PAN1080 UART Sample Application Note

PAN-CLT-VER-B0, Rev 0.1

PanchipMicroelectronics

www.panchip.com

修订历史

版本	修订日期	描述
V0.1	2022-10-17	初始版本创建

目录

第1章 例程演示内容

1.1 测试内容

1. 寄存器默认值 （Register default value）
2. 数据位与停止位选择 （Data length and stop bit select）
   a) 5 bit 数据，1.5 bit 停止位
   b) 6 bit 数据，2 bit 停止位
   c) 7 bit 数据，2 bit 停止位
   d) 8 bit 数据，2 bit 停止位
3. 奇偶校验 （Parity check）
   a) 奇校验（ODD）
   b) 偶校验（EVEN）
   c) =1 校验（MARK）
   d) =0 校验（SPACE）
4. 波特率修改 （Baudrate modify）
   a) 波特率 9600
   b) 波特率 115200
   c) 波特率 345600
   d) 波特率 1M
   e) 波特率 2M
   f) 波特率随意指定
5. 使能 FIFO 并设定中断触发阈值 （Enable FIFO and set interrupt trigger threshold）
   a) 设定 trigger level：Rx FIFO 有 1 字节，Tx FIFO 空
   b) 设定 trigger level：Rx FIFO 有 4 字节，Tx FIFO 剩 2 字节
   c) 设定 trigger level：Rx FIFO 有 8 字节，Tx FIFO 剩 4 字节
   d) 设定 trigger level：Rx FIFO 有 14 字节，Tx FIFO 剩 8 字节
6. 使能 FIFO 并开启 LSR 中断 （Enable FIFO and Line Status Interrupt）
7. 使能 Auto Flow Control （Enable Auto Flow Control）
8. DMA 传输 （DMA transfer）
   a) Burst Transaction 收发数据
   b) Single Transaction 收发数据
   c) Burst & Single Transaction 收发数据
   d) 使用 DMA 并开启 UART Auto Flow Control 功能收发数据
9. 使能 9-bit Mode （Enable 9-bit Mode）

1.2 环境准备

1.2.1 软件环境

1.2.1.1 待测代码

测试工程文件：

<PAN1080-DK>\03_MCU\mcu_samples\UART\keil\UART.uvprojx

测试源主文件目录：

<PAN1080-DK>\03_MCU\mcu_samples\UART\src

1.2.1.2 软件工具

1、SecureCRT（用于显示 PC 与 EVB 的交互过程，打印 log 等）

2、KingstVIS（逻辑分析仪 LA1010 配套软件）

1.2.2 硬件环境

1、PAN1080 EVB 2 块

a) UART0（测试交互接口，TX: P00，RX: P01，波特率: 921600）

b) UART1（待测模块，TX: P14，RX: P15，CTS: P16，RTS: P17）

c) SWD（用来调试和烧录程序，SWDCLK: P46，SWDIO: P47）

2、逻辑分析仪（波形抓取工具）

3、JLink（SWD 调试与烧录工具）

第2章 例程演示流程

2.1 环境配置

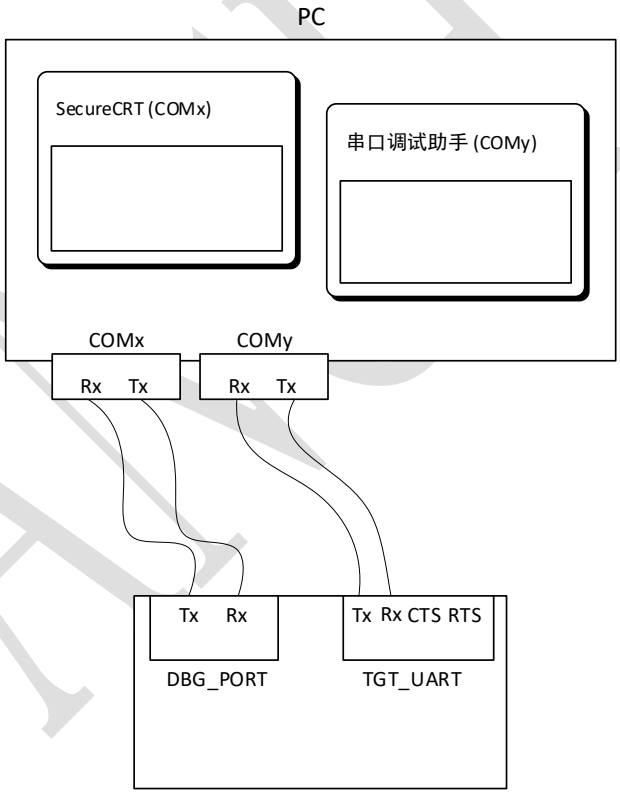
2.1.1 测试程序编译烧录

打开测试工程，确保可以编译通过。PAN1080 包含 2 个 UART Module：UART0 与 UART1，但由于 UART0 被用作日志输出，本测试工程默认将 UART1 作为待测 UART。

2.1.2 硬件接线

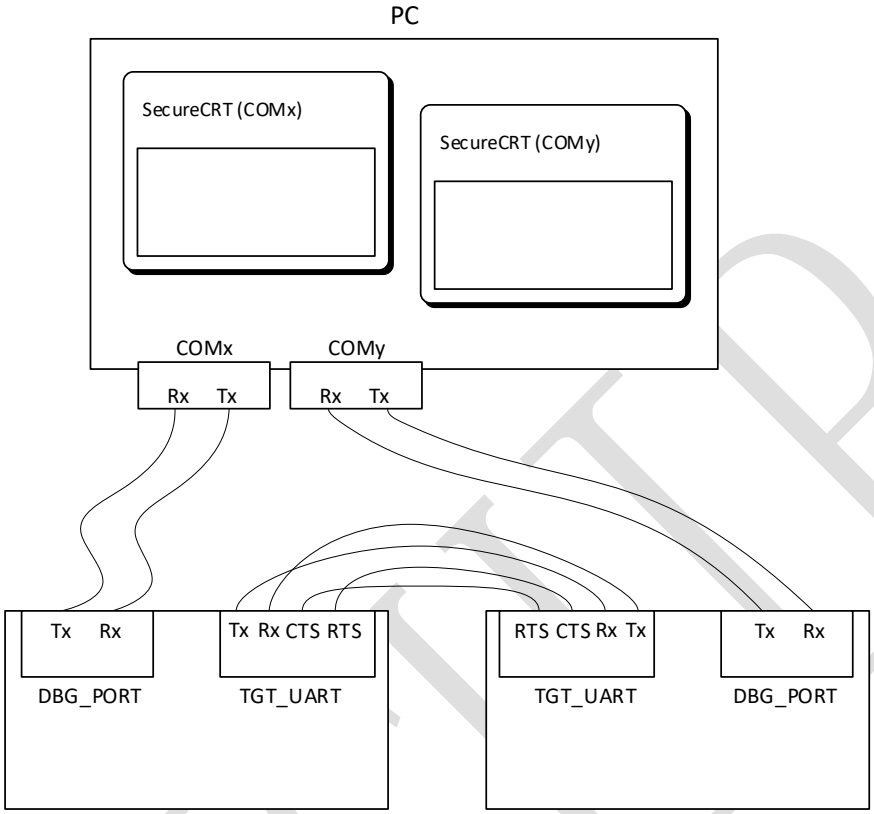
接线方面，不同的测试内容会需要不同的连线方法。

1. 测试项目 1~6，只需要与 PC 串口调试助手通信即可，按照下图的方式接线。其中 DBG_PORT（UART0）用来输入测试指令和打印 Log，与 PC 端 SecureCRT 对应的串口相连；TGT_UART（UART1）是待测模块，与 PC 端串口调试助手对应的串口相连。



PN108 COB Test Board

2. 测试项目 7~9，需要两个 EVB 互相通信，需要按照下图的方式接线。其中 DBG_PORT（UART0）用来输入测试指令和打印 Log，与 PC 端 SecureCRT 对应的串口相连；TGT_UART（UART1）是待测模块，需要两个 EVB 的待测模块对应引脚相连。

