

# 通用 X2C 插件库使用说明

## 1.概述

该应用库提供使用的初始化方法和工作方法，在应用项目的主循环模式下，在单次串口发送时长间隔内进行调用处理方法，即软件查询串口下主机命令，根据主机意图完成与项目程序的数据交换，包括读取、修改、连续监控等功能模式。

使用 KF32 MotorDebugTool 上位机软件实现与从机设备的信息交换，即可可视化查看运行过程值或监控值运行时变化趋势。

## 2.组成

头文件与库文件已集成在 **ChipON** 集成开发环境中。

头文件	<b>SeriesDIServer.h</b>
库文件	<b>libSeriesDIServices.a</b>
APP 文件	(引用 配置串口 初始化 调用)

## 3.使用配置说明

该说明下代码使用外设库的服务方法，并根据需要进行外设库方法的调整，也可以重命名复制独立副本。

### ➤ 包含头文件

```
#include "SeriesDIServer.h"
```

### ➤ 声明使用串口

```
#define X2C_Usart_SEL USART0_SFR
```

## ➤ 添加库接口串口方法

```
// 发
static void (sendSerialFcn)(uint8_t data)
{
    USART_SendData(X2C_Usart_SEL,data); // 基本串口发送方法
}
// 收
static uint8_t (receiveSerialFcn)()
{
    return USART_ReceiveData(X2C_Usart_SEL); // 基本串口读接收方法
}
// 是否有一个要接收字节
static uint8_t (isReceiveDataAvailableFcn)()
{
    uint8_t flag=USART_Get_Receive_BUFR_Ready_Flag(X2C_Usart_SEL);
    // X2C_Usart_SEL->STR |= 0x03FF0000; // 各种错误标志清零
    // X2C_Usart_SEL->STR &= 0x0000FFFF; // 各种错误标志允许
    return flag;
}
// 是否发送空闲状态监测
static uint8_t (isSendReadyFcn)()
{
    // SFR_SET_BIT_ASM(X2C_Usart_SEL->STR, USART_STR_TEIC_POS);
    // SFR_CLR_BIT_ASM(X2C_Usart_SEL->STR, USART_STR_TEIC_POS);
    return USART_Get_Transmitter_Empty_Flag(X2C_Usart_SEL);
}
```

## ➤ 初始化进程初始化

```
//~~~~~应用级~~~~~
GPIO_USART(); // 配置使用串口的功能引脚
USART_Async_config(X2C_Usart_SEL); // 外设库样例配置串口模式、波特率等信息，但不开启串口中断使能
//~~~~~库工作初始化~~~~~
X2CScope_HookUARTFunctions(sendSerialFcn, receiveSerialFcn, isReceiveDataAvailableFcn, isSendReadyFcn);
X2CScope_Initialise();
```

## ➤ 主循环周期调用处理方法

```
X2CScope_Communicate();  
X2CScope_Update(); // 主循环或定时间隔调用
```

## 附录 库使用涉及外设库部分代码

```
void  
main()  
{  
    SystemInit();  
    GPIO_USART();  
    USART_Async_config(X2C_Usart_SEL);  
    X2CScope_HookUARTFunctions(sendSerialFcn,  
                                receiveSerialFcn,  
                                isReceiveDataAvailableFcn,  
                                isSendReadyFcn);  
  
    X2CScope_Initialise();  
    While(1)  
    {  
        X2CScope_Communicate(); // <0.8byte 串口传输时间调用  
        X2CScope_Update();  
    }  
}  
  
void  
USART_SendData(USART_SFRmap* USARTx, uint8_t Data)  
{  
    /* 参数校验 */  
    CHECK_RESTRICTION(CHECK_USART_ALL_PERIPH(USARTx));  
    /*----- 设置 USART_TBUFR 寄存器 -----*/  
    USARTx->TBUFR = Data;  
}  
  
uint32_t  
USART_ReceiveData(USART_SFRmap* USARTx)  
{  
    /* 参数校验 */  
    CHECK_RESTRICTION(CHECK_USART_ALL_PERIPH(USARTx));  
    /*----- 设置 USART_RBUFR 寄存器 -----*/  
    return USARTx->RBUFR;  
}
```

```

}
FlagStatus
USART_Get_Receive_BUFReady_Flag (USART_SFRmap* USARTx)
{
    /* 参数校验 */
    CHECK_RESTRICTION(CHECK_USART_ALL_PERIPH(USARTx));
    /*----- 读取 USART_STR 寄存器 RDRIF 位 -----*/
    if (USARTx->STR & USART_STR_RDRIF)
    {
        /* USART 接收 BUF 中有数据可读*/
        return SET;
    }
    else
    {
        /* USART 接收 BUF 中无数据可读 */
        return RESET;
    }
}
FlagStatus
USART_Get_Transmitter_Empty_Flag (USART_SFRmap* USARTx)
{
    /* 参数校验 */
    CHECK_RESTRICTION(CHECK_USART_ALL_PERIPH(USARTx));
    /*----- 读取 USART_STR 寄存器 TXEIF 位 -----*/
    if (USARTx->STR & USART_STR_TXEIF)
    {
        /* USART 发射器为空*/
        return SET;
    }
    else
    {
        /* USART 发射器不为空 */
        return RESET;
    }
}

