深圳市拓普微科技开发有限公司
SHENZHEN TOPWAY TECHNOLOGY CO., LTD.

TCB8000 series

TFT Controller Board
Application Note

Rev.	Descriptions	Release Date
0.1	New release	2009-03-04
0.2	Add: Alpha Numeric Char Size Note in “6.8 Font Draw”	2009-04-14
0.3	Add: MC68000(16bit) I/F support, fast READ option	2011-06-15

Table of Content

1. Introduction

This Application Note describes the application example of using TCB8000 series TFT Controller board. TCB8000 series TFT Controller board is using T8000 TFT LCD controller which is a high performance graphics controller supporting mono LCD display to TFT color LCD display. It embed with 2D geometry drawing engine (line draw, circle draw, etc.) as well as font handling engine data (ASCII, Big5, GB2312).

2. Product Series

The followings are the main TCB8000 series product and their descriptions

Controller Board Model	Major Target LCM	Host Terminals				Note
		8bit data 1bit add	8bit data 18bit add	16bit data 18bit add	RS232C	
TCB8000A	LMT057DNAFWU	O	O	O	O	Built-in TFT LED back light driver Power Circuit
TCB8000C	LMT035DNAFWU	O	O	O	O	
TCB8000D	LMT057DNAFWU	O	X	X	O	
TCB8000J	LMT035DNAFWU	O	X	X	X	Built-in TFT LED back light driver Power Circuit, with extended ESD performance
TCB8000K	LMT057DNAFWU	O	X	X	O	
TCB8000E	Mono LCM	O	X	X	O	For mono LCD (4bit controller I/F) with negative booster Circuit

Some of the TOPWAY product are in kit form, TFT + controller board. Optional touch add-on is also available. Please contact our sales office for details

3. Features Highlight

- Power Supply and Logics Level
  - Single 5.0V supply voltage
  - 3.3V MCU interface / logic (tolerate to 5.0V logic input)
  - Built in DC-DC converter for LCD supply
  - Built in backlight supply, with software enable
- Host Interface (selectable)
  - 8bit data, 1bit address (indirect/command mode)
  - 8bit data, 18bit address (generic mode)
  - 16bit data, 18bit address (generic mode, little edian)
  - 16bit data, 18bit address (MC68000 mode, big edian)
  - UART serial (RS232C)
- Display Support (depends on model)
  - Up to 320x240, 64k color, TFT LCD display
  - Up to 640x240, 16 gray, Mono passive LCD display (4bit interface)
- 2D Drawing Engine
  - Line Draw
  - Circuit Draw (with or without solid fill)
  - Box Draw (with or without solid fill)
- Font Draw Engine
  - Support 8bit (ASCII) char coding & 16bit char coding (GB2312 or BIG5)
  - Support 8x8, 8x16 and 16x16 font size with color expansion
  - Built-in internal CGRAM (2k-byte), 8x8, 8x16, 16x16 font size is possible.
- Display Features
  - Virtual display with pixel panning in all color depths
  - Hardware Sprite (256x256 max)

4. Host Interfacing

There 4 kind of host interface could be use on the TCB8000 series. (not all are available, please check with the user manual for jumper setting details and availability)

Interface Mode	Interface to host			Note
	Address	Data	Control Bus	
8-bit generic (18bit add, 8bit data) (*1)	A17~A0	D7~D0	/CS, /WE0, /RD0, /WAIT	Similar to a 8-bit ram bus Full Address Range byte write. Full Address Range byte read.
16-bit generic (18bit add, 16bit data) (*3)	A17~A0	D15~D0	/CS, /WE0, /WE1, /RD0, /WAIT	Similar to a 16-bit ram bus Full Address Range 16bit or byte write Full Address Range byte read only A0=0, Read Low Byte on D[7:0] A0=1, Read High Byte on D[17:8]
16-bit generic (18bit add, 16bit data) (*3)	A17~A1	D15~D0	/LDS, /UDS R/W, /AS /DTACK	Full Address Range 16bit or byte write Full Address Range 16bit or byte read
Indirect Mode (1bit add, 8bit data) (*2)	A1	D7~D0	/CS, /WE0, /RD0	A17~A12,A2=Hi; A11~A3,A0=Lo Only “Command Packet Port” (F004, F006) could be accessed Command Packet write Status byte read only
UART Serial Mode	TXD, RXD			

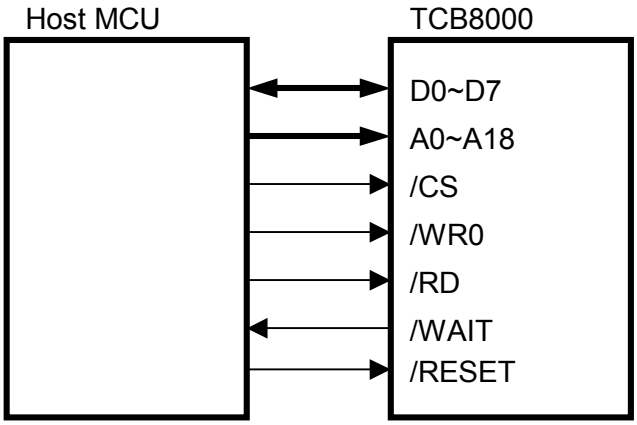
*1. Using 8-bit or 16bit generic interface, user could access the controller internal memory directly. Where there are two special address 0x2F004 and 0x2F006.
0x2F004 is for Command input
0x2F006 is for Command control

*2. Using Command Mode, is base on using the above two address only. By fixing all the address line except the A1. With the memory access commands, host could send the data to any dedicated register or memory location.

*3. Only TCB8000A (using 16bit Display RAM) support 16bit data read.

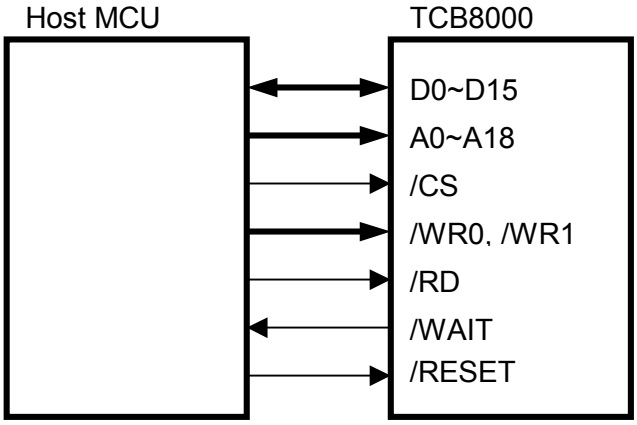
4.1 8-bit Generic mode

In this mode, host is accessing the TCB8000 as a RAM. Where the /WAIT signal may used during read. (it could leave open when not used) please refer T8000 Technical Manual for details



