Video Drivers

4th July 2017
V1.1

Agenda

- FVID2: Introduction
- DSS Video Driver
- VIP Video Driver
- VPE Video Driver

FVID2 Interface

FVID2: Introduction

- What is FVID2?
  - Next version of FVID and it addresses the different limitations of FVID
  - Provides interface to streaming operations like queuing of buffers to driver and getting back a buffer from the driver
  - Abstracts the underlying hardware for the video application with a standard set of interface
  - Gives a same look and feel for video applications across different SOC
  - Interface is independent of OS/Hardware/Driver
  - FVID2 is currently supported on SYSBIOS OS
- What is not FVID2?
  - Not the actual driver
  - Does not define hardware specific APIs and structures

Understanding FVID2 - Interfaces

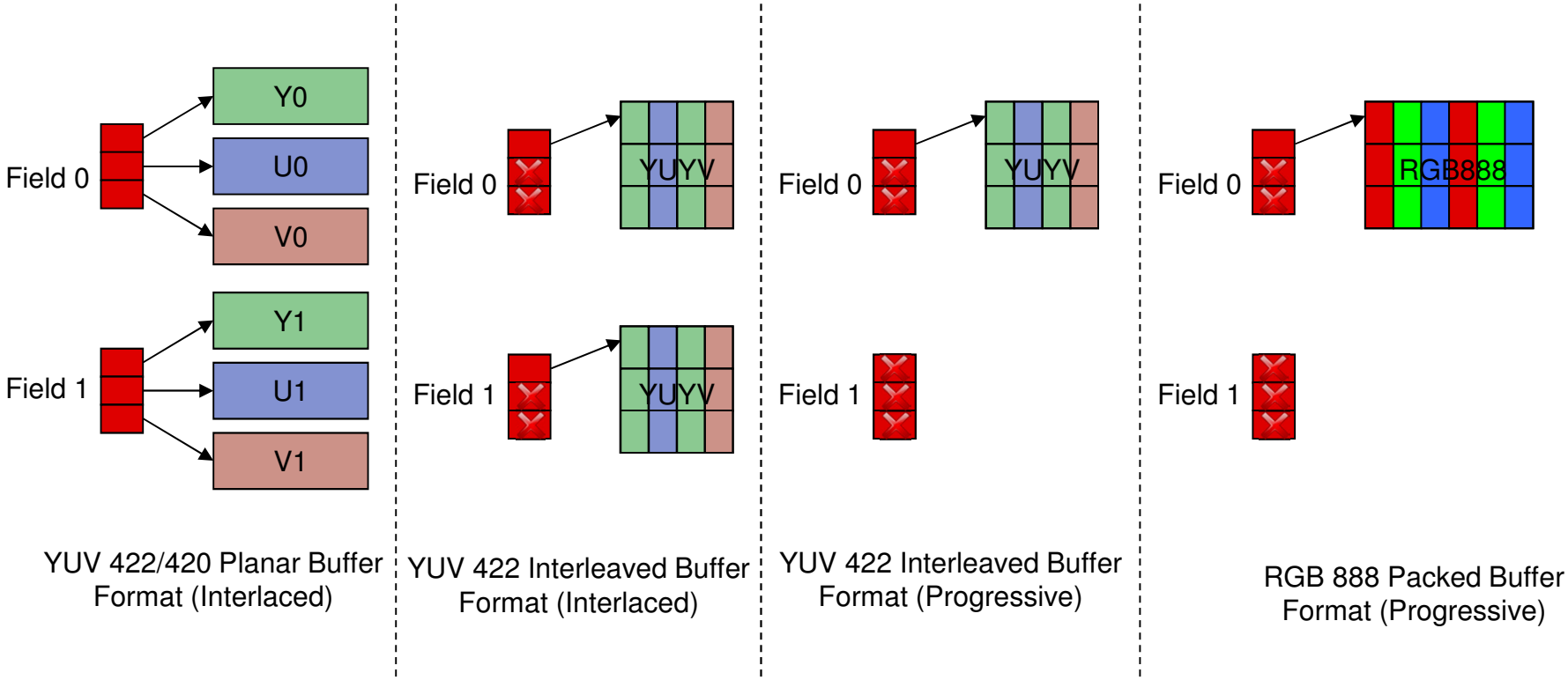
- *FVID2_init*
  - Initializes the drivers and the hardware. Should be called before calling any of the FVID2 functions
- *FVID2_deInit*
  - Un-initializes the drivers and the hardware
- *FVID2_create*
  - Opens a instance/channel video driver
- *FVID2_delete*
  - Closes a instance/channel of a video driver
- *FVID2_control*
  - To send standard (set/get format, alloc/free buffers etc..) or device/driver specific control commands to video driver
- *FVID2_queue*
  - Submit a video buffer to video driver. Used in display/capture drivers
- *FVID2_dequeue*
  - Get back a video buffer from the video driver. Used in display/capture drivers
- *FVID2_processFrames*
  - Submit video buffers to video driver for processing. Used only in M2M drivers
- *FVID2_getProcessedFrames*
  - Get back the processed video buffers from video driver. Used only in M2M drivers
- *FVID2_start*
  - Start video capture or display operation. Not used in M2M drivers
- *FVID2_stop*
  - Stop video capture or display operation. Not used in M2M drivers

Understanding FVID2 - Interfaces

- FVID2_Frame
  - Represents the video frame buffer along with other meta data
  - This is the entity which is exchanged between driver and application and not the buffer address pointers
  - Meta data include timestamp, field ID, per frame configuration, application data etc…
  - Since video buffers can have up to 3 planes and two fields (in the case of YUV planar interlaced), buffer addresses are represented using a two dimensional array of pointers of size 2 (field) x 3 (planes)

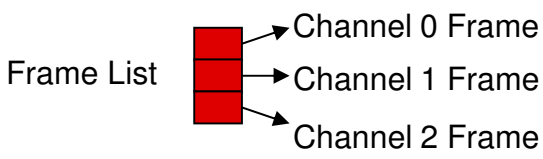
Understanding FVID2 - Interfaces

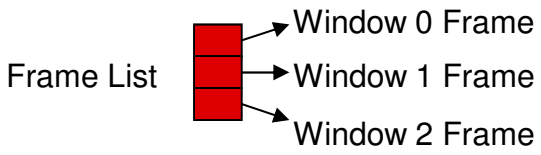
- FVID2_Frame – How address pointers are used?



Understanding FVID2 - Interfaces

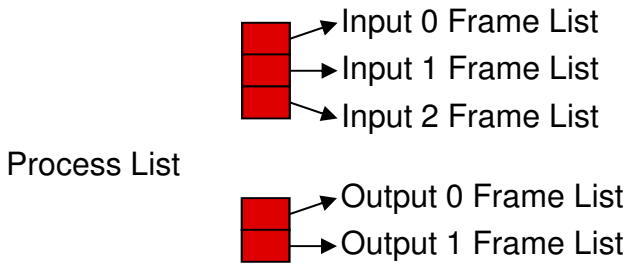
- FVID2_FrameList
  - Represents N FVID2_Frame
  - N Frames could represent
    - Different capture channels in multiplexed capture
    - Buffer address for each of the window in multi window mode
  - N is fixed at 64 in the current implementation



