

Contents Page

1	GENERAL FEATURES	1
2	ADVANCED FEATURES	2
3	APPLICATION NOTE I – ADVANCED FEATURES	3
	3.1 Contrast Control Calculation	3
	3.2 OTP Setting and Programming	4
4	APPLICATION NOTE II – HARDWARE CONFIGURATION	5
	4.1 COG Application - ITO Layout Requirement	5
	4.2 Electrostatic Discharge (ESD) Protection Circuit on RESET Pin Handling	5
	4.3 Connection in Bus Interface Mode	6
	4.4 COM Output Scan Direction	7
5	APPLICATION NOTE III – COLOR CONFIGURATION	10
	5.1 RGB Concept in LCD Driver	10
	5.2 RGB Arrangement for Color Data	11
	5.3 Data Bus Arrangement for Color Data	12
	5.4 Color Look Up Table (Color LUT)	13
	5.5 Concept of the Color LUT Content	14
6	APPLICATION NOTE IV - PROGRAMMING NOTE	17
	6.1 Prototypes Examples	17
	6.2 Graphic Command Examples	20
	6.3 Graphic Command Application Examples	23

Application of CSTN Series

Application Note

A series of single-chip CMOS color STN LCD driver with controller for dot-matrix graphic liquid crystal display system – SSD1780, SSD1770, SSD1781, SSD1783, SSD1788 and SSD1789. They are all highly integrated single chip solution with low power consumption for different display resolutions and color depth.

1 GENERAL FEATURES

Product Name	SSD1780	SSD1770	SSD1781	SSD1783	SSD1788	SSD1789	
Max. Display Size (Seg x Com)	104RGB x 80	104RGB x 80	132RGB x 132	132RGB x 160	98RGB x 68	132RGB x 132	
Embedded Icon Line	1	1	-	-	1	-	
Max. no. of Colors	4096	4096	262k	262k	4096	262k	
Supply Voltage	1.8 ~ 3.6V	2.4 ~ 3.6V	2.4 ~ 3.6V	2.4 ~ 3.6V	2.4 ~ 3.6V	2.4 ~ 3.6V	
Max. LCD Voltage	13.5V	13.5V	18V	18V	13.5V	18V	
Current Consumption (Typ.)	350uA	250uA	500uA	500uA	230uA	500uA	
Die Size (mm)	17.65 x 1.88	17.74 x 1.88	22.10 x 1.96	22.10 x 1.96	15.68 x 1.65	20.99 x 1.78	
Die Thickness (um)	457	457	457	457	457	457	
Min. Bump Size (um)	28 x 130	28 x 130	27 x 118	27 x 118	27 x 118	27 x 118	
Min. Bump Pitch (um)	43.4	43.4	41.8	41.8	42	41.8	
Typ. Bump Height (um)	15	15	15	15	15	15	
Interface	8bit Parallel	7.7MHz 6800/8080	6MHz 6800/8080	6800/8080	6800/8080	6800/8080	
	16bit Parallel	No	No	6800/8080	6800/8080	6800/8080	
	Serial Interface	20MHz 3 or 4 wires	15MHz 3 or 4 wires	3 or 4 wires	3 or 4 wires	3 or 4 wires	3 or 4 wires
	I2C	No	No	No	No	No	No
Features	Graphic RAM (bits)	104x81x12 = 101,088	104x81x12 = 101,088	132x168x18 = 399,168	132x168x18 = 399,168	98x3x68x4 = 79,968	132x132x18 = 313,632
	Build-in Booster Capacitor	Yes	Yes	No	No	Yes	No
	Build-in Divider Capacitor	Yes	Yes	No	No	Yes	No
	DC/DC Level	3 ~ 6X	3 ~ 6X	4 ~ 7X	4 ~ 7X	3 ~ 6X	4 ~ 7X
	Bias Ratio	1:4 ~ 1:10	1:4 ~ 1:10	1:7 ~ 1:14	1:7 ~ 1:14	1:4 ~ 1:9	1:7 ~ 1:10
	Partial Display	1/8 ~ 1/81	1/ ~ 1/81	1/32 ~ 1/132	1/32 ~ 1/160	1/8 ~ 1/68	1/32 ~ 1/132
	Contrast Level	8 internal gain + 64 steps contrast adjustment	8 internal gain + 64 steps contrast adjustment	8 internal gain + 64 steps contrast adjustment	8 internal gain + 64 steps contrast adjustment	8 internal gain + 64 steps contrast adjustment	8 internal gain + 64 steps contrast adjustment
	Frame Frequency (Hz)	77.5	66	75	75	78	75
	Operation Temperature (°C)	-40 ~ +85	-40 ~ +85	-40 ~ +85	-40 ~ +85	-40 ~ +85	-40 ~ +85

This document contains information on a new product under definition stage. Solomon Systech Ltd. reserves the right to change or discontinue this product without notice.



2 ADVANCED FEATURES

Product Name	SSD1780	SSD1770	SSD1781	SSD1783	SSD1788	SSD1789
Sleep Mode	Yes	Yes	Yes	Yes	Yes	Yes
N-Line Inversion	Yes	Yes	Yes	Yes	Yes	Yes
Temperature Compensation	4 Setting	4 Setting	4 Setting	4 Setting	4 Setting	4 Setting
On Chip OTP	Yes 1X contrast step	Yes 2X contrast step	Yes	Yes	Yes	Yes
2D GIGA	Yes	Yes	Yes	Yes	Yes	Yes
Grayscale Control	4 bits PWM, 4 bits FRC, 2 bits PWM + 2 bits FRC,	4 bits PWM, 4 bits FRC, 2 bits PWM + 2 bits FRC,	6bits FRC, 4 bits PWM + 2 bits FRC, 5 bits PWM + 1 bit FRC	6bits FRC, 4 bits PWM + 2 bits FRC, 5 bits PWM + 1 bit FRC	4 bits PWM, 4 bits FRC, 2 bits PWM + 2 bits FRC,	6bits FRC, 4 bits PWM + 2 bits FRC, 5 bits PWM + 1 bit FRC
Interlace COM Output Pin	No	No	Yes (Software)	Yes (Software)	Yes	Yes (Software)

CSTN series embedded graphic engine to have graphic acceleration function - draw line, draw rectangular, draw circle, area fill color, copy, clear and dim window. The graphic engine provides rapid drawing and display function by using inventive 2D graphic command and facilitate user to develop their complicated graphic and animation program. The clear demarcation of bitmap-based operation handled by MCU and rendering operation handled by graphic engine to improves the system performance.

3 APPLICATION NOTE I – Advanced Features

3.1 Contrast Control Calculation

81 Hex is a command to adjust the contrast of the LCD panel by changing LCD driving voltage, V_{out} , provided by the on-chip power circuits. V_{out} is set with 6-bits 64 contrast steps (α) in the contrast control register by a set of compound commands. The command will set the feedback gain of the internal regulator. There are 8 internal regulator gains which are used to adjust the V_{out} level by enable any one out of the 8 internal resistor settings (IRS) for regulator gain when using internal regulator resistor network. The below formula can provide a quick estimation of the voltage generated by the software setting in the driver IC.

$$V_{out} = \text{Internal Regulator Gain} * \left(1 - \frac{63 - \alpha}{210}\right) * V_{ref}$$

$$; \text{ where Internal Regulator Gain} = \left(1 + \frac{R_2}{R_1}\right) \text{ and } V_{ref} = 1.7$$

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) * \left(1 - \frac{63 - \alpha}{210}\right) * V_{ref}$$

Internal Regulator Gain (1+R2/R1)	Y ₂ Y ₁ Y ₀ = 000	Y ₂ Y ₁ Y ₀ = 001	Y ₂ Y ₁ Y ₀ = 010	Y ₂ Y ₁ Y ₀ = 011	Y ₂ Y ₁ Y ₀ = 100	Y ₂ Y ₁ Y ₀ = 101	Y ₂ Y ₁ Y ₀ = 110	Y ₂ Y ₁ Y ₀ = 111
SSD1780	2.84	3.71	4.57	5.44	6.30	7.16	8.03	8.89
SSD1770	2.84	3.71	4.57	5.44	6.30	7.16	8.03	8.89
SSD1781	7.02	7.89	8.76	9.63	10.5	11.37	12.24	13.11
SSD1783	7.02	7.89	8.76	9.63	10.5	11.37	12.24	13.11
SSD1788	2.84	3.71	4.57	5.44	6.30	7.16	8.03	8.89
SSD1789	3.95	4.7	5.64	6.58	7.52	8.46	9.4	10.34

3.2 OTP Setting and Programming

OTP (One Time Programming) is a channel to adjust V_{out} at module level to achieve the best contrast on every LCD modules. In order to have variations on every LCD module, there are affecting the module performance and contrast level, OTP feature is going to adjust the V_{out} by external high voltage applying to the OTP cells through the V_{out} .

