CLV-8073 用户手册

(MIL-STD-1553B　RS232/422 协议转换模块)

第二版

成都科洛威尔科技有限公司

地址：成都市高新西区双柏路68号23栋

TEL：1878-0222-336 191-3621-6517

EMAIL：clovertech@163.com　公司官网：www.clvtech.net

目 录

文档版本

V1.0.0

编写日期：2017.11.10

说明：初始版本

V1.02.1

编写日期：2017.12.2

说明：

修改 2.1 章节，跟 V1.02.1 硬件设计一致。

修改 2.2 章节“接口位置分布图”，跟 V1.02.1 硬件设计一致。

修改 2.2.2 章节“DSP 启动模式”跳线说明，跟 V1.02.1 硬件设计一致。

修改 2.2.4 章节“1553 信号接口”说明，跟 V1.02.1 硬件设计一致。

修改 2.2.10 章节“LED 信号灯引出接口”说明，跟 V1.02.1 硬件设计一致。

V2.0

编写日期：2021.07.26

说明：

根据 V2.0 版本硬件，更新了第二章 产品外观及接口说明。

更新了第 5 章 主机与协议转换板的交互相关内容。

2021 年 11 月 13 日

更新目录；

更新了 2.2 接口说明；

2022 年 2 月 22 日

增加了附录， 2 上位机端示例程序说明

增加了 3 CLV-8073 B 型使用简介；

2022 年 3 月 12 日

增加 CCS5.5 环境支持；

增加 7.2 CCS5.5 环境下保存数据代码数据文件；

2022 年 3 月 30 日

增加了第 8 章，上位机例程的详细描述；

2024 年 6 月 14 日

补充 A 型，机械尺寸，固定孔孔径。

2025 年 2 月 17 日

B 型，外壳连接器增加 RS422 信号定义；

更新了“模块简介”章节；

1. 模块简介

1.1. 产品描述

CLV-8073是科洛威尔开发的1553B-RS232/422协议转换产品。支持1553B通信数据到RS232/422数据的自定义协议转换。本产品采用了FPGA+DSP处理器构架。DSP处理器完成1553通信接口的配置和收发数据管理、RS-232/422 收发数据管理，以及1553和RS232/422间的数据转换。而两者之间的转换协议可以根据实际应用自定义。

1.2. 产品特性

遵循MIL-STD-1553B Notice2规范；

符合 EIA-RS422/232 标准；

RS422/232模式可选；

开放DSP开发接口；

32位时间标签；

BC功能

支持循环帧发送，帧周期最小为100μs，最大为6.5535s；

支持单帧发送；

支持消息重试，重试条件为：无响应、消息错、总线忙；

重试通道可选，且最大支持7次重试；

消息间隔可设置，为4～65535μs；

支持4种错误注入：校验错、数据字错、同步头错、Bi-phase 错；

支持直接耦合和变压器耦合方式；

RT功能

RT地址软件可设置；

支持4种错误插入：校验错、数据字错、同步头错、Bi-phase 错；

支持单、双Buffer数据缓存；

RS-232/422功能

发送接收FIFO为512字节；

支持CCITT通用波特率，默认频率为115200bps，最高支持921600bps；

校验方式：无校验和ODD，EVEN；

数据位可设置：5，6，7，8 bit；

停止位可设置：1，2bit；

1.3. 开发工具

该模块提供：

DSP示例代码工程；

上位机串口通信协议及基于协议的应用层封装示例（C代码）；

串口操作用户例程；

DSP工程代码开发环境为：Win7 32bit,CCS Version: 5.5.0.00077。

串口操作用户例程开发环境：Win7 32bit,Labwindows CVI。

2. CLV-8073 A型（不含外壳）使用简介

2.1. 产品外观

图 1 CLV-8073 A 型产品外观图

2.2. 机械尺寸

物理尺寸：97mm×52mm；具体结构如图2所示。

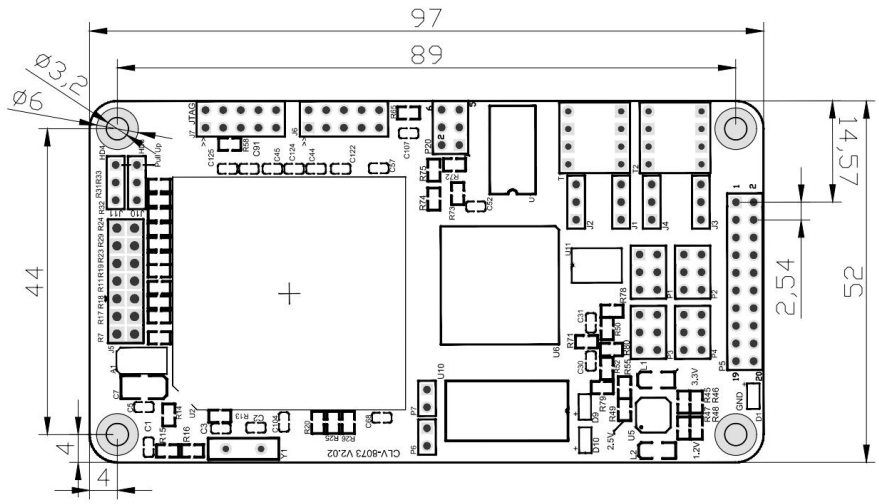


图 2 结构尺寸图

4个固定孔：内径3.2mm;外径6mm。

2.3. 工作环境

工作温度：-20℃～70℃；

存储温度：-40℃～+85℃；

相对湿度：0%～85%（无凝结）；

电源电流：DC 5V ，最大0.2A；

*可提供宽温版本；

2.4. 接口说明

表 1 接口说明表

接口标号	功能
J10	HD3 上下拉跳线
J11	HD4 上下拉跳线
J5	DSP JTAG 连接器
P5	供电、1553 信号、RS422 信号接口
J1、J2、J3、J4	1553 耦合方式选择跳线
P1、P2、P6	串口 0， RS232/422 模式切换跳线
P3、P4、P7	串口 1(保留)， RS232/422 模式切换跳线

各个接口位置分布如图 3 所示：

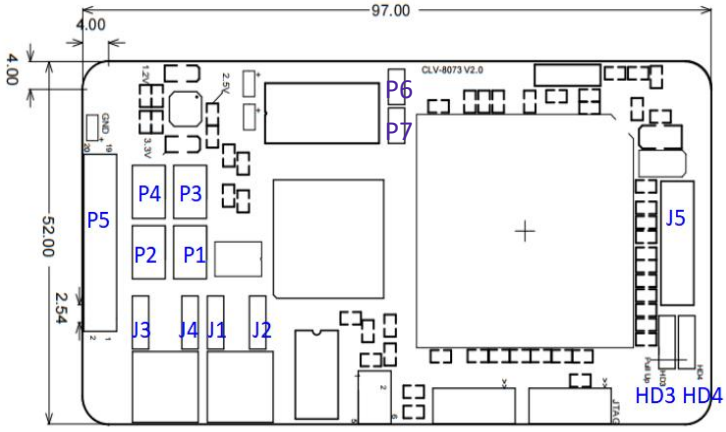


图 3 接口位置分布图

2.4.1. DSP启动模式设置 (J10)

HD3 信号直接接入 DSP，跳线控制 DSP 的启动方式（板卡上有 HD3 标号的位置为 pin1），具体设置如表 2 所示：

表 2 DSP 启动模式设置表

标号	功能	跳线方式		启动方式
J10	HD3 跳线	Pin2-Pin3	HD3=0	HD3=0，仿真器启动。
		Pin2-Pin1	HD3=1	HD3=1 ，DSP 从外部 8 位 ROM 加载程序启动。

2.4.2. DSP JTAG接口 (J5)

表 3 DSP JTAG 接口表

J5	DSP JTAG 连接器

J5 提供 2.54mm 7*2pin 的接口，连接 DSP 仿真器。在出厂时，已经设置好了防插错针脚。

注：CLV-8073 出厂时，默认配置为:从外部 8 位 ROM 加载程序启动。

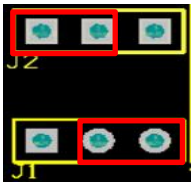
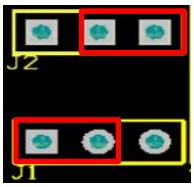
2.4.3. 供电、1553信号、RS422信号接口(P5)

表 4 供电、1553 信号、RS422 信号接口

管脚号	信号	定义	说明
1	M1553B_A+	1553 BUS A 正	
2	M1553B_A-	1553 BUS A 负	
3	M1553B_B+	1553 BUS B 正	
4	M1553B_B-	1553 BUS B 负	
7	RS422_TX0+/RS232_TX0	RS422 通道 0 发送正 /RS232 通道 0 发送信号	
8	RS422_TX0-/GND	RS422 通道 0 发送负/地	
9	RS422_RX0+/RS232_RX0	RS422 通道 0 接收正 /RS232 通道 0 接收信号	
10	RS422_RX0-/GND	RS422 通道 0 接收负/地	
11	RS422_TX1+/RS232_TX1	RS422 通道 1 发送正 /RS232 通道 1 发送信号	保留
12	RS422_TX1-/GND	RS422 通道 1 发送负/地	保留
13	RS422_RX1+/RS232_RX1	RS422 通道 1 接收正 /RS232 通道 0 接收信号	保留
14	RS422_RX1-/GND	RS422 通道 1 接收负/地	保留
5、6、17、18	GND	地	
19、20	5V	DC，5V	
15、16	NC	未定义	

2.4.4. 1553耦合方式选择跳线(J1、J2、J3、J4)

J1、J2 对应 BUSA；J3、J4 对应 BUSB，具体如表 5 所示：

标号	功能	跳线方式	说明
J1、J2	BUSA 信号的耦合		变压器耦合方式
			直接耦合

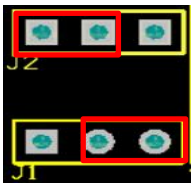
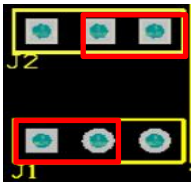
J3、J4	BUSB+信号的耦合 BUSB-信号的耦合		变压器耦合方式
			直接耦合

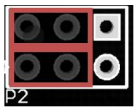
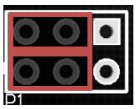
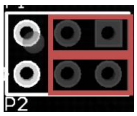
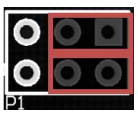
表 5 1553 耦合方式跳线表