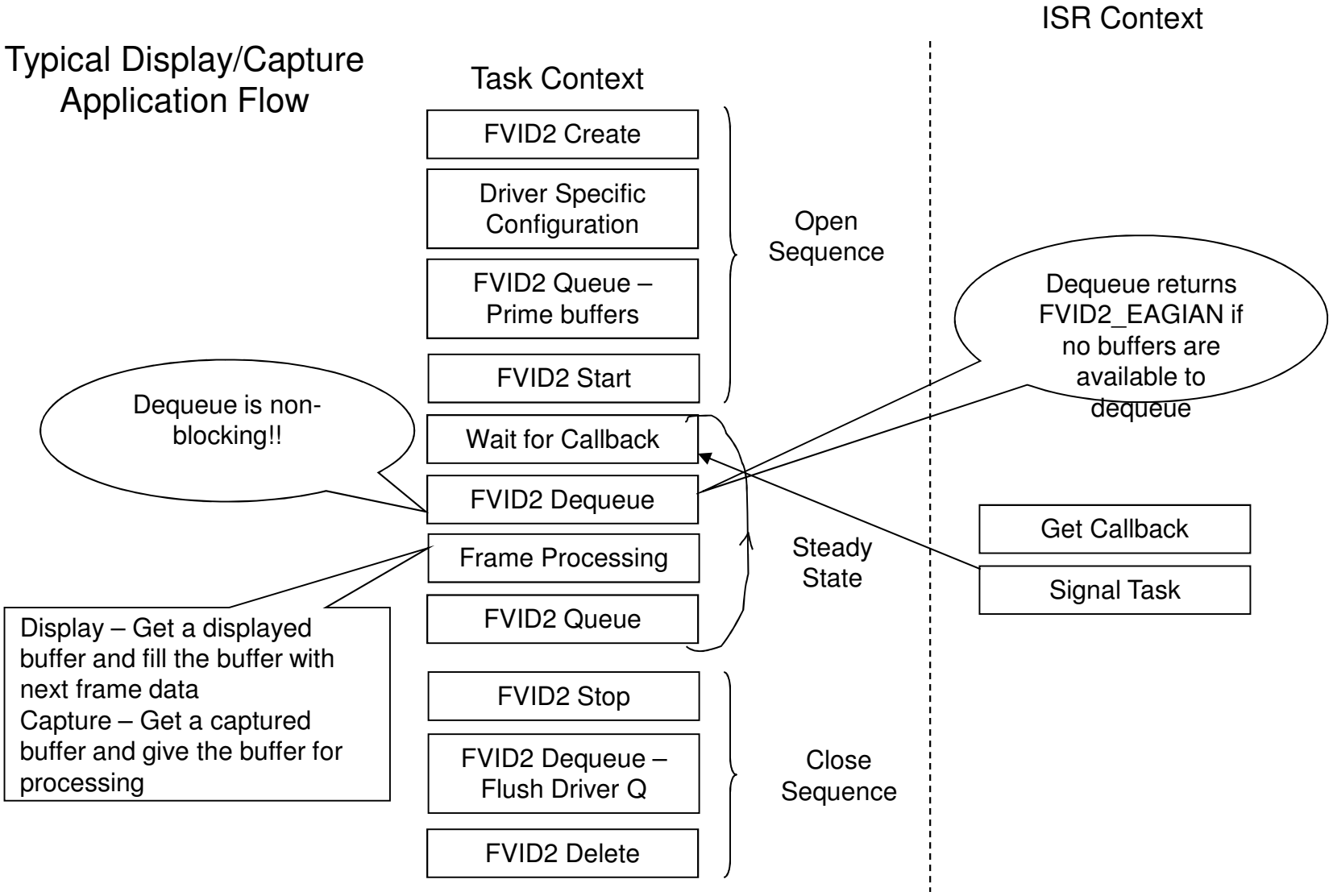


Understanding FVID2 - Interfaces

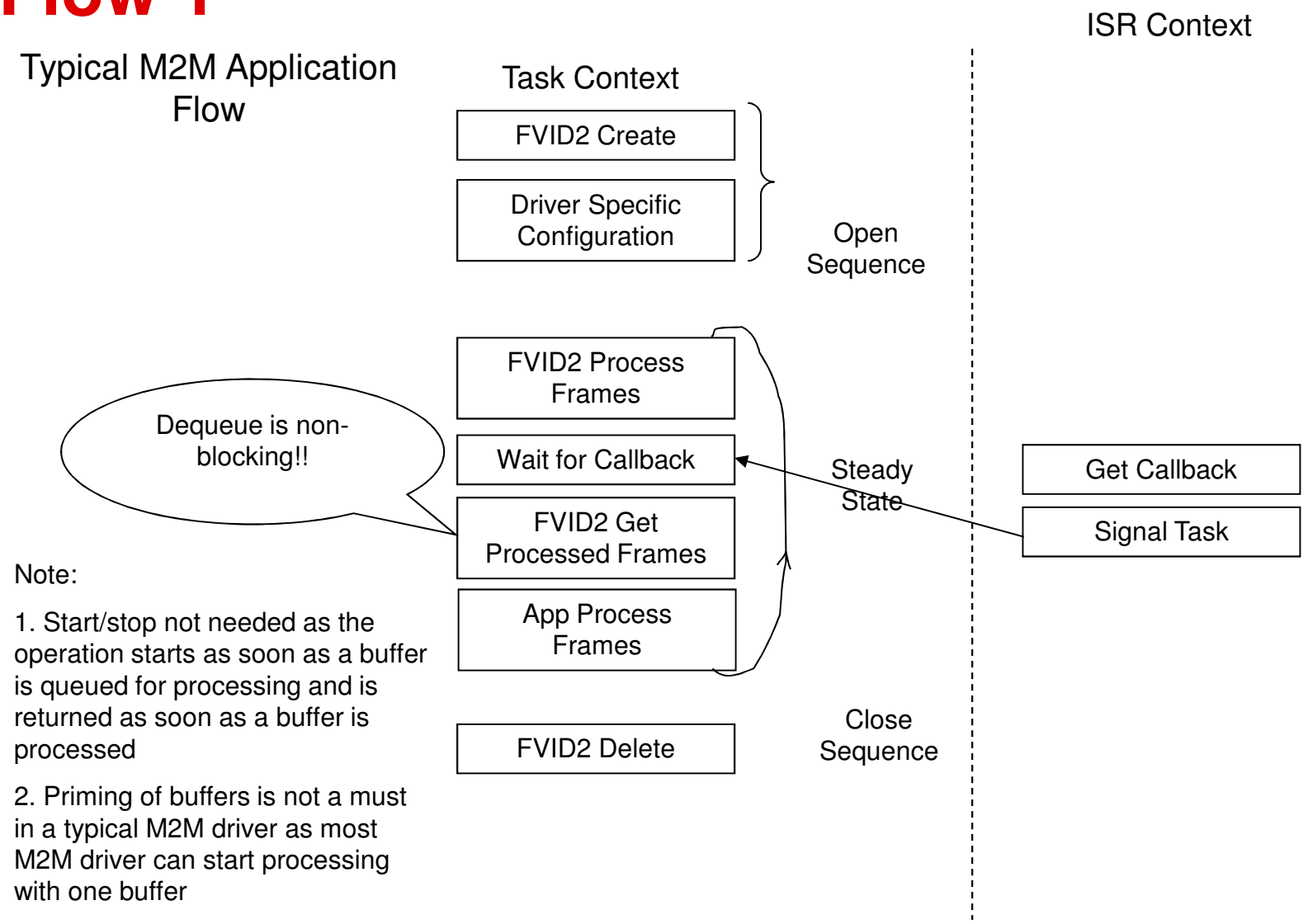
- FVID2_ProcessList
  - Represents M FVID2_FrameList for input and output frames
  - Each frame list represents a N frame buffers for each of the inputs and outputs in M2M drivers
  - Used only in M2M drivers



Understanding FVID2 – Application Flow 1



Understanding FVID2 – M2M Application Flow 1

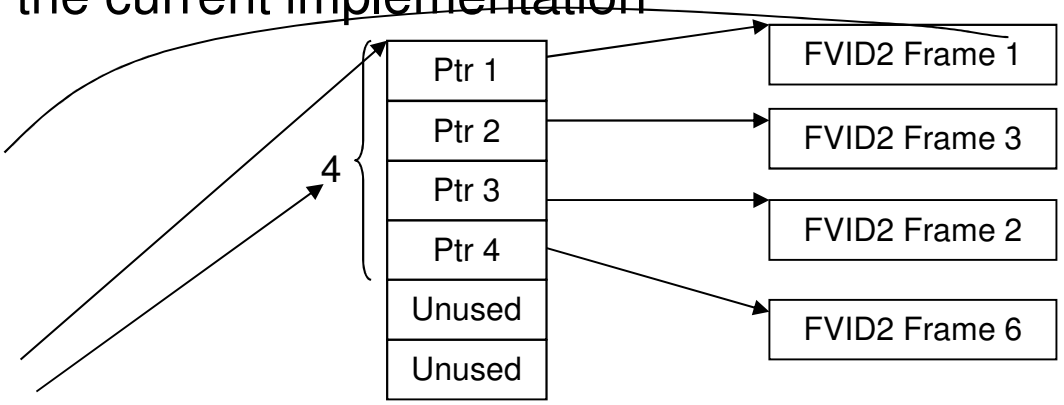


Understanding FVID2 – Multiple Frames Per Request Feature

- This feature supports
  - Multi-window display (Multiple buffers belonging to one stream)
  - Multiplexed capture (Multiple buffers belonging to multiple streams)
  - Multiple M2M request per call (Multiple request belonging to multiple streams/channels)
- FVID2 Frame -> one buffer/request
- FVID2 FrameList -> Multiple buffers/requests
- FVID2_FrameList contains an array of FVID2_Frame pointers whose size if fixed at 64 in the current implementation

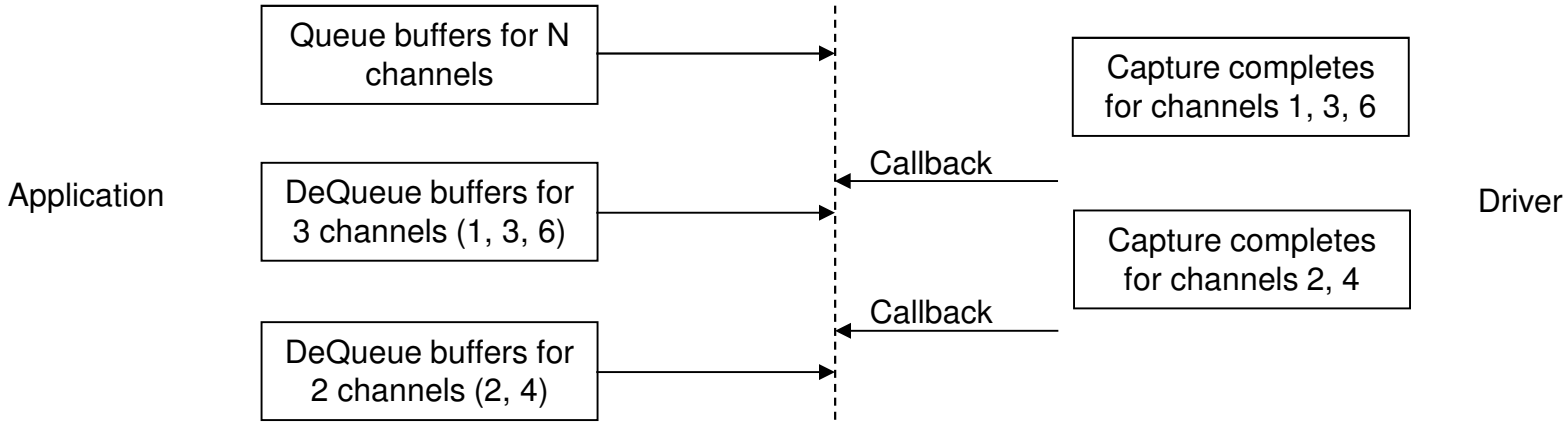
```
typedef struct FVID2_Frame_t
{
    Ptr                 addr[2][3];
    UInt32              channelNum;
    /* Other members not shown */
} FVID2_Frame;

typedef struct FVID2_FrameList_t
{
    FVID2_Frame        *frames[64];
    UInt32              numFrames;
    /* Other members not shown */
} FVID2_FrameList;
```



