文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 UART 应用指南

目录

1 概述

本文介绍了在APT32F110x中UART模块的应用。

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 UART 配置

3.1 UART 基本特性

UART 是一个简单通用的异步串行接收和发送接口，支持 8 位的数据通信，支持校验位，每次发送都以一个停止位结束。模块包含 UART0,UART1,UART2。

当模块中的 FIFO 功能有效时，UART 发送和接收都会分别通过发送 FIFO 和接收 FIFO。接收 FIFO 是一个 8 位宽，8 地址深的先进先出缓冲区。

具有接收发送 Break 信号功能。Break 信号是一个逻辑低电平，持续时间至少一个帧长。

- 可配置的波特率　波特率 = PCLK / DIV
- 固定的8位发送长度，支持8个单独的收发FIFO
- 固定一位停止位
- 发送接收溢出检测
- 发送接收完成中断和溢出中断
- 支持4种校验位，奇偶校验和0/1校验
- 使用DMA实现连续通信

管脚名称	功能	I/O类型	有效电平	说明
UART_TX	UART发送数据线	O	–	–
UART_RX	UART接收数据线	I	–	–

图3.1.1管脚功能

3.2 功能框图

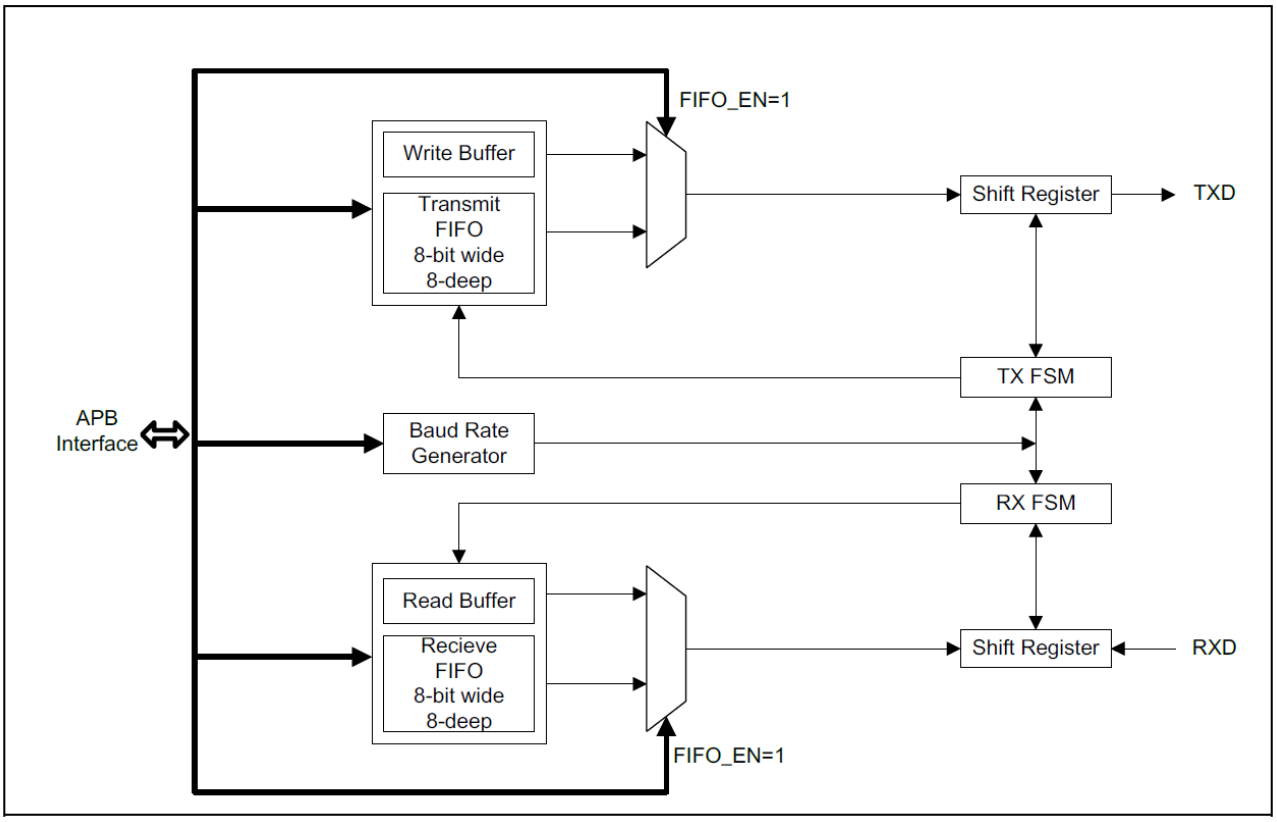

图 3.2.1UART 功能框图

3.3DMA 发送

在 SYSTEM.C 里，system_init()里初始化 uart1_config()；实现使用 FIFO 通过中断接收，DMA 轮询发送

```
void uart1_config(void)
{
    csi_uart_config_t tUartConfig;

    csi_pin_set_mux(PB02, PB02_UART1_TX);
    csi_pin_set_mux(PA06, PA06_UART1_RX);
    csi_pin_pull_mode(PA06,GPIO_PULLUP);
```

```
    csi_uart_set_buffer(UART1, &g_tRingbuf, g_byRxBuf, sizeof(g_byRxBuf));
    tUartConfig.byParity = UART_PARITY_NONE;
    tUartConfig.wBaudRate = 115200;
    tUartConfig.wInt = UART_INTSRC_RXFIFO;
    tUartConfig.byTxMode = UART_TX_MODE_POLL;
    tUartConfig.byRxMode = UART_RX_MODE_INT_FIX;
    csi_uart_init(UART1, &tUartConfig);
    csi_uart_start(UART1, UART_FUNC_RX_TX);
    csi_etb_init();
    csi_uart_dma_tx_init(UART1, DMA_CH1, ETB_CH10);
}
```

- **代码说明：**

1) **csi_pin_set_mux(PB02, PB02_UART1_TX);**

2) **csi_pin_set_mux(PA06, PA06_UART1_RX);**

   配置 UART1 的接收发送扣

3) **csi_pin_pull_mode(PA06,GPIO_PULLUP);**

   配置接受口为内部上拉

4) **csi_uart_set_buffer(UART1, &g_tRingbuf, g_byRxBuf, sizeof(g_byRxBuf));**

