文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 USART 应用指南

目录

1 概述

本文介绍了在APT32F110x中USART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 USART 配置

3.1 USART 基本特性

通用同步异步收发器(USART)用来在不同单片机之间进行通讯。USART串行发送比特位数据(低位优先)，在接收端，另外一个USART则将这些数据组合成完成数据字节。串行数据传输通常使用在电脑之间的非网络通讯，以及终端和其它设备间的通讯。

- 可编程波特率发生器
- 校验位，帧检测和数据溢出错误检测
- J1587协议的Idle标志
- 支持产生传输线打断(Break)和检测
- 支持自动应答，本地回环模式，和远程回环模式
- Multi-drop模式：地址检测和产生
- 中断产生
- 5到9位的字符长度
- 可控制的数据传输起始位
- 支持智能卡协议：产生错误信号和重新发送
- 异步模式最大波特率：PCLK/16
- 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
USART_TX	USART发送数据线	O	–	–
USART_RX	USART接收数据线	I	–	–

图3.1.1管脚功能

3.2 功能框图

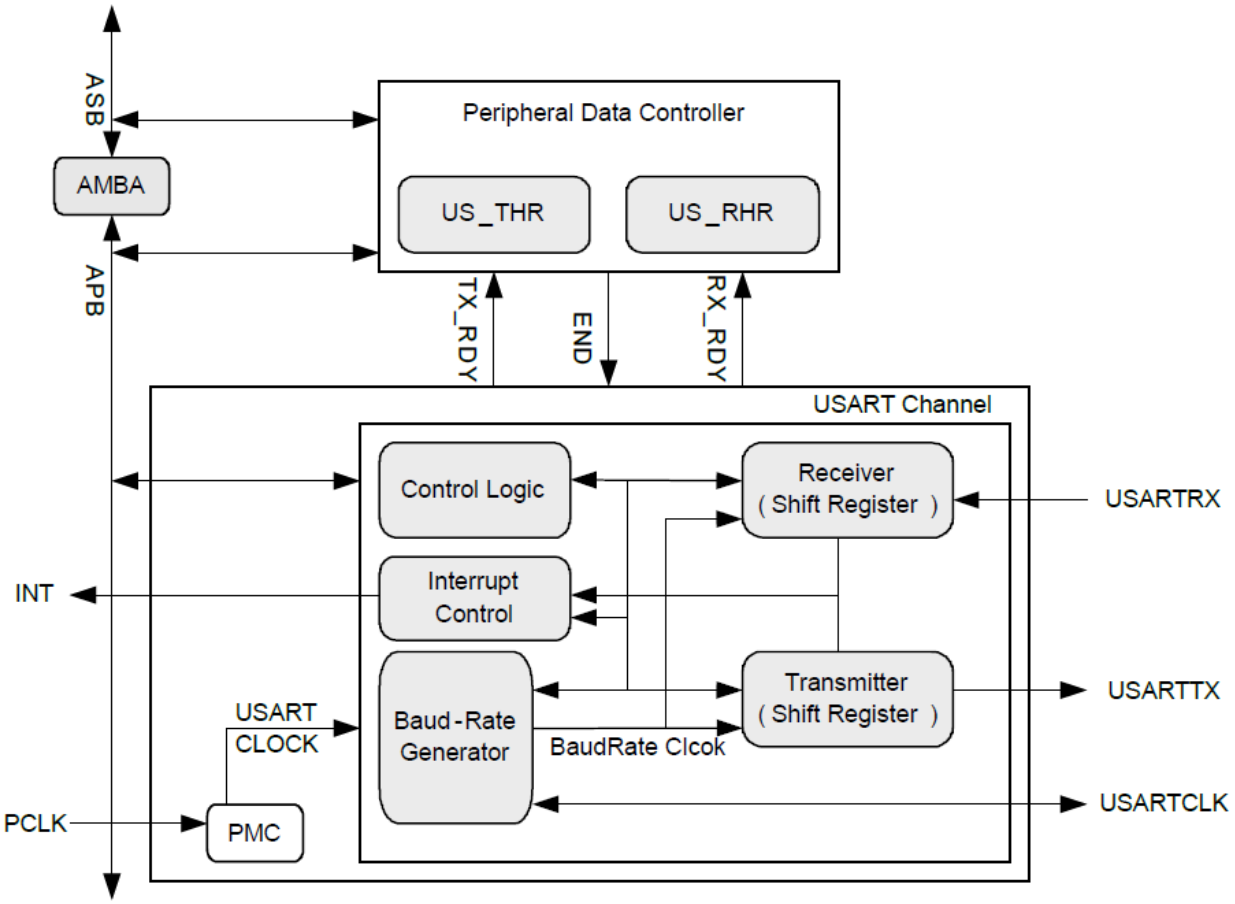

图 3.2.1USART 功能框图

3.3USART 发送数据

USART 发送一串数据，TX 采取轮询的方式。

```
int usart_send_demo(void)

{

    int iRet = 0;

    uint8_t bySdData[30]={31,25,20,34,55,6,7,8,9,10,21,22,23,24,25,26,10,11,12,13,14,15,16,17,18,19,1,2,3};
```

```
volatile uint8_t byRecv;

csi_usart_config_t tUsartCfg;                       //USART0 参数配置结构体


csi_pin_set_mux(PB02, PB02_USART0_TX);              //USART0 TX 管脚配置

csi_pin_set_mux(PA06, PA06_USART0_RX);              //USART0 RX 管脚配置

//csi_pin_set_mux(PA07, PA07_USART0_CK);            //CK，同步模式时使用

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


tUsartCfg.byClkSrc          = USART_CLKSRC_DIV1;    //clk = PCLK

tUsartCfg.byMode            = USART_MODE_ASYNC;     //异步模式

tUsartCfg.byDatabit   = USART_DATA_BITS_8;          //字节长度，8bit

tUsartCfg.byStopbit   = USART_STOP_BITS_1;          //停止位，1 个