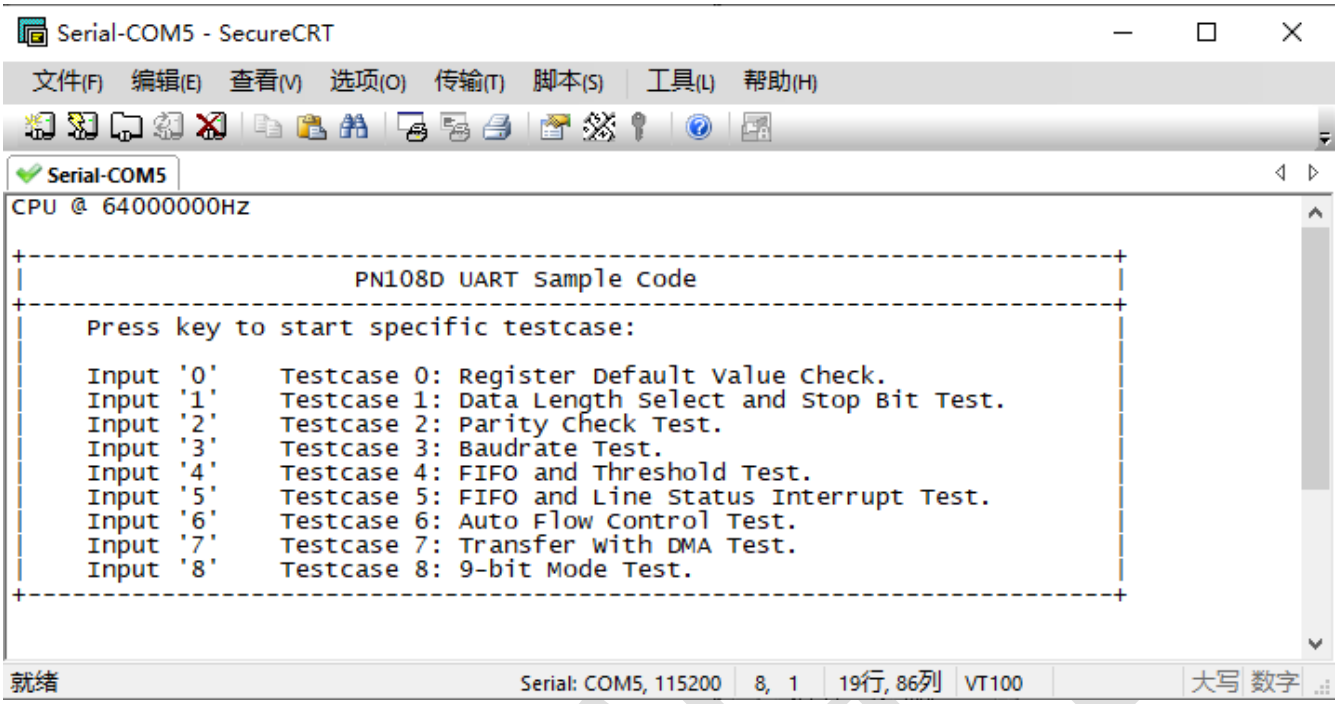


2.2 UART 工作流程

参考 User Manual 文档。

2.3 测试程序初始化

硬件连线完成并烧录测试程序后，EVB 上电，观察串口是否正常打印测试主菜单。



2.4 基本功能验证

2.4.1 UART 所有寄存器默认状态 UART_RegisterDefaultValueCheckCase0()

在主菜单下，输入‘0’命令 打印所有寄存器默认值：

测试目的：

检查所有 UART 相关寄存器复位 Default 值状态。

测试预期：

寄存器默认值应和 Datasheet 上 UART 模块默认值一致。

测试现象：

```
0
Target UART0 Register Default Values:
--------------------------
RBR_THR_DLL = 0x00000000
IER_DLH     = 0x00000000
IIR_FCR     = 0x00000001
LCR         = 0x00000000
MCR         = 0x00000000
LSR         = 0x00000060
MSR         = 0x00000001
SCR         = 0x00000000
LPDLL       = 0x00000000
LPDLH       = 0x00000000
USR         = 0x00000006
TFL         = 0x00000000
RFL         = 0x00000000
DMASA       = 0x00000000
DLF         = 0x00000000
RAR         = 0x00000000
TAR         = 0x00000000
LCR_EXT     = 0x00000000
UART Test OK, Success case: 0
```

测试分析：

参考芯片手册对比寄存器信息，均与手册一致，测试 OK。

2.4.2 数据位与停止位选择 UART_DataLenSelectAndStopBitTestCase1()；

在主菜单下，输入‘1’命令 进入 Subcase 菜单：

```
+---------------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):        |
|                                                                     |
|    Input 'A'    5-bit data with 1.5 stop bits.                      |
|    Input 'B'    6-bit data with 2   stop bits.                      |
|    Input 'C'    7-bit data with 2   stop bits.                      |
|    Input 'D'    8-bit data with 2   stop bits.                      |
|    Press ESC key to back to the top level case list.                |
+---------------------------------------------------------------------+
```

2.4.2.1 5 bit 数据，1.5 bit 停止位

测试目的：

验证 5-bit 数据、1.5-bit 停止位的设定下收发数据是否正常。

测试预期：

能够准确收发 5-bit 数据，无法准确收发 5-bit 以上的数据。

测试现象：

PC 端，串口调试助手先将数据位配置为 5，停止位配置为 1.5，然后打开串口准备接收数据。

EVB 端，输入‘A’命令，串口调试助手收到 16 字节的数据，其中前 8 字节正确，后 8 字节错误；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
a
Send data:
        0x00 0x01 0x02 0x03 0x1C 0x1D 0x1E 0x1F (should be valid)
        0x20 0x21 0x22 0x23 0xCC 0xDD 0xEE 0xFF (should be invalid)
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    5-bit data with 1.5 stop bits.                  |
|    Input 'B'    6-bit data with 2   stop bits.                  |
|    Input 'C'    7-bit data with 2   stop bits.                  |
|    Input 'D'    8-bit data with 2   stop bits.                  |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

前 8 字节 PC 端发送的均是小于 32 的数据，5-bit 能够 cover 到，因此 EVB 端能够准确接收

到，符合预期；

后 8 字节 PC 端发送的均是大于等于 32 的数据，5-bit 已经无法 cover 到，因此 EVB 端无法准确接收，符合预期；

最后 EVB 端发送 8 字节数据，PC 端也准确接收到，符合预期。

2.4.2.2 6 bit 数据，2 bit 停止位

测试目的：

验证 6-bit 数据、2-bit 停止位的设定下收发数据是否正常。

测试预期：

能够准确收发 6-bit 数据，无法准确收发 6-bit 以上的数据。

测试现象：

PC 端，串口调试助手先将数据位配置为 6，停止位配置为 2，然后打开串口准备接收数据。

EVB 端，输入‘B’命令，串口调试助手收到 16 字节的数据，其中前 8 字节正确，后 8 字节错误；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
b
Send data:
        0x00 0x01 0x02 0x03 0x3C 0x3D 0x3E 0x3F (should be valid)
        0x40 0x41 0x42 0x43 0xCC 0xDD 0xEE 0xFF (should be invalid)
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    5-bit data with 1.5 stop bits.                  |
|    Input 'B'    6-bit data with 2   stop bits.                  |
|    Input 'C'    7-bit data with 2   stop bits.                  |
|    Input 'D'    8-bit data with 2   stop bits.                  |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

前 8 字节 PC 端发送的均是小于 64 的数据，6-bit 能够 cover 到，因此 EVB 端能够准确接收

到，符合预期；

后 8 字节 PC 端发送的均是大于等于 64 的数据，6-bit 已经无法 cover 到，因此 EVB 端无法准确接收，符合预期；

最后 EVB 端发送 8 字节数据，PC 端也准确接收到，符合预期。

2.4.2.3 7 bit 数据，2 bit 停止位

测试目的：

验证 7-bit 数据、2-bit 停止位的设定下收发数据是否正常。

测试预期：

能够准确收发 7-bit 数据，无法准确收发 7-bit 以上的数据。

测试现象：

PC 端，串口调试助手先将数据位配置为 7，停止位配置为 2，然后打开串口准备接收数据。

EVB 端，输入‘C’命令，串口调试助手收到 16 字节的数据，其中前 8 字节正确，后 8 字节错误；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
C
Send data:
        0x00 0x01 0x40 0x44 0x68 0x6C 0x7E 0x7F (should be valid)
        0x80 0x81 0xAA 0xBB 0xCC 0xDD 0xFE 0xFF (should be invalid)
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    5-bit data with 1.5 stop bits.                  |
|    Input 'B'    6-bit data with 2   stop bits.                  |
|    Input 'C'    7-bit data with 2   stop bits.                  |
|    Input 'D'    8-bit data with 2   stop bits.                  |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

前 8 字节 PC 端发送的均是小于 128 的数据，7-bit 能够 cover 到，因此 EVB 端能够准确接

收到，符合预期；

后 8 字节 PC 端发送的均是大于等于 128 的数据，7-bit 已经无法 cover 到，因此 EVB 端无法准确接收，符合预期；

最后 EVB 端发送 8 字节数据，PC 端也准确接收到，符合预期。

2.4.2.4 8 bit 数据，2 bit 停止位

测试目的：

验证 8-bit 数据、2-bit 停止位的设定下收发数据是否正常。

测试预期：

能够准确收发 8-bit 数据。

测试现象：

PC 端，串口调试助手先将数据位配置为 8，停止位配置为 2，然后打开串口准备接收数据。

EVB 端，输入‘D’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
d
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF (should be valid)
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    5-bit data with 1.5 stop bits.                  |
|    Input 'B'    6-bit data with 2   stop bits.                  |
|    Input 'C'    7-bit data with 2   stop bits.                  |
|    Input 'D'    8-bit data with 2   stop bits.                  |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端发送的 8 个字节，而 8-bit 的数据格式能够 cover 到所有情况，因此 EVB 端准确接收

