

# NV32F100x    UART 串口通信编程

## 第一章 所有库函数简介

### 库函数列表

void UART\_Init(UART\_Type \*pUART, UART\_ConfigType \*pConfig)

串口初始化

uint8\_t UART\_GetChar(UART\_Type \*pUART)

接收字符函数

void UART\_PutChar(UART\_Type \*pUART, uint8\_t u8Char)

发送字符函数

void UART\_SetBaudrate(UART\_Type \*pUART, UART\_ConfigBaudrateType \*pConfig)

波特率设置

void UART\_EnableInterrupt(UART\_Type \*pUART, UART\_InterruptType InterruptType)

开启串口中断

void UART\_DisableInterrupt(UART\_Type \*pUART, UART\_InterruptType InterruptType)

关闭串口中断

uint16\_t UART\_GetFlags(UART\_Type \*pUART)

获取 UART 所有状态标志位

uint8\_t UART\_CheckFlag(UART\_Type \*pUART, UART\_FlagType FlagType)

检查目标状态标志位

void UART\_SendWait(UART\_Type \*pUART, uint8\_t \*pSendBuff, uint32\_t u32Length)

查询方式发送字符

寄存器操作的内联函数，调用内联函数和直接操作寄存器效率一样高

uint8\_t UART\_ReadDataReg(UART\_Type \*pUART)

返回 UART 读取的数据

void UART\_WriteDataReg(UART\_Type \*pUART, uint8\_t u8Char)

发送 UART 数据

uint8\_t UART\_CharPresent(UART\_Type \*pUART)

返回接收器满标志位

void UART\_EnableTx(UART\_Type \*pUART)

打开发送使能

void UART\_DisableTx(UART\_Type \*pUART)

关闭发送使能

void UART\_EnableRx(UART\_Type \*pUART)

打开接收使能

void UART\_DisableRx(UART\_Type \*pUART)

关闭接收使能

void UART\_EnableLoopback(UART\_Type \*pUART)

选择循环模式

void UART\_EnableSingleWire(UART\_Type \*pUART)

选择单线模式

void UART\_Set8BitMode(UART\_Type \*pUART)

[选择 8bit 数据模式](#)

UART\_Set9BitMode(UART\_Type \*pUART)

[选择 9bit 数据模式](#)

UART\_EnableTxCompleteInt(UART\_Type \*pUART)

[打开发送完成的中断使能](#)

void UART\_EnableRxBuffFullInt(UART\_Type \*pUART)

[打开接收器满的中断使能](#)

UART\_DisableTxBuffEmptyInt(UART\_Type \*pUART)

[关闭发送器空的中断使能](#)

void UART\_DisableTxCompleteInt(UART\_Type \*pUART)

[关闭发送完成的中断使能](#)

void UART\_DisableRxBuffFullInt(UART\_Type \*pUART)

[关闭接收器满的中断使能](#)

void UART\_PutBreak(UART\_Type \*pUART)

[将断点字符加入发送队列](#)

uint8\_t UART\_IsTxComplete(UART\_Type \*pUART)

[返回发送完成中断位的值](#)

uint8\_t UART\_IsTxBuffEmpty(UART\_Type \*pUART)

[返回发送器空中断标志位的值](#)

uint8\_t UART\_IsRxBuffFull(UART\_Type \*pUART)

[返回接收器满中断标志位的值](#)

## UART 特性说明

- \*具有单独使能的双缓冲传送器和接收器
- \*可编程波特率（13 位模数分频器）
- \*中断驱动或轮询操作：
- \*传送数据寄存器为空且传送完成
- \*接收数据寄存器已满
- \*接收溢出、奇偶错误、成帧错误和噪声错误
- \*空闲接收器检测
- \*接收引脚上的活动边沿
- \*支持 LIN 的分割符检测
- \*硬件奇偶生成和校验
- \*可编程 8 位或 9 位字符长度
- \*可编程 1 位或 2 位停止位
- \*由闲置线路或地址标志唤醒接收器
- \*可选 13 位分隔字符生成/11 位分隔字符检测
- \*可选择传送器输出极性
- \*唤醒接收器通过空闲线或地址标记

