Technical Note

基本型 MCU 例程_V1.0

工程技术文档

类别	内容
关键词	基本型 MCU 程序
摘要	

修订历史

版本	日期	原因	编制	审查
V1.0	2018/6/6	创建文档	林青田	

销售与服务

广州大彩光电科技有限公司

电话：020-82186683

传真：020-82187676

Email：hmi@gz-dc.com（公共服务）

网站：http://www.gz-dc.com

地址：广州高新技术产业开发区玉树工业园富康西街 8 号 C 栋 303 房

官网零售淘宝店：https://gz-dc.taobao.com

目录

1. 程序架构介绍

我司提供了串口屏驱动代码和范例程序，目前支持的单片机平台有 51、 STM32。我司例程上机测试的单片机是 STM32F103VCT6 和 STC89CX 系列，用户可以直接修改范例程序，参考程序上已有的功能，然后添加并修改自己的功能代码（例如温湿采集、开关控制等）。此文档所引用的程序出自 STM32 的例程，但 51 实现功能代码也是一样的，不同的只是定时器和串口的配置。

指令的解析和处理流程完全由驱动代码提供。范例程序结构如下图 1-1 所示：

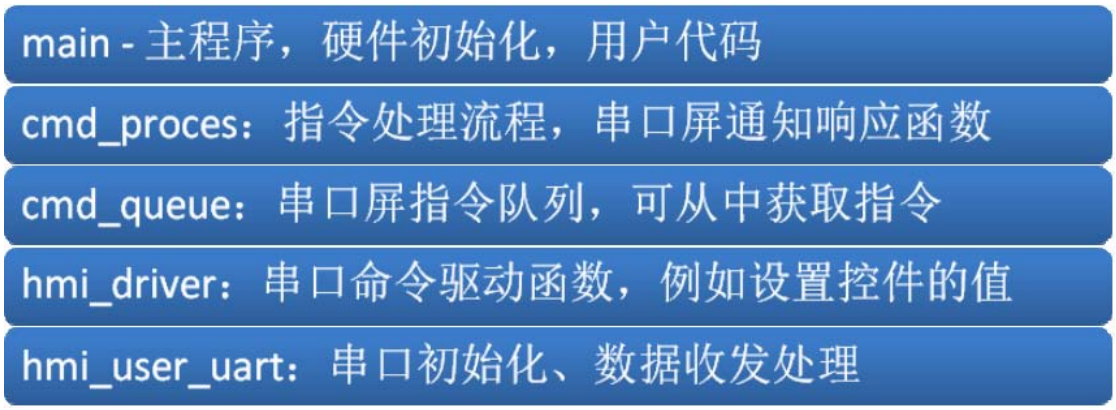

图 1-1 程序结构

2. 例程分析

2.1 程序工作流程图

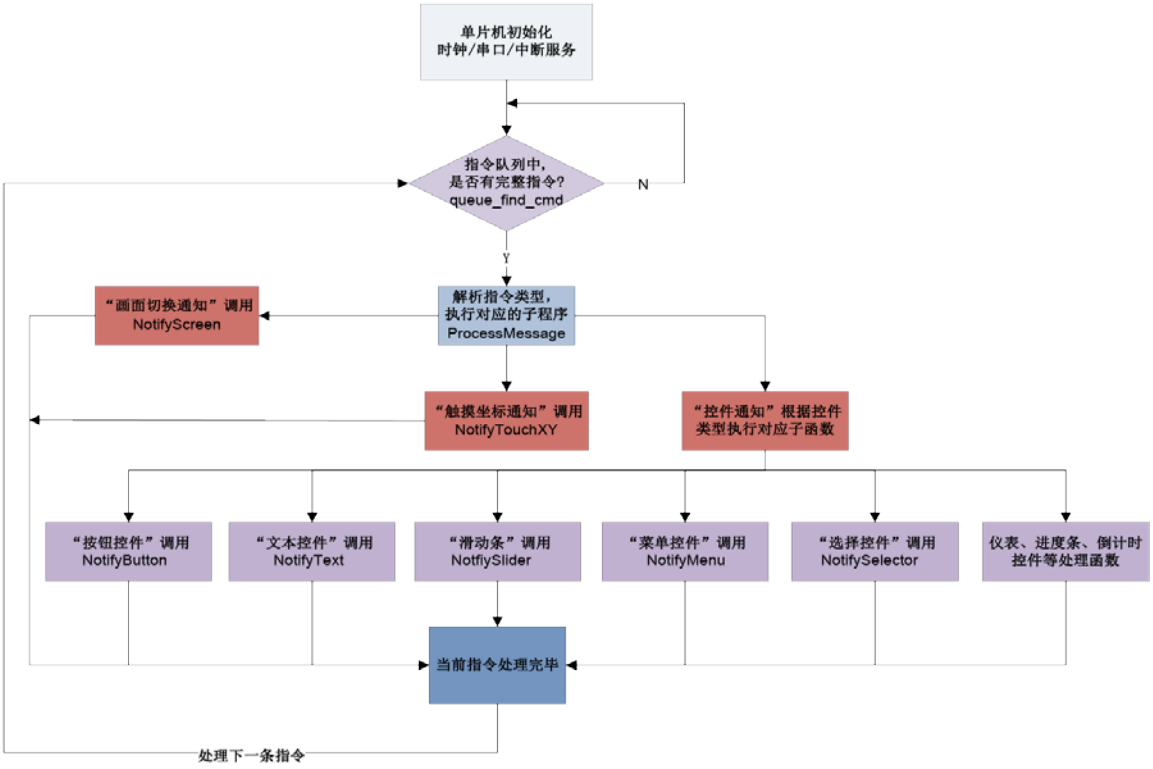

图 2-1 程序工作流程图

2.1 串口屏下发指令

每当对串口屏进行操作，串口屏都会下发相对应的指令，如切换画面、文本控件输入、进度条值改变等等的操作都会发出对应控件值变化的指令通知 MCU。

2.2 指令

当串口屏与 MCU 通过串口连接后，它们间的通讯依靠指令。而指令是我司为了方便和规范数据交互而定的协议，具体各种功能的指令可以参考指令集文档。

2.2.1 指令解析过程

MCU 的中断函数接收到指令后，将指令存储到指令缓冲区，在 main 函数的 while 循环中，queue_find_cmd 会一直检测指令缓冲区，若缓冲区收到指令，立即进行提取；接着调用 ProcessMessage 函数解析指令，然后调用对应指令类型的函数并将指令的数据作为参数传进调用的函数。

2.2.2 接收指令

MCU 中断接收串口屏发送的指令到缓冲区

```
void USART1_IRQHandler(void)
{
```

```
        if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
        {
            uint8_t data = USART_ReceiveData(USART1);
            queue_push(data);                          //接收数据到缓冲区
        }
    }
```

2.2.3 cmd_queue 串口屏指令队列

指令也理解为数据帧，数据帧一般分为三部分：帧头，数据部分，帧尾。

将指令队列定义为一个结构体 QUEUE，结构体成员包括了队列头，队列尾和队列的数据缓冲区。MCU 开机时调用队列函数 queue_reset 清空队列的缓冲区，防止缓冲区中存在错误的数据。串口屏发送指令到 MCU 后，MCU 的串口中断会调用函数 queue_push 提取数据到队列缓冲区，当队列的缓冲区存在数据后，queue_find_cmd 函数将队列缓冲区的数据调用 queue_pop 一个一个提出来然后拼接成一个完整的指令。

```
#define    CMD_HEAD    0XEE                    //帧头
#define    CMD_TAIL    0XFFFCFFFF              //帧尾
typedef   struct _QUEUE
{
    qsize _head;                               //队列头
    qsize _tail;                               //队列尾
    qdata _data[QUEUE_MAX_SIZE];               //队列数据缓存区
}QUEUE;
static QUEUE que = {0,0,0};                    //指令队列
static uint32 cmd_state = 0;                   //队列帧尾检测状态
static qsize cmd_pos = 0;                      //当前指令指针位置
void queue_reset()
{
