VISION SDK
Links Framework - Deep Dive

16 Mar 2015

Agenda

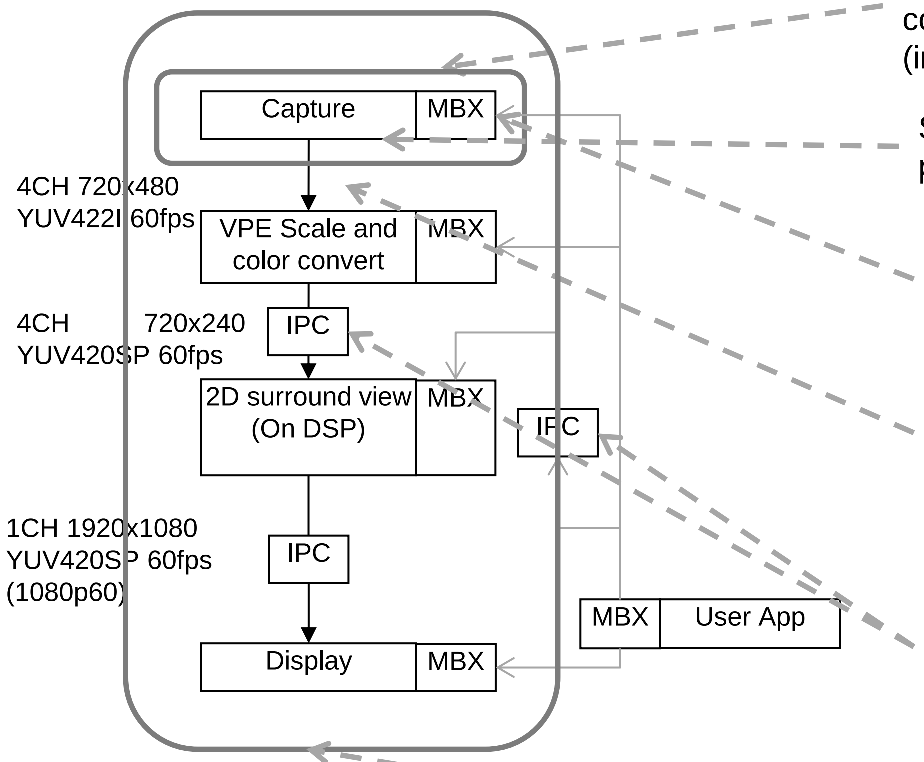
- Link Basics
  - Link State Diagram
  - Buffer exchange between links
  - Common Properties of a link
- Properties of each supported link
- More details
  - VIP Capture Link
  - VPE Link
  - Display Link
  - ISS Links Overview

Steps in working with Vision SDK

- Assuming a user wants to integrate an algorithm on DSP and/or EVE with whole system, they need to follow below steps
  1. Implement and test algorithm APIs in standalone manner on DSP/EVE, typically using CCS
  2. Design the whole use-case (on paper) involving the algorithm, input source/pre-processing, results drawing/visualization
  3. Identify additional links and/or algorithm plugins that need to be implemented to complete the use-case.
     1. In most cases only algorithm plugin's for the new algorithm would need to be written
     2. Other links can be re-used from what is already available in Vision SDK
  4. Implement a Algorithm plugin for this algorithm API using Algorithm Plugin interface
  5. (Optional) Modify use-case generation tool to add support for the newly written algorithm plugin
  6. Write use-case description text file and generate use-case .c/.h files using the use-case generation tool
  7. Write the missing parts of the use-case
  8. Compile the use-case, algorithm plugin and run on target SoC

“Links and Chains” framework

- VISION SDK is based on the “Links and Chains” framework.



A link is the basic processing step in a video data flow. A link consists of a OS 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.

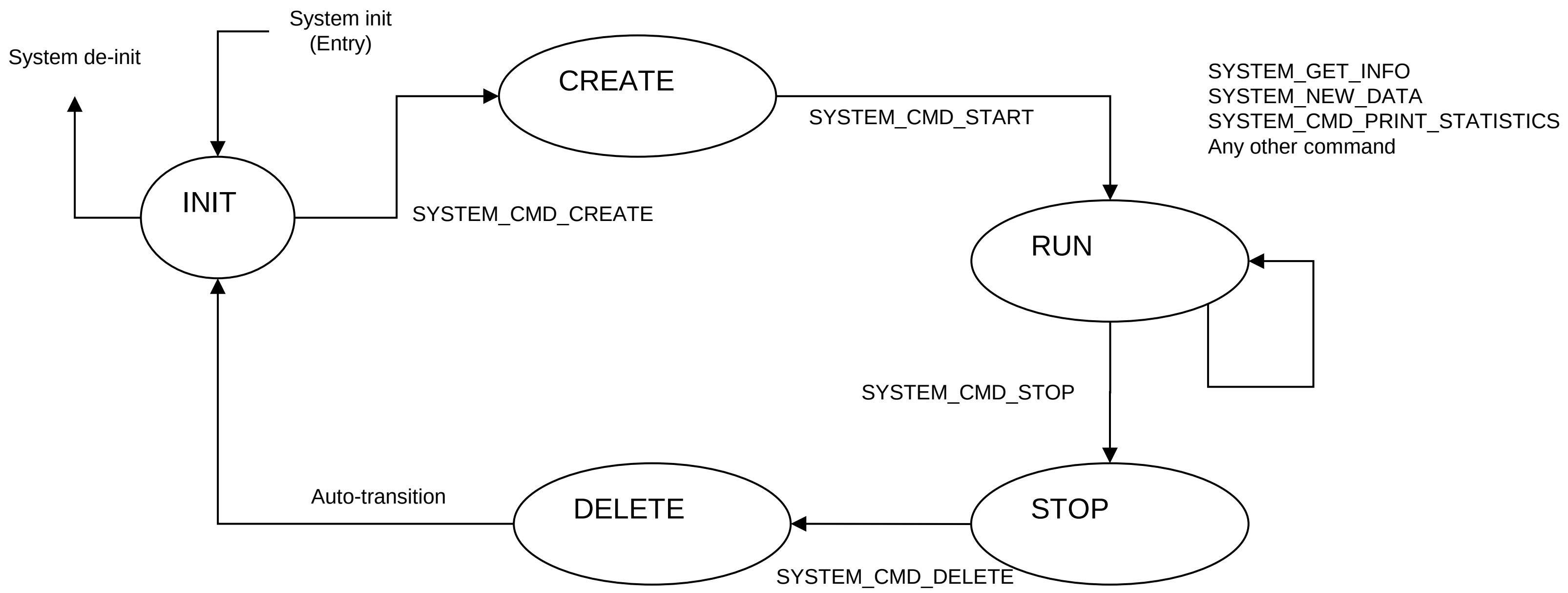
The message box associated with a link allows user application as well as other links to talk to that link.

