altera
An Intel Company

Intel® FPGA Streaming Video Protocol Specification

Online Version
Send Feedback

MNL-1111

683397
2024.05.15

Contents

1. About the Intel FPGA Streaming Video Protocol

This protocol allows very high resolution and frame rate video processing. The protocol supports any number of pixels in parallel per clock cycle and video resolutions of up to 65536 by 65536 pixels.

An AMBA AXI4-Stream protocol underpins this architecture, which meets the needs of video and vision processing IPs, and allows easy interoperability with the latest Intel FPGA IP and other third-party IPs.

The AMBA AXI4-Stream protocol is natively supported in Platform Designer, allowing you to easily make connections between components.

The protocol allows interfacing to Intel FPGA video IPs or other AXI4-Stream compliant third-party video IPs.

Note: A lite variant of the protocol specifies how video packets transport video data. The full variant adds transport of control packets. The full-raster protocol adds support for full-raster signalling.

The protocol runs on top of the AXI4-Stream wire-level protocol, with extensions for transporting control and video data.

The protocol moves color planes in parallel, with one or more pixels in parallel in one beat of data. You can specify as many pixels in parallel as the IPs can process using the protocol.

Related Information

- Intel FPGA Streaming Video Full-Raster Protocol on page 32
- AMBA 4 AXI4-Stream Protocol
- Lite versus Full Variants

1.1. Protocol Signals

The Intel FPGA streaming video protocol allows you to transfer a variety of video data following the AXI4-Stream protocol signals.

Table 1. Protocol Signals

Signal	Description
TDATA	Set TDATA width according to need. The minimum allowable width of TDATA on all IP interfaces is 16 bits. The width of TDATA is byte aligned (i.e. multiple of 8). Systems that require smaller TDATA interfaces must pad their data. The TKEEP and TSTRB signals are unused, so every byte of TDATA is valid (no empty pixels). You deduce the exact length of a video packet from the image dimensions.
TUSER[n:0]	Sized as 1 bit per byte of TDATA.

continued...

Signal	Description
	Strobe TUSER[0] high for the first beat of a frame (or interlaced field) of video and drive low for subsequent packet beats.. For full variants, strobe TUSER[1] for the first beat of a packet to indicate whether it carries control or data information and drive low for subsequent packet beats.. For lite variants, IPs use TUSER[1] for interlaced video to indicate fields of type F1. TUSER[1] must remain high for the remainder of the field. The IP should drive low the other bits of TUSER.
TLAST	The protocol transmits each line of video as one AXI4-S packet. TLAST strobes high to indicate the last beat of the packet. TLAST is always coincident with the last pixel of the line..
TREADY and TVALID	Indicate the valid cycles on the bus.
TID	The TID data stream identifier is unused.
TDEST	The TDEST routing information signal is unused.
TSTRB	The TSTRB byte qualifier is unused. All bytes are valid in the protocol.
TKEEP	The TKEEP byte qualifier is unused. The protocol has no null bytes.

The TDATA bus carries either pixel data or other information. The TDATA width depends on the color depth, color space, and number of pixels in parallel (*PIP*).

TDATA byte width is the width of one pixel in bytes multiplied by the number of pixels in parallel.

The width of one pixel, in bytes, is the number of bits per color symbol (*BPS*) multiplied by the number of color symbols per pixel (*SYM*), rounded to the next byte. If the width is less than 16 bits, round it to 16 bits.

TDATA byte width = $\max(2, \text{ceil}((BPS \times SYM)/8)) \times PIP$

The TUSER bus must be TDATA/8 bits wide (and at least 2 bits wide to match the minimum TDATA width of 16 bits).

The TLAST signal indicates the end of a packet.

TDATA, TUSER and TLAST are undefined when TVALID is low.

Related Information

Lite versus Full Variants

1.2. Data Exchange

The TVALID and TREADY handshake determines when information passes across the AXI4-Stream. A two-way flow control mechanism enables both the input and output interfaces to control the rate at which the data and control information transmits across the interface.

For a valid transfer to occur the output interface and input interface must assert TVALID and TREADY respectively for one clock cycle. The output interface must not wait until the input interface asserts TREADY before asserting TVALID. When the output interface asserts TVALID, it keeps asserted until the handshake occurs. Other signals (TDATA, TUSER, TLAST) must remain stable. An input interface can wait for TVALID to be asserted before asserting the corresponding TREADY. If an input interface asserts TREADY, it can deassert TREADY before TVALID is asserted.

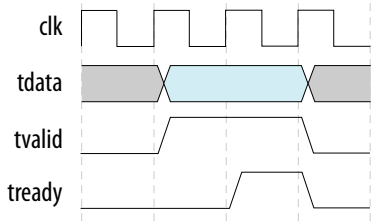
A successful handshake occurs when:

- TVALID is asserted before TREADY
- TREADY is asserted before TVALID
- TREADY and TVALID asserted on the same cycle

The figures do not show TUSER or TLAST but these signals also follow the same rules for validity as shown for TDATA.

Figure 1. AXI4-Stream Handshake: TVALID before TREADY

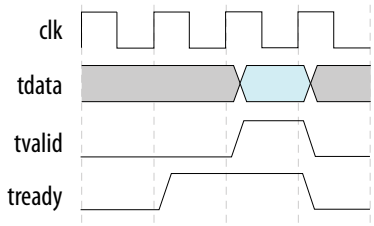
The figure shows the output interface producing data and asserting the TVALID signal active-high.



When the output interface asserts TVALID, the data from the output interface must remain unchanged until the input interface drives the TREADY signal active-high, indicating that it can accept the data. In this case, transfer takes place when the input interface asserts TREADY active-high.

Figure 2. AXI4-Stream Handshake: TREADY before TVALID

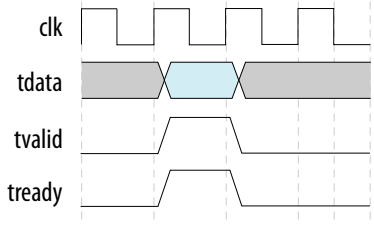
The figure shows the input interface driving TREADY active-high before the data and control information is valid.



The destination can accept the data and control information in a single clock cycle. In this case, transfer takes place when the output interface asserts TVALID active-high.

Figure 3. AXI4-Stream Handshake: TVALID and TREADY asserted at the same time

The figure shows both output and input interfaces asserting TVALID and TREADY active-high in the same clock cycle, respectively. In this case, data transfer takes place in the same cycle.



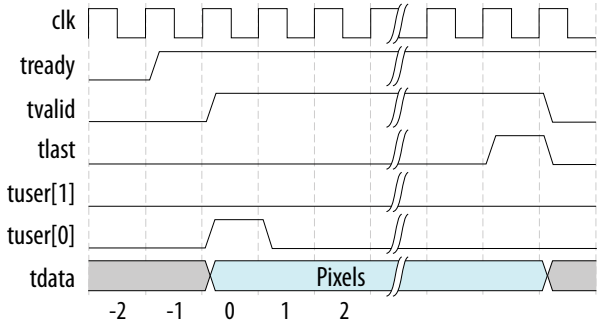
2. Intel FPGA Streaming Video Protocol Description

The Intel FPGA streaming video protocol supports video frames (or fields) as small as 1 pixel by 1 pixel and as large as 65536 by 65536 pixels. The protocol accepts error conditions where IPs create empty frames. The TUSER bus contains header information such as the start of video fields and whether a packet is a metapacket (full variant only) or video packet.

Unless stated, the figures apply to both full and lite variants. Video packets contain video data for one line of video.

Figure 4. Intel FPGA streaming video packet – first line of a new field

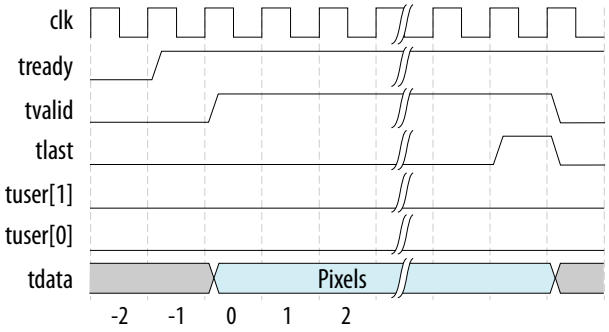
The figure shows TUSER[0] indicating the start of a new field of video.



TUSER[0] is decoded during the first valid beat of the packet (beat 0 in the figure) and a high value indicates the start of a new field of video. TUSER[0]should be low for the remainder of the packet.

Figure 5. Intel FPGA streaming video packet

The figure shows an example video packet. TUSER[0] is low so does not indicate the start of a new field.



Metapackets (Full Variants) on page 7

Video Packets on page 14

Related Information

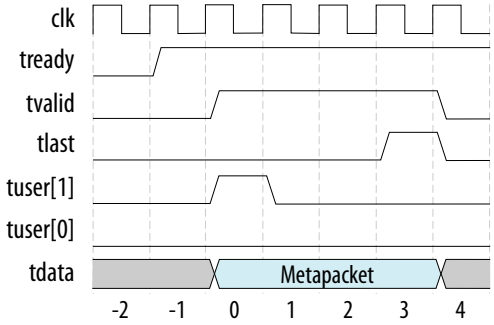
Lite versus Full Variants

2.1. Metapackets (Full Variants)

Metapackets usually carry control information. Use metapackets in the full variant of the Intel FPGA streaming video protocol but not in the lite variant. Metapacket lengths are between 1 to 4 beats (clock cycles).

Figure 6. Intel FPGA streaming metapacket (full variant only)

The figure shows a packet where TUSER[1] is high in beat 0, which shows this packet is a metapacket.



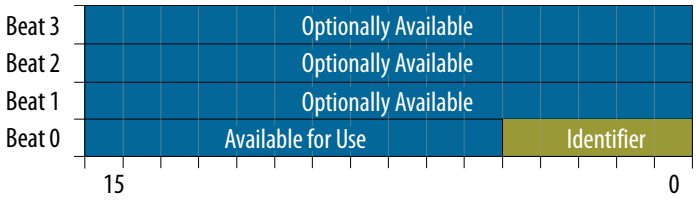
TUSER[1] is decoded during the first valid beat of the packet (beat 0 in the figure) and a high value indicates the start of a new metapacket. TUSER[0] and TUSER[1] should be low for the remainder of the packet.

Table 2. TUSER bits decode (Full variants only)

TUSER[1:0]	Meaning
00	Video packet (from either an interlaced (F0 or F1) field or progressive frame).
01	Video packet (start of either an interlaced (F0 or F1) field or progressive frame).
10	Metapacket.
11	Reserved.

Metapackets carry their information in the low 16 bits of TDATA. The protocol does not require any additional TDATA bits. The protocol defines different metapacket types, each with a 5 bit packet identifier carried in the 5 LSBs of the first beat of the packet.

Figure 7. Metapacket format



The protocol classifies metapackets into image information, end of field, auxiliary control and user-defined auxiliary control packets.

Table 3. Metapacket types

Packet identifier	Packet type
0	Image information
1	End of field
Auxiliary control packets	
2	Timestamp
3	Register update
4	Commit
5-15	Reserved
User-defined auxiliary control packets	
16 - 31	General-purpose user

Related Information

Lite versus Full Variants

2.1.1. Image Information Packets

Image information packets mark the start of a field of video and contain all the information you need to describe the characteristics of the field in the Intel FPGA streaming video protocol.

Figure 8. Example image information packet

The figure shows an example image information packet. The packet identifier in beat 0 is 0 and in this example, the higher bits in this beat are also set to 0.

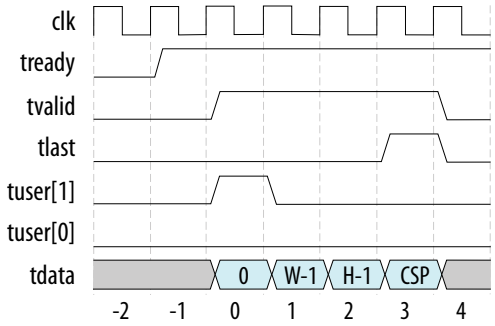
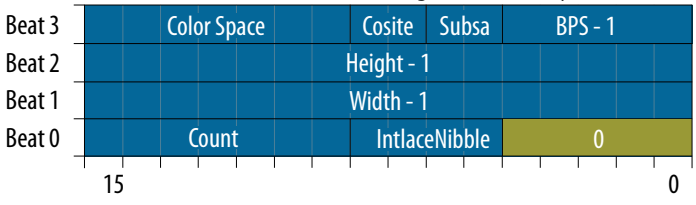


Figure 9. Image information packet structure

The figure shows the detailed structure of each beat of the image information packet.



