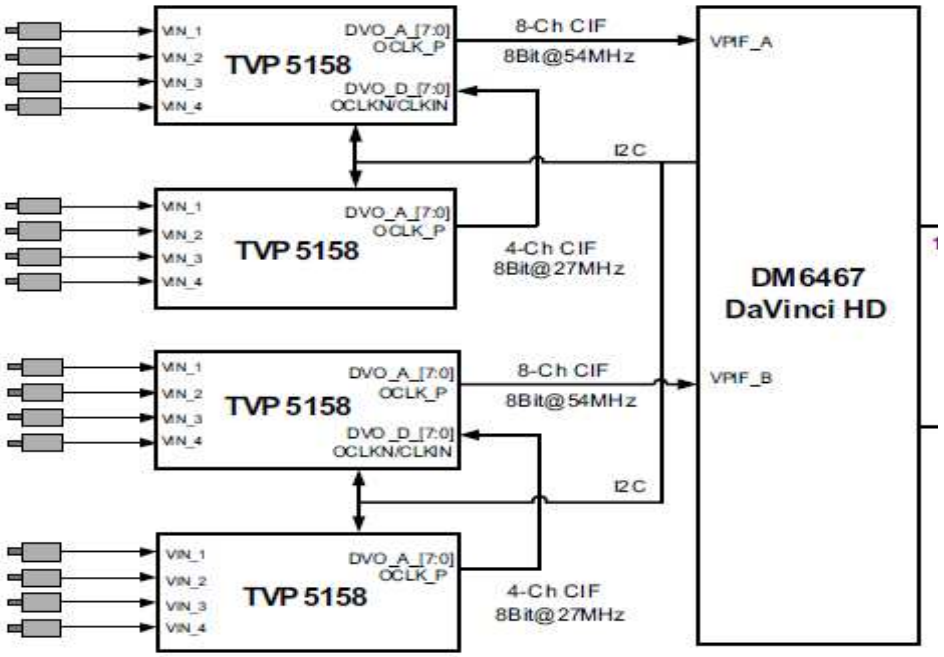

Figure 1. Example, 16CH CIF capture using 4 TVP5158 over two video ports with DM6467 as host

2.2 Software Driver Structure

The software driver structure is shown in the below figure

- As shown below the user application will use the MCVIP interface to configure, control the TVP5158 and capture de-multiplexed frames
- MCVIP User space library implements the super-frame capture, demuxing, I2C programming logic for the TVP5158 and also handles the exchanges of channel frame buffers between the driver and the application
- The DMA driver is used to implement efficient line-by-line DMA which is used by the demuxing logic.
- The I2C driver is used to do send commands over I2C to the TVP5158 device
- The DRV user space library provides the interface to the DMA and I2C driver which are implemented as kernel loadable modules
- The OSA library provides utility functions to create tasks, message queues, frame queues and so on, which help in implementing buffer exchange between application and driver
- The video port capture driver is used to program the platform specific video port in order to capture the super frames from the TVP5158 device.

