1 基于 LWIP 的网络数据与串口交互

工程源码	|- ACZ702 v2.0 开发板 |-- UART1_and_UART0_LWIP
相关视频课程	无

章节导读

本节将介绍如何使用 LWIP 和 ACZ702 开发板实现通信，让大家初步了解 TCP 的使用。进一步地，我们还将深入介绍如何使用 PL 端和 PS 端的串口转发网络数据，以及通过网络上位机来调整数据流向，从而扩大应用场景的可能性。通过这一系列步骤，希望帮助读者对 ACZ702 开发板的通信使用有更深入的理解。

1.1 LWIP 通信

lwIP（lightweight IP）是一个面向嵌入式系统的轻量级网络协议栈，主要用于为支持 TCP/IP 协议的应用提供网络通信能力。lwIP 提供了丰富的网络协议和功能，包括 IP、ICMP、UDP、TCP 等协议，以及 DNS、SNMP、DHCP 等服务。

更加详细资料，请看文档：“基于 ZYNQ7020 的 LWIP+RS485 回环通信”与“LwIP 的官方模板使用”

1.2 UartLite IP 核

UARTLite 是 Xilinx 提供的一款简化的通用异步收发器(Universal Asynchronous Receiver Transmitter) IP 核，它设计用于低速率和低资源要求的应用。

更加详细资料，请阅读《基于 C 编程的 Zynq 裸机程序设计与应用教程》中关于 AXI UartLite 核串口中断章节，里面详细介绍了该核的使用教程。

1.3 硬件逻辑系统设计

1.3.1 添加 IP 核

本次设计我们使用到 Zynq 处理系统 IP 核、AXI Uartlite IP 核。

1.3.2 IP 核配置

IP 核添加完成后，通常还需要配置后才能在硬件逻辑系统内正常工作。

首先配置 Zynq 处理系统 IP 核，参考如下：

(1) DDR

首先配置 Zynq IP 核，打开 DDR 配置界面，设置 DDR 型号。

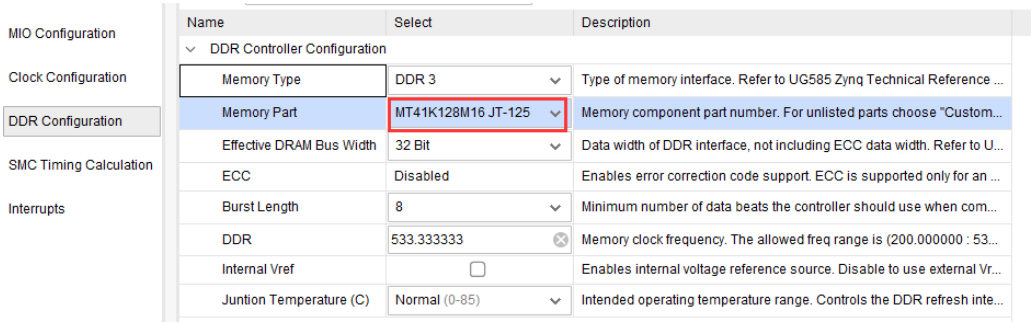

图 1-1 DDR 型号

(2) Clock Configuration

AXI UartLite 核的工作时钟应该低于 120MHz，因此这里我们打开 Clock Configuration 界面，配置输出到 PL 的时钟为 100MHz。

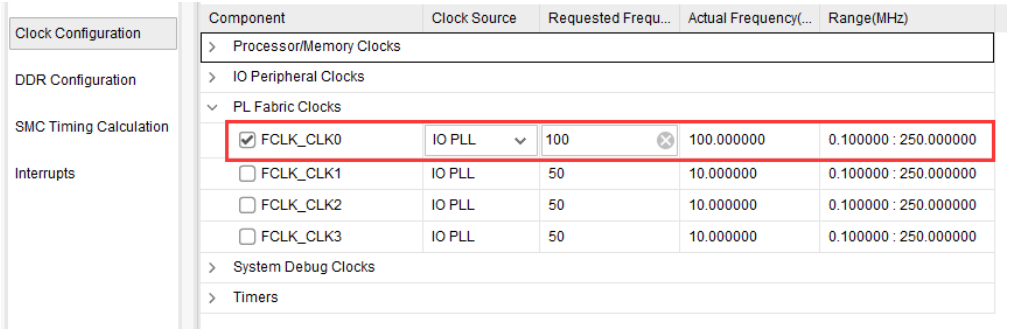

图 1-2 Clock Configuration

(3) Interrupts

PL 侧的中断需要使能对应中断端口才能连接到 PS，本次设计所产生的中断类型属于共享外设中断的 PL 中断。因此，如图 1-3 所示，使能 IRQ_F2P[15:0]端口。

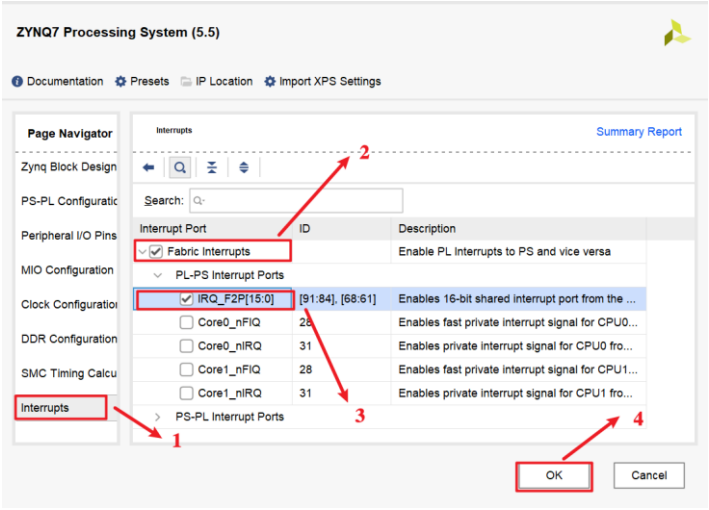

图 1-3 中断端口

(4) PS_UART

在通信过程中，会通过 PS 串口打印相关数据，所以这里我们还需要使能 PS 端串口外设，ACZ702 开发板上 PS 侧串口引脚对应 MIO48...49，外设使能如图 1-4 所示：

