UART库

UART库是有关于单片机的串口操作库。在使用本库之前先到ecbm_core.h里使能。双击打开ecbm_core.h文件，然后进入图形化配置界面，使能UART库。虽然为了自动下载功能，UART库是默认使能的，但也最好确认一遍。

所谓的【自动下载功能】是指ECBM库会在串口接收中断处，做一个重复下载流的判断。当接收到233个重复的下载流数据之后，就会重启单片机已达到免冷启动的效果。

API

uart_init

函数原型：void uart_init(void);

描述

串口初始化函数。

输入

无

输出

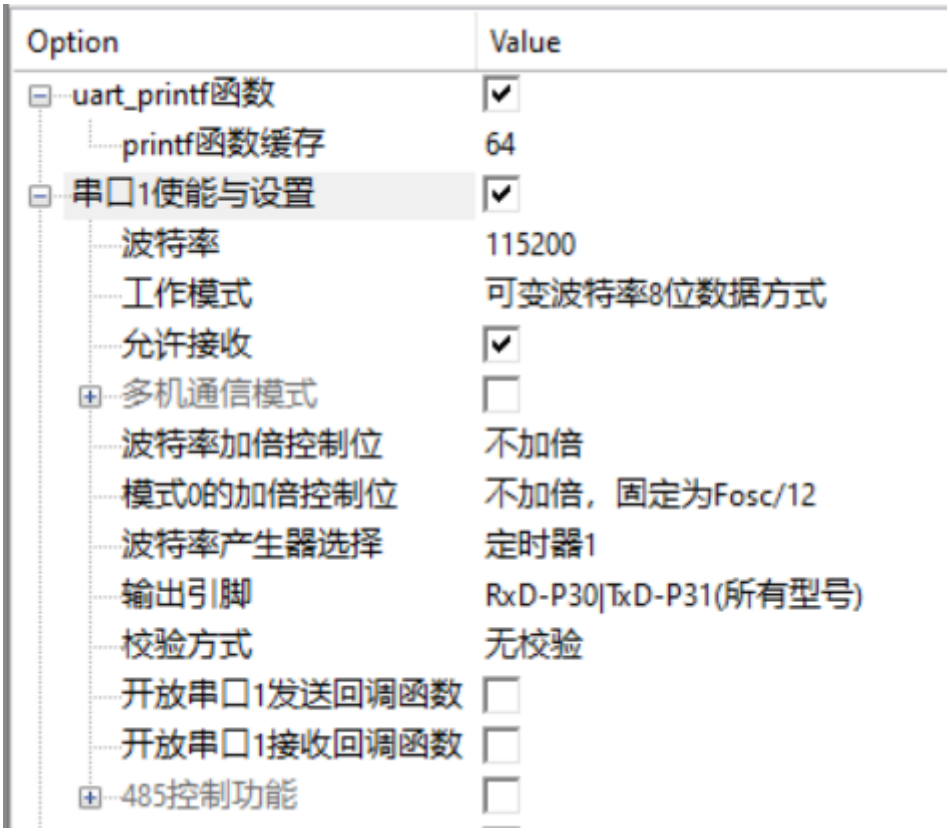
无

返回值

无

参数配置

本函数初始化的参数都是由图形化配置界面来设置，双击打开uart.h文件，进入图形化配置界面。



设置说明如下：

串口1

- **串口1使能与设置**：勾选了这个之后，串口1相关的代码才会编译。
- **波特率**：就是常说的波特率，这里列出来的值都是标准的波特率。
- **工作模式**：串口虽然有同步和异步之分，但90%的应用都是异步的。选择【可变波特率8位数据方式】或者【可变波特率9位数据方式】就行，其中不用校验就用8位，用校验就用9位。
- **允许接收**：需要接收数据就勾选吧，一般这都是要的。
- **多机通信模式**：通过软件协议也能实现多机通信，而且多机通信基本也都要有协议。所以这个功能算鸡肋了。
- **波特率加倍控制位**：建议保持默认。
- **模式0的加倍控制位**：建议保持默认。
- **波特率产生器选择**：可以在【定时器1】和【定时器2】中选一个。
- **输出引脚**：根据需要在选项里选择吧。注意括号里的提示，有些型号不能映射到某些IO口的。
- **校验方式**：无校验、奇校验、偶校验、0校验、1校验可选。
- **开放串口1发送回调函数**：使能之后必须定义发送回调函数，否则单片机会跑飞。
- **开放串口1接收回调函数**：使能之后必须定义接收回调函数，否则单片机会跑飞。
- **485控制功能**：使能之后，该串口的所有发送函数都会追加一个或两个控制脚去控制485芯片的收发方向。