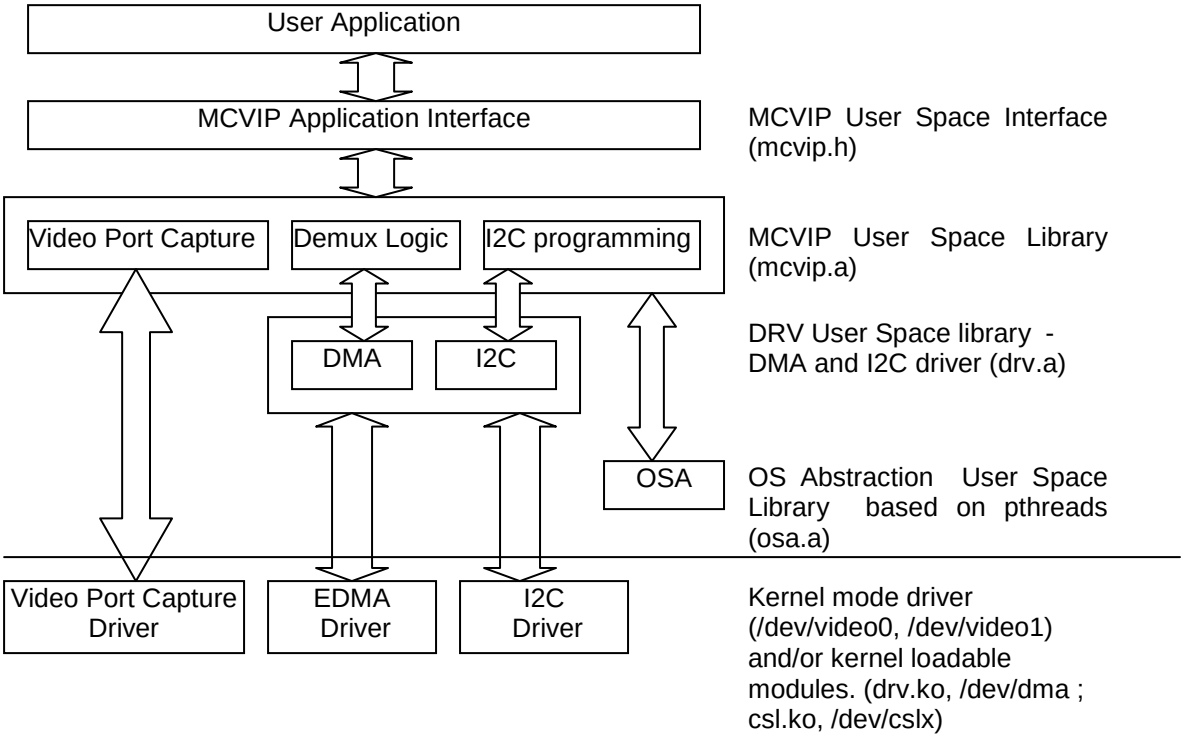

Figure 2. Software Driver Structure

3 Application Interface (API)

3.1 Libraries

User Space Libraries – application must link to these libraries	
Mcvip.a	MCVIP TVP5158 user space driver
Drv.a	DMA and I2C user space driver
Osa.a	OS Abstraction library for buffer and task control
Csl.a (only in DM365)	CSL user space driver for DM365 video port
Kernel Space modules – application must load these before executing the application	
Drv.ko	DMA and I2C kernel side driver implementation
Csl.ko (only in DM365)	Video Port driver for DM365 ISIF (video port). Note, in DM6467 video port driver is part of the base kernel image itself

3.2 Include files

Interface header files – application must include these files in their application to make use of the MCVIP interface	
Mcvip/inc/mcvip.h	MCVIP interface header file.
Include Path – application must include the below paths in their compile include search path	
Mcvip/inc	MCVIP interface header file path.
Osa/inc	OSA header file include path, osa.h file is included by mcvip.h

3.3 Constants

Below constants are defined in mcvip.h

Video Input Port ID	
MCVIP_VIDEO_INPUT_PORT_0	Video Port 0
MCVIP_VIDEO_INPUT_PORT_1	Video Port 1 (only in DM6467)
MCVIP_VIDEO_INPUT_PORT_MAX	Max supported video ports
Video capture and demuxing thread priority	
MCVIP_CAPTURE_THR_PRI_LOW	Lowest priority
MCVIP_CAPTURE_THR_PRI_MED	Medium priority
MCVIP_CAPTURE_THR_PRI_HIGH	Highest priority (recommended)
Video Decoder ID	
MCVIP_VIDEO_DECODER_ID_TVP5158	TVP5158 – as of now only TVP5158 is supported

