DVR RDK

Document Revision 1.05

Multi Channel FrameWork Software 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	14 July 2011	[KC] First Draft
1.01	27 Aug 2011	[KC] More Updates
1.02	29 Sep 2011	Updates for different dataflows (like interlaced, dsp inclusive)
1.03	30 Dec 2011	Further updates
1.04	27 Feb 2012	Upgraded for GA
1.05	21 May 2012	Features for 2.80.xx.xx release

TABLE OF CONTENTS

1 Introduction

1.1 Overview

The DVR RDK is a multi-processor software development framework for TI81xx platform and is optimized for multi-channel applications like Surveillance DVR, NVR, Hybrid-DVR, HD-DVR.

The software framework in DVR RDK allows a user to create different multi-channel data flows involving video capture, video processing (DEI, Noise Filter, Encode, Decode, SW Mosaic) and video display.

1.2 Key Acronyms and Vocabulary

Term	Description
DVR	Digital Video Recorder – Analog CCTV inputs, converted to digital encoded bitstream and stored on media. Supports playback of the stored media
NVR	Network Video Recorder – Instead of analog CCTV inputs, accepts encoded bitstream over IP. Decoders and display along with storage
HD-DVR/NVR	High Definition Digital/Network Video Recorder – Provides support for analog 720P,1080i and 1080P inputs
Hybrid DVR	Combination of NVR and DVR
DVR RDK	DVR Reference Design Kit – Includes Multi Channel software framework and hardware platform
HDVPSS	High Definition Video Processing Subsystem – Referred in the document for both hardware block as well as software driver package
HDVICP	High Definition Video and Image CoProcessor – Referred in the document mainly for the software codecs package and for hardware IP Block
Ducati	Dual core M3 processors controlling HDVPSS and HDVICP hardware engines
Video M3	ARM Cortex M3 core (inside Ducati subsystem) controlling HDVICP codecs
VPSS M3 / DSS M3	ARM Cortex M3 core (inside Ducati subsystem) controlling HDVPSS drivers
DEI	Deinterlacer – Referred in document for hardware deinterlacer block as well as the

	software components supporting it
NF	Noise Filter - Referred in document for hardware noise filter block as well as the software components supporting it
SC	Scalar - Referred in document for hardware scalar block as well as the software components supporting it
McFW	Multi Channel Framework – Software framework developed for multi-channel DVR applications.
Links	Smallest software component controlling the functionality (like capture, DEI,display) – has input queue and output queue
IPC	Inter Processor Communication
ListMP	List Multi Processor – Syslink component for sharing a list of buffer pointers across processors
SR	Shared Region – Syslink component for having a shared memory across processors

2 Installation and build

2.1 Dependant packages

The DVR RDK is dependant on the following additional packages.

Please refer to the Release Notes for exact package version required for the current release of DVR RDK

Package Name	**Package Version**
A8 Linux - Code Sorcery Code Generation tools	Refer to Release Notes for package version
ARM M3 Code Generation tools	
DSP c6x Code Generation tools	
Linux PSP	
XDC	
BIOS	
Syslink	
IPC	
XDIAS	
Framework components	
IVAHD HDVICP2 API	

H264 decoder	
H264 encoder	
HDVPSS drivers	

2.2 Installing DVR RDK

Refer to Install Guide provided with the released package for the instructions to install DVR RDK.

2.3 Building the DVR RDK

Refer to Install Guide provided with the released package for the instructions to build DVR RDK.

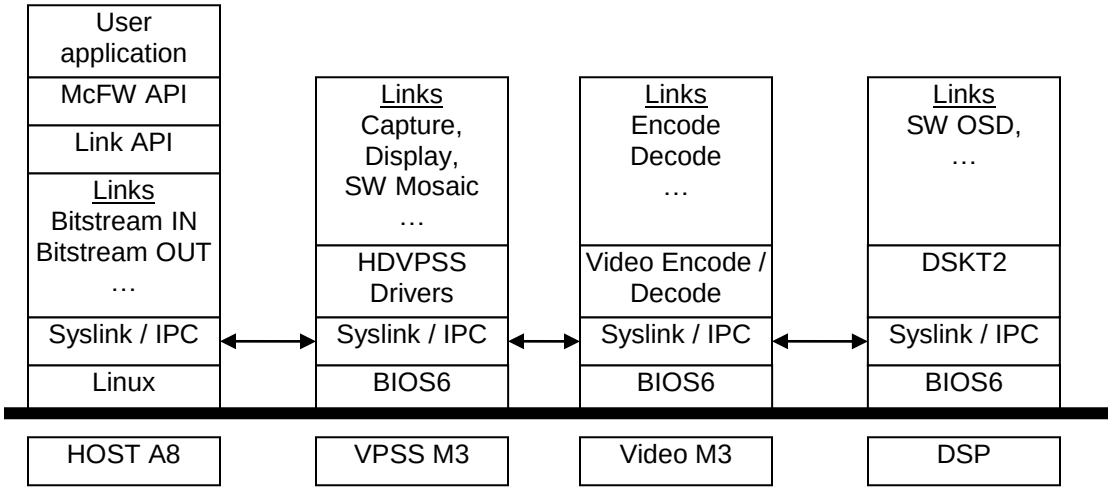
2.4 Running the DVR RDK

Refer to Install Guide provided with the released package for the instructions to execute the default application provided with DVR RDK.

3 Top Level Design

3.1 Software Layers

The Software is implemented using different layers as shown below. The amount of layering is done based on a balance between modularity (more abstraction/layers) and ease of use (less number of layers).



The different layers of the software are described below,

Layer	Processor Applicable	Description	TI SW Package
Linux	HOST A8	Linux OS, includes, filesystems, SATA, Ethernet, USB and other IO drivers	Linux PSP
BIOS6	VPSS M3 Video M3 DSP	BIOS RTOS used as OS on Video-M3, VPSS-M3, DSP. Provides features like threads, semaphores, interrupts. Queues and message passing between links is implemented using BIOS semaphores.	BIOS XDC (used for BIOS and other configuration)
Syslink / IPC	HOST A8 VPSS M3 Video M3 DSP	Software APIs used for communicating between processors. Provides features like processor loading and booting, multiprocessor heaps, multiprocessor linked list (ListMP), message queues, notify etc	Syslink IPC
HDVPSS Drivers	VPSS M3	HDVPSS drivers like capture, display, deinterlacer, scaling based on FVID2 interface to control and configure the HDVPSS HW	HDVPSS
Video Encode/Decode	Video M3	Video encode / decode APIs based on XDM / XDIAS interface. Uses framework components for resource allocation	XDIAS Framework components IVAHD HDVICP2 API H264 decoder H264 encoder
Links	HOST A8 VPSS M3 Video M3 DSP	Implementation of individual links. Some links are specific to a processor while some links are common across processors	DVR RDK
Link API	HOST A8	The link API allows users to create, connect, and control links on HOSTA8, VPSS M3, Video M3 and DSP. Link API is used to create a chain of links which forms a user	McFW

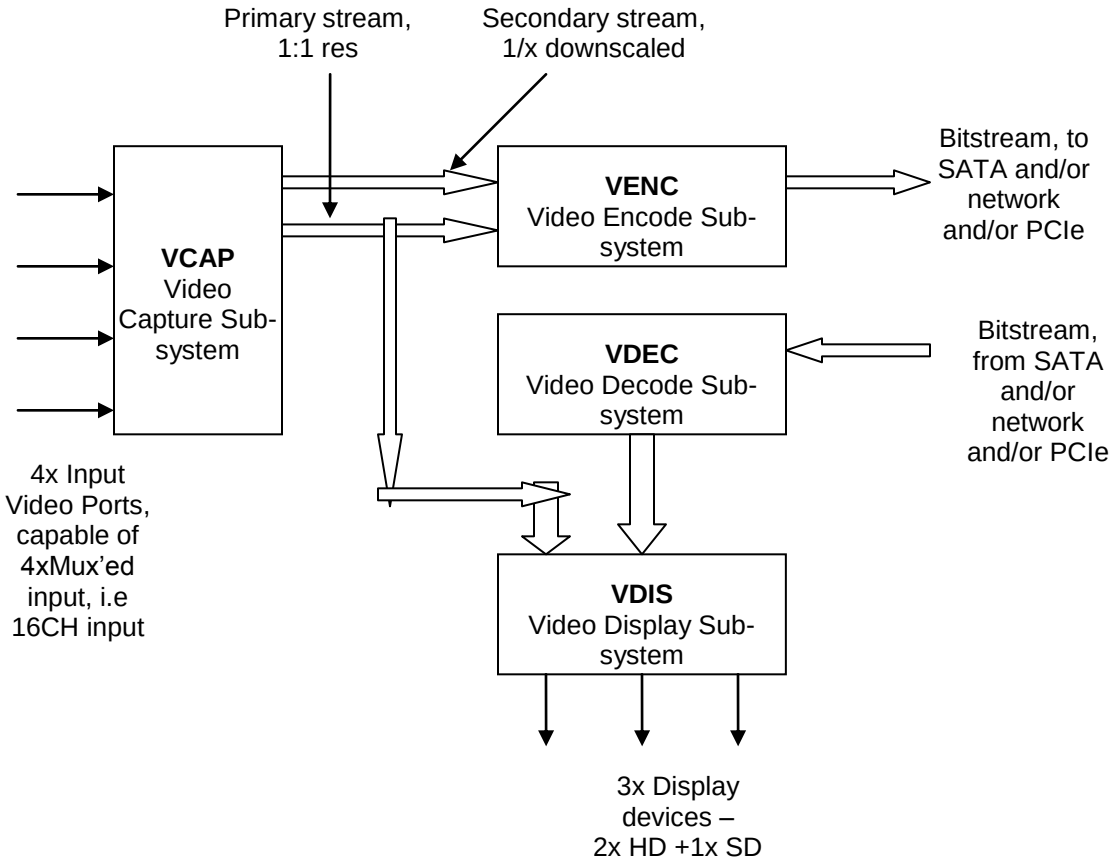
Layer	Processor Applicable	Description	TI SW Package
		defined use-case. The connection of links to each other is platform dependant.	
McFW API	HOST A8	**M**ulti-**C**hannel **F**rame**W**ork API. Multi-Channel Application specific API which allows user to setup and control pre-defined application specific chains for DVR, NVR, using a single simplified API interface. This allows users to directly use the links without having to understand the detailed link API. The McFW API is platform independent and same API will work on DM816x, DM814x, DM810x	McFW
User Application	HOST A8	Typically GUI and other application specific components like file read/write, network streaming. User application can use the McFW API for pre-defined use-cases OR User application can use the link API and create its own custom chains. NOTE: User application NEED NOT create "links" of their own for say file write. Users can write their own custom implementation of processing steps outside of the link API. DVR RDK provides tow kinds of demos . "chains" demo, this uses the link API . "McFW" demo, this uses the McFW API -	Customer specific