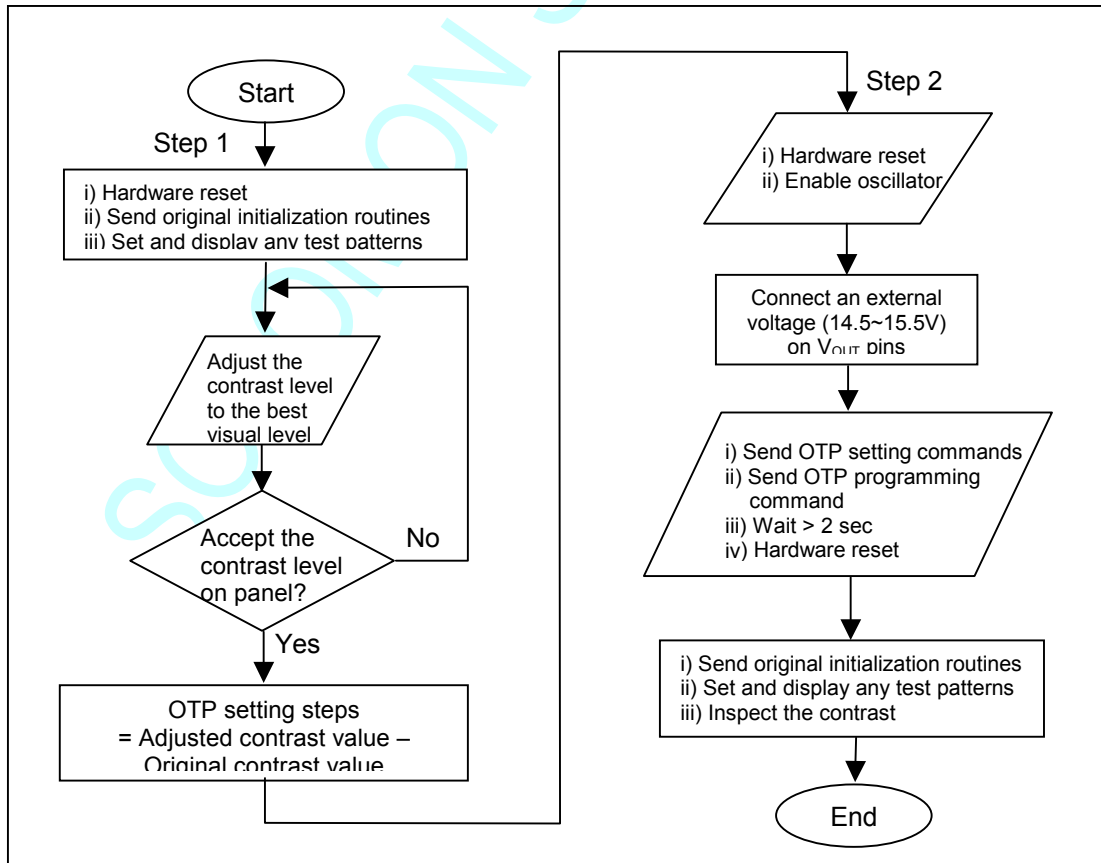
To process the OTP setting and programming, there are 2 major steps.

Step 1. Find the target voltage with OTP offset

- a) Hardware Reset (sending an active low reset pulse to RES pin)
- b) Send original initialization routines
- c) Set and display any test patterns
- d) Adjust the contrast value until there is the best visual contrast
- e) OTP setting steps = Contrast value of the best visual contrast - Contrast value of original initialization

Step 2. OTP programming

- f) Hardware Reset (sending an active low reset pulse to RES pin)
- g) Enable Oscillator and Exit Sleep Mode
- h) Connect an external V_{OUT} circuit (refer to datasheet for details)
- i) Send OTP setting commands that we find in step 1
- j) Send OTP programming command
- k) Wait at least 2 seconds
- l) Hardware Reset
- m) Verify the result by repeating step 1. (b) – (c)



** Remark: Only one time for each of OTP bit to be programmed to '1'.

4 APPLICATION NOTE II – Hardware Configuration

4.1 COG Application - ITO Layout Requirement

In order to optimize the performance, the resistance due to the ITO must be minimized. Below gives a guideline on how to optimize the ITO traces for a 10nF-loading panel.

For SSD1780/1770/1788:

Below 500 ohm : I/O pin (D0 ~ D7, D/C, R/W, E(RD), /CS and /RES

Below 30 ohm : VSS < RVSS < CVSS < VCI < VDD (Resistance for VSS should be smallest)

Below 100 ohm : Vout

For SSD1781/1783/1789:

Below 300 ohm : I/O pin (D0 ~ D7, D/C, R/W, E(RD), /CS and /RES

Below 30 ohm : VSS < VCI < VDD (Resistance for VSS should be smallest)

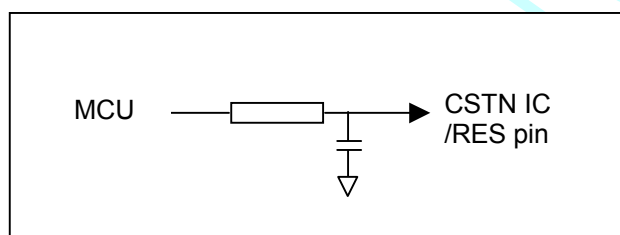
Below 60 ohm : Vout, VL2 ~ VL5

Below 60 ohm : C4P, C2P, C2N, C1P, C1N, C1Y, C3P

4.2 Electrostatic Discharge (ESD) Protection Circuit on RESET Pin Handling

The discharge sometimes will affect the normal operation of the device, causing data loss inside RAM or internal registers, or even re-initialize / reset the device. The driver has built-in protection circuit to protect the I/O pins from external charge or ESD.

In order to strength and avoid resetting the IC during electrostatic discharge, it is suggested to design a protection circuit across the IC reset pin and MCU to prevent any unexpected external interference or electrostatic discharge, a common example circuit is shown on below.



The reset pin of the driver is connected directly to one of the port of the MCU, when external charge is brought near or ESD through the device, it will discharge by finding its way to the shortest path to ground.

In addition, there is more protection can be done by adding a resistor in series between the reset pin of the driver IC and the I/O port of the MCU. This will create a filter effect to eliminate external noise entering the reset pin of the driver IC.

Adoption of protection circuit and the value of passive component used, it depends on the application printed circuit board design and it is recommended to test and evaluate to find out the best capacitor value for a particular application.

4.3 Connection in Bus Interface Mode

CSTN LCD driver series offers a selection of microcontroller interfaces by setting pins PS0, PS1 and PS2 namely the 6800 parallel, 8080 parallel, 3-wires serial and 4-wires serial interfaces. Inputs of PS0 to PS2 must be connected to either VDD or VSS where VDD is 1 and VSS is 0 shown on below table.

		Serial Mode		8-bits Parallel		16-bits Parallel	
		3 wires	4 wires	6800	8080	6800	8080
SSD1780	PS0	0	0	1	1	-	-
SSD1770	PS1	1	0	1	0	-	-
SSD1788							
SSD1781 SSD1783 SSD1789	PS0	1	0	1	0	0	1
	PS1	0	0	1	1	0	1
	PS2	0	0	0	1	1	1

Parallel Mode:

In 6800/8080 8-bits parallel mode: D0 ~ D7, E(/RD), R/W(/WR), D/C are used as I/O control

In 6800/8080 16-bits parallel mode: D0 ~ D15, E(/RD), R/W(/WR), D/C are used as I/O control

Serial Mode:

D0 ~ D7, E(/RD) and R/W(/WR) should connected to VDD or VSS, please refer to datasheet for different driver connection.

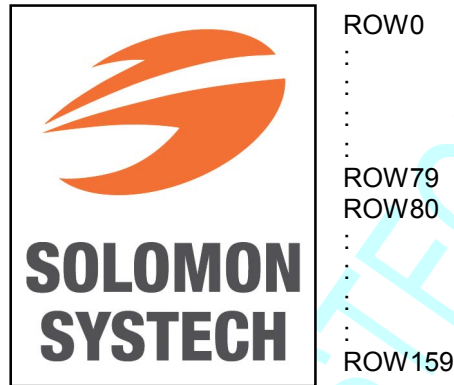
In 3-wires serial mode: SDA, SCK and R/W are used as I/O control

In 4-wires serial mode: SDA, SCK, R/W and D/C are used as I/O control

4.4 COM Output Scan Direction

This command sets the scan direction of the COM output allowing layout flexibility in LCD module assembly. In addition, the display will have immediate effect once this command is issued. That is, if this command is sent during normal display, the graphic display will have vertical flipping effect.