tUsartCfg.byParity          = USART_PARITY_EVEN;    //偶校验

tUsartCfg.bClkOutEn         = DISABLE;  //禁止 USARTCLK 输出：同步模式时，USARTCLK 可以给另外设备上的 USART 提供 clk，作为同步输入时钟使用

tUsartCfg.wBaudRate         = 115200;               //波特率：115200

tUsartCfg.wInt              = USART_INTSRC_NONE;    //不使用中断

tUsartCfg.byTxMode          = USART_TX_MODE_POLL;   //发送模式：轮询/中断模式

tUsartCfg.byRxMode          = USART_RX_MODE_POLL;   //接收模式：轮询模式


csi_usart_init(USART0, &tUsartCfg);                 //初始化串口

csi_usart_start(USART0, USART_FUNC_RX_TX);          //开启 USART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能


while(1)

{

    byRecv = csi_usart_getc(USART0);

    if(byRecv == 0x06)

        byRecv = csi_usart_send(USART0,(void *)bySdData,18);    //采用轮询方式,调用该函数时，UART 发送中断关闭


    nop;
```

```
    }


    return iRet;
}
```

- **代码说明：**

  1. **csi_usart_init():** ----- 初始化 USART
  2. **csi_usart_start():** ----- 开启 USART 收发功能
  3. **csi_usart_getc():** ----- 接收一个字符
  4. **csi_usart_send():** ----- 发送数据,有两种模式（中断/轮询）

- **函数参数说明：**

  1. **csi_usart_init(csp_usart_t *ptUsartBase, csi_usart_config_t * ptUsartCfg);**

     ptUsartBase: USART 基地址

     ptUsartCfg: USART 配置结构体

     ptUsartCfg-> byClkSrc: 时钟选择(波特率发生器的输入时钟)

     ptUsartCfg-> byMode: 同步/异步模式

     ptUsartCfg-> byDatabit: 字节长度

     ptUsartCfg-> byStopbit: 停止位

     ptUsartCfg-> byParity: 校验位

     ptUsartCfg-> bClkOutEn: 禁止 USARTCLK 输出；同步模式时，USARTCLK 可以给另外设备上的 USART 提供 clk，作为同步输入时钟使用

