Freescale Semiconductor
Application Note

Document Number:AN4672
Rev. 0, 01/2013

Using the Run-Length Decoding Features on Vybrid Devices

by: Luis Olea and Michael Staudenmaier

Contents

1 Introduction

Run-length encoding (RLE) is a method that allows data compression for information in which symbols are repeated constantly. The method is based on the fact that the repeated symbol can be substituted by a number indicating how many times the symbol is repeated and the symbol itself. To cope with different type of data the RLE decoder implemented in Vybrid supports the following constructs:

- Repeated symbols: This subset of data consists of the symbols that can be compressed by replacing them with a number indicating how many times the symbol is repeated and the repeated symbol.
- Gradients: This subset of data consists of the symbols that can be compressed using a linear approximation. The approximation is expressed using an initial symbol, a delta, and the number of symbols that will be generated adding a delta to the initial symbol.
- Non-repeated symbols: This subset of data consists of the symbols that cannot be compressed by any of the prior methods. It is stored as non-compressed data.

For example, consider the pixel row marked as original data in Figure 1. Notice how the colors are repeated in some areas. By applying RLE we get the compressed data (bottom row of Figure 1, consisting only in nine elements achieving a compression of 2.8 times.

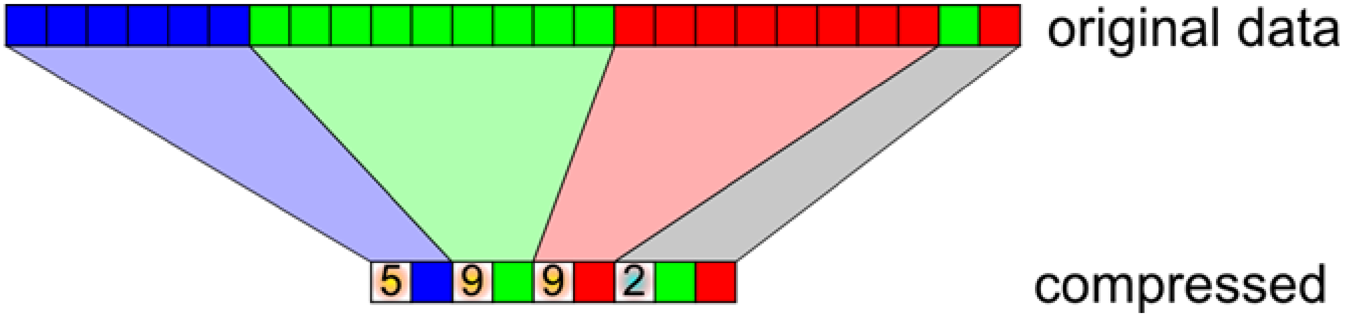


Figure 1. RLE compression

In the case of gradients or any serial data that is linear, consider the pixels marked as original data in Figure 2. The linear gradient consisting of 12 colors can be represented by a number indicating the length of the gradient, the starting color, and the difference between one color and the next. By applying gradient RLE on Vybrid, we can compress the data to three elements, achieving a compression of four times.

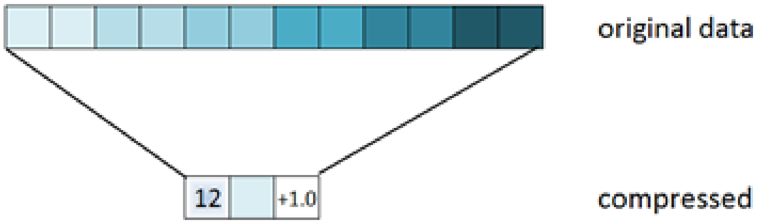


Figure 2. Gradient compression

The RLE decoder implemented on Vybrid does not know the concept of rows and lines in a displayed frame but treats the data as a single dimension array as shown in Figure 3.

