VISION SDK
Software Architecture

Oct 2016

Agenda

- Vision SDK Overview
  - Vision SDK Goals and Features
  - Example ADAS data flows
- Components in Vision SDK
- Vision SDK – Architecture Overview
- Vision SDK – Architecture Details
  - Link API
  - Inter processor communication
  - Algorithm link

Vision SDK Overview

What is Vision SDK ?

- VISION SDK is multi processor software development platform for TI family of ADAS SoCs.
- The software framework allows users to create different ADAS application data flows involving video capture, video pre-processing, video analytics algorithms, and video display.
- The SDK has sample ADAS data flows which exercises different CPUs and HW accelerators in the ADAS SoC and shows customers how to effectively use different SoC sub-systems.
- VISION SDK is based on a framework named as the “Links and Chains” framework and user API to this framework is called “Link API”
- The SDK installer package includes all tools and components necessary to build applications, including code gen tools, BIOS, IPC, starterware, BSP drivers, networking stacks, codecs, algorithm kernels

Vision SDK - Goals

- Allows rapid prototyping of algorithms on DSP/EVE and creation of data flows in full system context.
- Provides consistent APIs for creating new and customizable system data flows
- Enables optimization and instrumentation for power, latency, performance, load
- Enables use of the software framework and components in customer systems

Vision SDK – Features

- Provides a framework and middleware which allows customer to run multiple algorithms on DSP/EVE/A15
  - Ex, 1CH VIP capture + DSP/EVE based Pedestrian detection + 4CH VIP/AVB capture + 2D Surround view using multiple DSP/EVE + Display – all running concurrently
  - Intent is to deliver a framework which runs multiple concurrent ADAS algorithms optimally (ex, least possible overheads from framework, low latency)
- Provides a framework and middleware which can be used by customers in their systems
  - high quality code (eg. consistent coding rules, etc.), best coding practices (ex, no dynamic memory allocation during run time, static analysis, MISRA C compliance), and structured development process (ex, DOORS for requirements, etc.)
  - Note: TI provides reference algorithms only.

Vision SDK – Features

- Provides buffer, DMA, data-flow and control flow management across different CPUs in the system
  - TI understands working with multiple CPUs and sharing resources across different CPUs is a challenge
  - The framework intends to give a view to individual algorithm developers as if they are working on a single processor and framework will take care of data-flow and control-flow across different processors
- Enables easy integration and benchmarking of custom algorithms
  - An algorithm plugin framework will allow customer to quickly integrate their own algorithms in context of full system
  - Instrumentation infrastructure from underlying components like BIOS is exported in a consistent way to allow easy analysis of algorithm/system execution
  - Customers can implement algorithms using VLIB on DSP, EVE LIB on EVE and then using the framework “connect” the algorithms so that
    - One algorithm executes as multiple stages with each stage running on different CPUs like DSP or EVE
    - Or multiple algorithms can run concurrently on different CPUs like DSP and EVE
    - Or a combination of above

Vision SDK – Features

- Provides a well defined, consistent API between different sub-systems
  - The intent is to allow clean separation of different sub-systems so as to allow changes in one sub-system to not affect other
    - Ex, change capture resolution without changing other stages in the pipeline i.e. display, VPE scaling, algorithms
  - Allow application to scale to different use-cases using existing sub-systems
    - Ex, change from AVB capture to VIP capture without changing rest of system
  - Allow customers to separate their custom code from TI delivered code (ex, board specific code, algorithm specific code) so as to allow easy integration of customer sub-systems within TI framework

- Allows flexibility in changing system level parameters without having to make changes all over the codebase
  - Examples,
    - Configure how many and what type of CPUs to use
    - Configure how to distribute an algorithm across different CPUs core
    - Configure how to increase or decrease system memory requirements

Vision SDK – Features

- Customer can have code/algorithms outside the SDK and only interface to the SDK modules which are useful and relevant for them
- Framework will NOT result in any API change in the components (BSP drivers, BIOS, IPC, DSP/EVE libs) that are available today on TDAxxx.
- Vision SDK scales to current and future ADAS SoCs from TI
  - Currently supported on TDA2xx, TDA3xx, TDA2Ex SoC's
- Vision SDK scales to different EVM/Boards
  - TDA2xx EVM with optional Multi-Deserializer board
  - TDA3xx EVM with optional Multi-Deserializer board
  - TDA2xx MonsterCam – Multi Sensor Fusion board
  - TDA2Ex EVM with optional Multi-Deserializer board
- Vision SDK supports Linux as OS on A15
  - Framework to connect Linux user-space application with DSP/EVE/M4 is included

Vision SDK – Features

- Utility tools are provided
  - to generate code for use-cases in order to ease development
  - to save and load test data via TCP/IP from PC side
  - to control application for calibration and tuning via TCP/IP (ISS DCC Image tuning, Stereo Calibration)
  - To flash and boot application from SD card, QSPI
- The framework has been previously used by many TI customers in production systems like DVR (Digital Video Recorder), NVR (Network Video Recorder) IPNetCam (IP Network Camera), Car Black Box (CBB), VC (Video Conferencing)

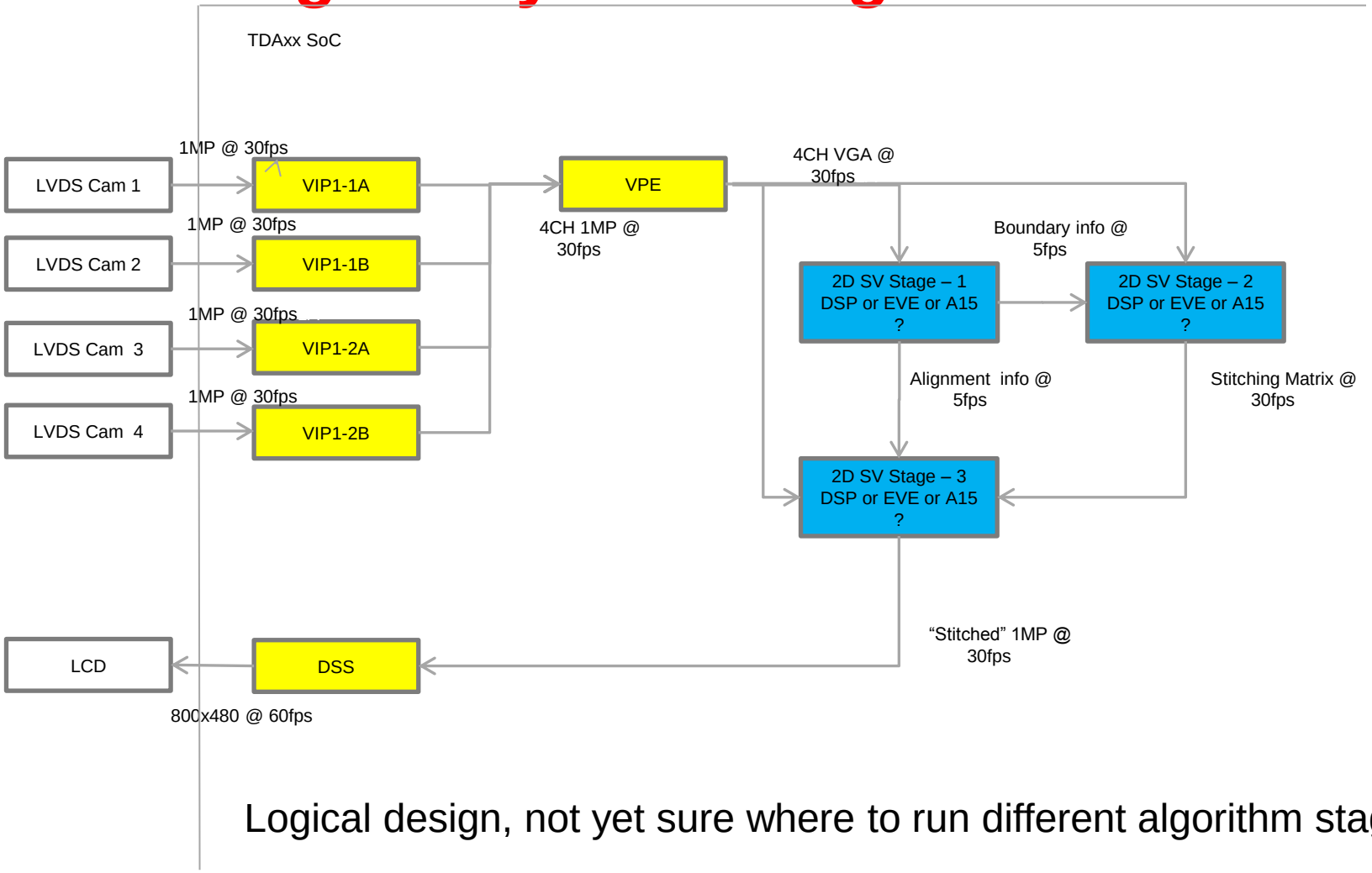
What is not covered by Vision SDK ?

- Framework does not provide automatic scheduling and execution of vision kernels to form an algorithm
  - i.e users have to write code to call the vision kernel APIs in the appropriate sequence to form the algorithm or algorithm stage on the given DSP and/or EVE
  - Any change needed in the sequence of kernels on the same core would need code change on that CPU core
  - Framework provides means to “connect” multiple algorithms stages across CPU cores with system elements like capture / scaling / display
- Framework does not provide a means of auto load balancing of DSP and EVEs
  - Framework provides instrumentation to find load and utilization of DSP/EVE/EDMA but user need to manually setup the distribution of algorithms so that CPUs are loaded in balanced way and performance requirements are met
- TI Algorithms included in Vision SDK like LDW, PD, TSR are reference only and would not be tested for all corner cases

What is not covered by Vision SDK ?

