Appplication Note

AN00000002

如何通过 Visual TFT 建立一个工程、下载、仿真并和用户处理器协同调试、工作

V1.00 Date: 2013/10/23

产品应用笔记

文件信息

类别	内容
关键词	7.0DEMO 例程配套文档
摘要	基于 ST Cortex M3 和 51 单片机芯片来讲解如何和大彩工业串口屏联合仿真测试和工作

修订历史

版本	日期	原因
V1.00	2013/10/23	创建文档。
V1.01	2013/11/28	去掉 VISUAL TFT 工程创建说明，更新驱动部分说明
V1.02	2013/12/16	51 单片机和 STM32 合并成一篇文档

目 录

1. 准备工作

Visual TFT 工程的建立请参考开发包中相关的视频和文档，这里只介绍如何用 ST 公司 CrotexM3 芯片 STM32F103VCT6 和 ST 公司的 51 单片机 STC89C52RC 来驱动大彩工业串口屏。开发包的下载请到广州大彩光电科技有限公司的官方网站地址：www.gz-dc.com 上的下载资料一栏下载。

2. 打开用户微处理器的工程

开发包提供了基于 51 和 STM32 的工程，用 keil4 打开 STM32 开发包中的工程 DCDEMO7.uvproj 或者 51 开发包中的工程 51demo.uv2 来分析一下整个程序的构架，整个工程都是用通用的 C 语言描述，方便用户移植到不同的平台上来实现。请注意工程的波特率都是配置成 115200。但是 51 单片机要达到这个波特率，必须安装 22.1184MHz 的外部晶振，如果只安装了 11.0592MHz 晶振，最高波特率只能达到 57600，所以请修改 Visual TFT 工程中波特率为 57600 使之波特率相匹配这样才能通信上。

2.1 驱动文件的构架分析

驱动部分的架构如图 2.1 所示，最下层为用户处理器串口的硬件驱动，往上一层为为串口屏的命令帧驱动，再上面一层是串口消息响应处理函数，最上面为用户应用代码。

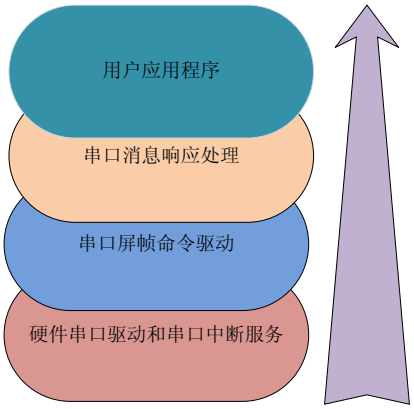

图 2.1 驱动架构

驱动文件的架构如图 2.2 所示。跟用户芯片有关的文件是 hmi_user_uart.c 文件，不同的芯片只需要修改此文件的串口驱动函数。其他的文件不需要做修改。cmd_queue.c 文件描述了一个 FIFO 结构（先进先出队列缓冲区），接收到串口屏返回的数据会先存入到这个 FIFO 缓冲区，然后等待主函数处理。如果芯片资源允许，这个缓冲区的大小尽可能大一些，防止溢出后串口数据丢失。在文件 cmd_queue.h 文件里面的参数 QUEUE_MAX_SIZE 描述了这个 FIFO 大小。hmi_driver.c 文件有大彩工业串口屏所有指令的驱动函数，具体的驱动函数声明请查看 hmi_driver.h 文件。下面分小节来具体介绍。