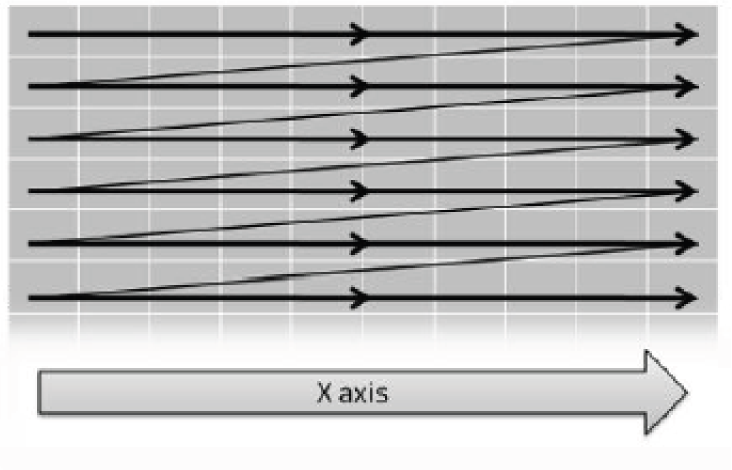


Figure 3. RLE encoding direction

2 RLE on the Vybrid devices

Vybrid devices have hardware supported runtime RLE decompression optimized for pixel data. The module can decode the following different pixel formats for source data: 8 bpp, 16 bpp, 24 bpp, and 32 bpp. This allows the selection of atomicity of source data for more efficient compression.

The RLE capabilities of the Vybrid can be divided into two sections:

- 2D-ACE embedded RLE: The DCU3 can automatically read RLE compressed data from images and display it on a screen seamlessly. The DCU3 has a dedicated decompression module and can be used simultaneously with the standalone RLE module.
- Standalone RLE decoder: This module allows the decompression of RLE data independently from the DCU3. It can be used for any type of RLE data, including non-graphics data. It also supports gradient RLE compression.

2.1 Gradient RLE mode

Gradients in an image are basically areas that have a starting color which gradually fades into a different color. Gradients are reconstructed by taking the starting color, adding a numeric delta to generate a second color, then adding the delta to the second color to generate the third color and so on until the number of required output pixels are generated.

The Gradient RLE functionality on Vybrid allows improving the compressed size of a gradient from 80% (traditional RLE compression) down to approximately 5% of the original size. The Gradient RLE feature uses a specific symbol which is not used in the traditional RLE format, thus keeping compatibility with previous devices such as MPC5645S.

2.2 RLE format

The Vybrid RLE decoder supports uncompressed sequences of pixel data called RAW and compressed pixel data. Each sequence starts with an 8-bit command indicating whether the sequence is raw or compressed.

Format of the marker:

Table 1. Format of the marker

Bit	7	6	5	4	3	2	1	0
Function	CR	Count						

- CR
  - 1: Compressed sequence
  - 0: Raw sequence (uncompressed)
  - Count: Number of pixels encoded in the successive sequence. The number of pixels is actually Count+1.

A raw encoded sequence consists of Count+1 pixels, pixel width depends on the selected mode and can either be 8-bit, 16-bit, 24-bit, or 32-bit wide. All pixels are copied unmodified to the output.

In the Vybrid RLE decoder block, there are two styles of compressed sequences: RLE compressed and Gradient compressed.

To differentiate gradient compressed sequences from RLE compressed ones, the redundant coding of a RLE sequence with length 1 is used as a special marker indicating that the successive sequence is gradient encoded.

An RLE sequence with length one is function-wise the same as an RAW compressed sequence with length one.

An RLE sequence command is followed by one pixel in the stream. This pixel is then output count+1 times to the output. A pixel can either be 8-bit, 16-bit, 24-bit, or 32-bit wide.

Gradient sequences use the following modified layout:

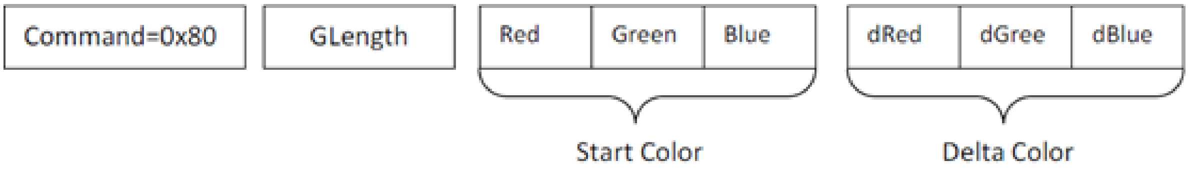


Figure 4. Compressed gradient sequence

Gradients are stored slightly different, reflecting the nature of a gradient.