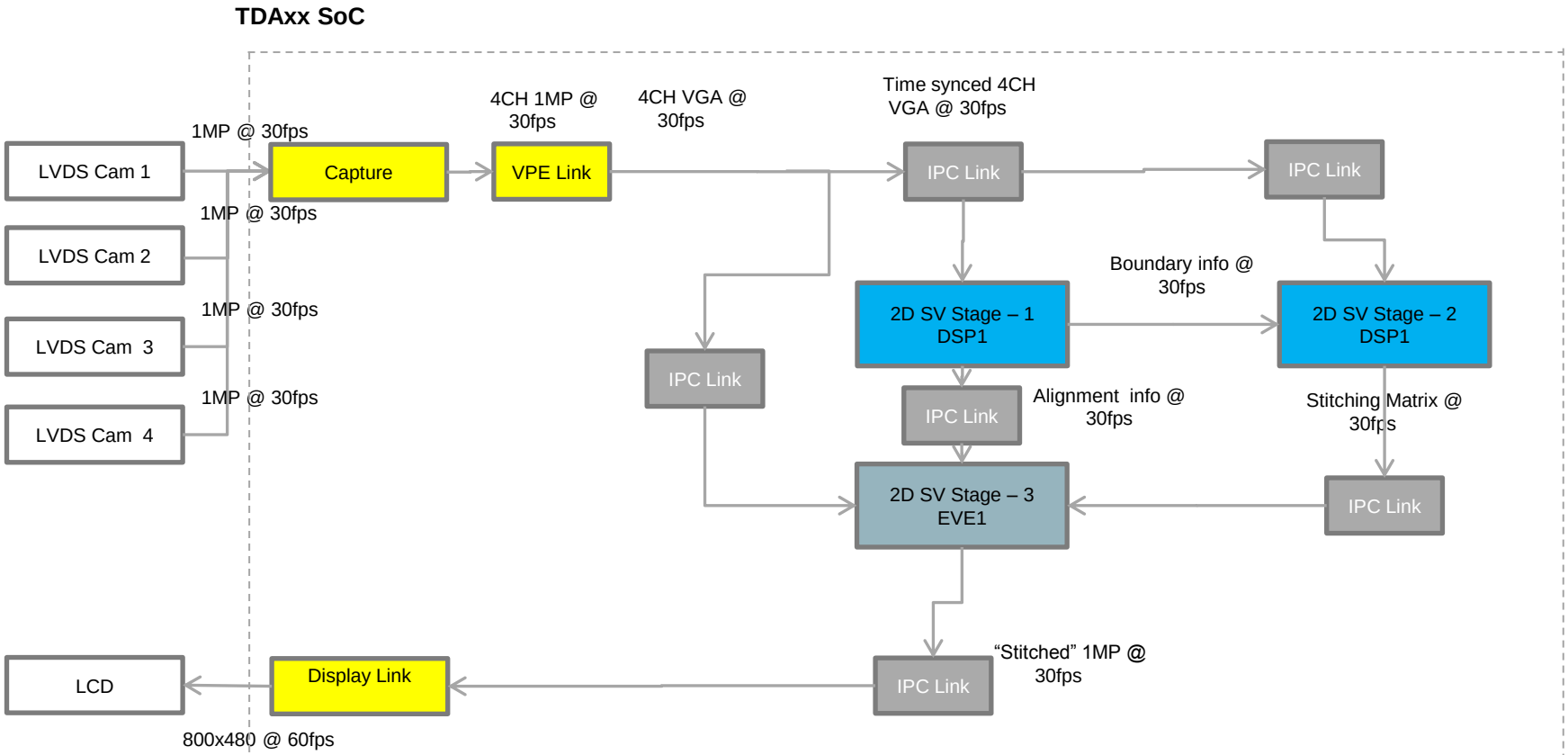
- Auto-SAR integration
  - Vision SDK will be implemented so as to have a M4 core free for Auto-SAR stack if needed, but Vision SDK itself will not include any Auto-SAR components
- OS independency – framework is based on BIOS for DSP/EVE/M4/A15. Linux is supported on A15
- Inter processor communication external to TDA2xx, ex, between TDA2xx and external uC

Example ADAS data flow – LVDS surround view – Logical system diagram



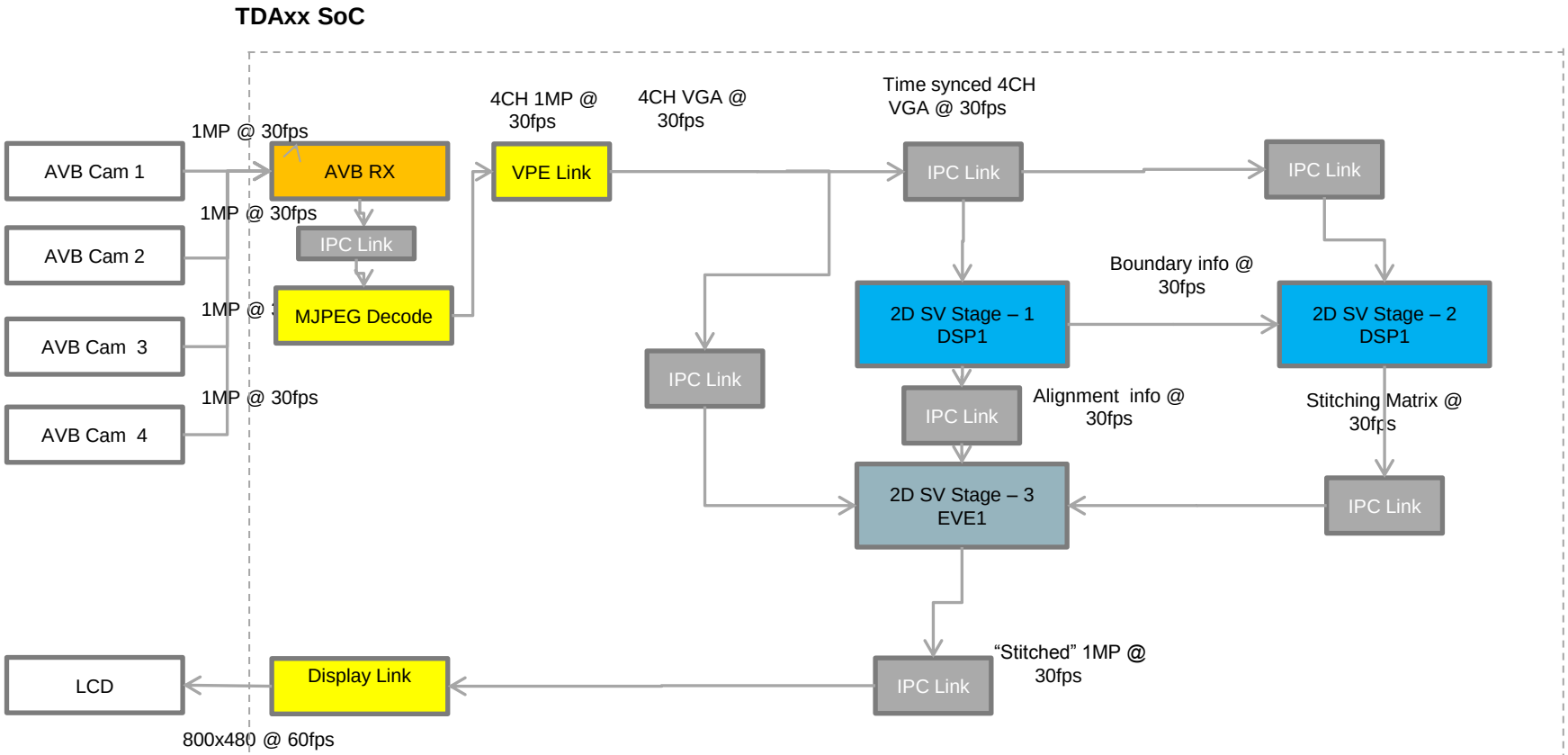
Logical design, not yet sure where to run different algorithm stages

