EmbeddedGUI User Manual

Version 0.6.2

Wang Chao

eluneyun@gmail.com

Aug 2011

Contents

Chapter 1 The EmbeddedGUI Library

1.1 Introduction

The EmbeddedGUI Library is designed for embedded systems with very small RAM and flash space. The LCD interface and graphic functions are implemented in C. It also provides many controls and desktop implemented in C++.

This library is designed without any dynamic memory allocation. So it is easy to be transplanted to embedded systems based on ARM or DSP.

1.2 Architecture

Figure 1 Library Architecture

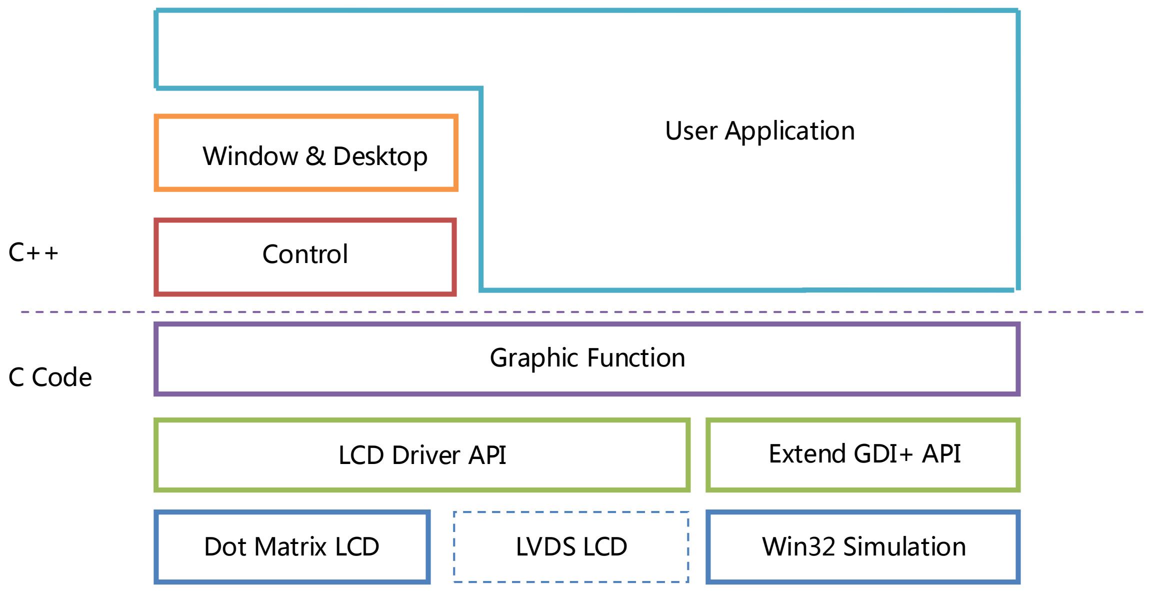


Table 1 Library Files

Library Module	Files:
Dot Matrix LCD	IO.h
Win32 Simulation	BufferDriver.h BufferDriver.cpp
LCD Driver API	LCDDriver.h LCDDriver.cpp
Extend GDI+ API	ExtGDIplus.h
Graphic Function	GraphicFunc.h GraphicFunc.cpp EFont.h
Control	EControl.h EControl.cpp EComponent.h
Window	EFrame.h EFrame.cpp ECursor.h ECursor.cpp

1.3 LCD interface

1.3.1 LCD Buffer

In order to save more RAM resources, only one image buffer exists in RAM. You may find the EG_LCD_Buffer in LCDDriver.cpp . It defines as follows:

```
unsigned char EG_LCD_Buffer[SUB_ROW][COLUMN];
```

All the graphic functions declared in GraphicFunc.cpp will draw on this buffer. Buffer in the RAM will not be flushed into LCD, until EG_LCDFlushBuffer() is invoked, displaying the image finally.

By defining the macro EG_NO_EBUFFER, you can remove LCD buffer variable. Instead, EmbeddedGUI will write the pixel data directly to LCD driver IC. This feature might be very useful for some embedded systems with limited RAM resource. However, system will run much slower without this LCD buffer. Since plenty of time will be wasted in system bus IO operation. Note that this macro can't be defined when using a monochrome screen in current version.

1.3.2 LCD Functions

There are two groups of LCD functions:

1 LCD Operation API

These functions will write commands and values directly into LCD module. Note that you should change the code below if using other LCD modules.

```
void EG_LCDInit(void);
void EG_LCDWriteCommand(unsigned char cmd);
void EG_LCDClearScreen(void);
void EG_LCDClose();
void EG_LCDOpen();
void EG_LCDShowByte(unsigned char val,unsigned int row,unsigned int
column);
void EG_LCDShowBuffer(unsigned char* pBuf,unsigned char
size,unsigned int row,unsigned int column);
void EG_LCDWriteValue(unsigned char val,unsigned char CS1,unsigned
char CS2);
void EG_LCDFlushBuffer();
```

2 LCD Buffer API

```
void EG_LCDClearBuffer();
void EG_LCDSetPixel(unsigned int PixelRow,unsigned int PixelColumn);
void EG_LCDClearPixel(unsigned int PixelRow,unsigned int
PixelColumn);
```

```
unsigned char EG_LCDGetPixel(unsigned int PixelRow,unsigned int
PixelColumn);
unsigned char EG_LCDGetPixel(unsigned int PixelRow,unsigned int
PixelColumn);
void EG_LCDSetByte(unsigned int PixelRow,unsigned int
PixelColumn,unsigned char val);
void EG_LCDSetVerticalByte(unsigned int PixelRow,unsigned int
PixelColumn,unsigned char val);
```

1.3.3 LCD Driver

There are six LCD drivers supported now:

- KS0107 & KS0108
- T6393C
- S6B33C
- RA8803
- S1D13305
- HX8238

Define the macros in EGUIMacro.h to use these drivers.
EG_KS010X_LCD_DRIVER or EG_T6963C_LCD_DRIVER
or EG_S6B33C_LCD_DRIVER or EG_RA8803_LCD_DRIVER
or EG_S1D13305_LCD_DRIVER or EG_HX8238_LCD_DRIVER

1.3.4 MCU Pin Mapping

These pin definitions can be found in IO.h

In most cases, you need to change these pin definitions to adapt to your hardware. Meanwhile, there are some platform macros which rely on hardware MCU. EmbeddedGUI designer is highly recommended for EGUIMacro.h and IO.h macro configuration.

Table 2 Pin definitions of KS010X LCD driver

Pin Function	Pin NO.
RS	P2.0
RW	P2.1
EN	P2.2
CS1	P2.3
CS2	P2.6
RST	P2.7
DB0~DB7	P3.0~P3.7

Table 3 Pin definitions of T6963C LCD driver

Pin Function	Pin NO.
WR	P3.1
RD	P3.2
CE	P3.3
CD	P3.0
DB0~DB7	P1.0~P1.7

Table 4 Pin definitions of S6B33C LCD driver

Pin Function	Pin NO.
A0	P3.1
WR	P3.2
RD	P3.3
D7~D0	P1.0~P1.7
CS	P3.4
BL_EN(backlight enable)	P3.5
RST	P3.6

Table 5 Pin definitions of RA8803 LCD driver

Pin Function	Pin NO.
BUSY	P3.0
WR	P3.2
RD	P3.3
D7~D0	P1.0~P1.7
CS	P3.4
RS(H: Data L: Command)	P3.1
RST	P3.5

Table 6 Pin definitions of S1D13305 LCD driver

Pin Function	Pin NO.
A0	P3.1
WR	P3.2
RD	P3.3
D7~D0	P1.0~P1.7
CS	P3.4
RST	P3.5

Table 7 Pin definitions of S3C2440 LCD driver

Pin Function	Pins of S3C2440
DEN	VDEN
VSYNC	VSYNC
HSYNC	HSYNC
DOTCLK	VCLK
RR[7:0]	VD[23:19]
GG[7:0]	VD[15:10]
BB[7:0]	VD[7:3]
SHUT	LCD_PWREN

1.4 Basic graphic functions

1.4.1 RGB System

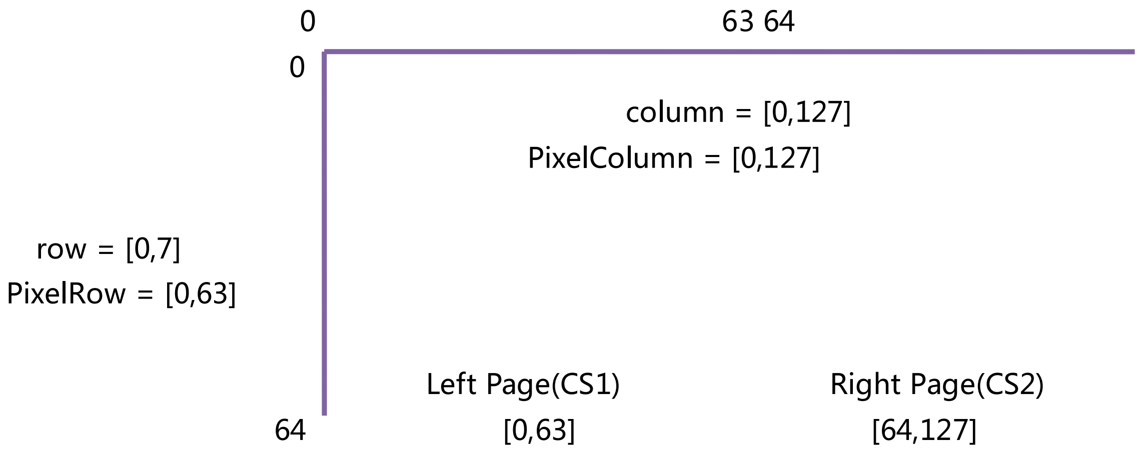
In some cases, it is necessary to use a RGB system, which means that every pixel needs 3 bytes in RAM, other than only one bit. You can simply define the macro EG_RGB_COLOR in EGUIMacro.h .In RGB system, LCD buffer will be

```
        EColor EG_Sim_Buffer[ROW][COLUMN];
  or    EColor EG_LCD_Buffer[EG_ROW][EG_COLUMN];
```

in the file BufferDriver.h or LCDDriver.h

1.4.2 Pixel Addressing

In the monochrome system, the parameters of graphic functions are based on the pixel addressing below. A dot matrix LCD with 128*64 pixels is illustrated as follows:



1.4.3 Build a system without controls

There are some simple systems that do not require any control or window. You can only use the graphic functions in GraphicFunc.cpp.
Refer to 8051 example for more details.

1.4.4 Font

There are five fonts in current library.

Table 8 Fonts in EmbeddedGUI library

Font Name	Font Index	Size	Index
EG_SmallFont	0	8*8	ASCII code only 0x20 to 0x7F in ASCII 1 to 95 in font index
EG_NormalFont	1	16*8	
EG_NormalCambriaFont	2	16*8	
EG_NormalBoldCambriaFont	3	16*8	
EG_NormalBoldFont	4	16*8	

Note that EFont will only be used in Font String Graphic Functions such as

```
void EG_Graphic_DrawFontString(unsigned int x,unsigned int y,
                               char* string,const EFont& font);
```

String Graphic Functions such as EG_Graphic_DrawString use EG_SmallFont as default font.