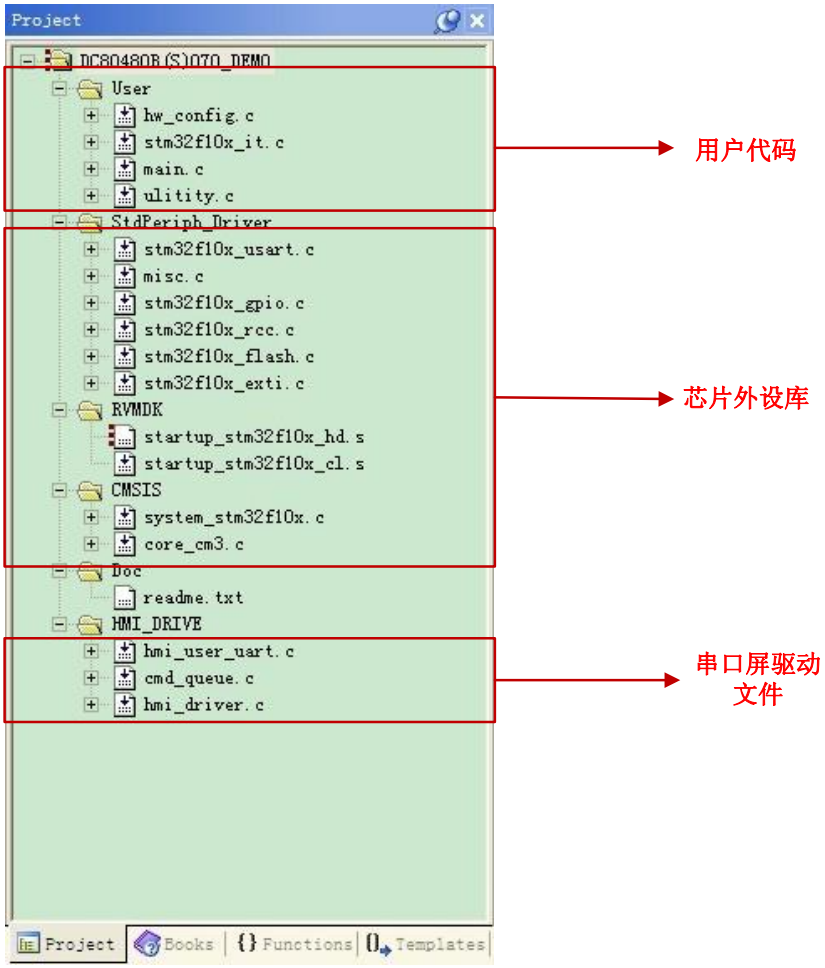

图 2.2 文件框架列表

1.1 硬件串口驱动和中断服务

串口屏相关的驱动文件一共有三个。

1. 文件 hmi_user_uart.c 中，描述了与硬件有关的串口的初始化函数

包含三个函数分别是

UartInit(uint32 BaudRate)； 串口初始化函数

SendChar(uchar t); 通过串口发送一个字节驱动函数

SendStrings(uchar *str); 通过串口发送字符串驱动函数

2. 文件 cmd_queue.c 中，描述了串口接收数据放入先入先出数据缓冲区（FIFO）中的描述文件
3. 文件 hmi_driver.c 封装了大彩工业串口指令集中所有指令的驱动函数，所有串口指令驱动的函数都在 hmi_driver.h 文件中定义。

硬件串口的接收中断服务函数在文件 stm32f10x_it.c 中，名称为 void USART1_IRQHandler(void)，在串口接收中断函数中收到一个字节的数据后，会自动存入到 FIFO 中。

以上这些函数描述了与硬件相关的驱动。用户要移植到不同的平台上，只需要修改这些函数即可。

2.2 串口帧命令驱动

所有串口指令驱动的函数都在 hmi_driver.h 文件中定义，用户可以在这个文件里面查找，这个函数的具体实现在 hmi_driver.c 文件中描述，如程序清单 2.2 中所示。

程序清单 2.1

```
void GraphChannelDel(uint16 screen_id,uint16 control_id,uint8 channel)
CONTROL_BODY(0x31,TX_8(channel))
```

串口的所有发送动作都封装在另外结构体中，如程序清单 2.2 所示。

程序清单 2.2

```
//COMMAND_BODY-封装基本命令
#define COMMAND_BODY(CMD,BODY) \
    {SendBeginCMD();TX_8(CMD) BODY SendEndCmd();}

//CONTROL_BODY-封装控件更新指令
#define CONTROL_BODY(SUB_CMD,BODY) \
    {SendBeginCMD();TX_8(0xB1)TX_8(SUB_CMD)TX_16X2(screen_id,control_id) BODY
SendEndCmd();}
```

