1 千兆以太网传输 ACM9238 数据采集

工程源码	|- BX71 开发板 |-- BX71_ad9238_udp_ddr3
相关视频课程	无
说明	如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

章节导读

本节内容将介绍基于 ACM9238 模块利用网口进行数据采集的相关内容。FPGA 采用 RGMII 接口与以太网 PHY 芯片 RTL8211 通信，接收以太网数据包，并提取出以太网数据包中 UDP 协议报文的数据内容，然后将数据转化成控制命令，从而实现对 ACM9238 模块的采样频率、数据采样个数以及采样通道的合理配置，采集完成后的数据存放至 DDR3 中，然后通过网口以 UDP 协议传输到电脑。用户可以在电脑上通过网口调试工具进行指令的下发，并以文件的形式保存接收到的数据，然后使用 MATLAB 软件进行进一步的数据处理分析。

1.1 系统整体设计

通过电脑上的网络调试助手，将命令帧进行发送，然后通过 BX71 开发板上的以太网芯片接收，随后将接收到的数据转换成命令，从而实现对 ACM9238 模块采样频率、数据采样个数以及采样通道的配置。配置完成之后，ACM9238 模块开始采集数据，将采集的数据存储至 DDR 中，最后数据通过网口传输至电脑，电脑端将采集到的数据通过 MATLAB 进行进一步的分析，系统的整体设计框图如图 1-1 所示。

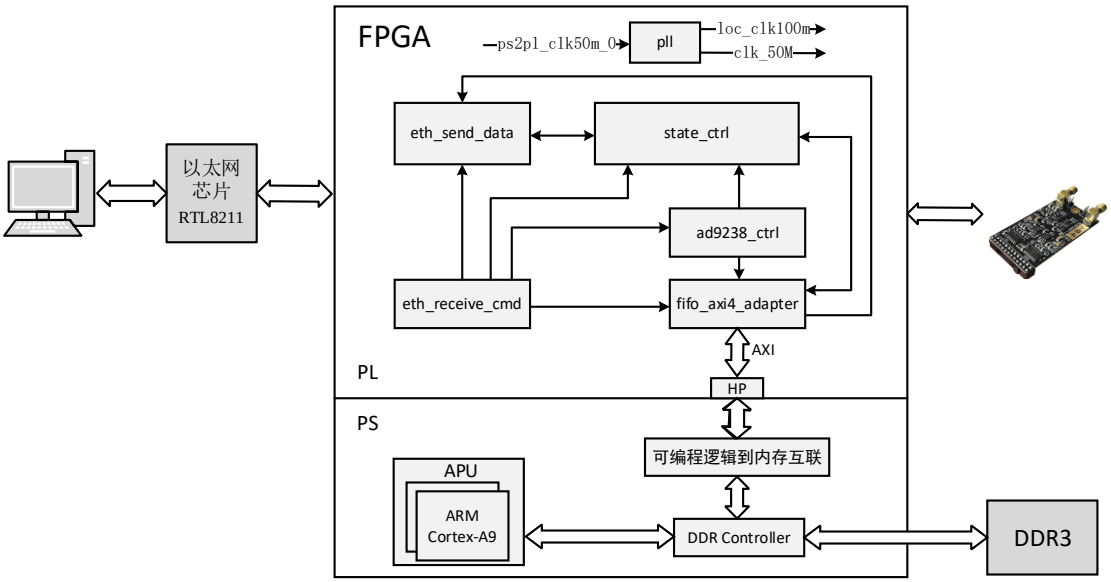

图 1-1 千兆以太网传输 9238 数据采集

对于上述结构框图，这里我们仅对 PL 侧的模块设计进行讲解。至于 PS 侧数据如何写入到 DDR3 中，我们在开发板 02 文档的《AXI4 转 FIFO 接口模块设计》章节中有过说明，读者也可以参看下帖：

【ZYNQ】ZYNQ 器件的 DDR3 存储器使用相关知识介绍

PL 部分的模块说明如下：

1. pll 模块：锁相环模块，输入时钟 50M，由 PS 输出给 PL；输出 100M 的时钟用于 AXI 接口；输出 50M 的时钟给其它模块使用。
2. eth_receive_cmd 模块：以太网接收命令模块，对以太网接收到的数据进行分析，将接收的数据转换成相应的控制数据并输出到对应的模块。
3. ad9238_ctrl 模块：ACM9238 控制器模块，该模块内部包含速度控制模块，以及数据位宽转换模块。
4. state_ctrl 模块：ADC 采集数据 DDR3 缓存以太网发送状态控制模块，协调各个模块的信号控制，程序状态的总控制模块。
5. fifo_axi4_adapter 模块：fifo 接口到 AXI4 接口的转换模块(含 2 个 FIFO)。

1.2 ACM9238 模块简介

该模块如下图 1-2 所示。ACM9238 模块配合前端模拟信号调理电路，实现了±5V 电压范围内信号的高速采样。该模块共使用 2 路完全相同的 AD 采样和信号调理电路，构成了双通道高速 AD 采样电路。两路 ADC 电路完全独立，结

构和元器件参数相同，确保了两个通道有较高的一致性。本模块与 FPGA 连接采用并行接口，每路 ADC 包括 12 位数据信号（ADC_DATA），1 位时钟信号（ADC_CLK），1 位超量程指示信号（ADC_OTR），该模块接口图如下图 1-3 所示。

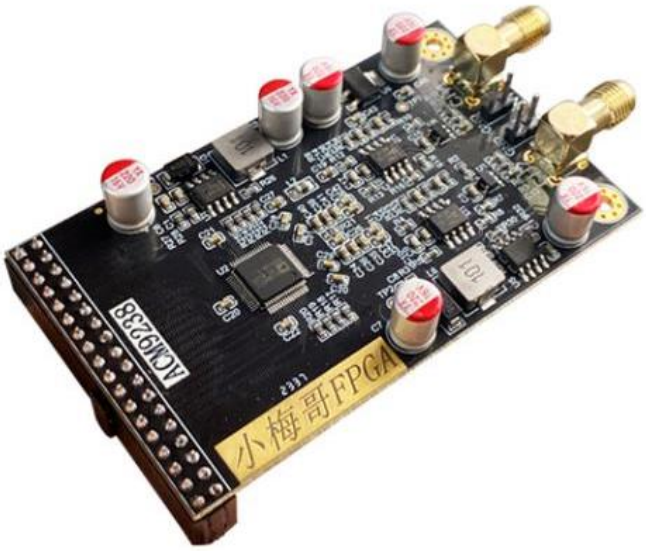

图 1-2 ACM9238 模块图

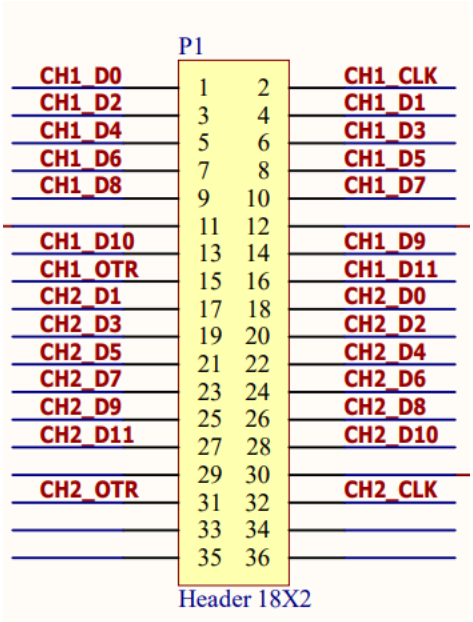

图 1-3 ACM9238 模块接口图

使用该模块时，仅需 FPGA 为每路 ADC 提供一路时钟信号，ADC 则会在每个时钟周期输出一个 12 位的采样结果。当 9238BSTZ 模拟输入端接-5V 至+5V 之间变化的正弦波电压信号时，其转换后的数据也是成正弦波波形变化，转换

波形如下图 1-4 所示，从图中可以看出 9238BSTZ 采集到的数据是无符号数据。

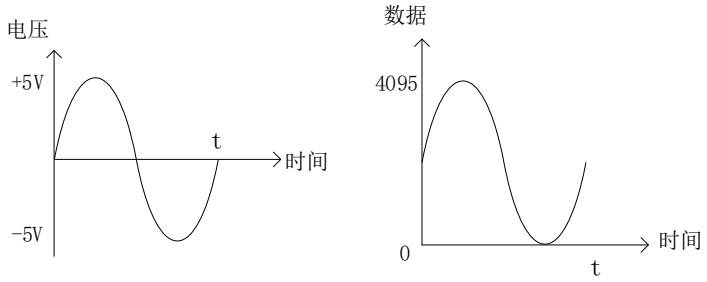

图 1-4 9238BSTZ 正弦波模拟电压值（左）、数据（右）

本模块采样率上限为 50Msps，采样率就等于 FPGA 提供给 ADC 的时钟频率。如需使用低于时钟频率的采样率，可以依旧给 ADC 提供 50MHz 的时钟信号，但在 FPGA 内部，对 50Msps 的采样结果数据进行抽取重采样的方法实现。比如期望以 1Msps 的采样速率采样，则只需要每间隔 50 个采样数据取一个结果存储或使用，其他 49 个数据直接舍弃，这样就能实现 1MSPS 的采样率了。十分不建议采用直接对提供给 ADC 芯片的时钟信号降频以实现降低采样率的效果的方法，因为时钟太低，会影响 ADC 芯片内部采样保持电路的工作情况，导致采样误差偏大。

本模块可用于小梅哥全系列 FPGA、SOC、Zynq 开发板，包括国产开发板和各核心板的评估底板。BX71、AC820、AC620、AC6102、ACX720、ACZ702、ACZ7015、智多晶 FPGA 开发板（AC208-SA5Z）、AC608 评估底板、AC601 评估底板、AC675 评估底板。

1.3 Block Design 与 ZYNQ 核

ZYNQ 核的全称为 ZYNQ7 Processing System，是围绕 Zynq-7000 处理系统的软件接口。该 IP 核充当 PS 和 PL 之间的逻辑连接，本次设计对 PS 端相关硬件资源的配置便需要通过该 IP 核完成。

在工程中添加并配置 ZYNQ 核的方式有两种，一种是通过 IP Catalog 创建，另一种则是通过 IP INTEGRATOR 的 Block Design 创建。考虑到设计的直观性，为了使设计更易理解，接下来我们通过创建 Block Design 的方式来为设计添加并配置 ZYNQ 核。

1.3.1.1 创建 Block Design

点击 IP INTEGRATOR 下的 Create Block Design 新建一个模块设计，在弹出的窗口中为模块设计命名为 system，如图 1-5 所示：

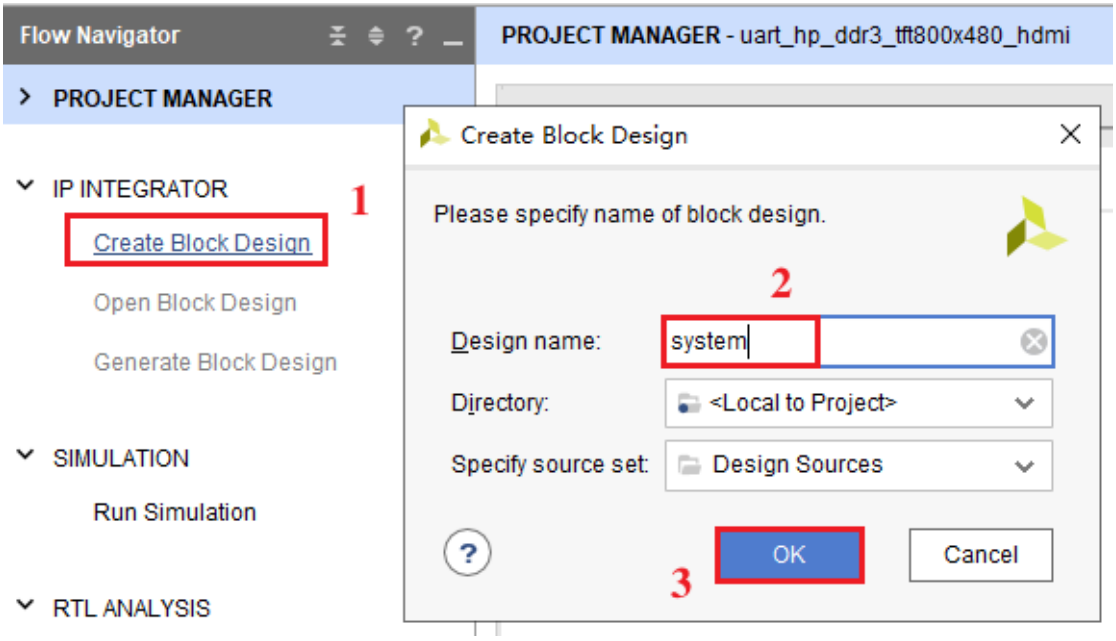

图 1-5 创建模块设计

创建好的 Block Design 界面如图 1-6 所示，左侧为工程资源窗口，能够查看工程设计结构、模块资源、接口信号等。右侧为视图窗口，用户添加的 IP 会直接展示在该窗口中，用户通过单击“+”为设计添加 IP。