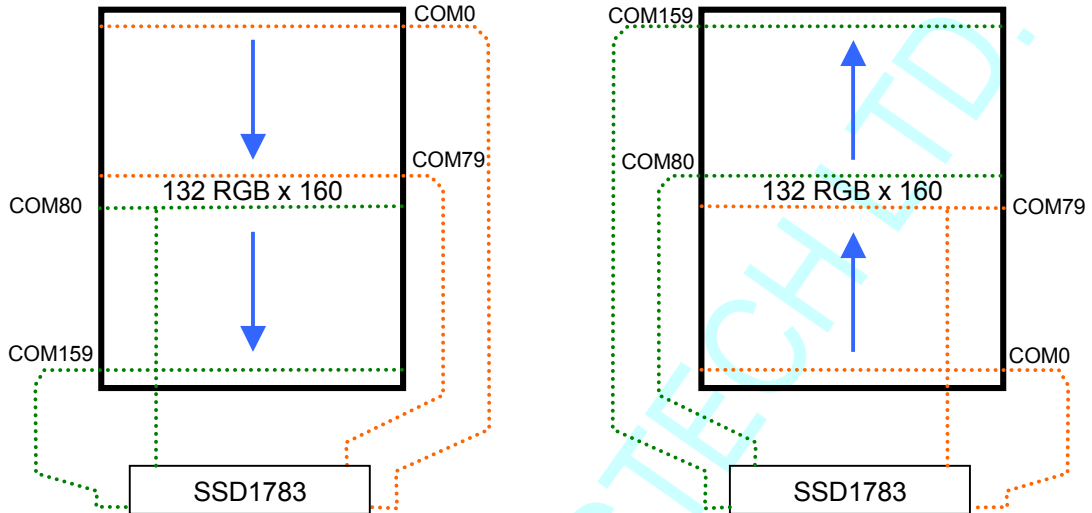
In case of a graphic pattern is shown as below and the pattern is already located at the memory of driver IC – SSD1783 (132 RGB x 160 resolution). The pattern will mapped to COMMOM according to the setting of COM Output Scan Direction (BB Hex command).



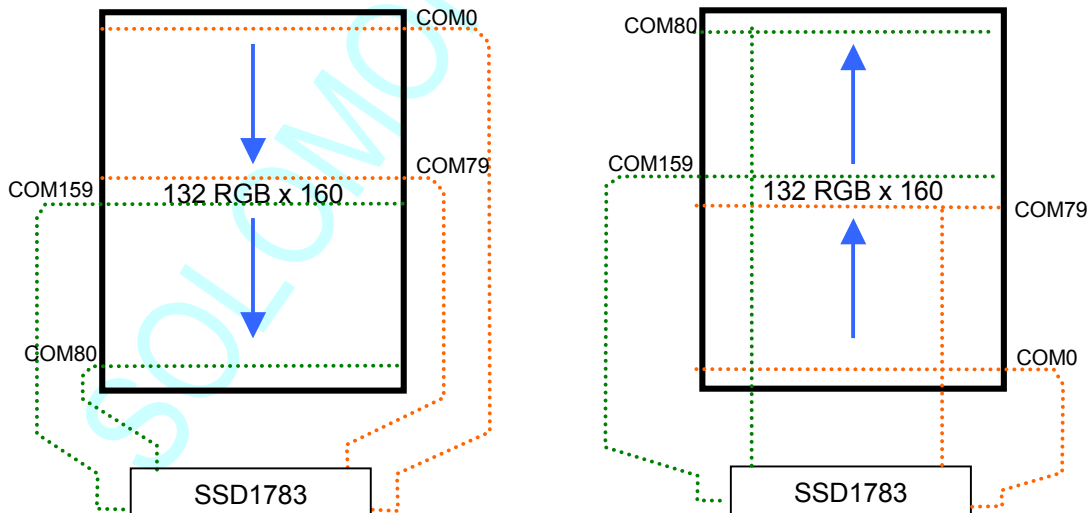
D/C	Hex	Command	Description
0	BB	Set COM Output Scan Direction	X2 X1 X0 ROW0...ROW65 ROW66...ROW131
1	00 ~ 03		0 0 0 COM0 ->COM65 COM66 -> COM131 (POR)
			0 0 1 COM0 ->COM65 COM131<-COM66
			0 1 0 COM65<-COM0 COM66 -> COM131
			0 1 1 COM65<-COM0 COM131<-COM66

Following is the examples as different setting of COM output scan direction for flexible panel layout design by using SSD1783 with 132 RGB x 160 resolutions.

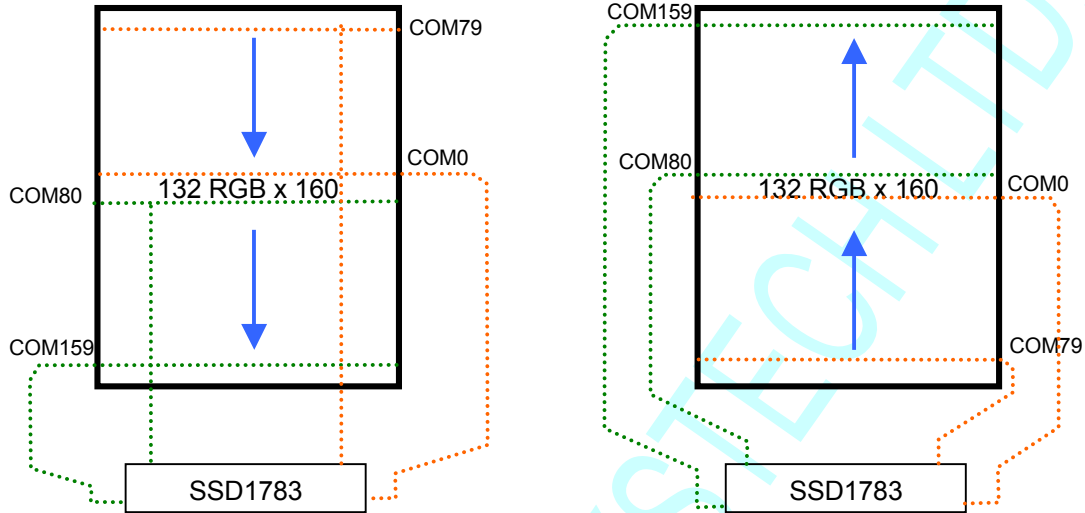
D/C	Hex	Command	Description
0	BB	Set COM Output Scan Direction	X2 X1 X0 ROW0...ROW79 ROW80...ROW159
1	00		0 0 0 COM0 ->COM79 COM80 -> COM159 (POR)



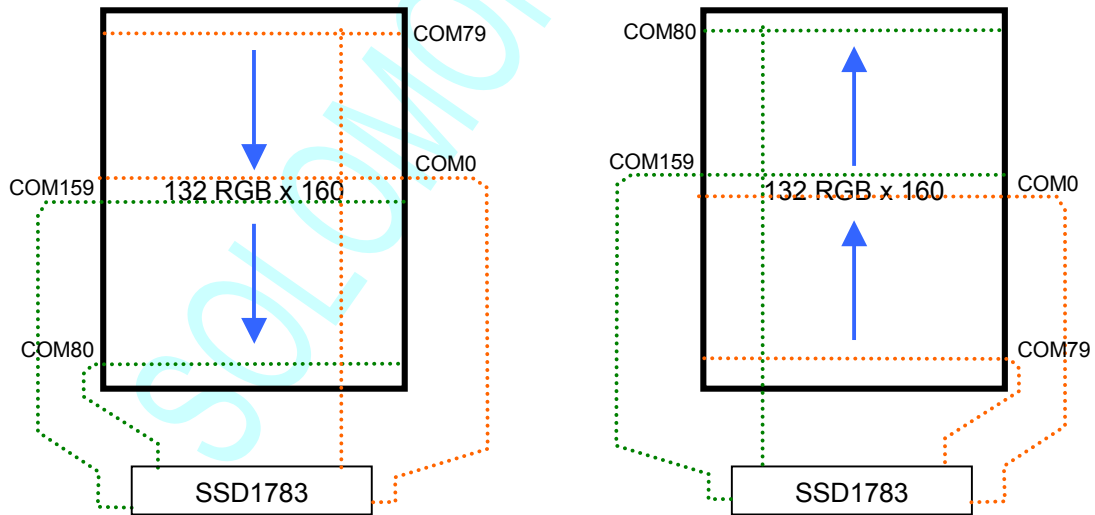
D/C	Hex	Command	Description
0	BB	Set COM Output Scan Direction	X2 X1 X0 ROW0...ROW79 ROW80...ROW159
1	01		0 0 1 COM0 ->COM79 COM159<-COM80



D/C	Hex	Command	Description
0	BB	Set COM Output Scan Direction	X2 X1 X0 ROW0...ROW79 ROW80...ROW159
1	02		0 1 0 COM79<-COM0 COM80 -> COM159