The identifier is in beat 0, the field width minus 1 in beat 1, field height minus 1 in beat 2, and bits per symbol (BPS) minus 1 and color space information in beat 3.

One or more video packets should follow image information packets, which must have an associated end-of-field packet. If no video packets follow an image information packet (which may sometimes occur in some applications), an end-of-field packet must still follow the image information packet, possibly with other metapackets in between.

The metapacket identifier of '0' is in the low 5 bits of beat 0, the other fields contained in the packet are:

- `IntlaceNibble`
- `Count[6:0]`
- `Width-1` and `Height-1`
- `SubSa`
- `Cosite`
- `ColSpace`

`IntlaceNibble`

The interlaced nibble field describes whether the next video packet is interlaced or progressive.

Table 4. Image Information Packet `IntlaceNibble`

`IntlaceNibble[3]`	`IntlaceNibble[2]`	`IntlaceNibble[1]`	`IntlaceNibble[0]`	Description	Frame Type
1	1	1	1	Interlaced F1 field, pairing don't care	Interlaced
		1	0	Interlaced F1 field, pairing don't care	
		0	1	Interlaced F1 field, paired with the F0 field preceding it	
		0	0	Interlaced F1 field, paired with the F0 field following it	
	0	1	1	Interlaced F1 field, pairing don't care	
		1	0	Interlaced F1 field, pairing don't care	
		0	1	Interlaced F1 field, paired with the F0 field following it	
		0	0	Interlaced F1 field, paired with the F0 field preceding it	
0	1	1	1	Progressive frame, natively progressive	Progressive
0	1	1	0	Progressive frame, deinterlacing status unknown	Progressive

continued...

IntlaceNibble[3]	IntlaceNibble[2]	IntlaceNibble[1]	IntlaceNibble[0]	Description	Frame Type
		0	1	Progressive frame, deinterlaced from an f1 field	
		0	0	Progressive frame, deinterlaced from an f0 field	
	0	1	1	Progressive frame, natively progressive	
		1	0	Progressive frame, deinterlacing status unknown	
		0	1	Progressive frame, deinterlaced from an f1 field	
		0	0	Progressive frame, deinterlaced from an f0 field	

`Count[6:0]`

These 7 bits are a copy of the low 7 bits of a field count field found in the end-of-field packet.

`Width-1` and `Height-1`

The protocol supports video field widths and heights up to 65536 pixels. The values in the fields require you to add 1 to give the width and height. The protocol then supports 65536 (2^{16}). The protocol considers zero width or height video fields error conditions, so you need not code for these.

SubSa

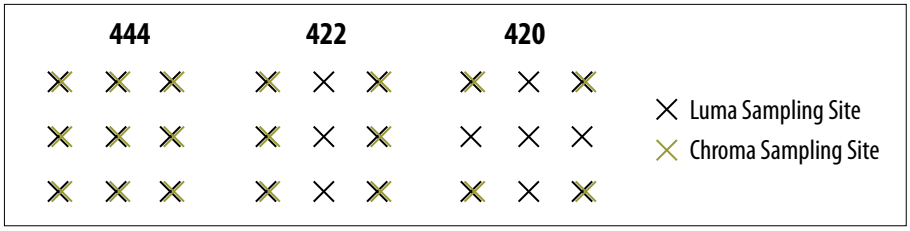
These two bits decode to indicate the video field chroma sampling.

Table 5. Image Information Packet SubSa

SubSa	Vertical chroma sub-sampling	Horizontal chroma sub-sampling	Chroma sub-sampling term
00	2	2	420
01	2	1	UNUSED
10	1	2	422
11	1	1	444

The MSB of the SubSa field represents the vertical chroma subsampling, with the horizontal represented by the LSB.

Figure 10. Image Information Packet SubSa sampling sites



Cosite

These two bits encode the cositing of the chroma samples. The cositing is the position of the chroma sample sites with respect to the luma sample sites.

The cositing of chroma may be unknown, in which case set this field to 00 (top and left).

Table 6. Image Information packet Cosite

CoSite	Cositing
00	Top and left
01	Top and center
10	Center and left
11	Center and center

ColSpace

These 7 bits encode the color space of the video. You may assign custom color spaces to encodings 64-127.

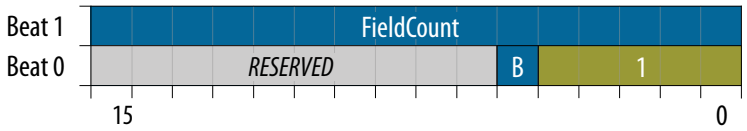
Table 7. Image Information Packet ColSpace

ColSpace	Color space
0	RGB
1	YCbCr
2	YCbCrSD
3	YCbCrHD
4	Mono
5	Raw
6	RGB565
6-63	Reserved
64-127	User-defined color spaces

2.1.2. End-of-Field Packets

The end-of-field packet indicates the end of a field of video in the full variant of the Intel FPGA streaming video protocol.

Figure 11. End-of-Field Packet



End-of-field packets have a packet identifier of 1.

Bit 5 is the `Broken` flag. The broken bit may be set to indicate that something is wrong with the frame.

Bits 6-15 are unused.

A 16 bit `field count` is stored in the end-of-field packet.

The `count` quantity in the image information packet is `field count` modulo 128.

2.1.3. Auxiliary Control Packets

Systems that use the full variant Intel FPGA streaming video protocol must use the image information packets and end-of-field packets, but support for auxiliary control packets is optional. However, all IPs implementing the protocol must still forward any auxiliary control packets, unless the IP terminates the video pipeline or the behavior of the IP requires deviation from this behavior.

Timestamp Packets on page 12

Register Update Packets on page 13

Commit Packets on page 13

User-defined Auxiliary Control Packets on page 14

Related Information

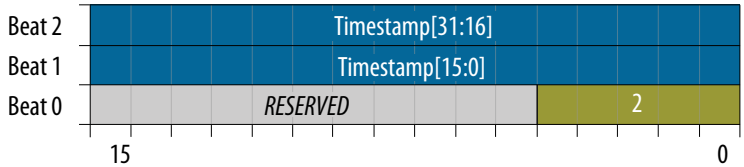
Rules for Packets on page 28

2.1.3.1. Timestamp Packets

Intel FPGA streaming video timestamp packets are a synchronization facility for audio and video data synchronization.

Figure 12. Timestamp Packet

The figure shows the structure of the packets.

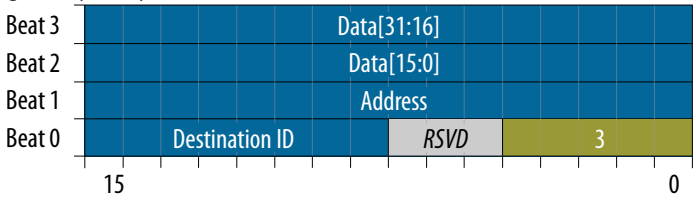


2.1.3.2. Register Update Packets

Intel FPGA streaming video register update packets allow individual video fields to individually receive different video processing, as you can inject register update packets between fields. Applications for register update packets include Dolby Vision, HDR10 or video processing applications that stream the control.

Figure 13. Register Update Packet

The figure shows register update packets.



A destination ID in bits 15 to 8 in the first beat of the register update packet allows the packet to be targeted at a particular destination IP. The second beat contains the address of a register to update. Beats 3 and 4, respectively, contain the least significant short word and most significant short word of the data you write.

Destination IDs `0xF0` to `0xFF` are reserved.

Place register update packets between the end of field packet of the previous field and the image information packet of the field to which you apply the new control data.

Refer to *Legal packet ordering rules* and *Packet ordering examples* figures in *Rules for Packets*.

Related Information

Rules for Packets on page 28

2.1.3.3. Commit Packets

Intel FPGA streaming video commit packets allow you to switch in a new set of control register values for synchronized operations such as crop and scale on an exact field basis.

Figure 14. Commit Packet

The commit packet triggers an update to a new set of control values.

When an IP receives a commit packet, it changes to any new register values at the start of the next video frame. The commit register ensures that all register updates occur simultaneously, which avoids the IP only applying some of the updated register values for the next incoming frame.

Like register update packets, the protocol does not allow commit packets between the last line of a field and an end-of-field packet. They should be in between the end of field of a previous field and the image information packet of the next field.

Refer to *Legal packet ordering rules* and *Packet ordering examples* figures in *Rules for Packets*.

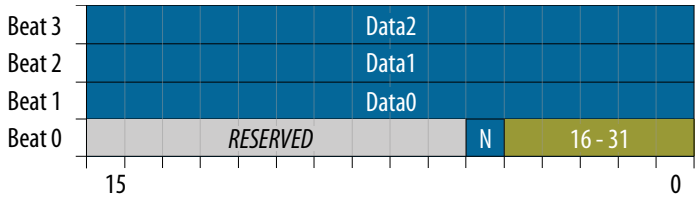
Related Information

Rules for Packets on page 28

2.1.3.4. User-defined Auxiliary Control Packets

The Intel FPGA streaming video protocol classifies metapackets with packet identifiers of 16 to 31 as user-defined auxiliary control packets.

Figure 15. User-Defined Auxiliary Control Packets Structure



User-defined auxiliary control packets may be either standalone with a payload held in the last three beats, or a mechanism to synchronize with other packets transmitted over another interface.

User-defined auxiliary control packets can have packet identifiers from 16 to 31.

The first beat includes a no data bit in bit 5 (*N* in the figure). If an auxiliary control packet has associated data with it on another interface, this bit is clear. However, if the auxiliary control packet stands alone, with no additional data, set this bit.

The data carried by auxiliary control packets comprises three short-words in the three following beats of the transaction.

A typical application either uses the data fields for a small amount of auxiliary data for the next video field, or leaves these empty and populates auxiliary data on another bus.

2.2. Video Packets

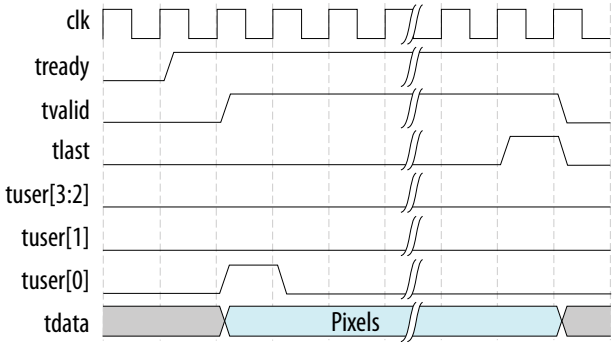
For the full variant Intel FPGA streaming video protocol, `TUSER[1]` is low in cycle 0 to indicate this packet is a video packet. For the lite protocol, `TUSER[1]` indicates the field polarity, and must be constant throughout the packet. A low indicates this packet is either from a progressive frame or an interlaced field of type F0. A high indicates this packet is from an interlaced field of type F1.

`TUSER[0]` is decoded in the first valid cycle of the packet. The start of a new video field is indicated when `TUSER[0]` is high in this cycle.

Video packets carry pixel data. Both full and lite variants of the protocol use video packets. Video packets may be 1 or more beats in length.

Figure 16. A video packet for the first line of a new video field (1x30bit pixel interface)

The figure shows an example video packet for the first line of a video field. TUSER[0] is indicating this line is the start of a new video field.

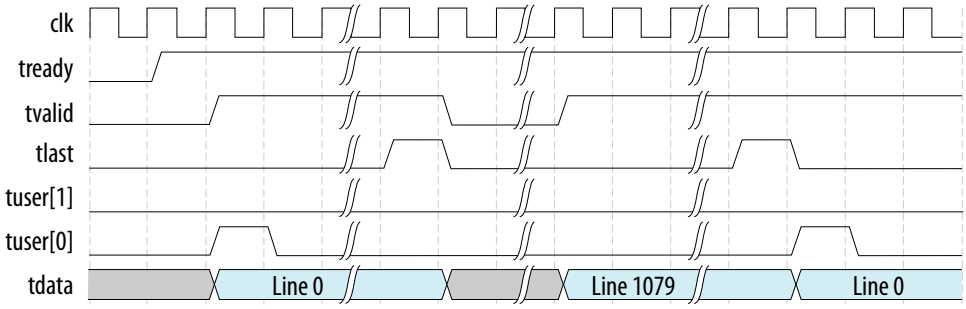


In the example, TUSER is 4 bits wide, as the 30 bits of pixel data occupy 4 bytes of TDATA. In the first beat of the transfer, TUSER indicates a new field of video by asserting TUSER[0] high and TUSER[1] low to indicate a video packet. Other TUSER bits, e.g. TUSER[3:2], can take any value, but IPs compliant with the Intel FPGA streaming protocol do not have to propagate these values and it is recommended that these are held low. TLAST indicates the last pixel in each line.