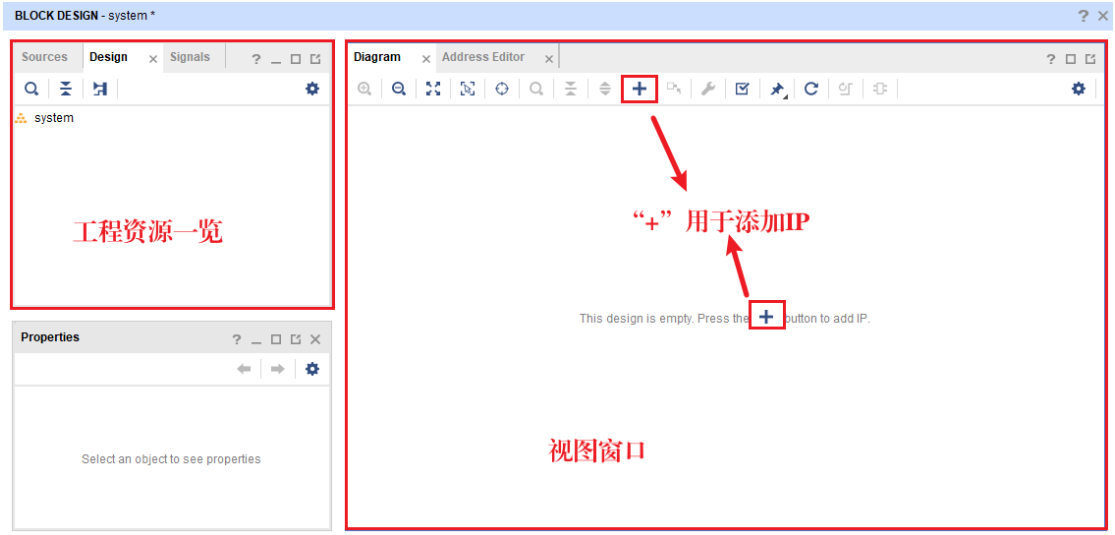

图 1-6 Block Design 界面

1.3.1.2 添加并配置 ZYNQ 核

点击“+”，在弹出的窗口中搜索 ZYNQ，会出现图 1-7 所示名为 ZYNQ7 Processing System 的 IP 核，双击添加 IP。

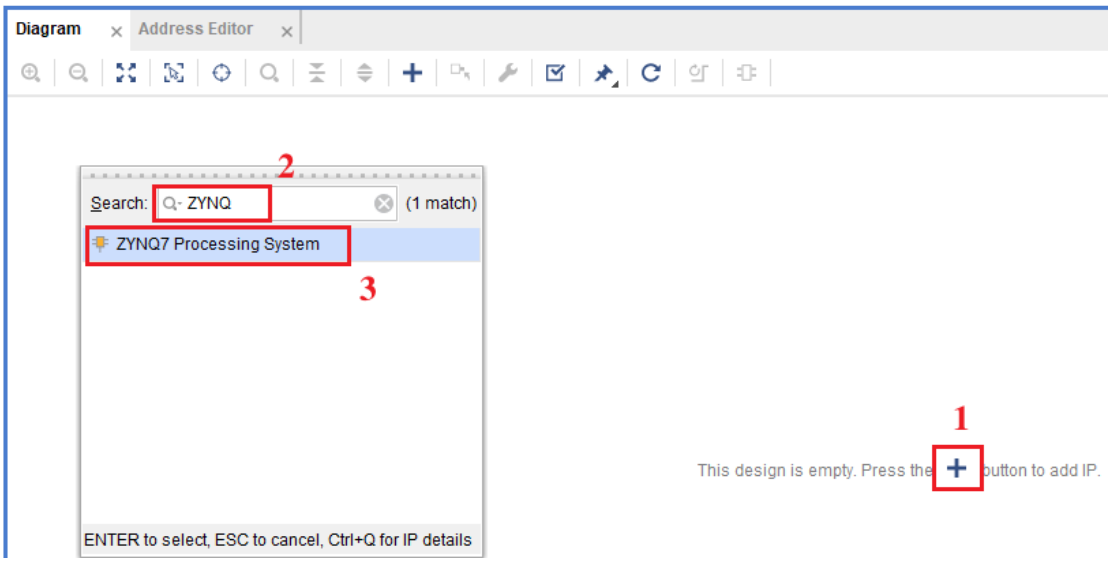

图 1-7 添加 ZYNQ 核

添加完成后的 ZYNQ 核如图 1-8 所示：

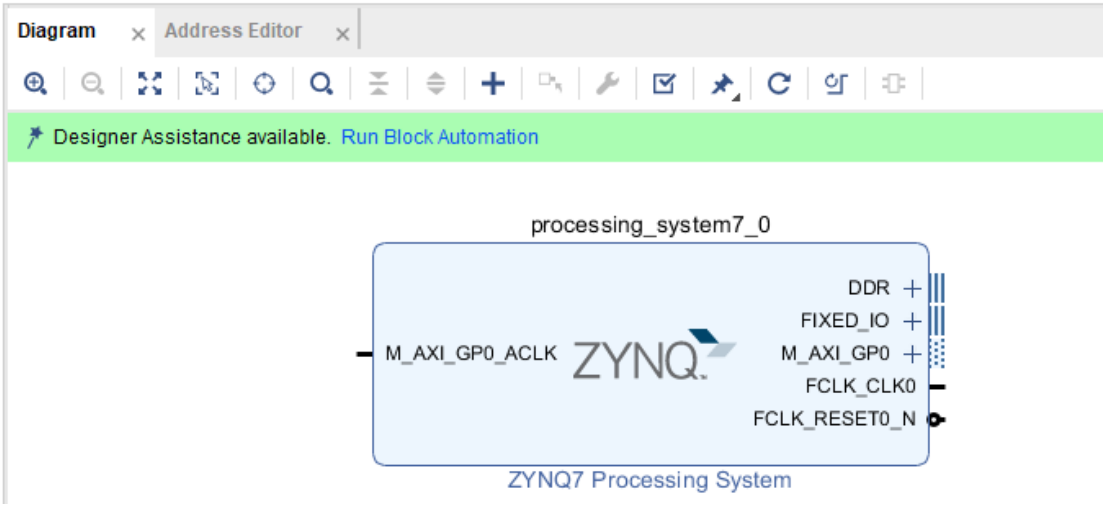

图 1-8 ZYNQ 核

此时的 ZYNQ 核还未配置，双击 IP 核进入到配置界面，如图 1-9 所示：

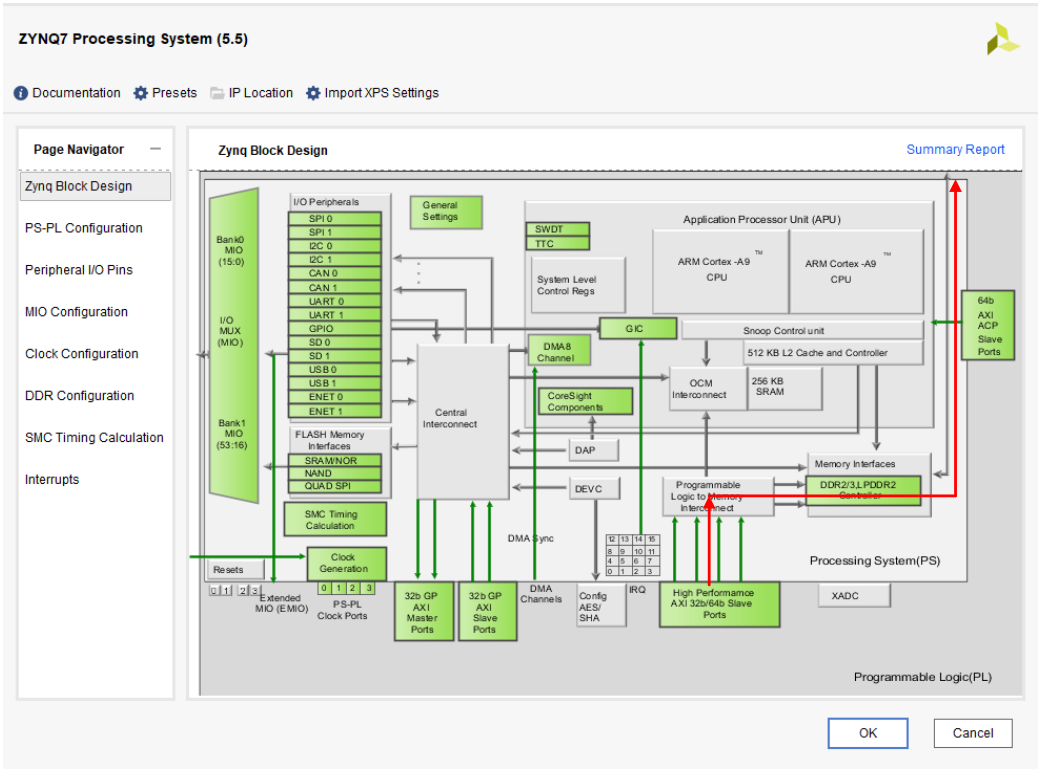

图 1-9 ZYNQ 核配置界面

本次设计我们需要借助 HP 接口来实现数据写入到 PS 侧的 DDR3，如图 1-9 中红色线条。点击图中绿色的 High Performance AXI 32b/64b Slave Ports 实现快速跳转，如图 1-10 所示：

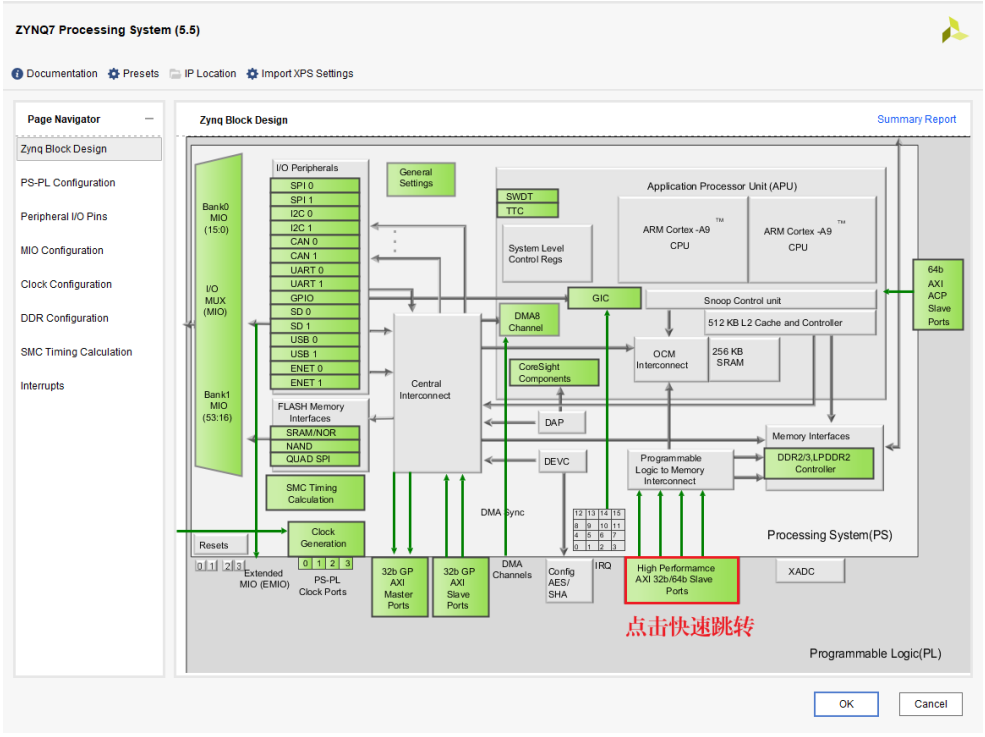

图 1-10 快速跳转

在跳转后的界面中勾选 S AXI HP0 interface，如图 1-11 所示：

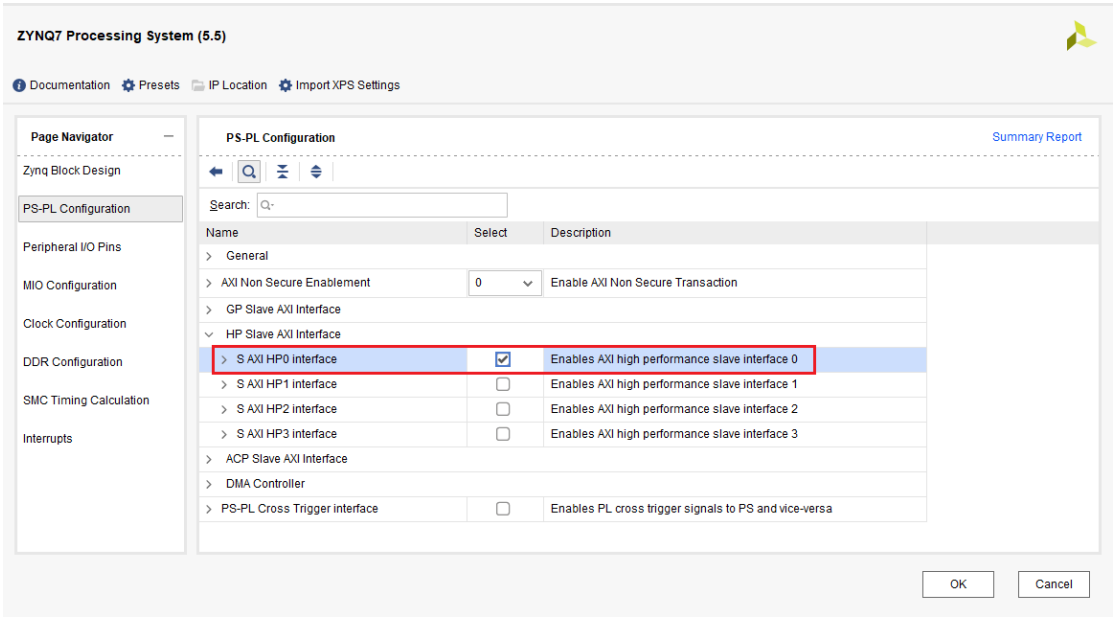

图 1-11 使能 HP0 接口

由于设计需要使用到 DDR3，接下来我们还需要配置 DDR 控制器。点击 DDR Configuration 跳转到对应界面，这里我们直接配置 DDR 型号为“MT41K256M16 RE-125”，位宽选择“16bit”如图 1-12 所示：

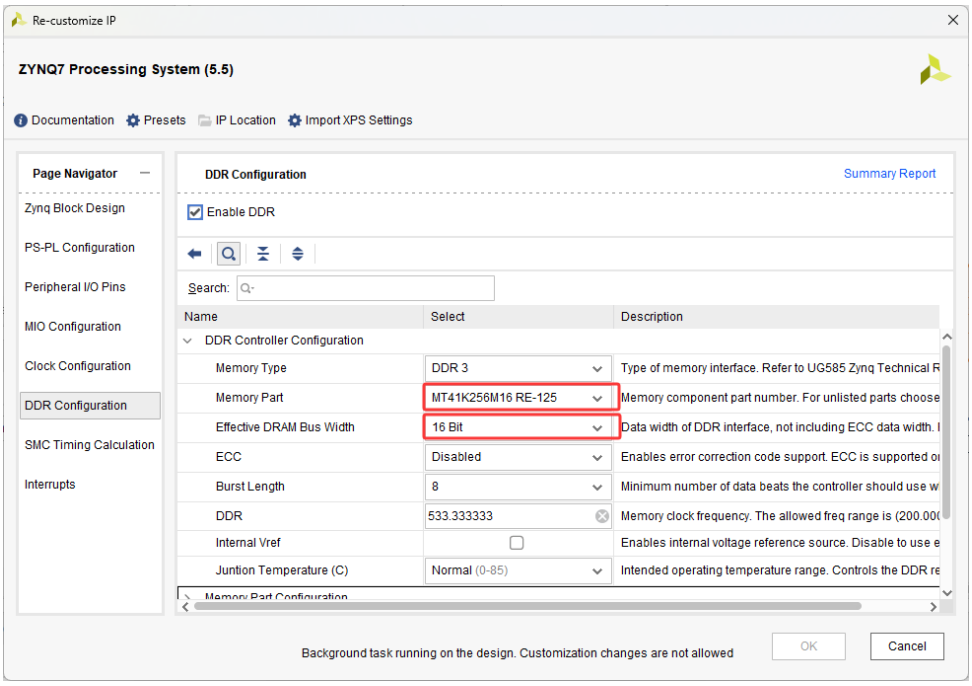

图 1-12 配置 DDR 型号

接着打开 Clock Configuration 页，勾选 PL Fabric Clocks 项下的 FCLK_CLK0

（默认勾选），如图 1-13 所示：

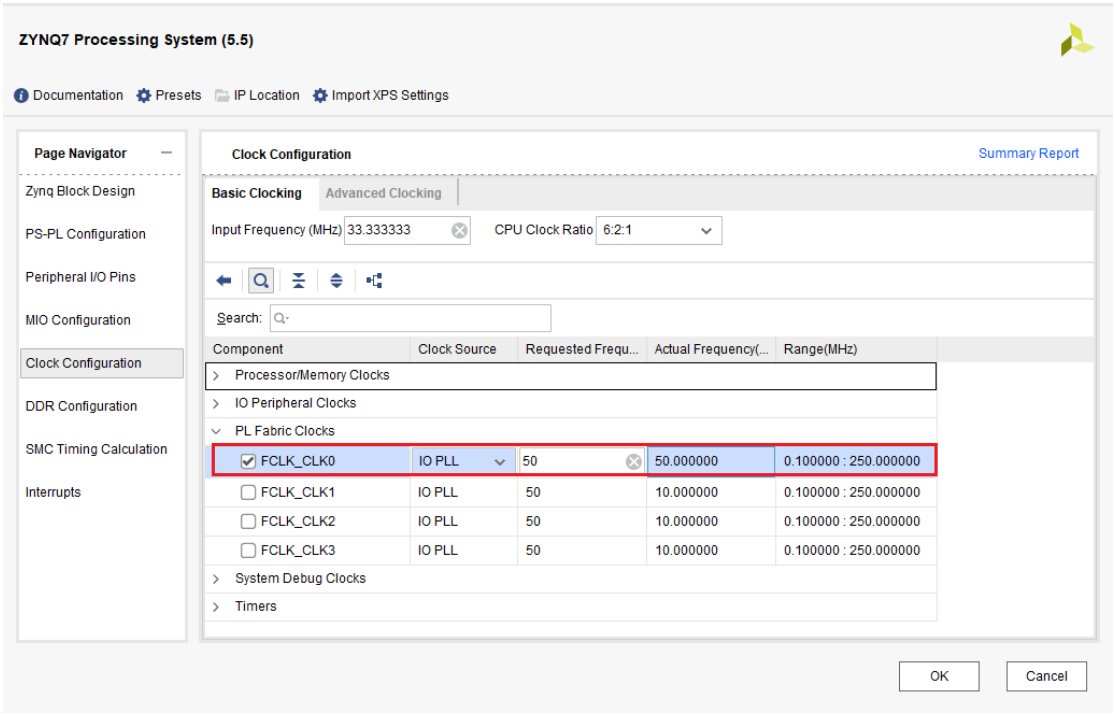

图 1-13 使能 PS 端输出时钟

该接口为 PS 端时钟输出接口，用来为 PL 提供时钟，这里我们保持默认的 50MHz 即可。关于该时钟如何实现，我们可以在 Advanced Clocking 中看到，计算方法如图 1-14 所示：