到，符合预期；

然后 EVB 端发送 8 字节数据，PC 端也准确接收到，符合预期。

2.4.3 奇偶校验 UART_ParityCheckTestCase2();

在主菜单下，输入‘2’命令 进入 Subcase 菜单：

```
+-------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):|
|                                                             |
|    Input 'A'    8-bit data with ODD   parity check.         |
|    Input 'B'    8-bit data with EVEN  parity check.         |
|    Input 'C'    8-bit data with MARK  parity check.         |
|    Input 'D'    8-bit data with SPACE parity check.         |
|    Press ESC key to back to the top level case list.        |
+-------------------------------------------------------------+
```

2.4.3.1 奇校验（ODD）

测试目的：

验证奇校验功能是否正常。

测试预期：

开启奇校验的情况下，能够准确收发数据。

测试现象：

PC 端，串口调试助手先将校验位设置为 ODD（奇校验），然后打开串口准备接收数据。

EVB 端，输入‘A’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到，且 Error Flag（从 LSR 中获取）均为 No Error。


串口调试助手
串口设置
串口号
COM24 #Si
波特率
115200
校验位
ODD
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
用户支持
UartAssist V4.3.13
[2022-02-16 10:34:51.664]# RECV HEX>
00 01 02 03 CC DD EE FF
[2022-02-16 10:34:52.728]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GND
清除
清除
12345678
发送
就绪!
19/9
RX:148
TX:72
复位计数

```
a
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data: 0x01, Error Flag: | No Error |
Data: 0x02, Error Flag: | No Error |
Data: 0x03, Error Flag: | No Error |
Data: 0x04, Error Flag: | No Error |
Data: 0x05, Error Flag: | No Error |
Data: 0x06, Error Flag: | No Error |
Data: 0x07, Error Flag: | No Error |
Data: 0x08, Error Flag: | No Error |

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    8-bit data with ODD   parity check.             |
|    Input 'B'    8-bit data with EVEN  parity check.             |
|    Input 'C'    8-bit data with MARK  parity check.             |
|    Input 'D'    8-bit data with SPACE parity check.             |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端均开启奇校验，互相收发数据均正常，符合预期。

2.4.3.2 偶校验（EVEN）

测试目的：

验证偶校验功能是否正常。

测试预期：

开启偶校验的情况下，能够准确收发数据。

测试现象：

PC 端，串口调试助手先将校验位设置为 EVEN（偶校验），然后打开串口准备接收数据。

EVB 端，输入‘B’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到，且 Error Flag（从 LSR 中获取）均为 No Error。


串口调试助手
串口设置
串口号
COM24 #Si
波特率
115200
校验位
EVEN
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
用户支持
UartAssist V4.3.13
[2022-02-16 10:35:26.518]# RECV HEX>
00 01 02 03 CC DD EE FF
[2022-02-16 10:35:27.968]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GNI
清除
清除
12345678
发送
就绪!
20/10
RX:156
TX:80
复位计数

```
b
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data: 0x01, Error Flag: | No Error |
Data: 0x02, Error Flag: | No Error |
Data: 0x03, Error Flag: | No Error |
Data: 0x04, Error Flag: | No Error |
Data: 0x05, Error Flag: | No Error |
Data: 0x06, Error Flag: | No Error |
Data: 0x07, Error Flag: | No Error |
Data: 0x08, Error Flag: | No Error |

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    8-bit data with ODD   parity check.             |
|    Input 'B'    8-bit data with EVEN  parity check.             |
|    Input 'C'    8-bit data with MARK  parity check.             |
|    Input 'D'    8-bit data with SPACE parity check.             |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端均开启偶校验，互相收发数据均正常，符合预期。

2.4.3.3 =1 校验（MARK）

测试目的：

验证固定奇校验功能是否正常。

测试预期：

开启固定奇校验的情况下，能够准确收发数据。

测试现象：

PC 端，串口调试助手先将校验位设置为 MARK（固定奇校验，=1 校验），然后打开串口准备接收数据。

EVB 端，输入‘C’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到，且 Error Flag（从 LSR 中获取）均为 No Error。


串口调试助手
串口设置
串口号
COM24 #Si
波特率
115200
校验位
MARK
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
用户支持
UartAssist V4.3.13
[2022-02-16 10:35:56.835]# RECV HEX>
00 01 02 03 CC DD EE FF
[2022-02-16 10:35:58.047]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GND
清除
清除
12345678
发送
就绪!
21/11
RX:164
TX:88
复位计数

```
c
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data: 0x01, Error Flag: | No Error |
Data: 0x02, Error Flag: | No Error |
Data: 0x03, Error Flag: | No Error |
Data: 0x04, Error Flag: | No Error |
Data: 0x05, Error Flag: | No Error |
Data: 0x06, Error Flag: | No Error |
Data: 0x07, Error Flag: | No Error |
Data: 0x08, Error Flag: | No Error |

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    8-bit data with ODD   parity check.             |
|    Input 'B'    8-bit data with EVEN  parity check.             |
|    Input 'C'    8-bit data with MARK  parity check.             |
|    Input 'D'    8-bit data with SPACE parity check.             |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端均开启固定奇校验，互相收发数据均正常，符合预期。

2.4.3.4 =0 校验（SPACE）

测试目的：

验证固定偶校验功能是否正常。

测试预期：

开启固定偶校验的情况下，能够准确收发数据。

测试现象：

PC 端，串口调试助手先将校验位设置为 SPACE（固定偶校验，=0 校验），然后打开串口准备接收数据。

EVB 端，输入‘D’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到，且 Error Flag（从 LSR 中获取）均为 No Error。


串口调试助手
串口设置
串口号
COM24 #Si
波特率
115200
校验位
SPACE
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
用户支持
UartAssist V4.3.13
[2022-02-16 10:36:36.002]# RECV HEX>
00 01 02 03 CC DD EE FF
[2022-02-16 10:36:36.977]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GND
清除
清除
12345678
发送
就绪!
22/12
RX:172
TX:96
复位计数

```
d
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data: 0x01, Error Flag: | No Error |
Data: 0x02, Error Flag: | No Error |
Data: 0x03, Error Flag: | No Error |
Data: 0x04, Error Flag: | No Error |
Data: 0x05, Error Flag: | No Error |
Data: 0x06, Error Flag: | No Error |
Data: 0x07, Error Flag: | No Error |
Data: 0x08, Error Flag: | No Error |

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    8-bit data with ODD   parity check.             |
|    Input 'B'    8-bit data with EVEN  parity check.             |
|    Input 'C'    8-bit data with MARK  parity check.             |
|    Input 'D'    8-bit data with SPACE parity check.             |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端均开启固定偶校验，互相收发数据均正常，符合预期。

2.4.3.5 校验错误演示

测试目的：

故意造出 Parity Bit（校验位）错误的情况，确认 EVB 的 UART 是否能够正确识别奇偶校验错误的情况。

测试预期：

PC 端发送具有错误校验位的数据，EVB 能够接收并识别到校验错误。

EVB 端发送具有错误校验位的数据，PC 端能够接收并识别到校验错误。

测试现象：

PC 端，串口调试助手先将校验位设置为 ODD（奇校验），然后打开串口准备接收数据；EVB 端，输入‘B’命令，进入 EVEN（偶校验）收发流程，这样对 PC 和 EVB 端来说，接收到的数据的校验位均是错的。

现象是，PC 端串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到，Error Flag 显示有奇偶校验错误。


串口调试助手
串口设置
串口号
COM24 #Si
波特率
115200
校验位
ODD
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
用户支持
UartAssist V4.3.13
[2022-02-16 10:37:06.335]# RECV HEX>
00 01 02 03 CC DD EE FF
[2022-02-16 10:37:08.272]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GND
清除
清除
12345678
发送
就绪!
23/13
RX:180
TX:104
复位计数

```
b
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data: 0x01, Error Flag: | Parity Error |
Data: 0x02, Error Flag: | Parity Error |
Data: 0x03, Error Flag: | Parity Error |
Data: 0x04, Error Flag: | Parity Error |
Data: 0x05, Error Flag: | Parity Error |
Data: 0x06, Error Flag: | Parity Error |
Data: 0x07, Error Flag: | Parity Error |
Data: 0x08, Error Flag: | Parity Error |

+-----------------------------------------------------------------+
|    Press key to test specific function (Baudrate = 115200):     |
|                                                                 |
|    Input 'A'    8-bit data with ODD   parity check.             |
|    Input 'B'    8-bit data with EVEN  parity check.             |
|    Input 'C'    8-bit data with MARK  parity check.             |
|    Input 'D'    8-bit data with SPACE parity check.             |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

从现象来看，PC 端串口调试助手是准确收到 EVB 发来的数据，但由于串口助手没有显示奇偶校验错误的功能，因此无法判断奇偶校验错误，这是符合预期的；随后使用串口调试助手发送数据，EVB 端也准确接收到，但 Error Flag（从 LSR 中获取）显示有奇偶校验错误，也符合预期。

2.4.4 波特率修改 UART_BaudrateTestCase3();

在主菜单下，输入 '3' 命令 进入 Subcase 菜单：