1.5 Window and Controls

1.5.1 Common Base Class EControl

EControl is the common base class of all the other controls. It defines the size and position of the control. It also provides a common method to invoke redraw function and callback function.

EControl members (some non-important members omitted):	
EControl(void)	Constructor
RedrawControl(void)	Call the redraw function
NoticeControl(EEvent *pevent)	Call the callback function
m_bNeedRedraw	Its value is true, if the control need redraw. Normally, this value will be modified in the callback functions.
m_fpRedrawFunc	The function pointer of redraw and callback function.

m_pControlPos	Position of the control
m_pStyle	draw style, used in redraw function
m_zArea	Size of the control

Table 9 EControl members

1.5.2 The Redraw and Callback Function

The Redraw Function will be invoked when updating a control. Generally, it is a global function. When a control is initializing, its function pointer will be used as a parameter and saved in EControl::m_fpRedrawFunc .

Its prototype defines as follows:

```
typedef void (*EG_RedrawFunc)(EControl* control);
```

The default redraw functions of most controls are already defined. However, you can define custom redraw function to change its appearance.

```
EG_RedrawFunc_DefaultEButton(EControl *control)
EG_RedrawFunc_DefaultECheckButton(EControl *control)
EG_RedrawFunc_DefaultEComboBox(EControl *control)
EG_RedrawFunc_DefaultEEdit(EControl *control)
EG_RedrawFunc_DefaultEExtLable(EControl *control)
EG_RedrawFunc_DefaultELable(EControl *control)
EG_RedrawFunc_DefaultEProgress(EControl *control)
```

The Callback Functions are those functions invoked when an event is related to the control. For example, if the cursor generates a left click event on a button, the button callback function will be called to handle this event.

Its prototype is as follows:

```
typedef void (*EG_CallbackFunc)(EControl* control,EEvent* curevent);
```

Some controls need the default callback, while others don't. Same as those redraw functions, default callback functions have been already defined.

```
EG_CallbackFunc_DefaultECheckButton(EControl *control, EEvent *curevent)
EG_CallbackFunc_DefaultEComboBox(EControl *control, EEvent *curevent)
EG_CallbackFunc_DefaultEEdit(EControl *control, EEvent *curevent)
EG_CallbackFunc_DefaultEExtLable(EControl *control, EEvent *curevent)
```

Note that you MUST invoke the default callback function at the end of your own callback function.

Example:

```
void EG_CallbackFunc_EEdit1(EControl* control,EEvent* curevent){
    //your code
    EEdit* currentControl=(EEdit*)control;
    if ((currentControl-> m_saString==" abc" )
        &&(currentControl-> m_ulength==3)){
         BSP_SendUART( "abc in edit.\0" );
   }
    //default callback function
    EG_CallbackFunc_DefaultEEdit(control,curevent);
}
```

1.5.3 Window and Desktop

EWindow is a container of controls. The EWindow provides two redraw functions.

```
//redraw all the controls
//Call the redraw functions of all controls in current window.
virtual void ForceRedrawAllControls();

//only redraw those controls which need to be updated.
//The m_bNeedRedraw of these controls are true.
virtual void RedrawSelectedControls();
```

If the macro EG_WINDOW_HOOK_FUNC is defined, you can use the hook functions to draw a custom window. The hook functions are invoked in the window redraw functions above.

```
void EG_HookFunc_RedrawESimpleWindow(EPlainWindow* wnd);
void EG_HookFunc_RedrawEWindow(EPlainWindow* wnd);
```

EDesktop is a container of EWindows. Note that the redraw functions of EDesktop and EWindow are different. The redraw functions of EWindow, such as EWindow::ForceRedrawAllControls will merely draw all the controls on LCD buffer. No change will be found on LCD screen until copying the buffer to RAM in LCD. On the contrary, the redraw functions of EDesktop will also do the buffer copying. EDesktop provides schedule redraw function ScheduleRedraw() , which invoke the global redraw function after several times of selected redraw function in addition to global and selected redraw functions.

ESimpleDesktop do not display anything on the screen. You can derive from it to create your own desktop. EDesktop is the default platform derived from ESimpleDesktop, including icons, start menus and a system bar.
Figure 2 illustrates the EDesktop, using the default EStyle.

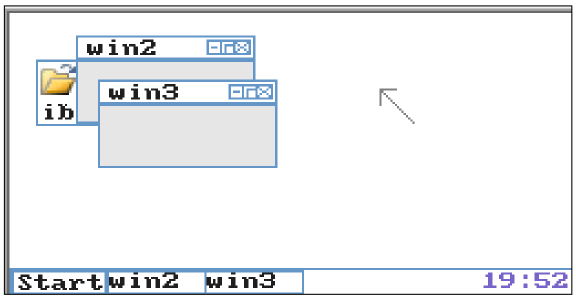


Figure 2 appearance of default EDesktop

1.5.4 Dispatch the event

Event indicates something happened from cursor, keyboard, screen even some controls. The EEvent has three data members:

unsigned int	m_uSource	the event source
unsigned int	m_uMessage	content of the event
EPosition	m_pPosition	position of the event

Table 10 EEvent data members

m_uSource	m_uMessage	m_pPosition
0 N/A (Idle Event)	0	0
1 Cursor	0 N/A	Cursor Position
	1 LeftClick	
	2 RightClick	
	3 LeftDoubleClick	
	4 RightDoubleClick	
	5 MidClick	
2 Keyboard	Pressed Key	Cursor Position
>=3 User Define		

Table 11 the definition of EEvent

The events dispatch procedure:

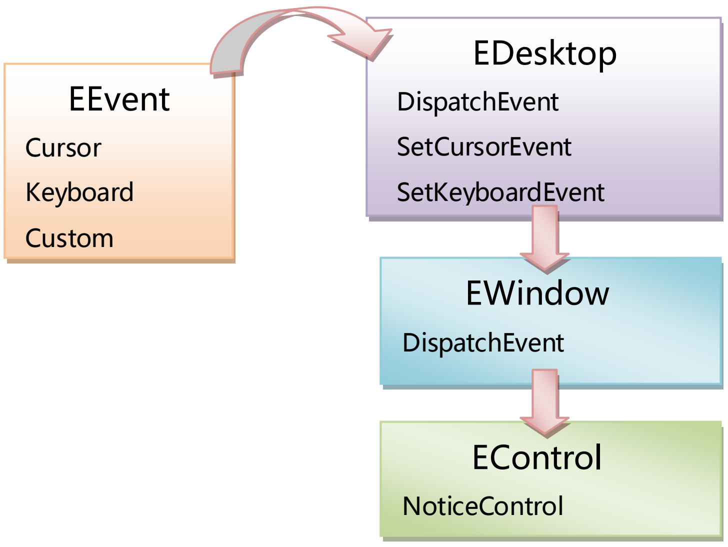


Figure 3　The process of events dispatch

1.5.5　Rich Style

Most controls have two appearances: normal and rich style appearance. Note that you should also define EG_RGB_COLOR and include the font graphic functions and other fonts, when using the EG_RICH_STYLE.

You should also evaluate the performance of your platform. Because in the rich style, all the controls require much more time to be paint. Extra RAM and flash space is also required because of the font.

Based on the tests on my development board, I highly recommend that when using the rich style controls, you should use 32bits MPU such as ARM7 or better. Some low performance MCUs such as 8051 or MSP430 will spend most of time in redraw functions.

1.6　RTOS Support

1.6.1　FreeRTOS Support

The EmbeddedGUI library supports FreeRTOS now. You can transplant EmbeddedGUI library to FreeRTOS easily via EmbeddedGUI designer. For more details, please refer to the example in chapter 3.4.

```
void vGUIInit(void){
	//init the LCD
	EG_LCDInit();
	// Clear the buffer, set the buffer to 0
	EG_LCDClearBuffer();
	// Global Redraw upon the buffer
	desk.GlobalRedraw();
	//create the redraw task
	xTaskCreate(  vGUIRedraw,  "Redraw",  configMINIMAL_STACK_SIZE,  NULL,
				mainREDRAW_TASK_PRIORITY, NULL );
}

void vGUIRedraw( void * pvParameters ){
	/* Parameters are not used. */
	( void ) pvParameters;
	portTickType xLastWakeTime;
	const portTickType xFrequency = 10;

	// Initialise the xLastWakeTime variable with the current time.
	xLastWakeTime = xTaskGetTickCount ();
	for( ;; ){
		// Wait for the next cycle.
		vTaskDelayUntil( &xLastWakeTime, xFrequency );
		//redraw all the desktop here.
		desk.ScheduleRedraw();
	}
}
```

When using the EmbeddedGUI in FreeRTOS, you need to create a task to invoke the ESimpleDesktop::ScheduleRedraw() or other redraw function periodically.

In most cases, you also need another task to create the events and dispatch them, since all the controls are driven by events. In the example in chapter 3.4, LPC2148 receives events from UART and dispatches them. The utility program SerialCursor is used to generate EEvent in this case.

Chapter 2 Library Reference

2.1 Global Functions

2.1.1 Graphic Functions

There are two series of graphic functions depending on different color systems: monochrome or RGB. The functions using RGB usually use EPosition,ESize and EColor as parameters. We will mainly discuss the monochrome graphic functions and omit part of RGB graphic functions due to similarity.

Graphic	Monochrome	RGB
Point	EG_Graphic_DrawWhitePoint EG_Graphic_DrawBlackPoint	EG_Graphic_DrawPoint
Line	EG_Graphic_DrawHorizonLine EG_Graphic_DrawVerticalLine EG_Graphic_DrawWhiteHorizonLine EG_Graphic_DrawWhiteVerticalLine EG_Graphic_DrawLine EG_Graphic_DrawWidthLine	EG_Graphic_DrawHorizonLine EG_Graphic_DrawVerticalLine EG_Graphic_DrawLine EG_Graphic_DrawWidthLine
Rectangle	EG_Graphic_DrawEmptyRectangle EG_Graphic_DrawFilledRectangle EG_Graphic_DrawWhiteFilledRectangle	EG_Graphic_DrawEmptyRectangle EG_Graphic_DrawFilledRectangle
Circle	EG_Graphic_DrawEmptyCircle EG_Graphic_DrawFilledCircle EG_Graphic_DrawWhiteFilledCircle	EG_Graphic_DrawEmptyCircle EG_Graphic_DrawFilledCircle
Arc	EG_Graphic_DrawQuarterArc EG_Graphic_DrawArc EG_Graphic_DrawPieSlice	EG_Graphic_DrawQuarterArc EG_Graphic_DrawArc EG_Graphic_DrawPieSlice
Ellipse	EG_Graphic_DrawEmptyEllipse EG_Graphic_DrawFilledEllipse	EG_Graphic_DrawEmptyEllipse EG_Graphic_DrawFilledEllipse
String	EG_Graphic_DrawChar EG_Graphic_DrawString EG_Graphic_DrawConstString EG_Graphic_DrawNotChar EG_Graphic_DrawNotString	EG_Graphic_DrawChar EG_Graphic_DrawString EG_Graphic_DrawConstString EG_Graphic_DrawNotChar EG_Graphic_DrawNotString
Bitmap	EG_Graphic_DrawBitmap	EG_Graphic_DrawBitmap
Font String	EG_Graphic_DrawFontChar EG_Graphic_DrawNotFontChar EG_Graphic_DrawFontString	EG_Graphic_DrawFontChar EG_Graphic_DrawNotFontChar EG_Graphic_DrawFontString

	EG_Graphic_DrawConstFontString EG_Graphic_DrawNotFontString	EG_Graphic_DrawConstFontString EG_Graphic_DrawNotFontString
Number	EG_Graphic_DrawNumber	EG_Graphic_DrawNumber

Table 12 Graphic Function

EG_Graphic_DrawWhitePoint(x,y)
This is a macro, x and y are both unsigned int, indicating the coordinate of the point. This macro will set the point in the buffer to white.