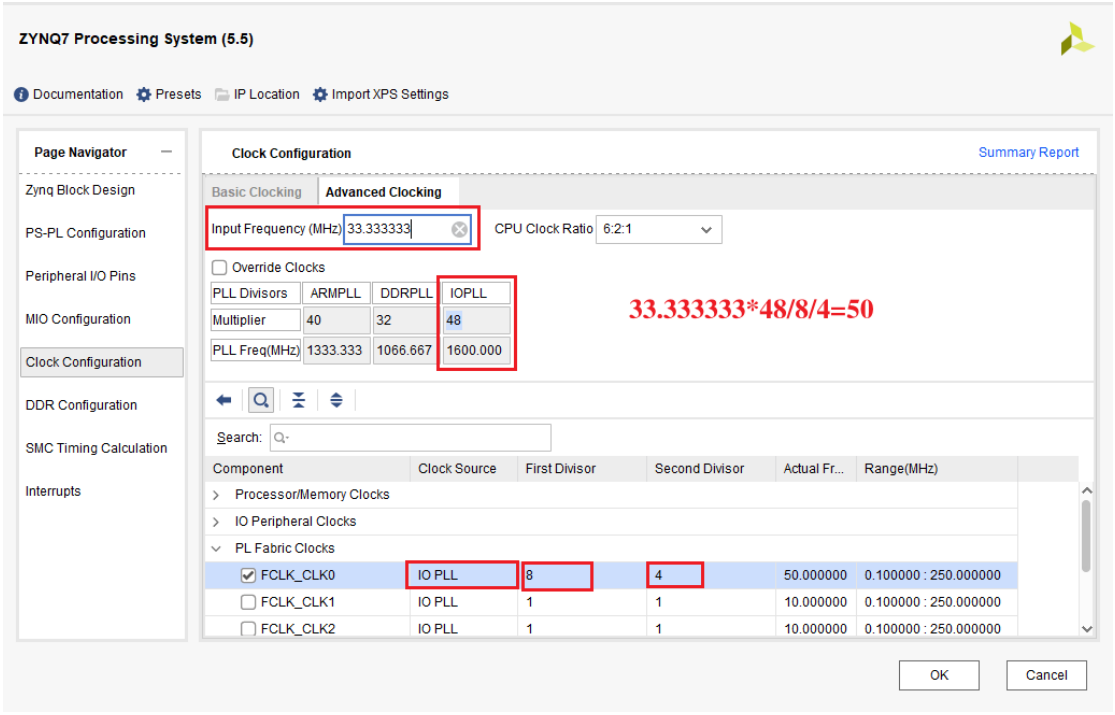

图 1-14 输出时钟分频与倍频系数

除了以上这些配置外，ZYNQ 核默认还为我们使能了一些接口，为了减少不必要的麻烦，这里我们需要取消其中 M AXI GP0 接口的使能，如图 1-15 所示：

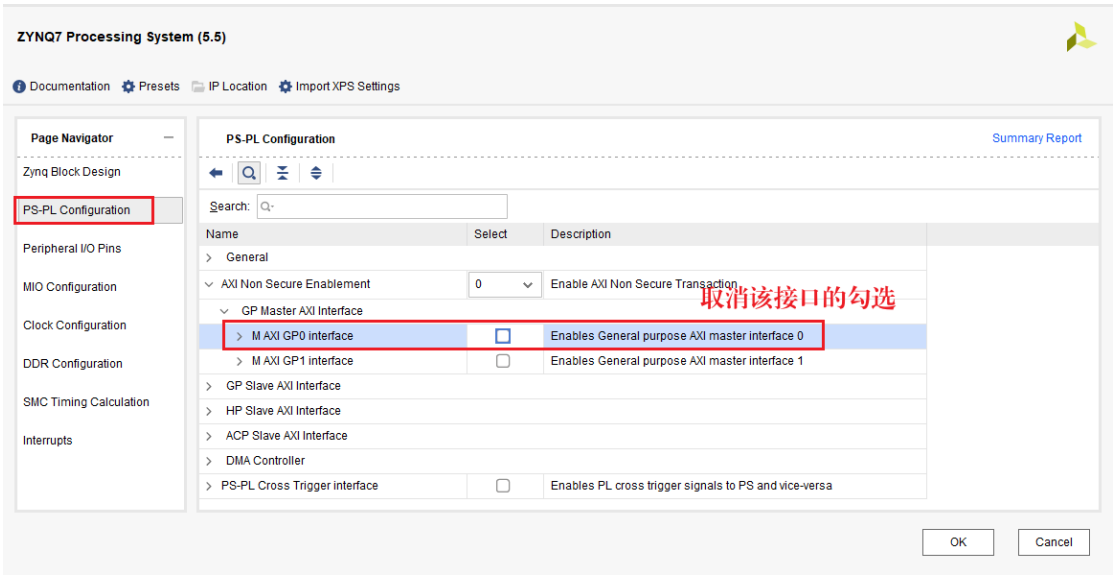

图 1-15 取消 M AXI GP0 接口

至此，我们便完成了对 ZYNQ 核的配置，点击 OK，配置完成后的 ZYNQ 核如图 1-16 所示：

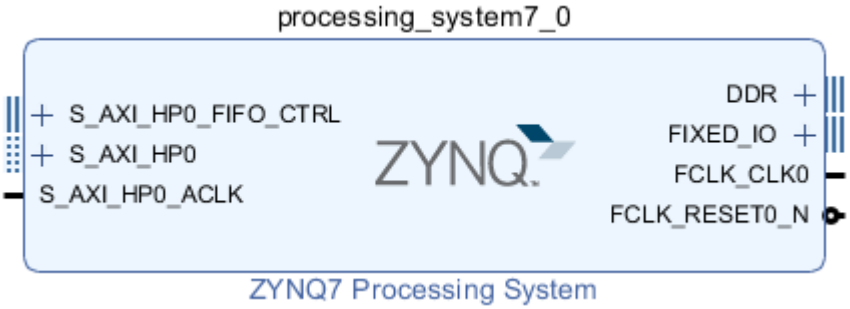

图 1-16 配置完成后 ZYNQ 核

1.3.1.3 添加并配置互联 IP

在本次设计中，PL 侧的数据在经过 fifo_axi4_adapter 模块转换成 AXI4 接口协议后通过 HP0 接口写入 PS 侧。但是，HP 接口使用的 AXI3 协议（双击 HP0 接口即可看到其接口协议），二者之间由于所使用的协议不同，不能直接连接。对于这种情况，就需要添加一个互联 IP 核，用以实现协议之间的转换。

使用快捷键 ctrl+i 搜索并添加 AXI smartconnect 核，随后配置 IP 的主从接口数为 1，其余项保持默认即可，如图 1-17 所示：

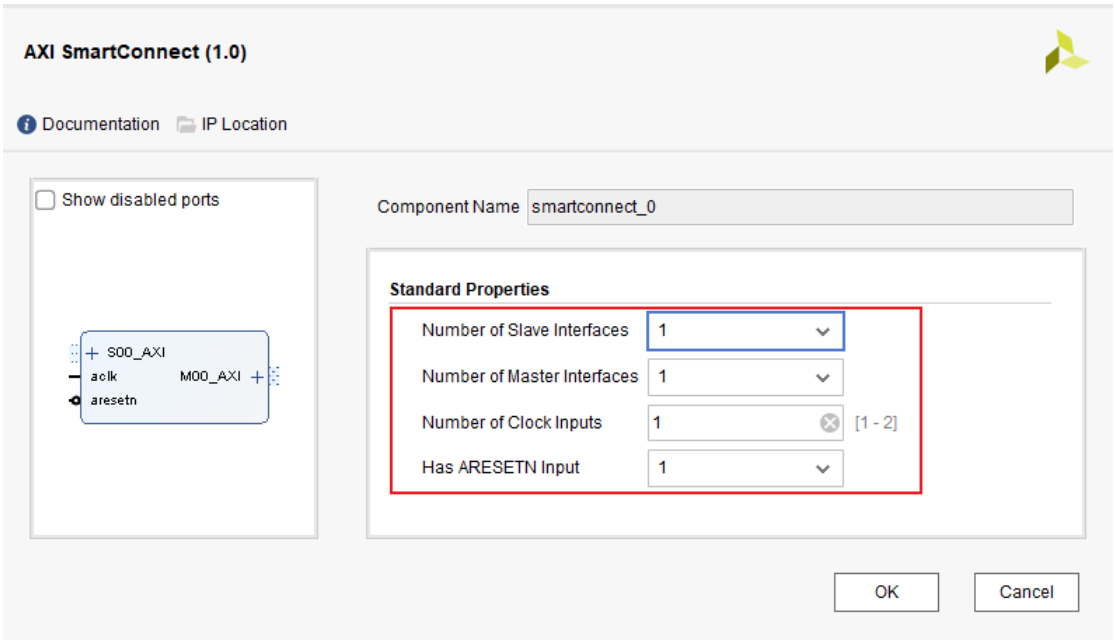

图 1-17 配置 AXI smartconnect 核

配置完成后的 AXI smartconnect 核如图 1-18 所示：

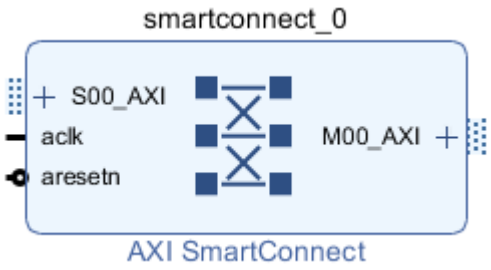

图 1-18 配置完成后的 AXI smartconnect 核

1.3.1.4 端口连接

对于设计来说，我们创建的 Block Design 就是一个模块，而我们添加在其中的 IP 核则是这个模块中的一个个子模块。为了让模块能够正常的工作，在配置完子模块后，就需要将它们彼此之间的接口联系起来，具体操作步骤如下：

1) 选中 ZYNQ 核的 FCLK_RESET0_N 接口，右键选择 Make External 导出该接口。该接口将作为 PS 侧输出给 PL 的复位信号，为了方便区分，在左侧的 External Port Properties 中将端口重命名为 ps2pl_resetn_0。操作步骤参考图 1-19：

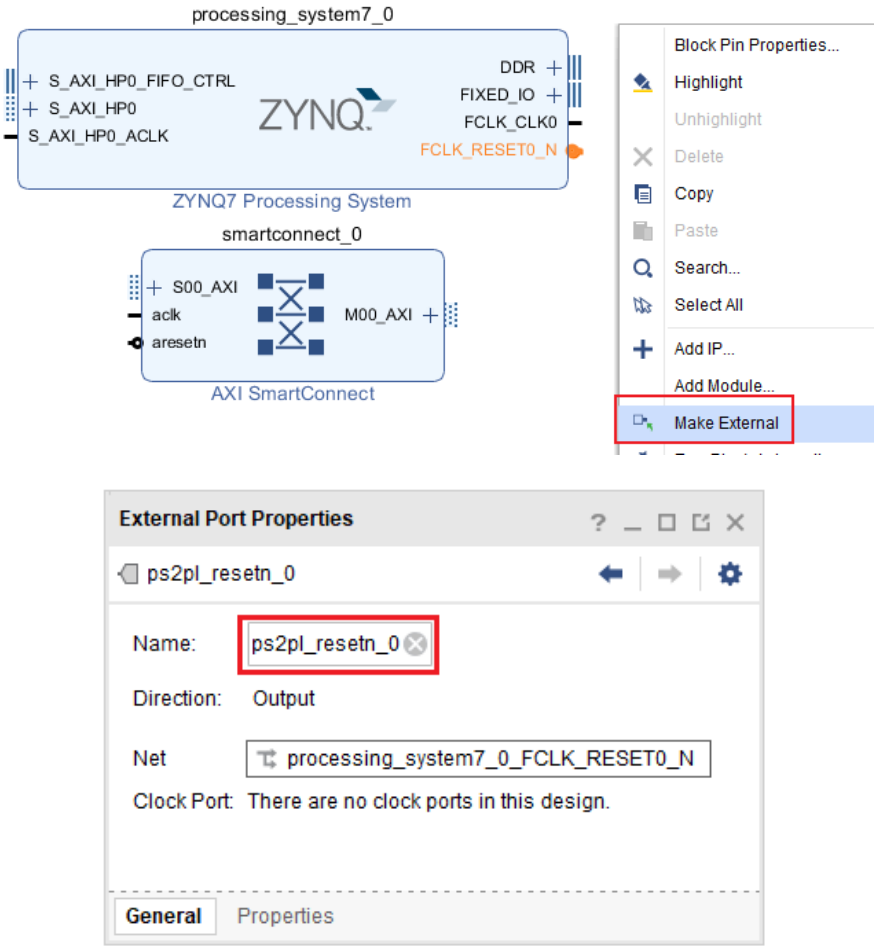

图 1-19 导出引脚与重命名

2) 依据步骤 1，导出 ZYNQ 核的 FCLK_CLK0 接口并重命名为 ps2pl_clk50m_0。随后如图 1-20 所示点击上方的 Run Block Automation 导出 DDR 以及 FIXED IO，这一部分是 PS 的相关引脚，导出即可。最终 zynq 核如图 1-21 所示：

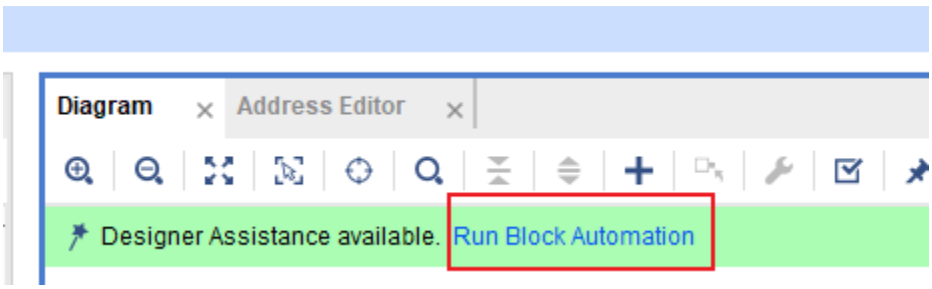

图 1-20 Run Block Automation

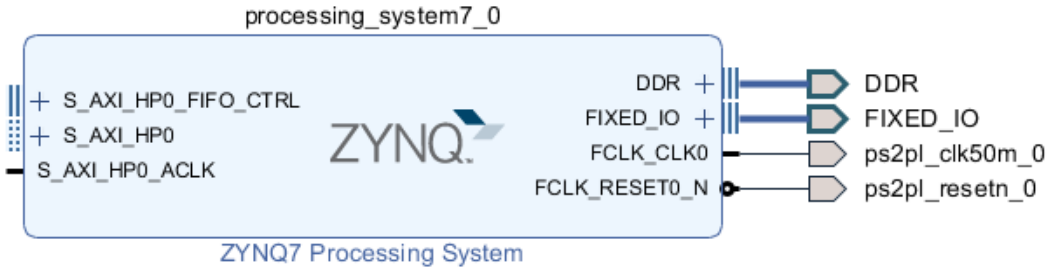

图 1-21 导出端口

3) 导出 AXI Smartconnect 核的 S00_AXI、aclk、aresetn 接口，并参考步骤 1，将信号依次重命名为 pl2ps_axi_0 、pl2ps_axi_aclk_0、pl2ps_axi_resetn_0。完成后如图 1-22 所示：

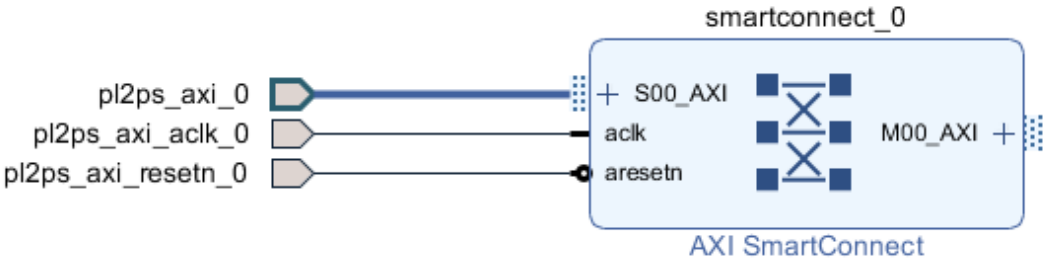

图 1-22 PL 输出给 PS 信号

4) 连接 HP 接口和 HP 接口工作时钟。即连接 M00_AXI 和 HP 接口，连接 aclk 和 S_AXI_HP0_ACLK 接口即可完成 block design 设计，最终完整设计如图 1-23 所示：