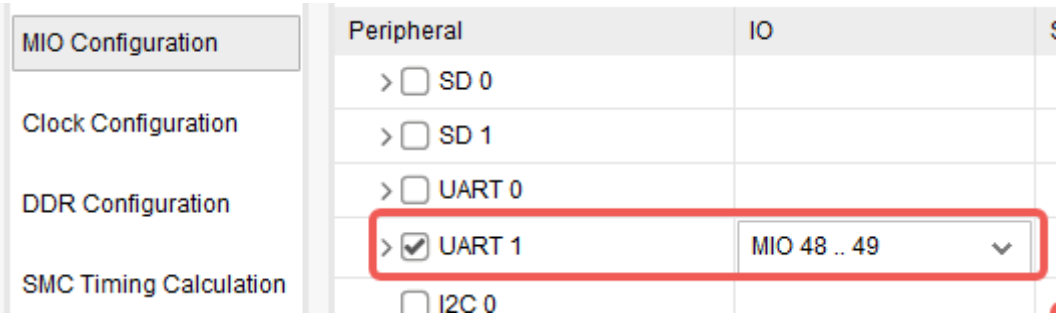

图 1-4 PS_UART 配置

(5) LWIP

按照下图所示，选择 ENET 0 和下方的 MDIO，并在 IO 列中，并设置 ENET 0 的 IO 为 MIO16…27、MDIO 的 IO 为 MIO52…53。

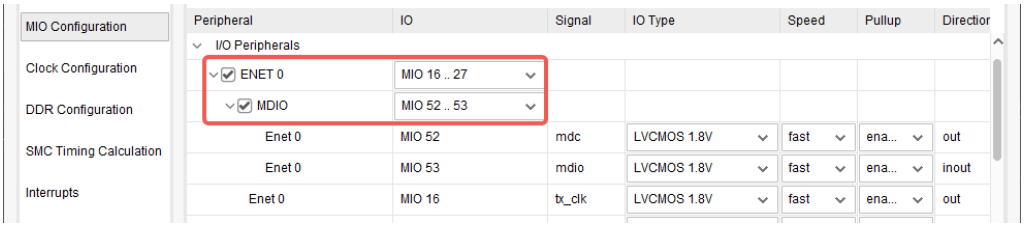

图 1-5 LWIP 配置

最后对 Speed 列，选择速度为 fast（也可以保持默认），点击 OK。

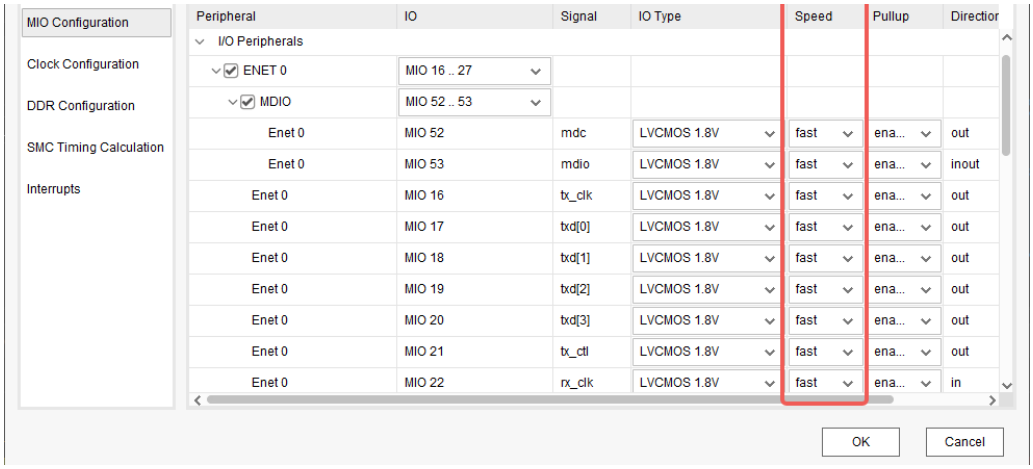

图 1-6 Speed 配置

在 Zynq IP 核配置完成后，继续配置 AXI Uartlite IP 核；其中，时钟保持默认的即可，波特率为 115200，数据位设置为 8，奇偶校验位设置为无。

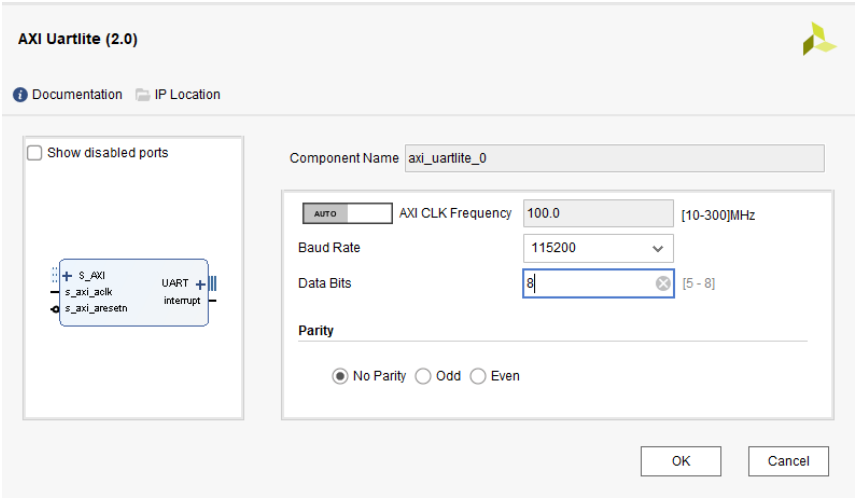

图 1-7 配置 AXI Uartlite IP 核

1.3.3 导出引脚

接下来我们需要将 IP 核的引脚导出，点击上方的蓝色小字“Run Block Automation”，让系统自动帮我们导出引脚。此时软件会弹出如图所示弹窗，勾选“ALL Automation”，其余部分保持默认，点击 OK，软件便会帮我们导出。