注：CLV-8073 出厂时，默认配置为变压器耦合方式，并以 0 欧姆电阻短接的形式（非跳线）进行固定。如需要调整，请提前说明。

2.4.5. 串口0模式跳线（P1、P2）

切换串口模式时需要软件上同步设置相应的模式，具体如表 6 所示。

表 6 串口 0 模式跳线表

跳线 / 模式	P2 跳线（发送）	P1 跳线（接收）	P6 跳线
RS232 模式	pin4-pin6 pin3-pin5 	pin4-pin6 pin3-pin5 	短接
RS422 模式	pin4-pin2 pin3-pin1 	pin4-pin2 pin3-pin1 	断开

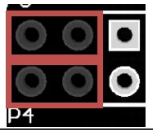
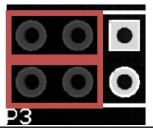
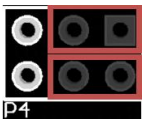
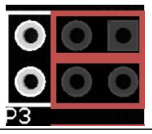
注：

1、CLV-8073 出厂时，默认配置为 RS422 通信方式，并以 0 欧姆电阻短接的形式（非跳线）进行固定。如需要调整，请提前说明。

2、实际应用中推荐首选 RS422 通信模式。

2.4.6. 串口1（保留）模式跳线（P3、P4）

表 7 串口 1 模式跳线表

跳线 / 模式	P4 跳线方式	P3 跳线方式	P7 跳线
RS232 模式	pin4-pin6 pin3-pin5 	pin4-pin6 pin3-pin5 	短接
RS422 模式	pin4-pin2 pin3-pin1 	pin4-pin2 pin3-pin1 	断开

注：

1、CLV-8073 出厂时，默认配置为 RS422 通信方式，并以 0 欧姆电阻短接的形式（非跳线）进行固定。如需要调整，请提前说明。

2、实际应用中推荐首选 RS422 通信模式。

3. CLV-8073 B型（含外壳）使用简介

3.1. 产品外观

图 4 CLV-8073 B型产品外观图

3.2. 机械尺寸

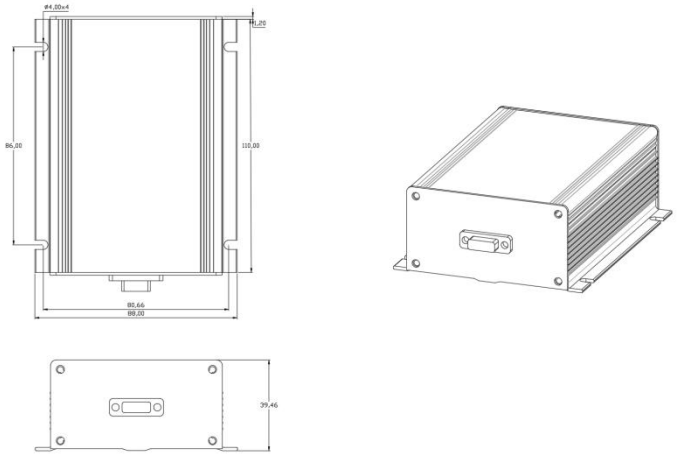


图 5 CLV-8073 B型机械尺寸

3.3. 工作环境

工作温度：-20℃～70℃；

存储温度：-40℃～+85℃；

相对湿度：0%～85%（无凝结）；

电源电流：DC 9~36V， 最大2W；

*可提供宽温版本

3.4. 接口说明

CLV-8073 B型，提供一个J30J-15TJP连接器，信号定义如下：

表 8 CLV-8073 B 型连接器定义

线序号	信号定义	说明

1 2	28V	9~36V 电源输入
3 4	VGND	电源地
5	RS232TX/RS422_T+	RS232 发送信号/RS422 发送正
6	RS232RX/RS422_R+	RS232 接收信号/RS422 接收正
7	GND	信号地
10	M1553B_A+	1553B 总线 A 信号正
11	M1553B_A-	1553B 总线 A 信号负
12	M1553B_B+	1553B 总线 B 信号正
13	M1553B_B-	1553B 总线 B 信号负
14	RS422_T-	RS422 发送负
15	RS422_R-	RS422 接收负

4. 1553功能描述

4.1. 变压器耦合和直接耦合

MIL-STD-1553允许以两种方式连接到总线上：直接耦合和变压器耦合。变压器耦合具有更好的抗干扰能力，能够用在长距离通讯上，要求有外部的变压器。直接耦合不需要外部变压器，但通讯距离有限（最大1英尺，约等于0.3048米），一般用在实验室环境。耦合方式的可以跳线设置，设置方法参考《2.2.4 1553耦合方式选择跳线》。

4.2. 时间戳

时间戳为每个消息的时间标记。在CLV-8073中有一个32Bit的时间寄存器，在一个1MHz的时钟控制下在开始某一功能的同时开始计时。时间寄存器的开始值默认为0，也可以调用API函数，设置不同的开始值。

4.3. 远程终端（Remote Terminals）

本节描述了RT功能和API函数设置。CLV-8073可仿真终端地址0到终端地址31

之间的任何一个RT。RT设置以下信息：

发送、接收子地址；

支持一个或多个发送数据Buffer；

指定RT支持的Mode code；

4.3.1. RT硬件描述

每一个RT（address 0 to 31）有32个子地址。每个RT包含相应控制字，在API_RT_ABUF结构（结构说明见软件手册数据结构章节）中，enable_a、enable_b对应BusA、BusB允许/禁止位，当BusA和BusB都禁止时，1553协议处理模块忽略发送到该RT的消息，当BusA、BusB其中一条或全部允许时，RT响应出现在允许的总线上的命令字。

RT的地址信息中还包含以下信息：

1、16-bit的状态字(status)。状态字在RT响应时由RT发送。“DBA”，“MessageError”，“Broadcast Command Received”，and “Terminal Flag”由1553协议处理模块按1553B规范产生，其它位由应用程序初始化同时不能被1553协议处理模块改变。

2、最后一个指令字(command)。这个指令字由1553协议处理模块编辑，用于响应Mode code 的“发送上一个指令字”。

3、终端标志位(inhibit_term_flag)，由应用程序设置，被1553协议处理单元编辑。用于响应Mode code，禁止终端标志和取消禁止终端标志。

4、1 bit的RT监视模式。该功能保留。当RT在监视模式时RT不响应发到RT的命令字，仅仅记录发送到该RT的消息。

5、自检字(bit_word)，初始化自检字，当BC发送MODE CODE 的发送自检字命令时，RT 发送该初始化的自检字。

当RT 开始仿真时，1553协议处理单元开始处理收到的命令字。当命令字合法时1553协议处理单元将收到的消息存入RT Buffer，RT Buffer 可以设置为单buffer 结构或双buffer结构，由应用程序指定，发送时将循环发送多个数据缓冲区中的数据。消息处理完成后，1553协议处理单元将根据中断位判断是否产生中断，如果要求产生中断，1553协议处理单元 将把中断消息加入中断队列。

4.3.2. RT软件设置流程

本节描述了RT 的软件操作，API函数的详细描述和错误信息请参见《CLV-8073软件参考手册》。

首先调用API函数M1553B_RT_Init()初始化RT模式，初始化操作设置32个RT的初始值和分配存储器。

以下几步主要完成RT的设置：

1．设置RT地址Buffer，在M1553B_RT_AbufWrite()函数中设置BUSA、BUSB允许位，同时设置RT状态字和BIT字。

2．设置RT控制Buffer ，调用函数M1553B_RT_CbufWrite()为（subaddress，transmit/receive）分配数据Buffer，分配的数据Buffer是首尾相连的链表，在发送数据或接收数据时自动循环，同时设置RT允许接收消息。在设置1个RT时可以调用64次M1553B_RT_CbufWrite()函数（ 32个地址，每一个分为发送和接收）。在1553B规范中SA0和SA31表示Mode code方式。

3．初始化RT数据Buffer，M1553B_RT_MessageWrite() 函数设置中断允许位和

写入发送数据。在开始RT仿真之前，要求对每个RT Buffer进行初始化。

4．在以上设置后，可以开始RT仿真。应用程序调用函数M1553B_RT_StartStop()h后开始RT仿真。

5．应用程序在RT开始仿真后，RT在中断控制下工作，dsp可以获得中断类型和存储器地址，应用程序用M1553B_RT_MessageRead()从存储器读RT消息。

6．API函数M1553B_RT_StartStop()能够停止所有RT的仿真。

代码示例：

以下代码实现了RT地址1初始化

```
void RT_Init()
{
    ViUInt32 rtaddr=1;
    //初始化模块的RT功能
    M1553B_RT_Init(0,0);
```

```
    //下面三个函数分别用来配置RT的A-Buffer、C-Buffer和M-Buffer
    //具体这三个函数的代码可以参考demo.c源代码。
    demo_rt_download_abuf(…);
    demo_rt_download_cbuf(…);
    demo_rt_download_mbuf(…);
    //启动RT
    M1553B_RT_StartStop(1);
}
```

然后编写回调函数，RT在收到一条BC消息后，进入中断回调中读数据。中断回调函数可参考《CLV-8073软件参考手册》中的例程。

4.3.3. 动态总线控制（Dynamic Bus Control）

Dynamic Bus Control允许BC转换Bus Control到RT。可以编程允许RT进行Dynamic Bus Control Acceptance。Bus Control发送Dynamic Bus Control (DBC) Mode Code(0000)到RT开始这一请求，必须编程RT接受DBC和并保证其作为一个BC运行。

在BC发出DBC请求后，如果RT的Dynamic Bus Control Acceptance(DBCA)为1，RT 在应答BC时状态字的Bit1为1，同时RT将开始BC控制，API函数M1553B_RT_AbufWrite()设置RT的DBCA位(通过参数inhibit_term_flag)。RT_ABUF_DBC_RT_OFF标志设置在多功能模块中RT开始BC功能时，是否停止RT操作。

在RT接受DBC前要求RT完成BC功能的设置，创建消息并初始化，但不调用M1553B_BC_StartStop 函数，当接受DBC时开始BC操作。

下面代码实现了允许指定RT地址执行DBCA。

```
void Config_abuf(int rtaddr)
{
    API_RT_ABUF abuf;
    abuf.enable_a = 1; // Enable Bus A
    abuf.enable_b = 1; // Enable Bus B
    abuf.inhibit_term_flag = RT_ABUF_DBC_ENA;//允许DBCA
```

```
    abuf.status = 0;
    abuf.command = 0xfefe;
    abuf.bit_word = 0xc0c0;
    status = M1553B_RT_AbufWrite(0,rtaddr,&abuf);
    return status;
}
```

