文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 SIO 应用指南

目录

1 概述 1

2. 适用的硬件 1

3. 应用方案代码说明 1

3.1 主要特性： 1

3.2 管脚描述： 1

3.3 功能模块： 2

3.4 发送软件流程图： 3

3.5 发送软件配置： 4

3.6 接收软件流程图： 10

3.7 接收软件配置： 11

4. 程序下载和运行 15

1 概述

本文介绍了在 APT32F110x 中 SIO 模块的应用，以驱动 LED 芯片 2812 为例。

2. 适用的硬件

该例程使用于 APT32F110x 系列学习板

3. 应用方案代码说明

基于 APT32F110x 完整的 CSI 库文件系统，进行配置 SIO 模块。

3.1 主要特性

串行输入输出 SIO 可模拟多种串行通讯协议。

- 单线通信管脚，支持双向数据传输
- 可通过配置时钟分频得到多种通信速率。
- 接受模式需同步开始标志
- 接收模式下，可调整每位 BIT 的采集数和抽样点
- 接收模式下，可灵活配置输入滤波

3.2 管脚描述

管脚名称	功能		I/O类型	有效电平	说明
SIO	SIO输入输出口		数字	-	-

图 3.2.1 管脚描述

管脚名称 SIO 是此模块在 GPIO 上映射的双向输入输出端口。可在**数据手册 2.3 管脚功能分配表**中查询。

PA0.1,PA0.3,PA0.5,PA013,PB0.1,PB0.5,PC0.0

3.3 功能模块

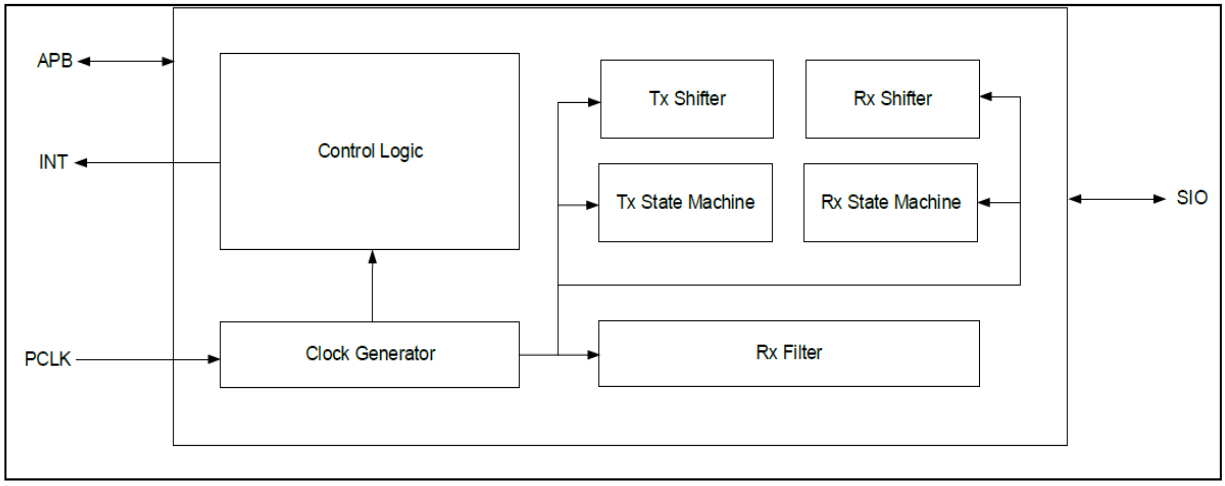

图 3.3.1 功能框图

3.4 发送软件流程图

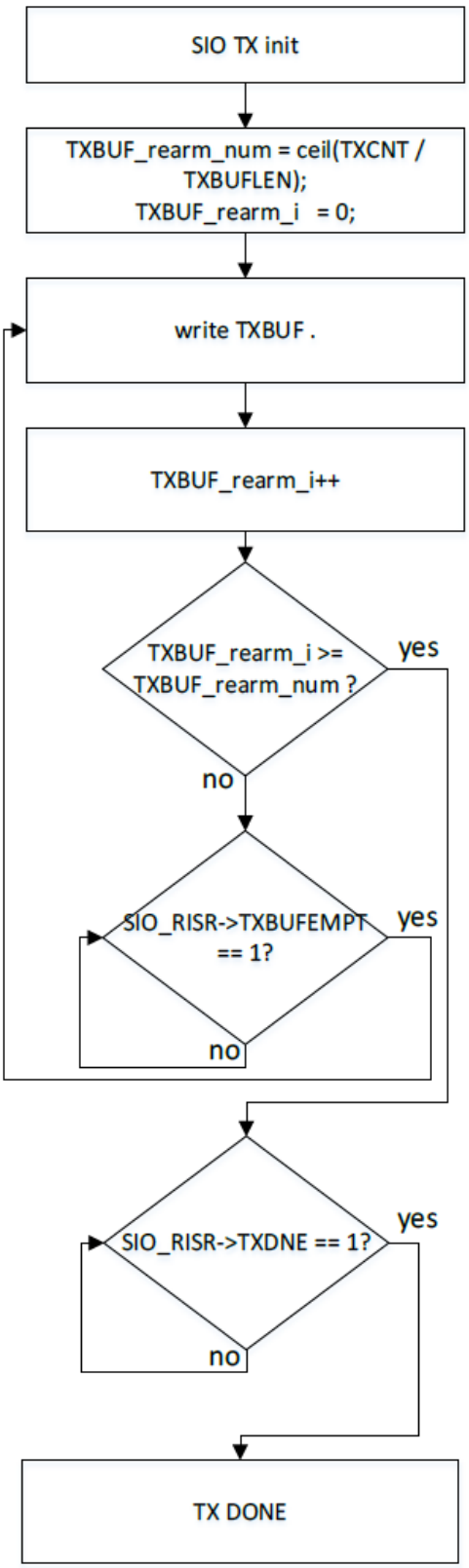

图 3.4.1 发送流程图

3.5 发送软件配置

可在 system.c 文件中 sio_tx_config()函数进行初始化的配置。

```
void sio_tx_config(void)
{
    csi_sio_tx_config_t tSioTxCfg;
//ref pin(PA0.1,PA0.3,PA0.5,PA013,PB0.1,PB0.5,PC0.0)
    //csi_pin_set_mux(PA01, PA01_SIO);
    //csi_pin_set_mux(PA03, PA03_SIO);
    //csi_pin_set_mux(PA05, PA05_SIO);
    //csi_pin_set_mux(PA013, PA05_SIO);
    //csi_pin_set_mux(PB01, PB01_SIO);