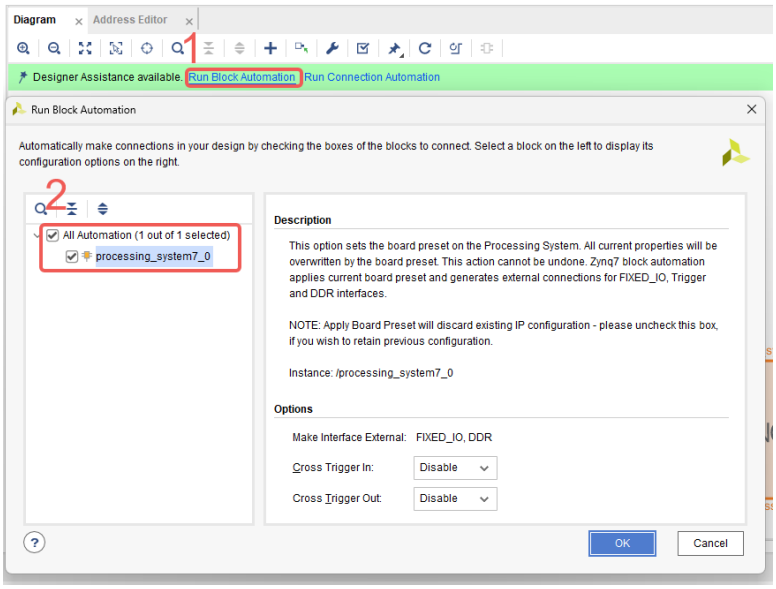

图 1-8 引脚导出

1.3.4 端口连接

首先，将 FCLK_CLK0 和 M_AXI_GPIO_ACLK 连接。点击“Run Connection Automation”，并勾选弹窗中全部对象，点击 OK，等待自动连接。

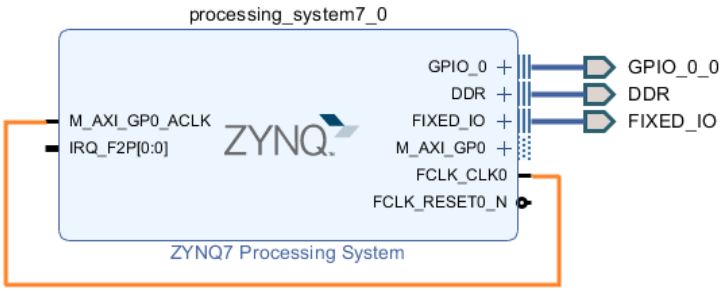

图 1-9 FCLK_CLK0

继续点击 Regenerate Layout ，重新生成布局。

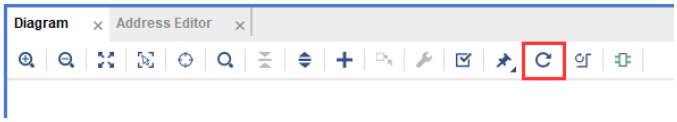

图 1-10 重新布局

最后，验证设计，出现 successful，说明无错误。

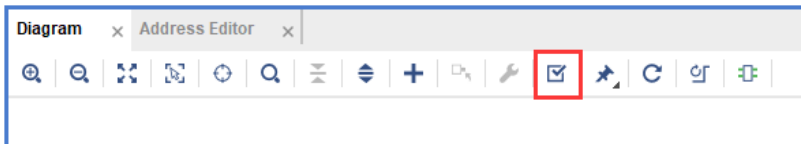

图 1-11 验证设计

1.3.5 生成封装

点击 sources 资源栏下我们创建的 system 模块设计，单击右键，在展开的功能中选择“Generate Output Products...”生成输出。

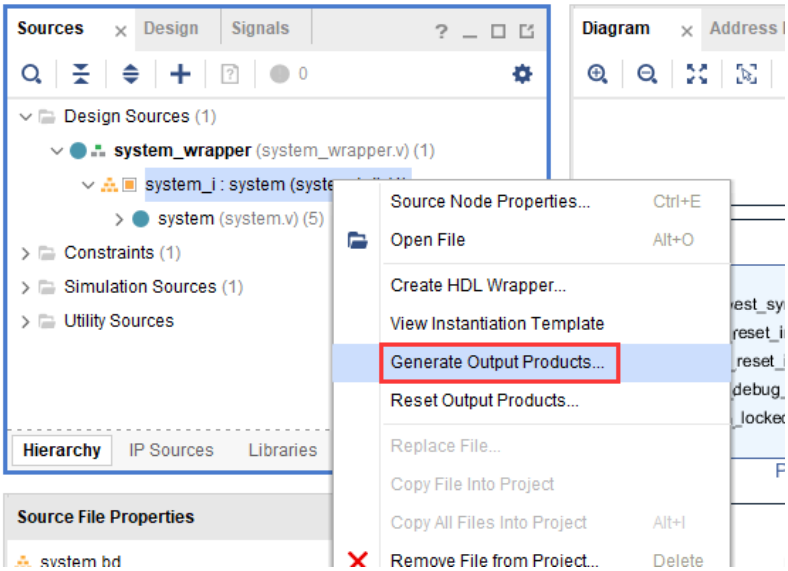

图 1-12 Generate Output Products

接下来软件会弹出生成输出前的设置界面。在合成选项栏直接选择“Out Of context per IP”即可，下方的“Number of jobs”选项选择最大值。设置完成后点击“Generate”开始生成。

