



S1D13706 Embedded Memory LCD Controller

Programming Notes and Examples

Document Number: X31B-G-003-03

Copyright © 2001 Epson Research and Development, Inc. All Rights Reserved.

Information in this document is subject to change without notice. You may download and use this document, but only for your own use in evaluating Seiko Epson/EPSON products. You may not modify the document. Epson Research and Development, Inc. disclaims any representation that the contents of this document are accurate or current. The Programs/Technologies described in this document may contain material protected under U.S. and/or International Patent laws.

EPSON is a registered trademark of Seiko Epson Corporation. All other trademarks are the property of their respective owners.

THIS PAGE LEFT BLANK

Table of Contents

1	Introduction	9
2	Initialization	10
3	Memory Models	14
3.1	Display Buffer Location	14
3.2	Memory Organization for One Bit-per-pixel (2 Colors/Gray Shades)	14
3.3	Memory Organization for Two Bit-per-pixel (4 Colors/Gray Shades)	15
3.4	Memory Organization for Four Bit-per-pixel (16 Colors/Gray Shades)	15
3.5	Memory Organization for 8 Bpp (256 Colors/64 Gray Shades)	16
3.6	Memory Organization for 16 Bpp (65536 Colors/64 Gray Shades)	16
4	Look-Up Table (LUT)	17
4.1	Registers	17
4.1.1	Look-Up Table Write Registers	17
4.1.2	Look-Up Table Read Registers	18
4.2	Look-Up Table Organization	19
4.2.1	Gray Shade Modes	20
4.2.2	Color Modes	22
5	Power Save Mode	26
5.1	Overview	26
5.2	Registers	27
5.2.1	Power Save Mode Enable	27
5.2.2	Memory Controller Power Save Status	27
5.3	Enabling Power Save Mode	28
5.4	Disabling Power Save Mode	28
6	LCD Power Sequencing	29
6.1	Enabling the LCD Panel	30
6.2	Disabling the LCD Panel	30
7	SwivelView'	31
7.1	Registers	32
7.2	Examples	33
7.3	Limitations	36
7.3.1	SwivelView 0° and 180°	36
7.3.2	SwivelView 90° and 270°	36
8	Picture-In-Picture Plus	37
8.1	Concept	37
8.2	Registers	37
8.3	Picture-In-Picture-Plus Examples	48

8.3.1	SwivelView 0° (Landscape Mode)	48
8.3.2	SwivelView 90°	51
8.3.3	SwivelView 180°	54
8.3.4	SwivelView 270°	57
8.4	Limitations	60
8.4.1	SwivelView 0° and 180°	60
8.4.2	SwivelView 90° and 270°	60
9	Identifying the S1D13706	61
10	Hardware Abstraction Layer (HAL)	62
10.1	API for 13706HAL	62
10.2	Initialization	65
10.2.1	General HAL Support	68
10.2.2	Advance HAL Functions	75
10.2.3	Surface Support	76
10.2.4	Register Access	80
10.2.5	Memory Access	82
10.2.6	Color Manipulation	84
10.2.7	Virtual Display	87
10.2.8	Drawing	89
10.2.9	Register/Display Memory	95
10.3	Porting LIBSE to a new target platform	96
10.3.1	Building the LIBSE library for SH3 target example	97
11	Sample Code	98

List of Tables

Table 2-1: Example Register Values	11
Table 4-1: Look-Up Table Configurations	19
Table 4-2: Suggested LUT Values for 1 Bpp Gray Shade	20
Table 4-3: Suggested LUT Values for 4 Bpp Gray Shade	20
Table 4-4: Suggested LUT Values for 4 Bpp Gray Shade	21
Table 4-5: Suggested LUT Values for 1 bpp Color	22
Table 4-6: Suggested LUT Values for 2 bpp Color	22
Table 4-7: Suggested LUT Values to Simulate VGA Default 16 Color Palette	23
Table 4-8: Suggested LUT Values to Simulate VGA Default 256 Color Palette	24
Table 7-1: SwivelView Enable Bits	32
Table 8-1: 32-bit Address Increments for Color Depth	41
Table 8-2: 32-bit Address Increments for Color Depth	42
Table 8-3: 32-bit Address Increments for Color Depth	44
Table 8-4: 32-bit Address Increments for Color Depth	46
Table 10-1: HAL Functions	62

THIS PAGE LEFT BLANK

List of Figures

Figure 3-1: Pixel Storage for 1 Bpp in One Byte of Display Buffer	14
Figure 3-2: Pixel Storage for 2 Bpp in One Byte of Display Buffer	15
Figure 3-3: Pixel Storage for 4 Bpp in One Byte of Display Buffer	15
Figure 3-4: Pixel Storage for 8 Bpp in One Byte of Display Buffer	16
Figure 3-5: Pixel Storage for 16 Bpp in Two Bytes of Display Buffer	16
Figure 8-1: Picture-in-Picture Plus with SwivelView disabled	37
Figure 8-2: Picture-in-Picture Plus with SwivelView disabled	48
Figure 8-3: Picture-in-Picture Plus with SwivelView 90° enabled	51
Figure 8-4: Picture-in-Picture Plus with SwivelView 180° enabled	54
Figure 8-5: Picture-in-Picture Plus with SwivelView 270° enabled	57
Figure 10-1: Components needed to build 13706 HAL application	96

THIS PAGE LEFT BLANK

1 Introduction

This guide provides information on programming the S1D13706 Embedded Memory LCD Controller. Included are algorithms which demonstrate how to program the S1D13706. This guide discusses Power-on Initialization, Panning and Scrolling, LUT initialization, LCD Power Sequencing, SwivelView™, Picture-In-Picture Plus, etc. The example source code referenced in this guide is available on the web at www.eea.epson.com or www.erd.epson.com.

This guide also introduces the Hardware Abstraction Layer (HAL), which is designed to simplify the programming of the S1D13706. Most SED135x and SED137x products have HAL support, thus allowing OEMs to do multiple designs with a common code base.

This document will be updated as appropriate. Please check the Epson Electronics America Website at www.eea.epson.com for the latest revision of this document and source before beginning any development.

We appreciate your comments on our documentation. Please contact us via email at documentation@erd.epson.com.

2 Initialization

This section describes how to initialize the S1D13706. Sample code for performing initialization of the S1D13706 is provided in the file **init13706.c** which is available on the internet at www.eea.epson.com or www.erd.epson.com.

S1D13706 initialization can be broken into the following steps.

1. Disable the display using the Display Blank bit (set REG[70h] bit 7 = 1).
2. If the system implementation uses a clock chip instead of a fixed oscillator, program the clock chip. For example, the S5U13706 Evaluation Board uses a Cypress clock chip.
3. Set all registers to initial values. Table 2-1; “Example Register Values” contains the correct values for an example panel discussed below.
4. Program the Look-Up Table (LUT) with color values. For details on programming the LUT, see Section 4, “Look-Up Table (LUT)” on page 17.
5. Power-up the LCD panel. For details on powering-up the LCD panel, see Section 5.4, “Disabling Power Save Mode” on page 28.
6. Enable the display using the Display Blank bit (set REG[70h] bit 7 = 0).
7. Clear the display buffer (if required).

Note

The simplest way to generate initialization tables for the S1D13706 is to use the utility program 13706CFG.EXE which generates a header file that can be used by the operating system or the HAL. Otherwise modify the **init13706.c** file directly.

The following table represents the sequence and values written to the S1D13706 registers to control a configuration with these specifications.

- 320x240 color single passive LCD @ 70Hz.
- 8-bit data interface, format 2.
- 8 bit-per-pixel (bpp) color depth - 256 colors.
- 50MHz input clock for CLKI.
- MCLK = BCLK = CLKI = 50MHz.
- PCLK = CLKI ÷ 8 = 6.25MHz.

Note

On the S5U13706B00C evaluation board, CNF[7:6] must be set to 00.

Table 2-1: Example Register Values

Register	Value (Hex)	Value (Binary)	Description	Notes
Clock Configuration (MCLK, BCLK, PCLK)				
04h	00	0000 0000	Sets BCLK to MCLK divide to 1:1	
05h	43	0100 0011	Sets PCLK = (PCLK source ÷ 8) and the PCLK source = CLKI2	
Panel Setting Configuration				
10h	D0	1101 0000	Selects the following: <ul style="list-style-type: none"> • panel data format = 2 • color/mono panel = color • panel data width = 8-bit • active panel resolution = don't care • panel type = STN 	
11h	00	0000 0000	MOD rate = don't care	
12h	2B	0010 1011	Sets the horizontal total	
14h	27	0010 0111	Sets the horizontal display period	
16h	00	0000 0000	Sets the horizontal display period start position	
17h	00	0000 0000		
18h	FA	1111 1010	Sets the vertical total	
19h	00	0000 0000		
1Ch	EF	1110 1111	Sets the vertical display period	
1Dh	00	0000 0000		
1Eh	00	0000 0000	Sets the vertical display period start position	
1Fh	00	0000 0000		
20h	87	1000 0111	Sets the FPLINE pulse polarity and FPLINE pulse width	
22h	00	0000 0000	Sets the FPLINE pulse start position	
23h	00	0000 0000		
24h	80	1000 0000	Sets the FPFRAME pulse polarity and FPFRAME pulse width	
26h	01	0000 0001	Sets the FPFRAME pulse start position	
27h	00	0000 0000		

Table 2-1: Example Register Values (Continued)

Register	Value (Hex)	Value (Binary)	Description	Notes
Display Mode Setting Configuration				
70h	83	1000 0011	Selects the following: <ul style="list-style-type: none">• display blank = screen is blanked• dithering = enabled• hardware video invert = disabled• software video invert = video data is not inverted• color depth = 8 bpp	
71h	00	0000 0000	Selects the following: <ul style="list-style-type: none">• display data word swap = disabled• display data byte swap = disabled• sub-window enable = disabled• SwivelView Mode = not rotated	
74h	00	0000 0000	Sets the main window display start address	
75h	00	0000 0000		
76h	00	0000 0000		
78h	50	0101 0000	Sets the main window line address offset	
79h	00	0000 0000		
7Ch	00	0000 0000	Sets the sub-window display start address	
7Dh	00	0000 0000		
7Eh	00	0000 0000		
80h	50	0101 0000	Sets the sub-window line address offset	
81h	00	0000 0000		
84h	00	0000 0000	Sets the sub-window X start position	
85h	00	0000 0000		
88h	00	0000 0000	Sets the sub-window Y start position	
89h	00	0000 0000		
8Ch	4F	0100 1111	Sets the sub-window X end position	
8Dh	00	0000 0000		
90h	EF	1110 1111	Sets the sub-window Y end position	
91h	00	0000 0000		
Miscellaneous Register Configuration				
A0h	00	0000 0000	Disables power save mode	
A1h	00	0000 0000	Reserved register. Must be written 00h.	
A2h	00	0000 0000	Set reserved bit 7 to 0	
A3h	00	0000 0000	Reserved register. Must be written 00h.	
A4h	00	0000 0000	Clears the scratch pad registers	
A5h	00	0000 0000		
GPIO Pin Configuration				
A8h	00	0000 0000	GPIO[6:0] pins are configured as input pins	
A9h	80	1000 0000	Bit 7 set to 1 to enable GPIO pin inputs.	

Table 2-1: Example Register Values (Continued)

Register	Value (Hex)	Value (Binary)	Description	Notes
ACh	00	0000 0000	GPIO[6:0] pins are driven low	Bit 7 controls the LCD bias power for the panel on the S5U13706B00C.
ADh	00	0000 0000	Set the GPO control bit to low	
PWM Clock and CV Pulse Configuration				
B0h	00	0000 0000	Selects the following: <ul style="list-style-type: none">• PWMOUT pin is software controlled• PWM Clock circuitry is disabled• CVOUT pin is software controlled• CV Pulse circuitry is disabled	For this example the divides are not required. For this example, the burst length is not required.
B1h	00	0000 0000	Sets the PWM Clock and CV Pulse divides	
B2h	00	0000 0000	Sets the CV Pulse Burst Length	
B3h	00	0000 0000	Sets the PWMOUT signal to always low	

3 Memory Models

The S1D13706 contains a display buffer of 80K bytes and supports color depths of 1, 2, 4, 8, and 16 bit-per-pixel. For each color depth, the data format is packed pixel.

Packed pixel data may be envisioned as a stream of pixels. In this stream, pixels are packed adjacent to each other. If a pixel requires four bits, then it is located in the four most significant bits of a byte. The pixel to the immediate right on the display occupies the lower four bits of the same byte. The next two pixels to the immediate right are located in the following byte, etc.

3.1 Display Buffer Location

The S1D13706 display buffer is 80K bytes of embedded SRAM. The display buffer is memory mapped and is accessible directly by software. The memory block location assigned to the S1D13706 display buffer varies with each individual hardware platform.

For further information on the display buffer, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

For further information on the S1D13706 Evaluation Board, see the *S5U13706B00C Evaluation Board Rev. 1.0 User Manual*, document number X31B-G-004-xx.

3.2 Memory Organization for One Bit-per-pixel (2 Colors/Gray Shades)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pixel 0	Pixel 1	Pixel 2	Pixel 3	Pixel 4	Pixel 5	Pixel 6	Pixel 7

Figure 3-1: Pixel Storage for 1 Bpp in One Byte of Display Buffer

At a color depth of 1 bpp, each byte of display buffer contains eight adjacent pixels. Setting or resetting any pixel requires reading the entire byte, masking out the unchanged bits and setting the appropriate bits to 1.

One bit pixels provide 2 gray shades/color possibilities. For monochrome panels the gray shades are generated by indexing into the first two elements of the green component of the Look-Up Table (LUT). For color panels the 2 colors are derived by indexing into the first 2 positions of the LUT.

3.3 Memory Organization for Two Bit-per-pixel (4 Colors/Gray Shades)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pixel 0 Bits 1-0		Pixel 1 Bits 1-0		Pixel 2 Bits 1-0		Pixel 3 Bits 1-0	

Figure 3-2: Pixel Storage for 2 Bpp in One Byte of Display Buffer

At a color depth of 2 bpp, each byte of display buffer contains four adjacent pixels. Setting or resetting any pixel requires reading the entire byte, masking out the unchanged bits and setting the appropriate bits to 1.

Two bit pixels provide 4 gray shades/color possibilities. For monochrome panels the gray shades are generated by indexing into the first 4 elements of the green component of the Look-Up Table (LUT). For color panels the 4 colors are derived by indexing into the first 4 positions of the LUT.

3.4 Memory Organization for Four Bit-per-pixel (16 Colors/Gray Shades)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pixel 0 Bits 3-0				Pixel 1 Bits 3-0			

Figure 3-3: Pixel Storage for 4 Bpp in One Byte of Display Buffer

At a color depth of 4 bpp, each byte of display buffer contains two adjacent pixels. Setting or resetting any pixel requires reading the entire byte, masking out the upper or lower nibble (4 bits) and setting the appropriate bits to 1.

Four bit pixels provide 16 gray shades/color possibilities. For monochrome panels the gray shades are generated by indexing into the first 16 elements of the green component of the Look-Up Table (LUT). For color panels the 16 colors are derived by indexing into the first 16 positions of the LUT.

3.5 Memory Organization for 8 Bpp (256 Colors/64 Gray Shades)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pixel 0 Bits 7-0							

Figure 3-4: Pixel Storage for 8 Bpp in One Byte of Display Buffer

At a color depth of 8 bpp, each byte of display buffer represents one pixel on the display. At this color depth the read-modify-write cycles of 4 bpp are eliminated making the update of each pixel faster.

Each byte indexes into one of the 256 positions of the LUT. The S1D13706 LUT supports six bits per primary color. This translates into 256K possible colors when color mode is selected. Therefore the displayed mode has 256 colors available out of a possible 256K colors.

When a monochrome panel is selected, the green component of the LUT is used to determine the gray shade intensity. The green indices, with six bits, can resolve 64 gray shades.

3.6 Memory Organization for 16 Bpp (65536 Colors/64 Gray Shades)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Red Component Bits 4-0					Green Component Bits 5-3		
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Green Component Bits 2-0			Blue Component Bits 4-0				

Figure 3-5: Pixel Storage for 16 Bpp in Two Bytes of Display Buffer

At a color depth of 16 bpp the S1D13706 is capable of displaying 64K (65536) colors. The 64K color pixel is divided into three parts: five bits for red, six bits for green, and five bits for blue. In this mode the LUT is bypassed and output goes directly into the Frame Rate Modulator.

Should monochrome mode be chosen at this color depth, the output sends the six bits of the green LUT component to the modulator for a total of 64 possible gray shades. Note that 8 bpp also provides 64 gray shades using less memory.

4 Look-Up Table (LUT)

This section discusses programming the S1D13706 Look-Up Table (LUT). Included is a summary of the LUT registers, recommendations for color/gray shade LUT values, and additional programming considerations. For a discussion of the LUT architecture, refer to the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

The S1D13706 is designed with a LUT consisting of 256 indexed red/green/blue entries. Each LUT entry is six bits wide. The color depth (bpp) determines how many indices are used to output the image to the display. For example, 1 bpp uses the first 2 indices, 2 bpp uses the first 4 indices, 4 bpp uses the first 16 indices and 8 bpp uses all 256 indices. Note that 16 bpp color depths bypass the LUT entirely.

In color modes, the pixel values stored in the display buffer index directly to an RGB value stored in the LUT. In monochrome modes, the pixel value indexes into the green component of the LUT and the amount of green at that index controls the intensity. Monochrome mode look-ups are done based on the Color/Mono Panel Select bit (REG[10h] bit 6).

4.1 Registers

4.1.1 Look-Up Table Write Registers

REG[08h] Look-Up Table Blue Write Data Register							
LUT Blue Write Data Bit 5	LUT Blue Write Data Bit 4	LUT Blue Write Data Bit 3	LUT Blue Write Data Bit 2	LUT Blue Write Data Bit 1	LUT Blue Write Data Bit 0	n/a	n/a

REG[09h] Look-Up Table Green Write Data Register							
LUT Green Write Data Bit 5	LUT Green Write Data Bit 4	LUT Green Write Data Bit 3	LUT Green Write Data Bit 2	LUT Green Write Data Bit 1	LUT Green Write Data Bit 0	n/a	n/a

REG[0Ah] Look-Up Table Red Write Data Register							
LUT Red Write Data Bit 5	LUT Red Write Data Bit 4	LUT Red Write Data Bit 3	LUT Red Write Data Bit 2	LUT Red Write Data Bit 1	LUT Red Write Data Bit 0	n/a	n/a

These registers contain the data to be written to the blue/green/red components of the Look-Up Table. The data is stored in these registers until a write to the LUT Write Address Register (REG[0Bh]) moves the data to the Look-Up Table.