    csi_pin_set_mux(PC00, PC00_SIO);
    //csi_pin_set_mux(PB05, PB05_SIO);
    //SIO TX 参数配置
    tSioTxCfg.byD0Len          = 1;
    tSioTxCfg.byD1Len          = 1;
    tSioTxCfg.byDLLen          = 4;
    tSioTxCfg.byDHLen          = 4;
    tSioTxCfg.byDLLsq          = 0x01;
    tSioTxCfg.byDHHsq          = 0x07;
    tSioTxCfg.byTxBufLen       = 8;
    tSioTxCfg.byTxCnt          = 24;
    tSioTxCfg.byIdleLev  = SIO_IDLE_L;
    tSioTxCfg.byTxDir          = SIO_TXDIR_LSB;
    tSioTxCfg.wTxFreq          = 4000000;
    tSioTxCfg.byInt            = SIO_INTSRC_NONE;
    csi_sio_tx_init(SIO0, &tSioTxCfg);
}
```

- **代码说明：**

1. **csi_pin_set_mux(PC00, PC00_SIO);**

   用于配置 SIO 映射到 GPIO 上的 PIN 脚输入输出口

   可选的 PIN 脚有：PA0.1, PA0.3, PA0.5, PA013, PB0.1, PB0.5, PC0.0

2. **csi_sio_tx_init(SIO0, &tSioTxCfg);**

   根据结构体变量的赋值，对 SIO 模块配置（驱动 2812 LED 芯片）

   tSioTxCfg.byD0Len---设置 D0 的长度，1 为不需要 D0

   tSioTxCfg.byD1Len---设置 D1 的长度，1 为不需要 D1

   tSioTxCfg.byDLLen---设置 DL 电平的长度，4 为 3 个 BIT 的 TICK 的时间长度，最长 8BITS

   tSioTxCfg.byDHLen---设置 DH 电平的长度，4 为 3 个 TICK 的时间长度，最长 8BITS

   tSioTxCfg.byDLLsq---设置 DL 发送的序列，0x01 为每个 TICK 对应的电平 0，0，0，1
   从 LSB 到 MSB 的顺序，长度为 tSioTxCfg.byDLLen 的值