4.4. 总线控制器（Bus Control）

本节描述了BC功能和API函数介绍。BC在总线上是消息的发起者。决定发送哪条消息和在什么时候发送。在一个1553通信网络中必须有一个且只能有一个总线控制器。我们将一组消息叫做Minor（帧），在Minor里可以有多条消息，消息之间可以设置间隔。在BC Buffer中可以定义多个Minor，Minor之间设置一定的发送频率循环发送。

例如：如果定义了3个Minor，第一个5条消息，第2个10条，第3个3条。Minor的发送频率为每100ms发送一个Minor：

当BC开始时，立即开始发送第1帧数据（发送5条消息），然后空闲。

当第2个100ms开始时，发送第2帧数据（发送10条消息），然后空闲。

当第3个100ms开始时，发送第3帧数据（发送3条消息），然后空闲。

如果BC设置为ONESHOT模式时，BC仿真结束。如果为LOOP模时当下一个100ms开始时，重新开始发送第1帧数据。在应用程序设计中必须保证在帧间隔时间里完成消息的发送，如果帧间隔的时间小于发送所需的时间，那么将会出现消息丢失的情况。

4.4.1. BC初始化

在进行BC其它设置之前，要求必须先初始化BC。API函数MC1553_BC_Init()

初始化BC设置并进行Minor Frame频率设置、不响应时间设置、最迟响应时间和重试条件设置。

初始化函数同时还对重试条件进行设置。可以设置在以下情况出现时重试：

BC_RETRY_NRSP (No Response) 总线无响应

BC_RETRY_ME (Message Error) 消息错误

BC_RETRY_BUSY (Busy Bit Set) 总线忙

BC_RETRY_TF (Terminal Flag) (保留)

BC_RETRY_SSF (Subsystem Flag) (保留)

BC_RETRY_INSTR (Instrumentation) (保留)

BC_RETRY_SRQ (Service Request) (保留)

初始化函数仅仅设置重试条件，M1553B_BC_RetryInit()设置重试次数和路径，最多支持七次重试，允许BC消息重试必须通过BC消息管理结构API_BC_MBUF中的control参数OR上BC_CONTROL_RETRY进行设置。

初始化函数设置延迟响应时间和不响应时间。如果在设置的“不响应时间”范围内RT没有响应，那么BC将视为RT无响应，如果RT响应在“延迟响应时间”到“不响应时间”之间，BC将发出一个RT迟响应的消息。在1553规范中“不响应时间”规定为14us，“延迟响应时间”规定为12us。

初始化函数设置Minor 发送间隔时间，如设置为100ms，消息队列中有三个Minor，那么发送完成3 个Minor 需300ms。

M1553B_BC_Trigger函数设置BC触发方式：在CLV-8073中只支持立即触发模式—BC_TRIGGER_IMMEDIATE。

Examples：

```
void BC_INIT(void) //set BC mode
{
    ViUInt16 wRetry=0;
    ViUInt32 RtRespTime=14,RtDRespTime=12;
    ViUInt32 framerate = 100000; //Minor发送间隔时间为100ms
    ViUInt16 retrymode[8];
    int triggermode;
    if(tryAble)//如果允许重试，通过wRetry变量设置重试条件
    {
        wRetry=wRetry| BC_RETRY_ME;
        wRetry=wRetry| BC_RETRY_NRSP;
```

```
        wRetry=wRetry| BC_RETRY_BUSY;
    }
    hr =M1553B_BC_Init(
        0,
        0,
        0,              // (i) interrupt enables
        wRetry,         // (i) See Retry Enable definitions below
        RtRespTime,     // (i) no response time out
        RtDRespTime,    // (i) late response
        framerate,      // (i) Period of a minor frame, in microseconds
        0);             // (i) number of BC message buffers ( 0 or1)
    //设置重试条件后，通过retrymode变量设置重试路径及次数
    // RETRY_END重试结束符，支持最大7次重试。
    retrymode[0] = RETRY_SAME_BUS;
    retrymode[1] = RETRY_SAME_BUS;
    retrymode[2] = RETRY_SAME_BUS;
    retrymode[3] = RETRY_ALTERNATE_BUS;
    retrymode[4] = RETRY_ALTERNATE_BUS;
    retrymode[5] = RETRY_ALTERNATE_BUS;
    retrymode[6] = RETRY_SAME_BUS;
    retrymode[7] = RETRY_END;
    hr=M1553B_BC_RetryInit(0,retrymode);
    //trigger mode
    triggermode = BC_TRIGGER_IMMEDIATE;
    hr=M1553B_BC_Trigger(0, triggermode);
}
```

4.4.2. BC 消息类型

BC消息结构(API_BC_MBUF)定义了Minor 和消息及各种控制信息，包括Minor

开始、结束，消息间隔等信息。

BC消息包括命令字、数据指针、下一个消息指针、消息间隔，当消息为第1条时消息间隔被忽略。在函数参考手册中详细的描述了消息定义的含义。

API_BC_MBUF中的各项参数进行设置：

control ：此参数设置BC消息类型和设置控制字。

消息类型：

BC_CONTROL_MESSAGE

BC_CONTROL_BRANCH

BC_CONTROL_CONDITION

BC_CONTROL_CONDITION3

BC_CONTROL_NOP

消息控制字：

BC_CONTROL_MFRAME_BEG，帧开始标志。一帧的第一条消息必须加上该标志。

BC_CONTROL_MFRAME_END，帧结束标志。一帧的最后一条必须加上该标志位。

BC_CONTROL_INTERRUPT，保留，设置本消息允许产生中断。

BC_CONTROL_RETRY，允许本消息重试。重试条件在M1553B_BC_Init中设置。

BC_CONTROL_CHANNELA，本消息在Bus A发送时。

BC_CONTROL_CHANNELB，本消息在Bus B发送时。

messno 消息编号，此序列号是唯一的。

messno_next 设置BC下一条消息的编号（最后一条该项固定设置为0xFFFF）。

messno_prev 保留，暂时未用。

errorid 插入的消息错误，可选值：

EI_NONE(不插入错误)

EI_PARITY（ 校验错）

EI_WORDCOUNT（ 数据字错）

EI_SYNC（ 同步头错）

EI_BIPHASE（Bi-phaseerror）。

gap_time间隔时间，从上一条消息发送完开始计数（4~65535us）。

data[0][0-31] 初始化发送Buffer A。

data[1][0-31] 初始化发送Buffer B。

mess_command2 命令字2，仅用在RT to RT和RT to RT Broadcast时。

mess_command1命令字1，设置见如下例程。

mess_command1.rtaddr 设置RT 地址，有效地址为(0-30) ，31为广播消息。

mess_command1.tran_rec发送接收位：

1表示RT应发送

0表示RT应接受

在Mode code时由mode code确定。

mess_command1.subaddr设置RT 子地址，0和31表示此消息是mode code.

mess_command1.wcount设置发送或接收WORD数量（0代表32），或Modecode。

data_value这是一个16bit的数据，用于在条件分支中作为比较值。

data_mask这是一个16bit的数据，用于在条件分支中作为比较值数据掩码。

address: 这个参数用于设置哪一个数据用于比较，可能的值有，

0 Command word.

1 Command word #2 (for “RT-to-RT” messages only).（保留）

2 Status word.

3 Status word #2 (for “RT-to-RT” messages only). （保留）

4～35 Data words #1 through #32.

messno_branch：设置条件比较为真时下一条执行消息号。

当比较结果为假时“messno_next” 参数指定的消息将执行。

messno_compare：在BC_CONTROL_CONDITION3 参数时指定比较的消息。

使用BC_CONTROL_CONDITION 参数时，比较的消息固定为上一条。

在BC开始之前要求初始化BC消息队列，API函数M1553B_BC_MessageWrite()初始化消息队列。在定义消息队列时，每个消息中的message no是唯一的，从0开始依次递增的。如果你定义了3个Minor每个10条消息，那么你需要初始化30个message，NO.从0到29。在定义Minor时，关键是定义Minor的开始和结尾。Message Buffer中的Control Word中分别与BC_CONTROL_MFRAME_BEG、CONTROL

_MFRAME_END进行“逻辑或”表示Minor开始和结束。在Minor的第1条消息中与BC_CONTROL_MFRAME_BEG进行“逻辑或”，在最后一条与CONTROL _MFRAME_END进行“逻辑或”，在Minor中可以有任意数量的Message。如果你只有1条Message，那么BC_CONTROL_MFRAME_BEG、CONTROL_MFRAME_END将同时出现在message上。

我们可以定义多个Minor ， 每出现一对BC_CONTROL_MFRAME_BEG 、CONTROL_MFRAME_END 操作便定义一个Minor，BC 开始发送数据从第一个Minor的第一条消息开始，从No.0 开始每次增加1。Minor 控制BC 的传输，在message中定义了BUS 、Gap time 、message type 和发送的数据。调用函数M1553B_BC_MessageWrite()完成以上功能。

Examples：

下面的代码初始化了1 个Minor,这个Minor 中只有一条消息BC->RT)。

```
Void BC_SingleMsg_Init()
{
    API_BC_MBUF bcmessage;
    ViUInt16 subaddr=2,count=10,chnum=0;
    ViUInt16 subaddr2=1,count2=10, messno_bc=0;
    ViStatus status=0;
    bcmessage.messno = messno_bc;
    bcmessage.messno_next=0xffff;
    bcmessage.control = BC_CONTROL_MESSAGE;
    bcmessage.control |= BC_CONTROL_BUFFERA; // use buffer A
    //bcmessage.control |= BC_CONTROL_RETRY;//如需重试，则需此控制
    bcmessage.control |= BC_CONTROL_CHANNELA; // Channel A
    bcmessage.control |= BC_CONTROL_MFRAME_BEG;
    bcmessage.control |= BC_CONTROL_MFRAME_END;
    bcmessage.mess_command1.subaddr =subaddr;
    bcmessage.mess_command1.tran_rec = 0; //receive
    bcmessage.mess_command1.rtaddr = rtaddr;
```

```
    bcmessage.mess_command1.wcount = 0; //发送32个数据
    for (j=0; j<32; j++)
    {
      bcmessage.data[0][j] = 0x5a5a+j;
      bcmessage.data[1][j] = 0x5a5a+j;
    }
    M1553B_BC_MessageWrite(0,messno_bc,&bcmessage);
}
```

