prediction is done using both forward and reconstructed frame pointers. The field number (first or second) is computed by the MPEG2 video decoder. It may be overwritten by the application with FFN (in VID_PPR). This may be useful for trick mode applications.

**Picture parameters**

These parameters are extracted from the bitstream. They have to be programmed in VID_PPR.

**Decoding task instruction**

Decoding task instructions are programmed in VID_TIS. This register gives options for error recovery and decoding (simplified B or overwrite mode).

**Stream pointers for variable length decoder read access**

The starting position of the encoded picture in the external memory must be programmed in the variable length decoder read pointer register (VID_VLDPTR).

A variable length decoder read limit may be set in register VID_VLDRL. When the read pointer reaches VID_VLDRL (variable length decoder read limit), reading in the buffer is stopped and the VLDRL is set in VID_STA.

A bit buffer area can be defined with VID_BBS (bit buffer start) and VID_BBE (bit buffer end).

- If these registers are left in their reset state (0x0000 0000) or if the same value is written in both of them, they have no effect.
- If different values are programmed in both registers when the internal variable length decoder read pointer reaches VID_BBE it jumps to VID_BBS. This allows a video ES to be wrapped in a predefined memory area.

*Note: 1 The variable length decoder read pointer (VID_VLDPTR) has to be programmed before each decoding task, even if the variable length decoder has stopped on the picture start code of the next picture to be decoded. This is because the picture start code could have been stored in the variable length decoder input FIFO at the end of the previous picture decode and this FIFO is flushed before starting the next decoding task.*

*2 The read limit to be programmed must be enlarged by the size of one burst, that is 128 bytes. This is due to internal pipeline processing in the variable length decoder. When the variable length decoder FIFO is empty, there is still about 64 bytes in the variable length decoder pipeline that are not processed.*

Confidential

### 42.5.4 Main and secondary reconstruction

The decoder has two ways of reconstructing the picture in external memory. The normal way is through the main reconstruction. The reference frames used for MPEG2 prediction are always stored through this path. When required to reduce picture size before memory storage, in preparation for a reduced display size, a secondary reconstruction path can be used.

The use of hardware secondary reconstruction depends on the application.

It is useful when large zoom out factors are needed; part of the zoom-out may be done using secondary reconstruction, the rest being done in the display.

In any case, reference pictures (I and P) must be stored in full size (main reconstruction). If secondary reconstruction is needed for I and P pictures then they have to be stored in both formats (full size and decimated).

Main reconstruction is enabled by register VID_RCM.ENM, secondary reconstruction by VID_RCM.ENS. B pictures may be only reconstructed with the secondary reconstruction. I and P frames, since they are used as a reference, must be reconstructed using both paths.

The secondary reconstruction is processed according to the value of VID_RCM, which gives the horizontal and vertical ratio (1,1/2 and 1/4 for both direction) and the progressive sequence flag. Horizontal decimation is performed as described in Figure 113 and vertical decimation is

performed as described in Figure 112. When decimation is needed in both direction, the vertical decimation is performed after the horizontal decimation.

If, due to decimation, the resulting picture is not a multiple of 16 pixels horizontally or vertically (size of luma stored macroblock), the actual stored picture is rounded up to the closest multiple of 16 pixels in the appropriate direction(s). Right side and/or bottom side of the stored picture is the garbage and has to be removed within the display processor prior to display.

When the decoded picture belongs to a progressive sequence, vertical decimation is done in the frame. Else, vertical decimation is done in the field.

Reconstructed frame pointers for luminance and chrominance for secondary reconstruction are VID_SRFP and VID_SRCHP.

At any time, the CPU may read the position of the next macroblock to be decoded in the reconstructed frame buffer in register VID_MBNM for the main reconstruction and VID_MBNS for the secondary reconstruction.

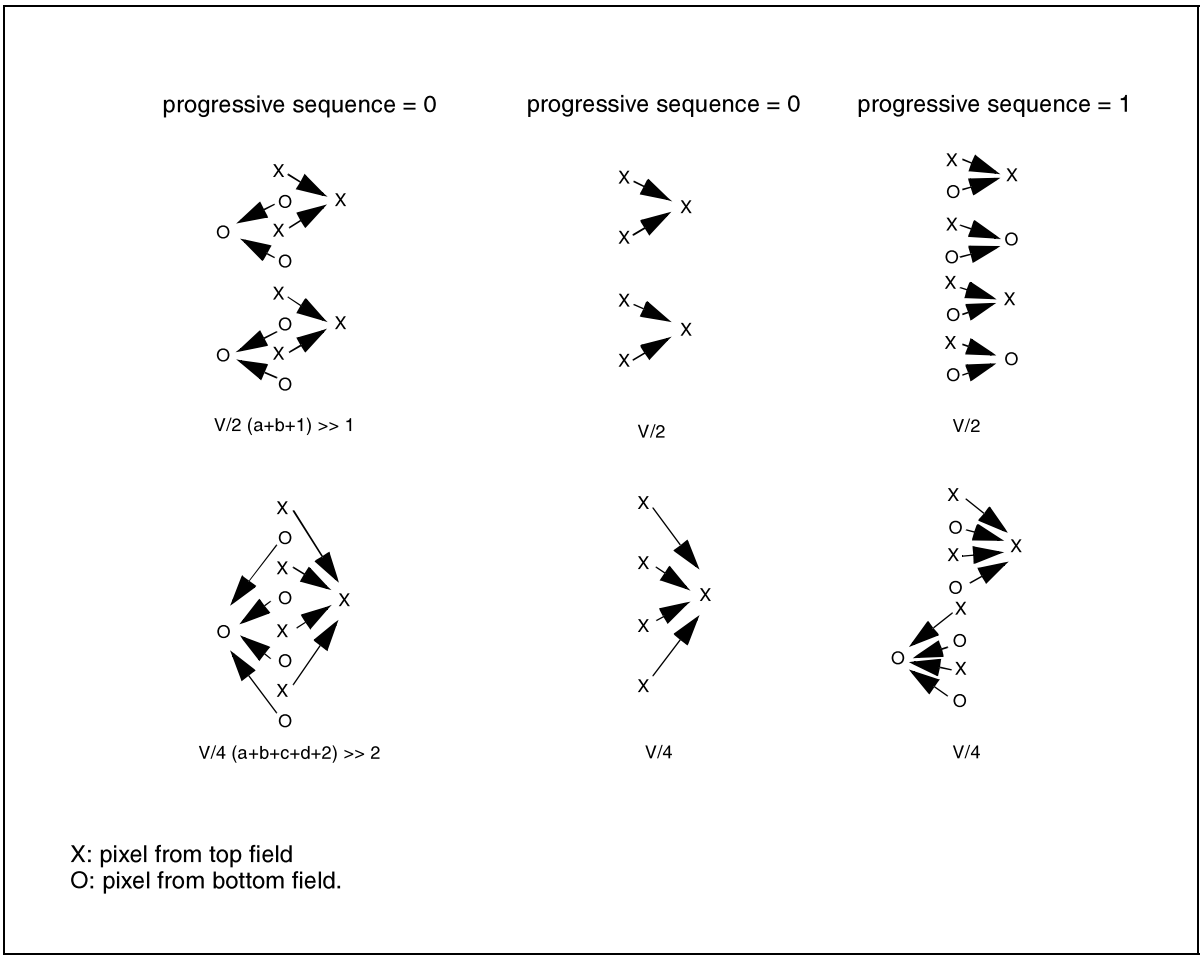



**Figure 112: Vertical decimation in field picture secondary reconstruction**

Confidential

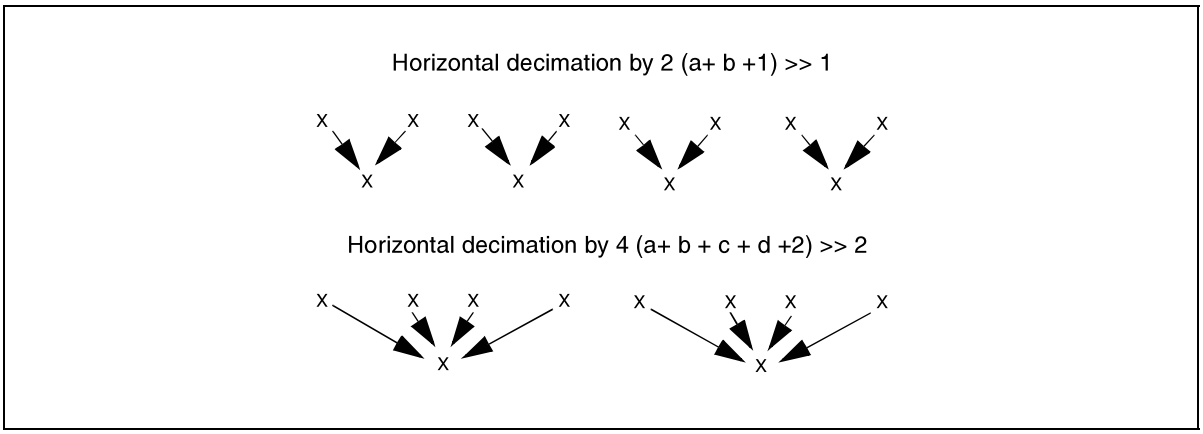



**Figure 113: Horizontal decimation in secondary reconstruction**

### 42.5.5 Error recovery

There are three levels of error detection in the video decoder:

- bitstream syntax/semantic error detection with automatic missing macroblocks concealment,
- pipeline underflow error detection,
- pipeline overflow error detection.

#### Syntax and semantic error detections

The variable length decoder detects the syntax and semantic errors listed below. The way to conceal errors is different depending on the detected errors. Table 123 describes errors that are detected by the MPEG2 video decoder. Error concealment is described below the table.

Confidential

**Table 123: Error detection**

| Layer | Description | Concealment | MBE[a] |
|---|---|---|---|
| **Syntax error:** A variable length code which does not exist in the tables or a fixed length code with a forbidden value | | | |
| Slice | **quantizer_scale_code** = 0 | Previous **quantizer_scale_code** is used (macroblock or slice). After a reset (HW, SRS), value is set to 1. | *N* |
| Macroblock | **macroblock_address_increment** | See *Procedure one* on page 405 | Yes |
| | **macroblock_type** | | |
| | **frame_motion_type** and **field_motion_type**: the value 00 is reserved | | |
| | **quantiser_scale_code** | Previous **quantizer_scale_code** (slice or macroblock) is used. After a reset (HW, SRS), value is set to 1. | No |
| | **marker_bit** | See *Procedure one* on page 405 | Yes |
| | **coded_block_pattern_420** | | |
| | **end_of_macroblock** for D pictures: it shall be equal to 1. | | |
| | **motion_code[r][s][t]** | | |
| Block | **first_dct_coef** | | |
| | **next_dct_coef** | | |
| **Semantic error:** The way of encoding slices and macroblocks does not follow MPEG2 rules. | | | |
| Slice | Slice start code combined with macroblock address increment gives a macroblock number already decoded. | See *Procedure one* on page 405 | Yes |
| | Slice start code combined with macroblock address increment gives a macroblock number higher than the macroblock number to be decoded. | | |
| Macroblock | mb_addr_increment may result in a macroblock address greater than the picture size. The macroblock is outside the picture. | See *Procedure two* on page 406 | Yes |
| | The processing of the different parameters in **motion_vectors**(s) may result in a motion vector outside the reference picture. | If bit MVC in VID_TIS is one, the vector is clipped to point in the picture area. Else, a prediction out of the reference picture is done. | No |
| | QFS[0] may not lie in the range 0 to ((2^(8+intra_dc_precision))-1). | If negative value, the coefficient is set to 0 but the predictor remains negative | No |
| Block | Too many coefficients may be found in a block. | See *Procedure one* on page 405 | Yes |

a. MBE: when yes, the error is taken into account in VID_MBE*n* registers. When no, the error is not taken into account.

Confidential

### Procedure one

If the variable length decoder detects a syntax or semantic error in the bitstream, the pipeline copies macroblocks from the previous picture. It uses the motion vectors reconstructed for the previous row of macroblocks in the current picture, and scans the bitstream until a slice start code is detected. At this point, normal decoding resumes.

If the error occurred in the last slice in the picture, concealment continues until the end of the picture. The pipeline then stops normally, assuming that the following picture start code is intact. Macroblocks are concealed using the vectors of the macroblock immediately above the lost macroblock.

Concealment macroblocks are accessed using the forward and backward reference frames. Lost macroblocks in the first row are copied directly from the previous pictures, that is as P-macroblocks with zero motion vectors. If an intra macroblock is coded with concealment motion vectors, the concealment motion vectors are used. If not, concealment is a simple copy from the previous picture using zero vectors.

Table 124 shows the rules used to fetch concealment macroblocks. Skipped macroblocks are not mentioned since they always refer to one of the prediction types listed below.

**Table 124: Macroblock type recovery in case of error concealment.**

| Picture structure/ picture type | Prediction type of the macroblock aligned vertically in the row above | Prediction type of the concealed macroblock 1 vector = (H,V) |
|---|---|---|
| I,P,B frame picture | Intra no concealment | Forward frame (vector = 0) |
| | Intra with concealment | Forward frame (with sent vector) |
| I, P, B field picture | Intra no concealment | Forward field (vector = 0) |
| | Intra with concealment | Forward field (with sent vectors) |
| P frame picture | Forward frame | Forward frame (with same vectors) |
| | Forward field | Forward field (with same vectors) |
| | Dual | Forward field (with vectors of same field parity, that is vectors sent) |
| P field picture | Forward field | Forward field (with same vectors) |
| | 16x8 | Forward field (with same top vectors) |
| | Dual | Bidirectional field (with same vectors) |
| B frame picture | Forward frame | Forward frame (with same vectors) |
| | Forward field | Forward field (with same vectors) |
| | Back frame | Back frame (with same vectors) |
| | Back field | Back field (with same vectors) |
| | Bidirectional frame | Bidirectional frame (with same vector) |
| | Bidirectional field | Forward field (with same forward vectors) |
| B field picture | Forward field | Forward field (with same vectors) |
| | Forward 16 x 8 | Forward field (with same top vectors) |
| | Back field | Back field (with same vectors) |
| | Back 16 x 8 | Back field (with same top vectors) |
| | Bidirectional field | Bidirectional field (with same vectors) |
| | Bidirectional 16 x 8 | Forward field (with same top vectors) |

Confidential

#### Procedure two

When the decoder detects that the input stream is not compliant with the restricted slice structure, the decoder reconstructs missing macroblocks by copying macroblocks from the forward reference picture with a null motion vector using frame prediction in a frame picture and field prediction in a field picture.

The decoder detects a nonrestricted slice structure when the slice start code combined with the first macroblock address increment gives an absolute address which is higher (+2 at least) than the previous decoded macroblock number. The decision (restricted or nonrestricted slice structure) is taken on the first macroblock following the slice start code.

An underflow occurs when the last slices are missing and a different start code to a slice start code or an error start code is detected. Missing macroblocks are reconstructed if RMM in VID_TIS = 1, and are not reconstructed if RMM = 0. For underflow errors, the underflow flag in the status register is always set.

#### Overflow and underflow errors

An overflow error occurs when the variable length decoder detects more macroblocks in the bitstream than the number corresponding to the picture size, VID_DFS. The variable length decoder stops processing macroblocks and resumes a start code search on the next picture start code, then, it returns in idle state. The overflow condition is flagged by the bit VID_STA.DOE.

An underflow error occurs when the variable length decoder has found in the bitstream fewer macroblocks than the number defined by the decoded picture size, VID_DFS, and when it stops on a picture start code. Then, it returns in idle state. The underflow condition is flagged by the bit DUE in VID_STA.

The number of macroblocks reconstructed in memory is equal to VID_DFS if RMM in VID_TIS = 1 (missing macroblocks are concealed).

### 42.5.6 Error statistics

The number and the location of the errors in the pictures determine whether the picture is to be displayed. The decoder gives statistics in the VID_MBE*n* (macroblock errors) registers. The VID_MBE*n* registers give the locations of both semantic and syntax errors in the picture. These registers are stable at the end of the decoding task until the start of the next picture decoding task whatever the video channel. At the end of picture decoding, if at least one error occurred during picture reconstruction, bit VID_STA.MSE is set. If no error occurred, this bit is 0.

The decoded picture is divided into 16 areas, defined by four horizontal and four vertical slides. The areas are numbered from 0 (top left) to 15 (bottom right). This number is incremented from left to right, top to bottom (see Figure 114). For each area, VID_MBE*n* gives the number of macroblocks reconstructed with error concealment.

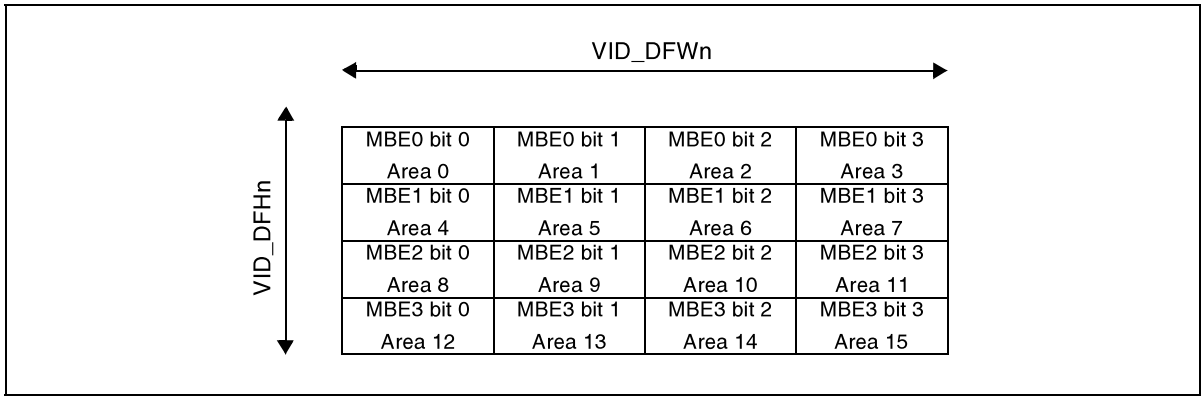
If the picture size in macroblock units is a multiple of four in both directions, each of the 16 areas contains the same number of macroblocks DFS >> 4. If it is not a multiple of four macroblocks, the number of macroblocks in each area is:

```
For i and j in [0..2]:
  area(i*4+j)= (DFH >> 2) * (DFW >> 2) MBs
For i=3 and j in [0..2]:
  area(12+j) = (DFH >> 2 + DFH mod 4) * (DFW >> 2) MBs
For i in [0..2] and j=3:
  area(i*4+3) = (DFH >> 2) * (DFW >> 2 + DFW mod 4) MBs
For i = 3 and j=3:
  area(15) = (DFH >> 2 + DFH mod 4) * (DFW >> 2 + DFW mod 4) MBs
```

Confidential

*Note: In the field picture structure, DFH must be divided by two.*

**Figure 114: Macroblock error statistics**




### 42.5.7 Simplified B decoding

Simplified B decoding allows bandwidth consumption to be reduced up to 40% while decoding a B-picture. This is done by converting all bidirectional predictors of B-pictures in forward predictors.

The maximum bandwidth required to decode a B-picture is usually 4216 Mbit/s, but for simplified B, this is equivalent to a P-picture at 2712 Mbits/s.

This feature has an impact on picture quality, but has no effect on the P dual prime picture.

Secondary decoding requires an additional 50% of main picture bandwidth - 2108 Mbit/s (B) or 1356 Mbit/s (SB/P).

Confidential

## 42.6 Resets

### 42.6.1 Hardware reset

After a hardware reset, the MPEG2 video decoder is in an idle state. No processing is done and no request is sent until video decoding is launched.

### 42.6.2 Software reset

Two software resets are implemented.

- VID_SRS: the reset is active when the CPU writes the least significant byte of the VID_SRS register. Software reset is a synchronous active high reset. It resets:
  - variable length decoder flags: DID, MSE, DOE and DUE in VID_STA,
  - variable length decoder input FIFO controller,
  - error statistic registers: VID_MBE0, VID_MBE1, VID_MBE2 and VID_MBE3.

  After the reset no processing is performed until video decoding is started. The MPEG2 video decoder is in an idle state.

  Software reset has no impact on register contents.
- VID_EXE: has the same effect as software reset (VID_SRS), but it also starts the variable length decoder.

# 43 MPEG2 video decoder registers

Register addresses are provided as *MPEGBaseAddress* + offset.

The *MPEGBaseAddress* is:

0x1924 0000.

All registers are reset by hardware. Read only registers can also be affected by SRS and EXE; see Section 42.6.2: *Software reset on page 407*. When registers are read, the value of the reserved bits are 0 unless specified. No registers are buffered: write action effect is immediate.

**Table 125: MPEG2 video decoder register summary**

| Register | Description | Offset | Type |
|---|---|---|---|
| VID_EXE | Execute decoding task | 0x0008 | WO |
| Reserved | - | 0x0010 | - |
| VID_MBEn | Macroblock error statistic (n = 0 to 3) | 0x0070 - 0x007C | RO |
| VID_QMWIp | Quantization matrix data, intra-table | 0x0100 - 0x013F | R/W |
| VID_QMWNIp | Quantization matrix data, non–intra-table | 0x0180 - 0x01BF | |
| VID_TIS | Task instruction | 0x0300 | |
| VID_PPR | Picture parameters | 0x0304 | |
| VID_SRS | Decoding soft reset | 0x030C | WO |
| VID_ITM | Interrupt mask | 0x03F0 | R/W |
| VID_ITS | Interrupt status | 0x03F4 | RO |
| VID_STA | Status | 0x03F8 | |
| VID_DFH | Decoded frame height | 0x0400 | R/W |
| VID_DFS | Decoded frame size | 0x0404 | |
| VID_DFW | Decode frame width | 0x0408 | |
| VID_BBS | Video elementary stream bit buffer start | 0x040C | |
| VID_BBE | Video elementary stream bit buffer end | 0x0414 | |
| VID_VLDRL | Variable length decoder read limit | 0x0448 | |
| VID_VLD_PTR | Variable length decoder read pointer | 0x045C | |
| Reserved | - | 0x0480 - 0x0484 | - |

Confidential

**Table 125: MPEG2 video decoder register summary**

| Register | Description | Offset | Type |
|---|---|---|---|
| VID_BCHP | Backward chroma frame buffer | 0x0488 | R/W |
| VID_BFP | Backward luma frame pointer | 0x048C | |
| VID_FCHP | Forward chroma frame buffer | 0x0490 | |
| VID_FFP | Forward luma frame pointer | 0x0494 | |
| VID_RCHP | Main reconstructed chroma frame pointer | 0x0498 | |
| VID_RFP | Main reconstructed luma frame pointer | 0x049C | |
| VID_SRCHP | Secondary reconstructed chroma frame buffer | 0x04A0 | |
| VID_SRFP | Secondary reconstructed luma frame buffer | 0x04A4 | |
| VID_RCM | Reconstruction mode | 0x04AC | |

## VID_EXE Execute decoding task

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VID_EXE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *MPEGBaseAddress* + 0x0008

Type: WO

Reset: Undefined

Description: Writing to the least significant byte of this register starts a decoding task.

## VID_MBEn Macroblock error statistic (n = 0 to 3)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MBEx3[7:0] | | | | | | | | MBEx2[7:0] | | | | | | | | MBEx1[7:0] | | | | | | | | MBEx0[7:0] | | | | | | | |

Address: *MPEGBaseAddress* + 0x0070 (VID_MBE0), 0x0074 (VID_MBE1), 0x0078 (VID_MBE2), 0x007C (VID_MBE3)

Type: RO

Reset: 0

Description: Decoded pictures are divided into 16 areas. For each area, MBE gives the number of macroblock errors (semantic and syntax) detected by the variable length decoder. These data are stable at the end of the picture decoding task until the start of the next picture decoding task (whether the video channel is 1 or 2).

Confidential

### VID_QMWIp Quantization matrix data, intra-table

| 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| IQCOEFF(4p+3)[7:0] | IQCOEFF(4p+2)[7:0] | IQCOEFF(4p+1)[7:0] | IQCOEFF(4p)[7:0] |

Address: *MPEGBaseAddress* + 0x0100 to 0x013F

Type: R/W

Reset: 0

Description: The quantization coefficients for an intra table must be written to these addresses in increasing order in the order in which they appear in the bitstream (in zig-zag order). Thus, the first coefficient which appears in the bitstream is located at the lower register address and the last coefficient in the bitstream is located at the higher register address. The order must be strictly respected. Each 32-bit word contains four 8-bit coefficients.

### VID_QMWNIp Quantization matrix data, non–intra-table

| 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| NIQCOEFF(4p+3)[7:0] | NIQCOEFF(4p+2)[7:0] | NIQCOEFF(4p+1)[7:0] | NIQCOEFF(4p)[7:0] |

Address: *MPEGBaseAddress* +0x0180 to 0x01BF

Type: R/W

Reset: 0

Description: The quantization coefficients for a non-intra table must be written to these addresses in increasing order in the order in which they appear in the bitstream (in zig-zag order). Thus, the first coefficient which appears in the bitstream is located at the lower register address and the last coefficient in the bitstream is located at the higher register address. The order must be strictly respected. Each 32-bit word contains four 8-bit coefficients.

### VID_TIS Task instruction

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| Reserved | RMM | MVC | SBD | OVW |

Address: *MPEGBaseAddress* +0x0300

Type: R/W

Reset: 0

This register contains the decoding task instruction.

[31:4] **Reserved**

[3] **RMM**: reconstruct missing macroblock.
1: In case of underflow, the missing macroblocks are reconstructed.

[2] **MVC:** motion vector check
1: Ensures that motion vectors used for prediction remain inside the picture (see Table 123: *Error detection* on page 404).

[1] **SBD:** simplified B picture decoding
1: B picture decoding is simplified to save bandwidth: all bidirectional macro blocks are processed as forward.

[0] **OVW**
1: Enable overwrite mechanism during decoding

Confidential

## VID_PPR Picture parameters

| 31 | 30 | 29 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 20 | 19 18 | 17 16 | 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | MP2 | FFN | TFF | FRM | CMV | QST | IVF | AZZ | PCT | DCP | PST | BFV | FFV | BFH | FFH |

Address: *MPEGBaseAddress* + 0x0304

Type: R/W

Reset: 0

Description: This register contains parameters of the picture to be decoded. These parameters are extracted from the bitstream. In MPEG-1 mode (when MP2 in VID_PPR is reset), only PCT, BFH and FFH have to be set. Other bits must be reset.

[31] **Reserved**

[30] **MP2**: MPEG-2 mode
0: An MPEG-1 bitstream is expected
1: The decoder expects an MPEG-2 video bitstream

[29:28] **FFN**: Force field number
This is used for prediction to select the reference pictures.
01 or 10: The field is forced to first field picture 11: The field is forced to second field picture
00: Internal field number is used
The field number use for prediction is computed internally by the MPEG2 video decoder. Meanwhile, if an error occurs in the stream and a field is lost, this bit must be set at the proper value by the application. It may be used also in still picture decoding.

[27] **TFF**: Top field first
Set equal to the TOP_FIELD_FIRST bit of the MPEG-2 picture coding extension.

[26] **FRM**
Set equal to the FRAME_PRED_FRAME_DCT bit of the picture coding extension.

[25] **CMV**: Set equal to the CONCEALMENT_MOTION_VECTORS bit of the MPEG-2 picture coding extension. Indicates that motion vectors are coded for intra macroblocks.

[24] **QST**: Set equal to the Q_SCALE_TYPE bit of the picture coding extension.

[23] **IVF**: Set equal to the INTRA_VLC_FORMAT bit of the picture coding extension.

[22] **AZZ**
Set equal to the ALTERNATE_SCAN bit of the picture coding extension.

[21:20] **PCT[1:0]**: Set to equal to the two least significant bits of PICTURE_CODING_TYPE in the picture header.

[19:18] **DCP[1:0]**: Set equal to INTRA_DC_PRECISION of the picture coding extension. The value 11, defining a precision of 11 bits, is not allowed.

[17:16] **PST[1:0]**: Set equal to the PICTURE_STRUCTURE bits of the MPEG-2 picture coding extension.
*Note: code 00 also indicates frame structure, even though this value is illegal in the MPEG-2 variable.*
00: Frame picture 01: Top field
10: Bottom field 11: Frame picture

[15:12] **BFV[3:0]**: Set equal to BACKWARD_VERTICAL_F_CODE of the picture coding extension.

[11:8] **FFV[3:0]**: Set equal to FOWARD_VERTICAL_F_CODE of the picture coding extension.

[7:4] **BFH[3:0]**
In MPEG-1 mode BFH[3] is set equal to FULL_PEL_BACKWARD_VECTOR of the picture header, and BFH[2:0] is set equal to BACKWARD_F_CODE of the picture header. In MPEG-2 mode BFH[3:0] is set equal to BACKWARD_HORIZONTAL_F_CODE of the picture coding extension.

[3:0] **FFH[3:0]**
In MPEG-1 mode FFH[3] is set equal to FULL_PEL_FORWARD_VECTOR of the picture header, and FFH[2:0] is set equal to FORWARD_F_CODE of the picture header. In MPEG-2 mode FFH[3:0] is set equal to FORWARD_HORIZONTAL_F_CODE of the picture coding extension.

Confidential

### VID_SRS Decoding soft reset

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VID_SRS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *MPEGBaseAddress* + 0x030C

Type: WO

Reset: Undefined

Description: Writing to the least significant byte of this register starts the software reset sequence. The reset is just activated once on writing.

### VID_ITM Interrupt mask

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | | R_OPC | VLDRL | DSE | MSE | DUE | DOE | DID |

Address: *MPEGBaseAddress* + 0x03F0

Type: R/W

Reset: 0

Description: Any bit set in this register enables the corresponding interrupt on the VID_IRQ line. An interrupt is generated when a bit in the VID_STA register changes from 0 to 1 and the corresponding mask bit is set.

### VID_ITS Interrupt status

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | | R_OPC | VLDRL | DSE | MSE | DUE | DOE | DID |

Address: *MPEGBaseAddress* + 0x03F4

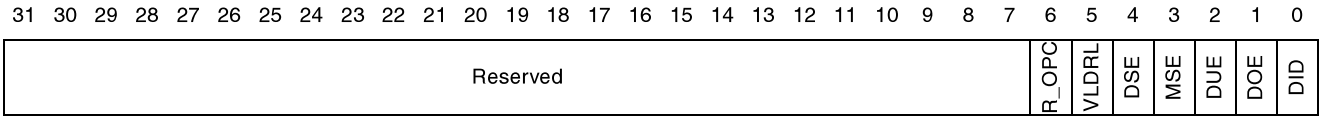
Type: RO

Reset: 0

Description: When a bit in the VID_STA register changes from 0 to 1, the corresponding bit in VID_ITS is set (independently of ITM). When there is at least one bit set to one in VID_ITS which is not masked, the interrupt line is set to 1. The reading of VID_ITS clears all bits in that register and de-asserts the interrupt line.

Confidential

## VID_STA Status




Address: *MPEGBaseAddress* +0x03F8

Type: RO

Reset: 0x0000 0021

Description: This register contains a set of bits which represent the status of the decoder at any instant. Any change from 0 to 1 of any of these bits sets the corresponding bit of the VID_ITS register, and can thus potentially cause an interrupt on the GL1_IRQ line. Some bits are pulses and are unlikely ever to be read as 0.

[31:7] **Reserved**

[6] **R_OPC:** set when an R_OPC is 1 for one of the two interconnect plugs. This signal is a one cycle pulse.

[5] **VLDRL:** variable length decoder read limit. This flag is set to 1 when the variable length decoder read pointer reaches VID_VLDRL pointer. The bit is reset when the condition is not true.

[4] **DSE:** a one pulse bit, set to 1 when decoding semantic or syntax error is detected by the variable length decoder.

[3] **MSE:** set to 1 at the end of a decoding task if a semantic or a syntax error has been detected by the variable length decoder during the decoding process of the picture. It is reset on a write action on VID_EXE or VID_SRS LS byte.

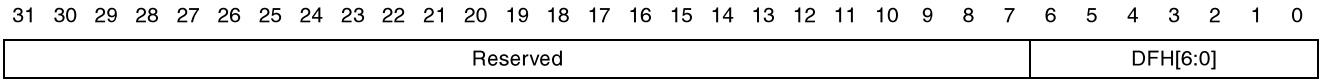
[2] **DUE:** set to 1 at the end of a decoding task if a underflow error has been detected by the variable length decoder during the decoding process of the picture. It is reset on a write action on VID_EXE or VID_SRS LS byte.

[1] **DOE:** set to 1 at the end of a decoding task if an overflow error has been detected by the variable length decoder during the decoding process of the picture. It is reset on a write action on VID_EXE or VID_SRS LS byte.

[0] **DID:** decoder idle. Set to 1 when pipeline is idle and the variable length decoder is in idle state or on a write action on VID_SRS LS byte. The condition is reset on a write action on VID_EXE.

Confidential

## VID_DFH Decoded frame height



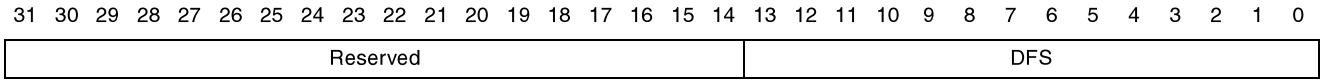

Address: *MPEGBaseAddress* + 0x0400

Type: R/W

Reset: 0

Description: Decoded frame height (in rows of macroblocks). This register is used for error concealment and macroblocks error statistics. This is derived from the vertical size value transmitted in the sequence header. It is divided internally by two for field picture decoding.

### VID_DFS Decoded frame size



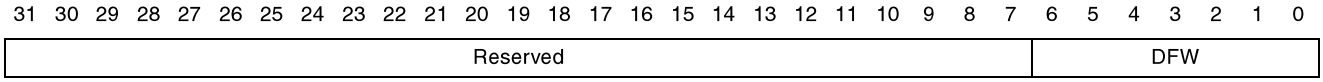

Address: *MPEGBaseAddress* + 0x0404

Type: R/W

Reset: 0

Description: This register is set up with a value equal to the number of macroblocks in the decoded picture. This is derived from the horizontal and vertical size values transmitted in the sequence header. It is divided internally by two for field picture decoding.

### VID_DFW Decode frame width



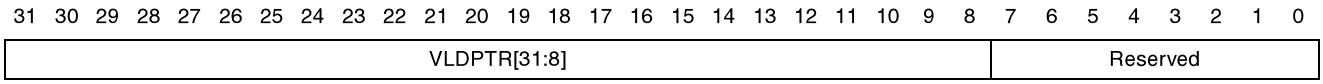

Address: *MPEGBaseAddress* + 0x0408

Type: R/W

Reset: 0

Description: This register is set up with a value equal to the width in macroblocks of the decoded picture. This is derived from the horizontal size value transmitted in the sequence header.

### VID_BBS Video elementary stream bit buffer start



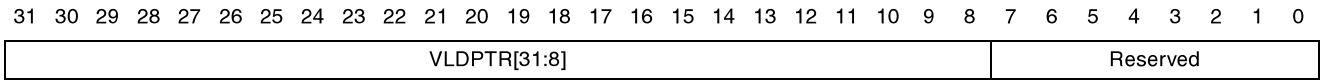

Address: *MPEGBaseAddress* + 0x040C

Type: R/W

Reset: 0

Description: Memory address of the video elementary stream bit buffer start, defined in 256-byte units.

### VID_BBE Video elementary stream bit buffer end




Address: *MPEGBaseAddress* + 0x0414

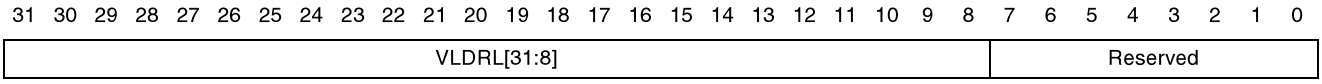
Type: R/W

Reset: 0

Memory address of the video elementary stream bit buffer end, defined in 256-byte units.