Example ADAS data flow – LVDS surround view in Vision SDK



IPU1 Core0 | IPU1 Core1 | DSP1 | DSP2 | EVE1 | EVE2 | A15

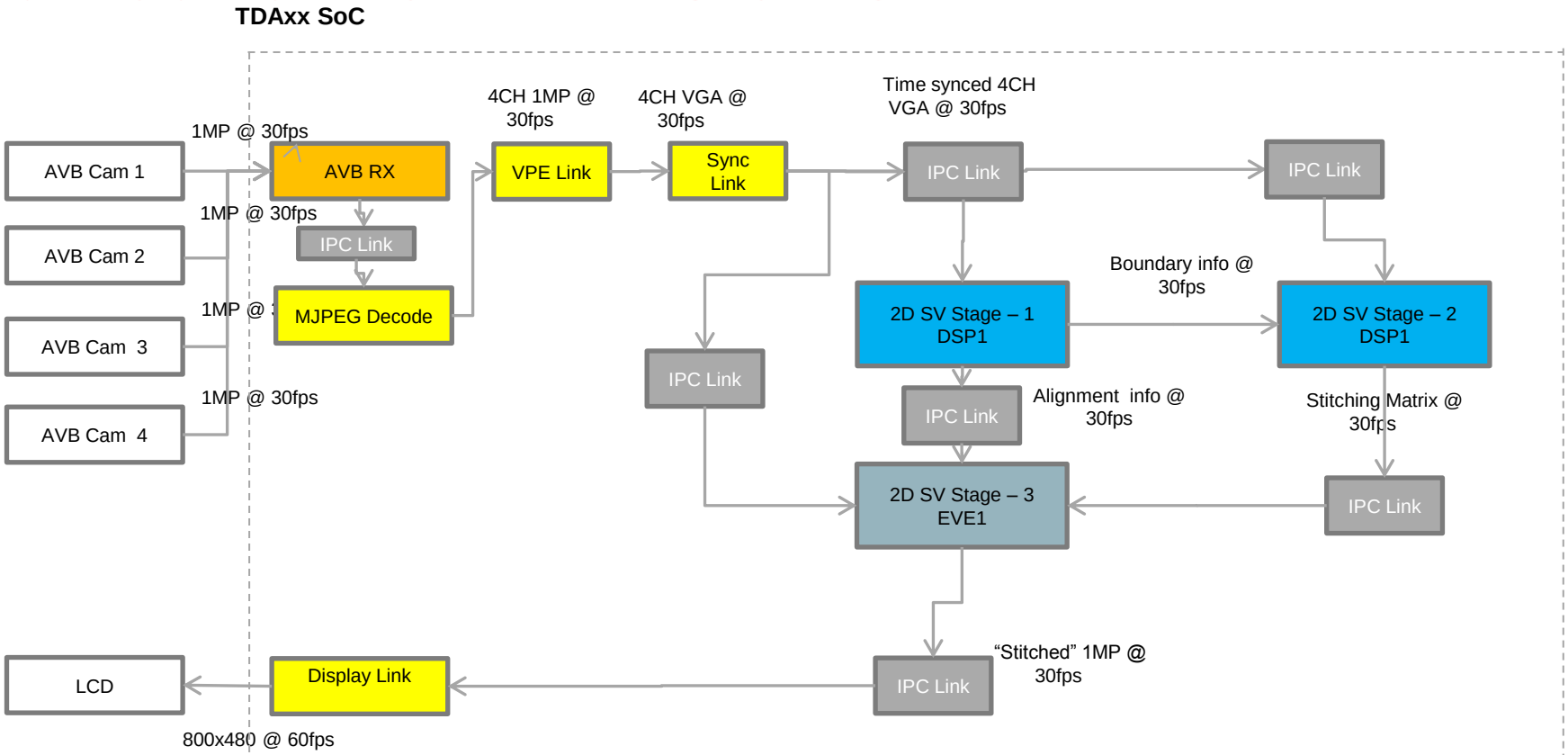
Example ADAS data flow – AVB Camera surround view in Vision SDK



Assume camera type changed from LVDS cameras to AVB camera's

IPU1 Core0 | IPU1 Core1 | DSP1 | DSP2 | EVE1 | EVE2 | A15

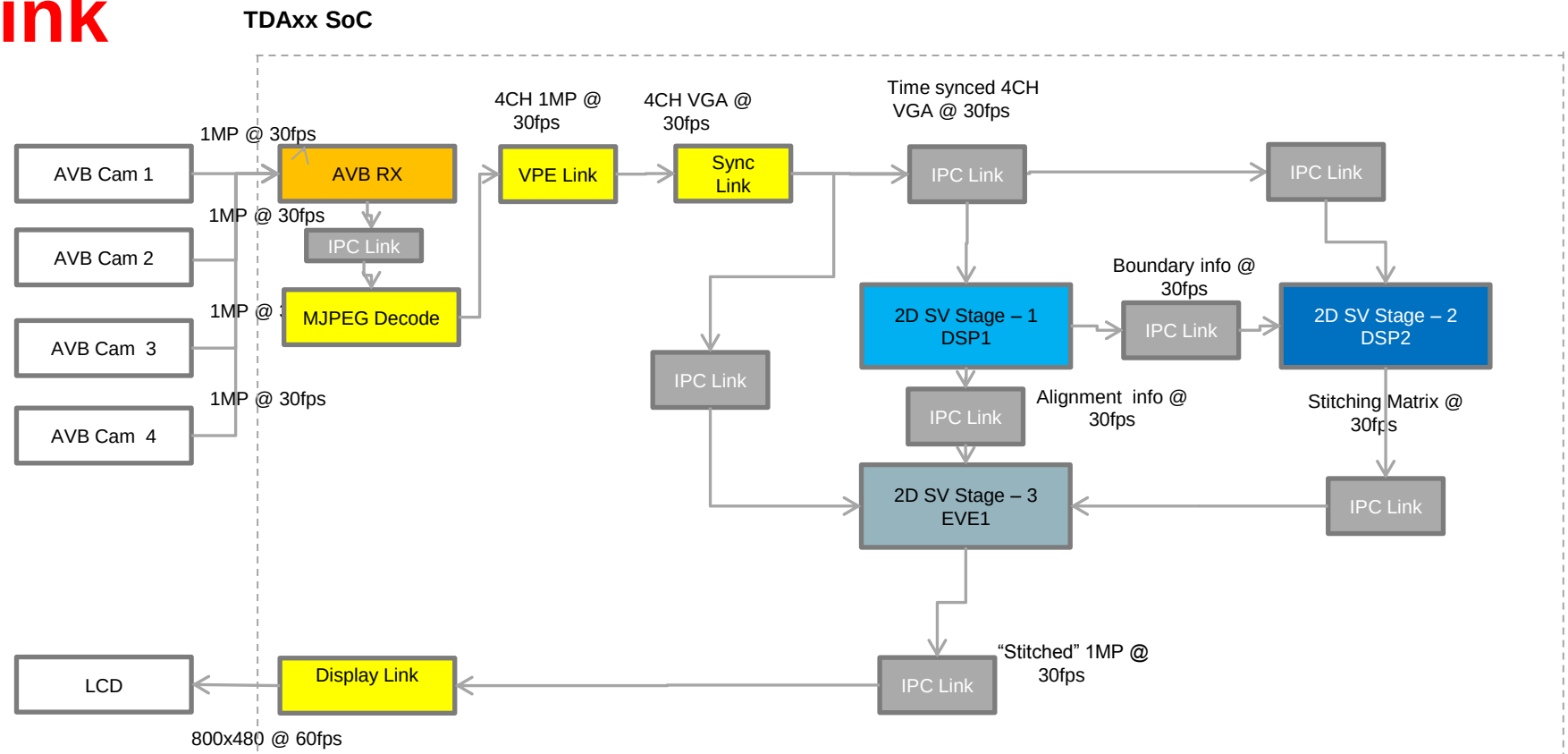
Example ADAS data flow – AVB Camera surround view in Vision SDK



Assume design flaw where we need to sync 4Ch channels based on timestamp

IPU1 Core0 | IPU1 Core1 | DSP1 | DSP2 | EVE1 | EVE2 | A15

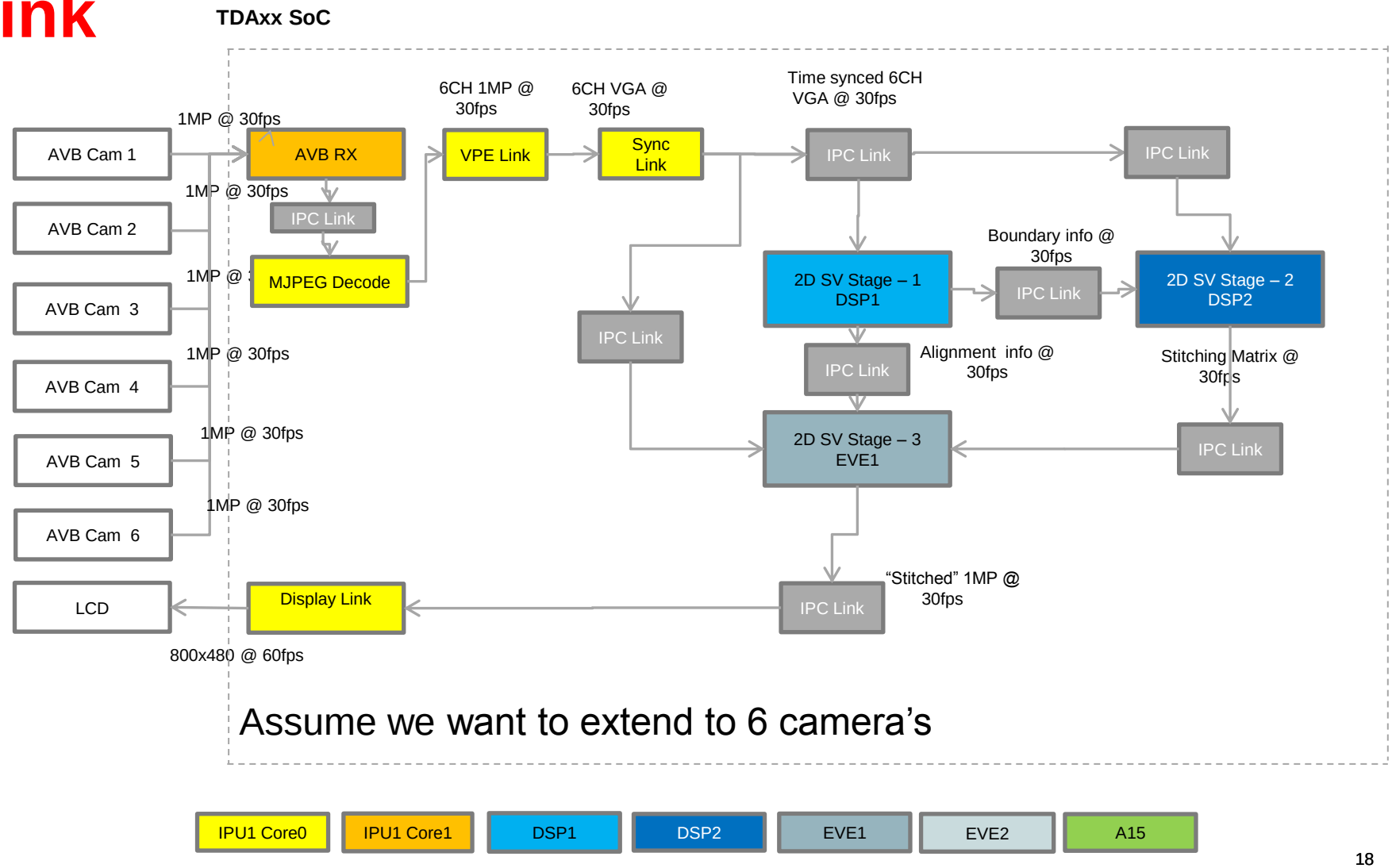
Example ADAS data flow – AVB Camera surround view in Vision SDK with "sync" link