## UART 使用说明

串口只需要配置时钟源，波特率，打开 UART 系统时钟和 UART 使能，UART 就可以工作了，最常用的 UART

8bit 数据发送，但也有需要添加奇偶校验位，添加中断发送等等，根据个人需求添加以后的库函数即可。

## 1.1 串口初始化

串口初始化步骤如下：

- 1.使能 UART 时钟
- 2.设置 UART 的复用引脚
- 3.在配置 UART 的时候，禁止 UART 的发送与接收
- 4.设置 UART 数据格式、奇偶校验方式
- 5.计算并设置 UART 波特率用于通信
- 6.使能 UART 发送和接收

函数名	UART_Init
函数原形	UART_Init(UART_Type *pUART, UART_ConfigType *pConfig)
功能描述	以配置结构体 pConfig 来初始化 UART 对象
输入参数	配置结构体 UART_ConfigType，模块结构体 UART_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置配置结构体，UART_Init(UART0, &UART_Config);

```
/**
 *
 * @基本的初始化串口操作，关中断，无硬件流控制
 *
 * @无返回
 *
 */
```

```
void UART_Init(UART_Type *pUART, UART_ConfigType *pConfig)
```

```
{
    uint16_t u16Sbr;
    uint8_t u8Temp;
    uint32_t u32SysClk = pConfig->u32SysClkHz; //定义系统时钟
    uint32_t u32Baud = pConfig->u32Baudrate; //定义波特率

    /* 检查配置第几个串口模块 */
    ASSERT((pUART == UART0) || (pUART == UART1) || (pUART == UART2));
    /* 设置时钟选通控制用来选择相应的 UART 口 */
    if (pUART == UART0)
    {
        SIM->SCGC |= SIM_SCGC_UART0_MASK; //使能相应功能位，选通对应 UART
```

```
}
#if defined (CPU_NV32)
    else if (pUART == UART1)
    {
        SIM->SCGC |= SIM_SCGC_UART1_MASK;//同上
    }
    else
    {
        SIM->SCGC |= SIM_SCGC_UART2_MASK;//同上
    }
#endif

/*确保在我们进行配置时，禁止发送和接收*/
pUART->C2 &= ~(UART_C2_TE_MASK | UART_C2_RE_MASK );
/* 配置 UART 为 8 位模式, 无奇偶校验位 */
pUART->C1 = 0;
/* 波特率计算 */
u16Sbr = (((u32SysClk)>>4) + (u32baud>>1))/u32baud;
/*把当前数据存放在串口波特率寄存器中，且 SBR 位清 0，即波特率发生器被禁止*/
u8Temp = pUART->BDH & ~(UART_BDH_SBR_MASK);
pUART->BDH = u8Temp | UART_BDH_SBR(u16Sbr >> 8);
pUART->BDL = (uint8_t)(u16Sbr & UART_BDL_SBR_MASK);
/*使能 UART 接收和发送 */
pUART->C2 |= (UART_C2_TE_MASK | UART_C2_RE_MASK );
}
```

## 1.2 接收字符函数

函数名	UART_GetChar
函数原形	UART_GetChar (UART_Type *pUART)
功能描述	获取 pUART 对象收到的数据
输入参数	pUART 对象
输出参数	无
返回值	获取 pUART 获得的数据
先决条件	已初始化 UART
函数使用实例	UART_GetChar (UART0) ;

```
/**
 *
 */
```

