1 基于 ZYNQ7020 的 LWIP+RS485 回环通信

工程源码	|-ACZ702 v2.0 开发板 |-- LWIP_485
相关视频课程	无

章节导读

在本节中将运用 LWIP 和 RS485 通信协议实现基于 ZYNQ7020 的回环通信系统。我们将简要介绍 LWIP 的技术原理，并步骤化地说明如何实现数据通过网络线路被传输到 ACX702 开发板，然后通过 RS485 双路评估模块（AC_CANFD_RS485）进行转发的实践操作。本实验的目标是使读者能对 RS485 通信和 LWIP 的应用有一个基本的了解和初步的实践经验。

1.1 RS485 介绍

RS485 是一种差分信号的串行通讯协议。它以其稳健和简洁性质，常被用于远距离和电磁环境恶劣的场所。RS485 使用差分信号进行通讯，可以抵消电线上的噪音信号，保证数据传输的准确无误。

(1) RS-485 与 RS-232 的差异

表 1-1 RS485 与 RS232 对比图

串口协议	操作方式	工作方式	传输距离	电平标准	连接方式
RS232	单端	全双工	15m	逻辑 1:-3V～-15V 逻辑 0:+3V～+15V	点到点
RS485	差分	半双工	1200m	逻辑 1(两线电压差为) +2V～+6V 逻辑 0(两线电压差为) -2V～-6V	多点到多点

(2) RS485 发送与接收

如果你对串行通信的不太了解，首先推荐阅读《基于 C 编程的 Zynq 裸机程序设计与应用教程》中关于 PS 端和 PL 端串口中断实验。

RS485 通信与 UART 串口通信实验对比起来，最明显区别是增加了 RE、DE 控制发送和接收，以及信号从单端改为差分；在一般的通信电路中，会将 RE 和 DE 直接连接，由单个 IO 口来掌控 SP3485 的发送和接收过程。因此，在传输或接收数据时，需要根据实际需要设定 IO 口电平值。

当发送数据时，控制信号应该设置为高电平。

当接收数据时，控制信号应该设置为低电平。

1.2 LWIP 简介

LwIP 是一款针对嵌入式系统设计的开源 TCP/IP 网络协议栈。它既资源消耗小，内存使用及代码量极少，只需要不到几十 KB 的 RAM 和大约 40KB 的 ROM 就能运行。虽然 LwIP 可以和操作系统一起使用，但它并不依赖于任何操作系统。因此，LwIP 非常适合应用于一些规模较小的嵌入式系统中。

(1) TCP/IP 网络协议

TCP/IP,一种富有弹性与通用性的协议簇，具备了在各种不同网络环境中进行信息传输的强大能力。请注意，TCP/IP 并不仅仅只是关于 TCP 和 IP 两个协议，相反，它的含义其实涵盖了一个由 FTP、SMTP、TCP、UDP、IP 等多种协议所构成的协议家族。这是因为在这个协议簇中，TCP 和 IP 两个协议具备极高的代表性，因此我们习惯性地称之为 TCP/IP 协议。每一个这里面的协议都各自履行其职责，从而共同确保消息能够顺利无阻地在网络中传递，形成完整而高效的通信网络。

(2) TCP/IP 网络模型

TCP/IP 协议在构建其框架时，部分参考了 OSI 的网络体系架构。OSI 模型细分为七个层级，由底层到顶层依次包括物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。然而，此七层模型在实际应用中可能较为复杂，因此在 TCP/IP 协议中，这些层次被整合并简化为了四个层级，以提升实施的简便性和高效性。

表 1-2 OSI 模型与 TCP/IP 模型对比图

OSI 模型	TCP/IP 模型
应用层	应用层
表示层	
会话层	
传输层	传输层
网络层	网络层
数据链路层	链路层
物理层	

底层协议为相连的上层协议提供必要的支撑和服务，构成了上层协议能够顺利实施的基石。

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、2 个 AXI Uartlite IP 核、Concat IP 核。

1.3.2 IP 核配置

首先，我们需要配置 Zynq IP 核。

(1) DDR

打开 DDR 配置界面，设置 DDR 型号为“MT41K128M16 JT-125”，如图 1-1 所示。

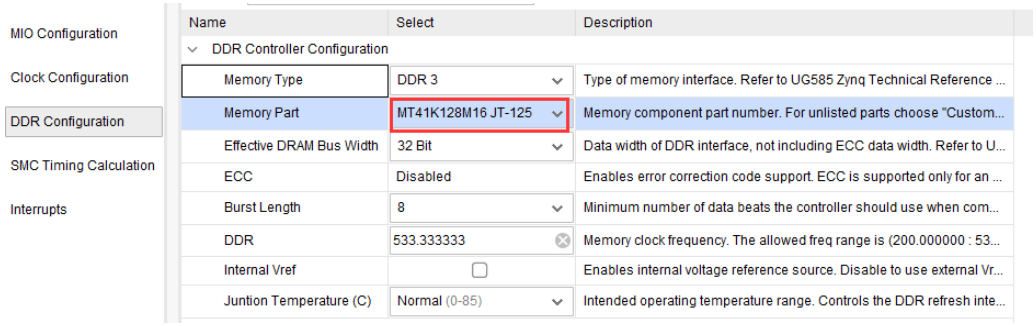

图 1-1 DDR 型号

(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，按照图 1-2 配置 PL 的时钟为 100MHz。

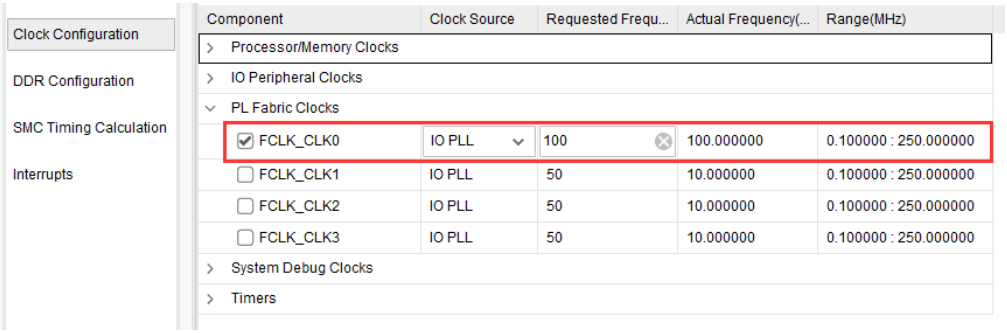

图 1-2 PL 时钟配置

(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。

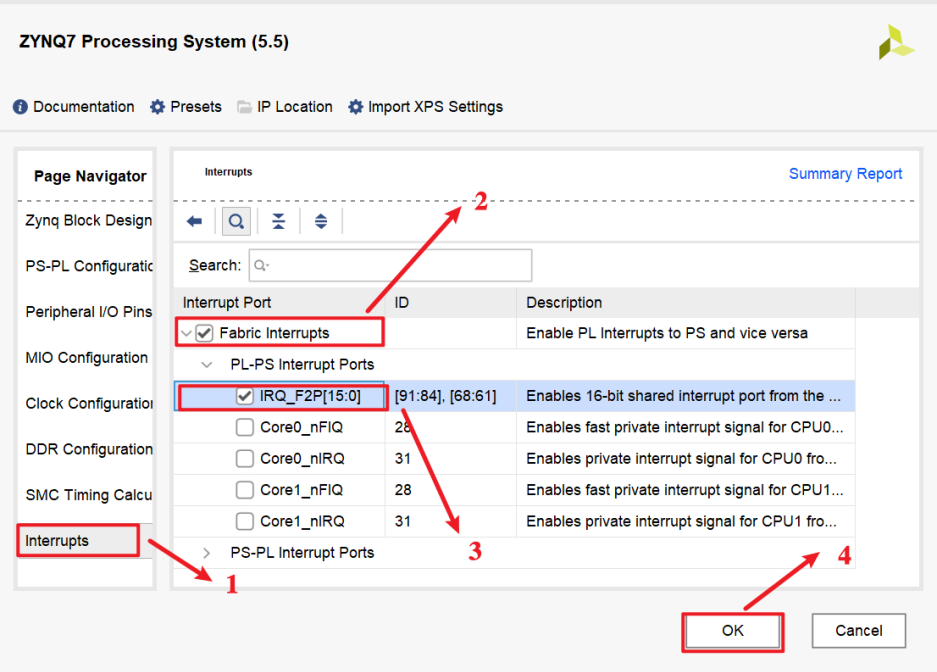

图 1-3 使能中断端口

(4) GPIO

RS485_UART0 和 RS485_UART1 使能位都需要一个 EMIO 信号，因此我们将 EMIO 的位宽设置为 2。设置完成后检查电平状态，将 Bank1 的电平设置为 1.8V。

