ARC

uCGUI

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20章 uCGUI

20.1 uCGUI 简介

uCGUI 是 Micrium 公司设计的一款图形用户界面（GUI），它独立于 MCU 和 LCD 控制器，可以使用在任何的使用 LCD 显示的应用上。所有代码使用 ANSI C 编写。可惜使用上要授权，非免费开源软件。

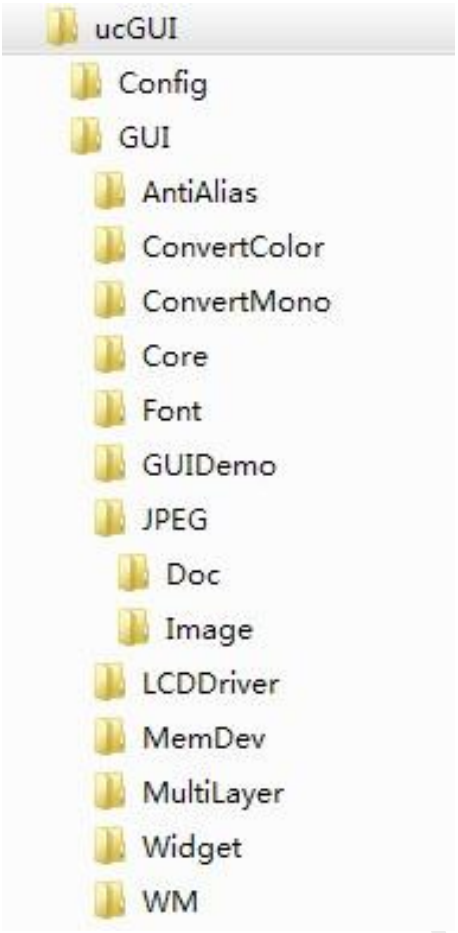
一般特性如下：

- 适用任何 8/16/32 位 CPU，只要有相对应的标准 C 编译器。
- 任何的控制器的 LCD 显示器(单色,灰度,颜色)，只要有适合的 LCD 驱动可用。
- 在小模式显示时无须 LCD 控制器。
- 所有接口支持使用宏进行配制。
- 显示尺寸可定制。
- 字符和位图可在 LCD 显示器上的任意起点显示，并不仅局限于偶数对齐的地址起点。
- 程序在大小和速度上都进行了优化。
- 编译时允许进行不同的优化。
- 对于缓慢一些的 LCD 控制器，LCD 显存可以映射到内存当中，从而减少访问次数到最小并达到更高的显示速度。
- 清晰的设计架构。
- 支持虚拟显示，虚拟显示可以比实际尺寸大。

20.2 uCGUI 移植

20.2.1 源码文件目录组织

首先得到 uCGUI 源代码，然后按以下的目录结构组织。

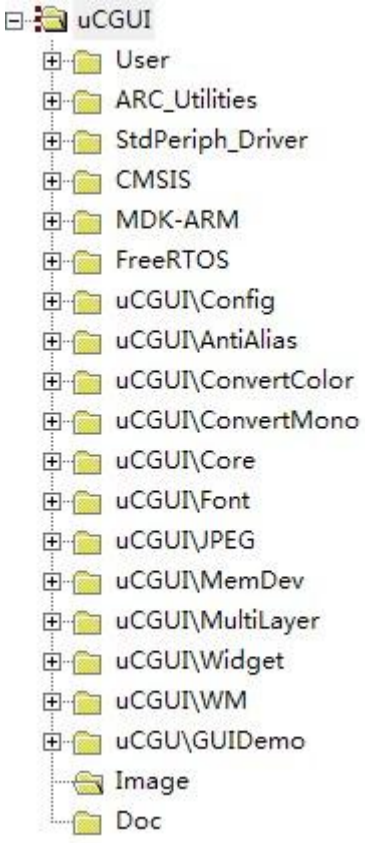


各个子目录的说明如下：

config	配置文件
GUI\AntiAlias	抗锯齿支持（可选）
GUI\ConvertMono	用于灰度显示的色彩转换程序（可选）
GUI\ConvertColor	用于彩色显示的色彩转换程序（可选）
GUI\Core	uCGUI 核心文件
GUI\Font	字体文件
GUI\LCDDriver	LCD 驱动
GUI\MemDev	存储器件支持（可选）
GUI\Widget	视窗控件库（可选）
GUI\WM	视窗管理器（可选）
GUI\JPEG	JPEG 解码（可选）