     ptUsartCfg-> wBaudRate: 波特率

     ptUsartCfg-> wInt: 中断源选择

     ptUsartCfg-> byTxMode: 发送模式

     ptUsartCfg-> byRxMode: 接收模式

2. **csi_usart_start(csp_usart_t *ptUsartBase, csi_usart_func_e eFunc);**

   ptUsartBase: USART 基地址

   eFunc: USART 的 RX/TX 使能，可以全部使能，也可以对单独的 RX/TX 使能

3. **csi_usart_getc(csp_usart_t *ptUsartBase);**

   ptUsartBase: USART 基地址

4. **csi_usart_send(csp_usart_t *ptUsartBase, const void * pData, uint16_t hwSize);**

   ptUsartBase: USART 基地址

   pData: 需要发送数据的地址

   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

3.4 USART 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，这里将接收的数据发送出去。

```
int usart_recv_demo(void)

{

    int iRet = 0;

    volatile uint8_t byRxBuf[32] = {0};

    csi_usart_config_t tUsartCfg;                       //USART0 参数配置结构体

    volatile uint8_t byRecv;



    csi_pin_set_mux(PB02, PB02_USART0_TX);              //TX

    csi_pin_set_mux(PA06, PA06_USART0_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_usart_set_buffer(USART0, &g_tRingbuf1, g_byRxBuf1, sizeof(g_byRxBuf1));

    tUsartCfg.byClkSrc      = USART_CLKSRC_DIV1;        //clk = PCLK
    tUsartCfg.byMode        = USART_MODE_ASYNC;         //异步模式
    tUsartCfg.byDatabit     = USART_DATA_BITS_8;        //字节长度，8bit
    tUsartCfg.byStopbit     = USART_STOP_BITS_1;        //停止位，1 个
    tUsartCfg.byParity      = USART_PARITY_EVEN;        //偶校验
    tUsartCfg.bClkOutEn     = DISABLE;                  //禁止 USARTCLK 输出：同步模式时，USARTCLK 可以给另外设备上的 USART 提供 clk，
作为同步输入时钟使用
    tUsartCfg.wBaudRate     = 115200;                   //波特率：115200
    tUsartCfg.wInt          = USART_INTSRC_NONE;        //不使用中断
    tUsartCfg.byTxMode      = USART_TX_MODE_POLL;       //发送模式：轮询模式
    tUsartCfg.byRxMode      = USART_RX_MODE_POLL;       //接收模式：轮询模式

    csi_usart_init(USART0, &tUsartCfg);                 //初始化串口
    csi_usart_start(USART0, USART_FUNC_RX_TX);          //开启 USART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能

    while(1)
    {
        byRecv = csi_usart_receive(USART0, (void *)byRxBuf,16,1000);  //UART 接收采用轮询方式(同步)
        if(byRecv == 16)
            csi_usart_send(USART0,(void *)byRxBuf,byRecv);          //UART 发送采用轮询方式(同步)
    }

    return iRet;
}
```

- **代码说明：**

  1. csi_usart_receive(): ----- 获取 USART 接收到的数据

- **函数参数说明：**

  1. **csi_usart_receive(csp_usart_t *ptUsartBase, void *pData, uint16_t hwSize, uint32_t wTimeOut);**