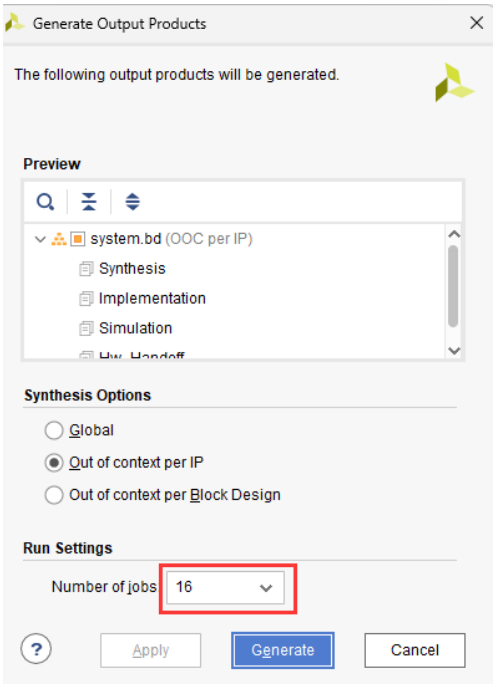

图 1-13 设置输出

继续右键单击 system 模块，在展开的功能中选择“Create HDL Wrapper...”创建 HDL 封装。

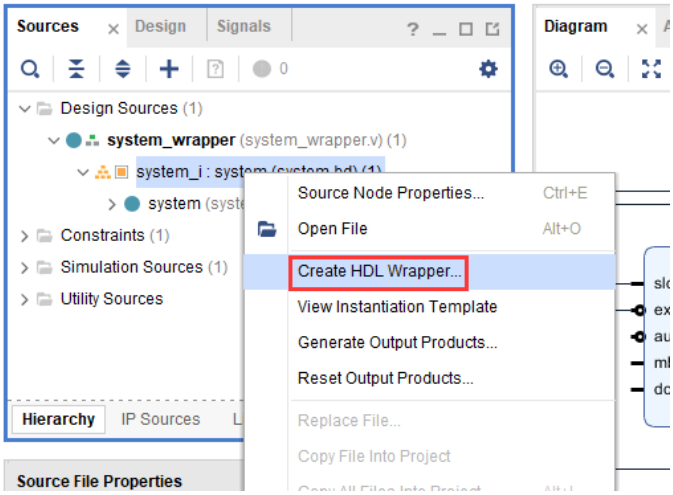

图 1-14 创建 HDL 封装

1.3.6 管脚约束

根据下图设置“Package Pin”以及“I/O std”。

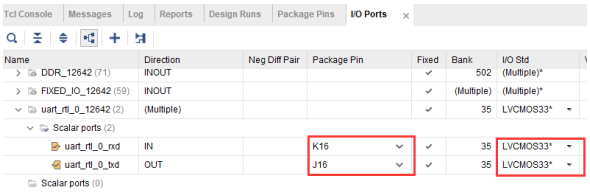

图 1-15 管脚约束

完成分配后使用快捷键 Ctrl+S 对约束文件进行保存。然后生成比特流并导出到 SDK 中即可开始本次工程的软件设计。

1.4 CPU 软件程序设计

新建工程，然后为工程选择模板 lwIP Echo Server ，此时右边窗口便会出现当前工程模板的描述，点击 Finish 完成工程创建。

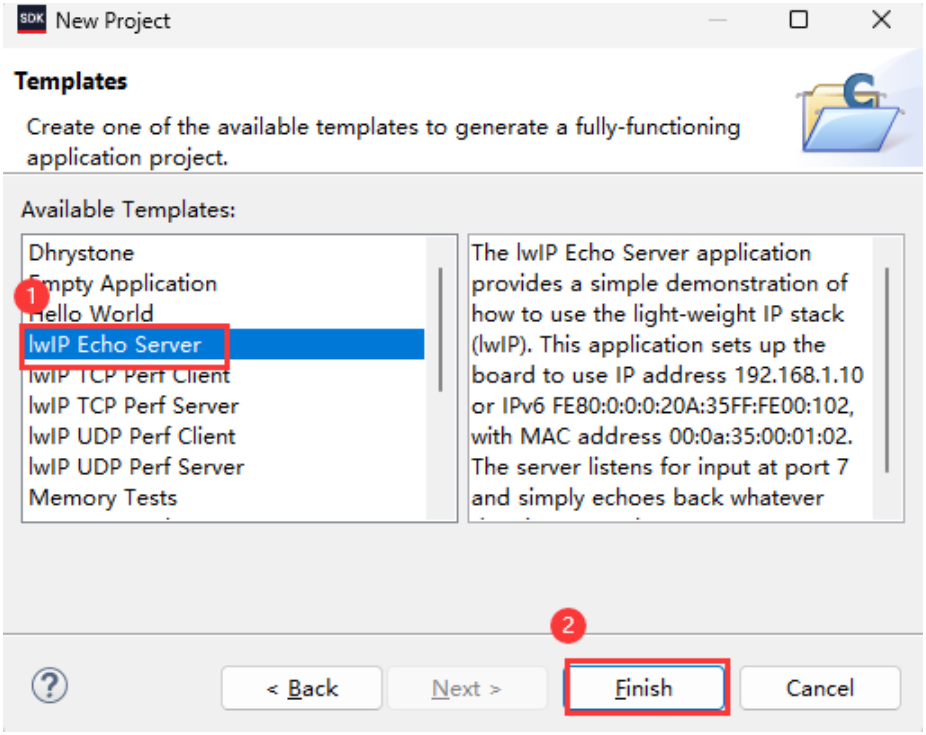

图 1-16 lwIP Echo Server 模板

1.4.1 添加应用库

创建完 SDK 工程，然后为工程添加应用文件夹。

(1) ACZ702_Lib 文件夹

打开提供的资料，路径为：*小梅哥 ACZ702 型 Zynq 开发板资料\盘 A_ACZ702 开发板标准配套资料\02_设计实例\03_【裸机例程】基于 C 编程的*