   tSioTxCfg.byDHHsq ---设置 DH 发送的序列，0X07 为为每个 TICK 对应的电平 0，1，1，1
   从 LSB 到 MSB 的顺序，长度为 tSioTxCfg.byDHLen 的值

   tSioTxCfg.byTxBufLen---设置 TXBUF 的传送长度为 8BITS，最大为 16BITS

   tSioTxCfg.byTxCnt---设置 TXBUF 完成传送的次数计数。设置为 24，代表发送 24 个 BYTES 数据

   tSioTxCfg.byIdleLev---空闲时输出状态选择：0，高阻态 1，高电平 2 低电平 3 高阻态

   tSioTxCfg.byTxDir---TXBUF 发送时移位方向：0 ，LSB 开始 1，MSB 开始

   tSioTxCfg.wTxFreq---设置发送的频率。根据频率计算每个 TICK 的周期，由此计算 D0,D1,DH,DL 的周期，如：4000000， tx clk =4MHz, Ttxshift = 1/4 = 250ns。DH 和 DL 的周期为 1us,DH 的高电平为 750ns,DL 的高电

平为 250ns，满足 LED 芯片 2812 的时序。

tSioTxCfg.byInt---不使用中断。目前只支持轮询的模式。

在 main.c 文件中

```
    //rgb 原始数据
    uint8_t byDipData[3] =
    {
     //R            G            B
     0x01,  0x02,  0x03,//rgb1
    //      0x00,  0x07,  0x00,//rgb2
    //      0x00,  0x07,  0x00,//rgb3
    //      0x00,  0x07,  0x00,//rgb4
    //      0x00,  0x07,  0x00,//rgb5
    //      0x00,  0x07,  0x00,//rgb6
    //      0x00,  0x07,  0x00,//rgb7
    //      0x00,  0x07,  0x00 //rgb8
    };


static uint32_t sio_led_data_conver(uint8_t byData)
{
    uint8_t i;
    uint32_t wData = 0xaaaa;
    for(i = 0; i < 8; i++)
    {
        if(byData & 0x80)
            wData |= (0x01 << (2*i));
        byData = byData << 1;
    }
    return wData;
}


static void set_led_rgb_store(uint32_t *pwLeddData,uint16_t hwLedNum)
{


    *(pwLeddData+hwLedNum*3) = sio_led_data_conver(byDipData[hwLedNum*3+1]) ;
    *(pwLeddData+hwLedNum*3+1) = sio_led_data_conver(byDipData[hwLedNum*3]);
    *(pwLeddData+hwLedNum*3+2) = sio_led_data_conver(byDipData[hwLedNum*3+2]);
}


void led_rgb_display(uint8_t *byColData, uint16_t hwLedNum)
```

```
{
        uint16_t i;
        uint32_t wRgbData[24];
        for(i = 0; i < hwLedNum; i++)
        {
                set_led_rgb_store(wRgbData,i);
        }
        for(i = 0; i < hwLedNum; i++)
                csi_sio_send(SIO0, &wRgbData[3*i],3);
}
       //-------------------------------------------------------------------------
int main()
{
        mdelay(500);
        system_init();
        board_init();
        my_printf("Hello World-~~~~~~~\n");
        while(1)
        {
                 CK_CPU_DISALLNORMALIRQ;
             led_rgb_display(byDipData, 1);
            CK_CPU_ENALLNORMALIRQ;
        }
             return 0;
}
```

