Serial port Transmitting and Receiving experiment

Technical Email: alinx@aithtech.com Sales Email: rachel.zhou@alinx.com

1 Document Introduction

This article mainly explains how to write the transceiver program of FPGA serial communication, and uses the state machine in the program, which is an important experiment to learn the state machine.

2 Experiment Environment

- ALINX Brand FPGA Development Board (AXKU040 FPGA Development Board)
- Serial debugging assistant

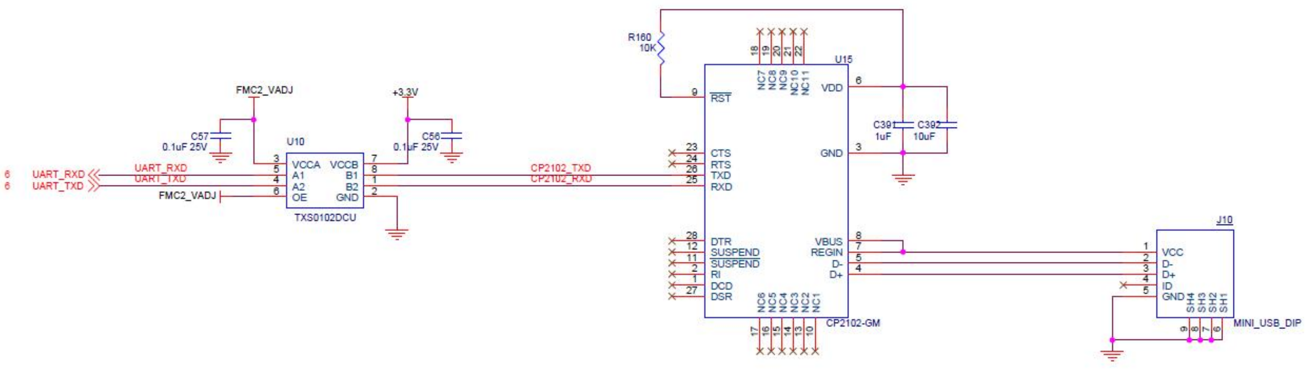
3 Experiment Principle

3.1 Introduction to serial communication

The serial port described in this document refers to asynchronous serial communication, and asynchronous serial refers to UART (Universal Asynchronous Receiver/Transmitter), universal asynchronous receiving/sending. The UART is a chip that turns the parallel input into a serial output and is usually integrated on the motherboard. The UART includes a TTL level serial port and an RS232 level serial port. The TTL level is 3.3V, and RS232 is a negative logic level. It defines +5~+12V as low level, while -12~-5V is high level. MDS2710, MDS SD4, EL805, etc. are RS232 interface. The EL806 has a TTL interface.

The serial interface includes RS-232-C, RS-422, RS485, etc. according to electrical standards and protocols. The RS-232-C, RS-422, and RS-485 standards only specify the electrical characteristics of the interface and do not involve connectors, cables, or protocols.

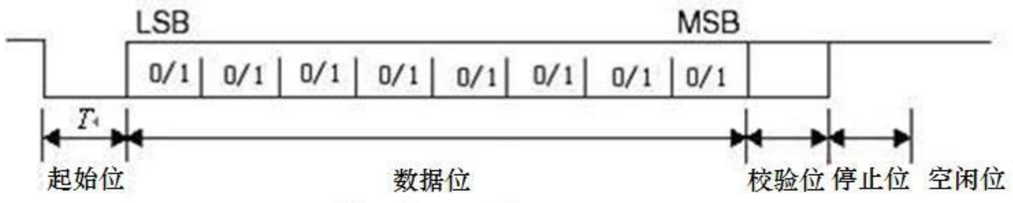
The serial communication of the FPGA development board is through the USB to serial port mode, which mainly solves the problem that many people do not have a serial port interface, so the electrical protocol standard is not involved here, and the usage is similar to the TTL level serial port. The FPGA chip uses two IO ports and is connected to the USB to serial port chip CP2102.



USB to serial port part of the AXKU040 FPGA development board

3.2 Asynchronous serial communication protocol

The message frame begins with a low start bit followed by 7 or 8 data bits, an available parity bit and one or more high stop bits. When the receiver finds the start bit it knows that the data is ready to be sent and tries to synchronize with the transmitter clock frequency. If parity is selected, the UART adds a parity bit after the data bit. The parity bit can be used to help with error checking. During reception, the UART removes the start and end bits from the message frame, performs parity on the incoming bytes, and converts the data bytes from serial to parallel. The UART transmission timing is shown below:



It can be seen from the waveform that the start bit is low, and the stop bit and the idle bit are both high, that is to say, it is high when there is no data transmission. With this feature we can receive data accurately, and when a falling edge event occurs, we believe that a data transfer will take place.