TUSER[0] indicates the start of a new video field, but you cannot tell which video packet is the last line of the field. The full variant of the protocol allows the detection of the end of the current video field by the arrival of an end of field packet. The lite variant has no signaling for the end of the current field, but TUSER[0] indicates the first pixel of the next field.

Figure 17. Video frame transmission (lite variant)

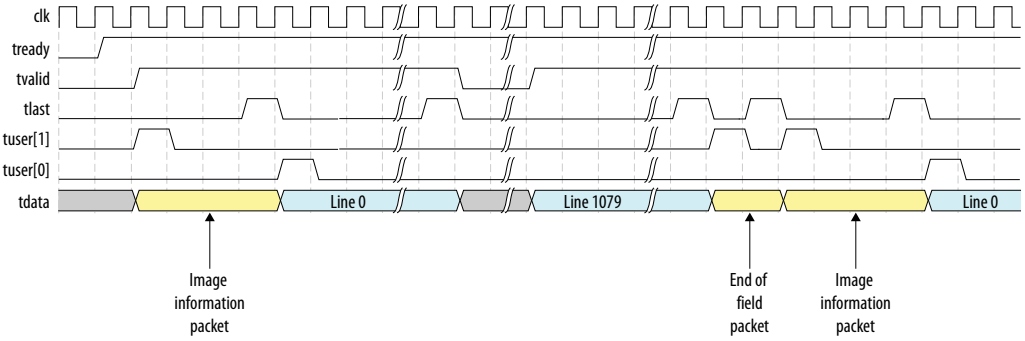
The figure shows an example of progressive video frame propagation for lite variants.



Lite variants have no metapackets so TUSER[1] is always low for progressive video. The video packet for the first line of each frame is indicated by TUSER[0].

Full variants add image information and end of field packets to describe the frame properties and mark the end of each frame.

Figure 18. Video frame transmission (full variants)



Related Information

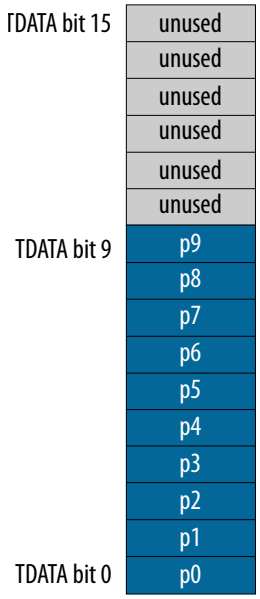
Lite versus Full Variants

2.2.1. TDATA Pixel Packing

Intel FPGA streaming video packets should follow these rules:

- Ensure pixels comprise between 1 and 4 symbols of pixel data.
- Pack video packet data across the TDATA bytes with the LSB of the first symbol of the first pixel in bit 0.
- Byte align each pixel when packing multiple pixels in parallel.
- When a pixel does not perfectly fill a given number of bytes, pad MSBs with undefined data.
- For less than 16 bits of pixel data (for example single pixel 10 bit mono video), pad bits from the MSB of pixel data to bit 15 with undefined data

Figure 19. Example – 10 bit mono data packed to Intel FPGA streaming protocol minimum TDATA width



Parameterize video pipelines using the protocol for a specific number of bits per symbol, pixels in parallel, and color space. These parameters specify the size of the TDATA and TUSER buses.

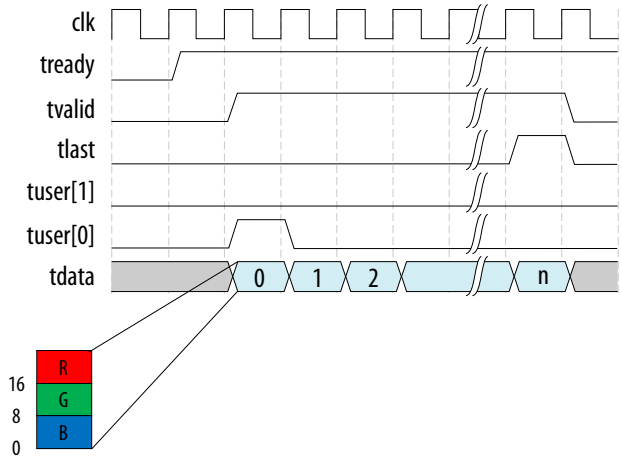
The protocol also allows for video data of different color spaces to propagate down compatible pipelines by virtue of the CSP and SubSa fields in image information packets.

2.2.2. RGB Pixel Packing

The Intel FPGA Streaming Video Protocol specifies a packing scheme for RGB pixels.

Figure 20. 1 pixel in parallel, 24 bit RGB video packet

The figure shows a video packet with one pixel in parallel and RGB24 pixels.



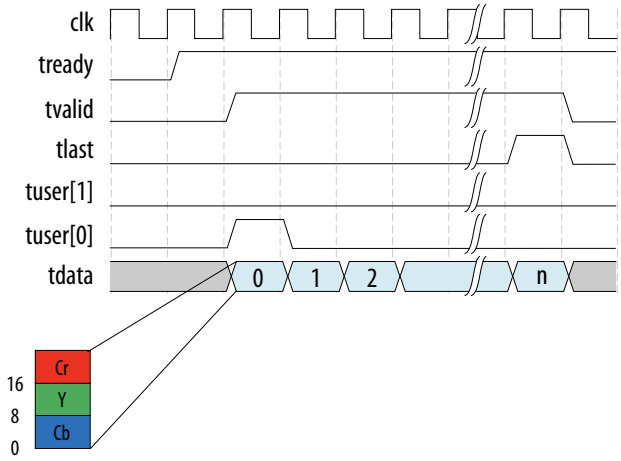
The least significant symbol of an RGB pixel must always be blue. With 8 bits per color plane, each RGB24 symbol occupies one byte of TDATA, TDATA is 24 bits wide and no padding is required.

2.2.3. YCbCr 444 Pixel Packing

The Intel FPGA streaming video protocol specifies a packing scheme for YCbCr 444 pixels.

Figure 21. 1 pixel in parallel, 24 bit YcbCr444 video packet

The figure shows the same video packet but with YCbCr24 444 pixels.



YCbCr pixels are always packed with Cb as the least significant symbol, then the Y symbol and then the Cr symbol.

Cr is the red-chroma information of a YUV stream, which aligns to the red channel information for RGB video. With this ordering, if a YUV image displays accidentally on an RGB monitor, the colors appear almost correctly.

The luma component aligns with the RGB green, as it is the main luma contributor for some algorithms.

Often in math algorithms green is the primary luma contributor when determining brightness of an RGB stream, so keeping these buses aligned is critical.

2.2.4. YCbCr 422 Pixel Packing

The Intel FPGA streaming video protocol specifies a packing scheme for YCbCr 422 pixels.

Note: YCbCr 422 video packets must be an even number of pixels in length to ensure complete pairs of chroma information are transported.

Figure 22. 8 bit YCbCr 422 video packet 1PIP

The figure shows for 8 bit YCbCr video, TDATA is 16 bits wide, which is the minimum protocol width.

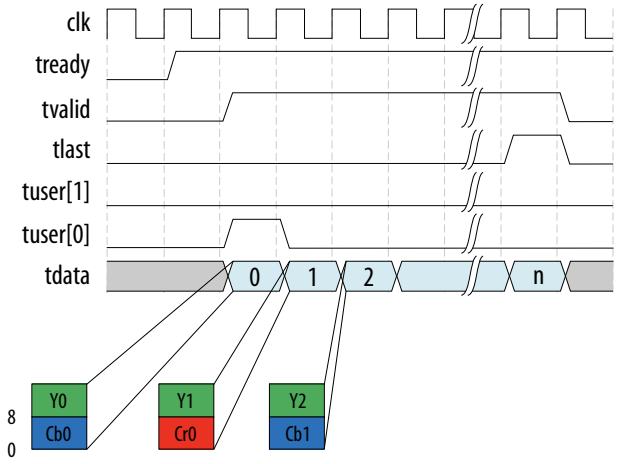


Figure 23. 10 bit YCbCr 422 video packet 4 PIP

In this figure, each pixel comprises a Y (luma) value and a chroma value (either blue or red). Ten bits per symbol requires four bits of padding between pixels so that each pixel (luma and chroma pair) aligns with a byte boundary

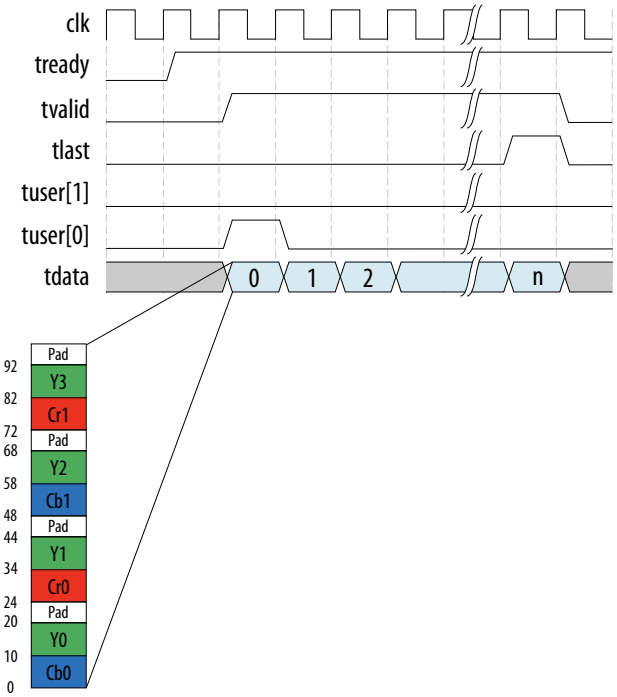
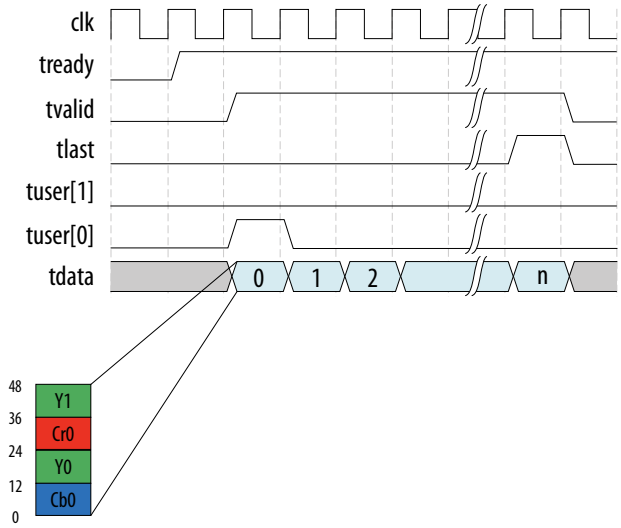


Figure 24. 12 bit YCbCr 422 video packet 2 PIP

In this figure, each pixel comprises a Y (luma) value and a chroma value (either blue or red). Twelve bits per symbol gives a packing that aligns each pixel with a byte boundary so requiring no additional padding..



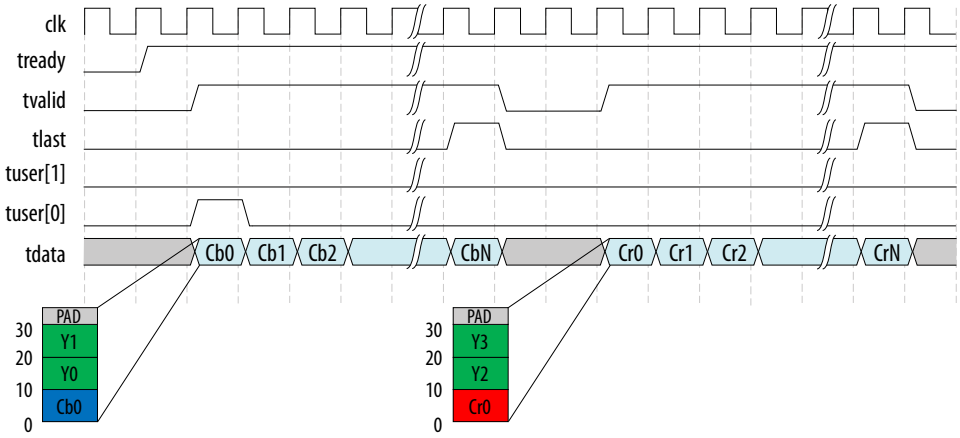
2.2.5. YCbCr 420 Pixel Packing

The Intel FPGA streaming video protocol specifies a packing scheme for YCbCr 420 pixels with 2 pixels in parallel.