3.2 McFW API

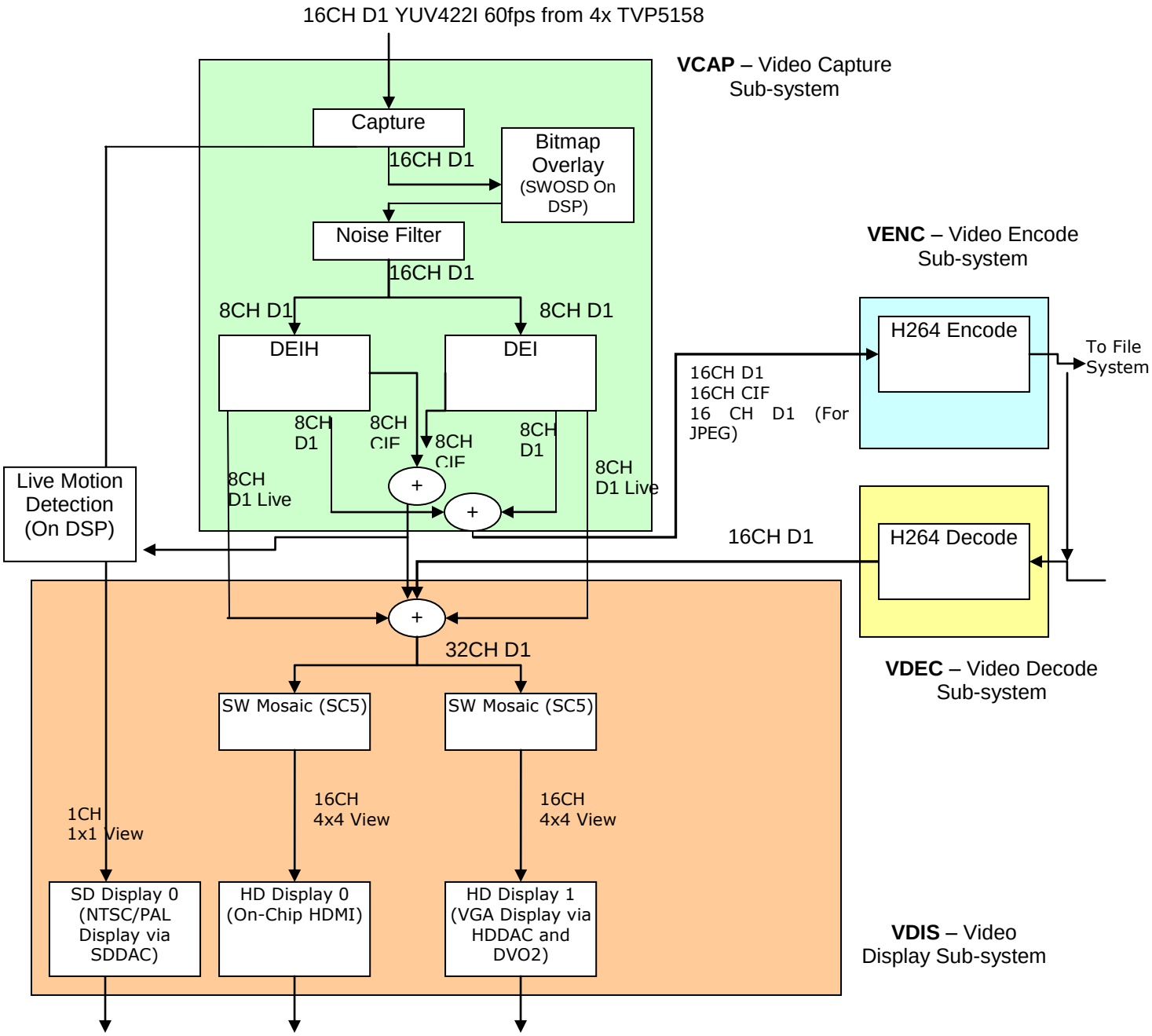
The McFW API is based on the following principles

- A Multi-Channel video system is considered to consist of four sub-systems as listed below
    - Capture – This will capture multi-channels from input video ports, optionally noise filter and/or deinterlace and/or chroma downsample based on the input source
    - Display – This will take input from capture and decode sub-system, and show them multiple channels in different user-defined mosaic combinations on multiple display devices.
    - Encode – This will take input from capture and encode the video including "sub-stream" encode and give the encode bitstream to the user
    - Decode – This will take input bitstreams for multi-channels from user and provide as input to the display subsystem
- The API provide a means of selecting how the channels within a sub-system are connected to other sub-systems
- The API hides platform level details like YUV format conversion, scalars to be used, deinterlacers to be used and allows user to focus on broad level sub-systems rather than low level hardware resources and constraints.
- The hardware blocks (noise filter, scaler, deinterlacer) used inside a sub-system depend on the top level system configuration done by the user depending on their use-case.
- User will see the same block diagram for all use-cases on all platforms. The detailed blocks inside the sub-system will depend on the system level configuration selected by the user. The blocks inside the subsystem will also depend on the platform like DM816x or DM814x.
- **Thus McFW API allows user to use the same API for different products across different platforms, thus allowing user to keep their GUI and other applications portable to different product lines and platforms.**

3.2.1 Block Diagram of System as viewed by McFW API



Example, of detailed internal block diagram for 16CH D1 encode + decode DVR Use-case for DM816x platform is shown below.

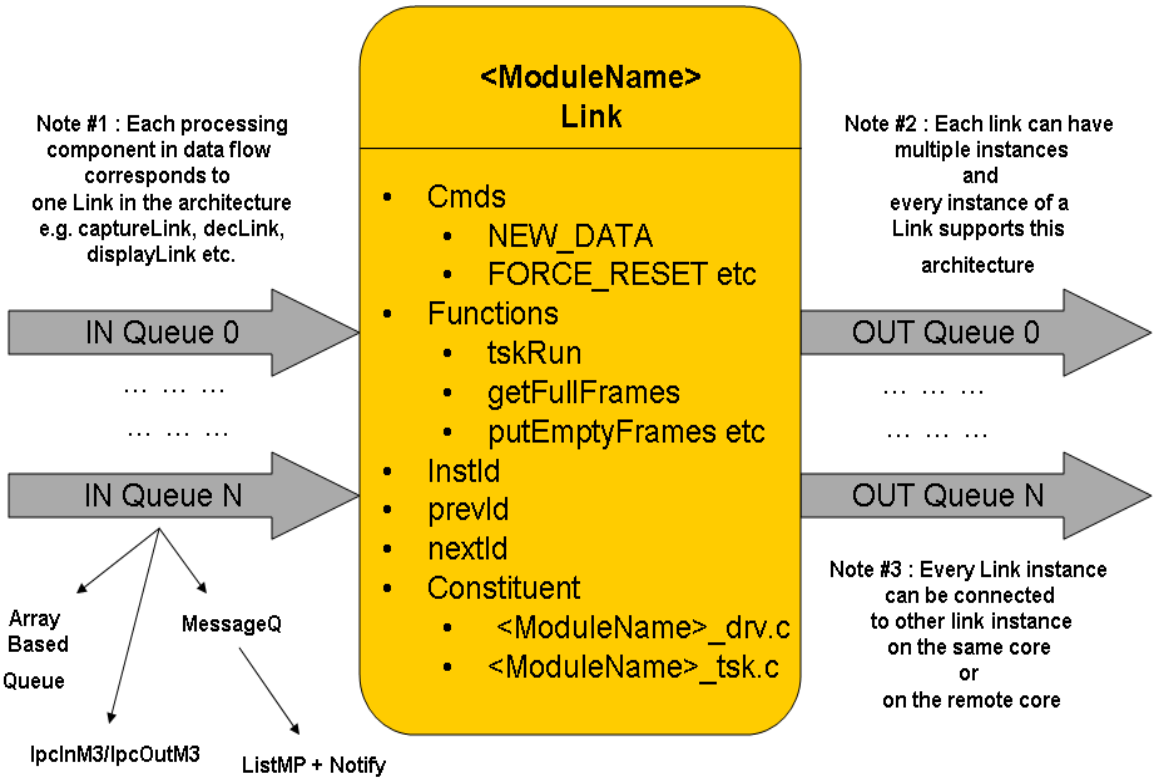


3.3 Link API

NOTE: Most users don't need to know low level details of the internal software architecture, but it useful to know how the software operates internally in order to get the most out of the system.

- A link is the basic processing step in a video data flow. A link consists of a BIOS6/Linux thread coupled with a message box (implemented using OS semaphores). Since each link runs as a separate thread, links can run in parallel to each other. The message box associated with a link allows user application as well as other links to talk to that link. The link implements a specific interface which allows other links to exchange video frames and/or bit streams with the link.
- Link API allows user to create, control and connect the links.
- McFW API uses Link API to make a chain depending on the top level system configuration provided by the user.
- Alternatively user's can use the link API directly to make custom use-cases not supported by the McFW API.

3.3.1 Internal Software Architecture of "Links"



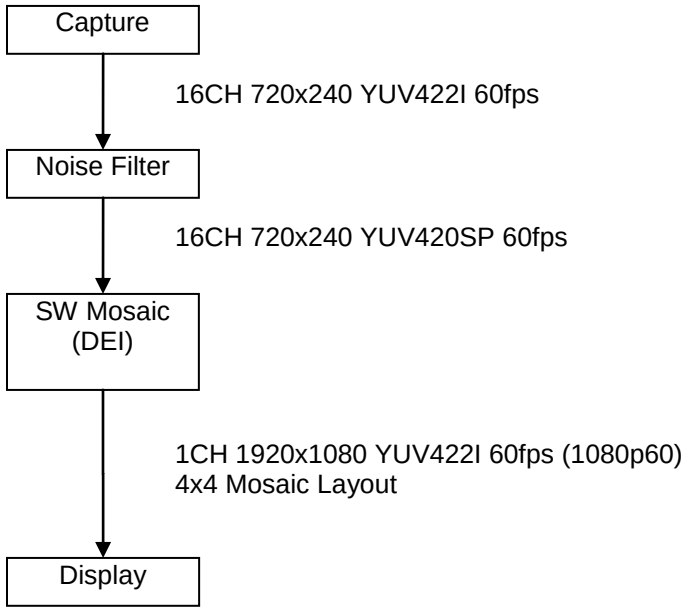
The SW Architecture used internally in McFW is based on the following principles,

- The video processing workload is divided between different processors as shown below

Processor	OS	Used for
HOST A8	Linux	System setup and control, GUI, IO peripheral control like SATA,

		Ethernet, USB, Audio
VPSS M3	BIOS6	HDVPSS control for video capture, video display, scaling, de – interlacing
Video M3	BIOS6	HDVICP2 Video compression / decompression (H264 encode, H264 decode)
DSP	BIOS6	SW OSD, custom video processing algorithms

- Within each processor, each processing step like capture or display will run in its own independent thread. Such a independent thread of execution is called a "link" in this framework.
- Example links include, capture, display, DEI, Noise Filter, encode, decode
- Each thread or link is capable of handling processing of video frames from multiple channels, each having different properties like width, height, data format etc.
- A link will "connect" to other links to make a chain or a data flow. This connection and control can be done by the user from the HOST A8 side.
- Once a chain is setup and started, each link in the chain will exchange frames will its next link, using a well defined interface, to make the video processing data flow.
- The framework allows links on different processors to exchange frames directly with each other without any intervention of the HOST A8.
- Once a chain is running, user can send control commands to individual links to control their run-time behavior. Example, changing Mosaic layout for the display.
- An example chain is shown below, in this chain multiple channels of video are captured, processed via SW Mosaicing and then displayed on a display device



3.3.2 Link Interface

A link interface consists of

- Link API – which is used by chains or user applications for configuring and controlling the link
- Inter Link API – which is used by other links for exchanging frames between two links
- Link Output Queue – is the queue which is used by another link (via the inter link API) to exchange frames with that link

3.3.3 Passing messages to a link

Each link is identified by a system wide unique 32-bit link ID as defined in "system_linkId.h".