下面函数定义了一个Minor，其中有三条消息：BC->RT、RT->BC、RT->RT。

```
Void BC_ThreeMsg_INIT()
{
  API_BC_MBUF api_message[3];
  ViUInt16 cmdword=0;
  ViUInt16 k=0,i=0,count, modecode;
  ViStatus hr=0;
  ViUInt32 framerate=100000, tryAble=0;
  char databuf[200],databuf1[50];
  //msg0 BC->RT
  api_message[i].control =BC_CONTROL_MESSAGE;
  api_message[i].control |=BC_CONTROL_MFRAME_BEG;//Minor 开始标志
  api_message[i].control |=BC_CONTROL_CHANNELA;
  api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
  api_message[i].gap_time =4;
  api_message[i].messno =0;
  api_message[i].messno_next=0x1;
  api_message[i].mess_command1.rtaddr=1;
  api_message[i].mess_command1.tran_rec=0;
  api_message[i].mess_command1.subaddr=1;
```

```
    api_message[i].mess_command1.wcount=10;
    api_message[i].errorid =0 ;
    for (k=0;k<32;k++)
    api_message[i].data [0][k]=0xf000+k;
    M1553B_BC_MessageWrite(0,api_message[i].messno,&api_message[i])
;
    //msg 1 RT->BC
    i++;
    api_message[i].control =BC_CONTROL_MESSAGE;
    api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
    api_message[i].control |=BC_CONTROL_CHANNELA;
    api_message[i].gap_time =4;
    api_message[i].mess_command1.rtaddr=2;
    api_message[i].mess_command1.tran_rec=1;
    api_message[i].mess_command1.subaddr=2;
    api_message[i].mess_command1.wcount=10;
    api_message[i].messno =i;
    api_message[i].messno_next=0x2;
    api_message[i].errorid =0 ;
    M1553B_BC_MessageWrite(0,api_message[i].messno,&api_message[i])
;
    //msg 2
    i++;
    api_message[i].control =BC_CONTROL_MESSAGE;
    api_message[i].control |=BC_CONTROL_CHANNELA;
    api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
    api_message[i].control |=BC_CONTROL_MFRAME_END;//minor 结束标志
    api_message[i].gap_time =4;
```

```
    api_message[i].messno =i;
    api_message[i].messno_next=0xffff;
    api_message[i].errorid =0 ;
    api_message[i].control |=BC_CONTROL_RTRTFORMAT;
    api_message[i].mess_command1.rtaddr=rtaddr;
    api_message[i].mess_command1.tran_rec=0;//tr;
    api_message[i].mess_command1.subaddr=subaddr;
    api_message[i].mess_command1.wcount=count;
    api_message[i].mess_command2.rtaddr=rtaddr2;
    api_message[i].mess_command2.tran_rec=1;//tr2;
    api_message[i].mess_command2.subaddr=subaddr2;
    api_message[i].mess_command2.wcount=count2;// count须等于count2
    M1553B_BC_MessageWrite(
        0,api_message[i].messno,&api_message[i]);
}
```

以下代码实现了2 个Minor，每个Minor 有2 个消息，共四个消息

```
Void DoubleMinor_Init()
{
    API_BC_MBUF api_message[4];
    ViUInt16 cmdword=0;
    ViUInt16 k=0,i=0,count;
    ViStatus hr=0;
    ViUInt32 framerate=100000, tryAble=0;
    char databuf[200],databuf1[50];
    //Minor0，msg0 BC->RT
    api_message[i].control =BC_CONTROL_MESSAGE;
    api_message[i].control |=BC_CONTROL_MFRAME_BEG;//Minor0 开始标志
    api_message[i].control |=BC_CONTROL_CHANNELA;
```

```
api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
api_message[i].gap_time =4;
api_message[i].messno =0;
api_message[i].messno_next=0x1;
api_message[i].mess_command1.rtaddr=1;
api_message[i].mess_command1.tran_rec=0;
api_message[i].mess_command1.subaddr=1;
api_message[i].mess_command1.wcount=10;
api_message[i].errorid =0 ;
for (k=0;k<32;k++)
    api_message[i].data [0][k]=0xf000+k;
M1553B_BC_MessageWrite(
    0,api_message[i].messno,&api_message[i]);
///Minor0, msg 1 RT->BC
i++;
api_message[i].control =BC_CONTROL_MESSAGE;
api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
api_message[i].control |=BC_CONTROL_CHANNELA;
api_message[i].control |=BC_CONTROL_MFRAME_END;//minor0 结束标志
api_message[i].gap_time =4;
api_message[i].mess_command1.rtaddr=2;
api_message[i].mess_command1.tran_rec=1;
api_message[i].mess_command1.subaddr=2;
api_message[i].mess_command1.wcount=10;
api_message[i].messno =i;
api_message[i].messno_next=0x2;
api_message[i].errorid =0 ;
M1553B_BC_MessageWrite(
```

```
    0,api_message[i].messno,&api_message[i]);
//Minor1,  msg 2 RT->RT
i++;
api_message[i].control =BC_CONTROL_MESSAGE;
api_message[i].control |=BC_CONTROL_CHANNELA;
api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
api_message[i].control |=BC_CONTROL_MFRAME_BEG;//Minor1 开始标志
api_message[i].gap_time =4;
api_message[i].messno =i;
api_message[i].messno_next=3;
api_message[i].errorid =0 ;
api_message[i].control |=BC_CONTROL_RTRTFORMAT;
api_message[i].mess_command1.rtaddr=rtaddr;
api_message[i].mess_command1.tran_rec=0;//tr;
api_message[i].mess_command1.subaddr=subaddr;
api_message[i].mess_command1.wcount=count;
api_message[i].mess_command2.rtaddr=rtaddr2;
api_message[i].mess_command2.tran_rec=1;//tr2;
api_message[i].mess_command2.subaddr=subaddr2;
api_message[i].mess_command2.wcount=count2;// count 须等于count2
M1553B_BC_MessageWrite(
    0,api_message[i].messno,&api_message[i]);
//Minor1,  msg 3 modecode
i++;
api_message[i].control =BC_CONTROL_MESSAGE;
api_message[i].control |=BC_CONTROL_CHANNELA;
api_message[i].control |= BC_CONTROL_BUFFERA; // use buffer A
api_message[i].control |=BC_CONTROL_MFRAME_END;//minor1 结束标志
```

```
    api_message[i].gap_time =4;
    api_message[i].messno =i;
    api_message[i].messno_next=0xffff;
    api_message[i].errorid =0 ;
    api_message[i].mess_command1.subaddr =0;
    if ((modecode<=16) ||(modecode==18)||(modecode==19))
        api_message[i].mess_command1.tran_rec = 1;
    else
        api_message[i].mess_command1.tran_rec = 0;
    api_message[i].mess_command1.rtaddr = rtaddr;
    api_message[i].mess_command1.wcount =modecode;
    M1553B_BC_MessageWrite(
        0,api_message[i].messno,&api_message[i]);
}
```

4.4.3 插入消息（Aperiodic 1553 BC Messages）

在BC消息中，一般来说发送的数据是一个可重复发送的消息序列，我们叫作Minor或bus message List。Aperiodic Messages是插入到bus message List中的一次性发送的数据，你可以发送任意种类的消息。但Aperiodic Messages不能在错误时重试。应用程序调用M1553B_BC_AperiodicRun()开始发送Aperiodic Messages，此操作

可以在开始BC前或后调用。Aperiodic Messages有两种类型的消息：

High Priority

Low Priority（保留）

高优先级的消息在1553协议处理单元发送完一条消息后，立即开始发送（如果存在高优先级的消息），低优先级的消息在发送完一个Minor后开始发送。需要注意的是在加入Aperiodic Messages时同样应保证在帧间隔时间内能完成所有消息的发送且使用M1553B_BC_AperiodicRun函数插入的消息ID，不能和现有的Minor或Bus List中的消息ID重复。

Examples:

本例在 3.5.2例子函数 void BC_ThreeMsg_INIT()函数中定义的Minor(其中有三条消息：BC->RT、RT->BC、RT->RT）基础上，插入一条BC->RT 的一次性发送消息。

```
void Aperiodic_insert(void)
{
  int ch=0,k=0;
  API_BC_MBUF api_message;
  api_message.control =BC_CONTROL_MESSAGE;
  api_message.control |=BC_CONTROL_CHANNELA;
  api_message.control |= BC_CONTROL_BUFFERA; // use buffer A
  api_message.errorid =0 ;
  api_message.gap_time = 4;
  api_message.messno =3;
  api_message.mess_command1.rtaddr=1;
  api_message.mess_command1.tran_rec=0;
  api_message.mess_command1.subaddr=1;
  api_message.mess_command1.wcount=1;
  for (k=0;k<32;k++)
  api_message.data [0][k]=0xf000+k;
  M1553B_BC_MessageWrite(0,api_message.messno,&api_message);
  M1553B_BC_AperiodicRun(
      0,
      api_message.messno,
      0, // 1 -> Hi Priority msgs, 0 -> Low Priority msgs
      0, // 1 -> Wait for BC to complete executing msgs
      0); // Timeout in seconds(16-bit) or milliseconds(32-bit)
}
```

4.4.4 条件分支消息(Conditional Branch BC Message Block)

条件分支用于动态的改变消息帧的执行顺序。BC 按给定的测试条件决定下一条发送的消息，测试条件你可以在API 函数中设置，包含：测试值，数据掩码和比较的消息。根据指定的参数1553协议处理单元测试选择的数据字，根据测试结果执行不同的消息。选择的测试数据字可以为命令字、状态字和数据字。条件分支消息不发送任何的数据到总线上，也不占用任何的额外时间，它的执行与上一条消息的执行时同步的。

