R-Car Series, 3rd Generation

Adjusting Image Quality

Introduction

The document gives examples of images actually processed by the hardware IP modules (VSP: video signal processor and FDP: fine display processor), and describes the features and advantages of hardware modules for use in image quality adjustment and gives guidance in using the modules, including examples for reference of the setting of adjustment parameters.

Target Device

- Version 2.0 and 3.0 of the R-Car H3
- Version 1.x of the R-Car M3-W
- R-Car M3-W+
- R-Car M3-N
- R-Car E3
- R-Car D3
## Contents

1. **Video Signal Processor (VSP)** ................................................................. 3  
   1.1 Overview of the VSP .................................................................................. 3  
      1.1.1 Alpha Blending (by the RPF and BRU) .................................................. 3  
      1.1.2 Alpha Fading (by the RPF and BRU) ..................................................... 6  
      1.1.3 Rotation and Flipping (by the WPF) ...................................................... 7  
      1.1.4 Scaling Up or Down (by the UDS) ....................................................... 9  
      1.1.5 Sharpness Processing (by the SHP) ..................................................... 10  
      1.1.6 Super Resolution Processing (by the SRU) ......................................... 11  
   1.2 Guide to Using the VSP ............................................................................ 12  
      1.2.1 Image Partition for Use in VSPI Processing ........................................... 12  
      1.2.2 Swapping-related Settings ................................................................... 46  
      1.2.3 Recommended Use of Dithering .......................................................... 47  
      1.2.4 Alpha Blending .................................................................................... 48  
      1.2.5 Alpha Fading ...................................................................................... 56  

2. **Fine Display Processor (FDP)** ................................................................. 59  
   2.1 Overview of the FDP ................................................................................ 59  
      2.1.1 Motion Adaptive Interlaced-to-progressive (IP) Conversion .................... 59  
      2.1.2 Diagonal Line Interpolation .................................................................. 60  
   2.2 Guide to Using the FDP ........................................................................... 61  
      2.2.1 Swapping-Related Settings .................................................................. 61  
      2.2.2 Outline of FDP Parameters .................................................................. 61  
      2.2.3 Setting the Stillness and Motion Adjustment Tables ............................... 63  
      2.2.4 Setting the Register for the Detection of Comb Noise ........................... 64
1. **Video Signal Processor (VSP)**

1.1 **Overview of the VSP**

1.1.1 **Alpha Blending (by the RPF, BRU and BRS)**

The blend raster operation unit (BRU) handles alpha blending of up to five planes plus a virtual plane (internally created) each time processing to start proceeds.

The blend raster operation sub unit (BRS) handles alpha blending of up to two planes plus a virtual plane (internally created) each time processing to start proceeds.

The VSP uses the color keying function of the read pixel formatter (RPF) to composite non-rectangular objects.

The VSP uses the fixed alpha function of the RPF to handle alpha blending with the use of up to 8 bits of transparency information (set in a register). The alpha value for a single plane is uniform.

Using a format with pixel alpha, such as ARGB or an alpha plane, allows changes to 8-bit transparency information in pixel units.
(1) Color Keying

The VSP uses the color keying function to composite non-rectangular objects.

The figure below shows an example of blending with the use of color keying.

![Figure 1-1 Blending with the Use of Color Keying](image)

- The color key (blue) is made completely transparent.
- Blended image
- Processing by the RPF
- Processing by the BRU
(2) Pixel Alpha

Using pixel alpha (ARGB or alpha plane) allows change to alpha values in pixel units. This can be used to produce various effects, such as using semi-transparent alpha values to make jaggies at boundaries inconspicuous in the drawing of curves.

The figure below shows an example of alpha blending with the use of an alpha plane.

![Figure 1-2 Alpha Blending with the Use of Pixel Alpha](image-url)
1.1.2 Alpha Fading (by the RPF, BRU and BRS)

Using the multiply-alpha function of the RPF allows the fading in and out of images while pixel alpha (ARGB, alpha plane) is also in use.

The fade-alpha function is executed by using the relevant register to change the fading rate in frame units.

![Figure 1-3 Alpha Fading](image-url)
1.1.3 **Rotation and Flipping (by the WPF)**

The write pixel formatter (WPF) supports horizontal flipping, vertical flipping, 180-degree rotation, 90-degree rotation, 270-degree rotation, 90-degree rotation + horizontal flipping, and 90-degree rotation + vertical flipping.

![Rotation and Flipping (1)](image)

*Note: * Not applicable to the R-Car D3.
Figure 1-5  Rotation and Flipping (2)
### 1.1.4 Scaling Up or Down (by the UDS)

The up down scaler (UDS) scales images by any value from 1/16 to 16 times.

- The method of interpolation is selectable from among the following three types. The processing performance does not differ with the selected type of interpolation.
  - Nearest-neighbor
  - Bilinear
  - Bicubic (multi-tap filter)

- The UDS can use the filter for use in preprocessing to remove aliasing distortion due to scaling-down.

- In general, for the scaling of image data without alpha values, we recommend specifying the bicubic (multi-tap filter) method. This enables the scaling up or down of images with high picture quality.

- The bicubic method is not selectable in the scaling of images with alpha values (transparency information), so specify the bilinear method instead.

---

**Figure 1-6 Scaling Up or Down**

<table>
<thead>
<tr>
<th>Original image</th>
<th>Nearest-neighbor</th>
<th>Bilinear</th>
<th>Bicubic</th>
</tr>
</thead>
</table>

**Advantages and main usage**

- **Nearest-neighbor**: As pixels are copied without change, edges can be preserved. This is suitable for graphics such as characters with low resolution.

- **Bilinear**: As multiple taps are used, the pixels generated by interpolation are smoother than with the bilinear method. This is suitable for scaling natural images, etc. with high picture quality.

- **Bicubic**: As interpolation proceeds between adjacent pixels, jaggies are less prominent than in images interpolated by using the nearest-neighbor method. This is suitable for the scaling of images that include alpha values.

Note: * Not applicable to the R-Car D3.
1.1.5 Sharpness Processing (by the SHP)*

The sharpness module (SHP) improves the sense of focus by emphasizing edges. Moreover, it can also reduce aliasing distortion (such as of diagonal lines) by applying gradation processing to edges.

Figure 1-7 Sharpness Processing

Note: * Not applicable to the R-Car D3.
1.1.6 Super Resolution Processing (by the SRU)*

The super resolution unit (SRU) applies super resolution processing to images.

Super resolution processing spontaneously covers roof tiles other than edges and details of leaves and other objects, and parts with no edges, too.

Moreover, the SRU is capable of scale-up of an original image without change to a sense of resolution.

Note: * Not applicable to the R-Car D3.
1.2 Guide to Using the VSP

1.2.1 Image Partition for Use in VSPI Processing*

(1) Role of This Application Note with Regard to Image Partitioning

When the VSPI uses the UDS, SRU, or rotation and flipping function, a restriction applies to the width of images. For this reason, if the width of a frame of an image is greater than the maximum width, the VSPI must partition the image in the horizontal direction and handle the individual partitioned images every time it is started-up. These steps are hereinafter referred to as image partitioning.

To realize image partitioning, determine the method of partitioning images in accord with the specification in the section on “Image partition for VSPI processing” in section 32, Video Signal Processor (VSP2) of the hardware manual, and implement software programs to calculate the register settings for each image to be partitioned.

This application note describes the methods of partitioning images and gives examples of programs to calculate the register settings for each of the images partitioned by using the respective partition methods.

Note that an image partitioning function has been implemented in the VSP manager from Renesas. If you are not using the VSP manager, implement the software for image partitioning with reference to this application note.

Figure 1-9 Specifications on which the Hardware Manual Guides You and the Range Software is to Cover

Note: * Not applicable to the R-Car D3.
(2) Example of Implementing Software for Image Partitioning

Figure 1-10 shows the flow of implementing software for image partitioning, in which this application note guides you, as an example. The numbers enclosed in square brackets in the figure correspond to those given before the titles of passages (a) to (j) on subsequent pages.

![Flowchart of Implementing Software for Image Partitioning](image)

**Figure 1-10 Flow of Implementing Software for Image Partitioning**

Figure 1-11 illustrates the concept of implementing software for image partitioning. First of all, calculate the register settings to realize the desired function. This step is required in all processing by the VSP, regardless of whether image partitioning is in use. Determine the settings according to the descriptions in the hardware manual. After that, determine the basic width for division and the method of partitioning for each image according to the register settings common to the frame as shown in figure 1-11 ([3] in figure 1-10). In addition, use the register settings common to the frame, basic
size for division, method of partitioning, and calculation program to calculate the register settings for the size, address, clipping, and UDS phase, which differ with the image to be partitioned ([5-1] to [5-6] in figure 1-10). Note that image partitioning as described in passage (c) applies to an input for image to the WPF, as shown in figure 1-12.

**Figure 1-11  Concept of Image Partitioning**

<table>
<thead>
<tr>
<th>Register</th>
<th>Partition 0 (after the initial start-up)</th>
<th>Partition 1 (after the second start-up)</th>
<th>Partition 2 (after the third start-up)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size setting</td>
<td>Setting 0</td>
<td>Setting 1</td>
<td>Setting 2</td>
</tr>
<tr>
<td>Address setting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clipping size setting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDS phase setting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other registers</td>
<td>The setting for processing of the frame are passed on (the setting are common to the partitioned images).</td>
<td>The setting for processing of the frame are passed on (the setting are common to the partitioned images).</td>
<td>The setting for processing of the frame are passed on (the setting are common to the partitioned images).</td>
</tr>
</tbody>
</table>

**Figure 1-12  Position where Image Partitioning is Applied**
The width for input to the WPF (referred to as \( \text{h}_{\text{wpfin}} \)) shown in figure 1-12 represents the width of the image before rotation and flipping by the WPF. Note that the vertical size for output from the VSPI becomes the width for input to the WPF when any of the following settings is made: 90-degree rotation (VI6_WPF0_OUTFMT.ROT = 4), 90-degree rotation + vertical flipping (VI6_WPF0_OUTFMT.ROT = 5), 90-degree rotation + horizontal flipping (VI6_WPF0_OUTFMT.ROT = 6), or 270-degree rotation (VI6_WPF0_OUTFMT.ROT = 7).

- \( \text{h}_{\text{wpfin}} \) = Width of an output image from the VSPI (when VI6_WPF0_OUTFMT.ROT = 0 to 3)
- \( \text{h}_{\text{wpfin}} \) = Vertical size of an output image from the VSPI (when VI6_WPF0_OUTFMT.ROT = 4 to 7)

(a) [1] Checking whether Image Partitioning is Required

As described under “Necessity of image partition” in the section on “Image partition for VSPI processing” in section 32, Video Signal Processor (VSP2) of the hardware manual, the restriction on the maximum width shown in table 1-1 applies to processing by the UDS, SRU, and ROT (rotation and flipping module) of the VSPI. Therefore, if the width of a frame is greater than the maximum width shown in table 1-1, the frame needs to be partitioned in the horizontal direction (image partition) for processing. If the maximum width is not exceeded under any of the conditions listed in the table, image partitioning is not required. Figure 1-13 is a flowchart for checking whether image partitioning is required.

### Table 1-1 Maximum Widths in Processing by the UDS, SRU, and ROT Modules

<table>
<thead>
<tr>
<th>Condition under which the Maximum Width Must be Taken into Account</th>
<th>Maximum Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Horizetal scaling-up by the UDS</td>
<td>Width for input to the UDS</td>
</tr>
<tr>
<td>2 Horizontal scaling-down or no scaling by the UDS</td>
<td>Width for output from the UDS*</td>
</tr>
<tr>
<td>3 The SRU is in use.</td>
<td>Width for input to the SRU</td>
</tr>
<tr>
<td>4 Rotation or horizontal flipping is applied. (VI6_WPF0_OUTFMT.ROT &gt; 1)</td>
<td>Width for input to the WPF</td>
</tr>
</tbody>
</table>

Note: * This is the width for output from the UDS after horizontal clipping for output by the UDS according to the setting of VI6_UDS_CLIP_SIZE.
(b) **Confirmation of Restrictions when Image Partitioning is in Use**

As described under “Restriction when VSPI image partition is used” in the section on “Image partition for VSPI processing” in section 32, Video Signal Processor (VSP2) of the hardware manual, several restrictions apply to image partitioning by the VSPI. If image partitioning breaks any of the restrictions, change some parts of one round of processing by the VSPI.