```
+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    Input 'A'    8-bit data under 9600       baudrate.           |
|    Input 'B'    8-bit data under 115200     baudrate.           |
|    Input 'C'    8-bit data under 345600     baudrate.           |
|    Input 'D'    8-bit data under 1M         baudrate.           |
|    Input 'E'    8-bit data under 2M         baudrate.           |
|    Input 'F'    8-bit data under customized baudrate.           |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

2.4.4.1 波特率 9600

测试目的：

验证 9600 波特率下是否工作正常。

测试预期：

9600 波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为 9600，然后打开串口准备接收数据。

EVB 端，输入‘A’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
a
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    Input 'A'    8-bit data under 9600       baudrate.           |
|    Input 'B'    8-bit data under 115200     baudrate.           |
|    Input 'C'    8-bit data under 345600     baudrate.           |
|    Input 'D'    8-bit data under 1M         baudrate.           |
|    Input 'E'    8-bit data under 2M         baudrate.           |
|    Input 'F'    8-bit data under customized baudrate.           |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

2.4.4.2 波特率 115200

测试目的：

验证 115200 波特率下是否工作正常。

测试预期：

115200 波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为 115200，然后打开串口准备接收数据。

EVB 端，输入‘B’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
b
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-------------------------------------------------------------------+
|    Press key to test specific function:                           |
|                                                                   |
|    Input 'A'    8-bit data under 9600       baudrate.             |
|    Input 'B'    8-bit data under 115200     baudrate.             |
|    Input 'C'    8-bit data under 345600     baudrate.             |
|    Input 'D'    8-bit data under 1M         baudrate.             |
|    Input 'E'    8-bit data under 2M         baudrate.             |
|    Input 'F'    8-bit data under customized baudrate.             |
|    Press ESC key to back to the top level case list.              |
+-------------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

2.4.4.3 波特率 345600

测试目的：

验证 345600 波特率下是否工作正常。

测试预期：

345600 波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为 345600，然后打开串口准备接收数据。

EVB 端，输入‘C’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。

```
c
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+---------------------------------------------------------------------+
|    Press key to test specific function:                             |
|                                                                     |
|    Input 'A'    8-bit data under 9600       baudrate.               |
|    Input 'B'    8-bit data under 115200     baudrate.               |
|    Input 'C'    8-bit data under 345600     baudrate.               |
|    Input 'D'    8-bit data under 1M         baudrate.               |
|    Input 'E'    8-bit data under 2M         baudrate.               |
|    Input 'F'    8-bit data under customized baudrate.               |
|    Press ESC key to back to the top level case list.                |
+---------------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

2.4.4.4 波特率 1M

测试目的：

验证 1M 波特率下是否工作正常。

测试预期：

1M 波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为 1M，然后打开串口准备接收数据。

EVB 端，输入‘D’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
d
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    Input 'A'    8-bit data under 9600       baudrate.           |
|    Input 'B'    8-bit data under 115200     baudrate.           |
|    Input 'C'    8-bit data under 345600     baudrate.           |
|    Input 'D'    8-bit data under 1M         baudrate.           |
|    Input 'E'    8-bit data under 2M         baudrate.           |
|    Input 'F'    8-bit data under customized baudrate.           |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

注意：当使用 USB 转串口板与 PC 通信时，某些品牌的转板可能无法支持非标准波特率（1M、2M 等）通信。

2.4.4.5 波特率 2M

测试目的：

验证 2M 波特率下是否工作正常。

测试预期：

2M 波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为 2M，然后打开串口准备接收数据。

EVB 端，输入‘E’命令，串口调试助手收到 8 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
e
Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    Input 'A'    8-bit data under 9600       baudrate.           |
|    Input 'B'    8-bit data under 115200     baudrate.           |
|    Input 'C'    8-bit data under 345600     baudrate.           |
|    Input 'D'    8-bit data under 1M         baudrate.           |
|    Input 'E'    8-bit data under 2M         baudrate.           |
|    Input 'F'    8-bit data under customized baudrate.           |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

注意：当使用 USB 转串口板与 PC 通信时，某些品牌的转板可能无法支持非标准波特率（1M、2M 等）通信。

2.4.4.6 波特率随意指定

测试目的：

验证自定义波特率是否工作正常。

测试预期：

自定义波特率下能够准确收发数据。

测试现象：

PC 端，串口调试助手先将波特率设置为一个自定义值，如 500000，然后打开串口准备接收数据。

EVB 端，输入‘F’命令，按照提示再输入‘500000’并回车，此时串口调试助手收到 8 字节的数据，检查发现均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


```
f
Please input new baudrate (Press <Enter> to confirm):
500000
Baudrate changed, new value: 500000

Send data:
        0x00 0x01 0x02 0x03 0xCC 0xDD 0xEE 0xFF
Data sent successfully.

Try to receive 8 bytes of data...
Data received: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    Input 'A'    8-bit data under 9600       baudrate.           |
|    Input 'B'    8-bit data under 115200     baudrate.           |
|    Input 'C'    8-bit data under 345600     baudrate.           |
|    Input 'D'    8-bit data under 1M         baudrate.           |
|    Input 'E'    8-bit data under 2M         baudrate.           |
|    Input 'F'    8-bit data under customized baudrate.           |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

PC 端与 EVB 端互相收发数据均正常，符合预期。

注意：当使用 USB 转串口板与 PC 通信时，某些品牌的转板可能无法支持非标准波特率（1M、2M 等）通信。

2.4.5 使能 FIFO 并设定中断触发阈值 UART_FifoAndThresholdTestCase4();

在主菜单下，输入‘4’命令 进入 Subcase 菜单：

```
+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=None, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Trigger Rx: FIFO 1 char,   Tx: FIFO Empty.      |
|    Input 'B'    Trigger Rx: FIFO 1/4 full, Tx: FIFO 2 chars.    |
|    Input 'C'    Trigger Rx: FIFO 1/2 full, Tx: FIFO 1/4 full.   |
|    Input 'D'    Trigger Rx: FIFO full-2,   Tx: FIFO 1/2 full.   |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

2.4.5.1 设定 trigger level：Rx FIFO 有 1 字节，Tx FIFO 空

测试目的：

验证阈值设定为“Rx FIFO 有 1 字节，Tx FIFO 空”的情况下，收发数据是否正常。

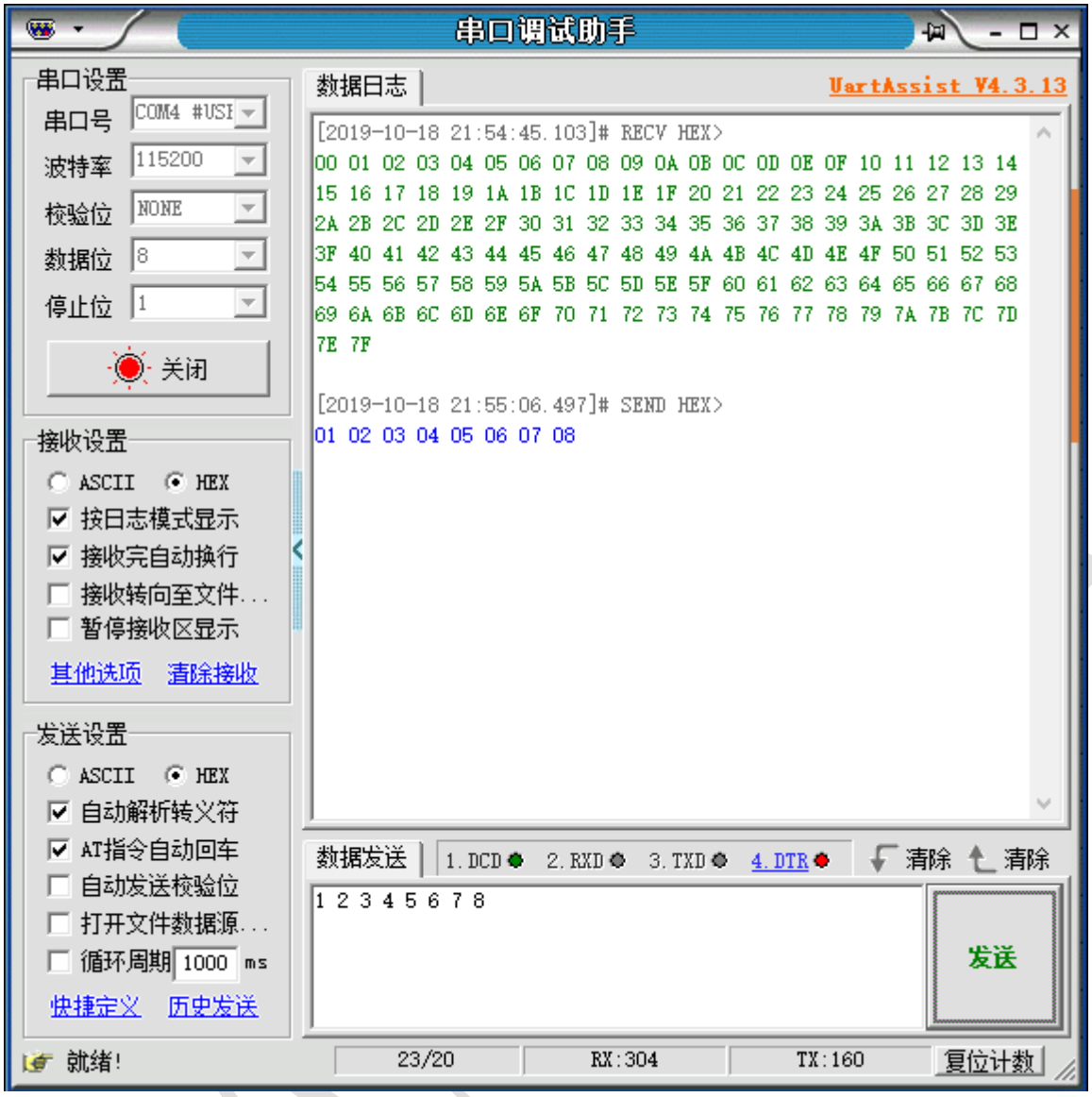
测试预期：

收发数据正常。

测试现象：

PC 端，串口调试助手无需特别设置，直接打开串口准备接收数据。

EVB 端，输入‘A’命令，串口调试助手收到 128 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


串口调试助手
串口设置
串口号 COM4 #USI
波特率 115200
校验位 NONE
数据位 8
停止位 1
关闭
接收设置
ASCII HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项 清除接收
发送设置
ASCII HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期 1000 ms
快捷定义 历史发送
数据日志
UartAssist V4.3.13
[2019-10-18 21:54:45.103]# RECV HEX>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29
2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E
3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D
7E 7F
[2019-10-18 21:55:06.497]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD 2.RXD 3.TXD 4.DTR
清除 清除
1 2 3 4 5 6 7 8
发送
就绪!
23/20
RX:304
TX:160
复位计数

