DSS Overview
(Display Sub system)

9th March 2017
Version 1.0

Agenda

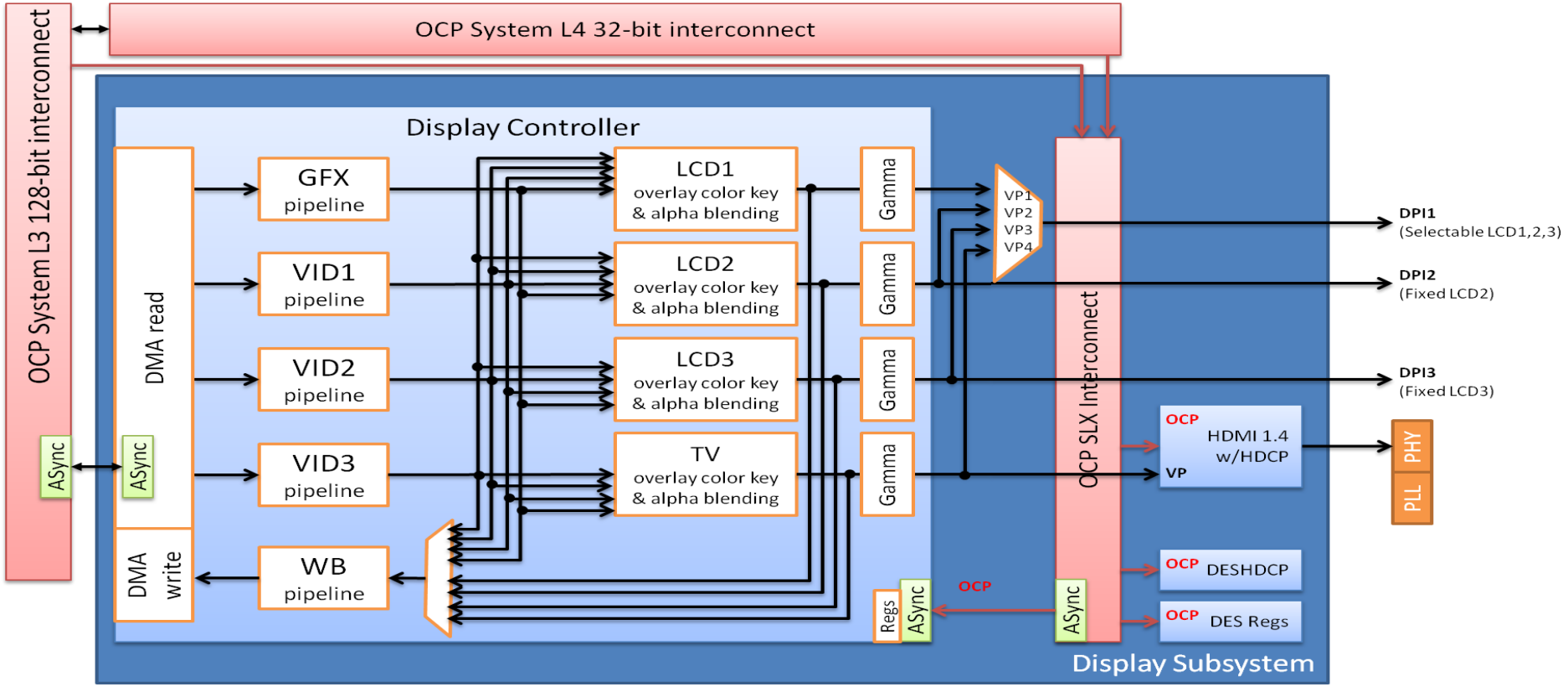
- DSS Hardware Overview
  - Display Controller
  - Pipelines
  - Overlay Manager
  - Interfaces
- DSS Capabilities
  - Scaling Capabilities
  - Overlaying capabilities
- DSS Software
  - FVID2 Driver Architecture

What is DSS?