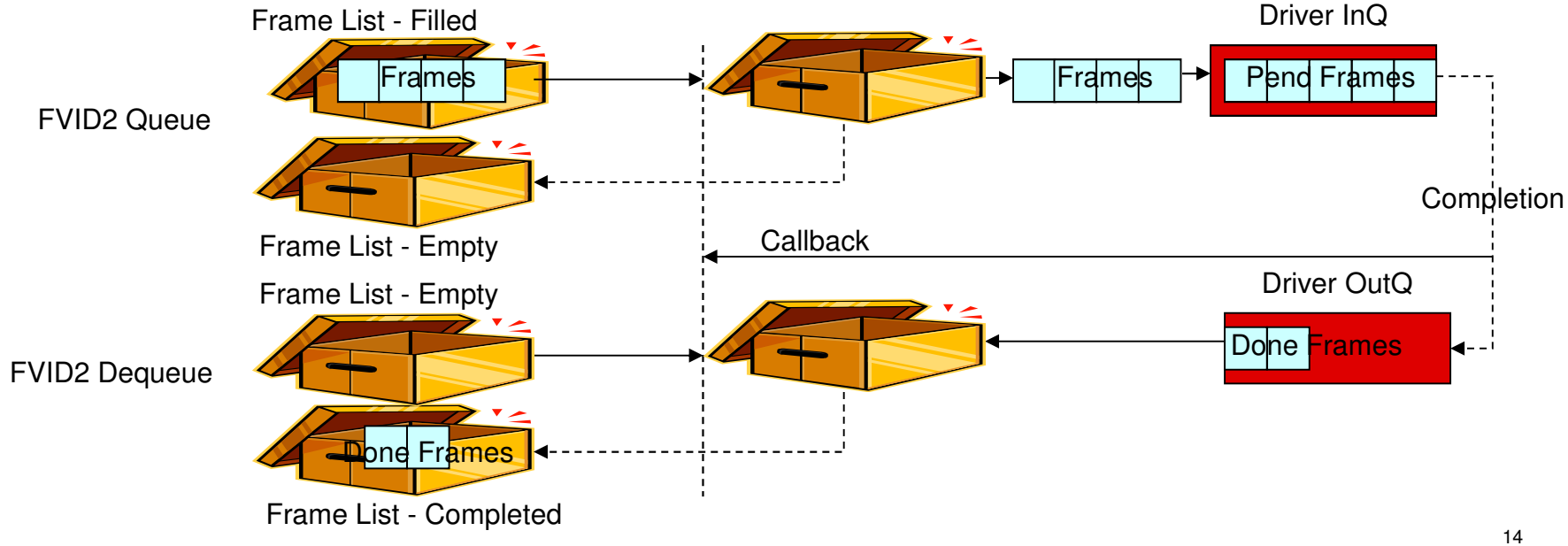
Understanding FVID2 – One Q, Multiple DQ

- Used in multiplexed capture
  - While priming, application submits buffers for all the channels using Queue call
  - Since multiplexed inputs could be asynchronous, capture could complete at different time for each of the inputs
  - Application wants to process the buffers as soon as they are captured
  - Hence they are de-queued immediately without waiting for other channels to complete
  - This will result in multiple dequeue for a single queue



Understanding FVID2 – One Q, Multiple DQ Contd…

- How is this achieved?
  - Only frames are queued/dequeued in/from the driver
  - Frame list is not queued/dequeued
  - Frame list acts like a container to submit the frames to the driver in Queue call and take back the frames from the driver in dequeue call
  - For queue call, application is free to re-use the same frame list again without dequeuing
  - For dequeue call, the application has to provide the frame list to the driver and the driver copies the captured frame to the frame list



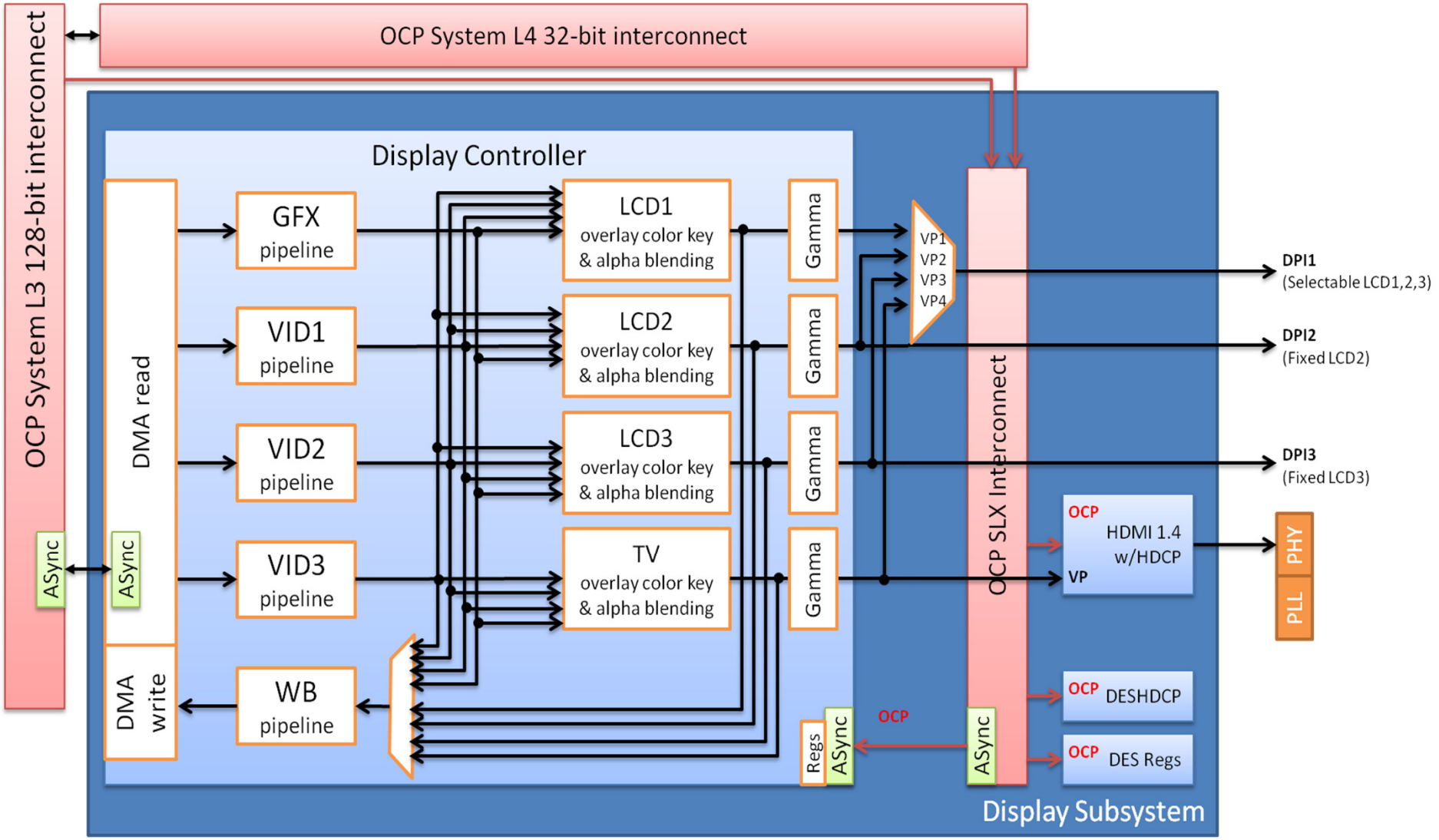
DSS Overview
(Display Sub system)

What is DSS?

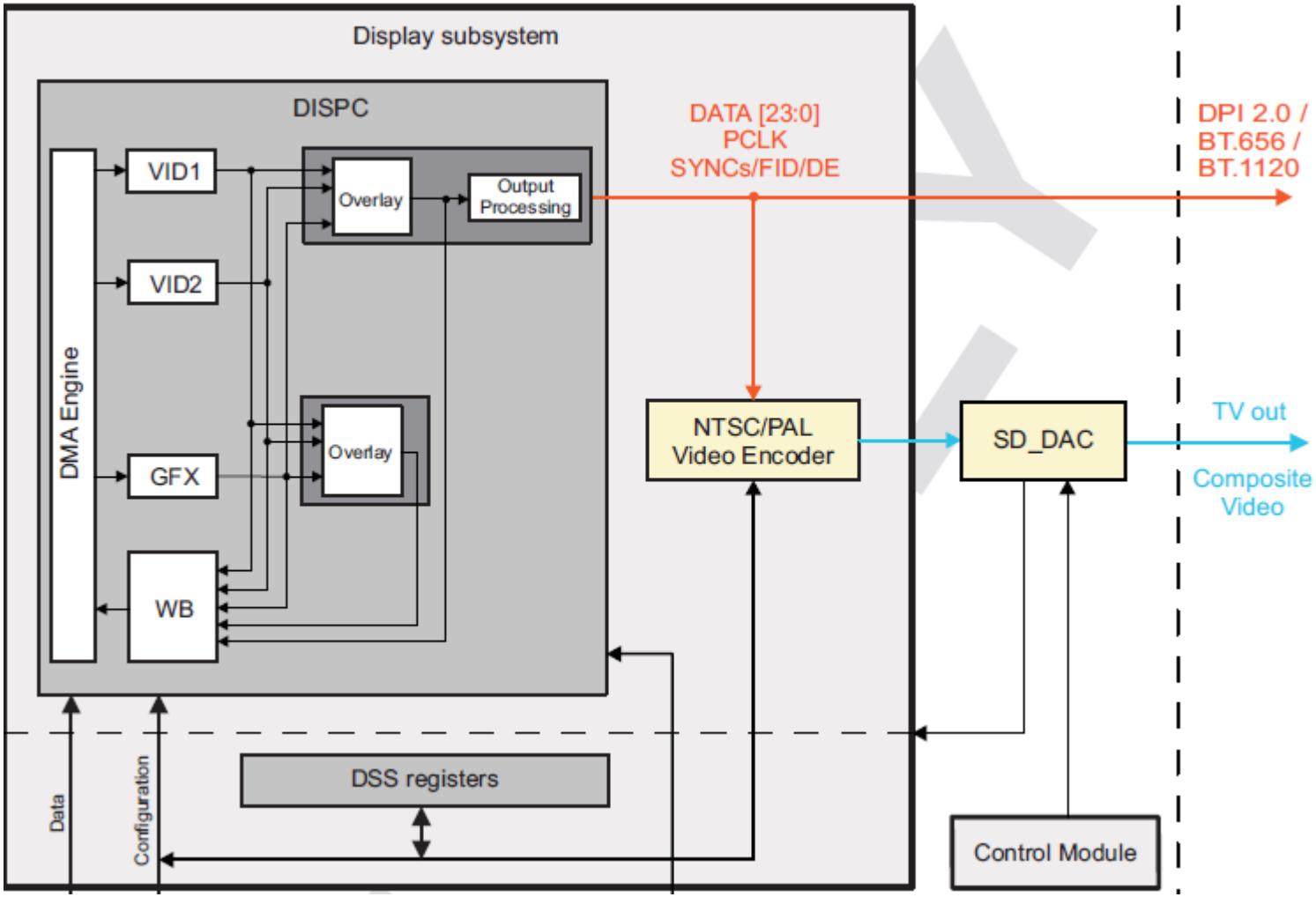
- DSS – Display Sub System for TDA2xx/TDA3xx
- Displays video frames from memory to LCD1 or TV (HDMI)
- Composition of graphics and video sources
- Scaling, Color Space Conversion, Blending, Color Keying

