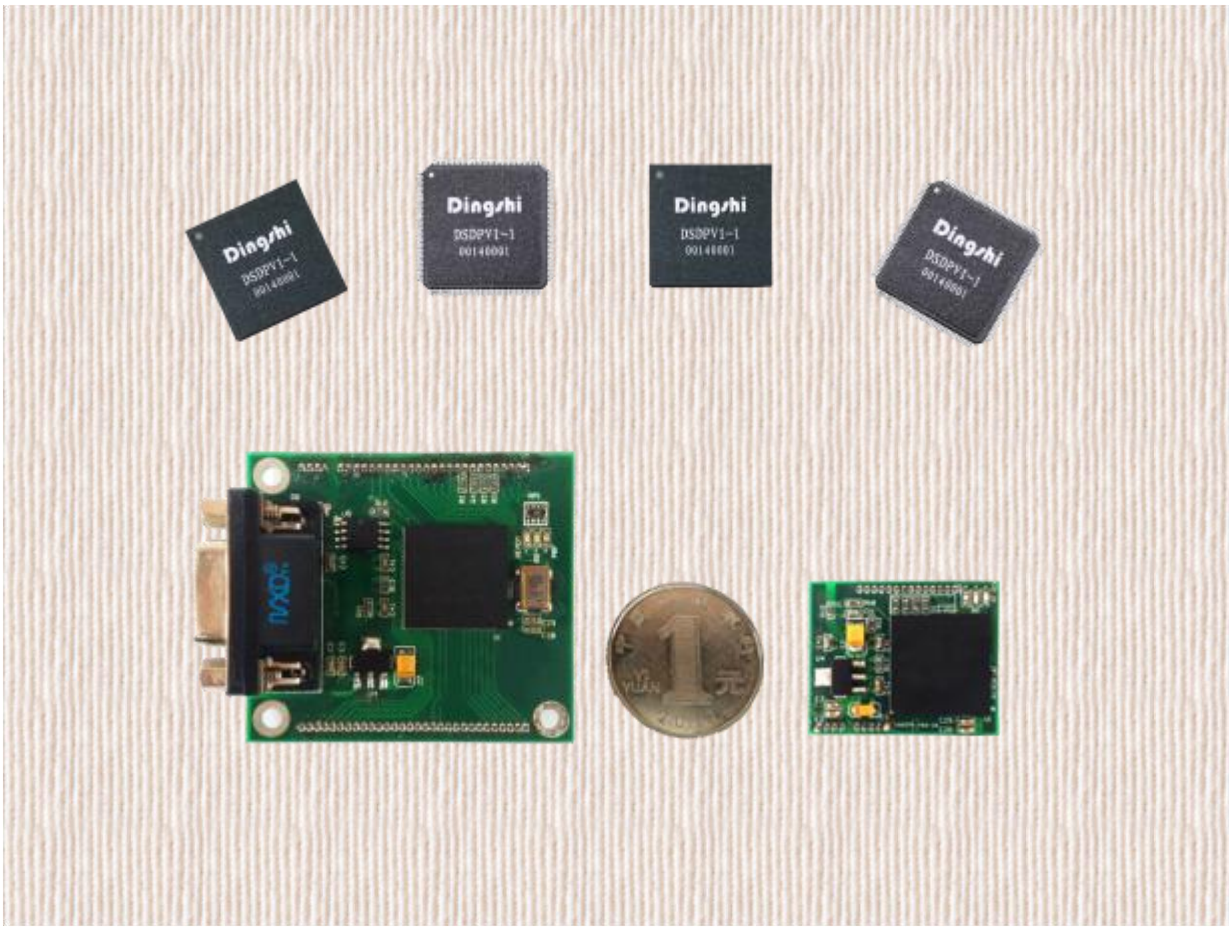
用户软件设计手册

（包含 M 系列板卡、DSDPV1-R 及 DSDPV1-RSU 芯片）

V1.1



北京鼎实创新科技股份有限公司

2016.08

目录

关于本手册

1．PROFIBUS-DP　M 系列板卡（DSDPV1 芯片系列）手册分为《用户软件设计手册》和《用户硬件设计手册》。

2．“M 系列板卡”简称“M 卡”，“DSDPV1 系列芯片”简称“芯片”。

3．《用户软件设计手册》主要描述数据结构定义、用户接口类型、通讯协议以及初始化和数据交换流程（C 源代码说明），《用户硬件设计手册》主要描述本公司产品的外观、安装尺寸，管脚定义以及评估板介绍等内容。

4．《用户软件设计手册》涉及到的 C 源代码说明分别摘自“FSMC_SRAM.C”、“FSMC_SRAM.h”“SPI.C”、“DSUART.C”，“DSUART.h”用户可从本公司赠送的光盘中找到对应文档具体了解详细内容。

5．本手册包含的产品种类及型号分别是：M 系列嵌入式总线板卡（型号：M1-PB-V20、M2-PB-SPI、M2-PB-UART）及 DSDPV1 系列芯片（型号：DSDPV1-R、DSDPV1-RSU）。

6．获取手册方式：随本公司产品赠送光盘（包含本公司所有产品的手册），客户也可以从本公司网站直接下载。

http://www.c-profibus.com.cn

第一章 产品概述

一、主要用途

鼎实公司推出了专为自主开发具有 PROFIBUS-DPV0/V1 通信功能的 M 系列板卡及 DSDPV1 系列芯片，用户可以快速完成 PROFIBUS-DP 从站通信接口的开发，解决了采用国外芯片的开发方案要求了解 PRofibus 协议技术细节、专业性强、增加 Profibus 接口开发难度和工作量的的问题。

1．产品特点

鼎实自主开发的 PROFIBUS-DSDPV1 系列及 M 系列板卡，基于鼎实十几年 OEM 板卡用户接口经验，接口编程极其简单、绝不需要固件。在功能上涵盖了全部 V1(C1+C2)协议。 采用鼎实 DSDPV1 系列芯片或 M 系列板卡既有接口成本低、结构尺寸灵活的优点，又有开发简单、不需要固件，全套 V1 功能及提供 DTM/EDD 软件支持的优点。

2．硬件实现全部 DP/V0/V1(C1+C2)功能不需要固件

芯片内置 DP/V0/V1(C1+C2)协议栈完全由硬件实现功能，用户开发不需要固件。用户接口支持 3 种模式：双口 RAM、SPI、UART。只需要凭借双口 RAM 数据格式定义，通过数据存取就可以实现 PROFIBUS-DP/V0/V1(C1+C2)功能。不涉及 PROFIBUS 技术细节，与鼎实 OEM1-2 系列板卡协议类似，简单易学、可以快速掌握。

3．技术性能优异

自适应波特率 9.6K~12M、DP/V0 全部功能（包括：同步冻结、故障模式、外部诊断、全局控制）、V1(C1+C2)全部功能（读写、诊断报警）；最大输入输出各 244 字节、最大用户参数 237 字节、最大诊断数据 238 字节；内部 2K 字节 DPRAM。

4．提供 DTM/EDD 解决方案

与鼎实 D 系列接口板卡相同，鼎实为基于 DSDPV1 系列芯片及 M 系列板卡的产品开发提供 ETM/EDD 解决方案。

5．提供 PROFIBUS 冗余从站的解决方案

鼎实为基于 DSDPV1 系列芯片及 M 卡的用户产品开发提供 PROFIBUS 冗余从站解决方案。

6．开发包提供完整调试诊断平台

开发包提供完整的开发调试平台；包括评估板、样片、PROFIBUS 电缆及插头等，可以组成典型的主站+2 从站的 PROFIBUS-DP 调试平台。特别是开发包调试工具 PBSTUDIO，系鼎实自主开发的 PROFIBUS 诊断工具，可完成在线站点状态监控与统计、报文记录及辅助分析、模拟主站功能、自主认证预测试、网络波形质量分析功能。为全面、快速排查产品通信故障提供强大工具。

7．保证通过产品标准认证测试

鼎实以多年雄厚 PROFIBUS 产品标准测试认证技术为基础，完成开发芯片开发测试，评估板开发测试，并提供用户开发例程、认证测试例程；保证用户开发产品通过国际 PROFIBUS 测试实验室认证测试。

8．技术培训优势

（1）开发相关技术培训，包括：PROFIBUS 基础、DSDPV1 芯片应用、开发包应用、故障诊断排除等。

（2）认证测试相关技术培训：自主认证测试技术、测试工具使用

（3）DTM/EDD 开发技术

（4）产品行规技术标准

9．技术服务优势

自主认证测试技术、测试工具使用

（1）立项阶段方案选择技术咨询

（2）开发阶段技术方案及技术交流与培训

（3）认证测试阶段的指导与服务

（4）制造阶段的工装设计与质量保证

（5）用户技术服务人员的技术培训

（6）用户产品市场拓展中的技术支持

（7）用户产品型号种类扩展需要的技术支持

（8）技术性能提高需要的技术升级

二、技术指标

- PROFIBUS 电气接口：标准 PROFIBUS-DP 驱动接口，波特率自适应：9600～12M；
- PROFIBUS 接口通信协议符合：GB/T 20540-2006: 测量和控制数字数据通信工业控制系统用现场总线第 3 部分：PROFIBUS 规范, 支持 PROFIBUS-DP/V0、V1-MSAC1/V1-MSAC2 协议.
- M 卡与用户 MCU 接口模式：
  - M 卡-DPRAM 模式：双口 RAM，2K Bytes
  - M 卡-SPI 模式： SPI(时钟为 9MHz)。

    4 线模式：CS、SCK、MISO、MOSI
  - M 卡-UART 模式： UART，通讯过程中采用主从方式

    (1)支持 9.6K、19.2K、38.4K、57.6K、115.2K、230.4K、460.8K 和 1.8432M 共 8 种波特率。

    (2)通讯报文中的 1 个字符长度为 11 个 Bit，含 1 个起始位、8 个数据位、1 个偶校验位、1 个停止位；
- M 卡与用户 MCU 通信使用 CRC 校验，保证数据安全性

◆ M 卡与用户 MCU 通信数据数量可根据 PROFIBUS 数据量自定义：

- Profibus 输入输出数据数量：最大为 244 字节输入 ， 244 字节输出
- Profibus 用户诊断数据数量：最大诊断数据长度为 238 个字节；
- Profibus 用户参数数据数量：最大参数数据长度为 237 个字节；
- Profibus 用户配置数据数量：最大配置数据长度为 200 个字节；
- Profibus DPV1 槽-索引个数：最大为 12 个；
- Profibus DPV1 槽-索引数据长度：最大为 240 个字节；
- Profibus DPV1/MSAC2 支持的连接通道个数：最大为 2 个；
- 支持 IM 设备维护功能(slot：0 ,index：255 ， subindex：65000)

◆ M 卡板与 M 卡之间硬件握手方式进行数据交换，保证了数据交换的实时性和完整性（一致性）；

◆ DPV1 C1/C2 读写功能简述：

此 M 卡支持 Profibus DPV1 的 C1/C2 读写功能，及 IM 功能。其中槽(Slot)、索引(Index)等相关参数都可通过初始化设置。

◆ M 卡须供电两组隔离电源：

Ⅰ组：DC 5V±5% 5V/0V-40mA

Ⅱ组：DC3.3V±5%

M1 卡：VCC/GND 常温（27℃）：3.3V/115mA，高温（55℃）：3.3V/140mA

M2 卡：VCC/GND 常温（27℃）：3.3V/160mA，高温（55℃）：3.3V/200mA

◆ DSDPV1-RSU 芯片须供电两组电源：

Ⅰ组：DC 3.3V±5%

Ⅱ组：DC1.2V±5%

功耗（max）：0.315W

◆ 建议用户提供 DC5V/0V 电源（隔离外电源），并将此电源引到 PROFIBUS 接口插座处。

◆ 1.2 V 电源芯片(NCP1512B)要离 FPGA 芯片一定距离，建议大于 20mm，因为电源芯片的发热会引起 FPGA 功耗增加。

◆ 环境温度：

运输与存储：-40℃～70℃

工作温度：-20℃～55℃

工作相对湿度：5～90％

- M-PB-V20 外形尺寸：(长)×(宽)=54mm×48mm
- M2 外形尺寸：(长)×(宽)=33mm×27mm
- DSDPV1 系列芯片封装详见《用户硬件设计手册》

第二章 DPRAM 数据结构

一、概述

本章主要用来描述北京鼎实创新科技有限公司 M 系列板卡及 DSDPV1 系列芯片(型号:M1-PB-V20、M2-PB-UART、M2-PB-SPI)和 DSDPV1 系列芯片(型号:DSDPV1-R、DSDPV1-RSU) 产品内置 DPRAM 的数据结构。

二、DSDPV1 系列芯片及 M 系列板卡内置 DPRAM 数据结构定义

PROFIBUS-DP 从站协议芯片(DSDPV1)及 M 系列板卡内部集成了 2Kbytes Dual-port-RAM，DPRAM 地址范围为 0x000~0x7FF。其定义如下表所示。

表 1：DPRAM 地址定义 0x000~0x7FF

字节数	地址	内容	注释	接口板	用户板	
4 个字节	0000	版本代码	（保留）	RW	R	
	0001	版本年份		RW	R	
	0002	版本月份		RW	R	
	0003	版本日期		RW	R	
1 个字节	0004	接口板有效标志	0xC5：标志接口板已完成自身硬件初始化 ！0xC5：标志接口板还未完成自身硬件初始化	RW	R	
1 个字节	0005	V1 功能开启标志	0x1D：开启接口板 V1 功能 ！0x1D：未开启接口板 V1 功能	R	RW	

2 个字节	0006 0007(reserved)	双边 DPRAM 访问计数	见注 1 数据类型是：int（带符号字-32768 ~ +32767）	RW	RW	
2 个字节	0008 0009	DP 通信掉线次数计数	见注 2 数据类型为：unsigned int（无符号字 0 ~ 65535）	RW	R	
2 个字节	000A	接口板状态字节 1	见表 2	RW	R	
	000B	接口板状态字节 2				
2 个字节	000C	用户板状态字节 1	见表 3	R	RW	
	000D	用户板状态字节 2				
2 个字节	000E	接口板命令字节 1	见表 4	RW	RW	
	000F	接口板命令字节 2				
2 个字节	0010	用户板命令字节 1	见表 5	RW	RW	
	0011	用户板命令字节 2				
1 个字节	0012	接口板状态字节 3	见表 7	RW	R	
1 个字节	0013	接口板状态字节 4	见表 8			
12 个字节	0014	SPI 状态字(SPI 控制权申请)	见表 10			
	0015		SPI U_timer 使能 （暂时未开放）			
	0016		保留			
	0017		保留			
	0018		UART 控制权申请			
	0019		UART 字符间隔变更命令			
	001A		UART TSDR			
	…					
	…					
	001F					
215 个字节	0020	V0 初始化数据	V0 初始化数据长度字节	R	RW	
	0021		从站站地址			

	0022		ID 号高字节			
	0023		ID 号低字节			
	0024		SPC3_Register_status0 (Fail_safe ， Syn- Freeze，Slave_address_change 见表 9			
	0025		预留备用(SPC3_Register_status1)			
	0026		预留备用(SPC3_Register_status2)			
	0027		PROFIBUS 输入数据最大可能长度 a（≤244）			
	0028		PROFIBUS 输出数据最大可能长度 b（≤244）			
	0029		用户诊断数据长度 i（≤238 不包含标准 6 字节诊断数据） i=0：无用户诊断数据（仅标准 6 字节诊断数据）			
	002A		用户参数数据最大可能长度 j（≤237 不包含标准 7 字节参数数据） j=0：无用户参数数据（仅标准 7 字节用户参数数据）			
	002B		用户参数正确与否判断方式（见表 6）			
	002C		配置数据的最大可能长度 m（≤200）			
	002D		配置数据正确与否判断方式（见表 6）			
	002E		CFG 数据长度 = n（≤m）			
	002F		CFG 数据 1 见表 11			
	…		…			
	…		CFG 数据 n 见表 11			
	…		…			
	00F6		预留备用			
2 个字节	00F7	V0 初始化数据校验和	V0 初始化 CRC 校验和高字节	R	RW	
	00F8		V0 初始化 CRC 校验和低字节			
55 个字节	00F9	保留空置	预留备用			
	…		…			
	012F		预留备用			
245 个字	0130	PROFIBUS 输入数据	当前 PROFIBUS 输入数据长度	R	RW	

节	0131		PROFIBUS 输入数据字节 1			
	0132		…			
	0133		PROFIBUS 输入数据字节 a			
	…		…			
	0224		预留备用			
2 个字节	0225	PROFIBUS 输入数据校验和	V1 初始化 CRC 校验和高字节	R	RW	
	0226		V1 初始化 CRC 校验和低字节			
240 个字节	0227	诊断数据	诊断数据高低优先权属性 *0：低优先权扩展诊断* *1：高优先权外部诊断* *2：高优先权静态诊断* *//==========2013-1-17 娟子改======//* 诊断数据高低优先权属性 0x00：低优先权扩展诊断 0x01：高优先权外部诊断，从站无错误 0x81：高优先权外部诊断，从站有错误 0x02：高优先权静态诊断，从站无错误 0x82：高优先权静态诊断，从站有错误			
	0228		当前用户诊断数据长度			
	0229		诊断数据字节 1			
			…			
	…		诊断数据字节 i			
	…		…			
	0316		预留备用			
2 个字节	0317	诊断数据校验和	诊断数据 CRC 校验和高字节			
	0318		诊断数据 CRC 校验和低字节			
245 个字节	0319	PROFIBUS 输出数据	当前 PROFIBUS 输出数据长度	R	RW	
	031A		PROFIBUS 输出数据字节 1			

	…		…			
	…		PROFIBUS 输出数据字节 a			
	…		…			
	040D		预留备用			
2 个字节	040E	PROFIBUS 输出数据校验和	输出数据 CRC 校验和高字节	R	RW	
	040F		输出数据 CRC 校验和低字节			
238 个字节	0410	PRM 参数化数据	当前用户参数数据长度	R	RW	
	0411		用户参数字节 1			
	…		…			
	…		用户参数字节 j			
	…		…			
	04FD		预留备用			
2 个字节	04FE	PRM 参数化数据校验和	用户参数数据 CRC 校验和高字节	R	RW	
	04FF		用户参数数据 CRC 校验和低字节			
203 个字节	0500	CFG 配置数据	当前配置输出数据长度	RW	R	
	0501		当前配置输入数据长度			
	…		当前配置数据长度			
	…		配置数据字节 1			
			…			
	…		配置数据字节 a			
	05CA		预留备用			
2 个字节	05CB	CFG 配置数据校验和	配置数据 CRC 校验和高字节	RW	R	
	05CC		配置数据 CRC 校验和低字节			
3 个字节	05CD	保留空置				
	05CE					
	05CF					
244 个字	05D0	V1/C1 非循环数据	主站寻址槽-索引读写 Function_Num	RW	RW	

节	05D1		寻址槽号 或者 Error_Decode			
	05D2		寻址索引号 或者 Error_Decode1			
	05D3		本槽-索引数据长度 z（≤240）或者 Error_Decode2			
	05D4		本槽-索引数据字节 1			
	…		…			
	…		本槽-索引数据字节 z			
	…		…			
	06C3		预留备用			
2 个字节	06C4	V1/C1 非循环通信数据校验和	V1/C1 非循环通信数据 CRC 校验和高字节	RW	RW	
	06C5		V1/C1 非循环通信数据 CRC 校验和低字节			
10 个字节	06C6	保留空置				
	…					
	06CF					
244 个字节	06D0	V1/C2 非循环数据	主站寻址槽-索引读写 Function_Num			
	06D1		寻址槽号 或者 Error_Decode			
	06D2		寻址索引号 或者 Error_Decode1			
	06D3		本槽-索引数据长度 z（≤240）或者 Error_Decode2			
	06D4		本槽-索引数据字节 1			
	…		…			
			本槽-索引数据字节 z			
			…			
	7C3		预留备用			

2 个字节	7C4	V1/C2 非循环通信数据校验和	V1/C2 非循环通信数据 CRC 校验和高字节			
	7C5		V1/C2 非循环通信数据 CRC 校验和高字节			
59 个字节	07C6	保留空置				
	…					
	0800					

1. 表 1 注解：

注 1：双边 DPRAM 访问计数 W0006

2 BYTES,INT 带符号整型（-32768-+32767）。该字在接口板完成自身硬件初始化后，由接口板将此字置值+100。当该计数字的值保持+100 不变时，接口板不对这个字进行操作；一旦该计数字的值开始变化，接口板每次访问到 DPRAM 后，将该字递+1。用户板每次访问到 DPRAM 后，将此字递-1。

注 2：DP 通信掉线次数计数 W0008

此字用于监测接口板在 DP 通信中脱离数据交换状态的次数。初始值为 0x0000，每次接口板退出数据交换状态，此计数值递增；接口板恢复数据交换状态，不对此计数值有任何影响。

2. 表 2：接口板状态字节 1

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
									接口板：RW 用户板：RW
000A	接口板状态字节 1								**接口板 DP 信息初始化状态：** 0：接口板 DP 初始化未完成 1：接口板 DP 初始化已完成

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
		保留		接口板：RW 用户板：R	接口板：RW 用户板：R	接口板：RW 用户板：R	接口板：RW 用户板：R	接口板：RW 用户板：R	接口板：RW 用户板：R
000B	接口板状态字节 2			用户板 C2 非循环数据 CRC 错： 0：正确 1：错误	用户板 C1 非循环数据 CRC 错： 0：正确 1：错误	诊断数据 CRC 错： 0：正确 1：错误	输入数据 CRC 错： 0：正确 1：错误	V1 初始化数据 CRC 错： 0：正确 1：错误	V0 初始化数据 CRC 错： 0：正确 1：错误

3. 表 3：用户板状态字节

地址	名称	7	6	5	4	3	2	Bit1	Bit0
								接口板：R 用户板：RW	接口板：R 用户板：RW
000C	用户板状态字节 1	保留	保留	保留	保留	保留	保留	判断配置数据结果 0：用户判断配置数据正确 1：用户判断配置数据错误 注：本位仅当 B0029=2 且 B0010.6=1 时有效。 B0029：配置数据正确与否判断方式 B0010.6：用户板命令字节 1_配置数据判断结果回传标志。	判断用户参数数据结果 0：用户判断用户参数数据正确 1：用户判断用户参数数据错误 注：本位仅当 B0027=2 且 B0010.7=1 时有效。 B0027：用户参数正确与否判断方式 B0010.7：用户板命令字节 1_用户参数判断结果回传标志。

地址	内容	7	6	5	Bit4	Bit3	Bit2	Bit1	Bit0
					接口板：R	接口板：R	接口板：R	接口板：R	接口板：R

					用户板：RW	用户板：RW	用户板：RW	用户板：RW	用户板：RW
000D	用户板状态字节 2	保留	保留	保留	从接口板获得 C1 的输出数据 CRC： 0：正确 1：错	从接口板获得 C2 的输出数据 CRC： 0：正确 1：错	从接口板获得的非循环数 CRC 错： 0：正确 1：错	从接口板获得的用户参数数据 CRC 错： 0：正确 1：错	从接口板获得的配置数据 CRC 错： 0：正确 1：错

4．表 4：接口板命令字节 1

地址	内容	7	6	5	4	3	Bit2	Bit1	Bit0
							接口板：W1 用户板：R/WO	接口板：W1 用户板：R/WO	接口板：W1 用户板：R/WO
000E	接口板命令字节 1	保留	保留	保留	保留	保留	配置数据有效标志： 接口板 W1：接口板将新配置数据放入 DPRAM 后，将此位置 1。 用户板 R1：用户板读到此位为 1 将取出 DPRAM 中的新配置数据。然后将此位清 0。. 用户板 WO：用户板读取出 DPRAM 中的新配置数据后将此位清 0。 用户板 RO：DPRAM 中没有新配置数据。	用户参数数据有效标志： 接口板 W1：接口板将新用户参数数据放入 DPRAM 后，将此位置 1。 用户板 R1：用户板读到此位为 1 将取出 DPRAM 中的新用户参数数据。然后将此位清 0。 用户板 WO：用户板读取出 DPRAM 中的新用户参数数据后将此位清 0。 用户板 RO：DPRAM 中没有新用户参数数据。	输出数据有效标志： 接口板 W1：接口板将新输出数据放入 DPRAM 后，将此位置 1。 用户板 R1：用户板读到此位为 1 将取出 DPRAM 中的新输出数据。然后将此位清 0。. 用户板 WO：用户板读取出 DPRAM 中的新输出数据后将此位清 0。 用户板 RO：DPRAM 中没有新输出数据。

注释：

- **输出数据有效标志 B000E.0**：接口板将输出数据放入 DPRAM 后，将此位置 1，用于通知用户板获取输出数据。当用户板取走输出数据后，此位被用户板清 0。
- **用户参数数据有效标志 B000E.1**：接口板将主站发送的用户参数数据放入 DPRAM 后，将此位置 1，用于通知用户板获取用户参数数据。当用户板取走用户参数数据后，此位被用户板清 0。
- **配置数据有效标志 B000E.2**：接口板将主站发送的配置数据放入 DPRAM 后，将此位置 1，用于通知用户板获取配置数据。当用户板取走配置数据后，此位被

用户板清 0。

地址	内容	7	6	5	4	3	2	Bit1	Bit0
地址								接口板：W1 用户板：R/WO	接口板：W1 用户板：R/WO
000F	接口板命令字节 2	保留	保留	保留	保留	保留	保留	接口板 C2 非循环数据标志： 接口板 W1：通知用户板有新 V1/C2 请求数据在 DPRAM 中。 用户板 WO：用户板在取走非循环数据后清 0。	接口板 C1 非循环数据标志： 接口板 W1：通知用户板有新 V1/C1 请求数据在 DPRAM 中。 用户板 WO：用户板在取走非循环数据后清 0。

注释：

◆ **非循环数据有效标志 B000F.0**：接口板接收到一个来自主站的 DPV1 服务请求后，先判断 B0011.0（用户板命令字节 2_非循环数据有效标志）。

如果 B0011.0=0，则把 DPV1 服务请求放入 DPRAM 中，并将此位 B000F.0 置 1。此位 B000F.0 由用户板在取走非循环数据后清 0。

如果 B0011.0=1，是错误：A、接口板上次从 DPRAM 中取走 V1 响应后没有清 B0011.0，这是程序 BUG；或 B、主站在没有得到上次 V1 响应情况下又发另一个 V1 请求。应忽略此次 V1 请求，先取出用户板放在 DPRAM 中的 V1 响应，且对 B0011.0 清 0。

◆ **IM 数据有效标志 B000F.1**：接口板接收到主站写入的 IM 信息后，先判断 B0011.1（用户板命令字节 2 中的 IM 数据有效标志 bit1）。

如果 B0011.1=0，则把接口板接收到的 IM 信息放入 DPRAM 中，并将此位 B000F.1 置 1，此位 B000F.1 由用户板在取走 IM 数据后清 0。

如果 B0011.1=1，是错误：A、接口板上次从 DPRAM 中取走 IM 信息后没有清 B0011.0，这是程序 BUG；或 B、主站在没有得到上次 IM 信息响应情况下又发另一个 IM 信息请求。应忽略此次 IM 信息请求，先去出用户板放在 DPRAM 中的 IM 信息响应，且对 B0011.1 清 0。

5. 表 5：用户板命令字节

地址	内容	Bit7	Bit6	5	4	3	Bit2	Bit1	Bit0
地址	内容	接口板：R/WO 用户板：W1	接口板：R/WO 用户板：W1				接口板：R/WO 用户板：W1	接口板：R/WO 用户板：W1	接口板：R/WO 用户板：W1

0010	用户板命令字节 1	用户参数判断结果回传标志： 用户板 W1： 用户板已将用户参数判断结果存入 B000C.0。 接口板 R1：B000C.0 中用户参数判断结果有效。 接口板 WO：接口板从 B000C.0 中取走用户参数判断结果后将该标志清 0。 接口板 RO：B000C.0 中用户参数判断结果无效。	配置数据判断结果回传标志： 用户板 W1： 用户板已将配置数据判断结果存入 B000C.1。 接口板 R1：B000C.1 中配置数据判断结果有效。 接口板 WO：接口板从 B000C.1 中取走配置数据判断结果后将该标志清 0。 接口板 RO：B000C.1 中配置数据判断结果无效。	保留	保留	保留	初始化信息有效： 用户板 W1： 用户板已更新初始化数据。 接口板 R1： DPRAM 中有新初始化数据。 接口板 WO：接口板取走新初始化数据后该标志清 0。 接口板 RO：没有新初始化数据。	诊断数据有效标志： 用户板 W1： 用户板已更新 DPRAM 中诊断数据。 接口板 R1：DPRAM 中有新诊断数据。 接口板 WO：接口板取走新诊断数据后该标志清 0。 接口板 RO：没有新诊断数据。	输入数据有效标志： 用户板 W1： 用户板已更新 DPRAM 中输入数据 接口板 R1：DPRAM 中有新输入数据。 接口板 WO：接口板取走新输入数据后该标志清 0。 接口板 RO：没有新输入数据。

注释：

- **输入数据有效标志 B0010.0**：用户板将输入数据放入 DPRAM 后，将此位置 1，用于通知接口板获取输入数据。当接口板取走输入数据后，此位 **B0010.0** 被接口板清 0。
- **诊断数据有效标志 B0010.1**：用户板将诊断数据放入 DPRAM 后，将此位置 1，用于通知接口板获取诊断数据。当接口板取走诊断数据后，此位 **B0010.1** 被接口板清 0。
- **初始化信息有效 B0010.2**：用户板将 DP 初始化信息数据放入 DPRAM 后，将此位置 1，用于通知接口板获取 DP 初始化信息。当接口板取走 DP 初始化信息数据后，此位 **B0010.2** 被接口板清 0。
- **用户参数判断结果回传标志 B0010.6**：用户板判断用户参数数据是否正确后，将判断结果放入用户板状态字节中，并将此位置 1。当接口板取走用户参数判断结果后，此位被接口板清 0。
- **配置数据判断结果回传标志 B0010.7**：用户板判断配置数据是否正确后，将判断结果放入用户板状态字节中，并将此位置 1。当接口板取走配置数据判断结果后，此位被接口板清 0。

地址	内容	Bit7	Bit6	5	4	3	2	Bit1	Bit0
地址	内容	接口板：R/WO 用户板：W1	接口板：R 用户板：RW1/0					接口板：R/WO 用户板：W1	接口板：R/WO 用户板：W1
0011	用户板命令字节 2	清接口板状态字输入数据 CRC 错误标记： 用户板 W1：令接口板清一次 000B.2 接口板 R1：接口板清一次 000B.2. 接口板 WO：接口板清一次 000B.2 后将此位写 0	输入数据 CRC 错误标志位”历史/现时”选择 W1：令接口板 B000B.2 保持 CRC 历史错误 WO：令接口板对 000B.2 显示 CRC 现时错误。	保留	保留	保留	保留	用户板 C2 非循环数据有效标志： 用户板 W1：用户板已经将 DPV1/C2 服务响应放入 DPRAM 中，等待接口板获得此响应 接口板 WO：接收 DPRAM 中 DPV1 服务响应后清 0。 接口板 RO：用户板还没有准备好 DPV1 服务响应数据。 接口板 R1：用户板已经将 DPV1 服务响应数据放入 DPRAM 中，等待接口板获得此响应。	用户板 C1 非循环数据有效标志： 用户板 W1：用户板已经将 DPV1/C1 服务响应放入 DPRAM 中，等待接口板获得此响应 接口板 WO：接收 DPRAM 中 DPV1 服务响应后清 0。 接口板 RO：用户板还没有准备好 DPV1 服务响应数据。 接口板 R1：用户板已经将 DPV1 服务响应数据放入 DPRAM 中，等待接口板获得此响应。

6. 表 6：用户参数/配置数据正确与否判断方式

地址	内容	取值	含义
002B	用户参数正确与否判断方式	Default: 0(判长度)	仅由接口板判断从主站接收的用户参数是否正确，判断条件为实际用户参数数据长度≤V0 初始化信息中的用户参数数据最大可能长度（地址 002B）PRM_LEN 属于[0,v0_LEN]
		1(判长度)	仅由接口板判断从主站接收的用户参数是否正确，判断条件为实际用户参数数据长度=V0

			初始化信息中的用户参数数据最大可能长度（地址 002B）
		2	接口板将接收的用户参数通过 DPRAM 发送给用户板，由用户判断是否正确。判断结果由“用户板命令字节 1_用户参数判断结果回传标志”（B0010.7）通知接口板获取结果。由”用户板状态字节 1_判断用户参数数据结果”（B000C.0）报告给接口板。 注意：选择此方式时，用户板必须保证尽快通知接口板判断结果。如果判断延时时间过长，则①产品标准测试可能不能通过；②实际应用中可能存在主站连通时间偏长，甚至连通困难的现象。
002D	配置数据正确与否判断方式	Default: 0(判长度)	仅由接口板判断从主站接收的配置数据是否正确，判断条件为实际配置数据长度≤V0 初始化信息中的配置数据的最大可能长度（地址 002D）CFG_LEN 属于(0,v0_LEN]
		1(判数据)	仅由接口板判断从主站接收的配置数据是否正确，判断条件为实际配置数据=V0 初始化信息中的配置数据（地址 002F~00F6）
		2	接口板将接收的配置数据通过 DPRAM 发送给用户板，由用户判断是否正确。判断结果由”用户板命令字节 1_配置数据判断结果回传标志”（B0010.6）通知接口板获取结果。由“用户板状态字节 1_判断配置数据结果”（B000C.1）报告给接口板。 注意：选择此方式时，用户板必须保证尽快通知接口板判断结果。如果判断延时时间过长，则①产品标准测试可能不能通过；②实际应用中可能存在主站连通时间偏长，甚至连通困难的现象

7．表 7 接口板状态字节 3

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			接口板：RW 用户板：R		接口板：RW 用户板：R				
0012	接口板状态字节 3		当前 DP 通信波特率：		从站 DP 通信当前状态：				

		0000 = 12 MBaud 0001 = 6 MBaud 0010 = 3 MBaud 0011 = 1.5 MBaud 0100 = 500 kBaud 0101 = 187.5 kBaud 0110 = 93.75 kBaud 0111 = 45.45 kBaud 1000 = 19.2 kBaud 1001 = 9.6 kBaud 1111: 波特率获取中 其它: 非有效值	00: OFFLINE 01: WAIT_PRM 10: WAIT_CFG 11: DATA_EXCHANGE		

8．表 8：接口板状态字节 4

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
							初始化信息报错：		
0013							接口板：RW 用户板：R	接口板：RW 用户板：R	接口板：RW 用户板：R
	接口板状态字节 4	保 留	保 留	保 留			槽-索引个数超限： 0：正确 1：错误	C2 通道个数超限： 0：正确 1：错误	各种数据长度超限： 0：正确 1：错误

9．表 9：SPC3_register_status

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	SPC3_register_status					接口板：R 用户板：RW	接口板：R 用户板：RW	接口板：R 用户板：RW	接口板：R 用户板：RW
0024	SPC3 寄存器预置					Fail_Safe： 1=支持 0=不支持	SYNC： 1=支持 0=不支持	FREEZE： 1=支持 0=不支持	Slave_addr_change： 1=支持 0=不支持

注释：

- 非循环数据有效标志 B0011.0：用户板在准备好 DPV1 服务响应放入 DPRAM 后，将此位置 1。此位由接口板在取走非循环服务响应数据后清 0。
- IM 数据有效标志 B0011.1：用户板在准备好 IM 信息放入 DPRAM 后，并将此位置 1。此位由接口板在取走 IM 数据后清 0。
- CRC 错误消除 B0011.7：此位仅对用户板提供的输入数据有效。

用户板 W1：令接口板清一次 000B.2

接口板 R1：接口板检测该位由 0 变 1 时清一次 000B.2.