EG_Graphic_DrawBlackPoint(x,y)
This is a macro, x and y are both unsigned int, indicating the coordinate of the point. This macro will set the point in the buffer to black.

void EG_Graphic_DrawPoint(const EPosition& pos,const EColor& color);
This function will draw *const EColor& color* in the position of *const EPosition& pos*.

void EG_Graphic_DrawHorizonLine(unsigned int x,unsigned int y,unsigned int length);
This function will draw a horizontal line, (x,y) will be the coordinate of start point.
Note that you should avoid any operation that might write or read a pixel out of the LCD buffer range. And this rule applies to all graphic functions.
For example,

```
EG_Graphic_DrawHorizonLine(0,20,200);
```

This function might cause troubles for a LCD module with 128 * 64 pixels due to the fact that the length of horizontal line exceeds 128 pixels.

Also note that even though we have boundary checking in driver layer such as

```
if((PixelRow>EG_ROW)||(PixelColumn>EG_COLUMN)){
     return;
}
```

in the EG_LCDClearPixel() in BufferDriver.cpp ,yet you should still try not to draw anything out of the range.

void EG_Graphic_DrawVerticalLine(unsigned int x,unsigned int y,unsigned int length);
This function will draw a vertical line, (x,y) will be the coordinate of start point.

void EG_Graphic_DrawWhiteHorizonLine(unsigned int x,unsigned int y,unsigned int length);
This function will draw a white horizontal line, (x,y) will be the coordinate of start point. Only available in monochrome graphic functions.

void EG_Graphic_DrawLine(unsigned int x0,unsigned int y0,unsigned int x1,unsigned int y1,unsigned char virtualLine);

This function can draw any straight line, (x0,y0) and (x1,y1) are the coordinates of start point and end point. If the virtualLine is not zero, it will draw a virtual line.

Note that this function can draw horizontal and vertical line. However, you’d better avoid drawing them with it considering system performance.

void EG_Graphic_DrawWidthLine(unsigned int x0,unsigned int y0,unsigned int x1,unsigned int y1,unsigned char lineWidth);
This function can draw any straight line, whose width is bigger than 1.
(x0,y0) and (x1,y1) are the coordinates of start point and end point.
lineWidth should be less than 50.

void EG_Graphic_DrawEmptyRectangle(unsigned int x,unsigned int y,unsigned int height,unsigned int width);
This function can draw an empty rectangle. (x,y) is the coordinate of left top point of this rectangle and (height,width) is the size of it.

void EG_Graphic_DrawEmptyRectangle(const EPosition& pos,const ESize& size,const EColor& color);
This function can draw an empty rectangle. const EPosition& pos is the coordinate of left top point of this rectangle and const ESize& size defines its height and width.

void EG_Graphic_DrawWhiteFilledRectangle(unsigned int x,unsigned int y,unsigned int height,unsigned int width);

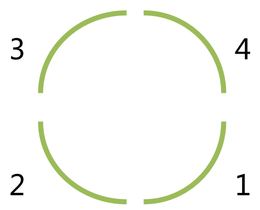
This function will draw a white rectangle. It is particularly useful when you need to erase part of the screen. (x,y) is the coordinate of left top point of this rectangle and (height,width) is the size of it.

void EG_Graphic_DrawEmptyCircle(unsigned int x0,unsigned int y0,unsigned int r);
This function will draw a circle whose center is (x0,y0) and radius is r.

```
void EG_Graphic_DrawQuarterArc(unsigned int x0,unsigned int y0,unsigned int r,unsigned char angle);
```

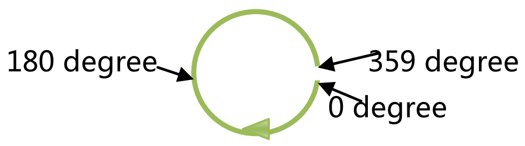
This function will draw arc of any quadrant. (x0,y0) is the center and r is the radius of arc. unsigned char angle could only be from 1 to 4.

angle:



```
void EG_Graphic_DrawArc(unsigned int x0,unsigned int y0,unsigned int r,unsigned int startAngle,unsigned int endAngle);
```

This function will draw arc of any angle. (x0,y0) is the center and r is the radius of arc. If startAngle is bigger than endAngle, this function will draw a minor arc. On the contrary, it will draw a major arc, if endAngle > startAngle.

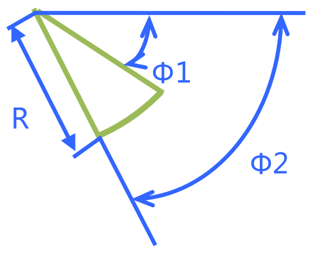


```
void EG_Graphic_DrawPieSlice(unsigned int x0,unsigned int y0,unsigned int r,unsigned int startAngle,unsigned int endAngle);
```

This function will draw a pie slice. (x0,y0) is the center and r is the radius of the circle. If startAngle is bigger than endAngle, this function will draw a minor arc. On the contrary, it will draw a major arc, when endAngle > startAngle.

The range of angle is [0,359].

Eg: `EG_Graphic_DrawPieSlice(EPosition(40,40),20,EColor(100,200,100),70,40);`



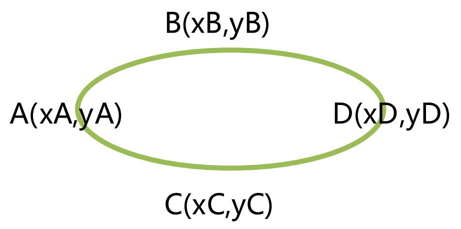
R = 20

Φ1 = 40

Φ2 = 70

```
void EG_Graphic_DrawEmptyEllipse(unsigned int leftX,unsigned int topY,unsigned int rightX,unsigned int bottomY);
```

This function will draw an ellipse. The parameters are as follows:



leftX = xA   topY = yB
rightX = xD   bottomY = yC

```
void EG_Graphic_DrawChar(unsigned int x,unsigned int y,char val);
```

This function will draw a character at the position (x,y).

Note that not all characters are supported; some characters that cannot be displayed on screen have been deleted from the font in order to save ROM space.

These characters below have been deleted.

Hex	Value	Hex	Value
01	SOH (Start of Header)	10	DLE (Data Link Escape)
02	STX (Start of Text)	11	DC1 (XON) (Device Control 1)
03	ETX (End of Text)	12	DC2 (Device Control 2)
04	EOT (End of Transmission)	13	DC3 (XOFF)(Device Control 3)
05	ENQ (Enquiry)	14	DC4 (Device Control 4)
06	ACK (Acknowledgment)	15	NAK (Negative Acknowledgement)
07	BEL (Bell)	16	SYN (Synchronous Idle)
08	BS (Backspace)	17	ETB (End of Trans. Block)
09	HT (Horizontal Tab)	18	CAN (Cancel)
0A	LF (Line Feed)	19	EM (End of Medium)
0B	VT (Vertical Tab)	1A	SUB (Substitute)
0C	FF (Form Feed)	1B	ESC (Escape)
0D	CR (Carriage Return)	1C	FS (File Separator)
0E	SO (Shift Out)	1D	GS (Group Separator)
0F	SI (Shift In)	1E	RS (Request to Send)
		1F	US (Unit Separator)

void EG_Graphic_DrawString(unsigned int x,unsigned int y,char* string);
This function will draw a string at the position (x,y).
Note that the string should end with '\0' . It must not end with '\n' . This applies to all string functions.

void EG_Graphic_DrawConstString(unsigned int x,unsigned int y,char* string,unsigned char size);
This function will draw a string at the position (x,y).The string can end with any character. Size is the length of the string.

void EG_Graphic_DrawNotChar(unsigned int x,unsigned int y,char val);
This function will draw a character at the position (x,y). However, this character is white with black background.
Eg: NotChar

void EG_Graphic_DrawBitmap(unsigned int x,unsigned int y,EBitmap* pBitmap);
This function will draw a bitmap at the position(x,y).

void EG_Graphic_DrawFontChar(unsigned int x,unsigned int y,char val,const EFont& font);
This function will draw a character at the position (x,y). The font of character is defined in font.

void EG_Graphic_DrawNumber(unsigned int x,unsigned int y,int number);
This function will draw the number at the position (x,y). Note that the number must be less than 100000.

2.1.2 Redraw and Callback Functions

EmbeddedGUI Library has already defined the default redraw and callback functions for all controls. For more details, please refer to section 1.5.2

2.2 Component Structure

2.2.1 General Structure

EPosition	Define the position of a point
ESize	Define the height and width of an area
EColor	Define the color of a pixel
EEvent	Define the event from cursor or keyboard

2.2.2 EStyle

EStyle defines the colors of different parts of control. The default redraw functions use these colors to draw controls.

There is a default global EStyle defined in EControl.cpp

```
EStyle EG_DefaultStyle;
```

Note that this global variable only exists when macro EG_RGB_COLOR is defined.

EStyle has five data members:

```
EColor m_cTextColor;
EColor m_cFrameColor;
EColor m_cContentColor;
EColor m_cBackgroundColor;
EColor m_cSelectColor;
```

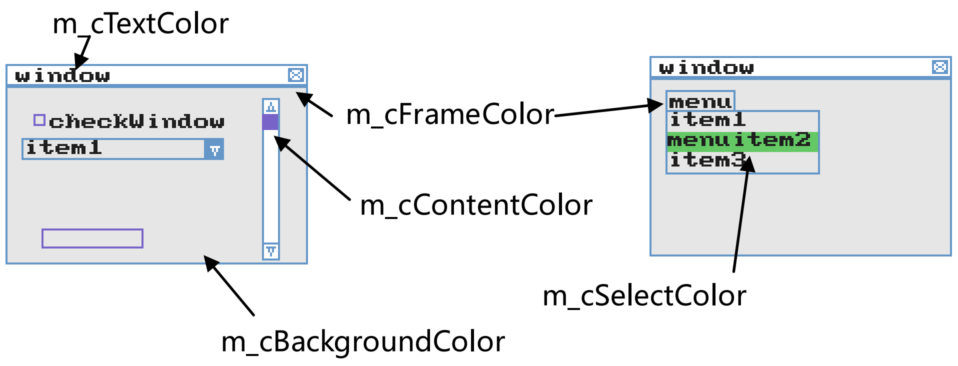


Figure 4 Colors of default EStyle

2.2.3 EBitmap

EBitmap defines the bitmap shown on the screen.