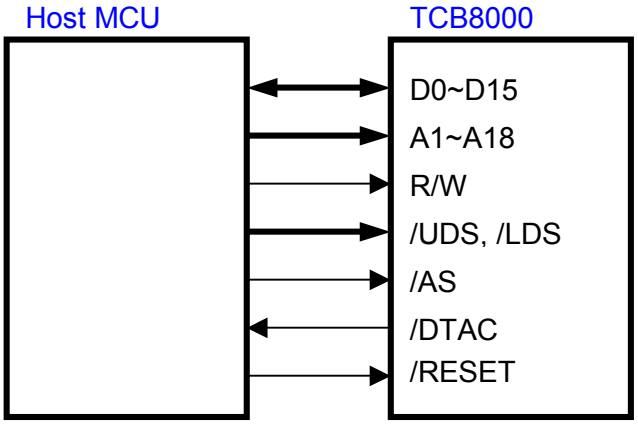
4.2 16-bit Generic Mode

In this mode, host is accessing the TCB8000 as a 16bit RAM. Where the /WAIT signal may used during read. (it could leave open when not used)
In this mode a 8-bit write Timing Sequence is <Little Edian>.
A0=1, for high byte data;
A0=0, for low byte date
please refer T8000 Technical Manual for details



4.3 16-bit 68000 Mode

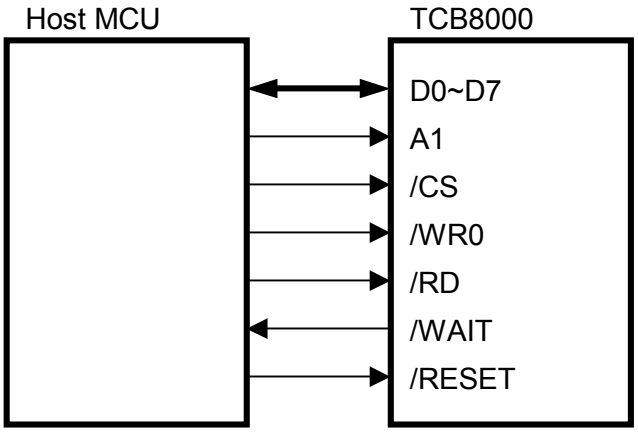
In this mode, host is accessing the TCB8000 as a 16bit RAM. Where the M68000 interface make it possible to be full access with 16bit bus.
In this mode a 8-bit write Timing Sequence is <Big Edian>.
/UDS and /LDS could work as the same for full 16bit access
please refer T8000 Technical Manual for details



4.4 Indirect Mode

This is the simplest connection for a MCU Host. It uses 1 bit of address, where all the command and data are sending to the command port only. (some times we called it command mode)
By controlling A1, the command address (0x2F004 , 0x2F006) could be access.
In this mode, display data could be input via the memory access command (write only).

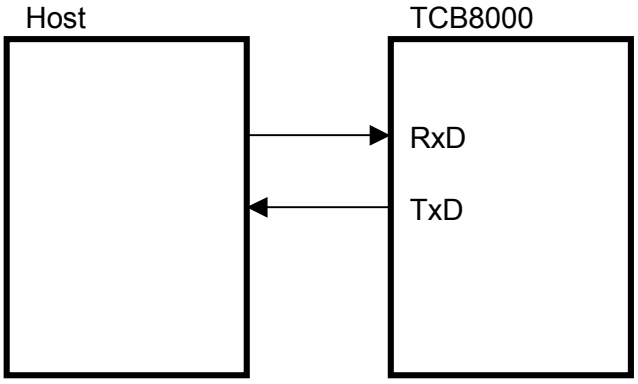
Please see “Command Packet Format” for using the command.



4.5 UART serial mode(RS232C)

TCB8000 equipped RS232C interface signal conversion IC for receiving the RS-232C signal directly, say PC.
The default (after reset) baud rate is 9600 and it could be adjusted by command.

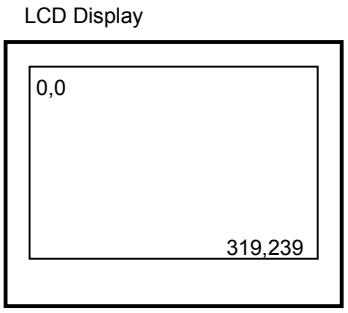
The built in power on reset will start up the TCB8000 and work with the command that received. If necessary, user could applied a reset signal via the /RESET pin of the TCB8000.



Note,
UART serial mode packet format is not the same as the Indirect Mode.

5. Display and Memory Address

The Memory Address is related to the Memory on board configuration. The following are the summary of the address.



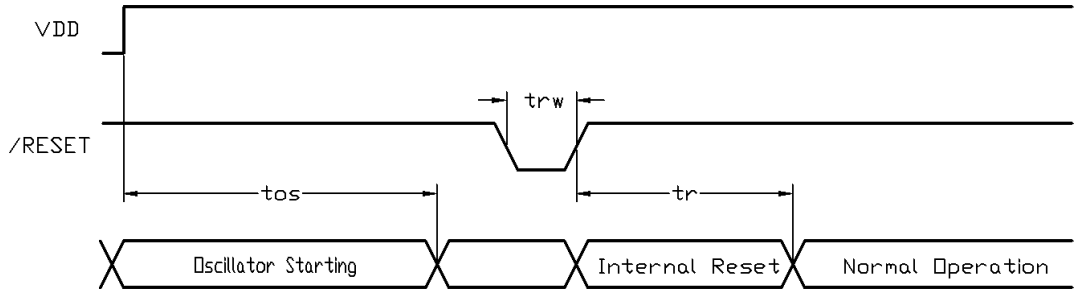
Access mode	320x240 TFT application. (56k +128k byte memory)	320x240 Mono display application (56k byte memory)
Command access	top-left : X=0, Y=0; bottom-right : x=319, y = 239	top-left : X=0, Y=0; bottom-right : x=319, y = 239
Direct memory access	Full display add (top-left) =0x00000 Sprite start add = 0x2C800 CGRAM start add = 0x2D800 Config Register = 0x2F000 ~ 0x2FFFF	Full display add (top-left) =0x00000 Sprite start add = 0x0C800 CGRAM start add = 0x0D800 Config Register = 0x0F000 ~ 0x0FFFF

Inside the display memory, the data is arranged as little endian.
When sending 16bit data into the controller board via 8bit data interface, it should be LSB(lo-byte) first, then MSB(hi-byte).

6. Function

6.1 Reset

After power on, a hardware-reset signal should apply to the TCB8000.



As T8000 internal PLL takes some time to start up, For save, it is suggested to leave some time before sending the reset plus to it.
The following this is the typical value to be used: tos=800ms, trw=1ms,tr=200ms

6.2 Display on/off

After the power on hardware reset, the on board DC-DC booster for the TFT Driver and the backlight are disabled, where these booster is controlled by the internal “display on” signal.

They could be enabled by REG[F006].3, “display-on” setting. Where, modifying the REG[F08E].1, invert the “display-on” signal polarity, could have the same result.

Note, the REG[F08E].3, blank screen, setting could show a full black screen, without disable the DC-DC circuit.

6.3 Sending Data and Command

Using TCB8000, the display data register could be accessed by two way.