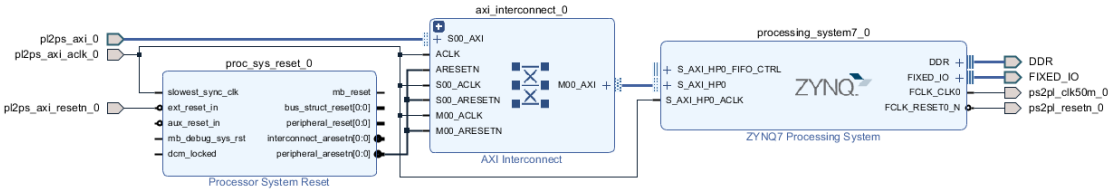

图 1-23 Block Design 完整结构

此时我们并不知道设计是否正确，因此点击上方的√来验证设计，如图 1-24 所示：

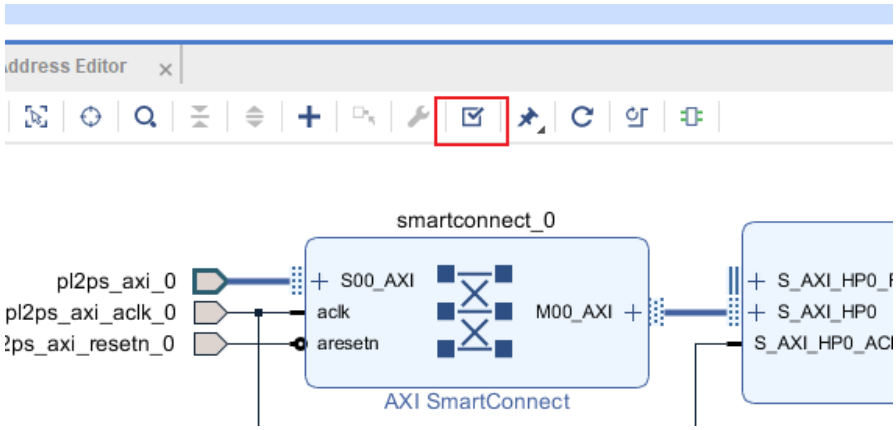

图 1-24 验证设计

此时可以看到，软件弹出严重警告，告诉我们 HP0_DDR_LOWOCM 接口并没有被映射到 pl2ps_axi_0 接口上，并告诉了我们解决方法，如图 1-25 所示：

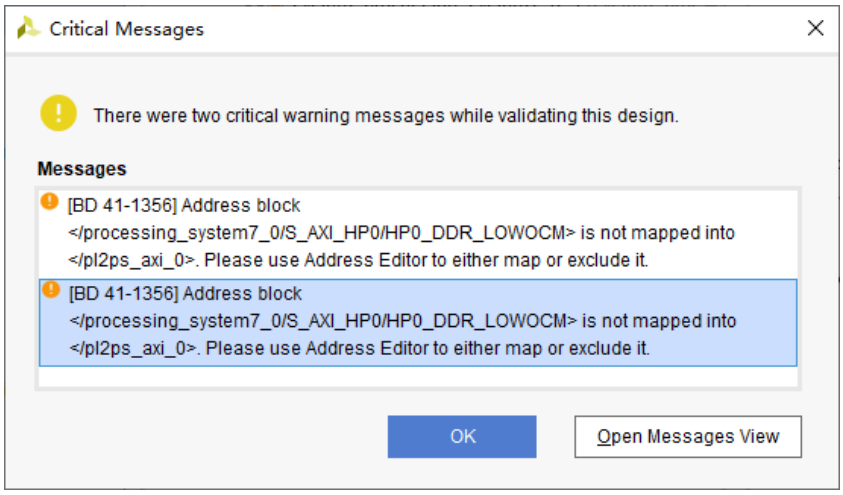

图 1-25 严重警告

此时点击上方的 Address Editor，展开 Unmapped Slaves 就能看到，此时的 HP0 并未被分配任何地址，如图 1-26 所示：

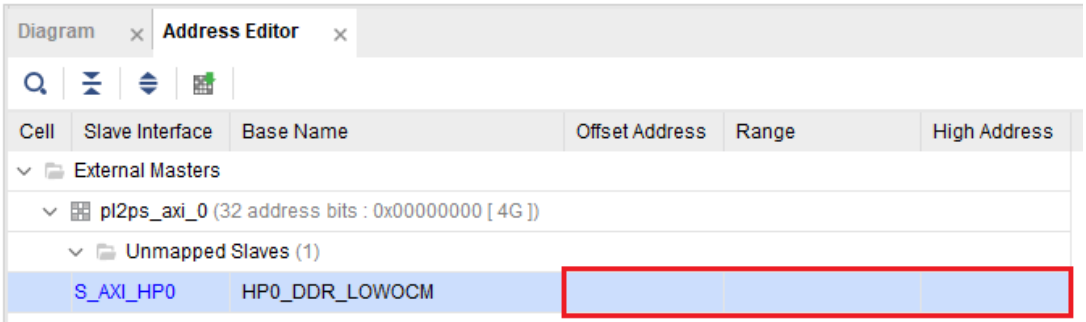

图 1-26 查看地址编辑器

点击上方的 Auto Assign Address，让软件自动帮我们分配地址，如图 1-27 所

示：

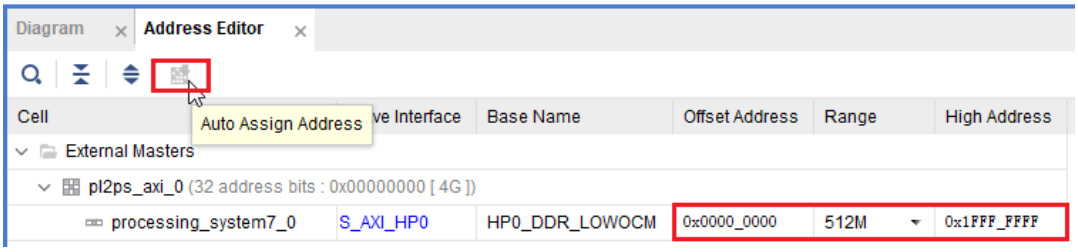

图 1-27 自动分配地址

再次点击√验证设计，可以看到，此时已经没有了警告，如图 1-28 所示：

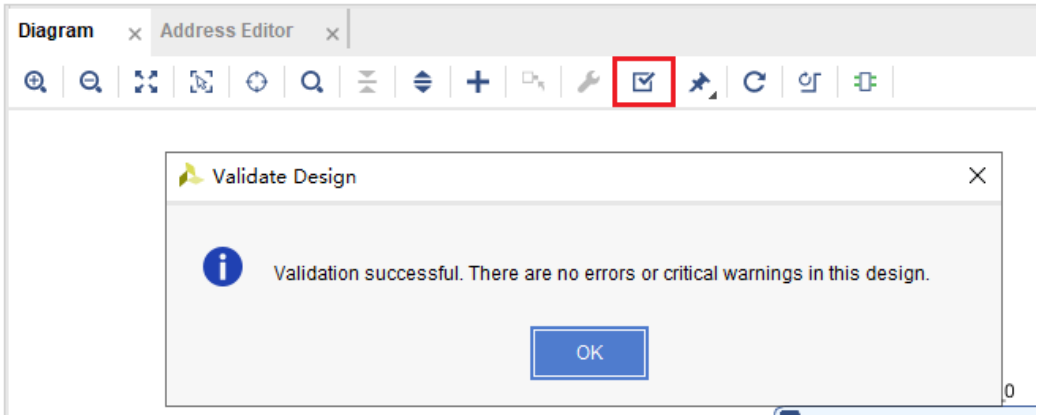

图 1-28 验证设计

1.3.1.5 创建 Block design 封装

经过以上操作，我们已经完成对“Block Design 模块”中各个“子模块”的配置和连接。但是，要想将“Block Design 模块”添加到我们整个设计中，我们还需要为 Block Design 创建一个封装。