串口2

- **串口2使能与设置**：勾选了这个之后，串口2相关的代码才会编译。
- **波特率**：就是常说的波特率，这里列出来的值都是标准的波特率。
- **通信位数**：异步模式。不用校验就选择【8位数据】，用校验就用【9位数据】。
- **允许接收**：需要接收数据就勾选吧，一般这都是要的。
- **多机通信模式**：通过软件协议也能实现多机通信，而且多机通信基本也都要有协议。所以这个功能算鸡肋了。
- **输出引脚**：根据需要在选项里选择吧。注意括号里的提示，有些型号不能映射到某些IO口的。
- **校验方式**：无校验、奇校验、偶校验、0校验、1校验可选。
- **开放串口2发送回调函数**：使能之后必须定义发送回调函数，否则单片机会跑飞。
- **开放串口2接收回调函数**：使能之后必须定义接收回调函数，否则单片机会跑飞。

- **485控制功能**：使能之后，该串口的所有发送函数都会追加一个或两个控制脚去控制485芯片的收发方向。

串口3

- **串口3使能与设置**：勾选了这个之后，串口3相关的代码才会编译。
- **波特率**：就是常说的波特率，这里列出来的值都是标准的波特率。
- **通信位数**：异步模式。不用校验就选择【8位数据】，用校验就用【9位数据】。
- **波特率产生器选择**：可以在【定时器2】和【定时器3】中选一个。
- **允许接收**：需要接收数据就勾选吧，一般这都是要的。
- **多机通信模式**：通过软件协议也能实现多机通信，而且多机通信基本也都要有协议。所以这个功能算鸡肋了。
- **输出引脚**：根据需要在选项里选择吧。注意括号里的提示，有些型号不能映射到某些IO口的。
- **校验方式**：无校验、奇校验、偶校验、0校验、1校验可选。
- **开放串口3发送回调函数**：使能之后必须定义发送回调函数，否则单片机会跑飞。
- **开放串口3接收回调函数**：使能之后必须定义接收回调函数，否则单片机会跑飞。
- **485控制功能**：使能之后，该串口的所有发送函数都会追加一个或两个控制脚去控制485芯片的收发方向。

串口4

- **串口4使能与设置**：勾选了这个之后，串口4相关的代码才会编译。
- **波特率**：就是常说的波特率，这里列出来的值都是标准的波特率。
- **通信位数**：异步模式。不用校验就选择【8位数据】，用校验就用【9位数据】。
- **波特率产生器选择**：可以在【定时器2】和【定时器4】中选一个。
- **允许接收**：需要接收数据就勾选吧，一般这都是要的。
- **多机通信模式**：通过软件协议也能实现多机通信，而且多机通信基本也都要有协议。所以这个功能算鸡肋了。
- **输出引脚**：根据需要在选项里选择吧。注意括号里的提示，有些型号不能映射到某些IO口的。
- **校验方式**：无校验、奇校验、偶校验、0校验、1校验可选。
- **开放串口4发送回调函数**：使能之后必须定义发送回调函数，否则单片机会跑飞。
- **开放串口4接收回调函数**：使能之后必须定义接收回调函数，否则单片机会跑飞。
- **485控制功能**：使能之后，该串口的所有发送函数都会追加一个或两个控制脚去控制485芯片的收发方向。

小科普

- **奇校验**：当要发送的8位数据里有奇数个1时，校验位为0；有偶数个1时，校验位为1。总体的1的数量为奇数。
- **偶校验**：当要发送的8位数据里有奇数个1时，校验位为1；有偶数个1时，校验位为0。总体的1的数量为偶数。
- **0校验**：检验位恒定为0。
- **1校验**：校验位恒定为1。

调用例程

打开【自动下载功能】的情况下：

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//uart_init函数已经在这里面执行了。
    while(1){
    }
}
```

不打开【自动下载功能】的情况下：

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    uart_init();//初始化串口。
     while(1){
    }
}
```