```

# 附录 部分通信协议简介

## ➤ 交换通信协议格式

帧头 数据长度 节点号 长度下数据 校验和 【特殊见说明】

帧头: 0x55

节点号: 0x01

数据: 首数据为功能编号, 不同的功能号下具有其需要的数据格式。

0x00 获取设备信息

0x01 获取设备状态

0x02 设置设备状态

0x11 获取设备参数

0x12 设置设备参数

0x07 块读取数据

0x08 块修改数据

0x09 读取 RAM

0x0A 修改 RAM

当为应答时, 仅挨功能号发生应答结果状态, 0 为识别通过,

校验和: 从帧头数据开始的字节数据的加和值。

其他: 当长度或数据内容遇到 0x55 或 0x02 下, 额外发送一个 0x00。当计算的校验和为 0x55 或 0x02, 传输反码 0xAA 或 0xFD。

举例

获取 RAM 块下发 55 07 01 09 00 00 00 10 04 01 7B 应答接收 55 06 01 09 00 00 00 00 65

## ➤ 同步上传通信协议格式

帧头 数据长度 节点号 长度下数据 校验和 【特殊见说明】

帧头: 0x02

数据长度: 至少为 2

节点号: 0x01

数据: 前两个字节为库运行时间戳, 即主循环调用周期数。

随后为最多 8 组监控的数据内容。

校验和: 从帧头数据开始的字节数据的加和值。

其他: 当长度或数据内容遇到 0x55 或 0x02 下, 不额外发送 0x00。当计算的校验和为 0x02, 传输反码 0xFD。当开启同步下, 每次周期扫描均会上传配置的同步请求内容, 若需停止, 发送 0x04。另外同步上传可能插入到交换应答的结果发生过程。

举例

间隔接收 02 06 01 00 00 00 00 02 10 1B

打断接收 0x55 0x02 0x00 02 06 01 00 00 00 00 02 10 1B 0x01 0x09 0x61

## ➤ 错误编码

```
#define ERRORSuccess                ((uint8)0x00) // 无错误
#define ERRORChksum                 ((uint8)0x13) // 校验和异常
#define ERRORFormat                 ((uint8)0x14) // 格式异常
#define ERRORSizeTooLarge           ((uint8)0x15) // 类型大小异常
#define ERRORSERVICENotAvail        ((uint8)0x21) // 服务未就绪
#define SVErrInvalidDspState        ((uint8)0x22) // 错误机器状态
#define ERRORFlashWrite             ((uint8)0x30) // flash 写
#define ERRORFlashWriteProtect      ((uint8)0x31) // flash 写保护
#define SVErrInvalidParamId         ((uint8)0x40) // 错误的参数
#define ERRORBlkID                  ((uint8)0x41) // blkID
#define ERRORParLimit               ((uint8)0x42) // 对限定
#define SVErrParamTableNotInit      ((uint8)0x43) // 参数表未初始化
#define SVErrFncTableNotInit        ((uint8)0x44) // 函数表未初始化
#define ERRORPowerIsOn              ((uint8)0x50) // 已上电工作
```

## ➤ 举例 RAM 操作协议说明

### 获取下发

数据 0: 功能码 0x09

数据 1~4: 小格式 RAM 地址

数据 5: 字节大小, 即读取 RAM 长度

数据 6: 类型, 即数据读取加载模型 1、2、4 对应读模式 8bit 16bit 32bit

### 获取应答

数据 0: 功能码 0x09

数据 1: 结果码 0x00

数据 2~X: 读取数据结果

### 修改下发

数据 0: 功能码 0x09

数据 1~4: 小格式 RAM 地址

数据 5: 类型, 即数据写模型 1、2、4 对应读模式 8bit 16bit 32bit

数据 6~X: 待写 RAM 小格式数据

### 修改应答

数据 0: 功能码 0x09

数据 1: 结果码 0x00

## ➤ 举例修改配置操作协议说明

获取参数：55 03 01 11 01 00 6B

应答数据 0:0x11，数据 1:0x00 成功

数据 3~N：工作状态、通道数量、采样时间 2、数据使用偏移 4、数组地址 4、延迟时间 4、时间位置 4、最大使用长度 4、数组容量大小 4、版本信息。

设置下发

数据 0：功能码 0x12

数据 1~2:配置 id，0x01 0x00

数据 3：状态:同步上传、离线采样、触发采样等。

数据 4：监控通道数 1~8。

数据 5~6：时间间隔因子。

---

数据 7~7+6N： 1N 字节：源的类型，非地址时 2N 字节块函数 id，2N 参数 id。地址时 4N 字节地址。1N 数据字节长度。

---

数据+1： 触发数据类型： 低 4bit 为数据类型字节大小，1bit 保留 1bit 符号，1bit 整数或浮点，1bit 固定为 1，即 0x80|0xXX。

数据 +1+4：源的类型，非地址时 2N 字节块函数 id，2N 参数 id。地址时 4N 字节地址

数据 +4：触发水平值

数据 +4 触发延迟

数据 +1 触发边缘。

可选数据+1 是否 0x01 的立即锁定触发原参考值，默认无数据下锁定。