Table 13 Members of EBitmap

Data Members	Function
◆ m_pImageBuffer	Pointer of the bitmap
◆ m_uHeight	Bitmap height
◆ m_uWidth	Bitmap width

EBitmap has three constructors. One of them is

EBitmap ::EBitmap(unsigned char iconNO);

Its parameter iconNO is the index of the default icons. You may use these icons when EG_DEFAULT_ICON is defined.

Note that in the monochrome mode, the height of bitmap is RealHeight/8.

Eg: 16*16 pixels monochrome bitmap

EBitmap bmp(16,2,pBuffer);

16*16 pixels RGB bitmap

EBitmap bmp(16,16,pBuffer);

Table 14 Icon index of default icons

IconNO	monochrome	RGB
0(File)	🗋	🗋
1(Folder)	🗀	📁
2(Open Folder)	🗁	📂
3(Question)	?	❓
4(Right Arrow)	⇨	↷
5(Left Arrow)	⇦	↶
6(Error)	✕	⊗
7(Warning)	⚠	⚠
8(Correct)	✓	📋
9(Save)	💾	💾

2.3.1 EButton

Button is perhaps the most common control in all GUI systems. In EmbeddedGUI library, it can be created easily.

```
EButton(EG_CallbackFunc  cf,EPosition  pos,char*  pstring,unsigned  int
        buttonWidth);
```

The first parameter of its constructor is the pointer of callback function. It will be invoked whenever an event happens on this button.
The fourth parameter is the width of the button.

```
Eg:     void buttonCallback(EControl* control,EEvent* curevent){
            if((curevent->m_uSource==1)&&(curevent->m_uMessage==1))
                BSP_UART_SendString( “info clicked.\0” );
        }
        EButton gbtn (buttonCallback,EPosition(40,70),"Button",48);
```

The button will be created on (40,70). When the left cursor click event happens, it will send “info clicked.\0” to UART.

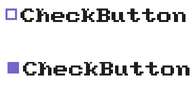
Figure 5 The appearance of EButton

2.3.2 ECheckButton

Unlike button, ECheckButton does not often use the custom callback function. Instead, the library has already defined its default callback function.

```
void EG_CallbackFunc_DefaultECheckButton(EControl* control,EEvent* curevent);
```

The boolean variable m_bCheckButtonChecked of ECheckButton indicates its the checked state. Use GetCheckState() and SetCheckState() to access this protect member.



check == false
check == true

Figure 6 The appearance of ECheckButton

2.3.3 EComboBox

The EComboBox can’ t have more than 6 items at a time, in order to save RAM space. Modify the MaxItem value if necessary.

```
static const int EComboBox::MaxItem=6;
```

After creation, EComboBox does not contain any item. You need to call EComboBox::AddItem() to add the items. This function will return the items index, which is used to identify the items.

Same as the ECheckButton, the default callback function has been predefined.

In this function, m_uSelectItem might be modified. GetSelectIndex() will return the index of chosen item.

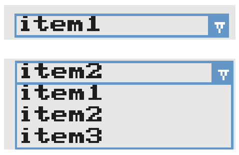


Figure 7 The appearance of EComboBox

2.3.4 EDialogBox

The EDialogBox will show a simple dialog box with two buttons, OK and Cancel. It has four types:

DialogBox Type	
0	No Title String
1	Error
2	Warning
3	Info

GetDialogBoxState() will return the dialog box state:

DialogBox State	
0	Open
1	Exit with OK
2	Exit with Cancel

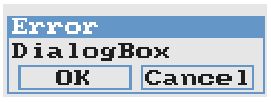


Figure 8 The appearance of EDialogBox

2.3.5 EEdit

You may define edit control as a password edit.

```
EEdit(bool password,EPosition pos,ESize size)
```

Any input character will be shown as '#', if the m_bPassword is true.

Please note that we don't have caret in current library. You need to move your cursor or set the position of the event to the edit to make sure the EEvent will be dispatched to EEdit control.

Figure 9 The appearance of EEdit

2.3.6 EExtLabel

The EExtLabel has a special data member EExtLabel::m_bState.

After clicking on extend label, it will toggle its state. If its state is true, extend label will be shown in SelectColor defined in EStyle.

Figure 10 The appearance of EExtLabel

2.3.7 EIcon

Same as the EButton, you need to define its callback function in most cases. Set the callback function parameter as NULL, if you don't want to handle its events.

Use the utility program EBitmapCreator to create a icon from BMP or JPG files. Please refer to chapter 4.1 for more details.

If the macro EG_DEFAULT_ICON is defined, you can use the default icons in this library. Table 14 lists all the icons you can use.

2.3.8 EIconButton

EIconButton is derived from the EIcon. It has a button string on its right or bottom side.

Eg:

```
EIconButton iconButton1(IconButton1CallbackFunc,EPosition(12,21),
                              ESize(28,29),2,"ib1\0",false);
```

Figure 11 The appearance of EIconButton

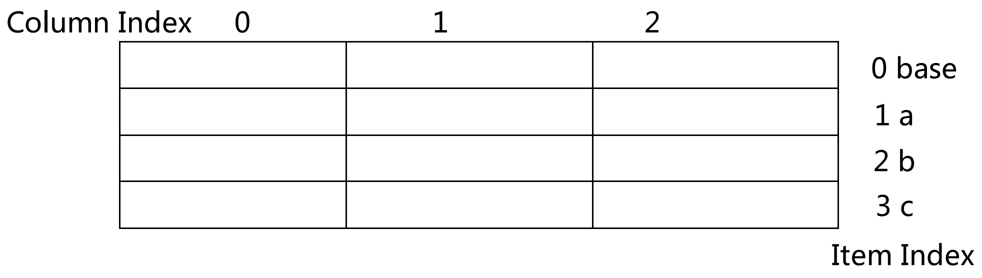
2.3.9 EList

EList has its own data structure EList::EListItem.

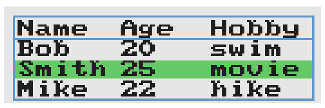
The max column and row numbers have been predefined to save more RAM space.

```
static const unsigned int MaxColumn=5;
static const unsigned int MaxRow=15;
```

The index of column and row is illustrated as follows:



Eg:



```
Name   Age  Hobby
Bob    20   swim
Smith  25   movie
Mike   22   hike
```

```
//global variable
EList gl(EPosition(20,40));
EList::EListItem base,a,b,c;

gw.AddControl(&gl);  //add EList to EWindow
gl.AddColumn(46);    //add three column
gl.AddColumn(40);    // parameter is the column width
gl.AddColumn(47);
gl.AddItem(&base);   // add four items
gl.AddItem(&a);
gl.AddItem(&b);
gl.AddItem(&c);
gl.SetItem(0,0,"Name\0");  // set the item content
gl.SetItem(0,1,"Age\0");
gl.SetItem(0,2,"Hobby\0");
gl.SetItem(1,0,"Bob\0");
gl.SetItem(1,1,"20\0");
gl.SetItem(1,2,"swim\0");
gl.SetItem(2,0,"Smith\0");
gl.SetItem(2,1,"25\0");
gl.SetItem(2,2,"movie\0");
gl.SetItem(3,0,"Mike\0");
gl.SetItem(3,1,"22\0");
gl.SetItem(3,2,"hike\0");
gl.SelectItem(2);   //default selected item
```

2.3.10 EMenu

EMenu and EPopupMenu do not support the overlap menu style.

EMenu(char* pString,unsigned int strLength,EPosition pos,boolupperExtend=false)

The last parameter of EMenu() will determine its pop up direction.

Eg:

```
EMenu gm("menu\0",4,EPosition(20,40));
gw.AddControl(&gm);
gm.AddLabel("item1\0",5,menuitem1CBF);
gm.AddLabel("menuitem2\0",9,menuitem2CBF);
gm.AddLabel("item3\0",5,menuitem3CBF);
```

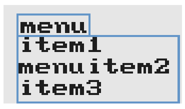


Figure 12 The appearance of EMenu

2.3.11 EMessageBox

EMessageBox provides a simple way to report event.

EMessageBox(unsigned char type,EPosition pos,char* pstring,unsigned char length)

Its type is the same as EDialogBox:

MessageBox Type	
0	No Title String
1	Error
2	Warning
3	Info

Eg:

```
EMessageBox gmb(1,EPosition(15,40),"ErrorMessage\0",12);
```

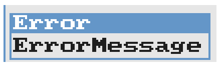


Figure 13 The appearance of EMessageBox

2.3.12 EPopupMenu

Unlike the EMenu, EPopupMenu can only pop up in one direction. EPopupMenu::ShowPopupMenu() and ClosePopupMenu() are used to switch the state of a popup menu.

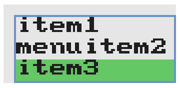


Figure 14 The appearance of EPopupMenu

2.3.13 EProgress

This control can show the progress of any procedure. EProgress::m_uProgress indicates current progress value. Note that this variable is the real progress length of a progress bar. If the width of a progress is 200, it will show 50% ,if m_uProgress is equal to 100.

Eg:

```
EProgress p(10,EPosition(23,73),ESize(60,8));
```

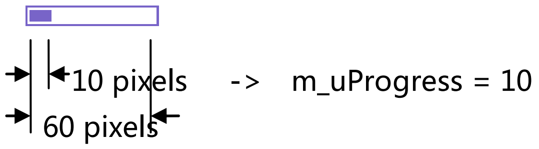


Figure 15 The appearance of EProgress

2.3.14 EScroll

EScroll has two important data members:

```
unsigned int m_uCurPosition;
unsigned int m_uStepValue;
```

m_uCurPosition is current bar position of scroll. Its range is from 8 to the height of EScroll minus 16. m_uStepValue is the step value of bar. Its default value is five. Usually, it should be the height of scroll / 10.

Eg:

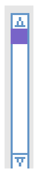
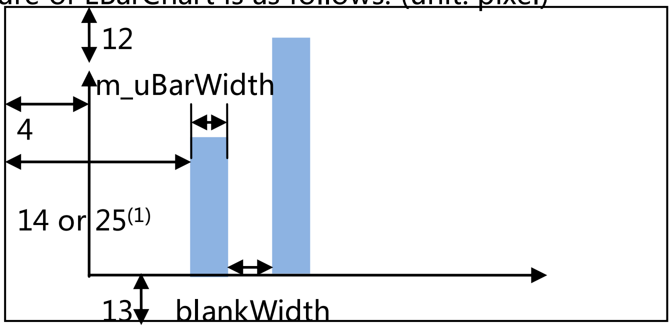


Figure 15 The appearance of EProgress

2.3.15 EBarChart

Same as the EList, EBarChart also has its own data structure EBarChartItem. You may call EBarChart::AddBarItem() to add it. The max number of bar chart items is eight, defined by EBarChart::MaxBarItem.

The structure of EBarChart is as follows: (unit: pixel)



(1) If YAxisRange is equal to 0, this distance is 14 pixels. Otherwise, it is 25 pixels.