20.2.2 工程文件组织

我们新建一个 MDK 工程，组织如下：



把对应的文件添加到 MDK 工程文件内。
在工程文件内你需要包含以下目录（不分先后顺序）：

- Config
- GUI\Core
- GUI\Widget (如果使用了视窗控件库)
- GUI\WM (如果使用了视窗管理器)

20.2.3 底层配置（LCDConf.h）

根据你的 LCD 控制器的具体情况配置，我的配置如下：

```
#ifndef LCDCONF_H
#define LCDCONF_H

#define LCD_XSIZE          (240) /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE          (320) /* Y-resolution of LCD, Logical coor. */
#define LCD_CONTROLLER          0x9325
#define LCD_BITSPERPIXEL         (16)
#define LCD_FIXEDPALETTE         (565)
#define LCD_SWAP_RB              (1)

#endif /* LCDCONF_H */
```

20.2.4 上层配置（GUIConfig.h）

根据你需要打开的模块具体配置，我的配置如下：

```
#ifndef GUICONF_H
#define GUICONF_H

#define GUI_OS                          (1)   /* Compile with multitasking
support */
#define GUI_SUPPORT_TOUCH               (1)   /* Support a touch screen
(req. win-manager) */
#define GUI_SUPPORT_MOUSE               (0)   /* Support a mouse */
#define GUI_SUPPORT_UNICODE             (1)   /* Support mixed
ASCII/UNICODE strings */

#define GUI_DEFAULT_FONT                &GUI_Font6x8
#define GUI_ALLOC_SIZE                  (5 * 1024)   /* Size of dynamic
memory ... For WM and memory devices*/

/*********************************************************************
*
*             Configuration of available packages
*/

#define GUI_WINSUPPORT                  1   /* Window manager package
available */
#define GUI_SUPPORT_MEMDEV              1   /* Memory devices available */
#define GUI_SUPPORT_AA                  1   /* Anti aliasing available */

#endif   /* Avoid multiple inclusion */
```

20.2.5 LCD 驱动移植

在移植 LCD 驱动之前，建议在此之前你能初始化 LCD，画简单的点，线，面。成功后在把它关联到 uCGUI 函数名。在这里我使用的是函数别名关联。实现了如下的 uCGUI LCD 驱动，参考文件 ARC_LCD.c。具体代码参考本章软件设计一节。

```
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")));
```

20.2.6 触摸屏移植

参考文件 ARC_TouchScreen.c，最主要的两个函数为

1. void ARC_TouchScreen_start(void)
2. uint8_t ARC_TouchScreen_Rd_LCD_XY(void)

第一个函数用于校正触摸屏，关于如何矫正触摸屏，请参考第 16 章，触摸屏_TouchScreen。

第二个函数为读取转换过后的在 LCD 上对应的点。uCGUI 函数 void GUI_TOUCH_Exec(void) 会调用该函数。

具体代码参考本章软件设计一节。

20.2.7 上层移植

移植了如下函数，本实例使用了嵌入式操作系统 FreeRTOS。具体代码参考本章软件设计一节。

int GUI_X_GetTime (void)， void GUI_X_Delay (int period)， void GUI_X_ExecIdle (void)， void GUI_X_InitOS (void)，void GUI_X_Lock (void)，void GUI_X_Unlock (void)，U32 GUI_X_GetTaskId (void)，void GUI_X_Init (void)，void GUI_X_Log(const char *s)，void GUI_X_Warn(const char *s)，void GUI_X_ErrorOut(const char *s)

20.3 uCGUI 应用实例 ----- FreeRTOS + uCGUI

20.3.1 实例描述

本实例使用了 FreeRTOS 和 uCGUI，实现了 LCD 显示和触摸两大功能，主要创建了 三个任务：

1. 初始化任务，初始化所有的硬件设备（除了 LCD 显示，它会在 UCGUI 内初始化），并创建 UCGUI 任务。该任务完成所有的初始化和任务创建后，将会销毁。所以只是在启动时占用 CPU。
2. UCGUI 任务用于显示，一旦 LCD 初始化成功后，再创建触摸任务，因为只有能显示，触摸任务才有必要创建。该任务一直在运行。
3. 触摸屏任务，该任务大部分时间处于挂起状态，不会消耗 CPU 资源，只有触摸笔按下时，该任务才会处于运行状态，和 UCGUI 分时占用 CPU 资源。