The link ID determines on which processor the link runs as shown below.

Bits	Description
0..27	Link ID
28..31	Processor ID on which this link runs 0: DSP 1: Video M3 2: VPSS M3 3: HOST A8

Each link API needs the link ID as an argument when sending a message to the link.

When a message is sent to a link by the user, based on the link ID the function internally knows whether this is a local link, in which case it sends the message using normal BIOS/Linux APIs, else it will use Syslink MessageQ to send the message to the appropriate processor. Once the MessageQ message reaches the target processor, it invokes the local BIOS APIs to forward the message to the intended link. This allows user to control the links on VPSS M3, Video M3 and DSP from HOST A8. The user need not directly know about the processor and mechanism (Syslink message Q) that is used for this message passing.

3.3.4 Link API

This API allows a link to be controlled by the user or chains.

The following APIs are part of the link API.

The API arguments are typically specific to the link implementation.

API	Description
System_linkCreate	Creates a link - allocates driver, codec, memory resources.
System_linkGetInfo	Get information about a link like number of channels, properties of each channel. MUST be called after System_linkCreate() for a link
System_linkStart	Start the link - starts the driver or codec

System_linkControl	Send a link specific control command with optional arguments
System_linkStop	Stop the link - stops the driver or codec
System_linkDelete	Deletes a link - free's driver, codec, memory resources

3.3.5 Inter Link API

This API is used by links to exchange frames with each other. Users of a link typically need not be aware of this API.

Each link needs to implement a few functions and register the function pointers with the system frame work along with its link ID. This registration is done once during system init.

API	Description
System_GetLinkInfoCb	Function to return information about a link like number of channels, properties of each channel
System_LinkGetOutputFramesCb	Function to return captured or generated or output frames to the caller (another link)
System_LinkPutEmptyFramesCb	Function to release consumed frames back to the original link for reuse
System_LinkGetOutputBitBufsCb	Function to return generated or output bitstream frame to the caller (another link) – Valid only for Encode Link
System_LinkPutEmptyBitBufsCb	Function to release consumed bitstream frames back to the original link for reuse – Valid only for Encode Link

Any link which wants to get frames to/from another link will use the system API "System_getLinksFullFrames()" to get frames from the previous link. This internally will index into the system wide link information table and invoke the link specific System_LinkGetOutputFramesCb() function callback.

Similarly when a link wants to release the frames back to the original link after the frames have been consumed, it will call the API "System_putLinksEmptyFrames()". This internally will index into the system wide link information table and invoke the link specific System_ LinkPutEmptyFramesCb () function callback.

This way a link need not exactly know which link it is exchanging frames with. All it needs is a link ID of the previous link in the data flow. This allows user to user the same link in many different data flows without modifying the link implementation.

3.3.6 Link Output Queues

A link will have one or more output queues into which it will put the captured or generated frames. A link owns it output queue and takes care of memory allocation for the frames that will go into its output queue.

Most links have only one output queue, but some links have multiple output queue's. These multiple output queue's allow that link to be used in different data flows without changing the link implementation.

Example, Noise filter link can be configured to output its channel frames over two output queues, such that 8CH of 16CH goto one output queue and other 8CH go to other output queue. This allows noise filter to feed to two different DEI links in some data flows.

An output queue can hold frames from multiple channels of multiples sizes and different data formats. i.e it's a heterogeneous queue.

The information of the content in the queue can be known by using the System_linkGetInfo() API. This internally will call the link specific System_GetLinkInfoCb() function callback.

The data structure FIVD2_Frame, used by VPSS driver, is used for exchanging frame information between links. This allows frame information to flow between links without any additional translation. Among other information it has a "channelNum" field which allows a link to indentify the channel with the frame data.

A link will typically call the System_getLinksFullFrames() with the link ID and queue ID of the previous link when it wants to process the input frames.

A link when it has generated output frames for consumption by the next link will send a message "SYSTEM_CMD_NEW_DATA" to the next link.

When a link receives "SYSTEM_CMD_NEW_DATA" it will call System_getLinksFullFrames(). After processing the input frames it will release the input frames using System_putLinksEmptyFrames()

Thus a link needs to know

- Previous link ID and Previous Link Queue ID to get input frames
- And Next Link ID, in order to inform the next link when new frames are generated.

This information of previous link ID and next link ID is passed to a link using the System_linkCreate() API.

Thus previous link ID and next link ID is what "connects" one link to another link.

3.3.7 IPC Link

A special link called IPC (Inter processor communication) Link is used for exchanging frames across processors.

Example, A local processor link like capture will exchange frames with the IPC Link and the IPC Link will turn make use of use of appropriate Syslink/IPC APIs to send the received frames across processors.

Thus a capture link can be implemented as if it will only talk to a link on the same processor. This will keep the implementation of the capture link simple and efficient.

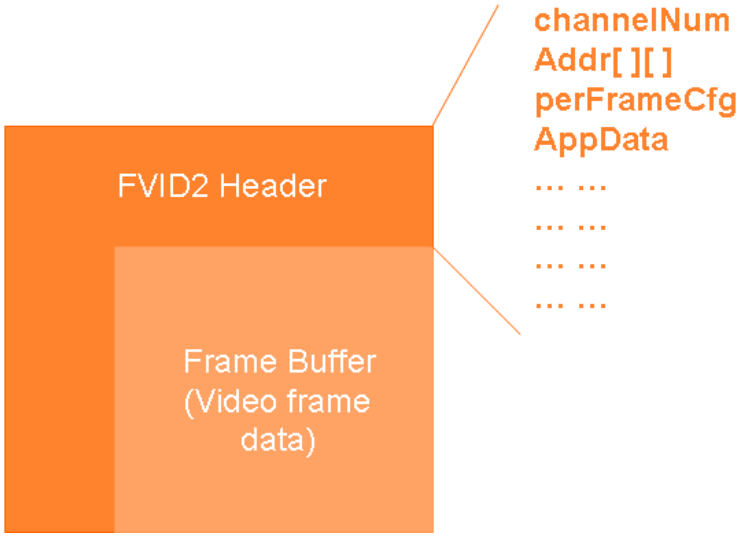
The special IPC link will handle the complexity of sending frames between processors and it will take care of any cache operations or other such inter processor synchronization functions.

The inter link frame exchange mechanisms that are used, depend on where the links are located in order to reduce inter link frame exchange overheads.

The following three inter link frame exchange mechanisms are used

- Intra-processor links
  - Example, from capture to noise filter which run on the same processor.
  - Simple and efficient array based queue's are used for frame exchange.
- Inter M3 (Video / VPSS) links
  - Example from NF to encode (via IPC M3 OUT/IN Link) which run on VPSS M3 and Video M3 (sharing a uni-cache).
  - IPC ListMP with Notify is used with frame information pointer (FVID2_Frame) being passed directly without any cache operations and address translation since both M3 share the same uni-cache.
- Inter processor (M3 to A8 or DSP)
  - Example from encode to Bitstream IN (via IPC OUT/IN Link) which run on Video M3 and Host A8.
  - IPC ListMP with Notify is used with frame information being passed with address and information translation including cache operations, if applicable

3.3.8 Frame Structure



- Links exchange frames. The frame buffers (video data) can be sent across processors for processing using these frames.
- The FVID2 header has sufficient information that can be used to in transportation of frame buffers.
- FVID2 header has many parameters but as an application writer its important to understand following ones
  1. channelNum - Identifies channel
  2. Addr[][] - Pointer to the frame buffer
  3. perFrameCfg - Information about frame configuration

4. AppData - Application data can be plugged in here, currently it has systemInformation

3.4 Chains

NOTE: Most users don't need to know low level details of how to make chains since McFW API hides this detail from the user, but its useful to know how the software operates internally in order to get the most out of the system.

A chain is connection of links in a logical order to make a video processing data flow.

A chain is constructed using Link APIs.

Different chains can be constructed for different applications. Chains can be "destroyed" and then reconstructed in a different ways without needing a reboot.

McFW API constructs some pre-defined chains depending on top level system configuration passed by the user.

3.4.1 Chain execution sequence

A chain execution is done in the following sequence.

- A chain creates the links using System_linkCreate() API. Links MUST be created in the order of source link to sink link.

  A source is a link which has no previous link. Example, Capture
  A sink is a link which has no next link. Example, Display

  The order of creating a link is important since when a link is created it queries its previous link to get information about the expected number of channels and channel properties, based on a which a link will configure itself. Hence a "previous" link needs to created before the "next" link
- Next the chain is started by calling System_linkStart() for each link in the chain. The links are started in the order of sink link to source link. This is not a hard requirement but is usually desired in the application, since this allows all links to get ready to receive input before the source link itself is started.
- Once a chain is running control commands can be sent to individual links to control the data flow. System_linkControl() API is used to send control commands. Example, command to SW Mosaic Link to chang Mosaic layout at run-time.
- The control commands that are supported depend on the type of link.
- NOTE, depending on the link System_linkControl() can be called even before calling System_linkCreate().
- After the user is done with a chain he can stop it by calling System_linkStop() API for each link of the chain. A chains MUST be stopped in the order of source to sink.