1. Command Access (generic mode, indirect mode, UART serial mode)
   - use memory access command to access all the register and display data
   - use command to draw (line, circle, etc)
   - use command to show char
2. Direct Memory Access (generic mode only)
   - control the TCB8000 as a RAM, and directly access the register and display memory

6.4 Command Structure

Except in UART serial mode, the Command FIFO can store up to eight commands. Each command consists 1 byte op-code and up to 64 bytes of parameters (data). In UART serial host mode, only one command can be stored in the Command FIFO with a maximum length of 64 bytes. Furthermore, there is no read back path for the Command Interpreter, i.e. it is a write only. However in UART serial mode, register content can be read back through the “Acknowledge Packet”.

6.4.1 Command Packet Format

All commands are organized in packet with a 1 byte “Opcode” followed by optional parameters / data.

	Sequence	Add (A15~A0)	/RD	/WR	Data	Descriptions
Up to 64byte ↑ ↓	1	0xF004	1	0	Opcode	Command Opcode
	2	0xF004	1	0	Parameter	Parameter / Data
	:	:	:	:	:	:
	:	:	:	:	:	:
	:	0xF006	1	0	01(hex)	Command End, Display On

- For matching the FIFO size, the command packet should not larger than 64byte (exclude, the Command End)
- For multi-byte parameter/data, send LSB (low byte)first, MSB (highest byte) last.
- The A16, A17 value depend on Memory Configuration.

6.4.2 UART Serial Mode Packet Format

In UART serial host mode, each communication packet starts with a byte of “FF” and ended with “FE”. Length of parameters (one byte) is also required into the packet. An Acknowledge Packet will be sent back to the UART serial host by the T8000 once the command is finished execution.

UART Serial Mode Packet Format

Sequence	No of byte	Content
1	1	0xFF (hex) <START BYTE>
2	1	Opcode
3	1	Length of Parameters
4	1 to 64	Parameters / Data (up to 64bytes)
5	1	0xFE (hex) <END BYTE>

Note: A “FF” bytes sequence of length equal to or more than 65 will cause re-synchronization.

UART Serial Mode re-synchronization Packet Format

Sequence	No of byte	Content
1	Equal or more than 65	0xFF (hex)
2		0xFF (hex)
:		:
:		:

Note: A “FF” bytes sequence of length equal to or more than 65 will cause re-synchronization.

For commands required read data (Opcode 82) from the T8000, it will send read data embedded in the Acknowledge Packet automatically when data is ready.

UART Serial host Acknowledge Packet Format, without “register read data”

Sequence	No of byte	Content
1	1	0x00 (hex)

UART Serial host Acknowledge Packet Format, with “register read data”

Sequence	No of byte	Content
1	1	Register read data
2	1	0x00 (hex)

6.5 Command (Opcode) Descriptions

Opcode (hex)	Operations	No of Parameters / Data (byte)	Parameters / Data
00	Set “Control & Status Port” of the Command Interpreter	1	The value of this data will be directly written to the Control & Status register.
10	charset_config	1	00: Built in 8x8 ASCII 01: 8x8 CGRAM (Embedded RAM) 02: 8x16 CGRAM (Embedded RAM) 03: 16x16 CGRAM (Embedded RAM) 04: 16x16 GB2312-80 (External ROM) 05: 16x16 BIG5 (External ROM) 06: 8x8 Custom 8-bit encoding (External ROM) 07: 8x8 Custom 16-bit encoding (External ROM)
12	set_print_coord	4	Character Print Coordinates x-coordinate (2-bytes) y-coordinate (2-bytes) * Mono LCD, x = (multiple of 8) – 1 * Color LCD, no restriction on the value of x coordinate

Command (Opcode) Descriptions (cont')

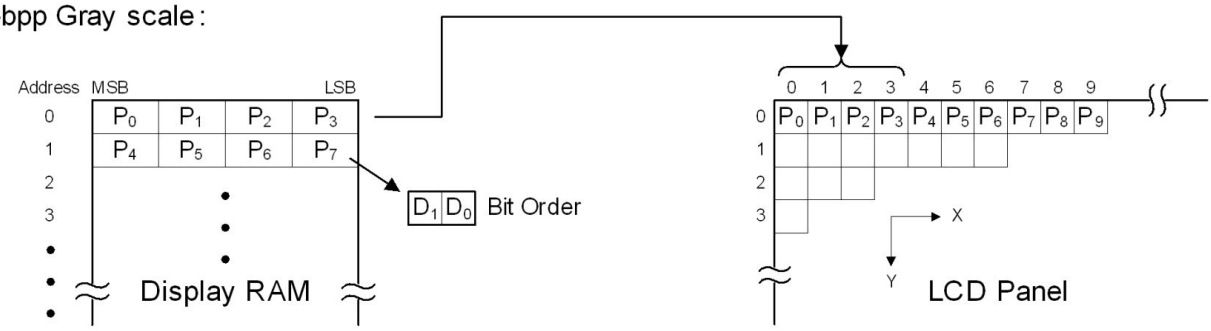
<table>
<tr><td>14</td><td>set_font_fgcolor</td><td>2</td><td>Character Foreground Color (same as td_fgcolor)
* Mono LCD: 1bpp, 2bpp, 4bpp
* Color LCD: 12-bit STN (4R:4G:4B)</td></tr>
<tr><td>15</td><td>set_font_bgcolor</td><td>2</td><td>Character Background Color
* LCD: 1bpp, 2bpp, 4bpp
* LCD: 12-bit STN (4R:4G:4B)</td></tr>
<tr><td>16</td><td>show_char</td><td>1 or 2</td><td>Display Character</td></tr>
<tr><td>17</td><td>show_string</td><td>1 + (1 to 63)</td><td>Display String
No. of characters (1-byte)
Character String (1 to 63-bytes)</td></tr>
<tr><td>19</td><td>define_CGRAM</td><td>1 + (1 to 32)</td><td>CGRAM Bitmap
Character code (1 byte)
Bitmap pattern (1 to 32 bytes)
(before “define_CGRAM “, use the “charset config” to tell what kind of CGRAM font are defining)</td></tr>
<tr><td>20</td><td>td_fgcolor</td><td>2</td><td>Set Foreground Color
*Mono LCD: 1bpp, 2bpp, 4bpp
*Color LCD: 12-bit STN (4R:4G:4B)</td></tr>
<tr><td>23</td><td>draw_pixel</td><td>4</td><td>Draw Pixel
x-coordinate (2-bytes)
y-coordinate (2-bytes)</td></tr>
<tr><td>24</td><td>draw_line</td><td>8</td><td>Draw Line
x_start (2-bytes)
y_start (2-bytes)
x_end (2-bytes)
y_end (2-bytes)</td></tr>
<tr><td>26</td><td>draw_rect</td><td>8</td><td>Draw Hollow Rectangle (Box)
x_start (2-bytes)
y_start (2-bytes)
x_end (2-bytes)
y_end (2-bytes)</td></tr>
<tr><td>27</td><td>fill_rect</td><td>8</td><td>Fill Rectangle (Box)
x_start (2-bytes)
y_start (2-bytes)
x_end (2-bytes)
y_end (2-bytes)</td></tr>
<tr><td>28</td><td>draw_circle</td><td>5</td><td>Draw Circle
x_center (2-bytes)
y_center (2-bytes)
radius (1-byte)</td></tr>
<tr><td>29</td><td>fill_circle</td><td>5</td><td>Fill Circle
x_center (2-bytes)
y_center (2-bytes)
radius (1-byte)</td></tr>
<tr><td>60</td><td>set_baud</td><td>2</td><td>Set baud rate
divisor (lower byte)
divisor (upper byte)
<table>
<tr><th>Divisor</th><th>RS232 baud rate</th></tr>
<tr><td>1047</td><td>110</td></tr>
<tr><td>24</td><td>4800</td></tr>
<tr><td>12</td><td>9600 <default></td></tr>
<tr><td>6</td><td>19200</td></tr>
<tr><td>3</td><td>38400</td></tr>
<tr><td>2</td><td>57600</td></tr>
<tr><td>1</td><td>115200</td></tr>
</table></td></tr>
<tr><td>80</td><td>refresh_setting</td><td>0</td><td>-</td></tr>
<tr><td>81</td><td>set_mem_ptr</td><td>3</td><td>Set memory pointer
address (3-bytes)</td></tr>
<tr><td>82</td><td>read_reg</td><td>2</td><td>Read register(*ONLY used in UART serial host mode)
address (2 bytes)</td></tr>
<tr><td>83</td><td>write_reg</td><td>2 + 1</td><td>Write register
address (2-bytes)
data (1-byte)</td></tr>
<tr><td>84</td><td>write_mem</td><td>1 + (1 to 63)</td><td>Write memory
No. of data (1 byte)
data (1 to 63 bytes)</td></tr>
<tr><td>8F</td><td>clk_en</td><td>6</td><td>Enable memory clock; [69, 45, 61, 67, 6C, 65 (6-bytes in hex)]</td></tr>
</table>

