ARC

触摸屏

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16章 触摸屏

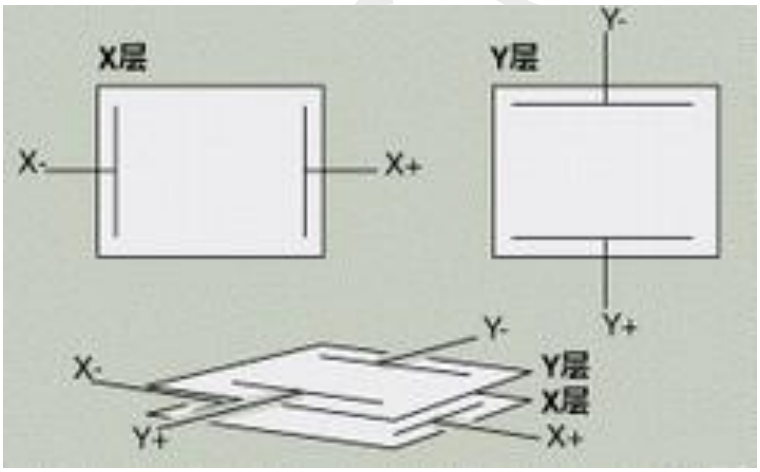
16.1 触摸屏简介

本平台使用电阻触摸屏，它利用压力感应进行控制。电阻触摸屏的主要部分是一块与显示器表面非常配合的电阻薄膜屏，这是一种多层的复合薄膜，它以一层玻璃或硬塑料平板作为基层，表面涂有一层透明氧化金属（透明的导电电阻）导电层，上面再盖有一层外表面硬化处理、光滑防擦的塑料层、它的内表面也涂有一层涂层、在他们之间有许多细小的（小于 1/1000 英寸）的透明隔离点把两层导电层隔开绝缘。

当手指触摸屏幕时，平常绝缘的两层导电层在触摸点位置就有了一个接触，控制器侦测到这个接通后，其中一面导电层接通 y 轴方向的 VREF 均匀电压场，另一导电层将接触点的电压引至控制卡进行 A/D 转换，得到电压值后与 VREF 相比即可得触摸点的 y 轴坐标，同理得出 x 轴的坐标，这就是所有电阻技术触摸屏共同的最基本原理。

本平台使用电阻屏中的四线触摸屏，包含两个阻性层。其中一层在屏幕的左右边缘各有一条垂直总线，另一层在屏幕的底部和顶部各有一条水平总线。为了在 X 轴方向进行测量，将左侧总线偏置为 0V，右侧总线偏置为 VREF。将顶部或底部总线连接到 ADC，当顶层和底层相接触时即可作一次测量。

为了在 Y 轴方向进行测量，将顶部总线偏置为 VREF，底部总线偏置为 0V。将 ADC 输入端接左侧总线或右侧总线，当顶层与底层相接触时即可对电压进行测量。



使用的触摸屏控制芯片为 XP2046，它是一款 4 导线制触摸屏控制器，内含 12 位分辨率 125KHz 转换速率逐步逼近型 A/D 转换器。XPT2046 支持从 1.5V 到 5.25V 的低电压 I/O 接口。XPT2046 能通过执行两次 A/D 转换查出被按的屏幕位置， 除此之外，还可以测量加在触摸屏上的压力。内部自带 2.5V 参考电压可以作为辅助输入、温度测量和电池监测模式之用，电池监测的电压范围可以从 0V 到 6V。XPT2046 片内集成有一个温度传感器。 在 2.7V 的典型工作状态下，关闭参考电压，功耗可小于 0.75mW。XPT2046 的工作温度范围为-40℃～+85℃。

16.2 触摸屏应用实例 ----- 在触摸屏上画图

16.2.1 实例描述

本实例实现了 LCD 屏幕画图功能，左上角有个 clear 按键，如果按住这个区域，将会清屏。如果触摸屏没有校正，首先会在屏幕的左上角出现一个十字，按住这个十字中心，放开，将会在屏幕另外角绘制十字，按住十字中心，如此重复，直到屏幕四个角的十字全部按下后会开始自动校正。另外如果之前做过自动校正，也可以按下 KEY0 或者 KEY1 实现自动校正。

