Application Note

Application Note

Document No.: AN1104

APM32F4xx Series Software Simulating USART

Version: V1.0

1 Introduction

During the use of the system, the system resources (e.g. serial ports) may be insufficient. In this case, the IO port can be used to simulate the serial port to serve the same function. This application note introduces how to realize simulation of serial port by IO port on the APM32F4xx series through external interrupts and timers.

Contents

2 Introduction to APM32F4xx USART

USART (universal synchronous/asynchronous receiver transmitter) is a serial communication device that can flexibly exchange full-duplex and half-duplex data with external devices, and meets the requirements of external devices for industry standard NRZ asynchronous serial data format. USART also provides a wide range of baud rate and supports multiprocessor communication. USART supports not only standard asynchronous transceiver mode, but also some other serial data exchange modes, such as LIN protocol, smart card protocol, IrDA SIR ENDEC specification and hardware flow control mode. USART also supports DMA function to realize high-speed data communication.

2.1 Serial port mode

Serial communication refers to the communication mode of sequentially transmitting data bit by bit. The serial ports are divided into synchronous serial interfaces and asynchronous serial interfaces.

Synchronous mode: The data blocks in one transmission contains a large amount of data, so the receive clock shall be strictly synchronized with the transmit clock, and there is an additional signal line USART_CK that can output the synchronous clock compared with asynchronous mode. It is applicable to the situations where a large amount of data needs to be transmitted.

The figure below is the USART synchronous transmission timing diagram, which shows four situations of free combination of polarity and phase. In Figure 1, DBLCFG=0 for the USART_CTRL1 register, corresponding to the situation of 1 start bit, 8 data bits and 1 stop bit. In Figure 2, DBLCFG=1 for USART_CTRL1 register, corresponding to the situation of 1 start bit, 9 data bits and 1 stop bit. The clock polarity of USART_CK is determined by CPOL bit of USART_CTRL2 register: when CPOL is 0, the idle state of the CK pin is low level; when CPOL is 1, the idle state of the CK pin is high level. The phase of USART_CK is determined by CPHA bit of USART_CTRL2 register: when CPHA is 0, it means sampling is conducted on the edge of the first clock; when CPOL is 1, it means sampling is conducted on the edge of the second clock.

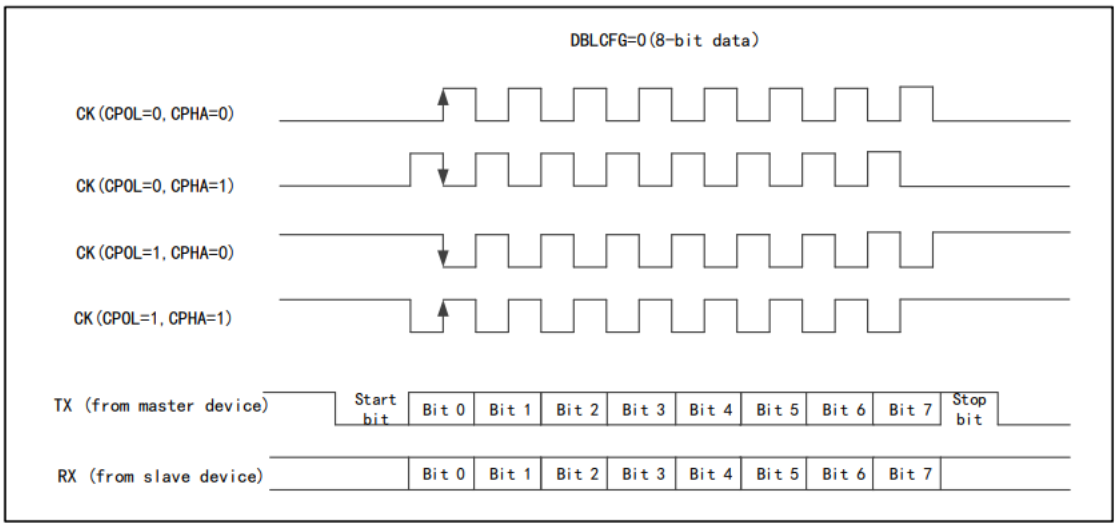

Figure 1 USART Synchronous Transmission Timing Diagram (DBLCFG=0)