The link implements a specific interface which allows other links to directly exchange video frames and/or bit streams with the link. i.e intervention of the user application is not needed on a frame to frame basis

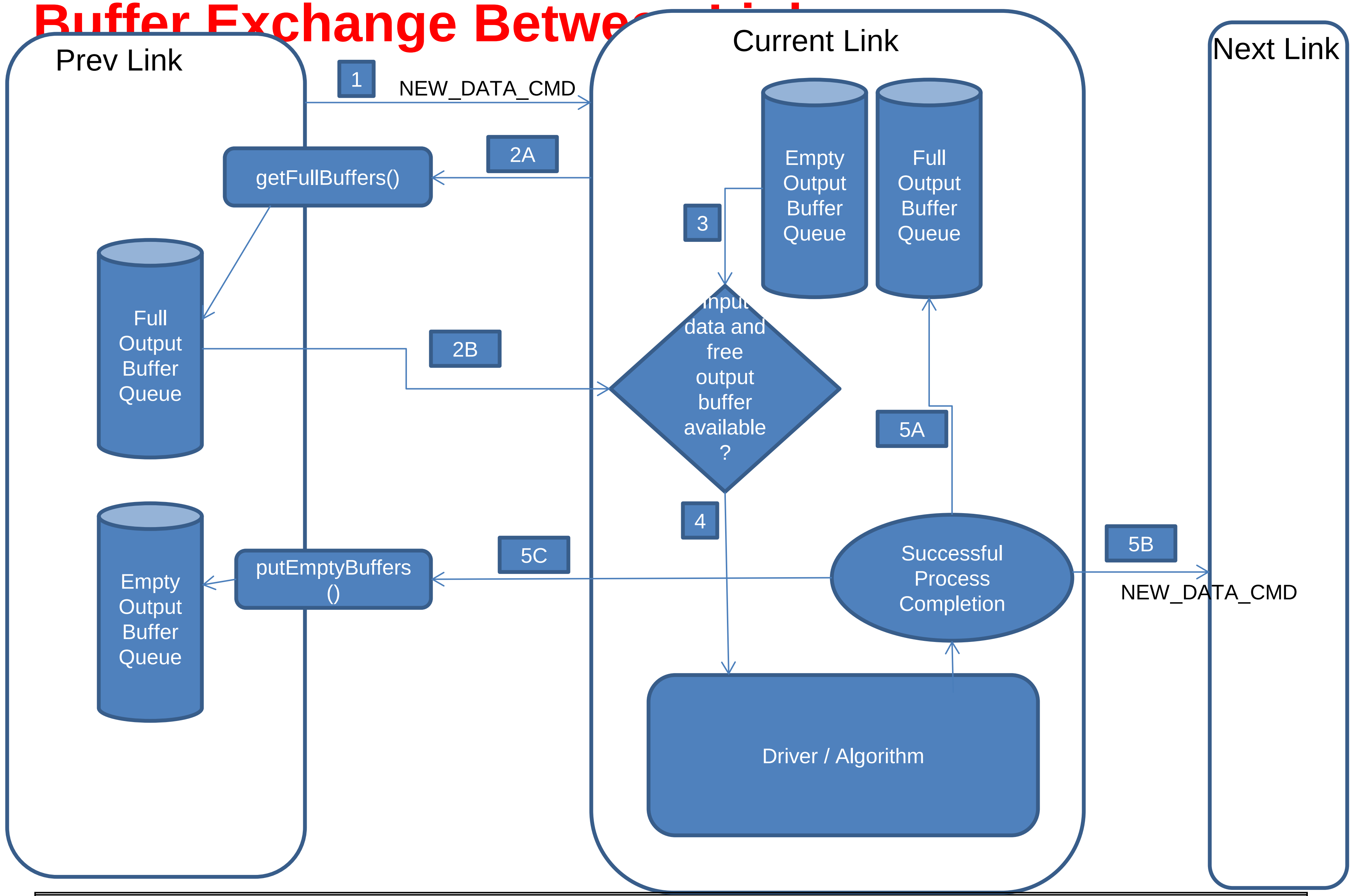
Link API allows user to create, control and connect the links. This control code is written on a single processor. Internally Link API uses IPC to actually control the links on different processors. User of Link API need not worry about the lower level inter processor communication details

A connection of links is called a chain. A chain is created on a processor designated as HOST CPU (IPU1-M4-0 in case of TDA2xx family of ADAS SoCs)

Typical State Diagram of a Link



Buffer Exchange Betwe[illegible]



Properties of a “Link”

- It has One or more input “connections” or Q’s
- It has One or more output “connections” or Q’s
- Each connection (input or output) holds one or more logical CHs of buffers
  - If a link has multiple input or output Q’s, each connection can have different number of CHs
    - Ex, a link can have one input and two output Q’s, it can have
      - 4 CHs in the input Q, 3CHs in output Q0, 1CH in output Q1
  - Channel number in a input or output Q always starts from 0, 1, … N-1
- Each CH is associated with a buffer of data. The buffer can be of different types as shown below
  1. Video Frame – holds YUV422I or YUV420SP or RAW or RGB data
  2. Bitstream – holds MJPEG or H264 compressed data
  3. Meta Data – holds any user defined data like histogram, disparity map etc
  4. Composite Video Buffer – holds pointers to a “group” of Video Frame or Meta data buffers