6.6 Pixel Data Format

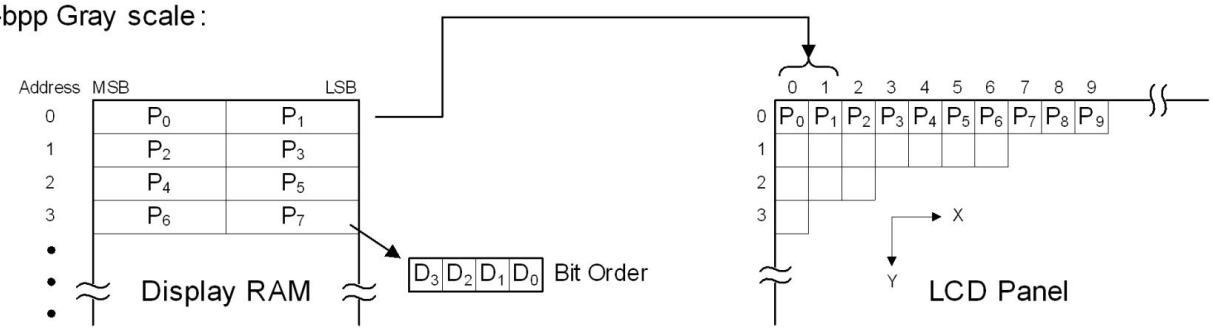
For the TFT display, select 65k color mode. It is 5:6:5 format, where first 5bit for Red, next 6bit for Green, last 5bit for blue. For mono LCD, generally, 4 gray level will be selected.

	Pixel Data Format						
Color mode	65k Color 16-bpp 5:6:5(R:G:B)		16 Gray Level 4-bpp		4 Gray Level 2-bpp		
Display Data	DB15~DB8	DB7~DB0	DB7~DB4 pixel(n)	DB3~DB0 pixel(n+1)	DB7, DB6 pixel(n)	…	DB1, DB0 pixel(n+3)
Content	R4~R0, G5~G3	G4~G0, B4~B0	D3~D0	D3~D0	D1,D0	…	D1,D0

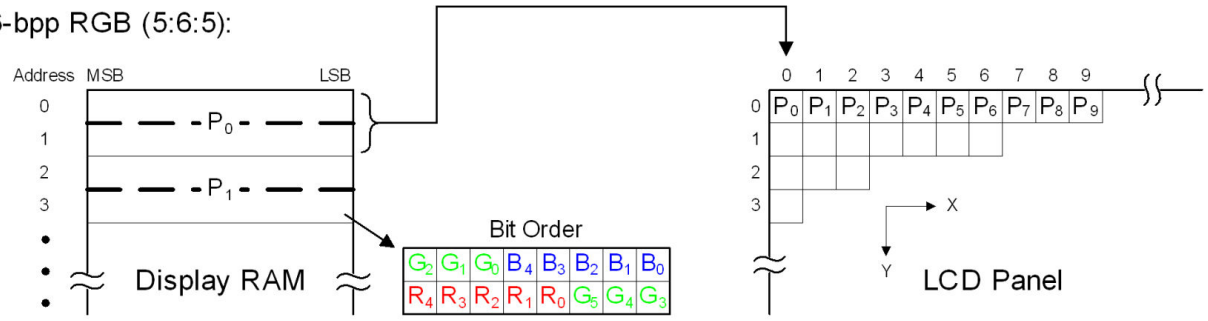
2-bpp Gray scale:



4-bpp Gray scale:



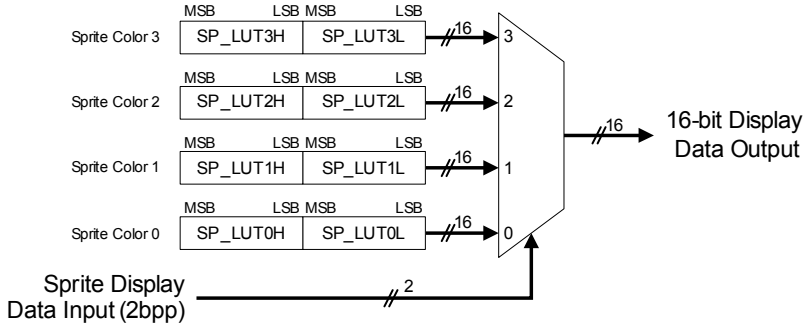
16-bpp RGB (5:6:5):



6.7 Sprite / Overlay

A small image (256x256 max.) which could overlaid on a background picture, is called “sprite”. A sprite may consist of a sequence of images. Viewers can experience animation effect if the sequence of images is loaded to screen continually. Moreover, the sprite image colors can be changed at anytime through the built-in look-up table.

A general application of a sprite could be a mouse pointer.
It contains any 4-color form the 16bit color palette by using the look up table.



Further more, REG[F100].6 could turn on the transparent color option. The color 00 in side the sprite will becomes transparent.

