FTF FREESCALE TECHNOLOGY FORUM 2014

A Deep Dive into **Image Processing** for i.MX 6 Application Processors

FTF-CON-F0119

Oliver Brown | Senior Software Engineer

APR.10.2014

freescale™

External Use

Agenda

- Introduction
- IPUv3 – system overview
- IPUv3 – fundamentals
- IPU & the iMX Linux BSP
- Use case examples / tips

Introduction

- IPU (Image processing Unit) is present on most of i.MX products
- IPUv1 was 1st introduced on i.MX31 and upgraded on i.MX35
- IPUv3 is a family of IPs that are present on MX37, i.MX51, i.MX53, i.MX6Q/D/DL

Introduction

- This presentation will describe IPUv3 on i.MX5 and i.MX6.
- The slides are based on i.MX6
- IPUv3 architecture is common for all products
- The differences between one product to another are on
  - Processing speed (from 133Mhz to 264Mhz)
  - The modules included (CSI, VDI, ISP etc)
  - The connectivity options and interfaces (HDMI, LVDS, MIPI etc)

Introduction

- The following slide set is based on past versions of IPUv3
  - The slides used for the i.MX51's NPI training can be found on:
    - http://compass.freescale.net/doc/195331621/0201_IPUv3EX_In_MX51.ppt
  - The slides used for the i.MX53's NPI training can be found on:
    - http://compass.freescale.net/livelink/livelink/218704608/iMX53_IPUv3M.ppt?func=doc.Fetch&nodeid=218704608
  - The slides used for the i.MX6's NPI training can be found on:
    - http://compass.freescale.net/livelink/livelink/224514161/Day2-1-iMX6_Dual-Quad_NPI_Training_-_IPU.pptx?func=doc.Fetch&nodeid=224514161

IPUv3 resources

Freescale Internal Links

- IPU on Compass
  - http://compass.freescale.net/go/ipu
  - http://compass.freescale.net/go/ipudes
- IPUv3 code examples for MX6/Q
  - http://compass.freescale.net/livelink/livelink?func=ll&objId=222977460&objAction=browse&viewType=1
- IPUv3 code examples
  - http://compass.freescale.net/go/189478969
- IPUv3 users mail list
  - IPUFORUM@freescale.com

External Links

- https://community.freescale.com/community/imx

Video/Graphics System in i.MX6 D/Q

Video & Graphics System in i.MX6 D/Q

Full HW Support –> Multiple Advantages

- The CPU does not have to touch pixels –> available to run application
- Optimized data path –> reduced DDR load –> complex use cases with only 32-bit DDR memories
- Lower power consumption (because of both aspects above)

Video Sources

Displays

Interface Bridges
LVDS, HDMI, MIPI

DCICs
Display Content Integrity Check

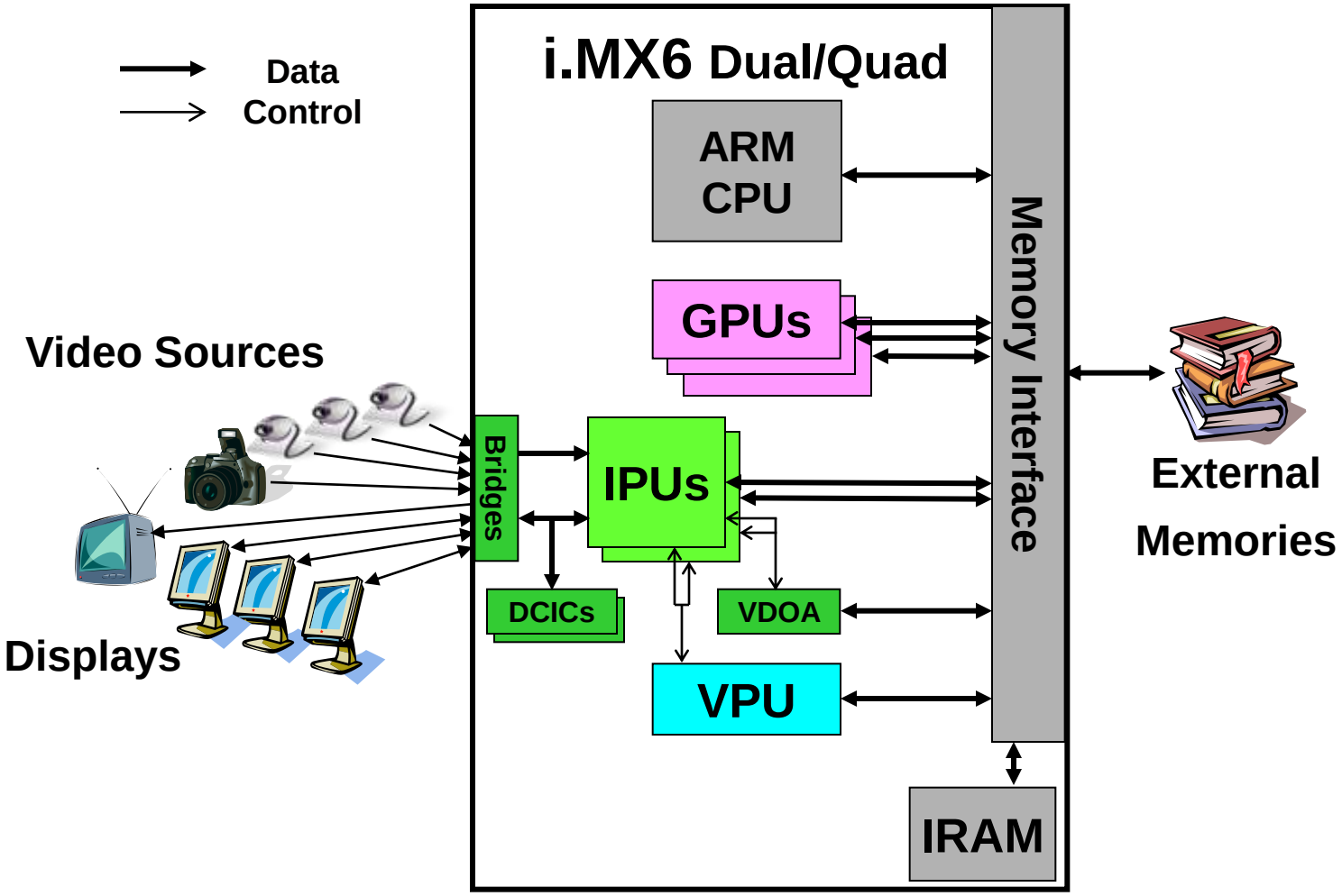
IPUs
(Image Processing Unit)

- Connectivity to relevant devices
- Image processing: conversions, enhancement…
- Synchronization and control

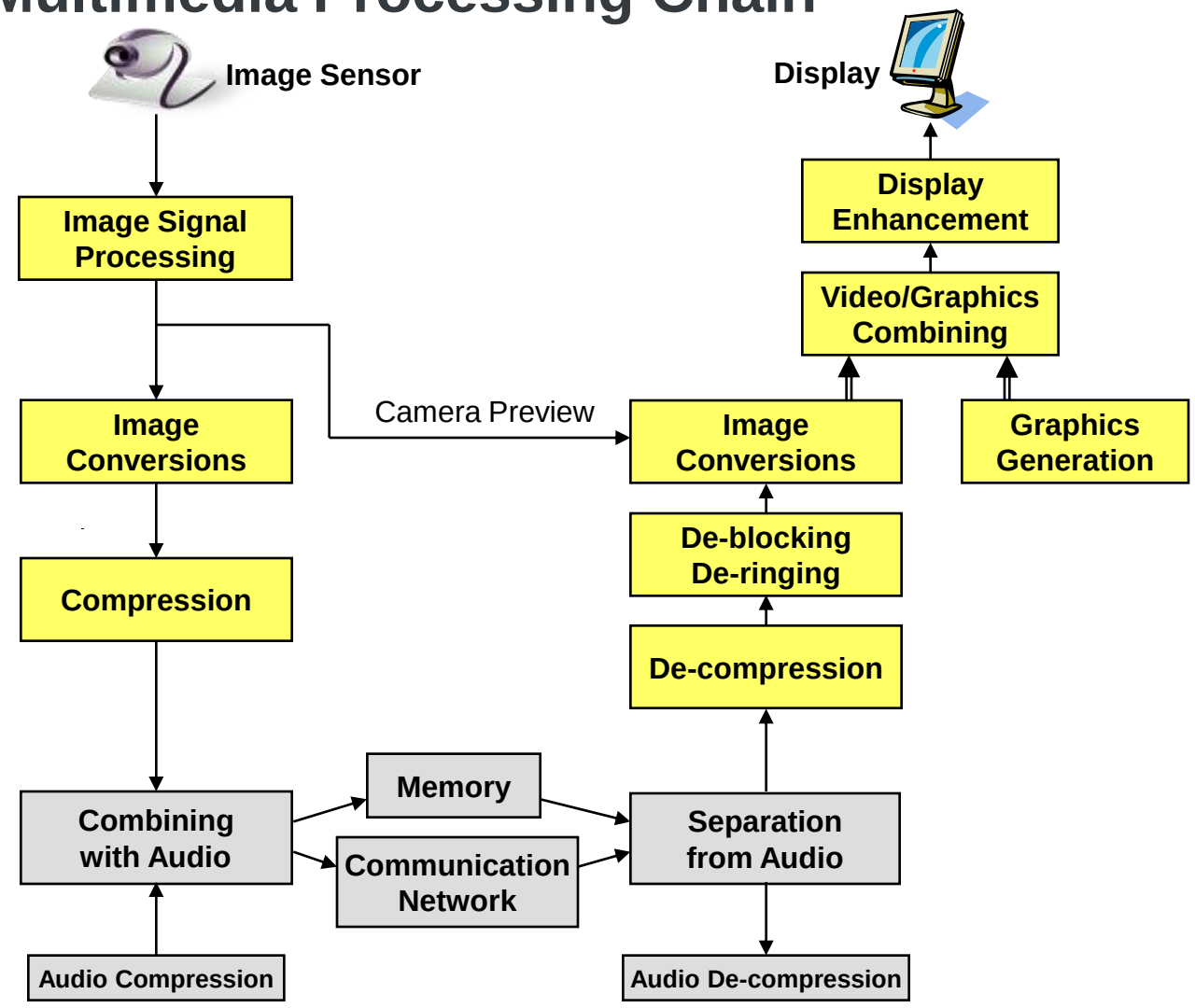
VPU
(Video Processing Unit)
Video encoding and decoding

GPUs
(Graphics Processing Units)
Graphics generation

Video/Graphics Subsystem in i.MX6 D/Q

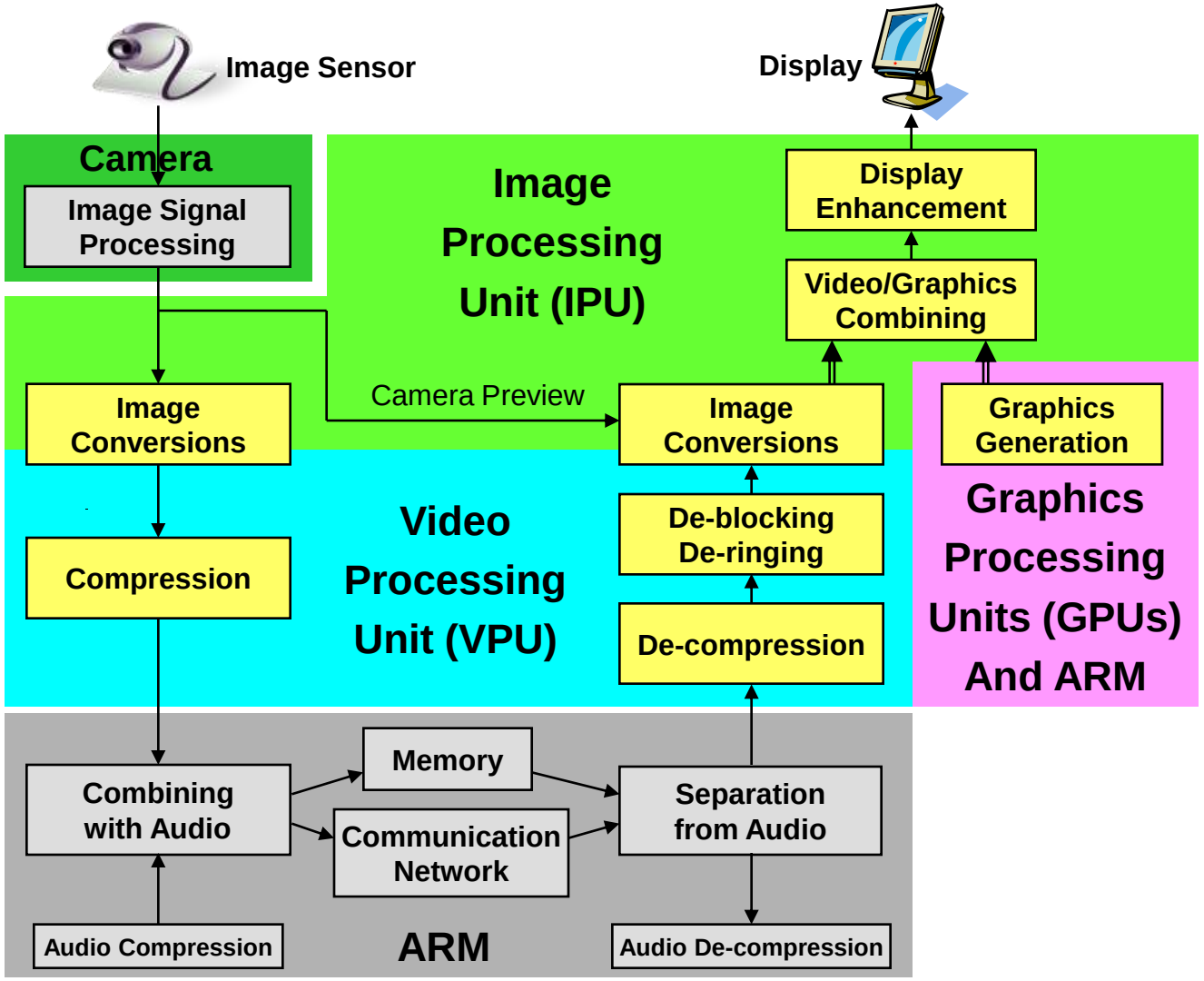


Multimedia Processing Chain



- Image Signal Processing
  - Bayer -> YUV conversion
  - Image quality enhancement
  - Camera corrections
- Image Conversions
  - De-interlacing
  - Resizing (resolution adjustment)
  - Rotation & Inversion
  - Color Space Conversion
  - Pixel Format Conversion (packing…)
- Display Enhancement:
  - Color adjustments and gamut mapping
  - Gamma correction and contrast stretching
  - Compensation for low-light conditions and backlight reduction

...imedia Processing Chain – Implementation



- Comprehensive HW support:
- Video/graphics fully handled by IPU, VPU and GPU.
  -> The CPU does not have to touch pixels

HW Accelerated

Display Support

i.MX37, i.MX51, i.MX53, i.MX6 D/Q – Display Support

How to calculate the display resolution?

- FW = Frame Width
- FH = Frame Height
- FPS = Frame rate (fps)
- BI = Blanking interval
  - Provided in the display's DS up to 35% (1.35).
  - Use min values

The pixel clock [MHz] is calculated according to:

F = FW X FH X FPS X BI

- Few things to consider:
  - Data format (pixel per clock?)
  - Display's clock source (DI#_CLK_EXT bit)
  - The load on the display controller (DC)

i.MX37, i.MX51, i.MX53, i.MX6 D/Q – Display Support

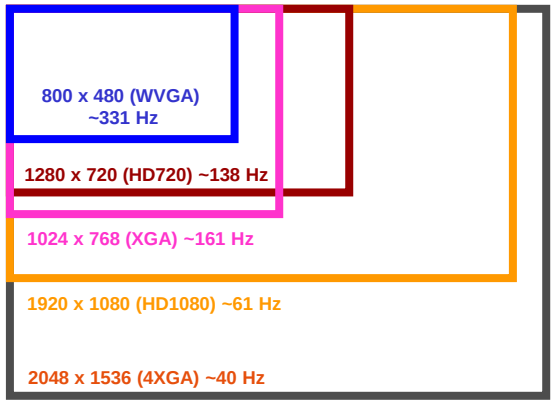
Feature		i.MX51 (IPUv3EX) i.MX37 (IPUv3D)	i.MX53 (IPUv3M)	i.MX6 D/Q (2 x IPUv3H)
Throughput	# of outputs	2		4
	Pixel clock rate	Up to 133 MHz	Up to 200 MHz	Up to 266 MHz per IPU
	Resolution (@ 60 Hz)	WXGA+ (1600x900) 720p (1280x720) + SVGA (800x600)	WUXGA (1920x1200) 1080p (1920x1080) + WVGA (800x480)	2 x 4XGA (2048x1536) 2 x [1080p + WXGA (1280x720)]
Interfaces	Parallel	Two ports 24 bits + 18 bits	Two ports 24 bits + 24 bits	
		Synchronous (for display refresh) and asynchronous (to memory) Very flexible - glue-less connection to RAM-less displays, display controllers, and TV encoders.		
	LVDS	No	Two channels; consumer version (multiple pairs); 2x 85 MHz or 170MHz	
	HDMI	No		One port
	MIPI/DSI	No		One port, 2 lanes x 1 Gbps (non-Automotive)
	Analog	One port; TV-out i.MX37: SDTV i.MX51: also 720p60 or 1080i/p30	Also VGA Rate increased to 1080p60	None (phased out)
Display Content Authentication (CRC)		No		Yes, for 2 displays
Processing	On-the-fly combining (for high resolution displays)	2 planes (up to 3 more planes for lower resolutions)		For 2 displays, 2 planes for each (6 more planes for lower resolutions)
	Off-line combining	Up to 20 MP/sec	Up to 200+ MP/sec	Up to 500+ MP/sec
	Display enhancement	Color adjustment and smart gamut mapping; gamma correction and contrast enhancement Supporting effective proprietary algorithms		
	Backlight power optimization	Yes; Supporting efficient proprietary algorithms		

…n i.MX6 D/Q (IPUv3H) – Maximal Resolution & Refresh Rate

- Capabilities
  - Maximal display resolution: 4096x4096 pixels
  - Maximal pixel rate: 264 MP/sec (200MP/sec on MX53, 133MP/sec on MX51)

- Display refresh rate
  - The maximal refresh rate is: 264M / (W * H * B)
    - W*H is the display resolution
    - B is a factor >1 reflecting blanking overhead, e.g. as specified by VESA, CEA-861-D, etc.
  - The table provides the maximal refresh rates for some typical resolutions
  - Usually, the refresh rate is required to be at least 60 Hz, to prevent blinking.
  - The blanking overhead factor assumed for the calculation is 1.3.
    - The actual factor depends on the display and is often closer to 1, allowing higher resolutions @ 60 Hz (e.g. HD1440).
    - For example, for HD1080, the standard specifies B~1.2
  - This is the capability of each of the two IPUs, so the total capability of the processor is doubled.
  - Note: these rates refer only to screen refresh, gated by the capabilities of the display port. A full use case typically includes additional activities and to confirm its support with a given refresh rate, additional aspects – video processing capabilities, capacity of the memory system, etc. – should be also analyzed carefully.

Name	Resolution				Maximal Refresh Rate [Hz]
	Width	x	Height	Total [MP]	
VGA	640	x	480	0.31	666
PAL	720	x	480	0.35	592
WVGA	800	x	480	0.38	533
NTSC	720	x	576	0.41	493
SVGA	800	x	600	0.48	426
WSVGA	1024	x	600	0.61	333
XGA	1024	x	768	0.79	260
HD720	1280	x	720	0.92	222
WXGA	1366	x	768	1.05	195
WXGA+	1440	x	900	1.30	158
SXGA	1280	x	1024	1.31	156
SXGA+	1400	x	1050	1.47	139
WSXGA+	1680	x	1050	1.76	116
UXGA	1600	x	1200	1.92	107
HD1080	1920	x	1080	2.07	99
WUXGA	1920	x	1200	2.30	89
9VGA	1920	x	1440	2.76	74
4XGA	2048	x	1536	3.15	65
HD1440	2560	x	1440	3.69	56
4WXGA	2560	x	1600	4.10	50
4K x 2K	4096	x	2048	8.39	25

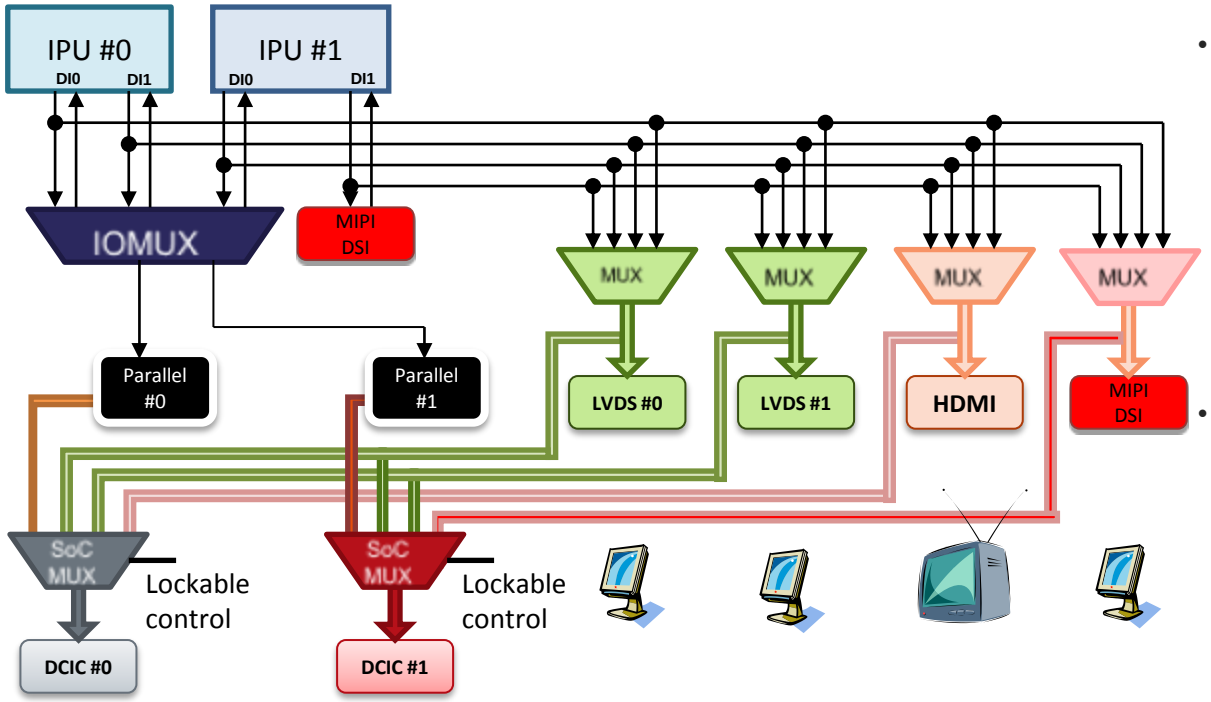


IPU in i.MX6 D/Q (IPUv3H) – Dual-Display Capabilities

First Display \ Second Display	SDTV 480i30/576i25 (27 MHz)	WSVGA (1024x600) (44-49 MHz)	HDTV 720p60/1080i30 (74.25 MHz)	WXGA (1366x768) (72-85 MHz)	WSXGA+ (1680x1050) (119-146 MHz)	HDTV 1080p60 (148.5 MHz)
WXGA (1366x768 ~ 1.0 MP; 72-85 MHz)	**Full**	**Full**	**Full**	**Full**	**Full**	**Full**
SXGA (1280x1024 ~ 1.3 MP; 91-109 MHz)	**Full**	**Full**	**Full**	**Full**	**Partial**	**Partial**
SXGA+ (1400x1050 ~ 1.5 MP; 101-122 MHz)	**Full**	**Full**	**Full**	**Full**	**Partial**	
WSXGA+ (1680x1050 ~ 1.8 MP; 119-146 MHz)	**Full**	**Full**	**Full**	**Full**	**Partial**	
UXGA (1600x1200 ~ 1.9 MP; 130-161 MHz)	**Full**	**Full**	**Full**	**Partial**		
WUXGA (1920x1200 ~ 2.3 MP; 154-193 MHz)	**Full**	**Partial**	**Partial**			
9VGA (1920x1440 ~ 2.8 MP; 185-234 MHz)	**Partial**	**Partial**				
4XGA (2048x1536 ~ 3.2 MP; 209-267 MHz)	**Partial**					

