Intel® Arc™ A-Series Graphics and Intel Data Center GPU Flex Series

Open-Source Programmer's Reference Manual

For the discrete GPUs code named "Alchemist" and "Arctic Sound-M"

Volume 12: Display Engine

March 2023, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exceptions that a) you may publish an unmodified copy and b) code included in this document is licensed subject to Zero-Clause BSD open source license (0BSD). You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Table of Contents

Display

Terminology

Term	Description
DP	DisplayPort
SST, DP SST	DisplayPort Single Stream Transport
MST, DP MST	DisplayPort Multi Stream Transport

Register Access Field	Description	Implementation
R/W (Read/Write)	The value written into this register will control hardware and is the same value that will be read.	Write data is stored. Read is from the stored data. Stored value is used to control hardware.
Reserved	Unused register bit. Don't assume a value for these bits. Writes have no effect.	Write data is ignored. Read is zero.
MBZ (Must Be Zero)	Always write a zero to this register.	May be implemented as Reserved or as R/W.
PBC (Preserve Bit Contents)	Software must write the original value back to this bit. This allows new features to be added using these bits.	May be implemented as Reserved or as R/W.
Read Only	The read value is determined by hardware. Writes to this bit have no effect.	Write data is ignored. Read is from a status signal or some other internal source.
Write Only	The value written into this register will control hardware. Reads return zero.	Write data is stored. Read is zero. Stored value is used to control hardware.
R/W Clear (Read/Write Clear)	Sticky status bit. Hardware will set the bit, software can clear it with a write of 1b.	Internal hardware events set a sticky bit. Read is from the sticky bit. A write of 1b clears the sticky bit.
R/W Set (Read/Write Set)	Sticky status bit. Software can set the bit with a write of 1b. Hardware will clear the bit.	A write of 1b sets a sticky bit. Internal hardware events clear a sticky bit. Read is from the sticky bit.

Register Access Field	Description	Implementation
Double Buffered	Write when desired and the written value will take effect at the time of the update point specified in the 'Double Buffer Update Point' parameter. Reads will return the written value, which is not necessarily the value being currently used to control hardware. Some double-buffered registers have a corresponding "LIVE" read only register that provides the value being use to control hardware. Some have a specific arming sequence where a write to another register, specified in the 'Double Buffer Armed By' parameter, is required before the update can take place. Once the armed by register is written to, the written values of all registers controlled by that arming will take effect at the time of the double buffer update point. This is used to ensure atomic updates of several registers. Note: Once armed, by write to the armed by register, the registers controlled by this arming should not be changed until the double buffer update point is reached. If changed, this will disarm the sequence and will require another write to the armed by register to get it to the armed status again.	Two stages of registers used. Write data is stored into first stage. Read is from the first stage stored data. First stage stored value is transferred to second stage storage at the double buffer update point. Second stage stored value is used to control hardware. Arm/disarm logic may be used for some registers to control the double buffer update point.
Write/Read Status	The value written into this register will control hardware. The read value is determined by hardware.	Write data is stored. Stored value is used to control hardware. Read is from a status signal or some other internal source.

VGA and Extended VGA Registers

This section describes the registers and the functional operation notations for the observable registers in the VGA section. This functionality is provided as a means for support of legacy applications and operating systems.

It is important to note that these registers in general have the desired effects only when running VGA display modes. The main exceptions to this are the palette interface which allows real mode DOS applications and full screen VGA applications under an OS control running in high resolution (non-VGA) modes to access the palette through the VGA register mechanisms and the use of the ST01 status bits that determine when the VGA enters display enable and sync periods. Other exceptions include the register bits that control the memory accesses through the A000:0000 and B000:0000 memory segments which are used during operating system emulation of VGA for "DOS box" applications.

Some of the functions of the VGA are enabled or defeated through the programming of the VGA control register bits that are located in the MMIO register space.

Given the legacy nature of this function, it has been adapted to the changing environment that it must operate within. The three most notable changes are the addition of high-resolution display mode support, new operating system support, and the use of fixed resolution display devices (such as LCD panels). Additional control bits in the PCI Config space will affect the ability to access the registers and memory aperture associated with VGA.

Mode of Operation	VGA Disable	VGA Display	VGA Registers	Palette (VGA)	VGA Memory	VGA Banking
VGA DOS	No	Yes	Yes	Yes	Yes	No
HiRes DOS	Yes	No	Yes	Yes	No	Yes
Fullscreen DOS	Yes/No	No/Yes	Yes	Yes	Yes	Yes
DOS Emulation	Yes	No	Yes	Yes	Yes	Yes

VGA Display Mode	Dot Clock Select	Dot Clock Range	132 Column Text Support	9-Dot Disable Support	Main Use
Native	VGA Clock Select	25/28 MHz	No	No	Analog CRT (VGA connector)
Centered	Fixed at display Requirements	Product Specific	No	Yes	Digital Display
Upper Left Corner	Fixed at display Requirements	Product Specific	No	Yes	Internal Panel

Native, Centered, and Upper Left Corner support varies from product to product.

Even in the native VGA display operational modes, not all combinations of bit settings result in functional operating modes. VGA display modes have the restriction that they can be used only when all other display planes are disabled.

These registers are accessed via I/O space. The I/O space resides in the PCI compatibility hole and uses only the addresses that were part of the original VGA I/O space (which includes EGA and MDA emulation). Accesses to the VGA I/O addresses are steered to the proper bus and rely on proper setup of bridge registers. Extended VGA registers such as GR10 and GR11 use additional indexes for the already defined I/O addresses. VGA register accesses are allowed as 8 or 16 bit naturally aligned transactions only. Word transactions must have the least significant bit of the address set to zero. DWORD I/O operations should not be performed on these registers.

Some products may support access to these registers through MMIO. The access method varies and is documented elsewhere.

General Control and Status Registers

The setup, enable, and general registers are all directly accessible by the CPU. A sub indexing scheme is not used to read from and write to these registers.

Various bits in these registers provide control over the real-time status of the horizontal sync signal, the horizontal retrace interval, the vertical sync signal, and the vertical retrace interval. The horizontal retrace interval is the period during the drawing of each scan line containing active video data, when the active video data is not being displayed. This period includes the horizontal front and back porches, and the horizontal sync pulse. The horizontal retrace interval is always longer than the horizontal sync pulse. The vertical retrace interval is the period during which the scan lines not containing active video data are drawn. This includes the vertical front porch, back porch, and the vertical sync pulse. The vertical retrace interval is normally longer than the vertical sync pulse.

ST00 - Input Status 0

Address: 3C2h

Default: 00h

Attributes: Read Only

Bit	Descriptions
7	**CRT Interrupt Pending.** This bit is here for EGA compatibility and **will always return zero**. The generation of interrupts was originally enabled, through bits [4,5] of the Vertical Retrace End Register (CR11). This ability to generate interrupts at the start of the vertical retrace interval is a feature that is typically unused by DOS software and therefore is only supported through other means for use under a operating system support. 0 = CRT (vertical retrace interval) interrupt is not pending. 1 = CRT (vertical retrace interval) interrupt is pending
6:5	**Reserved.** Read as 0s.
4	**RGB Comparator / Sense.** This bit is here for compatibility and **will always return one**. Monitor detection must be done be done through the programming of hotplug registers in the MMIO space. 0 = Below threshold 1 = Above threshold
3:0	**Reserved.** Read as 0s.

ST01 - Input Status 1

Address: 3BAh/3DAh

Default: 00h

Attributes: Read Only

The address selection is dependent on CGA or MDA emulation mode as selected via the MSR register.

Bit	Descriptions
7	**Reserved (as per VGA specification).** Read as 0s.
6	**Reserved.** Read as 0.
5:4	**Video Feedback 1, 0.** These bits are connected to 2 of the 8 color bits sent to the palette. Bits 4 and 5 of the Color Plane Enable Register (AR12) selects which two of the 8 possible color bits become connected to these 2 bits of this register. These bits exist for EGA compatibility.
3	**Vertical Retrace/Video.** 0 = VSYNC inactive (Indicates that a vertical retrace interval is not taking place). 1 = VSYNC active (Indicates that a vertical retrace interval is taking place). VGA pixel generation is not locked to the display output but is loosely coupled. A VSYNC indication may not occur during the actual VSYNC going to the display but during the VSYNC that is generated as part of the VGA pixel generation. The exact relationship will vary with the VGA display operational mode. This status bit will remain active when the VGA is disabled, and the device is running in high resolution modes (non-VGA) to allow for applications that (now incorrectly) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. Bits 4 and 5 of the Vertical Retrace End Register (CR11) previously could program this bit to generate an interrupt at the start of the vertical retrace interval. This ability to generate interrupts at the start of the vertical retrace interval is a feature that is largely unused by legacy software. Interrupts are not supported through the VGA register bits.
2:1	**Reserved.** Read as 0s.
0	**Display Enable Output.** Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. This bit was used with the EGA graphics system (and the ones that preceded it, including MDA and CGA). In those cases, it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to the frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer. Those behaviors resulted in either "snow" or a flickering display. This bit provides compatibility with software designed for those early graphics controllers. This bit is currently used in DOS applications that access the palette to prevent the sparkle associated with read and write accesses to the palette RAM with the same address on the same clock cycle. **This status bit remains active when the VGA display is disabled, and the device is running in high resolution modes (non-VGA) to allow for applications that (now considered incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. When in panel fitting VGA or centered VGA operation, the meaning of these bits will not be consistent with native VGA timings.** 0 = Active display data is being sent to the display. Neither a horizontal retrace interval or a vertical retrace interval is currently taking place. 1 = Either a horizontal retrace interval (horizontal blanking) or a vertical retrace interval (vertical blanking) is currently taking place.

FCR - Feature Control

Address: 3BAh/3DAh - Write; 3CAh - Read

Default: 00h

Attributes: Read/Write

The address used for writes is dependent on CGA or MDA emulation mode as selected via the MSR register. In the original EGA, bits 0 and 1 were used as part of the feature connector interface. Feature connector is not supported in these devices and those bits will always read as zero.

Bit	Descriptions
7:4	**Reserved.** Read as 0.
3	**VSYNC Control.** This bit is provided for compatibility only and has no other function. Reads and writes to this bit have no effect other than to change the value of this bit. The previous definition of this bit selected the output on the VSYNC pin. 0 = Was used to set VSYNC output on the VSYNC pin (default). 1 = Was used to set the logical 'OR' of VSYNC and Display Enable output on the VSYNC pin. This capability was not typically very useful.
2:0	**Reserved.** Read as 0.

MSR - Miscellaneous Output

Address: 3C2h - Write; 3CCh - Read

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7	**CRT VSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. Sync polarity was used in VGA to signal the monitor how many lines of active display are being generated. 0 = Positive Polarity (default). 1 = Negative Polarity.
6	**CRT HSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. 0 = Positive Polarity (default).

Bit	Descriptions
	1 = Negative Polarity
5	**Page Select.** In Odd/Even Memory Map Mode 1 (GR6), this bit selects the upper or lower 64 KB page in display memory for CPU access: 0 = Upper page (default) 1 = Lower page. Selects between two 64KB pages of frame buffer memory during standard VGA odd/even modes (modes 0h through 5h). Bit 1 of register GR06 can also program this bit in other modes. This bit is would normally set to 1 by the software.
4	**Reserved.** Read as 0.
3:2	**Clock Select.** These bits can select the dot clock source for the CRT interface. The bits should be used to select the dot clock in standard native VGA modes only. When in the centering or upper left corner modes, these bits should be set to have no effect on the clock rate. The actual frequencies that these bits select, if they have any affect at all, is programmable through the PLL MMIO registers. 00 = CLK0, 25.175 MHz (for standard VGA modes with 640 pixel (8-dot) horizontal resolution) (default) 01 = CLK1, 28.322 MHz. (for standard VGA modes with 720 pixel (9-dot) horizontal resolution) 10 = Was used to select an external clock (now unused) 11 = Reserved
1	**A0000-BFFFFh Memory Access Enable.** VGA Compatibility bit enables access to video memory (frame buffer) at A0000-BFFFFh. When disabled, accesses to VGA memory are blocked in this region. This bit is independent of and does not block CPU access to the video linear frame buffer at other addresses. 0 = Prevent CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture (default). 1 = Allow CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture. This memory must be mapped as UC by the CPU.
0	**I/O Address Select.** This bit selects 3Bxh or 3Dxh as the I/O address for the CRT Controller registers, the Feature Control Register (FCR), and Input Status Register 1 (ST01). Presently ignored (whole range is claimed), but will "ignore" 3Bx for color configuration or 3Dx for monochrome. It is typical in AGP chipsets to shadow this bit and properly steer I/O cycles to the proper bus for operation where a MDA exists on another bus such as ISA. 0 = Select 3Bxh I/O address (MDA emulation) (default). 1 = Select 3Dxh I/O address (CGA emulation).

In standard VGA modes using the analog VGA connector, bits 7 and 6 indicate which of the three standard VGA vertical resolutions the standard VGA display should use. Extended modes, including those with a vertical resolution of 480 scan lines, may use a setting of 0 for both of these bits. Different connector standards and timing standards specify the proper use of sync polarity. This setting was "reserved" in the VGA standard.

Analog CRT Display Sync Polarities

V	H	Display	Horizontal Frequency	Vertical Frequency
P	P	200 Line	15.7 KHz	60 Hz
N	P	350 Line	21.8 KHz	60 Hz
P	N	400 Line	31.5 KHz	70 Hz
N	N	480 Line	31.5 KHz	60 Hz

Sequencer Registers

To access registers the VGA Sequencer Index register (SRX) at address 3C4h is written with the index of the desired register. Then the desired register is accessed through the data port for the sequencer registers at address 3C5.

SRX - Sequencer Index

Address: 3C4h

Default: 00h

Attributes: Read/Write

Bit	Description
7:3	**Reserved.** Read as 0s.
2:0	**Sequencer Index.** This field contains a 3-bit Sequencer Index value used to access sequencer data registers at indices 0 through 7.

SR00 - Sequencer Reset

Address: 3C5h(Index=00h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:2	**Reserved.** Read as 0.
1	**Reserved.** Reserved for VGA compatibility (was reset).
0	**Reserved.** Reserved for VGA compatibility. (was reset)

SR01 - Clocking Mode

Address: 3C5h (Index=01h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
5	**Screen Off.** 0 = Normal Operation (default). 1 = Disables video output (blanks the screen) and turns off display data fetches. Synchronization pulses to the display, however, are maintained. Setting this bit to 1 had been used as a way to more rapidly update and improve CPU access performance to the frame buffer during VGA modes. In non-VGA modes (VGA Disable=1), this bit has no effect. Before the VGA is disabled through the MMIO VGA control register, this bit should be set to stop the memory accesses from the display. The following sequence must be used when disabling the VGA plane. 1. Write SR01 to set bit 5 = 1 to disable video output. 2. Wait for 100us. 3. Disable the VGA plane via Bit 31 of the MMIO VGA control register (location found in the MMIO display register programming specification).
4	**Shift 4.** 0 = Load video shift registers every 1 or 2 character clocks (depending on bit 2 of this register) (default). 1 = Load shift registers every 4th character clock.
3	**Dot Clock Divide.** Setting this bit to 1 stretches doubles all horizontal timing periods that are specified in the VGA horizontal CRTC registers. This bit is used in standard VGA 40-column text modes to stretch timings to create horizontal resolutions of either 320 or 360 pixels (as opposed to 640 or 720 pixels, normally used in standard VGA 80-column text modes). The effect of this is that there will actually be twice the number of pixels sent to the display per line. 0 = Pixel clock is left unaltered (used for 640 (720) pixel modes); (default). 1 = Pixel clock divided by 2 (used for 320 (360) pixel modes).
2	**Shift Load.** Bit 4 of this register must be 0 for this bit to be effective. 0 = Load video data shift registers every character clock (default). 1 = Load video data shift registers every other character clock.
1	**Reserved.** Read as 0.
0	**8/9 Dot Clocks.** This bit determines whether a character clock is 8 or 9 dot clocks long if clock doubling is disabled and 16 or 18 clocks if it is. This also changes the interpretation of the pixel panning values (see chart). An additional control bit determines if this bit is to be ignored and 8-dot characters are to be used always. The 9-dot disable would be used when doubling the horizontal pixels on a 1280 wide display or non-doubling on a 640 wide display. Panning however will occur according to the expected outcome. 0 = 9 dot clocks (9 horizontal pixels) per character in text modes with a horizontal resolution of 720 pixels. 1 = 8 dot clocks (8 horizontal pixels) per character in text or graphics modes with a horizontal resolution of 640 pixels.

SR02 - Plane/Map Mask

Address: 3C5h (Index=02h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:4	**Reserved.** Read as 0s.
3:0	**Memory Planes [3:0] Processor Write Access Enable.** In both the Odd/Even Mode and the Chain 4 Mode, these bits still control access to the corresponding color plane. 0 = Disable. 1 = Enable. This register is referred to in the VGA standard as the Map Mask Register.

SR03 - Character Font

Address: 3C5h (index=03h)

Default: 00h

Attributes: Read/Write

In text modes, bit 3 of the video data's attribute byte normally controls the foreground intensity. This bit may be redefined to control switching between character sets. This latter function is enabled whenever there is a difference in the values of the Character Font Select A and the Character Font Select B bits. If the two values are the same, the character select function is disabled and attribute bit 3 controls the foreground intensity.

Bit 1 of the Memory Mode Register (SR04) must be set to 1 for the character font select function of this register to be active. Otherwise, only character maps 0 and 4 are available.

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
3:2,5	**Character Map Select Bits for Character Map B.** These three bits are used to select the character map (character generator tables) to be used as the secondary character set (font). The numbering of the maps is not sequential.

Bit [3:2,5]	Map Number	Table Location
00,0	0	1st 8KB of plane 2 at offset 0 (default)
00,1	4	2nd 8KB of plane 2 at offset 8K
01,0	1	3rd 8KB of plane 2 at offset 16K
01,1	5	4th 8KB of plane 2 at offset 24K
10,0	2	5th 8KB of plane 2 at offset 32K
10,1	6	6th 8KB of plane 2 at offset 40K
11,0	3	7th 8KB of plane 2 at offset 48K

Bit	Descriptions
	11,1 | 7 | 8th 8KB of plane 2 at offset 56K
1:0,4	**Character Map Select Bits for Character Map A.** These three bits are used to select the character map (character generator tables) to be used as the primary character set (font). The numbering of the maps is not sequential.

Bit [1:0,4]	Map Number	Table Location
00,0	0	1st 8KB of plane 2 at offset 0 (default)
00,1	4	2nd 8KB of plane 2 at offset 8K
01,0	1	3rd 8KB of plane 2 at offset 16K
01,1	5	4th 8KB of plane 2 at offset 24K
10,0	2	5th 8KB of plane 2 at offset 32K
10,1	6	6th 8KB of plane 2 at offset 40K
11,0	3	7th 8KB of plane 2 at offset 48K
11,1	7	8th 8KB of plane 2 at offset 56K

SR04 - Memory Mode Register

Address: 3C5h (index=04h)

Default: 00h

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3	**Chain 4 Mode.** The selections made by this bit affect both CPU Read and write accesses to the frame buffer. 0 = The manner in which the frame buffer memory is mapped is determined by the setting of bit 2 of this register (default). 1 = The frame buffer memory is mapped in such a way that the function of address bits 0 and 1 are altered so that they select planes 0 through 3. This setting is used in mode x13 to allow all four planes to be accessed via sequential addresses.
2	**Odd/Even Mode.** Bit 3 of this register must be set to 0 for this bit to be effective. The selections made by this bit affect only non-paged CPU accesses to the frame buffer through the VGA aperture. 0 = The frame buffer memory is mapped in such a way that the function of address bit 0 such that even addresses select planes 0 and 2 and odd addresses select planes 1 and 3 (default). 1 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).
1	**Extended Memory Enable.** This bit must be set to 1 to enable the selection and use of character maps in plane 2 via the Character Map Select Register (SR03). 0 = Disable CPU accesses to more than the first 64KB of VGA standard memory (default). 1 = Enable CPU accesses to the rest of the 256KB total VGA memory beyond the first 64KB.
0	**Reserved.** Read as 0.

SR07 - Horizontal Character Counter Reset

Address: 3C5h (index=07h)

Default: 00h

Attributes: Read/Write

For standard VGAs, writing this register (with any data) causes the horizontal character counter to be held in reset (the character counter output will remain 0). It remained in reset until a write occurred to any other sequencer register location with SRX set to an index of 0 through 6. In this implementation that sequence has no such special effect.

The vertical line counter is clocked by a signal derived from the horizontal display enable (which does not occur if the horizontal counter is held in reset). Therefore, if a write occurs to this register during the vertical retrace interval, both the horizontal and vertical counters will be set to 0. A write to any other sequencer register location (with SRX set to an index of 0 through 6) may then be used to start both counters with reasonable synchronization to an external event via software control. Although this was a standard VGA register, it was not documented.

Bit	Description
7:0	**Horizontal Character Counter.**

Graphics Controller Registers

Accesses to the registers of the VGA Graphics Controller are done through the use of address 3CEh written with the index of the desired register. Then the desired register is accessed through the data port for the graphics controller registers at address 3CFh. Indexes 10 and 11 must only be accessed through the I/O space.

GRX - GRX Graphics Controller Index Register

Address: 3CEh

Default: 000UUUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:0	**Graphics Controller Register Index.** This field selects any one of the graphics controller registers (GR00-GR18) to be accessed via the data port at address 3CFh.

GR00 - Set/Reset Register

Address: 3CFh (index=00h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Set/Reset Plane [3:0].** When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 0, all 8 bits of each byte of each memory plane are set to either 1 or 0 as specified in the corresponding bit in this register, if the corresponding bit in the Enable Set/Reset Register (GR01) is set to 1. When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 3, all CPU data written to the frame buffer is rotated, then logically ANDed with the contents of the Bit Mask Register (GR08), and then treated as the addressed data's bit mask, while value of these four bits of this register are treated as the color value.

GR01 - Enable Set/Reset Register

Address: 3CFh (Index=01h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Enable Set/Reset Plane [3:0].** This register works in conjunction with the Set/Reset Register (GR00). The Write Mode bits (bits 0 and 1) must be set for Write Mode 0 for this register to have any effect. 0 = The corresponding memory plane can be read from or written to by the CPU without any special bitwise operations taking place. 1 = The corresponding memory plane is set to 0 or 1 as specified in the Set/Reset Register (GR00).

GR02 - Color Compare Register

Address: 3CFh (Index=02h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Color Compare Plane [3:0].** When the Read Mode bit (bit 3) of the Graphics Mode Register (GR05) is set to select Read Mode 1, all 8 bits of each byte of each of the 4 memory planes of the frame buffer corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison, wherein value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.

GR03 - Data Rotate Register

Address: 3CFh (Index=03h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:3	**Function Select.** These bits specify the logical function (if any) to be performed on data that is meant to be written to the frame buffer (using the contents of the memory read latch) just before it is actually stored in the frame buffer at the intended address location. 00 = Data being written to the frame buffer remains unchanged and is simply stored in the frame buffer. 01 = Data being written to the frame buffer is logically ANDed with the data in the memory read latch before it is actually stored in the frame buffer. 10 = Data being written to the frame buffer is logically ORed with the data in the memory read latch before it is actually stored in the frame buffer. 11 = Data being written to the frame buffer is logically XORed with the data in the memory read latch before it is actually stored in the frame buffer.

Bit	Description
2:0	**Rotate Count.** These bits specify the number of bits to the right to rotate any data that is meant to be written to the frame buffer just before it is actually stored in the frame buffer at the intended address location.

GR04 - Read Plane Select Register

Address: 3CFh (Index=04h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**Read Plane Select.** These two bits select the memory plane from which the CPU reads data in Read Mode 0. In Odd/Even Mode, bit 0 of this register is ignored. In Chain 4 Mode, both bits 1 and 0 of this register are ignored. The four memory planes are selected as follows: 00 = Plane 0 01 = Plane 1 10 = Plane 2 11 = Plane 3 These two bits also select which of the four memory read latches may be read via the Memory read Latch Data Register (CR22). The choice of memory read latch corresponds to the choice of plane specified in the table above. The Memory Read Latch Data register and this additional function served by 2 bits are features of the VGA standard that were never documented.

GR05 - Graphics Mode Register

Address: 3CFh (Index=05h)

Default: 0UUU U0UUb (U=Undefined)

Attributes: Read/Write

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved. Read as 0.</td></tr>
<tr><td>6:5</td><td>
Shift Register Control. In standard VGA modes, pixel data is transferred from the 4 graphics memory planes to the palette via a set of 4 serial output bits. These 2 bits of this register control the format in which data in the 4 memory planes is serialized for these transfers to the palette.

Bits [6:5]=00

One bit of data at a time from parallel bytes in each of the 4 memory planes is transferred to the palette via the 4 serial output bits, with 1 of each of the serial output bits corresponding to a memory plane. This provides a 4-bit value on each transfer for 1 pixel, making possible a choice of 1 of 16 colors per pixel.

Serial
<table>
<tr><td>Out</td><td>1st Xfer</td><td>2nd Xfer</td><td>3rd Xfer</td><td>4th Xfer</td><td>5th Xfer</td><td>6th Xfer</td><td>7th Xfer</td><td>8th Xfer</td></tr>
<tr><td>Bit 3</td><td>plane3 bit7</td><td>plane3 bit6</td><td>plane3 bit5</td><td>plane3 bit4</td><td>plane3 bit3</td><td>plane3 bit2</td><td>plane3 bit1</td><td>plane3 bit0</td></tr>
<tr><td>Bit 2</td><td>plane2 bit7</td><td>plane2 bit6</td><td>plane2 bit5</td><td>plane2 bit4</td><td>plane2 bit3</td><td>plane2 bit2</td><td>plane2 bit1</td><td>plane2 bit0</td></tr>
<tr><td>Bit 1</td><td>plane1 bit7</td><td>plane1 bit6</td><td>plane1 bit5</td><td>plane1 bit4</td><td>plane1 bit3</td><td>plane1 bit2</td><td>plane1 bit1</td><td>plane1 bit0</td></tr>
<tr><td>Bit 0</td><td>plane0 bit7</td><td>plane0 bit6</td><td>plane0 bit5</td><td>plane0 bit4</td><td>plane0 bit3</td><td>plane0 bit2</td><td>plane0 bit1</td><td>plane0 bit0</td></tr>
</table>
Bits [6:5]=01

Two bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that alternates per byte between memory planes 0 and 2, and memory planes 1 and 3. First the even-numbered and odd-numbered bits of a byte in memory plane 0 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of a byte in memory plane 2 are transferred via serial output bits 2 and 3. Next, the even-numbered and odd-numbered bits of a byte in memory plane 1 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of memory plane 3 are transferred via serial out bits 1 and 3. This provides a pair of 2-bit values (one 2-bit value for each of 2 pixels) on each transfer, making possible a choice of 1 of 4 colors per pixel.
</td></tr>
</table>

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td></td><td>
Serial
<table>
<tr><td>Out</td><td>1st Xfer</td><td>2nd Xfer</td><td>3rd Xfer</td><td>4th Xfer</td><td>5th Xfer</td><td>6th Xfer</td><td>7th Xfer</td><td>8th Xfer</td></tr>
<tr><td>Bit 3</td><td>plane2 bit7</td><td>plane2 bit5</td><td>plane2 bit3</td><td>plane2 bit1</td><td>plane3 bi7t</td><td>plane3 bit5</td><td>plane3 bit3</td><td>plane3 bit1</td></tr>
<tr><td>Bit 2</td><td>plane2 bit6</td><td>plane2 bit4</td><td>plane2 bit2</td><td>plane2 bit0</td><td>plane3 bit6</td><td>plane3 bit4</td><td>plane3 bit2</td><td>plane3 bit0</td></tr>
<tr><td>Bit 1</td><td>plane0 bit7</td><td>plane0 bit5</td><td>plane0 bit3</td><td>plane0 bit1</td><td>plane1 bit7</td><td>plane1 bit5</td><td>plane1 bit3</td><td>plane1 bit1</td></tr>
<tr><td>Bit 0</td><td>plane0 bit6</td><td>plane0 bit4</td><td>plane0 bit2</td><td>plane0 bit0</td><td>plane1 bit6</td><td>plane1 bit4</td><td>plane1 bit2</td><td>plane1 bit0</td></tr>
</table>
This alternating pattern is meant to accommodate the use of the Odd/Even mode of organizing the 4 memory planes, which is used by standard VGA modes 2h and 3h.

Bits [6:5]=1x

Four bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that iterates per byte through memory planes 0 through 3. First the 4 most significant bits of a byte in memory plane 0 are transferred via the 4 serial output bits, followed by the 4 least significant bits of the same byte. Next, the same transfers occur from the parallel byte in memory planes 1, 2 and lastly, 3. Each transfer provides either the upper or lower half of an 8 bit value for the color for each pixel, making possible a choice of 1 of 256 colors per pixel. This is the setting used in mode x13.

Serial
<table>
<tr><td>Out</td><td>1st Xfer</td><td>2nd Xfer</td><td>3rd Xfer</td><td>4th Xfer</td><td>5th Xfer</td><td>6th Xfer</td><td>7th Xfer</td><td>8th Xfer</td></tr>
<tr><td>Bit 3</td><td>plane0 bit7</td><td>plane0 bit3</td><td>plane1 bit7</td><td>plane1 bit3</td><td>plane2 bit7</td><td>plane2 bit3</td><td>plane3 bit7</td><td>plane3 b3it</td></tr>
<tr><td>Bit 2</td><td>plane0 bit6</td><td>plane0 bit2</td><td>plane1 bit6</td><td>plane1 bit2</td><td>plane2 bit6</td><td>plane2 bit2</td><td>plane3 bit6</td><td>plane3 bit2</td></tr>
<tr><td>Bit 1</td><td>plane0 bit5</td><td>plane0 bit1</td><td>plane1 bit5</td><td>plane1 bit1</td><td>plane2 bit5</td><td>plane2 bit1</td><td>plane3 bit5</td><td>plane3 bit1</td></tr>
<tr><td>Bit 0</td><td>plane0 bit4</td><td>plane0 bit0</td><td>plane1 bit4</td><td>plane1 bit0</td><td>plane2 bit4</td><td>plane2 bit0</td><td>plane3 bit4</td><td>plane3 bit0</td></tr>
</table>
This pattern is meant to accommodate mode 13h, a standard VGA 256-color graphics mode.
</td></tr>
<tr><td>4</td><td>
Odd/Even Mode.

0 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).

1 = The frame buffer is mapped in such a way that the function of address bit 0 is such that even addresses select memory planes 0 and 2 and odd addresses select memory planes 1 and 3.

This works in a way that is the inverse of (and is normally set to be the opposite of) bit 2 of the
</td></tr>
</table>

Bit	Description
	Memory Mode Register (SR02).
3	**Read Mode.** 0 = During a CPU read from the frame buffer, the value returned to the CPU is data from the memory plane selected by bits 1 and 0 of the Read Plane Select Register (GR04). 1 = During a CPU read from the frame buffer, all 8 bits of the byte in each of the 4 memory planes corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison. A value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all 4 of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.
2	**Reserved.** Read as 0.
1:0	**Write Mode.** 00 = Write Mode 0 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written with the CPU write data after it has been rotated by the number of counts specified in the Data Rotate Register (GR03). If, however, the bit(s) in the Enable Set/Reset Register (GR01) corresponding to one or more of the memory planes is set to 1, then those memory planes will be written to with the data stored in the corresponding bits in the Set/Reset Register (GR00). 01 = Write Mode 1 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written to with the data stored in the memory read latches. (The memory read latches stores an unaltered copy of the data last read from any location in the frame buffer.) 10 = Write Mode 2 - During a CPU write to the frame buffer, the least significant 4 data bits of the CPU write data is treated as the color value for the pixels in the addressed byte in all 4 memory planes. The 8 bits of the Bit Mask Register (GR08) are used to selectively enable or disable the ability to write to the corresponding bit in each of the 4 memory planes that correspond to a given pixel. A setting of 0 in a bit in the Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with value of their counterparts in the memory read latches. A setting of 1 in a Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with the 4 bits taken from the CPU write data to thereby cause the pixel

Bit	Description
	corresponding to these bits to be set to the color value. 11 = Write Mode 3 - During a CPU write to the frame buffer, the CPU write data is logically ANDed with the contents of the Bit Mask Register (GR08). The result of this ANDing is treated as the bit mask used in writing the contents of the Set/Reset Register (GR00) are written to addressed byte in all 4 memory planes.

GR06 - Miscellaneous Register

Address: 3CFh (Index=06h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0s.
3:2	**Memory Map Mode.** These 2 bits control the mapping of the VGA address range for frame buffer into the CPU address space as follows: 00 = A0000h - BFFFFh 01 = A0000h - AFFFFh 10 = B0000h - B7FFFh 11 = B8000h - BFFFFh This function is used in standard VGA modes, extended VGA modes (132 column text), and in non-VGA modes (hi-res). 132 column text modes are no longer supported. VGA aperture memory accesses are also controlled by the PCI configuration Memory Enable bit and MSR<1>. For accesses using GR10 and GR11 to paged VGA RAM or to device MMIO registers, set these bits to 01 to select the (A0000-AFFFF) range. The CPU must map this memory as uncacheable (UC).
1	**Chain Odd/Even.** This bit provides the ability to alter the interpretation of address bit A0, so that it may be used in selecting between the odd-numbered memory planes (planes 1 and 3) and the even-numbered memory planes (planes 0 and 2). 0 = A0 functions normally. 1 = A0 is switched with a high order address bit, in terms of how it is used in address decoding. The result is that A0 is used to determine which memory plane is being accessed (A0=0 for planes 0 and 2 and A0=1 for planes 1 and 3).

Bit	Description
0	**Graphics/Text Mode.** This is one of two bits that are used to determine if the VGA is operating in text or graphics modes. The other bit is in AR10[0], these two bits need to be programmed in a consistent manner to achieve the proper results. 0 = Text mode. 1 = Graphics mode.

GR07 - Color Don't Care Register

Address: 3CFh (Index=07h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Ignore Color Plane [3:0].** These bits have effect only when bit 3 of the Graphics Mode Register (GR05) is set to 1 to select read mode 1. 0 = The corresponding bit in the Color Compare Register (GR02) will not be included in color comparisons. 1 = The corresponding bit in the Color Compare Register (GR02) is used in color comparisons.

GR08 - Bit Mask Register

Address: 3CFh (Index=08h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Bit Mask.** 0 = The corresponding bit in each of the 4 memory planes is written to with the corresponding bit in the memory read latches. 1 = Manipulation of the corresponding bit in each of the 4 memory planes via other mechanisms is enabled. This bit mask applies to any writes to the addressed byte of any or all of the 4 memory planes, simultaneously. This bit mask is applicable to any data written into the frame buffer by the CPU, including data that is also subject to rotation, logical functions (AND, OR, XOR), and Set/Reset. To perform a proper read-modify-write cycle into frame buffer, each byte must first be read from the frame buffer by the CPU (and this will

Bit	Description
	cause it to be stored in the memory read latches), this Bit Mask Register must be set, and the new data then written into the frame buffer by the CPU.

GR10 - Address Mapping

Address: 3CFh (Index=10h)

Default: 00h

Attributes: Read/Write

This register must only be accessed using I/O operations.

Bit	Description
7:4	**Page Select Extension - Unused** These bits form the upper bits of a 12-bit page selection value. When combined with the GR11 <7:0> bits they define the offset into stolen memory to the 64KB page that is accessible via the VGA Memory paging mechanism. These bits are ignored.
3	**Reserved**
2:1	**Paging Map Target.** When paging is enabled, these bits determine the target for data cycle accesses through the VGA memory aperture. VGA graphics memory starts from the base of graphics data stolen memory defined in the PCI configuration BDSM register. VGA display uses the first four 64KB pages of VGA graphics memory. 00 = VGA Graphics Memory 01 = Reserved 10 = Reserved 11 = Reserved
0	**Page Mapping Enable.** This mode allows the mapping of the VGA memory address space. Once this is enabled, no VGA memory address swizzle will be performed, addresses are directly mapped to memory. A single paging register is used to map the 64KB [A0000:AFFFF] window. An internal address is generated using GR11 as the address lines extension to the lower address lines of the access A[15:2]. When mapping is enabled, the B0000:BFFFF area must be disabled using GR06<3:2>=01. The use of addresses in the A0000-BFFFF range require that both the graphics device PCI configuration memory enable and MSR<1> be enabled. 0 = Disable (default)

Bit	Description
	1 = Enable

GR11 - Page Selector

Address: 3CFh (Index=11h)

Default: 00h

Attributes: Read/Write

This register must only be accessed using I/O operations.

Bit	Description
7	**Reserved**
6:0	**Page Select.** When concatenated with the GR10<7:4> bits, selects a 64KB window within target area when Page Mapping is enabled (GR10[0]=1). This requires that the graphics device PCI configuration space memory enable, the GR06<3:2> bits to be 01 (select A0000-AFFFF only), and the MSR<1:1> bit to be set. This register provides the Address[22:16] bits for the access. VGA paging of frame buffer memory is for non-VGA packed modes only and should not be enabled when using basic VGA modes.

GR18 - Software Flags

Address: 3CFh (Index=18h)

Default: 00h

Attributes: Read/Write

Bit	Description
7:0	**Software Flags.** Used as scratch pad space in video BIOS. These bits are separate from the bits which appear in the MMIO space. They are used specifically by the SMI BIOS which does not have access to MMIO at the time they are required. These register bits have no effect on H/W operation.

Attribute Controller Registers

Unlike the other sets of indexed registers, the attribute controller registers are not accessed through a scheme employing entirely separate index and data ports. Address 3C0h is used both as the read and write for the index register, and as the write address for the data port. Address 3C1h is the read address for the data port.

To write to the attribute controller registers, the index of the desired register must be written to address 3C0h, and then the data is written to the very same address. A flip-flop alternates with each write to address 3C0h to change its function from writing the index to writing the actual data, and back again. This flip-flop may be deliberately set so that address 3C0h is set to write to the index (which provides a way to set it to a known state) by performing a read operation from Input Status Register 1 (ST01) at address 3BAh or 3DAh, depending on whether the graphics system has been set to emulate an MDA or a CGA as per MSR[0].

To read from the attribute controller registers, the index of the desired register must be written to address 3C0h, and then the data is read from address 3C1h. A read operation from address 3C1h does not reset the flip-flop to writing to the index. Only a write to 3C0h or a read from 3BAh or 3DAh, as described above, will toggle the flip-flop back to writing to the index.

ARX - Attribute Controller Index Register

Address: 3C0h

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0s.
5	**Video Enable.** In the VGA standard, this is called the "Palette Address Source" bit. Clearing this bit will cause the VGA display data to become all 00 index values. For the default palette, this will cause a black screen. The video timing signals continue. Another control bit will turn video off and stop the data fetches. 0 = Disable. Attribute controller color registers (AR[00:0F]) can be accessed by the CPU. 1 = Enable. Attribute controller color registers (AR[00:0F]) are inaccessible by the CPU.
4:0	**Attribute Controller Register Index.** These five bits are used to select any one of the attribute controller registers (AR[00:14]), to be accessed.

AR[00:0F] - Palette Registers [0:F]

Address: Read at 3C1h and Write at 3C0h; (index=00h-0Fh)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:0	**Palette Bits P[5:0].** In each of these 16 registers, these are the lower 6 of 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors available to be selected in the palette. Bits 3 and 2 of the Color Select Register (AR14) supply bits P7 and P6 for the values contained in all 16 of these registers. Bits 1 and 0 of the Color Select Register (AR14) can also replace bits P5 and P4 for the values contained in all 16 of these registers, if bit 7 of the Mode Control Register (AR10) is set to 1.

AR10 - Mode Control Register

Address: Read at 3C1h and Write at 3C0h; (index=10h)

Default: UUh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Palette Bits P5, P4 Select.** **0** = P5 and P4 for each of the 16 selected colors (for modes that use 16 colors) are individually provided by bits 5 and 4 of their corresponding Palette Registers (AR[00:0F]). **1** = P5 and P4 for all 16 of the selected colors (for modes that use 16 colors) are provided by bits 1 and 0 of Color Select Register (AR14).
6	**Pixel Width/Clock Select.** **0** = Six bits of video data (translated from 4 bits via the palette) are output every dot clock. **1** = Two sets of 4 bits of data are assembled to generate 8 bits of video data which is output every other dot clock, and the Palette Registers (AR[00:0F]) are bypassed. This bit is set to 0 for all of the standard VGA modes, except mode 13h.
5	**Pixel Panning Compatibility.** **0** = Scroll both the upper and lower screen regions horizontally as specified in the Pixel Panning Register (AR13). **1** = Scroll only the upper screen region horizontally as specified in the Pixel Panning Register (AR13).

Bit	Description
	This bit has application only when split-screen mode is being used, where the display area is divided into distinct upper and lower regions which function somewhat like separate displays.
4	**Reserved.** Read as 0.
3	**Enable Blinking/Select Background Intensity.** **0** = Disables blinking in graphics modes, and for text modes, sets bit 7 of the character attribute bytes to control background intensity, instead of blinking. **1** = Enables blinking in graphics modes and for text modes, sets bit 7 of the character attribute bytes to control blinking, instead of background intensity. The blinking rate is derived by dividing the VSYNC signal. The Blink Rate Control field of the VGA control register defines the blinking rate.
2	**Enable Line Graphics Character Code.** **0** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the background of the character of which the given pixel is a part. **1** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the 8th pixel if the character of which the given pixel is a part. This setting is intended to accommodate the line-drawing characters of the PC's extended ASCII character set -- characters with an extended ASCII code in the range of B0h to DFh. In some literature describing the VGA standard, the range of extended ASCII codes that are said to include the line-drawing characters is mistakenly specified as C0h to DFh, rather than the correct range of B0h to DFh.
1	**Select Display Type.** **0** = Attribute bytes in text modes are interpreted as they would be for a color display. **1** = Attribute bytes in text modes are interpreted as they would be for a monochrome display.
0	**Graphics/Alphanumeric Mode.** This bit (along with GR06[0]) select either graphics mode or text mode. These two bits must be programmed in a consistent manner to achieve the desired results. **0** = Alphanumeric (text) mode. **1** = Graphics mode.

AR11 - Overscan Color Register

Address: Read at 3C1h and Write at 3C0h; (index=11h)

Default: UUh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:0	**Overscan.** These 8 bits select the overscan (border) color index value. The actual border color will be determined by the contents of the palette at the selected index. The border color is displayed between the end of active and the beginning of blank or the end of blank and the beginning of active on CRT type devices driven from the DAC output port. For native VGA modes on digital display ports, some devices have the option of including the border in the active region or not, depending on a control bit in the port control register. For centered VGA modes, the VGA control register determines if the border is included in the centered region or not. For monochrome displays, this value should be set to 00h.

AR12 - Memory Plane Enable Register

Address: Read at 3C1h and Write at 3C0h; (index=12h)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:4	**Video Status Mux.** These 2 bits are used to select 2 of the 8 possible palette bits (P7-P0) to be made available to be read via bits 5 and 4 of the Input Status Register 1 (ST01). The table below shows the possible choices. **Bit [5:4]** / **ST01 Bit 5** / **ST01 Bit 4** 00 / P2 (default) / P0 (default) 01 / P5 / P4 10 / P3 / P1 11 / P7 / P6 These bits are typically unused by current software; they are provided for EGA compatibility.
3:0	**Enable Plane [3:0].** These 4 bits individually enable the use of each of the 4 memory planes in providing 1 of the 4 bits used in video output to select 1 of 16 possible colors from the palette to be displayed. 0 = Disable the use of the corresponding memory plane in video output to select colors, forcing the bit that the corresponding memory plane would have provided to a value of 0.

Bit	Description
	1 = Enable the use of the corresponding memory plane in video output to select colors. AR12 is referred to in the VGA standard as the Color Plane Enable Register.

AR13 - Horizontal Pixel Panning Register

Address: Read at 3C1h and Write at 3C0h; (index=13h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:0	**Horizontal Pixel Shift 3-0.** This field holds a 4-bit value that selects the number of pixels by which the image is shifted horizontally to the left. This function is available in both text and graphics modes and allows for pixel panning. In text modes with a 9-pixel wide character box, the image can be shifted up to 9 pixels to the left. In text modes with an 8-pixel wide character box, and in graphics modes other than those with 256 colors, the image can be shifted up to 8 pixels to the left. A pseudo 9-bit mode is when the 9-dot character is selected but overridden by the VGA control bit. In standard VGA mode 13h (where bit 6 of the Mode Control Register, AR10, is set to 1 to support 256 colors), bit 0 of this register must remain set to 0, and the image may be shifted up to only 4 pixels to the left. In this mode, the number of pixels by which the image is shifted can be further controlled using bits 6 and 5 of the Preset Row Scan Register (CR08).

Number of Pixels Shifted				
Bits [3:0]	**9-dot**	**Pseudo 9-dot**	**8-dot**	**256-Color**
0	1	1	0	0
1	2	2	1	Undefined
2	3	3	2	1
3	4	4	3	Undefined
4	5	5	4	2
5	6	6	5	Undefined
6	7	7	6	3
7	8	7	7	Undefined
8	0	0	Undefined	Undefined

AR14 - Color Select Register

Address: Read at 3C1h and Write at 3C0h; (index=14h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:2	**Palette Bits P[7:6].** These are the 2 upper-most of the 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors contained in the palette. These 2 bits are common to all 16 sets of bits P5 through P0 that are individually supplied by Palette Registers 0-F (AR[00:0F]).
1:0	**Alternate Palette Bits P[5:4].** These 2 bits can be used as an alternate version of palette bits P5 and P4. Unlike the P5 and P4 bits that are individually supplied by Palette Registers 0-F (AR[00:0F]), these 2 alternate palette bits are common to all 16 of Palette Registers. Bit 7 of the Mode Control Register (AR10) is used to select between the use of either the P5 and P4 bits that are individually supplied by the 16 Palette Registers or these 2 alternate palette bits.

VGA Color Palette Registers

In devices that have multiple display pipes, there is one palette for each display pipe. These palettes are the same for VGA modes and non-VGA modes. Accesses through VGA register methods will read or write from the palette of the pipe selected through MMIO VGA control register.

For each palette, the color data stored in these 256 color data positions can be accessed only through a complex sub-addressing scheme, using a data register and two index registers. The Palette Data Register at address 3C9h is the data port. The Palette Read Index Register at address 3C7h and the Palette Write Index Register at address 3C8h are the two index registers. The Palette Read Index Register is the index register that is used to choose the color data position that is to be read from via the data port, while the Palette Write Index Register is the index register that is used to choose the color data position that is to be written to through the same data port. This arrangement allows the same data port to be used for reading from and writing to two different color data positions. Reading and writing the color data at a color data position involves three successive reads or writes since the color data stored at each color data position consists of three bytes.

To read a palette color data position, the index of the desired color data position must first be written to the Palette Read Index Register. Then all three bytes of data in a given color data position may be read at the Palette Data Register. The first byte read from the Palette Data Register retrieves the 8-bit value specifying the intensity of the red color component. The second and third bytes read are the corresponding 8-bit values for the green and blue color components respectively. After completing the third read operation, the Palette Read Index Register is automatically incremented so that the data of the next color data position becomes accessible for being read. This allows the contents of all of the 256 color data positions of the palette to be read in sequence. This is done by specifying only the index of the 0th color data position in the Palette Read Index Register, and then simply performing 768 successive reads from the Palette Data Register.

Writing a color data position, entails a very similar procedure. The index of the desired color data position must first be written to the Palette Write Index Register. Then all three bytes of data to specify a given color may be written to the Palette Data Register. The first byte written to the Palette Data Register specifies the intensity of the red color component, the second byte specifies the intensity for the green color component, and the third byte specifies the same for the blue color component. One important detail is that all three of these bytes must be written before the hardware will actually update these three values in the given color data position. When all three bytes have been written, the Palette Write Index Register is automatically incremented so that the data of the next color data position becomes accessible for being written. This allows the contents of all of the 256 color data positions of the palette to be written in sequence. This is done by specifying only the index of the 0th color data position in the Palette Write Index Register, and then simply performing 768 successive writes to the Palette Data Register.

DACMASK - Pixel Data Mask Register

Address: 3C6h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Pixel Data Mask.** In indexed-color mode, the 8 bits of this register are logically ANDed with the 8 bits of pixel data received from the frame buffer for each pixel. The result of this ANDing process becomes the actual index used to select color data positions within the palette. This has the effect of limiting the choice of color data positions that may be specified by the incoming 8-bit data. 0 = Corresponding bit in the resulting 8-bit index being forced to 0. 1 = Allows the corresponding bit in the resulting index to reflect the actual value of the corresponding bit in the incoming 8-bit pixel data.

DACSTATE - DAC State Register

Address: 3C7h

Default: 00h

Attributes: Read Only

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**DACState.** This field indicates which of the two index registers was most recently written. **Bits [1:0]** Index Register Indicated 00 = Palette Write Index Register at Address 3C7h (default) 01 = Reserved 10 = Reserved 11 = Palette Read Index Register at Address 3C8h

DACRX - Palette Read Index Register

Address: 3C7h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Read Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being read from via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been read. A write to this register will abort an uncompleted palette write sequence. This register allows access to the palette even when running non-VGA display modes.

DACWX - Palette Write Index Register

Address: 3C8h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Write Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being written via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been written. This register allows access to the palette even when running non-VGA display modes.

DACDATA - Palette Data Register

Address: 3C9h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Palette Data.** This byte-wide data port provides read or write access to the three bytes of data of each color data position selected using the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX). The three bytes in each color data position are read or written in three successive read or write operations. The first byte read or written specifies the intensity of the red component of the color specified in the selected color data position. The second byte is for the green component, and the third byte is for the blue component. When writing data to a color data position, all three bytes must be

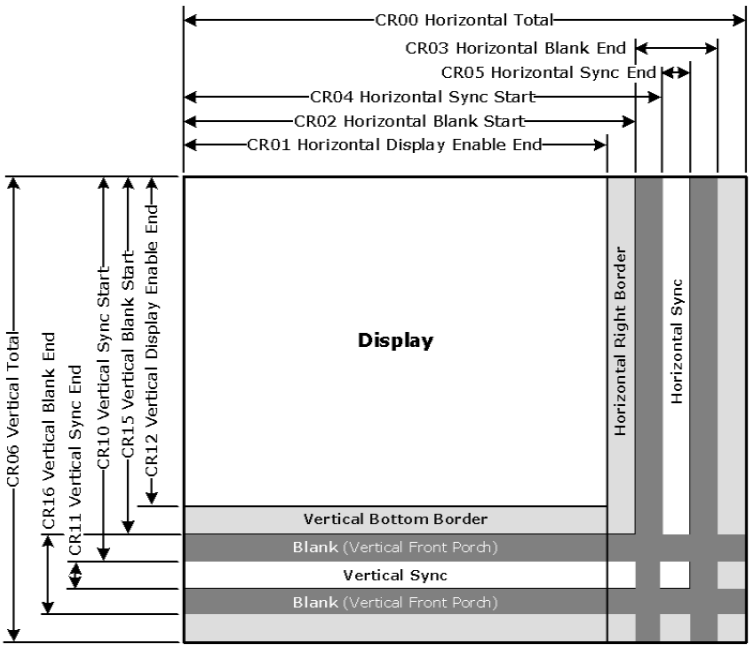
Bit	Description
	written before the hardware will actually update the three bytes of the selected color data position. When reading or writing to a color data position, ensure that neither the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX) are written to before all three bytes are read or written. A write to either of these two registers causes the circuitry that automatically cycles through providing access to the bytes for red, green and blue components to be reset such that the byte for the red component is the one that will be accessed by the next read or write operation via this register. This register allows access to the palette even when running non-VGA display modes. Writes to the palette can cause sparkle if not done during inactive video periods. This sparkle is caused by an attempt to write and read the same address on the same cycle. Some devices contain anti-sparkle circuits which will substitute the previous pixel value for the read output.

CRT Controller Register

For native VGA modes, the CRTC registers determine the display timing that is to be used. In centered VGA modes, these registers determine the size of the VGA image that is to be centered in the larger timing generator defined rectangle.

The CRT controller registers are accessed by writing the index of the desired register into the CRT Controller Index Register at address 3B4h or 3D4h, depending on whether the graphics system is configured for MDA or CGA emulation. The desired register is then accessed through the data port for the CRT controller registers located at address 3B5h or 3D5h, again depending upon the choice of MDA or CGA emulation as per MSR[0].

The following figure shows display fields and dimensions and the particular CRxx register that provides the control.



Group 0 Protection: In the original VGA, CR[0:7] could be made write-protected by CR11[7]. In BIOS code, this write protection is set following each mode change. Other protection groups have no current use, and would not be used going forward by the BIOS or by drivers. They are the result of an industry fad some years ago to attempt to write protect other groups of registers; however, all such schemes were chip specific. Only the write protection (Group 0 Protection) is supported.

CRX - CRT Controller Index Register

Address: 3B4h/3D4h

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:0	**CRT Controller Index.** These 7 bits are used to select any one of the CRT controller registers to be accessed via the data port at location 3B5h or 3D5h, depending upon whether the graphics system is configured for MDA or CGA emulation.

CR00 - Horizontal Total Register

Address: 3B5h/3D5h (index=00h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Total.** This register is used to specify the total length of each scan line. This encompasses both the part of the scan line that is within the active display area and the part that is outside of it. Programming this register to a zero has the effect of stopping the fetching of display data. This field should be programmed with a value equal to the total number of character clocks within the entire length of a scan line, minus 5.

CR01 - Horizontal Display Enable End Register

Address: 3B5h/3D5h (index=01h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Display Enable End.** This register is used to specify the end of the part of the scan line that is within the active display area relative to its beginning. In other words, this is the horizontal width of the active display area. This field should be programmed with a value equal to the number of character clocks that occur within the horizontal active display area, minus 1. Horizontal display enable will go active at the beginning of each line during vertical active area, it will go inactive based on the programming of this register or the programming of the horizontal total (CR00) register. When this register value is programmed to a number that is larger than the total number of characters on a line, display enable will be active for all but the last character of the horizontal display line.

CR02 - Horizontal Blanking Start Register

Address: 3B5h/3D5h (index=02h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Blanking Start.** This register is used to specify the beginning of the horizontal blanking period relative to the beginning of the active display area of a scan line. Horizontal blanking should always be set to start no sooner than after the end of horizontal active. This field should be programmed with a value equal to the number of character clocks that occur on a scan line from the beginning of the active display area to the beginning of the horizontal blanking.

CR03 - Horizontal Blanking End Register

Address: 3B5h/3D5h (index=03h)

Default: 1UUU UUUUb (U=Undefined)

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Reserved.** Values written to this bit are ignored, and to maintain consistency with the VGA standard, a value of 1 is returned when this bit is read. At one time, this bit was used to enable access to certain light pen registers. At that time, setting this bit to 0 provided this access, but setting this bit to 1 was necessary for normal operation.
6:5	**Display Enable Skew Control.** Defines the degree to which the start and end of the active display area are delayed along the length of a scan line to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = no delay 01 = delayed by 1 character clock 10 = delayed by 2 character clocks 11 = delayed by 3 character clocks
4:0	**Horizontal Blanking End Bits [4:0].** This field provides the 5 least significant bits of a 6-bit value that specifies the end of the blanking period relative to its beginning on a single scan line. Bit 7 of the Horizontal Sync End Register (CR05) supplies the most significant bit. This 6-bit value should be programmed to be equal to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02). End of blanking should occur before horizontal total.

CR04 - Horizontal Sync Start Register

Address: 3B5h/3D5h (index=04h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Sync Start** This register is used to specify the position of the beginning of the horizontal sync pulse relative to the start of the active display area on a scan line. This field should be set equal to the number of character clocks that occur from beginning of the active display area to the beginning of the horizontal sync pulse on a single scan line. Horizontal sync should always occur at least 2 clocks after the start of horizontal blank and 2 clocks before the end of horizontal blank. The actual start of sync will also be affected by both the horizontal sync skew register field and whether it is a text or graphics mode.

CR05 - Horizontal Sync End Register

Address: 3B5h/3D5h (index=05h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Horizontal Blanking End Bit 5.** This bit provides the most significant bit of a 6-bit value that specifies the end of the horizontal blanking period relative to its beginning. Bits [4:0] of Horizontal Blanking End Register (CR03) supplies the 5 least significant bits. See CR03[4:0] for further details. This 6-bit value should be set to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02).
6:5	**Horizontal Sync Delay.** This field defines the degree to which the start and end of the horizontal sync pulse are delayed to compensate for internal pipeline delays. This capability is supplied to implement VGA compatibility. These field describes the delay in terms of a number character clocks. **Bit [6:5]** — **Amount of Delay** 00 — no delay 01 — delayed by 1 character clock 10 — delayed by 2 character clocks 11 — delayed by 3 character clocks

Bit	Description
4:0	**Horizontal Sync End.** This field provides the 5 least significant bits of a 5-bit value that specifies the end of the horizontal sync pulse relative to its beginning. A value equal to the 5 least significant bits of the horizontal character counter value at which time the horizontal retrace signal becomes inactive (logical 0). Thus, this 5-bit value specifies the width of the horizontal sync pulse. To obtain a retrace signal of W, the following algorithm is used: Value of Horizontal Sync start Register (CR04) + width of horizontal retrace signal in character clock units = 5 bit result to be programmed in this field

CR06 - Vertical Total Register

Address: 3B5h/3D5h (index=06h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Vertical Total Bits [7:0].** This field provides the 8 least significant bits of either a 10-bit or 12-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. In standard VGA modes, the vertical total is specified with a 10-bit value. The 8 least significant bits of this value are supplied by these 8 bits of this register, and the 2 most significant bits are supplied by bits 5 and 0 of the Overflow Register (CR07).

CR07 - Overflow Register (Vertical)

Address: 3B5h/3D5h (index=07h)

Default: UU0U UUU0b (U=Undefined)

Attributes: Read/Write (Group 0 Protection on bits [7:5, 3:0])

Bit	Description
7	**Vertical Sync Start Bit 9.** The vertical sync start is a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by this bit and bit 2, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
6	**Vertical Display Enable End Bit 9.** The vertical display enable end is a 10-bit that specifies the number of the last scan line within the active display area. In standard VGA modes, the vertical display enable end is specified with a 10-bit value. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the most and second-most significant bits are supplied by this bit and bit 1, respectively, of this register. This 10-bit value should be programmed to be equal to

Bit	Description
	the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
5	**Vertical Total Bit 9.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by this bit and bit 0, respectively, of this register. This 10-bit value should be programmed equal to the total number of scan lines, minus 2.
4	**Line Compare Bit 8.** This bit provides the second most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display what data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display what data exists in the frame buffer starting at the first byte of the frame buffer.
3	**Vertical Blanking Start Bit 8.** The vertical blanking start is a 10-bit that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by bit 5 of the Maximum Scan Line Register (CR09) and this bit of this register, respectively. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
2	**Vertical Sync Start Bit 8.** The vertical sync start is a 10-bit value that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by bit 7 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

Bit	Description
1	**Vertical Display Enable End Bit 8.** The vertical display enable end is a 10-bit value that specifies the number of the last scan line within the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the two most significant bits are supplied by bit 6 and this bit, respectively, of this register. This 10-bit or value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
0	**Vertical Total Bit 8.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by bit 5 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the total number of scan lines, minus 2.

CR08 - Preset Row Scan Register

Address: 3B5h/3D5h (index=08h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0s.
6:5	**Byte Panning.** This field holds a 2-bit value that selects number of bytes (up to 3) by which the image is shifted horizontally to the left on the screen. This function is available in both text and graphics modes. In text modes with a 9-pixel wide character box, the image can be shifted up to 27 pixels to the left, in increments of 9 pixels. In text modes with an 8-pixel wide character box, and in all standard VGA graphics modes, the image can be shifted up to 24 pixels to the left, in increments of 8 pixels. When the Nine dot disable bit of the VGA control register is set, the pixel shift will be equivalent to the 8-dot mode. The image can be shifted still further, in increments of individual pixels, through the use of bits [3:0] of the Horizontal Pixel Panning Register (AR13).

Number of Pixels Shifted		
Bit [6:5]	**9-Pixel Text**	**8-Pixel Text & Graphics**
00	0	0
01	9	8
10	18	16
11	27	24

Bit	Description
4:0	**Starting Row Scan Count.** This field specifies which horizontal line of pixels within the character boxes of the characters used on the top-most row of text on the display will be used as the top-most scan line. The horizontal lines of pixels of a character box are numbered from top to bottom, with the top-most line of pixels being number 0. If a horizontal line of these character boxes other than the top-most line is specified, then the horizontal lines of the character box above the specified line of the character box will not be displayed as part of the top-most row of text characters on the display. Normally, the value specified by these 5 bits should be 0, so that all the horizontal lines of pixels within these character boxes will be displayed in the top-most row of text, ensuring that the characters in the top-most row of text do not look as though they have been cut off at the top.

CR09 - Maximum Scan Line Register

Address: 3B5h/3D5h (index=09h)

Default: 00h

Attributes: Read/Write

Bit	Description
7	**Double Scanning Enable.** 0 = Disable. When disabled, the clock to the row scan counter is equal to the horizontal scan rate. This is the normal setting for many of the standard VGA modes. 1 = Enable. When enabled, the clock to the row scan counter is divided by 2. This is normally used to allow CGA-compatible modes that have only 200 scan lines of active video data to be displayed as 400 scan lines (each scan line is displayed twice).
6	**Line Compare Bit 9.** This bit provides the most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 4 of the Overflow Register (CR07) supplies the second most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

Bit	Description
5	**Vertical Blanking Start Bit 9.** The vertical blanking start is a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by this bit and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal to the number of scan line from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
4:0	**Starting Row Scan Count.** This field provides all 5 bits of a 5-bit value that specifies the number of scan lines in a horizontal row of text. This value should be programmed to be equal to the number of scan lines in a horizontal row of text, minus 1.

CR0A - Text Cursor Start Register

Address: 3B5h/3D5h (index=0Ah)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5	**Text Cursor Off.** This text cursor exists only in text modes, so this register is entirely ignored in graphics modes. 0 = Enables the text cursor. 1 = Disables the text cursor.
4:0	**Text Cursor Start.** This field specifies which horizontal line of pixels in a character box is to be used to display the first horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the first horizontal line of pixels on which the cursor is to be shown.

CR0B - Text Cursor End Register

Address: 3B5h/3D5h (index=0Bh)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Text Cursor Skew.** This field specifies the degree to which the start and end of each horizontal line of pixels making up the cursor is delayed to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = No delay 01 = Delayed by 1 character clock 10 = Delayed by 2 character clocks 11 = Delayed by 3 character clocks
4:0	**Text Cursor End.** This field specifies which horizontal line of pixels in a character box is to be used to display the last horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the last horizontal line of pixels on which the cursor is to be shown.

CR0C - Start Address High Register

Address: 3B5h/3D5h (index=0Ch)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [15:8].** This register provides either bits 15 through 8 of a 16-bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes, the start address is specified with a 16-bit value. The eight bits of this register provide the eight most significant bits of this value, while the eight bits of the Start Address Low Register (CR0D) provide the eight least significant bits.

CR0D - Start Address Low Register

Address: 3B5h/3D5h (index=0Dh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [7:0]** This register provides either bits 7 through 0 of a 16 bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes the start address is specified with a 16-bit value. The eight bits of the Start Address High Register (CR0C) provide the eight most significant bits of this value, while the eight bits of this register provide the eight least significant bits.

CR0E - Text Cursor Location High Register

Address: 3B5h/3D5h (index=0Eh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [15:8].** This field provides the 8 most significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bit 7:0 of the Text Cursor Location Low Register (CR0F) provide the 8 least significant bits.

CR0F - Text Cursor Location Low Register

Address: 3B5h/3D5h (index=0Fh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [7:0].** This field provides the 8 least significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bits 7:0 of the Text Cursor Location High Register (CR0E) provide the 8 most significant bits.

CR10 - Vertical Sync Start Register

Address: 3B5h/3D5h (index=10h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Sync Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area of a screen. In standard VGA modes, this value is described in 10 bits with bits [7,2] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should equal the vertical sync start in terms of the number of scan lines from the beginning of the active display area to the beginning of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

CR11 - Vertical Sync End Register

Address: 3B5h/3D5h (index=11h)

Default: 0U00 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Protect Registers [0:7].** The ability to write to Bit 4 of the Overflow Register (CR07) is not affected by this bit (i.e., bit 4 of the Overflow Register is always writeable). 0 = Enable writes to registers CR[00:07]. (default) 1 = Disable writes to registers CR[00:07].
6	**Reserved.** In the VGA standard, this bit was used to switch between 3 and 5 frame buffer refresh cycles during the time required to draw each horizontal line.
5	**Vertical Interrupt Enable.** This bit is reserved for compatibility only. While this bit may be written or read, it's value will have no effect. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. Bit 7 of Input Status Register 0 (ST00) originally indicated the status of the vertical retrace interrupt. 0 = Enable the generation of an interrupt at the beginning of each vertical retrace period. 1 = Disable the generation of an interrupt at the beginning of each vertical retrace period.
4	**Vertical Interrupt Clear.** This is reserved for compatibility only. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller.

Bit	Description
	0 = Setting this bit to 0 clears a pending vertical retrace interrupt. This bit must be set back to 1 to enable the generation of another vertical retrace interrupt.
3:0	**Vertical Sync End.** This 4-bit field provides a 4-bit value that specifies the end of the vertical sync pulse relative to its beginning. This 4-bit value should be set to the least significant 4 bits of the result of adding the length of the vertical sync pulse in terms of the number of scan lines that occur within the length of the vertical sync pulse to the value that specifies the beginning of the vertical sync pulse (see the description of the Vertical Sync Start Register for more details).

CR12 - Vertical Display Enable End Register

Address: 3B5h/3D5h (index=12h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Display Enable End Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the number of the last scan line within the active display area. In standard VGA modes, this value is described in 10 bits with bits [6,1] of the Overflow Register (CR07) supplying the two most significant bits. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area,minus 1.

CR13 - Offset Register

Address: 3B5h/3D5h (index=13h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Offset Bits [7:0].** This register provides either all 8 bits of an 8-bit value that specifies the number of words or DWords of frame buffer memory occupied by each horizontal row of characters. Whether this value is interpreted as the number of words or DWords is determined by the settings of the bits in the Clocking Mode Register (SR01). In standard VGA modes, the offset is described with an 8-bit value, all the bits of which are provided by this register. This 8-bit value should be programmed to be equal to either the number of words or DWords (depending on the setting of the bits in the Clocking Mode Register, SR01) of frame buffer memory that is occupied by each horizontal row of characters.

CR14 - Underline Location Register

Address: 3B5h/3D5h (index=14h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved. Read as 0.</td></tr>
<tr><td>6</td><td>DWord Mode.
0 = Frame buffer addresses are interpreted by the frame buffer address decoder as being either byte addresses or word addresses, depending on the setting of bit 6 of the CRT Mode Control Register (CR17).
1 = Frame buffer addresses are interpreted by the frame buffer address decoder as being DWord addresses, regardless of the setting of bit 6 of the CRT Mode Control Register (CR17).
This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to select how frame buffer addresses from the CPU are interpreted by the frame buffer address decoder as shown below:
<table>
<tr><th>CR14[6]</th><th>CR17[6]</th><th>Addressing Mode</th></tr>
<tr><td>0</td><td>0</td><td>Word Mode</td></tr>
<tr><td>0</td><td>1</td><td>Byte Mode</td></tr>
<tr><td>1</td><td>0</td><td>DWord Mode</td></tr>
<tr><td>1</td><td>1</td><td>DWord Mode</td></tr>
</table></td></tr>
<tr><td>5</td><td>Count By 4.
0 = The memory address counter is incremented either every character clock or every other character clock, depending upon the setting of bit 3 of the CRT Mode Control Register.
1 = The memory address counter is incremented either every 4 character clocks or every 2 character clocks, depending upon the setting of bit 3 of the CRT Mode Control Register. . This is used in mode x13 to allow for using all four planes.
This bit is used in conjunction with bit 3 of the CRT Mode Control Register (CR17) to select the number of character clocks are required to cause the memory address counter to be incremented as shown, below:
<table>
<tr><th>CR14[5]</th><th>CR17[3]</th><th>Addressing Incrementing Interval</th></tr>
<tr><td>0</td><td>0</td><td>every character clock</td></tr>
<tr><td>0</td><td>1</td><td>every 2 character clocks</td></tr>
<tr><td>1</td><td>0</td><td>every 4 character clocks</td></tr>
<tr><td>1</td><td>1</td><td>every 2 character clocks</td></tr>
</table></td></tr>
<tr><td>4:0</td><td>Underline Location. This field specifies which horizontal line of pixels in a character box is to be used to display a character underline in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the horizontal line on which the character underline mark is to be shown.</td></tr>
</table>

CR15 - Vertical Blanking Start Register

Address: 3B5h/3D5h (index=15h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Blanking Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area of the screen. In standard VGA modes, the vertical blanking start is specified with a 10-bit value. The most and second-most significant bits of this value are supplied by bit 5 of the Maximum Scan Line Register (CR09) and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal the number of scan lines from the beginning of the active display area to the beginning of the vertical blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which vertical blanking begins.

CR16 - Vertical Blanking End Register

Address: 3B5h/3D5h (index=16h)

Default: Undefined

Attributes: Read/Write

This register provides a 8-bit value that specifies the end of the vertical blanking period relative to its beginning.

Bit	Description
7:0	**Vertical Blanking End Bits [7:0].** This 8-bit value should be set equal to the least significant 8 bits of the result of adding the length of the vertical blanking period in terms of the number of scan lines that occur within the length of the vertical blanking period to the value that specifies the beginning of the vertical blanking period (see the description of the Vertical Blanking Start Register for details).

CR17 - CRT Mode Control

Address: 3B5h/3D5h (index=17h)

Default: 0UU0 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**CRT Controller Reset.** This bit has no effect except in native VGA modes (non-centered). 0 = Forces horizontal and vertical sync signals to be inactive. No other registers or outputs are affected. 1 = Permits normal operation.

Bit	Description
6	**Word Mode or Byte Mode.** 0 = The memory address counter's output bits are shifted by 1 bit position before being passed on to the frame buffer address decoder such that they are made into word-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. 1 = The memory address counter's output bits remain unshifted before being passed on to the frame buffer address decoder such that they remain byte-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to control how frame buffer addresses from the memory address counter are interpreted by the frame buffer address decoder as shown below: CR14[6] CR17[6] **Address Mode** 0 0 Word Mode - Addresses from the memory address counter are shifted once to become word-aligned 0 1 Byte Mode - Addresses from the memory address counter are not shifted 1 0 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned 1 1 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned
5	**Address Wrap.** This bit is only effective when word mode is made active by setting bit 6 in both the Underline Location Register and this register to 0. 0 = Wrap frame buffer address at 16 KB. This is used in CGA-compatible modes. 1 = No wrapping of frame buffer addresses.
4	**Reserved.** Read as 0.
3	**Count By 2.** This bit is used in conjunction with bit 5 of the Underline Location Register (CR14) to select the number of character clocks are required to cause the memory address counter to be incremented. 0 = The memory address counter is incremented either every character clock or every 4 character clocks, depending upon the setting of bit 5 of the Underline Location Register. 1 = The memory address counter is incremented either every other clock. CR14[5] CR17[3] Address Incrementing interval 0 0 every character clock 0 1 every 2 character clocks

Bit	Description		
	1	0	every 4 character clocks
	1	1	every 2 character clocks
2	**Horizontal Retrace Select.** This bit provides a way of effectively doubling the vertical resolution by allowing the vertical timing counter to be clocked by the horizontal retrace clock divided by 2 (usually, it would be undivided). 0 = The vertical timing counter is clocked by the horizontal retrace clock. 1 = The vertical timing counter is clocked by the horizontal retrace clock divided by 2.		
1	**Select Row Scan Counter.** 0 = A substitution takes place, where bit 14 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 1 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.		
0	**Compatibility Mode Support.** 0 = A substitution takes place, where bit 13 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 0 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.		

The following tables show the possible ways in which the address bits from the memory address counter can be shifted and/or reorganized before being presented to the frame buffer address decoder. First, the address bits generated by the memory address counter are reorganized, if need be, to accommodate byte, word or DWord modes. The resulting reorganized outputs (MAOut15-MAOut0) from the memory address counter may also be further manipulated with the substitution of bits from the row scan counter (RSOut1 and RSOut0) before finally being presented to the input bits of the frame buffer address decoder (FBIn15-FBIn0).

Memory Address Counter Address Bits [15:0]

	Byte Mode CR14 bit 6=0 CR17 bit 6=1 CR17 bit 5=X	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=1	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=0	DWord Mode CR14 bit 6=1 CR17 bit 6=X CR17 bit 5=X
MAOut0	0	15	13	12
MAOut1	1	0	0	13
MAOut2	2	1	1	0
MAOut3	3	2	2	1
MAOut4	4	3	3	2
MAOut5	5	4	4	3
MAOut6	6	5	5	4
MAOut7	7	6	6	5
MAOut8	8	7	7	6
MAOut9	9	8	8	7
MAOut10	10	9	9	8
MAOut11	11	10	10	9
MAOut12	12	11	11	10
MAOut13	13	12	12	11
MAOut14	14	13	13	12
MAOut15	15	14	14	13

X = Don't Care

Frame Buffer Address Decoder

	CR17 bit 1=1	CR17 bit 1=1	CR17 bit 1=0	CR17 bit 1=0
	CR17 bit 0=1	CR17 bit 0=0	CR17 bit 0=1	CR17 bit 0=0
FBIn0	MAOut0	MAOut0	MAOut0	MAOut0
FBIn1	MAOut1	MAOut1	MAOut1	MAOut1
FBIn2	MAOut2	MAOut2	MAOut2	MAOut2
FBIn3	MAOut3	MAOut3	MAOut3	MAOut3
FBIn4	MAOut4	MAOut4	MAOut4	MAOut4
FBIn5	MAOut5	MAOut5	MAOut5	MAOut5
FBIn6	MAOut6	MAOut6	MAOut6	MAOut6

	CR17 bit 1=1 CR17 bit 0=1	CR17 bit 1=1 CR17 bit 0=0	CR17 bit 1=0 CR17 bit 0=1	CR17 bit 1=0 CR17 bit 0=0
FBIn7	MAOut7	MAOut7	MAOut7	MAOut7
FBIn8	MAOut8	MAOut8	MAOut8	MAOut8
FBIn9	MAOut9	MAOut9	MAOut9	MAOut9
FBIn10	MAOut10	MAOut10	MAOut10	MAOut10
FBIn11	MAOut11	MAOut11	MAOut11	MAOut11
FBIn12	MAOut12	MAOut12	MAOut12	MAOut12
FBIn13	MAOut13	MAOut13	RSOut0	RSOut0
FBIn14	MAOut14	RSOut1	MAOut14	RSOut1
FBIn15	MAOut15	MAOut15	MAOut15	MAOut15

CR18 - Line Compare Register

Address: 3B5h/3D5h (index=18h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Line Compare Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bit 4 of the Overflow Register (CR07) supplies the second most significant bit. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. (This register is only used in split screening modes, and this is not a problem because split screening is not actually used for extended modes. As a result, there is no benefit to extending the existing overflow bits for higher resolutions.) When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

CR22 - Memory Read Latch Data Register

Address: 3B5h/3D5h (index=22h)

Default: 00h

Attributes: Read Only

Bit	Description
7:0	**Memory Read Latch Data.** This field provides the value currently stored in 1 of the four memory read latches. Bits 1 and 0 of the Read Map Select Register (GR04) select which of the four memory read latches may be read via this register.

CR24 - Toggle State of Attribute Controller Register

Address: 3B5h/3D5h (index=24h)

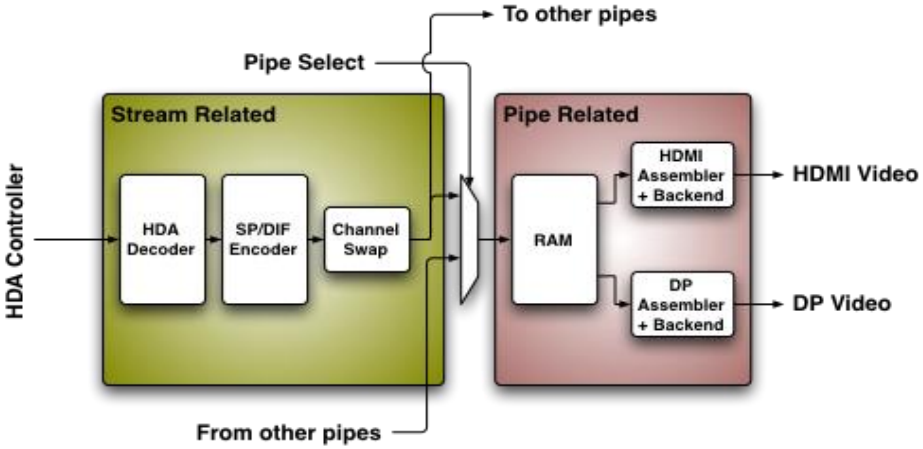
Default: 00h

Attributes: Read Only

Bit	Description
7	**Toggle Status.** Indicates where the last write to attribute register was to: 0 = index port 1 = data port
6:0	**Reserved.** Read as 0.

Display Audio Codec Verbs

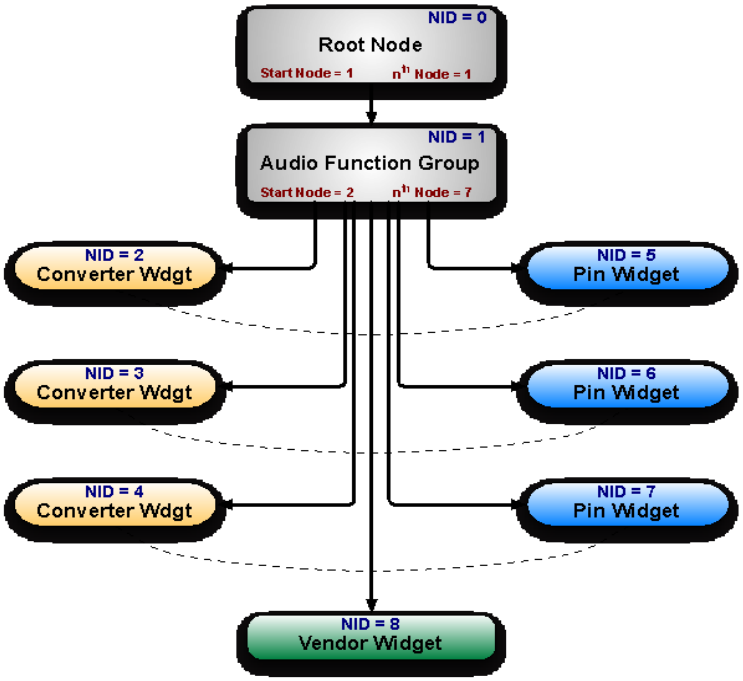
Block Diagram



Codec Node Hierarchy

The diagram below shows the hierarchy of the internal codec. The codec is presented as a single codec with multiple endpoints. By operating as a single codec, only one driver needs to be loaded on the system.

Inside the codec are three "converter widgets" and three "pin widgets", responsible for taking data from HD Audio DMA engines and placing into an HDMI/DP stream. Each pin widget has a 1-1 connection to a converter widget (as indicated by the dotted lines in the diagram).



Programming

Programming of the codec is performed by "verbs" as described in the HD Audio specification. These verbs travel over the internal HD Audio link at a rate of 1 verb per frame. A verb can either come from the CORB, with responses using the RIRB, or using an immediate command and response mechanism (ICR). Device 2 contains its own copy of an ICR mechanism as a backdoor into the audio codec.

Port To Pin Node Mapping

Port To Pin Node Mapping

The table below provides the mapping between the external port names (such as DDIA or USBC1) and the generic names (such as Port1, Port2, ... Port9) used in the 781h Verb and Node ID Description pages.

The external port names can vary per-project and are officially documented in the Overview pages under **"North Display Engine Registers"**.

The external port names are also described in the valid-values list of the **TRANS_DDI_FUNC_CTL** register's "DDI Select" bit field.

Mapping	
Pin Node Name	**External Port Name**
Port1	DDIA
Port2	DDIB
Port3	DDIC
Port4	USBC1
Port5	USBC2
Port6	USBC3
Port7	USBC4
Port8	DDID
Port9	DDIE

Verb Support

Verb ID			Node ID							
Set	Get	Verb Name/Description	01h	02h	03h	04h	05h	06h	07h	08h
2h	Ah	Stream Descriptor Format		Y	Y	Y				
3h	Bh	Set Amplifier Mute					Y	Y	Y	
-	F00h	Get Parameters	Y	Y	Y	Y	Y	Y	Y	Y
701h	F01h	Connection Select Control					Y	Y	Y	
-	F02h	Connection List Entry					Y	Y	Y	
705h	F05h	Power State		Y	Y	Y	Y	Y	Y	
706h	F06h	Channel and Stream ID		Y	Y	Y				
707h	F07h	Pin Widget Control					Y	Y	Y	
708h	F08h	Unsolicited Response Enable					Y	Y	Y	Y
-	F09h	Pin Sense					Y	Y	Y	
-	F0Dh	Digital Converter		Y	Y	Y				
70Dh	-	Digital Converter 1		Y	Y	Y				
70Eh	-	Digital Converter 2		Y	Y	Y				
-	F1Ch	Configuration Default					Y	Y	Y	
71Ch	-	Configuration Default Byte 0					Y	Y	Y	
71Dh	-	Configuration Default Byte 1					Y	Y	Y	
71Eh	-	Configuration Default Byte 2					Y	Y	Y	
71Fh	-	Configuration Default Byte 3					Y	Y	Y	
-	F20h	Subsystem ID	Y							
-	F21h	Subsystem ID	Y							
-	F22h	Subsystem ID	Y							
-	F23h	Subsystem ID	Y							
720h	-	Subsystem ID[7: 0]	Y							
721h	-	Subsystem ID[15: 8]	Y							
722h	-	Subsystem ID[23:16]	Y							
723h	-	Subsystem ID[31:24]	Y							
72Dh	F2Dh	Converter Channel Count		Y	Y	Y				
-	F2Eh	HDMI/DP Info Size					Y	Y	Y	
730h	F30h	HDMI Info Index					Y	Y	Y	
731h	F31h	HDMI Info Data					Y	Y	Y	
732h	F32h	HDMI Info Transmit Control					Y	Y	Y	
734h	F34h	Converter Channel Map					Y	Y	Y	
735h	F35h	Device Select					Y	Y	Y	

Verb ID		Verb Name/Description	Node ID							
-	F36h	Display Device List Entry					Y	Y	Y	
73Ch	73Ch	DisplayPort Stream ID					Y	Y	Y	
73Eh	-	Digital Converter 3		Y	Y	Y				
73Fh	-	Digital Converter 4		Y	Y	Y				
-	F80h	HDMI / DP Status								Y
781h	F81h	HDMI Vendor Verb								Y
782h	-	GTC Capture Trigger								Y
-	F83h	Captured Wall Clock Value								Y
-	F84h	Captured GTC Value								Y
-	F85h	Get GTC Offset Value								Y
785h	-	Set GTC Offset Value[7: 0]								Y
786h	-	Set GTC Offset Value[15: 8]								Y
787h	-	Set GTC Offset Value[23:16]								Y
788h	-	Set GTC Offset Value[31:24]								Y
789h	F89h	Converter Channel Count								Y

Parameter Support

Param ID	Parameter Name	Node ID								
		00h	01h	02h	03h	04h	05h	06h	07h	08h
00h	Vendor ID	Y								
02h	Revision ID	Y								
04h	Subordinate Node Count	Y	Y							
05h	Function Group Type		Y							
08h	Audio Function Group Capabilities									
09h	Audio Widget Capabilities			Y	Y	Y	Y	Y	Y	Y
0Ah	Sample Size, Rate CAPs			Y	Y	Y				
0Bh	Stream Formats			Y	Y	Y				
0Ch	Pin Capabilities						Y	Y	Y	
0Dh	Input Amp Capabilities									
0Eh	Connection List Length						Y	Y	Y	
0Fh	Supported Power States		Y							
10h	Processing Capabilities									
11h	GPIO Count									
12h	Output Amp Capabilities						Y	Y	Y	
13h	Volume Knob Capabilities									
15h	Device List Length						Y	Y	Y	

Node ID Descriptions

Below is the description of the valid settings of the Vendor Verb 781h at node ID 02h for enabling the features of the Display Audio Codec. The bits 7:4 are not applicable when bit 0 is set to 1.

Please refer to the "Port To Pin Node Mapping" page to determine the external port name associated with each Port1, Port2, ... Port9 used below.

Single Converter/Pin Mode:

When verb 781h Bit 0 is set to 0 then mapping as described below.

- Only Converter1 is exposed.
- Only one pin node is exposed, and the Port Select bit field in bits 7:4 can be used to select which pin node is used.

Vendor Verb 781h Bits					Node ID														
Port Select [7:4]	Reserved [3:2]	Enable DP1.2 [1]	Enable all pins and all Converters [0]	Description of the control bits	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
0000	0	0	0	Single Pin is exposed with Port 1 on Node ID 4	Root	Function	Vendor	Converter1	Port1	X	X	X	X	X	X	X	X	X	X	X
0001	0	0	0	Single Pin is exposed with Port 2 on Node ID 4	Root	Function	Vendor	Converter1	Port2	X	X	X	X	X	X	X	X	X	X	X
0010	0	0	0	Single Pin is exposed with Port 3 on Node ID 4	Root	Function	Vendor	Converter1	Port3	X	X	X	X	X	X	X	X	X	X	X
0011	0	0	0	Single Pin is exposed with Port 4 on Node ID 4	Root	Function	Vendor	Converter1	Port4	X	X	X	X	X	X	X	X	X	X	X
0100	0	0	0	Single Pin is exposed with Port 5 on Node ID 4	Root	Function	Vendor	Converter1	Port5	X	X	X	X	X	X	X	X	X	X	X
0101	0	0	0	Single Pin is exposed with Port 6 on Node ID 4	Root	Function	Vendor	Converter1	Port6	X	X	X	X	X	X	X	X	X	X	X
0110	0	0	0	Single Pin is exposed with Port 7 on Node ID 4	Root	Function	Vendor	Converter1	Port7	X	X	X	X	X	X	X	X	X	X	X
0111	0	0	0	Single Pin is exposed with Port 8 on Node ID 4	Root	Function	Vendor	Converter1	Port8	X	X	X	X	X	X	X	X	X	X	X
1000	0	0	0	Single Pin is exposed with Port 9 on Node ID 4	Root	Function	Vendor	Converter1	Port9	X	X	X	X	X	X	X	X	X	X	X

All Converters and Pins Mode:

When verb 781h Bit 0 is set to 1 then mapping as described below.

- Any converter can be mapped to any pin widget.
- The bits 7:4 are not applicable when bit 0 is set to 1.
- The value of bit 1 has no effect on the Node ID assignment below.

Node ID	Node Description
00h	Root
01h	Function
02h	Vendor
03h	Converter1
04h	Port1
05h	Converter2
06h	Port2
07h	Converter3
08h	Port3
09h	Converter4
0Ah	Port4
0Bh	Port5
0Ch	Port6
0Dh	Port7
0Eh	Port8
0Fh	Port9

Node ID 00h Root Node Verbs

The root node only contains a single verb - the "Get Parameters" verb at F00h.

F00h - Get Parameters

Parameters

Parameter	Symbol	Register Name
00h	PARAM_VID	Vendor ID
02h	PARAM_RID	Revision ID
04h	PARAM_SNC	Subordinate Node Count

Parameter 00h: VID - Vendor ID

Bit	Reset	Description
31:16	8086h	**Vendor ID (VID):** Indicates the 16-bit Vendor ID values used to identify the codec to the PnP subsystem.
15:00	<value in table below>	**Device ID (DID):** Indicates the 16-bit Device ID values used to identify the codec to the PnP subsystem.

Device ID
2819h

Parameter 02h: RID - Revision ID

Bit	Reset	Description
31:24	*0*	*Reserved*
23:20	1h	**Major Revision (MJR):** Indicates the major revision number (left of the decimal) of the High Definition Audio Specification to which the codec is fully compliant.
19:16	0h	**Minor Revision (MNR):** Indicates the minor revision number (right of the decimal) or "dot number" of the High Definition Audio Specification to which the codec is fully compliant.
15:08	00h	**Revision ID (RID):** Indicates the vendor's revision number for this given Device ID.
07:00	00h	**Stepping ID (SID):** Indicates optional vendor stepping number within the revision.

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	0h	**Starting Node Number (SNN):** Indicates the first sub-node's ID is 01h.
15:08	00h	Reserved
07:00	01h	**Total Number of Nodes (TNN):** Indicates one sub-node

F37h GET CCF - Get Current Clock Frequency

Bits	Default	Description
31:6	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz. Reserved for Display Codec
04	1	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz. Reserved for Display Codec
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz. Reserved for Display Codec
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz. Reserved for Display Codec

Node ID 01h Audio Function Group Verbs

Set Verb	Get Verb	Symbol	Name
-	F00h	GET PARAM	Get Parameters
705h	F05h	SET_PS / GET_PS	Set Power State
-	F20h	GET_SSID	Get Subsystem ID
720h	F20h	SET_SSID0	Set/Get Subsystem ID

Set Verb	Get Verb	Symbol	Name
721h	F21h	SET_SSID1	Set/Get Subsystem ID
722h	F22h	SET_SSID2	Set//Get Subsystem ID
723h	F23h	SET_SSID3	Set/Get Subsystem ID
724h	F24h	SET CCF	Set/Get Current Clock Frequency

F00h Get Parameters

Parameter	Symbol	Register Name
04h	PARAM_SNC	Subordinate Node Count
05h	PARAM_FGT	Function Group Type
08h	PARAM_FGC	Function Group Capabilities
0Fh	PARAM SPS	Supported Power States

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	03h	**Start Node Number (SNN):** Indicates the start node number of widget or functional nodes in the Functional Group.
15:08	*0*	*Reserved*
07:00	07h	**Total Number of Nodes (TNN):** Indicates 7 widgets in the Functional Group. (HDMI/DP converters (3) + HDMI/DP pins (3) + Vendor Defined Widget (1)). If Wigig is enabled in the vendor defined widget then the total number of nodes should be reported as 08h. Default for D11 is 9 [HDMI/DP converters (4) + HDMI/DP pins (5)]. If wireless widgets are enabled then total is 10 or 11. Default for D11p5 is 13 [HDMI/DP converters (4) + HDMI/DP pins (9)]. If wireless widgets are enabled then total is 14 or 15.

Parameter 05h: PARAM_FGT - Function Group Type

Bit	Reset	Description
31:09	*0*	*Reserved*
08	0	**Unsolicited Capable (UC):** Not capable of generating an unsolicited response.
07:00	01h	**Node Type (NT):** Indicates Audio Function Group.

Parameter 08h: PARAM_FGC - Function Group Capability

Bit	Reset	Description
31:04	*0*	*Reserved*
03:00	00h	**Output Delay (OD)**Output Delay.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30	1	**Clock Stop (CS):** Indicates support for D3 when clock is stopped.
29:04	*0*	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

Parameter 16h: PARAM_A2CAP - Azalia 2 Capabilities

Bits	Reset	Description
31:17	*0*	*Reserved*
16	0	**Independent Codec Clock (ICC):** When set, this indicates that the codec generates its own clock, which may drift from the link clock. When cleared, the codec's clock is locked to the link clock.
15:06	*0*	*Reserved*
05	*0*	*Reserved for 192 MHz support.*
04	1	**96MHz Supported (S96):** Indicates 96 MHz clock is supported.
03	1	**48MHz Supported (S48):** Indicates 48 MHz clock is supported. This bit must always be set.
02	0	**24MHz Supported (S24):** Indicates 24 MHz clock is supported. Reserved for Display Codec
01	0	**12MHz Supported (S12):** Indicates 12 MHz clock is supported. Reserved for Display Codec
00	0	**6MHz Supported (S6)**Indicates 6 MHz clock is supported. Reserved for Display Codec

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Default values will not be changed. This bit will report 0 in all cases.
09	1	**Clock Stop OK (CSOK):** Clock stopping in D3 is OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

F20h GET SSID - Get Subsystem ID0

Bits	Reset	Description
31:00	80860101h	**Subsystem ID (SSID):** Reports the sub-system ID set via SET_SSIDx verbs.

720h SET SSID0 - Set Subsystem ID0

Bits	Description
07:00	Subsystem ID Bits [7:0]

721h SET SSID1 - Set Subsystem ID1

Bits	Description
07:00	Subsystem ID Bits [15:8]

722h SET SSID2 - Set Subsystem ID2

Bits	Description
07:00	Subsystem ID Bits [23:16]

723h SET SSID3 - Set Subsystem ID3

Bits	Description
07:00	Subsystem ID Bits [31:24]

724h SET CCF - Set Current Clock Frequency

Bits	Description
07:06	Reserved
05	**Set Clock 192 MHz (S192)**: Set clock to 192 MHz.
04	**Set Clock 96 MHz (S96)**: Set clock to 96 MHz
03	**Set Clock 48 MHz (S48)**: Set clock to 48 MHz
02	**Set Clock 24 MHz (S24)**: Set clock to 24 MHz
01	**Set Clock 12 MHz (S12)**: Set clock to 12 MHz
00	**Set Clock 6 MHz (S6)**: Set clock to 6 MHz

F24h GET CCF - Get Current Clock Frequency

Bits	Bits	Description
31:06	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz.
04	0	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz.
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz.
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz.

7FFh SET Function Group Reset

Bits	Reset	Description
07:00	00h	The Function Reset command causes the functional unit, and all widgets associated with the functional unit, to return to their power-on reset values. Note that some controls such as the Configuration Default controls should not be reset with this command. It is also possible that certain other controls, such as Caller-ID, should not be reset. This command does not affect the Link interface logic, which must be reset with the link RST# signal. Therefore, a codec must not initiate a Status Change request on the link.

Audio Output Convertor Widget Verbs

Verb	Symbol	Verb Name
2h	SET_SDF	Set Stream Descriptor Format
Ah	GET_SDF	Get Stream Descriptor Format
F00h	GET_PARAM	Get Parameters
705h	SET_PS	Set Power State
F05h	GET_PS	Get Power State
706h	SET_CSID	Set Channel and Stream ID
F06h	GET_CSID	Get Channel and Stream ID
F0Dh	SET_DC1	Get Digital Converter
70Dh	SET_DC1	Set Digital Converter 1
70Eh	SET_DC2	Set Digital Converter 2
73Eh	SET_DC3	Set Digital Converter 3
73Fh	SET_DC4	Set Digital Converter 4
72Dh	SET_CCC	Set Converter Channel Count
F2Dh	GET_CCC	Get Converter Channel Count

2hAh SETGET_SDF - SetGET Stream Descriptor Format

Bits	Reset	Description
31:15	0	Reserved
14	0	**Sample Base Rate (SBR):**
13:11	000	**Sample Base Rate Multiplier (SBRM):**
10:08	000	**Sample Base Rate Divisor (SBRD):**
07	0	Reserved
06:04	011	**Bits / Sample (BPS):** • 001b: Data is packed in memory in 16 bit containers on 16 bit boundaries • 010b: Data is packed in memory in 20 bit containers on 32 bit boundaries • 011b: Data is packed in memory in 24 bit containers on 24 bit boundaries • 100b: Data is packed in memory in 32 bit containers on 32 bit boundaries All other bit combinations reserved

Bits	Reset	Description
03:00	1h	**# Channels in Stream (NCS):** 2 channels in each frame

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ah	PARAM_PSB	Parameter Sizes and Bit Rates
0Bh	PARAM_SF	Stream Formats
0Fh	PARAM_SPS	Power Supported States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	0h	**Widget Type (TYPE):** Indicates this is an audio output widget
19:16		Sample Delay in Widget (DELAY):
15:13	011	**Channel Count Extension (CCE):** These three bits, combined with STRO, indicate that there are 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	0	**Connection List (CL):** Indicates no connection list
07	0	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	1	**Format Override (FO):** Indicates support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier support.
02	0	**Out Amp Present (OAP):** Indicates no output amplifier present.

