32bit Colour Channel Shifting using 8bit and 16bit texture formats

Author: Mark Breugelmans (Sony Computer Entertainment Europe)
Date: 26 October 2001 – October 2004

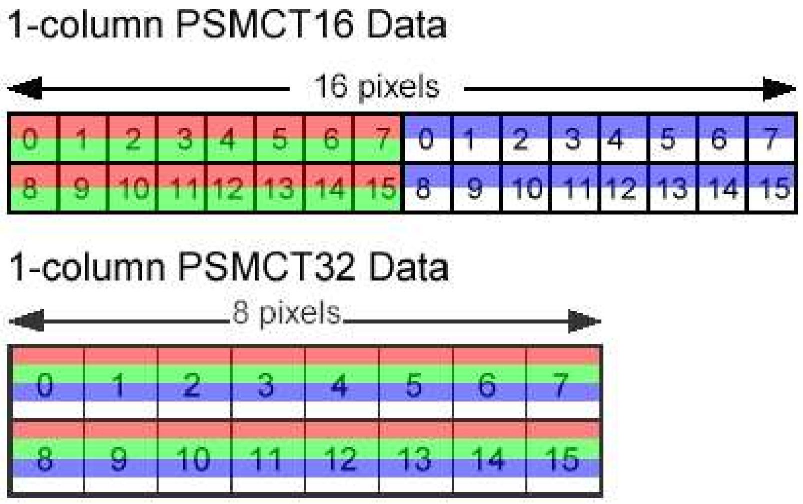
Overview
This document has been written to give a simplified understanding of GS memory and the mappings between 32, 16 and 8bit pixel formats for use in colour channel copying operations.

Introduction
There are occasions where you may wish to perform channel copying operations for example Green->Alpha, etc. GS memory is laid out in a non-linear fashion internally to achieve optimal speed during rasterization.

16bit channel operations:
Up to 2 channels are copied at a time: {red and green} or {blue and alpha}

As you can see in this diagram for 16bit textures the 32bit colour channels become split up into Red/Green and Blue/Alpha in groups of 8x2pixels.
Note: In these diagrams for alpha I used the colour white.

Mapping within 1-column from 16bit to 32bit format:



Moving Red/Green into Blue/Alpha just requires you to copy 8 pixels to the right.
Similarly moving Blue/Alpha into Red/Green requires you to copy 8 pixels to the left.

You can ignore the block arrangement within each page, as you are both reading and writing in 16bit mode.

Simply set Texture mode to PSMT16 and Frame mode to PSMT16 and copy using no texture filtering (point sampling). You will need to copy double the height of the 32bit frame buffer.

Using vertical strips of 8pixels wide will give you optimal performance for this. If you don't want to copy one of the channels you can use the FBMASK to inhibit writes.

Note: This type of operation will not work in-place if you are trying to add colour values to one another.