DSS Hardware Overview

DSS Hardware for TDA2xx



DSS Hardware for TDA3xx



DSS Hardware TDA2xx

- Display Controller
- 4 Pipelines (3 Video and 1 Graphics)
- 4 Overlay managers
- Write Back pipeline
- Direct Memory Access
- 4 Interfaces (3 DPI, 1 HDMI)

DSS Hardware TDA3xx

- Display Controller
- 3 Pipelines (2 Video and 1 Graphics)
- 2 Overlay managers
- Write Back pipeline with region based WB support
- Direct Memory Access
- 2 Interfaces (1 DPI, 1 SD-DAC)

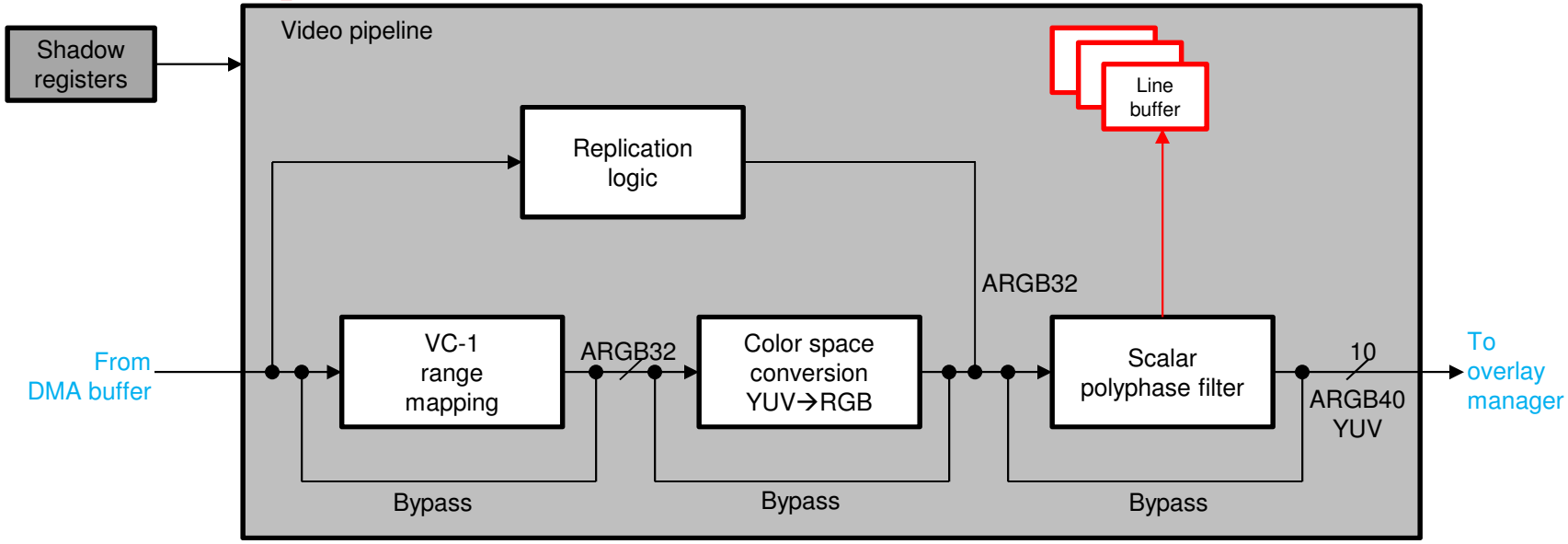
Display Controller

- Processes on-the-fly video streams and graphics
- No extra memory needed for processing
- Fetches pipeline data through DMA transfers
- 4 Overlay managers
  - 3 LCD & 1 TV output
- Can process maximum at 192Mpix/sec
  - 1920x1200 @60fps OR 2048x1536 @59fps

Video Pipeline

- Three Video Pipelines, all are identical
- Fetch data from frame buffers using DMA
- VID – RGB, YUV422, YUV420 NV12 up to 1920x2048
- Each pipeline has a scalar ,color space converter, VC1 Range mapping

Video Pipeline



- VC-1 Range Mapping is to remap the Y, Cb and Cr components (mainly used when the video frame picture is decoded using a VC-1 codec).

CSC Unit: YUV to RGB

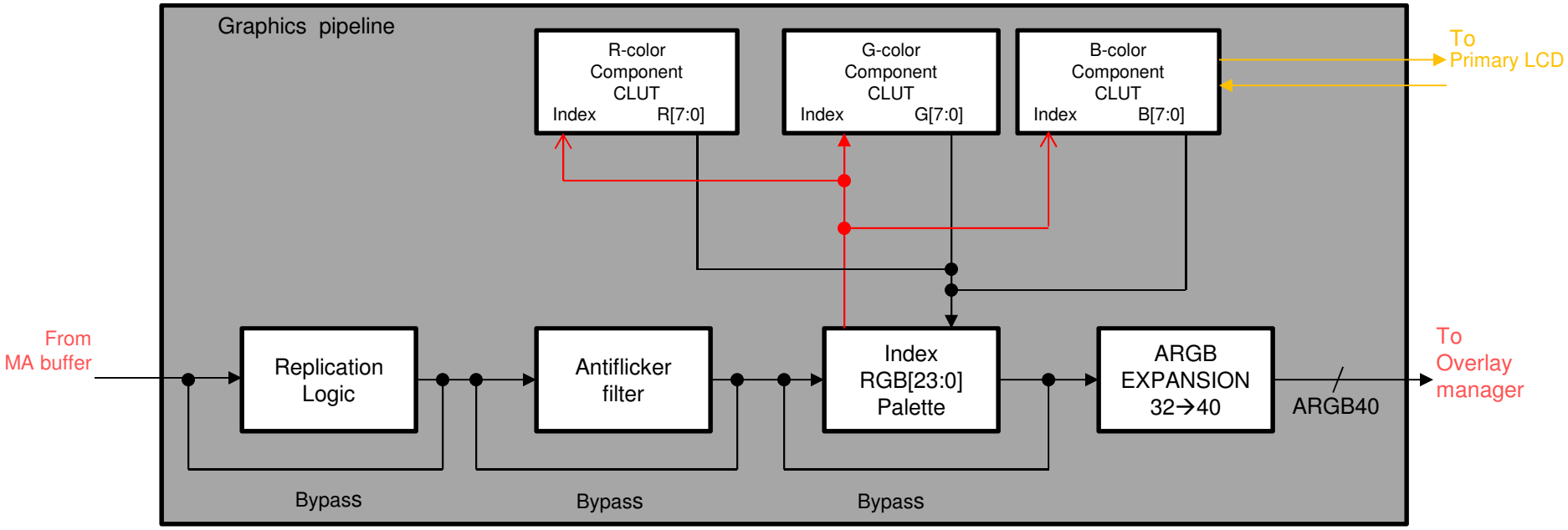
- CSC Unit will convert Video encoded pixels from YUV4:4:4 format into RGB24 or RGB30 format.
- In case of YUV4:2:0 or YUV4:2:2 formats chrominance resampling is required before converting to RGB format.