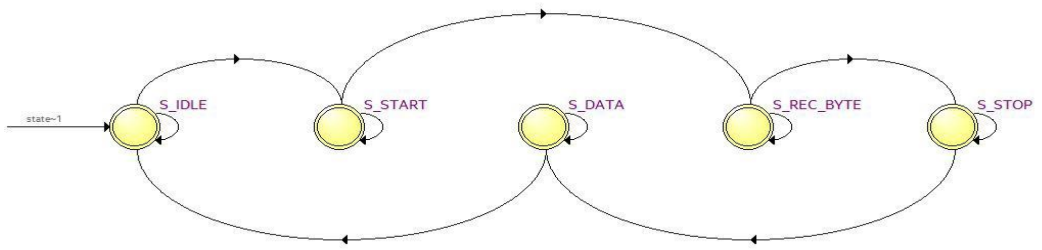
3.3 About baudrate

The common serial communication baud rate is 2400, 9600, 115200, etc. The transmit and receive baud rates must be consistent in order to communicate correctly. The baud rate refers to the maximum number of data bits transmitted in 1 second, including the start bit, data bit, parity bit, and stop bit. If the communication baud rate is set to 9600, then the length of one data bit is 1/9600 seconds.

4 Programming

4.1 Receiving moduledesign

The serial port receiving module is a parameterized configurable module. The parameter “CLK_FRE” defines the system clock frequency of the receiving module, the unit is Mhz, and the parameter “BAUD_RATE” is the baud rate. The receiving state machine state transition diagram is as follows:



The "S_IDLE" state is idle. After power-on, it enters "S_IDLE". If the signal "rx_pin" has a falling edge, we consider it the start bit of the serial port and enter the state "S_START". After the end of a “BIT” time start bit, enter the data bit reception status "S_REC_BYTE". In this experiment, the data bit is 8 bits. After receiving, the system enters the "S_STOP" state. In the "S_STOP", there is no waiting for a BIT cycle, *and only half of the BIT time is waited*. This is because if you wait for one cycle, you may miss the start bit judgment of the next data, and finally enter the "S_DATA" state, and send the received data to other modules. In this module we mention that in order to satisfy the sampling theorem, each data is sampled at the midpoint of the baud rate counter when accepting data to avoid data errors:

```
//receive serial data bit data
always@(posedge clk or negedge rst_n)
begin
        if(rst_n == 1'b0)
                rx_bits <= 8'd0;
        else if(state == S_REC_BYTE && cycle_cnt == CYCLE/2 - 1)
                rx_bits[bit_cnt] <= rx_pin;
        else
                rx_bits <= rx_bits;
end
```

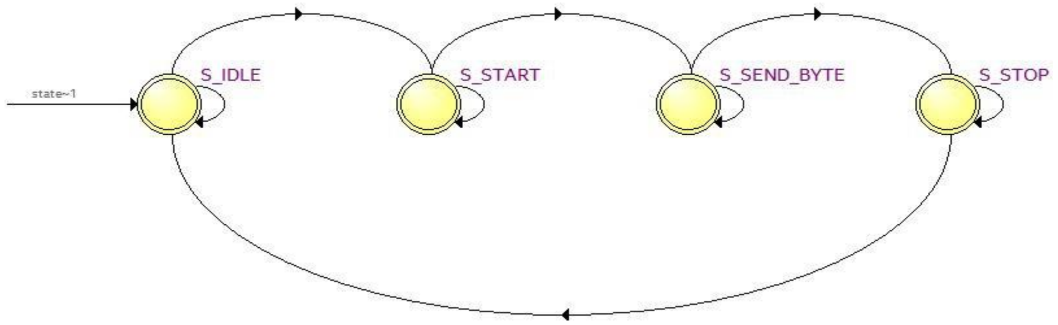
Note: ***This experiment does not have a parity bit***

Signal Name	Direction	Width (bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
rx_data	out	8	Received serial data (8-bit data)
rx_data_valid	out	1	The received serial port data is valid (high effective)
rx_data_ready	in	1	Can receive data, send data when “rx_data_ready” and “rx_data_valid” are both high
rx_pin	in	1	Serial port receiving data input