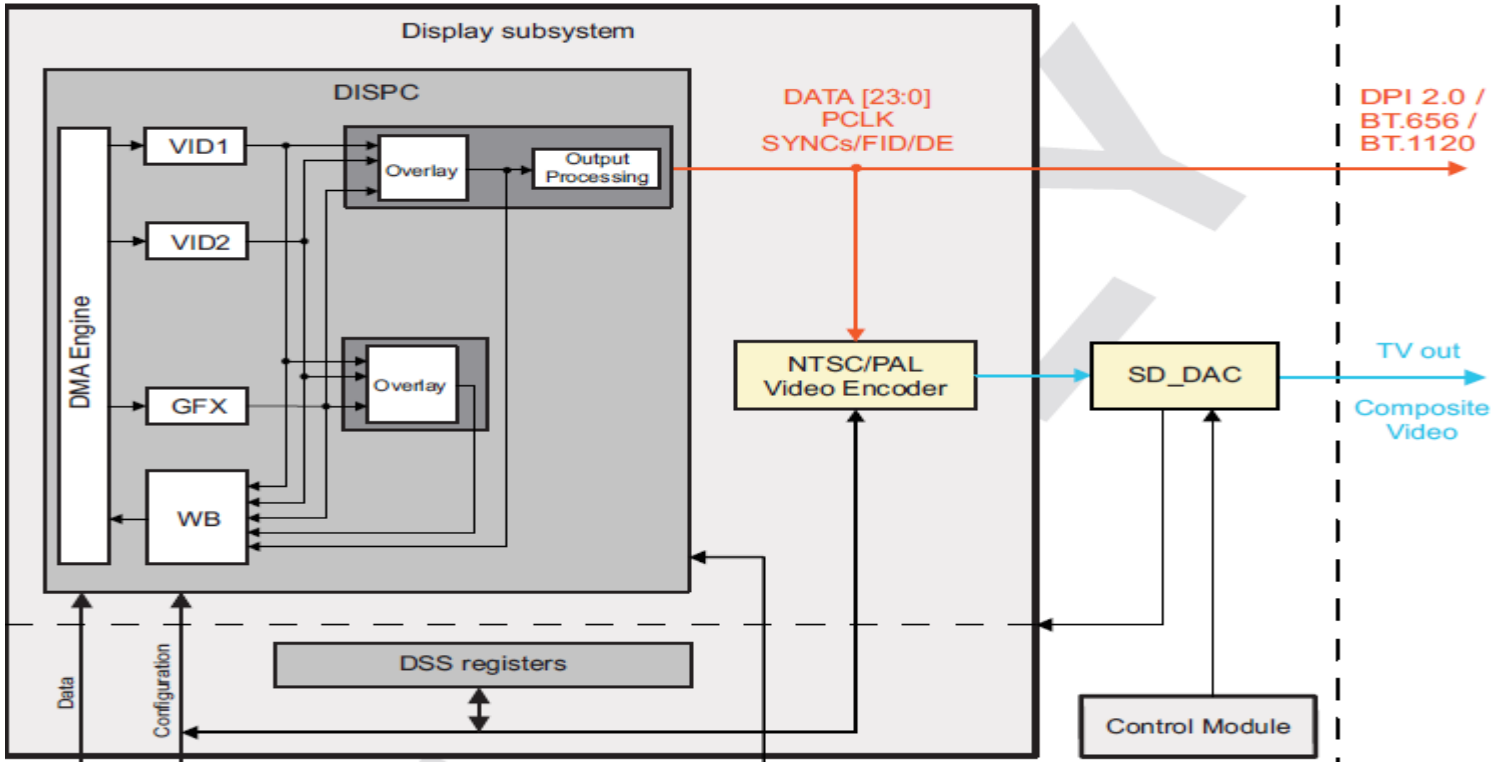
- DSS – Display Sub System for TDA2xx/AM572x/DRA72x/TDA3xx/DRA78x
- Displays video frames from memory to LCD1 or TV (HDMI)
- Composition of graphics and video sources
- Scaling, Color Space Conversion, Blending, Color Keying

DSS Hardware Overview

DSS Hardware for TDA2xx/AM572x/DRA72x



DSS Hardware for TDA3xx/DRA78x



DSS Hardware TDA2xx/AM572x/DRA72x

- 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/DRA78x

- 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)

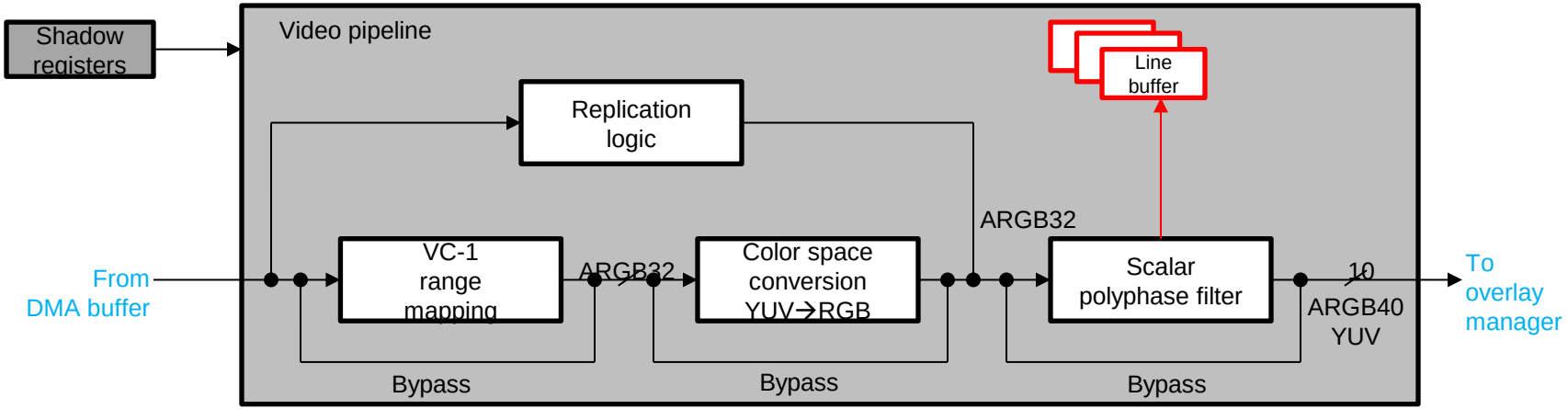
Display Controller

- 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

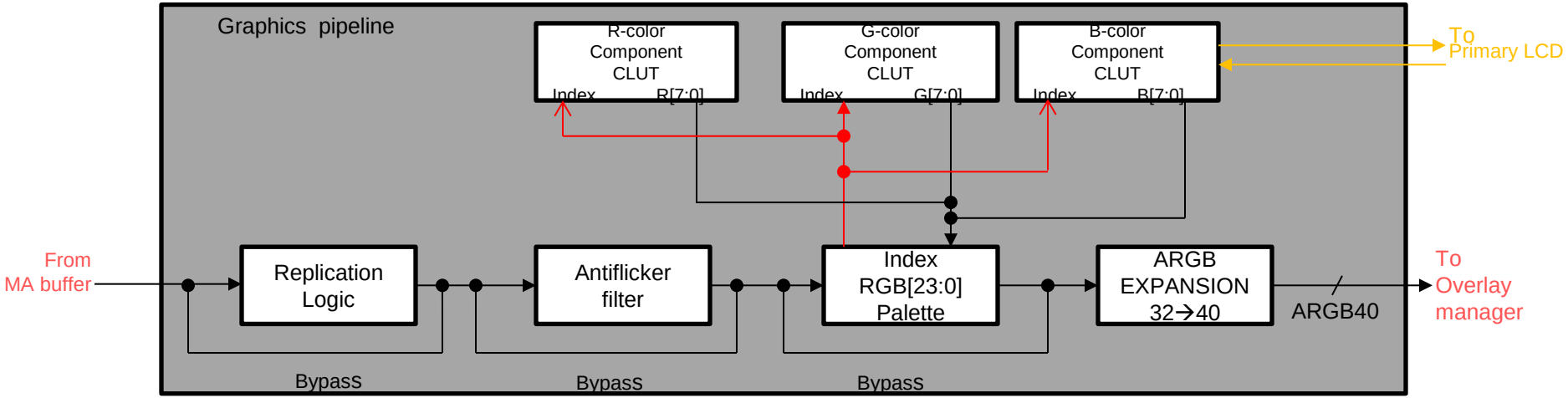
- CSC Unit will convert Video encoded pixels from YUV4:4:4 format into RGB24 or RGB30 format.
- In case of YUV4:2:0 or YUV4:2:2 formats chrominance resampling is required before converting to RGB format.