- **代码说明:**

1. **led_rgb_display(byDipData, 1);**

   从数组 byDipData 中取预设的灯串第一个节点的 RGB 显示数据，3 个 BTYES 的数据。

   此处为了验证时序，所以显示效果数组取的最短 1 个节点的数据，重复发送。

2. **static void set_led_rgb_store(uint32_t *pwLeddData,uint16_t hwLedNum)**

   数组第一列和第二列的位置交换，按照 G,R,B 的顺序。实际上也可以编辑数组 byDipData 中第一列和第二列的位置互换。

3. **static uint32_t sio_led_data_conver(uint8_t byData)**

   把数组里的逻辑“1”和“0”转换为寄存器 TXBUF 里的对应编码值

4. **CK_CPU_DISALLNORMALIRQ**

禁止所有中断

5. CK_CPU_ENALLNORMALIRQ

使能中断

- **波形图：**

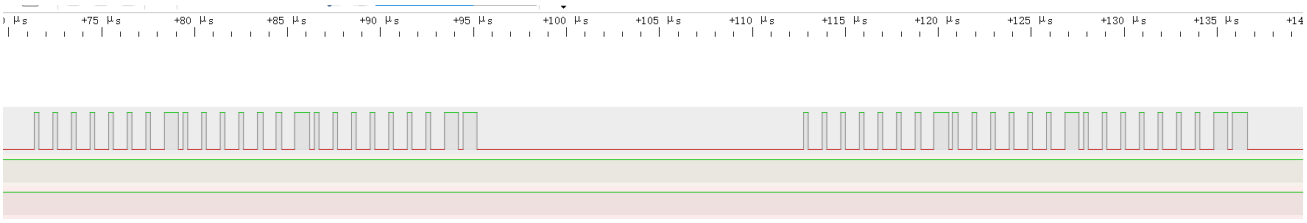

图 3.5.1 波形图

上图中是程序运行了两次 led_rgb_display(byDipData, 1)的波形;