\* @接收一个字符

\*

\* @输入                    pUART                    一个 UART 口的基址

\*

\* @返回字符

\*

\*\*\*\*\*/

```
uint8_t UART_GetChar(UART_Type *pUART)
```

```
{
    /*断言检查串口是否合法*/
    ASSERT((pUART == UART0) || (pUART == UART1) || (pUART == UART2));
    /* 等待，直到接收满了为止*/
    while (!(pUART->S1 & UART_S1_RDRF_MASK));
    Return  pUART->D; //读数据寄存器的内容，并返回
}
```

### 1.3 发送字符函数

函数名	UART_PutChar
函数原形	UART_PutChar(UART_Type *pUART, uint8_t u8Char)
功能描述	pUART 发送数据 u8Char
输入参数	pUART 对象, 要发送的数据 u8Char
输出参数	无
返回值	无
先决条件	已初始化 UART
函数使用实例	UART_PutChar(UART0,0x5c) ;    发送的数据为 5c

\*\*\*\*\*/

\*

\* @发送一个字符.

\*

\* @输入                    pUART                    一个 UART 口的基址

\* @输入                    u8Char                    发送的字符

\*

\* @无返回

\*

\*\*\*\*\*/

```
void UART_PutChar(UART_Type *pUART, uint8_t u8Char)
```

```
{
```

```

/* 一直等待，直到缓冲区为空*/
while (!(pUART->S1 & UART_S1_TDRE_MASK));
/* 发送字符到数据寄存器 */
pUART->D = (uint8_t)u8Char;
}

```

## 1.4 波特率设置

函数名	UART_SetBaudrate
函数原形	UART_SetBaudrate (UART_Type *pUART, UART_ConfigBaudrateType *pConfig)
功能描述	设置 pUART 的波特率
输入参数	pUART 对象, 波特率配置结构体 pConfig
输出参数	无
返回值	无
先决条件	已初始化 UART
函数使用实例	UART_SetBaudrate (UART0, &Config_T);

```

/*****
*
* @波特率设置
*
* @输入      pUART      一个 UART 口的基址
* @输入      pConfig    波特率相关配置
* @无返回
*
*****/

```

```

void UART_SetBaudrate(UART_Type *pUART, UART_ConfigBaudrateType *pConfig)
{
    uint8_t u8Temp;
    uint16_t u16Sbr;
    uint32_t u32SysClk = pConfig->u32SysClkHz;
    uint32_t u32baud    = pConfig->u32Baudrate;
    /*通道的合法性检查*/
    ASSERT((pUART == UART0) || (pUART == UART1) || (pUART == UART2));
    /*计算波特率 */
    u16Sbr = (((u32SysClk)>>4) + (u32baud>>1))/u32baud;
    /* Save off the current value of the UARTx_BDH except for the SBR field */
    u8Temp = pUART->BDH & ~(UART_BDH_SBR_MASK);

```

```

pUART->BDH = u8Temp |  UART_BDH_SBR(u16Sbr >> 8);
pUART->BDL = (uint8_t)(u16Sbr & UART_BDL_SBR_MASK);

/* Enable receiver and transmitter */
pUART->C2 |= (UART_C2_TE_MASK | UART_C2_RE_MASK );
}

```

## 1.5 开启 UART 中断

函数名	UART_EnableInterrupt
函数原形	UART_EnableInterrupt (UART_Type*pUART, UART_InterruptType InterruptType)
功能描述	使能 UART 对应 InterruptType 类型的中断
输入参数	pUART 对象, 中断类型
输出参数	无
返回值	无
先决条件	已初始化 UART
函数使用实例	UART_EnableInterrupt (UART0, UART_TxCompleteInt); 开启 UART0 传输完成中断

详细配置见技术手册 7.4.6.4 UART 控制寄存器 2 和 7.4.6.7 UART 控制寄存器 3

```

/*****
*
* @开启 UART 中断
*
* @输入      pUART      UART 口的基址
* @输入      InterruptType 中断类型
*
* @无返回
*
*****/

void UART_EnableInterrupt(UART_Type *pUART, UART_InterruptType InterruptType)
{
    /* 通道合法性检查*/
    ASSERT((pUART == UART0) || (pUART == UART1) || (pUART == UART2));

    if (InterruptType == UART_TxBuffEmptyInt) //发送中断使能

