Intel FPGA VIP webinar series

Session 2
Strategies to debug a VIP pipeline

Francisco Perez
Intel FPGA Field Applications Engineer
v.1 – November 2020

Video processing on FPGAs made easy

Objectives

- What resources are available to develop Video Processing solutions?
- From the basics: step through an incremental series of example designs
- End2End flow demonstration: hardware architecture design, software development & debug
- Sessions will be recorded. Exercise manuals and project files will be available for on-demand consumption

Content available in the GitHub repo:
https://github.com/perezfra/VIP_webinars_Intel_FPGA

Webinar Series

Soft start -> Increasing Complexity

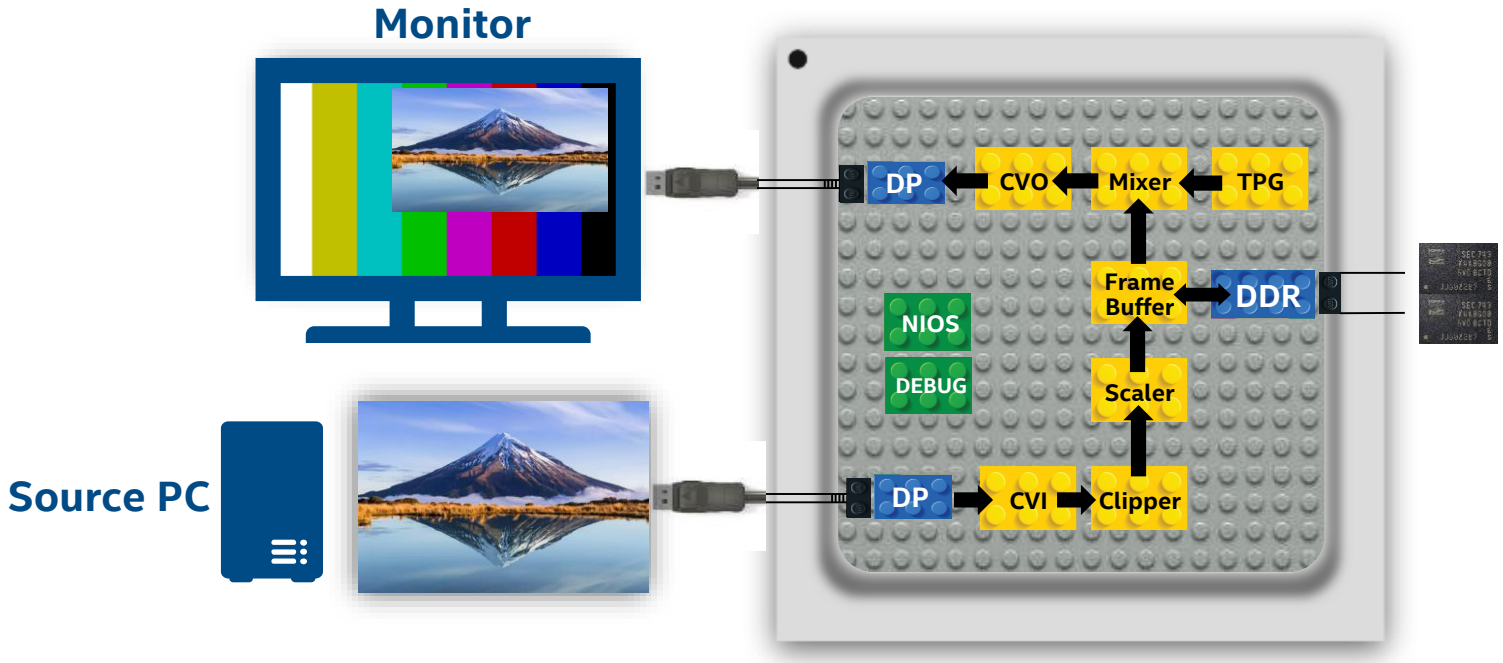
1. Building a video processing pipeline (Oct, 14th and 21st)
   - 1.1- DisplayPort loopback example design implementation
   - 1.2- Building our first video pipeline using VIP suite cores
   - 1.3- Complete End2End VIP pipeline
2. Strategies to debug a VIP pipeline (Nov, 4th)
   - Overview of system level considerations and key video concepts
   - Overview of Avalon-ST Video protocol
   - Bring up your pipeline using System Console
3. Integrate a simple custom component (Nov, 19th)
   - How to add your “secret sauce” to the application
   - Step flow on how to develop a custom component compliant with VIP
4. Adding On Screen Display overlay (Dec, 16th)
   - Use a lightweight graphic library with Nios
   - Add text and graphic content overlay on top of your live video

https://github.com/perezfra/VIP_webinars_Intel_FPGA

Building a Video Processing Pipeline

Previous sessions

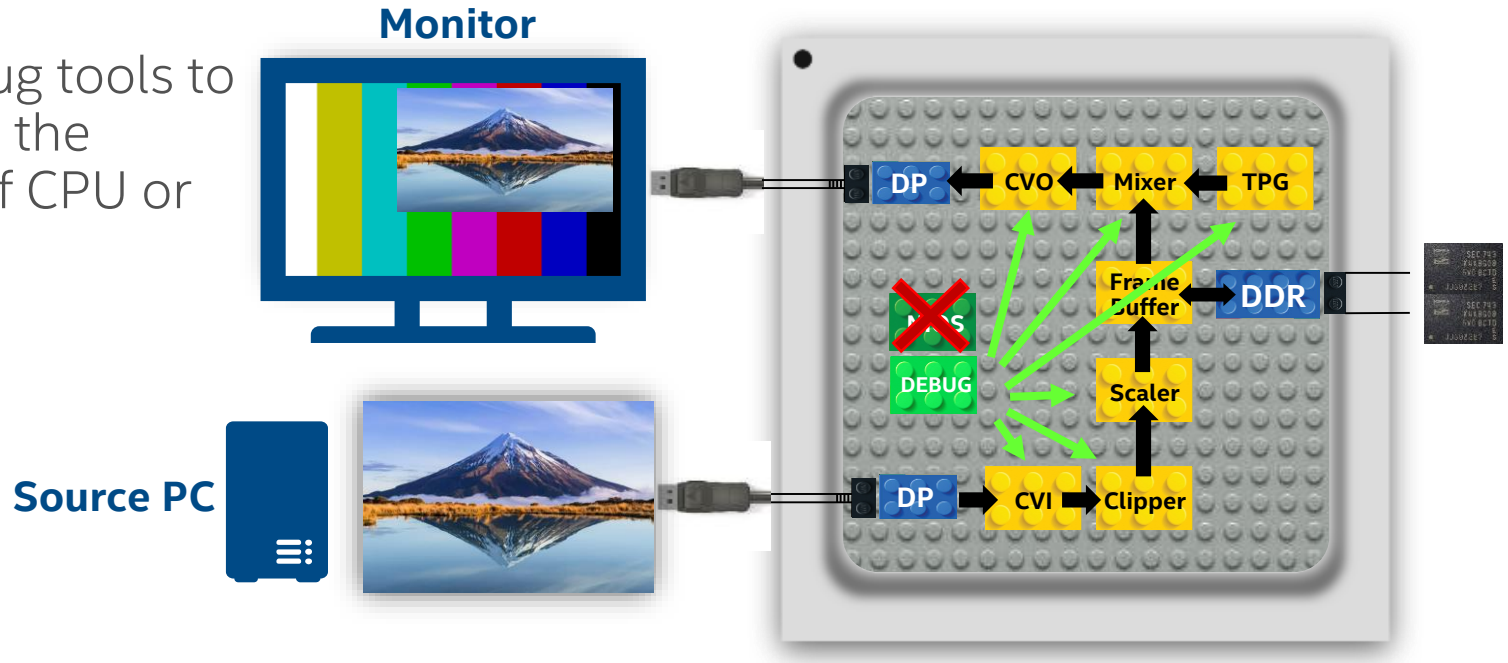
1. Build a video pipeline using VIP cores to manipulate the video (cropping, scaling and mixing with a background) connecting external DisplayPort source and sink



Building a Video Processing Pipeline

Session 2 – Nov, 4th

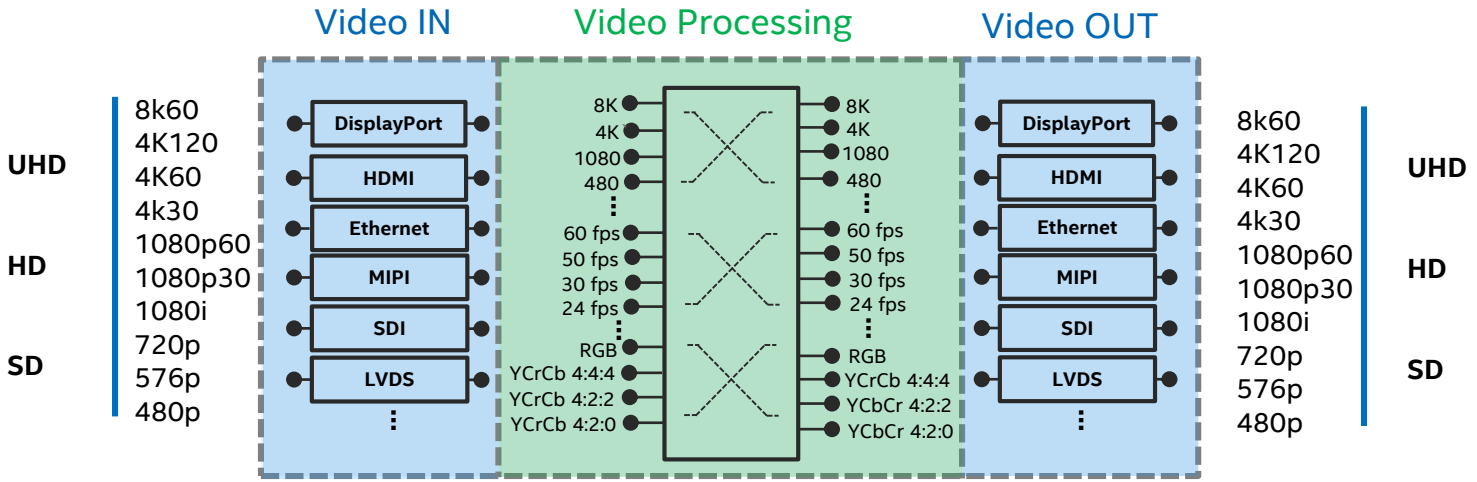
1. Build a video pipeline using VIP cores to manipulate the video (cropping, scaling and mixing with a background) connecting external DisplayPort source and sink
2. We are using system level debug tools to identify problems and bring up the pipeline, even in the absence of CPU or software



Session 2 – Debug VIP pipelines

Key Video Concepts – High level topology

Video Application topology



- Connectivity
  - Used to bring video on and off chip
  - Collects transceivers, PLLs and protocol blocks into RX and TX cores
  - Protocol typically drives/receives a variant of clocked video format
  - Examples: DisplayPort, HDMI, SDI, MIPI, LVDS …

- Video IP
  - The Video IP sits between RX and TX connectivity
  - Provides the means to modify: Video resolution, format and rate
  - Combine video streams
  - Manipulate video streams

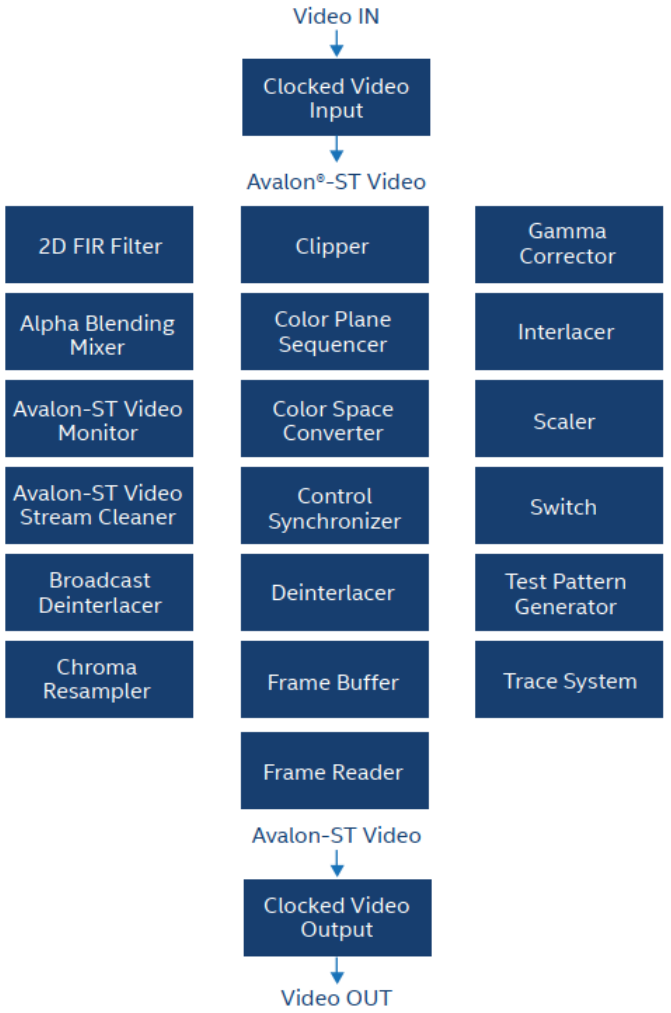
Video Connectivity IPs

- DisplayPort
  - Compliant with 1.4 -> 2.0 in dev
  - Up to 4-ch MST Support, up to 8-ch audio
  - Link Training quality analysis
- HDMI
  - Compliant with 2.0 and prev
  - Supporting new 2.1 FRL
  - Deep color mode, up to 8-ch audio
- HDCP
  - Certified 1.4/2.3 with HDMI and DP
- SDI
  - Multistandard up to 12G-SDI

https://www.intel.com/content/www/us/en/broadcast/products/programmable/applications/connectivity-solutions.html

8K-Ready Video and Image Processing Suite

- Collection of 20+ IP functions with support for all Intel FPGAs
- VIP Suite II provides a design philosophy for rapid new design creation and easy integration of custom value-add features
- Easy integration with video connectivity IP cores, such as: HDMI, DisplayPort, 12G-SDI, SMPTE 2110 and MIPI
- Supports a wide range of resolutions, fps, bits per color
  - 1080p/4K/8K, HDR ready, 120+ fps, 16 bpc
- Visually exceeds most of the ASSPs



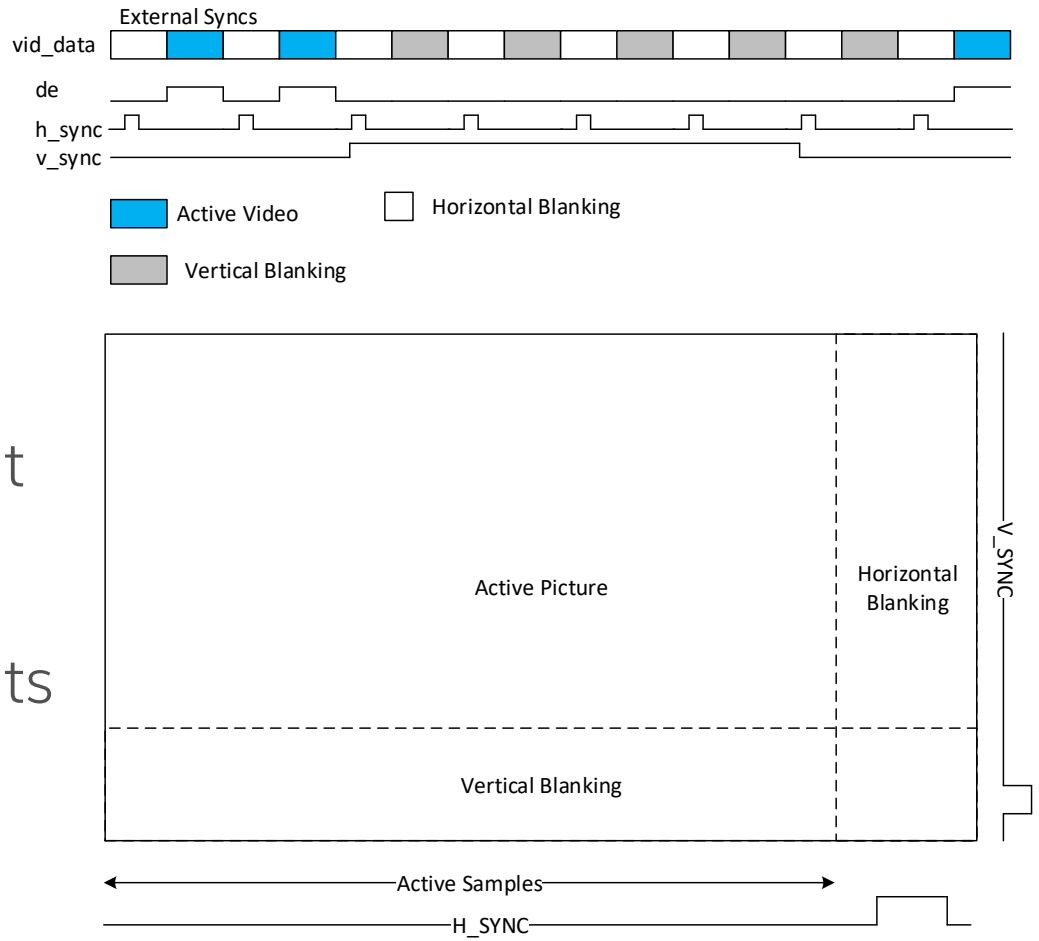
https://www.intel.com/content/www/us/en/programmable/products/intellectual-property/ip/dsp/m-alt-vipsuite.html

Session 2 – Debug VIP pipelines

Key Video Concepts - Transport conversion

Key Concept - Clocked Video Overview

- Connectivity cores (HDMI/DP/SDI) output video in a Clocked Video Format
- A combination of horizontal and vertical synchronization signals along with a data enable signal indicate line, frame and valid data boundaries respectively.
- On reception, data must be accepted when it is presented or it will be lost.
- On transmission, if valid data isn't available during the active picture region sample points will be missed and the output image distorted.
- There is no mechanism to stall data.