Video Decoder Mode – uncropped modes	
MCVIP_VIDEO_DECODER_MODE_2CH_D1_PORT_A	2CH D1 via TVP5158 Port A
MCVIP_VIDEO_DECODER_MODE_2CH_D1_PORT_B	2CH D1 via TVP5158 Port B
MCVIP_VIDEO_DECODER_MODE_4CH_D1	4CH D1
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1	4CH Half-D1
MCVIP_VIDEO_DECODER_MODE_4CH_CIF	4CH CIF
MCVIP_VIDEO_DECODER_MODE_4CH_D1_16	4CH D1 16-bit BT1120 mode (only in DM6467)
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1_16	4CH Half-D1 16-bit BT1120 mode (only in DM6467)
MCVIP_VIDEO_DECODER_MODE_8CH_HALF_D1	8CH Half-D1 cascaded mode
MCVIP_VIDEO_DECODER_MODE_8CH_CIF	8CH CIF cascaded mode
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1_PLUS_D1	4CH Half-D1 + 1CH D1
MCVIP_VIDEO_DECODER_MODE_4CH_CIF_PLUS_D1	4CH CIF + 1CH D1
MCVIP_VIDEO_DECODER_MODE_8CH_CIF_PLUS_D1	8CH CIF + 1 CH D1
Video Decoder Modes – cropped (only in DM6467)	
MCVIP_VIDEO_DECODER_MODE_2CH_D1_PORT_A_CROP	2CH D1 via TVP5158 Port A
MCVIP_VIDEO_DECODER_MODE_2CH_D1_PORT_B_CROP	2CH D1 via TVP5158 Port B
MCVIP_VIDEO_DECODER_MODE_4CH_D1_CROP	4CH D1
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1_CROP	4CH Half-D1
MCVIP_VIDEO_DECODER_MODE_4CH_CIF_CROP	4CH CIF
MCVIP_VIDEO_DECODER_MODE_4CH_D1_16_CROP	4CH D1 16-bit BT1120 mode
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1_16_CROP	4CH Half-D1 16-bit BT1120 mode
MCVIP_VIDEO_DECODER_MODE_8CH_HALF_D1_CROP	8CH Half-D1 cascaded mode
MCVIP_VIDEO_DECODER_MODE_8CH_CIF_CROP	8CH CIF cascaded mode
MCVIP_VIDEO_DECODER_MODE_4CH_HALF_D1_PLUS_D1_CROP	4CH Half-D1 + 1CH D1
MCVIP_VIDEO_DECODER_MODE_4CH_CIF_PLUS_D1_CROP	4CH CIF + 1CH D1
MCVIP_VIDEO_DECODER_MODE_8CH_CIF_PLUS_D1_CROP	8CH CIF + 1 CH D1
Video Interface	
MCVIP_VIDEO_IF_MODE_BT656	8-bit BT656 interface
MCVIP_VIDEO_IF_MODE_BT1120	16-bit BT1120 interface
Video System	
MCVIP_VIDEO_SYSTEM_NTSC	NTSC video system
MCVIP_VIDEO_SYSTEM_PAL	PAL video system
Buffer flags	

MCVIP_FLAG_ENCODER_DONE	Mark buffer as processed by encoder
MCVIP_FLAG_DISPLAY_DONE	Mark buffer as processed for display
MCVIP_FLAG_ALL_DONE	Mark buffer as processed by all including encoder and display
Other constants	
MCVIP_CHANNELS_MAX	Maximum channels per MCVIP handle
MCVIP_TVP5158_MAX_CASCADE	Max TVP5158 that can be connected in cascade
MCVIP_BUF_PER_CH_MIN	Minimum buffers that are needed to be provided by application per CH to the driver
MCVIP_BUF_MAX	Max number of buffer's for all CH's that can be provided by application to the driver

3.4 Data structures

Below data structures are defined in mcvip.h

MCVIP_CreatePrm – parameter structure that is passed as input during MCVIP_create	
int videoInputPort	video input port ID, MCVIP_VIDEO_INPUT_PORT_0 or MCVIP_VIDEO_INPUT_PORT_1
int captureThrPri	capture thread priority, 1 to 100. User can also use MCVIP_CAPTURE_THR_PRI_xxx
int videoDecoderId	Video decoder ID, MCVIP_VIDEO_DECODER_ID_xxx
int videoDecoderMode	Video decoder mode, MCVIP_VIDEO_DECODER_MODE_xxx
int videoIfMode	Video interface mode, MCVIP_VIDEO_IF_MODE_BTxxx
int videoSystem	Video system, MCVIP_VIDEO_SYSTEM_xxx
int numBuf	Number of buffer's to use for capture, must be >= (number of channels * MCVIP_BUF_PER_CH_MIN)
int bufSize	Size of each buffer, must >= ROUND(frame width, 32)* frame height*2
Uint8 *bufPhysAddr[MCVIP_BUF_MAX]	Physical buffer address of each buffer, must be 32byte aligned – must be contiguous in memory, use a memory allocator like CMEM to allocate contiguous buffers
Uint8 *bufVirtAddr[MCVIP_BUF_MAX]	Virtual buffer address of each buffer, must be 32byte aligned – must be contiguous in memory, use a memory allocator like CMEM to allocate contiguous buffers
MCVIP_BufInfo – buffer information. User can use this when extracting information about a captured buffer. User should treat this information as READ-ONLY	
int chId	Channel ID (0..MCVIP_CHANNELS_MAX-1)