```

```
{
    pUART->C2 |= UART_C2_TIE_MASK;
}
else if (InterruptType == UART_TxCompleteInt) //传输完成中断使能
{
    pUART->C2 |= UART_C2_TCIE_MASK;
}
else if (InterruptType == UART_RxBuffFullInt) //接收器中断使能
{
    pUART->C2 |= UART_C2_RIE_MASK;
}
else if (InterruptType == UART_IdleLineInt) //空闲线中断使能
{
    pUART->C2 |= UART_C2_ILIE_MASK;
}
else if (InterruptType == UART_RxOverrunInt) //过载中断使能
{
    pUART->C3 |= UART_C3_ORIE_MASK;
}
else if (InterruptType == UART_NoiseErrorInt) //噪声错误中断使能
{
    pUART->C3 |= UART_C3_NEIE_MASK;
}
else if (InterruptType == UART_FramingErrorInt) //帧错误中断使能
{
    pUART->C3 |= UART_C3_FEIE_MASK;
}
else if (InterruptType == UART_ParityErrorInt) // 奇偶校验中断使能
{
    pUART->C3 |= UART_C3_PEIE_MASK;
}
else
{
    /* un-supported Interrupt type */ //其他暂不支持类型的中断
}
```

```
}
```

## 1.6 关闭 UART 中断

函数名	UART_DisableInterrupt
函数原形	UART_DisableInterrupt (UART_Type*pUART, UART_InterruptType InterruptType)
功能描述	禁止 UART 对应 InterruptType 类型的中断
输入参数	pUART 对象, 中断类型 InterruptType
输出参数	无
返回值	无
先决条件	已初始化 UART
函数使用实例	UART_DisableInterrupt (UART0, UART_TxCompleteInt); 禁止 UART0 传输完成中断

```
/* ******************************************************************
```

```
*/
```

```
* @关闭 UART 中断
```

```
*/
```

```
* @输入      pUART      UART 端口的基址
```

```
* @输入      InterruptType 中断类型
```

```
*/
```

```
* @无返回
```

```
*/
```

```
*****
```

```
void UART_DisableInterrupt(UART_Type *pUART, UART_InterruptType InterruptType)
```

```
{
```

```
    /* 断言检测通道合法性*/
```

```
    ASSERT((pUART == UART0) || (pUART == UART1) || (pUART == UART2));
```

```
    /*以下中断类型与上面函数相同，详见技术手册*/
```

```
    if (InterruptType == UART_TxBuffEmptyInt)
```

```
    {
```

```
        pUART->C2 &= (~UART_C2_TIE_MASK);
```

```
    }
```

```
    else if (InterruptType == UART_TxCompleteInt)
```

```
    {
```

```
        pUART->C2 &= (~UART_C2_TCIE_MASK);
```

```
    }
```

```

else if (InterruptType == UART_RxBuffFullInt)
{
    pUART->C2 &= (~UART_C2_RIE_MASK);
}
else if (InterruptType == UART_IdleLineInt)
{
    pUART->C2 &= (~UART_C2_ILIE_MASK);
}
else if (InterruptType == UART_RxOverrunInt)
{
    pUART->C3 &= (~UART_C3_ORIE_MASK);
}
else if (InterruptType == UART_NoiseErrorInt)
{
    pUART->C3 &= (~UART_C3_NEIE_MASK);
}
else if (InterruptType == UART_FramingErrorInt)
{
    pUART->C3 &= (~UART_C3_FEIE_MASK);
}
else if (InterruptType == UART_ParityErrorInt)
{
    pUART->C3 &= (~UART_C3_FEIE_MASK);
}
else
{
}
}