当接口板对此位在 CRC 错误标志位含义选择位置 1 时有效，当此位被置 1，状态字节中的 CRC 错误标志被清 0。

- **输入数据 CRC 错误标志位”历史/现时”选择 B0011.6**：此位仅对“接口板状态字节 2_输入数据 CRC 错”（B000B.2）有效，“接口板状态字节 2_输入数据 CRC 错”（B000B.2）可以有两个含义：当 **B0011.6**=0 时，B000B.2 的含义是：当数据接收方发现 CRC 有误后，将此位置 1，并在得到新的 CRC 正确的数据后，将此位清 0。当 B000B.2=1 时，CRC 错误标志的含义是：当数据接收方发现 CRC 有误后，将此位置 1，在得到新的 CRC 正确的数据后，此位并不清 0，而是作为历史记录保留下来，直到接口板读到用户板命令字节 2 中 **CRC 错误消除 B0011.7** 为 1，B000B.2 被清 0 一次。

10. 表 10：SPI 控制权申请

空闲状态字(0x014)定义：

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
保留	保留	保留	保留	保留	保留	MCU 正在访问 DPRAM 标志： 0:表示空闲 1:表示正在访问	从站芯片正在访问 DPRAM 标志： 0：表示空闲 1：表示正在访问

11. 表 11：I/O 配置数据（CFG 数据）

- I/O 配置数据：在 PROFIBUS 通讯中，用特定的 16 进制数据定义 PROFIBUS 输入和输出数据。
- 本产品 PROFIBUS 的 I/O 数量（CFG 数据数量）最大可为 200； I/O 配置数据请见下表中的 “代码” ；
- 在 PROFIBUS 通讯中，为确保 PROFIBUS 输入/输出数据的完整性，将数据分成如下类型：

  Byte 完整： 用于开关量；

  Word 完整： 用于模拟量；

  全部输入/输出完整： 用于浮点数；

代码	说明	代码	说明
	byte input, Byte 完整		**byte input, 全部输入/输出完整**
0x10	1 byte input, Byte 完整	0x90	1 byte input, 全部输入/输出完整
0x11	2 byte input, Byte 完整	0x91	2 byte input, 全部输入/输出完整
0x12	3 byte input, Byte 完整	0x92	3 byte input, 全部输入/输出完整
0x13	4 byte input, Byte 完整	0x93	4 byte input, 全部输入/输出完整
0x14	5 byte input, Byte 完整	0x94	5 byte input, 全部输入/输出完整
0x15	6 byte input, Byte 完整	0x95	6 byte input, 全部输入/输出完整
0x16	7 byte input, Byte 完整	0x96	7 byte input, 全部输入/输出完整
0x17	8 byte input, Byte 完整	0x97	8 byte input, 全部输入/输出完整
0x18	9 byte input, Byte 完整	0x98	9 byte input, 全部输入/输出完整
0x19	10 byte input, Byte 完整	0x99	10 byte input, 全部输入/输出完整
0x1A	11 byte input, Byte 完整	0x9A	11 byte input, 全部输入/输出完整
0x1B	12 byte input, Byte 完整	0x9B	12 byte input, 全部输入/输出完整
0x1C	13 byte input, Byte 完整	0x9C	13 byte input, 全部输入/输出完整
0x1D	14 byte input, Byte 完整	0x9D	14 byte input, 全部输入/输出完整
0x1E	15 byte input, Byte 完整	0x9E	15 byte input, 全部输入/输出完整
0x1F	16 byte input, Byte 完整	0x9F	16 byte input, 全部输入/输出完整

	byte output, Byte 完整		byte output, 全部输入/输出完整
0x20	1 byte output, Byte 完整	0xA0	1 byte output, 全部输入/输出完整
0x21	2 byte output, Byte 完整	0xA1	2 byte output, 全部输入/输出完整
0x22	3 byte output, Byte 完整	0xA2	3 byte output, 全部输入/输出完整
0x23	4 byte output, Byte 完整	0xA3	4 byte output, 全部输入/输出完整
0x24	5 byte output, Byte 完整	0xA4	5 byte output, 全部输入/输出完整
0x25	6 byte output, Byte 完整	0xA5	6 byte output, 全部输入/输出完整
0x26	7 byte output, Byte 完整	0xA6	7 byte output, 全部输入/输出完整
0x27	8 byte output, Byte 完整	0xA7	8 byte output, 全部输入/输出完整
0x28	9 byte output, Byte 完整	0xA8	9 byte output, 全部输入/输出完整
0x29	10 byte output, Byte 完整	0xA9	10 byte output, 全部输入/输出完整
0x2A	11 byte output, Byte 完整	0xAA	11 byte output, 全部输入/输出完整
0x2B	12byte output, Byte 完整	0xAB	12byte output, 全部输入/输出完整
0x2C	13 byte output, Byte 完整	0xAC	13 byte output, 全部输入/输出完整
0x2D	14 byte output, Byte 完整	0xAD	14 byte output, 全部输入/输出完整
0x2E	15 byte output, Byte 完整	0xAE	15 byte output, 全部输入/输出完整
0x2F	16 byte output, Byte 完整	0xAF	16 byte output, 全部输入/输出完整
	byte input/output, Byte 完整		**byte input/output, 全部输入/输出完整**
0x30	1 byte input/output, Byte 完整	0xB0	1 byte input/output, 全部输入/输出完整
0x31	2 byte input/output, Byte 完整	0xB1	2 byte input/output, 全部输入/输出完整
0x32	3 byte input/output, Byte 完整	0xB2	3 byte input/output, 全部输入/输出完整
0x33	4 byte input/output, Byte 完整	0xB3	4 byte input/output, 全部输入/输出完整
0x34	5 byte input/output, Byte 完整	0xB4	5 byte input/output, 全部输入/输出完整
0x35	6 byte input/output, Byte 完整	0xB5	6 byte input/output, 全部输入/输出完整
0x36	7 byte input/output, Byte 完整	0xB6	7 byte input/output, 全部输入/输出完整
0x37	8 byte input/output, Byte 完整	0xB7	8 byte input/output, 全部输入/输出完整
0x38	9 byte input/output, Byte 完整	0xB8	9 byte input/output, 全部输入/输出完整
0x39	10 byte input/output, Byte 完整	0xB9	10 byte input/output, 全部输入/输出完整
0x3A	11 byte input/output, Byte 完整	0xBA	11 byte input/output, 全部输入/输出完整
0x3B	12byte input/output, Byte 完整	0xBB	12byte input/output, 全部输入/输出完整
0x3C	13 byte input/output, Byte 完整	0xBC	13 byte input/output, 全部输入/输出完整
0x3D	14 byte input/output, Byte 完整	0xBD	14 byte input/output, 全部输入/输出完整
0x3E	15 byte input/output, Byte 完整	0xBE	15 byte input/output, 全部输入/输出完整
0x3F	16 byte input/output, Byte 完整	0xBF	16 byte input/output, 全部输入/输出完整
	word input, word 完整		**word input, 全部输入/输出完整**
0x50	1 word input, word 完整	0xD0	1 word input, 全部输入/输出完整
0x51	2 word input, word 完整	0xD1	2 word input, 全部输入/输出完整
0x52	3 word input, word 完整	0xD2	3 word input, 全部输入/输出完整
0x53	4 word input, word 完整	0xD3	4 word input, 全部输入/输出完整
0x54	5 word input, word 完整	0xD4	5 word input, 全部输入/输出完整
0x55	6 word input, word 完整	0xD5	6 word input, 全部输入/输出完整
0x56	7 word input, word 完整	0xD6	7 word input, 全部输入/输出完整
0x57	8 word input, word 完整	0xD7	8 word input, 全部输入/输出完整
0x58	9 word input, word 完整	0xD8	9 word input, 全部输入/输出完整
0x59	10 word input, word 完整	0xD9	10 word input, 全部输入/输出完整
0x5A	11 word input, word 完整	0xDA	11 word input, 全部输入/输出完整
0x5B	12 word input, word 完整	0xDB	12 word input, 全部输入/输出完整

0x5C	13 word input, word 完整	0xDC	13 word input, 全部输入/输出完整
0x5D	14 word input, word 完整	0xDD	14 word input, 全部输入/输出完整
0x5E	15 word input, word 完整	0xDE	15 word input, 全部输入/输出完整
0x5F	16 word input, word 完整	0xDF	16 word input, 全部输入/输出完整
	word output, word 完整		**word output, 全部输入/输出完整**
0x60	1 word output, word 完整	0xE0	1 word output, 全部输入/输出完整
0x61	2 word output, word 完整	0xE1	2 word output, 全部输入/输出完整
0x62	3 word output, word 完整	0xE2	3 word output, 全部输入/输出完整
0x63	4 word output, word 完整	0xE3	4 word output, 全部输入/输出完整
0x64	5 word output, word 完整	0xE4	5 word output, 全部输入/输出完整
0x65	6 word output, word 完整	0xE5	6 word output, 全部输入/输出完整
0x66	7 word output, word 完整	0xE6	7 word output, 全部输入/输出完整
0x67	8 word output, word 完整	0xE7	8 word output, 全部输入/输出完整
0x68	9 word output, word 完整	0xE8	9 word output, 全部输入/输出完整
0x69	10 word output, word 完整	0xE9	10 word output, 全部输入/输出完整
0x6A	11 word output, word 完整	0xEA	11 word output, 全部输入/输出完整
0x6B	12 word output, word 完整	0xEB	12 word output, 全部输入/输出完整
0x6C	13 word output, word 完整	0xEC	13 word output, 全部输入/输出完整
0x6D	14 word output, word 完整	0xED	14 word output, 全部输入/输出完整
0x6E	15 word output, word 完整	0xEE	15 word output, 全部输入/输出完整
0x6F	16 word output, word 完整	0xEF	16 word output, 全部输入/输出完整
	word input/output, Word 完整		**word input/output, 全部输入/输出完整**
0x70	1 word input/output, Word 完整	0xF0	1 word input/output, 全部输入/输出完整
0x71	2 word input/output, Word 完整	0xF1	2 word input/output, 全部输入/输出完整
0x72	3 word input/output, Word 完整	0xF2	3 word input/output, 全部输入/输出完整
0x73	4 word input/output, Word 完整	0xF3	4 word input/output, 全部输入/输出完整
0x74	5 word input/output, Word 完整	0xF4	5 word input/output, 全部输入/输出完整
0x75	6 word input/output, Word 完整	0xF5	6 word input/output, 全部输入/输出完整
0x76	7 word input/output, Word 完整	0xF6	7 word input/output, 全部输入/输出完整
0x77	8 word input/output, Word 完整	0xF7	8 word input/output, 全部输入/输出完整
0x78	9 word input/output, Word 完整	0xF8	9 word input/output, 全部输入/输出完整
0x79	10 word input/output, Word 完整	0xF9	10 word input/output, 全部输入/输出完整
0x7A	11 word input/output, Word 完整	0xFA	11 word input/output, 全部输入/输出完整
0x7B	12word input/output, Word 完整	0xFB	12word input/output, 全部输入/输出完整
0x7C	13 word input/output, Word 完整	0xFC	13 word input/output, 全部输入/输出完整
0x7D	14 word input/output, Word 完整	0xFD	14 word input/output, 全部输入/输出完整
0x7E	15 word input/output, Word 完整	0xFE	15 word input/output, 全部输入/输出完整
0x7F	16 word input/output, Word 完整	0xFF	16 word input/output, 全部输入/输出完整

第三章 通讯流程

一、通讯基本过程

1．选定一种通讯模式

用户根据需要选择适合自己产品的通讯接口，设置方式详见本手册“第三章”，其中芯片型号：DSDPV1-R，只支持双口 RAM 接口,故在使用此型号芯片时可忽略此步骤。

2．获取 DPRAM 访问权限

在用户选择与 M 卡的通讯接为双口 RAM 或 SPI 时,才需要在读写 DPRAM 前获取控制权，但 M 卡或芯片的通讯接口为 UART 时则不涉及 DPRAM 控制权限问题,关于 DPRAM 控制权限问题详见“第五章 DPRAM 控制权”或“C 代码说明”。

3．选择 M 卡是否支持 DPV1 功能

本手册所包含的所有产品都支持 DPV1 功能，用户需要在“软件初始化”前选择是否开启 DPV1 功能，具体选择方式详见“C 源代码说明”，如用户没有选择开启 DPV1 功能而直接进行“软件初始化”，则不支持 DPV1 功能。

4．软件初始化

用户模板上电后，若检测到“接口板有效标志==0XC5”则可将初始化参数（包括站号、ID 号、SPC3_Register_status、Profibus 输入数据长度、Profibus 输出数据长度、用户参数长度、用户参数最大长度、用户参数判断方式、Cfg 数据长度、Cfg 数据判断方式、Cfg 数据以及初始化校验和)写入双口 RAM 的初始化数据区(双口 RAM 地址：0x0020-0x00F8),用户 MCU 只有完成对 M 卡进行初始化后才可以进入数据交换，具体初始化过程详见“C 源代码说明”。

5．数据交换

初始化完成后，M 卡进入数据交换状态。此时，用户 MCU 主动向 M 卡发送“输入数据报文”包含 PROFIBUS 输入数据，M 卡回答“输出数据报文”包含 PROFIBUS 输出数据。

第四章 DPRAM 控制权

一、概述

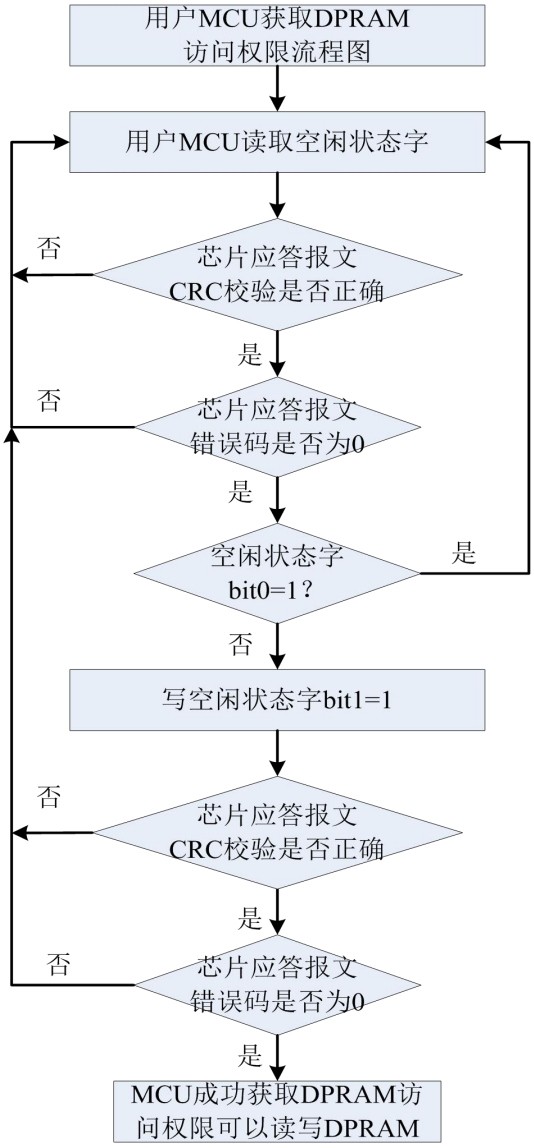
用户 MCU 完成一次访问 DPRAM 的过程主要包括获取 DPRAM 访问权限、读写 DPRAM 据以及释放 DPRAM 访问权限，用户 MCU 和 M 卡进行数据交换时，都需要访问 DPRAM，但二者不能同时访问 DPRAM，这就是 DPRAM 卡访问权限（即 M 卡访问权限）的问题。

M 系列板卡及 DSDPV1 系列芯片的通讯接口为双口 RAM 及 SPI 接口时，用户 MCU 在对 DPRAM 读写操作前要获取 DPRAM 控制权，但是这两种接口（双口 RAM、SPI）获取 DPRAM 控制权的方式不同，现做详细介绍。

1．M 卡工作在 SPI 接口模式下的 DPRAM 控制权

MCU 和从站芯片访问 DPRAM 采用查询的方式查看 DPRAM 是否空闲，从而获取访问权限。具体方法是:当用户 MCU 想要访问 DPRAM 是否空闲,这时 MCU 需要读取空闲状态字(地址为 0x014)，空闲状态字表明 DPRAM 忙时(空闲状态字为 0x01)，表示此时芯片占用 DPRAM 访问权限，用户 MCU 不得访问 DPRAM 数据,此时用户 MCU 可以不断的查询访问空闲状态字，直到空闲状态字为 0x00 时，表示此时芯片不访问 DPRAM，用户 MCU 此时需要写空闲状态字为 0x02，表明 MCU 获得 DPRAM 访问权限，此时 MCU 可以访问 DPRAM 数据，当 MCU 访问完成后，MCU 需要释放访问权限，此时需要在写空闲状态字为 0x00。详见对应“C 源代码说明”和空闲状态字(0x014)定义“第二章 DPRAM 数据结构/表 10”

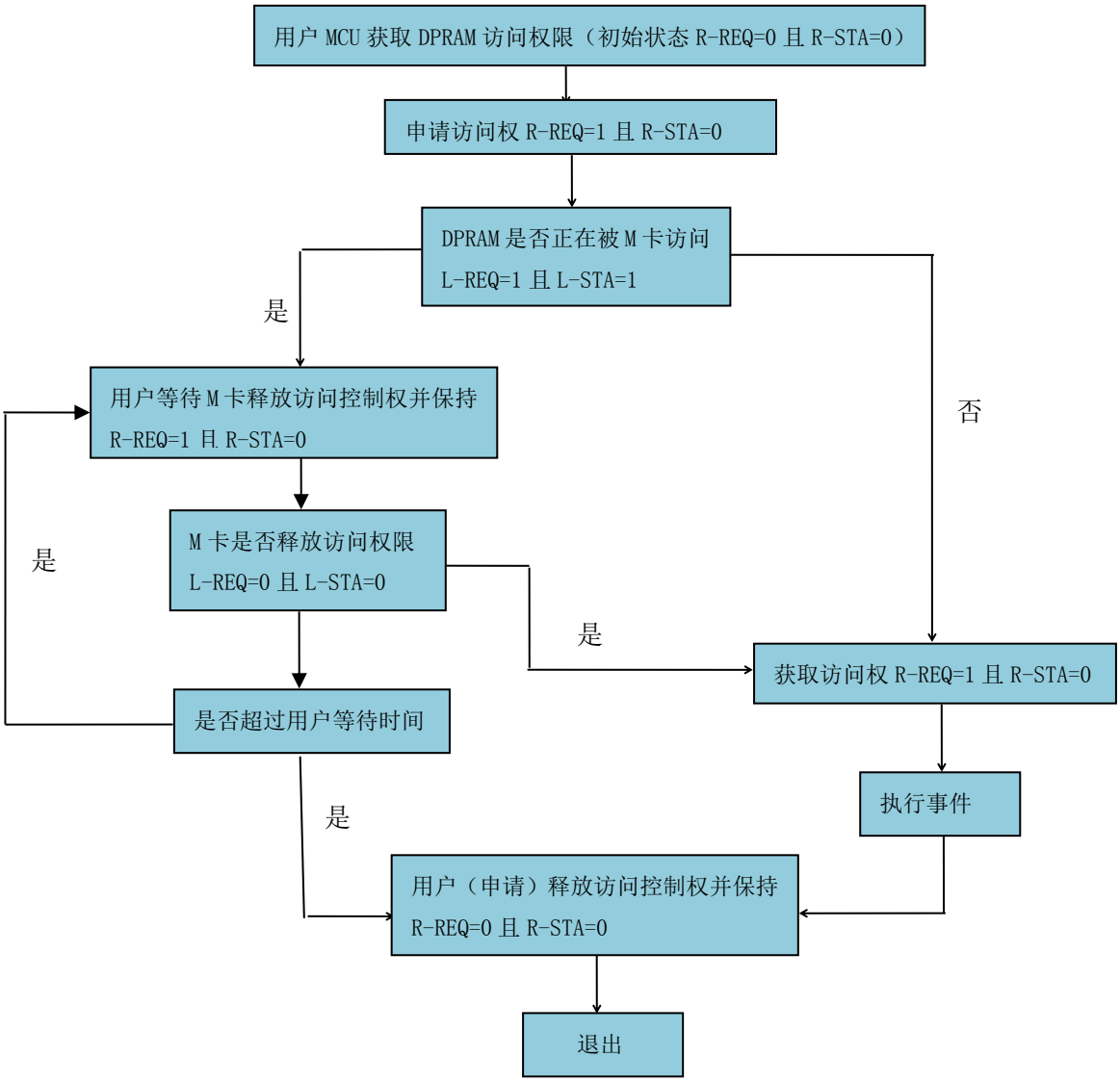
若 MCU 占用 DPRAM 访问权限,但用户 MCU 长时间不访问 DPRAM,或者处于其它特殊情况 MCU 没有放弃访问权限(没有置空闲状态字为 0x00)，这样会导致从站芯片始终不能获得 DPRAM 访问权限，因此从站芯片设置了 U_Timer 定时器，监控用户 MCU 是否长时间未访问 DPRAM，MCU 在初始化时通过 DPRAM0x015 单元可以设置是否开启 U_Timer 定时器,当用户关闭此定时器后，芯片不启动 U_Timer。如果开启 U_Timer，则在用户 MCU 获得 DPRAM 访问权限后，U_Timer 开始计时，MCU 每发送一条报文都会重新启动 U_Timer 计时，当 MCU 放弃 DPRAM 访问权限后，芯片关闭 U_Timer 计时。U_Timer 定时器溢出时间暂定为 0.5ms。



MCU 获取 DPRAM 访问权限程序流程示例

2．M 卡工作在双口 RAM 接口模式下的 DPRAM 控制权

用户访问 DPRAM 时，M 卡不会对 DPRAM 进行读写操作，直到用户读写完成后，才可以访问 DPRAM。用户 CPU 和 M 卡之间的通信接口采用获得访问权限的机制，用户在读写 DPRAM 之前，必须先获得 DPRAM 访问权限。用户申请访问权限通过用户请求访问权限(R_REQ)、用户获得访问权限(R_STA)、M 卡占用访问权限(L_REQ)、M 卡占用访问权限(L_STA)握手信号实现，具体过程详见以下流程图或“C 代码说明”。


用户 MCU 获取 DPRAM 访问权限（初始状态 R-REQ=0 且 R-STA=0）
申请访问权 R-REQ=1 且 R-STA=0
DPRAM 是否正在被 M 卡访问
L-REQ=1 且 L-STA=1
是
否
用户等待 M 卡释放访问控制权并保持
R-REQ=1 且 R-STA=0
M 卡是否释放访问权限
L-REQ=0 且 L-STA=0
是
是
获取访问权 R-REQ=1 且 R-STA=0
是否超过用户等待时间
执行事件
是
用户（申请）释放访问控制权并保持
R-REQ=0 且 R-STA=0
退出

第五章 双口 RAM 接口

一、用户 MCU 与 M 卡的通讯接口为双口 RAM

1. “fsmc_sram.h”

（1）用户板拨码定义

```
/*********************************/
#ifndef __FSMC_SRAM_H
#define __FSMC_SRAM_H
/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
/*********************************/
/*      拨码开关设置 定义          */
/*********************************/

//--------SW5 : DSDPV1 芯片地址 设置---------------------------------//
#define   SW5_1   (PBin(0)&0x01)
#define   SW5_2   (PBin(1)&0x01)
#define   SPISlave_Addr1   (((SW5_2<<1)|(SW5_1))&0xFF)
//--------SW3 : DSDPV1 芯片 通讯模式选择(DPRAM/SPI/UART)设置---------//
#define   SW3_1   (PCin(10)&0x01)
#define   SW3_2   (PCin(11)&0x01)
#define   DSDPV1_Mode (((SW3_2<<1)|(SW3_1))&0xFF)
//-------用于  波特率  PD(8 9 10) ------------//
#define  SW15_1    (PDin(8)&0x01)
#define  SW15_2    (PDin(9)&0x01)
#define  SW15_3    (PDin(10)&0x01)
#define  DSDPV1_UartMode_Baud (((SW15_3<<2)|(SW15_2<<1)|(SW15_1))&0xFF)
//-------用于 报文中字符间隔 PE(13 14 15) ------------//
#define  SW14_1    (PEin(13)&0x01)
#define  SW14_2    (PEin(14)&0x01)
#define  SW14_3    (PEin(15)&0x01)
#define  DSDPV1_UartMode_Interval (((SW14_3<<2)|(SW14_2<<1)|(SW14_1))&0xFF)
```

（2）槽索引相关定义

```
/*********************************/
/*      槽-索引相关定义           */
/*********************************/
#define MAX_SI_NUM    13
#define MAX_SI_LEN    240
/*-------SLOT_00 定义--------------*/
#define SLOT_00          0
#define S00_INDEX_01     1
#define S00_INDEX_02     2

/*-------SLOT_01 定义--------------*/
#define SLOT_01          1
#define S01_INDEX_01     1
#define S01_INDEX_02     2
#define S01_INDEX_03     3
#define S01_INDEX_04     4
#define S01_INDEX_05     5
```

```
#define SO1_INDEX_06    6
#define SO1_INDEX_07    7
#define SO1_INDEX_08    8
#define SO1_INDEX_09    9
#define SO1_INDEX_10    10
#define SO1_INDEX_11    11
/*-------SLOT_02 定义--------------*/
#define SLOT_02         2
#define SO2_INDEX_01    1
#define SO2_INDEX_02    2
//=================  IM    ================//
#define IM_SLOT         0
#define IM_INDEX        255
#define IM0_SUBINDEX    65000
//--------------为 RM 使用-----------------//
#define SLOT_03         4
#define SO3_INDEX_01    1
//---------------为 RM 使用-----------------//
#define SI_R     0     //当前槽-索引为只读
#define SI_W     1    //当前槽-索引为只写
#define SI_RW    2    //当前槽-索引为可读可写

#define SI_C1     0    //当前槽-索引仅用于 C1
#define SI_C2     1    //当前槽-索引仅用于 C2
#define SI_C12    2    //当前槽-索引为 C1，C2 都可用
#define SI_OK               0
#define SI_NOK              0xFF
#define SI_WRONG_SLOT       1
#define SI_WRONG_INDEX      2
#define SI_WRONG_LENGTH     3
#define SI_WRONG_RW         4
#define NO_IMCALL       0
#define RE_IMCALL       1
#define RES_IMCALL      2
#define Bank1_SRAM1_ADDR    ((uint32_t)0x60000000)
#define Bank1_SRAM2_ADDR    ((uint32_t)0x64000000)
#define Bank1_SRAM3_ADDR    ((uint32_t)0x68000000)
#define Bank1_SRAM4_ADDR    ((uint32_t)0x6C000000)
#define MaxPrmLen 237
#define MaxCfgLen 200
#define Fosc          14.7456
#define DELAY_NUM     ((Fosc*1000-11)/6)
#define dpramadr 0x0000
#define dpramend 0x07FE
```

（3）握手信号定义

```
/*************************/
/*  握手信号定义-双口 RAM */
/*************************/
/*
#define SET_R_REQ  (PORTB=PORTB|0x02)  //置位
#define SET_R_STA  (PORTB=PORTB|0x04)    //置位
#define L_REQ  (PINB&0x10)
#define L_STA  (PINB&0x20)
#define ABORT_DPRAM  (PORTB=PORTB&0xF9)  //释放控制权
#define SET_R_REQ (GPIOC->BSRR = GPIO_Pin_6)
#define SET_R_STA (GPIOC->BSRR = GPIO_Pin_7)
#define L_REQ  (GPIOC->IDR & 0x0100)
```

```
#define L_STA  (GPIOC->IDR & 0x0200)
#define ABORT_DPRAM (GPIOC->BRR = 0x00C0)
```

（4）IO 的位操作实现

```
/*  IO 口的位带操作实现 */
//----------------------支持位带操作的两个内存区的范围是：---------//
//                0x2000 0000 - 0x200F FFFF（SRAM 区中的最低 1MB）     //
//                0x4000 0000 - 0x400F FFFF（片上外设区中的最低 1MB） //
//-----------------------------------------------------------------//
#define  BITBAND(addr, bitnum)   ((addr &
0xF0000000)+0x2000000+((addr&0xFFFFF)<<5)+(bitnum<<2))
#define  MEM_ADDR(addr)          *((volatile unsigned long *)(addr))
#define  BIT_ADDR(addr,bitnum)     MEM_ADDR(BITBAND(addr,bitnum))
#define    GPIOA_ODR_Addr (GPIOA_BASE+12)   //0x4001080C
#define    GPIOB_ODR_Addr (GPIOB_BASE+12)   //0x40010C0C
#define    GPIOC_ODR_Addr (GPIOC_BASE+12)   //0x4001100C
#define    GPIOD_ODR_Addr (GPIOD_BASE+12)   //0x4001140C
#define    GPIOE_ODR_Addr (GPIOE_BASE+12)   //0x4001180C
#define    GPIOF_ODR_Addr (GPIOF_BASE+12)   //0x40011A0C
#define    GPIOG_ODR_Addr (GPIOG_BASE+12)   //0x40011E0C

#define    GPIOA_IDR_Addr (GPIOA_BASE+8)    //0x40010808
#define    GPIOB_IDR_Addr (GPIOB_BASE+8)    //0x40010C08
#define    GPIOC_IDR_Addr (GPIOC_BASE+8)    //0x40011008
#define    GPIOD_IDR_Addr (GPIOD_BASE+8)    //0x40011408
#define    GPIOE_IDR_Addr (GPIOE_BASE+8)    //0x40011808
#define    GPIOF_IDR_Addr (GPIOF_BASE+8)    //0x40011A08
#define    GPIOG_IDR_Addr (GPIOG_BASE+8)    //0x40011E08

#define   PAout(n)    BIT_ADDR(GPIOA_ODR_Addr ,n)
#define   PAin(n)     BIT_ADDR(GPIOA_IDR_Addr ,n)
#define   PBout(n)    BIT_ADDR(GPIOB_ODR_Addr ,n)
#define   PBin(n)     BIT_ADDR(GPIOB_IDR_Addr ,n)
#define   PCout(n)    BIT_ADDR(GPIOC_ODR_Addr ,n)
#define   PCin(n)     BIT_ADDR(GPIOC_IDR_Addr ,n)
#define   PDout(n)    BIT_ADDR(GPIOD_ODR_Addr ,n)
#define   PDin(n)     BIT_ADDR(GPIOD_IDR_Addr ,n)
#define   PEout(n)    BIT_ADDR(GPIOE_ODR_Addr ,n)
#define   PEin(n)     BIT_ADDR(GPIOE_IDR_Addr ,n)
#define   PFout(n)    BIT_ADDR(GPIOF_ODR_Addr ,n)
#define   PFin(n)     BIT_ADDR(GPIOF_IDR_Addr ,n)
#define   PGout(n)    BIT_ADDR(GPIOG_ODR_Addr ,n)
#define   PGin(n)     BIT_ADDR(GPIOG_IDR_Addr ,n)
#define   DSDPV1_PBF   PGin(12)// P12=PBF
#define   DSDPV1_BR    PGin(13)//P13=BR
#define   DSDPV1_INIT  PGin(14)//P14=Init
#define   PBF_LED   PCout(6)
#define   BR_LED    PCout(7)
#define   INIT_LED  PCout(8)
//0x4000-0x47FF
#pragma abs_address:dpramadr DPRAM  dpram;
#pragma end_abs_address
```

（5）结构体定义

```
typedef struct
{
 uint8_t slot;  //槽号
```

```
 uint8_t index; //索引号
 uint8_t rw;    //本槽-索引的读写 0：R  1：W  2：RW
 uint8_t commu; //本槽-索引可用于 C1/C2  0：C1  1：C2  2：C12
 //uint8_t data[MAX_SI_LEN];  //读写数据
}SLOT_INDEX;
  /*-------请详细了解以下结构体内容----------*/
typedef struct
{
  uint8_t Version_Code;  //0000 代码版本
  uint8_t Year_Code;     //0001 版本年份
  uint8_t Month_Code;    //0002 版本月份
  uint8_t Day_Code;      //0003 版本日期
  uint8_t HardwareInit_Ldpram;  //0004   标志  接口板是否已完成自身硬件初始化
  uint8_t EnableDPv1_Rdpram;    //0005   开启  接口板 V1 功能标志
  uint8_t AccessCounter_LRdpram;//0006 双边 DPRAM 访问计数(带符号字 -32768 ~ +32767)
uint8_t AccessCounter_LRdpram_reserved;//0007
 uint16_t OfflineCounter_DP;    //0008~9 DP 通信掉线次数计数(无符号字 0 ~ 65535)
    /*-------双口 RAM----------*/
  uint8_t L_status1;           //000A  接口板状态字节 1
  uint8_t L_status2;           //000B  接口板状态字节 2
  uint8_t R_status1;           //000C  用户板状态字节 1
  uint8_t R_status2;           //000D  用户板状态字节 2
  uint8_t L_command1;          //000E  接口板命令字节 1
  uint8_t L_command2;          //000F  接口板命令字节 2
  uint8_t R_command1;          //0010  用户板命令字节 1
  uint8_t R_command2;          //0011  用户板命令字节 2
  uint8_t L_status3;           //0012  接口板状态字节 3
  uint8_t L_status4;           //0013  接口板状态字节 4
   /*-------SPI 接口----------*/
  uint8_t SPI_Status;          //0014  控制权申请
    uint8_t SPI_UTimer_Enable;  //0015 暂时保留
    uint8_t Reserved11[2];       //0016-0017 保留
     /*-------UART 接口----------*/
    uint8_t UART_Status;         //0018
    uint8_t UART_BaudRate;       //0019
    uint8_t UART_TSDR;           //001A
    uint8_t Reserved1[5];       //001B~001F 预留字节
//=================================================//
  uint8_t InitData_DPv0[215];  //0020~00F6 V0 初始化数据
  uint8_t CRC_H_DPv0;          //00F7 V0 初始化 CRC 校验和高字节
  uint8_t CRC_L_DPv0;          //00F8 V0 初始化 CRC 校验和低字节
  uint8_t Reserved2[55];       //00F9~012F 预留字节
  uint8_t InData_DP[245];      //0130~0224 PROFIBUS 输入数据
  uint8_t CRC_H_InData_DP;     //0225      PROFIBUS 输入数据 CRC 校验和高字节
  uint8_t CRC_L_InData_DP;     //0226      PROFIBUS 输入数据 CRC 校验和低字节
  uint8_t DiagData_DP[240];    //0227~0316 PROFIBUS 诊断数据
  uint8_t CRC_H_DiagData_DP;   //0317      PROFIBUS 诊断数据 CRC 校验和高字节
  uint8_t CRC_L_DiagData_DP;   //0318      PROFIBUS 诊断数据 CRC 校验和低字节
  uint8_t OutData_DP[245];     //0319~040D PROFIBUS 输出数据
  uint8_t CRC_H_OutData_DP;    //040E      PROFIBUS 输出数据 CRC 校验和高字节
  uint8_t CRC_L_OutData_DP;    //040F      PROFIBUS 输出数据 CRC 校验和低字节
  uint8_t PrmData_DP[238];     //0410~04FD PROFIBUS 参数化数据
  uint8_t CRC_H_PrmData_DP;    //04FE      PROFIBUS 参数化数据 CRC 校验和高字节
  uint8_t CRC_L_PrmData_DP;    //04FF      PROFIBUS 参数化数据 CRC 校验和低字节
  uint8_t CfgData_DP[203];     //0500~05CA PROFIBUS 配置数据
  uint8_t CRC_H_CfgData_DP;    //05CB     PROFIBUS 配置数据 CRC 校验和高字节
  uint8_t CRC_L_CfgData_DP;    //05CC      PROFIBUS 配置数据 CRC 校验和低字节
  uint8_t Reserved4[3];        //05CD~05CF 预留字节
```

```
  uint8_t AcycData_DPv1c1[244];  //05D0~006C3 PROFIBUS V1 非循环通信数据
  uint8_t CRC_H_AcycData_DPv1c1; //06C4      PROFIBUS V1 非循环通信数据 CRC 校验和高字节
  uint8_t CRC_L_AcycData_DPv1c1; //06C5      PROFIBUS V1 非循环通信数据 CRC 校验和低字节
  uint8_t Reserved5[10];        //06C6~06CF 预留字节
  uint8_t AcycData_DPv1c2[244];  //06D0~07C3 PROFIBUS V1 非循环通信数据
  uint8_t CRC_H_AcycData_DPv1c2; //07C4      PROFIBUS V1 非循环通信数据 CRC 校验和高字节
  uint8_t CRC_L_AcycData_DPv1c2; //07C5      PROFIBUS V1 非循环通信数据 CRC 校验和低字节
  uint8_t Reserved6[56];        //07C6~07FF 预留字节
  uint8_t Write_L_Int;       //07FE
  uint8_t Read_R_Int;        //07FF
}DPRAM;
// DPRAM  *dpram  __at (0x60000000);
//=============CRC 查找表(字节 8bit)============================//
/* Table of CRC values for high-order byte       */
static unsigned char auchCRCHi[] = { 略} ;//详见 fsmc_sram.h
/* Table of CRC values for low-order byte        */
static unsigned char auchCRCLo[] = {略} ; //详见 fsmc_sram.h
uint16_t FCS(uint8_t *start_ptr,uint16_t len)
{
  uint16_t x;
  uint16_t i;
  x=0;
  for(i=0;i<len;i++)
  {
    x = x+(*(start_ptr + i));
  }
  return x;
}
*/
extern DPRAM  dpram;
```

（6）相关函数

```
void DPRAM_Init(void);
void SRAM_WriteBuffer(uint16_t* pBuffer, uint32_t WriteAddr, uint32_t NumHalfwordToWrite);
void SRAM_ReadBuffer(uint16_t* pBuffer, uint32_t ReadAddr, uint32_t NumHalfwordToRead);
void reset_FPGA(void);
void reset_DSDPV1(void);
void get_dpram(void);
uint8_t dpram_hold(void);
uint8_t dp_address(void);
uint8_t PrmCfg_mode(void);
void Init_Buf(void);
void slot_index_init(void);
void Y_HardWare_InitData(void);
void Y_SoftWare_InitData(void);
void DP_DSDPRAM_DiagHandler(void);
void DPV0V1_DataEx_DSdpramMode(void);
void UserApplication_IoBuf_Dealwith(void);
void Shared_GPIO_Init(void);
void Y_GPIO_Init(void);
void DPRAMMode_GPIO_Init(void);
void delay_ms(uint16_t count);
void delay_us(uint16_t count);
void LED(void);
void out_led(uint32_t val);
void Y_Dpram_IntR_EXIT_config(void);
void EXTI0_IRQHandler(void);
uint16_t CRC_avr(uint8_t *puchMsg ,uint16_t usDataLen);
```

```
uint16_t in_led(void);
uint8_t deal_acyclic_read_data(uint8_t slot, uint8_t index, uint8_t len);
void deal_acyclic_write_data(uint8_t slot, uint8_t index, uint8_t len);
void D_Y_GPIO_Init(void);
void DSDPV1_LEDStatus(void);
void SAMPLE_GPIO_Init(void);
void reset_STM32(void);
void deal_im_data(void);
/*-------------------------------------------------------------------------------*
/*  coding of error_code_1 at DPV1, negative response                            */
/*-------------------------------------------------------------------------------*
/* error_class */
#define DPSE_ERRCL_APPLICATION          10
#define DPSE_ERRCL_ACCESS               11
#define DPSE_ERRCL_RESOURCE             12
#define DPSE_ERRCL_USER                 13
/* error_code for  DPSE_ERRCL_APPLICATION *//
#define DPSE_ERRCL_APP_READ             0
#define DPSE_ERRCL_APP_WRITE            1
#define DPSE_ERRCL_APP_MODULE           2
#define DPSE_ERRCL_APP_VERSION          8
#define DPSE_ERRCL_APP_NOTSUPP          9
#define DPSE_ERRCL_APP_USER             10
/* error_code for  DPSE_ERRCL_ACCESS */
#define DPSE_ERRCL_ACC_INV_INDEX        0
#define DPSE_ERRCL_ACC_WRITE_LEN        1
#define DPSE_ERRCL_ACC_INV_SLOT         2
#define DPSE_ERRCL_ACC_TYPE             3
#define DPSE_ERRCL_ACC_INV_AEREA        4
#define DPSE_ERRCL_ACC_STATE            5
#define DPSE_ERRCL_ACC_ACCESS           6
#define DPSE_ERRCL_ACC_INV_RANGE        7
#define DPSE_ERRCL_ACC_INV_PARAM        8
#define DPSE_ERRCL_ACC_INV_TYPE         9
#define DPSE_ERRCL_ACC_USER             10
/* error_code for  DPSE_ERRCL_RESOURCE  */
#define DPSE_ERRCL_RES_READ_CONSTRAIN   0
#define DPSE_ERRCL_RES_WRITE_CONSTRAIN  1
#define DPSE_ERRCL_RES_BUSY             2
#define DPSE_ERRCL_RES_UNAVAIL          3
#define DPSE_ERRCL_RES_USER             8
#endif  /* __FSMC_SRAM_H */
/*-------------------------end-----------------------------------*
```

2. “Fsmc_sram.c”

（1）变量声明

```
#include "stm32f10x_exti.h"
#include "LCD_serial.h"
#include "spi.h"
#include "dsuart.h"
uint8_t EnableDPv1_Flag;
uint8_t diag_input[240];
uint8_t v0_output[244];  //V0 输出数据区
uint8_t v0_input[244];   //V0 输入数据区
uint8_t v1_output[240];  //V1 输出数据区
uint8_t v1_input[240];   //V1 输入数据区
```

```
uint8_t IM_output[64];    //IM 输出数据区
uint8_t IM_input[64];     //IM 输入数据区
uint8_t im0_body[64];
uint8_t IM_CallHeader[4];
uint8_t CRC_Buf[250];     //CRC 数据缓冲区
uint8_t   DP_Input_flag,trigger=0;
uint8_t   NewPrmData_HandleMode,NewCfgData_HandleMode;
uint8_t   New_Prmdata_Len,New_Cfgdata_Len;
uint8_t   New_Prmdata_Buf[MaxPrmLen],New_Cfgdata_Buf[MaxCfgLen];
uint8_t   si_status,im_status;
uint8_t   c_slot,c_index,c_len,c_subindex_H,c_subindex_L,c_rw,c_commu;
uint8_t Tn,Tn1,DIO_7;
uint8_t current_diag_len,flag_diag;
uint8_t dsdpv1_status;
uint8_t SPISlave_Addr;
uint8_t UARTMode_ADDR1;
extern uint8_t DpAddr;
extern uint8_t v1c1_output1,v1c1_output2,v1c2_output1,v1c2_output2;
extern uint8_t v1c1_input1,v1c1_input2,v1c2_input1,v1c2_input2;
extern uint8_t  ExtFlag,StaFlag;
extern uint8_t  ExtDiagData_input[3],StaDiagData_input[3];
SLOT_INDEX slot_index[MAX_SI_NUM];
extern DPRAM  gdpram;
//extern DPRAM  dpram;//在 spi.c 中定义
//DPRAM  dpram __attribute__((at((uint32_t)0x60000000)));
```

（2）CRC 查找表

```
//============================CRC 查找表=====================================
uint16_t CRC_avr(uint8_t *puchMsg ,uint16_t usDataLen)
{
  //uint8_t *puchMsg ;                  /* message to calculate CRC upon       */
  //uint16_t usDataLen ;                /* quantity of bytes in message        */
    uint8_t uchCRCHi = 0xFF ;           /* high byte of CRC initialized        */
    uint8_t uchCRCLo = 0xFF ;           /* low byte of CRC initialized         */
    uint8_t uIndex = 0 ;                /* will index into CRC lookup table    */
    while (usDataLen--)                /* pass through message buffer         */
    {
      uIndex = uchCRCHi ^ *puchMsg++ ;      /* calculate the CRC      */
      uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
      uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCLo << 8 | uchCRCHi);
}
```