Note: YCbCr 420 video fields must always be an even height to ensure complete pairs of chroma information are transported.

Figure 25. 10 bit YCbCr 420 video packet

The figure shows an example for 10 bit YCbCr 420 video, with two pixels in parallel.



The first video packet carries line 0 with Cb data in the chroma symbol and the second video packet carries line 1 with Cr chroma data. In this example, pixels do not perfectly fill a given number of bytes so the 2 MSBs are padded with undefined data.

2.2.6. Four-Channel Video Pixel Packing

The Intel FPGA streaming video protocol specifies a packing scheme for four-channel video. Typically, four-channel video uses the fourth channel for an alpha or transparency layer.

Figure 26. 8 bit RGBA video packet

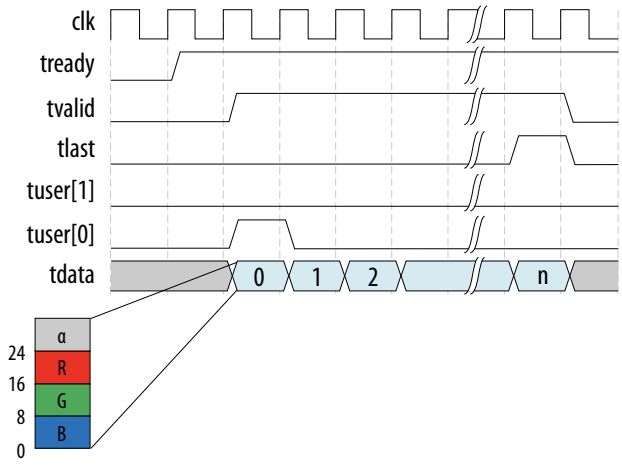
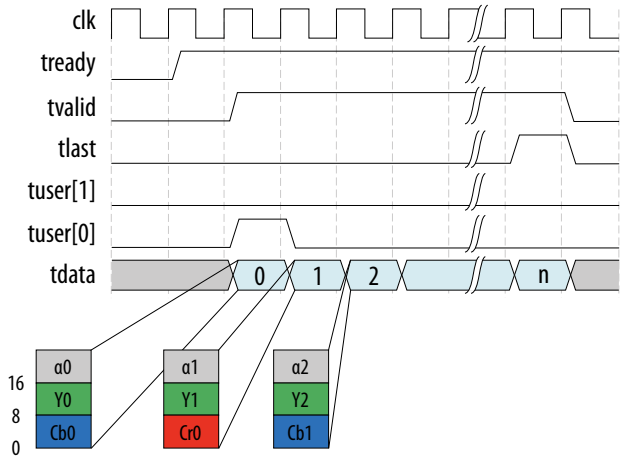


Figure 27. 8 bit 422 YCbCrA video packet

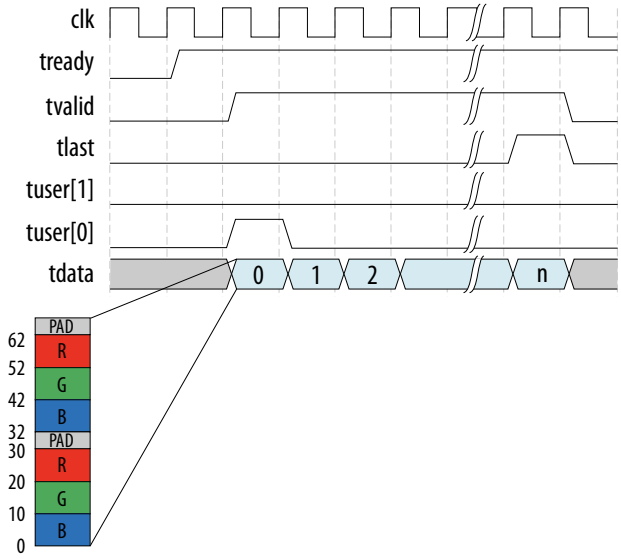


2.2.7. Packing with Multiple Pixels in Parallel

The Intel FPGA streaming video protocol specifies a packing scheme for multiple pixels in parallel.

Figure 28. 2 Pixels in Parallel, 30 bit RGB video packet

The second pixel begins at bit 32 to maintain byte-alignment.



The figures show more examples of pixel packing for 1, 2 and 4 pixels in parallel.

Figure 29. TDATA RGB Layout: One pixel in parallel configuration

Numbers refer to pixel LSBs

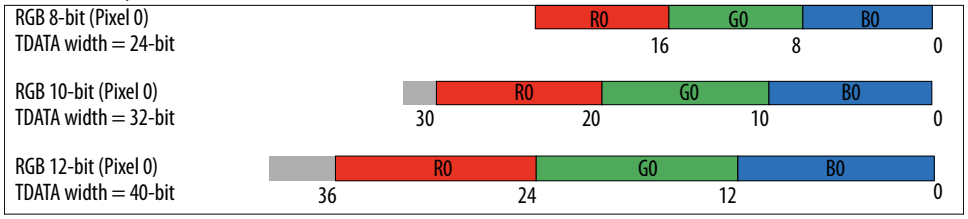


Figure 30. TDATA RGB Layout: Two pixels in parallel configuration

Numbers refer to pixel LSBs

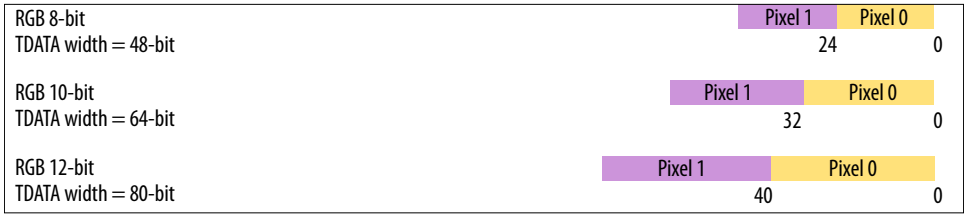
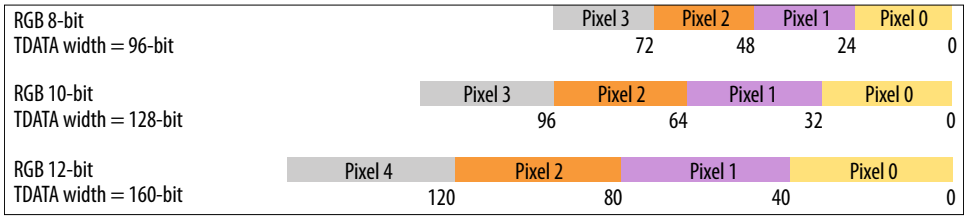


Figure 31. TDATA RGB Layout: Four pixels in parallel configuration

Numbers refer to pixel LSBs

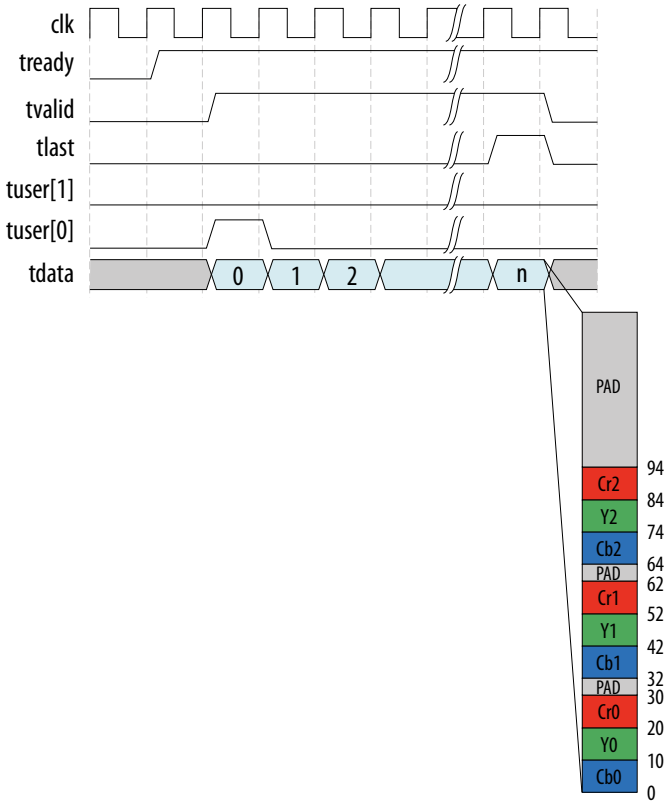


2.2.8. Multiple Pixels in Parallel and Empty Pixels

The Intel FPGA video streaming protocol does not use the AXI-S TKEEP or TSTRB signals, so all bytes within a packet are valid. Where video lines are not an integer multiple of the numbers of pixels in parallel, and empty pixels exist, the final beat of video packets is padded with unused data.

Figure 32. End-of-field pixel packing

4 pixels in parallel, 30 bit YCbCr, empty last pixel. The figure shows an example 4 pixels in parallel system where a line of 1920x1080p HD video has had the final pixel cropped and is now 1919 pixels wide. The final beat of the packet (beat *n*) is padded with unused data.

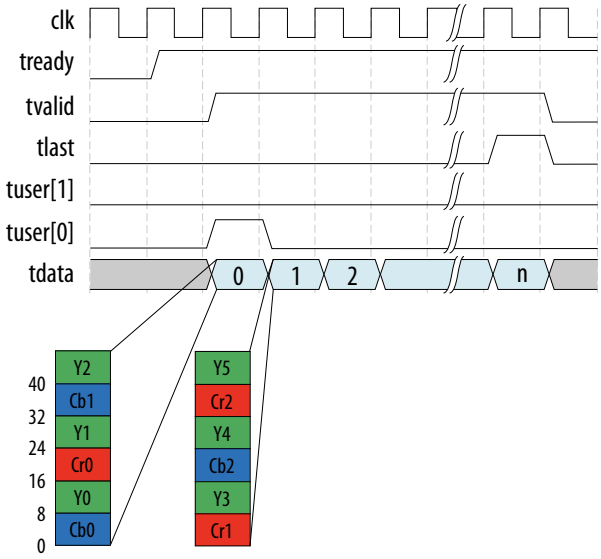


Padding because of empty pixels always occurs in the last beat of the packet. The first beat of the packet is always full of pixels, apart from the case of very short lines where the first beat is also the last beat. In the full variant of the protocol, the expected width of each video line is transmitted via image information control packets and so a receiving IP disregards the top bits of the final beat of the packet.

2.2.9. YCbCr 422 Video with Multiple Pixels in Parallel

The protocol allows YCbCr 422 data with odd numbers of pixels in parallel. For odd numbers of 3 or more pixels in parallel, the symbol alignments for Cb and Cr alternate on each beat of the bus. Ensure your system IP can process this alignment.

Figure 33. 3 pixels in parallel, 8 bit YCbCr 422 video packet showing alternating chroma symbol alignments



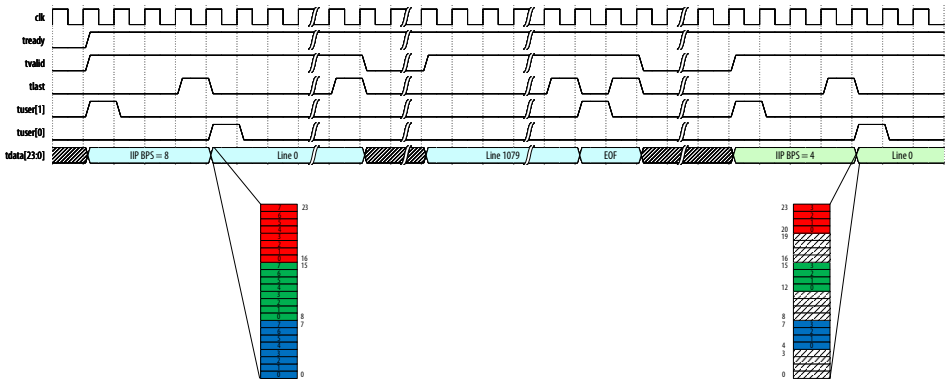
2.2.10. Packing RGB444 onto an RGB888 Interface

The protocol imposes no bounds on the BPS values it supports. However, any physical implementation of the protocol necessarily imposes restrictions on those BPS values.