Serial port receiving module port

4.2 Transmit ing module design

The transmit mode design is similar to the receive module, and the state machine is also used. The state transition diagram is as follows:



After power-on, enter the "S_IDLE" idle state. If there is a transmission request, enter the transmission start bit status "S_START". After the start bit transmission is completed, the transmission data bit status "S_SEND_BYTE" is entered. After the data bit is sent, the transmission stop bit status is entered. "S_STOP", the stop bit transmission is completed and then enters the idle state. In the data transmission module, the data written from the top-level module is directly passed to the register 'tx_reg', and the data transfer is performed under the conditional transition of the state machine by the 'tx_reg' register emulation serial transmission protocol:

```
always@(posedge clk or negedge rst_n)
begin
        if(rst_n == 1'b0)
                tx_reg <= 1'b1;
        else
                case(state)
                        S_IDLE,S_STOP:
                                tx_reg <= 1'b1;
                        S_START:
                                tx_reg <= 1'b0;
                        S_SEND_BYTE:
                                tx_reg <= tx_data_latch[bit_cnt];
                        default:
                                tx_reg <= 1'b1;
                endcase
end
```

Signal Name	Direction	Width (bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
tx_data	in	8	Serial data to be sent (8-bit data)
tx_data_valid	in	1	The serial port data transmit is valid (high effective)
tx_data_ready	out	1	Data can be transmit, and data is transmit when both "tx_data_ready" and "tx_data_valid" are high
tx_pin	out	1	Serial port transmitting data input

Serial port transmitting module port

4.3 Test Program

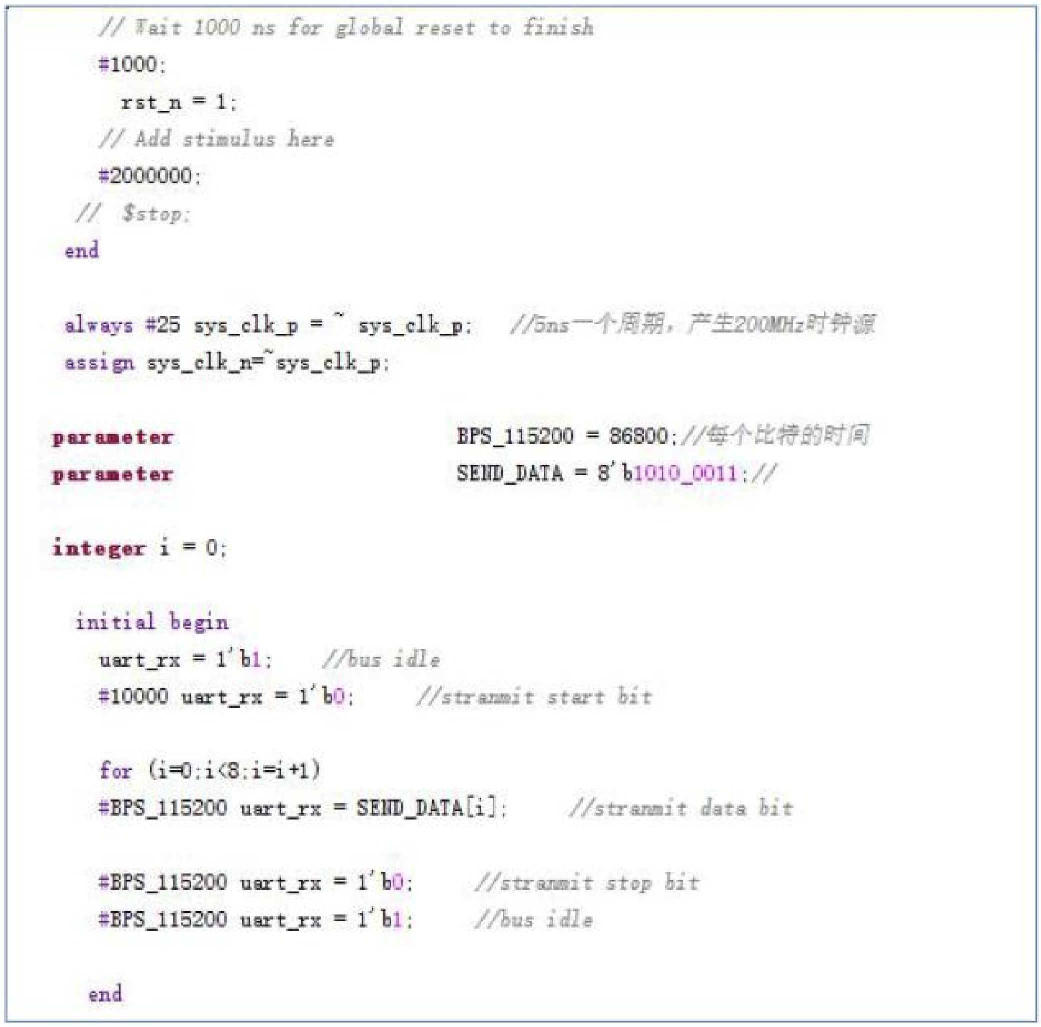
The test program is that the FPGA sends a "HELLO ALINX\r\n" to the serial port for 1 second. If the serial port data is received during the non-transmission period, the received data is sent directly to the sending module and then returned. "\r\n", here is consistent with the C language, both are “Enter” and change Row.