```
aFIFO trigger level setting done, prepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Rx thres trig!
BeRx thres trig!
gin to receive data...
Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=None, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Trigger Rx: FIFO 1 char,   Tx: FIFO Empty.      |
|    Input 'B'    Trigger Rx: FIFO 1/4 full, Tx: FIFO 2 chars.    |
|    Input 'C'    Trigger Rx: FIFO 1/2 full, Tx: FIFO 1/4 full.   |
|    Input 'D'    Trigger Rx: FIFO full-2,   Tx: FIFO 1/2 full.   |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

首先 EVB 端使用中断方式发送 128 字节数据，PC 端准确接收到，符合预期。

随后 PC 端发送 8 个字节数据，EVB 端使用中断方式也准确接收到，符合预期。

2.4.5.2 设定 trigger level：Rx FIFO 有 4 字节，Tx FIFO 剩 2 字节

测试目的：

验证阈值设定为“Rx FIFO 有 4 字节，Tx FIFO 剩 2 字节”的情况下，收发数据是否正常。

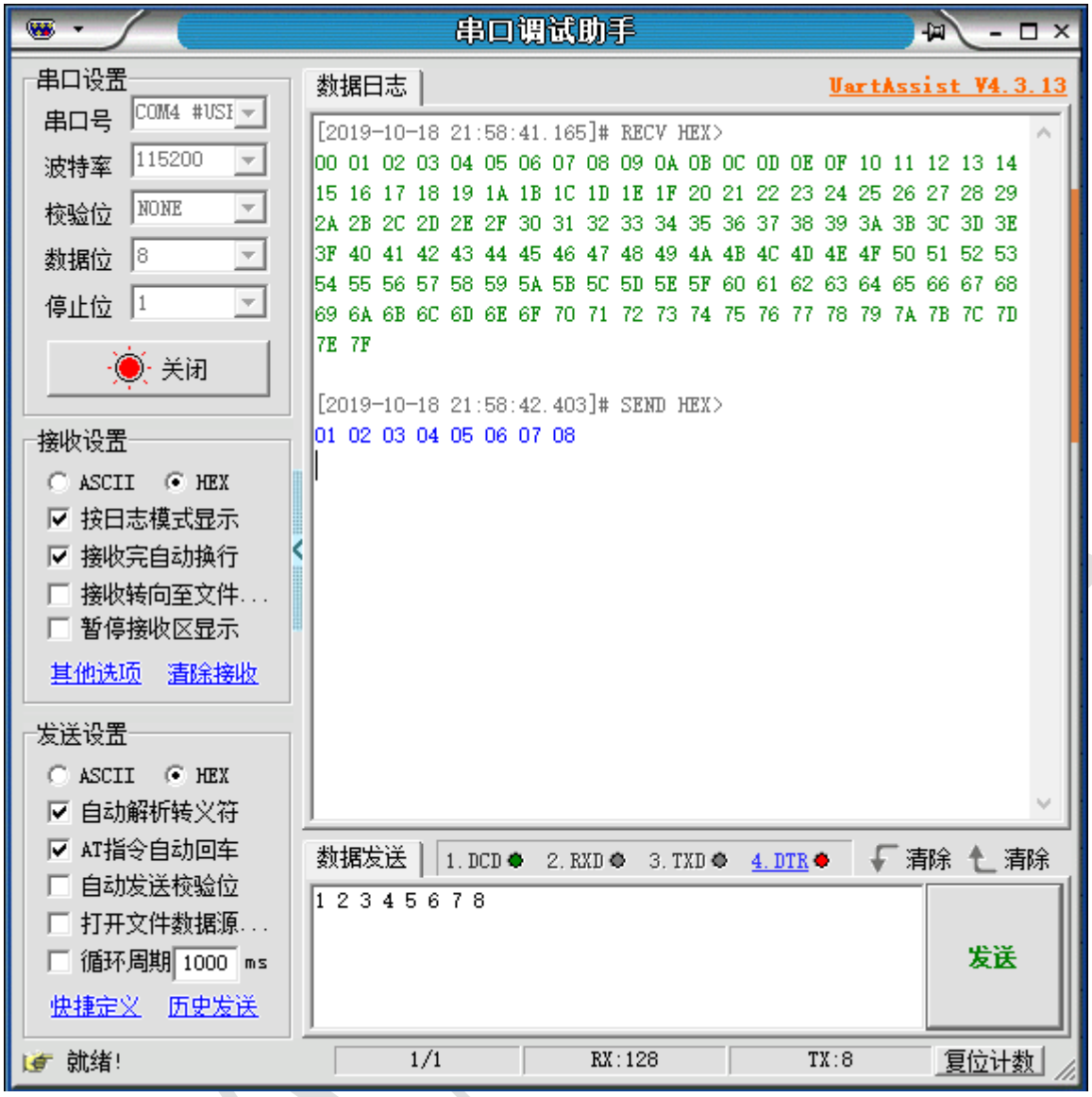
测试预期：

收发数据正常。

测试现象：

PC 端，串口调试助手无需特别设置，直接打开串口准备接收数据。

EVB 端，输入‘B’命令，串口调试助手收到 128 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


串口调试助手
串口设置
串口号
COM4 #USI
波特率
115200
校验位
NONE
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
UartAssist V4.3.13
[2019-10-18 21:58:41.165]# RECV HEX>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29
2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E
3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D
7E 7F
[2019-10-18 21:58:42.403]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
清除
清除
1 2 3 4 5 6 7 8
发送
就绪!
1/1
RX:128
TX:8
复位计数

```
bFIFO trigger level setting done, prepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Begin to receive daRx thres trig!
ta...
Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=None, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Trigger Rx: FIFO 1 char,   Tx: FIFO Empty.      |
|    Input 'B'    Trigger Rx: FIFO 1/4 full, Tx: FIFO 2 chars.    |
|    Input 'C'    Trigger Rx: FIFO 1/2 full, Tx: FIFO 1/4 full.   |
|    Input 'D'    Trigger Rx: FIFO full-2,   Tx: FIFO 1/2 full.   |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