在BC 跳转控制(API_BC_MBUF 结构的control 参数控制，具体说明请参照软件参考手册中)有三种跳转方式：

BC_CONTROL_BRANCH：此消息类型定义了一个分支语句，无条件跳转到messno_branch 。

BC_CONTROL_CONDITION:此类消息根据与上一条消息的比较，BC 跳转到指定的消息。

BC_CONTROL_CONDITION3: 此类消息根据与指定消息的比较，BC 跳转到指定的消息

Examples：

下面的例子实现了使用BC_CONTROL_CONDITION3 控制跳转：该Minor 中有四条消息，第一条是RT->BC 消息，第二条是跳转消息，该消息根据比较第一条消息中BC 接收到从RT 发来的的第一个数据是否为0x101 进行判断，如果是，则调转到最后一条消息。第三条、第四条为BC->RT 消息。

```
void BC_CondiBranch()
{
  int msno=0,k=0;
  API_BC_MBUF api_message;
  //msg0 RT->BC
  api_message.control =BC_CONTROL_MESSAGE;
  api_message.control |=BC_CONTROL_MFRAME_BEG;
  api_message.control |=BC_CONTROL_CHANNELA;
```

```
api_message.control |= BC_CONTROL_BUFFERA; // use buffer A
api_message.errorid =0 ;
api_message.gap_time = 50;
api_message.messno =msno;
api_message.messno_next=msno+1;
api_message.mess_command1.rtaddr=1;
api_message.mess_command1.tran_rec=1;
api_message.mess_command1.subaddr=1;
api_message.mess_command1.wcount=5;
M1553B_BC_MessageWrite(0,api_message.messno,&api_message);
//msg1
memset(&api_message,0,sizeof(api_message));
msno++;
api_message.control|=BC_CONTROL_CONDITION3;
api_message.control |=BC_CONTROL_CHANNELA;
api_message.control |= BC_CONTROL_BUFFERA; // use buffer A
api_message.errorid =0 ;
api_message.gap_time = 50;
api_message.messno =msno;
api_message.messno_next=msno+1;
api_message.data_value=0x101 ; //如果为无条件跳转，该参数保留
api_message.data_mask=0x101;//如果为无条件跳转，该参数保留
api_message.messno_compare=0;//如果为BC_CONTROL_CONDITION，保留
api_message.messno_branch=3;
api_message.address=4; //如果为无条件跳转，该参数保留
M1553B_BC_MessageWrite(0,api_message.messno,&api_message);
//msg2
memset(&api_message,0,sizeof(api_message));
```

```
msno++;
api_message.control =BC_CONTROL_MESSAGE;
api_message.control |=BC_CONTROL_CHANNELA;
api_message.control |= BC_CONTROL_BUFFERA; // use buffer A
api_message.errorid =0 ;
api_message.gap_time = 50;
api_message.messno =msno;
api_message.messno_next=msno+1;
api_message.mess_command1.rtaddr=1;
api_message.mess_command1.tran_rec=0;
api_message.mess_command1.subaddr=3;
api_message.mess_command1.wcount=5;
for (k=0;k<32;k++)
api_message.data [0][k]=0x3;
M1553B_BC_MessageWrite(0,api_message.messno,&api_message);
//msg3
memset(&api_message,0,sizeof(api_message));
msno++;
api_message.control =BC_CONTROL_MESSAGE;
api_message.control |=BC_CONTROL_CHANNELA;
api_message.control |= BC_CONTROL_BUFFERA; // use buffer A
api_message.control |=BC_CONTROL_MFRAME_END;
api_message.errorid =0 ;
api_message.gap_time = 50;
api_message.messno =msno;
api_message.messno_next=0xffff;
api_message.mess_command1.rtaddr=1;
api_message.mess_command1.tran_rec=0;
```

```
    api_message.mess_command1.subaddr=4;
    api_message.mess_command1.wcount=5;
    for (k=0;k<32;k++)
        api_message.data [0][k]=0x4;
    M1553B_BC_MessageWrite(0,api_message.messno,&api_message);
}
```

4.4.5 BC端动态总线控制（Dynamic Bus Control）

BC可以发送DBC（Mode code 0）要求Remote Terminal开始总线控制。如果RT响应的DBCA位为1，BC停止总线控制并将总线控制权交给RT；如果为0，BC继续总线控制。

4.4.6 BC 消息错误插入

BC可以插入的消息错误有EI_NONE(不插入错误)、EI_PARITY（校验错）、EI_WORDCOUNT（ 数据字错）、EI_SYNC（ 同步头错）、EI_BIPHASE（Bi-phase error），

由API_BC_MBU的errorid参数控制。

```
API_BC_MBU bcmessage;
bcmessage.errorid = EI_SYNC表示插入一个同步头错。
Bcmessage.errorid = EI_NONE表示不插入错误。
```

4.5. 1553自检(保留)

1553B支持两种自检模式，内部自检和外部自检。分别使用下面两个函数：M1553B_BIT_InternalBit 函数执行一个内部测试，不需要外部将同一通道的Bus A 与Bus B 的连接。此项功能可以运行在单通道、多通道；单功能或多功能的模块中，在测试时不必进行BC、BM 或RT 设置。BC 发送指定组数据分别到BM、RT。1553协议处理单元将BM、RT收到的数据与BC发送的数据比较，如果接收的数据都与发送的数据相等，那么一次测试通过。NumMessages 参数指定测试次数，可以为多次。每次的数据将不同，当多次测试都通过时，那么本次测试通过。

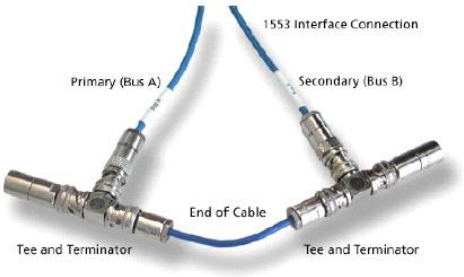


图 6 Bus A 与Bus B 外部连接图

M1553B_BIT_CableWrap 函数提供了一个在Bus A 和Bus B 上的测试（外回环测试），在测试时要求将Bus A 与Bus B 外部相连，如图4。

外部测试测试方式与内部测试方法相同，差别在于M1553B_BIT_InternalBit仅仅在测试内部的存储器、处理器等器件;而外部测试需要外部将BUSA和BUSB连接起来,进行存储器级到外部连接器的从内到外测试。当测试完成后，1553 模块被重新复位，所有的通道设置都要求重新设置。

5. 串口功能描述

模块的串口模式支持RS-232、RS-422模式，串口参数由API函数CLV_Serial_Config()设置。

5.1. 多种波特率支持

可设置的波特率为921600bps~300bps，在DSP工程SerialAPI.h文件中有如下定义：

```
#define  BIT_RATE_921600    2
#define  BIT_RATE_460800    3
#define  BIT_RATE_230400    4
#define  BIT_RATE_115200    5
#define  BIT_RATE_57600     6
#define  BIT_RATE_38400     7
#define  BIT_RATE_28800     8
#define  BIT_RATE_19200     9
#define  BIT_RATE_14400    10
```

```
#define  BIT_RATE_9600     11
#define  BIT_RATE_7200     12
#define  BIT_RATE_4800     13
#define  BIT_RATE_2400     14
#define  BIT_RATE_1200     15
#define  BIT_RATE_600      16
#define  BIT_RATE_300      17
```

5.2. 支持多种发送校验方式

发送校验方式有3种。在DSP工程SerialAPI.h文件中有如下定义：

```
#define  PARITY_NONE       0 //无校验
#define  PARITY_ODD        2 //奇校验
#define  PARITY_EVEN       3 //偶校验
```

5.3. 数据位可设置

数据位可设置为5bit/6bit/7bit/8bit。在DSP工程SerialAPI.h文件中有如下定义：

```
#define  BITCNT_8          0 //数据位8
#define  BITCNT_7          1 //数据位7
#define  BITCNT_6          2 //数据位6
#define  BITCNT_5          3 //数据位5
```

5.4. 停止位可设置

停止位可设置为1位、2 位。在DSP工程SerialAPI.h文件中有如下定义：

```
#define  STOPBIT_1         1 //1位停止位
#define  STOPBIT_2         2 //2位停止位
```

5.5. 数据收发

串口发送接收FIFO 大小为512 字节。收发数据分别采用CLV_Serial_Recv和CLV_Serial_Send 函数实现，这两个函数的返回值为实际接收或发送数据的字节个数。

5.6. 默认状态

产品串口默认设置为：波特率115200bps、无校验位、停止位为1位、数据位为8 位。

6. 主机与协议转换板的交互

6.1. 主机与协议转换板的连接

主机可以通过串口与协议转换板进行对接。在主机端可以使用任何通用串口程序（工具）与协议转换板进行通信。请保持通信两端串口的波特率等参数设置一致（转换板的参数设置在 DSP 代码中完成）。

6.2. 通信协议

主机和协议转换板之间的通信还需要在两者之间约定一套用于通信的协议。比如说DSP示例工程interrupt.c文件中有这样的定义：

```
#define     CMDHEADER_RT_INIT          0xA5A1
#define     CMDHEADER_RT_TXDATA        0xA5A2
#define     CMDHEADER_RT_STARTSTOP     0xA5A3
#define     CMDHEADER_RT_READ          0xA5A4
#define     CMDHEADER_BC_INIT          0xA5B1
#define     CMDHEADER_BC_MSGWRITE      0xA5B2
#define     CMDHEADER_BC_STARTSTOP     0xA5B3
#define     CMDHEADER_BC_READ          0xA5B4
#define     CMDHEADER_BM_INIT          0xA5C1
#define     CMDHEADER_BM_STARTSTOP     0xA5C2
#define     CMDHEADER_BM_READ          0xA5C3
#define     CMDHEADER_FLASH_PROGRAM    0xA555
#define     CMDHEADER_GET_VERSION      0xA566
#define     CMDHEADER_SERIAL_PARA      0xA5F5
```

我们用这种方式定义了一套简单的协议，比如当主机向协议转换板发送

A5B1xxxxxxxxxxxx 时要求协议转换板初始化 BC 功能。例程中用一串 16 进制数据来表示一些通信约定，这也要求主机端串口必须以 16 进制方式进行数据发送；协议转换板收到串口数据后将会对命令进行解析，并执行相应的动作。关于命令的详细说明，请参见《CLV-8073 模块交互通信协议 V2.0》。

当然这只是例程中的定义和处理方式，用户完全可以根据实际应用情况对这些“通信协议”进行修改或重定义。

7. DSP程序的下载与更新

建议在RS422通信模式下进行程序下载更新。

协议转换模块的DSP代码的固化，也即DSP的烧写，可通过串口进行。专门提供了Windows系统下的串口发送工具，向协议转换板上DSP发送烧写数据，DSP工程中有支持FLASH烧写的代码，两者相互配合完成代码的固化。

注意：

1、本章介绍如何通过串口更新CLV-8073的固件。如果更新中断或其他形式的更新失败，将导致模块无法正常工作。

2、在进行7.3节所述程序烧写之前请做好相关准备：

(1) 请保证RS422通信可靠连接；

(2) 保证电源可靠持续供电；

3、如果烧写后CLV-8073无法正常工作，可设置DSP启动模式为“仿真器启动”模式（2.4.1 DSP启动模式设置 (J10)），接DSP仿真器进入调试模式，运行DSP程序，重新固化正确的程序。操作中如有疑问，可直接联系我们。