int flags	Used internally, SHOULD NOT BE MODIFIED BY USER
unsigned char *physAddr	Buffer physical address
unsigned char *virtAddr	Buffer virtual address
int timestamp	Buffer timestamp in msecs
MCVIP_ChInfo – channel specific information.	
int width	data width, in pixels
int height	data height, in lines
int offsetH	horizontal line offset, in pixels
int offsetV	vertical line offset, in lines, (In DM6467, for YUV422SP data format, chroma data offset = lineOffsetH*lineOffsetV bytes)
MCVIP_ChList – information of all channels	
int numCh	Number of channels (0..MCVIP_CHANNELS_MAX-1)
MCVIP_ChInfo info[MCVIP_CHANNELS_MAX]	Channel specific information

3.5 Functions

Below functions are defined in mcvip.h

int MCVIP_init(int devAddr[], int max_tvp5158, int audio_flag);	
Description	This function should be called by application during system init. This sets up the I2C addresses for the TVP5158 devices connected to various video ports of the host.
Parameters	
int devAddr[]	I2C device addresses for each TVP5158 cascade stage and each video port
int max_tvp5158	Max TVP5158's connected in cascade mode 1: One TVP5158 connected to one video port 2: Two TVP5158 connected in cascade to one video port
int audio_flag	Flag to control audio initialization during TVP5158 init 0: Do not init audio 1: Init audio
Return Value	OSA_SOK on success, else failure

int MCVIP_exit();	
Description	This function should be called by application during system exit. This free's up system resources allocated during MCVIP_init()
Parameters	
NONE	

Return Value	OSA_SOK on success, else failure

void *MCVIP_create(MCVIP_CreatePrm *prm);	
Description	Creates a thread for capturing data and allocates required resources specific to a video input port Should be called for each HW Video Input port, with appropriate input port ID On success it returns a handle to the MCVIP driver instance. This handle should be used in subsequent MCVIP API calls
Parameters	
MCVIP_CreatePrm *prm	Create time parameters
Return Value	MCVIP driver instance handle pointer on success, else NULL on failure

int MCVIP_delete(void *hndl);	
Description	Delete's the thread and other related resources allocated during MCVIP_create for that driver instance
Parameters	
void *hndl	Driver instance handle
Return Value	OSA_SOK on success, else failure

int MCVIP_start(void *hndl);	
Description	Start capture of video frames from TVP5158 This programs the I2C register in the TVP5158, including TVP5158 patch download, if required, and then starts the video port to begin capture of super-frame data
Parameters	
void *hndl	Driver instance handle
Return Value	OSA_SOK on success, else failure

int MCVIP_stop(void *hndl);	
Description	Stop capture of video frames from TVP5158
Parameters	
void *hndl	Driver instance handle
Return Value	OSA_SOK on success, else failure

int MCVIP_getChList(void *hndl, MCVIP_ChList *chInfo);	
Description	Get channel specific info
Parameters	

void *hndl	Driver instance handle
MCVIP_ChList *chInfo	[OUT] Channel info
Return Value	OSA_SOK on success, else failure

int MCVIP_getBuf(void *hndl, int *id, unsigned int timeout);	
Description	Get a demuxed frame buffer. This function returns a buffer ID, to get buffer info use the API MCVIP_getBufInfo() with this ID as parameter
Parameters	
void *hndl	Driver instance handle
int *id	[OUT] Buffer ID
unsigned int timeout	OSA_TIMEOUT_NONE – no timeout, non-blocking API. If no buffer is available -1 ID is returned and return status is -1 OSA_TIMEOUT_FOREVER – block until at least one buffer is available, i.e blocking API
Return Value	OSA_SOK on success, else failure

int MCVIP_putBuf(void *hndl, int id, int flags);	
Description	Release a demuxed frame buffer back to driver Typically, there are two tasks that could use the captured buffer 1. For encoding 2. For display Since these tasks can happen asynchronously with each other, the capture buffer should be released to driver only when both encoding and display of the buffer is done. To achieve this, user should set 'flags' parameter appropriately when calling this API. Set flags according to the task that has completed using this buffer i.e if display task is done using the buffer set flag to MCVIP_FLAG_DISPLAY_DONE if encoder task is done using the buffer set flag to MCVIP_FLAG_ENCODER_DONE Only when both MCVIP_FLAG_DISPLAY_DONE, MCVIP_FLAG_ENCODER_DONE are done is the buffer released In case both encode processing and display happen in the same task and user wishes to release the buffer in one go then they should use the flag MCVIP_FLAG_ALL_DONE
Parameters	

void *hndl	Driver instance handle
int id	buffer ID – same as the one got during MCVIP_getBuf
int flags	MCVIP_FLAG_xxx_DONE
Return Value	OSA_SOK on success, else failure