首先 EVB 端使用中断方式发送 128 字节数据，PC 端准确接收到，符合预期。

随后 PC 端发送 8 个字节数据，EVB 端使用中断方式也准确接收到，符合预期。

2.4.5.3 设定 trigger level：Rx FIFO 有 8 字节，Tx FIFO 剩 4 字节

测试目的：

验证阈值设定为“Rx FIFO 有 8 字节，Tx FIFO 剩 4 字节”的情况下，收发数据是否正常。

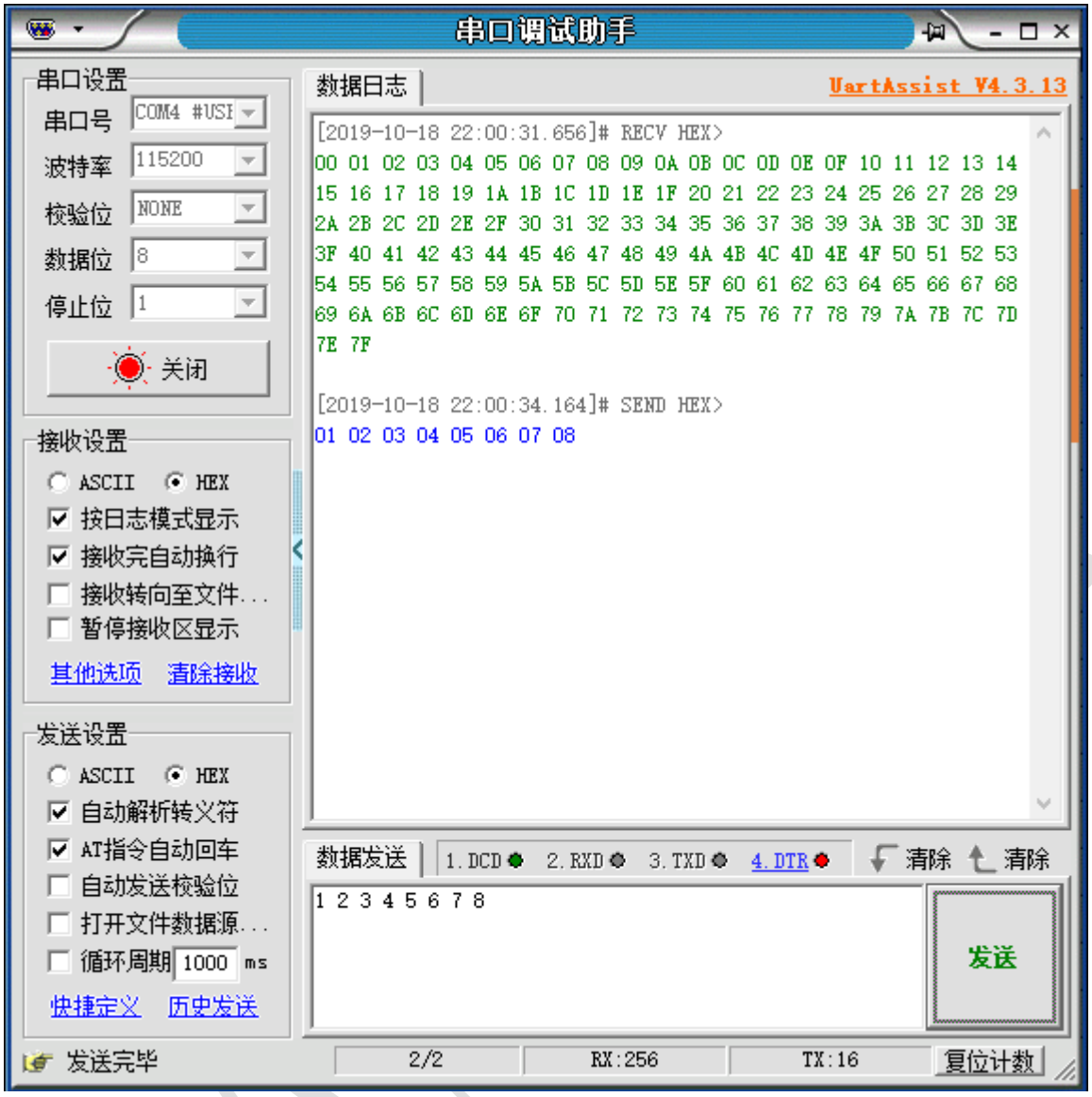
测试预期：

收发数据正常。

测试现象：

PC 端，串口调试助手无需特别设置，直接打开串口准备接收数据。

EVB 端，输入‘C’命令，串口调试助手收到 128 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


串口调试助手
串口设置
串口号 COM4 #USI
波特率 115200
校验位 NONE
数据位 8
停止位 1
关闭
接收设置
ASCII HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项 清除接收
发送设置
ASCII HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期 1000 ms
快捷定义 历史发送
数据日志
UartAssist V4.3.13
[2019-10-18 22:00:31.656]# RECV HEX>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29
2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E
3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D
7E 7F
[2019-10-18 22:00:34.164]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD 2.RXD 3.TXD 4.DTR
清除 清除
1 2 3 4 5 6 7 8
发送
发送完毕
2/2
RX:256
TX:16
复位计数

```
cFIFO trigger level setting done, prepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Begin to receive data..Rx thres trig!
.
Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=None, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Trigger Rx: FIFO 1 char,   Tx: FIFO Empty.      |
|    Input 'B'    Trigger Rx: FIFO 1/4 full, Tx: FIFO 2 chars.    |
|    Input 'C'    Trigger Rx: FIFO 1/2 full, Tx: FIFO 1/4 full.   |
|    Input 'D'    Trigger Rx: FIFO full-2,   Tx: FIFO 1/2 full.   |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

首先 EVB 端使用中断方式发送 128 字节数据，PC 端准确接收到，符合预期。

随后 PC 端发送 8 个字节数据，EVB 端使用中断方式也准确接收到，符合预期。

2.4.5.4 设定 trigger level：Rx FIFO 有 14 字节，Tx FIFO 剩 8 字节

测试目的：

验证阈值设定为“Rx FIFO 有 14 字节，Tx FIFO 剩 8 字节”的情况下，收发数据是否正常。

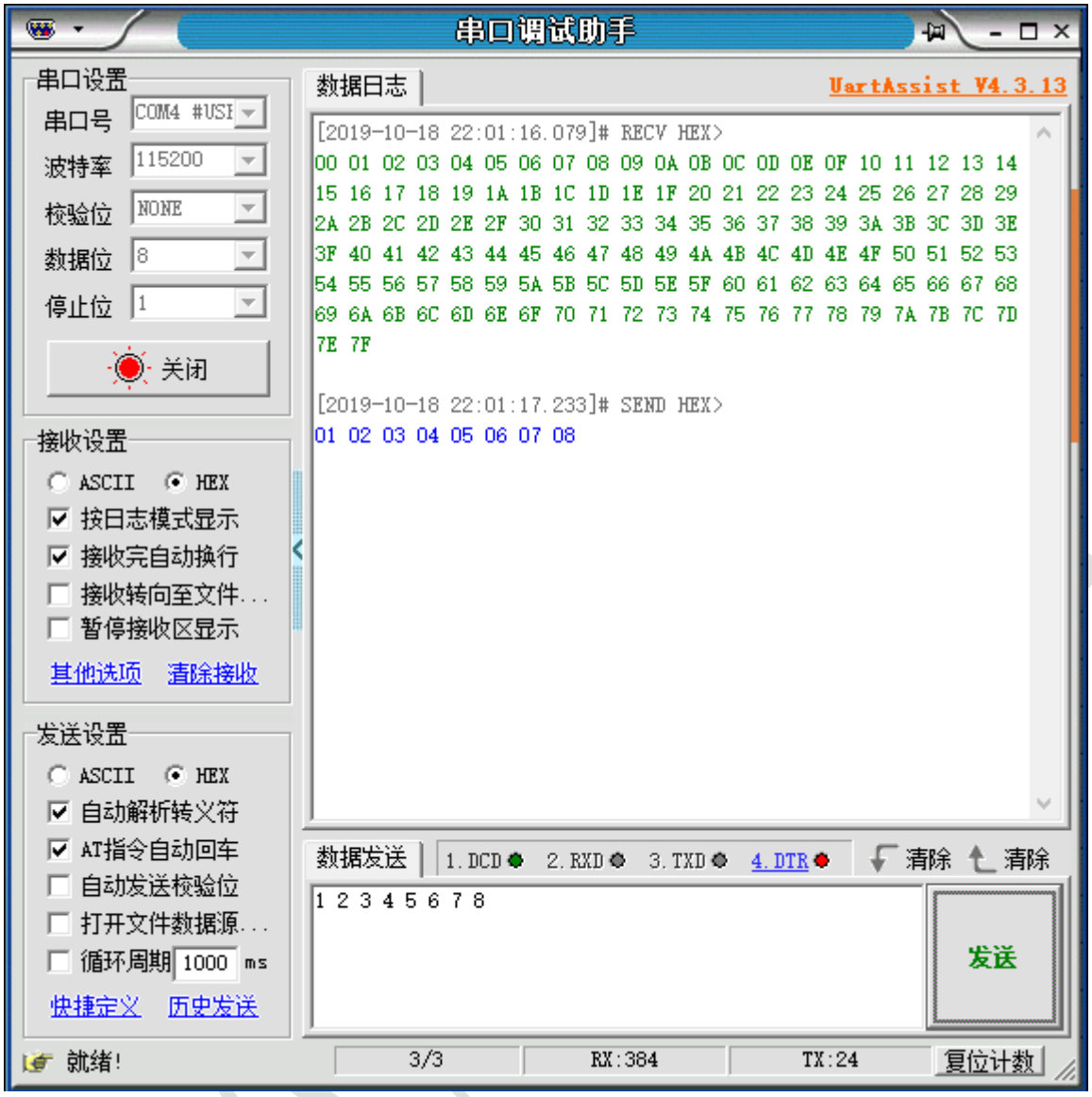
测试预期：

收发数据正常。

测试现象：

PC 端，串口调试助手无需特别设置，直接打开串口准备接收数据。

EVB 端，输入‘D’命令，串口调试助手收到 128 字节的数据均正确；随后使用串口调试助手发送 8 字节数据，从 EVB 端的 Log 可以看到也有准确接收到。