   实例化接收 ringbuf，将 ringbuf 接收数据缓存指向用户定义的的接收 buffer(g_byRxBuf)

   ringbuffer_t g_tRingbuf;

   uint8_t g_byRxBuf[UART_RECV_MAX_LEN];

   #define UART_RECV_MAX_LEN 128

5) **csi_uart_init(UART1, &tUartConfig);**

   根据结构体变量 **tUartConfig** 的赋值初始化 UART1 口

   tUartConfig.byParity---设置校验位：奇，偶，或者无

   tUartConfig.wBaudRate---设置波特率

tUartConfig.wInt---设置使用的中断

tUartConfig.byTxMode---设置发送模式

tUartConfig.byRxMode---设置接收模式

6) **csi_uart_start(UART1, UART_FUNC_RX_TX);**

启动 UART1 的 RX 和 TX 功能，也可单独开启其中一个

7) **csi_etb_init();**

使能 ETCB 模块

8) **csi_uart_dma_tx_init(UART1, DMA_CH1, ETB_CH10);**

选择 DMA 和 ETCB 通道

在中断函数 **void uart_irqhandler(csp_uart_t *ptUartBase,uint8_t byIdx)**中

```
case UART_RXFIFO_INT_S:
    if(g_tUartTran[byIdx].byRecvMode == UART_RX_MODE_INT_DYN)
        csp_uart_rto_en(ptUartBase);
    if(g_tUartTran[byIdx].ptRingBuf->hwDataLen < g_tUartTran[byIdx].ptRingBuf->hwSize)
    {
        while(csp_uart_get_sr(ptUartBase) & UART_RNE)
        {   g_tUartTran[byIdx].ptRingBuf->pbyBuf[g_tUartTran[byIdx].ptRingBuf->hwWrite] =
csp_uart_get_data(ptUartBase);
            g_tUartTran[byIdx].ptRingBuf->hwWrite = (g_tUartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUartTran[byIdx].ptRingBuf->hwSize;
            g_tUartTran[byIdx].ptRingBuf->hwDataLen ++;
        }
    }
    else
        csp_uart_rxfifo_rst(ptUartBase);
    break;
```

```
case UART_TXDONE_INT_S:
        csp_uart_clr_isr(ptUartBase,UART_TXDONE_INT_S);
        g_tUartTran[byIdx].hwTxSize --;
        g_tUartTran[byIdx].pbyTxData ++;
        if(g_tUartTran[byIdx].hwTxSize == 0)
            g_tUartTran[byIdx].bySendStat = UART_STATE_DONE;
        else
            csp_uart_set_data(ptUartBase, *g_tUartTran[byIdx].pbyTxData);
        break;
```