Connecting a “Link” to another “Link”

- Links are connected to each other by connecting the output Q of a “previous” link to the input Q of “next” link
- This connection is allowed under the below conditions
  1. The input Q of next link can accept the number of CHs output by the previous link
     - Ex, if next link can accept max 1 CH but previous link outputs 4CHs then the links cannot be connected to each other
  2. The buffer type for a given CH matches between input Q of next link and output Q of previous link
     - Ex, if Algorithm link outputs CH0 having buffer type as Meta data, it cannot be connected to display link which needs CHs with buffer type as Video frame
  3. The “previous” link and “next” link MUST be on the same CPU.
     - Only exception being IPU OUT and IPC IN link, which MUST be present on different CPUs

Information in a “Buffer” (1/2)

- The “buffer” which exchanges information between two links holds the below information
  - Information common to all buffer types (System_Buffer)
    - Buffer type – used to identify type of buffer “payload”
    - CH ID
    - Source Timestamp – timestamp value set at a “source” link like “Capture” or “IssCapture” or “NullSource” or “AvbRx” link.
      - Other links copy the “Source timestamp” from input buffer to output buffer
    - Payload – pointer to additional buffer information depending on buffer type
  - Information in Video Frame buffer (System_VideoFrameBuffer)
    - Buffer Address – address of pixel data associated with this video frame
    - Meta buffer address – optional meta data associated with this video frame
    - CH info – information like width, height, pitch, pixel format etc
  - Information in Meta data buffer (System_MetaDataBuffer)
    - Meta buffer address – address of memory holding actual meta data like histogram, disparity map, flow vectors etc

Information in a “Buffer” (2/2)

- Information in Bitstream buffer (System_BitstreamBuffer)
  - Buffer Address – address of compressed data associated with this bitstream frame
  - Meta buffer address – optional meta data associated with this bitstream frame
  - CH info – information like width, height, pitch, compression format etc
- Information in Video composite buffer (System_VideoFrameCompositeBuffer)
  - Number of “grouped” Channels (CH0..CHn-1)
  - Buffer Address [N] – address of pixel data associated with this video frame
    - One address for every CH which is grouped
  - Meta buffer address [N] – optional meta data associated with this video frame
    - One address for every CH which is grouped
  - CH info – information like width, height, pitch, pixel format etc – only for CH0

Link Properties - Details

VIP Capture link properties

Name	VIP Capture (captureLink.h)
Description	• Used to capture video frames from HW VIP port and write to memory • Same link instance can be used to capture from multiple HW VIP ports • Optional scaling, CHR_DS, CSC can be done before writing to memory • Sets “Source Timestamp”
# of Input Q’s	0
# of Output Q’s	1
Buffer Type	IN Qx: NA OUT Q0: Video Frame (YUV422I, YUV420SP, RGB888, Bayer RAW)
# CHs (Max)	12 (One CH from each HW VIP port)
Runs on	IPU1-0
# Instances	1

Display link properties

Name	**Display (displayLink.h)**
Description	• Used to display video frames from memory via HW PIPE in DSS • One link instance is used per HW PIPE • Optional scaling (YUV data formats), positioning can be done
# of Input Q's	1
# of Output Q's	0
Buffer Type	IN Q0: Video Frame (YUV422I, YUV420SP, RGB888, RGB565, ARGB4444) OUT Qx: NA
# CHs (Max)	16 (However only one CH, CH0 by default, is used for display, rest of the channels are ignored)
Runs on	IPU1-0
# Instances	4 (in TDA2x, limited by # of DSS HW PIPEs) 3 (in TDA3x, limited by # of DSS HW PIPEs)

VPE link properties

Name	**VPE (vpeLink.h)**
Description	• Used to scale (up/down) frames from memory to memory via HW VPE in TDA2x and ISS Resizer in TDA3x • Optional CHR_DS (YUV422I to YUV420), CSC (TDA2x ONLY) can be done
# of Input Q's	1
# of Output Q's	1
Buffer Type	IN Q0: Video Frame (YUV422I, YUV420SP) OUT Q0: Video Frame (YUV422I, YUV420SP, RGB888)
# CHs (Max)	8
Runs on	IPU1-0
# Instances	4

ISS Capture link properties

Name	**ISS Capture (issCaptureLink.h) – ONLY in TDA3x**
Description	• Used to capture video frames (Bayer RAW) from ISS CAL HW (CSI2/Parallel) to memory • Sets "Source Timestamp"
# of Input Q's	0
# of Output Q's	1
Buffer Type	IN Qx: NA OUT Q0: Video Frame (Bayer RAW)
# CHs (Max)	1 (Vision SDK v2.6) 4 (Vision SDK v2.7) – CSI2 Capture using virtual CHs
Runs on	IPU1-0
# Instances	1

ISS M2M ISP link properties

Name	**ISS M2M ISP (issM2mIspLink.h) – ONLY in TDA3x**
Description	• Used to process Bayer RAW frames from memory and output YUV frames and 2A statistics data to memory
# of Input Q's	1
# of Output Q's	3 (Q0 = Rsz A output, Q1 = H3A output, Q2 = Rsz B output)
Buffer Type	IN Q0: Video Frame (Bayer RAW) OUT Q0: Video Frame (YUV422I, YUV420SP) – Rsz A output OUT Q1: Meta Data - H3A Statistics output OUT Q2: Video Frame (YUV422I, YUV420SP) – Rsz B output
# CHs (Max)	1 (Vision SDK v2.6) 4 (Vision SDK v2.7)
Runs on	IPU1-0
# Instances	1

ISS M2M SIMCOP link properties

Name	**ISS M2M SIMCOP (issM2mSimcopLink.h) – ONLY in TDA3x**
Description	• Used to perform LDC (Lens distortion correction) and/or VTNF (Video temporal noise fitler) on video frames in memory to memory
# of Input Q's	1
# of Output Q's	1
Buffer Type	IN Q0: Video Frame (YUV420SP) OUT Q0: Video Frame (YUV420SP)
# CHs (Max)	4
Runs on	IPU1-0
# Instances	1