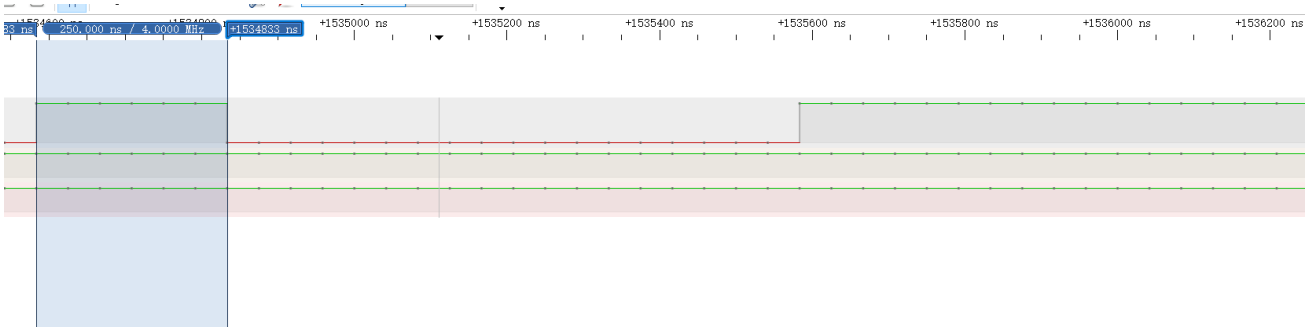

图 3.5.2 逻辑"0"高电平

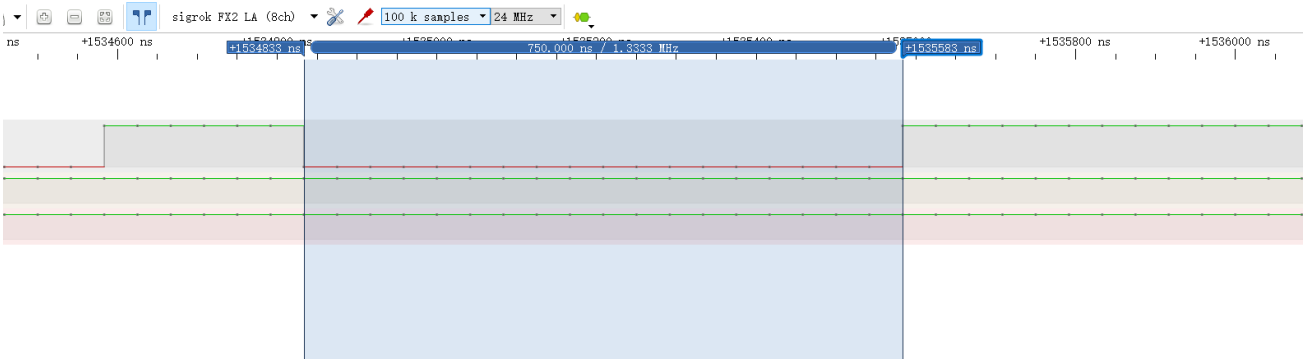

图 3.5.3 逻辑"0"低电平

上面两图是逻辑“0”的高、低电平部分

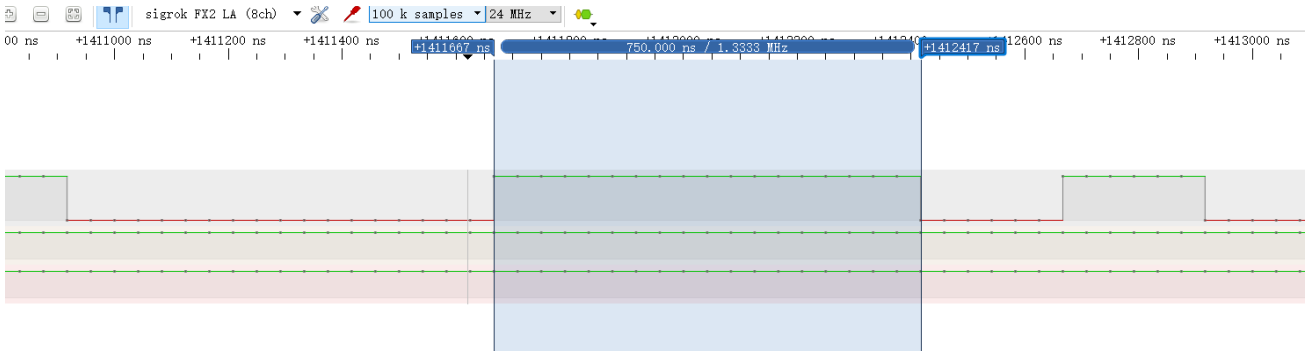

图 3.5.4 逻辑"1"高电平

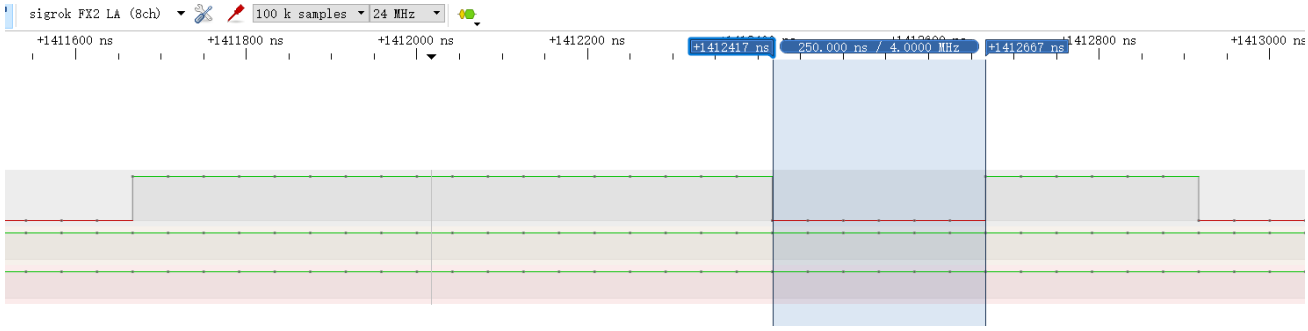

图 3.5.5 逻辑"1"低电平

上面两图是逻辑“1”的高低电平

图 3.5.6 数据间隔时间

上图是 2 组数据间的间隔时间，轮询的方式发送，取决于代码的逻辑

3.6 接收软件流程图

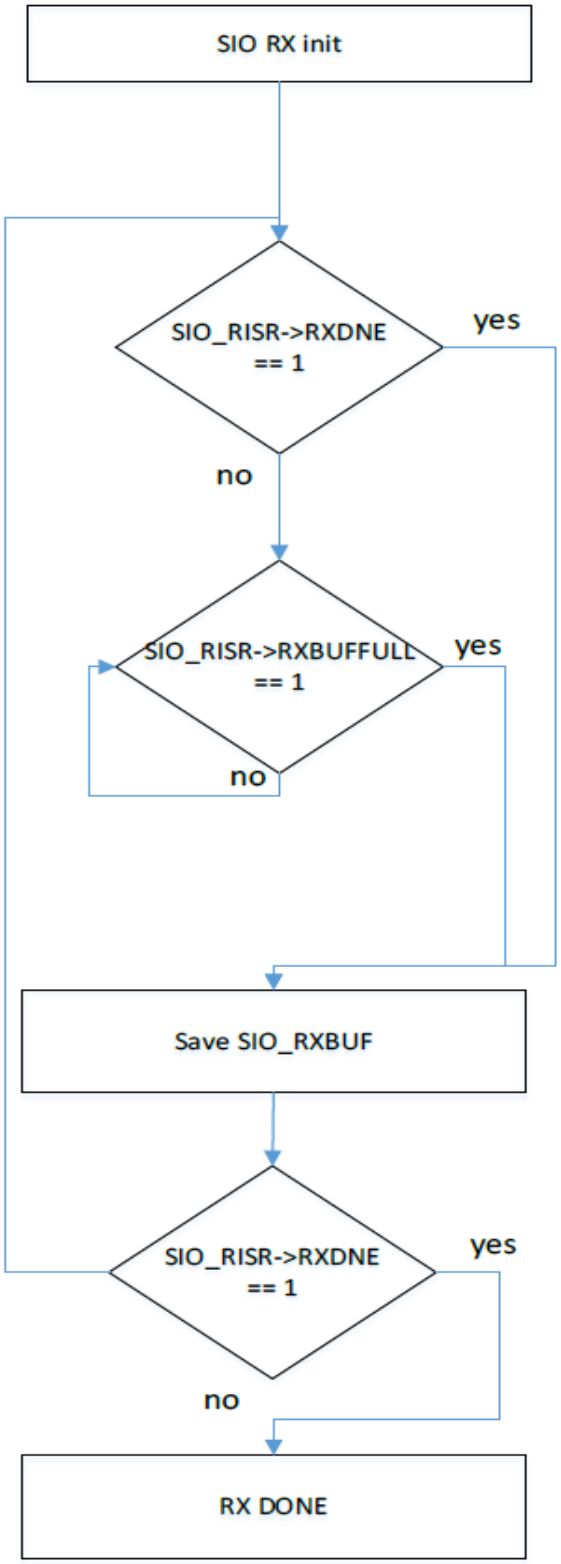

图 3.6.1 接收流程图

3.7 接收软件配置

可在 system.c 文件中 sio_rrx_config()函数进行初始化的配置,接收的目标数据为 3.5 章节的发送内容。数据为 0x01,x02,x03 重复。

```
void sio_rx_config(void)
{
    csi_sio_rx_config_t tSioRxCfg;

    //配置 GPIO 为 SIO 模式
    //ref pin(PA0.1,PA0.3,PA0.5,PA013,PB0.1,PB0.5,PC0.0)

    //csi_pin_set_mux(PA01, PA01_SIO);
    //csi_pin_set_mux(PA03, PA03_SIO);
    //csi_pin_set_mux(PA05, PA05_SIO);
    //csi_pin_set_mux(PA013, PA05_SIO);
