# 3.2inch 320x240 Touch LCD (C) User Manual

## Key Parameters

| | |
|---|---|
| LCD Controller | ILI9325 |
| Touch Screen Controller | XPT2046 |
| LCD Type | TFT |
| LCD Interface | 16-bit parallel |
| Touch Screen Interface | SPI |
| Backlight | LED |
| Colors | 65536 |
| Resolution | 320*240 DOTS |

## Contents

3.2inch 320x240 Touch LCD (C) User Manual....1
Key Parameters....1
1. Hardware Resources....2
1.1 ILI9325....2
1.2 XPT2046....4
2. LCD Pin Description....5
3. Example Analysis....6

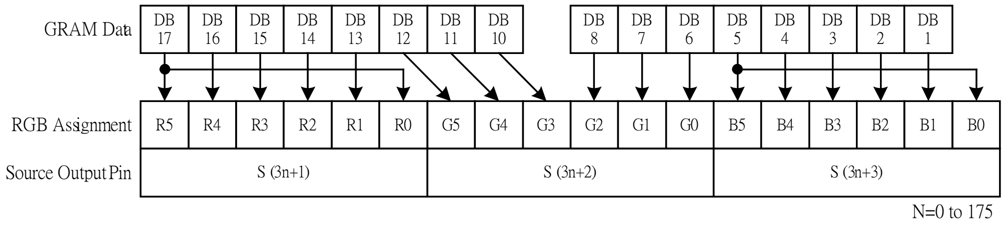
# 1. Hardware Resources

## 1.1 ILI9325

- ILI9325C is a 262,144-color one-chip SoC driver for a-TFT liquid crystal display with resolution of 240RGBx320 dots, comprising a 720-channel source driver, a 320-channel gate driver, 172,800 bytes RAM for graphic data of 240RGBx320 dots, and power supply circuit.
- ILI9325C has five kinds of system interfaces which are i80-system MPU interface (8-/9-/16-/18-bit bus width), VSYNC interface (system interface + VSYNC, internal clock, DB[17:0]), serial data transfer interface (SPI), RGB 6-/16-/18-bit interface (DOTCLK, VSYNC, HSYNC, ENABLE, DB[17:0]).

  The following figure shows correspondence between 18-bit RGB Assignment and 16-Bit GRAM.

*i80/M68 system 16-bit data bus interface*



You can see from figure, the useful data bus interfaces of ILI9325 under 16-bit mode are: D17~D10 and D8~D1. D9 and D0 are unused. Actually, D9 and D0 of ILI9341 are not welded in this LCD module. D17~D10 and D8~D1 of ILI9325 correspond to D15~D0 of the MCU. The lower five bits of the 16-bit MCU data indicate blue. The middle six bits indicate green. The higher five bits indicate red. When the value bigger, the color deeper.

**Important Register Introduction**

Please see ILI9325 datasheet for more details about ILI9325. Here are just some important register introduction.

**Entry Mode (R03h)**

| R/W | RS |
|---|---|
| W | 1 |
| Default | |

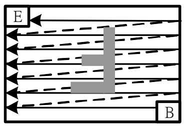
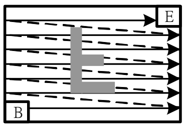
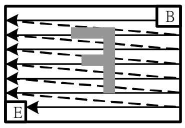
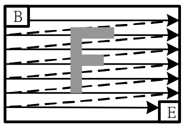
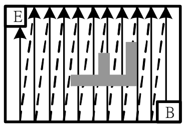
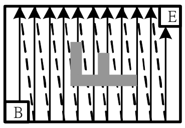
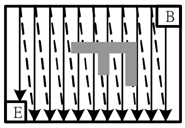
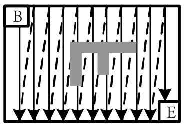
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TRI | DFM | 0 | BGR | 0 | 0 | 0 | 0 | ORG | 0 | I/D1 | I/D0 | AM | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**AM** Control the GRAM update direction.

When AM = “0”, the address is updated in horizontal writing direction.

When AM = “1”, the address is updated in vertical writing direction.