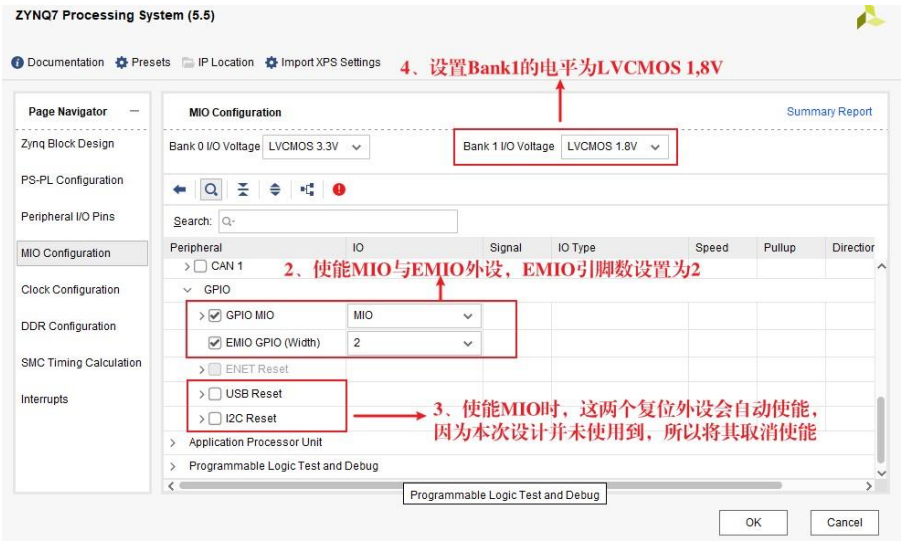

图 1-4 GPIO 配置

(5) 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 1-5 所示：

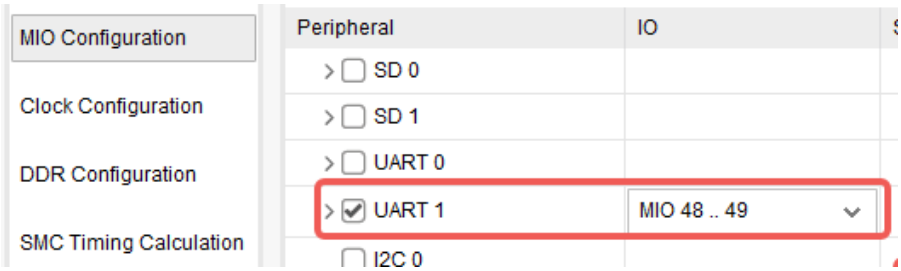

图 1-5 PS 端串口配置

(6) LWIP

按照图 1-6 所示，选择 ENET 0 和下方的 MDIO，并在 IO 列中，设置 ENET 0 的 IO 为 MIO16…27、MDIO 的 IO 为 MIO52…53。

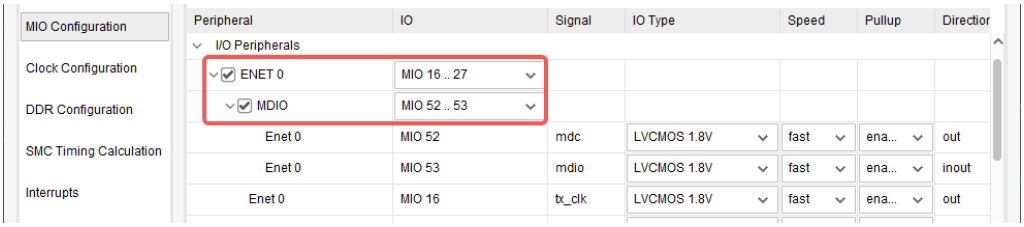

图 1-6 ENET 0

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

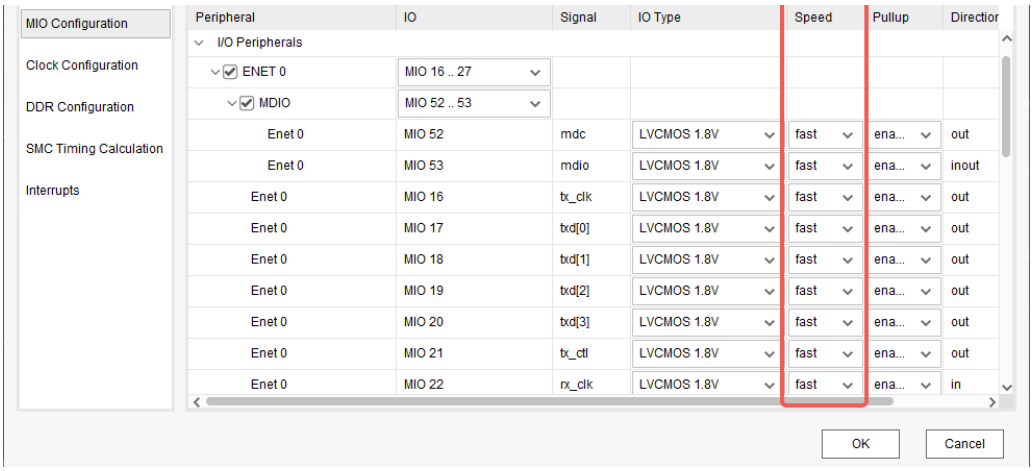

图 1-7 ENET 0 Speed

在 Zynq IP 核配置完成后，分别双击 2 个 AXI Uartlite IP 核，两者配置相同，以其中一个举例；如下图所示，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。