- **代码说明**

1) **csp_uart_rto_en(ptUartBase);**

   接收超时使能

2) **csp_uart_get_sr(ptUartBase)**

   读取 FIFO 的状态

3) **csp_uart_rxfifo_rst(ptUartBase)**

   接收 FIFO 重置

4) **csp_uart_clr_isr(ptUartBase,UART_TXDONE_INT_S);**

   清除中断标志位

5) **csp_uart_set_data(ptUartBase, *g_tUartTran[byIdx].pbyTxData);**

   发送数据

在 mian.c 中

```
int main()
{
    volatile uint8_t byRecv=0;
    uint8_t bySdData[36]={1,2,3,4,5,6,7,8,9,10,
                          11,12,13,14,15,16,17,18,19,20,
                          21,22,23,24,25,26,27,28,29,30,
                          31,32,33,34,35,36};

    uint8_t   byRxBuf[32];
    uint16_t hwRecvNum = 1;
    volatile uint16_t hwRecvLen;

    mdelay(100);
    system_init();
    board_init();
    my_printf("Hello World-~~~~\n");
    while(1)
    {
        if(hwRecvNum == 1)
        {
            hwRecvLen = csi_uart_receive(UART1,(void *)byRxBuf, hwRecvNum, 0);
            if(hwRecvLen == hwRecvNum)
                csi_uart_putc(UART1,*byRxBuf);
        }
        else if(hwRecvNum > 1)
        {
            hwRecvLen = csi_uart_receive(UART1,(void *)byRxBuf, hwRecvNum, 0);
            if(hwRecvLen == hwRecvNum)
                csi_uart_send(UART1,(void *)byRxBuf, hwRecvNum);
```

```
            }
            byRecv = byRxBuf[0];
            if(byRecv == 0x07)
                csi_uart_send_dma(UART1, DMA_CH1, (void *)bySdData, 10);
            mdelay(100);
            if(csi_dma_get_msg(DMA_CH1, ENABLE))
            {
                byRxBuf[0] =0;
            }
        }
    return 0;
}
```

- **代码说明**

初始化 hwRecvNum = 1

```
if(hwRecvNum == 1)
{
    hwRecvLen = csi_uart_receive(UART1,(void *)byRxBuf, hwRecvNum, 0);
    if(hwRecvLen == hwRecvNum)
        csi_uart_putc(UART1,*byRxBuf);
}
```

接收单个字节，打印出来。

```
byRecv = byRxBuf[0];
if(byRecv == 0x07)
    csi_uart_send_dma(UART1, DMA_CH1, (void *)bySdData, 10);
```

确认接收到的字节为 0X07，然后通过 DMA 通道打印 bySdData[36]前 10 个数。

csi_dma_get_msg(DMA_CH1, ENABLE);

获取 DMA 发送完毕的中断标志， 清除。

- **验证方法：**

  借助串口调试工具，看打印结果

3.4 DMA 接收

在 system.c 里，system_init()里初始化 uart1_config2()；实现接收用 DMA，发送轮询模式

```
void uart1_config2(void)

{

    int iRet = 0;

    csi_uart_config_t tUartConfig;                  //UART1 参数配置结构体



    csi_pin_set_mux(PB02, PB02_UART1_TX);           //TX

    csi_pin_set_mux(PA06, PA06_UART1_RX);           //RX

    csi_pin_pull_mode(PA06,GPIO_PULLUP);        //RX 管脚上拉使能, 建议配置



    tUartConfig.byParity = UART_PARITY_ODD;         //校验位，奇校验

    tUartConfig.wBaudRate = 115200;                 //波特率，115200

    tUartConfig.wInt = UART_INTSRC_NONE;         //UART 中断关闭，轮询(同步)方式

    tUartConfig.byTxMode = UART_TX_MODE_POLL;       //发送 轮询模式

    tUartConfig.byRxMode = UART_RX_MODE_POLL;       //接收 轮询模式



    csi_uart_init(UART1, &tUartConfig);             //初始化串口

    csi_uart_start(UART1, UART_FUNC_RX_TX);         //开启 UART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能



    csi_etb_init();                                         //使能 ETB 模块



    csi_uart_dma_rx_init(UART1, DMA_CH3, ETB_CH8);
```

```
    csi_uart_recv_dma(UART1, DMA_CH3, (void*)byRvUart,22);



    while(1)

    {

        if(csi_dma_get_msg(DMA_CH3, ENABLE))//获取接收完成消息，并清除消息

        {

            //添加用户代码

            nop;

            if((byRvUart[0] == 0xaa)&&(byRvUart[1] == 0x55))

                csi_uart_send(UART1,(void *)byRvUart, 3);

        }

        mdelay(10);

        nop;

    }



    return iRet;

}
```