The test program instantiates the transmitting module and the receiving module separately, and passes the parameters in, and the baud rate is set to 115200.

```
always@(posedge sys_clk or negedge rst_n)
begin
    if(rst_n == 1'b0)
    begin
        wait_cnt <= 32'd0;
        tx_data <= 8'd0;
        state <= IDLE;
        tx_cnt <= 8'd0;
        tx_data_valid <= 1'b0;
    end
    else
    case(state)
        IDLE:
            state <= SEND;
        SEND:
        begin
            wait_cnt <= 32'd0;
            tx_data <= tx_str;

            if(tx_data_valid == 1'b1 && tx_data_ready == 1'b1 && tx_cnt
< 8'd12)//Send 12 bytes data
            begin
                tx_cnt <= tx_cnt + 8'd1; //Send data counter
            end
            else if(tx_data_valid && tx_data_ready)//last byte sent is complete
            begin
                tx_cnt <= 8'd0;
                tx_data_valid <= 1'b0;
                state <= WAIT;
            end
            else if(~tx_data_valid)
            begin
                tx_data_valid <= 1'b1;
            end
        end
        WAIT:
        begin
            wait_cnt <= wait_cnt + 32'd1;

            if(rx_data_valid == 1'b1)
            begin
                tx_data_valid <= 1'b1;
                tx_data <= rx_data;    // send uart received data
            end
            else if(tx_data_valid && tx_data_ready)
            begin
                tx_data_valid <= 1'b0;
            end
            else if(wait_cnt >= CLK_FRE * 1000000)  // wait for 1 second
                state <= SEND;
        end
        default:
            state <= IDLE;
    endcase
end

//combinational logic
//Send "HELLO ALINX\r\n"
always@(*)
begin
    case(tx_cnt)
        8'd0 :  tx_str <= "H";
        8'd1 :  tx_str <= "E";
        8'd2 :  tx_str <= "L";
        8'd3 :  tx_str <= "L";
        8'd4 :  tx_str <= "O";
        8'd5 :  tx_str <= " ";
        8'd6 :  tx_str <= "A";
        8'd7 :  tx_str <= "L";
        8'd8 :  tx_str <= "I";
        8'd9 :  tx_str <= "N";
        8'd10:  tx_str <= "X";
        8'd11:  tx_str <= "\r";
        8'd12:  tx_str <= "\n";
        default:tx_str <= 8'd0;
    endcase
end

uart_rx#
(
    .CLK_FRE(CLK_FRE),
    .BAUD_RATE(115200)
) uart_rx_inst
(
    .clk                        (sys_clk                    ),
    .rst_n                      (rst_n                      ),
    .rx_data                    (rx_data                    ),
    .rx_data_valid              (rx_data_valid              ),
    .rx_data_ready              (rx_data_ready              ),
    .rx_pin                     (uart_rx                    )
);

uart_tx#
(
    .CLK_FRE(CLK_FRE),
    .BAUD_RATE(115200)
) uart_tx_inst
(
    .clk                        (sys_clk                    ),
    .rst_n                      (rst_n                      ),
    .tx_data                    (tx_data                    ),
    .tx_data_valid              (tx_data_valid              ),
    .tx_data_ready              (tx_data_ready              ),
    .tx_pin                     (uart_tx                    )
);
```

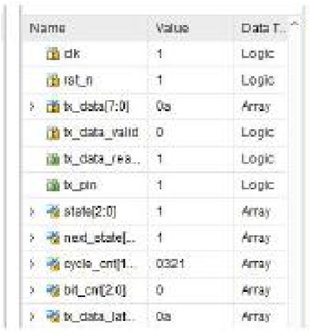
5 Simulation

Here we have added a stimulus program vtf_uart_test.v to simulate uart serial reception. Here transmits 0xa3 data to uart_rx of the serial module, each bit of data is transmitted at 115200 baud rate, 1 bit start, 8 bits 1 start bit, 8 data bits and 1 stop bit.