- Command byte: 0x80 to identify a Gradient encoded sequence
- GLength: output length of the gradient sequence. The number of generated output pixels is actually GLength+3
- Start Color: color of the first pixel of the sequence. The start color consists out of 1, 3, or 4 color components with 8 bits each. Figure 4 illustrates the case of 3 components per pixel.
- Delta color: the delta color contains a signed 8-bit fixed-point number for each color component identifying the incremental change in the specific color component of two adjacent pixels. Delta color has the format S2.6.

2.2.1 Gradient RLE compression with and without losses

Images with linear gradients that can be exactly represented with gradient RLE are common within artificial images, such as, wallpaper or other visual elements that are generated by software. Natural images usually contain non-linear gradients that cannot be represented by starting color, delta, and number of pixels.

For such use-cases, it is possible to accept a certain deviation of the de-compressed value from the original value to boost the compression ratio achievable. This lossy image compression is entirely implemented in the compressor software, that is, it does not imply a specific mode in the RLE decoder block.

This feature allows to trade of quality versus compressed image size. In most cases, small deviation of the decompressed color values from the perfect value are not even visible due to the limited color resolution of the attached TFT display.

Generally all kind of images might benefit in terms of compression ratio when allowing gradient RLE to represent the original image to within certain accuracy.

2.3 2D-ACE embedded RLE

The DCU3 and the DCULite have the ability decode and display RLE compressed images on the fly. Using an RLE image has the following limitations:

- Can be used only on either layer 0 or layer 1 on a single layer
- Supports 8 bpp, 16 bpp –except YUV, 24 bpp, and 32 bpp
- Tile mode is not permitted when RLE mode is active
- Gradient RLE is not supported for DCU3 embedded RLE (only supported in standalone RLE)

From the application point of view, there is no difference between using a non-RLE and an RLE image except that by using an RLE image a lot of memory space and bandwidth can be saved.

Configuring the DCU3 or the DCULite to display an RLE image is similar to configuring it to display any other kind of image. You must only perform the following additional steps to the layer initialization:

- Set DDR_MODE in DCU_MODE

- Set the compressed size of the bitmap on the Compression Image Size Register COMP_IMSIZE
- Set the RLE_EN bit on the control descriptor 4 register CTRLDESCLn_4

2.4 Standalone RLE decoder

RLE compression is not only useful to compress images but other types of data such as sound and program code. The standalone RLE decoder in the Vybrid allows decoding any type of data whether it is an image or not.

To decode RLE data, configure the eDMA channels to move the data to and from the RLE decoder module. DMA transfers will be triggered when required by input and output FIFOs.

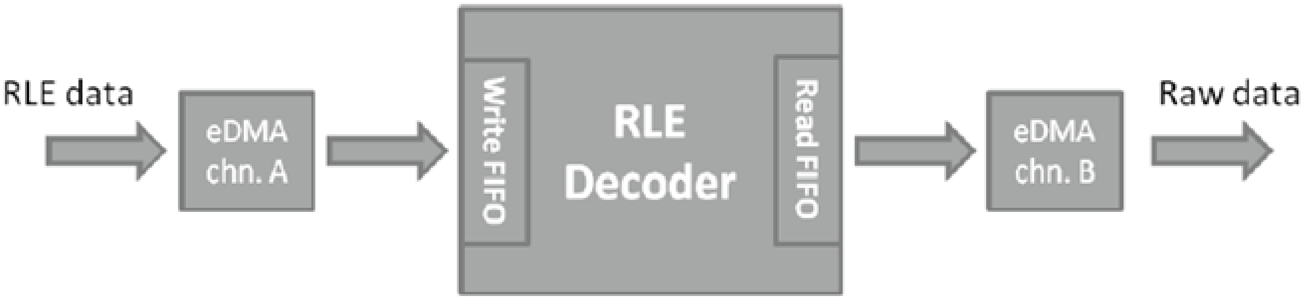


Figure 5. RLE application block

The RLE data in application diagram shown in Figure 5 can be fetched from any of the available memories and the decoded data can be put into SRAM, GRAM, or DRAM memory.