D/C	Hex	Command	Description
0	BB	Set COM Output Scan Direction	X2 X1 X0 ROW0...ROW79 ROW80...ROW159
1	03		0 1 1 COM79<-COM0 COM159<-COM80



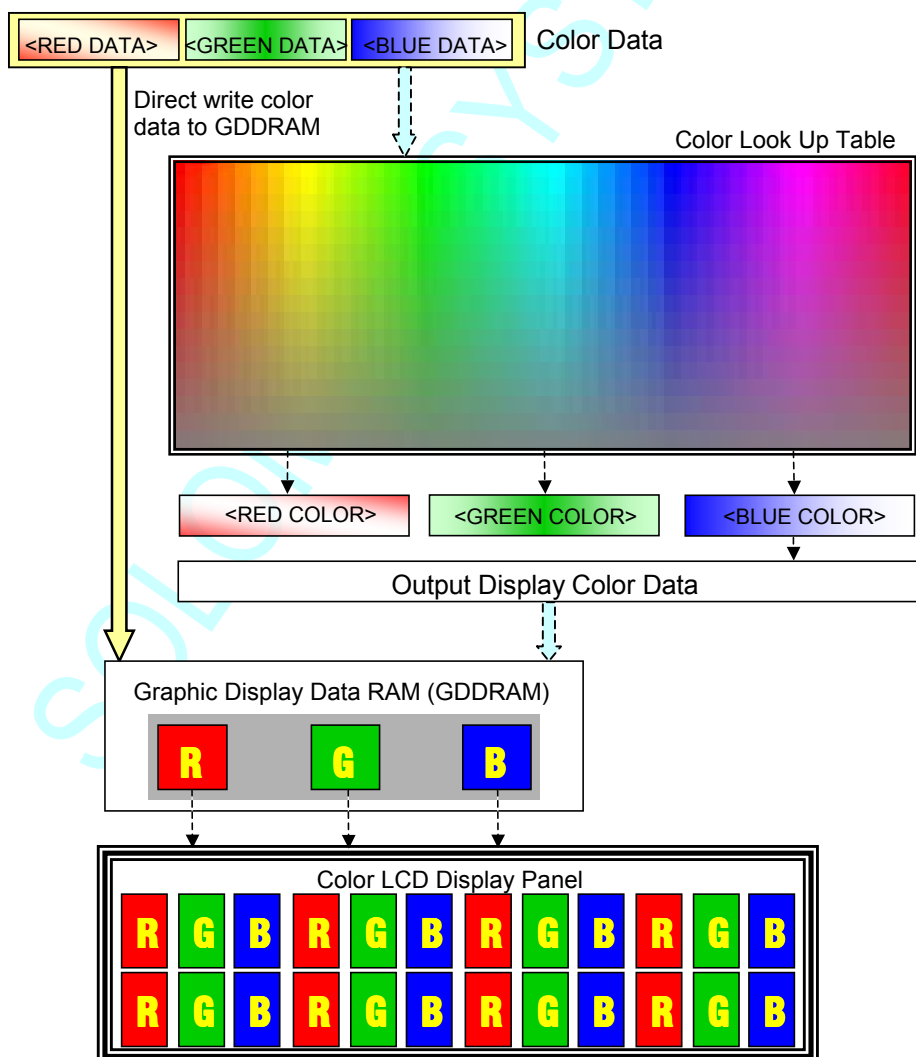
5 APPLICATION NOTE III – Color Configuration

5.1 RGB Concept in LCD Driver

Color STN LCD is one of LCD technology similar with monochrome and gray-scale, which realize color image forming by color filter layer with R,G,B color factor. It express color by combination of penetrate light through R,G,B of color filter and liquid crystal. The combination can express various colors according to gray-scale and number of color determined by driver IC. Below table is the summary of CSTN gray scale level of SSL driver IC.

IC Data Output Color	Gray Level			Color
	Red	Green	Blue	
8-bits	8	8	4	256
12-bits	16	16	16	4096
16-bits	32	64	32	65K
18-bits	64	64	64	262K

For CSTN LCD driver programming, it is also very similar with gray-scale programming, the color display data write to graphic display data RAM (GDDRAM) via 8 or 16-bits data bus provided by driver IC, and those color data will process under preset look up table and output to the LCD panel by LCD driver.



5.2 RGB Arrangement for Color Data

As CSTN LCD drivers support 262K maximum display color, and selectable 65K, 4096 and 256 color with internal preset color look up table, there are four different RGB arrangement for color data.

262K color : 18bits/pixel, color data with 6R-6G-6B write to graphic display data RAM (GDDRAM)

65K color : 16bits/pixel, color data with 5R-6G-5B write to GDDRAM

4096 color : 12bits/pixel, color data with 4R-4G-4B write to GDDRAM

256 color : 8bits/pixel, color data with 3R-3G-2B write to GDDRAM

262,144 Color	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18-bit/pixel	R	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B	B

65,536 Color	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit/pixel	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

4,096 Color	11	10	9	8	7	6	5	4	3	2	1	0
12-bit/pixel	R	R	R	R	G	G	G	G	B	B	B	B

256 Color	7	6	5	4	3	2	1	0
8-bit/pixel	R	R	R	G	G	G	B	B

5.3 Data Bus Arrangement for Color Data

As define the color depth by using different RGB arrangement on color data - 18-bits, 16-bits, 12-bits or 8-bits per pixel. CSTN LCD drivers also provide 2 types of data bus format on parallel interface, which are 8-bits and 16-bits data bus, to write those color data into the graphic display data RAM (GDDRAM).

By using 8-bits Interface, the RGB color data should be arrange as following data format:

Bit Mode	Type	Byte	MSB								LSB	
			7	6	5	4	3	2	1	0	7	6
18-bit/pixel	3 byte/1pixel	1	R	R	R	R	R	R	*	*	*	*
		2	G	G	G	G	G	G	*	*	*	*
		3	B	B	B	B	B	B	*	*	*	*
16-bit/pixel	2 byte/1pixel	1	R	R	R	R	R	G	G	G	*	*
		2	G	G	G	B	B	B	B	B	*	*
		3	B	B	B	B	B	B	B	B	*	*
12-bit/pixel	3 byte/2pixel	1	R ₁	R ₁	R ₁	R ₁	G ₁	G ₁	G ₁	G ₁	*	*
		2	B ₁	B ₁	B ₁	B ₁	R ₂	R ₂	R ₂	R ₂	*	*
		3	G ₂	G ₂	G ₂	G ₂	B ₂	B ₂	B ₂	B ₂	*	*
8-bit/pixel	1 byte/1pixel	1	R	R	R	G	G	G	B	B	*	*

By using 16-bits Interface, the RGB color data should be arrange as following data format:

Bit Mode	Type	Byte	MSB																LSB	
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14
18-bit/pixel	3 word/2pixel	1	R ₁	R ₁	R ₁	R ₁	R ₁	R ₁	*	*	G ₁	G ₁	G ₁	G ₁	G ₁	G ₁	G ₁	*	*	
		2	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	*	*	R ₂	R ₂	R ₂	R ₂	R ₂	R ₂	R ₂	*	*	
		3	G ₂	G ₂	G ₂	G ₂	G ₂	G ₂	*	*	B ₂	B ₂	B ₂	B ₂	B ₂	B ₂	B ₂	*	*	
	2 word/1pixel	1	R	R	R	R	R	R	*	*	G	G	G	G	G	G	G	*	*	
		2	*	*	*	*	*	*	*	*	B	B	B	B	B	B	B	*	*	
	2 word/1pixel	1	R	R	R	R	R	R	*	*	G	G	G	G	G	G	G	*	*	
2		B	B	B	B	B	B	*	*	*	*	*	*	*	*	*	*	*		
16-bit/pixel	1 word/1pixel	1	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	*	*	
		2	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	*	*	
12-bit/pixel	1 word/1pixel	1	R	R	R	R	G	G	G	G	B	B	B	B	*	*	*	*	*	*
		2	B	B	B	B	B	B	B	B	B	B	B	B	*	*	*	*	*	*
8-bit/pixel	1 word/1pixel	1	*	*	*	*	*	*	*	*	R	R	R	G	G	G	B	B	*	*
		2	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

** Remark:

- * is represented a don't care bit
- In 3 word/2pixel in 16-bits interface to sent out 18-bits/pixel data, it must sent out 2pixel data every time, and it must start from even no. of column (SEG0, SEG2, SEG4....and so on).
- For choosing of 2 word/pixel in 16-bits interface to sent out 18-bits/pixel data, it is only different on blue data sent from MSB part or LSB part, and it is no another different and advantage between using two mode, it is up to customer to select the blue data sent out from MSB part or LSB part.