DUP link properties

Name	**DUP (dupLink.h)**
Description	• Used to duplicate "buffer information" from input Q into multiple output Q's • Only duplicates "information", pixel data is not copied • Purpose is to send same input buffer to multiple consumers
# of Input Q's	1
# of Output Q's	2 to 6
Buffer Type	IN Q0: ANY OUT Qx: ANY
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	5 (per CPU)

MERGE link properties

Name	**Merge (mergeLink.h)**
Description	• Used to merge CHs from multiple input Q's into a single output Q • Does not "sync" or "group" multiple CHs, only "forwards" a individual buffer after manipulating CH ID • Purpose is to "merge" input buffers from multiple producers and "forward" as a single "connection" or Q to next "consumer"
# of Input Q's	6
# of Output Q's	1
Buffer Type	IN Qx: ANY OUT Q0: ANY
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	5 (per CPU)

SELECT link properties

Name	Select (selectLink.h)
Description	• Used to choose a incoming CH from input Q and forward it to one of many output Q's • Does not "DUP" the channel. A input CH can goto only a single output Q • Purpose is to "select" and "forward" CHs onto different output Q's, ex, send CH0 camera output to Algorithm A and CH1 camera output to Algorithm B
# of Input Q's	1
# of Output Q's	4
Buffer Type	IN Q0: ANY OUT Qx: ANY
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	2 (per CPU)

IPC OUT link properties

Name	IPC Out (ipcLink.h)
Description	• Used to send buffers from one CPU to IPC IN link on another CPU • “next” link ID MUST always be a IPC IN link on a different CPU
# of Input Q’s	1
# of Output Q’s	1
Buffer Type	IN Q0: ANY OUT Q0: ANY
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	8 (per CPU)

IPC IN link properties

Name	IPC IN (ipcLink.h)
Description	• Used to receive buffers from IPC OUT link present on different CPU • "previous" link ID MUST be a IPC OUT link on a different CPU
# of Input Q's	1
# of Output Q's	1
Buffer Type	IN Q0: ANY OUT Q0: ANY
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	8 (per CPU)

SYNC link properties

Name	**Sync (syncLink.h)**
Description	• Used to “group” buffers from multiple CHs into a single buffer (Video Composite buffer) • CHs are grouped based on below conditions to make a Video Composite buffer, which is sent to “next” link 1. One frame from each CH is available 2. “Source” timestamp of each CH is within a user specified threshold (syncDelta) • If above conditions are not satisfied then the incoming frame is not sent ahead and released back to “previous” link • Purpose is to group frames from multiple sources into a single “logical” buffer based on their capture timestamps
# of Input Q’s	1
# of Output Q’s	1
Buffer Type	IN Q0: Video Frame or Meta Data OUT Q0: Video Composite Buffer
# CHs (Max)	8
Runs on	IPU, DSP, A15, EVE
# Instances	4 (per CPU)

NULL link properties

Name	**Null (nullLink.h)**
Description	• Used to get buffers from "previous" link and do one of the below • Simply release them (Null operation) • Write to file (Bitstream buffer) • Copy to memory (Video frame or Bitstream) • Send to PC over ethernet (Video frame or bitstream or Meta Data)
# of Input Q's	4
# of Output Q's	0
Buffer Type	IN Qx: ANY OUT Qx: NA
# CHs (Max)	16
Runs on	IPU, DSP, A15, EVE
# Instances	2 (per CPU)

NULL SOURCE link properties

Name	**Null Source (nullSourceLink.h)**
Description	• Uses to send buffers to “next” link. • The buffers are sent using one of the following as data source 1. Read from memory (pre-filled data) (Video Frame) 2. Read from file (Video Frame, Bitstream) 3. Receive from network (Video Frame, Bitstream) • Sets “Source Timestamp”
# of Input Q’s	0
# of Output Q’s	1
Buffer Type	IN Qx: NA OUT Q0: Video Frame or Bitstream
# CHs (Max)	16
Runs on	IPU, A15
# Instances	1 (per CPU)

Video Encode link properties

Name	**Encode (encLink.h)**
Description	• Used to encode Video Frames into Bitstream using MJPEG or H264 compression
# of Input Q's	1
# of Output Q's	1
Buffer Type	IN Q0: Video Frame (YUV420SP) OUT Q0: Bistream (MJPEG or H264)
# CHs (Max)	8
Runs on	IPU1-0
# Instances	1

Video Decode link properties

Name	**Decode (decLink.h)**
Description	• Used to decompress Bitstream buffers into Video Frame buffers
# of Input Q's	1
# of Output Q's	1
Buffer Type	IN Q0: Bitstream (MJPEG or H264) OUT Q0: Video Frame (YUV420SP)
# CHs (Max)	8
Runs on	IPU1-0
# Instances	1

AVB RX link properties

Name	**AVB Receive (avbRxLink.h)**
Description	• Used to receive Bitstream (MJPEG) over ethernet using AVB and send it to “next” link • Sets “Source Timestamp”
# of Input Q’s	0
# of Output Q’s	1
Buffer Type	IN Qx: NA OUT Q0: Bitstream (MJPEG)
# CHs (Max)	5
Runs on	IPU1-1
# Instances	1

Display Controller link properties

Name	Display Controller (displayCtrlLink.h)
Description	• This is a control link which does not have any input or output Q’s • Used to control DSS properties like, VENC config, HW pipe ordering on display panel, HW pipe to VENC mapping, background color etc
# of Input Q’s	0
# of Output Q’s	0
Buffer Type	IN Qx: NA OUT Qx: NA
# CHs (Max)	NA
Runs on	IPU1-0
# Instances	1