Zynq 裸机例程\ACZ702_Lib。

将其中“AXI_UARTLite”、“PS_UART”、“SCU” 这 3 个文件夹复制添加到文件地址“*...\工程名.sdk\Uart0andUart1_lwip\ACZ702_Lib*”中。

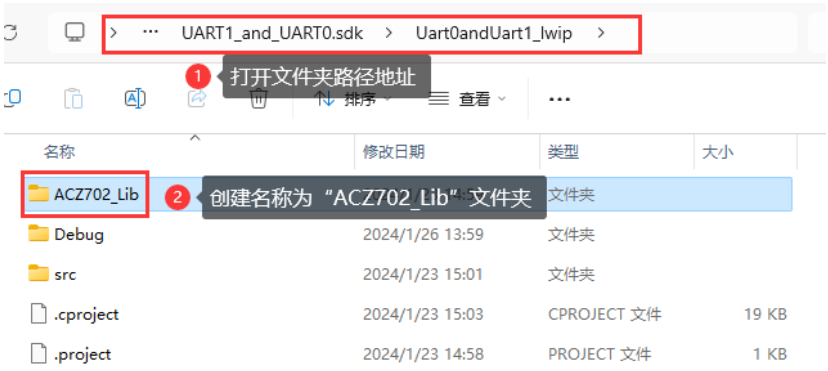

图 1-17 ACZ702_Lib 文件夹

注意：本工程中使用了 xil_printf 发送信息，故可以不添加 PS_UART 文件夹，Xilinx SDK 默认初始化了串口；但建议还是添加上，可以根据小梅哥讲解的 PS 例程修改，有助于学习使用 PS 串口中断

(2) USER 文件夹

将小梅哥例程中，ACZ702_Lib 目录下的 USER 文件夹中的文件拷贝。

图 1-18 USER 文件夹

全部复制到我们创建的新工程“*...\工程名.sdk\Uart0andUart1_lwip\src*”下方。

1.4.2 添加头文件路径

前面我们导入了库到工程中，每个库都包含着对应的头文件，对于 SDK 而言，此时这些头文件都是无效的，我们需要将这些头文件路径添加进工程中。

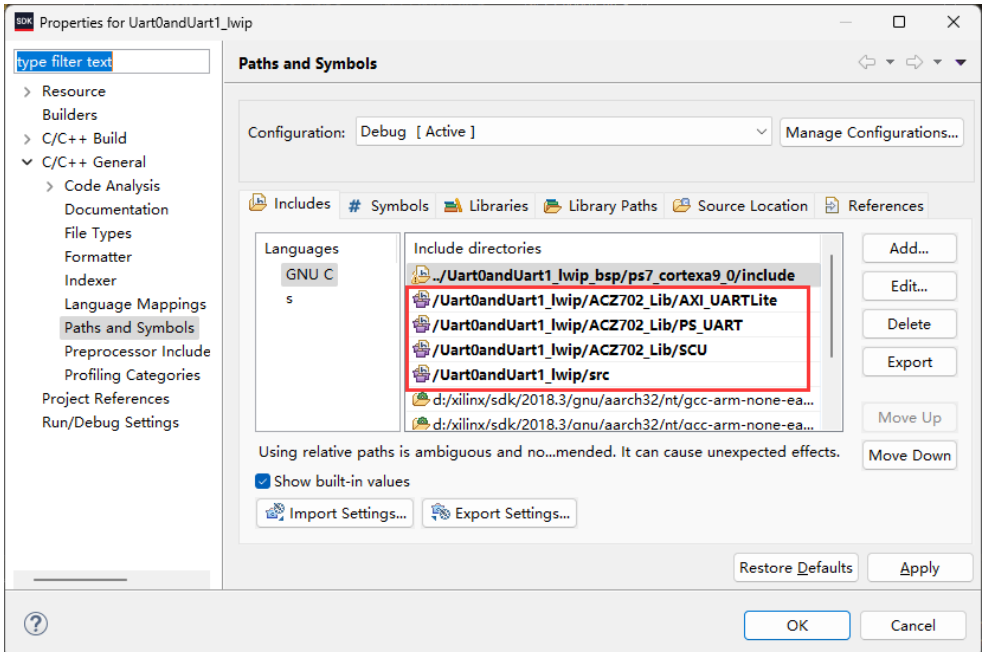

图 1-19 头文件路径

1.4.3 添加用户代码

下面挑选部分重点代码，进行讲解说明，注意**不要直接复制文档中的代码**，可能导致格式错误，推荐自己手打或者参考例程中源码：

(1) main.c

在 main 函数中，首先进行 AXI_UART 初始化、中断初始化；

```
//PL 使用变量
uint32_t Timeout;

//开启通用中断控制器
ScuGic_Init();

//初始化 AXI_UART
AXI_UartLite_Init(&AXI_UART0, XPAR_AXI_UARTLITE_0_DEVICE_ID);

//初始化 AXI_UART 中断
AXI_UARTLite_Intr_Init(&AXI_UART0,
            XPAR_FABRIC_AXI_UARTLITE_0_INTERRUPT_INTR,
             AXI_UART0_Send_IRQ_Handler,AXI_UART0_Recv_IRQ_Handler);

AXI_UARTLite_SendString(&AXI_UART0,"\n\n-----------通道--------\n\r");
```

```
//等待上一轮发送完成
while(!All_Send_Flag);
    All_Send_Flag = 0;

AXI_UARTLite_SendString(&AXI_UART0,"\n 默认：当前通信；\n 处于 uart1 通信
      时，可在网络端输入 uart0，切换到此线路\n\r");

//等待上一轮发送完成
while(!All_Send_Flag);
    All_Send_Flag = 0;
```