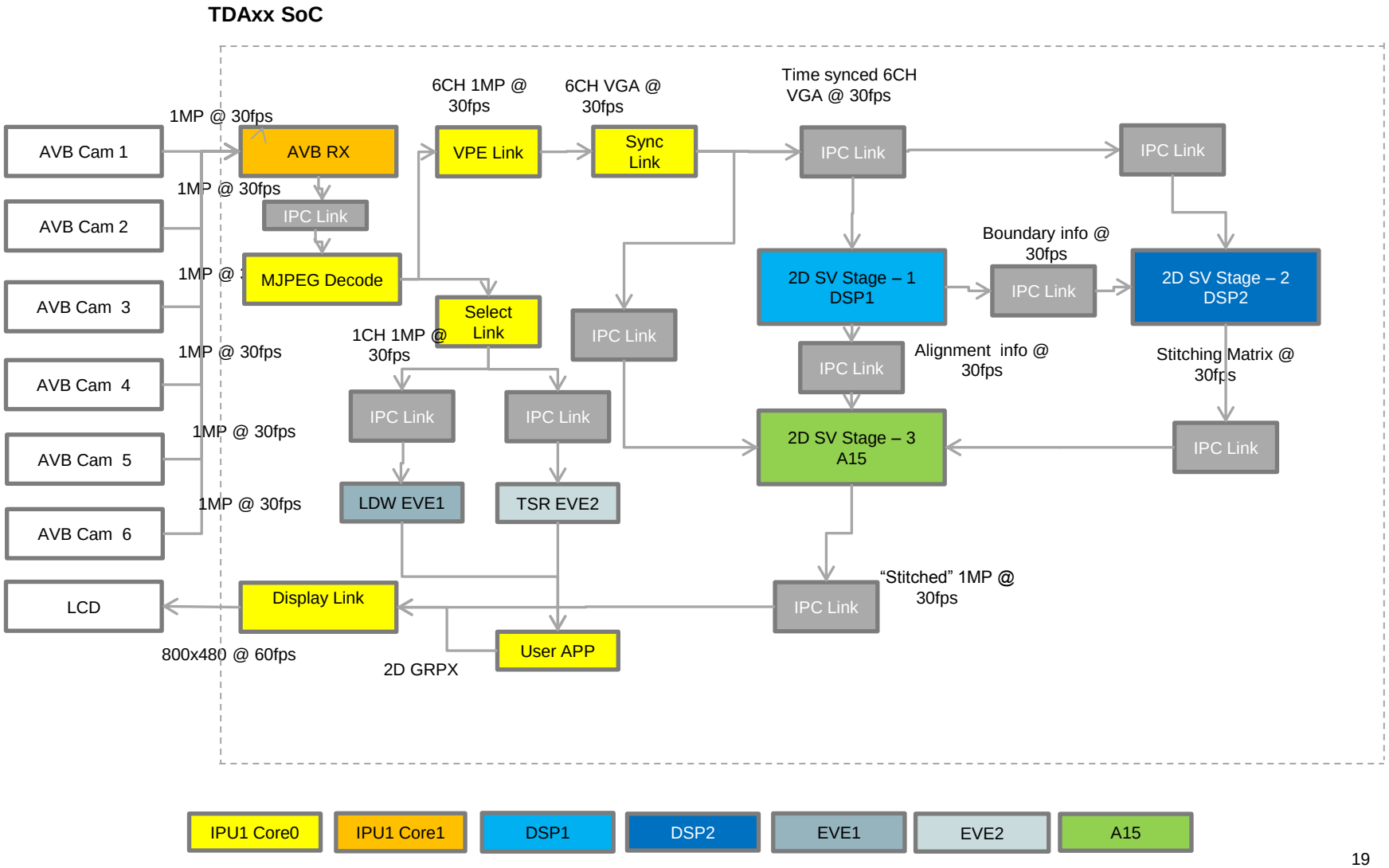
Assume we want more FPS performance on Stage – 1 and want to move Stage 2 to another DSP

IPU1 Core0 | IPU1 Core1 | DSP1 | DSP2 | EVE1 | EVE2 | A15

Example ADAS data flow – AVB Camera surround view in Vision SDK with "sync" link

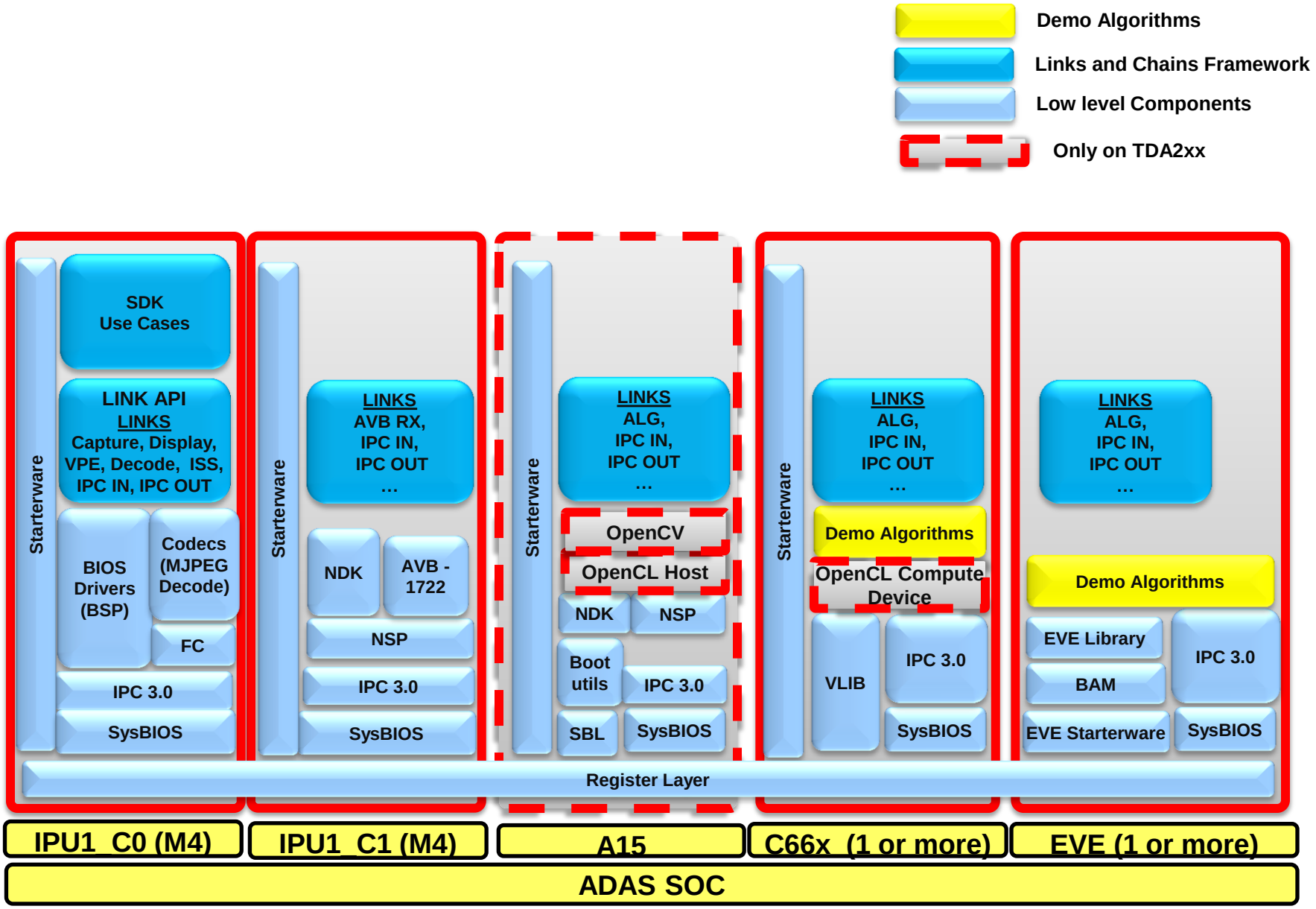


Example data flow – AVB Camera surround view with multiple algorithms in Vision SDK