5.4 Color Look Up Table (Color LUT)

To enable the use of Color LUT, it is simply to use the command “Set data output scan direction (BC Hex)” (shown as below), the bit named as P32 is used to select the write mode by direct write or using gamma correction via Color LUT, by default, the color data will send out by direct write mode.

	MSB							LSB		Remark
(D/C = 0)	1	0	1	1	1	1	0	0	BC Hex	
(D/C = 1)	*	*	*	*	*		P12	P11	P10	Normal or Reverse page/column/scan directions
(D/C = 1)	*	*	*	*	*		P22	P21	P20	RGB color arrangement
(D/C = 1)	*	*	*		P34	P33	P32	P31	P30	Gray scale selection.

To set color content on the Color LUT for different pixel mode, the command of “Set Color Look Up Table (CE Hex)” is used, maximum 32 byte of color content will set on Color LUT 6-bits entry follows the CE Hex command. The number of 6-bits entries is depends on different pixel mode,

For 262K and 65K color, the Color LUT will provide 32 entries to set the color content.

For 4096 color, the Color LUT will provide 16 entries to set the color content.

For 256 color, the Color LUT will provide 10 entries to set the color content.

	MSB							LSB		Remark
(D/C = 0)	1	1	0	0	1	1	1	0	CE Hex	
(D/C = 1)	*	*	X5 ₁	X4 ₁	X3 ₁	X2 ₁	X1 ₁	X0 ₁	Entry 1	
(D/C = 1)	*	*	X5 ₂	X4 ₂	X3 ₂	X2 ₂	X1 ₂	X0 ₂	Entry 2	
:	:	:	:	:	:	:	:	:	:	
:	:	:	:	:	:	:	:	:	:	
:	:	:	:	:	:	:	:	:	:	
:	:	:	:	:	:	:	:	:	:	
(D/C = 1)	*	*	X5 _{N-1}	X4 _{N-1}	X3 _{N-1}	X2 _{N-1}	X1 _{N-1}	X0 _{N-1}	Entry N-1	
(D/C = 1)	*	*	X5 _N	X4 _N	X3 _N	X2 _N	X1 _N	X0 _N	Entry N	

Color	Pixel Mode	No. of entries (N)
262K color	18bits / pixel	N = 32
65K color	16bits / pixel	N = 32
4096 color	12bits / pixel	N = 16
256 color	8bits / pixel	N = 10

5.5 Concept of the Color LUT Content

There are 3 Color LUT for Red, Green and Blue color, each Color LUT has maximum 32 entries for color content setting, each entry will assign an 5-bits address ($A_4 A_3 A_2 A_1 A_0$) for color mapping and contain 6-bits color content ($X_5 X_4 X_3 X_2 X_1 X_0$) when Color LUT is used.

Pixel Mode	No. of entries (N)	Start LUT Address	End LUT Address
18bits / pixel	N = 32	0 ($A_4 A_3 A_2 A_1 A_0 = 00000b$)	31 ($A_4 A_3 A_2 A_1 A_0 = 11111b$)
16bits / pixel	N = 32	0 ($A_4 A_3 A_2 A_1 A_0 = 00000b$)	31 ($A_4 A_3 A_2 A_1 A_0 = 11111b$)
12bits / pixel	N = 16	0 ($A_4 A_3 A_2 A_1 A_0 = 00000b$)	15 ($A_4 A_3 A_2 A_1 A_0 = 01111b$)
8bits / pixel	N = 10	0 ($A_4 A_3 A_2 A_1 A_0 = 00000b$)	9 ($A_4 A_3 A_2 A_1 A_0 = 01001b$)

For 18bits per pixel mode, there are totally 32 entries for this mode. 6-bits data for each red or green or blue color when using 262K color mode. The 6 bits will define as $C_5 C_4 C_3 C_2 C_1 C_0$, these 6-bits data will be retrieved by two look up method via mapping or calculated Color LUT address.

(a) By mapping Color LUT address:

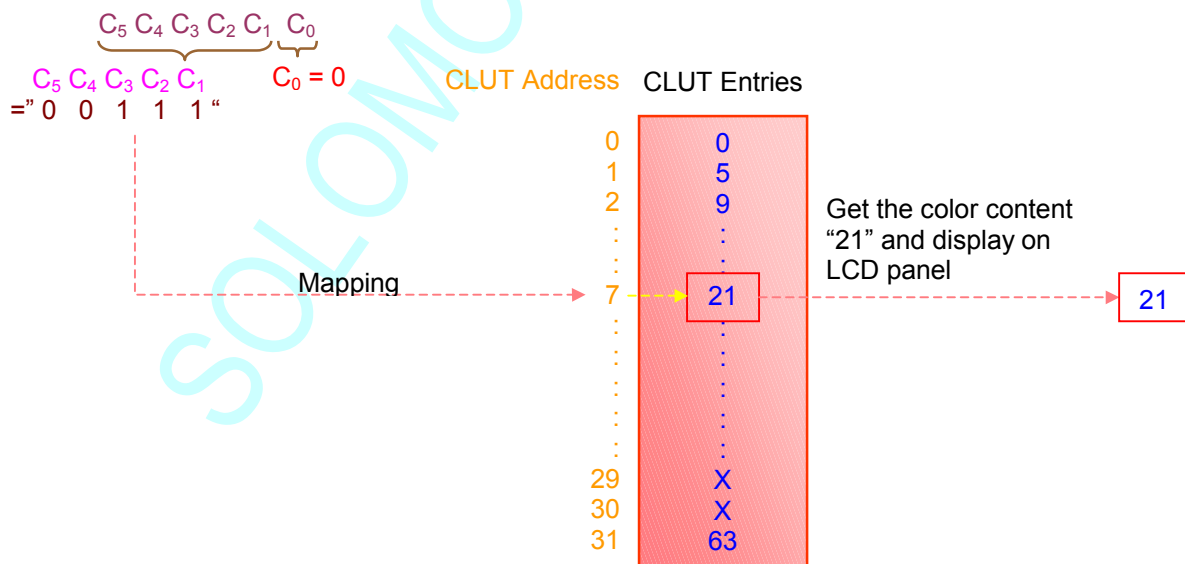
The bits $C_5 \sim C_1$ will direct mapped to the 5-bits Color LUT address and get the mapped color content set on Color LUT address when $C_0 = 0$.

Location of color content when $C_0 = 0$ is $C_5 \sim C_1 = \text{Color LUT address} [\text{color content}]$

Example: Assume the 8th entry of Color LUT (Color LUT address = 7 = 00111b) is 21 as color content ($X_{15} X_{14} X_{13} X_{12} X_{11} X_{10} = **010101b$) by using Set Color Look Up Table (CE Hex) command.

Now there is a color of Red color input as 001110 (6-bits), thus the $C_5 C_4 C_3 C_2 C_1$ should be "00111" and C_0 is "0".

Then, the $C_5 C_4 C_3 C_2 C_1$ of "00111" is map to Color LUT address = 00111b and get the color content on 8th entry, which is 21 (**010101b) to display on LCD panel.



(b) By calculating Color LUT address:

The bits $C_5 \sim C_1$ will look for the 5-bits Color LUT address to get the first color content set on Color LUT and also get the second color content set on next Color LUT entry when $C_0 = 1$. The final color content if $C_0 = 1$ is average of the two color content.

As $C_5 \sim C_1 = \text{Color LUT address}$, the final color content if $C_0 = 1$ is

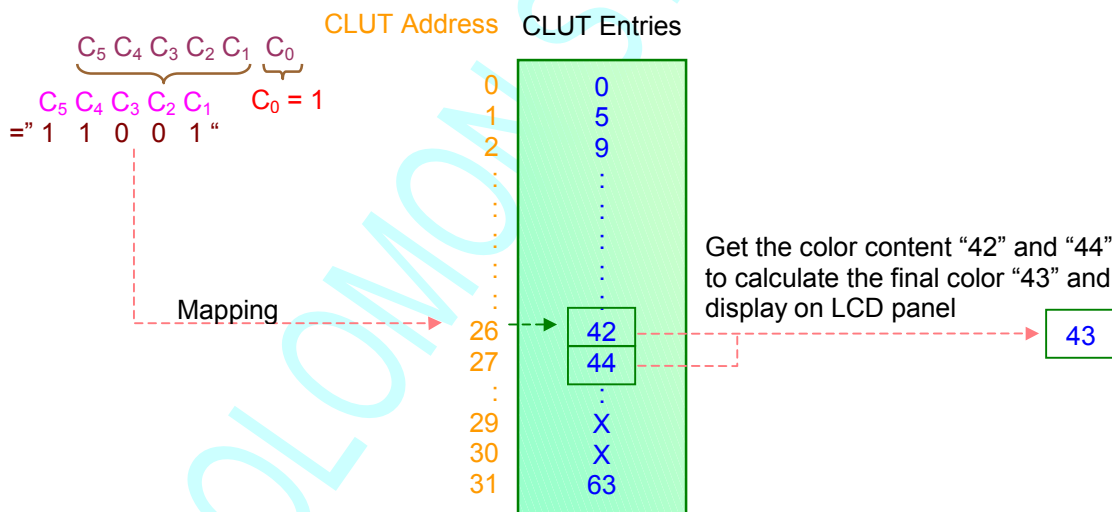
$$\frac{\text{Color LUT address [color content]} + \text{Color LUT address} + 1 [\text{color content}]}{2}$$

Example: Assume the 26th entry of Color LUT with Color LUT address = 25 = 11001b is 42 as color content ($X_{5_{25}} X_{4_{25}} X_{3_{25}} X_{2_{25}} X_{1_{25}} X_{0_{25}} = **101010b$) and the next entry of Color LUT, the 27th entry, with Color LUT address = 26 = 11010b is 44 as color content ($X_{5_{26}} X_{4_{26}} X_{3_{26}} X_{2_{26}} X_{1_{26}} X_{0_{26}} = **101100b$) by using Set Color Look Up Table (CE Hex) command.