串口调试助手
串口设置
串口号
COM4 #USI
波特率
115200
校验位
NONE
数据位
8
停止位
1
关闭
接收设置
ASCII
HEX
按日志模式显示
接收完自动换行
接收转向至文件...
暂停接收区显示
其他选项
清除接收
发送设置
ASCII
HEX
自动解析转义符
AT指令自动回车
自动发送校验位
打开文件数据源...
循环周期
1000
ms
快捷定义
历史发送
数据日志
UartAssist V4.3.13
[2019-10-18 22:01:16.079]# RECV HEX>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29
2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E
3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D
7E 7F
[2019-10-18 22:01:17.233]# SEND HEX>
01 02 03 04 05 06 07 08
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
清除
清除
1 2 3 4 5 6 7 8
发送
就绪!
3/3
RX:384
TX:24
复位计数

```
dFIFO trigger level setting done, prepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Begin to receive data...
Rx tout trig!
Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=None, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Trigger Rx: FIFO 1 char,   Tx: FIFO Empty.      |
|    Input 'B'    Trigger Rx: FIFO 1/4 full, Tx: FIFO 2 chars.    |
|    Input 'C'    Trigger Rx: FIFO 1/2 full, Tx: FIFO 1/4 full.   |
|    Input 'D'    Trigger Rx: FIFO full-2,   Tx: FIFO 1/2 full.   |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

首先 EVB 端使用中断方式发送 128 字节数据，PC 端准确接收到，符合预期。

随后 PC 端发送 8 个字节数据，EVB 端使用中断方式也准确接收到，符合预期。

2.4.6 使能 FIFO 并开启 LSR 中断 UART_FifoAndLineStatusInterruptTestCase5();

在主菜单下，输入‘5’命令 进入 Subcase 菜单：

```
+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=ODD, DataBits=8, StopBits=1 >      |
|                                                                 |
|    Input 'A'    Prepare to receive data.                        |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试目的：

验证有 Line Error 的情况下，中断是否正确触发以及 LSR 寄存器各 Flag 是否正常。

测试预期：

LSR 中断正确触发且寄存器 Flag 正常。。

测试现象：

EVB 端，输入‘A’命令，进入带有奇校验的数据接收流程，准备接收 PC 端发来的数据。

a) PC 端，串口调试助手将校验位设置为 NONE，然后打开串口，发送 8 字节数据。接着在

EVB 端看到数据准确接收到，并有 Line Error 中断触发，Error Flag 显示有 3 个 Error 被检测到，分别是 Receiver FIFO Error、Framing Error 以及 Parity Error。


```
aPrepare to receive data...

An Uart line error occured, error type: | Receiver FIFO Error | Framing Error | Parity Error |
Line status, THR or TxFIFO empty!
Rx thres trig!
Begin to receive data...
Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-------------------------------------------------------------------+
|    Press key to test specific function:                           |
|                                                                   |
|    < Baudrate=115200, Parity=ODD, DataBits=8, StopBits=1 >        |
|                                                                   |
|    Input 'A'    Prepare to receive data.                          |
|                                                                   |
|    Press ESC key to back to the top level case list.              |
+-------------------------------------------------------------------+
```

b) PC 端，串口调试助手将校验位设置为 ODD，然后打开串口，发送 8 字节数据。接着在

EVB 端看到数据准确接收到，且只有 Rx Threshold 中断被触发，没有 Line Error 中断触发。


```
aPrepare to receive data...

Begin to receive data...Rx thres trig!

Data received (length=8):
 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=ODD, DataBits=8, StopBits=1 >      |
|                                                                 |
|    Input 'A'    Prepare to receive data.                        |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

现象 a)，PC 端发送了带有无校验位的 8-bit 数据，而 EVB 端使用奇校验的模式来接收数据，于是会发现接收到的数据有帧格式错误和校验位错误，这两个错误也会附带上 Recv FIFO 错误，因此现象是符合预期的。

现象 b)，PC 端发送了带有奇校验为的 8-bit 数据，而 EVB 端也是使用奇校验的模式来接收数据，于是会发现接收到的数据都是正确的了，现象也是符合预期的。

2.4.7 使能 Auto Flow Control UART_AutoFlowControlTestCase6();

在主菜单下，输入‘6’命令 进入 Subcase 菜单：

```
+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Work as receiving end.                          |
|    Input 'B'    Work as transmitting end.                       |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

2.4.7.1 先启动 Receive，后启动 Transmit

测试目的：

验证 Auto Flow Control（AFC）功能打开后，先启动 Receive，后启动 Transmit 的情况下收发数据是否正常。

测试预期：

AFC 模式下先启动 Receive，后启动 Transmit 收发数据正常。

测试现象：

先操作 EVB 1，输入‘A’命令，进入带有自动流控的数据接收流程，准备接收 EVB 2 发来的数据，此时两个 EVB 均无中断被触发，使用逻辑分析仪监测 4 根 UART 线也发现没有任何电平变化。

再操作 EVB 2，输入‘B’命令，进入带有自动流控的数据发送流程，向 EVB 1 发送数据，此时发现两个 EVB 分别触发了 Rx 中断和 Tx 中断，随后 EVB 1 成功收到 EVB 2 发来的 128 字节数据。

```
aPrepare to receive data...
Begin to receive data...
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx tout trig!
Data received done (length=128):
 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f

+----------------------------------------------------------------------+
|    Press key to test specific function:                              |
|                                                                      |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >          |
|                                                                      |
|    Input 'A'    Work as receiving end.                               |
|    Input 'B'    Work as transmitting end.                            |
|                                                                      |
|    Press ESC key to back to the top level case list.                 |
+----------------------------------------------------------------------+



bPrepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Data transmitting done, size: 128

+----------------------------------------------------------------------+
|    Press key to test specific function:                              |
|                                                                      |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >          |
|                                                                      |
|    Input 'A'    Work as receiving end.                               |
|    Input 'B'    Work as transmitting end.                            |
|                                                                      |
|    Press ESC key to back to the top level case list.                 |
+----------------------------------------------------------------------+
```

测试分析：

AFC 模式下先启动 Receive，后启动 Transmit，发现收发数据正常，符合预期。

2.4.7.2 先启动 Transmit，后启动 Receive

测试目的：

验证 Auto Flow Control（AFC）功能打开后，先启动 Transmit，后启动 Receive 的情况下收发数据是否正常。

测试预期：

AFC 模式下先启动 Transmit，后启动 Receive 收发数据正常。

测试现象：

先操作 EVB 1，输入‘B’命令，进入带有自动流控的数据发送流程，向 EVB 2 发送数据，此时两个 EVB 均无中断被触发，使用逻辑分析仪监测 4 根 UART 线也发现没有任何电平变化。

再操作 EVB 2，输入‘A’命令，进入带有自动流控的数据接收流程，准备接收 EVB 1 发来的数据，此时发现两个 EVB 分别触发了 Tx 中断和 Rx 中断，随后 EVB 2 成功收到 EVB 1 发来的 128 字节数据。

```
bPrepare to transmit data (128 Bytes)...
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Tx thres trig!
Data transmitting done, size: 128

+-----------------------------------------------------------------------+
|    Press key to test specific function:                               |
|                                                                       |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >           |
|                                                                       |
|    Input 'A'    Work as receiving end.                                |
|    Input 'B'    Work as transmitting end.                             |
|                                                                       |
|    Press ESC key to back to the top level case list.                  |
+-----------------------------------------------------------------------+
```

```
aPrepare to receive data...
Rx thres trig!
Begin to receiRx thres trig!
ve data...
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Rx thres trig!
Data received done (length=128):
 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Work as receiving end.                          |
|    Input 'B'    Work as transmitting end.                       |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

AFC 模式下先在 EVB 1 启动 Transmit，这时候因为 EVB 2 的 Receive 流程还没启动，所以 EVB 1 的 CTS 信号线（即 EVB 2 的 RTS 信号线）是没有被 Receive 端拉低（Assert）的，因此数据不会被发出来；随后启动 EVB 2 启动 Receive 流程，RTS 信号拉低告诉 EVB 1 可以发送数据，于是 EVB 1 立刻开始发送数据，当接收端 FIFO 数据超过设定的阈值后，硬件自动将 RTS 信号拉高，告诉发送端暂停发送数据，带接收端 FIFO 被读空后，硬件再自动将 RTS 信号拉低，告诉发送端继续发送数据，如此往复直到所有数据都收发完成。从现象也看出收发数据正常，符合预期。

2.4.8 DMA 传输 UART_TransferWithDmaTestCase7();

在主菜单下，输入‘7’命令 进入 Subcase 菜单：