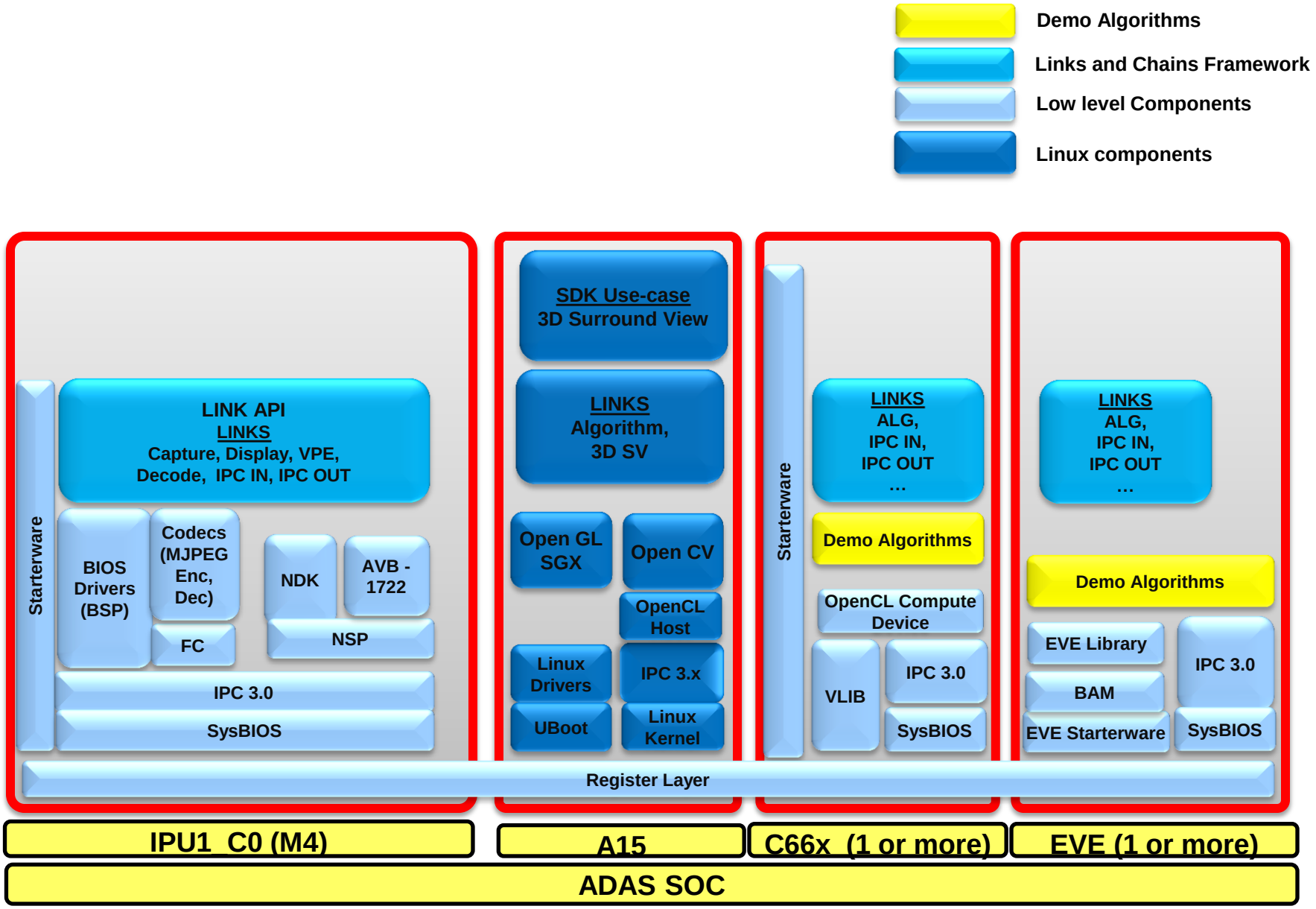
Components in Vision SDK

VISION SDK: SYS-BIOS SW Stack



21

VISION SDK: Linux + SYS-BIOS SW Stack



22

Components included Vision SDK (1/3)

SW Layer	Processor Applicable	Description	TI SW Package Name	Need XDC
BIOS	IPU1 M4-0 IPU1 M4-1 DSP EVE A15	BIOS RTOS is used as OS. Provides features like threads, semaphores, interrupts. Queues and message passing between links is implemented using BIOS semaphores. BIOS is used in non-SMP mode on all processors.	BIOS XDC (used for BIOS and other configuration)	Yes
IPC	IPU1 M4-0 IPU1 M4-1 DSP EVE A15	Software APIs used for communicating between processors. Provides features, multiprocessor heaps, multiprocessor linked list (ListMP), message queues, notify etc	IPC	Yes
Starterware / BSP Drivers	IPU1 M4-0	Video drivers like VIP capture, DSS display, ISS Capture, ISS processing, VPE scaling based on FVID2 interface to control and configure the VIP/VPE/DSS/ISS HW Serial drivers like I2C, SPI, UART Board specific drivers like sensor drivers	BSP Starterware	Yes (for serial drivers)

Components included Vision SDK (2/3)

SW Layer	Processor Applicable	Description	TI SW Package Name	Need XDC
EDMA Driver	IPU1 M4-0 IPU1 M4-1 DSP EVE A15	EDMA drivers	EDMA3 LLD	No
IVAHD Codecs	IPU1 M4-0	Video encode / decode APIs based on XDM / XDIAS interface. Uses framework components for resource allocation.	XDIAS Framework components IVAHD HDVICP2 API MJPEG, H264 decode MPJEG, H264 Encode	Yes (for framework components)
Networking Stack	IPU1 M4-0 or IPU1 M4-1 or A15	EMAC driver, TCP/IP stack, AVB stack for camera control and MJPEG bitstream receive over ethernet	NSP NDK AVB (only on IPU1-1)	Yes
Vision LIB	DSP	Vision algorithm kernels optimized for DSP.	VLIB	No
EVE LIB	EVE	Vision algorithm kernels optimized for EVE. Including framework for EVE algorithm execution	EVE Starterware EVE LIB EVE BAM framework	No

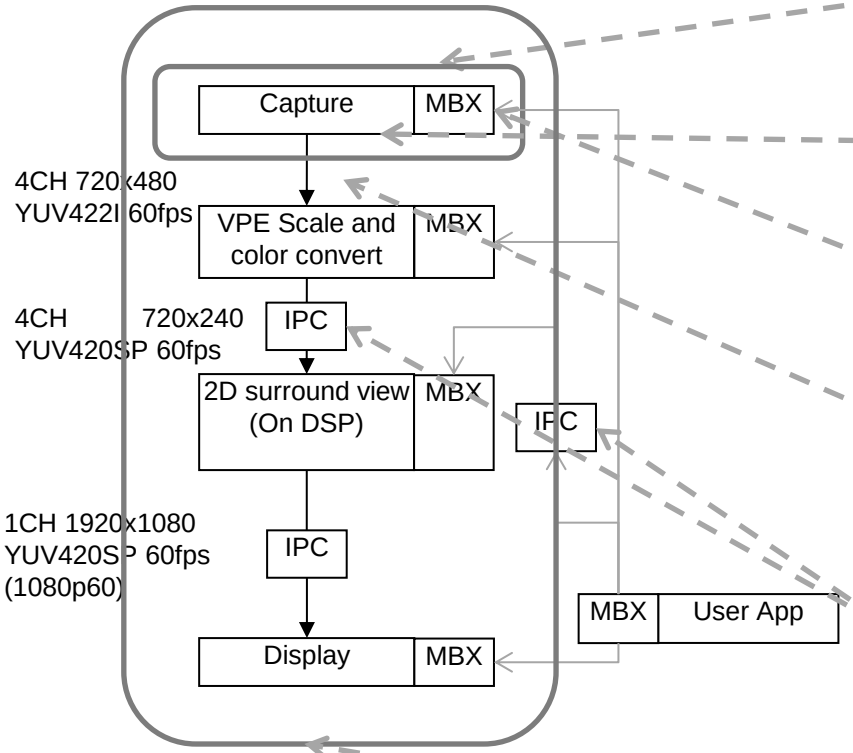
Components included Vision SDK (3/3)

SW Layer	Processor Applicable	Description	TI SW Package Name	Need XDC
Links	IPU1 M4-0 IPU1 M4-1 DSP EVE A15	Implementation of individual links. Some links are specific to a processor while some links are common across processors	VISION SDK	No
Link API	IPU1 M4-0	The link API allows users to create, connect, and control links on M4, DSP, EVE, A15. Link API is used to create a chain of links which forms a user defined use-case or data flow.	VISION SDK	No
User Application	IPU1 M4-0 A15 (Linux)	User application will use the link API and create its own custom chains. External device control like camera sensor, LCDs will be done by user application using serial IO drivers like I2C SPI User application can optionally also talk with IPU1-M4-1 (possibility doing car communication using AUTO-SAR stack)	Customer specific	Yes (for final linking only)

Vision SDK Architecture Overview

“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)

Advantages of link's (1/2)

- Once a link is implemented it can interface to any other supporting link including a link on another processor
  - Ex, once a algorithm is implemented as link, the input data to it can come from VIP capture link or VPE link (after scaling) or decode link (after MJPEG decode)
  - The algorithm link need not change
- Once a link is implemented it can be instantiated multiple times and on different processors (of same type)
  - Ex, once a algorithm is implemented on EVE, it can be instantiated on multiple EVEs to operate on multiple channels or on same channel but operate on different sub-frames/portions of the same input frame
- Once a link is implemented it can be instantiated multiple times and on different processors of different type provided the underlying link implementation uses same function API
  - Ex, if a algorithm writer provides same API to a algorithm on DSP as well as EVE, then the same algorithm link can be instantiated on DSP and/or EVE depending on user requirement

Advantages of link's (2/2)

- Each link needs to be written assuming it will exchange frames with a link on the same processor.
  - The IPC frames exchange is taken by special IPC links and is hidden form the link writer
- Once a chain is setup and started, frames "flow" through the whole system without user application intervention on a frame by frame basis.
  - User can intervene in between if required to control some aspect of the chain. Example, enable/disable a algorithm or dynamically modify algorithm "thresholds" etc

NOTE: Vision SDK provides a tool to generate source code for chains based on user description of the use-case. Refer "VisionSDK_UsecaseGen_Overview.pdf" for more details.