Key Concept – Packetized Video

Packetized video used in 2 ways

- Video Transport
  - Take advantage of modern networking technology to extend transport range and options
  - HDB-T encapsulates HDMI in ethernet packets
  - SMPTE 2022 encapsulates the entire SDI protocol in IP packets
  - SMPTE 2110 directly encapsulates video in IP packets.
- Video processing
  - Simplifies many aspect of flow control as video is manipulated
  - Avalon-ST Video is an example

Avalon-ST Video

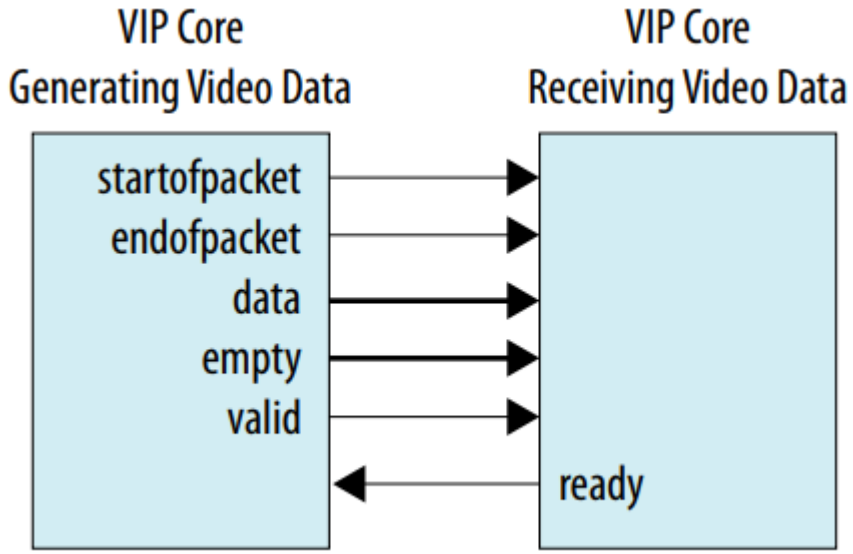
Introduction

- The Video and Image Processing IPs conform to the Avalon streaming video standard of data transmission.
- This standard is a configurable protocol layer that sits on top of the Intel Avalon streaming standard.
- The standard comprises video packets, control packets, and/or user packets.

Avalon-ST Video

Introduction

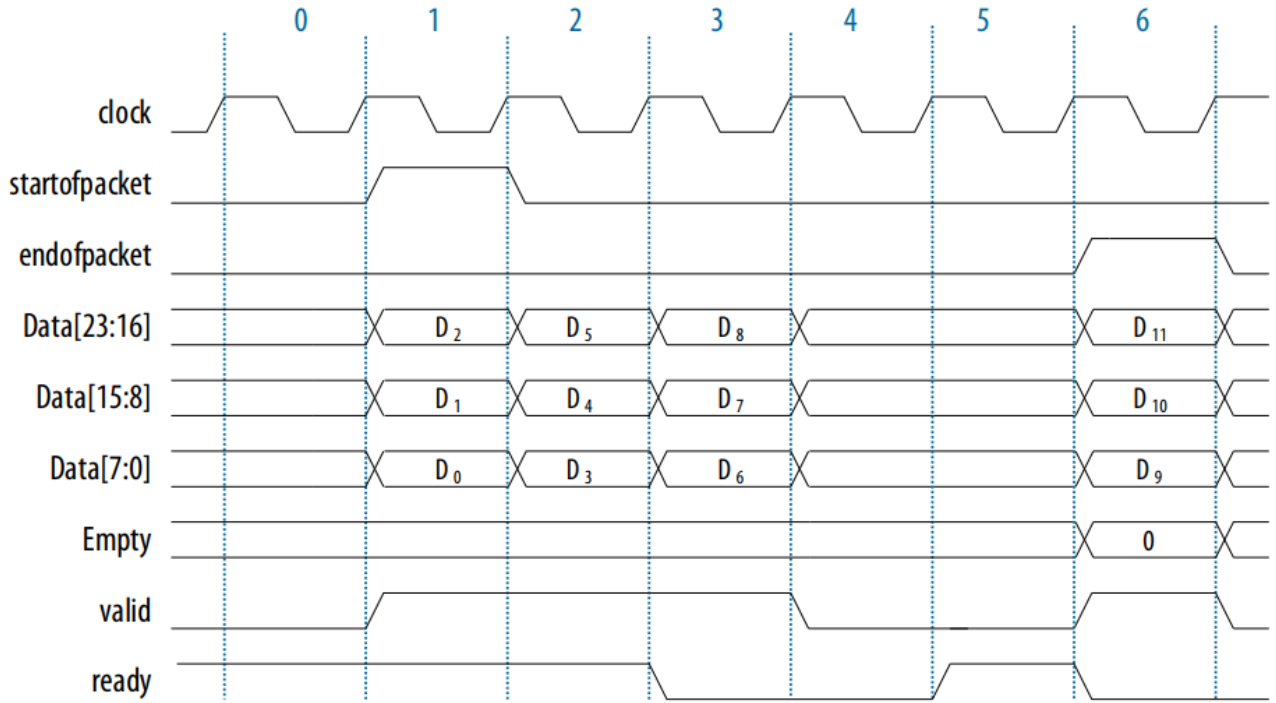
- VIP cores are connected using packet control signals, backpressure is controlled by the **'ready'** signal.
- The individual video formats supported (i.e. NTSC, 1080p, UHD 4K) depend primarily on the configuration of the Avalon streaming video standard and the clock frequency.
- The IPs may transmit pixel information either in sequence or in parallel, in RGB or YCbCr color spaces, and under a variety of different chroma samplings and bit depths.



Avalon-ST Video

Avalon-ST Video packet transmission

- A “ready latency” of 1 is used for Avalon streaming video.
- The receiving video sink drops its ready signal in cycle 3, to indicate that it is not ready to receive any data in cycles 4 or 5.
- As the ready signal returns high in cycle 5, the video source data in cycle 6 is safely registered by the sink



Avalon-ST Video

Avalon-ST Video Packet Type Identifiers

Type Identifier D0[3:0]	Description
0x0 (0)	Video data packet
0x1–0x8 (1–8)	User data packet
0x9–0xC (9–12)	Reserved
0xD (13)	Clocked Video data ancillary user packet
0xE (14)	Reserved
0xF (15)	Control packet

The type of packet is determined by the lowest 4 bits of the first symbol transmitted.

Avalon-ST Video

Control Packet

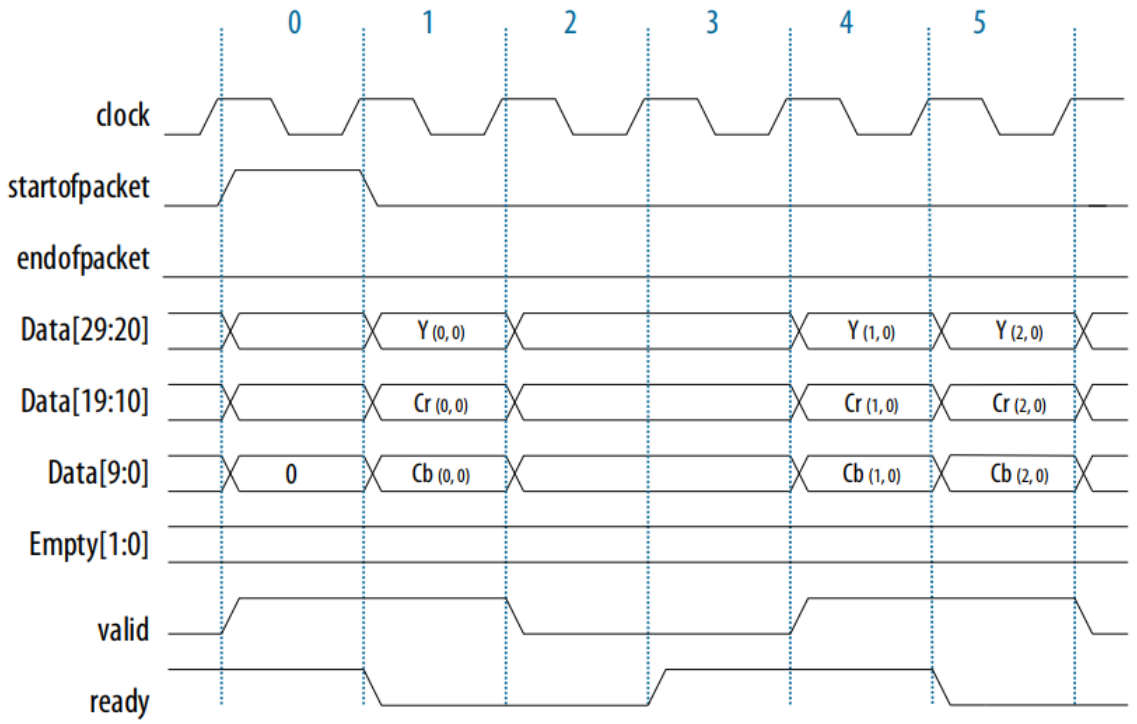
- A control packet is identified when the low nibble of the first symbol is set to decimal 15 (0xF).
- An Avalon-ST Video control packet comprises the identifier nibble, and 9 other nibbles which indicate the height, width and interlacing information of any subsequent AvalonST Video packets.

Nibble	
D_0	Width[15:12]
D_1	Width[11:8]
D_2	Width[7:4]
D_3	Width[3:0]
D_4	Height[15:12]
D_5	Height[11:8]
D_6	Height[7:4]
D_7	Height[3:0]
D_8	Interlacing[3:0]

Avalon-ST Video

Video Packet

- A video packet is identified when the low nibble of the first symbol is set to decimal 0
- A video packet has a length equal to a full frame.
- Color channel data for RGB video packets is transmitted in the order of Blue, Green, Red (or Cb, Cr, Y)



Avalon-ST Video

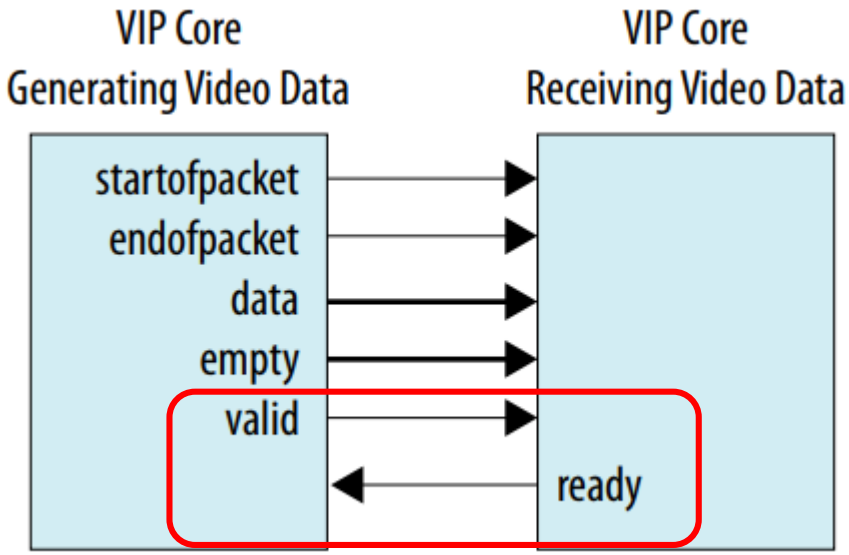
Operation

- Most Avalon-ST Video compliant VIP IP cores require an Avalon-ST control packet to be received before any video packets, so that line buffers and other sub-components can be configured.
- When a VIP IP core receives an Avalon-ST Video control packet, the IP core decodes the height, width, and interlacing information from that packet and interprets any following Avalon-ST Video packets as being video of that format until it receives another control packet.

Avalon-ST Video

Stall Behavior

- The Video and Image Processing Suite IP cores do not continuously process data (internal calculations)
- Stalls propagate up and down the pipeline except where they can be absorbed through buffering within the cores themselves.
- With the exceptions of the Deinterlacer and Frame Buffer in double or triple-buffering mode, none of the IP cores overlap the processing of consecutive frames.
- The first sample of frame F + 1 is not input until after the IP cores produce the last sample of frame F.



A core is stalled when either:

- it is ready but there is no valid input data from previous
- has valid output data but the receiving core is not ready to receive it.

Bridging all together

Converts clocked video input to AV-ST Video

Measures Resolution of Input to set Control Packets

FIFO crosses data from video clock to av-st domain

If downstream pipe cannot process incoming video, FIFO overflows

Flow Control

Converts AV-ST Video to clocked video

Registers define timing of sync signals

FIFO crosses data from av-st to video clock domain

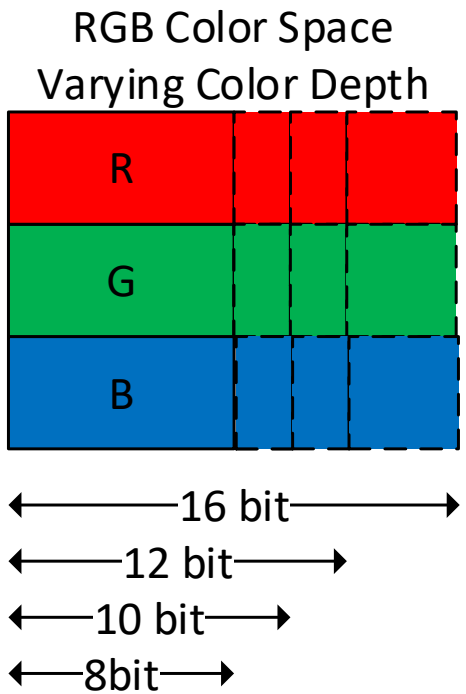
If input data rate is insufficient, FIFO will underflow.

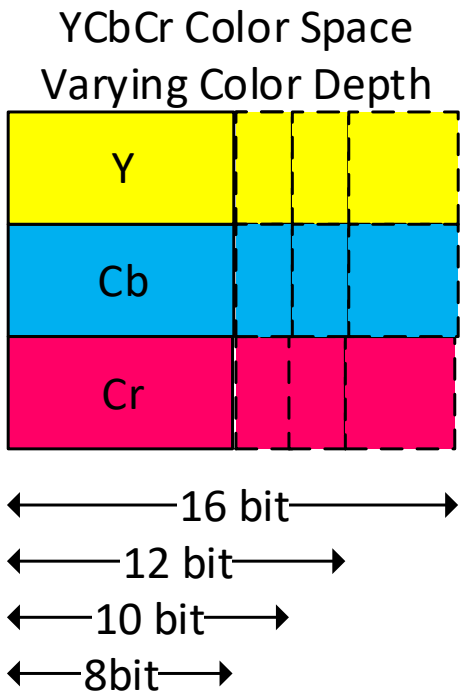
Session 2 – Debug VIP pipelines

Key Video Concepts - Format Conversion

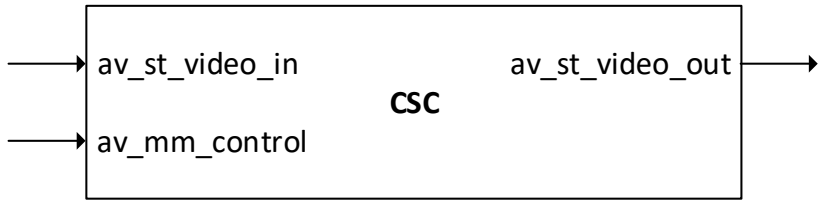
Key Concept – Color Space

- Video frames are made up of pixels.
- Pixels comprise a number of elements
  - Color Space – the digital components that captured light is sampled into
  - Color Depth – the number of bits used to represent each color space element
- There are other color spaces but these are the most common and directly supported by VIP cores.