When images are decoded, the RLE Decoder also has the ability to extract a smaller image or portion of the original RLE compressed image. The images are divided into a virtual grid of pixels where it is possible to select the start position and end position. Figure 6 is an example of an image with the grid division.

NOTE

The grid start point is located at (1,1), the end point is located at (20, 20). The start and the end points of the selection rectangle are included in the partial image area.

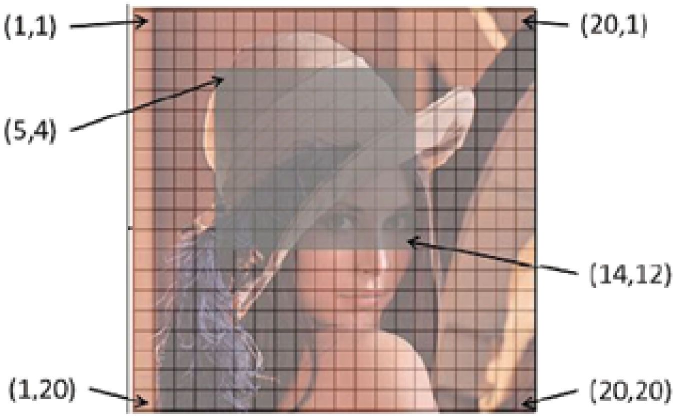


Figure 6. 20x20 pixels image example with grid

To configure the RLE module, use the steps below. A software example is provided at the end of this application note.

- Configure the read eDMA channel, Channel A in Figure 3
  - The channel reading the RLE data must point to the starting address where the RLE data is located and as destination the memory mapped write FIFO.

    - 8-, 16-, or 32-bit transfer size accesses can be used. Configure Source and Destination size accordingly.
    - Destination offset will be zero because the write FIFO is memory mapped while Source offset depends on the transfer size.
    - The minor loop count depends on the configured threshold of the write FIFO. The minor loop count must be calculated such that it never overflows the write FIFO.
    - Set the major loop count to 1 and configure the DMAMUX to trigger transfers each time the write FIFO is below the threshold.
- Configure the write eDMA channel, Channel B in Figure 3
    - The channel reading the uncompressed data must point to the read FIFO where the decoded data is placed and as destination the memory where to output the raw data.
    - 8-, 16-, or 32-bit transfer size accesses can be used. Configure Source and Destination size accordingly.
    - Source offset will be zero because the read FIFO is memory mapped while Destination offset depends on the transfer size.
    - The minor loop count depends on the configured threshold of the read FIFO. The minor loop count must be calculated such that it never overruns the read FIFO.
    - Set the major loop count to 1 and configure the DMAMUX to trigger transfers each time the read FIFO is above the threshold.
- Configure the RLE module
    - First, make sure the module is disabled, MDIS=1.
    - Set the compressed size of the RLE data in the register RLE_DEC_CISR.
    - Set the X and Y original image size in pixels in the register RLE_DEC_ DICR.
    - Configure the area of interest: set the start pixel coordinates, RLE_DEC_SPCR, and final pixel coordinates, RLE_DEC_EPCR. The area of interest can be the same original image or a smaller part from the original image. Check Figure 5 coordinates example.
    - Configure the pixel format, WIDTH, on the register, RLE_DEC_ICR.
    - If you are using gradient RLE, enable its setting GRLE_EN to 1 in RLE_DEC_MCR register.
    - Enable the module so the decoding process starts, MDIS=0.

3 Use case examples

It is possible to use statistical methods to analyze the entropy of certain images and decide whether the compression ratio will be good or not, but typically it is done empirically by just looking at the images. Alternatively, performing an RLE compression of an image is fairly fast in order to apply the trial and error method.

Because RLE is applied along the X axis, as it is how images are natively stored on memory, data having long runs of repeated data on the X axis will be compressed efficiently while data having long runs only on the Y axis will not be compressed.