Bits	Reset	Description
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ah: PSB - PCM Sizes and Bit Rates

Bits	Reset	Description
31:21	0	*Reserved*
20	1	**32-bit Support (B32):** Indicates 32-bit samples supported
19	1	**24-bit Support (B24):** Indicates 24-bit samples supported
18	0	**20-bit Support (B20):** Indicates 20-bit samples supported
17	1	**16-bit Support (B16):** Indicates 16-bit samples supported
16	0	**8-bit Support (B8):** Indicates 8-bit samples not supported
15:12	0	*Reserved*
11	0	**384 kHz Support (R12):** Indicates 384 kHz not supported
10	1	**192 kHz Support (R11):** Indicates 192 kHz supported
09	1	**176.4 kHz Support (R10):** Indicates 176.4 kHz supported
08	1	**96 kHz Support (R9):** Indicates 96 kHz supported
07	1	**88.2 kHz Support (R8):** Indicates 88.2 kHz supported
06	1	**48 kHz Support (R7):** Indicates 48 kHz supported
05	1	**44.1 kHz Support (R6):** Indicates 44.1 kHz supported
04	1	**32 kHz Support (R5):** Indicates 32 kHz supported
03	0	**22.05 kHz Support (R4):** Indicates 22.05 kHz not supported
02	0	**16 kHz Support (R3):** Indicates 16 kHz not supported

Bits	Reset	Description
01	0	**11.025 kHz Support (R2):** Indicates 11.025 kHz not supported
00	0	**8 kHz Support (R1):** Indicates 8 kHz not supported

Parameter 0Bh: SF - Stream Formats

Bits	Reset	Description
31:03	0	*Reserved*
02	1	**AC3 Support (AC3):** Indicates AC3 stream format is supported
01	0	**Float32 Support (F32):** Indicates float32 stream format not supported
00	1	**PCM Support (PCM):** Indicates PCM format is supported.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Default values will not be changed. This bit will report 0 in all cases.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

706hF06h GETSET_CSID - GetSet Channel and Stream ID

Bits	Reset	Description
07:04	0h	**Stream ID (SID):** Link stream used by the converter for data output.
03:00	0h	**Lowest Channel Number (LCN):** Lowest channel used by the converter.

Digital Converter Verbs

F0Dh: GET_DC - Get Digital Converter

Bits	Reset	Description
31:24	0	Reserved
23	0	**Keep Alive (KA):** See SET_DC3.KA
22:20	0	Reserved
19:16	0h	**IEC Coding Type (ICT):** See SET_DC3.ICT
15	0	Reserved
14:08	00h	**Category Code (CC):** See SET_DC1.CC
07	0	**Level (LVL):** See SET_DC1.LVL
06	0	**Professional (PRO):** See SET_DC1.PRO

Bits	Reset	Description
05	0	**Audio is not PCM (AUDIO):** See SET_DC1.AUDIO
04	0	**Copyright (COPY):** See SET_DC1.COPY
03	0	**Pre-emphasis (PRE):** See SET_DC1.PRE
02	0	**Validity Configuration (VCFG):** See SET_DC1.VCFG
01	0	**Validity (V):** See SET_DC1.V
00	1	**Digital Enable (DIGEN):** See SET_DC1.DIGEN

70Dh: SET_DC1 - Set Digital Converter 1

Bits	Description
07	**Level (LVL):** S/PDIF IEC Generation Level.
06	**Professional (PRO):** When set, indicates professional use of channel.
05	**Audio is not PCM (AUDIO):** When set, data is non-PCM format.
04	**Copyright (COPY):** When set, copyright asserted.
03	**Pre-emphasis (PRE):** When set, enables filter pre-emphasis.
02	**Validity Configuration (VCFG):** Determines S/PDIF transmitter behavior when data is not being transmitted.
01	**Validity (V):** Affects the validity flag transmitted in each sub-frame, and enables S/PDIF transmitter to maintain connection during error or mute conditions.
00	**Digital Enable (DIGEN):** When set, enables digital content

70Eh: Digital Converter 2

Bits	Description
07	Reserved
06:00	**Category Code (CC):** S/PDIF IEC Category Code.

73Eh: Digital Converter 3

Bits	Description
07	Keep Alive
06:04	Reserved
03:00	IEC Coding Type

73Fh: Digital Converter 4

Bits	Description
07:00	Reserved

72DhF2Dh GETSET_CCC - GetSet Converter Channel Count

Bits	Reset	Description
07:04	0	Reserved
03:00	0000	Converter Channel Count 1 (0th order)

Pin Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
3h	-	SET_AM	Set Amplifier Mute
-	Bh	GET_AM	Get Amplifier Mute
-	F00h	-	Get Parameters
701h	F01h	SET_CSC / GET_CSC	Set/Get Connection Select Control
-	F02h	-	Get Connection List Entry
705h	F05h	SET_PS / GET_PS	Set/Get Power State
707h	F07h	SET_PWC / GET_PWC	Set/Get Pin Widget Control
708h	F08h	SET_UE / GET_UE	Set/Get Unsolicited Response Enable
-	F09h	-	Get Pin Sense
71Ch	-	SET_CD0	Set Configuration Default Byte 0
71Dh	-	SET_CD1	Set Configuration Default Byte 1
71Eh	-	SET_CD2	Set Configuration Default Byte 2
71Fh	-	SET_CD3	Set Configuration Default Byte 3
-	F1Ch	GET_CD	Get Configuration Default
-	F2Eh	GET_HDIS	Get HDMI/DP Info Size
730h	F30h	SET_HII / GET_HII	Set/Get HDMI Info Index
731h	F31h	SET_HID / GET_HID	Set/Get HDMI Info Data
732h	F32h	SET_HITC / GET_HITC	Set/Get HDMI Info Transmit Control

Set Verb	Get Verb	Symbol	Verb Name
733h	F33h	SET_PC / GET_PC	Set/Get Protection Control
734h	F34h	SET_CCM / GET_CCM	Set/Get Converter Channel Map
735h	F35h	SET_DS / GET_DS	Set/Get Device Select
-	F36h	GET_DDLE	Get Display Device List Entry
73Ch	F3Ch	SET_DPID / GET_DPID	Set/Get DisplayPort Stream ID

3h SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

B8h GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	0	Reserved
07	1	**Mute (MUTE):** When set, amp is muted.
06:00	0	Reserved

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
15h	PARAM_DLL	Device List Length

Parameter	Symbol	Register Name
0Fh	PARAM_SPS	Supported Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	0	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio
26	0	*Reserved*
25	0	*Reserved*
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	0	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI
06:05	0	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	0	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	0	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	0	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 15h: DLL - Device List Length

Bits	Reset	Description
31:06	0	*Reserved*
05:00	00h	**Length (LEN):** Indicates no devices

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

701hF01h SETGET_CSC - SetGet Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC):**

F02h GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:08	0	Reserved
07:00	Varies	**Connection List Entry (CLE):** 02h for NodeID 05h, 03h for NodeID 06h, and 04h for NodeID 07h.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Default values will not be changed. This bit will report 0 in all cases.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707hF07h SETGET_PWC - SetGet Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT):**

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG):**

F09h GET_PS - Get Pin Sense

Determining SST / MST Mode

The audio codec will use multi-stream based indexing (MST mode) as described in the following sections if the following conditions are met, else it will use Non-MST mode.

Conditions for MST Mode
• In register **TRANS_DDI_FUNC_CTL**, the bit-field "TRANS DDI Mode Select" is set to "DP MST" **AND** • In verb GetSet iDisp Codec Vendor Verb (781h), the bit-field "Enable DP1.2 Features" (bit 1) is set to 1b. **OR** • In register**AUD_CONFIG_BE_2**, the bit-fields "DP2 Multi Stream Enable for Pipe" are set to 1b for any pipe **AND** • In verb GetSet iDisp Codec Vendor Verb (781h), the bit-field "Enable DP1.2 Features" (bit 1) is set to 1b.

Verb Parameters (Non-MST Mode)

For non-MST mode, the F09h verb response will automatically provide the status of the single pipe associated with the given pin node.

Nothing needs to be set in the data byte of the F09h Get Pin Sense verb.

Bits	Reset	Description
07:00	0	Reserved

Verb Parameters (MST Mode)

For MST mode, the F09h verb response will provide the status of the pipe selected in the Pipe Select bit-field, among the pipe(s) associated with the given pin node.

The data byte of the F09h Get Pin Sense verb should be programmed to select the desired pipe as shown below.

Bits	Reset	Description
07:06	0	Reserved
05:00	0	**Pipe Select:** Set to 000000b for pipe A, 000001b for pipe B, 000010b for pipe C, 000011b for pipe D

Verb Response (All Modes)

The verb response for the F09h Get Pin Sence verb command contains the following data regardless of which mode (Non-MST or MST) is used.

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV):**
29	0	**Inactive (INA):**
28:00	28:00	Reserved

71Ch SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	Default Association (DA):
03:00	Sequence (SEQ):

71Dh SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	Color (COL):
03:00	Miscellaneous (MISC):

71Eh SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	Default Device (DD):
03:00	Connection Type (CT):

71Fh SET_CD3 - Set Configuration Default Byte 3

Bits	Description
07:06	**Port Connectivity (PC):** External connectivity of the pin complex. • 00 = Connected to jack • 01 = No physical connection • 10 = Fixed function device (integrated speaker, mic, etc.) • 11 = Both a jack and internal connection

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>05:00</td><td>Location (LOC):
<table>
<tr><td></td><td colspan="10">Bits 3:0</td></tr>
<tr><td>Bits 5:4</td><td>0h: N/A</td><td>1h: Rear</td><td>2h: Front</td><td>3h: Left</td><td>4h: Right</td><td>5h: Top</td><td>6h: Bottom</td><td>7h: Special</td><td>8h: Special</td><td>9h: Special</td><td>Ah-Fh Reserved</td></tr>
<tr><td>00: External</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
<tr><td>01: Internal</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td></tr>
<tr><td>10: Separate Chassis</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11: Other</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
</table>
</td></tr>
</table>

F1Ch GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Eh HDMIDP Info Size

Bits	Reset	Description
31:08	0	Reset
07:00	Varies	**Size (SZ):** Indexes 0 - 3 return 1Eh, index 1000 returns 53h, others reserved.

F2Fh Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

730hF30h SETGET_HII - SetGet HDMI Info Index

Bits	Reset	Description
07:05	000	**Infoframe Packet Index (IPI):** Value / Name / Value / Name 000 Audio 011 GP3 001 GP 100 GP4 010 GP2 Others Reserved
04:00	00h	**Byte Offset Index Pointer (BOI):**

731hF31h SETGET_HID - SetGet HDMI Info Data

Bits	Reset	Description
07:00	00h	**Data (DATA):** Data at current index pointed to from SET_HII verb.

732hF32h SETGET_HITC - SetGet HDMI Info Transmit Control

Bits	Reset	Description
07:06	00	**InfoFrame Control Current Indexed Frame (IFCCIF):** • 00 = Disable Transmit • 01 = Reserved • 10 = Transmit Once • 11 = Best Effort
05:00	0	*Reserved*

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734hF34h SETGET_CCM - GetSet Converter Channel Map

Bits	Reset	Description
07:04	0h	Converter Channel (CC):
03:00	0h	Slot (SN):

735h SET_DS - Set Device Select

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000000, 000001, 000010, 000011 (based upon number of devices present)

F35h: GET_DS - Get Device Select

Bits	Description
31:12	**Reserved:** Set to 0
11:06	**SinK Device ID:** Sink Device ID in the multi stream topology of the DP hierarchy. Device attached to Pipe A will have ID of "000000", PipeB will have "000001", Pipe C will have "000010" and Pipe D will have "000011".
05:00	**Device (D):**Device Entry index currently set

F36h GET_DDLE - Get Display Device List Entry

Bits	Bits	Description
31:16	0	*Reserved*
15	0	*Reserved*
14	0	**IA of Entry 3**
13	0	**ELDV of Entry 3**
12	0	**PD of Entry 3**
11	0	*Reserved*
10	0	**IA of Entry 2**

Bits	Bits	Description
09	0	**ELDV of Entry 2**
08	0	**PD of Entry 2**
07	0	*Reserved*
06	0	**IA of Entry 1**
05	0	**ELDV of Entry 1**
04	0	**PD of Entry 1**
03	0	*Reserved*
02	0	**IA of Entry 0**
01	0	**ELDV of Entry 0**
00	0	**PD of Entry 0**

73ChF3Ch SETGET_DPID - SetGet DisplayPort Stream ID

Bits	Reset	Description
07:03	00h	**Tag (TAG):** Represents the SSID that will go in the lower 5 bits of the SSID
02:00	000	**Index (IDX):** Pointer to program multiple SSID

Intel Vendor Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
-	F00h	GET_PARAM	Get Parameters
-	F80h	GET_HDPS	Get HDMI/DP Status
781h	F81h	SET_HVV / GET_HVV	Set/Get iDisp Codec Vendor Verb
782h	-	SET_GTCT	Set GTC Trigger
-	F83h	GET_CWC	Get Captured Wall Clock
	F84h	GET_CGTC	Get Captured GTC Value
	F85h	GET_GOF	Get GTC Offset Value
785h	-	SET_GOF0	Set GTC Offset Value Byte 0
786h	-	SET_GOF1	Set GTC Offset Value Byte 1
787h	-	SET_GOF2	Set GTC Offset Value Byte 2

Set Verb	Get Verb	Symbol	Verb Name
788h	-	SET_GOF3	Set GTC Offset Value Byte 3
789h	F89h	SET_GDI / GET_GDI	Set/Get GTC Offset Device Index

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	Fh	**Widget Type (TYPE):** Indicates this is a vendor defined widget
19:00	0	Reserved

71Eh SET_GET_GFXMAILBOX - Set Get GFX MAILBOX Byte 2

71Eh: SET GFX MAILBOXM

Bits	Default	Description
07:00	00	Contents to be defined by GFX driver and audio driver

F1Eh: GET GFX MAILBOX

Bits	Default	Description
31:24	00h	Other values of 71F, verbs
23:16	00h	Contents to be defined by GFX driver and audio driver
15:00	0000h	Other values of 71D, 71C verb

728h SET CLOCK OFF - Set Clock Off Command

Bits	Description
07:00	Data is Irrelevant

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses from GFX MAIL BOX register writes are allowed.
06	Reserved
05:00	**Tag (TAG):** Tag for GFX Mail box register unsol responses.

781hF81h GETSET_VV - GetSet iDisp Codec Vendor Verb

Bits	Default	Description
07:04	0	**Port Select:** This filed is programmed to select the port to be exposed to the inbox driver in the vanilla mode. Will not reset on double function group reset. Please refer to the "Port To Pin Node Mapping" page to determine the external port name associated with each Port1, Port2, ... Port9 used below. 0000 -> Port 1 0001 -> Port 2 0010 -> Port 3 0011 -> Port 4 0100 -> Port 5 0101 -> Port 6 0110 -> Port 7 0111 -> Port 8 1000 -> Port 9
03:02	0	**Reserved**
01	0	**Enable DP1.2 Features (EDP12):** When set, DP1.2 features are enabled. Will not reset on double function group reset.
00	0	**Enable all Pins and all Converter Widget (E3P):** When set, all the pins and converter widgets are enabled and can respond to HD Audio Verbs. When cleared, only one Pin and one converter widgets are exposed. Pin selection is done programming bits 7:5. Will not reset on double function group reset.

782h SET_GTCT - Set GTC Trigger

Bits	Bits	Description
07:00	0	**Any data:** The value of this field is irrelevant. The access of the SET causes a capture to occur.

F83h GET_CGTC - Get Captured GTC Value

Bits	Bits	Description
07	0	**GTC Value:** 32-bit GTC value captured on the SET_GTCT verb

F84h GET_CWC - Get Captured Wall Clock Value

Bits	Bits	Description
31:00	0	**Wall Clock Value:** 32-bit wall clock value captured on the SET_GTCT verb

F85h GET GOF - Get GTC Offset Value

Bits	Reset	Description
31:00	0h	**Value (VAL):** Reports the GTC Offset Value.

785h SET GOF0 - Set GTC Offset Value Byte 0

Bits	Description
07:00	GTC Offset Value Bits [7:0]

786h SET GOF1 - Set GTC Offset Value Byte 1

Bits	Description
07:00	GTC Offset Value Bits [15:8]

787h SET GOF2 - Set GTC Offset Value Byte 2

Bits	Description
07:00	GTC Offset Value Bits [23:16]

788h SET GOF3 - GTC Offset Value Byte 3

Bits	Description
07:00	GTC Offset Value Bits [31:24]

789hF89h SETGET_GDI - SetGet GTC Device Index

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present) This is the pipe based. 000000 is pipeA and so on and so forth.

North Display Engine Registers

This chapter contains the register descriptions for the display engine portion of a family of graphics devices.

These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

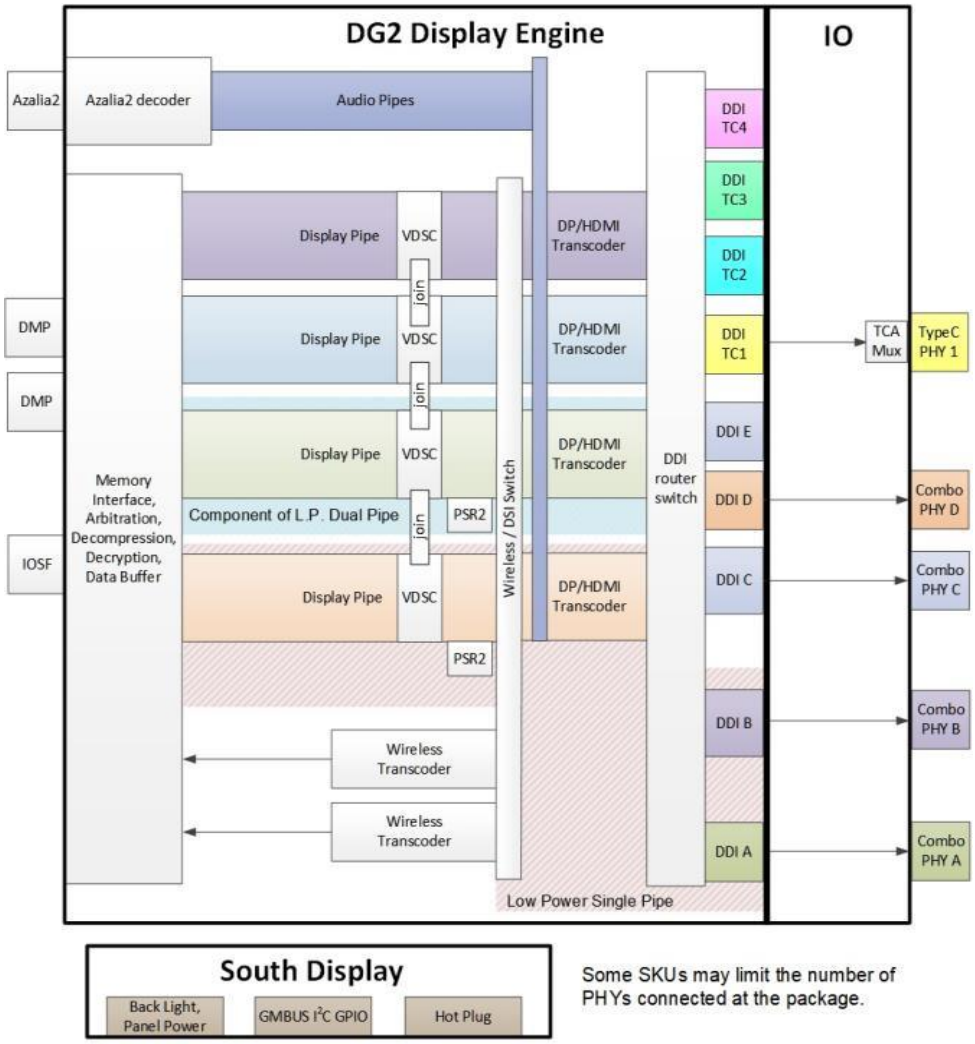
Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

Overview

Genesis

This project uses Xe_{HPD}, based on Xe_{D}.

Block Diagram



The front end of the display contains the pipes. The pipes connect to the transcoders. The transcoders, except for wireless, connect to the DDIs to drive the IO/PHY. Wireless writes back to memory.

Refer to the South Display Engine Registers section for the south display engine block diagram.

General Capabilities

Four simultaneous displays (pipes A, B, C, D)

- 5 planes and 1 cursor per pipe
- Audio streams per pipe to go to external ports
- HDR support for 3 planes per pipe
- VESA DSC compression support for all pipes
- Post-DSC joining to connect 2 adjacent pipes for resolutions that require more bandwidth than one pipe can support
- Pipe A is optimized for low power

External display connections

- 2 wireless
- 4 combo ports (DisplayPort, HDMI, or eDP)
    - 100 MHz PHY reference clock
- 1 Type-C capable port
    - v2.0 ALT mode capable, but not supported by platform
    - 38.4 MHz PHY reference clock default
    - 100.00 MHz PHY reference clock for native/legacy connectors
    - Some SKUs/platforms will not use this port
- No Thunderbolt tunneling support
- AUX channels for DisplayPorts
- Multi-stream support for DisplayPorts
- Genlock support on all combo ports
    - Mutex with eDP
    - Genlock supported across identical genlock systems

Embedded/local display connections

- 4 combo ports that can support eDP and external ports, but only 2 low power optimized ports
- AUX channel for eDP
- PSR1 and MSO (multi-segmented operation, chip on glass) for eDP

South Display

- Backlight modulation PWM, panel power sequencing, GMBUS I2C, non-typeC hot plug detection

Port Configurations

Port Type	# of ports[1]	Protocol	Maximum Speed[5]	Maximum Resolution[2]
Combo	4[3]	eDP 1.4b	HBR3 8.1 Gbps	5k 60Hz 24bpp
		DP 1.4a	HBR3 8.1 Gbps	5k 60Hz 24bpp
		HDMI 1.4	3.40 Gbps	4k 30Hz 24bpp
		HDMI 2.0b	6.00 Gbps	4k 60Hz 24bpp
		DP 2.1	UHBR 10 & 13.5 Gbps	8k 60Hz compressed, 5k 120Hz compressed
USB Type-C capable[4]	1	DP 1.4a	HBR3 8.1 Gbps	8k 60Hz compressed, 5k 120Hz compressed
		HDMI 1.4	3.40 Gbps	4k 30Hz 24bpp
		HDMI 2.0b	6.00 Gbps	4k 60Hz 24bpp
		DP 2.1	UHBR 10 & 13.5 Gbps	8k 60Hz compressed, 5k 120Hz compressed
Capture	2	N/A	N/A	Single display 4k 60Hz 30bpp Dual display 2560x1600 60Hz 30bpp

[1] Many SKUs will limit the number of ports connected at the package.

[2] Resolution restrictions

- Resolutions supported, but not a guarantee of user experience.
- Thermal constraints may apply.
- Multi-display resolution support limited.
- Subject to memory bandwidth availability. Software configuration constrained to fit available bandwidth.
- These are single cable panel (non-tiled panel) resolutions. Panels with two cables may support higher resolution by using two ports.
- Higher bits per color may be possible at lower resolutions.
- Some resolutions require compression (VDSC). VDSC input and output bits per color are limited.
- Supports other resolutions and bits per color that fit the same bandwidth and size constraints.
- Software configuration constrained to fit available bandwidth with current memory configuration.

[3] Only two power optimized ports for eDP.

[4] USB type-C PHY can support DP alternate mode, DP with legacy connector, or HDMI with legacy connector. The platform limits it to only the legacy connectors.

[5] OEM must use VBT to specify a maximum that is tolerated by the board design.

Multiple Display Resolution Support

This table gives the required resolution support. Actual results may have higher resolution, depending on exact details of the configuration, such as the number of planes enabled and pixel formats.

Number of simultaneous displays	Maximum Common Resolution
1 or 2	8k 60Hz HDR or 5k 120Hz HDR
3 or 4	5k 60Hz HDR or 4k 120Hz HDR

All subject to resolution restrictions noted under the Port Configurations table.

HDR (High Dynamic Range) increases memory bandwidth compared to standard dynamic range (SDR).

Software does not have explicit restrictions for multiple display cases, just constraints to fit available bandwidth with current memory configuration.

See the Resolution Support page for detailed restrictions and calculations.

Port Availability

DDI	Capability	IO name
A	eDP, DP, HDMI	Port A, Combo Port A
B	eDP, DP, HDMI	Port B, Combo Port B
C	eDP, DP, HDMI	Port C, Combo Port C
D	eDP, DP, HDMI	Port D, Combo Port D
E	No connection	No connection
TC 1	eDP, DP, HDMI, Alt DP	TC 1
TC 2	No connection	No connection
TC 3	No connection	No connection
TC 4	No connection	No connection
TC 5	No connection	No connection
TC 6	No connection	No connection

Some SKUs will limit which ports are connected in the die and package. Software should rely on hotplug to determine which ports are actually available.

Power Wells

Display engine functions are spread across several different power gated (PG) wells that can be shut down to optimize power for different configurations.

Hardware can dynamically control some PGs for display power states. That is enabled through the DC State programming.

Refer to Sequences for Power Wells for more details on the functions in each power well and the sequences for enabling and disabling power.

Supported power wells - main and per pipe.

Pipes

Refer to the pipe section top level page for details.

Transcoders

The display transcoders contain the timing generators, port encoders, DisplayPort transport control, Audio/Video mixers, Video Data Island Packet mixers, and Panel Self Refresh controllers.

Except for the WD (Wireless Display, Capture) transcoders, the transcoders convert pixel data to the appropriate format for the port, mix in data islands and audio, and output the data to the DDIs.

The WD transcoders convert pixel data to the appropriate format, then write it to system memory.

Transcoder A-D can support DP or HDMI, with audio.

Only transcoder A supports eDP CoG.

The transcoders, except for wireless transcoders, can make use of VDSC compression.

Audio

The Azalia2 interface provides data to the audio codec.

The audio codec connects to the Audio/Video mixers in the transcoders.

The audio codec can also write data back to system memory for wireless audio.

DDIs (Digital Display Interfaces)

The DDIs contain port logic to interface to the DDI physical (PHY) layer.

The DDIs going to the combo PHY support lane reversal where the internal lane to package lane mapping is swapped.

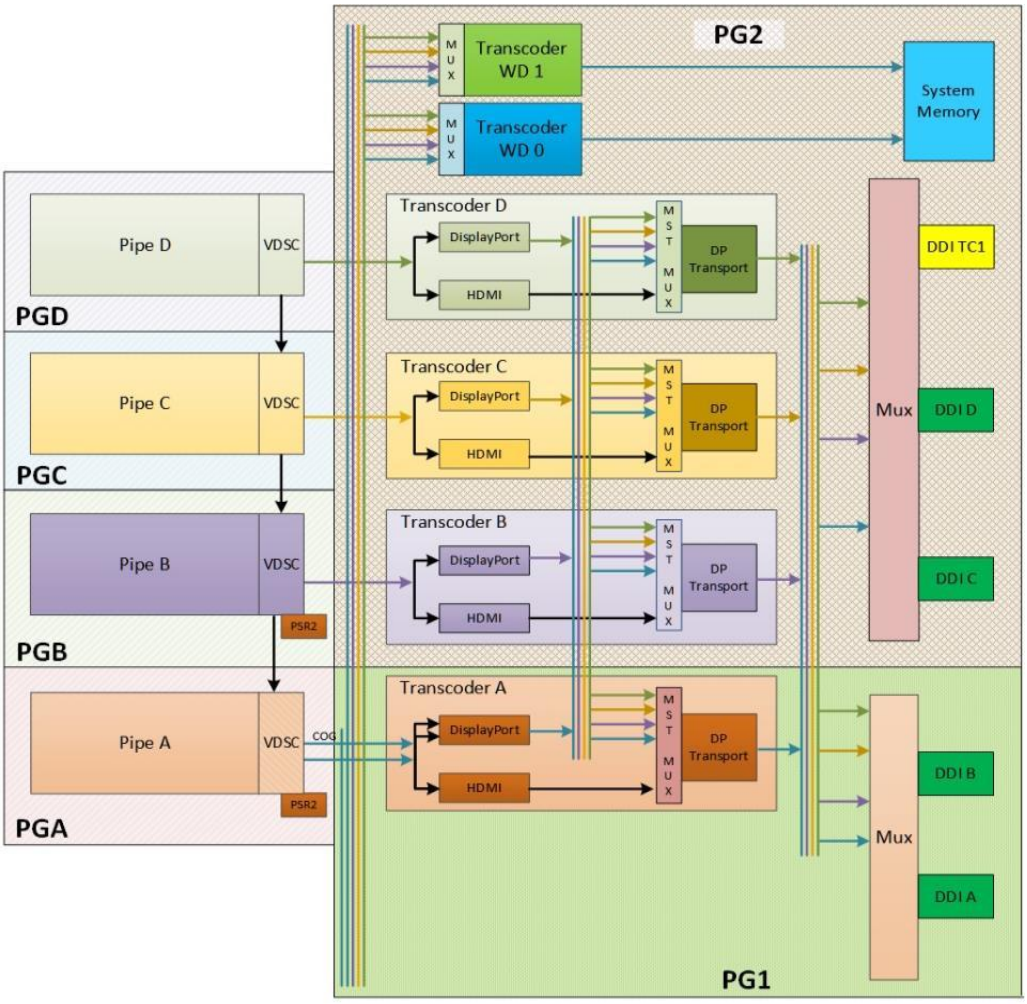
A Combo PHY capable of supporting DDI is attached to DDI A through DDI E.

DDI Lane Mapping:

DE Output	DisplayPort Non-Reversed	DisplayPort Reversed	HDMI/DVI Non-Reversed	HDMI/DVI Reversed
DDI data 3	Main Link Lane 3	Main Link Lane 0	TMDS Clock	TMDS Data2
DDI data 2	Main Link Lane 2	Main Link Lane 1	TMDS Data0	TMDS Data1
DDI data 1	Main Link Lane 1	Main Link Lane 2	TMDS Data1	TMDS Data0
DDI data 0	Main Link Lane 0	Main Link Lane 3	TMDS Data2	TMDS Clock

Note: For mapping to package pins, please refer to the Type-C pin assignments and PHY documentation.

Pipe to Transcoder to DDI Mappings



Twin modes are not supported. A pipe cannot drive more than one display.

A pipe cannot connect to more than one transcoder simultaneously.

With DisplayPort multistream it is possible to have multiple pipes/transcoders driving a single DDI. Multistream requires at transcoder to be enabled to drive the DP transport, even if the associated pipe streams are disabled.

Transcoders WD* do not support multistream.

Two pipes can drive a single display through two transcoders and two DDIs that are joined in the panel (tiled display) or by using DSC compression and joining the pipe outputs to go to a single transcoder and DDI.

Transcoders A-D are tied to the respective Pipes A-D. Each pipe output can go to either the respective transcoder or to transcoders WD*.

Pipe A VDSC outputs 2 streams to split the display for eDP Chip on Glass or MSO.

Transcoders A-D can connect to any DDI. Transcoders WD* can only go to system memory.

Mode Set

A mode set sequence is the programming sequence that must be followed when enabling or disabling output to a display. There are several different mode set sequences documented in the following sections. The sequence to use depends on which type of port is being enabled or disabled.

Sequences for Half Refresh Rate

There is a significant power cost for simply handling the CPU interrupts on alternate frames even though there is no new content to be delivered to the screen at this time. Half Refresh rate is a power saving feature to disable CPU interrupts on the frames that are repeated. This feature ensures two things:

1. Ensures that VBI's are generated at the half rate
2. Ensures that PLANE1 Base Address only updates produce flip done interrupts at the same time as VBI

Any operations outside of the above may result in screen updates and Flip Done interrupts being generated out of phase with VBI. This is expected to be transitional for full screen video playback (e.g. cursor movements), the bulk of normal video operations should be covered by point 2 above.

Requirements

Can be applied with only single plane for RGB or two planes for YUV case.

Half Refresh Rate entry sequence

1. Disable PSR2 Deep Sleep. This can be done using either of the following mechanisms (or both) to ensure PSR2 exits Deep Sleep (if it is currently in the Deep Sleep state) and blocks it from entering Deep Sleep going forward:
    - Enable and unmask the Pipe VBI
    - Zero out PSR2_CTL[Idle Frames] and program a Pipe palette register
2. Proceed with the rest of HRR entry programming.
    - Setting 0x8F080[31] will enable HRR
    - 0x8F080[30] is an indicator that DMC is busy executing half refresh rate

Half refresh Rate exit sequence

1. Finish the HRR exit programming by disabling HRR (0x8F080[31] = 0).
2. Re-enable Deep Sleep to allow PSR2 to enter Deep Sleep, if conditions allow. Both of the following must be true to re-enable Deep Sleep:
    - Disable and mask the Pipe VBI
    - Program PSR2_CTL[Idle Frames] to a non-zero value

Sequences to Initialize Display

These sequences are used to initialize the display engine before any display engine functions can be enabled. To save power, do not initialize display if it will not be used, such as on "headless" systems.

Most display engine functions will not operate while display is not initialized. Only basic PCI, I/O, and MMIO register read/write operations are supported when display is not initialized.

Initialize Sequence

1. Enable PCH Reset Handshake
    a. Set NDE_RSTWRN_OPT RST PCH Handshake En to 1b.
    b. Configure south display Raw Clock before first enabling GMBUS, south display hotplug detection, or panel power sequencing.
2. Enable Power Well 1 (PG1)
    a. Poll for FUSE_STATUS Fuse PG0 Distribution Status = 1b.
        - Timeout and fail after 20 us.
    b. Set PWR_WELL_CTL Power Well 1 Request to 1b.
        - There are two sets of PWR_WELL_CTL registers for software use. It is expected that BIOS uses PWR_WELL_CTL1 and driver uses PWR_WELL_CTL2.
    c. Poll for PWR_WELL_CTL Power Well 1 State = 1b.
        - Timeout and fail after 100us.
            - Typically expected to take 30us with 38.4 MHz reference frequency and 45us with 24 MHz reference frequency.
        - Register DSSM Reference Frequency indicates the reference frequency.
    d. Poll for FUSE_STATUS Fuse PG1 Distribution Status = 1b.
        - Timeout and fail after 20 us.
3. Enable CD clock following the Sequences for Changing CD Clock Frequency
4. Enable first DBUF
    a. Set DBUF_CTL_<first DBUF> DBUF Power Request to 1b.
    b. Poll for DBUF_CTL_<first DBUF> DBUF Power State = 1b.
        - Timeout and fail after 10 us.
    - Additional DBUFs may be enabled later as needed for the display bandwidth and number of pipes enabled.
5. Setup MBUS. Refer to MBus section for the MBus credits programming.
6. Poll PHY_MISC_* dp_tx<0,1,2,3>_ack==0 to indicate that PHYs have completed calibration and adaptation after reset.
    a. Timeout and fail after 25ms.
    b. Poll can be skipped for PHYs that will not be used.

Un-initialize Sequence

Software should only run the un-initialize sequence as part of DC9. It can conflict with other DC states.

1. Disable all display engine functions using the full mode set disable sequence on all pipes, transcoders, ports, planes, and power wells above PG1.
2. Disable DBUFs
    a. Clear DBUF_CTL_* DBUF Power Request to 0b.
    b. Poll for DBUF_CTL_* DBUF Power State = 0b.
        - Timeout and fail after 10 us.
3. Disable CD clock following the Sequences for Changing CD Clock Frequency
4. Disable Power Well 1 (PG1) and Aux IO Power.
    a. Follow AUX Channel Aux IO Power Disabling for every Aux that is powered up.
    b. Clear PWR_WELL_CTL_* Power Well 1 Request to 0b.
    c. Wait for 10us. Do not poll for the power well to disable. Other clients may be keeping it enabled.

Sequences for DisplayPort

This topic describes how to enable and disable DisplayPort.

The DP transport is within the transcoder. Multi-stream and DP2.0 128b/132b requires a transcoder to be enabled to act as the primary transcoder, hosting the DP transport that consolidates the multiple streams. The primary transcoder's clocking, TRANS_DDI_FUNC_CTL, and DP_TP_CTL must be kept enabled while the multistream DP link is enabled, and not enabled and disabled with the video stream.

Enable Sequence

Display must already be initialized.

1. **Enable Power Wells**
    1. Based on the resources to be used, enable the appropriate power wells following the Sequences for Power Wells
2. **If Panel Power Sequencing is required - Enable Panel Power**
    1. Enable panel power sequencing with south display panel power registers.
    2. Wait for panel power sequencing to reach enable state.
3. **Enable Port PLL**
    1. If DP v2.0/128b, then PHY Shim forwarded clock mux is set to DP v2.0 mode by setting SNPS_PHY_MPLLB_DIV[dp2_mode] and div32 clock is selected in SNPS_PHY_MPLLB_DIV[dp_shim_div32_clk_sel] register field.

a) For all other modes, SNPS_PHY_MPLLB_DIV[dp2_mode] is set to '0' and SNPS_PHY_MPLLB_DIV[dp_shim_div32_clk_sel] is set to '0'.

2. If PLL is not already enabled, follow port clock programming sequence from Clocks section

4. **Enable IO power**

    1. Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
    2. Wait for PWR_WELL_CTL DDI IO Power Request = Enabled; timeout after 20 uS

5. **Enable and Train Display Port (this refers to primary transcoder in DP 1.4x and DP v2.0 128b/132b or SST)**

    1. Configure Transcoder Clock Select to direct the Port clock to the Transcoder
    2. If DP v2.0/128b mode – Configure TRANS_DP2_CTL register settings.
    3. Configure TRANS_DDI_FUNC_CTL DDI Select, DDI Mode Select, and MST Transport Select
    4. Configure and enable DP_TP_CTL with link training pattern 1 selected
    5. Configure voltage swing and related IO settings (refer to PHY DDI Buffer section)
    6. Program CoG/MSO configuration bits in DSS_CTL1 if CoG/MSO is selected.
    7. Configure and enable DDI_BUF_CTL
    8. Wait for DDI_BUF_CTL Idle Status = 0b (Not Idle), timeout after 1200 us
    9. Follow DisplayPort specification training sequence (see notes for failure handling).
    10. If DP v2.0/128b, set DP_TP_CTL link training pattern 2.
    11. If DP 1.4 MST, Set DP_TP_CTL link training to Idle Pattern, wait for 5 idle patterns (DP_TP_STATUS Min_Idles_Sent) (timeout after 800 us)
    12. Set DP_TP_CTL link training to Normal
        - If DP 2.0/128b, HW switches to Normal only after sending POST_LT_SCRAMBLER_RESET
    13. If not DP v2.0/128b, Configure and enable FEC in DP_TP_CTL as needed.
        - Hardware automatically configures and enables FEC for DP v2.0/128b.

6. **If not in compliance mode: Enable Planes, Pipe, and Transcoder (repeat to add multiple pipes on a single port for multi-streaming).**

    1. If DP 1.4 MST or DP v2.0/128b - use AUX to program receiver VC Payload ID table to add stream
    2. Configure Transcoder Clock Select to direct the Port clock to the Transcoder. Skip this step if clock select already programmed above.
    3. If DP v2.0/128b mode, configure TRANS_DP2_CTL register settings. Skip this step if this register is already programmed above.
    4. Configure TRANS_DDI_FUNC_CTL DDI mode select, MST transport select and Port Width Select. Skip this step if it is already programmed above.

5. Configure and enable planes (VGA or hires). This can be done later if desired.
6. If VGA - Clear VGA I/O register SR01 bit 5
7. Configure uncompressed joiner in PIPE_DSS_CTL if needed.
8. Configure and enable VDSC if needed.
9. Enable panel fitter if needed (must be enabled for VGA)
10. Configure transcoder timings, M/N/TU/VC payload size, and other pipe and transcoder settings
11. If DP2.0/128b, Configure TRANS_DP2_VFREQHIGH and TRANS_DP2_VFREQLOW registers.
12. Configure TRANS_DDI_FUNC_CTL2 if port sync mode needs to be configured.
    - Refer to "Sequences for DisplayPort Sync Mode" section for big joiner mode operation
13. Configure and enable TRANS_DDI_FUNC_CTL
14. Configure VRR if needed.
    - Refer to VRR programming page for details
    - VRR needs to be programmed after TRANS_DDI_FUNC_CTL enable and before TRANS_CONF enable
15. If DisplayPort multistream - Clear ACT sent status in primary
16. If DP 1.4 MST or DP v2.0/128b - Enable pipe VC payload allocation in TRANS_DDI_FUNC_CTL
17. If DP 1.4 MST or DP v2.0/128b - Wait for ACT sent status in mater transcoder DP_TP_STATUS and receiver DPCD (timeout after >410us)
18. For DP 1.4 MST with FEC, set offset 0x420C0 (for DP on pipe A), 0x420C4 (pipe B), 0x420C8 (pipe C), 0x420D8 (pipe D), bit 23 = 1.
19. Configure and enable TRANS_CONF
20. If panel power sequencing is required - Enable panel backlight

SRD and/or Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Link Training Notes

Changing voltage swing during link training

1. Change the swing setting following the PHY DDI Buffer section. The port does not need to be disabled.

Changing port width (lane count) or frequency during link training

1. Follow Disable Sequence for DisplayPort to Disable Port.
2. If PLL frequency needs to change, follow the Disable Sequence for DisplayPort to Disable PLL.
3. Follow the Enable Sequence for DisplayPort to Enable PLL, using the new frequency settings.
4. Follow the Enable Sequence for DisplayPort to enable and Train DisplayPort, using the new port width settings.

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Disable Sequence

SRD and Audio must be disabled first. Follow the audio disable sequence in the audio registers section. If the flip queue is being used, it should be disabled following the disable sequence in the "Simple Flip Queue Programming Sequences" section.

1. **If panel power sequencing is required - Disable panel backlight**
2. **If not in compliance mode: Disable Planes, Pipe, and Transcoder (repeat to remove multiple pipes from a single port for multi-streaming).**
    1. If VGA
        a) Set VGA I/O register SR01 bit 5 for screen off
        b) Wait for 100 us
    a) Disable planes (VGA or hires)
    b) Disable TRANS_CONF
    c) Clear offset 0x420C0 (for DP on pipe A), 0x420C4 (pipe B), 0x420C8 (pipe C), 0x420D8 (pipe D), bit 23 to 0 if set to 1.
    d) Wait for off status in TRANS_CONF, timeout after two frame times
    e) If DP 1.4 DisplayPort multistream or DP v2.0/128b - use AUX to program receiver VC Payload ID table to delete stream
    f) If done with this VC payload
        a) Clear ACT sent status in primary
        b) Disable VC payload allocation in TRANS_DDI_FUNC_CTL
        c) Wait for ACT sent status in DP_TP_STATUS (multi-stream use the primary transcoder DP_TP_STATUS) and receiver DPCD
    8. Disable VRR, if enabled.
    9. Disable push bit if enabled.

2. If DP 1.4 MST primary transcoder or DP v2.0/128b primary: Disable TRANS_DDI_FUNC_CTL and do not change DDI_Select All other transcoders: Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
3. If DP 2.0/128b secondary transcoder, disable TRANS_DP2_CTL
4. Disable PIPE_DSS_CTL1 Joiner Enable if enabled
5. Disable PIPE_DSS_CTL2 left and right VDSC branch enables if enabled
6. Disable panel fitter
7. If DP 1.4 MSR secondary transcoder or DP v2.0/128b secondary transcoder: Configure Transcoder Clock Select to direct no clock to the transcoder

3. **Disable Port (all pipes and VC payloads on this port must already be disabled). This refers to primary transcoder in DP 1.4x and DP v2.0 128b/132b or SST.**
    1. If DP 1.4 MST primary transcoder or DP v2.0/128b primary transcoder: Set DDI_Select set to None
    2. Disable DDI_BUF_CTL
    3. Disable DP_TP_CTL (do not set port to idle when disabling)
    4. Disable DP_TP_CTL FEC Enable if it is enabled.
    5. Wait 8 us or poll on DDI_BUF_CTL Idle Status for buffers to return to idle
    6. If DP v2.0/128b, disable TRANS_DP2_CTL 128b_132b_channel_coding.
    7. Configure Transcoder Clock select to direct no clock to the transcoder
4. **If panel power sequencing is required - Disable Panel Power**
    1. Disable panel power sequencing with south display panel power registers.
5. **Disable IO Power**
    1. Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used.
6. **Disable Port PLL**
    1. If this PLL is no longer needed, follow PLL disable sequence from Clocks section
7. **Disable Power Wells**
    1. Disable power wells that are no longer required, following the Sequences for Power Wells

Sequences for DisplayPort Sync Mode and Joiner Mode

NOTE: All references are for DisplayPort Sequences

Enabling Display Port Sync Mode (non-MST)

See TRANS_DDI_FUNC_CTL2 Port Sync Mode Enable for restrictions.

1. Follow the enable sequence for the DisplayPort secondary, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
    1. Set secondary TRANS_DDI_FUNC_CTL2 Port Sync Mode Primary Select and Port Sync Mode Enable before configuring and enabling secondary TRANS_DDI_FUNC_CTL.
2. Follow the enable sequence for the DisplayPort primary, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
3. Set DisplayPort secondary DP_TP_CTL link training to Normal.
4. Wait 200 uS.
5. Set DisplayPort primary DP_TP_CTL link training to Normal.

Software may need to use DOUBLE_BUFFER_CTL to ensure updates to plane and pipe registers will take place in the same frame.

For example: If pipe A and pipe B are synchronized together and software needs the surface addresses for two planes to update at the same time, software should use DOUBLE_BUFFER_CTL when writing the surface address registers for both planes, otherwise there is a possibility that the updates could be split across a vertical blank such that one plane would update on the current vertical blank and the other plane would update on the next vertical blank.

Enabling DisplayPort Sync Mode (DP1.4 MST or DPv2.0/128b with 1 DDI)

General constraint is that secondary transcoder has to be enabled first before primary transcoder.

See TRANS_DDI_FUNC_CTL2 Port Sync Mode Enable for restrictions.

1. Follow MST enable sequence for the DisplayPort through the steps 'Enable and Train Display Port'.
2. Enable the secondary pipe following the step 'Enable Planes, Pipe, and Transcoder' of MST enable sequence.
    a) Skip the steps to enable TRANS_DDI_FUNC_CTL2, TRANS_DPT_PAT, TRANS_DDI_FUNC_CTL and TRANS_CONF.
3. Repeat step 'Enable Planes, Pipe, and Transcoder' of MST enable sequence to enable the primary pipe.
    a) Skip the steps to enable TRANS_DPT_PAT, TRANS_DDI_FUNC_CTL and TRANS_CONF.
4. Set secondary TRANS_DDI_FUNC_CTL2 Port Sync Mode Primary Select and Port Sync Mode Enable before configuring and enabling secondary TRANS_DDI_FUNC_CTL. Configure and enable secondary TRANS_CONF and secondary TRANS_DPT_PAT.

5. Configure and enable primary TRANS_DDI_FUNC_CTL. Configure and enable primary TRANS_CONF and primary TRANS_DPT_PAT.

Enabling DisplayPort Sync Mode (DP1.4 MST or DPv2.0/128b with 2 DDIs)

General constraint is that secondary transcoder has to be enabled first before primary transcoder.

See TRANS_DDI_FUNC_CTL2 Port Sync Mode Enable for restrictions.

1. secondary DDI: Follow MST enable sequence for the DisplayPort through the steps 'Enable and Train Display Port'.
2. Enable the secondary pipe following the step 'Enable Planes, Pipe, and Transcoder' of MST enable sequence.
    a) Skip the steps to enable TRANS_DDI_FUNC_CTL2, TRANS_DPT_PAT, TRANS_DDI_FUNC_CTL and TRANS_CONF.
3. primary DDI: Follow MST enable sequence for the DisplayPort steps through the steps 'Enable and Train Display Port'.
4. Repeat step 'Enable Planes, Pipe, and Transcoder' of MST enable sequence to enable the primary pipe.
    a) Skip the steps to enable TRANS_DPT_PAT, TRANS_DDI_FUNC_CTL and TRANS_CONF.
5. Set secondary TRANS_DDI_FUNC_CTL2 Port Sync Mode Primary Select and Port Sync Mode Enable before configuring and enabling secondary TRANS_DDI_FUNC_CTL. Configure and enable secondary TRANS_CONF and secondary TRANS_DPT_PAT.
6. Configure and enable primary TRANS_DDI_FUNC_CTL. Configure and enable primary TRANS_CONF and primary TRANS_DPT_PAT.

Enabling Display Port with Big Joiner OR Uncompressed 2 pipe joiner

Big joiner and uncompressed 2 pipe joiner (such as for 8K on a single port) uses two pipes to drive a single transcoder.

1. Follow the enable sequence through the steps to Enable and Train DisplayPort
2. Follow the steps to Enable Planes and Pipe for secondary pipe. Do not enable transcoder associated with the secondary pipe.
3. Follow the steps to Enable Planes, Pipe, and Transcoder for primary pipe.

Disabling DisplayPort Sync Mode (non-MST)

1. Follow the disable sequence for the DisplayPort Secondary
2. Follow the disable sequence for the DisplayPort Primary

Disabling DisplayPort Sync Mode (DP1.4 MST or DPv2.0/128b with 1 DDI)

1. Follow the disable sequence for the MST DisplayPort secondary pipe but keep secondary VC enabled. Step 'Disable Planes, Pipe, and Transcoder' of Disable Sequence.
2. Follow the disable sequence for the MST DisplayPort primary stream. Step 'Disable Planes, Pipe, and Transcoder' of Disable Sequence.
3. Disable secondary VC.

Disabling DisplayPort Sync Mode (DP1.4 MST or DPv2.0/128b with 2 DDI)

1. Follow the disable sequence for the MST DisplayPort secondary stream . Step 'Disable Planes, Pipe, and Transcoder' of Disable Sequence.
2. Follow the disable sequence for the MST DisplayPort primary stream. Step 'Disable Planes, Pipe, and Transcoder' of Disable Sequence.

Disabling DisplayPort with Big Joiner OR Uncompressed 2 pipe joiner

1. Follow the steps to disable Planes, Pipe, and Transcoder for primary pipe and primary transcoder.
2. Follow the steps to Disable Planes and Pipe for secondary pipe.
3. Disable port associated with primary Transcoder.

Sequences for HDMI and DVI

This topic describes how to enable and disable HDMI and DVI.

Enable Sequence

Display must already be initialized

1. **Enable Power Wells**
   a) Based on the resources to be used, enable the appropriate power wells following the Sequences for Power Wells
2. **Enable Port PLL**
   a) If PLL is not already enabled, follow port clock programming sequence from Clocks section
      - Make sure SNPS_PHY_MPLLB_DIV[dp2_mode] is set to '0' and SNPS_PHY_MPLLB_DIV[dp_shim_div32_clk_sel] is set to '0'.
3. **Enable IO Power**
   a) Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
   b) Wait for PWR_WELL_CTL DDI IO Power Request = Enabled, timeout after <TBD> us
4. **Enable Planes, Pipe, and Transcoder**
   a) Configure Transcoder Clock Select to direct the Port clock to the Transcoder
   b) Configure and enable planes (VGA or hires). This can be done later if desired.

c) If VGA - Clear VGA I/O register SR01 bit 5
d) Enable panel fitter if needed (must be enabled for VGA)
e) Configure transcoder timings and other pipe and transcoder settings
f) Configure and enable TRANS_DDI_FUNC_CTL
g) Configure and enable TRANS_CONF

5. **Enable Port**
   a) Configure voltage swing and related IO settings. Refer to the DDI Buffer section.
   b) Configure and enable DDI_BUF_CTL
   c) Wait for DDI_BUF_CTL DDI Idle Status = 0b (Not Idle), timeout after 1200 us.

Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Notes

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Disable Sequence

Audio must be disabled first. Follow the audio disable sequence in the audio registers section. If the flip queue is being used, it should be disabled following the disable sequence in the "Simple Flip Queue Programming Sequences" section.

1. **Disable Planes, Pipe, and Transcoder**
   a) If VGA
      - Set VGA I/O register SR01 bit 5 for screen off
      - Wait for 100 us
   b) Disable planes (VGA or hires)
   c) Disable TRANS_CONF
   d) Wait for off status in TRANS_CONF, timeout after two frame times
   e) Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
   f) Disable panel fitter
2. **Disable Port**
   a) Disable DDI_BUF_CTL
   b) Wait 8 us or poll on DDI_BUF_CTL Idle Status for buffers to return to idle
   c) Configure Transcoder Clock Select to direct no clock to the transcoder
3. **Disable IO Power**
   a) Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used
4. **Disable Port PLL**

a) If PLL to port mapping is flexible, configure PLL to port mapping to direct no clock to the DDI
b) If this PLL is no longer needed, disable it

5. **Disable Power Wells**
   a) Disable power wells that are no longer required, following the Sequences for Power Well

Sequences for WD

Enable Sequence

Display must already be initialized

1. **Enable Power Wells**
   a) Based on the resources to be used, enable the appropriate power wells following the Sequences for Power Wells
2. **Enable Planes, Pipe, and Transcoder**
   a. Configure WD_STRIDE, WD_SURF, and WD_TAIL_CFG
   b. Configure and enable planes (VGA or hires). This can be done later if desired.
   c. If VGA - Clear VGA I/O register SR01 bit 5
   d. Enable panel fitter if needed (must be enabled for VGA)
   e. Configure transcoder timings and other pipe and transcoder settings
   f. Configure and enable TRANS_WD_FUNC_CTL
   g. Disable underrun recovery. Set 0x70038 (for WD on pipe A), 0x71038 (pipe B), 0x72038 (pipe C), or 0x73038 (pipe D), bit 30 = 1.
   h. Configure and enable TRANS_CONF

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Disable Sequence

If using Triggered Capture Mode (TRANS_WD_FUNC_CTL) wait for the frame to complete (polling WD_FRAME_STATUS Frame Complete or using the WD Frame Complete interrupt) before starting the disable sequence.

1. **Disable Planes, Pipe, and Transcoder**
   a. If VGA
      i. Set VGA I/O register SR01 bit 5 for screen off
      ii. Wait for 100 us
   b. Disable planes (VGA or hires)
   c. If using triggered capture mode, wait for current capture to complete, and do not trigger any more captures

d. Disable TRANS_CONF
e. Disable TRANS_WD_FUNC_CTL
f. Disable panel fitter

2. **Disable Power Wells**
   a) Disable power wells that are no longer required, following the Sequences for Power Wells

Sequences for Display C5 and C6

Display C5 (DC5) is a power saving state where hardware dynamically disables power wells and PLLs and saves the associated registers.

DC5 can be entered when software allows it, only certain power wells are enabled, and hardware detects that all pipes are disabled or low power pipe(s) are enabled with PSR active.

Display C6 (DC6) is a deeper power saving state where hardware dynamically disables power and saves the associated registers.

DC6 can be entered when software allows it, the conditions for DC5 are met, and the PCU allows DC6.

DC6 is required for S0ix.

The context save and restore program is reset on cold boot, warm reset, PCI function level reset, and hibernate/suspend.

DC6 not supported.

Ports A IO/PHY programming is preserved when DC6 is enabled. The programming for other combo PHY ports may not be. After disabling DC6, before enabling the other ports, follow the combo phy initialization sequence and enable Aux IO Power for those ports.

Sequence to Allow DC5 or DC6

1. Load the correct stepping specific Display Context Save and Restore (CSR) program from the binary package.
   a. The Pipe A firmware must be loaded along with the Main DMC firmware .
   b. Other pipes' firmware can also be loaded at this time for Flip Queue, but they will lose the context when the power wells are disabled for those pipes.
   c. Read the package header and extract the correct individual firmware. Binary package format details can be found in sections below.
   d. Skip the header section at the start of the program binary.
   e. Copy the payload into Display CSR Program Storage.
   f. Perform the MMIO writes specified in the header section.
2. Configure display engine to disable power wells from table below, following the appropriate mode set disable sequences for any ports using those power wells. This can be done earlier if desired. DMC will prevent entry if these wells are not disabled.
   - Disable power wells PG2, PGB, and greater pipes

3. Set display register 0x45520 bits [1:0] to 2'b11. The bits do not need to be cleared at any time.
4. Disable flip queue following the section on Simple Flip Queue Programming Sequences.
5. Set DC_STATE_EN Dynamic DC State Enable = "Enable up to DC5" for DC5 or "Enable up to DC6" for DC6.

Programming Note	
Context:	Display C5 and C6
Do not switch between "Enable up to DC5" and "Enable up to DC6" without moving to "Disable" and reloading the CSR program in between. Disable DC5/DC6 during mode set and re-enable after the mode set programming is completed. Disable DC5 and DC6 before a DDI AUX channel transaction is sent. MMIO accesses have more latency when DC5/DC6 is enabled. For optimal performance, disable DC5/DC6 when programming a set of registers and re-enable them after the programming is completed.	

Disable DC5 and DC6 before a DDI AUX channel transaction is sent.

Sequence to Reload DMC Firmware

Before reloading the DMC firmware, the following steps must be taken to disable DMC FW execution.

1. Disable DMC features, DC5/6/6v, DC3CO, flip queue and HRR.

2. Program all of the main DMC and all of the PIPEDMC event control and event HTP registers with the following values:

- DMC_EVT_CTL_<0-7>= 0x00030100
- DMC_EVT_HTP_<0-7> = 0x00000000
- PIPEDMC_EVT_CTL_<0-7>_<All pipes> = 0x00030100
- PIPEDMC_EVT_HTP_<0-7>_<All pipes> = 0x00000000
- The control setting selects a null event, preventing DMC from being triggered to run any program. If control bit 31 was set before this, it will remain set because it can only be cleared by hardware after an even is triggered.

Sequence to Disallow DC5 and DC6

1. Set DC_STATE_EN Dynamic DC State Enable = "Disable".
2. Flip queue can be re-enabled if it will be used, following the section on Simple Flip Queue Programming Sequences.

MIPI DSI

Hardware does not support DC5 or DC6 with MIPI DSI enabled.

Sequence for 3DLUT programming with DC5 and DC6

1. Set DC_STATE_EN Dynamic DC State Enable = "Disable".
2. Driver must program the DSB head tail pointer and control registers to load the 3DLUT DSB program. Driver must follow the DSB programming sequence and instructions to upload the DSB program for 3DLUT. Driver can use any one of the DSB engines in the pipe.
3. Driver must program the following DSB shadow registers with same values as in the DSB head tail pointers and control registers. Each of the pipes have these shadow register as shown below. Driver can use any DSB engine and program these shadow registers with same value.

Pipe A

SHADOW_DSB_HEAD_PTR Offset="0x5F300"

SHADOW_DSB_TAIL_PTR Offset="0x5F304"

SHADOW_DSB_CTRL Offset="0x5F308"

SHADOW_DSB_MMIOCTRL Offset="0x5F30C"

PipeB

SHADOW_DSB_HEAD_PTR Offset="0x5F700"

SHADOW_DSB_TAIL_PTR Offset="0x5F704"

SHADOW_DSB_CTRL Offset="0x5F708"

SHADOW_DSB_MMIOCTRL Offset="0x5F70C"

PipeC

SHADOW_DSB_HEAD_PTR Offset="0x5FB00"

SHADOW_DSB_TAIL_PTR Offset="0x5FB04"

SHADOW_DSB_CTRL Offset="0x5FB08"

SHADOW_DSB_MMIOCTRL Offset="0x5FB0C"

PipeD

SHADOW_DSB_HEAD_PTR Offset="0x5FF00"

SHADOW_DSB_TAIL_PTR Offset="0x5FF04"

SHADOW_DSB_CTRL Offset="0x5FF08"

SHADOW_DSB_MMIOCTRL Offset="0x5FF0C"

4. Set DC_STATE_EN Dynamic DC State Enable = "Enable up to DC5" for DC5 or "Enable up to DC6" for DC6.

5. DMC will do the context save restore of these registers and program the DSB engine to restore the 3DLUT programming.

6. Driver must clear the DSB shadow registers when it disables 3DLUT.

Note: It is driver responsibility to make sure the all the 3DLUT DSB program and all the DSB and shadow registers are correctly updated everytime there is a 3DLUT change.

DMC Firmware Package

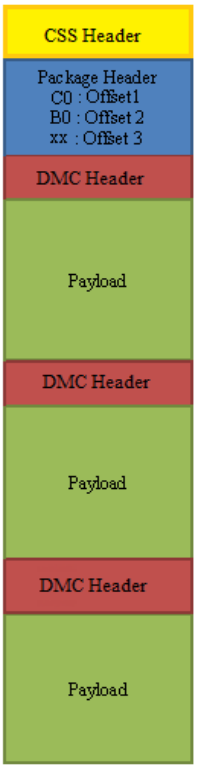
Display Micro-Controller firmware package includes all the firmwares that are required for different steppings of the product. The stepping dependent firmwares are all packaged and released as a single binary package. The package contains the CSS header, followed by the package header and the actual DMC firmwares.

Packaged firmware uses the following naming convention - <project>_dmc_ver<major>_<minor>.bin. The major version will get incremented whenever there is a change in the header layout and would require an update to the driver firmware loading module.

Major version 2

CSS Header	1.0
Package Header	2
DMC Header	3

Package Layout



CSS Header

```
typedef struct _CssHeader {
uint32_t moduleType; // 0x09 for DMC
uint32_t headerLen; // CSS header length in dwords
uint32_t headerVer; // 0x10000
uint32_t moduleID; // Not used
uint32_t moduleVendor; // Not used
uint32_t date; // YYYYMMDD(YYYY « 16 + MM « 8 + DD)
uint32_t size; // Total dmc fw binary size in dwords - (CSS_Headerlen +
PackageHeaderLen + dmc FWsLen)/4
uint32_t keySize; // Not used
uint32_t modulusSize; // Not used
uint32_t exponentSize; // Not used
uint32_t reserved1[12]; // Not used
uint32_t version; // Major Minor
uint32_t reserved2[8]; // Not used
uint32_t uKernelHeaderInfo; // Not used
} CssHeader;
```

Package Header

Package header contains the firmware/stepping mapping table and the corresponding firmware offsets to the individual binaries, within the package. Mapping table will list the exceptions first, followed by the default entries. An Offset value of "0xFFFFFFFF" in the mapping table indicates that there is no firmware available/supported for that stepping. The offsets to the individual binary are DWord aligned. The first individual binary starts at an offset value of "0x00000000" after the CSS Header and the Package Header.

Stepping/Version mapping example

Stepping	FW Version
A1	1.1
B*	1.6
**	2.3

DMC IDs

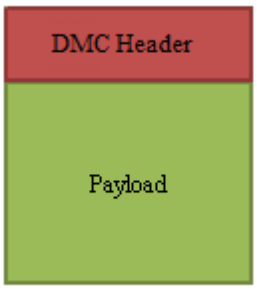
DMC	DMC ID
Main	0
Pipe A	1
PipeB	2
PipeC	3
PipeD	4

```
typedef struct _PackageHeader {
uint8_t headerLen; // DMC package header length in dwords
uint8_t headerVer; // 0x02
uint8_t reserved[10]; // Reserved
uint32_t numEntries; // Number of valid entries in the FWInfo array below
struct _FWInfo_ {
uint8_t reserved1; // Reserved
uint8_t dmc_id; // DMC ID (Main-0, pipeA-1, pipeB-2, pipeC-3, pipeD-4)
char stepping; // Stepping (A, B, C, ..., *). * is a wildcard
char substepping; // Sub-stepping (0, 1, ..., *). * is a wildcard
uint32_t offset; // FW offset within the package in dwords
uint32_t reserved2; // Reserved
} FWInfo[32];
} PackageHeader;
```

DMC firmware binary

Each individual DMC firmware binary has a header followed by a payload whose size is specified in the header section. Along with the version, length, firmware size etc. the header section also specifies a list of MMIO addresses and data. These MMIO write cycles (shown in MMIO programming section below) should be executed as part of the initial CSR program setup.

DMC firmware Layout



MMIO Programming

```
for i = 1 to <mmioCount>
 {
     // Limit MMIO writes to DMC config register range
     if ((mmioaddr[i] >= 0x8F000 and mmioaddr[i] <= 0x8FFFF) or (mmioaddr[i] >= 0x5F000 and
mmioaddr[i] <= 0x5FFFF))
     {
         Perform MMIO write to address <mmioaddr[i]> with data <mmiodata[i]>


    }

}
```

DMC Header - Version 3

```
typedef struct _DMCHeader {
       uint32_t        signature;                      // 0x40403E3E
       uint8_t         headerLen;                      // DMC specific header length in dwords
       uint8_t         headerVer;                      // 0x03
       uint16_t        dmccVer;                        // dmcc compiler version
       uint32_t        project;                        // Major, Minor
       uint32_t        fwSize;                         // Firmware program size in dw
       uint32_t        fwVersion;                      // Major Minor
       uint32_t        startMMIOAddr;                  // DMC RAM start MMIO address
       uint32_t        reserved[9];
       uint8_t         dfile [32];                     // .d file name (encoded using ascii - 40)

  uint32_t mmioCount; // number of mmio
  uint32_t mmioaddr[20]; // MMIO address
  uint32_t mmiodata[20]; // MMIO data
 } DMCHeader;
```

Header field	Value
project	0x000C0000

Sequences for Display C9

Display C9 (DC9) is a power saving state where the display engine is powered off.

DC9 supports S0ix with more power savings than DC6.

Display software must follow certain programming sequences to allow or dis-allow DC9.

Hardware will dynamically enter and exit DC9 when allowed, saving and restoring some of the display state.

Sequence to Allow DC9

1. Follow Sequence to Disallow DC5.
2. Disable all display engine functions using the full mode set disable sequence on all pipes, transcoders, ports, planes, and power wells above PG1.
    - Disable the port filter PLL to save power.
    - Follow the Multichip Genlock section, Sequence to enable MPLLB using filtered genlock reference, steps to Disable filter PLL.
3. Disable and mask all graphics interrupts in north and south display.
4. Save state of MMIO display registers.
    - The exact registers depend on software policy.
    - Hardware will save and restore the PCI Config and DGunit registers
5. Follow Sequences to Initialize Display - Un-initialize Sequence.
6. Set DC_STATE_EN DC9 Allow to 1b.

Sequence to Disallow DC9

1. Clear DC_STATE_EN DC9 Allow to 0b.
2. Follow Sequences to Initialize Display - Initialize Sequence.
3. Restore state of MMIO display registers:
    - The exact registers depend on software policy.
    - The context save and restore program is reset on DC9 and has to be restored.
    - Re-enable the port filter PLL.
    - Follow the Multichip Genlock section, Sequence to enable MPLLB using filtered genlock reference, steps to Enable filter PLL.
4. Enable and unmask graphics interrupts as needed.

Resolution Support

A display resolution is only supported if it meets all the restrictions below for Maximum Pipe Pixel Rate, Maximum Port Link Rate, Maximum Size, Maximum Bandwidth, and Maximum Watermark.

Core Display Clock (CDCLK)

Refer to the Clocks section for details of the frequencies. The frequency is selected to meet the requirements for rates below and DSC bandwidth (DSC section, bandwidth calculations).

Scaling

A scaler (pipe or plane scaler) is down scaling when it is enabled and the scaler input size is greater than the scaler output size.

Down scaling effectively increases the pixel rate before the scaler and reduces the pixel rate after the scaler.

Up scaling is not counted towards the pixel rate.

For plane scaling, the scaler input size is the plane size and the output size is the scaler window size.

For pipe scaling, the scaler input size is the pipe source size and the output size is the scaler window size.

Hscale raw = Horizontal scaler input size / Horizontal scaler output size

If Hscale raw > 1 // Downscaling

Hscale PPC = Hscale raw * NUMPPC // NUMPPC=2

Hscale PPC int = INTEGER(Hscale PPC)

Hscale PPC frac = Hscale PPC - Hscale PPC int

If Hscale PPC frac > 0, Hscale PPC adjusted frac = 1/ROUNDUP[1/Hscale PPC frac], else Hscale PPC adjusted frac = 0 // Account for PPC granularity

Horizontal down scale amount = (Hscale PPC int/NUMPPC) + Hscale PPC adjusted frac

Else Horizontal down scale amount = 1

Vertical down scale amount = maximum[1, Vertical scaler input size / Vertical scaler output size]

// The progressive fetch - interlace display mode is equivalent to a pipe scaler 2.0 vertical down scale multiplied with any additional scaling

Down scale amount = Horizontal down scale amount * Vertical down scale amount

Maximum Pipe Pixel Rate

The display resolution must fit within the maximum pixel rate output from the pipe.

For each enabled plane on the pipe {

    If plane scaling enabled {

        Plane Ratio = 1 / Plane down scale amount

    }

    Else {

        Plane Ratio = 1

    }

}

Pipe Ratio = Minimum Plane Ratio of all enabled planes on the pipe

If pipe scaling is enabled {

    Pipe Ratio = Pipe Ratio / Pipe down scale amount

}

Pipe maximum pixel rate = 2 * CDCLK frequency * Pipe Ratio

Pipe maximum pixel rate = MIN(1200 MHz, Pipe maximum pixel rate) // Limit to maximum validated frequency

YUV420 Full Blend Mode

Pipe Maximum Y channel pixel rate = 2 * CDCLK frequency * Y channel Pipe ratio

Resolutions Requiring Joined Pipes

For resolutions requiring multiple pipes to be combined together (pipe joining), the pixel rate seen by each pipe is 1/<# of joined pipes> of the pixel rate of the full resolution.

The overlapping excess horizontal pixels added for scaling smoothly across the seam between pipes do not impact the pixel rate.

For example: 7680x4320 CVT1.2 RB1 pixel rate is 2089.75 MHz. That is split across 2 pipes, so each pipe is 3840x4320 with a pixel rate of 1044.875 MHz.

Maximum Port Link Rate

The display resolution must fit within the maximum link rate for each port type.

Refer to the project overview sections for the maximum rates.

YUV420 Full Blend Mode

Maximum Port Link Rate = CDCLK frequency * Y channel Pipe Ratio * (bits per color / 8)

Maximum Size

There are limits on the maximum horizontal and vertical size.

Plane stride maximum listed in PLANE_STRIDE register.

Multiple pipes can be joined together side by side in the display engine to drive a single port or a tiled panel.

Unjoined pipe source size, plane size, and pipe active size maximum vertical 4096.
Unjoined pipe source size, plane size, and pipe active size maximum horizontal 5120.
Unjoined pipe source size, plane size, and pipe active size maximum horizontal 5120.
Joined pipes active size maximum is 7680x4320, with each pipe and maximum horizontal plane size at 7680/<# of joined pipes> and vertical size at 4320, plus extra overlapping horizontal pixels for scaling excess.
PSR2 maximum pipe horizontal active size 5120 pixels. PSR2 has additional restrictions on minimum vblank size and horizontal total line time. See the PSR section for those.
LACE DPST maximum size 5120x3200 pixels.
Pipe scalers maximum horizontal source size 5120 pixels.

Maximum Bandwidth

Refer to the Bandwidth Restrictions section

Maximum Watermark

The display resolution must not exceed the level 0 maximum watermark value. See the section on Watermark Programming.

Example Display Resolution Capabilities

Refer to the project overview sections for the common resolutions that meet all the resolution restrictions.

Examples

Example pipe pixel rate

Plane 1 enabled at 64bpp and plane down scale amount 1.25, plane 2 enabled at 32bpp, no pipe scaling enabled, and CDCLK 312 MHz:

Plane 1 ratio = 1/1.25 = 0.8

Plane 2 ratio = 1

Pipe ratio = Minimum[1, 0.8] = 0.8

Pipe maximum pixel rate = 2 * 312 MHz * 0.8 = 499.2 MHz

Bandwidth Restrictions

There are bandwidth restrictions that limit the display resolution and configuration. A display resolution is only supported if it meets all the restrictions below for Maximum Memory Read Bandwidth found from the Available Memory Bandwidth Calculation and Required Memory Bandwidth Calculation, Maximum Data Buffer Bandwidth, and Maximum Pipe Read Bandwidth.

Maximum Memory Read Bandwidth

The display required memory bandwidth for a resolution configuration must not exceed the available memory bandwidth.

Calculate the available memory bandwidth and the required memory bandwidth. If there is not enough available memory bandwidth, then the display configuration cannot be supported.

Available Memory Bandwidth Calculation

If SoC type is x128 (based on device ID or fuses), available bandwidth is 38 GB/s, else 50 GB/s.

Memory efficiency does not need to be calculated and reduced by number of planes.

Required Memory Bandwidth Calculation

This calculates the display required memory bandwidth.

Note on planar formats: The NV12 format counts as 2 planes of 2 Bpp each. The P01x formats count as 2 planes of 4 Bpp each. Vertical sub-sampling does not reduce this calculation because the bandwidth has to be counted over a few microsecond period.

DBUF maximum data buffer bandwidth MB/s = CDCLK frequency MHz * 64 Bytes

For each DBUF {

> DBUF maximum pipe bandwidth MB/s = DBUF maximum data buffer bandwidth / number of enabled pipes using this buffer
>
> For each pipe using this buffer {

Pipe cumulative bytes per pixel = 0

DBUF maximum plane bandwidth MB/s = DBUF maximum pipe bandwidth / number of enabled planes

For each plane enabled on the pipe { // cursor can be ignored because it only impacts bandwidth for a very short time

Plane required bandwidth MB/s = pixel rate MHz * source pixel format in bytes * plane down scale amount * pipe down scale amount

Display required memory bandwidth MB/s += Plane required bandwidth

Pipe cumulative bytes per pixel += plane source pixel format in bytes

If plane required bandwidth > DBUF maximum plane bandwidth {Return: Failure maximum data buffer bandwidth is exceeded}

}

If pipe cumulative bytes per pixel> (CDCLK frequency MHz /(pixel rate MHz * plane down scale amount * pipe down scale amount)) * 51.2 {Return: Failure maximum pipe read bandwidth is exceeded}

}

}

If VTD is enabled {

Display required memory bandwidth MB/s *= 1.05

}

Return: Display required memory bandwidth MB/s

Maximum Data Buffer Bandwidth

The display resolution must not exceed the maximum bandwidth from each display data buffer (DBUF). There are multiple buffers that can be enabled, and planes can be allocated data blocks from multiple. See the Display Buffer Programming section for requirements on how and when to use multiple buffers.

This is calculated as part of the Required Memory Bandwidth Calculation where it returns a failure if maximum data buffer bandwidth is exceeded. The calculation for this check can be separated out if required.

Maximum Pipe Read Bandwidth

The display resolution must not exceed the maximum memory read bandwidth from each display pipe. This restriction is met by limiting the cumulative bytes per pixel for the pipe, which accounts for the limits of memory reads, translation reads, and data buffer reads.

This is calculated as part of the Required Memory Bandwidth Calculation where it returns a failure if maximum pipe read bandwidth is exceeded. The calculation for this check can be separated out if required.

High Refresh Rate and Small Vblank Support

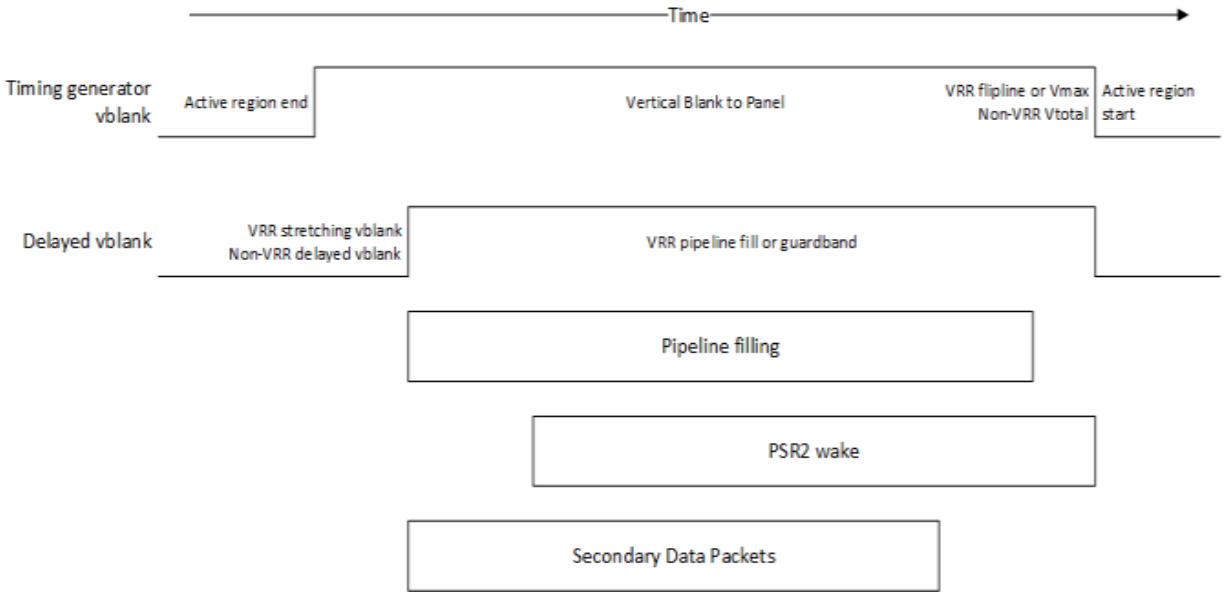
High refresh rate panels may have small line times and vertical blank (vblank). The vblank internal to the display pipe can also be reduced to provide an extended region for programming. The vblank is used by hardware to prepare for the next frame. Very small vblanks limit the display features.

With fixed refresh rate the frame preparation happens within the programmed transcoder vertical blank start to vertical blank end. If the start of vblank internal to the display pipe is delayed relative to the end of vertical active by programming vblank start greater than vertical active, typically to provide a DSB programming window or faster flip performance, then the preparation time is reduced.

TRANS_SET_CONTEXT_LATENCY is used to delay the start of vblank internal to the display pipe.

With variable refresh rate the frame preparation happens during the guardband (also called pipeline fill + framestart) time.

The vblank must be sized so that the frame preparation time is large enough for enabled features. The horizontal line time must also account for PSR2.



Calculate the requirements for each feature:

Framestart delay = line time * 1

Framestart delay cannot be disabled, so vblank must meet the requirement for it.

Package C state latency = latency used to calculate the highest enabled watermark level 1 and up, across enabled planes and pipes

There are multiple package C states and increasing latencies for the higher numbered states.

If certain package C states have too much latency, they can be disabled by disabling the associated low power watermarks across enabled planes and cursors, with power impact.

If any low power watermark (level 1 and up) is disabled because the package C state has too much latency for the size of Vblank and PSR1 or PSR2 is enabled, set the register bit for this pipe (listing below) to 1 to disable a PSR optimization to override to the maximum watermark. Clear the bit if the size of Vblank does not require low power watermarks to be disabled or PSR* is disabled.

Pipe A 0x46430 bit 23

Pipe B 0x46430 bit 24

Pipe C 0x46430 bit 25

Pipe D 0x46430 bit 31

Panel Replay (PR) requires the same bit setting as PSR1 and PSR2 above.

SAGV latency = SAGV block time if SAGV is enabled

SAGV can be disabled if it will not fit the vblank. To disable SAGV, follow the SAGV section SAGV Point Selection Runtime flow to mask off all but the one QGV point that supplies the highest bandwidth for display.

The SoC power controller runs SAGV mutually exclusive with package C states, so the max of package C and SAGV latencies are used in the final calculation for vblank time requirement.

Watermark 0 pre-fill time = Maximum time to fill the data buffer up to watermark 0 = line time * highest enabled plane or cursor watermark 0 result in lines

Watermark 0 cannot be disabled, so vblank must meet the requirement for it.

This accounts for the latency for level 0 and buffering up enough data to generate a line.

Pipe scaler pre-fill time = Time for first scaler in pipeline + Time for second scaler in pipeline

Time for first scaler in pipeline = first scaler enable * 4 * line time

downscale amount <for each direction and scaler> = MAX(1, scaler input / scaler output)

Chroma subsampling is a 2x downscale

Time for second scaler in pipeline = second scaler enable * 4 * line time * first scaler vertical downscale amount * first scaler horizontal downscale amount

First scaler downscale reduces how quickly the first scaler can output pixels to pre-fill the second scaler.

Pipe scalers can be disabled and fall back to composition outside of display.

DSC pre-fill time = DSC enable * 1.5 * line time * first scaler vertical downscale amount * first scaler horizontal downscale amount * second scaler vertical downscale amount * second scaler horizontal downscale amount

DSC can be disabled if there is enough link bandwidth to support that.

Scaler downscales reduce how quickly DSC can pre-fill.

PSR2 vblank time = PSR2 enable * Minimum block count * line time.

Minimum block count = Register PSR2_CTL Block Count Number maximum line count

PSR2 line time = PSR2 enable * PSR2 IO wake time / PSR2 wake lines

For PSR2 IO wake time, refer to the section on Panel Self Refresh.

PSR2 wake lines = Register PSR2_CTL IO Buffer Wake maximum line count

PSR2 can be disabled, with power impact and loss of DC6v.

SDP vblank time = MAX(PPS enable * 7, GMP = GMP enable * 8, VSC_EXT enable * 10) * line time

These secondary data packets are sent on specific lines.

PPS SDP can be disabled if DSC can be disabled.

VSC_EXT at 10 lines assumes it is enabled before vblank line 8 and delivers only one full buffer.

GMP and VSC_EXT SDPs cannot be disabled if the panel or video requires them.

Pre-fills are calculated for the worst case to take the same time to fill as to drain.

Calculate the resolution requirements:

Vblank time >= MAX(framestart delay + package C state latency + watermark 0 time + pipe scaler pre-fill time + DSC pre-fill time, PSR2 vblank time, SDP vblank time)

Line time >= PSR2 line time

Disable features to fit within the vblank and line time requirements or restrict the resolution or refresh rate to a larger vblank time or line time.

DRRS changes pixel rate, increasing line time and Vblank, so the calculations can be re-evaluated and enabled features adjusted. Note that DRRS and other features may not have hardware for atomic updates, so software must align programming to the vertical active region of the screen or in a multiple frame sequence.

Clocks

Registers

CDCLK_CTL

CDCLK_PLL_ENABLE

CDCLK_SQUASH_CTL

SNPS_PHY_MPLLB_CP

SNPS_PHY_MPLLB_DIV

SNPS_PHY_MPLLB_DIV2

SNPS_PHY_MPLLB_SSCEN

SNPS_PHY_MPLLB_FRACN1

SNPS_PHY_MPLLB_FRACN2

PHY_PLL_ENABLE

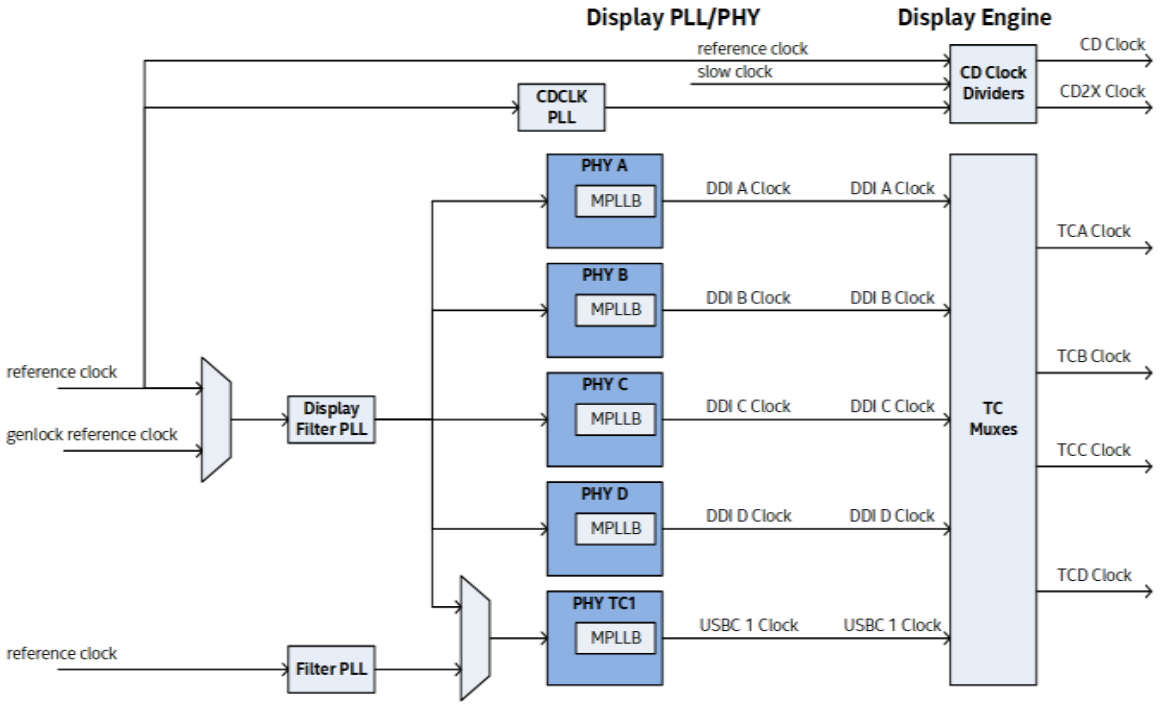
TRANS_CLK_SEL

TIMESTAMP_CTR

Overview of Display Clock Paths

The display engine clocking structure has multiple PLLs and clocks. The flow is from PLL to DDI (port) clock to transcoder clock.

PLL Arrangement



PHY PLLs

Each port has its own dedicated PLL from its PHY.

The PLL output is divided by 10 (non-DP2.0) or 20 (DP2.0) to become the symbol/word/TMDS clock frequency used in the display engine.

	PHY PLLs
Usage	Sources for DDI and UC clocks used by the display ports.
Input	Filtered reference
Frequency	Programmable** eDP/DP link bit rates: 1.62, 2.16, 2.7, 3.24, 4.32, 5.4, 8.1 GHz, SSC and Non-SSC DP 2.0 link bit rates: 10 and 13.5 GHz, SSC and Non-SSC HDMI/DVI symbol rates: 20 to 600 MHz, Non-SSC **OEM must use VBT to specify a maximum that is tolerated by the board design
Default after reset	Disabled
Programming	Must be programmed by software when enabling and disabling a display output. See the section on Port Clock Programming. PLLs are automatically disabled and re-enabled by hardware for some power states.

Filter PLLs

The filter PLLs cleans up the reference inputs for use by the PHY PLLs.

	Display Filter PLL	Audio Filter PLL
Usage	Reference clock for dedicated display PHY PLLs and the type-C PHY PLL when strapped to support a native display connection.	Reference clock for type-C PHY PLL when strapped to support a USB connection.
Input	Muxed between genlock reference input and crystal	Crystal
Frequency	100 MHz	38.4 MHz
Default after reset	Enabled by SoC firmware	Enabled by SoC firmware
Programming	Must be programmed by display software when enabling or disabling genlock.	Not programmable by display software.

CDCLK PLL

The CDCLK PLL is the main source for the display core clock (CDclk). A programmable divider inside the PLL controls the PLL frequency.

	CDCLK PLL
Alternate Name	DEPLL
Usage	Source for CD clock
Input	Reference clock (38.4MHz)
Frequency	Programmable - Frequencies in table below
Default after reset	Disabled
Programming	Must be programmed by software when enabling and disabling a display. May be automatically enabled and disabled by hardware for some power states. Programming is done through the CDCLK_PLL_ENABLE register.

Display Engine Clocks

Reference Clock

There is one display engine reference clock.

	Reference Clock
Usage	Reference for the PLLs and for miscellaneous timers in display engine. It is also used as a source for the core clock (CDclk) when the CDCLK PLL is not enabled.
Frequency	38.4 MHz Non-SSC (Register DSSM Reference Frequency indicates the frequency)
Default after reset	Enabled
Programming	Not programmable by display software.

Slow Clock

The slow clock is used as a source for the core clock when the CDCLK PLL is not enabled.

	Slow Clock
Usage	Used as a source for the core clock (CDclk) when the CDCLK PLL is not enabled.
Frequency	38.4 MHz Non-SSC
Default after reset	Enabled
Programming	Not programmable by display software.

Core Clock

CD clock refers to the Core Display clock which includes the Core Display 1X Clock (CD clock, CDclk, cdclk, CDCLK) and the Core Display 2X Clock (CD2X clock, cd2xclk, CD2XCLK).

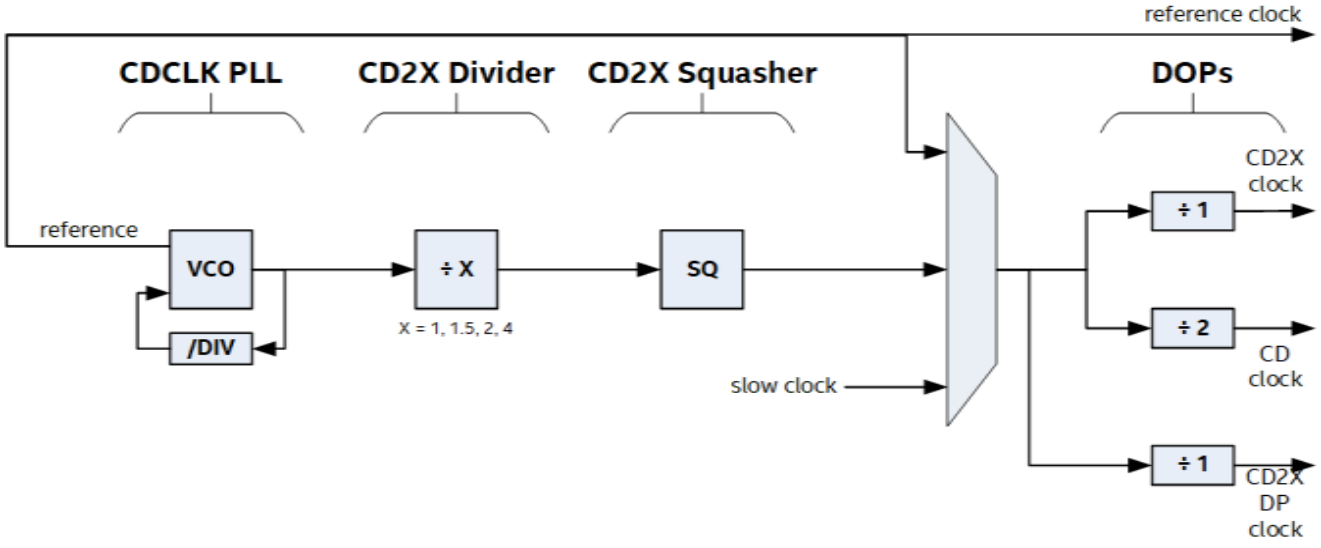
	CD clock
Usage	Clocking for most display engine functions.
Input	CDCLK PLL output, slow clock or reference clock.
Frequency	CDCLK PLL disabled - reference or slow clock divided by 2. CDCLK PLL enabled - Frequencies listed in the table below.
Default after reset	Running on the reference clock.
Programming	Must be programmed by software when enabling and disabling a display. Programming is done through the CDCLK_CTL and CDCLK_PLL_ENABLE registers.

CD Clock Generation

The CD clock has multiple clock sources (CDCLK PLL, reference clock and cro/slow clock). When the CDCLK PLL is disabled, the CD clock runs at reference clock divided by 2 to maintain register accessibility. The slow clock source is used by hardware to speed up some power state transitions (slow clock runs at a lower frequency so this benefit is negated).

The CD2X Divider is a clock divider internal to the display engine and can be programed by the CDCLK_CTL register. It enables glitch-free, on-the-fly CD clock frequency change through clock division. This method of frequency change does not require a PLL relock.

The CD2x Squasher is a dynamic clock gate that can be programmed by the CDCLK_SQUASH_CTL register. It enables glitch-free, on-the-fly CD clock frequency change by squashing (clock gating) clock edges to change the effective frequency. This method of frequency change does not require a PLL relock.



CDCLK Frequencies, PLL Ratio, Divider and Frequency Decimal Programming

The traditional method for programming the CDclk frequency is to set the PLL to the desired frequency x2.

CDCLK (MHz)	CDCLK_PLL_ENABLE PLL Ratio	CDCLK_CTL CD2X Divider	CDCLK_CTL Freq Decimal	AUD_TS_CDCLK_M Value	AUD_TS_CDCLK_N Value
38.4MHz Reference					
172.8	9	1	00101011000b	60 (3Ch)	432 (1B0h)
192	10	1	00101111110b	60 (3Ch)	480 (1E0h)
307.2	16	1	01001100100b	60 (3Ch)	768 (300h)
326.4	34	2	01010001011b	60 (3Ch)	816 (330h)
556.8	29	1	10001011000b	60 (3Ch)	1392 (570h)
652.8	34	1	10100011000b	60 (3Ch)	1632 (660h)

CDCLK Frequencies, PLL Ratio, Divider, Squash Window and Frequency Decimal Programming

The clock squashing method for programming the CDclk frequency is to set the PLL to the max frequency (1305.6MHz) and then use the internal squasher block to achieve the desired CDclk frequency. CDCLK frequencies below 150 MHz not supported, limiting the minimum squash setting.

CDCLK (MHz)	CDCLK_PLL_ENABLE PLL Ratio	CDCLK_CTL CD2X Divider	CDCLK_SQUASH_CTL Squash Wave	CDCLK_CTL Freq Decimal	AUD_TS_CDCLK_M Value	AUD_TS_CDCLK_N Value
38.4MHz Reference						
163.2	34	1	1000100010001000b	00101000100b	60 (3Ch)	408 (198h)
204	34	1	1001001001001000b	00110010110b	60 (3Ch)	510 (1FEh)
244.8	34	1	1010010010100100b	00111101000b	60 (3Ch)	612 (264h)
285.6	34	1	1010010101001010b	01000111001b	60 (3Ch)	714 (2CAh)
326.4	34	1	1010101010101010b	01010001011b	60 (3Ch)	816 (330h)
367.2	34	1	1010110101011010b	01011011100b	60 (3Ch)	918 (396h)
408	34	1	1011011010110110b	01100101110b	60 (3Ch)	1020 (3FCh)
448.8	34	1	1101101110110110b	01110000000b	60 (3Ch)	1122 (462h)
489.6	34	1	1110111011101110b	01111010001b	60 (3Ch)	1224 (4C8h)
530.4	34	1	1111011111011110b	10000100011b	60 (3Ch)	1326 (52Eh)
571.2	34	1	1111111011111110b	10001110100b	60 (3Ch)	1428 (594h)
612	34	1	1111111111111110b	10011000110b	60 (3Ch)	1530 (5FAh)
652.8	34	1	1111111111111111b	10100011000b	60 (3Ch)	1632 (660h)

DDI clocks

There is one DDI clock tied to each DDI port.

A single DDI clock output may be used by multiple transcoders simultaneously for DisplayPort Multi-streaming.

	DDI clocks
Usage	DDI ports I/O bit clock, symbol/TMDS clock and source for transcoder clocks.
Input	PHY PLL
Frequency	PLL output frequency divided by 10 (non-DP2.0) or 20 (DP2.0).
Default after reset	Disabled
Programming	Must be programmed by software when enabling and disabling a display. Programming is done through the PLL registers. See the section on Port Clock Programming.

Transcoder Clocks

There is one transcoder clock tied to each display transcoder.