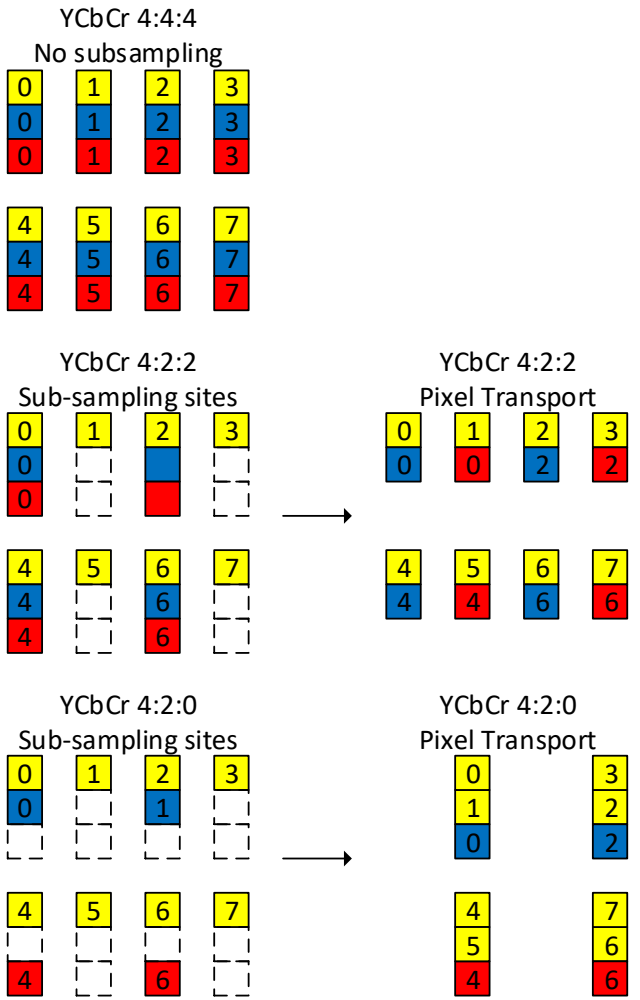
Color Space Converter (CSC) Description



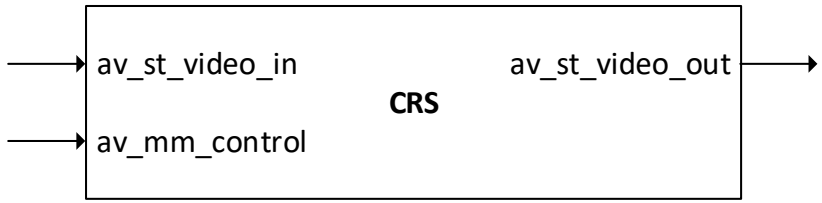
- Uses a set of coefficients to convert from one color space to another.
- There are predefined conversion between standard color spaces
- Allows the entry of custom coefficients
- Typical use case is moving between RGB <> YCbCr.

Key Concept – Sub-sampling

- The YCbCr format may be subsampled to reduce bandwidth requirements.
- Standard formats are
  - 4:4:4
  - 4:2:2 (Standard format for SDI)
  - 4:2:0 (HDMI/DP like to combine this with half frame rate refresh to offer new resolutions on old standards).



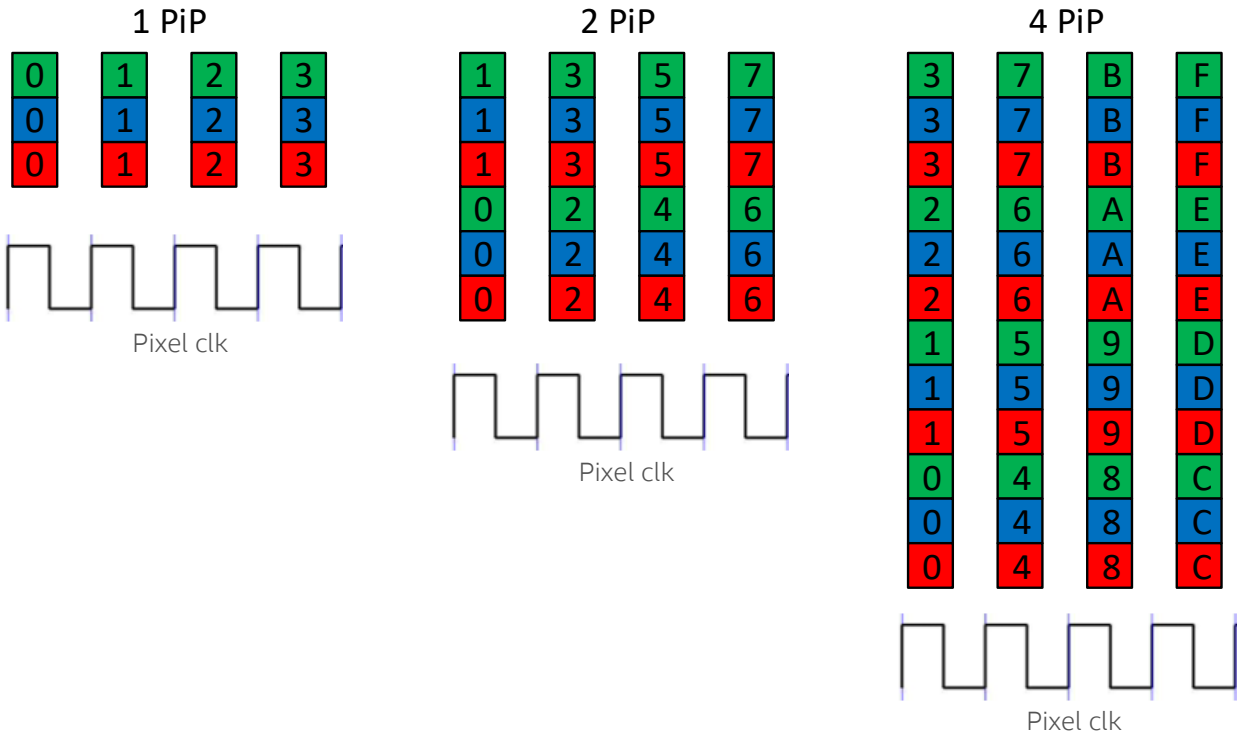
Chroma Resampler (CRS) Description



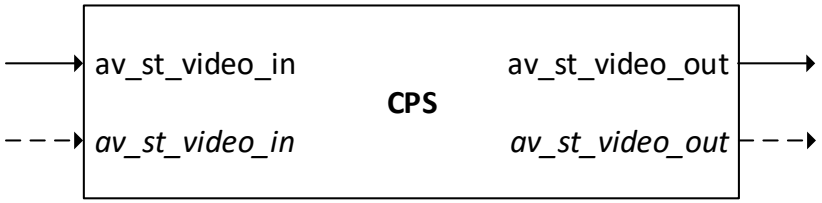
- Converts YCbCr subsampling patterns. 2 operation modes configurable
  - Fixed : Both the input and output interfaces are fixed to a set subsampling format
  - Configurable : Configure either the input or output interface as a variable subsampling
- Supports 3 chroma resampling algorithms
  - Nearest Neighbor, Bilinear or Filtered
  - Tradeoff visual quality <> resources

Key Concept – Pixels In Parallel (PiP)

- Increasing spatial resolutions demand ever greater bandwidth for processing.
- Increasing core frequencies is not always a viable solution to handle this.
- Carriage of multiple pixels each clock cycle is another approach to managing the bandwidth.



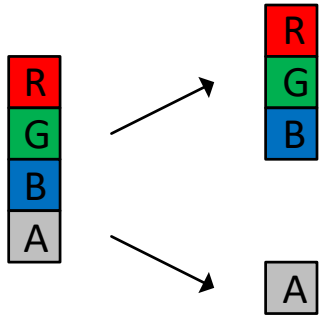
Color Plane Sequencer (CPS) Description



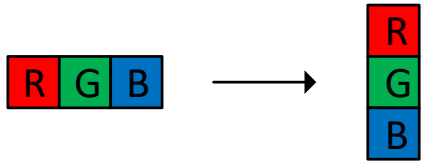
- The color plane sequencer allows the user to define a mapping that can
  - Re-arrange symbol order of input stream(s)
  - Combine input streams to a single output stream
  - Split an input stream to multiple output streams
  - Duplicate an input stream to 2 output streams.

CPS Example Use Cases (not exhaustive)

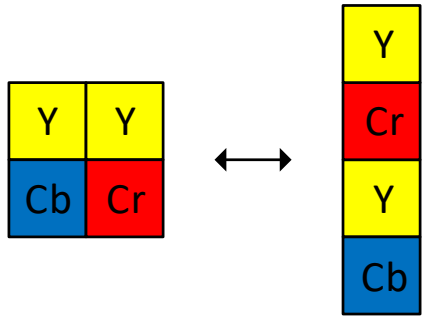
- Split/Merge alpha channel and color data :



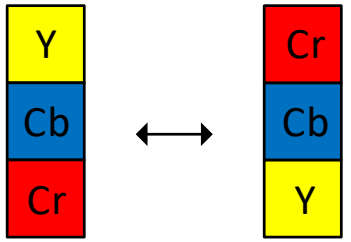
- Serialise/De-serialise pixel data :



- Pip conversion



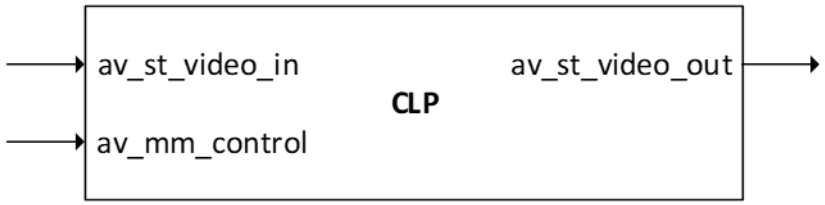
- Color swap

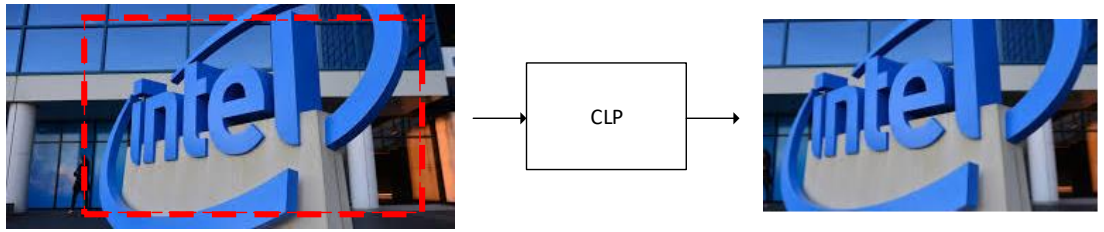


Session 2 – Debug VIP pipelines

Key Video Concepts - Resolution change

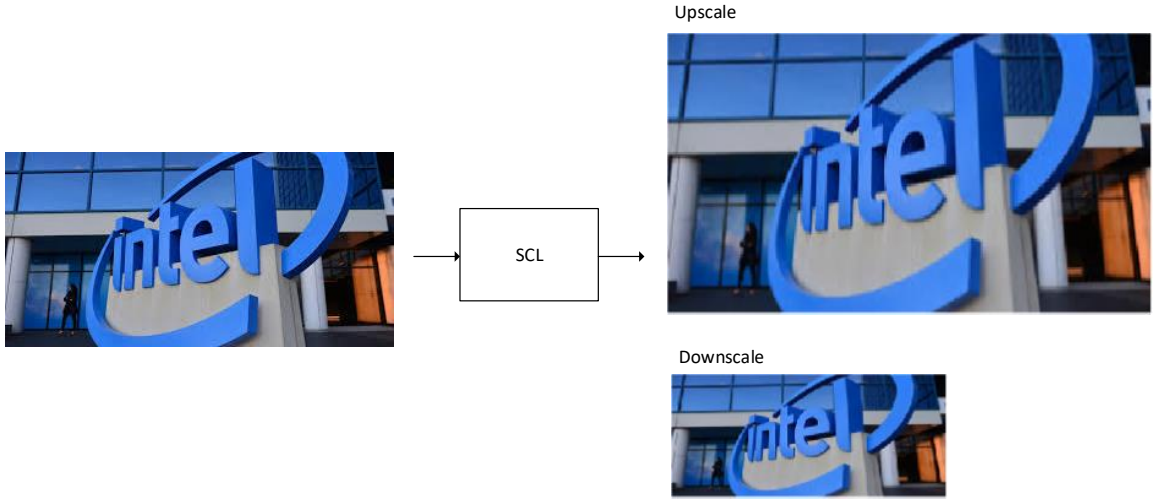
Clipper (CLP) description





- Selects an active area of video from incoming frames.
- Data outside this area is discarded.
- The active video is output as a frame with a new resolution.
- The clipper core generates control packets according to the new output resolution generated

Scaler (SCL) Description

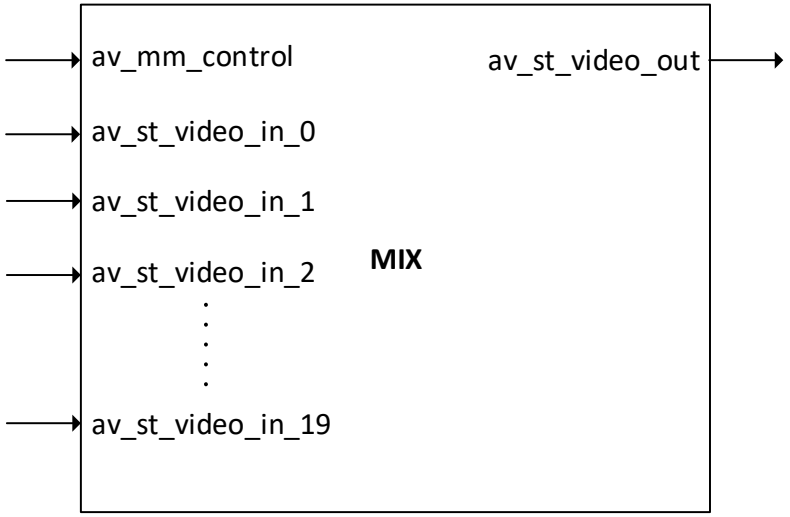


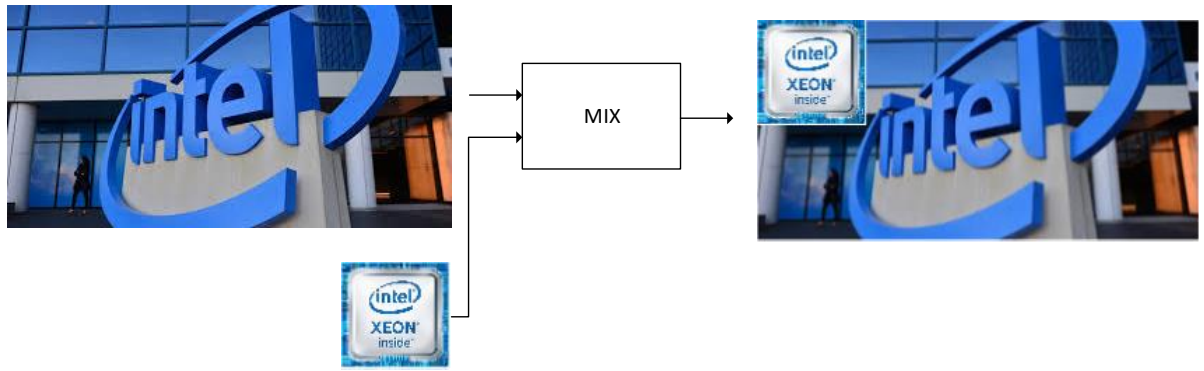
- The scaler generates an output image of specified resolution.
- Sets of software controlled coefficients define the scaling operation.
- Multiple sets of coefficients can be stored, ready to be selected based on input and output resolutions.
- The software must track the resolutions and ensure appropriate coefficients are available.

Session 2 – Debug VIP pipelines

Key Video Concepts - Image Compositing

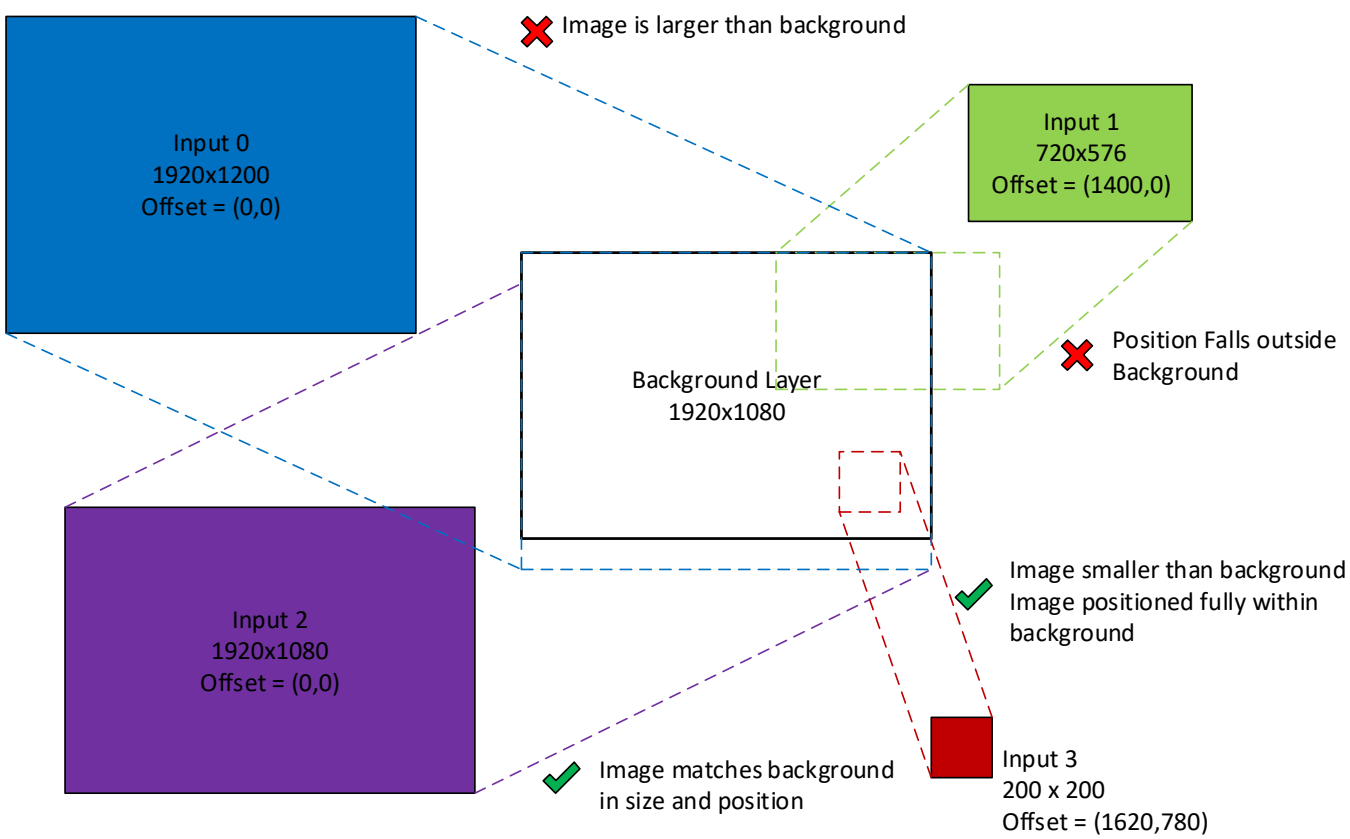
Mixer Description





- Receives between 1 and 20 input streams.
- Each input can be ignored, blended or overlaid with the other inputs, all specified at run time.
- Position of each input within the output frame is runtime configurable.
- Interlaced input is not handled by the mixer.
- The output will stall if inputs are not valid when the mixer reaches their position. This can create large bubbles of invalid data in the pipeline