     ptUsartBase: USART 基地址

     pData: 存放读取数据地址

     hwSize: 获取数据的长度

     wTimeOut: 获取 USART 串口数据超时处理，轮询模式有效

- **验证方法：**

  借助串口调试工具，看打印结果

3.5 USART 中断发送数据

串口发送数据，TX 采取中断的方式。

```
int usart_send_int_demo(void)
{
    int iRet = 0;
    uint8_t bySdData[30]={31,25,20,34,55,6,7,8,9,10,21,22,23,24,25,26,10,11,12,13,14,15,16,17,18,19,1,2,3};
    volatile uint8_t byRecv;
    csi_usart_config_t tUsartCfg;                       //USART0 参数配置结构体

    csi_pin_set_mux(PB02, PB02_USART0_TX);              //USART0 TX 管脚配置
    csi_pin_set_mux(PA06, PA06_USART0_RX);              //USART0 RX 管脚配置
    //csi_pin_set_mux(PA07, PA07_USART0_CK);            //CK，同步模式时使用
    csi_pin_pull_mode(PA06,GPIO_PULLUP);                //RX 管脚上拉使能，建议配置
```

```
    tUsartCfg.byClkSrc      = USART_CLKSRC_DIV1;        //clk = PCLK
    tUsartCfg.byMode        = USART_MODE_ASYNC;         //异步模式
    tUsartCfg.byDatabit     = USART_DATA_BITS_8;        //字节长度，8bit
    tUsartCfg.byStopbit     = USART_STOP_BITS_1;        //停止位，1 个
    tUsartCfg.byParity      = USART_PARITY_EVEN;        //偶校验
    tUsartCfg.bClkOutEn     = DISABLE;                  //禁止 USARTCLK 输出；同步模式时，USARTCLK 可以给另外设备上的 USART 提供 clk，
作为同步输入时钟使用
    tUsartCfg.wBaudRate     = 115200;                   //波特率：115200
    tUsartCfg.wInt          = USART_INTSRC_TXRIS;       //使用 TXFIFO 中断（默认推荐）
    tUsartCfg.byTxMode      = USART_TX_MODE_INT;        //发送模式：轮询/中断模式
    tUsartCfg.byRxMode      = USART_RX_MODE_POLL;       //接收模式：轮询模式

    csi_usart_init(USART0, &tUsartCfg);                 //初始化串口
    csi_usart_start(USART0, USART_FUNC_RX_TX);          //开启 USART 的 RX 和 TX 功能，也可单独开启 RX 或者 TX 功能

    while(1)
    {
        byRecv = csi_usart_getc(USART0);
        if(byRecv == 0x06)
        {
            csi_usart_send(USART0,(void *)bySdData,18);     //采用中断方式。调用改函数时，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_usart_get_msg(USART0, USART_SEND, ENABLE))       //获取发送完成消息，并清除状态(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,
                                nop;;
                                break;
                        }
                    }
                }
                nop;
        }

        return iRet;
}

/*****************中断处理函数*********************/
__attribute__((weak)) void usart_irqhandler(csp_usart_t *ptUsartBase,uint8_t byIdx)
{
        switch(csp_usart_get_isr(ptUsartBase) & 0x5100)                 //rxfifo/txfifo/rxtimeout interrupt
        {
                case US_RXRIS_INT:                                      //rx fifo interrupt
                        if(g_tUsartTran[byIdx].ptRingBuf->hwDataLen < g_tUsartTran[byIdx].ptRingBuf->hwSize)
                        {
                                while(csp_usart_get_sr(ptUsartBase) & US_RNE)           //Rxfifo non empty
                                {
                                        g_tUsartTran[byIdx].ptRingBuf->pbyBuf[g_tUsartTran[byIdx].ptRingBuf->hwWrite] = csp_usart_get_data(ptUsartBase);
                                        g_tUsartTran[byIdx].ptRingBuf->hwWrite = (g_tUsartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUsartTran[byIdx].ptRingBuf->hwSize;
                                        g_tUsartTran[byIdx].ptRingBuf->hwDataLen ++;
                                }
                        }
                        else
                        {
```

```
                //csp_usart_rxfifo_rst(ptUsartBase);                    // reset rxfifo
                csp_usart_cr_cmd(USART0, US_RSTRX | US_FIFO_EN | US_RXFIFO_1_2);   //reset rx
                g_tUsartTran[byIdx].ptRingBuf->hwDataLen = 0;                       //clear hwDataLen
            }
            break;
        case US_TXRIS_INT:              //tx fifo interrupt
            csp_usart_set_data(ptUsartBase, *g_tUsartTran[byIdx].pbyTxData);        //send data
            g_tUsartTran[byIdx].hwTxSize --;
            g_tUsartTran[byIdx].pbyTxData ++;

            if(g_tUsartTran[byIdx].hwTxSize == 0)
            {
                //disable usart tx interrupt
                csp_usart_int_enable(ptUsartBase, US_TXRIS_INT, DISABLE);
                g_tUsartTran[byIdx].bySendStat = USART_STATE_DONE;                  //send complete

            }
            break;
        case US_TIMEOUT_INT:                    //receive timeout interrupt

            if(g_tUsartTran[byIdx].ptRingBuf->hwDataLen < g_tUsartTran[byIdx].ptRingBuf->hwSize)
            {
                while(csp_usart_get_sr(ptUsartBase) & US_RNE)
                {
                    g_tUsartTran[byIdx].ptRingBuf->pbyBuf[g_tUsartTran[byIdx].ptRingBuf->hwWrite] = csp_usart_get_data(ptUsartBase);
                    g_tUsartTran[byIdx].ptRingBuf->hwWrite = (g_tUsartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUsartTran[byIdx].ptRingBuf->hwSize;
                    g_tUsartTran[byIdx].ptRingBuf->hwDataLen ++;
                }
            }
```

```
                else
                    csp_usart_cr_cmd(USART0, US_RSTRX | US_FIFO_EN | US_RXFIFO_1_2);   //reset rx
                    //csp_usart_rxfifo_rst(ptUsartBase);

                g_tUsartTran[byIdx].byRecvStat = USART_STATE_FULL;                      //receive complete
                csp_usart_clr_isr(USART0,US_TIMEOUT_INT);                               //clear interrupt status
                csp_usart_cr_cmd(USART0, US_STTTO | US_FIFO_EN | US_RXFIFO_1_2);        //enable receive timeover
                break;

            default:
                csp_usart_clr_isr(USART0, 0x7fff);                                      //clear all interrupt
                break;
        }
}
```