（3）延时及复位

```
void delay_ms(uint16_t count)
{uint16_t i,n;
 for(n=count;n>0;n--)
  for(i=60000;i>0;i--); // count * (2664*(5+1)+11)/16M == 1ms (DELAY_NUM =2664)
}
void delay_us(uint16_t count)// 1--> 2x500nS(1us), 2--> 3x500nS(1.5us), 3--> 4x500nS(2us), 5--> 6x500nS(3us)
{uint16_t i,n;
 for(n=count;n>0;n--)
  for(i=2;i>0;i--);
}
//reset ASIC chip
void reset_STM32(void)
```

```
{//
  uint16_t i;
    for (i=0;i<800;i++){GPIOE->BSRR = GPIO_Pin_4;} //后高电平
    for (i=0;i<800;i++){GPIOE->BRR  = GPIO_Pin_4;} //先低电平
    for (i=0;i<800;i++){GPIOE->BSRR = GPIO_Pin_4;} //后高电平
    for (i=0;i<800;i++){GPIOE->BSRR = GPIO_Pin_5;} //后高电平
    for (i=0;i<800;i++){GPIOE->BRR  = GPIO_Pin_5;} //先低电平
    for (i=0;i<800;i++){GPIOE->BSRR = GPIO_Pin_5;} //后高电平
}
void reset_DSDPV1(void)
{
  uint16_t i;
    //复位 FPGA
    for (i=0;i<65000;i++){GPIOE->BSRR = GPIO_Pin_4;} //先高电平
    for (i=0;i<65000;i++){GPIOE->BRR  = GPIO_Pin_4;} //后低电平
}
```

（4）获取 DPRAM 控制权

```
//==============FPGA 双口 RAM 握手,获取 DPRAM 控制权=========================//
void get_dpram(void)
{
  uint16_t tem;
  tem=0;
  SET_R_REQ;
    while((GPIOC->IDR&0x00C0)!=0x00C0)
 {
   SET_R_REQ;
   // 第一次判断
   if((L_REQ==0) && (L_STA==0))
   {
     SET_R_STA;  //占有控制权
       tem=255;    //准备退出
   }
   else  //第一次判断失败
   {
       tem++;        //失败一次
         if(tem==0x100)tem=0;
   }
 } //end of while
 //if (tem==tryt){tem=0;}//没有获得控制权，返回值为 0
 //return(tem);//tem 等于 255 时，有可能没有获得 控制权
}
```

（5）设置 PROFIBUS 从站地址

```
//==============设置 PROFIBUS 从站地址=========================//
uint8_t dp_address(void)
{ uint8_t i;
    //i = (GPIOA->IDR & 0x007f) ;
    //i=get_keyboard();
    i = DpAddr;
    if(i>126){i=126;}
  return i;
}
```

（6）配置数据及用户参数正确与否判断方式

```
//==========配置数据及用户参数正确与否判断方式===============//
uint8_t PrmCfg_mode(void)
{ uint8_t i;
```

```
    //i = (GPIOA->IDR & 0x0080) ;
    i = 0x00;
  return i;
}
```

（7）指示灯

```
//==========指示灯===============//
void DSDPV1_LEDStatus(void)
{
    if(DSDPV1_PBF ==1){PBF_LED =0;} else  {PBF_LED =1;}//PBF 亮：与主站未连接
    if(DSDPV1_BR  ==1){BR_LED  =0;} else  {BR_LED  =1;}//BR 亮：识别到总线波特率
    //if(DSDPV1_INIT==1){INIT_LED=0;}else {INIT_LED=1;}//ready 亮：初始化完成

}
```

（8）初始化内容

```
//==========以下为初始化内容===============//
void Init_Buf(void)
{uint8_t i;
 for(i=0;i<244;i++){v0_input[i]=0;v0_output[i]=0;}
 for(i=0;i<240;i++){v1_output[i]=v1_input[i]=0;}
 //for(i=0;i<240;i++){v1_input[i]=i+1;}//C1_read_response
 for(i=0;i<240;i++){diag_input[i]=0;}
 for(i=0;i<64;i++) {im0_body[i]=0;}
 for(i=0;i<4;i++)  {IM_CallHeader[i]=0;}
 //=================== IM 数据初始化=================================//
 //==========制造商自定义(10),自定义用途（此 10 个字节可删除不用）=======//
 IM_input[0] = 0x00;
 IM_input[1] = 0x00;
 IM_input[2] = 0x00;
 IM_input[3] = 0x00;
 IM_input[4] = 0x00;
 IM_input[5] = 0x00;
 IM_input[6] = 0x00;
 IM_input[7] = 0x00;
 IM_input[8] = 0x00;
 IM_input[9] = 0x00;
 //================================制造商 ID (2)
 IM_input[10] = 0x0C;
 IM_input[11] = 0xC9;
 //================================设备订货号(20)
 IM_input[12] = 'P';
 IM_input[13] = 'B';
 IM_input[14] = '-';
 IM_input[15] = 'D';
 IM_input[16] = 'S';
 IM_input[17] = 'D';
 IM_input[18] = 'P';
 IM_input[19] = 'V';
 IM_input[20] = '1';
 IM_input[21] = 0;
 IM_input[22] = 0;
 IM_input[23] = 0;
 IM_input[24] = 0;
 IM_input[25] = 0;
 IM_input[26] = 0;
 IM_input[27] = 0;
 IM_input[28] = 0;
```

```
IM_input[29] = 0;
IM_input[30] = 0;
IM_input[31] = 0;
//==================================设备序列号(16)
IM_input[32] = 'S';
IM_input[33] = 'P';
IM_input[34] = 'I';
IM_input[35] = ' ';
IM_input[36] = '-';
IM_input[37] = 'D';
IM_input[38] = 'P';
IM_input[39] = 'v';
IM_input[40] = '1';
IM_input[41] = '-';
IM_input[42] = '1';
IM_input[43] = '4';
IM_input[44] = '-';
IM_input[45] = '1';
IM_input[46] = '2';
IM_input[47] = 0;
//==================================硬件版本(2)
IM_input[48] = 0x00;
IM_input[49] = 0x01;
//==================================软件版本(4)
IM_input[50] = 'V';
IM_input[51] = 1;
IM_input[52] = 0;
IM_input[53] = 0;
//====================更改次数计数(2):设备硬件模块配置更改的次数
IM_input[54] = 0x00;
IM_input[55] = 0x00;
//==================================行规 ID(2)
IM_input[56] = 0x00;
IM_input[57] = 0x00;
//==================================特殊行规类型（自定义）(2)
IM_input[58] = 0x00;
IM_input[59] = 0x00;
//==================================IM 版本(2)
IM_input[60] = 0x00;
IM_input[61] = 0x00;
//==================================IM 支持(2)
IM_input[62] = 0x00;
IM_input[63] = 0x00;
//================ IM 数据初始化 完毕 ================================//
if(PrmCfg_mode()==0x80)
{
    NewPrmData_HandleMode = 2; //用户参数正确与否判断方式
    NewCfgData_HandleMode = 2; //配置数据正确与否判断方式
    //DSDPV1_SPI1_PrmCfg_EXIT_config();//外部中断初始化，  开启外部中断
    DSDPV1_DSUART1_PrmCfg_EXIT_config();
}
else
{
    NewPrmData_HandleMode = 0; //用户参数正确与否判断方式
  NewCfgData_HandleMode = 0; //配置数据正确与否判断方式
    EXTI_DeInit();//关闭 外部中断
}
EnableDPv1_Flag  = 1; //DPv1 功能使能标志
```

```
}
/*****************************************/
/*       DPv1 槽-索引信息初始化           */
/*初始化时，可以按照 slot index 的升序排列  */
/*****************************************/
void slot_index_init(void)
{
  //example: 本例中，有两个可用槽-索引，槽-索引信息都在 myhead.h 进行了预定义
  slot_index[0].slot  = SLOT_00;        //0
  slot_index[0].index = S00_INDEX_01;   //1
  slot_index[0].rw    = SI_R;           //0
  slot_index[0].commu = SI_C12;

  slot_index[1].slot  = SLOT_01;        //1
  slot_index[1].index = S01_INDEX_01;   //1
  slot_index[1].rw    = SI_R;           //1
  slot_index[1].commu = SI_C1;
  slot_index[2].slot  = SLOT_01;        //1
  slot_index[2].index = S01_INDEX_02;   //2
  slot_index[2].rw    = SI_R;           //0
  slot_index[2].commu = SI_C2;

  slot_index[3].slot  = SLOT_01;        //1
  slot_index[3].index = S01_INDEX_03;   //3
  slot_index[3].rw    = SI_W;           //1
  slot_index[3].commu = SI_C12;

  slot_index[4].slot  = SLOT_01;        //1
  slot_index[4].index = S01_INDEX_04;   //4
  slot_index[4].rw    = SI_W;           //1
  slot_index[4].commu = SI_C12;

  slot_index[5].slot  = SLOT_01;        //1
  slot_index[5].index = S01_INDEX_05;   //5
  slot_index[5].rw    = SI_R;           //0
  slot_index[5].commu = SI_C12;

  slot_index[6].slot  = SLOT_01;        //1
  slot_index[6].index = S01_INDEX_06;   //6
  slot_index[6].rw    = SI_R;           //0
  slot_index[6].commu = SI_C12;

  slot_index[7].slot  = SLOT_01;        //1
  slot_index[7].index = S01_INDEX_07;   //7
  slot_index[7].rw    = SI_R;           //0
  slot_index[7].commu = SI_C12;

  slot_index[8].slot  = SLOT_01;        //1
  slot_index[8].index = S01_INDEX_08;   //8
  slot_index[8].rw    = SI_R;           //0
  slot_index[8].commu = SI_C12;

  slot_index[9].slot  = SLOT_01;        //1
  slot_index[9].index = S01_INDEX_09;   //9
  slot_index[9].rw    = SI_R;           //0
  slot_index[9].commu = SI_C12;

  slot_index[10].slot  = SLOT_01;        //1
```

```
  slot_index[10].index = SO1_INDEX_10;   //10
  slot_index[10].rw    = SI_R;           //0
  slot_index[10].commu = SI_C12;

 slot_index[11].slot  = SLOT_01;        //1
  slot_index[11].index = SO1_INDEX_11;   //11
  slot_index[11].rw    = SI_R;           //0
  slot_index[11].commu = SI_C12;
 //================================IM============================/
  slot_index[12].slot  = IM_SLOT;        //0
  slot_index[12].index = IM_INDEX;       //255
  slot_index[12].rw    = SI_RW;          //2
  slot_index[12].commu = SI_C12;
//================================================================/
  im_status=NO_IMCALL;
}
void deal_im_data(void)
{
  uint8_t i;
  for(i=0;i<64;i++){im0_body[i] = IM_input[i];}
  //return DATA_DONE;
}
```

（9）用户板硬件初始化

```
//==================用户板硬件初始化==============================//
void Y_HardWare_InitData(void)
{ uint8_t w,i;
 // while(dpram_hold()!=0){*LED_OUT1 = 0x81;}
    get_dpram();
    for(w=0;w==0;)
    //for(;;)
  {
        gdpram.AccessCounter_LRdpram += 1 ;
        if(gdpram.HardwareInit_Ldpram == 0xC5)
       {
         gdpram.R_status1  = 0;
          gdpram.R_status2  = 0;
          gdpram.R_command1 = 0;
          gdpram.R_command2 = 0;
          for(i=0;i<215;i++) {gdpram.InitData_DPv0[i] = 0;}//V0 初始化数据区
          for(i=0;i<245;i++) {gdpram.InData_DP[i]     = 0;}//PROFIBUS 输入数据区
          for(i=0;i<240;i++) {gdpram.DiagData_DP[i]   = 0;}//Diag 诊断数据区
          w=1;
            ABORT_DPRAM;
       }
      else
         {
          w=0;
          ABORT_DPRAM;
            delay_ms(5);
            get_dpram();
         }
        //ABORT_DPRAM;
  }
}
void Y_HardWare_InitData(void)
{ uint8_t w,i;
  get_dpram();
```

```
    //for(w=0;w==0;)
    for(;;)
     {
    if(dpram.Version_Code == 0x01) //0000
    if(dpram.Year_Code == 0x0D)//0001
    if(dpram.Month_Code == 0x07)//0002
    if(dpram.Day_Code == 0x01)//0003
    if(dpram.HardwareInit_Ldpram == 0xC5) //0004  标志接口板已完成自身硬件初始化
   {i=1;}
        }
}
```

（10）软件初始化内容

```
//==================软件初始化内容==============================//
void Y_SoftWare_InitData(void)
{ uint8_t w;
   for(w=0;w==0;)
   {
    get_dpram();                                        //获取控制权
    gdpram.AccessCounter_LRdpram +=1;
  gdpram.InitData_DPv0[0]=22;             //V0 初始化数据长度字节
    gdpram.InitData_DPv0[1]=dp_address();//*ADDRESS;    //Profibus 从站站地址
      //从站地址须灵活可设置，其范围：0～126，方式可为拨码（建议）或手持终端等
    gdpram.InitData_DPv0[2]=0x0C;     //ID 号高字节
    gdpram.InitData_DPv0[3]=0xC9;      //ID 号低字节
    //ID 号可临时使用，也可通过认证获取用户自己的 ID 号
    gdpram.InitData_DPv0[4]=0x0F;    //SPC3_Register_status0:0000 fS SYN FREEZE (SSA)
    gdpram.InitData_DPv0[5]=0;       //预留备用
    gdpram.InitData_DPv0[6]=0;        //预留备用
    gdpram.InitData_DPv0[7]=244; //48 //PROFIBUS 输入数据最大可能长度 a（244 字节）
    gdpram.InitData_DPv0[8]=244; //48 //PROFIBUS 输出数据最大可能长度 b（244 字节）
gdpram.InitData_DPv0[9]=238; //9 //用户诊断数据长度（≤238 不包含标准 6 字节诊断数据）
gdpram.InitData_DPv0[10]=MaxPrmLen;//MaxPrmLen237//用户参数数据最大可能长度 j（≤237 不包含标
准 7 字节参数数据）
 gdpram.InitData_DPv0[11]=NewPrmData_HandleMode;  //用户参数正确与否判断方式
//建议用户使用 M 卡判断用户参数是否正确，NewPrmData_HandleMode 等于 0 或等于 1
 gdpram.InitData_DPv0[12]=MaxCfgLen;//MaxCfgLen200//配置数据的最大可能长度 m（≤200）
 gdpram.InitData_DPv0[13]=NewCfgData_HandleMode;  //配置数据正确与否判断方式
//建议用户使用 M 卡判断配置数据是否正确,NewCfgData_HandleMode 等于 0 或等于 1
    dpram.InitData_DPv0[14]=6;                          //默认 CFG 数据长度 = n
    dpram.InitData_DPv0[15]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[16]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[17]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[18]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[19]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[20]=0x2F;                       //默认用户诊断数据长度
     //用户根据需求更改，其中“代码”见表 11
gdpram.CRC_H_DPv0=CRC_avr(gdpram.InitData_DPv0,gdpram.InitData_DPv0[0]+1)>>8;
      //i=dpram.CRC_H_DPv0;
    gdpram.CRC_L_DPv0=CRC_avr(gdpram.InitData_DPv0,gdpram.InitData_DPv0[0]+1);
    //j=dpram.CRC_L_DPv0;
    //CRC 高字节在前，低字节在后
    if(EnableDPv1_Flag == 1)
      {
       gdpram.EnableDPv1_Rdpram=0x1D;                   //V1 功能开启标志 B0005
       /*
       dpram.InitData_DPv1[0] = 20;                    //V1 初始化数据长度
```

```
        dpram.InitData_DPv1[1] = 2;                    //C2 通信的通道个数（≤3）
        dpram.InitData_DPv1[2] = 0;                    //C2 通信超时时间高字节
        dpram.InitData_DPv1[3] = 100;                  //C2 通信超时时间低字节
        dpram.InitData_DPv1[4] = 0;                    //槽-索引个数（x ≤ 12）
        dpram.InitData_DPv1[5] = 240;                  //每个槽-索引的最大数据长度( <=240 )

        dpram.InitData_DPv1[6] = 0;                    //槽-索引 1 信息:槽号
        dpram.InitData_DPv1[7] = 2;                    //槽-索引 1 信息:索引号
        dpram.InitData_DPv1[8] = 0;                    //槽-索引 1 信息:读写属性
        dpram.InitData_DPv1[9] = 2;                    //槽-索引 1 信息:C1/C2
        dpram.InitData_DPv1[10] = 10;                  //槽-索引 1 信息:数据长度

        dpram.InitData_DPv1[11] = 1;                   //槽-索引 2 信息:槽号
        dpram.InitData_DPv1[12] = 1;                   //槽-索引 2 信息:索引号
        dpram.InitData_DPv1[13] = 1;                   //槽-索引 2 信息:读写属性
        dpram.InitData_DPv1[14] = 2;                   //槽-索引 2 信息:C1/C2
        dpram.InitData_DPv1[15] = 10;                  //槽-索引 2 信息:数据长度


        dpram.CRC_H_DPv1 = CRC_avr(dpram.InitData_DPv1,dpram.InitData_DPv1[0]+1)>>8;
        dpram.CRC_L_DPv1 = CRC_avr(dpram.InitData_DPv1,dpram.InitData_DPv1[0]+1);
        */
       }
       else
       {gdpram.EnableDPv1_Rdpram=0x00;} //V1 功能开启标志 B0005
        gdpram.R_command1 |= 0x04;       //用户板命令字节 1_初始化信息有效标志
        ABORT_DPRAM;
      delay_ms(10);//此处需要等待接口板处理初始化结果，可能需要延时较长时间
      get_dpram();
      gdpram.AccessCounter_LRdpram +=1;
      if((gdpram.L_status1&0x01)==0x01)
     {
         w=1;
        ABORT_DPRAM;
      }
        else
              {
                  w=0;/* *LED_OUT2 = 0x99 ;*/
                  ABORT_DPRAM;
                  delay_ms(10);
              }
    } //for(w)
}
```

（11）DPV1 数据处理

```
/*=================Acyclic_data_Process====================*/
void Acyclic_data_C1(void)
{ uint16_t i;
    uint8_t w_slot,w_index,slot_curr;
    uint8_t len_status;//error_status
    i=0;
    w_slot=w_index=0;
    slot_curr=0;
    len_status=0;
    switch(gdpram.AcycData_DPv1c1[0]) //进来的是:0x5E 或 0x5F
   {
     case 0x5E :   //进来的是:0x5E Read
```

```
    {
       c_slot  = gdpram.AcycData_DPv1c1[1];
       c_index = gdpram.AcycData_DPv1c1[2];
       c_len   = gdpram.AcycData_DPv1c1[3];
        for(i=0;i<MAX_SI_NUM;i++)
          {//1
      if((slot_index[i].rw==SI_R)||(slot_index[i].rw==SI_RW))  //仅在具有读属性的槽-索引内寻
找
//2
                if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                {//3
                          slot_curr=0x01;//声明列表中出现过正确的槽号

                     if(c_index==slot_index[i].index)
                  {si_status=SI_OK;break;}
                     else
                     {
                           si_status=SI_NOK;
                             //error_status=SI_WRONG_INDEX;
                           //p2 = 1;
                           w_index=1;
                        }
                }
                         else
                         {
                              if(slot_curr==0)
                              {
                                si_status=SI_NOK;
                                w_slot=1;
                              }
                            else
                              {
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                         }//3
             }
                    else
                    {
                           if(slot_curr==0)
                             {
                               si_status=SI_NOK;
                               w_slot=1;
                             }
                           else
                             {
                               si_status=SI_NOK;
                               w_slot=0;
                             }
                    }//2

           }
           if(si_status == SI_OK)
           {
              if((c_len>240)||(c_len==0))
                {len_status = 1;}
              if(len_status == 0)
              {
```

```
                if(c_len>240)
                {
                 c_len=240;
                }
                  if((c_slot==IM_SLOT)&&(c_index==IM_INDEX)&&(c_len==0x44))//为 IM 槽索引
            {
                  if(im_status==RE_IMCALL)
                    {
                   deal_im_data(); //im0_body[i] = IM_read[i]
                   gdpram.AcycData_DPv1c2[0] = 0x5E;
                   gdpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                   gdpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                   gdpram.AcycData_DPv1c2[3] = c_len;  //本槽-索引数据长度
                   for(i=0;i<64;i++)
                  {
                    gdpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
                    gdpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
                    gdpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
                    gdpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
                   gdpram.AcycData_DPv1c2[i+8]=im0_body[i]; //把 CALL-DU 部分取出
                }
//for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=IM_input[i];}
gdpram.CRC_H_AcycData_DPv1c2 =
CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);
im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
si_status=0;
 }
else if(im_status==RES_IMCALL)
{
deal_im_data();
gdpram.AcycData_DPv1c2[0] = 0x5E;
gdpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
gdpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
 gdpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
for(i=0;i<64;i++)
 {
gdpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
 gdpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
 gdpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
gdpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
gdpram.AcycData_DPv1c2[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                        }
 gdpram.CRC_H_AcycData_DPv1c2 =
CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);

 im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
si_status=0;
}
else if(im_status==NO_IMCALL)
{
 gdpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_STATE);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
```

```
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
si_status = 0;
}
}
 else //为普通槽索引
{
 c_len=deal_acyclic_read_data(c_slot,c_index,c_len);
gdpram.AcycData_DPv1c2[0] = 0x5E;
gdpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
 gdpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
gdpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                    for(i=0;i<c_len;i++){gdpram.AcycData_DPv1c2[i+4]=v1_input[i];}
gdpram.CRC_H_AcycData_DPv1c2 =
CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);
  }
}
  else //长度错误
{
gdpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
 gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
 gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
si_status = 0;
 }
 }
 else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误
{
 gdpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
 gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
si_status = 0;
 }
else if((si_status == SI_NOK)&&(w_index==1))//索引错误
 {
gdpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
 gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
si_status = 0;
 }
}break;
case 0x5F :
{
 //c_rw    = SI_W;
c_slot  = gdpram.AcycData_DPv1c2[1];//slot
c_index = gdpram.AcycData_DPv1c2[2];//index
c_len   = gdpram.AcycData_DPv1c2[3];//length
```

```
for(i=0;i<MAX_SI_NUM;i++)
{//1
 if((slot_index[i].rw==SI_W)||(slot_index[i].rw==SI_RW))//仅在具有读属性的槽-索引内寻找
 {//2
 if(c_slot==slot_index[i].slot)  //槽号是否在列表中
{//3
slot_curr=0x01;//声明列表中出现过正确的槽号
if(c_index==slot_index[i].index)
{si_status=SI_OK;break;}
 else
{
si_status=SI_NOK;//error_status=SI_WRONG_INDEX; //p2=1;
 w_index=1;
 }
}
                          else
                          {
                               if(slot_curr==0)
                               {
                                 si_status=SI_NOK;
                                 w_slot=1;
                               }
                             else
                               {
                                 si_status=SI_NOK;
                                 w_slot=0;
                               }
                          }//3
                }
                       else
                       {
                              if(slot_curr==0)
                               {
                                 si_status=SI_NOK;
                                 w_slot=1;
                               }
                              else
                               {
                                 si_status=SI_NOK;
                                 w_slot=0;
                               }
                       }//2
              }//1
              if(si_status == SI_OK)
              {
                 if((c_len>240)||(c_len==0))
                   {len_status = 1;}
                 if(len_status == 0)
                 {
                     if((c_slot==IM_SLOT)&&(c_index==IM_INDEX))
                     {// 5F 00 FF 04 08 00 FD E8
                      if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                          {
                             if((gdpram.AcycData_DPv1c2[3]==4)
&&(gdpram.AcycData_DPv1c2[4]==8)&&(gdpram.AcycData_DPv1c2[5]==0)
&&(gdpram.AcycData_DPv1c2[6]==0xFD)&&(gdpram.AcycData_DPv1c2[7]==0xE8))
                        {  //下面 4 条可不要
                          gdpram.AcycData_DPv1c2[0] = 0x5F;
```

```
                        gdpram.AcycData_DPv1c2[1] = c_slot;
                        gdpram.AcycData_DPv1c2[2] = c_index;
                        gdpram.AcycData_DPv1c2[3] = c_len;

                        IM_CallHeader[0] = gdpram.AcycData_DPv1c2[4];// 0x08
                        IM_CallHeader[1] = gdpram.AcycData_DPv1c2[5];// 0x00
                        IM_CallHeader[2] = gdpram.AcycData_DPv1c2[6];// 0xFD
IM_CallHeader[3] = gdpram.AcycData_DPv1c2[7];// 0xE8
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
 si_status=0;
 }
else //application  & feature not supported (不是 65000) B6
{
gdpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
 gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
im_status=NO_IMCALL;
si_status=0;
}
}//if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
else //(im_status==RE_IMCALL)
{
  if((gdpram.AcycData_DPv1c2[3]==4)
&&(gdpram.AcycData_DPv1c2[4]==8)&&(gdpram.AcycData_DPv1c2[5]==0)
&&(gdpram.AcycData_DPv1c2[6]==0xFD)&&(gdpram.AcycData_DPv1c2[7]==0xE8))
{  //下面 4 条可不要
gdpram.AcycData_DPv1c2[0] = 0x5F;
gdpram.AcycData_DPv1c2[1] = c_slot;
gdpram.AcycData_DPv1c2[2] = c_index;
gdpram.AcycData_DPv1c2[3] = c_len;
IM_CallHeader[0] = gdpram.AcycData_DPv1c2[4];// 0x08
IM_CallHeader[1] = gdpram.AcycData_DPv1c2[5];// 0x00
IM_CallHeader[2] = gdpram.AcycData_DPv1c2[6];// 0xFD
IM_CallHeader[3] = gdpram.AcycData_DPv1c2[7];// 0xE8
 gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
 gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
 im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
 si_status=0;
 // im_status=READ_IMCALL;
 }
  else //application  & feature not supported (不是 65000) B6
 {
 gdpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
 gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
 gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
 im_status=NO_IMCALL;
 si_status=0;
}
```

```
}
}
  else
 {
gdpram.AcycData_DPv1c2[0] = 0x5F;
 gdpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
 gdpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
gdpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
for(i=0;i<c_len;i++){v1_output[i]=gdpram.AcycData_DPv1c2[i+4];}
deal_acyclic_write_data(c_slot,c_index,c_len);
 gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
 gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
                    }
                }
                  else //长度错误  B7
                  {
           gdpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
           gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
           gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
           gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
                   gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
 gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
                  si_status = 0;
                }
             }
             else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误  B6
             {
         gdpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
         DPSE_ERRCL_ACC_INV_SLOT); //Error_Code_1 :invalid slot
         gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
        gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
               si_status = 0;
             }
             else if((si_status == SI_NOK)&&(w_index==1))//索引错误 B0
             {
         gdpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
         gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
        gdpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
        gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
        gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
               si_status = 0;
             }
      }break;
        case 0x51 :
          {
                c_slot  = gdpram.AcycData_DPv1c2[1];//slot
                c_index = gdpram.AcycData_DPv1c2[2];//index
                c_len   = gdpram.AcycData_DPv1c2[3];//length
                if((c_slot==SLOT_03)&&(c_index==S03_INDEX_01))           //slot_index=[4,1]
                {
               gdpram.AcycData_DPv1c2[0] = 0x51;
               gdpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
```

```
                gdpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                gdpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度

                for(i=0;i<c_len;i++){gdpram.AcycData_DPv1c2[i+4]=i;}
                gdpram.CRC_H_AcycData_DPv1c2 =
CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
                gdpram.CRC_L_AcycData_DPv1c2 =
CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);
              }
                else
                {
                 gdpram.AcycData_DPv1c2[0] |= 0xD1; //Function Code
                 gdpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                 gdpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_APPLICATION <<4)|
DPSE_ERRCL_APP_NOTSUPP); //Error_Code_1 :invalid slot
                 gdpram.AcycData_DPv1c2[3]  = 0x51; //Error_Code_2
        gdpram.CRC_H_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
         gdpram.CRC_L_AcycData_DPv1c2 = CRC_avr(gdpram.AcycData_DPv1c2,4);
                }
           }break;
           default : break;
  }
}
```

（12）外部诊断处理

```
/*================Acyclic_data_Process=======================*/

void DP_DSDPRAM_DiagHandler(void)
{
        uint8_t i;
  //=================Ext_Diag 处理====================================//
        if(((ExtFlag&0x0F)==1)&&((StaFlag&0x0F)==0))//Ext.diag 高优先权外部诊断，从站有错误
        {
                if(flag_diag == 0)
                {
                    gdpram.R_command1 |= 0x02;
                    gdpram.DiagData_DP[0] = 0x81;
                  current_diag_len=gdpram.InitData_DPv0[9]-6;
for(i=0;i<gdpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF1
                    Tn=1;
                  flag_diag = 1;
                }

          }
else if(((ExtFlag&0x0F)==0)&&((StaFlag&0x0F)==0)&&(Tn==1))//Ext.diag 高优先权外部诊断，从站无
错误
        {
                if(flag_diag == 1)
                {
                  gdpram.R_command1 |= 0x02;
                    gdpram.DiagData_DP[0] = 0x01;
                    current_diag_len=gdpram.InitData_DPv0[9]-6;
    for(i=0;i<gdpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF0
                    Tn=0;
                    flag_diag = 0;
              }
            }
else if(((StaFlag&0x0F)==1)&&((ExtFlag&0x0F)==0))    //Stat.diag  高优先权静态诊断，从站有错误
```

```
        {
                if(flag_diag == 0)
                {
                  gdpram.R_command1 |= 0x02;
                    gdpram.DiagData_DP[0] = 0x82;
                  current_diag_len=gdpram.InitData_DPv0[9]-6;
for(i=0;i<gdpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF2
                    Tn1=1;
                  flag_diag = 1;
              }
            }
else if(((StaFlag&0x0F)==0)&&((ExtFlag&0x0F)==0)&&(Tn1==1))//Stat.diag 高优先权静态诊断，从站无错误
        {
                if(flag_diag == 1)
                {
                  gdpram.R_command1 |= 0x02;
                    gdpram.DiagData_DP[0] = 0x02;
                    current_diag_len=gdpram.InitData_DPv0[9]-6;
    for(i=0;i<gdpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF0
                    Tn1=0;
                    flag_diag = 0;
              }
            }
    if((gdpram.R_command1&0x02) == 0x02)
      {
        gdpram.DiagData_DP[1] = current_diag_len;
        for(i=0;i<gdpram.DiagData_DP[1];i++)
           {gdpram.DiagData_DP[i+2] = diag_input[i];}
    gdpram.CRC_H_DiagData_DP=CRC_avr(gdpram.DiagData_DP,gdpram.DiagData_DP[1]+2)>>8;
gdpram.CRC_L_DiagData_DP=CRC_avr(gdpram.DiagData_DP,gdpram.DiagData_DP[1]+2);
     }
}
```

（13）DPV0 数据交换

```
//=====================================================================//
void DPV0V1_DataEx_DSdpramMode(void) //IO 数据缓冲区与双口 RAM 之间的数据交换
{
    uint8_t i;
   uint8_t Rdpram_CRC_H=0 , Rdpram_CRC_L=0;
   get_dpram();
   gdpram.AccessCounter_LRdpram +=1;
   dsdpv1_status=(gdpram.L_status3&0x0C);
    if(dsdpv1_status==0x0c) //DP 进入数据交换开始 v0 数据处理
    {
   //=========================DP_OUT 处理=============================//
   if( (gdpram.L_command1&0x01) == 0x01 )
    {
      gdpram.L_command1 &= 0xFE;
      Rdpram_CRC_H=CRC_avr(gdpram.OutData_DP,gdpram.OutData_DP[0]+1)>>8;
      Rdpram_CRC_L=CRC_avr(gdpram.OutData_DP,gdpram.OutData_DP[0]+1);
if((Rdpram_CRC_H==gdpram.CRC_H_OutData_DP)&&(Rdpram_CRC_L==gdpram.CRC_L_OutData_DP))
        {
          gdpram.R_status2 &= 0xFB;//B000D.2 从接口板获得的输出数据 CRC_avr 正确
          gdpram.OutData_DP[0]=gdpram.CfgData_DP[0];
          for(i=0;i<gdpram.OutData_DP[0];i++)
          {v0_output[i] = gdpram.OutData_DP[i+1];}
            }
```

```
          else
          gdpram.R_status2 |= 0x04;//B000D.2 从接口板获得的输出数据 CRC_avr 错误
        }
  //=========================DP_IN 处理=============================//
     gdpram.InData_DP[0]=gdpram.CfgData_DP[1];
      for(i=0;i<gdpram.InData_DP[0];i++)
      {
        gdpram.InData_DP[i+1] = v0_input[i];

      }
gdpram.CRC_H_InData_DP=CRC_avr(gdpram.InData_DP,gdpram.InData_DP[0]+1)>>8;
      gdpram.CRC_L_InData_DP=CRC_avr(gdpram.InData_DP,gdpram.InData_DP[0]+1);
      gdpram.R_command1 |= 0x01;
//=================诊断处理=================================//
          DP_DSDPRAM_DiagHandler();
//==========================================================//
    }
```

（14）DPV1/C1 数据交换

```
  //=========================Acyclic_data_C1 处理=====================//
    if( (gdpram.L_command2&0x01) == 0x01 )
     {
         gdpram.L_command2 &= 0xFE;//
      if(gdpram.AcycData_DPv1c1[0]==0x5E)
      {
         Rdpram_CRC_H=CRC_avr(gdpram.AcycData_DPv1c1,4)>>8;
         Rdpram_CRC_L=CRC_avr(gdpram.AcycData_DPv1c1,4);
      }
      if(gdpram.AcycData_DPv1c1[0]==0x5F)
      {
Rdpram_CRC_H=CRC_avr(gdpram.AcycData_DPv1c1,gdpram.AcycData_DPv1c1[3]+4)>>8;
Rdpram_CRC_L=CRC_avr(gdpram.AcycData_DPv1c1,gdpram.AcycData_DPv1c1[3]+4);
      }
if((Rdpram_CRC_H==gdpram.CRC_H_AcycData_DPv1c1)&&(Rdpram_CRC_L==gdpram.CRC_L_AcycData_DPv1c1)
)
    {
          gdpram.R_status2 &= 0xEF;
          Acyclic_data_C1();
          gdpram.R_command2 |= 0x01;
        }
      else
    {gdpram.R_status2 |= 0x10;} //写 1 到 B000D.4;从接口板获得的非循环数据 CRC 错
     }
```

（15）DPV1/C2 数据交换

```
  //=========================Acyclic_data_C2 处理=====================//
   if( (gdpram.L_command2&0x02) == 0x02 )
     {
       gdpram.L_command2 &= 0xFD;//
      if(gdpram.AcycData_DPv1c2[0]==0x5E)
      {
         Rdpram_CRC_H=CRC_avr(gdpram.AcycData_DPv1c2,4)>>8;
         Rdpram_CRC_L=CRC_avr(gdpram.AcycData_DPv1c2,4);
      }
      if(gdpram.AcycData_DPv1c2[0]==0x5F)
      {
Rdpram_CRC_H=CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
```

```
Rdpram_CRC_L=CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);
      }
      if(gdpram.AcycData_DPv1c2[0]==0x51)
        {
           Rdpram_CRC_H=CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4)>>8;
Rdpram_CRC_L=CRC_avr(gdpram.AcycData_DPv1c2,gdpram.AcycData_DPv1c2[3]+4);
}
if((Rdpram_CRC_H==gdpram.CRC_H_AcycData_DPv1c2)&&(Rdpram_CRC_L==gdpram.CRC_L_AcycData_DPv1c2)
)
    {
          gdpram.R_status2 &= 0xF7;
          Acyclic_data_C2();
          gdpram.R_command2 |= 0x02;
         i=0;
        }
      else
      {gdpram.R_status2 |= 0x08;} // 写 1 到 B000D.3;从接口板获得的 IM 数据 CRC 错
     }
    ABORT_DPRAM;
}
void Y_Dpram_IntR_EXIT_config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOC,GPIO_PinSource0);
    EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt ;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling ;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
void DPRAMMode_GPIO_Init(void)
{
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO|RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB
2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG, ENABLE);
//------------R_REQ && R_STA-----------------------------------//
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);//用户板握手控制信号(输出)
//-----------L_REQ && L_STA-----------------------------------------//
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOC, &GPIO_InitStructure);//接口板握手控制信号(输入)
}
void Shared_GPIO_Init(void)
{   GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO|RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB
2Periph_GPIOC|RCC_APB2Periph_GPIOD
|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG, ENABLE);
```

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_WWDG, ENABLE);
```

（16）DPV1/C1 数据交换

```
//------------Interactive Mode0 && Interactive Mode1-------------------//
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10|GPIO_Pin_11;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOC, &GPIO_InitStructure);//
//-----------多芯片从站地址设置(SPI /UART)-----------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_0|GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IPD;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
//----------- PRM_CFG_INIT-----------------------------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9|GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
//------------复位 FPGA-------------------------------------------------//
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5 ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOE, &GPIO_InitStructure);//用于 复位 FPGA
//------------ PRM_CFG_INIT-----------------------------------------------//
//------------ INTR(M1_DS_nINTR)-----------------------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IPD;
  GPIO_Init(GPIOG, &GPIO_InitStructure);// INTR(M1_DS_nINTR)
//------------ INTR(M2_DS_nINTR)-----------------------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_12;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IPD;
  GPIO_Init(GPIOD, &GPIO_InitStructure);// INTR(M2_DS_nINTR)
//=================SPI/Uart 地址=======================================//
    SPISlave_Addr=SPISlave_Addr1;
  UARTMode_ADDR1=SPISlave_Addr;
}
```

（17）DPRAM 初始化

```
void DPRAM_Init(void)
{
  FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
  FSMC_NORSRAMTimingInitTypeDef  p;
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE | RCC_APB2Periph_GPIOF
|RCC_APB2Periph_GPIOG, ENABLE); RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);
/*-- GPIO Configuration ------------------------------------*/
  /*!< SRAM Data lines configuration */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //复用推挽输出
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    /*-------- PD14 PD15 PD0 PD1 <-> D0 D1 D2 D3 */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_14 | GPIO_Pin_15 ;
 GPIO_Init(GPIOD, &GPIO_InitStructure);
  /*-------- PE7 PE8 PE9 PE10 <-> D4 D5 D6 D7  */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10;
```

```
 GPIO_Init(GPIOE, &GPIO_InitStructure);
  /*!< SRAM Address lines configuration */ //A0-A9
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4
| GPIO_Pin_5 | GPIO_Pin_12 | GPIO_Pin_13 |  GPIO_Pin_14 | GPIO_Pin_15 GPIO_Init(GPIOF,
&GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;//A10
  GPIO_Init(GPIOG, &GPIO_InitStructure);
  /*!< NOE and NWE configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 |GPIO_Pin_5;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
  /*!< NE1 configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
  /*!< NBL0, NBL1 configuration */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
/*-- FSMC Configuration ------------------------------------------------*/
  p.FSMC_AddressSetupTime = 0;
  p.FSMC_AddressHoldTime = 0; //14ns
  p.FSMC_DataSetupTime = 12;  //数据建立时间（1 个时钟周期：14ns ）
  p.FSMC_DataLatency = 0;     //数据保持时间   （同步存储器使用）
  p.FSMC_BusTurnAroundDuration = 0; //总线恢复时间
  p.FSMC_CLKDivision = 0;           //时钟分频
  p.FSMC_AccessMode = FSMC_AccessMode_A;//访问模式，其中 FSMC_AccessMode(访问模式)成员的设置只
在开启了扩展模式才有效，而且开启了扩展模式后，读时序和写时序的设置可以是独立的
FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM1;//选择设置的 BANK 及片选信号
 FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;//设置是否数据地
址总线分时复用
 FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_SRAM; //设置存储器类型
FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_8b;//设置数据宽度
  FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode = FSMC_BurstAccessMode_Disable;//设置是否使
用迸发访问模式(应该就是连续读写模式 )
  FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait = FSMC_AsynchronousWait_Disable;//设定是否
使用异步等待信号
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;//设置 WAIT
信号的有效电平
  FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;  //配置是否使用非对齐方式
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState;//
配置等待信号什么时期产生
  FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;//设定是否使能写
操作
  FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;//设定是否使用 WAIT 信号
  FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Enable;//设定是否使用单独的
写时序
  FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable; //设定是否使用突发写模
式
 FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;       //设定读写时序
  FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;
  FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure); //配置参数写到控制寄存器
  /*!< Enable FSMC Bank1_SRAM Bank */
  FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);//调用 FSMC_NORSRAMCmd()使能 BANK1
}
/**
  * @brief  Writes a Half-word buffer to the FSMC SRAM memory.
  * @param  pBuffer : pointer to buffer.
  * @param  WriteAddr : SRAM memory internal address from which the data will be
  *         written.
```