Summary of links implemented by TI (1/3)

Link Name	CPU applicable	Purpose
Video processing links		
VIP Capture Link	IPU1-M4-0	Capture video frames from VIP ports and output data to DDR memory
ISS Capture Link (TDA3x & TDA2Ex)	IPU1-M4-0	Capture frames from CSI2/Parallel sensors using ISS
ISS M2M ISP (TDA3x ONLY)	IPU1-M4-0	Process raw data from memory and convert to YUV
ISS M2M SIMCOP (TDA3x ONLY)	IPU1-M4-0	Process video data from memory and perform LDC, Temporal Noise filter
Display Link	IPU1-M4-0	Read video frames from DDR memory and display on configured display device
Display Controller Link	IPU1-M4-0	Configure display VENC and overlay manager related portion of the DSS
VPE Link	IPU1-M4-0	Read video frames from DDR memory, deinterlaced, scale, color convert and write to DDR memory
SW Mosaic Link	IPU1-M4-0 DSP	Read video frames from DDR for multiple channels and arrange them in memory in a user specified mosaic configuration using EDMA
Video Encode	IPU1-M4-0	Read video frames from DDR and encode them using different user specified compression schemes like H264, MJPEG
Video Decode	IPU1-M4-0	Read bitstreams from DDR and decode them using different user specified decompression schemes like H264, MJPEG

Summary of links implemented by TI (2/3)

Link Name	CPU applicable	Purpose
IPC Links		
IPC OUT	ALL	Get buffer information from previous link and sent to IPC IN link on another processor
IPC IN	ALL	Get buffer information from IPC OUT on another processor and sent to next link on same processor
Connector Links		
DUP	ALL	Get frame information from previous link and duplicate the information for 'N' next link's. DPU link keeps track of reference count before releasing the frames to previous link
Merge	ALL	Get frame information from 'N' previous link and make it appear like one input queue having multiple CHs to the next link. This allows multiple source to feed frames to the same consumer link
Select	ALL	Get frame information from previous link and segregate and route the information based on CH ID to 'N' next link's as specified by user.
Sync	ALL	Get multiple channels of input and synchronize them based on timestamp. This is required when running algorithms like 2D surround view
Null	ALL	Get frame information from previous link and do one of below - Do nothing – i.e NULL operation - Save to file - Transfer over network to PC using TCP/IP
NullSrc	ALL	Send frame to next link using one of below as content source - A PreDetermined frame in memory - Read from file - Receive frames from PC over network using TCP/IP

Summary of links implemented by TI (3/3)

Link Name	CPU applicable	Purpose
Networking Link		
AVB RX	IPU1-M4-1	Receive AVB packets from Ethernet interface, reassemble them into bitstream frames and output the bitstream frames to next link
Algorithm Link		
Alg Link	ALL	Get video frames from DDR and perform algorithm and output the results to the next link as meta data encapsulated in a system buffer. Algorithm are integrated into Vision SDK by implementing a "Algorithm Plugin" which is associated with a Algorithm Link

Use-case examples in Vision SDK

Usecases	TDA2xx	TDA3xx
1CH VIP capture + Display with optional Alg Frame Copy (DSP1 or EVE1 or A15)	√	√
1CH VIP capture + Dense Optical Flow (EVE) + HDMI Display	√	√
1CH VIP capture + Alg Subframe Copy (EVE) + Display	√	√
1Ch VIP capture + FrontCam Analytics PD+TSR+LD+SOF (DSP, EVE) + Display	√	√
5CH LVDS VIP Capture + Surround View (DSP) + PD+TSR (DSP, EVE) + HDMI Display	√	X
5CH AVB Capture + Decode + Surround View (DSP) + Edge Detect (EVE) + HDMI Display	√	X
1CH ISS capture + ISS ISP + ISS LDC+VTNF + Display	X	√
2CH VIP capture + Edge Detect (EVE1) + Dual Display	√	X
2Ch VIP capture + Stereo Disparity + HDMI Display	√	X

NOTE: Refer to VisionSDK_DataSheet.pdf for exact use-cases in a given release

Resource allocation – DDR Memory

- VISION SDK supports multiple memory map configurations (128MB/256MB/1GB),
  - It is possible to modify the memory map to use lesser or more memory as required.
  - It is possible to increase or reduce memory segment size as required via build time configuration
- Single EMIF 32-bit DDR is assumed.
  - It is possible to modify the memory map / DDR configuration to use dual 32-bit DDR or single 16-bit DDR
- DDR Memory allocation for buffers
  - All buffer memory for frame buffers/bitstream buffer/vision analytics algorithm buffers is allocated by the links at “create” time.
  - Once a link is in run phase no dynamic memory allocation is done.
  - Each link will allocate the buffer it needs for its processing output. A link will receive it input buffers from its previous link.
  - A Shared region in IPC, SR1, is used as a buffer heap. This heap will be a multi processor heap. i.e different processors can allocate memory from this heap using IPC APIs. IPC will take care of inter-processor mutual exclusion during memory allocation.

Resource allocation – EDMA

- TDAxx has multiple EDMA controllers
- System EDMA controller
  - The System EDMA controller is accessible to IPU1-M4-0, IPU1-M4-1, A15, DSP's via EDMA3 LLD APIs
  - A convenient means of configuring the channel allocation for each core is provided by the Vision SDK
  - EDMA3 LLD APIs will be used for System EDMA resource management
  - Utility APIs to copy / fill buffer will be provided by Vision SDK for system EDMA controller. Both interrupt / polling method will be supported
- DSP/EVE local EDMA controller
  - Each DSP and EVE has its own EDMA controller. This is dedicated for use of the respective DSP and EVE
  - EDAM3LLD can be used to access local EDMA controller
  - Algorithms typically have very specific needs from EDMA, customers are free to use this EDAM3LLD APIs or directly use the EDMA controller as they require
  - Vision SDK will give utility APIs to do resource management of the EDMA channels/PaRAMs

Resource allocation – Internal memory

- DSP and EVE both have internal memory which can be used by algorithms to store algorithm data
- In additional there is additional On-Chip OCMC memory which can be used by any of the CPU cores
- Vision SDK framework will provide utility APIs for algorithms to allocate memory from these internal memories
- Allocation from DSP/EVE local internal memory can be done only by the respective CPU
- Allocation from OCMC internal memory can be done by any core DSP/EVE/A15/M4
- OCMC Internal memory can be allocated in “persistent mode”, i.e data in memory block needs to preserved across algorithm buffer processing invocations and therefore the memory block must be dedicated for this algorithm
- DSP/EVE local internal memory is always allocated in “scratch” mode, i.e memory is shared between algorithms and data can be overwritten by another alogrithm across processing invocation.