- **代码说明：**

  1. **csi_usart_get_msg()：** ----- 获取USART接收/发送完成信息，并清除/保留状态

- **函数参数说明：**

  1. **csi_usart_get_msg(csp_usart_t *ptUsartBase, csi_usart_wkmode_e eWkMode, bool bClrEn);**

     ptUsartBase: USART 基地址

     eWkMode: USART 工作模式，是处于发送还是接收模式

     bClrEn: 清除/保留接收(或发送)完成状态

- **验证方法：**

  借助串口调试工具，看打印结果

3.6 USART 中断接收数据

串口接收指定长度数据，RX 采取中断的方式，并且将接收到的数据转发出去。

```
int usart_recv_int_demo(void)

{

    int iRet = 0;

    uint8_t byRxBuf[32];

    csi_usart_config_t tUsartCfg;                    //USART0 参数配置结构体

    uint16_t hwRecvNum = 1;

    volatile uint16_t hwRecvLen;



    csi_pin_set_mux(PB02, PB02_USART0_TX);              //TX

    csi_pin_set_mux(PA06, PA06_USART0_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_usart_set_buffer(USART0, &g_tRingbuf1, g_byRxBuf1, sizeof(g_byRxBuf1));



    tUsartCfg.byClkSrc      = USART_CLKSRC_DIV1;      //clk = PCLK

    tUsartCfg.byMode        = USART_MODE_ASYNC;       //异步模式

    tUsartCfg.byDatabit   = USART_DATA_BITS_8;        //字节长度，8bit

    tUsartCfg.byStopbit   = USART_STOP_BITS_1;        //停止位，1 个

    tUsartCfg.byParity      = USART_PARITY_EVEN;      //偶校验

    tUsartCfg.bClkOutEn     = DISABLE;                //禁止 USARTCLK 输出：同步模式时，USARTCLK 可以给另外设备上的 USART 提供 clk，
作为同步输入时钟使用

    tUsartCfg.wBaudRate     = 115200;                 //波特率：115200

    tUsartCfg.wInt          = USART_INTSRC_RXRIS;     //使用 FXFIFO 中断（默认推荐）
```

```
    tUsartCfg.byTxMode          = USART_TX_MODE_POLL;               //发送模式：轮询/中断模式
    tUsartCfg.byRxMode          = USART_RX_MODE_INT_FIX;    //接收模式：中断定长接收模式

    csi_usart_init(USART0, &tUsartCfg);                             //初始化串口
    csi_usart_start(USART0, USART_FUNC_RX_TX);              //开启 USART 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_usart_receive(USART0,(void *)byRxBuf, hwRecvNum, 0);    //读取接收循环 buffer 数据，有数据返回数据
            if(hwRecvLen == hwRecvNum)
                csi_usart_putc(USART0,*byRxBuf);
        }
        else if(hwRecvNum > 1)      //多个字节收数据(读接收 ringbuf)
        {
            hwRecvLen = csi_usart_receive(USART0,(void *)byRxBuf, hwRecvNum, 0);    //读取接收循环 buffer 数据
            if(hwRecvLen == hwRecvNum)
                csi_usart_send(USART0,(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 usart_irqhandler(csp_usart_t *ptUsartBase,uint8_t byIdx)
```

```
{
    switch(csp_usart_get_isr(ptUsartBase) & 0x5100)                    //rxfifo/txfifo/rxtimeout interrupt
    {
        case US_RXRIS_INT:                                             //rx fifo interrupt
            if(g_tUsartTran[byIdx].ptRingBuf->hwDataLen < g_tUsartTran[byIdx].ptRingBuf->hwSize)
            {
                while(csp_usart_get_sr(ptUsartBase) & US_RNE)          //Rxfifo non empty
                {
                    g_tUsartTran[byIdx].ptRingBuf->pbyBuf[g_tUsartTran[byIdx].ptRingBuf->hwWrite] = csp_usart_get_data(ptUsartBase);