For example: Image partitioning cannot be applied in cases where the VSPI uses the UDS and then the SRU with scaling-up by 2. In such a case, proceed with UDS processing in the first round of processing by the VSPI, and processing by the SRU with scaling-up by 2 in the second round.

---

**Figure 1-13  Flow of Checking whether Image Partitioning is Required**

- **Start**
  - **Is the UDS in use?**
    - Yes
      - Horizontal scaling-down or no scaling
    - No
  - **Is the image to be horizontally scaled up?**
    - Yes
      - **Width for input to the UDS > 304**
        - Yes: **Image partitioning is required.**
        - No: **Width for output from the UDS* > 304**
          - Yes: **Image partitioning is required.**
          - No: **Image partitioning is not required.**
  - **No**
    - **Is the SRU in use?**
      - Yes
        - **Width for input to the SRU > 288**
          - Yes: **Image partitioning is required.**
          - No: **Image partitioning is not required.**
      - No: **Is the image to be rotated or horizontally flipped?**
        - Yes
          - **Width for input to the WPF > 256**
            - Yes: **Image partitioning is required.**
            - No: **Image partitioning is not required.**
        - No: **Image partitioning is not required.**

Note: * This is the width for output from the UDS after horizontal clipping for output by the UDS according to the setting of VI6_UDS_CLIP_SIZE.
### (c) Classifying Calculation Programs and Determining the Basic Size of the Horizontal Divisions

Start by obtaining the basic horizontal division size \( \text{(div\_size)} \) from table 1-2. Partitioning the input image to the WPF in accord with the basic horizontal division size \( \text{(div\_size)} \) from table 1-2 leads to satisfaction of the restriction on the maximum width shown in table 1-1.

Next, classify the calculation programs. In implementation of software for image partitioning in which this application note guides you, the procedure differs according to whether any of modules by which processing involves scaling up or down is to be used. The calculation programs are classified into 4 groups based on the register settings as listed in table 1-2. Select one from among calculation programs 1 to 4 (passes (e) to (h)) in accord with the given register settings, and proceed with calculation of the register settings. Note that the number of register settings to be calculated differs with the type of calculation program. For example, as calculation programs 1 and 2 are for cases where the UDS is not used, the register settings related to the UDS are not required. In addition, the procedure of calculation for the source address register (passage (i)) and the destination address register (passage (j)) is common to all calculation programs. Note that the definitions of functions referred to in calculation programs are given in passage (k) by using the pseudo-code.

#### Table 1-2 Classification of Basic Horizontal Division Sizes and Calculation Programs by Settings for the SRU, UDS, and Rotation and Flipping Processing

<table>
<thead>
<tr>
<th>No.</th>
<th>SRU with no scaling</th>
<th>UDS</th>
<th>ROT</th>
<th>Order of Processing by the SRU and UDS</th>
<th>Calculation Program</th>
<th>Basic Horizontal Division Size (div_size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SRU with no scaling</td>
<td>Unused</td>
<td>0 to 7</td>
<td>—</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>SRU with scaling-up by 2</td>
<td>Unused</td>
<td>0 to 1</td>
<td>—</td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td>3</td>
<td>SRU with scaling-up by 2</td>
<td>Unused</td>
<td>2 to 7</td>
<td>—</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>SRU with no scaling</td>
<td>1/16 &lt; HSCL ≤ 4</td>
<td>0 to 7</td>
<td>The UDS is followed by the SRU.</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>SRU with no scaling</td>
<td>4 &lt; HSCL ≤ 16</td>
<td>0 to 7</td>
<td>The UDS is followed by the SRU.</td>
<td>3</td>
<td>128</td>
</tr>
<tr>
<td>6</td>
<td>SRU with no scaling</td>
<td>1 ≤ HSCL ≤ 16</td>
<td>2 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>7</td>
<td>SRU with no scaling</td>
<td>1 ≤ HSCL ≤ 2</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>8</td>
<td>SRU with no scaling</td>
<td>2 &lt; HSCL ≤ 4</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>3</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>SRU with no scaling</td>
<td>4 &lt; HSCL ≤ 8</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>3</td>
<td>1024</td>
</tr>
<tr>
<td>10</td>
<td>SRU with no scaling</td>
<td>8 &lt; HSCL ≤ 16</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>3</td>
<td>2048</td>
</tr>
<tr>
<td>11</td>
<td>SRU with no scaling</td>
<td>1/16 &lt; HSCL &lt; 1</td>
<td>0 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td>Image partitioning is prohibited.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Unused</td>
<td>1/16 &lt; HSCL &lt; 1</td>
<td>—</td>
<td>—</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>13</td>
<td>Unused</td>
<td>1 ≤ HSCL ≤ 2</td>
<td>0 to 1</td>
<td>—</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>14</td>
<td>Unused</td>
<td>2 &lt; HSCL ≤ 4</td>
<td>0 to 1</td>
<td>—</td>
<td>3</td>
<td>512</td>
</tr>
<tr>
<td>15</td>
<td>Unused</td>
<td>4 &lt; HSCL ≤ 8</td>
<td>0 to 1</td>
<td>—</td>
<td>3</td>
<td>1024</td>
</tr>
<tr>
<td>16</td>
<td>Unused</td>
<td>8 &lt; HSCL ≤ 16</td>
<td>0 to 1</td>
<td>—</td>
<td>3</td>
<td>2048</td>
</tr>
<tr>
<td>17</td>
<td>Unused</td>
<td>1 ≤ HSCL ≤ 16</td>
<td>2 to 7</td>
<td>—</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>18</td>
<td>SRU with scaling-up by 2</td>
<td>1/4 &lt; HSCL ≤ 1/2</td>
<td>0 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td>No.</td>
<td>SRU</td>
<td>UDS</td>
<td>ROT</td>
<td>Order of Processing by the SRU and UDS</td>
<td>Calculation Program</td>
<td>Basic Horizontal Division Size (div_size)</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------</td>
<td>------------------------</td>
<td>-----</td>
<td>----------------------------------------</td>
<td>---------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>19</td>
<td>SRU with scaling-up by 2</td>
<td>1/2 &lt; HSCL ≤ 1</td>
<td>0 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>20</td>
<td>SRU with scaling-up by 2</td>
<td>1 &lt; HSCL ≤ 16</td>
<td>2 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>21</td>
<td>SRU with scaling-up by 2</td>
<td>1 &lt; HSCL ≤ 2</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>22</td>
<td>SRU with scaling-up by 2</td>
<td>2 &lt; HSCL ≤ 4</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>23</td>
<td>SRU with scaling-up by 2</td>
<td>4 &lt; HSCL ≤ 8</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>24</td>
<td>SRU with scaling-up by 2</td>
<td>8 &lt; HSCL ≤ 16</td>
<td>0 to 1</td>
<td>The SRU is followed by the UDS.</td>
<td>4</td>
<td>2048</td>
</tr>
<tr>
<td>25</td>
<td>SRU with scaling-up by 2</td>
<td>1/16 &lt; HSCL ≤ 1/4</td>
<td>0 to 7</td>
<td>The SRU is followed by the UDS.</td>
<td></td>
<td>Image partitioning is prohibited.</td>
</tr>
<tr>
<td>26</td>
<td>SRU with scaling-up by 2</td>
<td>1/16 &lt; HSCL ≤ 1/4</td>
<td>0 to 7</td>
<td>The UDS is followed by the SRU.</td>
<td></td>
<td>Image partitioning is prohibited.</td>
</tr>
<tr>
<td>27</td>
<td>Unused</td>
<td>Unused</td>
<td>2 to 7</td>
<td>—</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>28</td>
<td>Unused</td>
<td>Unused</td>
<td>0 to 1</td>
<td>—</td>
<td>—</td>
<td>8190</td>
</tr>
</tbody>
</table>

The register settings corresponding to the parameters used in the table are as follows.

- **HSCL**: \( \frac{4096}{(VI6\_UDS\_SCALE.HMANT * 4096 + VI6\_UDS\_SCALE.FRAC)} \)
- **SRU with no scaling**: \((VI6\_SRU\_CTRL0.SRU\_MODE == 3'b000) || !VI6\_SRU\_CTRL0.SRU\_EN) && (VI6\_DPR\_SRU\_ROUTE != 63)\)
- **SRU with scaling-up by 2**: \((VI6\_SRU\_CTRL0.SRU\_MODE == 3'b100) && VI6\_SRU\_CTRL0.SRU\_EN) && (VI6\_DPR\_SRU\_ROUTE != 63)\)
- **Unused (SRU)**: \((VI6\_DPR\_SRU\_ROUTE == 63)\)
- **Unused (UDS)**: \((VI6\_DPR\_UDS\_ROUTE == 63)\)
- **ROT**: \(VI6\_WPFO\_OUTFMT.ROT\)

The conditions for Nos. 11, 25, and 26 in the table are cases where the result of [2] in figure 1-10 is “Yes”.

The condition for No. 28 in the table is the case where the result of [1] in figure 1-10 is “Image partitioning is not required”. 
(d) Determining the Method of Partitioning

Next, determine the method of partitioning images. In the example of implementation of software in which this application note provides guidance, input images to the WPF are horizontally partitioned in accord with the given div_size in table 1-2. Taking the efficiency of bus access for output into account, div_size is in powers of 2. At times, however, the width for input to the WPF may not be a multiple of div_size. In such cases, the partition method is classified into the four types (2 × 2) under the two classification conditions listed in table 1-3. Details on classification conditions 1 and 2 in the table are as follows. The reason why the change to the method of partitioning is required under each of classification conditions 1 and 2 is also given below.

- **Classification condition 1**
  - Methods of partitioning under each condition
    - When \((\text{ROT} \leq 1) \| (\text{ROT} \geq 4)) \| (\text{FCNL}==0)\)
      - The size of the partitioned image on the extreme right is a fraction of div_size.
    - When \((\text{ROT} == 2) \| (\text{ROT} == 3)) \&\& (\text{FCNL}==1)\)
      - The size of the partitioned image on the extreme left is a fraction of div_size.

  Note: \(\text{ROT} = \text{VI6\_WPFO\_OUTFMT.ROT}[2:0]\)
  \(\text{FCNL} = \text{VI6\_WPFO\_OUTFMT.FCNL}\)

  - Reason why the change to the method of partitioning is required
    The frame compression IP module for VSP (FCPV) handles compression processing in 16- or 32-pixel units horizontally from the left edge. For this reason, the size of the partitioned image on the extreme right of the output image must be a fraction (a partitioned image with a size that is not a multiple of 16 pixels) as shown in figure 1-14. When horizontal flipping is to be applied to an image, make the size of the partitioned image on the extreme left of the input image to the WPF before horizontal flipping the fractional portion, as shown in the lower example in figure 1-14. In figure 1-14, the upper example shows the partition method under the condition \((\text{ROT} \leq 1) \| (\text{ROT} \geq 4)) \| (\text{FCNL}==0)\), while the lower example shows that under the condition \((\text{ROT} == 2) \| (\text{ROT} == 3)) \&\& (\text{FCNL}==1)\).

- **Classification condition 2**
  - Methods of partitioning under each condition
    - When the width for input to the WPF (hwpfin) divided by div_size leaves at least div_size/2
      - Leave the width of the second to last partitioned image at div_size.
    - When the width for input to the WPF (hwpfin) divided by div_size leaves a value less than div_size/2
      - Make the width of the second to last partitioned image div_size/2.

