ARC

LCD

支持	www.armrunc.com http://armrunc.taobao.com

Change History

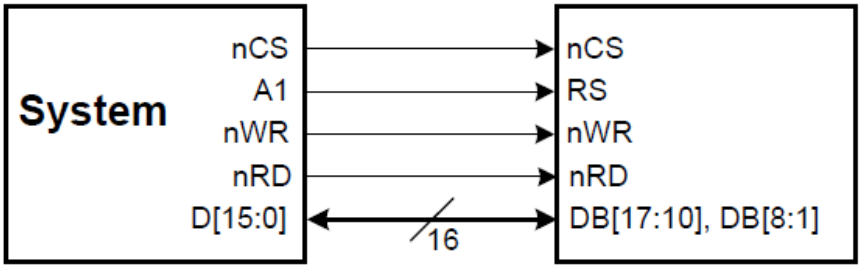
版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-03-05

目录

第15章 LCD

15.1 LCD 简介

该实例 LCD 采用 i80/16 位并口接口，它与系统的连接图如下：



各个引脚的定义如下：

nCS	芯片选中脚，低电平选中芯片，可访问。高电平芯片不可访问。
RS	寄存器选中脚，低电平选中索引或者状态寄存器，高电平选中控制寄存器。
nWR	写触发脚，低电平使能写操作。
nRD	读触发脚，低电平使能读操作。
D[15:0]	数据脚

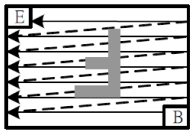
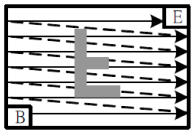
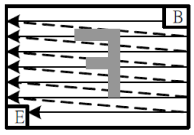
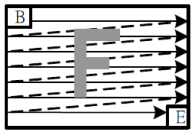
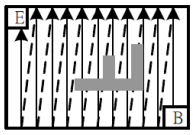
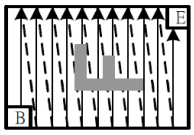
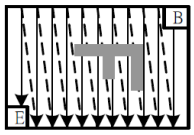
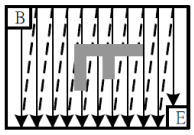
15.1.1 LCD 控制器显示更新

以 ILI9325 为例，控制 LCD 扫描的寄存器为 R3H，如下：

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

AM 控制 GRAM 更新的方向，当 AM = 0 时，地址沿着水平方向更新，当 AM = 1 时，地址沿着垂直方向更新。当你设置了 R50H – R53H 来定义一个显示区域时，GRAM 的更新只会在此设定的区域内更新。

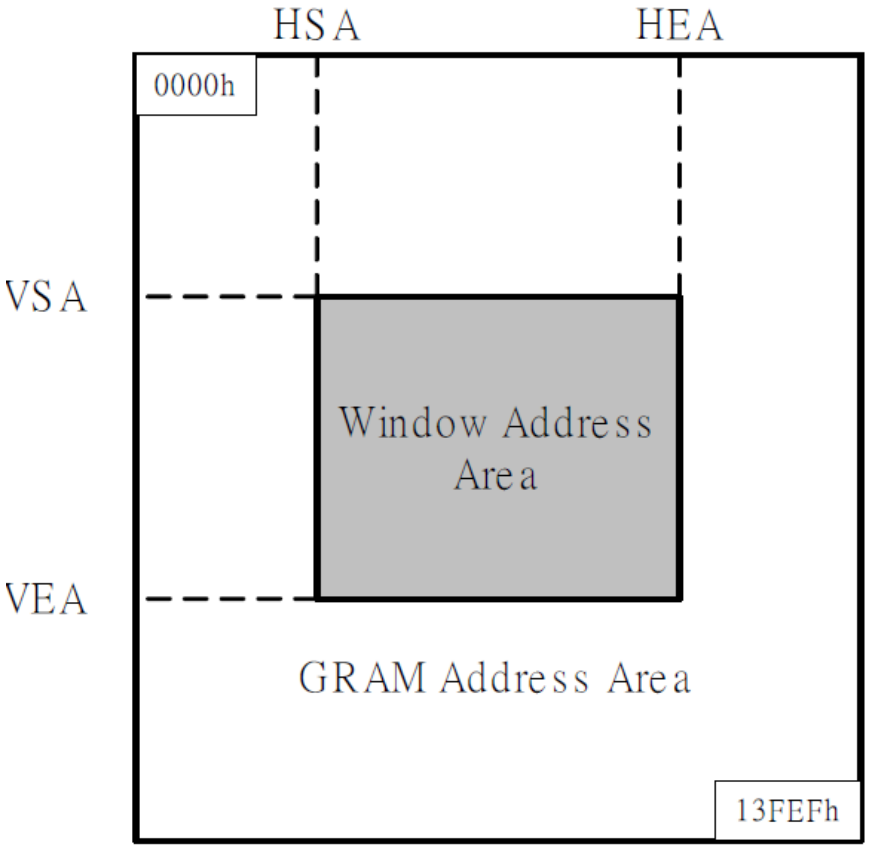
ID[1:0]，当更新一个显示区域的一个点时，他用来控制 AC(address counter)为自动减一还是加一。下面的图比较直观：

	I/D[1:0] = 00 Horizontal : decrement Vertical : decrement	I/D[1:0] = 01 Horizontal : increment Vertical : decrement	I/D[1:0] = 10 Horizontal : decrement Vertical : increment	I/D[1:0] = 11 Horizontal : increment Vertical : increment
AM = 0 Horizontal				
AM = 1 Vertical				

R50H – R53H 寄存器描述如下：

	R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R50h	W	1	0	0	0	0	0	0	0	0	HSA7	HSA6	HSA5	HSA4	HSA3	HSA2	HSA1	HSA0
R51h	W	1	0	0	0	0	0	0	0	0	HEA7	HEA6	HEA5	HEA4	HEA3	HEA2	HEA1	HEA0
R52h	W	1	0	0	0	0	0	0	0	VSA8	VSA7	VSA6	VSA5	VSA4	VSA3	VSA2	VSA1	VSA0
R53h	W	1	0	0	0	0	0	0	0	VEA8	VEA7	VEA6	VEA5	VEA4	VEA3	VEA2	VEA1	VEA0