You can only size a physical AXI to suit a particular BPS. However the protocol allows for transport of video with lower BPS values by specifying the lower BPS in the image information packet and packing the video into the MSBs of the `tdata` bus.

Figure 34. A physical AXI with a 24 bit native tdata bus

The figure shows a physical AXI implementation for video with BPS=8. A second frame of RGB444 (BPS=4) video follows a frame of RGB24 video. Each frame's preceding image information packet specifies the BPS for that frame. `tdata` holds the MSBs for each symbol in the MSB positions for the native 8-bit video symbols. In this figure, they are bits 7, 15 and 23.

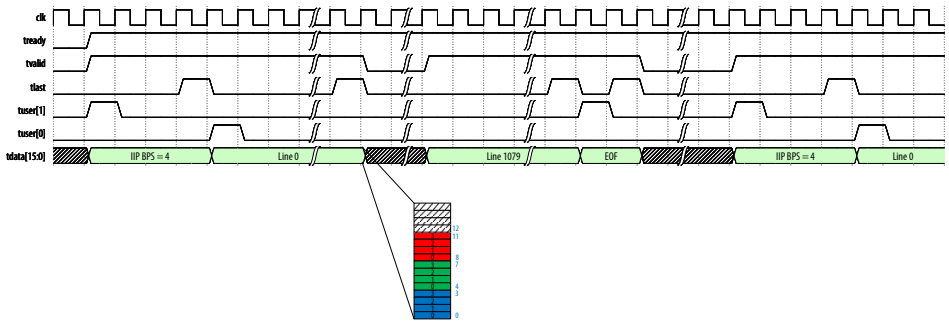


2.2.11. Packing with Less than 8 bits per Symbol Natively

You should always size the Intel video streaming interface for the largest BPS that you want to support.

If RGB444 is the largest video data that you transmit over the Intel video streaming interface, you need not space the data bits as in the previous figure. You can size the `tdata` bus to accommodate the 12 bits of video data plus another 4 bits of padding to satisfy the minimum tdata bus width requirement of 16 bits. The video data packs naturally.

Figure 35. Video data packing naturally



The standard rules for pixel packing still apply. A 2 pixels-in -parallel interface of RGB444 requires you to pad bits 15 to 12 to meet the requirement that each pixel begins on a byte boundary. Therefore, the total `tdata` size for a 2 PiP RGB444 interface is 28 bits: 16 bits for the first 12 bits of pixel data plus the 4 padding bits plus another 12 bits for the second pixel. You do not need to pad this second pixel as it is over the 16 bits minimum tdata width requirement and you are not packing a third pixel above the second one.

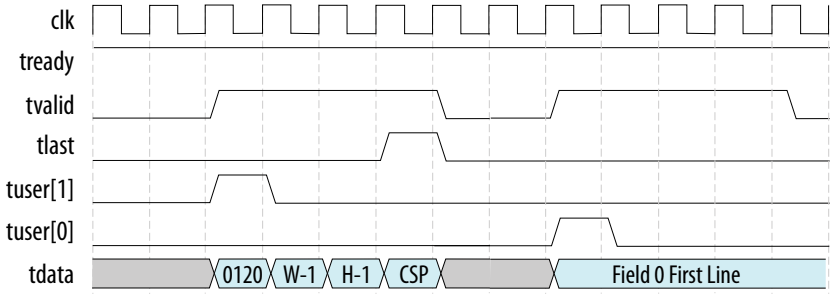
2.2.12. Interlaced Fields

Drive interlaced fields in the same way as progressive frames. The only difference is the nibble coding for full variants of the Intel FPGA streaming video protocol and the TUSER[1] signal for lite variants. Transmit interlaced fields in pairs so that an F1 field follows an F0 field or vice versa. You may drive interlaced fields of the same type consecutively but IPs receiving the fields need to act accordingly.

Full Variants

The image information packet specifies the field type via the interlaced nibble bits in the first beat.

Figure 36. Example interlaced field (full variants)



Lite Variants

For the lite variant of the protocol, the following figures show TUSER[1] defined for all valid cycles in the packet that indicates F1 fields. TUSER[1] is high for F1 fields and low for F0 fields.

Figure 37. Intel FPGA streaming video packet (start of F1 field)

Lite variants only.

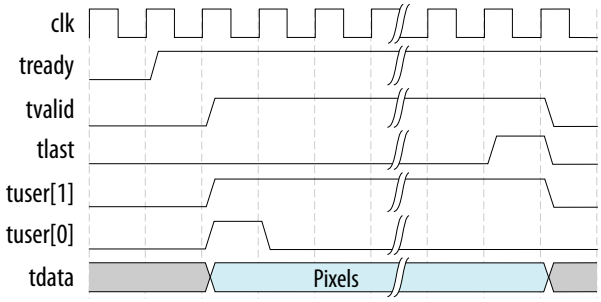


Figure 38. Intel FPGA streaming video packet (F1 field)

Lite variants only.

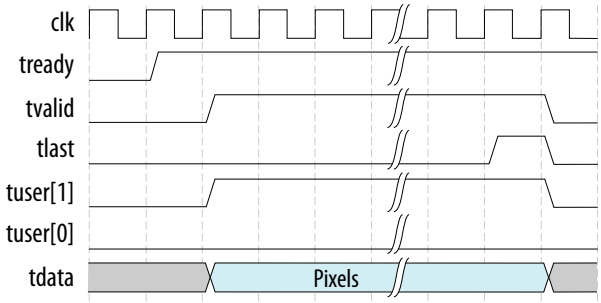


Figure 39. Intel FPGA streaming video packet (start of an F0 field or progressive frame)

Lite variants only.

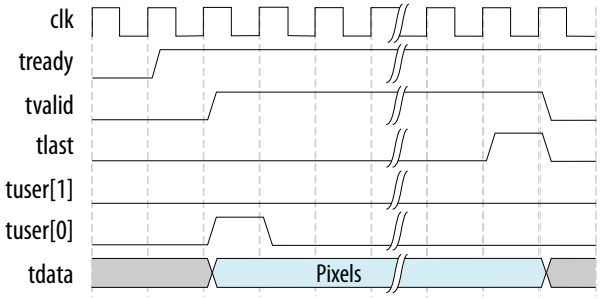


Figure 40. Intel FPGA streaming video packet (F0 field or progressive frame).

Lite variants only.

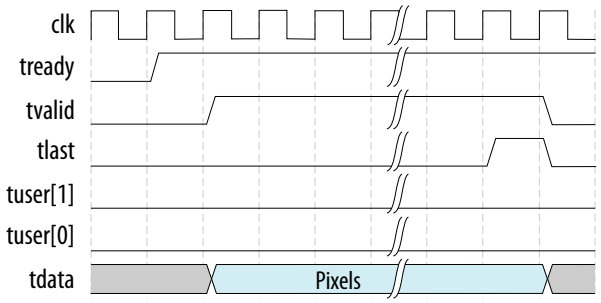


Table 8. Lite variant TUSER bits decode

TUSER[1:0]	Meaning
00	Video packet (from either an interlaced F0 field or progressive frame).
01	Video packet (start of either an interlaced F0 field or progressive frame).
10	Video packet (from an F1 field).
11	Video packet (start of F1 field).

3. Intel FPGA Streaming Video Protocol Rules

The rules apply to full variants only. Lite variants do not require any additional rules on the order of packets because it only has video packets.

3.1. Rules for Packets

The Intel FPGA streaming video protocol has various rules for metapackets and video packets.

The following rules apply to all packets (the numbers refer to the numbers in Illegal packet ordering examples:

- A set of video packets must always follow image information packets. No other packet type can follow an image information packet unless the set of video packets is empty. [1]
- Each set of video packets can only have one image information packet. [2]
- Either a set of video packets, an image information packet (if the set of video packets is empty), or a set of auxiliary control packets must precede end-of-field packets. [3]
- Each set of video packets can only have one end-of-field packet. [4]
- Video packets within a set must not have other packet types between them. [5]
- Auxiliary control packets may occur anywhere apart from between an image information packet and the final video packet of the set. [6]
- Commit and register update packets must be between end-of-field and image information packets. [7]
- The protocol does not allow partial or malformed metapackets.

Figure 41. Legal packet ordering examples

The figure shows some examples of legal packet placements.

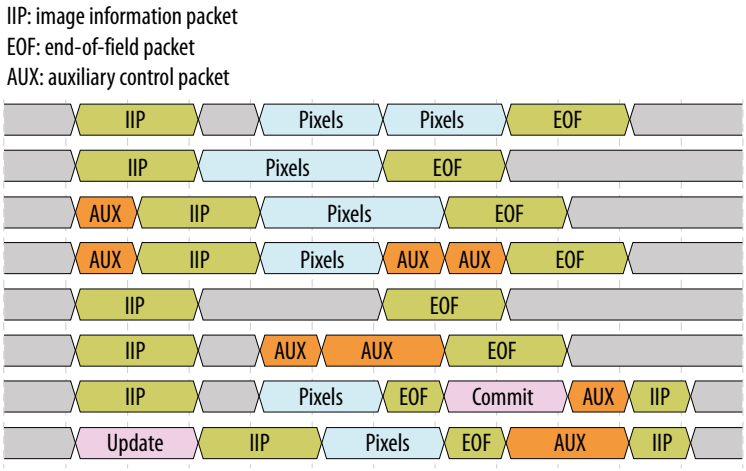


Figure 42. Illegal packet ordering examples

The figure shows some examples of illegal packet placements. The numbers refer to the rules.

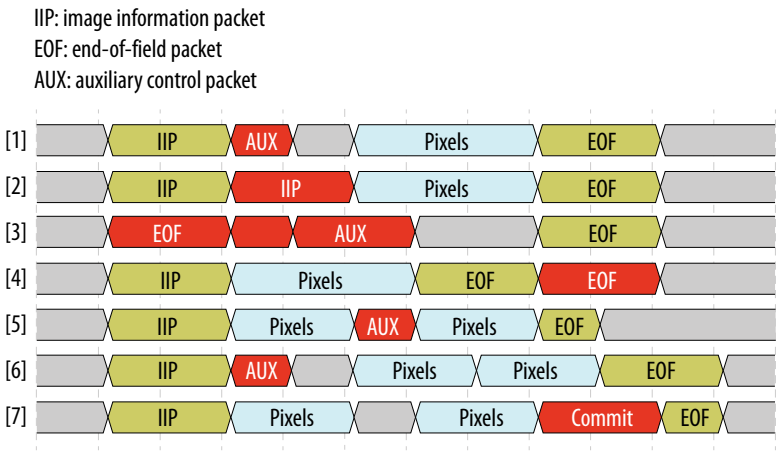
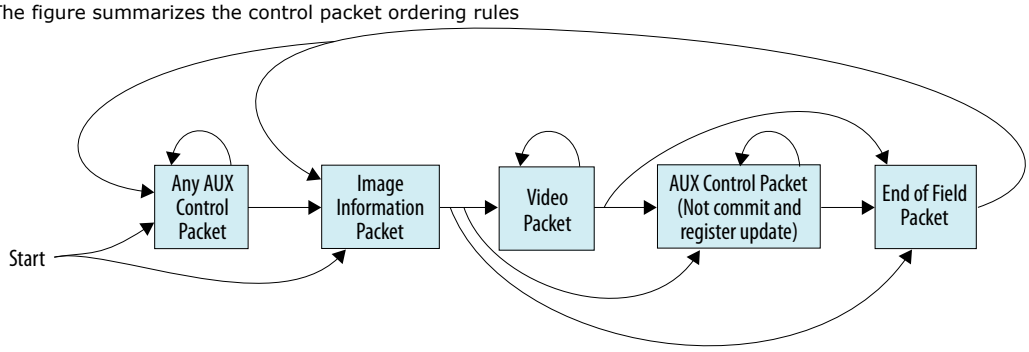


Figure 43. Packet ordering rules

The figure summarizes the control packet ordering rules



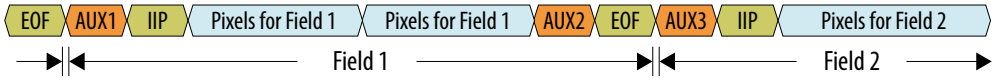
3.1.1. Auxiliary Control Packet Rules

To conform to the Intel FPGA streaming video protocol, IPs must follow these rules:

- IPs must propagate all auxiliary control packets unchanged unless the semantics of the packet indicate that it should terminate at that IP.
- IPs must not change the order of any auxiliary control packets unless this order is an intended part of the functionality for that IP.
- IPs that generate one field of video for every field received must retain the count and field count quantities, unchanged, in the image information packets and end-of-field packets respectively.