由于 AXI 初始化可能会被 LWIP 通信操作打乱，所以需要重新进行一次 AXI 初始化。

```
//重新初始化 AXI_UART 串口，以及中断
AXI_UartLite_Init(&AXI_UART0, XPAR_AXI_UARTLITE_0_DEVICE_ID);
AXI_UARTLite_Intr_Init(&AXI_UART0,
             XPAR_FABRIC_AXI_UARTLITE_0_INTERRUPT_INTR,
            AXI_UART0_Send_IRQ_Handler,AXI_UART0_Recv_IRQ_Handler);
```

(2) echo.c

该文件里面包含了发送/接收功能的关键函数，因本工程需要用 UART 传输 LWIP 发送的数据，所以必须修改 recv_callback 函数，添加上以下内容：

```
/* 使用 flag 判断选择那条 uart 通信*/
if (1 == uart0_1_flag) //默认是 ps_uart 通信，即 uart1
{
    /* 用于判断网络端发送的字符是否包含 uart1 */
    if(strstr((char *)p->payload, "uart1") != NULL) {
    /* 因为此时是 uart1 通信，所以打印出 Uart1 */
        xil_printf("\n 已经是 uart1 通信，如果要切换到 uart0，请在网络端发送
                uart0\n");
     }
    /*切换到 uart0 通信*/
    if(strstr((char *)p->payload, "uart0") != NULL) {
        /* 切换到 uart0 通信 */
        xil_printf("\n\n----------------");
```

```
        xil_printf("\n 已经为你切换到 uart0 通信\n");
        uart0_1_flag = 0; //设置 flag 标志位 0
       }

    xil_printf("\nUART1 通信: %s\n", (char *)p->payload);
}

if (0 == uart0_1_flag) //此时进入 uart0 通信
{
    /* 用于判断网络端发送的字符是否包含 uart0 */
    if(strstr((char *)p->payload, "uart0") != NULL) {
        /* 因为此时是 uart0 通信，所以打印出 Uart0 */
        xil_printf("\n 已经是 uart0 通信，如果要切换到 uart1，请在网络端发送
                uart1\n");
    }
    /* 切换到 uart1 通信 */
    if(strstr((char *)p->payload, "uart1") != NULL) {
        //发送前言
        AXI_UARTLite_SendString(&AXI_UART0,"\n\n----------------");
        while(!All_Send_Flag);
        All_Send_Flag = 0;
        uart0_1_flag = 1; //设置 flag 标志位 1
        AXI_UARTLite_SendString(&AXI_UART0,"\n 已切换到 uart1 通信\n");

        while(!All_Send_Flag);
            All_Send_Flag = 0;
            uart0_1_flag = 1; //设置 flag 标志位 1
       }

      //uart0 通信，注意 Xuartlite_send 函数影响，此函数一次最大接收 16 字节
      /* 假设 p->payload 包含的是数据，且 p->len 是这些数据的长度。 */
      char *original_payload = (char *)p->payload;

      /* 创建一个新的缓冲区，用于保存预先的字符串和原始数据。 */
      char *new_payload = malloc(strlen("UART0 通信:") + p->len + 1);

      /* 使用 sprintf 将预先的字符串和原始数据拼接起来。 */
      sprintf(new_payload, "\nUART0 通信:%s", original_payload);

      /* 现在可以发送新的数据 new_payload */
```

```
        AXI_UARTLite_SendString(&AXI_UART0,new_payload);
        while(!All_Send_Flag);
          All_Send_Flag = 0;

    /* 当不再需要 new_payload 时，使用 free(new_payload) 释放空间。 */
         free(new_payload);
}
```

(3) xemacpsif_physpeed.c 文件修改

在使用官方 lwip 模板出现无法自动协商，是因为该模板默认使用 Realtek 的 RTL8211E 芯片，而 ACZ702 开发板上使用的网卡芯片是 Realtek 的 RTL8211FDI 芯片，由于两种芯片的 PHYSR 寄存器有差异，因此需要小小的修改一下，参考：

【Zynq】【Lwip】解决使用官方 lwip 模板时自动协商失败的问题

简单方法：可以选择拷贝例程中 xemacpsif_physpeed.c 文件，将其覆盖到软件自动生成的 xemacpsif_physpeed.c 文件。

1.5 板级调试与验证

本次实验的板级测试阶段主要集中在以下任务：

首先，我们通过 LWIP 向 ACZ702 开发板发送数据，然后，在网络上位机上配置发送通道，可以选择 PL 端的 uart0 或者 PS 端的 uart1 进行数据发送。最后，利用串口调试助手将接收的数据进行打印。这样的步骤确保了数据的全方位传输和接收。

注意：**下文中 uart0 代指 AXI_UART0 通信（usb 接在 EDA 扩展板端），uart1 代指 PS 端串口通信（usb 接在 ACZ702 v2.0 开发板上）**