Note

The LUT entries are updated only when the LUT Write Address Register (REG[0Bh]) is written to.

REG[0Bh] Look-Up Table Write Address Register

LUT Write Address Bit 7	LUT Write Address Bit 6	LUT Write Address Bit 5	LUT Write Address Bit 4	LUT Write Address Bit 3	LUT Write Address Bit 2	LUT Write Address Bit 1	LUT Write Address Bit 0
----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------

This register forms a pointer into the Look-UpTable (LUT) which is used to write LUT data stored in REG[08h], REG[09h], and REG[0Ah]. The data is updated to the LUT only with the completion of a write to this register. This is a write-only register and returns 00h if read.

Note

For further information on the S1D13706 LUT architecture, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

4.1.2 Look-Up Table Read Registers**REG[0Ch] Look-Up Table Blue Read Data Register**

LUT Blue Read Data Bit 5	LUT Blue Read Data Bit 4	LUT Blue Read Data Bit 3	LUT Blue Read Data Bit 2	LUT Blue Read Data Bit 1	LUT Blue Read Data Bit 0	n/a	n/a
--------------------------------	--------------------------------	--------------------------------	--------------------------------	--------------------------------	--------------------------------	-----	-----

REG[0Dh] Look-Up Table Green Read Data Register

LUT Green Read Data Bit 5	LUT Green Read Data Bit 4	LUT Green Read Data Bit 3	LUT Green Read Data Bit 2	LUT Green Read Data Bit 1	LUT Green Read Data Bit 0	n/a	n/a
---------------------------------	---------------------------------	---------------------------------	---------------------------------	---------------------------------	---------------------------------	-----	-----

REG[0Eh] Look-Up Table Red Read Data Register

LUT Red Read Data Bit 5	LUT Red Read Data Bit 4	LUT Red Read Data Bit 3	LUT Red Read Data Bit 2	LUT Red Read Data Bit 1	LUT Red Read Data Bit 0	n/a	n/a
-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-----	-----

These registers contains the data returned from the blue/green/red components of the Look-Up Table. The data is read and placed in these registers only when a write to the LUT Write Address Register (REG[0Fh]) copies the data from the Look-Up Table.

REG[0Fh] Look-Up Table Read Address Register

LUT Read Address Bit 7	LUT Read Address Bit 6	LUT Read Address Bit 5	LUT Read Address Bit 4	LUT Read Address Bit 3	LUT Read Address Bit 2	LUT Read Address Bit 1	LUT Read Address Bit 0
---------------------------	---------------------------	---------------------------	---------------------------	---------------------------	---------------------------	---------------------------	---------------------------

This register forms a pointer into the Look-UpTable (LUT) which is used to read LUT data to REG[0Ch], REG[0Dh], and REG[0Eh]. The data is placed in REG[0Ch], REG[0Dh], and REG[0Eh] only with the completion of a write to this register. This is a write-only register and returns 00h if read.

Note

For further information on the S1D13706 LUT architecture, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

4.2 Look-Up Table Organization

- The Look-Up Table treats the value of a pixel as an index into an array of colors or gray shades. For example, a pixel value of zero would point to the first LUT entry, whereas a pixel value of seven would point to the eighth LUT entry.
- The value contained in each LUT entry represents the intensity of the given color or gray shade. This intensity can range in value between 0 and 0Fh.
- The S1D13706 Look-Up Table is linear. This means increasing the LUT entry number results in a lighter color or gray shade. For example, a LUT entry of 0Fh in the red bank results in bright red output while a LUT entry of 05h results in dull red.

Table 4-1: Look-Up Table Configurations

Color Depth	Look-Up Table Indices Used			Effective Gray Shades/Colors
	RED	GREEN	BLUE	
1 bpp gray		2		2 gray shades
2 bpp gray		4		4 gray shades
4 bpp gray		16		16 gray shades
8 bpp gray		16		64 gray shades
16 bpp gray				64 gray shades
1 bpp color	2	2	2	2 colors
2 bpp color	4	4	4	4 colors
4 bpp color	16	16	16	16 colors
8 bpp color	256	256	256	256 colors
16 bpp color				65536 colors

= Indicates the Look-Up Table is not used for that display mode

4.2.1 Gray Shade Modes

Gray shade (monochrome) modes are defined by the Color/Mono Panel Select bit (REG[10h] bit 6). When this bit is set to 0, the value output to the panel is derived solely from the green component of the LUT.

1 bpp gray shade

The 1 bpp gray shade mode uses the green component of the first 2 LUT entries. The remaining indices of the LUT are unused.

Table 4-2: Suggested LUT Values for 1 Bpp Gray Shade

Index	Red	Green	Blue
00	00	00	00
01	00	FC	00
02	00	00	00
...	00	00	00
FF	00	00	00

	Unused entries
--	----------------

2 bpp gray shade

The 2 bpp gray shade mode uses the green component of the first 4 LUT entries. The remaining indices of the LUT are unused.

Table 4-3: Suggested LUT Values for 4 Bpp Gray Shade

Index	Red	Green	Blue
00	00	00	00
01	00	54	00
02	00	A8	00
03	00	FC	00
04	00	00	00
...	00	00	00
FF	00	00	00

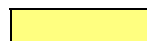
	Unused entries
--	----------------

4 bpp gray shade

The 4 bpp gray shade mode uses the green component of the first 16 LUT entries. The remaining indices of the LUT are unused.

Table 4-4: Suggested LUT Values for 4 Bpp Gray Shade

Index	Red	Green	Blue
00	00	00	00
01	00	10	00
02	00	20	00
03	00	30	00
04	00	44	00
05	00	54	00
06	00	64	00
07	00	74	00
08	00	88	00
09	00	98	00
0A	00	A8	00
0B	00	B8	00
0C	00	CC	00
0D	00	DC	00
0E	00	EC	00
0F	00	FC	00
10	00	00	00
...	00	00	00
FF	00	00	00



Unused entries

8 bpp gray shade

When configured for 8 bpp gray shade mode, the green component of all 256 LUT entries may be used. However, the green component alone only provides 64 intensities (6 bits).

16 bpp gray shade

The Look-Up Table is bypassed at this color depth, therefore programming the LUT is not required.

As with 8 bpp there are limitations to the colors which can be displayed. In this mode the six bits of green are used to set the absolute intensity of the image. This results in 64 gray shades.

4.2.2 Color Modes

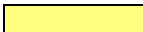
In color display modes, the number of LUT entries used is automatically selected depending on the color depth.

1 bpp color

When the S1D13706 is configured for 1 bpp color mode the first 2 entries in the LUT are used. Each byte in the display buffer contains eight adjacent pixels.

Table 4-5: Suggested LUT Values for 1 bpp Color

Index	Red	Green	Blue
00	00	00	00
01	FC	FC	FC
02	00	00	00
...	00	00	00
FF	00	00	00

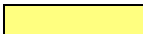
 = Indicates unused entries in the LUT

2 bpp color

When the S1D13706 is configured for 2 bpp color mode the first 4 entries in the LUT are used. Each byte in the display buffer contains four adjacent pixels.

Table 4-6: Suggested LUT Values for 2 bpp Color

Index	Red	Green	Blue
00	00	00	00
01	00	00	FF
02	FF	00	00
03	FC	FC	FC
04	00	00	00
...	00	00	00
FF	00	00	00

 = Indicates unused entries in the LUT

4 bpp color

When the S1D13706 is configured for 4 bpp color mode the first 16 entries in the LUT are used. Each byte in the display buffer contains two adjacent pixels. The upper and lower nibbles of the byte are used as indices into the LUT.

The following table shows LUT values that simulate those of a VGA operating in 16 color mode.

Table 4-7: Suggested LUT Values to Simulate VGA Default 16 Color Palette

Index	Red	Green	Blue
00	00	00	00
01	80	00	00
02	00	80	00
03	80	80	00
04	00	00	80
05	80	00	80
06	00	80	80
07	C0	C0	C0
08	80	80	80
09	FC	00	00
0A	00	FC	00
0B	FC	FC	00
0C	00	00	FC
0D	FC	00	FC
0E	00	FC	FC
0F	FC	FC	FC
10	00	00	00
...	00	00	00
FF	00	00	00

= Indicates unused entries in the LUT

8 bpp color

When the S1D13706 is configured for 8 bpp color mode all 256 entries in the LUT are used. Each byte in the display buffer corresponds to one pixel and is used as an index value into the LUT.

The S1D13706 LUT has six bits (64 intensities) of intensity control per primary color which is the same as a standard VGA RAMDAC.

The following table shows LUT values that simulate the VGA default color palette.

Table 4-8: Suggested LUT Values to Simulate VGA Default 256 Color Palette

Index	R	G	B	Index	R	G	B	Index	R	G	B	Index	R	G	B
00	00	00	00	40	F0	70	70	80	30	30	70	C0	00	40	00
01	00	00	A0	41	F0	90	70	81	40	30	70	C1	00	40	10
02	00	A0	00	42	F0	B0	70	82	50	30	70	C2	00	40	20
03	00	A0	A0	43	F0	D0	70	83	60	30	70	C3	00	40	30
04	A0	00	00	44	F0	F0	70	84	70	30	70	C4	00	40	40
05	A0	00	A0	45	D0	F0	70	85	70	30	60	C5	00	30	40
06	A0	50	00	46	B0	F0	70	86	70	30	50	C6	00	20	40
07	A0	A0	A0	47	90	F0	70	87	70	30	40	C7	00	10	40
08	50	50	50	48	70	F0	70	88	70	30	30	C8	20	20	40
09	50	50	F0	49	70	F0	90	89	70	40	30	C9	20	20	40
0A	50	F0	50	4A	70	F0	B0	8A	70	50	30	CA	30	20	40
0B	50	F0	F0	4B	70	F0	D0	8B	70	60	30	CB	30	20	40
0C	F0	50	50	4C	70	F0	F0	8C	70	70	30	CC	40	20	40
0D	F0	50	F0	4D	70	D0	F0	8D	60	70	30	CD	40	20	30
0E	F0	F0	50	4E	70	B0	F0	8E	50	70	30	CE	40	20	30
0F	F0	F0	F0	4F	70	90	F0	8F	40	70	30	CF	40	20	20
10	00	00	00	50	B0	B0	F0	90	30	70	30	D0	40	20	20
11	10	10	10	51	C0	B0	F0	91	30	70	40	D1	40	20	20
12	20	20	20	52	D0	B0	F0	92	30	70	50	D2	40	30	20
13	20	20	20	53	E0	B0	F0	93	30	70	60	D3	40	30	20
14	30	30	30	54	F0	B0	F0	94	30	70	70	D4	40	40	20
15	40	40	40	55	F0	B0	E0	95	30	60	70	D5	30	40	20
16	50	50	50	56	F0	B0	D0	96	30	50	70	D6	30	40	20
17	60	60	60	57	F0	B0	C0	97	30	40	70	D7	20	40	20
18	70	70	70	58	F0	B0	B0	98	50	50	70	D8	20	40	20
19	80	80	80	59	F0	C0	B0	99	50	50	70	D9	20	40	20
1A	90	90	90	5A	F0	D0	B0	9A	60	50	70	DA	20	40	30
1B	A0	A0	A0	5B	F0	E0	B0	9B	60	50	70	DB	20	40	30
1C	B0	B0	B0	5C	F0	F0	B0	9C	70	50	70	DC	20	40	40
1D	C0	C0	C0	5D	E0	F0	B0	9D	70	50	60	DD	20	30	40
1E	E0	E0	E0	5E	D0	F0	B0	9E	70	50	60	DE	20	30	40
1F	F0	F0	F0	5F	C0	F0	B0	9F	70	50	50	DF	20	20	40
20	00	00	F0	60	B0	F0	B0	A0	70	50	50	E0	20	20	40
21	40	00	F0	61	B0	F0	C0	A1	70	50	50	E1	30	20	40
22	70	00	F0	62	B0	F0	D0	A2	70	60	50	E2	30	20	40

Table 4-8: Suggested LUT Values to Simulate VGA Default 256 Color Palette (Continued)

Index	R	G	B	Index	R	G	B	Index	R	G	B	Index	R	G	B
23	B0	00	F0	63	B0	F0	E0	A3	70	60	50	E3	30	20	40
24	F0	00	F0	64	B0	F0	F0	A4	70	70	50	E4	40	20	40
25	F0	00	B0	65	B0	E0	F0	A5	60	70	50	E5	40	20	30
26	F0	00	70	66	B0	D0	F0	A6	60	70	50	E6	40	20	30
27	F0	00	40	67	B0	C0	F0	A7	50	70	50	E7	40	20	30
28	F0	00	00	68	00	00	70	A8	50	70	50	E8	40	20	20
29	F0	40	00	69	10	00	70	A9	50	70	50	E9	40	30	20
2A	F0	70	00	6A	30	00	70	AA	50	70	60	EA	40	30	20
2B	F0	B0	00	6B	50	00	70	AB	50	70	60	EB	40	30	20
2C	F0	F0	00	6C	70	00	70	AC	50	70	70	EC	40	40	20
2D	B0	F0	00	6D	70	00	50	AD	50	60	70	ED	30	40	20
2E	70	F0	00	6E	70	00	30	AE	50	60	70	EE	30	40	20
2F	40	F0	00	6F	70	00	10	AF	50	50	70	EF	30	40	20
30	00	F0	00	70	70	00	00	B0	00	00	40	F0	20	40	20
31	00	F0	40	71	70	10	00	B1	10	00	40	F1	20	40	30
32	00	F0	70	72	70	30	00	B2	20	00	40	F2	20	40	30
33	00	F0	B0	73	70	50	00	B3	30	00	40	F3	20	40	30
34	00	F0	F0	74	70	70	00	B4	40	00	40	F4	20	40	40
35	00	B0	F0	75	50	70	00	B5	40	00	30	F5	20	30	40
36	00	70	F0	76	30	70	00	B6	40	00	20	F6	20	30	40
37	00	40	F0	77	10	70	00	B7	40	00	10	F7	20	30	40
38	70	70	F0	78	00	70	00	B8	40	00	00	F8	00	00	00
39	90	70	F0	79	00	70	10	B9	40	10	00	F9	00	00	00
3A	B0	70	F0	7A	00	70	30	BA	40	20	00	FA	00	00	00
3B	D0	70	F0	7B	00	70	50	BB	40	30	00	FB	00	00	00
3C	F0	70	F0	7C	00	70	70	BC	40	40	00	FC	00	00	00
3D	F0	70	D0	7D	00	50	70	BD	30	40	00	FD	00	00	00
3E	F0	70	B0	7E	00	30	70	BE	20	40	00	FE	00	00	00
3F	F0	70	90	7F	00	10	70	BF	10	40	00	FF	00	00	00

16 bpp color

The Look-Up Table is bypassed at this color depth, therefore programming the LUT is not required.

5 Power Save Mode

The S1D13706 is designed for very low-power applications. During normal operation, the internal clocks are dynamically disabled when not required. The S1D13706 design also includes a Power Save Mode to further save power. When Power Save Mode is initiated, LCD power sequencing is required to ensure the LCD bias power supply is disabled properly. For further information on LCD power sequencing, see Section 6, “LCD Power Sequencing” on page 29.

For Power Save Mode AC Timing, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

5.1 Overview

The S1D13706 includes a software initiated Power Save Mode. Enabling/disabling Power Save Mode is controlled using the Power Save Mode Enable bit (REG[A0h] bit 0).

While Power Save Mode is enabled the following conditions apply.

- LCD display is inactive.
- LCD interface outputs are forced low.
- Memory is in-accessible.
- Registers are accessible.
- Look-Up Table registers are accessible.

5.2 Registers

5.2.1 Power Save Mode Enable

REG[A0h] Power Save Configuration Register							Read/Write
VNDP Status (RO)	n/a	n/a	n/a	Memory Controller Power Save Status (RO)	n/a	n/a	Power Save Mode Enable

The Power Save Mode Enable bit initiates Power Save Mode when set to 1. Setting the bit back to 0 returns the S1D13706 back to normal mode.

Note

Enabling/disabling Power Save Mode requires proper LCD Power Sequencing. See Section 6, “LCD Power Sequencing” on page 29.

5.2.2 Memory Controller Power Save Status

REG[A0h] Power Save Configuration Register							Read/Write
VNDP Status (RO)	n/a	n/a	n/a	Memory Controller Power Save Status (RO)	n/a	n/a	Power Save Mode Enable

The Memory Controller Power Save Status bit is a read-only status bit which indicates the power save state of the S1D13706 SRAM interface. When this bit returns a 1, the SRAM interface is powered down. When this bit returns a 0, the SRAM interface is active. This bit returns a 0 after a chip reset.

Note

The memory clock source may be disabled when this bit returns a 1.

5.3 Enabling Power Save Mode

Power Save Mode must be enabled using the following steps.

1. Disable the LCD bias power using GPO.

Note

The S5U13706B00C uses GPO to control the LCD bias power supplies. Your system design may vary.

2. Wait for the LCD bias power supply to discharge. The discharge time must be based on the time specified in the LCD panel specification.
3. Enable Power Save Mode - set REG[A0h] bit 0 to 1.
4. At this time, the LCD pixel clock source may be disabled (Optional).
5. Optionally, when the Memory Controller Power Save Status bit (REG[A0h] bit 3) returns a 1, the Memory Clock source may be safely shut down.

5.4 Disabling Power Save Mode

Power Save Mode must be disabled using the following steps.

1. If the Memory Clock source is shut down, it must be started and the Memory Controller Power Save Status bit must return a 0. **Note if the pixel clock source is disabled, it must be started before step 2.**
2. Disable Power Save Mode - set REG[A0h] bit 0 to 0.
3. Wait for the LCD bias power supply to charge. The charge time must be based on the time specified in the LCD panel specification.
4. Enable the LCD bias power using GPO.