	Transcoder clocks
Usage	Transcoder symbol/TMDS clocks.
Input	Programmable selection between DDI clocks.
Frequency	DDI clock frequency
Default after reset	Disabled
Programming	Must be programmed by software when enabling and disabling a display. Programming is done through the TRANS_CLK_SEL registers. See the section on Port Clock Programming.

IOSF Clocks

There is 1 IOSFP endpoint clock. This clock follows clkreq/clkack protocol.

There is 1 IOSFSB endpoint clock. This clock follows clkreq/clkack protocol.

	IOSF clocks (primary/sideband)
Usage	Clocking for IOSF interface functions.
Input	Fabric clock derived from SA PLL.
Frequency	Primary/PrimaryDP: 533MHz Side: 400MHz Static; no frequency change.
Default after reset	Clock edges while SAPLL booting then gated until clkreq asserted.
Programming	Not programmable by display software.

Direct Memory Path Clock

The direct memory path IOSF interface runs on the display CD2X clock. The clock is forwarded from the display engine to the memory controller along with the data.

	DPIOSF clock
Usage	Clocking for direct path IOSF interface functions.
Input	Comes from the CDCLK PLL forwarded from display engine.
Frequency	Same as the programmed cd2xclk.
Default after reset	Running on the reference clock.
Programming	Must be programmed by software when enabling and disabling a display. Programming is done through the CDCLK_CTL and CDCLK_PLL_ENABLE registers.

Port Clock Programming

PLL and Clock Usage

Port PLL = MPLLB instance of PHY

Port PLL is embedded in PHY common block. No DDI clock muxing.

PLL Frequency Changes

PLL frequency should not be changed while the PLL is enabled.

1. Follow PLL Disable Sequence
2. Follow PLL Enable Sequence using the new frequency

PLL Enable Sequence

DE runs DPLL bring up sequence after SoC completes PHY initialization flow during reset. MPLLB default is set to 13.5GHz.

1. If enabling or disabling genlock, first configure the reference clock and enable it before programming the port PLL. See Multichip Genlock section.
2. Hardware ensures correct P-state (dp_txX_pstate[1:0]) for configuring PLL.
3. Software programs the following PLL registers for the desired frequency.
    - **SNPS_PHY_MPLLB_CP**
    - **SNPS_PHY_MPLLB_DIV**
    - **SNPS_PHY_MPLLB_DIV2**
    - **SNPS_PHY_MPLLB_SSCEN**
    - **SNPS_PHY_MPLLB_SSCSTEP**
    - **SNPS_PHY_MPLLB_FRACN1**
    - **SNPS_PHY_MPLLB_FRACN2**
4. If the frequency will result in a change to the voltage requirement, follow the Display Voltage Frequency Switching - Sequence Before Frequency Change.
5. Software sets **DPLL_ENABLE** [PLL Enable] to "1".
6. Hardware ensures correct P-state for configuring PLL.
7. Hardware generates a new transmitter setting request (dp_txX_req = 1).
8. Hardware waits for PHY acknowledgement (dp_txX_ack) that the new transmitter setting request is completed.
9. Software sets SNPS_PHY_MPLLB_DIV dp_mpllb_force_en to "1". This will keep the PLL running during the DDI lane programming and any typeC DP cable disconnect. Do not set the force before enabling the PLL because that will start the PLL before it has sampled the divider values.
10. Software polls on register DPLL_ENABLE [PLL Lock] to confirm PLL is locked at new settings. This register bit is sampling PHY dp_mpllb_state interface signal.
11. If the frequency will result in a change to the voltage requirement, follow the Display Voltage Frequency Switching - Sequence After Frequency Change.

PLL Disable Sequence

1. If the frequency will result in a change to the voltage requirement, follow the Display Voltage Frequency Switching - Sequence Before Frequency Change.
2. Software programs DPLL_ENABLE [PLL Enable] to "0"
3. Hardware generates a new transmitter setting request (dp_txX_req = 1)
4. Software programs SNPS_PHY_MPLLB_DIV dp_mpllb_force_en to "0". This will allow the PLL to stop running.
5. Software polls DPLL_ENABLE [PLL Lock] for PHY acknowledgement (dp_txX_ack) that the new transmitter setting request is completed.
6. If the frequency will result in a change to the voltage requirement, follow the Display Voltage Frequency Switching - Sequence After Frequency Change.

Link Rates Supported

<table>
<tr><th>Protocol</th><th>Refclk</th><th>Ports</th><th>Details</th><th>SSC</th></tr>
<tr><td>HDMI</td><td>100 MHz</td><td>A/B/C/D/TC1</td><td>
<table>
<tr><th>PHY designation</th><th>Pixel Clock</th><th>Base Bit Rate per lane</th></tr>
<tr><td>hdmi_251p75</td><td>25.175M pixel/sec</td><td>251.75 Mbps</td></tr>
<tr><td>hdmi_270</td><td>27.0M pixel/sec</td><td>270 Mbps</td></tr>
<tr><td>hdmi_742p5</td><td>74.25M pixel/sec</td><td>742.5 Mbps</td></tr>
<tr><td>hdmi_1p485</td><td>148.5M pixel/sec</td><td>1485 Mbps</td></tr>
<tr><td>hdmi_5G94</td><td>594M pixel/sec</td><td>5.94 Gbps</td></tr>
</table>
Base bit rate pixel clock assumes 8bpc.

Base bit rate = Pixel clock * 8bpc * 10/8 where 10/8 is the code rate

An additional multiplier accounts for the color depth.

So, actual bit rate = Base bit rate * (color depth/8)

So, for 24bpp, 8bpc, the per-lane bit rate is

25.175M pixel/second * 8b/pixel *10/8 code overhead = 251.75Mbps for one lane. (251.75Mbps for each R, G, B lane)

For 30bpp, 10bpc, the per-lane bit rate is

25.2M pixel/second * 10b/pixel *10/8 code overhead = 315Mbps for one lane

For 36bpp, 12bpc, the per-lane bit rate is
</td><td>SSC disable</td></tr>
</table>

<table>
<tr><th>Protocol</th><th>Refclk</th><th>Ports</th><th>Details</th><th>SSC</th></tr>
<tr><td></td><td></td><td></td><td>25.2M pixel/second * 12b/pixel *10/8 code overhead = 378Mbps for one lane.

A multiplier of 1000/1001 enters into the calculation for exact TV frequencies.</td><td></td></tr>
<tr><td>DP 1.4 (ALT)</td><td>38.4 MHz</td><td>TC1</td><td><table><tr><th>PHY designation</th><th>Link Symbol Clock</th><th>Base Bit Rate per lane</th></tr><tr><td>dp_rbr</td><td>162 MHz</td><td>1.62 Gbps</td></tr><tr><td>dp_hbr1</td><td>270 MHz</td><td>2.70 Gbps</td></tr><tr><td>dp_hbr2</td><td>540 MHz</td><td>5.40 Gbps</td></tr><tr><td>dp_hbr3</td><td>810 MHz</td><td>8.10 Gbps</td></tr></table></td><td>SSC enable/disable</td></tr>
<tr><td>DP 1.4 (fixed)</td><td>100 MHz</td><td>A/B/C/D/TC1</td><td><table><tr><th>PHY designation</th><th>Link Symbol Clock</th><th>Base Bit Rate per lane</th></tr><tr><td>dp_rbr</td><td>162 MHz</td><td>1.62 Gbps</td></tr><tr><td>dp_hbr1</td><td>270 MHz</td><td>2.70 Gbps</td></tr><tr><td>dp_hbr2</td><td>540 MHz</td><td>5.40 Gbps</td></tr><tr><td>dp_hbr3</td><td>810 MHz</td><td>8.10 Gbps</td></tr></table></td><td>SSC enable/disable</td></tr>
<tr><td>DP 2.0</td><td>38.4 MHz</td><td>TC1</td><td><table><tr><th>PHY designation</th><th>Link Symbol Clock</th><th>Base Bit Rate per lane</th></tr><tr><td>dp_uhbr10</td><td>312.5 MHz</td><td>10.0 Gbps</td></tr><tr><td>dp_uhbr13p5</td><td>421.875 MHz</td><td>13.5 Gbps</td></tr></table></td><td>SSC enabled

DP UHBR has a minimum SSC requirement.</td></tr>
<tr><td>DP 2.0</td><td>100 MHz</td><td>TC1/A/B/C/D</td><td><table><tr><th>PHY designation</th><th>Link Symbol Clock</th><th>Base Bit Rate per lane</th></tr><tr><td>dp_uhbr10</td><td>312.5 MHz</td><td>10.0 Gbps</td></tr><tr><td>dp_uhbr13p5</td><td>421.875 MHz</td><td>13.5 Gbps</td></tr></table></td><td>SSC enabled

DP UHBR has a minimum SSC requirement.</td></tr>
<tr><td>eDP</td><td>100 MHz</td><td>A/B/C/D</td><td><table><tr><th>PHY designation</th><th>Link Symbol Clock</th><th>Base Bit Rate per lane</th></tr><tr><td>eDP_R216</td><td>216 MHz</td><td>2.16 Gbps</td></tr><tr><td>eDP_R243</td><td>243 MHz</td><td>2.43 Gbps</td></tr><tr><td>eDP_R324</td><td>324 MHz</td><td>3.24 Gbps</td></tr><tr><td>eDP_R432</td><td>432 MHz</td><td>4.32 Gbps</td></tr></table></td><td>SSC enable/disable</td></tr>
</table>

PLL Programming DP Values

Note that SoC programs ref_range parameter shown in tables below and display software can optionally check it for error detection on non Type-C ports.

The following table provides programming information for **basic DP link rates** with reference clock set to **100** MHz. All values are in decimal.

PLL Value	Register and Field	dp_rbr	dp_rbr_ssc	dp_hbr1	dp_hbr1_ssc	dp_hbr2	dp_hbr2_ssc	dp_hbr3	dp_hbr3_ssc
Link bit rate Gbps	N/A	1.62	1.62	2.7	2.7	5.4	5.4	8.1	8.1
Link symbol rate MHz	N/A	162	162	270	270	540	540	810	810
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	3	3	3	3	3	3	3	3
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	2	2	2	2	2	2	2	2
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	0	1	0	1	0	1	0	1
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	226	226	184	184	184	184	292	292
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	1	1	0	0	0	0	0	0
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	39321	39321	0	0	0	0	0	0
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	3	3	0	0	0	0	0	0
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	5	5	1	1	1	1	1	1
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	0	0	0	0	0	0	0	0
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	0	38221	0	31850	0	31850	0	47776
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	0	49314	0	41095	0	41095	0	61642
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	0	0	0	0	0	0	0	0
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	0	0	0	0	0	0	0	0
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	0	0	0	0	0	0	0	0
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	2	2	1	1	0	0	0	0
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	1	1	0	1	0	1	0	1

PLL Value	Register and Field	dp_rbr	dp_rbr_ssc	dp_hbr1	dp_hbr1_ssc	dp_hbr2	dp_hbr2_ssc	dp_hbr3	dp_hbr3_ssc
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	0	0	0	0	0	0	0	0
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	2	2	2	2	2	2
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	2	2	3	3	3	3	0	0
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	4	4	4	4	4	4	4	4
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	20	20	20	20	20	20	19	19
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	65	65	65	65	65	65	65	65
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	127	127	127	127	127	127	127	127
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0	0	0	0	0

The following table provides programming information for **basic DP link rates** with reference clock set to **38.4** MHz. All values are in decimal.

PLL Value	Register and Field	dp_rbr	dp_rbr_ssc	dp_hbr1	dp_hbr1_ssc	dp_hbr2	dp_hbr2_ssc	dp_hbr3	dp_hbr3_ssc
Link bit rate Gbps	N/A	1.62	1.62	2.7	2.7	5.4	5.4	8.1	8.1
Link symbol rate MHz	N/A	162	162	270	270	540	540	810	810
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	1	1	1	1	1	1	1	1
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	1	1	1	1	1	1	1	1
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	0	1	0	1	0	1	0	1
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	304	304	248	248	248	248	388	388
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	1	1	1	1	1	1	1	1
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	49152	49152	40960	40960	40960	40960	61440	61440
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	0	0	0	0	0	0	0	0
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	1	1	1	1	1	1	1	1
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	0	0	0	0	0	0	0	0
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	0	49766	0	41472	0	41472	0	62208

PLL Value	Register and Field	dp_rbr	dp_rbr_ssc	dp_hbr1	dp_hbr1_ssc	dp_hbr2	dp_hbr2_ssc	dp_hbr3	dp_hbr3_ssc
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	0	83608	0	69673	0	69673	0	104509
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	0	0	0	0	0	0	0	0
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	0	0	0	0	0	0	0	0
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	0	0	0	0	0	0	0	0
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	2	2	1	1	0	0	0	0
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	1	1	1	1	1	1	1	1
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	0	0	0	0	0	0	0	0
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	2	2	2	2	2	2
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	2	2	3	3	3	3	0	0
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	5	5	5	5	5	5	6	6
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	25	25	25	25	25	25	26	26
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	65	65	65	65	65	65	65	65
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	127	127	127	127	127	127	127	127
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0	0	0	0	0

The following table provides programming information for **UHBR DP link rates** with reference clock set to **100** MHz. All values are in decimal.

PLL Value	Register and Field	dp_uhbr10	dp_uhbr10_ssc	dp_uhbr13p5	dp_uhbr13p5_ssc
Link bit rate Gbps	N/A	10	10	13.5	13.5
Link symbol rate MHz	N/A	312.5	312.5	421.875	421.875
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	3	3	3	3
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	2	2	2	2
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	0	1	0	1
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	368	368	508	508
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	0	0	0	0
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	0	0	0	0

PLL Value	Register and Field	dp_uhbr10	dp_uhbr10_ssc	dp_uhbr13p5	dp_uhbr13p5_ssc
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	0	0	0	0
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	1	1	1	1
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	0	0	0	0
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	0	58982	0	79626
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	0	76101	0	102737
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	1	1	1	1
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	8	8	8	8
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	0	0	0	0
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	0	0	0	0
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	0	1	0	1
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	1	1	1	1
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	3	3
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	0	0	0	0
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	4	4	5	5
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	21	21	45	45
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	65	65	65	65
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	127	127	127	127
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0

The following table provides programming information for **UHBR DP link rates** with reference clock set to **38.4** MHz. All values are in decimal.

Note that DP UHBR protocol has a minimum SSC requirement.

PLL Value	Register and Field	dp_uhbr10	dp_uhbr10_ssc	dp_uhbr13p5	dp_uhbr13p5_ssc
Link bit rate Gbps	N/A	10	10	13.5	13.5
Link symbol rate MHz	N/A	312.5	312.5	421.875	421.875
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	1	1	1	1
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	1	1	1	1
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	0	1	0	1
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	488	488	670	670
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	1	1	1	1

PLL Value	Register and Field	dp_uhbr10	dp_uhbr10_ssc	dp_uhbr13p5	dp_uhbr13p5_ssc
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	27306	27306	36864	36864
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	2	2	0	0
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	3	3	1	1
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	0	0	0	0
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	0	76800	0	103680
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	0	129024	0	174182
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	1	1	1	1
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	8	8	8	8
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	0	0	0	0
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	0	0	0	0
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	1	1	1	1
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	1	1	1	1
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	3	3
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	0	0	0	0
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	5	5	6	6
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	26	26	56	56
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	65	65	65	65
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	127	127	127	127
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0

The following table provides programming information for **eDP link rates** with reference clock set to **100** MHz. All values are in decimal.

PLL Value	Register and Field	eDP_R216	eDP_R243	eDP_R324	eDP_R432
Link bit rate Gbps	N/A	2.16	2.43	3.23	4.32
Link symbol rate MHz	N/A	216	243	432	432
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	3	3	3	3
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	2	2	2	2
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	1	1	1	1
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	312	356	226	312
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	1	1	1	1

PLL Value	Register and Field	eDP_R216	eDP_R243	eDP_R324	eDP_R432
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	52428	26214	39321	52428
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	4	2	3	4
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	5	5	5	5
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	0	0	0	0
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	50961	57331	38221	50961
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	65752	73971	49314	65752
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	0	0	0	0
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	0	0	0	0
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	0	0	0	0
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	2	2	1	1
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	1	1	1	1
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	0	0	0	0
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	2	2
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	0	0	2	0
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	4	4	4	4
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	19	20	20	19
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	65	65	65	65
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	127	127	127	127
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0

PLL Programming HDMI Values

HDMI does not use fixed link rates. The PLL programming for HDMI uses a table of values pre-calculated for certain frequencies and an algorithm to calculate values for other frequencies. The table is more accurate than the algorithm, so the table must be used for the frequencies it supports and the algorithm only used for other frequencies.

The algorithm must use floating point and complex functions such as square root to achieve enough accuracy. Software that is unable to calculate with that accuracy must use just the table and limit resolutions to the frequencies from the table.

SSC is always disabled for HDMI.

PLL Programming HDMI Table

The following table provides programming information for **HDMI link rates** with reference clock set to **100** MHz. All values are in decimal.

PLL Value	Register and Field	hdmi_251p75	hdmi_270	hdmi_742p5	hdmi_1p485	hdmi_5G94
Link bit rate Gbps	N/A	0.25175	0.270	0.7425	1.485	5.94
Link symbol rate MHz	N/A	25.175	27.0	74.25	148.5	594
ref_range[4:0]	SNPS_PHY_REF_CONTROL[ref_range]	3	3	3	3	3
ref_ana_mpllb_div[2:0]	SNPS_PHY_MPLLB_DIV2[dp_ref_clk_mpllb_div]	1	1	1	1	1
mpllb_ssc_en	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_en]	0	0	0	0	0
mpllb_div5_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div5_clk_en]	1	1	1	1	1
mpllb_multiplier[11:0]	SNPS_PHY_MPLLB_DIV2[dp_mpllb_multiplier]	128	140	86	86	86
mpllb_fracn_en	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_en]	1	1	1	1	1
mpllb_fracn_quot[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_quot]	36663	26214	26214	26214	26214
mpllb_fracn_rem[15:0]	SNPS_PHY_MPLLB_FRACN2[dp_mpllb_fracn_rem]	71	2	2	2	2
mpllb_fracn_den[15:0]	SNPS_PHY_MPLLB_FRACN1[dp_mpllb_fracn_den]	143	5	5	5	5
mpllb_ssc_up_spread	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_up_spread]	1	1	1	1	1
mpllb_ssc_peak[19:0]	SNPS_PHY_MPLLB_SSCEN[dp_mpllb_ssc_peak]	0	0	0	0	0
mpllb_ssc_stepsize[20:0]	SNPS_PHY_MPLLB_SSCSTEP[dp_mpllb_ssc_stepsize]	0	0	0	0	0
mpllb_div_clk_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_clk_en]	0	0	0	0	0
mpllb_div_multiplier[7:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_div_multiplier]	0	0	0	0	0
mpllb_hdmi_div[2:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_div]	1	1	1	1	1
mpllb_tx_clk_div[2:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_tx_clk_div]	5	5	3	2	0
mpllb_pmix_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_pmix_en]	1	1	1	1	1
mpllb_word_div2_en	SNPS_PHY_MPLLB_DIV[dp_mpllb_word_div2_en]	0	0	0	0	0
mpllb_ana_v2i[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_v2i]	2	2	2	2	2
mpllb_ana_freq_vco[1:0]	SNPS_PHY_MPLLB_DIV[dp_mpllb_freq_vco]	0	0	3	3	3
mpllb_ana_cp_int[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_int]	5	5	4	4	4
mpllb_ana_cp_prop[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop]	15	15	15	15	15

PLL Value	Register and Field	hdmi_251p75	hdmi_270	hdmi_742p5	hdmi_1p485	hdmi_5G94
mpllb_ana_cp_int_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_gs]	64	64	64	64	64
mpllb_ana_cp_prop_gs[6:0]	SNPS_PHY_MPLLB_CP[dp_mpllb_cp_prop_gs]	124	124	124	124	124
mpllb_hdmi_pixel_clk_div[1:0]	SNPS_PHY_MPLLB_DIV2[hdmi_mpllb_hdmi_pixel_clk_div]	0	0	0	0	0

Display Voltage Frequency Switching

Display Voltage and Frequency Switching (DVFS) is used to adjust the display voltage to match the display clock frequencies. If voltage is set too low, it will break functionality. If voltage is set too high, it will waste power. The voltage rail for display is shared by the entire system agent, so it has a large power impact.

When changing clock frequencies, graphics software must inform the power controller of the display voltage requirement. The power controller tracks requests from all users of the voltage rail and sets voltage to accommodate all the requests. The voltage requirements are separated into discrete levels aligned to the frequencies of several clocks used by display.

Voltage Requirement Selection

The voltage requirement is specified by selecting from several discrete voltage levels. The power controller will map these levels to the actual voltage values, which are usually determined on a part-by-part basis during manufacturing.

Only the CD clock and DDI clocks (max of all DDI clocks) are used to select the voltage levels. Other display clocks are either supported at all frequencies with the minimum voltage or have their voltage requirements accounted for by non-graphics software.

Voltage Level	CD clock MHz	Max DDI symbol clock MHz	Comment
0	307.2, 312, or lower	<=594	No 5k or 8k
1	324 or 326.4	<=594	These display clocks are mainly for resolutions under 5k with multiple displays enabled (cdclk PLL divided by 2 to allow for smooth increase to max cdclk)
2	556.8 or 552	>594	These display clocks are mainly for non-HDR 5K and 8k, or some cases with DP multistream or USB TypeC
3	652.8 or 648	>594	These display clocks are mainly for HDR 5k and 8k

If CD clock <= 312 MHz AND Max of DDI clocks <= 594 MHz, use level 0.

Else If CD clock <= 326.4 MHz AND Max of DDI clocks <= 594 MHz, use level 1.

Else If CD clock <= 556.8 AND Max of DDI clocks > 594 MHz, use level 2.

Voltage Requirement Selection
Else, use level 3. Note: For these calculations, disabling a clock is the same as switching it to the lowest frequency. The sequences below are used when changing CD clock frequency and when changing DDI clock frequency during a mode set.

The following sequences are not used on their own. They are called as part of other sequences which change the display clock frequencies.

Sequence Before Frequency Change

- This sequence requests the power controller to raise voltage to the maximum.

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low
    - Bits 1:0 = 0x3
    - If software knows that CDCLK frequency is increasing: Bits 25:16 = CEILING[Upcoming increased CDCLK frequency MHz]
    - If software knows that CDCLK frequency is decreasing or not changing (such as when only DDI clock is updating): Bits 25:16 = CEILING[Current CDCLK frequency MHz]
    - If software does not know the direction of CDCLK frequency change: Bits 25:16 = 0x28D (652.8 MHz maximum frequency CDCLK)
        - This increases power usage until the Sequence After Frequency Change sets the correct frequency
    - Bit 27 = 1 (CDCLK frequency update valid)
    - Other bits all 0s
3. Write GT Driver Mailbox Data High = 0x00000000.
4. Write GT Driver Mailbox Interface = 0x80000007.
5. Poll GT Driver Mailbox Interface for Run/Busy indication cleared (bit 31 = 0).
    - Timeout after 150 us. Do not change CD clock frequency if there is a timeout.
6. Read GT Driver Mailbox Data Low, if bit 0 is 0x1, continue, else wait at least 500us for Pcode to process the command, then go to step 2.
    - If the condition in step 6 is not satisfied after cycling through steps 2-6 for 3 ms (typically <200 us), timeout and fail and do not change clock frequency.

Sequence After Frequency Change

- This sequence requests the power controller to set the voltage to the selected level. The power controller may choose to keep the voltage higher to accommodate other users outside of display.

1. Write GT Driver Mailbox Data Low with the voltage level selection, following the table above.
    - Bits 1:0 with the voltage level selection, following the table above
        - For Level 0, write 0x0.

- For Level 1, write 0x1.
- For Level 2, write 0x2.
- For Level 3, write 0x3.

- Bits 25:16 = CEILING[Current CDCLK frequency MHz]
  - Disabled CDCLK PLL = 38.4 MHz, 39 decimal, 27 hex
- Bit 27 = 1 (CDCLK frequency update valid)
- Other bits all 0s

2. Write GT Driver Mailbox Data High = 0x00000000.
3. Write GT Driver Mailbox Interface = 0x80000007.
   - There is no need for display software to wait for the voltage to adjust.

Sequence for Pipe Count Change

- This sequence is run before enabling a pipe power well and after disabling a pipe power well to request the power controller to adjust the power estimate.

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low
   - Bits 1:0 = the voltage level selection from the last time the Sequence After Frequency Change, above, was run.
   - If enabling pipe power well, Bits 30:28 = number of pipe power wells that will be enabled after the enabling completes (the upcoming pipe count)
   - If disabling pipe power well, Bits 30:28 = number of pipe power wells that are still enabled after the disabling completes (the current pipe count)
     - Use 0 pipe count if all pipes are disabled
   - Bit 31 = 1 (Pipe count update valid)
   - Other bits all 0s
3. Write GT Driver Mailbox Data High = 0x00000000.
4. Write GT Driver Mailbox Interface = 0x80000007.
5. Poll GT Driver Mailbox Interface for Run/Busy indication cleared (bit 31 = 0).
   - Timeout after 150 us. Do not change pipe count if there is a timeout.
6. Read GT Driver Mailbox Data Low, if bit 0 is 0x1, continue, else wait at least 500us for Pcode to process the command, then go to step 2.
   - If the condition in step 6 is not satisfied after cycling through steps 2-6 for 3 ms (typically <200 us), timeout and fail and do not change pipe count.

Pcode will default to minimum pipe count and CDCLK frequency at boot, so boot software (GOP for example) needs to program the pipe count and CDCLK frequency when enabling display output.

Pipe count change and frequency change use the same GT Driver Mailbox command with different data bits updated. If desired, the two can be combined by merging both sets of data bits in one round of mailbox programming.

Sequences for Changing CD Clock Frequency

Restrictions
The CD clock frequency impacts the maximum supported pixel rate and display watermark programming. The CD clock frequency must be at least twice the frequency of the Azalia BCLK.

Sequence for Changing CD Clock Frequency

1. Unless changing only the CD2X Divider or using other method that does not require the PLL to be disabled when changing frequency, disable all display engine functions using the full mode set disable sequence on all pipes, ports, and planes.
   - Includes Global Time Code
   - Display power wells may be left enabled
2. Follow the Display Voltage Frequency Switching - Sequence Before Frequency Change
3. Enable or change the frequency of CD clock
   a. If enabling CDCLK PLL
      i. Write CDCLK_PLL_ENABLE with the PLL ratio, but not yet enabling it.
      ii. Set CDCLK_PLL_ENABLE PLL Enable
      iii. Poll CDCLK_PLL_ENABLE for PLL lock
      iv. Timeout and fail if not locked after 200 us
      v. Write CDCLK_SQUASH_CTL with the Squash Waveform and Squash Enable value to match the desired CD clock frequency.
      vi. Write CDCLK_CTL with the CD2X Divider selection and CD Frequency Decimal value to match the desired CD clock frequency
   b. If disabling CDCLK PLL
      i. Clear CDCLK_PLL_ENABLE PLL Enable
      ii. Poll CDCLK_PLL_ENABLE for PLL unlocked
      iii. Timeout and fail if not unlocked after 200 us
   c. If changing the CDCLK PLL frequency
      i. Follow steps above for disabling CDCLK PLL.
      ii.Follow steps above for enabling CDCLK PLL, using the new PLL ratio.
   d. If changing the CDCLK PLL frequency without squashing
- This is for changing PLL frequency with the PLL disabled, which requires display functions to be disabled.

i. Follow steps above for disabling CDCLK PLL.

ii. Follow steps above for enabling CDCLK PLL, using the new PLL ratio.

   e. If changing the CDCLK PLL frequency with squashing
- This is for changing PLL frequency while the PLL is enabled and display functions are active. This is used for adjusting frequency to match resolution requirements in situations without a mode set, such as enabling/disabling an external display when internal display is already enabled.

i. Temporarily disable PSR1, PSR2, and GTC.
ii. Wait for disable status from those functions.
iii. Wait for any pending Aux transactions to complete, and do not start any new Aux transaction.
iv. Write CDCLK_SQUASH_CTL with the desired squash waveform.
v. Write CDCLK_CTL with the CD2X Divider selection and CD Frequency Decimal value to match the desired CD clock frequency.
vi. Re-enable the temporarily disabled functions and resume using Aux.

f. If changing only the CD2X Divider
    i. Write CDCLK_CTL with the CD2X Pipe selection, CD2X Divider selection, and CD Frequency Decimal value to match the desired CD clock frequency
    ii. If pipe is enabled, wait for start of vertical blank for change to take effect

4. Follow the Display Voltage Frequency Switching - Sequence After Frequency Change
5. Update programming of functions that use the CD clock frequency. If these features are not currently enabled, the programming can be delayed to when they are enabled.
    i. Utility pin backlight frequency and duty cycle in the BLC_PWM_DATA register.

Resets

The north and south display engines are reset by PCI Function Level Resets (FLR) and the chip level resets.

The south display engine runs panel power down sequencing (if configured to do so by NDE_RSTWRN_OPT) before resetting.

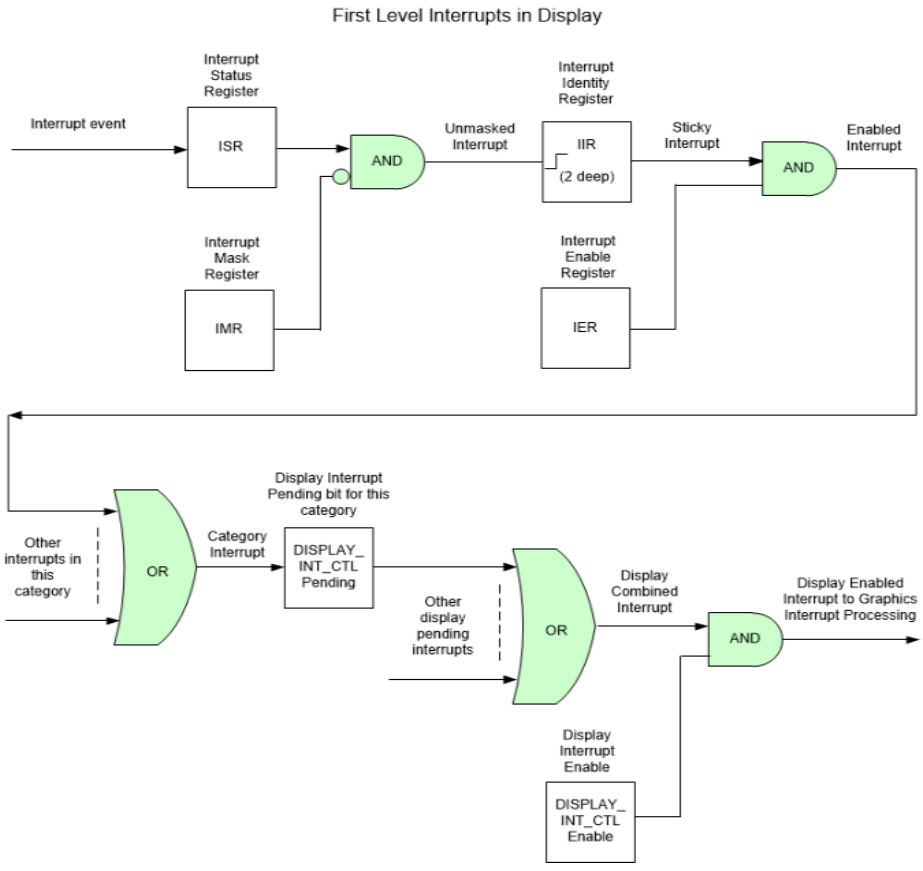
NDE_RSTWRN_OPT

Shared Functions

Display Interrupts

Registers
DISPLAY_INT_CTL
DE Pipe Interrupt Definition
DE Port Interrupt Definition
DE Misc Interrupt Definition
Audio Codec Interrupt Definition
INTERRUPT Structure
DE HPD Interrupt Definition
Graphics Primary Interrupt
Graphics Interrupt Introduction
Gdie Interrupt and Errors
Discrete interrupt handling

Interrupt Flow



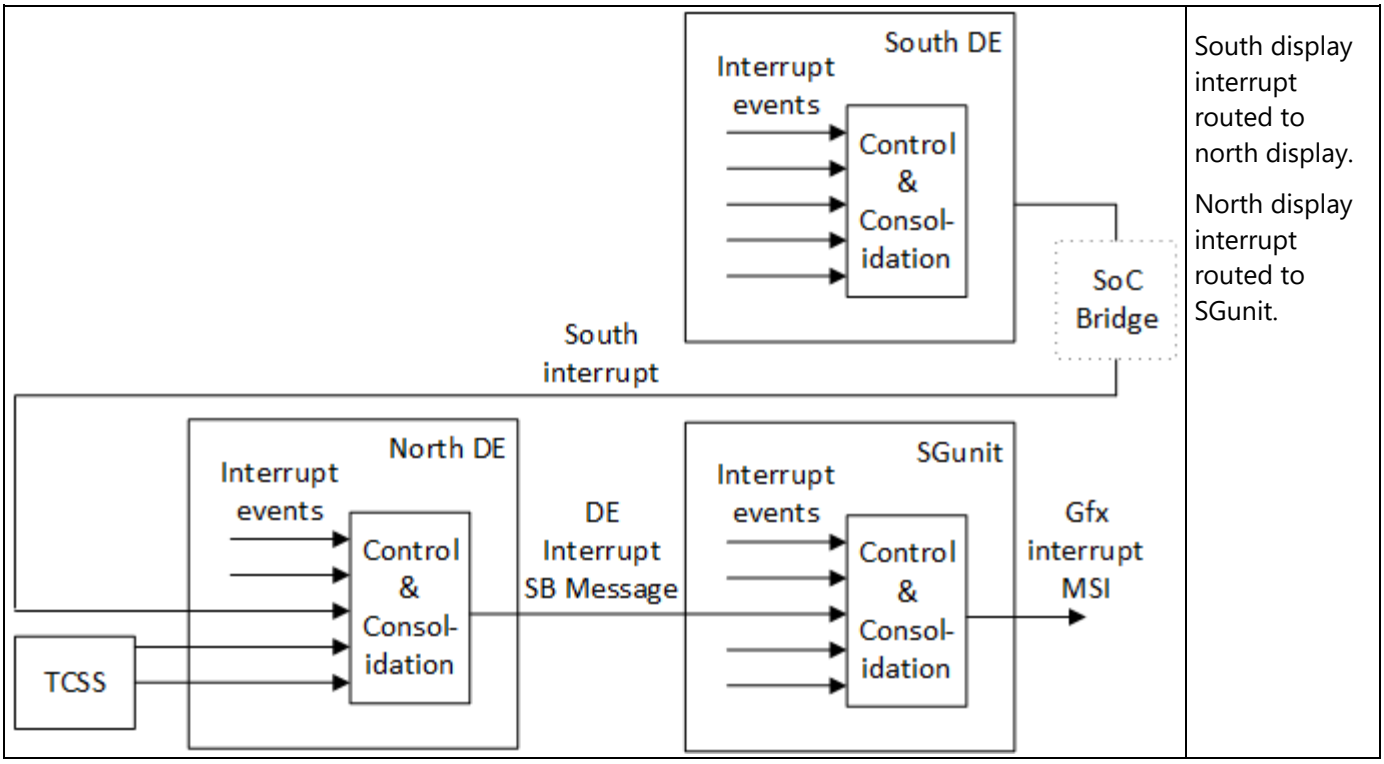
1. For every first level interrupt bit
    a. The interrupt event comes into the interrupt handling logic.
        - There may be more levels of interrupt handling behind each event. For example, the PCH Display interrupt event is the result of the SDE interrupt registers.
    b. The interrupt event goes to the Interrupt Status Register (ISR) where live status can be read back.
        - The live status is mainly useful for hotplug interrupts where it gives the live state of the hotplug line.
        - The live status is not useful for pulse interrupt events due to the short period that the status will be present.
    c. The interrupt event is ANDed with the inverted Interrupt Mask Register (IMR) to create the unmasked interrupt.
    d. The unmasked interrupt rising edge sets the sticky bit in the Interrupt Identity Register (IIR).
        - The IIR can be cleared by writing a 1 to it.
        - The IIR can queue up to two interrupt events. When the IIR is cleared, it will set itself again if a second event was stored.
    e. The sticky interrupt is ANDed with the Interrupt Enable Register (IER) to create the enabled interrupt.
2. All enabled interrupts are then ORed by category (Pipe, Audio, etc.) to create a category interrupt which is then visible in one of the Display Interrupt Control Register (DISPLAY_INT_CTL) pending category bits.
3. All pending interrupts are then ORed to create the display combined interrupt.
4. The display combined interrupt is ANDed with the Display Interrupt Enable (DISPLAY_INT_CTL Bit 31) to create the display enabled interrupt.
5. The display enabled interrupt then goes to graphics interrupt processing before eventually creating an interrupt message which will reach the OS.

Interrupt Service Routine

1. Read graphics primary interrupt register to find a display interrupt pending.
2. Write graphics primary interrupt register to clear display interrupt.
3. Disable Display Interrupt Control (can be done here, or anywhere before the final step)
    - Clear bit 31 of DISPLAY_INT_CTL.
    - This de-asserts the display enabled interrupt and is required in order for the final step to cause a new assertion when all interrupts are not cleared during the service routine.
4. Find the category of interrupt that is pending
    - Read DISPLAY_INT_CTL and record which interrupt pending category bits are set.
5. Find the source(s) of the interrupt and clear the Interrupt Identity bits (IIR)
    - Read the IIR associated with each pending interrupt category, record which bits are set, then write back 1s to clear the bits that are set.

- There can be up to 2 interrupts recorded per source, requiring multiple writes to the IIR to fully clear.

6. Process the interrupt(s) that had bits set in the IIRs.
7. Optionally go to step 4 again to check for any new display interrupts that have happened during previous steps
    - This is optional since any new interrupts will also be found when they trigger the interrupt service routine again after the final step and then re-enter from the start of the routine.
8. Re-enable Display Interrupt Control
    - Set bit 31 of DISPLAY_INT_CTL.
    - If interrupts were not fully cleared before this point, then the display enabled interrupt will re-assert and there will be a new display interrupt in graphics primary interrupt register.



MBus

Mbus is an internal ring type bus. Display high priority clients reside on this bus. The introduction of a ring architecture allows for easy addition/deletion of clients as well as improves Bandwidth. The various "boxes" are different types of tap points where display clients are plugged in.

MBus
MBUS_ABOX_CTL
MBUS_BBOX_CTL
MBUS_DBOX_CTL
MBUS_UBOX_CTL

MBus programming during display Initialization

Program credits in MBUS_ABOX_CTL0, MBUS_ABOX_CTL1

The MBus credits should be setup once with the following default values during the display initialization.

MBUS_ABOX_CTL for this pipe -> BT Credits Pool1 = 16

MBUS_ABOX_CTL for this pipe -> BT Credits Pool2 = 16

MBUS_ABOX_CTL for this pipe -> B Credits = 1

MBUS_ABOX_CTL for this pipe -> BW Credits = 1

DBUF_CTL for this pipe -> Tracker State Service = 8

The following programming must be done when enabling each pipe as a part of *configure other pipe settings* in the *Enable Planes, Pipe, and Transcoders* sequence.

MBUS_DBOX_CTL for this pipe -> A Credits = 2

MBUS_DBOX_CTL for this pipe -> BW Credits = 2

MBUS_DBOX_CTL for this pipe -> B Credits = 12

Fuses and Straps

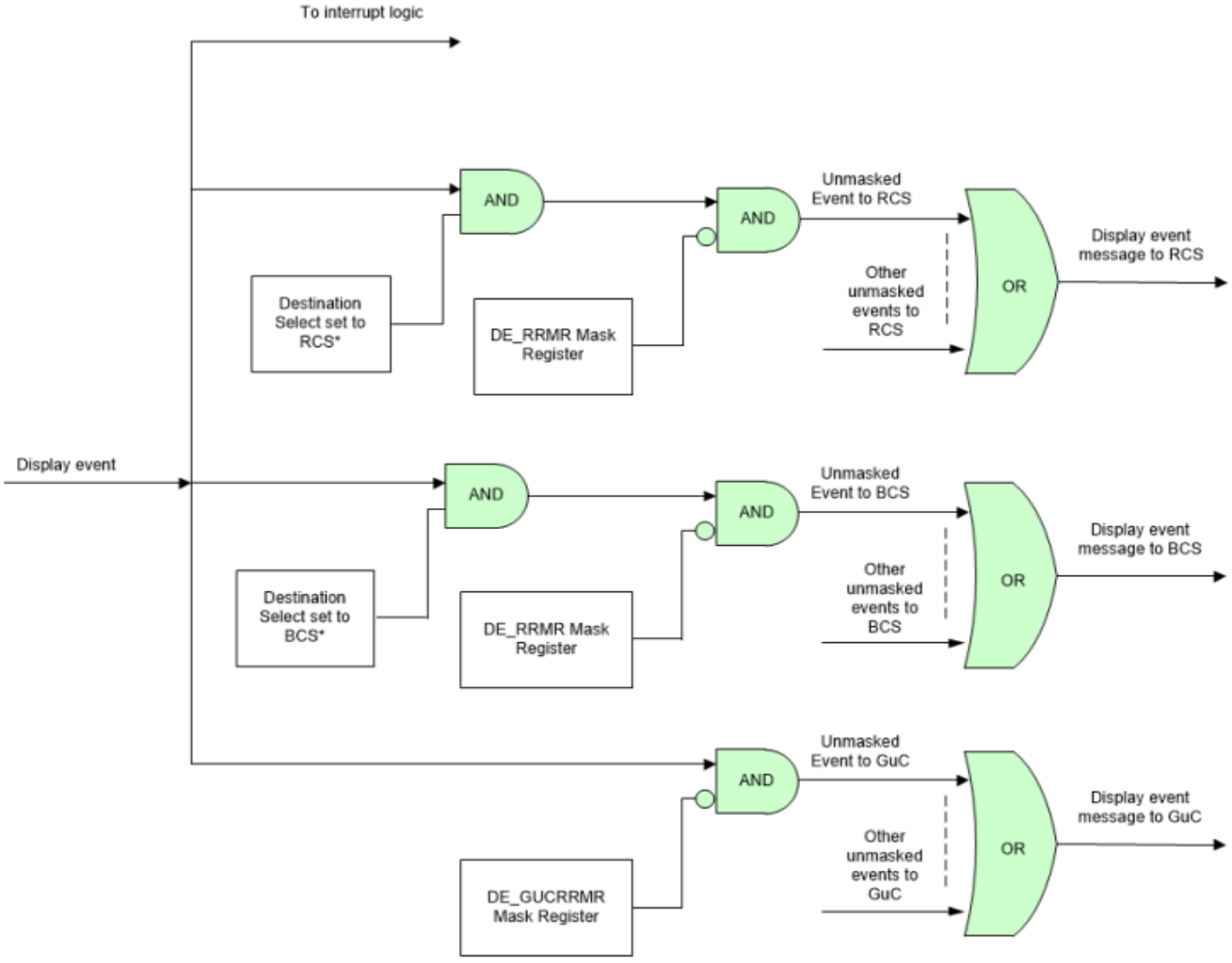
FUSE_STATUS

DFSM

DSSM

Render Response

Structure
Display Engine Render Response Message Definition
DE_RRMR
DE_RRMR_DW1
DE_RRMR_DW2
DE_RR_DEST



*For Vblank events, destination select is configured in DE_RR_DEST to direct the event to RCS, BCS, or both at once.
For MMIO triggered flip done events, destination select is configured in PLANE_SURF Ring Flip Source to direct the event to either RCS or BCS.
For ring (AKA message) triggered flip done events, destination select is configured automatically by decoding whether the flip came from BCS or RCS.

GuC supports some additional events beyond what RCS and BCS support. In that case the event can feed into the GuC masking directly without branching to RCS and BCS.

Older documentation refers to RCS as CS.

Arbiter

Arbiter
ARB_HP_CTL ARB_LP_CTL
PIPE_ARB_CTL
BW_BUDDY0_CTL
BW_BUDDY_CTL0 BW_BUDDY_CTL1 BW_BUDDY_PAGE_MASK0 BW_BUDDY_PAGE_MASK1

Note: BW Buddy registers are present but not used in these discrete projects.

Data Buffer

DBUF_CTL

DBUF_STATUS

DBUF_ECC_STAT

DBUF_CTL2

See the Display Buffer Programming section for information on how to allocate the buffer to planes.

Miscellaneous Shared Functions

UTIL_PIN_CTL

UTIL_PIN_BUF_CTL

UTIL2_PIN_CTL

UTIL2_PIN_BUF_CTL

The utility pin(s) can be used for various functions, including MIPI DSI TE and genlock, on SoCs supporting those functions. The pins must be configured for those functions as part of the function enable and disable sequences.

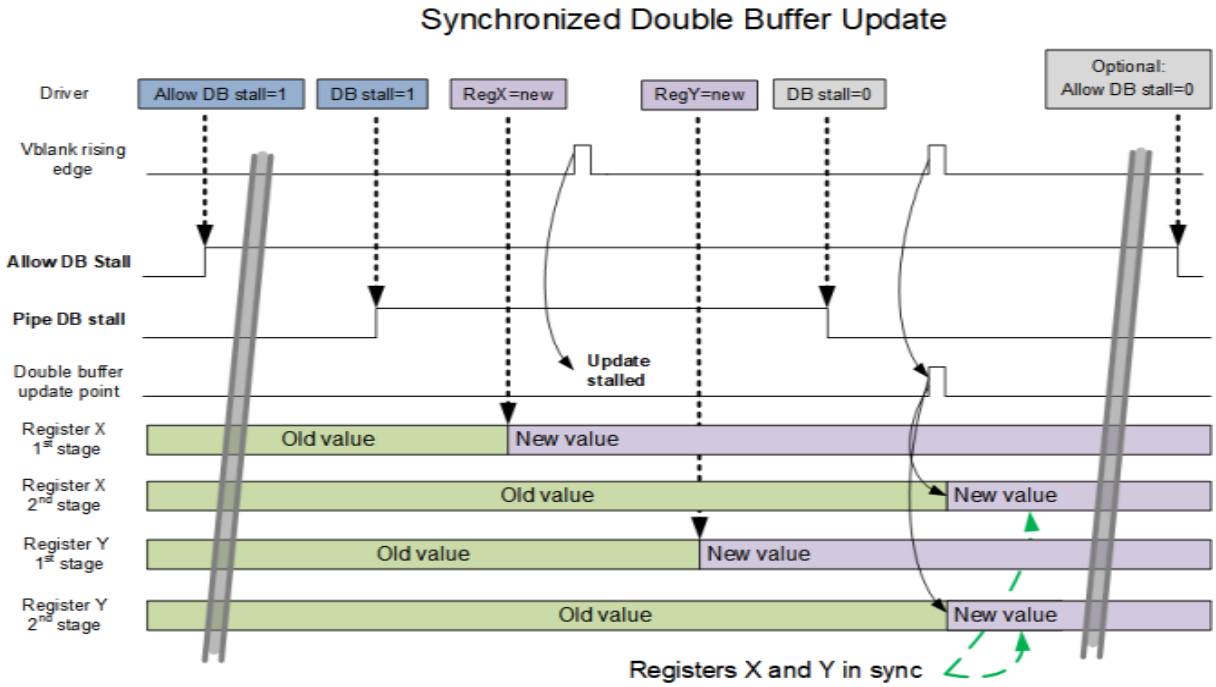
AUDIO_PIN_BUF_CTL

Double Buffer Control

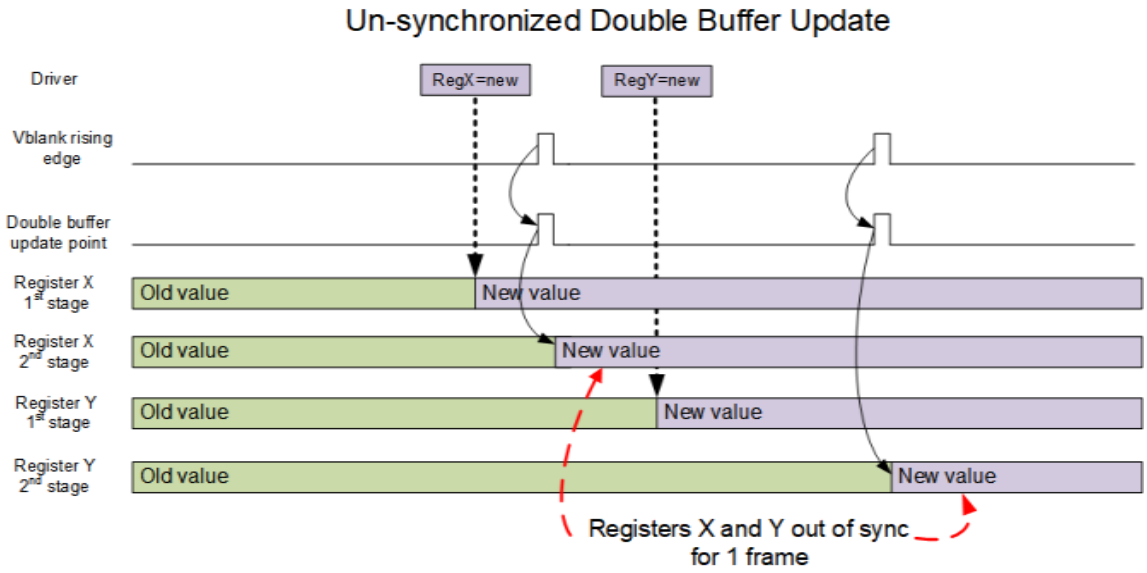
Registers
DOUBLE_BUFFER_CTL
PIPE_DB_CTL

Double Buffer Stalling

Double buffer stalling is used to synchronize register updates across multiple display resources for an atomic update. While stalled, double buffer registers can be written with new values, updating the 1st stage buffer, but the double buffering will not activate to transfer those values to the 2nd stage (live/working value) buffer to be used by the hardware resource. This allows for multiple register programming to cross the double buffer update point and keep the registers in sync.



Without double buffer stalling, updates to multiple registers can become un-synchronized if the double buffer update event (typically vblank rising edge) happens during the programming. Then some registers will have double buffer updates a frame before other registers and the screen contents will be incorrect.



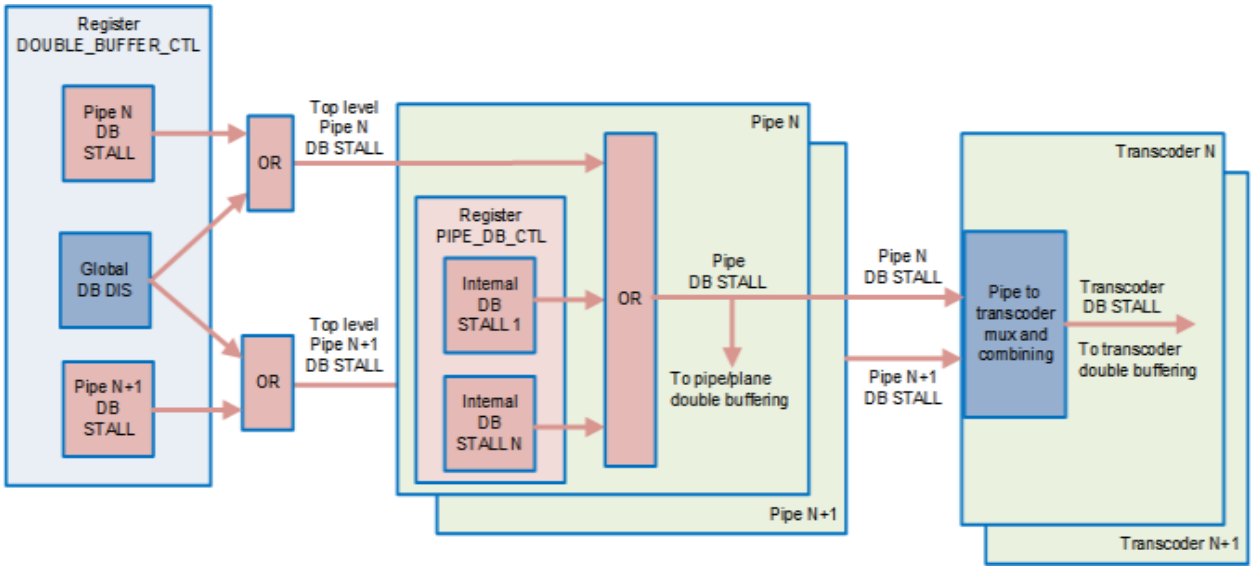
Pipe Double Buffer Stall

The double buffer stalling is controlled by the pipe double buffer stall signal. This signal is driven by multiple registers bits to support different usages.

- Legacy usage
  - DOUBLE_BUFFER_CTL Global Double Buffer Disable register bit is used to stall double buffering in all pipes and transcoders.
  - This is supported for legacy software and will be removed in future projects.
  - The Global Double Buffer Disable register bit is used alone and not combined with the newer pipe double buffer stalls.
- Multi-pipe usage
  - Multiple DOUBLE_BUFFER_CTL Pipe DB Stall register bits are set and cleared simultaneously to stall double buffering in multiple pipes and attached transcoders.
  - This is used for synchronizing register updates across multiple pipes and transcoders, which may be useful in some pipe joining, port sync, or genlock cases.
  - DSB cannot access DOUBLE_BUFFER_CTL.
- Independent pipe usage
  - PIPE_DB_CTL Internal DB Stall registers bits are used to stall double buffering in a single pipe and attached transcoder.
  - PIPE_DB_CTL has multiple Internal DB Stall bits in separate bytes to allow concurrent programming from DSBs and with driver MMIO.
  - DSB can access PIPE_DB_CTL.
  - There may be cases where multi-pipe usage and independent pipe usage are combined.

Each pipe's double buffer stall signal will assert if either DOUBLE_BUFFER_CTL Global Double Buffer Disable is set, DOUBLE_BUFFER_CTL Pipe DB Stall <this pipe> is set, or any PIPE_DB_CTL_<this pipe> Internal DB Stall <#> bit is set.

The pipe double buffer stall is routed to the attached transcoder to stall double buffering in the transcoder. If multiple pipes are attached to a single transcoder, such as for pipe joining, then the pipe double buffer stalls are combined by OR'ing so that either pipe can stall the double buffer in the transcoder.



Allow Double Buffer Stall

The double buffer stalling is also controlled by the Allow DB Stall register bit each resource has in one of its control registers. The Allow DB Stall is useful if certain resources need to update freely and not be tied into the synchronizing of other resources, such as if cursor update needs to happen as soon as possible while some plane updates are being stalled to synchronize together.

The double buffering for a particular resource will only be stalled if that resource's Allow DB Stall is set and pipe double buffer stall is set.

Details

This only stalls the double buffer update for periodic events, like the start of vertical blank. It does not change the behavior for constant events, like pipe not enabled.

Double buffer stalling does not stall asynchronous flips initiated by MMIO or command streamers.

Synchronous flips will not complete or give the flip done indication while double buffering is stalled for a plane. They will complete and give the flip done at the next double buffer update point after the double buffering is no longer stalled.

Sequence for synchronizing double buffer updates

1. If not already set, set the Allow DB Stall field for each resource to be synchronized together.
2. Set the register bit to set the pipe double buffer stall signal. Double buffering is now stalled.
    - Register bit depends on usage.
3. Program the registers that need to be synchronized together.
4. Clear the register to clear the pipe double buffer stall signal. Double buffering is now un-stalled. Any pending updates will take place at the next periodic update event.
5. Depending on configuration, variable refresh rate and PSR require extra programming to trigger a frame update after the registers have been updated. See the VRR and PSR programming pages.

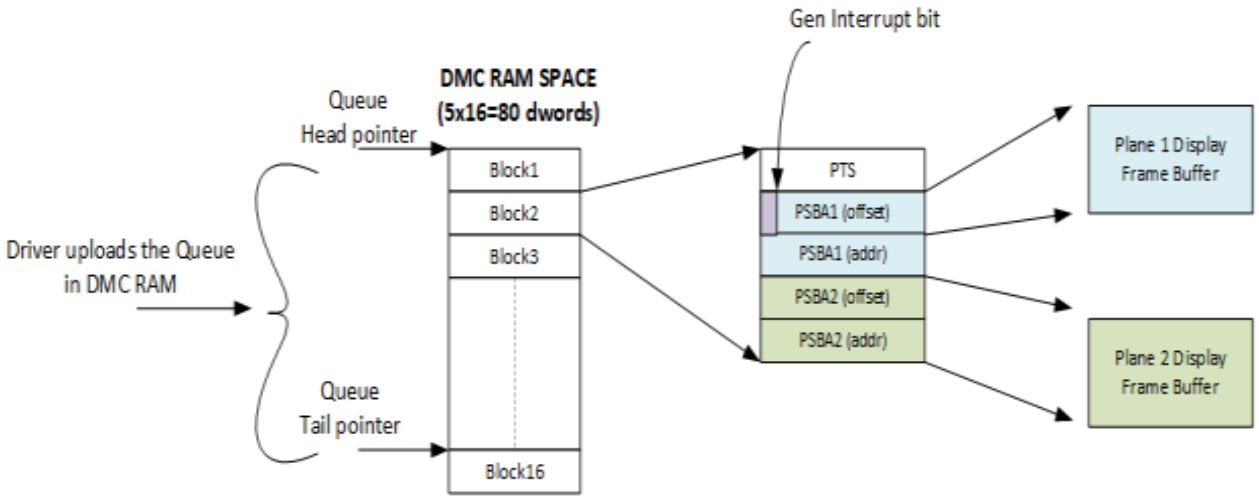
6. If a resource no longer needs to be synchronized, clear the Allow DB Stall field for that resource. If the resource will continue to be synchronized, the field can remain set.
    - It is not necessary to set and clear the Allow DBStall around every register update. Allow DB Stall only has to change if a resource's requirements for synchronization are changed.

Central Power

Simple Flip Queue Programming Sequences

Below is diagram of the Simple flip queue programming concept. There are 16 entries of DMC RAM space defined for each queue. There is only one queue in HW per pipe. Each of these entries contain information of the flip programming. It holds information on the time at which the HW must execute the flip.

If flip queue will not be used, follow the Disable Sequence below to disable it.



This page describes the Simple flip queue Programming Sequences and offsets to be used for the programming.

There are 5 registers per entry that needs to be programmed by the software.

- Presentation Time stamp: If flip queue is triggered and current time stamp in the HW exceeds or is equal to this value then DMC FW executes the contents of the entry.
- Plane 1 surface base address offset (MMIO offset for PLANE_SURF). Set top bit of the register to enable interrupt generation when DMC is done processing the flip.
- Plane 1 Surface base address content (Address that points to memory buffer location of Display frame)
- Plane 2 surface base address offset (MMIO offset for PLANE_SURF)
- Plane 2 Surface base address content (Address that points to memory buffer location of Display frame)

The following registers are used for each pipe per flip queue.

Pipe	Offset Start	Offset End	Queue
Pipe A	0x90008	0x90147	Q1
Pipe B	0x98008	0x98147	Q1
Pipe C	0x52008	0x52147	Q1
Pipe D	0x59008	0x59147	Q1

SW must load a queue of flips in the MMIO offsets provided

1. Queue consists of a batch of flips with Presentation Time stamps, Plane base addresses of two planes. SW will upload the Flip queue entries and update PIPEDMC_FQ1_HP and PIPEDMC_FQ1_TP.
2. SW will disable the VBI and the Flip done interrupt to get power savings.
3. Program the Event control to select the trigger event.
4. Enable the FQ in the Flip queue control register.

Software must program the registers in pipe DMC for generating the event for the Flip queue checking.

For fixed refresh rate, setup trigger on scanline range

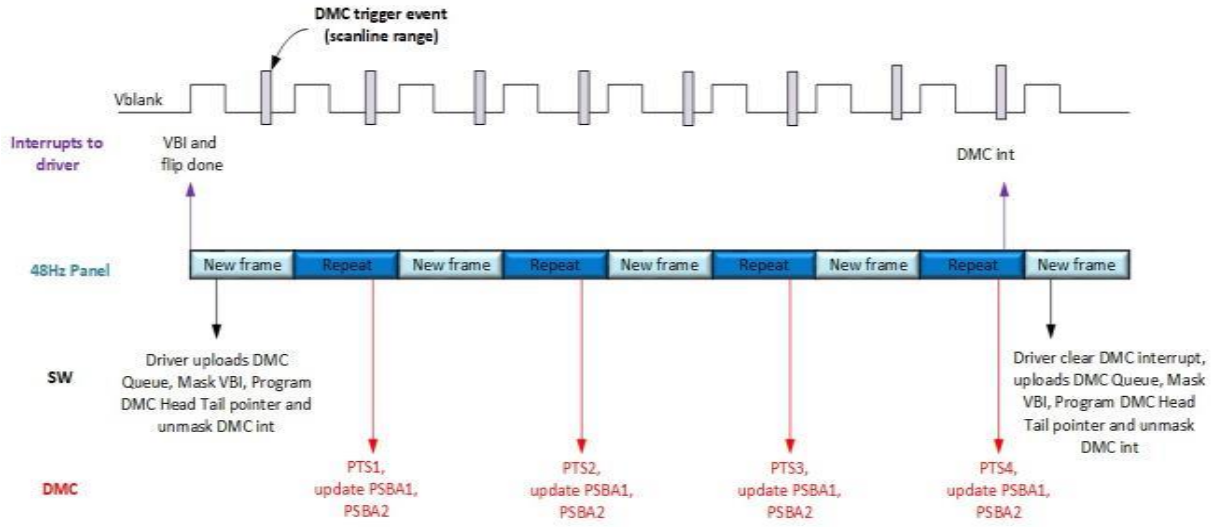
- Set bit 31 of the PIPEDMC_SCANLINECMPLOWER register
- Program the lower limit of the line range in the PIPEDMC_SCANLINECMPLOWER register
- Program the upper limit of the line range in the PIPEDMC_SCANLINECMPUPPER register
- Program the PIPEDMC_FQ_CTRL register to enable the Flip queue.
- Program the trigger event to select scanline_range event in the PIPEDMC_EVT_CTL and PIPEDMC_Event_ID. Refer to the DMC guide for programming these registers.

For variable refresh rate, setup trigger on 1KHz timer

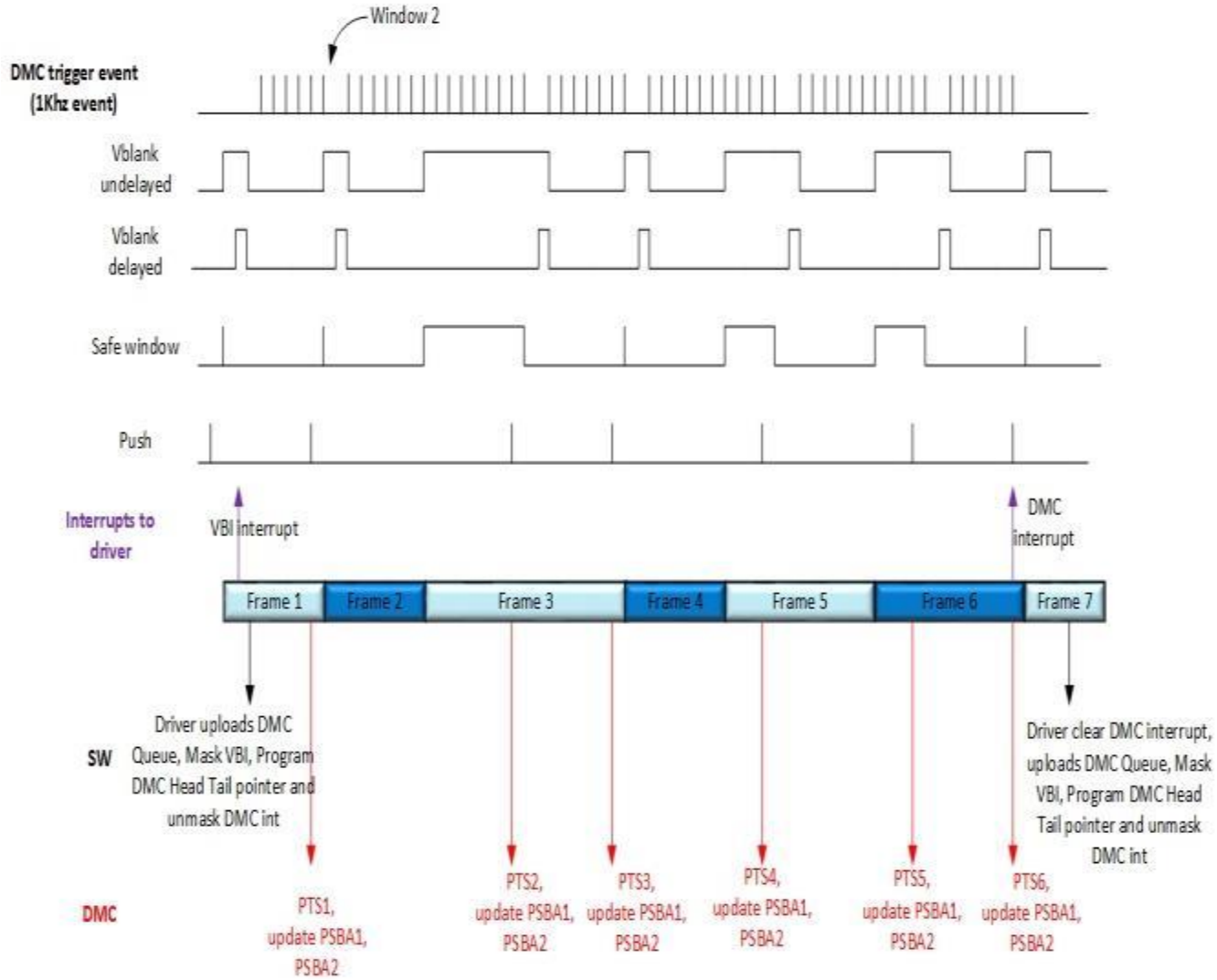
- Program the trigger event to select 1KHz event in the PIPEDMC_EVT_CTL and PIPEDMC_Event_ID. Refer to the DMC guide for programming these registers.

Below is the timing diagram of the flip queue. PTS values are shown in the blocks.

Fixed refresh with scanline range trigger.



Variable refresh with 1KHz timer trigger.



Queue Update Sequence:

1. Read the current head pointer to not overload the unprocessed entries.
2. Upload new entries.
3. Update the tailpointer.

Queue purge sequence:

1. Set the FQ pre-emption bit.
2. Read the status of the busy bit.
3. If not busy, then clear the FQ enable and reset the Head and tail pointers.

Enable and Trigger Event Selection Sequence:

For FQ with fixed refresh:

1. Program the trigger event to scanline range.
2. Program the desired scanline range.
3. Upload the queue entries.
4. Program the FQ HTP.
5. Enable FQ.

For FQ with variable refresh:

1. Program the trigger event to 1KHz.
2. Upload the queue entries.
3. Program the FQ HTP.
4. Enable FQ.

To change the triggers events between refresh cases:

1. Disable DC6v
2. Set the preemption.
3. Wait for DMC busy to clear.
4. Disable FQ and clear HTP.
5. Follow the trigger event selection sequence.

Disable Sequence

1. Disable flip queue.
2. Program PIPEDMC_EVT_CTL_2_<All pipes> = 0x00030100
    a) The control setting selects a null event, preventing DMC from being triggered to run any program. If control bit 31 was set before this, it will remain set because it can only be cleared by hardware after an even is triggered.
3. Program PIPEDMC_EVT_HTP_2_<All pipes> = 0x00000000

SAGV

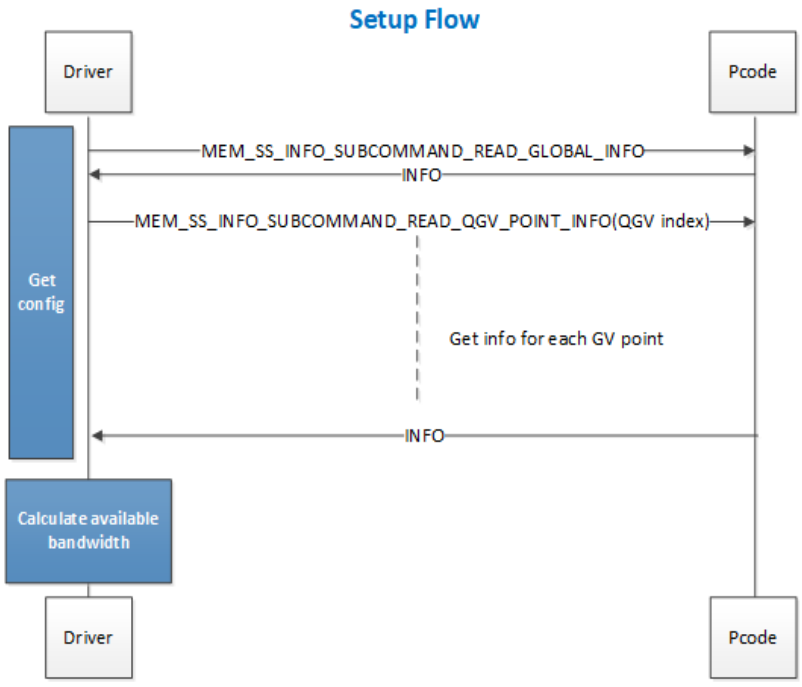
System Agent Geyserville (SAGV) dynamically adjusts the system agent voltage and clock frequencies depending on power and performance requirements. SAGV impacts display engine in two ways. SAGV point selections can limit the system memory bandwidth to display. SAGV transitions can temporarily block display engine access to system memory. Display software must restrict the SAGV point selection to control the bandwidth availability and setup watermarks to tolerate the temporary memory block.

SAGV Point Selection

Setup

Find the memory bandwidth availability for display. The results can be cached so that the flow does not have to be rerun.

The driver does not have to use the mailbox if it has other ways of finding the memory and GV point info.



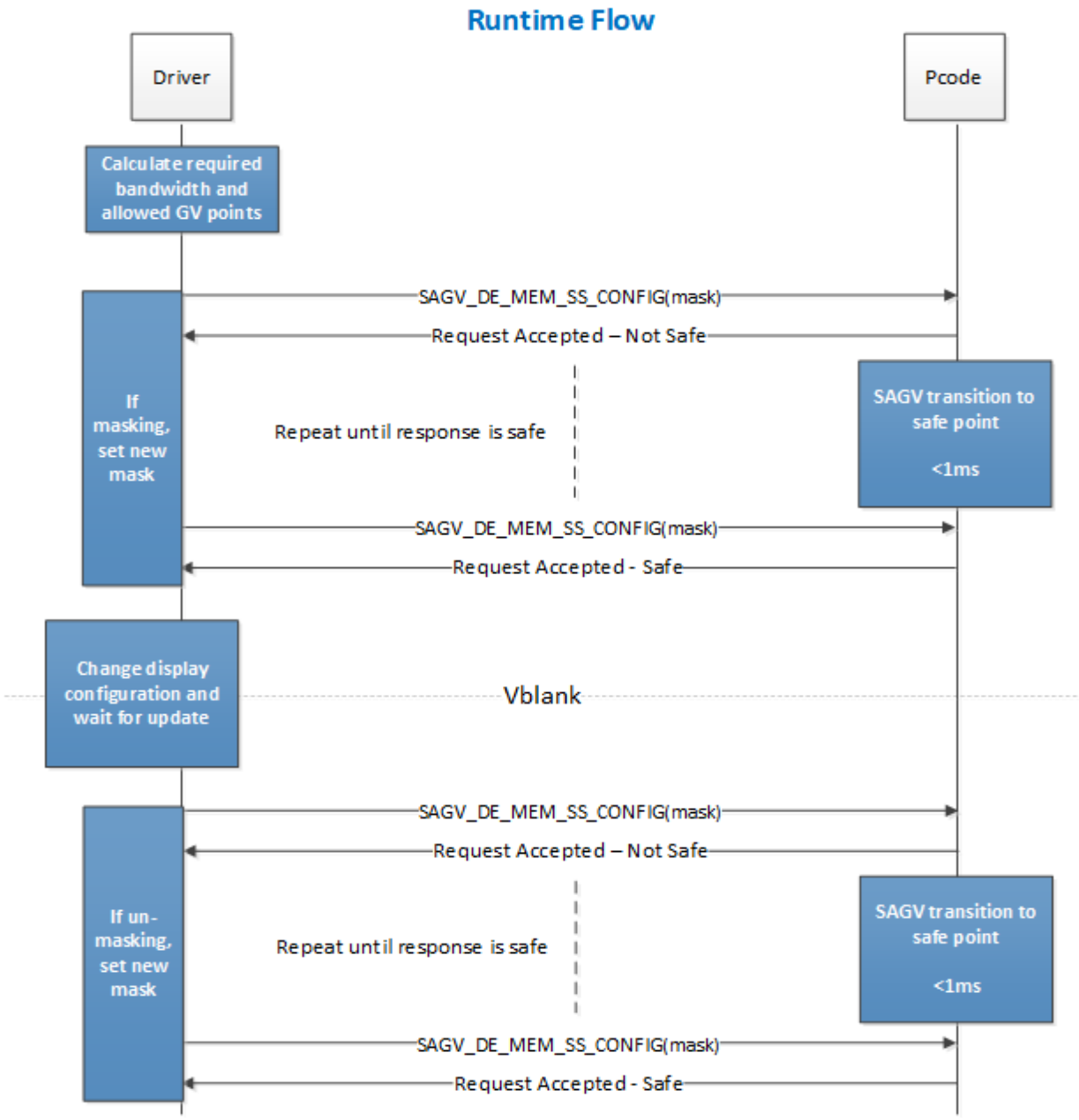
1. Driver reads memory subsystem (MEM SS, MemSS) configuration information from Pcode using the GT Pcode mailbox command MAILBOX_GTDRIVER_CMD_MEM_SS_INFO, first reading the global info and number of GV points, then reading GV info for each GV point.
    - Described in the Mailbox Commands section
2. Driver uses the mailbox info to calculate the available memory bandwidth for each GV point and number of enabled planes.
    - Described in the Bandwidth Restrictions chapter, Available Memory Bandwidth Calculation section.

There are cases where the highest numbered GV point allows less bandwidth for display than a lower point.

Overclocking reprogramming of memory parameters dynamically (system running without a reboot when changes are applied) can result in a display underrun since the display driver will not know that the configuration has changed.

Runtime

Adjust the allowed GV points to match display configuration changes.



1. Before changing display configuration in a way that impacts required bandwidth or number of planes, driver calculates the new bandwidth requirement.
    - Described in the Bandwidth Restrictions chapter, Required Memory Bandwidth Calculation section.
2. Driver finds which GV points supply enough available bandwidth to meet the required bandwidth and which GV points do not supply enough and must be restricted (masked off).
    - At least one GV point of each type must always remain unmasked. The GV point of each type providing the highest bandwidth for display must always remain unmasked.

- Each cycle through this sequence can only mask or unmask points of a GV type, not do both for a single GV type. It is allowed for one GV type to unmask while another masks, then one cycle through this sequence will mask one type of GV and the unmask another type of GV.
- If no GV point will provide enough bandwidth, then the display configuration must be constrained to fit within the available bandwidth.
- If driver has lost track of what was previously masked off, it can mask off all but the highest bandwidth point of each type to get to a known safe setting, then cycle through the full sequence again to set the correct mask.
- To disable SAGV when watermarks do not meet the SAGV block time requirement, mask off all the QGV points except for the point providing the highest bandwidth for display. To re-enable SAGV, unmask all of the QGV points that meet the bandwidth requirements for display. See the Watermark Calculations section for more info.

3. If masking GV points
    a) Driver uses GT Pcode mailbox command MAILBOX_GTDRIVER_CMD_SAGV_DE_MEM_SS_CONFIG to send Pcode the bitmask of restricted points and polls by repeatedly issuing the command with the same bitmask until receiving the acknowledge with point safe for each type, hitting a timeout after 1 millisecond, or receiving the error code.
        - Command described in the Mailbox Commands section
        - Pcode adjusts GV
            i. Pcode performs needed changes to ensure that all future transitions will be performed to the allowed states only.
            ii. If pcode currently resides in a point which is restricted, it issues a GV transition to an allowed state.
            iii. Pcode responds to the mailbox with point safe.
        - If only one type of GV is masking, keep the mask unchanged for other types.
4. Driver changes the display configuration
5. If unmasking GV points
    a) Driver waits for the configuration to be updated (typically at the next vertical blank). Driver can wait even longer, but should eventually complete unmasking in order to let GV move between points to give more optimal power and performance.
    b) Driver uses GT Pcode mailbox command MAILBOX_GTDRIVER_CMD_SAGV_DE_MEM_SS_CONFIG to send Pcode the bitmask of restricted points and polls by repeatedly issuing the command with the same bitmask until receiving the acknowledge with point safe for each type, hitting a timeout after 1 millisecond, or receiving the error code.
        - Command described in the Mailbox Commands section
        - Pcode adjusts GV
            i. Pcode performs needed changes to ensure that all future transitions will be performed to the allowed states only.

ii. If pcode currently resides in a point which is restricted, it issues a GV transition to an allowed state.

iii. Pcode responds to the mailbox with point safe.

- If only one type of GV is unmasking, keep the mask unchanged for other types.

All points are unrestricted by default until driver requests otherwise or BIOS disables SAGV.

Pcode will reset the bitmask from display on the graphics device function level reset (FLR).

The BIOS mailbox for SAGV gets precedence over Display Mailbox for GV request.

Display driver must allow at least one GV point of each type at all times. Driver must not mask off all the points of any type.

Display driver must wait for the previous GV point restriction to finish before starting a new restriction.

Legacy Behavior

The legacy command MAILBOX_GTDRIVER_CMD_DE_LTR_SETTING has the effect of masking off all but the highest numbered QGV point. the command continues to be supported and can be used as a limited alternative to MAILBOX_GTDRIVER_CMD_SAGV_DE_MEM_SS_CONFIG. The highest numbered QGV point is not always the highest bandwidth for display, and masking off more QGV points than necessary is not optimal for power and performance, so the legacy command is of limited use and should be phased out by newer software. Eventually a future project may de-feature the legacy command.

Mailbox Commands

Mailbox Access Routine

1. Ensure any previous GT Driver Mailbox transaction is complete
2. Write GT Driver Mailbox Data0=<request data 31:0> and GT Driver Mailbox Data1=<request data 63:32>
3. Write GT Driver Mailbox Interface RUN_BUSY=1, PARAM2=<parameter2>, PARAM1=<parameter1>, COMMAND=<command>
4. Poll GT Driver Mailbox Interface for RUN_BUSY==0
    - Timeout and fail after 150 us
    - Response COMMAND encoding listed below
5. Read GT Driver Mailbox Data0=<response data 31:0> and Data1=<response data 63:32>

MAILBOX_GTRDIVER_CMD_MEM_SS_INFO

This mailbox command provides the MemSS information requested by display. Pcode collates this information from MemSS registers, parses display relevant information and passes it to display on query. It contains subcommands selected through the field PARAM1[15:8] of the INTERFACE register.

MAILBOX_GTRDIVER_CMD_MEM_SS_INFO_SUBCOMMAND_READ_GLOBAL_INFO

Subcommand used to read MemSS global configuration.

Mailbox COMMAND=0xD and PARAM1=0x0

Response Data Bits	Description
3:0	DDR Type; 0:DDR4, 1:DDR5, 2:LPDDR5, 3:LPDDR4, 4:DDR3, 5:LPDDR3
7:4	Number of populated channels
11:8	Number of enabled QGV points

If GV is fused disabled, BIOS should configure GV such that only 1 point will be reported.

MAILBOX_GTRDIVER_CMD_MEM_SS_INFO_SUBCOMMAND_READ_QGV_POINT_INFO

Subcommand used to read QGV point related timing info. Input is the QGV index following the number of enabled QGV points from the global configuration. Out of bounds QGV index results in error code MAILBOX_GTDRIVER_CC_ILLEGAL_DATA error.

Mailbox COMMAND=0xD and PARAM1=0x1 and PARAM2=<QGV index, counting from 0>

Response Data Bits	Description
15:0	Dclk in multiples of 16.6666 MHz
23:16	TRP in DCLKs
31:24	tRCD in DCLKs
39:32	TRDPRE in DCLKs
48:40	TRAS in DCLKs

Pcode reports the maximums values from across multiple memory channels.

MAILBOX_GTDRIVER_CMD_SAGV_DE_MEM_SS_CONFIG

This mailbox command provides pcode with the bitmap containing the list of allowed GV operation points. Value of 1 in the mask is restricting the associated GV point.

Mailbox COMMAND=0xE

Request Data Bits	Description
0	Restricted QGV point 0
1	Restricted QGV point 1
2	Restricted QGV point 2
3	Restricted QGV point 3
4	Restricted QGV point 4
5	Restricted QGV point 5
6	Restricted QGV point 6
7	Restricted QGV point 7
Other bits	Reserved: Must program all 0s

The response data indicates when a safe GV point has been reached.

Response Data Encoding
Bit 1:0 0x0 = Request accepted. QGV point safe. 0x1 = Request accepted. QGV point not safe yet. Poll again. 0x2 = Request rejected error. QGV point cannot be be made safe. It may be blocked by a BIOS override or fuse. Retry with different points or fail.

The response command indicates if there were any errors.

Response COMMAND Encoding
00h = Success 03h = Illegal data or all GV points masked off 04h = Illegal subcommand 06h = Mailbox locked (BIOS loading not done or GV points not programmed, or BIOS has only enabled a single GV point) 11h = Rejected (BIOS overriding the GV selection)

Pcode will quickly respond to accept the request, then take up to 1 millisecond (typically 100-200us) to move to a safe point. Driver has to poll Pcode with repeated MAILBOX_GTDRIVER_CMD_SAGV_DE_MEM_SS_CONFIG commands (with unchanged request data field and a delay of at least 500us between retries) to find when the move to the safe point is complete.

MAILBOX_GTDRIVER_CMD_DE_LTR_SETTING (Legacy Command)

This mailbox command is the legacy method for disabling Qclk SAGV. It requests Pcode to move to the highest numbered enabled Qclk GV point and then hold there, equivalent to masking off all but one QGV point.

Mailbox COMMAND=0x21

Request Data Bits	Description
2:0	EL_THLD LTR Override: 0 = Disable Qclk GV 3 = Enable Qclk GV
3	Interlace Override: Unused must program with 0
4	VGA Override: Unused must program with 0
63:5	Reserved: Must program all 0s

The response data indicates when a safe Qclk point has been reached.

Response Data Encoding
0x0 = Request accepted. Qclk point not safe yet. Poll again.
0x1 = Request accepted. Qclk point safe.

Pcode will quickly respond to accept the request, then take up to 1 millisecond (typically 100-200us) to move to a safe point. Driver has to poll Pcode with repeated MAILBOX_GTDRIVER_CMD_DE_LTR_SETTING commands (with unchanged request data field and a delay of at least 500us between retries) to find when the move to the safe point is complete.

Frame Buffer Compression

FBC Registers

FBC_CFB_BASE

FBC_CTL

MSG_FBC_REND_STATE

FBC Overview

Frame Buffer Compression (FBC) gives a lossless compression of the display frame buffer to save power by reducing system memory read bandwidth and increasing the time between display engine reads to system memory.

FBC is only available on specific plane(s), depending on project. See FBC_CTL for details. FBC compresses pixels for the plane(s) it is attached to as they are displayed. The compressed data is written into the Compressed Frame Buffer (CFB) in graphics data stolen memory. The compressed data is then read the next time the same line needs to be displayed. Changes to the display front buffer (currently displayed memory surface) through host aperture (GMADR) tiling fences (on projects that support that), render (RCS), and blitter (BCS) are tracked and cause the compressed lines to be invalidated and recompressed. Flips or changes to plane size and panning cause the entire buffer to be recompressed (nuke).

FBC Compression Limit

The FBC compression limit reduces the size of the Compressed Frame Buffer (CFB) by limiting which lines will be compressed. This is used when the graphics stolen memory available for the FBC CFB is smaller than the size of the original uncompressed frame buffer. There is also a FBC compressed vertical limit, listed in FBC_CTL, which is the maximum number of lines FBC can compress. Lines beyond the vertical limit do not need to be accounted for in the CFB size.

When the compression limit is 1:1, every line is written to the CFB, so the CFB width is the same as the original uncompressed frame buffer.

When the compression limit is 2:1, only lines that compress to 1/2 their original size will be written to the CFB, so the CFB width can be 1/2 the original uncompressed frame buffer.

When the compression limit is 4:1, only lines that compress to 1/4 their original size will be written to the CFB, so the CFB width can be 1/4 the original uncompressed frame buffer.

DSM allocation in bytes = min (640, ceiling (plane height / 4))* Compressed buffer stride in cachelines * 64

The 16bpp plane pixel format requires compression limit to be 2:1 or 4:1.

Compressed buffer stride in cachelines = roundup_to_8 (at_least (ceiling [plane_width_in_pixels / (32 * compression_limit_factor)] * 8 + 1))

FBC Programming Overview

1. Set up the compressed frame buffer.
    - The compressed buffer resides in graphics data stolen memory.
    - The stolen memory must be contiguous and un-cached.
    - The stolen memory needed for compressed frame buffer must be greater or equal to CFB size (calculation above).
    - Manage the compressed buffer size at run-time by balancing other graphics memory needs with the FBC allocation, and implement appropriate memory needs prioritization schemes.
2. Tracking for CPU host front buffer modifications
    - CPU Host Front Buffer Tracking sequence below
3. Tracking for display front buffer rendering
    - Render Tracking sequence below
4. Tracking from display front buffer BLTs
    - Blitter Tracking sequence below

LRI commands to MSG_FBC_REND_STATE are used as part of the render and blitter tracking. Those LRIs must be followed SRM commands to the same address.

LRI to MSG_FBC_REND_STATE with data 0x00000004 tells FBC to nuke and invalidate the entire compressed buffer.

Render Tracking With Nuke

- Software must send the nuke LRI after each render to the display front buffer

1. Render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
    c. LRI to MSG_FBC_REND_STATE with data 0x00000004 (nuke)
    d. SRM to read MSG_FBC_REND_STATE and store to a scratch page
2. More render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL

   c. LRI to MSG_FBC_REND_STATE with data 0x00000004 (nuke)
   d. SRM to read MSG_FBC_REND_STATE and store to a scratch page
3. Render commands that do not touch the display front buffer
   a. Render submission
   b. PIPE_CONTROL

Blitter Tracking With Nuke

- Software must send the nuke LRI after each BLT to the display front buffer. Never set 221d0h bit 3 (Address Valid for FBC).

1. BLT commands that touch the display front buffer
   a. BLT submission
   b. MI_FLUSH_DW
   c. LRI to MSG_FBC_REND_STATE with data 0x00000004 (nuke)
   d. SRM to read MSG_FBC_REND_STATE and store to a scratch page
2. More BLT commands that touch the display front buffer
   a. BLT submission
   b. MI_FLUSH_DW
   c. LRI to MSG_FBC_REND_STATE with data 0x00000004 (nuke)
   d. SRM to read MSG_FBC_REND_STATE and store to a scratch page
3. BLT commands that do not touch the display front buffer
   a. BLT submission
   b. MI_FLUSH_DW

MSG_FBC_REND_STATE address for pipe A FBC is 0x50380

CPU Host Front Buffer Tracking

- Software must trigger nuke after each update to the display front buffer.

1. Host updates the display front buffer for the plane that FBC is attached to
2. MMIO write to MSG_FBC_REND_STATE with data 0x00000004 (nuke)

FBC must be disabled if host will update the front buffer without driver being aware of it, or if software is otherwise not able to trigger the nuke after the front buffer updates.

Display Plane Enabling and Disabling with FBC

FBC can be enabled before or after the plane. There is no required order for plane enabling and disabling relative to FBC enabling and disabling, and FBC can be enabled for multiple frames while plane is disabled.

Sequencing Display Plane Updates Together With FBC

The double buffer stall disable mechanism can be used to synchronize and commit the updates of the plane and FBC registers atomically. See the section on Double Buffer Control and use the sequence for synchronizing double buffer updates to synchronize the updates to plane and FBC.

Example: If plane size or format will be changed to a new value that requires FBC compression limit change, stall double buffering around the updates to the plane registers and FBC_CTL register, then when the stall is released the registers will all update on the same frame.

Modifying Stolen Memory

- FBC compressed data is stored in graphics data stolen. If stolen memory will be updated outside of FBC, such as wiping it, then FBC has to be fully disabled first and not re-enabled until after the modification is done.

1. Disable FBC as described in FBC_CTL register.
2. Wait for at least one start of vblank for the disable double-buffering.
3. Modify stolen memory.
4. Re-enable FBC as described in FBC_CTL register.

Watermarks

The watermark registers are used to control the display to memory request timing. The watermarks must be programmed according to the Display Watermark Programming section.

Plane and cursor watermark registers are in the planes section.

Registers
WM_MISC WM_LINETIME DE_POWER1
DE_POWER2_ABOX0 DE_POWER2_ABOX1

DC States

DC_STATE_EN

Xe_{HPD}+ Sequences for Power Wells

Registers

PWR_WELL_CTL

FUSE_STATUS

PWR_WELL_CTL_DDI

PWR_WELL_CTL_AUX

Functions Within Each Well

Except where noted, the registers for a function reside within the same power well as that function.

Note: Some views of the Bspec will show a power domain within each register definition. Those domains may be incorrect.

- PG0 contains the functions for the graphics PCI device and bringing up the display.
  - PCI
  - Clocks
    - The port PLLs are in the PLL and IO/PHY/AFE power domains and require power enabling which is explained in the mode set sequences
  - Shared Functions
    - Interrupts are in PG0, except for pipe interrupts which reside in the power wells associated with the pipes.
    - MBus, except for PIPE_MBUS_DBOX_CTL which reside in the power wells associated with the pipes.
    - Data Buffer (DBUF) registers are in PG0, but the function is in PG1.
  - Central Power, except for FBC which resides in pipe PG.
  - Top Level GTC. DDI Level GTC is in the power well associated with each DDI.
  - Audio MMIO/Verbs
- PG1 contains the functions for internal displays.
  - Data Buffer (DBUF) function is in PG1, but the registers are in PG0.
  - Transcoder A
  - DDI for combo PHYs A-B, including Aux. See note below about IOs.
- PG2 the functions for external displays and VGA.
  - Audio playback
  - Transcoder WD*
  - VGA. The VGA_CONTROL register is in PG0, but requires PG2 to be enabled before VGA is enabled. VGA palette programming uses the pipe A palette/gamma, requiring PGA to be enabled.

    - DDI for combo PHYs C-E and USB typeC PHYs. See note below about IOs, which have separate power control from the DDI and Aux functions in display.
    - KVMR. Hardware will automatically enable PG2 for KVMR.
- PGA contains the functions for pipe A.
    - Pipe A and associated Planes and VDSC/joining
    - FBC
- PGB the functions for pipe B.
    - Pipe B and associated Planes and VDSC/joining
    - Transcoder B (registers reside in PG2, but access path goes through associated pipe)
- PGC contains pipe C.
    - Pipe C and associated Planes and VDSC/joining
    - Transcoder C (registers reside in PG2, but access path goes through associated pipe)
- PGD contains pipe D.
    - Pipe D and associated Planes and VDSC/joining
    - Transcoder D (registers reside in PG2, but access path goes through associated pipe)
- The port PLLs and IOs and Aux IOs are in the IO/PHY/AFE power domains and require power enabling which is explained in the mode set and Aux channel sequences.
- South display is always powered up.
- Aux for DDI combo PHYs C-E and USB typeC PHYs is in PG2.

Sequence

PG0 is controlled by the SoC. The other power wells in display are controlled by software and display firmware.

Each power well is enabled with the following sequence

1. Set PWR_WELL_CTL Power Well # Request to Enable
2. Wait for PWR_WELL_CTL Power Well # State == Enabled; with timeout after 100us
    - Typically expected to take 20us
3. Wait for FUSE_STATUS FUSE PG# Distribution Status == Done, timeout after 20 us
4. Move to enabling sequence for next power well, if needed

Before enabling a pipe power well (PGA, PGB, PGC, PGD), follow the section Display Voltage Frequency Switching, Sequence for Pipe Count Change, with the number of pipe power wells that will be enabled after the enabling completes.

Each power well is disabled with the following sequence

1. Clear PWR_WELL_CTL Power Well # Request to Disable
2. Wait for PWR_WELL_CTL Power Well # State == Disabled; with timeout after 20us
3. Move to disabling sequence for next power well, if needed

After disabling a pipe power well (PGA, PGB, PGC, PGD), follow the section Display Voltage Frequency Switching, Sequence for Pipe Count Change, with the number of pipe power wells that are still enabled after the disabling completes.

Cross-PG Dependencies

There are dependencies that require some power wells to be enabled before others are enabled and disabled after others are disabled.

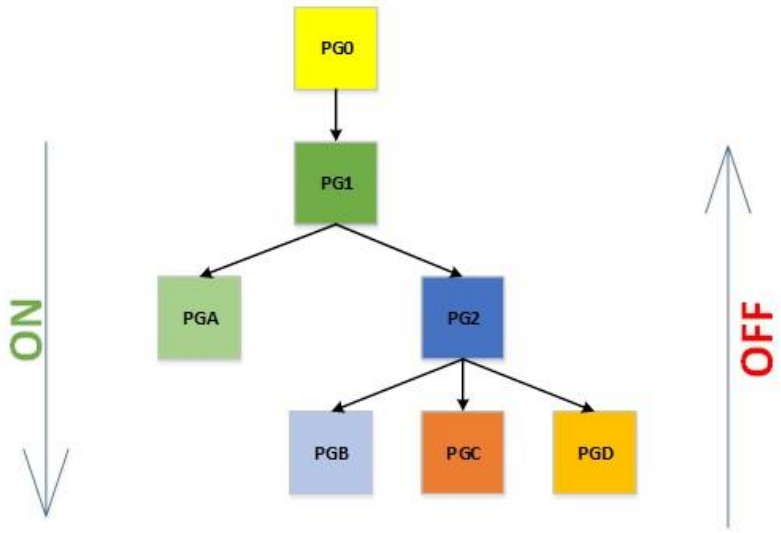
PG0 is enabled by the SoC before software can access display.

PG1 must be enabled before enabling PGA or PG2.

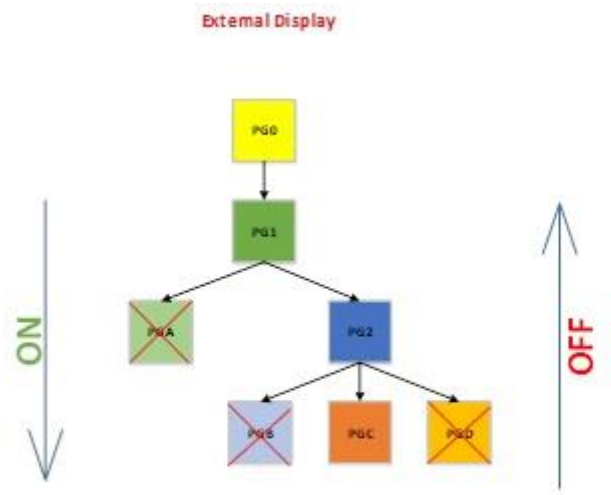
PG2 must be enabled before enabling PGB and pipe power wells besides PGA.

Each pipe power well is independent of the others. Software should save power by disabling unused pipe power wells.