- Notes
  - ***This is the capability of each of the two IPUs, so the total capability of the processor is doubled.***
  - The maximal pixel clock rate supported by the display ports
    - Each display: 220 MHz
    - Total: 240 MHz
  - For a TV, the clock rate is fixed by the corresponding standards
  - For other displays
    - The assumed screen refresh rate is 60 Hz
    - The blanking overhead – impacting the pixel clock rate – may vary between displays.
      The table refers – for concreteness – to the VESA CVT (Coordinated Video Timing) specification
      - "**Full support**": allowing full blanking (which is typically required for CRTs)
      - "**Partial support**": allowing only reduced blanking (which is still typically sufficient for digital displays, e.g. LCDs)
  - The above table describes only the capabilities of the display ports to perform screen refresh. A full use case typically includes additional activities and to confirm its support with a given display configuration, additional aspects – video processing capabilities, capacity of the memory system, etc. – should be also analyzed carefully.

i.MX6 D/Q Display Ports Muxing



- Six ports
  - Two parallel - driven directly by the IPU
  - Two LVDS channels - driven by the LVDS bridge
  - One HDMI – driven by the HDMI transmitter
  - One MIPI-DSI – driven by the MIPI-DSI transmitter
- Four simultaneous outputs
  - Each IPU has two display ports (DI0 and DI1)
  - Therefore, up to four external ports can be active at any given time.
  - Additional asynchronous data flows can be sent through the parallel ports and the MIPI-DSI port
- Display Content Integrity Check (DCIC)
  - For parallel interfaces: probes the I/O loopback (essentially equivalent to probing the external wires)
  - For other integrated interfaces (e.g. LVDS): probes the IPU output (essentially equivalent to the inputs to the serializers)

Max Display Port Resolutions on i.MX6Q/D

- MIPI DSI, 2 lanes
  - WXGA (1366 x 768) or 720p (1280 x 720)
- RGB
  - Port 1 – 4XGA (2048 x 1536)
  - Port 2 – 4XGA (2048 x 1536)
- LVDS
  - Single channel – WXGA (1366 x 768) or 720p (1280 x 720)
  - Dual channel – UXGA (1600 x 1200) or 1080p (1920 x 1080)
- HDMI
  - 1080p (1920 x 1080) or 4XGA (2048 x 1536)

Note: Assuming 30% blanking intervals overhead, 24bpp, 60fps

Connecting a display on the parallel interface

i.MX51	LCD							
		RGB/TV Signal Allocation (Example)						Smart
Port Name (x=1,2)	RGB, Signal name (General)	16 bit RGB	18 bit RGB	24 bit RGB	8 bit YCrCb	16 bit YCrCb	20 bit YCrCb	Signal name
DISPx_DAT0	DAT[0]	B[0]	B[0]	B[0]	Y/C[0]	C[0]	C[0]	DAT[0]
DISPx_DAT1	DAT[1]	B[1]	B[1]	B[1]	Y/C[1]	C[1]	C[1]	DAT[1]
DISPx_DAT2	DAT[2]	B[2]	B[2]	B[2]	Y/C[2]	C[2]	C[2]	DAT[2]
DISPx_DAT3	DAT[3]	B[3]	B[3]	B[3]	Y/C[3]	C[3]	C[3]	DAT[3]
DISPx_DAT4	DAT[4]	B[4]	B[4]	B[4]	Y/C[4]	C[4]	C[4]	DAT[4]
DISPx_DAT5	DAT[5]	G[0]	B[5]	B[5]	Y/C[5]	C[5]	C[5]	DAT[5]
DISPx_DAT6	DAT[6]	G[1]	G[0]	B[6]	Y/C[6]	C[6]	C[6]	DAT[6]
DISPx_DAT7	DAT[7]	G[2]	G[1]	B[7]	Y/C[7]	C[7]	C[7]	DAT[7]
DISPx_DAT8	DAT[8]	G[3]	G[2]	G[0]		Y[0]	C[8]	DAT[8]
DISPx_DAT9	DAT[9]	G[4]	G[3]	G[1]		Y[1]	C[9]	DAT[9]
DISPx_DAT10	DAT[10]	G[5]	G[4]	G[2]		Y[2]	Y[0]	DAT[10]
DISPx_DAT11	DAT[11]	R[0]	G[5]	G[3]		Y[3]	Y[1]	DAT[11]
DISPx_DAT12	DAT[12]	R[1]	R[0]	G[4]		Y[4]	Y[2]	DAT[12]
DISPx_DAT13	DAT[13]	R[2]	R[1]	G[5]		Y[5]	Y[3]	DAT[13]
DISPx_DAT14	DAT[14]	R[3]	R[2]	G[6]		Y[6]	Y[4]	DAT[14]
DISPx_DAT15	DAT[15]	R[4]	R[3]	G[7]		Y[7]	Y[5]	DAT[15]
DISPx_DAT16	DAT[16]		R[4]	R[0]			Y[6]	
DISPx_DAT17	DAT[17]		R[5]	R[1]			Y[7]	
DISPx_DAT18	DAT[18]			R[2]			Y[8]	
DISPx_DAT19	DAT[19]			R[3]			Y[9]	
DISPx_DAT20	DAT[20]			R[4]				
DISPx_DAT21	DAT[21]			R[5]				
DISPx_DAT22	DAT[22]			R[6]				
DISPx_DAT23	DAT[23]			R[7]				
DIx_DISP_CLK	PixCLK							
DIx_PIN1								VSYNC_IN
DIx_PIN2	HSYNC							
DIx_PIN3	VSYNC							
DIx_PIN4								
DIx_PIN5								
DIx_PIN6								
DIx_PIN7								
DIx_PIN8								
DIx_D0_CS								CS0
DIx_D1_CS								CS1
DIx_PIN11								WR
DIx_PIN12								RD
DIx_PIN13								RS1
DIx_PIN14								RS2
DIx_PIN15	DRDY/DV							DRDY
DIx_PIN16								
DIx_PIN17								

IPUv3H – Video In Support

i.MX51, i.MX53, i.MX6 D/Q – Video In Support

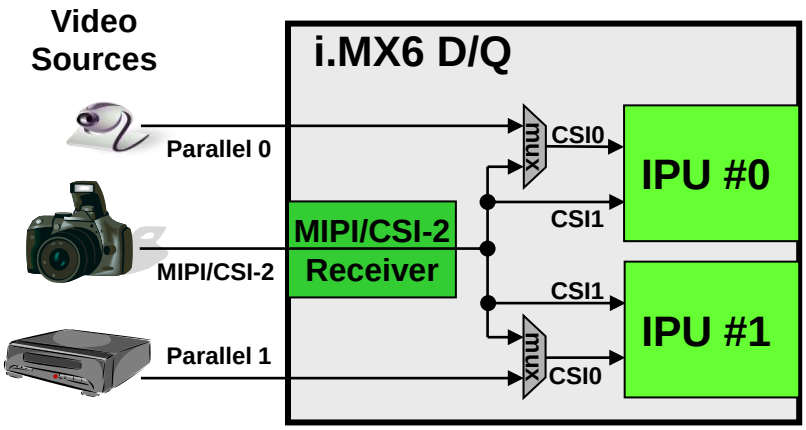
Feature		i.MX51 (IPUv3EX)	i.MX53 (IPUv3M)	i.MX6 D/Q (2 x IPUv3H)
Video Input Interfaces	Parallel	Two ports, 20 bits + 8 bits		
		120 MHz; e.g. 6 MP @ 15 fps	180 MHz; e.g. 9 MP @ 15 fps	200 MHz; e.g. 10 MP @ 15 fps
	MIPI/CSI-2	No		One port, 4 lanes x 1 Gbps
Video Rate	Playback	720p30 (1280x720) @ 30 fps	1080i/p (1920x1080) @ 30 fps	1080i/p + D1 @ 30 fps
	Record	D1 (720x480@30 fps or 720x576@25 fps)	720p30 (1280x720) @ 30 fps	1080p @ 30 fps
	2-way		720p @ 20 fps	720p @ 30 fps
Video Processing	De-interlacing	High-quality motion adaptive algorithm		
	Resizing	Yes – fully flexible		
	Rotation/inversion	Yes		
	Color conversion	Yes – fully flexible		
Memory Interface	Protocol	AXI – Including split transaction		
	Throughput	64-bit, 133 MHz	64-bit, 200 MHz	64-bit, 266 MHz
Efficient memory bus utilization		Selective read for combining		
Control capabilities		Display controller, DMA controller, Internal synchronization Autonomous operations: display refresh/update, view-finder		
Synchronization (to prevent tearing)		Double/triple buffering Frame-by-frame or tight – sub-frame (utilizing internal memory)		

Video Input Ports In i.MX6 D/Q

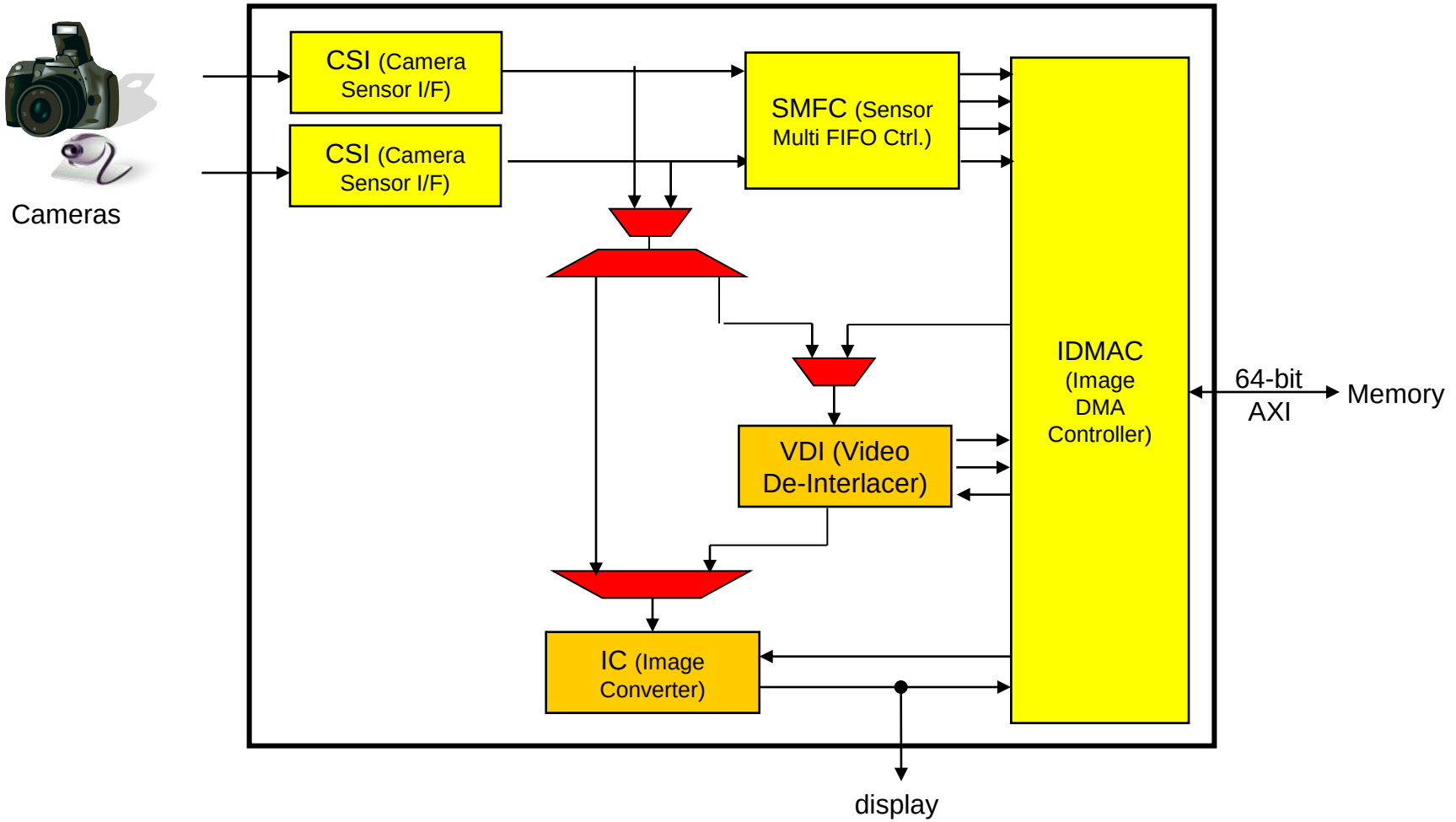
- Three ports; up to six input channels
  - Two parallel – connected directly to the IPUs; independent clock and format setting
  - One MIPI/CSI-2 – can transfer up to four concurrent channels
  - Each port: up to 150Mpxl/s @200MHz, e.g. 10Mpxl @ 15fps

- Four concurrent channels
  - Each IPU has two input ports (CSI0 and CSI1), each can process an input channel from one of the external ports.
  - The MIPI/CSI-2 bridge sends all its channels to all the IPU input ports and each port can select for processing a different channel, identified by its DI (Data Identifier).
  - Additional channels can be transferred through a CSI transparently – as generic data – directly to the system memory.

- Formats supported:
  - BT.656
  - BT.1120
  - YUV422, RGB888, YUV444 = over an 8 bit bus
  - RAW format up to 16bpp which will be translated to 8 bit using companding
  - Generic data up to 20bit



IPUv3H – The camera port



The Camera Sensor Interface - CSI

- Role: controls the camera port
  - Provides direct connectivity to relevant image sensors and connectivity bridges: CSI-2, HDMI receiver, TV decoder…
- Data bus – up to 20 bits
  - Single value – up to 16 bits
  - Two values – up to 10 bits each; e.g. HDTV YUV 4:2:2 input
- Variety of data formats
  - Main (with on-the-fly processing): YUV 4:2:2/4:4:4, RGB 16/24 bpp
  - Other: as generic data, including compressed streams
  - All primary CSI-2 formats
- Frame resolution
  - Up to 8192 x 4096 pixels
- Input rate
  - 240M values/sec peak (@ 264 MHz internal clock)
- Additional features
  - Frame rate reduction – by skipping (reduction ratio: m:n, m<=n<=12)
  - Window-of-interest selection – by cropping

i.MX37, i.MX51, i.MX53, i.MX6 D/Q – Video in Support

- How to calculate the video in pixel clock?

- FW = Frame Width

- FH = Frame Height

- FPS = Frame rate (fps)

- BI = Blanking interval
- Provided in the device's DS up to 35% (1.35).

- D = Data format
- How the data is arranged on the bus – (cycles/pixel)

The pixel clock [MHz] is calculated according to:

F = FW X FH X FPS X BI X D

16-bit camera support

- **16-bit YUV422**
  - CSI receives 2 components per cycle.
  - CSI [16 bit generic data.] => SMFC => MEM [16bit generic data] => IPU[YUV422]

- **16-bit RGB as generic data**
  - CSI receives 3 components per cycle.
  - Use a 16 bit sample of it (such as RGB565)
  - CSI [16 bit generic data.] => SMFC => MEM [16bit generic data] => IPU [16 bit RGB] => IPU [map to 24bpp RGB]

- **16-bit RGB565**
  - On the fly processing of 16 bit data.
  - CSI is programmed to receive 16 bit generic data.
  - The interface is restricted to be in "non-gated mode" and the CSI#_DATA_SOURCE bit has to be set
  - If the external device is 24bit - the user can connect a 16 bit sample of it (RGB565 format).
  - The IPU has to be configured in the same way as the case of CSI#_SENS_DATA_FORMAT=RGB565

BT.1120/BT.656 support

- BT.656
  - CCIR progressive/interlaced modes
- BT.1120
  - CCIR progressive/ interlaced mode
  - SDR/DDR mode
- The timing reference signals (Sync events) are embedded in the data.
- The CCIR codes are defined in the standard. IPU can support non standard codes using the CCIR registers.
- On the fly processing is supported in both modes

BT.1120/BT.656 support

- IPUv3 is an 8bit per component system.
- In a 10bit data inputs there are few ways to handle the data
- Companding - programmable piecewise-linear map
- Regular and tight packing to memory

BT.1120 mode	connectivity	companded	regular packing	tight packing
20bit, YUV422-10	D[19:0]	{8DC,8DC,8DC,8R}	{16DE,16DE,16DE,16R }	{10DE,10DE,10DE,2R}
20bit, YUV422-8	D[19:12], D[9:2]	{8DC,8DC,8DC,8R}	{8D,8D,8D,8R}	NA

DC - data after being companded
DE - data after being extended.
R- reserved bits

i.MX37, i.MX51, i.MX53, i.MX6 D/Q – Video in Support

Data Format	CSI Bus width	D (cycles/pixel)	comment
RGB888/YUV444	8	3	D[19:12]
YUV422	8	2	D[19:12]
YUV422	16	1	D[19:4]
Generic data	16	1	D[19:4]
RGB565	8	1	D[19:12]
RGB565	16	2	D[19:4]
BT.1120	20	1	D[19:0]
BT.1120	16	1	D[19:12], D[9:2]
BT.656	8	2	D[19:12]
Bayer	16	1	D[19:4]

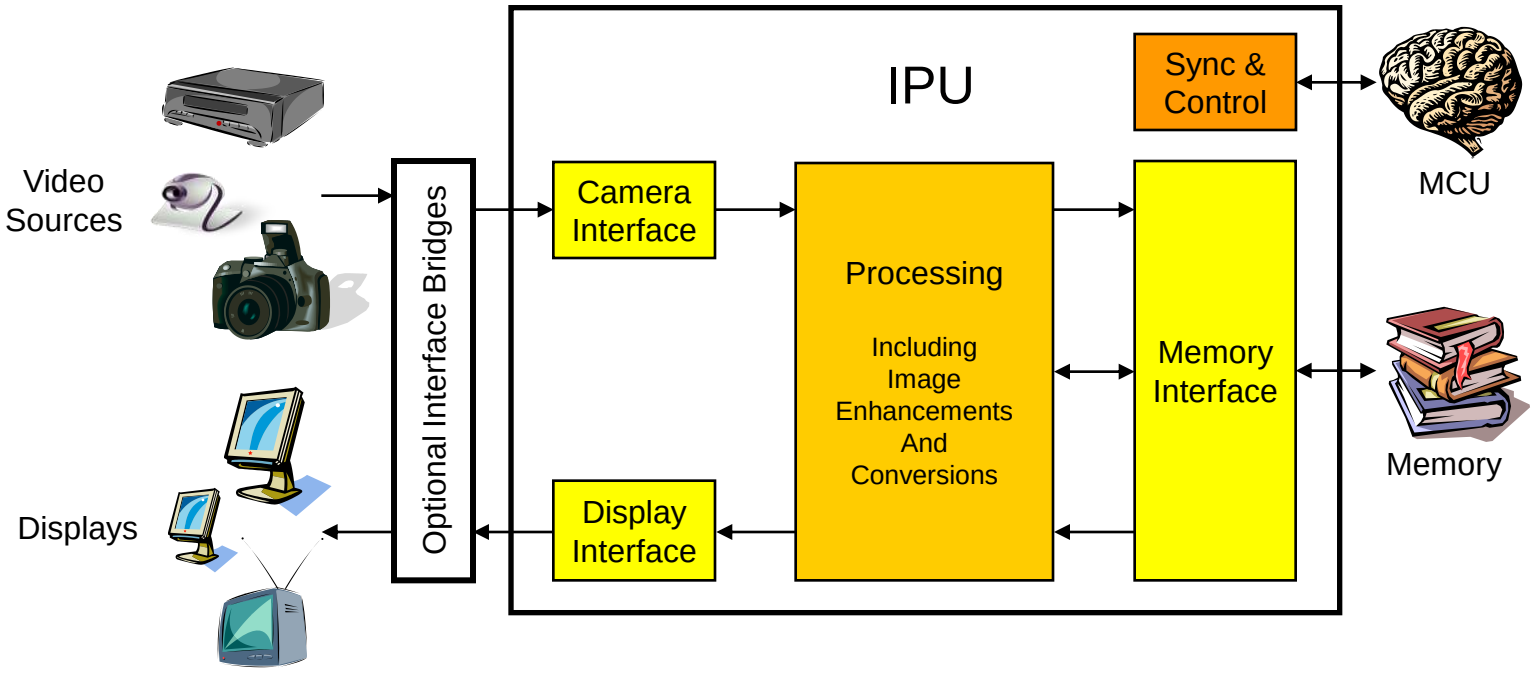
F = FW X FH X FPS X BI X D

CSI data mapping

	RGB888/ YUV444	RGB565 8bit	RGB565 16bit	YUV4:2:2	Generic data	BT.656	YUV422 16 bit	BT.1120 (YUV422-10)	BT.1120 (YUV422-8)
CSI1_D19	D7	D7	R4	D7	MSB	D7	Y7	Y9	Y7
CSI1_D18	D6	D6	R3	D6	MSB-1	D6	Y6	Y8	Y6
CSI1_D17	D5	D5	R2	D5	MSB-2	D5	Y5	Y7	Y5
CSI1_D16	D4	D4	R1	D4	MSB-3	D4	Y4	Y6	Y4
CSI1_D15	D3	D3	R0	D3	MSB-4	D3	Y3	Y5	Y3
CSI1_D14	D2	D2	G5	D2	MSB-5	D2	Y2	Y4	Y2
CSI1_D13	D1	D1	G4	D1	MSB-6	D1	Y1	Y3	Y1
CSI1_D12	D0	D0	G3	D0	MSB-7	D0	Y0	Y2	Y0
CSI1_D11			G2		MSB-8		CrCb7	Y1	0
CSI1_D10			G1		MSB-9		CrCb6	Y0	0
CSI1_D9			G0		MSB-10		CrCb5	CrCb9	CrCb7
CSI1_D8			B4		MSB-11		CrCb4	CrCb8	CrCb6
CSI1_D7			B3		MSB-12		CrCb3	CrCb7	CrCb5
CSI1_D6			B2		MSB-13		CrCb2	CrCb6	CrCb4
CSI1_D5			B1		MSB-14		CrCb1	CrCb5	CrCb3
CSI1_D4			B0		MSB-15		CrCb0	CrCb4	CrCb2
CSI1_D3								CrCb3	CrCb1
CSI1_D2								CrCb2	CrCb0
CSI1_D1								CrCb1	0
CSI1_D0								CrCb0	0