```

## 1.7 状态标志获取

函数名	UART_GetFlags
函数原形	UART_GetFlags(UART_Type *pUART)
功能描述	获取 pUART 对应的中断标识位
输入参数	pUART 对象
输出参数	无
返回值	标志位的值

先决条件	已初始化 UART
函数使用实例	UART_GetFlags (UART0) ; 获取 UART0 的标志位的值

```

/*****
*
* @用来返回两个 UART 状态寄存器中的一些状态标志位。
*
* @输入    pUART    UART 端口的基址
*
* @返回一个 16 位的标志数
*
*****/

uint16_t UART_GetFlags(UART_Type *pUART)
{
    uint16_t u16StatusFlags = 0; //先清空标志位
    u16StatusFlags = pUART->S2; //将状态寄存器 2 的值赋给标志参数
    u16StatusFlags = (u16StatusFlags<<8) | pUART->S1; //两个状态寄存器拼接赋给标志参数
    return u16StatusFlags; //返回标志参数的值
}

```

## 1.8 检查状态标志位

函数名	UART_CheckFlag
函数原形	UART_CheckFlag(UART_Type *pUART, UART_FlagType FlagType)
功能描述	获取 pUART 对应的中断标识位
输入参数	pUART 对象
输出参数	无
返回值	标志位的值是否为 1
先决条件	已初始化 UART
函数使用实例	UART_CheckFlag (UART0, FlagType) ;

```

/*****
*
* @检查是否有标志位被置位
*
* @输入    pUART    UART 端口的基址
* @输入    FlagType 标志位类型
* @返回    1.该标志位被置位
*          0.该标志位被清除
*
*****/

```

### 1.9 查询方式发送字符

```
void UART_SendWait(UART_Type *pUART, uint8_t *pSendBuff, uint32_t u32Length)
{
    uint8_t u8TxChar;
    uint32_t i;
    for (i = 0; i < u32Length; i++)
    {
        u8TxChar = pSendBuff[i];
        while (!UART_IsTxBuffEmpty(pUART))
        {
```

```
        #if defined(ENABLE_WDOG)//喂看门狗
            WDOG_Feed();
        #endif
    }
    UART_WriteDataReg(pUART, u8TxChar);
}
}
```

## 第二章 样例程序

### 2.1 串口中断例程

```

/*****
*
* @ 简介  利用一次外部中断触发回调函数，通过按键 SW1 触发中断，使得串口打印字符串，通过串口调
*          试助手观察，本例程提供了一个串口中断服务框架。
*
*****/

#include "common.h"
#include "rtc.h"
#include "uart.h"
#include "UART_app.h"
#include "sysinit.h"
#define SEND_BUF_LEN    50
uint8_t send_buf[SEND_BUF_LEN]; //发送缓冲区
volatile uint8_t u8IsSendDone; //定义发送完成标志位
/*  */
void printf_int(int8* str)
{
    uint32 len = 0;
    u8IsSendDone = 0;
    while(*str)
    {
        send_buf[len++] = *str;
        str++;
        if (len >= (SEND_BUF_LEN-1))
        {
            send_buf[SEND_BUF_LEN-1] = 0;
            break;
        }
    }
}

```

```

    UART_SendInt(UART1, send_buf, len); //UART1 口串口发送初始化
}
/*串口数据发送完成函数*/
void UART_SendDone(void) // 发送完成时，将标志位置位
{
    u8IsSendDone = 1;
}
int main (void)
{
    UART_ConfigType sConfig;
    /*执行系统初始化*/
    sysinit();
    u8IsSendDone = 1;
    sConfig.u32SysClkHz = BUS_CLK_HZ; //配置系统时钟和波特率
    sConfig.u32Baudrate = UART_PRINT_BITRATE;
    UART_Init(UART1,&sConfig); //初始化串口 1
    UART_SetTxDoneCallback(UART1, UART_SendDone);
    UART_SetCallback(UART_HandleInt); //串口中断回调函数
    LED0_Init(); //初始化 LED 灯
    printf("\nRunning the UART_Interrupt_demo project.\n");
    /* 打开串口 1 中断 */
    NVIC_EnableIRQ(UART1_IRQn);
    printf_int("\nPrint characters using interrupt mode.\n");
    while (!u8IsSendDone);          /* 等待发送完成 */
    printf_int("\nrepeat Print characters using interrupt mode.\n");
    while (!u8IsSendDone);          /* 等待发送完成 */
    while (1);
}