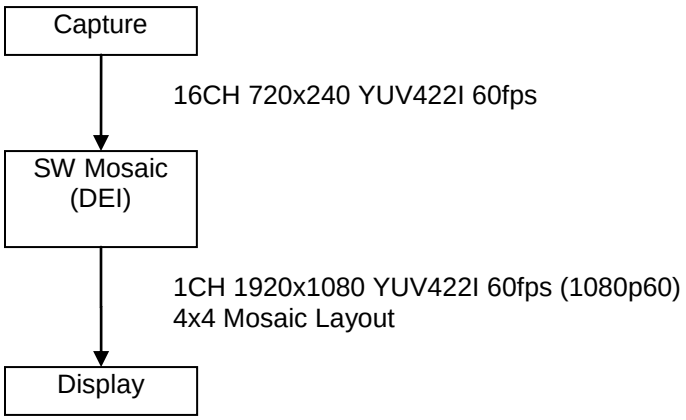
  A link can wait on a output buffer which could be held by the next link. If the next link is stopped before a previous link is stopped, then the previous could wait for ever on a buffer which never arrives since the next link is no longer active. Hence a previous link MUST be stopped before stopping the next link.

- Finally after a chain is stopped, it can be destroyed by calling System_linkDelete() API on each link in the chain. Link delete can be called in any order on the links.
- After a chain is deleted a new chain can be created and started by using the same sequence.

3.4.2 Chain Example Code

An example code for a Multi-CH capture to SW Mosaic to Display chain is shown below. Some details are left out for clarity. Refer to actual source code (chains_multiChCaptureNsfDei.c) for details.

Data flow



Link include files

```
#include <ti/vsi/interfaces/system.h>
#include <ti/vsi/interfaces/captureLink.h>
#include <ti/vsi/interfaces/displayLink.h>
#include <ti/vsi/interfaces/swMsLink.h>
#include <ti/vsi/interfaces/systemLink_m3vpss.h>
```

Link Create Parameters structures

```
CaptureLink_CreateParams    capturePrm;
SwMsLink_CreateParams       swMsPrm;
DisplayLink_CreateParams    displayPrm;
```

Setup Capture link parameters

```
capturePrm.tilerEnable = FALSE; // do not use tiler memory for output
capturePrm.numVipInst = 4; // use 4 video ports

// next link after capture link is SW Mosaic DEI link
capturePrm.outQueParams[0].nextLink = SYSTEM_LINK_ID_SW_MS_DEI_0;
```

```
// configure for each video capture port
for(vipInstId=0; vipInstId<capturePrm.numVipInst; vipInstId++)
{
  // set capture port ID
  capturePrm.vipInst[vipInstId].vipInstId =
          SYSTEM_CAPTURE_INST_VIP0_PORTA+vipInstId;

  // set capture external device ID
  capturePrm.vipInst[vipInstId].videoDecoderId     =
          SYSTEM_DEVICE_VID_DEC_TVP5158_DRV;

  // set input data format as 16-bit YUV422
  capturePrm.vipInst[vipInstId].inDataFormat =
          SYSTEM_DF_YUV422P;

  // set capture video standard as 4CH multiplxed D1 capture (per port)
   capturePrm.vipInst[vipInstId].standard =
           SYSTEM_STD_MUX_4CH_D1;

   // number of output per channel for a port is one
   capturePrm.vipInst[vipInstId].numOutput =

    // output data format is YUV422I
    capturePrm.vipInst[vipInstId].outParams[0].dataFormat =
             SYSTEM_DF_YUV422I_YUYV

     // inline scaling disabled since this is multi-channel capture
     capturePrm.vipInst[vipInstId].outParams[0].scEnable = FALSE;

     // all channels map to output que ID = 0
     capturePrm.vipInst[vipInstId].outParams[0].outQueId = 0;
}
```

Setup SW Mosaic link parameters

```
// previous link is capture, previous link que ID = 0
swMsPrm.inQueParams.prevLinkId = SYSTEM_LINK_ID_CAPTURE;
```

```
swMsPrm.inQueParams.prevLinkQueId = 0;

// next link is display
swMsPrm.outQueParams.nextLink     = SYSTEM_LINK_ID_DISPLAY_1;

// SW Mosaic invocation period is 16ms or 60fps
swMsPrm.timerPeriod               = 16;


// Initial layout is 4x4 and display frame size is 1080p60
swMsPrm.layoutPrm.outLayoutMode = SYSTEM_LAYOUT_MODE_16CH;
swMsPrm.layoutPrm.outRes = SYSTEM_DISPLAY_RES_1080P60;


// CHx mapped to WINx. Max 16 windows possible in 4x4 layout.
for(winId=0; winId<16; winId++)
   swMsPrm.layoutPrm.win2ChMap[winId] = winId;
```

Setup Display Link parameters

```
// previous link is SW Mosaic link, previous link que ID = 0
displayPrm.inQueParams.prevLinkId    = SYSTEM_LINK_ID_SW_MS_DEI_0;
displayPrm.inQueParams.prevLinkQueId = 0;


// display resolution is same as SW Mosaic link output frame size, 1080p60
displayPrm.displayRes                = swMsPrm.layoutPrm.outRes;
```

Initialize the display controller – this is required in order to enable the display. There is no separate display controller link. Instead a command is sent to the generic VPSS Link which in turn calls the display controller API on the VPSS M3 processor.

```
SystemVpss_DisplayCtrlInitParam prm;


// display controller resolution is same as display link resolution
prm.hdDisplayRes = displayPrm.displayRes;


// send command to VPSS M3 link to initialize the display controller
System_linkControl(
            SYSTEM_LINK_ID_M3VPSS,
            SYSTEM_M3VPSS_CMD_GET_DISPLAYCTRL_INIT,
            &prm,
            sizeof(prm),
```

```
            TRUE
            );
```

Create the links

```
// create in the order of source to sink
System_linkCreate (SYSTEM_LINK_ID_CAPTURE, &capturePrm, sizeof(capturePrm));
System_linkCreate(SYSTEM_LINK_ID_SW_MS_DEI_0, &swMsPrm, sizeof(swMsPrm));
System_linkCreate(SYSTEM_LINK_ID_DISPLAY_1, &displayPrm, sizeof(displayPrm));
```

Start the links

```
// start in the order of sink to source
System_linkStart(SYSTEM_LINK_ID_DISPLAY_1);
System_linkStart(SYSTEM_LINK_ID_SW_MS_DEI_0);
System_linkStart(SYSTEM_LINK_ID_CAPTURE);
```

Run-time Control the links

```
while(1)
{
   //   sleep few seconds
   sleep(10);

  //  change mosaic layout, keep other layout parameter same as init time parameters.
  swMsPrm.layoutPrm.outLayoutMode = SYSTEM_LAYOUT_MODE_7CH_PLUS_1CH;
  System_linkControl(
          SYSTEM_LINK_ID_SW_MS_DEI_0,
          SYSTEM_SW_MS_LINK_CMD_SWITCH_LAYOUT,
          &swMsPrm.layoutPrm,
          sizeof(swMsPrm.layoutPrm),
           TRUE
           );

  // check if chains execution is done based on user input
  if(userIsDone())
     break;
}
```

Stop the links

```
// stop in the order of source to sink
System_linkStop(SYSTEM_LINK_ID_CAPTURE);
```

```
System_linkStop(SYSTEM_LINK_ID_SW_MS_DEI_0);
System_linkStop(SYSTEM_LINK_ID_DISPLAY_1);
```

Delete the links

```
// any order is fine for delete
System_linkDelete(SYSTEM_LINK_ID_CAPTURE);
System_linkDelete(SYSTEM_LINK_ID_SW_MS_DEI_0);
System_linkDelete(SYSTEM_LINK_ID_DISPLAY_1);
```

De-init the display controller

```
// send command to VPSS M3 link to de-initialize the display controller. No parameters are passed for de-init
System_linkControl(
        SYSTEM_LINK_ID_M3VPSS,
        SYSTEM_M3VPSS_CMD_GET_DISPLAYCTRL_DEINIT,
        NULL,
        0,
        TRUE
        );
```

4 Directory Structure

4.1 Interface files

The interface files for the McFW can be found at the below location. All the interface APIs are callable from Host A8 side.

Interface files base path	
dvr_rdk\mcfw\interfaces	
McFW API - Interface files **dvr_rdk\mcfw\interfaces**	
ti_vcap.h	Video Capture Sub-system Interface
ti_vdec.h	Video Decode Sub-system Interface
ti_vdis.h	Video Display Sub-system Interface
ti_venc.h	Video Encode Sub-system Interface
ti_vsys.h	System Configuration Interface
ti_media_common_def.h	Common Data structure definitions
ti_media_error_def.h	Error codes
ti_media_std.h	Data Types
Link API - Interface files **dvr_rdk\mcfw\interfaces\link_api**	
system.h	Common Link APIs and data structures
system_common.h	System level const, data structure, functions which are common across all processors. User does not use the const's, function's, data structure defined in this file directly
system_const.h	Common system wide constants and enum's. User MAY use the const's, defined in this file in the link API
system_debug.h	#define's to control printing of debug messages. If modified this needs a rebuild of BIOS and linux side code for the change to take effect.
system_linkId.h	32-bit link ID, proc ID and other utility macros to operate on the link ID
system_tiler.h	APIs to allocate memory from Tiler region
systemLink_common.h	System Link API which is common across all slave processor's (VPSS M4, Video M3, DSP)
systemLink_m3video.h	System Link API specific to Video M3 processor
systemLink_m3vpss.h	System Link API specific to VPSS M3 processor
captureLink.h	Video Capture Link API

decLink.h	Video Decoder (H264) Link API
deiLink.h	Deinterlacer Link API
displayLink.h	Display Link API
dupLink.h	Frame Duplicator Link API
encLink.h	Video encoder (H264) Link API
grpxLink.h	Graphics Link API
ipcLink.h	IPC (Inter-processor communication) Link API
ivacodecif.h	IVA-HD interface. NOT used by user directly.
mergeLink.h	Frame merge Link API
nsfLink.h	Noise filter Link API
nullLink.h	Dummy / Null sink Link API
nullSrcLink.h	Dummy / Null source Link API
swMsLink.h	Software Mosaic Link API
vidbitstream.h	Video bitstream data structure's. NOT used directly by users.

4.2 Demo Example code

McFW API Examples	
dvr_rdk\demos\mcfw_api_demos	Multiple demos showing usage of McFW APIs
Link API Examples	
dvr_rdk\demos\link_api_demos	Multiple demos of links connected in different ways to form a chain using link API

Note: The support for link API demos is getting deprecated. Please refer to McFW interface implementation as an example of Link API usage.