它定义了如下的显示区域：



15.2 LCD 应用实例 ----- 在 LCD 上显示字符串

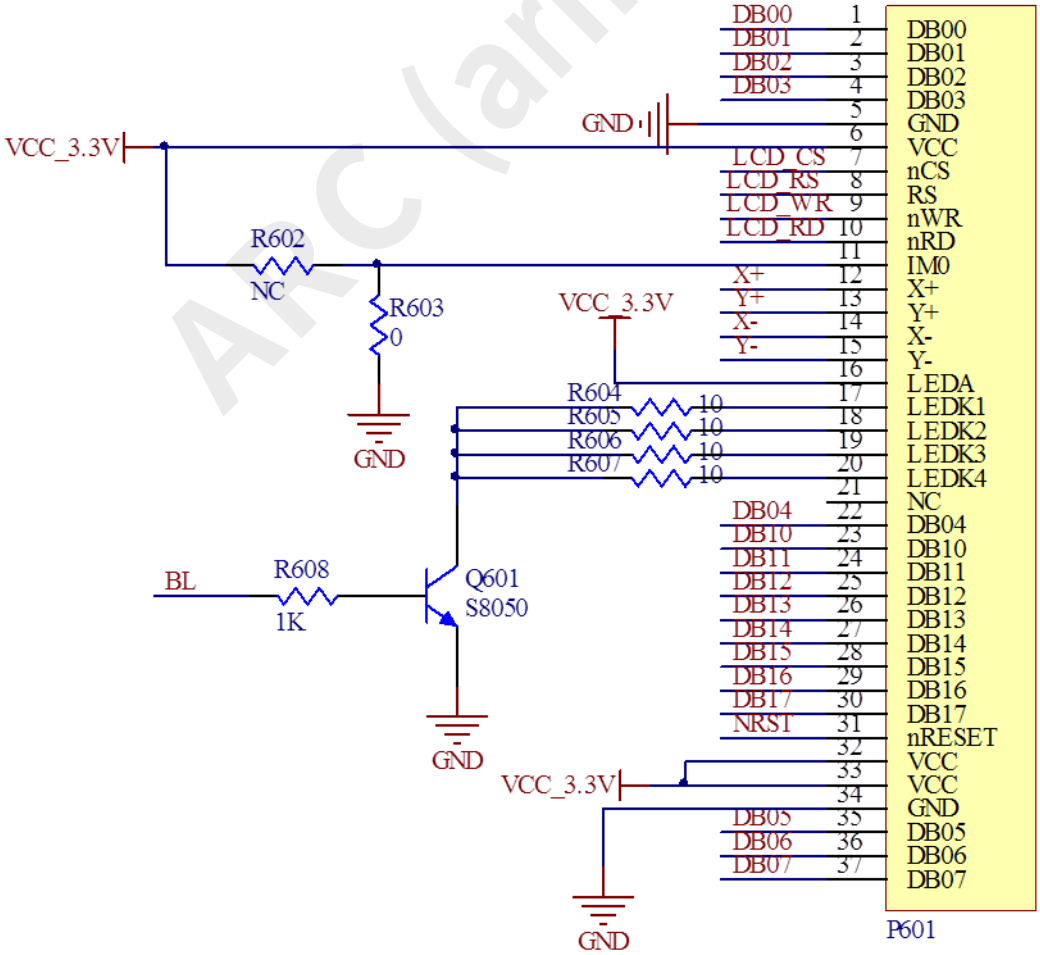
15.2.1 实例描述

本实例初始化 LCD 后，一直在 LCD 屏幕上循环显示字符串。 首先会读取 LCD 控制芯片型号，然后根据不同的 LCD 控制芯片分别进行初始化。

15.2.2 硬件设计

采用市面上通用的 37 pin 接口，pin 脚分配和电路图如下：

LCD_CS	PD2
LCD_RD	PC10
LCD_WR	PC11
LCD_RS	PC12
BL	PC5
RESET	NRST
DATA	PB0-PB15



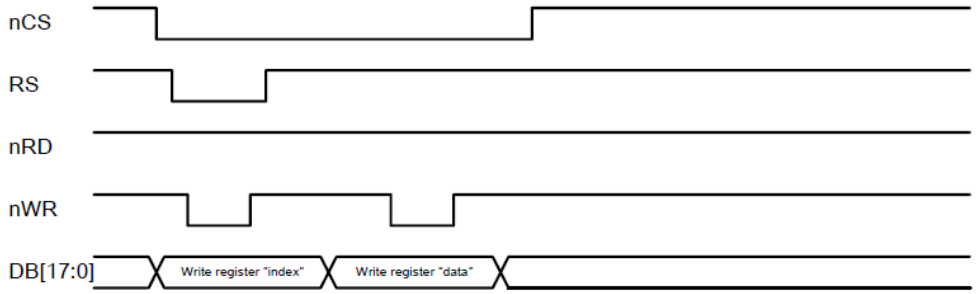
15.2.3 软件设计

本实例首先初始化 SysTick 用于精确延迟，然后初始化串口来调试输出信息。

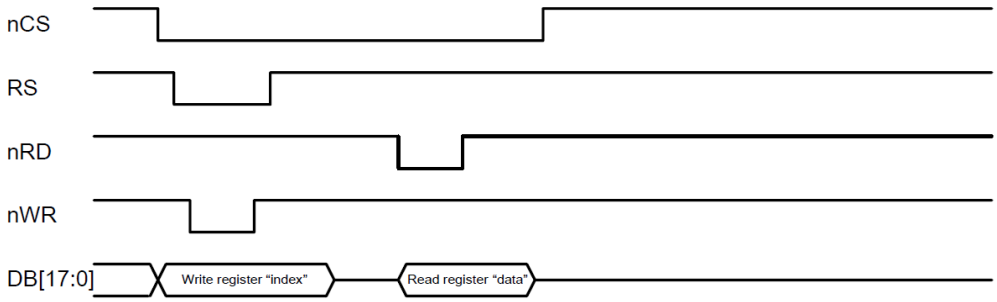
然后在读取 LCD 控制芯片寄存器 0x0 来得到该芯片的 ID，根据这个 ID 来分别进行初始化。

在读取寄存器是，需要将数据脚 PB[0:15]设置为悬浮输入，完成读取后，该数据线将会被设置为推挽输出。所以在写寄存器是，无需对数据线 GPIO 配置。

写寄存器的时序图如下，具体实现函数为 ARC_LCD_WriteReg():



读寄存器的时序图如下，具体实现函数为 ARC_LCD_ReadReg():



该实例的主要代码如下：

文件 LCD_main.c：

```
/**
  * @brief   Main program, show strings on LCD screen.
  * @param   None
  * @retval None
  */
int main(void)
{
    uint16_t x = 0;
    ARC_SysTick_Init();

    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);

    ARC_LCD_Init();
    ARC_LCD_On();
    while(1)
```

```
    {
        switch(x)
        {
            case 0:
                ARC_LCD_Clear(LCD_COLOR_BLACK);
                break;
            case 1:
                ARC_LCD_Clear(LCD_COLOR_MAGENTA);
                break;
            case 2:
                ARC_LCD_Clear(LCD_COLOR_GREEN);
                break;
            case 3:
                ARC_LCD_Clear(LCD_COLOR_RED);
                break;
            case 4:
                ARC_LCD_Clear(LCD_COLOR_BLUE);
                break;
        }

        x++;
        x = (x > 4) ? 0 : x;

        ARC_LCD_ShowString(30, 50, "ARC STM32");

        ARC_LCD_ShowString(30, 70, "TFT LCD Example");

        ARC_LCD_ShowString(30, 90, "www.armrunc.com");

        ARC_LCD_ShowString(30, 110, "http://armrunc.taobao.com");
        ARC_SysTick_Delay(2000);
    }
}
```