    que._head = que._tail = 0;
    cmd_pos = cmd_state = 0;
}
void queue_push(qdata _data)                   //在中断中调用获取指令数据
{
    qsize pos = (que._head+1)%QUEUE_MAX_SIZE;
    if(pos!=que._tail)                         //没有达到缓冲区的上限
    {
        que._data[que._head] = _data;
        que._head = pos;
    }
}
//从队列缓冲区中取一个数据
static void queue_pop( qdata   *   _data)
{
    if(que._tail!=que._head)                   //非空状态
    {
        *_data = que._data[que._tail];
```

```
            que._tail = (que._tail+1)%QUEUE_MAX_SIZE;
        }
    }
    //获取队列中有效数据个数
    static qsize queue_size()
    {
        return ((que._head+QUEUE_MAX_SIZE-que._tail)%QUEUE_MAX_SIZE);
    }
    qsize queue_find_cmd(qdata    *buffer , qsize    buf_len)
    {
        qsize cmd_size = 0;                                    //qsize：typedef unsigned char qdata;
        qdata    _data = 0;                                    //qdata：typedef unsigned short qsize;
        while(queue_size()>0)
        {
            //取一个数据
            queue_pop(&_data);
             if(cmd_pos==0&&_data!=CMD_HEAD)                   //指令第一个字节必须为帧头，否则跳
             {
                 continue;
             }
            if(cmd_pos<buf_len)                                //防止缓冲区溢出
            buffer[cmd_pos++] = _data;
            cmd_state = ((cmd_state<<8)|_data);                //拼接最后 4 个字节，组成 32 位的整数
            //最后 4 个字节与帧尾比较，得到完整的帧
             if(cmd_state==CMD_TAIL)
             {
                 cmd_size = cmd_pos;                           //指令的字节长度
                 cmd_state = 0;                                //重新检测帧尾
                 cmd_pos = 0;                                  //复位指令指针
    #if(CRC16_ENABLE)
                //去掉指令头尾 EE，尾 FFFCFFFF 共计 5 个字节，只计算数据部分 CRC
                 if(!CheckCRC16(buffer+1,cmd_size-5))          //CRC 校验
                        return 0;
                 cmd_size -= 2;//去掉 CRC16（2 字节）
    #endif
                 return cmd_size;
            }
        }
        return 0;                                              //没有形成一条完整的帧
     }
```