```
+-------------------------------------------------------------+
|    Press key to test specific function:                     |
|                                                             |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 > |
|                                                             |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                             |
|    Press ESC key to back to the top level case list.        |
+-------------------------------------------------------------+
```

2.4.8.1 Burst Transaction 收发数据

测试目的：

验证 DMA Burst Transaction 下 UART 收发数据是否正常。

测试预期：

DMA Burst Transaction 下 UART 可以正常收发数据。

测试现象：

先操作 EVB 1，输入‘A’命令，进入使用 DMA 的 UART 数据接收流程，准备接收 EVB 2 发来的数据。

再操作 EVB 2，输入‘B’命令，进入使用 DMA 的数据发送流程，向 EVB 1 发送数据，稍后发现两个 EVB 均触发了 DMA 接收和发送完成的中断，从 EVB 1 的 Log 可以看到成功接收到 128 字节的数据。

```
aStart to receive data by DMA, wait for data...
Data received done (channel=0, data_len=128):
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=16, dsttfr=0, err=0

 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+

bStart to send data by DMA...
Send data done (channel=0, data_len=128).
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=0, dsttfr=15, err=0

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

使用 DMA 收发 128 字节数据，测试程序中 UART Burst Length 设置为 8 字节，可以整除 128，因此 DMA 只会有 Burst Transaction，从现象看收发数据均正常，符合预期。

2.4.8.2 Single Transaction 收发数据

测试目的：

验证 DMA Single Transaction 下 UART 收发数据是否正常。

测试预期：

DMA Single Transaction 下 UART 可以正常收发数据。

测试现象：

先操作 EVB 1，输入‘C’命令，进入使用 DMA 的 UART 数据接收流程，准备接收 EVB 2 发来的数据。

再操作 EVB 2，输入‘D’命令，进入使用 DMA 的数据发送流程，向 EVB 1 发送数据，稍后发现两个 EVB 均触发了 DMA 接收和发送完成的中断，从 EVB 1 的 Log 可以看到成功接收到 4 字节的数据。

```
cStart to receive data by DMA, wait for data...
Data received done (channel=0, data_len=4):
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=4, dsttfr=0, err=0

 0x00 0x01 0x02 0x03

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

```
dStart to send data by DMA...
Send data done (channel=0, data_len=4).
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=0, dsttfr=1, err=0

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

使用 DMA 收发 4 字节数据，测试程序中 UART Burst Length 设置为 8 字节，大于需要收发的数据大小，因此 DMA 只会有 Single Transaction，从现象看收发数据均正常，符合预期。

2.4.8.3 Burst & Single Transaction 收发数据

测试目的：

验证 DMA Burst 和 Single Transaction 均有的情况下 UART 收发数据是否正常。

测试预期：

DMA Burst 和 Single Transaction 均有的情况下 UART 可以正常收发数据。

测试现象：

先操作 EVB 1，输入‘E’命令，进入使用 DMA 的 UART 数据接收流程，准备接收 EVB 2 发来的数据。

再操作 EVB 2，输入‘F’命令，进入使用 DMA 的数据发送流程，向 EVB 1 发送数据，稍后发现两个 EVB 均触发了 DMA 接收和发送完成的中断，从 EVB 1 的 Log 可以看到成功接收到 132 字节的数据。

```
eStart to receive data by DMA, wait for data...
Data received done (channel=0, data_len=132):
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=20, dsttfr=0, err=0

 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f
 0x80 0x81 0x82 0x83

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+

fStart to send data by DMA...
Send data done (channel=0, data_len=132).
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=0, dsttfr=19, err=0

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

使用 DMA 收发 132 字节数据，测试程序中 UART Burst Length 设置为 8 字节，无法整除 132，因此 DMA 会先进入 Burst Transaction 传输前 128 字节数据，然后进入 Single Transaction 传输剩余的 4 个字节数据，从现象看收发数据均正常，符合预期。

2.4.8.4 使用 DMA 并开启 UART Auto Flow Control 功能收发数据

测试目的：

验证 DMA 与 UART Auto Flow Control 同时打开的情况下收发数据是否正常。

测试预期：

DMA 与 UART Auto Flow Control 同时打开，能够正常收发数据。

测试现象：

先操作 EVB 1，输入‘G’命令，进入使用 DMA 和 AFC 的 UART 数据接收流程，准备接收 EVB 2 发来的数据。

再操作 EVB 2，输入‘H’命令，进入使用 DMA 和 AFC 的数据发送流程，向 EVB 1 发送数据，稍后发现两个 EVB 均触发了 DMA 接收和发送完成的中断，从 EVB 1 的 Log 可以看到成功接收到 128 字节的数据。

```
gStart to receive data by DMA, wait for data...
Data received done (channel=0, data_len=128):
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=16, dsttfr=0, err=0

 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47
 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67
 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77
 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

```
hStart to send data by DMA...
Send data done (channel=0, data_len=128).
DMA Interrupt Trigger Count: tfr=1, blk=1, srctfr=0, dsttfr=15, err=0

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=8, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Burst transaction receiving end (uart2mem).     |
|    Input 'B'    Burst transaction transmitting end (mem2uart).  |
|    Input 'C'    Single transaction receiving end (uart2mem).    |
|    Input 'D'    Single transaction transmitting end (mem2uart). |
|    Input 'E'    Both transactions receiving end (uart2mem).     |
|    Input 'F'    Both transactions transmitting end (mem2uart).  |
|    Input 'G'    DMA with UART AFC receiving end (uart2mem).     |
|    Input 'H'    DMA with UART AFC transmitting end (mem2uart).  |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

开启 AFC 的情况下，可以准确收发数据（无论是先开启接收流程还是先开启发送流程），符合预期。

2.4.9 使能 9-bit Mode UART_9BitModeTestCase8();

在主菜单下，输入‘8’命令 进入 Subcase 菜单：

```
+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=9, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Work as receiving end.                          |
|    Input 'B'    Work as transmitting end.                       |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试目的：

验证 9-bit mode 下 UART 是否能正常工作。

测试预期：

9-bit mode 下 UART 能正常收发数据。

测试现象：

先操作 EVB 1，输入‘A’命令，进入使用 9-bit mode 的 UART 数据接收流程，准备接收 EVB 2 发来的数据，接收地址设定为 0xb4。

再操作 EVB 2，输入‘B’命令，进入使用 9-bit mode 的 UART 数据发送流程，向 EVB 1 发送数据，先向 0xb4 的地址发送 32 字节的数据，随后向 0x87 的地址发送 32 字节的数据，接着重新向 0xb4 的地址发送 32 字节的数据。从 EVB 1 的 Log 可以看到成功接收到地址为 0xb4 的 64 字节的数据。

```
aPrepare to receive data (Addr: 0xb4)...
Data received done (length=64):
 0x00 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 0x2e 0x2f
 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37
 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=9, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Work as receiving end.                          |
|    Input 'B'    Work as transmitting end.                       |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+


bBegin to transmit address1 (0xb4) ...
Prepare to transmit the first half of data1 (32 Bytes)...
Transmitting done.
Begin to transmit address2 (0x87) ...
Prepare to transmit data2 (32 Bytes)...
Transmitting done.
Begin to transmit address1 (0xb4) ...
Prepare to transmit the second half of data1 (32 Bytes)...
Transmitting done.

+-----------------------------------------------------------------+
|    Press key to test specific function:                         |
|                                                                 |
|    < Baudrate=115200, Parity=NONE, DataBits=9, StopBits=1 >     |
|                                                                 |
|    Input 'A'    Work as receiving end.                          |
|    Input 'B'    Work as transmitting end.                       |
|                                                                 |
|    Press ESC key to back to the top level case list.            |
+-----------------------------------------------------------------+
```

测试分析：

9-bit mode 下，接收端将只会接收具有预先设定地址（0xb4）的数据流，对于地址不匹配的数据（0x87）硬件将其直接忽略掉，直到接收端再一次接收到具有预先设定地址的数据流。测试现象符合预期。

第3章 使用注意事项

1、 波特率不可以设置太高，原因如果很高（如超过 apbclk/16）的时候，Uart Init flow 计算 integerdivider 会小于 100，导致 DLH 和 DLL 都是 0，uart 硬件会被认为不使能而不工作；

2、 在使用 USB 转串口转接板，CP2102 不支持非标准波特率，而 CH340 是支持的，因此测波特率的 testcase 最好使用 CH340 来测；

3、 Cob 上使用的 usb 转串口对波特率速率有一定影响，使用 usb 转串口最大不超过 600K，直接使用 CH340 接 IO 可以达到最大 3M 以上（再大工具不支持）

4、 UART 中断 handler 中，某些情况下会触发 Uart_event_none（编号为 0x1）这个 event，软件无需处理，直接忽略掉即可；

5、 Auto flow control，Rx FIFO data 超过阈值，硬件自动拉高 RTS 电平，随后在 Rx handler 中需要将 FIFO 读空，硬件才会重新将 RTS 电平拉低；

6、 使用 Auto flow control 或 Uart DMA，可以开 Uart 发送中断(IER.bit[1])/接收中断(IER.bit[0])，然后在中断处理函数中处理 data，也可以不开这两个中断，直接使用主程序处理 data；

7、 若 dma source/destination 是 memory，则对应的 srctfr/dsttfr 中断应当 disable（enable 无意义，因为 memory 不存在 transaction level 的概念）