6.8 Geometry Draw and Font Draw

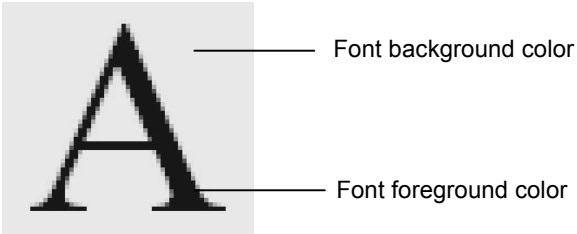
The Geometry Drawing Engine will perform line draw, circle draw, circle draw with solid fill, box draw and box draw with solid fill by using the assigned foreground color.

The Font Drawing Engine generates the char onto the display by using the input char code. Char showing on screen is based on the selected font background color and font foreground color.

The supported character formats are ASCII ,GB2312-80 and custom (inside CGRAM) font.

Input Char Code	Char Size	Note
ASCII char	8x8	Use charset_config command to select the char type and font size
GB2312-80 (half width) char	8x16	
GB2312-80 (full width) char	16x16	
User defined Char	8x8, 8x16, 16x16 (selectable)	

The Font Drawing Engine uses both the foreground color and background color.



Note.
When displaying Alpha Numeric char., size could be selected by char code and char_config.

char_config value	Font size	Alpha Numeric Char Code (example)	Display result	Note
0x00 (ASCII)	8x8	0x41, 0x42, 0x43, 0x31, 0x32, 0x33	ABC123	Standard ASCII
0x04 (GB2312-80)	8x16	0x41, 0x42, 0x43, 0x31, 0x32, 0x33	ABC123	Half-width GB char
0x04 (GB2312-80)	16x16	0xA3C1, 0xA3C2, 0xA3C3, 0xA3B1, 0xA3B2, 0xA3B3	ABC123	Full-width GB char

6.9 Using CGRAM

There is 2k-byte RAM space for user to create custom font call CGRAM (custom generated RAM). Customer could build

- 256 char in 8x8 size or
- 128 char in 8x16 size or
- 64 char in 16x16 size

Where, the font size is selected by charset_config command.

Defining a CGRAM should follow the following steps:

1> use opcode 0x10 to select the custom font size
2> use opcode 0x19 to built the font
3> use opcode 0x16 or 0x17 to draw the font.

Please see the Command (Opcode) Descriptions for operation details

The followings are the dot mapping for different font size.

8x8 font mapping

		← 8 dots →
↓ 8line →	1st line	1st byte D7….D0
	:	:
	8th line	8th byte D7….D0

8x16 font mapping

		← 8 dots →
↓16 line →	1st line	1st byte D7….D0
	:	:
	16th line	16th byte D7….D0

16x16 font mapping

		← 16 dots →	
↓ 16 line →	1st line	1st byte D7….D0	2nd byte D7….D0
	:	:	:
	16th line	31st byte D7….D0	32nd byte D7….D0

6.10 Registers Table

The register address A16,A17 are based on the controller board configuration.
(TCB8000E: A16, A17 = 0, 0; Others: A16, A17 = 1, 1)

Address (hex) A15 ~ A0	R/W	Reset Value	Descriptions
F000	Read / Write	1000 0000	Chip ID Port, Always read back 80 (hex) for T8000 Write “DE FC 0B” (hex) to enable memory clock, same as command with OPCODE “8F”.
F001	Read	0000 0000	Chip Revision Port, Always read back 00 (hex) for T8000
F004	Write	-	Command Packet Port - Writing of Command Packets.
F006	Write	xxx1 xxx0	Port for writing control or reading status Bit[7:4] : Reserved Bit[3] : DISPLAY ON / OFF 0=DISPLAY ON 1=DISPLAY OFF Bit[2:1] : Reserved Bit[0] : End of Command, Write “1” after each command packet
	Read	xxxx xxx0	Bit[7:1] : Reserved Bit[0] : FIFO full Read “1” if Command FIFO is full. Hosts must read this bit = “0” before writing to Command Packet Port.
F080	Read / Write	0000 0000	Bit[7:4] : Reserved Bit[3] : STN Panel I/F Data Width 0=4-bit single 1=8-bit single Bit[2] : Color Mode Select 0=Monochrome 1=Color Bit[1:0] : Color Depth Select If Monochrome (Bit[2]=0) 00=1 bit-per-pixel 01=2 bit-per-pixel 10=4 bit-per-pixel 11=Reserved If Color (Bit[2] = 1) 01=12 bit-per-pixel (CSTN panel) 00=Reserved 10=Reserved 11=Reserved
F081	Read / Write	x000 0000	Bit[7] : Reserved Bit[6:0] : Panel Horizontal Character Count – 1, Panel Horizontal Character Count[8:0] supports horizontal panel size up to 128 characters or 1024 pixels.
F082	Read / Write	0000 0000	Bit[7:0] : Panel Line Count - 1 bit[7:0]
F083	Read / Write	xxxx xxx0	Bit[7:1] : Reserved Bit[0] : Panel Line Count – 1 bit[8], Panel Line Count[8:0] supports vertical panel size up to 512 lines.
F084	Read / Write	0000 0000	Bit[7:0] : Display Start Position X Coordinate – bit[7:0]
F085	Read / Write	xxxx xx00	Bit[7:2] : Reserved Bit[1:0] : Display Start Position X Coordinate – bit[9:8]
F086	Read / Write	0000 0000	Bit[7:0] : Display Start Position Y Coordinate – bit[7:0]
F087	Read / Write	xxxx xx00	Bit[7:2] : Reserved Bit[1:0] : Display Start Position Y Coordinate – bit[9:8]
F088	Read / Write	0000 0000	LCD_LUT1 Bit[7:4] : for Gray level 3 Bit[3:0] : for Gray level 2
F089	Read / Write	0000 0000	LCD_LUT0 Bit[7:4] : for Gray level 1 Bit[3:0] : for Gray level 0
F08A	Read / Write	x000 0000	Bit[7] : Reserved Bit[6:0] : Virtual Display Character count – 1 It supports horizontal virtual size up to 128 characters or 1024 pixels.
F08B	Read / Write	xx00 0000	Bit[7:6] : Reserved Bit[5:0] : WF count for STN panels 000000=WF pin toggles every frame 000001=WF pin toggles every 2 LP pulses 000010=WF pin toggles every 3 LP pulses : 111111=WF pin toggles every 64 LP pulses
F08C	Read / Write	xxxx 0000	Bit[7:4] : Reserved Bit[3:0] : Horizontal non-display period 0000=2 characters (16 pixels) 0001=3 characters (24 pixels) : 1111=17 characters (136 pixels)