当查找出一条完整的指令帧，会返还 指令的大小，然后调用函数 ProcessMessage 解析指令：

```
size = queue_find_cmd(cmd_buffer,CMD_MAX_SIZE);
// size 大于零证明是完整的指令
```

```
if(size>0)
{
    ProcessMessage((PCTRL_MSG)cmd_buffer,size);          // 解析指令
}
```

2.2.4 ProcessMessage 解析指令类型

```
/*!
*  \brief   消息处理流程
*  \param    msg 待处理的指令
*  \param    size 指令长度
*  宏定义或者 msg 指令类型结构体请查看 main 函数的头文件#include "cmd_process.h"
*  这个函数是将指令中的数据分类获取，然后通过 switch 选择对应的控件类型调用函数并将指令的数据作为参数传入调用的函数。
*/
void ProcessMessage( PCTRL_MSG msg, uint16 size )
{
    uint8cmd_type = msg->cmd_type;                          //指令中的指令类型
    uint8 ctrl_msg = msg->ctrl_msg;                         //指令中的消息的类型
    uint8 control_type = msg->control_type;                 //指令中的控件类型
    uint16 screen_id = PTR2U16(&msg->screen_id);            //指令中的画面 ID
    uint16 control_id = PTR2U16(&msg->control_id);          //指令中的控件 ID
    uint32 value = PTR2U32(msg->param);                     //指令中的数值
    switch(cmd_type)
    {
      case NOTIFY_TOUCH_PRESS:                              //触摸屏按下
      case NOTIFY_TOUCH_RELEASE:                            //触摸屏松开
          NotifyTouchXY(cmd_buffer[1],PTR2U16(cmd_buffer+2),PTR2U16(cmd_buffer+4));
          break;
      case NOTIFY_WRITE_FLASH_OK:                           //写 FLASH 成功
          NotifyWriteFlash(1);
          break;
      case NOTIFY_WRITE_FLASH_FAILD:                        //写 FLASH 失败
          NotifyWriteFlash(0);
          break;
      case NOTIFY_READ_FLASH_OK:                            //读取 FLASH 成功
          NotifyReadFlash(1,cmd_buffer+2,size-6);           //去除帧头帧尾
          break;
      case NOTIFY_READ_FLASH_FAILD:                         //读取 FLASH 失败
          NotifyReadFlash(0,0,0);
          break;
      case NOTIFY_READ_RTC:                                 //读取 RTC 时间
          NotifyReadRTC( cmd_buffer[2], cmd_buffer[3], cmd_buffer[4] ,cmd_buffer[5], cmd_buffer[6]
                         cmd_buffer[7], cmd_buffer[8] );
          break;
      case NOTIFY_CONTROL:
```

```
        {
            if(ctrl_msg==MSG_GET_CURRENT_SCREEN)        //画面 ID 变化通知
            {
                NotifyScreen(screen_id);                //画面切换调动的函数
            }
            else
            {
                switch(control_type)
                {
                    case kCtrlButton:                   //按钮控件
                        NotifyButton(screen_id,control_id,msg->param[1]);
                        break;
                    case kCtrlText:                     //文本控件
                        NotifyText(screen_id,control_id,msg->param);
                        break;
                    case kCtrlProgress:                 //进度条控件
                        NotifyProgress(screen_id,control_id,value);
                        break;
                    case kCtrlSlider:                   //滑动条控件
                        NotifySlider(screen_id,control_id,value);
                        break;
                    case kCtrlMeter:                    //仪表控件
                        NotifyMeter(screen_id,control_id,value);
                        break;
                    case kCtrlMenu:                     //菜单控件
                        NotifyMenu(screen_id,control_id,msg->param[0],msg->param[1]);
                        break;
                    case kCtrlSelector:                 //选择控件
                        NotifySelector(screen_id,control_id,msg->param[0]);
                        break;
                    case kCtrlRTC:                      //倒计时控件
                        NotifyTimer(screen_id,control_id);
                        break;
                        default:
                        break;
                }
            }
            break;
        }
        default:
            break;
    }
}
```

2.2.5 获取画面 ID 和界面数据更新

MCU 的中断函数接收到“切换画面”指令后，将指令存储到指令缓冲区；在 main 函数的 while 循环中，queue_find_cmd 会检测到指令缓冲区收到指令，并进行提取；然后，调用 ProcessMessage 函数解析指令类型并调用对应指令类型的函数和将指令的数据作为函数参数；因收到的指令功能为“画面切换”，将调用处理“画面切换”消息的函数 NotifyScreen，在函数中获取当前画面的 ID。