  - Reason why the change to the method of partitioning is required
    As the calculation programs in which this application provides you with guidance do not support the condition where the width of a partitioned image is less than 4, an image is required to be partitioned such that the width of each of partitioned images is not less than 4. For this reason, when the fractional portion is narrower than div_size/2, make the width of the second to last partitioned image div_size/2 and make that of the last partitioned image the fractional portion. This avoids the width of the last partitioned image being less than 4.
Figure 1-14  Position of the Portion of the Partitioned Image with a Fractional Size in Images for Input and Output to and from the WPF

Table 1-3  Methods of Partitioning Images Depending on the ROT and FCNL Settings and the Width for Input to the WPF

<table>
<thead>
<tr>
<th>&lt;Classification Condition 1&gt; Register Setting*</th>
<th>&lt;Classification Condition 2&gt; Width for Input to the WPF (hwpfin)</th>
<th>Methods of Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>((ROT &lt;= 1)</td>
<td></td>
<td>(ROT &gt;= 4))</td>
</tr>
<tr>
<td></td>
<td>(hwpfin % div_size &lt;= 0)</td>
<td></td>
</tr>
<tr>
<td>((ROT == 2)</td>
<td></td>
<td>(ROT == 3)) &amp;&amp; (FCNL==1)</td>
</tr>
<tr>
<td></td>
<td>(hwpfin % div_size &lt;= 0)</td>
<td></td>
</tr>
</tbody>
</table>

Note:  * ROT = VI6_WPF0_OUTFMT.ROT [2:0]  
       FCNL = VI6_WPF0_OUTFMT.FCNL

Figures 1-15 to 1-18 show examples where images are partitioned into four portions.
1. Details of partition methods (A-1) and (A-2)

When the value of the ROT bits is 0, 1, or 4 to 7 or when near-lossless data compression (FCNL) is not to be applied to the output data, the image is partitioned in `div_size` units from the left edge of the input image to the WPF as shown in figure 1-15. Note that if the width in pixels of the partitioned image on the extreme right is less than `div_size/2`, the width of the second to last partitioned image becomes `div_size/2`, as shown in figure 1-16, to avoid violating the restriction on the minimum size (for example, at least four pixels in the horizontal direction for the UDS).

![Figure 1-15 Partition Method (A-1)](image1)

![Figure 1-16 Partition Method (A-2)](image2)
2. Details of partition methods (B-1) and (B-2)

When the value of the ROT bits is 2 or 3 and when near-lossless data compression (FCNL) is to be applied to the output data, the image after horizontal flipping in the WPF will be compressed with the use of FCNL. For this reason, the input image to the WPF is required to be partitioned in div_size units from the right edge instead of the left edge so that the output image after horizontal flipping is partitioned in the same way as in figures 1-15 and 1-16.
(e) [5-1] Calculating the Register Settings for Each Partitioned Image through Calculation Program 1

Calculation program 1 is used when the UDS and SRU are not in use but rotation or flipping is to be applied. Table 1-2 shows the details of the conditions under which calculation program 1 is used.

Figure 1-19 is a schematic view of the data path router (DPR) configuration when calculation program 1 is used. The focus of this figure is on the SHP and SRU with no scaling modules, which affect the calculation of overlapping areas. In this case, the SHP and SRU with no scaling serve as the MOD.

**Figure 1-19  Schematic View of DPR Configuration (for Calculation Program 1)**

Figure 1-20 illustrates the steps of obtaining overlapping areas of the source image from the positions where the input image to the WPF is partitioned, in calculation program 1. The steps of calculating the register settings for each partitioned image are described on the basis of figure 1-20 on the following pages.
Figure 1-20  Positions where the Input Image to the WPF is Partitioned and Overlapping Areas of the Source Image (with Calculation Program 1)

<Step 1> Obtaining the coordinates of the left and right edges of each partitioned image (partition) for input to the WPF

Obtain the horizontal coordinates of the left edge (dst_pos0) and of the right edge (dst_pos1) of each partition of the output image.

For example, when the width for input to the WPF (hwpfin) is 960 pixels and div_size = 256, the number of partitions will be 4 and the coordinates of the left and right edges of each partition are as follows. For details on the method of partitioning images, see passage (d).

\[\]

\[\]

\[\]

\[\]

\[\]

<Step 2> Obtaining the numbers of pixels in discontinuous lines at partition boundaries (mgn_1 and mgn_r) in the MOD

As described in the hardware manual, a discontinuous line at a partition boundary is generated in processing in the MOD (SRU with no scaling and SHP). mgn_1 and mgn_r are the numbers of pixels in discontinuous lines on the boundaries at the left and right edges of a partition, respectively. The number of pixels in a discontinuous line at a partition boundary generated by the SRU processing with no scaling is 1 pixel and the number generated by SHP...
processing is 4 pixels. Accordingly, mgn_l = mgn_r = 1 when the MOD exclusively includes the SRU with no scaling, mgn_l = mgn_r = 4 when the MOD exclusively includes the SHP, and mgn_l = mgn_r = 5 when the MOD includes both the SRU with no scaling and the SHP. Note that mgn_l = 0 for the partition on the extreme left and mgn_r = 0 for the partition on the extreme right.

Step 3> Obtaining the coordinate of the left edge (src_pos0) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the left edge (src_pos0) of each partition for input.

```plaintext
src_pos0_temp = dst_pos0 - mgn_l;
if ((src_pos0_temp % 2) && input_format_yuv422) {
    rpf_cor_l = 1;
} else {
    rpf_cor_l = 0;
}
src_pos0 = dst_pos0 - mgn_l - rpf_cor_l;
```

The variable input_format_yuv422 becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos0 is odd, the number of overlapping pixels to be read needs to be increased by one on the left side (horizontal coordinate of the left edge minus 1). This is why rpf_cor_l must be corrected.

Step 4> Obtaining the coordinate of the right edge (src_pos1) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the right edge (src_pos1) of each partition for input.

```plaintext
src_pos1_temp = dst_pos1 + mgn_r;
if (((src_pos1_temp - src_pos0 + 1) % 2) && input_format_yuv422) {
    rpf_cor_r = 1;
} else if ((VI6_RPF0_INFMT.CIPM == 1) && input_format_yuv422) {
    rpf_cor_r = 2;
} else {
    rpf_cor_r = 0;
}
src_pos1 = dst_pos1 + mgn_r + rpf_cor_r;
```

if ((dst_pos1 + mgn_r + rpf_cor_r) > (hsrc - 1)) {
    src_pos1 = hsrc - 1;
} else {
    src_pos1 = dst_pos1 + mgn_r + rpf_cor_r;
}
The variable input_format_yccbr42x becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos1 is even, the number of overlapping pixels to be read needs to be increased by one on the right side (horizontal coordinate of the right edge plus 1). This is why rpf_cor_r must be corrected.

The variable hsrc is the width for the input of a frame (before image partitioning). When the value obtained from src_pos1_sru + mgn_front_r + rpf_cor_r is greater than the horizontal coordinate of the right edge, hsrc will become hsrc – 1*, i.e. the upper limit of the horizontal coordinate of the right edge.

Note: * For example, when the number of pixels in the horizontal direction of a frame is 720 (hsrc = 720), the horizontal coordinate of the right edge will become 719.

<Step 5> Obtaining the register settings (other than addresses) with different values for each partition

Use the variables obtained in step 1 and subsequent steps to calculate the register settings for each partition in the ways described in table 1-4. For details on how to set the source and destination address registers, see passages (i) and (j), respectively.

### Table 1-4  Calculating the Register Settings through Calculation Program 1

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit Name</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPF0_SRC_BSIZE</td>
<td>BHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_RPF0_SRC_ESIZE</td>
<td>EHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_WPF0_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>mgn_l + rpf_cor_l</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>dst_pos1 – dst_pos0 + 1</td>
</tr>
</tbody>
</table>

(f) [5-2] Calculating the Register Settings for Each Partitioned Image through Calculation Program 2

Calculation program 2 is used when the SRU with scaling-up by 2 is in use but the UDS is not in use. Table 1-2 shows the details of the conditions under which calculation program 2 is used.

Figure 1-21 is a schematic view of the DPR configuration when calculation program 2 is used. The focus of this figure is on the SHP and SRU with scaling-up by 2 modules, which affect the calculation of overlapping areas. The modules placed prior to and after the SRU with scaling-up by 2 are referred to as MODfront and MODback, respectively. In this case, the SHP serves as the MODfront and MODback.

![Figure 1-21  Schematic View of DPR Configuration (for Calculation Program 2)](image)

Figure 1-22 illustrates the steps of obtaining overlapping areas of the source image from the positions where the input image to the WPF is partitioned, in calculation program 2. The steps of calculating the register settings for each partitioned image are described on the basis of figure 1-22 on the following pages.
Figure 1-22 Positions where the Input Image to the WPF is Partitioned and Overlapping Areas of the Source Image (with Calculation Program 2)

<Step 1> Obtaining the coordinates of the left and right edges of each partitioned image (partition) for input to the WPF
Obtain the horizontal coordinates of the left edge (dst_pos0) and of the right edge (dst_pos1) of each partition of the output image.

For example, when the width for input to the WPF ($h_{wpfin}$) is 960 pixels and div_size = 256, the number of partitions will be 4 and the coordinates of the left and right edges of each partition are as follows. For details on the method of partitioning images, see passage (d).

dst_pos0 [P0] = 0 dst_pos1 [P0] = 255
dst_pos0 [P1] = 256 dst_pos1 [P1] = 511
dst_pos0 [P2] = 512 dst_pos1 [P2] = 767
dst_pos0 [P3] = 768 dst_pos1 [P3] = 959
<Step 2> Obtaining the numbers of pixels in discontinuous lines at partition boundaries (mgn_back_l and mgn_back_r) in the MOD_back

As described in the hardware manual, a discontinuous line at a partition boundary is generated in processing in the MOD_back (SHP). mgn_back_l and mgn_back_r are the numbers of pixels in discontinuous lines on the boundaries at the left and right edges of a partition, respectively. The number of pixels in a discontinuous line at a partition boundary generated by SHP processing is 4 pixels. Accordingly, when the MOD_back includes the SHP, set mgn_back_l = mgn_back_r = 4. Meanwhile, when the MOD_back does not include the SHP, set mgn_back_l = mgn_back_r = 0. Note that mgn_back_l = 0 for the partition on the extreme left and mgn_back_r = 0 for the partition on the extreme right.

<Step 3> Obtaining pull_back_sru

Obtain pull_back_sru in the same way as in the pseudo-code below.

```c
if (dst_pos0 == 0) { /// For the partition on the extreme left
    pull_back_sru = 0;
} else { /// For partitions other than that on the extreme left
    pull_back_sru = (dst_pos0 - mgn_back_l % 2) ? 3 : 2;
}
```

<Step 4> Obtaining the coordinates of the left and right edges of each partition for input to the SRU

Use the functions described in passage (k) to obtain the coordinates of the left edge (src_pos0_sru) and the right edge (src_pos1_sru) of each partition for input to the SRU in the same way as in the pseudo-code below.

```c
src_pos0_sru = get_lpos_before_sru (dst_pos0 - mgn_back_l - pull_back_sru);
src_pos1_sru = get_rpos_before_sru (dst_pos1 + mgn_back_r);
```

<Step 5> Obtaining the numbers of pixels in discontinuous lines at partition boundaries (mgn_front_l and mgn_front_r) in the MOD_front

As described in the hardware manual, a discontinuous line at a partition boundary is generated in processing in the MOD_front (SHP). mgn_front_l and mgn_front_r are the numbers of pixels in discontinuous lines on the boundaries at the left and right edges of a partition, respectively.

mgn_front_l = mgn_front_r = 4 when the MOD_front includes the SHP and mgn_front_l = mgn_front_r = 0 when the MOD_front does not include the SHP. Note that mgn_front_l = 0 for the partition on the extreme left and mgn_front_r = 0 for the partition on the extreme right.

<Step 6> Obtaining the coordinate of the left edge (src_pos0) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the left edge (src_pos0) of each partition for input.

```c
src_pos0_temp = src_pos0_sru - mgn_front_l;
if ((src_pos0_temp % 2) && input_format_yCbcr42x) {
    rpf_cor_l = 1;
} else {
    rpf_cor_l = 0;
}
src_pos0 = src_pos0_sru - mgn_front_l - rpf_cor_l;
```
The variable input_format_ycbcr42x becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos0 is odd, the number of overlapping pixels to be read needs to be increased by one on the left side (horizontal coordinate of the left edge minus 1). This is why rpf_cor_l must be corrected.