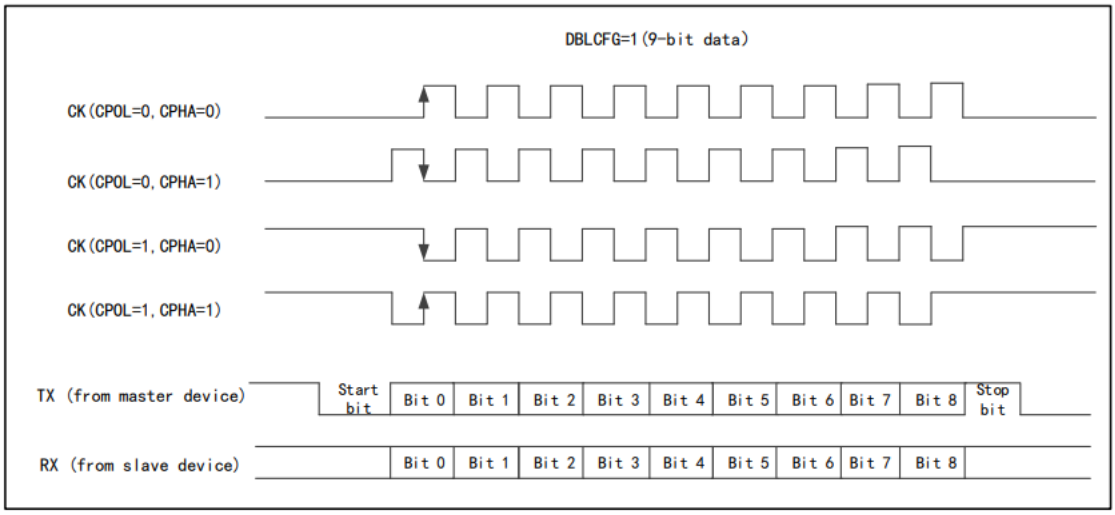

Figure 2 USART Synchronous Transmission Timing Diagram (DBLCFG=1)

The asynchronous mode is serial, asynchronous, and full-duplex communication. Synchronous communication is achieved by agreeing on the same baud rate between the transmitter and the receiver, and data is transmitted in the same frame format. Usually it is used in short-distances and low-speed industrial applications. This document mainly discusses the asynchronous serial port communication.

2.2 Overview of UART communication protocol

The asynchronous communication requires both the transmitter and receiver to agree on the baud rate and determine the duration of each bit to ensure the timing synchronization of both sides. The baud rate is the number of symbols of code element transmitted per unit time.

As shown in Figures 3 and 4, the message format of the serial port is: start bit (1 bit) + data bit (5~8 bits) + parity bit (0/1 bit) + stop bit (0.5~1.5 bits).

Start bit: When a serial port is used and the transmission is effective, a 1bit low-level start bit will be automatically generated.

Data bit: The length of valid data during communication, usually 5~8 bits. Before transmitting and receiving data, the corresponding configuration should be completed.

Parity check bit: Add the check bit to verify whether there are data transmission errors caused by interference in the transmission process. Set to odd parity to ensure that the number of logical high bits in the transmitted data is odd; set to even parity to ensure that the number of logical high bits in the transmitted data is even.

Stop bit: After the valid data is transmitted, transmit the high level of the set number of bits, indicating the end of transmission of one-frame data.

Idle bit: The data line remains in high-bit state before the logical low bit of next start bit arrives. The idle bits are not the content of the message.

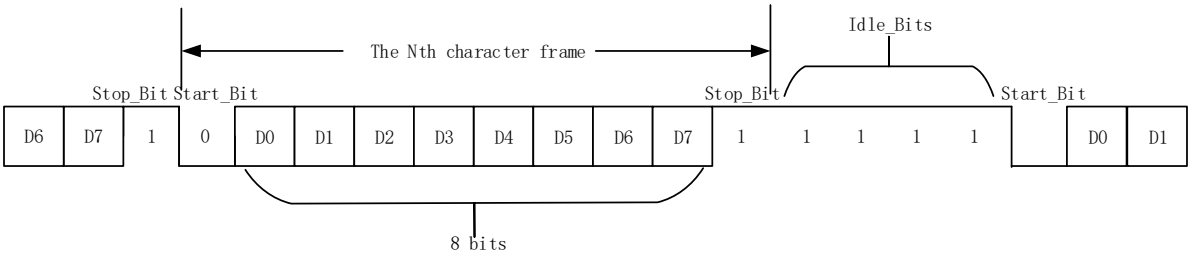

Figure 3 Serial Port Frame Structure Diagram (including idle bits)