Confidential

### VID_VLDRL Variable length decoder read limit



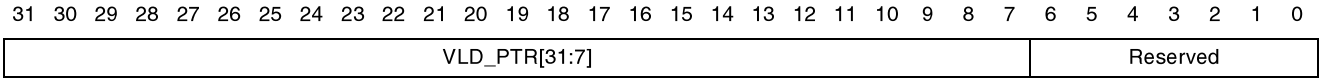

Address: *MPEGBaseAddress* + 0x0448

Type: R/W

Reset: 0

Description: This register stores the variable length decoder read limit. When the variable length decoder read pointer reaches VID_VLDRL, reading in the buffer is stopped and the bit VLDRL is set to 1 in the status register. This pointer is defined in 256-byte units.

### VID_VLD_PTR Variable length decoder read pointer




Address: *MPEGBaseAddress* + 0x045C

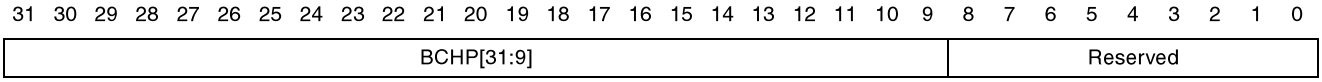
Type: R/W

Reset: 0

Description: Memory address of the variable length decoder read pointer, in bytes. The address where the variable length decoder should start decoding the task must be written here before each decoding task.

This value is taken into account as soon as VID_EXE is accessed. It then holds the current variable length decoder read pointer address given in 128-byte units.

### VID_BCHP Backward chroma frame buffer



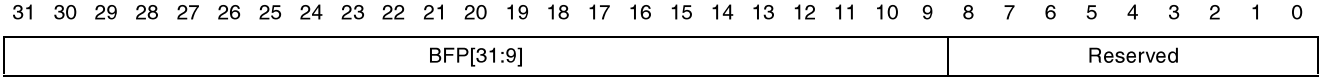

Address: *MPEGBaseAddress* + 0x0488

Type: R/W

Reset: 0

Description: Start address of the backward frame chroma prediction buffer, defined in 512-byte units.

### VID_BFP Backward luma frame pointer




Address: *MPEGBaseAddress* + 0x048C

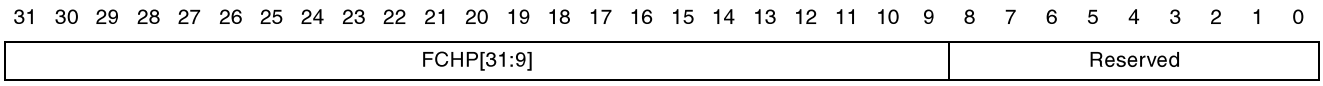
Type: R/W

Reset: 0

Description: Start address of the luma backward prediction frame picture buffer, defined in 512-byte units.

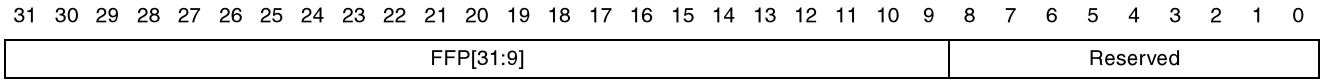
Confidential

### VID_FCHP Forward chroma frame buffer



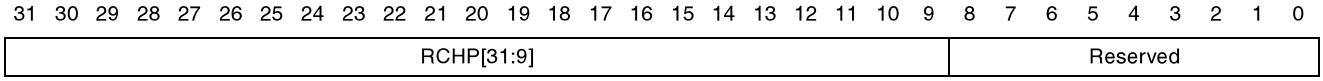

Address: *MPEGBaseAddress* + 0x0490
Type: R/W
Reset: 0
Description: Start address of the chroma forward prediction frame picture buffer, defined in 512-byte units.

### VID_FFP Forward luma frame pointer



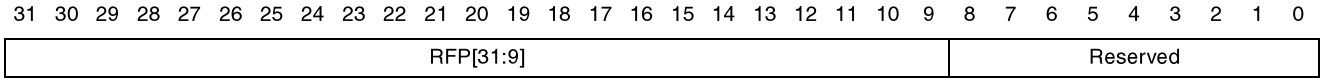

Address: *MPEGBaseAddress* + 0x0494
Type: R/W
Reset: 0
Description: Start address of the luma forward prediction frame picture buffer, defined in 512-byte units.

### VID_RCHP Main reconstructed chroma frame pointer




Address: *MPEGBaseAddress* + 0x0498
Type: R/W
Reset: 0
Description: Start address of the reconstructed chroma frame picture buffer, defined in 512-byte units.

### VID_RFP Main reconstructed luma frame pointer




Address: *MPEGBaseAddress* + 0x049C
Type: R/W
Reset: 0
Description: Start address of the reconstructed (decoded) luma picture buffer, defined in 512-byte units.

Confidential

### VID_SRCHP Secondary reconstructed chroma frame buffer

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRCHP[31:9] | | | | | | | | | | | | | | | | | | | | | | | Reserved | | | | | | | | |

Address: *MPEGBaseAddress* + 0x04AC
Type: R/W
Reset: 4
Description: Start address of the secondary reconstructed chroma frame buffer, defined in 512-byte units.

### VID_SRFP Secondary reconstructed luma frame buffer

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRFP[31:9] | | | | | | | | | | | | | | | | | | | | | | | Reserved | | | | | | | | |

Address: *MPEGBaseAddress* + 0x04A4
Type: R/W
Reset: 0
Description: Start address of the secondary reconstructed luma frame buffer, defined in 512-byte units.

### VID_RCM Reconstruction mode

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | | VDEC | | HDEC | | ENM | ENS | PS |

Address: *MPEGBaseAddress* + 0x04AC
Type: R/W
Reset: 0x04
Description: Parameters of reconstruction (main and secondary). After hardware reset, main reconstruction is enabled and secondary is disabled.

[31:7] **Reserved**

[6:5] **VDEC**: Vertical decimation mode
00: no decimation
01: V/2 decimation
10: V/4 decimation
Others: Illegal.

[4:3] **HDEC**: Horizontal decimation mode
00: no decimation
01: H/2 decimation
10: H/4 decimation
Others: Illegal.

[2] **ENM**: Enable reconstruction in main buffer

[1] **ENS**: Enable reconstruction in secondary buffer

[0] **PS**: Progressive sequence.
This bit should be set when a progressive sequence is being decoded.
1: Decimation is performed by averaging pixel lines in the frame.

Confidential

# 44 H.264 video decoder

## 44.1 Introduction

This chapter describes the Delta decoder (decode engine for low-bit-rate TV applications) which is the STx7100 H.264 high profile video decoder (also known as MPEG-4 part 10, MPEG-4/AVC and H26L).

## 44.2 Decoder main features

### 44.2.1 Supported formats

The Delta decoder is able to decode in real time:

- one H.264 high profile - level 4.1 video stream, or
- up to three H.264 high profile - level 3 video streams.

### 44.2.2 Input streams

The Delta decoder is a picture level decoder. Additional syntax elements like H.264 sequence parameter set and picture parameter set are handled by the ST40 host processor.

The Delta decoder accepts only **H.264 byte stream NALs**. It only processes slices; all other information (like PTS-DTS, sequence, picture, SEI) are managed externally. The host processor launches the picture decoding process interrupting the embedded processor (ST231).

### 44.2.3 Miscellaneous features

- Decimated second output (by two or four horizontally and/or by two vertically) in order to reduce memory bandwidth requirements for zoom out. If the picture horizontal size is not a multiple of the decimation factor, the Delta can insert dummy macroblocks. Vertically, the Delta can handle half-blocks.
- Error concealment

*Note: The Delta decoder does not support frame width larger than 1920 pels and frame height bigger than 2032 lines.*

*The minimum picture width is 2 MBs.*

*The minimum picture height is 2 lines of MBs.*

Confidential

### 44.2.4 Decoder overview

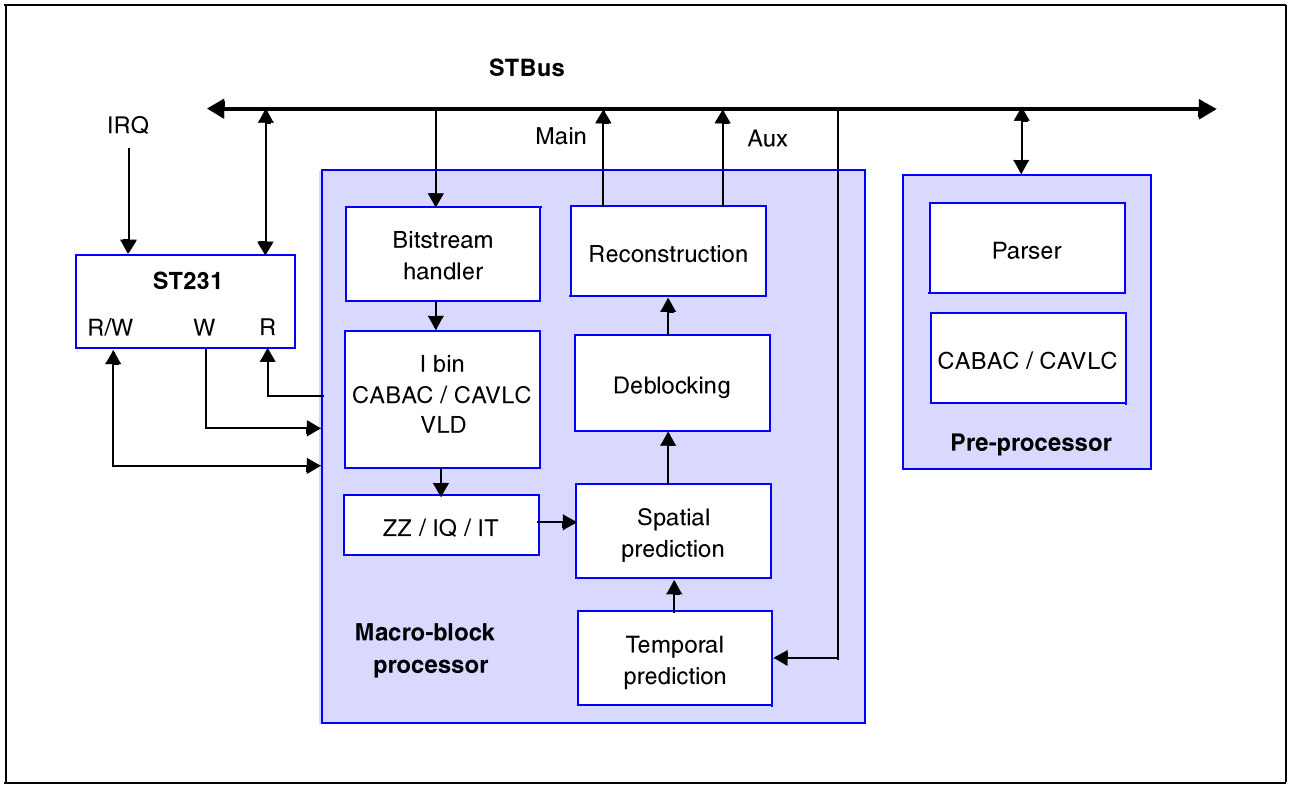
The Delta decoder is made of three units whose main functions are described below:

- Pre-processor unit (PP),
  - Slice header/data syntax analysis,
  - Macroblock header preprocessing,
  - CABAC and CAVLC entropy decoder (partial decoding),
- Macro-block engine (MBE),
  - DMA management,
  - UE/SE/iBin decoding for H.264 syntax elements (MB headers decoding),
  - CABAC and CAVLC Residual entropy decoding,
  - Inverse zig-zag, IQ, IT,
  - Inter MB motion vectors computation,
  - Temporal and spatial prediction,
  - Deblocking filtering,
  - Picture reconstruction in external memory,
- Embedded processor,
  - ST231 with firmware, used for syntax parsing, error concealment, intraprediction, motion vector processing,
  - Hardwired functions control.

Confidential

### 44.2.5 Decoder block diagram

**Figure 115: Delta decoder block diagram**



### 44.2.6 Decoding data flow

The Delta decoder is a frame decoder which decodes ES data under the control of the ST40. It is the FDMA which reads the PES previously stored in memory by the PTI and does the PES-to-ES parsing.

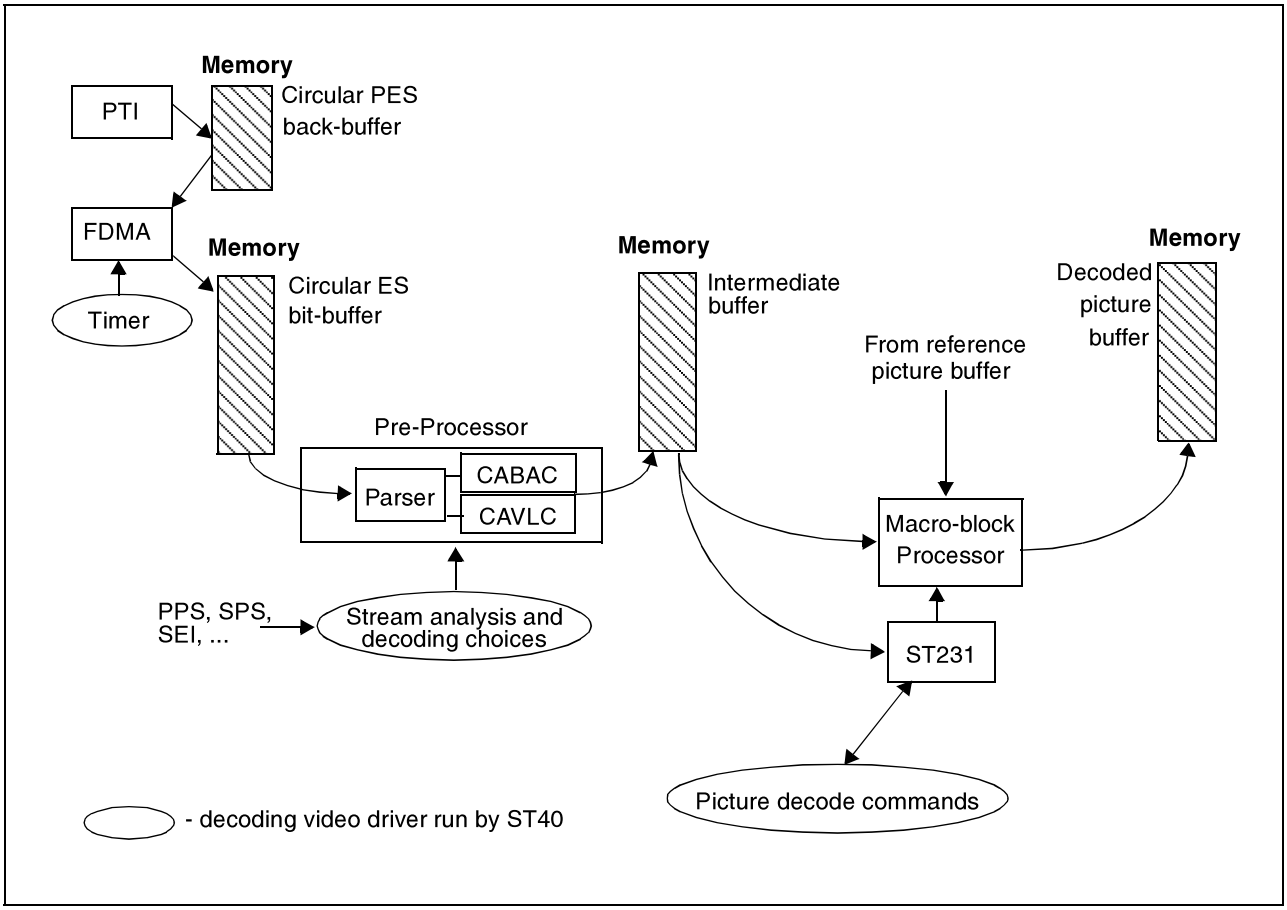
The Delta pre-processor reads the ES data from a circular bit-buffer in memory and stores the resulting data (partially decoded data) into an “intermediate buffer” in memory.

The intermediate buffer is then read by the Delta macro-block processor to complete the decoding. The final decoded frames are stored in the “decoded picture buffer” in memory.

A timer is used to interrupt regularly the ST40 which reads the PTI write-pointer and launches the FDMA to do the PES parsing.

The data flow is shown in Figure 116.

**Figure 116: H264 decode dataflow**




Confidential

### 44.2.7 Buffering requirements

The Delta decoder has the following buffering requirements:

- **Bit buffer (BB)**: circular buffer that contains an elementary stream - ES (NAL byte stream in H.264) - compressed data coming from the MPEG-TS demultiplexer or from a mass storage device (this buffer is compliant with the HRD (hypothetical reference decoder) CPB buffer).
- **Intermediate picture buffer (IB)**: this buffer contains pre-processed picture data (MB header and CABAC/CAVLC already partially decoded).
- **Decoded picture buffer (DPB)**: All decoded pictures (including decimated ones) are stored in memory using the same macroblock arrangement as the MPEG2 decoder, for display and reference purpose.
- **Decimated decoded picture buffer**: to reduce the display bandwidth consumption, it can be useful to decimate decoded picture before storing them in memory. The decoder can also write non-decimated picture in this buffer.
- **Picture parameter buffer (PPB)**: If a picture is marked as used for reference, additional data must be stored in a separate picture parameter buffer. This additional buffer is used for motion vector and reference picture prediction in H.264 temporal direct mode.

Each MB structure is stored in memory with the size defined by MBDescrSize (firmware) - 40 bytes are stored for level 4 and above, 160 for levels below level 4.

## 44.3 H.264 decoder interface

### 44.3.1 ST40 host CPU tasks overview

The ST40 host CPU runs the video driver and controls the Delta decoder.

The host is responsible for:

- FDMA control for PES processing / SCD (start code) extraction,
- SPS / PPS / SEI / VUI (video usability information) analysis:
  - sequence parameter set (SPS): the host CPU has to manage up to 32 sets,
  - picture parameter set (PPS): the host CPU has to manage up to 256 sets,
  - supplemental enhancement information messages (SEI): these messages assist in the processes related to decoding or display of video but are not required for reconstructing the luma or chroma samples,
- control of the decoder (pre-processor and core) on a picture basis,
- decoded picture buffer (DPB) management (reconstruction, prediction, display),
- intermediate buffer (IB) management,
- picture boundaries detection in the bit buffer using the slice start-code and the first field of the slice header. The read stop-address passed to pre-processor is the address of the first byte of next picture start code minus one.

### 44.3.2 Host ST40 - decoder ST231 communication

The Delta decoder is controlled by the ST40 on a picture-by-picture basis using the following methods:

- The ST40 writes picture level decoding commands in external memory.
- The ST40 and the Delta ST231 can each receive and generate interrupts with the dedicated mail box (see Chapter 27: *Mailboxes on page 223*).

The commands or the status placed in memory by the ST40 or by the embedded processor form the multimedia engine (MME) API.

Confidential

## 44.4 Firmware

The H.264 firmware is the code executed by the Delta ST231 and is delivered with the software driver.

### 44.4.1 Picture processing

The decoder firmware gets a picture decode request from the ST40 through a mechanism based on interruption and parameters placed in external memory. The firmware analyzes this request, sends a corresponding picture command to configure the hardware, and can start the processing of the first slice.

Before starting a new picture decode, the firmware checks that all the data of the previous picture has been written to memory.

### 44.4.2 Slice processing

The firmware gets the slice syntax from the intermediate buffer. It analyzes these syntax elements and sends the corresponding slice command to configure the hardware.

### 44.4.3 Macroblock processing

The firmware gets the macroblock syntax elements up to residuals, then, the firmware sends a pipeline command that requests the residual entropy decoder (hardware block) to read all the residuals of the current macroblock from the bitstream.

While the pipeline is emptying the input FIFO, the firmware computes all the parameters needed to control the prediction and the deblocking. If direct mode is used for this slice, the firmware read the collocated MB context from PPB buffer (picture parameter buffer). It then sends another MB command to configure the prediction and deblocking engines.

After this processing, the firmware reads the hardware status in order to determine if:

- all the residual have already been read,
- an error occurred on the residual entropy decoding.

Confidential

## 44.5 H.264 decoder reset

The preprocessor can be reset by software independently from the core decoder.

#### Pre-processor reset

The preprocessor is reset by writing a value in the PRE_SW_RST register. An interrupt is generated when the soft reset is completed.

#### Delta ST231 and macroblock processor reset

The core decoder (ST231 and MB processor) is reset by the global STx7100 reset signal. It is maintained in the reset state until the ST40 releases the reset signal using the configuration register SYS_CFG29. For more details, see Chapter 17: *Reset* *on page 161* and Chapter 20: *Boot modes* *on page 169*.

# 45 Teletext DMA

## 45.1 Teletext DMA overview

The STx7100 features a teletext block integrated with the DENC to support one video stream. Teletext data is retrieved from memory, serialized and transferred to the DENC by a dedicated teletext DMA. The digital encoder encodes teletext data according to the *CCIR/ITU-R Broadcast Teletext System B* specification (also known as world system teletext).

## 45.2 Teletext packet format

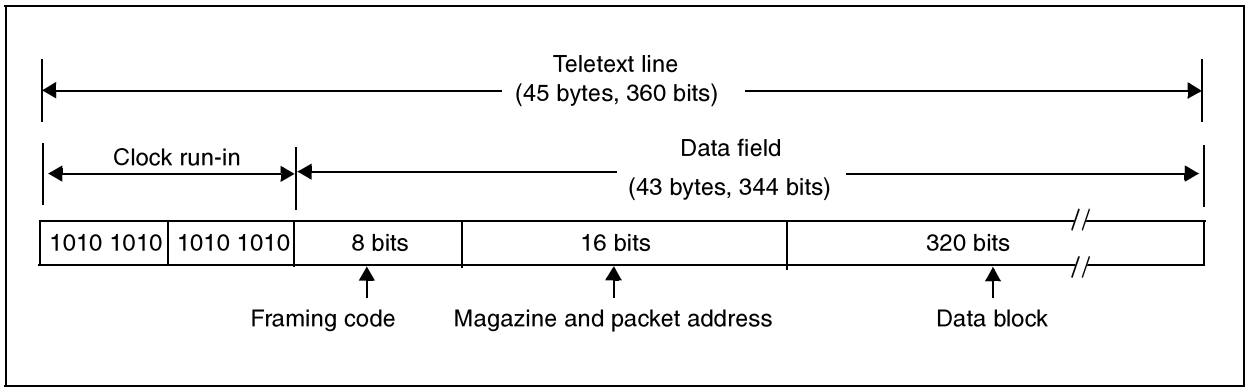
One teletext packet (also called a teletext line) is a stream of 360 bits, transferred at an average frequency of 6.9375 MHz. The data format is the same as the contents of the PES data packet as defined in the ETSI specification. The DMA reads in multiples of 46 bytes and transfers lines of 45 bytes to the digital encoder.

Each teletext packet is composed of the clock run-in and the data-field, as shown in Figure 117.

- The clock run-in is composed of two bytes, each with the value 0xAA (binary 1010 1010).
- The data-field consists of three fields: framing code, magazine and packet address, and data block fields. These three fields provide the block of teletext data.

The framing code is a single byte of hexadecimal value 0xE4[1]. The data is transmitted in order, from the LSB to the MSB of each byte in memory.

**Figure 117: Teletext packet format**




## 45.3 Data transfer sequence

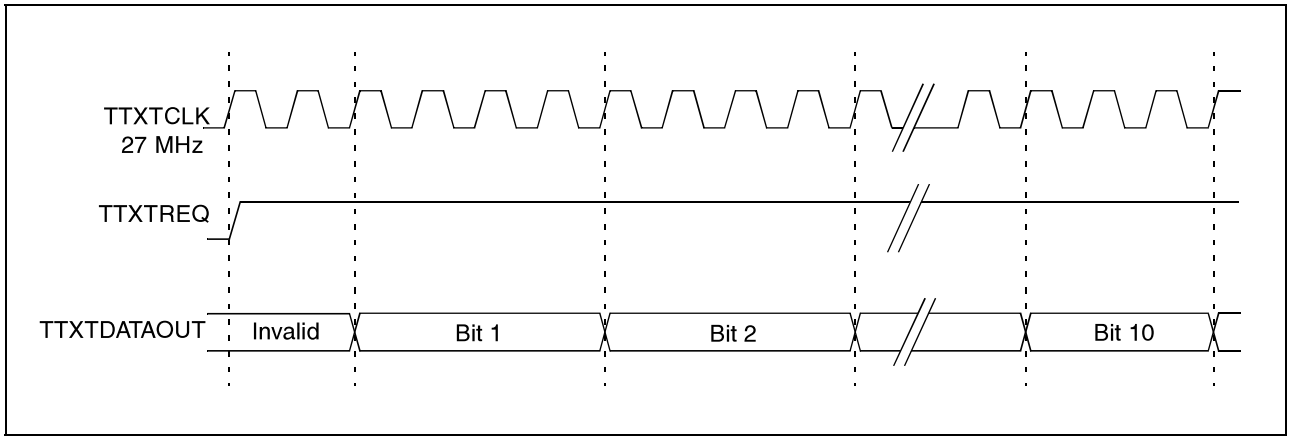
The digital encoder issues a teletext request signal to the teletext DMA, this is shown by the rising edge of signal TTXTREQ (PIO6[6]) in Figure 118. After a delay, programmable from 2 to 9 master-clock periods, the teletext DMA transmits the first valid teletext data bit of the teletext packet.

The 360 bits of output data are defined as nine 37-bit sequences, ending with one 27-bit sequence. Within each sequence, each bit is transmitted in four 27 MHz cycles, except bits 10, 19, 28 and 37, which are transmitted in three 27 MHz cycles. This is illustrated in Figure 118 for bits 0 to 10.

1. Specification for conveying ITU-R Systems B teletext in digital video broadcasting (DVB) bitstreams.

Confidential

**Figure 118: Teletext data transfer sequence**




The duration of the TTXS window is 1402 reference clock periods (51.926 µs), which corresponds to the duration of 360 teletext bits.

The delay between signal TTXTREQ becoming high and the transfer of the first bit of the teletext packet is between two and nine 27 MHz clock cycles. This delay is programmed by register bits DEN_TTX1.TTXDEL[2:0]. The value written to this register is increased by two 27 MHz clock cycles, so the value 0 corresponds to an overall delay of 2 x 27 MHz clock cycles, and the value 7 corresponds to a delay of 9 x 27 MHz clock cycles.

Confidential

## 45.4 Interrupt control

Teletext interrupts can be programmed by register TTXT_INT_EN to interrupt the CPU whenever one of the following occurs.

- A teletext data transfer is complete.
- The current video frame toggles odd-to-even or even-to-odd.

The interrupt status is given by register TTXT_INT_STA and masked by TTXT_INT_EN. The interrupt bits are reset when the CPU writes to the acknowledge register, or when a DMA operation is completed.

## 45.5 DENC teletext registers

Five dedicated digital encoder registers program the teletext encoding in various areas of the vertical blanking interval (VBI) of each field. Four of these areas (that is, blocks of contiguous teletext lines) can **independently** be defined within the two VBIs of one frame (for example, two blocks in each VBI, or three blocks in field 1 VBI and one in field 2 VBI). In certain circumstances, up to four areas may be defined in each VBI.

The five dedicated digital encoder registers are DEN_TTX1 to DEN_TTX5.

## 45.6 Teletext external interface

**Table 126: Teletext interface pins**

| Pin | Type | Function name | Function description |
|---|---|---|---|
| PIO6[6] | Input | TTXTREQ | Teletext request |
| PIO6[7] | Output | TTXTDATAOUT | Teletext output data |

# 46 Teletext DMA registers

Register addresses are provided as *TTXTBaseAddress* + offset.

The *TTXTBaseAddress* is:

0x0200 6000.

**Table 127: Teletext register summary**

| Register | Function | Offset | Type |
|---|---|---|---|
| TTXT_DMA_ADDR | Teletext DMA address | 0x00 | R/W |
| TTXT_DMA_CNT | Teletext DMA count | 0x04 | R/W |
| TTXT_OUT_DLY | Teletext output delay | 0x08 | R/W |
| TTXT_INT_STA | Teletext interrupt status | 0x18 | RO |
| TTXT_INT_EN | Teletext interrupt enable | 0x1C | R/W |
| TTXT_ACK_ODDEVEN | Teletext acknowledge odd or even | 0x20 | WO |
| TTXT_ABORT | Teletext abort | 0x24 | WO |

Confidential

### TTXT_DMA_ADDR Teletext DMA address

| 31 | 30 | 29 | 28 | 28 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DMA_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *TTXTBaseAddress* + 0x00

Type: R/W

Reset: Undefined

Description: This 32-bit register specifies the base address in memory for the DMA transfer from memory.

### TTXT_DMA_CNT Teletext DMA count

| 31 | 30 | 29 | 28 | 28 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | DMA_CNT | | | | | | | | | | |

Address: *TTXTBaseAddress* + 0x04

Type: R/W

Reset: Undefined

Description: This register specifies the number of bytes to be transferred from memory during the DMA operation. A write to this register also starts the teletext output operation.

- For teletext output operation, this value must be:
  46 x *number_of_teletext_lines_to_send* bytes.
- For teletext input operation, the value must be:
  42 x *number_of_teletext_lines_to_receive* bytes.

### TTXT_OUT_DLY Teletext output delay

| 31 | 30 | 29 | 28 | 28 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | | | DELAY | | | | | | | | |

Address: *TTXTBaseAddress* + 0x08

Type: R/W

Reset: Undefined

Description: This register programs the delay, in 27 MHz clock periods, from the rising edge of TTXTREQ to the first valid teletext data bit, that is, TTXTDATAOUT starting to transmit.

### TTXT_INT_STA Teletext interrupt status

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | EVEN | ODD | INOUT_ COMPLETE |

Address: *TTXTBaseAddress* + 0x18

Type: Read only

Reset: Undefined

Description: This register gives the current state of the teletext operations. If the appropriate bits in TTXT_INT_EN are set, interrupts can be driven by the state of this register.

[31:3] **Reserved**

[2] **EVEN**: Current (video encoder) field is even.

[1] **ODD**: Current (video encoder) field is odd.

[0] **INOUT_COMPLETE**: Teletext input or output operation completed.

### TTXT_INT_EN Teletext interrupt enable

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | EVEN_EN | ODD_EN | INOUT_ COMPLETE_EN |

Address: *TTXTBaseAddress* + 0x1C

Type: R/W

Reset: Undefined

Description: This register allows masking of register TTXT_INT_STA.

[31:3] **Reserved**

[2] **EVEN_EN**: Enable even field interrupt.

[1] **ODD_EN**: Enable odd field interrupt.

[0] **INOUT_COMPLETE_EN**: Enable teletext input or output operation completed interrupt.

### TTXT_ACK_ODDEVEN Teletext acknowledge odd or even

Address: *TTXTBaseAddress* + 0x20

Type: WO

Description: This register acknowledges the odd/even toggle interrupt. Any write to this register clears bits ODD and EVEN of register TTXT_INT_STA.

Confidential

### TTXT_ABORT Teletext abort

Address: *TTXTBaseAddress* + 0x24

Type: WO

Description: Any write to this address causes the teletext interface to abort the current operation. The state of the teletext operation is reset, and the teletext data transfer is interrupted. The DMA engine is reset only after the current word read or write is complete.

Confidential

# 47 2D graphics processor (blitter)

## 47.1 Overview

The 2D-graphics processor is a CPU accelerator for graphics picture handling. It is a dual-source DMA, with a set of powerful operators. It receives data from the local memory through two input sources, source 1 and source 2.

- Source 1 is used for frequent operations such as color-fill or simple source-copy; it has a 64-bit wide internal bus and performs according to the pixel format.
- All operators always apply to source 2. The processing pipeline bus is always a pixel bus (ARGB8888 format), whatever the format of the source inputs.
- Sources 1 and 2 are used simultaneously for read/modify/write operations.

The 2D-graphics processor is software-controlled by a link-list mechanism. Each node of the link list is an instruction that contains all the necessary information to proceed.

The 2D-graphics processor operates at a clock rate of up to 153 MHz. For each operation involving source 2, the maximum output rate is 153 Mpixel/s, whatever the pixel depth. This rate is constant except for 2D resizing for downsampling, where it is 100 x HSF x VSF (horizontal and vertical scaling factors). When source 1 is used in direct-copy mode, the internal bus width is 64 bits and the maximum output rate is 800 Mbyte/s. These values are maximum performances that the blitter can reach; they assume that no read/write memory word request has been postponed by the memory arbiter mechanism.

**Table 128: Performance for typical operations over a range of target pixel sizes (Mpix/s, at output), for a 100 MHz clock**

| Operation | Performance (in megapixels) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 bpp | 2 bpp | 4 bpp | 8 bpp | 16 bpp | 24 bpp | 32 bpp |
| Fill (source 1) | 4800 | 2400 | 1200 | 600 | 300 | 200 | 150 |
| One-source blit (source 1) | 4800 | 2400 | 1200 | 600 | 300 | 200 | 150 |
| One-source blit (source 2) shade | 100 x GSF[a] | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF |
| Two-source blit | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF | 100 x GSF |

a. GSF (global scaling factor) = HSF x VSF.

The 4:2:0 plane can be used as a source for the 2D-graphics engine. Because this plane uses both the source 1 and source 2 buffers, special modes are required. These modes are described in *Section 47.6 on page 452*.

Confidential

## 47.2 Functions

- Solid color fill of rectangular window.
- Solid color shade (fill plus alpha blending).
- One source copy, with one or several operators enabled (color format conversion, 2D scaling).
- Two-source copy with alpha blending or logical operations between them.
- 4:2:2 raster as source or destination format.
- 4:2:2 and 4:2:0 macroblock as a source format.
- Color space conversion RGB to or from YCbCr.
- Color expansion (CLUT to true color).
- Color correction (gamma, contrast, gain).
- Color reduction (true color to CLUT1, CLUT8 and CLUT4 or CLUT2) using an error diffusion algorithm.
- 2D resize engine with high quality filtering.
- Adaptive flicker filter from memory to memory.
- Color keying capability.
- Rectangular clipping.
- Programmable source/target scanning direction, both horizontally and vertically, in order to handle overlapping source and destination areas correctly.
- 1-bit/8-bit clipmask bitmap so random shape clipping can be achieved in two passes.
- Plane mask feature available.
- Special XYL access mode, to speed random pixel access, or horizontal line drawing (such as polygon filling or run-length decoder accelerator). See Section 47.7 on page 453.

Source and destination windows are all defined using an XY descriptor, with pixel accuracy from 1 to 32 bpp, whatever the format.