<Step 7> Obtaining the coordinate of the right edge (src_pos1) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the right edge (src_pos1) of each partition for input.

```c
src_pos1_temp = src_pos1_sru + mgn_front_r;
if (((src_pos1_temp – src_pos0 + 1) % 2) && input_format_ycbcr42x) {
    rpf_cor_r = 1;
}
else if ((VI6_RPF0_INFMT.CIPM == 1) && input_format_ycbcr42x) {
    rpf_cor_r = 2;
}
else {
    rpf_cor_r = 0;
}
if ((src_pos1_sru + mgn_front_r + rpf_cor_r) > (hsrc – 1)) {
    src_pos1 = hsrc – 1;
}
else {
    src_pos1 = src_pos1_sru + mgn_front_r + rpf_cor_r;
}
```

The variable input_format_ycbcr42x becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos1 is even, the number of overlapping pixels to be read needs to be increased by one on the right side (horizontal coordinate of the right edge plus 1). This is why rpf_cor_r must be corrected.

The variable hsrc is the width for the input of a frame (before image partitioning). When the value obtained from src_pos1_sru + mgn_front_r + rpf_cor_r is greater than the horizontal coordinate of the right edge, hsrc will become hsrc – 1*, i.e. the upper limit of the horizontal coordinate of the right edge.

Note: * For example, when the number of pixels in the horizontal direction of a frame is 720 (hsrc = 720), the horizontal coordinate of the right edge will become 719.

<Step 8> Obtaining the register settings (other than addresses) with different values for each partition

Use the variables obtained in step 1 and subsequent steps to calculate the register settings for each partition in the ways described in table 1-5. For details on how to set the source and destination address registers, see passages (i) and (j), respectively.
Table 1-5  Calculating the Register Settings through Calculation Program 2

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit Name</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPF0_SRC_BSIZE</td>
<td>BHSIZE</td>
<td>src_pos1 − src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_RPF0_SRC_ESIZE</td>
<td>EHSIZE</td>
<td>src_pos1 − src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_WPF0_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>mgn_back_l + pull_back_sru +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 × (mgn_front_l + rpf_cor_l)</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>dst_pos1 − dst_pos0 + 1</td>
</tr>
</tbody>
</table>

(g)  [5-3] Calculating the Register Settings for Each Partitioned Image through Calculation Program 3

Calculation program 3 is used when the SRU with scaling-up by 2 is not in use but the UDS is (including the case where the SRU with no scaling is in use). Table 1-2 shows the details of the conditions under which calculation program 3 is used.

Figure 1-23 is a schematic view of the DPR configuration when calculation program 3 is used. The focus of this figure is on the SHP, SRU with no scaling, and UDS modules, which affect the calculation of overlapping areas. The modules placed prior to and after the UDS are referred to as MOD_front and MOD_back in lumps, respectively. In this case, the SHP and SRU with no scaling serve as the MOD_front and MOD_back.

Figure 1-24 illustrates the steps of obtaining overlapping areas of the source image from the positions where the input image to the WPF is partitioned, in calculation program 3. The steps of calculating the register settings for each partitioned image are described on the basis of figure 1-24 on the following pages.
<Step 1> Obtaining the coordinates of the left and right edges of each partitioned image (partition) for input to the WPF

Obtain the horizontal coordinates of the left edge (dst_pos0) and of the right edge (dst_pos1) of each partition of the output image.

For example, when the width for input to the WPF (h_wpfin) is 720 pixels and div_size = 256, the number of partitions will be 3 and the coordinates of the left and right edges of each partition are as follows. For details on the method of partitioning images, see passage (d).

dst_pos0 [p0] = 0  dst_pos1 [p0] = 255
dst_pos0 [p1] = 256  dst_pos1 [p1] = 511
dst_pos0 [p2] = 512  dst_pos1 [p2] = 719
As described in the hardware manual, a discontinuous line at a partition boundary is generated in processing in the MOD(back) (SHP and/or SRU with no scaling). mgn_back_l and mgn_back_r are the numbers of pixels in discontinuous lines on the boundaries at the left and right edges of a partition, respectively. The number of pixels in a discontinuous line at a partition boundary generated by SRU processing with no scaling is 1 pixel and the number generated by SHP processing is 4 pixels. Accordingly, mgn_back_l = mgn_back_r = 1 when the MOD(back) exclusively includes the SRU with no scaling, mgn_back_l = mgn_back_r = 4 when the MOD(back) exclusively includes the SHP, and mgn_back_l = mgn_back_r = 5 when the MOD(back) includes both the SRU with no scaling and the SHP. Note that mgn_back_l = 0 for the partition on the extreme left and mgn_back_r = 0 for the partition on the extreme right.

In addition, mgn_back_l must be corrected depending on the value of mant_pre by using the pseudo-code below. The combination of the settings of the UDS registers (horizontal scaling factor and bilinear or nearest neighbor interpolation characteristic control (BLADV)) determines the value of mant_pre, as shown in table 1-6.

```
switch (mant_pre) {
    case 1: mgn_back_l = mgn_back_l; break;
    case 2: mgn_back_l = (8 - mgn_back_l) % 2 + mgn_back_l; break;
    case 4: mgn_back_l = (8 - mgn_back_l) % 4 + mgn_back_l; break;
    default: ;
}
```

**Table 1-6  Relationship between the UDS Factor and mant_pre**

<table>
<thead>
<tr>
<th>VI6_UDS_SCALE.HMANT</th>
<th>VI6_UDS_CTRL.BLADV</th>
<th>mant_pre</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4 to 7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8 to 15</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2 to 3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4 to 7</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1, 8 to 15</td>
<td>1</td>
<td>Prohibited</td>
</tr>
</tbody>
</table>

<Step 3> Obtaining pull_back_uds

The specifiable pull_back_uds values are listed in table 1-7.

**Table 1-7  Relationship between the UDS Factor and pull_back_uds**

<table>
<thead>
<tr>
<th>Partitioning Position</th>
<th>HMANT</th>
<th>HFRAC</th>
<th>pull_back_uds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition on the extreme left (P0)</td>
<td>—</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>Partitions other than that on the extreme left (P1 to Pn-1)</td>
<td>0</td>
<td>256 to 511</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512 to 1023</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024 to 2047</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048 to 4095</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1 to 15</td>
<td>—</td>
<td>4</td>
</tr>
</tbody>
</table>
<Step 4> Obtaining the coordinates of the left and right edges, the start phase, and the end phase of each partition for input to UDS

Use the functions described in passage (k) to obtain the coordinates of the left edge (src_pos0_uds) and the right edge (src_pos1_uds), the start phase (start_phase), and the end phase (end_phase) of each partition for input to the UDS in the same way as in the pseudo-code below.

```c
if ((VI6_UDS_CTRL.AMD == 0) || (VI6_UDS_SCALE.HMANT != 0)) {
    phase_edge = 0;
}
//// (HMANT == 0) indicates horizontal scaling-up and a value from 256 to 4095 is specifiable for HFRAC.
else {
    phase_edge = (4096 - VI6_UDS_SCALE.HFRAC)/2;
}
src_pos0_uds = get_lpos_before_uds (dst_pos0 - mgn_back_l - pull_back_uds, phase_edge);
src_pos1_uds = get_rpos_before_uds (dst_pos1 + mgn_back_r, phase_edge);

if (left_partition) { // For partition on the extreme left (P0)
    start_phase = phase_edge;
} else { // For partitions other than that on the extreme left (P1 to P(N-1)), where N represents the number of partitions
    start_phase = get_start_phase (dst_pos0 - mgn_back_l - pull_back_uds, phase_edge);
}

if (right_partition) { // For partition on the extreme right (PN), where N represents the number of partitions
    end_phase = phase_edge;
} else { // For partitions other than that on the extreme right (P0 to P(N-2)), where N represents the number of partitions
    end_phase = 0;
}
```

<Step 5> Obtaining the numbers of pixels in discontinuous lines at partition boundaries (mgn_front_l and mgn_front_r) in the MOD_front

As described in the hardware manual, a discontinuous line at a partition boundary is generated in processing in the MOD_front (SHP and/or SRU with no scaling). mgn_front_l and mgn_front_r are the numbers of pixels in discontinuous lines on the boundaries at the left and right edges of a partition, respectively.

Set mgn_front_l = mgn_front_r = 1 when the MOD_front exclusively includes the SRU with no scaling, mgn_front_l = mgn_front_r = 4 when the MOD_front exclusively includes the SHP, and mgn_front_l = mgn_front_r = 5 when the MOD_front includes both the SRU with no scaling and the SHP. Note that mgn_front_l = 0 for the partition on the extreme left and mgn_front_r = 0 for the partition on the extreme right.
<Step 6> Obtaining the coordinate of the left edge (src_pos0) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the left edge (src_pos0) of each partition for input.

```c
src_pos0_temp = src_pos0_uds - mgn_front_l;
if ((src_pos0_temp % 2) && input_format_ycbcr42x) {
    rpf_cor_l = 1;
} else {
    rpf_cor_l = 0;
}
src_pos0 = src_pos0_uds - mgn_front_l - rpf_cor_l;
```

The variable `input_format_ycbcr42x` becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of `src_pos0` is odd, the number of overlapping pixels to be read needs to be increased by one on the left side (horizontal coordinate of the left edge minus 1). This is why `rpf_cor_l` must be corrected.

<Step 7> Obtaining the coordinate of the right edge (src_pos1) of each partition for input to the VSPI

Use the following pseudo-code to obtain the horizontal coordinate of the right edge (src_pos1) of each partition for input.

```c
src_pos1_temp = src_pos1_uds + mgn_front_r;
if (((src_pos1_temp - src_pos0 + 1) % 2) && input_format_ycbcr42x) {
    rpf_cor_r = 1;
} else if ((VI6_RPF0_INFMT.CIPM == 1) && input_format_ycbcr42x) {
    rpf_cor_r = 2;
} else {
    rpf_cor_r = 0;
}
if ((src_pos1_uds + mgn_front_r + rpf_cor_r) > (hsrc - 1)) {
    src_pos1 = hsrc - 1;
} else {
    src_pos1 = src_pos1_uds + mgn_front_r + rpf_cor_r;
}
```

The variable `input_format_ycbcr42x` becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units.
Therefore, if a value of src_pos1 is even, the number of overlapping pixels to be read needs to be increased by one on the right side (horizontal coordinate of the right edge plus 1). This is why rpf_cor_r must be corrected.

The variable hsrc is the width for the input of a frame (before image partitioning). When the value obtained from src_pos1_uds + mgn_front_r + rpf_cor_r is greater than the horizontal coordinate of the right edge, hsrc will become hsrc – 1*, i.e. the upper limit of the horizontal coordinate of the right edge.

Note: * For example, when the number of pixels in the horizontal direction of a frame is 720 (hsrc = 720), the horizontal coordinate of the right edge will become 719.

<Step 8> Obtaining the register settings (other than addresses) with different values for each partition

Use the variables obtained in step 1 and subsequent steps to calculate the register settings for each partition in the ways described in table 1-8. For details on how to set the source and destination address registers, see passages (i) and (j), respectively.

**Table 1-8 Calculating the Register Settings through Calculation Program 3**

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit Name</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPF0_SRC_BSIZE</td>
<td>BHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_RPF0_SRC_ESIZE</td>
<td>EHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_WPF0_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>mgn_back_l + pull_back_uds</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>dst_pos1 – dst_pos0 + 1</td>
</tr>
<tr>
<td>VI6_UDS_CTRL</td>
<td>AMDSLH</td>
<td>1</td>
</tr>
<tr>
<td>VI6_UDS_HPHASE</td>
<td>HSTP</td>
<td>start_phase</td>
</tr>
<tr>
<td></td>
<td>HEDP</td>
<td>end_phase</td>
</tr>
<tr>
<td>VI6_UDS_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>mgn_front_l + rpf_cor_l</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>src_pos1_uds – src_pos0_uds + 1</td>
</tr>
<tr>
<td>VI6_UDS_CLIP_SIZE</td>
<td>CL_HSIZE</td>
<td>VI6_WPF0_HSZCLIP.HCL_OFST +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VI6_WPF0_HSZCLIP.HCL_SIZE +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mgn_back_r</td>
</tr>
</tbody>
</table>

(h) [5-4] Calculating the Register Settings for Each Partitioned Image through Calculation Program 4

The following describes an example of the implementation of image partitioning in the case where the UDS is used after SRU processing with scaling-up by 2. Table 1-2 shows the conditions under which calculation program 4 is used.