Eg:

```
      //global variable
      EBarChart gbc("XAxis\0","YAxis\0",200,105,ESize(130,70),
                              EPosition(20,40),true,16);
      EBarChart::EBarChartItem bci1,bci2,bci3; //Three bar items

#ifdef EG_RGB_COLOR
      bci1.m_cBarColor=EColor(20,100,150);   //define the bar item color
      bci2.m_cBarColor=EColor(20,100,150);
      bci3.m_cBarColor=EColor(20,100,150);
#endif
      bci1.m_pItemText="bar1\0";   //define the bar item string and area
      bci1.m_uBarHeight=30;
      bci1.m_uBarWidth=13;
      bci2.m_pItemText="bar2\0";
      bci2.m_uBarHeight=50;
      bci2.m_uBarWidth=13;
      bci3.m_pItemText="bar3\0";
      bci3.m_uBarHeight=20;
      bci3.m_uBarWidth=13;
      gbc.AddBarItem(&bci1);   // add these item to EBarChart
      gbc.AddBarItem(&bci2);
      gbc.AddBarItem(&bci3);
      gw.AddControl(&gbc);   // add the EBarChart to EWindow
```

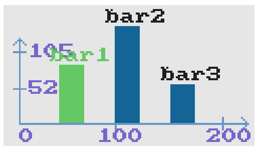


Figure 16　The appearance of EBarChart

Chapter 3 Examples

3.1 EmbeddedGUI based on W78E516 8051 based MCU

You may find the example in .\example\8051

This project is created by Keil C51 uVision2. It only uses the graphic functions and LCD driver parts of this library. All the CPP files have been changed into C files. It is only used to demonstrate the usage of graphic functions. The source code in this project is EmbeddedGUI v0.4.2 ,which should be obsoleted.

3.2 EmbeddedGUI based on LPC2214 ARM7

You may find the example in .\example\arm\LPC2214

This example project is created by Keil uVision3.
The Demo board uses a serial port to communicate with PC. By using this serial port, LPC2214 receives EEvents representing cursor and keyboard events.

In the Timer ISR, you need to invoke the EDesktop:: ScheduleRedraw() to update the LCD.

```
__irq void timerISR(){
        unsigned int* upoint=(unsigned int*)0xE0008000;
    *upoint=0x01;        //    reset the timer1 IR
    upoint=(unsigned int*)0xFFFFF030; //    Vector Address register
    *upoint=0x00;
    desk.ScheduleRedraw();
    }
```

EmbeddedGUI initialization:

```
    //First, initialize the LCD
    EG_LCDInit();
   //Second, Clear the buffer, set the buffer to 0
   //This function could be deleted in order to start faster.
   EG_LCDClearBuffer();
   //Add the controls to window
    introWin.AddControl(&lableCtrl);
    introWin.AddControl(&nextButn);
   //Add the window to desktop
```

```
desk.AddWindow(&introWin);
//If you do not use the cursor, this function could be deleted.
desk.SetCursorShape(EG_DefaultCursorShape);
//Redraw all the windows and controls.
//This function will write the buffer to LCD automatically
desk.GlobalRedraw();
```

3.3 Design controls with MFC

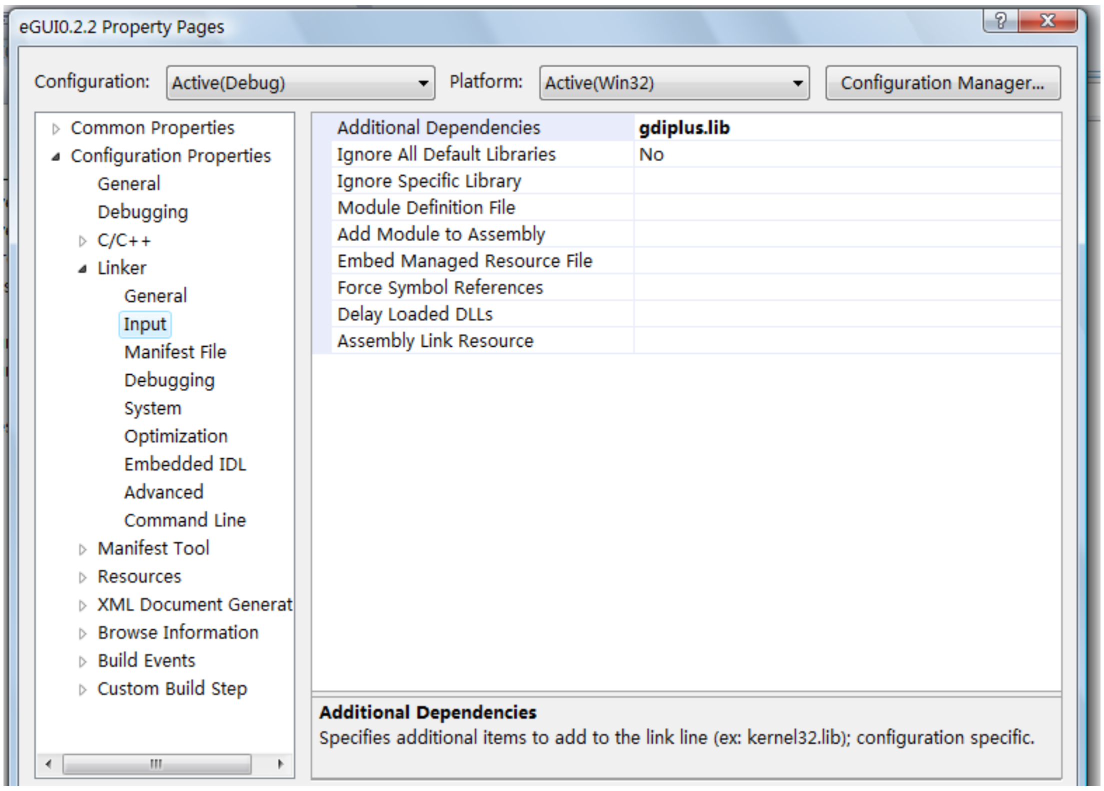
You may find the example in .\example\mfc

Note:

1 Since we use the GDI+ library, include it when linking programe. Add the gdiplus.lib to Additional Dependencies in figure 17.

2 Don’t use the precompiled head file

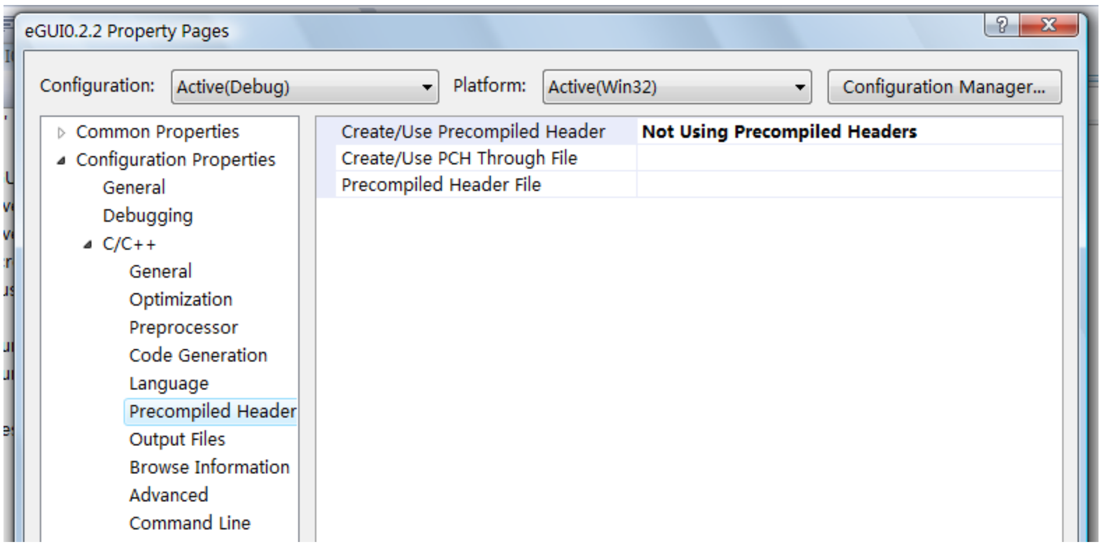
Figure 17 use the GDI+ library



Additional Dependencies	gdiplus.lib
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 18　avoid the precompiled heads


eGUI0.2.2 Property Pages

Configuration: Active(Debug)　Platform: Active(Win32)　Configuration Manager...

Create/Use Precompiled Header	**Not Using Precompiled Headers**
Create/Use PCH Through File	
Precompiled Header File	

3.4　EmbeddedGUI based on LPC2148 ARM7 and FreeRTOS

You may find the example in　.\example\arm\FreeRTOS

This example project is created by Keil uVision3.

The Demo board uses a serial port to communicate with PC. By using this serial port, LPC2148 receives the positions and messages of any event.

In the main function, you need to call the vGUIInit() before start the task scheduler. Eg:

```
//start EmbeddedGUI
vGUIInit();
/* Now all the tasks have been started - start the scheduler.   */
vTaskStartScheduler();
```

3.5　EmbeddedGUI based on S3C2440 ARM920T

You may find the example in　.\example\arm\S3C2440

This example project is created by ADS1.2.

The LCD part of hardware schematic is shown below.

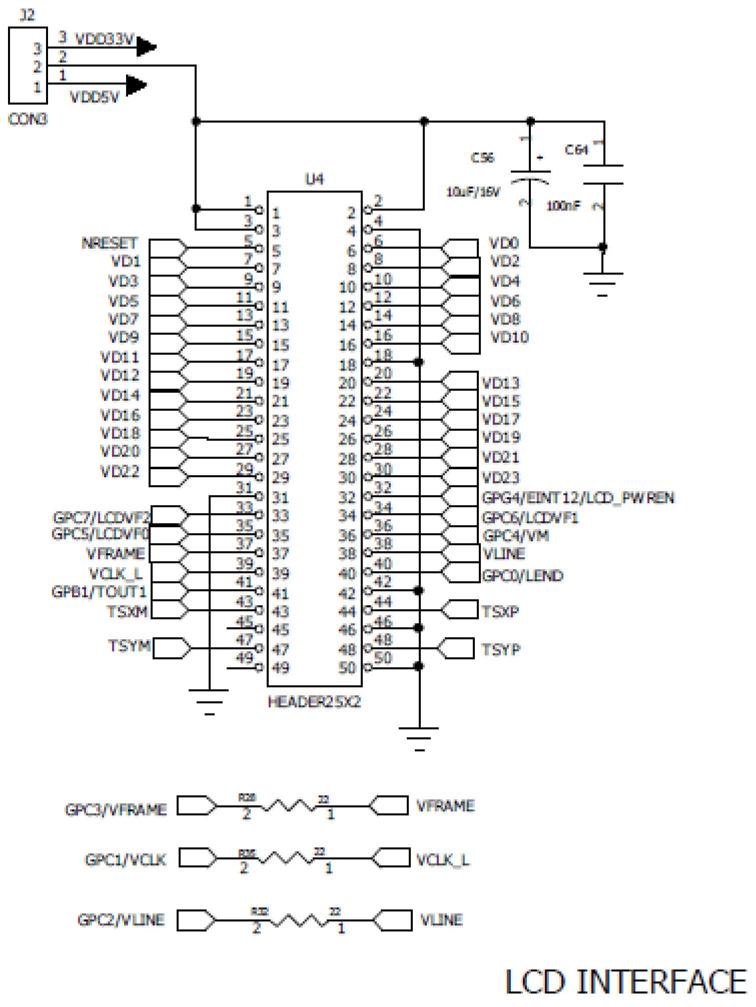


Figure 19 Hardware schematic of LCD interface

Note that the LCD Controller in S3C2440 ARM920T is used to generate TFT signals such as VSYNC and HSYNC. I leave all the SPI interface pins of HX8238A open(Please refer to serial interface block in HX8238 data sheet). This LCD module works in parallel RGB interface(SYNC mode).