Debug and instrumentation – Remote Log

- TI provides lots of debug and instrumentation support
  - via CCS tools for device related debug and instrumentation and
  - Via ROV tools for BIOS related debug and instrumentation
- Vision SDK will complement these tools by providing following debug logging via the UART port.
  - Using UART port allows debug information to be retrieved even when system is not connected to CCS via JTAG
- A remote logging feature will be provided which allows prints from all CPUs to be collected at the “host” CPU and then host CPU can print all the logs via UART. Each log will have a global timestamp to identify the time sequence of logs across processors
- A non-locking shared memory segment will be used so that exception logs can be made available even when normal IPC mechanisms like Notify, MessageQ have failed

Debug and instrumentation – Software Statistics

- Each link will additionally provide the following logs which will be useful to debug system issues like frame loss, FPS drop, buffer pipeline hangs, latency issues
  - Input / Output processing FPS
  - Input / Output dropped frame FPS
  - Driver Callback / interrupt count
  - Min/Max/Avg Source to current link Latency
- System level information like below will also be available
  - Per thread CPU load, HWI, SWI, total CPU load for each core
  - Memory heap usage of shared memory heaps, local heaps, internal memory heaps
- All this information will be available at host CPU via remote log feature

Working with custom HW board and custom applications

- The software is written such that board specific details are separate from framework related code
  - Example,
    - Capture link will provide parameters to user configure sensor specific parameters like Hsync/Vsync polarity
    - Sensor related I2C config will be kept outside of the framework code
  - This will allow examples to be ported to custom HW quickly
- Customer can write their own links and these links can reside outside of TI codebase. APIs are provided to register these links to the main framework
  - This allows customer to customize and write their own links without touching TI provided code and thus allow easier merging to bug fixes and updates from TI in the main framework codebase

Working with custom algorithms

- Customer can write their own algorithms and these algorithms can reside outside of Vision SDK codebase. APIs are provided to register these algorithms as “plugins” to the Algorithm link framework
  - This allows customer to write their own algorithms without touching TI provided code and thus allow easier merging to bug fixes and updates from TI in the main framework codebase

Vision SDK Architecture Details
Link API

Link API (1/2)

- These API's is used by the user to create, delete, connect, start, stop links in a chain
- Each link is identified by a system wide unique 32-bit link ID
- The link ID determines on which processor the link runs as shown below
- Each link API needs the link ID as an argument when sending a message to the link. This allows the framework to determine which thread on a given processor should receive the target link API command

Bits	Description
0..7	Link ID
8..11	Processor ID on which this link runs 0: IPU-1-M4-0 1: IPU-1-M4-1 2: A15 3: DSP-1 4: DSP-2 5: EVE-1 6: EVE-2 7: EVE-3 8: EVE-4

Link API (2/2)

API	Description
System_linkCreate	Creates a link - allocates driver, codec, memory resources. All link threads are created at init time. So this API will not create the link thread itself. This is the first API to be called for MOST links though for some links this API need not be called.
System_linkGetInfo	Get output information about a link like number of channels, properties of each channel. MUST be called after System_linkCreate() for a link. Usually used in the succeeding link in the chain to query the queue information of the previous link. A link output consists of • 'numQue' number of queues • Each queue consists of 'numCh' number of channels • Each channel has some properties like width, height, type etc This information is used by the next to configure itself based on input it will receive from the previous link or by the user to know about the link properties.
System_linkStart	Start the link - starts the driver or codec. This is optional API. Not all links would implement this API. Internally it would be implemented using System_linkControl API
System_linkStop	Stop the link - starts the driver or codec. This is optional API. Not all links would implement this API. Internally it would be implemented using System_linkControl API
System_linkDelete	Deletes a link - free's driver, codec, memory resources. Note, the OS thread itself is not deleted
System_linkControl	Send a link specific control command with optional arguments

Inter Link API

- These API's is used by links to exchange buffers with another link.
  - Users of a link typically need not be aware of this API.
  - However customers can use this API to write custom links depending on their specific requirements
- Each link needs to implement a few function callbacks and register the function pointers with the system frame work along with its link ID. This registration is done once during system init.
- Any link which wants to get buffers to/from another link will use the system API "System_getLinksFullBuffers()" to get buffers from the previous link. This internally will index into the system wide link information table and invoke the link specific function callback.
- Similarly when a link wants to release the buffer back to the original link after the buffers have been consumed, it will call the API "System_putLinksEmptyBuffers()". This internally will index into the system wide link information table and invoke the link specific function callback.
- This way a link need not exactly know which link it is exchanging buffers 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.
- This also allows to hide inter processor buffer exchange

Inter Link API – System Buffer

- The unit of exchange across link is a “system buffer”.
  - A system buffer itself can be different types like video frame, video bitstream, meta data etc.
  - The mechanism of exchange is same for all types of buffer.
  - More Buffer types can be added as required.
  - Internally a driver or a codec may use a different structure to represent a say a video frame.
  - The specific link implementation will take care of translating the System Buffer to driver or codec specific buffer information structure.
- Among other fields a System Buffer will have the below fields
  - Buf type – Video frame or Video bitstream or Composite video frame etc
  - Channel number – to identify a video/processing channel among multiple channels in the system
  - Timestamp
  - Sequence Number
  - Payload Size – size would be different for different buffer types
  - Payload pointer – pointer to specific buffer information

Inter Link API – Link callbacks

- When implementing a link the below callbacks needs to be implemented by the link writer

API	Description
System_GetLinkInfoCb	Function to return information about a link like number of channels, properties of each channel
System_LinkGetOutputBuffersCb	Function to return captured or generated or output buffer to the caller (another link) ONLY valid for links which output buffers
System_LinkPutEmptyBuffersCb	Function to release consumed buffers back to the original link for reuse ONLY valid for links which output buffers
System_getLinksFullBuffers	Function call by a link to get input buffers from previous link for processing. Internally the framework calls the link specific callback. ONLY links which takes buffers as input call this function
System_putLinksEmptyBuffers	Function called by a link to release consumed buffers to the previous link. Internally the framework calls the link specific callback. ONLY links which takes buffers as input call this function

Inter Link API – Link output queues (1/2)

- A link will have one or more output queues into which it will put the captured or generated buffers. A link owns it output queue and takes care of memory allocation for the buffers 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, Capture link can be configured to output its channel frames over two output queues, such that 4CH of 8CH goto one output queue and other 8CH go to other output queue. This allows Capture to feed to two different Display links in some data flows.
- An output queue can hold buffers 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 System_Buffer, is used for exchanging buffer information between links. This allows buffer information to flow between links without any additional translation. Among other information it has a "channelNum" field which allows a link to identify the channel with the buffer data.

Inter Link API – Link output queues (2/2)

- A link will typically call the System_getLinksFullBuffers() with the link ID and queue ID of the previous link when it wants to process the input buffers.
- A link when it has generated output buffers 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_getLinksFullBuffers(). After processing the input buffers it will release the input buffers using System_putLinksEmptyBuffers()
- Thus a link needs to know
- Previous link ID and Previous Link Queue ID to get input buffers
- And Next Link ID, in order to inform the next link when new buffers 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.

Vision SDK Architecture Details
Inter processor Communication

Inter processor communication (IPC)

- VISION SDK uses IPC package for inter processor communication.
- IPC needs one Shared region 0 to store meta data for IPC objects like MessageQ, ListMP etc. Shared region can also be used to create multi processor heaps.
- VISION SDK will configure shared region as below

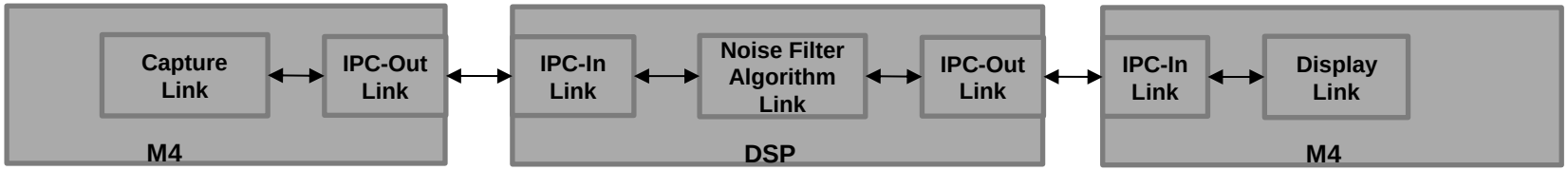
Shared region	Rough size	Location	Cache policy	Purpose
SR0	~ 16MB	DDR	NON CACHED at all processors (M4/DSP/A15/EVE)	IPC queue elements, MessageQ buffers, IPC internal multi processor data structures
SR1	~ 151MB	DDR	CACHED at all processors (M4/A15/DSP/EVE)	Frame/Bitstream/VideoAnalysis data heap

IPC - Sending control commands across processors

- Message queue will be used to send control commands to the individual links.
- This will be used for low frequency (once per 1sec kind of rate) control commands.
- This mechanism will NOT be used for frame to frame exchange (ex, at 30fps) between two processors.
- MessageQ in general does some parameter copies (user parameter pointer to shared memory buffer) and has higher overheads when passing message's across processor, so it is not used for frame to frame exchange.
- MessageQ however is easier to use and allows generic parameter passing, hiding many low level details from the user.
- Hence it is used for control messages and not for per frame level exchange

IPC - Exchanging buffers across processors (1/2)

- Each link itself is implemented thinking that it is exchanging buffers with another link on same processor.
  - This approach simplifies the implementation of the processing link since it need not worry about inter processor communication. Also it allows the same processing link to be used to talk to links on different processors and/or same processor without any changes
- A pair of special links are used when buffers need to go across different processor's
- The special links are
  - IPC OUT
  - IPC IN
- The IPC links always operate as pair as shown below



- Here Capture link on M4 will send its buffers to IPC OUT link running on the same CPU. So from Capture link point of view it is talking to a local link

IPC - Exchanging buffers across processors (2/2)

- The IPC OUT link will take buffer information translate it in a format which can be shared using IPC shared memory and send it the IPC IN link the CPU2.
  - The low level IPC mechanism used will be a non-locking queue implemented using IPC Notify + IPC SharedRegion
- The IPC OUT will signal the IPC IN on availability of new data via IPC Notify mechanism.
  - Here the 32-bit link ID of the IPC IN link is passed as the payload
  - This allows us to use a single Notify for a given processor and that processor could be running multiple instances of IPC IN each talking to a different IPC OUT
- The IPC IN link on receiving the Notify will get information from the IPC queue and translate this information in a format the Processing Link 2 on the same CPU can understand. This way processing link 2 can be written assuming it talking to a link on the same processor
- The IPC IN/OUT takes care of cache operation on the buffer information, address translation (since each CPU could be running on different address spaces)
- When the IPC OUT/IN pair operate on pair of CPUs which have uni-cache then it would directly send the buffer information pointer to the other CPU without any translation