在左侧的 Source 栏中，右键单击 system，在弹出的选项中选择生成输出产品，如图 1-29 所示：

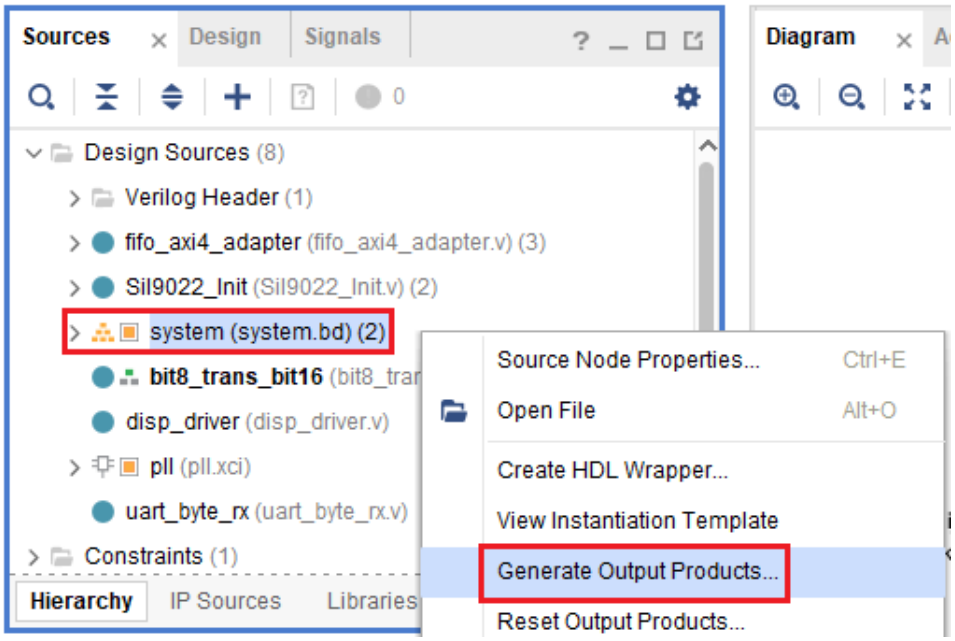

图 1-29 生成输出

在弹出的窗口中我们只需要设置 jobs 的数量为最大值即可（该值与电脑 CPU 性能有关），其余项保持默认，如图 1-30 所示：

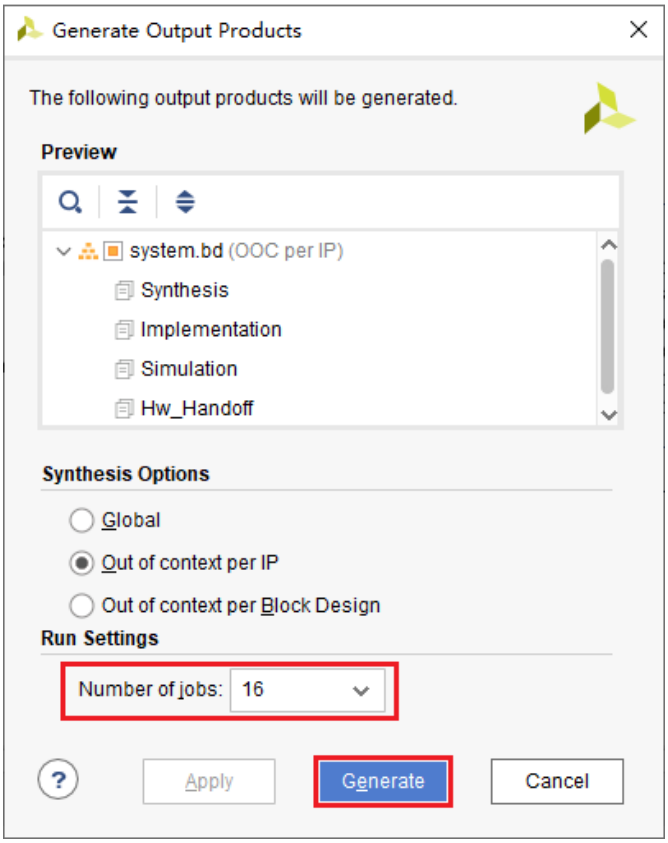

图 1-30 生成输出设置

随后等待输出生成完成，再次右键单击 system，选择 Create HDL Wrapper...，创建封装，如图 1-31 所示：

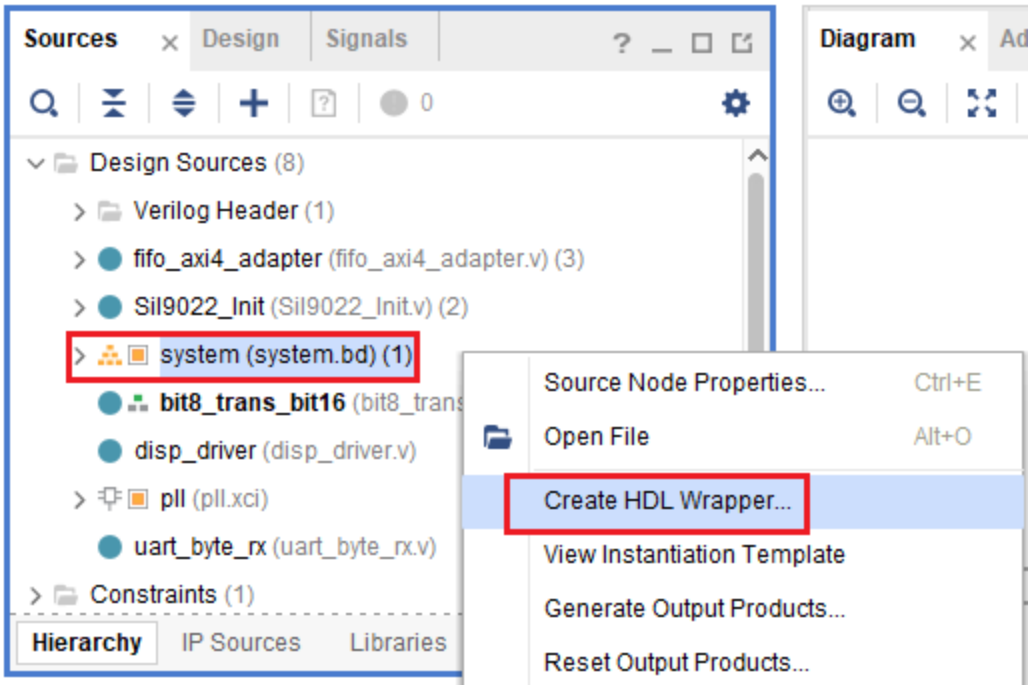

图 1-31 创建封装

在弹出的窗口中，我们选择让 vivado 来管理封装并自动更新，如图 1-32 所示：

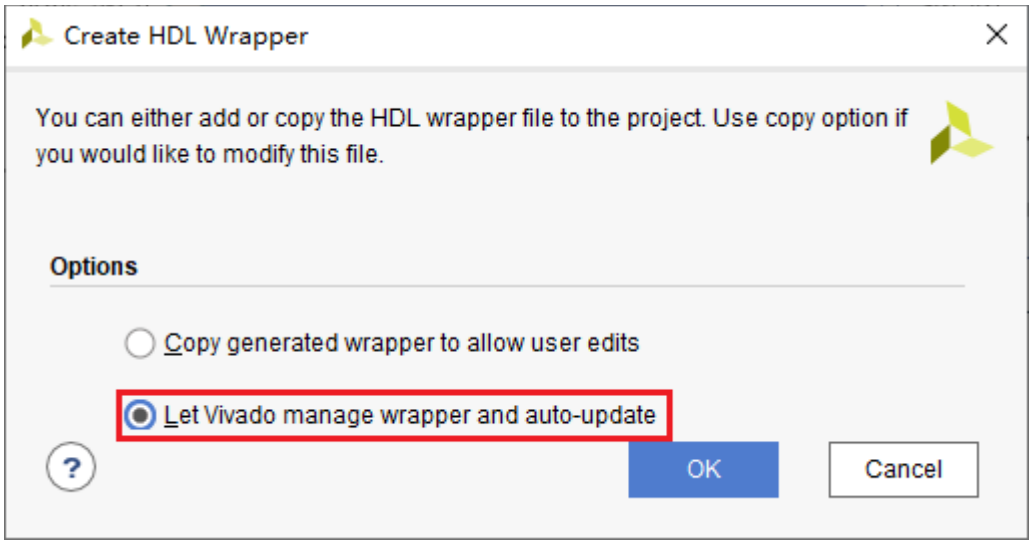

图 1-32 创建封装

点击 OK 后稍等片刻便能看到软件创建好的 system 的封装，如图 1-33 所示：

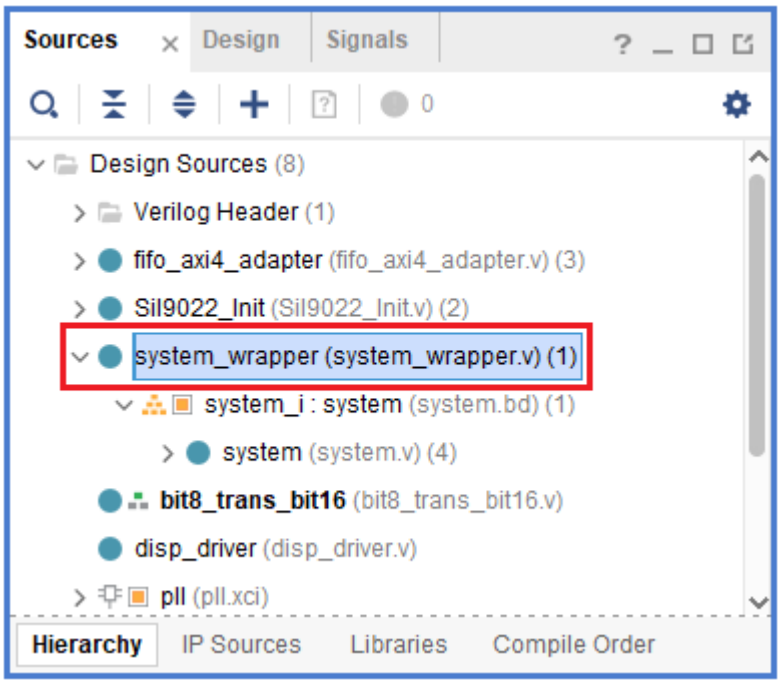

图 1-33 Block Design 封装

此时双击打开 system_wrapper.v，其中例化的便是我们的 system，例化部分代码如下：

```
system system_i
      (.DDR_addr(DDR_addr),
      .DDR_ba(DDR_ba),
      .DDR_cas_n(DDR_cas_n),
      .DDR_ck_n(DDR_ck_n),
      .DDR_ck_p(DDR_ck_p),
      .DDR_cke(DDR_cke),
      .DDR_cs_n(DDR_cs_n),
      .DDR_dm(DDR_dm),
      .DDR_dq(DDR_dq),
      .DDR_dqs_n(DDR_dqs_n),
      .DDR_dqs_p(DDR_dqs_p),
      .DDR_odt(DDR_odt),
      .DDR_ras_n(DDR_ras_n),
      .DDR_reset_n(DDR_reset_n),
      .DDR_we_n(DDR_we_n),
      .FIXED_IO_ddr_vrn(FIXED_IO_ddr_vrn),
      .FIXED_IO_ddr_vrp(FIXED_IO_ddr_vrp),
      .FIXED_IO_mio(FIXED_IO_mio),
      .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
      .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
      .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb),
      .pl2ps_axi_0_araddr(pl2ps_axi_0_araddr),
      .pl2ps_axi_0_arburst(pl2ps_axi_0_arburst),
```

```
        .pl2ps_axi_0_arcache(pl2ps_axi_0_arcache),
        .pl2ps_axi_0_arlen(pl2ps_axi_0_arlen),
        .pl2ps_axi_0_arlock(pl2ps_axi_0_arlock),
        .pl2ps_axi_0_arprot(pl2ps_axi_0_arprot),
        .pl2ps_axi_0_arqos(pl2ps_axi_0_arqos),
        .pl2ps_axi_0_arready(pl2ps_axi_0_arready),
        .pl2ps_axi_0_arsize(pl2ps_axi_0_arsize),
        .pl2ps_axi_0_arvalid(pl2ps_axi_0_arvalid),
        .pl2ps_axi_0_awaddr(pl2ps_axi_0_awaddr),
        .pl2ps_axi_0_awburst(pl2ps_axi_0_awburst),
        .pl2ps_axi_0_awcache(pl2ps_axi_0_awcache),
        .pl2ps_axi_0_awlen(pl2ps_axi_0_awlen),
        .pl2ps_axi_0_awlock(pl2ps_axi_0_awlock),
        .pl2ps_axi_0_awprot(pl2ps_axi_0_awprot),
        .pl2ps_axi_0_awqos(pl2ps_axi_0_awqos),
        .pl2ps_axi_0_awready(pl2ps_axi_0_awready),
        .pl2ps_axi_0_awsize(pl2ps_axi_0_awsize),
        .pl2ps_axi_0_awvalid(pl2ps_axi_0_awvalid),
        .pl2ps_axi_0_bready(pl2ps_axi_0_bready),
        .pl2ps_axi_0_bresp(pl2ps_axi_0_bresp),
        .pl2ps_axi_0_bvalid(pl2ps_axi_0_bvalid),
        .pl2ps_axi_0_rdata(pl2ps_axi_0_rdata),
        .pl2ps_axi_0_rlast(pl2ps_axi_0_rlast),
        .pl2ps_axi_0_rready(pl2ps_axi_0_rready),
        .pl2ps_axi_0_rresp(pl2ps_axi_0_rresp),
        .pl2ps_axi_0_rvalid(pl2ps_axi_0_rvalid),
        .pl2ps_axi_0_wdata(pl2ps_axi_0_wdata),
        .pl2ps_axi_0_wlast(pl2ps_axi_0_wlast),
        .pl2ps_axi_0_wready(pl2ps_axi_0_wready),
        .pl2ps_axi_0_wstrb(pl2ps_axi_0_wstrb),
        .pl2ps_axi_0_wvalid(pl2ps_axi_0_wvalid),
        .pl2ps_axi_aclk_0(pl2ps_axi_aclk_0),
        .pl2ps_axi_resetn_0(pl2ps_axi_resetn_0),
        .ps2pl_clk50m_0(ps2pl_clk50m_0),
       .ps2pl_resetn_0(ps2pl_resetn_0));
```

其中，DDR 与 FIXED_IO 相关端口为 PS 侧端口；pl2ps_axi 相关端口为我们导出的用于 PL 侧控制 AXI 总线将数据写入 PS 的相关端口；ps2pl 相关端口为我们导出的，PS 端输出给 PL 的控制信号端口。

1.4 模块设计

1.4.1 eth_receive_cmd 模块

eth_receive_cmd 模块将以太网接收到的数据进行解析，得到控制命令，最

终控制 ADC 采样数量、采样速率、采样通道等，该模块的基本结构框图如下图 1-34：

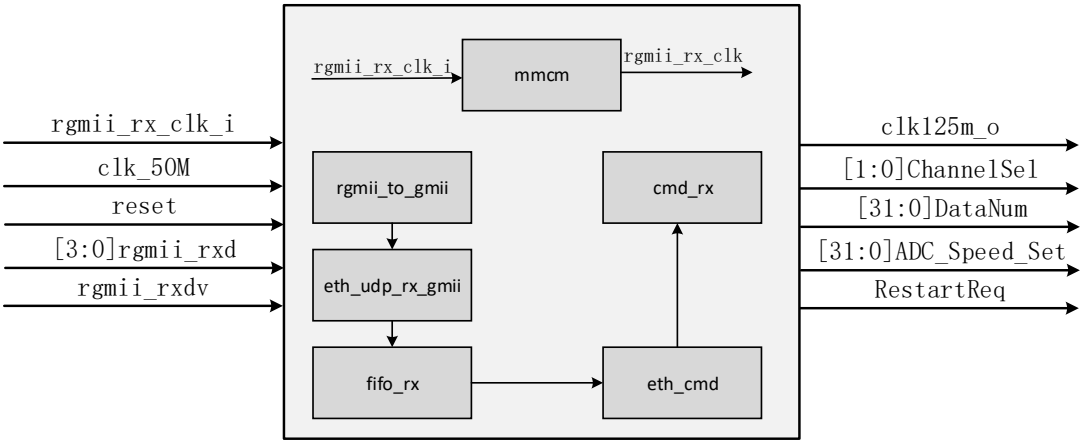

图 1-34 eth_receive_cmd 模块基本结构框图

上述图中的信号说明如下表 1-1 所示。

表 1-1 eth_receive_cmd 模块信号说明表

信号名称	I/O	信号意义
rgmii_rx_clk_i	I	以太网芯片接收时钟，125M
clk_50M	I	50M 时钟
reset	I	模块复位信号，低电平有效
rgmii_rxd[3:0]	I	以太网接收数据信号
rgmii_rxdv	I	接收数据有效信号
clk125m_o	O	输出 125M 的时钟信号
ChannelSel [1:0]	O	采样通道命令信号
DataNum [31:0]	O	采样数量命令信号
ADC_Speed_Set[31:0]	O	采样速率命令信号
RestartReq	O	启动采样命令信号

以太网接收模块 eth_udp_rx_gmii 和接口转换模块 rgmii_to_gmii 的设计在本章实验将不再进行说明，下面将对本次实验需要设计的模块和需要添加的 IP 进行说明。

1.4.1.1 mmcm 模块

锁相环模块，将 rgmii 接口时钟信号 rgmii_rx_clk_i 偏移 90° 得到 rgmii_rx_clk 时钟信号，这样做是为了在时钟信号的上升沿/下降沿取数据时，取得数据刚好是数据信号 rgmii_rxd 的正中间，使得采样的数据处于最稳定的状态，如下图 1-35 所示。

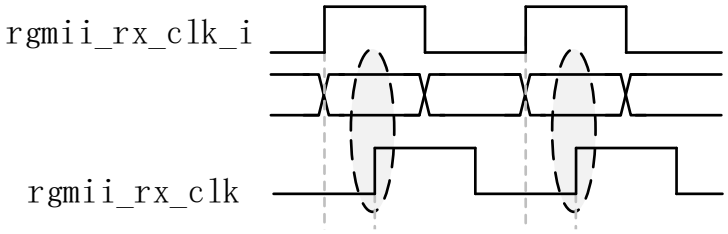

图 1-35 采集数据波形图

锁相环 IP 配置界面如下图 1-36 所示。

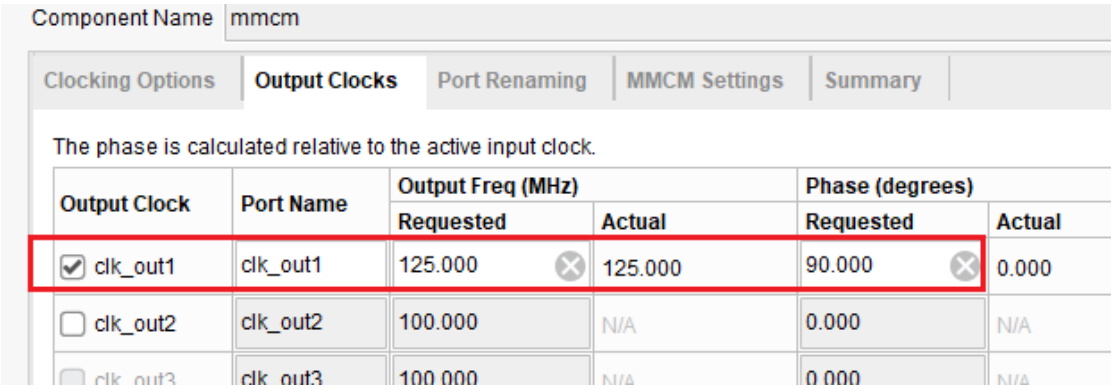

图 1-36 锁相环配置界面

1.4.1.2 fifo_rx 模块

FIFO IP 核，以太网接收模块工作时钟 125M，ACM9238 控制器工作时钟 50M，两者数据速率不匹配，使用该 IP 核解决采集过程中会出现的跨时钟域数据交互问题，所以配置时需要选择双端口，如下图 1-37 所示：

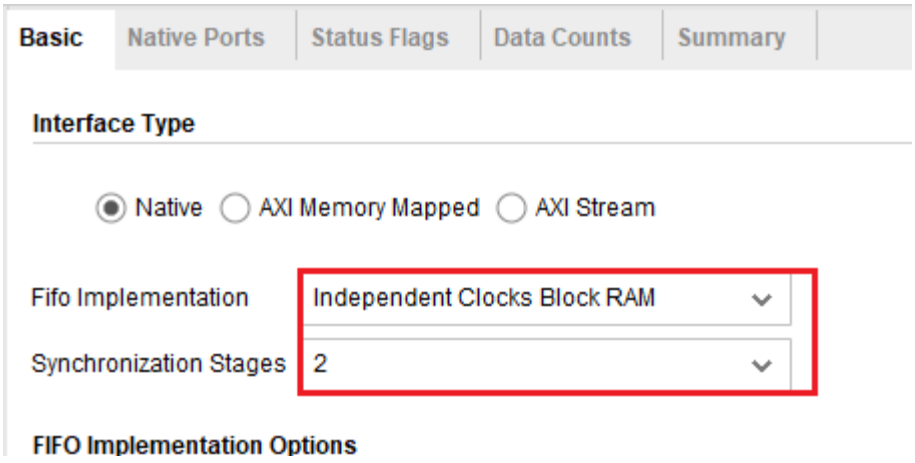

图 1-37 双端口 FIFO

然后配置读写位宽为 8 位（与以太网接收模块的数据位宽保持一致），写深度为 256，如下图 1-38 所示：

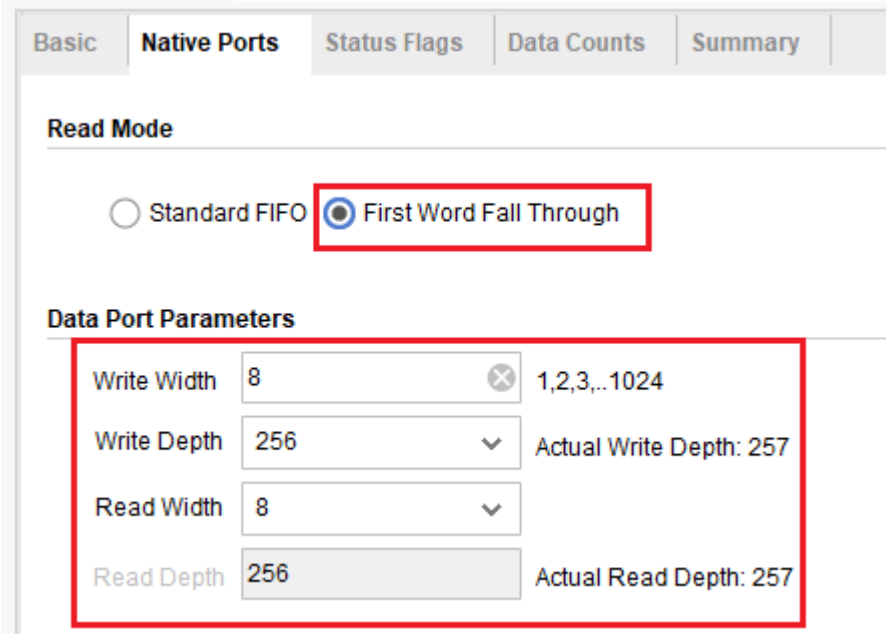

图 1-38 配置 FIFO 的读写位宽及深度

1.4.1.3 eth_cmd 模块

接收转命令模块 eth_cmd，将以太网传输过来的指令数据帧进行拆解，得到需要的指令数据传送给别的模块进行处理，该模块的结构框图如下图 1-39 所示。

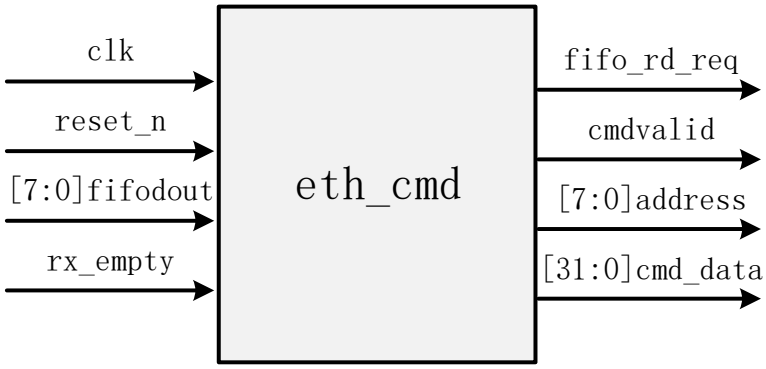

图 1-39 接收转命令模块框图

该模块的信号说明如下表 1-2 所示。

表 1-2 接收转命令模块信号说明表

信号名称	I/O	信号意义
clk	I	模块工作时钟
reset_n	I	模块复位信号，低电平有效
fifodout[7:0]	I	从 FIFO 中读出的 8 位数据
rx_empty	I	FIFO 为空标志信号
fifo_rd_req	O	FIFO 的读请求信号

cmdvalid	O	输出命令有效标志信号
address[7:0]	O	配置 AD9238 的寄存器地址信号
cmd_data[31:0]	O	写入到寄存器中的数据

网口一次发送的命令数据内容为 32 个字节，为了实现通过网口修改这些寄存器的值，需要对发送一次的数据进行拆解才能实现，对于设计的数据帧，一帧数据一共 8 个字节，包含帧头、帧尾、地址段、数据段。帧格式如下表 1-3 所示：

表 1-3 帧格式说明表

数据	D0	D1	D2	D3	D4	D5	D6	D7
功能	帧头 0	帧头 1	地址 address	data[31:24]	data[23：16]	data[15:8]	data[7:0]	帧尾
值	0x55	0xA5	XX	XX	XX	XX	XX	0xF0

从上表中可以看出，每帧数据一共 8 个字节，分别用 D0~D7 表示，其中，D0 和 D1 两个数据作为帧头，其值固定为 0x55、0xA5，D7 作为帧尾，其值固定为 0xF0。帧头和帧尾的作用是为了准确识别数据帧，确保接收的数据是我们需要分析的。D2 代表的是要操作的寄存器地址，D3 为要写入寄存器的数据的 24~31 位，D4 为要写入寄存器的数据的 16~24 位，D5 为要写入寄存器的数据的 8~15 位，D6 为要写入寄存器的数据的 0~7 位。

该模块的作用就是将网口接收到的数据拆解成上述帧格式，将 D2 作为地址 address 输出，指定修改哪个寄存器，D3~D6 共 32 位作为数据 data 输出，控制 AD9238 进行相应的配置。下面将对模块中的部分代码进行说明：