“Processor” link properties

Name	**System Link (systemLink_<CPU>.h)**
Description	• This is a control link which does not have any input or output Q’s • Used for general purpose or miscellaneous control of different CPUs from application, Ex, CPU load statistics, memory heap usage statistics
# of Input Q’s	0
# of Output Q’s	0
Buffer Type	IN Qx: NA OUT Qx: NA
# CHs (Max)	NA
Runs on	IPU, DSP, A15, EVE
# Instances	1 (per CPU)

Algorithm link properties

Name	**Algorithm link (algorithmLink.h)**
Description	• Connects to 1 or more input Q's, process buffers from input Q's and output's results over one or more output Q's • A "AlgorithmPlugin" is associated with a Algorithm link. "Algorithm" plugin run the actual algorithm • Algorithm link is the common code/implementation. "Algorithm plugin" is the algorithm specific implementation to complete the Algorithm link
# of Input Q's	Specified by algorithm plugin associated with the Algorithm link
# of Output Q's	Specified by algorithm plugin associated with the Algorithm link
Buffer Type	Specified by algorithm plugin associated with the Algorithm link
# CHs (Max)	Specified by algorithm plugin associated with the Algorithm link
Runs on	IPU, DSP, A15, EVE
# Instances	8 (per CPU)

AlgPlugin: ISS AEWB properties

Name	**ISS AEWB (algorithmLink_issAewb.h)**
Description	• Used to take H3A statistics Meta Data buffer from ISS M2M ISP as input • Send results as control commands to 1. ISS M2M ISP for Auto-expsure, Auto-white balance (AE/AWB) 2. External Sensor for Auto-exposure (AE)
# of Input Q's	1
# of Output Q's	0
Buffer Type	IN Q0: Meta Data (H3A Statistics from ISS M2M ISP) OUT Qx: NA
# CHs (Max)	4
Runs on	IPU1-0
# Instances	NA (equals # of Algorithm link instances)

Internal Implementation Details

VIP Capture Link

VIP Capture Link

Capture link operation can be divided into following states

- Initialization.
  - OS thread gets created and it's waiting for the commands from application
  - Link registers itself to the system link which is part and parent of the links and chains infrastructure.
- Create
  - This should be the first command for the link. Along with command link creation parameters are passed. Please refer to the API guide for the specific link create parameters.
  - As part of create command capture link creates the VIP capture driver and configures it based on the link create parameters.
  - Empty buffers are also primed to driver as a part of create command.
- Start/Run
  - As part of this, link is ready to receive data generated by hardware for capture (VIP)
  - Capture driver is started as a part of this command and link starts receiving the buffers.
  - Once the capture link is started it enters the steady running state.
  - Link is pending on a semaphore and is waiting for the message. This message is about waiting for new buffers available to link from the driver.

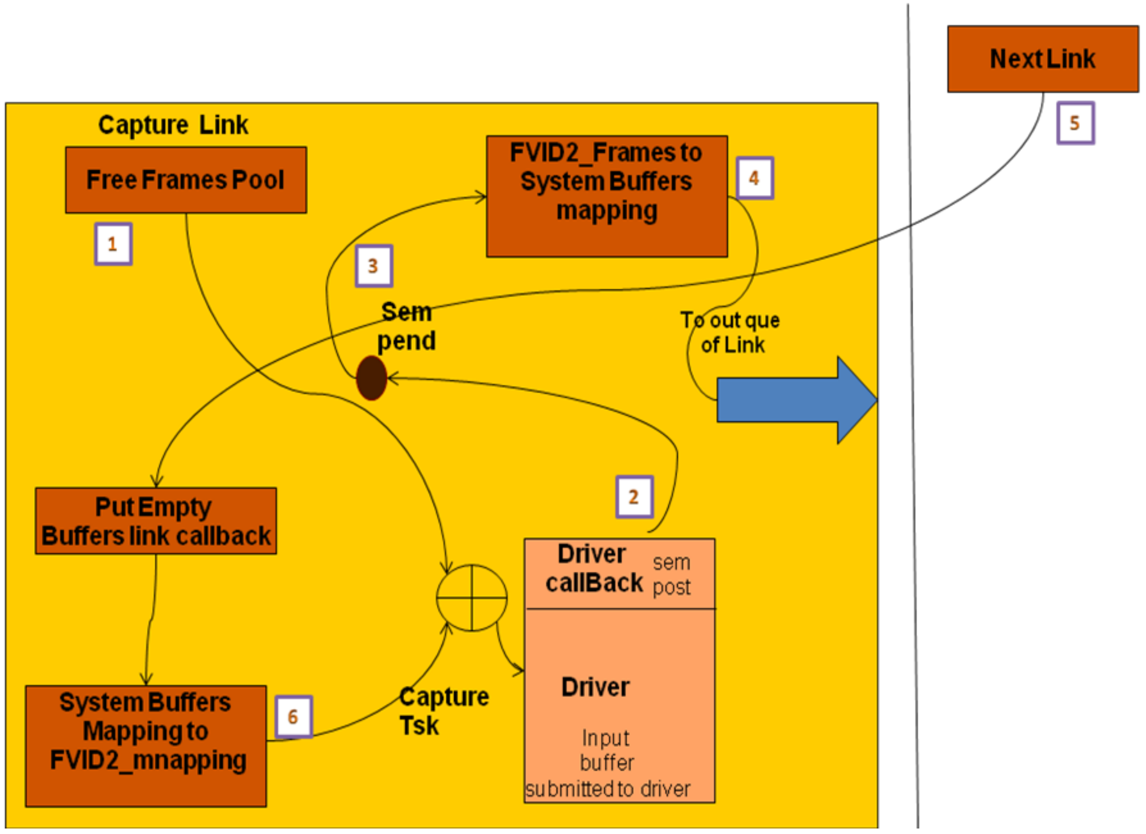
VIP Capture Link (Cont.)

  - Captured buffers are queued to the output queue of the link and next link is informed using the notify command.
  - Buffers consumed by the next link are returned back to capture link and these buffers queued back to driver for filling it up with new data.
  - Data flow in running state is explained in next section
  - Link also handles few run time commands during running state like printing of instrumentation and debug information. Please refer to API guide for more details lists of commands supported during running phase.