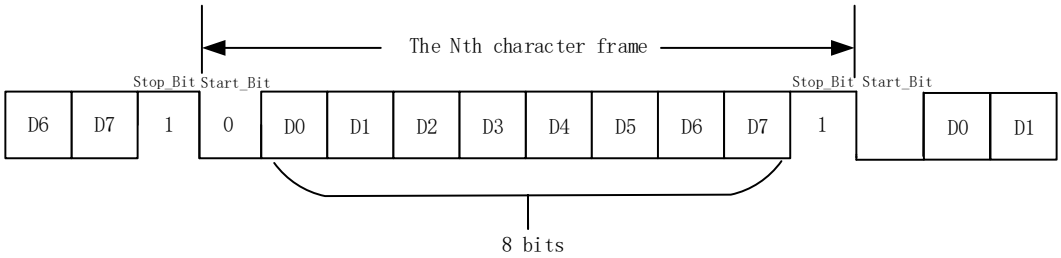

Figure 4 Serial Port Frame Structure Diagram (excluding idle bits)

3 Software simulating serial port

3.1 Hardware design

This design uses GPIO PB9 and PB10 of APM32F407ZGT6 to simulate the transmitting line TX and receiving line RX of the serial port, respectively. Use the USB-to-TTL line to connect RX to PB9 of the development board, TX to PB10, and connect the ground wires on both sides.

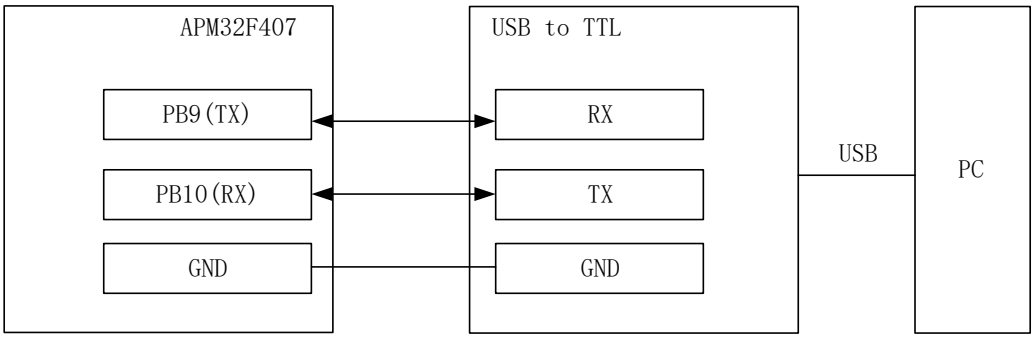

Figure 5 Hardware Connection Diagram

3.2 Software design

The implementation function of this design is to receive data transmitted from the serial assistant and then transmit the same data for echo. Set the baud rate to 9600 bps, start bit to 1 bit, data bit to 8 bits, no parity bit, and stop bit to 1 bit.

The design idea is to divide the function implementation into two parts i.e. receiving part and transmitting part.

The implementation of receiving function needs to take precise delay as a prerequisite to obtain correct data. The routine uses a general-purpose timer for timing of 104us (t=1/9600 s=104us) to enter an interrupt and the data is received bit by bit in the interrupt service function.

Store the received data in the buffer and transmit it to the serial assistant for echo.

The implementation of the transmitting function is relatively simple: set the data line to 0 and simulate the start bit; use a tick timer for delay of 104us; use the for loop to transmit and delay bit by bit. After transmission of 8 data bit is completed, set the data line to 1 and simulate the stop bit.