Scalar Unit

- Works on RGB and YUV data formats
- Filter used is FIR filter
- Filter can be used for
  - Upscaling, Downscaling
  - Antiflicker Reduction
  - Spatial De-Interlacing using BOB algorithm
  - Chrominance resampling for YUV formats
- Max Upscaling ratio is 8x
- Downscaling Using 3-tap configuration is x0.5 and using 5-tap its x0.25

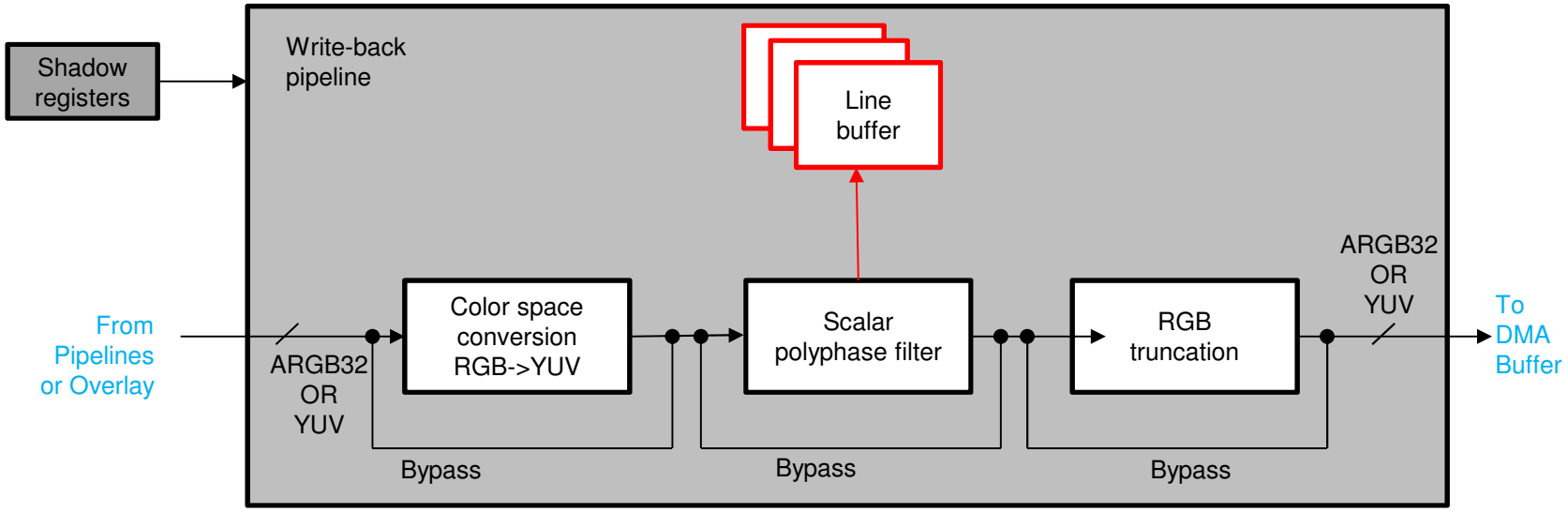
Graphics pipeline

- The replication logic is used to convert the RGB pixel formats into an ARGB40-based format
- The antiflicker filter processes the graphics data in RGB format to remove some of the vertical flicker
- The 256-entry palette is used to convert bitmap formats into RGB
- There is no scalar block in Graphics pipeline
- Supports only RGB and BITMAP Formats

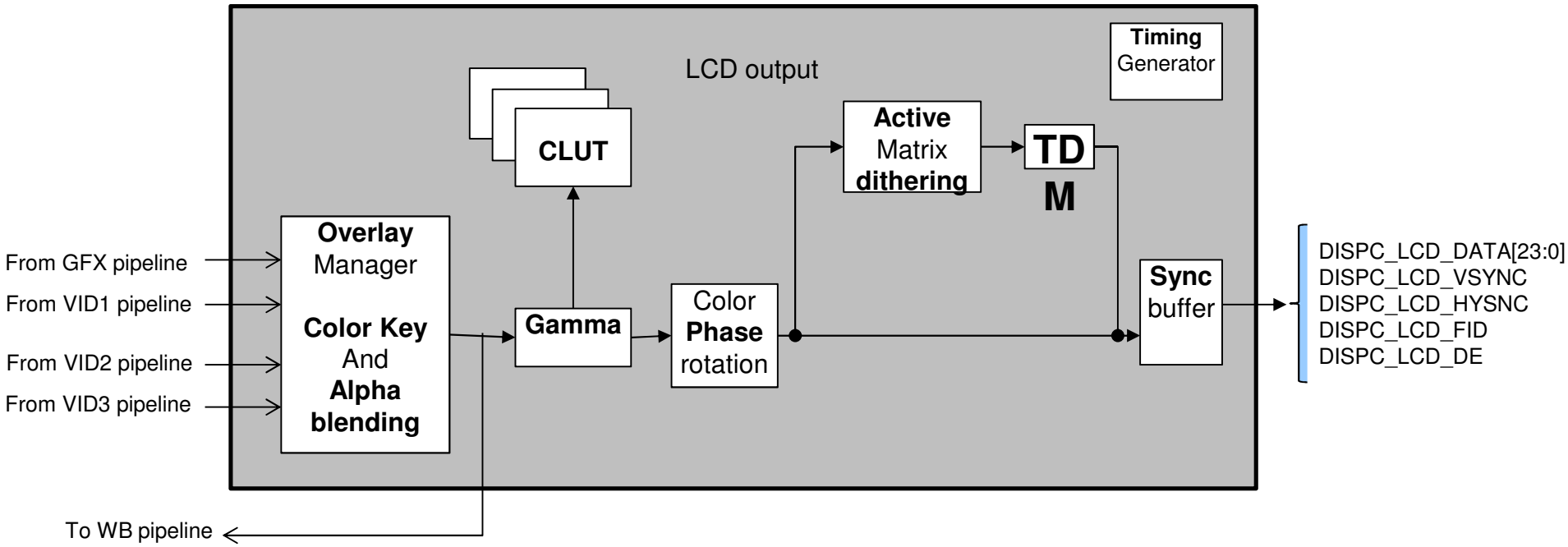


Write Back Pipeline

- Used to store in the system memory the capture of Overlay output or the output of one of the pipeline
- Supports WB memory-to-memory Mode and WB capture Mode.
- Truncation Logic is used to convert ARGB32 bit formats into lower color depth : 12 bit or 16 bit formats.
- CSC used to convert RGB to YUV formats.



Overlay Manager



Overlay Manger Capabilities

- Z-Order: App can set the Ordering of layers
- Transparency color keys: Source and destination (can only be used with RGB and BITMAP formats)
- Alpha Blending: Global alpha blending and pixel level alpha blending
- Gamma Correction, temporal dithering and phase rotation
- Configurable output size and position of pipeline

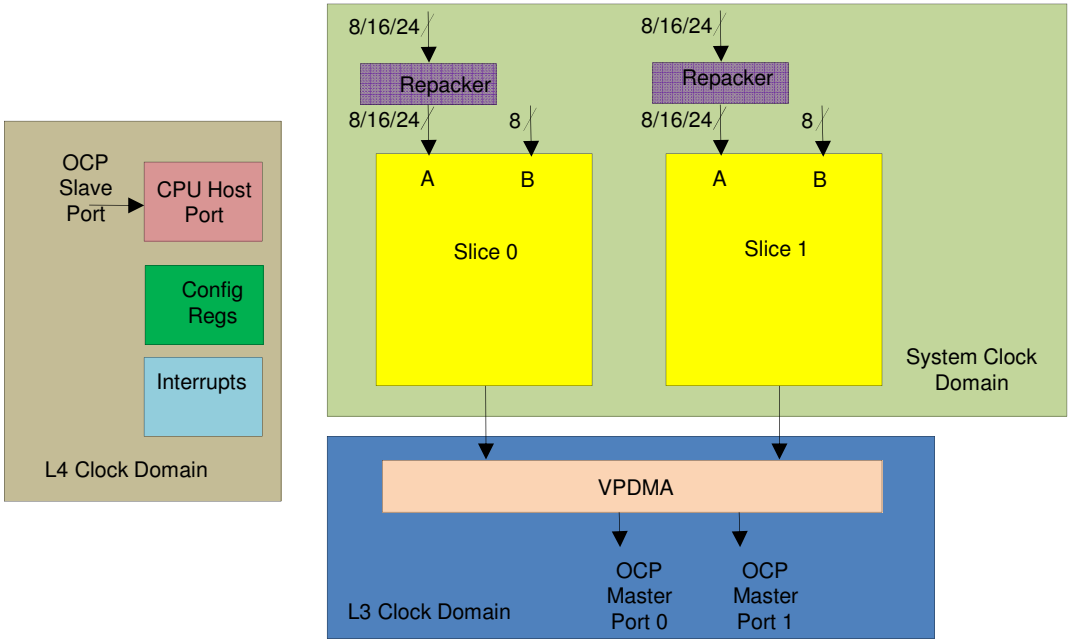
Overlay Manager Capabilities Cont..

- Color Phase rotation unit is used to correct the LCD output colorimetry in case of non pure white backlight, it can also be used to convert RGB to YUV.

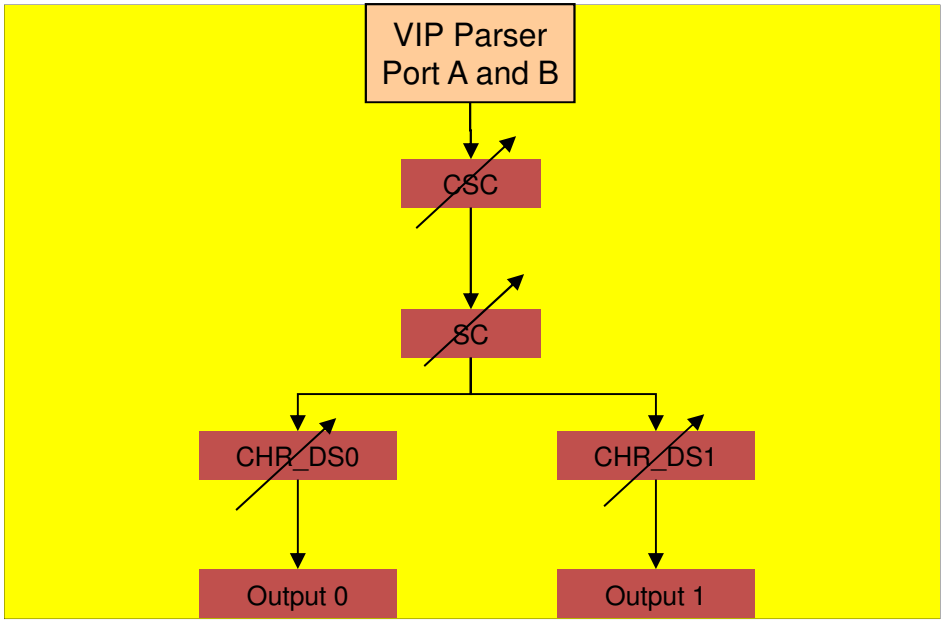
Z-Order Example

VIP Overview
(Video Input Port)