- Stop
  - Capture links stop receiving data after this command.
  - There is no data flow either from driver or between links once the link is stopped.
  - Capture driver is also stopped as a part of this command.
  - All the buffers queued to capture driver gets flushed as a part of this command.
- Delete
  - Capture link gets deleted as part of this command.
  - Driver also gets deleted.
- De-Initialization
  - Links gets de-initalized.
  - Task gets deleted and links de-registers itself from the system link.

VIP Capture Link (Cont.)

Dataflow Diagram: This shows buffer exchange between capture link and the capture driver and buffer exchange with next link.



Next Link
5
Capture Link
Free Frames Pool
1
FVID2_Frames to System Buffers mapping
4
3
Sem pend
To out que of Link
Put Empty Buffers link callback
2
Driver callBack
sem post
System Buffers Mapping to FVID2_mnapping
6
Capture Tsk
Driver
Input buffer submitted to driver

Capture Link (Cont.)

- Following are the important steps for the data flow diagram.
  1. Free frames get queued to the driver as a part of priming process. This is done as a create phase of the link.
  2. Once the link is started buffers get filled. Capture link callback function is called by the driver. Call back functions sends SYSTEM_CMD_NEW_DATA to the capture link. Capture link waiting on the semaphore wakes up and de-queue the FVID2 frames from the driver
  3. FVID2_Frames gets mapped onto system_buffer. System buffer is the entity which gets exchanged between the links.
  4. System buffers are queued to the capture output queue. After which notify command about the new data available in output queue is sent to next link connected to capture link.
  5. Next link consumes the filled system_buffers. Once next link is done with the capture buffer it calls the “linkPutEmptyBuffers” to give the consumed buffers back to the capture link.
  6. Capture link maps the system_buffer to FVID2_Frame to queue the consumed buffer back to capture driver for capturing the fresh data.
  7. Steps 2 to 6 gets repeated until the capture link is stopped.

VPE Link

VPE Link

VPE link operation can be divided into following states

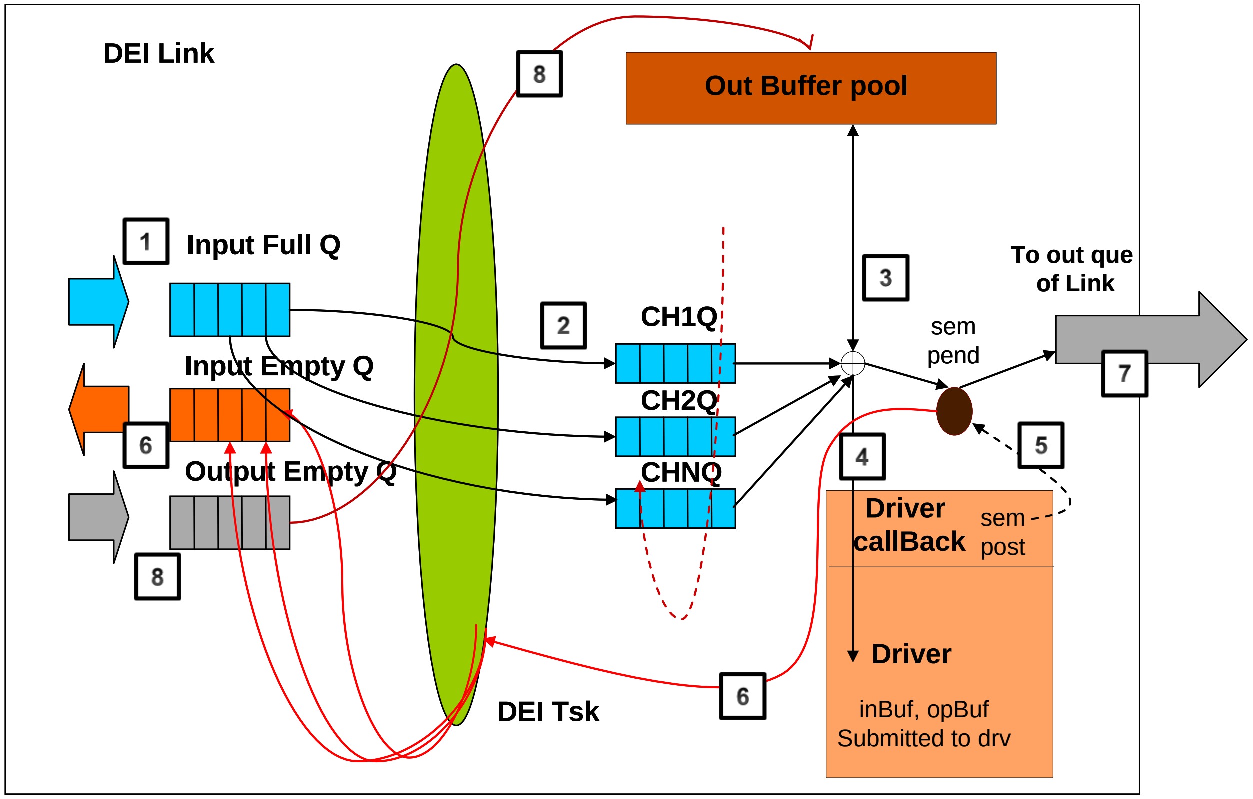
- Initialization.
  - Link thread gets created and it's waiting for the commands from
  - Links registers itself to the system link which is part and parent of the links and chains infrastructure.
- Create
  - This should be the first command for the link. Along with this command link creation parameters are also be passed. Please refer to the API guide for the specific and detailed link create parameters.
  - As part of create command VPE link creates the VPE driver and configures it based on the link create parameters.
  - VPE link allocates memory for output buffers. Output buffers are allocated per channel vise.
  - A link call back function is getting registered with the VPE driver during the driver creation

VPE Link (Cont.)