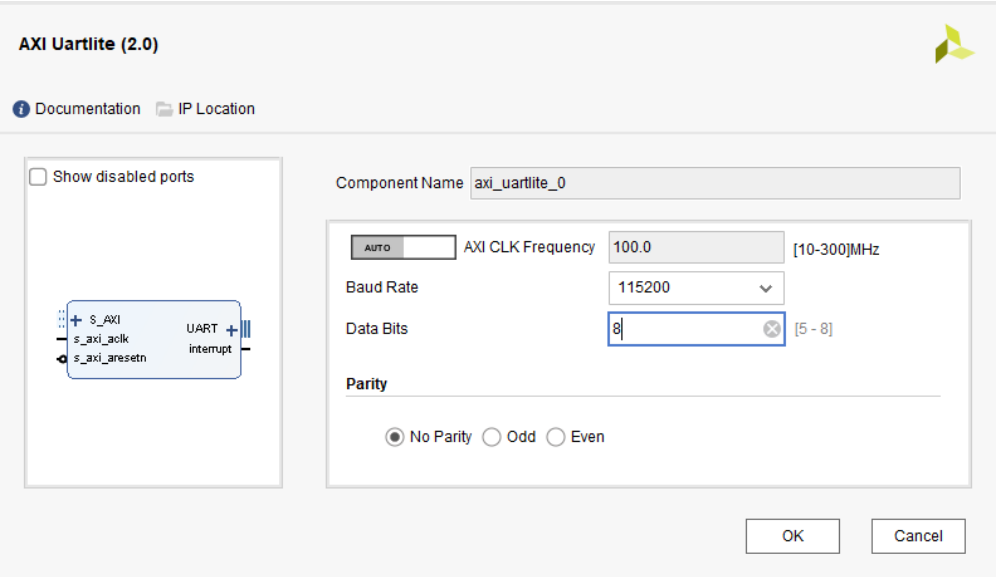

图 1-8 AXI Uartlite IP 核配置

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。

此时，GPIO_0 还未导出，因此我们需要将其手动导出，如图 1-9 所示。

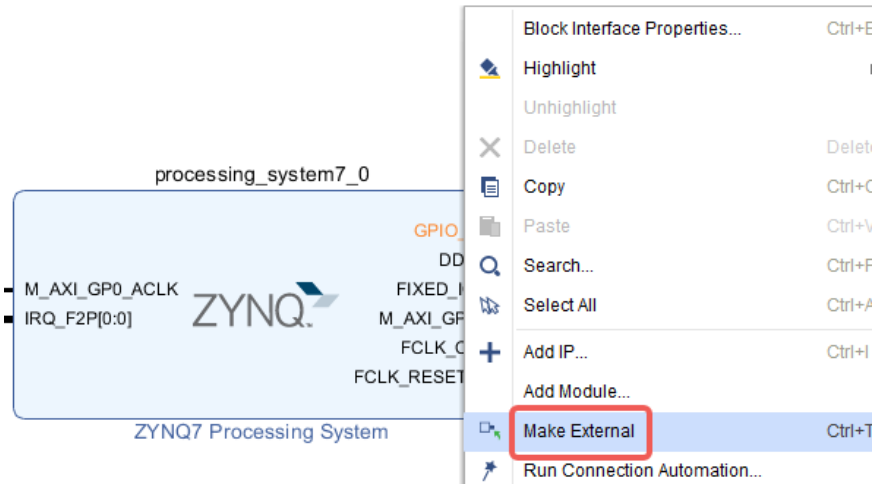

图 1-9 GPIO 导出引脚

1.3.4 端口连接

首先，将 FCLK_CLK0 和 M_AXI_GPIO_ACLK 连接。

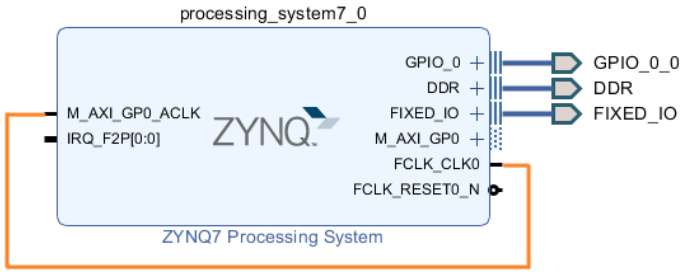

图 1-10 FCLK_CLK0

将两个 axi_uartlite 核的 interrupt 接口与 Concat 核的 In 接口相连接；并将输出接口 dout 与 ZYNQ 核的 IRQ_F2P[0:0] 接口相连接。

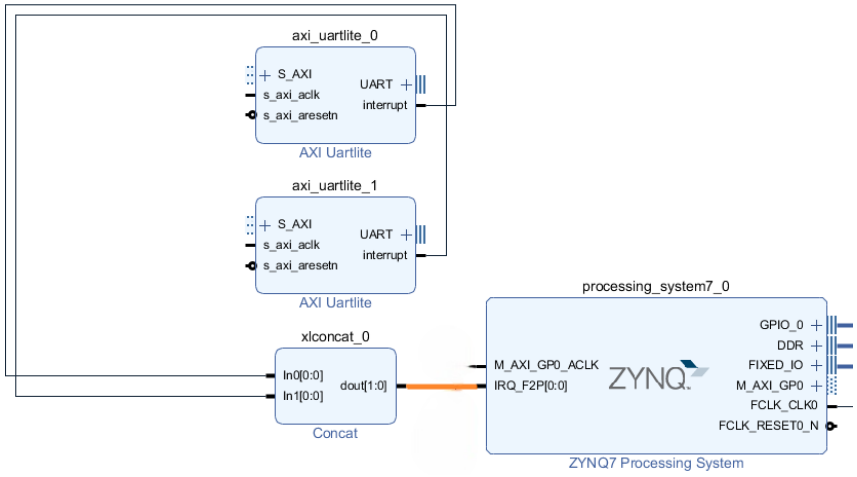

图 1-11 axi_uartlite 中断连接

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

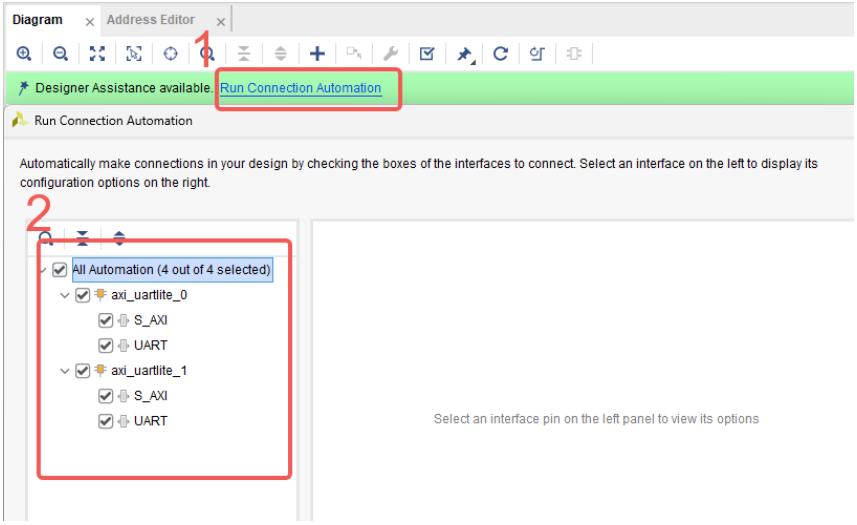

图 1-12 自动连接

点击 Regenerate Layout，重新生成布局，如图 1-13 所示。

图 1-13 重新生成布局

最后，验证设计，出现图 1-14 所示的“Validation successfu...”，说明无错误。

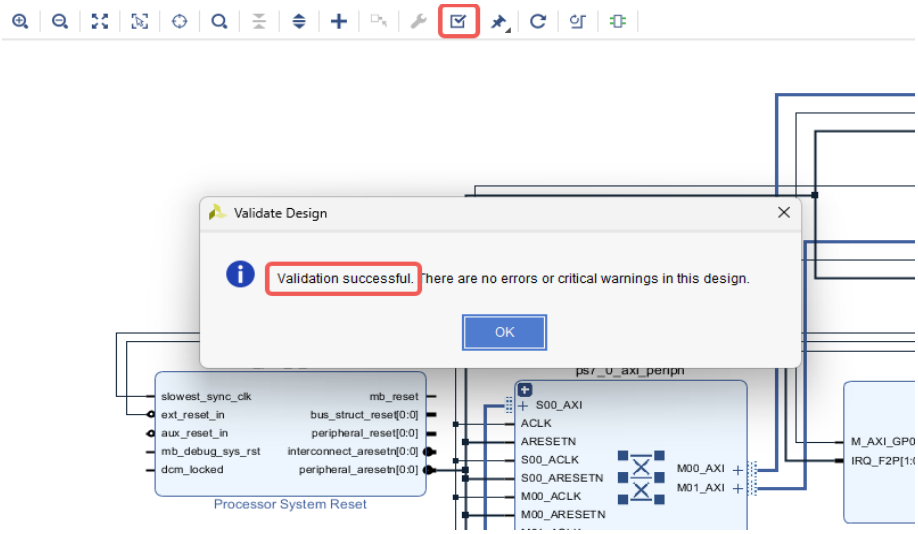

图 1-14 验证设计

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。

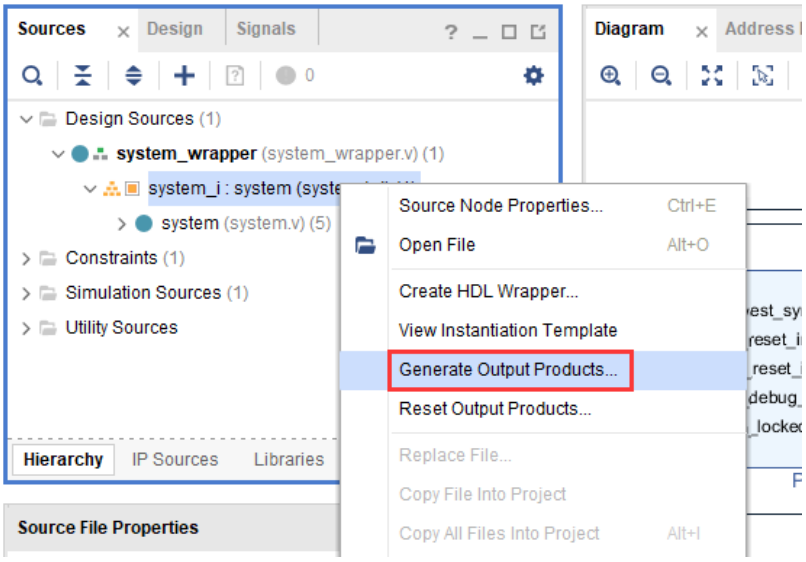

图 1-15 Generate Output Products

如图 1-16 所示，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 16。设置完成后点击“Generate”开始生成。