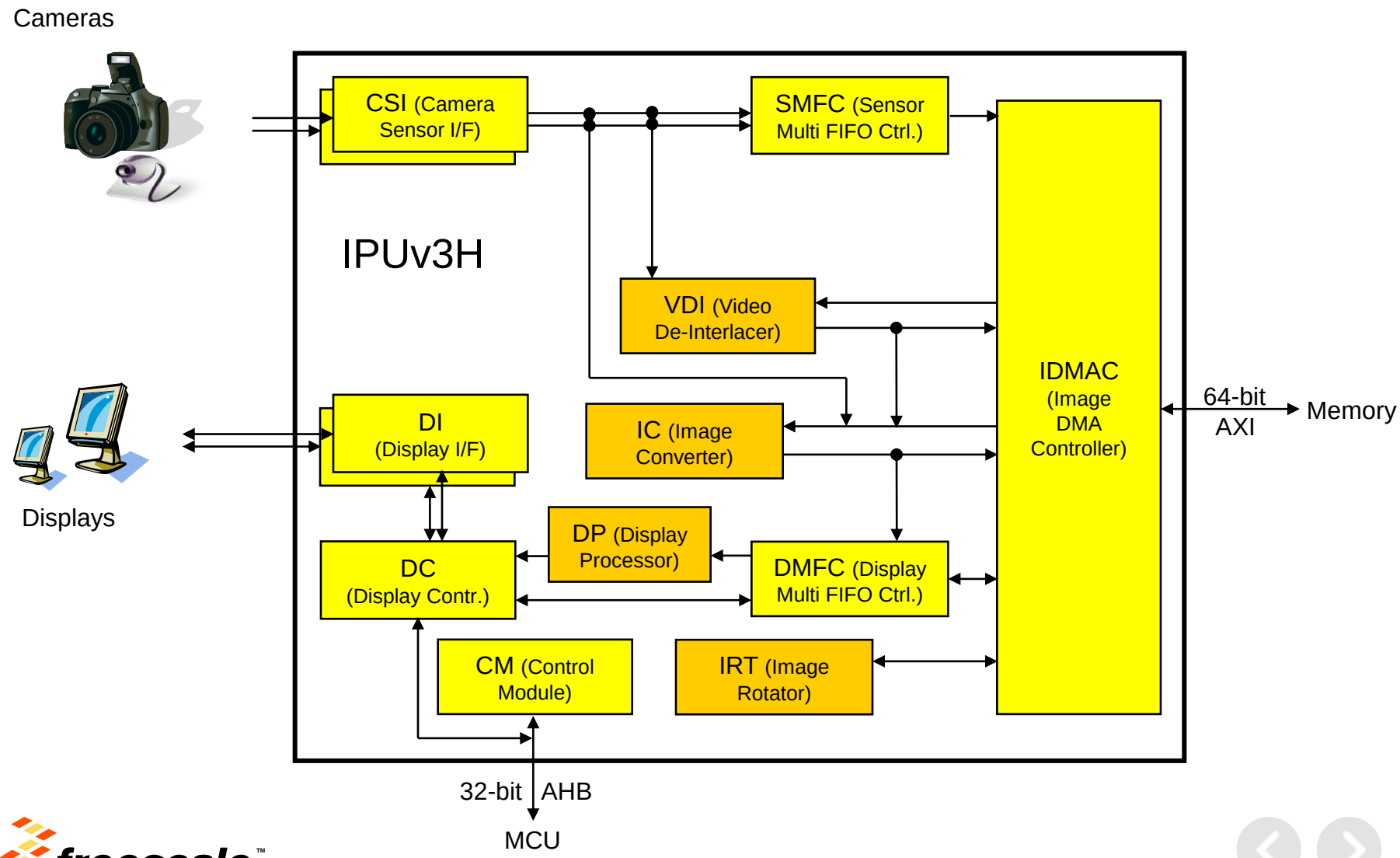
IPUv3H – Fundamentals

The Image Processing Unit



- Functions: comprehensive support for the flow of data from an image sensor and/or to a display device.
  - Connectivity to relevant devices
  - Related image processing and manipulation
  - Synchronization and control capabilities

IPUv3H – Internal Structure



IPUv3 Fundamentals - The display port

- The display port handles all the IPUv3 features targeted for controlling and sending data to the display.
- The display port consists of 4 modules:
- DC - a display controller
- DP - a display processor
- DMFC - a display multi-FIFO controller
- DI - a display interface. The DI is instantiated twice to provide two symmetrical display interfaces.

IPUv3 Fundamentals - Supported display interfaces

- The total number of supported displays by IPUv3 is 4.
- The display port has 2 DI interfaces.
- Each interface can handle up to 3 displays.
- Each DI can handle up to 2 asynchronous interfaces (e.g. Smart LCD, Graphic accelerator) - only one of them can be serial interface.
- Each DI can handle one synchronous interface (e.g. TV, dumb LCD).

IPUv3 Fundamentals – display channels' mapping

- The display port supports multiple flows that may have different characteristics
- In order to configure the IPU we need to identify some of the flow's characteristics and allocate the IPUv3's resources that will participate in that flow.
- First we need to understand how the channels are distributed.

IPUv3 Fundamentals – display channels' mapping

Ch #	Destination	DMFC/DC numbering	Flow's nature	Alpha channel	comment
21	DC		SYNC or ASYNC	NA	Direct flow via IC. If this flow is used it replaces once of the DMFC channels.
23	DP - primary	5B/5	SYNC	51	Ch 23 is associated with ch27. when there's only one plane in the flow – this channel should be used.
24	DP - Primary	6B/6	ASYNC	52	Ch 24 is associated with ch29. 2 ASYNC flows can use this channel via alternate flow. when there's only one plane in the flow – this channel should be used.
27	DP -Secondary	5F/5	SYNC	31	
28	DC	1/1	SYNC or ASYNC	NA	if ch28 is connected to DI0 then ch23 must be connected to DI1
29	DP -Secondary	6F/6	ASYNC	33	

IPUv3 Fundamentals – display channels' mapping

Ch #	Destination	DMFC/DC numbering	Flow's nature	Alpha channel	comment
40	DC	0/0	Read	NA	
41	DC	2/2	ASYNC	NA	
42	DC	1C	Command	NA	Refer to the spec for the command channel restrictions
43	DC	2C	Command	NA	Refer to the spec for the command channel restrictions
44	DC	3	Mask	NA	Mask channel can be associated with ch 23 or ch 28

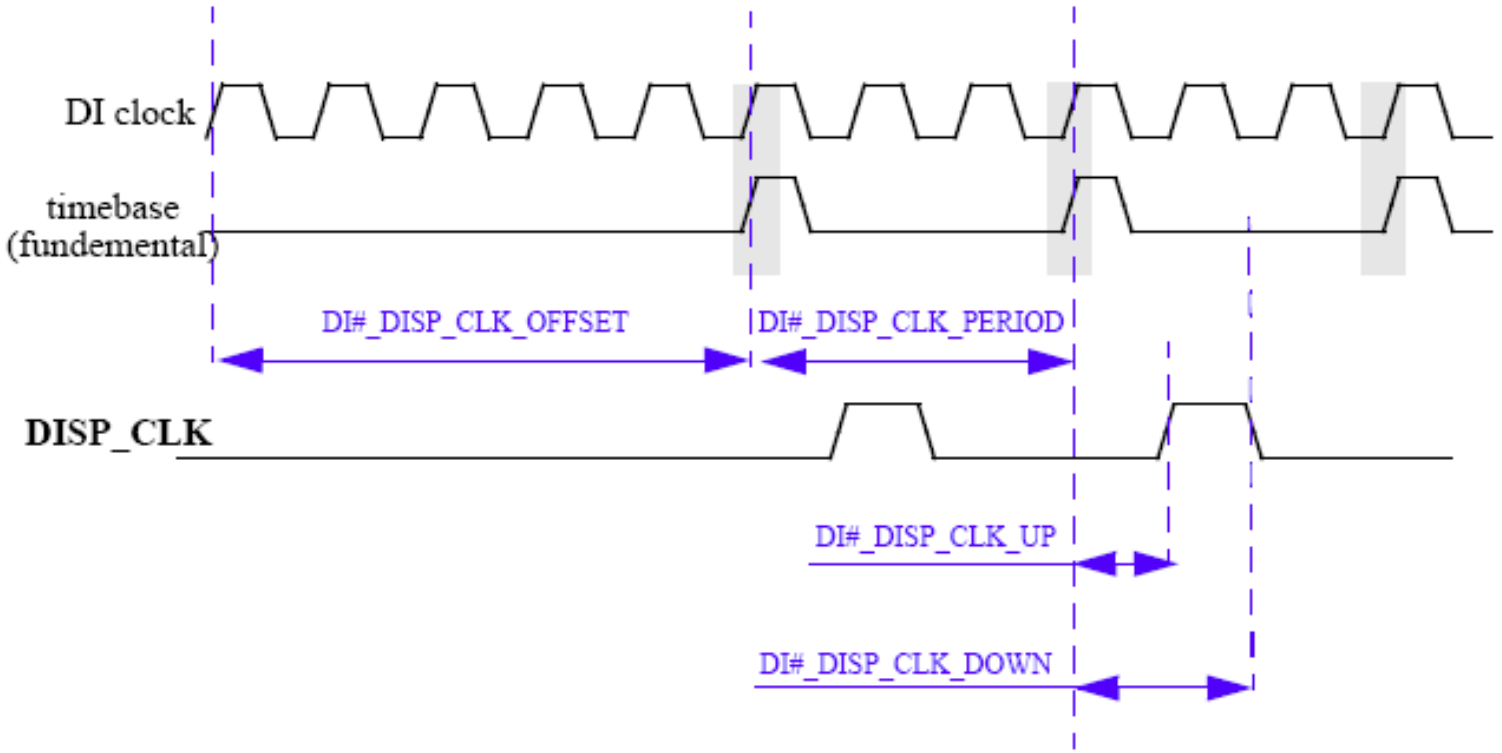
IPUv3 Fundamentals - DI

- The DI is responsible for the timing waveforms of each signal in the display's interface.

- The DI is composed of
  - 8 sets of waveform generators controlling signals associated with the DI's clock; These signals drive PIN1-PIN8. These pins can be used for signals like VSYNC,HSYNC
  - 12 sets of waveform generators controlling signals associated with the data; These signals drive PIN11-PIN17 + 2 CS signals. These pins can be used for signals like DRDY,CS,RS
  - The DI generates the clock to the display
    - The DI clock can be derived from the IPUv3 hsp_clk
    - The DI clock can be derived from an external to the IPU clock (PLL or pin)

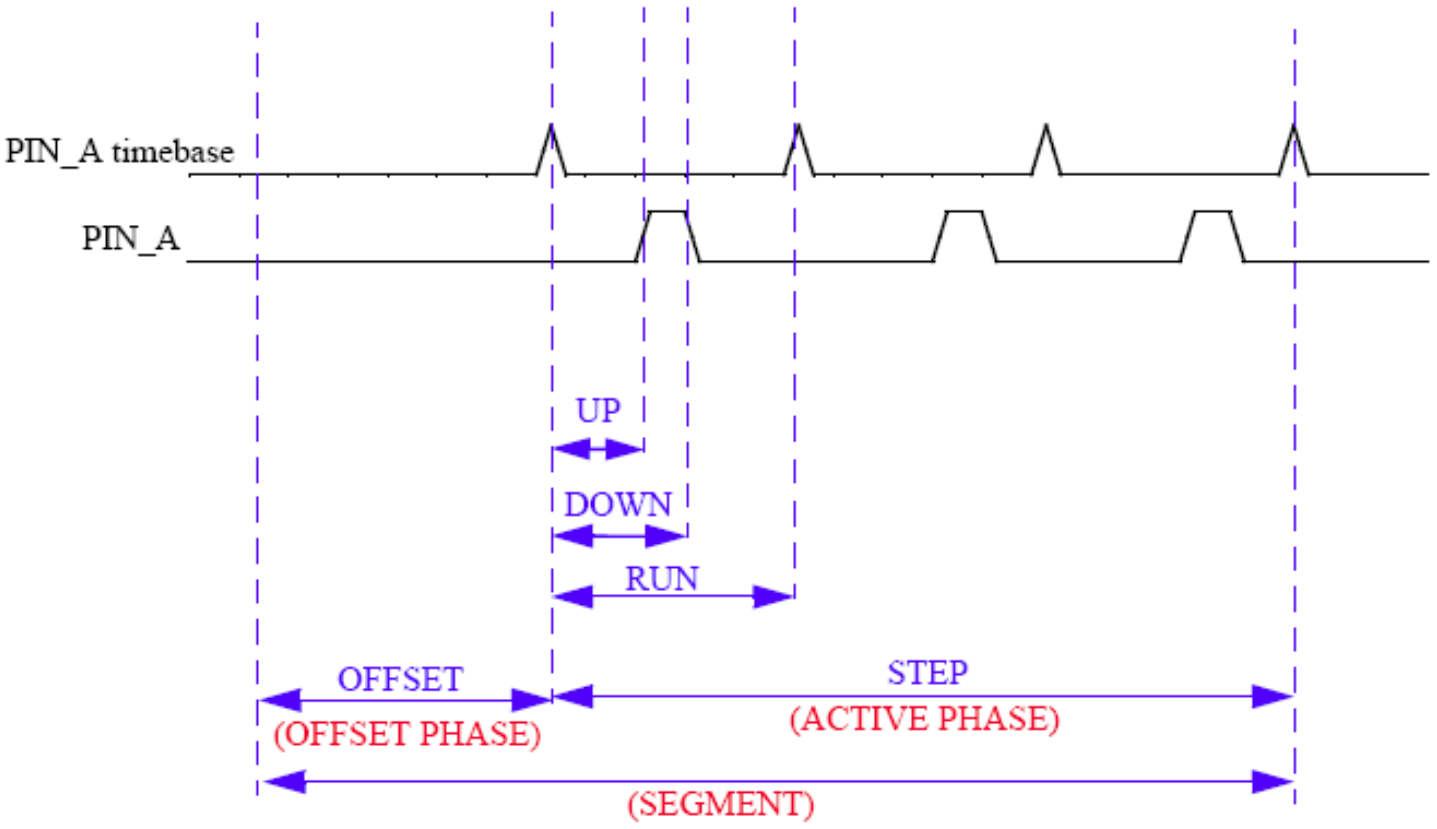
IPUv3 Fundamentals - DI

- This waveform describe how the display clock's parameters are set.



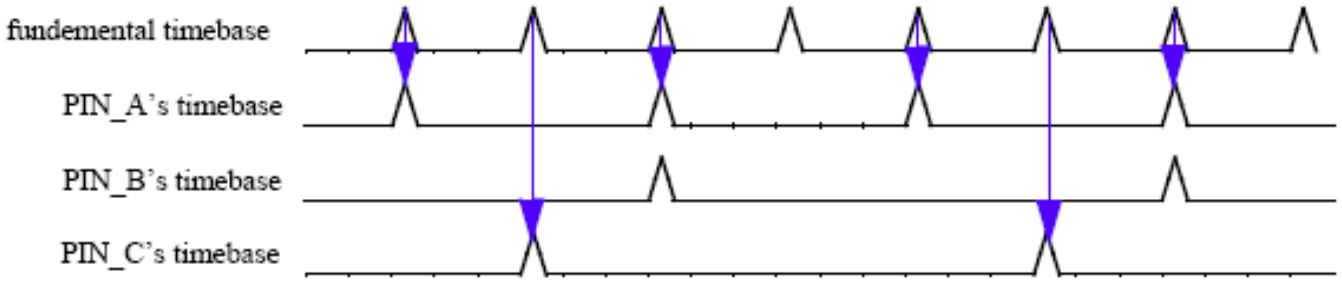
IPUv3 Fundamentals - DI

- This waveform describe how the DI's PIN parameters are set.



IPUv3 Fundamentals - DI

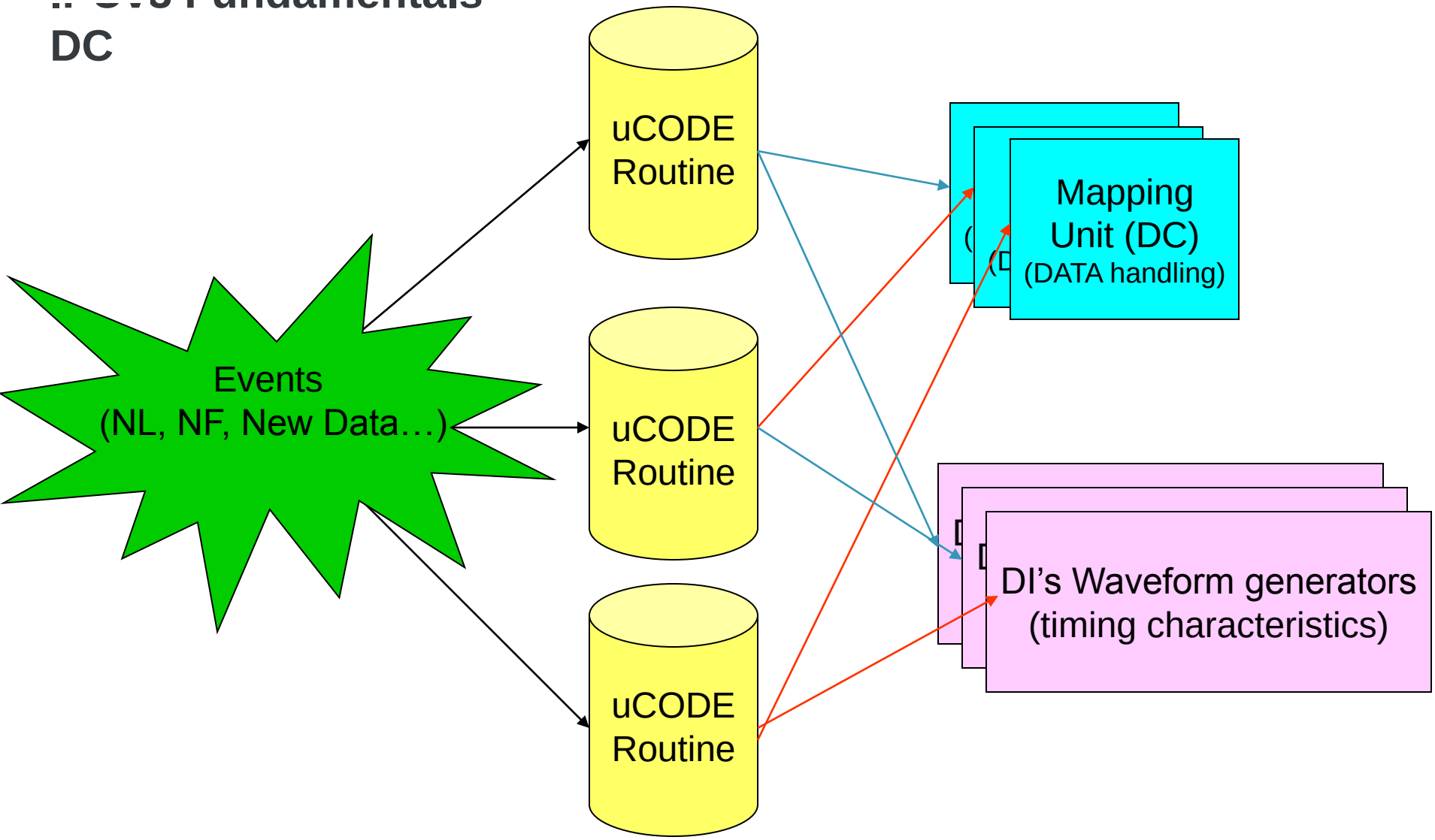
- This waveform provides an example of waveform concatenation.



IPUv3 Fundamentals - DC

- The DC (Display Controller) is responsible for:
- Activation of a flow
  - When there's new content to be displayed (asynchronous flow)
  - Upon an internal timer (synchronous flow)
- Linkage between the microcode to
  - mapping units (within DC)
  - and timing units (within DI)

..3 Fundamentals - DC



IPUv3 Fundamentals - DMFC

- The DMFC is a multi FIFO controller utilizing a single memory to serve the DC and DP channels.
- The FIFO is partitioned to 8 equal segments. The segments can be allocated asymmetrically to the channels.
- The memory allocation for a specific channel must not be greater than a certain number of rows. The exact number is different from one channel to another (see the spec).
- When the direct path from the IC is used, this channel replaces one of the existing DMFC channels.

The Image DMA Controller – IDMAC

- Role: control the memory ports; transfer data to/from system memory
- Memory ports - AXI
  - IMDAC: 1 read, 1 write
- Throughput:
  - External: 64 bits @ 264 MHz
  - Internal: up to 2 pixels/cycle @ 264 MHz (through each port)
  - Shared by all DMA channels: input from sensor, output to display and off-line processing
    -> efficient utilization of the bandwidth in different use cases
  - Efficient pipelining: 4 AXI ID's; multiple outstanding transactions: read – 8; write – 6
- Data arrangement in memory
  - Row-after-row, with flexible line-stride – as needed for a window in a video/graphics buffer
- Access order
  - Block-by-block – for rotation
  - Row-by-row – used for all other channels
    - Needed for output to display or input from sensor
    - Decreases the memory bus load by increasing its utilization efficiency

The Image DMA Controller – IDMAC (cont.)

- A variety of pixel formats
  - YUV 4:2:0/4:2:2/4:4:4 – for video
  - Conventionally packed RGB pixels – 8/16/32 bpp – for graphics
  - Tightly packed RGB pixels – 12/18/24 bpp
    - For reduced load on the memory bus and for power-efficient screen refresh
  - Optional independent alpha (translucency) input
    - For planes that do not have interleaved alpha
  - Coded color (using a LUT; 4/8 bpp)
    - An additional option to reduce bus load and power
  - Gray scale
  - Generic data

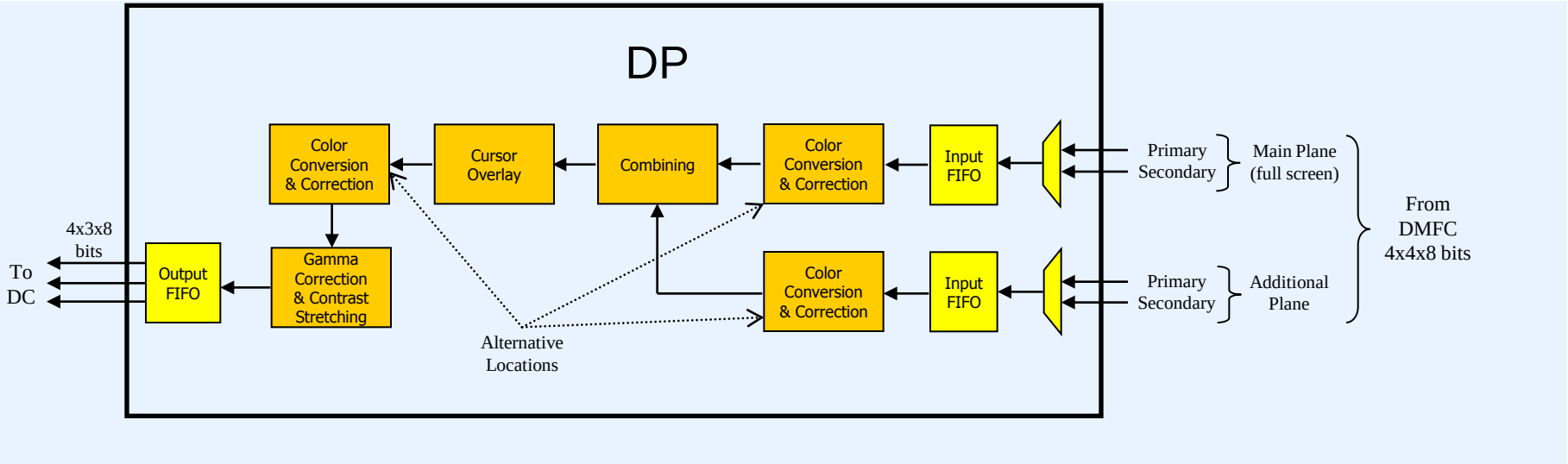
- Additional features
  - Conditional read (for combining) – transparent pixels are not read
    - Pixel transparency – identified from the independent alpha input
  - Scrolling and panning
  - Uniform programming model for all channels
  - (stored in the CPMEM – channel parameter memory)

IPUv3 Fundamentals - IDMAC

- The IDMAC of IPUv3 is connected to the AXI bus.
  - Full separation between read and write
  - All the channels are symmetric
  - 64-bit AXI bus, internal bus of 128-bit (4 pixels)
- Each channel uses 2X160 words of the CPMEM memory, holding the channels settings
- Ability to use alternate rows in the CPMEM for alternate flows
- Usage of alpha located in separate buffers (ATC)
- Dynamic and static arbitration between channels.