注意事项

1. 如果在ecbm_core.h设置了自动下载功能的话，uart_init函数是会自动在system_init函数执行的时候被调用的。不需要自己再执行一遍。
2. uart_init执行的时候，将会设置波特率、映射IO口等信息。
3. 串口2只能是定时器2产生波特率，不能更改。
4. 定时器2可以给两个甚至4个串口提供波特率。条件是它们的**波特率必须一样。**

uart_set_io

函数原型：void uart_set_io(u8 id,u8 io);

描述

串口输出IO设置函数，可以将串口映射到别的IO口上。

输入

- id：串口的编号，按通用说法从1开始。
- io：要切换的目标IO。

输出

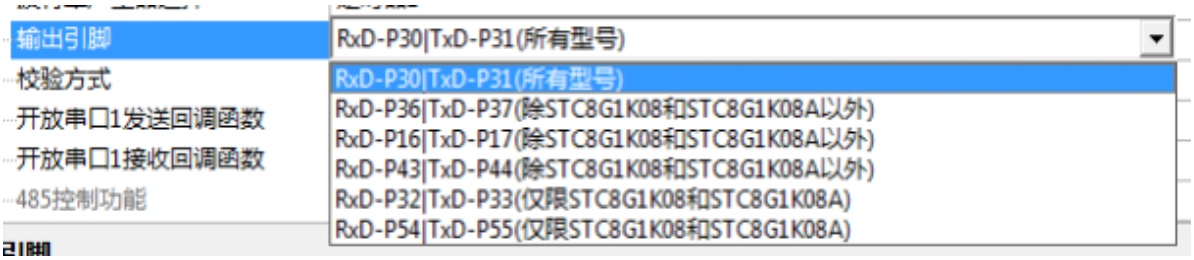
无

返回值

无

参数配置

在uart.h文件中的图形化配置界面里，打开所需的【串口n的使能与设置】，接下来在【输出引脚】处选择串口要映射的IO口。需要注意单片机的型号是否有这个IO口。



在图形化配置界面设置了之后，就算没有调动本函数，也会在uart_init函数里执行生效。

调用例程

由于STC8系列型号众多，封装脚数又会影响到IO的复用情况，所以**如果不是有串口反复切换的需求，还是推荐用图形化配置来设置IO映射。**图形化配置界面不仅有详细的型号说明，还不需要去背那一大串宏定义。

如果确实有切换IO的需求，那么请参考以下的宏定义：

- UART1_PIN_P30_P31：串口1映射到P3.0和P3.1，适合所有型号。
- UART1_PIN_P36_P36：串口1映射到P3.6和P3.7，适合除STC8G1K08和STC8G1K08A以外的型号。
- UART1_PIN_P16_P17：串口1映射到P1.6和P1.7，适合除STC8G1K08和STC8G1K08A以外的型号。
- UART1_PIN_P43_P44：串口1映射到P4.3和P4.4，适合除STC8G1K08和STC8G1K08A以外的型号。
- UART1_PIN_P32_P33：串口1映射到P3.2和P3.3，适合STC8G1K08和STC8G1K08A这两个型号。
- UART1_PIN_P54_P55：串口1映射到P5.4和P5.5，适合STC8G1K08和STC8G1K08A这两个型号。
- UART2_PIN_P10_P11：串口2映射到P1.0和P1.1，适合所有型号。
- UART2_PIN_P40_P42：串口2映射到P4.0和P4.2，适合STC8F和STC8A这两个系列。
- UART2_PIN_P46_P47：串口2映射到P4.6和P4.7，适合STC8G和STC8H这两个系列。
- UART3_PIN_P00_P01：串口3映射到P0.0和P0.1，适合所有型号。
- UART3_PIN_P50_P51：串口3映射到P5.0和P5.1，适合所有型号。
- UART4_PIN_P02_P03：串口1映射到P0.2和P0.3，适合所有型号。
- UART4_PIN_P52_P53：串口1映射到P5.2和P5.3，适合所有型号。

```
uart_set_io(1,UART1_PIN_P30_P31);//串口1映射到P3.0脚和P3.1脚。
uart_string(1,"串口1发送测试-3031");//发送一个字符串。
uart_set_io(1,UART1_PIN_P16_P17);//串口1映射到P1.6脚和P1.7脚。
uart_string(1,"串口1发送测试-1617");//发送一个字符串。
```