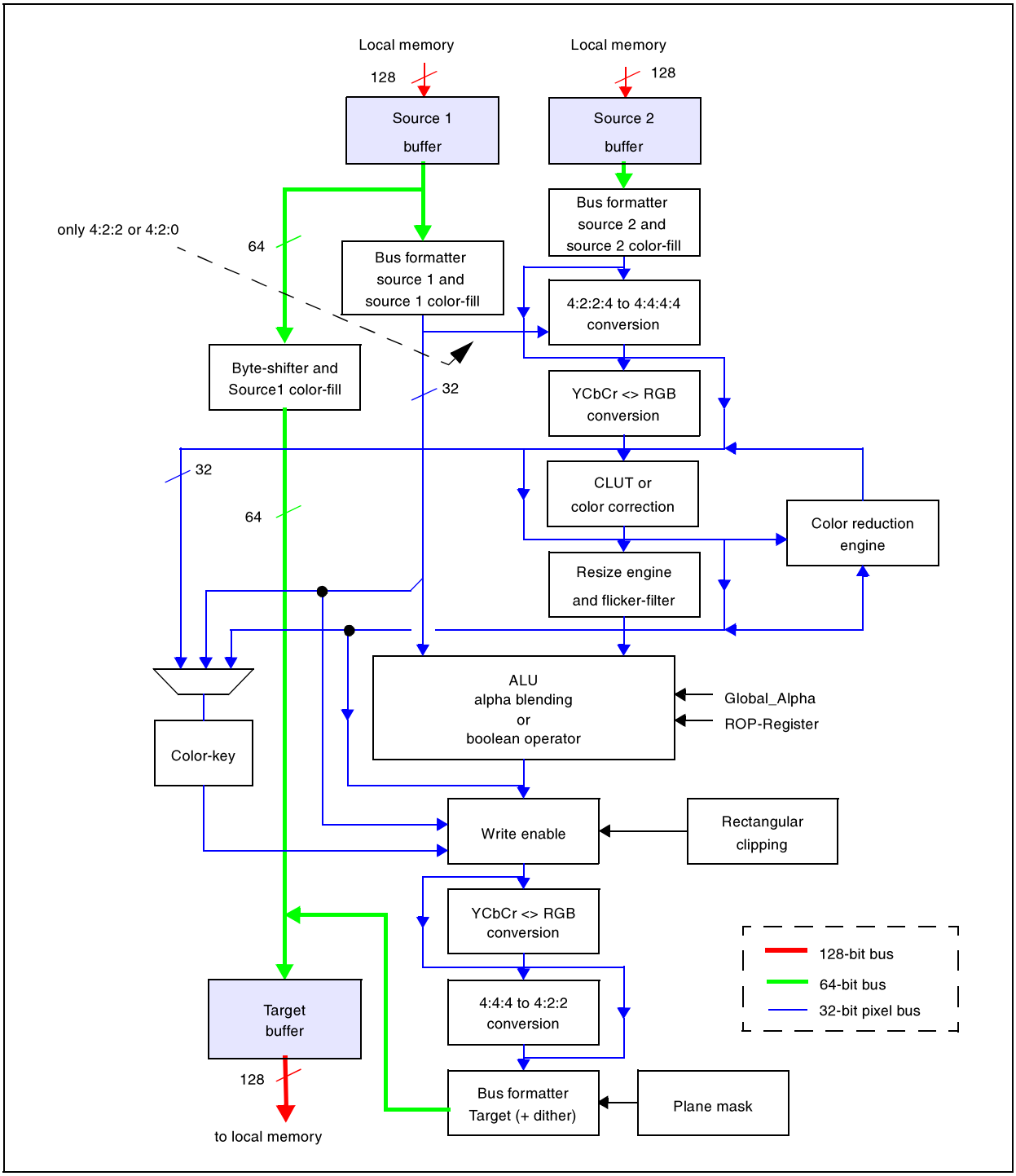
Most of these operators can be combined in a single blitter pass. For instance, a YCbCr 4:2:2 bitmap can be converted to 4:4:4 RGB, resized and finally blended on an RGB565 background picture.

Confidential

## 47.3 Functional block diagram

The following figure illustrates the blitter functions that can be performed. Each of the blitter functions is briefly described in Section 47.5 on page 433.

**Figure 119: 2D graphics processor data flow**




The blitter works from memory to memory with a dual or single source and one target. The target can be one of the sources. A set of overlaps between sources and target is supported, provided that the pixmap horizontal and vertical scan ordering are correctly programmed for each source and the target. Each source and target may be programmed independently.

Each source and the target is associated with a specific set of registers. Another register (BLT_INS) is used to control the data flow with operator enables. Each time an operator is

Confidential

enabled, the user has to specify its behavior with operator specific registers. If the blitter is not in direct mode (64-bit internal bus), the ALU operator is always enabled and must be programmed.

### 47.3.1 The blitter process

The 2D graphics processor has control and status registers allowing global blitter operation management:

- BLT_CTRL is the control register where the user can launch a blitter link-list sequence, suspend the blitter sequence and reset the hardware block.
- BLT_STA1, BLT_STA2, BLT_STA3 are status registers used to monitor the blitter operation states, such as the hardware block state (ready or busy), the current blitter node address being processed, the current target line number, and some interrupt management.

The basic concept of a blitter is to program one or several nodes in memory, where each node operates on all blitter registers except the control and status registers. For several nodes, a link-list mechanism must be set. The hardware can then automatically read the next node located inside the local memory and continue a sequence of blitter operations. This mechanism allows multipass operations for complex sequences in order to limit software load. This link-list mechanism is provided by register BLT_NIP that contains the next blitter address. If this pointer is set to zero, the hardware understands that the link-list mechanism has ended.

The procedure to start the blitter is the following:

1. Prepare the link list of nodes in the local memory.
2. Program the first blitter node address inside hardware BLT_NIP.
3. Set the trigger start condition if needed (BLT_INS.TRIG_COND_CTRL) and start the blitter by setting bit START (BLT_CTRL), and then setting it back to 0.
4. If trigger on a raster scan line has been selected, the user must program register BLT_RST to set the line number.

*Note: The blitter sees a 32-bit address space, but always writes inside a 64 Mbyte bank for a given object (for example, source-target node).*

The blitter can set three kinds of interrupt that may be masked by IRQ_MASK (BLT_INS):

- once a blitter pass is completed and before the next has been loaded,
- after the blitter instructions have been loaded,
- once the blitter has reached the idle state following the suspend command.

For each interrupt, the blitter stops and the user should restart it to continue.

Confidential

### 47.3.2 Data flow control

SOURCE1 (register BLT_INS) sets the source 1 mode: direct copy, direct fill, fetch pixel flow from memory, or color fill.

If the direct mode is selected for source 1 (high pixel throughput using the 64-bit internal bus), the bus formatter is removed from the path. The target format should thus match the source 1 format, and no graphics operation may be used except for color fill or pixel copy.

Color fill mode replaces the pixel flow from the local memory by an internal flow of constant color pixels that match the source specification (format, number of pixels: window size, expansion mode).

SOURCE2 (BLT_INS) sets the source 2 mode: either fetch pixel flow from memory, or color fill.

Bit ICSC enables the input color space converter for source 2, and bit OCSC enables the output color space converter for the target (BLT_INS).

*Note: The output color space converter (from RGB to YCbCr or from YCbCr to RGB) and the input color space converter, if enabled simultaneously, can only perform opposite conversions.*

Bit CLUTOP enables the CLUT operator on source 2, 2DRESCALE enables the 2D resize operator on source 2, and FLICK_FILT enables the flicker filter operator on source 2 (BLT_INS).

Bit RECT_CLIP enables the rectangle clipping operator on target, bit CKEY enables the color key operator from source 1 or source 2 to target, and bit PLANE_MASK enables the plane mask bit protection operation on target with respect to source 1 (BLT_INS).

#### User information

Register BLT_USR is used as user information for the node for software purposes.

*Note: As a result of the number of input formats, output formats, available operators and different configurations, many register combinations have no meaning in terms of the graphics blitter. Registers must be programmed with this in mind.*

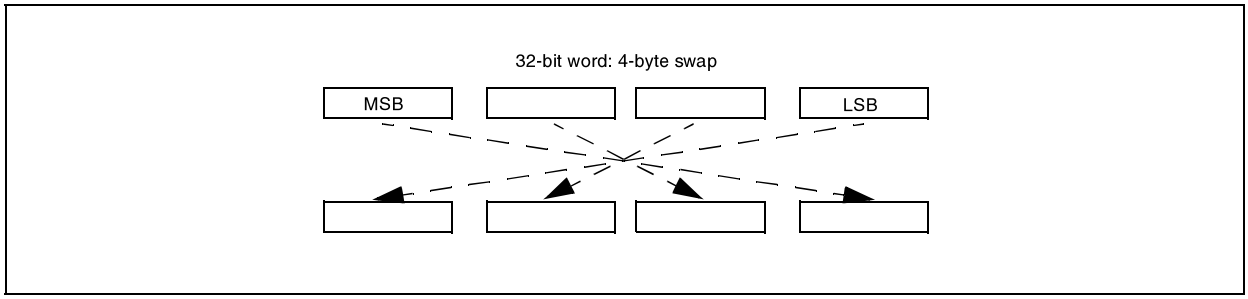
Confidential

## 47.4 Endianness support

Bit (BLT_PKZ.BIGNOTLITTLE) indicates whether the nodes, filters coefficients, CLUT palette, and XYL memory buffer content are stored in big or little endian format. This bit must be compliant with the host CPU endianness (little endian). The EMPI endianness setting should also be the same.

When the nodes, filter coefficients and CLUT palette are fetched, a four-byte swap is performed if BLT_PKZ.BIGNOTLITTLE is set.

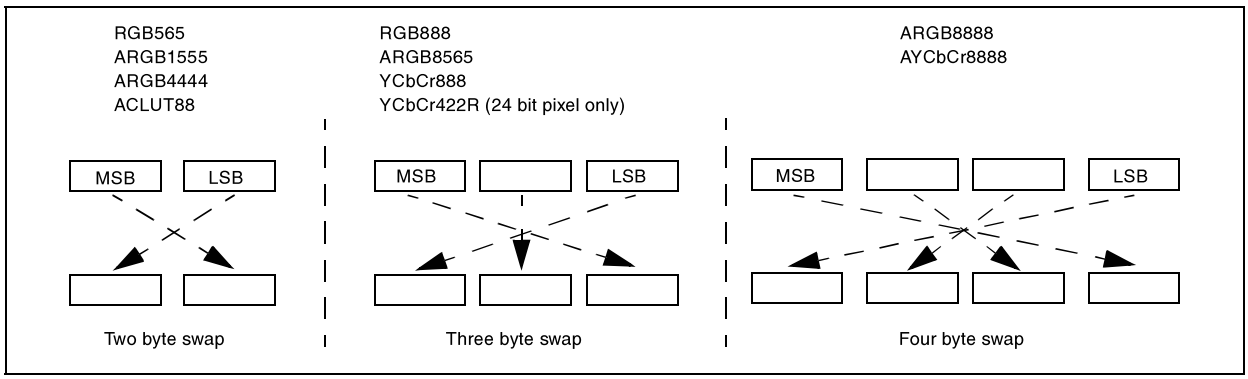
**Figure 120: Endianness conversion for nodes, filter coefficients, CLUT, XYL parameters**




For a given node, the endianness may be defined for the source 1 bitmap, source 2 bitmap, and target bitmap (BIGNOTLITTLE in BLT_S1TY, BLT_S2TY and BLT_TTY). Usually bitmap endianness is in line with the host CPU endianness, and these bits are programmed in the same way as BIGNOTLITTLE in BLT_PKZ. However, when importing a bitmap, for example, its endianness can be converted. This corrects all potential endianness issues. The endianness

format conversion affects only the 16 bpp, 24 bpp and 32 bpp formats. For YCbCr422R format, the pixels that only have 8 bit luminance remain unaffected, only the 24 bit pixels are swapped.

**Figure 121: Endianness conversion for big endian pixels**




In direct copy mode, the endianness of the bitmap is not taken into account. This is a pure memory to memory operation without any modification.

## 47.5 Blitter functions

### 47.5.1 Source 2 address generator and bus formatter

The source 2 address generator scans a rectangular window, according to an XY addressing scheme. The XY pixel address is converted into a physical memory address, using the source 2 description registers.

Reference formula for address conversion:

$$MemAddress\ (x, y) = BaseAddress + y * Pitch + x * (number\ of\ bytes\ per\ pixel)$$

Register BLT_S2BA is the memory base address of the bitmap selected for source 2 (absolute (0,0) location, top-left pixel).

Register BLT_S2TY provides the specific properties of source 2: format, pitch, horizontal and vertical scan order, sub-byte ordering (for sub-byte formats), bit accuracy expansion mode, chroma sign (for YCbCr formats), chroma features (YCbCr macro-block formats).

Register BLT_S2XY is the coordinate that specifies the start of the input source 2 window with respect to the base address.

Register BLT_S2SZ specifies the size of the source 2 window.

The source 2 bus formatter converts a 128-bit input bus into a pixel bus, depending on the current format selected for source 2. The internal pixel bus is 32 bits wide, but a subset of the data is used by many color format modes.

All formats are available.

Confidential

The pixel bus format is as follows:

**Table 129A: Source 2 formatter output / true color 4:4:4**

| True color | Alpha | | Color data | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:8 | 7:4 | 3:0 |
| RGB565 | 128 | | R5/3MSB or 000 | | G6/2MSB or 00 | | B5/3MSB or 000 | |
| RGB888 | 128 | | R8 | | G8 | | B8 | |
| ARGB1555 | A1/0000000 | | R5/3MSB or 000 | | G5/3MSB or 000 | | B5/3MSB or 000 | |
| ARGB8565 | A8 | | R5/3MSB or 000 | | G6/2MSB or 00 | | B5/3MSB or 000 | |
| ARGB8888 | A8 | | R8 | | G8 | | B8 | |
| ARGB4444 | 0/A4/100 (but keep 0 and 128) | | R4/4MSB or 0000 | | G4/4MSB or 0000 | | B4/4MSB or 0000 | |
| YCbCr888 | 128 | | Cr8 | | Y8 | | Cb8 | |
| AYCbCr8888 | A8 | | Cr8 | | Y8 | | Cb8 | |

**Table 129B: Source 2 formatter output / CLUT-based formats**

| CLUT-based formats | Alpha | | Index value | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:8 | 7:4 | 3:0 |
| ACLUT1 | 0 (don't care) | | | | | | 0000 000$I_0$ | |
| ACLUT2 | 0 (don't care) | | | | | | 0000 00$I_1I_0$ | |
| ACLUT4 | 0 (don't care) | | | | | | 0000 $I_3I_2I_1I_0$ | |
| ACLUT8 | 0 (don't care) | | | | | | $I_7I_6I_5I_4$ $I_3I_2I_1I_0$ | |
| ACLUT44 | 0/A4/100 (but keep 0 and 128) | | 0 (don't care) | | | | 0000 $I_3I_2I_1I_0$ | |
| ACLUT88 | A8 | | 0 (don't care) | | | | $I_7I_6I_5I_4$ $I_3I_2I_1I_0$ | |

**Table 130: Source 2 formatter output / alpha only formats**

| Alpha-only formats | Alpha value | | Color data | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:8 | 7:4 | 3:0 |
| A1 | A1/0000000 | | 0 | | | | | |
| A8 | A8 | | 0 | | | | | |

**Table 131: Source 2 formatter output / YCbCr 4:2:2 raster format**

| YCbCr 4:2:2 | Color data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:8 | 7:4 | 3:0 |
| YCbCr 4:2:2 (1st pixel YCbCr) | Cb 1st pixel | | Y 1st pixel | | Cr 1st pixel | | Y 2nd pixel | |
| YCbCr 4:2:2 (1st pixel Y-only) | Y 1st pixel | | Cb 2nd pixel | | Y 2nd pixel | | Cr 2nd pixel | |

*Note: In this last mode, no alpha value is produced by the block, and the bus carries two pixels simultaneously. The alpha channel (128 = opaque) is inserted by the 4:2:2 to 4:4:4 converter (next block in the pipeline), that is automatically enabled in this configuration.*

Confidential

When the input picture has an 8-bit per pixel alpha component, this component can have a 0 to 128 or 0 to 255 range. The internal 8-bit alpha format being 0 to 128, a 0 to 255 alpha component is converted using the following formula: $A_{0\ to\ 128} = (A_{0\ to\ 255} + 1) \times 2^{-1}$. The alpha range formatter is present for source 1, source 2 and target.

### 47.5.2 Source 1 address generator and bus formatter

The source 1 address generator scans a rectangular window, according to an XY addressing scheme. The XY pixel address is converted into a physical memory address, using the source 1 registers.

Each time the 2D graphics engine performs a read/modify/write cycle to combine background information with new data, the source 1 bus formatter converts a 128-bit input bus into a pixel bus that feeds the ALU operator.

Source 1 is used for frequent operations such as color-fill or simple source-copy.

In fast direct-copy mode, the source 1 bus formatter is unused, and the transfer uses a 64-bit wide internal bus, whatever the color format and its bit-depth. Rectangular graphics areas are transferred faster, but sub-byte color formats are not supported. Neither the plane mask nor rectangular clipping features can be used.

In normal mode, source 1 data can be combined with source 2 data.

Register BLT_S1BA is the memory base address of the bitmap selected for source 2 (absolute (0,0) location, top-left pixel).

BLT_S1TY provides the specific properties of source 1: format, pitch, horizontal and vertical scan order, sub-byte ordering (for sub-byte formats), color depth expansion mode.

BLT_S1XY is the coordinate that specifies the start of the source 1 input window with respect to the base address.

BLT_S1SZ specifies the size of the source 1 window (equals the target size).

### 47.5.3 Color fill

Registers BLT_S1CF and BLT_S2CF can provide the solid-color value for filling during a blitter operation. BLT_S1CF allows the direct-fill mode to be used, for faster performance, but not in sub-byte modes.

The number of significant bits varies from 1 bpp to 32 bpp according to the bit-depth of the color format.

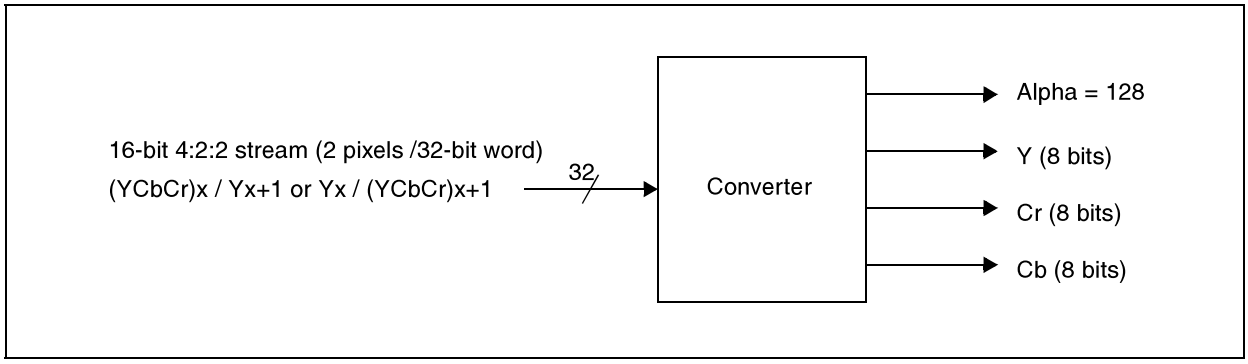
All color formats are supported, except YCbCr4:2:0MB and YCbCr4:2:2MB.

Confidential

### 47.5.4 4:2:2 to 4:4:4 conversion (horizontal chroma upsampler)

This conversion applies to source 2 only if the source format set in register BLT_S2TY is YCbCr422raster. The 4:2:2 input is a color-only raster signal. An opaque alpha channel is added at the block output.

**Figure 122: 4:2:2 to 4:4:4 converter**




Missing chroma samples are recovered with a 2-tap interpolator. Possible side effects on the edges are handled using sample duplication, if required, using the following schemes.

The input source line starts at address $x_0$, and contains *n* pixels.

Convention: when $x_0$ is even, the first pixel is a complete CbYCr pixel, when $x_0$ is odd, the first pixel is a Y-only pixel. Four cases must be taken into account:

**Table 132: $x_0$ even / *n* even**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Input** | | | | | | |
| $Cb_1Y_1Cr_1$ | $Y_2$ | $Cb_3Y_3Cr_3$ | $Y_4$ | ..... | $Cb_{n-1}Y_{n-1}Cr_{n-1}$ | $Y_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $(Cb_1+Cb_3) / 2$ | $Cb_3$ | $(Cb_3+Cb_5) / 2$ | ..... | $Cb_{n-1}$ | $Cb_{n-1}$ |
| $Cr_1$ | $(Cr_1+Cr_3) / 2$ | $Cr_3$ | $(Cr_3+Cr_5) / 2$ | ..... | $Cr_{n-1}$ | $Cr_{n-1}$ |

**Table 133: $x_0$ even / *n* odd**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Input** | | | | | | |
| $Cb_1Y_1Cr_1$ | $Y_2$ | $Cb_3Y_3Cr_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Cb_nY_nCr_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $(Cb_1+Cb_3) / 2$ | $Cb_3$ | $(Cb_3+Cb_5) / 2$ | ..... | $(Cb_{n-2}+Cb_n) / 2$ | $Cb_n$ |
| $Cr_1$ | $(Cr_1+Cr_3) / 2$ | $Cr_3$ | $(Cr_3+Cr_5) / 2$ | ..... | $(Cr_{n-2}+Cr_n) / 2$ | $Cr_n$ |

Confidential

**Table 134: $x_0$ odd / *n* odd**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Input** | | | | | | |
| $Y_1$ | $Cb_2Y_2Cr_2$ | $Y_3$ | $Cb_4Y_4Cr_4$ | ..... | $Cb_{n-1}Y_{n-1}Cr_{n-1}$ | $Y_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_2$ | $Cb_2$ | $(Cb_2+Cb_4) / 2$ | $Cb_4$ | ..... | $Cb_{n-1}$ | $Cb_{n-1}$ |
| $Cr_2$ | $Cr_2$ | $(Cr_2+Cr_4) / 2$ | $Cr_4$ | ..... | $Cr_{n-1}$ | $Cr_{n-1}$ |

**Table 135: $x_0$ odd / *n* even**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Input** | | | | | | |
| $Y_1$ | $Cb_2Y_2Cr_2$ | $Y_3$ | $Cb_4Y_4Cr_4$ | ..... | $Y_{n-1}$ | $Cb_nY_nCr_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_2$ | $Cb_2$ | $(Cb_2+Cb_4) / 2$ | $Cb_4$ | ..... | $(Cb_{n-2}+Cb_n) / 2$ | $Cb_n$ |
| $Cr_2$ | $Cr_2$ | $(Cr_2+Cr_4) / 2$ | $Cr_4$ | ..... | $(Cr_{n-2}+Cr_n) / 2$ | $Cr_n$ |

Confidential

### 47.5.5 4:4:4 to 4:2:2 conversion (horizontal chroma downsampler on target)

This block is automatically enabled by the hardware if the target output format is YCbCr 4:2:2 raster. In this case, neither vertical resize nor flicker filter can be used at the same time as horizontal resize.

The sampling rate conversion for the chroma component uses a three-tap 1:2:2 digital filter, as shown in the next tables.

The output target line starts at address $x_0$, and contains *n* pixels.

Convention: when $x_0$ is even, the first pixel is a complete CbYCr pixel, when $x_0$ is odd, the first pixel is a Y-only pixel. Four cases must be taken into account:

**Table 136: $x_0$ even / *n* even**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Inputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $Cb_2$ | $Cb_3$ | $Cb_4$ | ..... | $Cb_{n-1}$ | $Cb_n$ |
| $Cr_1$ | $Cr_2$ | $Cr_3$ | $Cr_4$ | ..... | $Cr_{n-1}$ | $Cr_n$ |
| **Outputs** | | | | | | |
| $Y_1$<br>$(Cb_1+2Cb_1+Cb_2) / 4$<br>$(Cr_1+2Cr_1+Cr_2) / 4$ | $Y_2$ | $Y_3$<br>$(Cb_2+2Cb_3+Cb_4) / 4$<br>$(Cr_2+2Cr_3+Cr_4) / 4$ | $Y_4$ | ..... | $Y_{n-1}$<br>$(Cb_{n-2}+2Cb_{n-1}+Cb_n) / 4$<br>$(Cr_{n-2}+2Cr_{n-1}+Cr_n) / 4$ | $Y_n$ |

**Table 137: $x_0$ even / *n* odd**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Inputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $Cb_2$ | $Cb_3$ | $Cb_4$ | ..... | $Cb_{n-1}$ | $Cb_n$ |
| $Cr_1$ | $Cr_2$ | $Cr_3$ | $Cr_4$ | ..... | $Cr_{n-1}$ | $Cr_n$ |
| **Outputs** | | | | | | |
| $Y_1$<br>$(\mathbf{Cb_1}+2Cb_1+Cb_2) / 4$<br>$(\mathbf{Cr_1}+2Cr_1+Cr_2) / 4$ | $Y_2$ | $Y_3$<br>$(Cb_2+2Cb_3+Cb_4) / 4$<br>$(Cr_2+2Cr_3+Cr_4) / 4$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$<br>$(Cb_{n-1}+2Cb_n+\mathbf{Cb_n}) / 4$<br>$(Cr_{n-1}+2Cr_n+\mathbf{Cr_n}) / 4$ |

**Table 138: $x_0$ odd / *n* odd**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Inputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $Cb_2$ | $Cb_3$ | $Cb_4$ | ..... | $Cb_{n-1}$ | $Cb_n$ |
| $Cr_1$ | $Cr_2$ | $Cr_3$ | $Cr_4$ | ..... | $Cr_{n-1}$ | $Cr_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$<br>$(Cb_1+2Cb_2+Cb_3) / 4$<br>$(Cr_1+2Cr_2+Cr_3) / 4$ | $Y_3$ | $Y_4$<br>$(Cb_3+2Cb_4+Cb_5) / 4$<br>$(Cr_3+2Cr_4+Cr_5) / 4$ | ..... | $Y_{n-1}$<br>$(Cb_{n-2}+2Cb_{n-1}+Cb_n) / 4$<br>$(Cr_{n-2}+2Cr_{n-1}+Cr_n) / 4$ | $Y_n$ |

**Table 139: $x_0$ odd / *n* even**

| Pixel 1 (x0) | Pixel 2 | Pixel 3 | Pixel 4 | ..... | Pixel *n*-1 | Pixel *n* |
|---|---|---|---|---|---|---|
| **Inputs** | | | | | | |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | ..... | $Y_{n-1}$ | $Y_n$ |
| $Cb_1$ | $Cb_2$ | $Cb_3$ | $Cb_4$ | ..... | $Cb_{n-1}$ | $Cb_n$ |
| $Cr_1$ | $Cr_2$ | $Cr_3$ | $Cr_4$ | ..... | $Cr_{n-1}$ | $Cr_n$ |
| **Outputs** | | | | | | |
| $Y_1$ | $Y_2$<br>$(Cb_1+2Cb_2+Cb_3) / 4$<br>$(Cr_1+2Cr_2+Cr_3) / 4$ | $Y_3$ | $Y_4$<br>$(Cb_3+2Cb_4+Cb_5) / 4$<br>$(Cr_3+2Cr_4+Cr_5) / 4$ | ..... | $Y_{n-1}$ | $Y_n$<br>$(Cb_{n-1}+2Cb_n+\mathbf{Cb_n}) / 4$<br>$(Cr_{n-1}+2Cr_n+\mathbf{Cr_n}) / 4$ |

### 47.5.6 YCbCr-to-RGB conversion

The 2D graphics engine color converter complies with ITU-R BT.601 and ITU-R BT.709 color systems, and the RGB components are gamma-corrected.

#### Graphics matrix

The following ranges are assumed:

0 ≤R, G, B ≤255
16 ≤Y ≤235
16 ≤Cb, Cr ≤240 in offset binary representation, -112 ≤Cb, Cr ≤+112 in two's complement signed representation

Confidential

Converting from YCbCr to RGB:

- The Y component is clipped between 16 and 235, before applying the matrix.
- The Cb and Cr components can be signed or unsigned.
- Once signed, the chroma range is -112 to +112.
- RGB output components are saturated between 0 and 255.

The next tables show the hardwired matrices used for the conversion (assuming offset binary chroma):

**Table 140: 601 colorimetry / floating-point matrix / digital range**

| YCbCr to RGB reference floating-point matrix | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 1.1641 | x (Y - 16) | + | 0.0 | x (Cb - 128) | + | 1.5958 | x (Cr - 128) |
| G | = | 1.1641 | x (Y - 16) | - | 0.3914 | x (Cb - 128) | - | 0.8135 | x (Cr - 128) |
| B | = | 1.1641 | x (Y - 16) | + | 2.0178 | x (Cb - 128) | + | 0.0 | x (Cr - 128) |

**Table 141: 601 colorimetry / integer matrix / digital range**

| YCbCr to RGB integer matrix as implemented (1.0 <> 256) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 298 | x (Y - 16) | + | 0 | x (Cb - 128) | + | 409 | x (Cr - 128) |
| G | = | 298 | x (Y - 16) | - | 100 | x (Cb - 128) | - | 208 | x (Cr - 128) |
| B | = | 298 | x (Y - 16) | + | 517 | x (Cb - 128) | + | 0 | x (Cr - 128) |

**Table 142: 709 colorimetry / floating-point matrix / digital range**

| YCbCr to RGB reference floating-point matrix | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 1.1644 | x (Y - 16) | + | 0.0 | x (Cb - 128) | + | 1.7930 | x (Cr - 128) |
| G | = | 1.1644 | x (Y - 16) | - | 0.2129 | x (Cb - 128) | - | 0.5326 | x (Cr - 128) |
| B | = | 1.1644 | x (Y - 16) | + | 2.1128 | x (Cb - 128) | + | 0.0 | x (Cr - 128) |

**Table 143: 709 colorimetry / floating-point matrix / digital range**

| YCbCr to RGB integer matrix as implemented (1.0 <> 256) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 298 | x (Y - 16) | + | 0 | x (Cb - 128) | + | 459 | x (Cr - 128) |
| G | = | 298 | x (Y - 16) | - | 54 | x (Cb - 128) | - | 136 | x (Cr - 128) |
| B | = | 298 | x (Y - 16) | + | 541 | x (Cb - 128) | + | 0 | x (Cr - 128) |

### Video matrix

In the case of video matrix, Y, Cb and Cr are not clipped as in the graphics matrix. The following ranges are assumed:

0 ≤R, G, B ≤255
0 ≤Y ≤255
0 ≤Cb, Cr ≤255 in offset binary representation, -128 ≤Cb, Cr ≤+127 in two's complement signed representation.

Converting from YCbCr to RGB:

- The Y component is not clipped before applying the matrix.
- The Cb and Cr components can be signed or unsigned.
- Once signed, the chroma range is -128 to +127.
- RGB output components are saturated between 0 and 255.

Confidential

The next tables show the hardwired matrices used for the conversion (assuming offset binary chroma):

**Table 144: 601 colorimetry / floating-point matrix / digital range**

| YCbCr to RGB reference floating-point matrix | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 1 | x (Y) | + | 0.0 | x (Cb - 128) | + | 1.3828 | x (Cr - 128) |
| G | = | 1 | x (Y) | - | 0.3359 | x (Cb - 128) | - | 0.6992 | x (Cr - 128) |
| B | = | 1 | x (Y) | + | 1.7344 | x (Cb - 128) | + | 0.0 | x (Cr - 128) |

**Table 145: 601 colorimetry / integer matrix / digital range**

| YCbCr to RGB integer matrix as implemented (1.0 <> 256) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 256 | x (Y) | + | 0 | x (Cb - 128) | + | 351 | x (Cr - 128) |
| G | = | 256 | x (Y) | - | 86 | x (Cb - 128) | - | 179 | x (Cr - 128) |
| B | = | 256 | x (Y) | + | 444 | x (Cb - 128) | + | 0 | x (Cr - 128) |

**Table 146: 709 colorimetry / floating-point matrix / digital range**

| YCbCr to RGB reference floating-point matrix | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 1 | x (Y) | + | 0 | x (Cb - 128) | + | 1.5391 | x (Cr - 128) |
| G | = | 1 | x (Y) | - | 0.1836 | x (Cb - 128) | - | 0.4570 | x (Cr - 128) |
| B | = | 1 | x (Y) | + | 1.8125 | x (Cb - 128) | + | 0 | x (Cr - 128) |

**Table 147: 709 colorimetry / integer matrix / digital range**

| YCbCr to RGB integer matrix as implemented (1.0 <> 256) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | = | 256 | x (Y)) | + | 0 | x (Cb - 128) | + | 394 | x (Cr - 128) |
| G | = | 256 | x (Y) | - | 47 | x (Cb - 128) | - | 117 | x (Cr - 128) |
| B | = | 256 | x (Y) | + | 464 | x (Cb - 128) | + | 0 | x (Cr - 128) |

Confidential

### 47.5.7 RGB-to-YCbCr conversion

#### Graphics matrix

Converting from RGB to YCbCr:

- The Y component is clipped between 16 and 235, after applying the matrix.
- The chroma range is -112 to +112, and can be optionally encoded in offset-binary format (+128, 16/240 range).

In register BLT_CCO, CCO_INCOL sets the input color converter colorimetry (601 or 709), and CCO_INSIGN sets the input color converter chroma color sign. CCO_INDIR sets the input color converter direction (RGB to YCbCr or YCbCr to RGB), and the output color converter is automatically programmed with the other direction.

CCO_OUTCOL sets the output color converter colorimetry (601 or 709), and CCO_OUTSIGN sets the output color converter chroma color format: signed or unsigned.

CCO_INGFX*n*VID sets the input color converter to graphics or video matrix, and CCO_OUTGFX*n*VID sets the output color converter to graphics or video matrix.

The next tables show the hardwired matrices used for the conversion (assuming offset binary chroma):

**Table 148: 601 colorimetry / floating-point matrix / digital range**

| RGB to YCbCr reference floating-point matrix | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 0.257 | xR | + | 0.504 | xG | + | 0.098 | xB | + | 16 |
| Cb | = | - | 0.148 | xR | - | 0.291 | xG | + | 0.439 | xB | + | 128 |
| Cr | = | | 0.439 | xR | - | 0.368 | xG | - | 0.071 | xB | + | 128 |

**Table 149: 601 colorimetry / integer matrix / digital range**

| RGB to YCbCr integer matrix as implemented (1.0 <> 1024) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 263 | xR | + | 516 | xG | + | 100 | xB | + | 16 |
| Cb | = | - | 152 | xR | - | 298 | xG | + | 450 | xB | + | 128 |
| Cr | = | | 450 | xR | - | 377 | xG | - | 73 | xB | + | 128 |

**Table 150: 709 colorimetry / floating-point matrix / digital range**

| RGB to YCbCr reference floating-point matrix | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 0.1825 | xR | + | 0.6144 | xG | + | 0.0619 | xB | + | 16 |
| Cb | = | - | 0.1006 | xR | - | 0.3386 | xG | + | 0.4392 | xB | + | 128 |
| Cr | = | | 0.4392 | xR | - | 0.3990 | xG | - | 0.0402 | xB | + | 128 |

**Table 151: 709 colorimetry / integer matrix / digital range**

| RGB to YCbCr integer matrix as implemented (1.0 <> 1024) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 187 | xR | + | 629 | xG | + | 63 | xB | + | 16 |
| Cb | = | - | 103 | xR | - | 347 | xG | + | 450 | xB | + | 128 |
| Cr | = | | 450 | xR | - | 409 | xG | - | 41 | xB | + | 128 |

Confidential

### Video matrix

Converting from RGB to YCbCr:

- The Y component is clipped between 1 and 254, after applying the matrix.
- The chroma range is -127 to +126, and can be optionally encoded in offset-binary format (+128, 1/254 range).