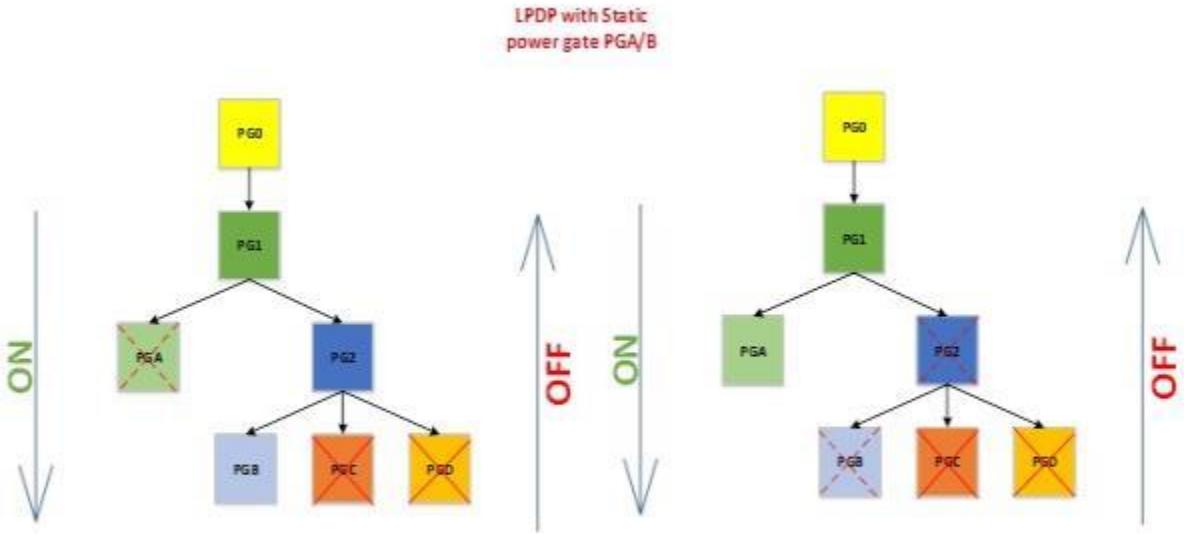
Power Well Sequence



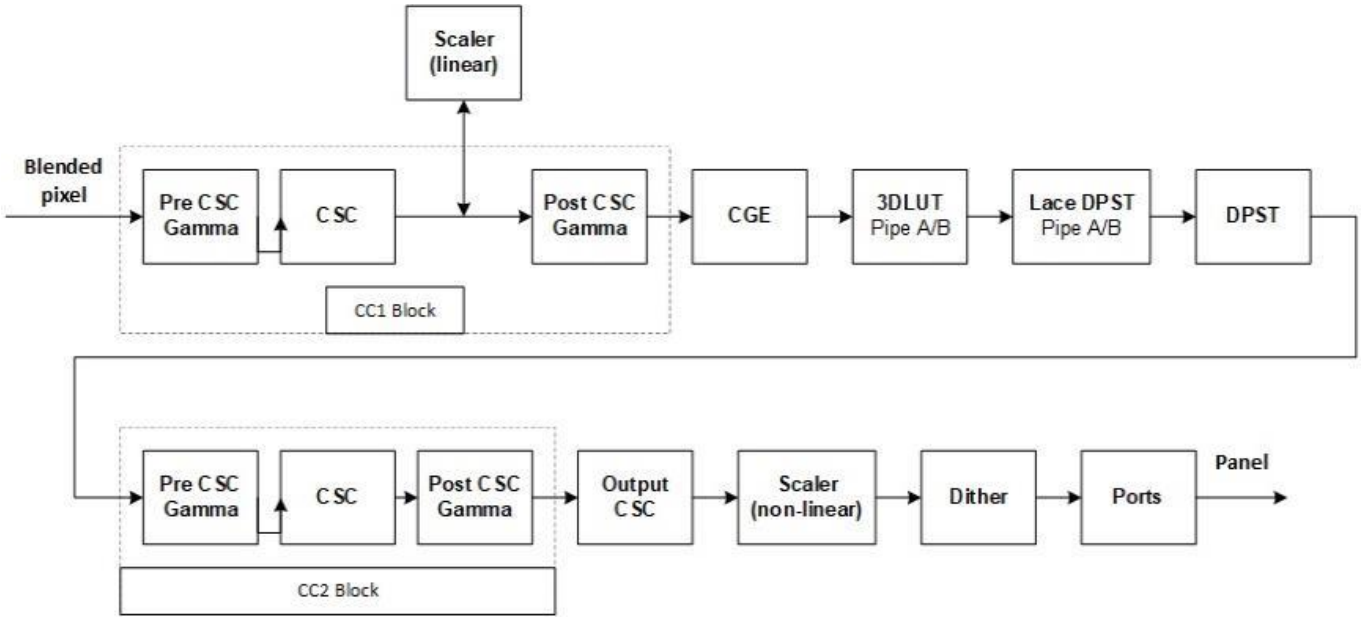
Example of PGC enabled with other pipes disabled



Example of PGB or PGA enabled with other pipes disabled



Pipe



The display pipes contain the planes, blending, color measurement and adjustment, scaling, and dithering.

FBC and LDPST are supported only on pipe A.

3D LUT is supported only on pipes A and B.

The planes read data from memory, format it into pixels, and can apply color correction and scaling.

Each display pipe has 5 planes and a cursor.

The plane blending combines the output from all the planes following a fixed Z-order. Plane 1 is the bottom most plane and higher numbered planes stack on top of it. Cursor goes ontop of all the planes.

Planes 1-3 support HDR. Planes 4-5 support SDR.

Plane details are in the Universal Plane and Plane Capability and Interoperability sections.

The background color that is seen under the bottom most plane is programmable.

The blended pixels pass through several color correction functions and then output to the transcoders.

The pipes each have two pipe scalers. Each pipe scaler can be assigned to scale a plane or scale the blended output.

Luminance Mapping LUT

This is an HDR tone mapping block with increased entries and programmable luma equation RGB coefficients.

It addresses the tone mapping quality improvements with the following key considerations

- More entries for tone mapping for better results.
- Tone-mapping in the real dark regions (<1 nit level).

There is one enhanced tone mapping block per pipe, but it can be attached to any of the HDR planes within the pipe.

The tone mapper is built using a LUT that has a power of two (2^x) segment spacing instead of a uniform segment spacing. Each segment is further divided into equally spaced entries for a total of 171 LUT entries.

x	2^x Segment	# Entries
	0	1
0	1	1
1	2	2
2	4	2
3	8	2
4	16	2
5	32	2
6	64	2
7	128	4
8	256	4
9	512	4
10	1024	4
11	2048	4
12	4096	8
13	8192	8
14	16384	8
15	32768	8
16	65536	8
17	131072	8
18	262144	8
19	524288	16
20	1048576	16
21	2097152	16

22	4194304	16
23	8388608	16
24	16777216	1

The luminance value of the pixel is used as the index into the LUT and the resultant tone map factor is multiplied across all channels. Programmable coefficients are used to calculate the luminance value

Luma = (Kr * Red) + (Kg * Green) + (Kb * Blue)

LM_CTRL

LM_LUMA_COEFF

LM_TONEFACT_INDEX

LM_TONEFACT_DATA

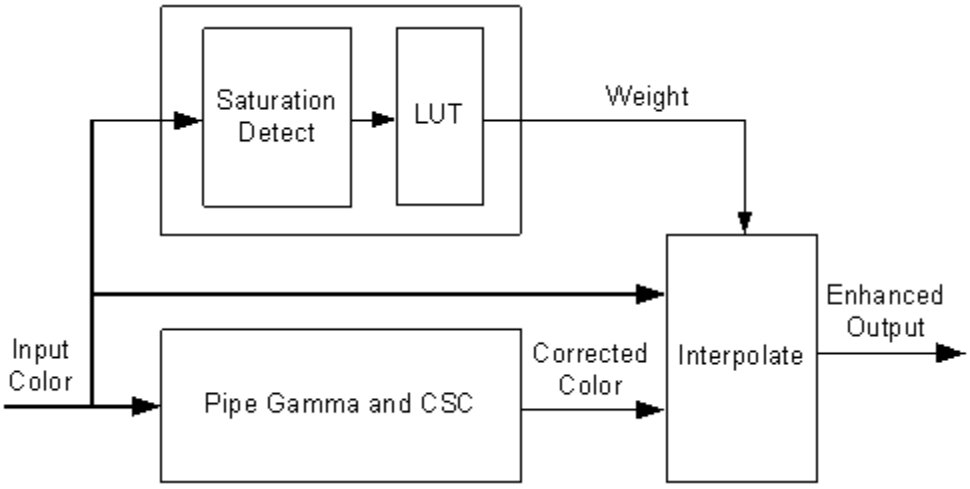
Pipe Color Gamut Enhancement

Pipe color gamut enhancement is used to enhance display of standard gamut content on wide gamut displays. It processes the color value from before and after the pipe gamma and color space correction blocks to create the color gamut enhanced output. The typical usage is to output the pipe gamma and CSC corrected color for areas of low saturated content and the input (not gamma or CSC corrected) color for areas of high saturated content. It is not recommended to use color gamut enhancement with wide gamut inputs.

CGE_CTRL

CGE_WEIGHT

The pipe Gamma and CSC must be programmed to either the split gamma mode or gamma after CSC mode when using pipe color gamut enhancement.



The saturation level of the pipe gamma and CSC input color is detected and used to index into a look up table (LUT) containing programmable weights. The saturation values are linearly distributed across the LUT indexes from the lowest index for lowest saturation to the highest index for highest saturation.

The enhanced output color is created by using the weight value to interpolate between the input color and corrected color. See the following table of weights to amount of input or corrected color used to create the enhanced output color.

Weighting of input and corrected colors

Weight from LUT	Amount of Input Color in Enhanced Output	Amount of Corrected Color in Enhanced Output
00 0000b (minimum)	0%	100%
...	...	...
00 1000b	25%	75%
...	...	...
01 0000b	50%	50%
...	...	...
01 1000b	75%	25%
...	...	...
10 0000b (maximum)	100%	0%

Example weight programming

CGE LUT Index	CGE Weight Value Decimal	CGE Weight Value Binary	CGE Weight Percent Input Color	CGE Weight Percent Corrected Color
0 (lowest saturation)	0	00 0000b	0%	100%
1	0	00 0000b	0%	100%
2	0	00 0000b	0%	100%
3	0	00 0000b	0%	100%
4	0	00 0000b	0%	100%
5	0	00 0000b	0%	100%
6	1.6	00 0010b	5%	95%
7	3.2	00 0011b	10%	90%
8	4.8	00 0101b	15%	85%
9	6.4	00 0110b	20%	80%
10	8.64	00 1001b	27%	73%
11	12.8	00 1101b	40%	60%
12	19.2	01 0011b	60%	40%
13	25.6	01 1010b	80%	20%
14	28.8	01 1101b	90%	10%

CGE LUT Index	CGE Weight Value Decimal	CGE Weight Value Binary	CGE Weight Percent Input Color	CGE Weight Percent Corrected Color
15	32	10 0000b	100%	0%
16 (highest saturation)	32	10 0000b	100%	0%

HDR

High Dynamic Range (HDR)

Key HDR features

- HDR mode supports up to 3 planes in each pipe.
- Tone mapping support in planes.
- Linear blending of HDR planes.
- Linear scaling support in planes and pipes.
- Dedicated chroma upsampler to handle YUV420.
- Programmable color space convertors in planes/cursor.
- Enhanced gamma mode for PQ encoding.

Hardware Capabilities

FP16 Normalizer:

FP16 Normalizer normalize the pixels to -1.0 to 1.0 range. FP16 Normalizer is programed in the PLANE_PIXEL_NORMALIZE register. The programmed FP16 value gets multiplied with the pixel value for normalizing them. Out of band values get clamped to -1.0 to 1.0 value. The output of this block directly feeds into plane CSC block bypassing Chroma up-sampler, input CSC and de-gamma. The FP16 source content must be linear with no gamma encoding.

Chroma up-sampler:

HDR planes have a dedicated bi-linear Chroma up-sampler for converting P0xx/NV12 source pixel formats to YUV444. Supports up to 4k resolution. Chroma up-sampler is programmed in the PLANE_CUS_CTL register.

CUS supports 4 different chroma siting positions that can be programmed through the initial phase. For initial phase programming, refer PLANE_CUS_CTL.

Input CSC:

A programmable 3x3 color space converter generally used for converting YUV content to RGB. This operates in non-linear space. Plane input CSC is programmed in the PLANE_INPUT_CSC_* registers along with the enable bit in the PLANE_COLOR_CTL register.

Plane pre-CSC Gamma:

Plane de-gamma LUT is used for linearizing HDR or SDR source content with gamma. The 128 entries LUT is uniformly spaced with additional 3 entries for 1.0,3.0 and 7.0. Plane input CSC is programmed in the PLANE_PRE_CSC_GAMC_INDEX_ENH and PLANE_PRE_CSC_GAMC_DATA_ENH registers along with the enable bit in the PLANE_COLOR_CTL register.

Plane CSC:

The programmable plane CSC can be used for color space conversion of source content to BT.2020 or panel color gamut. Plane CSC is programmed in the PLANE_CSC_* registers along with the register bit in the PLANE_COLOR_CTL register.

Plane post-CSC Gamma:

In HDR mode, this block can be used for tone mapping/dynamic range adjustment of each source content to a common reference luminance range before blending. The LUT must be programmed to use "Multiply mode" in the PLANE_POST_CSC_GAMC_INDEX_ENH, PLANE_POST_CSC_GAMC_DATA_ENH and PLANE_COLOR_CTL registers.

Hardware computes the pseudo luminance of the incoming pixel using the following equation, uses it index the LUT and compute an adjustment factor 'F'. Toned mapped output R, G and B values are computed by scaling the input R, G and B channel by 'F'.

Lin = 0.25*Red input + 0.625*Green input + 0.125*Blue input.

Plane scaling:

Plane scaling supports both linear and non-linear scaling modes. The scaling mode is programmed in the PS_CTRL. In HDR mode, scaling and blending operations are generally performed in linear mode.

Linear Blending:

With precision improvements, hardware supports blending in linear mode. Linear blending is enabled by programming the HDR mode bit in the PIPE_MISC register.

Cursor:

For HDR usages, the cursor plane supports a programmable De-gamma LUT, Color Space Convertor and a Luminance scaler. Luminance scaler scales each color component by a programmed 10 bit value fractional value.

Cursor de-gamma LUT is programmed in in CUR_PRE_CSC_GAMMA_INDEX and CUR_PRE_CSC_GAMMA_DATA registers along with the enable bit in CUR_CTL. Cursor CSC is programmed in the CUR_CSC_COEFF register along with the enable bit in CUR_CTL. Luminance scaling is enabled and programmed in CUR_COLOR_CTL.

Pipe Scaler:

Pipe scaler supports linear scaling.

Pipe Gamma:

Supports a 12 bit multi-segmented gamma mode that provides high quality HDR PQ encoding. Refer to the "Pipe Palette and Gamma" page for details

Pipe LDPST

DPLC_CTL

DPLC_HIST_INDEX

DPLC_HIST_DATA

DPLC_IE_INDEX

DPLC_IE_DATA

DSB Engine Programming

Display State Buffer Programming

A DSB (Display State Buffer) is a queue of MMIO instructions in the memory which can be offloaded to DSB HW in Display Controller. DSB HW is a DMA engine that can be programmed to download the DSB from memory. It allows driver to batch submit display HW programming. This helps to reduce loading time and CPU activity, thereby making the context switch faster.

It can be chiefly used as an extension of the current flip programming

There are three DSB DMA engines per pipe.

Different DSBs are required if different tasks need to be accomplished simultaneously

A DSB can access only the pipe DSB is attached to, including the planes and FBC, and the transcoder attached to the pipe.

A DSB cannot access any registers outside pipe and transcoder registers including, but not limited to clocks, audio, power well controls, south display.

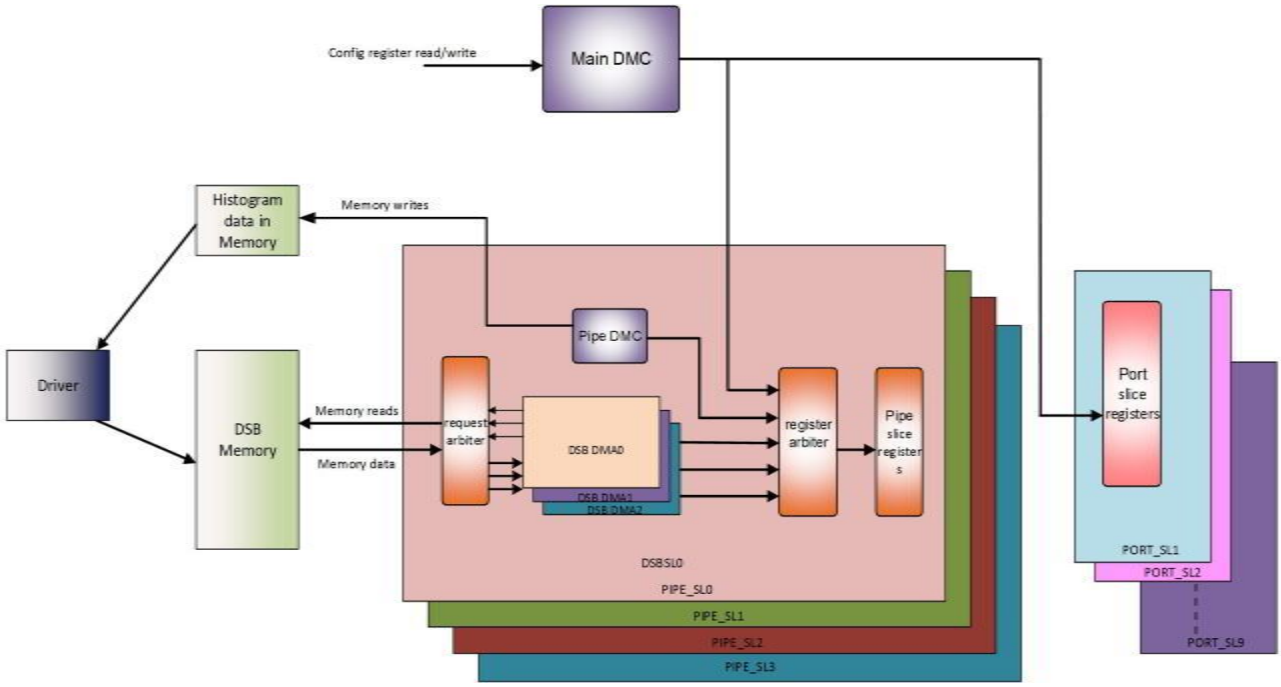
DSB HW can support only register writes (both indexed and direct MMIO writes). There are no registers reads possible with DSB HW engine.

DSB HW has poll function capability.

DSB HW cab be used in VSC Extension SDPHDR Meta data programming.

DSB HW is used in FAST LACE programming.

Below is the block diagram of the DSB HW and shows the general flow of the HW and SW flow. Driver will upload the Display State Buffer with instructions to program the Pipe registers.



Following registers will be used in the DSB HW. Software must ensure correct programming of these registers for the proper function of the HW. Illegal/incorrect programming of these register may result in unexpected behavior of hardware.

DSB HW Registers

DSB_CTRL

DSB_BUFRPT_CNT

DSB_HEAD_PTR

DSB_TAIL_PTR

DSB_MMIOCTRL

DSB_POLLMASK

DSB_POLLFUNC

DSB_INTERRUPT

DSB_PF_LN_LOWER

DSB_PF_LN_UPPER

DSB_PMCTRL

DSB_PMCTRL_2

DSB_CURRENT_HEAD_PTR

DSB_RM_TIMEOUT

DSB_RMTIMEOUTREG_CAPTURE

DSB_INTERRUPT

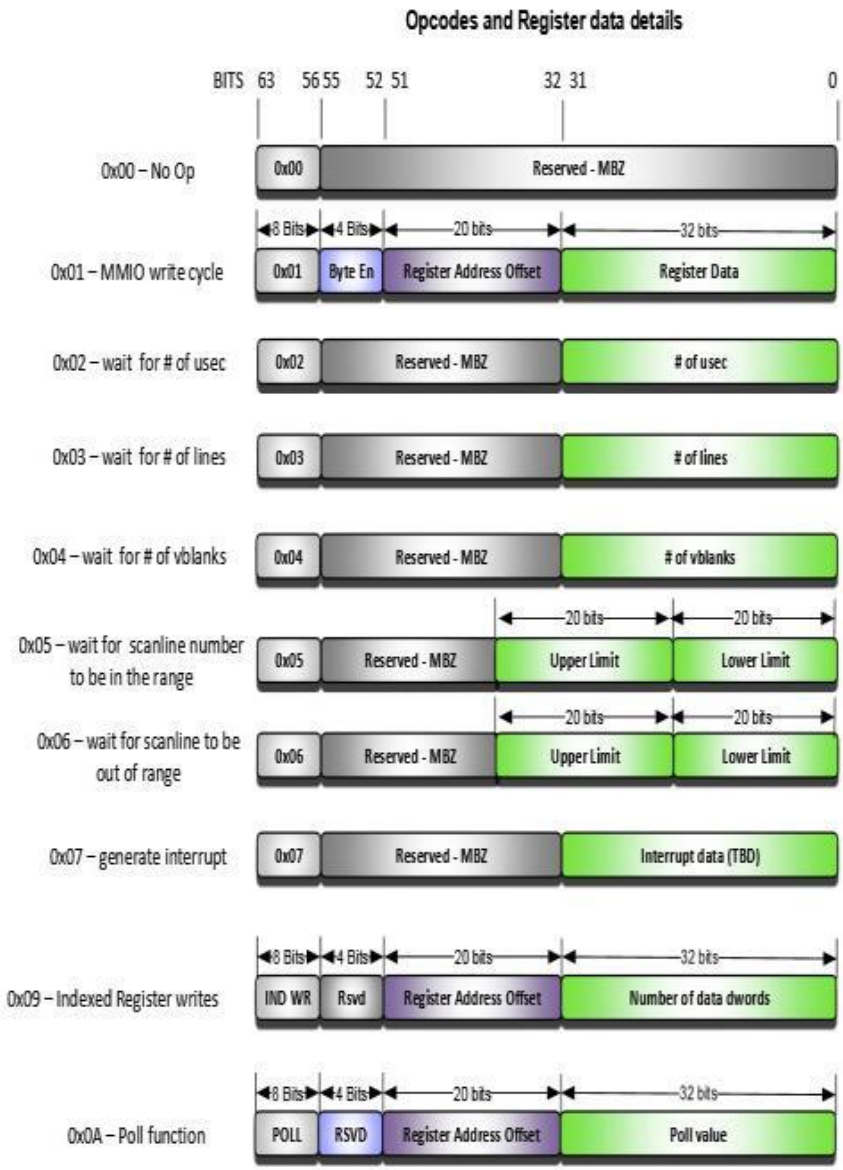
DSB_STATUS

PIPEDMC_CONTROL

PIPE_DMCSCANLINECOMP

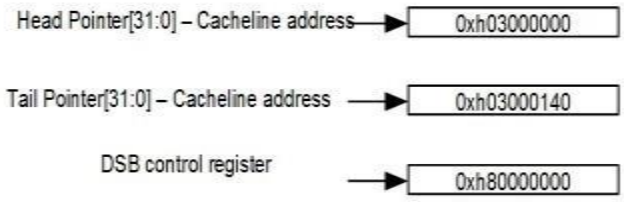
DSB Instructions

The following DSB instructions are supported by the DSB HW. Software must upload the instructions in the defined format shown below.



Below are some examples of the DSB instructions format in the memory.

											6 3		3 1	0	
20000000	10000000	09F7227c	00000003	00000008	00000007	00000006	00000005	00000004	00000003	00000002	00000001	09F7227c	00000008		0xh03000000
01F72250	8003801d	01F7224c	80020011	01F72248	80010009	01F72244	80004003	01F72240	80004003	01F721c8	00000000	00000000	30000000		0xh03000040
01F721cc	00002000	01F7219c	91440000	01F721a4	00000000	01F72268	00004007	01F7225c	8005c02f	01F72258	80050029	01F72254	80044023		0xh03000080
07000000	FFFFFFFF	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	01F72180	84000408		0xh03000100



General Programming Sequence of the DSB HW

1. Driver will form a display state buffer using the supported DSB instructions to program the display registers and upload it in the memory.

2. Driver must program the **DSB_CTRL** register to enable the DSB engine. SW may choose to start the DSB DMA engine to start the memory requests right away or wait for a vblank to trigger the DSB DMA engine. SW may also choose to repeat to play the same buffer again on every vblank by setting re-iterate bit of the **DSB_CTRL** register. SW must follow the programming rules defined in the **DSB_CTRL** register bits.

3. Driver cannot update or modify the DSB registers within the DSB program.

4. Driver must program the **DSB_HEAD_PTR** with the first cacheline address of the program and **DSB_TAIL_PTR** with the last cacheline address of the DSB program plus one. DSB HW DMA engine will be triggered on the tail pointer update.

- Head and Tail Pointers (32 bits each).
- SW must make sure that there must at least one cacheline in the DSB program when updating Head and Tail pointer. i.e. head and tail pointer cannot be equal.
- Cacheline aligned graphics address.
- TP points to last location+1 cacheline address.
- SW should not modify these DSB registers when DSB is busy.
- Each cacheline can have 8 instructions.
- 64 bits per instruction.
- Fill incomplete cachelines with NoOps
- Cacheline data is in little Endian.
- One DSB can write to the registers of another DSB engine within the pipe.

5. Driver can poll on the DSB Status bit in the DSB_CTRL register to check if the DSB engine has completed. If DSB program has an interrupt instruction as the last instruction, then an interrupt will generated to SW which can be used to check the status the DSB DMA engine.

- SW can enable the interrupts by programming the appropriate enabled in DSB_INTERRUPT register.
- A pipe interrupt at DE_PIPE_INTERRUPT will be generated when interrupts are correctly enabled in the IMR and IER registers. Bits 13,14,15 shows the status of interrupts from each DSB engine.
- SW must clear the sticky bits in the DSB_INTERRUPT first and then clear the ISR bits in the DE_PIPE_INTERRUPT

6. Each of the instructions are described below in detail. Each of the supported functions/capabilities are also described that SW can use to program DSB HW. Every instruction must start at a 64 bit boundary.

NOOPs

The opcode of this instruction is 0x00. All the instructions must start and end on the cacheline boundaries. NoOPs can be used to complete the cachelines. All the 64 bits of this instruction are 0's.

MMIO Writes

The opcode of this instruction is 0x01. Lower 32 bits of the instruction is the MMIO write data. Bits 51:32 is the offset address of the MMIO write. Bits 55:52 are the byte enables of the mmio write instruction.

Wait for number of Microseconds

The opcode of this instruction is 0x02. The lower 32 bits of the instructions will have the number of microseconds to wait. DSB HW will wait for the programmed number of microseconds before processing the next instruction.

Wait for number of lines

The opcode for this instruction is 0x03. The lower 32 bits of the instruction has the number of lines to wait. DSB HW will wait for the programed number of lines before processing the next instruction.

Wait for number of vblanks

The opcode for this instruction is 0x04. The lower 32 bits of the instruction has the number of vblanks to wait. DSB HW will wait for the programed number of vblanks (pipe undelayed vblank) before processing the next instruction.

Wait for scanline number in range

The opcode for this instruction is 0x05. The lower 40 bits of the instruction has the upper and lower scanline numbers. DSB HW will wait for the scanline number to be within the programmed range of the line numbers before processing the next instruction.

Wait for scanline number out of range

The opcode for this instruction is 0x06. The lower 40 bits of the instruction has the upper and lower scanline numbers. DSB HW will wait for the scanline number to be out of the programmed range of the line numbers before processing the next instruction.

Generate Interrupt

The opcode for this instruction is 0x07. DSB HW, after decoding this instruction, will generate a interrupt when interrupt is enabled in DSB_INTERRUPT register and the IMR and IER register bits are correctly programmed in DE_PIPE_INTERRUPT .

Indexed MMIO register writes

The opcode for this instruction is 0x09. The lower 32 bits of this instruction will have the number of the data dwords of the indexed MMIO writes. Bits 51:32 of this instruction will have the address offset.

- The data dwords are followed by these instructions.
- If the number of the data dwords is odd then the last dword of the instruction will be 0's to complete the 64 bit instruction. Example is shown below

Example of Indexed Register Writes with odd number of dwords

Poll Function

The opcode for this instruction is 0x0A. Lower 32 bits of the instruction will have the poll value to be compared and bits 51:32 have the offset of the register that needs to be polled on.

- SW must program the DSB_POLLMASK before this instruction is used. Poll mask will have the enable on each of the 32 bits that the data will be compared against.
- SW must also program the DSB_POLLFUNC register prior to using the poll function instruction. Poll function has to be enabled before using the poll function.
    - The default value of number of microseconds to wait before a poll retry is set to 2us.
    - The default number of retries is set to 50 times before a timeout or poll fails
- SW can choose to add these two registers as part of DSB program itself before using the POLL instruction. But SW must add 5 NOOPs after MMIO writes instructions to these (DSB_POLLMASK and DSB_POLLFUNC) registers to allow some time for DSB engine to process the register writes to itself.

Special programming

The following transcoder registers need special programming (switch to non-posted programming to give time for cross-clock synchronization) if they are part of the DSB program in the buffer. SW must follow the sequence listed below in the DSB program to program these registers.

1. Set the non-posted bit 8 to 1 in the **DSB_CTRL** register as part of the DSB program.
2. Add four No-Op instructions.
3. Add the transcoder register programming.
4. Set the non-posted bit 8 to 0 in the DSB_CTRL register.
5. Add four No-Op instructions
6. Add the rest of the programming (if any).

The list of these special registers is below.

TRANS_DDI_FUNC_CTL2
TRANS_DDI_FUNC_CTL
VIDEO_DIP_CTL
VIDEO_DIP_PPS_DATA_*
VIDEO_DIP_ADAPTIVE_SYNC_DATA_*
TRANS_CONF
TRANS_STEREO3D_CTL
TRANS_DPT_PAT
TRANS_SET_CONTEXT_LATENCY
TRANS_VRR_CTL
TRANS_VRR_FLIPLINE
TRANS_VRR_VMAXSHIFT
TRANS_VRR_VMAX
TRANS_VRR_VMIN
TRANS_VRR_VSYNC
TRANS_LINKN1
SRD_CTL
SRD_PERF_CNT
PSR_MASK
DP_COMP_CTL
DP_TP_CTL
PSR2_CTL

DSB Atomic Usage

The following is the framework for using DSB to program pipe and plane registers to atomically update (all update together for the same frame). DSB can also be used to update registers non-atomically or to update single resources that are self-atomic, with simpler sequences that don't require any wait for vblank.

For this atomic usage, transcoder timings must be programmed to create a window between start of undelayed vblank and delayed vblank. The typical requirement is a 100 microsecond window to provide enough time for DSB to program all the pipe and plane registers. Smaller values may be used for testing. Larger values may exceed required time for filling the display pipeline during vblank when using reduced blanking. ROUNDUP(desired window time/line time) = number of lines to program delayed vblank larger than undelayed vblank.

Fixed Refresh Rate

This requires VRR to be disabled.

Before enabling DSB, set bit 23=1b in the following register for the pipe and DSB to skip waits when PSR is entered.

Pipe A: DSB0 0x70BF0, DSB1 0x70CF0, DSB2 0x70DF0

Pipe B: DSB0 0x71BF0, DSB1 0x71CF0, DSB2 0x71DF0

Pipe C: DSB0 0x72BF0, DSB1 0x72CF0, DSB2 0x72DF0

Pipe E: DSB0 0x73BF0, DSB1 0x73CF0, DSB2 0x73DF0

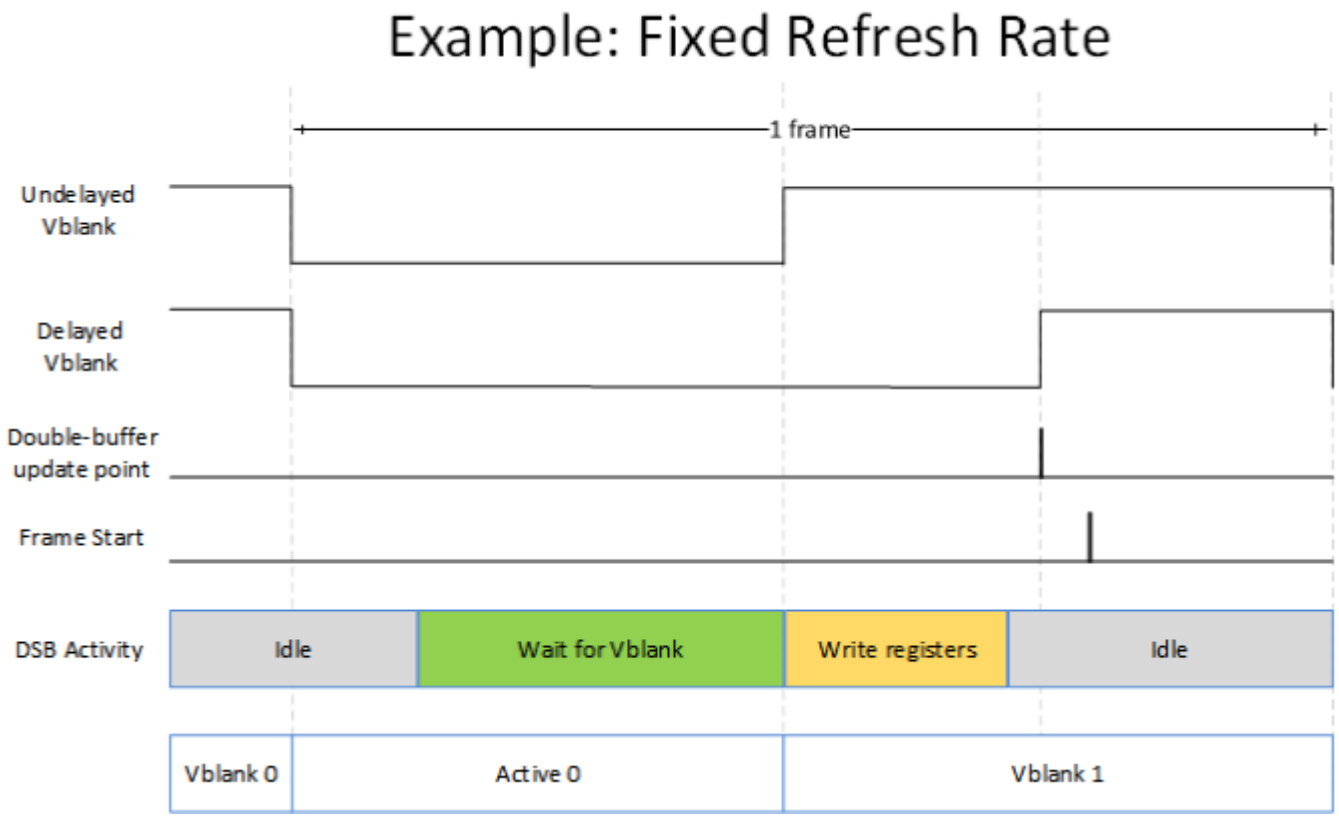
DSB will need to align instruction execution to the start of vblank. Either enable DSB_CTRL Wait for VBLANK or use the wait for vblank instruction.

Follow the General Programming Sequence of the DSB HW to setup and initiate DSB.

Use the following DSB instructions

1. If DSB_CTRL Wait for VBLANK not enabled, Wait for 1 vblank.
2. Write 1 to DSB_STATUS bit 16 to clear any previous indication that DSB was busy during delayed vblank.
3. Write pipe and plane registers to be updated for the next frame.
    - Depending on the registers to write, this can include register writes and indexed register writes.
    - Requirements for writing some registers in certain orders to arm them for double-buffer updates are still required.
4. Optional: Generate Interrupt

After DSB completes, found through interrupt or polling on DSB status, read DSB_STATUS bit 16 to find if delayed vblank started before DSB finished. If it did, then there was an error and the pipe and plane programming may not be complete. Software may log the error and attempt recovery.



Variable Refresh Rate

This requires VRR with push bit mode.

Before enabling VRR, set bit 31 in the following register for the transcoder attached to this pipe to enable VRR safe window generation.

Transcoder A: 0x420C0

Transcoder B: 0x420C4

Transcoder C: 0x420C8

Transcoder D: 0x420D8

Before enabling DSB, set bits 23=1b, 15:14=11b, bits 7:6=11b in the following register for the pipe and DSB to enable VRR safe window to be used instead of vblank and to skip waits when PSR is entered.

Pipe A: DSB0 0x70BF0, DSB1 0x70CF0, DSB2 0x70DF0

Pipe B: DSB0 0x71BF0, DSB1 0x71CF0, DSB2 0x71DF0

Pipe C: DSB0 0x72BF0, DSB1 0x72CF0, DSB2 0x72DF0

Pipe E: DSB0 0x73BF0, DSB1 0x73CF0, DSB2 0x73DF0

DSB will need to align instruction execution to the VRR safe window. Either enable DSB_CTRL Wait for VBLANK during DSB setup or use the wait for vblank instruction.

Follow the General Programming Sequence of the DSB HW to setup and initiate DSB.

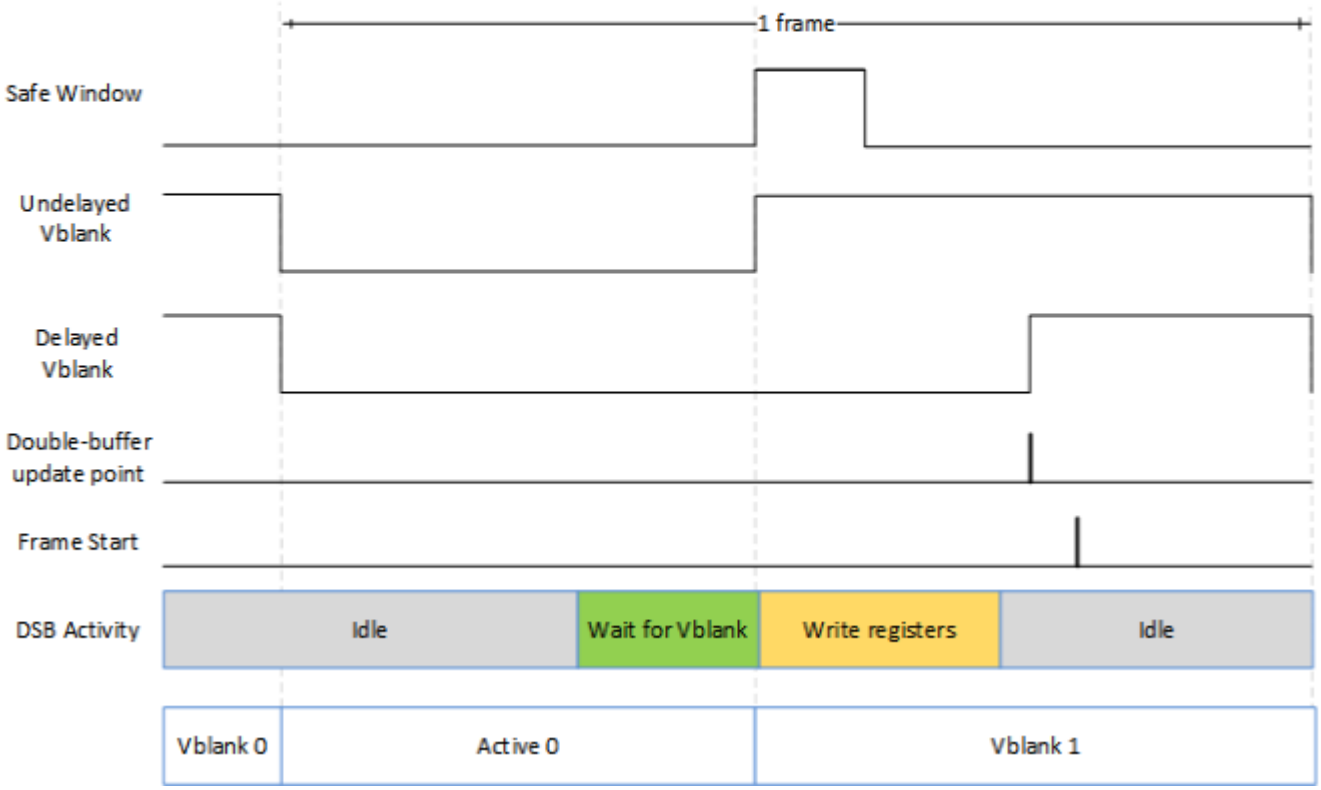
Use the following DSB instructions

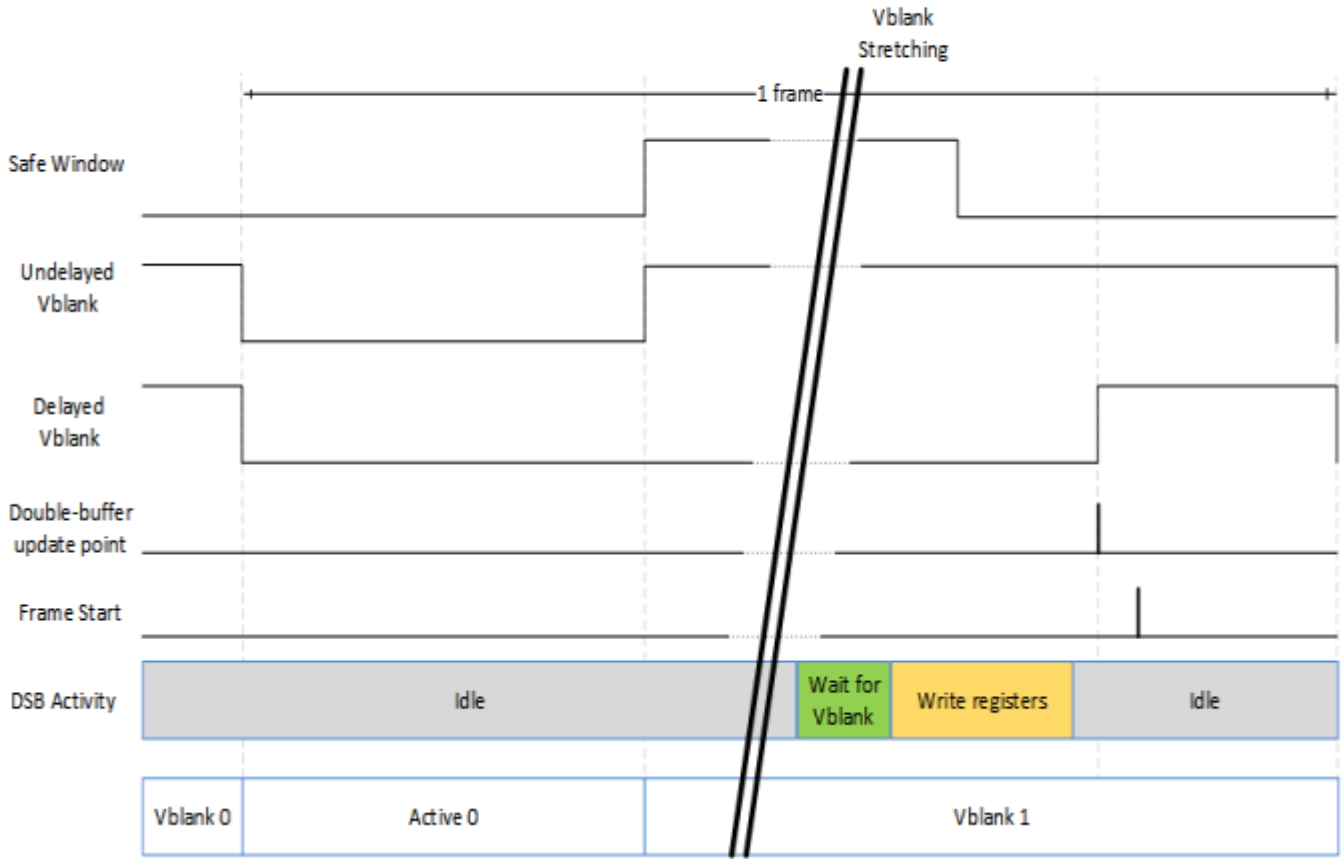
1. If DSB_CTRL Wait for VBLANK not enabled, Wait for 1 vblank.
2. Write 1 to DSB_STATUS bit 16 to clear any previous indication that DSB was busy during delayed vblank.
3. Write 1 to transcoder push bit*. This step can be done here or in step 5. Initiating push at this point will allow the vblank to end sooner, but is different from the non-DSB programming flow that does push at the end.
4. Write pipe and plane registers to be updated for the next frame.
    - Depending on the registers to write, this can include register writes and indexed register writes.
    - Requirements for writing some registers in certain orders to arm them for double-buffer updates are still required.
5. Write 1 to transcoder push bit* if not done in earlier step.
6. Optional: Generate Interrupt

After DSB completes, found through interrupt or polling on DSB status, read DSB_STATUS bit 16 to find if delayed vblank started before DSB finished. If it did, then there was an error and the pipe and plane programming may not be complete. Software may log the error and attempt recovery.

*If driver knows that push is already guaranteed to be set outside of this DSB, such as by another DSB running in parallel, then this push can optionally be skipped. This requires very careful sequencing to ensure that this DSB does not get out of alignment with whatever is issuing the push.

Example: VRR – DSB Wait Starts Before Vblank





Adding Dynamic Metadata

If dynamic metadata will be sent just once, then it can be programmed together with the pipe and plane registers in the previous sequences.

Color Space Conversion

CSC_COEFF

CSC COEFFICIENT FORMAT

CSC_PREOFF

CSC_POSTOFF

CSC_MODE

OUTPUT_CSC_COEFF

OUTPUT_CSC_PREOFF

OUTPUT_CSC_POSTOFF

CSC_CC2_COEFF

CSC_CC2_PREOFF

CSC_CC2_POSTOFF

PRE_CSC_CC2_GAMC_INDEX

PRE_CSC_CC2_GAMC_DATA

POST_CSC_CC2_INDEX

POST_CSC_CC2_DATA

The high color channel is the most significant bits of the color. The low color channel is the least significant bits of the color. The medium color channel is the bits between high and low. For example: In RGB modes Red is in the High channel, Green in Medium, and Blue in Low. In YUV modes, V is in the High channel, Y in Medium, and U in Low.

The color space conversion registers are double buffered and are updated on the start of vertical blank following a write to the CSC Mode register for the respective pipe.

The matrix equations are as follows:

OutputHigh = (CoefficientRY * InputHigh) + (CoefficientGY * InputMedium) + (CoefficientBY * InputLow)

OutputMedium = (CoefficientRU * InputHigh) + (CoefficientGU * InputMedium) + (CoefficientBU * InputLow)

OutputLow = (CoefficientRV * InputHigh) + (CoefficientGV * InputMedium) + (CoefficientBV * InputLow)

Example programming for RGB to YUV is in the following table:

The input is RGB on high, medium, and low channels respectively and the desired YUV output is VYU on high, medium, and low channels respectively.

Program CSC_MODE to put gamma before CSC.

Program the CSC Post-Offsets to +1/2, +1/16, and +1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

	Bt.601		Bt.709	
	Value	Program	Value	Program
RU	0.2990	0x1990	0.21260	0x2D98
GU	0.5870	0x0968	0.71520	0x0B70
BU	0.1140	0x3E98	0.07220	0x3940
RV	-0.1687	0xAAC8	-0.11460	0xBEA8
GV	-0.3313	0x9A98	-0.38540	0x9C58
BV	0.5000	0x0800	0.50000	0x0800
RY	0.5000	0x0800	0.50000	0x0800
GY	-0.4187	0x9D68	-0.45420	0x9E88
BY	-0.0813	0xBA68	-0.04580	0xB5E0

Example programming for YUV to RGB is in the following table:

The input is VYU on high, medium, and low channels respectively.

The output is RGB on high, medium, and low channels respectively.

Program CSC_MODE to put gamma after CSC.

Program the CSC Pre-Offsets to -1/2, -1/16, and -1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GY	1.000	0x7800	1.000	0x7800
BY	0.000	0x0000	0.000	0x0000
RY	1.371	0x7AF8	1.574	0x7C98
GU	1.000	0x7800	1.000	0x7800
BU	-0.336	0x9AC0	-0.187	0xABF8
RU	-0.698	0x8B28	-0.468	0x9EF8
GV	1.000	0x7800	1.000	0x7800
BV	1.732	0x7DD8	1.855	0x7ED8
RV	0.000	0x0000	0.000	0x0000

Dithering after Color Conversions

Dithering is present in the Post CSC Gamma blocks within both pipe color conversion blocks.

The dithering at these locations will have the ability to dither down to 12 bits while the dithering at end of pipe only dithers down to 10 bits. The dithering at these locations is enabled through the **GAMMA_MODE** register.

Pipe 3D LUT

The 3D LUT is a pixel modification function which resides in the post blend color processing section of the display pipeline. It is used to apply non-linear transforms on each color component on a per pixel basis. Our LUT implementation uses a 17x17x17 three dimensional matrix of color points, with each point holding a 30 bit pixel value (10 bpc).

3D LUT functionality is supported only in pipe A and pipe B.

LUT_3D_CTL

LUT_3D_INDEX

LUT_3D_DATA

Programming

Enabling 3D LUT

Software should follow the sequence below.

1. Check if the "New LUT Ready" bit is clear. If set, software must wait till the bit is clear. LUT entries must not be change the LUT entries when the "New LUT Ready" is set.
2. Load the desired 3D LUT entries.
3. Set the "Enable" and the "New LUT Ready" bits.

The LUT buffer is double buffered. When 3D LUT functionality is enabled, the hardware observes the "New LUT Ready" bit on every vblank start. If the "New LUT Ready" bit is set, hardware loads the LUT entries into its working RAM and clears the bit. The 3D LUT functionality works with programmed LUT values in the following frames until it gets disabled. When the "New LUT Ready" bit is clear, the software is allowed to modify the LUT entries.

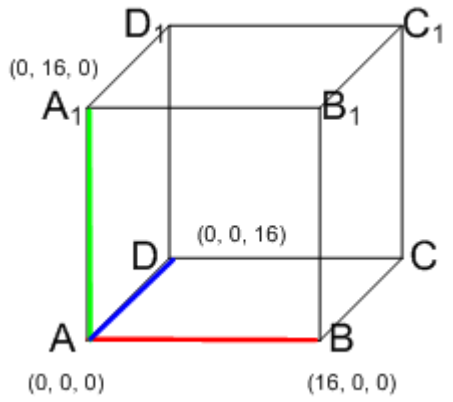
Disabling 3D LUT

1. Clear the "Enable" bit to '0'.

Programming 3D LUT Entries

The LUT array is accessed by an index/data register pair. The index register is read/writable and auto-increments after each read/write to the data register. Write '0' into the index register, followed by 4913 LUT entry writes to the data register.

Each LUT 3D entry is 30 bits and programmed as R10G10B10 (msb... lsb) value in the LUT_3D_DATA register. Since 10 bit values are used for all 17 points, the max value programmed is limited to 1023. A 1:1 mapping should use [0, 64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960, 1023].



The LUT entries should start at A and end at C_1 following the sequence specified below.

```
Iterate on Red axis from 0 - 16 {
 Iterate on Green axis 0 - 16 {
 Iterate on Blue axis 0 - 16 {
program 3D LUT entry
}
 }
 }
```

Pipe DPST

Registers

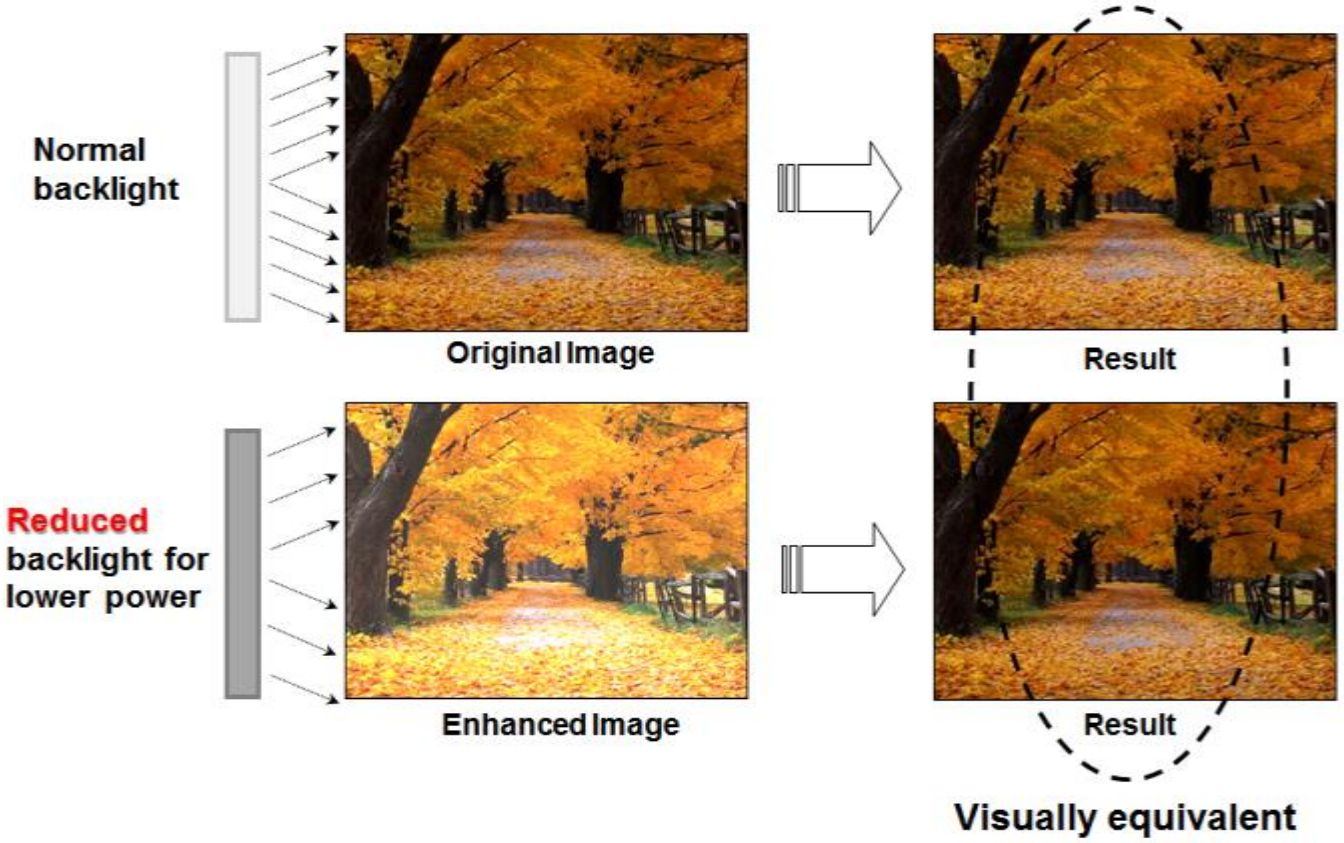
DPST_CTL

DPST_GUARD

DPST_BIN

Overview

Display Power Savings Technology (DPST) achieves significant platform average power savings by dynamically decreasing the display backlight brightness, while increasing the pixel values in the displayed image by a corresponding factor. The goal of DPST is to provide equivalent end-user-perceived image quality at a decreased backlight power level.



DPST generates statistics (histogram) for each image frame that is sent to the display. These statistics are used to determine if, and by how much, the backlight level can be reduced, saving backlight power. In order to maintain the same image brightness, the image pixel values are increased by an amount related to the backlight level reduction.

DPST is composed of three blocks.

Histogram Block

The hardware histogram block generates image statistics based on the pixel stream input. These statistics are used by the Processing block to determine how much the backlight level can be reduced.

The histogram block has the following requirements.

- Generate a histogram for each frame of display data
  - The histogram is generated based on the brightness of each pixel.
    - DPST_CTL Histogram Mode Select selects how the brightness is determined; either HSV max(RGB_{in}), or the luma after converting RGB_{in} to YUV.
  - The histogram is composed of 32 bins with each bin covering a range of 8 values for 8 bit pixel component values. The first bin covers the values 0 thru 7.
    - If the pixel component values use more than 8 bits, the most significant 8 bits are used to form the histogram.
  - Each bin value has enough bits to count every pixel within an image.

- Enabled by DPST_CTL IE Histogram Enable. The enable is double-buffered so it will not enable until the next vblank, and then the histogram will take one frame to complete, so the histogram will not be ready until at least two vblanks after setting the enable.

- Two sets of histograms
  - The histogram currently being generated and a saved histogram from a previous frame.
  - The current histogram is moved to the saved histogram at the end of a frame in which a threshold event occurs.
  - Each bin of the saved histogram is readable by software.
    1. Clear DPST_CTL Bin Register Function Select to TC
    2. Wait for vertical blank for switch to TC mode, can skip if step 1 was done more than 1 vblank previously
    3. Set DPST_CTL Bin Register Index to 0
    4. Read DPST_BIN
    5. If DPST_BIN Busy Bit is 1, go to step 3
    6. Store DPST_BIN Data
    7. Go to step 4 until all 32 bins are read
- Threshold register for comparing the histogram of the current frame to the histogram of the saved frame.
  - If any bin in the current histogram differs from the same bin in the saved histogram by more than the value in DPST_GUARD Threshold Guardband, then a threshold event is generated at the start of vertical blank.
  - DPST_GUARD Guardband Interrupt Delay specifies the number of consecutive frames the threshold must be exceeded before generating a threshold event.
    - This allows filtering out momentary variations from generating a threshold event.
  - DPST_GUARD Histogram Interrupt Enable enables the threshold event interrupt.
  - DPST_GUARD Histogram Event Status is a sticky bit that is set with the interrupt and must be cleared to receive more histogram events.

Enhancement Block

The hardware enhancement block adjusts the pixel values sent to the display, compensating for the brightness loss due to lowering of the display backlight level.

The enhancement block has the following requirements.

- Located after the histogram block
- Enabled by DPST_CTL IE Modification Table Enable
- Find the enhancement factor from a Look Up Table (LUT) with 33 entries
  - Switch (DPST_CTL Enhancement mode)
    - Case Direct: LUT address = most significant bits of RGB_{input}
    - Case Multiplicative: LUT address = most significant bits of HSV max(RGB_{input})

        - Case Additive: LUT address = most significant bits of Y channel after converting RGB_{input} to YUV
    - DPST_CTL IE Table Value Format selects if the enhancement factor is a 1 integer and 9 fractional bits format, or a 2 integer and 8 fractional bits format.
        - The 2 integer and 8 fractional bits format allows for brightness increases nearly to 4x, but with reduced precision.
    - The final enhancement factor is derived by interpolating between the addressed LUT entry and the next entry, using the lower bits of the input.
    - Each entry of the LUT is programmable by software.
        1. Set DPST_CTL Bin Register Function Select to IE
        2. Wait for vertical blank for switch to IE mode, can skip if step 1 was done more than 1 vblank previously
        3. Set DPST_CTL Bin Register Index to 0
        4. Write enhancement factor to DPST_BIN Data
        5. Go to step 4 until all 33 entries are written
    - The IE values are double-buffered and will update on the next start of pipe vblank. The values are not atomically updated or tied into the global double buffering control (disable/stall), so if programming straddles the pipe vblank start, some values will update in the upcoming frame and some in the frame after that.

- The enhancement factor modifies each input pixel component value with the method selected by DPST_CTL Enhancement Mode.
    - Direct lookup mode replaces the input pixel value with the enhancement factor.
    - Additive mode increases the input pixel value by the enhancement factor.
    - Multiplicative mode multiplies the input pixel value by the enhancement factor.

Processing Block

The software processing block responds to the histogram threshold interrupts, determines how much the backlight level can be reduced, then sets the backlight level and programs the Enhancement block.

The Processing block has the following requirements.

- Enable or disable the hardware blocks based on OS and user control inputs.
- Respond to the histogram threshold interrupts by reading the histogram and calculating a new backlight level based on statistics from the histogram.
    - The calculation is proprietary. In general, darker images will allow greater backlight reduction, and increased power savings.
    - The aggressiveness of backlight reduction can be controlled by the OS and user.
    - The final backlight reduction amount must be combined with backlight level requirements set by the OS, applications, and other power saving technologies.

- Program the Enhancement block to compensate for the brightness loss due to reducing the backlight level.
    - Because the maximum RGB component value is limited, not all pixel values can be perfectly compensated.
    - The number of pixel values which cannot be perfectly compensated is a function of the aggressiveness level.
- Phase-in the backlight level change and pixel enhancement gradually, in order to avoid flickering artifacts.

DC6v and DC6 with ASFU require different tuning of the Guardband Interrupt Delay because they only produce frames for each screen update. For example, DC6v with 30 Hz video playback on a 60 Hz refresh panel will generate only 30 frames per second, causing the Guardband Interrupt Delay to take twice as long to be reached as it would with 60 Hz video playback.

Pipe Palette and Gamma

The display palette provides a means to correct the gamma of an image stored in a frame buffer to match the gamma of the monitor or presentation device. Additionally, the display palette provides a method for converting indexed data values to color values for VGA and 8-bpp indexed display modes. The display palette is located after the plane blender. Using the individual plane gamma enables, the blended pixels can go through or bypass the palette on a pixel-by-pixel basis.

PAL_LGC

PAL_PREC_INDEX

PAL_PREC_DATA

PAL_GC_MAX

PAL_EXT_GC_MAX

GAMMA_MODE

PAL_EXT2_GC_MAX

PRE_CSC_GAMC_INDEX

PRE_CSC_GAMC_DATA

PAL_PREC_MULTI_SEG_INDEX

PAL_PREC_MULTI_SEG_DATA

If any gamma value to be programmed exceeds the maximum allowable value in the associated gamma register, then the programmed value must be clamped to the maximum allowable value.

Programming Modes

The display palette can be accessed through multiple methods and operate in one of four different modes as follows.

8 bit legacy palette/gamma mode:

This provides a palette mode for indexed pixel data formats (VGA and primary plane 8 bpp) and gamma correction for legacy programming requirements.

All input values are clamped to the 0.0 to 1.0 range before the palette/gamma calculation. It is not recommended to use legacy palette mode with extended range formats.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the 256 palette/gamma entries. The 256 entries are stored in the legacy palette with 8 bits per color in a 0.8 format with 0 integer and 8 fractional bits.

The legacy palette is programmable through both MMIO and VGA I/O registers. Through VGA I/O, the palette can look as though there are only 6 bits per color component, depending on programming of other VGA I/O registers.

Direct lookup (10 bit) gamma mode:

This provides the highest quality gamma for pixel data formats of 30 bits per pixel or less.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 1024 gamma entries. The first 1024 entries are stored in the precision palette with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 1024th and 1025th gamma entries to create the result value. The 1025th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For input values greater than or equal to 3.0 and less than 7.0, the input value is used to linearly interpolate between the 1025th and 1026th gamma entries to create the result value. The 1026th entry is stored in the PAL_EXT2_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

All input values are clamped to the greater than -7.0 and less than 7.0 range before the gamma calculation.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Interpolated gamma mode:

This mode uses up to 515 gamma entries and the gamma output gets computed through interpolation between the neighboring LUT entries.

The gamma correction curve is represented by specifying a set of gamma entry reference points spaced equally along the curve for values between -1 and 1. For extended values there is an extended gamma entry reference point at the maximum allowed input value.

For input values greater than or equal to 0 and less than 1.0, the input value is used to linearly interpolate between two adjacent points of the first 513 gamma entries to create the result value. The first 512 entries are stored in the precision palette with 16 bits per color in a 0.16 format with 0 integer and 16 fractional bits (upper 10 bits in odd indexes, lower 6 bits in even indexes). The 513th entry is stored in the PAL_GC_MAX register with 17 bits per color in a 1.16 format with 1 integer and 16 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 513th and 514th gamma entries to create the result value. The 514th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For input values greater than or equal to 3.0 and less than 7.0, the input value is used to linearly interpolate between the 514th and 515th gamma entries to create the result value. The 515th entry is stored in the PAL_EXT2_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

All input values are clamped to the greater than -7.0 and less than 7.0 range before the gamma calculation.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

To program the gamma correction entries, calculate the desired gamma curve for inputs from 0 to 3.0. The curve must be flat or increasing, never decreasing. For inputs of 0 to 1.0, multiply the input value by 512 to find the gamma entry number, then store the desired gamma result in that entry. For inputs greater than 1.0 and less than or equal to 3.0, store the result for an input of 3.0 in the 514th gamma entry.

12 bit Logarithmic Gamma Mode:

This mode provides the highest quality gamma for HDR with 513 LUT entries that use power of two (2^x) spacing instead of uniform segment spacing. Each segment of the table is further divided into equally spaced entries.

x	2^x Segment	# of Entries	Gamma Corr Location	Gamma Entry	Precision
	0	1			
0	1	1			
1	2	2			
2	4	2			
3	8	2			
4	16	2			
5	32	4			
6	64	4			
7	128	4			
8	256	8			
9	512	8			
10	1024	8			
11	2048	16	**Precision Palette**	**0-509**	**0.16**
12	4096	16			
13	8192	16			
14	16384	32			
15	32768	32			
16	65536	64			
17	131072	64			
18	262144	64			
19	524288	32			
20	1048576	32			
21	2097152	32			
22	4194304	32			
23	8388608	32			
24	16777216	1	**PAL_GC_MAX**	510	1.16
25	33554432	1	**PAL_EXT_GC_MAX**	511	3.16
26	67108864	1	**PAL_EXT2_GC_MAX**	512	3.16

	Total entries:	513	

Input Values between 0 (inclusive) and 1.0 (non-inclusive)

For input values greater than or equal to 0 and less than 1.0, the input value is used to linearly interpolate between two adjacent points of the first 511 gamma entries to create the result value. The first 510 gamma entries are stored in the precision palette with 16 bits per color in a 0.16 format with 0 integer and 16 fractional bits (upper 10 bits in odd indexes, lower 6 bits in even indexes). The 511th entry is stored in the **PAL_GC_MAX** register with 17 bits per color in a 1.16 format (1 integer and 16 fractional bits).

Input Values between 1.0 (inclusive) and 7.0 (non-inclusive)

For input values greater than 1.0 and less than 3.0, the input value is used to linearly interpolate between the 511th and 512th gamma entries to create the result value. The 512th entry is stored in the **PAL_EXT_GC_MAX** register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For input values greater than or equal to 3.0 and less than 7.0, the input value is used to linearly interpolate between the 512th and 513th gamma entries to create the result value. The 513th entry is stored in the **PAL_EXT2_GC_MAX** register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

Clamping and Negative Input Values

All input values are clamped to the greater than -7.0 and less than 7.0 range before the gamma calculation.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Programming Gamma Correction Entries

To program the gamma correction entries, calculate the desired gamma curve for inputs from 0 to 7.0. The curve must be flat or increasing, never decreasing.

For inputs of 0 to 1.0 (non-inclusive), program the first 509 gamma entries using the PAL_PREC* registers. Store the result for an input of 1.0 in the 511th gamma entry (**PAL_GC_MAX**).

For inputs greater than 1.0 and less than or equal to 3.0, store the result for an input of 3.0 in the 512th gamma entry (**PAL_EXT_GC_MAX**).

For inputs greater than 3.0 and less than or equal to 7.0, store the result for an input of 7.0 in the 513th gamma entry (**PAL_EXT2_GC_MAX**).

Pipe Control

PIPE_SRCSZ

PIPE_SCANLINE

PIPE_SCANLINECOMP

PIPE_MISC

PIPE_FRMTMSTMP

PIPE_FLIPTMSTMP

PIPE_BOTTOM_COLOR

PIPE_FLIPCNT

PIPE_FRMCNT

PIPE_FLIPDONETMSTMP

PIPE_MISC2

PIPE_STATUS

Pixel Passthrough Operation

For modes of operation where the input image CRC needs to match the output image CRC (i.e., the pixels need to flow through the pixel pipe unmodified from frame buffer to the port), the following programming needs to be done:

1. Use only a single Plane with no cursor (**CUR_CTL**)
2. Use a fixed point, non-planar pixel format without Alpha (**PLANE_CTL**). Make sure the frame buffer format matches the port output format
3. Disable all color correction (i.e., CSC, Gamma, etc.), Image Enhancement, Lace, Scaling, compression and dithering within the Plane and Pipe. See the other sub-sections of the Pipe/Planes chapters for more details:
    1. Universal Plane
    2. Luminance Mapping
    3. Color Space Conversion
    4. Pipe Color Gamut Enhancement
    5. Pipe Palette and Gamma
    6. Pipe LDPST
    7. Pipe 3D LUT
    8. Pipe DPST
    9. Pipe Scaler
    10. DSC (Display Stream Compression)
4. If the Plane that is being used (i.e. enabled) is an HDR Plane, then the HDR Mode bit within **PIPE_MISC** must be set. Otherwise, the HDR Mode bit of PIPE_MISC must be **cleared**.

5. Disable (i.e. truncate) Pixel Rounding (**PIPE_MISC**)

Pipe Scaler

Each scaler has its own set of registers.

Scaler
PS_PWR_GATE
PS_WIN_POS
PS_WIN_SZ
PS_CTRL
PS_VSCALE
PS_HSCALE
PS_VPHASE
PS_HPHASE
PS_ECC_STAT
PS_ADAPTIVE_CTRL
PS_COEF_INDEX
PS_COEF_DATA
SCALER_COEFFICIENT_FORMAT
PS_PROG_HSCALE PS_PROG_VSCALE

Scaler Programmed Coefficients

Two sets of programmed coefficients are available for each scaler. The horizontal filter and vertical filter can be configured individually to use one of these 2 sets. When used for YUV planar format plane scaling, the Y and UV scalers can be configured individually to use one of the 2 sets. Scaler coefficients are accessed through their respective index and data registers following the mapping shown below. The coefficients must be programmed in the SCALER_COEFFICIENT_FORMAT.

17 phase of 7 taps requires 119 coefficients in 60 dwords per set. The letter represents the filter tap (D is the center tap) and the number represents the coefficient set for a phase (0-16).

Coefficient Set		
Index Value	**Data Value Coefficient 2**	**Data Value Coefficient 1**
00h	B0	A0
01h	D0	C0
02h	F0	E0
03h	A1	G0
04h	C1	B1
...	...	...
38h	B16	A16

Coefficient Set		
39h	D16	C16
3Ah	F16	E16
3Bh	Reserved	G16

Programmed Coefficient Usage in YUV420 Encode

When a Pipe Scaler is encoding a YUV420 format (i.e., converting a YUV444 format to a YUV420 format), then the Scaler finite impulse response (FIR) filters are operating at a downgraded mode of operation (the vertical FIR will populate every other tap with a pixel and the horizontal FIR will only populate the inner most taps with pixels). If programmed coefficients are being used, then Software will have to comprehend the following when creating the coefficient tables:

1. Both coefficient sets will need to be used (one for the vertical filter and one for the horizontal filter)
2. The coefficient tables will need to be modified to match how pixels are populated within the FIR taps. Assuming a non-modified table entry is programmed with Ax, Bx, Cx, Dx, Ex, Fx, Gx where "x" is the table row number:
    1. For the vertical coefficient set: 0, Cx, 0, Dx, 0, Ex, 0
    2. For the horizontal coefficient set: 0, Bx, Cx, Dx, Ex, Fx, 0

Vertical Coefficient Set		
Index Value	**Data Value Coefficient 2**	**Data Value Coefficient 1**
00h	C0	0
01h	D0	0
02h	E0	0
03h	0	0
04h	0	C1
05h	0	D1
06h	0	E1
07h	C2	0
08h	D2	0
09h	E2	0
...	...	...

Horizontal Coefficient Set		
Index Value	**Data Value Coefficient 2**	**Data Value Coefficient 1**
00h	B0	0
01h	D0	C0
02h	F0	E0
03h	0	0
04h	C1	B1
05h	E1	D1
06h	0	F1
07h	B2	0
08h	D2	C2
09h	F2	E2
...	...	...

Nearest-neighbor scaling (Integer scaling ratios)

Nearest neighbor scaling can be used to maintain the rendering intent of some classic games in integer upscaling scenarios. For enabling nearest-neighbor scaling, the scaler must be set to use "programmed" mode with the center tap (Dxx) values set to 1 and all other values set to 0. The following coefficients values must be used with the coefficients programmed in the SCALER_COEFFICIENT_FORMAT.

Coefficient Set		
Index Value	**Data Value Coefficient 2**	**Data Value Coefficient 1**
00h	B0 = 0	A0 = 0
01h	D0 = 1	C0 = 0
02h	F0 = 0	E0 = 0
03h	A1 = 0	G0 = 0
04h	C1 = 0	B1 = 0
...	...	...
38h	B16 = 0	A16 = 0
39h	D16 = 1	C16 = 0
3Ah	F16 = 0	E16 = 0
3Bh	Reserved	G16 = 0

Register Double Buffering

Prior to D11, the Scaler has the following double buffer trigger points:

1. When the Pipe is disabled:
    a. Any write to the PS_CTRL register (will refer to this as Control DB trigger)
    b. Any write to the PS_WINSZ register (will refer to this as WinSize DB trigger)
2. When the Pipe is enabled:
    a. For registers that are part of the Control DB trigger group, the DB trigger point is on the rising edge of V. Blank when double buffering is armed
    b. For registers that are part of the WinSize DB trigger group, the DB trigger point is after the Frame Start when double buffering is armed

From D11 and onwards, the Scaler has the following double buffer trigger points:

1. When the Pipe is disabled:
    a. Any write to the PS_CTRL register (Control DB trigger)
    b. Any write to the PS_WINSZ register (WinSize DB trigger)
2. When the Pipe is enabled, the DB trigger point is on the rising edge of V. Blank when double buffering is armed, and double buffering is allowed (Allow Double Buffer Update Disable = 1 in PS_CTRL)

When the Pipe is enabled, any write to the PS_WINSZ will arm the double buffering for the Scaler registers.

Once double buffering is armed, the disarming point is dependent on the Display generation.

- Prior to D11: Double buffering is disarmed on the next WinSize DB trigger (i.e. after Frame Start)
- D11+ : Double buffering is disarmed on the next rising edge of V. Blank where double buffering is allowed

As implied above, the double buffering for each of the Scaler registers is dependent on the trigger group it is within (i.e. Control or WinSize).

Register	DB Trigger
PS_PWR_GATE	Control
PS_CTRL	Control
PS_VPHASE	WinSize
PS_HPHASE	WinSize
PS_ADAPTIVE_SET_*_CTRL	WinSize
PS_WINPOS	WinSize
PS_WINSZ	WinSize

Note that the Programmable Coefficient sets accessed through the PS_COEF_SET_*_INDEX and PS_COEF_SET_*_DATA registers are on the Control DB trigger.

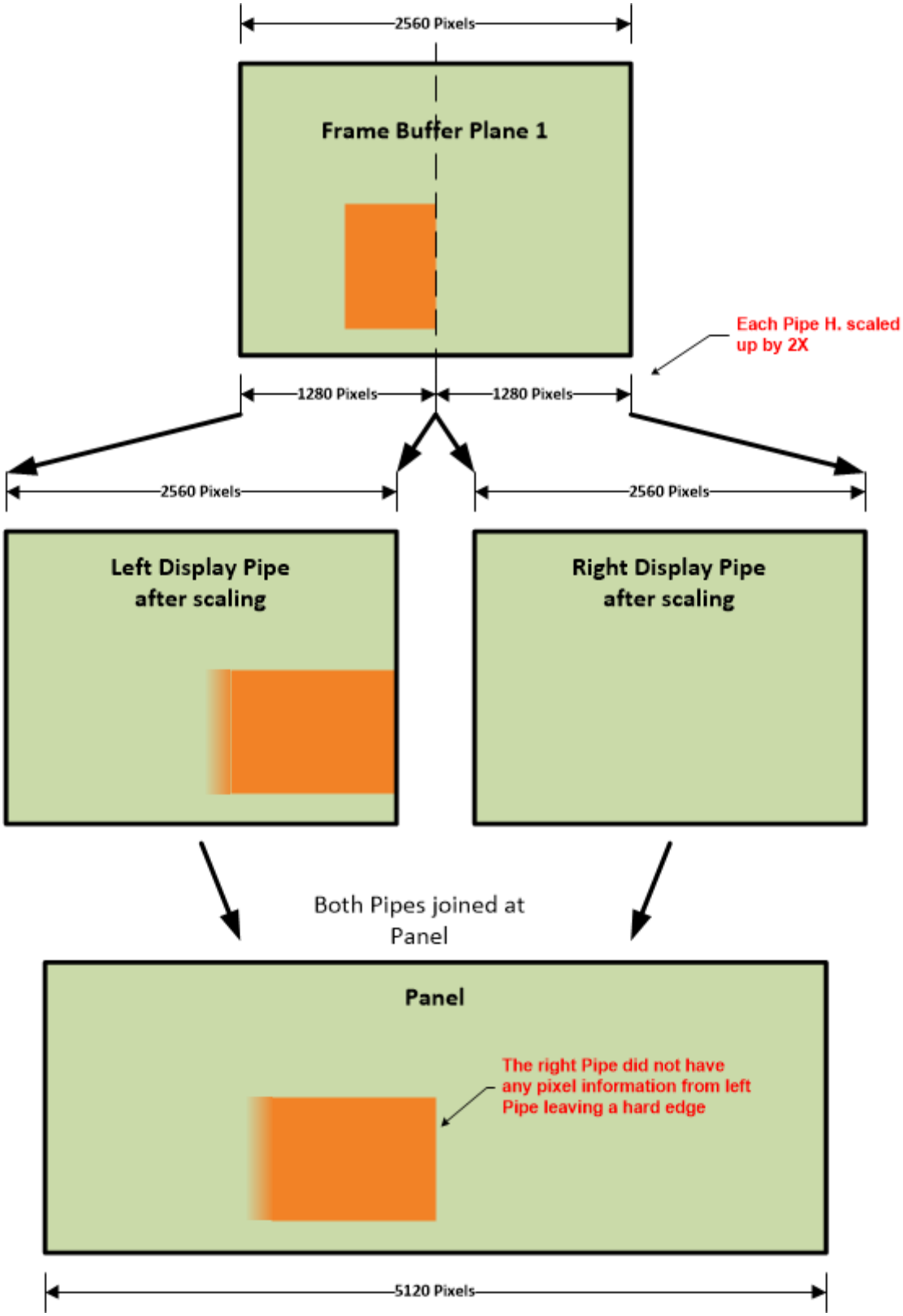
Programming Sequence

The below sequence should be followed when programming the Scaler to ensure proper double buffering:

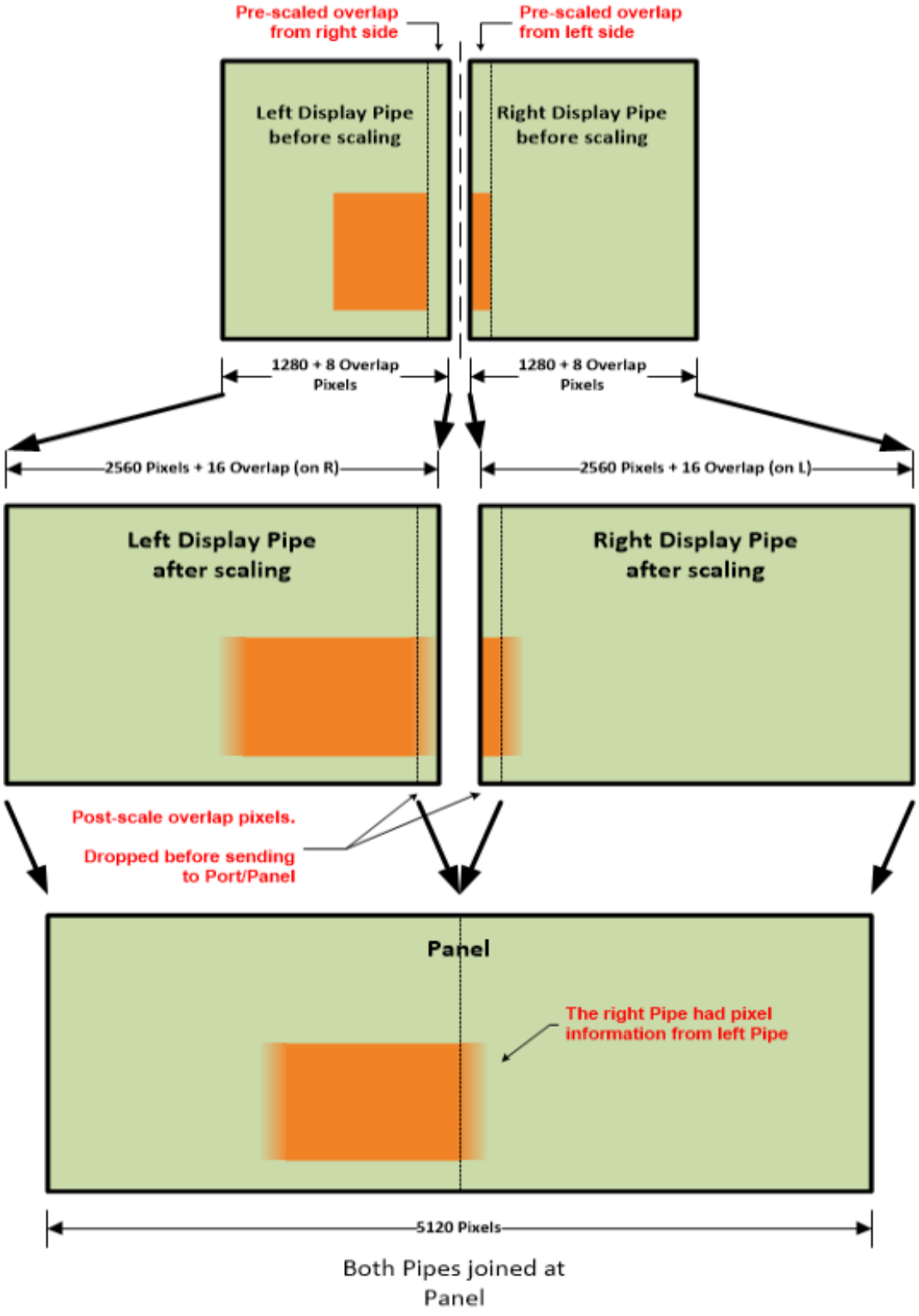
- Configure the Scaler's Programmable Coefficient sets through PS_COEF_SET_*_INDEX and PS_COEF_SET_*_DATA registers, if using Programmable Coefficients
- Configure power gating control of the Scaler SSA's (PS_PWR_GATE), if the Scaler is not already enabled
- Enable and/or configure Scaler (PS_CTRL), if needed
    - If the Scaler is not already enabled, the Scaler will begin the process of powering up the Scaler SSA's
    - Software doesn't have to write to this register to enable Control DB trigger registers (i.e. Programmable Coefficient sets) if the Scaler and Pipe are both enabled
- Configure initial phases (PS_*PHASE), adaptive control (PS_ADAPTIVE_SET_*_CTRL), and window position (PS_WINPOS), if needed
- Configure/write to the PS_WINSZ
    - For pre-D11 products, the write to this register should only be done outside of the V. Blank regime if the Pipe is enabled. There are no restrictions for D11+ products
    - If Software has changed any PS_* register programming and the Pipe is enabled, then Software has to write to this register regardless of whether the value is changing, or not.

Tiled Scaling (Seam Removal)

There are certain usage cases where an image from memory will be horizontally split across two Pipes, scaled up, and then joined at the Port/Panel. When the image is being horizontally scaled up across the seam of the split image if the Scalers within each Pipe do not have some additional pixels from the other Pipe's image, then an artifact at the seam can occur.


2560 Pixels
Frame Buffer Plane 1
Each Pipe H. scaled up by 2X
1280 Pixels
1280 Pixels
2560 Pixels
2560 Pixels
Left Display Pipe after scaling
Right Display Pipe after scaling
Both Pipes joined at Panel
Panel
The right Pipe did not have any pixel information from left Pipe leaving a hard edge
5120 Pixels

By adding some overlapping pixels of the split image around the seam, the Scalers within each Pipe will be able to correctly filter across the seam. At the end of the Pipe (before the image is delivered to the Port) the post-scaled overlap pixels will be dropped.



To perform the cross-seam scaling, Software will be responsible for the following:

- It will calculate the pre and post scale excesses needed for each Pipe
- It will include the pre-scale excess within the pre-scaled Horizontal image sizes (e.g. the Horizontal Source Size of the PIPE_SRCSZ register)
- It will include the post-scale excess within the Scaler's Window Size register (PS_WIN_SZ)
- It will program the post-scale excess within the PIPE_SEAM_EXCESS register
- It will program the Horizontal Active size at the transcoder (TRANS_HTOTAL) to be the Panel width
    - This should **not** include any seam pixels

The following sections will discuss how Software will determine the pre/post excess sizes and the initial phase needed for the Scaler processing the right side image.

Location of Seam and Splitting Source Image

When the location of an image in memory is going to cross the seam between the two Pipes, then the PLANE_SIZE, PLANE_POS, and PLANE_OFFSET registers of the Planes carrying the image (one within each Pipe) will determine how that image in memory is split across the two Pipes.

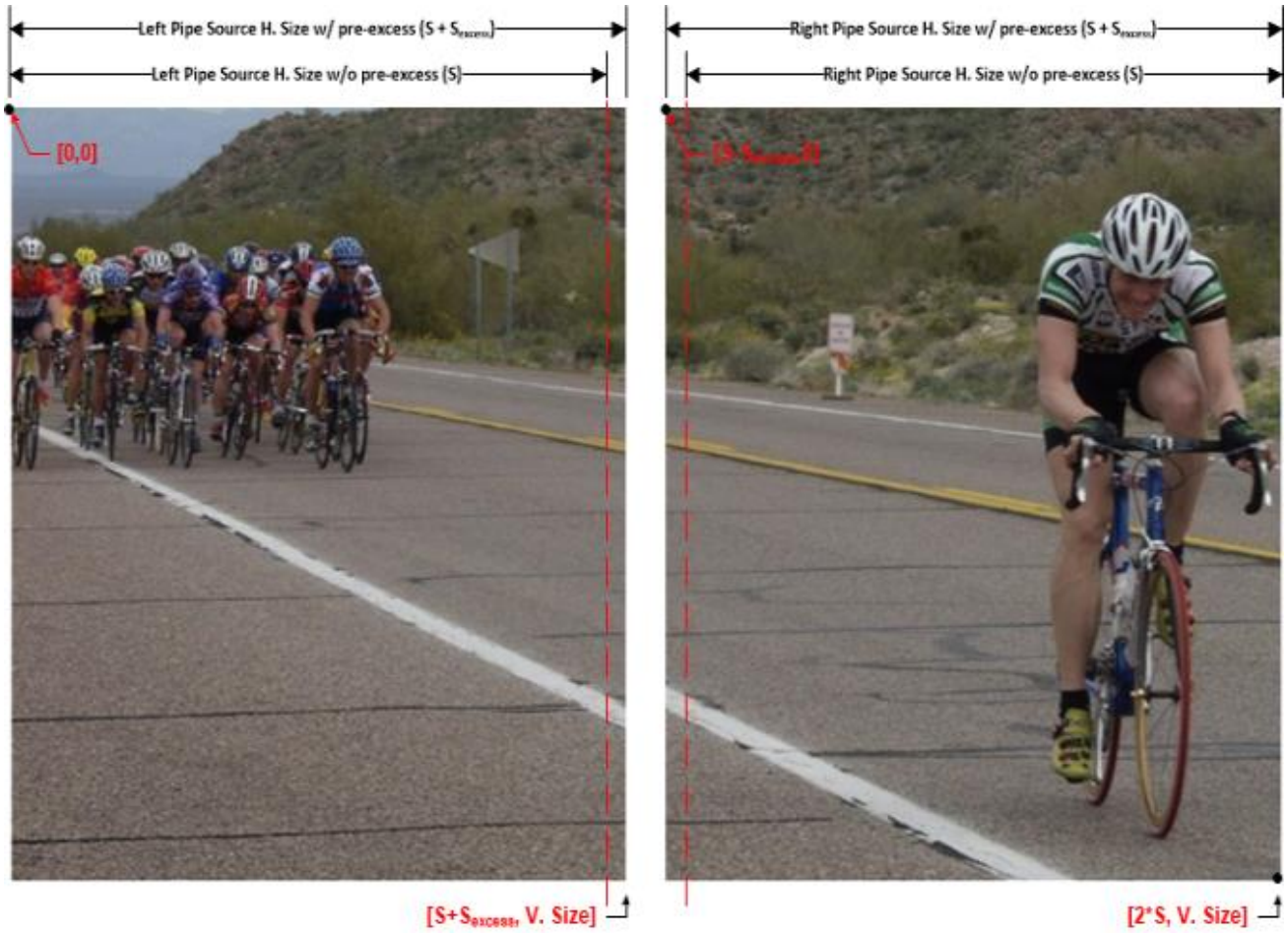
Seam Location

The location of the seam is relative to the left or right side of the Pipe Source Size window where the horizontal Pipe Source Size will include the pre-scaled excess (a.k.a. Source Excess, or S_{excess})

$$\text{H. Pipe Source Size} = \text{Target H. Pipe Source Size (S)} + \text{Source Excess } (S_{excess})$$

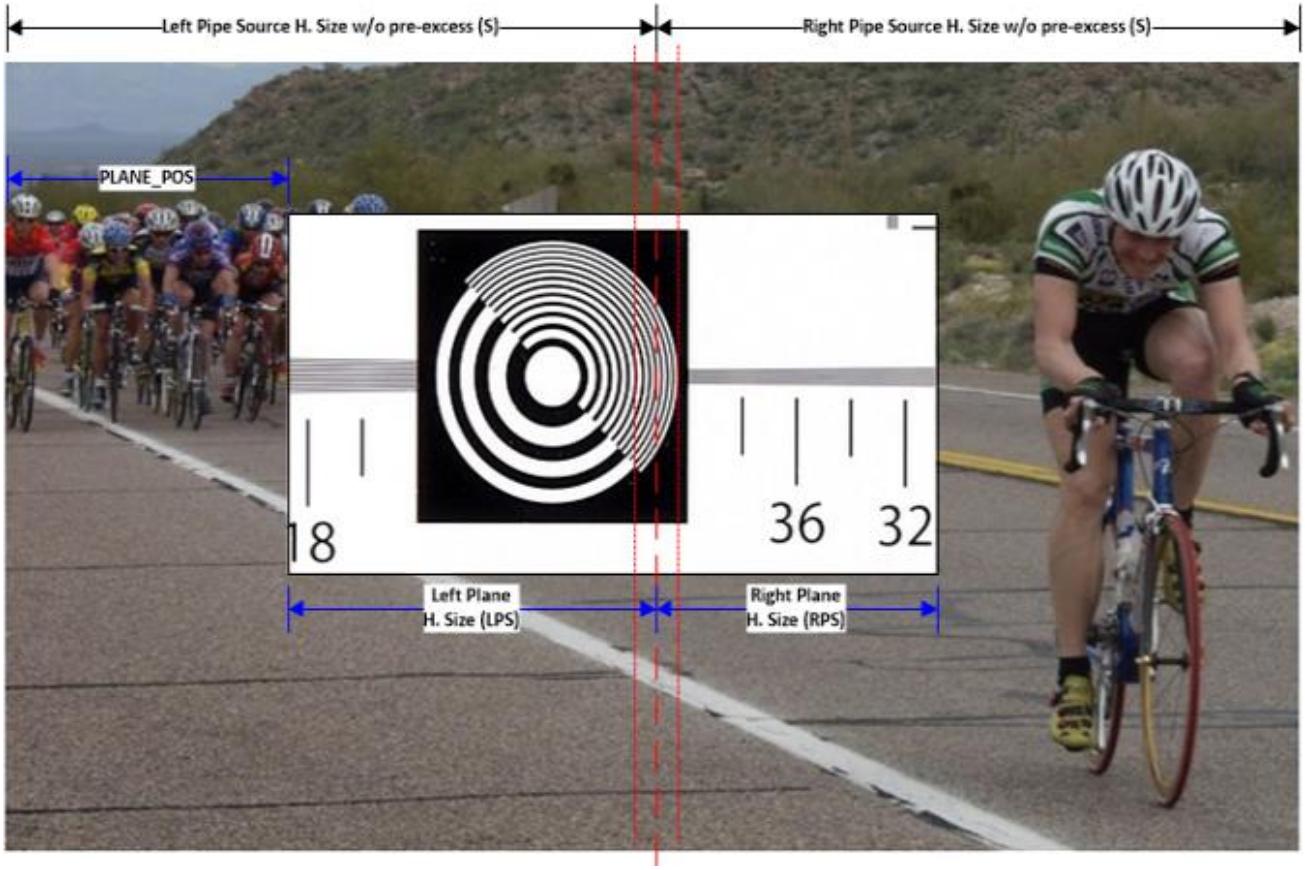
For the left-side Pipe, the seam will be located on the right-side of its source window, and for the right-side Pipe, the seam will be located on the left-side of its source window.


[0,0]
Seam
[2* S, V. Size]



Note that this discussion assumes symmetric horizontal Pipe Source Sizes without the source excess (i.e., S) across both Pipes. This is not a requirement, but it will be left to the reader to extrapolate the equations within this section for asymmetric S terms.

Locating the seam within an image that spans the horizontal Pipe Source Size across both Pipes (e.g., the desktop/background image) is simply a function of the Pipe. But, if a Plane image is smaller than the combined horizontal Pipe Source Sizes, then the location of the seam is dependent on the offset of the Plane within the Pipe Source Window.



So, if the addition of the Plane Position (PP) and the Full H. Plane Size (HPS) is greater than S, then the following equations define where the seam is located within the image in memory.

HPS = Left Plane Size (LPS) + Right Plane Size (RPS)

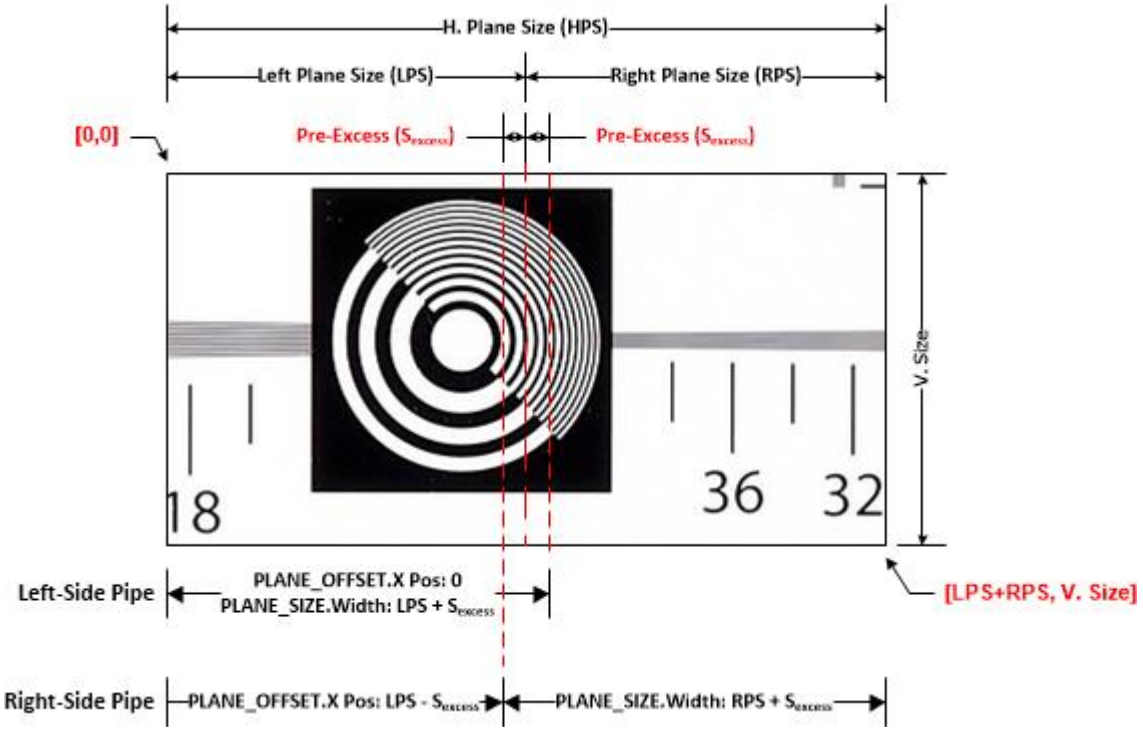
S = PP + LPS => **LPS = S - PP**

RPS = HPS - (S - PP)

Note that the Source Excess (S_{excess}) is a constant regardless of where the Plane image is located.