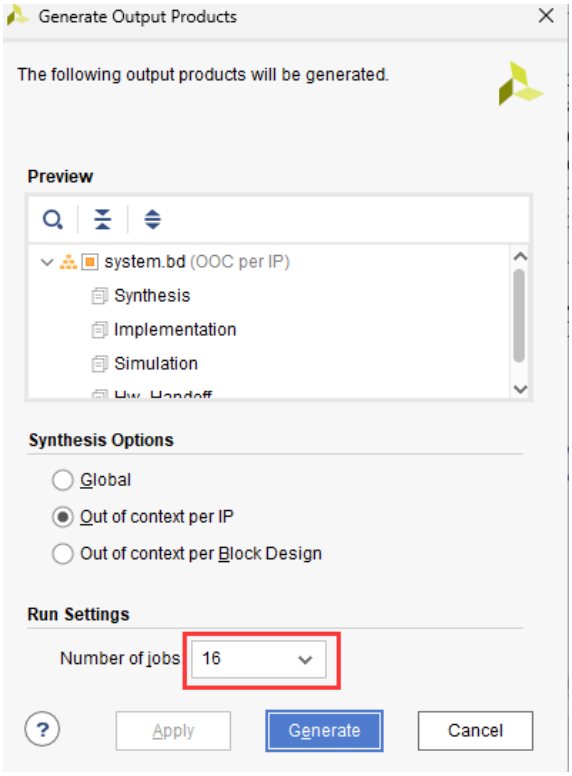

图 1-16 设置生成速度

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。

1.3.6 管脚约束

点击左侧导航栏的“Open Elaborated Design”进行约束和分配。

由于模块中包含两路 RS485 所以对应的驱动管脚如表 1-3、表 1-4 所示，RS485_0_RE 与 RS485_1_RE，单独设置 GPIO 控制：

表 1-3 RS485_0 管脚分配表

Signial Name	Pin NO.
RS485_0_RE	H18
RS485_0_TX	J19
RS485_0_RX	K19

表 1-4 RS485_1 管脚分配表

Signial Name	Pin NO.
RS485_1_RE	H17
RS485_1_TX	J18
RS485_1_RX	K18

(1) UART

打开管脚配置界面，本次设计我们需要为 UART_0 配置的发送和接收引脚编号为 J19、K19，对应引脚电平设置为 LVCMOS33；

同样，UART_1 配置的发送和接收引脚编号为 J18、K18，引脚电平设置为 LVCMOS33。

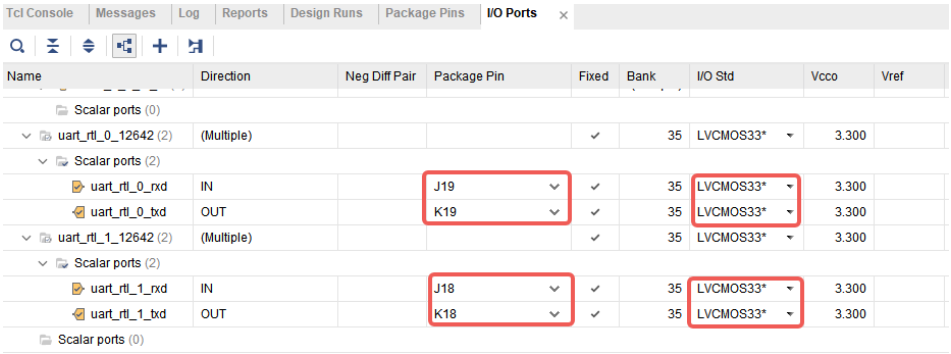

图 1-17 UART 管脚定义

(2) GPIO

GPIO_0_0_tri_io[0]引脚编号设置为 H18，GPIO_0_0_tri_io[1]引脚编号设置为 H17，两者引脚电平都设置为 LVCMOS33。

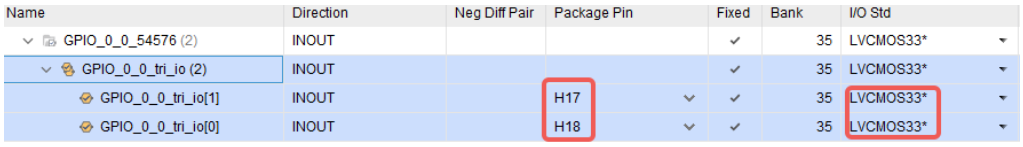

图 1-18 GPIO 管脚定义

完成分配后使用快捷键 Ctrl+S 对约束文件进行保存，命名为“uart0and1”，点击 OK 即可。

1.3.7 生成比特流

点击“Generate Bitstream”开始生成比特流；将生成速度设置到最大”16“；出现下图 1-19 所示弹窗，说明比特流生成成功，点击 cancel 即可。