系统所需硬件如下，相关模块资料可以点击超链接查看：

1. ACZ702 v2.0 开发板 x1
2. EDA 扩展板一个
3. Type-C 下载线两根

1.5.1 硬件连接

本次系统设计硬件方面连接如图 14-17 所示：

1. 将 EDA 扩展板连接在 ACZ702 开发板的扩展接口上
2. 连接好下载线与串口

3. 电源线（可以不连接）

连接拓展板时请注意要与开发板上的 40pin 信号端口一一对应，否则一旦上电，可能对开发板和 EDA 造成损坏。

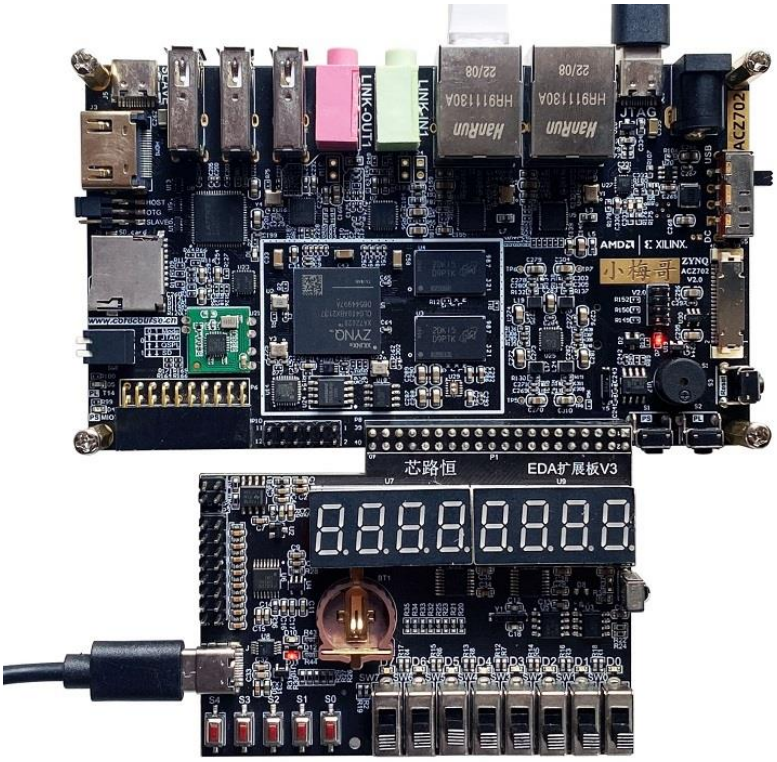

图 1-20 硬件连接图

1.5.2 上位机配置

首先，右键电脑操作系统左下角的 Windows 图标，打开设备管理器，在端口一栏可以看到电脑给设备分配的端口号。这里笔者以 ACZ702V2.0 为例，查询到的端口号如图 1-21 所示；

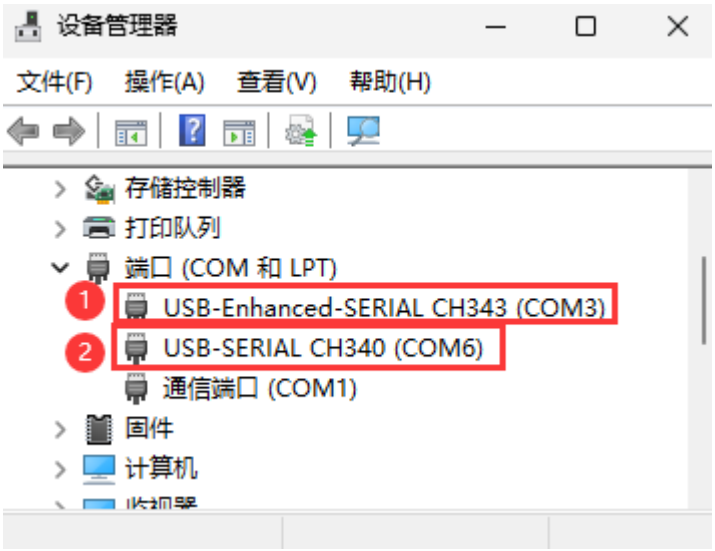

图 1-21 端口号

然后打开两个串口调试助手，除通信端口外，其他配置都相同：波特率设置为 115200，数据位 8 位，停止位 1 位，无校验。

图 1-22 串口调试助手

再配置网络上位机配置：

(1)协议类型：TCP Client

(2)远程主机地址：192.168.1.10

(3)远程主机端口：7

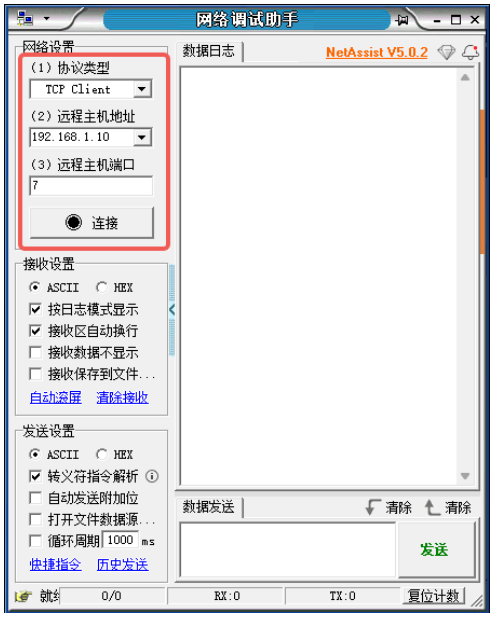

图 1-23 网络上位机配置

1.5.3 通信测试

(1) UART0 切换到 UART1 通信

打开工程，在 SDK 中运行程序。程序运行后会等待网络上位机发送的数据；当开发板接收到数据后，会默认从 AXI UART0 转发到电脑。

例如网络上位机发送“123”，串口会打印出“123”。

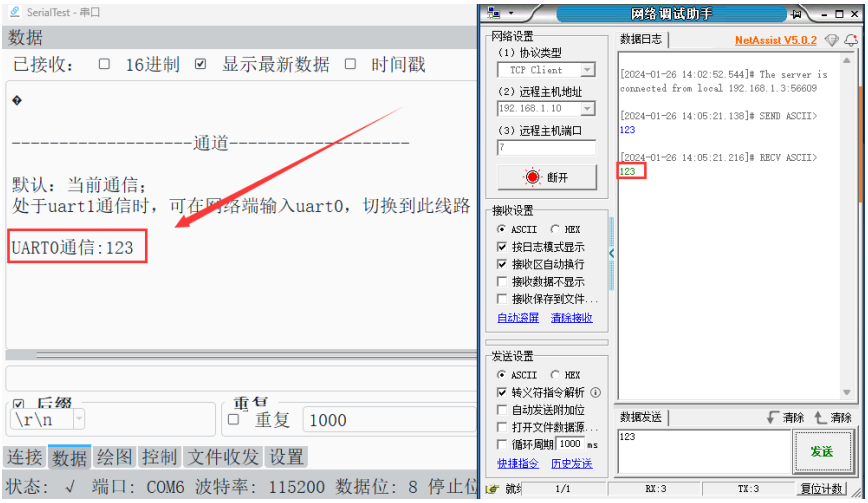

图 1-24 UART0 串口打印

如果想要更改通信方向，从 UART0 变更为 UART1 发送数据；那么可以直接在网络上位机发送“uart1”；此时，如下图 1-25 可以看到 uart0 通道（COM6 端口）上位机，会提示“已切换到 uart1 通信”字样；