4.3 Libraries

McFW API Examples	
dvr_rdk\demos\mcfw_api_demos	Multiple demos showing usage of McFW APIs

5 Additional Details

5.1 Inter Processor Communication Details

5.2 Memory Map Details

Refer to DM81xx_DVRRDK_MemoryMap document in AppNotes

5.3 Display Controller and Display Features Details

Refer to DM81xx_DVRRDK_Display_Output_Configuration document in AppNotes

5.4 Progressive / Interlaced McFW Usecases

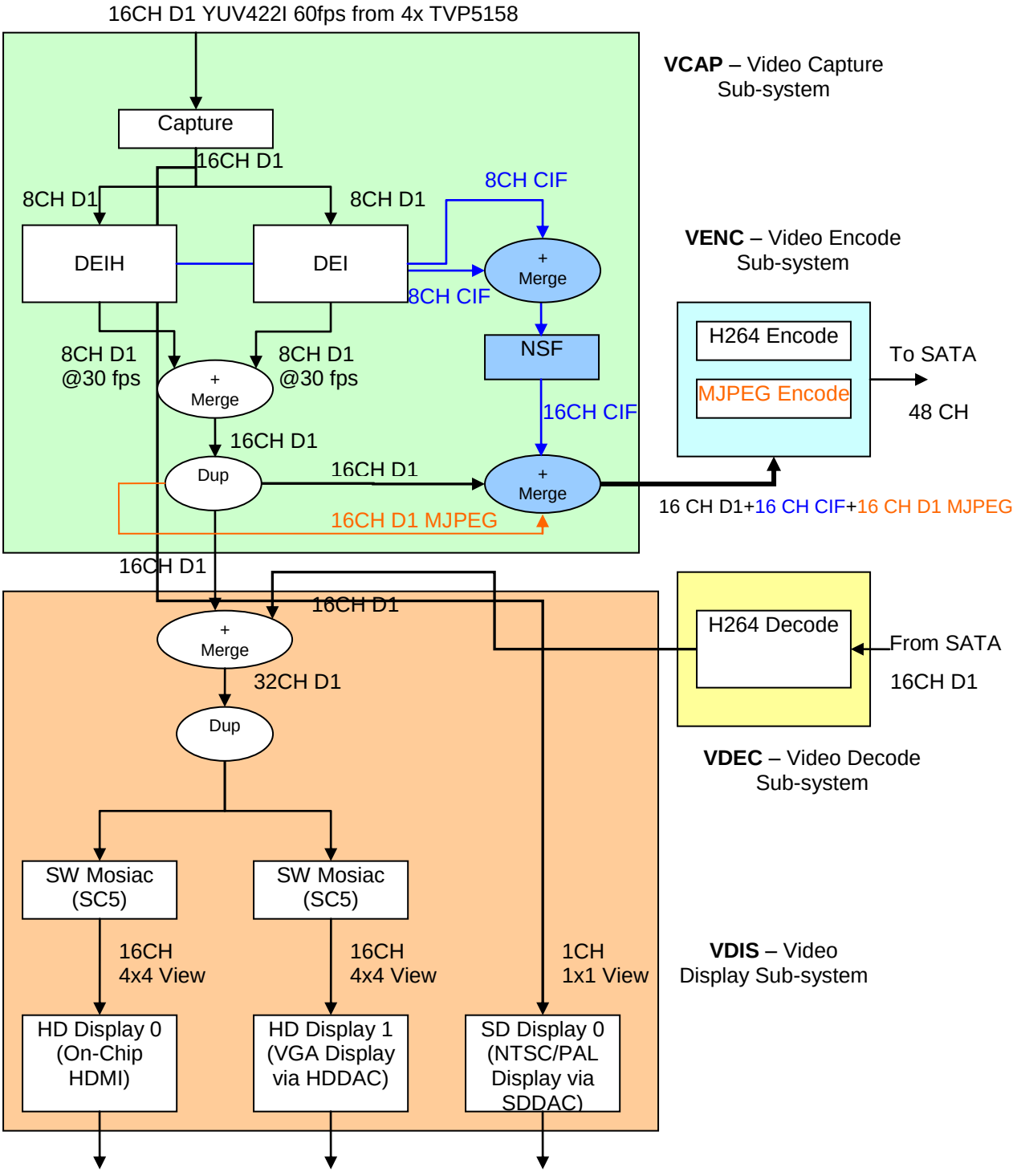
There are applications to demonstrate capture, encode, decode, display in progressive & interlaced modes. In progressive mode, dual output is possible. Along with 16 D1 Channels <16 number is based on channels input to capture>, 16 CIF channels and 16 D1 MJPEG channels (duplicated 16 D1 captured channels) can be encoded and exported to host which can be stored. 16 channels D1 input with dual output is available only on progressive application. In DM814x, 4 channel input demos are available outputting 4 D1, 4 CIF and 4 D1 MJPEG channels. D1 MJPEG channel will be encoded at 1 FPS in progressive demo for both Dm816x and Dm814x.

System Configuration interface parameter - enableSecondaryOut should be set to enable CIF output in System usecase defined as VSYS_USECASE_MULTICHN_PROGRESSIVE_VCAP_VDIS_VENC_VDEC.

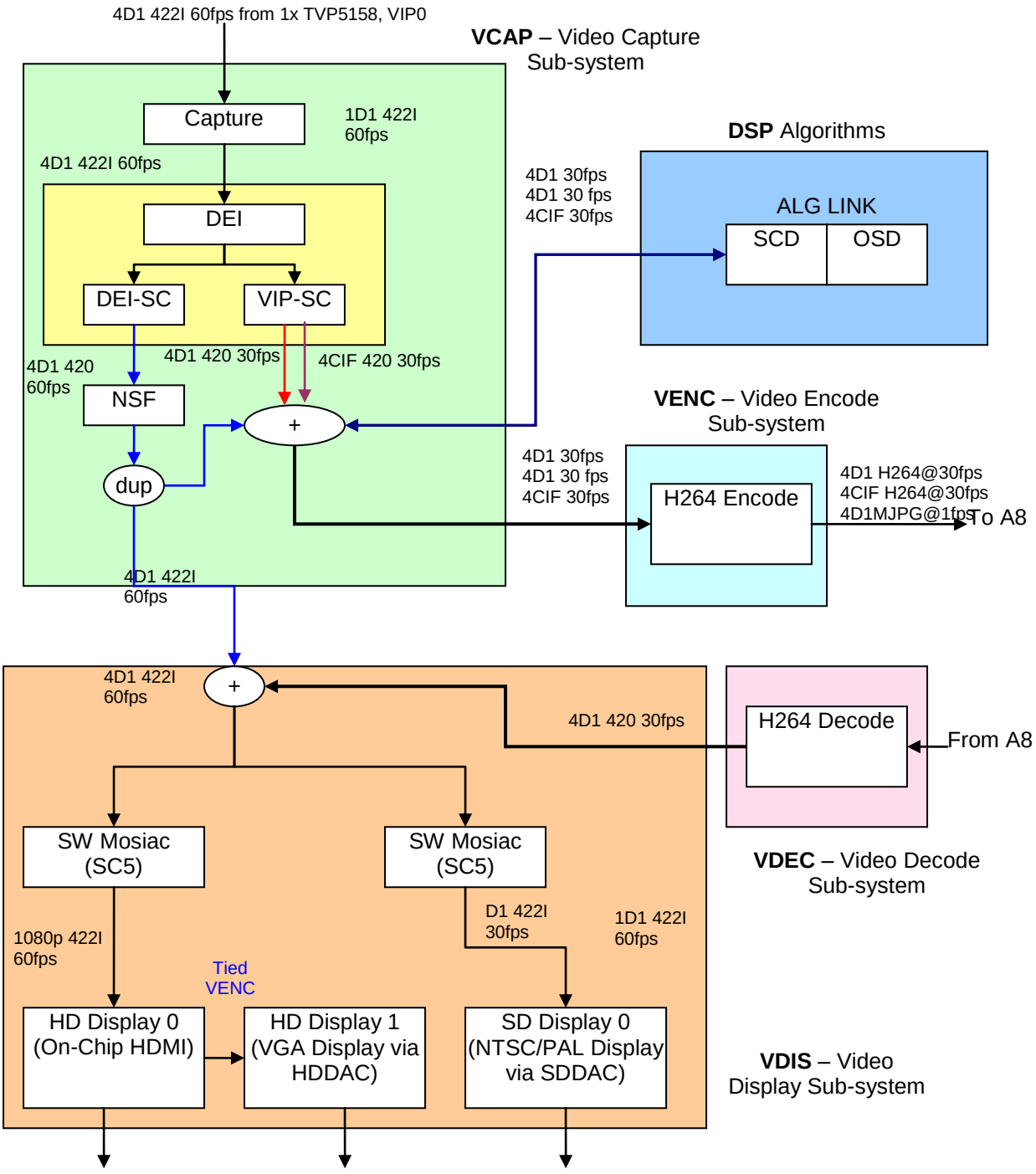
Dual output is not available in interlaced usecase defined as VSYS_USECASE_MULTICHN_INTERLACED_VCAP_VDIS_VENC_VDEC.

Secondary output fps can be controlled using Vcap_setFrameRate(). Any random frame rate is not supported <refer limitation note>. Stream 1 maps to CIF channel at VCAP level. At encoder <also at host>, channels 0~15 map to D1 channels, channels 16~31 map to CIF channels and channels 32~47 map to D1 MJPEG channels. In DM814x, channels 0~3 map to D1 channels, 4~7 map to CIF channels and channels 8~11 map to D1 MJPEG channels. Stream 0 / 1 concept is not applicable in encoder system as capture gives out all channels in a single queue. CIF bitrate is controlled using Venc_setDynamicParam() similar to D1 channels. Since only 5 CIF frames <in demo app> are given to encoder which still expects 30 fps, CIF bitrate calculation should be done accordingly to get expected result. Primary & secondary output resolutions can be controlled using absolute width or ratio on detected capture resolution. Refer the respective system usecase for exact details on using these parameters.

Progressive Demo <16 Channel Mode, Dual Output>



Progressive Demo <4 channel D1> - DM814X



5.5 SWOSD Details

Refer to DM81xx_DVRRDK_SWOSD_UserGuide in UserGuides folder.

5.6 Video Bitstream Buffer Interface

DVR RDK provides an interface to access the video bitstream. Application can receive / send multiple channels of encoded bitstream from the encoder / to the decoder.

McFW provides the following APIs for interacting with video bitstream

- Receiving encoder bitstream:
  - ***Venc_getBitstreamBuffer –*** API to get multiple channels of encoded bits
  - ***Venc_releaseBitstreamBuffer –*** API to free bitstream buffers consumed by the application

- Sending encoded bitstream:
  - ***Vdec_requestBitstreamBuffer –*** API to get empty bitstream buffers
  - ***Vdec_putBitstreamBuffer –*** API to send filled bistream buffers for decoding

- The key data structure having the bitstream info is:
  - **VCODEC_BITSBUF_LIST_S**.
  - Defined in dvr_rdk/mcfw/interfaces/ti_media_common_def.h as

```
typedef struct {
   UInt32            numBufs;
   VCODEC_BITSBUF_S  bitsBuf[VCODEC_BITSBUF_MAX];
} VCODEC_BITSBUF_LIST_S;
```

- The VCODEC_BITSBUF_LIST_S structure has the following members:
  - numBufs : Indicates the number of valid entries in the bitBuf array (details follow)
    - This number should be <= VCODEC_BITSBUF_MAX
  - bitsBuf: Array of VCODEC_BITSBUF_S structures
  - Each element in this array represents ***_one_*** encoded field/frame from a particular channel
- The VCODEC_BITSBUF_S has the following members:
  - reserved: For internal use in the McFW components. Application ***_must_not_*** modify this member
  - chnId: Channel ID. Should be from 0 to VENC_CHN_MAX-1
  - strmId: Encoder stream ID. Should be from 0 to VENC_STRM_MAX-1
  - codecType: Video compression format (eg:H264/MPEG4 etc..)