- Prioritizing screen refresh channels over other IPU channels and other DDR masters

DP (Display Processor)

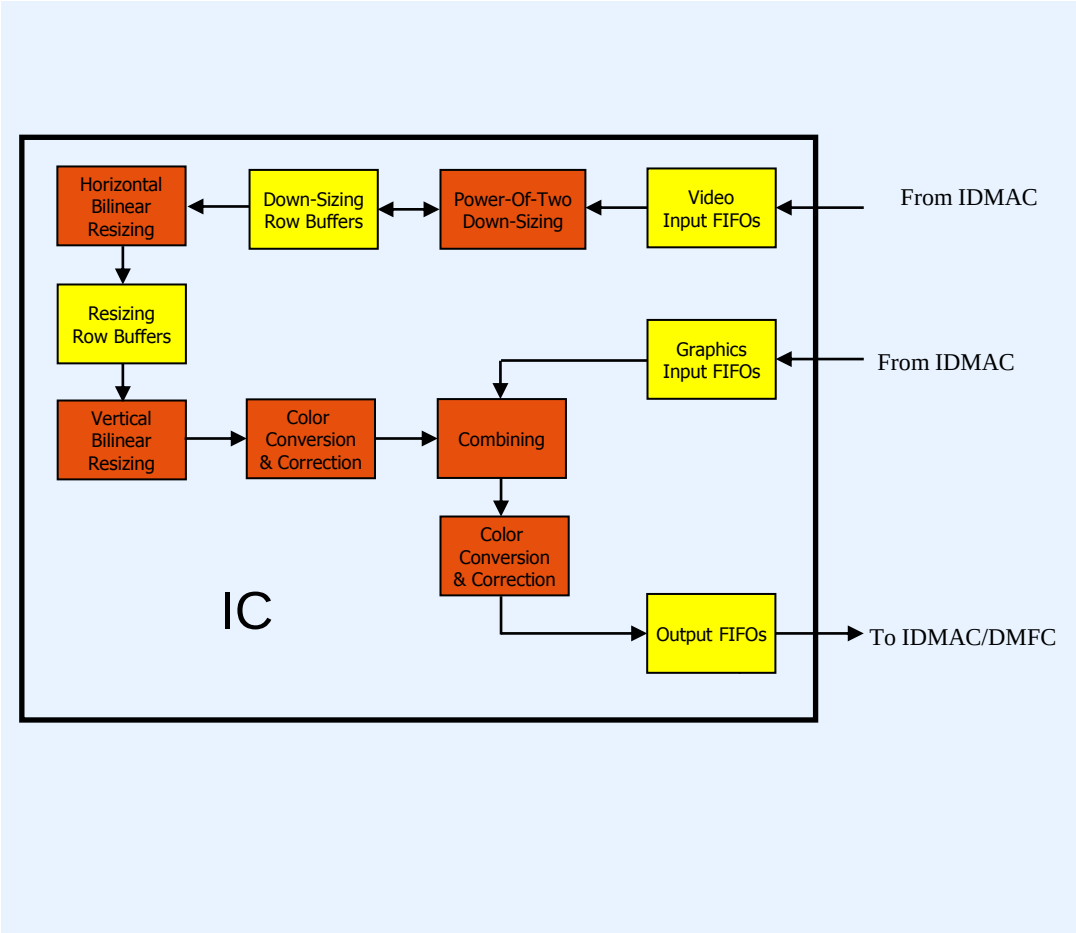


- DP has following features;
  - Support input format YUVA/RGBA
  - Combining 2 video/graphics planes
  - Color conversion (YUV <-> RGB, YUV<->YUV) & Correction (gamut-mapping)
  - Gamma correction and Contrast stretching
  - Support output format YUV/RGB
  - Dynamic task switching between async and sync flows

IPUv3 Fundamentals - DP

- The DP handles the content of the frame.
- It performs image processing on the way to the display (combining, CSC, gamma correction)
- The DP supports one synchronous flow and two asynchronous flows.
- All the DP configuration is done only via the SRM. The control module handles the automatic switch between DP settings when the DP switches from one flow to another.

IC (Image Converter)

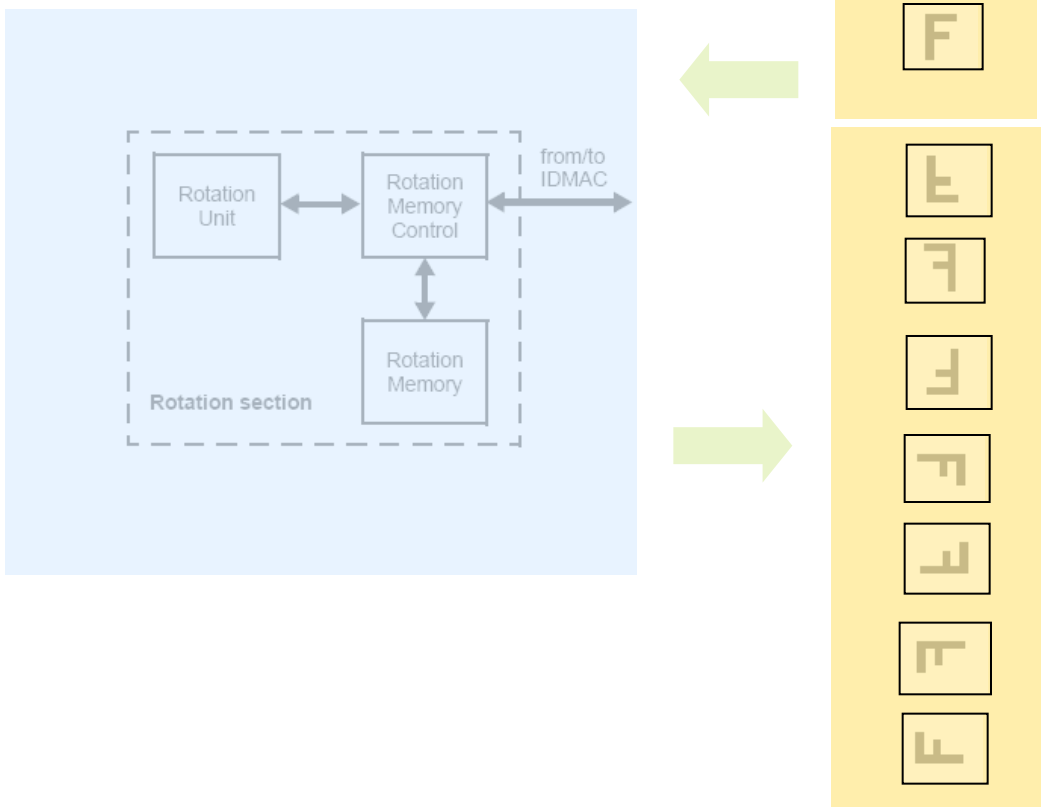


- Resizing
  - Fully flexible resizing ratio

    Maximal downsizing ratio: 8:1
    Maximal upsizing ratio: 1:8192
  - Independent horizontal and vertical resizing ratios
- Color conversion/correction
  - YUV <-> RGB, YUV <-> YUV conversion
- Combining with a graphic plane
- Max output width 1024 pixels. Larger images are processed in stripes

The Image Rotator - IRT

- Role: performs rotation and inversion
  - Rotation: 90, 180, 270 degrees
  - Inversion: horizontal and vertical
- Rate: up to 100M pixels/sec
  - (depends on use case)
- Additional features
  - Acts on 8x8 blocks
  - Multi-tasking: up to three tightly time-shared tasks – block-by-block
  - Pixel format: 24-bit



The Video De-Interlacer or combiner - VDIC

- Role 1: performs de-interlacing – converting interlaced video to progressive

- Method: a high-quality motion adaptive filter
  - For slow motion – retains the full resolution (of both top and bottom fields), by using temporal interpolation
  - For fast motion – prevents motion artifacts, by using vertical interpolation

- Resolution: field size up to 968x1024 for i.MX6 and 720x1024 in i.MX5 pixels. Larger frames are processed in stripes (split mode).
- Output rate: up to 120M pixels/sec

- Additional features
  - Uses three input fields for each output frame (the minimum needed for a reliable motion detection)
  - Vertical interpolation – 4-tap filter; using an internal row buffer
  - Single concurrent flow
  - Input may come from a video decoder (VPU) or directly from the CSI

The Video De-Interlacer or combiner - VDIC

- Role 2 : performs combining – overlaying of 2 frames at the same color space

- As an alternative to the de interlacing function the VDIC HW can perform combining

  - Combining of 2 planes
  - Doesn’t have to be of the same size
  - Must be of the same color space (no CSC)
  - Perform 1 pixel per cycle
  - Color keying, alpha blending

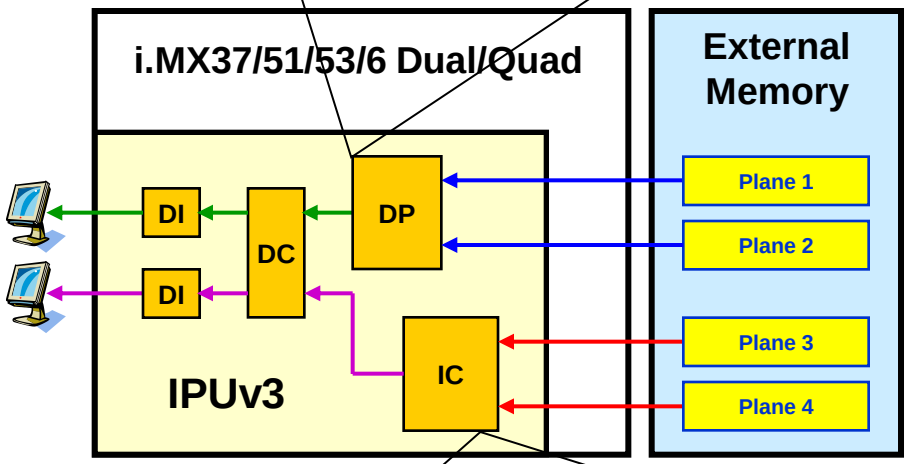
IPUv3H – Combining

IPUv3 – Basic Combining Capabilities

Combining in the Display Processor (DP)
Two planes
- One plane may have any size and location
- The other one must be “full-screen” (cover the full output area)

Maximal rate: i.MX37/51 – 133 MP/sec, i.MX53 – 200 MP/sec, i.MX6 Dual/Quad – 264 MP/sec

- Combining methods (in both cases)
  - Color keying and/or alpha blending
  - Alpha: global or per-pixel; interleaved with the pixels (upper plane) or as a separate input



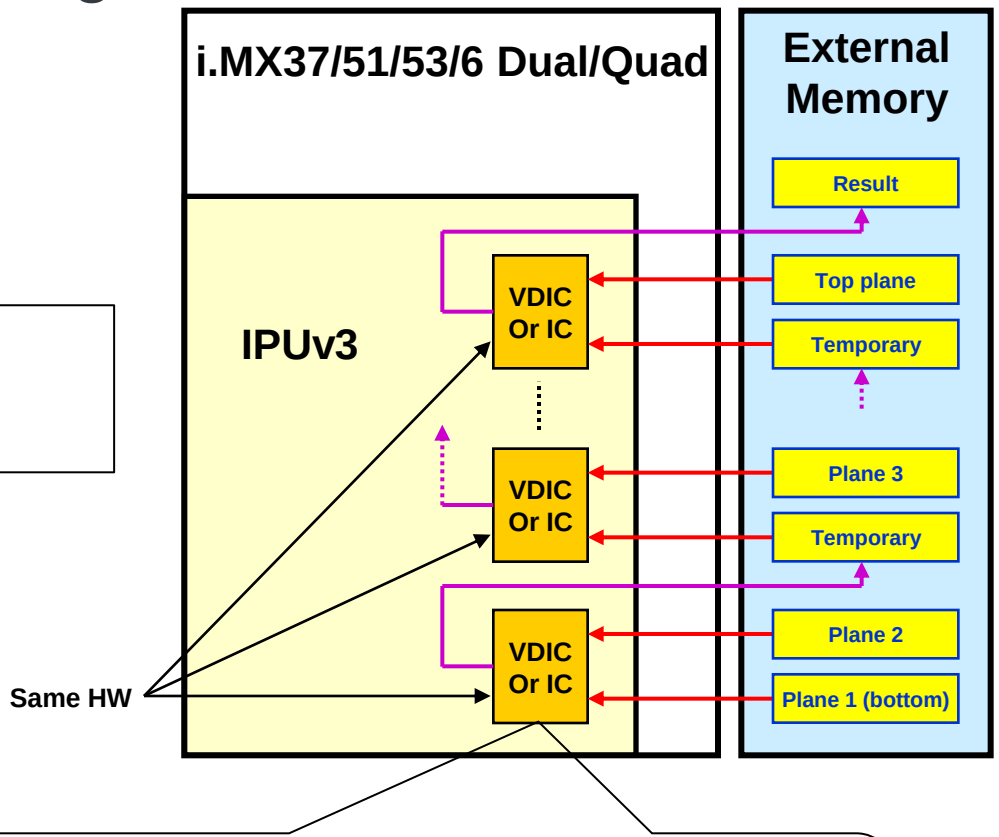
***Note**: This is the capability per IPUs, so the total capability of the processor is doubled in i.MX6DQ.*

Combining in the Image Converter (IC)
Two planes; both “full-screen” (cover the full output area)
Maximal rate: i.MX37/51 – 20 MP/sec , i.MX53 – 30 MP/sec, i.MX6 Dual/Quad – 40 MP/sec

IPUv3 – Off-Line Combining

- Unlimited number of planes combined sequentially

***Note**: This is the capability per IPUs, so the total capability of the processor is doubled in i.MX6DQ.*

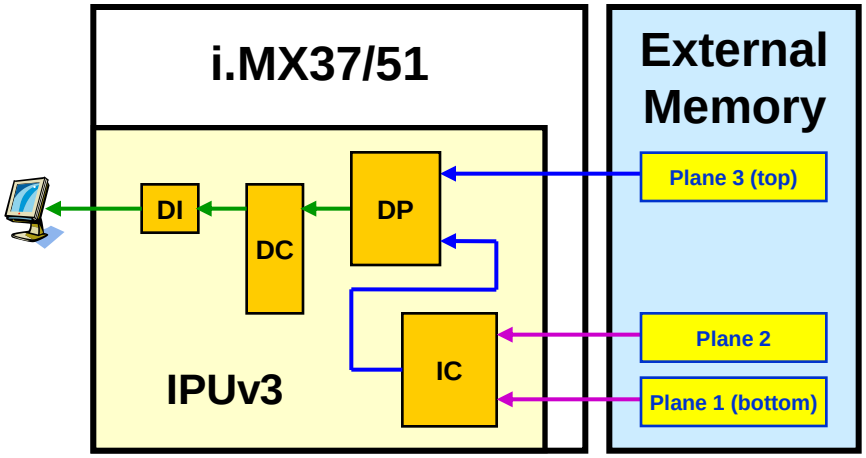


Combining in the VDIC (Video De-Interlacer & Combiner) – i.MX53/6 Dual/Quad only

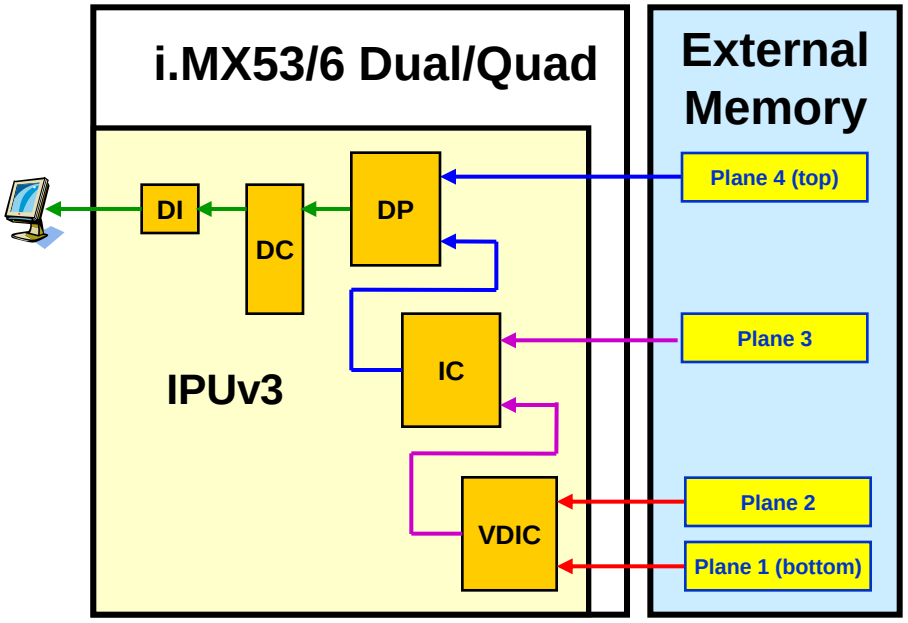
- Available when de-interlacing is not needed
- Two planes; each may have any size and location (supplemented by a “background color”)
- Maximal rate: i.MX53 – 180 MP/sec, i.MX6 Dual/Quad – 240 MP/sec
- Combining method – as in the DP and IC

IPUv3 – Maximal On-The-Fly Combining To A Single Display

3-planes
i.MX37/51 – up to 20 MP/sec



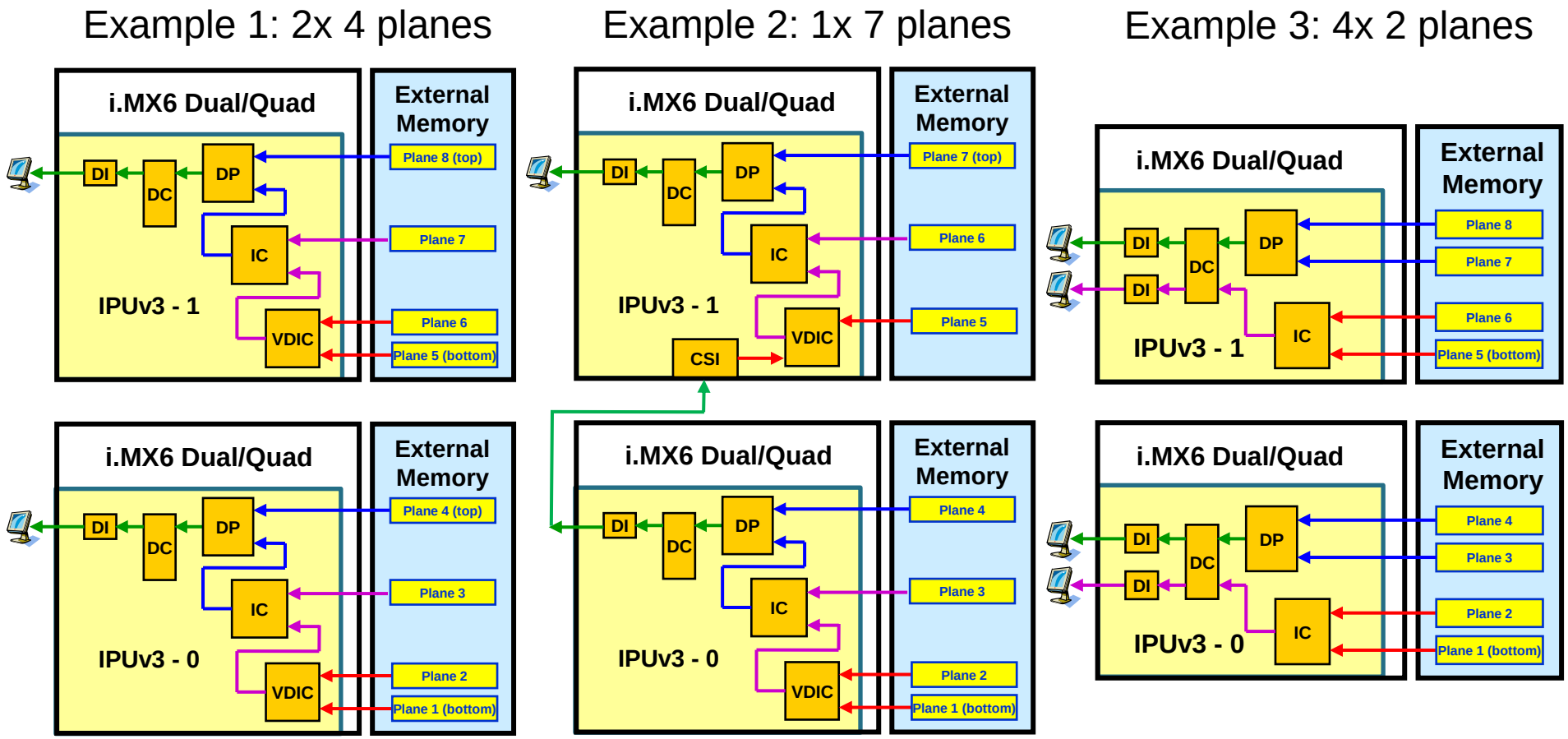
4-planes
i.MX53 – up to 30 MP/sec
i.MX6 Dual/Quad – up to 40 MP/sec



Note: the bottom plane may be a result of additional off-line combining of several planes

***Note**: This is the capability per IPUs, so the total capability of the processor is doubled in i.MX6DQ.*

i.MX6 Dual/Quad: On-The-Fly Combining Using 2x IPUv3



Note: *Some planes may be a result of additional off-line combining of several planes. Such combining may be performed either with IPUs or GPUs.*

IPUv3 combining capabilities – summary

	DP	IC	VDIC
Output	Display	Memory*	Memory
Relations between planes	PlaneA <= PlaneB	PlaneA = PlaneB	PlaneA <= PlaneB
Color space conversion	Yes	Yes	No
Performance	1 cycle/pixel	4 cycle/pixel	1 cycle/pixel
HW cursor	32x32 unified color	No	No
Output Image size	FW up to 2048	FW up to 1024	FW up to 1920
Color keying	Yes		
Alpha blending	Yes		

- *The output of the IC can be sent directly to a smart display

IPUv3 Fundamentals – programming steps

This is a high level example of IPUv3 programming flow.