下面的两个函数是等效的

程序清单 2.3

```
void setHandShake_1()
{
SendBeginCMD(); //发送帧头
SendChar(0x04);   //发送命令 0x04
SendEndCmd();   // 发送帧尾
}
```

```
void SetHandShake()
COMMAND_BODY(0x04,)
```

如果用户要添加新的驱动函数可以按照左边的格式添加。

2.3 串口消息处理函数分析

2.3.1 主函数框架

我们设计的美工主界面如图 2.3 所示。当用户按下“文本”区域的时候，转到对应的文本显示界面图 2.4 去显示变化的文本，“最高电压”处的文本控件是一个用户键盘，当按下这个区域时，会弹出一个键盘，让用户输入设置的数值，然后用户程序需要获得键盘输入的数字把这个数值除以 2，然后在“最高电压/2”处显示出来。“当前电流”“实时湿度”“实时温度”三个地方会显示不断更新的数值。

图 2.3 主界面

图 2.4 文本控件界面

当按下“图标、动画控件”区域的时候，跳到图 2.5 中执行，当用户按下”启动运行”的时候，图标控件 1 切换到状态运行的状态，吹风机的动画开始播放，当用户按下“停止运行”的时候，图标控件 1 切换到停止的状态，吹风机的动画停止播放，当按下“复位”的时

