Technical Article

基于 CC2340R5 蓝牙协议实现 IO 模拟 UART

Gao Wei, Albin Zhang, Wesley He, Leon Yan

1. 摘要:

CC2340R5 --- TI 第四代低功耗蓝牙 SOC，同时支持 ZigBee 和私有 2.4GHz 协议无线功能。芯片由主处理器 M0+、软件可编程射频处理器、电源管理和丰富的外设构成片上系统，最高达 512kB 的 Flash、36kB SRAM 以及 26 GPIOs 为用户提供了更灵活的系统设计能力，具有集成度高、宽温度范围、低成本、低功耗、优秀的射频性能等特点。

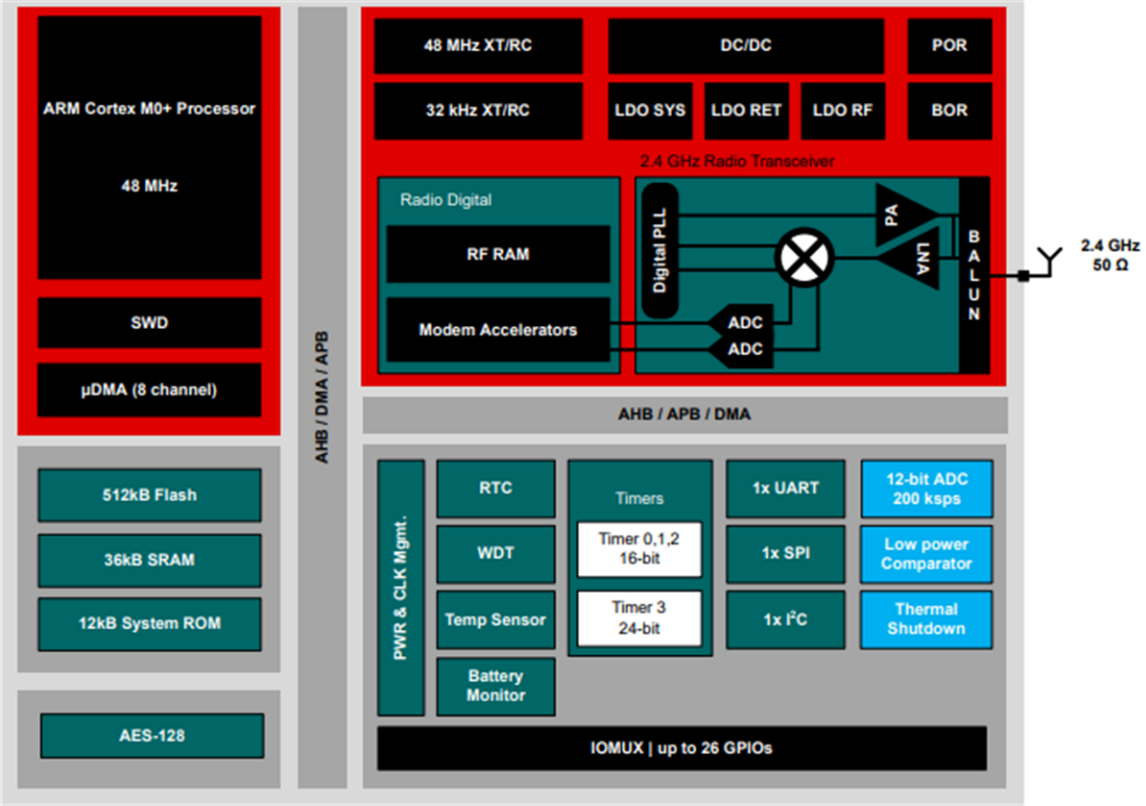

图 1. CC2340 芯片框图

本文旨在提供一种使用 IO 口模拟 UART 打印功能的方案，在 1 个硬件 UART 的基础上扩展一个额外的 UART 供用户打印和传输数据。本文基于 CC2340R5 EVM 板，根据 TI 开源的 SDK 中 basic_ble 蓝牙协议栈例程进行二次开发验证，为用户使用 IO 扩展 UART 功能提供参考。