- 代码说明

1) csi_pin_set_mux(PB02, PB02_UART1_TX);

配置 UART1 的发送口

2) csi_pin_set_mux(PA06, PA06_UART1_RX);

配置 UART1 的接收口

3) csi_pin_pull_mode(PA06,GPIO_PULLUP);

配置接收口为内部上拉

4) csi_uart_init(UART1, &tUartConfig);

根据结构体变量 tUartConfig 初始化 UART1

tUartConfig.byParity---设置校验方式

tUartConfig.wBaudRate---设置波特率

tUartConfig.wInt---设置中断方式

tUartConfig.byTxMode---设置发送方式

tUartConfig.byRxMode---设置接收方式

5) **csi_uart_start(UART1, UART_FUNC_RX_TX);**

开启 UART1 的接收和发送功能，也可单独开启其中之一

6) **csi_etb_init();**

使能 ETCB 模块

7) **csi_uart_dma_rx_init(UART1, DMA_CH3, ETB_CH8);**

配置 UART2 接收数据的 DMA 通道和 ETCB 通道

8) **csi_uart_recv_dma(UART1, DMA_CH3, (void*)byRvUart,22);**

DMA_CH3---配置 DMA 通道

(void*)byRvUart ---DMA 传送目标地址

“22”---设置接收的字节长度

9) **csi_dma_get_msg(DMA_CH3, ENABLE);**

获取 DMA 对应通道的传输完成信息，返回结果，并清除标志位

10) **csi_uart_send(UART1,(void *)byRvUart, 22);**

通过 UART1 口传输 byRvUart[]数组中前 22 个 BYTE 的数据

- **验证方法：**

借助串口调试工具，看打印结果

3.5 UART 发送数据

串口发送数据，TX 采取轮询的方式。

```
int uart_send_demo(void)
```

```
{
    int iRet = 0;
    uint8_t bySendData[30]={1,2,3,4,5,6,7,8,9,10,21,22,23,24,25,26,10,11,12,13,14,15,16,17,18,19,1,2,3};
    volatile uint8_t byRecv;
    csi_uart_config_t tUartConfig;                  //UART1 参数配置结构体

    csi_pin_set_mux(PB02, PB02_UART1_TX);           //TX
    csi_pin_set_mux(PA06, PA06_UART1_RX);           //RX
    csi_pin_pull_mode(PA06,GPIO_PULLUP);        //RX 管脚上拉使能，建议配置

    tUartConfig.byParity = UART_PARITY_ODD;         //校验位，奇校验
    tUartConfig.wBaudRate = 115200;                 //波特率，115200
    tUartConfig.wInt = UART_INTSRC_NONE;         //UART 中断关闭，轮询(同步)方式
    tUartConfig.byTxMode = UART_TX_MODE_POLL;       //发送 轮询模式
    tUartConfig.byRxMode = UART_RX_MODE_POLL;       //接收 轮询模式

    csi_uart_init(UART1, &tUartConfig);             //初始化串口
    csi_uart_start(UART1, UART_FUNC_RX_TX);         //开启 UART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能

    while(1)
    {
        /*先从串口获取一个数据*/
        byRecv = csi_uart_getc(UART1);
        if(byRecv == 0x06)    //数据为 0x06 则发送数据
            byRecv = csi_uart_send(UART1,(void *)bySendData,18);        //采用轮询方式,调用该函数时，UART 发送中断关闭

        mdelay(5);
     }
    return iRet;
```

```
}
```