1. What are the displays connected in the use case?
   a. Allocated the displays to each DI
   b. Define the timing characteristics of each signal for each display.
2. Define each flow in the DC
   a. sync/async
   b. Define the events that trigger the flow – and what to do upon their arrival
   c. Allocate mapping unit, and mapping scheme
   d. Allocate waveform generator in the DI
3. Configure the DP for each flow
4. Configure the IDMAC
   a. How the data is arranged in the memory (interleaved/not interleaved)
   b. What's the data's format (PFS, BPP) , mapping
5. Processing: VDIC, IC (Resizing, CSC) and rotation settings
6. Control module configuration for activation of a flow
   a. Define the trigger to start a flow
   b. Define is the processing chain

IPUv3H – Control

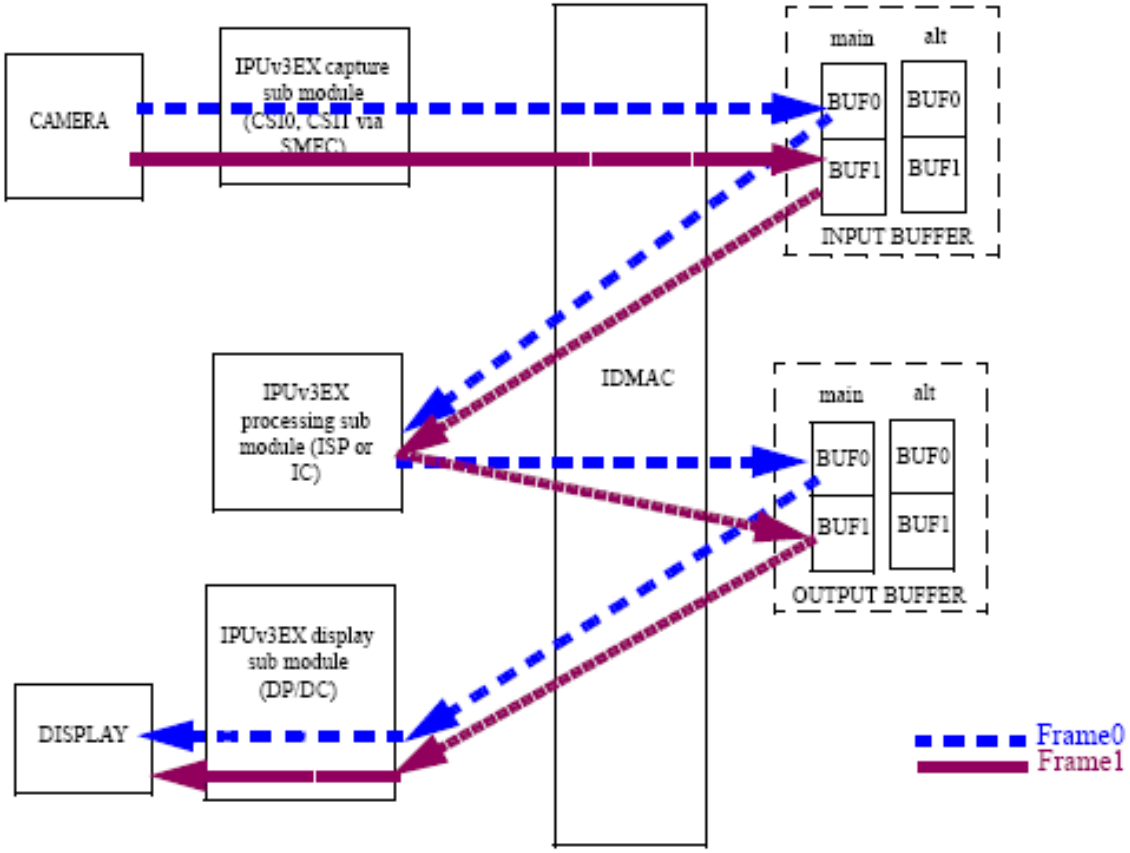
The control Module - CM

- The control module is responsible for the flow management within the IPU.
- The module is composed of

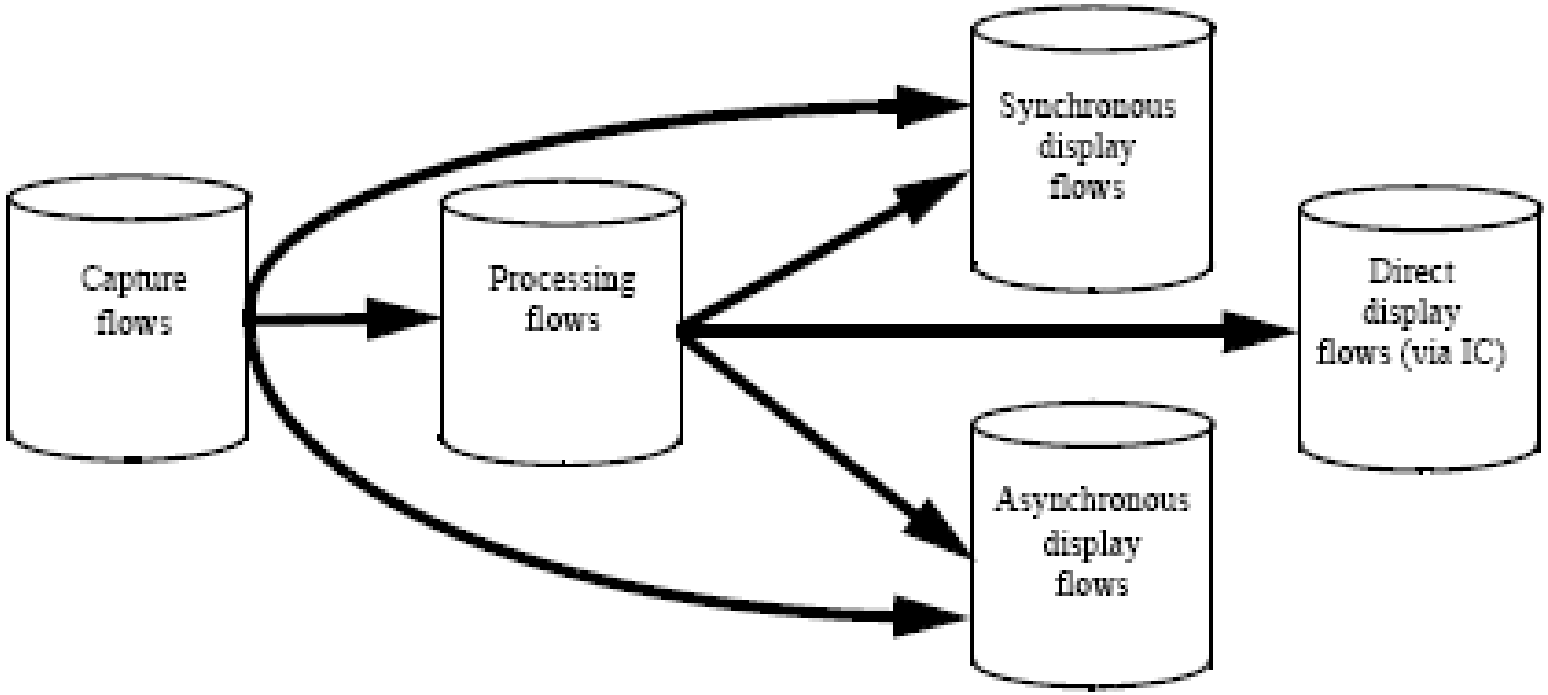
  - General control Registers (GCR)
  - Frame synchronization unit (FSU)
  - Shadow registers module (SRM)
  - Interrupt controller
  - Low Power modes controller
  - Debug unit

FSU – double buffering

- Similar to IPUv1 the data is tightly pipelined using double buffering



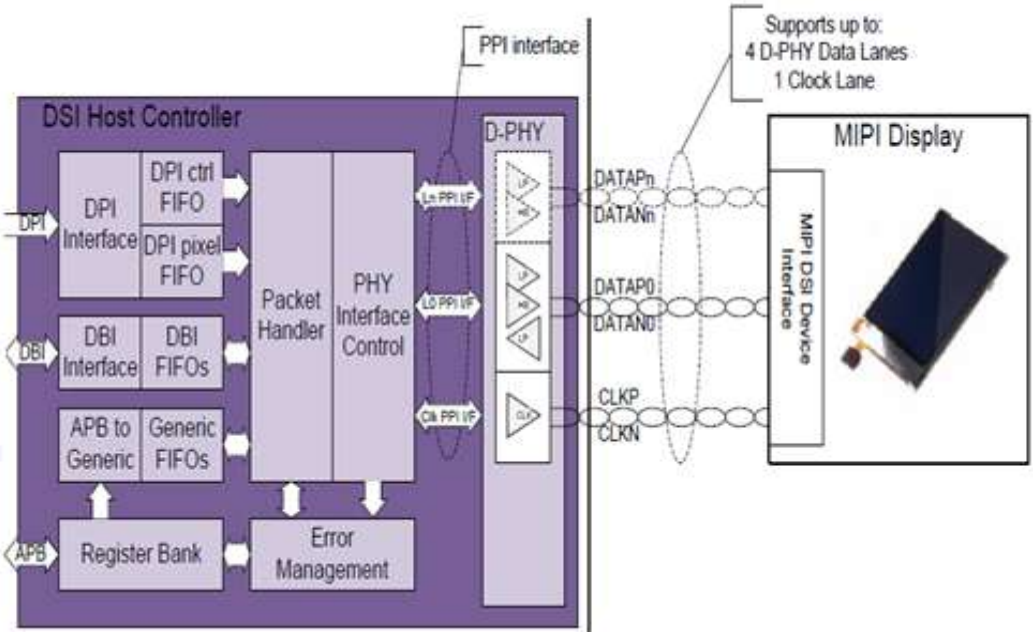
FSU – task chaining



Peripherals

MIPI DSI

The DSI MIPI Interface is a digital core accompanied with a multi-lane D-PHY that implements all protocol functions defined in the MIPI DSI Specification, providing an interface between the System and MIPI DSI compliant Display



Features of the MIPI DSI complex:
Supported standard version:

- MIPI DSI Compliant
- DSI Version 1.01
- DPI Version 2.0
- DBI Version 2.0
- DSC Version 1.02
- PPI for D-PHY
- MIPI D-PHY Version 1.0

Configuration: one clock lane, two data lanes

Speed: Up to 1Gb/s per lane (fast speed). Low speed/low power signaling supported

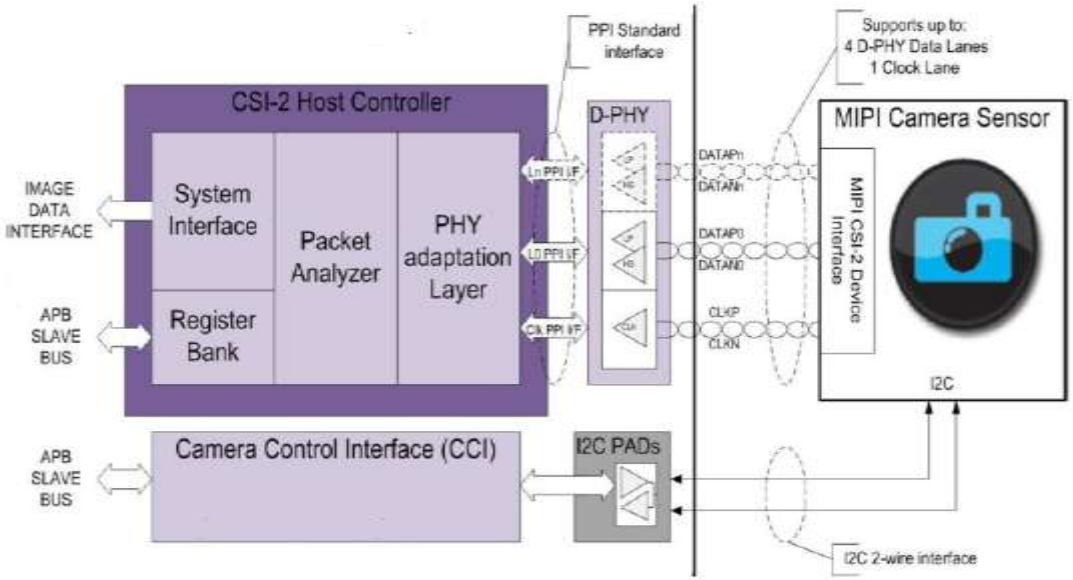
DSI can support both command and video modes and up to four virtual channels to accommodate multiple displays.

- Command and video mode support (type 1, 2, 3, and 4 display architecture)
- Mode switching: low power and ultra low power
- Burst mode/Non-burst mode
- Bus turnaround
- Fault error recovery scheme

Both DPI and DBI coexist in the system but only one of them could be active in a certain time

MIPI CSI-2

The CSI-2 MIPI Interface is a digital core accompanied with multi-lane D-PHY that implements all protocol functions defined in the MIPI CSI-2 Specification, providing an interface between the System and MIPI CSI-2 compliant Camera Sensor



The features of the MIPI CSI-2 complex:

Supported standard version: MIPI CSI-2 Version 1.0
Configuration: one clock lane, four data lanes
Speed: Up to 1Gb/s per lane
Throughput: 250MB/sec

- Timing accurate signaling of Frame and Line synchronization packets;
- Support for several frame formats such as:
  - General Frame or Digital Interlaced Video with or without accurate sync timing
  - Data type (Packet or Frame level) and Virtual Channel interleaving
- 32-bit Image Data Interface delivering data formatted as recommended in CSI-2 Specification;
  - Directly supports all primary data formats conversion to IPU input. Some secondary formats are treated as “generic” data
  - RGB, YUV and RAW color space definitions;
  - From 24-bit down to 6-bit per pixel;
  - Generic or user-defined byte-based data types

LVDS Interface in i.MX53 & i.MX6 D/Q – Key Features

- Structure
  - Two Channels
  - Each channel contains 4 data pairs + 1 clock pair
- Data
  - 18 bpp pixels – using 3 LVDS data pairs
  - 24 bpp pixels – using 4 LVDS data pairs
- Control signals: HSYNC, VSYNC, DE

- Pixel clock rate
  - Single Channel: up to 85 MHz; e.g. WXGA @ 60 fps or 720p60
  - Dual Channel: up to 170 MHz; e.g. UXGA @ 60 Hz or 1080p60
- Relevant Standards
  - PHY Standard: ANSI EIA-644A
  - Display Protocol Standards:
    - SPWG Standard Panel Working Group Specification 3.8 (May 2007)
    - VESA PSWG – Panel Standardization Working Group – set of standards for panels using LVDS.
    - JEIDA/JEITA DISM Standard JEIDA-59-1999
    - OpenLDI (National) – Revision 0.95 13/May/1999. *Only* Unbalanced operating mode supported (aligned with vast majority of LCD vendors).

LVDS – what is supported?

- Single Channel configuration
  - Pixel clock: up to 85 MHz; e.g. WXGA @ 60 fps or 720p60
  - LVDS Clock frequency = Pixel clock x 7 = 85*7 = 595Mhz
  - Data :
    - 18 bpp pixels – using 3 LVDS data pairs
    - 24 bpp pixels – using 4 LVDS data pairs
- Dual Channel configuration
  - Pixel clock: up to 170 MHz; e.g. UXGA @ 60 Hz or 1080p60
  - LVDS Clock frequency = Pixel clock x 7/2 = 170*7/2 = 595Mhz
  - Data :
    - 18 bpp pixels – using 3 LVDS data pairs per channel
    - 24 bpp pixels – using 4 LVDS data pairs per channel

LDB Features

- **LDB Structure**:
  - 2 Channels , same/independent data
  - Each channel contains 4 data pairs + 1 clock pair
- **Resolutions/Rates**:
  - Single Channel (up to WXGA): Up to 85 MHz, 3 or 4 data pairs
  - Dual Channel (up to UXGA): Up to 170 MHz, 6 or 8 data pairs
  - For example: can support 1080p60 or UXGA @60fps
- **Pixel Depths**:
  - 18 bpp – 3 LVDS data pairs
  - 24 bpp – 4 LVDS data pairs
- **Control signals:**
  - Supports HSYNC, VSYNC, DE

HDMI General Features

- **Description**: High-Definition Multimedia Interface (HDMI) Transmitter including both HDMI TX Controller and PHY
- **Standard Compliance**: HDMI 1.4a, DVI 1.0, HDCP 1.4 (with keys stored in embedded eFuses)
  - Supporting majority of primary 3D Video formats
- **TMDS Core Frequency**: From 25 MHz to 340 MHz
- **Consumer Electronic Control**: Supported
- **Monitor Detection**: Hot plug/unplug detection and link status monitor support
- **Testing Capabilities**: Integrated test module
- **Maximal Power Consumption**: 70mW
- **Temperature Range**: -40C to +125C (Tj)

i.MX6 Dual/Quad: HDMI Video/Audio Features

- **Video Standard Compliance**: EIA/CEA-861D
- **Supported Video Resolutions**: Up to 1080p@60Hz and 720p/1080i@120Hz HDTV display; up to QXGA graphics display
- **Pixel Clock Frequency**: From 25 MHz to 240 MHz
- **Video Data Formats**: YCbCr 4:4:4; RGB 4:4:4; YCbCr 4:2:2
- **Internal Video Processing**: Interpolation YCbCr 4:2:2 to 4:4:4; conversion YCbCr to RGB and vice versa
- **Audio Standard Compliance**: IEC60958, IEC61937
- **Supported Audio Formats**: All audio formats as specified by the HDMI Specification Version 1.4a
- **Audio Input Interfaces**: Embedded Audio DMA
- **Audio Sampling Rate**: Up to 192 kHz

IPU & the iMX Linux BSP

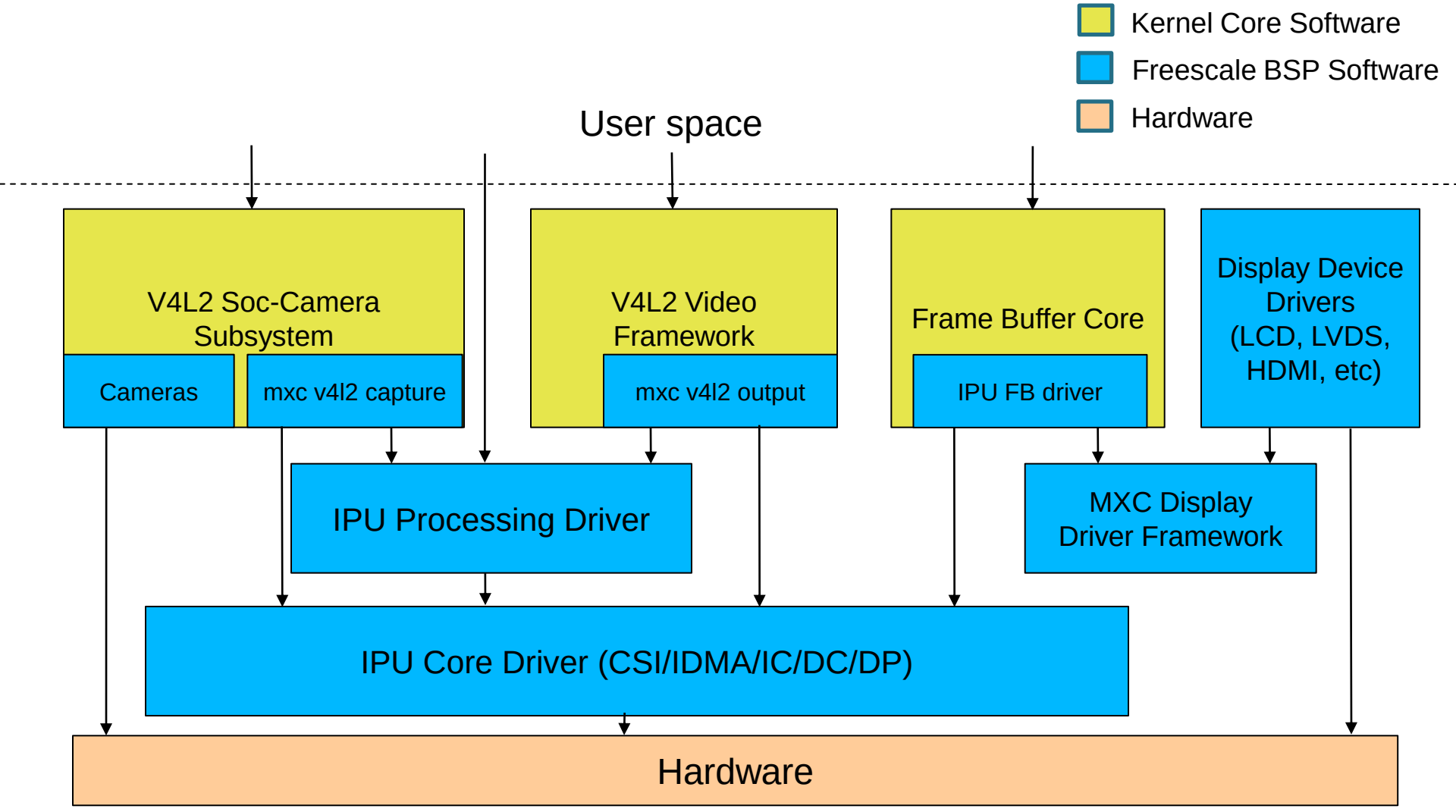
IPU Drivers

- IPU drivers are based on code re-used from MX5x IPU driver
  - Key modification: Support for multiple instances provides support for the 2 IPU modules in i.MX 6Quad/Dual
- IPU functionality accessed through multiple interfaces:
  - **IPU framebuffer (FB) driver**: Accessed through the Linux standard FB interface
    - Introduction of MXC Display Driver framework, to manage interaction between IPU and display device drivers (e.g., LCD, LVDS, HDMI, MIPI, etc.)
  - **IPU processing driver**: A custom API exposes IPU processing functionality
    - Resizing
    - Rotation
    - Combining of graphics planes
    - CSC
    - De-interlacing
  - **Video 4 Linux 2 (V4L2) output driver**: Based on V4L2 video API, leverages IPU processing driver
  - **V4L2 capture driver**: Based on V4L2 capture API; leverages IPU processing driver and IPU core driver

IPU Drivers

- **Architecture Design:**
  - IPU sub-module (CSI, IC, DI, DP, IDMAC, etc) functionality provided in a set of IPU Core driver functions
    - Largely unmodified between MX5x and MX 6Quad
  - Leverage existing Linux APIs:
    - For framebuffer access (IPU FB driver)
    - Video image processing and display (V4L2 output driver)
    - Image capture (V4L2 capture driver)
  - Fill in gaps with custom APIs and API extensions:
    - Extensions to FB interface to control certain IPU functionality – local and global alpha, gamma correction, etc.
    - IPU Processing driver to provide user space access to IPU processing capabilities
  - Provide MXC Display Driver (`mxc_dispdrv.h`) framework to simplify connection between display devices and IPU modules.