Mixer Positioning Rules



- Certain rules must be followed for the mixer to operate correctly.
- The runtime background layer cannot be larger than the compile time maximum resolution.
- No input resolution may be larger than the current background layer in height or width.
- Positioning offsets (x,y) must place input frames completely within the background layer

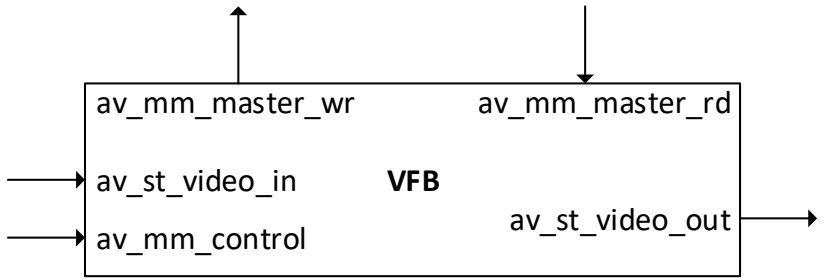
Session 2 – Debug VIP pipelines

Key Video Concepts - Rate change

Key Concept – Rate Changing

- Refresh rate is the number of video frames delivered each second.
- There are a number of common refresh rates rooted in different technologies e.g 24Hz, 30Hz, 50Hz, 60Hz/59.94, 120Hz
- Altering refresh rates demand discarding or adding entire frames, implemented using a frame buffer.
- Maintaining refresh rate but altering resolution is still a rate change, generally in-system buffering adding to handle the change. This is true, even when only changing blanking periods.
- Considering total data rate, rather than refresh rate will steer you clear of many system errors.

Frame Buffer (VFB) Description



- Writes frames to memory
- Reads Frames Back from memory
- Double or triple buffer implementation to facilitate handling of variations in data rate and frame rate depending on configuration.

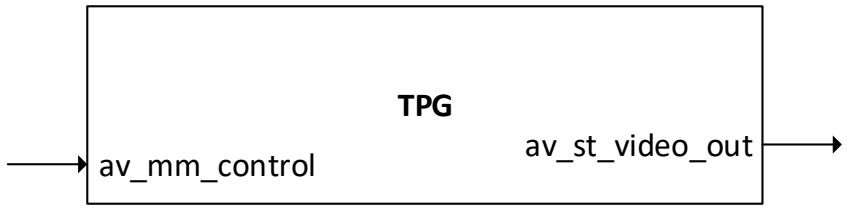
Session 2 – Debug VIP pipelines

Utility modules

Guard Bands Description

- Each color plane is assigned upper and lower limits
- The Guard Bands module compares each input color plane against it's respective limits.
- Any value exceeding the limit, is replaced with the limit.
- This core is typically used to ensure that processing pipelines haven't pushed data into the SDI reserved value ranges.

Test Pattern Generator (TPG) Description



- Generates AV-ST Video frames
- Test Pattern may be
  - Constant color
  - Vertical color bars
- Run time control allows configuration of
  - Color Space
  - Sub-sampling
  - Resolution

Stream Cleaner

- This module can be placed after cores which can produce broken packets (e.g CVI) to ensure pipeline is not exposed to them.
- Alternatively it can be placed immediately before modules which are sensitive to broken packets.

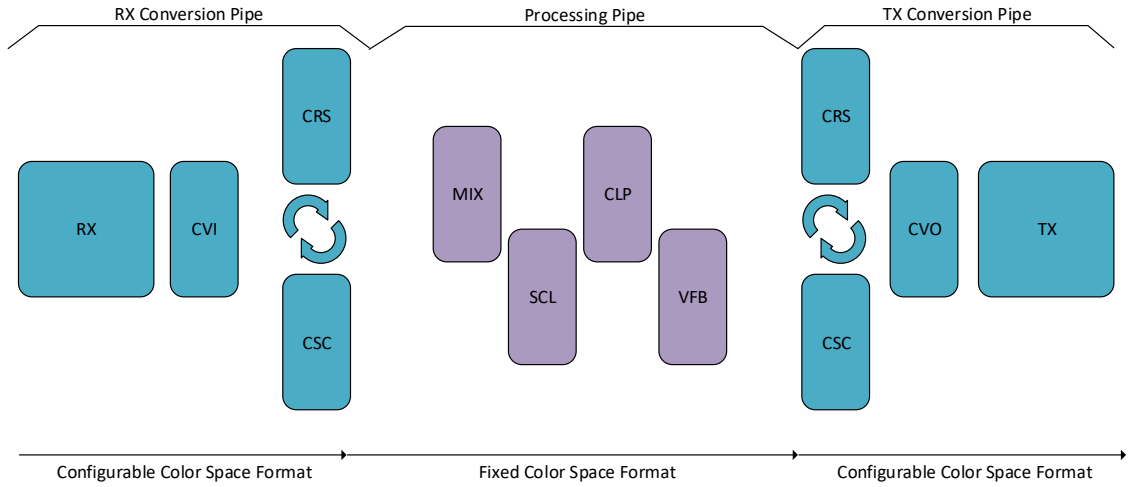
- The stream cleaner modifies the outgoing stream to match the preceding control packet.
- Ordinary Operation : output = input
- Early End Of Packet : appends pixels to end of input stream until control packet pixel count is reached.
- Late End Of Packet : Clips input pixels when they exceed the pixel count of the control packet.

Session 2 – Debug VIP pipelines

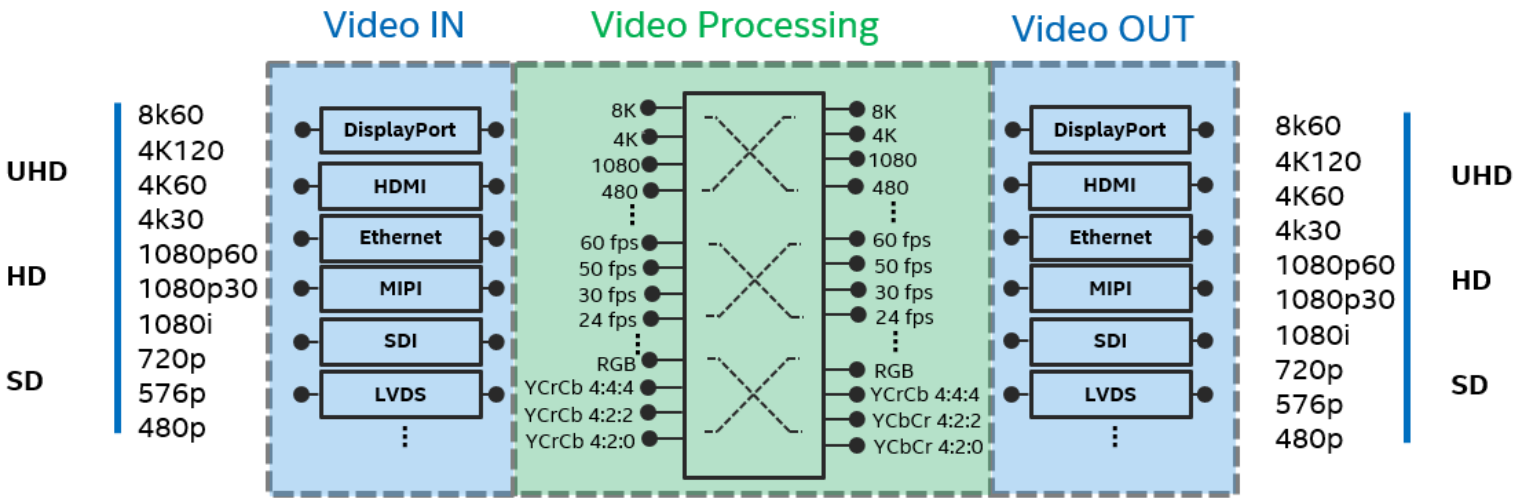
System Level Considerations

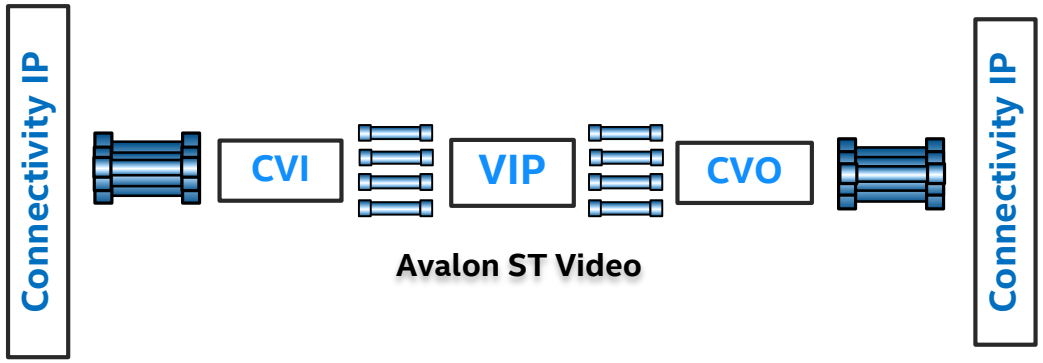
General Pipeline Structure

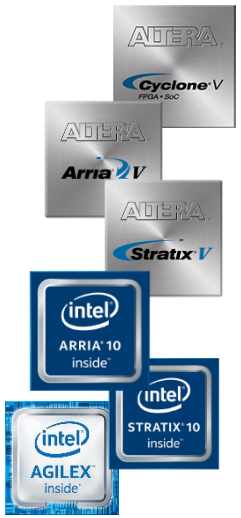
- A video pipeline generally contains 3 distinct sections
- A RX conversion pipe, taking any given input format and presenting a fixed format to the processing pipe
- A processing pipe that manipulates the video
- A TX conversion pipe that takes the processing pipes fixed format and maps it into the required output format.



How many pixels in parallel?



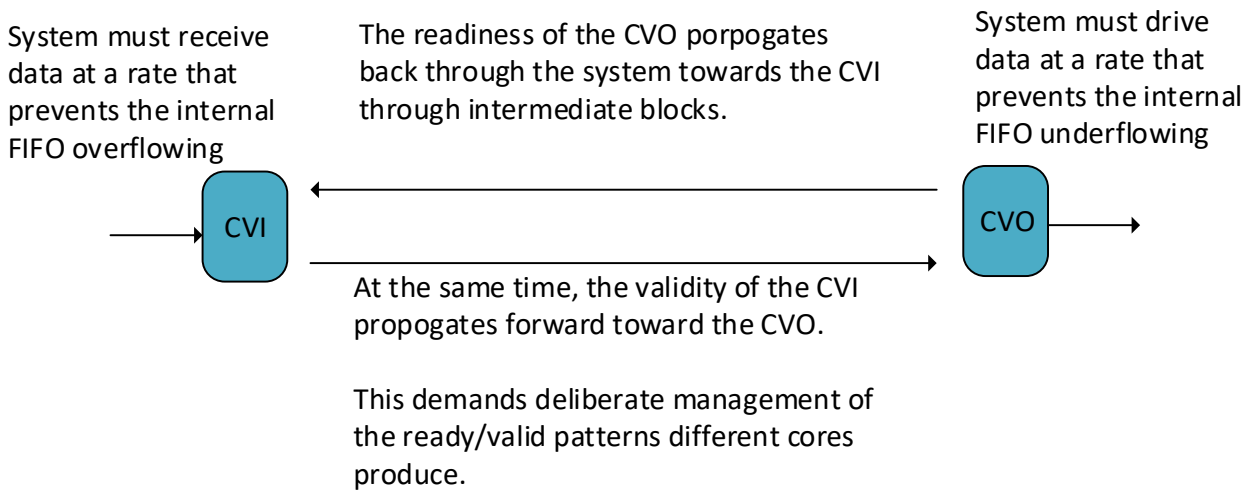




	148.5MHz	297MHz	594MHz
1 pixel / clock	1080p60	4Kp30	4Kp60
2 pixels / clock	4Kp30	4Kp60	8Kp30
4 pixels / clock	4Kp60	8Kp30	8Kp60
8 pixels / clk		8K60	8Kp120 (stretch)

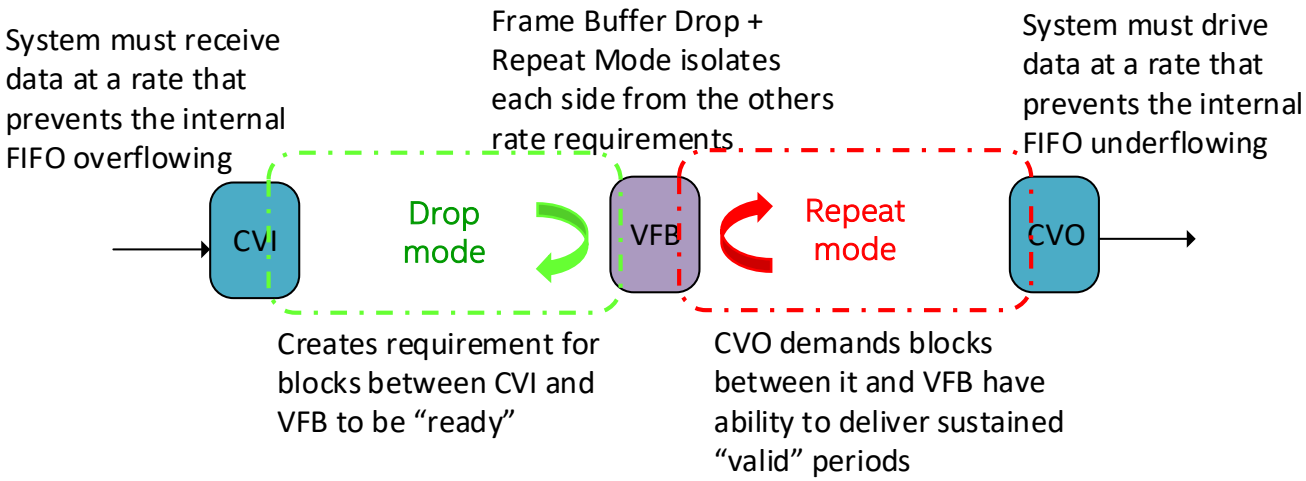
Clocked Video Interfacing

- The clocked video interfaces place certain constraints on a systems operation.
- Components downstream of a CVI, must be able to process or buffer data rapidly enough to prevent overflow
- Components upstream of a CVO must be able to generate data rapidly enough to prevent underflow



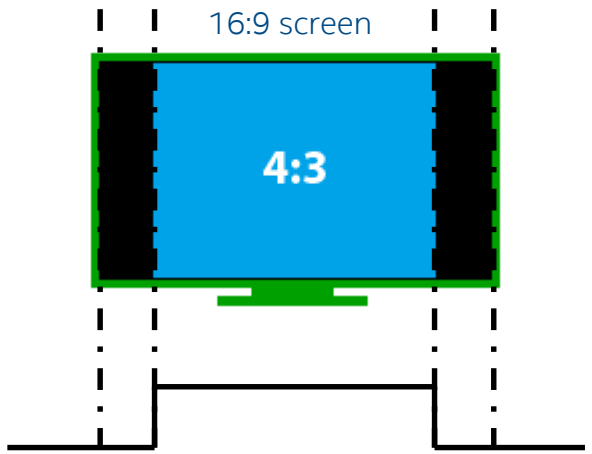
Clocked Video Interfacing

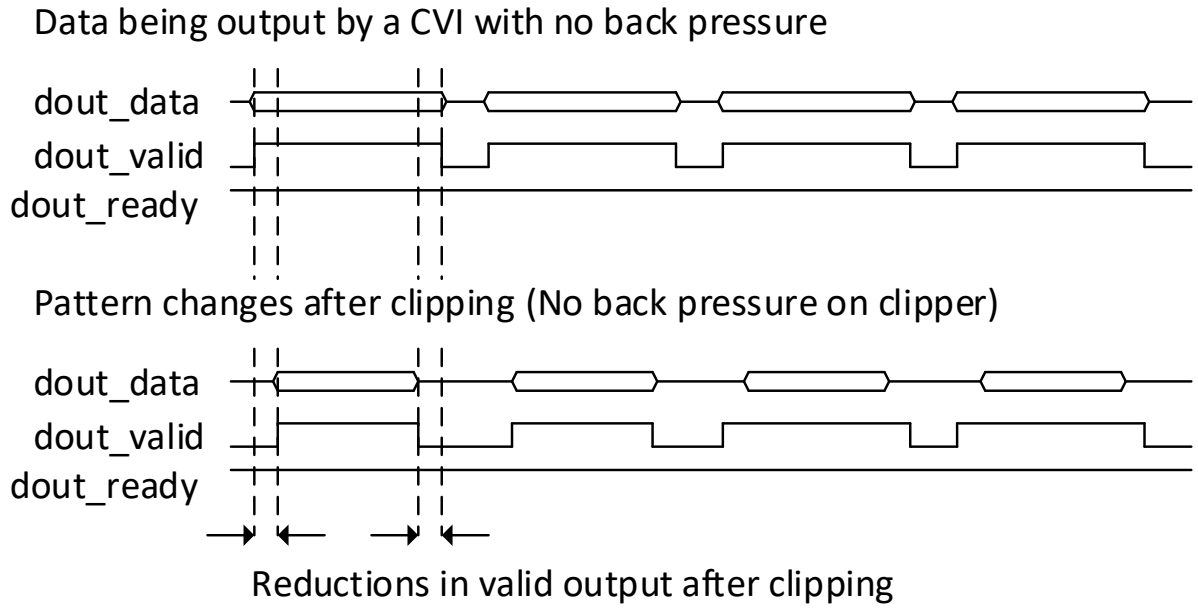
- Frame Buffers can be used to break the propagation chains between CVI and CVO, effectively creating 2 independent domains.
- There are still demands on the ready/valid management between modules.