```
  * @param  NumHalfwordToWrite : number of half-words to write.
  * @retval None*/
```

（18）写操作

```
void SRAM_WriteBuffer(uint16_t* pBuffer, uint32_t WriteAddr, uint32_t NumHalfwordToWrite)
{
  for(; NumHalfwordToWrite != 0; NumHalfwordToWrite--) /*!< while there is data to write */
  {
    /*!< Transfer data to the memory */
    *(uint16_t *) (Bank1_SRAM3_ADDR + WriteAddr) = *pBuffer++;
    /*!< Increment the address*/
    WriteAddr += 2;
  }
}
/**
  * @brief  Reads a block of data from the FSMC SRAM memory.
  * @param  pBuffer : pointer to the buffer that receives the data read from the
  *         SRAM memory.
  * @param  ReadAddr : SRAM memory internal address to read from.
  * @param  NumHalfwordToRead : number of half-words to read.
  * @retval None  */
```

（19）读操作

```
void SRAM_ReadBuffer(uint16_t* pBuffer, uint32_t ReadAddr, uint32_t NumHalfwordToRead)
{
  for(; NumHalfwordToRead != 0; NumHalfwordToRead--) /*!< while there is data to read */
  {
    /*!< Read a half-word from the memory */
    *pBuffer++ = *(__IO uint16_t*) (Bank1_SRAM3_ADDR + ReadAddr);
    /*!< Increment the address*/
    ReadAddr += 2;
  }
}
```

（20）LED 指示灯控制

```
void LED(void)
{
    uint16_t input_led=0;
  //uint16_t dp_addr=0;
    //uint16_t Output_value=0;
    uint32_t i;
        GPIOG->ODR = 0xFFFE;  //1111 1111 1111 1110
        for(i=0;i<300000;i++);
        */
        GPIOB->ODR = 0xFFFF;  //1111 1111 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFFFD;  //1111 1111 1111 1101
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFFFB;  //1111 1111 1111 1011
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFFF7;  //1111 1111 1111 0111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFFEF;  //1111 1111 1110 1111
        for(i=0;i<300000;i++);
      GPIOG->ODR = 0xFFDF;  //1111 1111 1101 1111
        for(i=0;i<300000;i++);
```

```
        GPIOG->ODR = 0xFFBF;  //1111 1111 1011 1111
        for(i=0;i<300000;i++);

            GPIOG->ODR = 0xFF7F;  //1111 1111 0111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFEFF;  //1111 1110 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFDFF;  //1111 1101 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xFBFF;  //1111 1011 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xF7FF;  //1111 0111 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xEFFF;  //1110 1111 1111 1111
        for(i=0;i<300000;i++);
         GPIOG->ODR = 0xDFFF;  //1101 1111 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0xBFFF;  //1011 1111 1111 1111
        for(i=0;i<300000;i++);
        GPIOG->ODR = 0x7FFF;  //0111 1111 1111 1111
        for(i=0;i<300000;i++);
         GPIOG->ODR = 0xFFFF;  //1111 1111 1111 1110
        for(i=0;i<300000;i++);
         GPIOB->ODR = 0xFFBF;  //1111 1111 1011 1111
        for(i=0;i<300000;i++);
        GPIOB->ODR = 0xFFFF;  //1111 1111 1111 1111
        for(i=0;i<300000;i++);
        input_led       = GPIO_ReadInputData(GPIOE);
        input_led       = input_led;
}


uint16_t in_led(void)
{  uint16_t in_led0,in_led1,in_led2,in_led3,in_led4,in_led5,in_led6,in_led7;
   uint16_t in_led8,in_led9,in_led10,in_led11,in_led12,in_led13,in_led14,in_led15;
     uint16_t led_in;
     in_led0=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_0); //DI00
     in_led1=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_1)<<1; //DI01
     in_led2=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)<<2; //DI02
     in_led3=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_3)<<3; //DI03
     in_led4=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_4)<<4; //DI04
     in_led5=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_5)<<5; //DI05
     in_led6=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_6)<<6; //DI06
     in_led7=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_11)<<7;//DI07
     in_led8=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_12)<<8;//DI08
     in_led9=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_13)<<9;//DI09
     in_led10=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_14)<<10;//DI010
     in_led11=GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_15)<<11;//DI011
     in_led12=GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10)<<12; //DI012
     in_led13=GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_11)<<13; //DI013
     in_led14=GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_12)<<14; //DI014
     in_led15=GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_13)<<15; //DI015
 led_in = in_led15|in_led14|in_led13|in_led12|in_led11|in_led10|in_led9|in_led8
|in_led7 |in_led6 |in_led5 |in_led4 |in_led3 |in_led2 |in_led1|in_led0 ;
return led_in ;
}
void out_led(uint32_t val)
{ //PGout(1)= (val&0x0001)>>0 ;
```

```
    GPIO_WriteBit(GPIOG,GPIO_Pin_1,(BitAction)((val&0x0001)>>0));  //out_led :0
    GPIO_WriteBit(GPIOG,GPIO_Pin_2,(BitAction)((val&0x0002)>>1));  //out_led :1
    GPIO_WriteBit(GPIOG,GPIO_Pin_3,(BitAction)((val&0x0004)>>2));  //out_led :2
    GPIO_WriteBit(GPIOG,GPIO_Pin_4,(BitAction)((val&0x0008)>>3));  //out_led :3
    GPIO_WriteBit(GPIOG,GPIO_Pin_5,(BitAction)((val&0x0010)>>4));  //out_led :4
    GPIO_WriteBit(GPIOG,GPIO_Pin_6,(BitAction)((val&0x0020)>>5));  //out_led :5
    GPIO_WriteBit(GPIOG,GPIO_Pin_7,(BitAction)((val&0x0040)>>6));  //out_led :6
    GPIO_WriteBit(GPIOG,GPIO_Pin_8,(BitAction)((val&0x0080)>>7));  //out_led :7
    GPIO_WriteBit(GPIOG,GPIO_Pin_9, (BitAction)((val&0x0100)>>8));  //out_led :8
    GPIO_WriteBit(GPIOG,GPIO_Pin_10,(BitAction)((val&0x0200)>>9)); //out_led :9
GPIO_WriteBit(GPIOG,GPIO_Pin_11,(BitAction)((val&0x0400)>>10));//out_led :10
GPIO_WriteBit(GPIOG,GPIO_Pin_12,(BitAction)((val&0x0800)>>11)); /out_led :11
GPIO_WriteBit(GPIOG,GPIO_Pin_13,(BitAction)((val&0x1000)>>12)); /out_led :12
GPIO_WriteBit(GPIOG,GPIO_Pin_14,(BitAction)((val&0x2000)>>13));//out_led :13
GPIO_WriteBit(GPIOG,GPIO_Pin_15,(BitAction)((val&0x4000)>>14));//out_led :14
GPIO_WriteBit(GPIOB,GPIO_Pin_6, (BitAction)((val&0x8000)>>15));//out_led :15
}
void UserApplication_IoBuf_Dealwith(void)
 { uint32_t output;
  /*---------------- Led_out------------------- */
     output = (v0_output[1]<<8)|v0_output[0];
     out_led(~output);
    /*---------------- Led_in------------------- */
     v0_input[1] = in_led()>>8;
     v0_input[0] = in_led();
 }
```

（21）处理 DPV1 读数据

```
uint8_t deal_acyclic_read_data(uint8_t slot, uint8_t index, uint8_t len)
{
  uint8_t i;
  //处理 DPV1 读数据
  //从 DPRAM 中获得 DPV1 本槽-索引的数据
  if((slot==SLOT_00)&&(index==S00_INDEX_01))
  {
    //SLOT: 0   INDEX: 1  DP 地址
    for(i=0;i<1;i++)
    {
        v1_input[i]=dp_address();
    }
      len = 1;
  }
  if((slot==SLOT_01)&&(index==S01_INDEX_01))
  {
        //SLOT: 1   INDEX: 1   非循环通信页 Input
        for(i=0;i<2;i++)
        {
                v1_input[i++] =v1c1_input1;
                v1_input[i++] =v1c1_input2;
         }
         len = 2;
    }
  if((slot==SLOT_01)&&(index==S01_INDEX_02))
  {
        //SLOT: 1   INDEX: 2   非循环通信页 Input
        for(i=0;i<2;i++)
        {
```

```
                v1_input[i++] =v1c2_input1;
                v1_input[i++] =v1c2_input2;
            }
           len = 240;
           //len = 2;
    }
  if((slot==SLOT_01)&&(index==S01_INDEX_05))
  {
        //SLOT: 1   INDEX: 2   非循环通信页 Input
        for(i=0;i<len;i++)
        {
                v1_input[i++] =v1c2_input1;
                v1_input[i++] =v1c2_input2;
            }
           //len = 3;
    }
//-------------------------------------------------------//
  if((slot==SLOT_01)&&(index==S01_INDEX_02))
  {
        //SLOT: 1   INDEX: 2   非循环通信页 Input
        for(i=0;i<20;i++)
        {
            v1_input[i] = v1_output[i];
            }
           len = 20;
    }
//-------------------------------------------------------//
    if((slot==SLOT_02)&&(index==S02_INDEX_02))
  {
        //SLOT: 2   INDEX: 2   非循环通信页 Input
        for(i=0;i<64;i++)
        {
            v1_input[i] = IM_input[i];
            }

           len = 64;
    }
  return len;
}
void deal_acyclic_write_data(uint8_t slot, uint8_t index, uint8_t len)
{
    uint8_t i;
    //uint32_t output;
    if((slot==SLOT_01)&&(index==S01_INDEX_03))
  {
        for(i=0;i<2;i++)
        {
            v1c1_output1 = v1_output[0];
            v1c1_output2 = v1_output[1];
        }
    }
    if((slot==SLOT_01)&&(index==S01_INDEX_04))
  {
        for(i=0;i<2;i++)
        {
            //output = (v1_output[1]<<8)|v1_output[0];
            v1c2_output1 = v1_output[0];
            v1c2_output2 = v1_output[1];
```

```
        }
    }

    if((slot==SLOT_03)&&(index==S03_INDEX_01))
  {
        for(i=0;i<2;i++)
        {
            //output = (v1_output[1]<<8)|v1_output[0];
            v1c2_output1 = v1_output[0];
            v1c2_output2 = v1_output[1];
        }
    }
}
```

（22）用户参数及配置数据处理

```
void EXTI9_5_IRQHandler(void)
{
  uint8_t i;
  uint8_t Rdpram_CRC_H=0 , Rdpram_CRC_L=0;
 if(EXTI_GetITStatus(EXTI_Line6) != RESET)
 {
    get_dpram();
  gdpram.AccessCounter_LRdpram += 1;
  if((gdpram.L_command1&0x02) == 0x02) //接口板用户参数数据有效标志       B000E.1
    {
        Rdpram_CRC_H = CRC_avr(gdpram.PrmData_DP,gdpram.PrmData_DP[0]+1)>>8;
      Rdpram_CRC_L = CRC_avr(gdpram.PrmData_DP,gdpram.PrmData_DP[0]+1); //CRC 计算消耗
1.093ms(比本地算 CRC 多 0.127ms)
        if((Rdpram_CRC_H == gdpram.CRC_H_PrmData_DP)&&(Rdpram_CRC_L ==
gdpram.CRC_L_PrmData_DP))
        {
           gdpram.R_status2 &= 0xFD ; //从接口板获得的用户参数数据 CRC 正确 B000D.1 =0
           New_Prmdata_Len = gdpram.PrmData_DP[0];

           for(i=0;i<gdpram.PrmData_DP[0];i++){New_Prmdata_Buf[i] =
gdpram.PrmData_DP[i+1];}//0.477ms(比本地拿多 0.124ms)
if(New_Prmdata_Len<=MaxPrmLen)  //新参数化数据长度 小于等于 初始化最大长度 237
{gdpram.R_status1 &= 0xFE;}   //用户判断用户参数数据正确     B000C.0 =0
else
{gdpram.R_status1 |= 0x01;}   //用户判断用户参数数据错误     B000C.0 =1
          }
        else
        {
 gdpram.R_status2 |= 0x02 ; //从接口板获得的用户参数数据 CRC 错误    B000D.1 =1
    gdpram.R_status1 |= 0x01 ; //用户判断用户参数数据错误              B000C.0 =1
          }
 gdpram.R_command1 |= 0x80;  //设置用户板用户参数判断结果回传标志    B0010.7 =1
                             //用户板已将用户参数判断结果存入        B000C.0
gdpram.L_command1 &= 0xFD ; //清除接口板用户参数数据有效标志        B000E.1 =0
      }
  else
    { //异常处理
      }
  if((gdpram.L_command1&0x04) == 0x04) //接口板配置数据有效标志      B000E.2
    {
      Rdpram_CRC_H = CRC_avr(gdpram.CfgData_DP,gdpram.CfgData_DP[2]+3)>>8;
      Rdpram_CRC_L = CRC_avr(gdpram.CfgData_DP,gdpram.CfgData_DP[2]+3);
```

```
       if((Rdpram_CRC_H == gdpram.CRC_H_CfgData_DP)&&(Rdpram_CRC_L ==
gdpram.CRC_L_CfgData_DP))
        {
gdpram.R_status2 &= 0xFE ;       //从接口板获得的配置数据 CRC 正确     B000D.0 =0
           New_Cfgdata_Len = gdpram.CfgData_DP[2];
           for(i=0;i<gdpram.CfgData_DP[2];i++){New_Cfgdata_Buf[i] = gdpram.CfgData_DP[i+3];}
if(New_Cfgdata_Len <= MaxCfgLen)//此处，可以根据用户实际需求情况，另外增加判断条件 200
{gdpram.R_status1 &= 0xFD ;}      //用户判断配置数据正确         B000C.1 =0
            else
{gdpram.R_status1 |= 0x02 ;}      //用户判断配置数据错误         B000C.1 =1
          }
        else
        {
 gdpram.R_status2 |= 0x01 ; //从接口板获得的配置数据 CRC 错误      B000D.0 =1
gdpram.R_status1 |= 0x02 ;        //用户判断配置数据错误         B000C.1 =1
          }
gdpram.R_command1 |= 0x40;  //设置用户板配置数据判断结果回传标志    B0010.6 =1
//用户板已将配置数据判断结果存入        B000C.1
 gdpram.L_command1 &= 0xFB ; //清除接口板配置数据有效标志            B000E.2 =0
      }
  else
    {
        //异常处理
      }    //写中断信箱 0x07FE，触发 INTL 中断
    ABORT_DPRAM;
        EXTI_ClearITPendingBit(EXTI_Line6);
  }

}
```

第六章 SPI 接口

一、MCU 和 M 卡 SPI 串口引脚定义

MCU 和 M 卡之间的 SPI 通讯，MCU 为 SPI master（主动询问），M 卡为 SPI slave（被动回答）,相关引脚信号定义如下：

命名	定义
XSS	从设备选通信号
SCK	数据读写同步时钟
MOSI	用户 MCU 数据输入 M 卡数据输出
MISO	用户 MCU 数据输出 M 卡数据输入

二、通讯协议-SPI 接口模式下的报文格式

起始符	读写首地址	读写数据长度	DPRAM 写数据	CRC 校验和
1 字节	2 字节	1 字节	1~245 字节	2 字节

注：整个报文帧最大为 251 字节，大于此报文长度的报文帧为不合法报文帧

1．起始符定义：（包括从站目标地址和读写属性）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
000:CPU 读(R) 001:CPU 写(W) 010:CPU 读写(RW) 100:读正确(CR) 101：写正确(CW) 110:读写正确(CRW) 111:出现错误(ER)			保留 默认为 0	保留 默认为 0	保留 默认为 0	主从目标地址： 00：目标站地址=用户 CPU； 01：目标站地址=1 号 M 卡； 10：目标站地址=2 号 M 卡；	

2．读写地址定义

读写 DPRAM 地址															
高字节								低字节							
Bit 7	Bit 6	Bit 5	bit4	bit 3	bit 2	bit1	bit 0	bit7	bit6	bit5	bit4	bit 3	bit 2	bit 1	bit0
保 留					DPRAM 访问首地址高 3 位			DPRAM 访问首地址低 8 位							

3．数据长度定义

长度为 1 字节，表示读写 DPRAM 单元对应地址单元数据长度。

4．数据的定义

初始化数据，诊断数据，用户参数数据、CFG 数据、V1C1/C2 非循环数据以及 PROFIBUS 输入输出数据，详见“第二章/DPRAM 数据结构”。

5．报文 CRC 校验和

从[起始符]到[数据]字段的所有数据字节的 CRC 校验和(多项式取值：0xA001)

三、用户 MCU 与 M 卡交互报文

用户 MCU 与 M 卡之间交互报文包括三类：只读报文、只写报文以及可读可写报文。

1．用户 MCU 向 M 卡发送只读数据报文：

起始符	读地址	读数据长度	CRC 校验和
000000xx	2 字节	1 字节	2 字节

注：长度为 6 字节固定

M 卡向 MCU 发送响应报文：

起始符	错误码	DPRAM 数据	CRC 校验和
100000xx	1 字节	1~245 字节	2 字节

注：当用户 MCU 读报文出现错误时，此处的起始符为 bit7~bit5 为 111，错误码详见后文，此报文长度可变，但最大报文长度不会超过 251 字节。

2．用户 MCU 向 M 卡发送只写数据报文：

起始符	写地址	写数据长度	DPRAM 写数据	CRC 校验和
001000xx	2 字节	1 字节	1~245 字节	2 字节

注： 报文长度可变 当用户发送写报文后至少间隔 2.7us 才能再发下一帧报文

M 卡向 MCU 发送的响应报文格式：

起始符	错误码	CRC 校验和
101000xx	1 字节	2 字节

注：当用户 MCU 写报文出现错误时，此处的起始符为 bit7~bit5 为 111，错误码详见后文， 报文长度固定，为 4 字节。

3．用户 MCU 向 M 卡发送可读可写数据报文：

起始符	DPRAM 写数据	CRC 校验和
010000xx	1~245 字节	2 字节

备注：

- 此报文只适用于输入数据和输出数据的交换，当用户 MCU 写输入数据到 M 卡后，M 卡向 MCU 回复输出数据。
- 当用户 MCU 发送可读可写报文后，至少间隔 2.7us 才能再发送下一帧报文。
- 用户不能利用此报文写除输入数据单元外的其它数据单元，否则输入数据错误，但 M 卡将可以正常回复输出数据 。
- 可读可写报文不包含数据长度

M 卡向 MCU 发送的应答报文格式：

起始符	错误码	DPRAM 数据	CRC 校验和
110000xx	1 字节	1~245 字节	2 字节

注：当用户 MCU 可读可写报文错误时，错误码 bit7~bit5 为 111。

用户 MCU 在发送该报文时，应预先知道从站响应报文长度，除了数据和 CRC 校验之外，还应包含起始符号 1 字节以及错误 1 字节。同时用户 MCU 应针对不同的报文类型来事先计算好从站响应报文的长度。

四、C 源代码说明

用户在阅读本小节内容前请先阅读“第六章/Fsamc_sram.h”。

1. “SPI.h”

```
#ifndef __SPI_H
#define __SPI_H
#include "fsmc_sram.h"
 (GPIO_ResetBits(GPIOA,GPIO_Pin_10),GPIO_ResetBits(GPIOA,GPIO_Pin_11))
#define   SPI1_NSS_SET() PAout(1)
#define   SPI1_NSS_RESET() PAout(0)
#define   GetDpramControl    0x02
#define   AbortDpramControl  0x00
#define   STX_SPIReply_NOK  0xE0
#define   GETDPRAM_SPIMODE    getDpram_SPIMode()
#define   ABORTDPRAM_SPIMODE  abortDpram_SPIMode()
#define   SPI_CS_ON   GPIO_ResetBits(GPIOA,GPIO_Pin_4)
#define   SPI_CS_OFF  GPIO_SetBits(GPIOA,GPIO_Pin_4)
#define BufferSize 11
#define RxBufferSize 255
extern uint8_t SPI1_Buffer_Rx[RxBufferSize];

void reset_DSDPV1_SPI(void);
void SPI1_Init(void);
void SPI2_Init(void);
void spi_read(u32 dpramMem,u16 readLen);
void spi_write(u32 dpramMem,u16 writeLen,u16 data);
void spi_writeBlock(u32 dpramMem,u16 writeLen,u8 *data);
void spi_ReadWrite(u8 *data);
void getDpram_SPIMode(void);
void abortDpram_SPIMode(void);
void SPIMode_GPIO_Init(void);
void DSDPV1_SPI1_PrmCfg_EXIT_config(void);
void Y_HardWare_InitData_SPIMode(void);
void Y_SoftWare_InitData_SPIMode(void);
void DPV0V1_DataEx_SPIMode(void);
void CopySPI_RxBufToDPRAM_DPV0Out(u8 *p,u16 len);
void CopySPI_RxBufToDPRAM_DPV1C1(u8 *p,u16 len);
void CopySPI_RxBufToDPRAM_DPV1C2(u8 *p,u16 len);
void CopySPI_RxBufToDPRAM_PRMData(u8 *p,u16 len);
void CopySPI_RxBufToDPRAM_CFGData(u8 *p,u16 len);
void IWDG_Configuration(uint8_t Prer, uint16_t Rlr);
void IWDG_Feed(void);
```

2. “SPI.C”

（1）定义结构体

```
#include "stm32f10x_spi.h"
#include "spi.h"
struct
   {
     uint8_t p;
     uint8_t box[260];
     uint8_t pend;
   }spi1_tr;
struct
   {
     uint8_t p;
     uint8_t box[260];
      uint8_t pend;
      uint8_t ok;
   }spi1_re;
//uint8_t SPI1_Buffer_Tx[BufferSize] =
{0x01,0x00,0x14,0x01,0x18,0xCF,0x00,0x00,0x00,0x00,0x00};
uint8_t SPI1_Buffer_Rx[RxBufferSize];
extern uint8_t SPISlave_Addr;
extern uint8_t   NewPrmData_HandleMode,NewCfgData_HandleMode;
extern uint8_t   EnableDPv1_Flag;
extern uint8_t   dsdpv1_status;
extern uint8_t diag_input[240];
extern uint8_t v0_output[244];  //V0 输出数据区
extern uint8_t v0_input[244];   //V0 输入数据区
extern uint8_t v1_output[240];  //V1 输出数据区
extern uint8_t v1_input[240];   //V1 输入数据区
extern uint8_t IM_output[64];   //IM 输出数据区
extern uint8_t IM_input[64];    //IM 输入数据区
extern uint8_t im0_body[64];
extern uint8_t IM_CallHeader[4];
extern uint8_t  DiagData_input[3];
extern uint8_t  New_Prmdata_Len,New_Cfgdata_Len;
extern uint8_t  New_Prmdata_Buf[MaxPrmLen],New_Cfgdata_Buf[MaxCfgLen];
extern uint8_t  Tn,Tn1,DIO_7;
extern uint8_t  ExtFlag,StaFlag;
extern uint8_t  ExtDiagData_input[3],StaDiagData_input[3];
extern uint8_t  current_diag_len,flag_diag;
extern uint8_t  si_status,im_status;
extern uint8_t  c_slot,c_index,c_len,c_subindex_H,c_subindex_L,c_rw,c_commu;
extern u8 PValue;
extern SLOT_INDEX slot_index[MAX_SI_NUM];
extern uint16_t  M_c1,M_cc1;
{//
  uint16_t i;
    //复位 SPI1
    for (i=0;i<65000;i++){GPIOE->BSRR = GPIO_Pin_4;} //后高电平
    for (i=0;i<65000;i++){GPIOE->BRR  = GPIO_Pin_4;} //先低电平
    //复位 SPI2
    for (i=0;i<800;i++){GPIOE->BRR  = GPIO_Pin_5;} //先高电平
    for (i=0;i<800;i++){GPIOE->BSRR = GPIO_Pin_5;} //后低电平
}
```

```
void IWDG_Configuration(uint8_t Prer, uint16_t Rlr)
{

  IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);//Enable write access to IWDG_PR and IWDG_RLR
registers   0x5555
  IWDG_SetPrescaler(Prer);                    //IWDG counter clock: 40KHz(LSI) / 32 = 1.25
KHz=0.8ms (0011=32 分频)
  IWDG_SetReload(Rlr);                        //Set counter reload value to ReloadR
  IWDG_ReloadCounter();                       //Reload IWDG counter  0xAAAA
    //RCC_LSICmd(ENABLE);                         //Enables or disables the Internal Low Speed
oscillator (LSI).
  IWDG_Enable();                              //Enable IWDG (the LSI oscillator will be enabled
by hardware) 0xCCCC
}
void IWDG_Feed(void)
{
  IWDG_ReloadCounter();
}
```

（2）初始化 SPI 接口/GPIO

```
void SPIMode_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO|RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB
2Periph_GPIOC|RCC_APB2Periph_GPIOD
|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG,
ENABLE);RCC_APB1PeriphClockCmd(RCC_APB1Periph_WWDG, ENABLE);
    //---------SPI1 pins: SCK, MISO and MOSI----------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
//------------SPI2 pins: NSS, SCK, MISO and MOSI-----------------------//
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_12 |GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
//-----------------SPI1 pins: NSS--------------------------------//
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);// SPI1-NSS
    SPI_CS_OFF;
}
SPI_InitTypeDef   SPI_InitStructure;

//-----------------------------------------------------------------------//
```

（3）设置外部中断

```
void DSDPV1_SPI1_PrmCfg_EXIT_config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOG,GPIO_PinSource6);
    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);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
void DSDPV1_SPI2_PrmCfg_EXIT_config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOD,GPIO_PinSource9);
    EXTI_InitStructure.EXTI_Line = EXTI_Line9 ;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt ;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling ;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
//------------------------------------------------------------------------//
```

（4）初始化 SPI 接口

```
void SPI1_Init()
{ //SCK 空闲状态为低电平，上升沿采样 (CPOL = 0,CPHA=0)
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,ENABLE);
  SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
  SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
  SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
  SPI_InitStructure.SPI_NSS =  SPI_NSS_Soft;//SPI_NSS_Hard
  SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStructure.SPI_CRCPolynomial = 0xA001;
  SPI_Init(SPI1, &SPI_InitStructure);
  SPI_Cmd(SPI1, ENABLE);
  SPI_I2S_ClearITPendingBit(SPI1, SPI_I2S_IT_RXNE);
  SPI_SSOutputCmd(SPI1, ENABLE);
}
void SPI2_Init()
{ //SCK 空闲状态为低电平，上升沿采样
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2,ENABLE);

   SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
  SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
  SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
  SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
  SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
```

```
  SPI_InitStructure.SPI_CRCPolynomial = 0xA001;
  SPI_Init(SPI2, &SPI_InitStructure);
  SPI_Cmd(SPI2, ENABLE);
}
```

（5）3 类通讯报文

```
void spi_read(u32 dpramMem,u16 readLen)
{
    uint8_t spi_CRC_H,spi_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t spi_CRC[5];
    uint16_t i=0,k=0;
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    spi_CRC[0] = (0x00|SPISlave_Addr);
    spi_CRC[1] = dpramMem_H;
    spi_CRC[2] = dpramMem_L;
    spi_CRC[3] = readLen;
    spi_CRC_H  = CRC_avr(spi_CRC,4)>>8;
    spi_CRC_L  = CRC_avr(spi_CRC,4);


  SPI_CS_ON;
  SPI_I2S_SendData(SPI1, spi_CRC[0]);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
  SPI_I2S_SendData(SPI1, dpramMem_H);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
  SPI_I2S_SendData(SPI1, dpramMem_L);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
  SPI_I2S_SendData(SPI1, readLen);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
  SPI_I2S_SendData(SPI1, spi_CRC_H);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
  SPI_I2S_SendData(SPI1, spi_CRC_L);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
  SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
  for(i=0;i<(readLen+4+2);i++)//dumy bytes generate SCK
  {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
     SPI_I2S_SendData(SPI1,0x00);
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
     SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    }
    i=k=0;
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET);
  SPI_CS_OFF;
}
void spi_write(u32 dpramMem,u16 writeLen,u16 data)
{
    uint8_t spi_CRC_H,spi_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t spi_CRC[5];
    uint16_t i=0,k=0;
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    spi_CRC[0] = (0x20|SPISlave_Addr);
```

```
    spi_CRC[1] = dpramMem_H;
    spi_CRC[2] = dpramMem_L;
    spi_CRC[3] = writeLen;
    spi_CRC[4] = data;
    spi_CRC_H  = CRC_avr(spi_CRC,5)>>8;
    spi_CRC_L  = CRC_avr(spi_CRC,5);
    SPI_CS_ON;
    SPI_I2S_SendData(SPI1, spi_CRC[0]);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
    SPI_I2S_SendData(SPI1, dpramMem_H);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
    SPI_I2S_SendData(SPI1, dpramMem_L);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
   SPI_I2S_SendData(SPI1, writeLen);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
   SPI_I2S_SendData(SPI1, data);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
   SPI_I2S_SendData(SPI1, spi_CRC_H);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(SPI1, spi_CRC_L);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
    SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);

    for(i=0;i<(writeLen+4+1);i++)//dumy bytes generate SCK
  {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
     SPI_I2S_SendData(SPI1,0x00);
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
     SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    }
    i=k=0;
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET);
  SPI_CS_OFF;
}
void spi_writeBlock(u32 dpramMem,u16 writeLen,u8 *data)
{
    uint8_t spi_CRC_H,spi_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t spi_CRC[250];
    uint16_t i=0,k=0;
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    spi_CRC[0] = (0x20|SPISlave_Addr);
    spi_CRC[1] = dpramMem_H;
    spi_CRC[2] = dpramMem_L;
    spi_CRC[3] = writeLen;
    for(i=0;i<writeLen;i++){spi_CRC[i+4] = *(data+i);}
    spi_CRC_H  = CRC_avr(spi_CRC,writeLen+4)>>8;
    spi_CRC_L  = CRC_avr(spi_CRC,writeLen+4);
    SPI_CS_ON;

    SPI_I2S_SendData(SPI1, spi_CRC[0]);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
    SPI_I2S_SendData(SPI1, dpramMem_H);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
    SPI_I2S_SendData(SPI1, dpramMem_L);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
```

```
    SPI_I2S_SendData(SPI1, writeLen);
    for(i=0;i<writeLen;i++)
    {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
   SPI_I2S_SendData(SPI1, *(data+i));
    }
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
  SPI_I2S_SendData(SPI1, spi_CRC_H);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
  SPI_I2S_SendData(SPI1, spi_CRC_L);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
  SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    for(i=0;i<6;i++)//dummy bytes generate SCK
  {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
     SPI_I2S_SendData(SPI1,0x00);
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
     SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    }
    i=k=0;
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET);
  SPI_CS_OFF;
    }
void spi_ReadWrite(u8 *data)
{
    uint8_t spi_CRC_H,spi_CRC_L;
    uint8_t spi_CRC[250];
    uint16_t i=0,k=0;
   spi_CRC[0] = (0x40|SPISlave_Addr);//41 30 01 02 03 .....
    for(i=0;i<(*data)+1;i++){spi_CRC[i+1] = *(data+i);}
   spi_CRC_H  = CRC_avr(spi_CRC,(*data)+2)>>8;
    spi_CRC_L  = CRC_avr(spi_CRC,(*data)+2);
    SPI_CS_ON;
    SPI_I2S_SendData(SPI1, spi_CRC[0]);
    for(i=0;i<(*data)+1;i++)
    {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
   SPI_I2S_SendData(SPI1, *(data+i));
    }
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(SPI1, spi_CRC_H);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(SPI1, spi_CRC_L);
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
    SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    for(i=0;i<(*data)+1+4+2;i++)//dummy bytes generate SCK
  {
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE)  == RESET);
     SPI_I2S_SendData(SPI1,0x00);
     while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
     SPI1_Buffer_Rx[k++] = SPI_I2S_ReceiveData(SPI1);
    i=k=0;
    while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET);
  SPI_CS_OFF;
}
```

（6）获取 DPRAM 读写控制权

```
void getDpram_SPIMode(void)
```

```
{
    u8 spiRx_CRC_H,spiRx_CRC_L;
    u8 w;
    for(w=0;w==0;)
        {
//       spi_read((u16)&dpram.SPI_Status,1);//reply :FF FF 81 00 00 E8 21
//
//       spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],3)>>8;
//       spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],3);
 //    if(((SPI1_Buffer_Rx[2]&0xE0)==0x80)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[5])&&(spiRx_CRC_L==SPI1_Buffer_Rx[6])&&(SPI1_Buffer_Rx[4]==0x00))
//     {
dpram.SPI_Status = GetDpramControl;
spi_write((u32)&dpram.SPI_Status,1,GetDpramControl);//reply : FF FF A1 00 20 78
         spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],2)>>8;
          spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],2);
             if(((SPI1_Buffer_Rx[2]&0xE0)==0xA0)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[4])&&(spiRx_CRC_L==SPI1_Buffer_Rx[5])&&(SPI1_Buffer_Rx[3]==0x00))
             {w=1;}
             else
             {w=0;}
    //   }
    //   else
    //   {w=0;}
      }
}
```

（7）释放 DPRAM 读写控制权

```
void abortDpram_SPIMode(void)
{
    u8 spiRx_CRC_H,spiRx_CRC_L;
    u8 w;
    for(w=0;w==0;)
        {
    dpram.SPI_Status = AbortDpramControl;
 spi_write((u32)&dpram.SPI_Status,1,AbortDpramControl);//reply : FF FF A1 00 20 78
             spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],2)>>8;
           spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],2);
             if(((SPI1_Buffer_Rx[2]&0xE0)==0xA0)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[4])&&(spiRx_CRC_L==SPI1_Buffer_Rx[5])&&(SPI1_Buffer_Rx[3]==0x00))
             {w=1;}
             else
             {w=0;}
      }
}
```

（8）用户板硬件初始化

```
//==================用户板硬件初始化================================//
void Y_HardWare_InitData_SPIMode(void)
{
    u8 w,i;
    GETDPRAM_SPIMODE;
  for(w=0;w==0;)
  {
          spi_read((u32)&dpram.HardwareInit_Ldpram,1);
          dpram.HardwareInit_Ldpram = SPI1_Buffer_Rx[4];
          if(dpram.HardwareInit_Ldpram == 0xC5)
       {
```

```
          dpram.R_status1  = 0;
          dpram.R_status2  = 0;
          dpram.R_command1 = 0;
          dpram.R_command2 = 0;
          for(i=0;i<215;i++) {dpram.InitData_DPv0[i] = 0;}//V0 初始化数据区
          for(i=0;i<245;i++) {dpram.InData_DP[i]     = 0;}//PROFIBUS 输入数据区
          for(i=0;i<240;i++) {dpram.DiagData_DP[i]   = 0;}//Diag 诊断数据区
        //----------SPI-----------------------------------------//
                spi_write((u32)&dpram.R_status1,1,  dpram.R_status1);
                spi_write((u32)&dpram.R_status2,1,  dpram.R_status2);
                spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
                spi_write((u32)&dpram.R_command2,1, dpram.R_command2);
    spi_writeBlock((u32)&dpram.InitData_DPv0,215,dpram.InitData_DPv0);
                spi_writeBlock((u32)&dpram.InData_DP,    245,dpram.InData_DP);
    spi_writeBlock((u32)&dpram.DiagData_DP,  240,dpram.DiagData_DP);
        //----------SPI-----------------------------------------//
                w=1;
            ABORTDPRAM_SPIMODE;
       }
      else
         {
          w=0;
          ABORTDPRAM_SPIMODE;
            delay_ms(5);
            GETDPRAM_SPIMODE;
         }
        //ABORT_DPRAM;
  }
}
```

（9）软件初始化

```
void Y_SoftWare_InitData_SPIMode(void)
{
    u8 w;
     dpram.InitData_DPv0[0]=22;                          //V0 初始化数据长度字节
    dpram.InitData_DPv0[1]=dp_address();//dp_address();//*ADDRESS;//从站站地址
    dpram.InitData_DPv0[2]=0x0C;                        //ID 号高字节
    dpram.InitData_DPv0[3]=0xC9;                        //ID 号低字节
    dpram.InitData_DPv0[4]=0x0F; //SPC3_Register_status0:0000 fS SYN FREEZE (SSA)
    dpram.InitData_DPv0[5]=0;                           //预留备用
    dpram.InitData_DPv0[6]=0;                           //预留备用
    dpram.InitData_DPv0[7]=244; //48            //PROFIBUS 输入数据最大可能长度 a
    dpram.InitData_DPv0[8]=244; //48             //PROFIBUS 输出数据最大可能长度 b
    dpram.InitData_DPv0[9]=238; //9 //用户诊断数据长度（≤238 不包含标准 6 字节诊断数据）
    dpram.InitData_DPv0[10]=MaxPrmLen;//MaxPrmLen237//用户参数数据最大可能长度 j（≤237 不包含
标准 7 字节参数数据）
    dpram.InitData_DPv0[11]=NewPrmData_HandleMode;  //用户参数正确与否判断方式
    dpram.InitData_DPv0[12]=MaxCfgLen;//MaxCfgLen200//配置数据的最大可能长度 m（≤200）
    dpram.InitData_DPv0[13]=NewCfgData_HandleMode;  //配置数据正确与否判断方式
    dpram.InitData_DPv0[14]=6;                          //默认 CFG 数据长度 = n
    dpram.InitData_DPv0[15]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[16]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[17]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[18]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[19]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[20]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[21]=0x00;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[22]=0x00;                       //默认用户诊断数据长度
```

```
   /*
      dpram.InitData_DPv0[14]=8;                          //默认 CFG 数据长度 = n
    dpram.InitData_DPv0[15]=0x40;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[16]=0x7c;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[17]=0x80;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[18]=0x7c;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[19]=0x40;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[20]=0x7c;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[21]=0x80;                        //默认用户诊断数据长度
    dpram.InitData_DPv0[22]=0x7c;                        //默认用户诊断数据长度
     */
   for(w=0;w==0;)
   {
    GETDPRAM_SPIMODE;
    spi_writeBlock((u32)&dpram.InitData_DPv0,23,dpram.InitData_DPv0);
    if(EnableDPv1_Flag == 1)
      {
       dpram.EnableDPv1_Rdpram=0x1D;                     //V1 功能开启标志 B0005
       spi_write((u32)&dpram.EnableDPv1_Rdpram,1, dpram.EnableDPv1_Rdpram);
      }
      else
      {
         dpram.EnableDPv1_Rdpram=0x00;
         spi_write((u32)&dpram.EnableDPv1_Rdpram,1, dpram.EnableDPv1_Rdpram);
       } //V1 功能开启标志 B0005
      dpram.R_command1 |= 0x04;       //用户板命令字节 1_初始化信息有效标志
        spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
        //spi_read ((u32)&dpram.R_command1,1);
      ABORTDPRAM_SPIMODE;
    delay_ms(20);//此处需要等待接口板处理初始化结果，可能需要延时较长时间
    GETDPRAM_SPIMODE;
      //dpram.AccessCounter_LRdpram +=1;
        spi_read((u32)&dpram.L_status1,1);
        dpram.L_status1=SPI1_Buffer_Rx[4];
      if((dpram.L_status1&0x01)==0x01)
            {
                w=1;
                spi_read((u32)&dpram.R_command1,1);
                dpram.R_command1 = SPI1_Buffer_Rx[4];
                ABORTDPRAM_SPIMODE;
            }
      else
            {
                w=0;
                ABORTDPRAM_SPIMODE;
                delay_ms(10);
            }
   }
}
void CopySPI_RxBufToDPRAM_PRMData(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.PrmData_DP[i+1]=*(p+i);}
}
void CopySPI_RxBufToDPRAM_CFGData(u8 *p,u16 len)
{
    u8 i;
```