在 main 函数的 while 循环中的 UpdateUI 函数，用于定时循环更新串口屏数据。

```
//主函数的 while 循环
while(1)
{
    size = queue_find_cmd(cmd_buffer,CMD_MAX_SIZE);          //从缓冲区中获取一条指令
     if(size>0)
     {
        ProcessMessage((PCTRL_MSG)cmd_buffer, size);          //指令处理
     }
    //定时 20 毫秒更新一次数据
    //进一次中断 10ms，timer_tick_count 值+1,100*timer_tick_count = 1s
    if(timer_tick_count%2==0)
    {
        UpdateUI();
    }
}
void NotifyScreen(screen_id)
{
    current_screen_id = screen_id;       //在工程配置中开启画面切换通知，记录当前画面 ID
  …….
}
void UpdateUI( )                                    //定时更新数据
{
    ……
    //画面 6 定时 20ms 刷新一次数据
    if(current_screen_id==6)
    {
        Set_picMeterValue(6,2,test_value);       //设置当前画面的图片指针控件的旋转角度
        test_value +=1;
        if(test_value>=260)                      //指针从 0 到 260 度，指针旋转共 260 度
        {
            test_value = 0;
        }
     }
    ……
}
```

2.3 MCU 例程功能实现

2.3.1 串口屏出厂例程的主界面

图 2-2 主界面一

图 2-3 主界面二

2.3.2 设置按钮按下

图 2-4 按钮界面

点击图 2-2 主界面一的按钮，切换到按钮界面，如图 2-4 按钮界面所示，同时串口屏往 MCU 输送两条指令，一条是点击按钮的指令，一条是画面切换的指令。MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen，在函数中获取当前画面的 ID。并调用驱动函数 SetButtonValue 设置按钮控件的状态值：

```
void NotifyScreen(screen_id)
{
    current_screen_id = screen_id;        //在工程配置中开启画面切换通知，记录当前画面 ID
    if(screen_id == 3)                    //进到画面 3 后按下一个按钮
    {
        //设置当前画面控件 1 的按钮值，按钮值为 1。按钮值 0 为弹起，1 为按下
        SetButtonValue(3,1,1);
    }
    ……
}
```

2.3.3 定时更新文本数据

图 2-5 文本界面

点击图 2-2 主界面一的文本后，进入文本界面，如图 2-5 文本界面所示，并向单片机

发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen，在函数中获取当前画面的 ID 后。在 while 循环中定时 20ms 调用更新画面函数 UpdateUI，然后在 UpdateUI 中调用驱动函数 SetTextInt32，SetTextValue，SetControlBackColor 更新文本界面的数据：

```
void UpdateUI()
{
    .......
  //文本设置和显示定时 20ms 刷新一次 。
    if(current_screen_id==4)
    {
        //当前电流、温度从 0 到 1000 循环显示，艺术字从 0-999 循环显示
        SetTextInt32(4,6,test_value%1000,1,1);          //发送数据到串口屏修改当前控件 6 的值
        SetTextInt32(4,7,test_value%1000,1,1);          //发送数据到串口屏修改当前控件 7 的值
        SetTextValue(4,1,"机房 10");                    //发送数据到串口屏修改当前控件 1 的值
        test_value++;
        if(test_value>= 1000)
        {
            test_value = 0;
        }
        if(test_value>0&&test_value<500)                //大于 0 小于 500 文本显示红色
        {
            SetControlBackColor(4,6,0xF800);            //发送数据到串口屏修改控件 6 的背景色
        }
        else if(test_value>=500)                        //大于 500 文本显蓝色
        {
            SetControlBackColor(4,6,0x00if);            //发送数据到串口屏修改控件 6 的背景色
        }
    }
    .......
}
```

2.3.4 定时更新仪表数据

图 2-6 仪表界面

点击图 2-2 主界面一中仪表后，进入仪表界面，如图 2-6 仪表界面所示，并向单片机发送点击按钮控件的指令和画面切换的指令。MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen，在函数中获取当前画面的 ID。在 while 循环中定时 20ms 调用更新画面函数 UpdateUI，然后在 UpdateUI 中调用驱动函数 Set_picMeterValue 更新仪表的的旋转角度：

```
void UpdateUI()
{
    ……
    //仪表界面和定时 20ms 刷新一次旋转角度
    if(current_screen_id==6)
    {
        Set_picMeterValue(6,2,test_value);      //设置当前画面的图片指针控件的旋转角度
        test_value +=1;
        if(test_value>=260)                      //仪表值 0~140，指针旋转共 260 度
        {
            test_value = 0;
        }
     }
    ……
}
```