为了实现这个功能，我们需要创建一个信号量，使能一个笔中断函数。笔中断函数 give semaphore，触摸屏处理任务 take semaphore，所以只有笔中断发生了，触摸屏才能 take semaphore，处于运行状态，这样子，触摸屏任务只要在笔按下时，才会占用 CPU， 极大的减轻了 CPU 的负担。

uCGUI 原本的 demo 使用定时器，不管有没有按下笔，定时查询，消耗了很多 CPU 资源。

20.3.2 硬件设计

参考第 15 章，LCD。

20.3.3 软件设计

文件 uCGUI_main.c

```
/**
  * @brief   Initialize take.
  */
void InitTask ( void *pvParameters )
{
    portBASE_TYPE task_error;
    ARC_Button_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);

    task_error = xTaskCreate(uCGUI, "uCGUI task", (1 * 1024), NULL, 8,
NULL );
    assert( task_error == pdPASS);
```

```
    vTaskDelete(NULL);
}
```

文件 GUIDEMO.c：

```
/**
  * @brief   Main program, the touch screen example.
  * @param   None
  * @retval None
  */
int main(void)
{

    portBASE_TYPE task_error;
    task_error = xTaskCreate(InitTask, "Init task", (100), NULL, 5, NULL );

    assert( task_error == pdPASS);
    vTaskStartScheduler();
    assert(0);
}

void uCGUI ( void *pvParameters )
{
    portBASE_TYPE task_error;
    GUI_Init();
    ARC_TouchScreen_Init();
    SPI_Cmd(SPI1, ENABLE);
    ARC_TouchScreen_start();
    task_error = xTaskCreate(Touchscreen, "TS task", (500), NULL, 8,
NULL );
    assert( task_error == pdPASS);

    while(1)
    {
        GUIDEMO_main();
        ARC_SysTick_Delay(100);
    }

}
```