- Start/Run
  - On successful completion of “create” state, link is moved to the run state, where it is ready to receive and process the input video frames
  - Link also handles a few run time commands during running state like output resolution change, printing of instrumentation and debug information etc. Please refer to API guide for more detail list of commands supported during run state
- Stop
  - VPE links stop processing of video frames on this command.
  - There is no data flow either from driver or between links once the link is stopped.
  - VPE driver is also stopped as a part of this command.
  - All the buffers queued to driver gets flushed as a part of this command.
- Delete
  - VPE link gets deleted as part of this phase.
  - Driver also gets deleted.
- De-Initialization
  - Links gets de-initialized.
  - Task gets deleted and links de-registers itself from the system link.

VPE Link (Cont.)

Dataflow Diagram: This show how data flows from VPE link to next link and back to the VPE link after consuming the data



VPE Link (Cont.)

- Following are the important steps for the data flow diagram.
  - Step1: On receiving the SYSTEM_CMD_NEW_DATA event from previous link, VPE link reads the input buffers from the Input FullQ
  - Step2: Read all the available frames for all channels and put them into a link internal channel specific Q. For example all frames from channel 1 are kept in CH1Q and from channel 2 in CH2Q etc
  - Step3: As a next step, VPE link de-queue one output buffer for each and every input frame present in the intermediate channel specific Q.
  - Step4: If both input and output frames are available, link prepare the FVID2 process list (this is the format in which the job need to be submitted to the VPE driver). Once prepared the process list, it is submitted to the VPE driver by calling “FVID2_processFrames”
  - Step5: Driver will invoke a call back once the processing is completed. VPE link reclaim the buffers both input as well as output buffers by calling the driver function “FVID2_getProcessedFrames”
  - Step6: The input buffers are now freed by put them back to the input side emptyQ
  - Step7: The output buffers which are filled by VPE link is send out to the next link by putting them into output FullQ. VPE link also send a SYSTEM_CMD_NEW_DATA event to the next link to inform the data availability
  - Step8: Once the next link done with the VPE link output buffer, these buffers are put back to the VPE link output emptyQ. These buffers are now free to pick for the subsequent VPE link process
  - The above steps will be repeated in the RUN state

Display Link

Display Link

Display link states can be divided into following state diagrams.

- Initialization.
  - OS thread gets created and it’s waiting for the commands from
  - Links registers itself to the system link which is part and parent of the links and chains infrastructure.
- Create
  - This should be the first command for the link. Along with this command link creation parameters are also be passed. Please refer to the API guide for the specific and detailed link create parameters.
  - As part of create command display link creates the DSS driver and configures it based on the link create parameters.
  - Dummy blank buffers are also primed to driver as a part of create command.
- Start/Run
  - On start state link is ready to receive data generated by previous link.
  - DSS driver is started as a part of this command and link starts receiving the buffers.
  - Video buffers are getting queued in the input side Full queue of the display link from the previous link. Previous link also informed this frame availability by sending the SYSTEM_CMD_NEW_DATA notify command.

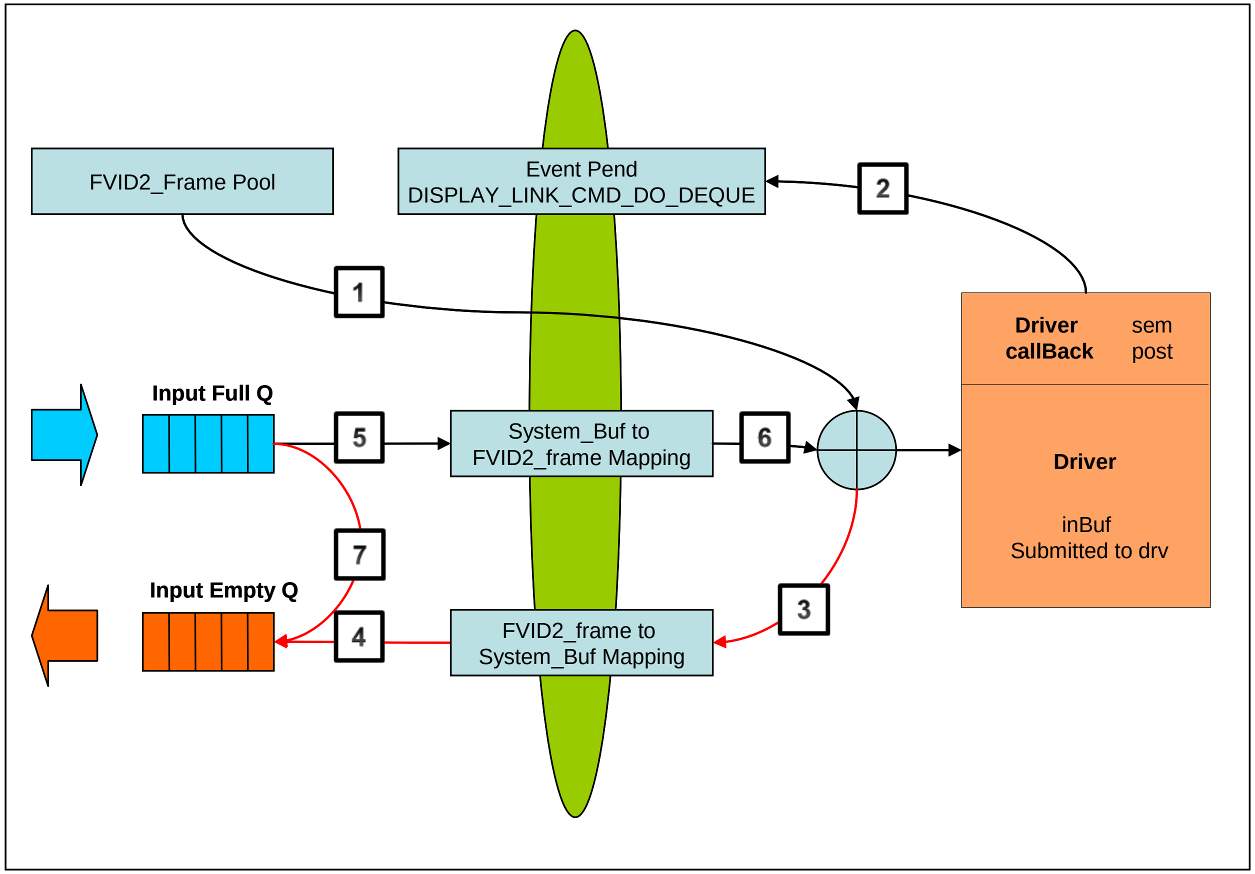
Display Link (Cont.)