注意事项

1. 宏定义没有标识型号，一定要先看单片机的引脚图确认该型号能映射该IO口。简单的例子：对于STC8G1K08A-8PIN这个型号来说，是没有P4口的，那么UART1_PIN_P43_P44这个宏定义在这个型号是无效的。
2. 不是每一个型号都有4个串口，在使用的时候也要留意。
3. STC的命名规则里，**RXD2和TXD2代表的是串口2，RXD_2和TXD_2代表的是串口1的第2组映射口。**一定得留意！

uart_set_baud

函数原型：void uart_set_baud(u8 id,u32 baud);

描述

串口波特率设置函数。

输入

- id：串口的编号，按通用说法从1开始。
- baud：要设置的波特率。

输出

无

返回值

无

调用例程

```
if(strcmp(buf,"AT9600")){//比较接收到的字符串里有没有“AT9600”的字样。
    uart_set_baud(1,9600);//有的话就设置波特率为9600。
}else if(strcmp(buf,"AT115200")){//比较接收到的字符串里有没有“AT115200”的字样。
    uart_set_baud(1,115200);//有的话就设置波特率为115200。
}
```

注意事项

1. 本函数执行之后，设置的波特率会立即生效。因此尽量在通信结束后再执行，否则会导致后续的数据错误。
2. 虽然参数是可以随便输入任意波特率，但还是推荐那些常用的9600或者115200。否则计算出来的波特率和设置的波特率可能会相差很大。

uart_char

函数原型：void uart_char(u8 id,u8 ch);

描述

串口单个字节发送函数。

输入

- id：串口的编号，按通用说法从1开始。
- ch：要发送的字符或者数据。

输出

无

返回值

无

调用例程

矩阵按键按下后发送按键键值：

```
key_deal();//矩阵键盘处理函数。
if(key_1==0){//判断按键1的标志位，
    uart_char(1,'1');//按下时向串口1发送字符‘1’。
}
if(key_2==0){//判断按键2的标志位，
    uart_char(1,'2');//按下时向串口1发送字符‘2’。
}
//剩余14个按键处理代码省略。
```

注意事项

1. 本函数兼顾了奇校验、偶校验、0校验和1校验这4种通用校验。在设置了校验模式之后，不需要修改本函数，函数内部会自动加上校验。

2. 由于永久内置校验功能会导致效率降低，所以各个检验功能都是由宏定义开关来决定是否编译。因此**没有办法在单片机运行的时候更换校验模式**，校验模式在编译的时候就已经固定下来了。
3. 每个串口的校验模式是独立的。

uart_char_9

函数原型：void uart_char_9(u8 id,u8 ch,u8 bit9);

描述

串口单个字节发送函数，发送9位数据。

输入

- id：串口的编号，按通用说法从1开始。
- ch：要发送的字符或者数据。
- bit9：要发送的第9位数据。

输出

无

返回值

无

调用例程

自定义协议之“0代表地址1代表数据”：

```
uart_char_9(1,14,0);//发送给ID为14的从机。(第9位是0，按自定义协议是地址。)
uart_char_9(1,0xA5,1);//帧头。（第9位是1，按自定义协议是数据。）
uart_char_9(1,0x05,1);//数据1。
uart_char_9(1,0x12,1);//数据2。
uart_char_9(1,0x17,1);//校验和，一般就是数据1+数据2。
uart_char_9(1,0x5A,1);//帧尾。
```

注意事项

1. 本函数和uart_char函数的区别就在于检验位是开放的，可以由用户自定义一种检验位。也可以像例程那样当做地址/数据区分位，反正就是十分自由。
2. 如果说uart_char函数的检验模式不能在运行时候修改很遗憾的话，可以用本函数来实现可变校验通信。

uart_string

函数原型：void uart_string(u8 id,u8 * str);

描述

串口字符串发送函数。

输入

- id：串口的编号，按通用说法从1开始。
- str：要发送的字符或者数据。

输出

无

返回值

无

调用例程

经典的hello world：

```
uart_string(1,"Hello World\r\n");//发送字符串Hello World并回车。
```

注意事项

1. 本函数基于uart_char函数，所以校验模式也是支持奇、偶、0、1这4种校验。

uart_printf

函数原型：void uart_printf(u8 id,u8 * str,...);