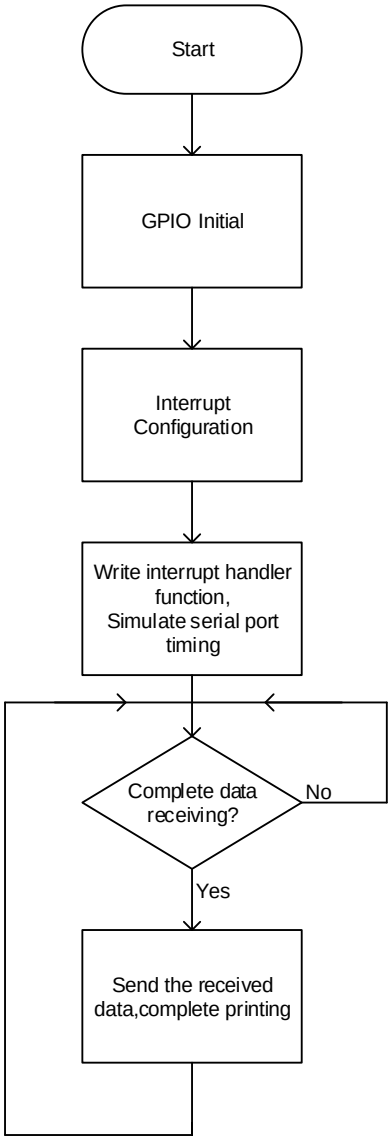

Figure 6 Software Implementation Flow Chart

3.2.1 GPIO configuration

The function completes the initialization configuration of GPIO PB9 and PB10.

Configure the transmitting pin TX PB9 to push-pull output mode with an output speed of 50MHz. When the serial port is idle, the data bit is in a high-level state. Therefore, after the configuration is completed, PB9 needs to be set to 1.

Configure the receiving pin RX PB10 to pull-up input mode, configure the external interrupt and a falling edge will be generated before data transmission begins. Therefore, the PB10 interrupt trigger method is set to falling edge trigger.

```
void Soft_Usart_Init(void)
{
    GPIO_Config_T    gpioConfig;
    EINT_Config_T    eintConfig;

    RCM_EnableAHB1PeriphClock(RCM_AHB1_PERIPH_GPIOB);
    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_SYSCFG);

    /* Tx GPIOB PIN9 */
    GPIO_ConfigStructInit(&gpioConfig);
    gpioConfig.mode = GPIO_MODE_OUT;
    gpioConfig.otype = GPIO_OTYPE_PP;
    gpioConfig.pin = GPIO_PIN_9;
    gpioConfig.speed = GPIO_SPEED_50MHz;
    GPIO_Config(GPIOB,&gpioConfig);
    GPIO_SetBit(GPIOB,GPIO_PIN_9);

    /* Rx GPIOB PIN10 */
    GPIO_ConfigStructInit(&gpioConfig);
    gpioConfig.mode = GPIO_MODE_IN;
    gpioConfig.pin = GPIO_PIN_10;
    gpioConfig.pupd = GPIO_PUPD_UP;
    gpioConfig.speed = GPIO_SPEED_50MHz;
    GPIO_Config(GPIOB,&gpioConfig);

    SYSCFG_ConfigEINTLine(SYSCFG_PORT_GPIOB,SYSCFG_PIN_10);

    eintConfig.line = EINT_LINE_10;
    eintConfig.mode = EINT_MODE_INTERRUPT;
    eintConfig.lineCmd = ENABLE;
    eintConfig.trigger = EINT_TRIGGER_FALLING;
    EINT_Config(&eintConfig);

    NVIC_EnableIRQRequest(EINT15_10_IRQn,2,3);
}
```

3.2.2 TMR configuration

The routine uses the TMR4 general-purpose timer. According to the baud rate of 9600 bps, the duration of each bit of data is 104.16us. Configure the single-counting time of 1us and a cycle of 104 and the desired timing effect can be produced.

The TMR4 clock line is APB1, as shown in Figure 7, Figure 8, and Figure 9. According to the *User Manual* and the system clock initialization function, it can be seen that APB1 is obtained through 4 frequency division of AHB1, with a maximum frequency of 42MHz. The TMR clock frequency is 42 * 2=84MHz. According to the calculation formula, set the prescaler factor to 83, and obtain the counter driven clock of 84/(83+1)=1MHz, which means that the single-counting time is 1/1M s=1 us. The time interval for the generation of an interrupt is the automatic reload value 104 * the time required for single counting 1 us=104 us.