然后，继续网络上位机发送数据“567”，可以看到 uart1 通道（COM3 端

口）上位机，会输出网路端传递的数据“567”，说明通信通道切换成功，如图 1-26 所示。

图 1-25 uart0 接收通道切换要求

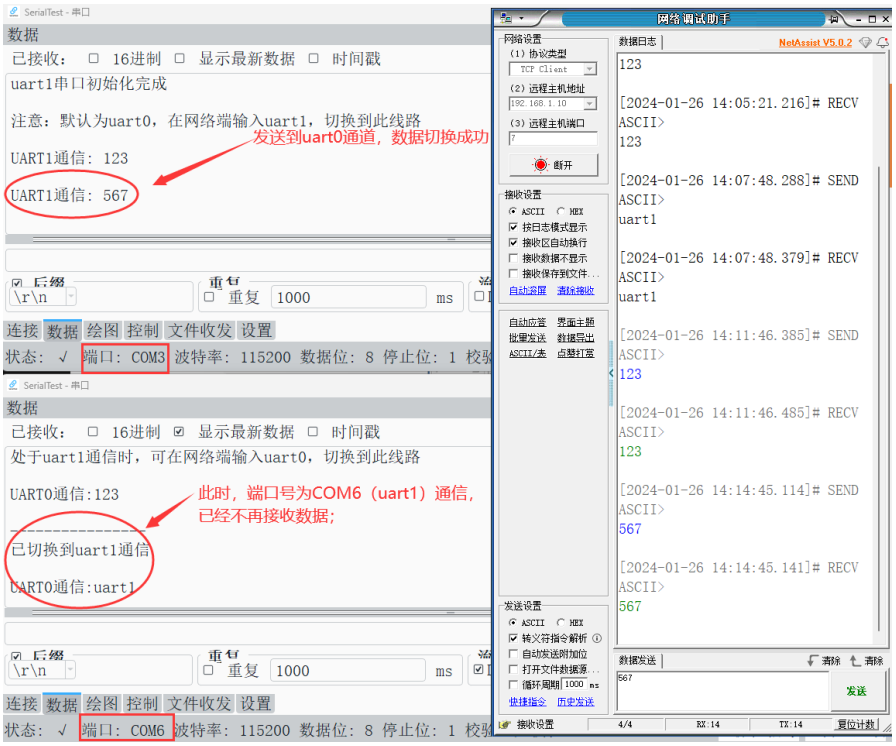

图 1-26 通道切换成功

(2) UART1 切换到 UART0 通信

同样，在网络上位机上发送“uart0”，切换到默认 UART0 发出数据。

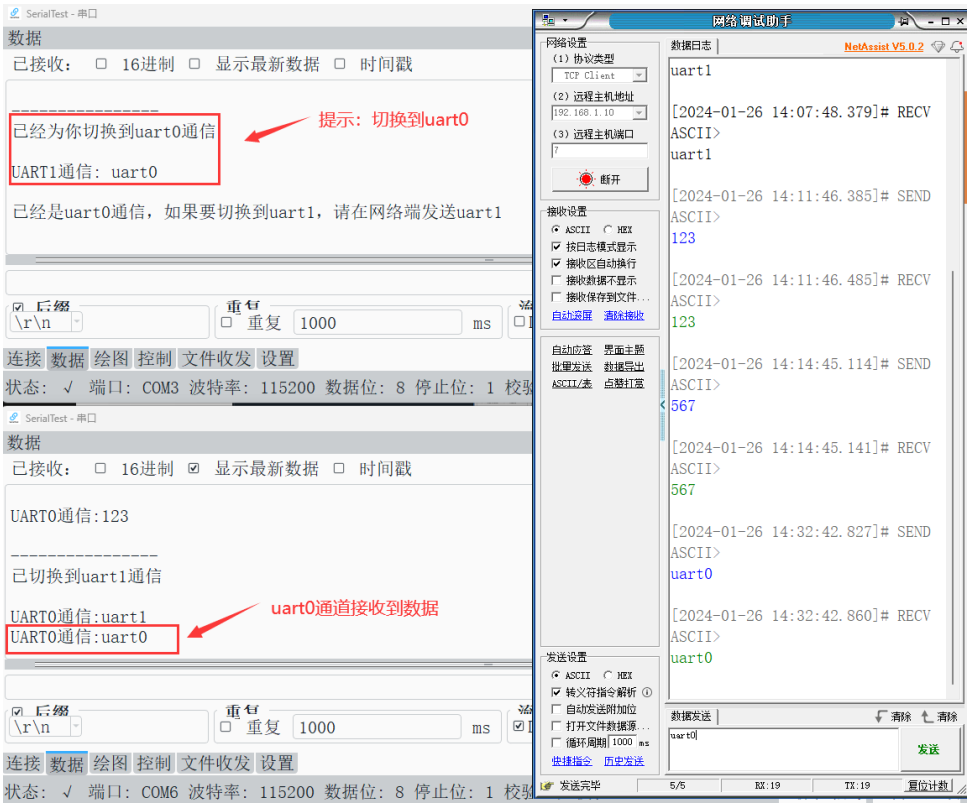

图 1-27 重新切换到 UART0 通信

1.6 总结

在本章中，我们对 SDK 软件自带的 lwIP Echo Server 例程模板进行了改进，增添了串口通信的功能。这一改进使得数据能够从电脑网络上位机通过网线发送到开发板，并再次通过串口返回到电脑端，实现了一种独特的回显程序。实施这次实验的主要目标是让大家对 lwip 的使用有个初步了解，并对之前的串口通信内容进行复习。