VIP Sub-sytem Block



VIP Slice Block



VIP Block

- TDA2xx Supports 3 VIP Block with two ports Port A and Port B per slice
  - VIP1/ VIP2 parser can operate as one 8/16/24 bit input port (Port A) or two 8-bit ports (Port A, Port B).
  - VIP3 parser can operate as 16 bit input port (Port A)
- Separate pixel clock and framing signals for each port
- Each VIP block has two slices : VIPx S0, VIPx S1
- VIP Block is used to capture video data from external video sources like video decoders or sensors
- Typical TDAxx system can be used for capture from four Mpixel imagers (1280x800) resolution at 30 FPS

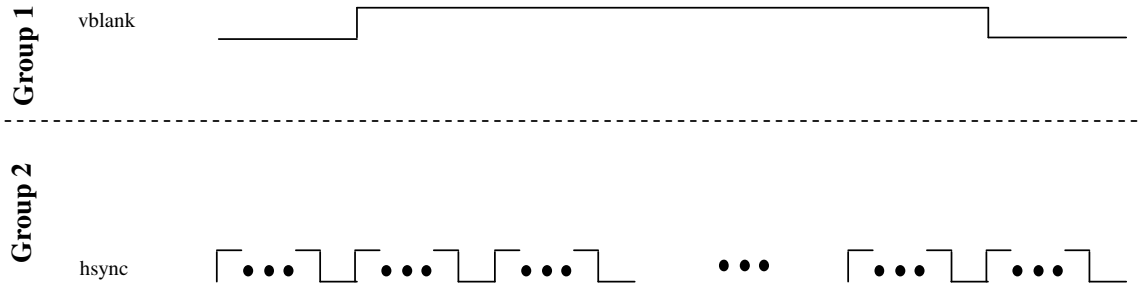
VIP Features

- Format conversion
  - Inputs: YUV422I, YUV444, RGB888
  - Outputs: YUV422I, YUV420SP (Uses CHR_DS), YUV422SP, RGB888 (Uses CSC)
- Supports optional scaling from 1/8x to 2048 pixels, only down scaling supported
- Supports optional color space conversion
  - YUV to RGB color space conversion using CSC block
- Supports optional chroma downsampler
  - YUV422I to YUV420SP conversion using CHR_DS
- Supports up to 165 MHz clock (includes blanking)
- Video Interface width
  - 8/16/24 bit mode
  - Combination: PortA 8-bit and PortB 8-bit or PortA 16 or 24-bit

Contd..

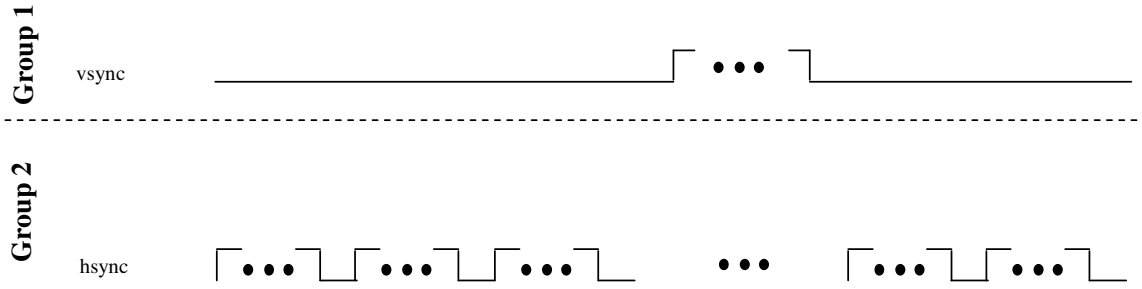
- Supports Embedded (BT.656/BT.1120 16/24b, BT.656 8b) or discrete (BT.601 style) sync
- Video Interface Mode
  - Discrete Sync
    - Single Channel non multiplexed mode with HSYNC and VBLK as control signals
    - Single Channel non multiplexed mode with HSYNC and VSYNC as control signals
    - Single Channel non multiplexed mode with AVID and VBLK as control signals
    - Single Channel non multiplexed mode with AVID and VSYNC as control signals
  - Embedded Sync
    - Single Channel non multiplexed mode
    - Multi-Channel pixel or line multiplexed mode

VIP Interface modes



- Active lines captured
- All pixels in line captured
- Frame Start: VBLK high to low
- Line start: HSYNC low to high

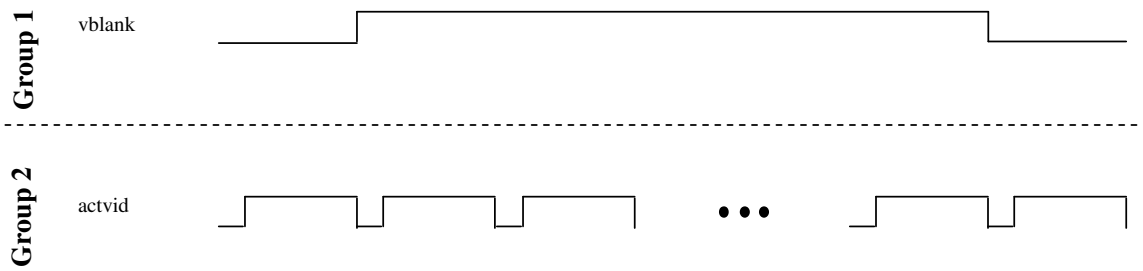
DS : Single Channel non multiplexed mode with HSYNC and VBLK as control signals



- All lines captured
- All pixels in line captured
- Frame Start: VSYNC low to high
- Line start: HSYNC low to high

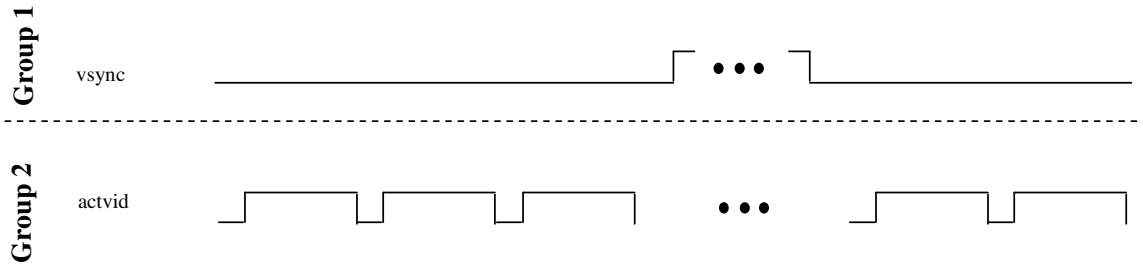
DS :Single Channel non multiplexed mode with HSYNC and VSYNC as control signals

Contd..



- Active lines captured
- Active pixels in line captured
- Frame Start: VBLK high to low
- Line start: ACTVID low to high

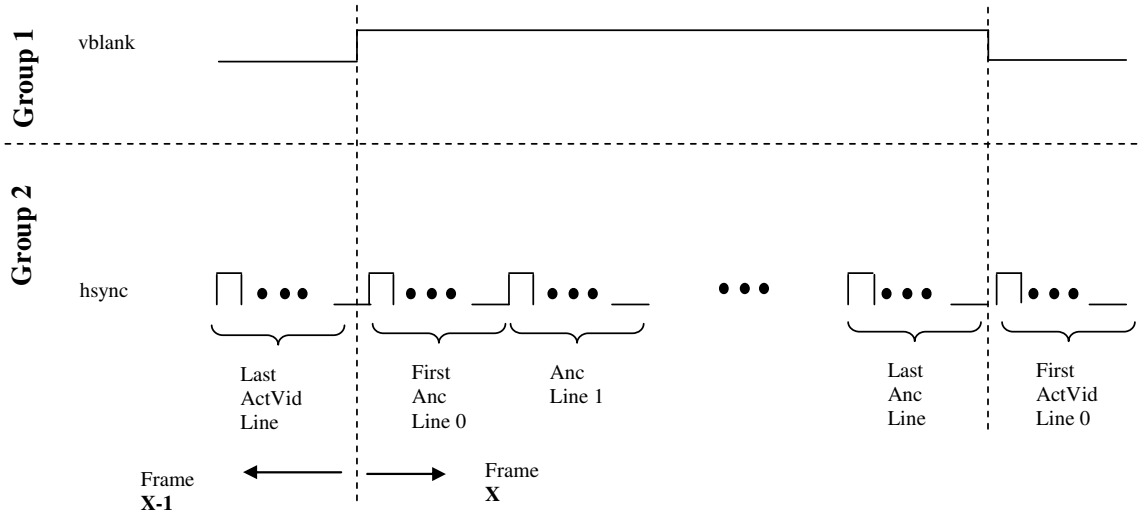
DS : Single Channel non multiplexed mode with AVID and VBLK as control signals



- All lines captured which has ACTVID high
- Active pixels in line captured
- Frame Start: VSYNC low to high
- Line start: ACTVID low to high