Since the TFT signals are generated by S3C2440, any TFT SYNC parallel input mode LCD module which directly connected to S3C2440 can use this LCD driver.

You can use the utility SerialCursor to send EEvents to S3C2440. In the main loop, EG_ReceiveEvent will receive those data packages containing EEvents.

```
while(1){
        desk.ScheduleRedraw();
        delay(4000);
        delay(4000);
        // Serial Cursor function
        if(EG_ReceiveEvent(&event)==0){
                desk.SetCursorEvent(&event);
        }
}
```

Important Notice:

1 When you debug the evaluation board via JTAG, the LCD controller peripheral might not operate properly. According to the S3C2440 data sheet, the clock signal of LCD controller is HCLK. However, in debug state, the system clock signal is driven by Debug CLK from JTAG.

Take ARM7TDMI as a typical example, it is driven by MCLK(Memory CLK) in normal state. In the debug state, on the other hand, ARM7TDMI core is separated with other parts of system and driven by DCLK(Debug CLK). Since DCLK is much slower than MCLK, the system will also run slower.

As to the S3C2440, HCLK is related to DCLK in debug mode. The problem is the timing sequence generated by LCD controller is not accurate any more. The only solution is to download your program to flash and run S3C2440 in normal mode, when using its LCD controller peripheral.

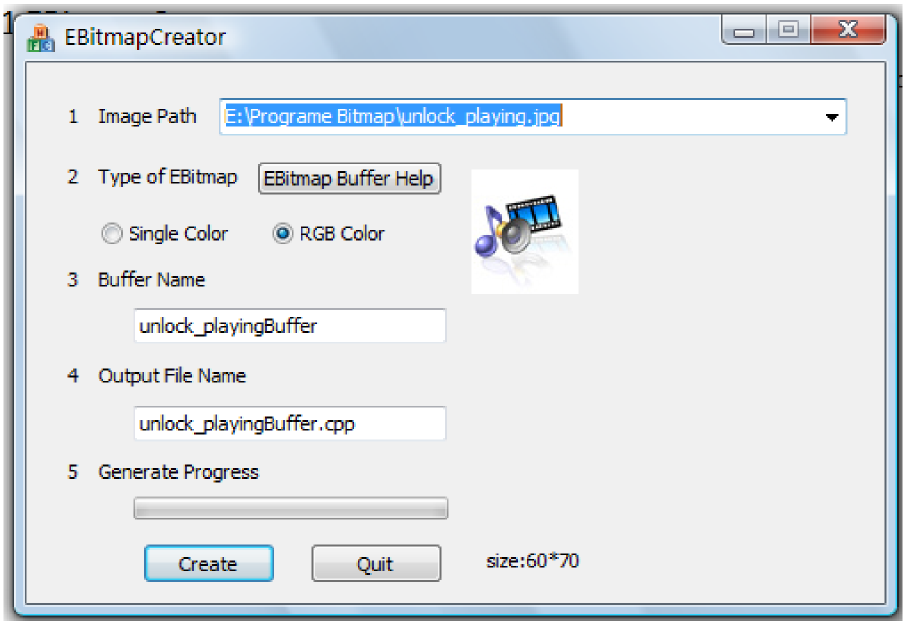
2 You may find that the LCD screen will flash periodically. When the LCD controller is working, it will read the data in LCD buffer via LCD controller DMA and send it to LCD driver IC continuously. This problem is similar to the bug in multi-thread programming. Suppose that you have two threads in a process. The first thread will copy data from a LCD buffer to the destination called driver IC. The second one will draw images to the LCD buffer. When these two threads access the LCD buffer at the same time, dirty data may occur. In this case, you will find that LCD screen flashes whenever GlobalRedraw function is invoked. This solution is quite simple. Pause the LCD controller peripheral whenever global redraw function is executed, which can be simply achieved by setting the ENVID bit in LCDCON1 register to zero.

Chapter 4 Utility Program

4.1 EBitmapCreator

EBitmapCreator is a utility program with which you can create the bitmap buffer from a BMP, JPG or GIF file.

Figure 20 The appearance of EBitmapCreator



1 Image Path: E:\Programe Bitmap\unlock_playing.jpg

2 Type of EBitmap

Single Color [✓ RGB Color]

3 Buffer Name: unlock_playingBuffer

4 Output File Name: unlock_playingBuffer.cpp

5 Generate Progress

size:60*70

After selecting an image, press the create button. This program will generate a CPP file shown below.

```
//unlock_playingBuffer
// size:60*70
static const unsigned char unlock_playingBuffer [12600]={
255,249,239,255,254,243,255,253,244,255,254,243,248,248,238,
254,255,246,252,255,246,246,252,242,244,251,243,251,255,250,
……………};
```

Copy this array to your source code. Declare an EBitmap variable and invoke the image graphic function EG_Graphic_DrawBitmap().

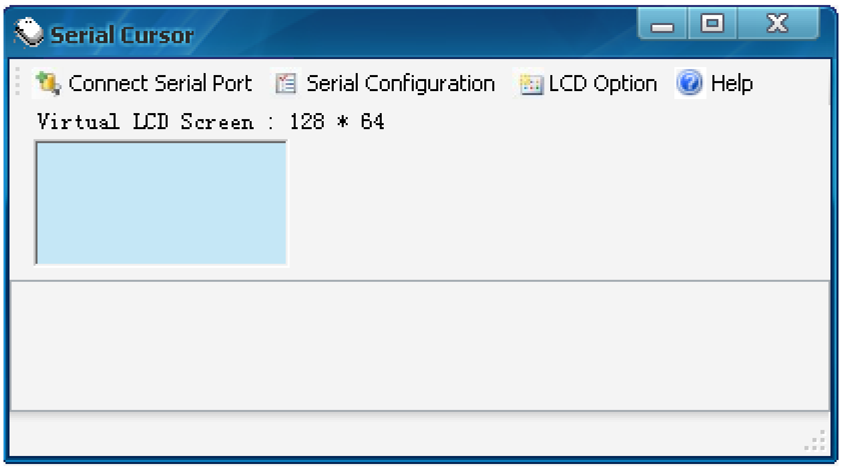
```
EBitmap pbmp(60,70,(unsigned char*) unlock_playingBuffer);
EG_Graphic_DrawBitmap(20,20,&pbmp);
```

4.2 SerialCursor

SerialCursor is a utility program which sends the cursor and keyboard information to your development board through serial port.

Note that SerialCursor requires .NET Framework 3.5 or above in your computer.

Figure 21 The appearance of SerialCursor



Click the LCDOption button to configure the size of your LCD screen first.
Connect the serial port after changing the serial configuration.
The sapphire frame is the virtual LCD screen. When your cursor moves upon it or clicks any button, a data package containing EEvent will be sent.

The data package format is shown below.

Package Head		EEvent				Package End	
0x5A	0xA5	source	message	cursorX	cursorY	0x5F	0xF5
1 byte	1 byte	4 bytes	4 bytes	4 bytes	4 bytes	1 byte	1 byte

Chapter 5　LCD Hardware

5.1　LCD based on KS0107 driver IC

We have tested the LCD driver functions based on a general LCD Module, whose schematic is shown in figure 22. This LCD Module is based on KS0107 and KS0108.

Such driver ICs are compliant with HD61203U and HD61202U.

For more details about the read and write timing sequences, please refer to figure 23 and 24.

Figure 22　LCD Hardware Schematic

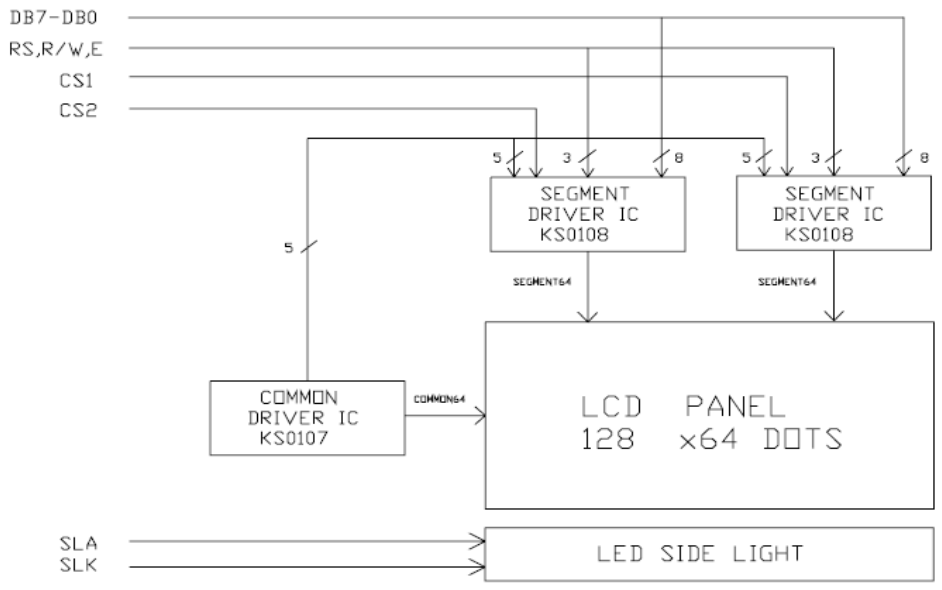
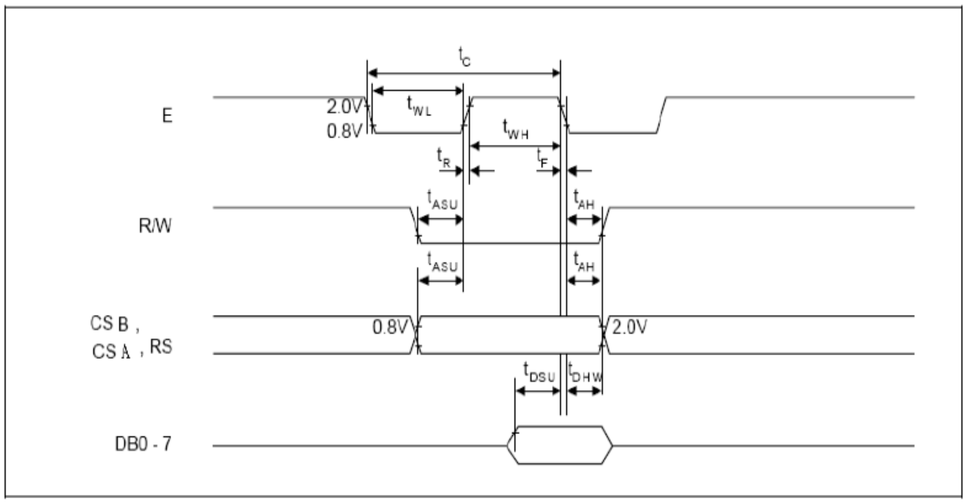
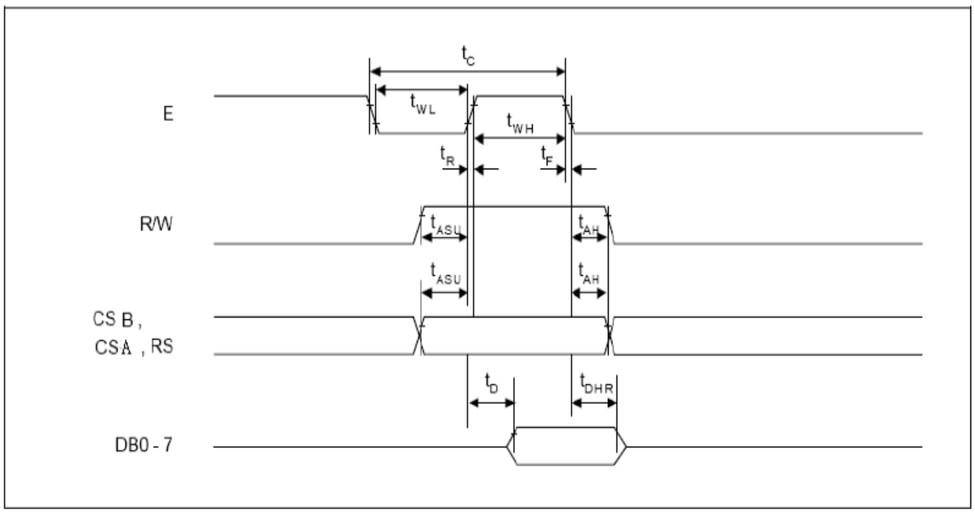