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

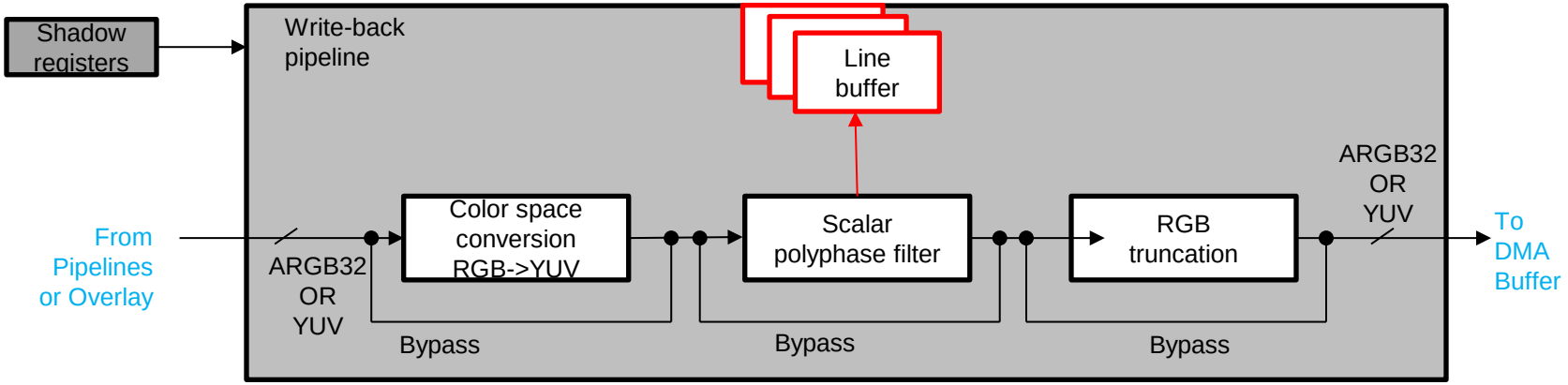
Graphics pipeline

- 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

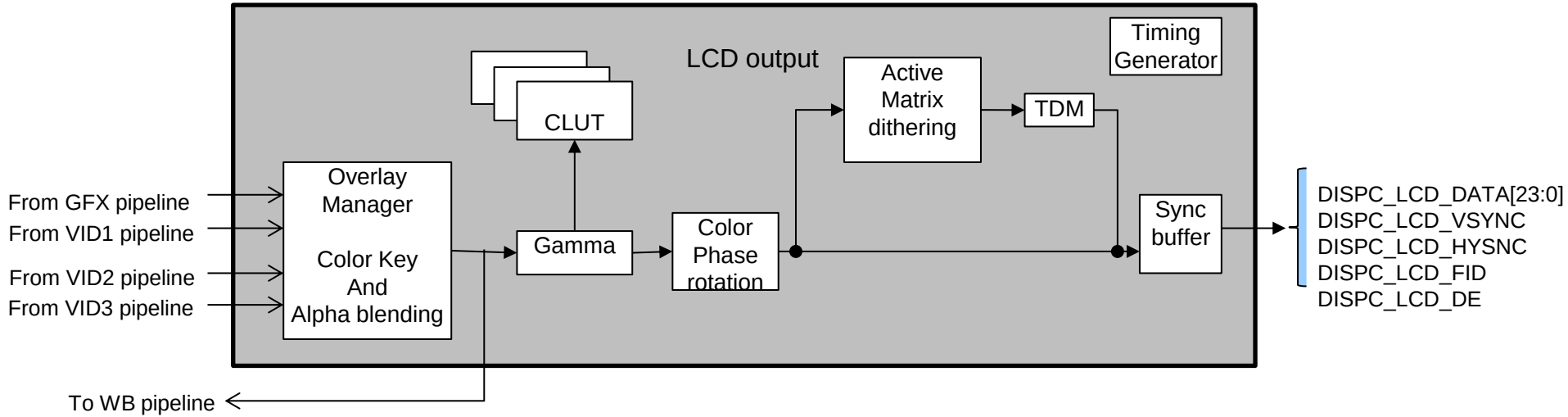


Write Back Pipeline

- 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



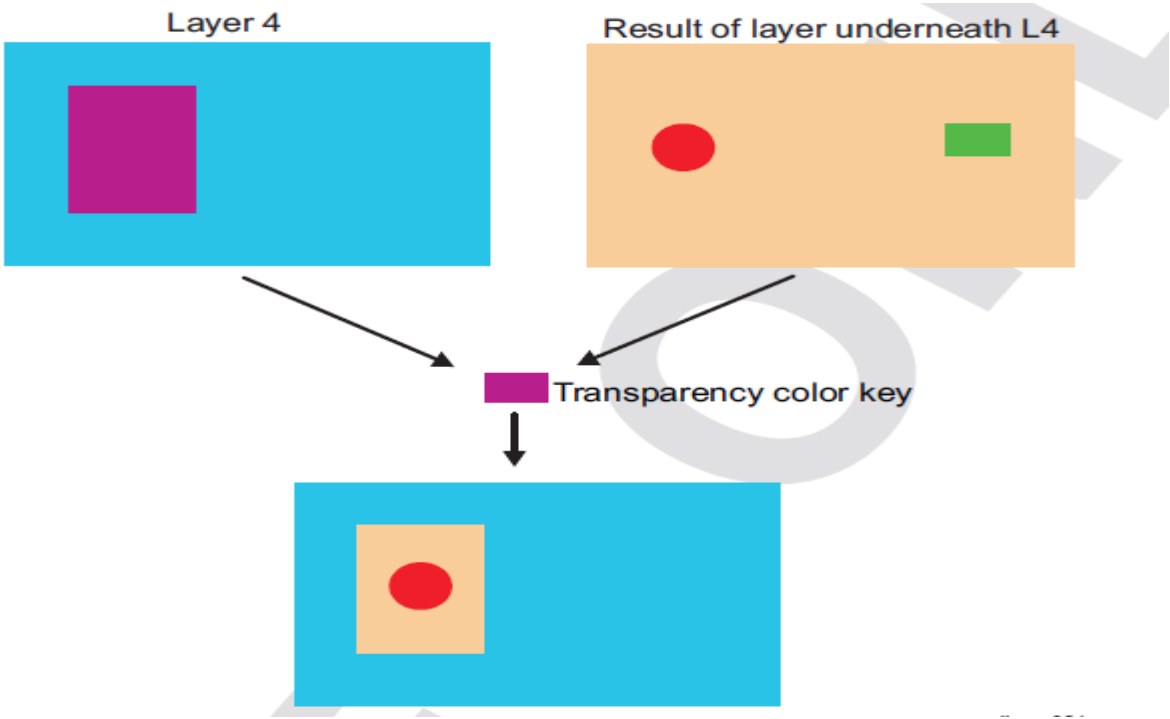
Overlay Manger Capabilities

- 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