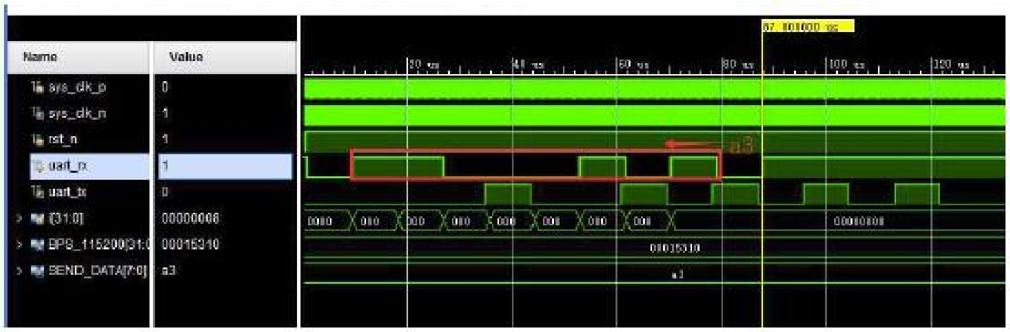


```
        // Wait 1000 ns for global reset to finish
        #1000;
          rst_n = 1;
        // Add stimulus here
        #2000000;
      //  $stop;
    end

    always #25 sys_clk_p = ~ sys_clk_p;   //5ns一个周期，产生200MHz时钟源
    assign sys_clk_n=~sys_clk_p;

   parameter                    BPS_115200 = 86800;//每个比特的时间
   parameter                    SEND_DATA = 8'b1010_0011;//

   integer i = 0;

     initial begin
        uart_rx = 1'b1;     //bus idle
        #10000 uart_rx = 1'b0;      //stranmit start bit

        for (i=0;i<8;i=i+1)
        #BPS_115200 uart_rx = SEND_DATA[i];      //stranmit data bit

        #BPS_115200 uart_rx = 1'b0;      //stranmit stop bit
        #BPS_115200 uart_rx = 1'b1;      //bus idle

       end
```

The result of the simulation is as follows: when the program receives 8-bit data rx_data[7:0] data bits a3, after receiving the data, uart_tx will keep transmitting out the received data.



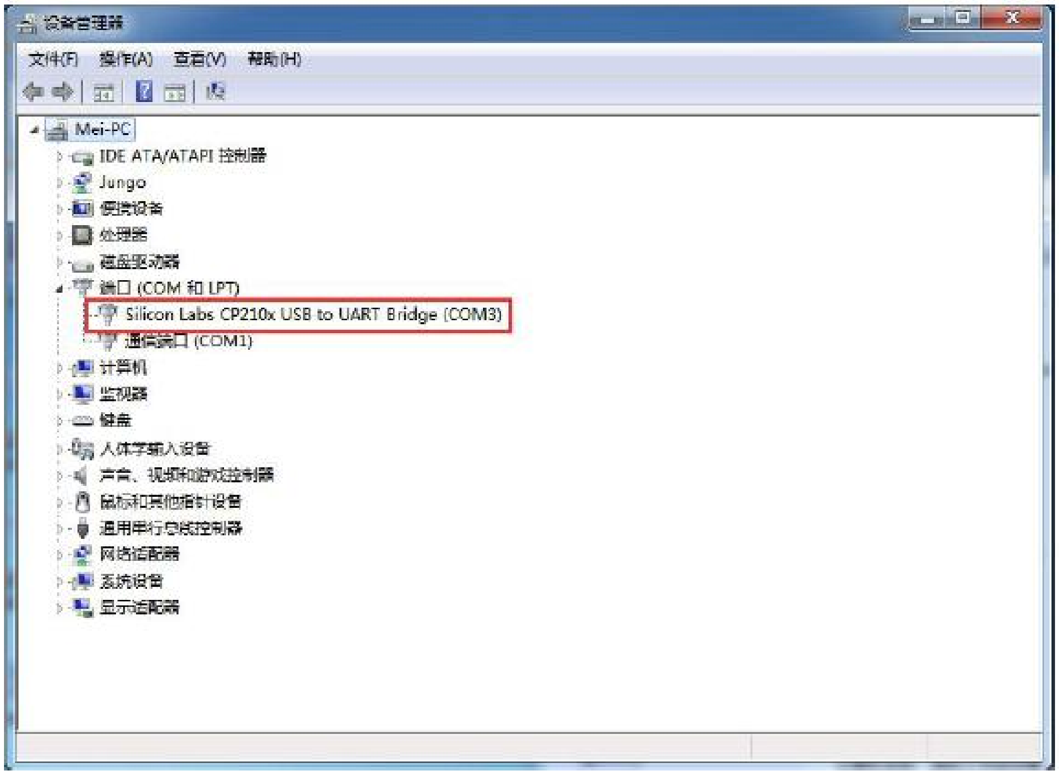
Name	Value	Data T...
clk	1	Logic
rst_n	1	Logic
tx_data[7:0]	0a	Array
tx_data_valid	0	Logic
tx_data_rea...	1	Logic
tx_pin	1	Logic
state[2:0]	1	Array
next_state[...	1	Array
cycle_cnt[1...	0321	Array
bit_cnt[2:0]	0	Array
tx_data_lat...	0a	Array