Now there is a color of Green color input as 110011 (6-bits), thus the $C_5 C_4 C_3 C_2 C_1$ should be "11001" and C_0 is "1".

The $C_5 C_4 C_3 C_2 C_1$ of "11001" will map to the Color LUT address = 11001b, to retrieve the first color content on 26th entry of 42 (**101010b) and the second color content on 27th entry of 44 (**101100b).

By calculation, the final color content should be $\frac{42 + 44}{2} = 43$, and this value of 43 will display on LCD panel.



For the case of $C_5 C_4 C_3 C_2 C_1 =$ the last Color LUT address with $C_0 = 1$, the first and the second Color LUT address will map to the same location in the last entry.

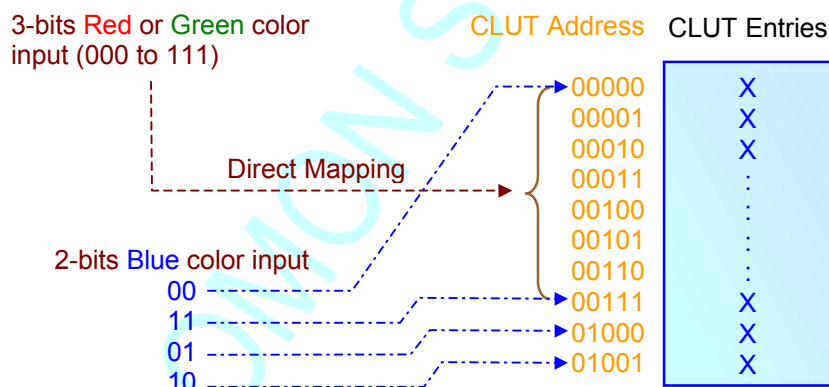
For 16bits per pixel mode, there are totally 32 entries for this mode. 5-bits data for red or blue color and 6-bits data for green color when using 65K color mode. The 5-bits red or blue color will direct mapped to Color LUT address and lookup a 6-bits color content set on Color LUT. The 6-bits Green color will define as $C_5 C_4 C_3 C_2 C_1 C_0$, these 6-bits data will be retrieved by two look up method via mapping or calculated Color LUT address similar with 262K color mode.

For 12bits per pixel mode, there are totally 16 entries for this mode. 4-bits data for red or green or blue color when using 4K color mode. The 4-bits red / green / blue color will direct mapped to Color LUT address as the address start from 00000b to 01111b, and get the 6-bits color content set on Color LUT.

For the 8bits per pixel mode, there are totally 10 entries for this 256 color mode, the first 8 entries are used for red or green color look up since there are 3-bits data for red or green color, while 2-bits data for blue color, the next 2 entries are used to set the gray of blue color.

The 3-bits red or green color will define as $C_2 C_1 C_0$, these 3-bits data will direct mapped to first 8 entries with Color LUT address from 00000b to 00111b and lookup a 6-bits color content set on Color LUT. The 2-bits blue color, it will reference to 1st, 8th, 9th, 10th entries with Color LUT address 00000b, 00111b, 01000b, 01001b.

Blue Color Data Input	Mapped to Entry	Color LUT address
00	1 st Entry	00000b
01	9 th Entry	01000b
10	10 th Entry	01001b
11	8 th Entry	00111b



6 APPLICATION NOTE IV - Programming Note

6.1 Prototypes Examples

Assume all data and command are sent out from MCU to LCD driver according to below routine by using SSD1783.

```
void output(int address, int data1)
{
    dataport = data1;           // "data1" is the data / command to be sent out
    PORT_SELECT = 0xFF;
    PORT_SELECT = address;     // "address" is a data bus port address
    PORT_SELECT = 0xFF;
}
```

6.1.1 Send Command

```
void comm_out(unsigned char i)
{
    output(PORT_B, i);
    output(PORT_A, 0x1E);      // E:0 D/C:0 RES:1 CS:0
    output(PORT_A, 0x3E);      // E:1 D/C:0 RES:1 CS:0
    output(PORT_A, 0x1E);      // E:0 D/C:0 RES:1 CS:0
    output(PORT_A, 0xDF);      // E:0 D/C:1 RES:1 CS:1
    output(PORT_B, 0xFF);
}
```

6.1.2 Send Data

```
void data_out(unsigned char i)
{
    output(PORT_B, i);
    output(PORT_A, 0x5E);      // E:0 D/C:0 RES:1 CS:0
    output(PORT_A, 0x7E);      // E:1 D/C:0 RES:1 CS:0
    output(PORT_A, 0x5E);      // E:0 D/C:0 RES:1 CS:0
    output(PORT_A, 0xDF);      // E:0 D/C:1 RES:1 CS:1
    output(PORT_B, 0xFF);
}
```

6.1.3 Show Pattern

```
void load_pattern(s_col,s_page,e_col,e_page,unsigned char code *pattern)
{
    int i,j,total_col,total_page;
    unsigned char code *pPattern=pattern;

    total_col = e_col-s_col+1;
    total_page = e_page-s_page+1;

    if(BitMode==18) // For 18-bits pattern
    {
        comm_out(0xBC);
        data_out(0x02);
        data_out(0x01);
        data_out(0x07); //set for 18-bit/pixel mode

        comm_out(0x15); // Set col address
        data_out(s_col); // Start col address
        data_out(e_col); // End col address

        comm_out(0x75); // Set page address
        data_out(s_page); // Start page address
        data_out(e_page); // End page address
        comm_out(0x5c); // Write Display Data
        for(i=0;i<total_page;i++)
        {
            for(j=0;j<total_col;j++)
            {
                data_out(*pPattern); //3bytes/1 pixel
                data_out(++pPattern);
                data_out(++pPattern);
                ++pPattern;
            }
        }
    }
    else if(BitMode==16) // For 16-bits pattern
    {
        comm_out(0xBC);
        data_out(0x02);
        data_out(0x01);
        data_out(0x04); //set for 16-bit/pixel mode

        comm_out(0x15); // Set col address
        data_out(s_col); // Start col address
        data_out(e_col); // End col address

        comm_out(0x75); // Set page address
        data_out(s_page); // Start page address
        data_out(e_page); // End page address

        comm_out(0x5c); // Write Display Data
        for(i=0;i<total_page;i++)
        {
            for(j=0;j<total_col;j++)
            {
                data_out(*pPattern); //2bytes/1 pixel
                data_out(++pPattern);
                ++pPattern;
            }
        }
    }
}
```

```

else if(BitMode==12) // For 12-bits pattern
{
    comm_out(0xBC);
    data_out(0x02);
    data_out(0x01);
    data_out(0x06); //set for 12-bit/pixel mode

    comm_out(0x15); // Set col address
    data_out(s_col); // Start col address
    data_out(e_col); // End col address

    comm_out(0x75); // Set page address
    data_out(s_page); // Start page address
    data_out(e_page); // End page address

    comm_out(0x5c); // Write Display Data
    for(i=0;i<total_page;i++)
    {
        for(j=0;j<total_col;j++)
        {
            data_out(*pPattern); //3bytes/2pixel
            data_out(++pPattern);
            data_out(++pPattern);
            ++pPattern;
        }
    }
}

else if (BitMode==8) // For 8-bits pattern
{
    comm_out(0xBC);
    data_out(0x02);
    data_out(0x01);
    data_out(0x05); //set for 8-bit/pixel mode

    comm_out(0x15); // Set col address
    data_out(s_col); // Start col address
    data_out(e_col); // End col address

    comm_out(0x75); // Set page address
    data_out(s_page); // Start page address
    data_out(e_page); // End page address
    comm_out(0x5c); // Write Display Data
    for(i=0;i<total_page;i++)
    {
        for(j=0;j<total_col;j++)
        {
            data_out(*pPattern); //1bytes/1pixel
            ++pPattern;
        }
    }
}
}

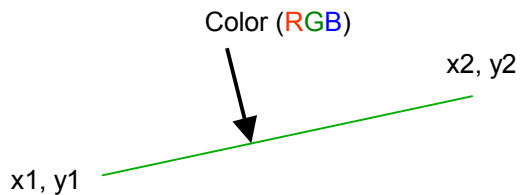
```