Figure 44. Auxiliary Packet Rules

The figure shows aux1 and aux2 associate with field 1 and aux3 associates with field 2.



Pairs of image information and end-of-field packets mark one field of video. Any auxiliary control packets before the current field's end-of-field packet are associated with that field. In addition, any auxiliary control packets that occur after the end-of-field packet of the preceding field are also associated with the current field.

These rules inform systems how to process any associated auxiliary control packets when switching or buffering video fields.

3.2. Rules for IPs

To support the Intel FPGA streaming protocol, IP must meet the following guidelines:

- IPs must be compliant to the AXI4-S protocol.
- IPs must support the full, lite or both modes of the protocol.
- IPs can only support the modes of the protocols as described in the *Intel FPGA Streaming Video Protocol Specification*.
- If receiving an end-of-field packet with a set broken bit, IPs must also output end-of-field packets with the broken bit set, unless the IP fixes the field.
- Full variant IPs must propagate unsupported auxiliary packets unmodified and with packet ordering unchanged.
- Full variant IPs must process all image information and end-of-field packets and extract and use that video field information.
- Full variant IPs must propagate any commit, timestamp, and any other auxiliary control packets, unless the auxiliary control packet's ultimate destination is that IP.
- Full variant IPs must set the broken bit in the outgoing end-of-field packet for any relevant fields. For example, fields break when you request a clipper IP clips all video fields down to a width of 1000 pixels, but incoming fields are of width 900 pixels.
- Lite variant IPs exhibit undefined behavior if receiving full variant packets.

Protocol Errors

This protocol mandates no specific behaviors for IPs using the Intel FPGA streaming video protocol for handling malformed packets or protocol errors on incoming interfaces.

Related Information

Lite versus Full Variants

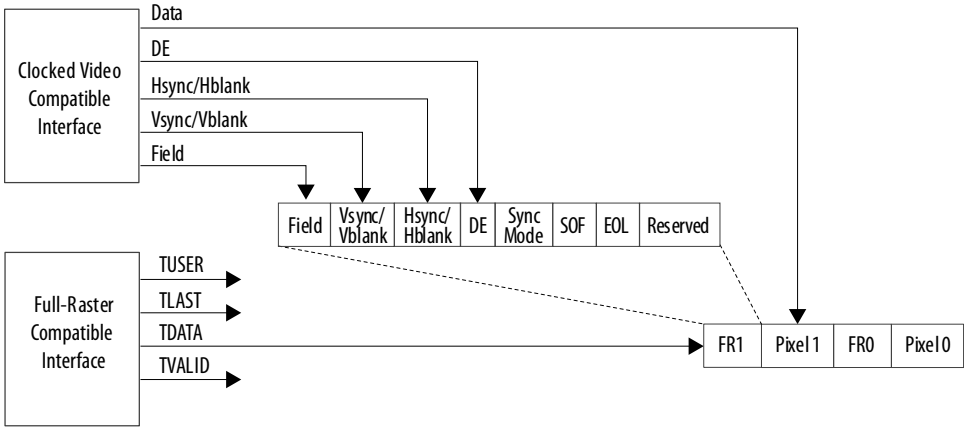
4. Intel FPGA Streaming Video Full-Raster Protocol

The full-raster protocol adds clocked video data to the lite variant of the Intel FPGA streaming video I/O.

Sometimes the off-screen parts of video frames are irrelevant. For most cases the streaming video protocol only preserves active video data and filters out all off-screen data. However, these areas of blanking may carry significant amounts of side-band data, such as audio, timestamps, HDR (High Dynamic Range) information, and any other metadata related to a specific video protocol.

Figure 45. Mapping video data and timing markers from clocked video to Intel FPGA streaming full-raster protocol

The figure shows how the full-raster protocol supports full-raster video on the lite variant of the Intel FPGA streaming video protocol. Video timing markers are traditionally explicitly exposed as part of a clocked video I/O interface (e.g., DE, Field, Vsync/ Vblank, and Hsync/ Hblank). The full-raster protocol takes video timing data markers and embeds them at the top of the TDATA payload of an Intel FPGA streaming video protocol, to allow for distribution and processing of full-raster video formats.



Embedding full-raster data into the lite variant of the Intel FPGA streaming video protocol means that you can use AMBA AXI4-Stream agnostic IPs to move, capture, or generate full-raster data. For example, cross-points, video switch, DMA engines, PCI-Express transfer, Ethernet transcoding. None of these IPs need to know the content in the TDATA stream, they just need to move it.

4.1. Full-Raster Frame Structure

Generally, video timing parameters are based on two concepts: blank timing, and sync timing. Each connectivity protocol uses one of these methods to define a full-raster video frame, e.g., SDI uses blank timing and HDMI uses sync timing.

For sync-timing systems you can have:

- Hsync – horizontal synchronization pulse
- Vsync – vertical synchronization pulse

For blank-timing systems you can have:

- Hblank – horizontal blanking interval
- Vblank – vertical blanking interval

Several video technical organizations, such as the Video Electronics Standard Association (VESA), and the Consumer Electronics Association (CEA), define full-raster video timing parameters. For example, the selected video standard (e.g., SMPTE 424, CEA-861-D and VESA) or custom timing parameters programmed in the host interface define the width and timing of the Hsync, Hblank, Vsync and Vblank signals. The full-raster protocol adheres to the industry norm.

Usually, the active video data is geometrically in between the horizontal and vertical blanking areas. Both vertical and horizontal blanking areas are in four distinctive groups:

- Front porch
- Sync pulse
- Back porch
- Active video

The figures superimpose full-raster streams on top of a video frame raster, The figures show where the IP injects full raster timing information (FRn), ancillary (An) data (Pn) and active video data into the payload of the TDATA bus.

Figure 46. Timing for a Progressive Video Image

The figure shows an example of a progressive video. The vertical dashed red line indicates the end of the video line. TLAST is asserted here.

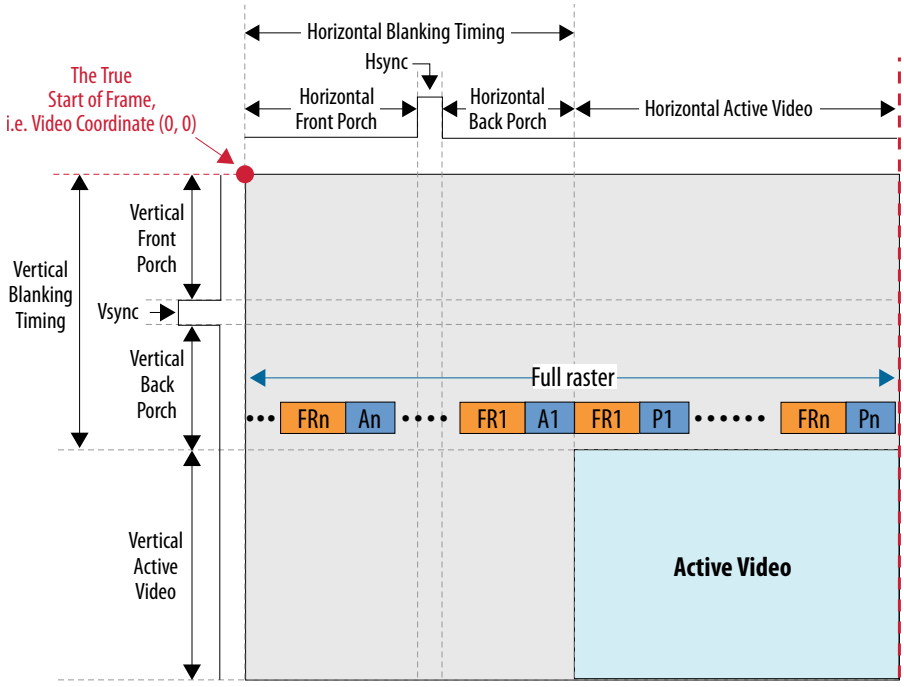
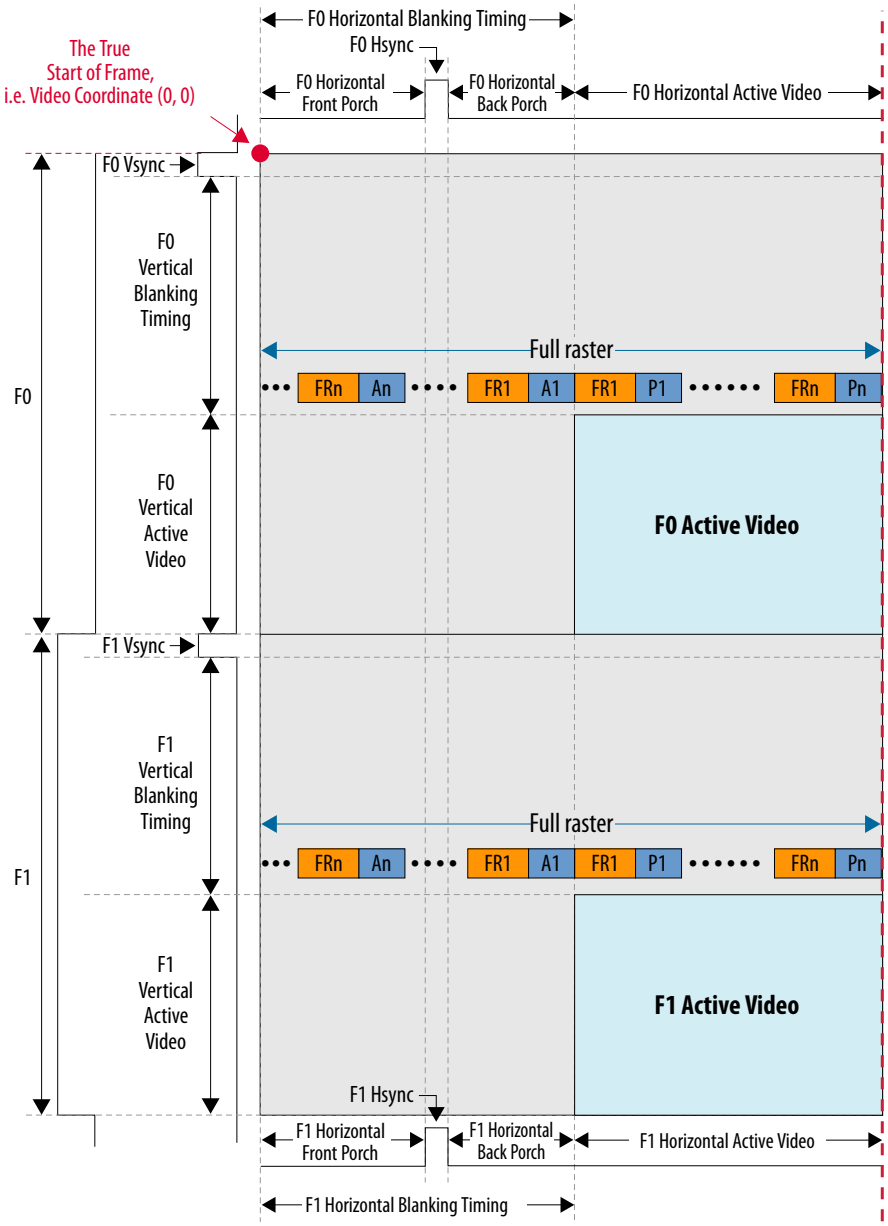


Figure 47. Timing for an Interlaced Image

The figure shows an example of a interlaced video. The vertical dashed red line indicates the end of the video line. TLAST is asserted here.



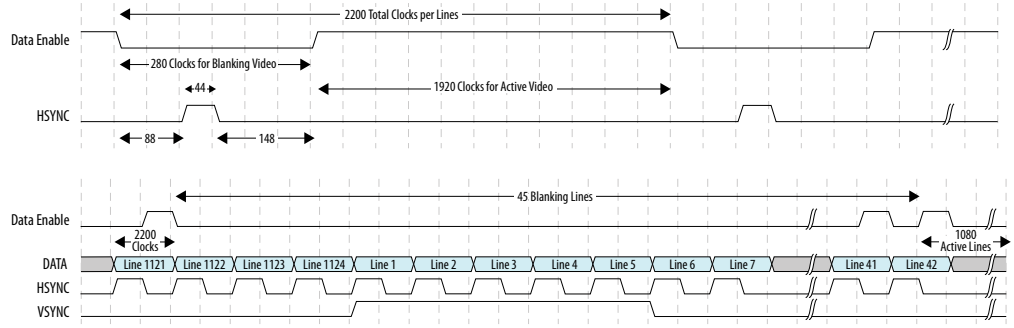
4.2. Full-Raster Start of Frame

The Intel FPGA streaming video full-raster protocol uses a combination of TUSER[0] signal and an embedded sideband start of frame bit field on the TDATA bus, to indicate the position of the start of frame. Where the streaming full-raster interface processes multiple pixels per clock in parallel, the TUSER[0] signal is asserted high for one valid transaction. The start of frame associated with each of the pixels transported on the TDATA bus indicates the exact position of the first pixel on the full

raster. The start of frame association is important where it might not be in pixel 1 of the data bus, because of a combination of full-raster data width and the number of pixels in parallel configured on the full-raster interface.