首先，产生 FIFO 的读请求信号。当检测到 FIFO 非空的时候，产生 FIFO 读请求信号，代码如下所示：

```
always@(posedge clk or negedge reset_n)
if(!reset_n)
  fifo_rd_req <= 1'b0;
else if(!rx_empty)
  fifo_rd_req <= 1'b1;
else
  fifo_rd_req <= 1'b0;
```

然后是得到帧命令数据，每产生一次读请求，将会从 FIFO 中读取一个 8 位的数据，连续存储 8 个字节的数据就得到一帧命令数据。代码如下所示：

```
always@(posedge clk)
if(fifo_rd_req)begin
  data_0[7] <= #1 fifodout;
  data_0[6] <= #1 data_0[7];
  data_0[5] <= #1 data_0[6];
  data_0[4] <= #1 data_0[5];
  data_0[3] <= #1 data_0[4];
```

```
  data_0[2] <= #1 data_0[3];
  data_0[1] <= #1 data_0[2];
  data_0[0] <= #1 data_0[1];
end
```

最后是判断得到的帧命令数据是否正确，当数据符合 D0 为 8'h55，D1 为 8'hA5，D7 为 8'hF0，则代表该数据格式正确，会生成一个指令正确信号 cmdvalid 输出到指令转控制模块，并将数据进行输出，代码如下所示：

```
always@(posedge clk or negedge reset_n)
if(!reset_n)begin
    address <= 0;
    cmd_data <= 32'd0;
    cmdvalid <= 1'b0;
end
else if(fifo_rx_done)begin
    if((data_0[0]==8'h55)&&(data_0[1]==8'hA5)&&(data_0[7]==8'hF0))
    begin
        cmd_data[7:0] <= #1 data_0[6];
        cmd_data[15:8] <= #1 data_0[5];
        cmd_data[23:16] <= #1 data_0[4];
        cmd_data[31:24] <= #1 data_0[3];
        address <= #1 data_0[2];
        cmdvalid <= #1 1;
    end
    else
        cmdvalid <= #1 0;
end
else
    cmdvalid <= #1 0;
```

1.4.1.4 cmd_rx 模块

指令转控制模块 cmd_rx 将从接收转命令模块接收到的数据转换为相应的控制数据，首先将对寄存器进行说明，其功能和地址分别如表 1-4 所示：

表 1-4 寄存器说明表

名称	地址	位宽	功能简介
RestartReq	0	1	重新开始采集请求寄存器，向该寄存器写入任意值即可启动新一轮的采样存储传输
ChannelSel	1	2	通道设置寄存器，共 2 位。ACM9238 模块提供了 ADC1、ADC2 两个通道进行数据采集。
DataNum	2	32	数据个数寄存器。如果采样 512 个数据，应该向寄存器中写入 0100。
ADC_Speed_Set	3	32	ADC 采样速率设置寄存器。如果设置为 0，采样和时钟保持一致 50M 时钟就是 50M 的采样速率，设置计数值后就可以改变采样频

			率，设置为 1 就是 25M。如果设置为 27 0F，换算成十进制是 9999，采样速率设置是 5k，计数值和采样频率之间的关系：设置计数值= Fclk/Fs – 1，Fs 是期望的采样率，Fclk 是系统时钟 50M。

指令转控制模块的结构框图如图 1-40 所示。

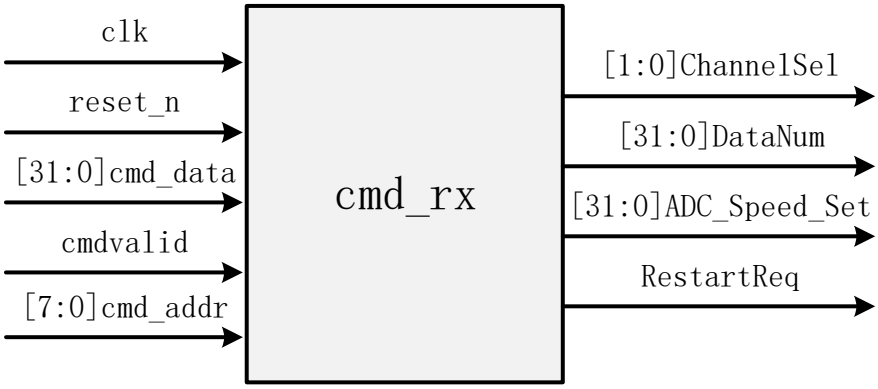

图 1-40 指令转控制模块结构框图

模块信号说明如表 1-5 所示。

表 1-5 指令转控制模块信号说明表

信号名称	I/O	信号意义
clk	I	模块时钟信号
reset_n	I	模块复位信号，低电平有效
cmd_data[31:0]	I	写入到寄存器中的值
cmdvalid	I	命令有效标志信号
cmd_addr[7:0]	I	寄存器地址信号
ChannelSel[1:0]	O	通道设置寄存器
DataNum[31:0]	O	数据个数寄存器
ADC_Speed_Set[31:0]	O	ADC 采样速率控制寄存器
RestartReq	O	重新开始采集请求信号

根据表 1-5 中的内容， 地址 cmd_addr 为 0 时，产生 RestartReq；cmd_addr 为 1 时，得到通道设置数据 cmd_data[1:0]；cmd_addr 为 2 时，得到需要采样的数量 cmd_data[31:0]；cmd_addr 为 3 时，得到设置的采样速率的值 cmd_data[31:0]，代码如下所示：

```
always@(posedge clk or negedge reset_n)
if(!reset_n)begin
    ChannelSel <= 2'b00;
    DataNum <= 32'd0;
    ADC_Speed_Set <= 32'd0;
    RestartReq <= 1'b0;
end
else if(cmdvalid)begin
  case(cmd_addr)
```

```
      0: RestartReq <= 1'b1;
      1: ChannelSel <= cmd_data[1:0];
      2: DataNum <= cmd_data[31:0];
      3: ADC_Speed_Set <= cmd_data[31:0];
  default:;
  endcase
end
else
      RestartReq <= 1'b0;
```

1.4.2 ad9238_ctrl 模块

ACM9238 控制器模块（ad9238_ctrl）的功能是控制 ADC 的采样速率，并且将 ADC 采集到的 12 位数据转换成 16 位的数据，以便于我们的电脑进行数据存储，该模块的基本结构如下图 1-41 所示：

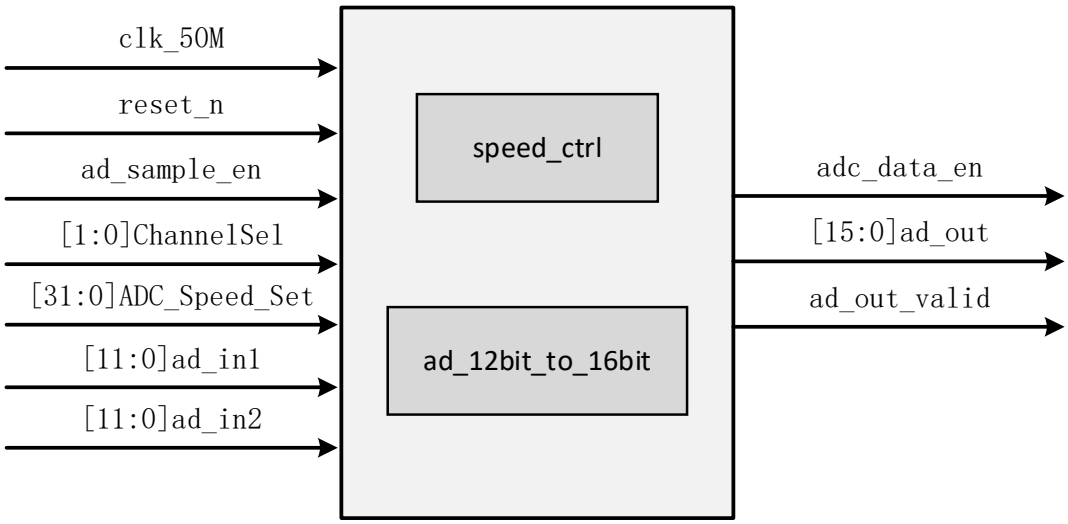

图 1-41 ad9238_ctrl 模块基本结构框图

ad9238_ctrl 模块的信号说明如下表 1-6 所示：

表 1-6 ad9238_ctrl 模块信号说明表

信号名称	I/O	信号意义
clk_50M	I	模块时钟信号，50M
reset_n	I	模块复位信号，低电平有效
ad_sample_en	I	输入 ADC 采样使能信号
ChannelSel[1:0]	I	采样通道命令信号
ADC_Speed_Set[31:0]	I	采样速度命令信号
ad_in1[11:0]	I	ACM9238 通道 1 采样到的 12 位数据
ad_in2[11:0]	I	ACM9238 通道 2 采样到的 12 位数据
adc_data_en	O	输出的 ADC 采样使能标志信号
ad_out[15:0]	O	输出的 16 位的 ADC 采样到的数据

ad_out_valid	O	ADC 输出数据有效信号

接下来将对该模块使用到的 speed_ctrl 模块和 ad_12bit_to_16bit 模块的功能和代码实现进行说明。

1.4.2.1 speed_ctrl 模块

采样速率控制（speed_ctrl）模块用来控制 ACM9238 的采样速率，该模块的结构框图如下图 1-42 所示。

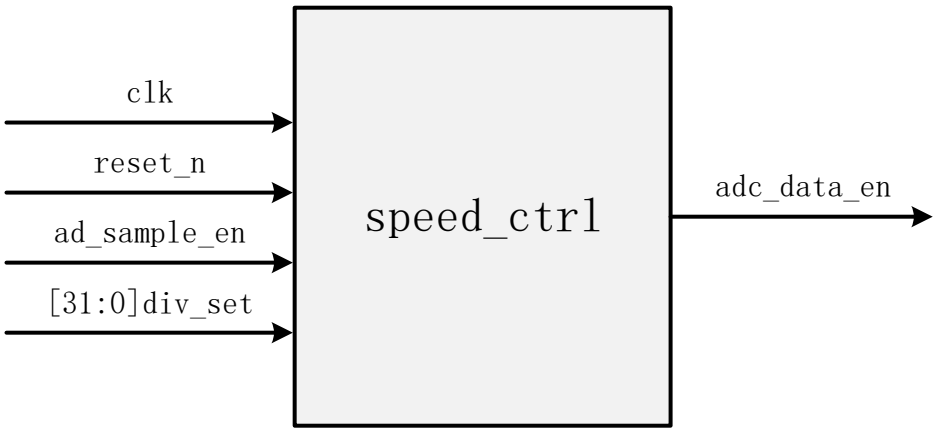

图 1-42 采样速率控制模块

对该模块的信号说明如下表 1-7 所示。

表 1-7 采样速率控制模块信号说明表

信号名称	I/O	信号意义
clk	I	模块时钟信号
reset_n	I	模块复位信号，低电平有效
ad_sample_en	I	输入的启动采样标志信号
div_set[31:0]	I	采样频率数据控制信号，div_set= Fclk/Fs – 1，Fs 是期望的采样率，Fclk 是系统时钟 50M
adc_data_en	O	ADC 采样结果存储使能信号

ACM9238 模块的最大采样速率为 50M，如需使用低于时钟频率的采样速率，可以依旧给 ADC 提供 50MHz 的时钟信号，但在 FPGA 内部，对 50Msps 的采样结果数据进行抽取重采样的方法实现。比如期望以 1Msps 的采样速率采样，则只需要每间隔 50 个采样数据取一个结果存储或使用，其他 49 个数据直接舍弃，这样就能实现 1MSPS 的采样率了。下面我们将编写相应代码实现上述功能。

设置一个计数器 div_cnt，当产生采样使能信号 ad_sample_en 之后，计数器加 1，当计数值等于设置的 div_set 的时候，将计数器清零。代码如下所示：

```
always@(posedge clk or negedge reset_n)
if(!reset_n)
```

```
    div_cnt <= 0;
else if(ad_sample_en)begin
    if(div_cnt >= div_set)
        div_cnt <= 0;
    else
        div_cnt <= div_cnt + 1'd1;
end
else
div_cnt <= 0;
```

计数器的计数值达到 div_set 的时候，使能 ADC 采样结果存储使能信号 adc_data_en，我们将该信号输出，最终实现每隔 div_set 个采样数据取一个结果存储或使用，从而达到对 ADC 采样频率的控制。代码如下所示：

```
always@(posedge clk or negedge reset_n)
if(!reset_n)
    adc_data_en <= 0;
else if(div_cnt == div_set)
    adc_data_en <= 1;
else
    adc_data_en <= 0;
```

1.4.2.2 ad_12bit_to_16bit 模块

数据采集模块 ACM9238 采集到的 12bit 数据不便于计算机存储，因为计算机对数据进行分析、存储的时候都以 8 位或 16 位数据作为统一的存储标准，所以我们需要通过数据位扩展模块（ad9238_12bit_to_16bit）将 12bit 数据转换成 16bit 数据进行存储。该模块的结构框图如下图 1-43 所示。

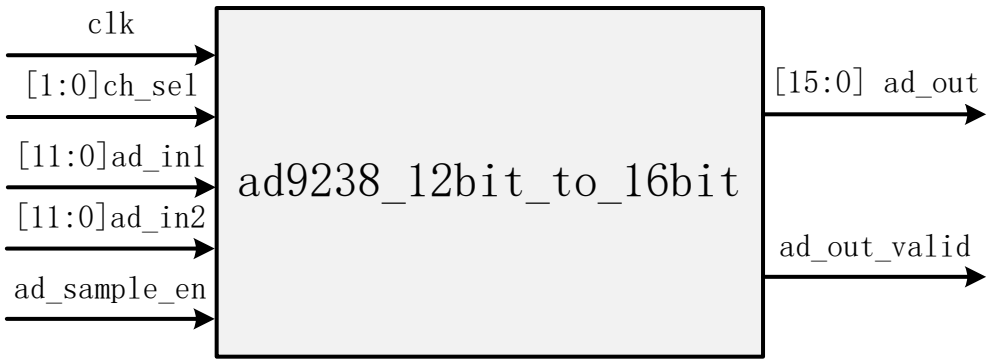

图 1-43 数据位扩展模块结构框图

对该模块的信号说明如下表 1-8 所示。

表 1-8 数据位扩展模块信号说明表

信号名称	I/O	信号意义

clk	I	模块时钟信号
ch_sel[1:0]	I	通道设置信号
ad_in1[11:0]	I	ACM9238 通道 1 的 12 位数据输入信号
ad_in2[11:0]	I	ACM9238 通道 2 的 12 位数据输入信号
ad_sample_en	I	控制 ADC 采样数据使能信号
ad_out[15:0]	O	16 位数据输出信号
ad_out_valid	O	输出数据有效信号

下面将编写模块实现代码。

首先，产生输出数据有效信号，将 ad_sample_en 信号给到 ad_out_valid，代码如下所示：

```
always @(posedge clk)
    ad_out_valid <= ad_sample_en;
```

然后将 ADC 采集到的无符号数据转换成有符号数据。如果采集的波形为+5V~-5V 的正弦波，ADC 模块最终输出的数据就为 4096~0 的正弦波，但是上位机在分析数据的时候需要数据是有符号的，在这里我们进行的操作就是将 ADC 采集得到的数据加上 2048，也就是将最高位取反，最后进行分析时将最高位作为符号位。举个例子，如果 ADC 采集到的数据分别为 0、511、1023，将这些数据分别加上 2048 之后得到的二进制值分别为 100000000000（-0）、1010 0000 0000（-511）、101111111111（+1023），这样将最高位作为符号位，采样的数据就变成了有符号的数据，从而可以提供给我们的上位机进行数据分析。代码如下所示：

```
assign s_ad_in1 = ad_in1 + 12'd2048;
assign s_ad_in2 = ad_in2 + 12'd2048;
```

最后模块根据选择通道（ch_sel）的不同，输出对应通道的数据。当 ch_sel= 2'b01（0x01），输出通道 1 的数据；当 ch_sel= 2'b10（0x02），输出通道 2 的数据。ADC 采集的数据是 12 位，这里通过补 0 的方式，实现 16 位的数据输出。代码如下所示：

```
always @(posedge clk)
if(ad_sample_en && ch_sel == 2'b01)
    ad_out<={4'd0,s_ad_in1};//这样补 0 为了适应上位机
else if(ad_sample_en && ch_sel == 2'b10)
    ad_out<={4'd0,s_ad_in2};//
else if(ad_sample_en && ch_sel == 2'b00)
    ad_out<={4'd0,adc_test_data};
else
    ad_out <= 16'd0;
```