    //csi_pin_set_mux(PB01, PB01_SIO);
    csi_pin_set_mux(PC00, PC00_SIO);
    //csi_pin_set_mux(PB05, PB05_SIO);

    //SIO RX 参数配置
    tSioRxCfg.byDebPerLen      = 3;
    tSioRxCfg.byDebClkDiv      = 2;
    tSioRxCfg.byTrgEdge        = SIO_TRG_RISE;
    tSioRxCfg.byTrgMode        = SIO_TRGMD_DEB;
    tSioRxCfg.byRxDir          = SIO_RXDIR_LSB;
    tSioRxCfg.bySpMode         = SIO_SPMD_EDGE_EN;
    tSioRxCfg.bySpExtra        = SIO_EXTRACT_HI;
    tSioRxCfg.byHithr          = 4;
    tSioRxCfg.byRxBufLen       = 8;
    tSioRxCfg.byRxCnt          = 24;
    tSioRxCfg.wRxFreq          = 8000000;
    tSioRxCfg.bySpBitLen       = 8;
    tSioRxCfg.byInt        = SIO_INTSRC_RXDNE;

    csi_sio_rx_init(SIO0, &tSioRxCfg);
    csi_sio_timeout_rst(SIO0, 10, ENABLE);
    csi_sio_set_buffer(g_wSioRxBuf, 24);

}
```