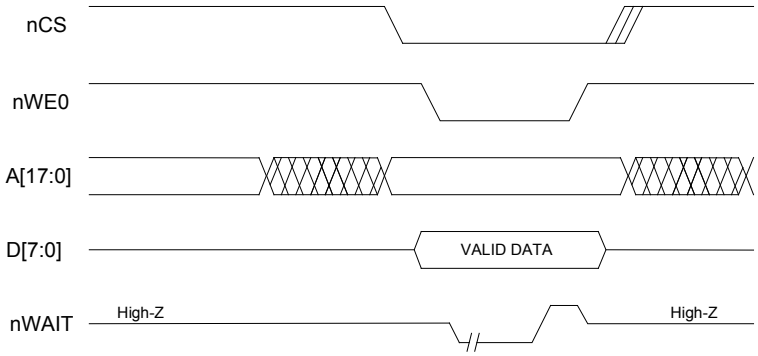
F08D	Read / Write	xxxx 0000	Bit[7:4] : Reserved Bit[3:0] : Vertical non-display period 0000: 1 line 0001: 2 lines : 1111: 16 lines
F08E	Read / Write	0000 000x	Bit[7:4] : Pixel Clock Divider 0000=24 MHz (divided by 1) 0001=12 MHz (divided by 2) 0010=8 MHz (divided by 3) 0011=6MHz (divided by 4) : 1111=1.5MHz (divided by 16) Bit[3] : Display Blank 0=Normal 1=Blank Bit[2] : Display Invert 0=Normal 1=Invert Bit[1] : LCD_ON Polarity 0=LCD_ON pin active low 1=LCD_ON pin active high Bit[0] : Reserved
F08F	Read /Write	x000 0000	Bit[7] : Reserved Bit[6:0] : Number of frames to start – 1 (Maximum 128 frames)
F090	Read /Write	xx00 0000	Bit[7:6] : Reserved Bit[5:0] : Horizontal Front Porch for TFT panels 000000=1 pixel 000001=2 pixels : 111111=64 pixels
F091	Read /Write	xx00 0000	Bit[7:6] : Reserved Bit[5:0] : Horizontal Back Porch for TFT panels 000000=1 pixel 000001=2 pixels : 111111=64 pixels
F092	Read /Write	xxx0 0000	Bit[7:5] : Reserved Bit[4:0] : Horizontal Pulse Width for TFT panels 00000=1 pixel 00001=2 pixels : 11111=32 pixels
F093	Read /Write	0000 0000	Bit[7:0] : Scratch Pad register
F094	Read /Write	xx00 0000	Bit[7:6] : Reserved Bit[5:0] : Vertical Front Porch for TFT panels 000000=1 line 000001=2 lines : 111111=64 lines
F095	Read /Write	xx00 0000	Bit[7:6] : Reserved Bit[5:0] : Vertical Back Porch for TFT panels 000000: 1 line 000001: 2 lines : 111111: 64 lines
F096	Read /Write	xxx0 0000	Bit[7:5] : Reserved Bit[4:0] : Vertical Pulse Width for TFT panels 00000: 1 line 00001: 2 lines : 11111: 32 lines
F100	Read /Write	00xx xx00	Bit[7] : Enable / Disable 0=Disable Sprite 1=Enable Sprite Bit[6] : Transparency 0=Transparency disable 1=Transparency enable When enabled: Sprite data = 00 becomes transparent and LCD background will be displayed instead. Bit[5:2] : Reserved Bit[1:0] : Sprite Modes Select 00=reserved 01=Sprite with 2 bit-per-pixel 10=reserved 11=reserved
F102	Read /Write	0000 0000	Bit[7:0] : SP_LUT0L[7:0]
F103	Read / Write	0000 0000	Bit[7:0] : SP_LUT0H[7:0]
F104	Read / Write	0000 0000	Bit[7:0] : SP_LUT1L[7:0]

F105	Read / Write	0000 0000	Bit[7:0] : SP_LUT1H[7:0]
F106	Read / Write	0000 0000	Bit[7:0] : SP_LUT2L[7:0]
F107	Read / Write	0000 0000	Bit[7:0] : SP_LUT2H[7:0]
F108	Read / Write	0000 0000	Bit[7:0] : SP_LUT3L[7:0]
F109	Read / Write	0000 0000	Bit[7:0] : SP_LUT3H[7:0]
F10A	Read / Write	0000 0000	Bit[7:0] : Sprite Horizontal Pixel Count – 1 Maximum 256 pixels
F10B	Read / Write	0000 0000	Bit[7:0] : Sprite Vertical Line Count – 1 Maximum 256 lines
F10C	Read / Write	0000 0000	Bit[7:0] : Sprite Horizontal Start Position bit[7:0]
F10D	Read / Write	xxxx xx00	Bit[7:2] : Reserved Bit[1:0] : Sprite Horizontal Start Position bit[9:8] Sprite Horizontal Start Position bit[9:0] is measured in pixels and counted from left to right of the edge of the panel display (i.e. not virtual display).
F10E	Read / Write	0000 0000	Bit[7:0] : Sprite Vertical Start Position bit[7:0]
F10F	Read / Write	xxxx xxx0	Bit[7:1] : Reserved Bit[0] : Sprite Vertical Start Position bit[8] Sprite Vertical Start Position bit[8:0] is measured in lines and counted from top to bottom of the edge of the panel display (i.e. not virtual display).
F142	Write Only	0000 0000	Bit[7:0] : Sprite / overlay storage starting address bit[7:0]
F143	Write Only	0000 0000	Bit[7:0] : Sprite / overlay storage starting address bit[15:8]
F144	Write Only	0000 0000	Bit[7:2] : Reserved Bit[1:0] Sprite / overlay storage starting address bit[17:16] This is the starting address to put the sprite/overlay image
F180	Read Only	0000 0000	Bit[7:0] : Background Color bit[7:0]
F181	Read Only	0000 0000	Bit[7:0] : Background Color bit[15:8]
F182	Read Only	0000 0000	Bit[7:0] : Foreground Color bit[7:0]
F183	Read Only	0000 0000	Bit[7:0] : Foreground Color bit[15:8]
F500	Read / Write	1110 1110	CS0 Configuration Port - Pulse Width Bit[7:4] : Write Cycle Pulse Width 0000=1 memory clock (24 MHz -> 41.6ns) 0001=2 memory clocks : 1110=15 memory clocks 1111=Reserved Bit[3:0] : Read Cycle Pulse Width 0000=1 memory clock (24 MHz -> 41.6ns) 0001: 2 memory clocks : 1110:15 memory clocks 1111: Reserved
F501	Read / Write	0000 0000	CS0 Configuration Port - Control Bit[7] : Enable bit 0=Disable CS0 1=Enable CS0 Bit[6] : Memory data bus width 0=8-bit memory data bus width 1=16-bit memory data bus width Bit[5] : 16-bit SRAM option 0=two 8-bit SRAMs 1=one 16-bit SRAM Bit[4] : Reserved Bit[3] : CS0 assertion time relative to address assertion. 0=CS0 and address assert at the same time 1=CS0 lags address by 1 memory clock. Bit[2] : CS0 Negation Timing 0=CS0 and Address negate at the same time 1=CS0 leads Address by 1 memory clock in write access. Bit[1] : Write Enable Assertion Time 0=Write Enable and Address Assert at the same time. 1=Write Enable lags Address by 1 memory clock. Bit[0] : Write Enable Negation Time 0=Write Enable and Address negate at the same time. 1=Write Enable leads Address by 1 memory clock.