可以实现自动识别触摸屏方向，你也可以通过配置 pen_state 的 ts_direction 固定触摸屏的方向。

16.2.2 触摸屏校正原理

假设 LCD 屏幕坐标为 (xLCD, yLCD)，触摸屏的显示坐标为 (xTS, yTS)，它们的关系为线性关系，所以转换公式如下：

xLCD = xfac * xTS + xoff;

yLCD = yfac * yTS + yoff;

未知数为 xfac, xoff, yfac, yoff，所以我们需要四个方程式解这四个值。方法是在屏幕的四个角落固定位置画四个十字，所以我们得到了四个点的 xLCD 和 yLCD 值，当用户按下这四个点时，我们可以读出四个点对应的 xTS 和 yTS 值，根据以下公式，我们求得四个未知数，xfac, xoff, yfac, yoff。

xfac = (LCD_pos[1][0] - LCD_pos[0][0]) / (TS_pos[1][0] – TS_pos[0][0]);

xoff = ((LCD_pos[1][0] + LCD_pos[0][0]) - xfac * (TS_pos[1][0] + TS_pos[0][0])) / 2;

yfac = (LCD_pos[2][1] - LCD_pos[0][1]) / (TS_pos[2][1] – TS_pos[0][1]);

yoff = ((LCD_pos[2][1] + LCD_pos[0][1]) - yfac * (TS_pos[2][1] + TS_pos[0][1])) / 2;

在这之前，我们会判断 LCD 显示的方向是否和触摸屏的方向一致。

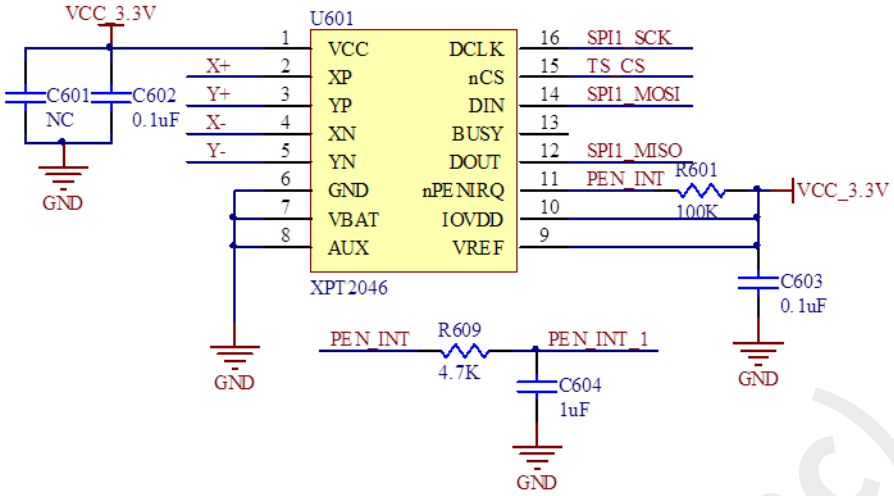
在屏幕校正完了以后，为了下次启动不再需要校正，我们将这四个值存储在掉电数据保存的外部 SPI FLASH， 在这四个值之前我们用一个 magic number 来指示接下来的是触摸屏校正数据，具体代码如下。

16.2.3 硬件设计

本实例用到了 LCD 显示，请参考 LCD 一章。触摸电路 GPIO 配置和原理图如下：

TS_CS	PC7
PEN_INT	PC6
SPI1_SCK	PA5

SPI1_MISO	PA6
SPI1_MOSI	PA7



16.2.4 软件设计

本实例先初始化 SysTick 用来精确延迟，初始化串口用来调试输出，然后初始化 LCD。再初始化 SPI flash 用来存储触摸屏校验数据。再初始化触摸屏幕。本实例的主要代码如下：