2.3.5 进度条值与文本关联

图 2-7 滑块界面

点击图 2-2 主界面一中的滑块后，进入滑块界面，如图 2-7 滑块界面所示，当拖动滑块的时候，滑动条的值发生改变，同时向 MCU 发送滑动条改变值的指令。MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析指令后，调用 ProcessMessage 函数解析指令并调用对应指令类型的函数和将指令所带的数据作为函数参数；此处的指令功能为“改变滑动条值”，所以调用函数 NotifySlider 处理：

```
/*!
*  \details  当滑动条改变(或调用GetControlValue)时，执行此函数
*  \paramscreen_id画面ID
*  \paramcontrol_id控件ID
*  \param value  值
*/
void NotifySlider(uint16 screen_id, uint16 control_id, uint32 value)
{
    if(screen_id==7&&control_id==5)              //滑块控制
    {
        if(value<100||value>0)
        {
            SetProgressValue(7,4,value);         //更新当前画面进度条控件 4 的数值
            SetTextValueInt32(7,6,value);        //更新当前画面文本控件 6 的数值
        }
    }
}
```

2.3.6 获取时间和时间倒计时

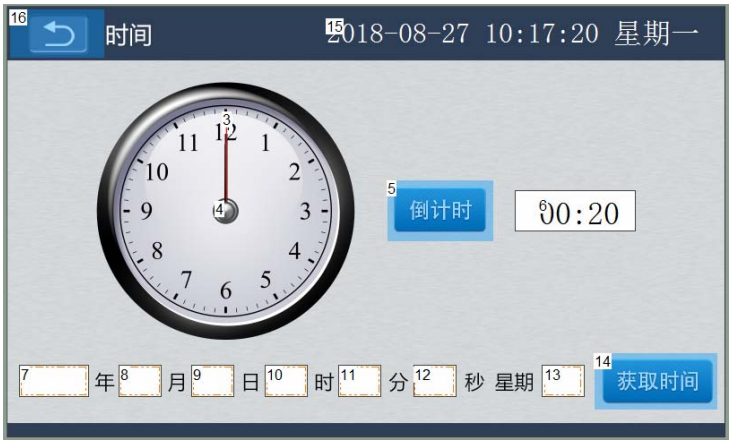

图 2-8 时间界面

点击图 2-2 主界面一的时间后，进入时间界面，如图 2-8 时间界面所示，并向单片机发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU解析切换画面的指令后，调用处理“画面切换”消息的函数NotifyScreen，在函数中获取当前画面的ID。然后在while循环中定时调用函数UpdateUI更新当前画面ID的数据，在函数UpdateUI中定时每 500 毫秒调用驱动函数 ReadRTC向串口屏发送读取屏内RTC时钟的指令，串口屏接收到读取RTC指令后往MCU串口发送当时间的BCD码。此处的指令为“RTC时间”，所以调用处理RTC的函数NotifyReadRTC，按下获取时间的按钮，将MCU获取到的RTC时间显示出来：

```
//在 UpdateUI 函数里添加下面代码
//进入时间画面，定时每 500 毫秒发送一次读取RTC时间的指令，然后每 10s响蜂鸣器一次
void UpdateUI()
{
    …….
    if(current_screen_id == 8 &&timer_tick_count %50 ==0)
    {
        ReadRTC();                                  //发送获取 RTC 时间指令
        if(sec%10==0)                               //当 10s, 20s , …… 60s 时蜂鸣器响一声
        {
            SetBuzzer(250);                         //启动蜂鸣器
        }
    }
   …….
}
void NotifyButton(uint16 screen_id, uint16 control_id, uint8  state)
{
     ……
    if(screen_id == 8)
    {
        if(control_id==14 )
        {
```

```
                SetTextInt32(8,7,years,1,1);                    //显示年
                SetTextInt32(8,8,months,1,1);                   //显示月
                SetTextInt32(8,9,days,1,1);                     //显示日
                SetTextInt32(8,10,hours,1,1);                   //显示时
                SetTextInt32(8,11,minutes,                      //显示分
                SetTextInt32(8,12,sec,1,1);                     //显示秒
                if(weeks == 1)
                {
                    SetTextValue(8,13,"一");                    //显示星期一
                }
            }
        }
        ……
//串口屏发送到MCU串口的的RTC时间为BCD码
void NotifyReadRTC(uint8 year,  uint8 month,  uint8 week,  uint8 day, uint8 hour,  uint8 minute, uint8
                   second)
{
    sec=(0xff& (second>>4))*10 +(0xf & second);          // BCD码转十进制，转换秒的值
    years   =(0xff & (year>>4))*10 +(0xf & year);
    months  =(0xff & (month>>4))*10 +(0xf & month);
    weeks   =(0xff & (week>>4))*10 +(0xf & week);
    days    =(0xff & (day>>4))*10 +(0xf & day);
    hours   =(0xff & (hour>>4))*10 +(0xf & hour);
    minutes =(0xff & (minute>>4))*10 +(0xf & minute);
}
```