2. CC2340 UART 特性:

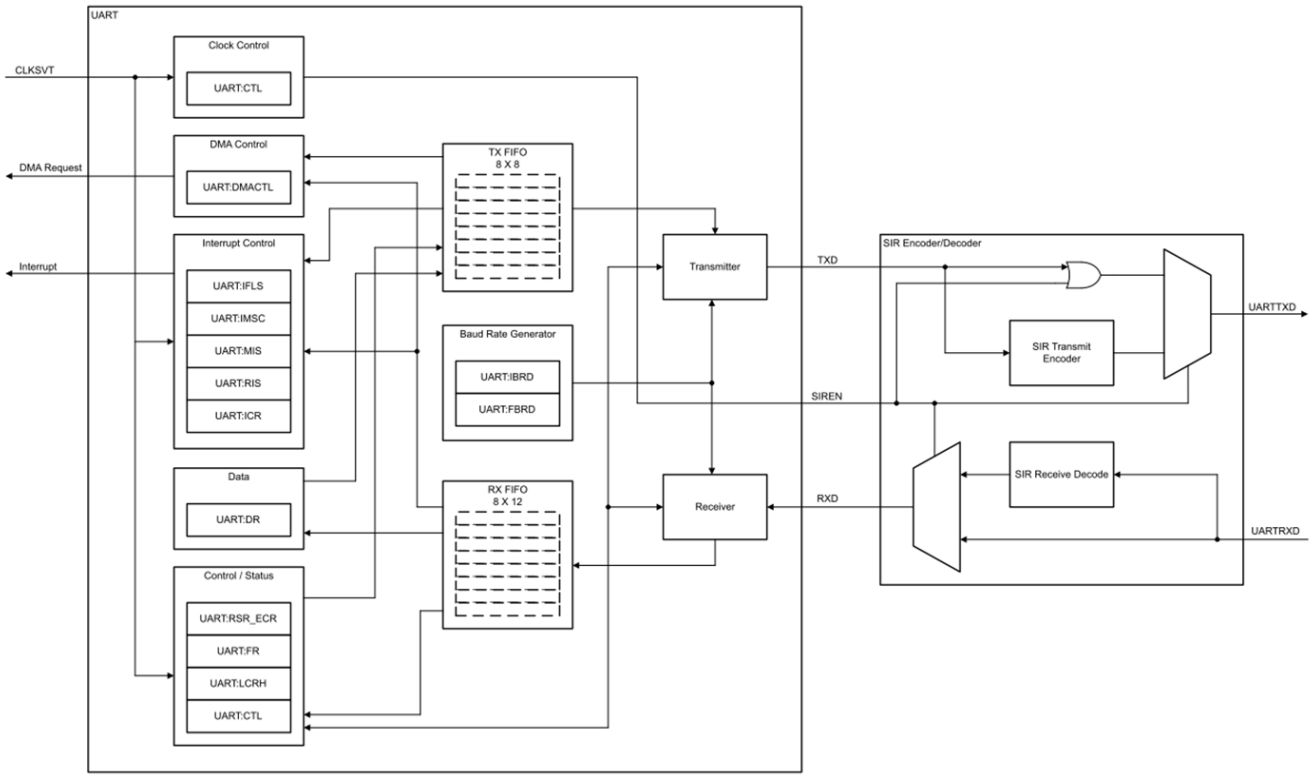

图 2. CC2340 UART Block Diagram

CC2340 UART 最高支持 3Mbsp 波特率，支持 8 X 8 深度的 TX FIFO 和 8 X 12 深度的 RX FIFO，可以配置 FIFO 深度触发中断，自带错误状态检测以及中断触发，支持 DMA 配合传输。

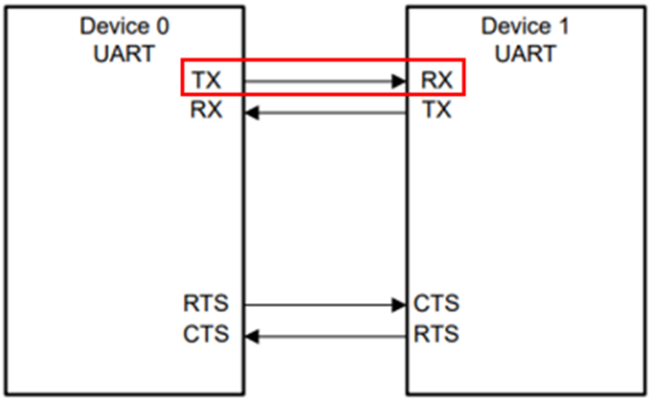

图 3. CC2340 Boot UART Connection

UART 只需要 TX 和 RX 两跟线就能实现一对一的数据通信，是一种异步全双工传输方式。本文通过一个 GPIO 模拟 TX 接口进行数据的打印。

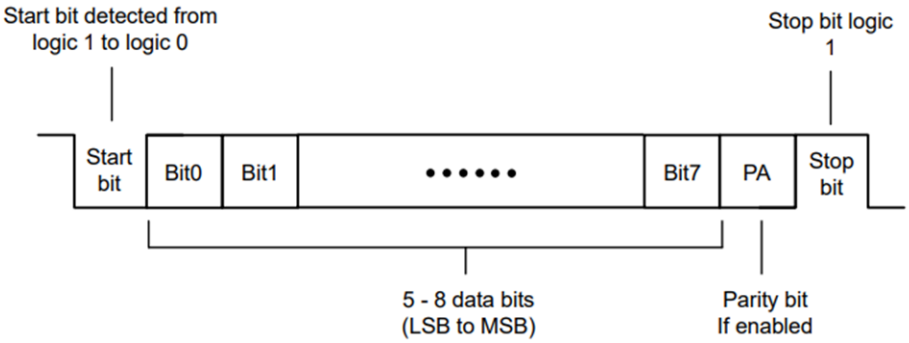

图 4. CC2340 UART Data Format

3. 基于 Basic_ble 例程扩展 IO 模拟 UART:

新建任务线程 :

CC2340 蓝牙协议例程是基于 FreeRTOS 开发的，所以首先需要创建一个新任务来作为 IO 模拟 UART 的载体，如下图所示创建一个优先级为 1 的任务，在 FreeRTOS 系统中，优先级数字越大优先级越高。此处需要注意的是，由于蓝牙协议栈任务线程优先级原始配置为 5，所以新建任务初始化优先级需小于 5，以此保证蓝牙协议栈的最高优先级运行。

```
/* Initialize the attributes structure with default values */
pthread_attr_init(&attrs);

/* Set priority, detach state, and stack size attributes */
priParam.sched_priority = 1;
retc                    = pthread_attr_setschedparam(&attrs, &priParam);
retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
retc |= pthread_attr_setstacksize(&attrs, THREADSTACKSIZE);
if (retc != 0)
{
  /* failed to set attributes */
  while (1) {}
}

retc = pthread_create(&thread, &attrs, MyIOThread, NULL);
if (retc != 0)
{
  /* pthread_create() failed */
  while (1) {}
}
```

图 5. 新建任务线程

Timer 启用:

想要依靠 IO 口精准的打印出数据，必须精准控制 IO 口电平的转换时序，这就需要依靠定时器来产生计时事件按 UART 通信标准进行 IO 口电平的翻转。

在 Timer 的选择上，虽然 FreeRTOS 有软件定时器的机制，但最小定时周期为 1ms 无法满足正常波特率的需求，故只能选择 CC2340 集成的 LGPTimer 硬件定时器。

Device type	LGPT0	LGPT1	LGPT2	LGPT3
CC2340R2	16 bits	16 bits	N.A	N.A
CC2340R5	16 bits	16 bits	16 bits	24 bits
CC27X5	16 bits	16 bits	16 bits	24 bits

图 6. CC2340 LGPTimer 资源

Timer 工作在周期中断模式，周期性产生中断并在中断服务函数中根据数据转换 IO 电平，Timer 周期就决定了波特率(例如波特率 9600 对应 Timer 周期为 104us)，如下图所示为波特率 9600 时 IO 打印 0x55 的电平状态：

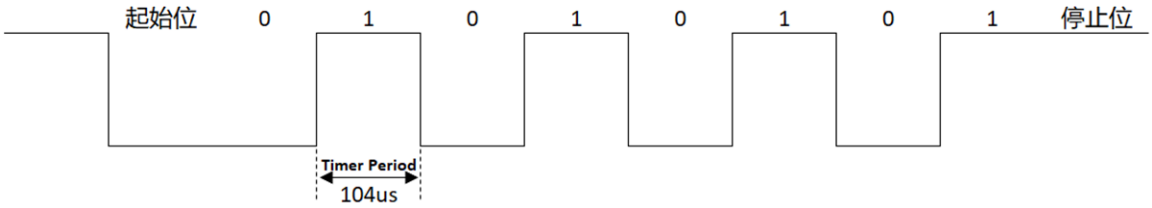

图 7. IO 电平转换时序

前后台工作模式:

整个功能的实现是以前后台配合的模式完成，前台为创建的任务线程，而后台则是 LGPTimer 定时器中断。前台任务主要实现数据的处理，包括对单个数据、数组进行拆分，为定时器中断做好数据打印准备，在定时器中断中尽可能少的处理事务，以减轻整个功能实现的 CPU 负荷。而定时器中断作为后台会根据波特率周期性的转换电平状态，实时获取前台任务的数据域，按照 UART 通信标准完成每一帧数据包起始位、数据位、停止位的电平输出。

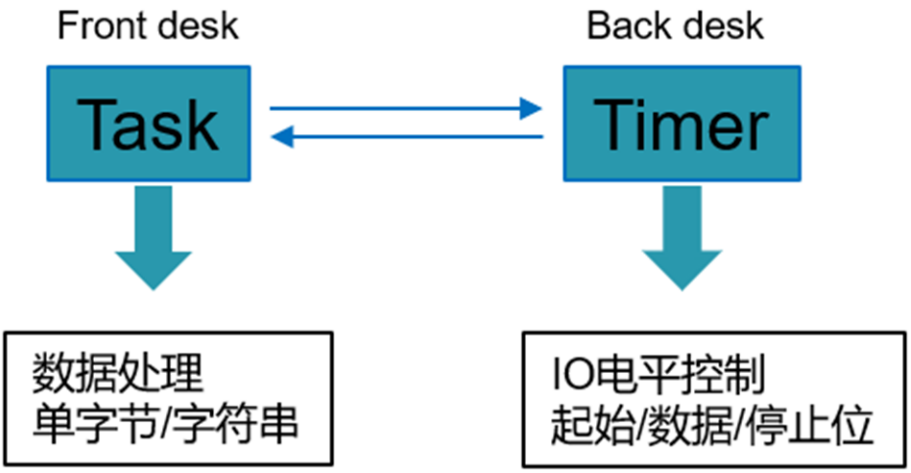

图 8. 前后台控制方式

4. 任务与 Timer 运行流程:

动态任务切换:

前面提到新建任务的初始优先级设置低于蓝牙协议栈优先级，由于优先级高的任务在会优先被执行，所以在系统需要使用 IO 口传输数据时就需要将该打印任务的优先级提高到大于蓝牙优先级。在本次数据打印完成后需要将打印任务恢复为低优先级，这样可以保证打印数据的准确性也可以最大程度保证蓝牙协议栈运行的稳定性。

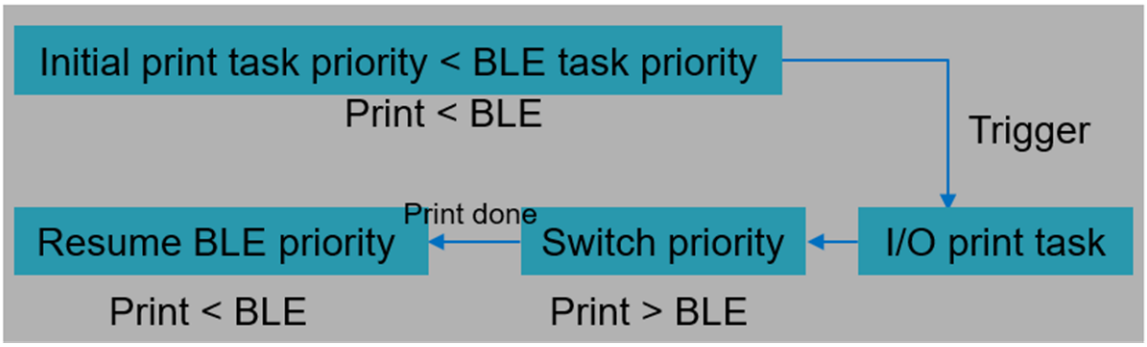

图 9. 任务切换流程

定时器挂起:

同样的，定时器开启后进出中断会占用 CPU，所以定时器需要在打印请求建立之后再开启并在打印完成后关闭，最大程度减轻 CPU 负荷。

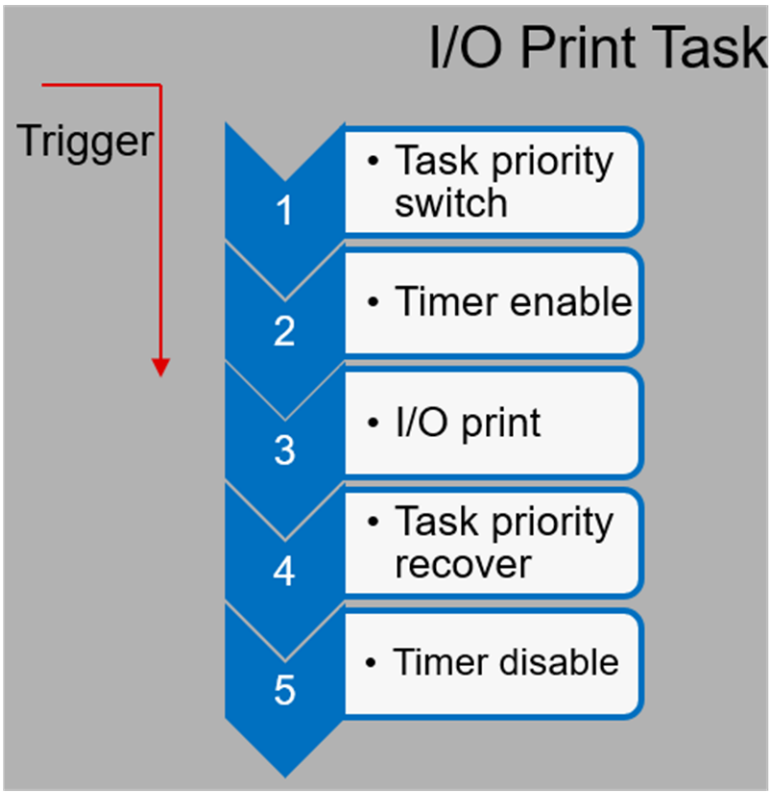

图 10. 任务与 Timer 执行流程

如上图所示，整个 IO 打印功能的实现流程包括任务优先级切换、定时器开启、IO 电平输出、任务优先级恢复、定时器关闭五个步骤。

5. 系统运行状态分析:

使用串口助手接收 IO 打印的数据验证数据的准确性，将 IO 口通过 CC2340 EVM 板载 UART 接口与 PC 连接：

图 11. EVM 连接

蓝牙协议栈一组广播包通常由 30 多个字节组成，为模拟蓝牙日志打印，在代码中创建一个 45 字节的数组来通过 IO 口进行打印：

```
uint8_t array[45] = {
                    0x01,   0x02,   0x03,   0x04,   0x05,   0x06,   0x07,   0x08,   0x09,
                    0x0A,   0x0B,   0x0C,   0x0D,   0x0E,   0x0F,   0x10,   0x11,   0x12,
                    0x13,   0x14,   0x15,   0x16,   0x17,   0x18,   0x19,   0x1A,   0x1B,
                    0x1C,   0x1D,   0x1E,   0x1F,   0x20,   0x21,   0x22,   0x23,   0x24,
                    0x25,   0x26,   0x27,   0x28,   0x29,   0x2A,   0x2B,   0x2C,   0x2D

};
```

图 12. Demo 数组定义

不同波特率 IO 打印数据验证：

波特率 4800：

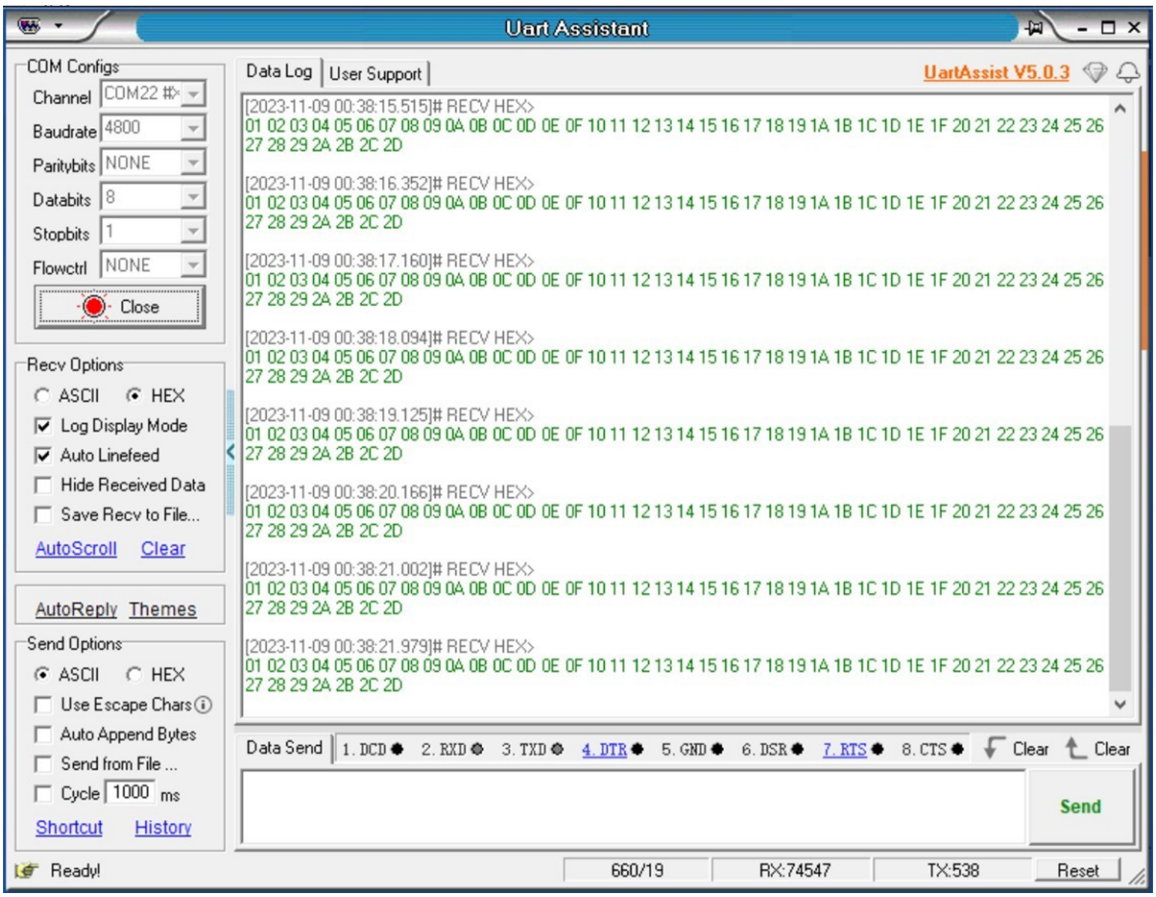

图 13. 4800 串口助手接收页

IO 在 4800 波特率下打印数据，串口助手接收的数据与代码中定义的数据一致。

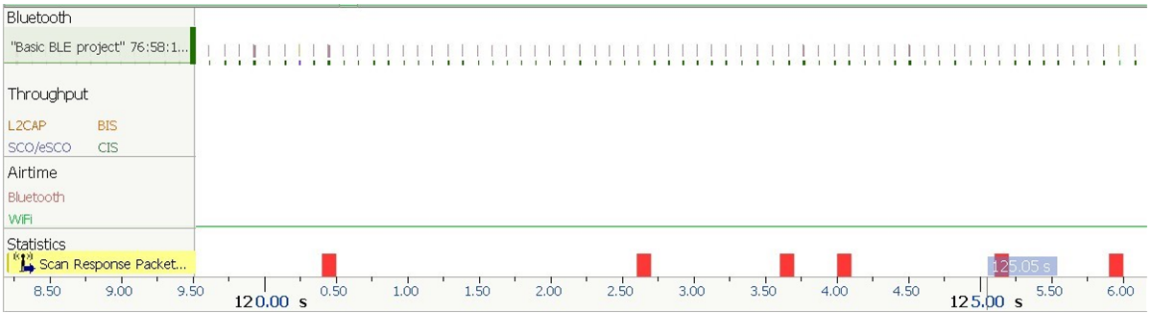

图 14. 4800 波特率 Sniffer 蓝牙空中抓包

IO 在 4800 波特率下打印数据，空中抓取的蓝牙广播包保持在正常水平，没有出现明显丢包。

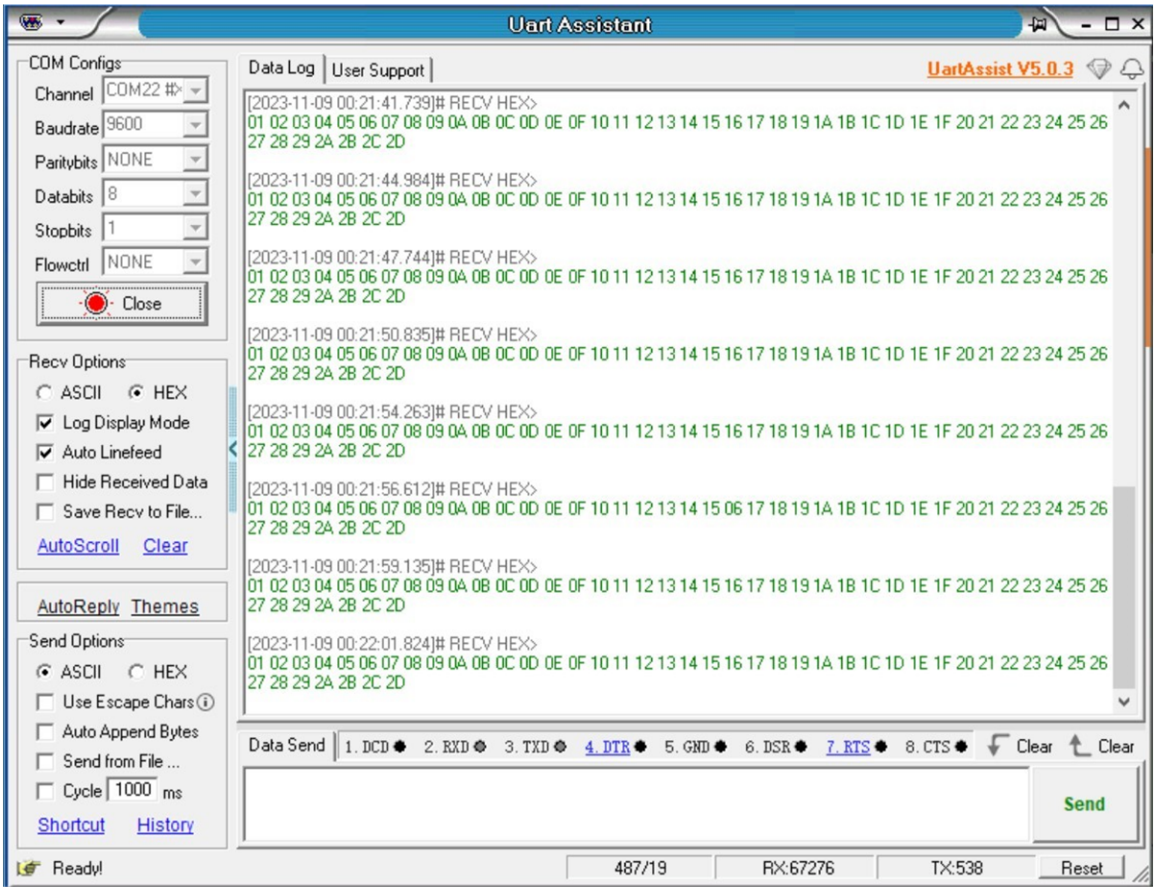

图 15. 9600 串口助手接收页