Table 1 Instructions and Use of Calculation Formula

Related formulas	Description	Calculation
t = 1 / Baud	Pulse width=1/baud rate	t = 1 / 9600 s ≈ 104 us
CK_CNT = CK_INT/(Division + 1)	Drive counter clock=internal clock/(prescaler+1)	CK_CNK = 84 /（83+1）= 1MHz
t1 = 1 / CK_CNT	Single-counting time=1/drive counter clock	t1 = 1/1M s = 1 us
t2 = t1 * period	Time interval for the generation of interrupts =Time required for single counting * cycle	t2 = t = 1 us * 104 = 104us

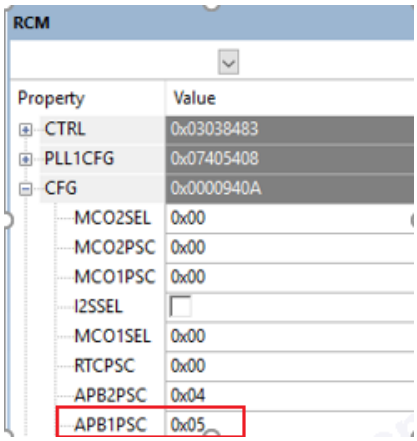

Figure 7 RCM Register Configuration

12:10	APB1PSC	R/W	APB1 Clock Prescaler Factor Configure To control APB low-speed clock division factor. 0xx:HCLK not divided 100:HCLK divided by 2 101:HCLK divided by 4 110:HCLK divided by 8 111:HCLK divided by 16 Caution:PCLK1 not to exceed 45 MHz

Figure 8 APB1PSC Field Description

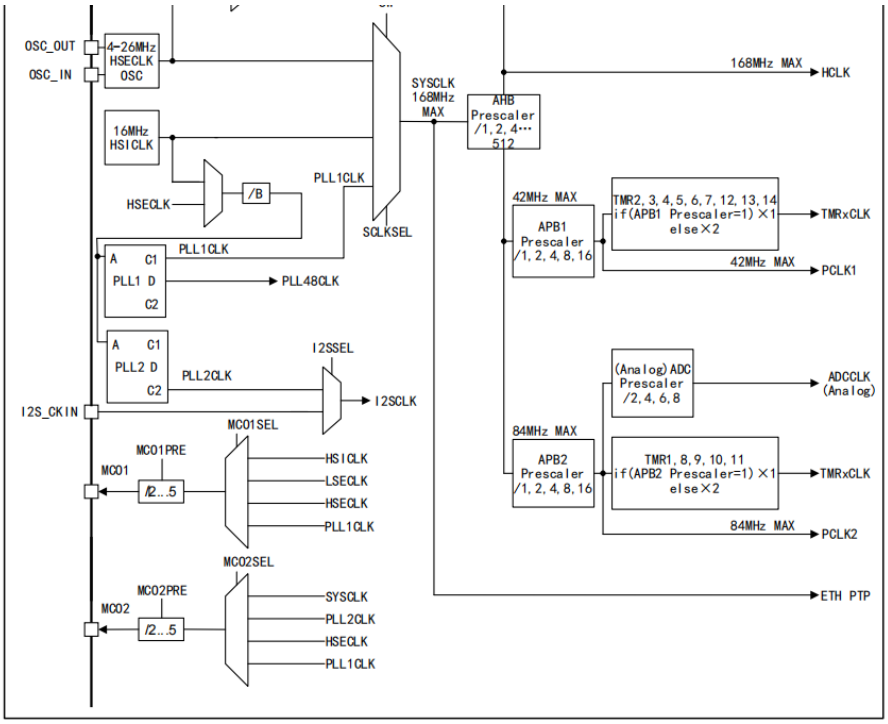

Figure 9 Clock Tree

```
void TMR4_Init(void)
{
    TMR_BaseConfig_T TMRBaseConfig;

    RCM_EnableAPB1PeriphClock(RCM_APB1_PERIPH_TMR4);

    TMRBaseConfig.clockDivision = TMR_CLOCK_DIV_1;
    TMRBaseConfig.countMode = TMR_COUNTER_MODE_UP;
    TMRBaseConfig.division = 84 - 1;
    TMRBaseConfig.period = 104;
    TMR_ConfigTimeBase(TMR4,&TMRBaseConfig);
    TMR_ClearStatusFlag(TMR4,TMR_FLAG_UPDATE);
    TMR_Enable(TMR4);

    /* Configure NVIC_IRQRequest */
    TMR_EnableInterrupt(TMR4,TMR_INT_UPDATE);
    NVIC_EnableIRQRequest(TMR4_IRQn,2,1);

}
```