**I/D[1:0]** Control the address counter (AC) to automatically increase or decrease by 1 when update one pixel display data. Refer to the following figure for the details.

| | I/D[1:0] = 00<br>Horizontal : decrement<br>Vertical : decrement | I/D[1:0] = 01<br>Horizontal : increment<br>Vertical : decrement | I/D[1:0] = 10<br>Horizontal : decrement<br>Vertical : increment | I/D[1:0] = 11<br>Horizontal : increment<br>Vertical : increment |
|---|---|---|---|---|
| AM = 0<br>Horizontal |  |  |  |  |
| AM = 1<br>Vertical |  |  |  |  |

**ORG** Moves the origin address according to the ID setting when a window address area is made. This function is enabled when writing data with the window address area using high-speed RAM write.

ORG = “0”: The origin address is not moved. In this case, specify the address to start write operation according to the GRAM address map within the window address area.

ORG = “1”: The original address “00000h” moves according to the I/D[1:0] setting.

**BGR** Swap the R and B order of written data.

BGR=”0”: Follow the RGB order to write the pixel data.

BGR=”1”: Swap the RGB data to BGR in writing into GRAM.

**GRAM Horizontal/Vertical Address Set (R20h, R21h)**

| R/W | RS | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD7 | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 |
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD16 | AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 |
| Default | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**AD[16:0]** Set the initial value of address counter (AC).

The address counter (AC) is automatically updated in accordance to the setting of the AM, I/D bits as data is written to the internal GRAM. The address counter is not automatically updated when read data from the internal GRAM.

| AD[16:0] | GRAM Data Map |
|---|---|
| 17’h00000 ~ 17’h000EF | 1st line GRAM Data |
| 17’h00100 ~ 17’h001EF | 2nd line GRAM Data |
| 17’h00200 ~ 17’h002EF | 3rd line GRAM Data |
| 17’h00300 ~ 17’h003EF | 4th line GRAM Data |
| | |
| 17’h13D00 ~ 17’ h13DEF | 318th line GRAM Data |
| 17’h13E00 ~ 17’ h13EEF | 319th line GRAM Data |
| 17’h13F00 ~ 17’h13FEF | 320th line GRAM Data |

**Write Data to GRAM (R22h)**

| R/W | RS |
|---|---|
| W | 1 |

| D17 | D16 | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RAM write data (WD[17:0], the DB[17:0] pin assignment differs for each interface. | | | | | | | | | | | | | | | | | |

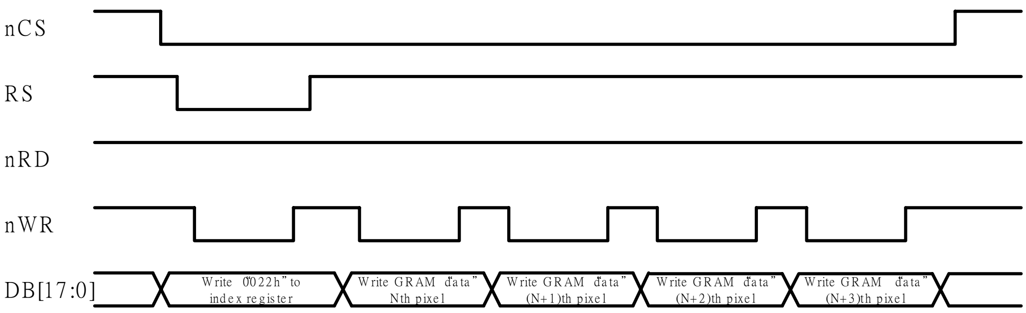
This register is the GRAM access port. When update the display data through this register, the address counter (AC) is increased/decreased automatically.