候，图标控件 1 和动画控件均不显示。

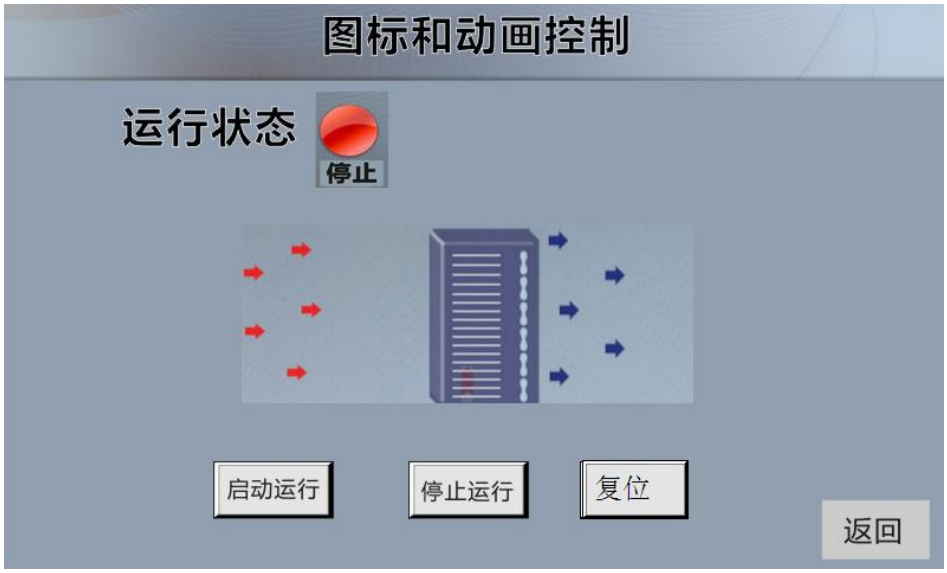

图 2.5 动画控件界面

当按下“曲线”区域的时候，跳到图 2.6 中执行，显示一条红色的变化曲线

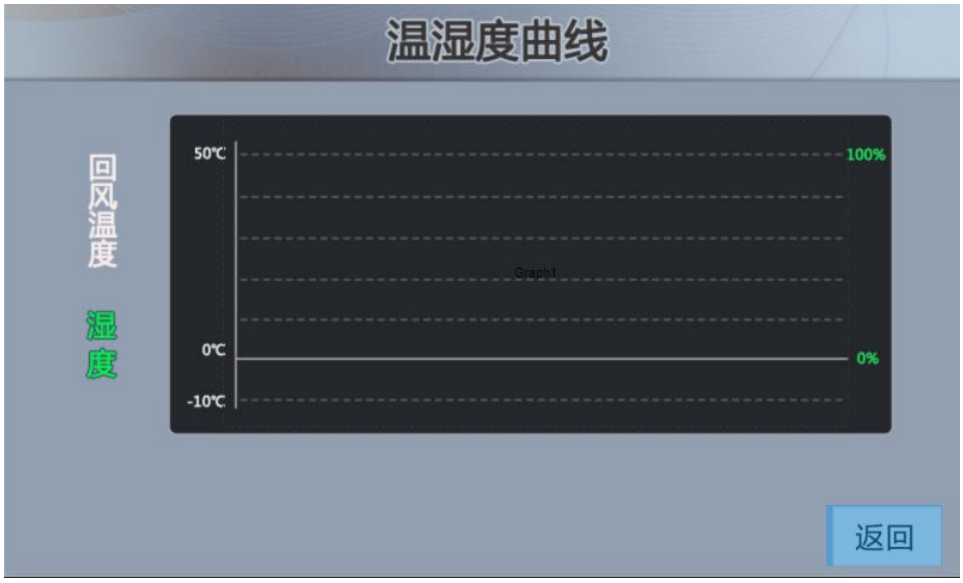

图 2.6 曲线控件界面

当按下“仪表”区域的时候，会跳转到图 2.7 中执行。在仪表页面中两个仪表会从 0 度转动到 9 度，来回循环

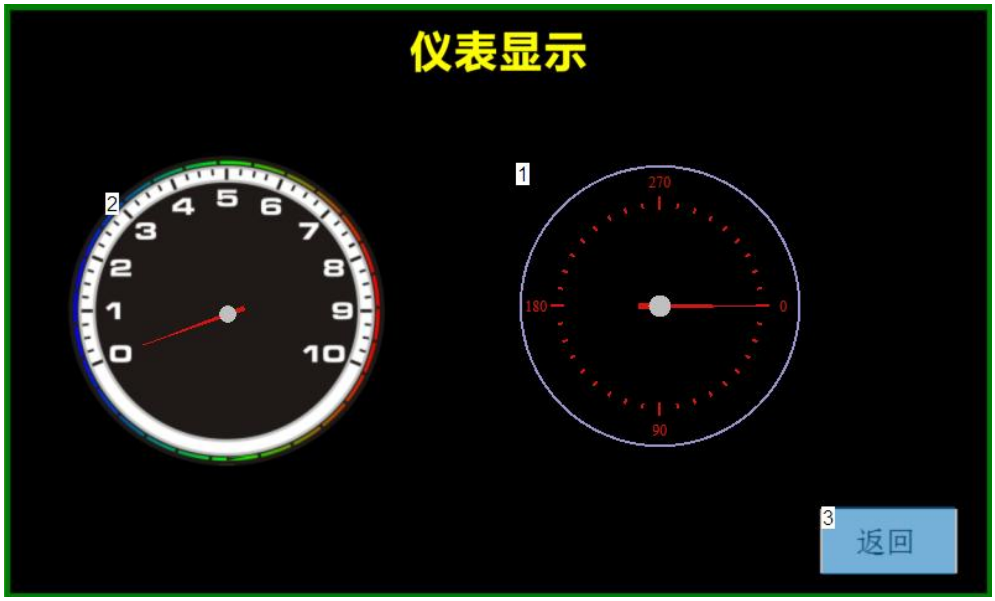

图 2.7 仪表控件界面

当按下“进度条、滑块”区域的时候，会跳转到图 2.8 中显示。进度条的进度会从 10%转动到 90%，来回循环，按动滑块的控件，屏幕的背光会进行调节，左边最亮，右边最暗。