IPU Drivers


Kernel Core Software
Freescale BSP Software
Hardware
User space
V4L2 Soc-Camera Subsystem
Cameras
mxc v4l2 capture
V4L2 Video Framework
mxc v4l2 output
Frame Buffer Core
IPU FB driver
Display Device Drivers (LCD, LVDS, HDMI, etc)
IPU Processing Driver
MXC Display Driver Framework
IPU Core Driver (CSI/IDMA/IC/DC/DP)
Hardware

Overview of IPU Drivers

- MXC Display Driver
  - Simple framework to manage MXC display device drivers.
  - Examples: LCD, TVE, MIPI, VGA, HDMI
- IPU Processing Driver
  - Manage IPU IC tasks in kernel space.
- MXC V4L2 Drivers
  - Based on IPU processing driver.

MXC Display Driver - Files

- MXC Display Driver files
  - drivers/video/mxc/mxc_dispdrv.h
  - drivers/video/mxc/mxc_dispdrv.c
- IPU framebuffer driver
  - drivers/video/mxc/mxc_ipuv3_fb.c
- Display device drivers
  - drivers/video/mxc/mxc_lcdif.c
  - drivers/video/mxc/mxc_hdmi.c
  - drivers/video/mxc/mipi_dsi.c
  - …

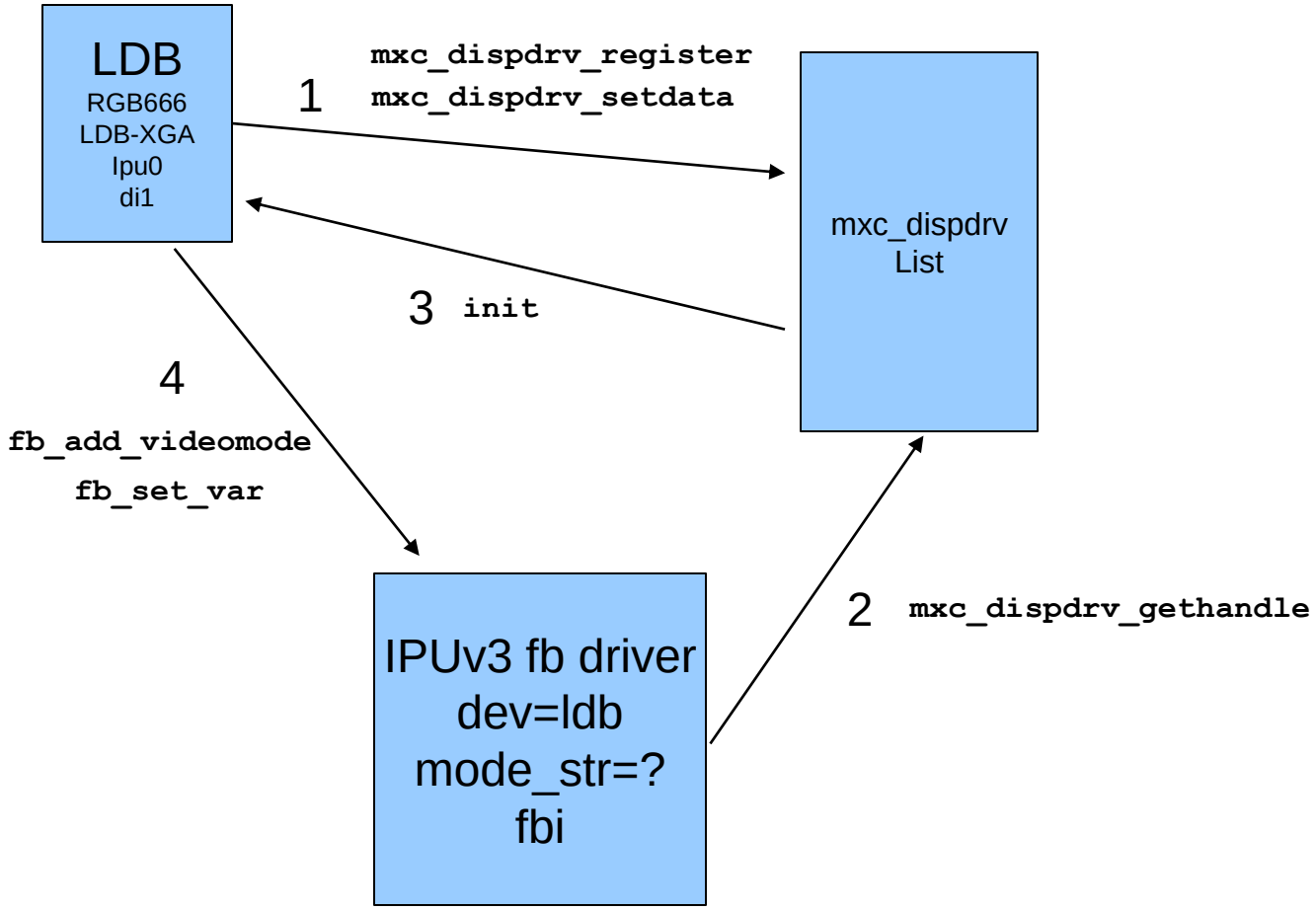
MXC Display Driver - Structures

```
struct mxc_dispdrv_driver {
      const char *name;
      int (*init) (struct mxc_dispdrv_handle *, struct mxc_dispdrv_setting *);
      void (*deinit) (struct mxc_dispdrv_handle *);
      /* display driver enable function for extension */
      int (*enable) (struct mxc_dispdrv_handle *, struct fb_info *);
      /* display driver disable function, called at early part of fb_blank */
      void (*disable) (struct mxc_dispdrv_handle *, struct fb_info *);
      /* display driver setup function, called at early part of fb_set_par */
      int (*setup) (struct mxc_dispdrv_handle *, struct fb_info *fbi);
};

struct mxc_dispdrv_setting {
       /*input-feedback parameter*/
        struct fb_info *fbi;
        int if_fmt;
        int default_bpp;
        char *dft_mode_str;
        /*feedback parameter*/
        int dev_id;
        int disp_id;
};
```

MXC Display Driver - Functions

```
struct mxc_dispdrv_entry *mxc_dispdrv_register(
                struct mxc_dispdrv_driver *drv);

int mxc_dispdrv_unregister(struct mxc_dispdrv_entry *entry);

struct mxc_dispdrv_handle *mxc_dispdrv_gethandle(char *name,
                struct mxc_dispdrv_setting *setting);

int mxc_dispdrv_setdata(struct mxc_dispdrv_entry *entry,
                void *data);

void *mxc_dispdrv_getdata(struct mxc_dispdrv_entry *entry);
```

MXC Display Driver – Configuration Flow



MXC Display Driver

- Command Line Options:

first display:

```
video=mxcfb0:dev=dispdrv_name,mode_str,if=if_fmt
```

second display:

```
video=mxcfb1:dev=dispdrv_name,mode_str,if=if_fmt
```

For example:

```
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
video=mxcfb2:dev=lcd,800x480M@55,if=RGB565
```

MXC Display Driver – Multi-Display Options

hdmi + lvds

```
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
```

lvds + lvds

```
video=mxcfb0:dev=ldb,LDB-XGA,if=RGB666
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
```

lcd + lvds

```
video=mxcfb0:dev=lcd,800x480M@55,if=RGB565
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
```

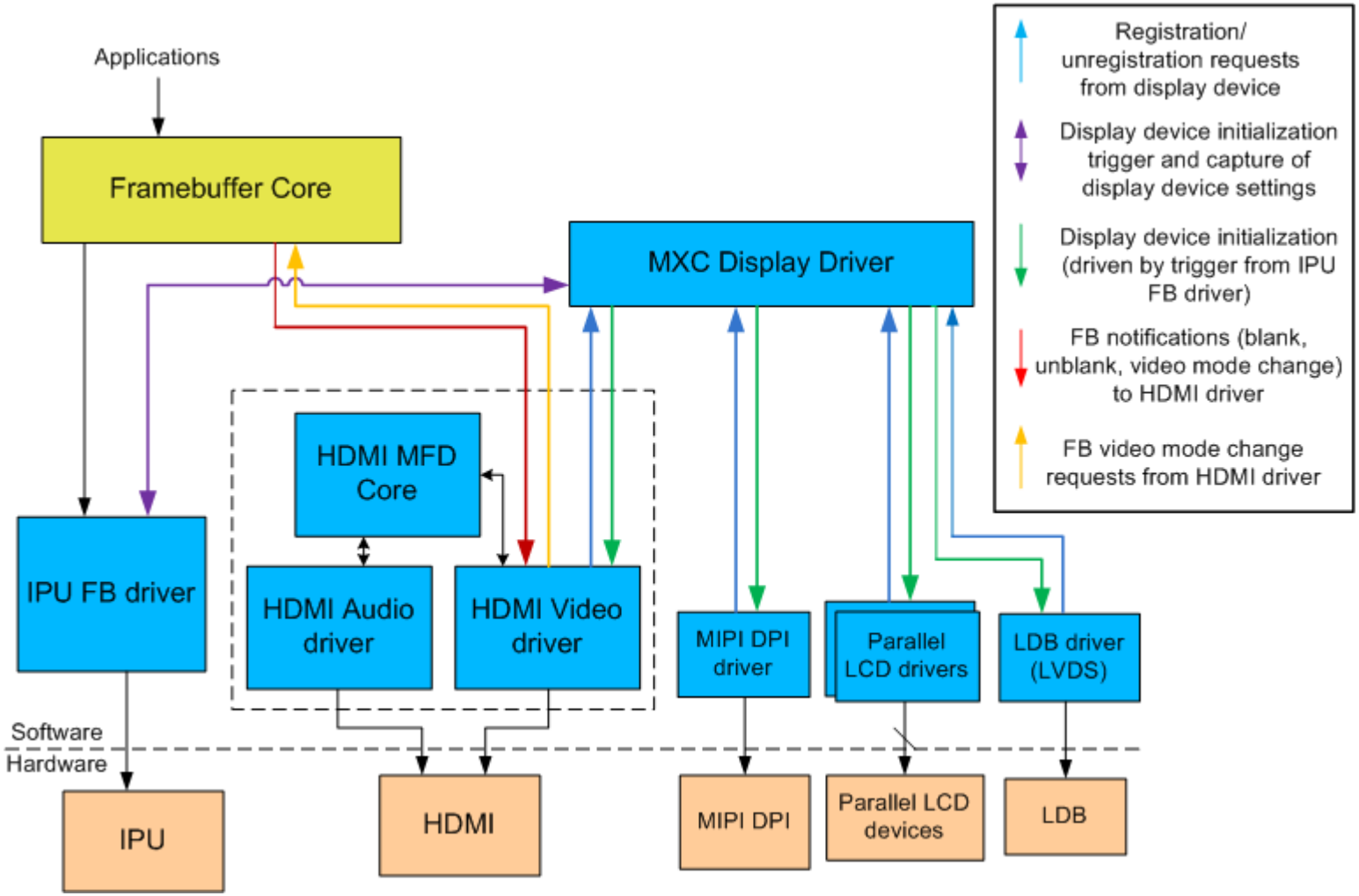
hdmi + lvds + lvds

```
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
video=mxcfb2:dev=ldb,LDB-XGA,if=RGB666
```

Example of MXC Display Driver - HDMI

- Software Architecture:
  - HDMI multifunction driver (MFD) manages software resources common to video and audio drivers
  - Audio driver uses ALSA/SoC audio framework.
  - Video driver:
    - MXC Display Driver API to register with IPU FB driver
    - Linux Framebuffer (FB) API to change the video mode and receive notifications of mode changes

...ll and the i.MX 6Quad Framebuffer and Display Device Architecture



Overview of IPU Drivers

- MXC Display Driver
  - Simple framework to manage MXC display device drivers.
  - Examples: LCD, TVE, MIPI, VGA, HDMI
- IPU Processing Driver
  - Manage IPU IC tasks in kernel space.
- MXC V4L2 Drivers
  - Based on IPU processing driver.

IPU Processing Driver - Introduction

- Each IPU has two kernel threads for IC task PP&VF
- Each kernel thread performs the tasks on its task queue list
- Each task executes the following sequence:
  `ipu_init_channel → ipu_init_channel_buffer → request_ipu_irq → ipu_enable_channel → wait_irq (task finish) → ipu_disable_channel → ipu_uninit_channel`
- Tasks are based on single buffer mode
- Split mode tasks will be split into 2 tasks per IPU.
- An application only needs to prepare a task and queue it
- Task operations include
  - Setting the task input/overlay/output/rotation/deinterlacing/buffer
  - Call ioctl `IPU_CHECK_TASK` first to adjust parameters according to feedback
  - Call ioctl `IPU_QUEUE_TASK` to queue task
- `IPU_QUEUE_TASK` is a blocking ioctl.

IPU Processing Driver

- Files

  `include/linux/ipu.h`

  `drivers/mxc/ipu3/ipu_device.c`

- Structures

see `include/linux/ipu.h`

- Ioctls

```
#define IPU_CHECK_TASK _IOWR('I', 0x1, struct ipu_task)
#define IPU_QUEUE_TASK _IOW('I', 0x2, struct ipu_task)
#define IPU_ALLOC      _IOWR('I', 0x3, int)
#define IPU_FREE       _IOW('I', 0x4, int)
```

IPU Processing Driver

- Example

`linux-test/test/mxc_ipudev_test/mxc_ipudev_test.c`

IPU Processing Driver

- Advantages
  - Easy to use.
  - Provides workaround for IPU suspend/resume issue
    - Cannot suspend when double buffering is enabled
  - All IC tasks may be based on this IPU processing driver, including
  - user applications and V4L2 output and capture drivers
    - Reason: Easier to debug if there is an issue.

- Disadvantages
  - Based on kernel thread, all control is done by software, so Linux scheduler may have undesirable impact on the system.
  - Single-buffer mode doesn't perform as well as double-buffer mode

Overview of IPU Drivers

- MXC Display Driver
  - Simple framework to manage MXC display device drivers.
  - Examples: LCD, TVE, MIPI, VGA, HDMI

- IPU Processing Driver
  - Manage IPU IC tasks in kernel space.

- MXC V4L2 Drivers
  - Based on IPU processing driver.

V4L2 – Common Kernel API

- What is Video4Linux (V4L)?
  - V4L is the original video capture/overlay API of the Linux kernel. It appeared late the 2.1.x development cycle in the Linux kernel.
- What about V4L2?
  - V4L2 is the second generation of the video4linux API which fixes a number of design bugs of the first version. It was integrated into the standard kernel in 2.5.x.
  - V4L2 is an interface for analog radio, video capture and output drivers.
  - Hardware acceleration capabilities (IPUv3) are leveraged in V4L2 drivers and provided in the Linux BSP
  - Upper level software that uses the V4L2 API, such as G-streamer source/sink and Android camera HAL, does not need to understand the underlying hardware.

- Documentation / Web Resource / API spec
  - Documentation/video4linux/ subdirectory in kernel tree.
  - http://v4l2spec.bytesex.org - the spec of V4L2
  - http://www.linuxtv.org/wiki/index.php/Main_Page - wiki for V4L and DVB

MXC V4L2 – User APIs

- VIDIOC_QUERYCAP
- VIDIOC_G_FMT / VIDIOC_S_FMT
- VIDIOC_REQBUFS
- VIDIOC_QUERYBUF
- VIDIOC_QBUF / VIDIOC_DQBUF
- VIDIOC_STREAMON / VIDIOC_STREAMOFF
- VIDIOC_G_CTRL / VIDIOC_S_CTRL
- VIDIOC_CROPCAP / VIDIOC_G_CROP / VIDIOC_S_CROP
- VIDIOC_ENUMOUTPUT / VIDIOC_G_OUTPUT / VIDIOC_S_OUTPUT

APIs used only for MXC V4L2 capture:

- VIDIOC_ENUMINPUT / VIDIOC_G_INPUT / VIDIOC_S_INPUT

APIs used only for MXC V4L2 TV-in:

- VIDIOC_ENUMSTD / VIDIOC_G_STD / VIDIOC_S_STD

MXC V4L2 – Internal APIs

New version of V4L2 framework supports master / slave device drivers:

- Support multiple master devices and multiple slave devices
- `mxc_v4l2_capture` driver is the V4L2 master driver
- Camera drivers and tv-in driver are V4L2 internal slave drivers

These ioctls are used in kernel internally for MXC V4L2 capture/tvin especially:

- `ioctl_dev_init / ioctl_dev_exit`
- `ioctl_s_power`
- `ioctl_g_ifparm`
- `ioctl_init`
- `ioctl_g_fmt_cap`
- `ioctl_g_parm / ioctl_s_parm`
- `ioctl_queryctrl / ioctl_g_ctrl / ioctl_s_ctrl`

V4L2 – Usage and Examples

- How to use V4L2:

  Generally, programming a V4L2 device consists of these steps:
  - Opening the device
  - Changing device properties, selecting a video and audio input, video standard, picture brightness, etc.
  - Negotiating a data format
  - Negotiating an input/output method
  - Executing the actual input/output loop
  - Closing the device

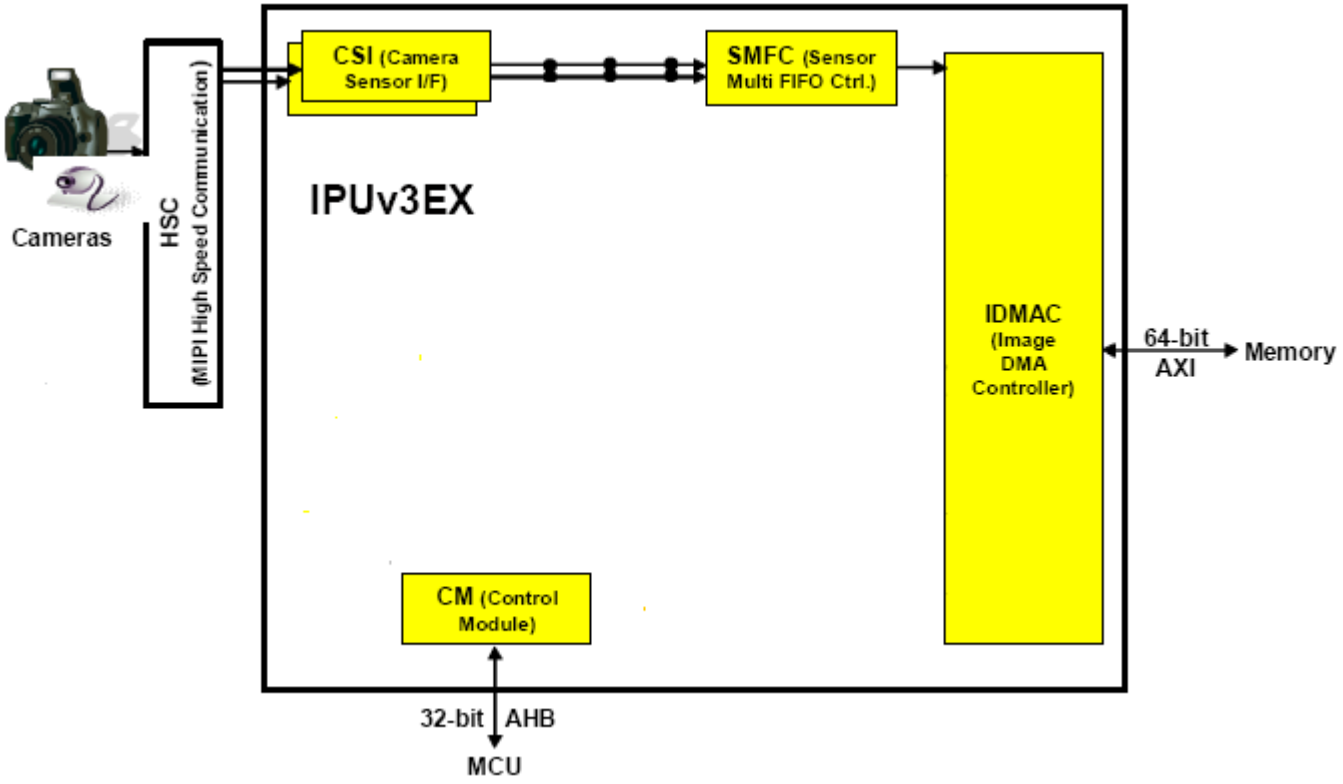
- Please refer to the following examples:
  - BSP team's test cases:
    `git clone git://sw-git01-tx30.am.freescale.net/linux-test.git`
    (without password)
  - Test team's VTE test cases
  - Camera HAL in Android source code
  - G-streamer source/sink source code

Features of MXC V4L2 Capture

- **Still capture:** capture a frame in a buffer, users can read the buffer and store the picture in a file.
- **Preview:** show the capture frames directly onto the framebuffer. Users can choose the framebuffer number on which the video will be shown.
- **Video capture:** capture frames in allocated buffers. Users can get the frames by calling `VIDIOC_DQBUF` and then send them to VPU for encoding or save them in a file.

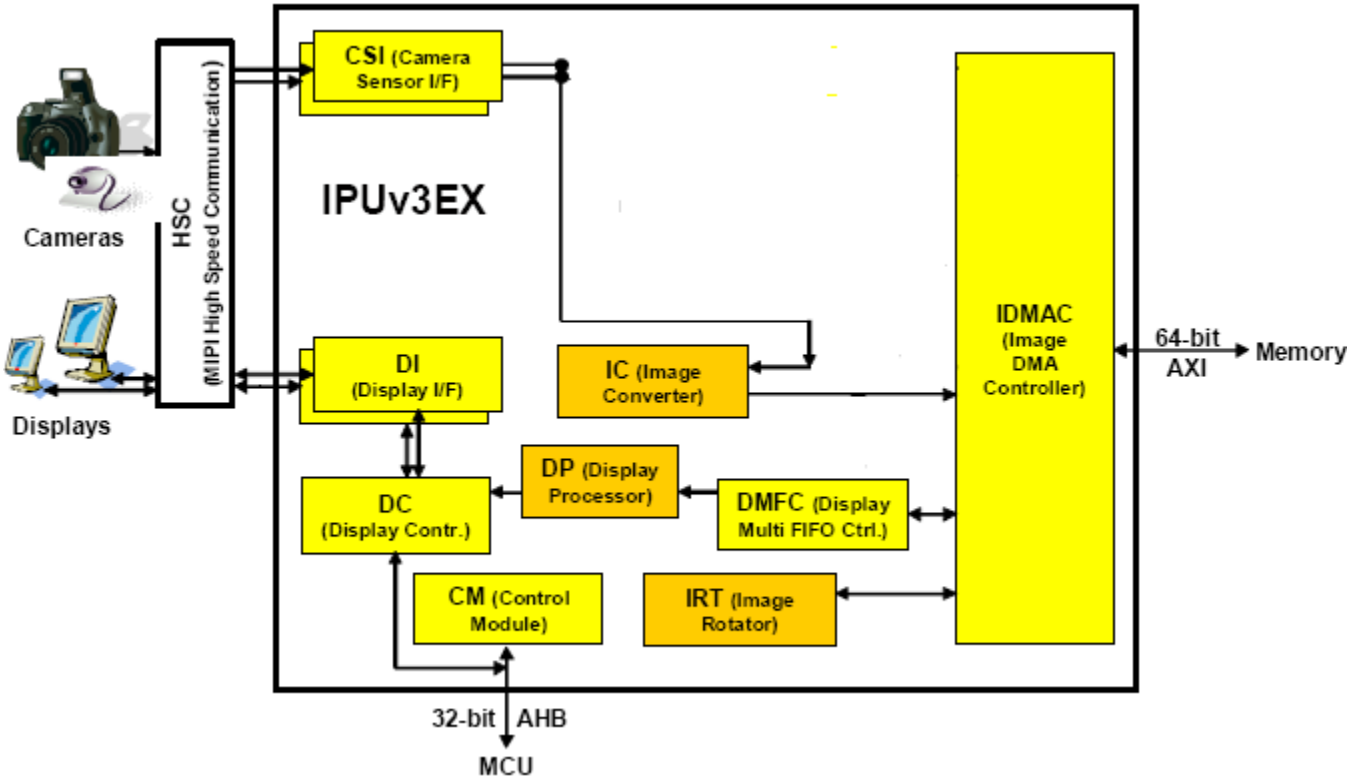
Features of MXC V4L2 Capture

- **To capture a still image**
- No resizing/rotation/CSC can be done.
  - One image can be converted to be in a different pixel format within the same color space with the raw data by using CSI->SMFC->MEM IDMAC channel.