- 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

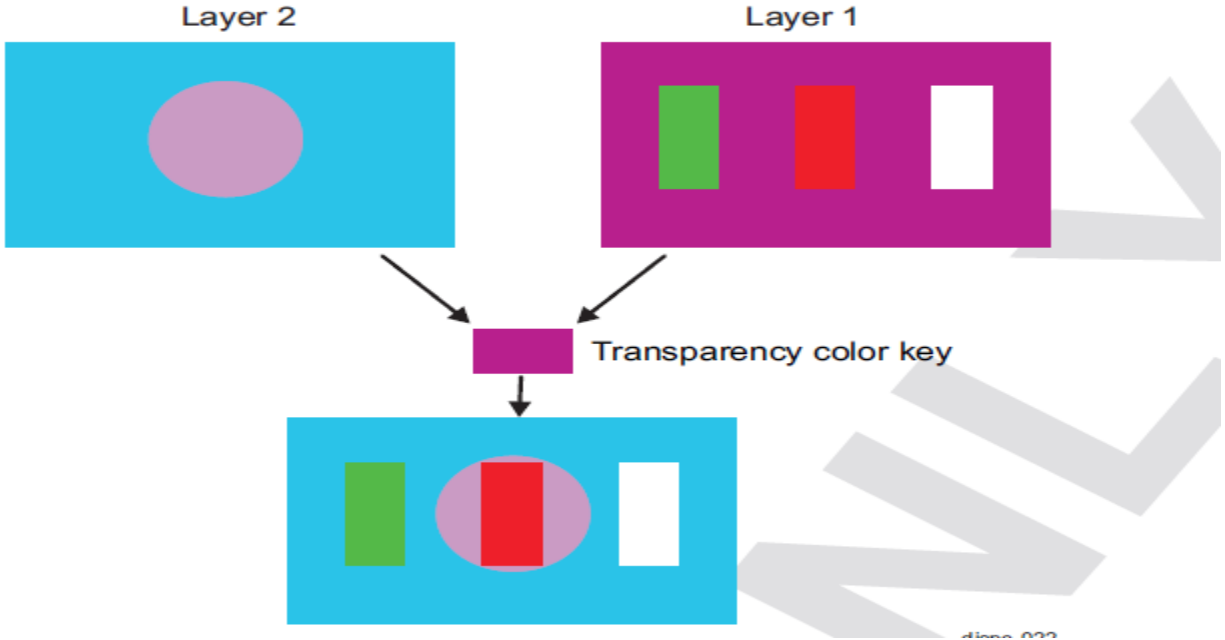
Color Keying Examples

- Source Transparency



Color keying Examples cont..

- Destination Transparency



DSS Driver Overview (based on 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 BIOS6
- What is not FVID2?
  - Not the actual driver
  - Does not define hardware specific APIs and structures