IO 在 9600 波特率下打印数据，串口助手接收的数据与代码中定义的数据一致。

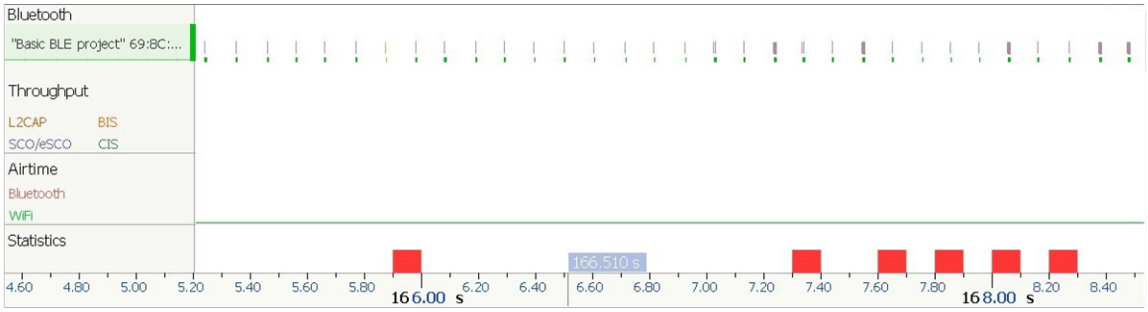

图 16. 9600 波特率 Sniffer 蓝牙空中抓包

IO 在 9600 波特率下打印数据，空中抓取的蓝牙广播包保持在正常水平，没有出现明显丢包。

波特率 14400：

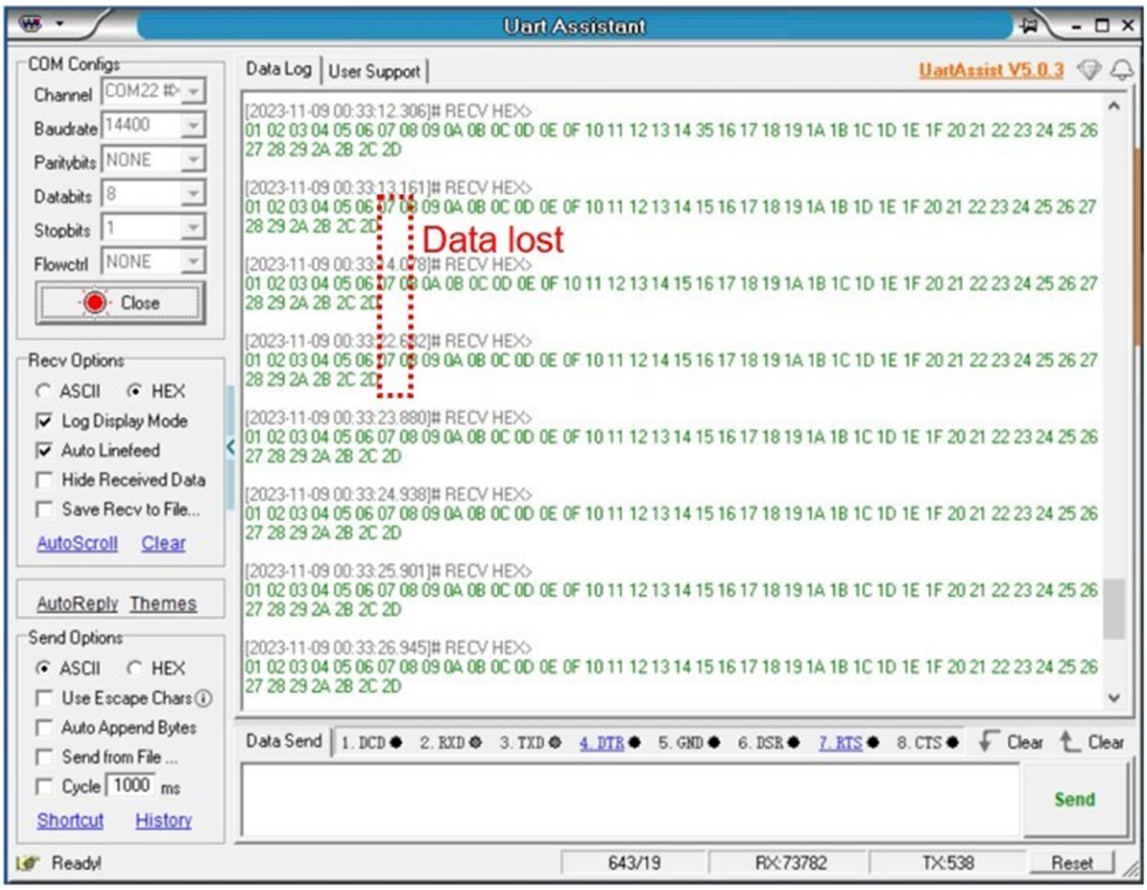

图 17. 14400 波特率串口助手接收页

IO 在 14400 波特率下打印数据，串口助手接收的数据会存在间接性丢失或错乱。

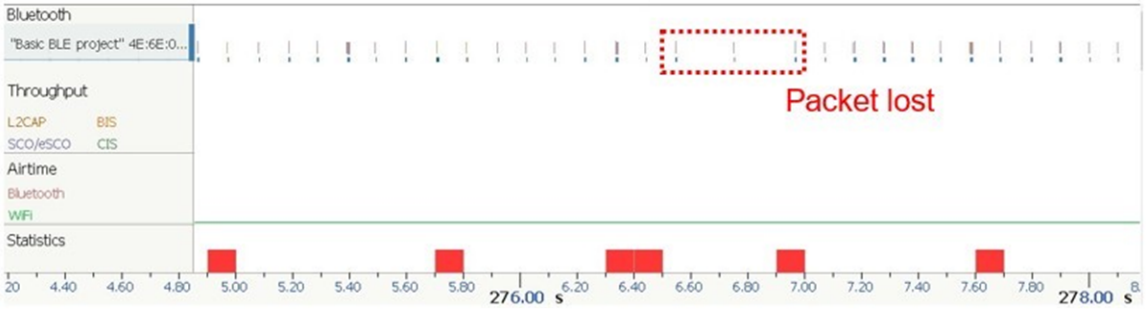

图 18. 14400 波特率 Sniffer 蓝牙空中抓包

IO 在 14400 波特率下打印数据，空中抓取的蓝牙广播包存在间接性广播间隔异常、丢包。

6. 总结:

根据 Demo 测试验证结果来看，CC2340 基于蓝牙协议栈扩展 IO 口模拟 UART 打印功能是能够实现的，但如果波特率过高会产生高频的中断处理，从而影响蓝牙协议栈的稳定运行，上述方案综合实验验证在波特率 9600 内可以保证 IO 打印数据的准确性，同时可以确保蓝牙广播维持在正常状态。

本文基于 CC2340 验证了 IO 模拟 UART 的可行性，根据现有例程进行扩展，在有限的样本和实验手法下进行了系统分析，用户在实际应用过程中需结合实际系统验证最终效果。

7. 参考文献:

1. CC23xx SimpleLink Wireless MCU Technical Reference Manual
2. FreeRTOS quick start guide
3. SimpleLink™ basic_ble examples academy
4. Snifer analyzer and user guide

8. 附录

8.1 部分伪代码

a.前台任务数据处理函数

```
void UARTIO_bufwrite(uint8_t *buffers, uint32_t size)
{
    static uint8_t *p_buf;
if(!cntsend)
{
        p_buf = buffers;