2.3.7 播放动画和播放音乐

图 2-9 动画界面

图 2-10 音频界面

点击图 2-2 主界面一中的动画或者音乐后，进入动画图 2-9 动画界面界面后自动播放动画或音乐图 2-10 音频界面界面后自动播放音乐。MCU 接收到切换画面的指令后会进行解析，具体请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen，并在函数中调用播放动画的驱动函数 AnimationStart 或者调用播放音乐的驱动函数 PlayMusic：

```
void NotifyScreen(screen_id)
{
    current_screen_id = screen_id;                  //（在工程配置中开启画面切换通知）记录当前画面 ID
    //进到画面自动播放 GIF
    if(current_screen_id == 9)
    {
        AnimationStart(9,1);                        //动画开始播放
    }
    //进入音乐画面自动播放
    if(current_screen_id == 17)
    {
        //十六进制表示的音频的路径及音频名字，可以用软件 visual TFT 的指令助手转换
        uint8 buffer[6] = {0x90,0x01 ,0x00 ,0x01 ,0x01};;
        SetButtonValue(17,3,1);                     //设置音频画面的按钮控件 3 的值
        PlayMusic(buffer);                          //播放音乐
    }
}
```

2.3.8 显示图标

图 2-11 图标界面

点击图 2-2 主界面一中的图标后，进入图标界面，如图 2-11 图标界面所示，并向单片机发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen 并在函数中获取当前画面 ID。在 while 循环中定时调用更新画面函数 UpdateUI，然后在 UpdateUI 中再调用驱动函数 SetButtonValue 设置按钮 2,3,4,5 的状态，和调用驱动函数 AnimationPlayFrame 设置播放指定图标的帧，以达到循环显示图标的效果：

```
void UpdateUI()
{
    .......
     //当前画面为图标界面时，图标 40ms 轮流显示
    if(current_screen_id == 10)
    {
        if(timer_tick_count %40 == 0 && icon_flag == 0)
          {
              SetButtonValue(9,5,0);                    //设置按钮 5 的状态值
              SetButtonValue(9,2,1);                    //设置按钮 1 的状态值
              AnimationPlayFrame(9,1,0);                //显示图标控件的指定帧
              icon_flag = 1 ;                           //标志位
          }
          else if(timer_tick_count %40 == 0 && icon_flag == 1)
          {
              SetButtonValue(9,2,0);                    //设置按钮 2 的状态值
              SetButtonValue(9,3,1);                    //设置按钮 3 的状态值
              AnimationPlayFrame(9,1,1);                //显示图标控件的指定帧
              icon_flag = 2;
          }
          else if(timer_tick_count %40 == 0 && icon_flag == 2)
          {
```

```
                    SetButtonValue(9,3,0);
                    SetButtonValue(9,4,1);
                    AnimationPlayFrame(9,1,2);
                    icon_flag = 3 ;
                }
                else if(timer_tick_count %40 == 0 && icon_flag == 3)
                 {
                    SetButtonValue(9,4,0);
                    SetButtonValue(9,5,1);
                    AnimationPlayFrame(9,1,3);
                    icon_flag    = 0 ;
                 }
        }
        ……
}
```

2.3.9 定时更新曲线数据

图 2-12 正弦波

图 2-13 锯齿波

点击图 2-3 主界面二中的曲线后，进入曲线界面，并向单片机发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen 并在函数中获取当前画面 ID。按下界面中正弦波的按钮，MCU 接收到按钮按下的指令，就调用按钮函数 NotifyButton 处理并更改曲线类型为正弦波，并在定时周期为 20 毫秒一次更新画面函数 UpdateUI 中，调用驱动函数 GraphChannelDataAdd 更新正弦波曲线的值，如果按下锯齿波按钮，则改为更新锯齿波数值：

```
void NotifyButton(uint16 screen_id, uint16 control_id, uint8  state)
{
    if(screen_id == 11)
    {
        if(control_id==2)                                   //正弦波控件
        {
            curves_type = 0;                                //曲线类型  正弦波
        }
        else if(control_id==3)                              //锯齿波控件
        {
            curves_type = 1;                                //曲线类型  正弦波
        }
    }
}
void UpdateUI()
{
    //实时曲线，正弦波数组。定时 50ms 刷新一次数据
    if(current_screen_id == 11 &&timer_tick_count %5 == 0)
    {
        if(curves _type== 0)
        {
```

```
            //正弦数组
            uint8sin[256]={};                                    // 数组详细请参考具体代码
            //添加数据到当前画面的曲线控件中一次一个数据
            GraphChannelDataAdd(11,1,0,&sin(num),1);
            num++;
            if(num>= 255)
            {
                num =0;
            }
        }
        else if(curves_type == 1)
        {
            //锯齿波数组
            uint8 sawtooth[180] = {}                             // 数组详细请参考具体代码
            / /添加数据到当前画面的曲线控件中一次一个数据
            GraphChannelDataAdd(11,1,0, & sawtooth (num),1);
            num++;
            if(num>= 180)
            {
                num =0;
            }
        }
    }
}
```