首先在CCS环境下做开发调试。当代码功能调试完成后，进行如下步骤进行代码的固化：

7.1. CCS2.2 、CCS3.3环境下保存代码数据文件

（1）仿真器模式下，编译好目标文件，加载到 DSP 中，但不运行，然后在 CCS 上操作：

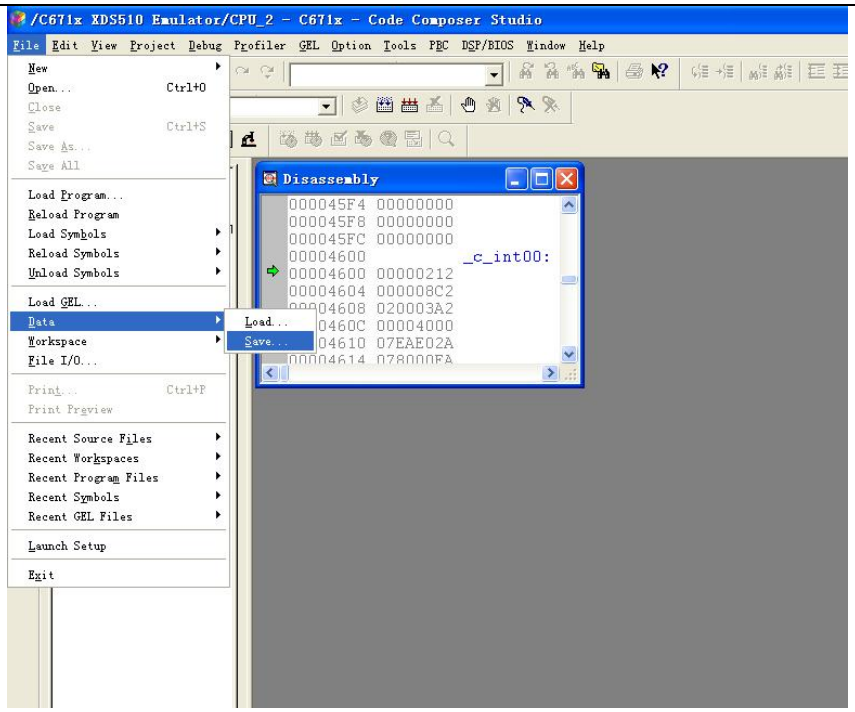


图 7 加载到 DSP 图

（2）出现保存设置窗口，首先将文件名保存成 data.bat（为了与主机串口烧写程序保持一致，这个名字固定为 data.bat）：

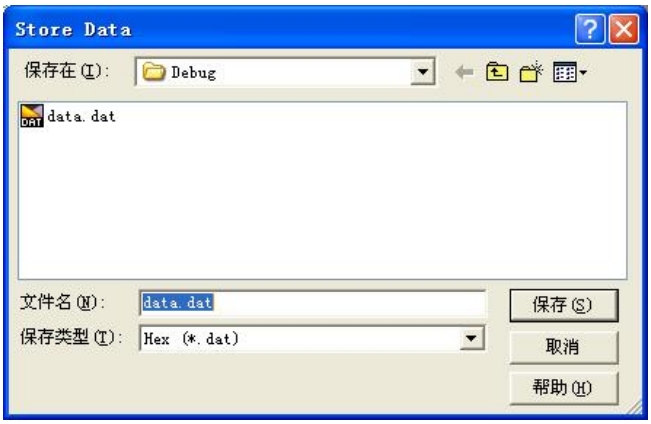


图 8 保存设置窗口图

（3）点击保存后出现地址和长度设置将窗口中的地址设置为 0x00000000，长度设置为 0xC000：

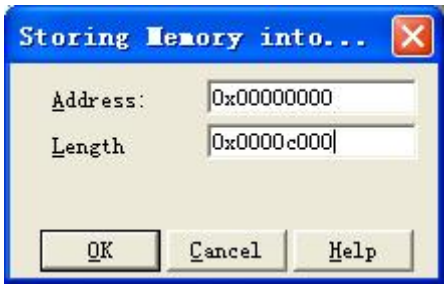


图 9 地址与长度设置窗口图

点击OK，进行保存。这个过程需要一点时间。

（4）完成上述步骤后，关闭协议转换板电源，取下仿真器，并将J10、J11跳线设置为DSP自启动模式。跳线设置方法见2.2.1节。

7.2. CCS5.5环境下保存数据代码数据文件

仿真器模式下，编译好目标文件，加载到 DSP 中，但不运行，然后在 CCS 上操作：

（1）通过 view 菜单下的 Memory browser 打开内存查看窗口。在 Memory browser 窗口内起始地址填 0，点击“go”按钮，读取从地址 0 开始的数据内容。