6.2 Graphic Command Examples

6.2.1 Fill Color

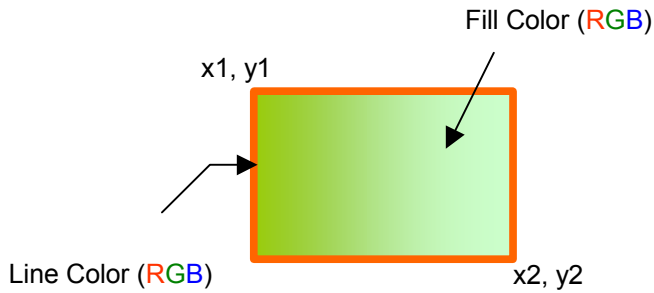
```
void fill(unsigned char Mode)
{
  switch(Mode)
  {
    case 0:      comm_out(0x92);           // Fill command
                 data_out(0x00);         // no fill, only border
                 break;
    case 1:      comm_out(0x92);           // Fill command
                 data_out(0x01);         // enable plain fill color
                 break;
  }
}
```

6.2.2 Draw Line



```
void draw_line(x1,y1,x2,y2, int Color1,int Color2)
{
  comm_out(83);           // draw line
  data_out(x1);           // start x-coordinate
  data_out(y1);           // start y-coordinate
  data_out(x2);           // end x-coordinate
  data_out(y2);           // end y-coordinate
  data_out(Color1);       // line color arrangement of Red and Green
  data_out(Color2);       // line color arrangement of Blue
}
```

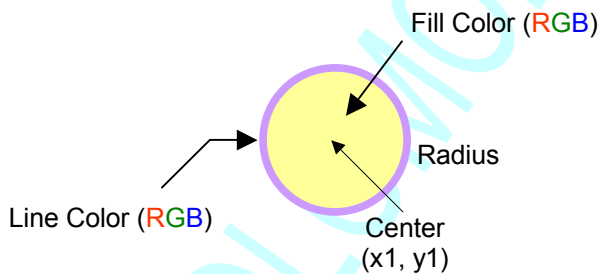
6.2.3 Draw Rectangle



```
void draw_rect(x1,y1,x2,y2,int line_byte_A,int line_byte_B,int fill_byte_A,int fill_byte_B)
{
    comm_out(0x92);           // fill enable
    data_out(0x01);

    comm_out(0x84);           // draw rectangle
    data_out(x1);             // start x-coordinate
    data_out(y1);             // start y-coordinate
    data_out(x2);             // end x-coordinate
    data_out(y2);             // end y-coordinate
    data_out(line_byte_A);    // line color arrangement
    data_out(line_byte_B);    // line color arrangement
    data_out(fill_byte_A);    // fill color arrangement
    data_out(fill_byte_B);    // fill color arrangement
}
```

6.2.4 Draw Circle (SSD1781, SSD1783 & SSD1789 only)



```
void draw_circle(x1,y1,radius,int line_byte_A,int line_byte_B,int fill_byte_A,int fill_byte_B)
{
    comm_out(0x92);           // fill enable
    data_out(0x01);

    comm_out(0x86);           // draw circle
    data_out(x1);             // center x-coordinate
    data_out(y1);             // center y-coordinate
    data_out(radius);         // radius
    data_out(line_byte_A);    // line color arrangement
    data_out(line_byte_B);    // line color arrangement
    data_out(fill_byte_A);    // fill color arrangement
    data_out(fill_byte_B);    // fill color arrangement
}
```

6.2.5 Copy Region

```
void copy_region(x1,y1,x2,y2,z1,z2)
{
  comm_out(0x8A);           // copy
  data_out(x1);             // start x-coordinate
  data_out(y1);             // start y-coordinate
  data_out(x2);             // end x-coordinate
  data_out(y2);             // end y-coordinate
  data_out(z1);             // new x-coordinate
  data_out(z2);             // new y-coordinate
}
```

6.2.6 Clear Region

```
void clear_region(x1,y1,x2,y2)
{
  comm_out(0x8E);           // clear
  data_out(x1);             // start x-coordinate
  data_out(y1);             // start y-coordinate
  data_out(x2);             // end x-coordinate
  data_out(y2);             // end y-coordinate
}
```

6.2.7 Dim Window

```
void dim_demo(x1,y1,x2,y2)
{
  comm_out(0x8c);           // dim area of (x1,y1) to (x2,y2)
  data_out(x1);             // start x-coordinate
  data_out(y1);             // start y-coordinate
  data_out(x2);             // end x-coordinate
  data_out(y2);             // end y-coordinate
}
```

As graphic command will occupy the driver during process the graphic update, and no data or command can send into the driver, BUSY pin will indicates the occupy status to let the MCU knows when can resume to send data/command into the driver.

On the other hand, there is an alternative way to applications which do not check the BUSY pin status. After the graphic command was sent, a waiting time is required to wait before send out the next data/command. This waiting time is used for update GDDRAM content to complete the graphic command cycle. To calculate the waiting time, it is depends on number of pixel needs to update the GDDRAM content, below is a reference waiting time.

For SSD1770: When $2.4 \leq VDD \leq 3.6V$, waiting time = 125 ns/pixel

For SSD1780/1788: When $1.8 \leq VDD < 2.6V$, waiting time = 250ns/pixel
When $2.6 \leq VDD \leq 3.6V$, waiting time = 125ns/pixel

For SSD1781/1783/1789: When $2.4 \leq VDD < 3.0V$, waiting time = 340ns/pixel
When $3.0 \leq VDD \leq 3.6V$, waiting time = 270ns/pixel

6.3 Graphic Command Application Examples

6.3.1 Watch the time

```
void clock_circle()
{
    int i,r,d,sec;

    fill(1); // clock outline and color
    draw_circle(64,64,50,0xf8,0x00,0xFF,0xC0); // (64,64) r=50 with light yellow background color filled
    delay(wait);
    fill(0); // pointer routine
    draw_circle(64,64,40,0xf8,0x00,0xF8,0x00); // (64,64) r=40 without fill - line red and light yellow fill

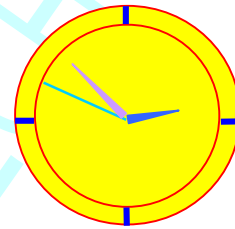
    draw_line(64,15,64,24,0x00,0x1F); // 3,6,9,12 o'clock marking with blue color
    draw_line(114,64,105,64,0x00,0x1F);
    draw_line(64,114,64,105,0x00,0x1F);
    draw_line(15,64,24,64,0x00,0x1F);
    delay(wait*20);

    draw_line(65,63,91,60,0x00,0x1F); // Hour pointer
    draw_line(65,64,91,60,0x00,0x1F);
    draw_line(66,65,88,61,0x00,0x1F);
    delay(wait*10);

    draw_line(57,28,63,64,0xF8,0xF8); // Minute pointer
    draw_line(57,28,64,64,0xF8,0xF8);
    draw_line(58,33,65,64,0xF8,0xF8);

    draw_line(58,32,66,64,0xF8,0xF8);
    delay(wait*10);

    r=36; // second pointer moving
    d=3-2*r;
    for(sec=0;sec<r;sec+=2) // r-5
    {
        draw_line(64,64,64-r,64-sec,0x0F,0x0F);
        delay(wait*20); // assume wait = 1sec
        draw_line(64,64,64-r,64-sec,0xFF,0xC0);
        for(i=0;i<2;i++)
        {
            if (d<0)
                d+=4*sec+6;
            else
            {
                d+=4*(sec-r)+10;
                r--;
            }
        }
    }
}
```