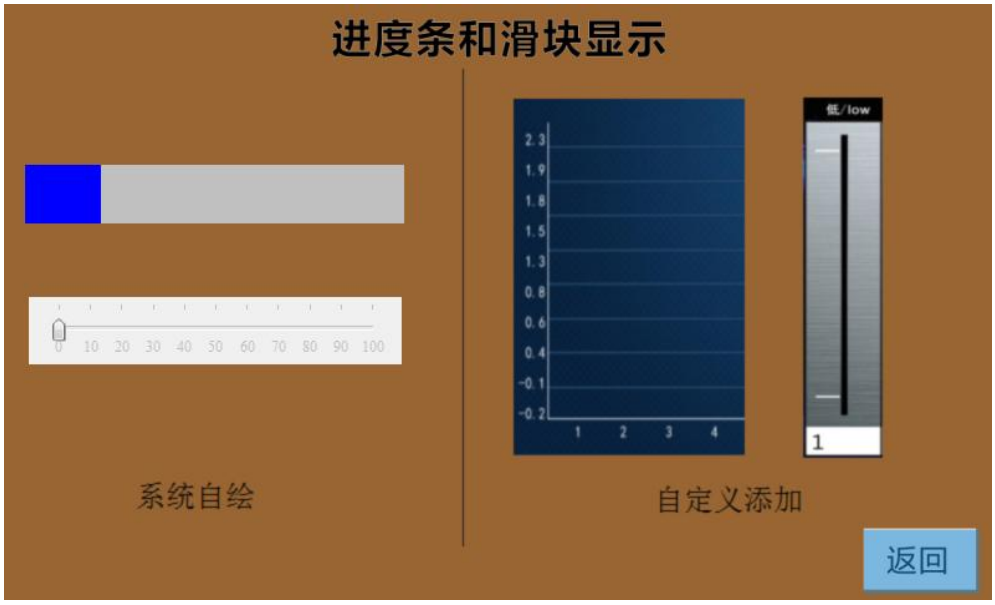

图 2.8 进度条和滑块控件界面

打开 main.c 文件。文件的流程如下图所示。

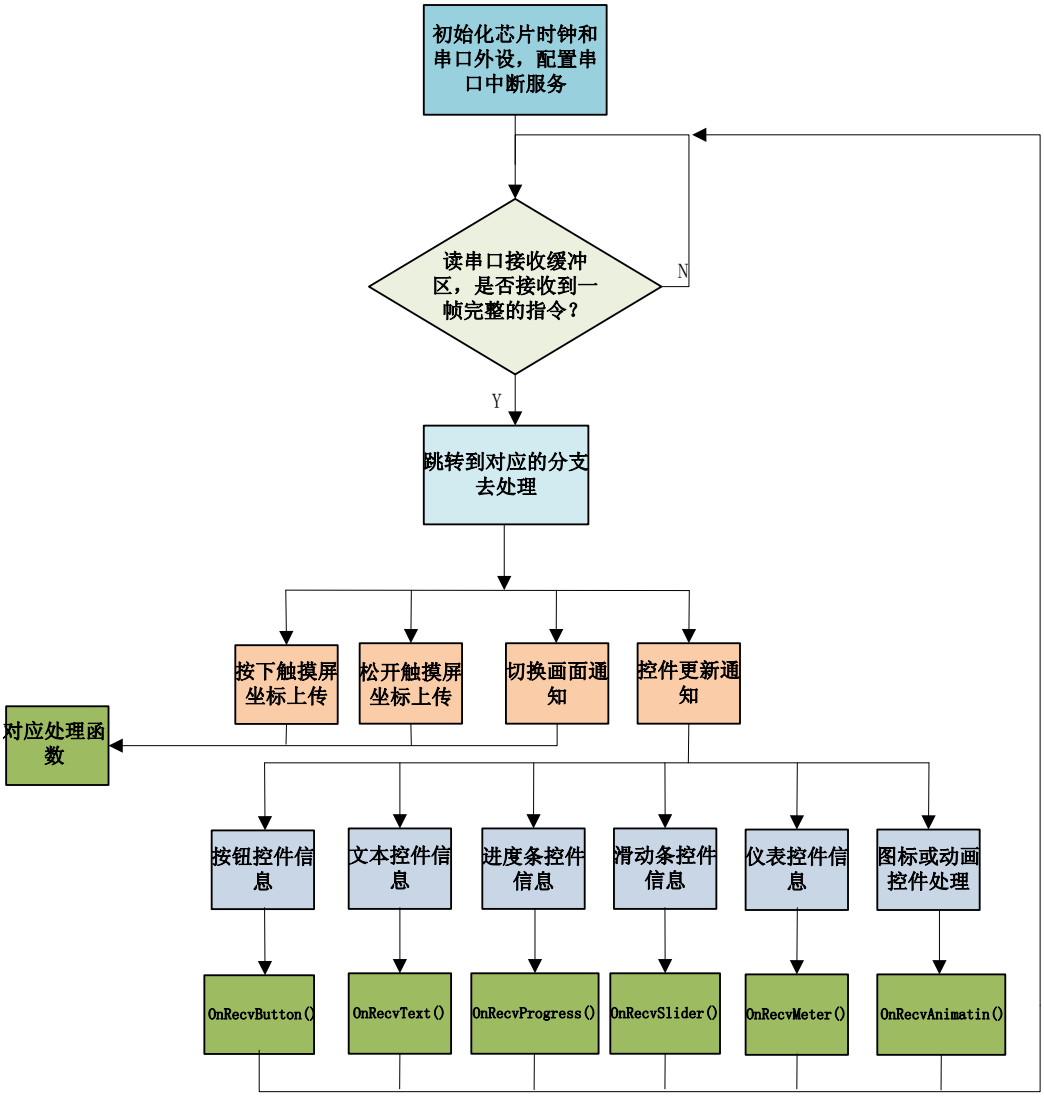

图 2.9 main（）函数流程图

从上面的流程整个程序是一个循环体，主函数 main()在不断读取串口缓冲区 FIFO 的数据，检测是否收到一个完整的命令帧，如果收到了就对这个命令进行解析。然后跳到对应的分支去处理。如程序清单 2.4 所示。

程序清单 2.4

```
 switch(cmd_type)//指令类型
{
case CMD_TOUCH_PRESS_RETURN://按下触摸屏坐标上传
case CMD_TOUCH_RELEASE_RETURN://松开触摸屏坐标上传
      OnRecvTouchXY(cmd_buffer[1],PTR2U16(cmd_buffer+2),PTR2U16(cmd_buffer+4));
      break;

case CMD_TYPE_UPDATE_CONTROL://控件更新消息
     msg_type = msg->ctrl_msg;
      switch(msg_type)//消息类型
      {
                              case kCtrlMsgGetCurrentScreen:
                                    OnRecCurrentScreen(msg,size);//  获取当前画面值
                                    break;
                              case kCtrlMsgGetData:
                                    control_type = msg->control_type;//控件类型
                                    switch(control_type)
                                    {
                                    case  kCtrlMsgGetCurrentScreen:
                                          OnRecCurrentScreen(msg,size);//  更新当前画面值
                                    case kCtrlButton:
                                                OnRecvButton(msg,size);
                                                break; //按钮控件处理
                                    case kCtrlText:
                                                OnRecvText(msg,size);
                                                break; //文本控件处理
                                    case kCtrlProgress:
                                                OnRecvProgress(msg,size);
                                                break; //进度条控件处理
                                    case kCtrlSlider:
                                                OnRecvSlider(msg,size);
                                                break; //滑动条控件  处理
                                    case kCtrlMeter:
                                                OnRecvMeter(msg,size);
                                                break; //仪表控件处理
                                    case kCtrlAnimation:
                                                OnRecvAnimation(msg,size);
                                                break; //图标或者动画控件处理
```

```
                default:
                    break;
                }
            break;
        }
```

2.3.2 消息响应处理函数

消息响应处理函数主要是对串口屏回传的事件消息进行解析和处理，例如当第一个画面的第一个图标按下去的时候，用户需要去读取“最高电压”文本控件的值，然后除以 2，在另外一个文本控件“最高电压/2”里面显示出来。

在按钮消息响应函数 OnRecvButton()和 OnOnRecvText()这两个函数添加部分代码就可以完成这样的功能。

程序清单 2.5

```
OnRecvButton（）部分代码：
……
    if(screen_id == 0 && control_id == 1)        //画面 0 ID 为 1 的按键被按下了
    {
        GetControlValue(1,2);                    //获取最高电压文本控件的值
    }
……
OnOnRecvText()部分代码
…
    //获取系统自带键盘输入
    if(screen_id == 1 &&    control_id == 2)
    {
        while(((uint8 *)(&msg->param))[i])
            voltage = voltage*10+ (((uint8 *)(&msg->param))[i++] - 0x30); //从接收缓冲区取出键盘输入的数字，再转换成十进制数字
        sprintf(buf,"%d 伏特",voltage/2);                    //获取键盘输入的数据除以 2 后再显示出来
        SetTextValue( 1, 5,buf);                             //显示到最高电压/2
    }
…
```