### GRAM Address Map & Read/Write

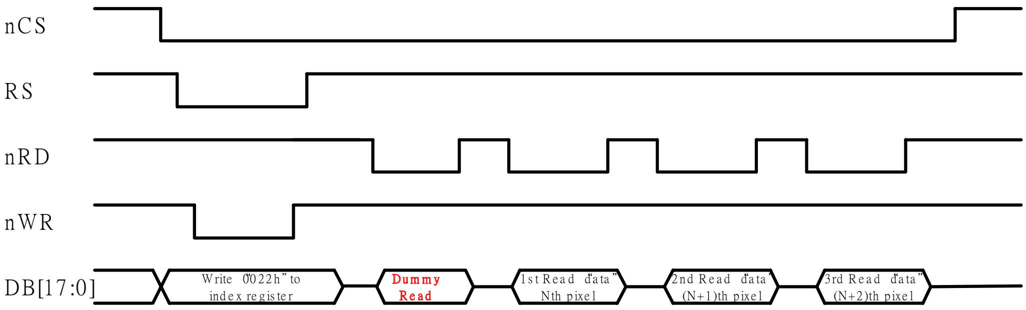
ILI9325C has an internal graphics RAM (GRAM) of 172,800 bytes to store the display data and one pixel is constructed of 18 bits. The GRAM can be accessed through the i80 system, SPI and RGB interfaces.

*i80 18-/16-bit System Bus Interface Timing*

(a) Write to GRAM


nCS
RS
nRD
nWR
DB[17:0]
Write "0022h" to index register
Write GRAM "data" Nth pixel
Write GRAM "data" (N+1)th pixel
Write GRAM "data" (N+2)th pixel
Write GRAM "data" (N+3)th pixel

(b) Read from GRAM


nCS
RS
nRD
nWR
DB[17:0]
Write "0022h" to index register
Dummy Read
1st Read "data" Nth pixel
2nd Read "data" (N+1)th pixel
3rd Read "data" (N+2)th pixel

## 1.2 XPT2046

- The XPT2046 is a 4-wire resistive touch screen controller that incorporates a 12-bit 125 kHz sampling SAR typeA/D converter.
- The XPT2046 supports digital I/O interface voltage from 1.5V to VCC in order to connect low voltage uP.
- The XPT2046 can detect the pressed screen location by performing two A/D conversions. In addition to location, the XPT2046 also measures touch screen pressure.On-chip VREF can be utilized for analog auxiliary input, temperature measurement and battery monitoring withthe ability to measure voltage from 0V to 5V.

- The XPT2046 also has an on-chip temperature sensor.
- The XPT2046 is available in 16pin QFN thin package(0.75mm in height) and has the operating temperature range of -40°C to +85°C