- frameType: Frame type (I_FRAME/P_FRAME/B_FRAME)
- bufSize: Size of bitstream buffer in bytes. Encoded frame size should be <= this bufSize
- filledBufSize: Actual size of encoded field/frame in bytes
- bufVirtAddr: Buffer start address of .the bitstream buffer. This is the user space virtual address.The application should use this address if it wants to access the bitstream buffer via CPU (eg: memcpy)
- bufPhysAddr: Physical address of start of bistream buffer.Can be used for EDMA APIs
- timestamp: Timestamp associated with the frame.
- fieldID: Indentifies whether this is a top field/bottom field/frame.
  - 0: Even field or Frame based, 1: Odd Field
- frameWidth: Width of encoded frame/field in pixels
- frameHeight: Height of encoded frame in lines.

Bit Receive Operation

The TI_VENC module in McFW framework supports APIs to receive multiple channels of encoded bistream.

The bitstream receive interface is defined in

- dvr_rdk/mcfw/interfaces/ti_venc.h

The following functions are related to bitstream receive:

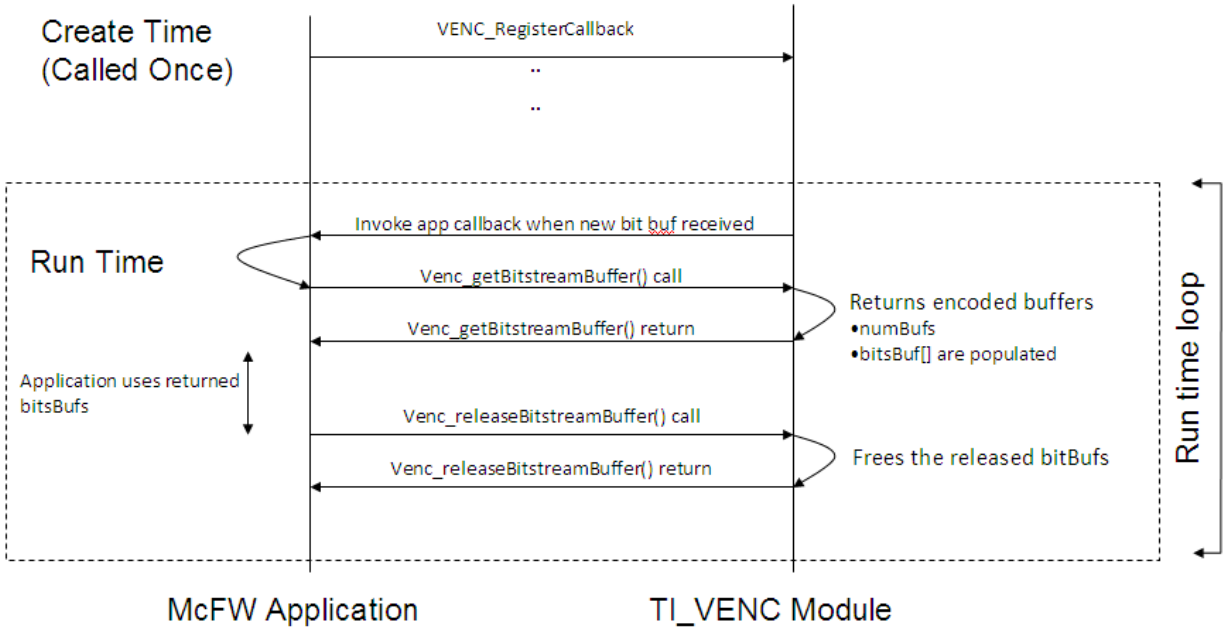
- Int32 **Venc_registerCallback**(VENC_CALLBACK_S * callback, Ptr arg)
  - Create time API to register application callback.
  - The callback will be invoked when new bistream buffers are available.
  - The application can use the callback to synchronize invocation of the getBitstream API.

- Int32 **Venc_getBitstreamBuffer**(VCODEC_BITSBUF_LIST_S *pBitsBufList, UInt32 timeout)
  - Runtime API to get filled bistream buffers from the encoder.
  - pBitsBufList: Pointer to VCODEC_BITSBUF_LIST_S structure. Will be populated by TI_VENC module.
  - timeout : TIMEOUT_WAIT_FOREVER or TIMEOUT_NO_WAIT or timeout in units of msec

- Int32 **Venc_releaseBitstreamBuffer**(VCODEC_BITSBUF_LIST_S *pBitsBufList)
  - Runtime API to release consumed bitstream buffers back to the encoder

– pBitsBufList: Pointer to VCODEC_BITSBUF_LIST_S structure passed by application.

Bits Receive Sequence Diagram



Bit Send Operation

The TI_VDEC module in McFW framework supports APIs to send multiple channels of encoded bistream to decode.

The bitstream receive interface is defined in

- dvr_rdk/mcfw/interfaces/ti_vdec.h

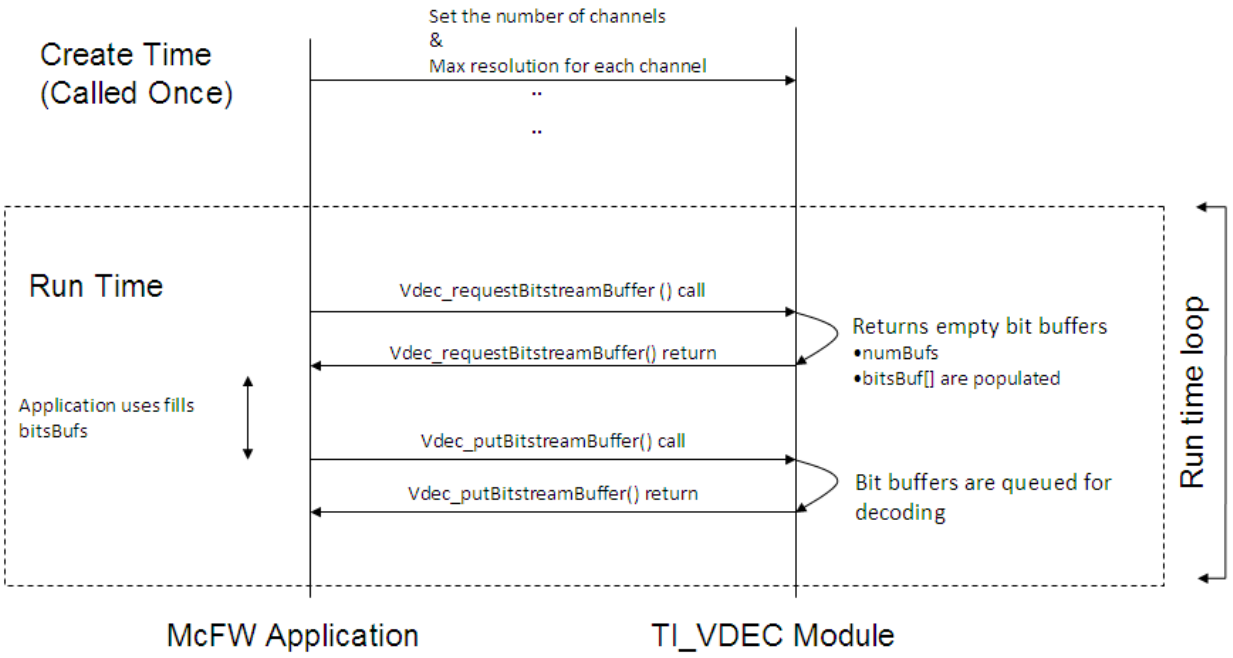
The following functions are related to bitstream send:

- Int32 **Vdec_requestBitstreamBuffer**(VDEC_BUF_REQUEST_S *bufReq,
  VCODEC_BITSBUF_LIST_S *pBitsBufList, UInt32 timeout)
  - Runtime API to get empty bitBuffers
  - bufReq: Information of number of buffers and size of each buffer
  - numBufs: Number of buffers requested by application
  - minBufSize: Minimum size of each buffer in bytes
  - pBitsBufList: Pointer to VCODEC_BITSBUF_LIST structure . Will be populated by TI_VDEC module

  - timeout: TIMEOUT_WAIT_FOREVER or TIMEOUT_NO_WAIT or timeout in msecs

- Int32 **Vdec_putBitstreamBuffer**(VCODEC_BITSBUF_LIST_S *pBitsBufList)
  - Runtime API to release consumed bitstream buffers back to the encoder
  - pBitsBufList: Pointer to VCODEC_BITSBUF_LIST_S structure passed by application.

Bits Send Sequence Diagram



5.7 Audio Video Synchronization Interface

DVR RDK provides an interface to enable and control the Audio-video synchronization. Application can configure Avsync using the following interfaces <avsync .h>.

- **AVSYNC_Init & AVSYNC_DeInit –** API to start /stop @ linux(A8 core).
- **AVSYNC_Control -** API to configure several parameters used in AVSync processing.

5.7.1 Interface to set / get parameters

Int32 AVSYNC_Control(UInt8 cmd, Void *pPrm)

Cmd – The possible commands are explained in next section.

pPrm – Each command will take a different parameter structure.

5.7.2 Commands used to set avsync structure params :

- ***AVSync_ComponentEnable :***
  - To enable / disable avsync processing for all channels.
  - Parameter structure used: ***AVSYNC_AvSyncEnable_t***

- ***AVSync_ChannelEnable***
  - To enable avsync processing of a particular channel. This should be set explicitly for a channel associated with a display.
  - Parameter structure used: ***AVSYNC_Enable_t***

- ***AVSync_ChannelDisable***
  - To disable avsync processing of a particular channel
  - Parameter structure used: ***AVSYNC_Enable_t***

- ***AVSync_StreamMode :***
  - To make avsync active for 1 channel or N channels.
  - Parameter structure used: ***AVSYNC_STREAM_MODE***

- ***AVSync_ExecutionMode***
  - To enable / disable avsync processing in free run mode or time sync mode.
  - Parameter structure used: ***AVSYNC_EXECUTION_MODE***

- ***AVSync_PlayRate***
  - To set speed for a particular channel.
  - Parameter structure used: ***AVSYNC_PlayRate_t***

- ***AVSync_StreamActive***
  - To set if a channel is active or not.
  - Parameter structure used: ***AVSYNC_StrmInfo_t***

- ***AVSync_TimeActiveRefClock***
  - To set a particular channel <either audio or video> as reference clock for avsync processing.
  - Parameter structure used: ***AVSYNC_RefClk_t***

- ***AVSync_ClientStartTime:***
  - To set the timestamp of the frame from which play back should be started.