文件 ARC_LCD.c：

```
#define ARC_LCD_CS_SET()        GPIOD->BSRR = GPIO_Pin_2    /*
chip select pin */
#define ARC_LCD_RD_SET()        GPIOC->BSRR = GPIO_Pin_10    /*
register select pin */
#define ARC_LCD_WR_SET()        GPIOC->BSRR = GPIO_Pin_11    /*
read strobe signal, low active */
#define ARC_LCD_RS_SET()        GPIOC->BSRR = GPIO_Pin_12    /*
write strobe signal, low active */
```

```
#define ARC_LCD_CS_RESET()      GPIOD->BRR = GPIO_Pin_2
#define ARC_LCD_RD_RESET()      GPIOC->BRR = GPIO_Pin_10
#define ARC_LCD_WR_RESET()      GPIOC->BRR = GPIO_Pin_11
#define ARC_LCD_RS_RESET()      GPIOC->BRR = GPIO_Pin_12

#define ARC_LCD_BL_RESET()      GPIOC->BRR = GPIO_Pin_5
#define ARC_LCD_BL_SET()        GPIOC->BSRR = GPIO_Pin_5

#define ARC_LCD_OUT(PortVal)    GPIOB->ODR = PortVal
#define ARC_LCD_IN(PortVal)     PortVal = GPIOB->IDR

#define ARC_LCD_FLOAT_INPUT()\
{\
    GPIOB->CRH = 0x44444444;\
    GPIOB->CRL = 0x44444444;\
}

#define ARC_LCD_PP_OUTPUT()\
{\
    GPIOB->CRH = 0x33333333;\
    GPIOB->CRL = 0x33333333;\
}
static ARC_LCD_Params ARC_LCD_Param;
/**
  * @brief   get the pointer to the LCD parameters.
  * @param   None
  * @retval the pointer to the LCD parameters
  */
ARC_LCD_Params *ARC_LCD_get_param(void)
{
    return &ARC_LCD_Param;
}
/**
  * @brief   write register index. Becareful, no CS operation.
  * @param   irData, the index register value to be written.
  * @retval None
  */
#define ARC_LCD_WriteRegIndex(irData)\
{\
    ARC_LCD_RS_RESET();\
    ARC_LCD_RD_SET();\
    ARC_LCD_OUT(irData);\
    ARC_LCD_WR_RESET();\
    ARC_LCD_WR_SET();\
```

```
    ARC_LCD_RS_SET();\
}

/**
  * @brief   write register data, becareful, no CS or RD operation.
  * @param   Data, the data to be written.
  * @retval None
  */
#define ARC_LCD_WriteRegData(RegData)\
{\
    ARC_LCD_OUT(RegData);\
    ARC_LCD_WR_RESET();\
    ARC_LCD_WR_SET();\
}

/**
  * @brief   write GRAM data, becareful, no CS or RD operation.
  * @param   Data, the data to be written.
  * @retval None
  */
#define ARC_LCD_WriteGRAMData ARC_LCD_WriteRegData

/**
  * @brief   read register data, becareful, no CS or RD operation.
  * @param   Data, the data read from IR.
  * @retval None
  */
#define ARC_LCD_ReadRegData(RegData)\
{\
    ARC_LCD_RD_RESET();\
    ARC_LCD_RD_SET();\
    ARC_LCD_IN(RegData);\
}

/**
  * @brief   read GRAM data, becareful, no CS or RD operation.
  * @param   Data, the data read from IR.
  * @retval None
  */

#define ARC_LCD_ReadGRAMData(GRAMData)\
{\
    ARC_LCD_ReadRegData(GRAMData)\
    ARC_LCD_ReadRegData(GRAMData)\
```

```
}

/**
   * @brief    Sets the cursor position.
   * @param    Xpos: specifies the X position.
   * @param    Ypos: specifies the Y position.
   * @retval None
   */
#define ARC_LCD_SetCursor(x, y)\
{\
      ARC_LCD_WriteRegIndex(LCD_REG_32)\
      ARC_LCD_WriteRegData(x)\
      ARC_LCD_WriteRegIndex(LCD_REG_33)\
      ARC_LCD_WriteRegData(y)\
}

/**
   * @brief    Write LCD register.
   * @param    LCD_Index, the register index to be written.
   * @param    LCD_Data, value to be written.
   * @retval None.
   */
/*****************************************************************************
  **
**
  ** nCS          ----\_____________________________________________/-------
**
  ** RS           ------\_____________/----------------------------------   **
  ** nRD          ------------------------------------------------------   **
  ** nWR          --------\________/--------\_____/-----------------------   **
  ** DB[0:15]    ---------[index]----------[data]-----------------------   **
  **
**
  *****************************************************************************/

__inline void ARC_LCD_WriteReg(uint16_t LCD_Index, uint16_t LCD_Data)
{
      ARC_LCD_CS_RESET();
      ARC_LCD_WriteRegIndex(LCD_Index);
      ARC_LCD_WriteRegData(LCD_Data);
      ARC_LCD_CS_SET();
}

/**
```

```
  * @brief   Write to the LCD RAM a number of value.
  * @param   *LCD_RegValue, the pointers to the values to be written.
  * @param   NumValue, the number of values to be written.
  * @retval None.
  */
__inline void ARC_LCD_WriteGRAM(const uint16_t *LCD_RegValue, uint32_t NumValue)
{
    ARC_LCD_WriteRegIndex(LCD_REG_34);

    for(; NumValue; NumValue--)
    {
        ARC_LCD_WriteGRAMData(*LCD_RegValue);
        LCD_RegValue++;
    }
}

/**
  * @brief   Write to the LCD RAM a number of value.
  * @param   LCD_RegValue, the values to be written.
  * @param   NumValue, the number of values to be written.
  * @retval None.
  */
__inline void ARC_LCD_WriteGRAM_Same(uint16_t LCD_RegValue, uint32_t NumValue)
{
    ARC_LCD_WriteRegIndex(LCD_REG_34);

    for(; NumValue; NumValue--)
    {
        ARC_LCD_WriteGRAMData(LCD_RegValue);
    }
}

/**
  * @brief   Write to the LCD RAM a number of value.
  * @param   *LCD_RegValue, the pointers to the values to be written.
  * @param   NumValue, the number of values to be written.
  * @retval None.
  */
__inline void ARC_LCD_SetMultiPixelsIndex(uint16_t x, uint16_t y, uint16_t const *LCD_RegValue, uint32_t NumValue)
{
    ARC_LCD_CS_RESET();
```

```
    ARC_LCD_SetCursor(x, y);
    ARC_LCD_WriteGRAM(LCD_RegValue, NumValue);
    ARC_LCD_CS_SET();
}

/**
  * @brief   Read LCD register.
  * @param   LCD_Reg, the register index to be read.
  * @retval value read from LCD.
  */
__inline uint16_t ARC_LCD_ReadReg(uint16_t LCD_Index)
{
    uint16_t LCD_Value;
    ARC_LCD_CS_RESET();
    ARC_LCD_WriteRegIndex(LCD_Index);