Name	Value
sys_clk_p	0
sys_clk_n	1
rst_n	1
uart_rx	1
uart_tx	0
i[31:0]	00000008
BPS_115200[31:0]	00015310
SEND_DATA[7:0]	a3

6 Experiment Testing

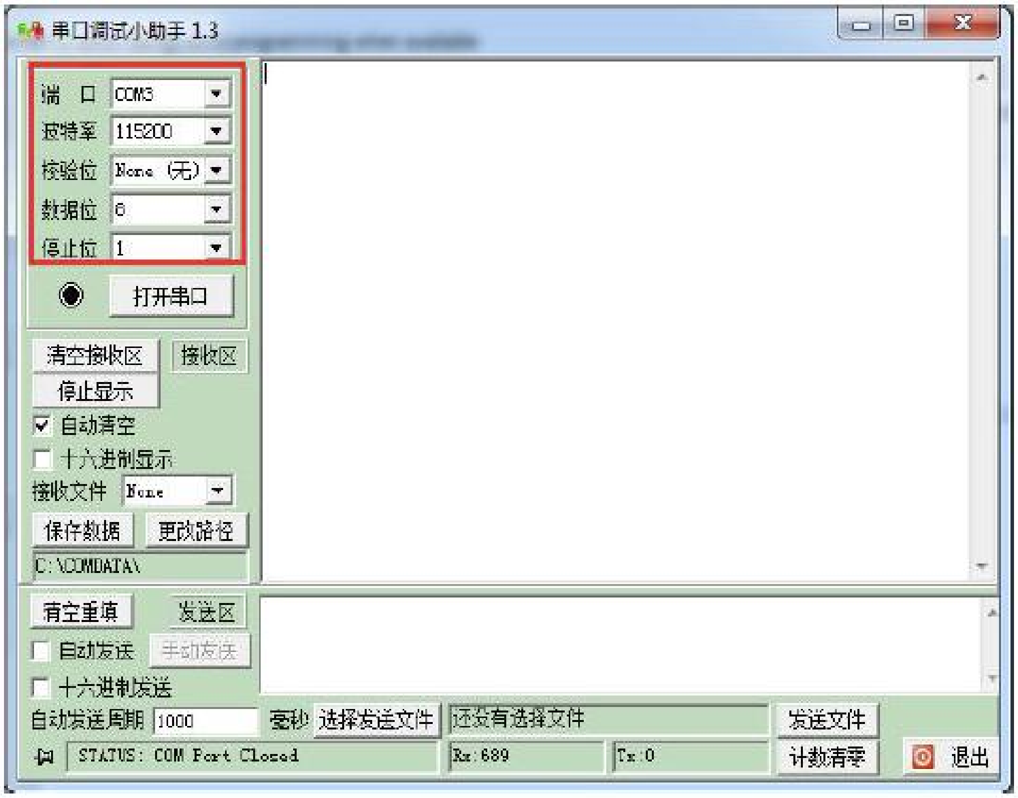
As the serial port of the FPGA development board uses a USB to serial chip, the serial driver should be installed first. The correct installation status of the driver is shown in the figure below (of course, the USB serial port should be connected to the computer). If not correctly connected, please refer to the appendix of this article "serial port driver installation".