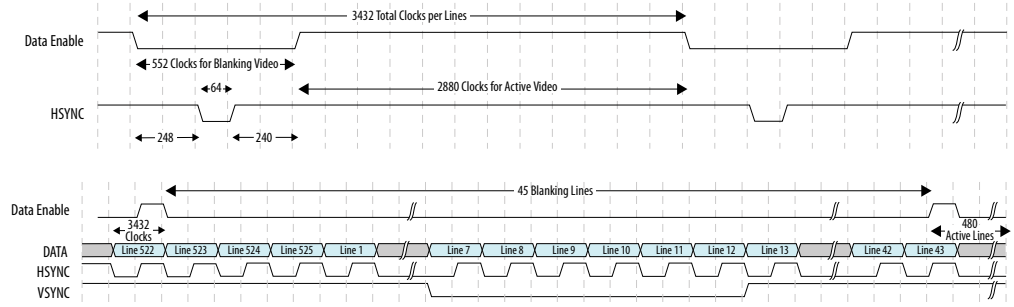
Figure 48. Sync-Timing Mode: Timing Diagram for 1920x1080p at 60 Hz

The figure shows CEA-861-D video timing definition for 1920x1080p60, which follows the true start of frame convention.



For most standardized video resolutions the leading edge of the VSYNC pulse coincides with the true start of the frame boundary where video geometry counters are in coordinates (1,1). One exception is where the leading edge of VSYNC is usually offset by a certain amount of video lines, causing the VSYNC to misalign with the actual true start of frame. For instance, 720x240p60 VSYNC pulse is offset by 3 video lines, while 2880x480p60 is offset by 7 video lines. The position of the leading edge of VSYNC relative to the true start of frame location is defined in the relevant standards, such as SMPTE, VESA, and CEA. The full-raster protocol states you assert TUSER[0] at video coordinates (1,1), irrespective of where the actual leading edge of VSYNC occurs.

Figure 49. Sync-Timing Mode: Timing Diagram for 2880x480p at 60 Hz



4.3. Full-Raster TDATA Signal Layout

The full-raster protocol adds *n* bits of payload sideband signaling to each pixel into the TDATA bus. This timing markers sideband data is a full-raster stream. The size of a full raster stream requires the same number of bits as defined for a pixel with a single-color plane (bits per symbol). You define the full-raster stream data width to match a color plane per symbol. You treat the full-raster stream as an extra or fake color plane for each pixel, to calculate the total TDATA bus width. The full-raster protocol follows similar equations and rules proposed by the Intel FPGA streaming video protocol, which uses the number of pixels in parallel to determine the size of the TDATA bus.

Full-raster TDATA byte width = max(2, ceil ((bps × (SYM + 1))/8)) × PIP

Where:

- BPS is bits per symbol
- SYM is number of color planes
- PIP is pixels in parallel

Figure 50. Streaming Full Raster Code Snippet to Calculate TDATA bus width

The figure shows an example code snippet to show you can calculate the full-raster TDATA bus width by increasing the original number of color planes by 1. The full-raster stream payload in the TDATA bus conveys full-raster timing markers information for each pixel.

```
module intel_vvp_axis_fr_generic_ip
#(
 parameter PIP      = 4,  // Pixels in Parallel (PIP)
 parameter SYM      = 3,  // Number Color plane per Symbols
 parameter BPS      = 12,  // Bits Per Symbol
 parameter AXIS_FR_WIDTH = (((((BPS*(SYM+1))+(7))/(8))*(8))*(PIP))
)
(
          .
          .
  input logic [AXIS_FR_WIDTH -1 : 0] axi4s_fr_in_tdata,
  output logic [AXIS_FR_WIDTH -1 : 0] axi4s_fr_out_tdata,
          .
          .
);
endmodule
```

Figure 51. Intel FPGA Streaming video lite-mode vs full-raster TDATA Formats

The figure shows an example of the TDATA bus for both lite variants and full-raster protocols. Both carry the same video data information. The data packet carrying full-raster information has appended extra streams on the TDATA bus to include timing video information per pixel: FR 0 for pixel 0 and FR 1 for pixel 1.

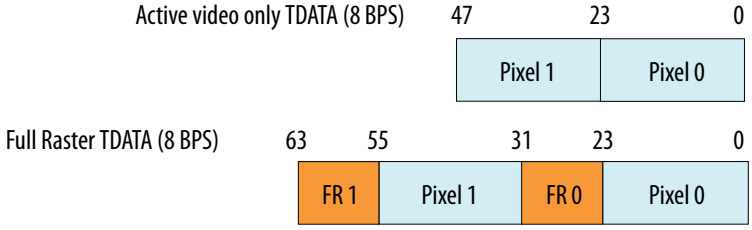
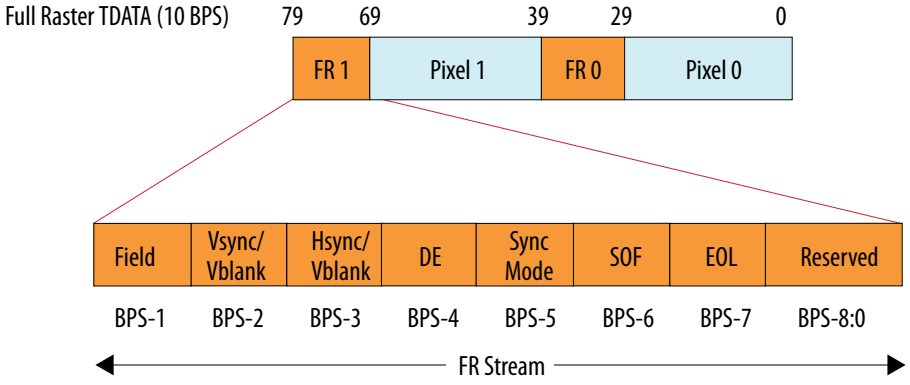


Figure 52. Streaming Full Raster TDATA Format

The figure shows the full-raster stream stacked on top of the pixel data such that FR 0 relates to Pixel 0 and FR 1 relates to Pixel 1. If the bus is wider, this algorithmic stacking of pixels and full-raster stream data continues.



The full-raster protocol encodes pixels and their associated timing information by stacking them across TDATA. The first pixel is placed in the lowest bit position. All pixels are byte-packed to comply with the byte granularity of AXI4-S specification. When needed, some padding bits in the full-raster stream ensure byte-alignment.

The table shows the internal structure of a full-raster stream packet, where:

- Bit (BPS-7) is an end of line bit to indicate streaming full-raster end of line, and its value should coincide with the TLAST signal.
- Bit (BPS-6) is a start of frame bit to indicate streaming full-raster start of frame, and its value should coincide with the TUSER[0] signal.
- Bit (BPS-5) is a sync mode bit to indicate which timing methods (sync-timing or blank-timing) are transported.
- Bit (BPS-4) contains data enable information.
- Bits (BPS-3) and (BPS-2) provide horizontal and vertical timing information, respectively. Bits (BPS-4) and (BPS-3) work with bit (BPS-5), as sync-timing and blank-timing modes are mutually exclusive.
- Bit (BPS-1) carries information about the field sync that defines the start of a field on a video frame. In case of interlaced format, Bit (BPS-1) indicates whether the incoming data belongs to either the first (F0) or second (F1) field. For progressive format, bit (BPS-1) is tied-off to logic-low all the time.

Table 9. Full Raster Stream Payload Encoding

Full-raster Bit	Name	Description
BPS-1	Field	'0' = first field (F0) for interlaced or Progressive format '1' = second field (F1) for interlaced format
BPS-2	V-sync or V-blank	Indicates vertical blanking.
BPS-3	H-sync/ H-blank	Indicates horizontal blanking.
BPS-4	Data enable	Indicates active portion of video image.
BPS-5	Sync mode	'0' = sync fields are valid '1' = blank fields are valid
BPS-6	Start of frame	Indicates full-raster true start of frame where video geometry counters are in coordinates (1,1), irrespective of where the actual leading edge of VSYNC occurs. The value of this field should coincide with TUSER[0] signal value.
BPS-7	End of line	Indicates full-raster end of line, which is the last pixel on each full rater video line. The value of this field should coincide with TLAST signal value.
BPS-8:0	Reserved	Reserved.

The full-raster stream payload aligns with the most significant bit (MSB). Its data is always in the same place regardless of the number of bits per sample of the defined Intel FPGA streaming video pixels size. The MSB alignment also allows bits per sample conversion without affecting or breaking the internal structure of the full-raster stream payload. The rest of the bits in the full-raster stream payload are reserved, and you should ignore them as they do not carry meaningful information. You should pad them with logic-zero values.

The figures show how the full-raster stream looks when configured for a specific sync mode.

Figure 53. FR Stream with Sync-Timing

The figure shows valid fields for sync-timing.

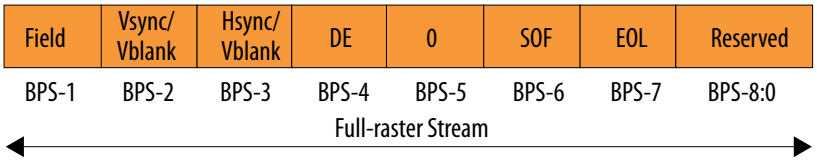
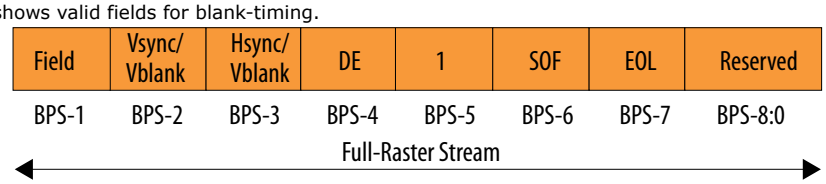


Figure 54. FR Stream with Blank-Timing

The figure shows valid fields for blank-timing.



Both blank and sync timing can exhibit dual polarities as defined by video standards. To support the different variants of sync-timing and blank-timing polarities, an optional processor register interface should allow user-control to define the polarity (active high or low) for the following video timing signals:

- Vsync or Vblank
- Hsync or Hblank

4.4. Full-Raster TREADY Signal

4.4.1. Full-Raster TREADY Handshake Process

AMBA AXI4-S protocol specifies that the TVALID and TREADY handshake determines when information passes across the AXI4-S interface. For more information, refer to *Data Exchange*.

A two-way flow control mechanism enables both the transmitter and receiver to control the rate at which the data and control information is transmitted across the interface. For a transfer to occur both the TVALID and TREADY signals must be asserted. Either TVALID or TREADY can be asserted first, or both can be asserted in the same ACLK cycle.

Related Information

Data Exchange on page 4

4.4.2. Optional TREADY Signal: Receiver Interface

The Intel FPGA streaming full-raster video protocol follows the AMBA AXI4-S protocol recommendation.

It states that TREADY is an optional signal that you can omit in certain circumstances if the receiver interface can always accept a transfer. However, if TREADY is not supported as part of the full-raster interface, but the actual signal cannot be removed from the receiver interface, drive TREADY high all the time.

4.4.3. Optional TREADY Signal: Transmitter Interface

For a transmitter interface, omitting the TREADY signal indicates that the interface cannot accept backpressure.

If TREADY is not on the full-raster interface, the interface assumes TREADY coming from the receiver interface is high all the time. If a full-raster interface cannot accept backpressure and the TREADY signal cannot be removed from the transmitter interface, you can use the TREADY signal to indicate an error condition if you drive TREADY low during a transfer. By including the TREADY signal in the interface, it enables you to correctly identify the source of the error.

4.4.4. Optional Backpressure

Most of the connectivity video IPs such as SDI, HDMI and DisplayPort do not support backpressure and you can omit or drive high TREADY.