```
    for(i=0;i<len;i++)
    {dpram.CfgData_DP[i+3]=*(p+i);}
}
void CopySPI_RxBufToDPRAM_DPV0Out(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.OutData_DP[i+1]=*(p+i);}
}
void CopySPI_RxBufToDPRAM_DPV1C1(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.AcycData_DPv1c1[i]=*(p+i);}
}
void CopySPI_RxBufToDPRAM_DPV1C2(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.AcycData_DPv1c2[i]=*(p+i);}
}
```

（10）DPV1/C1 数据处理

```
void Acyclic_data_C1_SPIMode(void)
{
    u16 i;
    u8  w_slot,w_index,slot_curr;
    u8  len_status;//error_status
    i=0;
    w_slot=w_index=0;
    slot_curr=0;
    len_status=0;
    switch(dpram.AcycData_DPv1c1[0]) //进来的是:0x5E 或 0x5F
      {
        case 0x5E :   //进来的是:0x5E Read
        {
                  //c_rw    = SI_R;
            c_slot  = dpram.AcycData_DPv1c1[1];
            c_index = dpram.AcycData_DPv1c1[2];
            c_len   = dpram.AcycData_DPv1c1[3];
            for(i=0;i<MAX_SI_NUM;i++)
            {//1
            if((slot_index[i].rw==SI_R)||(slot_index[i].rw==SI_RW))  //仅在具有读属性的槽-
索引内寻找
              {//2

            if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                  {//3

             slot_curr=0x01;//声明列表中出现过正确的槽号

              if(c_index==slot_index[i].index)
              {si_status=SI_OK;break;}
               else
                        {
                           si_status=SI_NOK;
                           w_index=1;
                                }
```

```
                    }
                            else
                            {
                        if(slot_curr==0)
                              {
                                si_status=SI_NOK;
                                w_slot=1;
                              }
                            else
                              {
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                            }//3
                 }
                      else                          {
                         //读写属性错误
                           //p0=1;
                       if(slot_curr==0)
                       {
                          si_status=SI_NOK;
                          w_slot=1;
                       }
                       else
                       {
                          si_status=SI_NOK;
                            w_slot=0;
                       }
                       }//2
               }//1
                   if(si_status == SI_OK)
                   {
                     if((c_len>240)||(c_len==0))
                       {len_status = 1;}
                     if(len_status == 0)
                     {
  if((c_slot==IM_SLOT)&&(c_index==IM_INDEX)&&(c_len==0x44))//为 IM 槽索引
                    {
                     if(im_status==RE_IMCALL)
                    {
                     deal_im_data(); //im0_body[i] = IM_read[i]
                     dpram.AcycData_DPv1c1[0] = 0x5E;
                     dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                     dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                     dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
                     for(i=0;i<64;i++)
                         {
                       dpram.AcycData_DPv1c1[4] = IM_CallHeader[0];// 0x08
                       dpram.AcycData_DPv1c1[5] = IM_CallHeader[1];// 0x00
                       dpram.AcycData_DPv1c1[6] = IM_CallHeader[2];// 0xFD
                       dpram.AcycData_DPv1c1[7] = IM_CallHeader[3];// 0xE8
                       dpram.AcycData_DPv1c1[i+8]=im0_body[i];//把 CALL-DU 部分取出
                           }
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4,&dpram.AcycData_DPv1c1
[0]);
//dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4)>>8;
//dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4);
 im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
```

```
si_status=0;
}
else if(im_status==RES_IMCALL)
{
  deal_im_data();
  dpram.AcycData_DPv1c1[0] = 0x5E;
  dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
  dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
  dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
 for(i=0;i<64;i++)
 {
   dpram.AcycData_DPv1c1[4] = IM_CallHeader[0];// 0x08
   dpram.AcycData_DPv1c1[5] = IM_CallHeader[1];// 0x00
   dpram.AcycData_DPv1c1[6] = IM_CallHeader[2];// 0xFD
   dpram.AcycData_DPv1c1[7] = IM_CallHeader[3];// 0xE8
   dpram.AcycData_DPv1c1[i+8]=im0_body[i];  //把 CALL-DU 部分取出
 }
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4,&dpram.AcycData_DPv1c1
[0]);
 im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
si_status=0;
  }
 else if(im_status==NO_IMCALL)
 {
 dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
 dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
 dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_STATE);   spi_writeBlo
ck((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
 si_status = 0;
  }
 }
else //为普通槽索引
{
  c_len=deal_acyclic_read_data(c_slot,c_index,c_len);
  dpram.AcycData_DPv1c1[0] = 0x5E;
  dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
  dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
  for(i=0;i<c_len;i++){dpram.AcycData_DPv1c1[i+4]=v1_input[i];}
  spi_writeBlock((u32)&dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4,&dpram.AcycData_DPv1
c1[0]);
   }
  }
 else //长度错误   B7
{
 dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
 dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
 dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
 dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
len_status=0;
si_status=0;
 }
 }
   else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误 B2
 {dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
 dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
 dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT); dpram.AcycD
```

```
ata_DPv1c1[3]  = 0x00; //Error_Code_2
 spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
  si_status=0;
 }
else if((si_status == SI_NOK)&&(w_index==1))//索引错误 B0
{
 dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
 dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
 dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
 dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
 spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
 si_status=0;
}
 }break;
 case 0x5F :   //进来的是:0x5F Write
{// c_rw    = SI_W;
 c_slot  = dpram.AcycData_DPv1c1[1];//slot
 c_index = dpram.AcycData_DPv1c1[2];//index
 c_len   = dpram.AcycData_DPv1c1[3];//length
for(i=0;i<MAX_SI_NUM;i++)
{//1if((slot_index[i].rw==SI_W)||(slot_index[i].rw==SI_RW))  //仅在具有读属性的槽-索引内寻找
{//2
if(c_slot==slot_index[i].slot)  //槽号是否在列表中
{//3 slot_curr=0x01;//声明列表中出现过正确的槽号

     if(c_index==slot_index[i].index)
     {si_status=SI_OK;break;}
     else
    {
     si_status=SI_NOK;
     w_index=1;
     }
    }
    else
     {
        if(slot_curr==0)
         {
            si_status=SI_NOK;
            w_slot=1;
          }
         else
          {
             si_status=SI_NOK;
             w_slot=0;
            }
         }//3
       }
       else
          {
            if(slot_curr==0)
              {
                 si_status=SI_NOK;
                  w_slot=1;
               }
               else
                 {
                     si_status=SI_NOK;
```

```
                    w_slot=0;
                  }
        }//2

     }//1
if(si_status == SI_OK)
 {
    if((c_len>240)||(c_len==0))
    {len_status = 1;}
      if(len_status == 0)
        {
         if((c_slot==IM_SLOT)&&(c_index==IM_INDEX))
            {// 5F 00 FF 04 08 00 FD E8
                if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                    {

if((dpram.AcycData_DPv1c1[3]==4)&&(dpram.AcycData_DPv1c1[4]==8)&&(dpram.AcycData_DPv1c1[5]==
0)&&(dpram.AcycData_DPv1c1[6]==0xFD)&&(dpram.AcycData_DPv1c1[7]==0xE8))
                          {  //下面 4 条可不要
                              dpram.AcycData_DPv1c1[0] = 0x5F;
                              dpram.AcycData_DPv1c1[1] = c_slot;
                              dpram.AcycData_DPv1c1[2] = c_index;
                              dpram.AcycData_DPv1c1[3] = c_len;

        IM_CallHeader[0] = dpram.AcycData_DPv1c1[4];// 0x08
        IM_CallHeader[1] = dpram.AcycData_DPv1c1[5];// 0x00
        IM_CallHeader[2] = dpram.AcycData_DPv1c1[6];// 0xFD
        IM_CallHeader[3] = dpram.AcycData_DPv1c1[7];// 0xE8
       spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
        im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
        si_status=0;
     }

      else //application  & feature not supported (不是 65000) B6
        {
          dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
          dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
          dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS); //Er
ror_Code_1 :invalid slot
         dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
       spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
        im_status=NO_IMCALL;
        si_status=0;
     }
    }//if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
      else //(im_status==RE_IMCALL)
        {  if((dpram.AcycData_DPv1c1[3]==4)&&(dpram.AcycData_DPv1c1[4]==8)&&(dpram.AcycData_
DPv1c1[5]==0) &&(dpram.AcycData_DPv1c1[6]==0xFD)&&(dpram.AcycData_DPv1c1[7]==0xE8))
            {  //下面 4 条可不要
              dpram.AcycData_DPv1c1[0] = 0x5F;
              dpram.AcycData_DPv1c1[1] = c_slot;
              dpram.AcycData_DPv1c1[2] = c_index;
              dpram.AcycData_DPv1c1[3] = c_len;

             IM_CallHeader[0] = dpram.AcycData_DPv1c1[4];// 0x08
             IM_CallHeader[1] = dpram.AcycData_DPv1c1[5];// 0x00
             IM_CallHeader[2] = dpram.AcycData_DPv1c1[6];// 0xFD
             IM_CallHeader[3] = dpram.AcycData_DPv1c1[7];// 0xE8
```

```
    spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
              im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
              si_status=0;
              // im_status=READ_IMCALL;
                          }
                     else //application  & feature not supported (不是 65000) B6
                         {
                  dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
                  dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
 dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
//dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4)>>8;
//dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4);

                                im_status=NO_IMCALL;
                                  si_status=0;
                       }
                           }
                      }
                      else  //普通槽索引
                      {
                      dpram.AcycData_DPv1c1[0] = 0x5F;
                      dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                      dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                      dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
for(i=0;i<c_len;i++){v1_output[i]=dpram.AcycData_DPv1c1[i+4];}
deal_acyclic_write_data(c_slot,c_index,c_len);
 //dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4)>>8;
 //dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4);
                      }
                  }
                    else //长度错误  B7
                    {
dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
//dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4)>>8;
//dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4);

                    si_status=0;
                  }
                }
                else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误  B6
                {
            dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
            dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
//dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
//dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4)>>8;
```

```
 //dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4);
                  si_status=0;
                }
                else if((si_status == SI_NOK)&&(w_index==1))//索引错误  B0
                {
            dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
            dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
            dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
 dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycData_DPv1c1[0]);
 //dpram.CRC_H_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4)>>8;
 //dpram.CRC_L_AcycData_DPv1c1 = CRC_avr(dpram.AcycData_DPv1c1,4);
                  si_status=0;
                }

               }break;
         default : break;
            }
}
void Acyclic_data_C2_SPIMode(void)
{
    u16 i;
    u8  w_slot,w_index,slot_curr;
  u8  len_status;
  w_slot=w_index=slot_curr=0;
    len_status=0;
    i=0;
  switch(dpram.AcycData_DPv1c2[0]) //进来的是:0x5E 或 0x5F
   {
    case 0x5E :
      {
                   //c_rw    = SI_R;
                   c_slot  = dpram.AcycData_DPv1c2[1];
                   c_index = dpram.AcycData_DPv1c2[2];
                   c_len   = dpram.AcycData_DPv1c2[3];
             for(i=0;i<MAX_SI_NUM;i++)
             {//1
 if((slot_index[i].rw==SI_R)||(slot_index[i].rw==SI_RW))   //仅在具有读属性的槽-索引内寻找
               {//2
                 if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                 {//3
                  slot_curr=0x01;//声明列表中出现过正确的槽号

                      if(c_index==slot_index[i].index)
                   {si_status=SI_OK;break;}
                      else
                      {
                            si_status=SI_NOK;
                                       w_index=1;
                         }
                 }
                        else
                        {
                             if(slot_curr==0)
                             {
                               si_status=SI_NOK;
                               w_slot=1;
```

```
                              }
                            else
                              {
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                          }//3
                }
                      else
                      {
                            if(slot_curr==0)
                              {
                                si_status=SI_NOK;
                                w_slot=1;
                              }
                            else
                              {
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                      }//2
              }//1 for(i=0;i<MAX_SI_NUM;i++)
              if(si_status == SI_OK)
              {
                 if((c_len>240)||(c_len==0))
                   {len_status = 1;}
                 if(len_status == 0)
                 {
                         if(c_len>240)
                 {
                  c_len=240;
                 }
if((c_slot==IM_SLOT)&&(c_index==IM_INDEX)&&(c_len==0x44))//为 IM 槽索引
              {
                        if(im_status==RE_IMCALL)
                          {
                            deal_im_data(); //im0_body[i] = IM_read[i]
                            dpram.AcycData_DPv1c2[0] = 0x5E;
                            dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                            dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                            dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                            for(i=0;i<64;i++)
                          {
                             dpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
    dpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
    dpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
    dpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
    dpram.AcycData_DPv1c2[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                          }
}
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4,&dpram.AcycData_DPv1c2
[0]);
 im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                        si_status=0;
                           }
                              else if(im_status==RES_IMCALL)
                              {
                     /read_done=deal_im_data(); //im0_body[i] = IM_read[i]
```

```
                                    deal_im_data();
                                    dpram.AcycData_DPv1c2[0] = 0x5E;
                            dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                            dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                            dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                               for(i=0;i<64;i++)
                              {
                                 dpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
                                 dpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
                                 dpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
                                 dpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
                dpram.AcycData_DPv1c2[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                              }
    spi_writeBlock((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4,&dpram.AcycData_DP
v1c2[0]);
                          im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                          si_status=0;
                                }
                                else if(im_status==NO_IMCALL)
                                {
                                     dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                     dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                     dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_STATE);
//Error_Code_1 :invalid slot
                     dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
//dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
//dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
                           si_status = 0;
                                }
                        }
                        else //为普通槽索引
                        {
                        c_len=deal_acyclic_read_data(c_slot,c_index,c_len);
                       dpram.AcycData_DPv1c2[0] = 0x5E;
                       dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                       dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                       dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=v1_input[i];}
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4,&dpram.AcycData_DPv1c2
[0]);
                     }
                       }
                      else //长度错误
                      {
                     dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                     dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                     dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)|
DPSE_ERRCL_ACC_INV_RANGE); //Error_Code_1 :invalid slot
                     dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
                     si_status = 0;
                    }
                }
                else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误
                {
                  dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                  dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
```

```
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
    spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
 //dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
 //dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
                si_status = 0;
              }
              else if((si_status == SI_NOK)&&(w_index==1))//索引错误
              {
                dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
    spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
    //dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
    //dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
                si_status = 0;
              }
     }break;
    case 0x5F :
        { //c_rw    = SI_W;
              c_slot  = dpram.AcycData_DPv1c2[1];//slot
              c_index = dpram.AcycData_DPv1c2[2];//index
               c_len   = dpram.AcycData_DPv1c2[3];//length

              for(i=0;i<MAX_SI_NUM;i++)
              {//1
if((slot_index[i].rw==SI_W)||(slot_index[i].rw==SI_RW))   //仅在具有读属性的槽-索引内寻找
                {//2
                  if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                  {//3
                             slot_curr=0x01;//声明列表中出现过正确的槽号
                        if(c_index==slot_index[i].index)
                     {si_status=SI_OK;break;}
                         else
                         {
                              si_status=SI_NOK;
                      w_index=1;
                             }
                  }
                           else
                           {
                                if(slot_curr==0)
                                {
                                  si_status=SI_NOK;
                                  w_slot=1;
                                }
                              else
                                {
                                  si_status=SI_NOK;
                                  w_slot=0;
                                }
                           }//3
              }
                    else
                    {
```

```
                            if(slot_curr==0)
                              {
                                si_status=SI_NOK;
                                w_slot=1;
                              }
                            else
                              {
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                        }//2

                }//1

                if(si_status == SI_OK)
                {
                   if((c_len>240)||(c_len==0))
                     {len_status = 1;}
                   if(len_status == 0)
                   {
                       if((c_slot==IM_SLOT)&&(c_index==IM_INDEX))
                       {// 5F 00 FF 04 08 00 FD E8
                        if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                            {
                               if((dpram.AcycData_DPv1c2[3]==4)
&&(dpram.AcycData_DPv1c2[4]==8)&&(dpram.AcycData_DPv1c2[5]==0)
&&(dpram.AcycData_DPv1c2[6]==0xFD)&&(dpram.AcycData_DPv1c2[7]==0xE8))
                             {  //下面 4 条可不要
                                dpram.AcycData_DPv1c2[0] = 0x5F;
                                dpram.AcycData_DPv1c2[1] = c_slot;
                                dpram.AcycData_DPv1c2[2] = c_index;
                                dpram.AcycData_DPv1c2[3] = c_len;

                              IM_CallHeader[0] = dpram.AcycData_DPv1c2[4];// 0x08
                              IM_CallHeader[1] = dpram.AcycData_DPv1c2[5];// 0x00
                               IM_CallHeader[2] = dpram.AcycData_DPv1c2[6];// 0xFD
                               IM_CallHeader[3] = dpram.AcycData_DPv1c2[7];// 0xE8
     spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
    //dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
    //dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
     im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                                si_status=0;
                              // im_status=READ_IMCALL;
                              }
                            else //application  & feature not supported (不是 65000) B6
                              {
                    dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
                    dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                    dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
                    dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
     spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
    //dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
    //dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);

                                im_status=NO_IMCALL;
                                  si_status=0;
                        }
```

```
                    }//if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                        else //(im_status==RE_IMCALL)
            {
                          if((dpram.AcycData_DPv1c2[3]==4)
&&(dpram.AcycData_DPv1c2[4]==8)&&(dpram.AcycData_DPv1c2[5]==0)
&&(dpram.AcycData_DPv1c2[6]==0xFD)&&(dpram.AcycData_DPv1c2[7]==0xE8))
                     {  //下面 4 条可不要
                          dpram.AcycData_DPv1c2[0] = 0x5F;
                          dpram.AcycData_DPv1c2[1] = c_slot;
                          dpram.AcycData_DPv1c2[2] = c_index;
                          dpram.AcycData_DPv1c2[3] = c_len;

                          IM_CallHeader[0] = dpram.AcycData_DPv1c2[4];// 0x08
                          IM_CallHeader[1] = dpram.AcycData_DPv1c2[5];// 0x00
                           IM_CallHeader[2] = dpram.AcycData_DPv1c2[6];// 0xFD
                           IM_CallHeader[3] = dpram.AcycData_DPv1c2[7];// 0xE8
 spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
//dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
//dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
 im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                          si_status=0;
                        // im_status=READ_IMCALL;
                       }
                      else //application  & feature not supported (不是 65000) B6
                        {
                dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
 dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
 //dpram.CRC_H_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4)>>8;
 //dpram.CRC_L_AcycData_DPv1c2 = CRC_avr(dpram.AcycData_DPv1c2,4);
     im_status=NO_IMCALL;
                                  si_status=0;
                    }
                         }
                        }
                    else
                    {
                   dpram.AcycData_DPv1c2[0] = 0x5F;
                   dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                   dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                   dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
for(i=0;i<c_len;i++){v1_output[i]=dpram.AcycData_DPv1c2[i+4];}
 deal_acyclic_write_data(c_slot,c_index,c_len);
                    }
                }
                  else //长度错误  B7
                  {
           dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
           dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
           dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
           dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
     spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
                si_status = 0;
```

```
                }
            }
            else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误  B6
            {
         dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
       //dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
         dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);
              si_status = 0;
            }
            else if((si_status == SI_NOK)&&(w_index==1))//索引错误 B0
            {
         dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
         dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot        dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);

            }
      }break;
        case 0x51 :
          {
                c_slot  = dpram.AcycData_DPv1c2[1];//slot
                c_index = dpram.AcycData_DPv1c2[2];//index
                c_len   = dpram.AcycData_DPv1c2[3];//length
                if((c_slot==SLOT_03)&&(c_index==S03_INDEX_01)) //slot_index=[4,1]
                {
                 dpram.AcycData_DPv1c2[0] = 0x51;
               dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
               dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
               dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
               for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=i;}
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4,&dpram.AcycData_DPv1c2
[0]);
;
             }
                else
                {
                 dpram.AcycData_DPv1c2[0] |= 0xD1; //Function Code
                 dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                 dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_APPLICATION <<4)|
DPSE_ERRCL_APP_NOTSUPP); //Error_Code_1 :invalid slot
                 dpram.AcycData_DPv1c2[3]  = 0x51; //Error_Code_2
spi_writeBlock((u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycData_DPv1c2[0]);

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

（11）诊断数据处理

```
void DP_DiagHandler(void)
{
        uint8_t i;
```

```
      spi_read((u32)&dpram.R_command1,1);
      dpram.R_command1=SPI1_Buffer_Rx[4];
      dpram.R_command1 &=0xC7;
  //==================Ext_Diag 处理=====================================//
if(((ExtFlag&0x0F)==1)&&((StaFlag&0x0F)==0))//Ext.diag 高优先权外部诊断，从站有错误
        {
                if(flag_diag == 0)
                {
                  dpram.R_command1 |= 0x02;
                  spi_write((u32)&dpram.R_command1,1, dpram.R_command1);

                    dpram.DiagData_DP[0] = 0x81;
                   current_diag_len=dpram.InitData_DPv0[9]-6;
for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF1
                    Tn=1;
                  flag_diag = 1;
                }
          }
else if(((ExtFlag&0x0F)==0)&&((StaFlag&0x0F)==0)&&(Tn==1))//Ext.diag 高优先权外部诊断，从站无
错误
        {
                if(flag_diag == 1)
                {
                  dpram.R_command1 |= 0x02;
                  spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
                  dpram.DiagData_DP[0] = 0x01;
                  current_diag_len=dpram.InitData_DPv0[9]-6
for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF0
                    Tn=0;
                    flag_diag = 0;
              }
            }
else if(((StaFlag&0x0F)==1)&&((ExtFlag&0x0F)==0))    //Stat.diag  高优先权静态诊断，从站有错误
        {
                if(flag_diag == 0)
                {
                  dpram.R_command1 |= 0x02;
                    spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
                    dpram.DiagData_DP[0] = 0x82;
                  current_diag_len=dpram.InitData_DPv0[9]-6;
for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF2
                    Tn1=1;
                  flag_diag = 1;
              }
            }
else if(((StaFlag&0x0F)==0)&&((ExtFlag&0x0F)==0)&&(Tn1==1))//Stat.diag 高优先权静态诊断，从站
无错误
        {
                if(flag_diag == 1)
                {
                  dpram.R_command1 |= 0x02;
                    spi_write((u32)&dpram.R_command1,1, dpram.R_command1);

                    dpram.DiagData_DP[0] = 0x02;
                    current_diag_len=dpram.InitData_DPv0[9]-6;
    for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF0
                    Tn1=0;
                    flag_diag = 0;
```

```
            }
          }
    if((dpram.R_command1&0x02) == 0x02)
      {
        dpram.DiagData_DP[1] = current_diag_len;

        for(i=0;i<dpram.DiagData_DP[1];i++)
           {dpram.DiagData_DP[i+2] = diag_input[i];}
spi_writeBlock((u32)&dpram.DiagData_DP,dpram.DiagData_DP[1]+2,dpram.DiagData_DP)
     }
```

（12）DPV0V1 数据交换

```
 //=========================DPV0V1 数据交换=============================//

void  DPV0V1_DataEx_SPIMode(void) //IO 数据缓冲区与双口 RAM 之间的数据交换
{
     u8 i;
   u8 spiRx_CRC_H=0,spiRx_CRC_L=0;
   GETDPRAM_SPIMODE;
    spi_read((u32)&dpram.L_status3,1);
    spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],3)>>8;
    spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],3);
    if(((SPI1_Buffer_Rx[2]&0xE0)!=STX_SPIReply_NOK)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[5])&&(spiRx_CRC_L==SPI1_Buffer_Rx[6]))
     {
       dpram.L_status3=SPI1_Buffer_Rx[4];
     dsdpv1_status=(dpram.L_status3&0x0C);
       if(dsdpv1_status==0x0c) //表示 DP 芯片进入数据交换状态，开始 DPv0 数据处理
       {
          spi_read((u32)&dpram.L_command1,1);
        spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],3)>>8;
          spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],3);
              if(((SPI1_Buffer_Rx[2]&0xE0)!=STX_SPIReply_NOK)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[5])&&(spiRx_CRC_L==SPI1_Buffer_Rx[6]))
        {
             dpram.L_command1=SPI1_Buffer_Rx[4];
           if((dpram.L_command1&0x01) == 0x01 )
           {
//===================DPV0_OUT 处理==============================//
                dpram.L_command1 &= 0xFE;
                spi_write((u32)&dpram.L_command1,1, dpram.L_command1);
                    spi_read((u32)&dpram.CfgData_DP,1);//读取 DP 输出数据长度
                    dpram.CfgData_DP[0]=SPI1_Buffer_Rx[4];
                    dpram.OutData_DP[0]=dpram.CfgData_DP[0];
                    spi_read((u32)&dpram.OutData_DP,dpram.OutData_DP[0]+1);        //通过 SPI
接口，获取 DPV0 输出数据
spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.OutData_DP[0]+3)>>8;
 spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.OutData_DP[0]+3);
if(((SPI1_Buffer_Rx[2]&0xE0)!=STX_SPIReply_NOK)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[dpram.OutData_DP[0]+5])&&(spiRx_CRC_L==SPI1_Buffer_Rx[dpram.OutData_DP[0]+6])
)
              {
CopySPI_RxBufToDPRAM_DPV0Out(&SPI1_Buffer_Rx[5],dpram.OutData_DP[0]);//复制 DPv0 输出数据到本
地缓存
 dpram.R_status2 &= 0xFB;//B000D.2 从接口板获得的输出数据 CRC_avr 正确
spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
 for(i=0;i<dpram.OutData_DP[0];i++)
```

```
    {v0_output[i] = dpram.OutData_DP[i+1];}
                  }
               else
                           {
 dpram.R_status2 |= 0x04;//B000D.2 从接口板获得的输出数据 CRC_avr 错误
spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
                           }
             }
}
//=========================DPV0_IN 处理================================//
            spi_read((u32)&dpram.CfgData_DP[1],1);//读取 DP 输入数据长度
                   dpram.CfgData_DP[1]=SPI1_Buffer_Rx[4];
                dpram.InData_DP[0] =dpram.CfgData_DP[1];
            for(i=0;i<dpram.InData_DP[0];i++)
              {
               dpram.InData_DP[i+1] = v0_input[i];
             //dpram.InData_DP[i+1] = dpram.OutData_DP[i+1];
              }
spi_writeBlock((u32)&dpram.InData_DP,dpram.InData_DP[0]+1,dpram.InData_DP);
dpram.R_command1 |= 0x01;
    spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
       }
   }
     //=================诊断处理=====================================//
      DP_DiagHandler();
```

<14>、Acyclic_data_C1 处理

```
  //=========================Acyclic_data_C1 处理=====================//
      spi_read((u32)&dpram.L_command2,1);
    dpram.L_command2 = SPI1_Buffer_Rx[4];
      spi_read((u32)&dpram.R_command2,1);
        dpram.R_command2 = SPI1_Buffer_Rx[4];
   if((dpram.L_command2&0x01) == 0x01 )
     {
         dpram.L_command2 &= 0xFE;
           spi_write((u32)&dpram.L_command2,1, dpram.L_command2);
           spi_read((u32)&dpram.AcycData_DPv1c1,4);                                    //FF FF
81 00 5E 01 02 08
          dpram.AcycData_DPv1c1[0]=SPI1_Buffer_Rx[4];
          dpram.AcycData_DPv1c1[3]=SPI1_Buffer_Rx[7];
//dpv1c1 数据长度
      if(dpram.AcycData_DPv1c1[0]==0x5E)
      {
             spi_read((u32)&dpram.AcycData_DPv1c1,4);
             CopySPI_RxBufToDPRAM_DPV1C1(&SPI1_Buffer_Rx[4],4);
             spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],6)>>8;
          spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],6);
      }
      if(dpram.AcycData_DPv1c1[0]==0x5F)
      {
             spi_read((u32)&dpram.AcycData_DPv1c1,dpram.AcycData_DPv1c1[3]+4);          //读
取整个 dpv1c1 报文
CopySPI_RxBufToDPRAM_DPV1C1(&SPI1_Buffer_Rx[4],dpram.AcycData_DPv1c1[3]+4);
spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c1[3]+6)>>8;
spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c1[3]+6);
      }
      if(((spiRx_CRC_H==SPI1_Buffer_Rx[8])&&(spiRx_CRC_L==SPI1_Buffer_Rx[9]))||
((spiRx_CRC_H==SPI1_Buffer_Rx[8+SPI1_Buffer_Rx[7]])&&(spiRx_CRC_L==SPI1_Buffer_Rx[9+SPI1_Buf
fer_Rx[7]])) )
```

```
    {
        //dpram.R_status2 &= 0xEF;
        //spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
         Acyclic_data_C1_SPIMode();
         dpram.R_command2 |= 0x01;
           spi_write((u32)&dpram.R_command2,1, dpram.R_command2);
       }
     else
     { //写 1 到 B000D.4;从接口板获得的非循环数据 CRC 错
        // dpram.R_status2 |= 0x10;
        // spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
       }
    }
```

<15>、Acyclic_data_C2 处理

```
  //=========================Acyclic_data_C2 处理======================//
   if((dpram.L_command2&0x02) == 0x02 )
     {
      dpram.L_command2 &= 0xFD;
        spi_write((u32)&dpram.L_command2,1, dpram.L_command2);
        spi_read((u32)&dpram.AcycData_DPv1c2,4);                                    //FF FF
81 00 5E 01 02 08
        dpram.AcycData_DPv1c2[0]=SPI1_Buffer_Rx[4];
        dpram.AcycData_DPv1c2[3]=SPI1_Buffer_Rx[7];                                 //dpv1c1
数据长度
     if(dpram.AcycData_DPv1c2[0]==0x5E)
     {
           spi_read((u32)&dpram.AcycData_DPv1c2,4);
           CopySPI_RxBufToDPRAM_DPV1C2(&SPI1_Buffer_Rx[4],4);
           spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],6)>>8;
         spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],6);
     }
     if(dpram.AcycData_DPv1c2[0]==0x5F)
     {
        spi_read((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4);           //读取整
个 dpv1c1 报文
CopySPI_RxBufToDPRAM_DPV1C2(&SPI1_Buffer_Rx[4],dpram.AcycData_DPv1c2[3]+4);

           spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c2[3]+6)>>8;
         spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c2[3]+6);
     }
     if(dpram.AcycData_DPv1c2[0]==0x51)
       {
        spi_read((u32)&dpram.AcycData_DPv1c2,dpram.AcycData_DPv1c2[3]+4);           //读取整
个 dpv1c1 报文
CopySPI_RxBufToDPRAM_DPV1C2(&SPI1_Buffer_Rx[4],dpram.AcycData_DPv1c2[3]+4);
spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c2[3]+6)>>8;
spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.AcycData_DPv1c2[3]+6);
       }
     if(((spiRx_CRC_H==SPI1_Buffer_Rx[8])&&(spiRx_CRC_L==SPI1_Buffer_Rx[9]))||
((spiRx_CRC_H==SPI1_Buffer_Rx[8+SPI1_Buffer_Rx[7]])&&(spiRx_CRC_L==SPI1_Buffer_Rx[9+SPI1_Buf
fer_Rx[7]])) )
    {
         //dpram.R_status2 &= 0xF7;
         Acyclic_data_C2_SPIMode();
         dpram.R_command2 |= 0x02;
           spi_write((u32)&dpram.R_command2,1, dpram.R_command2);
       }
     else
```

```
      {
            //dpram.R_status2 |= 0x08;
        } // 写 1 到 B000D.3;从接口板获得的 IM 数据 CRC 错
     }
  //=================================================================//
    ABORTDPRAM_SPIMODE;
}
```

（13）用户参数及配置数据处理

```
/*
void EXTI9_5_IRQHandler(void)
{
  u8 i;
  u8 spiRx_CRC_H=0 , spiRx_CRC_L=0;
 if(EXTI_GetITStatus(EXTI_Line6) != RESET)
 {
    GETDPRAM_SPIMODE;
    spi_read((u32)&dpram.L_command1,1);
  dpram.L_command1 = SPI1_Buffer_Rx[4];
  if((dpram.L_command1&0x02) == 0x02) //接口板用户参数数据有效标志       B000E.1
    { //PORTE |=0x40;
      spi_read((u32)&dpram.PrmData_DP,1);           //读取整个 dpv1c1 报文
    dpram.PrmData_DP[0] = SPI1_Buffer_Rx[4];
    spi_read((u32)&dpram.PrmData_DP,dpram.PrmData_DP[0]+1);
    CopySPI_RxBufToDPRAM_PRMData(&SPI1_Buffer_Rx[5],dpram.PrmData_DP[0]);
        spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.PrmData_DP[0]+3)>>8;
          spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.PrmData_DP[0]+3);
        //PORTE &=0xBF;
        if(((SPI1_Buffer_Rx[2]&0xE0)!=STX_SPIReply_NOK)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[dpram.PrmData_DP[0]+5])&&(spiRx_CRC_L==SPI1_Buffer_Rx[dpram.PrmData_DP[0]+6])
)
        {
 dpram.R_status2 &= 0xFD ; //从接口板获得的用户参数数据 CRC 正确 B000D.1 =0
 spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
New_Prmdata_Len = dpram.PrmData_DP[0];
for(i=0;i<dpram.PrmData_DP[0];i++){New_Prmdata_Buf[i] = dpram.PrmData_DP[i+1];}//0.477ms(比本
地拿多 0.124ms)
          //PORTE &=0xBF;
if(New_Prmdata_Len<=MaxPrmLen)  //新参数化数据长度 小于等于 初始化最大长度 237
            {
                dpram.R_status1 &= 0xFE;
                spi_write((u32)&dpram.R_status1,1, dpram.R_status1);
              }   //用户判断用户参数数据正确     B000C.0 =0
            else
            {
                dpram.R_status1 |= 0x01;
                spi_write((u32)&dpram.R_status1,1, dpram.R_status1);
              }   //用户判断用户参数数据错误     B000C.0 =1
         }
      else
    {
    dpram.R_status2 |= 0x02 ; //从接口板获得的用户参数数据 CRC 错误    B000D.1 =1
    dpram.R_status1 |= 0x01 ; //用户判断用户参数数据错误              B000C.0 =1
             spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
             spi_write((u32)&dpram.R_status1,1, dpram.R_status1);
         }

    dpram.R_command1 |= 0x80;  //设置用户板用户参数判断结果回传标志    B0010.7 =1
```

```
    //用户板已将用户参数判断结果存入        B000C.0

dpram.L_command1 &= 0xFD ; //清除接口板用户参数数据有效标志        B000E.1 =0
spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
spi_write((u32)&dpram.L_command1,1, dpram.L_command1);
        //NewPrm_Handled_Flag = 1;
        // *LED_OUT1 = 0x55;
      }
  else
    {
       //异常处理
      }
       spi_read((u32)&dpram.L_command1,1);
     dpram.L_command1 = SPI1_Buffer_Rx[4];

  if((dpram.L_command1&0x04) == 0x04) //接口板配置数据有效标志            B000E.2
    {
      spi_read((u32)&dpram.CfgData_DP,3);
            dpram.CfgData_DP[0] = SPI1_Buffer_Rx[4];
            dpram.CfgData_DP[1] = SPI1_Buffer_Rx[5];
            dpram.CfgData_DP[2] = SPI1_Buffer_Rx[6];
            spi_read((u32)&dpram.CfgData_DP,dpram.CfgData_DP[2]+3);
    CopySPI_RxBufToDPRAM_CFGData(&SPI1_Buffer_Rx[7],dpram.CfgData_DP[2]);
            spiRx_CRC_H = CRC_avr(&SPI1_Buffer_Rx[2],dpram.CfgData_DP[2]+5)>>8;
          spiRx_CRC_L = CRC_avr(&SPI1_Buffer_Rx[2],dpram.CfgData_DP[2]+5);
        if(((SPI1_Buffer_Rx[2]&0xE0)!=STX_SPIReply_NOK)&&(spiRx_CRC_H==
SPI1_Buffer_Rx[dpram.CfgData_DP[2]+7])&&(spiRx_CRC_L==SPI1_Buffer_Rx[dpram.CfgData_DP[2]+8])
)
        {
           dpram.R_status2 &= 0xFE ;       //从接口板获得的配置数据 CRC 正确     B000D.0 =0
             spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
           New_Cfgdata_Len = dpram.CfgData_DP[2];
           for(i=0;i<dpram.CfgData_DP[2];i++){New_Cfgdata_Buf[i] = dpram.CfgData_DP[i+3];}

            if(New_Cfgdata_Len <= MaxCfgLen)//此处，可以根据用户实际需求情况，另外增加判断条
件
            {
                  dpram.R_status1 &= 0xFD ;
                    spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
               }      //用户判断配置数据正确         B000C.1 =0
            else
            {
                  dpram.R_status1 |= 0x02 ;
                  spi_write((u32)&dpram.R_status1,1, dpram.R_status1);
               }      //用户判断配置数据错误         B000C.1 =1
          }
        else
        {
        dpram.R_status2 |= 0x01 ; //从接口板获得的配置数据 CRC 错误      B000D.0 =1
        dpram.R_status1 |= 0x02 ;        //用户判断配置数据错误          B000C.1 =1
       spi_write((u32)&dpram.R_status2,1, dpram.R_status2);
       spi_write((u32)&dpram.R_status1,1, dpram.R_status1);
          }
 dpram.R_command1 |= 0x40;  //设置用户板配置数据判断结果回传标志    B0010.6 =1
                              //用户板已将配置数据判断结果存入        B000C.1
 dpram.L_command1 &= 0xFB ; //清除接口板配置数据有效标志            B000E.2 =0
  spi_write((u32)&dpram.R_command1,1, dpram.R_command1);
  spi_write((u32)&dpram.L_command1,1, dpram.L_command1);
```

```
      }
  else
    {
        //异常处理
      }
    ABORTDPRAM_SPIMODE; //PORTE &=0xBF;
    EXTI_ClearITPendingBit(EXTI_Line6);
  }
}
```

第七章 UART 接口

一、通讯协议

1. M 卡与用户 MCU 的数据交换

M 卡与用户 MCU（如 M 卡的评估板 DPM-VB-V10）之间的数据交换，通过 UART 串口连接。Txd，Rxd，为 TTL 电平，半双工。

2. 异步串口数据帧格式

每 1 个字节用 11 bits 传送：1 个启始位、8 个数据位、1 个偶校验位、1 个停止位。

3. 通信方式

应答方式：用户 MCU 主动询问，M 卡被动回答。

4. 初始化报文

读报文：MCU: Inquire

起始符	数据首地址	数据长度	报文 CRC 校验和
1 字节	2 字节	1 字节	2 字节

M 卡：Response

起始符	错误码	DPRAM 数据	CRC 校验和
100000xx	1 字节	1~245 字节	2 字节

写报文：MCU: Inquire

起始符	数据首地址	数据长度	数据	报文 CRC 校验和
1 字节	2 字节	1 字节	1-250 字节	2 字节

M 卡：Response

起始符	错误码	CRC 校验和
101000xx	1 字节	2 字节

注释 1：起始符定义：（包括从站目标地址和读写属性）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
000:CPU 读(R) 001:CPU 写(W) 010:CPU 读写(RW) 100:读正确(CR) 101：写正确(CW) 110:读写正确(CRW) 111:出现错误(ER)			保留 默认为 0	保留 默认为 0	保留 默认为 0	主从目标地址： 00：目标站地址=用户 CPU； 01：目标站地址=1 号 M 卡； 10：目标站地址=2 号 M 卡；	