Figure 1-25 is a schematic view of the DPR configuration when calculation program 4 is used. The focus of this figure is on the SRU with scaling-up by 2 and UDS modules, which affect the calculation of overlapping areas.

![Schematic View of DPR Configuration (for Calculation Program 4)](image)

Figure 1-25 Schematic View of DPR Configuration (for Calculation Program 4)

Figure 1-26 illustrates the steps of obtaining overlapping areas of the source image from the positions where the input image to the WPF is partitioned, in calculation program 4. The steps of calculating the register settings for each partitioned image are described on the basis of figure 1-26 on the following pages.
Figure 1-26  Positions where the Input Image to the WPF is Partitioned and Overlapping Areas of the Source Image (with Calculation Program 4)

<Step 1> Obtaining the coordinates of the left and right edges of each partitioned image (partition) for input to the WPF

Obtain the horizontal coordinates of the left edge (dst_pos0) and of the right edge (dst_pos1) of each partition of the output image.

For example, when the width for input to the WPF (hwpfin) is 720 pixels and div_size = 256, the number of partitions will be 3 and the coordinates of the left and right edges of each partition are as follows. For details on the method of partitioning images, see passage (d).

\[
\begin{align*}
dst\_pos0\ [p0] &= 0 & dst\_pos1\ [p0] &= 255 \\
dst\_pos0\ [p1] &= 256 & dst\_pos1\ [p1] &= 511 \\
dst\_pos0\ [p2] &= 512 & dst\_pos1\ [p2] &= 719
\end{align*}
\]
<Step 2> Obtaining pull_back_uds

Pull_back_uds can be obtained in the same way as in calculation program 3, so the results are as given in table 1-7.

<Step 3> Obtaining the coordinates of the left and right edges, the start phase, and the end phase of each partition for input to UDS

Use the functions described in passage (k) to obtain the coordinates of the left edge (src_pos0_uds) and the right edge (src_pos1_uds), the start phase (start_phase), and the end phase (end_phase) of each partition for input to the UDS in the same way as in the pseudo-code below.

if ((VI6_UDS_CTRL.AMD == 0) || (VI6_UDS_SCALE.HMANT != 0)) {
    phase_edge = 0;
}

/** (HMANT == 0) indicates horizontal scaling-up and a value from 256 to 4095 is specifiable for HFRAC.

else {
    phase_edge = (4096-VI6_UDS_SCALE.HFRAC)/2;
}

src_pos0_uds = get_lpos_before_uds (dst_pos0 - pull_back_uds, phase_edge);
src_pos1_uds = get_rpos_before_uds (dst_pos1, phase_edge);

if (left_partition) {  /** For partition on the extreme left (P0)
    start_phase = phase_edge;
}
else {  /** For partitions other than that on the extreme left (P1 to PN-1), where N represents the number of partitions
    start_phase = get_start_phase (dst_pos0 - mgn_back_l - pull_back_uds, phase_edge);
}

if (right_partition) {  /** For partition on the extreme right (PN-1), where N represents the number of partitions
    end_phase = phase_edge;
}
else {  /** For partitions other than that on the extreme right (P0 to PN-2), where N represents the number of partitions
    end_phase = 0;
}

<Step 4> Obtaining pull_back_sru

Obtain pull_back_sru in the same way as in the pseudo-code below.

if ( dst_pos0 == 0) {  /** Partition on the extreme left
    pull_back_sru = 0;
}
else {  /** Partitions other than that on the extreme left
    pull_back_sru = (dst_pos0 – mgn_back_l % 2) ? 3 : 2;
}
<Step 5> Obtaining the coordinates of the left and right edges of each partitioned image (partition) for input to the SRU
Use the functions described in passage (k) to obtain the coordinates of the left edge (src_pos0_sru) and the right edge (src_pos1_sru) of each partition for input to the SRU in the same way as in the pseudo-code below.

\[
\begin{align*}
\text{src_pos0_sru} &= \text{get_lpos_before_sru}(\text{src_pos0_uds} - \text{pull_back_sru}) ; \\
\text{src_pos1_sru} &= \text{get_rpos_before_sru}(\text{src_pos1_uds}) ; \\
\end{align*}
\]

<Step 6> Obtaining the coordinate of the left edge (src_pos0) of each partition for input to the VSPI
Use the following pseudo-code to obtain the horizontal coordinate of the left edge (src_pos0) of each partition for input.

\[
\begin{align*}
\text{if } \left( (\text{src_pos0_sru} \% 2) \land \land \text{input_format_ycbcr42x} \right) \{ \\
\text{rpf_cor_l} = 1 ; \\
\} \text{ else } \{ \\
\text{rpf_cor_l} = 0 ; \\
\} \\
\text{src_pos0} = \text{src_pos0_sru} - \text{rpf_cor_l} ; \\
\end{align*}
\]

The variable input_format_ycbcr42x becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos0 is odd, the number of overlapping pixels to be read needs to be increased by one on the left side (horizontal coordinate of the left edge minus 1). This is why rpf_cor_l must be corrected.

<Step 7> Obtaining the coordinate of the right edge (src_pos1) of each partition for input to the VSPI
Use the following pseudo-code to obtain the horizontal coordinate of the right edge (src_pos1) of each partition for input.

\[
\begin{align*}
\text{if } \left( ((\text{src_pos1_sru} - \text{src_pos0} + 1) \% 2) \land \land \text{input_format_ycbcr42x} \right) \{ \\
\text{rpf_cor_r} = 1 ; \\
\} \text{ else if } ((\text{VI6_RPF0_INFMT.CIPM} == 1) \land \land \text{input_format_ycbcr42x}) \{ \\
\text{rpf_cor_r} = 2 ; \\
\} \text{ else } \{ \\
\text{rpf_cor_r} = 0 ; \\
\} \\
\text{if } ((\text{src_pos1_sru} + \text{rpf_cor_r}) > (\text{h_src} - 1)) \{ \\
\text{src_pos1} = \text{h_src} - 1 ; \\
\} \text{ else } \{ \\
\end{align*}
\]
src_pos1 = src_pos1_sru + rpf_cor_r;
}

The variable input_format_ycbcr42x becomes 1 when YCbCr420 or YCbCr422 is in use as the input format and 0 with other formats. With YCbCr420 or YCbCr422 in use as the input format, the VSPI handles the width in 2-pixel units. Therefore, if a value of src_pos1 is even, the number of overlapping pixels to be read needs to be increased by one on the right side (horizontal coordinate of the right edge plus 1). This is why rpf_cor_r must be corrected.

The variable hsrc is the width for the input of a frame (before image partitioning). When the value obtained from src_pos1_sru + rpf_cor_r is greater than the horizontal coordinate of the right edge, hsrc will become hsrc – 1*, i.e. the upper limit of the horizontal coordinate of the right edge.

Note: * For example, when the number of pixels in the horizontal direction of a frame is 720 (hsrc = 720), the horizontal coordinate of the right edge will become 719.

<Step 8> Obtaining the register settings with different values for each partition

Use the variables obtained in step 1 and subsequent steps to calculate the register settings for each partition in the ways described in table 1-9. For details on how to set the source and destination address registers, see passages (i) and (j), respectively.

Table 1-9  Calculating the Register Settings through Calculation Program 4

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit Name</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPF0_SRC_BSIZE</td>
<td>BHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_RPF0_SRC_ESIZE</td>
<td>EHSIZE</td>
<td>src_pos1 – src_pos0 + 1</td>
</tr>
<tr>
<td>VI6_WPF0_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>pull_back_uds</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>dst_pos1 – dst_pos0 + 1</td>
</tr>
<tr>
<td>VI6_UDS_CTRL</td>
<td>AMDSLH</td>
<td>1</td>
</tr>
<tr>
<td>VI6_UDS_HPHASE</td>
<td>HSTP</td>
<td>start_phase</td>
</tr>
<tr>
<td></td>
<td>HEDP</td>
<td>end_phase</td>
</tr>
<tr>
<td>VI6_UDS_HSZCLIP</td>
<td>HCEN</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HCL_OFST</td>
<td>pull_back_sru + rpf_cor_l × 2</td>
</tr>
<tr>
<td></td>
<td>HCL_SIZE</td>
<td>src_pos1_uds – src_pos0_uds + 1</td>
</tr>
<tr>
<td>VI6_UDS_CLIP_SIZE</td>
<td>CL_HSIZE</td>
<td>VI6_WPF0_HSZCLIP.HCL_OFST +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VI6_WPF0_HSZCLIP.HCL_SIZE</td>
</tr>
</tbody>
</table>
(i) [5-5] Setting the Source Address Registers (VI6_RPF0_SRCM_ADDR_Y/C0/C1/A1) for Each Partition

Calculate the upper-left address (PHA[Pa]) for the n-th partition. If we take the n-th partition as \( P_n \), the upper-left address of a frame as FHA (frame header address), the coordinate of the left edge of individual partitions in the input image to the VSPI as src_pos0 [Pn] (as calculated in passages (e) to (h)), and the number of bytes per pixel as BPP, PHA[Pa] can be calculated from the following formula.

\[
PHA[Pa] = FHA + \text{src\_pos0}[Pn] \times \text{BPP}
\]

PHA [Pn] is the set of values to be set in the source address registers (VI6_RPF0_SRCM_ADDR_Y/C0/C1/A1).

Figure 1-27 Input Image to the VSPI and Partition Header Addresses (PHA) for Each Partition
(j) Setting the Destination Address Registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1) for Each Partition

Follow the procedure below to obtain the settings of the destination address registers for each of the partitions.

- Cases where 0 <= VI6_WPF0_OUTFMT.ROT <= 3

**<Step 1> Calculating the horizontal coordinate of the left edge (dst_pos’) of each partition in the output image from the WPF**

Calculate the horizontal coordinate of the left edge (dst_pos’[Pn]) of each partition in the output image from the WPF. Figure 1-28 shows details of how the coordinates in the output image from the WPF depend on the setting of VI6_WPF0_OUTFMT.ROT for rotation and flipping. dst_pos’[Pn] in the figure represents the horizontal coordinate of partition n in the output image from the WPF, and Hwpfin[Pn] represents the width of each partition (VI6_WPF0_HISZCLIP.HCL_SIZE) at the stage of input to the WPF, as was calculated in passages (e) to (h). In this case, the horizontal coordinates are calculated on the assumption that that of the left edge of the frame is 0. When ROT = 0 or 1, the values of dst_pos’[Pn] and dst_pos0[Pn] are the same. When ROT = 2 or 3, the images are horizontally flipped, so the order of partitions in the horizontal direction becomes reversed. Therefore, calculate dst_pos’[Pn] based on Hwpfin[Pn] as calculated in passages (e) to (h).

**<Step 2> Obtaining the upper-left address (PHA[Pn]) for the n-th partition**

Calculate the upper-left address (PHA[Pn]) for the n-th partition. If we take the n-th partition as Pn, the upper-left address of a frame as FHA (frame header address), the coordinate of the left edge of individual partitions in the output image from the WPF as dst_pos’[Pn], and the number of bytes per pixel as BPP, PHA[Pn] can be calculated from the following formula.

\[
PHA[Pn] = FHA + dst_pos’[Pn] \times BPP
\]

**<Step 3> Calculating the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1) from PHA[Pn]**

The relationship between the upper-left address for each partition PHA[Pn] and the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1) is the same as that between the FHA and the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1), which is described in the section on "WPFn Destination Y/RGB Address Registers (VI6_WPFn_DSTM_ADDR_Y)" in section 32, Video Signal Processor (VSP2), of the hardware manual. Refer to the hardware manual when calculating the settings.
Cases where 4 <= VI6_WPF0_OUTFMT.ROT <= 7

Step 1: Calculating the vertical coordinate of the upper edge (dst_pos') of each partition in the output image from the WPF.