    /* Configure port B as foating input */
    ARC_LCD_FLOAT_INPUT();
    ARC_LCD_ReadRegData(LCD_Value);
    ARC_LCD_PP_OUTPUT();
    ARC_LCD_CS_SET();
    return LCD_Value;
}

/**
  * @brief   Sets a display window
  * @param   x1: specifies the X top left position.
  * @param   y1: specifies the Y top left position.
  * @param   x2: specifies the X buttom right position.
  * @param   y2: specifies the Y buttom right position.
  * @retval None
  */
__inline void ARC_LCD_SetDisplayWindow(uint16_t x0,uint16_t y0,uint16_t
x1,uint16_t y1)
{
    /* Horizontal GRAM Start Address */
    ARC_LCD_WriteRegIndex(LCD_REG_80);
    ARC_LCD_WriteRegData(x0);
    /* Horizontal GRAM End Address */
    ARC_LCD_WriteRegIndex(LCD_REG_81);
    ARC_LCD_WriteRegData(x1);
    /* Vertical GRAM Start Address */
    ARC_LCD_WriteRegIndex(LCD_REG_82);
    ARC_LCD_WriteRegData(y0);
    /* Vertical GRAM End Address */
```

```
	ARC_LCD_WriteRegIndex(LCD_REG_83);
	ARC_LCD_WriteRegData(y1);
	ARC_LCD_SetCursor(x1, y1);
}


/**
  * @brief  initialize panel.
  * @param  None.
  * @retval None
  */
void ARC_ILI9325_Init(LCD_Direction_TypeDef lcd_dir)
{
	/* Start Initial Sequence ----------------------------------------------*/
	ARC_LCD_WriteReg(LCD_REG_0, 0x0001); /* Start internal OSC. */
	ARC_LCD_WriteReg(LCD_REG_1, 0x0100); /* Set SS and SM bit */
	ARC_LCD_WriteReg(LCD_REG_2, 0x0700); /* Set 1 line inversion */
	ARC_LCD_WriteReg(LCD_REG_3, 0x1018); /* Set GRAM write direction
and BGR=1. */
	ARC_LCD_WriteReg(LCD_REG_4, 0x0000); /* Resize register */
	ARC_LCD_WriteReg(LCD_REG_8, 0x0202); /* Set the back porch and
front porch */
	ARC_LCD_WriteReg(LCD_REG_9, 0x0000); /* Set non-display area
refresh cycle ISC[3:0] */
	ARC_LCD_WriteReg(LCD_REG_10, 0x0000); /* FMARK function */
	ARC_LCD_WriteReg(LCD_REG_12, 0x0000); /* RGB interface setting */
	ARC_LCD_WriteReg(LCD_REG_13, 0x0000); /* Frame marker Position
*/
	ARC_LCD_WriteReg(LCD_REG_15, 0x0000); /* RGB interface polarity */

	/* Power On sequence ---------------------------------------------------*/
	ARC_LCD_WriteReg(LCD_REG_16, 0x0000); /* SAP, BT[3:0], AP, DSTB,
SLP, STB */
	ARC_LCD_WriteReg(LCD_REG_17, 0x0000); /* DC1[2:0], DC0[2:0],
VC[2:0] */
	ARC_LCD_WriteReg(LCD_REG_18, 0x0000); /* VREG1OUT voltage */
	ARC_LCD_WriteReg(LCD_REG_19, 0x0000); /* VDV[4:0] for VCOM
amplitude */
	ARC_SysTick_Delay(200);                    /* Dis-charge
capacitor power voltage (200ms) */
	ARC_LCD_WriteReg(LCD_REG_16, 0x17B0); /* SAP, BT[3:0], AP,
DSTB, SLP, STB */
	ARC_LCD_WriteReg(LCD_REG_17, 0x0137); /* DC1[2:0], DC0[2:0],
VC[2:0] */
```

```
    ARC_SysTick_Delay(50);                          /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_18, 0x0139); /* VREG1OUT voltage */
    ARC_SysTick_Delay(50);                          /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_19, 0x1d00); /* VDV[4:0] for VCOM
amplitude */
    ARC_LCD_WriteReg(LCD_REG_41, 0x0013); /* VCM[4:0] for VCOMH */
    ARC_SysTick_Delay(50);                          /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_32, 0x0000); /* GRAM horizontal
Address */
    ARC_LCD_WriteReg(LCD_REG_33, 0x0000); /* GRAM Vertical Address
*/

    /* Adjust the Gamma Curve (ILI9325)-------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_48, 0x0007);
    ARC_LCD_WriteReg(LCD_REG_49, 0x0302);
    ARC_LCD_WriteReg(LCD_REG_50, 0x0105);
    ARC_LCD_WriteReg(LCD_REG_53, 0x0206);
    ARC_LCD_WriteReg(LCD_REG_54, 0x0808);
    ARC_LCD_WriteReg(LCD_REG_55, 0x0206);
    ARC_LCD_WriteReg(LCD_REG_56, 0x0504);
    ARC_LCD_WriteReg(LCD_REG_57, 0x0007);
    ARC_LCD_WriteReg(LCD_REG_60, 0x0105);
    ARC_LCD_WriteReg(LCD_REG_61, 0x0808);

    /* Set GRAM area -------------------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_80, 0x0000); /* Horizontal GRAM Start
Address */
    ARC_LCD_WriteReg(LCD_REG_81, 0x00EF); /* Horizontal GRAM End
Address */
    ARC_LCD_WriteReg(LCD_REG_82, 0x0000); /* Vertical GRAM Start
Address */
    ARC_LCD_WriteReg(LCD_REG_83, 0x013F); /* Vertical GRAM End
Address */

    ARC_LCD_WriteReg(LCD_REG_96,    0xA700); /* Gate Scan Line(GS=1,
scan direction is G320~G1) */
    ARC_LCD_WriteReg(LCD_REG_97,    0x0001); /* NDL,VLE, REV */
    ARC_LCD_WriteReg(LCD_REG_106, 0x0000); /* set scrolling line */

    /* Partial Display Control ---------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_128, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_129, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_130, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_131, 0x0000);
```

```
	ARC_LCD_WriteReg(LCD_REG_132, 0x0000);
	ARC_LCD_WriteReg(LCD_REG_133, 0x0000);

	/* Panel Control ----------------------------------------------------*/
	ARC_LCD_WriteReg(LCD_REG_144, 0x0010);
	ARC_LCD_WriteReg(LCD_REG_146, 0x0000);
	ARC_LCD_WriteReg(LCD_REG_147, 0x0003);
	ARC_LCD_WriteReg(LCD_REG_149, 0x0110);
	ARC_LCD_WriteReg(LCD_REG_151, 0x0000);
	ARC_LCD_WriteReg(LCD_REG_152, 0x0000);

	/* set GRAM write direction and BGR = 1 */
	/* I/D=00 (Horizontal : increment, Vertical : decrement) */
	/* AM=1 (address is updated in vertical writing direction) */
	ARC_LCD_WriteReg(LCD_REG_3, (1<<12)|(3<<4)|(0<<3));

	ARC_LCD_WriteReg(LCD_REG_7, 0x0133); /* 262K color and display
ON */
}

/**
  * @brief   initialize panel.
  * @param   None.
  * @retval None
  */
void ARC_ILI9320_Init(LCD_Direction_TypeDef lcd_dir)
{
	ARC_SysTick_Delay(50); /* Delay 50 ms */