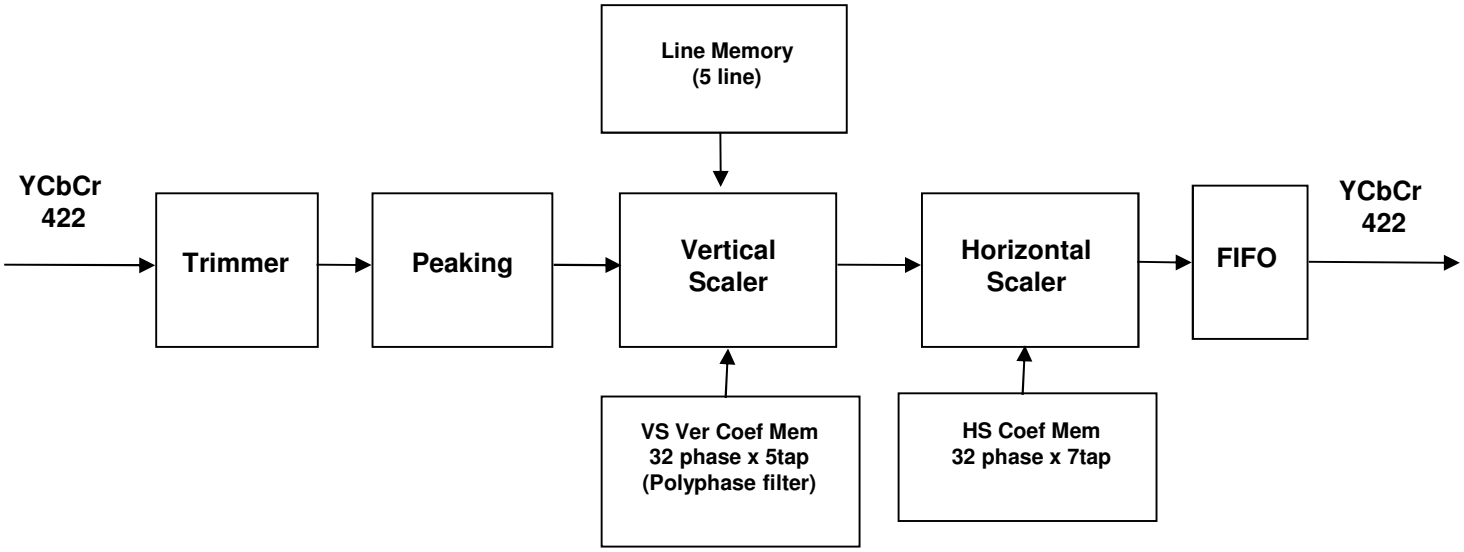
DS : Single Channel non multiplexed mode with AVID and VSYNC as control signals

Contd



ES : Single Channel non multiplexed mode

SC Block


Line Memory
(5 line)
YCbCr
422
Trimmer
Peaking
Vertical
Scaler
Horizontal
Scaler
FIFO
YCbCr
422
VS Ver Coef Mem
32 phase x 5tap
(Polyphase filter)
HS Coef Mem
32 phase x 7tap

SC Features

- Vertical and horizontal up and down scaling
- Polyphase filter upscaling
- Running average vertical down scaling
- Decimation and polyphase filtering for horizontal scaling
- Non-linear scaling for stretched/compressed left and right sides
- Input image trimmer for pan/scan support
- Pre-scaling peaking filter for enhanced sharpness
- Scale field as frame
- Interlacing of scaled output
- Full 1080p input and output support
- Scaling filter Coefficient memory download

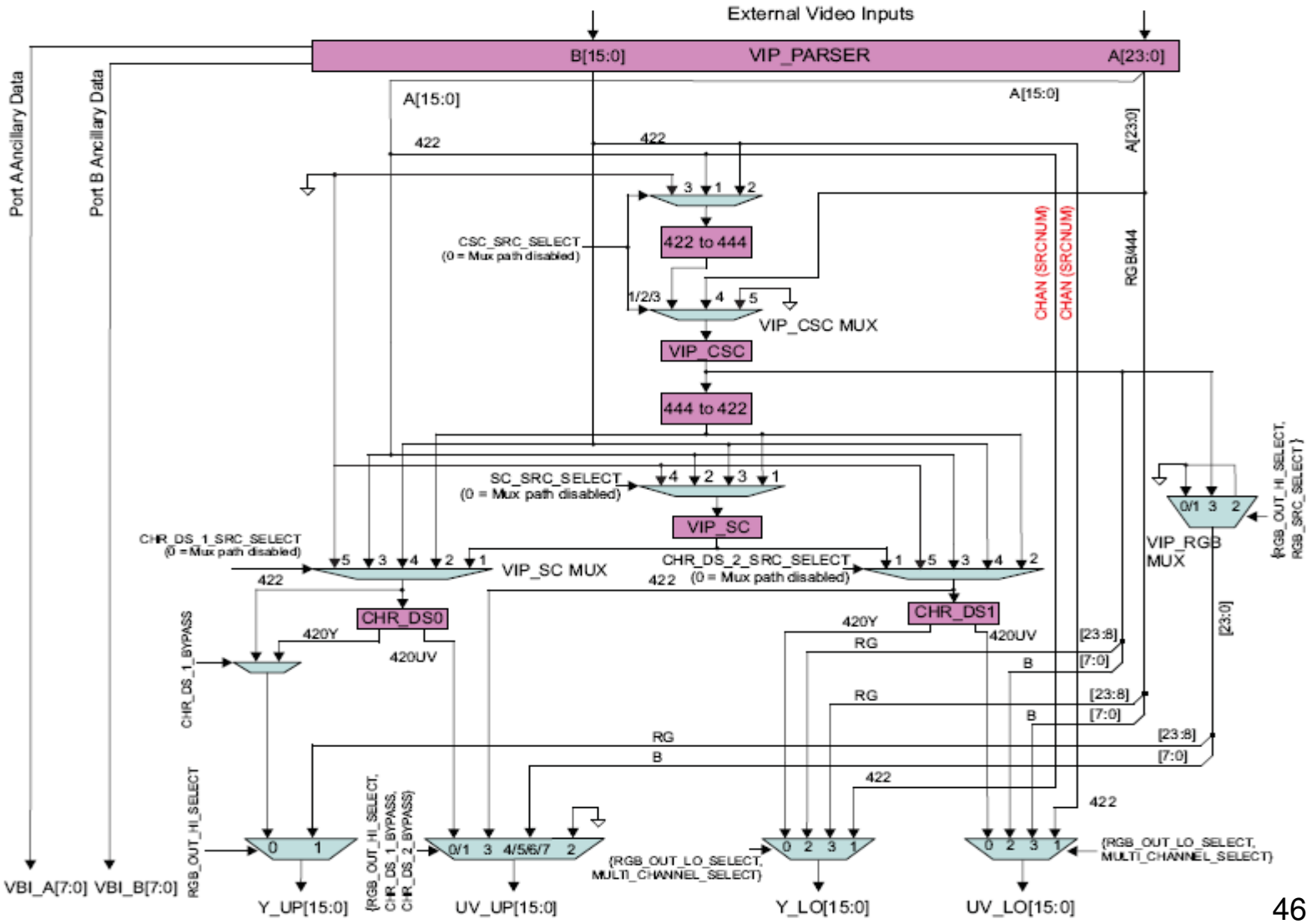
CSC Features

- A fully programmable color space converter. With the programmability, input video data in any color space can be converted to another color space.
- It could convert YCbCr to RGB and vice versa.
- This module could be put by pass as well if no conversion is required.

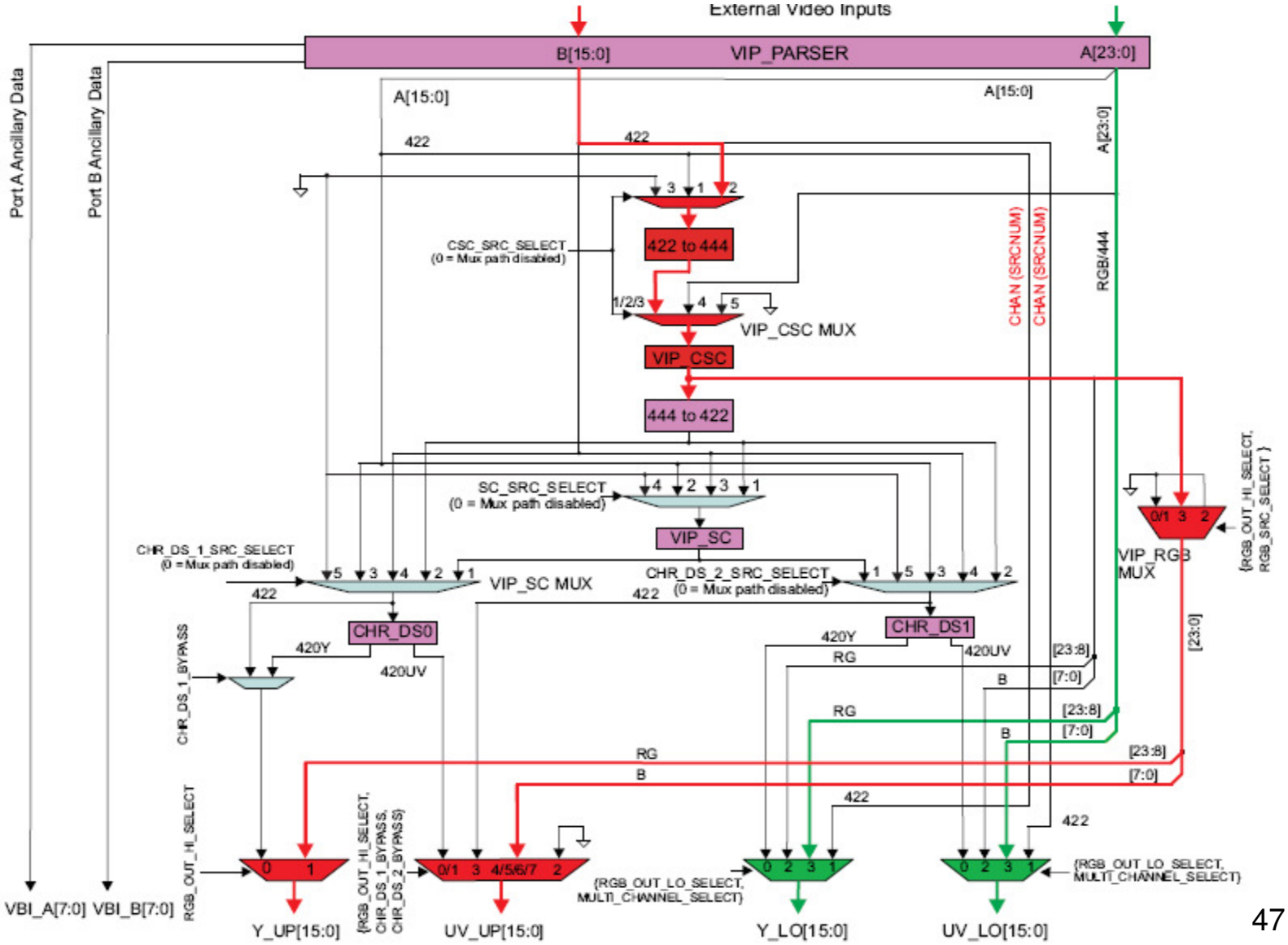
CHR_DS Features

- It is used to downsample picture input in the format 4:2:2 to 4:2:0
- This downsampling is required because typical video encoders expects input in 4:2:0 format before compression
- Down sampling is performed using averaging filter.