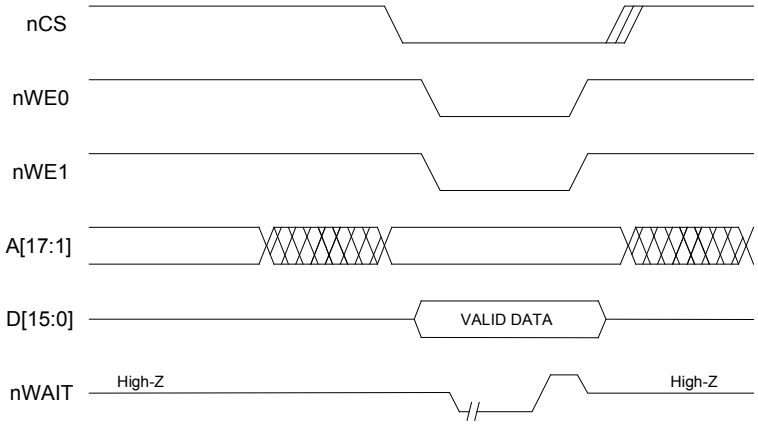
F504	Read / Write	Bit[3:0] = 1110	CS1 Configuration Port - Pulse Width Bit[7:4] : Reserved Bit[3:0] : Read Cycle Pulse Width 0000=1 memory clock (24 MHz -> 41.6ns) 0001=2 memory clocks 0001=3 memory clocks : 1101=14 memory clocks 1110=15 memory clocks 1111=Reserved
F505	Read / Write	0000 0000	CS1 Configuration Port - Control Bit[7] : Enable bit 0=Disable CS1 1=Enable CS1 Bit[6] : Memory data bus width 0=8-bit memory data bus width 1=16-bit memory data bus width Bit[5] : Reserved Bit[4] : Reserved Bit[3] : CS1 assertion time relative to address assertion. 0=CS1 and Address assert at the same time 1=CS1 lags Address by 1 memory clock. Bit[2] : CS1 Negation Timing 0=CS1 and Address negate at the same time 1=CS1 leads Address by 1 memory clock in write access. Bit[1:0] : Reserved
F6C4	Read / Write	xx11 0011	Set Memory Clock Divide Bit[7:6] = Reserved Bit[5:0] = 010000 to set 24MHz memory clock for normal operations (with sprite) Bit[5:0] = 000000 reserved for high memory clock (for fast read)

7. Timing Sequence

7.1 8bit Host Interface - Write Timing

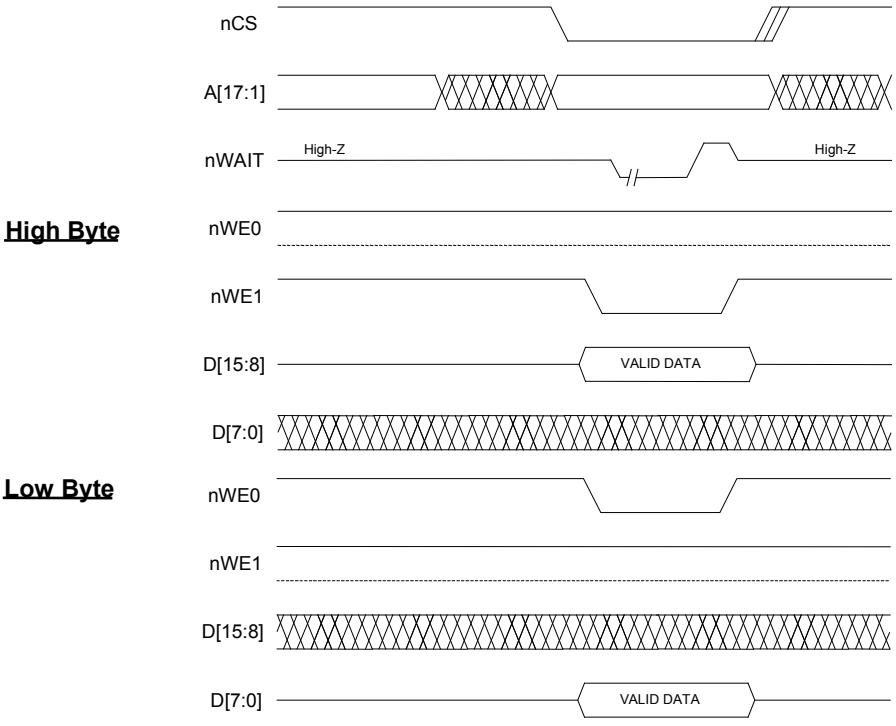


7.2 16bit Data Host Interface, 16-bit Data Write, Timing Sequence



7.3 16bit Data Host Interface, 8-bit Data Write, Timing Sequence

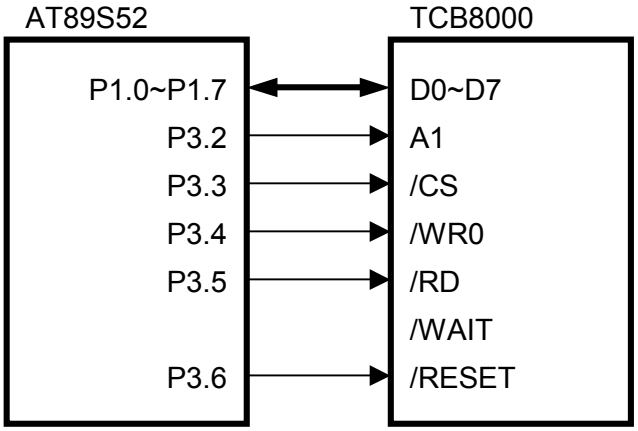
(Little Edian, A0=1, for high byte data; A0=0, for low byte date)



Please refer to T8000 technical manual, for details.

8. Programming Example

The following are the programming example of using indirect mode (1bit add, 8bit data).
The application is based on using AT89S52.