Note

The S5U13706B00C uses GPO to control the LCD bias power supplies. Your system design may vary.

6 LCD Power Sequencing

The S1D13706 requires LCD power sequencing (the process of powering-on and powering-off the LCD panel). LCD power sequencing allows the LCD bias voltage to discharge prior to shutting down the LCD signals, preventing long term damage to the panel and avoiding unsightly “lines” at power-on/power-off.

Proper LCD power sequencing for power-off requires a delay from the time the LCD power is disabled to the time the LCD signals are shut down. Power-on requires the LCD signals to be active prior to applying power to the LCD. This time interval depends on the LCD bias power supply design. For example, the LCD bias power supply on the S5U13706 Evaluation board requires 0.5 seconds to fully discharge. Other power supply designs may vary.

This section assumes the LCD bias power is controlled through GPO. The S1D13706 GPIO pins are multi-use pins and may not be available in all system designs. For further information on the availability of GPIO pins, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

Note

This section discusses LCD power sequencing for passive and TFT (non-HR-TFT/D-TFD) panels only. For further information on LCD power sequencing the HR-TFT, see *Connecting to the Sharp HR-TFT Panels*, document number X31B-G-011-xx. For further information on LCD power sequencing the D-TFD, see *Connecting to the Epson D-TFD Panels*, document number X31B-G-012-xx.

6.1 Enabling the LCD Panel

The HAL function `seDisplayEnable(TRUE)` can be used to enable the LCD panel. The function enables the LCD panel using the following steps.

1. Enable the LCD signals - Set Display Blank bit (REG[70h] bit 7) to 0.
2. Wait the required delay time as specified in the LCD panel specification (must be set using 13706CFG). For further information on 13706CFG, see the *13706CFG User Manual*, document number X31B-B-001-xx.
3. Enable GPO to activate the LCD bias power.

Note

`seLcdDisplayEnable` is included in the C source file **hal_misc.c** available on the internet at www.eea.epson.com.

6.2 Disabling the LCD Panel

The HAL function `seDisplayEnable(FALSE)` can be used to disable the LCD panel. The function disables the LCD panel using the following steps.

1. Disable the LCD power using GPO.
2. Wait for the LCD bias power supply to discharge (based on the delay time as specified in the LCD panel specification).
3. Disable the LCD signals - Set Display Blank bit (REG[70h] bit 7) to 1.
4. At this time, the LCD pixel clock source may be disabled (Optional). Note the LUT must not be accessed if the pixel clock is not active.

Note

`seLcdDisplayEnable` is included in the C source file **hal_misc.c** available on the internet at www.eea.epson.com.

7 SwivelView™

Most computer displays operate in landscape mode. In landscape mode the display is wider than it is high. For example, a standard display size of 320x240 is 320 pixels wide and 240 pixels high.

SwivelView rotates the display image counter-clockwise in ninety degree increments, possibly resulting in a display that is higher than it is wide. Rotating the image on a 320x240 display by 90 or 270 degrees yields a display that is now 240 pixels wide and 320 pixels high.

SwivelView also works with panels that are designed with a “portrait” orientation. In this case, when SwivelView 0° is selected, the panel will be in a “portrait” orientation. A selection of SwivelView 90° or SwivelView 270° rotates to a landscape orientation.

The S1D13706 provides hardware support for SwivelView in all color depths (1, 2, 4, 8 and 16 bpp).

For further details on the SwivelView feature, see the *S1D13706 Hardware Functional Specification*, document number X31B-A-001-xx.

7.1 Registers

These are the registers which control the SwivelView feature.

REG[71h] Special Effects Register							
Display Data Word Swap	Display Data Byte Swap	n/a	Sub-Window Enable	n/a	n/a	SwivelView Mode Select Bit 1	SwivelView Mode Select Bit 0

The SwivelView modes are selected using the SwivelView Mode Select Bits [1:0]. The combinations of these bits provide the following rotations.

Table 7-1: SwivelView Enable Bits

SwivelView Enable Bit 1	SwivelView Enable Bit 0	SwivelView Orientation
0	0	0° (normal)
0	1	90°
1	0	180°
1	1	270°

REG[74h] Main Window Display Start Address Register0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[75h] Main Window Display Start Address Register 1							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

REG[76h] Main Window Display Start Address Register 2							
n/a	n/a	n/a	n/a	n/a	n/a	n/a	Bit 16

These registers represent a dword address which points to the start of the main window image in the display buffer. An address of 0 is the start of the display buffer. For the following SwivelView mode descriptions, the *desired byte address* is the starting display address for the main window image, and *panel width* and *panel height* refer to the physical panel dimensions.

Note

Truncate all fractional values before writing to the address registers.

In SwivelView 0°, program the start address

$$= \text{desired byte address} \div 4.$$

In SwivelView 90°, program the start address

$$= ((\text{desired byte address} + (\text{panel height} \times \text{bpp} \div 8)) \div 4) - 1.$$

In SwivelView 180°, program the start address

$$= ((\text{desired byte address} + (\text{panel width} \times \text{panel height} \times \text{bpp} \div 8)) \div 4) - 1.$$

In SwivelView 270°, program the start address

$$= (\text{desired byte address} + ((\text{panel width} - 1) \times \text{panel height} \times \text{bpp} \div 8)) \div 4.$$

Note

SwivelView 0° and 180° require the panel width to be a multiple of $32 \div \text{bits-per-pixel}$. SwivelView 90° and 270° require the panel height to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not possible, a virtual display (one larger than the physical panel size) is required which does satisfy the above requirements. To create a virtual display, program the main window line address offset to values which are greater than that required for the given display width.

REG[78h] Main Window Line Address Offset Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[79h] Main Window Line Address Offset Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These registers indicate the number of dwords per line in the main window image (typically the panel width).

number of dwords per line = $\text{image width} \div (32 \div \text{bpp})$

Note

The image width must be a multiple of $32 \div \text{bpp}$. If the panel width is not such a multiple, a slightly larger width is chosen.

Note

Round up to the nearest integer all line address values that have fractional parts.

7.2 Examples

Example 1: In SwivelView 0° (normal) mode, program the main window registers for a 320x240 panel at color depth of 4 bpp.

1. Confirm the main window coordinates are valid.
The horizontal coordinates must be a multiple of $32 \div \text{bpp}$.

$$320 \div (32 \div 4) = 40$$

Main window horizontal coordinate is valid.

- Determine the main window display start address.
The main window is typically placed at the start of display memory which is at display address 0.

$$\begin{aligned}\text{main window display start address register} &= \text{desired byte address} \div 4 \\ &= 0\end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 00h, REG[75h] is set to 00h, and REG[76h] is set to 00h.

- Determine the main window line address offset.

$$\begin{aligned}\text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 320 \div (32 \div 4) \\ &= 40 \\ &= 28\text{h}\end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 28h, and REG[79h] is set to 00h.

Example 2: In SwivelView 90° mode, program the main window registers for a 320x240 panel at a color depth of 4 bpp.

- Confirm the main window coordinates are valid.
The vertical coordinates must be a multiple of $32 \div \text{bpp}$.

$$240 \div (32 \div 4) = 30$$

Main window vertical coordinate is valid.

- Determine the main window display start address.
The main window is typically placed at the start of display memory, which is at display address 0.

$$\begin{aligned}\text{main window display start address register} &= ((\text{desired byte address} + (\text{panel height} \times \text{bpp} \div 8)) \div 4) - 1 \\ &= ((0 + (240 \times 4 \div 8)) \div 4) - 1 \\ &= 29 \\ &= 1\text{Dh}\end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 1Dh, REG[75h] is set to 00h, and REG[76h] is set to 00h.

- Determine the main window line address offset.

$$\begin{aligned}\text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 240 \div (32 \div 4) \\ &= 30 \\ &= 1\text{Eh}\end{aligned}$$

Program the Main Window Line Address Offset register. REG[78h] is set to 1Eh, and REG[79h] is set to 00h.

Example 3: In SwivelView 180° mode, program the main window registers for a 320x240 panel at a color depth of 4 bpp.

1. Confirm the main window coordinates are valid.
The horizontal coordinates must be a multiple of $32 \div \text{bpp}$.

$$320 \div (32 \div 4) = 40$$

Main window horizontal coordinate is valid.

2. Determine the main window display start address.
The main window is typically placed at the start of display memory which is at display address 0.

main window display start address register

$$\begin{aligned} &= ((\text{desired byte address} + (\text{panel width} \times \text{panel height} \times \text{bpp} \div 8)) \div 4) - 1 \\ &= ((0 + (320 \times 240 \times 4 \div 8)) \div 4) - 1 \\ &= 9599 \\ &= 257Fh. \end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 7Fh, REG[75h] is set to 25h, and REG[76h] is set to 00h.

3. Determine the main window line address offset.

$$\begin{aligned} \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 320 \div (32 \div 4) \\ &= 40 \\ &= 28h \end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 28h, and REG[79h] is set to 00h.

Example 4: In SwivelView 270° mode, program the main window registers for a 320x240 panel at a color depth of 4 bpp.

1. Confirm the main window coordinates are valid.
The vertical coordinates must be a multiple of $32 \div \text{bpp}$.

$$240 \div (32 \div 4) = 30$$

Main window coordinates are valid.

2. Determine the main window display start address.
The main window is typically placed at the start of display memory, which is at display address 0.

$$\begin{aligned}
 &\text{main window display start address register} \\
 &= (\text{desired byte address} + ((\text{panel width} - 1) \times \text{panel height} \times \text{bpp} \div 8) \div 4) \\
 &= (0 + ((320 - 1) \times 240 \times 4 \div 8) \div 4) \\
 &= 9570 \\
 &= 2562h
 \end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 62h, REG[75h] is set to 25h, and REG[76h] is set to 00h.

3. Determine the main window line address offset.

$$\begin{aligned}
 \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\
 &= 240 \div (32 \div 4) \\
 &= 30 \\
 &= 1Eh
 \end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 1Eh, and REG[79h] is set to 00h.

7.3 Limitations

7.3.1 SwivelView 0° and 180°

In SwivelView 0° and 180°, the main window line address offset register requires the *panel width* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the main window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image where the width is *main window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the main window image must be drawn right-justified to this virtual width.

7.3.2 SwivelView 90° and 270°

In SwivelView 90° and 270°, the main window line address offset register requires the *panel height* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the main window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image whose width is *main window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the main window image must be drawn right-justified to this virtual width.

8 Picture-In-Picture Plus

8.1 Concept

Picture-in-Picture Plus enables a sub-window within the main display window. The sub-window may be positioned anywhere within the main window and is controlled through the Sub-Window control registers (see Section 8.2, “Registers”). The sub-window retains the same color depth and SwivelView orientation as the main window.

The following diagram shows an example of a sub-window within a main window.

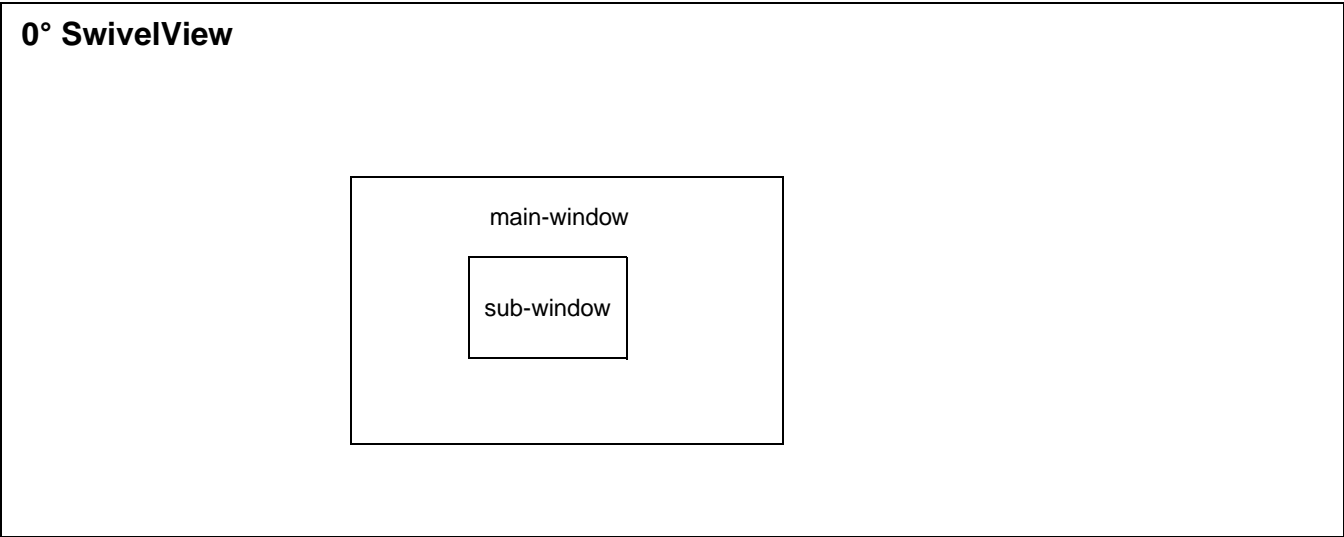


Figure 8-1: Picture-in-Picture Plus with SwivelView disabled

8.2 Registers

These are registers which control the Picture-In-Picture Plus feature.

REG[71h] Special Effects Register							
Display Data Word Swap	Display Data Byte Swap	n/a	Sub-Window Enable	n/a	n/a	SwivelView Mode Select Bit 1	SwivelView Mode Select Bit 0

This bit enables a sub-window within the main window. The location of the sub-window within the landscape window is determined by the Sub-Window X Position registers (REG[84h], REG[85h], REG[8Ch], REG[8Dh]) and Sub-Window Y Position registers (REG[88h], REG[89h], REG[90h], REG[91h]). The sub-window has its own Display Start Address register (REG[7Ch, REG[7Dh], REG[7Eh]) and Memory Address Offset register (REG[80h], REG[81h]). The sub-window shares the same color depth and SwivelView orientation as the main window.

REG[74h] Main Window Display Start Address Register0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[75h] Main Window Display Start Address Register 1							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

REG[76h] Main Window Display Start Address Register 2							
n/a	n/a	n/a	n/a	n/a	n/a	n/a	Bit 16

These registers represent a dword address which points to the start of the main window image in the display buffer. An address of 0 is the start of the display buffer. For the following SwivelView mode descriptions, the *desired byte address* is the starting display address for the main window image, and *panel width* and *panel height* refer to the physical panel dimensions.

Note

Truncate all fractional values before writing to the address registers.

In SwivelView 0°, program the start address

$$= \text{desired byte address} \div 4.$$

In SwivelView 90°, program the start address

$$= ((\text{desired byte address} + (\text{panel height} \times \text{bpp} \div 8)) \div 4) - 1.$$

In SwivelView 180°, program the start address

$$= ((\text{desired byte address} + (\text{panel width} \times \text{panel height} \times \text{bpp} \div 8)) \div 4) - 1.$$

In SwivelView 270°, program the start address

$$= (\text{desired byte address} + ((\text{panel width} - 1) \times \text{panel height} \times \text{bpp} \div 8)) \div 4.$$

Note

SwivelView 0° and 180° require the panel width to be a multiple of $32 \div \text{bits-per-pixel}$. SwivelView 90° and 270° require the panel height to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not possible, a virtual display (one larger than the physical panel size) is required which does satisfy the above requirements. To create a virtual display, program the main window line address offset to values which are greater than that required for the given display width.

REG[78h] Main Window Line Address Offset Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[79h] Main Window Line Address Offset Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These registers indicate the number of dwords per line in the main window image (typically the panel width).

number of dwords per line = image width ÷ (32 ÷ bpp)

Note

The image width must be a multiple of 32 ÷ bpp. If the panel width is not such a multiple, a slightly larger width is chosen.

Note

Round up to the nearest integer all line address values that have fractional parts.

REG[7Ch] Sub-Window Display Start Address Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[7Dh] Sub-Window Display Start Address Register 1							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

REG[7Eh] Sub-Window Display Start Address Register 2							
n/a	n/a	n/a	n/a	n/a	n/a	n/a	Bit 16

These registers represent a dword address which points to the start of the sub-window image in the display buffer. An address of 0 is the start of the display buffer. For the following SwivelView mode descriptions, the *desired byte address* is the starting display address for the sub-window image, and *panel width* and *panel height* refer to the physical panel dimensions. Width and height are used respective to the given SwivelView mode. For example, the sub-window height in SwivelView 90° is the sub-window width in SwivelView 180°.

In SwivelView 0°, program the start address
= desired byte address ÷ 4.

In SwivelView 90°, program the start address
= ((desired byte address + (sub-window width × bpp ÷ 8)) ÷ 4) - 1

In SwivelView 180°, program the start address

$$= ((\text{desired byte address} + (\text{sub-window width} \times \text{sub-window height} \times \text{bpp} \div 8)) \div 4) - 1$$

In SwivelView 270°, program the start address

$$= (\text{desired byte address} + ((\text{sub-window height} - 1) \times \text{sub-window width} \times \text{bpp} \div 8)) \div 4$$

Note

SwivelView 0° and 180° require the panel width to be a multiple of $32 \div \text{bpp}$. SwivelView 90° and 270° require the panel height to be a multiple of $32 \div \text{bpp}$. If this is not possible, a virtual display (one larger than the physical panel size) is required which does satisfy the above requirements. To create a virtual display, program the sub-window line address offset to values which are greater than that required for the given display width.

REG[80h] Sub-Window Line Address Offset Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[81h] Sub-Window Line Address Offset Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These registers indicate the number of dwords per line in the sub-window image.

$$\text{number of dwords per line} = \text{image width} \div (32 \div \text{bpp})$$

Note

The image width must be a multiple of $32 \div \text{bpp}$.

REG[84h] Sub-Window X Start Position Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[85h] Sub-Window X Start Position Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These bits determine the X start position of the sub-window in relation to the origin of the panel. Due to the S1D13706 SwivelView feature, the X start position may not be a horizontal position value (only true in 0° and 180° SwivelView). For further information on defining the value of the X Start Position registers, see Section 8.3, “Picture-In-Picture-Plus Examples” on page 48.

The registers are also incremented differently based on the SwivelView orientation. For 0° and 180° SwivelView the X start position is incremented by X pixels where X is relative to the current color depth.