5．数据交换报文

MCU： Inquire

目标站地	功能码	数据首地址	数据长度	数据	报文 CRC 校验和
1 字节	1 字节	2 字节	1 字节	1-250 字节	2 字节

M 卡：Response

目标站地址	功能码	数据首地址	数据长度	数据	报文CRC校验和
1 字节	1 字节	2 字节	1 字节	1-250 字节	2 字节

注释：

- **[目标站地址]：**MCU 地址（00）,M 卡站地址(取值范围：1～2)
- **[功能码]：**此字段用来区分通信时使用的不同类型报文
- **[数据首地址]：**不同类型数据在 DPRAM 中的首地址详细 RAM 地址定义见“第二章 DPRAM 数据结构”
- **[数据长度]：**报文中包含的用户数据长度（1-250 字节）
- **[数据]：**包括初始化数据、诊断数据长度、用户参数数据据、CFG 数据长度、,V1C1/C2 非循环数据 PROFIBUS 输入输出数据
- **[报文 CRC 校验和]：**从[目标站地址]到[数据]字段的所有数据字节的 CRC 校验和(多项式取值：0xA001)

6. 通讯错误报文：

MCU: Inquire

目标站地址	功能码	数据首地址	数据长度	数据	报文CRC校验和
1 字节	1 字节	2 字节	1 字节	1-250 字节	2 字节

M 卡：Response

目标站地址	功能码	错误码	报文 CRC 校验和
1 字节	1 字节	1 字节	2 字节

注释 1：M 卡返回报文中的【错误码】定义

Bit7	Reserved			
Bit6	Reserved			
Bit5	Reserved			
Bit4	Reserved			
Bit3	报文长度是否合法	M 卡：W1/0 MCU：R1/0	1：长度不合法	0：长度合法
Bit2	数据长度是否超界：	M 卡：W1/0 MCU：R1/0	1：数据长度超界	0:数据长度不超界
Bit1	首地址是否合法：	M 卡：W1/0 MCU：R1/0	1：表示首地址不合法	0:表示首地址合法
Bit0	报文 CRC 校验错误标志位	M 卡:W1/0 MCU：R1/0	1:CRC 校验错误	0：CRC 校验正确

注释 2：用户 MCU 发送请求报文中的【功能码】定义

Bit7 — Bit2	Reserved			
Bit1	诊断数据更新状态	MCU：W 1/0 M 卡：R 1/0	1：诊断数据有更新	0：诊断数据无更新
Bit0	Reserved			

注释 3：M 卡返回报文中的【功能码】定义

Bit7	通信错误状态	M 卡：W1/0 MCU：R1/0	1：通信有错误	0：通信无错误
Bit6	Reserved			
Bit5	CFG 数据更新状态	M 卡：W1/0 MCU：R1/0	1：CFG 数据有更新	0：CFG 数据无更新
Bit4	PRM 数据更新状态	M 卡：W1/0 MCU：R1/0	1：PRM 数据有更新	0：PRM 数据无更新
Bit3	DPV1-C2 更新状态	M 卡：W1/0 MCU：R1/0	1：V1/C2 有更新	0：V1/C2 无更新
Bit2	DPV1-C1 更新状态	M 卡： W1/0 MCU： R1/0	1：V1/C1 有更新	0：V1/C1 无更新
Bit1	Reserved			
Bit0	DPV0 数据交换状态	M 卡：W1/0 MCU：R1/0	1：处于 DP 数据交换状态	0：退出 DP 数据交换状态

注释 4：读写地址定义

<table>
<tr><td colspan="16">读写 DPRAM 地址</td></tr>
<tr><td colspan="8">高字节</td><td colspan="8">低字节</td></tr>
<tr><td>Bit7</td><td>Bit6</td><td>Bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td colspan="5">保 留</td><td colspan="3">DPRAM 访问首地址高 3 位</td><td colspan="8">DPRAM 访问首地址低 8 位</td></tr>
</table>

二、C 源代码说明

1. “DSuart.h”说明

```
//----------------------------UARTMODE---------------------//
#ifndef __DSuart_H
#define __DSuart_H
#include <stm32f10x.h>                  /* STM32F4xx Definitions             */
#define   SPI1_NSS_SET() PAout(1)
#define   SPI1_NSS_RESET() PAout(0)
#define   GetDpramControl    0x02
#define   AbortDpramControl  0x00
#define   STX_SPIReply_NOK  0xE0
#define   GETDPRAM_SPIMODE    getDpram_SPIMode()
#define   ABORTDPRAM_SPIMODE  abortDpram_SPIMode()
define   SPI_CS_ON   GPIO_ResetBits(GPIOA,GPIO_Pin_4)
#define   SPI_CS_OFF  GPIO_SetBits(GPIOA,GPIO_Pin_4)
#define BufferSize 11
#define RxBufferSize 255
//----------------------------UARTMODE---------------------//
//#define   UARTMode_ADDR1  1
//#define   UARTMode_ADDR2  2
#define   STX_UARTReply_NOK  0xE0
#define   GETDPRAM_UARTMODE    getDpram_DSuartMode()
#define   ABORTDPRAM_UARTMODE  abortDpram_DSuartMode()
//-------COM2    ------------------------------------------------//
#define   SET_COM2_RTS     GPIO_SetBits(GPIOA, GPIO_Pin_1);
#define   RSET_COM2_RTS    GPIO_ResetBits(GPIOA, GPIO_Pin_1);//USART RTS 清零
#define   SET_COM2_TCIE    USART_ITConfig(USART2,USART_IT_TC, ENABLE)
#define   SET_COM2_RXNEIE  USART_ITConfig(USART2,USART_IT_RXNE, ENABLE)
#define   Clear_COM2_TCIE    USART_ClearITPendingBit(USART2,USART_IT_TC)
#define   Clear_COM2_RXNEIE  USART_ClearITPendingBit(USART2,USART_IT_RXNE)
#define   SET_COM2_TE  (USART2->CR1 |= 0x0008)
#define   SET_COM2_RE  (USART2->CR1 |= 0x0004)
#define   RSET_COM2_TE  (USART2->CR1 &= 0xFFF7)
#define   RSET_COM2_RE  (USART2->CR1 &= 0xFFFB)
#define   COM2_SR_PE    (USART2->SR & USART_FLAG_PE)
//----COM3    ----------------------------------------//
#define   SET_COM3_TCIE    USART_ITConfig(USART3,USART_IT_TC, ENABLE)
#define   SET_COM3_RXNEIE  USART_ITConfig(USART3,USART_IT_RXNE, ENABLE)
#define   Clear_COM3_TCIE    USART_ClearITPendingBit(USART3,USART_IT_TC)
#define   Clear_COM3_RXNEIE  USART_ClearITPendingBit(USART3,USART_IT_RXNE)
#define   SET_COM3_TE  (USART3->CR1 | 0x0008)
#define   SET_COM3_RE  (USART3->CR1 | 0x0004)
#define   RSET_COM3_TE  (USART3->CR1 & 0xFFF7)
#define   RSET_COM3_RE  (USART3->CR1 & 0xFFFB)
#define   COM3_SR_PE    (USART3->SR & USART_FLAG_PE)
//-------------------- FC ---------------------------------//
#define  FC_DPDataEx 0x01
#define  FC_DPV0In   0x00
#define  FC_DPDiag   0x02
#define  FC_DPV1C1   0x04
#define  FC_DPV1C2   0x08
#define  FC_DPPRM    0x10
#define  FC_DPCFG    0x20
//-----------------------------------------------------------//
```

```
void UARTMode_GPIO_Init(void);
void UARTModeRS232_Init(void);
void reset_DSDPV1_UART(void);
void COMstart_trans(void);
void COM2_Init(uint32_t baudrate,uint16_t datalen,uint16_t stopbit,uint16_t parity);
void NVIC_Config(void);
void Y_HardWare_InitData_DSuartMode(void);
void Y_SoftWare_InitData_DSuartMode(void);
void DSDPV1_DSUART1_PrmCfg_EXIT_config(void);
void EXTI_Handler(void);
//void DSuart_ReadCFGData(void);
u8 DSuart_ReadCFGData(void);
void Acyclic_data_C1_DSuartMode(void);
void Acyclic_data_C2_DSuartMode(void);
void DPV0V1_DataEx_DSuartMode(void);
#endif  /* __DSuart_H */
```

2. “DSuart.c”

（1）结构体定义

```
//#include "serial.h"
//#include "spi.h"
#include "fsmc_sram.h"
#include "DSuart.h"
struct
   {
     uint8_t p;
     uint8_t box[260];
     uint8_t pend;
   }DSuart_tr;
struct
   {
     uint8_t p;
     uint8_t box[260];
     uint8_t pend;
     uint8_t ok;
   }DSuart_re;
extern uint8_t   SPISlave_Addr;
extern uint8_t   NewPrmData_HandleMode,NewCfgData_HandleMode;
extern uint8_t   EnableDPv1_Flag;
extern uint8_t   dsdpv1_status;
extern uint8_t diag_input[240];
extern uint8_t v0_output[244];  //V0 输出数据区
extern uint8_t v0_input[244];   //V0 输入数据区
extern uint8_t v1_output[240];  //V1 输出数据区
extern uint8_t v1_input[240];   //V1 输入数据区
extern uint8_t IM_output[64];   //IM 输出数据区
extern uint8_t IM_input[64];    //IM 输入数据区
extern uint8_t im0_body[64];
extern uint8_t IM_CallHeader[4];
extern uint8_t  DiagData_input[3];
extern uint8_t  New_Prmdata_Len,New_Cfgdata_Len;
extern uint8_t  New_Prmdata_Buf[MaxPrmLen],New_Cfgdata_Buf[MaxCfgLen];
extern uint8_t  Tn,Tn1,DIO_7;
extern uint8_t  ExtFlag,StaFlag;
extern uint8_t  ExtDiagData_input[3],StaDiagData_input[3];
```

```
extern uint8_t  current_diag_len,flag_diag;
extern uint8_t  si_status,im_status;
extern uint8_t  c_slot,c_index,c_len,c_subindex_H,c_subindex_L,c_rw,c_commu;
extern SLOT_INDEX slot_index[MAX_SI_NUM];
extern u8 PValue;
extern uint8_t UARTMode_ADDR1;
u8 COM2_a_time;
u8  COM2_auto_t;
u8  COM2_automt;
u8  COM2_rs232_prm[100];
u8  COM2_tn_1,COM2_tn;
u8  COM2_re;
u16 COM2_re_len,COM2_relen;
u8  COM2_ODD_EVEN,COM2_YES_OE;
u8  COM2_error_code;
u8  SendOnce;
u16 COM2_M_h,COM2_M_l;
u16 COM2_M_c,COM2_M_cc;
u16 COM2_M_c1,COM2_M_cc1;
u16 COM2_Gap_Period,COM2_Gap_Prescaler;
u16 COM2_datalen,COM2_stopbit,COM2_parity;
u32 COM2_baudrate;
u8  SW15_BaudRate;
u8  SW14_Interval;
u8  DZ_cont=0;
```

（2）GPIO 初始化

```
void UARTMode_GPIO_Init(void)
{
    /*u8 i;
    u8 x1,x2,x3;
    u8 y1,y2,y3;
    u8 z1,z2;
    u8 spi1_Add0,spi1_Add1;
  */
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO|RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC
_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD
|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_WWDG, ENABLE);
//--------------UartMode Interval ------------------------------------------//
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15 ;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOE, &GPIO_InitStructure);//用于 报文中字符间隔 PE(13 14 15)
//--------------UartMode BaudRate ------------------------------------------//
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOD, &GPIO_InitStructure);//用于  波特率  PD(8 9 10)
//======================SPI/Uart 地址=====================================//
/*
    x1=SW15_1;
    x2=SW15_2;
    x3=SW15_3;
    y1=SW14_1;
    y2=SW14_2;
    y3=SW14_3;
```

```
    */
    SW15_BaudRate =     DSDPV1_UartMode_Baud;
  SW14_Interval =   DSDPV1_UartMode_Interval;
}
```

（3）COM 口初始化

```
void COM2_Init(uint32_t baudrate,uint16_t datalen,uint16_t stopbit,uint16_t parity)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 ; //USART2_RTS(PA.1)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /*
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 ; //USART2_RTS(PA.1)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IPD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    */
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_2;  //USART2_TX (PA.2)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_3; //USART2_RX (PA.3)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    USART_InitStructure.USART_BaudRate = baudrate;
    USART_InitStructure.USART_WordLength = datalen;
    USART_InitStructure.USART_StopBits = stopbit;
    USART_InitStructure.USART_Parity = parity;
    USART_InitStructure.USART_HardwareFlowControl =   USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
  USART_Init(USART2,&USART_InitStructure);
    Clear_COM2_TCIE;
    USART_Cmd(USART2, ENABLE);}
void COM3_Init(uint32_t baudrate,uint16_t datalen,uint16_t stopbit,uint16_t parity)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,ENABLE);
    /*
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9 ; //USART3_RTS(PB.9)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    */
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_10;  //USART3_TX(PB.10)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_11; //USART3_RX(PB.11)
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    USART_InitStructure.USART_BaudRate = baudrate;
```

```
    USART_InitStructure.USART_WordLength = datalen;
    USART_InitStructure.USART_StopBits = stopbit;
    USART_InitStructure.USART_Parity = parity;
    USART_InitStructure.USART_HardwareFlowControl =      USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
  USART_Init(USART3,&USART_InitStructure);
    USART_Cmd(USART3, ENABLE);
}
void TIM2_Init(uint16_t period,uint16_t prescaler)
{
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
  TIM_DeInit(TIM2);
 // TIM_TimeBaseStructure.TIM_Period=1; // 自动重装载寄存器的值
T=(TIM_Period+1)*(TIM_Prescaler+1)/72MHz
 // TIM_TimeBaseStructure.TIM_Prescaler= 359; //时钟预分频数 T= 2*360/72MHz=10us
    TIM_TimeBaseStructure.TIM_Period=period*72-1;
    TIM_TimeBaseStructure.TIM_Prescaler=(prescaler-1);//T
=(period-1+1)(prescaler-1+1)*72/72M=(period*prescaler) us
  TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1; //采样分频 :不分频 CKD=0
  TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//计数方式
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ClearITPendingBit(TIM2, TIM_IT_Update);//清除溢出中断标志
  TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
  TIM_Cmd(TIM2, DISABLE); //关闭定时器
}
void TIM3_Init(uint16_t period,uint16_t prescaler)
{
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
  TIM_DeInit(TIM3);
 // TIM_TimeBaseStructure.TIM_Period=1; // 自动重装载寄存器的值
T=(TIM_Period+1)*(TIM_Prescaler+1)/72MHz
 // TIM_TimeBaseStructure.TIM_Prescaler= 359; //时钟预分频数 T= 2*360/72MHz=10us
    TIM_TimeBaseStructure.TIM_Period=(period-1);
    TIM_TimeBaseStructure.TIM_Prescaler=prescaler*72-1;//T
=(period-1+1)(prescaler-1+1)*72/72M=(period*prescaler) us
  TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1; //采样分频 :不分频 CKD=0
  TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//计数方式
  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
    TIM_ClearITPendingBit(TIM3, TIM_IT_Update);//清除溢出中断标志
  TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);
  TIM_Cmd(TIM3, DISABLE); //关闭定时器
}
void NVIC_Config(void)
{
  NVIC_InitTypeDef  NVIC_InitStructure;
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
  NVIC_InitStructure.NVIC_IRQChannel                   = USART1_IRQn;//LCD
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x4;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority        = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd                = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel                   = USART2_IRQn;//Profibus DP1 UARTMode
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority        = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd                = ENABLE;
```

```
  NVIC_Init(&NVIC_InitStructure);
  /*
  NVIC_InitStructure.NVIC_IRQChannel                    = USART3_IRQn;//Profibus DP2 UARTMode
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority         = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelCmd                 = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
    */
    NVIC_InitStructure.NVIC_IRQChannel                    = TIM2_IRQn; //UART2
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2; //抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority         = 0; //副优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd                 = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
NVIC_InitStructure.NVIC_IRQChannel                    = TIM3_IRQn; //通道 TIM2
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3; //抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority         = 0; //副优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd                 = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
NVIC_InitStructure.NVIC_IRQChannel                    = TIM4_IRQn; // LCD COM1 Tx
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 5; //抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority         = 0; //副优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd                 = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
NVIC_InitStructure.NVIC_IRQChannel                    = TIM5_IRQn; // LCD COM1 Rx
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 6; //抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority         = 0; //副优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd                 = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

（4）配置 COM 口

```
void COM2_rs232_config(void)
{
 u8 temp1,temp2,i;
/*=========================================================*/
/*  RS-232 口配置 if master then  禁止接收 else 启动接收
*/
 uint32_t BR[10]={9600,19200,38400,57600,115200,230400,460800,932100,1843200};
 uint16_t BRR[20]={401,10, 200,10, 100,10, 66,10, 33,10, 16,10, 8,10, 4,10, 2,10};//3.5 个字符
（每个字节按 11 个位计）T2
 // uint16_t BRR[20]={228,10, 114,10, 58,10, 38,10,18,10, 10,10, 4,10, 2,10, 1,10};//2 个字符
（每个字节按 11 个位计）T
 //uint16_t BRR[20]={114,10, 57,10, 29,10, 19,10, 9,10, 5,10, 2,10, 1,10, 1,6};   //1 个字符（每
个字节按 11 个位计）T
/*以上是：对应不同波特率时，3.5 个字节（每个字节按 11 个位计）T2 的 TIM_Perio
 uint8_t  mcc[15]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
 i     = COM2_rs232_prm[0]; //波特率 选择
 temp1 = COM2_rs232_prm[1]; //数据位/停止位/校验格式 选择
 temp2 = COM2_rs232_prm[2]; // Modbus 主从模式 选择
 COM2_M_c= mcc[i];
 COM2_YES_OE   = temp1;
 COM2_ODD_EVEN = temp1;


 COM2_baudrate=BR[i];
 COM2_Gap_Period=BRR[i*2];
 COM2_Gap_Prescaler=BRR[i*2+1];
 //COM2_Gap_Period=COM2_baudrate;
```

```
//COM2_Gap_Prescaler=BRR[i*2+1];
switch (temp1)
{
   /*
 0:"8 DataLen,1 StopBit,No Parity"
 1:"8 DataLen,1 StopBit,Even Parity"
 2:"8 DataLen,1 StopBit,Odd Parity"
 3:"7 DataLen,1 StopBit,Even Parity"
 4:"7 DataLen,1 StopBit,Odd Parity"
 5:"8 DataLen,Address/Data Flag"
 6:"8 DataLen,2 StopBit,No Parity"
 7:"7 DataLen,2 StopBit,No Parity"
 8:"7 DataLen,2 StopBit,Even Parity"
 9:"7 DataLen,2 StopBit,Odd Parity"
 */
   case 0: //8,n,1
         {
                COM2_datalen = USART_WordLength_8b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_No;
               }
           break;
 case 1: //8,e,1
         {
                COM2_datalen = USART_WordLength_9b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_Even;
               }
           break;
   case 2: //8,o,1
         {
                COM2_datalen = USART_WordLength_9b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_Odd;
               }
           break;
   case 3: //7,e,1
         {
                COM2_datalen = USART_WordLength_8b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_Even;
               }
           break;
 case 4: //7,o,1
         {
                COM2_datalen = USART_WordLength_8b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_Odd;
               }
           break;
 case 5: //8 DataLen,Address/Data Flag
         {
                COM2_datalen = USART_WordLength_8b;
         COM2_stopbit = USART_StopBits_1;
         COM2_parity  = USART_Parity_No;
               }
           break;
 case 6:  //8,n,2
```

```
        {
                COM2_datalen = USART_WordLength_8b;
        COM2_stopbit = USART_StopBits_2;
        COM2_parity  = USART_Parity_No;
              }
          break;
 case 7: //7,n,2
        {
                COM2_datalen = USART_WordLength_8b;
        COM2_stopbit = USART_StopBits_2;
        COM2_parity  = USART_Parity_No;
              }
          break;
 case 8: //7,e,2
        {
                COM2_datalen = USART_WordLength_8b;
        COM2_stopbit = USART_StopBits_2;
        COM2_parity  = USART_Parity_Even;
              }
          break;
 case 9: //7,o,2
        {
        COM2_datalen = USART_WordLength_8b;
        COM2_stopbit = USART_StopBits_2;
        COM2_parity  = USART_Parity_Odd;
              }
          break;
   default:
          break;
          }
COM2_Init(COM2_baudrate,COM2_datalen,COM2_stopbit,COM2_parity);
 switch (temp2)
 {//temp2:M_S   temp1:Odd_Even

case 0:
  {
    SET_COM2_TCIE;
    SET_COM2_RXNEIE;
                }
                break;
               case 1:
                 {

                   SET_COM2_RE;
                   RSET_COM2_TE;
                   SET_COM2_RXNEIE;

                                 }
                      break;
             default:
                        break;

}
TIM_Cmd(TIM2, DISABLE);
TIM_Cmd(TIM3, DISABLE);
 switch (COM2_a_time)
 {
    case 0:{ COM2_M_h=5;COM2_M_l=1000;COM2_M_c1=1;};break;//10ms
```

```
     case 1:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=1;};break;//10ms
     case 2:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=2;};break;//20ms
    case 3:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=3;};break;//30ms
    case 4:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=4;};break;//40ms
    case 5:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=5;};break;//50ms

    case 6:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=6;};break;//60ms
    case 7:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=7;};break;//70ms
    case 8:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=8;};break;//80ms
    case 9:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=9;};break;//90ms
    case 10:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=10;};break;//100ms
    case 11:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=11;};break;//110ms
    case 12:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=12;};break;//120ms
    case 13:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=13;};break;//130ms
    case 14:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=14;};break;//140ms
    case 15:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=15;};break;//150ms

    case 16:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=20;};break;//200ms
    case 17:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=30;};break;//300ms
    case 18:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=40;};break;//400ms
    case 19:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=50;};break;//500ms
    case 20:{ COM2_M_h=10;COM2_M_l=1000;COM2_M_c1=60;};break;//600ms
         default:break;
   }
    if(COM2_rs232_prm[2] == 0)    /**** 主站，置发送允许位_接收完成位=1 ******/
    {
      }
    else
    {
    }
}
```

（5）启动 COM 发送进程

```
void COMstart_trans(void)
{
/*============================================================*/
/* 启动发送进程                                               */
/*============================================================*/
    uint8_t x;
    SET_COM2_RTS;
    SET_COM2_TE;
    DSuart_tr.p=0;
    x=DSuart_tr.box[DSuart_tr.p];
  USART_SendData(USART2,x); //启动发送
    DSuart_tr.p++;
}
void UARTModeRS232_Init(void)
{
      uint16_t i;
     for(i=0;i<260;i++)
    {
        DSuart_tr.box[i]=0;
        DSuart_re.box[i]=0;
    }
    DSuart_tr.p=DSuart_tr.pend=0;
    DSuart_re.p=DSuart_re.pend=DSuart_re.ok=0;
    switch (SW15_BaudRate)
    {
```

```
        case 0:{COM2_rs232_prm[0]=0x00;}break;//9.6k
        case 1:{COM2_rs232_prm[0]=0x01;}break;//19.2k
        case 2:{COM2_rs232_prm[0]=0x02;}break;//38.4k
        case 3:{COM2_rs232_prm[0]=0x03;}break;//57.6k
        case 4:{COM2_rs232_prm[0]=0x04;}break;//115.2k
        case 5:{COM2_rs232_prm[0]=0x05;}break;//230.4k
          case 6:{COM2_rs232_prm[0]=0x06;}break;//460.8k
          case 7:{COM2_rs232_prm[0]=0x08;}break;//1.8432M
        default:break;
        }
        //------------------串口参数获取-----------------------------------//
        //COM2_rs232_prm[0]=0x04; //9600
        COM2_rs232_prm[1]=0x01;   //even
        COM2_rs232_prm[2]=0x00;   //0 主站
        COM2_a_time=1;          //10ms
    //-----------------------------------------------------------------//
        COM2_rs232_config();// 配置 RS-232 接口
        COM2_error_code=0;
      COM2_tn_1=COM2_tn=0;
      COM2_re=0;
      COM2_auto_t=1;
      COM2_automt=1;
      COM2_relen=0;//按长度接收
}
void reset_DSDPV1_UART(void)
{
  uint16_t i;//复位 SPI1
    for (i=0;i<65000;i++){GPIOE->BSRR = GPIO_Pin_4;} //后高电平
    for (i=0;i<65000;i++){GPIOE->BRR  = GPIO_Pin_4;} //先低电平
}
```

（6）设置外部中断

```
void DSDPV1_DSUART1_PrmCfg_EXIT_config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOG,GPIO_PinSource6);
    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);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 7;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
void DSDPV1_DSUART2_PrmCfg_EXIT_config(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOD,GPIO_PinSource12);
    EXTI_InitStructure.EXTI_Line = EXTI_Line12 ;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt ;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling ;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
```

```
    EXTI_Init(&EXTI_InitStructure);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 9;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

（7）复制数据

```
void CopyUART_RxBufToDPRAM_PRMData(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.PrmData_DP[i+1]=*(p+i);}
 }
void CopyUART_RxBufToDPRAM_CFGData(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.CfgData_DP[i+3]=*(p+i);}
 }
void CopyUART_RxBufToDPRAM_DPV0Out(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.OutData_DP[i+1]=*(p+i);}
 }
void CopyUART_RxBufToDPRAM_DPV1C1(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.AcycData_DPv1c1[i]=*(p+i);}
 }
void CopyUART_RxBufToDPRAM_DPV1C2(u8 *p,u16 len)
{
    u8 i;
    for(i=0;i<len;i++)
    {dpram.AcycData_DPv1c2[i]=*(p+i);}
 }
```

（8）只读操作

```
void DSuart_read(u8 slave_addr,u32 dpramMem,u16 readLen)
{
    u16 i=0;
    uint8_t uart_CRC_H,uart_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t uart_CRC[6];
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    uart_CRC[0] = (0x00|slave_addr);
    uart_CRC[1] = dpramMem_H;
    uart_CRC[2] = dpramMem_L;
    uart_CRC[3] = readLen;
    uart_CRC_H  = CRC_avr(uart_CRC,4)>>8;
    uart_CRC_L  = CRC_avr(uart_CRC,4);
    //-----------------------------------------//
```

```
    DSuart_tr.box[i++]=uart_CRC[0];
    DSuart_tr.box[i++]=uart_CRC[1];
    DSuart_tr.box[i++]=uart_CRC[2];
    DSuart_tr.box[i++]=uart_CRC[3];
    DSuart_tr.box[i++]=uart_CRC_H;
    DSuart_tr.box[i++]=uart_CRC_L;
    //--------------------------------------------//
    DSuart_tr.pend=i;
    COMstart_trans();

 }
```

（9）只写操作

```
void DSuart_write(u8 slave_addr,u32 dpramMem,u16 writeLen,u16 data)
{
    u16 i=0;
    uint8_t uart_CRC_H,uart_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t uart_CRC[5];
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    uart_CRC[0] = (0x20|slave_addr);
    uart_CRC[1] = dpramMem_H;
    uart_CRC[2] = dpramMem_L;
    uart_CRC[3] = writeLen;
    uart_CRC[4] = data;
    uart_CRC_H  = CRC_avr(uart_CRC,5)>>8;
    uart_CRC_L  = CRC_avr(uart_CRC,5);
    //--------------------------------------------//
    DSuart_tr.box[i++]=uart_CRC[0];
    DSuart_tr.box[i++]=uart_CRC[1];
    DSuart_tr.box[i++]=uart_CRC[2];
    DSuart_tr.box[i++]=uart_CRC[3];
    DSuart_tr.box[i++]=uart_CRC[4];
    DSuart_tr.box[i++]=uart_CRC_H;
    DSuart_tr.box[i++]=uart_CRC_L;
    //--------------------------------------------//
    DSuart_tr.pend=i;
    COMstart_trans();
```

（10）可读可写操作

```
void DSuart_ReadWrite(u8 slave_addr,u8 *data)
{//此指令只在 DP 输入输出数据长度 都不为 0 时 使用
    u16 i=0,j;
   uint8_t uart_CRC_H,uart_CRC_L;
   uint8_t uart_CRC[250];
   uart_CRC[0] = (0x40|slave_addr);
  for(i=0;i<(*data)+1;i++){uart_CRC[i+1] = *(data+i);}
   uart_CRC_H  = CRC_avr(uart_CRC,(*data)+2)>>8;
    uart_CRC_L  = CRC_avr(uart_CRC,(*data)+2);
//--------------------------------------------//
    for(i=0,j=0;j<(*data)+2;j++)
    {
     DSuart_tr.box[i++]=uart_CRC[j];
    }
```

```
     DSuart_tr.box[i++]=uart_CRC_H;
   DSuart_tr.box[i++]=uart_CRC_L;
//-----------------------------------------//
    DSuart_tr.pend=i;
    COMstart_trans();
}
void DSuart_ReadWrite_N(u8 slave_addr,u8 *data)
{//此指令只在 DP 输入输出数据长度 都不为 0 时 使用
     u8 w,i;
     u8 uartRx_CRC_H,uartRx_CRC_L;
   for(w=0;w==0;)
     {
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
         DSuart_ReadWrite(slave_addr,data);
        }
            if(DSuart_re.ok==1)
            {
                 DSuart_re.ok=0;
        uartRx_CRC_H =  CRC_avr(&DSuart_re.box[0],dpram.OutData_DP[0]+3)>>8;
        uartRx_CRC_L = CRC_avr(&DSuart_re.box[0],dpram.OutData_DP[0]+3);
        if(((DSuart_re.box[0]&0xE0)==0xC0)&&(uartRx_CRC_H==
DSuart_re.box[dpram.OutData_DP[0]+3])&&(uartRx_CRC_L==DSuart_re.box[dpram.OutData_DP[0]+4]))
                {
     CopyUART_RxBufToDPRAM_DPV0Out(&DSuart_re.box[3],dpram.OutData_DP[0]);//复制 DPv0 输出数据
到本地缓存
                          for(i=0;i<dpram.OutData_DP[0];i++)
                {
                              v0_output[i] = dpram.OutData_DP[i+1];
                          }
                }
                else
                {
                    }
               w=1;
        }
            else
          {w=0;}
     }
}

void DSuart_writeBlock(u8 slave_addr,u32 dpramMem,u16 writeLen,u8 *data)
{
    u16 i=0,j;
    uint8_t uart_CRC_H,uart_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t uart_CRC[250];
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    uart_CRC[0] = (0x20|slave_addr);
    uart_CRC[1] = dpramMem_H;
    uart_CRC[2] = dpramMem_L;
    uart_CRC[3] = writeLen;
    for(i=0;i<writeLen;i++){uart_CRC[i+4] = *(data+i);}
    uart_CRC_H  = CRC_avr(uart_CRC,writeLen+4)>>8;
    uart_CRC_L  = CRC_avr(uart_CRC,writeLen+4);
```

```
//-------------------------------------------//
    for(i=0,j=0;j<writeLen+4;j++)
    {
     DSuart_tr.box[i++]=uart_CRC[j];
    }
     DSuart_tr.box[i++]=uart_CRC_H;
   DSuart_tr.box[i++]=uart_CRC_L;
//-------------------------------------------//
    DSuart_tr.pend=i;
    COMstart_trans();
}
u8 DSuart_read_N(u8 slave_addr,u32 dpramMem,u16 readLen)
{
     u8 w;
     u8 temp;
   for(w=0;w==0;)
     {
           if(COM2_auto_t==1)
               {
           COM2_auto_t=0;
           DSuart_read(slave_addr,dpramMem,readLen);
           }
           if(DSuart_re.ok==1)
               {
                   DSuart_re.ok=0;
if(((DSuart_re.box[0]&0xE0)==0x80)&&(DSuart_re.box[1]==0x00))
                    {
                        temp = DSuart_re.box[2];
                    }
                    w=1;
               }
                 else
                 {w=0;}
     }
     return temp ;
}
void DSuart_write_N(u8 slave_addr,u32 dpramMem,u16 writeLen,u16 data)
{
     u8 w;
   u8 temp;
   for(w=0;w==0;)
     {
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
          DSuart_write(slave_addr,dpramMem,writeLen,data);
          w=1;
        }
            if(DSuart_re.ok==1)
            {
                 DSuart_re.ok=0;
                 if(((DSuart_re.box[0]&0xE0)==0xA0)&&(DSuart_re.box[1]==0x00))
               {
            temp = DSuart_re.box[2];
                 }
                    w=1;
        }
            else
```

```
          {w=0;}
     }
}
void DSuart_writeBlock_N(u8 slave_addr,u32 dpramMem,u16 writeLen,u8 *data)
{
     u8 w;
   u8 temp;
   for(w=0;w==0;)
     {
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
         DSuart_writeBlock(slave_addr,dpramMem,writeLen,data);
        }
            if(DSuart_re.ok==1)
            {
                 DSuart_re.ok=0;
                 if(((DSuart_re.box[0]&0xE0)==0xA0)&&(DSuart_re.box[1]==0x00))
               {
            temp = DSuart_re.box[2];
                 }
                    w=1;
        }
            else
          {w=0;}
     }
}
```

（11）配置数据处理

```
u8 DSuart_ReadCFGData(void)
{
     u8 v,w;
     u8 flag=0;
     //for(v=0;v==0;)
     {
     for(w=0;w==0;)
       {
           if(COM2_auto_t==1)
               {
           COM2_auto_t=0;
                 DSuart_read(UARTMode_ADDR1,(u32)&dpram.L_command1,1);
                     flag=1;
           }
           if((DSuart_re.ok==1)&&(flag==1))
               {
                   DSuart_re.ok=0;
                     flag=0;
       if(((DSuart_re.box[0]&0xE0)==0x80)&&(DSuart_re.box[1]==0x00))
                     {
                         dpram.L_command1=DSuart_re.box[2];
                     }
                     w=1;
               }
                 else
                 {w=0;}
          }
       if((dpram.L_command1&0x04) == 0x04 ) //配置数据有效
```

```
        {
        for(w=0;w==0;)
             {
              if(COM2_auto_t==1)
                  {
               COM2_auto_t=0;
                        flag=8;
 DSuart_read(UARTMode_ADDR1,(u32)&dpram.CfgData_DP,2);//读取 DP 输出输入数据长度 0500 0501

              }
                  if((DSuart_re.ok==1)&&(flag==8))
                  {
                       DSuart_re.ok=0;
                           flag=0;
if(((DSuart_re.box[0]&0xE0)==0x80)&&(DSuart_re.box[1]==0x00))
                           {
 dpram.CfgData_DP[0]=DSuart_re.box[2];//OutLen
 dpram.CfgData_DP[1]=DSuart_re.box[3];//InLen
 dpram.OutData_DP[0]=dpram.CfgData_DP[0];                                    dpram.InData_DP[0]
=dpram.CfgData_DP[1];
                          }
                            w=1;
                  }
                    else
                {w=0;}
             }
              v=1;
        }
        else
        {v=0;}
    }
      //return v;
}
```

（12）数据交换 1

```
void DSuart_DataExchange_master(u8 slave_addr,u8 fc,u32 dpramMem,u16 writeLen,u8 *data)
{
    u16 i=0,j;
    uint8_t uart_CRC_H,uart_CRC_L;
    uint8_t dpramMem_H,dpramMem_L;
    uint8_t uart_CRC[250];
    dpramMem= (0x7FF&dpramMem);
    dpramMem_H = (uint8_t)(dpramMem>>8);
    dpramMem_L = (uint8_t)(dpramMem);
    uart_CRC[0] = (0x40|slave_addr);//0x41
    uart_CRC[1] = fc;
    uart_CRC[2] = dpramMem_H;
    uart_CRC[3] = dpramMem_L;
    uart_CRC[4] = writeLen;
    for(i=0;i<writeLen;i++){uart_CRC[i+5] = *(data+i);}
    uart_CRC_H  = CRC_avr(uart_CRC,writeLen+5)>>8;
    uart_CRC_L  = CRC_avr(uart_CRC,writeLen+5);
//-----------------------------------------//
    for(i=0,j=0;j<writeLen+5;j++)
    {
     DSuart_tr.box[i++]=uart_CRC[j];
    }
```

```
     DSuart_tr.box[i++]=uart_CRC_H;
   DSuart_tr.box[i++]=uart_CRC_L;
//-----------------------------------------//
    DSuart_tr.pend=i;
    COMstart_trans();
}

void DSuart_DataExchange_TR(u8 slave_addr,u8 fc,u32 dpramMem,u16 writeLen,u8 *data)
{//此指令只在 DP 输入输出数据长度 都不为 0 时 使用
     u8 w;
     u8 flag=0;
   for(w=0;w==0;)
     {
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
  DSuart_DataExchange_master(slave_addr,fc,dpramMem,writeLen,data);

                flag=0x70;
        }
            if((DSuart_re.ok==1)&&(flag==0x70))
            {
                 DSuart_re.ok=0;
                 flag=0x00;
               w=1;
        }
            else
          {w=0;}
     }
}
void DSuart_Diag_MCUUartSend(void)
{
     u8 w;
     u8 flag=0;
     for(w=0;w==0;)
     {
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
DSuart_DataExchange_master(UARTMode_ADDR1,FC_DPDiag,(u32)&dpram.DiagData_DP,dpram.DiagData_D
P[1]+2, dpram.DiagData_DP);
                flag=0x60;
        }
            if((DSuart_re.ok==1)&&(flag==0x60))
            {
                 DSuart_re.ok=0;
                 w=1;
                 flag=0x00;
        }
            else
            {w=0;}
     }
}
```

（13）用户板硬件初始化

```
//==================用户板硬件初始化==============================//
```

```
void Y_HardWare_InitData_DSuartMode(void)
{
    u8 v,i;
    u8 temp1;
    dpram.R_status1  = 0;
    dpram.R_status2  = 0;
    dpram.R_command1 = 0;
    dpram.R_command2 = 0;
    dpram.UART_TSDR = 0x00;
    for(i=0;i<215;i++) {dpram.InitData_DPv0[i] = 0;}//V0 初始化数据区
    for(i=0;i<245;i++) {dpram.InData_DP[i]     = 0;}//PROFIBUS 输入数据区
    for(i=0;i<240;i++) {dpram.DiagData_DP[i]   = 0;}//Diag 诊断数据区
  for(v=0;v==0;)
  {
        temp1=DSuart_read_N(UARTMode_ADDR1,(u32)&dpram.HardwareInit_Ldpram,1);
        dpram.HardwareInit_Ldpram = temp1;
        if(dpram.HardwareInit_Ldpram == 0xC5)
       {
DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.UART_TSDR,1,   dpram.UART_TSDR);
 DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.R_status1,1,   dpram.R_status1);
DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.R_status2,1,   dpram.R_status2);
DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.R_command1,1,  dpram.R_command1);
DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.R_command2,1,  dpram.R_command2);
DSuart_writeBlock_N(UARTMode_ADDR1,(u32)&dpram.InitData_DPv0,215,dpram.InitData_DPv0);
DSuart_writeBlock_N(UARTMode_ADDR1,(u32)&dpram.InData_DP,     245,dpram.InData_DP);
DSuart_writeBlock_N(UARTMode_ADDR1,(u32)&dpram.DiagData_DP,   240,dpram.DiagData_DP);
            v=1;
       }
       else
         {
          v=0;
         }
  }
}
```