文件 ARC_TouchScreen.c：

```
/**
  ******************************************************************************
  * @file      ARC_TouchScreen.c
  * @author    armrunc (www.armrunc.com)
```

```
  * @version V1.0.0
  * @brief     ARC middleware.
  *                This file provides Touch screen middleware functions.
  ******************************************************************************
  * @copy
  *
  * For non-commercial research and private study only.
  *
  * <h2><center>&copy; COPYRIGHT www.armrunc.com </center></h2>
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include "ARC_TouchScreen.h"
#include "ARC_SPI.h"
#include "ARC_SPI_Flash.h"
#include "ARC_GPIO.h"
#include "ARC_RCC.h"
#include "ARC_EXTI.h"
#include "ARC_NVIC_API.h"
#include "ARC_LCD.h"
#include "ARC_SysTick.h"
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#if (defined ARC_UCGUI && defined ARC_FREERTOS)
#include "LCD_Private.h"        /* private modul definitions & config */
#include "GUI_Private.h"
#include "GUIDebug.h"
#endif


/** @addtogroup Utilities
  * @{
  */

/** @addtogroup ARC_TouchScreen
  * @{
  */

/** @defgroup ARC_TouchScreen_Private_TypesDefinitions
  * @{
  */
```

```
/**
  * @}
  */

/** @defgroup ARC_TouchScreen_Private_Defines
  * @{
  */

#define ARC_TS_XPT2046_X            0xD0
#define ARC_TS_XPT2046_Y            0x90
#define ARC_TS_XPT2046_DUMMY        0x0

#define ARC_TS_READ_TIMES           16
#define ARC_TS_IGNORE_TIMES         5 /* ignore twice, up and down each
*/

#define ARC_DRAW_RANGE              5
#define ARC_MAGIC                   0xFEDCBA0
#define ARC_TS_FLASH_ADDR           (256 * 16)

#define ARC_TS_CAL_X_LEFT           20
#define ARC_TS_CAL_X_RIGHT          (LCD_XSIZE - 1 -
ARC_TS_CAL_X_LEFT)
#define ARC_TS_CAL_Y_BOTTOM         20
#define ARC_TS_CAL_Y_TOP            (LCD_YSIZE - 1 -
ARC_TS_CAL_Y_BOTTOM)

/**
  * @}
  */

/** @defgroup ARC_TouchScreen_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @defgroup ARC_TouchScreen_Private_Variables
  * @{
  */
```

```
#if (defined ARC_UCGUI && defined ARC_FREERTOS)
xSemaphoreHandle xBinarySemaphore_ts;
#endif

static pen_state_struct pen_state;

/**
  * @}
  */

/** @defgroup ARC_TouchScreen_Private_FunctionPrototypes
  * @{
  */

/**
  * @}
  */

/** @defgroup ARC_TouchScreen_Private_Functions
  * @{
  */
#if (defined ARC_UCGUI && defined ARC_FREERTOS)

/**
  * @brief   Initialize a binary semaphore for TS.
  * @param   None
  * @retval None
  */
void ARC_TouchScreen_OS_Init(void)
{
    vSemaphoreCreateBinary( xBinarySemaphore_ts );
}

/**
  * @brief   the touch screen task.
  * @param   pvParameters
  * @retval None
  */
void Touchscreen ( void *pvParameters )
{
    while(1)
    {
        StoreUnstable_Invalid();
        xSemaphoreGive(xBinarySemaphore_ts);
```

```
            while(!ARC_PEN_STATE())
            {
                GUI_TOUCH_Exec();
                ARC_SysTick_Delay(10); /*recommanded 100 times per second
by uCGUI */
            }
        }
}

#endif

/**
   * @brief    initialize the LCD struct.
   * @param    None
   * @retval None
   */
void ARC_TouchScreen_Struct_Init(void)
{
      pen_state_struct *pen_st;
      pen_st = ARC_get_penstate();
      memset((void*) pen_st, 0, sizeof(pen_state_struct));
}

/**
   * @brief    Initialize TouchScreen.
   * @param    None
   * @retval None
   */
void ARC_TouchScreen_Init(void)
{
      ARC_SPI_Init();
      ARC_TouchScreen_RCC_Init();
      ARC_TouchScreen_GPIO_Init();
      ARC_TouchScreen_NVIC_Init();
      ARC_TouchScreen_EXTI_Init();
      #if (defined ARC_UCGUI && defined ARC_FREERTOS)
      ARC_TouchScreen_OS_Init();
      #endif
      ARC_TouchScreen_Struct_Init();
}

/**
   * @brief    get pen_state.
```

```
   * @param    None
   * @retval the pointer to the pen_state
   */
pen_state_struct *ARC_get_penstate(void)
{
      return &pen_state;
}

/**
   * @brief    start up the TouchScreen, do adjustment here.
   * @param    None
   * @retval None
   */
void ARC_TouchScreen_start(void)
{
      ARC_SPI_FLASH_ID_check();
      ARC_TouchScreen_Adjust();
}

/**
   * @brief    read the original position on the touch screen.
   * @param    None
   * @retval None
   */
void ARC_TouchScreen_Rd_Phisical_XY(void)
{
      uint8_t i, j, xy_index;
      uint16_t value;
      uint16_t buff[2][ARC_TS_READ_TIMES];
      uint16_t sum[2] = {0, 0};
      uint16_t temp;
      pen_state_struct *pen_st;

      pen_st = ARC_get_penstate();

      EXTI_SetInt(EXTI_Line6, 0);

      SPI_BaudRateConfig(SPI1, ARC_SPI_XPT2046_SPEED);
      ARC_TS_CS_LOW();

      for (i = 0; i < ARC_TS_READ_TIMES; i++)
      {
            ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_X);
```

```
            value = ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_DUMMY);

            buff[0][i] = value << 8;
            value = ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_DUMMY);
            buff[0][i] |= value;
            buff[0][i] >>= 3;
            buff[0][i] &= 0XFFF;

            ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_Y);
            value = ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_DUMMY);