Some images with good chances to have a good compression ratio are:

- Horizontal gradients (gradient RLE)
- Any type of shape having a solid background
- Images having single color areas

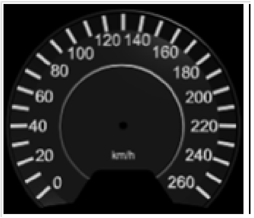
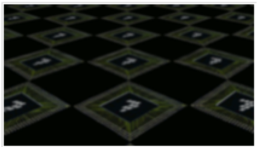
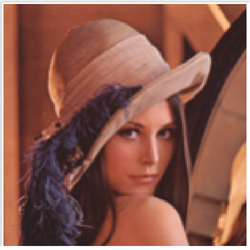
Natural images such as photographs usually achieve very limited gain using RLE compression. The table below showcases some examples of images with its compression ratios.

Table 2. RLE use examples

Image	Description	Size	Comp. Size	Comp. Ratio

Table continues on the next page...

Table 2. RLE use examples (continued)

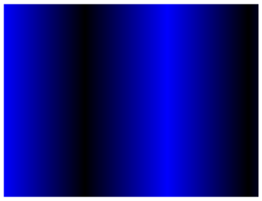
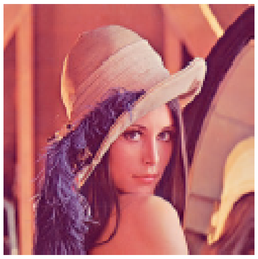
	This image is a round shape with a solid background and single color areas around the center. This type of graphics always results in very good compression ratios. 200x176@8bpp	35200 Bytes	11864 Bytes	3:1
	This is a background image that has identifiable black color areas between the green diamonds and further black inside them. This allows a 47% size reduction. 480x272@8bpp	130560 Bytes	69015 Bytes	<2:1
	This example has a horizontal color gradient with a big black color area; all these characteristics give this picture an excellent RLE compression ratio. 500x400@24bpp	600000 Bytes	111031 Bytes	>5:1
 你好世界	Raster text is often a good candidate for compression, especially if it is big. 255x57@4bpp (Grayscale)	11400 Bytes	2781 Bytes	4:1
	Although traditional RLE does not compress this image, see Use case examples for gradient RLE for compression with gradient RLE. 512x512@24bpp	786432 Bytes	774304 Bytes	1:1

4 Use case examples for gradient RLE

This section provides some examples of images being compressed with gradient RLE. With this data we get a clear view of what can be achieved by using both traditional and gradient RLE to compress an image. For each of the images the resulting image size for the following compressions is given:

- Regular RLE: traditional RLE (only series of identical pixels are compressed)
- GRLE Err=0: Gradient RLE encoding requiring bit accurate output
- GRLE Err=1: Gradient RLE allowing a deviation of maximum 1 for the pixel intensity channel

Table 3. Gradient RLE use examples

Image	Regular RLE	RLE+Gradient Err = 0	RLE+Gradient Err=1	RLE+Gradient Err=2
	47.50%	45.62%	1.83%	1.83%
	102.26%	1.83%	1.83%	1.83%
	81.17%	81.36%	2.77%	1.83%
	100.11%	100.11%	99.81%	96.28%

5 Using the Freescale Image Encoder

The following instructions explain how to turn an image stored in a PC to a c code file that can be used in an embedded project for Vybrid.

1. Download and install Freescale Image Encoder (part of the software for this application note)
2. Run the Freescale Image Encoder