IPC – Performance

- Frame exchange done using IPC incurs low CPU overhead and latency due to
  - links talk to each directly without any host CPU intervention,
  - non-locking queue implementation ensures get/put from the queue incur minimum overheads
  - Interrupt to the next CPU is done using the lowest level IPC API (Notify), this in turn uses the HW mailbox to trigger the interrupt
- The table shows the measured latency for a frame of exchange via IPC (Refer datasheet for details)

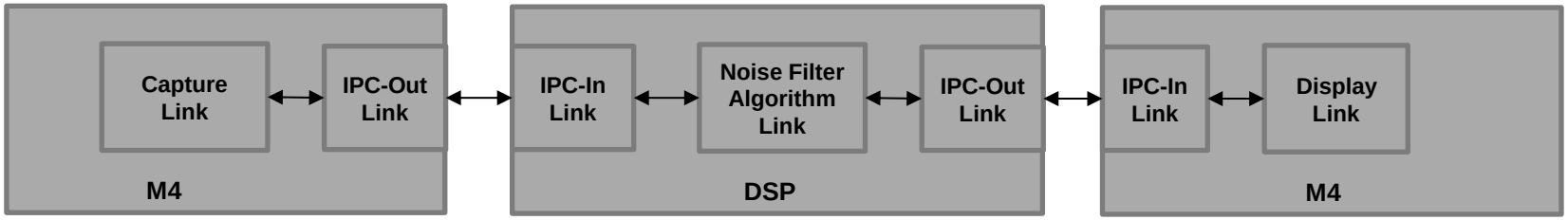
DST \ SRC	A15	DSP	IPU	EVE
A15	NA	23	45	54
DSP	20	21	45	57
IPU	66	109	135	134
EVE	152	159	179	201

(*) Latency measured in micro-seconds

Vision SDK Architecture Details
Algorithm Link

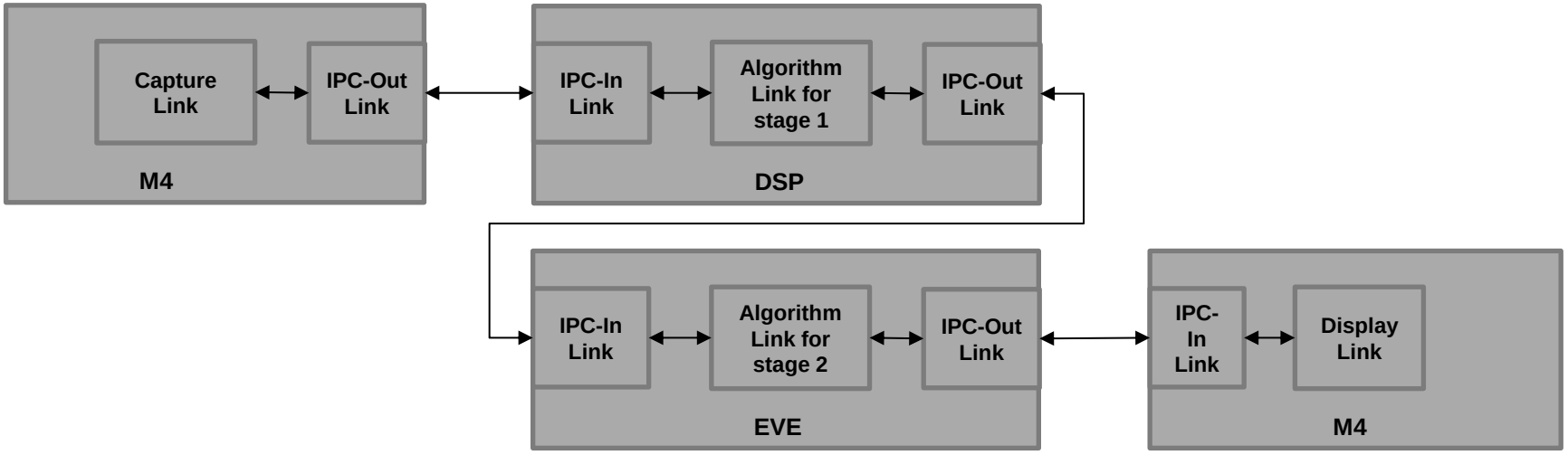
Algorithm link for DSP/EVE/A15 (1/2)

- Algorithm link is a link like any other link in the system chain.
- While the capture or display link encapsulate the corresponding drivers in them, Algorithm link encapsulates an algorithm in it.
- Algorithm links can be connected with other links in the system to form a chain.
- An example of a noise filtering algorithm link on DSP, connected in the system chain, looks like below



Algorithm link for DSP/EVE/A15 (2/2)

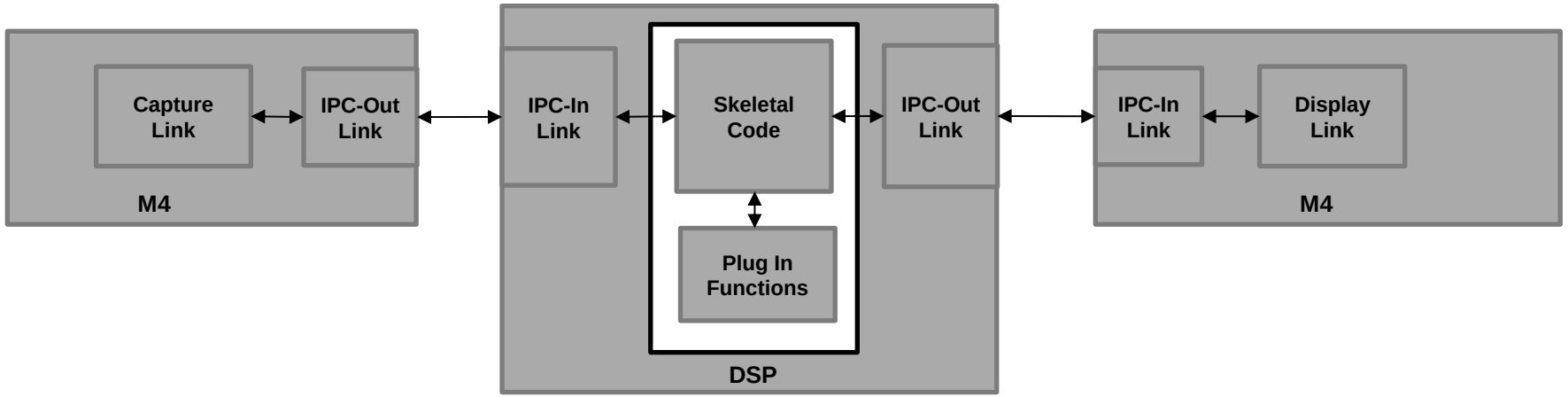
- Algorithm encapsulated within the algorithm link can be a complete application or a stage of an application (For Ex Pedestrian detection, Median filtering etc..).
- Several stages of an application spread across several cores can be connected by forming a chain between multiple algorithm links as follows:
- A two stage application spread across two cores is as shown below



- If several stages of an application execute on a same core in a sequential manner, they can be encapsulated within a single Algorithm link, by calling the stages in a sequential manner.

Algorithm link Design (1/2)

- To enable easy and fast development of algorithm links, the link is designed to consist of two portions - Skeletal and plug-in functions
- Skeletal part of algorithm link:
  - Comprises of portions of algorithm link implementation, which are common across algorithms
  - Takes care of generic aspects of link implementation like link creation, link state machine, communication with other links etc.
  - Provided by TI
- Plug-In Functions:
  - Comprises of functions which cater to algorithm dependent functionality
  - Needs to be written, specific to the algorithm being integrated



Algorithm link Design (2/2)

- Skeletal portion of the code and plug in functions communicate with each other via a predefined API (Refer - \vision_sdk\include\link_api\algorithmLink_algPluginSupport.h).
- Skeletal code implementation and the communication API is kept same, independent of the processing core (EVE/DSP/A15/M4)
- Skeletal code shall call the Plug-In functions based on the state of the algorithm link.
- Plug in functions have the implementation to create and use the actual algorithm functions (Provided by the algorithm provider)
- Plug-In functions can interact with algorithm functions via iVision or any other custom interface.
- List of Plug in functions are as follows:

AlgorithmLink_AlgPluginCreate	Plug in function which will perform algorithm instance creation
AlgorithmLink_AlgPluginProcess	Plug in function which will process new data. Internally it will call the process function of the algorithm
AlgorithmLink_AlgPluginControl	Plug in function which will perform Control (Configuration) of the algorithm. Internally it will call the control function of the algorithm.
AlgorithmLink_AlgPluginStop	Plug in function which will perform all functionality which needs to be done at the end of algorithm. Example: If any buffers are locked inside the algorithm, they can be flushed in this function.
AlgorithmLink_AlgPluginDelete	Plug in function which will perform algorithm instance deletion

Integrating Algorithm into Vision SDK

- To integrate an algorithm into Vision SDK, the corresponding algorithm link needs to be developed
- Developing an algorithm link means development of the plug-in functions for that algorithm
- These plug-in functions need to be registered via register API
- Each algorithm will have an unique Alg Id.
- Once the plug-in functions for an algorithm are registered, the use case can create an algorithm link with the corresponding Alg Id and this link can be used like any link in the system