Note.
The given program routines are an example only. It is necessary to evaluate before apply into the product applications

```
// define the data connection
define com_add P1;    // define data port
sbit   A1 = P3^2;     // A0 signal
sbit   CS = P3^3;     // CS signal
sbit   WR0 = P3^4;    // WR signal
sbit   RD = P3^5;     // RD signal


// send a command
void SdCmd(uchar Command)
{
  A1=0;               // select add
  com_add = Command;  // prepare the command
  CS=0;               // enable transfer
  WR0=0;
  WR0=1;              // latch in the command
  CS=1;               // disable data port
}



// Command End
void CmdEnd()
{
  A1=1;               // select add
  com_add = 1;        // prepare the command
  CS=0;               // enable transfer
  WR0=0;
  WR0=1;              // latch in the command
  CS=1;               // disable data port
}



// send a full command packet
void WritePKG(uchar *pkg)
// in the pkg,
// first byte is no. of command and data
// following is the opcode
// then the data
{
  uchar i;
  for(i=*pkg;i;i--)
  SdCmd(*(++pkg));
  CmdEnd();
}
```

```
// access the TFT internal contrast setting
// via T8000 GPIO port
// TCB8000C only
uchar bdata GpioData;
sbit SDIN  = GpioData^3;
sbit SCLK  = GpioData^2;
sbit SENA  = GpioData^1;

void SdGamma(uchar regcode, uint regdata)
{
 uchar i,j,Buffer[4];
  GpioData=0xff;
  Buffer[0]=4;
  Buffer[1]=0x83;
  Buffer[2]=0x41;
  Buffer[3]=0xf2;
  Buffer[4]=GpioData;
  WritePKG(Buffer);
  SENA=0;
  Buffer[0]=4;
  Buffer[1]=0x83;
  Buffer[2]=0x41;
  Buffer[3]=0xf2;
  Buffer[4]=GpioData;
  WritePKG(Buffer);
  for(j=0;j<3;j++)
  {
    switch (j)
      {
      case 0:BitData=regcode;break;
      case 1:BitData=regdata>>8;break;
      case 2:BitData=regdata;break;
      }
    for (i=0;i<8;i++)
      {
      SCLK=0;
      SDIN=BitData7;
      Buffer[0]=4;
      Buffer[1]=0x83;
      Buffer[2]=0x41;
      Buffer[3]=0xf2;
      Buffer[4]=GpioData;
      WritePKG(Buffer);
    SCLK=1;
      Buffer[0]=4;
      Buffer[1]=0x83;
      Buffer[2]=0x41;
      Buffer[3]=0xf2;
      Buffer[4]=GpioData;
      WritePKG(Buffer);
      BitData=BitData<<1;
      }
  }
  GpioData=0xff;
  Buffer[0]=4;
  Buffer[1]=0x83;
  Buffer[2]=0x41;
  Buffer[3]=0xf2;
  Buffer[4]=GpioData;
  WritePKG(Buffer);
}
```

```
// init for 320x240 TFT LCM
uchar code Set_F500[]={4,0x83,0x00,0xf5,0x00};
uchar code Set_F500[]={4,0x83,0x00,0xf5,0x00};
uchar code Set_F504[]={4,0x83,0x04,0xf5,0x04};
uchar code Set_F505[]={4,0x83,0x05,0xf5,0x80};
uchar code Set_F6C4[]={4,0x83,0xc4,0xf6,0x10};
uchar code Set_F080[]={4,0x83,0x80,0xf0,0xfc};
uchar code Set_F08E[]={4,0x83,0x8e,0xf0,0x32};
uchar code Set_F090[]={4,0x83,0x90,0xf0,0x14};
uchar code Set_F091[]={4,0x83,0x91,0xf0,0x25};
uchar code Set_F092[]={4,0x83,0x92,0xf0,0x1e};
uchar code Set_F094[]={4,0x83,0x94,0xf0,0x05};
uchar code Set_F095[]={4,0x83,0x95,0xf0,0x0e};
uchar code Set_F096[]={4,0x83,0x96,0xf0,0x03};
uchar code Set_F08F[]={7,0x8f,0x69,0x45,0x61,
                       0x67,0x6c,0x65};
uchar code Set_F240[]={4,0x83,0x40,0xf2,0x00};
uchar code Set_F241[]={4,0x83,0x41,0xf2,0xff};

void initLCDM(void)
{
// execute all the setting in above
  WritePKG(Set_F500);
  WritePKG(Set_F504);
  WritePKG(Set_F505);
  WritePKG(Set_F6C4);
  WritePKG(Set_F080);
  WritePKG(Set_F08E);
  WritePKG(Set_F090);
  WritePKG(Set_F091);
  WritePKG(Set_F092);
  WritePKG(Set_F094);
  WritePKG(Set_F095);
  WritePKG(Set_F096);
  WritePKG(Set_8F);
  WritePKG(Set_F240);
  WritePKG(Set_F241);
  SdGamma(0x70,0x000a); // gamma and contrast set
  SdGamma(0x72,0x4008); // for TCB8000C
  SdGamma(0x70,0x001e); // for TCB8000C
  SdGamma(0x72,0x00a8); // for TCB8000C
}



// write ascii string
void PrintASCII(uint X, Y, uchar *pstr)
{
  uchar Buffer[6], NoOfChar;
  Buffer [0]=2;     // packet size
  Buffer [1]=0x10;  // select char opcode
  Buffer [2]=0x00;  // select 8x8 AXCII
  WritePKG(TempData);
  Buffer[0]=5;      // packet size
  Buffer[1]=0x12;   // set location
  Buffer[2]=X;      // X-location LSB
  Buffer[3]=X>>8;   // X-location MSB
  Buffer[4]=Y;      // Y-location LSB
  Buffer[5]=Y>>8;   // Y-location MSB
  WritePKG(Buffer);
  NoOfChar=strlen(pstr);
  SdCmd(0x17);          // string input opcode
  SdCmd(NoOfChar);      // send no of char
  while(*pstr>0)
  {
    SdCmd(*pstr++);     //send the char
  }
  CmdEnd();
}
```

```
// draw drawing sub routine
void Draw_Dot(uint X, Y)
{
  uchar Buffer[6];
  Buffer[0]=5;       // packet size
  Buffer[1]=0x23;    // draw dot opcode
  Buffer[2]=X;       // x-location LSB
  Buffer[3]=X>>8;    // x-location MSB
  Buffer[4]=Y;       // y-location LSB
  Buffer[5]=Y>>8;    // y-location MSB
  WritePKG(Buffer);
}



// line drawing sub routine
void Draw_Line(uint x1, y1, x2, y2)
{
  uchar Buffer[10];
  Buffer[0]=9;         // packet size
  Buffer[1]=0x24;      // line draw opcode
  Buffer[2]=x1;        // x1-location LSB
  Buffer[3]=x1>>8;     // x1-location MSB
  Buffer[4]=y1;        // y1-location LSB
  Buffer[5]=y1>>8;     // y1-location MSB
  Buffer[6]=x2;        // x2-location LSB
  Buffer[7]=x2>>8;     // x2-location MSB
  Buffer[8]=y2;        // y2-location LSB
  Buffer[9]=y2>>8;     // y2-location MSB
  WritePKG(Buffer);
}



// Rectangle Drawing sub-rountine
void Draw_Rect(uint x1, y1, x2, y2)
{
  uchar Buffer[10];
  Buffer[0]=9;         // packet size
  Buffer[1]=0x26;      // rectangle draw opcode
  Buffer[2]=x1;        // x1-location LSB
  Buffer[3]=x1>>8;     // x1-location MSB
  Buffer[4]=y1;        // y1-location LSB
  Buffer[5]=y1>>8;     // y1-location MSB
  Buffer[6]=x2;        // x2-location LSB
  Buffer[7]=x2>>8;     // x2-location MSB
  Buffer[8]=y2;        // y2-location LSB
  Buffer[9]=y2>>8;     // y2-location MSB
  WritePKG(Buffer);



void Draw_Circle(uint X, Y, uchar R)
{
  uchar Buffer[7];
  Buffer[0]=6;         // packet size
  Buffer[1]=0x28;      // circle  draw opcode
  Buffer[2]=X;         // center-x LSB
  Buffer[3]=X>>8;      // center-x MSB
  Buffer[4]=Y;         // center-y LSB
  Buffer[5]=Y>>8;      // center-y MSB
  Buffer[6]=R;         // radius
  WritePKG(Buffer);
}
```