3.2.3 GPIO simulating serial port transmission

According to Figure 3, it can be seen that the data line is in a high-level state when idle, and the jump from high level to low level indicates the start of communication. Therefore, PB9 is set to 0

before transmission of valid data; after the start bit is transmitted, delay for 104us according to the set baud rate, and use the for loop to start transmitting data bits. It should be noted that the serial port starts transmission from the least significant bit. After data bit transmission is over, set the pin to 1 to simulate the stop bit.

```
void Soft_Usart_TXData(u8 ch)
{
    uint8_t i = 0;
    GPIO_ResetBit(GPIOB,GPIO_PIN_9);
    Delay_us(BaudRate_9600);
    for(i = 0; i < 8; i++)
    {
        if(ch & 0x01)
        {
            GPIO_SetBit(GPIOB,GPIO_PIN_9);
        }
        else
        {
            GPIO_ResetBit(GPIOB,GPIO_PIN_9);
        }
        Delay_us(BaudRate_9600);
        ch = ch >> 1;
    }
    GPIO_SetBit(GPIOB,GPIO_PIN_9);
    Delay_us(BaudRate_9600);
}

void Soft_Usart_Send(u8 *buf,u8 len)
{
    uint8_t t;
    for(t = 0;t < len;t++)
    {
        Soft_Usart_TXData(buf[t]);
        Delay_ms(1);
    }
}
```

3.2.4 GPIO interrupt service function

When a low level is received and the receiving bit exceeds or equals the stop bit, reset the receiving state to the start bit, enable the timer, and start receiving a new frame of data. If the

receiving state does not reach the stop bit, it indicates that the falling edge is generated by valid data, and does not indicate the arrival of new data, and it can directly jump out of the interrupt.

```
void EINT15_10_IRQHandler(void)
{
    if(EINT_ReadStatusFlag(EINT_LINE_10) != RESET)
    {
        if(Soft_Usart_RXData() == 0)
        {
          if(recvState >= COM_STOP_BIT)
           {
              recvState = COM_START_BIT;
              TMR_Enable(TMR4);
              flag = 1;
           }
        }
       EINT_ClearStatusFlag(EINT_LINE_10);
    }

}
```

3.2.5 TMR interrupt service function

After the timer is enabled, an interrupt will be triggered every 104us. Enter the interrupt service function and complete the bitwise receiving of data. As the data is transmitted from the least significant bit during serial port transmission, after one-bit data is received, it needs to be shifted and placed in the correct position to ensure the accuracy of the received data. When the stop bit is read, the received value is stored in the buffer.

```
void TMR4_IRQHandler(void)
{
    if(TMR_ReadStatusFlag(TMR4,TMR_FLAG_UPDATE) != RESET)
    {
        TMR_ClearStatusFlag(TMR4,TMR_FLAG_UPDATE);
        recvState++;
        if(recvState >= COM_STOP_BIT)
        {
            TMR_Disable(TMR4);
            USART_buf[len++] = recvData;
            return;
        }
        if(Soft_Usart_RXData())
        {
          recvData |= (1 << (recvState - 1));
        }
        else
        {
          recvData &= ~(1 << (recvState - 1));
        }
    }
}
```

3.2.6 Main function

After initialization is completed, continue to transmit the received data.

```
int main(void)
{
    NVIC_ConfigPriorityGroup(NVIC_PRIORITY_GROUP_2);
    Delay_Init();
    Soft_Usart_Init();
    TMR4_Init();
    while (1)
    {
        if(len > 10)
        {
          len = 0;
          Soft_Usart_Send(USART_buf,11);
        }
    }
}
```

3.3 Experimental Phenomena

Open the serial debugging assistant, set the baud rate, start bit, data bit, and stop bit, then transmit the characters and echo them successfully.