描述

串口打印函数。

输入

- id：串口的编号，按通用说法从1开始。
- str：要发送的格式化内容。
- ...：格式化的参数。

输出

无

返回值

无

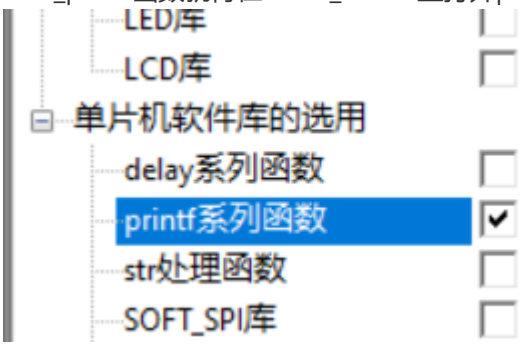
调用例程

调试运行次数：

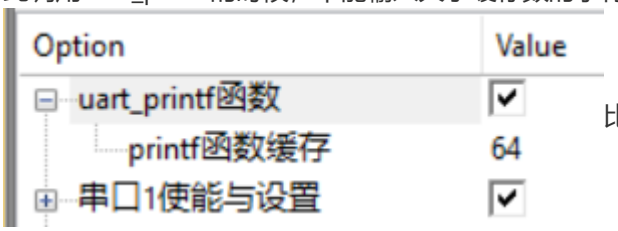
```
while(1){//主循环里
    if(in_io){//判断输入标志位，如果是有信号来的话，
        in_io=0;//先清除标志位。
        count++;//统计信号数量。
    }
    if(time>=1000){//判断时间变量，如果等于1000，说明到1秒了。
        time=0;//清零变量，从0开始计时。
        uart_printf(1,"一秒内收到%u个信号\r\n",count);//打印count变量的值。
        count=0;//清零，为下一次的统计做准备。
    }
}
```

注意事项

1. 本函数基于uart_string函数，所以校验模式也是支持奇、偶、0、1这4种校验。
2. 本函数是使用stdio库实现的格式化转换，一些高手会自己写printf来节约flash空间。所以本函数是可以通过关闭使能来优化掉的。
3. 占位符%d和%u在C51的printf函数里只能用于u16型变量的显示，uart_printf函数也是基于printf系列的，所以也有这个问题。解决办法有二，一是将占位符换成%bd和%bu，二是把u8的变量强转成u16型。
4. ECBM库所有的串口发送函数都是中断发送，因此**不要在其他中断函数中调用uart_printf函数**，否则会引起中断嵌套直至跑飞。
5. 由于除了uart_printf外还有其他库会用到sprintf，为了优化空间做了个总使能，因此如果需要使用uart_printf函数就得在ecbm_core.h里打开printf使能。



6. uart_printf的原理就是建立一个字符串缓存，然后调用sprintf函数来分析字符串里面的占位符。因此调用uart_printf的时候，不能输入大于缓存数的字符串。