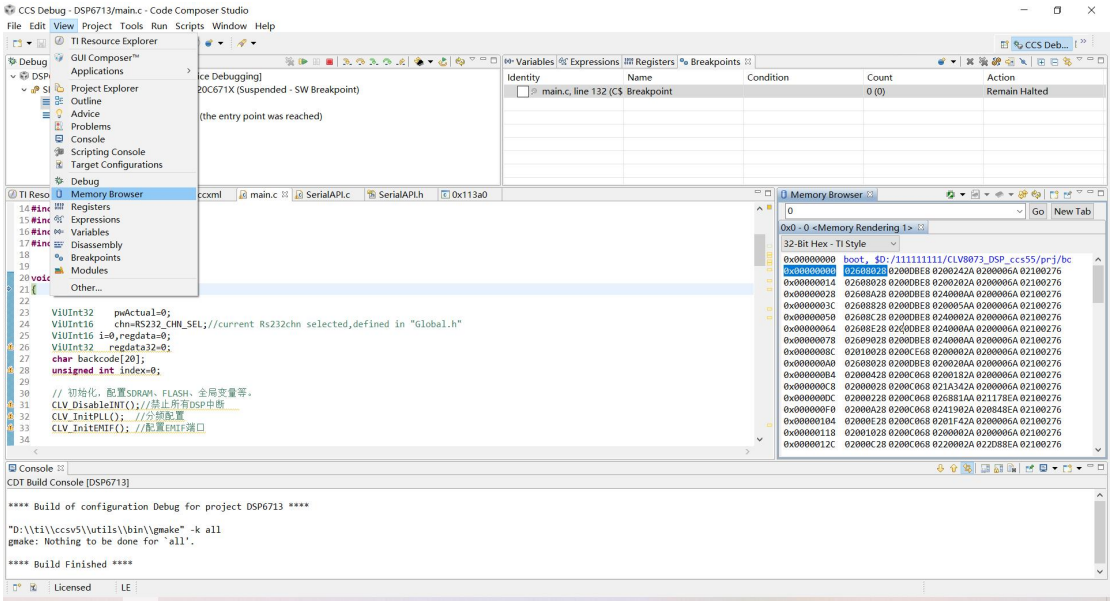


图 10 打开 memory browser

（2）Memory browser 窗口的数据区点击鼠标右键，在右键菜单中选择“save memory...”项：

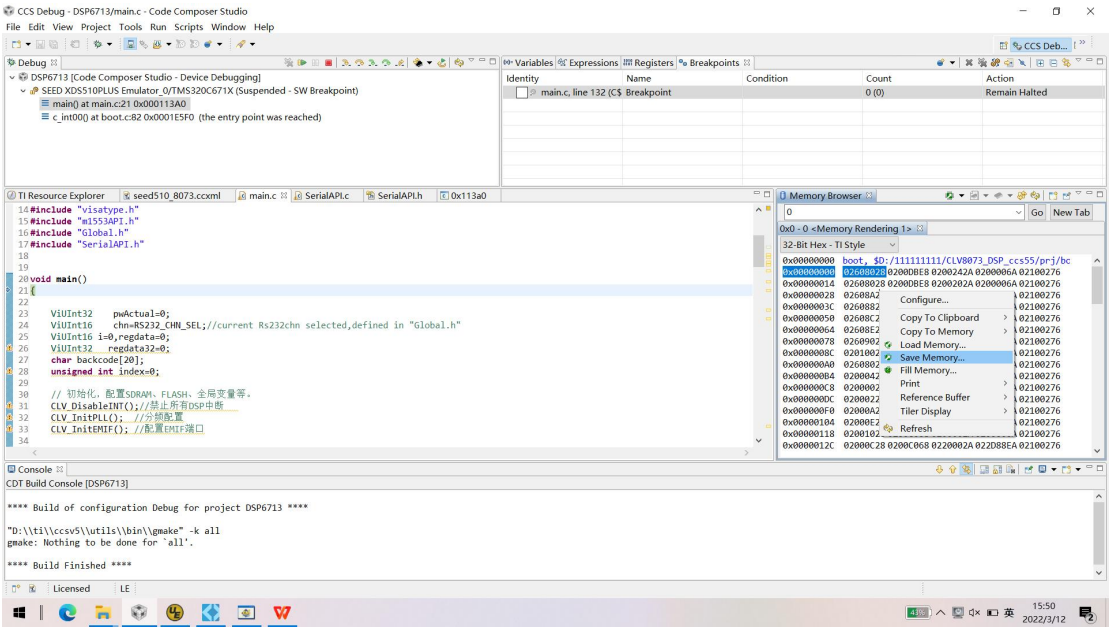


图 11 点击“save memory”选项

（3）在弹出的路径设置界面里，选择文件保存路径，文件名必须是 data.dat。

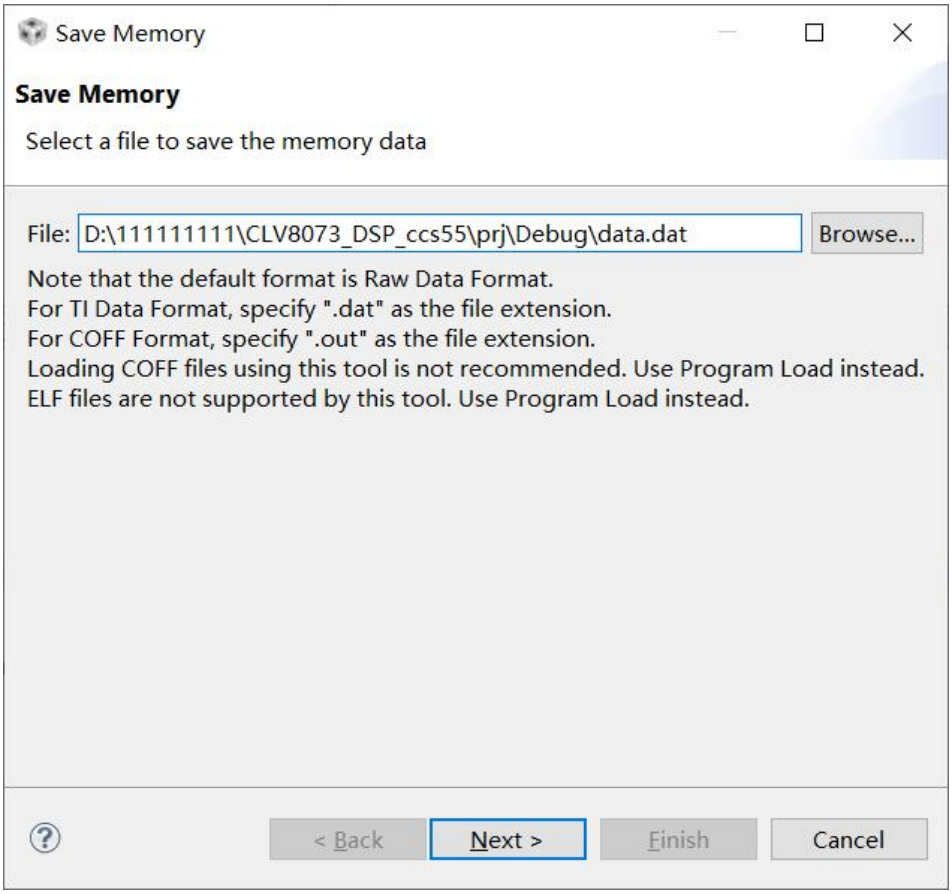


图 12 设置文件保存路径

（4）点击“Next”进入下一步，设置保存格式、起始地址和数据长度，设置值如下图所示：

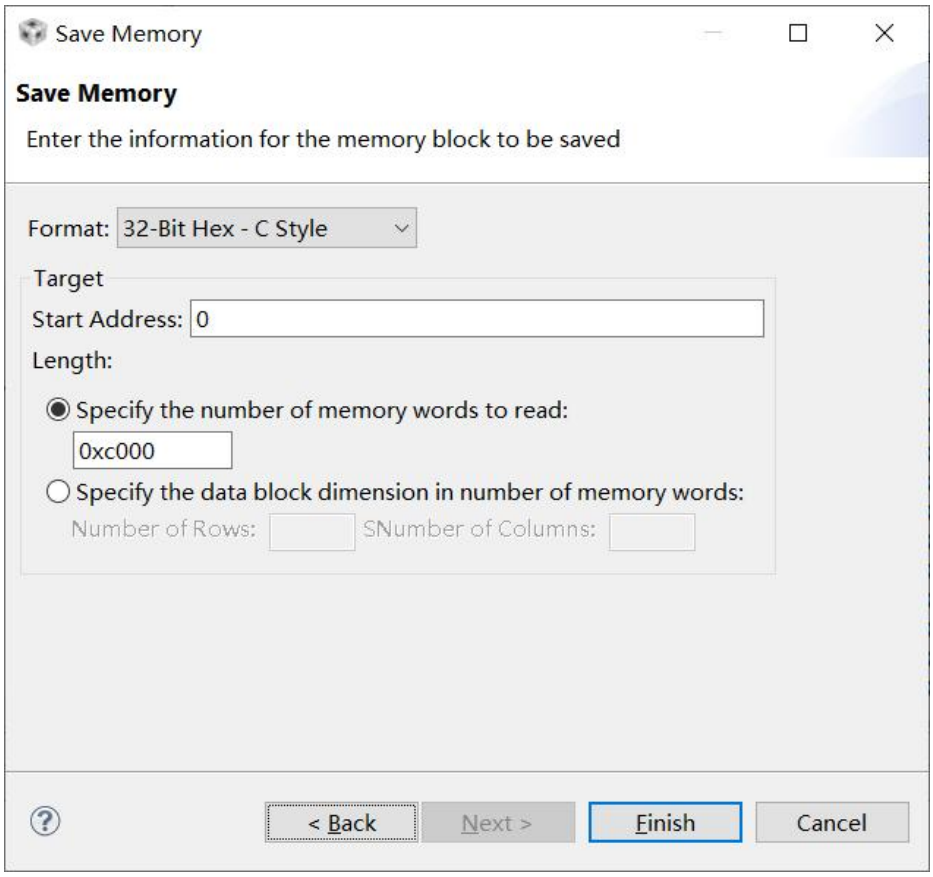


图 13 数据格式、起始地址和长度设置

（5）点击“Finish”按钮，等待CCS软件保存完成。

注意：CCS5.5 保存的数据文件需要手动修改，将第一行最末尾的“ 0”删除（0 及 0 前的空格都需要删除），才能进行烧写。

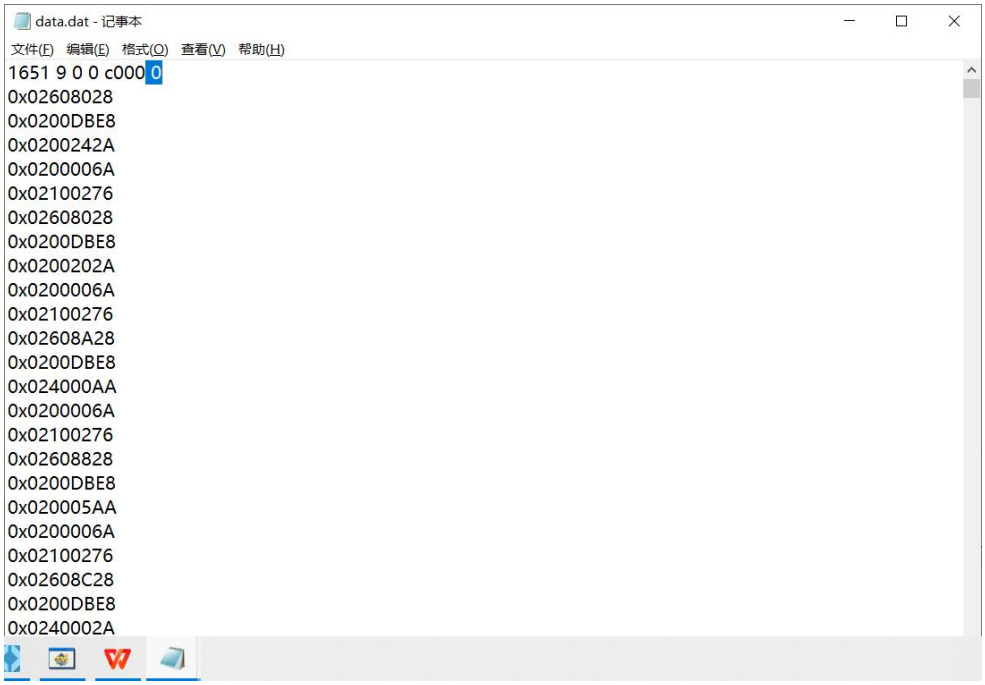


图 14 需要将 data 文件中第一行末尾“ 0”删除

7.3. 启动协议转换板进行程序烧写

（1）连接转换板串口0到开发主机串口。

（2）打开串口软件工具

注意上述步骤保存的数据文件要跟串口工具软件放在同一目录下。

点击“打开串口”按钮，连接计算机串口。成功后，“串口状态灯”会亮为红色。

（2）给协议转换板上电

（3）点击“连接DSP”按钮。如果通信正常，会有如下字样提示：

图 15 连接DSP通信正常图

（4）开始擦除板卡上的FLASH，需要等待一定时间，擦除才能完成，软件提示如下：

图 16 擦除完成软件提示图

（5）点击“打开数据文件”按钮，数据长度显示为“49152”这是文件的行数，实际的代码大小为49152*4字节。

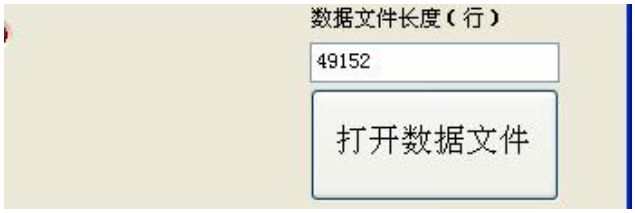


图 17 数据长度显示图

（6）点击“烧写”按钮，进行数据烧写并等待烧写完成，完成后重新上电，重启协议转换板。

图 18 串口下载工具图

8. 开发包使用说明

CLV-8073资料包括以下内容：

DOC：用户手册、软件手册、通信协议；

DSP project：DSP示例工程；

烧写工具：用于目标文件固化的工具软件；

烧写工具安装文件；

上位机端示例程序（串口端）；

8.1. DSP示例工程

配套的开发包光盘中，提供了DSP示例程序开发包。客户可以参考其中的程序，配合《软件参考手册》，方便的完成程序的开发。

配套DSP示例程序编译环境为CCS Version: 5.5.0.00077 ，打开后工程如下图所示：

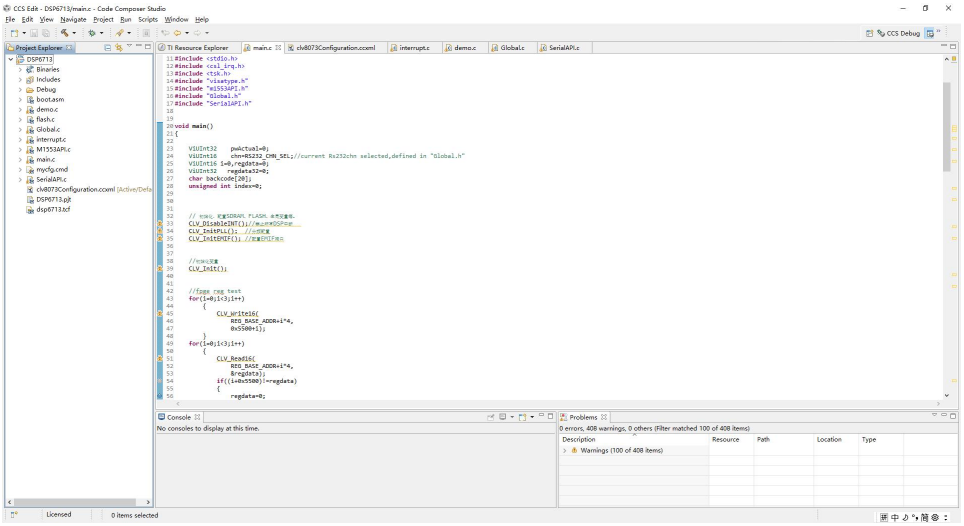


图 19 Demo 程序工程图

相关文件说明：

main.c ：主程序文件；

interrupt.c ：中断响应程序文件，用来处理1553 消息和串口消息；

flash.c ：对板载FLASH 进行擦除、读写管理函数；

SerialAPI.c ：串口API 函数，实现串口数据收发；

M1553API.c ：1553API 函数，实现1553 的BC、RT、BM功能；

Global.c ：公共函数，实现DSP 的EMIF口、锁相环、及中断配置。

demo.c ：实例程序（对1553、串口通信各个API功能封装）；

8.2. 上位机端示例程序（串口端）

上位机端示例程序基于PC机 window系统，通用串口开发。示例程序里对CLV-8073的串口通信协议进行了封装，可以通过该程序，与CLV-8073的串口进行通信，以实现启动CLV-8073的1553B功能、配置串口通信参数、下位机程序下载