Splitting the Image

Now that the location of the seam is known, splitting the image across both Pipes can be performed. This is done using the PLANE_OFFSET and PLANE_SIZE registers.



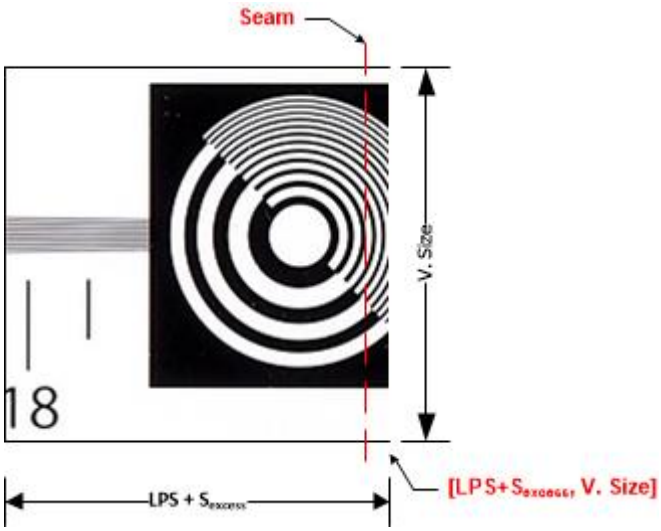
Using the figure above as a reference, to split the image in memory across the two Pipes.

Left PLANE_OFFSET:

- Starting X Position = H. Cropping Position (HCP) + 0
- Starting Y Position = V. Cropping Position (VCP) + 0

Left PLANE_SIZE:

- Width = Left Plane Size (LPS) + Pre-Excess Size (S_{excess})
- Height = V. Size

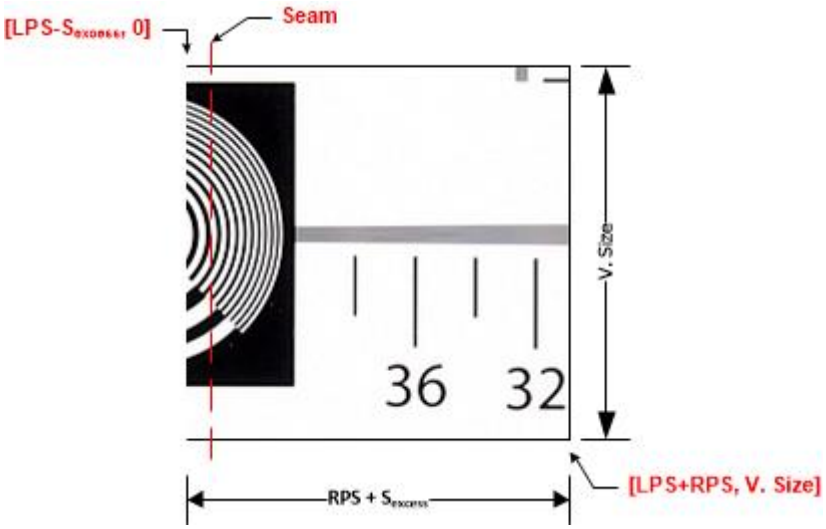


Right PLANE_OFFSET:

- Starting X Position = HCP + LPS - S_{excess}
- Starting Y Position = VCP + 0

Right PLANE_SIZE:

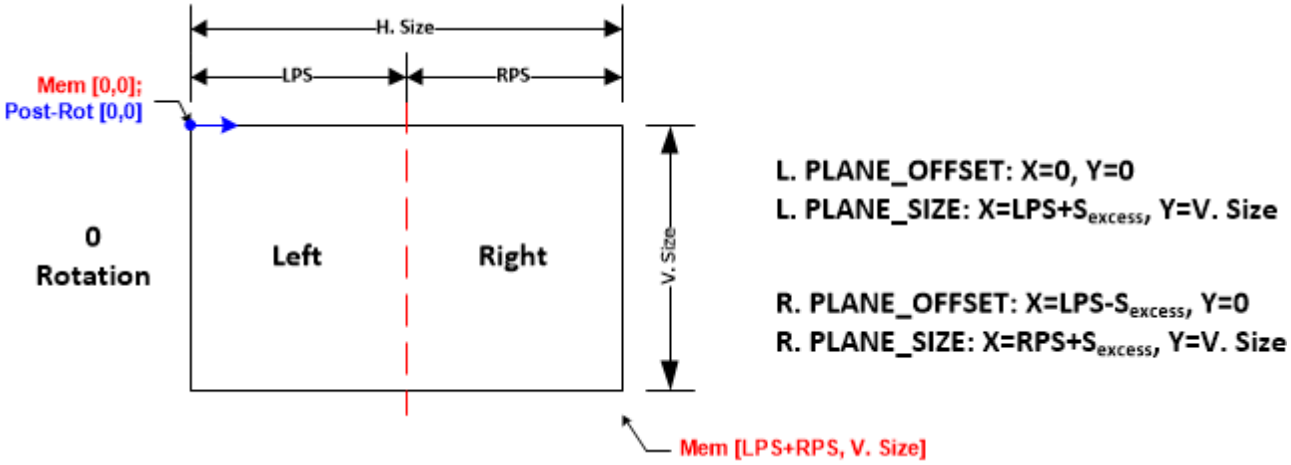
- Width = Right Plane Size (RPS) + S_{excess}
- Height = V. Size



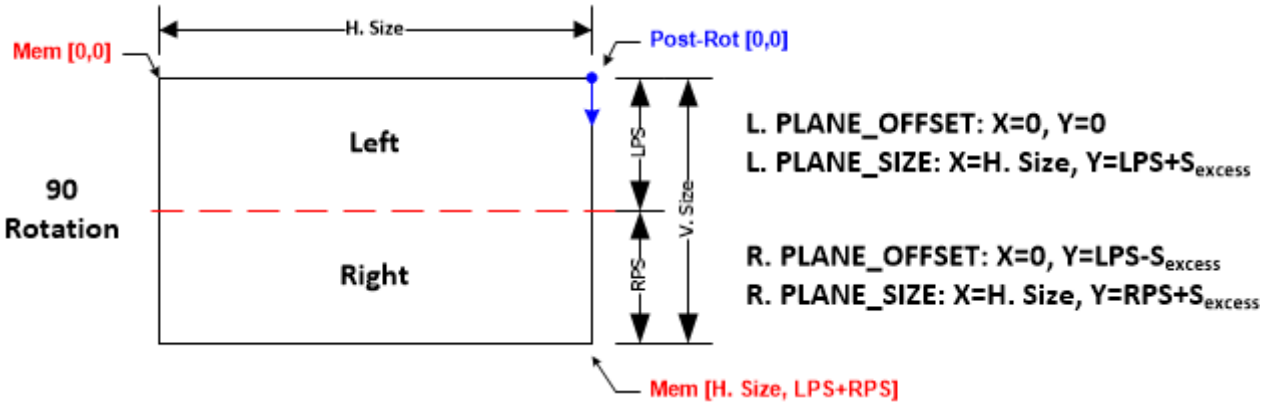
Rotation

The above discussion for splitting an image is when there is no rotation. If the image is to be rotated, then the programming of the PLANE_OFFSET and PLANE_SIZE needs to be adjusted accordingly. Since rotation is applied after the Plane Position and Size, the programming of these registers "rotates" to account for the image rotation.

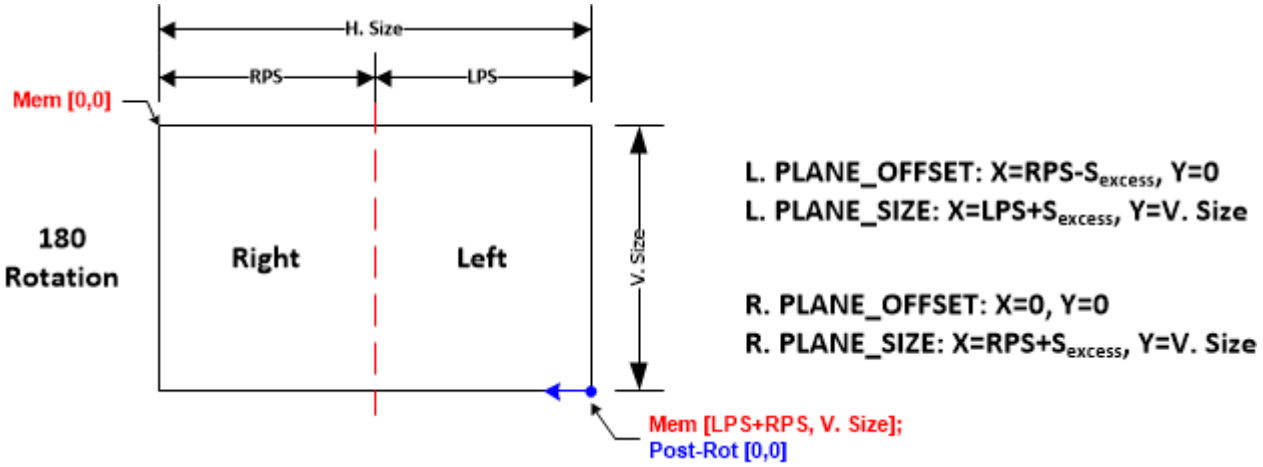
If we look at an image in memory that is going to be split without rotation, the regions of the image that are sent down each Pipe is shown below. Note that the blue dot and arrow represent the post rotation origin of the image and direction of scan.



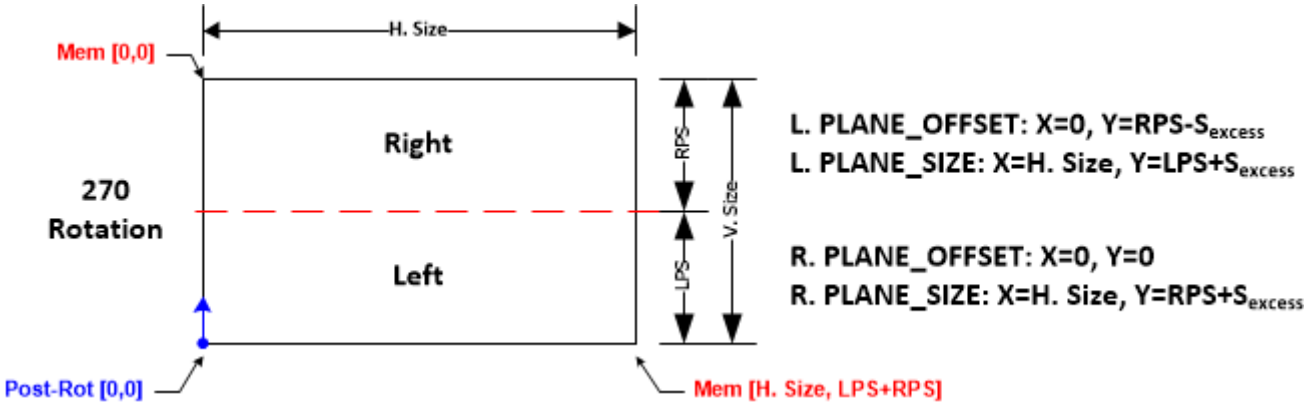
When the image is rotated by 90^o, then the horizontal and vertical sizes programmed within the Plane/Pipe are swapped. So, the left and right regions swap as well.



When rotating another 90^o (i.e. 180^o rotation), then the left and right regions swap positions with respect to the no rotation scenario.



Finally, when the image is rotated 270^o then the left and right regions swap positions with respect to the 90^o rotation scenario.

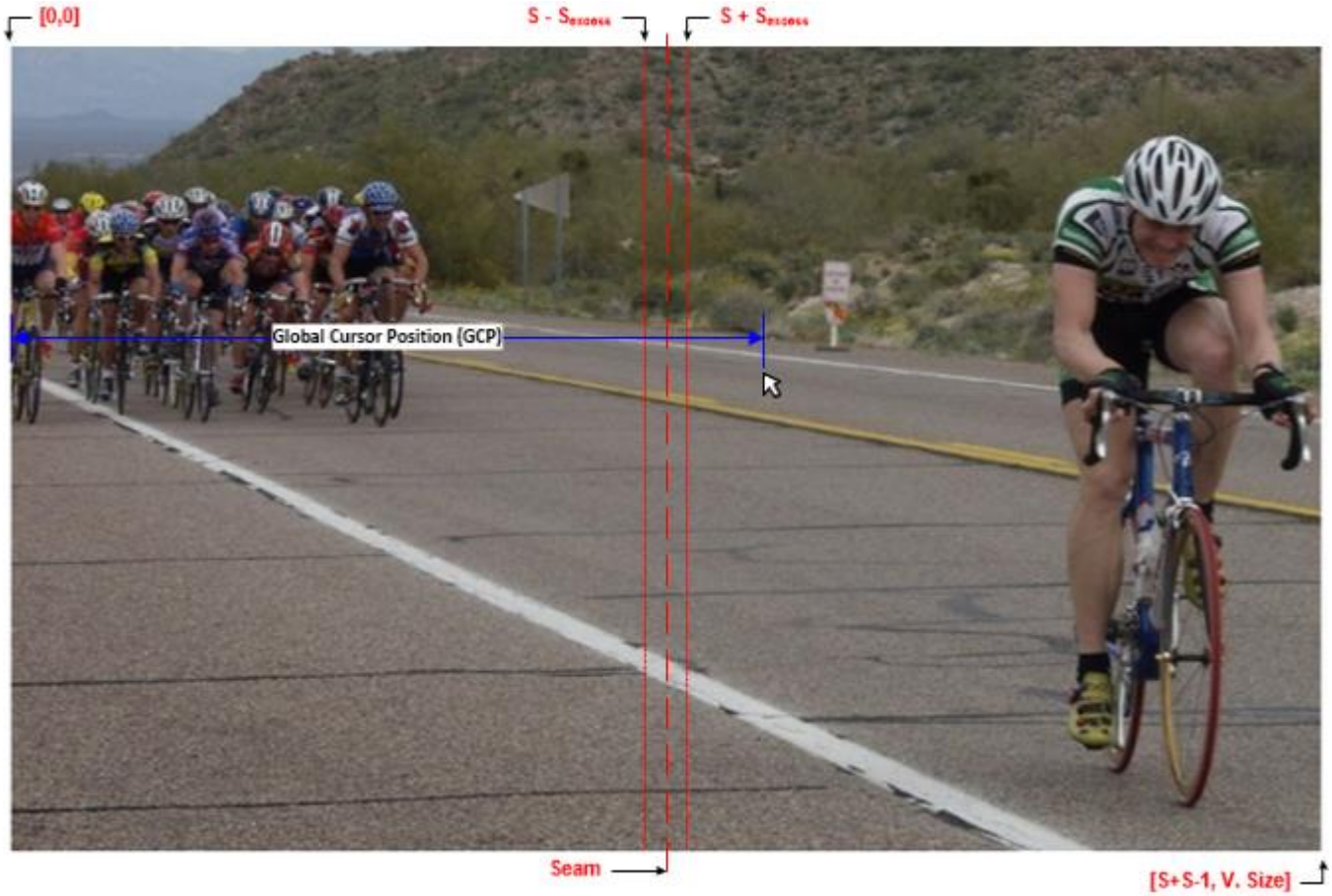


Note that Software should ensure that the horizontal Plane position for the image in the right Pipe is zero (i.e., there should be no positional offset of the image with respect to the seam)

Refer to the table within the Source Image Manipulation below for a summary of the Plane offset and size programming.

Cursor

The mechanism that Software uses to determine the horizontal position of the Cursor across both Pipes is beyond the scope of this document. However, to explain how the Cursor is enabled/programed around the seam, this document will use a global coordinate system that spans across both Pipe source windows.

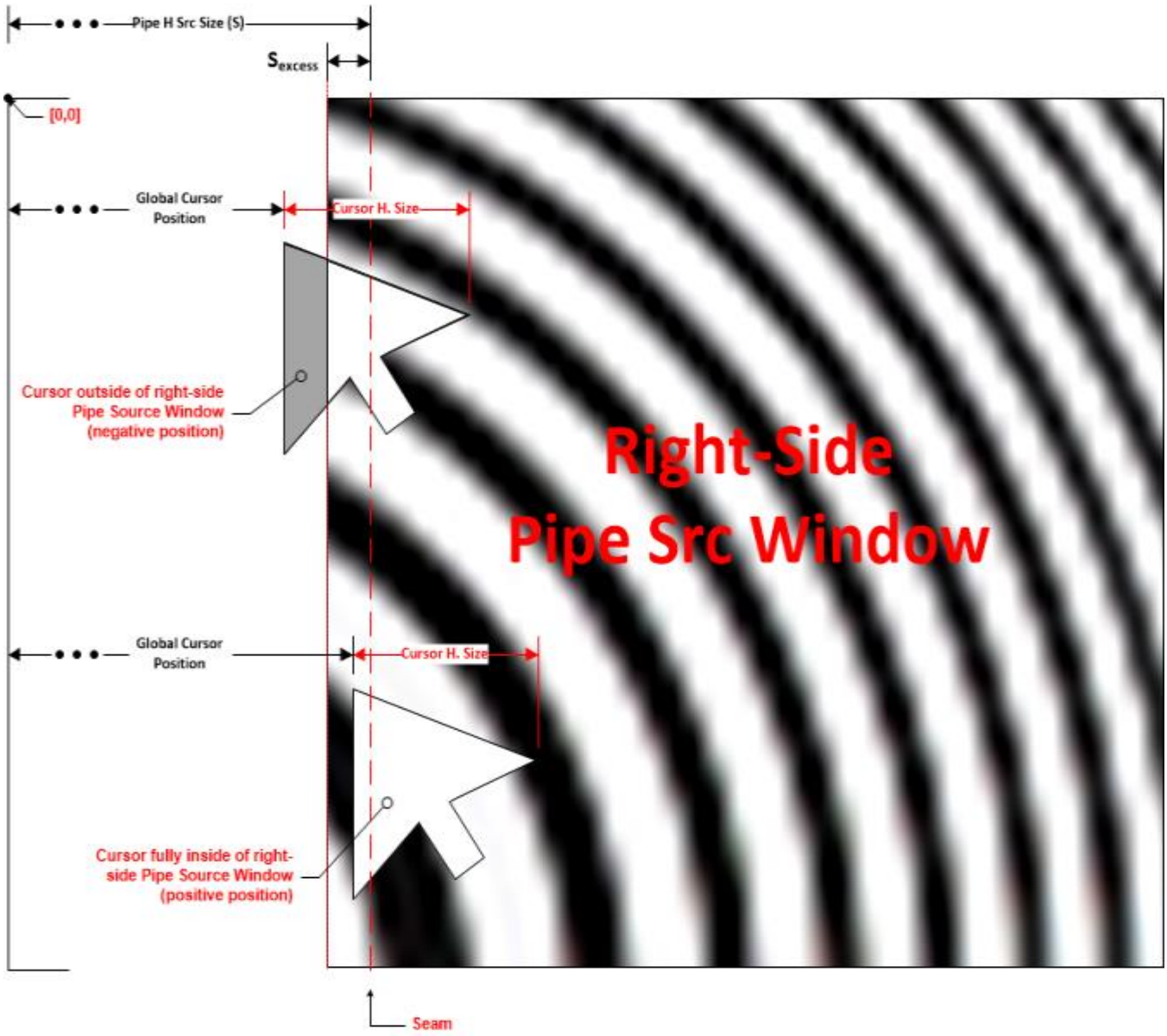


The horizontal position of the Cursor will be referred to as the global cursor position (GCP) that will span across both Pipe source windows from 0 to (2*S)-1.

When the Cursor is outside of the seam "window" (S +/- S_{excess}), then only one Pipe will have its Cursor enabled. As the Cursor moves into the seam window, then both Pipes will need to enable its Cursor. Determining when the Cursor is within this window is dependent on the global cursor position and the Cursor size (CS).

Global Cursor Position	Left Pipe Cursor	Right Pipe Cursor
GCP < (S - S_{excess} - CS)	Enabled	Disabled
(S - S_{excess} - CS) <= GCP < (S + S_{excess})	Enabled	Enabled
GCP >= (S + S_{excess})	Disabled	Enabled

The positioning of the Cursor will be dependent on where the Cursor is within the seam window. The left-side Cursor position is directly based off of the global cursor position, but the right-side Cursor is a little more involved.



The following table illustrates how to determine the Cursor position when it is crossing the seam.

Global Cursor Position	Left Pipe Cursor Position	Right Pipe Cursor Position
GCP < (S - S_{excess} - CS)	Magnitude = GCP Sign = SW determines	N/A (Disabled)
(S - S_{excess} - CS) <= GCP < (S - S_{excess})	Magnitude = GCP Sign = Positive	Magnitude = S - S_{excess} - GCP Sign = Negative
(S - S_{excess}) <= GCP < (S + S_{excess})		Magnitude = GCP - S - S_{excess} Sign = Positive
GCP >= (S + S_{excess})	N/A (Disabled)	

Calculating Pre and Post Excess Sizes

To get an accurate sampling across the seam, the number of pre-scaled excess pixels (a.k.a. Source Excess, S_{excess}) needs to (at a minimum) fill the number of FIR taps to the left or right of the horizontal FIR's Center Tap.

$$\mathbf{S_{excess} >= (NUMTAPS - 1)/2}$$

Note that the Display Pipe requires an even number of line pixels, so the source excess may need to be rounded up to the next even number.

To determine the number of post-scaled excess pixels (a.k.a. Destination Excess, D_{excess}) we can use the following equation:

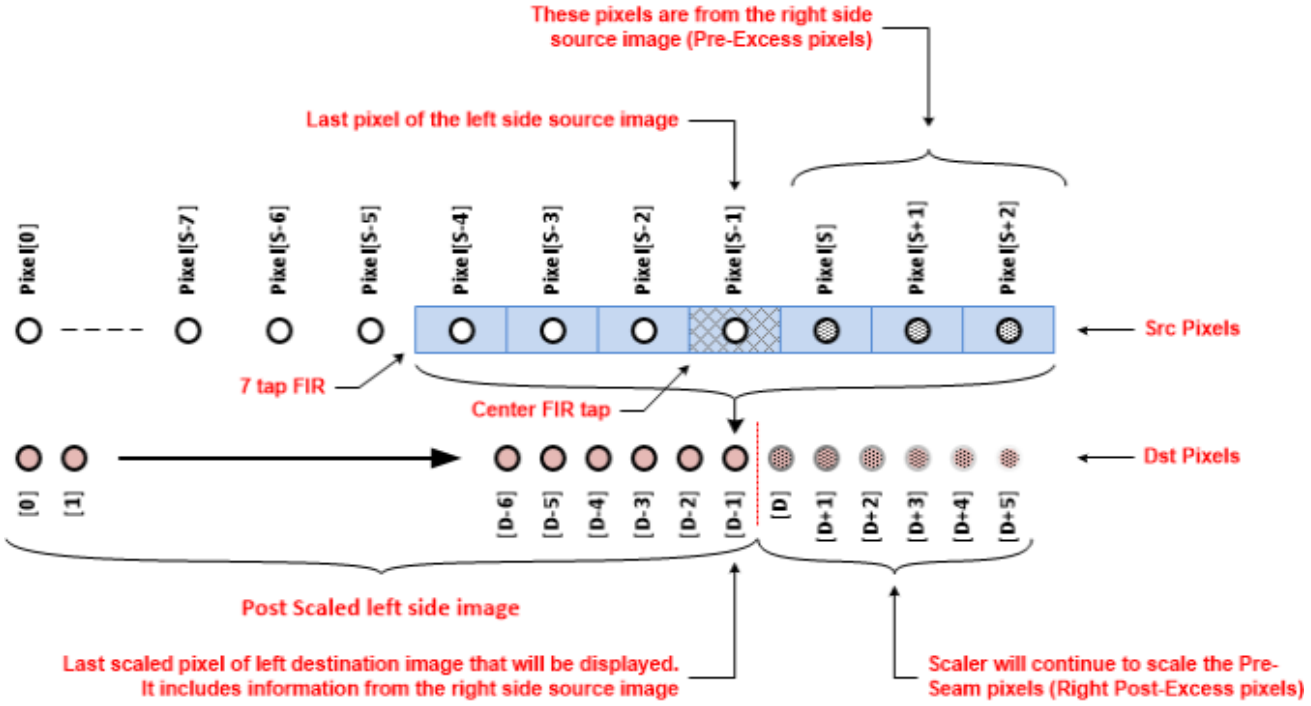
$$SF_{target} = (S + S_{excess}) / (D + D_{excess})$$

$$\mathbf{D_{excess} = Round(((S + S_{excess}) / SF_{target}) - D)}$$

Where:

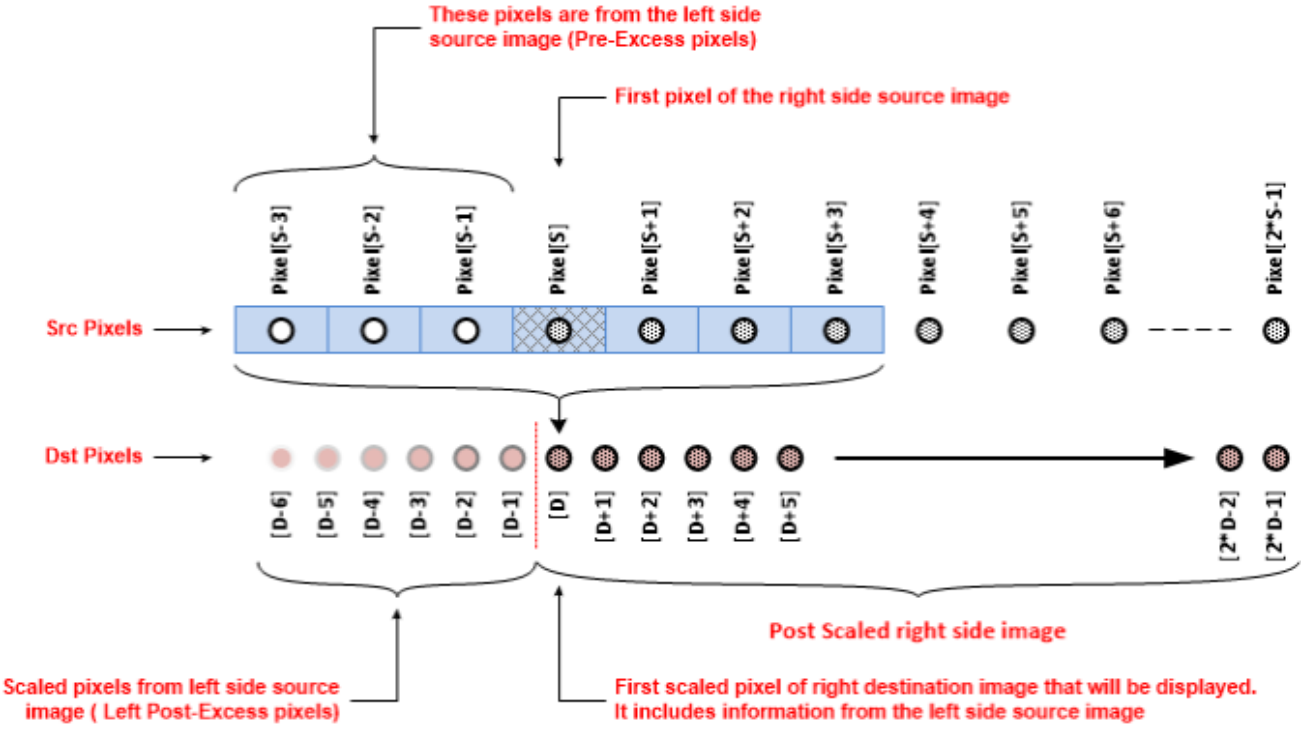
- S is the Source size in pixels per Pipe (one-based and without excess pixels)
- D is the Destination size in pixels per Pipe (one-based and without excess pixels)
- SF_{target} is the target Scale Factor (i.e. S/D)

Note that the D_{excess} must be a multiple of DISP_NUMPPC since the pixel dropping logic works at that granularity.



The above illustrates the Scaler output for the left side image (seam is on the right side). As the last pixel of the left side source image moves into the center tap of the Scaler's FIR, all of the taps to the right of the center tap contain pixels from the right side source image (Pixels S to S+2). The resulting destination pixel (this is the last displayable pixel) will contain all of the necessary information from the right side of

the source image. All of the pixels beyond the last pixel of the destination image are the post excess pixels and those will be dropped.

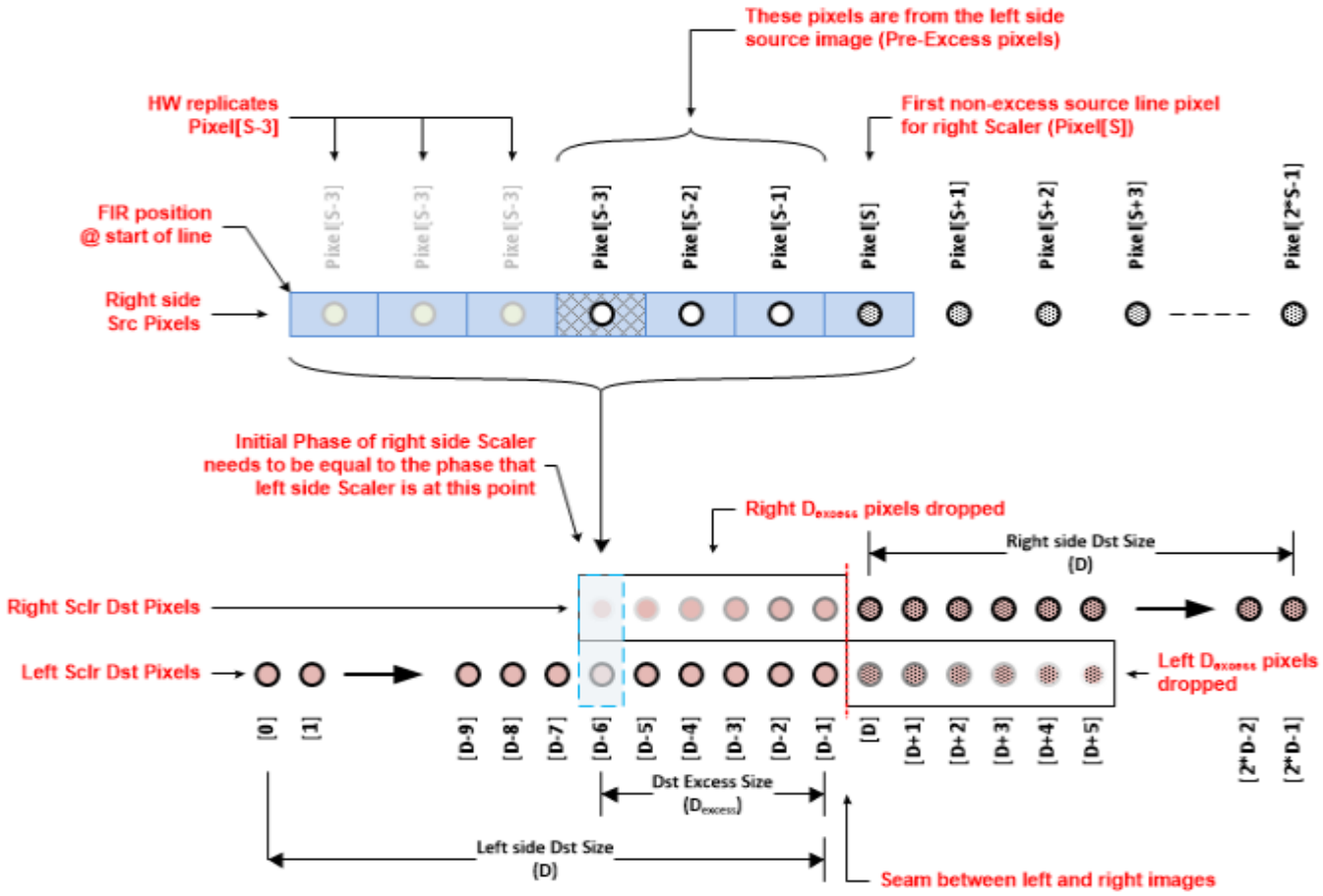


The above illustrates the Scaler output for the right-side image (seam is on the left side). As the FIR progresses, the first pixel of the right-side source image eventually moves into the center tap and all of the taps to the left of the center tap contain pixels from the left side source image (Pixels S-3 to S-1). All pixels up to this point are the post excess pixels that will be dropped. The resultant destination pixel when the first right side source image pixel makes it to the center pixel contains all of the necessary information from the left side of the source image.

Calculating Initial Phases

To ensure the image from both Pipes appears to be scaled by a single Scaler, the horizontal initial phase for the Scaler processing the right-side image will need to be set such that the phase of the horizontal FIR of the right Scaler picks up where the horizontal FIR of the left Scaler leaves off.

When the horizontal filter starts processing a line, it will load the first source pixel of the line into the Center Tap of the FIR and then apply an initial phase from that point (note that the Scaler can only apply positive initial phases). For the right-side Scaler, the first pixel of the source line will be the left most Source Excess pixel (i.e. a pixel from the left source image). To match the phase of the horizontal FIR of the right-side Scaler at the **beginning** of its source line with the phase of the horizontal FIR of the left-side Scaler at the **end** of its source line we need to determine the phase of the left-side Scaler's horizontal FIR (D_{excess}-1) pixels **before** it reaches the final Destination pixel (D-1).



The following equation is used to determine the phase that the right-side Scaler needs to start at.

$$\mathbf{P_{right} = SF_{actual} * (D - D_{excess})}$$

Notes:

1. To get an accurate initial phase, need to use the actual scale factor (SF_{actual}) that the Scaler on the left is using. Depending on configuration, this is not always the target scale factor
   a. If the Scaler HW is calculating the Scale Factor: $SF_{actual} = (S + [Left, Right]\ S_{excess}) / (D + D_{excess})$
   b. If the Scaler HW is using the programmed Scale Factor: SF_{actual} = PS_PROG_HSCALE
      i. The PS_PROG_HSCALE register should equal the target Scale Factor (SF_{target})
2. The Scale Factor has a precision of 2.14, regardless of how it is calculated. However, HW will automatically round the Scale Factor (Software should do the same)

The absolute phase difference between the left and the right will be used as the Horizontal Initial Phase (HIP) to be applied to either the left or right side.

$$\mathbf{Absolute\ Horizontal\ Initial\ Phase = |\ (S - S_{excess}) - P_{right}\ |}$$

The side that the initial phase is applied will depend on which side is ahead of the other.

- If the left-side Scaler phase is less than or equal to the source pixel that the right-side Scaler starts at (i.e. (S - S_{excess}) - P_{right} is positive) then the HIP will be applied to the left-side Scaler
- Otherwise, the HIP will be applied to the right-side Scaler

Notes:

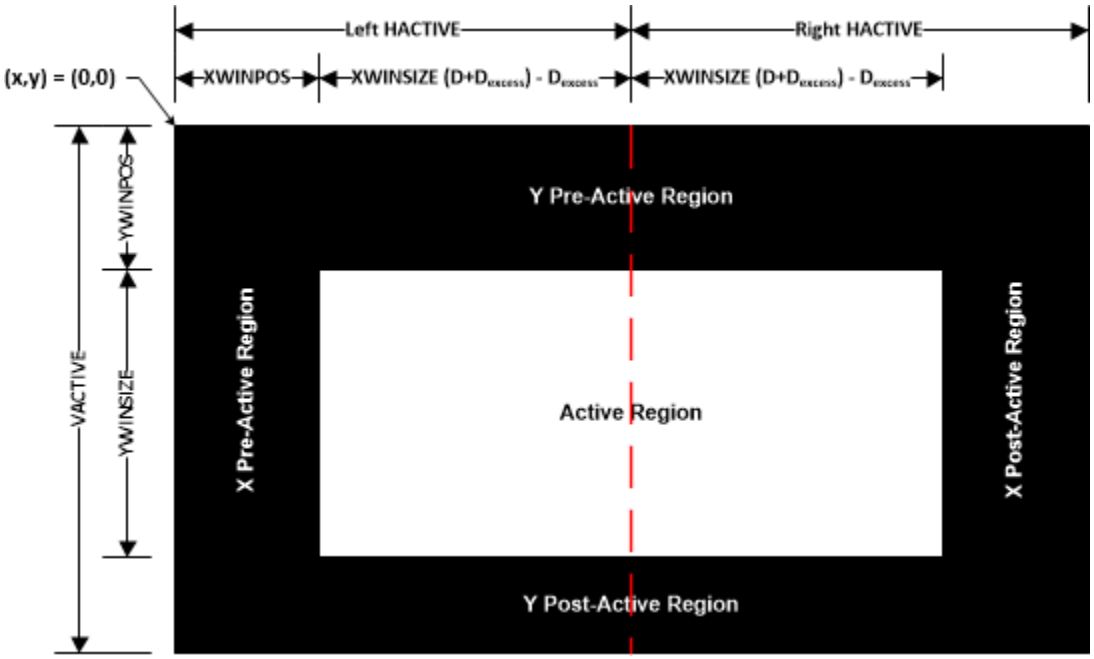
1. Make sure to set the Initial Phase Trip bit of PS_HPHASE
2. The act of applying an initial phase will cause the image (on the left or the right) to appear to "shift". The shifting will be more evident for the left-side.

The following table illustrates an example of a 1920x1080 source image that is split across two Pipes with Pipe Source Sizes of 960 + S_{excess}.

		Notes
Source Size (S)	960	Horizontal Pipe Source Size w/o S_{excess}
Source Excess (S_{excess})	4	EVEN((NUMTAPS -1)/2)
Destination Size (D)	2576	Number of destination pixels per Pipe w/o D_{excess}
Target Scale Factor (SF_{target})	0.372670807	S/D
Actual Scale Factor (SF_{actual})	0.372497559	HW calculated by left Scaler: (S + S_{excess}) / (D + D_{excess}) Result is rounded to 2.14 precision
Destination Excess (D_{excess})	12	ROUND(((S + S_{excess}) / SF_{target}) - D)
Initial Phase needed on Right (P_{right})	955.0837402	SF_{actual} * (D - D_{excess})
First Src Pixel on Right	956	S - S_{excess}
Absolute Horizontal Initial Phase	0.9162598	| (S - S_{excess}) - P_{right} |
Left Scaler Initial Phase	0.9162598	(S - S_{excess}) >= P_{right}
Right Scaler Initial Phase	0.0	(S - S_{excess}) >= P_{right}

Pillar Box Windowing

If pillar-box borders are going to be applied by the Scaler, then Software will need to account for this across both Pipes when programming the PS_WIN_POS register.



As we can see from the above illustration, the left-side border is applied by the PS_WIN_POS programming of the left-side Scaler and the right-side border is applied by the H. Active programming of the right-side transcoder.

Left Side:

- PS_WIN_POS.X Position = Left side border size
- TRANS_HTOTAL.H Active = Destination image size w/o excess (D) + Left border size

Right Side:

- PS_WIN_POS.X Position = 0
- TRANS_HTOTAL.H Active = Destination image size w/o excess (D) + Right border size

Summary:

The following is a summary of the above discussion that Software will need to perform for tiled Pipe scaling.

Some general notes:

1. Adding excess pixels for tiled Pipe scaling is only necessary when upscaling the tiled images. If downscaling the tiled images, then excess pixels do not need to be added.
2. This document assumes that the pre-excess horizontal Pipe Source Size (i.e. S) is symmetric across both Pipes (i.e. S is the same for both the left and right-side Pipes). It is not a requirement that this term be symmetric, but it will be left to the reader to extrapolate the equations within this document for asymmetric S terms.

Source Image Manipulation

- **S is the horizontal Pipe Source Size without S_{excess} pixels (one-based)**
- **S_{excess} >= (NUMTAPS - 1)/2**
    - **NUMTAPS = 7**
    - **Round up the result to the nearest NUMPPC (NUMPPC=2)**
- **If the sum of the Plane Position (PP) and the Full H. Plane Size (HPS) is greater than S (i.e. (PP + HPS) > S):**
    - **HPS = Left Plane Size (LPS) + Right Plane Size (RPS)**
    - **LPS = S - PP**
    - **RPS = HPS - (S - PP)**

Note that the "Full H. Plane Size" is the size of the Plane image after rotation.

The following table summarizes how the Plane offset, position, and size programming changes with rotation. Some notes:

1. The programming reflected in this table is only applicable for Plane images that straddle the seam (i.e. (PO + HPS) > S)
2. The Plane offset programming in the table is not accounting for any image cropping that software may be applying to the image in memory
3. Software should ensure that the horizontal Plane position for the image in the right Pipe is zero (i.e. there should be no positional offset of the image with respect to the seam)
4. As the diagrams above illustrate, depending on rotation the LPS and RPS terms are either based off the horizontal or vertical size of the image in memory. The terms will be denoted with "h" or "v" subscripts to differentiate which size the term is based off.

Rotation	Side	Register	Field	Programming
0	Left	PLANE_OFFSET	Starting X Pos	0
			Starting Y Pos	0
		PLANE_SIZE	Width	$LPS_h + S_{excess}$
			Height	V. Size
	Right	PLANE_OFFSET	Starting X Pos	$LPS_h - S_{excess}$
			Starting Y Pos	0
		PLANE_SIZE	Width	$RPS_h + S_{excess}$
			Height	V. Size
90	Left	PLANE_OFFSET	Starting X Pos	0

			Starting Y Pos	0
		PLANE_SIZE	Width	H. Size
			Height	$LPS_v + S_{excess}$
	Right	PLANE_OFFSET	Starting X Pos	0
			Starting Y Pos	$LPS_v - S_{excess}$
		PLANE_SIZE	Width	H. Size
			Height	$RPS_v + S_{excess}$
180	Left	PLANE_OFFSET	Starting X Pos	$RPS_h - S_{excess}$
			Starting Y Pos	0
		PLANE_SIZE	Width	$LPS_h + S_{excess}$
			Height	V. Size
	Right	PLANE_OFFSET	Starting X Pos	0
			Starting Y Pos	0
		PLANE_SIZE	Width	$RPS_h + S_{excess}$
			Height	V. Size
270	Left	PLANE_OFFSET	Starting X Pos	0
			Starting Y Pos	$RPS_v - S_{excess}$
		PLANE_SIZE	Width	H. Size
			Height	$LPS_v + S_{excess}$
	Right	PLANE_OFFSET	Starting X Pos	0
			Starting Y Pos	0
		PLANE_SIZE	Width	H. Size
			Height	$RPS_v + S_{excess}$

Cursor

Global Cursor Position	Left Pipe Cursor	Right Pipe Cursor
GCP < (S - S_{excess} - CS)	Enabled	Disabled
(S - S_{excess} - CS) <= GCP < (S + S_{excess})	Enabled	Enabled
GCP >= (S + S_{excess})	Disabled	Enabled

Global Cursor Position	Left Pipe Cursor Position	Right Pipe Cursor Position
GCP < (S - S_{excess} - CS)	Magnitude = GCP Sign = SW determines	N/A (Disabled)
(S - S_{excess} - CS) <= GCP < (S - S_{excess})	Magnitude = GCP Sign = Positive	Magnitude = S - S_{excess} - GCP Sign = Negative
(S - S_{excess}) <= GCP < (S + S_{excess})		Magnitude = GCP - S - S_{excess} Sign = Positive
GCP >= (S + S_{excess})	N/A (Disabled)	

Post-Excess

- **D is the number of non-excess destination pixels per Pipe (one-based)**
- **SF_{target} = S/D**
- **SF_{actual}**
    - **Scaler calculates Scale Factor: (S + [Left, Right] S_{excess}) / (D + D_{excess})**
    - **Programmed Scale Factor: SF_{target} (PS_PROG_HSCALE)**
    - Hardware maintains the Scale Factor at a precision of **2.14. When the Scaler calculates the Scale Factor, it will round the result.**
- **D_{excess} = ROUND(((S + Left S_{excess}) / SF_{target}) - D)**
    - **Round up to the nearest NUMPPC**

Initial Phase

- **P_{right} = SF_{actual} * (D - D_{excess})**
- **Absolute H. Initial Phase = | (S - S_{excess}) - P_{right} |**
    - **Absolute HIP applied to left-side Scaler if (S - S_{excess}) >= P_{right}**
    - **Absolute HIP applied to right-side Scaler if (S - S_{excess}) < P_{right}**

Impacts of Plane Scaling

If a Plane is being scaled across a seam, then the following variables will be impacted:

- The S_{excess} in the PLANE_SIZE will be unchanged
- The S_{excess} in the Pipe Source size will be equal to the D_{excess} from the Plane Scaler
- The D_{excess} at the outlet of the Pipe will be calculated based off the S_{excess} in the Pipe Source
- The Pipe Scaler's actual scale factor will be based off the S_{excess} in the Pipe Source and the D_{excess} at the outlet of the Pipe

Affected Registers

The following table summarizes the registers directly affected when multi-Pipe Scaling is being employed

Register	Field	Programming
PLANE_POS	X/Y Position	See above "Source_Image_Manipulation" section
PLANE_OFFSET	Start X/Y Position	See table above in "Source_Image_Manipulation"
PLANE_SIZE	Width/Height	See table above in "Source_Image_Manipulation"
PIPE_SRCSIZE	H. Source Size	S + S_{excess}
PIPE_SEAM_EXCESS		**Left Side:** Right Excess Amount = D_{excess} **Right Side:** Left Excess Amount = D_{excess}
PS_WIN_SZ	X Size	D + D_{excess}
PS_HPHASE	RGB Initial H. Phase (Integer & Fraction)	Programming dependent on comparison of (S - S_{excess}) and P_{right} Note that the Initial H. Phase Trip bit should be set
PS_WIN_POS	X Position	**Left Side:** Left side border size, **if a border is present** **Right Side:** This should always be zero
TRANS_HTOTAL	H. Active	**Left Side:** D + Left side border size (if border present) **Right Side:** D + Right side border size (if border present)

Planes

Flips

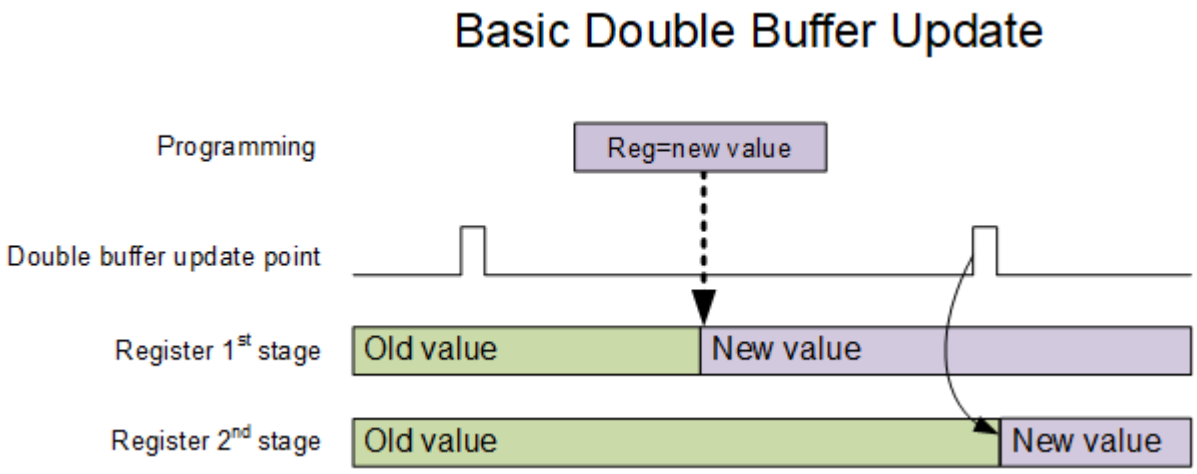
Overview

Display flips originally referred to switching between frame buffer surfaces to change the image. The term has evolved to refer more generally to an update to a plane configuration. Cursor configuration changes can also be considered to have flips, although it's more rare to describe a cursor update that way. Even updates to an enabled display pipe, such as new color correction settings, can be considered a flip, but usually only when combined with changes to the planes. A flip can be as small as updating one plane's surface address to a new frame buffer location, or as large as a synchronized update to planes, cursors, color correction, scaling, and transcoder metadata, across multiple running pipes.

Double Buffering

Most configuration that is allowed to be changed while a function is enabled, such as flips, is done through double buffered registers. Double buffer registers have two stages of registers to align the configuration update to a safe point, typically the start of vertical blank.

Writes and reads to double buffer registers access the first stage. The first stage is transferred to the second stage at the double buffer update point. The second stage is the "live value" that goes to the function being configured.



Some functions, such as planes, have read only registers to read the "live value" from the second stage.

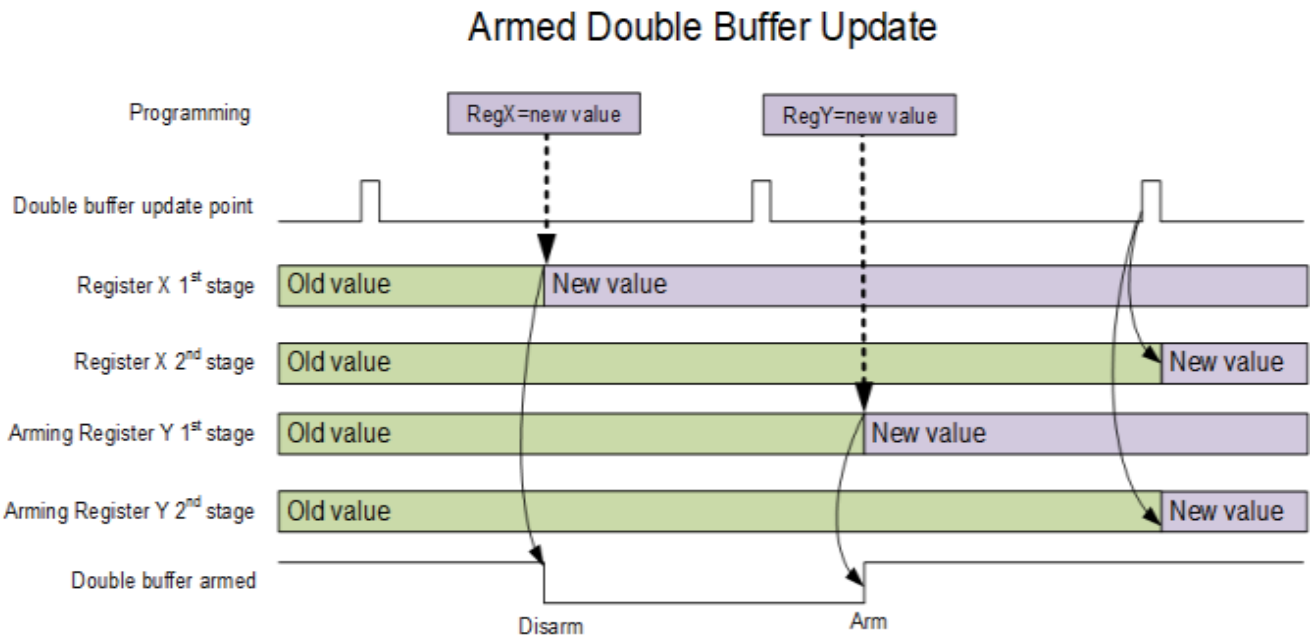
Arming and double buffer stalling are used for some registers to control the double buffer update point for atomic updates across multiple registers and functions.

Atomic Updates

Updates to multiple registers of a single function can be done atomically using the arming mechanism that some functions, such as planes, use. One of the registers for the function is defined as the arming register, described in the double buffer fields of the registers. Writes to the arming register 1st stage will

arm the double buffer to update at the next double buffer update point. Writes to the other double buffered registers for that function will disarm and prevent updates at the double buffer point. The programming sequence is to write the disarming registers first and the arming register last, then all the registers update at the next double buffer update point.

Updates across multiple planes and other resources require double buffer stalling as explained in the Double Buffer Control section.



Note: Once armed, by writing to the armed by register, the registers controlled by this arming should not be changed until the double buffer update point is reached. If changed, this will disarm the sequence and will require another write to the armed by register to get it to the armed status again.

Flip Types

There are two broad types of flip; sync and async.

Synchronous Flips

Synchronous flips (sync flips) update the plane surface base address, and other double-buffered registers, at the frame boundary (start of next vertical blank). These can be used for updates across multiple functions, not just planes.

Asynchronous Flips

Asynchronous flips (async flips) update the plane surface base address as soon as possible (next TLB boundary reached or start of next vertical blank). This can cause the image to change mid-frame with a tear, but gives better responsiveness than sync flips.

Cursors cannot do async flips.

Asynchronous flip can update only the plane surface (PLANE_SURF).

Asynchronous flip can update only the plane surface (PLANE_SURF).

Asynchronous flip completion time depends greatly on how much data has been prefetched for power savings, and can take up to 1 full frame to complete. For faster flip completion, disable FBC and render/unified compression and allocate a minimum amount of data buffer for the plane.

Asynchronous flip completion is delayed during the vblank pipeline fill region. With adaptive sync, the pipeline fill region is constrained by the VRR programming. Without adaptive sync, the pipeline fill begins near the start of vblank at the frame start, and non-adaptive sync panels with very large vblanks, such as those with fixed pixel rate that lower refresh by extending vblank, can have significant flip delays. The delay can be reduced by delaying the start of vblank internal to the display pipe by programming vblank start greater than vertical active. Refer to the High Refresh Rate and Small Vblank Support page for details on features that are impacted when the vblank is reduced.

TRANS_SET_CONTEXT_LATENCY is used to delay the start of vblank internal to the display pipe

Flip Programming

There are two paths to programming a flip; MMIO and command ring.

MMIO Flips

MMIO writes may be used to flip.

General MMIO Sequence for Plane or Cursor Flips

1. If interrupt will be used to wait for the flip to complete, unmask and enable the flip done interrupt for the plane that will be flipped. See the Display Interrupts section. For cursors, use the vblank interrupt.
2. Write registers to update plane configuration fields besides the arming register, including PLANE_CTL Async Address Update Enable to select sync or async flip.
3. Arm the update: For planes, write PLANE_SURF with the plane surface address. For cursors, write CUR_BASE. For other resources, write the arming register as specified in registers for that resource. Some resources do not have arming.
4. Optional: Wait for the interrupt for the flipped plane. Not required if software uses other methods to find when the old surface is no longer being used.

Double buffering control does not apply to PLANE_SURF updates that occur when the plane is disabled so an interrupt event is generated immediately when the PLANE_SURF is written.

Command Ring Flips

The render command MI_DISPLAY_FLIP may be used to flip. Typically, it is only used for plane surface address updates, but it can be extended to more complicated flips.

MI_DISPLAY_FLIP can be combined with LOAD_REGISTER_IMMEDIATE, which behaves like a MMIO write, to update other parts of a plane configuration.

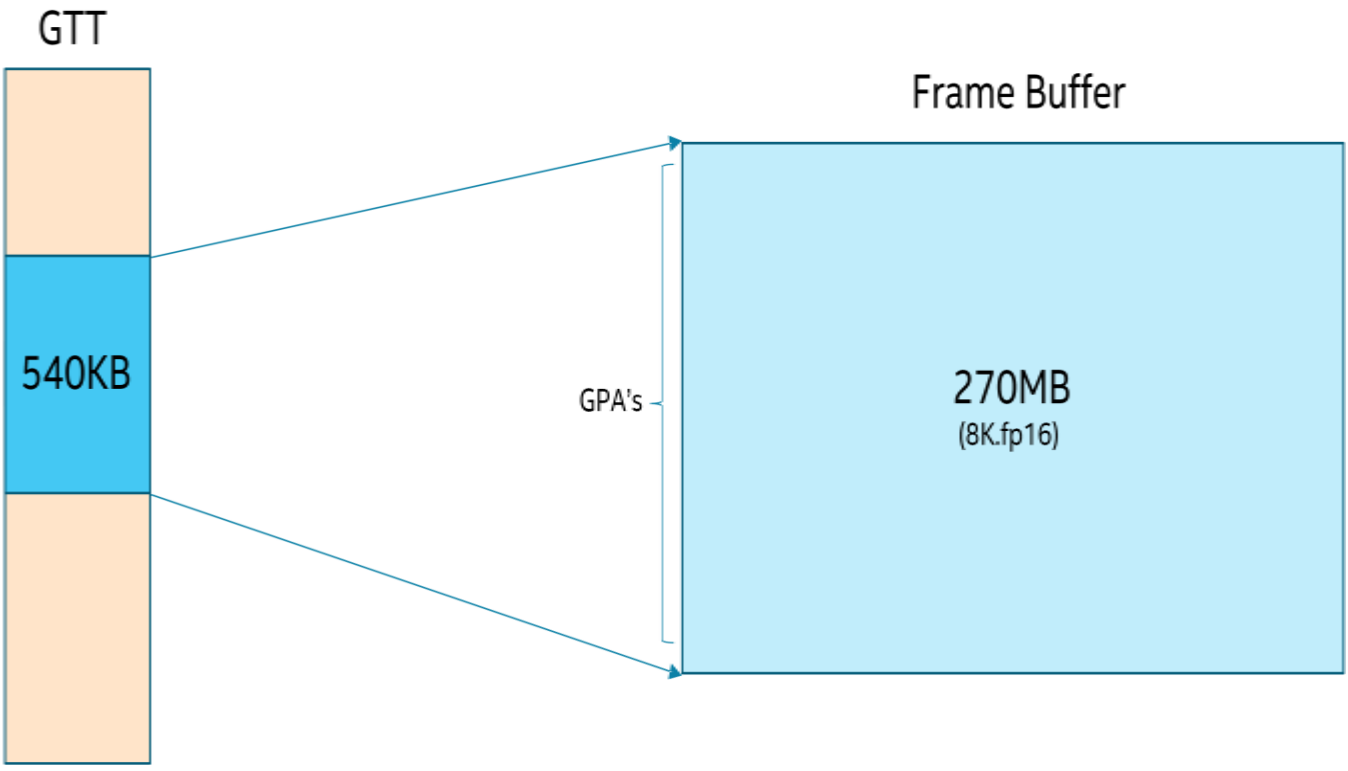
General Command Sequence for Plane Flips

1. If MI_WAIT_FOR_EVENT will be used to wait for the flip to complete, unmask flip done for the plane that will be flipped in DE_RRMR using MMIO or LOAD_REGISTER_IMMEDIATE. See the Render Response section.
2. Optional: LOAD_REGISTER_IMMEDIATE to write registers to update plane configuration fields that are not included in the MI_DISPLAY_FLIP_COMMAND.
3. MI_DISPLAY_FLIP to update plane surface address and other configuration listed in the command details (stride, tile format, and more), and select sync or async flip. The flip is self arming.
4. Optional: MI_WAIT_FOR_EVENT to wait for the flip to complete for the flipped plane. Not required if software uses other methods to find when the old surface is no longer being used.
5. Re-mask the flip done in DE_RRMR using MMIO or LOAD_REGISTER_IMMEDIATE. It does not need to be immediately re-masked, but unmasked events will wake GT as they occur, so for improved power savings it is recommended to only unmask events as they are required.

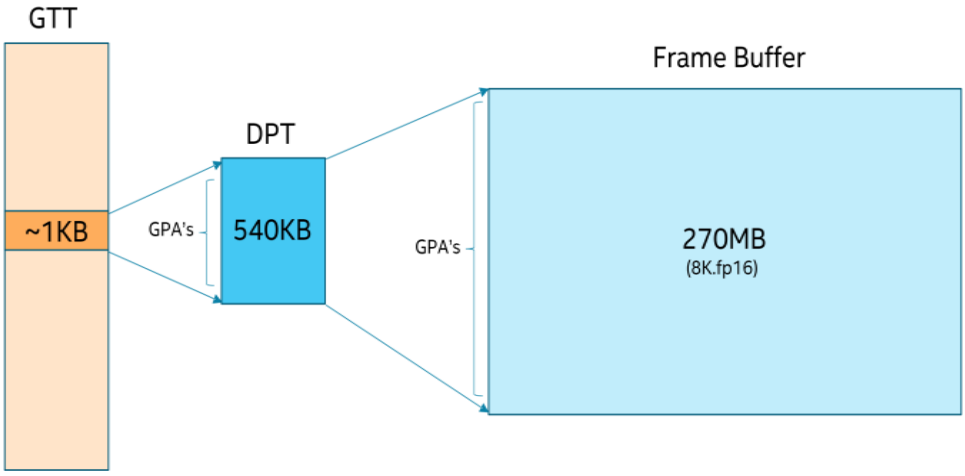
Flips to a plane that is disabled or on a disabled pipe will complete immediately and send the flip done.

Display Page Tables

Linear frame buffers are GGTT-mapped for Display access



But tiled frame buffers must use *Display Page Tables*, and those DPT's themselves are GGTT-mapped.



Note this is not a *"2-level page table"*, so much as a *"1-level page table, that itself is in the VA space of another"*—i.e. GGTT VA space remains 4GB, and a DPT-mapped frame buffer does *not* live in that 4GB. A DPT-mapped frame buffer lives at address-zero of its own DPT VA space. SW programs flips to DPT-mapped frame buffers by specifying the *DPT's* GGTT VA (since such frame buffers don't have GGTT VA's themselves).

DPT's use the same PTE format as the GGTT—with each PTE mapping 4KB of frame buffer memory. A DPT need only be large enough to map its frame buffer (including required padding), and a DPT is free to use physically discontiguous pages (same as a frame buffer is free to). The padded entries do not need to be valid.

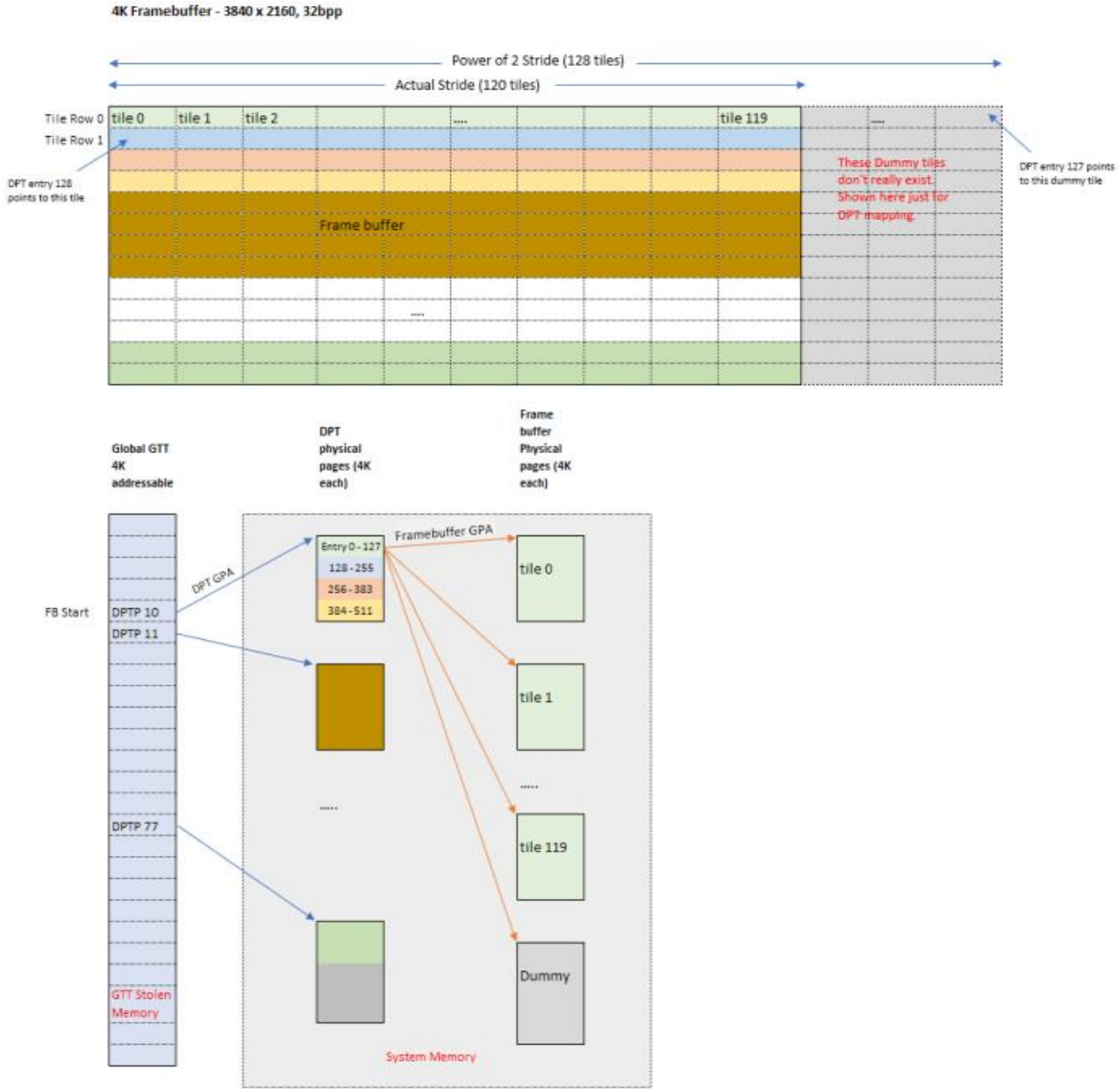
> **[!]** Since DPT's contain physical addresses, they must never be mapped into an untrusted CPU nor GPU VA space (i.e., don't be tempted to include DPT pages in the Frame Buffer's own physical allocation).

Display scan-out is the only HW path that supports DPT—If GTT access to a DPT-mapped frame buffer is needed for any other purpose, the frame buffer must also have a direct GTT mapping.

Programming notes

- Frame buffer surfaces always start on a new 4K display page table (DPT) at offset 0.
- Planar YUV format - Y and UV surface will have its own individual DPT and start at offset 0.
- Compression control surfaces (integrated graphics only) will have its own DPT and start at offset 0.

Display Page Table Mapping Example

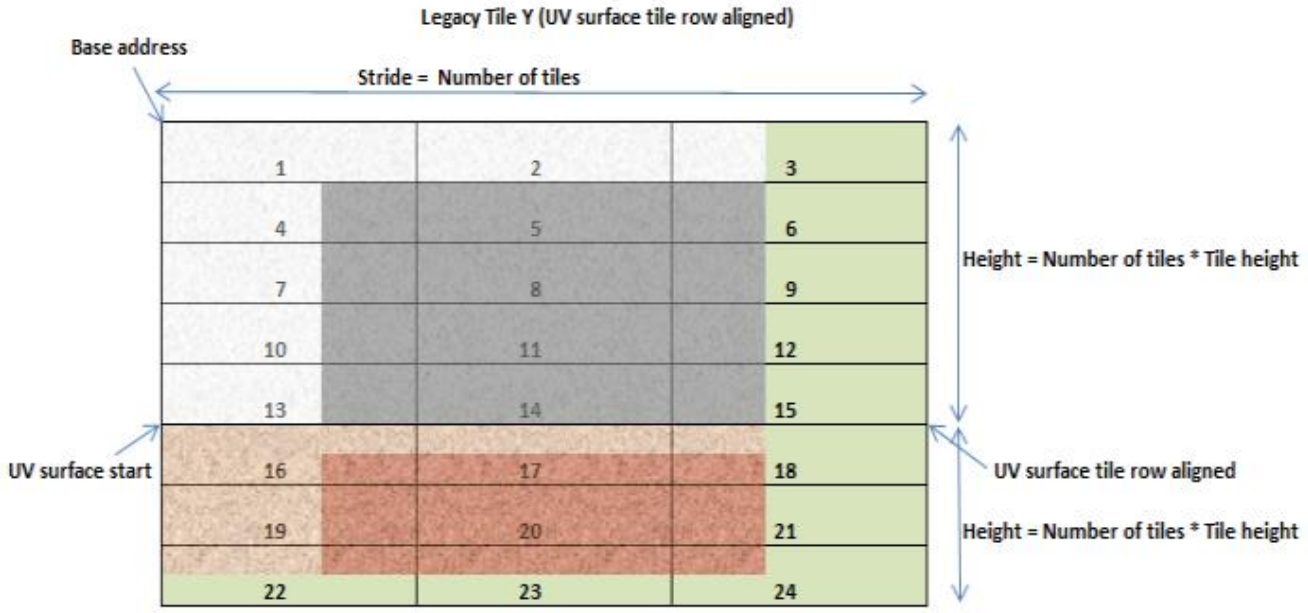


Plane Planar YUV programming 0

YUV 420 hybrid planar formats (NV12, P0xx) have Y and UV surfaces stored separately. Display supports YUV 420 planar frame buffers in linear and tiled surface formats. Each display pipe can support up to two hybrid planar YUV 420 frame buffers. Display hardware uses two planes to handle each of these framebuffers - one for Y surface data and the other for UV surface. In addition to the two planes, a plane scaler must be enabled for YUV420 to YUV444 upsampling. See the Plane Capability and Interoperability section for UV and Y surface plane assignments.

Y and UV surface Base Address, Stride and Surface Offset (Start X and Y position) must be programmed separately for Y and UV planes. Software must make sure that the Y and UV plane programming gets applied together in the same frame. The Y and UV plane programming can be synchronized using the double buffer synchronization mechanism defined in DOUBLE_BUFFER_CTL. In the case of DisplayPort VRR mode (Variable Refresh Rate), the VRR push must be used for the Y and UV plane synchronization. Software must make sure that there is no push between the Y and UV plane flips.

With tiled surfaces, the UV surface must start on a new tile row.



Y Surface

Base Address, Stride and Surface Offset (Start X and Y position) should follow the standard plane programming.

UV Surface

The UV surface has some additional requirements.

Base Address:

For UV surfaces, the programmed Base Address should satisfy the following alignment requirements:

Linear:

The UV surface Base Address programmed in the PLANE_SURF register should be aligned to Stride in bytes of the Y surface * 64.

Tiled surfaces:

The start of the UV surface programmed in the PLANE_SURF register should be Tile Row Aligned.

Surface Offset:

The UV surface offset is programmed in the PLANE_OFFSET register. In the 0/180 rotate cases, the programmed X and Y start position should be half of the Y surface start X and Y positions.

For 0/180 rotate cases:

UV surface Start X Position = half of Y plane X start position

UV surface Start Y Position = half of Y plane Y start position

For 90/270 rotate cases:

UV surface Start X Position = (UV surface height in tiles * tile height) - UV plane Y start position (non-rotated) - UV plane height (non-rotated)

UV surface Start Y Position = UV plane Start X Position (non-rotated)

90/270 Rotation

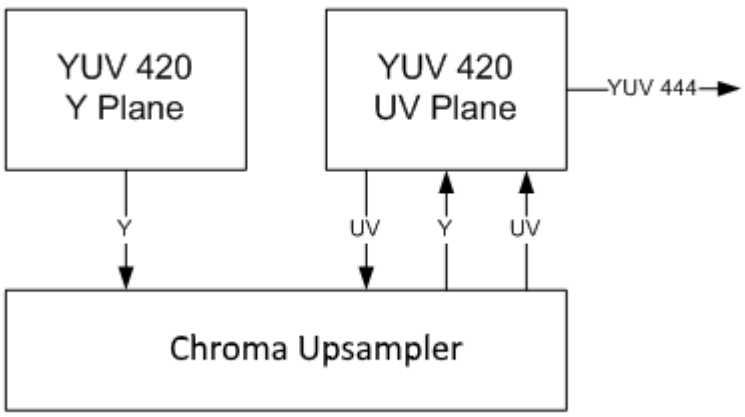
Plane 90/270 rotation support requires GTT remapping. GTT remapping allows the hardware to walk the pages sequentially. The Y and UV surfaces should be remapped separately, and the pages that contain portions of both Y and UV data will get GTT remapped twice, once for Y surface and the other for UV surface.

Upsampling and further processing

Planes 1-3 have dedicated chroma upsampler that is programmed in PLANE_CUS_CTL register.

The Y and UV plane pixels go through the chroma upsampler, gets upsampled, scaled and merged to form the YUV444 pixels for further processing.

The Y plane binding is specified in the 'PLANE_CUS_CTL->Y Binding' field.



Destination keying, Color correction and Gamma must be programmed in the UV plane. Destination keying, Color correction and Gamma programming in the Y plane gets ignored. The UV plane position is used for blending order determination.

Flips

Since two different planes are involved in handling the pixels from a single planar YUV 420 frame buffer, we have to ensure that both the Y and UV planes' flip gets applied at the same time.

Double buffer stall disable mechanism can be used to synchronize and commit the flips of both planes atomically. See the section on Double Buffer Control and use the sequence for synchronizing double buffer updates to synchronize the updates to both Y and UV planes. For command streamer flips, the LOAD_REGISTER_IMMEDIATE (LRI) command can be used to program the double buffer stalling registers.

Watermarks

Watermarks should be calculated and programmed for Y and UV planes separately. Refer to Watermarks page for further details.

Plane Capability and Interoperability

Plane Assignments and Capabilities

Plane capabilities:

- There are 5 planes + 1 cursor/pipe.
- Each planar YUV 420 surface uses 2 planes. Y and UV surfaces are handled by separate planes which gets combined later in the pipeline. Each pipe can support up to 2 YUV 420 planar surfaces.
- HDR mode supports up to 3 planes per pipe.

Plane	Pipe A	Pipe B	Pipe C and Pipe D
Cursor	Cursor	Same as pipe A	Same as pipe A
Plane 5	Decompression Planar YUV 420 Y Surface OLED Compensation	Same as pipe A	Same as pipe A
Plane 4	Decompression Planar YUV 420 Y Surface	Same as pipe A	Same as pipe A
Plane 3	Decompression Planar YUV 420 UV Surface HDR	Same as pipe A	Same as pipe A

Plane	Pipe A	Pipe B	Pipe C and Pipe D
Plane 2	Decompression Planar YUV 420 UV Surface HDR	Same as pipe A	Same as pipe A
Plane 1	Decompression Planar YUV 420 UV Surface HDR Frame Buffer Compression (FBC)	Same as pipe A, minus FBC	Same as pipe A, minus FBC

Display Plane Mapping to Command Streamer Plane Number

Used by MI_DISPLAY_FLIP and MI_WAIT_FOR_EVENT

Display Plane Name	Command Streamer Plane Number
Plane 1 A	Plane 1
Plane 1 B	Plane 2
Plane 1 C	Plane 3
Plane 2 A	Plane 4
Plane 2 B	Plane 5
Plane 2 C	Plane 6
Plane 3 A	Plane 7
Plane 3 B	Plane 8
Plane 3 C	Plane 9
Plane 4 A	Plane 10
Plane 4 B	Plane 11
Plane 4 C	Plane 12
Plane 5 A	Plane 13
Plane 5 B	Plane 14
Plane 5 C	Plane 15
N/A	Plane 16
N/A	Plane 17
N/A	Plane 18
N/A	Plane 19
N/A	Plane 20
N/A	Plane 21
Plane 1 D	Plane 22
Plane 2 D	Plane 23

Display Plane Name	Command Streamer Plane Number
Plane 3 D	Plane 24
Plane 4 D	Plane 25
Plane 5 D	Plane 26
N/A	Plane 27
N/A	Plane 28
N/A	Plane 29
N/A	Plane 30
N/A	Plane 31
N/A	Plane 32

Plane Feature Interoperability

Display Features / Surface Formats:

All surface formats support 0 and 180 degree rotation with all tile formats.

Frame Buffer Compression (FBC) supported only with RGB32 8:8:8:8 without per-pixel alpha and RGB 16-bit 5:6:5.

Horizontal flip supported with all surface formats.

Per-pixel alpha supported with RGB formats that have alpha channel.

Color keying supported in non-HDR mode on surface formats other than FP16.

Async flip supported with RGB pixel formats and not YUV.

	RGB32 2:10:10:10	RGB32 8:8:8:8	RGB 32-bit XR_BIAS 10:10:10	RGB64 16:16:16:16 FP16	RGB64 16:16:16:16 UINT	RGB 16-bit 5:6:5	Indexed 8-bit	All YUV formats
Render Compression	X	X		X				
Media Compression	X	X		X				X
Unified Compression	X	X		X				X
Plane Scaling	X	X		X	X	X		X

90/270 rotation not supported.

Tiling Modes / Rotation Modes / Display Features:

Render/media/unified decompression supported only with Y tile or tile 4.

Rotation 0 and 180 degrees supported with all tile modes. Rotation 0 and 180 degrees supported with interlacing, frame buffer compression, and render/media/unified compression.

Horizontal flip (mirror the image from right to left) supported with tile modes other than linear. Horizontal flip supported with interlacing, frame buffer compression, and render/media/unified compression.

Frame buffer compression supported with all tile modes.

Interlacing with interlaced fetch supported only with linear and X tile.

90/270 rotation not supported.

Render Decompression

GT Render engines uses a lossless scheme to compress the color Render targets. The goal of the compression is to reduce the memory bandwidth. The memory footprint increases slightly due to the need of the control surface.

- Decompression is supported with RGB8888, RGB1010102 and FP16 formats.
- Decompression is supported only with legacy tile Y or tile 4 surfaces.
- Decompression support is limited to left-right cache line pair mode. Top-bottom mode is not supported.
- Decompression is not supported with 90/270 degree rotation.
- Compressed displayable surfaces must be 16KB aligned and have pitches padded to multiple of 4 tiles.

Color Control Surface

The Color Control Surface (CCS) contains the compression status of the cache-line pairs**.** CCS stride is programmed separately independent of the main surface stride. When the main render surface is encrypted, the corresponding control surface must be encrypted. The compression state of the cache-line pair is specified by 4 bits in the CCS. The address of CCS surface is specified as an offset from the start of the Render Target main surface.

Decompression Programming

When compressed Render targets are presented to Display, the display decompression must be enabled. Along with main surface programming, the following additional programming is required to enable the decompression

- **Decompression Enable**

  Decompression is enabled by programming the 'Render Decomp' bit in the PLANE_CTL register.

- **Color Control Surface Distance**

  The start of the CCS surface is programmed as the distance from the start of the main surface in the PLANE_AUX_DIST register. The CCS is always placed after the main surface and is 4K page aligned.

- **Clear color value**

  When clear color is enabled in PLANE_CTL register, the unencrypted clear color value must be programmed in the PLANE_CC_VAL register.

Media Decompression

The goal of the compression is to reduce the memory bandwidth. The memory footprint increases slightly due to the use of the control surface.

Display supports decompression of compressed media surfaces. 4 horizontally adjacent cache lines form a compression unit and its compression status is stored in the control surface (Tile Status Surface). The address of the control surface is specified as an offset from the start of the main surface. The control surface is always linear. When the main media surface is encrypted, the corresponding control surface(s) must also be encrypted.

With Y tiling and planar YUV surfaces UV surface, the UV surface starts right after the Y surface, and it can be in the middle on the tile row. The only guarantee is that the Y surface start will be 4 lines (cache line) aligned. Y and UV surfaces have their own individual control surface and they both have to be programmed independently. In cases where the UV plane start is not aligned to the tile row start, the UV main surface start is programmed to the previous tile row aligned address with a plane offset to the start of the UV. In this case, the UV control surface will match the main surface start and hence will have the tile status for all compression units from the previously tile row aligned main surface address.

Restrictions
• 90/270 rotation not supported. • Async flips not supported.
• Only Y tiling is supported.

Decompression Programming

When compressed media targets are presented to display, the media decompression must be enabled. Along with main surface programming, the following additional programming is required for both Y and UV planes individually.

- **Decompression Enable**

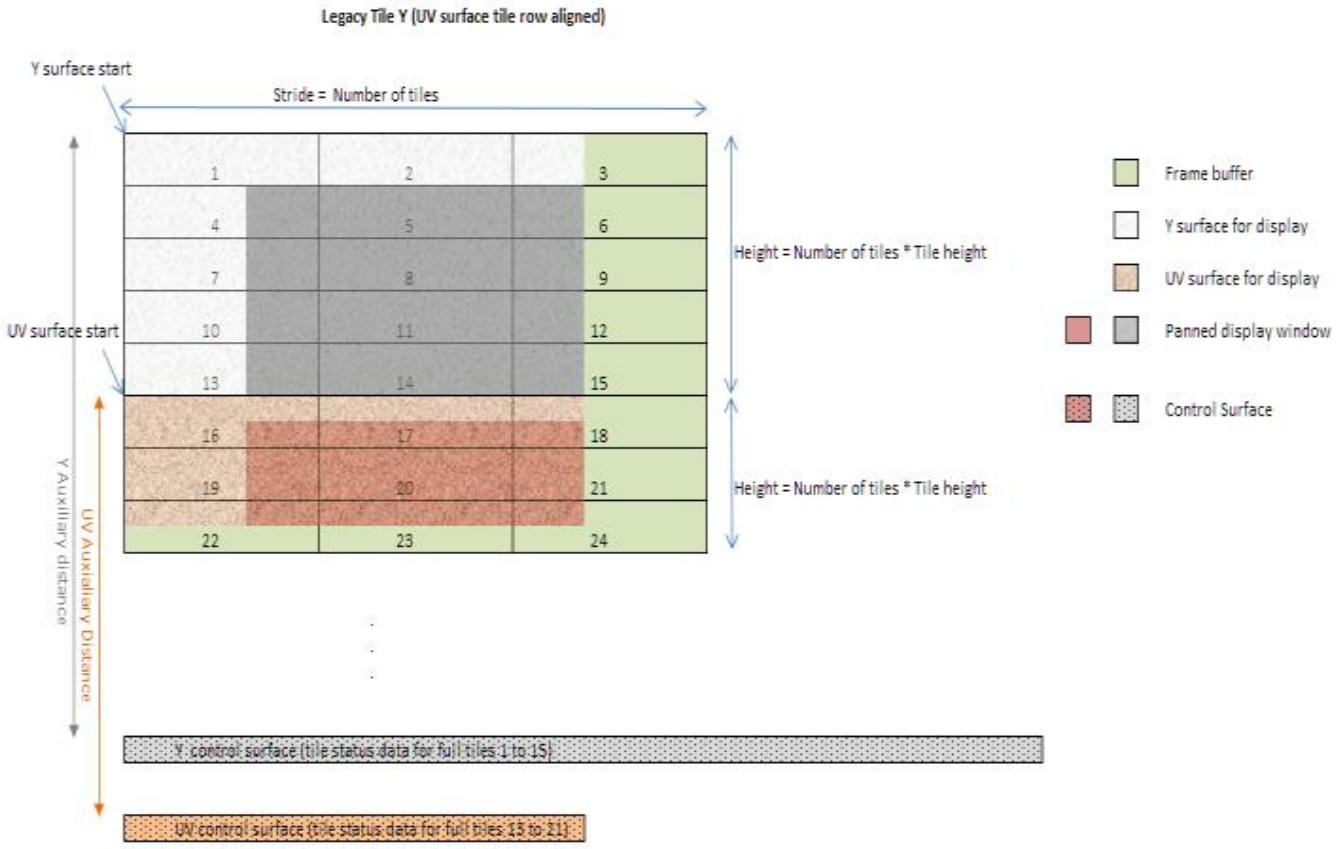
  Decompression is enabled by programming the 'Media Decomp' bit in the PLANE_CTL register.

- **Control Surface Distance**

The start of the control surface is programmed as the distance in bytes from the start of the main surface in the PLANE_AUX_DIST register. The control surface must always be placed after (not necessarily immediately) the main surface and must be 4K page aligned.

Control surface stride is not used for media compression.

Y and UV control surfaces must both be 4k aligned.



Plane Rotation Programming

This topic provides programming information for plane rotation.

The 180-rotation mode is unchanged and will continue to use the same programming. In the 180-rotation mode display hardware is responsible for walking the pages in the reverse order. The cacheline walk within the page is also reversed. The 90 and 270 rotation modes require more complicated page walk mechanism. The page walk is made transparent to the hardware by providing a different set of page translations (remapped GTT) for the same rendered surface. The remapping completely abstracts the page walk away from the hardware and the hardware walks the pages as if there is no rotation. Hardware is still responsible for handling the walk within the page appropriately. Also, the 90, 270 Rotation requires a new parameter - surface height. The changes needed in the driver programming is discussed below.

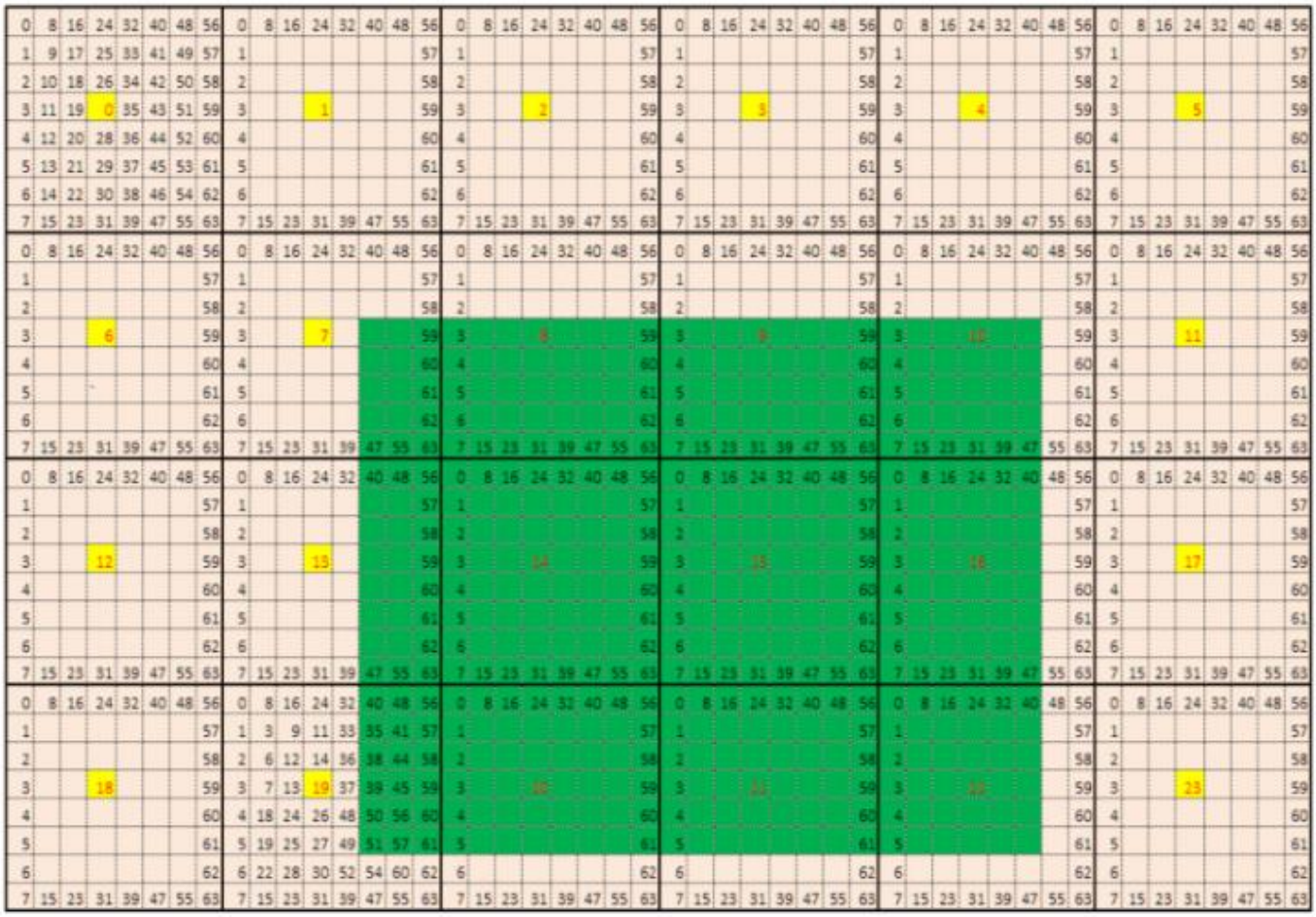
90 Rotation

For the 90-rotation programming, the plane parameters must use the following mapping

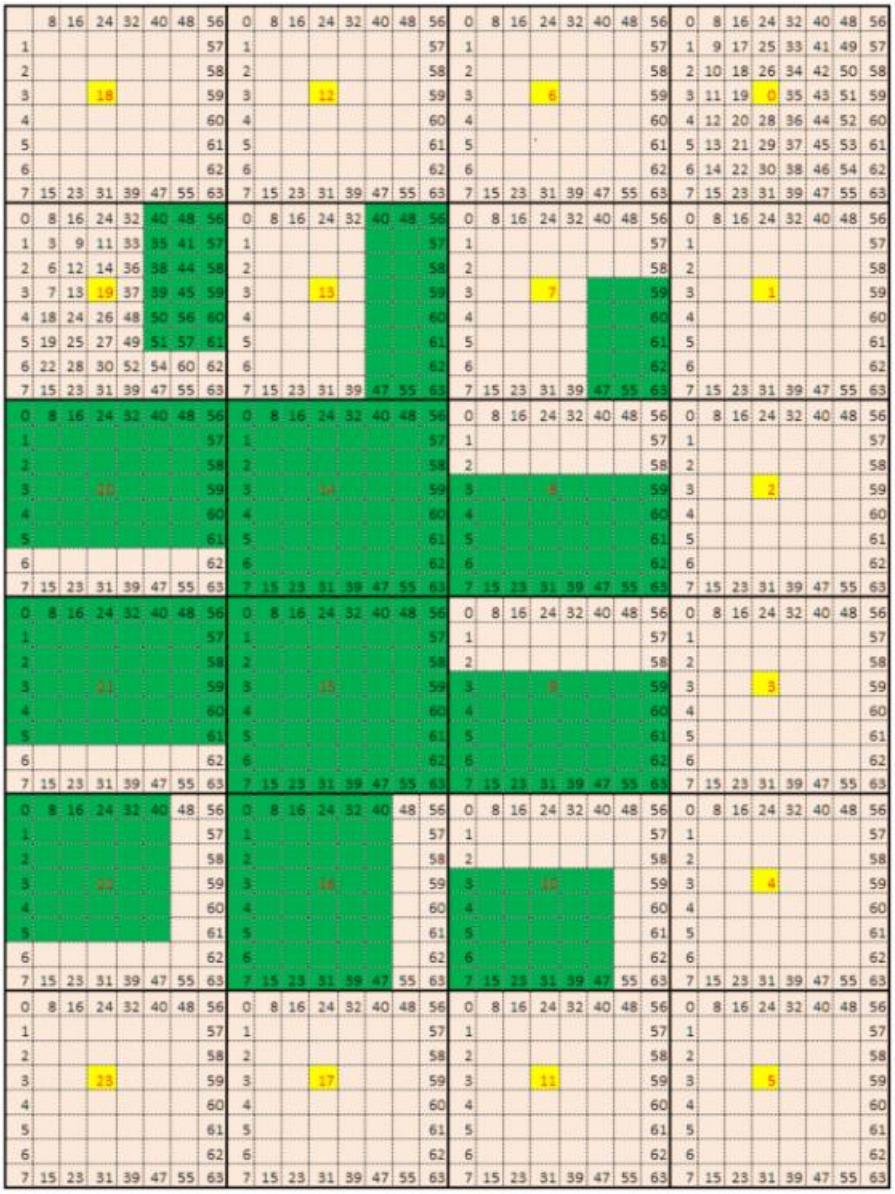
- Base address = New address (remapped GTT)
- Stride = Surface height in tiles
- Plane Width = Plane Height
- Plane Height = Plane Width

- X offset = (Surface height in tiles * tile height) - Plane Y offset - Plane Height *[Note: the calculated X offset will always be >= 0 since Plane Y Offset + Plane Height <= Surface Height in lines]*
- Y offset = X offset
- Driver can enable the Scaler, as needed, to fit the rotated content in to the pipe active display area.

Sample frame buffer surface



Page access sequence in the rotated 90 mode



Example

Let us assume the following display programming for a single pipe - single plane, Y tiled, non-rotated, 1920 x 1200, 4Bpp with the plane panned (100, 150), covering full active area and scaler not enabled.

GTT mapping

Here is a sample GTT mapping for 90 rotation mode.

Original GTT

Assumed Surface base = 0x200000

0x200000 = page 0

0x201000 = page 1

0x202000 = page 2

0x203000 = page 3

...

Remapped Display GTT - 90 rotation

Assumed new Surface base = 0x400000

0x400000 = page 18

0x401000 = page 12

0x402000 = page 6

0x403000 = page 0

Register programming for non-rotated scenario

- *PLANE_SURF->Surface Base Address = 0x200000*
- *PLANE_STRIDE->Stride = 60* [(1920 * 4)/128] [(width *bpp)/tile width]
- *PLANE_SIZE->Width = 1920*
- *PLANE_SIZE->Height = 1200*
- *PLANE_OFFSET->Start X Position = 100*
- *PLANE_OFFSET->Start Y Position = 150*
- *Surface Height in tiles (assumed) = 50 (allocated surface height in number of scan lines/tile height. For plane height = 1200, the surface height should be a minimum of 38 tiles (ceiling (1200/32). When panning is used, the rendered frame buffer surface will be larger than the plane size. Here, let us assume that the rendered surface height in tiles = 50).*

The programming changes to following for a 90 rotation scenario

- *PLANE_SURF->Surface Base Address = 0x400000* [uses remapped GTT]
- *PLANE_STRIDE->Stride = 50* [Surface height in tiles (assumed earlier)]
- *PLANE_SIZE->Width = 1200* [non-rotated Height]
- *PLANE_SIZE->Height = 1920* [non-rotated Width]
- *PLANE_OFFSET->Start X Position = 250* [(50*32)-150-1200] [(Surface height * tile height) - non rotated Y position - non rotated Height]
- *PLANE_OFFSET->Start Y Position = 100* [non-rotated X position]

The scaler should be programmed appropriately to fit the rotated plane in the pipe active area and the window position should be adjusted if it is desired to maintain the same apparent position on a physically rotated display.

270 rotation

Uses the same GTT remapping and register programming as 90 rotation mode with the Plane control register rotation mode set as 270.

NV12 (YUV 420) rotation

For NV12 90/270 rotation, the Y and UV surfaces should be treated as separate surfaces and thus the GTT remapping for rotation should be done separately.

Display Buffer Programming

Display Buffer Allocation

Allocation of the display buffer is programmable for each display plane, using the buffer start and buffer end values in **PLANE_BUF_CFG**.

Proper display buffer allocation is important for Display hardware to function correctly. Optimal allocation provides better display residency in memory low power modes. Display Buffer allocation must be recalculated and programmed when pipes/planes get enabled or disabled.

Display Buffer Size

Buffer Count	Total Buffer Size	Total Buffer Blocks	Blocks Available for Programming
4	2 MB (512 KB per buffer)	4096 (1024 per buffer)	Pipe A and B = 2048 Pipe C and D = 2048 Total 4096

Each display buffer block is 8 cache lines.

Allocation Requirements

Allocation must not overlap between any enabled planes.

A minimum allocation is required for any enabled plane. See Minimum Allocation Requirements below.

A gap between allocation for enabled planes is allowed.

The allocation for enabled planes should be as large as possible to allow for higher watermarks and better residency in memory power saving modes.

Minimum Allocation Requirements

Allocation for each enabled plane must meet the minimum requirement for watermark 0 from the Watermark Calculations section: Level 0 Minimum Display Buffer allocation Needed

Multi-Buffer Enabling and Allocation Requirements

There are 2 Mbus, each with 2 display pipes and 2 display buffers (DBUFs). The display buffers on a particular Mbus can only be used by the pipes on that Mbus.