代码编写完毕后编译下载到用户的处理器，下一节将介绍如何仿真和调试。

2.4 和 Visual TFT 虚拟串口屏进行联机仿真

Visual TFT® 提供了一个虚拟串口屏，用户可以在脱离硬件的调试下来调试代码。

有两种方式可以对虚拟串口屏进行联机仿真，一种就是用户真实的处理器和虚拟串口屏通过连接硬件串口来进行仿真，另外一种是用户虚拟的处理器和虚拟串口屏通过虚拟串口进行仿真，这个前提是用户处理器的开发环境支持软件模拟器并把串口映射到 PC 的串口上。

先介绍第一种仿真方式。用户硬件处理器和虚拟串口屏联机仿真，把用户处理器的串口交叉连接到电脑的串口上，打开 Visual TFT 虚拟串口屏（注意电平匹配，RS232 电平匹配 23 电平，TTL 电平匹配 TTL 电平，用户处理的串口 TX 连到 PC 串口的 RX，用户处理器串口的 RX 连到 PC 串口的 TX）。设置好波特率，就可以和用户处理器进行通信了。点击通信记录可以看到它们的通信过程，这个时候虚拟串口屏和硬件就可以进行交互了。如图 2.10 所示。这时候可以查看文本控件内容是否更新正确，按下触摸屏的按钮是否得到对应的响应。

图 2.10 联机测试

第二种仿真方式需要原先装一个虚拟串口，如果用户处理器的开发环境支持模拟器并可以把串口映射出来，推荐用这个方式来仿真和调试。因为纯软件的调试更容易排除早期的错误，从而加快整个项目的进度。需要一个虚拟串口把用户模拟的处理器串口和虚拟串口屏连接起来，这里需要用到一个软件来实现创建一个虚拟串口，安装开发包中的 VSPD 软件后打开这个软件，点击主界面的 Add paris 来添加一对连接的串口，如图 2.11 所示。此时 COM3 已经和 COM4 相连。下面的流程中会把虚拟串口屏连接到 COM4，MDK 的的软件模拟器串口映射到 COM3，这样就可以联机调试了。

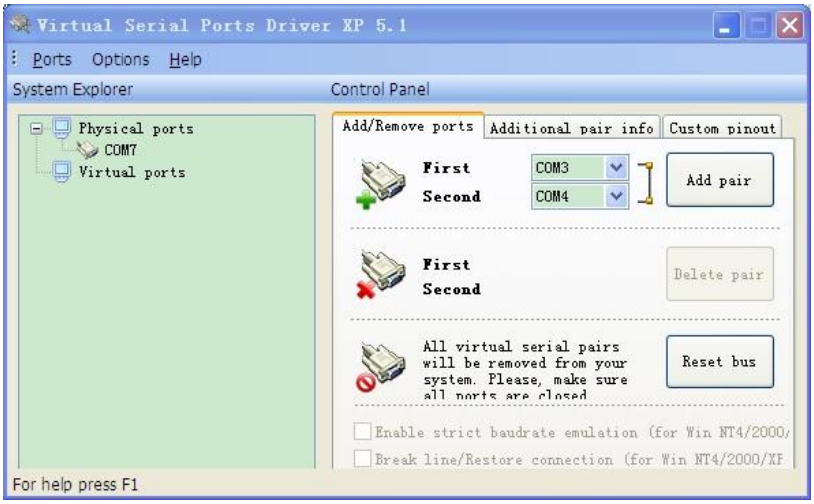

图 2.11 创建一对相互连接的虚拟串口

接下来在 MDK 的开发环境中，把软件模拟器的串口映射到 COM3，我们需要写一个调试初始化文件 COMDEBUG.ini，文件的内容如下所示

```
MODE COM3 115200,0,8,1
ASSIGN COM3 <S1IN> S1OUT
```