Figure 1-29 shows details of how the coordinates in the output image from the WPF depend on the setting of VI6_WPF0_OUTFMT.ROT for rotation and flipping. dst_pos'[P_n] in the figure represents the vertical coordinate of partition n in the output image from the WPF, and H_wpin [P_n] represents the width of each partition in the input image to the WPF, as was calculated in passages (e) to (h). In this case, the vertical coordinates are calculated on the assumption that the upper edge of the frame is 0. When ROT = 4 or 6, the values of dst_pos'[P_n] and dst_pos0 [P_n] are the same. When ROT = 5 or 7, the images are vertically flipped, so the order of partitions in the vertical direction becomes reversed. Therefore, calculate dst_pos'[P_n] based on H_wpin [P_n] as calculated in passages (e) to (h).

Step 2: Obtaining the upper-left address (PHA[P_n]) for the n-th partition.

Calculate the upper-left address (PHA[P_n]) for the n-th partition. If we take the n-th partition as P_n, the upper-left address of a frame as FHA (frame header address), the coordinate of the upper edge of individual partitions in the output image from the WPF as dst_pos'[P_n], and the setting for the stride as S, PHA[P_n] can be calculated from the following formula.

\[ \text{PHA}[P_n] = \text{FHA} + \text{dst_pos}'[P_n] \times S \]
<Step 3> Calculating the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1) from PHA[\(P_n\)]

The relationship between the upper-left address for each partition PHA[\(P_n\)] and the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1) is the same as that between the FHA and the settings of the destination address registers (VI6_WPF0_DSTM_ADDR_Y/C0/C1), which is described in the section on “WPFn Destination Y/RGB Address Registers (VI6_WPFn_DSTM_ADDR_Y)” in section 32, Video Signal Processor (VSP2), of the hardware manual. Refer to the hardware manual when calculating the settings.

![Diagram](image)

**Figure 1-29** Coordinates of the Right Edges and Partition Header Addresses (PHA) for Each Partition in the Output Image (when ROT = 4 to 7)