The next tables show the hardwired matrices used for the conversion (assuming offset binary chroma):

**Table 152: 601 colorimetry / floating-point matrix / digital range**

| RGB to YCbCr reference floating-point matrix | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 0.2988 | xR | + | 0.5869 | xG | + | 0.1143 | xB | + | |
| Cb | = | - | 0.1729 | xR | - | 0.3389 | xG | + | 0.5107 | xB | + | 128 |
| Cr | = | | 0.5107 | xR | - | 0.4277 | xG | - | 0.0830 | xB | + | 128 |

**Table 153: 601 colorimetry / integer matrix / digital range**

| RGB to YCbCr integer matrix as implemented (1.0 <> 1024) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 306 | xR | + | 601 | xG | + | 117 | xB | + | |
| Cb | = | - | 177 | xR | - | 347 | xG | + | 523 | xB | + | 128 |
| Cr | = | | 523 | xR | - | 438 | xG | - | 85 | xB | + | 128 |

**Table 154: 709 colorimetry / floating-point matrix / digital range**

| RGB to YCbCr reference floating-point matrix | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 0.2129 | xR | + | 0.7148 | xG | + | 0.0723 | xB | + | |
| Cb | = | - | 0.1172 | xR | - | 0.3945 | xG | + | 0.5117 | xB | + | 128 |
| Cr | = | | 0.5117 | xR | - | 0.4648 | xG | - | 0.0469 | xB | + | 128 |

**Table 155: 709 colorimetry / integer matrix / digital range**

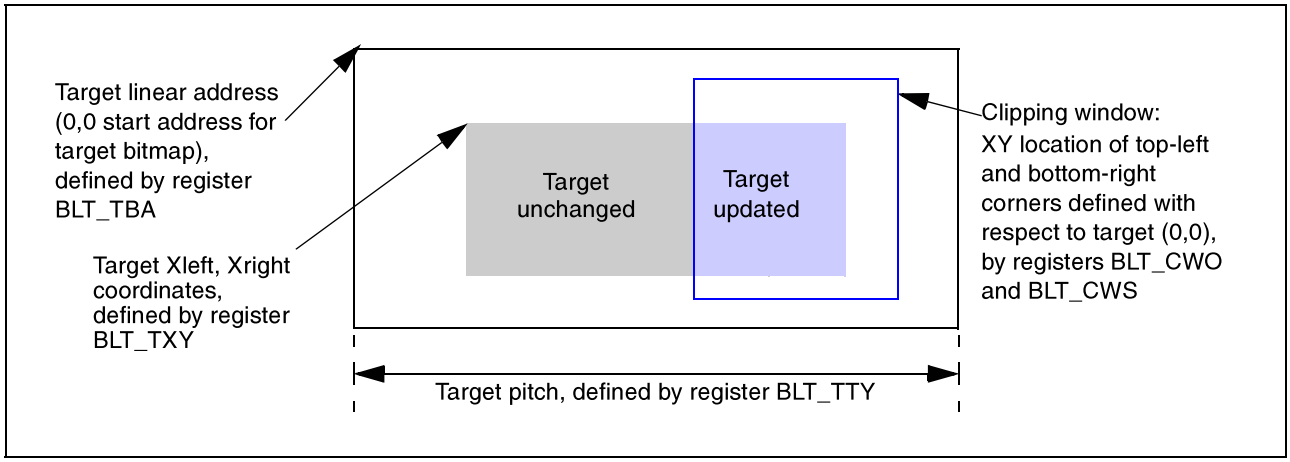
| RGB to YCbCr integer matrix as implemented (1.0 <> 1024) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | = | | 218 | xR | + | 732 | xG | + | 74 | xB | + | |
| Cb | = | - | 120 | xR | - | 404 | xG | + | 524 | xB | + | 128 |
| Cr | = | | 524 | xR | - | 476 | xG | - | 48 | xB | + | 128 |

### 47.5.8 Rectangular clipping

Each write access to the target plane can be enabled on a pixel-per-pixel basis, with respect to a rectangular window.

The base address format of the 2D-graphics engine is in x-y coordinates. Hardware clipping uses four registers to define the rectangular clipping area. The figure below shows a block transfer (BLT) operation inside a buffer, using a hardware clipping window. The buffer is defined using a linear start address. The BLT area and the clipping window are defined by x-y coordinates. The block transfer function is only valid inside the clipping window.

**Figure 123: Hardware clipping window**




Configuration bit BLT_CWO.INTNL inverses the clipping window; the target area is updated outside the clipping window and is protected inside the clipping window.

XDO and YDO (register BLT_CWO) define the window start with respect to the target base address.

XDS and YDS (BLT_CWS) define the window stop with respect to the target base address.

Confidential

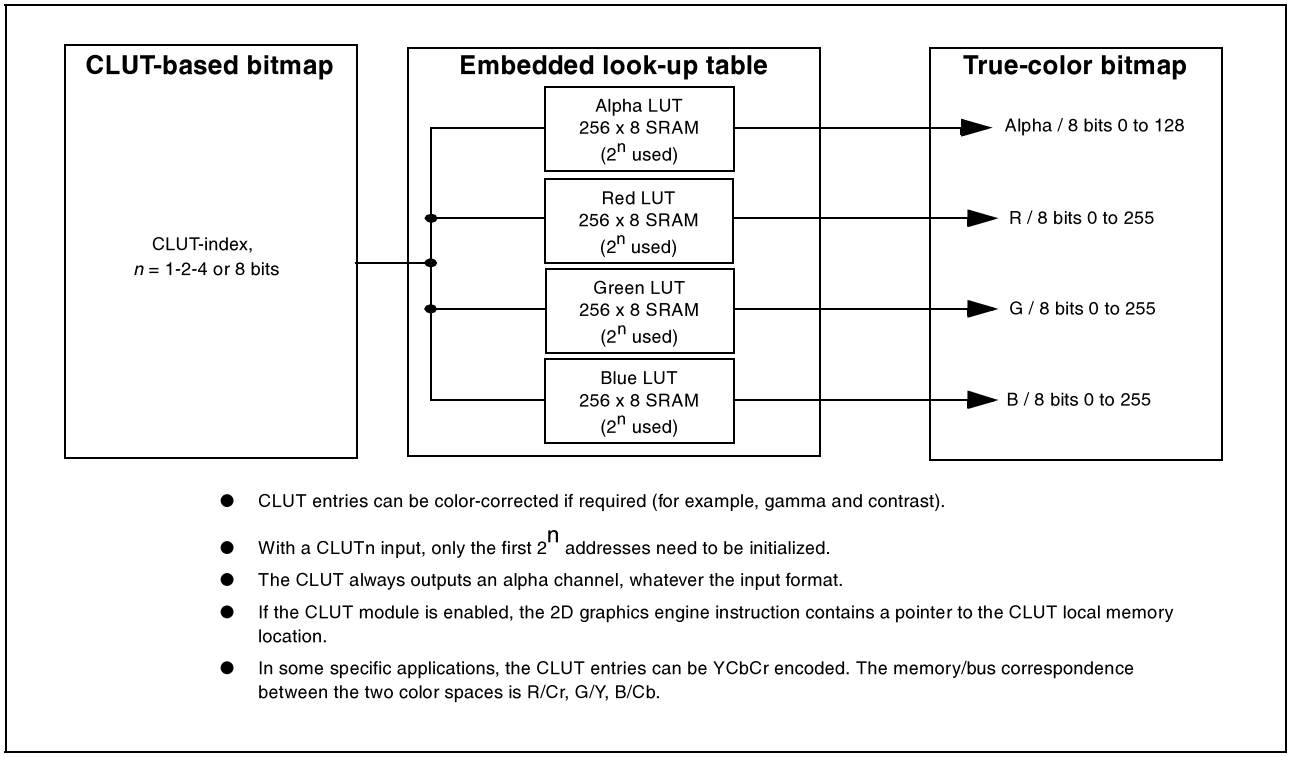
### 47.5.9 CLUT and color correction on source 2

The internal 256 x 32 LUT can perform color expansion, color reduction and color correction.

#### Color expansion

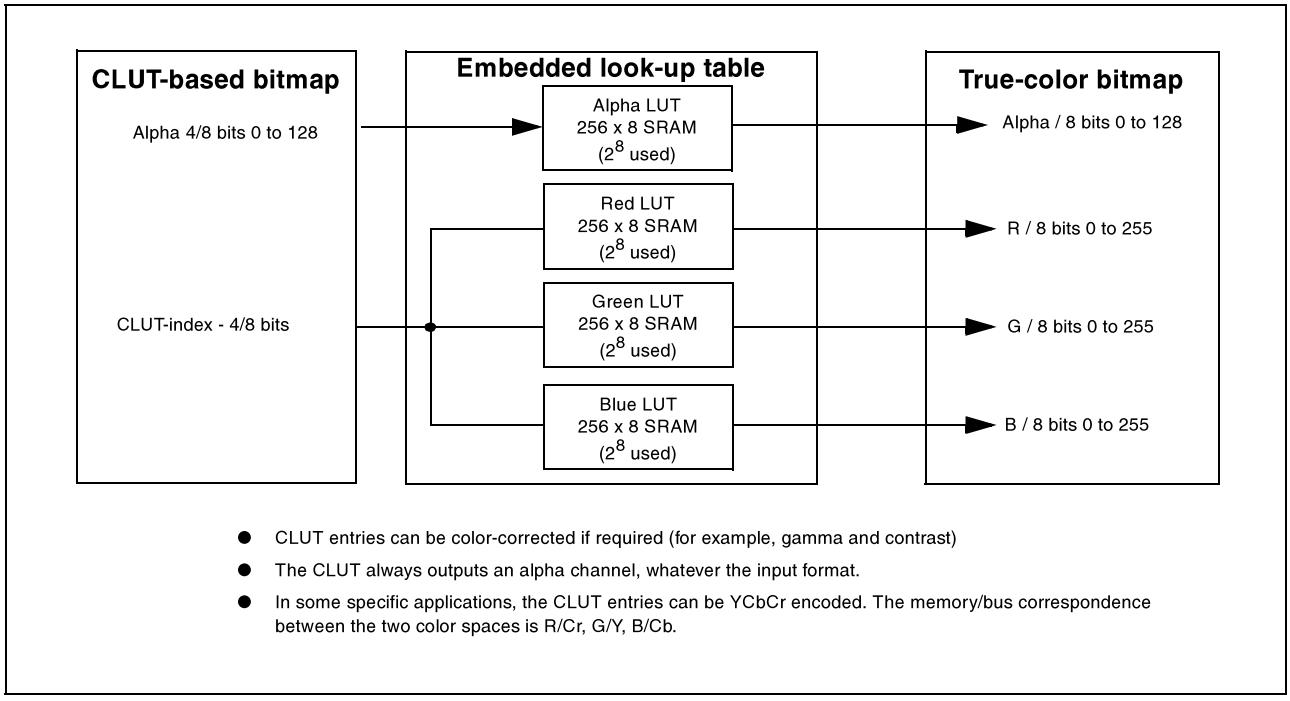
During color expansion a CLUT-based bitmap is transformed into a true-color bitmap, using the embedded look-up table (256x32 SRAM). The following two figures illustrate a CLUT-to-RGB conversion and a CLUT88/44-to-RGB conversion.

**Figure 124: ACLUTn to (A)RGB conversion**




- CLUT entries can be color-corrected if required (for example, gamma and contrast).
- With a CLUTn input, only the first $2^n$ addresses need to be initialized.
- The CLUT always outputs an alpha channel, whatever the input format.
- If the CLUT module is enabled, the 2D graphics engine instruction contains a pointer to the CLUT local memory location.
- In some specific applications, the CLUT entries can be YCbCr encoded. The memory/bus correspondence between the two color spaces is R/Cr, G/Y, B/Cb.

**Figure 125: ACLUT88/44 to ARGB conversion**




- CLUT entries can be color-corrected if required (for example, gamma and contrast)
- The CLUT always outputs an alpha channel, whatever the input format.
- In some specific applications, the CLUT entries can be YCbCr encoded. The memory/bus correspondence between the two color spaces is R/Cr, G/Y, B/Cb.

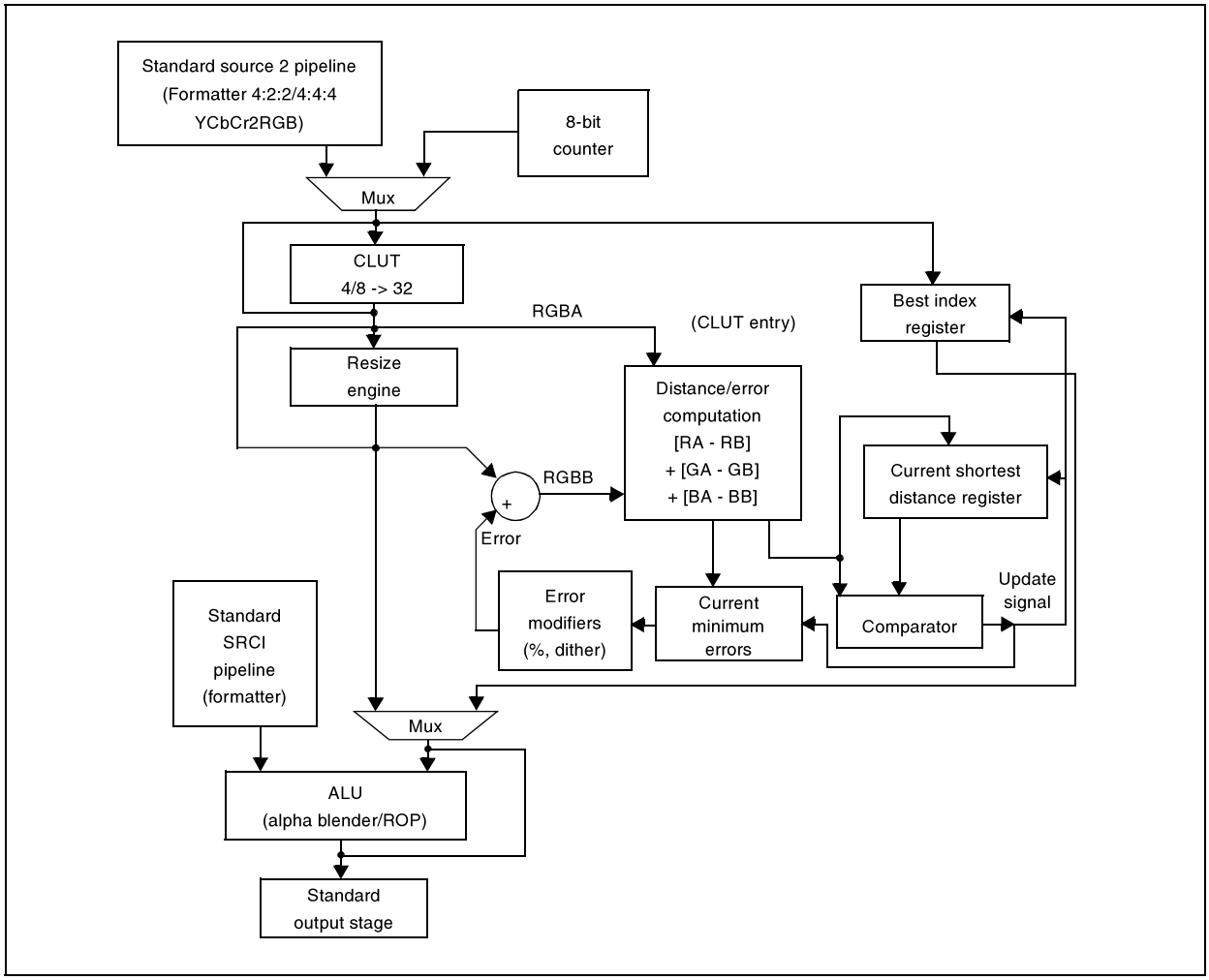
Confidential

## Color reduction

The color reduction engine performs the following tasks:

- Converts true-color RGB bitmaps into CLUT*n* bitmaps.
- Resizes a CLUT*n* bitmap.
- Converts ACLUT$n_1$ bitmaps into ACLUT$n_2$ bitmaps in two stages, as follows:
  - The ACLUT$n_1$ bitmap is expanded to an intermediate RGB bitmap (optionally resized), using its CLUT,
  - The intermediate RGB bitmap is mapped to the CLUT of the ACLUT$n_2$ bitmap.

**Figure 126: Color reduction engine architecture**




The color reduction mechanism can be summarized as follows (see also Figure 126):

- Best match strategy: the whole CLUT is scanned for each new RGB pixel from source 2. A distance is computed for each CLUT entry and the current source 2 pixel. The entry with the shortest distance is considered as the best possible match, and its index is used to represent the RGB pixel in the target CLUT-based format.
- Adaptive mode: a mono-dimensional error diffusion algorithm weighting can be added to the best match. All or part of the error made can be diffused on the next pixel (X + 1, same Y), depending on the LSBs of the address in the target bitmap (*x*LSB, *y*LSB). The distance is evaluated using the sum of the absolute values of the differences on each component:

  distance = abs[R(scr2) - R(CLUT)] + abs[G(scr2) - G(CLUT)] + abs[B(scr2) - B(CLUT)]

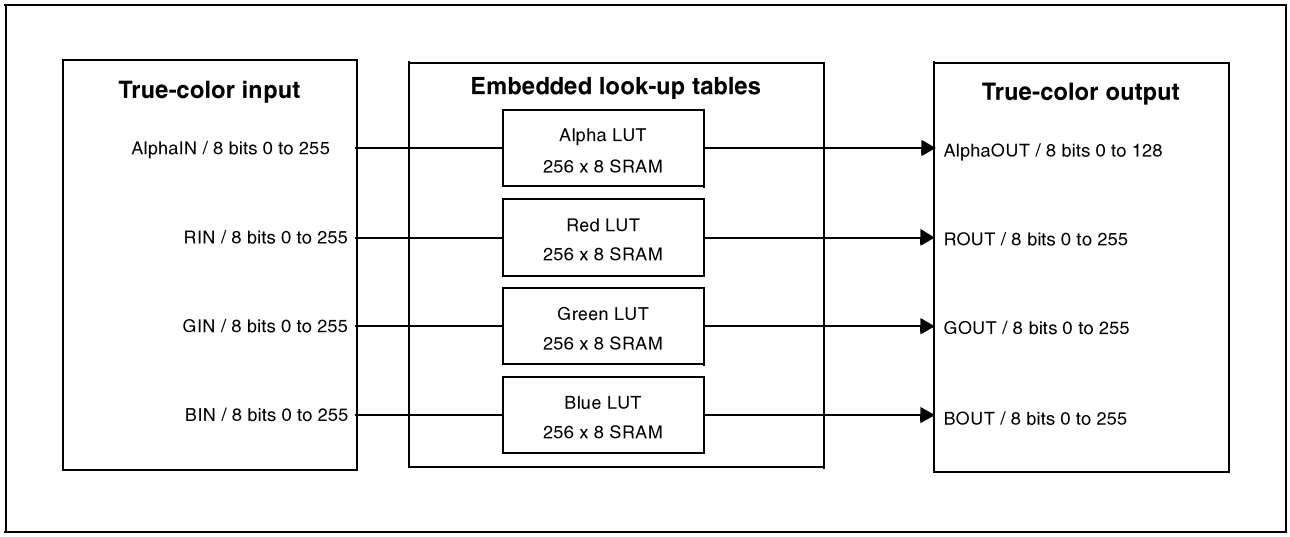
*Note: The best match search can be made in the YCbCr domain: the input color space converter must be used on source 2, and the target CLUT converted from RGB to YCbCr entries.*

Confidential

### Color correction

The CLUT can be used as four independent 256 x 8 LUTs, applying any transformation on the input components. This can be used for gamma correction, contrast adjustment, gain, offset. Used in conjunction with the 2D-graphics engine color space converters (at the input and the output), the correction can also be made in the YCbCr space (for example, color effect, conversion to a gray-scale bitmap).

**Figure 127: Color correction for true-color inputs**



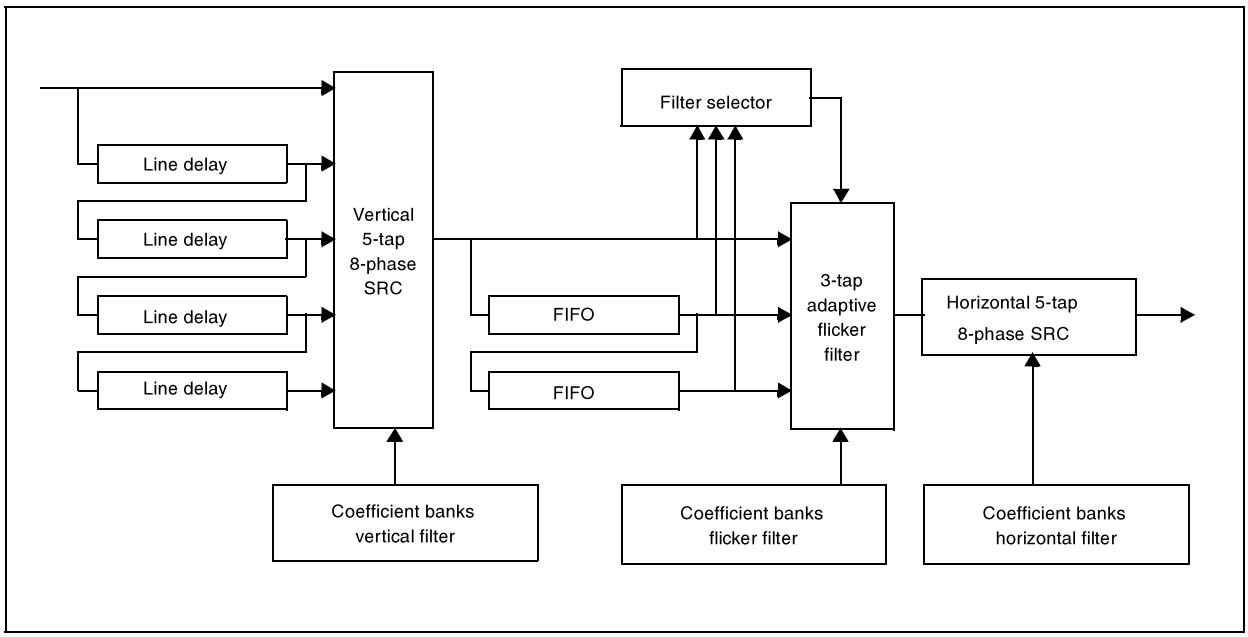

Register BLT_CML is a pointer to the local memory CLUT address.

CLUT_UPDATE enables CLUT refresh from memory, CLUT_MODE sets the CLUT mode (color expansion, color correction, color reduction), and CLUT_ERRDIFF sets the working mode of the diffusion weight for CLUT color reduction mode (register BLT_CCO).

## 47.5.10 2D resize engine

The memory-to-memory 2D resize engine includes a vertical SRC, a horizontal SRC and a context-sensitive flicker filter. It is illustrated in the figure below.

**Figure 128: 2D resize engine architecture**




Confidential

In a single pass, the 2D graphics engine can output a flicker-free downscaled/upscaled picture. The operator is neither limited by source size nor destination size. It contains sufficient memory to apply a 5 x 5 filter. The source is automatically split into vertical stripes, the maximum width of which is 128 pixels.

*Note: This is a parallel four-component operator: the alpha channel is processed in the same way as the RGB components.*

Typically, the 2D resize engine application is used to generate high-quality anti-aliased fonts: a 1 bpp source 2 bitmap of any oversampled font character is first expanded with the CLUT operator, then downsized with high-quality filtering.

There are two ways to resize CLUT-based bitmaps:

- Use a two-pass 2D graphics engine operation, as described in *Color reduction* *on page 444*.
- Use a single-pass resize engine and disable the filters. This uses a skip/repeat technique, and can be used for data that represent indexes, not true-color pixels. This method usually gives poor quality graphics.

OFFSET_HSRC (register BLT_RZC) sets the initial phase of the filter for the 1st pixel of the line.

HSRC_INC (BLT_RSF) sets the horizontal scaling factor. 2DHF_MODE (BLT_RZC) enables the horizontal filter and sets the part of the ARGB bus where it is available. Register BLT_HFP is a pointer to the filter coefficients in the local memory.

OFFSET_VSRC (BLT_RZC) sets the initial phase of the filter for the 1st pixel of the column.

VSRC_INC (BLT_RSF) sets the vertical scaling factor. 2DVF_MODE (BLT_RZC) enables the vertical filter and sets the part of ARGB bus where it is available. Register BLT_VFP is a pointer to the filter coefficients in the local memory

The CLUT pixmap may only be resized (work on index) if the horizontal and vertical filters are inactive, unless they are expanded to true color using the CLUT operator.

- The vertical and horizontal sample-rate-converter are based on a 5-tap polyphase filter (8 phases) followed by a decimator.
  - For correct filtering of the first two lines, the first source line is used three times
  - For correct filtering of the last two lines, the last source line is used three times.
  - For correct filtering of the first two pixels in a line, the first source pixel is used three times.
  - For correct filtering of the last two pixels in a line, the last source pixel is used three times.

  The initial phase can be programmed to achieve subpixel positioning (1/8 pixel).
- Coefficients:
  - Coefficient format: 8 bits S1.6 (1.0 is encoded as 64)
  - Coefficient sum = 1.0 = $2^6$ = 64
  - SRC accumulator, increment has a 6.10 format.
- Total memory: 3 Kbyte (FIFO depth can be tuned to allow maximum efficiency on burst accesses to external memory, while keeping a reasonable amount of embedded SRAM).
- Total number of U8 x S8 multipliers: approximately 52 x (5 x 4 + 5 x 4 + 3 x 4)
- All the filter coefficients can be downloaded from the instruction block (stored in external memory). Generally, they are computed depending on the rescaling factor, but the cell can be used as a general-purpose 2D filter as well (for example to produce a blurring effect).

Confidential

### 47.5.11 Flicker filter

Three vertically adjacent pixels are input to the flicker filter. The luminance steps between line *n* - 1 and line *n*, and line *n* and line *n* + 1 are estimated for each pixel.They are encoded by four bits (0 to 15). The absolute value of the maximum step is compared to three programmable threshold values, and the vertical 3 x 1 filter with the most appropriate response is selected:

| | | | |
|---|---|---|---|
| 0 | ≤step (line *n*) | ≤threshold 1 | filter 0 is selected - equivalent to no filter |
| threshold 1 | < step (line *n*) | ≤threshold 2 | filter 1 is selected - soft filter |
| threshold 2 | < step (line *n*) | ≤threshold 3 | filter 2 is selected - medium filter |
| threshold 3 | < step (line *n*) | ≤15 | filter 3 is selected - strong filter |

In test mode, the pixel color information can be replaced by a value to identify the filter (for example, filter 3 = white, filter 2 = green, filter 1 = red, filter 0 = black).

In a read/modify/write operation involving the alpha blender, the pixel alpha value can be divided by two before being used for blending on the source 2 window borders. This removes flicker on edge lines, as the three-tap vertical flicker-filter cannot operate correctly on edge lines. Horizontally, a similar mechanism is provided for the first and last pixels of each line to smooth vertical edges of the target window.

FF_MODE (BLT_RZC) selects:

- the filter 0 only mode,
- the adaptive mode, that follows an estimation of the luma difference,
- the test mode to evaluate the threshold selected in the adaptive mode.

Registers BLT_FF0, BLT_FF1, BLT_FF2, BLT_FF3 define the four filters in terms of their (*n* - 1), *n*, and (*n* + 1) coefficients, and the thresholds of the first three filters.

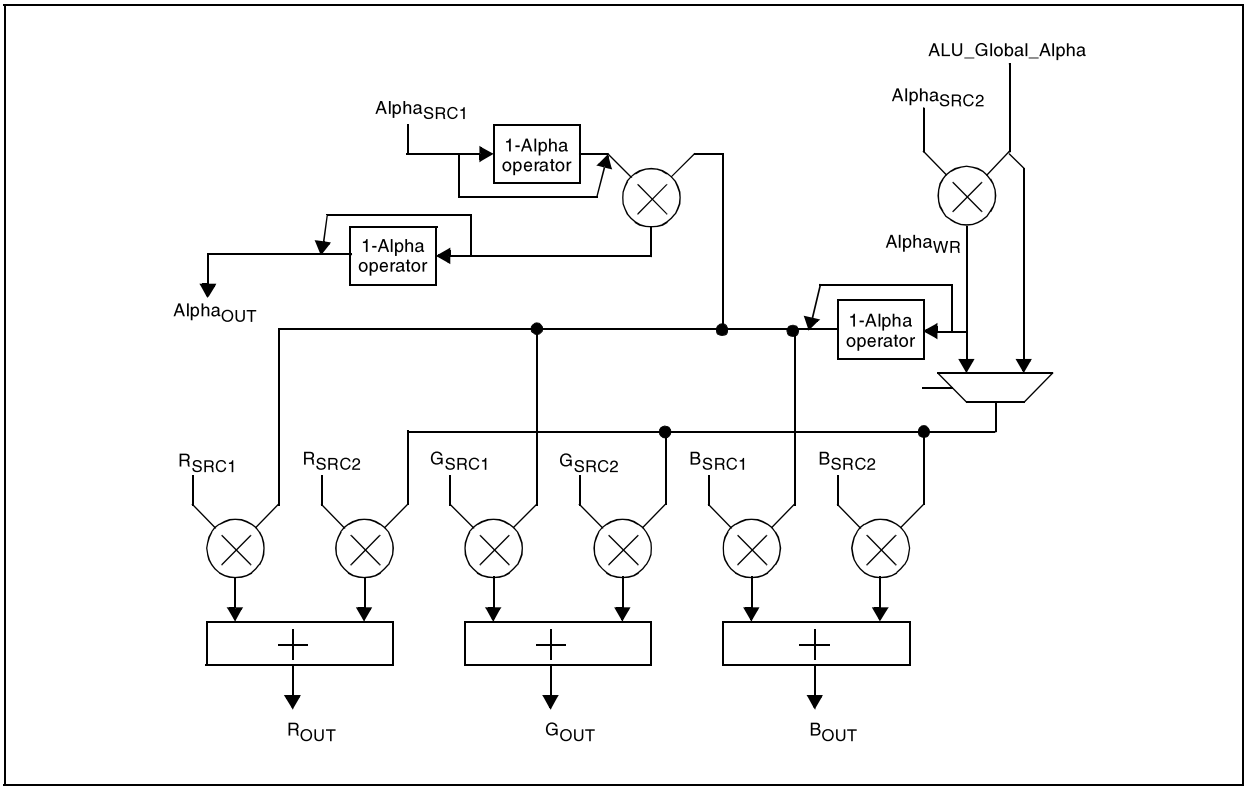
Confidential

## 47.5.12 ALU (alpha blending and boolean operator)

### Alpha blending

The alpha blender interpolates colors between source 1 and source 2 (see Figure 129).

**Figure 129: Alpha blender architecture**




Each source may have its own per-pixel alpha component. Source 2 is always blended on top of source 1; source 1 is usually the source for the background plane. Sources 1 and 2 are blended using their own alpha, combined using GALPHA_ROPID (register BLT_ACK).

Source 2 supports alpha-premultiplied and non–alpha-premultiplied color formats (premultiplied ARGB formats are in fact AR, AG and AB). The multiplexer is thus required to select between $Alpha_{WR}$ and ALU_Global_Alpha.

The blending equations are given below:

If source 2 is not pre-multiplied:

$$RGB_{OUT} = A_{SRC2} \times ALU_Global_Alpha \times RGB_{SRC2} + (1 - A_{SRC2} \times ALU_Global_Alpha) \times RGB_{SRC1}$$

If source 2 is pre-multiplied:

$$RGB_{OUT} = ALU_Global_Alpha \times RGB_{SRC2} + (1 - A_{SRC2} \times ALU_Global_Alpha) \times RGB_{SRC1}$$

In any case, the resulting translucency is:

$$\mathbf{(1 - Alpha_{OUT}) = (1 - A_{SRC2} \times ALU_Global_Alpha) \times (1 - A_{SRC1})}$$

This is equivalent to:

$$Alpha_{OUT} = A_{SRC2} \times ALU_Global_Alpha + A_{SRC1} \times (1 - A_{SRC2} \times ALU_Global_Alpha)$$

or $$Alpha_{OUT} = A_{SRC1} + A_{SRC2} \times ALU_Global_Alpha \times (1 - A_{SRC1})$$

The $Alpha_{OUT}$ component is written into the target bitmap; only the target format includes a per-pixel alpha component.

*Note: When a target format has a per-pixel alpha component, the color components $RGB_{out}$ are computed as pre-multiplied by this alpha value. The display pipeline (such as a GDP) should be aware of this when blending such a 2D graphics layer with the video layer.*

Confidential

A third source can be blended with sources 1 and 2, to create a single output. The third source must be a 1 bpp or 8 bpp bitmap mask. This three-source blending is implemented in two stages:

- The texture (or foreground picture) uses the source 1 pipeline, and the third source (bitmap mask) uses the source 2 pipeline. They are combined into an intermediate bitmap that must have a per-pixel alpha component, such as ARGB8888. In this case:

  $RGB_{OUT} = RGB_{SRC1}$

  $Alpha_{OUT} = A_{SRC1} - A_{SRC2} - ALU_Global_Alpha$

- The intermediate bitmap uses the source 2 pipeline and the background picture uses the source 1 pipeline. They are blended together.

### Boolean operators

This is a 2 operand logical unit. The operator always applies to the whole pixel width, including the alpha component if there is one.The boolean operator performs the following 16 operations.

**Table 156: Boolean operations**

| Mode name | Result | Memory cycle |
|---|---|---|
| CLEAR | *result* = all 0 | Write |
| AND | *new* AND *old* | Read/modify/write |
| ANDrev | *new* AND (NOT *old*) | Read/modify/write |
| COPY | *new* | Write |
| ANDinvert | (NOT *new*) AND *old* | Read/modify/write |
| NOOP | *old* | none |
| XOR | *new* XOR *old* | Read/modify/write |
| OR | *new* OR *old* | Read/modify/write |
| NOR | (NOT *new*) AND (NOT *old*) | Read/modify/write |
| EQUIV | (NOT *new*) XOR *old* | Read/modify/write |
| INVERT | (NOT *old*) | Read/modify/write |
| ORreverse | *new* OR (NOT *old*) | Read/modify/write |
| COPYinv | (NOT *new*) | Write |
| ORinvert | (NOT *new*) OR *old* | Read/modify/write |
| NAND | (NOT *new*) OR (NOT *old*) | Read/modify/write |
| SET | all 1 | Write |

In boolean operations, a third source can be blended with sources 1 and 2, to create a single output. The third source must be a 1 bpp bitmap mask. Combining is performed on a pixel-by-pixel basis. The raster operation is executed, depending on this 1-bit value.

Three-source boolean combining is implemented in two stages:

- The bitmap mask is combined with the texture, without destroying any existing texture per-pixel alpha. The result is stored in an intermediate bitmap, with an 8-bit per pixel fourth component (alpha or flag value).
  The texture uses source 1 pipeline, and the bitmap mask uses source 2 pipeline. If the bit value is 0, then the 8-bit fourth component is set to 255. If the bit value is 1, then the 8-bit fourth component is set to the alpha value of source 1 (128 if source 1 has no alpha channel). The color components remain unchanged.
- The intermediate bitmap uses the source 2 pipeline and the background picture uses the source 1 pipeline. These are combined. If the fourth component is 255, then the logical operation is ignored and the output directly corresponds to source 1. If the fourth component is not 255, then a standard logical operator is applied.

Confidential

Bypass and concatenation modes are available. In bypass mode, either source 1 or source 2 may be bypassed. Concatenation mode allows 4:2:0 source 1/source 2 YCbCr concatenation.

MODE (BLT_ACK) specifies 5 operation modes: single source bypass, logical operation, blending operation, clipmask mode for three-source management in two passes (two sources and one mask), and concatenation mode for macro-block formats.

- Single source bypass: bypass source 1 or source 2. For example, if source 2 is bypassed, and source 1 is set, only source 2 affects the target and takes data from the local memory.
- Logical operation: this operates on a single source (1 or 2) and on dual sources. GALPHA_ROPID (BLT_ACK) is used to program the logical operation code.
- Blending operation: this operates only on dual sources. GALPHA_ROPID (BLT_ACK) is used to program the global alpha blending value. It is also possible to activate a horizontal and/or vertical alpha border on 1 pixel width with bits AB_H and AB_V (BLT_RZC).
- Three-source clipmask management in blend or logical mode. This specific mode is used in a two-pass blitter operation to blend two sources without a uniform global alpha value or logical operator. The pixel accuracy is programmed inside a third source that plays the role of the mask. In logical mode, only the A1 format is available for the mask plane; in blending mode, A1 and A8 formats are available.
- The concatenation mode operates for YCbCr 4:2:0 macroblock formats for sources 1 and 2, when the luma from source 1 is not merged with the chroma inside source 2 (so it is usable only if LINE_RPT = 0 and COLOR_FORM = YCbCr420Mb in register BLT_S2TY).