## 2. LCD Pin Description

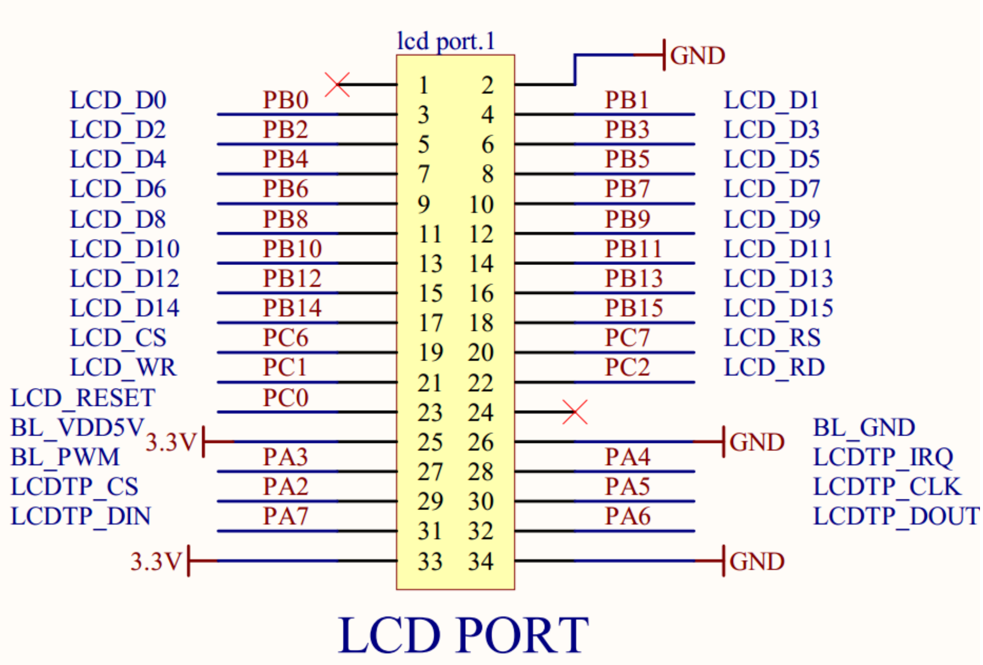
| PIN NO. | SYMBOL | DESCRIPTION | FUNCTION |
|---|---|---|---|
| 1 | 5V | 5V power supply | When powered from 5V supply,<br>Pin 1 & Pin 2 as power input,<br>Pin 33 & Pin 34 provide 3.3V output. |
| 2 | GND | Ground | GND |
| 3 | D0 | Data pin | D0-D15 |
| 4 | D1 | | |
| 5 | D2 | | |
| 6 | D3 | | |
| 7 | D4 | | |
| 8 | D5 | | |
| 9 | D6 | | |
| 10 | D7 | | |
| 11 | D8 | | |
| 12 | D9 | | |
| 13 | D10 | | |
| 14 | D11 | | |
| 15 | D12 | | |
| 16 | D13 | | |
| 17 | D14 | | |
| 18 | D15 | | |
| 19 | CS | LCD chip select | Low active |
| 20 | RS | Instruction/Data register selection | RS = 1 : Data Register<br>RS = 0 : Instruction Register |
| 21 | WR | Write | WR = 0， RD = 1 |
| 22 | RD | Read | WR = 1， RD = 0 |
| 23 | RESET | Reset the controller chip | Low active |
| 24 | NC | Not connect | Not connect |
| 25 | BLVCC | 5V or 3.3V | Backlight VCC |
| 26 | BLGND | Ground | Backlight GND |
| 27 | BLCNT | Backlight brightness adjustment | Control the backlight brightness via PWM |
| 28 | TP_IRQ | Touch screen interrupt | Low level while the touch screen detects pressing |

| | | | |
|---|---|---|---|
| 29 | TP_CS | Touch screen chip select | Low active |
| 30 | TP_SCK | Touch screen SPI clock | connects to SPI SCK |
| 31 | TP_SI | Touch screen data input | connects to SPI MOSI |
| 32 | TP_SO | Touch screen data output | connects to SPI MISO |
| 33 | 3.3V | 3.3V power supply | When powered from 3.3V supply,<br>Pin 33 & Pin 34 as power input,<br>Pin 1 & Pin 2 keep NC. |
| 34 | GND | Ground | |