            buff[1][i] = value << 8;
            value = ARC_SPI_SendByte(SPI1, ARC_TS_XPT2046_DUMMY);
            buff[1][i] |= value;
            buff[1][i] >>= 3;
            buff[1][i] &= 0XFFF;
        }
        ARC_TS_CS_HIGH();
        SPI_BaudRateConfig(SPI1, ARC_SPI_DEFAULT_SPEED);

        EXTI_SetInt(EXTI_Line6, 1);

        for(xy_index = 0; xy_index < 2; xy_index++)
        {
            for(i = 0;i < ARC_TS_READ_TIMES - 1; i++)
            {
                for(j = i + 1; j < ARC_TS_READ_TIMES; j++)
                {
                    if(buff[xy_index][i] > buff[xy_index][j])
                    {
                        temp =buff[xy_index][i];
                        buff[xy_index][i] = buff[xy_index][j];
                        buff[xy_index][j] = temp;
                    }
                }
            }
            for(i = ARC_TS_IGNORE_TIMES; i < ARC_TS_READ_TIMES -
ARC_TS_IGNORE_TIMES; i++)
                sum[xy_index] += buff[xy_index][i];
            sum[xy_index] = sum[xy_index] / (ARC_TS_IGNORE_TIMES - 2 *
ARC_TS_IGNORE_TIMES);
        }
        pen_st->x = sum[0];
        pen_st->y = sum[1];
}
```

```
/**
   * @brief    read the converted position on the touch screen.
   * @param    None
   * @retval 1 if the converted point is valid, otherwise return 0.
   */
uint8_t ARC_TouchScreen_Rd_LCD_XY(void)
{
      uint8_t ret_value = 1;
      static uint16_t pre_x, pre_y;
      pen_state_struct *pen_st;
      pen_st = ARC_get_penstate();

      ARC_TouchScreen_Rd_Phisical_XY();
      pen_st->x_converted = pen_st->xfac * pen_st->x + pen_st->xoff;
      pen_st->y_converted = pen_st->yfac * pen_st->y + pen_st->yoff;

      if((abs(pre_x - pen_st->x_converted) > ARC_DRAW_RANGE) ||
          (abs(pre_y - pen_st->y_converted) > ARC_DRAW_RANGE))
          ret_value = 0;

      pre_x = pen_st->x_converted;
      pre_y = pen_st->y_converted;

      return ret_value;
}

/**
   * @brief    calculate the distance of the two points.
   * @param    *point1: the first point.
   * @param    *point2: the second point.
   * @retval the distance of the two points..
   */
int32_t ARC_Calc_distance(int32_t *point1, int32_t *point2)
{
      return sqrt((point1[0] - point2[0]) * (point1[0] - point2[0]) +
                  (point1[1] - point2[1]) * (point1[1] - point2[1]));
}

/**
   * @brief    verify if the touch screen is a valid one.
   * @param    pos_temp[][2]: the four points on the touch screen corners.
   * @retval 1 if the touch screen is valid, otherwise return 0.
   */
```

```
uint8_t ARC_Verify_TouchScreen(int32_t pos_temp[][2])
{
	int32_t d1, d2;
	int32_t fac_x100;

	d1 = ARC_Calc_distance(pos_temp[0], pos_temp[1]);
	d2 = ARC_Calc_distance(pos_temp[2], pos_temp[3]);
	fac_x100 = d1 * 100 / d2;
	if(abs(fac_x100 - 100) > 5 || d1 == 0 || d2 == 0)
	{
		return 0;
	}

	d1 = ARC_Calc_distance(pos_temp[0], pos_temp[2]);
	d2 = ARC_Calc_distance(pos_temp[1], pos_temp[3]);
	fac_x100 = d1 * 100 / d2;
	if(abs(fac_x100 - 100) > 5 || d1 == 0 || d2 == 0)
	{
		return 0;
	}

	d1 = ARC_Calc_distance(pos_temp[0], pos_temp[3]);
	d2 = ARC_Calc_distance(pos_temp[1], pos_temp[2]);
	fac_x100 = d1 * 100 / d2;
	if(abs(fac_x100 - 100) > 5 || d1 == 0 || d2 == 0)
	{
		return 0;
	}

	return 1;
}

/**
  * @brief    Do touch screen adjustment.
  * @param    None
  * @retval None
  */
void ARC_TouchScreen_Adjust(void)
{
	uint8_t lcd_drawed = 0;
	int32_t pos_temp[4][2];
	uint8_t    cnt = 0;
	pen_state_struct *pen_st;
	uint32_t FlashAddr;
```

```
    uint8_t Rx_Buffer[8];
    uint8_t *Tx_Buffer;
    uint16_t BufferSize;
    uint32_t magic_num = ARC_MAGIC;
    const int32_t LCD_pos[4][2] = { {ARC_TS_CAL_X_LEFT,
ARC_TS_CAL_Y_BOTTOM},
                                    {ARC_TS_CAL_X_RIGHT,
ARC_TS_CAL_Y_BOTTOM},
                                    {ARC_TS_CAL_X_RIGHT,
ARC_TS_CAL_Y_TOP},
                                    {ARC_TS_CAL_X_LEFT,
ARC_TS_CAL_Y_TOP}};

    pen_st = ARC_get_penstate();

    if(!(pen_st->force_adjust))
    {
        if(spi_flash_found)
        {
            FlashAddr = ARC_TS_FLASH_ADDR;
            BufferSize = sizeof(uint32_t);
            /* Read data from SPI FLASH memory */
            ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr, BufferSize);
            if((*(uint32_t *)Rx_Buffer) == magic_num)
            {
                FlashAddr += BufferSize;
                BufferSize = sizeof(float);
                /* Read data from SPI FLASH memory */
                ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr,
BufferSize);
                pen_st->xfac = *(float *)Rx_Buffer;

                FlashAddr += BufferSize;
                BufferSize = sizeof(float);
                /* Read data from SPI FLASH memory */
                ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr,
BufferSize);
                pen_st->xoff = *(float *)Rx_Buffer;

                FlashAddr += BufferSize;
                BufferSize = sizeof(float);
                /* Read data from SPI FLASH memory */
                ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr,