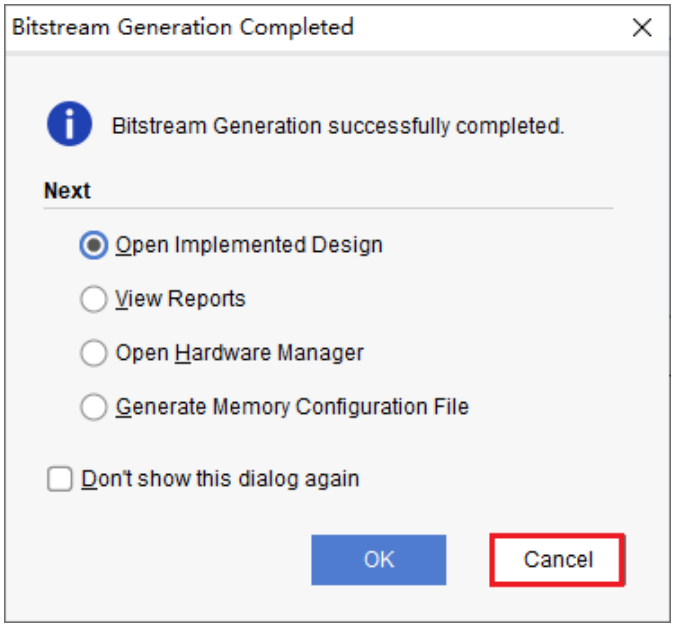

图 1-19 比特流生成成功

1.3.8 导出硬件

首先点击 File，然后在展开的功能栏中选择 Export，最后在 Export 的多个选择项中选择“Export Hardware...”将硬件描述文件导出。

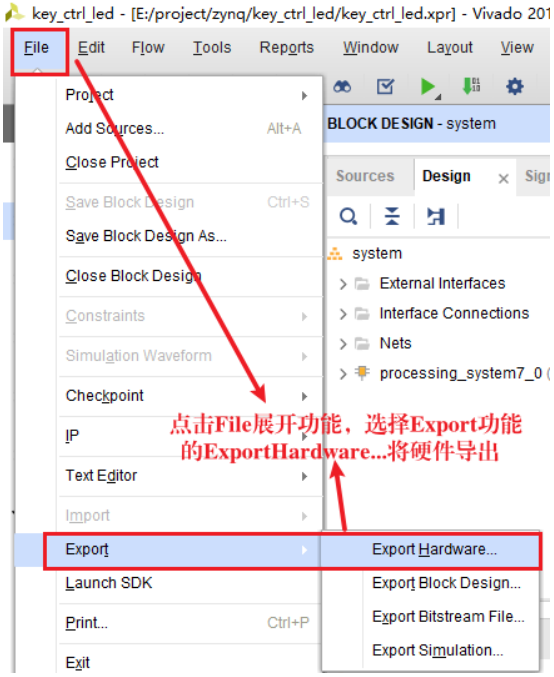

图 1-20 Export Hardware

此时软件会弹出弹窗询问我们是否包含比特流，对于本次设计，我们涉及到了 PL 端的资源使用，因此需要勾选比特流。如图 1-21 所示，勾选比特流后

点击 OK 开始导出。

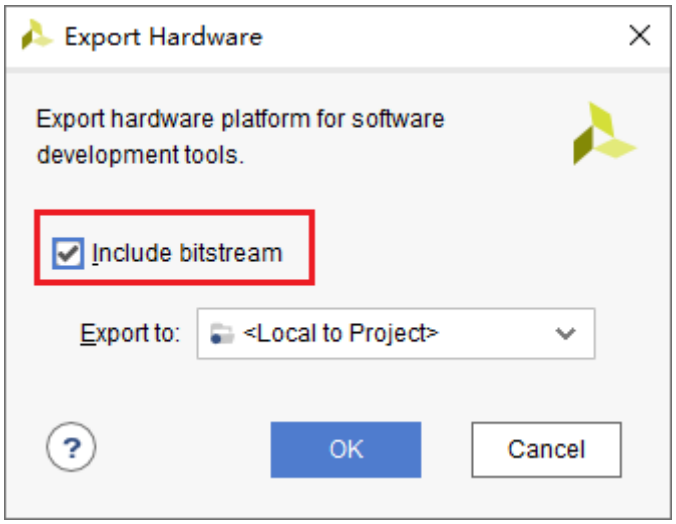

图 1-21 勾选比特流

1.4 CPU 软件程序设计

接下来打开 SDK，开始 CPU 软件程序设计。

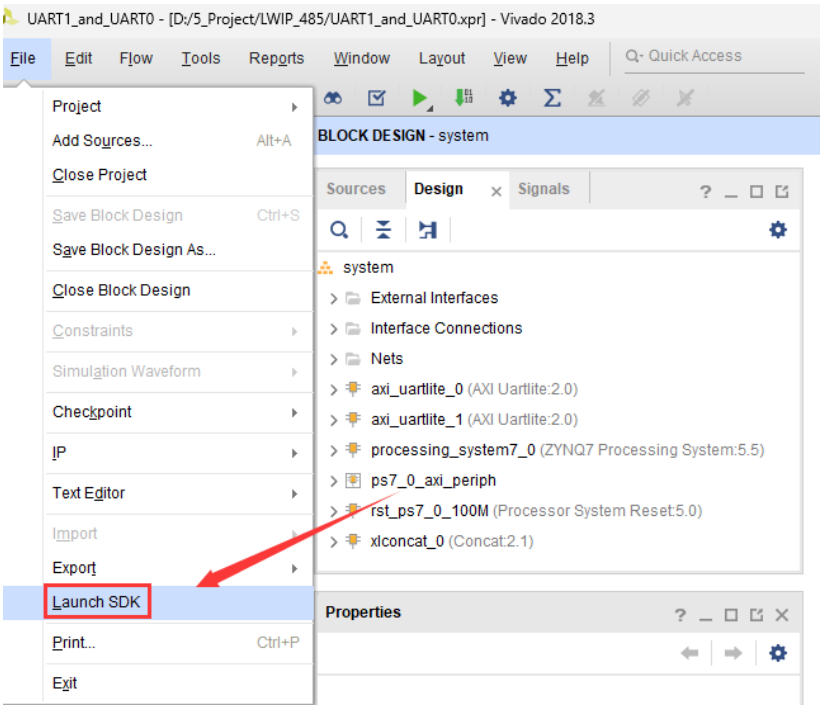

图 1-22 Launch SDK

1.4.1 创建 SDK 工程

打开 SDK 后我们需要新建一个 SDK 工程，点击软件左上方的 File，在展开

的功能栏中依次选择 New/Application Project，如图 1-23 所示。

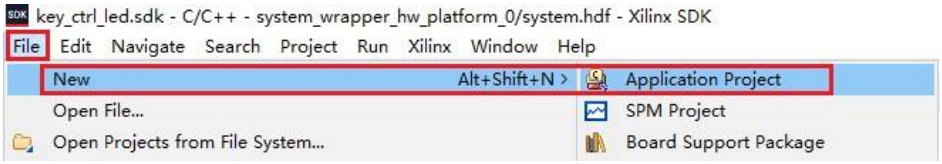

图 1-23 新建工程

接下来会进入工程创建界面。此时我们需要为工程命名，设置为“LWIP_485”。这里我们保持默认，点击 Next 进行下一步。

然后为工程选择模板，选择“lwIP Echo Server”，如图 1-24 所示，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。

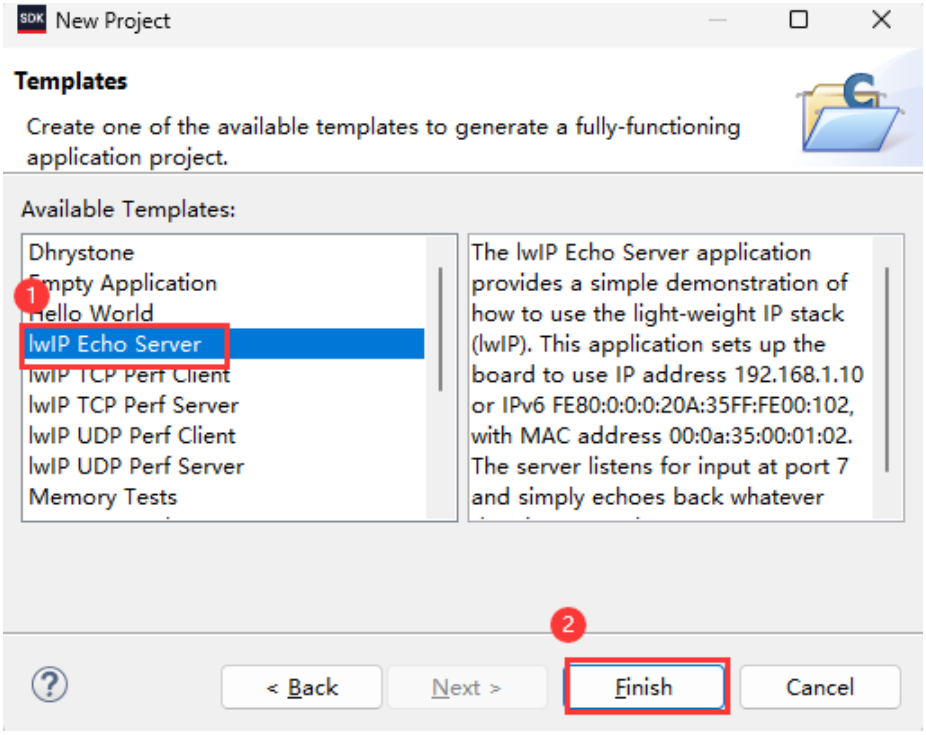

图 1-24 选择模板

1.4.2 添加应用库

(1) ACZ702_Lib 文件夹

打开我们提供的应用库资料，路径如下：*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*。

在创建的工程下面，新建一个 ACZ702_Lib 文件夹将“AXI_UARTLite”、“PS_GPIO”、“PS_UART”、“SCU” 这 4 个文件夹添加到其中。

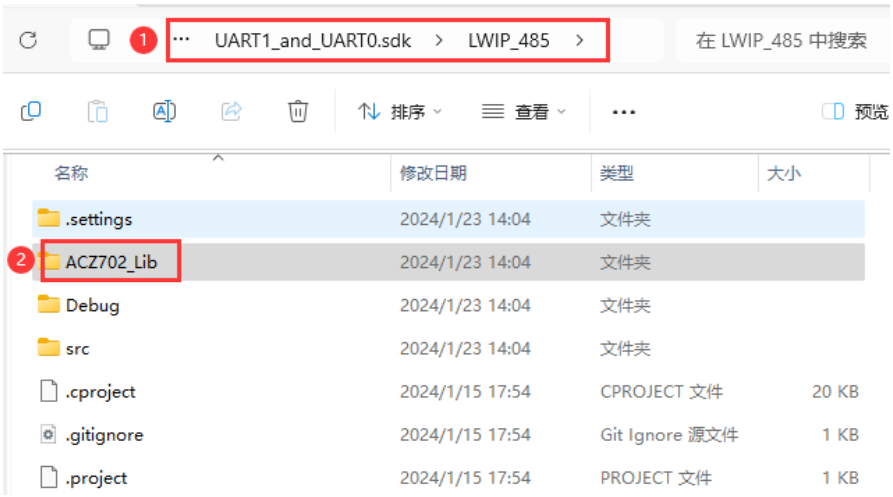

图 1-25 ACZ702_Lib 文件夹

注意：在本工程中我们使用 xil_printf 来发送信息，因此并不需要添加 PS_UART 文件夹，因为使用 LWIP_Echo_Server 模板创建的工程默认地初始化了 PS 端的串口。然而，依然建议大家添加 PS_UART 文件夹，然后参考《基于 C 编程的 Zynq 裸机程序设计与应用教程》的串口实例进行修改，这样有助于学习和理解处理器串口中断的使用。

(2) USER 文件夹

打开应用库资料，将 ACZ702_Lib 目录下的 USER 文件夹中的文件拷贝，粘贴到我们创建的新工程路径“*.../LWIP_485/src*”下方，结果如图 1-27 所示。