Clipper Data Rate Changes

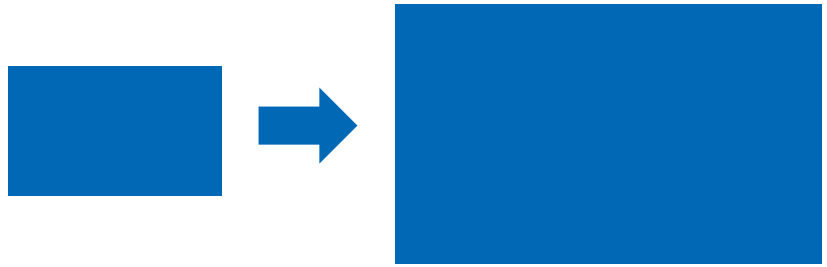
- If a clip is set to remove a vertical bar from the left and right of the input frame, each burst of data from the CVI will be narrowed.
- First the left part of each line is clipped, so the data becomes valid later.
- After a valid period, the right side is clipped and valid is pulled low earlier.
- Frame buffers (or multi-line FIFOs) can smooth the increased burstiness being generated.

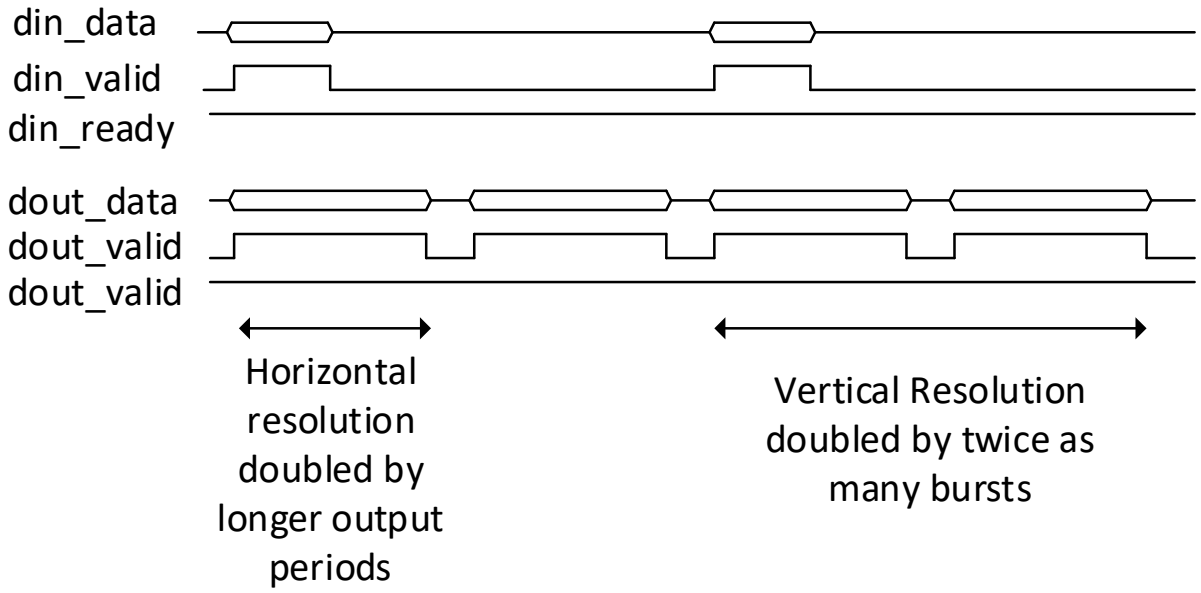




Scaler Data Rate Changes : Upscale

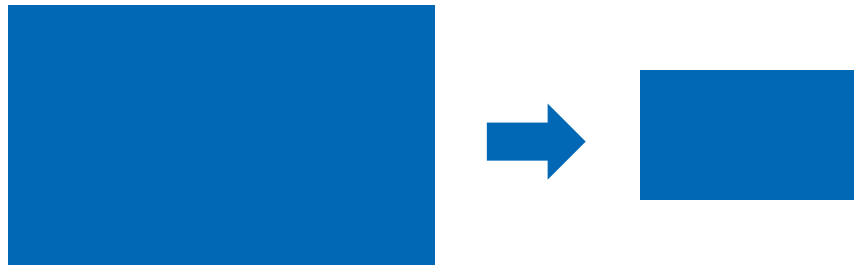
- Significant changes in resolution have a large impact on the data rate through the pipeline.
- Example : 2x upscale equates to 4x Output data rate
- The upscale creates back pressure, to hold off input, while all the additional pixels are generated.
- Use of a frame buffer before the scaler can prevent this backpressure propagating further back along the pipe.

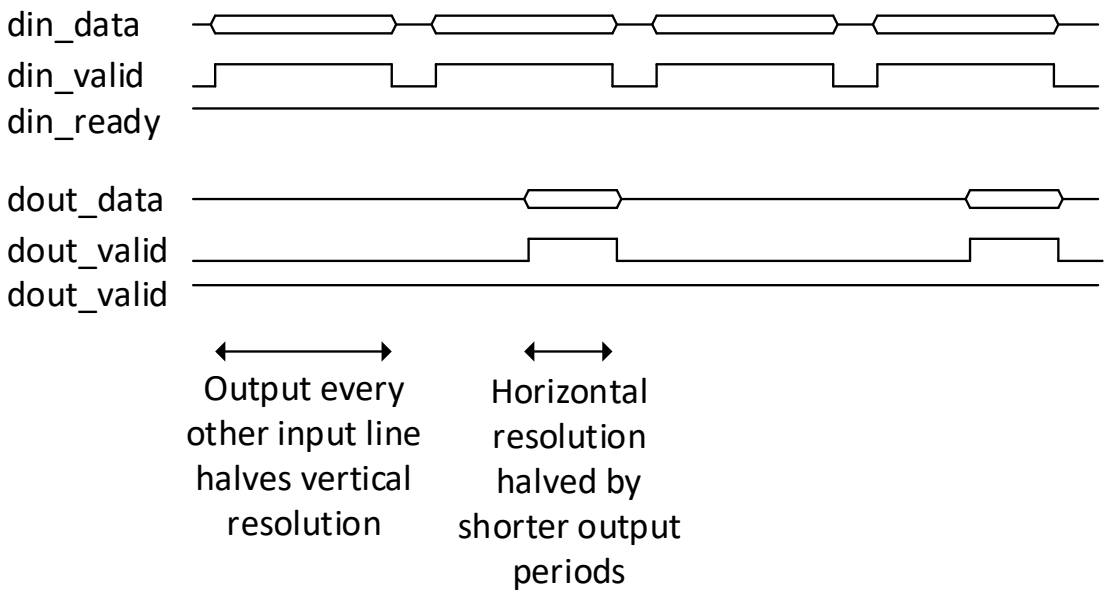




Scaler Data Rate Changes : Downscale

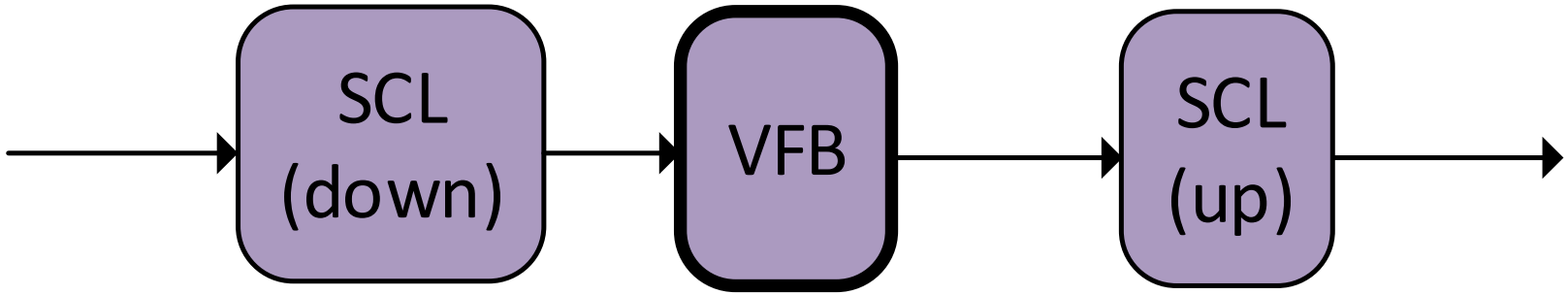
- Big downscales have the reverse effect
- Example : 2x downscale equates to ¼ output data rate
- The output becomes very bursty in nature.
- This becomes an issue when performing operations like mixing a downscaled HD image onto a 4k background.
- Placing a Frame Buffer after the scaler can smooth output bursts





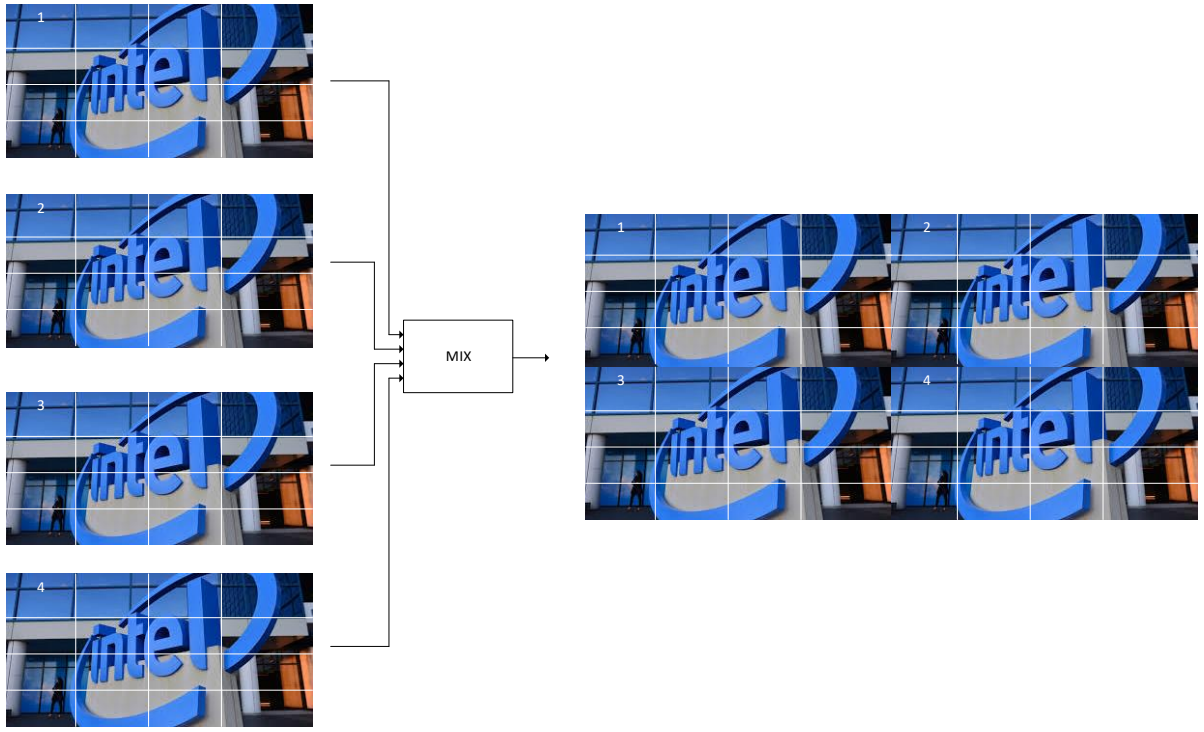
Scaling rate consequences

- The previous slides demonstrate that the buffering requirements of upscaling and downscaling are in conflict.
- The recommended approach to resolve this is use of 2 scalers, one before a frame buffer for downscaling operations, and one after a frame buffer for upscaling operations.



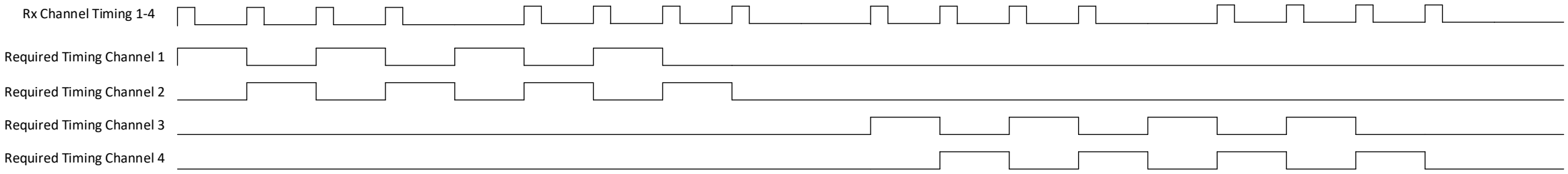
Mixing Rate Requirements

- To produce a steady output, the mixer must have all it's inputs ready at the appropriate times.
- It also need to be able to delay inputs during periods where they are not required.
- Since connectivity typically produces a constant stream of data, this typically demands each input be driven by a frame buffer.
- Consider the requirements to mix 4 independent HD inputs into a single UHD output.



Mixer Rate Requirements

- This timing diagram gives an example of how the 4 independent HD inputs would look timing wise. (There is no guarantee they would start their frames at the same time unless synchronised).
- To output each input in a different quadrant requires a minimum of half a frame of buffering for each input because of the relative position in the UHD output frame

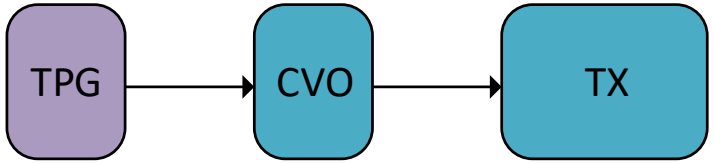


Session 2 – Debug VIP pipelines

Creating a Pipeline

System Development

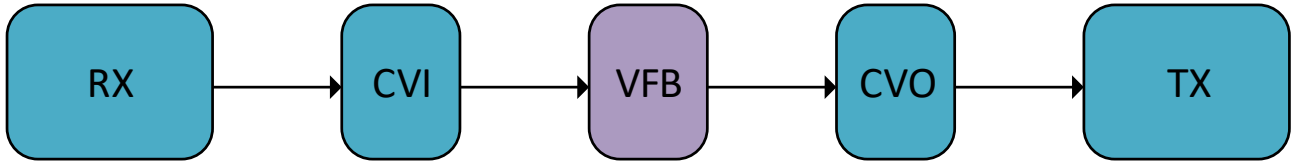
- When first developing a video pipeline it can be helpful to build up in stages.
- A very simple system that will validate the output configuration is :



- The TPG output is always valid so there is no danger of CVO underflow unless the configuration is fundamentally unable to deliver the output bandwidth (for example trying to use 1 PiP, 150MHz configuration to support 4k60)
- If the control interfaces on the TPG and CVO are enabled, it is possible to test all the CVO settings for the various output resolutions being supported.

System Development

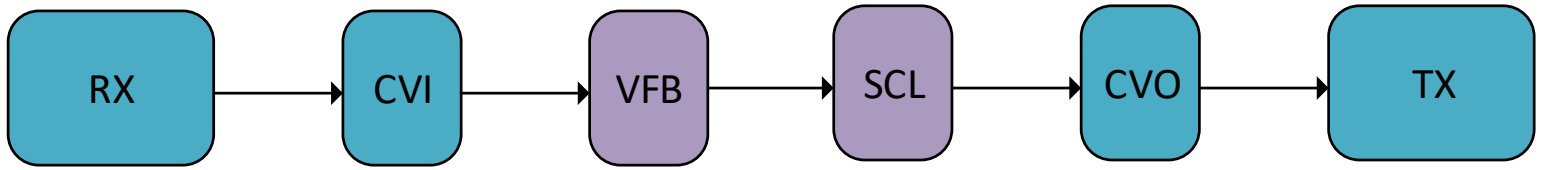
- It is a good idea to restrict how many modules are added simultaneously.
- A next step could be to check the RX to TX path :



- Frame Buffer is introduced here because :
  - It allows RX and TX to run on different clocks
  - It removes the possibility of slight rate mis-matches creating errors.
  - The system will be limited to outputting the same format as received.

