

VS1005 BITMAP GRAPHICS

VS1005g

All information in this document is provided as-is without warranty. Features are subject to change without notice.

Revision History			
Rev.	Date	Author	Description
1.00	2021-01-28	HH	Initial release.

Contents

VS1005 BitMap Graphics Front Page	1
Table of Contents	2
1 Introduction	4
2 Disclaimer	5
3 Definitions	5
4 Overview	6
5 Requirements	7
6 Testing the RenderEx.dl3 example	8
7 Using vs1005gfxgen.exe	9
7.1 The VS1005GfxGen Command File Format	10
7.1.1 #	10
7.1.2 verbose n (n = 0...3)	10
7.1.3 maxbits n (n = 1...8)	10
7.1.4 palettehints n (n = 0...2)	10
7.1.5 gfx FileName.ppm	10
7.1.6 meta text	10
7.2 The Supported PPM Graphics Format	11
7.3 PaletteHints Explained	12
7.3.1 PaletteHints 0	13
7.3.2 PaletteHints 1	14
7.3.3 PaletteHints 2	15
8 The VS1005 BitMap Graphics Format	16
8.1 Example VS1005 BitMap File	17
9 Latest Document Version Changes	20
10 Contact Information	21

List of Figures

1	Example screen of RenderEx	8
2	4x2 Pixel PPM Image	11
3	How a PPM pixel's bits are interpreted when palettehints = 0	13
4	How a PPM pixel's bits are interpreted when palettehints = 1	14
5	How a PPM pixel's bits are interpreted when palettehints = 2	15

1 Introduction

This document explains how to generate and render bitmapped graphics using a VS1005 board with an LCD.

After the disclaimer and definitions in Chapters 2 and 3, an overview of the graphics system is presented in Chapter 4, *Overview*, followed by requirements in Chapter 5, *Requirements*.

The VSOS bitmap rendering program *RenderEx.dl3* introduced in Chapter 6, *Testing the RenderEx.dl3 Example*.

The details on how to use the VS1005 graphics generation program, *vs1005gfxgen.exe*, is shown in Chapter 7, *Using vs1005gfxgen.exe*.

For those interested in making their own graphics generators, the graphics format itself is described in Chapter 8, *The VS1005 Bitmapped Graphics Format*.

The document ends with Chapter 9, *Latest Document Version Changes*, and Chapter 10, *Contact Information*.

2 Disclaimer

VLSI Solution makes everything it can to make this documentation as accurate as possible. However, no warranties or guarantees are given for the correctness of this documentation.

3 Definitions

DSP Digital Signal Processor.

I-mem Instruction Memory.

LSB Least Significant (8-bit) Byte.

LSW Least Significant (16-bit) Word.

MSB Most Significant (8-bit) Byte.

MSW Most Significant (16-bit) Word.

VS_DSP⁴ VLSI Solution's DSP core.

VSIDE VLSI Solution's Integrated Development Environment.

VSOS VLSI Solution's Operating System.

X-mem X Data Memory.

Y-mem Y Data Memory.

4 Overview

The VS1005 BitMap Graphics package makes it possible to combine several 1- to 8-bit bitmap images into a single file on a PC, each bitmap with a unique palette. Then, by using this file, it is possible to render graphics on an LCD display connected to the VS1005.

5 Requirements

To test the graphics system in this document, you need to have the following building blocks:

- VS1005g Developer Board or similar, equipped with a daughterboard with either 1.77" or 2.88" LCD.
- Latest version of VSOS installed (at least v3.60).
- Versions v3.62 (dated 2021-01-28 or later) for the appropriate LCD driver.
- VLSI Solution's Integrated Development Environment VSIDE.
- The RenderEx example solution for VSIDE.
- UART or USB->UART cable connected between DevBoard and PC for using the UART interface. Data speed is 115200 bps, format is 8N1.
- Your favorite UART Terminal Emulation program installed on the PC. Read document "VS1005 VSOS Shell" for further details.

When all of this is in order, you are ready to test the graphics package.