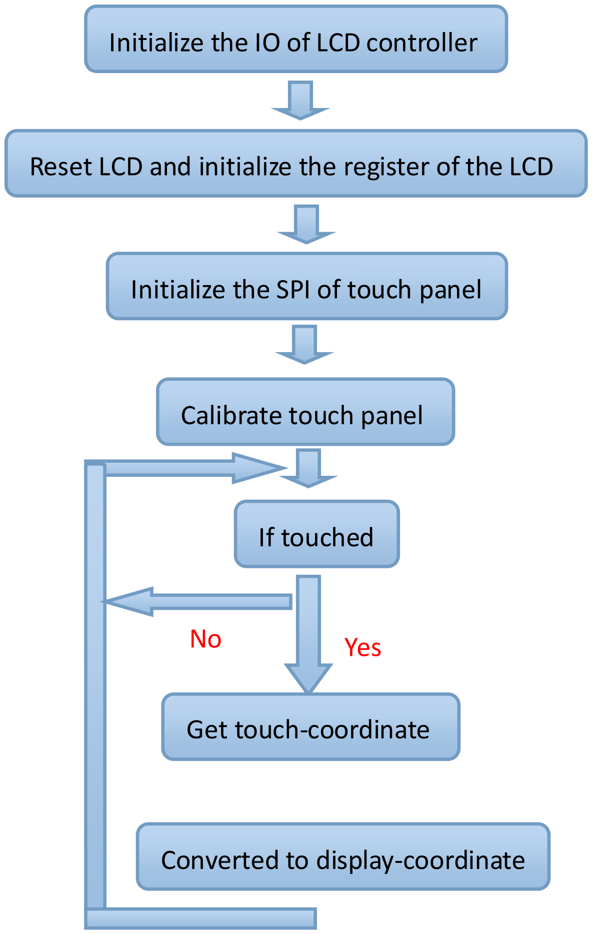
## 3. Example Analysis

We use STM32 development board (MCU STM32F103RCT6 onboard) to describe how to use the LCD. You can use the LCD with other similar products.

The following figure is a schematic of the LCD Port of the development board.


lcd port.1
GND
LCD_D0 PB0 1 2 PB1 LCD_D1
LCD_D2 PB2 3 4 PB3 LCD_D3
LCD_D4 PB4 5 6 PB5 LCD_D5
LCD_D6 PB6 7 8 PB7 LCD_D7
LCD_D8 PB8 9 10 PB9 LCD_D9
LCD_D10 PB10 11 12 PB11 LCD_D11
LCD_D12 PB12 13 14 PB13 LCD_D13
LCD_D14 PB14 15 16 PB15 LCD_D15
LCD_CS PC6 17 18 PC7 LCD_RS
LCD_WR PC1 19 20 PC2 LCD_RD
LCD_RESET PC0 21 22
23 24
BL_VDD5V 3.3V 25 26 GND BL_GND
BL_PWM PA3 27 28 PA4 LCDTP_IRQ
LCDTP_CS PA2 29 30 PA5 LCDTP_CLK
LCDTP_DIN PA7 31 32 PA6 LCDTP_DOUT
3.3V 33 34 GND
LCD PORT

LCD Demo Procedure


Initialize the IO of LCD controller
Reset LCD and initialize the register of the LCD
Initialize the SPI of touch panel
Calibrate touch panel
If touched
No
Yes
Get touch-coordinate
Converted to display-coordinate