        bufsize = size;
        g_flag_batch = 0;
        LGPTimerLPF3_start(lgptHandle, LGPTimerLPF3_CTL_MODE_UP_PER);
    }
while(!g_flag_batch)
{
        if((!bufsend) && (!g_flag_batch) && (!sequence))
        {
            if(bufsize)
            {
                bufsend = *p_buf;
                p_buf++;
                cntsend++;
                flag_send = 1;
            }
        }
    }
}
```

b.Timer 电平控制函数

```
void UARTIO_write1(void)
{
    buffer11 = bufsend;
    if((sequence < 3) && buffer11 && (!g_flag_batch)){
        switch(sequence){
            case 0:
                GPIO_write(CONFIG_GPIO_IOUART, CONFIG_GPIO_LED_OFF);
                sequence +=1;
                break;
            case 1:
                if(bit < 8){
                    CONFIG_GPIO_LEVEL = (buffer11 >> bit) & 0x01;
                    GPIO_write(CONFIG_GPIO_IOUART, CONFIG_GPIO_LEVEL);
                    bit++;
                    if(bit == 8){
                        sequence +=1;
                        bit = 0;
                    }
                }
                break;
            case 2:
                GPIO_write(CONFIG_GPIO_IOUART, CONFIG_GPIO_LED_ON);
                bufsize --;
                bufsend = 0;
                sequence = 0;
                if(bufsize == 0){
                    g_flag_batch = 1;
                    cntsend = 0;
                    g_flag_button = 0;
                    LGPTimerLPF3_stop(lgptHandle);
                }
        }
    }
}
```

b.Timer 电平控制函数

```
pthread_attr_init(&attrs);
priParam.sched_priority = 1;
retc  = pthread_attr_setschedparam(&attrs, &priParam);
retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
retc |= pthread_attr_setstacksize(&attrs, THREADSTACKSIZE);
if (retc != 0)
{
    while (1) {}
}
retc = pthread_create(&thread, &attrs, MyIOThread, NULL);
if (retc != 0)
{
    while (1) {}
}
```

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265