MCVIP_BufInfo *MCVIP_getBufInfo(void *hndl, int id);	
Description	Get buffer info for a particular buffer ID User gets a buffer ID when MCVIP_getBuf() API is called. Use this API to get the buffer information for a buffer ID. Note, this function returns a pointer to driver internal buffer information structure. Hence user should treat the returned information as READ ONLY. No fields of the return pointer structure should be modified by the user.
Parameters	
void *hndl	Driver instance handle
int id	Buffer ID
Return Value	Returns pointer to buffer information structure (MCVIP_BufInfo *) If buffer ID is invalid or incase of error, it returns NULL

.

int MCVIP_saveFrame(void *hndl);	
Description	This is a utility API to save one frame of super frame data. This is meant for debugging and would typically not be used by the application
Parameters	
void *hndl	Driver instance handle
Return Value	OSA_SOK on success, else failure

.

int MCVIP_getNumCh(int videoDecoderMode);	
Description	Utility API which returns number of video channels that are associated with a given video decoder mode. This API is useful when setting values for MCVIP_CreatePrm
Parameters	
int videoDecoderMode	Video decoder mode, MCVIP_VIDEO_DECODER_MODE_xxx
Return Value	number of video channels

.

int MCVIP_getBufSize(int videoDecoderMode, int videoSystem);	
Description	Utility API which returns size of a video frame buffer for a given mode, video system. This API is useful when setting values for MCVIP_CreatePrm

	This API is also useful for calculating size of buffer to be allocated by the user.
Parameters	
int videoDecoderMode	Video decoder mode, MCVIP_VIDEO_DECODER_MODE_xxx
int videoSystem	Video system, MCVIP_VIDEO_SYSTEM_xxx
Return Value	size of a video frame buffer, in bytes

4 Sample API Usage Sequence

The typically sequence of API usage is given below with examples

1. System init - MCVIP_init()
2. Create driver instance – MCVIP_create()
   - TVP5158 mode init and video port init for a specific mode
3. Get channel info – MCVIP_getChList()
4. Capture start – MCVIP_start()
5. Get captured frame – MCVIP_getBuf()
6. Get captured frame info – MCVIP_getBufInfo()
7. Process this captured frame, i.e resize, encode, display etc
8. Release capture frame buffer - MCVIP_putBuf()
9. Repeat steps 5-8 continuously until user is done
10. Capture stop – MCVIP_stop()
11. Delete driver instance – MCVIP_delete()
12. System exit – MCVIP_exit()