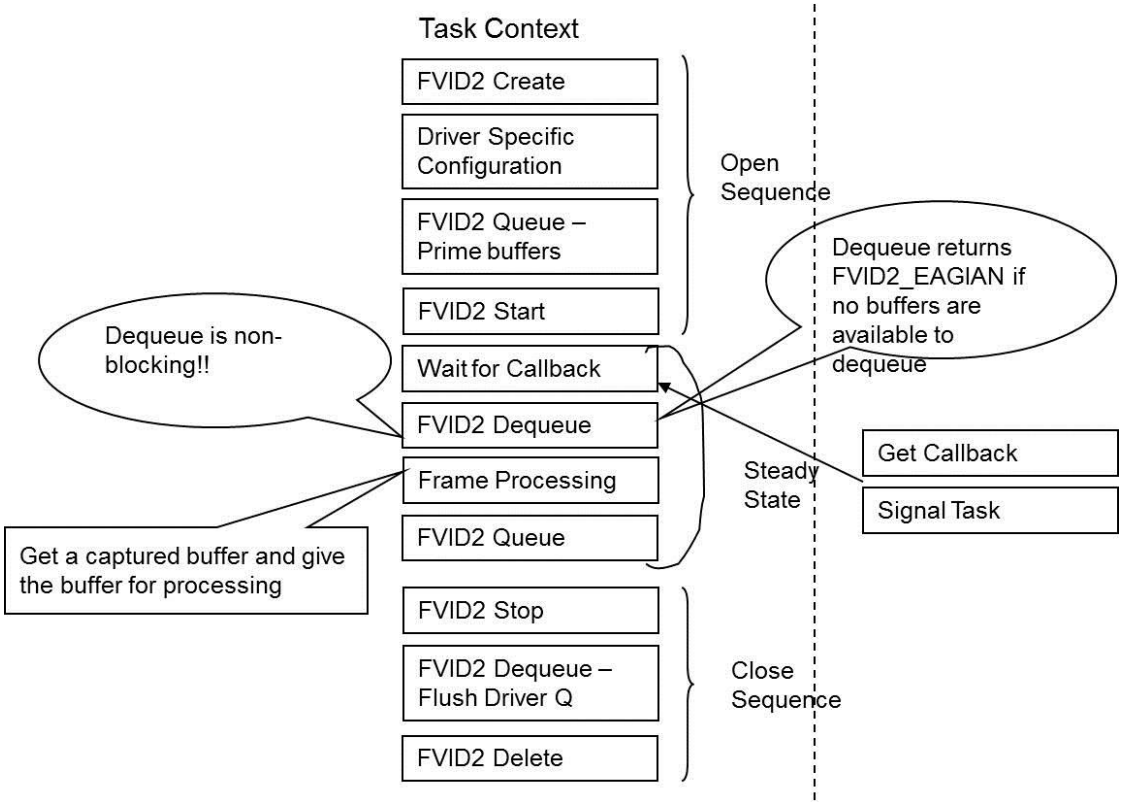
Understanding FVID2 - Interfaces

- *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

Understanding FVID2 – Interfaces Contd.

- *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

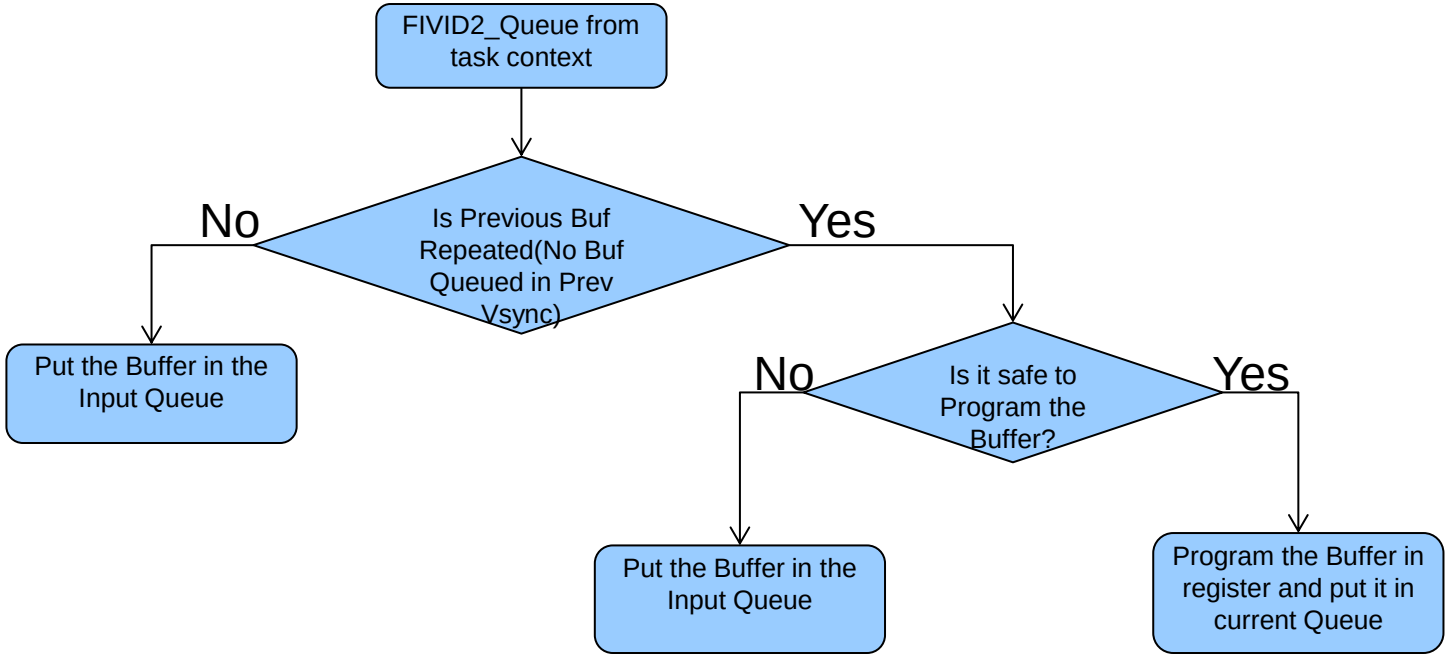
Typical Application Flow



Design

- FVID2_Queue
  - This will put the buffer into Input Queue.
- FVID2_DeQueue
  - This will take out the buffer from the output Queue
- Vsync ISR
  - Call back will be given to application if there is any frame in the output Queue or if periodic callback is enabled.
  - It will program the base address of the frame present in the input queue in shadow registers and the frame is put in the current Queue.
- Checks
  - If application queues from the task context and the previous buffer is repeated then driver checks if its safe to program the buffer, if its safe then it will updated the registers and put it in the current queue, else it will put the buffer in the input queue.