- **代码说明：**

1. **csi_uart_init():** ----- 初始化 UART 功能
2. **csi_uart_start():** ----- 开启 UART 收发功能
3. **csi_uart_getc():** ----- 接收一个字符
4. **csi_uart_send():** ----- 发送数据

- **函数参数说明：**

1. **csi_uart_init(csp_uart_t *ptUartBase, csi_uart_config_t *ptUartCfg);**

   ptUartBase: UART 基地址

   ptUartCfg: UART 配置结构体指针

   ptUartCfg-> byParity: 校验

   ptUartCfg-> wBaudRate: 波特率

   ptUartCfg-> wInt: 中断源选择

   ptUartCfg-> byTxMode: 发送模式

   ptUartCfg-> byRxMode: 接收模式

2. **csi_uart_start(csp_uart_t * ptUartBase, csi_uart_func_e eFunc);**

   ptUartBase： UART 基地址

   eFunc： UART 的 RX/TX 使能，可以全部使能，也可以对单独的 RX/TX 中某一个使能

3. **csi_uart_getc(csp_uart_t *ptUartBase);**

   ptUartBase：UART 基地址

4. **csi_uart_send(csp_uart_t *ptUartBase, const void *pData, uint16_t hwSize);**

   ptUartBase: UART 基地址

   pData: 需要发送数据的首地址

hwSize: 需要发送数据的长度

- **验证方法：**

  借助串口调试工具，看打印结果

3.6 UART 接收数据

串口接收指定长度数据，RX 采取轮询的方式，带超时处理，这里会将接收的数据发送出去。

```
int uart_receive_demo(void)

{

    int iRet = 0;

    uint8_t byRecvData[20]={0};

    volatile uint8_t byRecv;

    csi_uart_config_t tUartConfig;                  //UART1 参数配置结构体


    csi_pin_set_mux(PB02, PB02_UART1_TX);           //TX

    csi_pin_set_mux(PA06, PA06_UART1_RX);           //RX

    csi_pin_pull_mode(PA06,GPIO_PULLUP);        //RX 管脚上拉使能, 建议配置


    tUartConfig.byParity = UART_PARITY_ODD;         //校验位，奇校验

    tUartConfig.wBaudRate = 115200;                 //波特率，115200

    tUartConfig.wInt = UART_INTSRC_NONE;         //串口中断关闭


    tUartConfig.byTxMode = UART_TX_MODE_POLL;       //发送 轮询模式

    tUartConfig.byRxMode = UART_RX_MODE_POLL;       //接收 轮询模式


    csi_uart_init(UART1, &tUartConfig);             //初始化串口
```

```
    csi_uart_start(UART1, UART_FUNC_RX_TX);              //开启 UART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能



    while(1)

    {

        byRecv = csi_uart_receive(UART1,byRecvData,16,2000); //UART 接收采用轮询方式(同步)

        if(byRecv   == 16)

            csi_uart_send(UART1,(void *)byRecvData,byRecv);              //UART 发送采用轮询方式(同步)

    }



    return iRet;

}
```

- **代码说明：**

  1. **csi_uart_receive():** ----- UART 接收数据

- **函数参数说明：**

  1. **csi_uart_receive(csp_uart_t *ptUartBase, void *pData, uint16_t hwSize, uint32_t wTimeOut);**