2.3.10 选择控件

图 2-14 选择控件

点击图 2-3 主界面二选择控件后，进入选择控件界面，如图 2-14 选择控件所示。当选中某个时间后，点击存储按钮 MUC 会返还对应的时间段：

如选中 10 点，串口屏会向 MUC 发送指令（13 号画面，选择控件 1，选择 10 点），MCU

提取到指令后，调用 ProcessMessage 函数解析指令并调用对应指令类型的函数和将指令的数据作为函数参数；此处，因收到的指令为“选择控件改变的值”，所以调用处理选择控件的函数 NotifySelector 获取选择控件的值，按钮按下之后对获取到的值进行时间段判断并显示出选择的时间段：

```
/*brief   选择控件通知
*  \details   当选择控件变化时，执行此函数
*  \paramscreen_id 画面 ID
*  \paramcontrol_id 控件 ID
*  \param item  当前选项
*/
void NotifySelector(uint16 screen_id, uint16 control_id, uint8    item)
{
    if(screen_id == 13&&control_id == 1)
    {
        Select_H =    item;                          //获取选择控件 1 的值
    }
    if(screen_id == 13&&control_id == 2)
    {
        Select_M =    item;                          //获取选择控件 2 的值
    }
}
void NotifyButton(uint16 screen_id, uint16 control_id, uint8    state)
{
    ……
    if(screen_id == 13 && control_id==4)
    {
      if(Select_H>=0&&Select_H<=6)              //0 到 6 小时的时间段为凌晨
      {
        SetSelectorValue(13,3,0);
      }
      else if(Select_H>=7&&Select_H<=12)        //7 到 12 小时的时间段为凌晨
      {
        SetSelectorValue(13,3,1);
      }
      else if(Select_H>=13&&Select_H<=18)       //13 到 18 小时的时间段为凌晨
      {
        SetSelectorValue(13,3,2);
      }

      else if(Select_H>18&&Select_H<=23)        //18 到 23 小时的时间段为凌晨
      {
        SetSelectorValue(13,3,3);
      }
    }
```

```
}
```

2.3.11 触发警告和警告解除

图 2-15 数据记录

点击图 2-3 主界面二中的数据记录后，进入数据记录图 2-15 数据记录界面后，并向单片机发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析指令后，调用处理“画面切换”消息的函数 NotifyScreen，并在函数中调用驱动函数 Record_SetEvent 设置触发警告，延时两秒后调用驱动函数 Record_ResetEvent 发出解除警告：

```
void NotifyScreen(screen_id)
{
    current_screen_id = screen_id;          //记录当前画面 ID，current_screen_id 为全局变量
    ……
    if(current_screen_id == 15)             //数据记录显示
    {
        //Record_SetEvent(15,1,0,0);第三个参数为数据记录的警告值,第四个参数为解除警告时间，为 0
        //时使用串口屏时间
        Record_SetEvent(15,1,0,0);
        Record_SetEvent(15,1,1,0);
        Record_SetEvent(15,1,2,0);          //设置数据记录控件,设置警告值
        Record_SetEvent(15,1,3,0);
        Record_SetEvent(15,1,4,0);
        Record_SetEvent(15,1,5,0);
        Record_SetEvent(15,1,6,0);
        Record_SetEvent(15,1,7,0);
        delay_ms(2000);                     //延时两秒
        Record_ResetEvent(15,1,0,0);
        Record_ResetEvent(15,1,1,0);        //设置数据记录控件,设置解除警告
```

```
            Record_ResetEvent(15,1,2,0);
        }
    ……
}
```

2.3.12 历史曲线

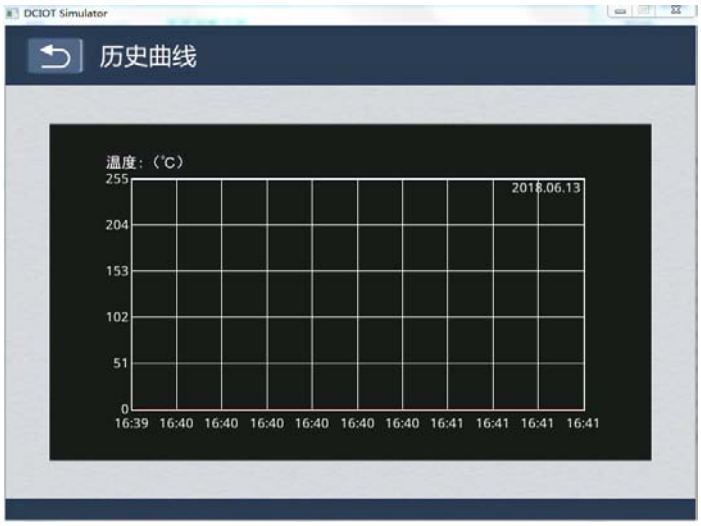

图 2-16 历史曲线界面

点击图 2-3 主界面二中的历史曲线按钮控件后，进入历史曲线界面，如图 2-16 历史曲线界面所示，并向单片机发送点击按钮控件的指令和画面切换的指令，MCU 接收到指令后会进行解析，具体的过程请查看目录的 2.2.1 指令解析过程。