(k) Definitions of Functions

```c
#define PEL_LEFT    0
#define PEL_RIGHT   1
#define HSTP_LEFT   2
```

```
../../../../
/// get_lpos_before_uds ///
../../../../
```

```c
int get_lpos_before_uds (int pos_after_uds, int hstp_ini)
```
```c
{  
    return get_boundary_param (pos_after_uds, int hstp_ini, PEL_LEFT);
}

////////////////////////////////////////
/// get_rpos_before_uds ///
////////////////////////////////////////

int get_rpos_before_uds (int pos_after_uds, int hstp_ini)
{
    return get_boundary_param (pos_after_uds, int hstp_ini, PEL_RIGHT);
}

////////////////////////////////////////
/// get_start_phase ///
////////////////////////////////////////

int get_start_phase (int pos_after_uds, int hstp_ini)
{
    return get_boundary_param (pos_after_uds, int hstp_ini, HSTP_LEFT);
}

////////////////////////////////////////
/// get_boundary_param ///
////////////////////////////////////////

int get_boundary_param (int pos_after_uds, int hstp_ini, int param_sel)
{
    int mant_pre;
    int prefilt_term;
    int prefilt_outpos;
    int phase_residual;
    int rt_val;
    int in_posx_l, in_posx_r, next_phase;
    int mant = VI6_UDS_SCALE.HMANT
    int frac = VI6_UDS_SCALE.HFRAC

    int alpha     = 4096 * mant + frac;
    int alpha_num = alpha * pos_after_uds;

    prefilt_term  = 4096 * mant_pre;
    prefilt_outpos = (alpha_num – hstp_ini * mant_pre) / prefilt_term;
    phase_residual = (alpha_num – hstp_ini * mant_pre) % prefilt_term;
}```
in_posx_l = mant_pre * (prefilt_outpos + (phase_residual? 1:0));

in_posx_r = mant_pre * (prefilt_outpos + 2) + mant_pre/2;

next_phase = phase_residual? (4096 - phase_residual/mant_pre): 0;

switch (param_sel) {
    case PEL_LEFT: rt_val = in_posx_l; break;
    case PEL_RIGHT: rt_val = in_posx_r; break;
    case HSTP_LEFT: rt_val = next_phase; break;
    default : ;
}

return rt_val;
}
1.2.2 Swapping-related Settings

The following describes how to set the swapping-related registers of the VSP. The VSP can set up swapping of data for both input and output. The settings to consistently arrange data in external memory as little endian are given in table 1-10. The settings vary according to whether data are to be handled in byte units (α plane, YCbCr, RGB332, CLUT-DATA, etc.), 2-byte units (RGB565 or ARGB1555, etc.), or 4-byte units (HGO or HGT histogram, display list, ARGB, etc.).

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Data Type</th>
<th>Data Unit</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_DSWAP</td>
<td>α plane</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
<tr>
<td>(A_LLS, A_LWS, A_WDS, A_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_RPFn_DSWAP</td>
<td>RGB332, CLUT-DATA (1 byte/pixel)</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
<tr>
<td>(P_LLS, P_LWS, P_WDS, P_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_RPFn_DSWAP</td>
<td>RGB565, RGB1555 (2 bytes/pixel)</td>
<td>2 bytes</td>
<td>4'b1110</td>
</tr>
<tr>
<td>(P_LLS, P_LWS, P_WDS, P_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_RPFn_DSWAP</td>
<td>RGB888 with stuffing (3 bytes/pixel)</td>
<td>1 byte*</td>
<td>4'b1111</td>
</tr>
<tr>
<td>(P_LLS, P_LWS, P_WDS, P_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_WPFn_DSWAP</td>
<td>RGB888 (4 bytes/pixel), RGB888 (4byte/pix)</td>
<td>4 bytes</td>
<td>4'b1100</td>
</tr>
<tr>
<td>(P_LLS, P_LWS, P_WDS, P_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_DL_SWAPn</td>
<td>YCbCr</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
<tr>
<td>(LWS, WDS, BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_HGO_HSWAP</td>
<td>HGO histogram data</td>
<td>4 bytes</td>
<td>3'b100</td>
</tr>
<tr>
<td>(H_LWS, H_WDS, H_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VI6_HGT_HSWAP</td>
<td>HGT histogram data</td>
<td>4 bytes</td>
<td>3'b100</td>
</tr>
<tr>
<td>(H_LWS, H_WDS, H_BTS)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: * Handling types with 3-byte units of data is not directly supported. Therefore, if RGB888 with stuffing (3 bytes/pixel) is to be used, handle the given data as a type for which data are handled in byte units and specify 4'b1111 as the setting to do so.

Note that when the VSP is to read FCNL data, follow the guide to the frame compression processor (FCP) in the hardware manual for the settings of VI6_RPFn_DSWAP.P_LLS, P_LWS, P_WDS, and P_BTS.

When the VSP is to output FCNL data, follow the guide to the frame compression processor (FCP) in the hardware manual for the settings of VI6_WPFn_DSWAP.P_LLS, P_LWS, P_WDS, and P_BTS.
1.2.3 Recommended Use of Dithering

This section describes the recommended use of dithering. The VSP has a data path of 24 bits/pixel (RGB888 when the RGB format is in use), and proceeds with dithering instead of simple quantization when the bit depth of the display panel is no greater than RGB666. The application of filtering (by the VSPI) such as for use with a scaler or of color adjustment processing (by the VSPI or VSPB) after dithering may lead to deterioration of the image quality. Therefore, we recommend the application of dithering in the VSPD, which is placed immediately before display, or in the WPF of the VSPDL. Note that if you wish to exclusively apply dithering to moving pictures or video in blending graphics and natural images, do so to individual input planes in the BRU or in the blend raster operation sub-unit (BRS). In such cases, also apply the dithering in a position as near as possible to the display so that dithering proceeds after filtering or color adjustment processing. Figure 1-30 shows the positions where dithering is applied in the VSPB and VSPD. In addition, two modes are selectable for dithering. We recommend the use of mode A, which produces higher quality than mode B (mode A enable/disable bits: VI6_WPF0_OUTFMT.ODE, VI6_BRU_INCTRL.ODEn (n = 0 to 4), VI6_BRS_INCTRL.ODEn (n = 0, 1)).

![Diagram of VSPB and VSPD](image)

**Figure 1-30** Positions where Dithering is Applied in the VSPB and VSPD
1.2.4 Alpha Blending

This section describes how to set up alpha blending. The following will guide you in setting the alpha-blending-related registers of the RPF, WPF, and BRU. Those of the BRS can be set in the same way as those of the BRU.

(1) Details of Equations for Alpha Blending

The following shows equations for alpha blending.

\[ C_s = \left((1 - \alpha_s) \times a_d \times C_d + \alpha_s \times C_s\right) / \alpha_o \quad \text{(equation 1-1)} \]
\[ \alpha_o = (1 - \alpha_s) \times a_d + \alpha_s \quad \text{(equation 1-2)} \]

\( \alpha_s \): Transparency information \( \alpha \) of a source (upper plane)*
\( C_s \): RGB color component of a source (upper plane)
\( \alpha_d \): Transparency information \( \alpha \) of a destination (lower plane)*
\( C_d \): RGB color component of a destination (lower plane)
\( \alpha_s \): Transparency information \( \alpha \) of a plane to be blended*
\( C_s \): RGB color component of a plane to be blended

Note: * Transparency (0 = transparent, 1 = opaque)

These correspond to the Porter-Duff source over destination equations, that is, the \( \alpha \) values of a plane for blending obtained by using the \( \alpha \) values of the source and destination for calculation are passed to the subsequent blending. This blending method is hereinafter referred to as \( \alpha \)-inherited alpha blending.

Images for which color component data have been multiplied by transparency information \( \alpha \) in advance are referred to as premultiplied \( \alpha \) images. Meanwhile, those with the data not multiplied by \( \alpha \) are referred to as straight \( \alpha \) images. The equations below are for \( \alpha \)-inherited alpha blending with the use of premultiplied \( \alpha \).

\[ C_o' = (1 - \alpha_s) \times C_d' + \alpha_s \times C_s' \quad \text{(equation 1-3)} \]

\( C_s' = \alpha_s \times C_s \): Premultiplied \( \alpha \) images of a source (upper plane)
\( C_d' = \alpha_d \times C_d \): Premultiplied \( \alpha \) images of a destination (lower plane)
\( C_o' = \alpha_o \times C_o \): Premultiplied \( \alpha \) images of a plane to be blended

In \( \alpha \)-inherited alpha blending, the data flow in the VSP differs with the type of image to be stored in memory, i.e., whether it is a premultiplied \( \alpha \) image or straight \( \alpha \) image. The case where straight \( \alpha \) images are to be stored in memory requires the following steps; the RPF converts the straight \( \alpha \) images to premultiplied \( \alpha \) images, and after blending, the divider for normalization (DIV unit) of the BRU converts the premultiplied \( \alpha \) images back to straight \( \alpha \) images.

In blending when the DST (lower plane) is opaque (\( \alpha_d = 1 \)), the blended plane is also opaque (\( \alpha_o = 1 \)). The equations for use in alpha blending in this case are shown below. These equations are equivalent to equations 1-1 and 1-2 except that 1 is substituted for \( \alpha_d \). Alpha blending with the use of equation 1-4 is referred to as normal alpha blending.

\[ C_o = (1 - \alpha_s) \times C_d + \alpha_s \times C_s \quad \text{(equation 1-4)} \]
\[ \alpha_o = 1 \]
Table 1-11 lists the equations and the passages where the register settings are described for use in the three types of alpha blending. Though the descriptions here are on the assumption of RGB color data being in use, normal alpha blending is also available for the YCbCr format with the use of the same register settings.

Table 1-11  Equation and Related Passage for Each Type of Alpha Blending

<table>
<thead>
<tr>
<th>Blending Type</th>
<th>I/O Image of the VSP</th>
<th>Related Passage</th>
<th>Equation for Blending</th>
</tr>
</thead>
<tbody>
<tr>
<td>α-inherited alpha blending</td>
<td>Straight α image</td>
<td>(2)</td>
<td>$C_o = ((1 - α_s) \times α_d \times C_d + \alpha_s x C_s) / \alpha_o$ (equation 1-1)</td>
</tr>
<tr>
<td>(Porter-Duff source over destination)</td>
<td>Premultiplied α image</td>
<td>(3)</td>
<td>$C_o' = (1 - \alpha_s) \times C_d' + \alpha_s x C_s' \text{ (equation 1-3)}$</td>
</tr>
<tr>
<td>Normal alpha blending</td>
<td>Straight α image</td>
<td>(4)</td>
<td>$C_o = (1 - \alpha_s) \times C_d + \alpha_s x C_s \text{ (equation 1-4)}$</td>
</tr>
</tbody>
</table>

Figure 1-31 shows the functions and registers related to alpha blending and processing by the respective units. The VSPB (VSPB, VSPBC, VSPBD), VSPD, and VSPDL support alpha blending. As the RPF supports step (1), Converting the straight α images to premultiplied α images in the figure, it’s available in the VSPI as well as the VSPB (VSPB, VSPBC, VSPBD), VSPD, and VSPDL.
(2) Setting Up Alpha Blending with an $\alpha$ Value Inherited (for the Input and Output of Straight $\alpha$ Images by the VSP)

Figure 1-32 shows the data flow and the corresponding equations in the case where blend/ROP unit A is used in alpha blending with an $\alpha$ value inherited (for the input and output of straight $\alpha$ images by the VSP). Table 1-12 shows the settings of RPF and BRU registers to realize $\alpha$-inherited alpha blending by using figure 1-32 as an example. Registers for use in blend/ROP units B to E can be set in the same way as those in blend/ROP unit A.

![Diagram](image-url)

**Figure 1-32** Data Flow and the Corresponding Equations in $\alpha$-Inherited Alpha Blending (for the Input and Output of Straight $\alpha$ Images by the VSP)

- $\alpha_s$: $\alpha$ value of the source for unit A
- $C_s$: RGB color component data of the source for unit A
- $C_s'$: RGB color component data of the source for unit A (premultiplied $\alpha$ images)
- $\alpha_d$: $\alpha$ value of the destination for unit A
- $C_d$: RGB color component data of the destination for unit A
- $C_d'$: RGB color component data of the destination for unit A (premultiplied $\alpha$ images)
- $\alpha_o$: $\alpha$ value for output from unit A
- $C_o'$: RGB color component data for output from unit A (premultiplied $\alpha$ images)
- $\alpha_r$: $\alpha$ value for output from the BRU
- $C_r'$: RGB color component data for output from the BRU (premultiplied $\alpha$ images)
## Table 1-12: Settings of the RPF and BRU in α-Inherited Alpha Blending (for the Input and Output of Straight α Images by the VSP)

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULT_ALPH H</td>
<td>A_MMD</td>
<td>0</td>
<td>The multiply-alpha unit does not convert α data.</td>
</tr>
<tr>
<td></td>
<td>P_MMD [1:0]</td>
<td>2</td>
<td>The multiply-alpha unit multiplies color component data by pixel α.</td>
</tr>
<tr>
<td></td>
<td>ALPH_RATIO [7:0]</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
<tr>
<td>VI6_BRU_INCTRL</td>
<td>NRM</td>
<td>1</td>
<td>The divider converts premultiplied α images to straight α images.</td>
</tr>
<tr>
<td>VI6_BRUA_CTRL</td>
<td>RBC</td>
<td>1</td>
<td>Blending instead of ROP is selected as the type of operation of blend/ROP unit A.</td>
</tr>
<tr>
<td>VI6_BRUA_BLD</td>
<td>CBES</td>
<td>0</td>
<td>The following is selected as the blending expression for the color component data.</td>
</tr>
<tr>
<td></td>
<td>CCMDX [2:0]</td>
<td>3</td>
<td>1 – αs is selected as the value of coefficient CX.</td>
</tr>
<tr>
<td></td>
<td>CCMGY [2:0]</td>
<td>4</td>
<td>COEFY is selected as the value of coefficient CY.</td>
</tr>
<tr>
<td></td>
<td>ABES</td>
<td>0</td>
<td>The following is selected as the blending expression with the alpha data.</td>
</tr>
<tr>
<td></td>
<td>ACMDX [2:0]</td>
<td>3</td>
<td>1 – αs is selected as the value of coefficient AX.</td>
</tr>
<tr>
<td></td>
<td>ACMGY [2:0]</td>
<td>4</td>
<td>COEFY is selected as the value of coefficient AY.</td>
</tr>
<tr>
<td></td>
<td>COEFX [7:0]</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
<tr>
<td></td>
<td>COEFY [7:0]</td>
<td>255</td>
<td>255 (100%) is selected for the COEFY bits.</td>
</tr>
<tr>
<td>VI6_WPFn_OUTFMT</td>
<td>WRFMT [6:0]</td>
<td>19*</td>
<td>Select the ARGB format, including padding.</td>
</tr>
<tr>
<td></td>
<td>PXA</td>
<td>1</td>
<td>The α value, the result of blending output from the DPR is to be stored in the padding field.</td>
</tr>
<tr>
<td></td>
<td>PDV [7:0]</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
</tbody>
</table>

Note: * In addition to 19, any value used for selection of the ARGB format with the padding field is settable.
(3) Setting Up Alpha Blending with an $\alpha$ Value Inherited (for the Input and Output of Premultiplied $\alpha$ Images by the VSP)

Figure 1-33 shows the data flow and the corresponding equations in the case where blend/ROP unit A is used in alpha blending with an $\alpha$ value inherited (for the input and output of premultiplied $\alpha$ images by the VSP). Table 1-13 shows the settings of RPF and BRU registers to realize $\alpha$-inherited alpha blending by using figure 1-33 as an example. Registers for use in blend/ROP units B to E can be set in the same way as those in blend/ROP unit A.

![Figure 1-33 Data Flow and the Corresponding Equations in $\alpha$-Inherited Alpha Blending (for the Input and Output of Premultiplied $\alpha$ Images by the VSP)](image)

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPF</td>
<td>Multiply-alpha unit</td>
</tr>
<tr>
<td>BRU</td>
<td>Multiply-alpha unit</td>
</tr>
</tbody>
</table>

$\alpha_s$: $\alpha$ value of the source for unit A  
$C_s'$: RGB color component data of the source for unit A (premultiplied $\alpha$ images)  
$\alpha_d$: $\alpha$ value of the destination for unit A  
$C_d'$: RGB color component data of the destination for unit A (premultiplied $\alpha$ images)  
$\alpha_o$: $\alpha$ value for output from unit A  
$C_o'$: RGB color component data for output from unit A (premultiplied $\alpha$ images)  
$\alpha_r$: $\alpha$ value for output from the BRU  
$C_r'$: RGB color component data for output from the BRU (premultiplied $\alpha$ images)
### Table 1-13 Settings of the RPF and BRU in α-Inherited Alpha Blending (for the Input and Output of Premultiplied α Images by the VSP)

| Register          | Bit         | Setting | Description                                                                
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULTI_ALPH</td>
<td>A_MMD</td>
<td>0</td>
<td>The multiply-alpha unit does not convert α data.</td>
</tr>
<tr>
<td></td>
<td>P_MMD [1:0]</td>
<td>0</td>
<td>The multiply-alpha unit does not convert color component data.</td>
</tr>
<tr>
<td></td>