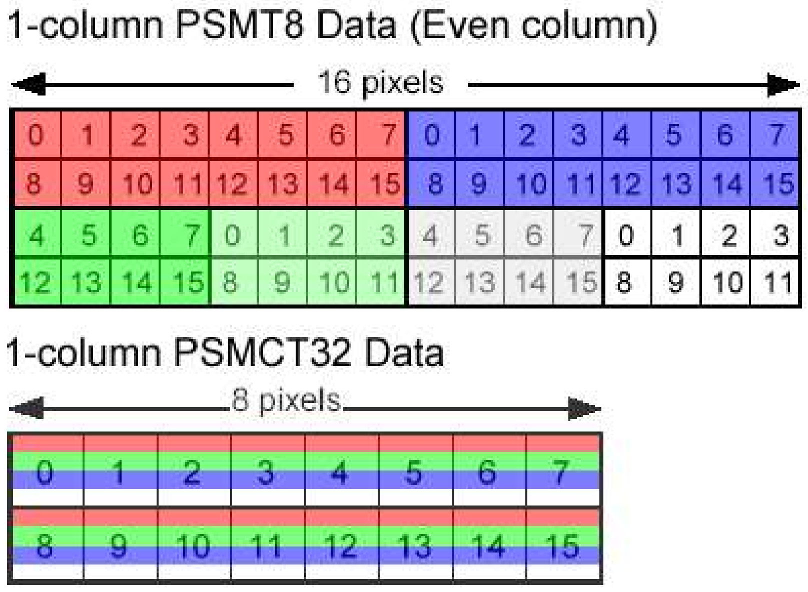
Example code for 16bit Red, Green copy into Blue, Alpha of a temporary texture for the whole frame buffer

```
SetGSReg( SCE_GS_XYOFFSET_1, SCE_GS_SET_XYOFFSET(0,0));
SetGSReg( SCE_GS_FRAME_1,SCE_GS_SET_FRAME(tempTextureFBP,
                      (frameBufferWidth>>6), SCE_GS_PSMCT16, 0));
SetGSReg( SCE_GS_ZBUF_1, SCE_GS_SET_ZBUF( 0, SCE_GS_PSMZ16S, 0));
SetGSReg( SCE_GS_DTHE, 0);                                     // Disable Dithering
SetGSReg( SCE_GS_TEST_1, SCE_GS_SET_TEST(1,0,0,1,  0,0,1,1)); // Disable Z-tests/writes
SetGSReg( SCE_GS_ALPHA_1,SCE_GS_SET_ALPHA(2, 2, 2, 0, 128));  // Disable alpha blend
SetGSReg( SCE_GS_TEXFLUSH, 0);                                 // Flush texture cache
SetGSReg( SCE_GS_TEX0_1, SCE_GS_SET_TEX0((frameFBP<<5),       // Texture = FRAME
                  (frameBufferWidth>>6), SCE_GS_PSMCT16,
                  10, 10, 1, 1, 0, 0, 0,
                  0, 0));
SetGSReg( SCE_GS_FBA_1, SCE_GS_SET_FBA(0));
SetGSReg( SCE_GS_TEXA, SCE_GS_SET_TEXA(0, 0, 0x80));
SetGSReg( SCE_GS_TEX1_1, SCE_GS_SET_TEX1(1, 0, 0, 0, 0, 0,0));// Point sampling

// Draw copy sprite to 0,0 in dest
SetGSReg( SCE_GS_PRIM, SCE_GS_SET_PRIM(SCE_GS_PRIM_SPRITE, 0, 1, 0, 1, 0, 1, 0, 0));
SetGSReg( SCE_GS_RGBAQ,SCE_GS_SET_RGBAQ(0x80, 0x80, 0x80, 0x80, 3f800000));

for (iVertStrip=0; iVertStrip<(frameBufferWidth>>4); iVertStrip++) // 8 pixel wide strips
{
 // In GS co-ordinates use (+8,+8) for centre of pixel
 SetGSReg( SCE_GS_UV,   SCE_GS_SET_UV ( 8+(iVertStrip<<8), 8) );
 SetGSReg( SCE_GS_XYZ2, SCE_GS_SET_XYZ( (8<<4)+(iVertStrip<<8), 0, 0) );
 SetGSReg( SCE_GS_UV,   SCE_GS_SET_UV ( 8+(iVertStrip<<8)+(8<<4), 8+(height<<1) ) );
 SetGSReg( SCE_GS_XYZ2, SCE_GS_SET_XYZ( (8<<4)+(iVertStrip<<8)+(8<<4), (height<<1), 0));
}
```

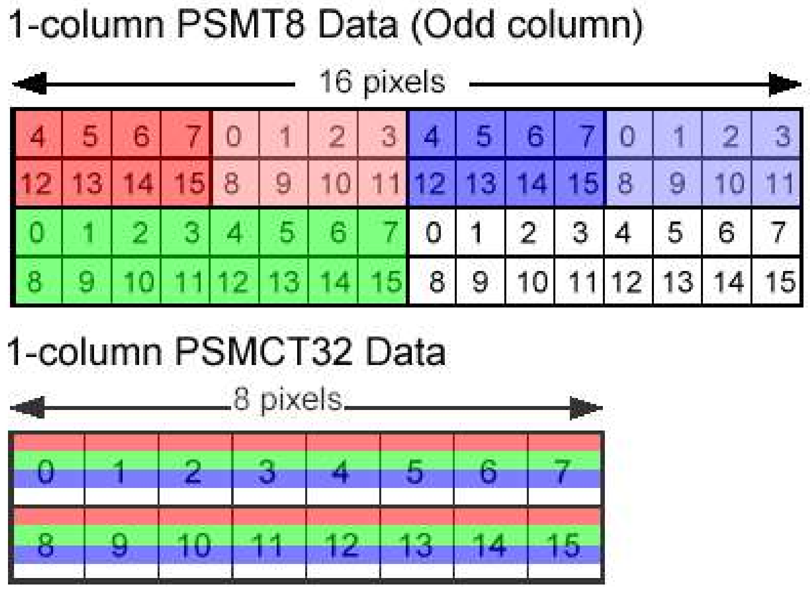
8bit channel operations
(For single channel moving eg: Blue->Alpha)

For 8bit textures the 32bit colour channels are nicely divided into quarters, which again map to groups of 8x2pixels. However the individual pixel mappings for each colour are arranged in 4x2 areas in which the mappings alternate between odd and even 'columns' (each column is actually a row). The mappings are as below:

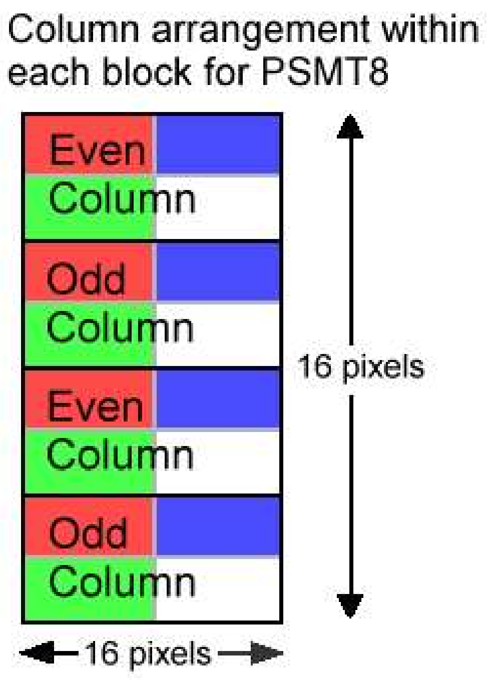
Even Column Mapping:



Odd Column Mapping:



The block arrangement in each page for PSMT8 and PSMCT32 is identical so you can ignore that. However you need to follow the odd and even column order within each block as below when choosing which mapping to use.



You can copy these pixels using PSMT8 as your texture source with a 32bit grey scale CLUT. Pixels are written back in PSMCT32 mode. Use FBMASK to mask channels you don't want to write to. This operation can be done in-place.

Using the texture **region-repeat** it is possible to obtain reasonable performance for 8bit channel copy operations. See example below:

Example code for Blue to Alpha in-place copy for 1 page.

```
SetGSReg( SCE_GS_XYOFFSET_1, SCE_GS_SET_XYOFFSET(0,0));
SetGSReg( SCE_GS_FRAME_1,
      SCE_GS_SET_FRAME(pageFBP, (frameBufferWidth>>6),
      SCE_GS_PSMCT32,
      0x00ff0000));  // mask write to blue channel
SetGSReg( SCE_GS_TEX0_1,
      SCE_GS_SET_TEX0((pageFBP<<5), // TextureSrc = Current page
      2, SCE_GS_PSMT8,
      10, 10, 1, 1, ((8192*509)>>8), // greyscale CLUT address
       0, 0, 0, 1 ));  // CLD=1(Load clut)

SetGSReg( SCE_GS_RGBAQ,
      SCE_GS_SET_RGBAQ(0x80,0x80,0x80,0x80,0x3f800000));
SetGSReg( SCE_GS_PRIM,
      SCE_GS_SET_PRIM(SCE_GS_PRIM_SPRITE, 0, 1, 0, 1, 0, 1, 0, 0));
SetGSReg( SCE_GS_CLAMP_1,
      SCE_GS_SET_CLAMP(3,0,0xf7,0x8,0,0)); // Region repeat CLAMP

SetGSReg( SCE_GS_TEXFLUSH, 0);  // Flush texture cache out
SetGSReg( SCE_GS_FBA_1, SCE_GS_SET_FBA(0));
SetGSReg( SCE_GS_TEXA,  SCE_GS_SET_TEXA(0, 0, 0x80));
SetGSReg( SCE_GS_TEST_1, SCE_GS_SET_TEST(0,0,0,0,  0,0,1,1));
SetGSReg( SCE_GS_ALPHA_1,SCE_GS_SET_ALPHA(2, 2, 2, 0, 128)); // DEST=SRC

for (y=0; y<32; y+=2)
{
 if ((y%4)==0)
 {
  // EVEN COLUMN (Four 16x2 pixel sprites)
  for (x=0; x<64; x+=16) // 4 sprites
  {
   // Using region repeat mask to skip 8 texel so you can use a 16 wide sprite
   // XY [0,0 to 16,2]
   SetGSReg(SCE_GS_UV,   SCE_GS_SET_UV(8 + ((8+x*2)<<4) , 8 + ((y*2)<<4) ));
   SetGSReg(SCE_GS_XYZ2, SCE_GS_SET_XYZ2( (x<<4)        , (y<<4), 0 ));
   SetGSReg(SCE_GS_UV,   SCE_GS_SET_UV(8 + ((24+x*2)<<4), 8 + ((2+y*2)<<4) ));
   SetGSReg(SCE_GS_XYZ2, SCE_GS_SET_XYZ2( ((x+16)<<4)    , ((y+2)<<4), 0 ));
   }
 }
 else
 {
   // ODD COLUMN (Eight 8x2 pixel sprite)
   for (x=0; x<64; x+=8) // 8 sprites
   {
    // Use region repeat mask to swap 4x2 regions over while fetching texels
     // XY [0,0 to 8,2]
    SetGSReg(SCE_GS_UV,   SCE_GS_SET_UV(8 + ((4+x*2)<<4) , 8 + ((y*2)<<4) ));
    SetGSReg(SCE_GS_XYZ2, SCE_GS_SET_XYZ2( (x<<4)        , (y<<4), 0 );
    SetGSReg(SCE_GS_UV,   SCE_GS_SET_UV(8 + ((12+x*2)<<4), 8 + ((2+y*2)<<4) ));
    SetGSReg(SCE_GS_XYZ2, SCE_GS_SET_XYZ2( ((x+8)<<4)    , ((y+2)<<4), 0 ));
  }
 }
}
```