	/* Start Initial Sequence -------------------------------------------*/
	ARC_LCD_WriteReg(LCD_REG_229, 0x8000); /* Set the internal vcore
voltage */
	ARC_LCD_WriteReg(LCD_REG_0,   0x0001); /* Start internal OSC. */
	ARC_LCD_WriteReg(LCD_REG_1,   0x0100); /* set SS and SM bit */
	ARC_LCD_WriteReg(LCD_REG_2,   0x0700); /* set 1 line inversion */
	ARC_LCD_WriteReg(LCD_REG_3,   0x1030); /* set GRAM write
direction and BGR=1. */
	ARC_LCD_WriteReg(LCD_REG_4,   0x0000); /* Resize register */
	ARC_LCD_WriteReg(LCD_REG_8,   0x0202); /* set the back porch and
front porch */
	ARC_LCD_WriteReg(LCD_REG_9,   0x0000); /* set non-display area
refresh cycle ISC[3:0] */
	ARC_LCD_WriteReg(LCD_REG_10, 0x0000); /* FMARK function */
	ARC_LCD_WriteReg(LCD_REG_12, 0x0000); /* RGB interface setting */
```

```
    ARC_LCD_WriteReg(LCD_REG_13, 0x0000); /* Frame marker Position
*/
    ARC_LCD_WriteReg(LCD_REG_15, 0x0000); /* RGB interface polarity */
    /* Power On sequence ---------------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_16, 0x0000); /* SAP, BT[3:0], AP, DSTB,
SLP, STB */
    ARC_LCD_WriteReg(LCD_REG_17, 0x0000); /* DC1[2:0], DC0[2:0],
VC[2:0] */
    ARC_LCD_WriteReg(LCD_REG_18, 0x0000); /* VREG1OUT voltage */
    ARC_LCD_WriteReg(LCD_REG_19, 0x0000); /* VDV[4:0] for VCOM
amplitude */
    ARC_SysTick_Delay(200);                    /* Dis-charge
capacitor power voltage (200ms) */
    ARC_LCD_WriteReg(LCD_REG_16, 0x17B0); /* SAP, BT[3:0], AP,
DSTB, SLP, STB */
    ARC_LCD_WriteReg(LCD_REG_17, 0x0137); /* DC1[2:0], DC0[2:0],
VC[2:0] */
    ARC_SysTick_Delay(50);                     /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_18, 0x0139); /* VREG1OUT voltage */
    ARC_SysTick_Delay(50);                     /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_19, 0x1d00); /* VDV[4:0] for VCOM
amplitude */
    ARC_LCD_WriteReg(LCD_REG_41, 0x0013); /* VCM[4:0] for VCOMH */
    ARC_SysTick_Delay(50);                     /* Delay 50 ms */
    ARC_LCD_WriteReg(LCD_REG_32, 0x0000); /* GRAM horizontal
Address */
    ARC_LCD_WriteReg(LCD_REG_33, 0x0000); /* GRAM Vertical Address
*/
    /* Adjust the Gamma Curve ----------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_48, 0x0006);
    ARC_LCD_WriteReg(LCD_REG_49, 0x0101);
    ARC_LCD_WriteReg(LCD_REG_50, 0x0003);
    ARC_LCD_WriteReg(LCD_REG_53, 0x0106);
    ARC_LCD_WriteReg(LCD_REG_54, 0x0b02);
    ARC_LCD_WriteReg(LCD_REG_55, 0x0302);
    ARC_LCD_WriteReg(LCD_REG_56, 0x0707);
    ARC_LCD_WriteReg(LCD_REG_57, 0x0007);
    ARC_LCD_WriteReg(LCD_REG_60, 0x0600);
    ARC_LCD_WriteReg(LCD_REG_61, 0x020b);

    /* Set GRAM area -------------------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_80, 0x0000); /* Horizontal GRAM Start
Address */
```

```
    ARC_LCD_WriteReg(LCD_REG_81, 0x00EF); /* Horizontal GRAM End
Address */
    ARC_LCD_WriteReg(LCD_REG_82, 0x0000); /* Vertical GRAM Start
Address */
    ARC_LCD_WriteReg(LCD_REG_83, 0x013F); /* Vertical GRAM End
Address */
    ARC_LCD_WriteReg(LCD_REG_96,   0x2700); /* Gate Scan Line */
    ARC_LCD_WriteReg(LCD_REG_97,   0x0001); /* NDL,VLE, REV */
    ARC_LCD_WriteReg(LCD_REG_106, 0x0000); /* set scrolling line */
    /* Partial Display Control ---------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_128, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_129, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_130, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_131, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_132, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_133, 0x0000);
    /* Panel Control -------------------------------------------------*/
    ARC_LCD_WriteReg(LCD_REG_144, 0x0010);
    ARC_LCD_WriteReg(LCD_REG_146, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_147, 0x0003);
    ARC_LCD_WriteReg(LCD_REG_149, 0x0110);
    ARC_LCD_WriteReg(LCD_REG_151, 0x0000);
    ARC_LCD_WriteReg(LCD_REG_152, 0x0000);
    /* Set GRAM write direction and BGR = 1 */
    /* I/D=01 (Horizontal : increment, Vertical : decrement) */
    /* AM=1 (address is updated in vertical writing direction) */
    ARC_LCD_WriteReg(LCD_REG_3, 0x1018);
    ARC_LCD_WriteReg(LCD_REG_7, 0x0173); /* 262K color and display
ON */
}

/**
  * @brief   Initialize LCD.
  * @param   None
  * @retval None
  */
int32_t ARC_LCD_Init(void)
{
    uint16_t DeviceCode;
    ARC_LCD_Params *lcd_par;
    lcd_par = ARC_LCD_get_param();

    lcd_par->LCD_Type = LCD_OTHER;
    lcd_par->LCD_BusType = LCD_I80;
```

```
    lcd_par->LCD_Direction = LCD_DIR_VERTICAL;
    ARC_LCD_RCC_Init();
    ARC_LCD_GPIO_Init();
    ARC_SysTick_Delay(50); /* delay 50 ms */
    ARC_LCD_WriteReg(0x0000,0x0001);
    ARC_SysTick_Delay(50);
    DeviceCode = ARC_LCD_ReadReg(0x0000);

    if(DeviceCode==0x9320)
    {
        lcd_par->LCD_Type = LCD_ILI9320;
        ARC_ILI9320_Init(lcd_par->LCD_Direction);
    }
    else if(DeviceCode==0x9325)
    {
        lcd_par->LCD_Type = LCD_ILI9325;
        ARC_ILI9325_Init(lcd_par->LCD_Direction);
    }
    else
    {
        return 1;
    }
    return 0;
}

/**
  * @brief   Enables the Display.
  * @param   None
  * @retval None
  */
void ARC_LCD_On(void)
{
    ARC_LCD_Params *lcd_par;
    lcd_par = ARC_LCD_get_param();
    if((lcd_par->LCD_Type == LCD_ILI9320) || (lcd_par->LCD_Type ==
LCD_SPFD5408))
    {
        /* Display On */
        ARC_LCD_WriteReg(LCD_REG_7, 0x0173); /* 262K color and
display ON */
    }