- Parameter structure used: ***AVSYNC_StartPTS_t***

- ***AVSync_BackEndDelay:***
  - To set the Backend delay constant added to PTS to account for any delays occurring after avsync module
  - Parameter structure used: ***AVSYNC_BackendDelay_t***

- ***AVSync_MaxLeadLagTime:***
  - To set threshold for lead / lag times which define the allowed render time window.
  - Parameter structure used: ***AVSYNC_LeadLag_t***

- ***AVSync_PrintStats:***
  - To print the avsync statistics for all channels.
  - Parameter structure used: no params needed.

5.7.3 AVSync Configuration Data Strctures

- ***AVSYNC_AvSyncEnable_t***
  - avSyncCompEnable : 1 to Enable; 0 to Disable

- ***AVSYNC_Enable_t***
  - Display: 0 to Max_Displays.
  - Channel: 0 to Max_Channels for one display

- ***AVSYNC_STREAM_MODE***
  - AVSYNC_OneStreamMode : Set this flag if the usecase is just for 1 stream
  - AVSYNC_NStreamMode: Set this flag to operate in multi Stream Mode.

- ***AVSYNC_EXECUTION_MODE***
  - AVSYNC_ExecutionModeTimeSynced: to enable scheduling of frames according to their PTS
  - AVSYNC_ExecutionModeFreeRun: to switch off avsync logic.

- ***AVSYNC_PlayRate_t***
  - scaleM: Scaling factor applicable to each stream. Multiplier part
  - scaleD: Scaling factor applicable to each stream. Divider part
  - strmId: stream identifier.

  (ex): for 2x speed ➔ set scaleM as 2 & scaleD as 1
  for 0.5x speed ➔ set scaleM as 1 & scaleD as 2

- ***AVSYNC_StrmInfo_t***
  - nAudioStrmActiveMask: bit set to 1 indicates that audio stream is available for this strmId.
  - nVideoStrmActiveMask: bit set to 1 indicates that video stream is available for this strmId.

- ***AVSYNC_RefClk_t***

- strmId: stream identifier
- clkType: Specifies if this timestamp is for audio or video
<use AVSYNC_REFCLK_TYPE >

- ***AVSYNC_StartPTS_t***
  - strmId: stream identifier
  - clkType: Specifies if this timestamp is for audio or video.
  - nTimeStamp: TimeStamp of starting frame.

- ***AVSYNC_BackendDelay_t***
  - nDelay: Delay or lag / lead time in millisecs.
  - clkType: Specifies if this timestamp is for audio or video.

- ***AVSYNC_LeadLag_t***
  - audioLead: acceptable lead time in millisecs for audio.
  - audioLag: acceptable lag time in millisecs for audio.
  - videoLead: acceptable lead time in millisecs for video.
  - videoLag: acceptable lag time in millisecs for video.

5.8 Trick Play Interface

DVR RDK provides an interface to do forward / rewind trick play.

- Decoder output frames are controlled based on the trick play speed / direction configured by the application

- Application specifies incoming frame rate & expected output frame rate.

- Application support is also required to support various trick play rates

(Ex): for faster rates - 2X 4X etc which require decoding frames before skipping them to achieve the required trick play rate, application should give out a channel's content at a faster rate – at 2X rate for 2X speed on a specified channel. For I frame based trick play modes, application can just send I Frames & set the required AVSync speed which ensures display of the frames at the required time instance.

5.8.1 Trick Play - McFW API

"void *Vdec_setTplayConfig* (VDIS_CHN vdispChnId, VDIS_AVSYNC speed)"

- Set playback control configuration to trickplay logic.

vdispChnId - decode channel ID

speed - trickplay speed.

This will internally sends the params to decoder using the command "DEC_LINK_CMD_SET_TRICKPLAYCONFIG" and set to trick play structure.

"void *Vdis_setAvsyncConfig* (VDIS_CHN vdispChnId, VDIS_AVSYNC speed)"

Set the AVSync playback speed. AVsync sends out frames based on the speed set.

vdispChnId - display channel ID

speed - trick play speed.

This will internally call **AVSYNC_Control** (*AVSync_Play, AVSYNC_PlayRate_t*).

5.8.2 Example <decoder-display demo>

Trick play mode playback is available decode-display demo.

The following McFW API's are called sequentially to set playback speed to avsync, Trickplay and application.

Vdis_setAvsyncConfig (chId, speed);

Vdec_setTplayConfig (chId, speed);

VdecVdis_setTplayConfig (chId, speed);

5.9 Motion Vector Data Interface for Encoder

Two data structures related to interpreting the Motion Vector (MV) data being given out by the encoder have been provided in ti_venc_common_def.h file.

These two structures namely EncLink_h264_AnalyticHeaderInfo & EncLink_h264_ElementInfo have been provided for ease of quickly interpreting and using the MV data.

This section describes the method to access MV and SAD (Analytic Information) data dumped by the encoder.

5.9.1 Description

The Motion Vector and SAD Access API is a part of the XDM process() call, used by the application to encode a frame. A parameter enabledAnalyticinfo is provided as a part of create time parameters, which can be set or reset at a frame level during create-time. Setting this flag to non-zero value indicates that the analytic info is needed. When this parameter is set to non-zero value, the process() call returns the motion vector and SAD data in the buffer provided by the application.

For every macro block, the data returned is 10 bytes, a signed horizontal displacement component (signed 16-bit integer) and a vertical displacement component (signed 16-bit integer) in L0 and L1 direction and SAD (16-bit integer).

The following sequence should be followed for Analytic Info access:

1) In the create time parameters, set the flag to access analytic data.

```
/* Enable MV access */
createParams ->enableAnalyticinfo = 1;
```

2) Allocate output buffers and define the output buffer descriptors

```
/* Output Buffer Descriptor variables */
XDM2 BufDesc  outputBufDesc;
/* Get the input and output buffer requirements for the
codec */
```

```
control(.., XDM_GETBUFINFO, extn dynamicParams, ..);
```

If Analytic info access is enabled in step1, this call returns the output buffer info as numBufs =2, along with the minimal buffer sizes.

```
/* Initialize the output buffer descriptor */
                    outputBufDesc.numBufs = 2;
/* Stream Buffer */
outputBufDesc.descs[0].buf = streamDataPtr; //pointer
to H264 bit-stream
outputBufDesc.descs[0].bufSize.bytes =
status.videnc2Status.bufInfo.minOutBufSize[0].bytes;


/* MV & SAD Buffer */
outputBufDesc.descs[1].buf = Output_Buffer_Base_Addr;
//pointer to MV and SAD data
outputBufDesc.descs[1].bufSize.bytes =
status.videnc2Status.bufInfo.minOutBufSize[1].bytes;
```

3) Call frame encode API

/* Process call to encode 1 frame */

process(.. ,.. , outputBufDesc, ..);

After this call, the buffer outputBufDesc.descs[1].buf will have SAD and Motion vector data. The data format of this buffer will be like,

AnalyticHeaderInfo	Data (MV and SAD)

Define a structure:

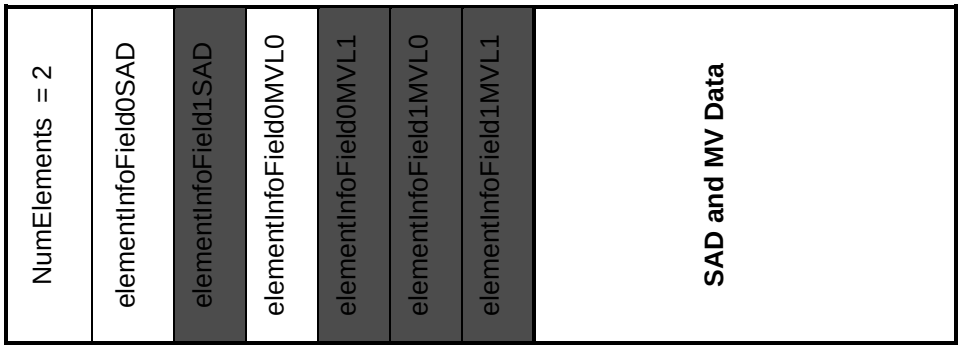
```
struct AnalyticHeaderInfo
{
U32 NumElements;
 ElementInfo elementInfoField0SAD;
ElementInfo elementInfoField1SAD;
ElementInfo elementInfoField0MVL0;
ElementInfo elementInfoField0MVL1;
ElementInfo elementInfoField1MVL0;
ElementInfo elementInfoField1MVL1;
} ;


Where as
NumElements -> Total number of elements in the buffer
(As of now SAD ,MV in L0 direction and MV in L1
direction for each field in case of interlace content)
ElementInfo is
```

```
typedef struct
{
/*Starting position of data from the buffer base
address*/
U32 StartPos;
 /* No. of bytes to jump from the current position to
get the next data of this element group */
U16 Jump;
/* Number of data elements in this group */
U32 Count;
}ElementInfo;
```

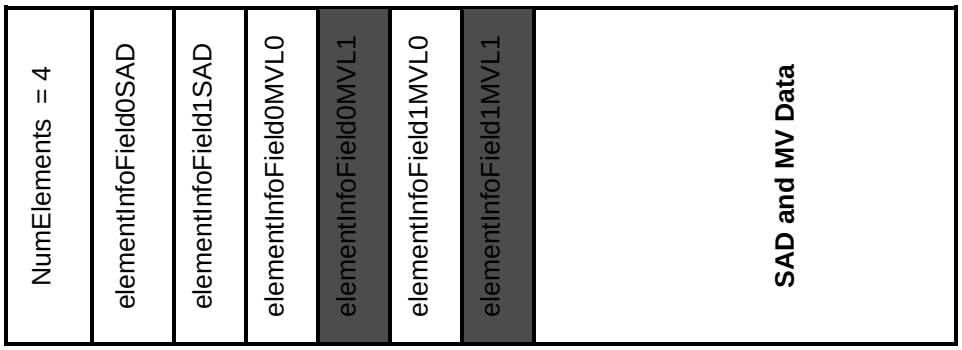
The data format will differ for each frame type; there can be four different formats as,

1. Process call which generates one P frame/field
2. Process call which generates two P fields
3. Process call which generates one B frame/field
4. Process call which generates two B fields