6 Testing the RenderEx.dl3 example

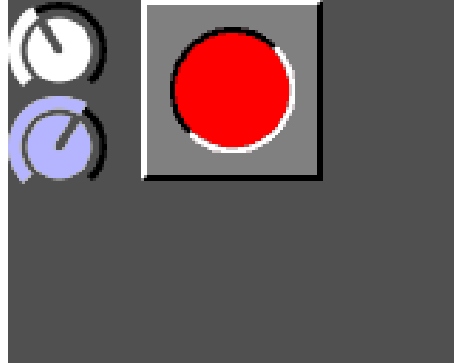


Figure 1: Example screen of RenderEx

To test the RenderEx.dl3 example program, open it in VSIDE, and select Build -> Build Solution. Alternatively, you can copy RenderEx.dl3 to S:SYS/ and RenderEx.v5g to S: .

Then, enter RenderEx command in the VSOS Shell:

```
S:>RenderEx
106 gfx bitmaps
```

Push buttons between 1 and 9 to test, Ctrl-C to exit

After you have pushed for example key '4', you should see approximately similar graphics to what is shown in Figure 1. The know should be animating clockwise, and the right hand symbol should change accoring to your keypresses. Refer on the source code of RenderEx to see how to render the graphics and how to modify the palette. See also the post-processing options of the RenderEx project.

7 Using vs1005gfxgen.exe

VS1005GfxGen.exe can read PPM files (24-bit P6 subformat), convert them into palette-based bitmat images, then write the resulting graphics in one combination file.

```
% ./vs1005gfxgen.exe -h
VS1005GfxGen 1.02 2021-01-27 VLSI Solution
```

License: Usage and modification absolutely free as long as there is a VLSI Solution IC involved.

Usage:

```
Z:\users\leopold\src\mingw\VS1005GfxGen\vs1005gfxgen.exe [-p|+p] [-c cFil] [-o oFil]
```

```
-c cFil Read commands / files from file cFil
-o oFil Output graphics file is oFil (default: gfx.v5g)
-p|+p Read/Don't read palette hints for 4-bit palette. (default: off)
      Note: Can be changed within command line.
```

The way VS1005GfxGen is used in the RenderEx demo post-processing steps is as follows:

```
vs1005gfxgen.exe -o RenderEx.v5g -c RenderEx.txt
```

This will read the command file RenderEx.txt, and write the resulting graphics in RenderEx.v5g. The output of VS1005GfxGen looks e.g. as follows:

```
0: knob100/KNOB1000.PPM          36x31, 16col/4bpp, 590 B
1: knob100/KNOB1001.PPM          36x31, 16col/4bpp, 590 B
2: knob100/KNOB1002.PPM          36x31, 16col/4bpp, 590 B
3: knob100/KNOB1003.PPM          36x31, 16col/4bpp, 590 B
[...]
99: knob100/KNOB1099.PPM         36x31, 16col/4bpp, 590 B
100: knob100/KNOB1100.PPM         36x31, 16col/4bpp, 590 B
101: recsym08/wait064.ppm        64x64, 8col/3bpp, 1552 B
102: recsym08/pause064.ppm       64x64, 7col/3bpp, 1552 B
103: recsym08/play064.ppm        64x64, 8col/3bpp, 1552 B
104: recsym08/rec064.ppm         64x64, 8col/3bpp, 1552 B
105: recsym08/stop064.ppm        64x64, 6col/3bpp, 1552 B
106 bitmaps to RenderEx.v5g, 68072 bytes.
```

From this output you can see the file indices (in this case, 0...105), and for each file its name, size, amount of individual colour found in the file, as well as the bit depth required to store the file, and finally the size of the encoded data in bytes.

7.1 The VS1005GfxGen Command File Format

The command file consists of lines of the following format:

7.1.1

Comment lines beginning with the hash symbol as ignored.

7.1.2 **verbose n (n = 0...3)**

Set verbosity. Higher is more verbose.

7.1.3 **maxbits n (n = 1...8)**

Set maximum amount of bits per pixel for following bitmaps. May be used to make sure there are no graphics with more bit-depth than expected.

7.1.4 **palettehints n (n = 0...2)**

Set palettehints. See Chapter 7.3 for details.

7.1.5 **gfx FileName.ppm**

Include graphics file FileName.ppm into the bitmap package. PPM file can be generated for example with the open source graphics editor package Gimp.