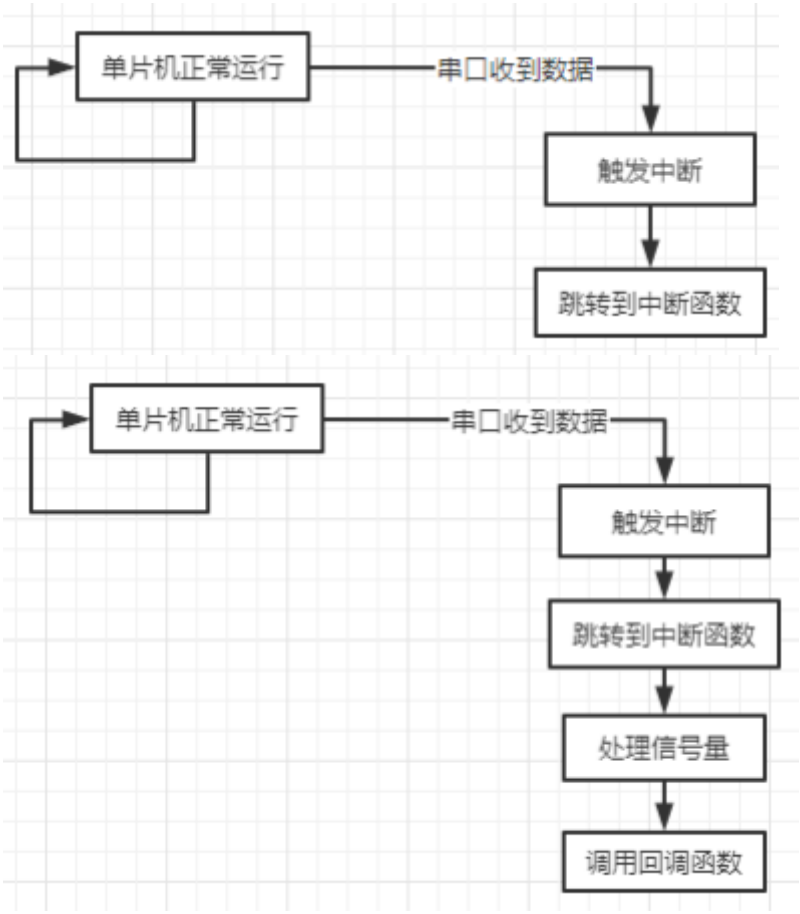


比如这里设置的是64字节，那么就不要输入大于64字节的字符串了，否则会出现意想不到的问题。如果确实有这方面的需求，ram大小足够的情况下可以增加缓存数；ram大小不够的时候可以将一段字符串拆分成几段用多个uart_printf函数去发送。

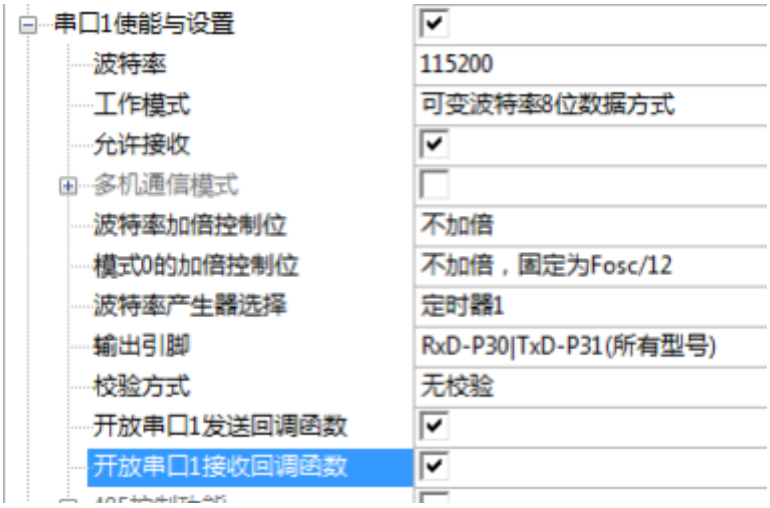
回调函数

串口的发送可以通过调用发送函数，那么接收主要是靠回调函数。“回调函数”没有那么神秘，一般而言正常的函数都是设计者定义好函数内容，由使用者决定在何处调用。回调函数是设计者先决定好在何处调用，然后由使用者来决定函数的内容。

串口的使用有查询法和中断法之分，回调函数是中断法的应用。与常规中断法的流程区别是：



在使用回调函数前，根据需要使能发送回调或者接收回调。



然后在某个.c文件里定义回调函数，为了快速和安全的跳转，本库没有把回调函数用函数指针传入中断处理函数，而是直接调用了指定名字的函数。这意味着回调函数的名字不能随便起，只能是以下这些名字：

- uart1_receive_callback：串口1接收回调函数。
- uart1_send_callback：串口1发送回调函数。
- uart2_receive_callback：串口2接收回调函数。
- uart2_send_callback：串口2发送回调函数。
- uart3_receive_callback：串口3接收回调函数。
- uart3_send_callback：串口3发送回调函数。
- uart4_receive_callback：串口4接收回调函数。
- uart4_send_callback：串口4发送回调函数。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数。
    while(1){
    }
}
void uart1_receive_callback(void){//接收回调函数
    if(SBUF=='1'){//收到字符‘1’的时候，
        LED_ON;//点亮LED。
    }else if(SBUF=='0'){//收到字符‘0’的时候，
        LED_OFF;//熄灭LED。
    }
}
```

注意事项

1. 每发送一个字节的数据或者接收一个字节的数据，就会执行一次对应的回调函数。
2. 回调函数在串口中断中调用，因此不要在回调函数内执行任何串口发送函数，否则中断嵌套会导致程序崩溃。
3. 最好在main.c里定义回调函数。

优化建议

1. 没有用到的串口不使能。
2. 不使能uart_printf函数，可以自己实现一个更小巧的打印函数。
3. 不是必要的话，就不开校验位。