                    g_tUsartTran[byIdx].ptRingBuf->hwWrite = (g_tUsartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUsartTran[byIdx].ptRingBuf->hwSize;
                    g_tUsartTran[byIdx].ptRingBuf->hwDataLen ++;
                }
            }
            else
            {
                //csp_usart_rxfifo_rst(ptUsartBase);                   // reset rxfifo
                csp_usart_cr_cmd(USART0, US_RSTRX | US_FIFO_EN | US_RXFIFO_1_2);   //reset rx
                g_tUsartTran[byIdx].ptRingBuf->hwDataLen = 0;                      //clear hwDataLen
            }
            break;
        case US_TXRIS_INT:                 //tx fifo interrupt
            csp_usart_set_data(ptUsartBase, *g_tUsartTran[byIdx].pbyTxData);       //send data
            g_tUsartTran[byIdx].hwTxSize --;
            g_tUsartTran[byIdx].pbyTxData ++;

            if(g_tUsartTran[byIdx].hwTxSize == 0)
            {
                //disable usart tx interrupt
                csp_usart_int_enable(ptUsartBase, US_TXRIS_INT, DISABLE);
```

```
                g_tUsartTran[byIdx].bySendStat = USART_STATE_DONE;                  //send complete


            }
            break;
        case US_TIMEOUT_INT:                    //receive timeout interrupt

            if(g_tUsartTran[byIdx].ptRingBuf->hwDataLen < g_tUsartTran[byIdx].ptRingBuf->hwSize)
            {
                while(csp_usart_get_sr(ptUsartBase) & US_RNE)
                {
                    g_tUsartTran[byIdx].ptRingBuf->pbyBuf[g_tUsartTran[byIdx].ptRingBuf->hwWrite] = csp_usart_get_data(ptUsartBase);
                    g_tUsartTran[byIdx].ptRingBuf->hwWrite = (g_tUsartTran[byIdx].ptRingBuf->hwWrite + 1) % g_tUsartTran[byIdx].ptRingBuf->hwSize;
                    g_tUsartTran[byIdx].ptRingBuf->hwDataLen ++;
                }
            }
            else
                csp_usart_cr_cmd(USART0, US_RSTRX | US_FIFO_EN | US_RXFIFO_1_2);   //reset rx
                //csp_usart_rxfifo_rst(ptUsartBase);
            g_tUsartTran[byIdx].byRecvStat = USART_STATE_FULL;                  //receive complete
            csp_usart_clr_isr(USART0,US_TIMEOUT_INT);                           //clear interrupt status
            csp_usart_cr_cmd(USART0, US_STTTO | US_FIFO_EN | US_RXFIFO_1_2);    //enable receive timeover
            break;

        default:
            csp_usart_clr_isr(USART0, 0x7fff);                                  //clear all interrupt
            break;
    }
}
```

- 代码说明：

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

- **函数参数说明：**

1. **csi_usart_set_buffer(csp_usart_t *ptUsartBase, ringbuffer_t *ptRingbuf, uint8_t *pbyRdBuf, uint16_t hwLen) ;**

   ptUsartBase: USART 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_usart_putc(csp_uart_t * ptUsartBase, uint8_t byData);**

   ptUsartBase: USART 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