     ptUartBase: UART 基地址

     pData: 存放接收数据的地址

     hwSize: 需要接收数据的长度

     wTimeOut: 获取 UART 串口数据超时处理，轮询模式该参数才有意义，其它模式可以忽略

- **验证方法：**

  借助串口调试工具，看打印结果

3.7 UART 中断发送数据

串口发送数据，TX 采取中断的方式。

```
int uart_send_int_demo(void)
```

```
{
    int iRet = 0;
    uint8_t bySendData[30]={1,2,3,4,5,6,7,8,9,21,22,23,24,25,26,27,28,29,30,10,11,12,13,14,15,16,17,18,19};
    volatile uint8_t byRecv;
    csi_uart_config_t tUartConfig;                  //UART1 参数配置结构体

    csi_pin_set_mux(PB02, PB02_UART1_TX);           //TX
    csi_pin_set_mux(PA06, PA06_UART1_RX);           //RX
    csi_pin_pull_mode(PA06,GPIO_PULLUP);        //RX 管脚上拉使能，建议配置

    tUartConfig.byParity = UART_PARITY_ODD;         //校验位，奇校验
    tUartConfig.wBaudRate = 115200;                 //波特率，115200
    tUartConfig.wInt = UART_INTSRC_TXDONE ;     //UART 发送中断使能，采用(发送完成)TXDONE 中断
    tUartConfig.byTxMode = UART_TX_MODE_INT;   //发送模式：中断模式
    tUartConfig.byRxMode = UART_RX_MODE_POLL;       //接收模式：轮询模式

    csi_uart_init(UART1, &tUartConfig);             //初始化串口
    csi_uart_start(UART1, UART_FUNC_RX_TX);         //开启 UART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能

    while(1)
    {
        byRecv = csi_uart_getc(UART1);
        if(byRecv == 0x06)
        {
            csi_uart_send(UART1,(void *)bySendData,28);          //采用中断方式。调用该函数时，UART 发送中断使能
            while(1)
            {
                //如果有需要，可用于判断发送是否完成：
                if(csi_uart_get_msg(UART1,UART_SEND, 1))          //获取发送完成消息，并清除消息(设置为 idle)，串口发送一串数据
```

```
                {
                    //发送状态有三种，IDLE(空闲)/SEND(发送中)/DONE(发送完成)
                    //具体定义参考：uart.h 中 csi_uart_state_e,
                    //csi_uart_clr_send_status(UART1);                    //清除发送状态位，状态设置为空闲(idle)
                    nop;
                    break;
                }
            }

        }

        else
            csi_uart_send(UART1, (void *)"mismatch", 9);          //TODO
        //mdelay(10);
    }

    return iRet;
}


/*******************中断处理函数**********************/
__attribute__((weak)) void uart_irqhandler(csp_uart_t *ptUartBase,uint8_t byIdx)
{
    switch(csp_uart_get_isr(ptUartBase) & 0x080240)          //get RXFIFO/TXDONE/RXTO interrupt
    {
        case UART_RXFIFO_INT_S:                                          //rx fifo interrupt; recommended use RXFIFO interrupt

            if(g_tUartTran[byIdx].byRecvMode == UART_RX_MODE_INT_DYN)
                csp_uart_rto_en(ptUartBase);                        //enable   receive timeout
```

```
            //uint8_t byData = csp_uart_get_data(ptUartBase);
            //ringbuffer_byte_in(g_tUartTran[byIdx].ptRingBuf, byData);
            if(g_tUartTran[byIdx].ptRingBuf->hwDataLen < g_tUartTran[byIdx].ptRingBuf->hwSize)  //the same as previous line of code
            {
                while(csp_uart_get_sr(ptUartBase) & UART_RNE)
                {
                    g_tUartTran[byIdx].ptRingBuf->pbyBuf[g_tUartTran[byIdx].ptRingBuf->hwWrite] = csp_uart_get_data(ptUartBase);
                    g_tUartTran[byIdx].ptRingBuf->hwWrite = (g_tUartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUartTran[byIdx].ptRingBuf->hwSize;
                    g_tUartTran[byIdx].ptRingBuf->hwDataLen ++;
                }
            }
            else
                csp_uart_rxfifo_rst(ptUartBase);

            break;
        case UART_TXDONE_INT_S:                 //tx send complete; recommended use TXDONE interrupt

            csp_uart_clr_isr(ptUartBase,UART_TXDONE_INT_S);                         //clear interrupt status
            g_tUartTran[byIdx].hwTxSize --;
            g_tUartTran[byIdx].pbyTxData ++;

            if(g_tUartTran[byIdx].hwTxSize == 0)
                g_tUartTran[byIdx].bySendStat = UART_STATE_DONE;                    //send complete
            else
                csp_uart_set_data(ptUartBase, *g_tUartTran[byIdx].pbyTxData);  //send data

            break;
        case UART_RXTO_INT_S:
            if(g_tUartTran[byIdx].ptRingBuf->hwDataLen < g_tUartTran[byIdx].ptRingBuf->hwSize)  //the same as previous line of code
```

```
                {
                        while(csp_uart_get_sr(ptUartBase) & UART_RNE)
                        {
                                g_tUartTran[byIdx].ptRingBuf->pbyBuf[g_tUartTran[byIdx].ptRingBuf->hwWrite] = csp_uart_get_data(ptUartBase);
                                g_tUartTran[byIdx].ptRingBuf->hwWrite = (g_tUartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUartTran[byIdx].ptRingBuf->hwSize;
                                g_tUartTran[byIdx].ptRingBuf->hwDataLen ++;
                        }
                }
                else
                        csp_uart_rxfifo_rst(ptUartBase);

                g_tUartTran[byIdx].byRecvStat = UART_STATE_FULL;
                csp_uart_clr_isr(ptUartBase, UART_RXTO_INT_S);
                //csp_uart_rto_en(ptUartBase);                          //enable   receive timeout
                csp_uart_rto_dis(ptUartBase);                           //disable   receive timeout
        default:
                break;
    }
}
```