1.4.3 state_ctrl 模块

DDR3 双端口转换模块控制信号的产生以及 ADC 何时启动数据传输都是通过 state_ctrl 模块控制的，该模块的结构框图如下图 1-44 所示。

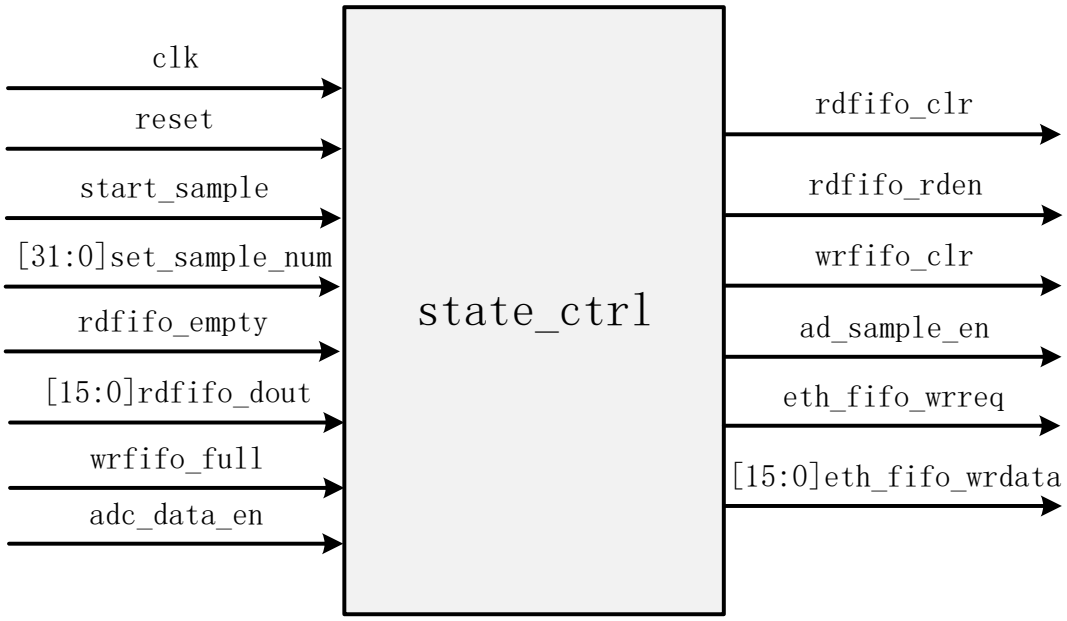

图 1-44 state_ctrl 模块结构框图

该模块的信号说明如下表 1-9 所示。

表 1-9 state_ctrl 模块信号说明表

信号名称	I/O	信号意义
clk	I	模块工作时钟
reset	I	模块复位信号，高电平有效
start_sample	I	ADC 模块开始采样标志信号
set_sample_num[31:0]	I	设置的采样个数
rdfifo_empty	I	读 FIFO 的读空标识信号，用于标识当前 FIFO 是否为空（即 FIFO 内有无数据）
rdfifo_dout[15:0]	I	读 FIFO 的数据输出，数据位宽为 16 位
wrfifo_full	I	写 FIFO 的写满标识信号，用于标识当前 FIFO 是否有被写满
adc_data_en	I	ADC 采样结果存储使能信号
wrfifo_clr	O	FIFO 的写清除信号，wrfifo_clr 向外打三拍输出，保证 wrfifo 的清零信号的生效节拍数
rdfifo_clr	O	读 FIFO 清空控制信号，给高电平表示执行清空，执行清空操作时，需保证给 3 个及以上个时钟（rdfifo_clk）周期的高电平
rdfifo_rden	O	读 FIFO 的读数据使能控制信号，给高电平表示往 FIFO 读数据，为避免读数据的丢失，确保在 FIFO 非空（rdfifo_empty =0）情况下读数据
ad_sample_en	O	ADC 采样使能标志信号
eth_fifo_wrreq	O	以太网发送缓存 FIFO 的写请求信号
eth_fifo_wrdata[15:0]	O	写入以太网发送缓存 FIFO 的 16 位数据

fifo_axi4_adapter 模块需要的控制信号有：wrfifo_clr、wrfifo_clk、wrfifo_wren、wrfifo_din、rdfifo_clr、rdfifo_clk、rdfifo_rden。上述信号中：wrfifo_clk、rdfifo_clk 应该与该模块的工作时钟保持一致，也就是和 ADC 数据采集模块的工作时钟保持一致为 50M；wrfifo_wren 信号为 ADC 数据位扩展模块输出数据有效信号 ad_out_valid；wrfifo_din 信号是 ADC 控制模块输出数据信号 ad_out，除去上述已经得到的控制信号外， state_ctrl 模块还需要产生的控制信号包括：wrfifo_clr、rdfifo_clr、rdfifo_rden。

根据上述描述，我们可以通过状态机的方式实现该模块的功能，状态定义如下所示：

```
localparam IDLE                 = 4'd0;   //空闲状态
localparam DDR_WR_FIFO_CLEAR    = 4'd1;   //DDR 写 FIFO 清除状态
localparam ADC_SAMPLE           = 4'd2;   //ADC 采样数据状态
localparam DDR_RD_FIFO_CLEAR    = 4'd3;   //DDR 读 FIFO 清除状态
localparam DATA_SEND_START      = 4'd4;   //数据发送状态
localparam DATA_SEND_WORKING    = 4'd5;   //数据发送完成状态
```

下面对每个状态的实现进行说明。

1. IDLE 状态

当处于空闲状态时，对 start_sample 采样起始位进行寄存，同时限定其只工作在状态 IDLE，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    start_sample_rm <= 1'b0;
else if(state==IDLE)
    start_sample_rm <= start_sample;
else
    start_sample_rm <= 1'b0;
end
```

当产生 start_sample_rm 信号之后跳转到 DDR_WR_FIFO_CLEAR 状态，代码如下所示：

```
begin
    if(start_sample_rm) begin   //DDR 初始化完成并且产生启动采样信号
        state <= DDR_WR_FIFO_CLEAR; //进入写 FIFO 清除状态
    end
    else begin
        state <= state;
    end
end
```

2. DDR_WR_FIFO_CLEAR 状态

当进入写 FIFO 清零状态后，开始清除写 FIFO 内的原始数据。设置清除 DDR 写 FIFO 的计数器，保证至少 10 拍的延时，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    wrfifo_clr_cnt<=0;
else if(state==DDR_WR_FIFO_CLEAR)//如果进入了清 fifo 状态
begin
    if(wrfifo_clr_cnt==9)
        wrfifo_clr_cnt<=5'd9;
    else
        wrfifo_clr_cnt<=wrfifo_clr_cnt+1'b1;
end
else
    wrfifo_clr_cnt<=5'd0;
end
```

然后等待 wrfifo_full（写端 fifo 满信号）的信号拉低，拉低后，表示可以往 FIFO 里写入数据，此时进入下一个状态。在清空（复位）FIFO 的时候，FIFO 的 full 信号会变高，可以认为在复位 FIFO 时是不允许对 FIFO 进行写操作的，即使写也是不可靠的，等 FIFO 的复位结束后，full 信号会变低，就允许对 FIFO 进行写操作。写 FIFO 清除状态代码如下所示：

```
begin
    if(!wrfifo_full && (wrfifo_clr_cnt==9))
        state<=ADC_SAMPLE;
    else
        state<=DDR_WR_FIFO_CLEAR;
end
```

当处于 DDR_WR_FIFO_CLEAR 状态时，我们需要产生清除写 FIFO 的信号 wrfifo_clr，由三拍延时信号拉高提供，之所以提供的延迟信号时间为 3 拍，是为了给清 FIFO 信号足够的拉高时间，以保证清空指令的可靠，也就是在 wrfifo_clr_cnt 为 0、1 或 2 时，wrfifo_clr 置 1，否则 wrfifo_clr 为 0。

```
always@(posedge clk or posedge reset)begin
if (reset)
    wrfifo_clr<=0;
else if(state==DDR_WR_FIFO_CLEAR)
    begin
        if(wrfifo_clr_cnt==0||wrfifo_clr_cnt==1||wrfifo_clr_cnt==2)
            wrfifo_clr<=1'b1;
        else
            wrfifo_clr<=1'b0;
    end
else
    wrfifo_clr<=1'b0;
```

```
end
```

3. ADC_SAMPLE 状态

进入 ADC 采样数据状态之后，首先设置 ADC 采样个数计数器 adc_sample_cnt，然后计数器根据 ADC 输出数据使能信号 adc_data_en 进行计数，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    adc_sample_cnt<=1'b0;
else if(state==ADC_SAMPLE)begin
    if(adc_data_en)
        adc_sample_cnt<=adc_sample_cnt+1'b1;
    else
        adc_sample_cnt<=adc_sample_cnt;
end
else
    adc_sample_cnt<=1'b0;
end
```

当 adc_sample_cnt 达到设定的采样数据个数，并且 adc_data_en 有效的时候时，ADC 模块数据采集完成，跳转到读 FIFO 清除状态，代码如下所示：

```
begin
    if((adc_sample_cnt>=set_sample_num-1'b1)&& adc_data_en)
        state<=DDR_RD_FIFO_CLEAR;
    else
        state<=state;
end
```

当处于 ADC_SAMPLE 状态时，我们还需要产生采样使能信号 ad_sample_en 给到其他模块使用，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    ad_sample_en<=0;
else if(state==ADC_SAMPLE)
    ad_sample_en<=1;
else
    ad_sample_en<=0;
end
```

4. DDR_RD_FIFO_CLEAR 状态

进入清 FIFO 状态之后，首先设置清除读 FIFO 的计数器 rdfifo_clr_cnt，保证至少 10 拍的延时。代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
```

```
    rdfifo_clr_cnt<=0;
else if(state==DDR_RD_FIFO_CLEAR)//如果进入了清 fifo 状态
begin
    if(rdfifo_clr_cnt==9)
        rdfifo_clr_cnt<=5'd9;
    else
        rdfifo_clr_cnt<=rdfifo_clr_cnt+1'b1;
end
else
    rdfifo_clr_cnt<=5'd0;
end
```

然后等待 rdfifo_empty（读端 FIFO 的空信号）信号拉低，拉低后，表示 FIFO 里已经有被写入数据，此时进入下一个状态。在清空(复位) FIFO 的时候，FIFO 的 empty 信号会变高，可以认为在复位 FIFO 时是不允许对 FIFO 进行读操作的，即使读也是不可靠的，等 FIFO 的复位结束后，FIFO 被写入数据后，empty 信号会变低，就允许对 FIFO 进行读操作，然后跳转到 DATA_SEND_START 状态，读 FIFO 清除状态代码如下所示：

```
begin
    if(!rdfifo_empty && (rdfifo_clr_cnt==9))begin
        state<=DATA_SEND_START;
    end
    else
        state<=state;
end
```

当处于 DDR_WR_FIFO_CLEAR 状态时，我们需要产生清除读 FIFO 的信号 rdfifo_clr，由三拍延时信号拉高提供，之所以提供的延迟信号时间为 3 拍，是为了给清 FIFO 信号足够的拉高时间，以保证清空指令的可靠，也就是在 rdfifo_clr_cnt 为 0、1 或 2 时，rdfifo_clr 置 1，否则 rdfifo_clr 为 0。

```
always@(posedge clk or posedge reset)begin
if (reset)
    rdfifo_clr<=0;
else if(state==DDR_RD_FIFO_CLEAR)
begin
    if(rdfifo_clr_cnt==0||rdfifo_clr_cnt==1||rdfifo_clr_cnt==2)
        rdfifo_clr<=1'b1;
else
        rdfifo_clr<=1'b0;
end
else
    rdfifo_clr<=1'b0;
end
```

5. DATA_SEND_START

进入DATA_SEND_START 状态之后，需要从 DDR 中读出数据，准备接下来的数据发送。所以这里我们产生 rdfifo_rden 信号，并将 state 直接跳转到数据发送状态，启动传输，代码如下所示：

```
begin
    state <= DATA_SEND_WORKING;
   rdfifo_rden <= 1'b1;
 end
```

6. DATA_SEND_WORKING 状态

进入数据发送状态之后，产生 rdfifo_rden 信号，将 DDR 中的数据读取出来，当读出的数据达到需要采集的数据信号时，跳转到 IDLE 状态，完成一次数据传输，代码如下所示：

```
begin
    if(send_data_cnt >= set_sample_num-1'b1) begin
        rdfifo_rden <= 1'b0;
        state <= IDLE;
    end
    else begin
        rdfifo_rden <= 1'b1;
        state <= DATA_SEND_WORKING;
    end
end
```

send_data_cnt 在 rdfifo_rden 有效的情况下进行计数，从而统计从 DDR 中读取的数据个数，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    send_data_cnt<=32'd0;
else if(state==IDLE)
    send_data_cnt<=32'd0;
else if(rdfifo_rden)
    send_data_cnt<=send_data_cnt+1;
else
   send_data_cnt<=send_data_cnt;
end
```

最后，由于 fifo_axi4_adapter 模块中，读 FIFO 是标准模式，数据相较于读请求信号会迟一拍，因此，这里对 rdfifo_rden 信号打拍，产生 rdfifo_rden_pre 信号。每当 rdfifo_rden_pre 信号有效的同时，读 FIFO 便会被读出一个数据。此时，便可以产生 eth_fifo 写请求信号，并将 DDR 读出的数据存放至以太网发送缓存 FIFO 中，代码如下所示：

```
reg rdfifo_rden_pre;
always@(posedge clk)
```

```
    rdfifo_rden_pre <= rdfifo_rden;

always@(posedge clk or posedge reset)
if(reset) begin
    eth_fifo_wrreq <= 1'b0;
    eth_fifo_wrdata <= 'd0;
end
else if(rdfifo_rden_pre) begin
    eth_fifo_wrreq <= 1'b1;
    eth_fifo_wrdata <= rdfifo_dout;
end
else begin
    eth_fifo_wrreq <= 1'b0;
    eth_fifo_wrdata <= 'd0;
end
```

1.4.4 fifo_axi4_adapter 模块

该模块用于实现 FIFO 接口与 AXI4 接口的转换，设计中用于读写 DDR。有关该模块的讲解请参考开发板对应 02 教程文档中《AXI4 转 FIFO 接口模块设计》章节，或参考以下帖子：

【ZYNQ 逻辑】AXI 接口转换模块设计

https://www.corecourse.cn/forum.php?mod=viewthread&tid=29356

1.4.5 eth_send_data 模块

以太网数据发送模块（eth_send_data）的作用是将 DDR 读出的 ADC 采集的数据传送至电脑端，然后我们才可以在电脑端对采集的数据进行分析，该模块的基本结构框图如下图 1-45 所示：

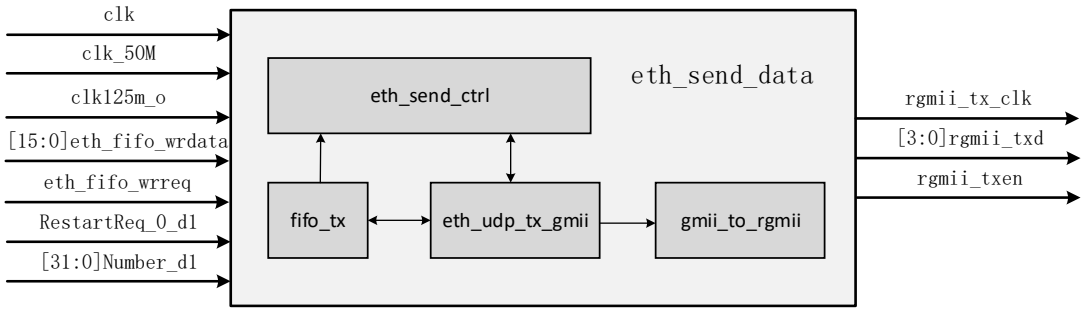

图 1-45 eth_send_data 模块基本结构框图

该模块的信号说明如下表 1-10 所示。

表 1-10 eth_send_data 模块信号说明表

信号名称	I/O	信号意义

reset	I	模块复位信号，高电平有效
clk_50M	I	50M 时钟信号
clk125m_o	I	125M 时钟信号
eth_fifo_wrdata [15:0]	I	16 位的 ADC 采样数据信号
eth_fifo_wrreq	I	发送 fifo 的写使能信号
RestartReq_0_d1	I	启动采样命令信号
Number_d1 [31:0]	I	采样数量命令信号
rgmii_tx_clk	O	发送参考时钟， 1000Mbps 速率下，时钟频率为 125MHz
rgmii_txd [3:0]	O	4 位的以太网发送数据信号
rgmii_txen	O	以太网的发送使能信号

eth_send_data 模块内部的以太网发送模块 eth_udp_tx_gmii 和接口转换模块 gmii_to_rgmii 本章将不再进行说明，接下来将介绍 eth_send_ctrl 模块和 fifo_tx 模块。