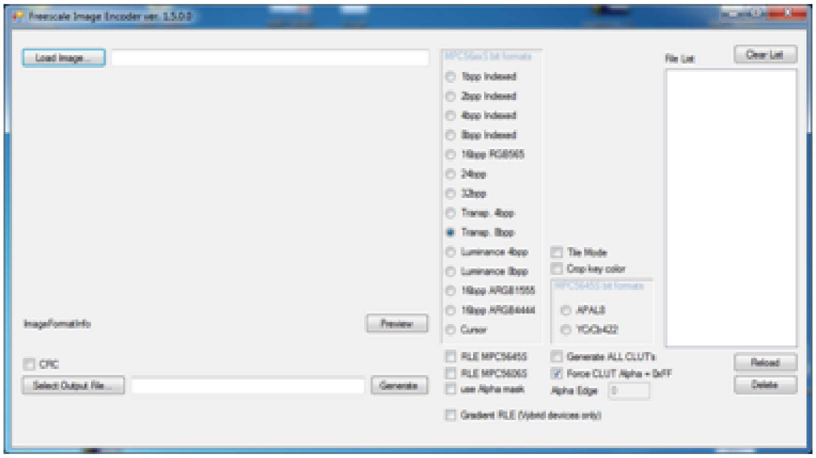


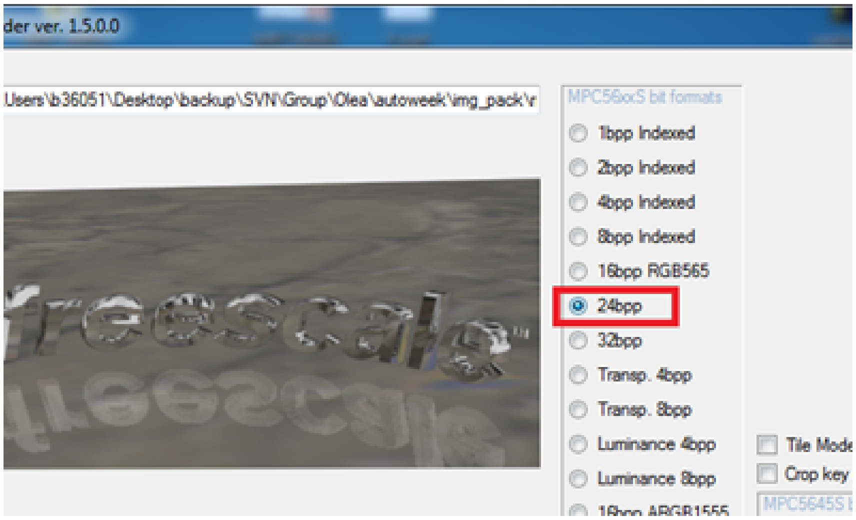
Figure 7. Image encoder main window

3. Choose the image file that you want to use

C:\Users\b36051\Desktop\backup\SVN\Group\Olea\autoweek\img_pack\r

Figure 8. Load image

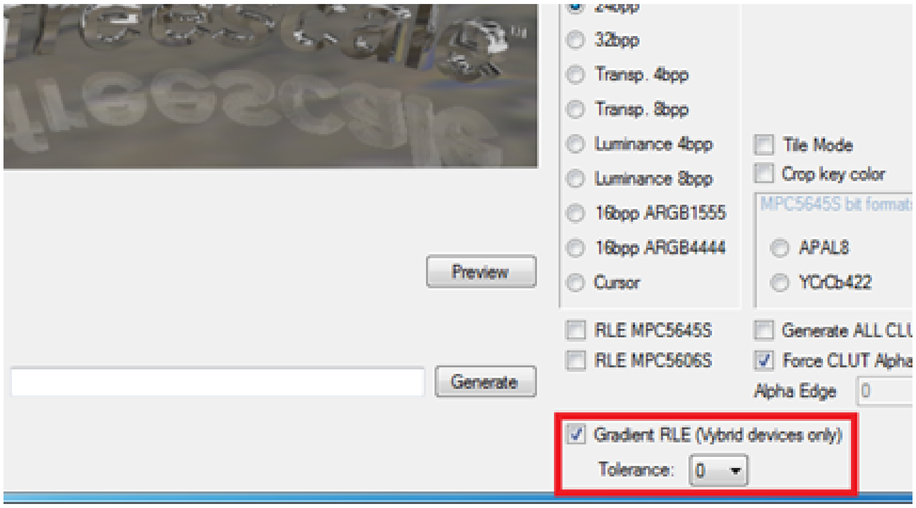
4. Choose the final image format that will be used to store the image in memory.



Users\b36051\Desktop\backup\SVN\Group\Olea\autoweek\img_pack\r

Figure 9. Choosing output format

5. If you want to compress the image, choose Vybrid RLE (only 8, 24, and 32 bpp formats are supported by gradient RLE. For traditional RLE then MPC5645S RLE can be used)