### Source Code Analysis

```
/* The following macro defines image rotation. */
//#define DISP_ORIENTATION                        0
//#define DISP_ORIENTATION                        90
//#define DISP_ORIENTATION                        180
#define DISP_ORIENTATION                          270
#define Set_Cs       GPIO_SetBits(GPIOC, GPIO_Pin_6);    //CS=1；
#define Clr_Cs       GPIO_ResetBits(GPIOC, GPIO_Pin_6);  //CS=0；

#define Set_Rs       GPIO_SetBits(GPIOC, GPIO_Pin_7);    //RS=1；
#define Clr_Rs       GPIO_ResetBits(GPIOC, GPIO_Pin_7);  //RS=0；

#define Set_nWr      GPIO_SetBits(GPIOC, GPIO_Pin_1);    //WR=1；
#define Clr_nWr      GPIO_ResetBits(GPIOC, GPIO_Pin_1);  //WR=0；

#define Set_nRd      GPIO_SetBits(GPIOC, GPIO_Pin_2);    //RD=1；
#define Clr_nRd      GPIO_ResetBits(GPIOC, GPIO_Pin_2);  // RD=0；
/* Command writing function */
__inline void LCD_WriteIndex(uint16_t index)
{
   Clr_Rs;           //RS=0
   Set_nRd;          //RD=0
   LCD_Delay(0);      //Delay
   GPIOB->ODR = index; /* index is an command’s address*/
   LCD_Delay(0);      //Delay
   Clr_nWr;          //WR=0
   Set_nWr;          //WR=1
}
/* Data writing function */
__inline void LCD_WriteData(uint16_t data)
{
   Set_Rs;            //RS=1
   LCD_Delay(0);       //Delay
   GPIOB->ODR = data;   /* Data writing address*/
   LCD_Delay(0);      //Delay
   Clr_nWr;          //WR=0
   Set_nWr;          //WR=1
}
/* Data reading function */
__inline uint16_t LCD_ReadData(void)
{
   uint16_t value;
   Set_Rs;
```

```
    Set_nWr;
    Clr_nRd;
    GPIOB->CRH = 0x44444444;   //Set PB0-PB15 as input pin
    GPIOB->CRL = 0x44444444;
    value = GPIOB->IDR;        //Reads data
    GPIOB->CRH = 0x33333333;   //Set PB0-PB15 as output pin
    GPIOB->CRL = 0x33333333;
    Set_nRd;
    return value;
}
/*****************************************************************
**************
Write data to a specified address, LCD_Reg indicates register address
while LCD_RegValue indicates register value.
******************************************************************
*************/
__inline void LCD_WriteReg(uint16_t LCD_Reg,uint16_t LCD_RegValue)
{
Clr_Cs;
LCD_WriteIndex(LCD_Reg);         //Writing command, LCD_Reg is an
address to be written in.
LCD_WriteData(LCD_RegValue);   //Writes data.
Set_Cs;
}

/*****************************************************************
**************
Read data from a specified address, LCD_Reg indicates register address.
This function will return a value from the address.
******************************************************************
*************/
__inline uint16_t LCD_ReadReg(uint16_t LCD_Reg)
{
uint16_t LCD_RAM;
Clr_Cs;
LCD_WriteIndex(LCD_Reg);    //Writing command, LCD_Reg is an address
to be read from.
LCD_RAM = LCD_ReadData();  //Reads data
Set_Cs;
return LCD_RAM;
}
//That's the basic read-and-write functions by IO emulation. If you
want use FSMC from STM32 to control the LCD, you can read another demo
LCD + TouchPanel(8080 FSMC)
```

```
/****************************************************************
**************
This is LCD initialization function. The initialization value of the
LCD is provided by the factory. So usually you can copy them directly
to initialize LCD. Please refer to ILI9325 datasheets for more details.
****************************************************************
*************/
void LCD_Initializtion(void)
{
  uint16_t DeviceCode;
  LCD_Configuration();                  //LCD Initialization
  GPIO_ResetBits(GPIOC, GPIO_Pin_0);   /* LCD reset*/
  delay_ms(100);
  GPIO_SetBits(GPIOC, GPIO_Pin_0);
  GPIO_SetBits(GPIOA, GPIO_Pin_3);     /* Enable back light */
  DeviceCode = LCD_ReadReg(0x0000);    /* Reads ID */
  if( DeviceCode == 0x9325 || DeviceCode == 0x9328 )
  {
    LCD_WriteReg(0x00e7,0x0010);
    LCD_WriteReg(0x0000,0x0001);
    LCD_WriteReg(0x0001,(0<<10)|(1<<8));