  - Once the link is started it enters the steady running state. On steady state display driver gives periodic callbacks.
  - On callback display link dequeue the buffers which are already displayed from DSS driver. Also immediately queue the next set of full frames to the driver by queuing them into the DSS driver
  - Buffers consumed by the display link are returned back to the previous link by putting them into the input Empty queue.
  - Link also handles a few run time commands during running state like printing of instrumentation and debug information. Please refer to API guide for more details lists of commands supported during running phase.

- Stop
  - Display links stop displaying video frames after this command.
  - There is no data flow either from driver or between links once the link is stopped.
  - DSS driver is also stopped as a part of this command.
  - All the buffers queued to DSS driver gets flushed as a part of this command.
- Delete
  - Display link gets deleted as part of this phase.
  - Driver also gets deleted.
- De-Initialization
  - Links gets de-initialized.
  - Task gets deleted and links de-registers itself from the system link.

Display Link (Cont.)

Dataflow Diagram: This show how data flows from Display link to next link and back to the Display link after consuming the data


FVID2_Frame Pool
Event Pend DISPLAY_LINK_CMD_DO_DEQUE
2
1
Driver callBack
sem post
Input Full Q
5
System_Buf to FVID2_frame Mapping
6
Driver
inBuf Submitted to drv
7
Input Empty Q
4
FVID2_frame to System_Buf Mapping
3

Display Link (Cont.)

- Following are the important steps for the data flow diagram.
  - Step1: BSP Display driver need to be started before the real frames arrived at the input side. The driver can not be started without priming a few video frames. To do this Display link has a dummy blank frame and the same is used to prime and start the display. Once it is primed and started the DSS driver, Driver call back will be occurred. The CB interval will be same as the VENC display interrupts interval.
  - Step2: On the display driver Call Back, display link internally post a data event “DISPLAY_LINK_CMD_DO_DEQUE” to kick start the next frame processing
  - Step3: On DISPLAY_LINK_CMD_DO_DEQUE event, Display link first dequeue the Fvid2_frames which are already displayed from the DSS driver
  - Step4: These FVID frames need to be mapped to corresponding System buffer before it is placed in the input Empty Q (this is the output Q of previous link). This mapping is required because inter link frame exchange is done with generic API such as system buffers
  - Step5: On DISPLAY_LINK_CMD_DO_DEQUE event, Display link also dequeue the full frames (system frames) from the Input Full Q (this is the output Q of previous link)
  - Step6: DSS driver accept only FVID2 frames and hence the system buffers need to be mapped to FVID frames before it is queued to the driver
  - Step7: Any input system buffers which not part of the active channel (channel which is selected for display) are freed to the Input Empty Q immediately.
  - On a steady state step2 to 7 will be repeated

ISS Links

ISS Links

- There are several data flows possible in ISS subsystem.
- A single ISS link for the entire subsystem would be very complicated and the interface would be very huge.
- Also a given use case might not demand several of the data flow paths to be used.
- As a tradeoff, multiple links are being developed for ISS subsystem.
- Next few slides discuss different ISS Links

ISS Capture Link

- This Link will be used for capture of RAW data or YUV data from CSI2/Parallel sensors using ISS susbsystem
- In this link, no ISP processing will be done. Data captured from Sensor will be written to DDR.
- Following data flows will be covered by this link
  - CAL (CSI2) -> CAL (WR_DMA) -> DDR (raw)
  - LVDS-RX (Parallel) -> VMUX (BYS-IN) -> CAL (BYS-IN) -> CAL (WR_DMA) -> DDR (raw)
- Above data flows can be used for YUV sensors as well. However output data format in this case will be YUV422

ISS M2M ISP Link

- This Link will be used for reading input from memory (DDR), perform ISP processing and writing back to memory (DDR)

- Typical Single pass IPIPE processing with this link, will have below data flow
  - CAL (RD_DMA) -> CAL (VP) -> VMUX(ISP-IN) -> IPIPEIF (VPORT) IPIPEIF(ISIF-OUT) -> NSF3v -> ISIF -> GLBCE -> IPIPE -> RSZ -> CNF -> DDR (YUV)

- Dual pass WDR flow will be handled in this link as follows
  - WDR 1st pass
    - CAL (RD_DMA) -> CAL (VP) -> VMUX(ISP-IN) -> IPIPEIF (VPORT) -> IPIPEIF(DEC+SAT+ISIF-OUT) -> NSF3v -> ISIF -> DDR
  - WDR 2st pass
    - Input read: CAL (RD_DMA) -> CAL (VP) -> VMUX(ISP-IN) -> IPIPEIF (VPORT) -> IPIPEIF(DEC+SHIFT+ISIF-OUT) ->NSF3v -> ISIF (IPIPEIF-IN) -> IPIPEIF (ISIF-IN)
    - Previous frame read: IPIPEIF (RD) + IPIPEIF(ISIF-IN) -> IPIPEIF (DFS_WDR) -> IPIPEIF(IPIPE-OUT) -> GLBCE -> IPIPE -> RSZ -> CNF -> DDR (YUV)

ISS M2M SIMCOP (LDC/VTNF) Link

- This Link will be used for reading input from memory (DDR), perform LDC / VTNF and writing back to memory (DDR)
- This link will have data path as shown below –
  - CAL (RD_DMA) -> LDC / VTBNF -> DDR (YUV)

VPE link for ISS M2M Resize only operation

- This will be implemented using the existing VPE link and will not be a separate link
- This Link will be used for reading input from memory (DDR), perform Resizing and writing back to memory (DDR)
- This link will have data path as shown below –
  - CAL (RD_DMA) -> RSZ -> DDR (YUV)