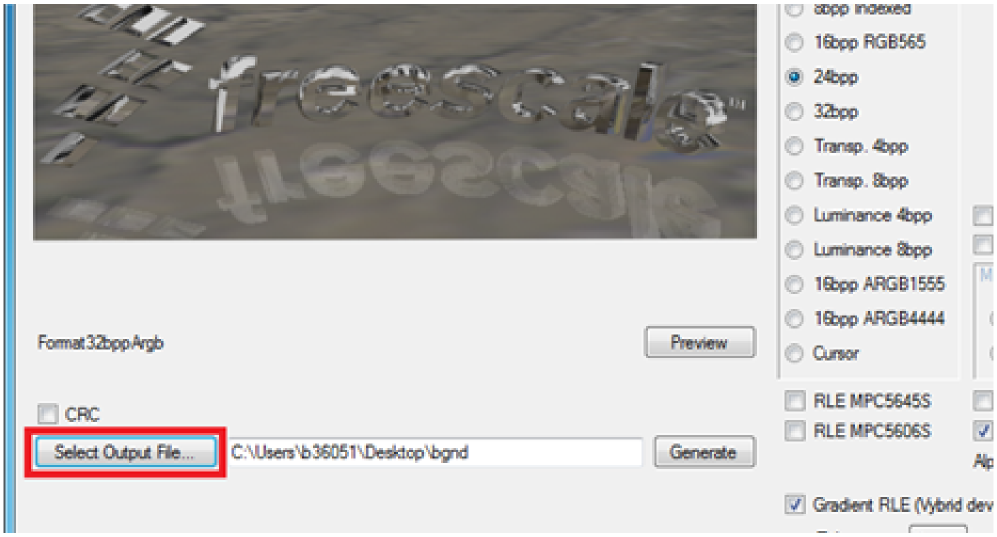
☑ Force CLUT Alpha

☑ Gradient RLE (Vybrid devices only)

Tolerance: 0

Figure 10. Choosing gradient type and tolerance

6. Set an output location and name for the resulting files



● 24bpp

Format32bppArgb

C:\Users\b36051\Desktop\bgnd

☑ Gradient RLE (Vybrid dev

Figure 11. Saving the output file

7. You can preview the resulting image by using the “preview” button
8. Click the generate button to save the image data for your embedded project in .c and .h c code format.