Confidential

### 47.5.13 Color key

The 2D graphics engine has two methods of color keying: source color key and destination color key. These methods are exclusive and are both used for sprite-based animation.

- In source color keying, the source color or range of source colors is not written to the destination area. It is provided by the source 2 pipeline, before or after the CLUT.
- In destination color keying, no color can be written onto the destination color or range of destination colors. The destination color or range of colors is provided by the source 1 pipeline.

Color keying can operate on true-color bitmaps or CLUT-based bitmaps, as follows:

- For RGB true-color bitmaps:
  - A range of colors can be specified for each component, with a minimum and a maximum value.
  - The current input color is compared to the range, on a component-by-component basis.
  - Each color comparator can be enabled or disabled, and the color match result set inside or outside the range.
  - A YCbCr input can be used with the following color correspondence: G/Y, Cb/B, Cr/R.
  - A CLUT-based bitmap can be used after it is expanded to true color (after the CLUT).
- For CLUT-based bitmaps:
  - The input is taken before the CLUT, if provided by source 2.
  - When operating on an index, the color key reacts when the input matches a unique or range of index values for the blue component, specified by registers BLT_KEY1/2.

CKEY_SEL (BLT_ACK) selects the color key mode: source color key on source 2 before CLUT, source color key on source 2 after CLUT (different from the previous case only if CLUTOP in BLT_INS is enabled), destination color key on source 1 (this supposes that the target matches with source 1: read/modify/write).

ACK_CKEY (BLT_ACK) defines for each component whether the color key is inside or outside the range, or always matches.

Register BLT_KEY1 defines the minimum range for each component on 8 bits. BLT_KEY2 defines the maximum range for each component on 8 bits.

The color key match C-equation is as follows:

```
COLOR_KEY_MATCH =
   [(Rin < Rmin) OR (Rin > Rmax)] AND (Routside = True) AND (EnableR = True)
OR (Rmin <= Rin <= Rmax) AND (Routside = False) AND (EnableR = True)
OR (EnableR = FALSE)
AND
   [(Gin < Gmin) OR (Gin > Gmax)] AND (Goutside = True) AND (EnableG = True)
OR (Gmin <= Gin <= Gmax) AND (Goutside = False) AND (EnableG = True)
OR (EnableG = FALSE)
AND
   [(Bin < Bmin) OR (Bin > Bmax)] AND (Boutside = True) AND (EnableB = True)
OR (Bmin <= Bin <= Bmax) AND (Boutside = False) AND (EnableB = True)
OR (EnableB = FALSE)
```

### 47.5.14 Plane mask and output formatter on target

The output formatter converts the pixel bus into a 128-bit STBus format, according to the destination color format.

In true-color mode, a 2 x 2 dither mechanism rounds-off (for example, to target an RGB565 format with an internal 32-bit ARGB8888 pixel bus).

*Note: The sub-byte format requires the blitter to operate in read/modify/write mode.*

The plane mask is a register, whose width equals the number of bits-per-pixel in the destination format. Each bit in the target pixel word can be individually protected.

- If bit *n* in the plane mask register is 1, the corresponding *n*-weight bit in source 1 is updated according to the current 2D graphics engine operation.
- If bit *n* in the plane mask register is 0, the corresponding *n*-weight bit in source 1 remains unchanged whatever the 2D-graphics engine operator applied.

Register BLT_PMK provides the write-protect mask. BLT_TBA is the memory base address of the target bitmap (absolute (0,0) location, top-left pixel).

BLT_TTY is the specific properties of source 2: format, pitch, horizontal and vertical scan order, sub-byte ordering (for sub-byte formats), bit accuracy reduction mode. BLT_TXY is the coordinate that specifies the start of the input source 2 window with respect to the base address.

Register BLT_S1SZ is the size of the target window (equals source 1 size).

All formats are available except YCbCR420MB and YCbCR422MB.

Confidential

## 47.6 Using the 4:2:x macroblock-based plane as a source

The 4:2:x plane can be used as a 2D graphics engine source. Because the 4:2:x plane is built from two buffers (one for luma and one for chroma), source 1 and source 2 input buffers must both be used to retrieve one 4:2:x plane. Consequently, 2D graphics engine operations with a 4:2:x input must be a one-source blitter. Blending or boolean combinations, involving a 4:2:x macroblock source, require a two-pass blitter operation.

The following two methods use the 4:2:x plane as a source:

- 4:2:0 chroma vertical repeat mode (BLT_S2TY.LINE_RPT) or 4:2:2 macroblock formats:
  For 4:2:0, the 2D graphics engine duplicates the chroma lines to generate a 4:2:2 stream. Source 1 is used for the luma component, source 2 for chroma components. All the operators except ALU can be used.
  For 4:2:2/4:2:0 macroblock conversion, if using the resize operator on the entire 4:2:x picture:
  LINE_RPT (source 2 chroma line repeat) = enable for 4:2:0 macroblock format,
  BLT_S2SZ.SRC2_WIN_HEIGHT = luma line height,
  BLT_S2SZ.SRC2_WIN_WIDTH = luma pixel width,
  BLT_S1SZ.WIN_HEIGHT = new line height,
  BLT_S1SZ.WIN_WIDTH = new pixel width,
  BLT_ACK.MODE = bypass source 2,
  BLT_INS.2DRESCALE = enabled, if desired,
  BLT_RZC = required parameters,
  BLT_RSF = required parameters.
- Chroma vertical interpolation mode:
  Source 1 is used for luma, source 2 for chroma. Chroma information is 2D resized to upscale these components by two and add a half-line vertical translation correction. The ALU is configured to concatenate the unchanged luma, with the upscaled chroma components, to build 4:4:4 YCbCr pixels. These pixels can be optionally converted to RGB or YCbCr 4:2:2 raster, in the last 2D graphics engine processing stages.
  For 4:2:0 macroblock conversion using the resize operator for chroma upsampling:
  BLT_S2SZ.SRC2_WIN_HEIGHT = chroma line height = luma line height / 2,
  BLT_S2SZ.SRC2_WIN_WIDTH = luma pixel width,
  BLT_S1SZ.WIN_HEIGHT = luma line height,
  BLT_S1SZ.WIN_WIDTH = luma pixel width,
  BLT_ACK.MODE = logical operation OR or 4:2:0 source1/ source 2 YCbCr concatenation,
  BLT_INS.2DRESCALE = enable,
  BLT_RZC.2DHF_MODE = disable,
  BLT_RZC.2DVF_MODE = enable on color channel,
  BLT_RZC.VSRC_OFFSET = 0,
  BLT_RSF.VSRC_INC = 512.

A macro-block frame buffer may be accessed in frame mode, or in field mode, depending on bits BLT_S1TY.MB_FIELD and BLT_S2TY.MB_FIELD. In frame mode, all source lines are read, in field mode, the blitter reads every other line. This is very useful for handling interlaced video, in a blitter-based PIP application for instance.

Confidential

## 47.7 Blitter XYL mode

### 47.7.1 XYL standard mode

Instead of scanning a rectangular area to generate XY addresses, the blitter can directly access random XY addresses, pre-computed by the CPU, and stored linearly in a memory buffer. Typical use: shapes (circles, ellipses), fonts, polygon lines and fill, run-length decoder.

The address of this buffer is given in BLT_XYP. The buffer contains a list of subinstructions. A subinstruction consists either of a single pixel processing, or of a horizontal line processing. For a given blitter XYL operation (that is, corresponding to a single blitter node), all the subinstructions must be of the same type.

Four subinstruction formats are supported in XYL standard mode:

- XY: single pixel access. Only XY coordinates are provided, the drawing color is given by the drawing context (color fill register BLT_S2CF).
- XYC: single pixel access. XY coordinates are provided, as well as the drawing color C that may change for each pixel (including the alpha component).
- XYL: horizontal line access. XY coordinates correspond the left pixel of the line, the L parameter provides the line length, in pixel unit.The drawing color is given by the drawing context (color fill register BLT_S2CF).
- XYLC: all the parameters are specified for any subinstruction, XY coordinates, line length and drawing color.

Whatever the subinstruction format, a subinstruction is always stored as 4-consecutive 32-bit words in memory:

**Table 157: XYL subinstruction formats in memory**

| | XY | XYC | XYL | XYLC |
|---|---|---|---|---|
| **1st 32-bit word** | X | X | X | X |
| **2nd 32-bit word** | Y | Y | Y | Y |
| **3rd 32-bit word** | unused | unused | L | L |
| **4th 32-bit word** | unused | C | unused | C |

The number of subinstructions involved in a blitter XYL instruction is provided in register BLT_XYL. An XYL operation can be considered as a normal blitter operation, except that instead of self-addressing rectangular areas, the blitter reads the pixel addresses in memory, in what is called the subinstructions buffer.

In standard XYL mode, the blitter performs a write-only access at the specified XY location in the target bitmap if the ALU is in bypass source 2 operating mode. Source 1 must be disabled.

The blitter performs read/modify/write, if source 1 is enabled and a combining operation (blending or logical boolean) has been programmed in the ALU. In this case, source 1 and the target must have the same bitmap context. The blitter first reads the current pixel at the XY location, using the source 1 resources for memory access. It then combines this pixel with the source 2 color fill register (BLT_S2CF), or the color provided in the subinstruction if XYC/XYLC. The result is written back to the same memory location.

When XYL or XYLC subinstructions are selected, this sequence reiterates for each pixel of the horizontal line (pipeline implementation).

The XYL blitter feature supports any color format with a 4:4:4 sampling structure; 4:2:2 and 4:2:0 modes are not supported.

The color (register BLT_S2CF or C field in the XYC or XYLC subinstruction) must be formatted (that is, programmed as internally represented at the source 2 formatter output, see Section 47.5.1 on page 433).

Confidential

For example, in RGB565 mode, to write 0xABCD (R = 21, G = 30, B = 11), BLT_S2CF must be programmed with 0x80A8 7858 (an opaque alpha channel added plus expansion to 8 bits per component).

Table 158 shows the features available when working in XYL mode.

**Table 158: Blitter XYL capabilities**

| Blitter feature | XYL mode availability | Comment |
|---|---|---|
| Color fill | Yes | Source 2 color fill only, with XY and XYL subinstruction format |
| 4:2:X to 4:4:4 | No | No meaning |
| 4:4:4 to 4:2:X | No | No meaning |
| Programmable H/V scan | No | No meaning |
| Color space conversion | Yes | Output color space converter only |
| CLUT operator | No | |
| 2D-resize | No | |
| Flicker reduction | No | |
| Color keying | Yes | |
| Rectangular clipping | Yes | |
| Plane-mask | Yes | |
| Clipmask | Yes | See description below |

Confidential

### Clipmask feature when working in XYL mode:

The clipmask feature, when working in XYL mode, requires source 2 memory access to access the clipmask data. That means the whole source 2 address generator must be programmed to access this 1-bpp or 8-bpp data, and each register takes the following meaning:

- BLT_S2TY: clipmask bitmap parameters (such as pitch, format).
- BLT_S2BA: memory address for the (0,0) clipmask value.
- BLT_S2XY: clipmask positioning, that is, XY location of the (0,0) clipmask origin, with respect to the target (0,0) origin.
- BLT_S2SZ: unused.

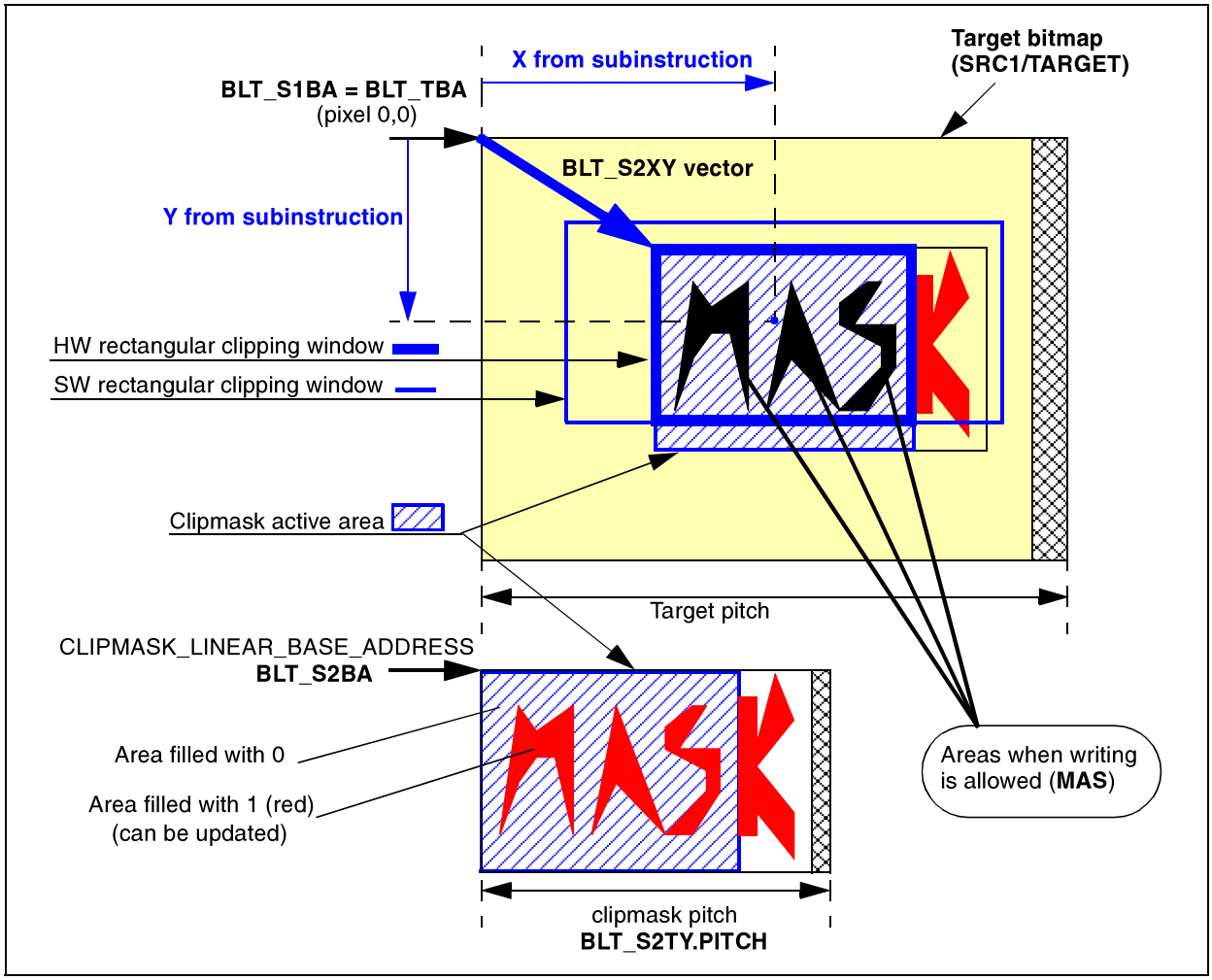
The ALU must be programmed for clipmask in XYL mode with either a logical operation, or blending with source 2 not premultiplied, or blending with source 2 premultiplied (MODE in register BLT_ACK).

In a MediaHighway drawing context, a clipmask validity window is defined, as well as a standard rectangular clipping window. For the blitter XYL to be compliant, the hardware rectangular clipping window must be programmed (BLT_CWO and BLT_CWS), with the intersection of the two software drawing context windows. Outside this resulting window, no write is performed.

The following hardware sequence occurs:

- Read the current target pixel at the (X,Y) location, in the target color bitmap, using source 1 address generator.
- Read the corresponding clipmask data XCOORD and YCOORD (BLT_S2XY) in the clipmask bitmap, using source 2 address generator.
- Combine the source 1 pixel with the BLT_S2CF color in the ALU, according to the ALU operation code, and depending on the current clipmask data. See Section 47.5.12 on page 448 for details about ALU behavior with respect to clipmask data (clipmask is equivalent to bitmap mask).
- If this x, y location is valid (inside the clipping window), then write the resulting pixel at the (X, Y) location, in the target color bitmap, using the target address generator.

Figure 130: Example of an XY access with clipmask (logical ROP)




Confidential

## 47.7.2 How to use XYL mode

Set bit XYL (register **BLT_INS**) to program the blitter in XY mode.

The byte memory location for the XYL subinstruction buffer is set in register BLT_XYP. The XYL subinstruction size is always 128 bits, whatever the format of these subinstructions: XY, XYL, XYC, or XYLC. The subinstruction format is set using SUBINS (BLT_XYL), and their number is given by NUM_SUBINS.

The different ALU operations set by MODE (BLT_ACK) can then be performed. Not all ALU operations are allowed in XYL mode. The following ones are allowed and are described below:

0001: logical operation

0010: Blending mode, source 2 unweighted

0011: Blending mode, source 2 weighted

0111: bypass source 2

1001: Clipmask in XY mode, logical op

1010: Clipmask in XY mode, source 2 unweighted

1011: Clipmask in XY mode, source 2 weighted

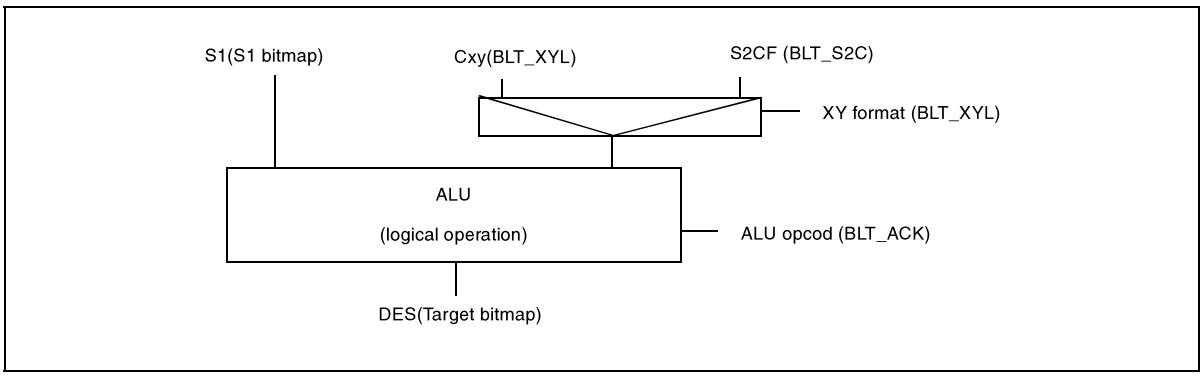
### 0001, 0010, 0011: logical operation, and blend

These operations are performed between two sources. Source 1 accesses the destination area using BLT_S1BA and BLT_S1TY.

If a color is defined inside the XY subinstruction (XYuC or XYLC), this color is used. Otherwise the color set in BLT_S2CF is used (source 2 color fill register). The ALU performs the operation between source 1, unmodified, and the static color. Source 2 weight is handled using the normal ALU datapath. The resulting composition is sent back to memory at its initial position using BLT_TBA and BLT_TTY, which are equal to BLT_S1BA and BLT_S1TY.

If these two registers (BLT_TBA and BLT_TTY) are not set as equal to their S1 equivalent (BLT_S1BA and BLT_S1TY), the XYL result can be stored in another memory destination.

**Figure 131: Datapath in ROP or blend (weighted/unweighted)**




### 0111: bypass source 2

In this mode, source 1 is not accessed, as no operation is performed with the background. It only writes the new color at the destination location. This is the only mode where read/modify/write are not performed for data in the memory. Only a write is performed.

### 1001: clipmask pass in XY mode (ROP)

The clipmask is accessed through the source 2 interface. It means source 2 has to be fully set to access the clipmask bitmap (BLT_S2BA, BLT_S2TY and BLT_S2SZ).

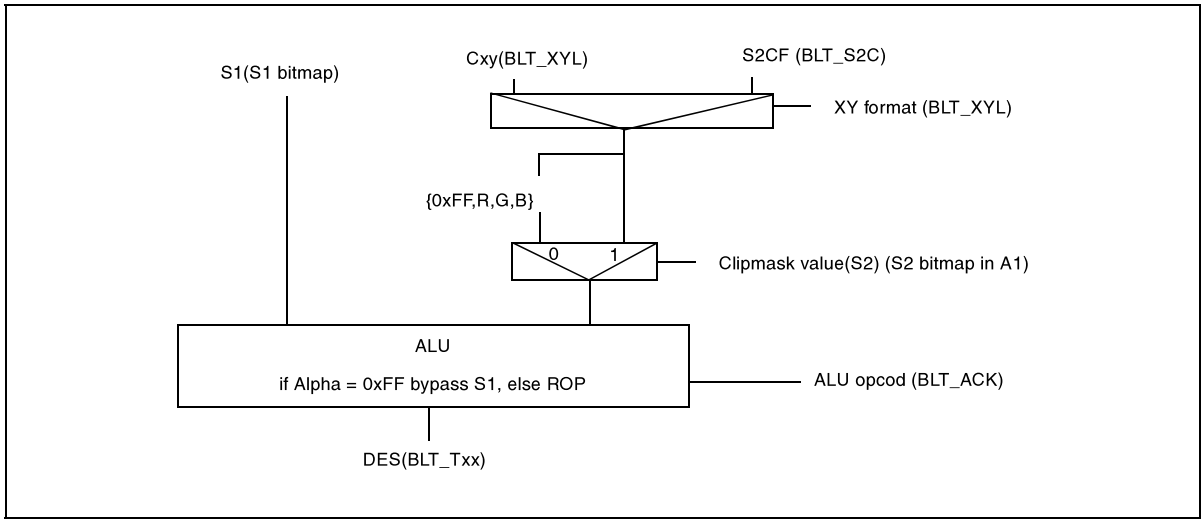
BLT_S2CF remains the second operand for ROP if no color is defined in the XYLC subinstruction.

The source 2 bitmap is used as clipmask. In ROP (logical operation) mode, the clipmask (source 2 bitmap) can be used only in A1 format. When A1 = 0, the background is passed. This means that the source 2 alpha is replaced by 0xFF, so that the existing datapath (Aa in clipmask pass 2) is not modified.

When A1 = 1 the operation is executed normally.

Confidential

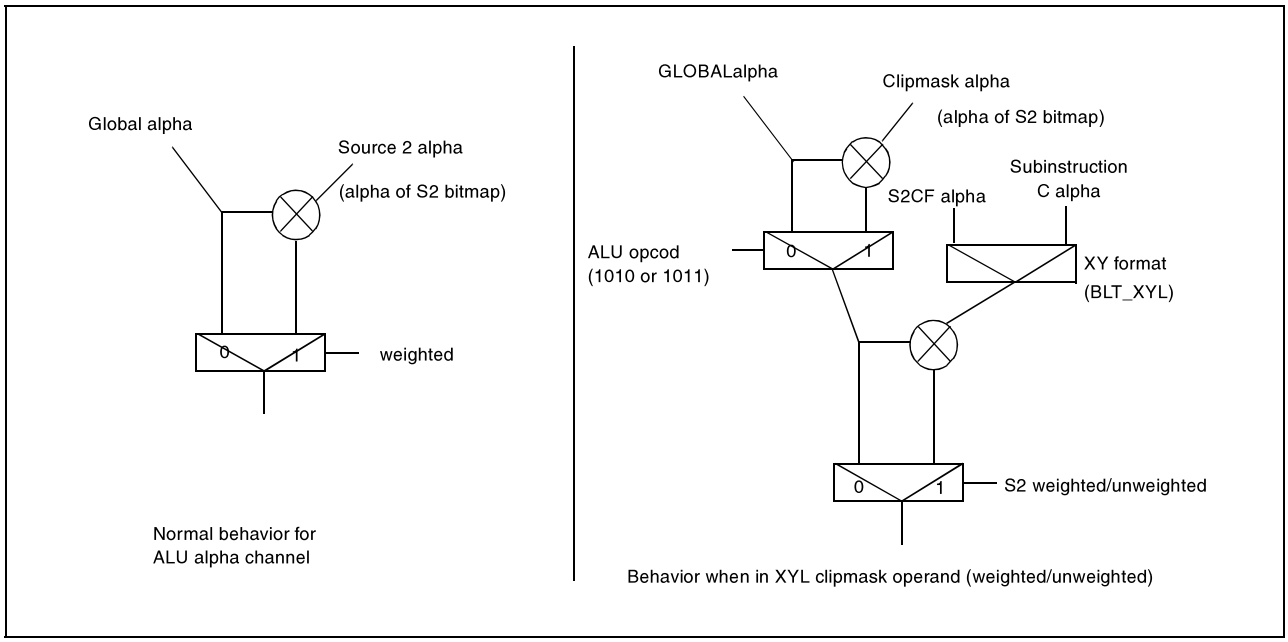
**Figure 132: Clipmask ROP in XYL format**




### 1010/1011: Clipmask pass in XY mode (blend, source 2 unweighted/weighted)

The clipmask is again accessed through the source 2 interface, as in the previous mode, but this time it can be either A1 or A8 format. Only the alpha channel is modified, and the computing of alpha changes as shown in Figure 133.

**Figure 133: Destination alpha computation**



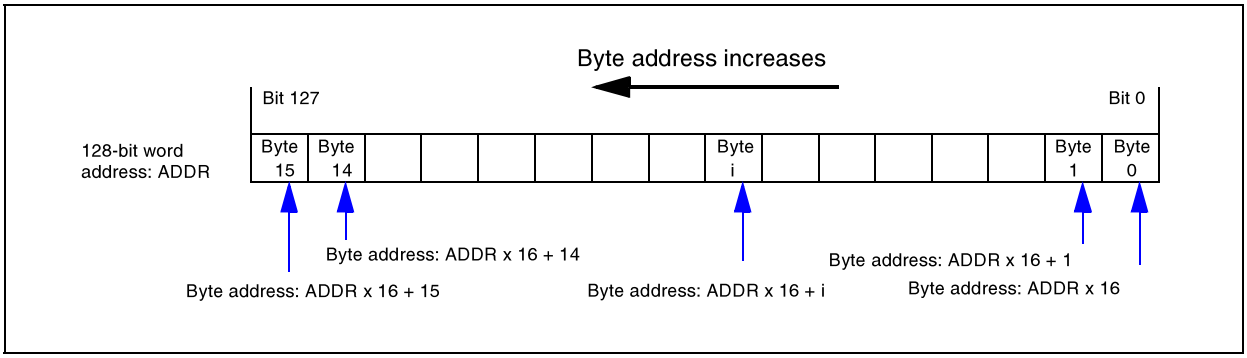

Confidential

## 47.8 Local memory storage of supported graphics formats

### 47.8.1 General aspects

Memory storage (128-bit word) for all graphics data is little endian.

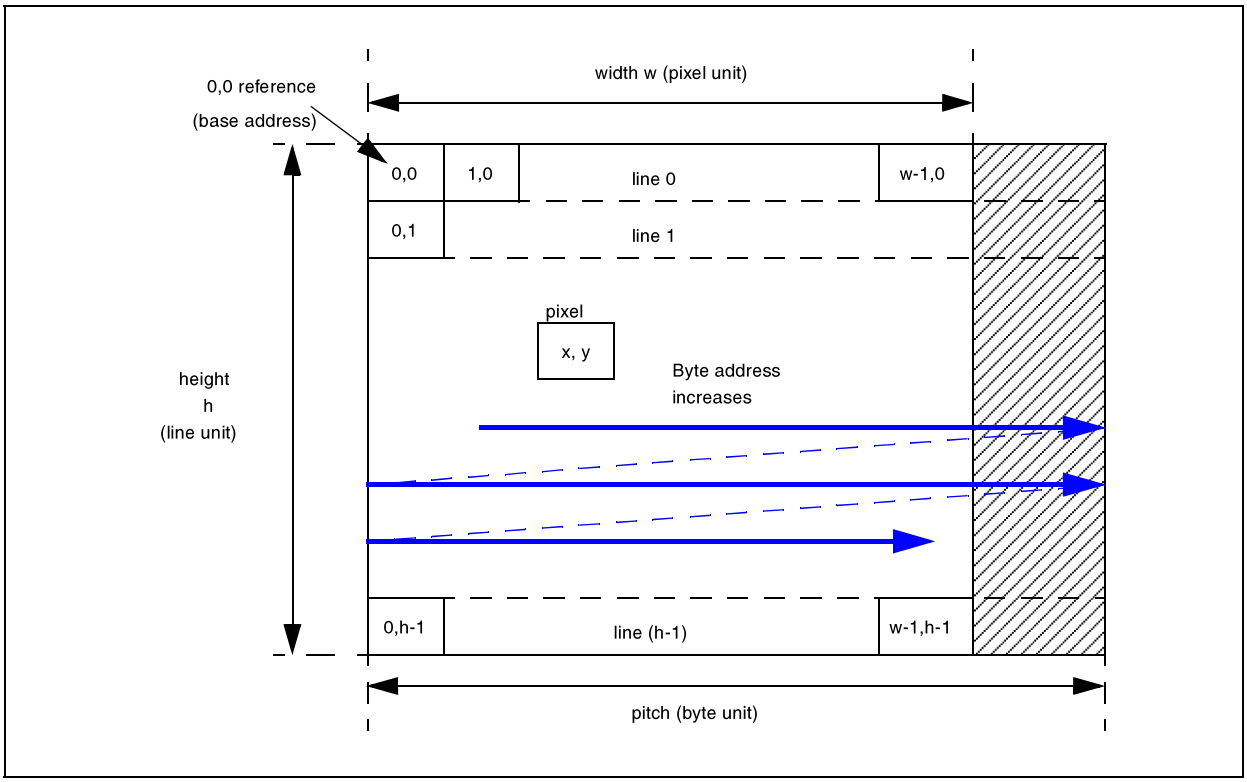
**Figure 134: Little endian convention for a 128-bit word**




For any graphics format (except 4:2:0 planes), a bitmap can always be considered as a rectangular area, with a width in pixel units, and a height in line units.

The (0, 0) origin is always the upper-left corner of the bitmap. Any pixel can be internally addressed using an (x, y) model, with the (0, 0) reference pixel being the top-left location of the rectangular area.

**Figure 135: 2D-graphics frame-based raster storage**




When stored in the local memory, pixel (0,0) is stored at the lowest byte address. The scan order is from left to right horizontally, and from top to bottom vertically. For each line (constant y coordinate), the pixel address increases when the x coordinate increments.

The pitch is the distance in bytes between any pair of vertically adjacent pixels (same x coordinate, y/y+1). The pitch is at least the width multiplied by the number of bytes per pixel, but

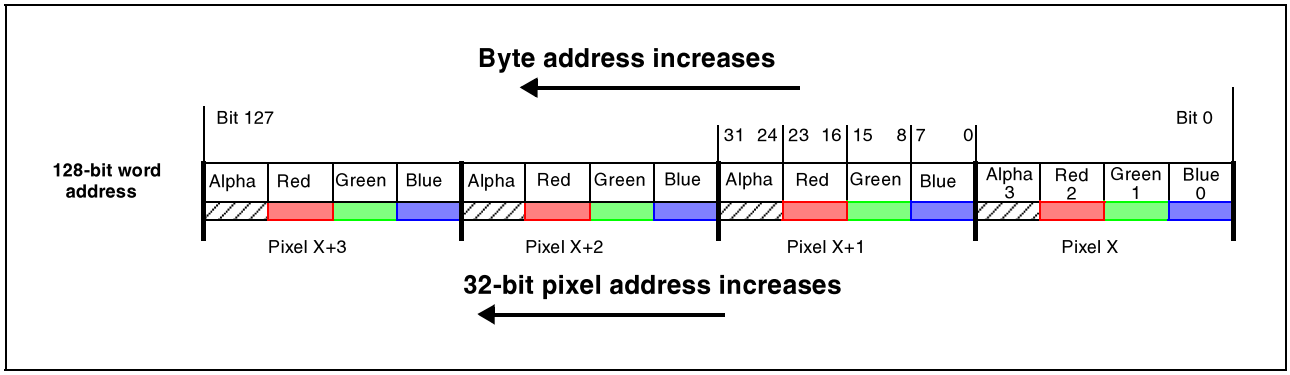
Confidential

can be greater (for instance, byte stuffing for memory alignment considerations, or global bitmaps containing elementary bitmaps, side-by-side).

The specific organization within a 128-bit word is described below for each format.