- **代码说明：**

  1. **csi_uart_get_msg():** ----- 获取 UART 接收/发送数据是否完毕

- **函数参数说明：**

  1. **csi_uart_get_msg(csp_uart_t *ptUartBase, csi_uart_wkmode_e eWkMode, bool bClrEn);**

     ptUartBase: UART 基地址

     eWkMode: UART 工作状态，接收/发送

     bClrEn: 获取到信息后是否清除接收/发送状态(设置为空闲)，使用时一般使能该选项

- **验证方法：**

  借助串口调试工具，看打印结果

3.8 UART 中断接收数据

串口接收数据，RX 采取中断的方式，这里将接收到的数据转发出去。

```
int uart_recv_int_demo(void)

{

    int iRet = 0;

    uint8_t   byRxBuf[32];

    uint16_t hwRecvNum = 1;

    volatile uint16_t hwRecvLen;



    csi_uart_config_t tUartConfig;                          //UART1 参数配置结构体



    csi_pin_set_mux(PB02, PB02_UART1_TX);                   //TX

    csi_pin_set_mux(PA06, PA06_UART1_RX);                   //RX

    csi_pin_pull_mode(PA06,GPIO_PULLUP);          //RX 管脚上拉使能，建议配置



    //接收缓存配置，实例化接收 ringbuf，将 ringbuf 接收数据缓存指向用户定义的的接收 buffer(g_byRxBuf)

    //需要传入参数：串口设备/ringbuf 结构体指针/接收 buffer/接收 buffer 长度

    csi_uart_set_buffer(UART1, &g_tRingbuf, g_byRxBuf, sizeof(g_byRxBuf));



    tUartConfig.byParity = UART_PARITY_ODD;                 //校验位，奇校验

    tUartConfig.wBaudRate = 115200;                         //波特率，115200

    tUartConfig.wInt = UART_INTSRC_RXFIFO;        //串口接收中断打开，使用 RXFIFO 中断(默认推荐使用)

    tUartConfig.byTxMode = UART_TX_MODE_POLL;       //发送模式：轮询模式

    tUartConfig.byRxMode = UART_RX_MODE_INT_FIX;//接收模式：中断指定接收模式
```

```
    csi_uart_init(UART1, &tUartConfig);                 //初始化串口
    csi_uart_start(UART1, UART_FUNC_RX_TX);             //开启 UART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能

    while(1)
    {
        //从串口缓存（UART 接收循环 buffer）里面读取数据，返回读取数据个数
        //用户应用根据实际不同协议来处理数据

        if(hwRecvNum == 1)          //单个字节收数据(读接收 ringbuf)
        {
            hwRecvLen = csi_uart_receive(UART1,(void *)byRxBuf, hwRecvNum, 0);      //读取接收循环 buffer 数据，有数据返回数据
            if(hwRecvLen == hwRecvNum)
                csi_uart_putc(UART1,*byRxBuf);
        }
        else if(hwRecvNum > 1)      //多个字节收数据(读接收 ringbuf)
        {
            hwRecvLen = csi_uart_receive(UART1,(void *)byRxBuf, hwRecvNum, 0);      //读取接收循环 buffer 数据
            if(hwRecvLen == hwRecvNum)
                csi_uart_send(UART1,(void *)byRxBuf, hwRecvNum);                    //UART 发送采用轮询方式(同步)
        }
    }

    return iRet;
}
/**********中断处理函数**********/
__attribute__((weak)) void uart_irqhandler(csp_uart_t *ptUartBase,uint8_t byIdx)
{
    switch(csp_uart_get_isr(ptUartBase) & 0x080240)         //get RXFIFO/TXDONE/RXTO interrupt
    {
```

```
case UART_RXFIFO_INT_S:                                    //rx fifo interrupt; recommended use RXFIFO interrupt


        if(g_tUartTran[byIdx].byRecvMode == UART_RX_MODE_INT_DYN)
                csp_uart_rto_en(ptUartBase);                                    //enable    receive timeout
        //uint8_t byData = csp_uart_get_data(ptUartBase);
        //ringbuffer_byte_in(g_tUartTran[byIdx].ptRingBuf, byData);
        if(g_tUartTran[byIdx].ptRingBuf->hwDataLen < g_tUartTran[byIdx].ptRingBuf->hwSize)  //the same as previous line of code