For details on the PPM format supported, and special requirements, see Chapter 7.2.

7.1.6 **meta text**

meta can be used to add metadata to the file, like copyright notices or similar.

7.2 The Supported PPM Graphics Format

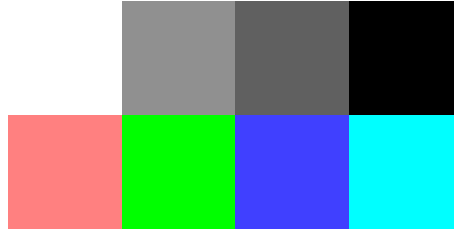


Figure 2: 4x2 Pixel PPM Image

The 24-bit portable pixmap format (PPM / P6) supported by VS1005GfxGen, and output by default by e.g. Gimp when PPM output is selected, looks for example as follows:

```
00000000  50 36 0a 34 0a 32 0a 32  35 35 0a ff ff ff 90 90 |P6.4.2.255.   ..|
00000010  90 60 60 60 00 00 00 ff  80 80 00 ff 00 40 40 ff |.‘‘...   .   |
00000020  00 ff ff                                     |.  |
00000023
```

This 4x2 pixel image is presented in Figure 2.

Note that while PPM is a 24-bit format, you may not use more than 256 distinct colour, and to keep files and rendering times small, should always try to keep the number of colours to the minimum required for the symbol in question, e.g. 16.

For more information on the PPM P6 format, see <https://en.wikipedia.org/wiki/Netpbm>.

7.3 PaletteHints Explained

Normally, as VS1005GfxGen reads a PPM file, it will assign palette entries as it sees new colours.

However, in some cases it may be useful to be able to determine into which palette entries the PPM image colours go. This can be used for example for highlighting purposes: a number of menu items may have an apparently similar colour, but by changing the palette on the fly while rendering the subject, it may be possible to highlight the items in various ways.

PaletteHints is a way to encode palette index information into the actual colours of a PPM file.