- **代码说明：**

1. **csi_pin_set_mux(PC00, PC00_SIO);**

   配置 SIO 的输入接收 GPIO 口

   可选的 PIN 脚有 PA0.1,PA0.3,PA0.5,PA013,PB0.1,PB0.5,PC0.0

2. **csi_sio_rx_init(SIO0, &tSioRxCfg);**

   根据结构体 tSioRxCfg 的初始化，对 SIO 接收模块初始化。

   tSioRxCfg.byDebPerLen ---接收去抖滤波周期，以 SIO 模块的时钟为基础

   tSioRxCfg.byDebClkDiv --- 接收滤波时钟分频比设置

   tSioRxCfg.byTrgEdge --- 接收采样触发边沿设置，要根据接收的波形来确定

   tSioRxCfg.byTrgMode --- 接收采样触发模式，沿触发后以设置的去抖周期为准 OR 去抖延时 30ns

   tSioRxCfg.byRxDir --- 接收数据方向，LSB->MSB OR MSB-LSB,根据发送数据的方向确定

   tSioRxCfg.bySpMode --- 接收采样数据边沿对齐使能

   tSioRxCfg.wRxFreq --- 接收模块的采集频率设置

   tSioRxCfg.bySpExtra --- 采样提取的策略

   设置为 0~0x1f：触发采样后以第 N（设置的值）个 TICK 位的电平为采样结果

   设置为0x20：以HITHR位置设置的值作为接收数据采集高的判断阈值，需大于此值

   tSioRxCfg.byHithr---接收原始数据采集高的判断阈值

   tSioRxCfg.byRxBufLen---接收数据 BUF 的长短设置（8BIT）

   tSioRxCfg.byRxCnt---接收总的数据长度设置（24BIT）

   tSioRxCfg.wRxFreq--- 采样的时钟频率设置

   tSioRxCfg.bySpBitLen---采样 1 个 BIT 位的次数

   tSioRxCfg.byInt --- 中断使能 SIO_INTSRC_RXDNE 接收效率高于 SIO_INTSRC_RXBUFFULL

3. **csi_sio_timeout_rst(SIO0, 10, ENABLE);**

   接收超时复位使能和配置

4. **csi_sio_set_buffer(g_wSioRxBuf, 24);**

   初始化接收数据的结构体 g_tSioTran g_t，配置接收缓存和长度，长度根据实际使用设置。

在 SIO.C 模块中，中断函数__attribute__((weak)) void sio_irqhandler(csp_sio_t *ptSioBase)里

```
case SIO_RXDNE:
        csp_sio_clr_isr(ptSioBase, SIO_RXDNE | SIO_RXBUFFULL);
        if(NULL == g_tSioTran.pwData || 0 == g_tSioTran.hwSize)
        {
                csp_sio_get_rxbuf(ptSioBase);
                g_tSioTran.byRxStat = SIO_STATE_ERROR;
        }
        else
        {
                *(g_tSioTran.pwData+g_tSioTran.hwTranLen)=csp_sio_get_rxbuf(ptSioBase);
                if(g_tSioTran.hwTranLen < g_tSioTran.hwSize)
                        g_tSioTran.hwTranLen ++;
                else
                        g_tSioTran.byRxStat = SIO_STATE_FULL;
        }
break;
```

5. **csp_sio_clr_isr(ptSioBase, SIO_RXDNE | SIO_RXBUFFULL);**

   清掉置位的中断标志位

6. **csp_sio_get_rxbuf(ptSioBase);**

   从硬件 32BIT 的 BUF 中取接收数据,放入*g_tSioTran.pwData 这个缓存中，一次 3 个 BYTE 的数据 G,B,R

在 main.c 中(此处只会打印一次)

```
iRet = csi_sio_receive(SIO0, wRxBuf, 24);
        if(iRet == 24)
        {
            for(i=0;i<24;i++)
                my_printf("wRxBuf[%d]:    0x%x \n", i,wRxBuf[i]);
        }
        while(1);
```