6.3.2 Play a game

```

void Bubbles_circle()
{
    int i;

    fill(1);
    draw_rect(0,0,127,159,0xF7,0x80,0xF7,0x80);           //line & fill brown
    delay(wait);
    draw_rect(10,10,117,149,0x00,0x00,0x00,0x00);         //line & fill Black
    delay(wait*10);

    fill(0);
    draw_circle(63,149,15,0xC0,0x18,0xC0,0x18);           // half circle with line purple
    fill(1);
    draw_rect(10,150,117,159,0xF7,0x00,0xF7,0x00);       // cover below half circle with line & fill brown
    delay(wait);
    draw_line(63,125,63,149,0xFF,0xC0);                   // pointer with line yellow
    draw_line(63,125,60,129,0xFF,0xC0);                   // pointer with line yellow
    draw_line(63,125,66,129,0xFF,0xC0);                   // pointer with line yellow
    delay(wait*10);

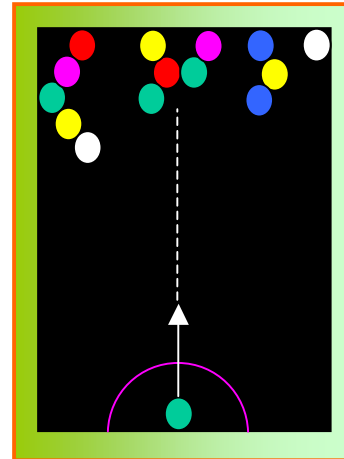
    fill(1); //bubbles
    draw_circle(33,21,5,0xF8,0x00,0xF8,0x00);             delay(wait*5);
    draw_circle(57,21,5,0xFF,0xC0,0xFF,0xC0);             delay(wait*5);
    draw_circle(81,21,5,0x00,0x1F,0x00,0x1F);             delay(wait*5);
    draw_circle(93,21,5,0x07,0xE0,0x07,0xE0);             delay(wait*5);
    draw_circle(105,21,5,0xF8,0xF8,0xF8,0xF8);            delay(wait*5);
    draw_circle(27,30,5,0x00,0x1F,0x00,0x1F);             delay(wait*5);
    draw_circle(63,30,5,0x00,0x1F,0x00,0x1F);             delay(wait*5);
    draw_circle(87,30,5,0xF8,0x00,0xF8,0x00);            delay(wait*5);
    draw_circle(99,30,5,0x07,0xE0,0x07,0xE0);            delay(wait*5);
    draw_circle(21,39,5,0xF8,0xF8,0xF8,0xF8);            delay(wait*5);
    draw_circle(69,39,5,0x07,0xE0,0x07,0xE0);            delay(wait*5);
    draw_circle(81,39,5,0xFF,0xC0,0xFF,0xC0);            delay(wait*5);
    draw_circle(27,48,5,0x07,0xE0,0x07,0xE0);            delay(wait*5);
    draw_circle(39,48,5,0xF8,0xF8,0xF8,0xF8);            delay(wait*5);
    draw_circle(63,48,5,0xFF,0xC0,0xFF,0xC0);            delay(wait*5);
    draw_circle(87,48,5,0xFF,0xC0,0xFF,0xC0);            delay(wait*5);
    draw_circle(57,57,5,0x00,0x1F,0x00,0x1F);             delay(wait*5);
    draw_circle(69,57,5,0x00,0x1F,0x00,0x1F);             delay(wait*20);

    for(i=0;i<=50;i+=10)                                   //dotted line
    {
        draw_line(63,123-i,63,118-i,0xFF,0xFF);          //white
        delay(wait*2);
    }

    copy_region(58,139,69,144,0,160);
    copy_region(58,144,69,149,11,160);
    draw_circle(63,144,5,0x00,0x1F,0x00,0x1F);           delay(wait*10);

    for(i=1;i<79;i+=2)
    {
        copy_region(0,160,10,165,58,140-i);
        copy_region(11,160,21,165,58,145-i);
        copy_region(58,138-i,69,144-i,0,160);
        copy_region(58,143-i,69,149-i,11,160);
        draw_circle(63,143-i,5,0x00,0x1F,0x00,0x1F);     delay(wait*2);
    }
    draw_circle(57,57,5,0x00,0x00,0x00,0x00);
    draw_circle(69,57,5,0x00,0x00,0x00,0x00);
    draw_circle(63,66,5,0x00,0x00,0x00,0x00); delay(wait*10);
}

```



6.3.3 Have a good shot

```

void Snooker_demo()
{
    int i;

    fill(1);
    draw_rect(10,10,117,149,0xFB,0xE0,0xFB,0xE0);           //table
    delay(wait);
    draw_rect(15,15,112,144,0xFB,0xE0,0x07,0xE0);
    delay(wait);
    draw_circle(15,15,7,0x00,0x00,0x00,0x00);               // 6 pockets
    draw_circle(112,15,7,0x00,0x00,0x00,0x00);
    draw_circle(15,144,7,0x00,0x00,0x00,0x00);
    draw_circle(112,144,7,0x00,0x00,0x00,0x00);
    draw_circle(15,79,7,0x00,0x00,0x00,0x00);
    draw_circle(112,79,7,0x00,0x00,0x00,0x00);

    fill(1);
    for(i=0;i<48;i+=12)                                     // 4 red balls on first line
        draw_circle(46+i,28,5,0xF8,0x00,0xF8,0x00);
    for(i=0;i<36;i+=12)                                     // 3 red balls on second line
        draw_circle(52+i,38,5,0xF8,0x00,0xF8,0x00);
    for(i=0;i<24;i+=12)                                     // 2 red balls on third line
        draw_circle(58+i,48,5,0xF8,0x00,0xF8,0x00);
    draw_circle(64,58,5,0xF8,0x00,0xF8,0x00);              // 1 red ball on bottom line
    delay(wait*20);
    draw_circle(64,129,5,0xFF,0xFF,0xFF,0xFF);             // white ball
    delay(wait*30);

    draw_rect(63,147,65,149,0xFF,0xFF,0xFF,0xFF);         // sticker moving - line & fill = white
    for(i=1;i<11;i++)
    {
        draw_rect(63,147-i,65,149-i,0xFF,0xFF,0xFF,0xFF); // line & fill = white
        delay(wait);
        draw_rect(63,149-i,65,149,0x00,0x00,0x00,0x00);  // line & fill = black
        delay(wait);
    }

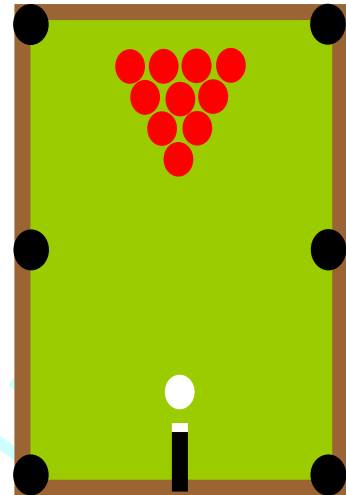
    for(i=0;i<59;i++)
    {
        copy_region(58,124-i,69,135,58,123-i);           // white ball moving
        delay(wait);
    }

    copy_region(25,22,91,64,25,21);                       // all balls move up 2 pixels
    copy_region(25,21,91,63,25,20);
    delay(wait);

    fill(1);
    for(i=1;i<6;i++)
    {
        draw_circle(48-i*2,28-i,5,0x07,0xE0,0x07,0xE0);  // top left ball clear
        draw_circle(80+i*2,28-i,5,0x07,0xE0,0x07,0xE0); // top right ball clear
        draw_circle(58,28-i,5,0x07,0xE0,0x07,0xE0);     // middle 1 ball clear
        draw_circle(70,28-i,5,0x07,0xE0,0x07,0xE0);     // middle 2 ball clear

        draw_circle(46-i*2,27-i,5,0xF8,0x00,0xF8,0x00); // top left ball move
        draw_circle(82+i*2,27-i,5,0xF8,0x00,0xF8,0x00); // top right ball move
        draw_circle(58,27-i,5,0xF8,0x00,0xF8,0x00);     // middle 1 ball move
        draw_circle(70,27-i,5,0xF8,0x00,0xF8,0x00);     // middle 2 ball move
        delay(wait);
    }
    for(i=1;i<4;i++)
    {
        draw_circle(39-i*3,21+i,5,0x07,0xE0,0x07,0xE0); // top left ball clear
        draw_circle(89+i*3,21+i,5,0x07,0xE0,0x07,0xE0); // top right ball clear
    }
}

```



```
draw_circle(58,21+i,5,0x07,0xE0,0x07,0xE0); // middle 1 ball clear
draw_circle(70,21+i,5,0x07,0xE0,0x07,0xE0); // middle 2 ball clear

draw_circle(36-i*3,22+i,5,0xF8,0x00,0xF8,0x00); // top left ball move
draw_circle(92+i*3,22+i,5,0xF8,0x00,0xF8,0x00); // top right ball move
draw_circle(58,22+i,5,0xF8,0x00,0xF8,0x00); // middle 1 ball move
draw_circle(70,22+i,5,0xF8,0x00,0xF8,0x00); // middle 2 ball move
delay(wait*3);
}
draw_circle(101,25,5,0x07,0xE0,0x07,0xE0); // top right ball clear
}
```

SOLOMON SYSTECH LTD.

SOLOMON SYSTECH LTD.

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