Table 8-1: 32-bit Address Increments for Color Depth

Bits-per-pixel (Color Depth)	Pixel Increment (X)
1 bpp	32
2 bpp	16
4 bpp	8
8 bpp	4
16 bpp	2

For 90° and 270° SwivelView the X start position is incremented in 1 line increments.

In SwivelView 0°, these registers set the horizontal coordinates (x) of the sub-windows’s top left corner. Increasing values of x move the top left corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-1:).

Program the Sub-Window X Start Position registers so that
 $\text{sub-window X start position registers} = x \div (32 \div \text{bits-per-pixel})$

Note

x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 90°, these registers set the vertical coordinates (y) of the sub-window’s top right corner. Increasing values of y move the top right corner downward in steps of 1 line.

Program the Sub-Window X Start Position registers so that
 $\text{sub-window X start position registers} = y$

In SwivelView 180°, these registers set the horizontal coordinates (x) of the sub-window's bottom right corner. Increasing values of x move the bottom right corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-1:)

Program the Sub-Window X Start Position registers so that
 $\text{sub-window X start position registers} = (\text{panel width} - x) \div (32 \div \text{bits-per-pixel})$

Note

panel width - x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 270°, these registers set the vertical coordinates (y) of the sub-window's bottom left corner. Increasing values of y move the bottom left corner downwards in steps of 1 line.

Program the Sub-Window X Start Position registers so that
 $\text{sub-window X start position registers} = \text{panel width} - y$

REG[88h] Sub-Window Y Start Position Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[89h] Sub-Window Y Start Position Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These bits determine the Y start position of the sub-window in relation to the origin of the panel. Due to the S1D13706 SwivelView feature, the Y start position may not be a vertical position value (only true in 0° and 180° SwivelView). For further information on defining the value of the Y Start Position registers, see Section 8.3, "Picture-In-Picture-Plus Examples" on page 48.

The registers is also incremented differently based on the SwivelView orientation. For 0° and 180° SwivelView the Y start position is incremented in 1 line increments. For 90° and 270° SwivelView the Y start position is incremented by Y pixels where Y is relative to the current color depth.

Table 8-2: 32-bit Address Increments for Color Depth

Bits-Per-Pixel (Color Depth)	Pixel Increment (Y)
1 bpp	32
2 bpp	16
4 bpp	8
8 bpp	4
16 bpp	2

In SwivelView 0°, these registers set the vertical coordinates (y) of the sub-windows's top left corner. Increasing values of y move the top left corner downwards in steps of 1 line.

Program the Sub-Window Y Start Position registers so that
sub-window Y start position registers = y

In SwivelView 90°, these registers set the horizontal coordinates (x) of the sub-window's top right corner. Increasing values of x move the top right corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-2:)

Program the Sub-Window Y Start Position registers so that
sub-window Y start position registers = (panel height - x) \div ($32 \div \text{bits-per-pixel}$)

Note

panel height - x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 180°, these registers set the vertical coordinates (y) of the sub-window's bottom right corner. Increasing values of y move the bottom right corner downwards in steps of 1 line.

Program the Sub-Window Y Start Position registers so that
sub-window Y start position registers = panel height - y

In SwivelView 270°, these registers set the horizontal coordinates (x) of the sub-window's bottom left corner. Increasing values of x move the bottom left corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-2:).

Program the Sub-Window Y Start Position registers so that
sub-window Y start position registers = x \div ($32 \div \text{bits-per-pixel}$)

Note

x must be a multiple of $32 \div \text{bits-per-pixel}$.

REG[8Ch] Sub-Window X End Position Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[8Dh] Sub-Window X End Position Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These bits determine the X end position of the sub-window in relation to the origin of the panel. Due to the S1D13706 SwivelView feature, the X end position may not be a horizontal position value (only true in 0° and 180° SwivelView). For further information on defining the value of the X End Position register, see Section 8.3, “Picture-In-Picture-Plus Examples” on page 48.

The register is also incremented differently based on the SwivelView orientation. For 0° and 180° SwivelView the X end position is incremented by X pixels where X is relative to the current color depth.

Table 8-3: 32-bit Address Increments for Color Depth

Bits-Per-Pixel (Color Depth)	Pixel Increment (X)
1 bpp	32
2 bpp	16
4 bpp	8
8 bpp	4
16 bpp	2

For 90° and 270° SwivelView the X end position is incremented in 1 line increments.

In SwivelView 0°, these registers set the horizontal coordinates (x) of the sub-windows’ bottom right corner. Increasing values of x move the bottom right corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-3:).

Program the Sub-Window X End Position registers so that
sub-window X end position registers = $x \div (32 \div \text{bits-per-pixel}) - 1$

Note

x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 90°, these registers set the vertical coordinates (y) of the sub-window’s bottom left corner. Increasing values of y move the bottom left corner downward in steps of 1 line.

Program the Sub-Window X End Position registers so that
sub-window X end position registers = $y - 1$

In SwivelView 180°, these registers set the horizontal coordinates (x) of the sub-window's top left corner. Increasing values of x move the top left corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-3:)

Program the Sub-Window X End Position registers so that

$$\text{sub-window X end position registers} = (\text{panel width} - x) \div (32 \div \text{bits-per-pixel}) - 1$$

Note

panel width - x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 270°, these registers set the vertical coordinates (y) of the sub-window's top right corner. Increasing values of y move the top right corner downwards in steps of 1 line.

Program the Sub-Window X End Position registers so that

$$\text{sub-window X end position registers} = \text{panel width} - y - 1$$

REG[90h] Sub-Window Y End Position Register 0							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

REG[91h] Sub-Window Y End Position Register 1							
n/a	n/a	n/a	n/a	n/a	n/a	Bit 9	Bit 8

These bits determine the Y end position of the sub-window in relation to the origin of the panel. Due to the S1D13706 SwivelView feature, the Y end position may not be a vertical position value (only true in 0° and 180° SwivelView). For further information on defining the value of the Y End Position register, see Section 8.3, “Picture-In-Picture-Plus Examples” on page 48.

The register is also incremented differently based on the SwivelView orientation. For 0° and 180° SwivelView the Y end position is incremented in 1 line increments. For 90° and 270° SwivelView the Y end position is incremented by *Y* pixels where *Y* is relative to the current color depth.

Table 8-4: 32-bit Address Increments for Color Depth

Bits-Per-Pixel (Color Depth)	Pixel Increment (Y)
1 bpp	32
2 bpp	16
4 bpp	8
8 bpp	4
16 bpp	2

In SwivelView 0°, these registers set the vertical coordinates (y) of the sub-windows’s bottom right corner. Increasing values of y move the bottom right corner downwards in steps of 1 line.

Program the Sub-Window Y End Position registers so that
sub-window Y end position registers = y - 1

In SwivelView 90°, these registers set the horizontal coordinates (x) of the sub-window’s bottom left corner. Increasing values of x move the top right corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-4:)

Program the Sub-Window Y End Position registers so that
sub-window Y end position registers = (panel height - x) \div (32 \div bits-per-pixel) - 1

Note

panel height - x must be a multiple of $32 \div \text{bits-per-pixel}$.

In SwivelView 180°, these registers set the vertical coordinates (y) of the sub-window’s top left corner. Increasing values of y move the top left corner downwards in steps of 1 line.

Program the Sub-Window Y End Position registers so that
sub-window Y end position registers = panel height - y - 1

In SwivelView 270°, these registers set the horizontal coordinates (x) of the sub-window's top right corner. Increasing values of x move the top right corner towards the right in steps of $32 \div \text{bits-per-pixel}$ (see Table 8-4:).

Program the Sub-Window Y End Position registers so that
sub-window Y end position registers = $x \div (32 \div \text{bits-per-pixel}) - 1$

Note

x must be a multiple of $32 \div \text{bits-per-pixel}$.

8.3 Picture-In-Picture-Plus Examples

8.3.1 SwivelView 0° (Landscape Mode)

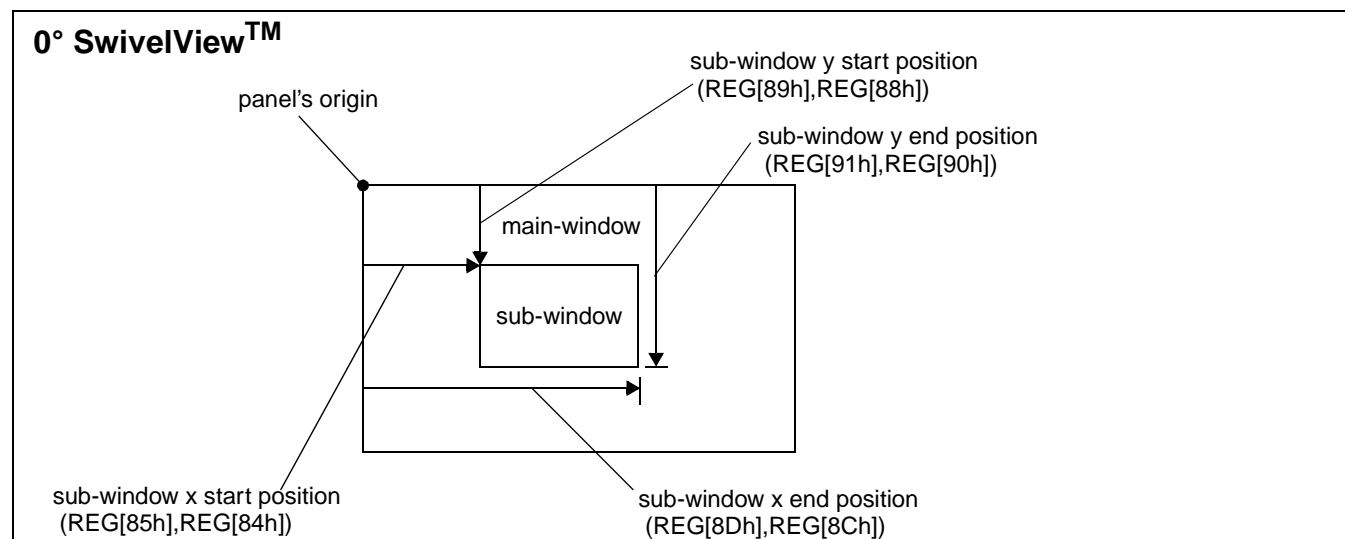


Figure 8-2: Picture-in-Picture Plus with SwivelView disabled

SwivelView 0° (or landscape) is a mode in which both the main and sub-window are non-rotated. The images for each window are typically placed consecutively, with the main window image starting at address 0 and followed by the sub-window image. In addition, both images must start at addresses which are dword-aligned (the last two bits of the starting address must be 0).

Note

It is possible to use the same image for both the main window and sub-window. To do so, set the sub-window line address offset registers to the same value as the main window line address offset registers.

Example 5: Program the main window and sub-window registers for a 320x240 panel at 4 bpp, with the sub-window positioned at (80, 60) with a width of 160 and a height of 120.

1. Confirm the main window coordinates are valid.
The horizontal coordinates must be a multiple of $32 \div \text{bpp}$.

$$320 \div (32 \div 4) = 40$$

Main window horizontal coordinate is valid.

2. Confirm the sub-window coordinates are valid.
The horizontal coordinates and horizontal width must be a multiple of $32 \div \text{bpp}$.

$$80 \div (32 \div 4) = 10$$

$$160 \div (32 \div 4) = 20$$

Sub-window horizontal coordinates and horizontal width are valid.

3. Determine the main window display start address.
The main window is typically placed at the start of display memory which is at display address 0.

$$\begin{aligned}\text{main window display start address register} &= \text{desired byte address} \div 4 \\ &= 0\end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 00h, REG[75h] is set to 00h, and REG[76h] is set to 00h.

4. Determine the main window line address offset.

$$\begin{aligned}\text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 320 \div (32 \div 4) \\ &= 40 \\ &= 28\text{h}\end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 28h, and REG[79h] is set to 00h.

5. Determine the sub-window display start address.
The main window image must take up 320×240 pixels \div 2 pixels per byte = 9600h bytes. If the main window starts at address 0h, the sub-window can start at 9600h.

$$\begin{aligned}\text{sub-window display start address} &= \text{desired byte address} \div 4 \\ &= 9600\text{h} \div 4 \\ &= 2580\text{h}.\end{aligned}$$

Program the Sub-window Display Start Address register. REG[7Ch] is set to 80h, REG[7Dh] is set to 25h, and REG[7Eh] is set to 00h.

6. Determine the sub-window line address offset.

$$\begin{aligned}\text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 160 \div (32 \div 4) \\ &= 20 \\ &= 14\text{h}\end{aligned}$$

Program the Sub-window Line Address Offset register. REG[80h] is set to 14h, and REG[81h] is set to 00h.

7. Determine the value for the sub-window X and Y start and end position registers. Let the top left corner of the sub-window be (x1, y1), and let $x2 = x1 + \text{width}$, $y2 = y1 + \text{height}$.

The X position registers set the horizontal coordinates of the sub-window top left and bottom right corners. Program the X Start Position registers = $x1 \div (32 \div \text{bpp})$. Program the X End Position registers = $x2 \div (32 \div \text{bpp}) - 1$.

The Y position registers, in landscape mode, set the vertical coordinates of the sub-window's top left and bottom right corners. Program the Y Start Position registers = y1. Program the Y End Position registers = $y2 - 1$.

X Start Position registers	= $80 \div (32 \div 4)$
	= 10
	= 0Ah
Y Start Position registers	= 60
	= 3Ch
X End Position registers	= $(80 + 160) \div (32 \div 4) - 1$
	= 29
	= 1Dh
Y End Position registers	= $60 + 120 - 1$
	= 179
	= B3h

Program the Sub-window X Start Position register. REG[84h] is set to 0Ah, and REG[85h] is set to 00h.

Program the Sub-window Y Start Position register. REG[88h] is set to 3Ch, and REG[89h] is set to 00h.

Program the Sub-window X End Position register. REG[8Ch] is set to 1Dh, and REG[8Dh] is set to 00h.

Program the Sub-window Y End Position register. REG[90h] is set to B3h, and REG[91h] is set to 00h.

8. Enable the sub-window.

Program the Sub-window Enable bit. REG[71h] bit 4 is set to 1.

8.3.2 SwivelView 90°

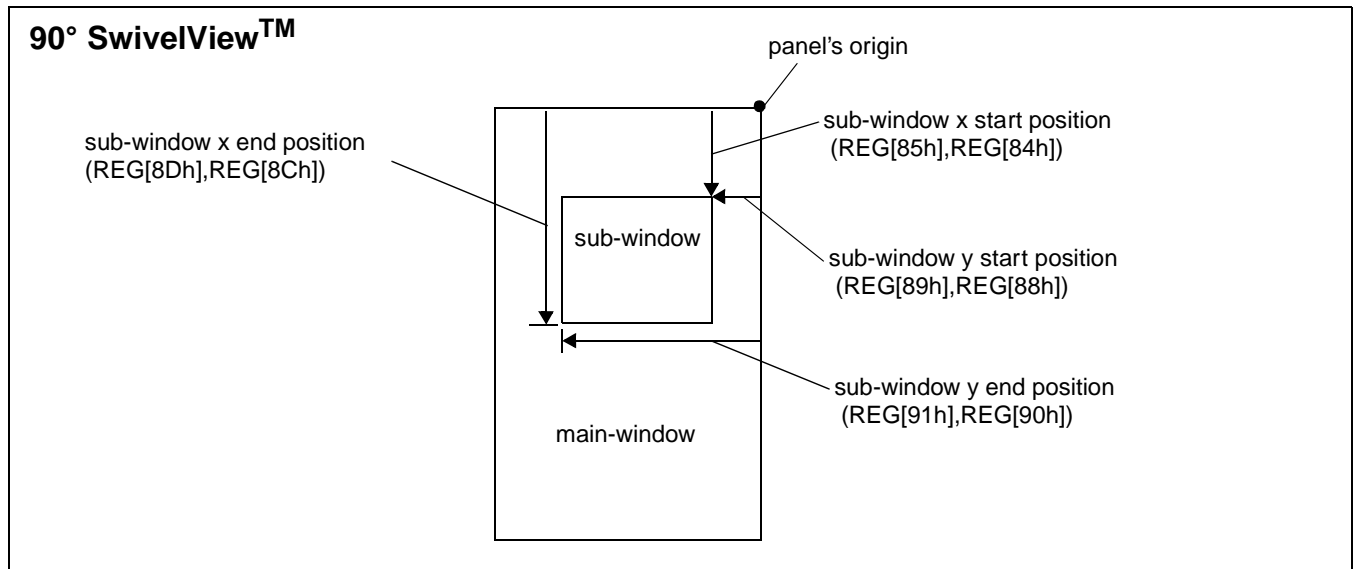


Figure 8-3: Picture-in-Picture Plus with SwivelView 90° enabled

SwivelView 90° is a mode in which both the main and sub-windows are rotated 90° counter-clockwise when shown on the panel. The images for each window are typically placed consecutively, with the main window image starting at address 0 and followed by the sub-window image. In addition, both images must start at addresses which are dword-aligned (the last two bits of the starting address must be 0).

Note

It is possible to use the same image for both the main window and sub-window. To do so, set the sub-window line address offset registers to the same value as the main window line address offset registers.

Note

The Sub-Window X Start Position registers, Sub-Window Y Start Position registers, Sub-Window X End Position registers, and Sub-Window Y End Position registers are named according to the SwivelView 0° orientation. In SwivelView 90°, these registers switch their functionality as described in Section 8.2, “Registers”.

Example 6: In SwivelView 90°, program the main window and sub-window registers for a 320x240 panel at 4 bpp, with the sub-window positioned at SwivelView 90° coordinates (60, 80) with a width of 120 and a height of 160.

1. Confirm the main window coordinates are valid.
The vertical coordinates must be a multiple of $32 \div \text{bpp}$.

$$240 \div (32 \div 4) = 30$$

Main window vertical coordinate is valid.