Display Buffer	Buffer Start	Buffer End	Buffer Pair	Pipe Pair	Mbus
DBUF_S0	0	1023	S0+S1	Pipe A + Pipe B	1
DBUF_S1	1024	2047	S0+S1	Pipe A + Pipe B	1
DBUF_S2	0	1023	S2+S3	Pipe C + Pipe D	2
DBUF_S3	1024	2047	S2+S3	Pipe C + Pipe D	2

Enable DBUF_S0 with the display initialization sequence so it will be always available for VGA and backwards compatibility. Enable other DBUFs when any planes are allocated to them as per the following table.

The table ensures that pipes are using the closest DBUF when there are multiple pipes enabled. Each Mbus operates independently.

MBus1 Pipe and DBUF ordering: PipeA - DBUF_S0 - DBUF_S1 - PipeB

MBus2 Pipe and DBUF ordering: PipeC - DBUF_S2 - DBUF_S3 - PipeD

When a pipe is allowed to allocate from 2 DBUFs, a plane on that pipe may use allocation that straddles the 2 DBUFs.

Pipe A Planes DBUF Allocation	Pipe B Planes DBUF Allocation	Pipe C Planes DBUF Allocation	Pipe D Planes DBUF Allocation	Pipes with Enabled Planes
N/A	N/A	N/A	N/A	None or VGA
S0+S1	N/A	N/A	N/A	A
N/A	S0+S1	N/A	N/A	B
S0	S1	N/A	N/A	A+B
N/A	N/A	S2+S3	N/A	C
S0+S1	N/A	S2+S3	N/A	A+C
N/A	S0+S1	S2+S3	N/A	B+C
S0	S1	S2+S3	N/A	A+B+C
N/A	N/A	N/A	S2+S3	D
S0+S1	N/A	N/A	S2+S3	A+D
N/A	S0+S1	N/A	S2+S3	B+D
S0	S1	N/A	S2+S3	A+B+D
N/A	N/A	S2	S3	C+D
S0+S1	N/A	S2	S3	A+C+D
N/A	S0+S1	S2	S3	B+C+D

Pipe A Planes DBUF Allocation	Pipe B Planes DBUF Allocation	Pipe C Planes DBUF Allocation	Pipe D Planes DBUF Allocation	Pipes with Enabled Planes
S0	S1	S2	S3	A+B+C+D

Basic Allocation Method

These are basic methods that can be used for basic functionality. For optimal power usage, the display driver can choose to use more advanced allocation techniques as desired.

Example Method 1

Enable display buffer(s). Refer to DBUF_CTL register for display buffer enabling.

*TotalBlocksAvailable = Number of DBUFs allocated to this pipe * 1024 blocks;* Use the display buffer allocation table above to find the number of DBUFs allocated to this pipe.

Allocate a fixed number of blocks to cursor and then allocate the remaining blocks among planes, based on each plane's data rate.

BlocksAvailable = TotalBlocksAvailable

1. Allocate a fixed number of blocks to cursor

   The driver frequently enables and disables the cursor or changes the cursor pixel format. Fixed allocation is preferred for cursor to minimize the buffer re-allocation. The optimal amount to allocate depends on how much is needed to support deeper low power states (based on the results of watermark calculations) and could be scaled with screen resolution.

   BlocksAvailable = BlocksAvailable - CursorBufAlloc

2. Check for minimum buffer requirement

   For each enabled plane

   PlaneMinAlloc = Watermark Calculations section: Level 0 Minimum Display Buffer allocation Needed

   If sum of PlaneMinAlloc > BlocksAvailable

   Error - Display Mode can't be supported.

   The driver can change the number of enabled planes or the plane configuration and rerun the algorithm.

3. Calculate Relative Data Rate for planes

   In this step the driver may want to use the expected maximum plane source sizes so it does not have to reallocate for a plane that is changing size.

   For each enabled plane

   If PlaneScalerEnabled

*PlaneScaleFactor = (Plane width/Scaler window X size) * (Plane height/Scaler window Y size)*

Else

PlaneScaleFactor = 1

*PlaneRelativeDataRate = Plane height * Plane width * plane source bytes per pixel * PlaneScaleFactor*

4. Allocate blocks for enabled planes as per the Data rate ratio.

   For each plane that needs allocation (PlaneBlockAllocFinal == false)

   *PlaneBufAlloc = floor (BlocksAvailable * PlaneRelativeDataRate/Sum of PlaneRelativeDataRate of all planes that need allocation).*

   *floor - rounds down to an integer value dropping the fractional part.

5. Adjust for minimum allocation requirement

   AdjustmentRequired = false

   For each plane needs allocation (PlaneBlockAllocFinal == false)

   If PlaneBufAlloc < PlaneMinAlloc

   AdjustmentRequired = true

   PlaneBufAlloc = PlaneMinAlloc

   PlaneBlockAllocFinal = true

   BlocksAvailable = BlocksAvailable - PlaneMinAlloc

   If AdjustmentRequired = true

   Go back to step 4

Example Method 2

This allocation is based on the Watermark calculations and helps to distribute the buffer more optimally to achieve consistent latency levels supported across all planes.

- For each enabled plane, calculate buffer allocation needed for all Latency levels 1 to 7.
- Calculate the total buffer allocation needed for each latency level by adding the individual allocation of all enabled planes that are using each DBUF.
- Choose the max latency level that can be supported with the available display buffer. For each enabled plane, program/enable all watermarks up to that latency level.
- Allocate the buffer to the planes as required by the latency level chosen.
- Use method 1 to allocate any remaining buffer.

Buffer Allocation Re-distribution

When an additional pipe is getting enabled, or an existing pipe requires more buffer to support a new mode or is disabled, buffer reallocation may be necessary for proper display functionality. Whenever a portion of the allocated buffer is taken away from one pipe and allocated to a different pipe, the following sequence should be followed to make sure that there are no buffer allocation overlaps at any point of time.

1. *For each pipe whose allocation is reduced*
    a. *Program the new buffer allocation.*
    b. *Wait for VBlank of that pipe for new allocation to update.*
2. *For each pipe whose allocation is increased*
    a. *Program the new buffer allocation.*
    b. *Wait for VBlank of that pipe for new allocation to update.*

Display Buffer Allocation and Watermark Programming Prior to OS Boot

Basic programming of the display buffer and watermarks to allow limited display usage prior to OS boot:

This will not allow package power saving states.

Supported usages:

- Up to 4 pipes enabled at once
- Up to one universal plane enabled per pipe. No cursor.
- Linear or Xtile memory
- Any RGB frame buffer pixel format 32bpp or less, without compression
- Any supported screen resolution
- Downscaling less than or equal to 12.5%

Enable DBUFs following the Multi-Buffer Enabling and Allocation Requirements.

Allocate 160 blocks per pipe

- Pipe A: 0-159, Pipe B: 160-319, Pipe C: 320-479, Pipe D: 480-639
- PLANE_BUF_CFG_<plane number>_A = 0x009F0000
- PLANE_BUF_CFG_<plane number>_B = 0x013F00A0
- PLANE_BUF_CFG_<plane number>_C = 0x01DF0140
- PLANE_BUF_CFG_<plane number>_D = 0x027F01E0

Set level 0 watermarks for any enabled plane to 160 blocks and 2 lines.

- PLANE_WM_<plane number>_<pipe>_0 = 0x800080A0

The higher-level watermarks for any enabled plane must have bit 31=0 to keep the low power watermarks disabled.

VGA

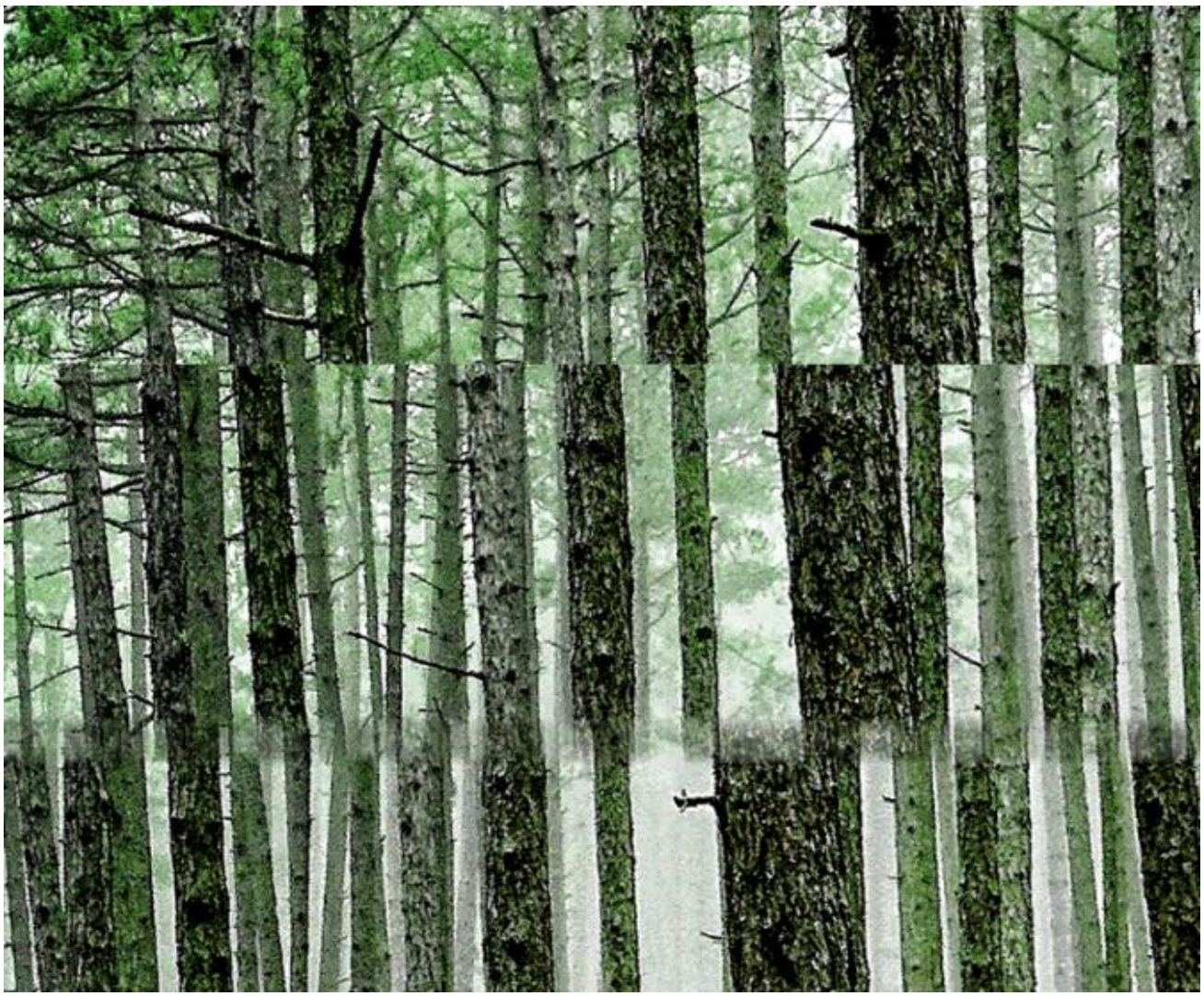
The VGA Control register is located here. The VGA I/O registers are located in the VGA Registers document.

VGA_CONTROL

Smooth Sync Tear Reduction for Asynchronous Flips

When async flips are enabled to reduce latency (e.g., when gaming), onscreen tears caused by the immediate transition from an older image to a newer image within a frame can be observable. The Smooth Sync feature uses both blending and dithering to smoothly transition from the old image to the new image over a programmable number of scanlines.

The picture below illustrates the existing single line transition on the top and the blended Smooth Sync solution on the bottom.



Enabling Sequence

To enable this feature, two adjacent Planes (referred to as the Back and Front Planes) will need to be used to blend between the old and the new images.

1. Software will configure and enable the Back Plane first
    a) Software will enable Smooth Sync **only** within the **Back** Plane's **PLANE_CTL** register
        i. Hardware will automatically assign the adjacent top Plane as the Front Plane (i.e., if Smooth Sync is enabled on Plane 1, then Plane 1 is the Back Plane and hardware will assign Plane 2 as the Front Plane)
        ii. Software must ensure that there is a Plane above the Back Plane (i.e., the Back Plane cannot be the topmost Plane), and the adjacent top Plane is of the same type as the Back Plane (i.e., HDR or SDR)
    b) The Back Plane must be opaque for the smoothing algorithm to work correctly. The Front Plane will automatically control when it is transparent and opaque (see the flow below)
    c) The Plane cannot be configured for scaling, or a planar pixel format
    d) Software should disable the Flip Done interrupts/messages for the Back Plane (**DE_PIPE_INTERRUPT**)
        i. If Flip Done interrupts are not masked by Software, then it will be responsible for managing/ignoring the Flip Dones from the Back Plane
2. Software will then configure and enable the Front Plane
    a) The Front Plane must have an identical configuration as the Back Plane (i.e., pixel format, size, panning, color correction, etc.) with the same surface base at the beginning. Notes:
        i. The Smooth Sync enable in the Plane's **PLANE_CTL** register should **not** be set
        ii. The number of lines of blending will need to be configured via the Step Size in the **PLANE_COLOR_CTL** for both Planes
            I. The number of blending lines will be equal to Step Size * 16
            II. This needs to be programmed in the Back Plane, so that it knows when to generate the Flip Done
    2. Software must ensure that the Plane that it is configuring to be the Front Plane is adjacent and above the Back Plane

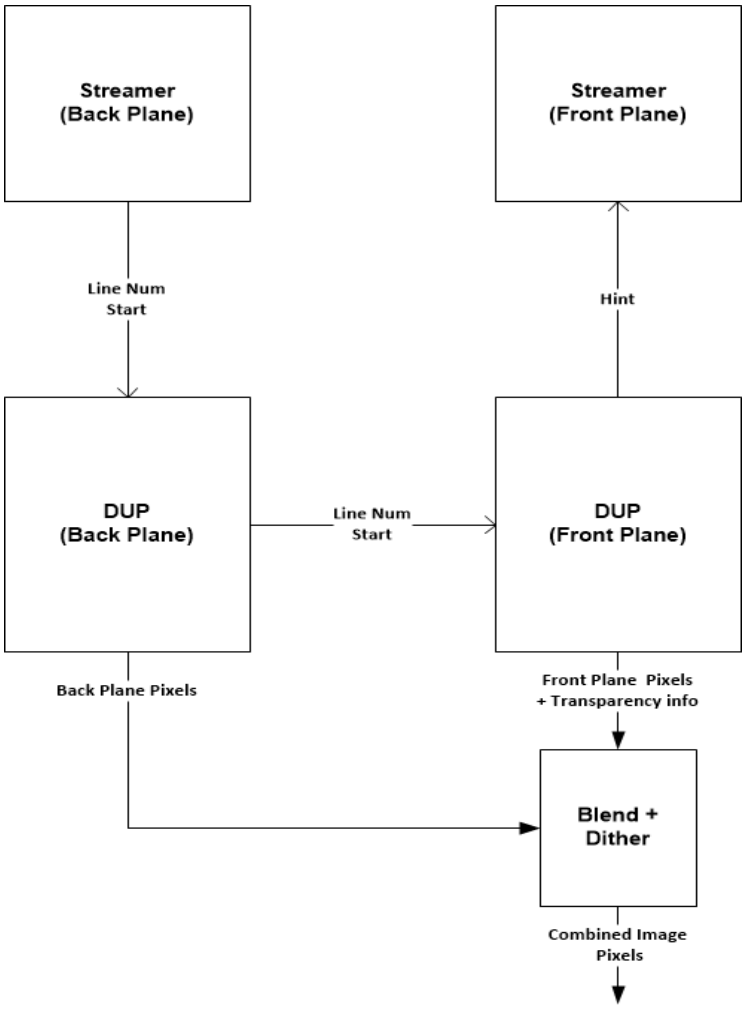
Disabling Sequence

1. Disable the Front Plane first (**PLANE_CTL**)
2. Disable Smooth Sync (**PLANE_CTL**) on the Back Plane

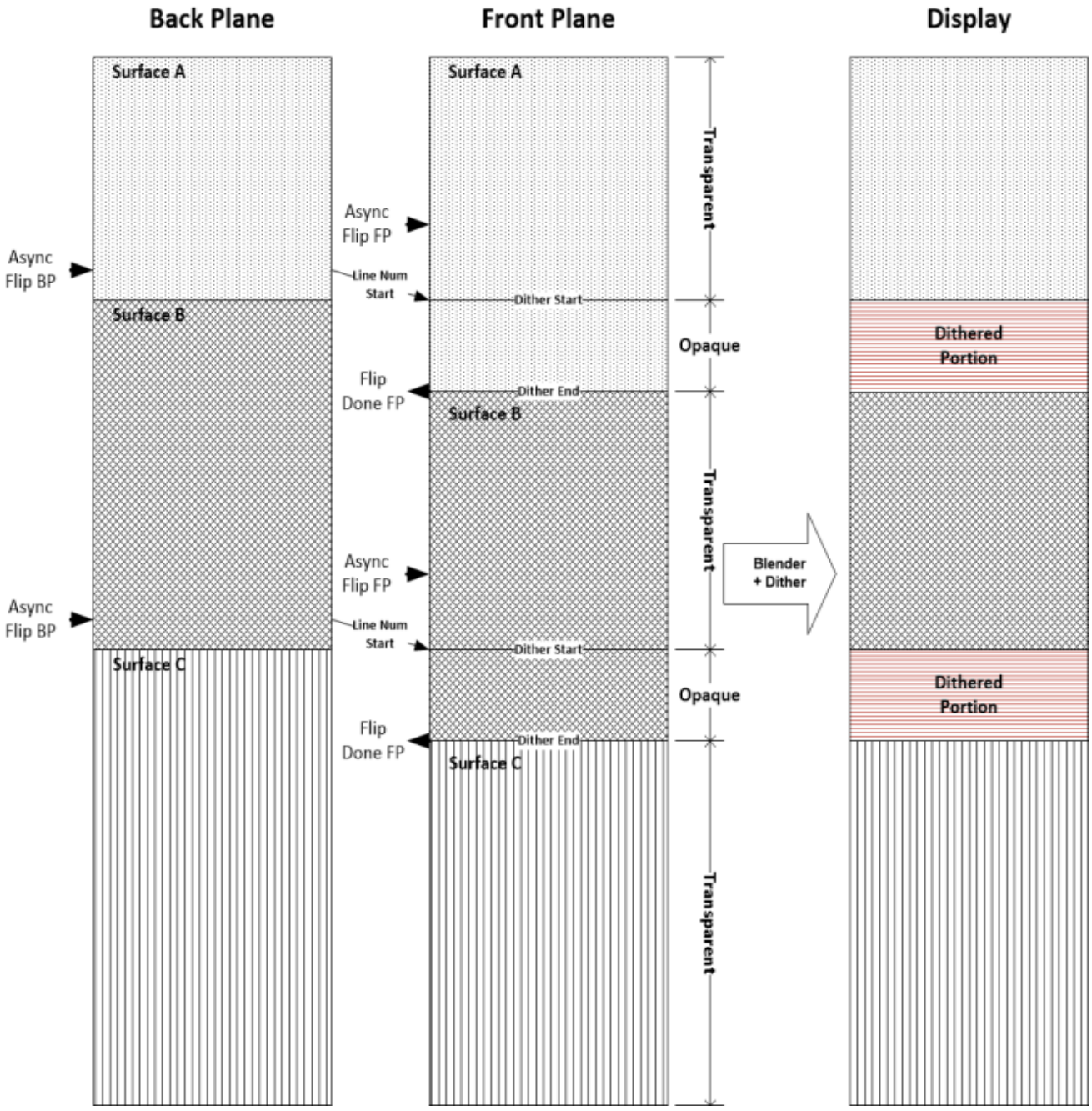
Smooth Sync Flow

1. Both Planes are configured identically with the same surface (A) base at the beginning, but the Front Plane will be transparent (this is handled by hardware)

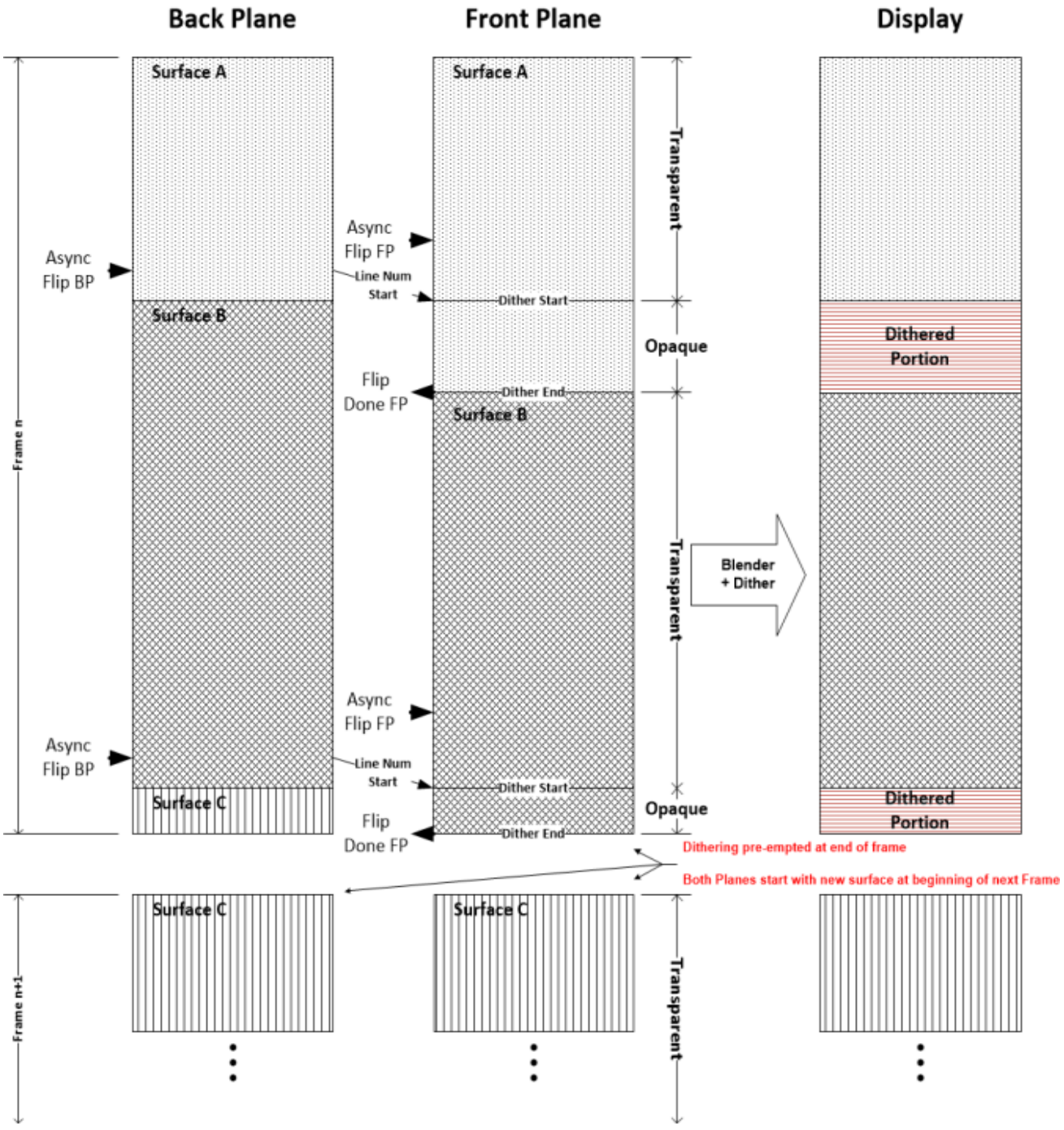
2. When the Driver receives an Async flip from the OS, the Driver will deliver the flip (i.e., updating the **PLANE_SURF** register) to the Front Plane first followed immediately to delivering the flip to the Back Plane.
3. When a given Plane receives its flip:
    a) The Back Plane will start to fetch the new surface (B) and will advertise to the Front Plane the line number of where surface B will start.
    b) The Front Plane will continue to fetch surface A and will remain transparent until it reaches the surface B start line it received from the Back Plane.
4. When the Front Plane reaches the surface B start line it will become opaque and starts the dithering process
    a) The dithering region is programmable (via the Smooth Sync Dithering Step Size of **PLANE_COLOR_CTL**) and is equal to Step Size * 16 lines
5. When the Planes have reached either the end of the dithering region or the end of frame:
    a) The Front Plane will generate the Flip Done for Software. Note that Software is not allowed to send any additional async flips to either Front or Back Planes until this point.
    b) The Front Plane will start fetching surface B and will go back to being transparent.



Async Flips during Frame

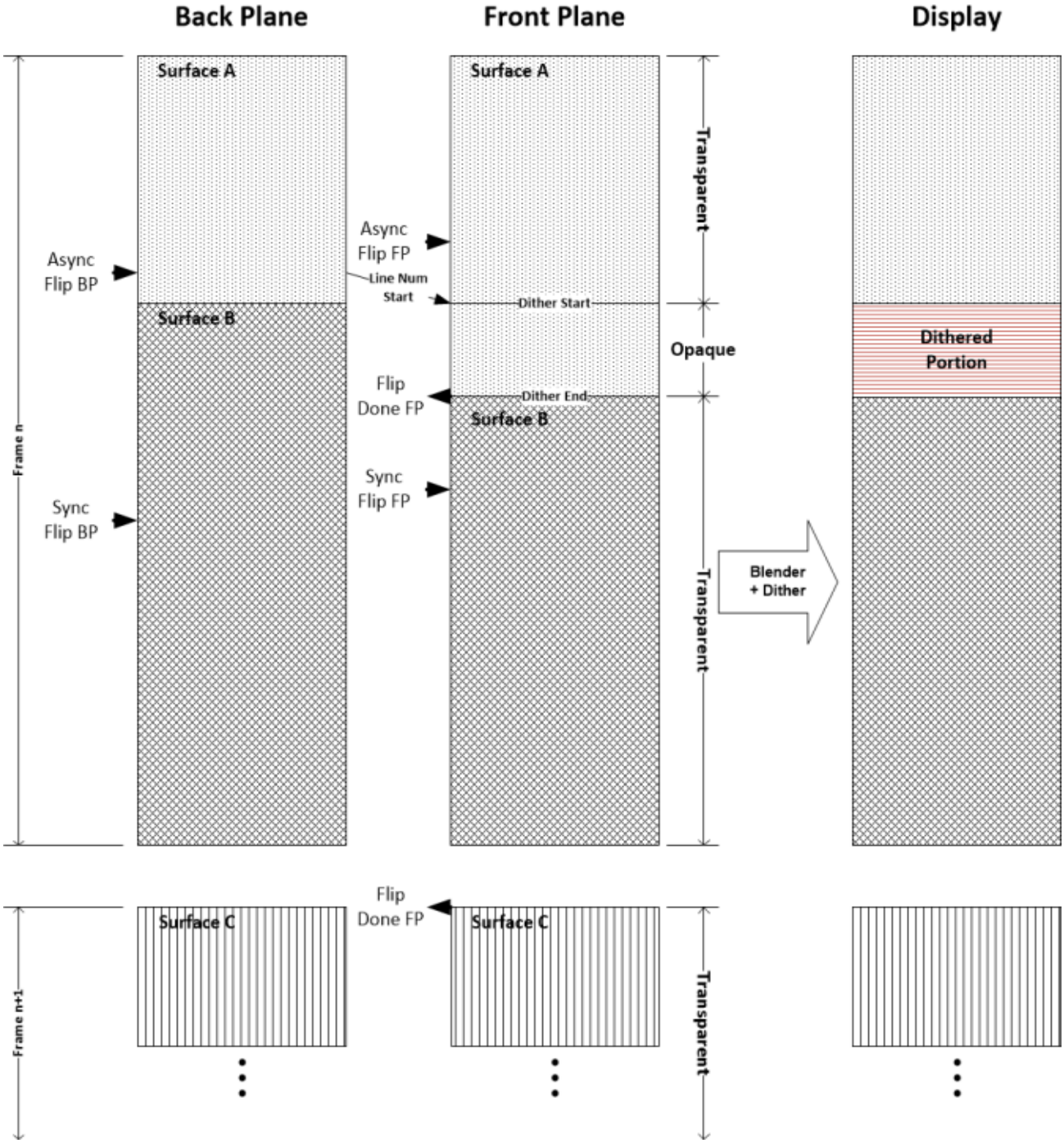


Async Flips at End-Frame



Async to Sync Flips

Software can dynamically send Async and Sync Flips to the Planes while Smooth Sync is enabled. Software will still need to send the Flips (synchronous) to both Front and Back Planes, and the expectation from Hardware remains that there should not be any async flips until the Flip Done for the Sync Flip is received.



Restriction Summary

- Both Planes (Back and Front) must be adjacent to each other (e.g. Plane 1 and Plane 2), must be configured exactly the same (unless noted otherwise within this page), and must be of the same type (i.e. HDR or SDR).
    - No Plane Scaling, or planar pixel formats allowed.
- Software must set the Smooth Sync Plane Enable within the **Back** Plane's **PLANE_CTL** register
- Software can only set the Smooth Sync Plane Enable within a Plane that has an adjacent top Plane (i.e., it cannot be set within the topmost Plane)
- The Back Plane must be opaque for the smoothing algorithm to work properly
- Flips need to be delivered to the Front Plane first followed by the Back Plane
- Software must not send any async flips to either Plane while a flip is in progress (either async or sync). Async flips can be sent once Software receives the Flip Done (generated by Front Plane)
- DBUF allocation should be set to a minimum for the Front Plane

Cursor Plane

Planes
CUR_CTL
CUR_BASE
CUR_POS
CUR_PAL
CUR_FBC_CTL
CUR_SURFLIVE
PLANE_BUF_CFG
PLANE_WM
DPRC_INSTANCES
CUR_COLOR_CTL
CUR_VF
CUR_CSC_COEFF
CUR_PRE_CSC_GAMC_INDEX
CUR_PRE_CSC_GAMC_DATA

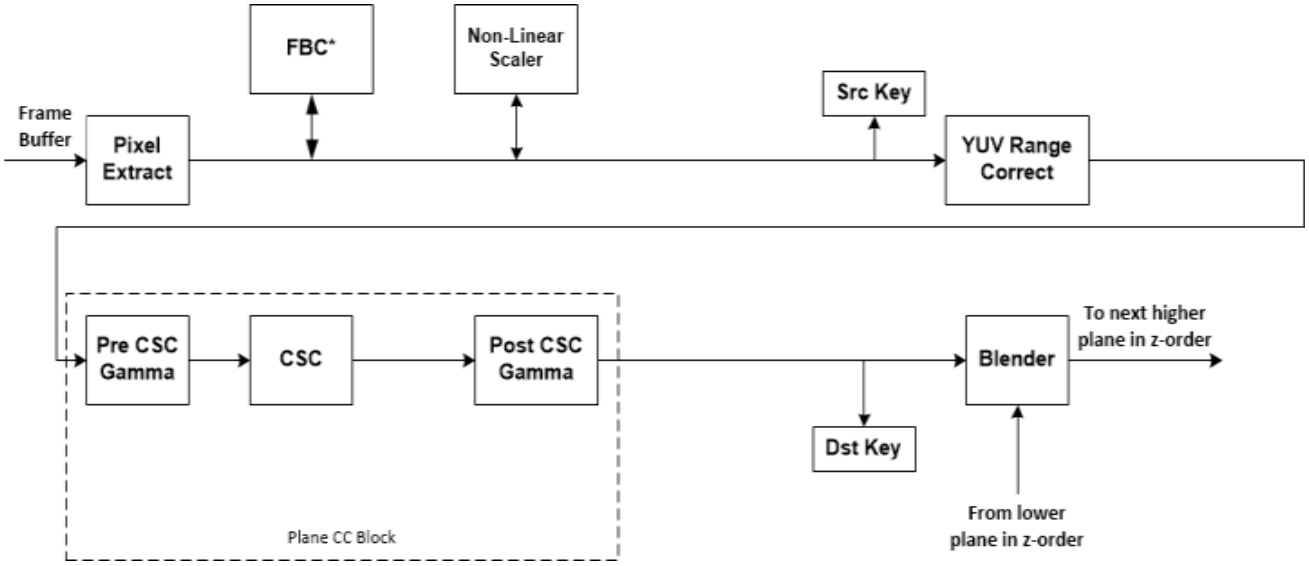
Many cursor registers are double-buffered and armed (see each register access description). The active registers will be updated on the vertical blank or when pipe is disabled, after the CUR_BASE register is written, or when cursor is not yet enabled, providing an atomic update of those registers together with the CUR_BASE register.

Universal Plane

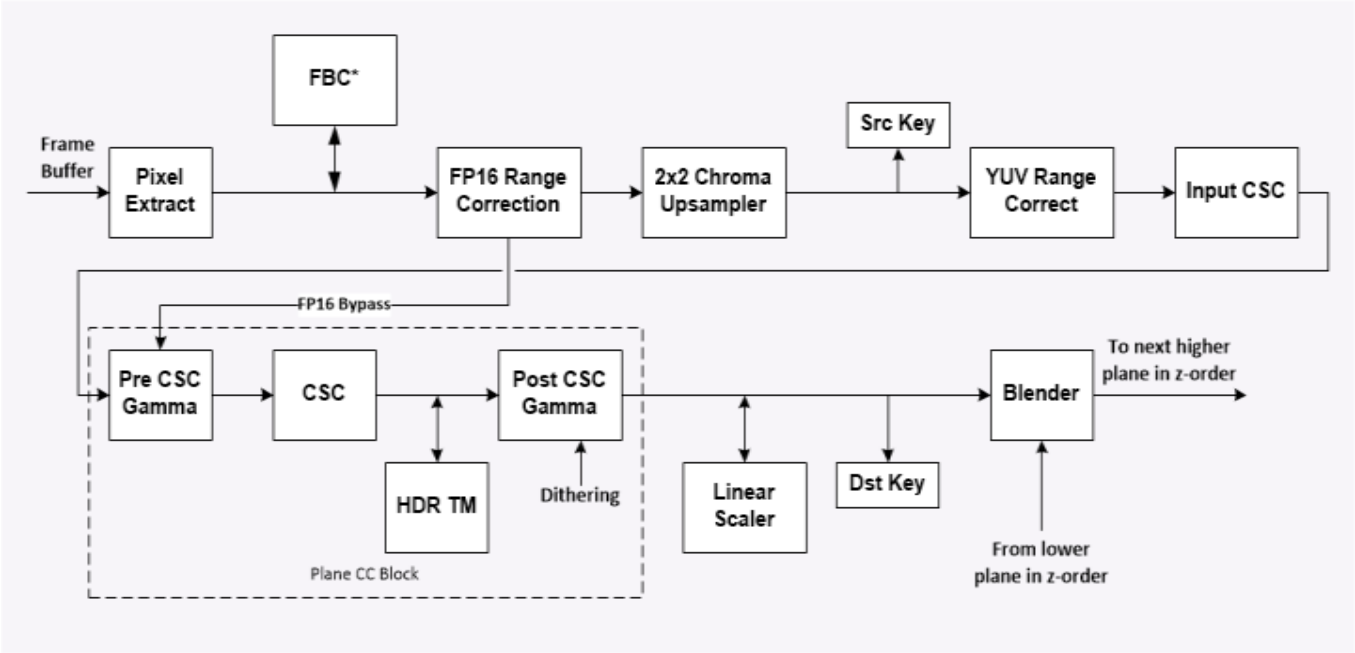
Planes
PLANE_CTL
PLANE_AFLIP_WIN_CTL
PLANE_STRIDE
PLANE_POS
PLANE_SIZE
PLANE_SURF
PLANE_LEFT_SURF
PLANE_SURFLIVE
PLANE_WM
PLANE_BUF_CFG
PLANE_OFFSET
PLANE_KEYVAL
PLANE_KEYMSK
PLANE_KEYMAX
PLANE_PRE_CSC_GAMC_INDEX
PLANE_PRE_CSC_GAMC_DATA
PLANE_POST_CSC_GAMC_INDEX
PLANE_POST_CSC_GAMC_DATA
PLANE_COLOR_CTL
PLANE_PIXEL_NORMALIZE
PLANE_INPUT_CSC_COEFF
PLANE_INPUT_CSC_PREOFF
PLANE_INPUT_CSC_POSTOFF
PLANE_CSC_COEFF
PLANE_CSC_PREOFF
PLANE_CSC_POSTOFF
PLANE_PRE_CSC_GAMC_DATA_ENH
PLANE_PRE_CSC_GAMC_INDEX_ENH
PLANE_POST_CSC_GAMC_DATA_ENH
PLANE_POST_CSC_GAMC_INDEX_ENH
PLANE_CUS_CTL
PLANE_CC_VAL
PLANE_VF
PLANE_POST_CSC_GAMC_SEG0_DATA_ENH
PLANE_POST_CSC_GAMC_SEG0_INDEX_ENH
SEL_FETCH_PLANE_CTL

Many of the plane control active registers will be updated on the vertical blank or when pipe is disabled, after the surface base address register is written, or when the plane is not yet enabled, providing an atomic update of those registers together with the surface base address register.

SDR Planes



HDR Planes



*FBC not on all planes or pipes.

HDR planes have higher bit precision throughout the processing stages.

PIPE_MISC HDR Mode must be enabled to get the higher precision output from the HDR planes, bypassing the SDR planes in blending.

Dithering after the color conversion is done down to 12 bits and is only available in HDR planes

Planes 1-3 are the HDR planes. The other planes are SDR planes.

Plane Pixel Formats

ARGB

Name	Alpha	Red	Green	Blue
RGB 32-bit 8:8:8:8 BGRA	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBA	31:24	7:0	15:8	23:16
RGB 32-bit 2:10:10:10 BGRA	31:30	29:20	19:10	9:0
RGB 32-bit 2:10:10:10 RGBA	31:30	9:0	19:10	29:20
RGB 64-bit 16:16:16:16 Float BGRA (FP16) Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 Float RGBA (FP16) Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	15:0	31:16	47:32
RGB 64-bit 16:16:16:16 UINT BGRA Each component is 16 bit unsigned integer	63:56	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 UINT RGBA Each component is 16 bit unsigned integer	63:56	15:0	31:16	47:32
RGB 32-bit XR_BIAS 2:10:10:10	31:30	9:0	19:10	29:20
16-bit BGR 5:6:5	N/A	15:11	10:5	4:0
64-bit formats supported only on the HDR planes.				

YUV 420 Planar

Name	Y	U	V
YUV 4:2:0 8 bpc - NV12	7:0	15:8	7:0
YUV 4:2:0 10 bpc - P010	15:6	31:22	15:6
YUV 4:2:0 12 bpc - P012	15:4	31:20	15:4
YUV 4:2:0 16 bpc - P016	15:0	31:16	15:0

YUV 422 Packed

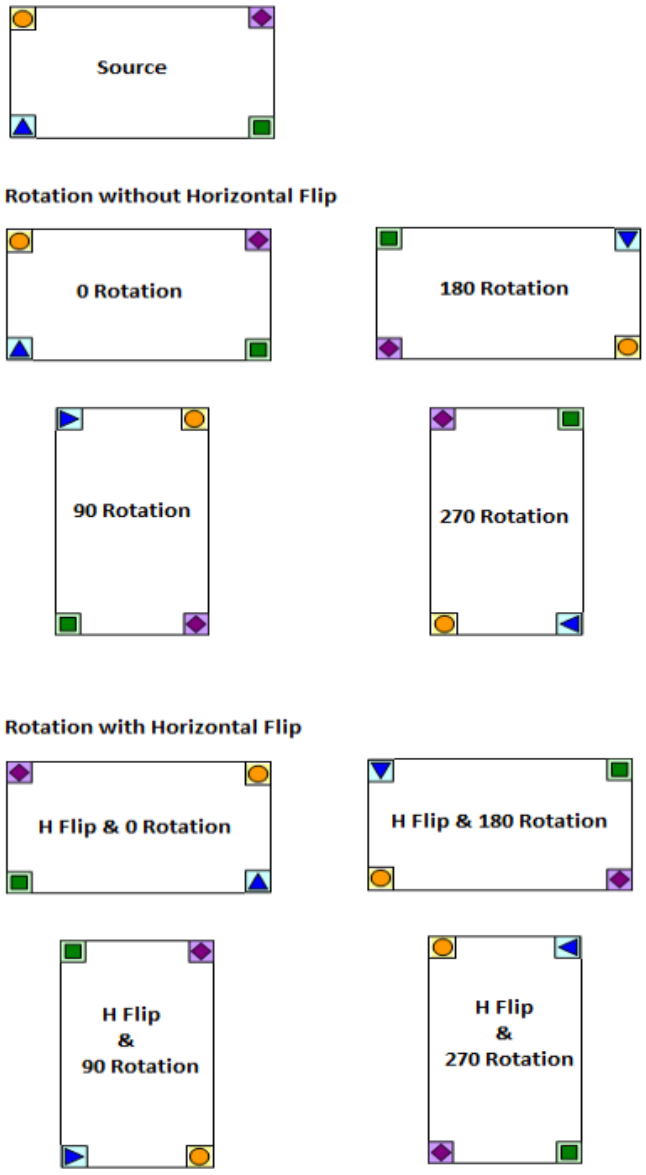
Name	Y1	U	Y2	V
YUV 4:2:2 YUYV 8 bpc	7:0	15:8	23:16	31:24
YUV 4:2:2 UYVY 8 bpc	15:8	7:0	31:24	23:16
YUV 4:2:2 YVYU 8 bpc	7:0	31:24	23:16	15:8
YUV 4:2:2 VYUY 8 bpc	15:8	23:16	31:24	7:0
YUV 4:2:2 YUYV 10 bpc - Y210	15:6	31:22	47:38	63:54
YUV 4:2:2 YUYV 12 bpc - Y212	15:4	31:20	47:36	63:52
YUV 4:2:2 YUYV 16 bpc - Y216	15:0	31:16	47:32	63:48

YUV 444 Packed

Name	Ignored	Y	U	V
YUV 4:4:4 8 bpc	31:24	23:16	15:8	7:0
YUV 4:4:4 10 bpc - Y410	31:30	19:10	9:0	29:20
YUV 4:4:4 12 bpc - Y412	63:52	31:20	15:4	47:36
YUV 4:4:4 16 bpc - Y416	63:48	31:16	15:0	47:32

Horizontal Flip

When plane horizontal Flip is enabled with rotation, the horizontal flip operation is logically performed first followed by the rotation operation. The sample results are shown below.



DSC

DSC is used for compressing pixel data on the link to a monitor. This allows the link to support a higher resolution than otherwise or save power on the link or buffering in the monitor. The VDSC feature compresses the pipe output raw pixel bytes into a compressed byte stream as per VESA DSC specification.

A VDSC engine (instance or branch) operates with 1 pixel per clock (PPC) throughput. This becomes a limitation when the pixel clock is higher than the VDSC clock (CDCLK), so multiple VDSC engines are used to increase throughput. The DSS (display stream splitter/joiner) function can be enabled to split the pixel stream across multiple VDSC engines in each pipe. In a typical use case, the stream is split into 2 horizontally equal parts for 2 VDSC engines compress each part in parallel and then the small joiner block re-joins the compressed byte streams into a single compressed byte stream that is sent to the transcoder. Each VDSC engine is configured with its own Picture Parameter Set (PPS) register set.

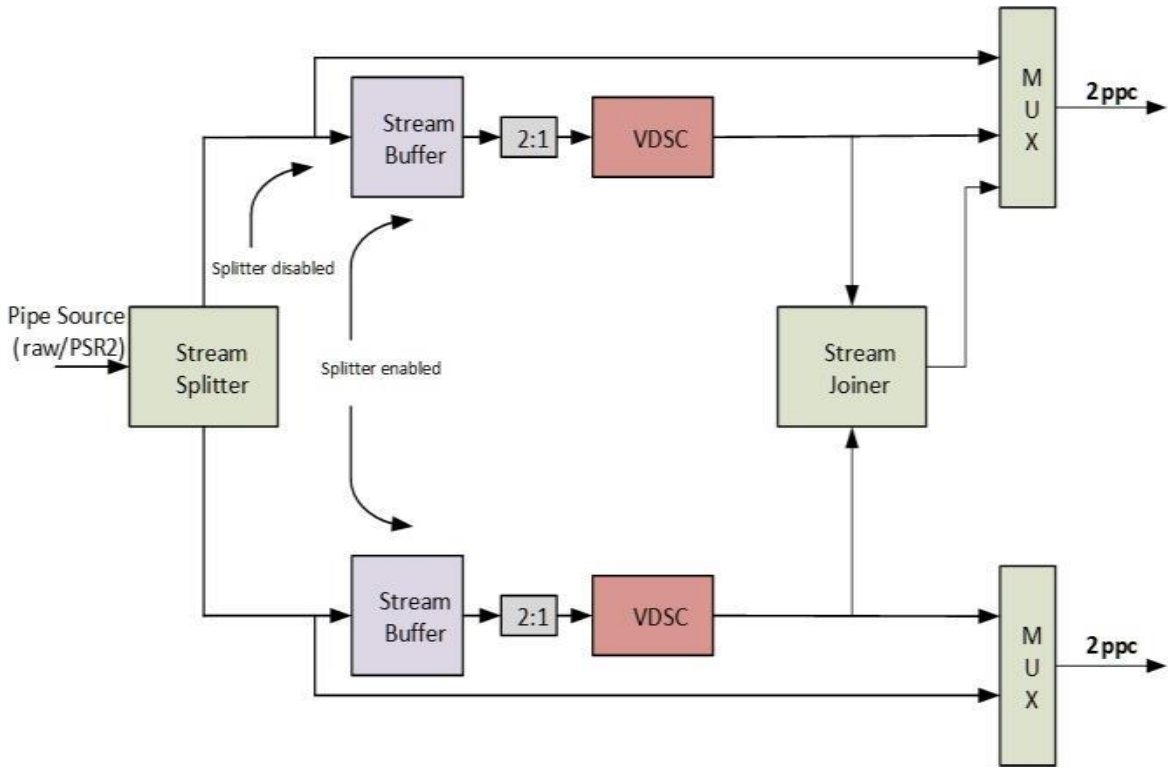
The DSS splitting block is also used for MSO/CoG.

Below is the list of DSS and DSC encoder registers. There are instances of the DSC registers for each VDSC engine.

Register List
PIPE_DSS_CTL1
PIPE_DSS_CTL2
DSC_PICTURE_PARAMETER_SET_0 • Field vbr_enable setting as "1" is not supported.
• Field "Allow DB Stall" is not from DSC standard.
DSC_PICTURE_PARAMETER_SET_1
DSC_PICTURE_PARAMETER_SET_2
DSC_PICTURE_PARAMETER_SET_3
DSC_PICTURE_PARAMETER_SET_4
DSC_PICTURE_PARAMETER_SET_5
DSC_PICTURE_PARAMETER_SET_6
DSC_PICTURE_PARAMETER_SET_7
DSC_PICTURE_PARAMETER_SET_8
DSC_PICTURE_PARAMETER_SET_9
DSC_PICTURE_PARAMETER_SET_10
DSC_PICTURE_PARAMETER_SET_11
DSC_PICTURE_PARAMETER_SET_12
DSC_PICTURE_PARAMETER_SET_13
DSC_PICTURE_PARAMETER_SET_14
DSC_PICTURE_PARAMETER_SET_15

Register List
DSC_PICTURE_PARAMETER_SET_16 • Fields "slice_row_per_frame" and "slice_per_line" are not from DSC standard.
DSC_RC_BUF_THRESH_0
DSC_RC_BUF_THRESH_1
DSC_RC_RANGE_PARAMETERS_0
DSC_RC_RANGE_PARAMETERS_1
DSC_RC_RANGE_PARAMETERS_2
DSC_RC_RANGE_PARAMETERS_3

Block Diagram



Two DSC engines within a pipe.

The top mux output is designated as left (front) and bottom mux output is designated as right (back) in the case of split streams. The stream joiner (small joiner) output can only feed into the top (left) mux output.

If the input frame is divided into an even number of slices by enabling the splitter, then the corresponding branch parameters such as picture width and slice width need to be adjusted accordingly. For example, if the input frame (pipe source in the diagram above) is divided into 4 slices per scanline,

then the slice width on each branch will be HACTIVE/4 and the picture width on each branch will be HACTIVE/2.

Modes of Operation Within a Single Pipe

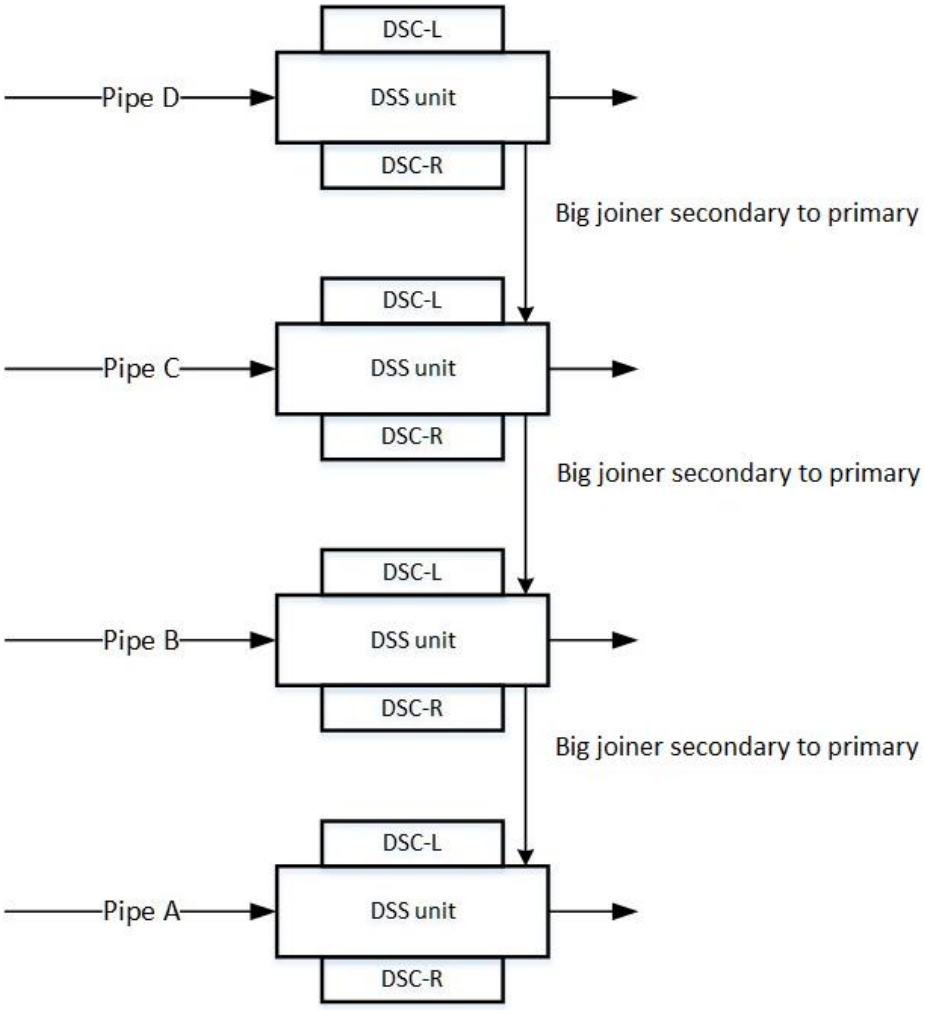
Mode	Splitter Enabled	VDSC Enabled	Small Joiner Enabled	Usage	Output Branch
Bypass mode	No	No	No	Uncompressed non-CoG/MSO.	Left
Compressed un-joined mode*	No	Yes, left	No	Single engine compressed, non-CoG/MSO. Pixel rate is limited to 1PPC.	Left
Uncompressed split mode	Yes	No	No	Uncompressed CoG/MSO.	Left and right
Compressed split mode	Yes	Yes, left and right	No	Compressed CoG/MSO.	Left and right
Compressed split and join mode	Yes	Yes, left and right	Yes	Compressed non-CoG/MSO	Left

*Although some PPS configurations and screen resolutions can be supported with a single VDSC engine (1 PPC), the recommendation is to use 2 VDSC engines where possible to keep the CDCLK lower for power savings, while meeting the DP/HDMI PPR spec provided slice size < DPCD provided MaxSliceWidth.

DSI Configuration	Splitter	Joiner	DSC to DSI Mapping	Notes
Single link - single pipe	Yes	Yes	DSC (L) + DSC (R) -> DSIx	Small joiner output always on left branch.
Dual independent links	Yes	Yes	PipeX -> DSC (L) + DSC (R) -> DSI0 PipeY -> DSC (L) + DSC (R) -> DSI1	Each pipe will compress pixels to one of the DSI ports.
Dual link - single pipe	Yes	No	Pipe A DSC(L) - DSI0 Pipe A DSC(R) - DSI1	Both ports bound to the same pipe AND DSI0 configured for Port Sync Mode. Only pipe A can be used.

Big Joiner

The DSC function can achieve compression over 2 pipes to support resolutions that require more bandwidth or pixel width than a single pipe can support, such as 8K. The frame is divided and processed in parallel by 2 adjacent pipes, compressed by the 2 VDSC engines in each pipe (small joiner), then the compressed pixels streams from the pipes are joined (big joiner) into a single compressed pixel stream.



Big joiner mode cannot be used if compression is bypassed in VDSC.

Big joiner mode cannot be used if small joiner is not enabled.

Big joiner mode cannot be used in conjunction with PSR1, PSR2, or Panel Replay

The front-end display requirements in the case of big joiner mode are the same as in a dual transcoder/port 8K configuration. Audio works same as in other display configurations.

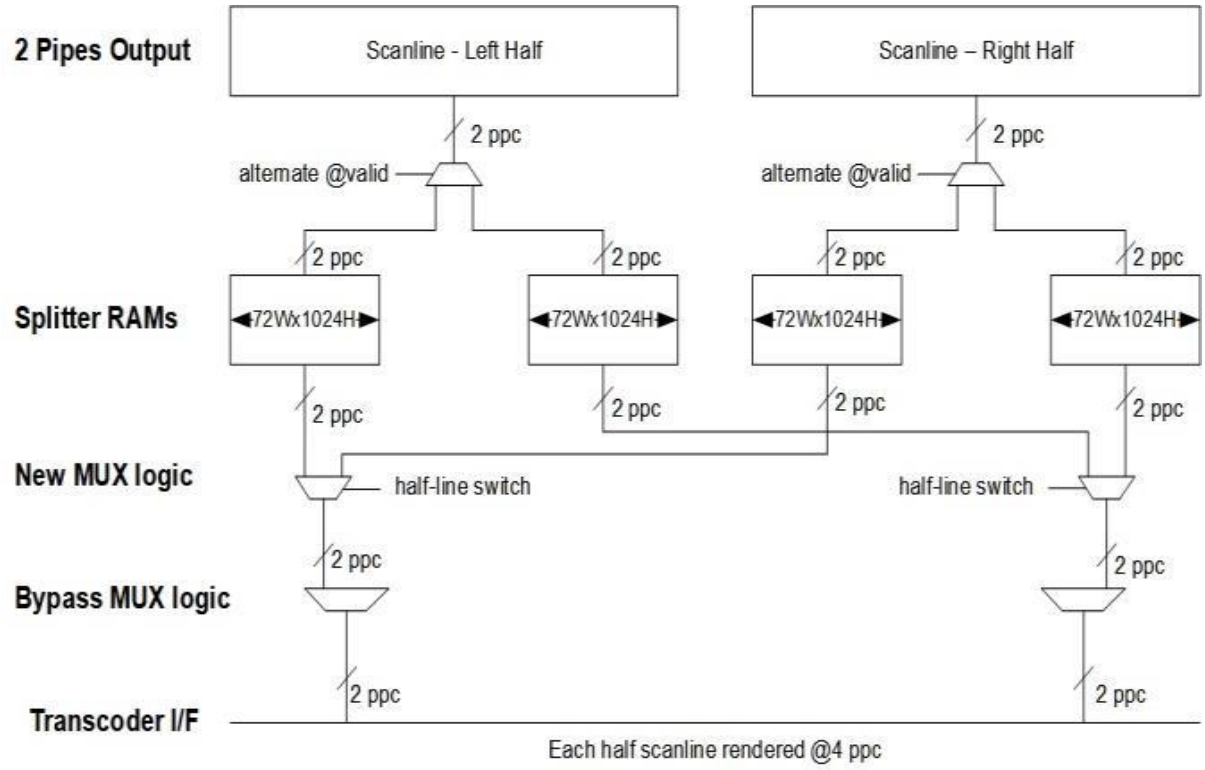
Any two adjacent pipes can be joined together as shown below.

The primary pipe of the pair sends the combined stream output to its transcoder. The secondary pipe transcoder is not enabled.

- Pipe A+B big joined: Pipe A is primary and B is secondary.
- Pipe B+C big joined: Pipe B is primary and C is secondary.
- Pipe C+D big joined: Pipe C is primary and D is secondary.

Uncompressed 2 Pipe Joiner

Two pipes can be big joined without compression to support uncompressed resolutions that require more bandwidth or pixels than a single pipe can support.



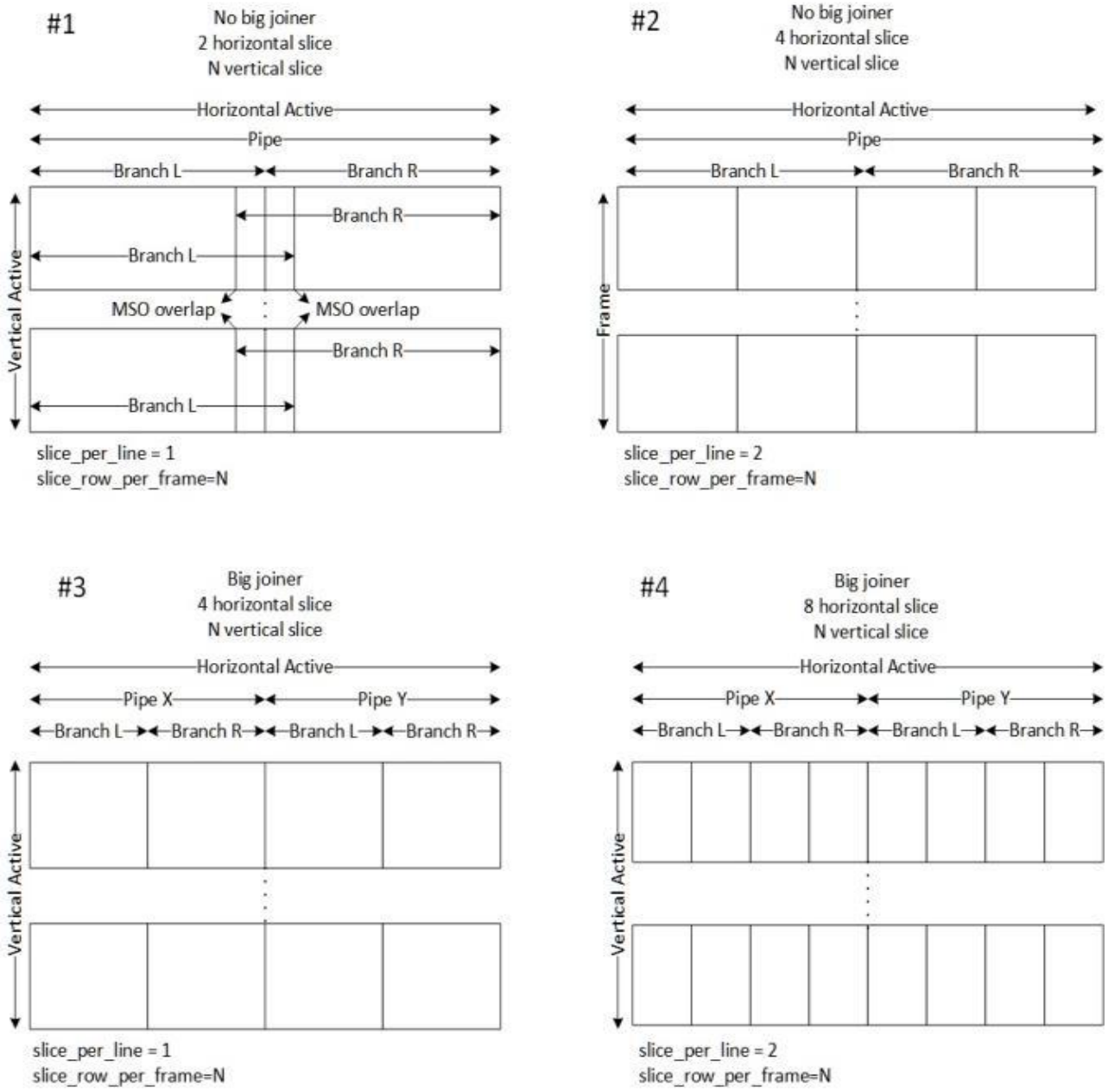
- CoG is not supported in uncompressed 2 pipe joiner mode.
- The requirement for adjacent pipes and primary pipe selection are the same as with compressed big joining.
- Total scanline size must be a multiple of 4 pixels.
- This feature is supported with all pixel formats, that is, RGB 4:4:4, YUV 4:2:2, and YUV 4:2:0.

VDSC Slice Options

Slices separate the frame into rectangular regions that are compressed independently. There are multiple slice width and height options, with dependencies on the DSC standard, output port standards, selective update usage, image quality recommendations, and implementation restrictions.

Joining increases the possible number of slices within a line.

Example Slice Arrangements



- Each DSC branch/device/instance supports multiple slices.
- 1, 2, or 4
- The number of slices in a pipe is multiplied by the number of DSC devices joined.
- The number of slices input to the transcoder is multiplied by the number of pipes joined and slices in each pipe.
- For non-MSO use cases, Horizontal active must be an integer multiple of slice width in pixels. For MSO use cases, overlap pixels need to be included.

- Each DSC branch slice_per_line = number of horizontal slices this DSC branch is processing = number of horizontal slices for full frame / number of DSC branches enabled for this video stream
- DP restriction: Active video height must be an integer multiple of slice height.
- Each DSC branch slice_row_per_frame = number of vertical slices = vertical active / slice height
- VDSC spec implies that 108 lines is an optimal slice height, but any size can be used as long as vertical active integer multiple and maximum vertical slice count requirements are met.
- The default slice height can be set to a value that satisfies the above as follows. Slice height = ~108 and picture height = N * slice height where N is an integer.
- Each DSC branch pic_height = vertical active
- Each DSC branch pic_width = horizontal active / number of DSC branches enabled for this video stream + overlap pixels. PPS transmitted to receiver must use full width.

Slice Design Requirements

	RGB	YUV 444
Slice Width	Multiple of 1 pixels	Multiple of 1 pixels
Slice Width	Minimum 18 pixels	Minimum 18 pixels
Slice Height	Multiple of 1 line	Multiple of 1 line
Slice Height	Minimum 1 line	Minimum 1 line
Slice Height	Maximum 4095 lines	Maximum 4095 lines
Pixels in Slice	Must be >= 15,000	Must be >= 15,000

Output Bits Per Pixel Calculations

The following rules determine the highest DSC output bpp that the design can support in various compression scenarios.

Output Bits Per Pixel Calculations
Pipe BW check: Pixel clock < PPC * CDCLK frequency * Number of pipes joined • PPC = 1 or 2 depending on number of DSC engines used within the pipe. • This is for reference on pipe pixel clock limitation as an input to the splitter.
Link BW check: Output bpp < Number of lanes * DDICLK frequency * Bits per lane / Pixel clock
• DSI bits per lane = 8
• DisplayPort 8b/10b bits per lane = 8
• DisplayPort 2 128b/132b bits per lane = 32
DPT BW check (DP2 UHBR): Output bpp * Pixel clock < DDICLK frequency * 72 bits

Output Bits Per Pixel Calculations
• This check only limits DP2 UHBR use cases. Other rates do not hit this limit.
Big Joiner BW check: Output bpp <= PPC * CDCLK frequency * Big joiner interface bits / Pixel clock • This check only needed if big joiner is enabled
• DP and DSI Big interface joiner bits = 24
Small Joiner RAM check: Output bpp <= Small joiner RAM size / Horizontal width in pixels • This check only needed if small joiner is enabled. • Note that horizontal width has to account for overlap and dummy pixels in CoG use cases.
• Small joiner RAM size = 138,240 bits per pipe, 276,480 bits for two joined pipes
Float greatest output bpp = MIN(Output bpps from Link BW check, Big joiner BW check, small joiner RAM check, DPT BW check)
Result: Greatest allowed DSC output bpp = INT(Float greatest output bpp) Reduce to integer.

For cases where FEC is enabled, pixel clock is replaced by pixel clock/0.972261 in the above calculations.

Examples
Greatest allowed DSC output bpp example: Resolution = 5120 x 2880 @60 CDCLK:648 , DDICLK:810, PPC:2, DP lanes:4, input Pixel bpp:36, Pixel clock:924 MHz, Horiz. width:5120 Greatest allowed DSC output bpp = INT(MIN (28.05, 50.49, 27)) = INT(27) = 27

Design Support

All pipes support VDSC.

Joined Pipes
2 (big joiner)

Input Pixel Formats
RGB444
YUV444 (requires color channel swap programming)

Input Bits Per Pixel
24
30
36

Maximum Input Width
5120 pixels per pipe, 8192 across joined pipes

Output Bits Per Pixel[1]
8 to 27[1]

Further constrained by the output bits per pixel calculations

DSC Standard
1.1

FEC Support with DSC
Yes

YUV444 Color Channel Swapping

YUV444 with DSC requires programming to swap color channels with pipe output CSC as follows.

Coeff	Bit Field	Color Channel	Without Channel Swapping (RGB to VYU)	With Channel Swapping (RGB to YUV)
Coeff0	[31:16]	Ry	6	0
Coeff0	[15:0]	Gy	7	1
Coeff1	[31:16]	By	8	2
Coeff1	[15:0]			
Coeff2	[31:16]	Ru	0	3
Coeff2	[15:0]	Gu	1	4
Coeff3	[31:16]	Bu	2	5
Coeff3	[15:0]			
Coeff4	[31:16]	Rv	3	6
Coeff4	[15:0]	Gv	4	7
Coeff5	[31:16]	Bv	5	8
Coeff5	[15:0]			

DSC Source Policies

Port Type	Source Policy
eDP	DSC output bits per pixel (bpp) as defined in DPCD. If eDP Sink reports capability for multiple link rates, choose the optimal link rate based on bandwidth calculations.
DP	DSC is enabled only when link bandwidth requires it. DSC output bpp is set to the highest possible value within link bandwidth.

Transport Mandatory DSC Features

The following features are options at DSC level but are mandatory at transport level.

DSC feature	Transport requirement	Transport type	Value
Minimum pixel component bit depth (bpp)	Mandatory	DP 2	4:4:4 - 8 4:2:2 - 7 4:2:0 - 6
DSC bit stream bit depth (bpp)	Mandatory	DP 2	8 to 18 in increments of 1 bpp

Overall DSC Programming Considerations

- Program DSS registers to configure splitting and joining and enable VDSC branches/engines
- Program PPS for each VDSC branch/engine
- Program DP M/N to account for compression amount
- Program transcoder data island packets to transmit the PPS

VDSC compression engines do not support interlaced mode of operation for any transport protocol.

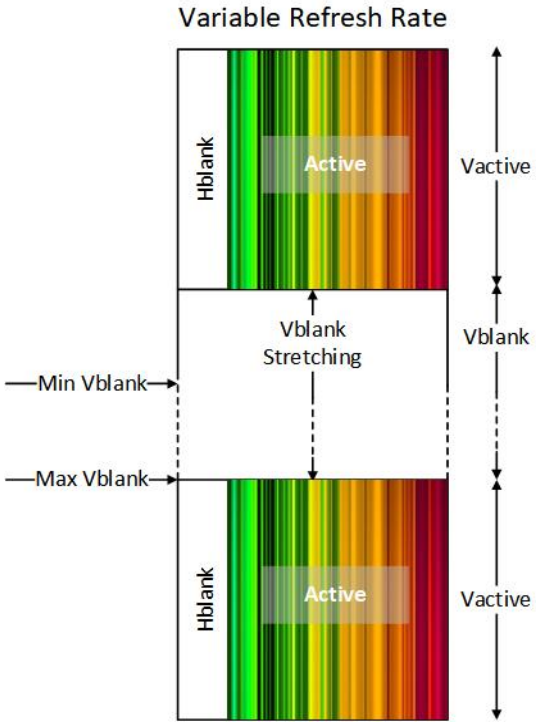
Transcoder

Transcoder VRR Function

VRR Registers
VRR Control TRANS_VRR_CTL
VRR Vmax TRANS_VRR_VMAX
VRR Vmin TRANS_VRR_VMIN
VRR Vsync TRANS_VRR_VSYNC
VRR Status TRANS_VRR_STATUS
VRR Vtotal Previous TRANS_VRR_VTOTAL_PREV
VRR Flipline TRANS_VRR_FLIPLINE
VRR Status2 TRANS_VRR_STATUS2
TRANS_PUSH

VRR Overview

In the adaptive sync mode (a.k.a. VRR or Variable Refresh Rate), the display engine adapts itself to render speed by adjusting its refresh rate dynamically. Adaptive sync mode is supported both on eDP and DP. The dot clock is set to support the peak desired refresh rate for a given resolution and link clock. A minimum and maximum vertical blank (vblank) period is specified and the display hardware stretches or shrinks the actual vblank period based on the required refresh rate.



VRR Details

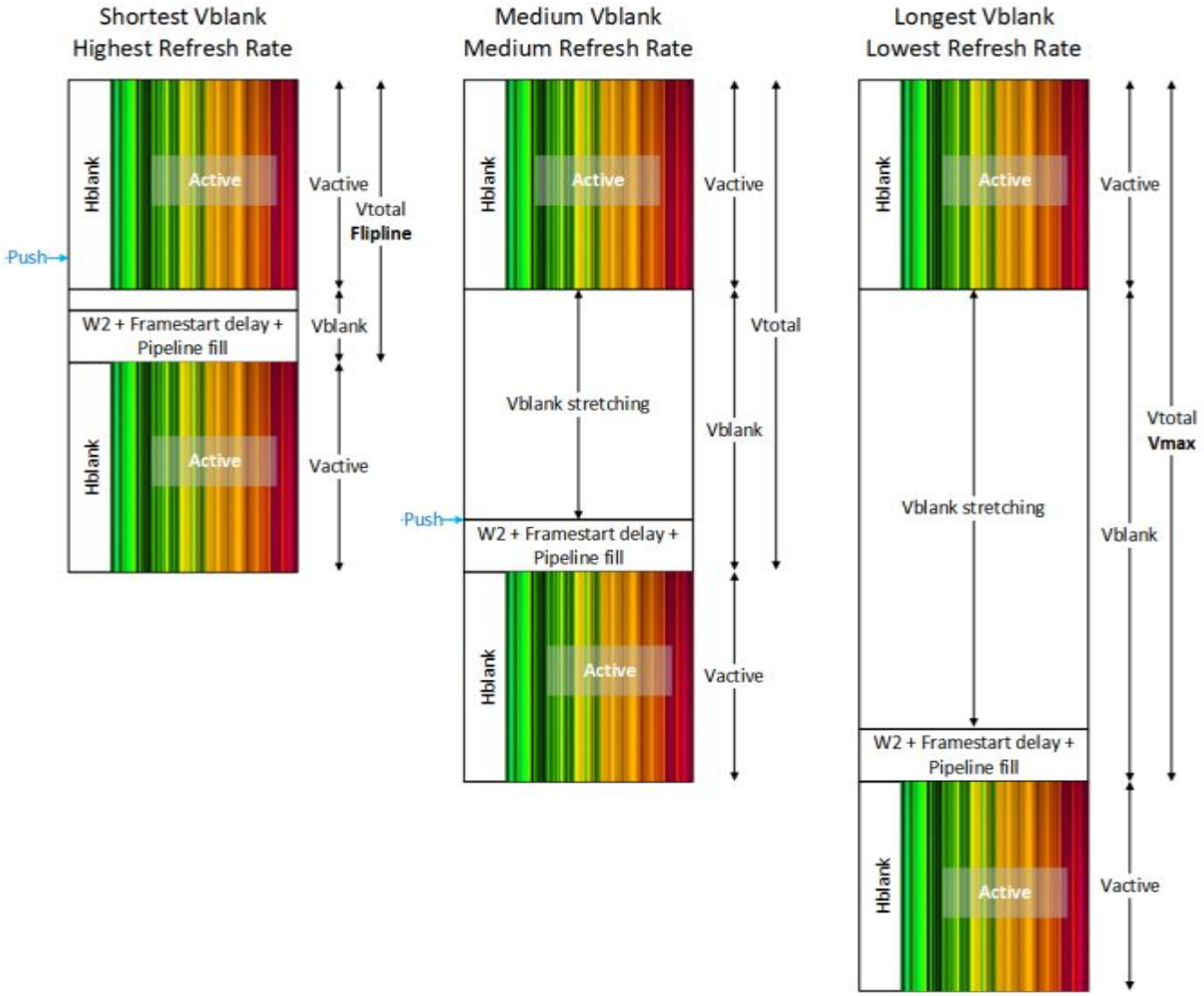
VRR stretches the beginning of vblank until the vblank needs to be terminated, then the Set Context Latency (a.k.a. delay window 2, or W2) happens, double buffer registers are updated (sync flips complete), framestart is triggered (after a delay), the display pipeline fills, and vblank ends. The VRR guardband (TRANS_VRR_CTL) defines the time from the double buffer point to the vblank end (see "High Refresh Rate and Small Vblank Support" page for sizing details).

The overall vblank period is constrained by the registers specifying vertical total (Vtotal), lines maxium (Vmax), and lines minimum (Vmin). The Vtotal contains the Vblank and vertical active (Vactive) region of each display frame. The minimum Vblank must at a minimum contain the framestart delay and pipeline fill (i.e. VRR Guardband).

When there is a frame update (flip or other update), software triggers the termination of vblank by setting the transcoder Send Push bit (push bit). The start of termination may be immediate, or delayed, depending on flip line. The vblank termination is the flip decision boundary, or latest time a push will be serviced. If a push is set after the start of the delayed vblank, it will be held until the next vblank and serviced there. Hardware clears the push bit at the start of the delayed vblank.

Flip Line provides a frame-by-frame programmable limit on the minimum Vtotal. The flip line is programmed before the push. Hardware calculates the flip line decision boundary = Flip line value - Guardband - W2. If push happened before that boundary point, the vblank termination starts at that boundary point. If push happens after that boundary point, the vblank termination starts immediately.

When software does not trigger termination, the vblank automatically terminates when the programmed vertical max (Vmax) is reached. Hardware calculates the Vmax decision boundary = Vmax - Guardband - W2, and starts termination at that boundary point.

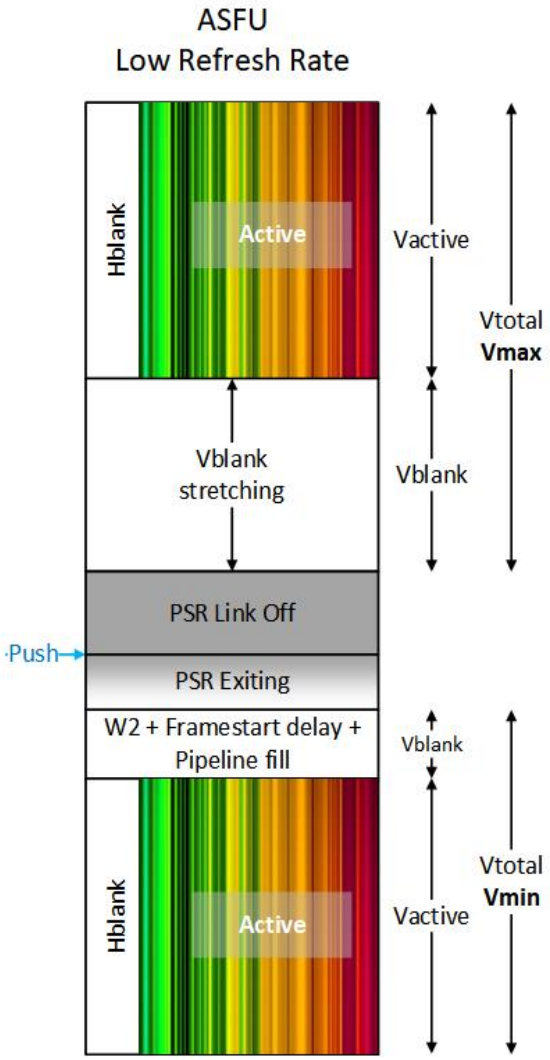


Double-Buffer Behavior

The push mode causes VRR to affect all pipe and plane double-buffered registers. As the vblank sent to the panel (timing generator vblank or undelayed vblank) stretches it delays the assertion of vblank sent to the pipe (delayed vblank), which is used as the double-buffer update for all of the pipe and planes. When VRR begins to terminate the vblank, delay window 2 (W2) happens, then the delayed vblank asserts, triggering the double buffer update before the framestart and pipeline fill (i.e. VRR guardband). As a result, double-buffer registers will update after the push is set or when vmax decision boundary is reached.

ASFU

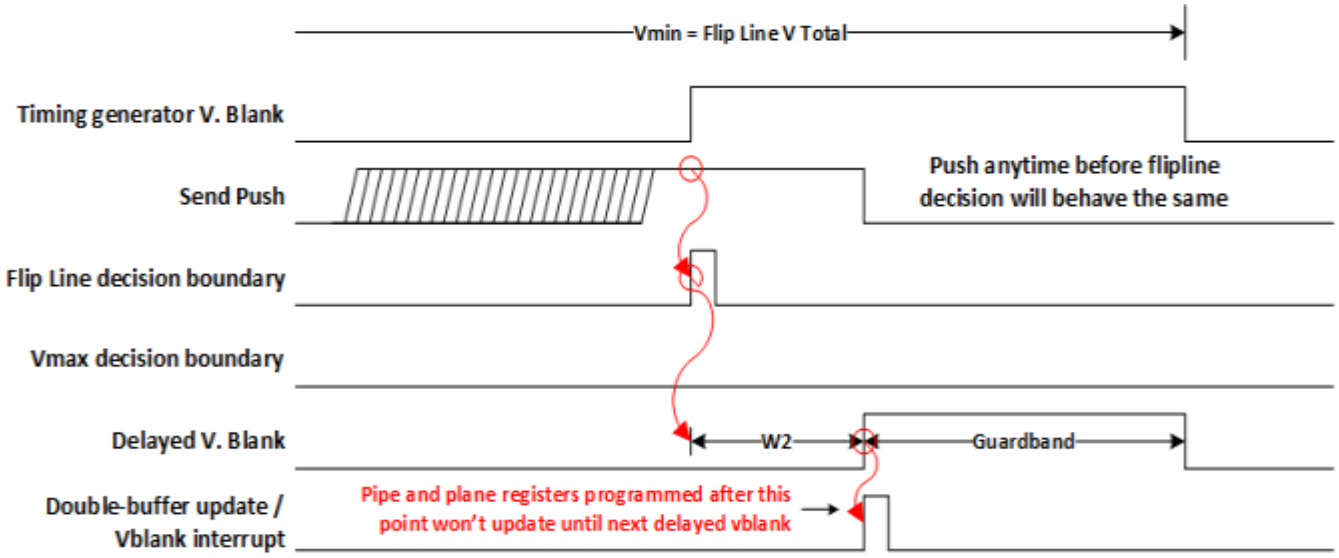
Adaptive Sync Frame Update (ASFU) changes the VRR behavior when max vertical is reached so that it enters PSR instead of terminating vblank and starting another frame. A later push then causes PSR exit with minimum vblank (flip line value ignored at PSR exit). The VRR behavior with push before Vmax decision boundary is unchanged by ASFU.



VRR Scenarios

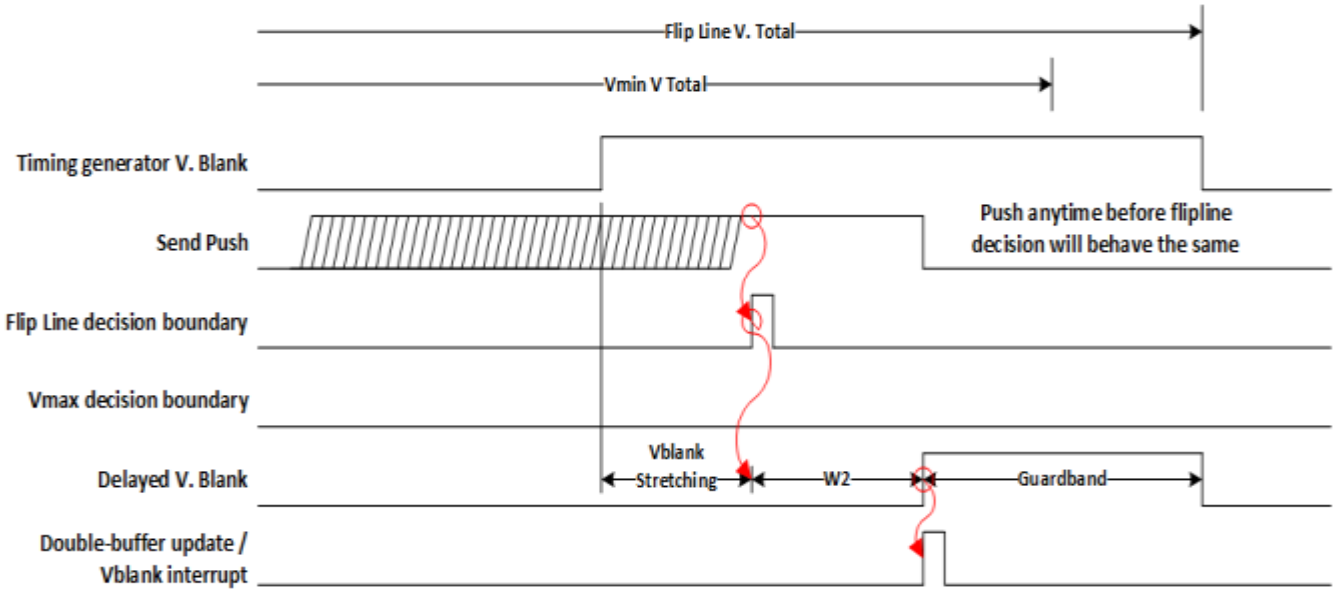
Minimum vblank (Vmin = Flip Line <= Vmax)

If push happens before flip line decision boundary and flip line value is Vmin, the vblank is minimized and the plane and pipe double-buffer registers update W2 lines after the flip line decision boundary (i.e., at the start of delayed vblank).



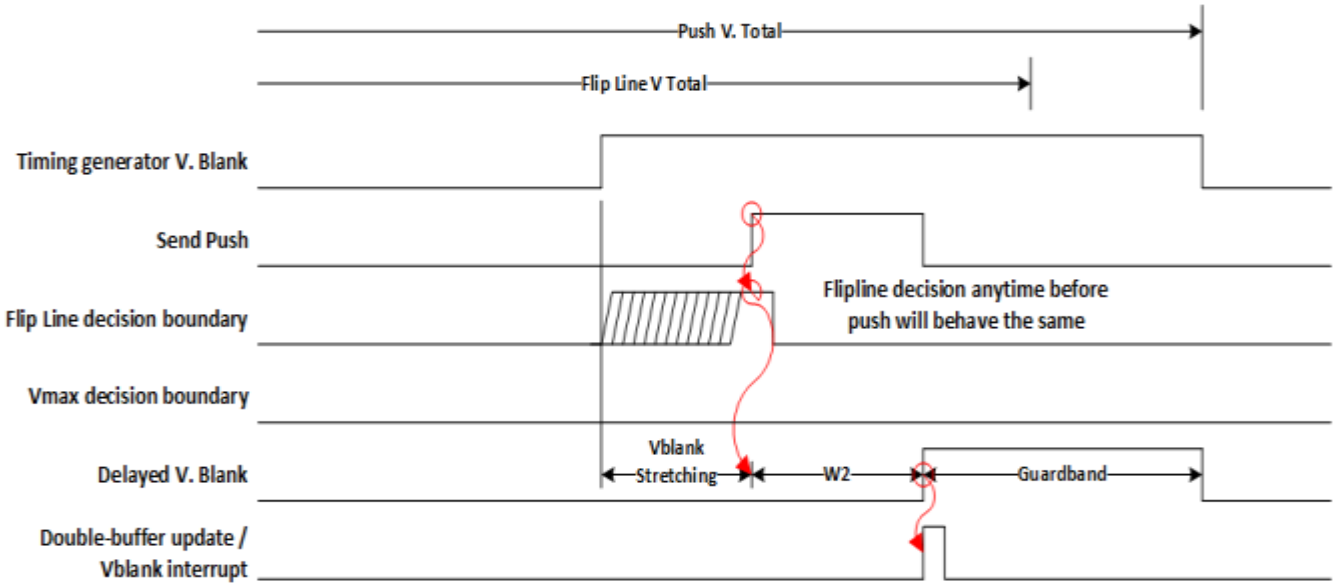
Medium vblank terminated at flip line (Vmin < Flip Line <= Vmax)

If push happens before flip line decision boundary and flip line value is greater than Vmin and less than or equal to Vmax, the vblank will stretch and the plane and pipe double-buffer registers update W2 lines after the flip line decision boundary.



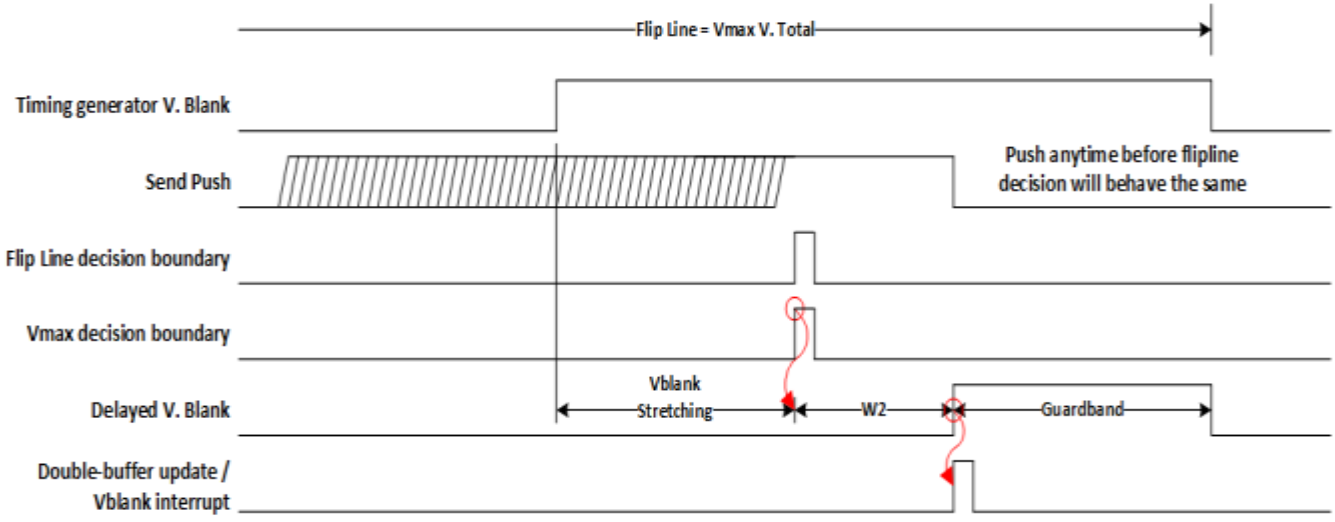
Medium vblank terminated at push (Vmin <= Flip Line < Vmax)

If push happens after flip line decision boundary and before Vmax decision boundary, the vblank will stretch and the plane and pipe double-buffer registers update W2 lines after the push.



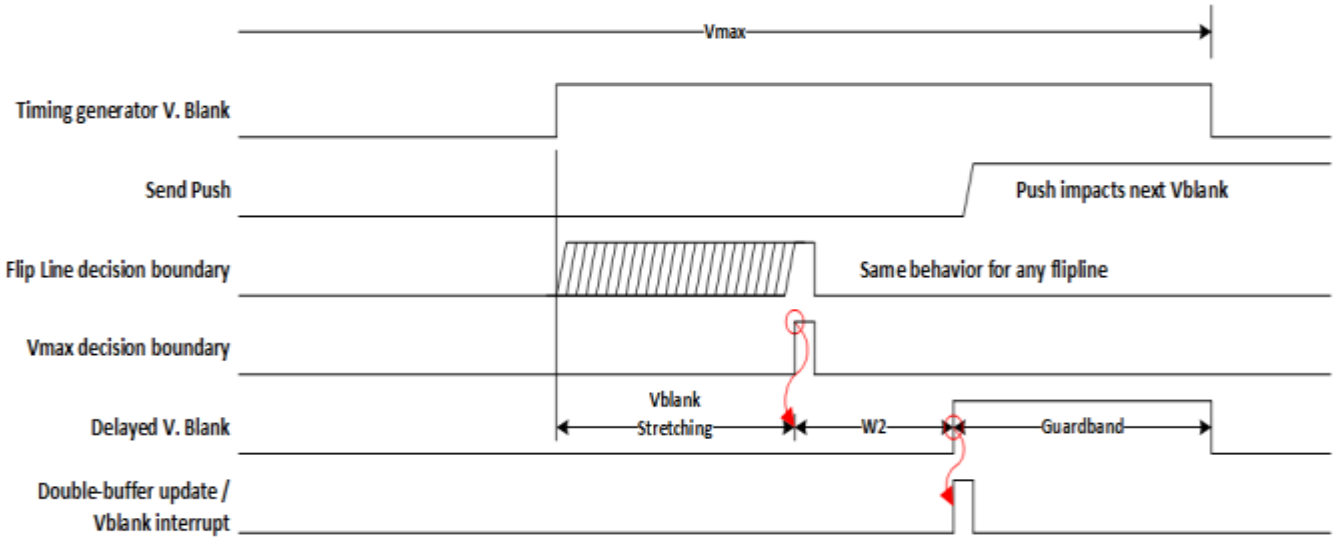
Maximum vblank terminated at flip line (Vmin < Flip Line = Vmax)

If push happens before flip line decision boundary and flip line value is equal to Vmax, the vblank will stretch to the max and the plane and pipe double-buffer registers update W2 lines after Vmax / flip line decision boundary.



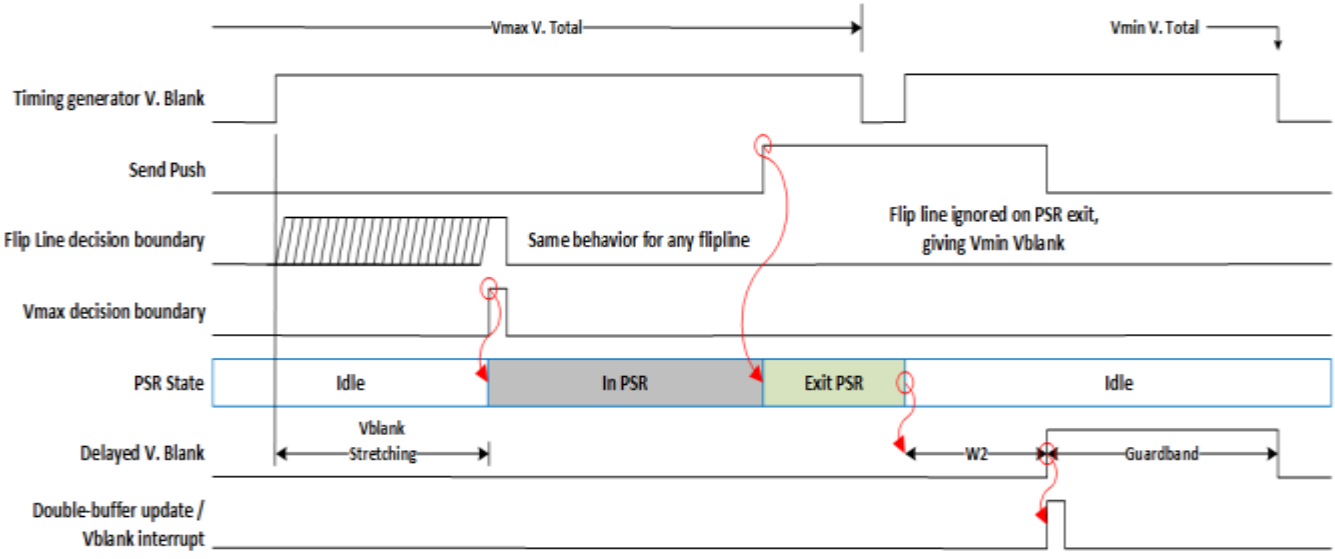
Maximum vblank terminated at Vmax

If push happens after Vmax decision boundary, the vblank will stretch and the plane and pipe double-buffer registers update W2 lines after the Vmax decision boundary.



Maximum vblank enters PSR

If PSR is enabled (for ASFU) and push happens after Vmax decision boundary, the vblank will stretch and enter PSR after the Vmax decision boundary. The push then causes PSR exit with minimum vblank (flip line ignored at PSR exit) and the plane and pipe double-buffer registers update W2 lines after the timing generator vblank.



Programming Sequence

1. Enable VRR
    - This can be done after mode set (called VRR Enable Sequence 1), or during mode set in the step that configures transcoder timings and other pipe and transcoder settings so that VRR is enabled from the first frame (called VRR Enable Sequence 2).

    a) Configure VRR timing and control registers
        - TRANS_VRR_CTL VRR Guardband programmed.
        - TRANS_PUSH Push Enable must be set.
        - TRANS_VRR_CTL Flip Line Enable must be set.

    b) For eDP/DP adaptive sync SDP use case
        - Program TRANS_VRR_VSYNC
        - Set "Adaptive Sync SDP Enable" in VIDEO_DIP_CTL register

    c) Set TRANS_VRR_CTL VRR Enable

    d) If ASFU, enable PSR - See Panel Self Refresh chapter

2. Screen updates, repeat as needed for each update

    a) Program the double-buffer registers that need updating
        - There will be a double-buffer update W2 lines after the Vmax decision boundary if ASFU is not used, so care must be taken to ensure multiple resource programming does not straddle the double-buffer update point and cause non-atomic updates.
            - Options
                - Use ASFU so that PSR is entered at Vmax.
                - Align programming to happen before the Vmax decision boundary + W2 lines.
                - Use double buffer disable/stall mechanisms to stall double buffering until all programming is complete.

    b) Program Flip Line value in TRANS_VRR_FLIPLINE (this can be programmed multiple times before the next step).
        - Flip Line value cannot be changed between when the push is initiated, and the push is done.

    c) Set TRANS_PUSH Send Push

    d) Hardware will clear the Send Push when the double buffer update happens (i.e., the start of delayed vblank)

    e) Poll for TRANS_PUSH[Send Push] cleared

3. Disable VRR

    a) If ASFU, disable PSR - See Panel Self Refresh chapter

    b) Clear TRANS_VRR_CTL VRR Enable

    c) Stop setting push

- Push is not needed when VRR is disabled because frames will terminate automatically.
- Setting push after VRR is disabled will cause the push to be held until later VRR enabling. That is not validated and not supported.

d) Poll for VRR live status indicating VRR has disabled.
- VRR live status takes one frame to change after VRR mode is disabled.

4. VRR double buffer update interrupt will remain active until VRR live status is de-asserted.
   a) TRANS_PUSH Push Enable can be cleared at this point or later
   b) VRR can be re-enabled by returning to step 1.
   c) If port will be disabled, continue to mode set disable sequence.
      - Hardware may be capable of transcoder disable with VRR enabled, but that is not validated and not supported.

VRR with Port Sync Mode

Port sync has multiple transcoders running synced together. The primary transcoder will send sync signals to the secondary transcoders.