Mei-PC
- IDE ATA/ATAPI 控制器
- Jungo
- 便携设备
- 处理器
- 磁盘驱动器
- 端口 (COM 和 LPT)
  - Silicon Labs CP210x USB to UART Bridge (COM3)
  - 通信端口 (COM1)
- 计算机
- 监视器
- 键盘
- 人体学输入设备
- 声音、视频和游戏控制器
- 鼠标和其他指针设备
- 通用串行总线控制器
- 网络适配器
- 系统设备
- 显示适配器

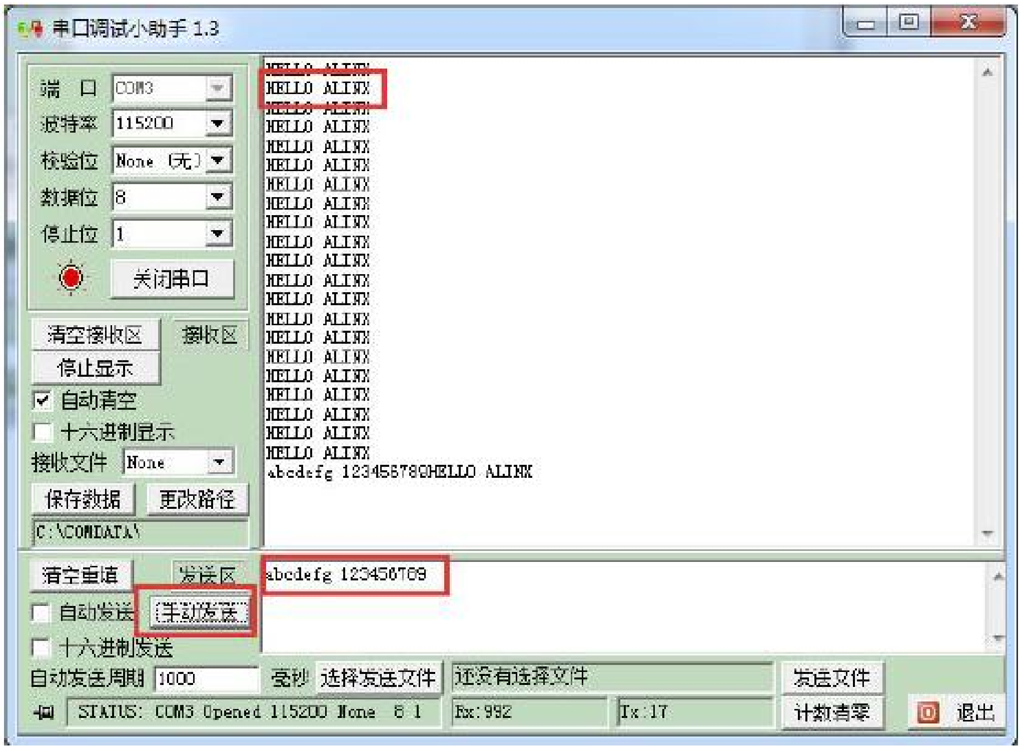
As you can see from the figure, the serial port number assigned by the system to the serial port is "COM3", the allocation of the serial port number is done by the system, the automatic allocation of each computer may be different, here is "COM3", use the serial port number to choose according to their own allocation.

Open the serial port debugging, select "COM3" (according to your choice), set the baud rate 115200, check bit select None, data bits select 8, stop bits select 1, and then click "open serial port". If you can't find this small software use windows search Search function, search for "serial debugging" in the information folder given by ALINX.



端 口 COM3
波特率 115200
校验位 None (无)
数据位 8
停止位 1
接收文件 None
C:\COMDATA\
自动发送周期 1000 毫秒
还没有选择文件
STATUS: COM Port Closed
Rx:689
Tx:0

After opening the serial port, you can receive "HELLO ALINX" every second, enter the text you want to transmit in the input box of the transmitting area, click "Transmit Manually", and you can see the characters you have transmit. You can see the characters you have transmitted by clicking "Transmit Manually".