2. Confirm the sub-window coordinates are valid.
The horizontal coordinates and horizontal width must be a multiple of $32 \div \text{bpp}$.

$$60 \div (32 \div 4) = 7.5 \text{ (invalid)}$$

$$120 \div (32 \div 4) = 15$$

The sub-window horizontal start coordinate is invalid. Therefore, a valid coordinate close to 60 must be chosen. For example, $8 \times (32 \div 4) = 64$. **Consequently the new sub-window coordinates are (64, 80).**

3. Determine the main window display start address.
The main window is typically placed at the start of display memory, which is at display address 0.

$$\begin{aligned} \text{main window display start address register} &= ((\text{desired byte address} + (\text{panel height} \times \text{bpp} \div 8)) \div 4) - 1 \\ &= ((0 + (240 \times 4 \div 8)) \div 4) - 1 \\ &= 29 \\ &= 1\text{Dh} \end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 1Dh, REG[75h] is set to 00h, and REG[76h] is set to 00h.

4. Determine the main window line address offset.

$$\begin{aligned} \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 240 \div (32 \div 4) \\ &= 30 \\ &= 1\text{Eh} \end{aligned}$$

Program the Main Window Line Address Offset register. REG[78h] is set to 1Eh, and REG[79h] is set to 00h.

5. Determine the sub-window display start address.
The main window image must take up $320 \times 240 \text{ pixels} \div 2 \text{ pixels per byte} = 9600\text{h}$ bytes. If the main window starts at address 0h, then the sub-window can start at 9600h.

$$\begin{aligned} \text{sub-window display start address register} &= ((\text{desired byte address} + (\text{sub-window width} \times \text{bpp} \div 8)) \div 4) - 1 \\ &= ((9600\text{h} + (120 \times 4 \div 8)) \div 4) - 1 \\ &= 9614 \\ &= 258\text{Eh} \end{aligned}$$

Program the Sub-window Display Start Address register. REG[7Ch] is set to 8Eh, REG[7Dh] is set to 25h, and REG[7Eh] is set to 00h.

6. Determine the sub-window line address offset.

$$\text{number of dwords per line} = \text{image width} \div (32 \div \text{bpp})$$

$$\begin{aligned}
 &= 120 \div (32 \div 4) \\
 &= 15 \\
 &= 0Fh
 \end{aligned}$$

Program the Sub-window Line Address Offset register. REG[80h] is set to 0Fh, and REG[81h] is set to 00h,

7. Determine the value for the sub-window X and Y start and end position registers. Let the top left corner of the sub-window be (x1, y1), and let x2 = x1 + width, y2 = y1 + height.

The X position registers set the vertical coordinates of the sub-window top right and bottom left corner. Program the X Start Position registers = y1. Program the X End Position registers = y2 - 1.

The Y position registers set the horizontal coordinates of the sub-window top right and bottom left corner. Program the Y Start Position registers = (panel height - x2) ÷ (32 ÷ bpp). Program the Y End Position registers = (panel height - x1) ÷ (32 ÷ bpp) - 1.

X Start Position registers	= 80
	= 50h
Y Start Position registers	= (240 - (64 + 120)) ÷ (32 ÷ 4)
	= 07h
X End Position registers	= (80 + 160) - 1
	= 239
	= EFh
Y End Position registers	= (240 - 64) ÷ (32 ÷ 4) - 1
	= 21
	= 15h

Program the Sub-window X Start Position register. REG[84h] is set to 50h, and REG[85h] is set to 00h.

Program the Sub-window Y Start Position register. REG[88h] is set to 07h, and REG[89h] is set to 00h.

Program the Sub-window X End Position register. REG[8Ch] is set to EFh, and REG[8Dh] is set to 00h.

Program the Sub-window Y End Position register. REG[90h] is set to 15h, and REG[91h] is set to 00h.

8. Enable the sub-window.

Program the Sub-window Enable bit. REG[71h] bit 4 is set to 1.

8.3.3 SwivelView 180°

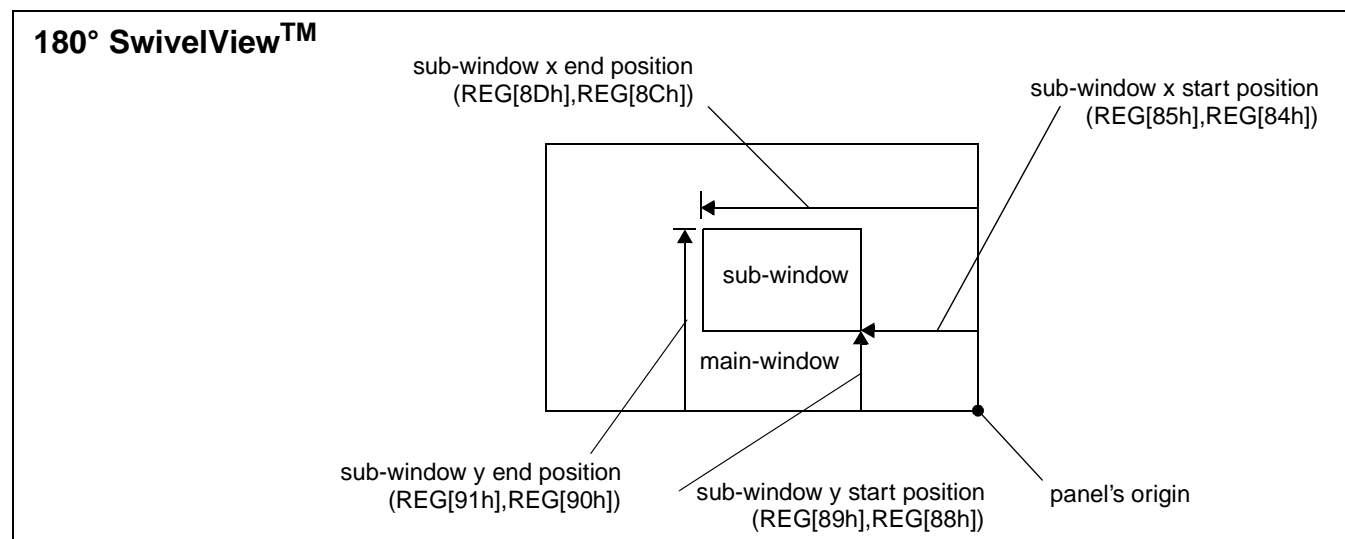


Figure 8-4: Picture-in-Picture Plus with SwivelView 180° enabled

SwivelView 180° is a mode in which both the main and sub-windows are rotated 180° counter-clockwise when shown on the panel. The images for each window are typically placed consecutively, with the main window image starting at address 0 and followed by the sub-window image. In addition, both images must start at addresses which are dword-aligned (the last two bits of the starting address must be 0).

Note

It is possible to use the same image for both the main window and sub-window. To do so, set the sub-window line address offset registers to the same value as the main window line address offset registers.

Note

The Sub-Window X Start Position registers, Sub-Window Y Start Position registers, Sub-Window X End Position registers, and Sub-Window Y End Position registers are named according to the SwivelView 0° orientation. In SwivelView 180°, these registers switch their functionality as described in Section 8.2, “Registers”.

Example 7: In SwivelView 180°, program the main window and sub-window registers for a 320x240 panel at 4 bpp, with the sub-window positioned at SwivelView 180° coordinates (80, 60) with a width of 160 and a height of 120.

1. Confirm the main window coordinates are valid.
The horizontal coordinates must be a multiple of $32 \div \text{bpp}$.

$$320 \div (32 \div 4) = 40$$

Main window horizontal coordinate is valid.

2. Confirm the sub-window coordinates are valid.
The horizontal coordinates and horizontal width must be a multiple of $32 \div \text{bpp}$.

$$\begin{aligned}80 \div (32 \div 4) &= 10 \\160 \div (32 \div 4) &= 20\end{aligned}$$

Sub-window horizontal coordinates and horizontal width are valid.

3. Determine the main window display start address.
The main window is typically placed at the start of display memory which is at display address 0.

$$\begin{aligned}\text{main window display start address register} \\&= ((\text{desired byte address} + (\text{panel width} \times \text{panel height} \times \text{bpp} \div 8)) \div 4) - 1 \\&= ((0 + (320 \times 240 \times 4 \div 8)) \div 4) - 1 \\&= 9599 \\&= 257\text{Fh}.\end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 7Fh, REG[75h] is set to 25h, and REG[76h] is set to 00h.

4. Determine the main window line address offset.

$$\begin{aligned}\text{number of dwords per line} \\&= \text{image width} \div (32 \div \text{bpp}) \\&= 320 \div (32 \div 4) \\&= 40 \\&= 28\text{h}\end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 28h, and REG[79h] is set to 00h.

5. Determine the sub-window display start address.
The main window image must take up $320 \times 240 \text{ pixels} \div 2 \text{ pixels per byte} = 9600\text{h}$ bytes. If the main window starts at address 0h, then the sub-window can start at 9600h.

$$\begin{aligned}\text{sub-window display start address} \\&= ((\text{desired byte address} + (\text{sub-window width} \times \text{sub-window height} \times \text{bpp} \div 8)) \div 4) - 1 \\&= ((9600\text{h} + (160 \times 120 \times 4 \div 8)) \div 4) - 1 \\&= 11999 \\&= 2\text{EDFh}\end{aligned}$$

Program the Sub-window Display Start Address registers. REG[7Ch] is set to DFh, REG[7Dh] is set to 2Eh, and REG[7Eh] is set to 00h.

6. Determine the sub-window line address offset.

$$\begin{aligned}
 \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\
 &= 160 \div (32 \div 4) \\
 &= 20 \\
 &= 14\text{h}
 \end{aligned}$$

Program the Sub-window Line Address Offset registers. REG[80h] is set to 14h, and REG[81h] is set to 00h.

7. Determine the value for the sub-window X and Y start and end position registers.
Let the top left corner of the sub-window be (x1, y1), and let x2 = x1 + width, y2 = y1 + height.

The X position registers set the horizontal coordinates of the sub-window bottom right and top left corner. Program the X Start Position registers = (panel width - x2) ÷ (32 ÷ bpp). Program the X End Position registers = (panel width - x1) ÷ (32 ÷ bpp) - 1.

The Y position registers set the horizontal coordinates of the sub-window bottom right and top left corner. Program the Y Start Position registers = panel height - y2. Program the Y End Position registers = panel height - y1 - 1.

$$\begin{aligned}
 \text{X start position registers} &= (320 - (80 + 160)) \div (32 \div 4) \\
 &= 10 \\
 &= 0\text{Ah} \\
 \text{Y start position registers} &= 240 - (60 + 120) \\
 &= 60 \\
 &= 3\text{Ch} \\
 \text{X end position registers} &= (320 - 80) \div (32 \div 4) - 1 \\
 &= 29 \\
 &= 1\text{Dh} \\
 \text{Y end position registers} &= 240 - 60 - 1 \\
 &= 179 \\
 &= \text{B3h}
 \end{aligned}$$

Program the Sub-window X Start Position registers. REG[84h] is set to 0Ah, and REG[85h] is set to 00h.

Program the Sub-window Y Start Position registers. REG[88h] is set to 3Ch, and REG[89h] is set to 00h.

Program the Sub-window X End Position registers. REG[8Ch] is set to 1Dh, and REG[8Dh] is set to 00h.

Program the Sub-window Y End Position registers. REG[90h] is set to B3h, and REG[91h] is set to 00h.

8. Enable the sub-window.

Program the Sub-window Enable bit. REG[71h] bit 4 is set to 1.

8.3.4 SwivelView 270°

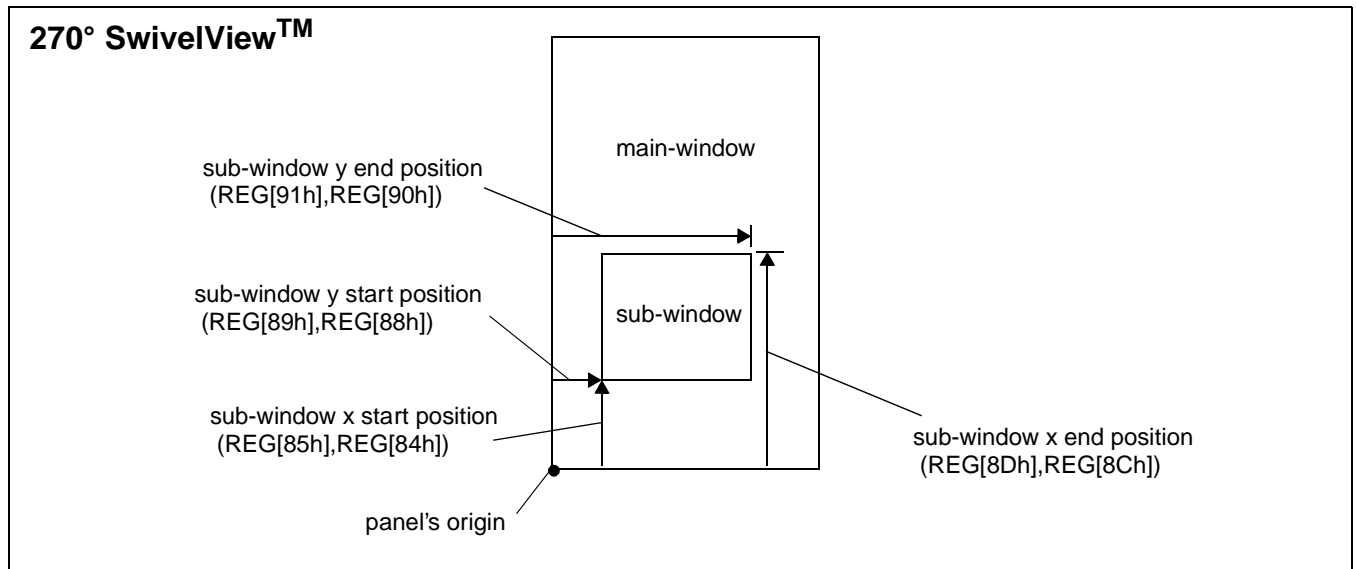


Figure 8-5: Picture-in-Picture Plus with SwivelView 270° enabled

SwivelView 270° is a mode in which both the main and sub-windows are rotated 270° counter-clockwise when shown on the panel. The images for each window are typically placed consecutively, with the main window image starting at address 0 and followed by the sub-window image. In addition, both images must start at addresses which are dword-aligned (the last two bits of the starting address must be 0).

Note

It is possible to use the same image for both the main window and sub-window. To do so, set the sub-window line address offset registers to the same value as the main window line address offset registers.

Note

The Sub-Window X Start Position registers, Sub-Window Y Start Position registers, Sub-Window X End Position registers, and Sub-Window Y End Position registers are named according to the SwivelView 0° orientation. In SwivelView 270°, these registers switch their functionality as described in Section 8.2, “Registers”.

Example 8: In SwivelView 270°, program the main window and sub-window registers for a 320x240 panel at 4 bpp, with the sub-window positioned at SwivelView 270° coordinates (60, 80) with a width of 120 and a height of 160.

1. Confirm the main window coordinates are valid.
The vertical coordinates must be a multiple of $32 \div \text{bpp}$.

$$240 \div (32 \div 4) = 30$$

Main window coordinates are valid.

2. Confirm the sub-window coordinates are valid.
The horizontal coordinates and horizontal width must be a multiple of $32 \div \text{bpp}$.

$$60 \div (32 \div 4) = 7.5 \text{ (invalid)}$$

$$120 \div (32 \div 4) = 15$$

The sub-window horizontal start coordinate is invalid. Therefore, a valid coordinate close to 60 must be chosen. For example, $8 \times (32 \div 4) = 64$. **Consequently the new sub-window coordinates are (64, 80).**

3. Determine the main window display start address.
The main window is typically placed at the start of display memory, which is at display address 0.

$$\begin{aligned} \text{main window display start address register} &= (\text{desired byte address} + ((\text{panel width} - 1) \times \text{panel height} \times \text{bpp} \div 8) \div 4) \\ &= (0 + ((320 - 1) \times 240 \times 4 \div 8) \div 4) \\ &= 9570 \\ &= 2562\text{h} \end{aligned}$$

Program the Main Window Display Start Address registers. REG[74h] is set to 62h, REG[75h] is set to 25h, and REG[76h] is set to 00h.

4. Determine the main window line address offset.

$$\begin{aligned} \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\ &= 240 \div (32 \div 4) \\ &= 30 \\ &= 1\text{Eh} \end{aligned}$$

Program the Main Window Line Address Offset registers. REG[78h] is set to 1Eh, and REG[79h] is set to 00h.

5. Determine the sub-window display start address.
The main window image must take up $320 \times 240 \text{ pixels} \div 2 \text{ pixels per byte} = 9600\text{h}$ bytes. If the main window starts at address 0h, then the sub-window can start at 9600h.

$$\begin{aligned} \text{sub-window display start address register} &= (\text{desired byte address} + ((\text{sub-window height} - 1) \times \text{sub-window width} \times \text{bpp} \div 8) \div 4) \\ &= (9600\text{h} + ((160 - 1) \times 120 \times 4 \div 8) \div 4) \\ &= 11985 \\ &= 2\text{ED}1\text{h} \end{aligned}$$

Program the Sub-window Display Start Address registers. REG[7Ch] is set to D1h, REG[7Dh] is set to 2Eh, and REG[7Eh] is set to 00h.

6. Determine the sub-window line address offset.

$$\begin{aligned}
 \text{number of dwords per line} &= \text{image width} \div (32 \div \text{bpp}) \\
 &= 120 \div (32 \div 4) \\
 &= 15 \\
 &= 0Fh
 \end{aligned}$$

Program the Sub-window Line Address Offset. REG[80h] is set to 0Fh, and REG[81h] is set to 00h.

7. Determine the value for the sub-window X and Y start and end position registers.
Let the top left corner of the sub-window be (x1, y1), and let x2 = x1 + width, y2 = y1 + height.

The X position registers sets the vertical coordinates of the sub-window top right and bottom left corner. Program the X Start Position registers = panel width - y2. Program the X End Position registers = panel width - y1 - 1.

The Y position registers sets the horizontal coordinates of the sub-window top right and bottom left corner. Program the Y Start Position registers = x1 \div (32 \div bpp). Program the Y End Position registers = x2 \div (32 \div bpp) - 1.

$$\begin{aligned}
 \text{X start position registers} &= 320 - (80 + 160) \\
 &= 80 \\
 &= 50h \\
 \text{Y start position registers} &= 64 \div (32 \div 4) \\
 &= 08h \\
 \text{X end position registers} &= 320 - 80 - 1 \\
 &= 239 \\
 &= EFh \\
 \text{Y end position registers} &= (64 + 120) \div (32 \div 4) - 1 \\
 &= 22 \\
 &= 16h
 \end{aligned}$$

Program the Sub-window X Start Position registers. REG[84h] is set to 50h, and REG[85h] is set to 00h.