An example code snippet showing above sequence is given below. Note, this code has been simplified and limited error checking is done for the sake of clarity.

```
#include <osa_cmem.h>
#include <drv.h> // only in DM365
#include <mcvip.h>


void *gMcvipHndl;
MCVIP_ChList gChInfo;
MCVIP_CreatePrm gCreatePrm;


int SYS_run()
{
  int status;

  status = SYS_init();
  if(status==OSA_SOK) {
    status = CAPTURE_start();
    if(status==OSA_SOK) {
      status = CAPTURE_loop();
      CAPTURE_stop();
    }
    SYS_exit();
  }
```

```
  return status;
}

int SYS_init()
{
  int devAddr[2], status;

  // memory allocator init, needed by MCVIP
  CMEM_init();
  // EDMA, I2C driver init
  DRV_init(); // only in DM365
  // TVP5158 I2C device address
  devAddr[0] = 0xB0;
  devAddr[1] = 0xB2; // cascaded TVP5158, if present, else set to 0xFF
  status = MCVIP_init(devAddr, 2, 1);
  return status;
}

int CAPTURE_start()
{
  Uint8 *virtAddr;
  int bufId;

  gCreatePrm.videoInputPort = MCVIP_VIDEO_INPUT_PORT_0;
  gCreatePrm.captureThrPri = MCVIP_CAPTURE_THR_PRI_HIGH;
  gCreatePrm.videoDecoderId = MCVIP_VIDEO_DECODER_ID_TVP5158;
  gCreatePrm.videoDecoderMode = MCVIP_VIDEO_DECODER_MODE_8CH_CIF;
  gCreatePrm.videoIfMode = MCVIP_VIDEO_IF_MODE_BT656;
  gCreatePrm.videoSystem = MCVIP_VIDEO_SYSTEM_NTSC;
  gCreatePrm.numBuf = MCVIP_getNumCh(gCreatePrm.videoDecoderMode) * MCVIP_BUF_PER_CH_MIN;
  gCreatePrm.bufSize = MCVIP_getBufSize(gCreatePrm.videoDecoderMode, gCreatePrm.videoSystem);
  for(bufId=0; bufId<gCreatePrm.numBuf; bufId++)
  {
    virtAddr = OSA_cmemAlloc(gCreatePrm.bufSize, 32);
    gCreatePrm.bufVirtAddr[bufId] = virtAddr;
    gCreatePrm.bufPhysAddr[bufId] = OSA_cmemGetPhysAddr(virtAddr);
  }

  gMcvipHndl = MCVIP_create(&gCreatePrm);
  if(gMcvipHndl==NULL)
    return -1;
```

```
  MCVIP_getChList(gMcvipHndl, &gChInfo);
  MCVIP_start(gMcvipHndl);
  return 0;
}

int CAPTURE_loop()
{
  int bufId, status;
  MCVIP_BufInfo *pBuf;

  while(1) {
    status = MCVIP_getBuf(gMcvipHndl, &bufId, OSA_TIMEOUT_FOREVER);
    if(status!=OSA_SOK || bufId < 0)
      return -1;

    pBuf = MCVIP_getBufInfo(gMcvipHndl, bufId);

    /* process buffer pBuf
      pBuf->chId is the channel to which this buffer belongs
      pBuf->timestamp is buffer time stamp
      pBuf->physAddr is the buffer physical address
      pBuf->virtAddr is the buffer virtual address
    */
    /* channel info for this buffer is
    gChInfo.chList[pBuf->chId].width
    gChInfo.chList[pBuf->chId].height
    gChInfo.chList[pBuf->chId].offsetH
    gChInfo.chList[pBuf->chId].offsetV
    */

    MCVIP_putBuf(gMcvipHndl, bufId, MCVIP_FLAG_ALL_DONE);

    // check stop condition and break loop when done
  }
  return 0;
}

void CAPTURE_stop()
{
  int bufId;

  MCVIP_stop(gMcvipHndl);
```

```
  MCVIP_delete(gMcvipHndl);
  for(bufId=0; bufId<gCreatePrm.numBuf; bufId++)
    OSA_cmemFree(gCreatePrm.bufVirtAddr[bufId]);
}

void SYS_exit()
{
  MCVIP_exit();
  DRV_exit(); // only in DM365
  CMEM_exit();
}
```

5 Comparison with V4L2 interface

The MCVIP interface used for the TVP5158 driver is different from the V4L2 interface typically used by linux application for video data capture.

The reasons for not using the V4L2 interface for the TVP5158 driver are given below,

1. V4L2 interface is designed with single channel per video port capture kind of system. So multi-channel V4L2 capture can be supported in a system if there are as many video ports in the system. However in a TVP5158 based system, we need to receive multiple channels over the same video port. Such a system scenario is difficult to fit and implement with a V4L2 interface
2. For implementing the TVP5158 driver there needs to be a separate thread inside the driver for doing the demultiplexing operation. This thread is required so that the demux operation does not block the main application thread. In a V4L2 kind of interface, typically, the complete driver is implemented in the kernel space. However for TVP5158 driver demuxing thread, it is difficult to create an independent thread in the kernel space, due to which significant part of the driver resides in the user space. It would have been difficult to implement the complete TVP5158 driver only in kernel space.

Even though the TVP5158 driver is not based on V4L2 interface, the MCVIP interface is designed to be similar to V4L2 interface so that it is relatively straight forward to migrate single channel V4L2 based applications to make use of the multi-CH MCVIP interface.

The correspondence between MCVIP and V4L2 APIs is shown in the below table.

This will help users to migrate their V4L2 applications.