7.3.1 PaletteHints 0

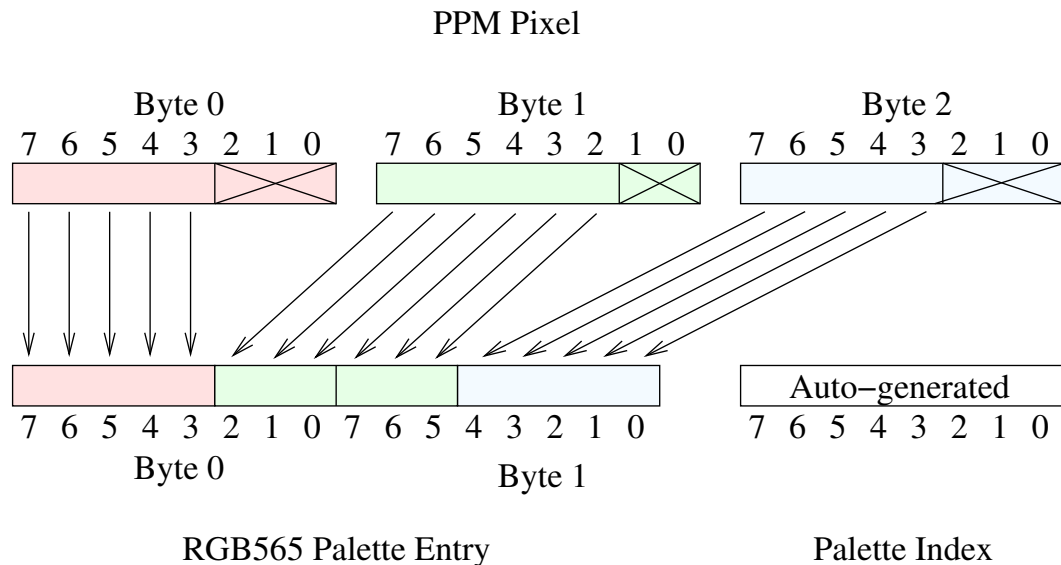


Figure 3: How a PPM pixel's bits are interpreted when palettehints = 0

Interpretation of PPM Pixels when palettehints = 0			
PPM RGB888	RGB-565	Palette Index	Description
00 00 00	00 00	0	First (black) pixel, store new Palette Index 0.
07 03 06	00 00	0	RGB565 is the same, get same Palette Index.
FF FF FF	FF FF	1	White pixel, store new Palette Index 1.
FF 00 01	F8 00	2	Red pixel, store new Palette Index 2.
FF FF FF	FF FF	1	Another white pixel, use existing palette index 1.
00 FF 08	07 E1	3	Green pixel, store new Palette Index 3.
00 00 F3	00 1E	4	Blue pixel, store new Palette Index 4.
FF 00 02	F8 00	2	Another red pixel, use existing palette index 2.

If palettehints is clear, then VS1005GFXGen examines each pixels in a PPM file as shown in Figure 3. It decodes bits R(7:3), G(7:2), and B(7:3) into an RGB565 entry. Whenever it encounters a new colour in the RGB565 space, it generates a new Palette Index, starting from 0, and stores that RGB565 entry there.

Note that bits B(2:0), R(1:0), and R(2:0) are ignored.

7.3.2 PaletteHints 1

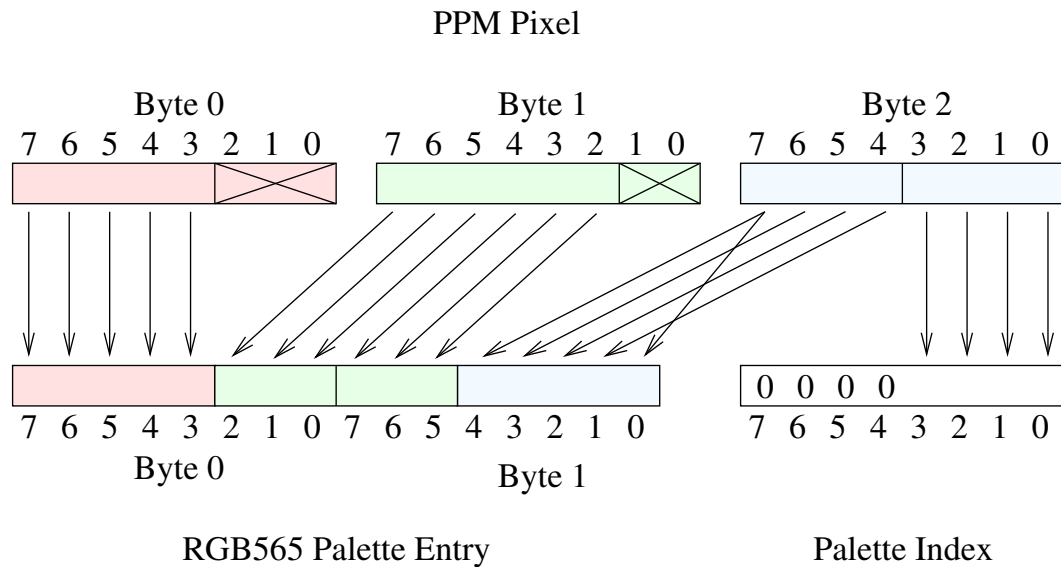


Figure 4: How a PPM pixel's bits are interpreted when palettehints = 1

Interpretation of PPM Pixels when palettehints = 1			
PPM RGB888	RGB-565	Palette Index	Description
00 00 00	00 00	0	Store black to Palette Index 0.
07 03 06	00 00	7	Store black to Palette Index 7.
FF FF FF	FF FF	f	Store white to Palette Index f.
FF 00 01	F8 00	1	Store red to Palette Index 1.
FF FF FF	FF FF	f	Another white pixel using Palette Index 7.
00 FF 08	07 E0	3	Store green to Palette Index 8.
00 00 F3	00 1F	8	Store blue to Palette Index 3.
FF 00 02	F8 00	2	Store red to Palette Index 2.

If palettehints is set to 1, then VS1005GFXGen examines each pixels in a PPM file as shown in Figure 4. It decodes bits B(3:0) into a 4-bit Palette Index. Whenever it encounters a new Palette Index, it stores the RGB565 Palette Entry to the Palette Index.