System Development

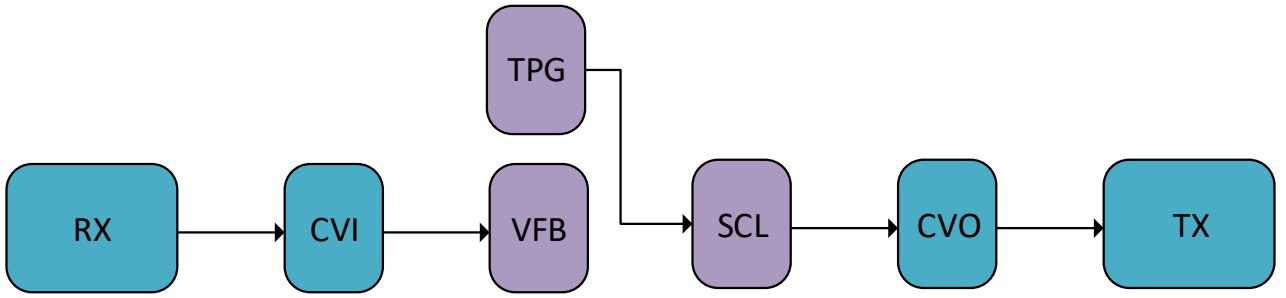
- With an RX-TX path established, it is now possible to start adding levels of functionality to the pipeline.



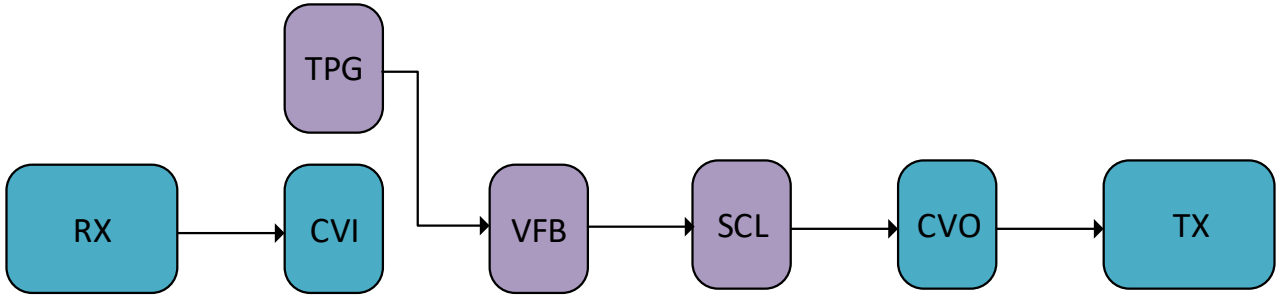
- Adding a scaler after the frame buffer would allow testing upscaling.
- Adding this feature would test
  - Upscaling function
  - Ability to control and vary the output configuration
  - Validate output operation is independent from input

System Development

- Suppose adding a scaler has resulted in intermittent output. Using a TPG to drive the scaler can establish if scaler settings are correct.



- If the output still fails, there must be an issue with the scaler (having already tested CVO).
- If the output stabilizes, the TPG can be moved further back along the pipe to check other components.



Pipeline Guidelines

- Systems with CVI and CVO modules require a frame buffer unless the output clock is derived from the input clock.
- Upscales should be performed after a frame buffer.
- Downscales should be performed before a frame buffer.
- Clips should be performed before a frame buffer.
- Systems without triple buffering enabled on a frame buffer can suffer ping-pong failures where a CVI overflows, causing a CVO to underflow, causing the CVI to overflow and so on.

Session 2 – Debug VIP pipelines

Debugging a Pipeline

System Debug

- When cores have the control interface configured, a coarse level of debug can be performed by reading the status of CVI, Frame Buffer and CVO cores.
- The user can poll :
  - CVI overflow
  - CVO underflow
  - Frame Buffer Counter
- The combined information can quickly indicate issues of data flow and whether they are situated between CVI and frame buffer or frame buffer and CVO.

System Debug – Status Register

- Each VIP core with control enabled implements a status register.
- Bit 0 of this register indicates
  - ‘0’ – no frame being processed
  - ‘1’ – frame being processed
- It is cleared after processing of each frame completes.
- Under normal operation it will be 0 for a short period of time and may appear to be “continuously” 1.
- When a module is indefinitely stalled after starting a frame, status will indicate processing although no progress is being made through the frame.

Pipeline Considerations (Mixer)

- Enabling a mixer layer that is undriven will result in stalling the output.
- Putting large downscales or clips immediately before a mixer input can cause stalled output. The output has to wait for the clip/scale to complete processing new output pixels before its own output can resume.
- It is recommended to drive each mixer input from a frame buffer to avoid such issues.

Pipeline Considerations (Frame Buffer)

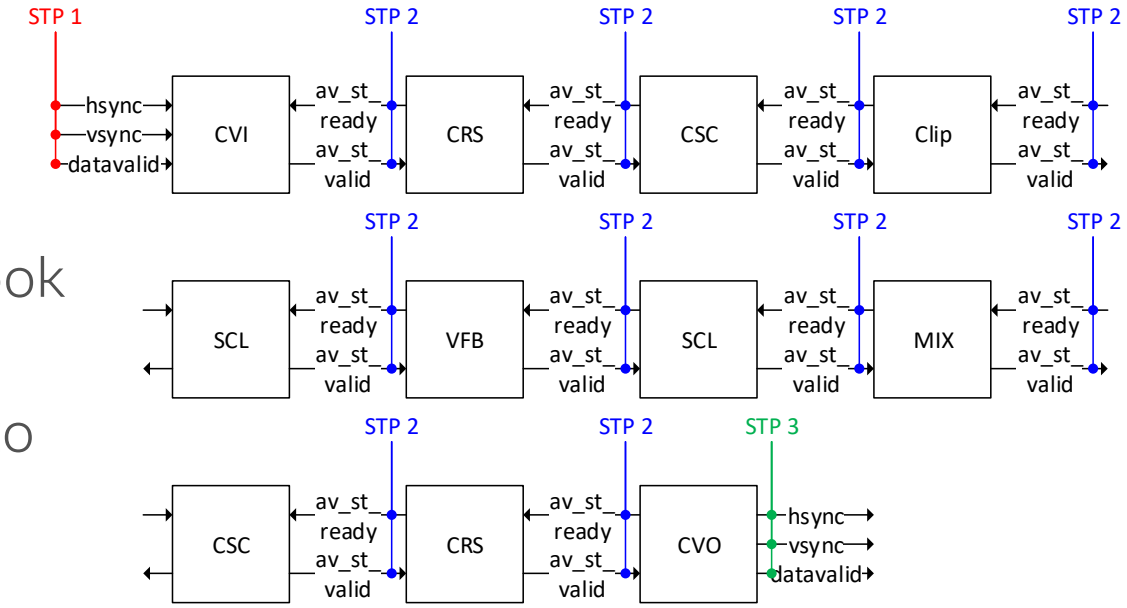
- Frame Buffer issues tend to relate to the DDR interface :
- Not reaching required transfer efficiency can result in inadequate bandwidth
  - This may backpressure the input
  - It may starve the output
- Sharing the DDR with other modules may hit efficiency, reducing an adequate theoretical bandwidth to an inadequate actual bandwidth. Consider varying the arbitration share in Platform Designer on the frame buffer interfaces to remedy sharing issues.
- Incorrect settings in the DDR interface module can significantly impact frames being correctly written and read.

PIPELINE Considerations (Scaler)

- Performing a large downscale after/without frame buffer causes bursty data that can be a problem for mixers and CVOs.
- Performing a large upscale before/without frame buffer creates back pressure that can be an issue for mixers and CVIs
- If the input or output resolution of a scaler changes, the scaling coefficients must be updated or the scaler must have multiple coefficient sets programmed.

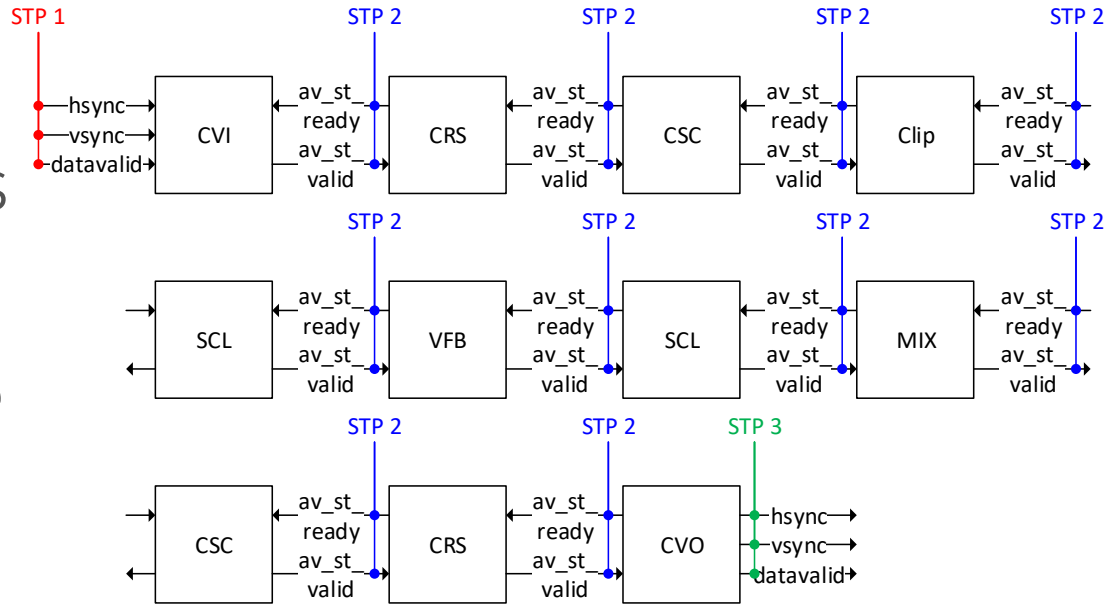
Debugging with Signaltap

- When using a TPG is insufficient, Signaltap allows waveform capture of internal signals in realtime.
- Helps to identify choke points in data flow
- Just checking flow control signals (ready and valid) will give good indications of where to look if output doesn't match expectations.
- Using the correct sampling clock is essential to collect useful debug data.
  - STP 1 Must use the Input Video clock
  - STP2 Must use the AV-ST system clock
  - STP3 must use the Output Video clock



Tapping Control Signals for dataflow debug

- The signals shown are all that is needed to see how data is proceeding through the system.
- If valid is low, the sending module is not producing data.
- If ready is low, the receiving module is not accepting data.
- Simply observing data progression is often enough to diagnose errors.



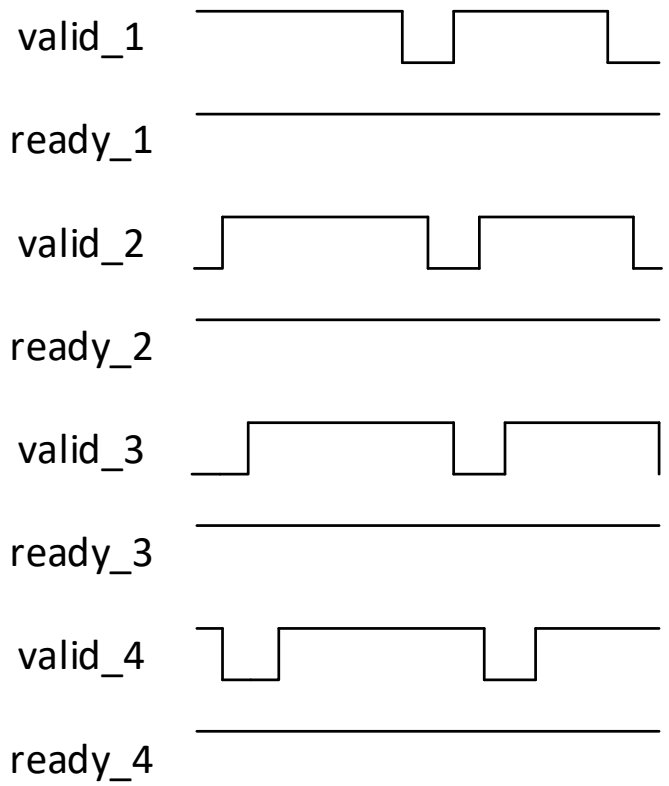
Signaltap Sample Size

- Where resource allows, try to set a sampling depth that will capture multiple lines of data simultaneously.
- Viewing multiple lines at once will highlight when there are rate issues.
- Example
- Input width = 3840 pixels
- Total Width (+ blanking) = 4400 pixels
- Data width = 2 pixels in parallel

- 2200 samples per line
- Sample Depth = 4096
- Captures 1 full line plus some data either side (Capture trigger determines where extra data is sampled)
- Sample Depth = 8192
- Completely captures 3 lines plus some data either side.

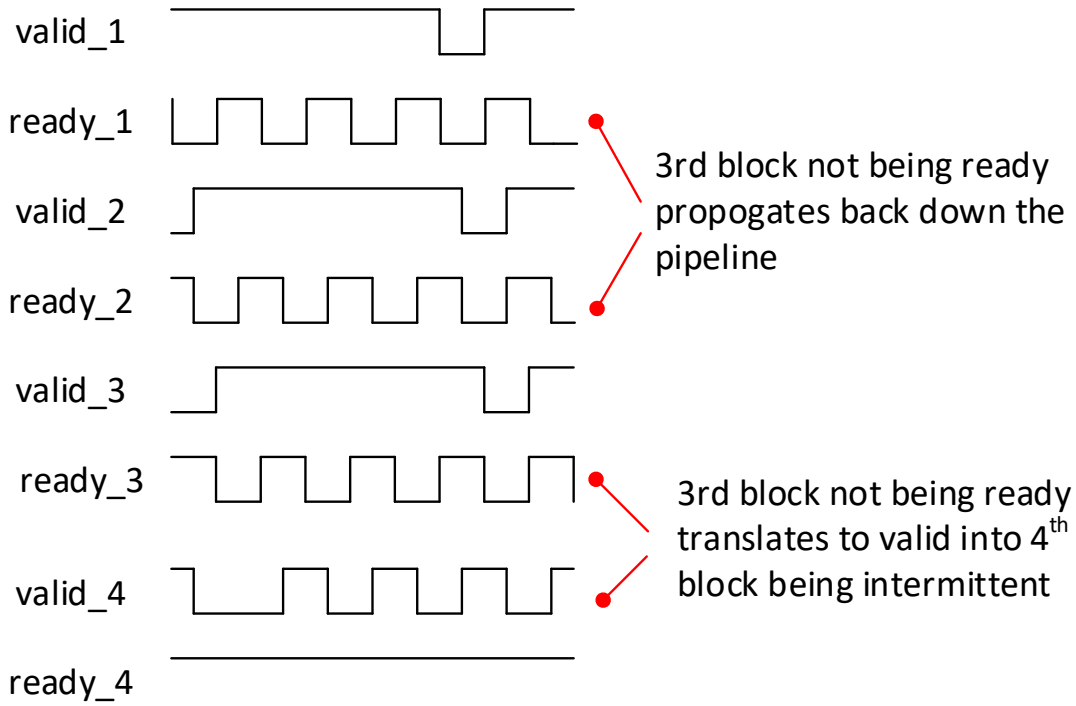
Reading Signaltap – an example Ready/Valid Operation with no backpressure

- Debugging thought process:
  - In this example all modules are continuously ready
  - Propagation of valid along the pipe limited by each module latency.
- Conclusion:
  - Input data rate is below the maximum rate the pipeline can handle



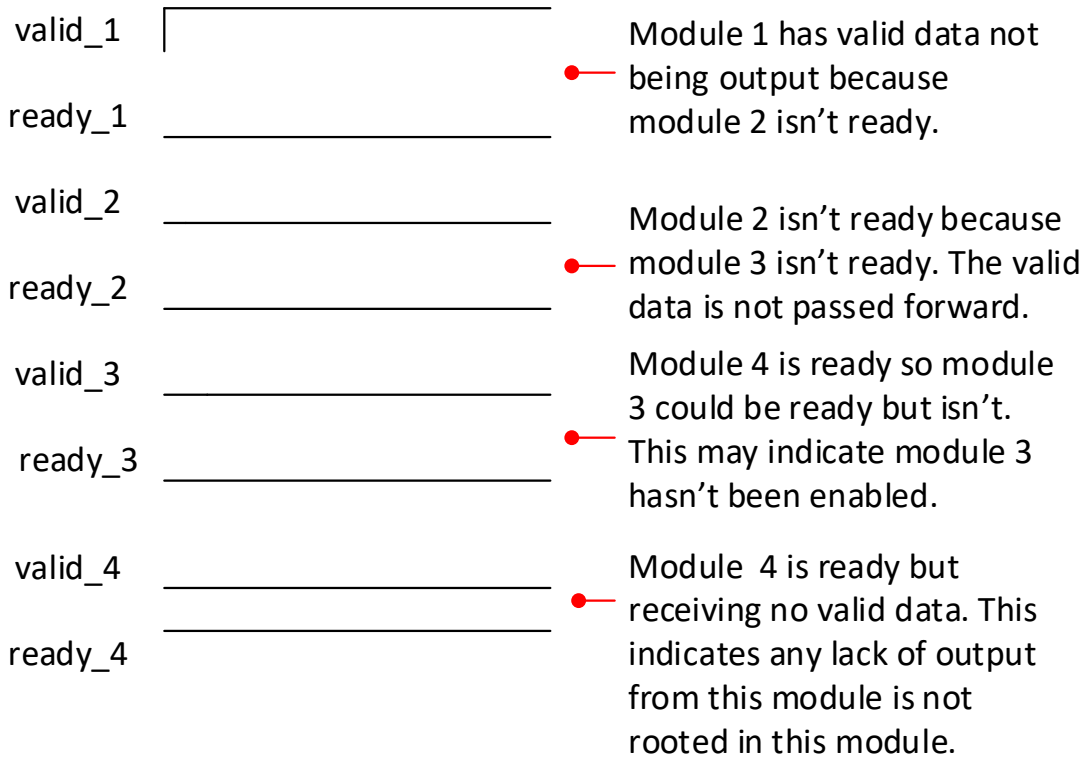
Reading Signaltap – an example Ready/Valid Operation with some backpressure

- Debugging thought process:
  - In this example the 4th module is always ready.
  - Waveforms show module 3 is origin of back pressure due to its intermittent ready.
- Conclusion : Processing bandwidth of module 3 could be insufficient. Consider
  - Increase pixels in parallel or clock frequency
  - Ensure that ready pattern isn't a feature of the configuration..