MCVIP Interface	**V4L2 Interface**
MCVIP_init()	NONE – V4L2 init happens during kernel init
Sequence of below function calls, MCVIP_create() MCVIP_getChInfo()	Sequence of below function calls, Open(/dev/videoX) Ioctl VIDIOC_QUERYCAP Ioctl VIDIOC_ENUMINPUT Ioctl VIDIOC_S_INPUT Ioctl VIDIOC_S_STD Ioctl VIDIOC_S_FMT Ioctl VIDIOC_REQBUFS Ioctl VIDIOC_QBUF – initial buffer queueing or priming
MCVIP_start()	Ioctl VIDIOC_STREAMON
Sequence of below function calls, MCVIP_getBuf() MCVIP_getBufInfo()	Ioctl VIDIOC_DQBUF

MCVIP_putBuf()	Ioctl VIDIOC_QBUF
MCVIP_stop()	Ioctl VIDIOC_STREAMOFF
MCVIP_delete()	Close()
MCVIP_exit()	NONE – V4L2 system shutdown happens during kernel shutdown

6 Output Data format and system data flow

The data format, that is output from or input to different modules on host system, is different in different platforms as shown below,

	Video port output	MCVIP Output	Display Input	H264 Encoder Input	Data format Conversion HW
DM365	YUV422 ILE	YUV422 ILE	YUV420 SP or YUV422 ILE	YUV420 SP	HW: Resizer Input: YUV422 ILE Output: YUV420 SP
DM6467	YUV422 SP	YUV422 SP	YUV422 SP	YUV420 SP	HW: VDCE Input: YUV422 SP Output: YUV420 SP

Note,

- YUV422 ILE means YUV422 InterLEaved. Here data is arranged as UYVY … in memory, i.e luminance and chrominance data are interleaved in memory
- YUV422 SP means YUV422 Semi-Planer. Here data is divided into two planes, first plane has luminance data arranged as YYYY…. Second plane has chrominance data arrange as UVUV …, luminance data is separated from chrominance and chrominance data is interleaved
- YUV420 SP means YUV420 Semi-Planer. This is similar to YUV422 SP except that chrominance plan is further sub-sampled vertically.

Note,

- MCVIP Output format will always be same Video port output format, i.e MCVIP driver does NOT do any data format conversion.

In order to build a video capture+display+encode system, application needs to make sure that it passes data to different modules in compatible data formats and does data format conversion, using appropriate HW, when required.

Example, data flow for H264 encode system for DM365 is shown below,

MCVIP → DDR (YUV422 ILE) → HW RESZ (out A) → DDR (YUV420SP) → H264 Encode

L (out B)→ DDR (YUV420SP) → EDMA → DDR (YUV420SP) → Display

Example, data flow for H264 encode system for DM6467 is shown below,

MCVIP → DDR (YUV422 SP) → HW VDCE → DDR (YUV420SP) → H264 Encode

|→ EDMA → DDR (YUV422SP) → Display

7 Resource and Performance Benchmarks

7.1 Resource requirements

EDMA Channels	1
EDMA PaRAM entries	24

7.1.1 Note about EDMA PaRAM Entries

The EDMA PaRAM entries used by the driver can be modified by changing the below constant and then recompiling the user as well as kernel side driver and reloading the kernel modules.

FILE: drv/inc/drv_dma.h

// must be multiple of 2, must be >=2

#define DRV_DMA_MAX_DEMUX_PARAM (24)

Users can reduce this to reduce the number of EDMA PaRAM entries, if they need to use the PaRAM entries for some other driver or algorithm/codec

However, the value of this #define affects system CPU loading. If the value is too low the CPU loading increases.

In general a value of 24 has been found to be high enough to reduce CPU loading and low enough so as keep sufficient PaRAM entries free for other drivers and algorithms/codecs

7.1.2 Note about EDMA channels

One EDMA channel is needed for demuxing the received super-frame into individual channel frames. In case user falls short of EDMA channels on the linux system then they can increase the number of EDMA channels available for linux side system by modifying the kernel EDMA configuration header file (ti-davinci/include/asm-arm/arch-davinci/edma.h, EDMA_DMxxxx_CHANNEL_TO_EVENT_MAPPING_x). Refer to LSP documentation for details.

7.2 CPU Loading

Due to the demuxing logic and interrupt overhead due to line-by-line DMA, the TVP5158 driver adds some CPU loading as shown below. This assumes I-cache and D-cache is ON in the ARM processor.