        {
                while(csp_uart_get_sr(ptUartBase) & UART_RNE)
                {
                        g_tUartTran[byIdx].ptRingBuf->pbyBuf[g_tUartTran[byIdx].ptRingBuf->hwWrite] = csp_uart_get_data(ptUartBase);
                        g_tUartTran[byIdx].ptRingBuf->hwWrite = (g_tUartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUartTran[byIdx].ptRingBuf->hwSize;
                        g_tUartTran[byIdx].ptRingBuf->hwDataLen ++;
                }
        }
        else
                csp_uart_rxfifo_rst(ptUartBase);

        break;
case UART_TXDONE_INT_S:                    //tx send complete; recommended use TXDONE interrupt


        csp_uart_clr_isr(ptUartBase,UART_TXDONE_INT_S);                                 //clear interrupt status
        g_tUartTran[byIdx].hwTxSize --;
        g_tUartTran[byIdx].pbyTxData ++;

        if(g_tUartTran[byIdx].hwTxSize == 0)
                g_tUartTran[byIdx].bySendStat = UART_STATE_DONE;                //send complete
        else
                csp_uart_set_data(ptUartBase, *g_tUartTran[byIdx].pbyTxData);  //send data
```

```
                break;
            case UART_RXTO_INT_S:
                if(g_tUartTran[byIdx].ptRingBuf->hwDataLen < g_tUartTran[byIdx].ptRingBuf->hwSize)  //the same as previous line of code
                {
                    while(csp_uart_get_sr(ptUartBase) & UART_RNE)
                    {
                        g_tUartTran[byIdx].ptRingBuf->pbyBuf[g_tUartTran[byIdx].ptRingBuf->hwWrite] = csp_uart_get_data(ptUartBase);
                        g_tUartTran[byIdx].ptRingBuf->hwWrite = (g_tUartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUartTran[byIdx].ptRingBuf->hwSize;
                        g_tUartTran[byIdx].ptRingBuf->hwDataLen ++;
                    }
                }
                else
                    csp_uart_rxfifo_rst(ptUartBase);

                g_tUartTran[byIdx].byRecvStat = UART_STATE_FULL;
                csp_uart_clr_isr(ptUartBase, UART_RXTO_INT_S);
                //csp_uart_rto_en(ptUartBase);                          //enable   receive timeout
                csp_uart_rto_dis(ptUartBase);                           //disable   receive timeout
            default:
                break;
        }
}
```

- **代码说明：**

  1. **csi_uart_set_buffer():** ----- 配置接收数据缓存(buffer)，中断接收的时候需要调用
  2. **csi_uart_putc():** ----- 发送一个字符

- **函数参数说明：**

  1. csi_uart_set_buffer(csp_uart_t *ptUartBase, ringbuffer_t *ptRingbuf, uint8_t

***pbyRdBuf, uint16_t hwLen);**

ptUartBase: UART 基地址

ptRingbuf: 循环 buf(ringbuf)结构体指针

ptRingbuf-> pbyBuf: buf 指针，指向缓存

ptRingbuf-> hwSize: 循环 buf 大小

ptRingbuf-> hwWrite: 写入数据长度

ptRingbuf-> hwRead: 读取数据长度

ptRingbuf-> hwDataLen: 数据长度

pbyRdBuf: 接收数据缓存，赋值给循环 buf 的 pbyBuf

hwLen: 接收数据长度，赋值给循环 buf 的 hwSize

2. **csi_uart_putc(csp_uart_t * ptUartBase, uint8_t byData);**

   ptUartBase: UART 基地址

   byData: 发送的字符

- **验证方法：**

  借助串口调试工具，看打印结果

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或下载进芯片
3. 通过串口调试工具，看打印输出