6 Example code

The following example code shows how to configure the RLE decoder module and the eDMA to decode and transfer the data to certain memory location:

```
/* Initialize eDMA channel TCD (0,1) to zeroes */
ptr = DMA0_TCD0_WORD0;
for(i = 0; i<16; i++){ *ptr++ = 0; }

/* configure eDMA channel 0 (for write FIFO) */
reg32_write(DMA0_TCD0_WORD0, (uint32_t)rleBitmap); //source bitmap
reg32_write(DMA0_TCD0_WORD4, 0x78000080) //mem mapped write FIFO
reg32_write_mask(DMA0_TCD0_WORD1, 2<<DMA_TCDn_WORD1_SSIZE_SHIFT,
DMA_TCDn_WORD1_SSIZE_MASK); // src size 32bits
```

```
reg32_write_mask(DMA0_TCD0_WORD1, 2<<DMA_TCDn_WORD1_DSIZE_SHIFT,
DMA_TCDn_WORD1_DSIZE_MASK); // dst size 32bits
reg32_write_mask(DMA0_TCD0_WORD1, 4<<DMA_TCDn_WORD1_SOFF_SHIFT,
DMA_TCDn_WORD1_SOFF_MASK);   // src offset 4 bytes
reg32_write_mask(DMA0_TCD0_WORD5, 0<<DMA_TCDn_WORD5_DOFF_SHIFT,
DMA_TCDn_WORD5_DOFF_MASK);   // dst offset 0 bytes
reg32_write_mask(DMA0_TCD0_WORD7, 1<<DMA_TCDn_WORD7_BITER_SHIFT,
DMA_TCDn_WORD7_BITER_MASK); // major loop = 1
reg32_write_mask(DMA0_TCD0_WORD5, 1<<DMA_TCDn_WORD5_CITER_SHIFT,
DMA_TCDn_WORD5_CITER_MASK); // major loop = 1
reg32_write_mask(DMA0_TCD0_WORD2, 28<<DMA_TCDn_WORD2_NBYTES_SHIFT,
DMA_TCDn_WORD2_NBYTES_MASK); // minor loop = 28
reg32_write_mask(DMAMUX0_CHCFG0, 1<<DMAMUX_CHCFGn_ENBL_SHIFT, DMAMUX_CHCFGn_ENBL_MASK); //
enable DMAMUX for channel 0
reg32_write_mask(DMAMUX0_CHCFG0, 54<<DMAMUX_CHCFGn_SOURCE_SHIFT,
DMAMUX_CHCFGn_SOURCE_MASK); //write fifo request

/* configure eDMA channel 1 (for read FIFO) */
reg32_write(DMA0_TCD1_WORD0, 0x780000C0); //mem mapped read FIFO
reg32_write(DMA0_TCD1_WORD4, 0x80000000); //destination start of external DRAM
reg32_write_mask(DMA0_TCD1_WORD1, 2<<DMA_TCDn_WORD1_SSIZE_SHIFT,
DMA_TCDn_WORD1_SSIZE_MASK); // src size 32bits
reg32_write_mask(DMA0_TCD1_WORD1, 2<<DMA_TCDn_WORD1_DSIZE_SHIFT,
DMA_TCDn_WORD1_DSIZE_MASK); // dst size 32bits
reg32_write_mask(DMA0_TCD1_WORD1, 0<<DMA_TCDn_WORD1_SOFF_SHIFT,
DMA_TCDn_WORD1_SOFF_MASK);   // src offset 0 bytes
reg32_write_mask(DMA0_TCD1_WORD5, 4<<DMA_TCDn_WORD5_DOFF_SHIFT,
DMA_TCDn_WORD5_DOFF_MASK);   // dst offset 4 bytes
reg32_write_mask(DMA0_TCD1_WORD7, 1<<DMA_TCDn_WORD7_BITER_SHIFT,
DMA_TCDn_WORD7_BITER_MASK); // major loop = 1
reg32_write_mask(DMA0_TCD1_WORD5, 1<<DMA_TCDn_WORD5_CITER_SHIFT,
DMA_TCDn_WORD5_CITER_MASK); // major loop = 1
reg32_write_mask(DMA0_TCD1_WORD2, 28<<DMA_TCDn_WORD2_NBYTES_SHIFT,
DMA_TCDn_WORD2_NBYTES_MASK); // minor loop = 28
reg32_write_mask(DMAMUX0_CHCFG1, 1<<DMAMUX_CHCFGn_ENBL_SHIFT, DMAMUX_CHCFGn_ENBL_MASK); //
enable DMAMUX for channel 1
reg32_write_mask(DMAMUX0_CHCFG1, 53<<DMAMUX_CHCFGn_SOURCE_SHIFT,
DMAMUX_CHCFGn_SOURCE_MASK); //write fifo request

/* Configure RLE module */
reg32setbit(RLE_MCR,RLE_MCR_MDIS_SHIFT); //disable module
reg32_write(RLE_CISR,sizeof(rleBitmap));//compressed size of the image
reg32_write_mask(RLE_DICR,320<<RLE_DICR_X_SHIFT,RLE_DICR_X_MASK); //image width
reg32_write_mask(RLE_DICR,240<<RLE_DICR_Y_SHIFT,RLE_DICR_Y_MASK); //image height
reg32_write_mask(RLE_ICR,0<<RLE_ICR_WIDTH_SHIFT,RLE_ICR_WIDTH_MASK); //8bpp format
//decompress full size
reg32_write_mask(RLE_SPCR,0<<RLE_SPCR_X_SHIFT,RLE_SPCR_X_MASK);  //start x pixel (origin)
reg32_write_mask(RLE_SPCR,0<<RLE_SPCR_Y_SHIFT,RLE_SPCR_Y_MASK);  //start y pixel (origin)
reg32_write_mask(RLE_EPCR,320<<RLE_EPCR_X_SHIFT,RLE_EPCR_X_MASK); //final x pixel (width)
reg32_write_mask(RLE_EPCR,240<<RLE_EPCR_Y_SHIFT,RLE_EPCR_Y_MASK); //final y pixel (height)
reg32clrbit(RLE_MCR,RLE_MCR_MDIS_SHIFT);//enable module
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductors products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-complaint and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to http://www.freescale.com/epp.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2013 Freescale Semiconductor, Inc.

Document Number: AN4672
Rev. 0, 01/2013