Video Decoder Mode	**ARM Total Mhz (ISR + Thread)**	**ISR Mhz**	**Thread Mhz**
2CH D1 4CH CIF	17 Mhz	8 Mhz	9 Mhz
4CH D1 4CH Half-D1 8CH CIF	34 Mhz	16 Mhz	18 Mhz
4CH Half-D1 + 1CH D1	51 Mhz	24 Mhz	27 Mhz

8 DVSDK / LSP dependencies

8.1 Supported versions

Given below are the LSP and DVSDK versions against which the TVP5158 is implemented,

Platform	LSP version	DVSDK version
DM365	2.10.xx.xx	2.10.xx.xx
DM6467	1.30.xx.xx	1.40.xx.xx

8.2 Migration to other versions

Usually there are significant LSP and DVSDK changes across different versions and applications and drivers need to be specifically ported when moving across different versions of LSP and DVSDK.

This section does not explain how to migrate to different versions but gives a feel of what all needs to be ported when moving to a new LSP/DVSDK version.

The following parts of the driver are dependant on the LSP

- Kernel Video port driver (in DM6467)
- CSL kernel module portions (in DM365)
- DMA and I2C kernel module portions

The following parts of the driver are dependant on the DVSDK

- CMEM – continuous memory allocator

In general if the application is written to use only the MCVIP interface then once the driver is migrated, application migration will be straightforward.

9 Source code information

Refer to platform specific README.txt or relevant platform installation guide for detailed platform specific directory structure, build and install instructions.

This section provides some information which is common to all platforms implementing the TVP5158 driver.

Source code file summary is given below,

MCVIP	
Mcvip/inc/mcvip.h	Top level MCVIP interface include file
Mcvip/priv/mcvip_priv.h	Internal Implementation header file
Mcvip/src/mcvip_api.c	MCVIP interface function implementation
Mcvip/src/mcvip_demux.c	Demux operation logic code
Mcvip/src/mcvip_tsk.c	Super-frame capture and demux thread
Mcvip/src/mcvip_tvp5158.c	TVP5158 top level control functions
Mcvip/src/mcvip_tvp5158_i2c.c	TVP5158 low level I2C programming sequence, including patch download sequence
Mcvip/src/mcvip_v4l2.c	Video port super-frame capture driver wrapper, uses V4L2 in DM6467, uses CSL interface in DM365
DRV	
Drv/inc/drv_dma.h	DMA APIs used during demuxing
Drv/inc/drv_i2c.h	I2C read, write APIs
Drv/kermod/*	Kernel modules to do EDMA and I2C programming
Drv/usermod/*	User space API implementation of EDMA and I2C
OSA	
Osa/inc/osa.h	Data types
Osa/inc/osa_cmem.h	DVSDK CMEM wrapper APIs
Osa/src/*	OSA implementation
TEST CODE	
Test/i2crw	Command line I2C read, write utility
Test/dma	EDMA API unit test
Test/mcvip	MCVIP API sample application

10 Other Useful Information

10.1 TVP5158 patch download

TVP5158 has an internally ROM code, which executes when TVP5158 is powered on. However, based on issues found during production, this ROM is "patched" to fix the known issues. This patch code needs to be downloaded to TVP5158 at least once, after power-on reset.

The patch code is downloaded using a specific sequence of I2C commands that are sent to TVP5158. This patch code download is taken care by the MCVIP driver when MCVIP_start() is called.

Below section provides additional information on how to identify the patch that is being downloaded and steps to change patch that gets downloaded.

10.1.1 Identifying the patch that is being used with the driver

"mcvip\priv\tvp5158_patch_v02_xx_xx.h" are the different patch files.

This .h file contains the patch code which is basically an array of byte data.
static const Uint8 gTVP5158_patch[] = { … };

The patch that finally gets downloaded is specified below,

FILE: "mcvip\src\mcvip_tvp5158_i2c.c"
#include <tvp5158_patch_v02_xx_xx.h>

*int TVP5158_patchDownload(DRV_I2cHndl *i2cHndl)* function downloads the patch to the TVP5158

10.1.2 Changing the patch that gets downloaded

To change the patch file simply copy the patch file to "mcvip\priv" and modify the #include in "mcvip\src\mcvip_tvp5158_i2c.c" to include this new patch file.