图 1-26 USER 文件夹

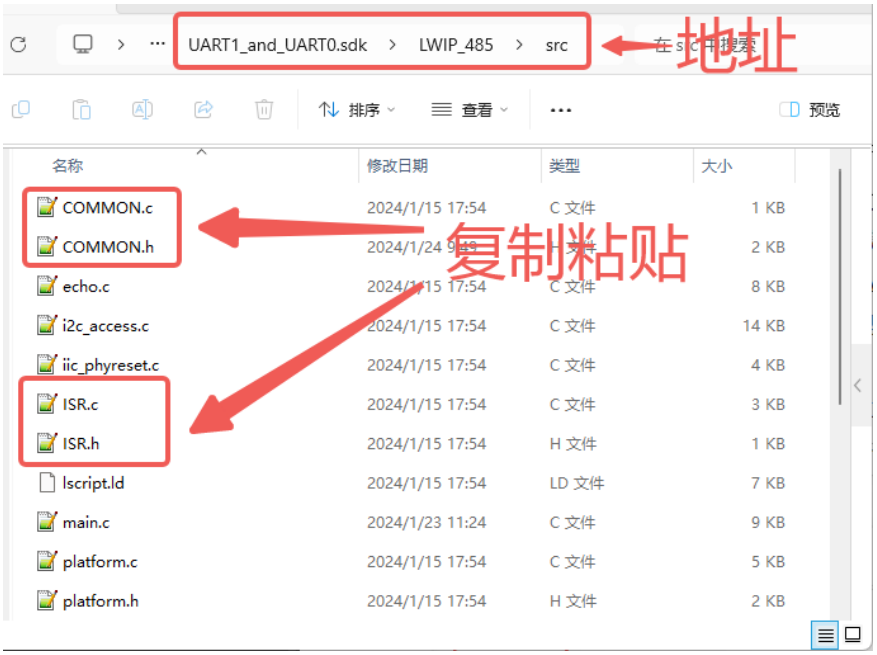

图 1-27 新工程 src 路径

此时打开创建的工程，左键点击“LWIP_485”，然后按下 F5，出现图 1-28、图 1-29 所示内容，说明添加应用库成功。

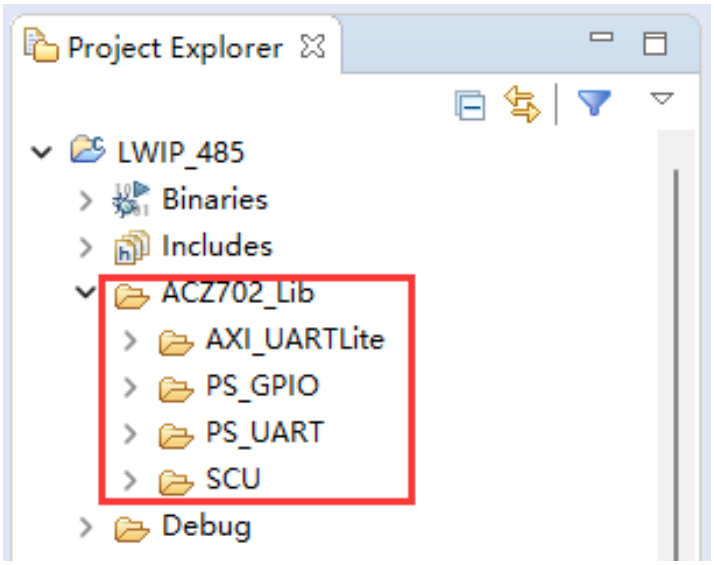

图 1-28 工程创建完成界面 1

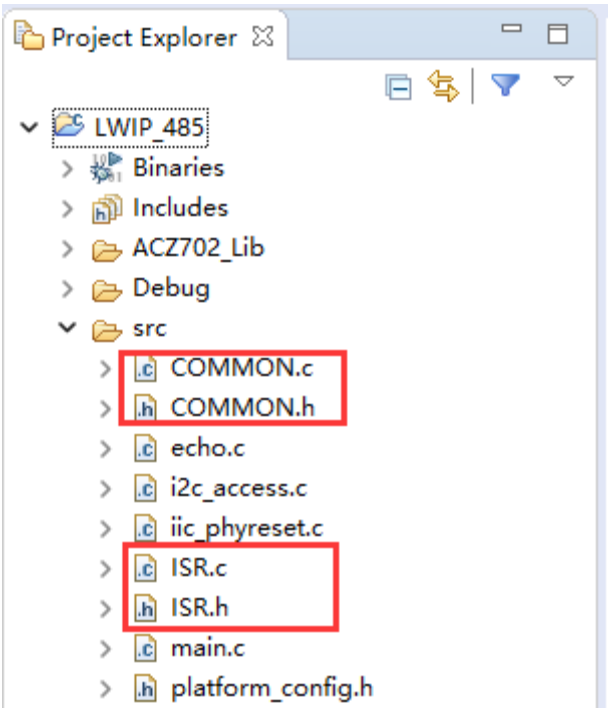

图 1-29 工程创建完成界面 2

1.4.3 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，完成后如图 1-30 所示。

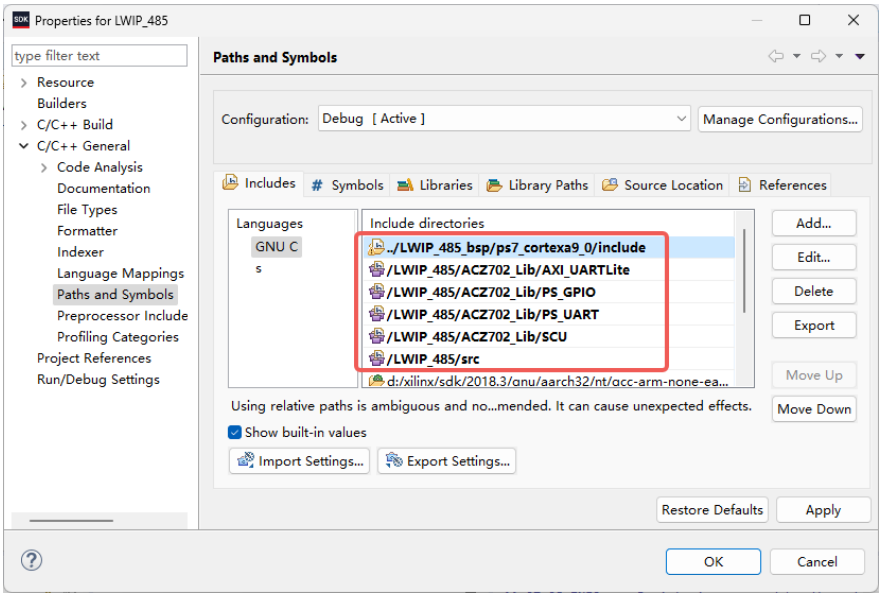

图 1-30 头文件路径

1.4.4 添加用户代码

具体代码，可以参考例程，下面挑选重点代码，进行讲解说明：

(1) main.c

在 main 函数中，首先进行 GPIO 与 AXI_UART 初始化、中断初始化；

```
/***************自定义程序部分 1，开始***************/
    PS_GPIO_Init(); //初始化 PS 端 MIO 和 EMIO

    PS_GPIO_SetMode(RS485_0_RE, OUTPUT, 1);//uart0，发送模式
    PS_GPIO_SetMode(RS485_1_RE, OUTPUT, 0);//uart1，接收模式

    //开启通用中断控制器
    ScuGic_Init();

    //初始化 AXI_UART_0
    AXI_UartLite_Init(&AXI_UART0, XPAR_AXI_UARTLITE_0_DEVICE_ID);

    //初始化 AXI_UART_1
    AXI_UartLite_Init(&AXI_UART 1, XPAR_AXI_UARTLITE_1_DEVICE_ID);

    //初始化 AXI_UART_0 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);

    //初始化 AXI_UART_1 中断
     AXI_UARTLite_Intr_Init(&AXI_UART1,
            XPAR_FABRIC_AXI_UARTLITE_1_INTERRUPT_INTR,
             AXI_UART1_Send_IRQ_Handler,AXI_UART1_Recv_IRQ_Handler);
/***************自定义程序部分 1，结束***************/
```

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 串口通信初始化。

```
/***************自定义程序部分 2，开始***************/

    //再次初始化
    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);
```

```
/***************自定义程序部分 2，结束***************/
```