7. csi_sio_receive(SIO0, wRxBuf, 24);

从*g_tSioTran.pwData 中取出来放入局部变量中使用

8. my_printf("wRxBuf[%d]: 0x%x \n", i,wRxBuf[i]);

打印出 wRxBuf 所有的接收数据

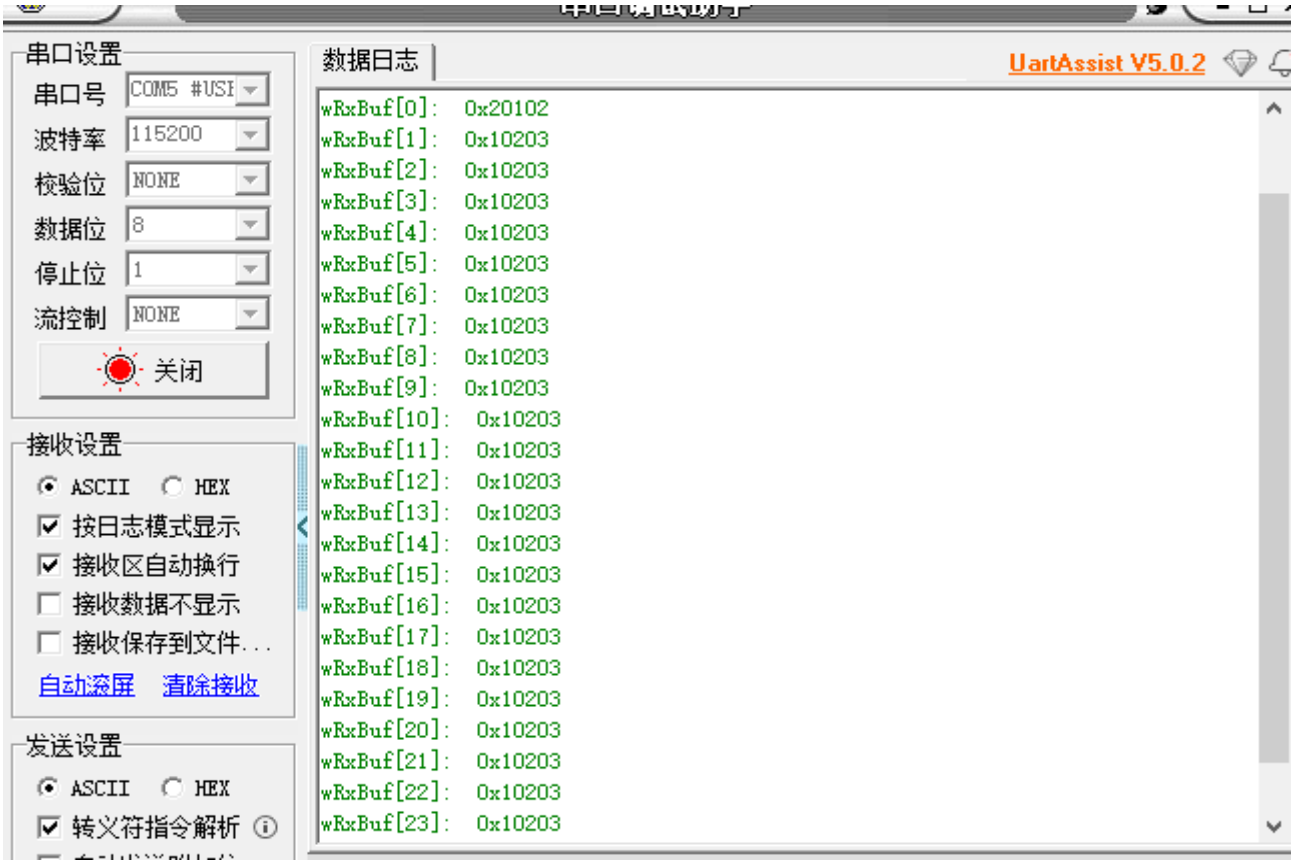

图 3.7.1 接收数据

4. 程序下载和运行

1．将目标板与仿真器连接，分别为 VDD SCLK SWIO GND

2. 程序编译后仿真运行

3. 通过串口调试工具查看打印输出，符合发送的数据。