文件 TouchScreen_main.c

```
/**
  * @brief   Main program, the touch screen example.
  * @param   None
  * @retval None
  */

int main(void)
{
    pen_state_struct *pen_st;
    pen_st = ARC_get_penstate();
    ARC_SysTick_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);
    ARC_LCD_Init();
    ARC_LCD_On();
    ARC_TouchScreen_Init();
    SPI_Cmd(SPI1, ENABLE);
    ARC_TouchScreen_start();

    ARC_Button_Init();
```

```
    ARC_LCD_Clear(LCD_COLOR_RED);
    ARC_LCD_ShowString(3, 3, "clear");

    while(1)
    {
        if (pen_st->force_adjust == 1)
        {
            ARC_TouchScreen_Adjust();
            pen_st->force_adjust = 0;
            ARC_LCD_Clear(LCD_COLOR_RED);
            ARC_LCD_ShowString(3, 3, "clear");
        }
        if (pen_st->pen_pressed == KEY_DOWN)
        {
            pen_st->pen_pressed = KEY_UP;
            while(!ARC_PEN_STATE())
            {
                if(ARC_TouchScreen_Rd_LCD_XY())
                {
                    if((abs(pen_st->x_converted - 20) < 18) &&
                       (abs(pen_st->y_converted - 8) < 6))
                    {
                        ARC_LCD_Clear(LCD_COLOR_RED);
                        ARC_LCD_ShowString(3, 3, "clear");
                    }
                    else if((abs(pen_st->x_converted - LCD_XSIZE / 2) <
LCD_XSIZE / 2) &&
                            (abs(pen_st->y_converted - LCD_YSIZE / 2) <
LCD_YSIZE / 2))
                    {
                        ARC_LCD_DrawBigPoint(pen_st->x_converted,
pen_st->y_converted, LCD_COLOR_WHITE);
                    }
                }
            }
        }
    }
}
```

文件 ARC_TouchScreen.c：

```
#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)

static pen_state_struct pen_state;

/**
  * @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;
                        }
                    }
                }
            }
}
```

文件 ARC_RCC.c

```
/**
  * @brief  Configures Touch screen clocks.
  * @param  None
  * @retval None
  */
void ARC_TouchScreen_RCC_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_GPIOC |
                           RCC_APB2Periph_AFIO, ENABLE);

    /*!< FLASH_SPI Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
}
```

文件 ARC_GPIO.c

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

/*
        ---------------
        | CLK   | PA5   |
        ---------------
        | MISO | PA6   |
        ---------------
        | MOSI | PA7   |
        ---------------
        | CS    | PC7   |
        ---------------
        | PEN   | PC6   |
        ---------------
```

```
*/
void ARC_TouchScreen_GPIO_Init()
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*!< Configure SCK */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure MOSI */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure MISO */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure CS pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /*!< Configure PEN interrupt pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

文件 ARC_NVIC_API.c：

```
/**
  * @brief   Initialize NVIC of pen of touch screen.
  * @param   None
  * @retval None
  */
void ARC_TouchScreen_NVIC_Init(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
```

```
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

文件 ARC_EXTI.c：

```
/**
  * @brief   Initialize EXTI of touch screen.
  * @param   None
  * @retval None
  */
void ARC_TouchScreen_EXTI_Init(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;

    /* Connect Button EXTI Line to Button GPIO Pin */
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource6);

    /* Configure Button EXTI line */
    EXTI_InitStructure.EXTI_Line = EXTI_Line6;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
}
```

文件 stm32f10x_it.c：

```
/**
  * @brief   This function handles External interrupt Line 2 request.
  * @param   None
  * @retval None
  */
void EXTI2_IRQHandler(void)
{
#ifdef ARC_TOUCHSCREEN
    pen_state_struct *pen_st;
    if(EXTI_GetITStatus(EXTI_Line2) != RESET)
    {
        pen_st = ARC_get_penstate();
        pen_st->force_adjust = 1;
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
#endif
}
```

```
/**
  * @brief  This function handles External interrupt Line 3 request.
  * @param  None
  * @retval None
  */
void EXTI3_IRQHandler(void)
{
#ifdef ARC_TOUCHSCREEN
    pen_state_struct *pen_st;
    if(EXTI_GetITStatus(EXTI_Line3) != RESET)
    {
        pen_st = ARC_get_penstate();
        pen_st->force_adjust = 1;
        EXTI_ClearITPendingBit(EXTI_Line3);
    }
#endif
}
```