```

## 2.2 串口环回例程

```

/*****
*
* @本例程为回环测试，UART1 口为正常模式完成收发，UART0 口为循环模式
*
*****/

```

```
#include "common.h"

#include "rtc.h"

#include "uart.h"

#include "uart_app.h"

#include "sysinit.h"

#define SEND_BUF_LEN      1

#define RECEIVE_BUF_LEN 1

uint8_t send_buf[SEND_BUF_LEN] = {'L'};

uint8_t receive_buf[RECEIVE_BUF_LEN] = {0};


int main (void)
{
    UART_ConfigType sConfig;
    /*执行系统初始化*/
    sysinit();
    sConfig.u32SysClkHz = BUS_CLK_HZ; //选择系统时钟
    sConfig.u32Baudrate = 115200;      //配置波特率为 115200

    LED0_Init();//初始化 LED
    printf("\nRunning the UART_Loopback_demo project.\n");
    printf("\nEnter any character to echo...\n");
    UART_WaitTxComplete(UART1);//等待串口 1 发送完成
    UART_Init(UART0,&sConfig); //初始化串口 0
    UART_EnableLoopback(UART0);//开启串口 0 环回，设定 UART0 为循环模式
    UART_SetCallback(UART_HandleInt);

    /* 禁用串口 1 收发中断 */
    UART_DisableInterrupt(TERM_PORT, UART_RxBuffFullInt);
    UART_DisableInterrupt(TERM_PORT, UART_TxBuffEmptyInt);
    /* 使能串口 1 接收溢出中断 */
    UART_EnableInterrupt(UART1, UART_RxOverrunInt);
    NVIC_EnableIRQ(UART1_IRQn);//打开串口 1 中断
    while (1)
    {
        send_buf[0] = UART_GetChar(TERM_PORT); //获取串口 1 上的字符，存放到发送缓冲区
    }
}
```

```

        UART_SendWait(UART0, send_buf, 1); //发送缓冲区字符到 UART0 口
        UART_ReceiveWait(UART0, receive_buf, 1); //把 UART0 的内容放入接收缓冲区中
        UART_PutChar(TERM_PORT, receive_buf[0]); //接收缓冲区的数据放入 UART1 口的数据寄存器
    }
}

```

按键 SW1 触发串口中断，在串口调试助手输入字符串“loopback test”完成环回测试  
该样例工程在 nv32\_pdk\build\keil\NV32\UART\_Loopback\_demo 下

## 2.3 串口查询法例程

```

/*****
*
*  串口轮回查询样例程序
*
*****/

#include "common.h"
#include "rtc.h"
#include "uart.h"
#include "sysinit.h"
#define SEND_BUF_LEN    50
uint8_t send_buf[SEND_BUF_LEN] = "\nUART send char by polling\n\r";
int main (void)
{
    UART_ConfigType sConfig;
    /* 执行系统初始化 */
    sysinit();
    sConfig.u32SysClkHz = BUS_CLK_HZ; //选择系统时钟为总线时钟
    sConfig.u32Baudrate = UART_PRINT_BITRATE; //设置波特率
    UART_Init(UART1, &sConfig); //初始化串口 1
    LED0_Init();
    printf("\nRunning the UART_Poll_demo project.\n"); //打印运行工程名
    UART_SendWait(UART1, send_buf, 50); //发送缓冲区的数据放入 UART1 口的数据寄存器中
    while (1);
}

```