Flow chart of Queue



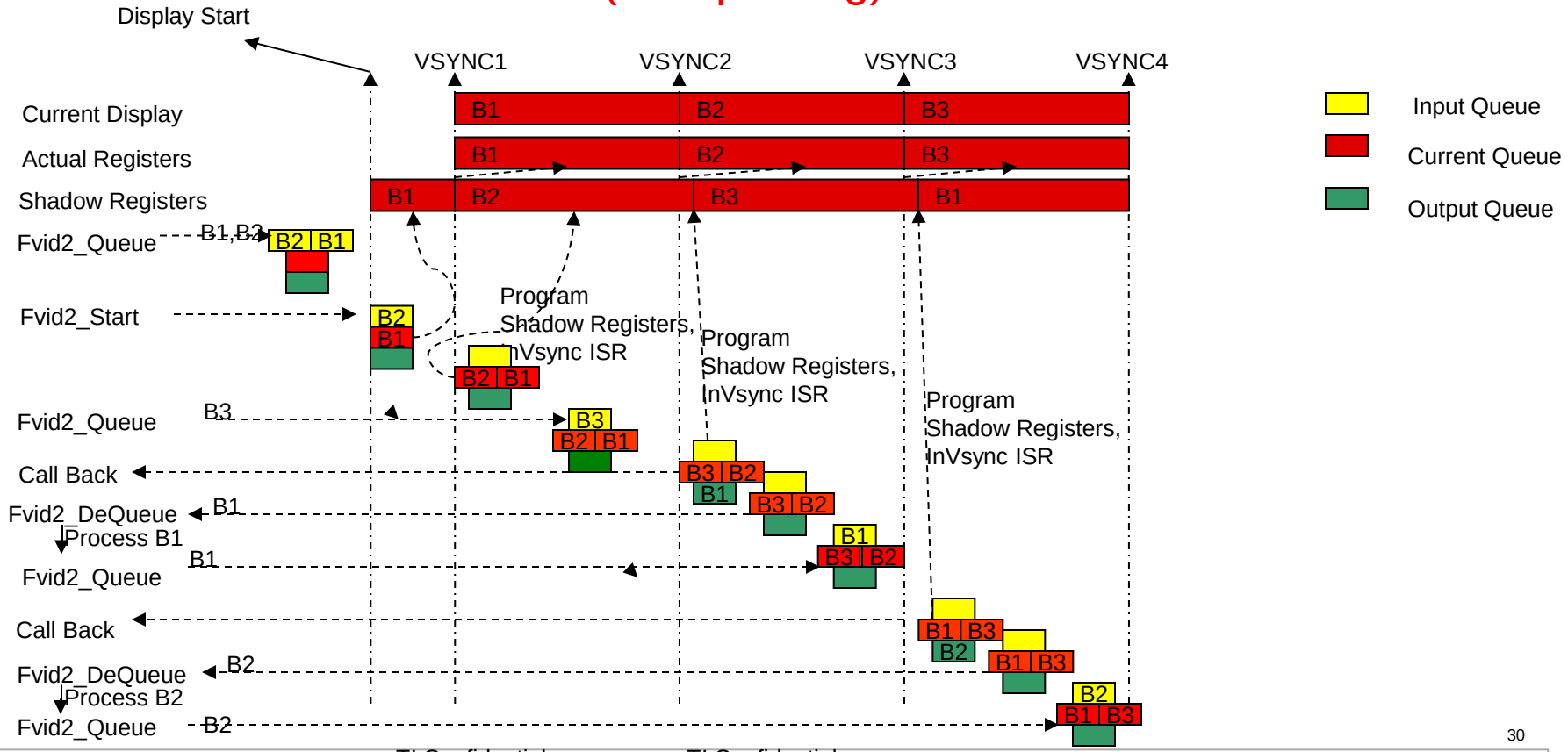
Buffer Queue latency

- Driver latency to program the buffer to DSS = Code execution time from APP queue to programming Registers (T1) + 5 line of display rate (T2).
- With Vayu EVM, T1 is measured to be around 20 micro seconds.
- The total latency comes around 180 us for 800x480 @ 60 FPS display. So if any buffer is queued 180 us before the Vsync then the buffer will be displayed in the next frame period.
- T2 in micro seconds for different resolution
  - - 800x480@60fps -→ 158.25 us
  - - 1280x720@60fps --→ 107.74 us
  - - 1920x1080@60fps -→ 74.07 us
- (Reason for 5 lines check)
  This check is required so that the driver won't program the buffer address around the display VSYNC period. Doing so would result in DSS HW not accepting the programmed buffer resulting in frame drop.
- Note - This measurement is done with the stand alone display application. In fully loaded system the interrupt latency will add to it.