（14）软件初始化

```
//==================初始化================================//
void Y_SoftWare_InitData_DSuartMode(void)
{
    u8 w,v;
     //EXTI_DeInit();//关闭 外部中断
     dpram.InitData_DPv0[0]=22;                          //V0 初始化数据长度字节
     dpram.InitData_DPv0[1]=dp_address();//dp_address();//*ADDRESS;//从站站地址
     dpram.InitData_DPv0[2]=0x0C;                        //ID 号高字节
    dpram.InitData_DPv0[3]=0xC9;                        //ID 号低字节
    dpram.InitData_DPv0[4]=0x0F; //SPC3_Register_status0:0000 fS SYN FREEZE (SSA)
    dpram.InitData_DPv0[5]=0;                           //预留备用
    dpram.InitData_DPv0[6]=0;                           //预留备用
    dpram.InitData_DPv0[7]=244; //48                   //PROFIBUS 输入数据最大可能长度 a
    dpram.InitData_DPv0[8]=244; //48                    //PROFIBUS 输出数据最大可能长度 b
    dpram.InitData_DPv0[9]=238;   //9                   //用户诊断数据长度（≤238 不包含标准 6 字节诊断数据）
    dpram.InitData_DPv0[10]=MaxPrmLen;//MaxPrmLen237//用户参数数据最大可能长度 j（≤237 不包含标准 7 字节参数数据）
    dpram.InitData_DPv0[11]=NewPrmData_HandleMode;  //用户参数正确与否判断方式
    dpram.InitData_DPv0[12]=MaxCfgLen;//MaxCfgLen200//配置数据的最大可能长度 m（≤200）
    dpram.InitData_DPv0[13]=NewCfgData_HandleMode;  //配置数据正确与否判断方式
```

```
    dpram.InitData_DPv0[14]=6;                          //默认 CFG 数据长度 = n
    dpram.InitData_DPv0[15]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[16]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[17]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[18]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[19]=0x1F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[20]=0x2F;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[21]=0x00;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[22]=0x00;                       //默认用户诊断数据长度
   /*
      dpram.InitData_DPv0[14]=8;                         //默认 CFG 数据长度 = n
    dpram.InitData_DPv0[15]=0x40;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[16]=0x7c;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[17]=0x80;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[18]=0x7c;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[19]=0x40;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[20]=0x7c;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[21]=0x80;                       //默认用户诊断数据长度
    dpram.InitData_DPv0[22]=0x7c;                       //默认用户诊断数据长度
     */
   for(v=0;v==0;)
{ DSuart_writeBlock_N(UARTMode_ADDR1,(u32)&dpram.InitData_DPv0,23,dpram.InitData_DPv0);
    if(EnableDPv1_Flag == 1)
      {
        dpram.EnableDPv1_Rdpram=0x1D;                   //V1 功能开启标志 B0005
        DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.EnableDPv1_Rdpram,1,
dpram.EnableDPv1_Rdpram);
      }
      else
      {
         dpram.EnableDPv1_Rdpram=0x00;
         DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.EnableDPv1_Rdpram,1,
dpram.EnableDPv1_Rdpram);
       } //V1 功能开启标志 B0005
       dpram.R_command1 |= 0x04;       //用户板命令字节 1_初始化信息有效标志
        DSuart_write_N(UARTMode_ADDR1,(u32)&dpram.R_command1,1, dpram.R_command1);
       //ABORTDPRAM_UARTMODE;
   //  delay_ms(20);//此处需要等待接口板处理初始化结果，可能需要延时较长时间
(20150508:20ms->10ms)
      for(w=0;w==0;)
        {
          if(COM2_auto_t==1)
              {
          COM2_auto_t=0;
                DSuart_read(UARTMode_ADDR1,(u32)&dpram.L_status1,1);
          }
              if(DSuart_re.ok==1)
              {
                  DSuart_re.ok=0;
                  if(((DSuart_re.box[0]&0xE0)==0x80)&&(DSuart_re.box[1]==0x00))
                  {
                      dpram.L_status1=DSuart_re.box[2];//校验后 再取值
                  }
                  w=1;
              }
                else
              {w=0;}
       }
```

```
        if((dpram.L_status1&0x01)==0x01)
             {
                 for(w=0;w==0;)
            {
              if(COM2_auto_t==1)
                  {
              COM2_auto_t=0;
               DSuart_read(UARTMode_ADDR1,(u32)&dpram.R_command1,1);
              }
                  if(DSuart_re.ok==1)
                  {
                      DSuart_re.ok=0;
if(((DSuart_re.box[0]&0xE0)==0x80)&&(DSuart_re.box[1]==0x00))
                      {
                         dpram.R_command1 = DSuart_re.box[2];//校验后 再取值
                      }
                          w=1;
                  }
                      else
                  {w=0;}
            }
                  v=1;
              }
        else
             {
                 v=0;
             }
   } //for(w)
//DSDPV1_DSUART1_PrmCfg_EXIT_config();
}
```

（15）DPV1/C1 数据处理

```
void Acyclic_data_C1_DSuartMode(void)
{
    u16 i;
    u8  w_slot,w_index,slot_curr;
    u8  len_status;//error_status
    i=0;
    w_slot=w_index=0;
    slot_curr=0;
    len_status=0;
    switch(dpram.AcycData_DPv1c1[0]) //进来的是:0x5E 或 0x5F      {
     case 0x5E :   //进来的是:0x5E Read
         {
             c_slot  = dpram.AcycData_DPv1c1[1];
             c_index = dpram.AcycData_DPv1c1[2];
             c_len   = dpram.AcycData_DPv1c1[3];
             for(i=0;i<MAX_SI_NUM;i++)
             {//1
if((slot_index[i].rw==SI_R)||(slot_index[i].rw==SI_RW))  //仅在具有读属性的槽-索引内寻找
               {//2
                    if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                    {//3
                    slot_curr=0x01;//声明列表中出现过正确的槽号

                         if(c_index==slot_index[i].index)
                         {si_status=SI_OK;break;}
```

```
                    else
                    {
                         si_status=SI_NOK;
                           w_index=1;
                       }
            }
                    else
                    {
                       if(slot_curr==0)
                      {
                        si_status=SI_NOK;
                        w_slot=1;
                      }
                    else
                      {
                        si_status=SI_NOK;
                        w_slot=0;
                      }
                    }//3
      }
            else
            {
              //读写属性错误
                //p0=1;
                       if(slot_curr==0)
            {
               si_status=SI_NOK;
               w_slot=1;
            }
            else
            {
               si_status=SI_NOK;
                 w_slot=0;
            }
            }//2
    }//1
        if(si_status == SI_OK)
        {
          if((c_len>240)||(c_len==0))
            {len_status = 1;}
          if(len_status == 0)
          {
if((c_slot==IM_SLOT)&&(c_index==IM_INDEX)&&(c_len==0x44))//为 IM 槽索引
       {
         if(im_status==RE_IMCALL)
                     {
                  deal_im_data(); //im0_body[i] = IM_read[i]
                dpram.AcycData_DPv1c1[0] = 0x5E;
                dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
                    for(i=0;i<64;i++)
                {
                   dpram.AcycData_DPv1c1[4] = IM_CallHeader[0];// 0x08
                   dpram.AcycData_DPv1c1[5] = IM_CallHeader[1];// 0x00
                   dpram.AcycData_DPv1c1[6] = IM_CallHeader[2];// 0xFD
       dpram.AcycData_DPv1c1[7] = IM_CallHeader[3];// 0xE8
       dpram.AcycData_DPv1c1[i+8]=im0_body[i];  //把 CALL-DU 部分取出
```

```
                        }
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,dpram.AcycData_D
Pv1c1[3]+4,&dpram.AcycData_DPv1c1[0]);
                       im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                        si_status=0;
                             }
                               else if(im_status==RES_IMCALL)
                               {
                                  deal_im_data();
                                  dpram.AcycData_DPv1c1[0] = 0x5E;
                        dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                        dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                        dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
                            for(i=0;i<64;i++)
                        {
                         dpram.AcycData_DPv1c1[4] = IM_CallHeader[0];// 0x08
                         dpram.AcycData_DPv1c1[5] = IM_CallHeader[1];// 0x00
                         dpram.AcycData_DPv1c1[6] = IM_CallHeader[2];// 0xFD
                         dpram.AcycData_DPv1c1[7] = IM_CallHeader[3];// 0xE8
                         dpram.AcycData_DPv1c1[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                        }
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,dpram.AcycData_D
Pv1c1[3]+4,&dpram.AcycData_DPv1c1[0]);
                         im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                         si_status=0;
                               }
                               else if(im_status==NO_IMCALL)
                               {
                     dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
                    dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
                    dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_STATE);
//Error_Code_1 :invalid slot
                    dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycDat
a_DPv1c1[0]);
                        si_status = 0;
                               }
                             }
                           else //为普通槽索引
                               {
    c_len=deal_acyclic_read_data(c_slot,c_index,c_len);
                        dpram.AcycData_DPv1c1[0] = 0x5E;
                        dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                        dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                        dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
    for(i=0;i<c_len;i++){dpram.AcycData_DPv1c1[i+4]=v1_input[i];}
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,dpram.AcycDa
ta_DPv1c1[3]+4,&dpram.AcycData_DPv1c1[0]);
                        }
                       }
                     else //长度错误   B7
                     {
              dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
              dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
              dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
              dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
```

```
cData_DPv1c1[0]);
                        len_status=0;
                        si_status=0;
                    }
                  }
                  else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误 B2
                  {
              dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
              dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
              dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
                    dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycDat
a_DPv1c1[0]);
                    si_status=0;
                  }
                  else if((si_status == SI_NOK)&&(w_index==1))//索引错误 B0
                  {
              dpram.AcycData_DPv1c1[0] |= 0xDE; //Function Code
              dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
              dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
              dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
          si_status=0;
                  }
          }break;
                case 0x5F :   //进来的是:0x5F Write
                {
              c_slot  = dpram.AcycData_DPv1c1[1];//slot
                    c_index = dpram.AcycData_DPv1c1[2];//index
                    c_len   = dpram.AcycData_DPv1c1[3];//length
              for(i=0;i<MAX_SI_NUM;i++)
              {//1
                  if((slot_index[i].rw==SI_W)||(slot_index[i].rw==SI_RW))  //仅在具有读属性的
槽-索引内寻找
                {//2
                  if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                  {//3
                      slot_curr=0x01;//声明列表中出现过正确的槽号
                       if(c_index==slot_index[i].index)
                    {si_status=SI_OK;break;}
                        else
                        {
                             si_status=SI_NOK;
                             w_index=1;
                           }
                  }
                          else
                          {
                               if(slot_curr==0)
                               {
                                 si_status=SI_NOK;
                                 w_slot=1;
                               }
                              else
                                {
```

```
                                si_status=SI_NOK;
                                w_slot=0;
                              }
                          }//3
                }
                      else
                      {
                             if(slot_curr==0)
                               {
                                 si_status=SI_NOK;
                                 w_slot=1;
                               }
                             else
                               {
                                 si_status=SI_NOK;
                                 w_slot=0;
                               }
                      }//2
              }//1
                 if(si_status == SI_OK)
                 {
                   if((c_len>240)||(c_len==0))
                    {len_status = 1;}
                   if(len_status == 0)
                   {

                              if((c_slot==IM_SLOT)&&(c_index==IM_INDEX))
                          {// 5F 00 FF 04 08 00 FD E8
                            if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                                {
                                    if((dpram.AcycData_DPv1c1[3]==4)
&&(dpram.AcycData_DPv1c1[4]==8)&&(dpram.AcycData_DPv1c1[5]==0)
&&(dpram.AcycData_DPv1c1[6]==0xFD)&&(dpram.AcycData_DPv1c1[7]==0xE8))
                           {  //下面 4 条可不要
                               dpram.AcycData_DPv1c1[0] = 0x5F;
                               dpram.AcycData_DPv1c1[1] = c_slot;
                               dpram.AcycData_DPv1c1[2] = c_index;
                               dpram.AcycData_DPv1c1[3] = c_len;
                               IM_CallHeader[0] = dpram.AcycData_DPv1c1[4];// 0x08
                               IM_CallHeader[1] = dpram.AcycData_DPv1c1[5];// 0x00
                               IM_CallHeader[2] = dpram.AcycData_DPv1c1[6];// 0xFD
                              IM_CallHeader[3] = dpram.AcycData_DPv1c1[7];// 0xE8

    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
                          im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                           si_status=0;
                             }
                         else //application  & feature not supported (不是 65000) B6
                             {
                    dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
                    dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
                    dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)|
DPSE_ERRCL_ACC_ACCESS); //Error_Code_1 :invalid slot
                    dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycDat
a_DPv1c1[0]);
                                  im_status=NO_IMCALL;
```

```
                                    si_status=0;
                                }
                                 }//if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                            else //(im_status==RE_IMCALL)
                            {
                                 if((dpram.AcycData_DPv1c1[3]==4)
  &&(dpram.AcycData_DPv1c1[4]==8)&&(dpram.AcycData_DPv1c1[5]==0)
&&(dpram.AcycData_DPv1c1[6]==0xFD)&&(dpram.AcycData_DPv1c1[7]==0xE8))
                                  {  //下面 4 条可不要
                                     dpram.AcycData_DPv1c1[0] = 0x5F;
                                     dpram.AcycData_DPv1c1[1] = c_slot;
                                     dpram.AcycData_DPv1c1[2] = c_index;
                                     dpram.AcycData_DPv1c1[3] = c_len;
                                     IM_CallHeader[0] = dpram.AcycData_DPv1c1[4];// 0x08
                                     IM_CallHeader[1] = dpram.AcycData_DPv1c1[5];// 0x00
                                     IM_CallHeader[2] = dpram.AcycData_DPv1c1[6];// 0xFD
                                      IM_CallHeader[3] = dpram.AcycData_DPv1c1[7];// 0xE8
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
                                  im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                                   si_status=0;
                                     }
                                 else //application  & feature not supported (不是 65000) B6
                                     {
                              dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
                              dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.AcycDat
a_DPv1c1[0]);
                                        im_status=NO_IMCALL;
                                         si_status=0;
                                  }
                                     }
                                 }
                                 else  //普通槽索引
                                 {
                                 dpram.AcycData_DPv1c1[0] = 0x5F;
                                 dpram.AcycData_DPv1c1[1] = c_slot;  //寻址槽号
                                 dpram.AcycData_DPv1c1[2] = c_index; //寻址索引号
                                 dpram.AcycData_DPv1c1[3] = c_len;   //本槽-索引数据长度
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
for(i=0;i<c_len;i++){v1_output[i]=dpram.AcycData_DPv1c1[i+4];}
                                   deal_acyclic_write_data(c_slot,c_index,c_len);
                                 }
                             }
                               else //长度错误  B7
                               {
                            dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
                            dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
                            dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
                            dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
                               si_status=0;
```

```
                }
              }
              else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误  B6
              {
          dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
          dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
        //dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
          dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
                dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
                si_status=0;
              }
              else if((si_status == SI_NOK)&&(w_index==1))//索引错误  B0
              {
          dpram.AcycData_DPv1c1[0] |= 0xDF; //Function Code
          dpram.AcycData_DPv1c1[1]  = 0x80; //Error_Decode
          dpram.AcycData_DPv1c1[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c1[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c1,4,&dpram.Acy
cData_DPv1c1[0]);
                si_status=0;
              }
             }break;
         default : break;
           }
}
```

（16）DPV1/C2 数据处理

```
void Acyclic_data_C2_DSuartMode(void)
{
    u16 i;
    u8  w_slot,w_index,slot_curr;
  u8  len_status;
  w_slot=w_index=slot_curr=0;
    len_status=0;
    i=0;
  switch(dpram.AcycData_DPv1c2[0]) //进来的是:0x5E 或 0x5F
   {
    case 0x5E :
      {
            c_slot  = dpram.AcycData_DPv1c2[1];
                  c_index = dpram.AcycData_DPv1c2[2];
                  c_len   = dpram.AcycData_DPv1c2[3];
            for(i=0;i<MAX_SI_NUM;i++)
            {//1
               if((slot_index[i].rw==SI_R)||(slot_index[i].rw==SI_RW))   //仅在具有读属性
的槽-索引内寻找
              {//2
                if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                {//3
                          slot_curr=0x01;//声明列表中出现过正确的槽号
                    if(c_index==slot_index[i].index)
                  {si_status=SI_OK;break;}
                      else
```

```
            {
                  si_status=SI_NOK;
                              w_index=1;
                }
     }
              else
              {
                   if(slot_curr==0)
                   {
                     si_status=SI_NOK;
                     w_slot=1;
                   }
                  else
                   {
                     si_status=SI_NOK;
                     w_slot=0;
                   }
              }//3
   }
         else
         {
                if(slot_curr==0)
                   {
                     si_status=SI_NOK;
                     w_slot=1;
                   }
                else
                   {
                     si_status=SI_NOK;
                     w_slot=0;
                   }
         }//2
 }//1 for(i=0;i<MAX_SI_NUM;i++)
 if(si_status == SI_OK)
 {
    if((c_len>240)||(c_len==0))
      {len_status = 1;}
    if(len_status == 0)
    {
            if(c_len>240)
    {
     c_len=240;
    }
        if((c_slot==IM_SLOT)&&(c_index==IM_INDEX)&&(c_len==0x44))//为 IM 槽索引
{
          if(im_status==RE_IMCALL)
                {
               deal_im_data(); //im0_body[i] = IM_read[i]
                        dpram.AcycData_DPv1c2[0] = 0x5E;
              dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
              dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
              dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                 for(i=0;i<64;i++)
             {
                dpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
                dpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
                dpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
                dpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
```

```
                dpram.AcycData_DPv1c2[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                    }
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c2,dpram.AcycDa
ta_DPv1c2[3]+4,&dpram.AcycData_DPv1c2[0]);
                    im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                    si_status=0;
                      }
                        else if(im_status==RES_IMCALL)
                        {
                                deal_im_data();
                                dpram.AcycData_DPv1c2[0] = 0x5E;
                     dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                     dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                     dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                        for(i=0;i<64;i++)
                    {
                    dpram.AcycData_DPv1c2[4] = IM_CallHeader[0];// 0x08
                    dpram.AcycData_DPv1c2[5] = IM_CallHeader[1];// 0x00
                    dpram.AcycData_DPv1c2[6] = IM_CallHeader[2];// 0xFD
                     dpram.AcycData_DPv1c2[7] = IM_CallHeader[3];// 0xE8
            dpram.AcycData_DPv1c2[i+8]=im0_body[i];  //把 CALL-DU 部分取出
                    }
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c2,dpram.AcycDa
ta_DPv1c2[3]+4,&dpram.AcycData_DPv1c2[0]);
                   im_status=RES_IMCALL;  //标记本次 IM0 读取已经完成
                   si_status=0;
                        }
        else if(im_status==NO_IMCALL)
            {
                dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_STATE);
//Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
                    si_status = 0;
                        }
                  }
                  else //为普通槽索引
                  {
                  c_len=deal_acyclic_read_data(c_slot,c_index,c_len);
                 dpram.AcycData_DPv1c2[0] = 0x5E;
                 dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                 dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                 dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
 for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=v1_input[i];}
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.AcycData_DPv1c2,dpram.AcycData_D
Pv1c2[3]+4,&dpram.AcycData_DPv1c2[0]);
                }
                  }
                else //长度错误
                {
                dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)|
DPSE_ERRCL_ACC_INV_RANGE); //Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
```

```
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
                  si_status = 0;
                 }
              }
              else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误
              {
                dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
                si_status = 0;
              }
              else if((si_status == SI_NOK)&&(w_index==1))//索引错误
              {
                dpram.AcycData_DPv1c2[0] |= 0xDE; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);

                si_status = 0;
              }
     }break;
    case 0x5F :
        {
             c_slot  = dpram.AcycData_DPv1c2[1];//slot
                   c_index = dpram.AcycData_DPv1c2[2];//index
                   c_len   = dpram.AcycData_DPv1c2[3];//length
             for(i=0;i<MAX_SI_NUM;i++)
             {//1
                 if((slot_index[i].rw==SI_W)||(slot_index[i].rw==SI_RW))    //仅在具有读属性
的槽-索引内寻找
               {//2
                 if(c_slot==slot_index[i].slot)  //槽号是否在列表中
                 {//3
                            slot_curr=0x01;//声明列表中出现过正确的槽号
                       if(c_index==slot_index[i].index)
                    {si_status=SI_OK;break;}
                        else
                        {
                             si_status=SI_NOK;
                     w_index=1;
                            }
                 }
                          else
                          {
                               if(slot_curr==0)
                               {
                                 si_status=SI_NOK;
                                 w_slot=1;
                               }
                             else
```

```
                                {
                                  si_status=SI_NOK;
                                  w_slot=0;
                                }
                            }//3
                    }
                          else
                          {
                                 if(slot_curr==0)
                                   {
                                     si_status=SI_NOK;
                                     w_slot=1;
                                   }
                                 else
                                   {
                                     si_status=SI_NOK;
                                     w_slot=0;
                                   }
                          }//2
                  }//1
                  if(si_status == SI_OK)
                  {
                     if((c_len>240)||(c_len==0))
                       {len_status = 1;}
                     if(len_status == 0)
                     {
                         if((c_slot==IM_SLOT)&&(c_index==IM_INDEX))
                         {// 5F 00 FF 04 08 00 FD E8
                          if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                              {
                                 if((dpram.AcycData_DPv1c2[3]==4)
&&(dpram.AcycData_DPv1c2[4]==8)&&(dpram.AcycData_DPv1c2[5]==0)
&&(dpram.AcycData_DPv1c2[6]==0xFD)&&(dpram.AcycData_DPv1c2[7]==0xE8))
                           {  //下面 4 条可不要
                                dpram.AcycData_DPv1c2[0] = 0x5F;
                                dpram.AcycData_DPv1c2[1] = c_slot;
                                dpram.AcycData_DPv1c2[2] = c_index;
                                dpram.AcycData_DPv1c2[3] = c_len;
                                IM_CallHeader[0] = dpram.AcycData_DPv1c2[4];// 0x08
                                IM_CallHeader[1] = dpram.AcycData_DPv1c2[5];// 0x00
                                IM_CallHeader[2] = dpram.AcycData_DPv1c2[6];// 0xFD
                                 IM_CallHeader[3] = dpram.AcycData_DPv1c2[7];// 0xE8
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);

                              im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                                 si_status=0;
                              }
                             else //application  & feature not supported (不是 65000) B6
                              {
                          dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
                          dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                          dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
                                    im_status=NO_IMCALL;
```

```
                                si_status=0;
                        }
                         }//if((im_status==NO_IMCALL)||(im_status==RES_IMCALL))
                             else //(im_status==RE_IMCALL)
                {
                                if((dpram.AcycData_DPv1c2[3]==4)
&&(dpram.AcycData_DPv1c2[4]==8)&&(dpram.AcycData_DPv1c2[5]==0)
&&(dpram.AcycData_DPv1c2[6]==0xFD)&&(dpram.AcycData_DPv1c2[7]==0xE8))
                             {  //下面 4 条可不要
                                dpram.AcycData_DPv1c2[0] = 0x5F;
                                dpram.AcycData_DPv1c2[1] = c_slot;
                                dpram.AcycData_DPv1c2[2] = c_index;
                                dpram.AcycData_DPv1c2[3] = c_len;

                                IM_CallHeader[0] = dpram.AcycData_DPv1c2[4];// 0x08
                                IM_CallHeader[1] = dpram.AcycData_DPv1c2[5];// 0x00
                                IM_CallHeader[2] = dpram.AcycData_DPv1c2[6];// 0xFD
                                IM_CallHeader[3] = dpram.AcycData_DPv1c2[7];//
0xE8DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.Acy
cData_DPv1c2[0]);
                               im_status=RE_IMCALL;  //标记已经接收到了 IM_CALL
                                 si_status=0;
                              }
                             else //application  & feature not supported (不是 65000) B6
                                {
                        dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
                        dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                        dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
//Error_Code_1 :invalid slot
                        dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
                                    im_status=NO_IMCALL;
                                      si_status=0;
                        }
                             }
                            }
                         else
                         {
                        dpram.AcycData_DPv1c2[0] = 0x5F;
                        dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
                        dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
                        dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
for(i=0;i<c_len;i++){v1_output[i]=dpram.AcycData_DPv1c2[i+4];}
                             deal_acyclic_write_data(c_slot,c_index,c_len);
                         }
                    }
                      else //长度错误  B7
                      {
              dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
              dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
              dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_RANGE);
//Error_Code_1 :invalid slot
              dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
```

```
                si_status = 0;
              }
            }
            else if((si_status == SI_NOK)&&(w_slot==1))//槽号错误  B6
            {
         dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
       //dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_SLOT);
//Error_Code_1 :invalid slot
         dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_ACCESS);
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
    DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.Acy
cData_DPv1c2[0]);
              si_status = 0;
            }
            else if((si_status == SI_NOK)&&(w_index==1))//索引错误 B0
            {
         dpram.AcycData_DPv1c2[0] |= 0xDF; //Function Code
         dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
         dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_ACCESS <<4)| DPSE_ERRCL_ACC_INV_INDEX);
//Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x00; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);

              si_status = 0;
            }
      }break;
        case 0x51 :
          {
               c_slot  = dpram.AcycData_DPv1c2[1];//slot
               c_index = dpram.AcycData_DPv1c2[2];//index
               c_len   = dpram.AcycData_DPv1c2[3];//length
           if((c_slot==SLOT_03)&&(c_index==S03_INDEX_01)) //slot_index=[4,1]
               {
                dpram.AcycData_DPv1c2[0] = 0x51;
              dpram.AcycData_DPv1c2[1] = c_slot;  //寻址槽号
              dpram.AcycData_DPv1c2[2] = c_index; //寻址索引号
              dpram.AcycData_DPv1c2[3] = c_len;   //本槽-索引数据长度
                /*
                for(i=0;i<c_len;i++){v1_output[i]=dpram.AcycData_DPv1c2[i+4];}
              deal_acyclic_write_data(c_slot,c_index,c_len);
              for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=0x51+i;}
               */
for(i=0;i<c_len;i++){dpram.AcycData_DPv1c2[i+4]=dpram.AcycData_DPv1c2[i+4];}
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,dpram.AcycData_D
Pv1c2[3]+4,&dpram.AcycData_DPv1c2[0]);
             }
               else
               {
                dpram.AcycData_DPv1c2[0] |= 0xD1; //Function Code
                dpram.AcycData_DPv1c2[1]  = 0x80; //Error_Decode
                dpram.AcycData_DPv1c2[2]  = ((DPSE_ERRCL_APPLICATION <<4)|
DPSE_ERRCL_APP_NOTSUPP); //Error_Code_1 :invalid slot
                dpram.AcycData_DPv1c2[3]  = 0x51; //Error_Code_2
DSuart_DataExchange_TR(UARTMode_ADDR1,FC_DPVOIn,(u32)&dpram.AcycData_DPv1c2,4,&dpram.AcycDat
a_DPv1c2[0]);
               }
```

```
            }break;

            default : break;
  }
}
```

（17）外部诊断处理

```
void DP_DSuart_DiagHandler(void)
{
        u8 i;
  //=================Ext_Diag 处理=====================================//
        if(((ExtFlag&0x0F)==1)&&((StaFlag&0x0F)==0))//Ext.diag 高优先权外部诊断，从站有错误
        {
                if(flag_diag == 0)
                {
                dpram.R_command1 |= 0x02;
                    dpram.DiagData_DP[0] = 0x81;
                  current_diag_len=dpram.InitData_DPv0[9]-6;
for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF1
                    Tn=1;
                  flag_diag = 1;
                }
          }
else if(((ExtFlag&0x0F)==0)&&((StaFlag&0x0F)==0)&&(Tn==1))//Ext.diag 高优先权外部诊断，从站无
错误
        {
                if(flag_diag == 1)
                {
                dpram.R_command1 |= 0x02;
                dpram.DiagData_DP[0] = 0x01;
                current_diag_len=dpram.InitData_DPv0[9]-6;
    for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=ExtDiagData_input[i];}//0xF0
                    Tn=0;
                    flag_diag = 0;
              }
            }
else if(((StaFlag&0x0F)==1)&&((ExtFlag&0x0F)==0))    //Stat.diag  高优先权静态诊断，从站有错误
        {
                if(flag_diag == 0)
                {
                    dpram.R_command1 |= 0x02;
                    dpram.DiagData_DP[0] = 0x82;
                  current_diag_len=dpram.InitData_DPv0[9]-6;
for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF2
                    Tn1=1;
                  flag_diag = 1;
              }
            }
else if(((StaFlag&0x0F)==0)&&((ExtFlag&0x0F)==0)&&(Tn1==1))//Stat.diag 高优先权静态诊断，从站
无错误
        {
                if(flag_diag == 1)
                {
                    dpram.R_command1 |= 0x02;
                    dpram.DiagData_DP[0] = 0x02;
                    current_diag_len=dpram.InitData_DPv0[9]-6;
    for(i=0;i<dpram.InitData_DPv0[9]-6;i++){diag_input[i]=StaDiagData_input[i];}//0xF0
```

```
                    Tn1=0;
                    flag_diag = 0;
                }
            }
    if((dpram.R_command1&0x02) == 0x02)
    {
            dpram.R_command1 &= 0xFD;
        dpram.DiagData_DP[1] = current_diag_len;
        for(i=0;i<dpram.DiagData_DP[1];i++)
           {dpram.DiagData_DP[i+2] = diag_input[i];}
        DSuart_Diag_MCUUartSend();
    }
  //=====================================================================//
}
```

（18）数据交换 2

```
void DPV0V1_DataEx_DSuartMode(void) //IO 数据缓冲区与双口 RAM 之间的数据交换
{
     u8 i;
     u8 indataLen,outdataLen;
     u8 uartRx_CRC_H=0,uartRx_CRC_L=0;
     u16 tCRC;
    //=========================DPV0_OUT 处理==============================//
      if(dpram.InData_DP[0]>0)
        {
        for(i=0;i<dpram.InData_DP[0];i++)
        {
          dpram.InData_DP[i+1] = v0_input[i];
        //dpram.InData_DP[i+1] = dpram.OutData_DP[i+1];
        indataLen=dpram.InData_DP[0];
        }
      }
        else
        {
            indataLen=0;
        }

        if(dpram.OutData_DP[0]>0)
        {
      outdataLen=dpram.OutData_DP[0];
      }
        else
        {
            outdataLen=0;
        }
    //=========================DPV0_IN 处理==============================//
    //=========================诊断处理==================================//
    //=========================Acyclic_data_C1 处理======================//
    //=========================Acyclic_data_C2 处理======================//
        if(COM2_auto_t==1)
            {
         COM2_auto_t=0;
DSuart_DataExchange_master(UARTMode_ADDR1,FC_DPV0In,(u32)&dpram.InData_DP,indataLen+1,dpram.
InData_DP);
                //01 00 01 30 31 30 11 22 33.....
        }
            if(DSuart_re.ok==1)
```

```
            {
                DSuart_re.ok=0;
                tCRC = CRC_avr(&DSuart_re.box[0],DSuart_re.box[4]+5);
                uartRx_CRC_H = tCRC >>8;
                uartRx_CRC_L = tCRC ;
            if((uartRx_CRC_H==
DSuart_re.box[DSuart_re.box[4]+5])&&(uartRx_CRC_L==DSuart_re.box[DSuart_re.box[4]+5+1]))
                 {
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&(DSuart_re.box[1]==FC_DPDataEx)&&(DSuart_r
e.box[2]==0x03)&&(DSuart_re.box[3]==0x19))
                        {//-------------接收 DPV0_OUT 数据
CopyUART_RxBufToDPRAM_DPV0Out(&DSuart_re.box[6],outdataLen);//复制 DPv0 输出数据到本地缓存

                            for(i=0;i<outdataLen;i++)
                    {
                                v0_output[i] = dpram.OutData_DP[i+1];
                            }
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x04)==FC_DPV1C1)&&(DSu
art_re.box[2]==0x05)&&(DSuart_re.box[3]==0xD0)&&(DSuart_re.box[5]==0x5E))
                        {//-------------Acyclic_data_C1-Read 处理
                          CopyUART_RxBufToDPRAM_DPV1C1(&DSuart_re.box[5],4);
                            Acyclic_data_C1_DSuartMode();
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x04)==FC_DPV1C1)&&(DSu
art_re.box[2]==0x05)&&(DSuart_re.box[3]==0xD0)&&(DSuart_re.box[5]==0x5F))
                        {//-------------Acyclic_data_C1-Write 处理
CopyUART_RxBufToDPRAM_DPV1C1(&DSuart_re.box[5],DSuart_re.box[8]+4);
                            Acyclic_data_C1_DSuartMode();
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x08)==FC_DPV1C2)&&(DSu
art_re.box[2]==0x06)&&(DSuart_re.box[3]==0xD0)&&(DSuart_re.box[5]==0x5E))
                        {//-------------Acyclic_data_C2-Read 处理
                          CopyUART_RxBufToDPRAM_DPV1C2(&DSuart_re.box[5],4);
                            Acyclic_data_C2_DSuartMode();
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x08)==FC_DPV1C2)&&(DSu
art_re.box[2]==0x06)&&(DSuart_re.box[3]==0xD0)&&(DSuart_re.box[5]==0x5F))
                        {//-------------Acyclic_data_C2-Write 处理
CopyUART_RxBufToDPRAM_DPV1C2(&DSuart_re.box[5],DSuart_re.box[8]+4);
                            Acyclic_data_C2_DSuartMode();
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x08)==FC_DPV1C2)&&(DSu
art_re.box[2]==0x06)&&(DSuart_re.box[3]==0xD0)&&(DSuart_re.box[5]==0x51))
                        {//-------------Acyclic_data_C2-DataTransport 处理
CopyUART_RxBufToDPRAM_DPV1C2(&DSuart_re.box[5],DSuart_re.box[8]+4);
                            Acyclic_data_C2_DSuartMode();
                        }
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x20)==FC_DPCFG)&&(DSua
rt_re.box[2]==0x05)&&(DSuart_re.box[3]==0x00))
                        {//-------------接收 CFG 数据
                           dpram.CfgData_DP[0]=DSuart_re.box[5];//OutLen
                            dpram.CfgData_DP[1]=DSuart_re.box[6];//InLen
                     dpram.OutData_DP[0]=dpram.CfgData_DP[0];
                            dpram.InData_DP[0] =dpram.CfgData_DP[1];
                            //01 20 05 00 07 30 30 06 1f 2f 1f 2f 1f 2f 3B F0
CopyUART_RxBufToDPRAM_CFGData(&DSuart_re.box[8],DSuart_re.box[7]);
                        }
```

```
    if(((DSuart_re.box[0]&0x01)==UARTMode_ADDR1)&&((DSuart_re.box[1]&0x10)==FC_DPPRM)&&(DSuart_re.box[2]==0x04)&&(DSuart_re.box[3]==0x10))
                        {//------------接收 PRM 数据
     CopyUART_RxBufToDPRAM_PRMData(&DSuart_re.box[5],DSuart_re.box[5]);
                        }
                    }
                    else
                    {}//CRC error
        }
            else
          {}// DSuart_re.ok !=1
        //==================诊断处理=====================================//
        DP_DSuart_DiagHandler();
      //=============================================================//
}
```

（19）串口 2 中断处理

```
//--------------------串口 2 中断处理-----------------------------//
void USART2_IRQHandler(void)
{
 uint8_t x,temp1;
 //temp1 = COM2_rs232_prm[1];
 if(USART_GetITStatus(USART2,USART_IT_RXNE)==SET)
 {/* 接收中断*/
    TIM_Cmd(TIM3, DISABLE);
    /*
  if(COM2_SR_PE ==1)      // 校验
  {COM2_error_code=1;}
  else
  {COM2_error_code=0;}
    */
  x=USART_ReceiveData(USART2);
    /*
    switch (temp1)
 {
      case 0:
      case 1:
        case 2:
               break;
        case 3:
        case 4:
             {x=x&0x7F;}
             break;
        case 5:
        case 6:
             break;
        case 7:
        case 8:
        case 9:
               {x=x&0x7F;}
             break;
      default:
             break;
}
  DSuart_re.box[DSuart_re.pend]=x;
  DSuart_re.pend++;
  SET_COM2_RE;  //接收允许
```

```
  Clear_COM2_RXNEIE;//清除接收中断

        if(COM2_relen==0)
        {
            //====按报文间隔接收====//
                COM2_M_cc=COM2_M_c;
        TIM3_Init(COM2_Gap_Period,COM2_Gap_Prescaler);
            TIM_Cmd(TIM3, ENABLE);
        }
      else
        {
            //====按报文长度接收====
            if(DSuart_re.pend<COM2_re_len)
            {
          //
            }
            else
            {
             DSuart_re.ok=1;
           TIM_Cmd(TIM3, DISABLE);
            }
        }
 }/* 接收中断完*/
 if(USART_GetITStatus(USART2,USART_IT_TC)==SET)
 {/*发送中断*/
     if (DSuart_tr.p<DSuart_tr.pend)
     {//未发送完
      x=DSuart_tr.box[DSuart_tr.p];
            Clear_COM2_TCIE;
      USART_SendData(USART2,x); //启动发送
      DSuart_tr.p++;
     }
     else
     {//已发送完
             RSET_COM2_RTS;
             Clear_COM2_TCIE;
             SET_COM2_RE;    //接收允许
       RSET_COM2_TE;
       DSuart_re.p=0;  //接收信箱清 0
       DSuart_re.pend=0;
       COM2_error_code=0;
       if(COM2_automt==1)
       {
                COM2_M_cc1=COM2_M_c1;
        TIM2_Init(COM2_M_h,COM2_M_l);
            TIM_Cmd(TIM2, ENABLE);
       }
     }
 }// 发送中断完
}
```

（20）定时器中断处理

```
//--------------------定时器 TIM2 中断处理-----------------------------//

void TIM2_IRQHandler(void)//发送控制定时器
{
```

```
    if(TIM_GetITStatus(TIM2,TIM_IT_Update) == SET)
  {
     TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
         COM2_M_cc1=COM2_M_cc1-1;
            if(COM2_M_cc1 == 0)
            {
                COM2_auto_t=1;
                    TIM_Cmd(TIM2, DISABLE);// 关闭定时器
            }
            else
            {
          TIM2_Init(COM2_M_h,COM2_M_l);
                TIM_Cmd(TIM2, ENABLE);
            }
  }     //end if timer_mode
}
void TIM3_IRQHandler(void)//接收控制定时器
{
    if(TIM_GetITStatus(TIM3,TIM_IT_Update) == SET)
  {
     TIM_ClearITPendingBit(TIM3,TIM_IT_Update);
          COM2_M_cc=COM2_M_cc-1;//COM1_M_c=COM1_M_c
            if(COM2_M_cc == 0)
            {
                DSuart_re.ok=1;
          RSET_COM2_RE;
                    TIM_Cmd(TIM3, DISABLE);// 关闭定时器
            }
            else
            {
                    TIM3_Init(COM2_Gap_Period,COM2_Gap_Prescaler);
                TIM_Cmd(TIM3, ENABLE);
            }
   }
}
```

第八章 DSDPV1-RSU 芯片软件设计

一、DSDPV1-RSU 芯片软件设计

DSDPV1-RSU 芯片软件设计与 M 系列板卡软件设计方法相同，详见本手册“第六章/第七章/第八章”

第九章 DSDPV1-R 芯片软件设计

一、 DSDPV1-R 芯片软件设计

DSDPV1-R 芯片使用方法与 M 卡软使用方法类似，不再一一复述，可参考“第六章双口 RAM 接口”，或“DSDPV1-R/fsmc_sram.c/fsmc_sram.h”,需要用户注意的是，DSDPV1-R 芯片与用户 MCU 只有一种通讯接口：双口 RAM 接口。

第十章 用户参数概述

一、什么情况下需要使用“用户参数”

对于工业现场设备，常需要用户根据现场应用设定一些参数；其中有些参数不需要在设备运行中实时改变，如变频器的电流上限保护与报警值；如温度传感器的测量温度范围、热电偶选型、输出 4-20mA/1-5V 选择等。如果这些参数作为PROFIBUS 主站的 I/O 输出，将占用 PROFIBUS 主站 I/O 资源和周期性轮循 PROFIBUS 从站的时间资源。