    else if(lcd_par->LCD_Type == LCD_HX8312)
    {
        ARC_LCD_WriteReg(LCD_REG_1, 0x50);
```

```
            ARC_LCD_WriteReg(LCD_REG_5, 0x04);
            /* Display On */
            ARC_LCD_WriteReg(LCD_REG_0, 0x80);
            ARC_LCD_WriteReg(LCD_REG_59, 0x01);
            ARC_SysTick_Delay(40);    /* Delay 40 ms */
            ARC_LCD_WriteReg(LCD_REG_0, 0x20);
        }
        ARC_LCD_BL_SET();
}

/**
   * @brief    Disables the Display.
   * @param    None
   * @retval None
   */
void ARC_LCD_Off(void)
{
        ARC_LCD_Params *lcd_par;
        lcd_par = ARC_LCD_get_param();
        ARC_LCD_BL_RESET();
        if((lcd_par->LCD_Type == LCD_ILI9320) || (lcd_par->LCD_Type ==
LCD_SPFD5408))
        {
            /* Display Off */
            ARC_LCD_WriteReg(LCD_REG_7, 0x0);
        }
        else if(lcd_par->LCD_Type == LCD_HX8312)
        {
            /* Display Off */
            ARC_LCD_WriteReg(LCD_REG_0, 0xA0);
            ARC_SysTick_Delay(40);    /* Delay 40 ms */
            ARC_LCD_WriteReg(LCD_REG_59, 0x00);
        }
}

/**
   * @brief    set a pixel at position (x, y) with a color ColorIndex.
   * @param    x, the x position of the LCD
   * @param    y, the y position of the LCD
   * @param    ColorIndex, the color to be set to the pixel.
   * @retval None
   */
void ARC_LCD_SetPixelIndex(int32_t x,int32_t y,int32_t ColorIndex)
{
```

```
    ARC_LCD_CS_RESET();
    ARC_LCD_SetCursor(x, y);
    ARC_LCD_WriteRegIndex(LCD_REG_34);
    ARC_LCD_WriteGRAMData(ColorIndex);
    ARC_LCD_CS_SET();
}

/**
  * @brief   get a pixel value at position (x, y).
  * @param   x, the x position of the LCD
  * @param   y, the y position of the LCD
  * @retval the pixel value
  */
uint32_t ARC_LCD_GetPixelIndex(int32_t x, int32_t y)
{
    uint16_t temp;
    ARC_LCD_CS_RESET();
    ARC_LCD_SetCursor(x, y);
    ARC_LCD_WriteRegIndex(LCD_REG_34);
    ARC_LCD_ReadGRAMData(temp);
    ARC_LCD_CS_SET();
    return   (((temp)   & 0x1f)<<11) + (((temp>>5)   & 0x3f)<<5) +
(((temp>>11) & 0x1f));
}

#if (defined ARC_UCGUI && defined ARC_FREERTOS)
/**
  * @brief   draw a line starting from position (x, y).
  */
__inline void   ARC_DrawBitLine1BPP(int32_t x, int32_t y, const uint8_t *p,
int32_t Diff, int32_t xsize, const uint16_t *pTrans)
{
    uint16_t Index0 = *(pTrans + 0);
    uint16_t Index1 = *(pTrans + 1);
    x += Diff;
    switch (GUI_Context.DrawMode & (LCD_DRAWMODE_TRANS |
LCD_DRAWMODE_XOR))
    {
        case LCD_DRAWMODE_NORMAL:
            do
            {
                ARC_LCD_SetPixelIndex(x++, y, (*p & (0x80 >> Diff)) ?
Index1 : Index0);
                if (++Diff == 8)
```

```
                    {
                        Diff = 0;
                        p++;
                    }
                } while (--xsize);
                break;

            case LCD_DRAWMODE_TRANS:
                do
                {
                    if (*p & (0x80 >> Diff))
                        ARC_LCD_SetPixelIndex(x, y, Index1);
                    x++;
                    if (++Diff == 8)
                    {
                        Diff = 0;
                        p++;
                    }
                } while (--xsize);
                break;

            case LCD_DRAWMODE_XOR | LCD_DRAWMODE_TRANS:
            case LCD_DRAWMODE_XOR:
                do
                {
                    if (*p & (0x80 >> Diff))
                    {
                        int32_t Pixel = LCD_L0_GetPixelIndex(x, y);
                        ARC_LCD_SetPixelIndex(x, y, LCD_NUM_COLORS -
1 - Pixel);
                    }
                    x++;
                    if (++Diff == 8)
                    {
                        Diff = 0;
                        p++;
                    }
                } while (--xsize);
                break;
            default:
                break;
        }
}
```

```
/**
  * @brief    draw a line starting from position (x, y).
  */
__inline void   ARC_DrawBitLine2BPP(int32_t x, int32_t y, const uint8_t * p,
int32_t Diff, int32_t xsize, const uint16_t * pTrans)
{
    uint16_t Pixels = *p;
    int32_t CurrentPixel = Diff;
    int32_t Shift;
    uint16_t Index;
    int32_t PixelValue;

    x += Diff;
    switch (GUI_Context.DrawMode & (LCD_DRAWMODE_TRANS |
LCD_DRAWMODE_XOR))
    {
        case LCD_DRAWMODE_NORMAL:
            do
            {
                Shift = (3 - CurrentPixel) << 1;
                Index = (Pixels & (0xC0 >> (6 - Shift))) >> Shift;
                PixelValue = pTrans ? (*(pTrans + Index)) : Index;

                ARC_LCD_SetPixelIndex(x++, y, PixelValue);
                if (++CurrentPixel == 4)
                {
                    CurrentPixel = 0;
                    Pixels = *(++p);
                }
            } while (--xsize);
            break;

        case LCD_DRAWMODE_TRANS:
            do
            {
                Shift = (3 - CurrentPixel) << 1;
                Index = (Pixels & (0xC0 >> (6 - Shift))) >> Shift;

                if (Index)
                {
                    PixelValue = pTrans ? (*(pTrans + Index)) : Index;
                    ARC_LCD_SetPixelIndex(x, y, PixelValue);
                }
                x++;
```

```
                    if (++CurrentPixel == 4)
                    {
                        CurrentPixel = 0;
                        Pixels = *(++p);
                    }
                } while (--xsize);
                break;