第一句命令的功能是把 COM7 的属性设置为 115200 波特率，没有奇偶校验，8 位数据位，1 位停止位。第二句命令的功能是把软件仿真器的串口映射到 COM3。在 MDK 的工作配置窗口设置为启动软件模拟器时自动载入此文件。如图 2.12 所示。

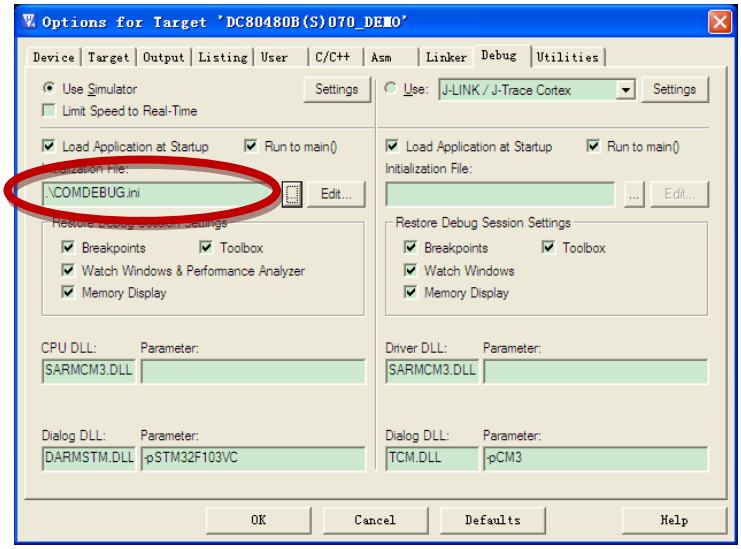

图 2.12 修改工程配置

启动 MDK 仿真器，打开虚拟串口屏，把 COM4 打开（COM3 和 COM4 已经连接起来了），设置为 115200，按下 MDK 全速运行仿真器，可以看到虚拟串口屏也运行起来，通信记录窗口可以看到他们交互的指令。

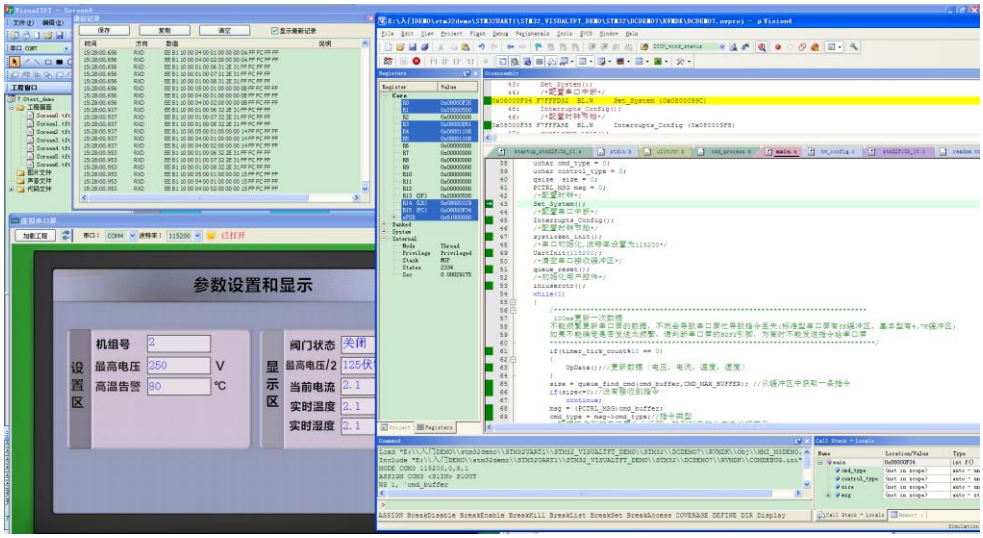

图 2.13 软件联合仿真

2.5 下载到用户处理器进行硬件测试

仿真正确之后，就基本确定用户部分代码是正常的，可以下载到用户的硬件进行测试了。