    LCD_WriteReg(0x0002,0x0700);
    #if (DISP_ORIENTATION == 0)
    LCD_WriteReg(0x0003,(1<<12)|(1<<5)|(1<<4)|(0<<3));
    #elif (DISP_ORIENTATION == 90)
    LCD_WriteReg(0x0003,(1<<12)|(0<<5)|(1<<4)|(1<<3));
    #elif (DISP_ORIENTATION == 180)
    LCD_WriteReg(0x0003,(1<<12)|(0<<5)|(0<<4)|(0<<3));
    #elif (DISP_ORIENTATION == 270)
    LCD_WriteReg(0x0003,(1<<12)|(1<<5)|(0<<4)|(1<<3));
    #endif
    LCD_WriteReg(0x0004,0x0000);
    LCD_WriteReg(0x0008,0x0207);
    LCD_WriteReg(0x0009,0x0000);
    LCD_WriteReg(0x000a,0x0000);
    LCD_WriteReg(0x000c,0x0001);
    LCD_WriteReg(0x000d,0x0000);
    LCD_WriteReg(0x000f,0x0000);
    /* Power On sequence */
    LCD_WriteReg(0x0010,0x0000);
    LCD_WriteReg(0x0011,0x0007);
    LCD_WriteReg(0x0012,0x0000);
    LCD_WriteReg(0x0013,0x0000);
    delay_ms(50);  /* delay 50 ms */
    LCD_WriteReg(0x0010,0x1590);
    LCD_WriteReg(0x0011,0x0227);
    delay_ms(50);  /* delay 50 ms */
    LCD_WriteReg(0x0012,0x009c);
    delay_ms(50);  /* delay 50 ms */
    LCD_WriteReg(0x0013,0x1900);
    LCD_WriteReg(0x0029,0x0023);
    LCD_WriteReg(0x002b,0x000e);
    delay_ms(50);  /* delay 50 ms */
    delay_ms(50);  /* delay 50 ms */
    LCD_WriteReg(0x0030,0x0007);
    LCD_WriteReg(0x0031,0x0707);
    LCD_WriteReg(0x0032,0x0006);
    LCD_WriteReg(0x0035,0x0704);
    LCD_WriteReg(0x0036,0x1f04);
    LCD_WriteReg(0x0037,0x0004);
    LCD_WriteReg(0x0038,0x0000);
    LCD_WriteReg(0x0039,0x0706);
    LCD_WriteReg(0x003c,0x0701);
    LCD_WriteReg(0x003d,0x000f);
    delay_ms(50);  /* delay 50 ms */
    LCD_WriteReg(0x0050,0x0000);
    LCD_WriteReg(0x0051,0x00ef);
    LCD_WriteReg(0x0052,0x0000);
    LCD_WriteReg(0x0053,0x013f);
    LCD_WriteReg(0x0060,0xa700);
    LCD_WriteReg(0x0061,0x0001);
    LCD_WriteReg(0x006a,0x0000);
    LCD_WriteReg(0x0080,0x0000);
    LCD_WriteReg(0x0081,0x0000);
    LCD_WriteReg(0x0082,0x0000);
    LCD_WriteReg(0x0083,0x0000);
    LCD_WriteReg(0x0084,0x0000);
    LCD_WriteReg(0x0085,0x0000);
    LCD_WriteReg(0x0090,0x0010);
    LCD_WriteReg(0x0092,0x0600);
    LCD_WriteReg(0x0093,0x0003);
    LCD_WriteReg(0x0095,0x0110);
    LCD_WriteReg(0x0097,0x0000);
    LCD_WriteReg(0x0098,0x0000);
    /* display on sequence */
    LCD_WriteReg(0x0007,0x0133);
  }
  delay_ms(50);
```

```
}
/****************************************************************
**************
Set window coordinate.
*****************************************************************
************/
static void LCD_SetCursor( uint16_t Xpos, uint16_t Ypos )
{
  uint16_t temp;
  #if (DISP_ORIENTATION == 0)
  #elif (DISP_ORIENTATION == 90)
    temp = Xpos;
    Xpos =Ypos;
    Ypos = MAX_X - 1 - temp;
  #elif (DISP_ORIENTATION == 180)
    Xpos = MAX_X - 1 - Xpos;
    Ypos = MAX_Y - 1 - Ypos;
  #elif (DISP_ORIENTATION == 270)
    temp = Ypos;
    Ypos = Xpos;
    Xpos = MAX_Y - 1 - temp;
  #endif
  LCD_WriteReg(0x0020, Xpos); // Sets the horizontal position X
  LCD_WriteReg(0x0021, Ypos); // Sets the vertical position Y
  }
/****************************************************************
**************
Clear the screen and fill it with one color.

*****************************************************************
************/
void LCD_Clear(uint16_t Color)
{
  uint32_t index=0;
  LCD_SetCursor(0,0);   //Set cursor coordinate X, Y
  Clr_Cs;
  LCD_WriteIndex(0x0022);//Start to write data into GRAM
  for( index = 0; index < MAX_X * MAX_Y; index++ )
  {
    LCD_WriteData(Color);
  }
  Set_Cs;
}
```

```
int main(void)
{
  //Delay and initialize your system
  LCD_Initializtion();     //LCD initialization
  //LCD touch panel initializaion
  LCD_Clear(Red);       //Clear the LCD with filled with red.
  //You can fill functions to calibrate touch screen.
  /* Infinite loop */
  while (1)
  {
       //You can fill functions to show touch coordinate on the LCD.
  }
}
```