Figure 23 MPU Write Timing



MPU Write Timing

Figure 24 MPU Read Timing



MPU Read Timing

1.3.4 LCD Instruction

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display ON/OFF	0	0	0	0	1	1	1	1	1	0/1	Controls the display on or off. Internal status and display RAM data are not affected. 0:OFF, 1:ON
Set Address	0	0	0	1	Y address (0~63)						Sets the Y address in the Y address counter.
Set Page (X address)	0	0	1	0	1	1	1	Page (0~7)			Sets the X address at the X address register.
Display Start Line	0	0	0	1	Display start line (0~63)						Indicates the display data RAM displayed at the top of the screen.
Status Read	0	1	BUSY	0	ON/OFF	RESET	0	0	0	0	Read status. BUSY 0 : Ready 1: In operation ON/OFF 0 : Display ON 1 : Display OFF RESET 0 : Normal 1 : Reset
Write Display Data	1	0	Write Data								Writes data (DB0:7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read Display Data	1	1	Read Data								Reads data (DB0:7) from display data RAM to the data bus.

Table 15 LCD Instruction

5.2 LCD based on T6963 driver IC

The reading and writing timing sequence of LCD driver is shown as follows. For more information about this LCD module, please refer to the datasheet of T6963, which can be found at /version0.X.X/Driver IC/t6963c.pdf

Figure 25 Timing sequence of T6963

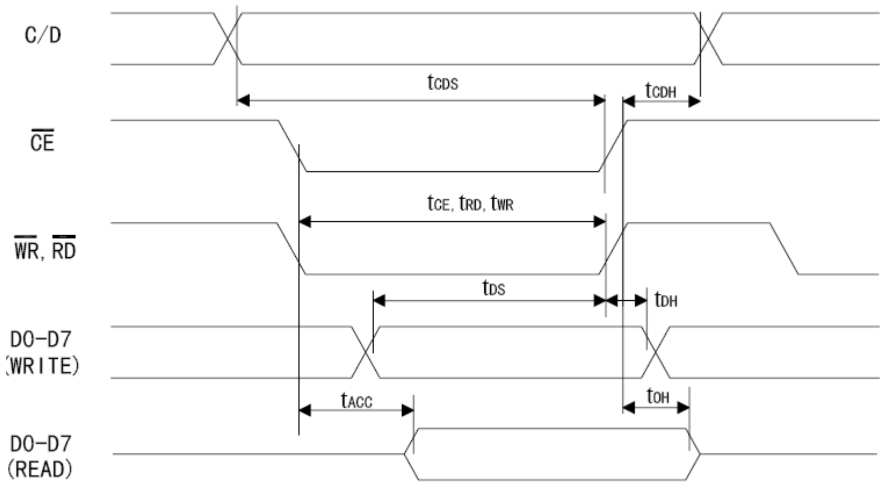


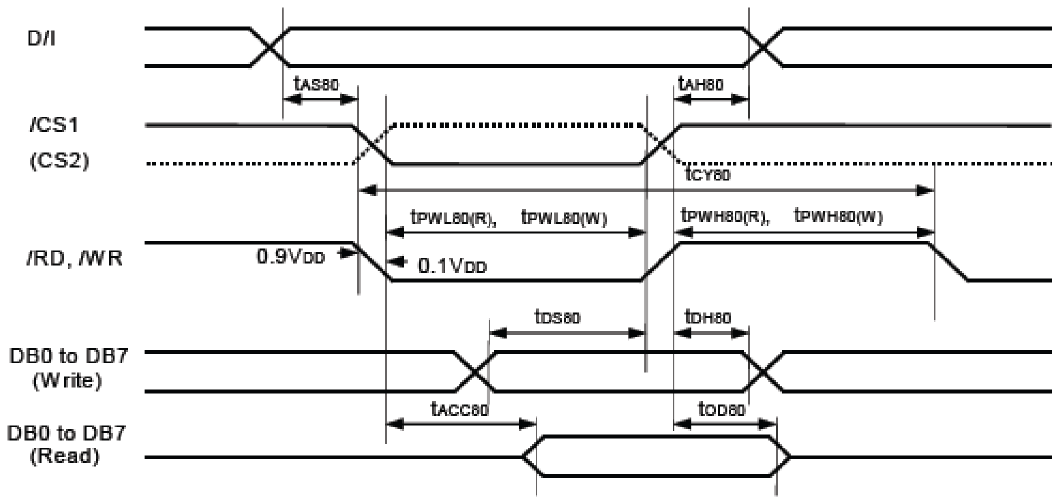
Table 16 Timing parameters of T6963

Timing Parameters	Label	Minimize Value	Maximize Value
C/D establish time	Tcds	100ns	N/A
C/D hold time	Tcdh	10ns	N/A
Chip select time	Tce	80ns	N/A
Data establish time(write)	Tds	80ns	N/A
Data hold time(write)	Tdh	40ns	N/A
Data establish time(read)	Tacc	N/A	150ns
Data hold time(read)	Toh	10ns	50ns

5.3 LCD based on S6B33C driver IC

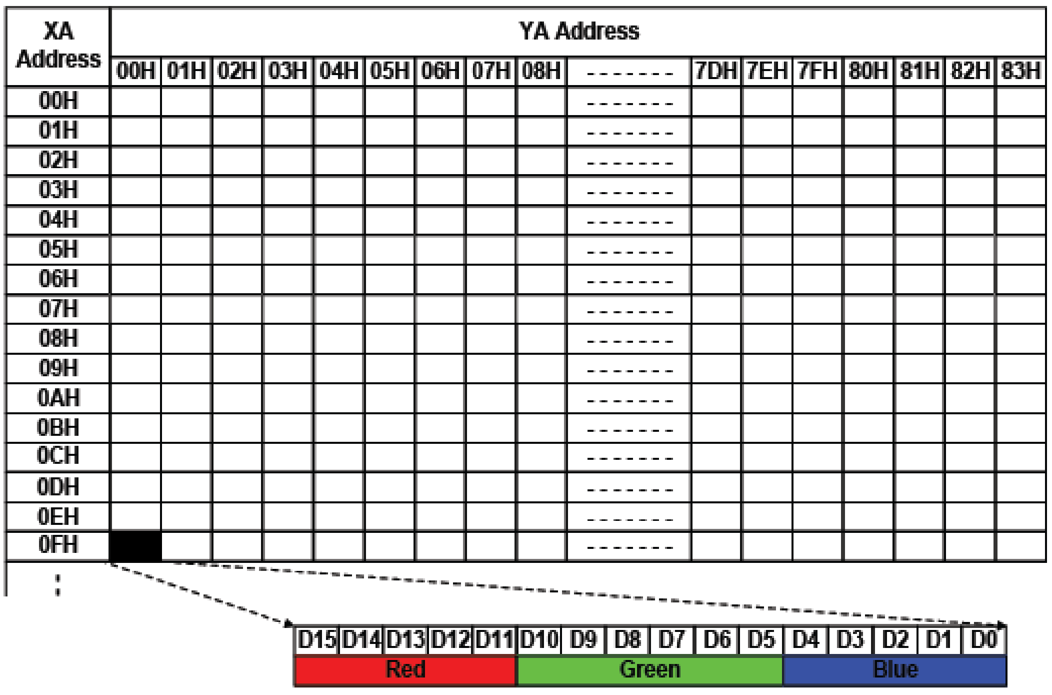
The reading and writing timing sequence of LCD driver is shown as follows. For more information about this LCD Module, please refer to the datasheet of S6B33C, which can be found at SAMSUNG Electronics Website.

Figure 26 Paralle interface timing sequence



** tPWL80(W) and tPWL80(R) is specified in the overlapped period when CS1B is low (CS2 is high) and /WR(/RD) is low.

Figure 27 S6B33C Display RAM Map



XA Address	YA Address																
	00H	01H	02H	03H	04H	05H	06H	07H	08H	-------	7DH	7EH	7FH	80H	81H	82H	83H
00H										-------							
01H										-------							
02H										-------							
03H										-------							
04H										-------							
05H										-------							
06H										-------							
07H										-------							
08H										-------							
09H										-------							
0AH										-------							
0BH										-------							
0CH										-------							
0DH										-------							
0EH										-------							
0FH	■									-------							
:																	

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Red					Green						Blue				

5.4 LCD based on RA8803 driver IC

The reading and writing timing sequence of LCD driver is shown as follows. For more information about this LCD Module, please refer to the datasheet of RA8803, which can be found at /version0.X.X/Driver IC/RA8803.pdf