Program the Sub-window Y Start Position registers. REG[88h] is set to 08h, and REG[89h] is set to 00h.

Program the Sub-window X End Position registers. REG[8Ch] is set to EFh, and REG[8Dh] is set to 00h.

Program the Sub-window Y End Position registers. REG[90h] is set to 16h, and REG[91h] is set to 00h.

8. Enable the sub-window.

Program the Sub-window Enable bit. REG[71h] bit 4 is set to 1.

8.4 Limitations

8.4.1 SwivelView 0° and 180°

In SwivelView 0° and 180°, the main window line address offset register requires the *panel width* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the main window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image where the width is *main window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the main window image must be drawn right-justified to this virtual width.

Similarly, the sub-window line address offset register requires the sub-window image *width* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the sub-window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image whose width is *sub-window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the sub-window image must be drawn right-justified to this virtual width.

8.4.2 SwivelView 90° and 270°

In SwivelView 90° and 270°, the main window line address offset register requires the *panel height* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the main window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image whose width is *main window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the main window image must be drawn right-justified to this virtual width.

Similarly, the sub-window line address offset register requires the sub-window image *width* to be a multiple of $32 \div \text{bits-per-pixel}$. If this is not the case, then the sub-window line address offset register must be programmed to a longer line which is a multiple of $32 \div \text{bits-per-pixel}$. This longer line creates a virtual image whose width is *sub-window line address offset register* $\times 32 \div \text{bits-per-pixel}$ and the sub-window image must be drawn right-justified to this virtual width.

9 Identifying the S1D13706

The S1D13706 can be identified by reading the value contained in the Revision Code Register (REG[00h]). To identify the S1D13706 follow the steps below.

1. Read REG[00h].
2. The production version of the S1D13706 returns a value of 28h (00101000b).
3. The product code is Ah (001010b based on bits 7-2).
4. The revision code is 0h (00b based on bits 1-0).

10 Hardware Abstraction Layer (HAL)

The HAL is a processor independent programming library designed to help port applications and utilities from one SED13xx product to another. Epson has provided this library as a result of developing test utilities for the SED13xx LCD controller products.

The HAL contains functions which are designed to be consistent between SED13xx products, but as the semiconductor products evolve, so must the HAL; consequently there are some differences between HAL functions for different SED13xx products.

Note

As the SED13xx line of products changes, the HAL may change significantly or cease to be a useful tool. Seiko Epson reserves the right to change the functionality of the HAL or discontinue its use if no longer required.

10.1 API for 13706HAL

This section is a description of the HAL library Application Programmers Interface (API). Updates and revisions to the HAL may include new functions not included in the following documentation.

Table 10-1: HAL Functions

Function	Description
Initialization	
seRegisterDevice	Registers the S1D13706 parameters with the HAL. seRegisterDevice MUST be the first HAL function called by an application.
seInitReg	Initializes the registers, LUT, and allocates memory for default surfaces.
seGetHalVersion	Returns HAL library version information.
seHalTerminate	Frees up memory allocated by the HAL before the application exits.
seGetId	Identifies the controller by interpreting the revision code register.
General HAL Support:	
seGetInstalledMemorySize	Returns the total size of the display buffer in bytes.
seGetAvailableMemorySize	Determines the last byte of display buffer available to an application.
seEnableHardwareDisplaySwapping	Enables hardware data swapping for Big-Endian systems.
seGetResolution seGetMainWinResolution seGetSubWinResolution	Returns the width and height of the active display surface.
seSetSubWinCoordinates	Sets the sub-window coordinates.
seGetSubWinCoordinates	Returns the sub-window coordinates.
seGetBytesPerScanline seGetMainWinBytesPerScanline seGetSubWinBytesPerScanline	Returns the number of bytes in each line of the displayed image. Note that the displayed image may be larger than the physical size of the LCD.
seSetPowerSaveMode	Enables/disables power save mode.
seGetPowerSaveMode	Returns the current state of power save mode.
seSetPowerUpDelay	Sets the power-on delay for power save mode.
seSetPowerDownDelay	Sets the power-down delay for power save mode.
seCheckEndian	Returns the Endian mode of the host CPU platform.

Table 10-1: HAL Functions (Continued)

Function	Description
seSetSwivelViewMode	Sets the SwivelView orientation of the LCD.
seGetSwivelViewMode	Returns the SwivelView orientation of the LCD.
seCheckSwivelViewClocks	Verifies the clocks are set correctly for the requested SwivelView orientation.
seDelay	Delays the given number of seconds before returning.
seDisplayBlank seMainWinDisplayBlank seSubWinDisplayBlank	Blank/unblank the display.
seDisplayEnable seMainWinDisplayEnable seSubWinDisplayEnable	Enable/disable the display.
Advanced HAL Functions:	
seBeginHighPriority	Increase thread priority for time critical routines.
seEndHighPriority	Return thread priority to normal.
seSetClock	Set the programmable clock.
Surface Support	
seGetSurfaceDisplayMode	Returns the display surface associated with the active surface.
seGetSurfaceSize	Returns the number of bytes allocated to the active surface.
seGetSurfaceLinearAddress	Returns the linear address of the start of display buffer for the active surface.
seGetSurfaceOffsetAddress	Returns the offset from the start of display buffer to the start of surface memory.
seAllocMainWinSurface seAllocSubWinSurface	Manually allocates display buffer memory for a surface.
seFreeSurface	Frees any allocated surface memory.
seSetMainWinAsActiveSurface seSetSubWinAsActiveSurface	Changes the active surface.
sePwmEnable	Enables the PWMCLK circuitry.
seCvEnable	Enables the CV Pulse circuitry.
sePwmControl	Configures the PWMCLK registers.
seCvControl	Configures the CV Pulse registers.
Register Access	
seReadRegByte	Reads one register using a byte access.
seReadRegWord	Reads two registers using a word access.
seReadRegDword	Reads four registers using a dword access.
seWriteRegByte	Writes one register using a byte access.
seWriteRegWord	Writes two registers using a word access.
seWriteRegDword	Writes four registers using a dword access.
Memory Access	
seReadDisplayByte	Reads one byte from display buffer.
seReadDisplayWord	Reads one word from display buffer.
seReadDisplayDword	Reads one dword from display buffer.
seWriteDisplayBytes	Writes one or more bytes to display buffer.
seWriteDisplayWords	Writes one or more words to display buffer.
seWriteDisplayDwords	Writes one or more dwords to display buffer.
Color Manipulation:	
seWriteLutEntry	Writes one RGB element to the lookup table.

Table 10-1: HAL Functions (Continued)

Function	Description
seReadLutEntry	Reads one RGB element from the lookup table.
seWriteLut	Write the entire lookup table.
seReadLut	Read the entire lookup table.
seSetMode	Sets the color depth of the display and updates the LUT.
seUseMainWinImageForSubWin	Sets the sub-window image to use the same image as the main window.
seGetBitsPerPixel	Gets the current color depth.
Virtual Display	
seVirtInit seMainWinVirtInit seSubWinVirtInit seMainAndSubWinVirtInit	Initialize a surface to hold an image larger than the physical display size. Also required for SwivelView 90° and 270°.
seVirtPanScroll seMainWinVirtPanScroll seSubWinVirtPanScroll seMainAndSubWinVirtPanScroll	Pan (right/left) and Scroll (up/down) the display device over the indicated virtual surface.
Drawing	
seSetPixel seSetMainWinPixel seSetSubWinPixel	Set one pixel at the specified (x,y) co-ordinate and color.
seGetPixel seGetMainWinPixel seGetSubWinPixel	Returns the color of the pixel at the specified (x,y) co-ordinate.
seDrawLine seDrawMainWinLine seDrawSubWinLine	Draws a line between two endpoints in the specified color
seDrawRect seDrawMainWinRect seDrawSubWinRect	Draws a rectangle. The rectangle can be outlined or filled.
seDrawCircle seDrawMainWinCircle seDrawSubWinCircle	Draws a circle of given radius and color at the specified center point.
seDrawEllipse seDrawMainWinEllipse seDrawSubWinEllipse	Draws an ellipse centered on a given point with the specified horizontal and vertical radius.
Register/Display Memory	
seGetLinearDisplayAddress	Returns the linear address of the start of physical display memory.
seGetLinearRegAddress	Returns the linear address of the start of S1D13706 control registers.

10.2 Initialization

Initialization functions are normally the first functions in the HAL library that an application calls. These routines return information about the controller and prepare the HAL library for use.

int seRegisterDevice(const LPHAL_STRUCT lpHalInfo)

Description: This function registers the S1D13706 device parameters with the HAL library. The device parameters include such items as address range, register values, desired frame rate, etc. These parameters are stored in the HAL_STRUCT structure pointed to by lpHalInfo. Additionally this routine allocates system memory as address space for accessing registers and the display buffer.

Parameters: lpHalInfo A pointer to a HAL_STRUCT structure. This structure must be filled with appropriate values prior to calling seRegisterDevice.

Return Value: ERR_OK operation completed with no problems
 ERR_UNKNOWN_DEVICE The HAL was unable to locate the S1D13706.
 ERR_FAILED The HAL was unable to map S1D13706 display memory to the host platform.

In addition, on Win32 platforms, the following two error values may be returned:

ERR_PCI_DRIVER_NOT_FOUND The HAL was unable to locate file SED13XX.VXD
 ERR_PCI_BRIDGE_ADAPTER_NOT_FOUND The driver file SED13XX.VXD was unable to locate the PCI bridge adapter board attached to the evaluation board.

Note

seRegisterDevice() MUST be called before any other HAL functions.

int seInitReg(unsigned Flags)

Description:	This function initializes the S1D13706 registers, the LUT, assigns default surfaces and allocates memory accordingly.	
Parameters:	Flags	Provides additional information about how to perform the initialization. Valid values for Flags are: CLEAR_MEM Zero display memory as part of the initialization. DISP_BLANK Blank the display, for aesthetics, during initialization.
Return Value:	ERR_OK	The initialization completed with no problems.
	ERR_NOT_ENOUGH_MEMORY	Insufficient display buffer.
	ERR_CLKI_NOT_IN_TABLE	Could not program CLKI in clock synthesizer because selected frequency not in table.
	ERR_CLKI2_NOT_IN_TABLE	Could not program CLKI2 in clock synthesizer because selected frequency not in table.

void seGetHalVersion(const char ** pVersion, const char ** pStatus, const char **pRevision)

Description:	Retrieves the HAL library version information. By retrieving and displaying the HAL version information along with application version information, it is possible to determine at a glance whether the latest version of the software is being run.	
Parameters:	pVersion	A pointer to the string containing the HAL version code.
	pStatus	A pointer to the string containing the HAL status code A “B” designates a beta version of the HAL, a NULL indicates the release version
	pRevision	A pointer to the string containing the HAL revision status.
Return Value:	The version information is returned as the contents of the pointer arguments. A typical return might be: *pVersion == “1.01” (HAL version 1.01) *pStatus == “B” (BETA release) *pRevision == “5” (fifth update of the beta)	

int seHalTerminate(void)

Description:	Frees up memory allocated by HAL before application exits.	
Parameters:	none.	
Return Value:	ERR_OK	HAL is now ready for application to exit.
	ERR_PCI_DRIVER_NOT_FOUND	Could not find PCI driver (Intel Windows platform only).
	ERR_PCI_BRIDGE_ADAPTER_NOT_FOUND	Could not find PCI Bridge Adapter board (Intel Windows platform only).
	ERR_FAILED	Could not free memory.

int seGetId(int * pId)

Description:	Reads the S1D13706 revision code register to determine the controller product and revision.	
Parameters:	pId	A pointer to an integer to receive the controller ID. The value returned is the revision code.
Return Value:	ERR_OK	The operation completed with no problems
	ERR_UNKNOWN_DEVICE	The product code was not for the S1D13706.

10.2.1 General HAL Support

This category of HAL functions provide several essential services which do not readily group with other functions.

DWORD seGetInstalledMemorySize(void)

Description: This function returns the size of the display buffer in bytes.

For the S1D13706, seGetInstalledMemorySize() and seGetAvailableMemorySize() return the same value.

Parameters: None

Return Value: The return value is the size of the display buffer in bytes (1 4000h for the S1D13706).

DWORD seGetAvailableMemorySize(void)

Description: This function returns an offset to the last byte of memory accessible to an application.

An application can directly access memory from offset zero to the offset returned by this function. On most systems the return value will be the last byte of physical display memory.

For the S1D13706, seGetInstalledMemorySize() and seGetAvailableMemorySize() return the same value.

Parameters: None.

Return Value: The return value is the size of the available amount of display buffer memory directly accessible to an application.

int seEnableHardwareDisplaySwapping(int Enable)

Description: The S1D13706 requires 16 bits-per-pixel data to be in little-endian format. On big-endian systems, the software or hardware needs to swap this data. seEnableHardwareDisplaySwapping() is intended to be used on big-endian systems, where system performance can be improved by utilizing hardware swapping of display memory bytes in 16 bits-per-pixel.

If the system is not big-endian, or if the bits-per-pixel is not 16, this function will not enable hardware display swapping. However, a flag is set in the HAL, and if seSetMode is later called to set the bits-per-pixel to 16 in a big-endian system, hardware display swapping is enabled. Also, if seSetMode is called to set the bits-per-pixel to a value other than 16, then hardware display swapping is disabled.

Parameters:

Enable	Call with Enable set to TRUE to enable hardware display swapping. Call with Enable set to FALSE to disable hardware display swapping.
--------	--

Return Value:

ERR_OK	Function completed successfully
ERR_FAILED	Returned when caller requested that hardware display swapping be enabled, but system not in 16 bits-per-pixel or system is not big-endian.

int seGetResolution(unsigned *Width, unsigned *Height)
void seGetMainWinResolution(unsigned *Width, unsigned *Height)
void seGetSubWinResolution(unsigned *Width, unsigned *Height)

Description: seGetResolution() returns the width and height of the active surface (main window or sub-window).

seGetMainWinResolution() and seGetSubWinResolution() return the width and height of the respective window.

Virtual dimensions are not accounted for in the return values for width and height. For example, seGetMainWinResolution() always returns the panel dimensions, regardless of the value of the line address offset registers.

The width and height are adjusted for SwivelView orientation.

Parameters:

Width	A pointer to an unsigned integer which will receive the width, in pixels, for the indicated surface.
Height	A pointer to an unsigned integer which will receive the height, in pixels, for the indicated surface.

Return Value: seGetResolution() returns one of the following:

ERR_OK	Function completed successfully
ERR_FAILED	Returned when there is not an active display surface.

seGetMainWinResolution() and seGetSubWinResolution() do not return any value.

void seSetSubWinCoordinates(DWORD x1, DWORD y1, DWORD x2, DWORD y2)

Description: seSetSubWinCoordinates sets the upper left and lower right corners of the sub-window display.

(x1, y1) and (x2, y2) are relative to the upper left corner of the panel as defined by the SwivelView mode.

Parameters:

x1	The sub-window x start position (upper left corner).
y1	The sub-window y start position (upper left corner).
x2	The sub-window x end position (lower right corner).
y2	The sub-window y end position (lower right corner).

Return Value: None.

void seGetSubWinCoordinates(DWORD *x1, DWORD *y1, DWORD *x2, DWORD *y2)

Description: seGetSubWinCoordinates return the upper left and lower right corners of the sub-window display.

The coordinates are adjusted for SwivelView orientation.

Parameters:

x1	A pointer to an unsigned long which will receive the sub-window x start position (upper left corner).
y1	A pointer to an unsigned long which will receive the sub-window y start position (upper left corner).
x2	A pointer to an unsigned long which will receive the sub-window x end position (lower right corner).
y2	A pointer to an unsigned long which will receive the sub-window y end position (lower right corner).

Return Value: None.

unsigned seGetBytesPerScanline(void)**unsigned seGetMainWinBytesPerScanline(void)****unsigned seGetSubWinBytesPerScanline(void)**

Description: These functions return the number of bytes in each line of the displayed image. Note that the displayed image may be larger than the physical size of the LCD.

seGetBytesPerScanline() returns the number of bytes per scanline for the current active surface.

seGetMainWinBytesPerScanline() and seGetSubWinBytesPerScanline() return the number of bytes per scanline for the surface indicated in the function name.

To work correctly these routines require the S1D13706 registers to be initialized prior to being called.

Parameters: None.

Return Value: The return value is the “stride” or number of bytes from the first byte of one scanline to the first byte of the next scanline. This value includes both the displayed and the non-displayed bytes on each logical scanline.

void seSetPowerSaveMode(BOOL Enable)

Description: This function enables or disables the power save mode.

When power save mode is enabled the S1D13706 reduces power consumption by making the displays inactive and ignoring memory accesses. Disabling power save mode re-enables the video system to full functionality.

When powering down, the following steps are implemented:

1. Disable LCD power
2. Delay for LCD power down time interval [see seSetPowerDownDelay()].
3. Enable power save mode

When powering up, the following steps are implemented:

1. Disable power save mode
2. Delay for LCD power up time interval [see `seSetPowerUpDelay()`]
3. Enable LCD power

Note

`seSetPowerSaveMode()` waits on vertical non-display (VNDP) cycles for delays. If there is no VNDP cycle, this function will freeze the system. To ensure VNDP cycles are being generated, ensure that there is a clock available for PCLK. Alternatively, set the power-up and power-down times to 0.

Parameters: Enable Call with Enable set to TRUE to set power save mode.
Call with Enable set to FALSE to disable power save mode.

Return Value: None.

BOOL seGetPowerSaveMode(void)

Description: `seGetPowerSaveMode()` returns the current state of power save mode.

Parameters: None.

Return Value: The return value is TRUE if power save mode is enabled. The return value is FALSE if power save mode is not enabled.

void seSetPowerUpDelay(WORD PowerupTime)

Description: `seSetPowerUpDelay()` sets the power-up delay for `seSetPowerSaveMode()`.

Parameters: PowerupTime Power-up time, in milliseconds.