Features of MXC V4L2 Capture

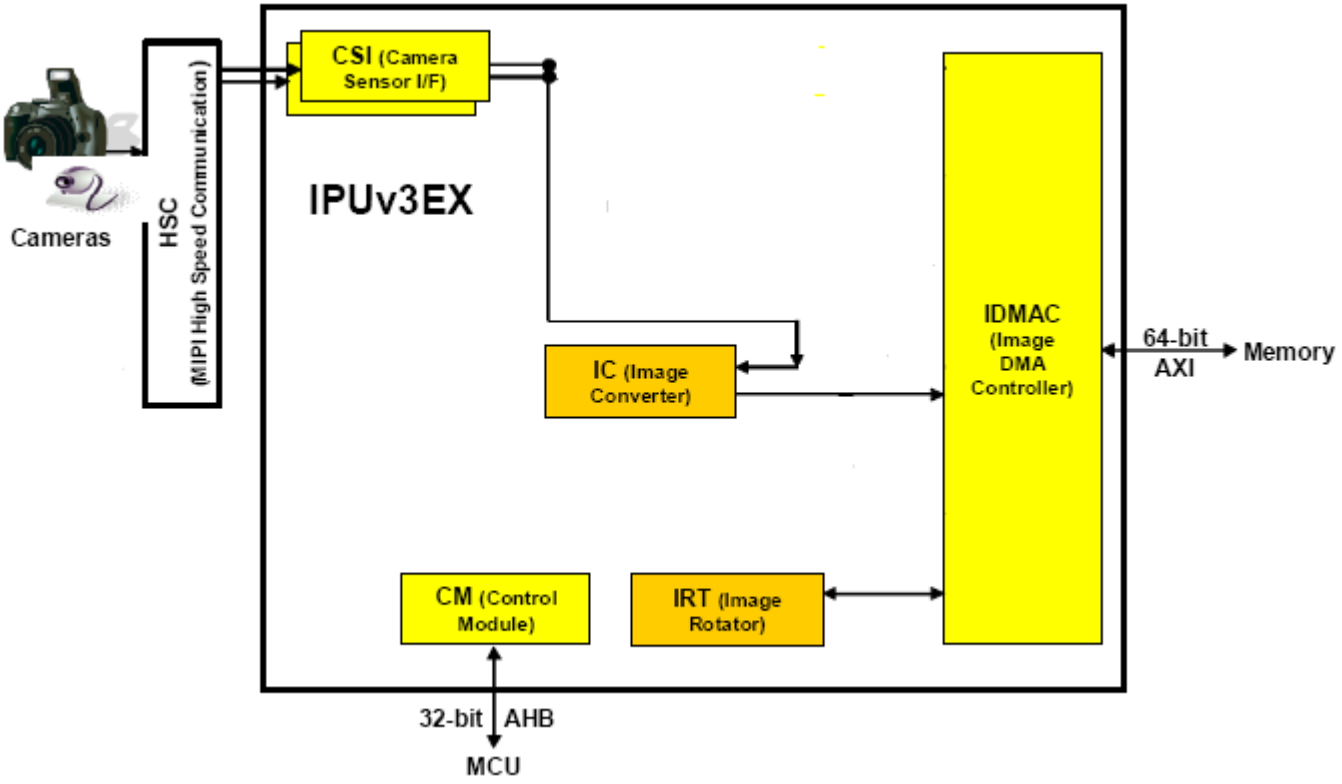
- **To preview a captured video on frame buffer**
  - Preview on fb0 – Resizing/rotation/CSC can be done in IC(PRP_VF) channels. Manually control the buffer ready flags in interrupt handler. Use DP(DP_BG) channel to display the captured frames.
  - Preview on fb2 - Resizing/rotation can be done in IC(PRP_VF) channels. CSC can be done in DP(DP_FG) channel. The flow is totally controlled by FSU.



Features of MXC V4L2 Capture

To capture frames in allocated buffers (using IC channel)

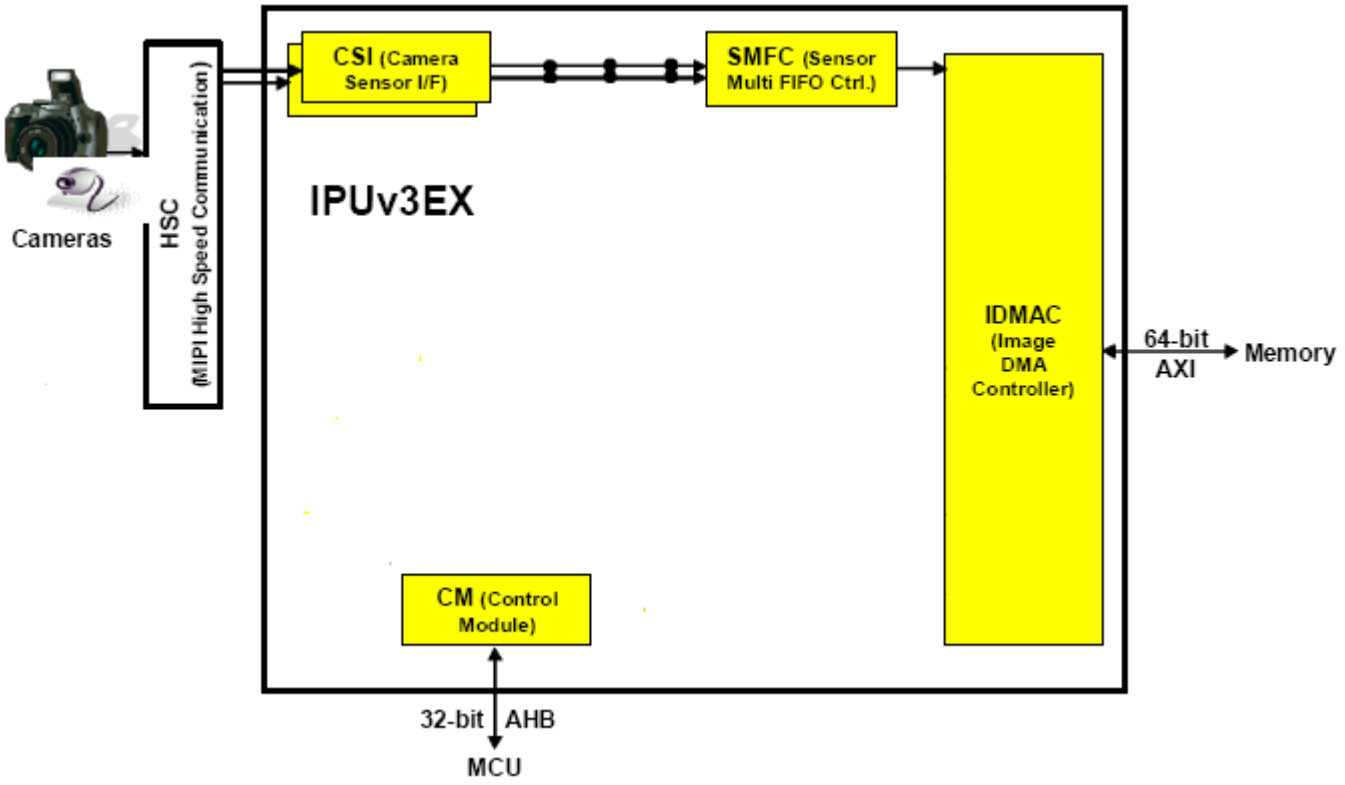
- Resizing/rotation/CSC can be done in IC(PRP_ENC) channels. Manually control the buffer ready flags in interrupt handler.
- MXC V4L2 capture maintains the numbers of buffers.
- Users can get the captured buffer by calling `VIDIOC_DQBUF` ioctl and return the buffer to the kernel by calling `VIDIOC_DQBUF` ioctl .
- NOTE: Camera preview and capturing frames into buffers can be used at the same time.



Features of MXC V4L2 Capture

To capture frames in allocated buffers (using SMFC channel)

- No resizing/rotation/CSC can be done. Manually control the buffer ready flags in interrupt handler.
- MXC V4L2 capture maintains numbers of buffers.
- Users can get the captured buffer by calling `VIDIOC_QBUF` ioctrl and return the buffer to the kernel by calling `VIDIOC_QBUF` ioctrl.



Features of MXC V4L2 Output

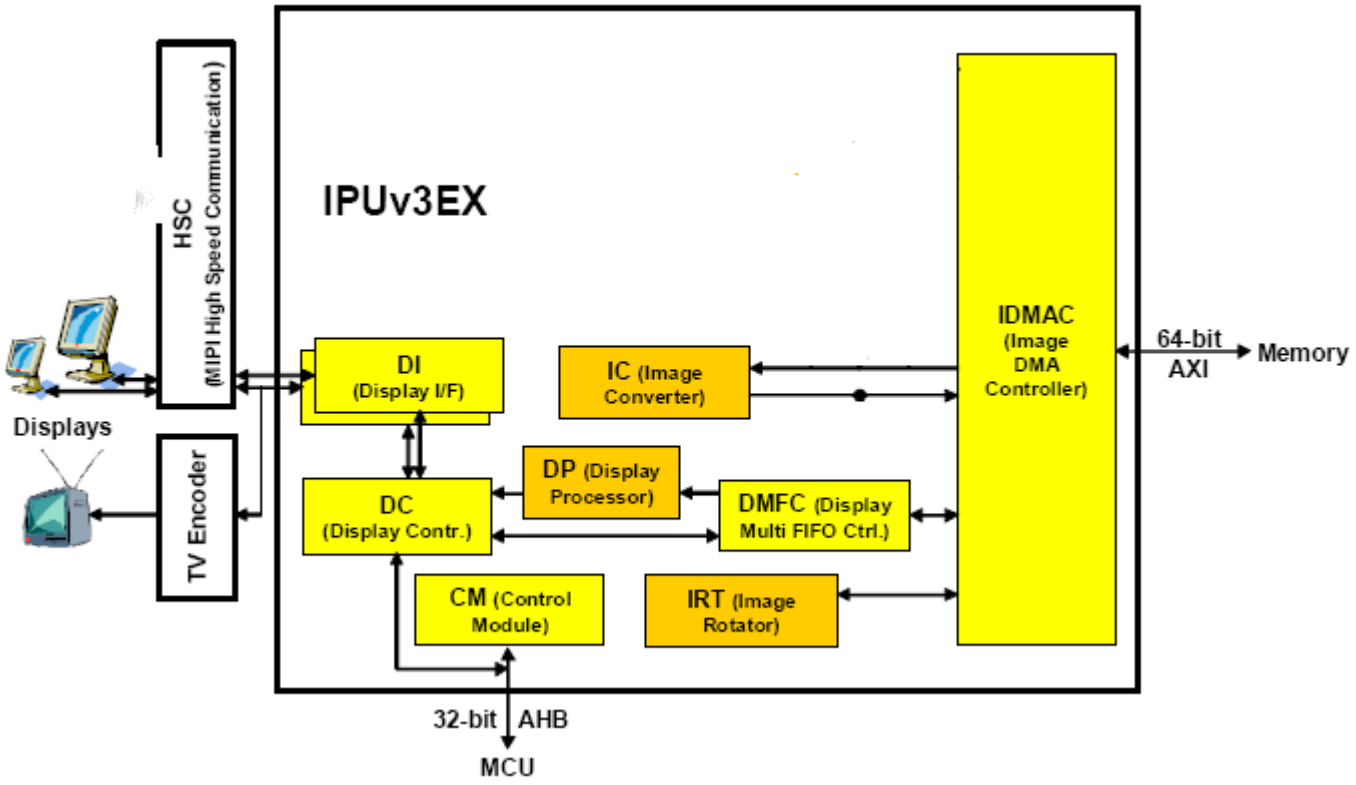
- Support for playing video using one framebuffer at a time:
  - DP-BG framebuffer
  - DP-FG framebuffer
  - DC framebuffer
- Support for the following modes:
  - IC normal mode – resizing / CSC / rotation, using PP channel
  - IC bypass mode – CSC, using DP or DC channel directly
  - IC horizontal/vertical split mode – resizing / CSC / rotation, support high resolution output, using PP channel
  - VDI-IC video deinterlacing mode - deinterlacing / resizing / CSC / rotation, using PRP_VF channel, including high motion mode and low motion mode.

Note: V4L2 output and V4L2 capture can run at the same time if there is no IC or DP/DC channel conflict.

Features of MXC V4L2 Output

IC normal mode (using PP channel)

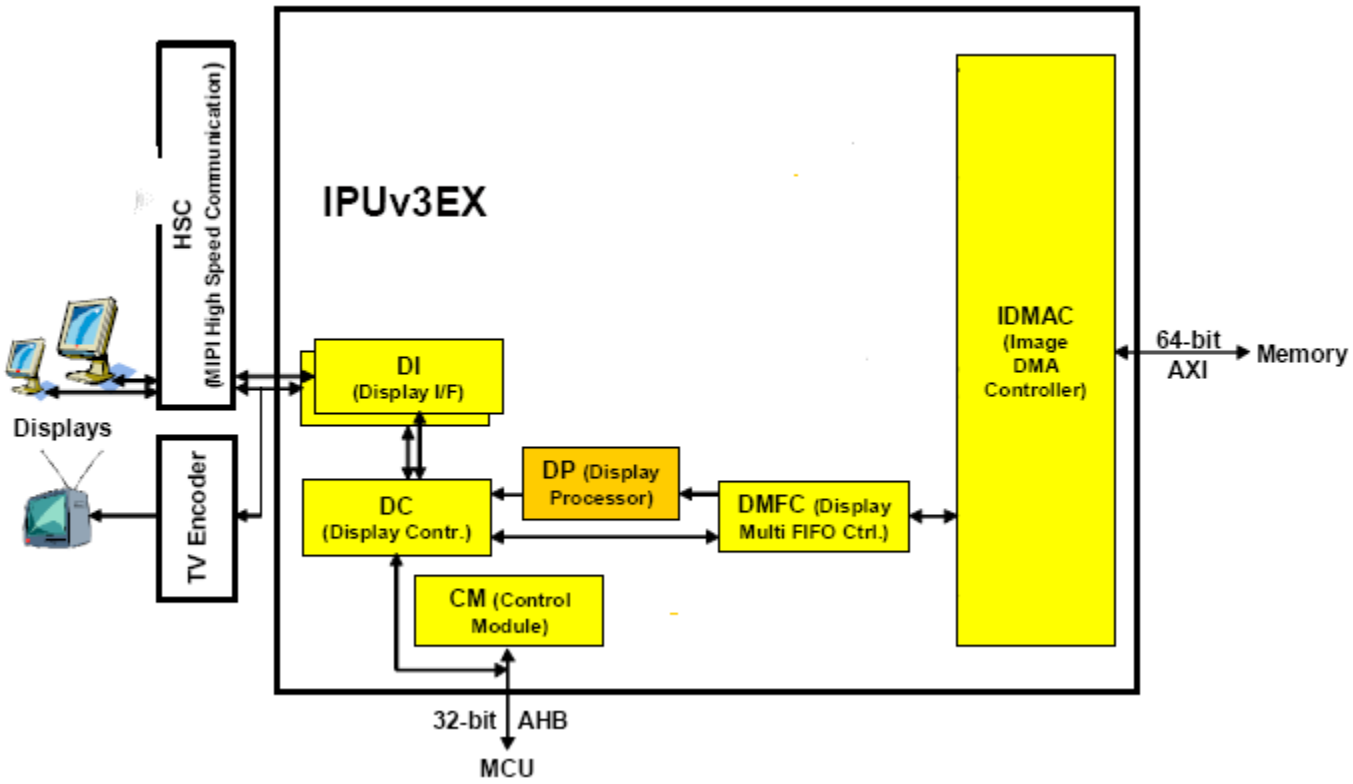
- Resizing/rotation/CSC. Manually control the IC output/display input buffer ready flags in interrupt handler and control IC input buffer ready flags in timer handler.
- MXC V4L2 output maintains numbers of buffers.
- Users can show the buffer on one framebuffer by calling `VIDIOC_DQBUF` ioctrl and return the buffer to the kernel by calling `VIDIOC_DQBUF` ioctrl.



Features of MXC V4L2 Output

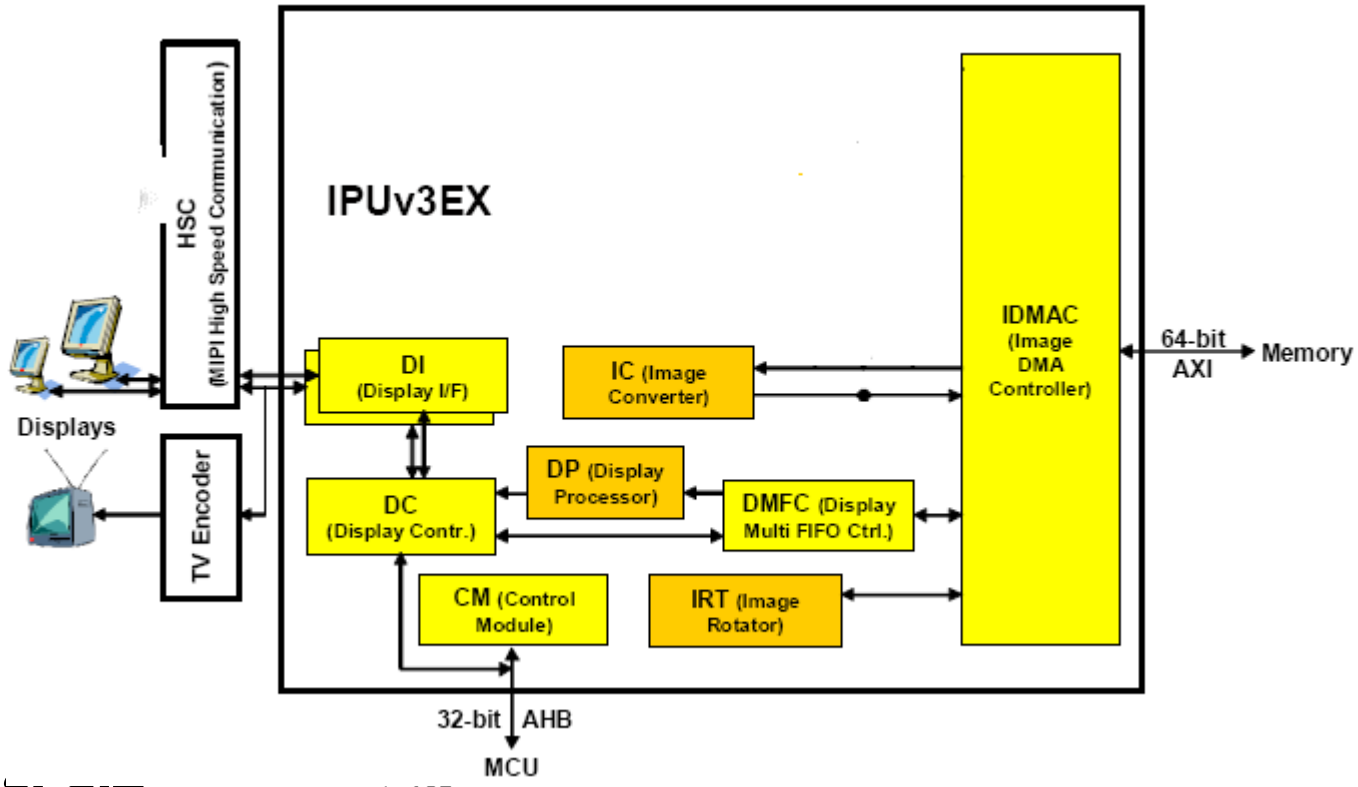
IC bypass mode (using DP or DC channel)

- CSC can be done, but no resizing or rotation can be done. Manually control the display output buffer ready flags in the interrupt handler.
- MXC V4L2 output maintains numbers of buffers.
- Users can show the buffer on one `VIDIOC_QBUF` VIDIOC_QBUF ioctrl.



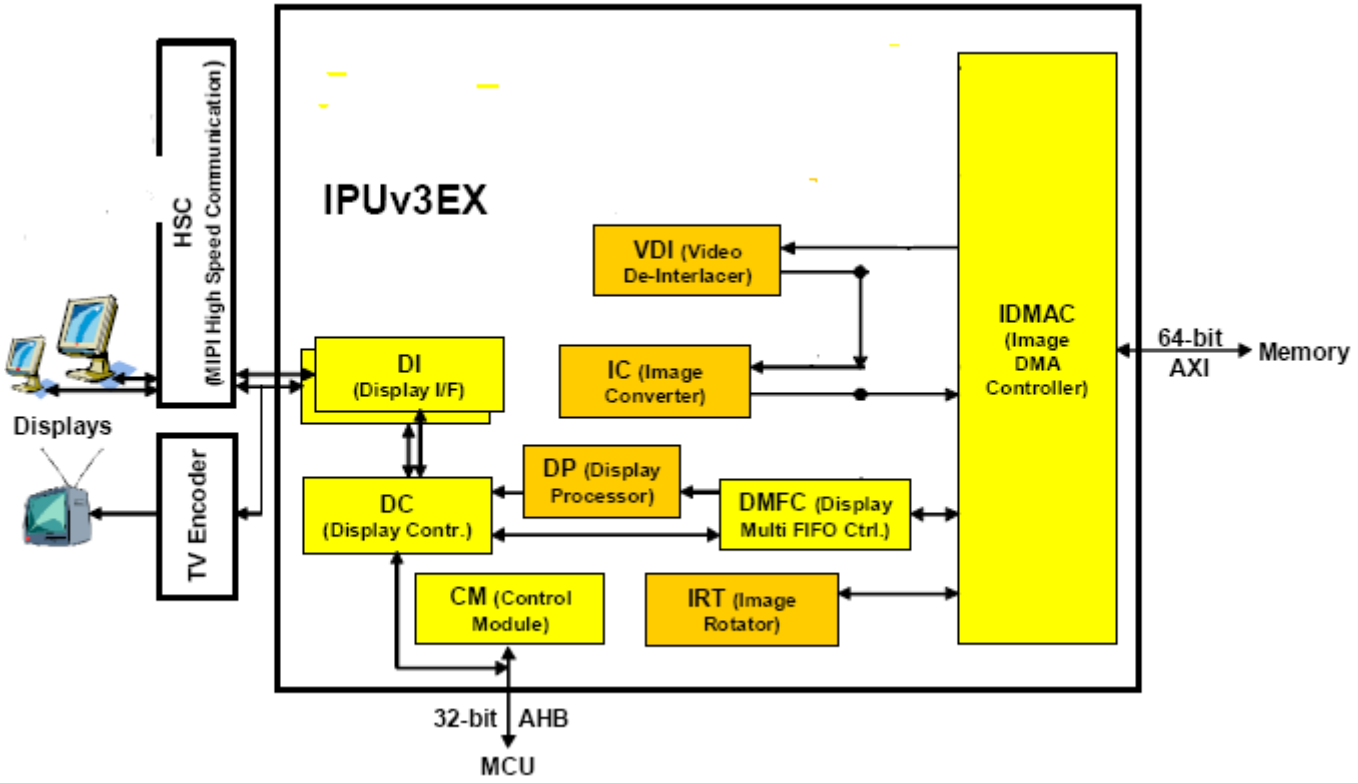
Features of MXC V4L2 Output

- **IC horizontal split mode (using PP channel)**
- Resizing/rotation/CSC. Manually control the IC output/IC input (right stripe)/display input buffer ready flags in the interrupt handler and control IC input (left stripe) buffer ready flags in the timer handler.
- MXC V4L2 output maintains the number of buffers.
- Users can show the buffer on one framebuffer by calling `VIDIOC_QBUF` ioctrl and return the buffer to kernel by calling `VIDIOC_QBUF` ioctrl.