等功能。

上位机端示例程序使用labwindows CVI 2017进行开发，提供工程源码。工程文件如下：

CLV_8073.c CLV_8073.h 串口协议封装；

Main.c 界面控制相关代码文件；

Main.uir 程序界面；

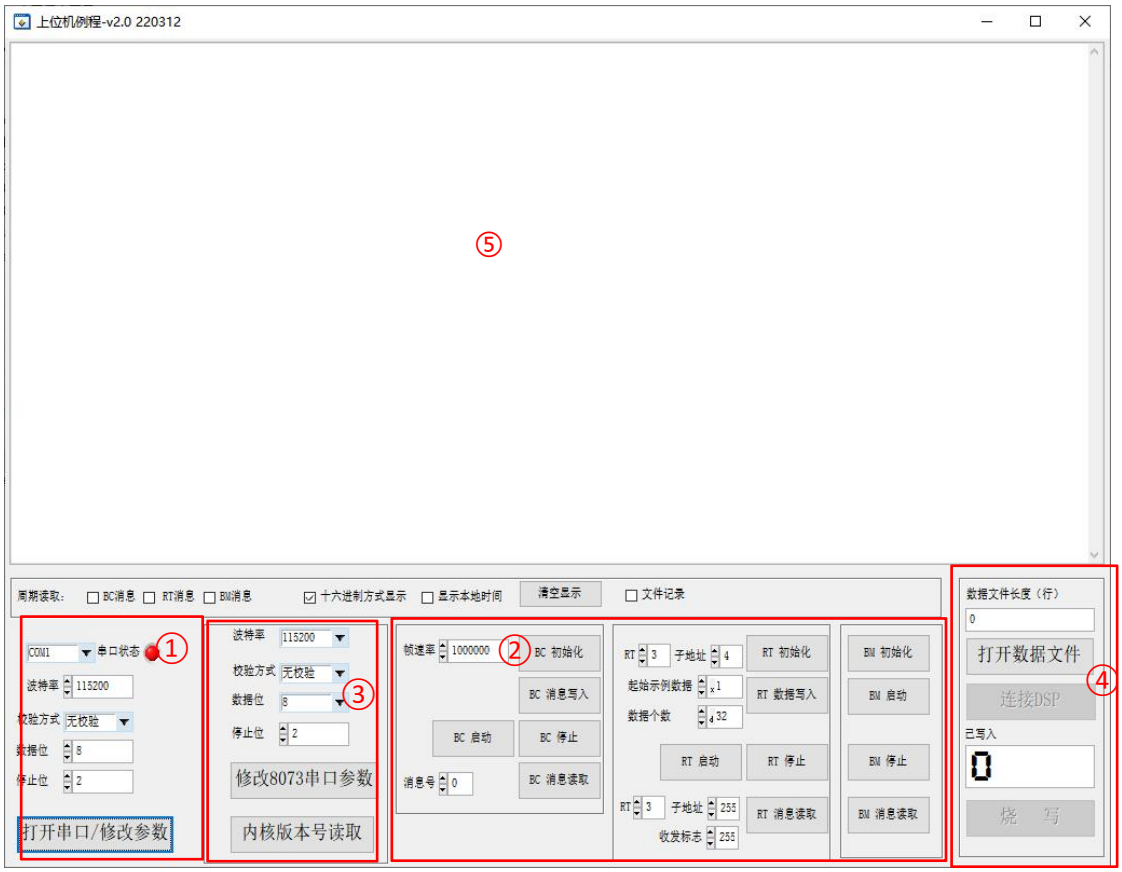


图 20 上位机端示例程序

如上图13所示，为示例程序界面。

①所标示区域为PC机端串口操作区域；

②所标示区域为1553B操作相关；

③所标示区域为CLV-8073串口参数设置及版本读取操作区域；

④所标示区域为CLV-8073程序更新操作区域；程序更新操作方法参见第6章节《DSP程序下载与更新》；

⑤所标示区域为PC机端，串口接收数据显示区。

8.2.1. 上位机端例程使用场景

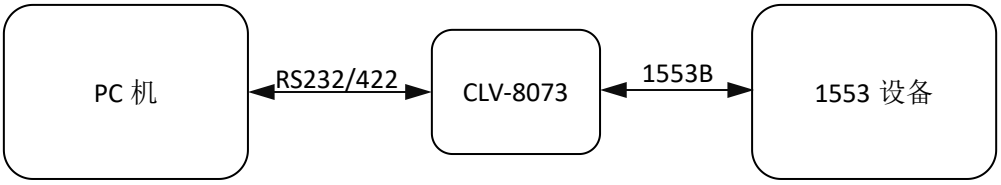


图 21 CLV-8073上位机例程使用场景

上图21展示了CLV-8073的一种使用（功能演示）场景。CLV-8073的串口通信(RS232/422)接口与PC机的串口连接（RX接TX，信号地连通）；CLV-8073的1553B通信接口与其他1553B设备对接；连接好后，即可以通过PC机端运行上位机例程，配置CLV-8073的工作模式，监测CLV-8073的状态，进行数据交互。具体接线方法，请参考本手册第2、第3章接口说明部分。

8.2.2. CLV-8073启动过程

CLV-8073上电默认串口参数如下：波特率115200bps、无校验位、停止位为1位、数据位为8 位。CLV-8073在每次重新上电后会自动恢复这些默认设置。

第一步，在接好线后，上电之前，运行上位机例程。如图20 上位机端示例程序所示界面，在①所标示区域，选择PC机端用于串口通信的串口号，并保持参数设置和CLV-8073串口参数一致，点击“打开串口/修改参数”按钮。打开串口成功，串口状态灯点亮为红色，此时PC机端已经准备就绪。

第二步，给CLV-8073上电。例程接收显示区，会显示CLV-8073上电发出的提示信息“7e7e5a5ab0”，该信息只在上电启动时发出一次。

8.2.3. 配置CLV-8073的1553B通信为BC模式

如前面所述，上位机和CLV-8073是通过串口命令帧进行交互的，上位机向CLV-8073发送指令，CLV-8073执行收到的正确指令并进行应答回复（帧定义及交互过程请参见《CLV-8073模块交互通信协议》），上位机示例程序实际上是对通信命令帧基于1553B通信功能进行了应用封装。例程上每个按钮对应一个1553配

置或操作功能。

CLV-8073提供单通道双冗余，单功能1553B通信能力。当配置为BC模式时，如果上一工作状态为RT模式或BM模式，将自动停止RT模式或BM模式，转为BC模式。如图20 上位机端示例程序所示界面，在②所标示区域，CLV-8073的BC功能配置启动流程如下：

（1）BC初始化

BC初始化时，设置BC消息的帧速率，单位为us,可设置范围：

100μs~6.5535s，点击“BC初始化”按钮，使设置生效。

（2）BC消息写入

例程软件中，点击“BC消息写入”将写入一条示例消息：RT1，SA1，BC->RT,WCOUNT=5(5个数据字)，数据：0x5500 0x5501 0x5502 0x5503 0x5504，在总线A上发送。

CLV-8073支持多条消息发送，具体请参见《CLV-8073模块交互通信协议》。

（3）BC启动/停止

点击“BC启动”按钮，启动CLV-8073的BC功能；

点击“BC停止”按钮，停止CLV-8073的BC功能；

（4）读取BC消息

填入消息号，点击“BC消息读取”按钮，读取对应的BC消息。也可以勾选周期读取框内的“BC消息”，例程将自动周期读取BC消息。

说明：1553B通信消息都是BC端发起的，CLV-8073的每条BC消息有消息编号；这个编号从0开始逐一累加，在BC消息帧构建的时候就唯一确定了。

8.2.4. 配置CLV-8073的1553B通信为RT模式

CLV-8073提供单通道双冗余，单功能1553B通信能力。当配置为RT模式时，如果上一工作状态为BC模式或BM模式，将自动停止BC模式或BM模式，转为RT模式。如图20 上位机端示例程序所示界面，在②所标示区域，RT模式配置流程如下：

（1）RT初始化

图 22 RT初始化，RT数据写入操作界面

CLV-8073支持RT地址、子地址软件配置。设置RT地址，RT子地址，点击“RT初始化”按钮，执行初始化操作。

（2）RT数据写入

设置RT地址，RT子地址，起始数据，数据个数，点击“RT数据写入”按钮，执行数据写入操作。CLV-8073对应RT,RT子地址的发送缓冲区将被填入示例数据：起始数据开始，按1累加，指定数据长度的数据序列。

（3）RT启动/停止

点击“RT启动”按钮，启动CLV-8073的RT功能；

点击“RT停止”按钮，停止CLV-8073的RT功能；

（4）RT消息读取

图 23 RT数据读取操作界面

填写RT地址，RT子地址，收发标志（0：BC->RT;1:RT->BC），点击“RT消息读取”按钮，读取对应RT消息。这些也可以勾选周期读取框内的“RT消息”，例程将自动周期读取RT消息。RT消息读取支持基于RT地址，RT子地址，收发标志的消息过滤，填255（0xFF）则表示忽略该过滤参数。

8.2.5. 配置CLV-8073的1553B通信为BM模式

CLV-8073提供单通道双冗余，单功能1553B通信能力。当配置为BM模式时，如果上一工作状态为BC模式或RT模式，将自动停止BC模式或RT模式，转为BM模式。如图20 上位机端示例程序所示界面，在②所标示区域，BM模式配置流程如下：

（1）BM初始化

点击“BM”初始化按钮，执行BM初始化操作。

（2）BM启动/停止

点击“BM启动”按钮，启动CLV-8073的BM功能；

点击“BM停止”按钮，停止CLV-8073的BM功能；

（3）BM消息读取

点击“BM消息读取”按钮，读取RT消息。也可以勾选周期读取框内的“BM消息”，例程将自动周期读取BM消息。

8.2.6. CLV-8073串口参数修改

CLV-8073支持通过串口参数配置指令，对其串口参数进行重新配置。

图 24 CLV-8073串口参数配置操作界面

CLV-8073波特率支持300~921600bps，修改CLV-8073端串口参数后，与之通信的PC端串口参数也要进行修改和CLV-8073保持一致，才能进行后续通信。CLV-8073 重新上电后，相关参数将恢复默认值，波特率 115200pbs。