### 47.8.2 32-bit per pixel format - true color

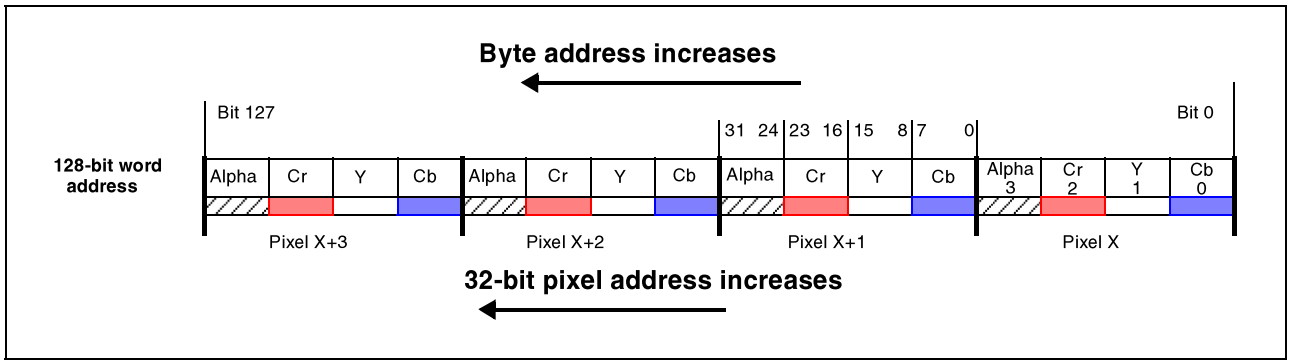
**Figure 136: ARGB8888 alignment in a 128-bit word**




*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

**Figure 137: AYCbCr8888 alignment in a 128-bit word**



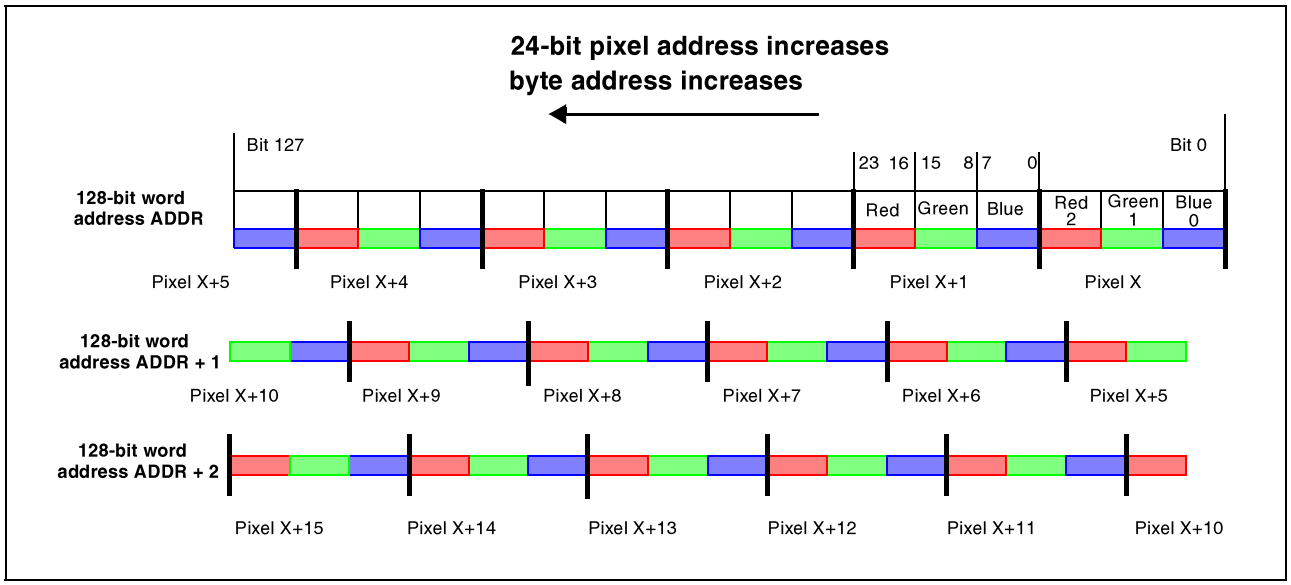

*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

Confidential

### 47.8.3 24-bit per pixel format - true color

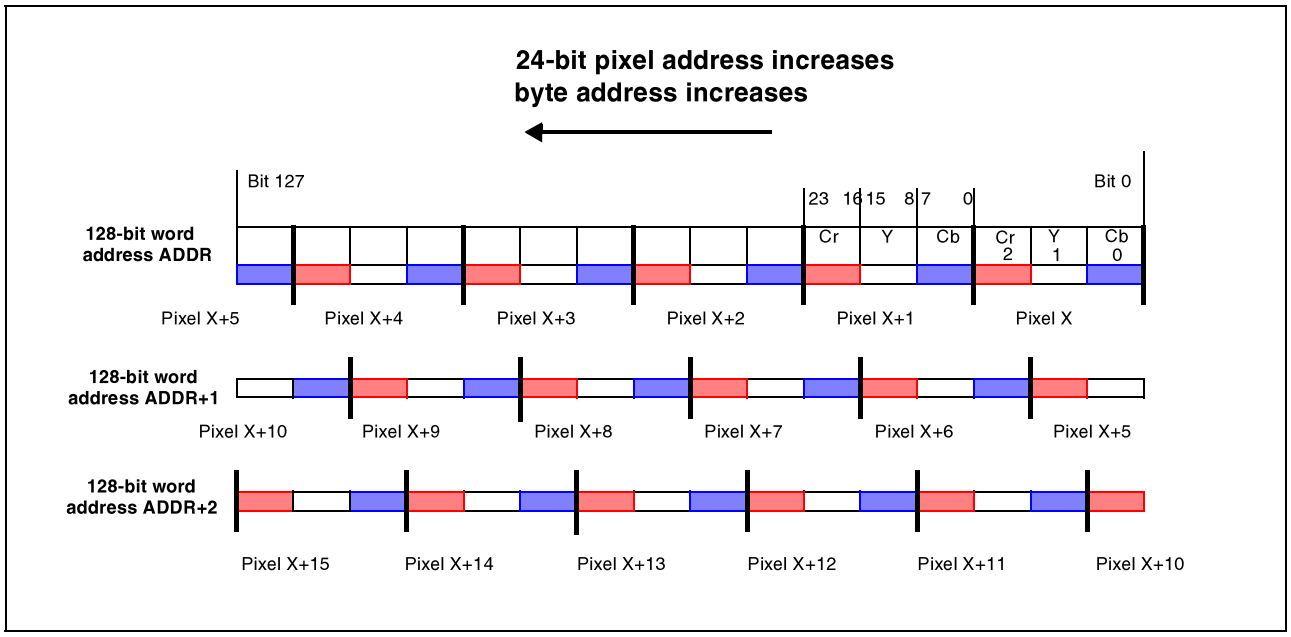
**Figure 138: RGB888 alignment in a 128-bit word (3-word period/16-pixel period)**




*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

**Figure 139: YCbCr888 alignment in a 128-bit word (3-word period/16-pixel period)**

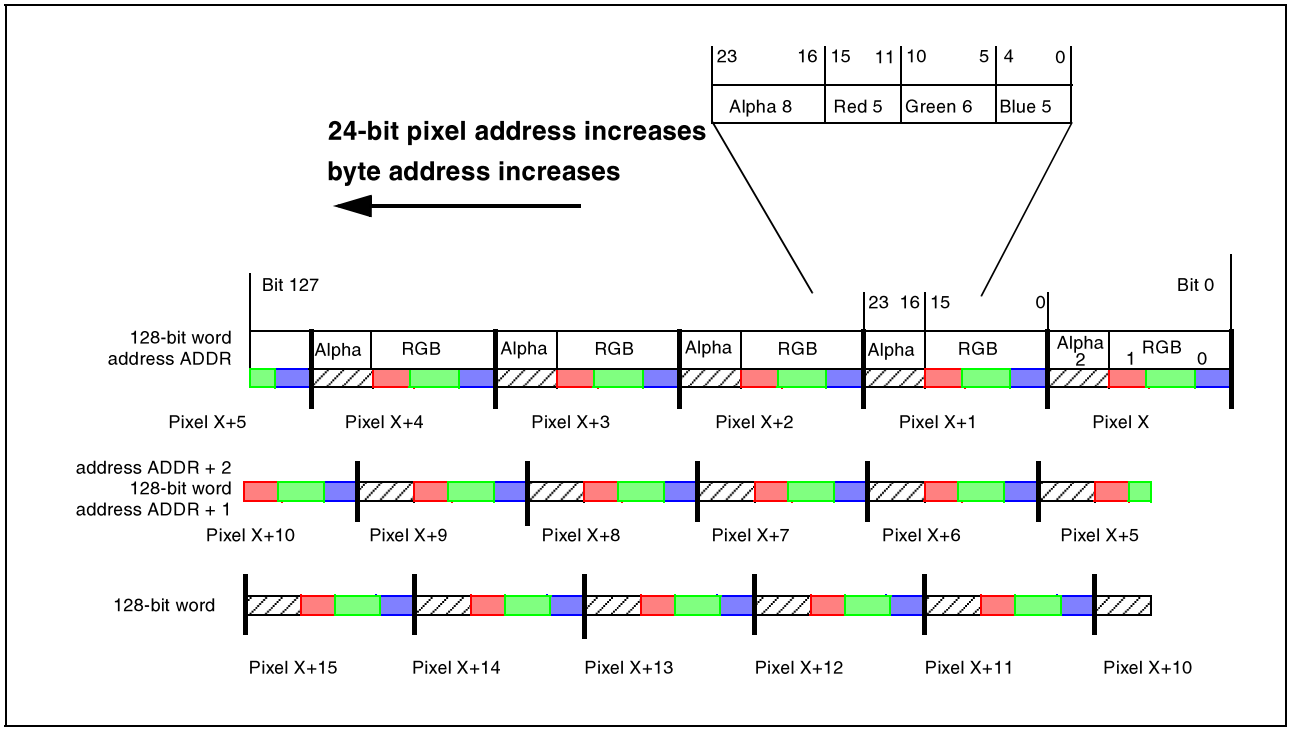



*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

Confidential

**Figure 140: ARGB8565 alignment in a 128-bit word (3-word period/16-pixel period)**

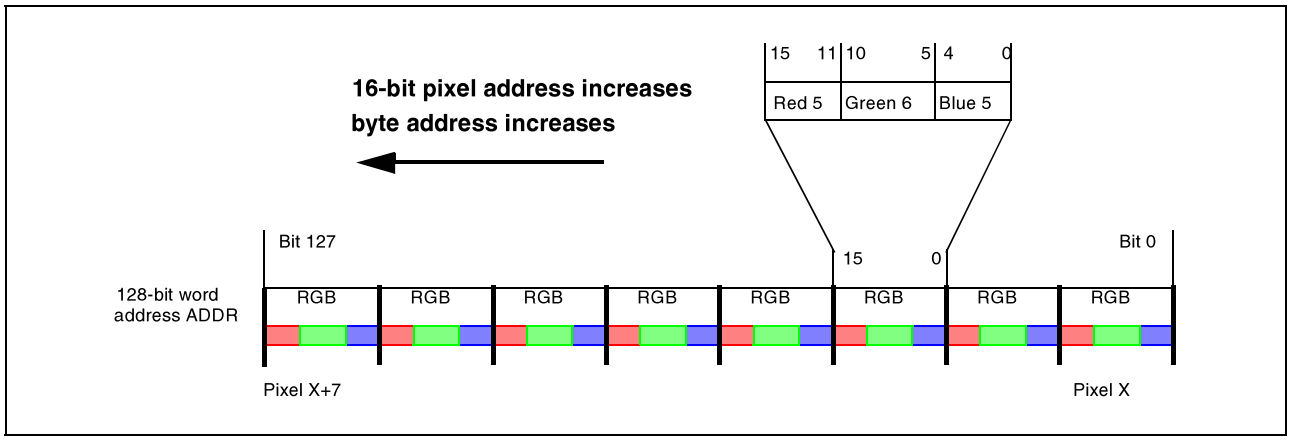



*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

### 47.8.4 16-bit per pixel format - true color

**Figure 141: RGB565 alignment in a 128-bit word**

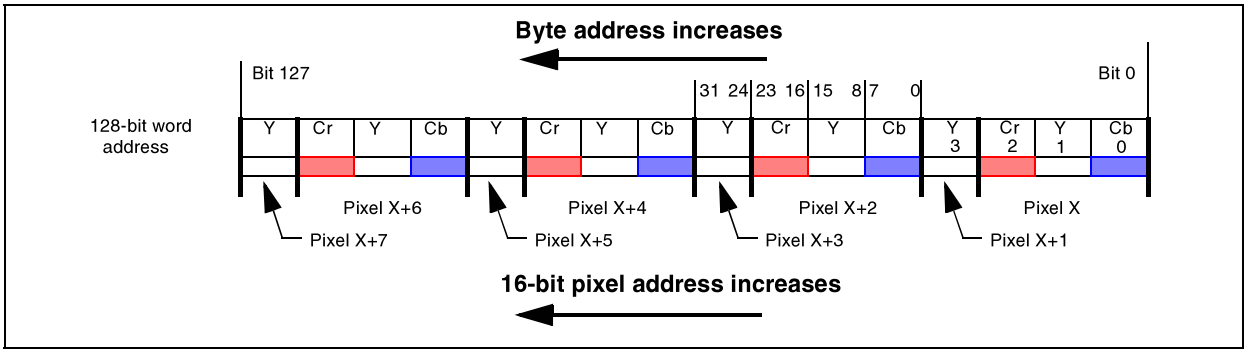



*Note: 1 The memory address for pixel (0,0) must be aligned on a 16-bit word address boundary.*

*2 The pitch value must be a multiple of two bytes.*

Confidential

**Figure 142: YCbCr 4:2:2 (16 bpp)**

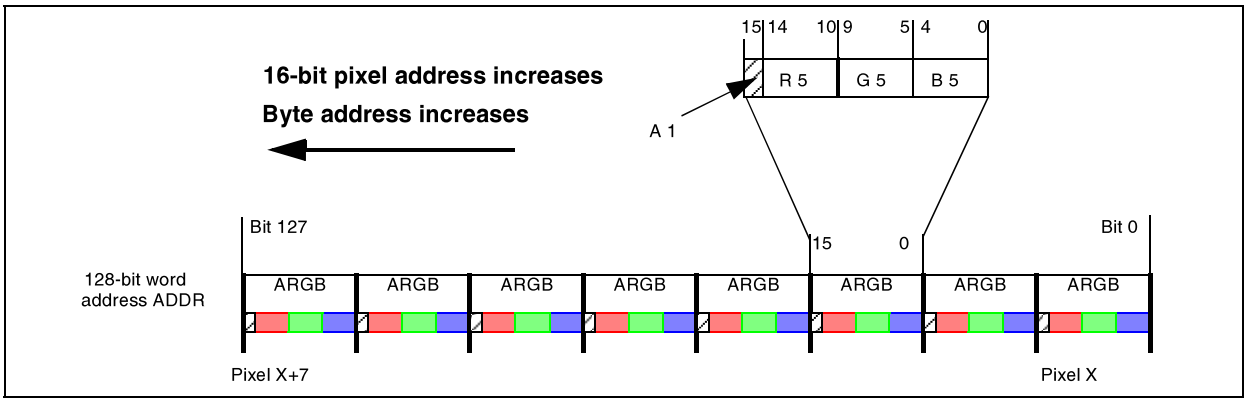



*Note: 1 The memory address for pixel (0,0) must be aligned on a 32-bit word address boundary.*

*2 The pitch value must be a multiple of four bytes.*

*3 X is even on this diagram.*

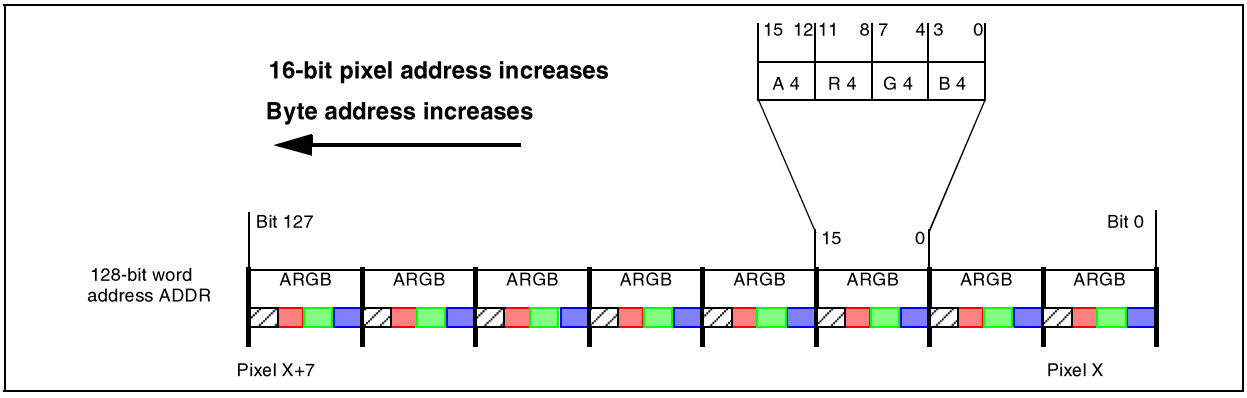
**Figure 143: ARGB1555 alignment in a 128-bit word**




*Note: 1 The memory address for pixel (0,0) must be aligned on a 16-bit word address boundary.*

*2 The pitch value must be a multiple of two bytes.*

**Figure 144: ARGB4444 alignment in a 128-bit word**



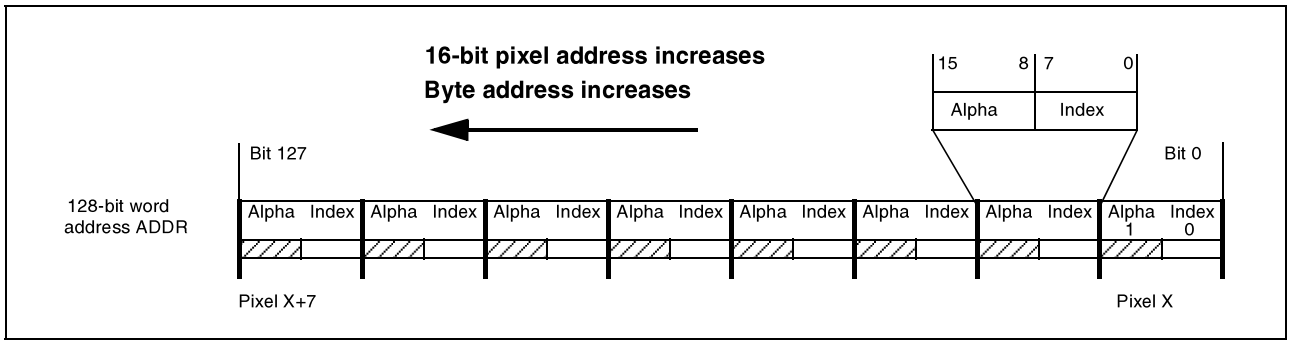

*Note: 1 The memory address for pixel (0,0) must be aligned on a 16-bit word address boundary.*

*2 The pitch value must be a multiple of two bytes.*

Confidential

### 47.8.5 16-bit per pixel format - CLUT based

**Figure 145: ACLUT88 alignment in a 128-bit word**

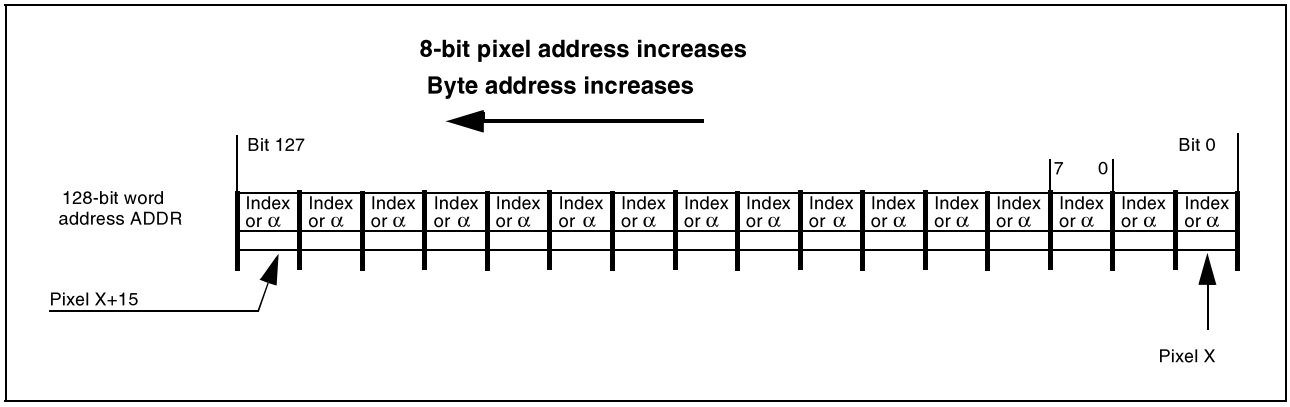



*Note: 1 The memory address for pixel (0,0) must be aligned on a 16-bit word address boundary.*

*2 The pitch value must be a multiple of two bytes.*

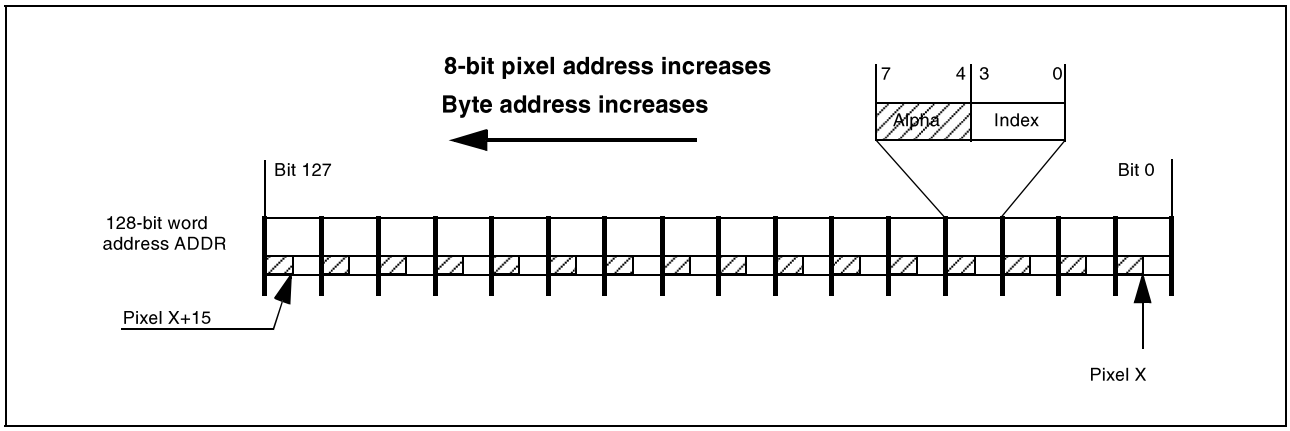
### 47.8.6 8-bit per pixel format

**Figure 146: CLUT8 / A8 alignment in a 128-bit word**




*Note: The memory address for pixel (0,0) must be byte-aligned.*

**Figure 147: ACLUT44 alignment in a 128-bit word**

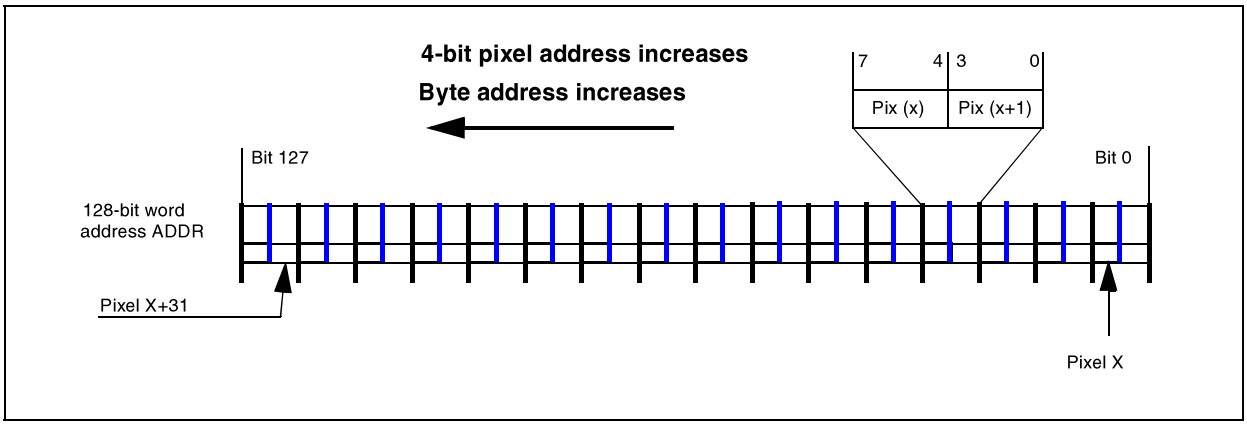



*Note: The memory address for pixel (0,0) must be byte-aligned.*

Confidential

### 47.8.7 1/2/4 bpp format

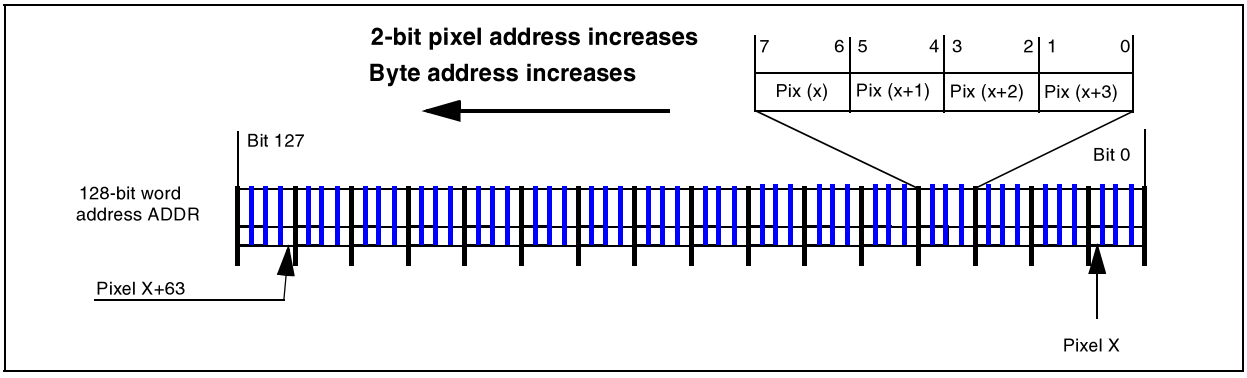
**Figure 148: ACLUT4 alignment in a 128-bit word**




*Note: 1 The memory address for pixel (0,0) must be byte-aligned.*

*2 The pitch is specified in bytes.*

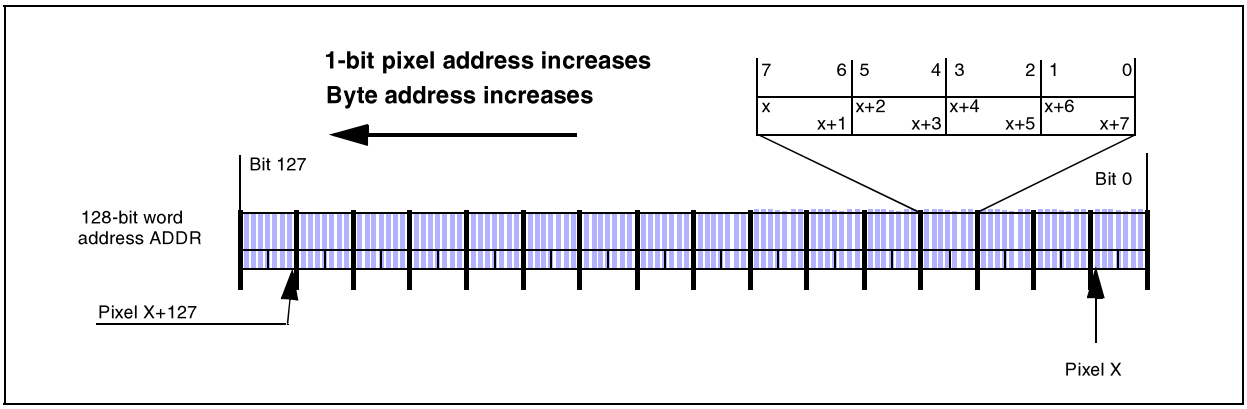
**Figure 149: CLUT2 alignment in a 128-bit word**




*Note: 1 The memory address for pixel (0,0) must be byte-aligned.*

*2 The pitch is specified in bytes.*

**Figure 150: ACLUT1 / A1 alignment in a 128-bit word**




*Note: 1 The memory address for pixel (0,0) must be byte-aligned.*

*2 The pitch is specified in bytes.*

*3 The pixel ordering within a byte, as shown in the previous three figures, is the one supported by the display pipelines. The blitter can support either the pixel ordering as shown, or the reverse order, so that the cell can handle any organization.*

Confidential

# 48 2D graphics processor registers

Blitter register addresses are provided as
*BlitterBaseAddress* + *BlitterRegsBaseAddress* + offset.

Blitter 2D coefficient addresses are provided as
*BlitterBaseAddress* + *Blitter2DFilterOffset* + offset.

The *BlitterBaseAddress* is:
0x1920 B000.

The *BlitterRegsBaseAddress* is:
0x000.

The *Blitter2DFilterOffset* is:
0x800.

## 48.1 Register map

**Table 159: Blitter register map overview**

| Register function | Register | Offset | Type | 128-bit word alignment | |
|---|---|---|---|---|---|
| CPU Control / status registers | BLT_CTRL | 0x00 | R/W | | |
| | BLT_STA1 | 0x04 | RO | | |
| | BLT_STA2 | 0x08 | RO | | |
| | BLT_STA3 | 0x0C | Mixed | | |
| General configuration | BLT_NIP | 0x10 | R/W/LLU | Memory node | Node 128-bit word 1 |
| | BLT_USR | 0x14 | RO/LLU | | |
| | BLT_INS | 0x18 | RO/LLU | | |
| Source 1 configuration | BLT_S1BA | 0x1C | RO/LLU | | |
| | BLT_S1TY | 0x20 | RO/LLU | | Node 128-bit word 2 |
| | BLT_S1XY | 0x24 | RO/LLU | | |
| | BLT_S1SZ | 0x28 | RO/LLU | | |
| | BLT_S1CF | 0x2C | RO/LLU | | |
| Source 2 configuration | BLT_S2BA | 0x30 | RO/LLU | | Node 128-bit word 3 |
| | BLT_S2TY | 0x34 | RO/LLU | | |
| | BLT_S2XY | 0x38 | RO/LLU | | |
| | BLT_S2SZ | 0x3C | RO/LLU | | |
| | BLT_S2CF | 0x40 | RO/LLU | | Node 128-bit word 4 |
| Target configuration | BLT_TBA | 0x44 | RO/LLU | | |
| | BLT_TTY | 0x48 | RO/LLU | | |
| | BLT_TXY | 0x4C | RO/LLU | | |

Confidential

**Table 159: Blitter register map overview**

| Register function | Register | Offset | Type | 128-bit word alignment | |
|---|---|---|---|---|---|
| Rectangular clipping window | BLT_CWO | 0x50 | RO/LLU | Memory node | Node 128-bit word 5 |
| | BLT_CWS | 0x54 | RO/LLU | | |
| CLUT and color space conversion | BLT_CCO | 0x58 | RO/LLU | | |
| | BLT_CML | 0x5C | RO/LLU | | |
| 2D-rescaler | BLT_RZC | 0x60 | RO/LLU | | Node 128-bit word 6 |
| | BLT_HFP | 0x64 | RO/LLU | | |
| | BLT_VFP | 0x68 | RO/LLU | | |
| | BLT_RSF | 0x6C | RO/LLU | | |
| Flicker filter | BLT_FF0 | 0x70 | RO/LLU | | Node 128-bit word 7 |
| | BLT_FF1 | 0x74 | RO/LLU | | |
| | BLT_FF2 | 0x78 | RO/LLU | | |
| | BLT_FF3 | 0x7C | RO/LLU | | |
| ALU / Color key | BLT_ACK | 0x80 | RO/LLU | | Node 128-bit word 8 |
| | BLT_KEY1 | 0x84 | RO/LLU | | |
| | BLT_KEY2 | 0x88 | RO/LLU | | |
| Plane mask | BLT_PMK | 0x8C | RO/LLU | | |
| Raster scan trigger | BLT_RST | 0x90 | RO/LLU | | Node 128-bit word 9 |
| XYL mode | BLT_XYL | 0x94 | RO/LLU | | |
| | BLT_XYP | 0x98 | RO/LLU | | |
| Reserved | - | 0x9C | - | | |
| STBus bandwidth limiter | BLT_STB | 0xF0 | R/W | | |
| STBus protocol / packet maximum size | BLT_PKZ | 0xFC | R/W | | |

Confidential

## 48.2 Register descriptions

### BLT_CTRL Blitter control

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | | | | | | | | | SUSP2 | SUSP1 | START | RST |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x00
Type: R/W
Buffer: Immediate
Reset: 0x00
Description: This register controls the blitter activity.

[31:4] **Reserved**

[3] **SUSP2**
1: Suspend the blitter as soon as possible:
- after the current instruction is completed if the 2D-resize engine or the flicker-filter is enabled,
- at the end of the current target line in other cases.

[2] **SUSP1**
1: Suspend the blitter after the current instruction is completed

[1] **START**
Write 1 then 0 to start the blitter: The node at the address stored in BLT_NIP is fetched and executed

[0] **RST**: Soft reset: the blitter goes back to an idle state.

### BLT_STA1 Blitter status 1

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INSTR_PTR_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x04
Type: RO
Buffer: Immediate
Reset: Undefined
Description: Provides the memory address of the instruction pointer for the current blitter operation.

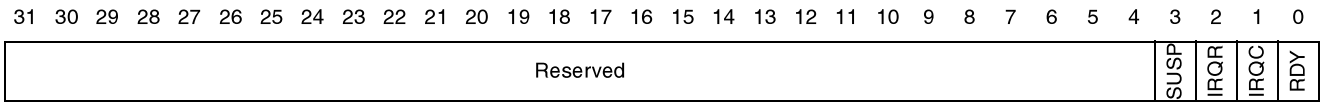
### BLT_STA2 Blitter status 2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | | | | | | TARGET_LINE_NUM | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x08
Type: RO
Buffer: Immediate
Reset: Undefined
Description: This register provides the current line number to be processed by the blitter, with reference to the target height. The first line transferred is line 0 (and can be the top or the bottom line, depending on the programmed vertical scan order in TTY).
*Note:* *This value is useful when a blitter instruction has been suspended to enable a higher priority blitter operation.*

Confidential

## BLT_STA3 Blitter status 3




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x0C

Type: Mixed: R/W except for bit 0, RO

Buffer: Immediate

Reset: Undefined

Description: This register shows the activity of the blitter (to identify the interrupt source, if any).

[31:4] **Reserved**

[3] **SUSP**: CURRENT_BLIT_SUSPENDED

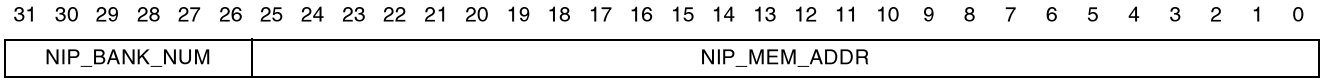
[2] **IRQR**: CURRENT_BLIT_READY2START IRQ
If the blitter has stopped in a Ready2Start state, a start command, register bit BLT_CTRL.START, is required to resume; no new node is fetched prior to the execution.

[1] **IRQC**: Current_Blit_Completed IRQ

[0] **RDY**: Ready (read only)
0: Busy
1: Ready
Bit RDY is reset on a start command, and is set again once the whole link-list has been executed (next instruction pointer = 0), or on a reset command.

*Note: Setting the appropriate bit in this register resets the interrupt request (for example, writing 0x2 in this register clears the* CURRENT_BLIT_COMPLETED *interrupt status bit, once processed).*
*Status bits 1, 2 and 3 have no meaning if the corresponding interrupt is disabled (see BLT_INS[20:18])*

## BLT_NIP Next instruction pointer




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x10

Type: R/W/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the memory location of the next instruction to execute. The blitter stops when this register is 0 (last node of the link-list).

[31:26] **NIP_BANK_NUM**: 64 MB bank number

[25:0] **NIP_MEM_ADDR**: Memory address for the next instruction pointer for current BLIT

Confidential

## BLT_USR User-specific modes

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GENERAL_PURPOSE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x14
Type: RO/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register is available for the software to implement special synchronization schemes, or any other required tricks.

## BLT_INS Blitter instruction

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | TRIG_COND_CTRL | | | | | | | XYL | Reserved | | IRQ_MASK | | | Reserved | | | PLANE_MASK | Reserved | OCSC | CKEY | RECT_CLIP | FILCK_FILT | 2DRESCALE | CLUTOP | ICSC | Reserved | SRC2 | | SRC1 | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x18
Type: RO/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register controls the data flow configuration of the blitter engine, the interrupt model and the trigger condition.

[31] **Reserved**

[30:24] **TRIG_COND_CTRL**: Set trigger condition control
[30]: wait for Vsync event, start of bottom field
[29]: wait for Vsync event, start of top field
[28]: wait for end of capture event, bottom field
[27]: wait for end of capture event, top field
[26]: wait for raster compare event, bottom field
[25]: wait for raster compare event, top field
[24]: VTG selection: 0:VTG1, 1:VTG2
*Note: A raster compare event is generated when the current video line number, provided by the selected VTG, equals the value programmed in BLT_RST (with a progressive display, use top-field control bits only).*

[23] **XYL**: When set to 1, this bit indicates that the current instruction is a XYL blitter operation

[22:21] **Reserved**

[20:18] **IRQ_MASK**: IRQ enable mask
[20]: Enable Current_Blit_Suspended interrupt. If set, an interrupt is generated once the blitter has reached the idle state, following a suspend command.
[19]: Enable CURRENT_BLIT_READY2START interrupt. If set, an interrupt is generated after the new instruction node has been captured, before the execution starts. The blitter is then in an idle state, waiting for the CPU to activate the start command in BLT_CTRL.
[18]: Enable Current_Blit_Completed interrupt. If set, an interrupt is generated once the blitter operation is completed. Then the blitter stops, and a start command is mandatory to continue.