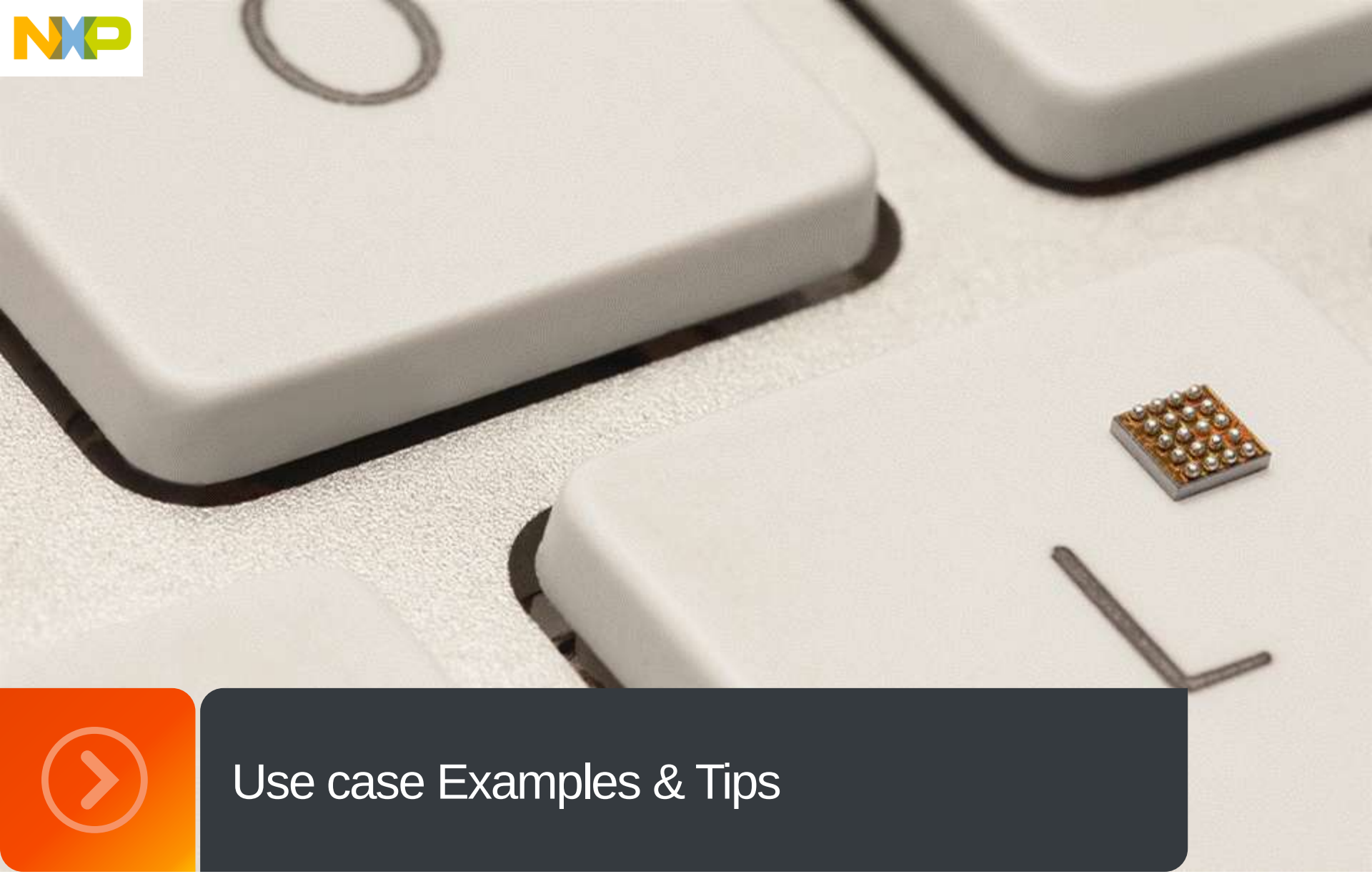
Features of MXC V4L2 Output

- **VDI-IC video deinterlacing mode (using PRP_VF channel)**
- Resizing/rotation/CSC can be done. Manually control the IC output/display input buffer ready flags in interrupt handler and control IC input buffer ready flags in timer handler.
- MXC V4L2 output maintains the numbers of buffers.
- Users can show the buffer on one framebuffer by calling VIDIOC_DQBUF ioctrl and return the buffer to the kernel by calling VIDIOC_QBUF ioctrl.



How do we integrate IPUv3 into MXC V4L2?

- Based on analysis of the IPUv3 spec
- What channel should we use for the framebuffer?
- What channel should we use for V4L2 capture and V4L2 output?
- IPU low-level API design – enable/disable channel, init/unit channel, init channel buffer, interrupt handler register interface…
- Invoke IPU low-level APIs from the MXC V4L2 driver.
- Ensure backwards compatibility in the IPU low-level APIs in cases where the hardware has not changed dramatically.



Use case Examples & Tips

IPUv3 tips

- Use VDOA
  - For more efficient DDR access pattern
- Refresh the display at the rate of the content
  - For displays the perform frame rate conversion.
  - Sometimes called 24P cinema
  - Significantly reduces the amount of data read by IPU.

IPUv3 tips

- Buffer management
  - IPU write channel needs a free buffer in the DDR to start writing data.
  - If there's no free buffer IPU's internal FIFOs are filled, causing additional latencies
  - Buffer management system should guarantee that there're always a free buffer for IPU's usage.
  - IPU can start writing the data to that free buffer immediately avoiding unnecessary

IPUv3 tips

- Move load from the IC
  - Perform CSC (Color Space Conversion), in DP (Display Processor), and not in the IC. (Save memory bandwidth, and lower load on the IC).
  - Move combining tasks to the VDIC (if not used as de interlacer)
  - Consider the IC processing speed, for the tasks
    - Resize – 2 cycles/pixel
    - Combine – 2 cycles/pixel
    - CSC – 3 cycles/pixel
- Flipping an image (a.k.a 180º rotation)
  - Use H-flip and V-flip transfers, done by IDMAC and IC, and not using the IRT module.

IPUv3 tips- Optimizing memory accesses

- Optimize Pixel formats
  - The larger the chunks of data are – the easier it is on the DDR
  - The smaller amount of bursts – better for the memory bus system
  - Choose the mode that works best for the specific use case and avoid the rest

Format	Amount of data per macro block	Burst size	DDR3 x64 BL	Amount of bursts per macro block	Target
YUV422 interleaved	256 bytes	16 bytes	2	16	Best: IPU => IPU; VDOA => IPU
YUV422 partial interleaved	256 bytes	8 bytes + 8 bytes	1 + 1	32	
YUV422 non interleaved	256 bytes	8 bytes + 4 bytes + 4 bytes	1 +1+1	48	
YUV420 interleaved	256 bytes	16 bytes	2	16	
YUV420 partial interleaved (NV12)	192 bytes	8 bytes + 8 bytes	1 + 1	32	Best: VPU => VDOA (decode) Best: IPU => VPU (encode)
YUV420 non interleaved	192 bytes	8 bytes + 4 bytes + 4 bytes	1 +1+1	48	

IPUv3M tips

- How to work efficiently with the memory system
  - Use real time channels
    - Marking IPU accesses with an AXI ID to bypass the PL301's arbitration
  - Lock feature
    - issue a series of IPU bursts the belong to the same channel – better chance for DDR hit
  - Conditional read
    - If an alpha mask is provided to the overlay plane transparent pixels are not read from memory.

IPUv3 tips

- Recommended Display Connectivity

i.MX51						
IPU_DISP1 port	24-bit RGB	RGB666	RGB565	RGB555	24-bit YCbCr	YCbCr4:4:4
DISP1_DAT0	B0	B0	B0	B0	Y0	Y0
DISP1_DAT1	B1	B1	B1	B1	Y1	Y1
DISP1_DAT2	B2	B2	B2	B2	Y2	Y2
DISP1_DAT3	B3	B3	B3	B3	Y3	Y3
DISP1_DAT4	B4	B4	B4	B4	Y4	Y4
DISP1_DAT5	B5	B5	G0	G0	Y5	Y5
DISP1_DAT6	B6	G0	G1	G1	Y6	Y6
DISP1_DAT7	B7	G1	G2	G2	Y7	Y7
DISP1_DAT8	G0	G2	G3	G3	Cb0	Cb0
DISP1_DAT9	G1	G3	G4	G4	Cb1	Cb1
DISP1_DAT10	G2	G4	G5	R0	Cb2	Cb2
DISP1_DAT11	G3	G5	R0	R1	Cb3	Cb3
DISP1_DAT12	G4	R0	R1	R2	Cb4	Cb4
DISP1_DAT13	G5	R1	R2	R3	Cb5	Cb5
DISP1_DAT14	G6	R2	R3	R4	Cb6	Cb6
DISP1_DAT15	G7	R3	R4		Cb7	Cb7
DISP1_DAT16	R0	R4			Cr0	Cr0
DISP1_DAT17	R1	R5			Cr1	Cr1
DISP1_DAT18	R2				Cr2	Cr2
DISP1_DAT19	R3				Cr3	Cr3
DISP1_DAT20	R4				Cr4	Cr4
DISP1_DAT21	R5				Cr5	Cr5
DISP1_DAT22	R6				Cr6	Cr6
DISP1_DAT23	R7				Cr7	Cr7
DI1_PIN2	HSYNC					
DI1_PIN3	VSYNC					
DI1_PIN15	DRDY					
DI1_DISP_CLK	CLK					

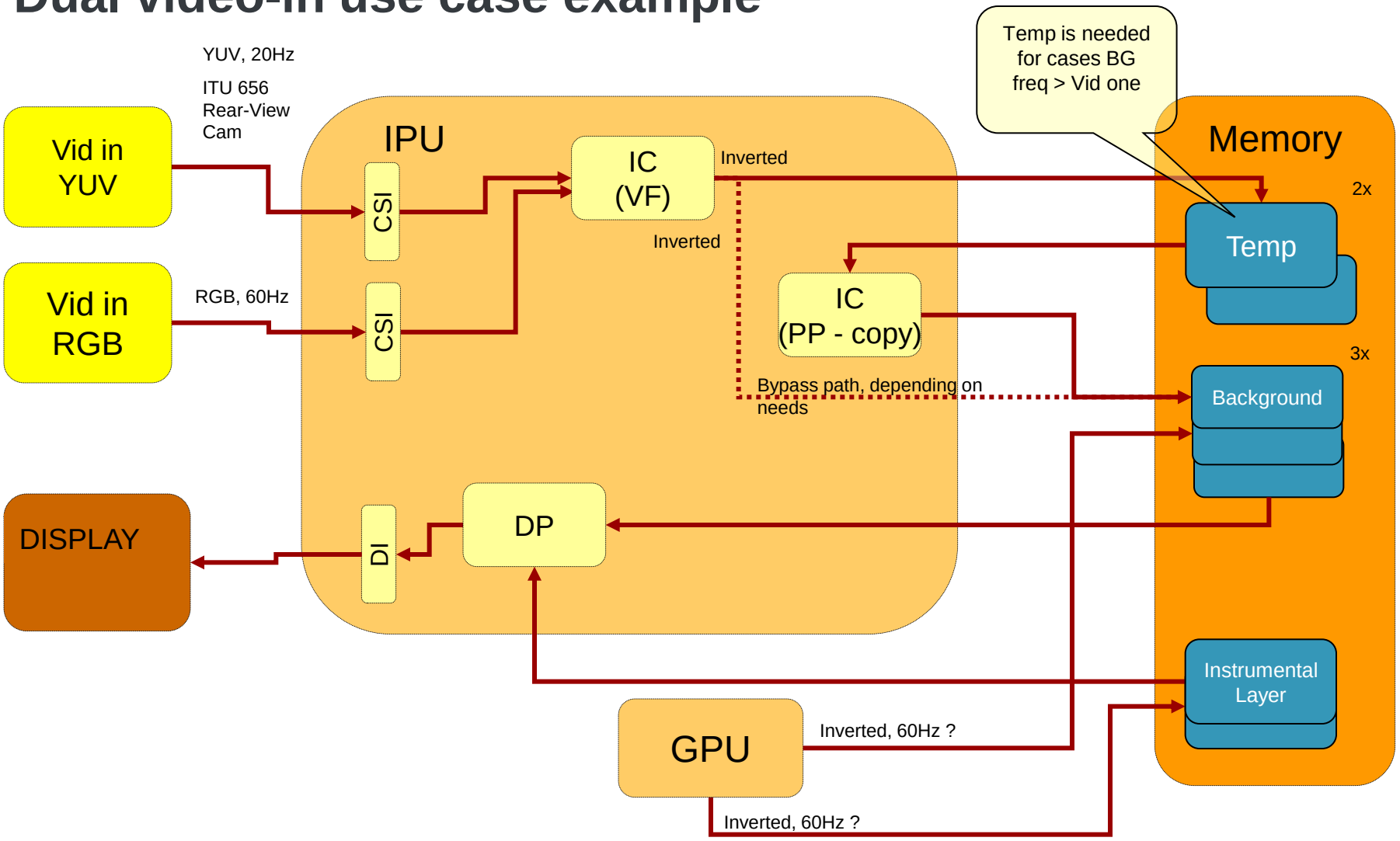
IPUv3 - debug

- IPU error interrupts & status bits
  - IPU errors are reported on the IPU_INT_STAT_5, IPU_INT_STAT_6, IPU_INT_STAT_9 and IPU_INT_STAT_10 registers. The 1st debug step should be inspecting these bits
    - A flickering display is normally a result of a system bus load (DDR).
    - These will be reported as “new frame before end of frame error” on IDMAC_NFB4EOF register.
    - Bus loads that causes errors on the CSI side will be reported on *FRM_LOST* status bits
    - Some of IPU internal signals can be routed to pins and measured using the IPU diagnostics unit. These can be used to capture errors/interupts and track internal flows. (the IOMUX needs to be configured to output the ipu_diagbus signals)

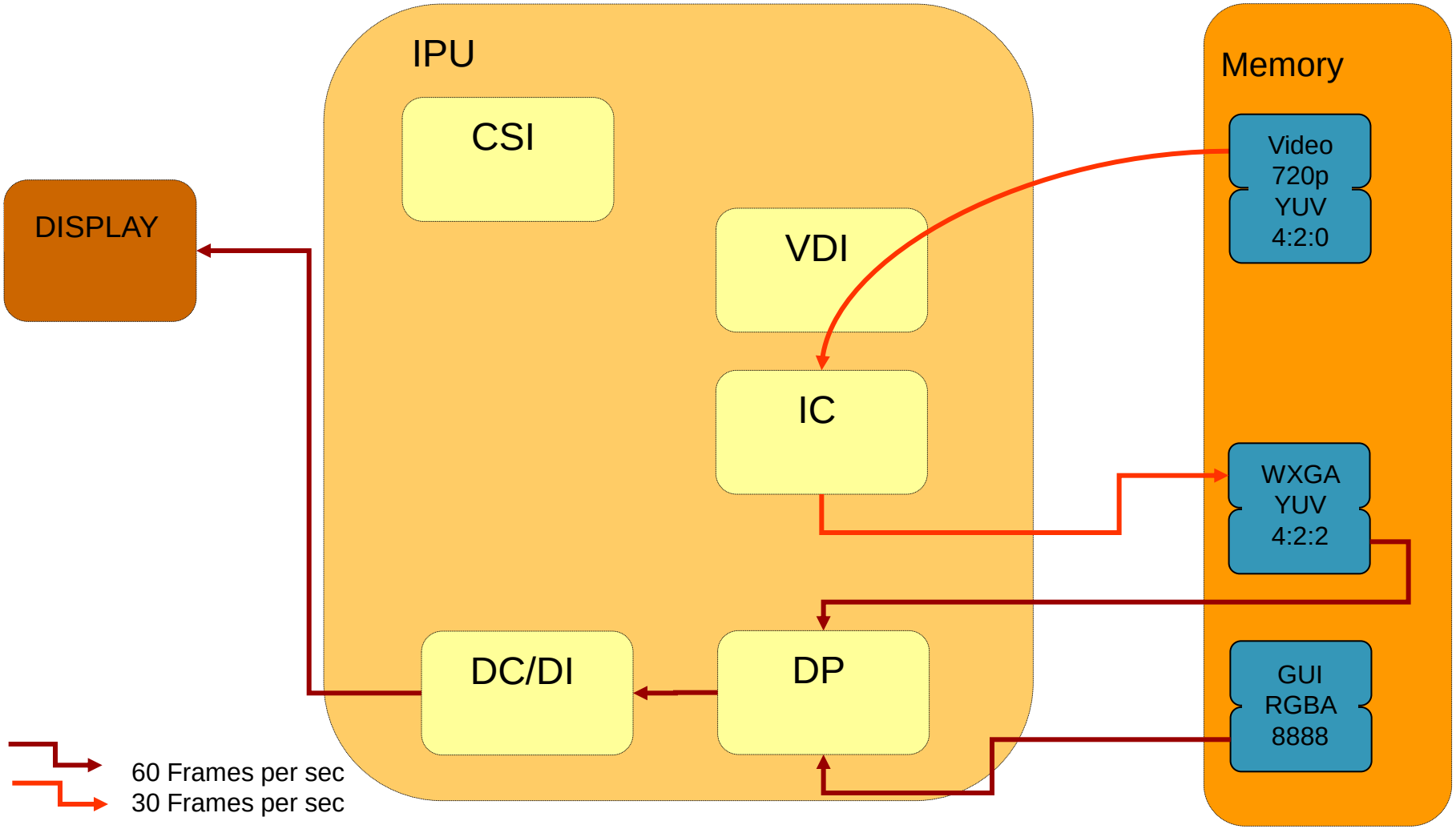
IPUv3 - debug

- IPU diagnostics unit
  - Some of IPU internal signals can be routed to pins and measured using the IPU diagnostics unit.
  - These can be used to capture errors/interupts and track internal flows.
  - The IOMUX needs to be configured to output the ipu_diagbus signals.
- Task status and flow control
  - A frozen display is sometimes a result of wrong control of the buffer management within the IPU.
  - The status of each flow controlled by the FSU can be monitored using the TASKS_STAT status registers.
  - In some cases a user may track the BUF_RDY and CUR_BUF indications of the flow to track the flow.

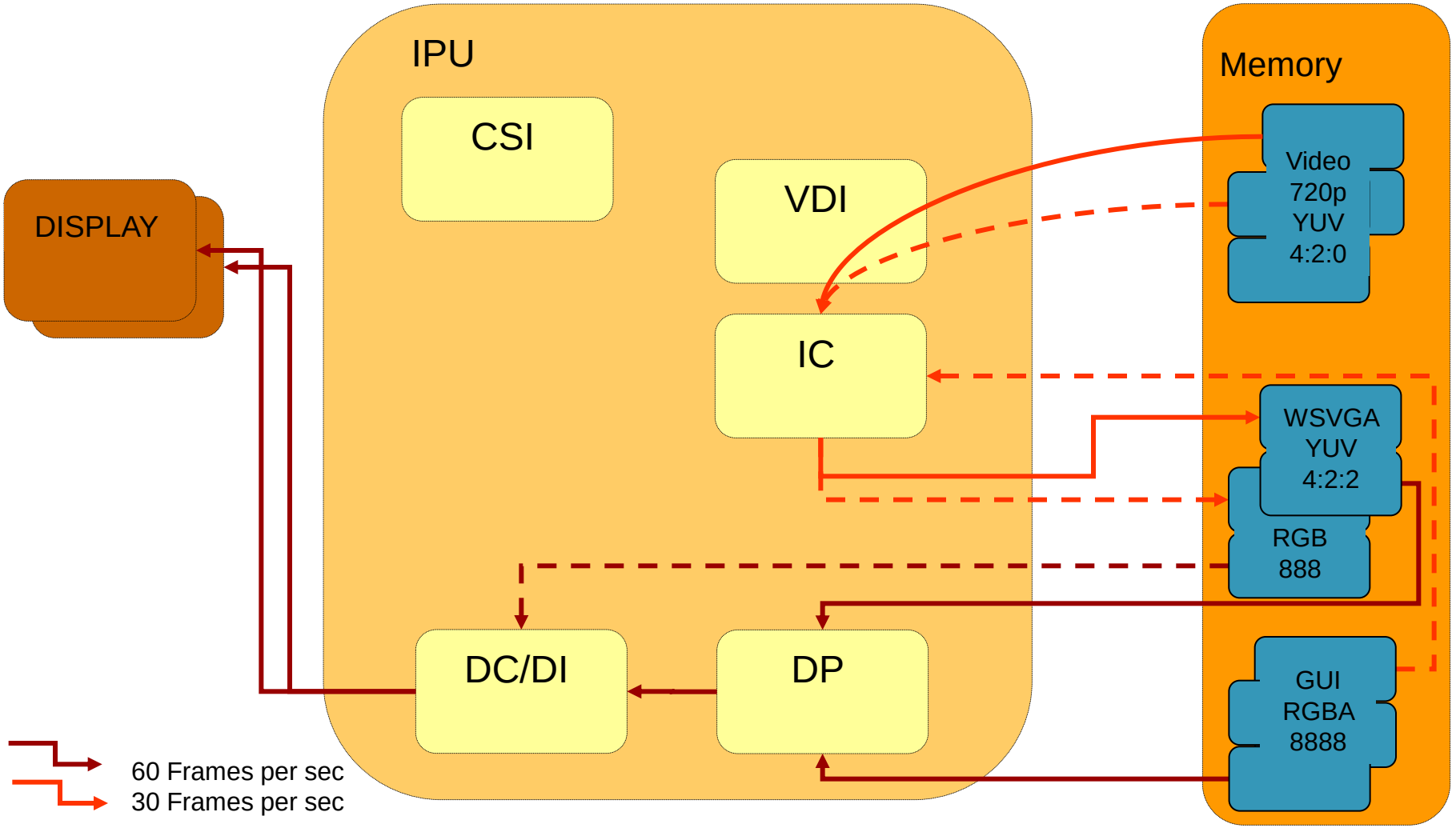
Dual video-in use case example



Playback, HD1080p H.264 HP –> Display



Dual Playback, HD720p H.264 HP –> WSVGA Display



Demo

Q & A

www.Freescale.com