<td>ALPH_RATIO</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
<tr>
<td>VI6_BRU_INCTRL</td>
<td>NRM</td>
<td>0</td>
<td>The divider does not divide the final results of blending by the α value.</td>
</tr>
<tr>
<td>VI6_BRUA_CTRL</td>
<td>RBC</td>
<td>1</td>
<td>Blending instead of ROP is selected as the type of operation of blend/ROP unit A.</td>
</tr>
<tr>
<td>VI6_BRUA_BLD</td>
<td>CBES</td>
<td>0</td>
<td>The following is selected as the blending expression for the color component data.</td>
</tr>
<tr>
<td></td>
<td>CCMDX [2:0]</td>
<td>3</td>
<td>$1 - \alpha_s$ is selected as the value of coefficient $CX$.</td>
</tr>
<tr>
<td></td>
<td>CCMDY [2:0]</td>
<td>4</td>
<td>COEFY is selected as the value of coefficient $CY$.</td>
</tr>
<tr>
<td></td>
<td>ABES</td>
<td>0</td>
<td>The following is selected as the blending expression with the alpha data.</td>
</tr>
<tr>
<td></td>
<td>ACMDX [2:0]</td>
<td>3</td>
<td>$1 - \alpha_s$ is selected as the value of coefficient $AX$.</td>
</tr>
<tr>
<td></td>
<td>ACMDY [2:0]</td>
<td>4</td>
<td>COEFY is selected as the value of coefficient $AY$.</td>
</tr>
<tr>
<td></td>
<td>COEFX [7:0]</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
<tr>
<td></td>
<td>COEFY [7:0]</td>
<td>255</td>
<td>255 (100%) is selected for the COEFY bits.</td>
</tr>
<tr>
<td>VI6_WPFn_OUTFMT</td>
<td>WRFMT [6:0]</td>
<td>19*</td>
<td>Select the ARGB format, including padding.</td>
</tr>
<tr>
<td></td>
<td>PXA</td>
<td>1</td>
<td>The α value, the result of blending output from the DPR is to be stored in the padding field.</td>
</tr>
<tr>
<td></td>
<td>PDV [7:0]</td>
<td>—</td>
<td>Setting not required (don’t care).</td>
</tr>
</tbody>
</table>

Note: * In addition to 19, any value used for selection of the ARGB format with the padding field is settable.
(4) Register Settings for Normal Alpha Blending

Figure 1-34 shows the data flow and the corresponding equations in the case where blend/ROP unit A is used in normal alpha blending. The output data from blend/ROP unit A becomes the DST for the blend/ROP unit that is the next stage. Table 1-14 shows the settings of BRU registers to realize normal alpha blending by using figure 1-34 as an example. Registers for use in other blend/ROP units can be set in the same way as those in blend/ROP unit A.

![Figure 1-34 Data Flow and the Corresponding Equation in Normal Alpha Blending](image)

\[
C_{o} = (1 - \alpha_s) C_{d} + \alpha_s C_{s}
\]

\(\alpha_s\): a value of the source for unit A  
\(C_d\): RGB color component data of the destination for unit A  
\(C_s\): RGB color component data of the source for unit A  
\(C_o\): RGB color component data for output from unit A  
\(C_r\): RGB color component data for output from the BRU
### Table 1-14 Settings of the BRU in Normal Alpha Blending

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULT_ALPH</td>
<td>A_MMD</td>
<td>0</td>
<td>The multiply-alpha unit does not convert α data.</td>
</tr>
<tr>
<td></td>
<td>P_MMD [1:0]</td>
<td>0</td>
<td>The multiply-alpha unit does not convert color component data.</td>
</tr>
<tr>
<td></td>
<td>ALPH_RATIO [7:0]</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td>VI6_BRU_INCTRL</td>
<td>NRM</td>
<td>0</td>
<td>The divider does not divide the final results of blending by the α value.</td>
</tr>
<tr>
<td>VI6_BRUA_CTRL</td>
<td>RBC</td>
<td>1</td>
<td>Blending instead of ROP is selected as the type of operation of blend/ROP unit A.</td>
</tr>
<tr>
<td>VI6_BRUA_BLD</td>
<td>CBES</td>
<td>0</td>
<td>The following is selected as the blending expression for the color component data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$CX \cdot C_d + CY \cdot C_s$</td>
</tr>
<tr>
<td></td>
<td>CCMDX [2:0]</td>
<td>3</td>
<td>$1 - \alpha_s$ is selected as the value of coefficient CX.</td>
</tr>
<tr>
<td></td>
<td>CCMDY [2:0]</td>
<td>2</td>
<td>$\alpha_s$ is selected as the value of coefficient CY.</td>
</tr>
<tr>
<td></td>
<td>ABES</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td></td>
<td>ACMDX [2:0]</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td></td>
<td>ACMDY [2:0]</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td></td>
<td>COEFX [7:0]</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td></td>
<td>COEFY [7:0]</td>
<td>—</td>
<td>Setting not required (don't care).</td>
</tr>
<tr>
<td>VI6_WPFn_OUTFMT</td>
<td>WRFMT [6:0]</td>
<td>19*</td>
<td>Select the ARGB format, including padding.</td>
</tr>
<tr>
<td></td>
<td>PXA</td>
<td>0</td>
<td>The value in the PDV bits (register setting) is to be stored in the padding field.</td>
</tr>
<tr>
<td></td>
<td>PDV [7:0]</td>
<td>255</td>
<td>α value to be stored in the padding field</td>
</tr>
</tbody>
</table>

Note: * If the output data are in the YCbCr format or in RGB format without padding, the α values are not output.
1.2.5 Alpha Fading
This section describes the settings of the RPF register to realize alpha fading as described in section 1.1.2. A register (VI6_RPFn_MULT_ALPH) of the multiply-alpha unit in the RPF is used to realize alpha fading. Figure 1-35 shows the data flow with alpha fading applied in the respective cases where the input image data are straight α images (case 1 in the figure) and premultiplied α images (case 2 in the figure). The transparency information (α_{pxl}) of the image is multiplied by the fading ratio α_{ratio} (the setting of VI6_RPFn_MULT_ALPH.ALPHA_RATIO[7:0]). With premultiplied α images in use as the input image data, since the image data have been multiplied by the α value in advance, the premultiplied α image data are also multiplied by the fading ratio α_{ratio}. The register settings of the multiply-alpha unit in the RPF in cases 1 and 2 are given in tables 1-15 and 1-16, respectively. In addition, examples of the fading proportion in alpha fading and the corresponding α_{ratio} values (the settings of VI6_RPFn_MULT_ALPH.ALPHA_RATIO[7:0]) are given in figure 1-36.

![Figure 1-35 Data Flow with Alpha Fading Applied](image)

**Table 1-15 RPF Register Settings in Case 1: Application of Alpha Fading to Straight α Images**

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULT_ALPH</td>
<td>A_MMD</td>
<td>1</td>
<td>The multiply-alpha unit multiplies the input pixel α (α_{pxl}) by α_{ratio}.</td>
</tr>
<tr>
<td></td>
<td>P_MMD</td>
<td>[1:0]</td>
<td>The multiply-alpha unit does not convert color component data.</td>
</tr>
<tr>
<td></td>
<td>ALPH_RATIO</td>
<td>[7:0]</td>
<td>Set the desired fading ratio.</td>
</tr>
</tbody>
</table>
Table 1-16  RPF Register Settings in Case 2: Application of Alpha Fading to Premultiplied α Images

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULT_ALPH</td>
<td>A_MMD</td>
<td>1</td>
<td>The multiply-alpha unit multiplies the input pixel α (α&lt;sub&gt;pxl&lt;/sub&gt;) by α&lt;sub&gt;ratio&lt;/sub&gt;.</td>
</tr>
<tr>
<td></td>
<td>P_MMD [1:0]</td>
<td>1</td>
<td>The multiply-alpha unit multiplies the values in input premultiplied α images by α&lt;sub&gt;ratio&lt;/sub&gt;.</td>
</tr>
<tr>
<td>ALPH_RATIO [7:0]</td>
<td></td>
<td>0 to 255</td>
<td>Set the desired fading ratio.</td>
</tr>
</tbody>
</table>

In addition to alpha fading, the multiply-alpha function (selected by setting VI6_RPFn_MULT_ALPH) can realize conversion from straight α images to premultiplied α images (see figure 1-32). Figure 1-37 shows the data flow for both alpha fading and the conversion from straight α images to premultiplied α images at the same time. The register settings of the multiply-alpha unit in the RPF in such a case (case 3 in the figure) are given in table 1-17.
Case 3: Applying alpha fading to straight α images and converting the images to premultiplied α images

Table 1-17 RPF Register Settings in Case 3: Applying Alpha Fading to Straight α Images and Converting the Images to Premultiplied α Images

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI6_RPFn_MULT_ALPH</td>
<td>A_MMD</td>
<td>1</td>
<td>The multiply-alpha unit multiplies the input pixel α ((\alpha_{\text{pix}})) by (\alpha_{\text{ratio}}).</td>
</tr>
<tr>
<td></td>
<td>P_MMD [1:0]</td>
<td>3</td>
<td>The multiply-alpha unit multiplies the data of input straight α images by (\alpha_{\text{ratio}}) and the input pixel α ((\alpha_{\text{pix}})).</td>
</tr>
<tr>
<td></td>
<td>ALPH_RATIO [7:0]</td>
<td>0 to 255</td>
<td>Set the desired fading ratio.</td>
</tr>
</tbody>
</table>
2. Fine Display Processor (FDP)

2.1 Overview of the FDP

2.1.1 Motion Adaptive Interlaced-to-progressive (IP) Conversion

There are two general methods of interlaced-to-progressive (IP) conversion, 2-dimensional (2D) IP conversion and 3-dimensional (3D) IP conversion. The respective methods have both advantages and disadvantages as shown in table 2-1.

The FDP detects motion in pixel units, and switches the method between 3D and 2D IP conversion in response to the results of detection. This takes the advantages of both methods to achieve IP conversion with high picture quality.

<table>
<thead>
<tr>
<th>Method of IP Conversion</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D IP conversion</td>
<td>• Suppressing comb noise in images with motion is possible.</td>
<td>• Still images become unclear.</td>
</tr>
<tr>
<td></td>
<td>• Still images become unclear.</td>
<td>• Line flicker occurs.</td>
</tr>
<tr>
<td>3D IP conversion</td>
<td>• Sharp high-resolution still images are obtained.</td>
<td>• Comb noise occurs in sections with motion.</td>
</tr>
<tr>
<td></td>
<td>• Suppressing line flicker is possible.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1 Comparison of Methods of IP Conversion

Note: * Not applicable to the R-Car D3.
2.1.2 Diagonal Line Interpolation

The FDP has a diagonal line interpolation function and applies it to reduce jaggies (stairlike line-segment noise) which occurs when 2D IP conversion is applied to images with motion, thus finely interpolating diagonal lines.

Figure 2-2  Diagonal Line Interpolation

With no diagonal line interpolation  With diagonal line interpolation
2.2 Guide to Using the FDP

2.2.1 Swapping-Related Settings

The following describes how to set the swapping-related registers of the FDP. The FDP can set up swapping of images for both input and output. The settings to consistently arrange data in external memory as little endian are given in table 2-2. The settings vary according to whether data are to be handled in byte units (YCbCr, RGB332, etc.), 2-byte units (RGB565 or ARGB1555, etc.), or 4-byte units (ARGB, etc.).

Table 2-2 Swapping-Related Settings for the FDP by Type of Data

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Bit Name</th>
<th>Usage of Data</th>
<th>Data Type</th>
<th>Data Unit</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD1_RPF_SWAP</td>
<td>ISWAP</td>
<td>Input image data</td>
<td>YCbCr</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
<tr>
<td>FD1_WPF_SWAP</td>
<td>OSWAP</td>
<td>Output image data</td>
<td>RGB332 (1 byte/pix)</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RGB565, RGB1555 (2 bytes/pix)</td>
<td>2 bytes</td>
<td>4'b1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RGB888 with stuffing (3 bytes/pix)</td>
<td>1 byte*</td>
<td>4'b1111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ARGB8888 (4 bytes/pix), RGB888 (4byte/pix)</td>
<td>4 bytes</td>
<td>4'b1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>YCbCr</td>
<td>1 byte</td>
<td>4'b1111</td>
</tr>
</tbody>
</table>

Note: * Handling a type with 3-byte units of data is not directly supported. Therefore, if RGB888 with stuffing (3 bytes/pixel) is to be used, handle the given data as a type for which data are handled in byte units and specify 4'b1111 as the setting to do so.

Table 2-2 (Continued)

Note that when the FDP is to output FCNL data, follow the guide to the FCP in the hardware manual for the setting of FD1_WPF_SWAP.OSWAP[3:0].

When the FDP is reading tile-addressed data, follow the guide to the FCP in the hardware manual for the setting of FD1_RPF_SWAP.ISWAP [3:0].

2.2.2 Outline of FDP Parameters

The following are details of the two FDP parameters which are described in this application note.

Table 2-3 Parameters Described in This Application Note

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIF_ADJ/SAD_ADJ</td>
<td>Stillness and motion adjustment table</td>
<td>Details of how to set the table (address) are given in the hardware manual. Note that the settings of DIF_ADJ and SAD_ADJ must be the same.</td>
</tr>
<tr>
<td>FD1_IPC_COMB_DET</td>
<td>Register settings for the detection of comb noise</td>
<td></td>
</tr>
</tbody>
</table>

<Stillness and motion adjustment tables>

The FDP determines the balance ratio (2D-3D mixing ratio) between 2D IP conversion and 3D IP conversion on the basis of differences between fields, i.e. the results of motion detection. The stillness and motion adjustment tables (DIF_ADJ and SAD_ADJ) can be used to set the 2D-3D mixing ratio from differences between fields.

<Register settings for the detection of comb noise>

In addition, during the process of motion judgment by the FDP, comb noise is detected, and exception handling makes changes in the motion judgment result.
The values in the FD1_IPC_COMB_DET register are set up for the detection of comb noise and the stillness and motion adjustment tables are corrected in response.
2.2.3 Setting the Stillness and Motion Adjustment Tables

Figure 2-3 shows the stillness and motion adjustment table (DIF_ADJ or SAD_ADJ) described in the section on the FDP in the hardware manual and the method of adjusting the table. The x-axis represents the difference between fields and the y-axis represents the 2D-3D mixing ratio. Set the individual 2D-3D mixing ratios in the stillness and motion adjustment table with the use of the differences between fields as index numbers of the table. Setting 1 (default) tends most strongly toward motion (the balance ratio of 2D IP conversion is high), and setting 3 tends most strongly toward stillness (the balance ratio of 3D IP conversion is high). If you wish to make a setting other than these three, adjust the table.

![Stillness and motion adjustment table diagram](image)

Figure 2-3 How to Adjust the Stillness and Motion Adjustment Table
2.2.4 Setting the Register for the Detection of Comb Noise

The FDP has exception handling that can be used to adjust the stillness and motion adjustment table such that its setting tends toward motion in response to the detection of comb noise. When comb noise is still bothering after the stillness and motion adjustment table has been adjusted, use the above function.

To enable the detection of comb noise, set the parameters of the FD1_IPC_COMB_DET register as follows. Note that these settings lead to judging a horizontal line to be comb noise, as a result, deleting gridlines in a GUI menu display and other lines.

CMB_OFST = 0x20
CMB_MAX = 0xFF
CMB_GRAD = 0xFF

This function is disabled with the default values as listed in the hardware manual.
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.02</td>
<td>September, 2017</td>
<td>—</td>
<td></td>
<td>First edition issued.</td>
</tr>
<tr>
<td>1.03</td>
<td>November, 2017</td>
<td>1</td>
<td>Target Device updated.</td>
<td>“R-Car M3-N”</td>
</tr>
<tr>
<td>1.04</td>
<td>March, 2018</td>
<td>1</td>
<td>Target Device updated.</td>
<td>“R-Car E3”</td>
</tr>
<tr>
<td>1.05</td>
<td>June, 2018</td>
<td>46 and 61</td>
<td>The data type “RGB888 (4byte/pix)” added in Table 1-10 and Table 2-2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Target Device updated.</td>
<td>“Version3.0 of the R-Car H3”</td>
</tr>
<tr>
<td>1.06</td>
<td>August, 2018</td>
<td>1,3,6,7,9,10,11,12 and 59</td>
<td>Target Device updated.</td>
<td>“R-Car D3” The description of R-Car D3 added.</td>
</tr>
<tr>
<td>1.07</td>
<td>September, 2019</td>
<td>1</td>
<td>Target Device updated from “Version 1.1 of the R-Car M3-W” to “Version 1.x of the R-Car M3-W” and “R-Car M3-W+”</td>
<td></td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL} (\text{Max})$ and $V_{IH} (\text{Min})$ due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL} (\text{Max})$ and $V_{IH} (\text{Min})$.

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
CONFIDENTIAL

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality.” The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; underwater repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

© 2019 Renesas Electronics Corporation. All rights reserved.