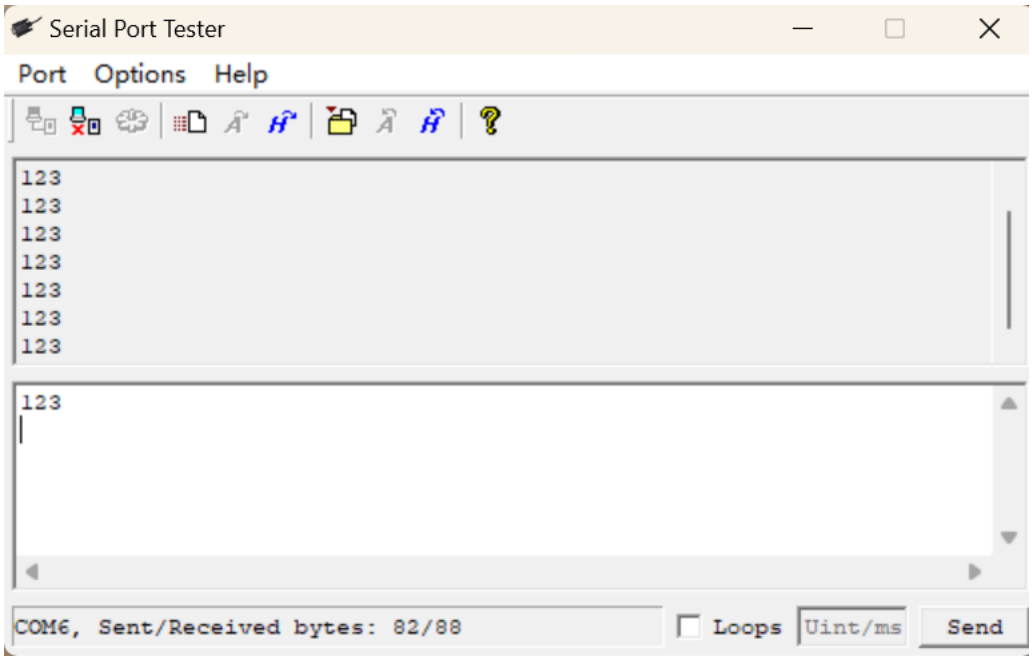

Figure 10 Experimental Phenomena

4 Revision history

Table 2 Document Revision History

Date	Version	Revision History
August 22, 2023	1.0	New edition

Statement

This manual is formulated and published by Zhuhai Geehy Semiconductor Co., Ltd. (hereinafter referred to as "Geehy"). The contents in this manual are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to correct and modify this manual at any time. Please read this manual carefully before using the product. Once you use the product, it means that you (hereinafter referred to as the "users") have known and accepted all the contents of this manual. Users shall use the product in accordance with relevant laws and regulations and the requirements of this manual.

1. Ownership of rights

This manual can only be used in combination with chip products and software products of corresponding models provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this manual for any reason or in any form.

The "Geehy" or "Geehy" words or graphics with "®" or "TM" in this manual are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No intellectual property license

Geehy owns all rights, ownership and intellectual property rights involved in this manual.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale and distribution of Geehy products and this manual.

If any third party’s products, services or intellectual property are involved in this manual, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property, unless otherwise agreed in sales order or sales contract of Geehy.

3. Version update

Users can obtain the latest manual of the corresponding products when ordering Geehy products.

If the contents in this manual are inconsistent with Geehy products, the agreement in Geehy sales order or sales contract shall prevail.

4. Information reliability

The relevant data in this manual are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this manual. The relevant data in this manual are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual and the matching Geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countries of the product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and promise to abide by all applicable laws and regulations on the export and re-export of Geehy products and/or technologies and direct products.

6. Disclaimer

This manual is provided by Geehy "as is". To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual "as is" shall not be liable for damages, including any general damages, special direct, indirect or collateral damages arising from the use or no use of the information in this manual (including without limitation data loss or inaccuracy, or losses suffered by users or third parties).

8. Scope of application

The information in this manual replaces the information provided in all previous versions of the manual.