BufferSize);
```

```
                    pen_st->yfac = *(float *)Rx_Buffer;

                    FlashAddr += BufferSize;
                    BufferSize = sizeof(float);
                    /* Read data from SPI FLASH memory */
                    ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr,
BufferSize);
                    pen_st->yoff = *(float *)Rx_Buffer;

                    return;
                }
            }
        }

        pen_st->pen_pressed = KEY_UP;

        while(1)
        {
            if(!lcd_drawed)
            {
                lcd_drawed = 1;
                #if (defined ARC_UCGUI && defined ARC_FREERTOS)
                GUI_SetBkColor(GUI_RED);
                GUI_Clear();
                GUI_SetColor(GUI_WHITE);
                GUI_DrawCircle(LCD_pos[cnt][0], LCD_pos[cnt][1], 5);
                GUI_SetColor(GUI_WHITE);
                GUI_DrawHLine(LCD_pos[cnt][1], LCD_pos[cnt][0] - 10,
LCD_pos[cnt][0] + 10);
                GUI_DrawVLine(LCD_pos[cnt][0], LCD_pos[cnt][1] - 10,
LCD_pos[cnt][1] + 10);
                GUI_SetColor(GUI_WHITE);
                GUI_SetTextAlign(GUI_TA_CENTER);
                GUI_DispStringAt("touchscreen calculation", LCD_XSIZE / 2,
LCD_YSIZE / 2 - 20);
                GUI_SetTextAlign(GUI_TA_CENTER);
                GUI_DispStringAt("press the middle of the cross", LCD_XSIZE /
2, LCD_YSIZE / 2 + 20);
                #else
                ARC_LCD_Clear(LCD_COLOR_RED);
                ARC_LCD_DrawCross(LCD_pos[cnt][0], LCD_pos[cnt][1],
LCD_COLOR_WHITE);
                #endif
            }
```

```
        if(pen_st->pen_pressed == KEY_DOWN)
        {
            lcd_drawed = 0;
            pen_st->pen_pressed = KEY_UP;
            ARC_TouchScreen_Rd_Phisical_XY();
            pos_temp[cnt][0] = pen_st->x;
            pos_temp[cnt][1] = pen_st->y;

            cnt++;

            if(cnt == 4)
            {
                if(ARC_Verify_TouchScreen(pos_temp) == 0)
                {
                    cnt = 0;
                }
                else
                {
                    if((pen_st->ts_direction == force_reverse) ||
                       (pen_st->ts_direction == auto_detect &&
                        abs(pos_temp[1][0] - pos_temp[0][0]) <
abs(pos_temp[1][1] - pos_temp[0][1])))
                    {
                        uint8_t i;
                        int32_t temp;
                        for(i = 0; i < 4; i++)
                        {
                            temp = pos_temp[i][0];
                            pos_temp[i][0] = pos_temp[i][1];
                            pos_temp[i][1] = temp;
                        }
                    }

                    pen_st->xfac = (float)(LCD_pos[1][0] - LCD_pos[0][0]) /
(pos_temp[1][0] - pos_temp[0][0]);
                    pen_st->xoff = ((LCD_pos[1][0] + LCD_pos[0][0]) -
pen_st->xfac * (pos_temp[1][0] + pos_temp[0][0])) / 2;

                    pen_st->yfac = (float)(LCD_pos[2][1] - LCD_pos[0][1]) /
(pos_temp[2][1] - pos_temp[0][1]);
                    pen_st->yoff = ((LCD_pos[2][1] + LCD_pos[0][1]) -
pen_st->yfac * (pos_temp[2][1] + pos_temp[0][1])) / 2;
                    if (spi_flash_found)
                    {
```

```
                    FlashAddr = ARC_TS_FLASH_ADDR;
                    /* Erase SPI FLASH Sector to write on */
                    ARC_FLASH_EraseSector(FlashAddr);

                    BufferSize = sizeof(uint32_t);
                    Tx_Buffer = (uint8_t *)(&magic_num);
                    /* Write Tx_Buffer data to SPI FLASH memory */
                    ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr,
BufferSize);

                    FlashAddr += BufferSize;
                    BufferSize = sizeof(float);
                    Tx_Buffer = (uint8_t *)(&(pen_st->xfac));
                    /* Write Tx_Buffer data to SPI FLASH memory */
                    ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr,
BufferSize);

                    FlashAddr += BufferSize;
                    BufferSize = sizeof(float);
                    Tx_Buffer = (uint8_t *)(&(pen_st->xoff));
                    /* Write Tx_Buffer data to SPI FLASH memory */
                    ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr,
BufferSize);

                    FlashAddr += BufferSize;
                    BufferSize = sizeof(float);
                    Tx_Buffer = (uint8_t *)(&(pen_st->yfac));
                    /* Write Tx_Buffer data to SPI FLASH memory */
                    ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr,
BufferSize);

                    FlashAddr += BufferSize;
                    BufferSize = sizeof(float);
                    Tx_Buffer = (uint8_t *)(&(pen_st->yoff));
                    /* Write Tx_Buffer data to SPI FLASH memory */
                    ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr,
BufferSize);
                }
                return;
            }
        }
    }
}
```

```
/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */

/****************** (C) www.armrunc.com *****END OF FILE****/
```