These modules cannot support backpressure because this type of connectivity IP works on a live video stream and uses a synchronous pixel clock rate to drive the video interface. Hence, any TREADY backpressure breaks the video timing relationship.

However, other connectivity IPs have a data packet exchange based on command driven protocols, such as PCIe and Ethernet. Backpressure requires TREADY to process the AXI4-S flow control mechanism and the full-raster compatible interfaces to support an optional TREADY signal. To analyze the effect TREADY has on the backpressure that a transmitter full-raster interface can experience during data transfer, look at the relationship of pixel clock rate between the video interfaces that exchange video data. The two scenarios are:

- Synchronous pixel clock rates
- Asynchronous pixel clock rates

4.4.5. Synchronous Pixel Clock Rates I/O Interfaces

This type of video interface has both transmitter and receiver video I/Os running based on the same video clock. The full-raster protocol allows you to configure compatible video interfaces with no TREADY signal so that they cannot accept backpressure.

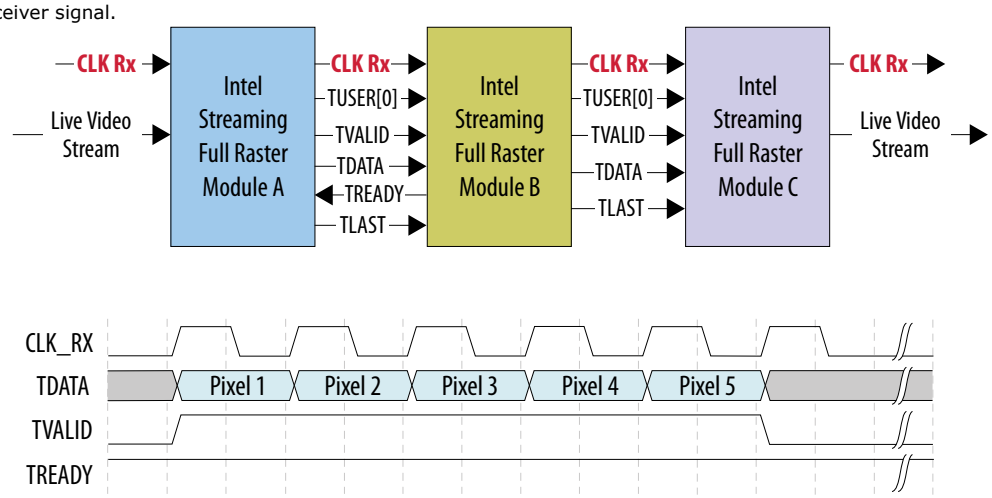
Consequently, the receiver full-raster interface should accept data transfers on the bus in definite time intervals, defined by the sampling video rate, which allows either:

- Removing TREADY from the video interface. It does not need it because of the lack of backpressure support
- Driving TREADY signal high all the time.

Having a streaming full-raster compatible interface that does not support backpressure, allows you to move data between IPs using the native video receiver clock frequency. For example, 148.5 MHz for 1920x1080p60 with 1 pixel in parallel or 297 MHz for 3840x2160p60 with 2 pixels in parallel.

Figure 55. Streaming Full Raster Data Rate Matching: Full-Rate Case

The figure shows a live video stream passing straight from a clocked-video domain to a streaming full-raster domain. All modules run at the same frequency (CLK receiver) and can accept the expected input data rate. None of the modules in the pipeline support backpressure. TREADY is active-high all the time, so Module A, Module B and Module C can accept data transfers on the TDATA bus at the specified data rate dictated by CLK receiver signal.



4.4.6. Asynchronous Pixel Clock Rates I/O Interfaces

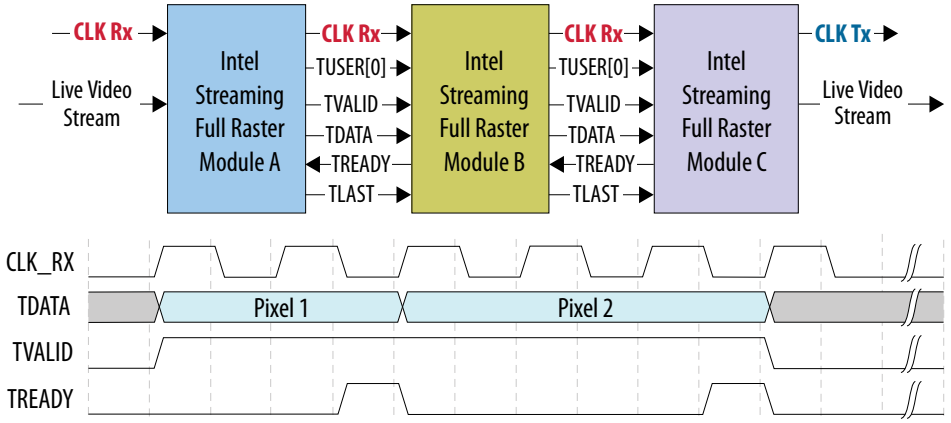
This type of video interface assumes that the transmitter generates video data with a slower or faster pixel clock rate compared to the receiver video interface.

Moving video data between two different clock domains is always challenging. You must apply clock domain crossings (CDC). The TVALID and TREADY flow-controlled AXI4-S video interface can cause control-flow problems on the video datapath, in either of the following two scenarios:

- Clock-skew matching where video receiver or transmitter and video processing clock do not match.
- In-rush data ingestion where one of the modules in the video pipeline cannot offer the necessary throughput to move the data at the expected rate.

In an example where CLK receiver signal drives some streaming full-raster, others are driven by a different clock domain signal, for example, CLK transmitter. Module B and Module C are both transferring data between them. Module C cannot keep up with the data rate imposed by Module B. The TREADY signal driven by Module C drops for a few clock cycles while it accepts the input data. For full-raster interfaces, any bubble inserted in between the TREADY and TVALID handshake process breaks the video timing data rate of the incoming data.

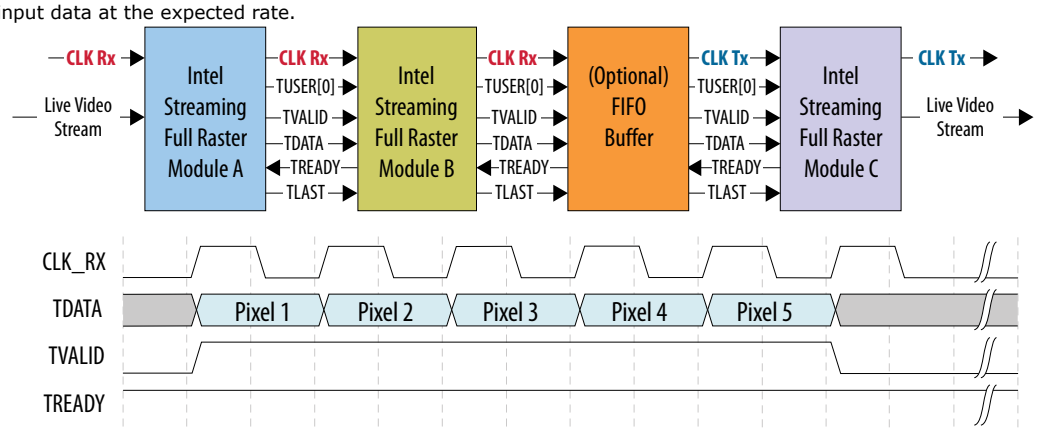
Figure 56. Streaming Full Raster Data Rate Mismatch: Backpressure Example



To accommodate a small amount of data-skew mismatching or in-rush data between streaming full-raster interfaces, compatible interfaces must provide an optional FIFO buffer. A FIFO buffer can accommodate bursts of data if and only if the data rate of the upstream full-raster video component is less than or equal to the data rate of the downstream components.

Figure 57. Streaming Full Raster Data Rate Matching: Full-Rate Case-Scenario

The figure shows an example where an AXI4-S compatible FIFO buffer is instantiated between the Modules B and C. This FIFO buffer temporally takes data that otherwise Module C misses allowing Module B to transfer the input data at the expected rate.

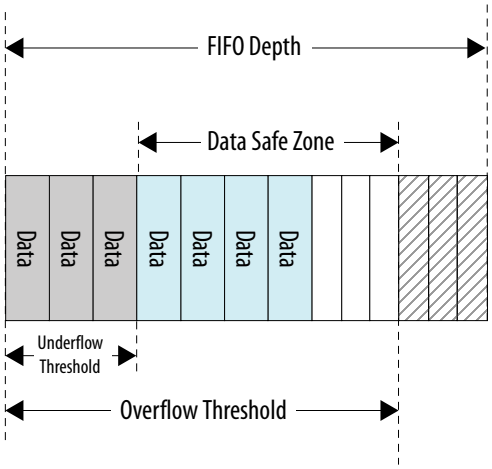


If the full-raster interface allows you to instantiate a FIFO buffer, the full-raster protocol enforces the addition of an overflow error signal to the corresponding processor register map of the IP using this video interface.

The full-raster protocol specifies and defines the way to exchange full-raster video data using a video stream interface. However, you must guarantee that the system can correctly process this data-skew mismatching or in-rush data ingestion between full-raster interfaces.

In your system, the data level in the FIFO buffer should always be above the underflow level and below the overflow threshold. This range is the FIFO data safe zone. An overflow can indicate that the receiver video interface driving the TREADY signal cannot ingest data at the correct data rate and starts creating excessive backpressure on the transmitter interface.

Figure 58. Example of a FIFO Data Safe Zone



4.5. Full-Raster I/O Signals

Table 10. Full-Raster Transmitter Interface Signals

Signal Name	Width	Direction	Default Value	Optional or Required	Description
TUSER[0]	As per Intel FPGA streaming video protocol specification. For TDATA bus, extend the number of color planes by 1, to consider the extra full-raster stream.	Output	Low	Required	Full-raster start of frame. Indicates full-raster true start of frame, i.e., where video geometry counters are in coordinates (0,0), irrespective of where the actual leading edge of VSYNC occurs.
TDATA		Output	-	Required	Data payload
TVALID		Output	-	Required	Indicates that the transmitter interface is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
TREADY		Input	-	Optional	Indicates that a receiver interface can accept a transfer. When TREADY is not present in the transmitter interface, it indicates that this interface does not support backpressure.
TLAST		Output	Low	Required	Full-raster end of line. Indicates the last pixel of the full-raster line.

Table 11. Full-Raster Receiver Interface Signals

Signal Name	Width	Direction	Default Value	Optional or Required	Description
TUSER[0]	As per Intel FPGA streaming video protocol specification. For TDATA bus, extend the number of color planes by 1, to consider the extra full-raster stream	Input	-	Required	Full-raster start of frame. Indicates full-raster true start of frame, where video geometry counters are in coordinates (0,0), irrespective of where the actual leading edge of VSYNC occurs.
TDATA		Input	-	Required	Data payload
TVALID		Input	-	Required	Indicates that the transmitter interface is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
TREADY		Output	High	Optional	Indicates that a receiver interface can accept a transfer. When TREADY is not present in the receiver interface, indicates that this interface can always accept a transfer.
TLAST		Input	-	Required	Full raster end of line. Indicates the last pixel of the full-raster line.

5. Document Revision History for Intel FPGA Streaming Video Protocol Specification

Document Version	Changes
2024.05.15	Corrected *10 bit YCbCr 422 video packet 4 PIP* figure title.
2024.04.15	Added *Packing with Less than 8 bits per Symbol Natively* and *Packing RGB444 onto an RGB888 Interface*
2024.01.11	• Corrected figures in *YCbCr 422 Pixel Packing* • Added more information to *Image Information Packet* `IntlaceNibble` table
2023.11.02	Added two new diagrams to *YCbCr 422 Pixel Packing*
2023.02.21	Added bits per symbol (BPS) minus 1 to *Image Information Packets*
2022.07.25	Added *Intel FPGA Streaming Video Full-Raster Protocol*
2022.02.15	• Deleted *Intel FPGA Streaming Video Errors* • Deleted *Ready and Valid Behavior* • Deleted *Reset* • Deleted text: the IPs also support multiple pixels in parallel • Added four signals to *Protocol Signals* • Changed TDATA in *AXI4-Stream Handshake: TVALID and TREADY asserted at the same time* • Renamed *TDATA Format* to *Packing with Multiple Pixels in Parallel* • Added *Rules for Packets* and *Rules for IPs* • Added *Metapackets*, *Video Packets*, *Auxiliary Packets*
2021.08.02	Initial release.