Note that bits B(2:0), and R(1:0) are ignored.

Note that there are only 4 bits available for the Palette Entry from the Blue channel, B(7:4). Because of this, the PPM Pixel's B(7) is copied to the RGB565 Palette Entry's B(0).

7.3.3 PaletteHints 2

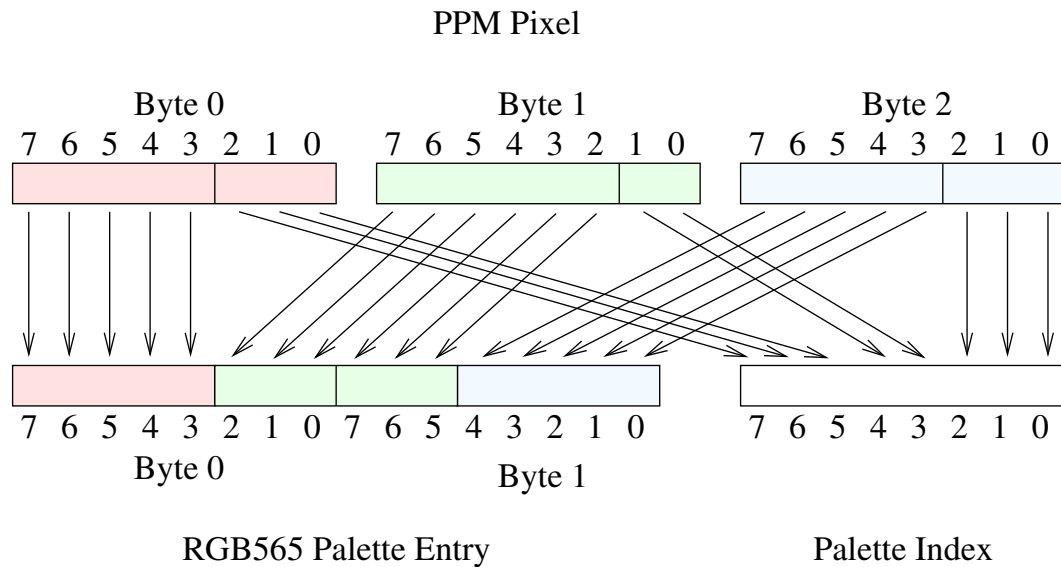


Figure 5: How a PPM pixel's bits are interpreted when palettehints = 2

Interpretation of PPM Pixels when palettehints = 2			
PPM RGB888	RGB-565	Palette Index	Description
00 00 00	00 00	0	Store black to palette index 0.
07 03 06	00 00	FE	Store black to palette index 254.
FF FF FF	FF FF	FF	Store white to palette index 255.
FF 00 01	F8 00	E1	Store red to palette index 241.
FF FF FF	FF FF	FF	Another white pixel using Palette Index 255.
00 FF 08	07 E1	18	Store green to palette index 24.
00 00 F3	00 1E	3	Store blue to palette index 3.
FF 00 02	F8 00	2	Store red to palette index 2.

If palettehints is set to 2, then VS1005GFXGen examines each pixels in a PPM file as shown in Figure 4. It decodes bits B(3:0), R(2:0), and B(3:0) into an 8-bit Palette Index. Whenever it encounters a new Palette Index, it stores the RGB565 Palette Entry to the Palette Index.

8 The VS1005 BitMap Graphics Format

The VS1005 BitMap Graphics Format (recommended file suffix: .V5G) is as follows.

Note that all numbers are store in the big-endian format, i.e. most significant bytes first.

VS1005 BitMap File				
Name	Offset	Size	Number	Description
Header	0	8	1	
GfxInfo	8	6×nGfx	nGfx	Data needed to decode bitmaps
GfxData	GfxDataA[n]		nGfx	Palettes and bitmaps
Meta	MetaAddr	MetaBytes		Data not needed to decode file

VS1005 BitMap Header				
Name	Offset	Size	Number	Description
ID	0	2	1	0x1c 0x9a
nGfx	2	2	1	Number of bitmaps in file
MetaAddr	4	4	1	

VS1005 BitMap GfxInfo[n]				
Name	Offset	Size	Number	Description
GfxDataA	0	3	1	Position in file for pixel data
Width	3	1	1	Bitmap width in pixels
Height	4	1	1	Bitmap height in pixels
Depth	5	1	1	Bitmap depth in bits per pixel

VS1005 BitMap Meta				
Name	Offset	Size	Number	Description
MetaBytes	0	4	1	
MetaData	4	MetaBytes	1	NUL-terminated strings.