(2) COMMON.h

重点添加 RE 宏定义，用于 RS485 发送和接收功能切换；

```
//RS485_0_RE  对应 H18，PL_GPIO21，KEY4
//RS485_1_RE  对应 H17，PL_GPIO24，LED6
#define RS485_0_RE (54 + 0)
#define RS485_1_RE (54 + 1)
```

(3) 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，下面只部分摘录，其他位置的修改参考例程：

```
/* 使用 flag 判断选择数据流向，默认数据流向 1：RS485_0->RS485_1*/
if (1 == uart0_1_flag)
{
    //当 RS485_0_RE=0，低电平,设置 485 为接收模式
    //当 RS485_0_RE=1，高电平，设置 485 为发送模式
    PS_GPIO_SetMode(RS485_0_RE, OUTPUT, 1);//uart0，发送模式
    PS_GPIO_SetMode(RS485_1_RE, OUTPUT, 0);//uart1，接收模式

    /* 用于判断网络端发送的字符是否包含 uart1 */
    if(strstr((char *)p->payload, "uart1") != NULL) {
        xil_printf("\n\n**********************\n");
        xil_printf("\n 选择错误，已是数据流向 1\n");
      }

    /* RS485_UART0 发送数据到 RS485_UART1。 */
    AXI_UARTLite_SendString(&AXI_UART0,(char *)p->payload);
    while(!All_Send_Flag);
    All_Send_Flag = 0;

    /* 等待 1000us，以确保数据发送完成。 */
    usleep(1000);

    /* RS485_UART1 接收数据 */
    AXI_UARTLite_RecvData(&AXI_UART1, Receive_Buffer_1, 20);

    /* 使用 PS 端串口，打印出 RS485_UART1 串口接收到的数据。 */
    xil_printf("\n 流向 1：RS485_0->RS485_1: %s\n", Receive_Buffer_1);
```

```
    /*切换到 uart0 通信*/
    if(strstr((char *)p->payload, "uart0") != NULL) {
        xil_printf("\n\n\n************************\n");
        xil_printf("\n 已切换：数据流向 2\n");
        uart0_1_flag = 0; //设置 flag 标志位 0
      }

    /* 清空 RS485_UART1 接收数据 */
    memset(Receive_Buffer_1, '\0', sizeof(Receive_Buffer_1));
}
```

(4) AXI_UARTLite.h 与 AXI_UARTLite.c

因为有 2 个 AXI_UARTLite IP 核，所以要对提供的应用文件进行修改；设置两个 AXI UARTLite 实例。

在 AXI_UARTLite.c 文件中添加 AXI UARTLite 实例定义。

```
#include "AXI_UARTLite.h"

//添加 AXI UARTLite 实例
XUartLite AXI_UART0;
XUartLite AXI_UART1;
```

在 AXI_UARTLite.h 文件中添加 AXI UARTLite 实例声明。

```
//AXI_UARTLite.h 文件
#ifndef __AXI_UART_LITE_H__
#define __AXI_UART_LITE_H__

#include "COMMON.h"
#include "xuartlite.h"

//声明 AXI UARTLite 实例
extern XUartLite AXI_UART0;
extern XUartLite AXI_UART1;
```

(5) ISR.h 与 ISR.c

在 ISR.h 中代码添加上下列代码。

```
//中断处理函数
void AXI_UART0_Send_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData);
```

```
void AXI_UART0_Recv_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData);

void AXI_UART1_Send_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData);

void AXI_UART1_Recv_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData);
```

对于 ISR.c 文件，将 AXI_UART0 的发送/接收中断处理函数复制并设置为：

```
void AXI_UART1_Send_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData)
{
      /* ↓↓↓用户处理↓↓↓ */
      All_Send_Flag = 1;
      /* ↑↑↑结束处理↑↑↑ */
}

void AXI_UART1_Recv_IRQ_Handler(void *CallBackRef, unsigned int
                                EventData)
{
      /* ↓↓↓用户处理↓↓↓ */
      All_Recv_Flag = 1;
      /* ↑↑↑结束处理↑↑↑ */
}
```

(6) 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

本次实验的板级验证阶段主要围绕以下任务进行：通过 LWIP 将数据传输到 ACZ702 开发板，接着，第一个 RS485 通道将输送到的数据进行串行发送，第二个通道接收来自第一个 RS485 的数据。最后，使用 PS 端串口将第二个通道接收到的数据打印出来。当串口通信软件打印出数据后，说明通信功能已经成功

实现。

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. AC_CANFD_RS485 模块
3. Type-c 线缆 x1

1.5.1 硬件连接

芯路恒 AC_CANFD_RS485 模块上配备了两路 CAN 控制器和两路 RS485 收发器，其中 RS485 简易电路图，如图 1-31 所示：

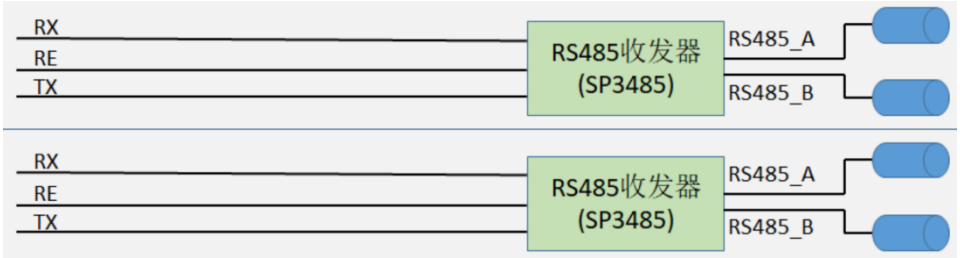

图 1-31 SP3485

所以硬件连接如下图所示：

（等待拍摄）

设计对供电的要求不高，因此可以直接使用 type-c 供电，开发板上一共有两个以太网接口，靠近调试接口的为 PS 侧的以太网接口，用户可以通过开发板背侧的丝印来确定自己当前连接的是哪个接口。连接网线时，一头连接在开发板 PS 侧网口，另一头连接在电脑/路由器的网口上。

1.5.2 下载验证

点击模板工程资源文件，将生成的烧录文件下载至开发板中如图 1-32 和图 1-33 所示，烧录流程如下：

1. 双击 GDB 或 System Debugger 新建配置任务，推荐使用 GDB。

2. 在右侧检查是否添加比特流文件和 PS 初始化脚本，通常软件会自动添加，如果没有，用户可以关闭并重新打开该界面或自己手动添加。

3. 确认下方三个选项都被勾选，这四个选项分别是系统复位、配置 FPGA、PS 初始化和 PL-PS 电平转换。后两个选项通常是默认勾选的，用户需要手动勾选前两项以确保 PL 部分能够被正确配置。

4. 点击上方的 Application 切换到应用界面。

5. 检查.elf 文件是否添加且烧录任务是否被选中。这里.elf 文件由软件编译产生，参与用户程序的运行。 SDK 默认情况下设置了自动编译，用户保存设计后软件便会编译并生成.elf 文件，所以当搜索不到.elf 文件时检查设计是否保存。