Program the VRR registers the same on both transcoders when enabling VRR.

> VRR_CTL VRR Enable and TRANS_PUSH Push Enable must be set on primary and secondary transcoders. The other registers may not all be necessary for the secondary transcoders but programming them differently is not validated and not supported.

When sending screen updates, only the primary transcoder VRR needs to be updated. Only set push on the primary transcoder and only update the flip line value on the primary transcoder.

Starting VRR from Nominal Refresh Rate

1. Driver determines the panel timings and the highest, lowest, and nominal refresh rates to support
2. Driver does mode set to the highest pixel rate and highest refresh rate timings (should be close to CVT1.2 RB), except it extends the vertical total so that the resulting refresh rate is nominal.
3. Driver enables VRR with Vmax for the lowest refresh rate and Vmin for the highest refresh rate. Vmin = the vertical total from the mode set before extending to nominal refresh rate.
4. Hardware will vary refresh rate between the Vmax and Vmin based on the timing of pushes.

- When VRR is disabled the refresh rate will return to Vtotal (nominal).
- Steps 2 and 3 may be separated (VRR Enable Sequence 1) or combined so that VRR is enabled during the mode set (VRR Enable Sequence 2).
- The pixel rate and horizontal timings are programmed to match the highest refresh rate and do not change.

Starting VRR from Maximum Refresh Rate

1. Driver determines the panel timings and the highest, lowest, and nominal refresh rates to support.
2. Driver does mode set to the highest pixel rate and highest refresh rate timings (should be close to CVT1.2 RB).
3. Driver enables VRR with Vmax for the lowest refresh rate and Vmin for the highest refresh rate. Vmin = vertical total from the mode set.
4. Hardware will vary refresh rate between the Vmax and Vmin based on the timing of pushes.

- When VRR is disabled the refresh rate will return to Vtotal (same as Vmin).
- Steps 2 and 3 may be separated (VRR Enable Sequence 1) or combined so that VRR is enabled during the mode set (VRR Enable Sequence 2).
- The pixel rate and horizontal timings are programmed to match the highest refresh rate and do not change.

Display Port Configuration Data (DPCD)

Video timing information in the DP MSA (mainstream attribute packet) data is designed for video modes where the parameters are static. For modes such as VRR that dynamically change the video timing, the mainstream attribute fields cannot be used. Therefore, panel DPCD registers need to be appropriately programmed for VRR use cases.

Restrictions

VRR Restrictions
• PSR2 is incompatible with VRR. • Interlaced mode is incompatible with VRR. • DRRS is incompatible with VRR. • Stereo 3D is incompatible with VRR. • Vmin >= Vactive + Window 2 + VRR Guardband • After push, software must wait for flip done before starting another push. • PSR is supported with VRR for the Adaptive Sync Frame Update (ASFU). • Flip Line must always be enabled when VRR is enabled. • Flip Line value cannot be changed between when the push is initiated and the push is done. • Flip Line value must be less than or equal to Vmax (Flip Line <= Vmax) and greater than or equal to Vmin (Vmin <= Flip Line) Vmin <= Flip Line <= Vmax • A fixed refresh rate is when Flip Line = Vmax • When CMTG is enabled with VRR: o Must be running at a fixed refresh rate o Changing the refresh rate must be done within the V. Active region

Transcoder Control

Control
TRANS_CONF

Transcoder LRR

LRR (Low refresh rate) is mainly about dynamically reducing panel refresh rate to lowest possible refresh rate or to media multiple refresh rate (e.g., 48Hz) for idle and media playback scenarios. LRR provides predictable VBI unlike flip-based refresh rate. LRR feature relies on software programmed Vtotal functionality and can be enabled on VRR capable panels.

Entry Sequence

1. If VRR is enabled, driver disables VRR.
2. Driver updates Vtotal and Vblank-end to any refresh rate supported by the panel.

Exit Sequence

1. Driver updates Vtotal and Vblank-end to nominal refresh rate.

Here is an example scenario with idle and LRR.

1. Configure eDP/DP at 60 Hz.
2. LRR idle entry: Update Vtotal and Vblank-end for 40 Hz
3. LRR idle exit: Update Vtotal and Vblank-end for 60 Hz
4. Enable VRR and program flips
5. Disable VRR
6. LRR media playback entry: Program flip and then update Vtotal and Vblank-end for 48 Hz
7. LRR media playback exit: Update Vtotal and Vblank-end for 60 Hz

LRR Restrictions

- After push, software must wait for push done before starting another push.
- Global Double Buffer Disable functionality is not supported with LRR.
- No asynchronous flips are allowed when LRR is enabled.
- Interlaced mode, DRRS, Stereo 3D, and PSR2 are incompatible with LRR.
- Since panels may have a restriction on the maximum change in refresh rate between two consecutive frames, disabling LRR (TRANS_VRR_CTL[31]==0) does not immediately bring the current refresh rate to the nominal refresh rate.
- When using the transcoder port sync mode, LRR must use Pipeline Full Override Programmed Pipeline Full Line Count setting in TRANS_VRR_CTL.
- PSR is incompatible with LRR.

Transcoder DP2

TRANS_DP2_CTL

TRANS_DP2_VFREQLOW

TRANS_DP2_VFREQHIGH

Modeset

Default behavior upon reset or disconnect for the source device and sink device is to come up with 8b/10b encoding.

A modeset is needed on source side to switch to DP2.0 128b/132b channel coding.

Link Rates

Refer to the project overview page.

Link Training

FEC parity symbols are not transmitted during link training, regardless of whether the DP Source device has set or cleared the DP Sink device's FEC_READY bit.

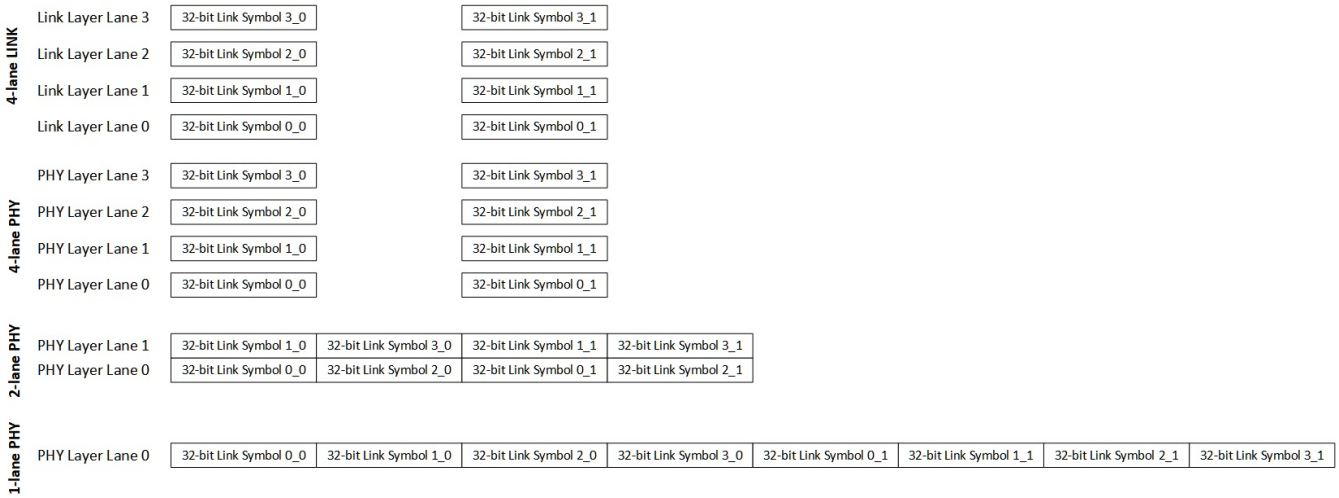
Scrambling and Pre-coding are disabled during Link Training.

DPTX may change the channel coding between 128b/132b channel coding and 8b/10b channel coding during Link Training.

Symbol Mapping and interlane skew

Link Layer always performs stream data to Link Symbols mapping for 4-lane Main Link. It is PHY Logical Sub-layer that converts the lane count to physical lane count.

Unlike DPTX PHY Logical Sub-layer for 8b/10b channel coding, DPTX PHY Logical Sub-layer for 128b/132b channel coding shall not insert any inter-lane Link Symbol cycles skew.



Channel Coding

RS (198,194) with a symbol size of 8-bits is used for Forward Error Correction when 128/132b coding is enabled.

PBN/VC Payload Calculation

The PBN (Payload Bandwidth Number) is an integer number calculated by DisplayPort Source device representing a peak bandwidth of a stream to be transported in a VC Payload. The PBN value has the unit of 54/64MBps. Actual PBN Indicates the PBN needed to drive the pixel clock and bits_per_pixel, considering overhead of 0.6% (accounting for deviation of the link rate because of down-spread). It is independent of the number of VC Slots.

One PBN is equal to 54/64 MBps.

EOC overhead: DSC 2.0 applies 4 link symbols per DSC slice

Actual PBN Calculation

Note that the actual bits_per_pixel depends on the pixel format and whether compression is enabled or not.

Bandwidth needed (in MBps) = (PixelClock * bits_per_pixel)/ 8

Bandwidth needed (in PBN) = ((PixelClock * bits_per_pixel)/8)/(54/64)

= (PixelClock * bits_per_pixel * 8)/54

ActualPBN = CEIL(((PixelClock * bits_per_pixel * 8)* 1.006)/54)

PBN Per TimeSlot Per MTP Calculation

For 128b/132b channel coding, data bandwidth efficiency is 96.71%

AvailableTotalBandwidth in Mbps = LinkRateMbps * MaxLaneCount * 0.9671

AvailableTotalBandwidth in MBps = (LinkRateMbps * MaxLaneCount * 0.9671) /8

AvailableTotalBandwidth in PBN = ((LinkRateMbps * MaxLaneCount * 0.9671) /8)/(54/64)

= (LinkRateMbps * MaxLaneCount * 0.9671 * 64) /(8 * 54)

There are 64 time slots in 1 MTP, so PBN Per TimeSlot/MTP = AvailableTotalBandwidth/64

PBN Per TimeSlot/MTP = (LinkRateMbps * MaxLaneCount * 0.9671 * 64) /(8 * 54 * 64)

= (LinkRateMbps * MaxLaneCount * 0.9671) /(8 * 54)

So, the Number Of Time Slots (VC Payload Size) actually needed for a given virtual channel

= CEIL(ActualPBN/(PBN Per TimeSlot/MTP))

Example

Consider 4-lane UHBR20 port and 8k@60 RB2 pixel stream.

Actual PBN = CEIL((2068.660 * 24 * 8 * 1.006)/54)

PBN Per Time Slot / MTP = 179.09 ... this is also given in table 2-150 (128b/132b Link Layer Per-time-slot PBN Values) of DP v2.0 standard.

VC Payload Size = CEIL(7400/179.09) = 42

The payload size calculation done using (DataM/DataN) below should be less than or equal to the above "payload size" result.

= CEIL((DataM/DataN) * 64)

Refer to transcoder section for (DataM/DataN) calculation for DP v2.0.

PHY logical Frame

Total number of Link Symbol bits transmitted per PHY Logical Frame per lane is as follows:

(12 * 128-bit Super Symbol)/RS blocks * 8 RS blocks/PHY Logical Frame - 32 bit PHY Sync Symbol/PHY Logical Frame

= 12,256 bits

Total number of bits transmitted per PHY Logical Frame per lane is as follows:

(12 of 129-bit codes + 4 padded bits + 32 bits of RS Parity Symbol) / RS block * 8 RS blocks/PHY Logical Frame

= 12,672 bits per PHY Logical Frame per lane

DP2.0 Link Quality Checks

The PHY supports generation of test patterns for measuring the link quality at UHBR link rates.

1. PRBS7, PRBS9, PRBS11, PRBS15, PRBS23 and PRBS31 bit patterns.
2. Square patterns up to twenty 1s and twenty 0s.

High level Programming Sequence

1. Software does normal bring up of the port in DP2 mode and completes link training.
2. Software follows Test Pattern Register Write Sequence below to write the PHY register (definition below) to set the desired test pattern.
3. Software optionally follows Test Pattern Register Read Sequence below to read back the PHY registers for verification or read/modify/write.
4. Software repeats as necessary to walk through patterns.
5. To return to normal output, software follows Test Pattern Register Write Sequence to write the PHY registers to mode 0.

PHY Register Definition

LANEN_DIG_TX_LBERT_CTL

There is an instance of the register for each PHY and each lane. Write and read access is not basic MMIO.

Test Pattern Register Write

The PHY registers cannot be written directly by MMIO. The writes must be bridged to the PHY by GSC.

Write access uses a Port ID and Address. The address broadcasts the write to all lanes in the PHY, updating them simultaneously. No single lane writes are supported.

Write Address Table

DDI	Port ID	Write Address* Broadcast write to all lanes in PHY
A	0x3A	0x0002_40C8
B	0x3C	0x0002_40C8
C	0x3D	0x0002_40C8
D	0x3E	0x0002_40C8
TC1	0x49	0x000A_40C8

*Absolute address

Test Pattern Register Write Sequence

This sequence is used to write the test pattern register to set a pattern or disable the patterns and read to normal output.

1. SW uses GSC interface to request GSC write to the PHY register.
    - The details of the interface are outside of this part of the spec.
    - Use Port ID and Write Address for selected PHY from Write Address Table.
    - Data is 16-bits to write into PHY register.
2. If the write request is composed correctly, GSC passes the write to the PHY and acknowledge the request.
    - Error handling timeout and retry policy is outside of this part of the spec.

Common Test Pattern Settings

These are the values to program in the test pattern register for the commonly required patterns.

Desired Pattern	PAT0 (hex)	Mode (hex)
Disabled	Don't care	0
PRBS31	Don't care	1
PRBS15	Don't care	5
SQ20 (20 1s and 20 0s)	000	B
SQ2 (alternating 1s and 0s)	2AA	9

Test Pattern Register Read

Read access uses MMIO through a window configured with an index register. The reads are to each lane individually.

Read Address Table

			Read Address*			
DDI	Index Register Address*	Index Data	Lane 0	Lane 1	Lane 2	Lane 3
A	0x28_0014	0x00	0x33_9064	0x33_9264	0x33_9464	0x33_9664
B	0x28_0014	0x01	0x33_9064	0x33_9264	0x33_9464	0x33_9664
C	0x28_0014	0x02	0x33_9064	0x33_9264	0x33_9464	0x33_9664
D	0x28_0014	0x03	0x33_9064	0x33_9264	0x33_9464	0x33_9664
TC1	0x28_001C	0x02	0x37_9032	0x37_9132	0x37_9232	0x37_9332

*Graphics MMIO offset

Test Pattern Register Read Sequence

This sequence is used to read the test pattern register as needed for read/modify/write or to verify the writes.

1. SW uses MMIO to write the index to select the PHY
    - Address = Index Register Address from Read Address Table.
    - Data = Index Data from Read Address Table.
2. SW uses MMIO to read the PHY registers per-lane for the selected PHY
    - Use 2-byte MMIO read.
    - Address = Read Address from Read Address Table.

Transcoder Panel Replay

Panel Replay feature supported over DP ports is very much like PSR described in eDP 1.4a, but the only option supported is Main Link-ON with the source transmitting rate governed dummy data framed in Control Symbols. In Panel Replay Mode, a DP Source device may only send the update region(s) instead of sending a complete frame every frame time. The DP Sink device discards the dummy data and renders frames based on pixel date stored in its local frame buffer according to the DP Source's video timing. Panel replay is transparent to the physical layer and the connector. This feature is supported on all ports. It is enabled in DP ALT, DP native and TBT tunneling modes. Panel Replay can be enabled at any time independent of modeset.

Selective fetch and update not supported with Panel Replay

Keypoints

- Source will need to query the sink DPCD registers to discover the Panel Replay Capability of the Sink and DP v2.0 support.
- Panel Replay feature works with Adaptive sync. For frames that are being sent to the panel and for the replay frames, the link is up and source will be sending embedded timings. Sink is expected to maintain timing sync with the source.
- VSC packet format and requirements are different from PSR. A VSC SDP is sent to indicate PSR entry and exit. VSC SDP is also sent to indicate the co-ordinates of the update region. The set-up time for the VSC packets are the same as PSR. A packet indicating the end of SU region is not needed.
- Display controller fills in the TUs/MTPs with "dummy data" when there is no update to send. Dummy data may be 0's in SST mode and SF control symbol in MST mode.
- Panel Replay works with 8b10b encoding as well as 128b/132b encoding but panels must be DP v2.0 compliant.
- Panel Replay works with SST and MST encoding.
- Each tile of a tiled display is independent from Panel Replay standpoint.
- S3D and interlace modes are not supported.

Comparison of DP Panel Replay and eDP Self-Refresh modes

Feature	eDP self-refresh	Panel Replay
availability	internal panels only	external panels only
stream compression	region aligned to DSC slice boundary	region aligned to DSC slice boundary
Link On mode	not supported	supported
Tracking mode	manual only	manual only
idle frame status indication	VRR: push no VRR: flip	VRR: push no VRR: flip

Feature	eDP self-refresh	Panel Replay
repeat frames	frame buffer not read framestart masked	frame buffer not read framestart masked
CRC	full frame CRC for full frames partial CRC for regions	full frame CRC for full frames partial CRC for regions
HDCP	not enabled	enabled

Additional programming considerations (repurposed eDP registers)

1. mask register: Only PSR_MASK[Mask FBC modify] and PSR_MASK[Mask Hotplug] are used in panel replay mode.
2. Status register: Only SRD_STATUS[SRD state] field is used in panel replay mode.

Transcoder Timing

Transcoder
TRANS_HTOTAL
TRANS_HBLANK
TRANS_HSYNC
TRANS_VTOTAL
TRANS_VBLANK
TRANS_VSYNC
TRANS_VSYNCSHIFT
TRANS_MULT

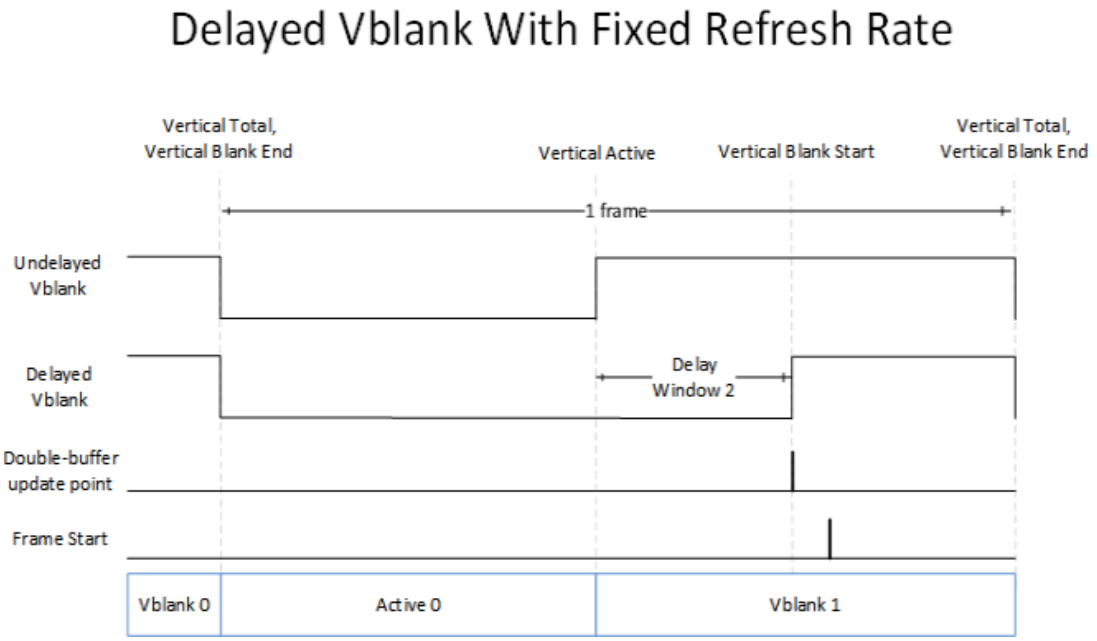
The transcoder timing generators create two versions of vertical blank (vblank) that are sent to the display pipes. The undelayed vblank asserts at the end of vertical active (TRANS_VTOTAL Vertical Active). The delayed vblank asserts at the vertical blank start (TRANS_VBLANK Vertical Blank Start) and must be programmed greater than or equal to the end of vertical active. Both de-assert at the end of vblank (TRANS_VBLANK Vertical Blank End), which must be programmed to match the vertical total (TRANS_VTOTAL Vertical Total).

The rising edge of the delayed vblank is the double-buffer register update point. The frame start signal asserts after that point, then pixel processing begins for the next frame.

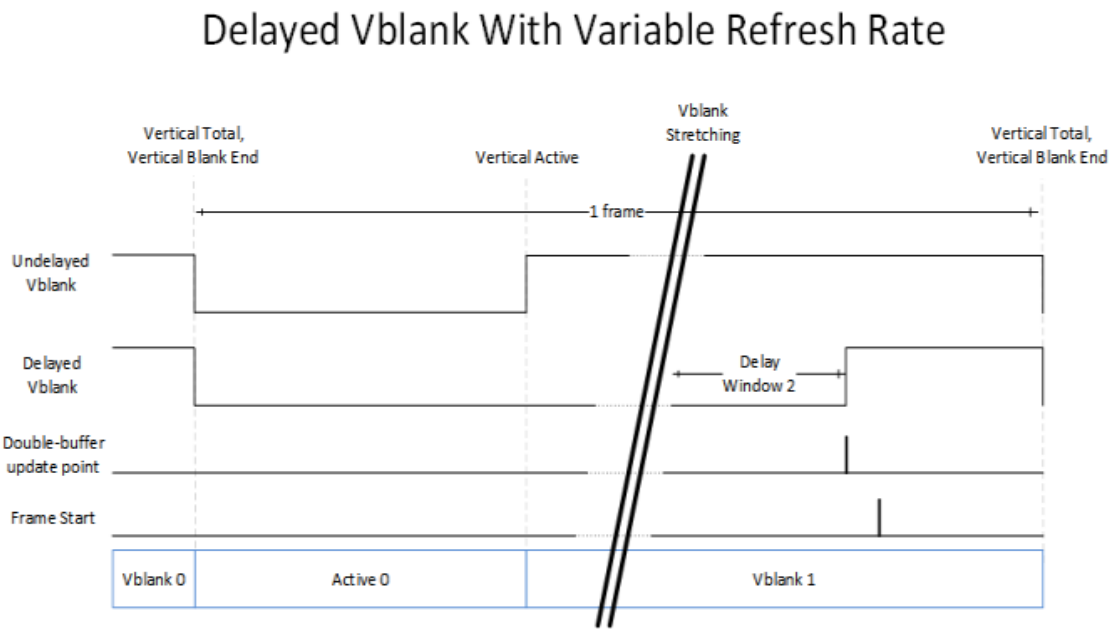
TRANS_VBLANK Vertical Blank Start can be programmed greater than TRANS_VTOTAL Vertical Active to create a window between start of undelayed vblank and delayed vblank where pixels are not being processed (called window 2) and programming can safely update double-buffered and non-double-buffered registers.

Window 2 size = TRANS_VBLANK Vertical Blank Start - TRANS_VTOTAL Vertical Active

Window 2 is typically only configured to be greater than zero when DSB will be used to program display registers. Without DSB, driver MMIO programming usually is too slow to fit in window 2, and instead must update only the double-buffered registers and use capabilities like the double buffer update disable to ensure atomic updates, and then there is no need to delay the vblank. See the DSB Engine Programming for window 2 size requirements.



When variable refresh rate is used, the vblank stretches while waiting for the indication to complete this vblank and start the next frame. The stretching happens before window 2.



Transcoder MN Values

There is one instance of these registers per each transcoder.

For dynamic switching between multiple refresh rates, M/N values may be reprogrammed on the fly. The link N should be programmed last to trigger the update of all the data and link M and N registers and then the new M/N values will be used in the next frame that is output.

DATAM

DATAN

LINKM

LINKN

Clocks:

ls_clk is the link symbol clock. i.e., 270 MHz for HBR.

strm_clk is the stream clock, which is the video pixel rate or dot clock.

cdclk is the core display clock.

The link only operates in Synchronous Clock mode.

The link clock and stream clock are synchronous, and the link M and N values stay constant for a given pixel rate.

Calculation of TU:

TU is the Transfer Unit used in SST.

TU size = 64 (recommended)

Calculation of Data M, and Data N:

Active/TU Size = Payload/Capacity = Data M/N

Note that for 4:2:0 format the number of bytes per pixel will be half the number of bytes of RGB 4:4:4 pixel.

Note that for 4:2:2 format the number of bytes per pixel will be 2/3 of the number of bytes of RGB 4:4:4 pixel.

Programming Note	
Context:	Low Bandwidth SST over High BW link
Active/TU size = Data M/N Select link rate such that there is at least 1 active symbol in every TU for any given dot clock. An example where this check fails is 720p with YUV 4:2:0 pixel format over HBR3 channel: Active/64 = (27 * 1.5) / (810 * 4) results in Active < 1.	

Compression Ratio (CR) = DisplayPort Compression enabled ? min(ratio1,ratio2) : 1

- ratio1 = Compressor BW / Link BW = (cdclk * bytes per pixel) / (ls_clk * number of lanes)
- ratio2 = (Horizontal active in pixels * bytes per pixel / 4) / ((Horizontal active in pixels * bytes per pixel / 8) +2)

Data M/N = (strm_clk * bytes per pixel) / (CR * ls_clk * number of lanes)

Note1: Data M/N remains unchanged between CoG and non-CoG because the reduction in the number of lanes is compensated by the reduction in stream clock.

Note2: For the CoG case above, the actual stream clock on pipe side = ($strm\_clk_{CoG}$ * number of segments).

Note3: $strm\_clk_{CoG}$ is calculated considering a single segment.

Calculation for DP v2.0 with 128b/132b channel coding is as follows.

Data M/N = (((strm_clk * bytes per pixel/CR) + (VActive * RR * 16 * number of slices))/(ls_clk * number of lanes)) * (1/4) * (1/0.9671)

where:

strm_clk is stream clock in MHz

ls_clk is link rate divided by 32. Example: 10G/32 = 312.5 MHz

RR is the refresh rate in MHz

number of slices refers to DSC number of slices per scanline

CR is the compression ratio

RGB444: CR = bits_per_component *3 / bits_per_pixel

DP 2.0 data bandwidth efficiency is 96.71%

(16 * number of slices) is the number of EOC bytes over all slices.

Note: MST EOC is 4 link symbols.

Note: EOC overhead changes based on DSC slice configuration.

Calculation of Link M and Link N:

Link M/N = strm_clk / ls_clk - for non-CoG use cases

Link M/N = $strm\_clk_{CoG}$ / ls_clk - for CoG use cases

Recommendation: In Link M/N calculation, M should be rounded up and N should be rounded down.

Restriction on clocks and number of lanes:

Number of lanes >= INT(strm_clk * bytes per pixel / ls_clk)

Restrictions on the Virtual Channel (VC) payload size in DisplayPort MST mode

- In a x1 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 4.
- In a x2 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 2.

- In a x4 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 1.

Transcoder MSO Operation

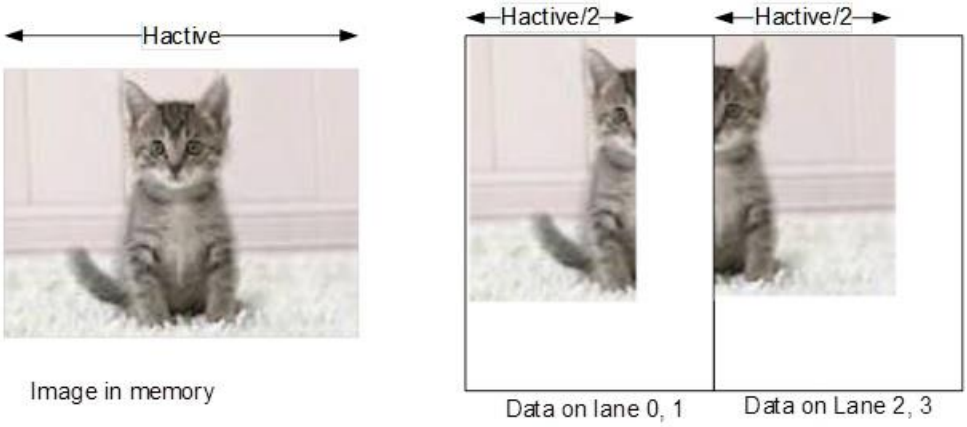
Related Registers

Register
PIPE_DSS_CTL1 Refer to the DSC section for programming.

Description

Multi-SST Operation (MSO) is defined in the eDP standard. It's a mechanism to simultaneously transmit multiple and separate SST Links over separate Main-Link lanes on any given port. This allows one Source device to directly connect to multiple Panel Segments within a single display module, each of which have a separate eDP Main-Link receiver connected to each Panel Segment. The panel will still have one connection for AUX CH and one connection for HPD. Link training will follow the eDP link training algorithm for each segment. Hence, DSS unit must be configured to CoG mode of operation prior to enabling DDI and subsequent link training.

2x2 CoG Example



MSO links use the same EDID timing for:

- Active number of lines and columns per Panel Segment
- Horizontal and vertical blanking per Panel Segment
- Horizontal and vertical front porch offset per Panel Segment
- Number of overlap pixels when this option is enabled

eDP MSO/CoG Configurations

Configuration	Lanes Enabled/Trained	Notation
Two SST links with 1 lane per link	0, 1	2x1 CoG
Two SST links with 2 lanes per link	0, 1, 2, 3	2x2 CoG

eDP MSO Support at Pipe/Transcoder Level

Pipes
A only

Note: MSO segmentation beyond pipe is not allowed.

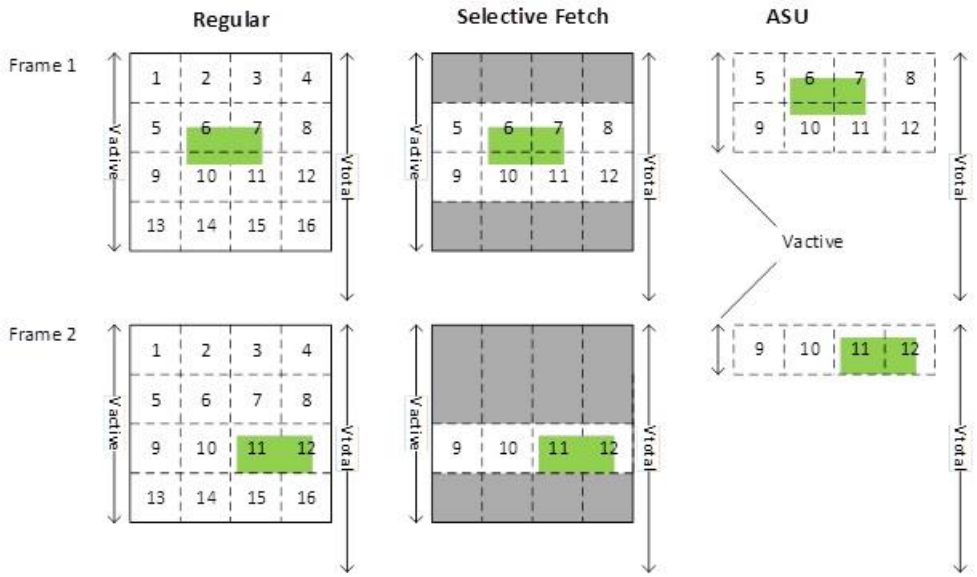
Key Design Aspects

- No requirement for MSO to work with YUV pixel formats.
- Link Training: Link training for a MSO configuration will remain unchanged from non-MSO. All the lanes are trained as a single link.
- DRRS: Timing on all tiles will stay in sync from the source side during DRRS. All tiles will see an updated MSA with the new M and N timings on the link at the same time.
- ASSR: All tiles must be configured to use alternate scrambler seed reset. Independent enable/disable of ASSR on each link will not be allowed.
- GTC: No change in the GTC lock and acquisitions phase. GTC will switch from 10ms to once-a-frame when both tiles go into PSR2 SU deep sleep state.
- PSR: All tiles will go in and out of sleep state synchronously. Timing on all the tiles will stay in sync through entry and exit from PSR.
- PSR2: Blocks on all tiles will be simultaneously updated, even if only one tile actually had an update.
- ALPM: Since the timings are in sync for all tiles, Frame syncs will be common for all the tiles.
- Fast_Wake from PSR2: Fast wake sent over AUX will wake up all links concurrently.
- Compression: Identical PPS syntax will be used across all MSO segments.
- Selective enabling of MSO segments is not supported.
- Per segment MSA and SDP are sent
    - on lanes 0,1 and repeated on lanes 2,3 in 2x2 configuration
    - on lane 0 and on lane 1 in a 2x1 configuration
- Software programs individual segment timings that are replicated for all other segments

Restriction
The number of horizontal active pixels including the overlap pixels should always be an even number.

Transcoder Asynchronous Selective Update

This feature extends ASFU to partial frame updates and adds the capability of updating one region of the display using the ASFU framework. Only one update region per frame is supported and the capability is only supported with manual tracking mode/selective fetch. A Metadata packet will transfer the co-ordinates of the update region as a proprietary secondary data packet, prior to sending the active data for the frame. All the framework available with ASFU will continue to apply to ASU. If compression is enabled, ASU mandates that the PPS sent to the panel be fixed and it should not change on the fly. Hence, DSS control registers once configured, should not be modified for partial updates. ASU does not support CoG use cases.



- Dashed lines represent DSC slice
- Solid line represents full frame size
- Greyed out area not fetched
- Green are is updated region
- Pipe scaling would require fetch of extra slices around the updated region slices in order to have the data to fill scaler filters before entering the updated slices.

ASU flows

1. Mode set enable.
2. PSR enable, wait for 1 Vblank.
3. If selective update, SW updates pipe source size, Vblank start, Vactive and slice_row per frame as per SU size.
4. Set TRANS_PUSH send PUSH.
5. SW to poll for push done bit.
6. HW will consume pipe source size and slice_row per frame @VBI and Vblank start and Vactive @Vtotal.
7. For subsequent selective updates, SW to poll on scanline interrupt, 1st line and go to step (3).

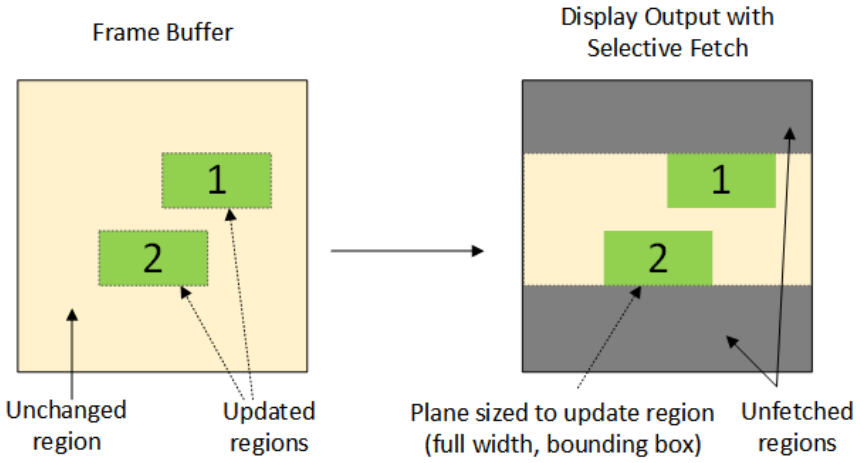
8. If PSR disable, wait for scanline interrupt, 1st line. Restore the original timing parameters.

The complete sequence is as follows.

1. Mode set enable
2. Push enable
3. DB Vactive enable
4. VRR enable
5. PSR enable, wait for 1 Vblank
6. Selective update
7. PUSH
8. Poll for PUSH done
9. SW to poll on scanline interrupt, 1st line for future selective updates
10. PSR disable
11. Restore original timing parameters
12. Program Vtotal to Vnominal.
13. VRR disable
14. DB Vactive disable
15. PUSH disable

Selective Fetch

Selective fetch is a power saving feature to reduce display engine use of memory bandwidth by only fetching (reading from memory) the updated regions of the frame buffer and sending those updated regions to a panel with a remote frame buffer and selective update capability. Software selects the amount of fetch by adjusting the size and position of the display planes to fit the updated regions in the frame, and configures panel selective update manual tracking to output that region to the panel. The result is that the bounding box around the updated regions is read from memory and delivered to the panel remote frame buffer and the other regions are not fetched and not sent to the panel, saving memory bandwidth, IO, and panel power.



Restrictions

- Supported with only full line width update region. Software must adjust the update region to full width.
- Supported with only a single update region per frame. Software must create the bounding box around all regions.
- Not supported with async flips. The plane size and position cannot be changed with async flips, so selective fetch cannot be used. Software must output a full frame for async flips.
- Not supported with command streamer flips. Command streamer flips do not update the selective fetch plane registers without adding in extra LOAD_REGISTER_IMMEDIATE commands, which are too complicated.
- Not supported with plane rotation. Software calculations to adjust update region coordinates are too complicated.
- Not supported with plane or pipe scaling. Software calculations to account for extra lines of scaler filter input and adjusted scale factor and filter phase are too complicated.
- Not fully supported with LACE/LDPST. The histogram accumulates across the frame and gives incorrect results when only part of the frame is updated. Software adjusts the histogram taking algorithm to get partial support.
- Not fully supported with DPST. The histogram accumulates across the frame and gives incorrect results when only part of the frame is updated. Software adjusts the histogram taking algorithm to get partial support.
- Not supported with PSR2.
- Supported with Panel Replay
- Not supported with DSC.

Hardware

Hardware support of selective fetch is through the panel selective update feature with manual tracking, the planes and cursor selective fetch registers, and the full frame update triggers.

Software

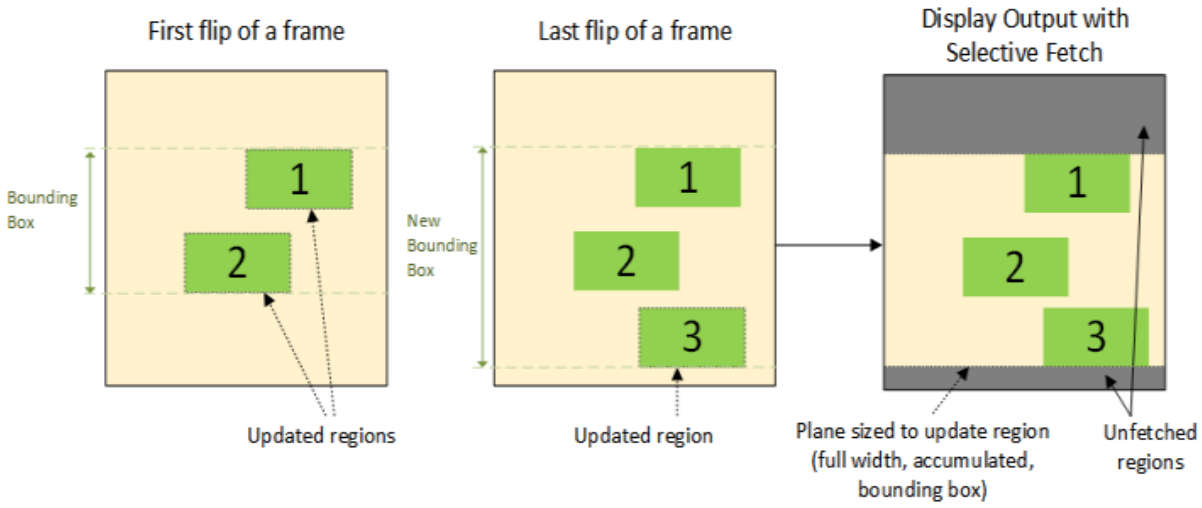
Software support of selective fetch

- Calculate the updated region
- Program the planes and cursor selective fetch registers to adjust the sizes, positions, and panning to fit the update region.
- Program the selective update manual tracking register to fit the update region.
- Program planes, cursor, and manual tracking together to atomically update.
- Trigger full frame updates when selective fetch cannot be used, such as when updating DPST image enhancement.

Update Region

Selective fetch requires software to know which areas of the screen are being updated. The update region coordinates are then used to adjust the planes and cursor to limit the fetch, and to notify the panel of which region has been selectively updated. The update region must be full width and just a single region per frame. Software must calculate the region as a bounding box to encompass all the updated sub-regions.

When there are multiple screen updates (flips) within a single frame, the update region can be different for each flip, but the final update region needs to be a bounding box that encompasses the update regions for all the flips in the frame. Hardware does not accumulate the update region across flips; it only uses the last programmed values. Software must either accumulate the update regions and program hardware with the accumulated result at each flip and reset accumulation at each frame or trigger a full frame update when multiple flips happen within a single frame.



If there is a flip where the update region is unknown (e.g., application doesn't support this kind of tracking) or there is update that impacts the entire frame (e.g., pipe color correction programming), the update region is the full frame.

Update Region Alignment

Some features require the update region to be aligned to boundaries, then software needs to expand the update region to meet the alignment, increasing the amount of fetch.

For PSR2 selective update, the frame is divided into blocks of four scan lines each. The update region must be expanded so it aligns to the 4-line groups of transcoder vertical active.

Adjusting Plane and Cursor Size and Position

The selective fetch registers are an extra set of registers for each plane and cursor where software programs the adjusted size, position, offset (panning), and control (just the enable) to be used when doing a selective fetch. The regular (non SEL_FETCH) plane registers are used for full frame updates. Both sets must be updated with each flip, so that any frame will appear correct whether it is a selective fetch

(uses the plane selective fetch registers) or a full frame fetch (uses the regular plane registers). Hardware will select between the sets of registers automatically. These registers are double-buffered and armed together with the regular plane and cursor registers.

SEL_FETCH_PLANE_CTL

PLANE_POS instances with SEL_FETCH prefix

PLANE_SIZE instances with SEL_FETCH prefix

PLANE_OFFSET instances with SEL_FETCH prefix

CUR_CTL instances with SEL_FETCH prefix

The plane selective fetch registers are programmed with the adjusted size and position of the planes, reduced down to the update region (must be the full width, accumulated, bounding box).

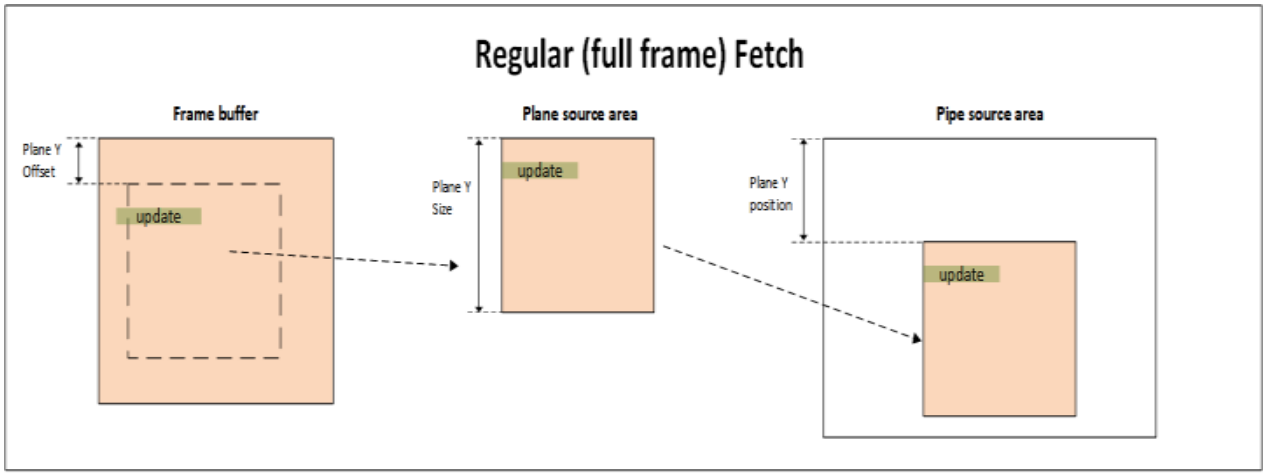
The plane adjustment calculations here are assuming the update region is relative to this plane's frame buffer. If the update is coming from the cursor or another plane, the position of cursor or other plane relative to this plane's frame buffer has to be calculated as the update region, or else the other plane and cursor changes can be simplified to do a full frame fetch if they are relatively infrequent. Other methods can be used to calculate using different starting coordinates.

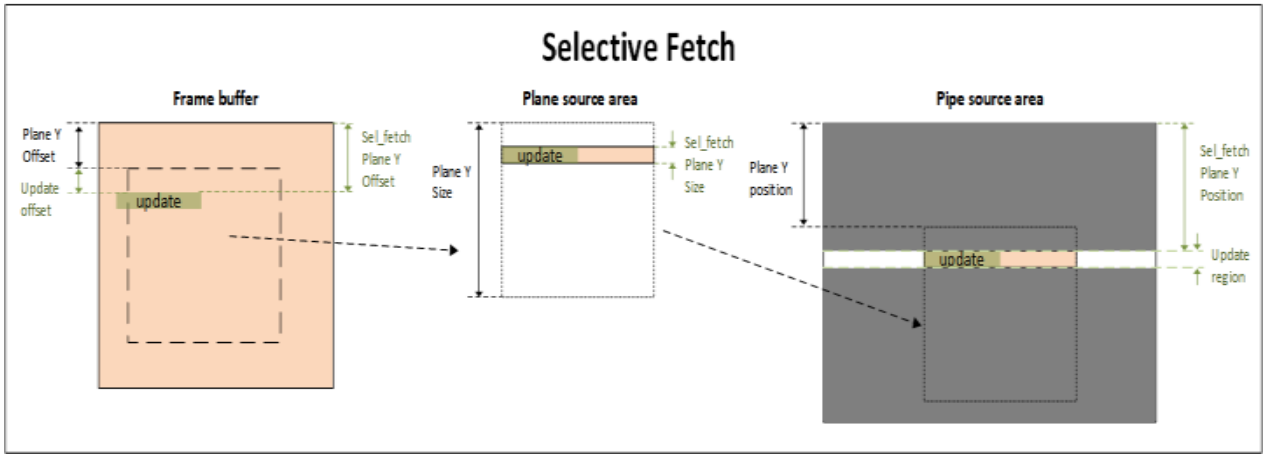
For each plane

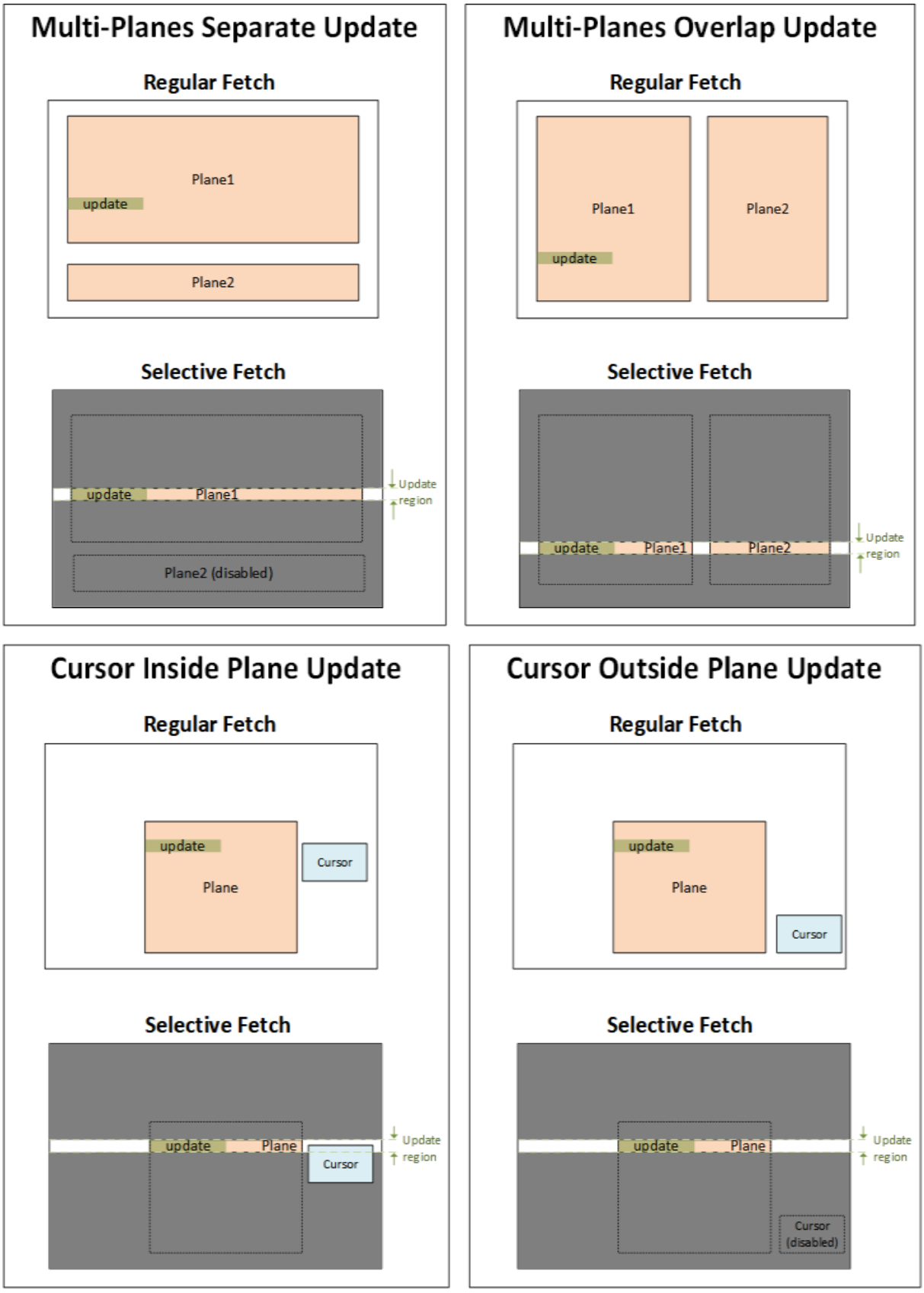
- SEL_FETCH_PLANE_OFFSET Start Y Position = first line of update region relative to start of frame buffer for this plane
- Update offset within this plane = update region vertical offset in this plane = SEL_FETCH_PLANE_OFFSET Start Y Position - PLANE_OFFSET Start Y Position
- SEL_FETCH_PLANE_SIZE Height = vertical size of update region within this plane
- SEL_FETCH_PLANE_POS Y Position = PLANE_POS Y Position + Update offset within this plane
- SEL_FETCH_PLANE_CTL Selective Fetch Plane Enable = If update region is within this plane ? Enable : Disable
    - This plane is disabled when the update region is fully outside of this plane.
- The plane width/X/horizontal values of the position, size, and offset are not changed with selective fetch, so program those fields the same in the normal plane registers and the selective fetch plane registers.

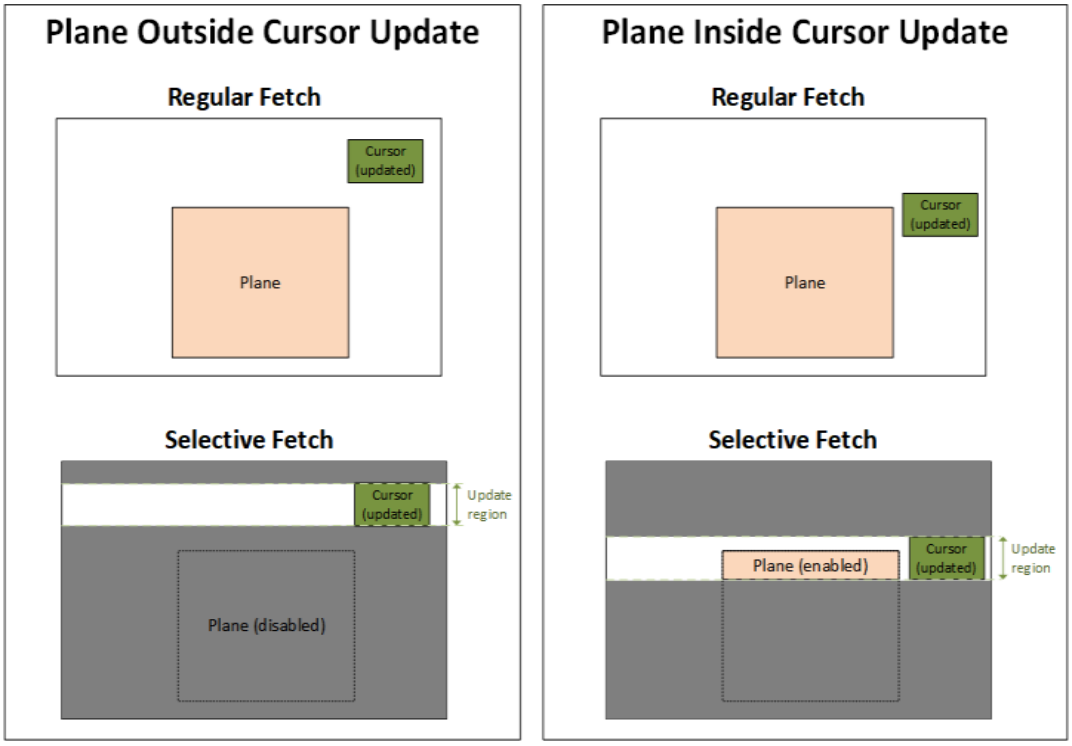
For cursor

- SEL_FETCH_CUR_CTL Cursor Mode Select = If update region, translated to pipe source coordinates, overlaps this cursor ? CUR_CTL Cursor Mode Select: Disable
    - Cursor size is not adjusted. Cursor is just disabled when the update region is fully outside of cursor.
    - Program the other fields in SEL_FETCH_CUR_CTL to match CUR_CTL.


Regular (full frame) Fetch
Frame buffer
Plane Y Offset
update
Plane source area
Plane Y Size
update
Pipe source area
Plane Y position
update


Selective Fetch
Frame buffer
Plane Y Offset
Update offset
update
Sel_fetch Plane Y Offset
Plane source area
Plane Y Size
update
Sel_fetch Plane Y Size
Pipe source area
Plane Y position
update
Sel_fetch Plane Y Position
Update region


Multi-Planes Separate Update
Regular Fetch
Plane1
update
Plane2
Selective Fetch
update
Plane1
Update region
Plane2 (disabled)
Multi-Planes Overlap Update
Regular Fetch
Plane1
update
Plane2
Selective Fetch
update
Plane1
Plane2
Update region
Cursor Inside Plane Update
Regular Fetch
update
Plane
Cursor
Selective Fetch
update
Plane
Cursor
Update region
Cursor Outside Plane Update
Regular Fetch
update
Plane
Cursor
Selective Fetch
update
Plane
Update region
Cursor (disabled)



Selective Update

The selective fetch programming must be combined with panel selective update (SU) programming so that only the selective fetch update region is sent to the panel.

Calculate the update region translated to transcoder vertical active coordinates, then program **PSR2_MAN_TRK_CTL** with that SU region.

- Depending on the project generation, the SU region address is programmed with blocks of either 4 line or 1 line granularity, and with different starting and ending behavior, as noted in the manual tracking register.
- PSR2_MAN_TRK_CTL SF Continuous Full Frame and SF Single Full Frame must be 0 for the SU region to be used.

Full Frame Updates

Sometimes software needs to send a full frame update, such as when async flips happen.

- Software sets PSR2_MAN_TRK_CTL SF Single Full Frame to trigger one full frame to be sent. Hardware will clear the bit after sending the frame.
- Software sets PSR2_MAN_TRK_CTL SF Continuous Full Frame to trigger full frames to be continuously sent until software clears the bit.
- These full frame fields do not cause a frame to be sent, they only set the frame to full size. To send a frame a flip is still needed.

When a full frame is sent, hardware uses the regular plane and cursor registers and not the SEL_FETCH registers, and sets the update region to the full frame, ignoring the programmed SU region.

Hardware will spontaneously send full frame updates on PSR exit, without software necessarily being aware of it, so the regular plane and cursor registers must always be programmed for the full frame.

DPST Handling

The DPST histogram is incorrect when used with selective fetch. DPST will usually give more power savings than selective fetch. Software can still use DPST with selective fetch, at least some of the time, by enabling the histogram only on selected frames (chosen periodically or algorithmically) and triggering selective fetch full frames while taking the histogram. Based on the histogram result, software can decide if the histogram needs to be taken more frequently for better power savings, with more full frames.

Selective Fetch Programming Sequence

1. DisplayPort enable mode set sequence.
2. For PSR2, configure PSR2_MAN_TRK_CTL SF Partial Frame Enable=1, and depending on project, PSR2_MAN_TRK_CTL PSR2 Manual Tracking Enable=1, then enable PSR2_CTL.
3. For Panel Replay, configure PSR2_MAN_TRK_CTL SF Partial Frame Enable=1, then enable Panel Replay in TRANS_DP2_CTL.
4. For each screen update
    a) If selective fetch for this frame, atomically update planes, cursor, selective fetch, and selective update registers, referring to the Adjusting Plane and Cursor Size and Position and Selective Update sections for the values.
        - All the registers must update atomically together in the same frame. Refer to the Double Buffer Control section, sequence for synchronizing double buffer updates.
            - The fields to allow double buffer stalling are located in PSR2_MAN_TRK_CTL, regular plane control, and regular cursor control registers.
        - The regular plane and cursor registers must be programmed for the full frame size because hardware can spontaneously send a full frame update.
    b) Else, send a full frame update, referring to the Full Frame Updates section.

YUV 4:2:0 Support

Overview

DP and HDMI ports support YUV 4:2:0 output (AKA YUV420). YUV 4:2:0 operates with the full blend mode where the scaler sub-samples YUV 4:4:4 to YUV 4:2:0 (Y & UV scaling) and a packer block formats the YUV 4:2:0 for DP and HDMI protocols.

YUV 4:2:0 operates only in full blend mode.

Programming

- Enabling or disabling YUV 4:2:0 pipe output requires a mode-set.
- Set PIPE_MISC YUV420 Enable to Enable and YUV420 Mode to Full Blend prior to enabling transcoder.
- The display planes pixel format can be YUV or RGB, but software has to ensure through pixel format selection and use of color space correction blocks that the color space is YUV at the input to scaling.
- Configure pipe scaler
    - Pipe scaler 1 must be enabled and located in the post-blend position (After CSC).
    - Pipe Scaler Window X and Y sizes must be even.
    - Pipe Scaler Window X and Y positions must be even.
    - Scaler input height minimum of 16 lines.
    - Limit downscaling to less than 1.5 (source/destination) in the horizontal direction and 1.0 in the vertical direction.
- The pipe vertical active display size must be a multiple of 2.
- For HDMI, the horizontal window size must meet the HDMI restriction of a minimum of 128 Y component transfers per scan line.
- For HDMI, when YCbCr 4:2:0 pixel encoding is active, pixel repetition is not permitted.
- Not supported with MSO/CoG.
- The transport of 4:2:0 pixels for interlaced video formats is not supported.
- YUV 4:2:0 and DSC compression are not supported concurrently.
- The horizontal active, sync start, sync end, and total must be a multiple of 4 for 8/10/12 bpc port output.
- When uncompressed pipe joining is enabled, then the horizontal active must be a multiple of 8.

YUV 422 Support

Keypoints

- YUV 4:2:2 (AKA YUV422) supported on both DP and HDMI.
- Not supported with eDP MSO/CoG.
- Supported with 8, 10 and 12 bpc.
- No dynamic switching from RGB to YUV 4:2:2 (and vice versa) or YUV 4:2:2 to YUV 4:2:0 (and vice versa).
- YUV 4:2:2 chroma sub-sampling (downscaling) has a dedicated function and does not require the pipe scaler.
- YUV 4:2:2 and DSC compression are not supported concurrently.

Configuration

Data rate for YUV 4:2:2 is 2/3 of 4:4:4 pixel formats. This requires adjustment in the link clock frequency for HDMI and M/N data rate for DP.

PIPE_MISC2

YUV 422 mode Enable

PIPE_MISC4

Left filter coeff, Center filter coeff, and Right filter coeff fields allow the sub-sampling coefficients to be adjusted.

Transcoder Video Data Island Packet

Data Island Packet (DIP) is a mechanism that allows data to be sent over a digital port during blanking, according to the HDMI and DisplayPort specifications. This includes header, payload, checksum, and ECC information.

Each type of Video DIP will be sent once each frame while it is enabled.

Video DIP
VIDEO_DIP_CTL
VIDEO_DIP_DATA
VIDEO_DIP_GCP
VIDEO_DIP_ECC
VIDEO_DIP_DRM_DATA
VIDEO_DIP_DRM_ECC
VSC_EXT_SDP_CTL
VSC_EXT_SDP_CONF
VSC_EXT_SDP_HEADER
VSC_EXT_SDP_DATA

Supported DIPs

HDMI	DP	eDP
General Control Packet (GCP) Auxiliary Video Information (AVI) Source Product Description (SPD) Vendor Specific (VS) Gamut Metadata Packet (GMP)	Gamut Metadata Packet (GMP) Video Stream Configuration (VSC)	Video Stream Configuration (VSC)
DRM		
	Picture Parameter Set (PPS)	

HDMI	DP	eDP
		Gamut Metadata Packet (GMP)
	VSC Extension SDP (VSC EXT SDP)	VSC Extension SDP (VSC EXT SDP)
	Adaptive Sync SDP	

Construction of DIP for AVI, VS, or SPD (HDMI only):

Dword	Byte3	Byte2	Byte1	Byte0
0	Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8 (RO)	Reserved	Reserved	Reserved	HB ECC
9 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0

HB = Header Byte, DB = Data Byte, RO = Read Only

Construction of DIP for GMP (HDMI or DisplayPort):

HDR (GMP) metadata, VSC and AVI are double buffered.

Most recent update at DB point will be used by HW.

1. Program video DIP data buffer registers for DIP being updated.
2. Enable the video DIP.

Dword	Byte3	Byte2	Byte1	Byte0
0	DP: HB3 HDMI: Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8	DB31	DB30	DB29	DB28

Dword	Byte3	Byte2	Byte1	Byte0
9 (RO)	Reserved	Reserved	Reserved	DP: Reserved HDMI: HB ECC
10 (RO)	DP: Reserved HDMI: DB ECC 3	DP: Reserved HDMI: DB ECC 2	DP: Reserved HDMI: DB ECC 1	DP: Reserved HDMI: DB ECC 0
11 (RO)	DP: HB ECC 3 HDMI: Reserved	DP: HB ECC 2 HDMI: Reserved	DP: HB ECC 1 HDMI: Reserved	DP: HB ECC 0 HDMI: Reserved
12 (RO)	DP: DB ECC 3 HDMI: Reserved	DP: DB ECC 2 HDMI: Reserved	DP: DB ECC 1 HDMI: Reserved	DP: DB ECC 0 HDMI: Reserved
13 (RO)	DP: DB ECC 7 HDMI: Reserved	DP: DB ECC 6 HDMI: Reserved	DP: DB ECC 5 HDMI: Reserved	DP: DB ECC 4 HDMI: Reserved

HB = Header Byte, DB = Data Byte, DP = DisplayPort, RO = Read Only

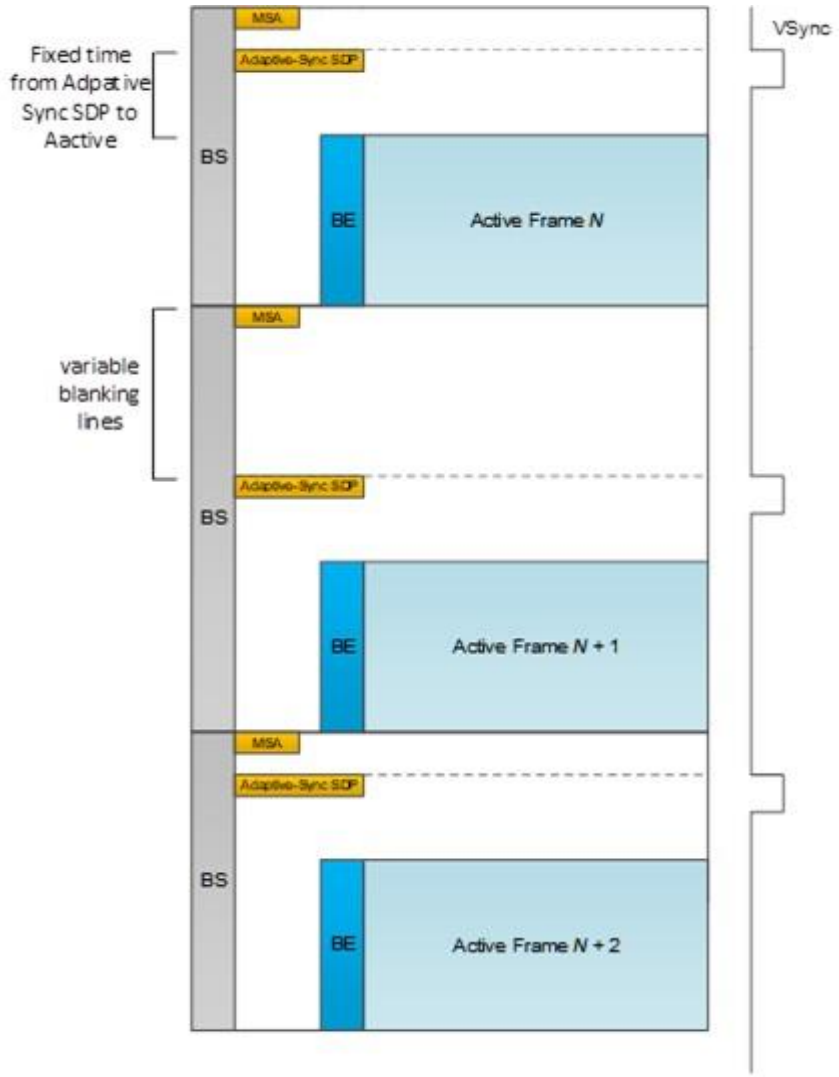
Construction of DIP for VSC (DisplayPort only):

Dword	Byte3	Byte2	Byte1	Byte0
0	HB3	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8	DB31	DB30	DB29	DB28
9 (RO)	HB ECC 3	HB ECC 2	HB ECC 1	HB ECC 0
10 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0
11 (RO)	DB ECC 7	DB ECC 6	DB ECC 5	DB ECC 4

HB = Header Byte, DB = Data Byte, RO = Read Only

Construction of DIP for DRM (HDMI Only)				
DWord	**Byte3**	**Byte2**	**Byte1**	**Byte0**
0	RSVD	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8 (RO)	RSVD	RSVD	RSVD	HB ECC
9 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0

Construction of adaptive sync SDP



Key Points

1. The start and end of adaptive sync SDP transmission occurs within the first half of the line that corresponds to the start of the Vsync pulse.
2. Valid HTotal[15:0], HStart[15:0], HSyncPolarity[0] (HSP), HSyncWidth[14:0], VStart[15:0], VSyncPolarity[] (VSP), VSyncWidth[14:0] (VSW), HWidth[15:0] and VWidth[15:0] are transmitted while transmitting an Adaptive-Sync SDP.
3. An Adaptive-Sync-capable DP protocol converter ignores only VTotal[15:0] while receiving an Adaptive-Sync SDP.
4. The SDP's presence marks the Vsync location for protocol converter timing output.
5. Adaptive sync SDP payload and parity bytes are cleared to 0.
6. Header byte 1 indicates adaptive sync SDP type.
7. Adaptive sync SDP should be enabled prior to VRR enable.

Exceptions

DP spec requires AdaptiveSync SDP to be sent in the first half of the line after BS. When audio is enabled, display HW sends it at BE.

For the following HDMI resolutions, HBlank > HActive and when SDP is sent at BE it violates the DP half-line requirement.

Note that the max refresh rate for these HDMI VICs is 30 Hz. Hence HDMI 2.1 compliance is not violated (base refresh rate < 50).

VIC	HActive	VActive	I/P	HTotal	HBlank	VFreq
60,65	1280	720	Prog	3300	2020	24
61,66	1280	720	Prog	3960	2680	25
62,67	1280	720	Prog	3300	2020	30

The audio subsystem is also capable of sending Data Island Packets. These packets are programmed by the audio driver and can be read by in MMIO space via the audio control state and audio HDMI widget data island registers.

Video DIP data write sequence:

1. Wait for 1 VSync to ensure completion of any pending video DIP transmissions
2. Disable the video DIP being updated (disable VDSC before updating PPS DIP)
3. Program video DIP data buffer registers for DIP being updated
4. Enable the video DIP

For MMIO programming of dynamic DIPs, Video DIP write sequence is as follows.

1. Wait to be outside the VBlank region so the following programming does not happen during the VBlank.
2. Program video DIP data buffer registers for DIP being updated.

The video DIP data and ECC buffers may be read at any time.

DIP data buffer registers must be programmed with valid data before enabling the DIP.

Partial DIPs are never sent out while the port is enabled. Disabling the DIP at the same time it is being transferred will result in the DIP being completed before the function is disabled.

Shutting off the port on which DIP is being transmitted will result in partial transfer of DIP data. There is no need to switch off the DIP enable bit if the port transmitting DIP is disabled.

When disabling both the DIP port and DIP transmission, first disable DIP and then disable the transcoder before the transcoder clock select is set to none.

Enabling a DIP function at the same time that the DIP would have been sent out (had it already been enabled) will result in the DIP being sent on the following frame.

For HDMI, even if no DIP is enabled, a single Null DIP will be sent at the same point in the stream that DIP packets would have been sent.

Notes on DIP use cases

GMP DIP can be used to transmit the CEA infoframe SDP since it allows full packet header customization.

There is no assumption in HW on the content of the SDPs (except when PSR is enabled).

For all non PSR cases HW transmits SDP using data programmed in the registers.

The controller does not use the data internally.

PPS is a unique packet as it is 128 bytes of data. No other SDP has this length.

The GMP is what is recommended for HDR metadata.

The VSC may also need to be transmitted with HDR content as it contains colorimetry information.

VSC Extension SDP Packets

VSC_EXT_SDP_CTL

VSC_EXT_SDP_CONF

VSC_EXT_SDP_HEADER

VSC_EXT_SDP_DATA

VSC extension packets can support frame synchronous meta data up to 1K bytes. VSC extension SDP packets add capability to support HDR Dynamic Meta Data. HDR dynamic meta data is supported for full frame updates only.

Key Assumptions

- Software should always write metadata in multiples of 8 DWs i.e., enough data to form one SDP packet.
- Hardware does not support padding of incomplete SDP packets.

- DP driver software has to determine whether a particular audio mode can be supported when transmitting extension packets.
- When Audio is enabled Meta Data will be sent only in horizontal active period of vblank.
- When Audio is disabled Meta Data will be sent both in horizontal active and horizontal blanking period of vblank.
- DSB or software should program the metadata and/or set the buffer ready bits when the relevant frame has started.
- Transcoder sends metadata in vblank region as long as at least one buffer is not empty and respective buffer done indication is set.
- Software should not start a new chain in the buffer while the previous chain data is still pending.
- If the chain doesn't fit in one buffer then software can continue to write the data to second buffer in ping-pong fashion. However, packet straddling across buffers is not allowed.
- DSB can continue writing to a buffer until either the chain is complete or 1KB data is programmed.
- The 8 deep xclkfifo + 2K buffers are replaced with 2KB cross clock fifos So, only auto increment mode is supported for both writes and reads. Random accesses to these 2 KB buffers are not supported for both writes and reads.
- Transcoder clock must be enabled while VSC_EXT_SDP_CTL is enabled or VSC_EXT_SDP_DATA is being written.

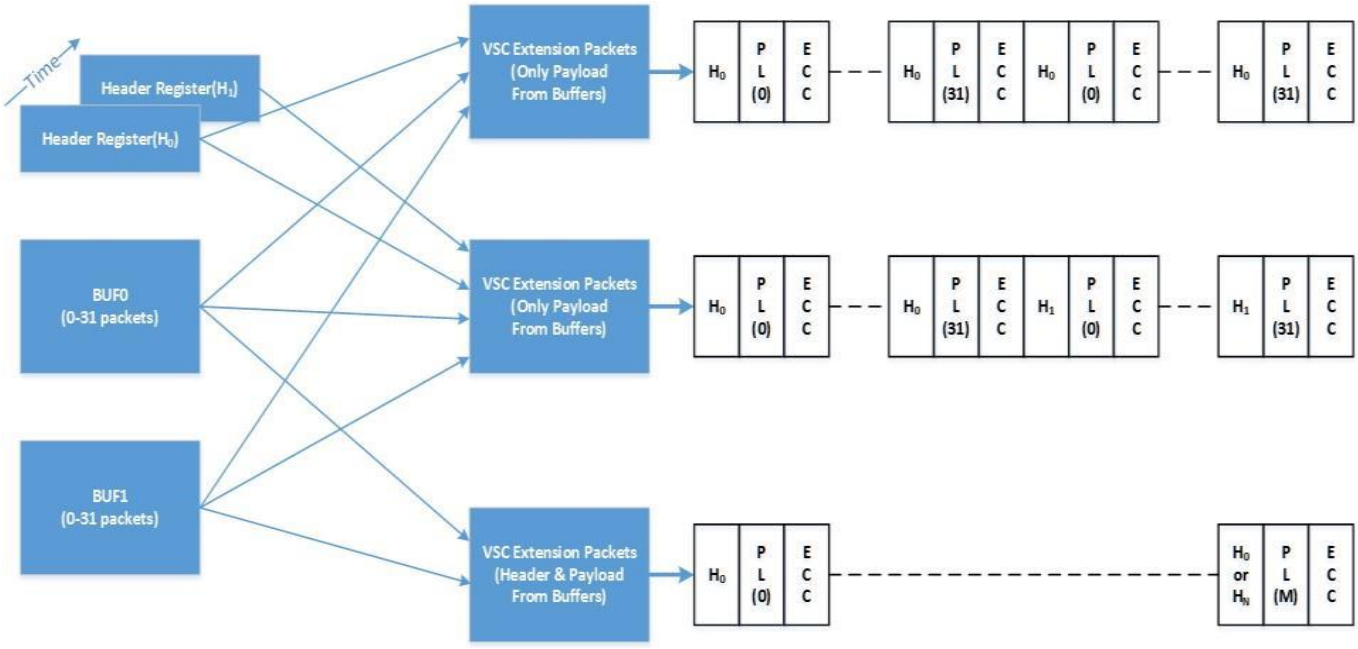
Programming sequence

There are 3 different ways of programming VSC extension packets based on the following considerations.

1. Packet header is from register and payload in buffer
2. Packet header and payload in buffer
3. Multiple VSC extension headers per frame with multiple chains per header

Type of Sequence	Max packets per Buffer
Only payload in buffer	32
Header and payload in buffer	28

The diagram below shows the final VSC extension packets from these 3 different programming methods.



Packet header is from register

In this use case, steps to program any buffer (BUF0 or BUF1) or as follows.

1. Program CTL Register - Auto Index Increment in VSC_EXT_SDP_CTL_0 [14] (or VSC_EXT_SDP_CTL_1 [14]) register bit.
2. Program BUF Register - Meta Data in VSC_EXT_SDP_DATA_0 (or VSC_EXT_SDP_DATA_1) register.
3. Set BUF ready bit - VSC EXT_SDP_CTL0 [16] (or VSC_EXT_SDP_CTL1[16])

The complete sequence is as follows.

1. Program number of packets per chain in VSC_EXT_SDP_CONF[9:0] register.
2. Program Header values in VSC_EXT_SDP_HEADER register.
3. Program BUF0
4. Program BUF1 if needed (payload > 1 KB)
5. Repeat steps 3 and 4 after poll/wait for corresponding BUF empty indication - VSC_EXT_SDP_CTL0[24] (or VSC_EXT_SDP_CTL1[24])

Packet header is from buffer

The number of packets per chain is implicit in the buffer.

In this use case, steps to program any buffer (BUF0 or BUF1) or as follows.

1. Program CTL Register - Auto Index Increment in VSC_EXT_SDP_CTL0 [14] (VSC_EXT_SDP_CTL1 [14]) or register bit.
2. Program BUF Register - Header & Meta Data in VSC_EXT_SDP_Data0 (or VSC_EXT_SDP_Data1) register.
3. Set BUF ready bit - VSC_EXT_SDP_CTL0 [16] (or VSC_EXT_SDP_CTL1[16])

To set this up, program "middle of chaining" bit in the header to '1' and enable "block mode" in VSC_EXT_SDP_CONF[22].

1. Program BUF0
2. Program BUF1 if needed (payload > 1 KB)
3. Poll/Wait for BUF0 Empty indication - VSC_EXT_SDP_CTL0 [24] .
4. Poll/Wait for BUF1 Empty indication - VSC_EXT_SDP_CTL1 [24] if BUF1 is programmed.
5. Repeat steps 1 to 4 until entire metadata is transmitted.

Multiple headers per frame

In this use case, steps to program any buffer (BUF0 or BUF1) are as follows.

1. Program CTL Register - Auto Index Increment in VSC_EXT_SDP_CTL0 [14] (VSC_EXT_SDP_CTL1 [14]) or register bit.
2. Program BUF Register - Meta Data in VSC_EXT_SDP_Data0 (or VSC_EXT_SDP_Data1) register.
3. Set BUF ready bit - VSC_EXT_SDP_CTL0 [16] (or VSC_EXT_SDP_CTL1[16]) .

The driver will program multiple chains with different number of packets and different header for each chain. As it requires programming of multiple headers per frame, driver has to poll for Chain done bit to avoid overwriting header register before hardware consumes it. Polling of Chain done instruction should be inserted after the programming of entire meta data for each chain is complete.

1. Program number of packets per chain in VSC_EXT_SDP_CONF[9:0] register.
2. Program Header Register - Header values in VSC_EXT_SDP_HEADER register.
3. Program BUF0
4. Program BUF1 if needed (payload per chain > 1 KB)
5. Poll/Wait for BUF0 Empty indication - VSC_EXT_SDP_CTL0 [24] .
6. Poll/Wait for Chain Done indication - VSC_EXT_SDP_CONF[31]
7. If BUF0 is empty but Chain is not done, poll/wait for BUF1 Empty indication. Otherwise, go to step 8.
8. Repeat steps 1 to 7 above until entire meta data is programmed.

DSB side programming considerations

If DSB is being used to program metadata with polling for buffer empty, then the DSB polling timeout (100 uS) will need to be increased, recommending >=500us if metadata programming is started in vblank, or >=frame time if metadata programming is started in vertical active.

SDP Re-transmission

VSC_EXT packets are transmitted every frame without requiring the 2KB metadata buffer being re-programed every frame by moving from DSB replay to the DisplayPort controller retransmitting the metadata from its local buffer. The options of header in buffer and header not included in the metadata buffer work with the buffer replay mode, for up to one chain of 2 full buffers. Multiple chains per frame does not work with the buffer replay mode. This mode is set/reset only when VSC_EXT buffers are invalidated.

VSC_EXT is transmitted starting on line 3 of delayed vblank (VRR) or line 8 of vblank (non-VRR). The VSC_EXT_SDP_CTL Buffer Empty bit will set when metadata transmission finishes and clear when metadata transmission starts in the next vblank.

For re-transmission, extra programming is added before the metadata programming from above to setup the re-transmit and ensure the data is being updated in a safe region.

1. Wait for safe region before metadata transmission. The metadata is not double-buffered and cannot be updated while transmission is happening during vblank.
    - DSB common usage achieves this by waiting for VRR safe window or the non-VRR start of vblank.
    - Non-DSB usage can achieve this by polling for VSC_EXT_SDP_CTL Buffer Empty (for both BUF0 and BUF1 if both are in use) and a scan line that is well before the metadata transmission start and controlling the use of VRR push.
    - Metadata can also be programmed while the transcoder is disabled to setup transmission for the first vblank or the first frame when transcoder is enabled.
2. Set VSC_EXT_SDP_CONF Buffer Replay = 1 if not already set
3. Clear VSC_EXT_SDP_CTL Buffer Ready in both buffers if not already cleared
4. Wait for buffer write pointer to clear (delay or status bit - **Buffer Clear Status**) in both buffers. This should complete within a few clock cycles.
5. Continue with rest of metadata programming. Write pointers are reset, so buffer data must be programmed from scratch again for any buffer that will be used.

Transcoder DDI Function

TRANS_DDI_FUNC_CTL

TRANS_MSA_MISC

TRANS_DDI_FUNC_CTL2

Panel Self Refresh

This section is about Panel Self Refresh (PSR). PSR1 at one point was named Self Refreshing Display (SRD).

Panel Self Refresh version 1 (PSR1)

PSR1 enable sequence:

- Prerequisite: The associated transcoder and port are running and Aux channel associated with this port has IO power enabled.
- Set PHY power state to P2 by following these steps.
  - Program Register_SNPS_PHY_TX_REQ [Lane disable power state in PSR] to "10".
- Prerequisite: The associated transcoder, port and at least one plane are running.

1. Configure FBC host and render tracking. The FBC function does not need to be enabled in FBC_CTL.
2. Program Transcoder EDP VSC DIP data with a valid setting for SRD/PSR.
3. Configure and enable SRD_CTL.

PSR1 disable sequence:

Prerequisite: The associated transcoder and port are running.

1. Disable SRD_CTL
2. Wait for SRD_STATUS to show SRD is Idle. This will take up to one full frame time (1/refresh rate), plus SRD exit training time (max of 6ms), plus SRD aux channel handshake (max of 1.5ms).
3. Set PHY power state to the default by following these steps.
   - Program Register_SNPS_PHY_TX_REQ [Lane disable power state in PSR] to "11".

PSR1 Exit Events:

The following events will cause the hardware to automatically perform a PSR1 disable if the source is Active or entering the Active state (i.e., sending the Capture frame).

Notes:

1. The Source does not make any optimizations when an Exit event is received during the Capture frame. The Capture frame will be completed, the Link will be put to sleep, the timing generator (TG) will be turned off and only then will the Source begin the exit (i.e., wake the Link back up, turn the TG back on, and then move back to the PSR1 Inactive state (i.e., SRD Idle)).
2. Software should expect the same exit latency (i.e., waiting for SRD to go to Idle) as described above in the "PSR1 Disable Sequence" when any of these exit events occur

The PSR1 exit events are:

- Pipe VBI's are enabled (DE_PIPE_INTERRUPT[Vblank] IMR=0 and IER=1)
- There is a flip when Single Frame Updates are Enabled (SRD_CTL)

- An HDCP session has started
- A KVMr session has started
- There is an FLR
- There is a Hot Plug event (this can be masked from the PSR_MASK register)
- The Display engine received a Memory Up without sending a Wake (this can be masked from the PSR_MASK register)
- There is a Max Sleep timeout (this can be masked from the PSR_MASK register)
- The front buffer of the FBC has been modified (this can be masked from the PSR_MASK register)
- There is a Low Power Single Pipe event (this can be masked from the PSR_MASK register)

Adaptive Sync Frame Update (ASFU)

ASFU combines PSR link disable mode, VRR, and PSR single frame update. ASFU allows the vertical blank size to vary as it does for VRR, but then it will enter PSR link disable after the maximum vertical blank, and on PSR exit it can send a single frame before re-entering link disable.

ASFU enable sequence:

- Prerequisite: The associated transcoder and port are running.
- Prerequisite: The associated transcoder, port and at least one plane are running.

1. If SRD_CTL Adaptive Sync Frame Update was not disabled in step 3 of the previous ASFU disable sequence, then disable it.
2. Configure FBC host and render tracking. The FBC function does not need to be enabled in FBC_CTL.
3. Enable VRR - see Transcoder VRR Function chapter
4. Program Transcoder EDP VSC DIP data with a valid setting for SRD/PSR.
5. Configure PSR/SRD Registers and set SRD_CTL Adaptive Sync Frame Update.
6. Set masking bits for PSR to use the VRR push mode
    - PIPE_MISC[Change Mask for Register Write] to '1'
    - PIPE_MISC2[ASFU Flip exception] to '1'.
    - PIPE A: 0x420B0[11] to '1' to mask flips from being frame update events
    - PIPE B: 0x420B4[11] to '1' to mask flips from being frame update events
    - PIPE C: 0x420B8[11] to '1' to mask flips from being frame update events
    - PIPE D: 0x420BC[11] to '1' to mask flips from being frame update events
7. Enable bit 31 of SRD_CTL.

Note that ASFU can be used together with VRR flip line. See the Transcoder VRR Function page for flip line programming since that can be used independent of PSR and ASFU.

ASFU disable sequence:

- Prerequisite: The associated transcoder and port are running.
- Prerequisite: The associated transcoder, port and at least one plane are running.

1. Disable SRD_CTL bit 31.
2. Wait for SRD_STATUS to show SRD is in Idle. This will take up to one full frame time (1/refresh rate), plus SRD exit training time (max of 6ms), plus SRD aux channel handshake (max of 1.5ms).
3. Disable SRD_CTL Adaptive Sync Frame Update, or alternatively, SRD_CTL Adaptive Sync Frame Update can remain enabled until the next ASFU enable sequence is entered.
4. Disable VRR - see Transcoder VRR Function chapter

Panel Self Refresh version 2 (PSR2)

PSR2 is not supported.

PSR2 enable sequence:

Prerequisite: The associated transcoder and port are running and Aux channel associated with this port has IO power enabled.

Set PHY power state to P2 by following these steps.

- Program Register_SNPS_PHY_TX_REQ [Lane disable power state in PSR] to "10".

Prerequisite: The associated transcoder, port and at least one plane are running.

1. Configure FBC host and render tracking. The FBC function does not need to be enabled in FBC_CTL.
2. Program Transcoder EDP VSC DIP header with a valid setting for PSR2 and configure VIDEO_DIP_CTL VSC fields.
3. Enable GTC/Aux Frame Sync, if required.
4. Program Aux control register Fast Wake Sync Pulse Count to 8 pulses.
5. Configure Idle Frame, Selective Update Tracking Enable, and Y-coordinate fields, and enable PSR2_CTL.

When configuring PSR2_CTL, also configure the Fast Wake and IO Buffer Wake fields

- IO buffer wake lines = ROUNDUP(PSR2 IO wake time / total line time in microseconds)
- Fast wake lines = ROUNDUP(10us Aux Sync Pattern + 32 microseconds Aux PHY_WAKE transaction time / total line time in microseconds)
    - eDP standard calculation for Aux maximum Fast Wake from FW_SLEEP: 8us Aux preamble + 4us PHY_WAKE pattern + 20us tFW_EXIT_LATENCY
- For both fields limit the minimum to 7 lines and maximum to 12 lines

PSR2 IO wake time = 10us PHY latency + 32us eDP standard maximum Fast Wake from FW_SLEEP

- eDP standard calculation for IO maximum Fast Wake from FW_SLEEP: 11us of valid 8b/10b symbols during Aux preamble and first part of PHY_WAKE pattern + 1us tML_PHY_LOCKsu + 20us tFW_EXIT_LATENCY = 32us

This maximum of 12 lines with 42 wake means PSR2 requires total line time of at least 3.5us.

Panels with relaxed requirements may allow the wake times to be customized.

When configuring PSR2_CTL, also configure "SU SDP scanline indication" field as required.

PSR2 disable sequence:

1. If Pipe VBI's are enabled and the port is synchronized with the CMTG, then the Driver should switch over to using the CMTG VBI's while disabling PSR2.
    - A PSR2 exit will cause the port to take the Deep Sleep path to the Inactive state which results in loss of timing (i.e., the port timing generator is turned off and then turned back on)
2. Program PSR2_CTL to clear PSR2 Enable.
3. Disable GTC if required.
4. Wait for PSR2_STATUS to show PSR2 is Idle. This will take up to one full frame time (1/refresh rate), plus exit training time (max of 6ms), plus aux channel handshake (max of 1.5ms), plus one more full frame time if exit began while in a capture frame.
5. If not LRR: Program PSR2_CTL to clear Selective Update Tracking Enable.
6. Set PHY power state to the default by following these steps.
    - Program Register_SNPS_PHY_TX_REQ [Lane disable power state in PSR] to "11".

Do not enable PSR2 if the V. Blank time is less than the Block Count Number value in lines. PSR2 can be enabled when:

- PSR2_CTL[SU SDP scanline indication] = 0: (TRANS_VBLANK Vertical Blank End- TRANS_VBLANK Vertical Blank Start) > PSR2_CTL Block Count Number value in lines
- PSR2_CTL[SU SDP scanline indication] = 1: (TRANS_VBLANK Vertical Blank End- TRANS_VBLANK Vertical Blank Start- 1) > PSR2_CTL Block Count Number value in lines

This minimum number of lines is required to give time to wake the IO when there is an update starting from the first active line.

This minimum block count of 8 lines means PSR2 requires vblank to be at least 8 lines.

PSR2_MAN_TRK_CTL

Hardware supported auto hardware tracking mode is defeatured. Only manual tracking mode is supported.

The PSR2_SU_STATUS registers will only return a value of 0.

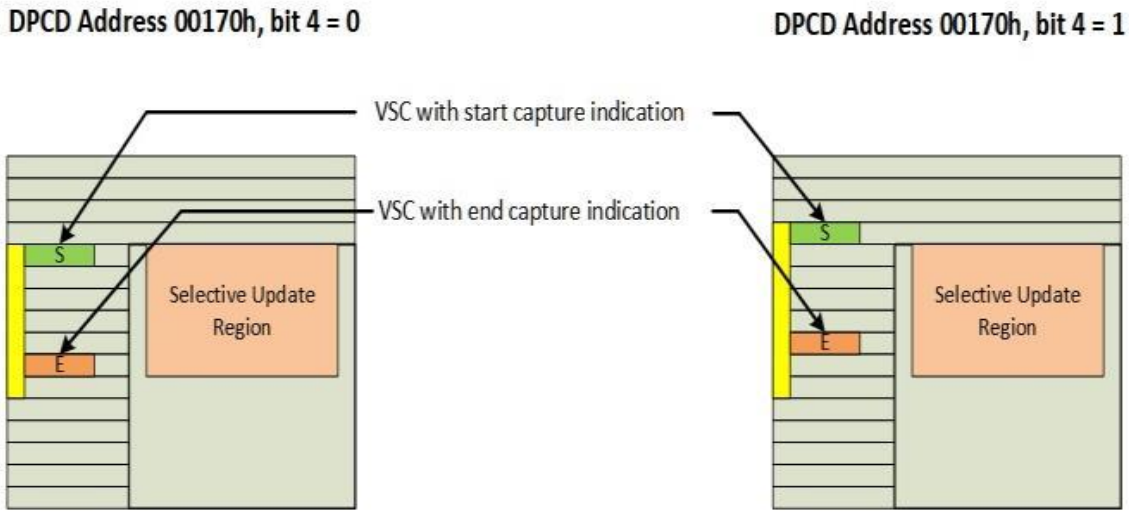
Sending an Update to Sink:

To send a selective update (SU) to the Sink, Software will need to do the following (programming order is up to Software):

1. Program PSR2_MAN_TRK_CTL with vertical region containing the update (hardware only supports full horizontal SU regions)
    - The "SU Region Start Address" (i.e., the first line of the SU) and "SU Region End Address" (i.e. the last line of the SU) defines the vertical coordinates of the SU region
        - Valid addresses: 0 <= Start Address <= End Address <= TRANS_VTOTAL[Vertical Active]
        - If Start Address = End Address, then the SU is just one line
    - The register can be programmed to send a single or multiple full frame SUs
        - The full frame bits will override the programmed Start and End addresses where hardware will assign the Start Address = 0 and the End Address = TRANS_VTOTAL[Vertical Active]
        - The Single Full Frame (SFF) bit is set by Software and cleared by hardware when the first rising edge of delayed vblank is seen
        - The Continuous Full Frame (CFF) bit is completely controlled by Software (i.e., the bit is set and cleared by Software)
    - Hardware samples the PSR2_MAN_TRK_CTL programming on the entry of the delayed vblank (i.e., the delayed vblank is the double buffer (DB) point of the SU coordinates)
2. Software will generate a "Frame Change" to the transcoder via any Pipe / Plane register write
    - The transcoder will log that there is a pending SU when it receives a Frame Change event and it will schedule the SU for transmission in the upcoming frame when it enters the next delayed vblank
        - The transmission of the SU will correspond to the Start / End coordinates specified by the PSR2_MAN_TRK_CTL programming
    - If there are Frame Change events for a given SU that span a delayed vblank (i.e., the DB point), then hardware will send multiple SU's over the next two frames which could lead to corruption if the SU coordinates are stale

Software is responsible for coordinating the PSR2_MAN_TRK_CTL programming with the Frame Change events and ensuring both happen atomically within a frame time (from delayed vblank entry to delayed vblank entry).

Scanline selection for PSR2 SDP transmission:



Display controller must completely transmit PSR2 SDPs with first and last scan line indications in the horizontal blank region of the respective scan lines, 100ns prior to the first pixel of the

SU region. However, this may not be possible for higher resolutions due to reduced blank duration. As an example, 5K@60 with RB2 (reduced blanking 2) would not have enough time in the Hblank period to completely transmit the VSC SDP 100ns prior to first pixel.

If the PSR2 SDPs cannot be transmitted 100ns prior to the SU region, the eDP standard allows display controller to transmit the PSR2 SDP during horizontal blanking of the previous scan line as shown above. For this purpose, display software must set the Selective Update Region Scan Line Capture Indication bit in the PANEL SELF REFRESH CONFIGURATION register (DPCD Address 00170h, bit 4) to 1.

Display Software Impact

- Read sink capabilities and check that the Sink supports at a minimum eDP1.4b and PSR2 SU.
- During modeset, check that (hblank time ns – (((60 / number of lanes) + 11) * 1000 / symbol clock frequency MHz) > 100 ns
- If the check fails
    - Program PSR2_CTL [SU SDP scanline indication] to '1'
    - Program DPCD 00170h bit 4 to '1' in the panel

PSR2 Deep Sleep:

While PSR2 is Active it has the ability to go into a deeper sleep state where the transcoder's timing generator is shut off (this will allow DC6 if enabled). The PSR2 function will look for a programmable number of "idle frames" while it is Active and move to the Deep Sleep state when it sees the required number of idle frames as long as all of the following conditions are met:

- Pipe Vertical blank interrupts (VBI's) are disabled (DE_PIPE_INTERRUPT[Vblank] IMR=1 or IER=0)
- The number of idle frames is non-zero (PSR2_CTL[Idle Frames] > 0)
- The PSR2 function has seen the programmed number of Idle Frames consecutively without any selective updates while it has been in the Sleep state (i.e., after Capture frame and sitting idle with the main link off)

Notes:

1. The PSR2 function will also move to the Deep Sleep state when there is an exit event (e.g. PSR2 is being Disabled, a KVMr session is starting, an HDCP session is enabled, a PCH hotplug event, or a functional level reset (FLR) is in progress)
    a) If Pipe VBI's are being used and the port is synchronized to the CMTG, then the Driver should switch over to using CMTG VBI's before the exit event happens.
2. The PSR2 function will only check the above conditions (including a PSR2 exit) when entering the delayed V. Blank (i.e., it will only move to the Deep Sleep state on an entry of the delayed V. Blank).
3. The value of PSR2_CTL[Idle Frames] should be set to a value that is greater than the frequency at which the transcoder sees frame changes (i.e. flips) if Software does not wish the PSR2 function to pre-maturely enter the Deep Sleep state.

PSR2_CTL[Idle Frames] > CEILING(Refresh Rate / Flip Rate)

As an example of the Idle Frames programming, if the refresh rate is set to 120Hz and the flip rate is 24Hz (e.g. video playback), then PSR2_CTL[Idle Frames] should be greater than 5 frames since flips will only be seen every 5 frames.

Once PSR2 has entered the Deep Sleep state, then any of the following actions will cause the PSR2 fucntion to exit the Deep Sleep state.