将这些参数处理成“用户参数”，将会缩短 PROFIBUS 主站通信时间、减小通信报文长度、提高总线通信效率。使用“用户参数”技术，只需要在主站配置中做出参数选择，主站在与从站连接时，一次性将这些参数传送到从站，从站就可以使用这些用户选择的参数对从站进行参数化（初始化、参数设定）。

二、确定“用户参数”类型、个数、字节长度

以一个 4 通道模拟量输入模块为例，假设需要用户设置的参数有：

▼ 输入类型：1~5V、0~10V、0~5V、 -10~+10V、4~20mA、0~20mA、0~10mA 共七种选择；占 1 个字节，以

INPUT=0-6 分别表示这 7 种输入类型；

▼ 数据类型：BCD（0000~9999）、无符号整型（0~65535）、有符号整型（-32767~+32767）共三种选择；占 1 个字节，以 VAR=0-2 分别表示这 3 种数据类型；

▼ 输入模式：单端输入（4 通道）、双端输入（2 通道）共二种选择；占 1 个字节，以 CHNO=0-1 分别表示这 2 种输入模式；

因此：用户参数长度 = 3

三、用户 MCU 怎样得到“用户参数”

1．在初始化报文中正确设定用户参数长度

M 卡用户参数的长度最大值为 237，无用户参数时置 0，用户参数长度的值应与 GSD 文件中 User_Prm_Data_Len 的值相等。

2．在 GSD 文件中正确设定相关参数

如上例中用户参数应为：

User_Prm_Data_Len = 3 ；用户参数长度为3字节

User_Prm_Data = 0x03,0x02,0x01 ；用户参数的分别赋值为3、2、1

3．M 卡获取“用户参数”

在主站与从站完成连接时，M 卡将接收到从主站传送来的用户参数。M 卡在与用户模版进行数据交换前把用户参数存放在双口 RAM 的用户参数数据区地址范围：0x410～0x4FD（L= User_Prm_Data_Len）其校验和存放在 0x040E～0x040F，如选择用户判断用户参数正确与否，用户则需要读取双口 RAM 用户参数数据区的数据，并将判断结果告知 M 卡，如用户选择 M 卡判断用户参数正确与否，用户只需要读取双口 RAM 用户参数数据区的数据即可，具体过程详见本手册中的“C 源代码说明”。

第十一章 关于 GSD 文件

一、GSD 文件（Electronic Data Sheet）

1．GSD 文件定义

每一个 PROFIBUS 从站或一类主站都要有一个“设备描述文件”称为 GSD 文件，用来描述该 PROFIBUS-DP 设备的特性。

2．GSD 文件包含了设备所有定义参数，包括：

•支持的波特率；

•支持的信息长度；

•输入/输出数据数量

•诊断数据的含义

•可选模块种类等。

3. **GSD 文件是文本类文件，可用“记事本”编辑。**
4. **无论使用什么样的系统配置软件，都要根据 GSD 文件来对设备配置。**
5. GSD 文件编辑软件

国际 PROFIBUS 组织 PI 提供了 GSD 文件编辑软件：gsdedit.exe。该软件依照 profbus 技术标准格式规定，对用户编辑的 GSD 文件进行格式检查。该软件的“帮助”功能强大,也是一种快速学习 GSD 文件技术的途径。

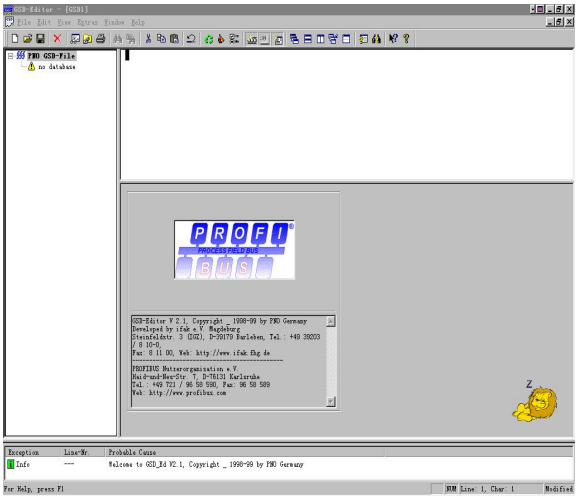


图 5-1 gsdedit 打开一个空文件

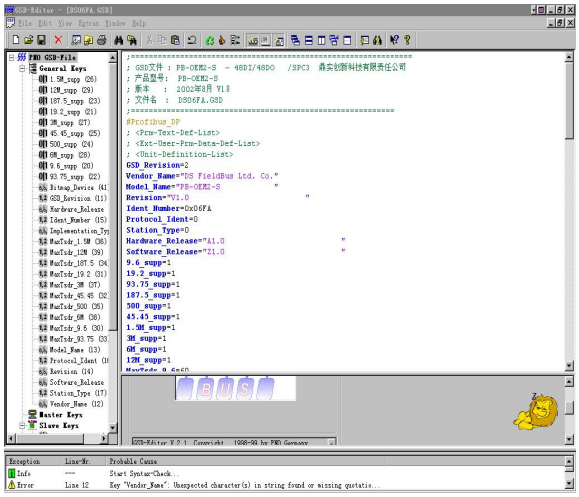


图 5-2 gsdedit 打开 DS_06FA.GSD 文件

二、DSDP0CC9.GSD 说明及如何修改成用户的 GSD 文件

```
GSD Files:
; Supported:Freeze_Mode_supp, Sync_Mode_supp, Auto_Baud_supp, 12Mbit/s
; Date  : 4.14.2014
; File  :
; Revision="V2.0"
//==== General DP Keywords ================//#Profibus_DP
GSD_Revision=5
Vendor_Name="D&S Fieldbus Technology Co.,Ltd"     ; 可改写为用户公司名
Model_Name="PB-DSDPV1"                           ; 可改写为用户产品名即组态时的产品名称
Revision="2.00"                                  ; 可改写为用户产品版本号
Ident_Number=0x0CC9                              ; 可改写为用户产品 ID 号，须用户申请自己的 ID 号,也可用本产品 ID 号
Protocol_Ident=0
```

```
Station_Type=0
Hardware_Release="V1.0"
Software_Release="V2.0"
//==== Supported baudrates ====================//
9.6_supp=1
19.2_supp=1
45.45_supp=1
93.75_supp=1
187.5_supp=1
500_supp=1
1.5M_supp=1
3M_supp=1
6M_supp=1
12M_supp=1
MaxTsdr_9.6=60
MaxTsdr_19.2=60
MaxTsdr_45.45=250
MaxTsdr_93.75=60
MaxTsdr_187.5=60
MaxTsdr_500=100
MaxTsdr_1.5M=150
MaxTsdr_3M=250
MaxTsdr_6M=450
MaxTsdr_12M=800
Redundancy=0
Implementation_Type="DSDPV1"
Bitmap_Device="DSDPV1"          ; 这是图标文件名
;"DPLINK_N"
Bitmap_SF="DPLINK_S"
;==== Slave specific values ======================================//
;PB-OEM1-DSDPV1
OrderNumber="PB-OEM1-DSDPV1"    ; 可改写为用户产品订货号
Slave_Family=9
Info_Text="GSD Version: 1.00,PROFIBUS DPV1 devices, Identification and Maintenance (I&M)"
Periphery="DPS"
Freeze_Mode_supp=1
Sync_Mode_supp=1
Fail_Safe=1
Auto_Baud_supp=1
Set_Slave_Add_supp=0
Min_Slave_Intervall=6
Modular_Station=0
Max_Module=1
```

```
Modul_Offset=0
Max_Input_Len=244
Max_Output_Len=244
Max_Data_Len=488
Max_Diag_Data_Len=9
User_Prm_Data_Len=18
;//==== User-Prm-Data ===================================//
Max_User_Prm_Data_Len=18          ; 用户参数长度，如果没有用户数据可注销此句
Ext_User_Prm_Data_Const(0)                                                    =
0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                                                  ; 用户参数初值，如果没有用户参数可注销此句
;Ext_User_Prm_Data_Ref(3)=3
;Ext_User_Prm_Data_Ref(4)=4
;Ext_User_Prm_Data_Ref(5)=6
;Ext_User_Prm_Data_Ref(6)=8
;Ext_User_Prm_Data_Ref(6)=9
;Ext_User_Prm_Data_Ref(7)=7
;//==== DPV1 ============================================//
DPV1_Slave = 1
C1_Read_Write_supp = 1
C1_Max_Data_Len = 100
C1_Response_Timeout = 10
C2_Read_Write_Supp = 1
C2_Max_Data_Len = 100
C2_Response_Timeout = 100
C2_Max_Count_Channels = 1
Max_Initiate_PDU_Length = 52
Prm_Block_Structure_supp = 0
Prm_Block_Structure_req = 0
Diagnostic_Alarm_supp = 0
Process_Alarm_supp = 0
Manufacturer_Specific_Alarm_supp = 0
Extra_Alarm_SAP_supp = 0
Alarm_Type_Mode_supp = 0
WD_Base_1ms_supp = 1
Publisher_supp = 0
Ident_Maintenance_supp = 1
;//==== Module-Definition-List ============================//
Module=" 48 Byte In, 48 Byte Out   "  0x1f,0x2f,0x1f,0x2f,0x1f,0x2f ; 初始化报文中的 I/O 配置
数据和长度必须和这里一致
 20
 Info_Text="Profibus Inputs and Outputs"
Preset=1
```

EndModule

第十二章 Profibus 接口简介

一、名词注解

Abort	PROFIBUS-DPV1 中止非循环数据交换
CRC16	16 位循环冗余校验
DSDPV1	PROFIBUS-DP 从站协议芯片
DPRAM	Dual-Port-RAM
DP_SAPs	PROFIBUS-DP 服务存取点
DPV0	PROFIBUS-DP 循环数据交换协议
DPV1	PROFIBUS-DP 非循环数据交换协议
DPV1/C1	MSAC_C1
DPV1/C2	MSAC_C2
DS_write	PROFIBUS-DPV1 写非循环数据
DS_read	PROFIBUS-DPV1 读非循环数据
DS_Transport	PROFIBUS-DPV1 非循环数据交换
Freeze	冻结
F_timer	主站报文超时定时器
GSD	PROFIBUS 设备数据文件
Index	索引号
I_timer	主站非循环数据交换请求超时定时器
LSB	数据位最低位
MSB	数据位最高位

OSI	Open System Interconnect(开放式系统互联)
RM	PROFIBUS-DPV1/C2 通道资源管理器
Slot	槽号
Sync	同步
UART	通用异步收发传输
Unsync	解除同步
Unfreeze	解除冻结
U_timer	用户响应超时定时器

二、Profibus 接口

1．DP 服务存取点及其缓存结构

表 17 DPV0 服务存取点 SAPS

SAPS	意义
Default SAP	Data exchange (Write_Read_Data)
SAP55	Changing the station address (Set_Slave_Address)
SAP56	Reading the inputs (Read_Inputs)
SAP57	Reading the outputs (Read_Outputs)
SAP58	Control commands to the DP-Slave (Global_control)
SAP59	Reading configuration data (Get_Config)
SAP60	Reading diagnostics information (Slave_Diagnosis)
SAP61	Sending parameter setting data (Set_Param)
SAP62	Checking configuration data (Check_Config)

DSDPV1 符合 DP 从站协议，集成了表 17 中完整的 DPV0 协议服务存取点(SAPS)，在上述服务存取点使用前，用户首先需要对 DSDPV1 初始化，除了 SAP55(变更从站地址)和 SAP58(全局控制)功能需要用户设定是否激活外，所有的服务存取点 DSDPV1 都支持。

DPV0 服务存取点涉及到的数据包括用户参数数据、配置数据、诊断数据、输入输出交换数据，DSDPV1 为这些数据设置了 SAPS 缓存单元，这里所说的缓存单元区别于前文所述的 DPRAM，SAPS 缓存单元是用于 DP 接口的，而 DPRAM 是用于用户接口的。设置缓存单元的原因：由于各个 SAP 涉及的数据不相同，由于各 SAP 可能交叉运行，而且用户有可能占用 DPRAM 访问权限，使得 SAP 数据共用一个缓存单元可能导致数据被覆盖；另外一个原因就是当主站发起 SAP56(Read_Inputs)、SAP57(Read_Outputs)、SAP59(Get_Config)服务请求时，假如 DSDPV1 不设置 SAPS 缓存，而是采用直接访问 DPRAM 获取响应数据的方式，如果此时用户正在访问 DPRAM，则 DSDPV1 对于主站的响应就需要等待用户释放 DPRAM 访问权限后才能获取数据，那么从站

的响应时间受用户的影响，显然这样做事不合理的。

用户不需要对 SAPS 缓存单元进行初始设定存储空间大小，各个 SAPS 缓存大小在设计时已经设定好。图 8 为各个服务存取点的缓存设置，输出数据缓存设置两个，每个缓存大小为 244bytes，数据交换(DefaultSAP)输出数据(Write_Data)缓存至输出数据缓存 1，当用户支持同步且主站请求同步时(SAP58 Global_control)，输出数据缓存 1 中的输出数据存储至缓存 2，当主站再发起读输出请求 SAP57(Read_Outputs)时，DSDPV1 将输出数据缓存 2 中的输出数据发送至 DP 总线。当主站发起参数化请求 SAP61(Set_Param)时，DSDPV1 将用户参数化数据(不包括参数化数据前 7 个字节)缓存至参数化数据缓存，最大支持 237bytes。当主站发起组态配置请求 SAP62(Check_Config)时，配置数据缓存值配置数据缓存中，最大支持 200bytes。数据交换(DefaultSAP)输入数据(Read_Data)缓存至输入数据缓存 1，当用户主持冻结且主站请求冻结(SAP58 Global_control)时，输入缓存 1 中的输入数据存储至缓存 2，当主站在发起读输入请求 SAP56(Read_Inputs)，DSDPV1 将输入数据缓存 2 中的输入数据发送至 DP 总线。当主站发起诊断请求 SAP60(Slave_Diagnosis)时，DSDPV1 将当前诊断数据缓存中的诊断数据发送至 DP 总线，最大支持 244bytes(其中用户扩展诊断数据最大支持 238bytes)。

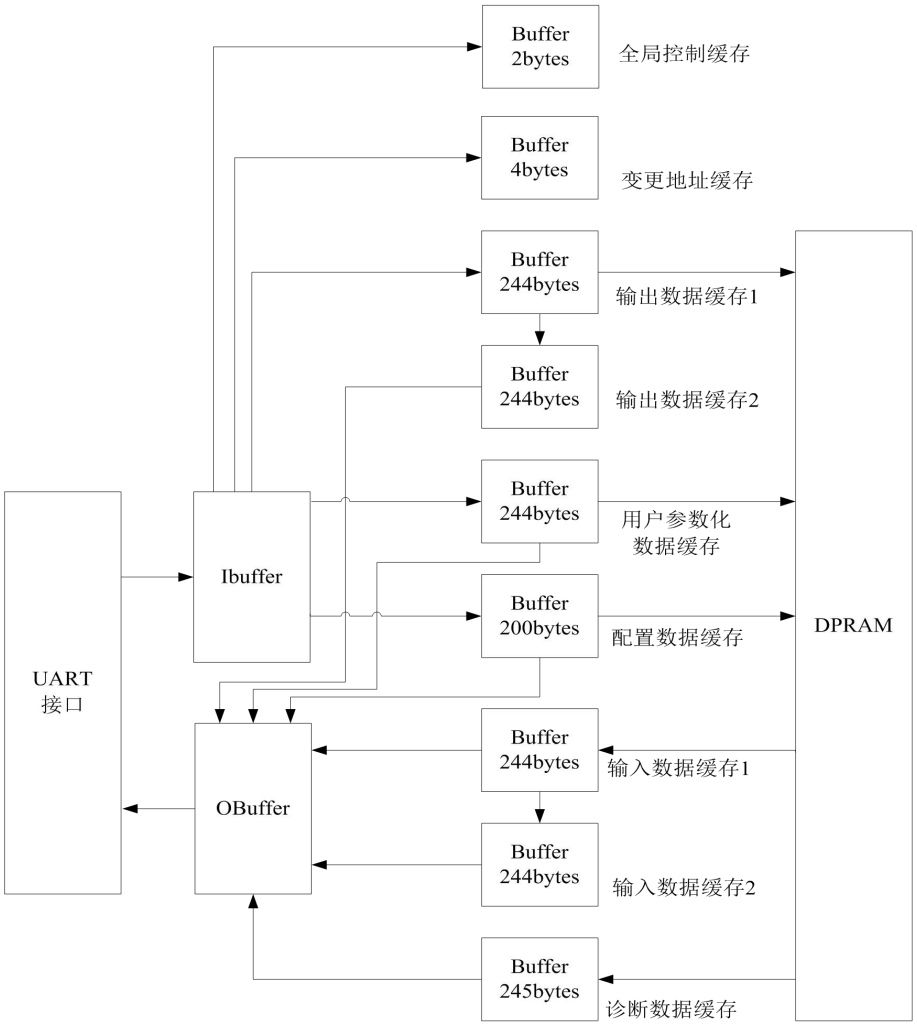


图 8 DPV0_SAPS 缓存结构示意图

需要注意的是 SAPS 缓存和 DPRAM 之间有数据交换，当用户不占用 DPRAM 访问权限时，SAPS 缓存中更新的输出数据、用户参数化数据和配置数据将被写入 DPRAM，DPRAM 中更新的输入数据、诊断数据将被写入 SAPS 缓存中。

2. SAP 描述

（1） Set_slave_Address(SAP55)

用户可以在初始化 DSDPV1 时设定从站地址，之后从站按照设定地址进行 DP 通信，DSDPV1 也支持主站通过 SAP55(Set_slave_Address)来更改从站地址。用户在初始化 DSDPV1 时通过初始化预置命令字节(B024.0)来定义是否支持 SAP55，值得注意的是如果需要通过 SAP55 更改从站地址，从站应处于等待参数化状态。从

站 DSDPV1 不支持 Set_Slave_Address(主站设从站地址)报文中的 Rem_Slave_Data(0~240bytes)。

表 18 Set_Slave_Address 缓存结构

字节	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	名称
0									New_Slave_Address
1									Ident_Number_High
2									Ident_Number_Low
3									No_Add_Chg

（2） Set_Param(SAP61)

DSDPV1 自动判别参数化数据前七个字节是否合法,用户参数数据(0~237bytes)可以由 DSDPV1 判断是否合法,也可以由用户判断是否合法,这需要用户在初始化 DSDPV1 时通过用户参数化数据判断方式字节(B02B)选择用户参数判断方式。

表 19 Set_Param 缓存结构

字节	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	名称
0	Lock.req	Unlock.req	Sync.req	Free.req	WD_ on	Res	Res	Res	Station_status
1									WD_Fact_1
2									WD_Fact_2
3									Min_Tsdr
4									ID_Number_High
5									ID_Number_Low
6									Group_ID
7	DPV1_Enable	Fail_safe	0	0	0	WD_Base	0	0	Spec_User_Prm_Byte
8~243									User_Prm_Data

注：WD_Fact_1 和 WD_Fact_1 的取值范围为 1~255

表 20 Lock_req 和 UnLock_req 的意义

Bit7 Lock_req	Bit6 UnLock_req	意义
0	0	只有 Min_TSDR 可以更改，其余的参数化数据不变
0	1	DP 从站释放当前主站

1	0	DP 从站锁定当前主站，接受所有的参数化数据
1	1	DP 从站释放当前主站

表 21 Spec_User_Prm_Byte 的意义

Bit	Spec_User_Prm_Byte		
	命名	意义	默认值
0~1	Res		0
2	WD_Base	监控主站的时间基准 WD_Base=0：时间基准为 10ms WD_Base=1：时间基准为 1ms	WD_Base=0：时间基准为 10ms
3~5	Res		0
6	Fail_safe	是否开启故障安全模式 Fail_safe=0：关闭 Fail_safe=1：开启	Fail_safe=0：关闭
7	DPV1_Enable	是否开启 DPV1 功能 DPV1_Enable=0 关闭 DPV1 功能 DPV1_Enable=1 开启 DPV1 功能	DPV1_Enable=0 关闭 DPV1 功能

（3） Check_Config(SAP62)

配置数据的判断方式可以由 DSDPV1 判断是否合法，也可以由用户来评判其合法性，用户需要在初始化时通过配置数据判断方式字节(B02D)选择配置数据判断方式。如果选择 DSDPV1 判断配置数据是否合法，如果合法则计算出当前配置数据所指的输入输出数据长度，并且将此长度和配置数据写入 DPRAM 中，一并告知用户。如果选择用户判断，DSDPV1 不判断配置数据是否合法，只计算当前配置数据对应的交换输入输出数据长度，通过中断的方式将配置数据交给用户判断是否合法，同时将计算出的输入输出数据长度一并告知用户(B500 和 B501)，此时 DSDPV1 等待用户判断结果。

用户 CPU 收到来自 DSDPV1 的中断后，立刻申请 DPRAM 访问权，查看芯片命令字节 1(B00E)，查看 bit2 是否有效，若有效将 DPRAM 中的配置数据读取，判断是否合法，之后需要将结果返回给 DSDPV1，收到结果后变更芯片 DP 状态。

（4） Slave_Diagnosis(SAP60)

从站诊断包括标准 6bytes 之外还包括用户诊断数据，用户诊断数据又可分为扩展诊断、外部诊断和静态诊断，除扩展诊断为低优先级诊断，其余两种为高优先级诊断。当用户置扩展诊断后，DSDPV1 将诊断数

据缓存于 Ext_Diag_Data 单元，并置 Diag_Type 为 0x00，当主站读诊断时，DSDPV1 将标准 6 字节诊断数据和扩展诊断数据发送至主站。当用户置外部诊断后，DSDPV1 也将该数据缓存与 Ext_Diag_Data 单元，置 Diag_Type 为 0x81，同时置功能码为高优先级，当主站读诊断时，DSDPV1 将标准 6 字节诊断数据和外部诊断数据发送至主站，当高优先级事件被处理后，用户可取消外部诊断，置 Diag_Type 为 0x01。当用户置静态诊断后，DSDPV1 将诊断数据缓存与 Ext_Diag_Data 单元，置 Diag_Type 为 0x82，同时置功能码为高优先级，当主站读诊断时，DSDPV1 将标准 6 字节诊断数据和静态诊断数据发送至主站，当高优先级事件被处理后，用户必须取消静态诊断，置 Diag_Type 为 0x02。

表 22 Slave_Diagnosis 缓存结构

字节	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	名称
0	0x00：低优先权扩展诊断 0x01：取消高优先权外部诊断 0x81：置高优先权外部诊断 0x02：取消高优先权静态诊断 0x82：置高优先权静态诊断								Diag_Type
1									Station_status_1
2									Station_status_2
3									Station_status_3
4									Master_Add
5									Ident_Number_High
6									Ident_Number_Low
7~244									Ext_Diag_Data（用户可选）

（5）Data_Exchange (Default_SAP)

输入/输出数据交换分别包括 2 个缓存，大小为 244bytes。在非同步模式下，主站发送输出数据，从站将输出数据缓存至输出数据缓存 1，同时将数据缓存至输出数据缓存 2，DSDPV1 将数据缓存 1 中的数据发送至用户，在 Sync 模式下，DSDPV1 将输出数据缓存至缓存 1，但不同时缓存至输入缓存 2，直到 Unsync 模式。在 Unfreeze 模式下，从站将输入数据缓存至输入缓存 1，同时也将数据缓存至输入缓存 2，数据交换时 DSDPV1 只将缓存 1 中的数据发送至主站，在 Freeze 模式下，DSDPV1 将输入数据缓存至输入缓存 1，但不同时缓存至输入缓存 2，数据交换时 DSDPV1 只将缓存 2 中的数据发送至主站。

（6）Global_Control (SAP58)

全局控制命令缓存结构如下表。

表 23　Set_Slave_Address 缓存结构

字节	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	名称
0	0	0	Sync	Unsync	Freeze	Unfreeze	Clear_data	0	Control_Command
1	1								Group_Select

（7） Read_Inputs (SAP56)

当主站请求读取从站输入数据时，从站将输入缓存 2 中的数据发送至主站，无论从站当前是否处于 Freeze 模式。

（8）Read_Outputs (SAP57)

当主站请求读取从站当前输出数据时，从站将输出缓存 2 中的数据发送至主站，无论从站当前是否处于 Sync 模式。

（9）Get_Config (SAP59)

当主站请求读取从站当前配置数据时，从站将配置数据缓存中的数据发送至主站。这里分为两种情况，一是当从站处于数据交换状态时，从站将当前配置数据发送至主站，二是当从站处于非数据交换状态时，从站将用户初始化时的配置数据发送至主站。

第十三章 Profibus-DPV1

芯片支持 Profibus-DPV1 非循环数据交换功能，从站 DPV1 非循环数据交换分为 MSAC1S（简称从站 C1）部分和 MSAC2S（简称从站 C2）部分用户在初始化时对 DPRAM 地址单元 B005 设置是否支持 DPV1 功能，DPV1 非循环数据最大输入输出数据长度 240bytes，DPV1/C1 服务存取点分别为 SAP51，DPV1/C2 支持数据 2 个通道，服务存取点分别为 SAP47、SAP48。

表 24　PROFIBUS-DPV1 SAP 服务访问点

服务	主站 SAP	从站 SAP	报文
MSAC1	0x33	0x33	DS_read（读非循环数据）、DS_write

			（读非循环数据）、Alarm_ack（报警确认）
MSAC1	0x33	0x32	Alarm_ack（报警确认）暂不支持
MSAC2	0x32	0x31	Initiate.req(初始化 C2 资源管理器)
MSAC2	0x32	0x00~0x30	DS_read（读非循环数据）、DS_write（读非循环数据）、DS_transport（读写非循环数据操作）以及 Abort（中止 C2 连接）、Idle（空闲报文）

一、 DPV1 用户扩展参数定义

DPV1 功能需要参数化数据补充 3bytes 来实现，也就是说开启 DPV1 功能至少需要参数化数据长度大于等于 10bytes，服务存取点仍为 SAP61，表 25 为扩展参数缓存结构。DSDPV1 暂不支持报警(MSAL1_Alarm)，因此 DPV1_Status_2、DPV1_Status_3 保留。

表 25 set_Param 缓存结构

字节	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	名称
0~6									
7	DPV1_ Enable	Fail_safe	reserved	reserved	reserved	WD_ Base	reserved	reserved	DPV1_Status_1
8	reserved								DPV1_Status_2
9	reserved								DPV1_Status_3
10~ 243									User_Prm_Data

二、 DPV1/C1 功能

DSDPV1 支持 DPV1/C1 功能，服务存取点 SAP51(0x33)，DPV0/C1 功能主要包括写非循环输出数据(DS_write)、写非循环输入数据(DS_read)以及报警功能，DSDPV1 支持的 C1 通讯主要包括 DS_write、DS_read，DS_write 功能码 0x5E，DS_read 功能码 0x5F。

DSDPV1 内部设置 240bytes 缓存，用于缓存 DPV1/C1 非循环数据，当芯片收到主站请求报文时，判断请求报文的合法性，如果合法将报文发送至 DPRAM，同时置 B00F.0(芯片命令字节 2)为 1 ，标识 C1 输出数据有效，用户得到 C1 非循环数据请求后，检查请求槽索引的合法性，之后将应答数据写入 DPRAM。

DPRAM 存储 C1 非循环数据的结构如表 2DPRAM 定义，地址范围 B5D0~ B6C5。其中槽索引读写属性 Function_Num 定义如表 26。值得注意的是该段地址单元对于 DSDPV1 和用户来说都是可读可写的，也就是

说 DSDPV1 和用户是分时复用该地址单元，当 DSDPV1 要将请求数据写入 DPRAM 时，该地址单元支持用户读取，当用户读取 C1 非循环请求后做出应答，将响应数据写入 DPRAM 时，该地址对于用户来说是可写的，此时 DSDPV1 读取该地址单元应答数据。这样如果用户长时间未将应答数据写入 DPRAM，DSDPV1 将一直等待用户应答，即使主站再次发送 C1 请求，DSDPV1 认为此时上次请求尚未完毕，本次请求不予支持，也就是说 DSDPV1 不会再向 DPRAM 写入非循环请求数据，将会一直等待用户响应，除非用户产生应答，或者主站请求从站 DSDPV1 退出数据交换状态。

地址单元 B5D1、B5D2、B5D3，为复用地址单元：当 DSDPV1 将非循环输出数据写入 DPRAM 时或者用户将积非循环输入数据写入 DPRAM 时，B5D1 地址单元数据表示寻址槽号(Slot_Number)， B5D2 地址单元数据表示寻址索引号(Index)，B5D3 地址单元数据表示寻址槽索引数据长度，如果 DSDPV1 将非循环输入数据请求写入 DPRAM 时，B5D3 的值为 0，或者用户将非循环输出数据应答数据写入 DPRAM 时，B5D3 的值也为 0；当用户判断请求数据不合法的情况下，用户回复的 Function_num 就为表 27 所示，同时地址单元 B5D1、B5D2、B5D3 就依次定义为 Error_Decode、Error_code1 和 Error_code2，地址范围 B5D4~ B6C3 非循环数据此时无内容，Error_Decode 标准定义如表 28 所示，Error_code1 标准定义如表 29 所示，Error_code2 内容由用户指定，无统一规定。

表 26 C1 读写属性

Function_Num	读写属性
0x5E	Read
0x5F	Write

表 27 C1 错误属性

Function_Num	错误属性
0xDE	Read error
0xDF	Write error

表 28 Error_Decode

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	0	0	0	0	0	0	0

表 29 Error_code1 标准定义

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
0~9									
				0				应用错误	读出错（Read Error）
				1					写出错（Write Error）
				2					模块出错（Module Failure）
10				3~7					保留

	8		版本冲突（version conflict）
	9		功能不支持（feature not supported）
	10~15		用户指定内容（User specific）
11	0	接入错误	Index 非法（Invalid index）
	1		写数据长度出错（Write length error）
	2		Slot 非法（Invalid slot）
	3		类型矛盾（Type conflict）
	4		Area 非法（Invalid area）
	5		状态矛盾（State conflict）
	6		拒绝接入（Access denied）
	7		范围非法（Invalid range）
	8		参数非法（Invalid parameter）
	9		类型非法（Invalid type）
	10~15		用户指定内容（User specific）
12	0	资源错误	读限制冲突（User specific conflict）
	1		写限制冲突（Write constrain conflict）
	2		资源正忙（Resource busy）
	3		资源不存在（Resource unavailable）
	4~7		保留（reserved）
	8~15		用户指定（User specific）
13~15		用户指定	用户自定义

典型的 C1 通讯机制如图 9 所示，主站发送读非循环数据请求 DS_read.req，从站收到请求后立即回复短应答，并且告知用户需要准备主站所需的非循环数据，主站收到来自从站的短应答之后，便开始发送轮询报文(POLL)，用于询问从站是否准备好非循环数据，如果没有准备好，那么主站继续轮询从站，直到从站准备好非循环数据，从站向主站回复 DS_read.res 应答报文，并将非循环数据送至主站，这样便完成了一次 C1 非循环数据交换。DS_write(非循环写操作)和 DS_read 通讯机制一样。

图 9 PROFIBUS-DPV1/C1 读非循环数据通讯机制

三、 DPV1/C2 功能

DSDPV1 支持 DPV1/C2 功能。在 2 类主站和从站之间定义的服务是 C2 通讯，二类主站能够同时和多个从站建立 C2 非循环数据通讯。二类主站通过 C2 通信可同时与不同的从站并行地建立多个通信通道，C2 从站定义的服务有 5 种，包括：

1. Initiate：建立连接(通道)。
2. DS_Read：读非循环数据。
3. DS_Write：写非循环数据。
4. DS_Transport：数据传输(包括读写非循环数据)。
5. Abort：中止通道连接。

DSDPV1 支持 2 通道的 C2 非循环数据交换，内部设置 2 个 240bytes 缓存。当主站发起 C2 通道连接请求(Initiate.req)时，DSDPV1 判断请求报文的合法性，并且查询本地资源管理器(RM_Registry)，如表 30，如果两通道中有其中之一的占用状态为未使用，则向主站分配通道号，回复 Immediate.res，并且记录主站地址，标注占用状态为使用以及 C2 连接状态尚未完成等，随后 DSDPV1 等待主站轮询,之后从站回复 Initiate.res，并且标注连接状态为完成，至此主从之间 C2 通道建立连接。建立通道连接的过程不需要用户 CPU 参与,这是由于 2 通道的资源管理由 DSDPV1 负责分配、记录和更改,两通道 SAP 分别为 0x30 和 0x2F，也就是说一个从站最多能同时和两个主站进行 C2 非循环数据交换或者和一个主站进行两个通道的 C2 非循环数据交换。

表 30 PROFIBUS-DPV1-C2 RM_Registry

通 道号	主站地址	SAP	占用状态	连接状态	非循环数据交换状态
1	Master_add1	0x30	In/not use	完成/没有完成	开始/没有开始

2	Master_add2	0x2F	In/not use	完成/没有完成	开始/没有开始

当主从建立 C2 连接，主站获得通道号（SAP0x30 或 0x2F），主站和从站就可以进行非循环数据交换，当主站发起 DS_Read 、DS_Write、DS_Transport 服务请求时，从站判断报文合法性，将合法的报文通过 DPRAM 发送至用户 CPU，置 B00F.1 有效，并将资源管理器中相应通道的非循环数据交换状态更新为开始。用户 CPU 读取数据后，继续判断请求的合法性，例如主站请求访问的槽索引号，用户是否支持等，如果合法用户将数据写入 DPRAM，并置，之后 DSDPV1 将数据回复主站，至此主从站完成了一次非循环数据交换。

DPRAM 存储 C2 非循环数据的结构如表 2DPRAM 定义，地址范围 B6D0~B7C5。其中槽索引读写属性 Function_Num 定义如表 31 所示。和 C1 一样，该段地址单元对于 DSDPV1 和用户来说都是可读可写的，也就是说 DSDPV1 和用户是分时复用该地址单元，当 DSDPV1 要将请求数据写入 DPRAM 时，该地址单元支持用户读取，当用户读取 C1 非循环请求后做出应答，将响应数据写入 DPRAM 时，该地址对于用户来说是可写的。如果用户长时间未将应答数据写入 DPRAM，DSDPV1 将一直等待用户应答，直至 DSDPV1 发现用户应答超时，此时 C2 非循环数据无效，主站再对从站轮询时，DSDPV1 发起中止 C2 连接响应，并且将资源管理器中相应通道的占用状态改为未使用，连接状态改为未连接，非循环数据交换状态改为未开始，释放当前通道。

表 31 C2-Function_Num

Function_Num	读写属性
0x5E	Read
0x5F	Write
0x51	Transport

和 C1 通讯一样，地址单元 B6D1、B6D2、B6D3，为复用地址单元：当 DSDPV1 将非循环输出数据写入 DPRAM 时或者用户将积非循环输入数据写入 DPRAM 时，B6D1 地址单元数据表示寻址槽号(Slot_Number)， B6D2 地址单元数据表示寻址索引号(Index)，B6D3 地址单元数据表示寻址槽索引数据长度，如果 DSDPV1 将非循环输入数据请求写入 DPRAM 时，B6D3 的值为 0，或者用户将非循环输出数据应答数据写入 DPRAM 时，B6D3 的值也为 0；当用户判断请求数据不合法的情况下，用户回复的 Function_num 就为表 32 所示，同时地址单元 B6D1、B6D2、B6D3 就依次定义为 Error_Decode、Error_code1 和 Error_code2，地址范围 B6D4~B7C3 非循环数据此时无内容，Error_Decode 标准定义如表 28 所示，Error_code1 标准定义如表 29 所示，Error_code2 内容由用户指定，无统一规定。

表 32 C2 错误属性

Function_Num	错误属性
0xDE	Read error

0xDF	Write error
0XD1	Transport error

典型的 C2 非循环数据交换如图 10 所示，主从进行 C2 非循环数据交换之前，首先要建立通道连接，首先由主站发起 C2 通道初始化连接请求 initiate.req，该报文 SAP 服务访问点为 0x31，非循环通讯功能码(Function_Num)为 0x57，从站收到请求后，判断报文的合法性，检查资源管理器是否有可用的通道，记录相关信息后立刻向主站发送立即应答报文 immediate.res，告知主站从站为该主站本次请求连接分配的 SAP 服务访问点，主站随后利用此 SAP 向从站发送轮询报文，向从站确认是否可以开通，如果从站确认并认为此次通信是合法的，那么向主站回复 initiate.res 报文表示确认。此时，从站便一直等待主站发送非循环数据交换请求，其流程和 C1 交换基本一致，需要注意的是在 C2 通讯过程中，主从任何一方都有权中止当前连接。

图 10 PROFIBUS-DPV1/C1 读非循环数据通讯机制

对于 C2 非循环数据交换，DSDPV1 内部集成了 3 类看门狗定时器，I_timer(主站数据交换请求超时定时器)、F_timer(主站报文超时定时器)和 U_timer(用户响应超时定时器)，这三个定时器的作用是监控主从通讯状态，一旦主站或用户 CPU 任何一方超时未做出请求或应答，那么从站 DSDPV1 有权中止当前 C2 通道连接，F_Timer 和 I_Timer 都是从站监控主站请求行为的定时器，当定时器期满，主站还未做出正确的请求，从站 DSDPV1 在应答时中止当前连接，释放 RM 资源管理器。U_Timer 是从站监控用户应答行为的定时器，如果用户在定时期满仍未向 DSDPV1 做出应答，从站芯片也将中止当前连接，释放 RM 资源管理器。图 11 说明了三种定时器的工作方式，当从站芯片未连接任何通道时(图中 Idle_state)状态，三个定时器都处于关闭状态，直到某主站发起 C2 连接请求 initiate.req 时，从站记录下 Send_Timeout 时间，利用式(1)确定三个定时器的定时时间，之后向主站回复 immediate.res，启动 F_Timer，等待主站轮询从站，保持 U_timer 和 I_Timer 为关闭状态，如果在 F_Timer 未满的情况下，DSDPV1 收到主站的轮询报文，关闭 F_Timer，保持 U_Timer 为关闭状态，开启 I_Timer，向主站回复 initiate.res，等待主站非循环数据交换请求，如果

I_Timer 定时未满的情况下，从站收到主站的非循环数据交换请求(DS_read/DS_write/DS_transport.req)，DSDPV1 关闭 I_Timer，向主站回复短应答(E5)，开启 F_Timer，监视主站轮询报文，并将请求通知用户，等待用户应答，开启 U_Timer，监视用户是否超时未响应，当主站发送轮询报文时，DSDPV1 查看用户是否准备好应答，F_Timer 重新计数， I_Timer 处于关闭状态，如果用户已准备好应答，则关闭 U_Timer，关闭 F_Timer，向主站回复(DS_read/DS_write/DS_transport.res)，同时开启 I_Timer，等待主站再一次发起非循环数据交换请求。需要用户注意的是，如果 U_Timer 超时，用户未作出应答，那么 DSDPV1 中止当前连接，释放 RM 资源管理器。

$$\begin{aligned} F\_Time &= Send\_Timeout \\ U\_Time &= Send\_Timeout \\ I\_Time &= 2 * Send\_Timeout \end{aligned} \qquad (1)$$

表 33 C2 定时器

定时器	位数	作用
F_Timer	16bit	主站数据交换请求超时定时器
I_Timer	16bit	主站报文超时定时器
U_Timer	16bit	用户响应超时定时器

北京鼎实创新科技股份有限公司
现场总线 PROFIBUS（中国）技术资格中心
电话：010-82078031、010-62054940　　传真：010-82285084
地址：北京德胜门外教场口 1 号，5 号楼 A-1　　邮编：100120
Web:www.c-profibus.com.cn　　Email: tangjy@c-profibus.com.cn