Figure 28 Timing sequence of RA8803

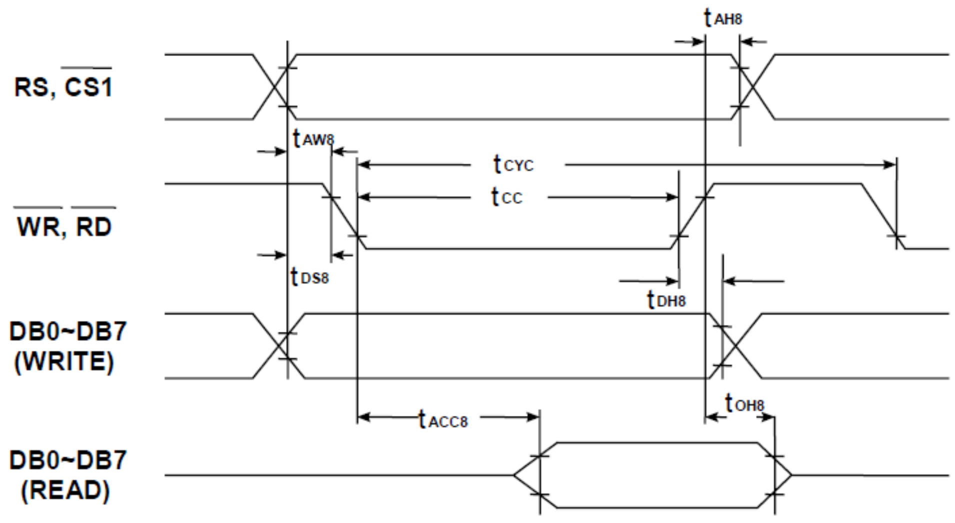


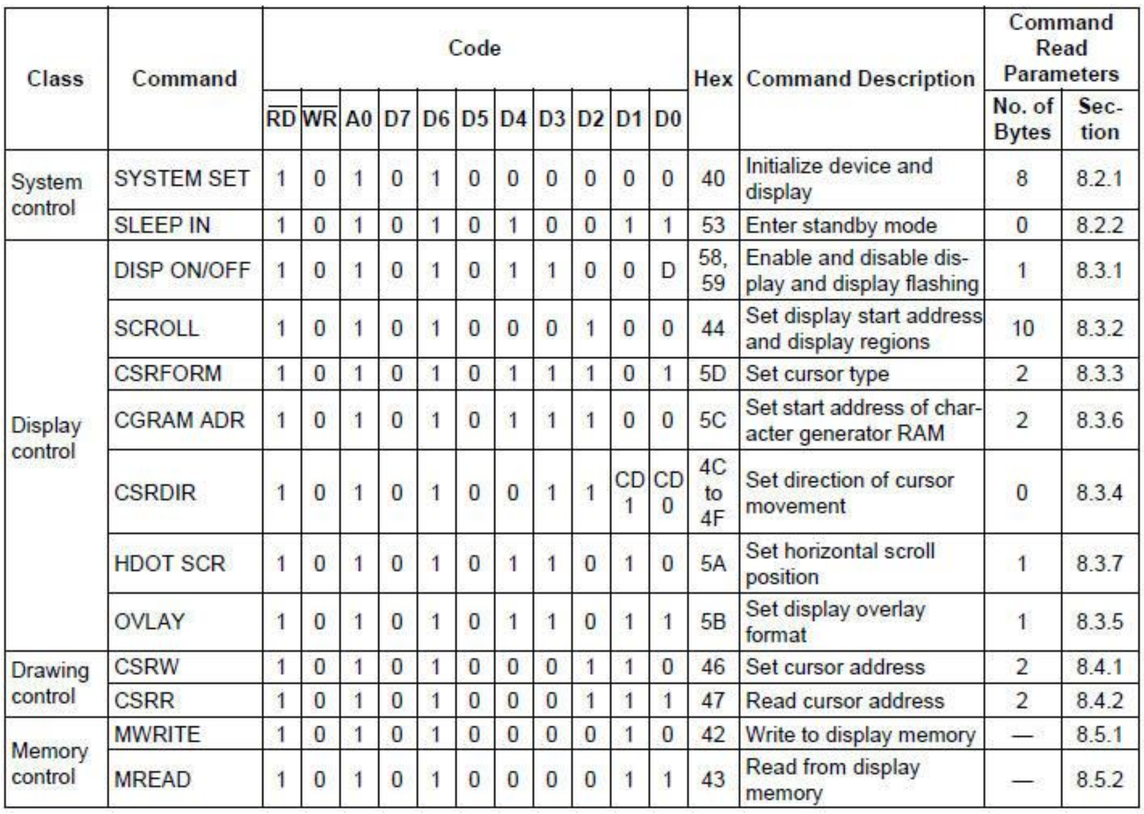
Figure 29 Timing sequence parameters of RA8803

Signal	Symbol	Parameter	Rating		Unit	Condition
			Min	Max		
RS, CS1#	t_{AH8}	Address hold time	10	--	ns	System Clock: 8MHz Voltage: 3.3V
	t_{AW8}	Address setup time	63	--	ns	
WR#, RD#	t_{CYC}	System cycle time	800	--	ns	
	t_{CC}	Strobe pulse width	400	--	ns	
DB0 to DB7	t_{DS8}	Data setup time	63	--	ns	
	t_{DH8}	Data hold time	10	--	ns	
	t_{ACC8}	RD access time	--	330	ns	
	t_{OH8}	Output disable time	10	--	ns	

5.5 LCD based on S1D13305 driver IC

The reading and writing timing sequence of LCD driver is shown as follows. For more information about this LCD Module, please refer to the datasheet of S1D13305, which can be found at /version0.X.X/Driver IC/S1D13305.pdf

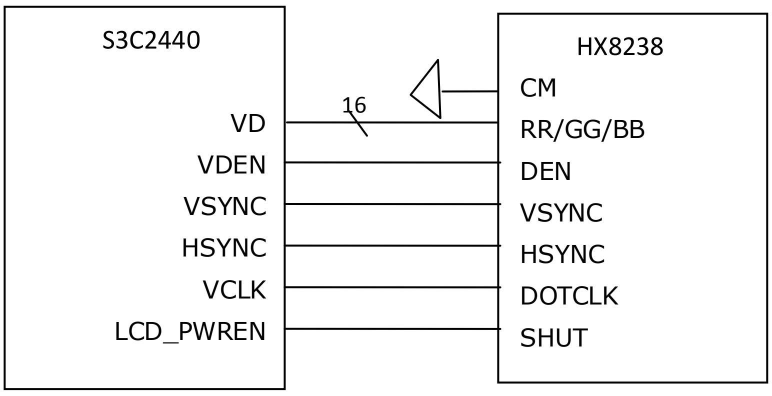
Figure 30 Instruction Set of S1D13305



Class	Command	Code											Hex	Command Description	Command Read Parameters	
		$\overline{RD}$	$\overline{WR}$	A0	D7	D6	D5	D4	D3	D2	D1	D0			No. of Bytes	Sec-tion
System control	SYSTEM SET	1	0	1	0	1	0	0	0	0	0	0	40	Initialize device and display	8	8.2.1
	SLEEP IN	1	0	1	0	1	0	1	0	0	1	1	53	Enter standby mode	0	8.2.2
Display control	DISP ON/OFF	1	0	1	0	1	0	1	1	0	0	D	58, 59	Enable and disable display and display flashing	1	8.3.1
	SCROLL	1	0	1	0	1	0	0	0	1	0	0	44	Set display start address and display regions	10	8.3.2
	CSRFORM	1	0	1	0	1	0	1	1	1	0	1	5D	Set cursor type	2	8.3.3
	CGRAM ADR	1	0	1	0	1	0	1	1	1	0	0	5C	Set start address of character generator RAM	2	8.3.6
	CSRDIR	1	0	1	0	1	0	0	1	1	CD 1	CD 0	4C to 4F	Set direction of cursor movement	0	8.3.4
	HDOT SCR	1	0	1	0	1	0	1	1	0	1	0	5A	Set horizontal scroll position	1	8.3.7
	OVLAY	1	0	1	0	1	0	1	1	0	1	1	5B	Set display overlay format	1	8.3.5
Drawing control	CSRW	1	0	1	0	1	0	0	0	1	1	0	46	Set cursor address	2	8.4.1
	CSRR	1	0	1	0	1	0	0	0	1	1	1	47	Read cursor address	2	8.4.2
Memory control	MWRITE	1	0	1	0	1	0	0	0	0	1	0	42	Write to display memory	—	8.5.1
	MREAD	1	0	1	0	1	0	0	0	0	1	1	43	Read from display memory	—	8.5.2

5.6 LCD based on LCD controller peripheral of S3C2440

Since the TFT signals are generated by S3C2440, any TFT SYNC parallel input mode LCD module which is directly connected to S3C2440 can use this LCD driver.



The data transaction timing sequence of LCD driver is shown as follows.

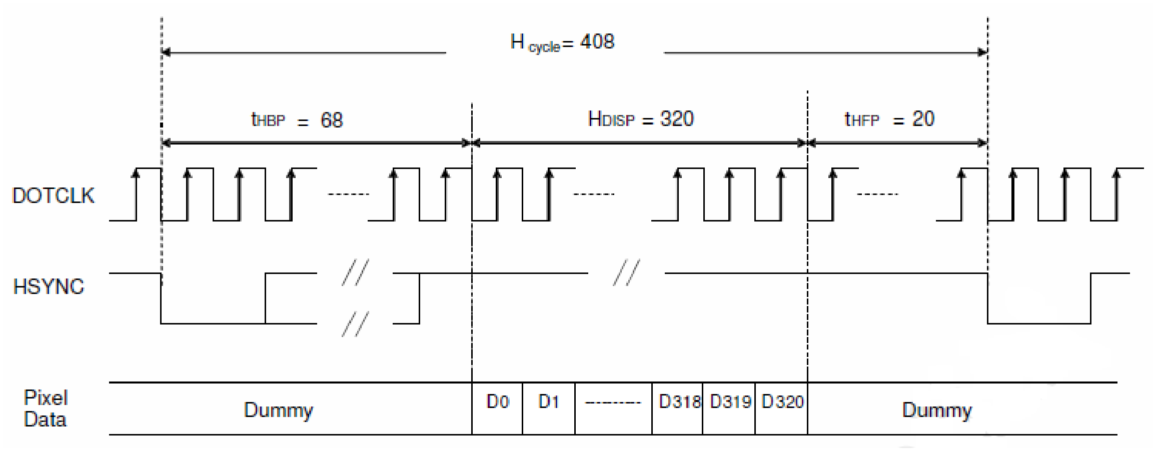


Figure 31 Horizontal Data Transaction Timing

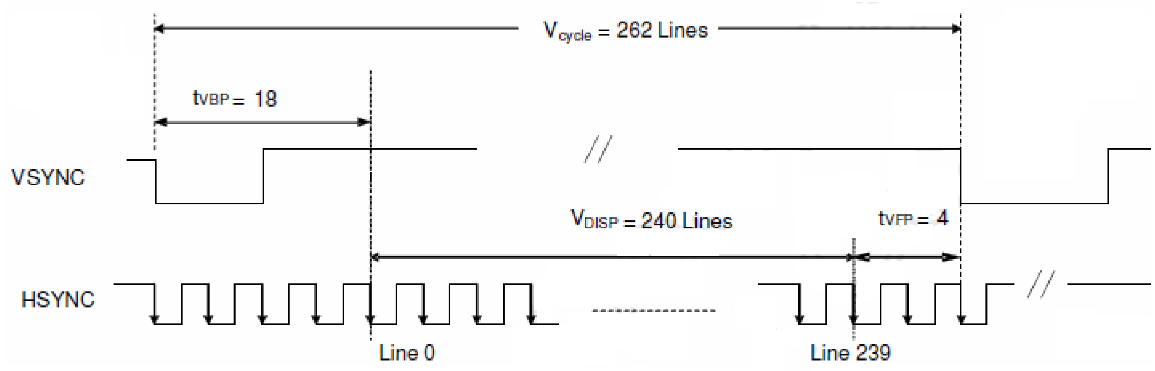


Figure 32 Vertical Data Transaction Timing