Reading Signaltap – an example Ready/Valid Operation with total back pressure

- Debugging thought process :
  - Module 4 is ready with no valid.
  - Module 3 does not assert its ready signal.
- Conclusion:
  - Module 3 is not accepting data which propagates back to module 1 and stops all data flow.
- Check :
  - Module 3 has been enabled
  - All required setup has been configured



Session 2 – Debug VIP pipelines

Controlling a Pipeline

Configuring Pipelines : Runtime Control

Discussed in previous session

- VIP cores offer runtime control.
- Each module with runtime control will have a memory mapped slave interface adhering to the AV-MM standard. Some modules, such as the CVI or CVO will also have an interrupt line.
- If configured, these interfaces must be connected to a controlling unit (typically a small micro-processor e.g NIOS-2). All cores with control enabled, start in the STOP state.
- Software can then poll and set registers to adapt to changing input and output requirements.

Runtime Control – BSP Support

Discussed in previous session

- When Platform Designer generates a system, it creates a .sopcinfo file.
- This file is used in the Eclipse environment to identify and generate supporting software files in a Board Support Package(BSP).
- The BSP includes :
  - System.h – file defining base address and IRQ settings for each module
  - Drivers – Many modules direct Platform Designer to add supporting drivers to the BSP including VIP modules

Runtime Control – System Console

What if you don't have CPU or software ready yet?

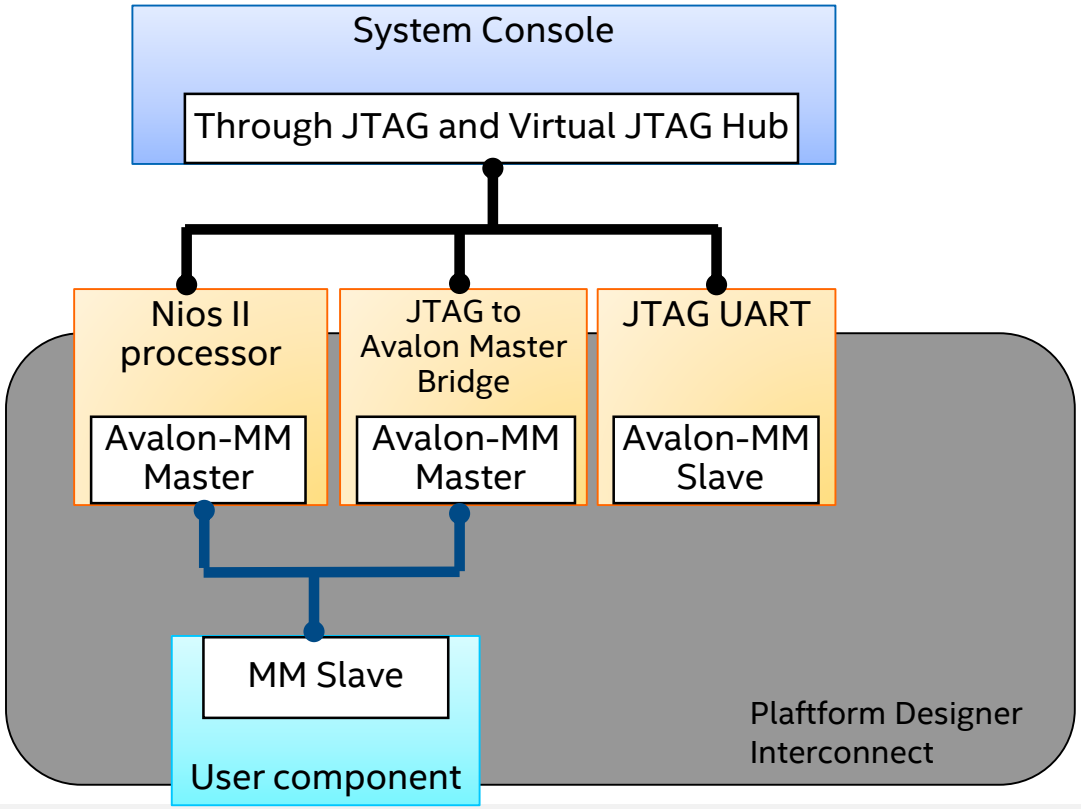
- Quick debug of Platform Designer systems through an interactive Tcl console

  Provides visibility into your design and allows you to perform system level debug on an FPGA at run-time

  Read from or write to memory-mapped components

  Debug over JTAG

  No processor required



Runtime Control – System Console

How to add to the system

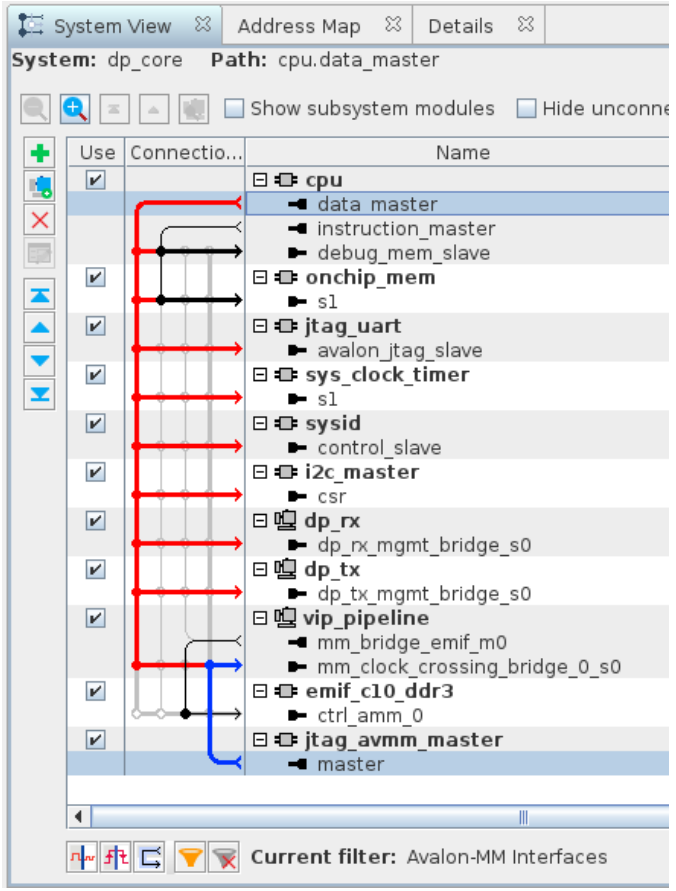
- IP Catalog -> Basic Functions -> Bridges and Adaptors -> Memory Mapped -> JTAG to Avalon Master Bridge

  Connect the *master* interface to the slave port of the **vip_pipeline**

  We can keep or eliminate the connection with the **cpu**

  Arbitration logic will be generated

  The system generates the Address Space for the **jtag_avmm_master**

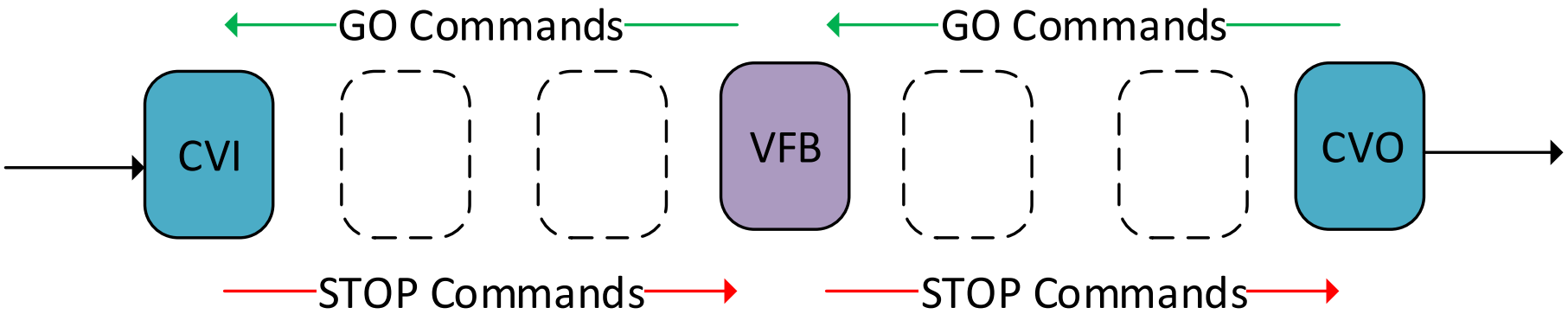


More in the lab exercise

General Pipeline Control Guidelines

- STARTING A PIPELINE : Issue GO commands starting at the module nearest the sink working back toward the source.
- STOPPING A PIPELINE : Issue STOP commands starting at the source and working towards the sink.

Each side of the Frame buffer can be treated as an independent pipeline for STOP/GO considerations



General Pipeline Control Guidelines

- These guidelines also apply when stopping/starting part of a pipeline E.G
  - Stopping the pipeline between input and the Frame buffer write side
  - Re-Starting the pipeline between Frame Buffer Read and output.
- It is not necessary to set the GO bit to issue writes to other registers. Pre-configuring modules to the required settings before issuing GO commands is recommended.
- It is recommended that cores are stopped before reconfiguring if the change is triggered by or will trigger connectivity changes e.g resolution or rate change.

Supported Live Updates

- There are reconfigurations that can be made on the fly without stopping parts of the pipeline.
- Picture in picture relationships
  - Scaling when the input and output resolutions aren't changed
  - Clipping when the input and output resolutions aren't changed
- Mixer transparency levels
- Mixer layer positioning

Session 2 – Debug VIP pipelines

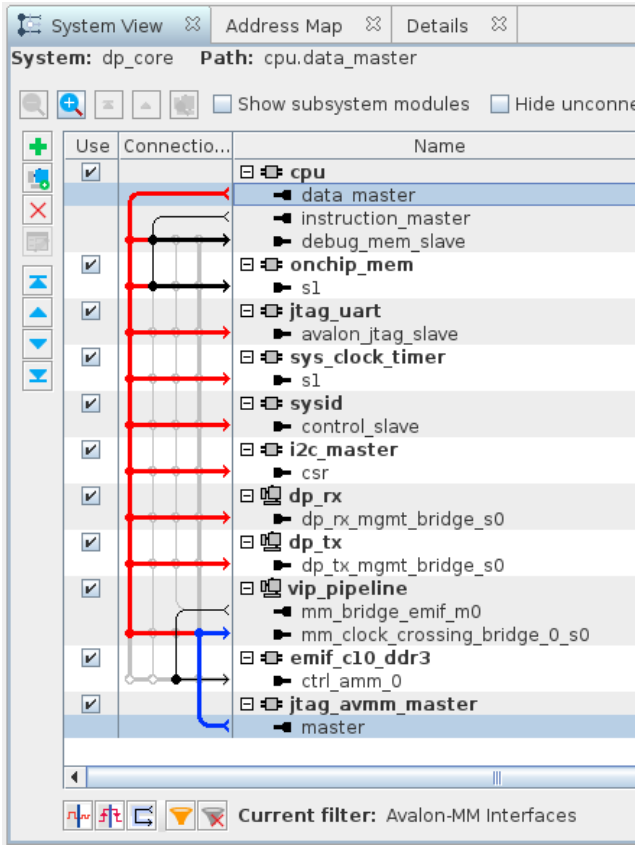
Lab – Using System Console

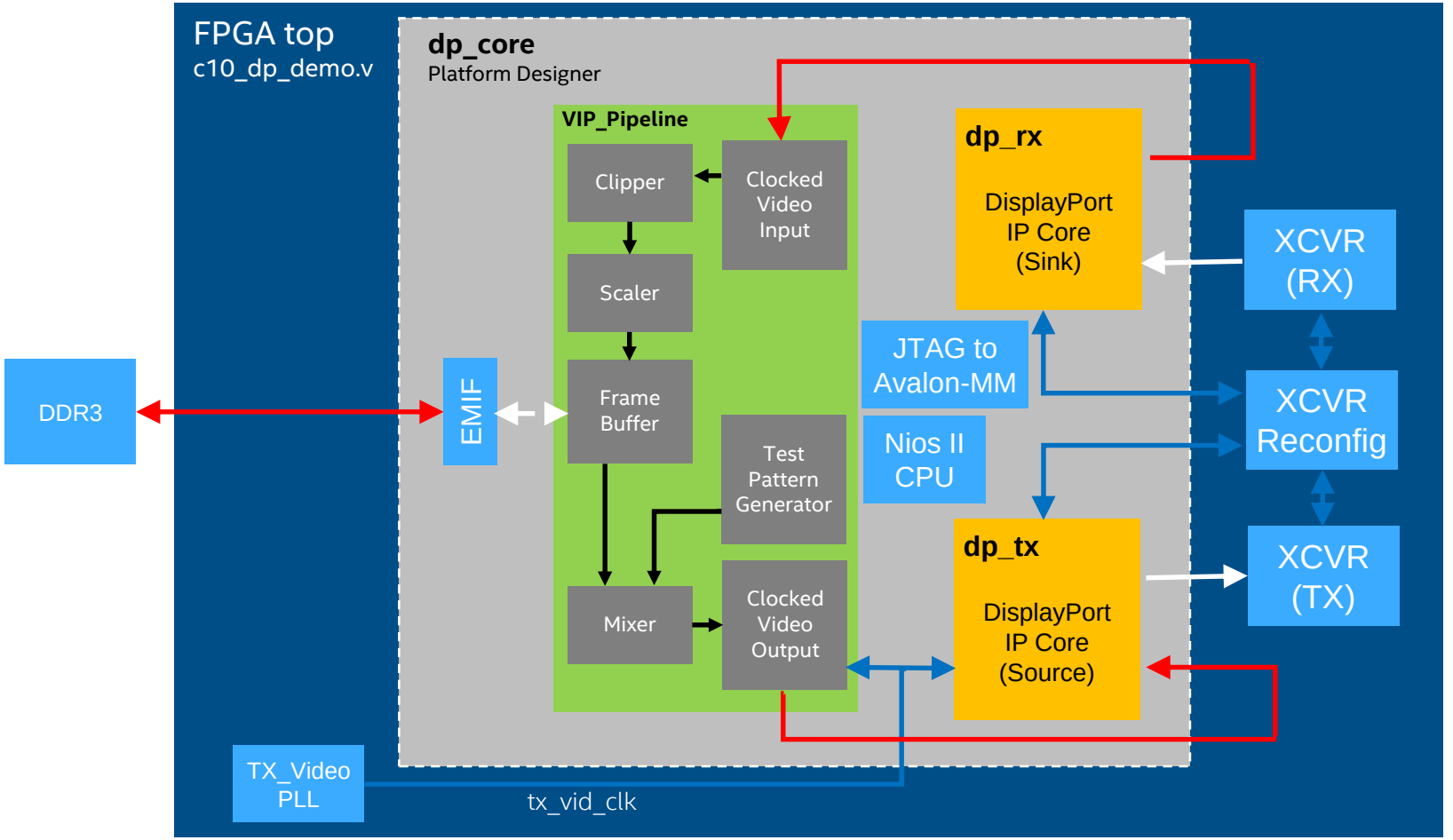
Lab Intro

- In this lab we are exercising a *JTAG to Avalon-MM master* module to use **System Console**.
- This will enable us to perform read/write transactions over the Avalon-MM interfaces of our VIP cores when:
  - CPU is not available in the system or
  - the software hasn't been developed yet.
- Using **System Console** we can modify, in run time, the parameters of our `vip_pipeline` to bring up the execution, apply different use case scenarios and/or debug potential problems.