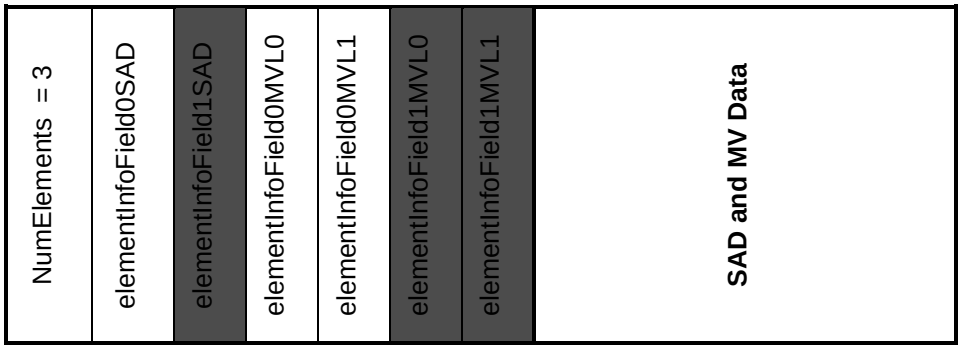
Process call, which generates one P frame/field:



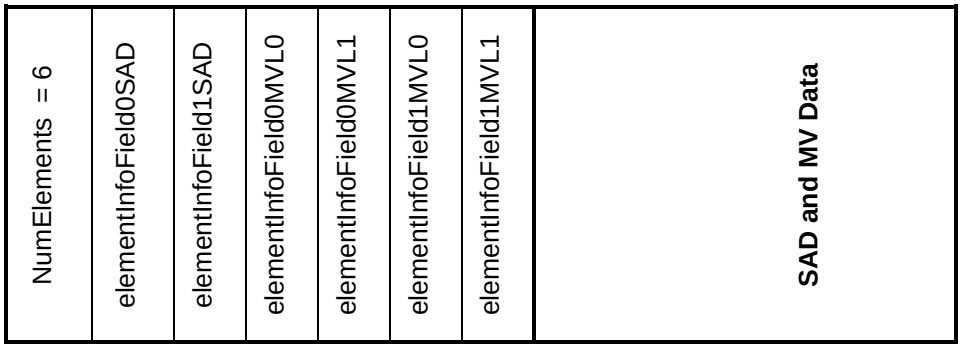
Process call, which generates two P fields:



Process call, which generates one B frame/field:



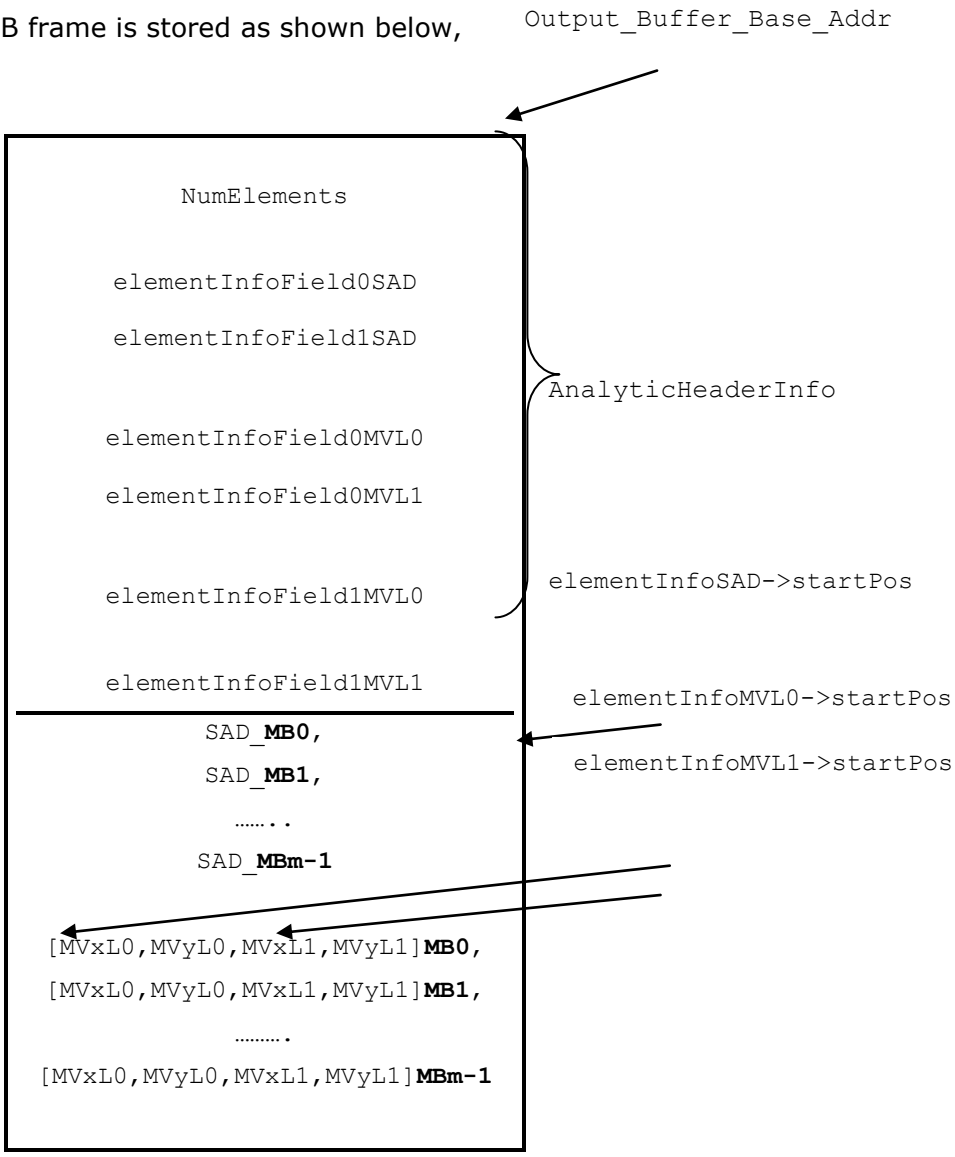
Process call, which generates two B fields:



Here data in shaded boxes are don't care values.

Example Usage

For example, consider the data in output buffer dumped by the codec for a B frame is stored as shown below,



To get the MVL0 data for all macroblocks, the application should have code as,

```
S16 *Src = (U32)Output_Buffer_Base_Addr +
    elementInfoMVL0->StartPos;
U16 Jump = elementInfoMVL0->Jump;
S16 *MVL0 = Addr_to_store_MV_inL0

for (i = 0; i < elementInfoMVL0->Count; i = i++)
```

```
{
    * MVL0 ++ = Src[i * Jump]; // To get MVx
    * MVL0 ++ = Src[((i *Jump) + 1)]; //To get MVy
}
```

Note: The structure EncLink_h264_ElementInfo has a 16bit member 'Jump', which is padded to 32 bits due to make it universal to all kinds of compilers.

bytes: $X^1X^2X^3X^4$ $X^5X^6I^7I^8$ $X^9X^{10}X^{11}X^{12}$

startPos: 1-4

jump: 5-6

ignore:7-8

count:9-12

5.10 AAC Encode / Decode

5.10.1 Feature

- AAC Encode / Decode <DSP based> feature available only on 816x/814x platforms
- Based on RPE Framework <remote call to DSP for encode & decode>
- High level ACAP APIs available to interact with DSP Encode / Decode algorithms
- AENC APIs
  - Provide APIs to create Encoder algorithm, do encode process.
- ADEC APIs
  - APIs to create Decoder algorithm, perform decode process.
- ACAP APIs
  - APIs to configure, start audio capture, retrieve captured data, and optionally get AAC encoded capture data.
  - ACAP APIs doing capture + encode assume TVP5158 based audio capture.

5.10.2 Audio APIs – Integration Note

- Encode / Decode System Init / DeInit
  - Audio_systemInit
  - Audio_systemDeInit

- Capture APIs
  - Acap_init
    - Initialize capture system, provide individual channel parameters – capture / encode buffers, set flag to optionally encode etc.
    - Will internally use AENC APIs if encode is requested.
  - Acap_start / Acap_startChannel
    - Start actual channel capture
  - Acap_stop / Acap_stopChannel
    - Stop channel capture

    - Acap_getData
        - Request captured raw or encoded data of individual channel.
    - Acap_setConsumedData
        - Notify after data consumption
- Encode APIs
    - Aenc_create
        - Create Encode algorithm <Only AAC right now>
    - Aenc_process
        - Encode processing call
    - Aenc_delete
        - Delete algorithm

- Decode APIs
    - Adec_create
        - Create Decode algorithm <Only AAC right now>
    - Adec _process
        - Decode processing call
    - Adec _delete
        - Delete algorithm

5.10.3 **Memory Requirement <for 1 channel>**

Memory Info	**Size**	**Location**	**Comment**
Capture Buffer	64 KB	SR 1	Can be put in any shared region. RPE buffer access mode flags should be set accordingly.
Encode Buffer	16 KB	SR 1	-Do-
Encode Buffer <Internal>	2 KB	SR 1	Depends on encoder's min output buffer
AAC Encoder memtab <for tvp5158 capture parameters - 16K sample rate, mono >	46 KB	SR 2	Can be put in any region. Refer Utils_getAlgMemoryHeapHandle()
AAC Decode memtab	17 KB	SR 2	For 16K sample rate, mono

Note
The capture buffers are tuned to avoid data loss with tvp5158 capture. Buffer size can be tuned depending on actual system without introducing capture data loss.

Refer demo guide on example usage of AAC Encode / Decode feature.

5.10.4 Limitations / Known Issues

1. Encode, Decode integrated only with Custom Encode demos. This is just a demo level limitation.
2. Capture + Encode, Decode can be enabled in any use case based on the memory availability.
3. Selecting same option <encode / decode> asks for stopping even if actual encode / decode has stopped automatically after all input is consumed. This is a demo level issue.

4. Tear down of multi channel encode / decode <Deletion> doesn't seem to work reliably.
5. Number of encode / decode instance creation is dependent on memory availability for algorithm & input / output buffers. With sufficient memory available, more encode / decode instances can be created.
6. Codec errors <arising due to memtab allocation failure or invalid parameter> not propagated properly.
7. AENC / ADEC APIs don't support G711 now – TBD.
8. Limited testing done on various codec parameter combinations.
9. Demo not enabled with 16ch video capture + 16ch audio capture in current release
    a. Based on use case's memory availability, this should be possible. Some memory tuning required.
    b. Capture data loss might happen – requires proper app thread scheduling.

5.10.5 Packages Required

1. rpe
2. c674x_aaclcdec_01_41_00_00_elf
3. c674x_aaclcenc_01_00_01_00_elf

5.10.6 ACAP APIs to work with AIC instead of TVP5158

Acap parameter enableTVP5158 is used to enable TVP5158 audio capture. captureDevice specifies the alsa device <TVP5158 assumed in release>.

For AIC capture, captureDevice should be modified to corresponding H/W & enableTVP5158 should be set to FALSE. Non TVP5158 based capture is not tested & only hooks are available. Customer might need to modify the audio demux routine based on capture mode / data format.