VIP Slice Detailed Block Diagram

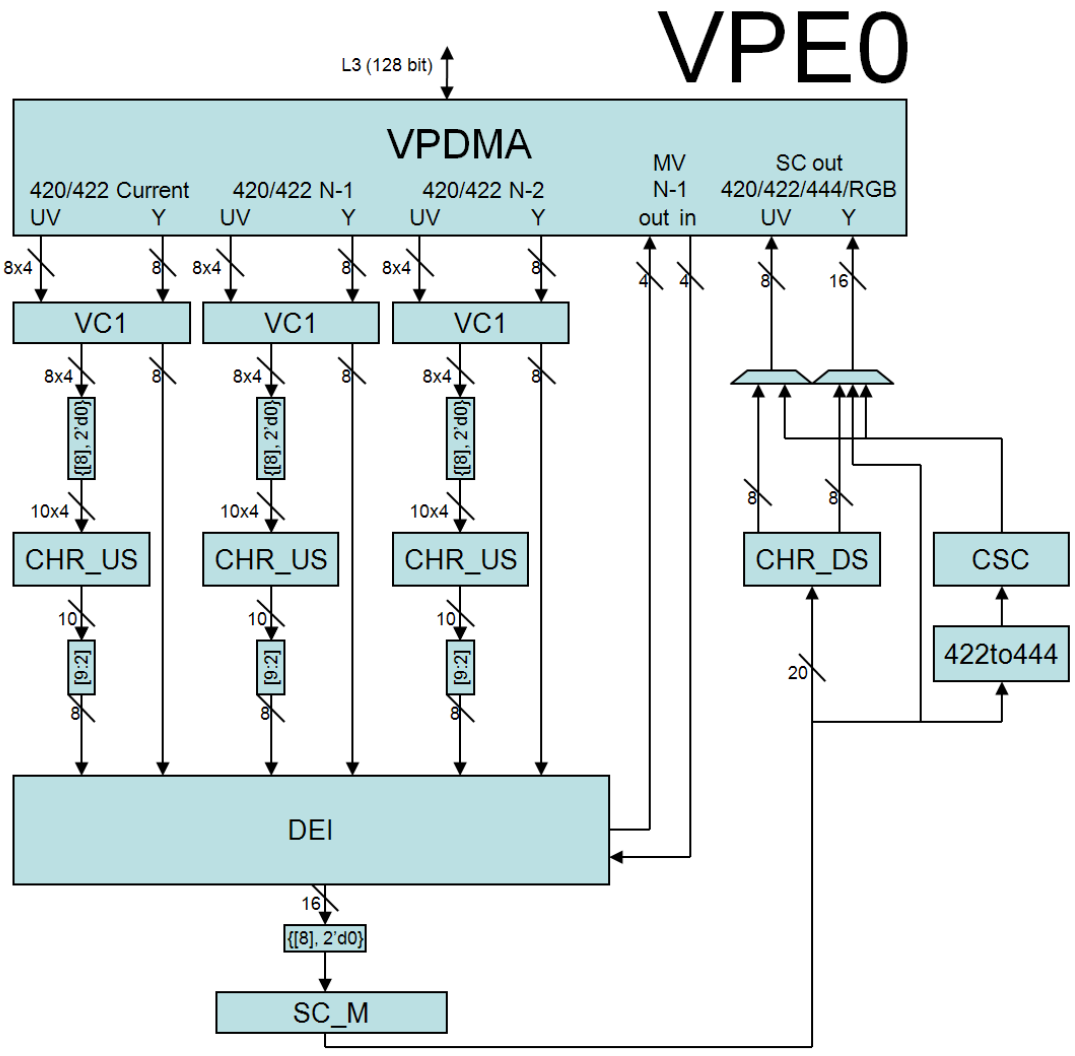


Example Single Channel RGB / YUV422 Capture



VPE Overview
(Video Processing Engine)

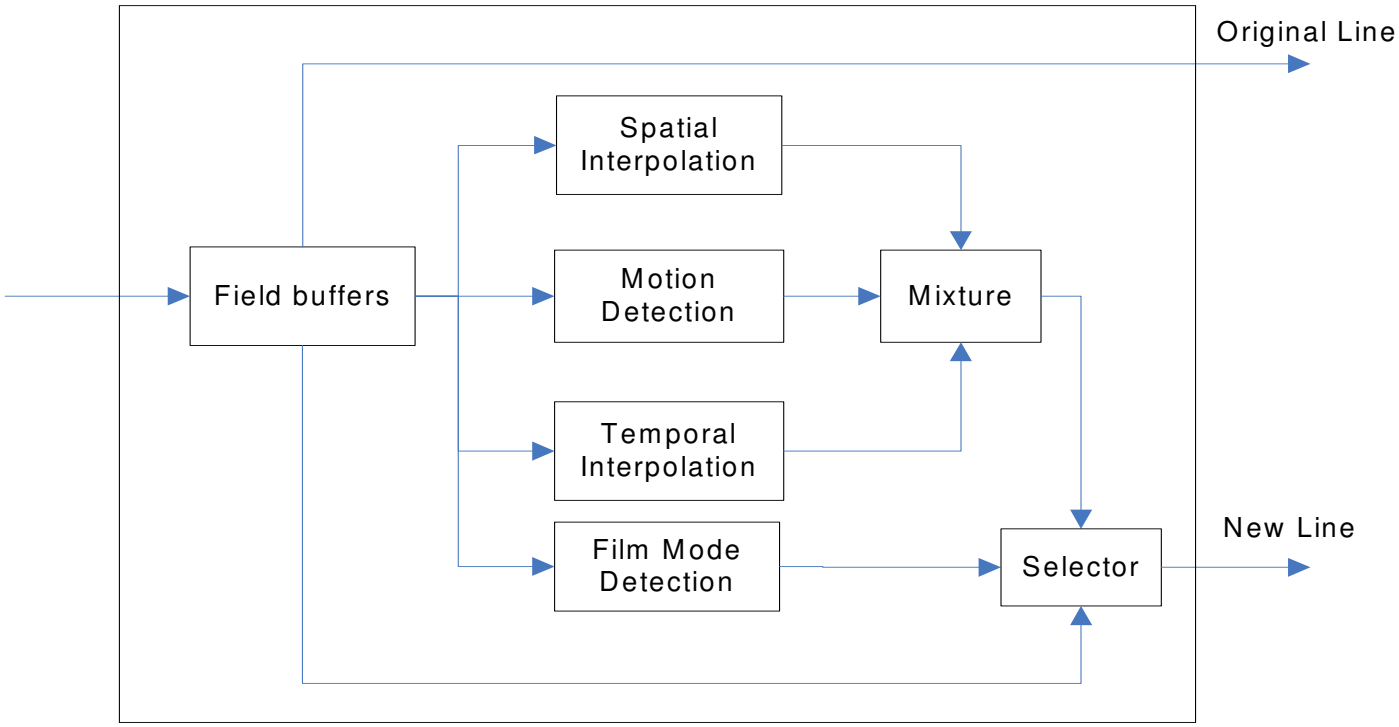
VPE Block Diagram



VPE Features

- Format conversion
  - Inputs: YUV422I, YUV420SP (Uses CHR_US), YUV422SP
  - Outputs: YUV422I, YUV420SP (Uses CHR_DS), YUV422SP, RGB888 (Uses CSC), YUV444I
- Deinterlacing (interlaced to progressive)
  - 4 field motion based algorithm
  - up to two 1080i video sources
  - DEI can be bypassed in case of progressive input
- Scaling from 1/8x to 2048 pixels
- Color space conversion
  - YUV to RGB color space conversion using CSC block
- VC-1 Range Mapping and Range Reduction
- Supports up to 304 MHz clock
- Tiled (2D) input/output

Motion-Adaptive Deinterlacer Block



$$\hat{y}(j,i,n) = \alpha y_{spat}(j,i,n) + (1-\alpha) y_{temp}(j,i,n)$$

DEI Features

- Motion-adaptive deinterlacing
  - Motion detection is based on Luma only
  - 4-field data is used
- Motion Detection (MDT)
  - Examines 3 fields of input video data (luma only) and calculates a 4 bit motion vector to drive the Edge Directed Interpolation Block
- Edge-Directed Interpolation (EDI)
  - Edge detection using luma pixels in a 2x7 window
  - Soft-switch between edge directed interpolation and vertical interpolation depending on the confidence factor
- Film Mode Detection (FMD)
  - 3-2 pull down detection (NTSC style)
  - 2-2 pull down detection (PAL style)
  - Bad Edit Detection (BED)
- Progressive or interlaced bypass mode

Questions?
Thank You