System Architecture

JTAG to Avalon-MM bridge connected to VIP cores

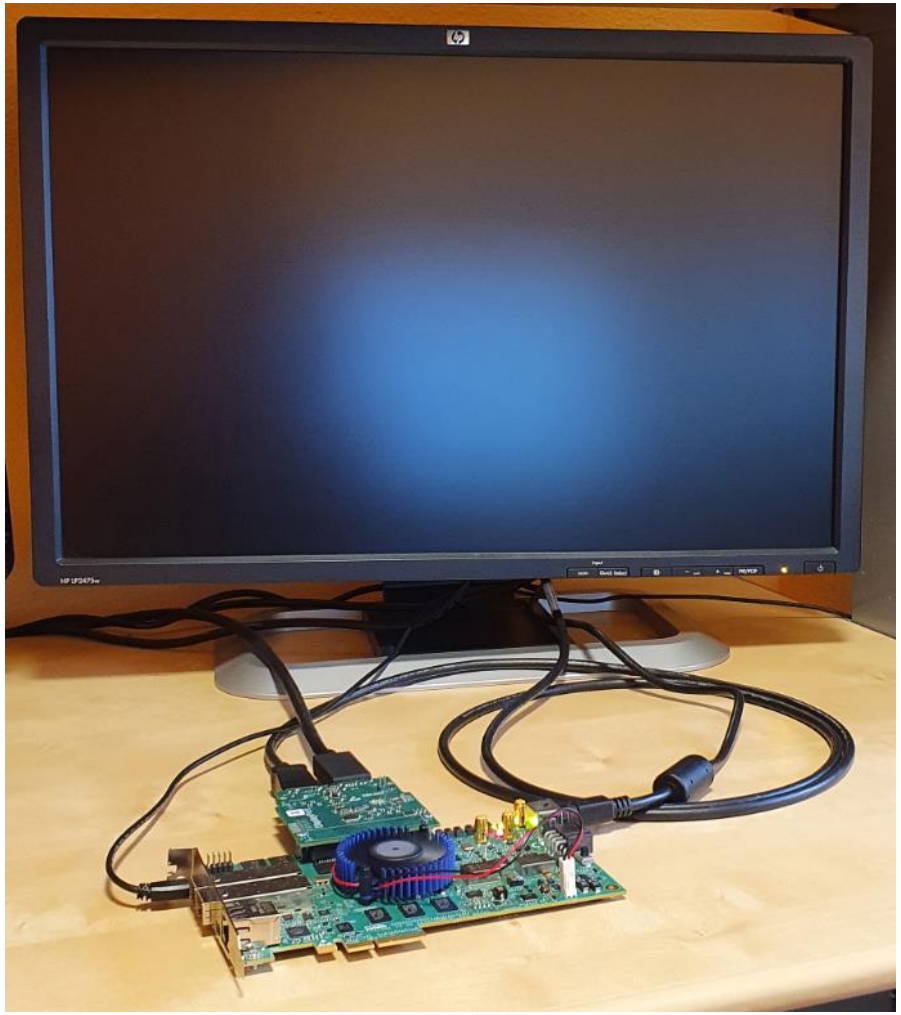




Preparing the project

Build programming files

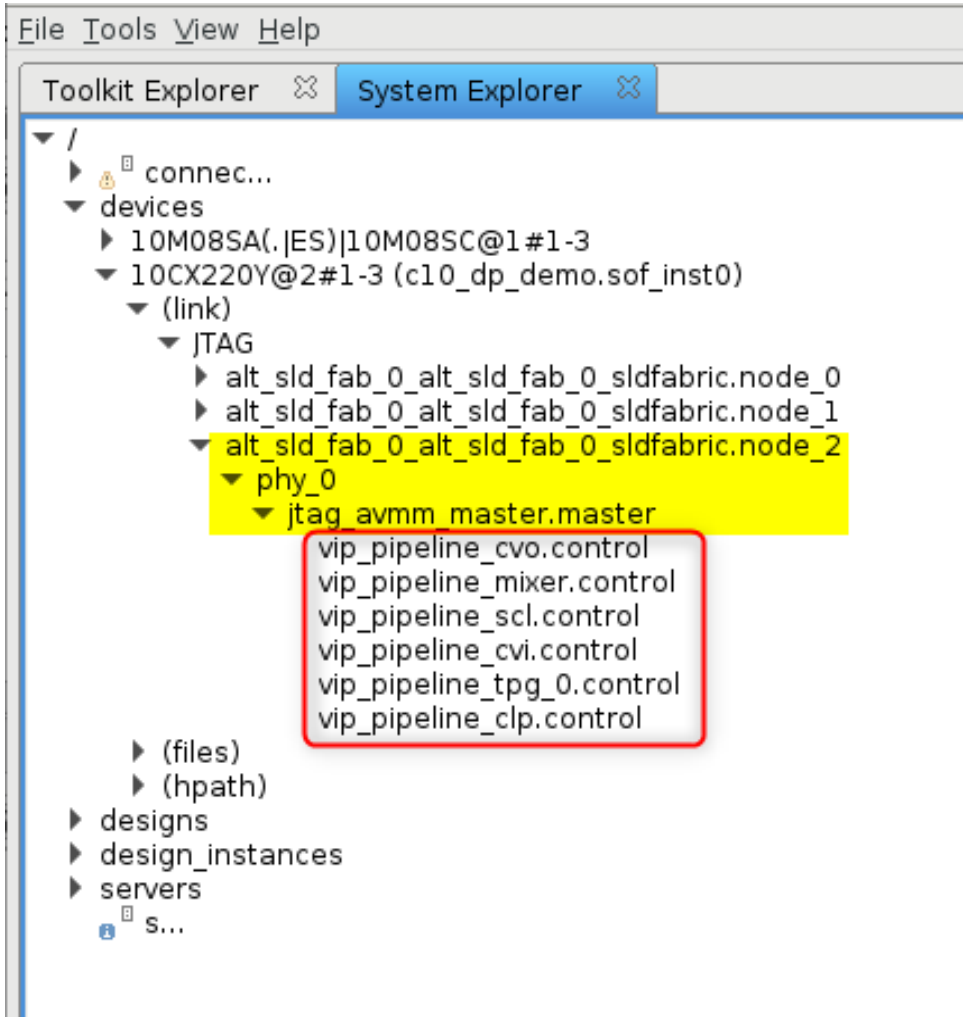
- Extract the supplied **2_System_Console_v1.tar.gz** archive file
- Generate the FPGA bitstream file
- Build the software application
  - This is needed for the DisplayPort RX and TX cores to perform link training and initialization, but not for VIP
- Download the FPGA bitstream and the software application
- You get nothing on the screen



Launch System Console

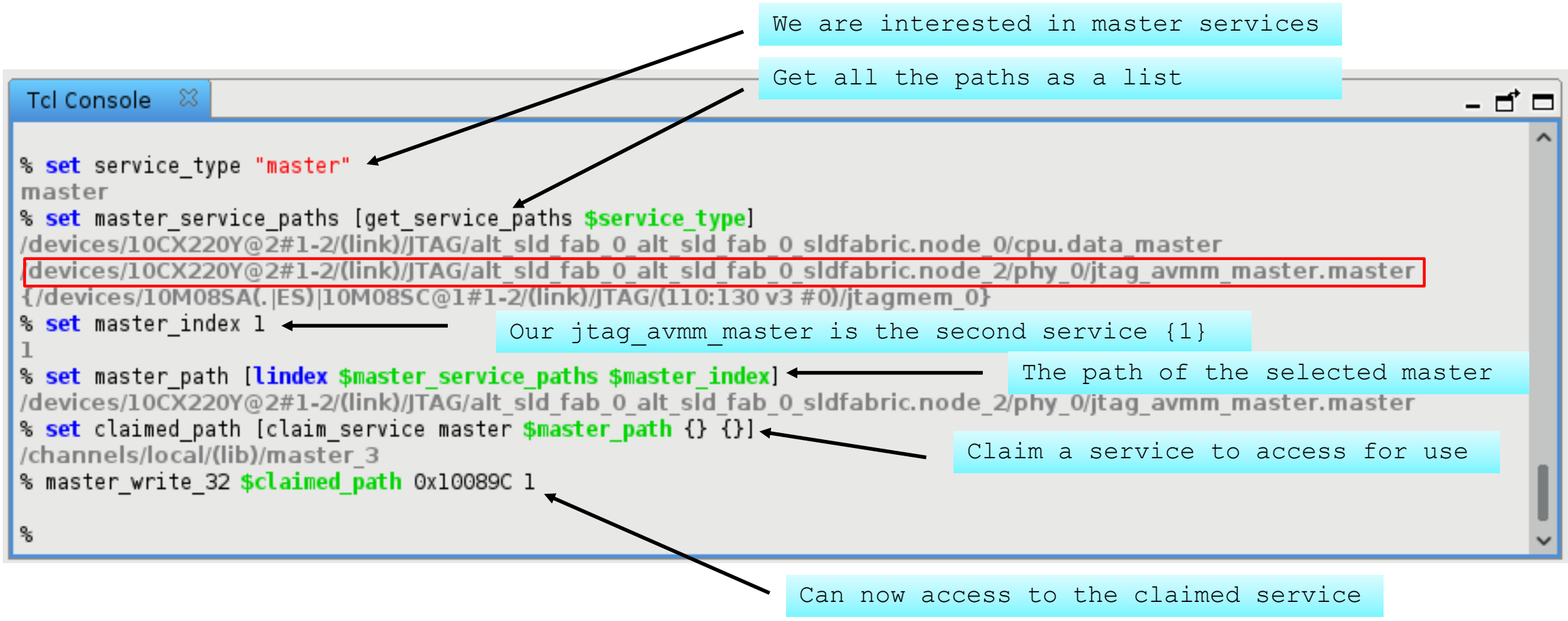
Identify our JTAG master target connection

- Go to **Tools -> System Debugging Tools -> System Console** in the Quartus main toolbar
- In System Explorer tab expand **devices -> 10CX220Y..-> JTAG** to see all accesible masters
- Look for jtag_avmm_master and verify what is connected downstream



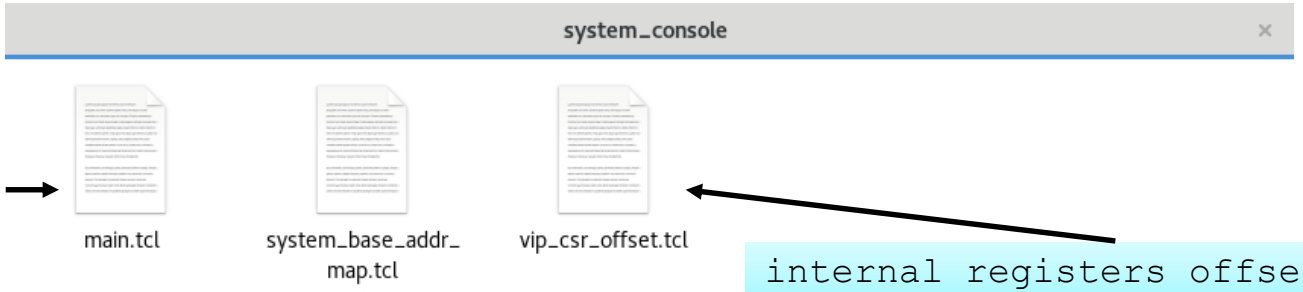
Get access to the master

Go to Tcl Console to enumerate masters and claim the service



Using scripts to ease the way

Look inside <project_dir>/system_console folder



collection of functions

```
# Enable all VIP IPs
proc go {} {
   cvo_go
   set_mixer_layer1_position 0 0
   mixer_go
   tpg_go
   set_scaler_resolution 1280 720
   scl_go
   set_clipper_area 0 0 1920 1080
   clp_go
   cvi_go
   enable_mixer_layer0 1
}

proc change_pattern {pattern} {
   global base_address_tpg
   global tpg_csr_select
   global claimed_path

   puts "Writing TPG select pattern register\n"
   master_write_32 $claimed_path [expr $base_address_tpg + $tpg_csr_select] $pattern
}

# Set Scaler output resolution
proc set_scaler_resolution {width height} {
   global base_address_scl
   global scl_csr_out_width
   global scl_csr_out_height
   global claimed_path

   puts "Writing SCL output registers\n"
   master_write_32 $claimed_path [expr $base_address_scl + $scl_csr_out_width] $width
   master_write_32 $claimed_path [expr $base_address_scl + $scl_csr_out_height] $height
}
```

base address of each VIP core from system.h

```
# set base addresses of VIP IP core
set base_address_cvi   0x100800
set base_address_cvo   0x100000
set base_address_mixer 0x100400
set base_address_tpg   0x100880
set base_address_clp   0x1008C0
set base_address_scl   0x100600
```

internal registers offset of each VIP core from VIP user guide document

```
#set SCALER II IP core CSR offset
set scl_csr_control              0x00
set scl_csr_status               0x04  ; #0x01 * 4
set scl_csr_out_width            0x0C  ; #0x03 * 4
set scl_csr_out_height           0x10  ; #0x04 * 4

#set MIXER II IP core CSR offset
set mixer_csr_control            0x00
set mixer_csr_background_width   0x0C  ; #0x03 * 4
set mixer_csr_background_height  0x10  ; #0x04 * 4
set mixer_csr_input0_x           0x20  ; #(0x08+5n) * 4 where n is the input number
set mixer_csr_input0_y           0x24  ; #(0x09+5n) * 4 where n is the input number
set mixer_csr_input0_control     0x28  ; #(0x0A+5n) * 4 where n is the input number
set mixer_csr_input1_x           0x34  ; #(0x08+5n) * 4 where n is the input number
set mixer_csr_input1_y           0x38  ; #(0x09+5n) * 4 where n is the input number
set mixer_csr_input1_control     0x3C  ; #(0x0A+5n) * 4 where n is the input number

#set CVO II IP core CSR offset
set cvo_csr_control              0x00
set cvo_csr_status               0x04  ; #0x01 * 4
set cvo_csr_mode_match           0x0C  ; #0x03 * 4
set cvo_csr_bank_select          0x10  ; #0x04 * 4
```

Perform live manipulations

System Console

File Tools View Help

Welcome | Tcl Console

```
% source main.tcl
/channels/local/(lib)/master_1
% go
Writing CVO control register

Writing MIXER layer1 position registers

Writing MIXER control register

Writing TPG control register

Writing SCL output registers

Writing SCL control register

Writing CLP area registers

Writing CLP control register

Writing CVI control register

Writing MIXER enable layer0 register


% enable_mixer_layer1 1
Writing MIXER enable layer1 register


% enable_mixer_layer1 0
Writing MIXER enable layer1 register


% enable_mixer_layer1 1
Writing MIXER enable layer1 register


% change_pattern 1
Writing TPG select pattern register


% set_mixer_layer1_position 300 300
Writing MIXER layer1 position registers


% set_scaler_resolution 720 480
Writing SCL output registers


%
```

Messages

source main.tcl script

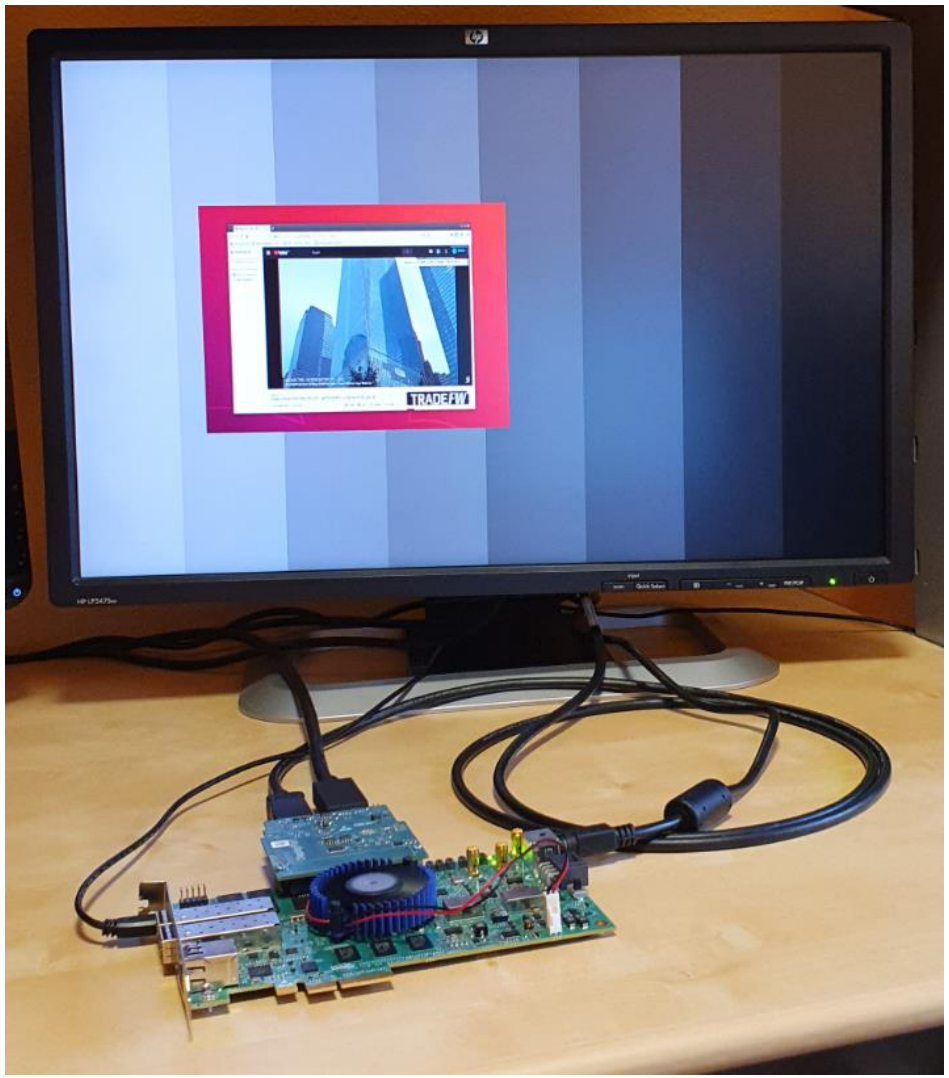
initialize the pipeline

enable/disable mixer layers

change pattern to grayscale bars

modify mixer layer position

Modify scaler resolution



LIVE DEMO

Video processing on FPGAs made easy

Content available in the GitHub repo:
https://github.com/perezfra/VIP_webinars_Intel_FPGA

Summary

- We have learnt about key video concepts and VIP cores available in the suite
- We explained some critical architectural choices: PiP, clock freq, buffering…
- We uncovered how Avalon-ST Video protocol transports video
- How to use SignalTap to detect backpressure and pipeline stalls
- Usage of System Console to bring up your pipeline without software
-