文件 ARC_LCD.c

```
/**
  ******************************************************************************
  * @file       ARC_LCD.c
  * @author    armrunc (www.armrunc.com)
  * @version V1.0.0
  * @brief     ARC middleware.
  *                This file provides LCD middleware functions.
  ******************************************************************************
  * @copy
  *
  * For non-commercial research and private study only.
  *
  * <h2><center>&copy; COPYRIGHT www.armrunc.com </center></h2>
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include "ARC_LCD.h"
#include "ARC_SysTick.h"
#include "ARC_Retarget.h"
#include "ARC_Font.h"
#include "ARC_GPIO.h"
#include "ARC_RCC.h"
#if (defined ARC_UCGUI && defined ARC_FREERTOS)
#include "LCD_Private.h"         /* private modul definitions & config */
#include "GUI_Private.h"
#include "GUIDebug.h"
#include <string.h>
```

```
#endif

/** @addtogroup Utilities
  * @{
  */

/** @addtogroup ARC_LCD
  * @{
  */

/** @defgroup ARC_LCD_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @defgroup ARC_LCD_Private_Defines
  * @{
  */
#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;\
}

/**
  * @}
  */

/** @defgroup ARC_LCD_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @defgroup ARC_LCD_Private_Variables
  * @{
  */
static ARC_LCD_Params ARC_LCD_Param;

/**
  * @}
  */

/** @defgroup ARC_LCD_Private_FunctionPrototypes
  * @{
  */

/**
  * @}
  */

/** @defgroup ARC_LCD_Private_Functions
  * @{
  */

/**
  * @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

/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */

/****************** (C) www.armrunc.com *****END OF FILE****/
```