- Pipe VBI's are enabled (DE_PIPE_INTERRUPT[Vblank] IMR=0 and IER=1)
- There is a frame update present (i.e. a Frame Change was received)
- The PSR2 function is exiting the Active state (i.e. there is an exit event)

Notes:

1. After a Deep Sleep exit event happens, the transcoder will move to waking the main link back up. The amount of time this takes is specified by PSR2_CTL[TP2 Time].
2. After the main link is awake the Port timing generator will be re-started and PSR2 will exit to the Inactive state.
3. If the Port is being synchronized to a CMTG, then the Port timing generator will wait until it sees the next sync event from the CMTG before exiting to the Inactive state

Registers
SRD_CTL
SRD_STATUS
SRD_PERF_CNT
PSR_IMR
PSR_IIR
PSR_MASK
PSR_EVENT
PSR2_CTL
PSR2_MAN_TRK_CTL
PSR2_SU_STATUS
PSR2_STATUS

Transcoder WD Function

Register Links
TRANS_WD_FUNC_CTL
WD_STRIDE
WD_SURF
WD_TAIL_CFG
WD_TAIL_CFG2
WD Interrupt Bit Definition
WD_IMR
WD_IIR
WD_FRAME_STATUS
TRANS_HTOTAL
TRANS_VTOTAL
TRANS_CONF
WD_VFID

WD is a transcoder for capturing display pipe pixel output to memory. It is generally intended for wireless display, but can be used for other functions.

Like the transcoders for wired ports, WD has a timing generator that initiates each frame, and it formats pipe pixel data output. Unlike wired ports, it does not pass the formatted pixel data to DDI or other port logic, but instead writes it to memory.

The WD timing generator uses TRANS_HTOTAL Horizontal Active and TRANS_VTOTAL Vertical Active to define the active pixel area that is written to memory, with the WD_SURF and WD_STRIDE specifying the memory surface attributes.

The maximum resolution is 3840x2160 60Hz, with any capture color format.

Setup

Enable WD following the Sequences for WD - Enable Sequence and set TRANS_WD_FUNC_CTL Triggered Capture Mode Enable before or at the same time the WD Function Enable it set. This will put WD into the Triggered Capture Mode where it will wait for a capture to be triggered.

Running (after setup)

1. Trigger a frame capture by writing TRANS_WD_FUNC_CTL with the intended Frame Number and Start Trigger Frame = 1. The vertical blank will start, causing double buffered registers to update, and then capture will begin.

2. Find when capture completes by polling WD_FRAME_STATUS Frame Complete or waiting for WD Frame Complete interrupt.
    - There is no hardware timeout for the triggered capture mode. If capture does not complete within 50 milliseconds, write 1 to TRANS_WD_FUNC_CTL Stop Trigger Frame, then correct the configuration before starting another capture.
3. If using WD_FRAME_STATUS Frame Complete to find the frame completion, write 1 to that field to clear it in preparation for the next frame.
4. If another frame is needed
    a. Update any configuration that needs to be changed for the next frame; like WD_SURF, WD_STRIDE, or pipe and plane double-buffered registers.
    b. Goto 1, using an incremented Frame Number

Do not trigger a frame capture in the last frame when disabling WD. Wait for the last capture frame to finish, or hang and timeout, before disabling WD.

The frame number is used to synchronize capture and encode. Software selects the frame number and starts the capture with it. Transcoder WD includes the frame number in the pointer message sent to the encoder. The encoder reads that to align itself to the correct frame. The frame number should increment on each captured frame.

Transcoder Port Sync

Feature Description

PORT SYNC is a transcoder level feature. This mode forces two or more transcoders to be in sync with one transcoder primary and one or more transcoder secondaries. In the case of DP/eDP, the primary is unaware that it is operating in Port Sync mode. Only the secondary is aware that it is operating in this mode. Hence, port sync mode is only enabled in the secondary transcoder.

Support
Port Sync mode can be enabled with both DisplayPort SST and MST

DP/eDP Port Sync Restrictions

1. The secondary and primary transcoders and associated ports must have identical parameters and properties.
2. They must have the same color format, link width (number of lanes enabled), resolution, refresh rate, dot clock, TU size, M and N programming, etc.
3. PSR/PSR2/ASFU/ASU would need to be disabled when port sync mode is enabled.
4. Port Sync Mode Primary Select must be programmed with a valid value when Port sync Mode is enabled.

Transcoder Port Sync Support Table

Below is the complete list of primary and secondary transcoders that are supporting this feature.

The 1st column represents the primary transcoders and the 1st row represents the secondary transcoders.

Secondary ->	TC A	TC B	TC C	TC D	TC WD0	TC WD1	TC DSI0	TC DSI1
TC A	N	Y	Y	Y	N	N	N	N
TC B	Y	N	Y	Y	N	N	N	N
TC C	Y	Y	N	Y	N	N	N	N
TC D	Y	Y	Y	N	N	N	N	N
TC WD0	N	N	N	N	N	N	N	N
TC WD1	N	N	N	N	N	N	N	N
TC DSI0	N	N	N	N	N	N	N	Y
TC DSI1	N	N	N	N	N	N	N	N

Multichip Genlock

Cross-feature compatibility

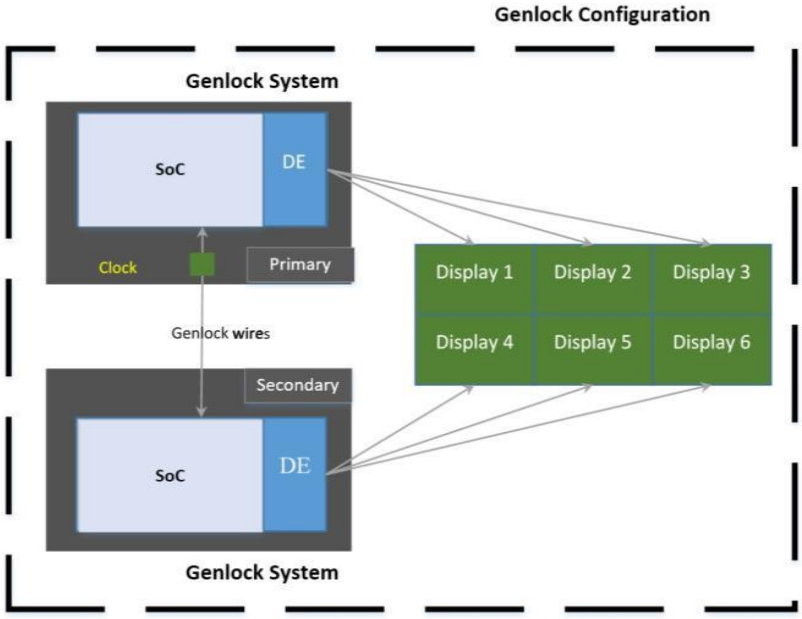
- Multichip genlock does not work with VRR (secondaries do not re-align vblank after enable mode set completes).

Description

With the multichip Genlock feature, platform/application software will synchronize frames (up to N-1 frame) across displays connected to multiple systems. The Nth frame synchronization is what HW portion of this Multi-chip Genlock will accomplish (i.e., all tiles display same scan line of same frame within +/- D pixels, as an example).

Below picture illustrates a wall of 2x3 displays connected to 2 systems in which one acts as genlock primary generating the clock and the other is genlock secondary receiving the clock from primary. DPLLs belonging to display 1, 2 and 3 are using the same PLL reference clock and that same reference clock propagates to DPLLs belonging to display 4, 5 and 6 over genlock wire as shown.

As part of multichip genlock configuration, each Genlock system below guarantees that (a) all display PLLs are running off of the same reference clock and (b) all displays receive their vertical reference (can be a separate wire or combined with reference clock) at the same time. However, flipping the right frame across displays is the responsibility of application software.



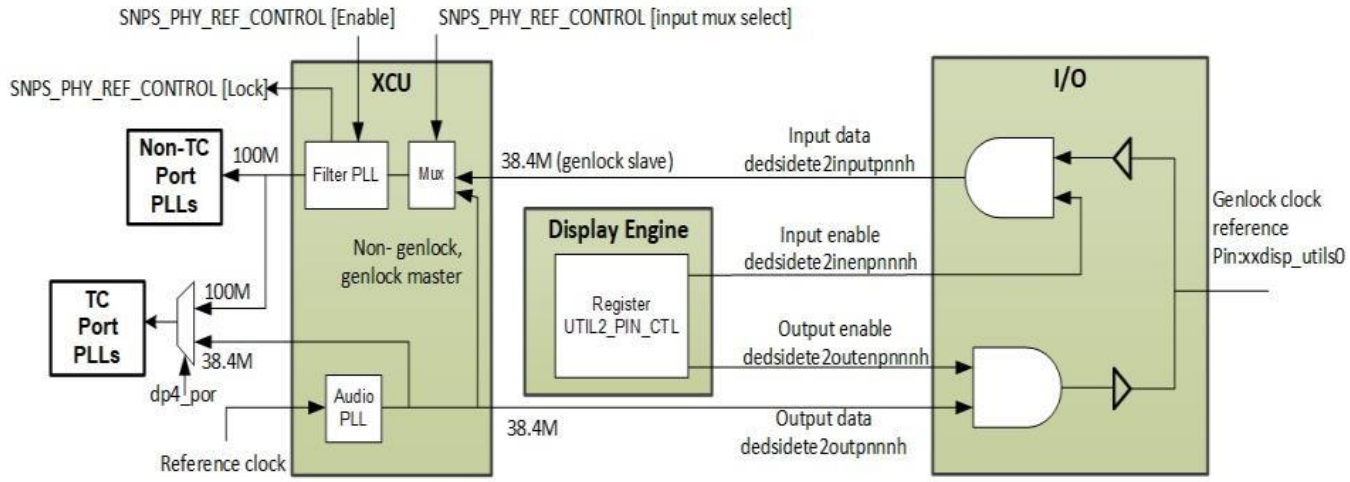
Key functional considerations

- South display is driving genlock enable and direction.
  - BIOS VBT can be configured to target unused GPIO pins.
- Within a genlock configuration, display driver will configure one transcoder as genlock primary (or secondary) and hooks up other transcoders as genlock secondaries.
- Only instances of identical platforms may be genlocked. Different types of platforms may not be mixed in a genlock topology.
  - For example, all platforms must use the same PLL reference clock frequency.
- All displays in a Multichip Genlock system must have the exact same timing requirement and support same transport protocol.
  - As an example, two panels with different 1080p timings are not allowed
- All genlock systems must train their links identically.
- Multichip Genlock is supported with both HDMI and DP (SST only) but not both on the same configuration. DP MST mode is not supported.
- Multichip genlock is not supported with software programmable Vtotal feature (LRR/LRR2).
- After genlock selection is made, chosen system enters genlock primary mode before any other system enters into genlock secondary mode.
  - Both primary and secondary ports finish training and have their transcoder enabled.
  - Primary pipe is enabled.
  - Secondary pipe is enabled.
- Runtime selection of Genlock primary or secondary requires a full mode set.
- Switching PLL reference for multichip Genlock operation requires mode set.

- Modeset across displays connected to the same genlock system can happen in multiple modeset calls.
- Genlock selection is not persistent across reboots.
- A genlock secondary will not generate vblank interrupts until after genlock sync is achieved.
- Genlock secondarys will exit first from genlock secondary mode to default mode before any genlock primary would.
- A genlock primary exits from genlock primary mode to default mode only after all genlock secondaries have switched to default mode.
- Multichip Genlock cannot be enabled at the same time port sync is enabled.
- Note: Genlock hardware may share pins with other mutually exclusive features as follows.

Genlock function	Pin	Pin functionality shared with
Vertical reference (frame sync)	Utility pin 1	MIPI DSI_DE_TE (TE1)
PLL reference clock	Utility pin 2	MIPI DSI_DE_TE2 (TE2)
Genlock Direction Enable	BL PWM/Genlock En	Backlight PWM
Genlock Direction Select	BL Enable/Genlock Sel	Backlight Enable

SNPS_PHY_REF_CONTROL



The genlock PLL has been repurposed as Display PHY filter PLL and is capable of filtering crystal clock in non GenLock mode or GenLock primary mode, and filtering GenLock input clock in the GenLock secondary mode. Display PHY filter PLL always produces 100 MHz output. Output of GenLock PLL will always be used by all display PHYs, irrespective of the GenLock mode.

A dedicated GPIO buffer is used for GenLock feature. In GenLock secondary mode the SoC will receive a clock from the platform to be used as reference clock for display PHY. On the other hand, in primary mode, the SoC will forward output of the crystal clock as GenLock clock to the platform.

Genlock filter PLL has to be brought up during reset. Genlock secondary mode mux select at input to genlock filter PLL defaults to 0 (non-genlock), so during reset the genlock filter PLL locks with crystal because genlock won't be enabled until after boot.

When genlock is started (after boot) the driver has to switch over to the genlock reference. It has to disable all the PHY PLLs (normally shouldn't be any on yet) which will cause screen blinks, and disable the genlock filter PLL, switch to secondary mode, then re-enable the PLLs.

After reset, the driver can disable the genlock filter PLL to save power if there will not be any use of display output ports until the next reboot (i.e., non-display SKUs).

The filter PLL is controlled through fields the SNPS_PHY_REF_CONTROL register in the **port A** DE shim. The same fields are ignored in the other shims.

Genlock Filter PLL ratio is hardwired at SoC. DE cannot change it.

Genlock filter PLL lock goes to PCU so it can sequence the reset and also goes to DE for when driver controls the PLL to switch into secondary mode.

Sequence to enable MPLLB using filtered genlock reference:

This switches the filter PLL to the genlock reference, which all the port PLLs will then use, and is required preparation before the mode set enable sequence for a genlock secondary device secondary transcoder.

1. Start from disable mode set to turn off any enabled port PLLs
2. Configure genlock system to send the genlock reference into this chip
3. Disable filter PLL
    a) Clear SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_en to 0
    b) Poll for SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_req == 0. Timeout and fail after 100us.
    c) Clear SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Enable to 0
    d) Poll for SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Lock == 0. Timeout and fail after 100us.
4. Set SNPS_PHY_REF_CONTROL_PORT_A Filter PLL input mux select to 1.
5. Enable filter PLL
    a) Set SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Enable to 1
    b) Poll for SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Lock == 1. Timeout and fail after 100us.
    c) Set SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_en to 1
6. Mode set enable sequence for the secondary device secondary transcoder

Sequence to disable a MPLLB using filtered genlock reference:

This switches the filter PLL to the crystal reference, which all the port PLLs will then use, and is required when the genlock primary device will stop sending the genlock reference to secondary devices.

1. Start from disable mode set to turn off any enabled port PLLs
2. Disable filter PLL
    a) Clear SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_en to 0
    b) Poll for SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_req == 0. Timeout and fail after 100us.

c) Clear SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Enable to 0

d) Poll for SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Lock == 0. Timeout and fail after 100us.

3. Clear SNPS_PHY_REF_CONTROL_PORT_A Filter PLL input mux select to 0.
4. Configure genlock system to stop sending the genlock reference into this chip
5. Enable filter PLL
   a) Set SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Enable to 1
   b) Poll for SNPS_PHY_REF_CONTROL_PORT_A Filter PLL Lock == 1. Timeout and fail after 100us.
   c) Set SNPS_PHY_REF_CONTROL_PORT_* <all shims> dp_ref_clk_en to 1

Display Controller Programming

If Primary device

- Primary device refers to the display controller that contains the primary transcoder.
- Primary transcoder
  - Primary transcoder outputs the frame sync to be used by local or remote secondaries.
  - TRANS_DDI_FUNC_CTL2 Genlock Enable = 1
  - TRANS_DDI_FUNC_CTL2 Genlock Mode = 10 (Primary)
  - Set TRANS_DDI_FUNC_CTL2 [port sync mode primary select] to register field default.
  - SNPS_PHY_REF_CONTROL[Filter PLL input mux select] = 0b (non-genlock reference)
- Secondary transcoders
  - Secondary transcoders within the primary device are local secondaries.
  - Local secondaries receive frame sync from the primary transcoder in the same device and use the local PLL reference.
  - TRANS_DDI_FUNC_CTL2 Genlock Enable = 1
  - TRANS_DDI_FUNC_CTL2 Genlock Mode = 00 (Local Secondary)
  - Set the primary transcoder in TRANS_DDI_FUNC_CTL2 [port sync mode primary select] field.
  - SNPS_PHY_REF_CONTROL[Filter PLL input mux select] = 0b (non-genlock reference)
- Pin configuration
  - Frame sync output from primary transcoder to secondary devices
    - UTIL_PIN_CTL Enable = 1
    - UTIL_PIN_CTL Direction = 0 (output)
    - UTIL_PIN_CTL Mode = 0110b (framestart/fsync)
    - UTIL_PIN_CTL Pipe Select = <pipe attached to the primary transcode>
  - Reference clock output to secondary devices
    - UTIL2_PIN_CTL Enable = 1
    - UTIL2_PIN_CTL Direction = 0 (output)

If Secondary device

- Secondary device refers to any display controller that contains only secondary transcoders.
- Secondary transcoders
  - Secondary transcoders within the secondary device are remote secondaries.
  - Remote secondaries receive frame sync from the primary transcoder in the primary device and use the primary device PLL reference.
  - Secondary PLL is programmed only after receiving reference clock from the primary device.
  - TRANS_DDI_FUNC_CTL2 Genlock Enable = 1
  - TRANS_DDI_FUNC_CTL2 Genlock Mode = 01 (Remote Secondary)
  - Set TRANS_DDI_FUNC_CTL2 [port sync mode primary select] to register field default.
  - SNPS_PHY_REF_CONTROL[Filter PLL input mux select] = 1b (genlock reference)
- Pin configuration
  - Frame sync input from primary device
    - UTIL_PIN_CTL Enable = 1
    - UTIL_PIN_CTL Direction = 1 (input)
  - Reference clock input from primary device
    - UTIL2_PIN_CTL Enable = 1
    - UTIL2_PIN_CTL Direction = 1 (input)

Audio

Audio Bios Programming Sequence

Audio Link Settings

When BIOS enables the audio link it must program both the audio controller and audio codec with the same BCLK and T-Mode settings before the link is enabled to ensure enumeration is successful.

In addition, the audio codec "Detect Frame Sync Early" value needs to be set per-project in AUD_FREQ_CTL according to the table below.

- The audio codec link setup is performed using the **AUD_FREQ_CTL** register.
  - Program "BCLK" in bits 4:3
  - Program "T-Mode" in bits 15:14
  - Program "Detect Frame sync early" in bits 12:11
  - These settings need to be programmed **before** programming the **AUDIO_PIN_BUF_CTL** register with any analog IO settings.
- The audio controller setup is performed using controller mmio register LCTL (0xc84) and EM1 (0x1000).
  - Please consult with the Audio Controller HW team for official information.

Table 1: *Audio Power Wells*

This table lists the power well containing the audio codec per-project.

Audio Power Well
PG0

Table 2: *Audio Link Settings*

The table below provides the project and stepping specific requirements for the "T-Mode" and "Detect Frame sync early" settings which must be used.

A0		B0 +	
T-Mode	Pull-In Clks	T-Mode	Pull-In Clks
8T	0	Same as A0	Same as A0

Codec Verb Table

For each codec present on the High Definition Audio codec link, a corresponding pre-defined "Codec Verb Table" must be available to System BIOS. The Codec Verb Tables are based on codec specific information (coded datasheet) and platform design specific information (schematics) and are built by System BIOS writers and platform designers. The table contains a list of 32-bit Verbs (command and data payload) to be sent to the corresponding codec over the High Definition Audio codec link.

Below is a sample High Definition Audio Codec Verb Table for a platform with 1 codec at codec address 01h.

```
;Sample HIGH DEFINITION AUDIO Codec Verb Table
;Codec Address (CAd) = 02h
;Codec Vendor: XYZ Company
;VenID DevID:
dd 12345678h
;-----------------------------------------------------
; FrontPanel_Supported? ; 1=Supported ,0=Not supported
db 01h
; # of Rear Panel Pin Complexes
dw 000Ch
; # of Front Panel Pin Complexes
dw 0002h
```

;---

; Turn on the Audio IO buffer control register to enable the 3 pin link.

Set bit 31 to 1 of AUDIO_PIN_BUF_CTL register.

Note: Set the bit 15 of register offset 0x65F10h of the Display Audio offset. Wait for the Codec to generate the wake event to the controller.

Following verbs should be send to the codec using the PIO method described in the below sections 9.1.3.

VerbTable0:

The Vendor Node ID and the Pin Node IDs may be updated per project.

Please refer to the **Node ID descriptions** page to get the node IDs of the vendor node and each supported Pin and replace the node IDs in the example below.

All the Pin widgets present in the current project should be programmed with config data as below.

Enable the third converter and Pin first (NID 02h)

0x20278101h

//

// Audio Verb Table

//

// Pin Widget 0x4 - PORT DDI-A

0x20471C10,

0x20471D00,

0x20471E56,

0x20471F18,

// Pin Widget 0x6 - PORT DDI-B

0x20671C20,

0x20671D00,

0x20671E56,

0x20671F18,

// Pin Widget 0x8 - PORT DDI-C

0x20871C30,

0x20871D00,

0x20871E56,

0x20871F18,

// Pin Widget 0xA - PORT USBC1

0x20A71C40,

0x20A71D00,

0x20A71E56,

0x20A71F18,

// Pin Widget 0xB - PORT USBC2

0x20B71C50,

0x20B71D00,

0x20B71E56,

0x20B71F18

// Pin Widget 0xC - PORT USBC3

0x20C71C60,

0x20C71D00,

0x20C71E56,

0x20C71F18

// Pin Widget 0xD - PORT USBC4

0x20D71C70,

0x20D71D00,

0x20D71E56,

0x20D71F18

// Pin Widget 0xE - PORT USBC5

0x20E71C80,

0x20E71D00,

0x20E71E56,

0x20E71F18

// Pin Widget 0xF - PORT USBC6

0x20F71C90,

0x20F71D00,

0x20F71E56,

0x20F71F18

Codec Initialization Programming Sequence

After System BIOS has determined the presence of High Definition Audio codecs, it must follow the programming sequence below to update the codec with the correct jack information specific to the platform for the High Definition Audio driver to retrieve and use later.

There are two ways to send verbs to and receive response data from codecs over the High Definition Audio codec link: using CORB/RIRB (Command Output Ring Buffer / Response Input Ring Buffer) or using the Immediate Command/Immediate Response register pair. The sequence below uses the latter which does not require the availability of a memory buffer.

- System BIOS should ensure that the High Definition Audio HDBAR D27:F0:10-17h contains a valid address value and is enabled by setting D27:F0:04h[1].
- System BIOS must ensure program as mentioned in section 9.6, and then the Controller Reset# bit of Global Control register in memory-mapped space (HDBAR+08h[0]) is set to 1b and read back as 1b.
- When clearing this bit and setting it afterward, System BIOS must ensure that minimum link timing requirements (minimum RESET# assertion time, etc.) are met.
- **Note:** To initialize the codec System BIOS should set the display mmio register 0x65F10 bit 15 to 1b. This bit needs to be set after the controller is brought out of reset.
- The codec requires **50 ms** to come out of reset prior to subsequent operations - System BIOS should wait for the Controller to detect the wake event and recognize the Codec.
- System BIOS can poll on display mmio register 0x65F10 bit 14 "Codec Sleep State" for value 0b (codec awake) with a timeout of **50 ms** before proceeding to additional steps.

For each High Definition Audio codec present as indicated by HDBAR + 0Eh[3:0], System BIOS should perform the codec initialization as described below:

1. Read the VendorID/DeviceID pair from the attached codec.

- Verify that the ICB bit, HDBAR + 68h[0], is 0.
- Write verb 200F0000h (dword) to the IC register, HDBAR + 60h, where: '2' (bits 31:28) represents the codec address (CAd).
- Program HDBAR + 68h[1:0] to 11b to send the verb to the codec.
- Poll the ICB bit, HDBAR+68h[0] until it returns 0 indicating the verb has been sent to the codec. System BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a **50 us** timeout period.
- If HDBAR + 68h[1] = 1b indicating the response data from the codec is now valid, read HDBAR + 64h; the data is the VID/DID value returned by the codec.

2. Check against internal list to determine if there is a stored verb table which matches the CAd/VID/DID information.

Steps 1 and 2 are System BIOS implementation-specific steps and can be done in different ways. If a System BIOS has prior knowledge of a fixed platform/codec combination (e.g., for a System BIOS having 3 stored verb tables for 3 known codecs at known codec addresses on a known platform), a simple pre-

defined codec-to-table matching can be used and steps 1 and 2 can be eliminated. For a System BIOS to support multiple codec/platform combinations, an internal match-list might be needed to match a platform/codec combination to a codec verb table.

3. If there is a match, send the entire list of verbs in the matching verb table one by one to the codec.

- Verify the ICB bit, HDBAR + 68h[0] is 0.
- Write the next verb (dword) in the table to HDBAR + 60h.
- Program HDBAR + 68h[1:0] to 11b to send the verb to codec.
- Poll the ICB bit, HDBAR + 68h[0] until it returns 0 indicating the verb has been sent to the codec. System BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a **50 us** timeout period.
- Repeat the steps until all the verbs in the table have been sent.

Some verbs in the table may be dependent on certain platform-specific conditions. For example, for the sample table above, the verbs for Pin Complex 7 and 8 (NID=14,16 respectively) should be sent only if the Front Panel Jacks are present and connected on the platform, which may be indicated by a software flag that is controlled by a certain GPIO pin.

Audio Programming Sequence for Link Wakeup

The following audio programming sequences are to be used for preventing the Unsolicited responses when 3 pin link is awake.

Display Audio codec generates a wake event whenever the power well (PGx - power well in which Display Audio codec HW resides) is powered up. If the link is already running, this wake event is considered as unsolicited response by audio controller in PCH. This may sometimes be considered as unnecessary URs. To avoid such URs following programming should be followed by SW. This sequence assumes communication between Audio and GFX drivers without HW to indicate Audio codec power well status.

Power down sequence:

1. Unplug event
2. PD goes low
3. PG2 low
4. Audio link should go low (there should be a communication between GFX driver and Audio Driver to turn off the link)

    a) To turn off iDisp-A link:

    b) Power off iDisp-A codec. Follow the PW2 turn off sequence.

    c) In HD Audio Controller (PCH) Program LCTL1.SPA = 0.

    d) Wait for LCTL1.CPA = 0 to indicate the link has clock stopped.

    e) Clear bit 31 to 0 of AUDIO_PIN_BUF_CTL register.

Power up sequence:

1. Plug event
2. PG2 goes high
3. Audio link to be enabled (there should be a communication between GFX driver and Audio Driver to turn on the link)
    a) To turn on iDisp-A link:
    b) In HD Audio Controller (PCH) Program LCTL1.SPA = 1.
    c) Wait for LCTL1.CPA = 1 to indicate the link has clock running.
    d) Set bit 31 to 1 of AUDIO_PIN_BUF_CTL register.
4. Check WAKESTS[2] = 1 to indicate the codec wake up occurs
5. PD bit set
6. Codec awake. Continue with codec init.

Audio Programming Sequence

The following HDMI and DisplayPort audio programming sequences are to be used when enabling or disabling audio or temporarily disabling audio during a display mode set.

- The audio codec and audio controller disable sequences must be followed prior to disabling the transcoder or port in a display mode set.
- The audio codec and controller enable sequences can be followed after the transcoder is enabled and the port is enabled and completed link training (not sending training or idle patterns if DisplayPort).
- The audio controller and audio codec sequences may be done in parallel or serial. In general, the change in ELDV/PD in the codec sequence will generate an unsolicited response to the audio controller driver to indicate that the controller sequence should start, but other mechanisms may be used. SW should make sure to set the Inactive (IA) bit to 0 before setting PD to 1.

In addition, the Audio Bandwidth Checks page provides algorithms and calculations that must be followed during the enabling sequence for audio, to ensure there is adequate bandwidth available for audio transmission.

Disabling Audio

Audio codec disable sequence:

1. Disable AUD_PIN_ELD_CP_VLD bit-field "CP_Ready" (bit 1, 5, 9) to ”0".
2. Disable timestamps
    a) Set AUD_CONFIG bit-field "N_value_index" (bit 29) to "0" for HDMI or "1" for DisplayPort.
    b) Set AUD_CONFIG bit-field "N_programming_enable" (bit 28) to "1".
    c) Set AUD_CONFIG bit-field "Upper_N_value" and "Lower_N_value" (bits 27:20, 15:4) to all "0"s.
3. Disable ELDV and ELD buffer

a) Set AUD_PIN_ELD_CP_VLD bit-field "ELD_valid" (bit 0, 4, or 8 based on which port is used) to "0".

4. Disable sample fabrication
    a) Set AUD_MISC_CTRL bit-field "Sample_Fabrication_EN" (bit 2) to "0".
5. Wait for 2 vertical blanks
6. Optional: Disable audio PD (Presence Detect)
    a) Software may choose to skip this in order to keep PD enabled during a resolution switch.
    b) Set AUD_PIN_ELD_CP_VLD bit-field "Audio_Inactive" (bit 3, 7, or 11) to "1". SW does not need to set this bit to enable Inactive bit.
    c) Set AUD_PIN_ELD_CP_VLD bit-field "Audio_Output_Enable" (bit 2, 6, or 10) to "0".

Audio controller disable sequence:

- Program Stream ID to 0 - Verb ID 706
- Disable audio info frames transmission - Verb ID 732
- Disable Digen - Verb ID 70D
- Program the codec to D3 state if needed.
- Audio driver may stop the audio controller DMA engine at this point if needed, but not required.

Enabling Audio

Pre-Enabling Notes:

The following tables (if present) list any project specific sequence requirements to be handled before initiating the audio codec enable sequence.

Programming Notes
• When using DP 2.0 128b/132b encoding, this mode must be enabled in its respective transcoder registers before enabling audio Presece Detect bits.

Audio codec enable sequence:

1. Enable audio Presence Detect
    a) Set AUD_PIN_ELD_CP_VLD bit-field "Audio_Inactive" (bit 3, 7, or 11) to "0".
    b) Set AUD_PIN_ELD_CP_VLD bit-field "Audio_Output_Enable" (bit 2, 6, or 10) to "1".
2. Wait for 1 vertical blank
3. Enable sample fabrication if this feature is needed
    a) Set AUD_MISC_CTRL bit-field "Sample_Fabrication_EN" (bit 2) to "1"
4. Load ELD buffer and Enable ELDV
    a) Set AUD_PIN_ELD_CP_VLD bit-field "ELD_valid" (bit 0, 4, or 8 based on which port is used) to "1".
5. Enable timestamps
    a) Set AUD_CONFIG bit-field "N_value_index" (bit 29) to "0" for HDMI or "1" for DisplayPort.

b) If a custom N value is not needed (default case)
   i. Set AUD_CONFIG bit-field "N_programming_enable" (bit 28) to "0".
c) If a custom N value is needed
   i. Set AUD_CONFIG bit-field "Upper_N_value" and "Lower_N_value" (bits 27:20, 15:4) to the custom N value.
   ii. Set AUD_CONFIG bit-field "N_programming_enable" (bit 28) to "1".

6. Enable AUD_PIN_ELD_CP_VLD bit-field "CP_Ready" (bit 1, 5, 9) to “1”.

Audio controller enable sequence:

- Program the codec to D0 state if in D3 state.
- Program Stream ID to non zero - Verb ID 706
- Enable audio info frames transmission - Verb ID 732
- Enable Digen - Verb ID 70D
- If audio controller DMA engine is stopped, audio driver can start the DMA engine at this point.

Audio Hblank early enable sequence:

- Bits 20:18 of the AUD_CONFIG_BE register have the Hblank early enable for each pipe for DP Audio. Bits 27:24 of te AUD_CONFIG_BE register have the Hblank early enable for each pipe for DP Audio. When DP Audio is enabled on that pipe, driver must set these bits when Audio presence detect is set for each of these pipe regardless of the video resolution. This programming is not needed for HDMI Audio.
- For the following cases of VDSC, 4K, 5K and 8K resolutions the following programming sequence needs to be performed by the driver before setting the Audio Presence detect of pipe for DP audio. This is not required for HDMI Audio.

  A. Driver must Program "Hblank Early Enable for Pipe X" = 1b always

  B. Determine required pixel clocks between hblank_early rise and hblank rise

  link_clks_available = {ROUDNDOWN[((h_total - h_active) * (link_clk / pixel_clk) - 28)]}

  link_clks_required = {ROUNDUP[(192000 / (refresh_rate * v_total))]} * ((48/lanes) + 2)

  If link_clks_available > link_clks_required

  - For Step C, use hblank_delta = 32

  else

  - For Step C, use hblank_delta = {ROUNDUP[(((5 / link_clk) + (5 / cdclk)) * pixel_clk)]}

  C. Determine hblank_early programming

  tu_data = (pixel_clk * vdsc_bpp * 8) / (link_clk * lanes * fec_coeff)

tu_line = ((h_active * link_clk * fec_coeff)/(64 * pixel_clk))

link_clks_active = (tu_line -1)*64 + tu_data

hblank_rise =((link_clks_active + 6*{ROUNDDOWN[(link_clks_active / 250)]} + 4) * (pixel_clk/link_clk))

hblank_early_prog = {ROUNDUP[h_active - hblank_rise + hblank_delta]} (from Step B)

if (hblank_early_prog < 32) then

- Program "*Hblank_start count for Pipe X*" to select value 32

elsif (hblank_early_prog > 32 < 64) then

- Program "*Hblank_start count for Pipe X*" to select value 64

elsif (hblank_early_prog > 64 < 96) then

- Program "*Hblank_start count for Pipe X*" to select value 96

elsif (hblank_early_prog > 96) then

- Program "*Hblank_start count for Pipe X*" to select value 128

D. Calculate samples per line required

samples_room = {ROUNDDOWN[(((h_total - h_active) * (link_clk / pixel_clk) - 12) / ((48/lanes) + 2))]}

if (samples_room < 3) then

- Program "*Number of samples per line for Pipe X*" to select value == samples_room

else

- Program "*Number of samples per line for Pipe X*" to select value == default value (00b = "All Samples available in buffer")

The following variables are referred to in the equations:

Variable	Units	Note
h_total	pixels	Total horizontal pixels
h_active	pixels	Active horizontal pixels
v_total	pixels	Total vertical pixels
refresh_rate	Hz	Screen refresh rate

input_bpp	bits per pixel	Bits per color "bpc" setting * 8
vdsc_bpp	bits per pixel	If not using compression, vdsc_bpp = input_bpp
fec_coeff	constant	Currently 0.972261. See "**Transcoder MN Values**".
lanes	lanes	1,2, or 4
link_clk	MHz	Link Rate / 10 (i.e. 162,270,540,810 MHz). See "**Clocks**" page per project.
pixel_clk	MHz	h_total * v_total * refresh_rate
cdclk	MHz	See "**Clocks**" page per project.
link_clks_available	link clocks	Link clocks within hblank available for audio use
link_clks_required	link clocks	Link clocks within hblank needed for audio each line
hblank_delta	pixel clocks	Pixel clocks to maintain between hblank_early_rise and hblank_rise
tu_data	link clocks	Link clocks per TU used for pixel data
tu_line	TUs	TUs required per horizontal line
link_clks_active	link clocks	Link clocks required to send active pixels per horizontal line
hblank_rise	pixel clocks	Pixel clock at which hblank will rise, accounting for FEC
hblank_early_prog	pixel clocks	Pixel clocks required to pull hblank_early back far enough from the end of h_active, to ensure there are hblank_delta pixel clocks before hblank rise.
samples_room	audio samples	Maximum number of samples which can fit within hblank, rounded down.

Audio Bandwidth Checks

This page provides the algorithms and calculations needed to ensure there is sufficient bandwidth available with the enabled video configuration for the transmission of audio.

These checks are broken down into DisplayPort and HDMI specific checks, and each section lists the criteria which determine when each check must be performed.

DisplayPort

The following section lists any required checks for audio bandwidth that are applicable in DisplayPort mode.

DP1 MST with DSC Enabled:

The following algorithm and calculation needs to be applied when DP1 MST + DSC + Audio is enabled.

Algorithm

1. Determine max_output_bpp that is supported per the current lanes, link rate, and video mode.
   - Bspec provides this calculation on the DSC page under "Programming Considerations and Restrictions"
2. GFX driver prunes all audio modes from the ELD except minimum audio (48KHz 2ch).
3. Check if minimum audio (48KHz 2ch) can work with max_output_bpp from step 1.
   - Use Calculation 1 below, with output_bpp = max_output_bpp, aud_freq_khz = 48, aud_chan = 2, and rest of video settings.
     - IF YES -> Proceed to Step 4
     - IF NO -> Disable DSC
4. Determine the min_output_bpp required to support minimum audio (48KHz 2ch).
   - For each vdsc_bpp in [8 to max_output_bpp]:
     - Use Calculation 1 below, with output_bpp = vdsc_bpp[i], aud_freq_khz = 48, aud_chan = 2, and rest of video settings.
     - If audio_supported = True, min_output_bpp = vdsc_bpp[i], loop can be stopped.
     - If audio_supported = False, continue loop with next vdsc_bpp[i+1].
5. Enable DSC using at least min_output_bpp from Step 4.

Calculation 1

Inputs:

link_rate_ghz

lanes

pixel_clk_mhz

output_bpp

hactive_pixels

hblank_pixels

aud_freq_khz

aud_chan

Constants:

mtp_size_clks = 64

link_overhead = 0.03

hblank_bytes_avail_overhead = 48

hblank_bytes_req_overhead = 56

General Calc:

```
link_clk_mhz                    = (link_rate_ghz / 10) * 1000
line_freq_khz                   = (pixel_clk_mhz / (hactive_pixels + hblank_pixels)) * 1000
pixel_bw_gbps                   = (pixel_clk_mhz * (output_bpp / 8)) / 1000
link_bw_gbps                    = ((link_clk_mhz * lanes) * (1-link_overhead)) / 1000
mtp_size_ns                     = (mtp_size_clks / link_clk_mhz) * 1000
hblank_size_ns                  = (hblank_pixels / pixel_clk_mhz) * 1000
```

vc_slots = CEIL[64 * (pixel_bw_gbps / link_bw_gbps)]

```
aud_samples_per_line            = CEIL[ aud_freq_khz / line_freq_khz ] + 1
```

Worst Case Hblank vs Audio BW:

hblank_reduced_ns = hblank_size_ns - (((64 - vc_slots) / link_clk_mhz)*1000)

hblank_slots_full_mtps = FLOOR[hblank_reduced_ns / mtp_size_ns] * vc_slots

hblank_slots_partial_mtps = MIN[CEIL[(MOD[hblank_reduced_ns / mtp_size_ns] * link_clk_mhz) / 1000] , vc_slots]

hblank_slots = hblank_slots_full_mtps + hblank_slots_partial_mtps

```
hblank_bytes_available          = (hblank_slots * lanes) - hblank_bytes_avail_overhead
IF (aud_chan > 2):
      hblank_bytes_required     = (aud_samples_per_line*10 + 4)*4 + hblank_bytes_req_overhead
ELSE:
      hblank_bytes_required     = (CEIL[aud_samples_per_line/2]*5 + 4)*4
      + hblank_bytes_req_overhead
```

audio_supported = IF hblank_bytes_available > hblank_bytes_required: **YES** ELSE: **NO**

DP2 SDP Splitting and Audio Support:

The following algorithm and calculation needs to be applied when DP2 + Audio is enabled:

Algorithm

1. Always enable SDP Splitting - Program AUD_DP_2DOT0_CTL "Enable SDP Split" bit 31 = 1b for each transcoder that will have DP2 + Audio enabled.

2. GFX driver retrieves all audio modes from the ELD, and makes a list of supported_freq and supported_chan.

3. Determine the maximum audio BW that can be used with output_bpp == 8, working downward from highest audio frequency and highest audio channels.

- For each audio frequency in supported_freq [max value .. min value] and supported_chan = max value
    - Use Calculation 1 below with output_bpp = 8, aud_freq_khz = supported_freq[i], aud_chan = [max value]
    - If audio_supported = True, use the current values of output_bpp, aud_freq_khz, and aud_chan in step #4 and loop can be stopped.
    - If audio_supported = False, continue loop with next supported_freq [next highest value].
- If audio_support = False for supported_freq [min value] and supported_chan = max value, decrease supported_chan to the next highest value, and repeat the supported_freq loop.

4. If even minimum audio (48KHz 2ch) is not supported with output_bpp == 8 as per step #3, increase output_bpp to determine the min_output_bpp required to support minimum audio (48KHz 2ch).

- Determine max_output_bpp that is supported per the current lanes, link rate, and video mode.
    - Bspec provides this calculation on the Display 11.5+ DSC page under “Programming Considerations and Restrictions”
- Determine the min_output_bpp required to support minimum audio (48KHz 2ch).
    - For each vdsc_bpp in [8 to max_output_bpp]:
        - Use Calculation 1 below, with output_bpp = vdsc_bpp[i], aud_freq_khz = 48, aud_chan = 2, and rest of video settings.
        - If audio_supported = True, min_output_bpp = vdsc_bpp[i], loop can be stopped.
        - If audio_supported = False, continue loop with next vdsc_bpp[i+1].
- Enable DSC using at least min_output_bpp determined above.

5. GFX driver prunes all audio modes from the ELD which are greater than the aud_freq_khz and aud_chan determined from step #3 or #4.

Calculation 1 (SDP Splitting Math)

Inputs:

link_rate_ghz

lanes

pixel_clk_mhz

output_bpp

hactive_pixels

hblank_pixels

aud_freq_khz

aud_chan

data_M, data_N (Calculate as per the Bspec page "Transcoder MN Values")

Constants:

hblank_bytes_avail_overhead = 64 bytes

hblank_bytes_req_overhead = 80 bytes

General Calc:

```
mtp_clks_per_slot               = ( 4 / lanes )
mtp_size_clks          = 64 * mtp_clks_per_slot
link_clk_mhz                    = (link_rate_ghz / 32) * 1000
line_freq_khz                   = (pixel_clk_mhz / (hactive_pixels + hblank_pixels)) * 1000
mtp_size_ns                     = (mtp_size_clks / link_clk_mhz) * 1000
hblank_size_ns                  = (hblank_pixels / pixel_clk_mhz) * 1000
vc_slots                        = CEIL[ data_M / data_N ] * 64
aud_samples_per_line            = CEIL[ aud_freq_khz / line_freq_khz ] + 1
lines_per_audio_sample          = MAX[ 1, FLOOR[ line_freq_khz / aud_freq_khz ] ]
```

Worst Case Hblank vs Audio BW:

```
mtps_in_hblank                  = hblank_size_ns / mtp_size_ns
hblank_slots                    = FLOOR[ mtps_in_hblank * mtp_size_clks * (data_M / data_N)
]
hblank_bytes_available          = ((hblank_slots * lanes * 4) - hblank_bytes_avail_overhead) *
lines_per_audio_sample
IF (aud_chan > 2):
      hblank_bytes_required     = ( CEIL[ aud_samples_per_line * 10 + 2) / 4 ] + 2) * 16
      + hblank_bytes_req_overhead
ELSE:
      hblank_bytes_required     = ( CEIL[ ( CEIL[ aud_samples_per_line / 2 ] * 5 + 2) / 4
      ] + 2) * 16 + hblank_bytes_req_overhead
```

audio_supported = IF hblank_bytes_available > hblank_bytes_required: **YES** ELSE: **NO**

HDMI

The following section lists any required checks for audio bandwidth that are applicable in HDMI mode.

Currently no audio bandwidth checks are required.

Audio Keep Alive Programming Sequence

The silent stream feature has been modified so that control of silent stream enabling depends on the digital converter verb 73Eh bit 7 "Keep Alive", as well as the existing "Sample Fabrication EN bit" field in the AUD_MISC_CTRL mmio register. This standardizes the silent stream implementation.

When Keep Alive is enabled, codec HW will automatically switch between generated silent stream data and real stream data as needed. During each frame sync, if real stream data with a valid stream ID is received from the controller it will be transmitted as normal, otherwise silent stream data will be generated and sent. In this way SW can enable Keep Alive after the audio mode change sequence is complete (as described below under "Audio Mode Change Sequence") to allow HW to generate silent stream data as needed, or SW can turn Keep Alive on and off directly based on whether stream playback is being started or stopped.

When the 3-pin link is turned off (BCLK off) during D3 sleep states, if Keep Alive is enabled the codec HW will continue generating silent stream and timestamp data using the display CDCLK instead of BCLK. For this purpose, the Gfx driver must program an M and N value based on the current CDCLK or when the CDCLK is being changed as described below under "CDCLK Change Sequence". Timestamp generation from CDCLK is not supported when CDCLK is running at the reference clock frequency.

Audio Mode Change Sequence

Keep alive is disabled by default in HW. For the first and any subsequent audio mode changes, the following sequence should be used:

1. Disable Keep Alive - program Keep Alive (bit 7) = 0b in the 73Eh verb (**digital converter verbs**).
2. Wait for 100us.
3. Perform standard mode change sequence.
4. Wait for 100us.
5. Enable Keep Alive - program Keep Alive (bit 7) = 1b in the 73Eh verb (**digital converter verbs**).

CDCLK Change Sequence

When the CDCLK is to be changed either through clock crawling or through standard CDCLK switching, the steps below need to be performed by the Gfx driver.

Please refer to the CDCLK tables within the main "**Clocks**" Bspec page per-project for the correct **AUD_TS_CDCLK_M** and **AUD_TS_CDCLK_N** values, based on the final CDCLK frequency that will be used after switching completes.

1. Before the CDCLK change starts, clear the "Enable Timestamp Generation During Link Off" bit (31) in **AUD_TS_CDCLK_M**.

2. Wait for the CDCLK change to be completed as per the steps in "Sequences for Changing CD Clock Frequency".
3. Program the "N Value" bits (23:0) in **AUD_TS_CDCLK_N** based on the final CDCLK value.
4. Program the "M Value" bits (23:0) in **AUD_TS_CDCLK_M** based on the final CDCLK value.
5. Set the "Enable Timestamp Generation During Link Off" bit (31) in **AUD_TS_CDCLK_M**.

Audio Configuration

Registers
AUD_CONFIG
AUD_CONFIG_2
AUD_CONFIG_BE
AUD_CONFIG_BE_2
AUD_TS_CDCLK_M
AUD_TS_CDCLK_N
AUD_MISC_CTRL
AUD_VID_DID
AUD_RID
AUD_M_CTS_ENABLE
Audio Power State Format
AUD_PWRST
AUD_EDID_DATA
AUD_FREQ_CNTRL
AUD_INFOFR
AUD_PIN_PIPE_CONN_ENTRY_LNGTH
AUD_PIPE_CONN_SEL_CTRL
AUD_DIP_ELD_CTRL_ST
AUD_PIN_ELD_CP_VLD

DisplayPort Transport

There is one instance of these registers per each DDI.

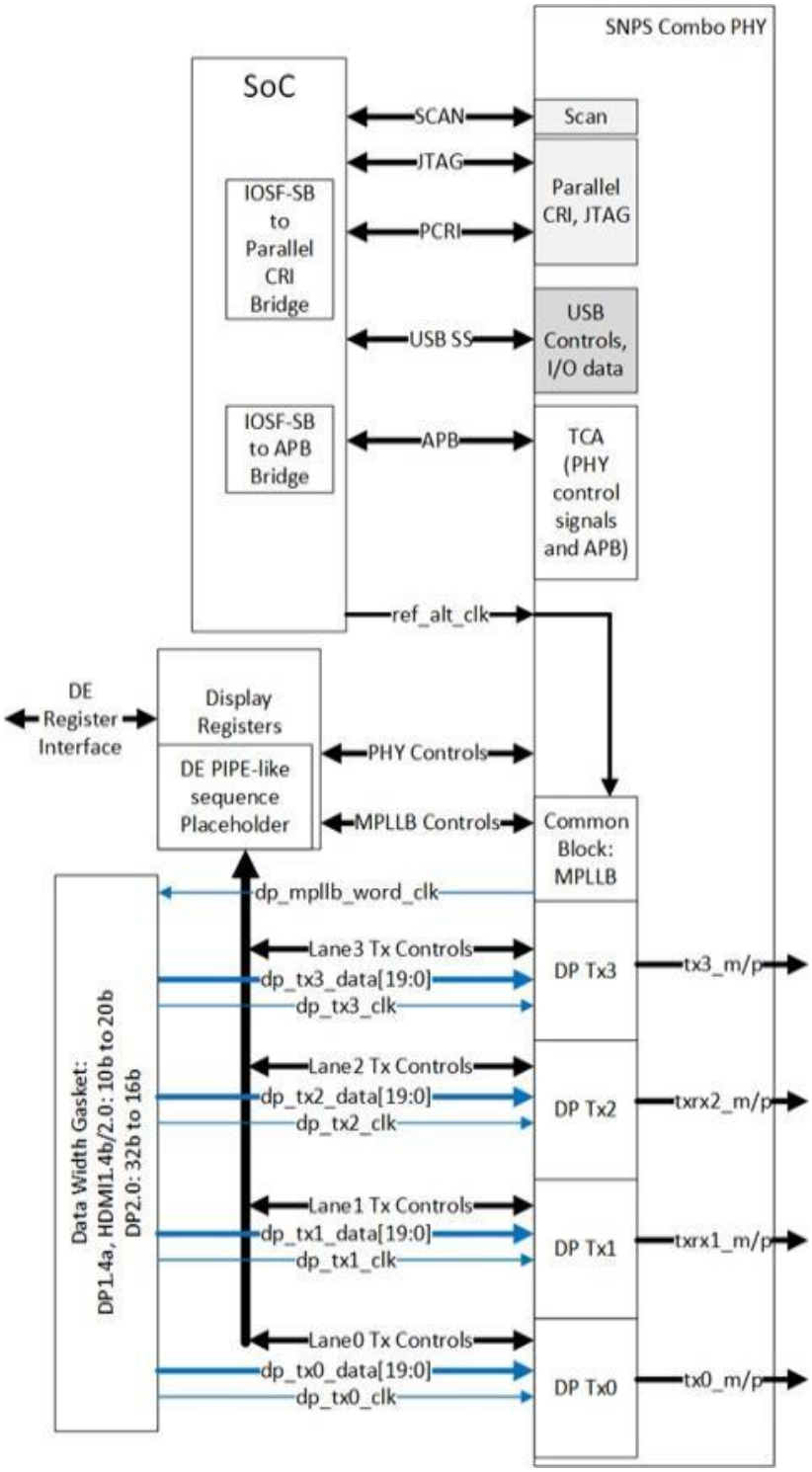
DP_TP_CTL

DP_TP_STATUS

Digital Display Interface

PHY DDI Buffer

Block diagram



Keypoints

- This PHY is designed to connect to a USB Type-C connector and to support DP alt-mode.
- This PHY also drives HDMI as a low-voltage AC coupled signals.
- MPLLB will always be used for HDMI and DP protocols.
- All TX lanes must be disabled for any PHY mode transitions.
- TX lanes must go through P3 power state for every mode transition.
- P3 is the designated lowest power state.
- PHY must be in P0 power state (a.k.a mission mode power state) for Vswing and Pre-emphasis adjustment.
- Changing number of lanes also changes which lane is the “primary” for de-skew. Hence we must assert dp_txX_reset when we change the number of lanes.
- PHY will ignore DE reset assertion on TX lanes not owned by display.

Registers

SNPS_PHY_REF_CONTROL

SNPS_PHY_MPLLB_CP

SNPS_PHY_MPLLB_DIV

SNPS_PHY_MPLLB_DIV2

SNPS_PHY_MPLLB_FRACN1

SNPS_PHY_MPLLB_FRACN2

SNPS_PHY_MPLLB_SSCEN

SNPS_PHY_MPLLB_SSCSTEP

SNPS_PHY_MPLLB_STATUS

SNPS_PHY_AUX_CNFG

SNPS_PHY_I2C_CNFG

SNPS_PHY_TX_COMMON

SNPS_PHY_TX_EQ

SNPS_PHY_TX_REQ

SNPS_PHY_TYPEC_STATUS

Voltage Swing Programming Sequence

This sequence is used to setup the voltage swing before enabling the DDI, as well as for changing the voltage during DisplayPort link training.

1. Program the per lane registers SNPS_PHY_TX_EQ to set main tap, pre-cursor tap and post-cursor tap coefficients with values from the tables below.

TX Equalization Settings for Non-UHBR rates

Voltage Swing	Pre-emphasis	Non-transition mV diff p-p	Transition mV diff p-p	Pre-emphasis dB	Voltage Level Swing Select dp_tx_eq_main[5:0]	Pre Cursor dp_tx_eq_pre[5:0]	Post Cursor dp_tx_eq_post[5:0]
Level 0	0	400	400	0	25	0	0
Level 0	1	400	600	3.5	32	0	6
Level 0	2	400	800	6	35	0	10
Level 0	3	400	1200	9.5	43	0	17
Level 1	0	600	600	0	35	0	0
Level 1	1	600	800	3.5	45	0	8
Level 1	2	600	1200	6	48	0	14
Level 2	0	800	800	0	47	0	0
Level 2	1	800	1200	3.5	55	0	7
Level 3 (HDMI default)	0	1200	1200	0	62	0	0

Tx Equalization Settings for DP2.0 UHBR

In the table below:

V1 = voltage level when pre-cursor is enabled and post-cursor is enabled. These values are to be used during normal link training flow.

V2 = voltage level when pre-cursor is disabled and post-cursor is enabled. These values are used in DP2.x compliance testing.

V3 = voltage level when pre-cursor is enabled and post-cursor is disabled. These values are used in DP2.x compliance testing.

V0/V15 = voltage level when pre-cursor is disabled and post-cursor is disabled. These values are used in DP2.x compliance testing.

Preset#	EQ	Pre-shoot (dB)	De-emphasis (dB)	Informative Filter Coefficients			Pre Cursor dp_tx_eq_pre[5:0]	Voltage Level Swing Select dp_tx_eq_main[5:0]	Post Cursor[5:0] dp_tx_eq_post[5:0]
				C-1	C0	C+1			
P0	V1	0	0	0	1	0	0	62	0
	V2	0	0	0	1	0	0	62	0
	V3	0	0	0	1	0	0	62	0
	V0	0	0	0	1	0	0	62	0
P1	V1	0	-1.9	0	0.9	-0.1	0	55	7
	V2	0	-1.9	0	0.9	-0.1	0	55	7
	V3	0	0	0	1	0	0	62	0
P2	V1	0	-3.6	0	0.83	-	0	50	12

Preset#	EQ	Pre-shoot (dB)	De-emphasis (dB)	Informative Filter Coefficients			Pre Cursor dp_tx_eq_pre[5:0]	Voltage Level Swing Select dp_tx_eq_main[5:0]	Post Cursor[5:0] dp_tx_eq_post[5:0]
						0.17			
	V2	0	-3.6	0	0.83	-0.17	0	50	12
	V3	0	0	0	1	0	0	62	0
P3	V1	0	-5.0	0	0.78	-0.22	0	48	14
	V2	0	-5.0	0	0.78	-0.22	0	48	14
	V3	0	0	0	1	0	0	62	0
P4	V1	0	-8.4	0	0.69	-0.31	0	38	18
	V2	0	-8.4	0	0.69	-0.31	0	38	18
	V3	0	0	0	1	0	0	56	0
P5	V1	0.9	0	-0.05	0.95	0	3	59	0
	V2	0	0	0	1	0	0	62	0
	V3	0.9	0	-0.05	0.95	0	3	59	0
P6	V1	1.1	-1.9	-0.05	0.86	-0.09	3	53	6
	V2	0	-1.9	0	0.91	-0.09	0	56	6
	V3	1.1	0	-0.05	0.95	0	3	59	0
P7	V1	1.4	-3.8	-0.05	0.79	-0.16	3	48	11
	V2	0	-3.8	0	0.84	-0.16	0	51	11
	V3	1.4	0	-0.05	0.95	0	3	59	0
P8	V1	1.7	-5.8	-0.05	0.73	-0.22	5	42	15
	V2	0	-5.8	0	0.78	-0.22	0	47	15
	V3	1.7	0	-0.05	0.95	0	5	57	0
P9	V1	2.1	-8.0	-0.05	0.68	-0.27	5	41	16
	V2	0	-8.0	0	0.73	-0.27	0	46	16
	V3	2.1	0	-0.05	0.95	0	5	57	0

Preset#	EQ	Pre-shoot (dB)	De-emphasis (dB)	Informative Filter Coefficients			Pre Cursor dp_tx_eq_pre[5:0]	Voltage Level Swing Select dp_tx_eq_main[5:0]	Post Cursor[5:0] dp_tx_eq_post[5:0]
P10	V1	1.7	0	-0.09	0.91	0	6	56	0
	V2	0	0	0	1	0	0	62	0
	V3	1.7	0	-0.09	0.91	0	6	56	0
P11	V1	2.2	-2.2	-0.09	0.82	-0.09	7	48	7
	V2	0	-2.2	0	0.91	-0.09	0	55	7
	V3	2.2	0	-0.09	0.91	0	7	55	0
P12	V1	2.5	-3.6	-0.09	0.77	-0.14	7	46	9
	V2	0	-3.6	0	0.86	-0.14	0	53	9
	V3	2.5	0	-0.09	0.91	0	7	55	0
P13	V1	3.4	-6.7	-0.09	0.69	-0.22	8	40	14
	V2	0	-6.7	0	0.78	-0.22	0	48	14
	V3	3.4	0	-0.09	0.91	0	8	54	0
P14	V1	3.6	0	-0.17	0.83	0	14	48	0
	V2	0	0	0	1	0	0	62	0
	V3	3.6	0	-0.17	0.83	0	14	48	0
P15	V1	1.7	-1.7	-0.05	0.55	-0.05	5	43	5
	V2	0	-1.7	0	0.60	-0.05	0	48	5
	V3	1.7	0	-0.05	0.60	0	5	48	0
	V15	0	0	0	0.55	0	0	43	0

Power Enabling

The Aux IO power request is used by hardware to power up the Aux PHY (de-assert the PWDNB signal). There is no ack from the PHY, so the power state (ack) is tied to the enable so it will immediately ack.

- Delay requirement to wait for power up is 600us, covered in the DDI Aux Channel sequence
- Aux IO power is required to be enabled only for using the DP Aux channel, and not necessary to be enabled for using the main link.

The PLL power enable (request) and DDI IO power request are unused by hardware. For these, the power state (ack) is tied to the enable so it will immediately ack, but the programming sequences continue to have power enable programming for backwards compatibility.

DDI AUX Channel

DDI_AUX_CTL

DDI_AUX_DATA

AUX IO Power

Each DisplayPort Aux channel has an Aux IO Power Request. If an Aux channel will not be used, it does not need to be powered up.

PSR spontaneously sends Aux transactions. If PSR is enabled on a port, then the associated Aux IO must be kept powered up.

For Type-C, Aux IO power use has extra requirements. Refer to Type-C Aux power requirements section under Type-C programming page.

AUX IO Power Enabling

1. Set **PWR_WELL_CTL_AUX** Aux IO Power Request to 1b.
    - There are two sets of PWR_WELL_CTL_AUX registers for software use. It is expected that BIOS uses PWR_WELL_CTL_AUX1 and driver uses PWR_WELL_CTL_AUX2.
2. Wait for 600 uS
3. Execute AUX functional sequences

AUX IO Power Disabling

1. Clear **PWR_WELL_CTL_AUX**Aux IO Power Request to 0b.
2. Wait for 10us. Do not poll for the power well to disable. Other clients may be keeping it enabled.

AUX programming sequence

A general purpose AUX functional programming sequence is provided below.

AUX Functional Sequence

Step	Description	Register	Notes
1	Display must already be initialized.		Power well1 enabled cdclk enabled
1a	Aux power enabled		IO powered up as needed for Aux on this project
1b	Power well containing Aux logic powered up	PWR_WELL_CTL	High level power well partitioning shown in display overview diagram
2	Disable PSR1/SRD, PSR2 and GTC if they are enabled on this DDI. Disable DC5 and DC6.	DC_STATE_EN	Power wells will disable automatically for DC5 or DC6. PSR and GTC use Aux spontaneously.
3	Program AUX data registers.	DDI_AUX_DATA_*_[0-4]	
4	Program control to configure AUX and START transaction.	DDI_AUX_CTL_*	Timeout timer value must be at least 600us. To accommodate LT-tunable PHY Repeater AUX delay, AUX Reply Timeout must be programmed to the maximum value. Timer values: 0: 400 us 1: 600 us 2: 800 us 3: 4000 us START trigger: DDI_AUX_CTL_*[31]='1'
5	Wait for AUX transaction complete.		AUX Transaction complete interrupt if set OR when DDI_AUX_CTL_*[31:30] = '01'.
6	Check that receive data has no errors	DDI_AUX_CTL_*[25]	If set: write a '1' to clear this bit and skip reading AUX data registers.
7	Read AUX data register	DDI_AUX_DATA_*_[0-4]	Condition: Aux Channel Control Register Send/Busy bit is NOT asserted
8	Clear status flags	DDI_AUX_CTL_*[30]	Transaction done status

There is a field in DDI_AUX_CTL that must be programmed for the type C ports to select if the Aux transaction will go to thunderbolt

DDI FEC

Reed-Solomon code Forward Error Correction (FEC) function RS (254, 250), with a symbol size of 10 bits is capable of correcting up to two RS symbol errors per FEC block. Display controller when plugged to an

FEC-capable DPRX and anticipates enabling FEC encoding sets the FEC_READY bit in the FEC_CONFIGURATION register (DPCD Address 00120h, bit 0) to 1 before initiating link training. Display controller needs to ensure completion of link training before starting FEC encoding. After link training is complete, display controller, if it needs to enable FEC encoding, shall send an FEC_DECODE_EN sequence to indicate the start of FEC encoding. This prompts DPRX to enable FEC decoding.

When DSC is enabled, FEC shall also be enabled.

DP Support of FEC

DP Support
FEC is supported with DP SST and MST transport modes at all lane widths.

eDP Support of FEC

eDP Support	FEC without PSR	FEC with PSR1	FEC with PSR2
FEC is supported with eDP and all lane widths. PSR restrictions apply.	Yes	Yes	**No**

Global Time Code (GTC)

Global Time Code

Top Level GTC

GTC_CTL

GTC_DDA_M

GTC_DDA_N

GTC_LIVE

GTC Interrupt Bit Definition

GTC_IMR

GTC_IIR

DDI Level GTC

GTC_PORT_CTL

GTC_PORT_TX_CURR

GTC_PORT_TX_PREV

GTC_PORT_MISC

GTC Target Frequency Selection

For GTC top level logic, CDCLK is taken as an input and scaled to a "target frequency" which has a period which is an exact multiple of 0.5ns. This period is also known as the accumulator increment.

Once a target frequency + accumulator increment is selected, an M and N value can be picked and fine-tuned to achieve the scaling from CDCLK to target frequency.

In order for the accumulated GTC Live value to match exactly with the real passage of time, the following must be true:

The target frequency selected must be the CLOSEST possible selection to CDCLK. This corresponds to rounding the accumulator increment to the NEAREST 0.5ns increment with respect to the period of CDCLK.

Example

	Frequency	Period
Given CDCLK	337.500 MHz	2.96 ns
1st closest target frequency/increment	333.333 MHz	3.00 ns
2nd closest target frequency/increment	400.000 MHz	2.50 ns
3rd closest target frequency/increment	285.714 MHz	3.50 ns
4th closest target frequency/increment	500.000 MHz	2.00 ns

Only using the 1st closest target frequency 333.333MHz / accumulator increment of 3.00 ns will result in the GTC Live Value correctly tracking the real passage of time.

Example Calculation Flow:

1. Find the period of CDCLK -- for 337.5 MHz = 2.96 ns.
2. Round to the nearest 0.5 ns -- 3.00 ns (this is your accumulator increment value).
3. Find the "target frequency" from the rounded period value of step 2 -- 333.333 MHz is the "target frequency".
4. Find the ratio of target frequency / CDCLK = 0.987654321.
5. Choose M and N to satisfy M / N = same ratio as step 4.

If the results are not within the 1% error tolerance, multiply the accumulator increment used by 2 (i.e., use 1/2 of the target frequency). Re-select M and N as necessary to achieve the new target frequency as shown in the example below.

Example

	Frequency	Period	with 2x accumulator increment	New Period
Given CDCLK	652.800 MHz	1.531 ns		
1st closest target frequency/increment	500.000 MHz	1.500 ns	3.000ns	333.333 MHz

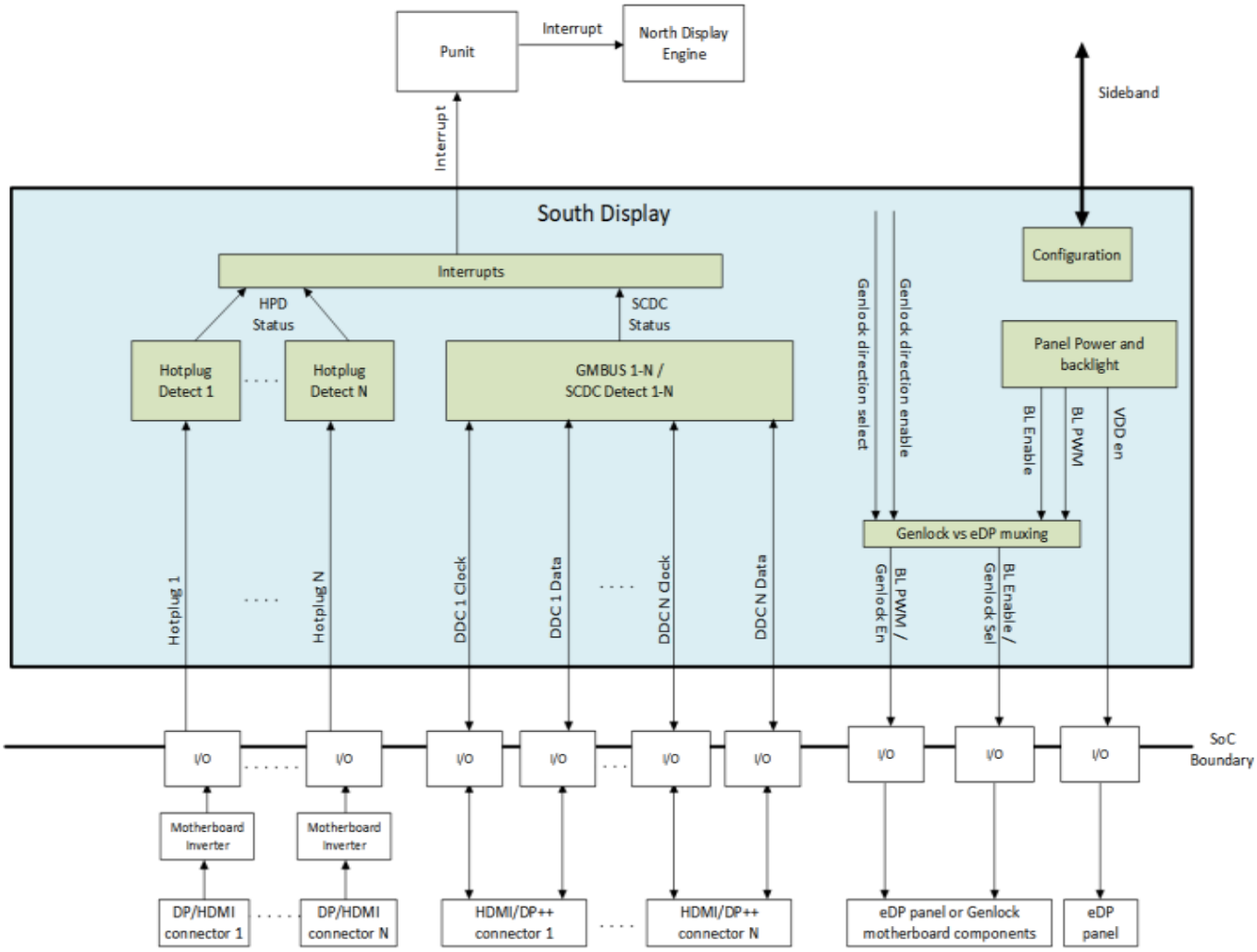
Example Calculation Flow:

1. Find the period of CDCLK -- for 652.800 MHz = 1.531 ns.
2. Round to the nearest 0.5 ns -- 1.500 ns
3. Multiply the above increment by 2 (this 3.00ns is your accumulator increment value).

4. Find the "target frequency" from the rounded period value of step 3 -- 333.333 MHz is the "target frequency".
5. Find the ratio of target frequency / CDCLK = 333.333 MHz / 652.800 MHz = 0.51062.
6. Choose M and N to satisfy M / N = same ratio as step 5.

South Display Engine Registers

South Display



The South Display Engine supports Hot Plug Detection, Panel Power Sequencing, Backlight Modulation, GMbus DDC, and motherboard genlock direction controls.

General

The south display uses the crystal oscillator raw clock for most functions.

Raw clock frequency is expected to be 38.4 MHz. The RAWCLK_FREQ register is used to configure dividers from raw clock to internal timer frequencies. It defaults to 38.4 MHz. If the actual frequency is something else, then RAWCLK_FREQ must be configured before enabling south display functions.

RAWCLK_FREQ

Genlock Direction

Genlock sends syncing signals between multiple graphics capable chips to align their display outputs for usages like a video wall. Each chip can be programmed as either a transmitter of sync signals (primary) or receiver of sync signals (secondary), and motherboard components control the routing of the syncs (direction in or out). The north display and PLLs are consuming and driving the sync signals. The south display is driving direction controls to the motherboard components.

There is a genlock direction enable signal to enable the motherboard components and a genlock direction select signal to specify the direction of the genlock signals. These are controlled by register fields.

To save pins, the genlock direction signals are muxed with the backlight signals since internal panel and genlock are mutually exclusive at the platform level.

- Backlight PWM muxed with Genlock direction enable
- Backlight Enable muxed with Genlock direction select

SBLC_PWM_CTL1 Register Field	Setting for Genlock Secondary Device	Setting for Genlock Primary Device	Setting for No Genlock
PWM Enable	0	0	As needed for backlight
Backlight Polarity	0	0	As needed for backlight
Genlock Direction IO Select	Genlock	Genlock	Backlight
Genlock Direction Enable Pin Value	1	1	0
Genlock Direction Select Pin Value	1	0	0

Panel Power and Backlight

Panel Power

PP_STATUS

PP_CONTROL

PP_ON_DELAYS

PP_OFF_DELAYS

IO pins are muxed between the backlight and genlock direction control signals. Set 0xC2000 bit 2 = 1 to switch the mux to allow the genlock direction controls to be driven. This setting is based on platform configuration and must be configured before using the backlight or genlock and then not changed afterwards.

Backlight

The backlight PWM output frequency is determined by the PWM clock frequency, increment, and frequency divider.

PWM output frequency = PWM clock frequency / PWM increment / PWM frequency divider

The frequency divider minimum must be greater than or equal to the number of brightness levels required by software; typically 100 or 256.

PWM clock frequency = Raw clock frequency.

PWM increment = 1

PWM frequency divider maximum = 2^32

Backlight Enabling Sequence

1. Set frequency and duty cycle in SBLC_PWM_FREQ Frequency and SBLC_PWM_DUTY Duty Cycle.

2. Enable PWM output and set polarity in SBLC_PWM_CTL1 PWM Enable and Backlight Polarity.

...

3. Change duty cycle as needed in SBLC_PWM_DUTY Duty Cycle.

If needed, granularity, polarity, and override can be programmed earlier than shown.

Backlight Frequency Change Sequence

1. Disable PWM output in SBLC_PWM_CTL1 PWM Enable.

2. Set new frequency and duty cycle in SBLC_PWM_FREQ Frequency and SBLC_PWM_DUTY Duty Cycle.

3. Enable PWM output in SBLC_PWM_CTL1 PWM Enable.

Backlight Registers

SBLC_PWM_CTL1

SBLC_PWM_FREQ

SBLC_PWM_DUTY

GMBUS and GPIO

Registers

GPIO_CTL - GPIO Control

GMBUS0 - GMBUS Clock/Port Select

GMBUS1 - GMBUS Command/Status

GMBUS2 - GMBUS Status

GMBUS3 - GMBUS Data Buffer

GMBUS4 - GMBUS Interrupt Mask

GMBUS5 - GMBUS 2 Byte Index

Pin Usage

These GPIO pins allow the support of simple query and control functions such as DDC interface protocols. The GMBUS controller can be used to run the interface protocol, or the GPIO pins can be manually programmed for a "bit banging" interface.

The following tables describe the expected GPIO pin to register mapping. OEMs have the ability to remap these functions onto other pins as long as the hardware limitations are observed. The GPIO pins may also be muxed with other functions such that they are only available when the other function is not being used.

Port #	Description
1	DDC for DDI A (combo port A).
2	DDC for DDI B (combo port B).
3	DDC for DDI C (combo port C).
4	DDC for DDI D (combo port D).
9	DDC for TypeC port 1.

GPIO Programming for I2C Bit Bashing

To drive GPIO pin low, program direction to "out" and data value to "0", along with mask bits.

To drive GPIO pin high (tristate to allow external pull up to activate), program direction to "in", along with mask bit. No need to set data value to "1".

The data value now has no effect but can continue to be programmed for backwards compatibility.

GMBUS Controller Programming Interface

The GMBUS (Graphic Management Bus) is used to access/control devices connected to the GPIO pins.

Basic features:

1. I²C compatible.
2. Bus clock frequency of 50 KHz or 100 KHz.
3. Attaches to any of the GPIO pin pairs.
4. 7-bit or 10-bit Secondary Address and 8-bit or 16-bit index.
5. Double buffered data register and a 9 bit counter support 0 byte to 256 byte transfers.
6. Supports stalls generated by the secondary device pulling down the clock line (Secondary Stall), or delaying the secondary acknowledge response.
7. Status register indicates error conditions, data buffer busy, time out, and data complete acknowledgement.
8. Detects and reports time out conditions for a stall from a secondary device, or a delayed or missing secondary acknowledge.
9. Interrupts may optionally be generated.
10. Does not directly support segment pointer addressing as defined by the Enhanced Display Data Channel standard.

Segment pointer addressing as defined by the Enhanced Display Data Channel standard:

1. Use bit bashing (manual GPIO programming) to complete segment pointer write **without terminating in a stop or wait cycle**.
2. Terminate bit bashing phase with both I²C lines pulled high by tri-stating the data line before the clock line. Follow EDDC requirement for response received from secondary device.
3. Initiate GMBUS cycle as required to transfer EDID following normal procedure.

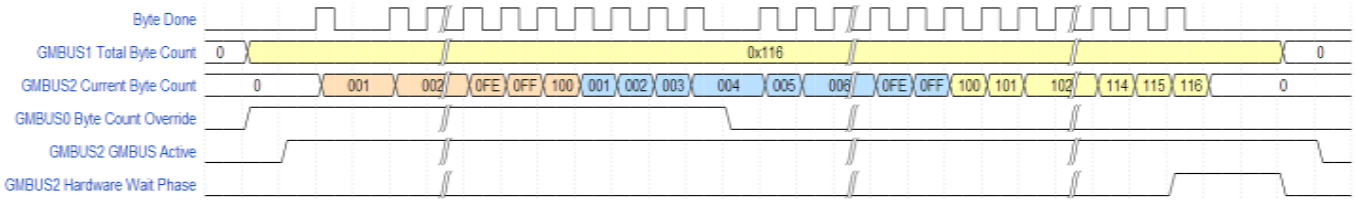
GMBUS cycles must not be initiated on platforms that do not have pull-ups on the clock and data lines. GMBUS cycles initiated without pull-ups present may fail to terminate correctly. GMBUS cycles initiated when pull-ups are present, and no receiver connected will complete with NACK.

Sequence for GMBUS Burst Reads Greater Than 511 Bytes

1. X=number of bytes to read. If X=512, use 513 and then ignore the extra read data.
2. N=INT(X/256)-1.
3. Set byte count override bit GMBUS0 bit 6. This makes the internal byte count rollover at 0x100.
4. Set total byte count field in GMBUS1 to X-N*256.
5. Read N*256+4 bytes.
6. Clear byte count override bit.
7. Read the remaining bytes and finish the transaction normally.

Example with 534 bytes

1. X=number of bytes to read=534.
2. N=INT(X/256)-1=1.
3. Set byte count override bit GMBUS0 bit 6.
4. Set total byte count field in GMBUS1 to X-N*256=278.
5. Read N*256+4 bytes=260.
6. Clear byte count override bit.
7. Read the remaining bytes and finish the transaction normally.



Interrupts and Hot Plug

South Display Engine Interrupt Bit Definition

SHPD_PULSE_CNT

SHPD_FILTER_CNT

SHOTPLUG_CTL_DDI

SHOTPLUG_CTL_TC

These registers are used for detecting hot plug. They will generate interrupts in SINTERRUPT. The interrupt ISR provides the live connect state of the HPD. SHOTPLUG_CTL has the status bits to indicate if long or short pulses are detected.

The SHOTPLUG_CTL status bits will not operate when hotplug is disabled. To find if a receiver was connected before hotplug was enabled, enable hotplug in SHOTPLUG_CTL and then read the interrupt ISR to find the live connect state.

Hot Plug Pins
DDIA (combo port A)
DDIB (combo port B)
DDIC (combo port C)
DDID (combo port D)
TC1 (TypeC port 1)

Hotplug Board Inversion

The hotplug level shifter on the board inverts the hotplug so that connect=0 and disconnect=1.

The hotplug logic re-inverts by default to account for the board inversion.

Hotplug with Device Power States

D0 Device Power State

The south display interrupt propagates to north display and then to the graphics interrupt MSI. The MSI will go to the CPU and trigger the graphics driver interrupt service routine. The driver then traverses the graphics interrupt structure down to south display and reads the status bits and live connect state to find which ports need servicing.

D3 Hot Device Power State

The graphics interrupt MSI is blocked when the graphics device is put into D3 through the PCI PMCS register. For D3 hot (device powered up), the SoC power controller will detect the south display interrupt assertion and send a PCI express PME message to the CPU. This will trigger the system to notify the graphics driver and bring the device to D0, which will release the interrupt MSI to the CPU. The graphics driver then reads the status bits and live connect state and clears out any interrupt like it does for D0.

D3 Cold Device Power State

The graphics device is powered down and unable to detect hotplug during D3 cold. As an option, while PCI aux power is present, card or motherboard logic can detect hotplug edge and drive the PCI WAKE# pin which propagates to the PCH. That will trigger the system to wake from S0ix and bring the graphics device up to D0. The graphics driver then scans through the live connect states to find which ports are connected.

HDMI Status and Control Data Channel Support

SCDC makes use of the same I2C interface utilized for E-EDID (DDC) accesses and for HDCP (I2C Bus) accesses. SCDC protocol extends the I2C standard by providing a mechanism for the Sink Device to request a Source Device to initiate a status check read. An interrupt is generated whenever such a read request (RR) is initiated by the sink (and RR_Enable bit is set in SCDCS). If the RR is not acknowledged within 1ms by the Source initiating a GMBUS read on the corresponding port, the Sink withdraws the RR and will attempt at later time.

As an example use case, a status flag has been defined to alert a Source when scrambling error occurs. Support to service a scrambling error flag will be required for HDMI display pixel rates greater than 340MHz (RGB 4K @ 60Hz).

Sequence to Enable a SCDC Interrupt:

1. Verify sink supports SCDC through DDC. Hardware requires at least one GMBUS transaction before enabling.
2. Set GMBUS0 Pin Pair Select to Disabled or to select a pin pair other than the one for this port. Hardware masks off SCDC when the pin pair is selected.
3. Program GPIO_CTL to set data direction to input (masked register write).

4. Set interrupt IMR to masked (1) for this interrupt. This is needed to prevent a false interrupt as SCDC enables.
5. Set interrupt IER to enabled (1) for this interrupt.
6. Clear interrupt IMR to unmasked (0) for this interrupt.

The SCDC interrupt can now happen and will appear in the interrupt IIR.

SCDC interrupt must be disabled before using GPIO_CTL to drive the data lane, such as for bit bashing. It does not need to be disabled for GMBUS transactions.

Sequence to Disable a SCDC Interrupt:

1. Clear interrupt IER to disabled (0) for this interrupt.

Display Watermark Programming

Watermark Overview

The display watermarks are used to control the display engine memory request behavior.

The default settings of the watermark configuration registers will **not** allow the display engine to operate. The watermark values must be properly calculated and programmed in order to enable a display and achieve optimum power and performance. Incorrectly programmed watermark values can result in screen corruption.

The watermarks should be calculated and programmed when any of the watermark calculation inputs change. This includes planes enabling or disabling, plane source format or size changing, etc.

Besides programming the watermark registers, there are other display configuration requirements and registers that must be programmed in order for the display to operate in a low power mode, and there are memory controller configuration requirements which are not documented here.

Watermark Calculations

The display watermarks are calculated using information from the display configuration and memory latencies. The watermarks must be calculated and programmed before enabling a plane or changing a plane configuration.

YUV 420 planar surface format	Plane source bytes per pixel
NV12	1 Bpp for Y surface and 2 Bpp for UV surface. Watermark values must be calculated and programmed for Y and UV surfaces separately in their watermark registers.
P0xx	2 Bpp for Y surface and 4 Bpp for UV surface. Watermark values must be calculated and programmed for Y and UV surfaces separately in their watermark registers.

The ceiling function rounds any non-integer value up to the next greater integer. Example: ceiling[0.3]=1, ceiling[2.1]=3, ceiling[4.8]=5, ceiling[4]=4

Resolutions Requiring Combined Pipes

For resolutions requiring 2 or more pipes to be joined together inside display engine, each pipe processes only 1/<# of joined pipes> of the image. The pixel rate and horizontal total pixels seen by each pipe is 1/<# of joined pipes> of that for the full resolution of the panel.

The excess pixels added for scaling smoothly across the seam between pipes do not impact the watermark.

For example: 7680x4320 CVT1.2 RB1 pixel rate is 2089.75 MHz with horizontal total 7840. That is split across 2 pipes, so each pipe is 3840x4320 with a pixel rate of 1044.875 MHz and horizontal total 3920.

Watermark Algorithm

1. Retrieve memory latency values
    - See the Memory Values section to find the memory latency values
    - The memory values do not change after boot, so software may cache them to avoid re-reading
2. For each enabled pipe (run each time pipe configuration changes)
    A. Calculate adjusted pipe pixel rate
        I. Adjusted pipe pixel rate = pixel rate for the screen resolution
            - If there will be dynamic switching between refresh rates, either use the fastest pixel rate, or re-calculate using the current pixel rate when the refresh rate is switched. Because Plane watermark registers and Link M/N registers have different double buffer update points, software will have to program both at the beginning of vactive. Alternatively, when switching from a low to high refresh rate, program the WM values first and when switching from a high to low refresh rate program the Link M/N values first. This second approach can result in a one frame delay in the refresh rate switch.
            - If plane 90 or 270 rotation is enabled, use the rotated width and height in pixel rate calculations.
        II. If TRANS_CONF Interlaced Mode == PF-ID, adjusted pipe pixel rate = adjusted pipe pixel rate * 2
        III. If pipe scaling enabled, adjusted pipe pixel rate = adjusted pipe pixel rate * pipe down scale amount
            - Refer to the Display Resolution Support section on Scaling to find the down scale amount
    B. Program WM_LINETIME Line Time = roundup[8 * pipe horizontal total pixels / adjusted pipe pixel rate MHz]
3. For each enabled plane (run each time pipe or plane configuration changes)
    A. Calculate adjusted plane pixel rate
        I. Adjusted plane pixel rate = adjusted pipe pixel rate

II. If plane scaling enabled, adjusted plane pixel rate = adjusted plane pixel rate * plane down scale amount
    - Refer to the Display Resolution Support section on Scaling to find the down scale amount

B. For each valid memory latency level

Plane Bytes per pixel	Minimum Scanlines for Y Tile	
	0/180 Rotation	90/270 Rotation
1	4	16
2	4	8
4	4	4
8	4	N/A

Plane Memory format	DBuf block size
8 bits per pixel surface format + Yf tiling	256
All other tiling and surface formats	512

I. Calculate method 1
    - Method 1 = (memory latency microseconds * adjusted plane pixel rate MHz * plane source bytes per pixel / DBuf block size) +1

II. Calculate method 2
    - plane bytes per line = plane source width pixels * plane source bytes per pixel
    - Calculate plane blocks per line
        - If plane memory format is Linear
            - plane blocks per line = ceiling[plane bytes per line / DBuf block size] + 1
        - Else If plane memory format is Y tile
            - plane blocks per line = ceiling[(Minimum Scanlines for Y tile * plane bytes per line/ DBuf block size)+1]/Minimum Scanlines for Y tile
        - Else
            - plane blocks per line = ceiling[plane bytes per line / DBuf block size] + 1
    - Method 2 = ceiling[(memory latency microseconds * adjusted plane pixel rate MHz) / Pipe horizontal total number of pixels] * plane blocks per line

III. Calculate Y tile minimum
    - Y tile minimum = Minimum Scanlines for Y tile * plane blocks per line

IV. Select the watermark result
    - line time microseconds = pipe horizontal total pixels/adjusted plane pixel rate MHz
    - If plane memory format is X tile or linear

- If (((plane source bytes per pixel * pipe horizontal total number of pixels) / DBuf block size) < 1) AND ((plane bytes per line / DBuf block size) < 1) // Special case for unrealistically small horizontal total
  - Selected Result Blocks = Method 2
- Else If ('plane buffer allocation' is known and (plane buffer allocation / plane blocks per line) >=1)
  - Selected Result Blocks = Method 2
- Else If (memory latency microseconds >= line time microseconds)
  - Selected Result Blocks = Method 2
- Else
  - Selected Result Blocks = Method 1

- Else // Y tile
  - Selected Result Blocks = maximum[Method 2, Y tile minimum]
- If Pipe YUV420 Bypass // PIPE_MISC YUV420 Enable == Enable and YUV420 Mode == Bypass
  - Selected Result Blocks = maximum[Selected Result Blocks, plane blocks per line] // Minimum is 1 line when using pipe YUV420 bypass

V. Convert result to blocks and lines

- Result Blocks = ceiling[Selected Result Blocks] + 1
- Result Lines = ceiling[Selected Result Blocks / plane blocks per line]
- If Y Tiling

  If 'Result Lines' is multiple of 'Minimum Scanlines for Y tile'

  Extra Lines = Minimum Scanlines for Y tile

  Else

  Extra Lines = (Minimum Scanlines for Y tile * 2) - (Result Lines % Minimum Scanlines for Y tile)

  Minimum Display Buffer allocation Needed = ceiling[(Result Lines + Extra Lines) * plane blocks per line]

  Else

  Minimum Display Buffer allocation Needed = ceiling[Result Blocks + (Result Blocks * 0.1)]

VI. Compare against the maximum

- If (Result Blocks >= plane buffer allocation), maximum exceeded for this latency level
- 255
- If (Result Lines > Maximum Lines), maximum exceeded for this latency level

- If (Minimum Display Buffer allocation Needed >= plane buffer allocation), maximum exceeded for this latency level
- or YUV 420 Planar formats, perform the above check for both Y and UV planes.

4. For each transition watermark
   - This includes<PLANE,CUR>_WM_TRANS, which is calculated relative to memory latency level 0.

   A. Calculate transition offset
      - Transition Offset Blocks = Transition minimum + Transition amount
      - See Transition Watermark section for transition minimum and transition amount

   B. Calculate transition Y tile minimum
      - Transition Y tile minimum = 2 * relative memory latency level Y tiled minimum

   C. Select the watermark result
      - If plane memory format is X tile or linear
        - Result Blocks = Relative memory latency level Selected Result Blocks + Transition Offset Blocks
      - Else // Y tile
        - Result Blocks = maximum[Relative memory latency level Selected Result Blocks, Transition Y tile minimum] + Transition Offset Blocks

   D. Convert result to blocks
      - Result Blocks = ceiling[Result Blocks] + 1

   E. Compare against the maximum
      - If (Result Blocks >= plane buffer allocation), maximum exceeded for transition watermark
      - For YUV 420 Planar formats, perform the above check for both Y and UV planes.

5. Program watermark registers

   A. For each latency level
      - If memory latency for this level is invalid, or the maximum was exceeded for this level or any previous level, program PLANE_WM_<latency level> Enable = 0
        - **If watermark latency level 0 exceeds the maximum, the plane must not be enabled.**
      - Else program PLANE_WM_<latency level> Enable = 1, Lines = Result Lines, Blocks = Result Blocks

   B. For transition watermarks
      - If the maximum was exceeded for the transition watermark, program the transition watermark register Enable = 0
      - Else program the transition watermark register Enable = 1, Blocks = Transition Result Blocks
        - The transition watermark Lines value is ignored by hardware

C. Write the plane surface base address register to trigger update of the watermarks and other plane double buffered registers. This should be done only after all plane configuration is configured to match the new watermark values.

Transition Watermark

The transition watermark is used for Isochronous Priority Control (IPC). When IPC is enabled, plane read requests are sent at high priority until filling above the transition watermark, then the requests are sent at lower priority until dropping below the level 0 watermark. The lower priority requests allow other memory clients to have better memory access. If the transition watermark is not enabled, the plane behaves as if the transition watermark was programmed to the top of the plane buffer allocation. When IPC is disabled, all plane read requests are sent at high priority. It is allowed for one of a pair of planes supporting YUV420 together to have transition watermark disabled while the other plane has it enabled.

The transition watermark is programmed as a tunable amount above the relative memory latency level watermark. Tuning to higher values will tend to cause longer periods of high priority reads followed by longer periods of lower priority reads. Tuning to lower values will tend to cause shorter periods of high and lower priority reads. The exact behavior depends on the memory bandwidth, display bandwidth, and other memory traffic in the system.

The transition watermark has a minimum value to ensure the demote does not happen before enough data has been read to meet the relative memory latency level watermark requirements.

Transition Minimum: 4 Blocks

IPC Enable Register Field
ARB_HP_CTL Enable IPC

System Agent Geyserville (SAGV) Interaction with Watermarks

SAGV dynamically adjusts the system agent voltage and clock frequencies depending on power and performance requirements. The display engine access to system memory is blocked outside of display engine, without a PM handshake, during the adjustment time.

To compensate for the block, display engine must increase the latency for watermark level 0 by the SAGV block time. If there is not enough data buffer to support that increase, then software must disable SAGV. Other watermark levels do not need increase because after a package C exit SAGV will not run until after display has refilled above watermark level 0.

SAGV defaults to enabled.

See the Memory Values section to find the SAGV block time.

The latency input to the watermark calculation for each level must be greater than or equal to the lower level. The latency increase to level 0 for SAGV requires the upper levels to be adjusted to meet that requirement. Use MIN(latency for this level, latency for next lower level) to correct the latency.

Requirement before plane enabling or configuration change: Calculate watermark level 0 with level 0 latency + SAGV block time. If the result can be supported (does not exceed maximum), then the plane

can tolerate SAGV, so proceed with watermark calculations for level 0 using latency increased by SAGV block time.

If the plane cannot tolerate SAGV, proceed with watermark calculations for all levels using the normal latency values.

Disable SAGV (sequence below) if any enabled plane cannot tolerate SAGV, or any transcoder is interlaced. Else, enable SAGV.

The expectation is that SAGV can be tolerated in most non-interlaced configurations and will only need to be disabled for some high resolution, multiple display scenarios.

If one plane was already enabled and able to tolerate SAGV, then a second plane becomes enabled and cannot tolerate SAGV, causing SAGV to disable, software can recalculate and reprogram the watermarks for the first plane without the SAGV block time. That reprogramming is preferred to give more optimal watermark results, but not required.

To disable SAGV, follow the SAGV section SAGV Point Selection Runtime flow to mask off all but the one QGV point that supplies the highest bandwidth for display.

To enable SAGV, follow the SAGV section SAGV Point Selection Runtime flow to unmask all of the QGV points that supply enough bandwidth for the display configuration.

Examples

Example pixel rate adjustments:

Pixel rate for screen resolution is 130 MHz. No interlacing. Pipe scale 1920x1080 pipe source size to 1714x1120 scaler window size. Plane scale 1920x1080 plane size to 800x600 scaler window size.

Pipe horizontal down scale amount = formula from Display Resolution Support section on Scaling with scaler input 1920 and scaler output 174 = 1.2

Pipe vertical down scale amount = maximum[1, 1080 / 1120] = 1 **// Max condition was hit**

Pipe total down scale amount = 1.2 * 1 = 1.2

Adjusted pipe pixel rate = 130 MHz * 1.2 = 156 MHz

Plane horizontal down scale amount = formula from Display Resolution Support section on Scaling with scaler input 1920 and scaler output 800 = 2.5

Plane vertical down scale amount = maximum[1, 1080 / 600] = 1.8

Plane total down scale amount = 2.5 * 1.8 = 4.5

Adjusted plane pixel rate = 145.6 MHz * 4.5 = 655.2 MHz

Example method, block, and line calculations:

Plane source 4 Bpp, Plane X tile, Plane source width 1920 pixels, Horizontal total 2200 pixels, Adjusted plane pixel rate 148.5 MHz, memory latency 7.5 us

Method 1 = 148.5 MHz * 4 Bpp * 7.5 us / 512 = 8.7 blocks

Plane bytes per line = 1920 pixels * 4 Bpp = 7680 Bytes/line

Plane blocks per lines = ceiling[7680 / 512] = 15 blocks

Method 2 = ceiling[(7.5 us * 148.5 MHz) / 2200 pixels] * 15 blocks = 15 blocks

Y tile minimum = 4 * 15 blocks = 60 blocks

Result Blocks = minimum[8.7 blocks, 15 blocks] = 8.7 blocks // X tile so does not use Y tile minimum

Result Blocks = ceiling[8.7 blocks] + 1 block = 10 blocks

Result Lines = ceiling[8.7 blocks / 15] = 1 lines

Memory Values

Retrieve Memory Latency Data

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data0=0x0000_0000 (first set of latency values) and GT Driver Mailbox Data1=0x0000_0000
3. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
4. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
5. Read GT Driver Mailbox Data0 for the first set of memory latency values
6. Write GT Driver Mailbox Data0=0x0000_0001 (second set of latency values) and GT Driver Mailbox Data1=0x0000_0000
7. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
8. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
9. Read GT Driver Mailbox Data0 for the second set of memory latency values

Memory Latency Data Definition

First Set:

Data0 Bit	Name	Description
31:24	Level 3	Number of microseconds*2 for level 3.
23:16	Level 2	Number of microseconds*2 for level 2.
15:8	Level 1	Number of microseconds*2 for level 1.
7:0	Level 0	Number of microseconds*2 for level 0.

Second Set:

Data0 Bit	Name	Description
31:24	Level 7	Number of microseconds*2 for level 7.
23:16	Level 6	Number of microseconds*2 for level 6.
15:8	Level 5	Number of microseconds*2 for level 5.
7:0	Level 4	Number of microseconds*2 for level 4.

If level 1 or any higher level has a value of 0x00, that level and any higher levels are unused and invalid, so the associated watermark registers must not be enabled. Level 0 is always valid.

It is allowed to have the same value in adjacent levels.

Programming Note	
Context:	Display Watermark Programming
The mailbox response data may not account for memory read latency. If the mailbox response data for level 0 is 0us, add 3 microseconds to the result for each valid level. The previous requirement was 2us, but high density DIMMs have increased refresh times that requires a 1us increase.	

The mailbox latency values are in units of microseconds * 2 to support values greater than 255 microseconds. Multiply the value by 2 to get microseconds.

SAGV Block Time

The block time is retrieved from the GT driver pcode mailbox.

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data0=0x0000_0000 (first set of latency values) and GT Driver Mailbox Data1=0x0000_0000
3. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=23h
4. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
5. Read GT Driver Mailbox Data0 for the block time in microseconds