端 口 COM3
波特率 115200
校验位 None (无)
数据位 8
停止位 1
接收文件 None
C:\COMDATA\

```
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
HELLO ALINX
abcdefg 123456789HELLO ALINX
```

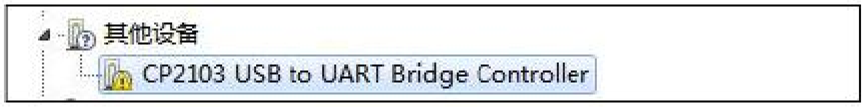
abcdefg 123456789

自动发送周期 1000 毫秒
还没有选择文件
STATUS: COM3 Opened 115200 None 8 1
Rx:992
Tx:17

7 Appendix

Serial port driver installation

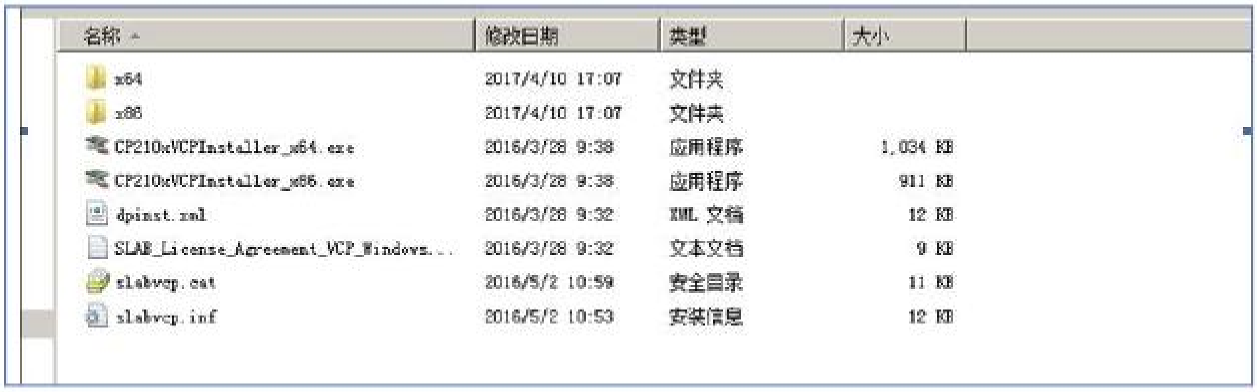
Without driver installation, the following situation will appear under the device manager after plugging in the usb to serial port.



其他设备

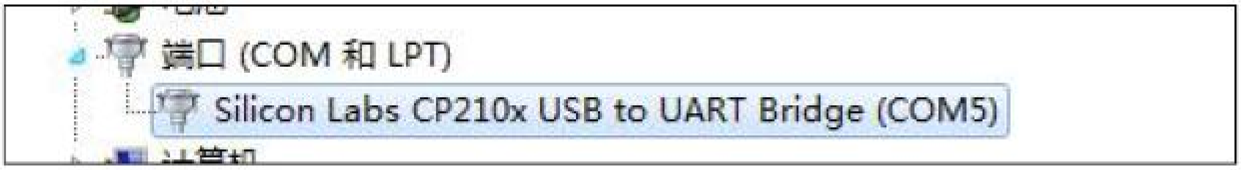
CP2103 USB to UART Bridge Controller

The driver installation file can be found in the "Software Tools & Drivers \ USB to Serial Driver" directory in the information we provide. If the operating system is 32-bit user double-click **CP210x_VCPInstaller_x86.exe** to start the installation; if the operating system is 64-bit user double-click **CP210x_VCPInstaller_x64.exe** to start the installation.



名称	修改日期	类型	大小
x64	2017/4/10 17:07	文件夹	
x86	2017/4/10 17:07	文件夹	
CP210xVCPInstaller_x64.exe	2016/3/28 9:38	应用程序	1,034 KB
CP210xVCPInstaller_x86.exe	2016/3/28 9:38	应用程序	911 KB
dpinst.xml	2016/3/28 9:32	XML 文档	12 KB
SLAB_License_Agreement_VCP_Windows...	2016/3/28 9:32	文本文档	9 KB
slabvcp.cat	2016/5/2 10:59	安全目录	11 KB
slabvcp.inf	2016/5/2 10:53	安装信息	12 KB

After the driver is successfully installed, open "Device Manager", open "Ports (**COM** and **LPT**)" and the corresponding **COM number** will appear. The assigned number is determined by the system.



端口 (COM 和 LPT)

Silicon Labs CP210x USB to UART Bridge (COM5)