MCU 解析切换画面的指令后，调用处理“画面切换”消息的函数 NotifyScreen 并在函数中获取当前画面 ID。在 while 循环中定时调用更新画面函数 UpdateUI，然后在 UpdateUI 中定时 1 秒钟调用一次驱动函数 HistoryGraph_SetValueInt8 更新历史曲线的值：

```
void UpdateUI()
{
    …….
    //历时曲线，正弦波数组
    if(current_screen_id == 16)                          //历史曲线控件采样周期 1s 一个点。
    {
        if(curves == 0)
        {
            uint8i,Sendsin[1]={0};                       //初始化或者清 0
            HistoryGraph_SetValueInt8(16,1, &sin[num],1); //添加历史曲线数据
            num++;
            if(num>= 255)
            {
                num =0;
            }
        }
        if(curves == 1)
        {
```

```
                uint8sawtooth[180] = {};                    //锯齿波数组详细请参考程序
                //添加历史曲线数据一次添加一个
                HistoryGraph_SetValueInt8(16,1,& sawtooth[num],1);
                num++;
                if(num>= 180)
                {
                    num =0;
                }
            }
        }
    …….
}
```

3. 程序配置

3.1 Visual TFT 配置

VisualTFT 中 CRC16 相应设置，如下图 3-1 所示：（注：可以不开启）

图 3-1 TFT 配置

VisualTFT 中工程的波特率设置要单片机初始化的一致。

3.2 KEIL 开发软件

Keil C51，兼容单片机 C 语言软件开发系统。Keil C51 的 STC 库需要用 STC-ISP 软件添加。（为了避免不必要的 BUG，所以尽量选对型号）。

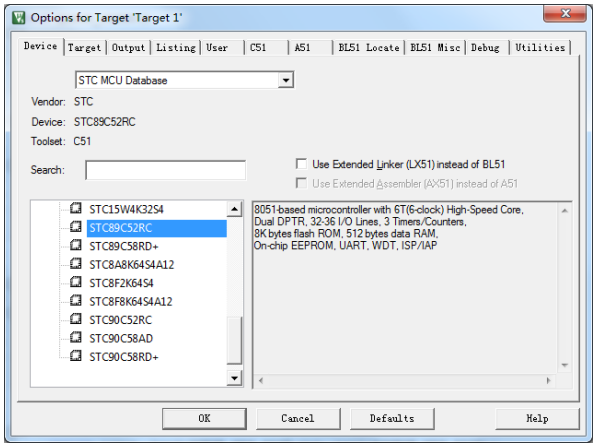

图 3-2 C51

Keil MDK，MDK-ARM 软件为基于 Cortex-M、Cortex-R4、ARM7、ARM9 处理器设备提供了一个完整的开发环境。配置如下图 3-3 STM32（选择对应的型号）

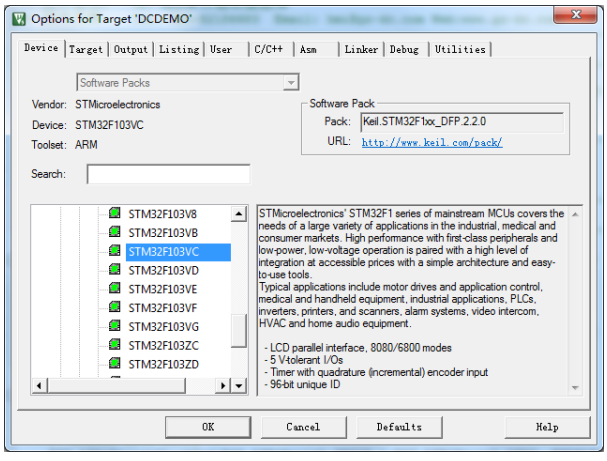

图 3-3 STM32

4. 如何移植到其它 MCU 系列

我司例程所用的单片机是 STM32F103VCT6 和 STC89CX 系列，如果客户需要把工程移植的其他系列上，建议复制所有源文件，然后根据需要修改下面几个地方：

1. 单片机需要修改为自身对应型号的头文件 例如：
   STC89CX 系列的头文件------ #include<reg52.h>
   STC15F2K60S2 的头文件---- #include<STC15F2K60S2.h>
2. hmi_user_uart.c - 串口初始化及数据发送(根据用户自己 MCU 数据设定)

```
void UartInit();                //初始化通信串口，设置波特率等
void SendChar(char ch);     //通过串口发送一个字节
```

3. 串口数据接收中断处理 （这里只是范例，主要看用户的 MCU）

```
//8051 平台下串口数据接收处理如下所示
//接收的串口数据通过 queue_push 添加的指令队列末尾
void serial() interrupt 4
{
    if(RI)                      //接收到窗口数据
    {
        RI= 0;
        queue_push(SBUF);//压入到指令缓冲区
     }
}
```

4. STM32 平台下串口数据接收处理如下所示：

```
void USART1_IRQHandler(void)
{
     if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
     {
         uint8_t data = USART_ReceiveData(USART1);
         queue_push(data);
     }
}
```