GfxData[n] consists of 16-bit big-endian encoded words. For each pixel GfxInfo[n].Depth least significant bits are read, an appropriate palette entry is applied, and the result is sent in the RGB565 format. If the bit depth isn't a power of two, words do not align with data.

Example 1:

if GfxInfo[n].Depth=3, and 5 pixels have been read from a 16-bit data word, there is 1 bit left. In this case, read the next word, shift it up by the amount of bits still left (in this case 1), then continue decoding.

Because of the way the pixels are encoded, bitmaps are always padded up to a boundary of 16 bits.

Example 2:

8 4-bit pixels with values of 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7 are encoded in the file as:

0x32 0x10 0x76 0x54

Example 3:

4 5-bit pixels with values of 0x0, 0x1, 0x2, 0x3 are encoded in the file as:
0x88 0x20 0x8a 0x41 0x00 0x39

The size of GfxData[n] is padded to the next higher 16 bits.

8.1 Example VS1005 BitMap File

Let's assume we have the following two (very small) existing .PPM files that we want to combine to a VS1005 BitMap File.

File 1, example/4col.ppm:

```
000000 50 36 0a 32 20 32 0a 32 35 35 0a 00 00 00 ff 00 |P6.2 2.255.....|
000010 00 00 ff 00 00 00 ff |.....|
```

File 2, example/16col.ppm:

```
000000 50 36 0a 38 20 32 0a 32 35 35 0a 00 00 00 00 00 |P6.8 2.255.....|
000010 ff 00 ff 00 00 ff ff ff 00 00 ff 00 ff ff ff 00 |.......|
000020 ff ff ff 10 10 10 20 20 20 30 30 30 40 40 40 50 |... 000@@@P|
000030 50 50 60 60 60 80 80 80 c0 c0 c0 |PP'''...|
```

We also have the following command file Example.txt:

```
verbose 3
gfx example/4col.ppm
gfx example/16col.ppm
```

Then we run the following command:

```
vs1005gfxgen.exe -o Example.v5g -c Example.txt
```

The output of the generator is as follows:

```
New unique: 00 00 00 0000
New unique: ff 00 00 f800
New unique: 00 ff 00 07e0
New unique: 00 00 ff 001f
0: example/4col.ppm 2x2, 4col/2bpp, 10 B
New unique: 00 00 00 0000
New unique: 00 00 ff 001f
New unique: 00 ff 00 07e0
New unique: 00 ff ff 07ff
New unique: ff 00 00 f800
New unique: ff 00 ff f81f
New unique: ff ff 00 ffe0
New unique: ff ff ff ffff
```

```
New unique: 10 10 10 1082
New unique: 20 20 20 2104
New unique: 30 30 30 3186
New unique: 40 40 40 4208
New unique: 50 50 50 528a
New unique: 60 60 60 630c
New unique: 80 80 80 8410
New unique: c0 c0 c0 c618
```

```
1: example/16col.ppm                8x2, 16col/4bpp, 40 B
2 bitmaps to Example.v5g, 99 bytes.
```

The resulting file looks as follows:

```
00000000 1c 9a 00 02 00 00 00 46 00 00 14 02 02 02 00 00 | .....F.....|
00000010 1e 08 02 04 00 00 f8 00 07 e0 00 1f 00 e4 00 00 | .....ø...ä...ä...|
00000020 00 1f 07 e0 07 ff f8 00 f8 1f ff e0 ff ff 10 82 | ...ä.ßø.ø.ßàßß...|
00000030 21 04 31 86 42 08 52 8a 63 0c 84 10 c6 18 32 10 | !.1.B.R.c...Æ.2.|
00000040 76 54 ba 98 fe dc 00 00 00 19 44 41 54 45 3d 32 | vTŽ.pÛ....DATE=2|
00000050 30 32 31 2d 30 31 2d 32 38 20 31 30 3a 33 36 3a | 021-01-28 10:36:|
00000060 30 35 00                                     |05.|
```