Return Value: None.

void seSetPowerDownDelay(WORD PowerdownTime)

Description: `seSetPowerDownDelay()` sets the power-down delay for `seSetPowerSaveMode()`.

Parameters: PowerdownTime Power-down time, in milliseconds.

Return Value: None.

void seCheckEndian(BOOL *ReverseBytes)

Description: This function returns the “endian-ness” of the CPU the application is running on.

Parameters: ReverseBytes A pointer to boolean value to receive the endian-ness of the system. On return from this function `ReverseBytes` is FALSE if the CPU is little endian (i.e. Intel). `ReverseBytes` will be TRUE if the CPU is big-endian (i.e. Motorola)

Return Value: None.

int seSetSwivelViewMode(int rotate)

Description: This function sets the SwivelView orientation of the LCD display. Display memory is automatically released and then reallocated as necessary for the display size.

IMPORTANT

When the SwivelView mode is changed, memory allocated for both the main window and sub-window display buffer is freed and the display buffer memory is reassigned. The application must redraw the display and re-initialize the sub-window (if used) and redraw after calling seSetSwivelViewMode().

Parameters: rotate The values for rotate are:
 LANDSCAPE: display not rotated
 ROTATE90: display rotated 90 degrees counterclockwise
 ROTATE180: display rotated 180 degrees counterclockwise
 ROTATE270: display rotated 270 degrees counterclockwise

Return Value: ERR_OK The new rotation was completed with no problems.
 ERR_NOT_ENOUGH_MEMORY Insufficient display buffer.

int seGetSwivelViewMode(void)

Description: This function retrieves the SwivelView orientation of the LCD display.

The SwivelView status is read directly from the S1D13706 registers. Calling this function when the LCD display is not initialized will result in an erroneous return value.

Note

seGetSwivelViewMode() was previously called seGetLcdOrientation(). It is now recommended to call seGetSwivelViewMode() instead of seGetLcdOrientation().

Parameters: None.

Return Value: LANDSCAPE Not rotated.
 ROTATE90 Display is rotated 90 degrees counterclockwise.
 ROTATE180 Display is rotated 180 degrees counterclockwise.
 ROTATE270 Display is rotated 270 degrees counterclockwise.

int seCheckSwivelViewClocks(unsigned BitsPerPixel, unsigned Rotate)

Description: This function verifies that the clocks are properly configured for the a SwivelView mode given the bits-per-pixel and rotation (see the section titled SwivelView in the S1D13706 Hardware Functional Specification document).

Parameters: BitsPerPixel The given color depth. BitsPerPixel can be one of the following:
 1, 2, 4, 8, 16.
 Rotate The values for Rotate are:
 LANDSCAPE: display not rotated
 ROTATE90: display rotated 90 degrees counterclockwise
 ROTATE180: display rotated 180 degrees counterclockwise
 ROTATE270: display rotated 270 degrees counterclockwise

Return Value: ERR_OK The function completed with no problems
 ERR_SWIVELVIEW_CLOCK The clocks are not configured correctly.

int seDelay(DWORD Seconds)

Description: This function, intended for non-Intel platforms, delays for the specified number of seconds then returns to the calling routine. On several evaluation platforms it was not readily apparent where to obtain an accurate source of time delays. seDelay() was the result of the need to delay a specified amount of time on these platforms.

For non-Intel platforms, seDelay works by calculating and counting the number of vertical non-display periods in the requested delay time. This implies two conditions for proper operation:

- a) The S1D13706 control registers must be configured to correct values.
- b) The display interface must be enabled (not in power save mode).

For Intel platforms, seDelay() calls the C library time functions to delay the desired amount of time using the system clock.

Parameters: Seconds The number of seconds to delay for.

Return Value: ERR_OK Returned by all platforms at the completion of a successful delay.
ERR_FAILED Returned by non-Intel platforms in which the power save mode is enabled.

void seDisplayBlank(BOOL Blank)

void seMainWinDisplayBlank(BOOL Blank)

void seSubWinDisplayBlank(BOOL Blank)

Description: These functions blank their respective display. Blanking the display is a fast convenient means of temporarily shutting down a display device.

For instance, updating the entire display in one write may produce a flashing or tearing effect. If the display is blanked prior to performing the update, the operation is perceived to be smoother and cleaner.

seDisplayBlank() will blank the display associated with the current active surface.

seDisplayMainWinBlank() and seDisplaySubWinBlank() blank the display for the surface indicated in the function name.

Parameters: Blank Call with Blank set to TRUE to blank the display. Call with Blank set to FALSE to un-blank the display.

Return Value: None.

void seDisplayEnable(BOOL Enable)
void seMainWinDisplayEnable(BOOL Enable)
void seSubWinDisplayEnable(BOOL Enable)

Description: These functions enable or disable the selected display device.

seDisplayEnable() enables or disables the display for the active surface.

seMainWinDisplayEnable() enables or disables the main window display (for the S1D13706, the display blank feature is used to enable or disable the main window).

seSubWinDisplayEnable() enables or disables the sub-window display.

Parameters: Enable Call with Enable set to TRUE to enable the display device. Call with Enable set to FALSE to disable the device.

Return Value: None.

10.2.2 Advance HAL Functions

The advanced HAL functions include a level of access that most applications will never need to access.

int seBeginHighPriority(void)

Description: Writing and debugging software under the Windows operating system greatly simplifies the development process for the S1D13706 evaluation system. One issue which impedes application programming is that of latency. Time critical operations (i.e. performance measurement) are not guaranteed any set amount of processor time.

This function raises the priority of the thread and virtually eliminates the question of latency for programs running on a Windows platform.

Note

The application should not leave it's thread running in a high priority state for long periods of time. As soon as a time critical operation is complete the application should call seEndHighPriority().

Parameters: None.

Return Value: The priority nest count which is the number of times seBeginHighPriority() has been called without a corresponding call to seEndHighPriority().

int seEndHighPriority(void)

Description: This function decreases the priority nest count. When this count reaches zero, the thread priority of the calling application is set to normal.

After performing some time critical operation the application should call seEndHighPriority() to return the thread priority to a normal level.

Parameters: None.

Return Value: The priority nest count which is the number of times seBeginHighPriority() has been called without a corresponding call to seEndHighPriority().

int seSetClock(CLOCKSELECT ClockSelect, FREQINDEX FreqIndex)

Description: Call seSetClock() to set the clock rate of the programmable clock.

Parameters: ClockSelect The ICD2061A programmable clock chip supports two output clock signals. ClockSelect chooses which of the two output clocks to adjust.

Valid ClockSelect values for CLKI or CLKI2 (defined in HAL.H).

FreqIndex FreqIndex is an enumerated constant and determines what the output frequency should be.

Valid values for FreqIndex are:

FREQ_6000	6.000 MHz
FREQ_10000	10.000 MHz
FREQ_14318	14.318 MHz
FREQ_17734	17.734 MHz
FREQ_20000	20.000 MHz
FREQ_24000	24.000 MHz
FREQ_25000	25.000 MHz
FREQ_25175	25.175 MHz
FREQ_28318	28.318 MHz
FREQ_30000	30.000 MHz
FREQ_31500	31.500 MHz
FREQ_32000	32.000 MHz
FREQ_33000	33.000 MHz
FREQ_33333	33.333 MHz
FREQ_34000	34.000 MHz
FREQ_35000	35.000 MHz
FREQ_36000	36.000 MHz
FREQ_40000	40.000 MHz
FREQ_49500	49.500 MHz
FREQ_50000	50.000 MHz
FREQ_56250	56.250 MHz
FREQ_65000	65.000 MHz
FREQ_80000	80.000 MHz
FREQ_100000	100.000 MHz

Return Value: ERR_OK The function completed with no problems.
ERR_FAILED seSetClock failed because of an invalid ClockSelect or an invalid frequency index.

10.2.3 Surface Support

The S1D13706 HAL library depends heavily on the concept of surfaces. Through surfaces the HAL tracks memory requirements of the main window and sub-window.

Surfaces allow the HAL to permit or fail function calls which change the geometry of the S1D13706 display memory. Most HAL functions either allocate surface memory or manipulate a surface that has been allocated.

The functions in this section allow the application programmer a little greater control over surfaces.

int seGetSurfaceDisplayMode(void)

Description: This function determines the type of display associated with the current active surface.

Parameters: None.

Return Value: The return value indicates the active surface display type. Return values will be one of:

MAIN_WIN	The main window is the active surface.
SUB_WIN	The sub-window is the active surface.

DWORD seGetSurfaceSize(void)

Description: This function returns the number of display memory bytes allocated to the current active surface.

Parameters: None.

Return Value: The return value is the number of bytes allocated to the current active surface.
The return value will be 0 if this function is called before initializing the registers.

DWORD seGetSurfaceLinearAddress(void)

Description: This function returns the linear address of the start of memory for the active surface.

Parameters: None.

Return Value: The return value is the linear address to the start of memory for the active surface. A linear address is a 32-bit offset, in CPU address space.
The return value will be NULL if this function is called before a surface has been initialized.

DWORD seGetSurfaceOffsetAddress(void)

Description: This function returns the offset, from the first byte of display memory to the first byte of memory associated with the active display surface.

Parameters: None.

Return Value: The return value is the offset, in bytes, from the start of display memory to the start of the active surface. An address of 0 indicates the surface starts in the first byte of display buffer memory.

Note

This function also returns 0 if there is no memory allocated to an active surface. You must ensure that memory is allocated before calling seGetSurfaceOffsetAddress().

DWORD seAllocMainWinSurface(DWORD Size)**DWORD seAllocSubWinSurface(DWORD Size)**

Description: These functions allocate display buffer memory for a surface. If the surface previously had memory allocated then that memory is first released. Newly allocated memory is not cleared.

Call seAllocMainWinSurface() or seAllocSubWinSurface() to allocate the requested amount of display memory for the indicated surface.

These functions allow an application to bypass the automatic surface allocation which occurs when functions such as seInitReg() or seSetMode() are called.

Parameters: Size The size in bytes of the requested memory block.

Return Value: If the memory allocation succeeds then the return value is the linear address of the allocated memory. If the allocation fails then the return value is 0. A linear address is a 32-bit offset, in CPU address space.

int seFreeSurface(DWORD LinearAddress)

Description: This function can be called to free any previously allocated display buffer memory.

This function is intended to complement seAllocMainWinSurface() and seAllocSubWinSurface().

After calling one of these functions, the application must switch the active surface to one which has memory allocated before calling any drawing functions.

Parameters: LinearAddress A valid linear address. The linear address is a dword returned to the application by any surface allocation call.

Return Value: ERR_OK Function completed successfully.
ERR_FAILED Function failed.

void seSetMainWinAsActiveSurface(void)

void seSetSubWinAsActiveSurface(void)

Description: These functions set the active surface to the display indicated in the function name.
Before calling one of these surface selection routines, that surface must have been allocated using any of the surface allocation functions.

Parameters: None.

Return Value: None.

void sePwmEnable(int Enable)

Description: This function enables or disables the Pulse Width Modulation (PWM) clock circuitry.

Parameters: Enable Set to TRUE or FALSE to enable or disable PWM.

Return Value: None.

void seCvEnable(int Enable)

Description: This function enables or disables the Contrast Voltage (CV) pulse circuitry.

Parameters: Enable Set to TRUE or FALSE to enable or disable CV.

Return Value: None.

void sePwmControl(CLOCKSELECT ClkSource, int ClkDivide, int DutyCycle)

Description: This function sets up the Pulse Width Modulation (PWM) clock configuration registers.

Parameters:

ClkSource	The clock source for PWM; set to either CLKI or CLKI2.
ClkDivide	The clock source is divided by $2^{\text{ClkDivide}}$. Legal values for ClkDivide are from 0 to 12 (decimal). For example, if ClkDivide is 3, the clock source is divided by $2^3=8$.
DutyCycle	The PWM clock duty cycle; values can be from 0 to 255. A value of 0 makes the PWM output always low, and a value of 255 makes the PWM output high for 255 out of 256 clock periods.

Return Value: None.

void seCvControl(CLOCKSELECT ClkSource, int ClkDivide, int BurstLength)

Description: This function sets up the Contrast Voltage (CV) pulse configuration registers.

Parameters:

ClkSource	The clock source for CV; set to either CLKI or CLKI2.
ClkDivide	The clock source is divided by $2^{\text{ClkDivide}}$. Legal values for ClkDivide are from 0 to 12 (decimal). For example, if ClkDivide is 3, the clock source is divided by $2^3=8$.
BurstLength	The number of pulses generated in a single CV pulse burst. Legal values are from 1 to 256.

Return Value: None.

10.2.4 Register Access

The Register Access functions provide a convenient method of accessing the control registers of the S1D13706 controller using byte, word or dword widths.

To reduce the overhead of the function call as much as possible, two steps were taken:

- To gain maximum efficiency on all compilers and platforms, byte and word size arguments are passed between the application and the HAL as unsigned integers. This typically allows a compiler to produce more efficient code for the platform.
- Index alignment for word and dword accesses is not tested. On non-Intel platforms attempting to access a word or dword on a non-aligned boundary may result in a processor trap. It is the responsibility of the caller to ensure that the requested index offset is correctly aligned for the target platform.
- The word and dword register functions will swap bytes if the endian of the host CPU differs from the S1D13706 (the S1D13706 is little-endian).

unsigned seReadRegByte(DWORD Index)

Description: This routine reads the register specified by Index and returns the value.

Parameters: Index Offset, in bytes, to the register to read.

Return Value: The least significant byte of the return value is the byte read from the register.

unsigned seReadRegWord(DWORD Index)

Description: This routine reads two consecutive registers as a word and returns the value.

Parameters: Index Offset to the first register to read.

Return Value: The least significant word of the return value is the word read from the S1D13706 registers.

DWORD seReadRegDword(DWORD Index)

Description: This routine reads four consecutive registers as a dword and returns the value.

Parameters: Index Offset to the first of the four registers to read.

Return Value: The return value is the dword read from the S1D13706 registers.

void seWriteRegByte(DWORD Index, unsigned Value)

Description: This routine writes Value to the register specified by Index.

Parameters: Index Offset to the register to be written
Value The value, in the least significant byte, to write to the register

Return Value: None

void seWriteRegWord(DWORD Index, unsigned Value)

Description: This routine writes the word contained in Value to the specified index.

Parameters: Index Offset to the register pair to be written.
Value The value, in the least significant word, to write to the registers.

Return Value: None.

void seWriteRegDword(DWORD Index, DWORD Value)

Description: This routine writes the value specified to four registers starting at Index.

Parameters: Index Offset to the first of four registers to be written to.
Value The dword value to be written to the registers.

Return Value: None.

10.2.5 Memory Access

The Memory Access functions provide convenient method of accessing the display memory on an S1D13706 controller using byte, word or dword widths.

To reduce the overhead of these function calls as much as possible, two steps were taken:

- To gain maximum efficiency on all compilers and platforms, byte and word size arguments are passed between the application and the HAL as unsigned integers. This typically allows a compiler to produce more efficient code for the platform.
- Offset alignment for word and dword accesses is not tested. On non-Intel platforms attempting to access a word or dword on a non-aligned boundary may result in a processor trap. It is the responsibility of the caller to ensure that the requested offset is correctly aligned for the target platform.
- These functions will not swap bytes if the endian of the host CPU differs from the S1D13706 (the S1D13706 is little-endian).

unsigned seReadDisplayByte(DWORD Offset)

Description: Reads a byte from the display buffer memory at the specified offset and returns the value.

Parameters: Offset Offset, in bytes, from start of the display buffer to the byte to read.

Return Value: The return value, in the least significant byte, is the byte read from display memory.

unsigned seReadDisplayWord(DWORD Offset)

Description: Reads one word from display buffer memory at the specified offset and returns the value.

Parameters: Offset Offset, in bytes, from start of the display buffer to the word to read.

Return Value: The return value, in the least significant word, is the word read from display memory.

DWORD seReadDisplayDword(DWORD Offset)

Description: Reads one dword from display buffer memory at the specified offset and returns the value.

Parameters: Offset Offset, in bytes, from start of the display buffer to the dword to read.

Return Value: The DWORD read from display memory.

void seWriteDisplayBytes(DWORD Offset, unsigned Value, DWORD Count)

Description: This routine writes one or more bytes to the display buffer at the offset specified by Offset.

Parameters: Offset Offset, in bytes, from start of display memory to the first byte to be written.

Value An unsigned integer containing the byte to be written in the least significant byte.

Count Number of bytes to write. All bytes will have the same value.

Return Value: None.

void seWriteDisplayWords(DWORD Offset, unsigned Value, DWORD Count)

Description: This routine writes one or more words to display memory starting at the specified offset.

Parameters:

Offset	Offset, in bytes, from the start of display memory to the first word to write.
Value	An unsigned integer containing the word to written in the least significant word.
Count	Number of words to write. All words will have the same value.

Return Value: None.

void seWriteDisplayDwords(DWORD Offset, DWORD Value, DWORD Count)

Description: This routine writes one or more dwords to display memory starting at the specified offset.

Parameters:

Offset	Offset, in bytes, from the start of display memory to the first dword to write.
Value	The value to be written to display memory.
Count	Number of dwords to write. All dwords will have the same value.

Return Value: None.

10.2.6 Color Manipulation

The functions in the Color Manipulation section deal with altering the color values in the Look-Up Table directly through the accessor functions and indirectly through the color depth setting functions.

Keep in mind that all lookup table data is contained in the upper six bits of each byte.

void seWriteLutEntry(int Index, BYTE *pRGB)

Description: seWriteLutEntry() writes one lookup table entry to the specified index of the lookup table.

Parameter:

Index	Offset to the lookup table entry to be modified (i.e. a 0 will write the first entry and a 255 will write the last lookup table entry).
pRGB	A pointer to a byte array of data to write to the lookup table. The array must consist of three bytes; the first byte contains the red value, the second byte contains the green value and the third byte contains the blue value.

Return Value: None

void seReadLutEntry(int Index, BYTE *pRGB)

Description: seReadLutEntry() reads one lookup table entry and returns the results in the byte array pointed to by pRGB.