[17:15] **Reserved**

[14] **PLANE_MASK**: Enable plane mask

[13] **Reserved**

[12] **OCSC**: Enable output color space converter

Confidential

[11] **CKEY**: Enable color key

[10] **RECT_CLIP**: Enable rectangular clipping

[9] **FLICK_FILT**: Enable flicker filter on source 2

[8] **2DRESCALE**: Enable 2D-rescaling engine on source 2

[7] **CLUTOP**: Enable CLUT-based operator on source 2

[6] **ICSC**: Enable input color space converter

[5] **Reserved**

[4:3] **SRC2**: Source 2 mode
00: Source 2 disabled
01: Source 2 fetched from memory
10: Reserved
11: Source 2 color fill register

[2:0] **SRC1**: Source 1 mode
000: Source 1 disabled
001: Source 1 fetched from memory
010: Reserved
011: Source 1 color fill register
100: Source 1 direct-copy mode
101: Reserved
110: Reserved
111: Source 1 direct-fill mode

## BLT_RST Raster scan trigger

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | | | | | | | VTG | Reserved | | | | | TRIG_LINE_NUM | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x90
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register contains the video line number for conditionally starting a blitter operation.

[31:17] **Reserved**

[16] **VTG**: 0: VTG1 1: VTG2

[15:11] **Reserved**

[10:0] **TRIG_LINE_NUM**: expected video line number to trigger

## BLT_S1BA Source 1 base address

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1BA_BANK_NUM | | | | | | S1BA_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x1C
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register provides the byte memory location of the pixmap (pixel (0,0)), for retrieving graphics data for blitter source 1. Pixel (0,0) is always the upper-left pixel.

[31:26] **S1BA_BANK_NUM**: 64 MB bank number

[25:0] **S1BA_MEM_ADDR**: byte address of the first pixel

*Note: In XYL standard mode, this register must be programmed with the base address of the target bitmap.*

Confidential

## BLT_S1XY Source 1 XY location

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| YCOORD | | | | | | | | | | | | | | | | XCOORD | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x24

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the XY location of the first pixel to be transferred, with reference to the top-left corner of the blitter source 1 bitmap [0,0]. This XY location depends on the programmed horizontal and vertical scan order (register BLT_S1TY).

[31:16] **YCOORD**: Y coordinate of the first pixel to be transferred (signed)

[15:0] **XCOORD**: X coordinate of the first pixel to be transferred (signed)

## BLT_S1TY Source 1 type

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | BIGNOTLITTLE | RGB_EXP | SUBBYTE | Reserved | | VSO | HSO | MB_FIELD | Reserved | ALPHA_RANGE | COLOR_FORM | | | | | PIXMAP_PITCH | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x20

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register configures the color format, pitch and raster scan order for the blitter source 1 pixmap.

[31] **Reserved**

[30] **BIGNOTLITTLE**: Bitmap endianness
0: Little endian 1: Big endian

[29] **RGB_EXP**: RGB expansion mode
0: Missing LSBs are filled with 0 1: MSBs are duplicated on missing LSBs

[28] **SUBBYTE**: Sub-byte formats, pixels ordering
0: Screen most right pixel in most significant bits 1: Screen most right pixel in least significant bits

[27:26] **Reserved**

[25] **VSO**: Vertical scan order
0: Top to bottom) 1: Bottom to top

[24] **HSO**: Horizontal scan order
0: Left to right 1: Right to left

[23] **MB_FIELD**: access mode in macro-block organized frame buffers (YCbCr420MB and YCbCr422MB)
0: Access in frame mode 1: Access in field mode (every other line)

[22] **Reserved**

[21] **ALPHA_RANGE**: 8-bit alpha range (for ARGB8565, ARGB8888, ACLUT88 and A8 formats only)
0: 0 to 128 (128 means opaque) 1: 0 to 255 (255 means opaque)

Confidential

[20:16] **COLOR_FORM:** pixmap color format

| RGB types | CLUT types |
|---|---|
| 0 0000: RGB565 | 0 1000: CLUT1 |
| 0 0001: RGB888 | 0 1001: CLUT2 |
| 0 0100: ARGB8565 | 0 1010: CLUT4 |
| 0 0101: ARGB8888 | 0 1011: CLUT8 |
| 0 0110: ARGB1555 | 0 1100: ACLUT44 |
| 0 0111: ARGB4444 | 0 1101: ACLUT88 |
| YCbCr types | Miscellaneous types |
| 1 0000: YCbCr888 | 1 1000: A1 |
| 1 0010: Reserved | 1 1001: A8 |
| 1 0011: YCbCr422MB | 1 1111: BYTE |
| 1 0100: YCbCr420MB | |
| 1 0101: AYCbCr8888 | |

For modes YCbCr4;2;2MB and YCbCr4;2;0MB (macroblock based, dual buffer), source 1 and source 2 are both used, and must be programmed with the same color format

[15:0] **PIXMAP_PITCH**: Pixmap pitch in bytes

*Note: In XYL standard mode, this register must be filled with the bitmap characteristics of the target bitmap. Bits 24 and 25 have no meaning and should be set to 0.*

## BLT_S1SZ Source 1/target window size

| 31 30 29 28 | 27 26 25 24 23 22 21 20 19 18 17 16 | 15 14 13 12 | 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| Reserved | WIN_HEIGHT | Reserved | WIN_WIDTH |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x28

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the width and height of the rectangular window to be transferred. The size of the target always matches source 1 size (no scaling is provided on source 1 path). Hence, there is only one physical register.

[31:28] **Reserved**

[27:16] **WIN_HEIGHT**: Height (in lines) of window to be transferred

[15:12] **Reserved**

[11:0] **WIN_WIDTH**: Width (in pixels) of window to be transferred

*Note: In XYL standard mode, this register is not used.*

## BLT_S1CF Source 1 color fill

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|
| SRC1_COLOR_FILL |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x2C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the solid color value to be used for filling or shading from source 1 during a blitter operation. The number of significant bits varies (from 1 bpp to 32 bpp) according to the bit depth of the selected color format. The value is always right-justified in the 32-bit physical register.

Confidential

## BLT_S2BA Source 2 base address

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S2BA_BANK_NUM | | | | | | S2BA_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x30

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the byte memory location address of the pixmap (pixel (0,0)), for retrieving graphics data for blitter source 2. Pixel (0,0) is always the upper-left pixel.

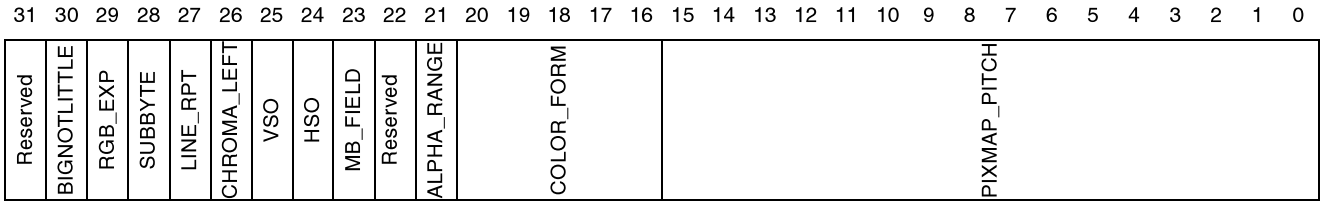
[31:26] **S2BA_BANK_NUM**: 64 MB bank number

[25:0] **S2BA_MEM_ADDR**: Byte address of the first pixel

*Note: In XYL standard mode, this register must be programmed with the base address of the clipmask bitmap, if any.*

Confidential

## BLT_S2TY Source 2 type




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x34

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register configures the color format, pitch and raster scan order for the blitter source 2 pixmap.

[31] **Reserved**

[30] **BIGNOTLITTLE**: Bitmap endianness
0: Little endian 1: Big endian

[29] **RGB_EXP**: RGB expansion mode
0: Missing LSBs are filled with 0 1: MSBs are duplicated on missing LSBs

[28] **SUBBYTE**: Sub-byte formats, pixels ordering
0: Screen most right pixel in most significant bits 1: Screen most right pixel in least significant bits

[27] **LINE_RPT**: Chroma line repeat mode if source is YCbCr420MB
0: No line repeat: the vertical chroma upsampling must be done using the 2D-resize
1: The vertical chroma upsampling uses a line repeat scheme

[26] **CHROMA_LEFT**: Chroma left extended
0: If the first chroma sample in a line is Y-only, its chroma is estimated from the following pixel only
1: If the first chroma sample in a line is Y-only, its chroma is estimated by averaging the following pixel and the previous sample (out of the defined S2 window)

[25] **VSO**: Vertical scan order
0: Top to bottom) 1: Bottom to top

[24] **HSO**: Horizontal scan order
0: Left to right 1: Right to left

[23] **MB_FIELD**: Access mode in macro-block organized frame buffers (YCbCr420MB and YCbCr422MB)
0: Access in frame mode 1: Access in field mode (every other line)

[22] **Reserved**

[21] **ALPHA_RANGE**: 8-bit alpha range (for ARGB8565, ARGB8888, ACLUT88 and A8 formats only)
0: 0 to 128 (128 means opaque) 1: 0 to 255 (255 means opaque)

Confidential

[20:16] **COLOR_FORM**: Pixmap color format

| RGB types | CLUT types |
|---|---|
| 0 0000: RGB565 | 0 1000: CLUT1 |
| 0 0001: RGB888 | 0 1001: CLUT2 |
| 0 0100: ARGB8565 | 0 1010: CLUT4 |
| 0 0101: ARGB8888 | 0 1011: CLUT8 |
| 0 0110: ARGB1555 | 0 1100: ACLUT44 |
| 0 0111: ARGB4444 | 0 1101: ACLUT88 |
| YCbCr types | MISC types |
| 1 0000: YCbCr888 | 1 1000: A1 |
| 1 0010: YCbCr422R | 1 1001: A8 |
| 1 0011: YCbCr422MB | 1 1111: BYTE |
| 1 0100: YCbCr420MB | |
| 1 0101: AYCbCr8888 | |

For modes YCbCr4;2;2MB and YCbCr4;2;0MB (macroblock based, dual buffer), sourceS 1 and 2 are both used, and must be programmed with the same color format.

[15:0] **PIXMAP_PITCH**: Pixmap pitch in bytes

*Note: In XYL standard mode, this register must be filled with the characteristics of the clipmask bitmap, if any. Bits [24, 25, 26, 27, 29] have no meaning and should be set to 0.*

### BLT_S2XY Source 2 XY location

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| YCOORD | | | | | | | | | | | | | | | | XCOORD | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x38

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the XY location of the first pixel to be transferred, with reference to the top-left corner of the blitter source 2 bitmap (0,0). This XY location depends on the programmed horizontal and vertical scan order (register BLT_S2TY).

[31:16] **YCOORD**: Y coordinate of the first pixel to be transferred (signed).

[15:0] **XCOORD**: X coordinate of the first pixel to be transferred (signed).

*Note: In XYL standard mode, this register contains the 2D-vector that gives the location of the (0,0) clipmask value, with respect to the (0,0) pixel location of the target bitmap. XCOORD provides the horizontal offset, YCOORD provides the vertical offset (both values are 16-bit signed integers).*

Confidential

### BLT_S2SZ Source 2 window size

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | SRC2_WIN_HEIGHT | | | | | | | | | | | | Reserved | | | | SRC2_WIN_WIDTH | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x3C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the width and height of the rectangular window to be transferred.

[31:28] **Reserved**

[27:16] **SRC2_WIN_HEIGHT**: Height (in lines) of source 2 window to be transferred

[15:12] **Reserved**

[11:0] **SRC2_WIN_WIDTH**: Width (in pixels) of source 2 window to be transferred

### BLT_S2CF Source 2 color fill

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRC2_COLOR_FILL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x40

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the solid color value to be used for filling or shading from source 2 during a blitter operation. The number of significant bits varies from 1 bpp to 32 bpp according to the bit depth of the selected color format. The value is always right-justified in the 32-bit physical register.

*Note: In XYL standard mode, configured for either XY or XYL sub-instruction format, this register contains the drawing color data. When XYC or XYLC format is used, the register content is "don't care".*

### BLT_TBA Target base address

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBA_BANK_NUM | | | | | | TBA_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x44

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the byte memory location of the target pixmap (pixel (0,0)), for storing the result of the blitter operation (target data). Pixel (0,0) is always the top-left pixel.

[31:26] **TBA_BANK_NUM**: 64 MB bank number

[25:0] **TBA_MEM_ADDR**: Byte address of the first pixel

*Note: In XYL standard mode, this register must be programmed with the base address of the target bitmap.*

Confidential

### BLT_TXY Target XY location

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| YCOORD | | | | | | | | | | | | | | | | XCOORD | | | | | | | | | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x4C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the XY location of the first pixel to be transferred, with reference to the top-left corner of the target bitmap (0,0). This XY location depends on the programmed horizontal and vertical scan order (register BLT_TTY).

[31:16] **YCOORD:** Y coordinate of the first pixel to be transferred (16-bit signed)

[15:0] **XCOORD:** X coordinate of the first pixel to be transferred (16-bit signed)

*Note: In XYL standard mode, this register is unused.*

Confidential

## BLT_TTY Target type

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 22 | 21 | 20 19 18 17 16 | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | BIGNOTLITTLE | Reserved | SUBBYTE | Reserved | RGB_ROUND | VSO | HSO | Reserved | ALPHA_RANGE | COLOR_FORM | PIXMAP_PITCH |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x48

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register configures the color format, pitch and raster scan order for the blitter target pixmap.

[31] **Reserved**

[30] **BIGNOTLITTLE**: Bitmap endianness
0: Little endian 1: Big endian

[29] **Reserved**

[28] **SUBBYTE**: Sub-byte formats, pixels ordering
0: Screen most right pixel in most significant bits
1: Screen most right pixel in least significant bits

[27] **Reserved**

[26] **RGB_ROUND**: Rounding mode
0: Normal rounding (+0.5 then truncation) 1: Enable 2x2 dither when rounding

[25] **VSO**: Vertical scan order
0: Top to bottom) 1: Bottom to top

[24] **HSO**: Horizontal scan order
0: Left to right 1: Right to left

[23:22] **Reserved**

[21] **ALPHA_RANGE:** 8-bit alpha range (for ARGB8565, ARGB8888, ACLUT88 and A8 formats only)
0: 0 to 128 (128 means opaque) 1: 0 to 255 (255 means opaque)

[20:16] **COLOR_FORM**: Pixmap color format

| RGB types | CLUT types |
|---|---|
| 0 0000: RGB565 | 0 1000: CLUT1 |
| 0 0001: RGB888 | 0 1001: CLUT2 |
| 0 0100: ARGB8565 | 0 1010: CLUT4 |
| 0 0101: ARGB8888 | 0 1011: CLUT8 |
| 0 0110: ARGB1555 | 0 1100: ACLUT44 |
| 0 0111: ARGB4444 | 0 1101: ACLUT88 |
| YCbCr types | Misc types |
| 1 0000: YCbCr888 | 1 1000: A1 |
| 1 0010: YCbCr422R | 1 1001: A8 |
| 1 0011: YCbCr422MB | 1 1111: BYTE |
| 1 0100: YCbCr420MB | |
| 1 0101: AYCbCr8888 | |

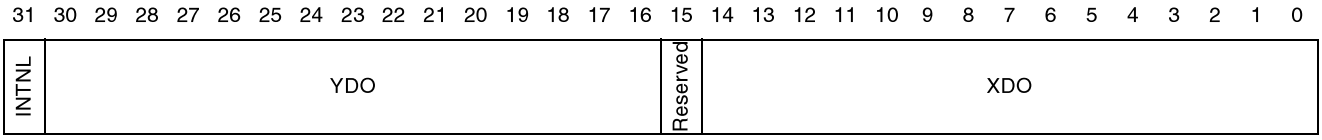
For modes YCbCr4;2;2MB and YCbCr4;2;0MB (macroblock based, dual buffer), source 1 and source 2 are both used, and must be programmed with the same color format

[15:0] **PIXMAP_PITCH**: Pixmap pitch in bytes

*Note: In XYL standard mode, this register must be filled with the characteristics of the target bitmap. Bits[24, 25, 27] have no meaning and should be set to 0.*

Confidential

## BLT_CWO Clipping window offset




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x50

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the XY location of the top-left corner of the clipping window, with reference to the target (0,0) origin (The clipping window boundaries are always specified with positive values, as there is a default clipping mechanism for any X or Y negative address).

[31] **INTNL**: Internal or external clipping
0: Writing is only allowed inside the window (including the borders)
1: Writing is only allowed outside the window (excluding the borders)

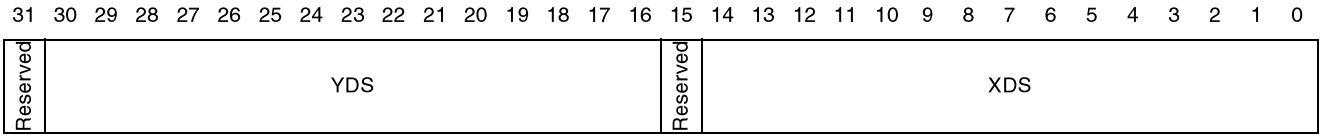
[30:16] **YDO**: Y coordinate (unsigned)

[15] **Reserved**

[14:0] **XDO**: X coordinate (unsigned)

*Note: In XYL standard mode, the rectangular clipping window can be used. In the MediaHighway clipmask implementation, the clipping window must be programmed to the intersection of the regular rectangular clipping and the rectangular clipmask window.*

## BLT_CWS Clipping window stop




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x54

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register gives the XY location of the bottom-right corner of the clipping window, with reference to the target (0,0) origin (The clipping window boundaries are always specified with positive values, as there is a default clipping mechanism for any X or Y negative address).

[31] **Reserved**

[30:16] **YDS**: Y coordinate (unsigned)

[15] **Reserved**

[14:0] **XDS**: X coordinate (unsigned)

*Note: In XYL standard mode, the rectangular clipping window can be used. In the MediaHighway clipmask implementation, the clipping window must be programmed to the intersection of the regular rectangular clipping, and the rectangular clipmask window.*

Confidential

## BLT_CCO CLUT and color conversion operators

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | | CLUT_ERRDIFF | | | | CLUT_UPDATE | CLUT_MODE | | Reserved | | | | | CCO_OUTVIDnGFX | CCO_OUTSIGN | CCO_OUTCOL | Reserved | | | | CCO_INVIDnGFX | CCO_INDIR | CCO_INSIGN | CCO_INCOL |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x58

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register controls the two color space converters and the CLUT operator.

**CLUT operators**

[31:23] **Reserved**

[22:19] **CLUT_ERRDIFF**: Error diffusion weight (if color reduction)
0000: 0% (disabled) 0001: 100%
0010: 75% 0011: 50%
0100: 25% 1000: Adaptive

[18] **CLUT_UPDATE**: Enable CLUT update

[17:16] **CLUT_MODE**: CLUT operation modes
00: Color expansion mode ((A)CLUT*n* >> true color)
01: Color correction (true color >> true color) 10: Color reduction mode (true color >> CLUT*n*)

[15:11] **Reserved**

**Color-space converters**

[10] **CCO_OUTVID*n*GFX**: Output color space converter uses
0: Graphics matrix 1: Video matrix

[9] **CCO_OUTSIGN:** Output color converter chroma format
0: Offset binary 1: Two's complement, signed

[8] **CCO_OUTCOL:** Output color converter colorimetry
0: ITU-R BT.601 1: ITU-R BT.709

[7:4] **Reserved**

[3] **CCO_INVID*n*GFX**: Input color space converter uses
0: Graphics matrix 1: Video matrix

[2] **CCO_INDIR**: Input color converter direction
0: YCbCr2RGB 1: RGB2YCbCr
The direction for the output color converter is automatically the opposite

[1] **CCO_INSIGN**: Input color converter chroma format
0: Offset binary 1: Two's complement, signed

[0] **CCO_INCOL**: Input color converter colorimetry
0: ITU-R BT.601 1: ITU-R BT.709

*Note: The CLUT and the input color space converter cannot be used in XYL mode. But the output color space converter is available.*

Confidential

## BLT_CML CLUT memory location

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CML_BANK_NUM | | | | | | CML_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | Reserved | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x5C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: Provides the byte-memory location to retrieve the CLUT data in the local memory. As the CLUTs are always aligned on 128-bit word boundary, the four LSBs are ignored.

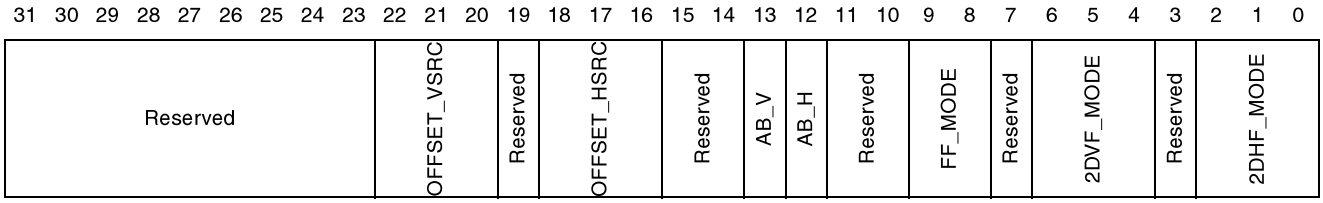
[31:26] **CML_BANK_NUM**: 64 MB bank number

[25:4] **CML_MEM_ADDR**: Byte address

[3:0] **Reserved**

Confidential

## BLT_RZC 2D resize control




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x60

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register configures the blitter 2D resize engine.

[31:23] **Reserved**

[22:20] **OFFSET_VSRC**: Initial subposition for the VSRC

[19] **Reserved**

[18:16] **OFFSET_HSRC**: Initial subposition for the HSRC

[15:14] **Reserved**

[13] **AB_V**: Enable AlphaVBorder

[12] **AB_H**: Enable AlphaHBorder

[10:11] **Reserved**

[9:8] **FF_MODE**: Flicker filter mode

00: Force filter 0
01: Adaptive flicker filtering (RGB format only)
10: Test mode
11: Reserved

[7] **Reserved**

[6:4] **2DVF_MODE**: 2D-filter mode (vertical)

0xx: Vertical resizer disabled

| | |
|---|---|
| 100: V.resizer enabled | Vfilter inactive |
| 101: V.resizer enabled | Vfilter active on color channels, inactive on alpha channel |
| 110: V.resizer enabled | Vfilter inactive on color channels, active on alpha channel |
| 111: V.resizer enabled | Vfilter active on both color and alpha channels |

[3] **Reserved**

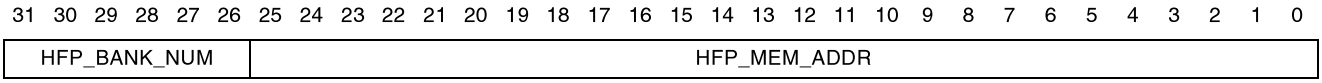
[2:0] **2DHF_MODE**: 2D-filter mode (horizontal)

0xx: Horizontal resizer disabled

| | |
|---|---|
| 100: H.resizer enabled | Hfilter inactive |
| 101: H.resizer enabled | Hfilter active on color channels, inactive on alpha channel |
| 110: H.resizer enabled | Hfilter inactive on color channels, active on alpha channel |
| 111: H.resizer enabled | VHilter active on both color and alpha channels |

Confidential

### BLT_HFP Horizontal filter coefficients pointer




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x64

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

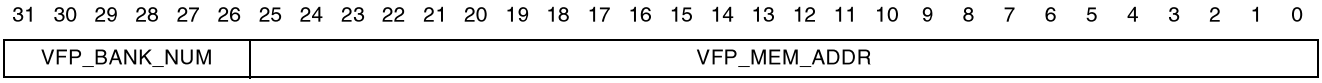
Reset: 0x00

Description: This register provides the byte memory location for retrieving the horizontal filter coefficients in the local memory (40 bytes).

[31:26] **HFP_BANK_NUM**: 64 MB bank number

[25:0] **HFP_MEM_ADDR**: Horizontal filter first coefficient byte address

### BLT_VFP Vertical filter coefficients pointer




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x68

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

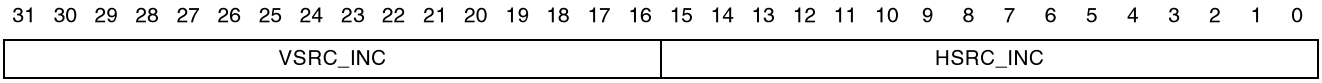
Description: This register provides the byte memory location for retrieving the vertical filter coefficients in the local memory (40 bytes).

[31:26] **VFP_BANK_NUM**: 64 MB bank number

[25:0] **VFP_MEM_ADDR**: Vertical filter first coefficient byte address

Confidential

### BLT_RSF Resize scaling factor




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x6C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the scaling factors for both the horizontal and the vertical sample rate converters.

[31:16] **VSRC_INC**: Increment value for VSRC (6.10 format)

[15:0] **HSRC_INC**: Increment value for HSRC (6.10 format)

### BLT_FF0 Flicker filter 0

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|
| Reserved | T0 | COEFF0_N3 | COEFF0_N2 | COEFF0_N1 |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x70

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the flicker filter 0 coefficients, and the threshold value used to select F0. F0 is used between 0 and T0-1.

[31:28] **Reserved**

[27:24] **T0**: Threshold value

[23:16] **COEFF0_N3**: (*n* + 1) coefficient (1.7 format)

[15:8] **COEFF0_N2**: (*n*) coefficient (1.7 format)

[7:0] **COEFF0_N1**: (n - 1) coefficient (1.7 format)

### BLT_FF1 Flicker filter 1

| 31 30 29 28 | 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|
| Reserved | T1 | COEFF1_N3 | COEFF1_N2 | COEFF1_N1 |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x74

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the flicker filter 1 coefficients, and the threshold value used to select F1. F1 is used between T0 and T1-1.

[31:28] **Reserved**

[27:24] **T1**: Threshold value

[23:16] **COEFF1_N3**: (*n* + 1) coefficient (1.7 format)

[15:8] **COEFF1_N2**: (*n*) coefficient (1.7 format)

[7:0] **COEFF1_N1**: (*n* - 1) coefficient (1.7 format)

Confidential

### BLT_FF2 Flicker filter 2

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | T2 | | | | COEFF2_N3 | | | | | | | | COEFF2_N2 | | | | | | | | COEFF2_N1 | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x78

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the flicker filter 2 coefficients, and the threshold value used to select F2. F2 is used between T1 and T2-1.

[31:28] **Reserved**

[27:24] **T2**: Threshold value

[23:16] **COEFF2_N3**: (*n* + 1) coefficient (1.7 format)

[15:8] **COEFF2_N2**: (*n*) coefficient (1.7 format)

[7:0] **COEFF2_N1**: (*n* - 1) coefficient (1.7 format)

### BLT_FF3 Flicker filter 3

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | | COEFF3_N3 | | | | | | | | COEFF3_N2 | | | | | | | | COEFF3_N1 | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x7C

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register provides the flicker filter 3 coefficients, which operate in the range T2 to 15.

[31:24] **Reserved**

[23:16] **COEFF3_N3:** (*n*+1) coefficient (1.7 format)

[15:8] **COEFF3_N2:** (*n*) coefficient (1.7 format)

[7:0] **COEFF3_N1**: (*n*-1) coefficient (1.7 format)

Confidential

## BLT_ACK ALU and color key control

| 31 30 29 28 27 26 25 24 | 23 22 | 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 | 3 2 1 0 |
|---|---|---|---|---|---|
| Reserved | CKEY_SEL | ACK_CKEY | GALPHA_ROPID | Reserved | MODE |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x80

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: This register configures the ALU and the color key module of the blitter.

[31:24] **Reserved**

[23:22] **CKEY_SEL**: Color key input selection
00: Source 1 pixel bus (destination color key)
01: Source 2 pixel bus before the CLUT (source color key)
10: Source 2 pixel bus after the CLUT (source color key)
11: Reserved

[21:16] **ACK_CKEY**: Color key configuration

[5:4] x0: Red / index component ignored (disabled = always match)
01: Red / index enabled: match if ($R_{min} \leq R \leq R_{max}$)
11: Red / index enabled: match if (($R < R_{min}$) or ($R > R_{max}$))
When using the color key feature on a CLUT bitmap, only the blue channel parameters are relevant, as internally, the blue component and the index information are sharing the same data path.

[3:2] **MODE[3:2]**
x0: Green component ignored (disabled = always match)
01: Green enabled: match if ($G_{min} \leq G \leq G_{max}$)
11: Green enabled: match if (($G < G_{min}$) or ($G > G_{max}$))

[1:0] **MODE[1:0]**
x0: Blue component ignored (disabled = always match)
01: Blue enabled: match if ($B_{min} \leq B \leq B_{max}$)
11: Blue enabled: match if (($B < B_{min}$) or ($B > B_{max}$))

[15:8] **GALPHA_ROPID**
ALU global alpha (8 bits / 0 to 128), in blending mode, or ROP identifier (4 LSBs), in logical mode.
Raster operation table (GALPHA_ROPID[3:0]):

| | | | |
|---|---|---|---|
| 0000: CLEAR | result = all 0 | 0001: AND | result = S2 AND S1 |
| 0010: ANDrev | result = S2 AND (NOT S1) | 0011: COPY | result = S2 |
| 0100: ANDinvert | result = (NOT S2) AND S1 | 0101: NOOP | result = S1 |
| 0110: XOR | result = S2 XOR S1 | 0111: OR | result = S2 OR S1 |
| 1000: NOR | result = (NOT S2) AND (NOT S1) | 1001: EQUIV | result = (NOT S2) XOR S1 |
| 1010: INVERT | result = (NOT S1) | 1011: ORreverse | result = S2 OR (NOT S1) |
| 1100: COPYinv | result = (NOT S2) | 1101: ORinvert | result = (NOT S2) OR S1 |
| 1110: NAND | result = (NOT S2) OR (NOT S1) | 1111: SET | result = all 1 |

[7:4] **Reserved**

[3:0] **MODE**: ALU operation modes

| | |
|---|---|
| 0000: Bypass source 1 | 0001: Logical operation |
| 0010: Blending mode, source 2 not premultiplied | 0011: Blending mode, source 2 premultiplied |
| 0100: Clipmask pass in logical mode / First pass | 0101: Clipmask pass in blending mode |
| 0110: 4:2:0 source 1 / 2 YCbCr concatenation | 0111: Bypass source 2 |
| 1000: Clipmask pass in logical mode / 2nd pass | 1001: Clipmask in XYL mode, logical operation |

1010: Clipmask in XYL mode, blending operation, source 2 not premultiplied
1011: Clipmask in XYL mode, blending operation, source 2 premultiplied

*Note: In XYL standard mode, only ALU operation modes 1, 2, 3, 7, 9, 10 and 11 are available.*

Confidential

### BLT_KEY1 Color key 1

| 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| Reserved | RED_MIN | GREEN_MIN | BLUE_MIN |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x84
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register contains the color key minimum values for the RGB components.

[31:24] **Reserved**

[23:16] **RED_MIN**: Red component minimum value

[15:8] **GREEN_MIN**: Green component minimum value

[7:0] **BLUE_MIN**: Blue component minimum value / index

### BLT_KEY2 Color key 2

| 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| Reserved | RED_MAX | GREEN_MAX | BLUE_MAX |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x88
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register contains the color key maximum values for the RGB components.

[31:24] **Reserved**

[23:16] **RED_MAX**: Red component maximum value

[15:8] **GREEN_MAX**: Green component maximum value

[7:0] **BLUE_MAX**: Blue component maximum value / index

*Note: When the color key module is intended to work on CLUT type data, only the blue path is used (green and red should be disabled). Thus, it is possible to track a single index, as well as a range of indexes.*

### BLT_PMK Plane mask

| 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| PLANE_MASK | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x8C
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register provides the plane mask when the feature is enabled in the blitter register bit BLT_INS[14]. The number of valid bits (1 to 32) in a given configuration must match the selected target pixel width. The value is always right-justified if bpp < 32.

*Note: A bit in the target pixel can be updated by the new color bit, only if the corresponding bit in the mask has been set to 1. The feature is available in XYL standard mode.*

Confidential

### BLT_XYL XYL mode control

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NUM_SUBINS | | | | | | | | | | | | | | | | Reserved | | | | | | SUBINS | | Reserved | | | | | | | |

Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x94
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register allows to control a blitter execution when operating in XYL mode.

[31:16] **NUM_SUBINS**: Number of subinstructions in the XYL blitter instruction

[15:10] **Reserved**

[9:8] **SUBINS**: Subinstructions format
00: XY subinstruction
01: XYL subinstruction
10: XYC subinstruction
11: XYLC subinstruction

[7:1] **Reserved**

[0] **Reserved**: Must be set to 0.

### BLT_XYP XY subinstructions pointer

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SUBI_BANK_NUM | | | | | | SUBI_MEM_ADDR | | | | | | | | | | | | | | | | | | | | | | | | | |

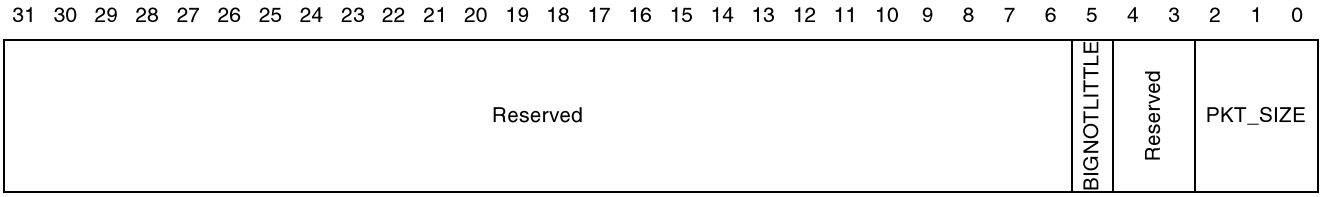
Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0x98
Type: Read/LLU
Buffer: Double-bank: automatic hardware toggle
Reset: 0x00
Description: This register provides the byte memory location for the subinstruction buffer

[31:26] **SUBI_BANK_NUM**: 64 MB bank number

[25:0] **SUBI_MEM_ADDR**: Subinstructions buffer address

Confidential

## BLT_PKZ Packet size control




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0xFC
Type: R/W
Buffer: Immediate
Reset: 0x00
Description: This register controls the packet size on an STBus transaction.

[31:6] **Reserved**

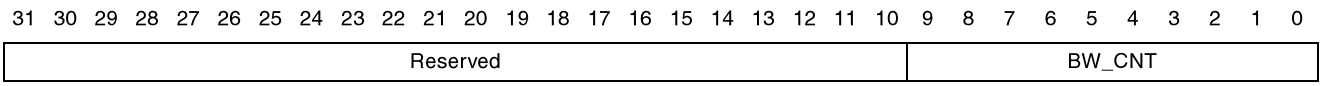
[5] **BIGNOTLITTLE**: Bitmap endianness
0: Little endian 1: Big endian

[4:3] **Reserved**

[2:0] **PKT_SIZE**: Should be set to 000.
000: Packet size = message size 001: Packet size = 16 x 128-bit words
010: Packet size = 8 x 128-bit words 011: Packet size = 4 x 128-bit words
100: Packet size = 2 x 128-bit words 101: Packet size = 1 x 128-bit words

*Note: The reason for splitting a message transaction into a smaller entity called packet, is that a CPU access can be inserted between two consecutive packets.*

## BLT_STB STBus bandwidth limiter




Address: *BlitterBaseAddress* + *BlitterRegsBaseAddress* + 0xF0
Type: R/W
Buffer: Immediate
Reset: 0x00
Description: This register controls STBus bandwidth limiter.