Broken down to its components, the file looks as follows:

Offset 0x00:

```
0x1c 0x9a                                ID 0x1c9a
```

Offset 0x02:

```
0x00 0x02                                NUMBEROFICONS = 2
```

Offset 0x04:

```
0x00 0x00 0x00 0x46                      METAADDR = 0x46
```

Offset 0x08:

```
0x00 0x00 0x14 0x02 0x02 0x02  ICONDESCR0: Data at 0x14, W=2, H=2, BPP=2
```

Offset 0x0e:

```
0x00 0x00 0x1e 0x08 0x02 0x04  ICONDESCR0: Data at 0x1e, W=8, H=2, BPP=4
```

Offset 0x14:

```
0x00 0x00                                PALETTE0_0: R= 0, G= 0, B= 0
0xf8 0x00                                PALETTE0_1: R=31, G= 0, B= 0
0x07 0xe0                                PALETTE0_2: R= 0, G=63, B= 0
0x00 0x1f                                PALETTE0_3: R= 0, G= 0, B=31
```

Offset 0x01d:

0x00 0xe4 BITMAP pixels 0x0, 0x1, 0x2, 0x3 (black, red, green, blue)
Note: only half of the 16 bits were needed to encode 4 2-bit pixels.

Offset 0x1e:

0x00 0x00	PALETTE1_0: R= 0, G= 0, B= 0
0x00 0x1f	PALETTE1_1: R= 0, G= 0, B=31
0x07 0xe0	PALETTE1_2: R= 0, G=63, B= 0
0x07 0xff	PALETTE1_3: R= 0, G=63, B=31
0xf8 0x00	PALETTE1_4: R=31, G= 0, B= 0
0xf8 0x1f	PALETTE1_5: R=31, G= 0, B=31
0xff 0xe0	PALETTE1_6: R=31, G=63, B= 0
0xff 0xff	PALETTE1_7: R=31, G=63, B=31
0x10 0x82	PALETTE1_8: R= 2, G= 4, B= 2
0x21 0x04	PALETTE1_9: R= 4, G= 8, B= 4
0x31 0x86	PALETTE1_A: R= 6, G=12, B= 6
0x42 0x08	PALETTE1_B: R= 8, G=16, B= 8
0x52 0x8a	PALETTE1_C: R=10, G=20, B=10
0x63 0x0c	PALETTE1_D: R=12, G=24, B=12
0x84 0x10	PALETTE1_E: R=16, G=32, B=16
0xc6 0x18	PALETTE1_F: R=24, G=48, B=24

Offset 0x3e:

0x32 0x10 BITMAP pixels 0x0, 0x1, 0x2, 0x3 (black, blue, green, cyan)
0x76 0x54 BITMAP pixels 0x4, 0x5, 0x6, 0x7 (red, magenta, yellow, white)
0xba 0x98 BITMAP pixels 0x8, 0x9, 0xa, 0xb (greyscale, dark to middle)
0xfe 0xdc BITMAP pixels 0xc, 0xd, 0xe, 0xf (greyscale, middle to light)

Offset 0x46:

0x00 0x00 0x00 0x19 METABYTES = 0x19

Offset 0x4A:

0x44 0x41 0x54 0x45 0x3d 0x32 0x30 0x32 "DATE=202"
0x31 0x2d 0x30 0x31 0x2d 0x32 0x38 0x20 "1-01-28 "
0x31 0x30 0x3a 0x33 0x36 0x3a 0x30 0x35 "10:36:05"
0x00 "\0"

9 Latest Document Version Changes

This chapter describes the latest changes to this document.

Version 1.00, 2021-01-28

First release.

10 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

URL: <http://www.vlsi.fi/>
Phone: +358-50-462-3200
Commercial e-mail: sales@vlsi.fi

For technical support or suggestions regarding this document, please participate at
<http://www.vsdsp-forum.com/>

For confidential technical discussions, contact
support@vlsi.fi