                default:
                    break;
        }
}

/**
   * @brief    draw a line starting from position (x, y).
   */
__inline void    ARC_DrawBitLine4BPP(int32_t x, int32_t y, uint8_t const    * p,
int32_t Diff, int32_t xsize, const uint16_t * pTrans)
{
        uint16_t Pixels = *p;
        int32_t CurrentPixel = Diff;
        int32_t Shift;
        uint16_t Index;
        int32_t PixelValue;

        x += Diff;
        switch (GUI_Context.DrawMode & (LCD_DRAWMODE_TRANS |
LCD_DRAWMODE_XOR))
        {
            case LCD_DRAWMODE_NORMAL:
                do
                {
                    Shift = (1 - CurrentPixel) << 2;
                    Index = (Pixels & (0xF0 >> (4 - Shift))) >> Shift;
                    PixelValue = pTrans ? (*(pTrans + Index)) : Index;

                    ARC_LCD_SetPixelIndex(x++, y, PixelValue);
                    if (++CurrentPixel == 2)
                    {
                        CurrentPixel = 0;
                        Pixels = *(++p);
                    }
                } while (--xsize);
                break;
```

```
            case LCD_DRAWMODE_TRANS:
                do
                {
                    Shift = (1 - CurrentPixel) << 2;
                    Index = (Pixels & (0xF0 >> (4 - Shift))) >> Shift;

                    if (Index)
                    {
                        PixelValue = pTrans ? (*(pTrans + Index)) : Index;
                        ARC_LCD_SetPixelIndex(x, y, PixelValue);
                    }
                    x++;
                    if (++CurrentPixel == 2)
                    {
                        CurrentPixel = 0;
                        Pixels = *(++p);
                    }
                } while (--xsize);
                break;

            default:
                break;
        }
}

/**
  * @brief  draw a line starting from position (x, y).
  */
__inline void  ARC_DrawBitLine8BPP(int32_t x, int32_t y, uint8_t const  * p,
int32_t xsize, const uint16_t * pTrans)
{
    int32_t PixelValue;
    switch (GUI_Context.DrawMode & (LCD_DRAWMODE_TRANS |
LCD_DRAWMODE_XOR))
    {
        case LCD_DRAWMODE_NORMAL:
            for (; xsize > 0; xsize--, x++, p++)
            {

                PixelValue = pTrans ? *(pTrans + *p) : *p;
                ARC_LCD_SetPixelIndex(x, y, PixelValue);
            }
            break;
```

```
            case LCD_DRAWMODE_TRANS:
                for (; xsize > 0; xsize--, x++, p++)
                {
                    if (*p)
                    {
                        PixelValue = pTrans ? *(pTrans + *p) : *p;
                        ARC_LCD_SetPixelIndex(x, y, PixelValue);
                    }
                }
                break;
            default:
                break;
        }
}

/**
  * @brief  draw a line starting from position (x, y).
  */
__inline void  ARC_DrawBitLine16BPP(int32_t x, int32_t y, const uint16_t *p,
int32_t xsize, const uint16_t *pTrans)
{
    const  uint16_t *PixelIndex;
    if ((GUI_Context.DrawMode & LCD_DRAWMODE_TRANS) ==
LCD_DRAWMODE_NORMAL)
    {
        PixelIndex = pTrans ? (pTrans + *p) : p;
        ARC_LCD_SetMultiPixelsIndex(x, y, PixelIndex, xsize);
    }
    else
    {
        for (; xsize > 0; xsize--, x++, p++)
        {
            if (*p)
            {
                PixelIndex = pTrans ? (pTrans + *p) : p;
                ARC_LCD_SetPixelIndex(x, y, *PixelIndex);
            }
        }
    }
}

/**
  * @brief  draw a bitmap.
  */
```

```
void ARC_LCD_DrawBitmap(int32_t x0, int32_t y0, int32_t xsize, int32_t ysize,
int32_t BitsPerPixel, int32_t BytesPerLine,
                                const uint8_t * pData, int32_t Diff, const uint16_t*
pTrans)
{
    int32_t i;
    for (i = 0; i < ysize; i++)
    {
        switch (BitsPerPixel)
        {
            case 1:
                ARC_DrawBitLine1BPP(x0, i + y0, pData, Diff, xsize,
pTrans);
                break;
            case 2:
                ARC_DrawBitLine2BPP(x0, i + y0, pData, Diff, xsize,
pTrans);
                break;
            case 4:
                ARC_DrawBitLine4BPP(x0, i + y0, pData, Diff, xsize,
pTrans);
                break;
            case 8:
                ARC_DrawBitLine8BPP(x0, i + y0, pData, xsize, pTrans);
                break;
            case 16:
                ARC_DrawBitLine16BPP(x0, i + y0, (const uint16_t *)pData,
xsize, pTrans);
                break;
        }
        pData += BytesPerLine;
    }
}

/**
  * @brief    draw a horizontal line.
  */
void ARC_LCD_DrawHLine    (int32_t x0, int32_t y,    int32_t x1)
{
    ARC_LCD_CS_RESET();
    ARC_LCD_SetCursor(x0, y);
    ARC_LCD_WriteGRAM_Same(LCD_COLORINDEX, (x1 - x0 + 1));
    ARC_LCD_CS_SET();
}
```

```
/**
  * @brief  draw a vertical line.
  */
void ARC_LCD_DrawVLine  (int32_t x, int32_t y0,  int32_t y1)
{
    /* set GRAM write direction and BGR = 1 */
    /* I/D=00 (Horizontal : increment, Vertical : decrement) */
    /* AM=1 (address is updated in horizontal writing direction) */
    ARC_LCD_WriteReg(LCD_REG_3, (1<<12)|(3<<4)|(1<<3));
    ARC_LCD_CS_RESET();
    ARC_LCD_SetCursor(x, y0);
    ARC_LCD_WriteGRAM_Same(LCD_COLORINDEX, (y1 - y0 + 1));
    ARC_LCD_CS_SET();
    /* set GRAM write direction and BGR = 1 */
    /* I/D=00 (Horizontal : increment, Vertical : decrement) */
    /* AM=0 (address is updated in vertical writing direction) */
    ARC_LCD_WriteReg(LCD_REG_3, (1<<12)|(3<<4)|(0<<3));
}

/**