6. 点击“ Run”开始烧录

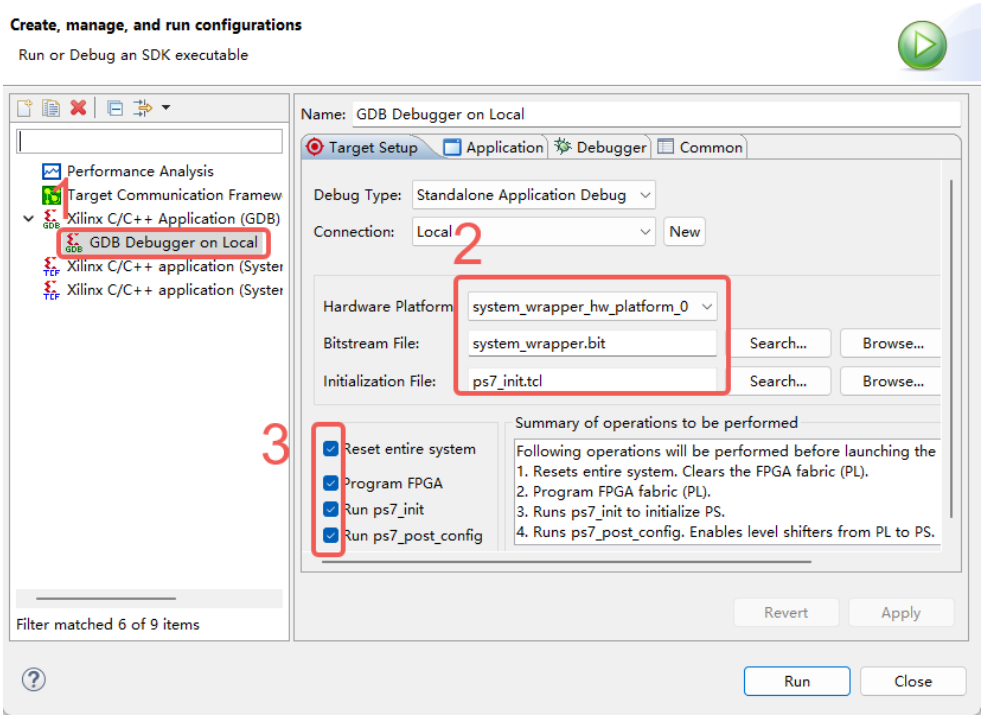

图 1-32 配置烧录任务

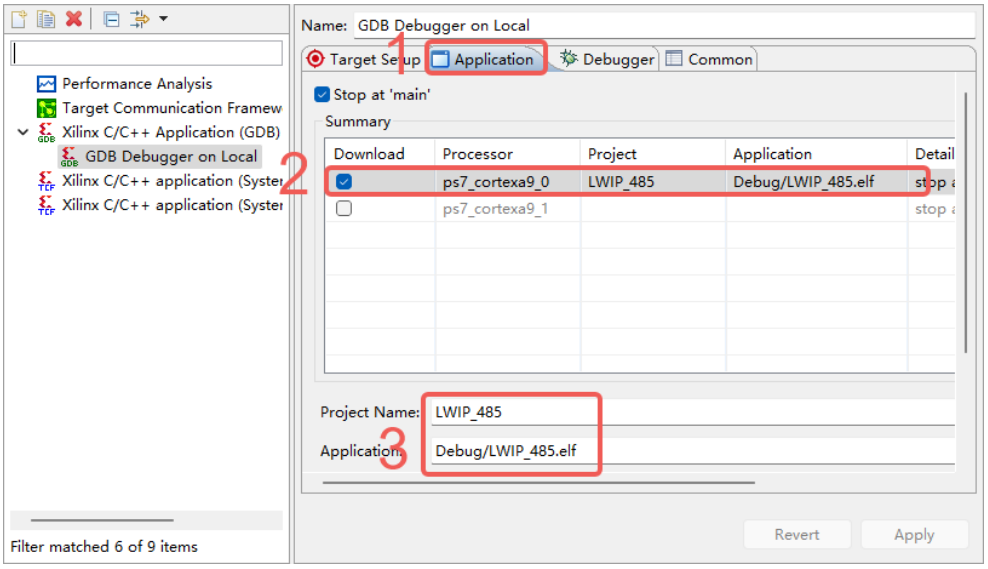

图 1-33 配置烧录任务

1.5.3 通信测试

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数据；当串口通信助手打印出下列信息，说明网络、串口初始化成功。

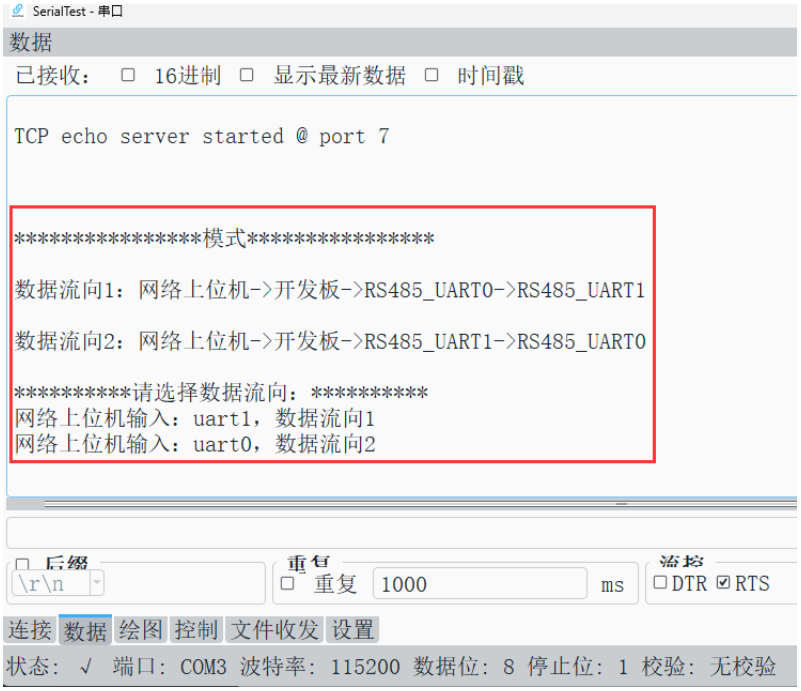

图 1-34 通信测试

(1) RS485_uart0 转发网络数据

当开发板接收到数据后，默认数据流向是从 RS485_uart0 发出，RS485_uart1 接收数据，并将数据通过 PS 端串口打印到屏幕。

此时在网络上位机发送数据“123”，可以看到串口通信助手也会显示相关的数据流向、以及打印出“123”。

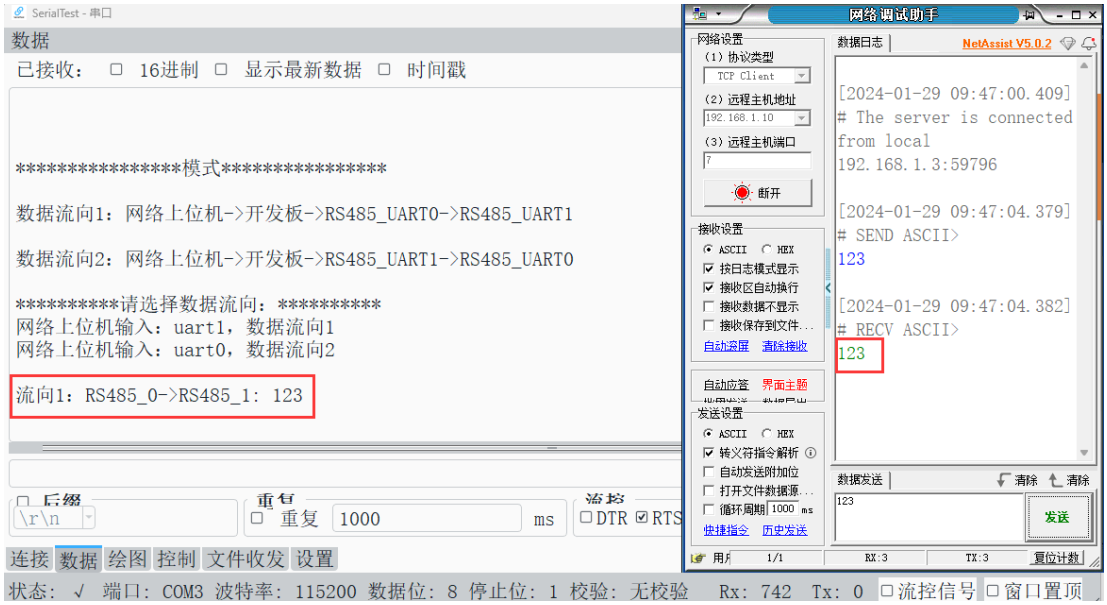

图 1-35RS485_uart0 发送数据

(2) RS485_uart1 转发网络数据

如果不想从 RS485_uart0 发送数据，想改为 RS485_uart1 端发送，操作也很简单。只需要在网络上位机上发送“uart0”，那么数据就会从 LWIP->开发板-> RS485_uart1-> RS485_uart0。

图 1-36RS485_uart1 发送数据

注意，在网络上位机上发送 uart0，代表 RS485_uart0 接收数据；发送 uart1，代表 RS485_uart1 接收数据。

1.6 总结

在这次实验中，我们使用了 AC_CANFD_RS485 模块来实现网络数据的转发功能，并同时利用处理器（PS 端）串口打印出数据。这种设置为读者提供了一个初步的实践场景，展示了数据如何从一个网络通过设备被传输和接收。

此外，为了增加操作的广度和深度，读者也可以考虑使用两块开发板以及两块 RS485 模块来实现远距离通信。这种配置不仅能进一步提高 RS485 通信技术的实践经验，还能更好地理解在更复杂和现实环境中网络数据转发的过程。