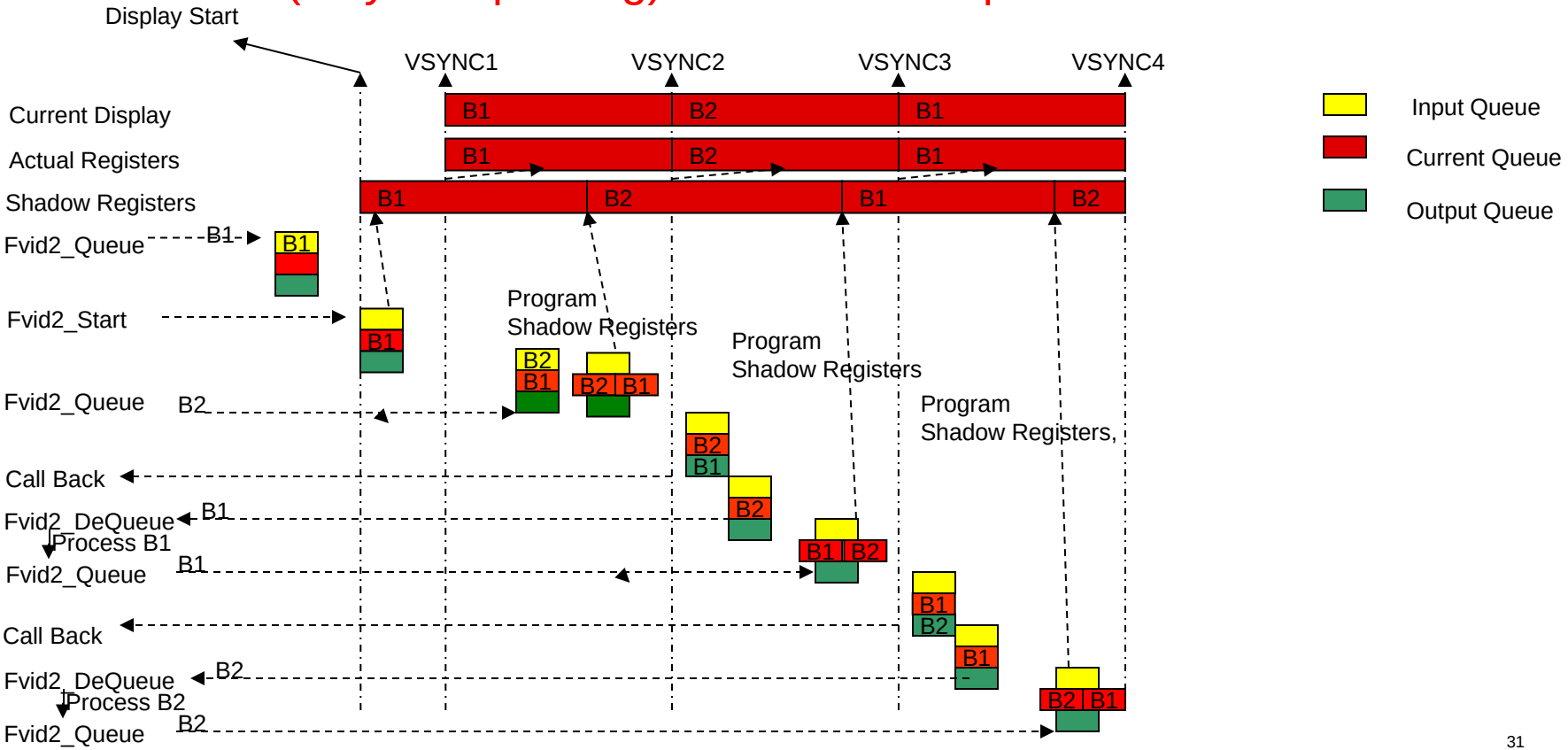
Background

- Some DISPC registers are termed *shadow registers*. A shadow register change has no direct effect on the configuration of the DISPC
- The registers are shadow registers let software change the values of the registers at any time
- When all the registers for a given configuration are into the registers, software must set 1 bit (GoBit) only to validate the configuration
- When hardware reaches the end of the current frame and sees that the bit field has been set by software, the new configuration is now the configuration used by the hardware.
- In some corner cases, even though application has queued a new buffer before Vsync event, due to software overhead the new frame address might be programmed after the actual VSYNC event. In such cases previous frame is repeated.

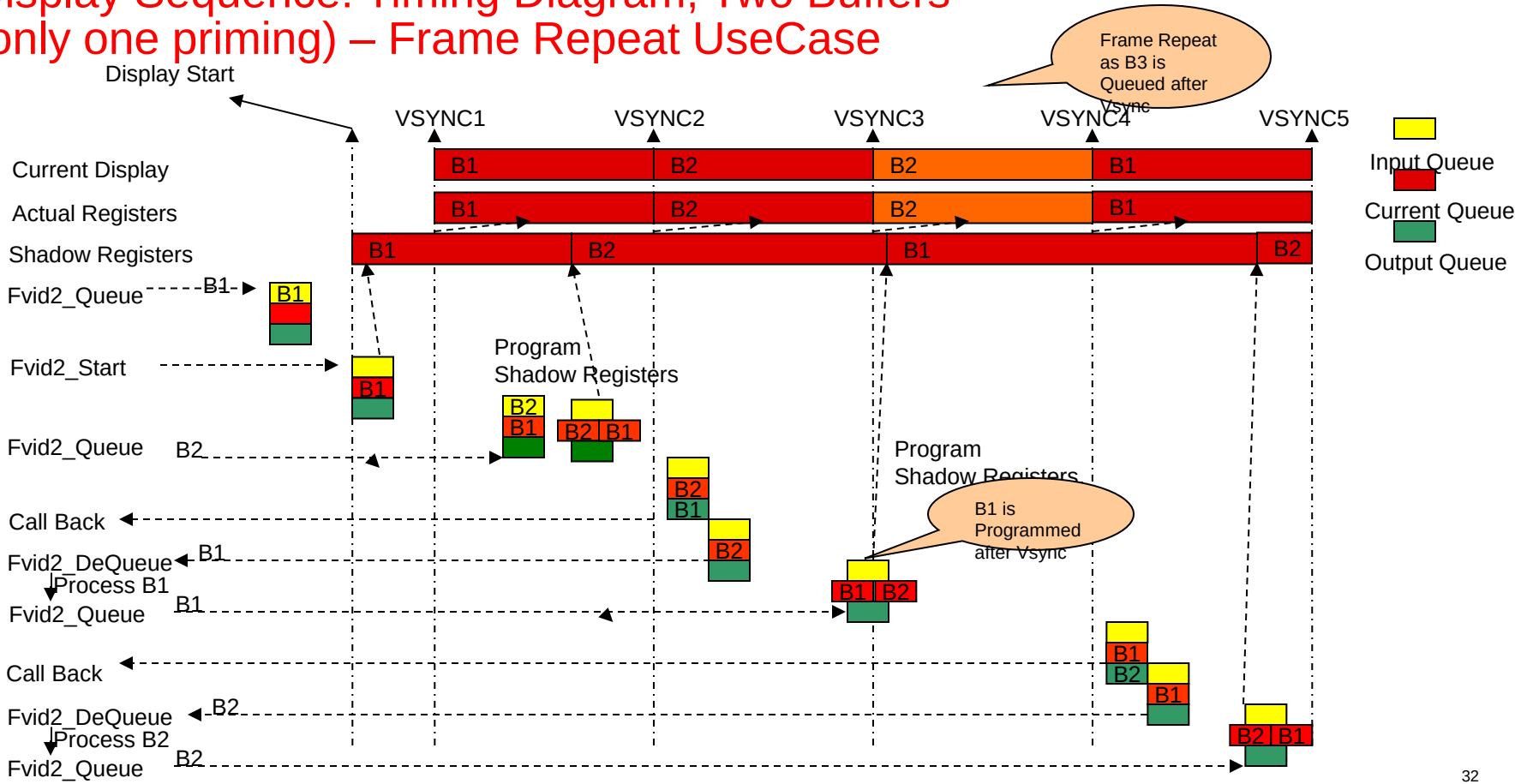
Display Sequence: Timing Diagram, Three Buffers (Two priming)