  * @brief  fill a rectangle.
  */
void ARC_LCD_FillRect  (int32_t x0, int32_t y0, int32_t x1, int32_t y1)
{
    ARC_LCD_CS_RESET();
    ARC_LCD_SetDisplayWindow(x0, y0, x1, y1);
    ARC_LCD_WriteGRAM_Same(LCD_COLORINDEX, ((x1 - x0 + 1) * (y1 -
y0 + 1) - 1));
    ARC_LCD_SetDisplayWindow(0, 0, LCD_XSIZE - 1, LCD_YSIZE - 1);
    ARC_LCD_CS_SET();
}

/**
  * @brief  xor a pixel.
  */
void ARC_LCD_XorPixel(int32_t x, int32_t y)
{
    uint16_t Index = ARC_LCD_GetPixelIndex(x,y);
    ARC_LCD_SetPixelIndex(x, y, LCD_NUM_COLORS - 1 - Index);
}

/**
  * @brief  dummy function.
```

```
  */
void ARC_LCD_SetOrg(uint8_t Pos, uint32_t color)
{
}

/**
  * @brief    dummy function.
  */
void ARC_LCD_SetLUTEntry(void)
{
}

/**
  * @brief    dummy function.
  */
void * ARC_LCD_GetDevFunc(int32_t Index)
{
    GUI_USE_PARA(Index);
    return NULL;
}

int32_t LCD_L0_Init(void) __attribute__ (( alias ("ARC_LCD_Init")));
void LCD_L0_On(void)   __attribute__ (( alias ("ARC_LCD_On")));
void LCD_L0_DrawBitmap(int32_t x0, int32_t y0, int32_t xsize, int32_t ysize,
int32_t BitsPerPixel, int32_t BytesPerLine,
                       const uint8_t GUI_UNI_PTR * pData, int32_t Diff,
const uint16_t* pTrans) __attribute__ (( alias ("ARC_LCD_DrawBitmap")));
void LCD_L0_DrawHLine (int32_t x0, int32_t y,   int32_t x1) __attribute__
(( alias ("ARC_LCD_DrawHLine")));
void LCD_L0_DrawVLine (int32_t x0, int32_t y,   int32_t x1) __attribute__
(( alias ("ARC_LCD_DrawVLine")));
void LCD_L0_FillRect (int32_t x0, int32_t y0, int32_t x1, int32_t y1)
__attribute__ (( alias ("ARC_LCD_FillRect")));
void LCD_L0_SetPixelIndex(int32_t x, int32_t y, int32_t ColorIndex)
__attribute__ (( alias ("ARC_LCD_SetPixelIndex")));
void LCD_L0_XorPixel(int32_t x, int32_t y)     __attribute__ (( alias
("ARC_LCD_XorPixel")));
uint32_t LCD_L0_GetPixelIndex(int32_t x, int32_t y)     __attribute__ (( alias
("ARC_LCD_GetPixelIndex")));
void LCD_L0_SetOrg(int32_t x, int32_t y)     __attribute__ (( alias
("ARC_LCD_SetOrg")));
void LCD_L0_SetLUTEntry(uint8_t Pos, LCD_COLOR color) __attribute__
(( alias ("ARC_LCD_SetLUTEntry")));
```

```
void * LCD_L0_GetDevFunc(int32_t Index) __attribute__ (( alias
("ARC_LCD_GetDevFunc")));
#else

/**
  * @brief   draw a big point (3 x 3) on the specific position on the LCD
screen.
  * @param   x: the center x position.
  * @param   y: the center y position.
  * @param   pointColor: the big point color.
  * @retval None
  */
void ARC_LCD_DrawBigPoint(uint16_t x,uint16_t y, uint16_t pointColor)
{
    uint8_t i, j;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            ARC_LCD_SetPixelIndex(x - 1 + i, y - 1 + j, pointColor);
}

/**
  * @brief   draw a cross on the specific position on the LCD screen.
  * @param   x: the center x position.
  * @param   y: the center y position.
  * @param   pointColor: the point color of the cross.
  * @retval None
  */
void ARC_LCD_DrawCross(uint16_t x, uint16_t y, uint16_t pointColor)
{
    int8_t i;
    for(i = -5; i <= 5; i++)
        ARC_LCD_SetPixelIndex(x + i, y, pointColor);

    for(i = -5; i <= 5; i++)
        ARC_LCD_SetPixelIndex(x, y + i, pointColor);
}


/**
  * @brief   show a character on the LCD screen.
  * @param   *LCDParam, the LCD parameters.
  * @retval None
  */
void ARC_LCD_ShowChar(int32_t x, int32_t y, const uint8_t *lcd_font)
```

```
{
    uint8_t point;
    uint8_t v, h;
    uint8_t font;

    font = *(lcd_font) - ' ';

    for(v = 0; v < 12; v++)
    {
        point = asc2_1206[font][v]; /* using 1206 font */
        for(h = 0; h < (12 / 2); h++)
        {
            if(point & 0x01)
                ARC_LCD_SetPixelIndex(x + h, y + v,
LCD_COLOR_WHITE);

            point >>= 1;
        }
    }
}

/**
  * @brief   show strings on the LCD screen.
  * @param   *LCDParam, the LCD parameters.
  * @retval None
  */
void ARC_LCD_ShowString(int32_t x, int32_t y, const uint8_t *lcd_font)
{
    while(*lcd_font != '\0')
    {
        ARC_LCD_ShowChar(x, y, lcd_font);
        x += 12 / 2;
        lcd_font++;
    }
}

void ARC_LCD_Clear   (uint16_t lcdColor)
{
    ARC_LCD_CS_RESET();
    ARC_LCD_WriteGRAM_Same(lcdColor, 240 * 320 - 1);
    ARC_LCD_CS_SET();
}

#endif
```

文件 ARC_RCC.c

```
/**
  * @brief  Configures LCD clocks.
  * @param  None
  * @retval None
  */
void ARC_LCD_RCC_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB |
RCC_APB2Periph_GPIOC |
                           RCC_APB2Periph_GPIOD |
RCC_APB2Periph_AFIO, ENABLE);
}
```

文件 ARC_GPIO.c

```
/**
  * @brief  Configures LCD GPIO ports.
  * @param  None
  * @retval None
  */

/*
        ----------------------
        | DATA OUTPUT | PB     |
        ----------------------
        | LCD_RD      | PC12 |
        ----------------------
        | LCD_WR      | PC11 |
        ----------------------
        | LCD_RS      | PC10 |
        ----------------------
        | LCD_CS      | PD2    |
        ----------------------
        | LCD_BL      | PC5    |
        ----------------------
*/
void ARC_LCD_GPIO_Init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /* Configure as push-pull output */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11 |
                                  GPIO_Pin_12 | GPIO_Pin_5;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    /* disable JTAG, SW still enabled */
    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
    /* Configure port B as push-pull output */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```