Parameter:

Index	Offset to the lookup table entry to be read. (i.e. setting index to 2 returns the value of the third RGB element of the lookup table).
pRGB	A pointer to an array to receive the lookup table data. The array must be at least three bytes long. On return from this function the first byte of the array will contain the red data, the second byte will contain the green data and the third byte will contain the blue data.

Return Value: None.

void seWriteLut(BYTE *pRGB, int Count)

Description: seWriteLut() writes one or more lookup table entries starting at offset zero.

These routines are intended to allow setting as many lookup table entries as the current color depth allows.

Parameter:

pRGB	A pointer to an array of lookup table entry values to write to the LUT. Each lookup table entry must consist of three bytes. The first byte must contain the red value, the second byte must contain the green value and the third byte must contain the blue value.
Count	The number of lookup table entries to modify.

Return Value: None.

void seReadLut(BYTE *pRGB, int Count)

Description: seReadLut() reads one or more lookup table entries and returns the result in the array pointed to by pRGB. The read always begins at the first lookup table entry.

This routine allows reading all the lookup table elements used by the current color depth in one library call.

Parameters: pRGB A pointer to an array of bytes large enough to hold the requested number of lookup table entries. Each lookup table entry consists of three bytes; the first byte will contain the red data, the second the green data and the third the blue data.

Count The number of lookup table entries to read.

Return Value: None.

int seSetMode(unsigned BitsPerPixel)

Description: seSetMode() changes the color depth of the display and updates the appropriate LUT. Display memory is automatically released and then reallocated as necessary for the display resolution.

Note

seSetMode() was previously called seSetBitsPerPixel(). It is now recommended to call seSetMode() instead of seSetBitsPerPixel(). In addition, hardware display swapping is enabled or disabled, based on the requirements described in seEnableHardwareDisplaySwapping().

IMPORTANT

When the LCD color depth is changed, memory allocated for both the main window and sub-window display buffer is freed and the display buffer memory is reassigned. The application must redraw the main window display and re-initialize the sub-window (if used) and redraw the sub-window after calling seSetMode().

Parameters: BitsPerPixel The new color depth. BitsPerPixel can be one of the following:
1, 2, 4, 8, 16.

Return Value: ERR_OK Function completed successfully.
ERR_NOT_ENOUGH_MEMORY There is insufficient free display memory for the given bits-per-pixel mode and display resolution.
ERR_FAILED Function failed because of invalid BitsPerPixel.

void seUseMainWinImageForSubWin(void)

Description: This function instructs the HAL to use the image pointed to by the main window registers as the image to be used by the sub-window. The sub-window start address and sub-window line address offset registers are programmed accordingly.

Note

It is the responsibility of the caller to first free any memory used by the sub-window before calling this function.

Parameters: None.

Return Value: None.

unsigned seGetBitsPerPixel(void)

Description: seGetBitsPerPixel() returns the current color depth for the associated display surface.

Parameters: None.

Return Value: The color depth of the surface. This value will be 1, 2, 4, 8, or 16.

10.2.7 Virtual Display

```
int seVirtInit(DWORD Width, DWORD Height)
int seMainWinVirtInit(DWORD Width, DWORD Height)
int seSubWinVirtInit(DWORD Width, DWORD Height)
int seMainAndSubWinVirtInit(DWORD width, DWORD height)
```

Description: These functions prepare the S1D13706 to display a virtual image.

“Virtual Image” describes the condition where the image contained in display memory is larger than the physical display. In this situation the physical display is used as a window into the larger display memory area (display surface). Panning (right/left) and scrolling (up/down) are used to move the display in order to view the entire image a portion at a time.

seVirtInit() prepares the current active surface for a virtual image display. Memory is allocated based on width, height and the current color depth.

seMainWinVirtInit() initializes and allocates memory for the main window based on width and height and color depth.

seSubWinVirtInit() initializes and allocates memory for the sub-window based on current width and height and color depth.

seMainAndSubWinVirtInit() initializes and allocates one block of memory for both the main window and sub-window based on width and height and color depth.

Memory previously allocated for the given display surface is released then reallocated to the larger size.

Note

The width programmed may be larger than that requested in the respective function argument. This is to ensure that the value programmed into the address offset registers is a multiple of 4 bytes. For example, suppose seVirtInit(240, 320) is called in SwivelView 90° and at 1 bits-per-pixel. Since four bytes corresponds to 32 pixels in 1 bits-per-pixel mode, the width must be a multiple of 32. Since 240 is not a multiple of 32, the width is automatically changed to the next available multiple, which in this case is 256.

Parameters:	Width	The desired virtual width of the display in pixels. Width must be a multiple of the number of pixels contained in one dword of display memory. In other words, Width must be a multiple of $32 \div \text{bits-per-pixel}$.
	Height	The desired virtual height of the display in pixels. The HAL performs internal memory management to ensure that all display surfaces have sufficient memory for operation. The Height parameter is required so the HAL can determine the amount of memory the application requires for the virtual image.
Return Value:	ERR_OK	The function completed successfully.
	ERR_HAL_BAD_ARG	The requested virtual dimensions are smaller than the physical display size.

ERR_NOT_ENOUGH_MEMORY There is insufficient free display memory to set the requested virtual display size.

void seVirtPanScroll(DWORD x, DWORD y)
void seMainWinVirtPanScroll(DWORD x, DWORD y)
void seSubWinVirtPanScroll(DWORD x, DWORD y)
void seMainAndSubWinVirtPanScroll(DWORD x, DWORD y)

Description: When displaying a virtual image the physical display is smaller than the virtual image contained in display memory. In order to view the entire image, the display is treated as a window into the virtual image.

These functions allow an application to pan (right and left) and scroll (up and down) the display over the virtual image.

seVirtPanScroll() will pan and scroll the current active surface.

seMainWinVirtPanScroll() and seSubWinVirtPanScroll() will pan and scroll the surface indicated in the function name.

seMainAndSubWinVirtPanScroll() will pan and scroll the surface which is used by both the main and sub-windows.

Note

Panning operations are limited to 32-bit boundaries; x must be a multiple of 32 ÷ bits-per-pixel.

Parameters:

x	The new x offset, in pixels, of the upper left corner of the display. x must be a multiple of 32 ÷ bits-per-pixel.
y	The new y offset, in pixels, of the upper left corner of the display.

Return Value: None.

10.2.8 Drawing

Functions in this category perform primitive drawing on the specified display surface. Supported drawing primitive include pixels, lines, rectangles, ellipses and circles.

All drawing functions are in relation to the given SwivelView mode. For example, co-ordinate (0, 0) is always the top left corner of the image, but this is physically in different corners of the panel depending on what SwivelView mode is selected.

void seSetPixel(long x, long y, DWORD Color)
void seSetMainWinPixel(long x, long y, DWORD Color)
void seSetSubWinPixel(long x, long y, DWORD Color)

Description: These routines set a pixel at the location (x,y) with the specified color.

Use seSetPixel() to set one pixel on the current active surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seSetMainWinPixel() and seSetSubWinPixel() to set one pixel on the surface indicated in the function name.

If no memory was allocated to the surface, these functions return without writing to display memory.

Parameters:	x	The X co-ordinate, in pixels, of the pixel to set.
	y	The Y co-ordinate, in pixels, of the pixel to set.
	Color	Specifies the color to draw the pixel with. Color is interpreted differently at different color depths. At 1, 2, 4 and 8 bpp, display colors are derived from the lookup table values. The least significant byte of Color forms an index into the lookup table. At 16 bpp the lookup table is bypassed and each word of display memory forms the color to display. In this mode the least significant word describes the color to draw the pixel with in 5-6-5 RGB format.

Return Value: None.

DWORD seGetPixel(long x, long y)
DWORD seGetMainWinPixel(long x, long y)
DWORD seGetSubWinPixel(long x, long y)

Description: Returns the pixel color at the specified display location.

Use seGetPixel() to read the pixel color at the specified (x,y) co-ordinates on the current active surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seGetMainWinPixel() and seGetSubWinPixel() to read the pixel color at the specified (x,y) co-ordinate on the display surface referenced in the function name.

Parameters:

x	The X co-ordinate, in pixels, of the pixel to read
y	The Y co-ordinate, in pixels, of the pixel to read

Return Value: The return value is a dword describing the color read at the (x,y) co-ordinate. Color is interpreted differently at different color depths.

If no memory was allocated to the surface, the return value is (DWORD) -1.

At 1, 2, 4 and 8 bpp, display colors are derived from the lookup table values. The return value is an index into the lookup table. The red, green and blue components of the color can be determined by reading the lookup table values at the returned index.

At 16 bpp the lookup table is bypassed and each word of display memory form the color to display. In this mode the least significant word of the return value describes the color as a 5-6-5 RGB value.

```
void seDrawLine(long x1, long y1, long x2, long y2, DWORD Color)
void seDrawMainWinLine(long x1, long y1, long x2, long y2, DWORD Color)
void seDrawSubWinLine(long x1, long y1, long x2, long y2, DWORD Color)
```

Description: These functions draw a line between two points in the specified color.

Use seDrawLine() to draw a line on the current active surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seDrawMainWinLine() and seDrawSubWinLine() to draw a line on the surface referenced by the function name.

If no memory was allocated to the surface, these functions return without writing to display memory.

Parameters:

x1	The X co-ordinate, in pixels, of the first endpoint of the line to be drawn.
y1	The Y co-ordinate, in pixels, of the first endpoint of the line to be drawn.
x2	The X co-ordinate, in pixels, of the second endpoint of the line to be drawn.
y2	The Y co-ordinate, in pixels, of the second endpoint of the line to be drawn.
Color	<p>Specifies the color to draw the line with. Color is interpreted differently at different color depths.</p> <p>At 1, 2, 4 and 8 bpp, display colors are derived from the lookup table values. The least significant byte of Color is an index into the lookup table.</p> <p>At 16 bpp the lookup table is bypassed and each word of display memory forms the color to display. In this mode the least significant word describes the color to draw the line with in 5-6-5 RGB format.</p>

Return Value: None.

```
void seDrawRect(long x1, long y1, long x2, long y2, DWORD Color, BOOL SolidFill)
void seDrawMainWinRect(long x1, long y1, long x2, long y2, DWORD Color, BOOL SolidFill)
void seDrawSubWinRect(long x1, long y1, long x2, long y2, DWORD Color, BOOL SolidFill)
```

Description: These routines draw a rectangle on the screen in the specified color. The rectangle is bounded on the upper left by the co-ordinate (x1, y1) and on the lower right by the co-ordinate (x2, y2). The SolidFill parameter allows the programmer to select whether to fill the interior of the rectangle or to only draw the border.

Use seDrawRect() to draw a rectangle on the current active display surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seDrawMainWinRect() and seDrawSubWinRect() to draw a rectangle on the display surface indicated by the function name.

If no memory was allocated to the surface, these functions return without writing to display memory.

Parameters:

x1	The X co-ordinate, in pixels, of the upper left corner of the rectangle.
y1	The Y co-ordinate, in pixels, of the upper left corner of the rectangle.
x2	The X co-ordinate, in pixels, of the lower right corner of the rectangle.
y2	The Y co-ordinate, in pixels, of the lower right corner of the rectangle.
Color	Specifies the color to draw the line with. Color is interpreted differently at different color depths. At 1, 2, 4 and 8 bpp, display colors are derived from the lookup table values. The least significant byte of Color is an index into the lookup table. At 16 bpp the lookup table is bypassed and each word of display memory forms the color to display. In this mode the least significant word describes the color to draw the line with in 5-6-5 RGB format.
SolidFill	A boolean value specifying whether to fill the interior of the rectangle. Set to FALSE to draw only the rectangle border. Set to TRUE to instruct this routine to fill the interior of the rectangle.

Return Value: None

```
void seDrawCircle(long xCenter, long yCenter, long Radius, DWORD Color)  
void seDrawMainWinCircle(long xCenter, long yCenter, long Radius, DWORD Color)  
void seDrawSubWinCircle(long xCenter, long yCenter, long Radius, DWORD Color)
```

Description: These routines draw a circle on the screen in the specified color. The circle is centered at the co-ordinate (x, y) and is drawn with the specified radius and Color. These functions only draw the border of the circle; there is no solid fill feature.

Use seDrawCircle() to draw the circle on the current active display surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seDrawMainWinCircle() and seDrawSubWinCircle() draw the circle on the display surface indicated by the function name

If no memory was allocated to the surface, these functions return without writing to display memory.

Parameters:	x	The X co-ordinate, in pixels, of the center of the circle.
	y	The Y co-ordinate, in pixels, of the center of the circle.
	Radius	Specifies the radius of the circle in pixels.
	Color	Specifying the color to draw the circle. Color is interpreted differently at different color depths. At 1, 2, 4 and 8 bpp display colors are derived from the lookup table values. The least significant byte of Color is an index into the lookup table. At 16 bpp the lookup table is bypassed and each word of display memory forms the color to display. In this mode the least significant word describes the color to draw the circle with in 5-6-5 RGB format.

Return Value: None.

```

void seDrawEllipse(long xc, long yc, long xr, long yr, DWORD Color)
void seDrawMainWinEllipse(long xc, long yc, long xr, long yr, DWORD Color)
void seDrawSubWinEllipse(long xc, long yc, long xr, long yr, DWORD Color)

```

Description: These routines draw an ellipse on the screen in the specified color. The ellipse is centered at the co-ordinate (x, y) and is drawn in the specified color with the indicated radius for the x and y axis. These functions only draw the border of the ellipse; there is no solid fill feature.

Use seDrawEllipse() to draw the ellipse on the current active display surface. See seSetMainWinAsActiveSurface() and seSetSubWinAsActiveSurface() for information about changing the active surface.

Use seDrawMainWinEllipse() and seDrawSubWinEllipse() to draw the ellipse on the display surface indicated by the function name.

If no memory was allocated to the surface, these functions return without writing to display memory.

Parameters:	xc	The X co-ordinate, in pixels, of the center of the ellipse.
	yc	The Y co-ordinate, in pixels, of the center of the ellipse.
	xr	A long integer specifying the X radius of the ellipse, in pixels.
	yr	A long integer specifying the Y radius of the ellipse, in pixels.
	Color	A dword specifying the color to draw the ellipse. Color is interpreted differently at different color depths.

At 1, 2, 4 and 8 bpp display colors are derived from the lookup table values. The least significant byte of Color is an index into the lookup table.

At 16 bpp the lookup table is bypassed and each word of display memory forms the color to display. In this mode the least significant word describes the color to draw the circle with in 5-6-5 RGB format.

Return Value: None.

10.2.9 Register/Display Memory

The S5U13706 Evaluation Board utilizes 2M bytes of display memory address space. The S1D13706 contains 80K bytes of embedded SDRAM.

In order for an application to directly access the S1D13706 display memory and registers, the following two functions are provided.

DWORD seGetLinearDisplayAddress(void)

Description: This function returns the linear address for the start of physical display memory.

Parameters: None.

Return Value: The return value is the linear address of the start of display memory. A linear address is a 32-bit offset, in CPU address space.

DWORD seGetLinearRegAddress(void)

Description: This function returns the linear address of the start of S1D13706 control registers.

Parameters: None.

Return Value: The return value is the linear address of the start of S1D13706 control registers. A linear address is a 32-bit offset, in CPU address space.

10.3 Porting LIBSE to a new target platform

Building Epson applications like a simple HelloApp for a new target platform requires the following:

- HelloApp code.
- 13706HAL library.
- LIBSE library which contains target specific code for embedded platforms.

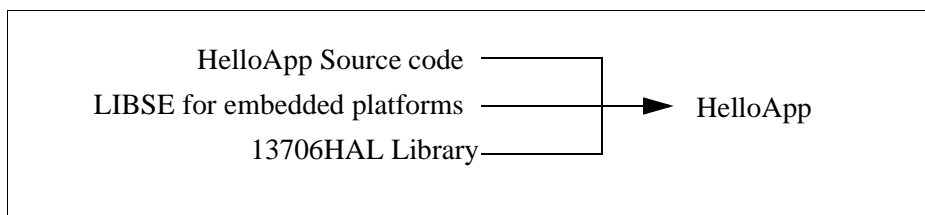


Figure 10-1: Components needed to build 13706 HAL application

For example, when building HELLOAPP.EXE for the x86 windows 32-bit platform, you need the HELLOAPP source files, the 13706HAL library and its include files, and some Standard C library functions (which in this case would be supplied by the compiler as part of its run-time library). As this is a 32-bit windows .EXE application, you do not need to supply start-up code that sets up the chip selects or interrupts, etc... What if you wanted to build the application for an SH-3 target, one not running windows?

Before you can build that application to load onto the target, you need to build a C library for the target that contains enough of the target specific code (like `putch()` and `getch()`) to let you build the application. Epson supplies the LIBSE for this purpose, but your compiler may come with one included. You also need to build the 13706HAL library for the target. This library is the graphics chip dependent portion of the code. Finally, you need to build the final application, linked together with the libraries described earlier. The following examples assume that you have a copy of the complete source code for the S1D13706 utilities, including the makefiles, as well as a copy of the GNU Compiler v2.8.1 for Hitachi SH3. These are available on the Epson Electronics America Website at www.eea.epson.com.

10.3.1 Building the LIBSE library for SH3 target example

In the LIBSE files, there are two main types of files:

- C and assembler files that contain the target specific code.
- makefiles that describe the build process to construct the library.

The C and assembler files contain some platform setup code (evaluation board communications, chip selects) and jumps into the main entry point of the C code that is contained in the applications main() function. For our example, the startup file, which is sh3entry.c, performs some board configuration (board communications and assigning memory blocks with chip selects) and a jump into the applications main() function.

In the embedded targets, putch (xxxputch.c) and getch (xxxgetch.c) resolve to serial character input/output. For SH3, much of the detail of handling serial IO is hidden in the monitor of the evaluation board, but in general the primitives are fairly straight forward, providing the ability to get characters to/from the serial port.

For our target example, the nmake makefile is makesh3.mk. This makefile calls the Gnu compiler at a specific location (TOOLDIR), enumerates the list of files that go into the target and builds a .a library file as the output of the build process.

To build the software for our target example, type the following at the root directory of the software (i.e. C:\13706).

```
make "TARGETS=SH3" "BUILDS=release"
```

11 Sample Code

Example source code demonstrating programming the S1D13706 using the HAL library is available on the internet at www.eea.epson.com.