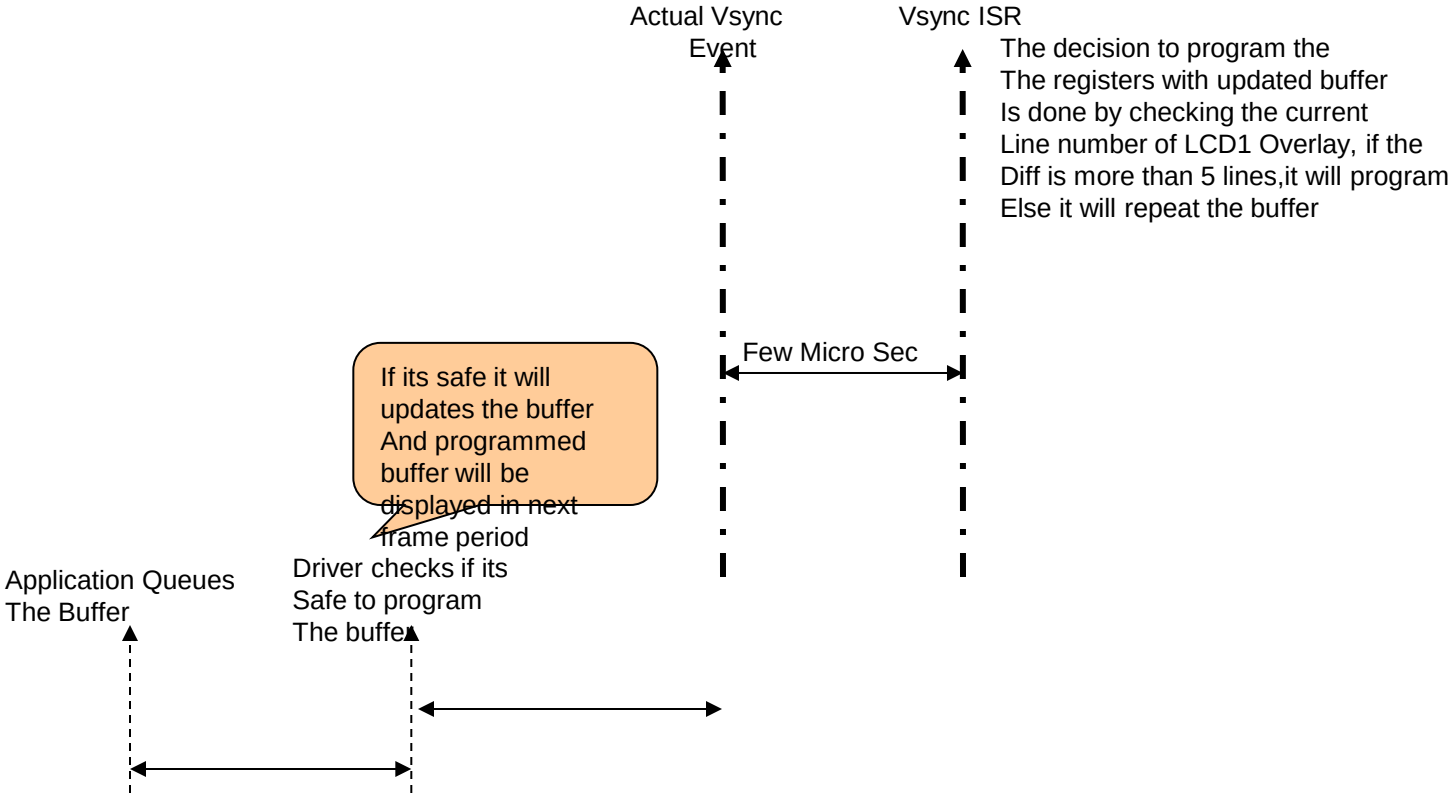
Display Sequence: Timing Diagram, Two Buffers (only one priming) – No frame Repeat case



Display Sequence: Timing Diagram, Two Buffers (only one priming) – Frame Repeat UseCase



Corner Case, where frame is queued Near Vsync Boundary



Questions?
Thank You