[31:10] **Reserved**

[9:0] **BW_CNT**: Bandwidth count
Sets the number of cycle between two requests on the STBus

Confidential

## 48.3 Blitter 2D-filter coefficients

### 48.3.1 Register map

**Table 160: Blitter 2D-filter register map overview**

| Register function | Offset | Name | 128-bit word alignment | |
|---|---|---|---|---|
| HFilter coefficients | 0x00 | BLT_HFC1 | HFilter | HFilter 128-bit word 1 |
| | 0x04 | BLT_HFC2 | | |
| | 0x08 | BLT_HFC3 | | |
| | 0x0C | BLT_HFC4 | | |
| | 0x10 | BLT_HFC5 | | HFilter 128-bit word 2 |
| | 0x14 | BLT_HFC6 | | |
| | 0x18 | BLT_HFC7 | | |
| | 0x1C | BLT_HFC8 | | |
| | 0x20 | BLT_HFC9 | | HFilter 128-bit word 3 |
| | 0x24 | BLT_HFC10 | | |
| Reserved | 0x28 | | | |
| | 0x2C | | | |
| VFilter coefficients | 0x30 | BLT_HFC1 | VFilter | VFilter 128-bit word 3 |
| | 0x34 | BLT_HFC2 | | |
| | 0x38 | BLT_HFC3 | | |
| | 0x3C | BLT_HFC4 | | |
| | 0x40 | BLT_HFC5 | | VFilter 128-bit word 4 |
| | 0x44 | BLT_HFC6 | | |
| | 0x48 | BLT_HFC7 | | |
| | 0x4C | BLT_HFC8 | | |
| | 0x50 | BLT_HFC9 | | VFilter 128-bit word 5 |
| | 0x54 | BLT_HFC10 | | |
| Reserved | 0x58 | | | |
| | 0x5C | | | |

The filter coefficients are loaded from memory each time a 2D-resize operation with filtering is programmed. The pointers, BLT_HFP and BLT_VFP must be aligned on a 128-bit word boundary (but the coefficients for the horizontal and vertical filters are totally independent, they can be stored anywhere in the blitter memory space, not necessarily at adjacent locations).

Confidential

### 48.3.2 Blitter 2D-filter registers description

**BLT_HFCn** **Horizontal filter coefficients 1 to 10**

| | 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|
| **HFC1** | K0.3 | K0.2 | K0.1 | K0.0 |
| **HFC2** | K1.2 | K1.1 | K1.0 | K0.4 |
| **HFC3** | K2.1 | K2.0 | K1.4 | K1.3 |
| **HFC4** | K3.0 | K2.4 | K2.3 | K2.2 |
| **HFC5** | K3.4 | K3.3 | K3.2 | K3.1 |
| **HFC6** | K4.3 | K4.2 | K4.1 | K4.0 |
| **HFC7** | K5.2 | K5.1 | K5.0 | K4.4 |
| **HFC8** | K6.1 | K6.0 | K5.4 | K5.3 |
| **HFC9** | K7.0 | K6.4 | K6.3 | K6.2 |
| **HFC10** | K7.4 | K7.3 | K7.2 | K7.1 |

Address: *BlitterBaseAddress* + *Blitter2DFilterOffset* + 0x00 (HFC1), 0x04 (HFC2), 0x08 (HFC3), 0x0C (HFC4), 0x10 (HFC5), 0x14 (HFC6), 0x18 (HFC7), 0x1C (HFC8), 0x20 (HFC9), 0x24 (HFC10)

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: These registers store the horizontal sample rate converter (HSRC) coefficients. There are eight subfilters for the HSRC; each subfilter has five 8-bit coefficients. Coefficient *j* of subfilter *i* (subposition *i*) is designated K*i.j*[7:0] and the format is S1.6 (1.0 is encoded as 0x40). The corresponding filter coefficient structure (40 bytes) must be aligned on a 128-bit word boundary when stored in the local memory.

Confidential

## BLT_VFCn Vertical filter coefficients 1 to 10

| | 31 30 29 28 27 26 25 24 | 23 22 21 20 19 18 17 16 | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|
| **VFC1** | K0.3 | K0.2 | K0.1 | K0.0 |
| **VFC2** | K1.2 | K1.1 | K1.0 | K0.4 |
| **VFC3** | K2.1 | K2.0 | K1.4 | K1.3 |
| **VFC4** | K3.0 | K2.4 | K2.3 | K2.2 |
| **VFC5** | K3.4 | K3.3 | K3.2 | K3.1 |
| **VFC6** | K4.3 | K4.2 | K4.1 | K4.0 |
| **VFC7** | K5.2 | K5.1 | K5.0 | K4.4 |
| **VFC8** | K6.1 | K6.0 | K5.4 | K5.3 |
| **VFC9** | K7.0 | K6.4 | K6.3 | K6.2 |
| **VFC10** | K7.4 | K7.3 | K7.2 | K7.1 |

Address: *BlitterBaseAddress* + *Blitter2DFilterOffset* + 0x30 (VFC1), 0x34 (VFC2), 0x38 (VFC3), 0x3C (VFC4), 0x40 (VFC5), 0x44 (VFC6), 0x48 (VFC7), 0x4C (VFC8), 0x50 (VFC9), 0x54 (VFC10)

Type: Read/LLU

Buffer: Double-bank: automatic hardware toggle

Reset: 0x00

Description: These registers store the vertical sample rate converter (VSRC) coefficients. There are eight subfilters for the VSRC; each subfilter has five 8-bit coefficients. Coefficient *j* of subfilter *i* (subposition *i*) is designated K*i.j*[7:0] and the format is S1.6 (1.0 is encoded as 0x40). The corresponding filter coefficients structure (40 bytes) must be aligned on a 128-bit word boundary when stored in the local memory.

Confidential

# 49 Main and auxiliary displays

## 49.1 Overview

The display processors form a high quality scaling engine to provide video display processing. They take video lines in 4:2:2 raster (R) format (one buffer) or 4:2:0 macroblock (MB) format (two buffers) from external memory. 4:2:2 MB input format is also supported. Images are read from memory in either frame format or field format independently programmable for chroma and luma. This allows frame or field based vertical filtering.

The display processors can perform zoom out up to a factor of four (horizontally and vertically) and unlimited zoom in. They provide YCbCr digital output in 4:4:4 or 4:2:2 10-bit format.

Sub-pixel and sub-line resolution pan/scan components can be specified within the source image to determine the start of the image to be displayed from the display buffer.

## 49.2 Fully programmable horizontal sample rate converters

The horizontal sample rate converters include the following features:

- 8 taps interpolation luma/chroma filters
- time varying coefficients using a polyphase filtering technique,
- 32 phases and 10 bit-coefficients (value -512 to 511) stored in RAM, that is, 16 x 8 x 8 + 3 x 32 bits for luma (the same for chroma). See Chapter 50: *Main and auxiliary display registers* *on page 506* for details,
- horizontal linear resizing of video,
- linear resizing: continuous resizing of video horizontally, by a programmable factor from 1/4 up to x 8192 with 2^(-13) step,
- equidistant fractional pixel position down to 1/32 of pixel,
- horizontal pan down to 1/32 pixel,
- 150 Mpixel/s maximum pixel generation rate.

## 49.3 Fully programmable vertical sample rate converters

Applications: zoom in/out, letter box resizing chroma upsampling (4:2:0 to 4:2:2).

The vertical sample rate converters include the following features:

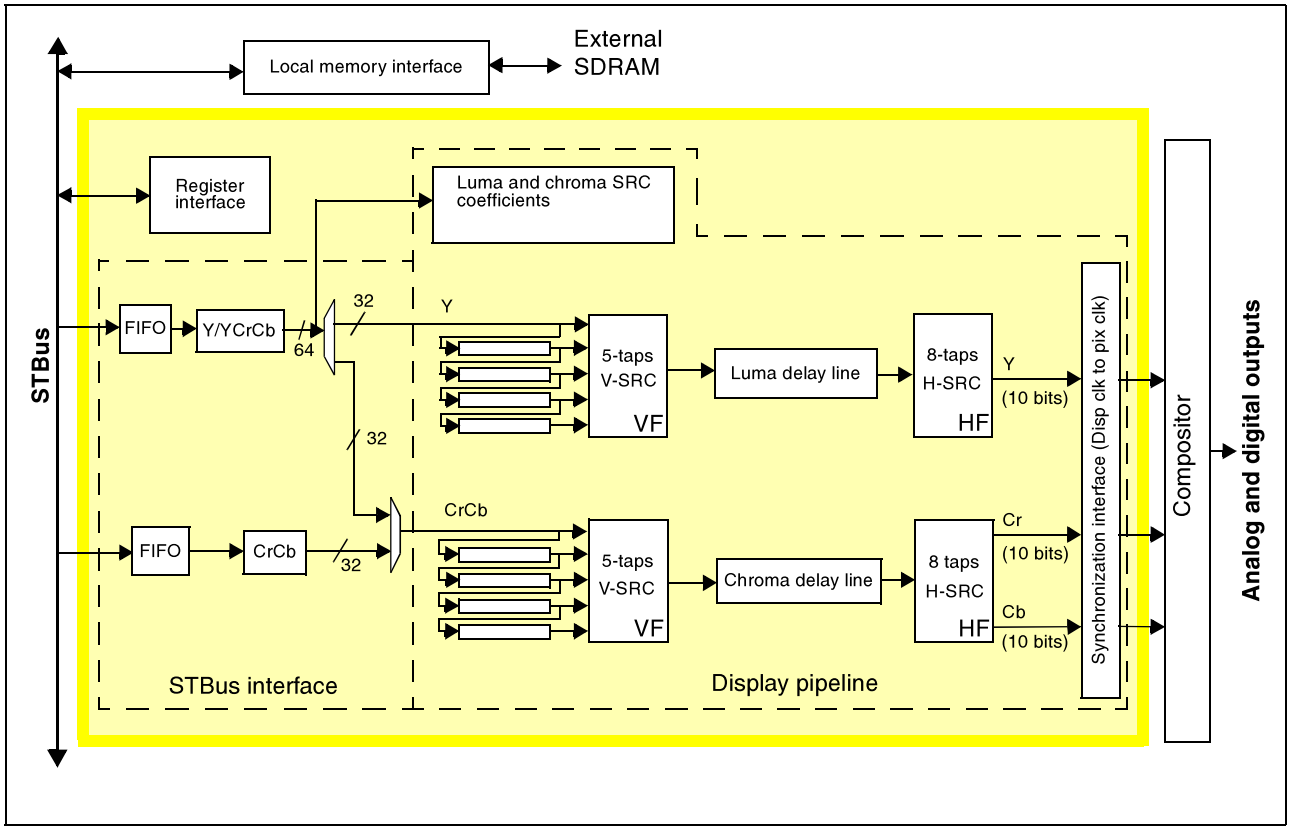
- five taps interpolation luma/chroma filters,
- time varying coefficients using a polyphase filtering technique,
- 32 phases and 10 bit-coefficients stored in RAM, that is, 16 x 5 x 8 + 2*32 bits for luma, same for chroma.
- equidistant fractional line position down to 1/32 of line,
- field or frame based interpolation modes,
- vertical scan down to 1/32 line,
- vertical linear resizing of video,
- linear resizing: continuous resizing of video vertically, by a programmable factor from 1/4 and up to x8192 with 2^(-13) step,
- 150 Mpixel/s maximum pixel generation rate.

Confidential

## 49.4 Block diagram

Figure 151 provides a functional system overview of the HD and ID display block.

**Figure 151: Graphics and video generic compositor overview**




There are four parallel vertical filters and two 8-tap chroma filters after the chroma delay lines.

## 49.5 Linear motion upconverter (LMU)

The LMU de-interlaces standard-definition pictures. When enabled, it generates twice as many lines as it receives, calculating every other line from the incoming field and the last two fields. The original lines pass through unchanged, and the last two fields are stored with motion estimation values in the LMU chroma and luma buffers. It interfaces to an external BIST controller for testing of the internal line memories.

The LMU uses a motion-adaptive, spatial/temporal de-interlacing algorithm controlled by external registers. The luma and chroma blocks are independent but share LMU_BPPL and LMU_VINL register values.

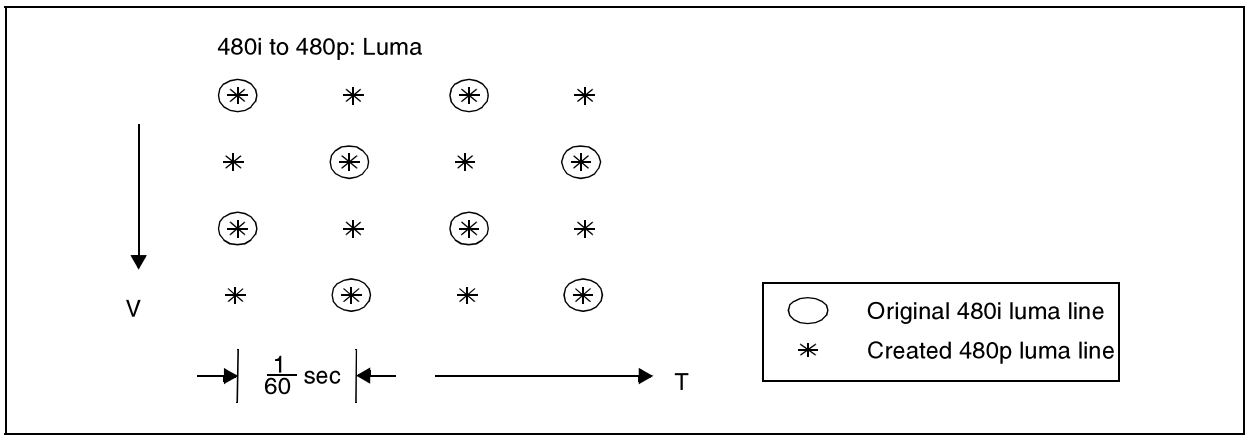
The input line size must not exceed 720 pixels due to the limited size of the internal line memories. The de-interlacing algorithm is only required for image sizes up to 720 x 480 interlaced, that is CCIR601 resolution. These images can originate from the MPEG decoding process or as input from the standard-definition pixel port.

Luminance and chrominance signals are converted by the LMU de-interlacing algorithm, using information from three adjacent source fields to generate each progressive output frame.

Confidential

Figure 152 illustrates the vertical/temporal relationship of lines before and after the luma de-interlacing process.

**Figure 152: Vertical/temporal input and output line relationships**




In the above example, the LMU block calculates extra luma lines to generate a progressive 480 line 60 frame/s display from an interlaced 480 line 30 frame/s input. For each incoming field, lines corresponding to that field are passed right through the block. Lines corresponding to the previous field must be estimated by the LMU algorithm. Four different algorithms can be selected:

- Motion adaptive de-interlacing, used for video sources. Missing lines are interpolated based on the amount of motion estimated from the previous field and frame.
- Missing lines are obtained from the following field, used for film sources.
- Missing lines are obtained from the previous field, used for film sources.
- Missing lines are blanked, simulating an interlaced display.

The associated film-mode detector accumulates frame differences over an entire field and passes to the micro two 8-bit values representing the frame difference of both the luma and chroma components. For optimum motion sensitivity in small regions of the picture and noise rejection, the data is collected as follows:

1. Pixels are partitioned in groups of 16.
2. The absolute value of the frame difference is accumulated for each 16 pixel group.
3. The largest accumulated value over all groups in the field is stored in a register and is available to the host processor.

The LMU can be activated independently for luma and chroma.

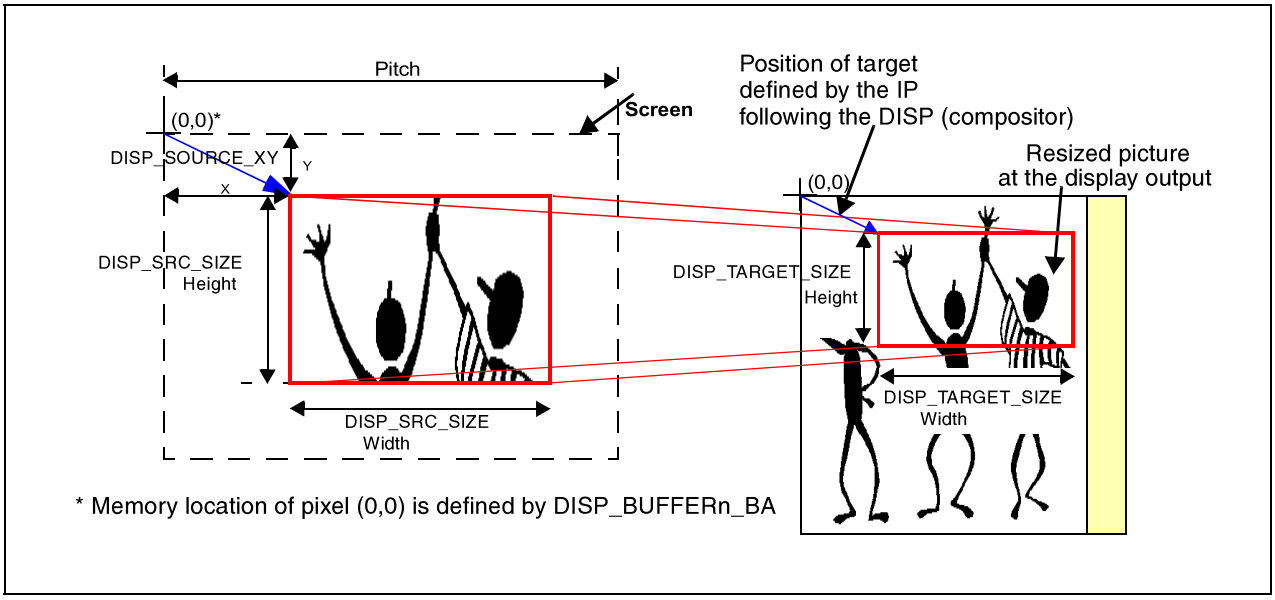
Confidential

## 49.6 Functional description

### 49.6.1 Memory addressing of the source picture

The source picture to be resized and sent to video output is stored in external memory. It is defined by the base address of the first pixel of picture stored in memory (DISP_LUMA_BA and DISP_CHR_BA). Two base addresses are needed when the picture is stored in 4:2:0 MB (or 4:2:2 MB) format (for luma and chroma) and only one base address for 4:2:2 R format (luma and chroma are stored together in the memory). The DISP_PMP register defines the pitch. This is the distance in memory between two vertically adjacent samples used for vertical filtering when 4:2:2 R input format is used, and line length in pixel unit when 4:2:0 MB or 4:2:2 MB input format is used. By changing pitch value, field or frame filtering is done in vertical SRC when 4:2:2 R input format is used. Inside this complete picture upper left corner of rectangular part to be resized is defined by DISP_LUMA(CHR)_XY and its size by DISP_LUMA(CHR)_SIZE (see Figure 153). This is the only part of picture read from external memory and is considered as ‘source’ in the following sections.

**Figure 153: Source and target definition**

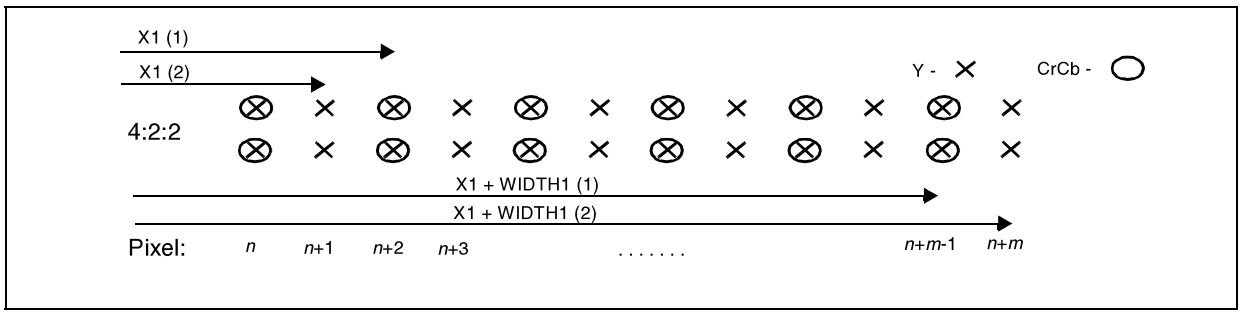



When the 4:2:2 R input format is used, the source picture position is defined by DISP_LUMA_XY.X1 and .Y1, and size by DISP_LUMA_SIZE.WIDTH1 and .HEIGHT1. Both position and size are given with pixel precision. This means that for odd values of X1 and/or (X1+WIDTH1), pixels with only luma information are pointed (see Figure 154). In that case (an odd number of pixels from (0,0) to top left corner of source) first existing chroma sample **right** to border defined by X1 is taken as first horizontal chroma sample. In the example of Figure 154, for both values of X1(1) and X1(2), chroma corresponding to pixel number *n*+2 is taken as first left chroma sample of the source picture. In a similar manner, for last horizontal chroma sample when X1 + WIDTH1 is odd, chroma from pixel left to one defined by X1+WIDTH1 is taken as the last horizontal chroma sample in the source picture. In the example of Figure 154, for both values

Confidential

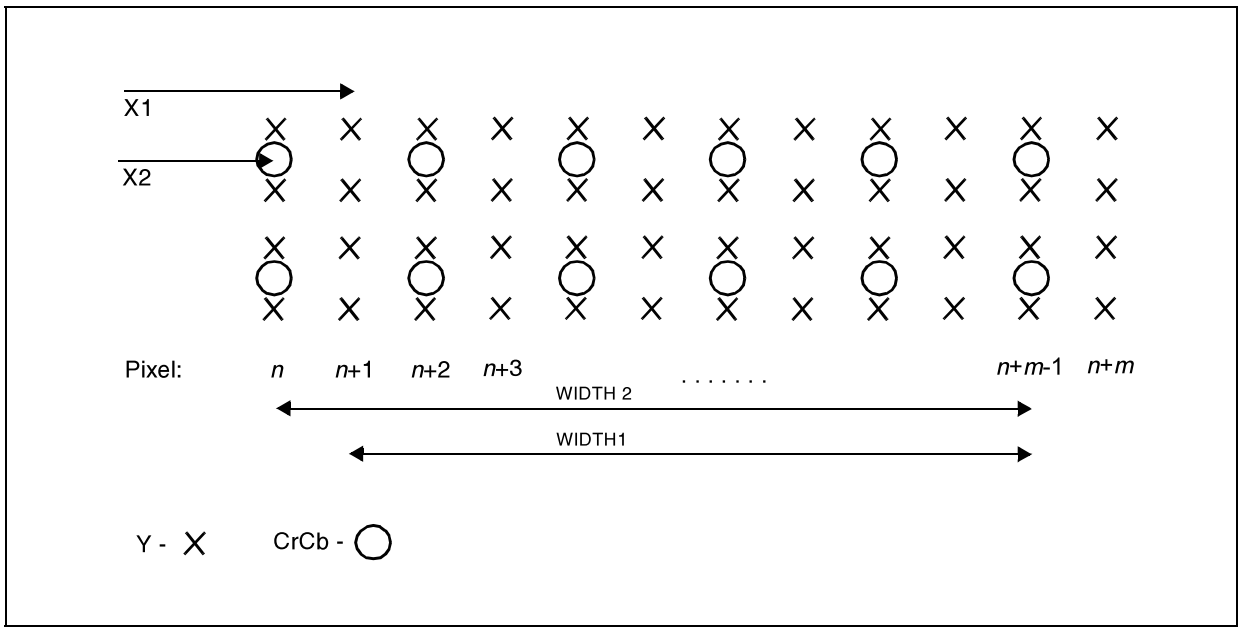
X1+WIDTH1 (1) and X1+WIDTH1 (2), chroma from pixel number *n*+*m*-1 is taken as the last horizontal chroma sample within the source.

**Figure 154: Chroma sampling for odd-number-sized sources (example)**




When the 4:2:0 MB or 4:2:2 MB input format is used, first and last luma and chroma samples are defined independently. Luma samples of source are defined by X1, Y1, WIDTH1 and HEIGHT1 (registers DISP_LUMA_XY and DISP_LUMA_SIZE) comparing to top-left luma sample of complete picture defined by DISP_LUMA_BA. In the same manner, chroma samples of the source picture are defined by X2, Y2, WIDTH2 and HEIGHT2 (registers DISP_CHR_XY and DISP_CHR_SIZE) comparing to top left luma sample of complete picture defined by DISP_CHR_BA. Figure 155 shows an example of this.

**Figure 155: 4:2:0 input format (example)**



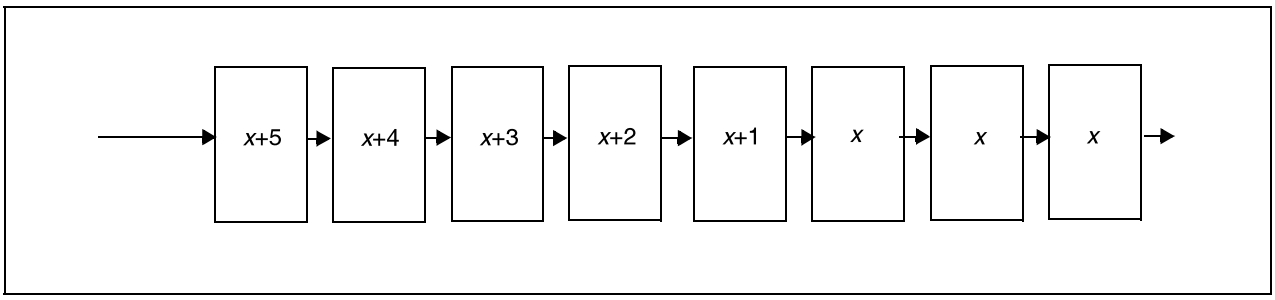

Confidential

## 49.6.2 Processing of first and last pixels

In order to have a clean picture on vertical and horizontal borders, some special configurable processing is performed. If the first (or last) pixel of source picture to be resized is not the first (or last) pixel of the whole picture stored in memory, it is possible to take into account pixels that are before (or after) that first (or last) pixel of the source. In this case, DISP_LUMA(CHR)_XY and DISP_LUMA(CHR)_SIZE should be programmed larger by a few pixels then the picture we want to resize. In other words, DISP_LUMA(CHR)_XY and DISP_LUMA(CHR)_SIZE define part of a picture that are loaded in filters (accessed from memory). DISP_LUMA(CHR)_V(H)SRC bits FYPR, FYLR, FCPR and FCLR define how many times the first/last pixel/line of that picture is

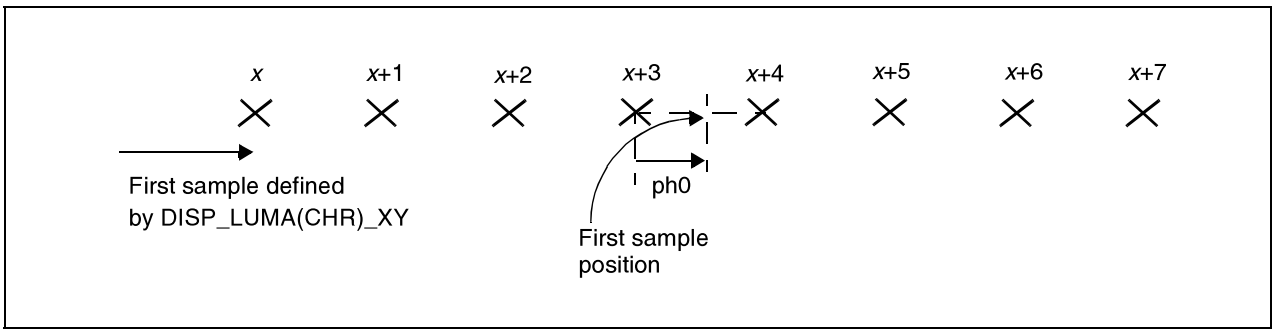
repeated in the horizontal/vertical source filter. Figure 156 shows an example of horizontal filter initialization where the first sample is repeated two times.

**Figure 156: Initiations of horizontal SRC pipeline example (first sample repeated twice)**




The horizontal and vertical position of the source picture can be defined with an accuracy of 1/32 of pixel/sample. It is defined by initial phase of polyphase DTO, which is a 13-bit register (DISP_LUMA/C_HSRC) giving 1/2^13 accuracy, but internally rounded to 1/32. Figure 157 shows an example of horizontal pan and scan, where FI(c)pr bits defines 0 repeat of the first sample and where initial phase (ph0) is different from 0. In Figure 157, the initial sample is between samples (*x*+1) and (*x*+2).

**Figure 157: Horizontal pan and scan example**

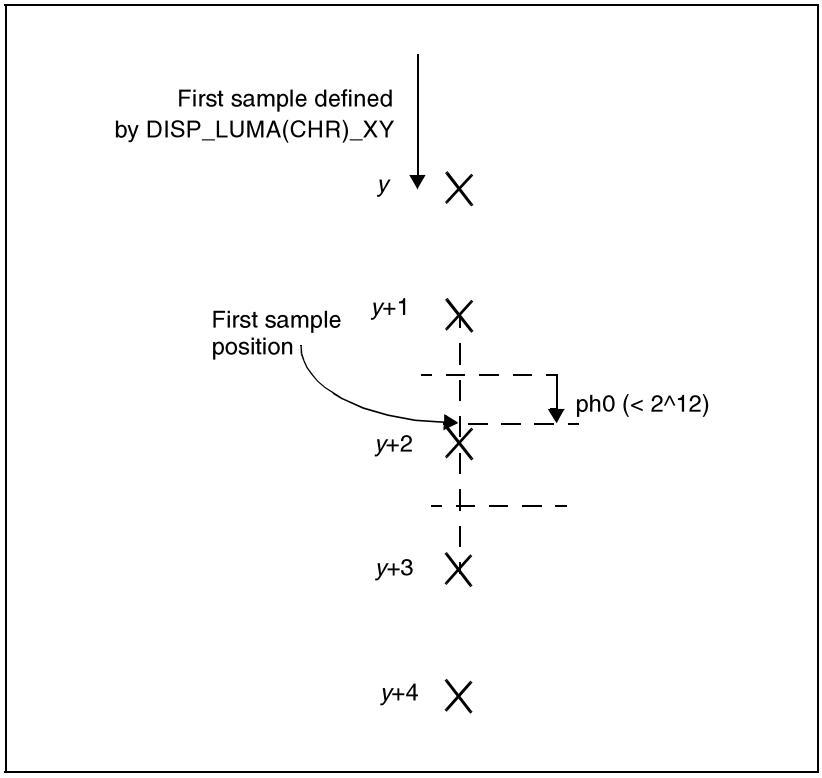



If initial phase is 0 then the first sample is (*x*+3) in previous example and (*x*+1) in the Figure 157 example.

Vertical pan and scan is defined in a similar manner, taking into account that the number of taps is five (an odd number). Where FI(c)Ir bits defines repeat of the first sample and where initial phase is 2^12 (1/2 of sample), first sample is (*y*+2). For smaller values of initial phase it is between half distance between *y*+1 and *y*+2 on one side and *y*+2 at the other side. For values greater then 2^12 it is between sample y+2 on one side and the half distance between samples y+2 and y+3 at the other side (see Figure 158):

Confidential

Figure 158: Vertical pan and scan example




Confidential

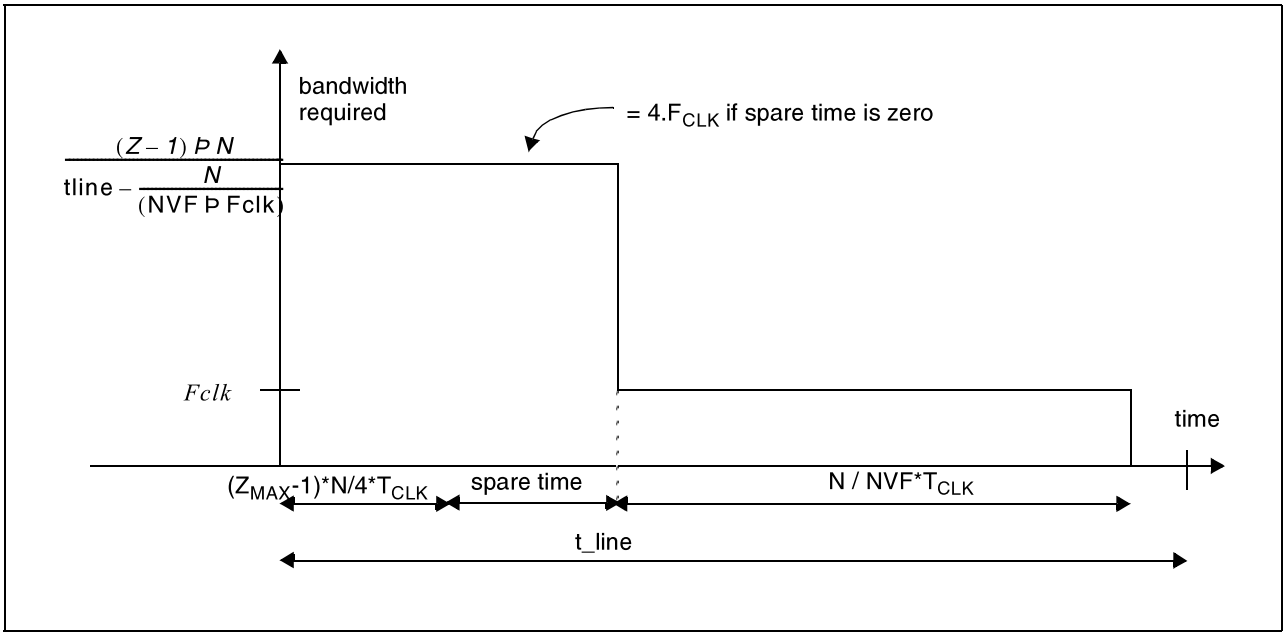
### 49.6.3 Bandwidth considerations

From the input buffers, luma and chroma samples are separated and sent to delay lines for vertical filtering (up and downsizing); see Figure 151 for block diagram. In HD, possible sources are 4:2:2 R, 4:2:0 MB and 4:2:2 MB formats, 1920 pixels per line maximum (960 for ID version). Each delay line is made of two 960 byte RAMs (two 480 byte RAMS for ID version). Pixels are grouped by four on each address as shown on Figure 160, that is, each delay line is made of two 240*32 bits RAMs (120*32 bits RAMS for ID version). Two RAMs are used to ping-pong write and read operations on a cycle basis, allowing maximum throughput.

#### 49.6.3.1 Operation of the vertical filter and associated delay lines

The vertical filter uses four delay lines and a fifth input directly provided by the STBus interface FIFOs to perform its five tap filter operation. Where zoom requires less than one line fetch per line generated (zoom in), the display can read the same line from memory several times. If zoom requires more than one line fetch per line generated (zoom out), the display fetches some lines while the vertical filter is not operating; filtering occurs while the last needed line is coming in from the STBus.

**Figure 159: Vertical filter input requirements**




**N**: number of pixels in the source video line (1920 max in HD, 960 max in ID)

**NVF**: number of vertical filters: 1 vertical filter

**Zmax**: maximum zoom out factor rounded to greater integer value (maximum number of lines to be fetched while generating one output line

**Fclk**, **Tclk**: clock frequency and clock period for pipeline (typical Fclk = 150 MHz)

**t_line** output (target) line duration: VTGW / Fpel (depends on the screen definition)

**VTGW**: VTG line width in pixels (full line width, typical 2200 for HD, 858 for NTSC)

**Fpel**: pixel frequency (typical 74.25 MHz in HD, 13.5 MHz in PAL or NTSC)

We need (Zmax-1)*N/4 cycles to load the delay lines without filtering operation. During this time, (Zmax-1) video lines are loaded in display.

If we have NVF vertical filters, we need N/NVF cycles to filter and generate a video line at the output. During this time one video line is read by the display (last line read is processed in real time).

This means that to be able to work in real time we need to respect:

$$\frac{N}{NVF} \cdot T_{ck} + \frac{Zmax - 1}{4} \cdot N \cdot T_{ck} \leq t_{line}$$

So with single VF (NVF =1), we have the maximum zoom out factor:

$$Z_{max} \leq \left( \frac{4.VTGW}{N} \cdot \frac{F_{clk}}{F_{pel}} \right) - 3$$

Confidential