1.4.5.1 fifo_tx 模块

FIFO 双端口 IP 核（fifo_tx），该模块需要接收从 DDR 读出的 16 位数据，数据经缓存后转换成 8 位数据由以太网发送模块发送出去。fifo_axi4_adapter 模块中的 FIFO IP 的工作时钟为 50M，以太网发送模块的工作时钟 125M，两者数据速率不匹配，使用 FIFO IP 核进行数据存储可以解决采集过程中会出现的跨时钟域数据交互问题。

在 VIVADO 软件中点击 IP Catalog，然后在搜索栏中输入 FIFO， 在下面搜索的结果中找到 FIFO Generator 并双击，操作如下图 1-46 所示。

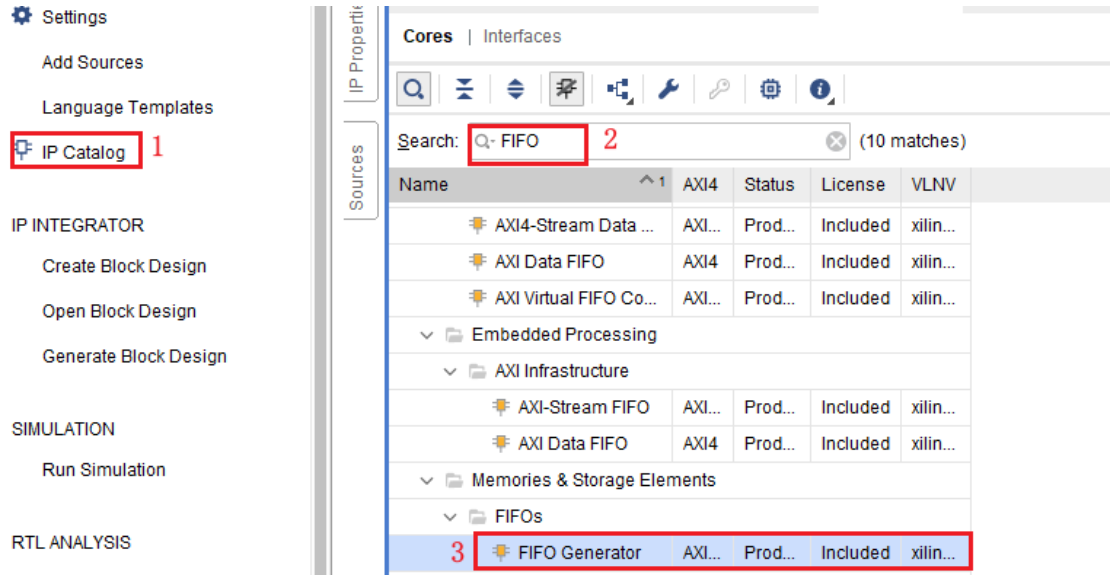

图 1-46 搜索 FIFO IP

双击 FIFO Generator 之后，进入 FIFO 配置界面。

首先将 FIFO 的名称修改为 fifo_tx，接口类型设置为 Native，FIFO 类型上根据使用的资源以及读写时钟是否相同分为多种，这里创建一个独立读写时钟，使用嵌入式 Block RAM 资源的 FIFO，选择 Independent Clocks Block RAM。操作如下图 1-47 所示。创建独立读写时钟是因为 DDR 双端口模块中的 FIFO IP 的工作时钟为 50M，而以太网发送模块工作时钟为 125M，写入和读出的时钟不一致，所以这里需要创建一个独立读写时钟 FIFO。

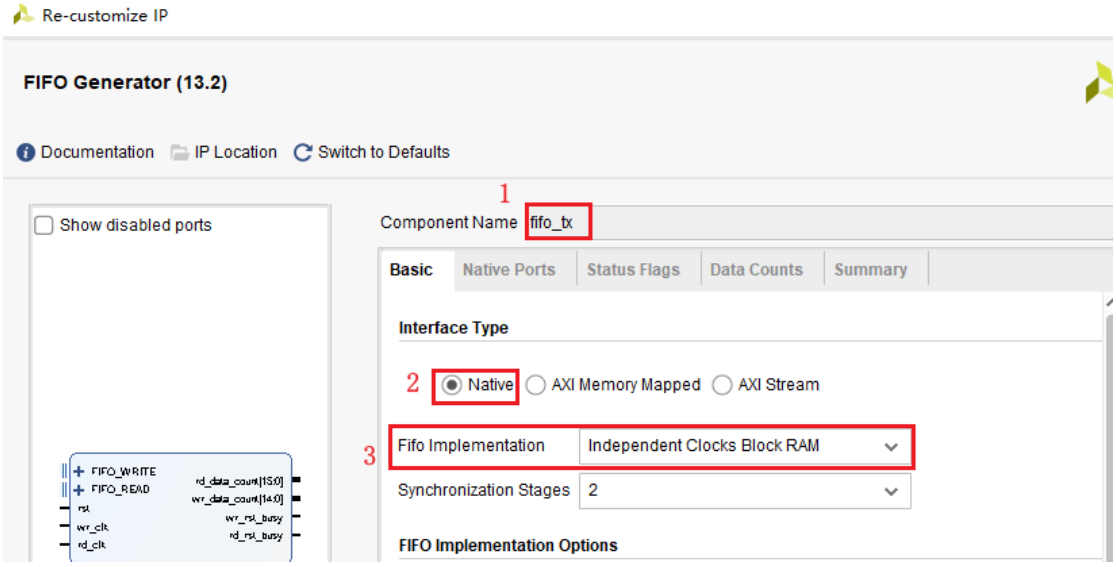

图 1-47 选择独立时钟 FIFO

然后点击“Native Ports”进行配置，配置如下：

1. 读模式选择“First Word Fall Through（FWFT）”，FWFT 模式可以不需要读命令，自动将最新的数据放在 dout 上。
2. 写位宽设置为 16，写入深度设置为 1024。因为 ADC 模块输出的数据是 16 位的，所以位宽设置为 16。写入深度用户可以修改，理论上只要大于以太网最大帧长度 1472 字节（写入深度大于 1472/2）就可以。
3. 读出位宽设置为 8。设置为 8 是因为以太网发送模块的数据位宽是 8 位的。

配置界面如图 1-48 所示。

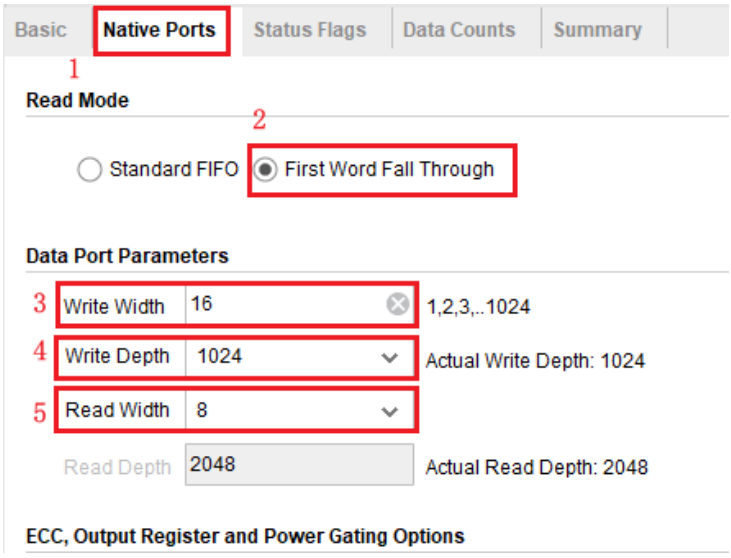

图 1-48 “Native Ports”配置界面

在“Data Counts”界面勾选“Write Data Count”和“Read Data Count”。FIFO 数据量计数信号输出， Write Data Count 和 Read Data Count 分别同步于写时钟和读时钟。位宽可以根据实际进行设置，这里保持默认位宽 10、11。配置界面如下图 1-49 所示。

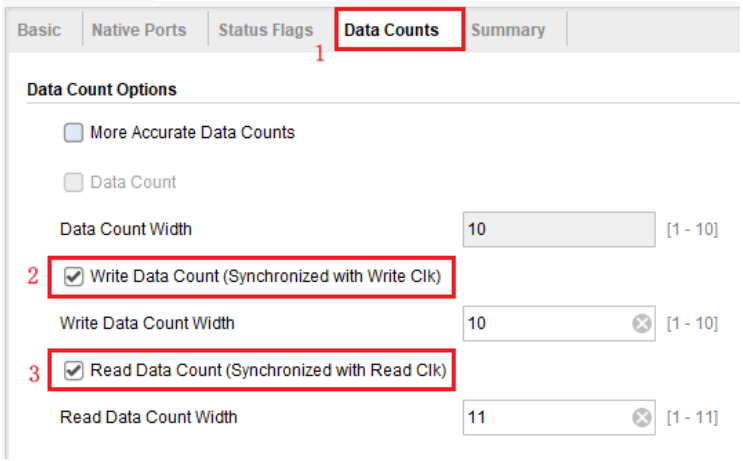

图 1-49 “Data Counts”配置界面

通过上述步骤，完成对 FIFO 的配置。

1.4.5.2 eth_send_ctrl 模块

网口发送控制模块（eth_send_ctrl）主要负责配置控制网口发送模块的使能控制信号 pkt_tx_en，并通过 pkt_length 信号对以太网数据帧数据长度进行控制，该模块的结构框图如下图 1-50 所示。

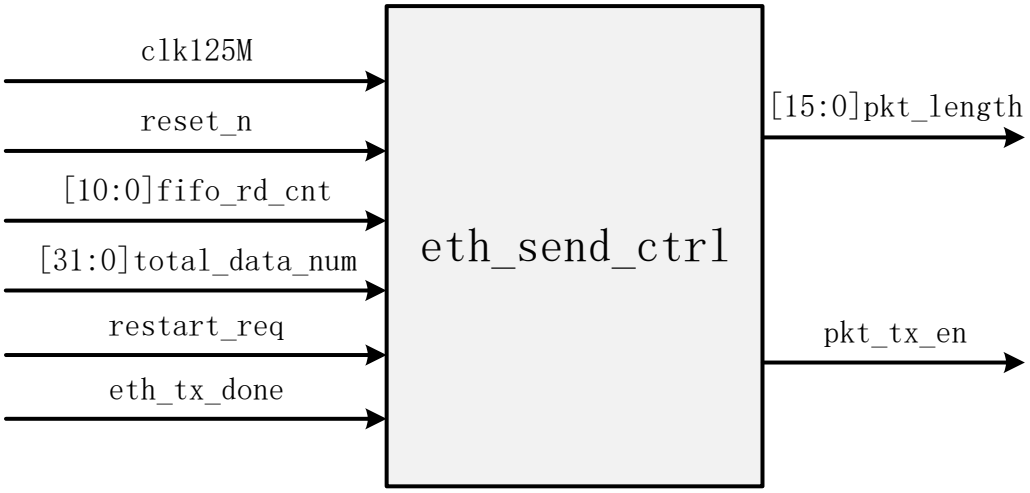

图 1-50 网口发送控制模块

对该模块的信号说明如下表 1-11 所示：

表 1-11 网口发送控制模块信号说明表

信号名称	I/O	信号意义
clk125M	I	模块时钟信号信号，以太网工作时钟 125M
reset_n	I	模块复位信号，低电平复位
fifo_rd_cnt [10:0]	I	FIFO 读数据计数
total_data_num [31:0]	I	需要采集的数据个数
restart_req	I	开始采样请求信号
eth_tx_done	I	以太网一个包传输完成标志信号
pkt_length [15:0]	O	以太网需要传输的数据长度
pkt_tx_en	O	以太网传输使能信号

以太网帧最大长度 1518 字节（数据段 1500 字节），其中数据段 1500 字节还包括 20 字节 IP 报文头部和 8 字节 UDP 报文头部，所以数据帧发送的 ACM9238 采集的数据最大长度为 1472 字节。

该模块可以通过编写状态机代码的方式实现功能，下面将对每个状态的代码进行介绍。

状态 0，得到 pkt_length 信号的初始值。这里需要注意的是经过数据位扩展模块输出的数据为 16 位的，每个数据占据 2 个字节，所以发送 N 个采样数据，则以太网需要发送 2*N 个字节数据。当产生开始采样请求 restart_req 之后，系统开始采样，同时，将需要采集的数据个数 total_data_num 左移一位（相当于乘以 2），得到实际需要以太网传输的数据，当数据大于 16'd1472 时，设置 pkt_length 为最大传输长度 1472；当数据大于 0 时，pkt_length 等于为 total_data_num 乘以 2。给 pkt_length 赋初值之后，跳转到状态 1，代码如下所示：

```
if(restart_req)begin
```

```
  data_num <= total_data_num;
  if((total_data_num << 1) >= 16'd1472)begin
    pkt_length <= 16'd1472; //一个数据 2 个字节
    state <= 1;
  end
  else if((total_data_num << 1) > 0)begin
    pkt_length <= total_data_num << 1; //一个数据 2 个字节
    state <= 1;
    end
else begin
  state <= 0;
  end
end
```

状态 1，当 FIFO 计数信号 fifo_rd_cnt 的数值满足一帧数据帧发送采集数据长度时，产生 pkt_tx_en 信号，以太网发送模块开始读取 FIFO 中的数据。代码如下所示：

```
if(fifo_rd_cnt >= (pkt_length -2))begin
  pkt_tx_en <= 1'b1;
  state <= 2;
end
else begin
  state <= 1;
  pkt_tx_en <= 1'b0;
end
```

状态 2，当以太网一个包传输完成之后，产生 eth_tx_done 信号，此时剩下需要传输的数据 data_num 应该减去 pkt_length 的一半，这是因为 ADC 采集的数据是 16 位的，但是以太网每次只传送 8 位数据，所以以太网实际传输的数据应该是 ADC 采集到的数据的一半。代码如下所示：

```
begin
  pkt_tx_en <= 1'b0;
  if(eth_tx_done)begin
    data_num <= data_num - pkt_length/2;
    state <= 3;
  end
end
```

状态3，设置以太网的帧间隙时间间隔。本次实验使用的千兆以太网，其相邻的两帧之间的最小间隔时间为 96ns，在设置的时候只要大于 96ns 就行，本次实验设置 128 个时钟周期，也就是 1024ns。当设置的时间比较小的时候，虽然以太网传输速率会加快，但是会增加电脑端解析数据包的压力，当时间过大，又会影响以太网传输速率，所以在设置的时候需要设定一个比较合理的值。

```
if(cnt_dly_time >= cnt_dly_min)begin
```

```
    state <= 4;
    cnt_dly_time <= 28'd0;
end
else begin
    cnt_dly_time <= cnt_dly_time + 1'b1;
    state <= 3;
end
```

状态 4，进行连续的包传输。当剩下的需要以太网传输的数据（data_num * 2）大于包传输最大数据 1472 时，使 pkt_length 为 1472，然后回到状态 1 继续传输；当剩下需要传输的数据不足包传输最大数据 1472 时，pkt_length 为剩下的需要传输的数据 data_num * 2，然后回到状态 1 传输剩下的数据。代码如下所示。

```
begin
    if(data_num * 2 >= 16'd1472)begin
        pkt_length <= 16'd1472;
        state <= 1;
    end
    else if(data_num * 2 > 0)begin
        pkt_length <= data_num * 2;
        state <= 1;
    end
    else begin
        state <= 0;
    end
end
```

模块设计完成之后，只需要在顶层文件中对各个模块之间的接口信号进行连接，完整的顶层文件代码请自行查看例程文件。

1.5 板级验证

本次实验的板级验证环节，主要验证：通过电脑上的网络调试助手，将命令帧进行发送， 然后通过 BX71 开发板上的以太网芯片接收，随后将接收到的数据转换命令，最终实现对 ACM9238 模块的采样频率、数据采样个数以及采样通道的配置。配置完成之后，ACM9238 模块开始采集数据，将 ACM9238 模块采集的数据通过网口传输到电脑。电脑端将接收到的数据进行保存，然后通过 MATLAB 进行进一步的分析。针对本次实验，我们也提供有专门的上位机软件，用户只需要在软件界面进行参数配置，便可以实时观察到实时的数据波形变化，使用起来非常方便。

1.5.1 系统所需硬件

本次设计所需硬件如下，相关模块资料可以点击超链接查看：

1. BX71 开发板一块
2. ACM9238 模块一个
3. 千兆网线一根
4. Type-C 下载线一根
5. DC 电源线一根
6. 信号发生器一台

1.5.2 硬件连接

本次设计系统硬件连接如下图 1-51 所示：

1. 使用 Type-C 线连接开发板调试接口和电脑 USB 口
2. 将网线连接至 PL 侧的网口上
3. ACM9238 模块连接至 40 pin 的排针上，靠网口连接，1 脚和 1 脚对应

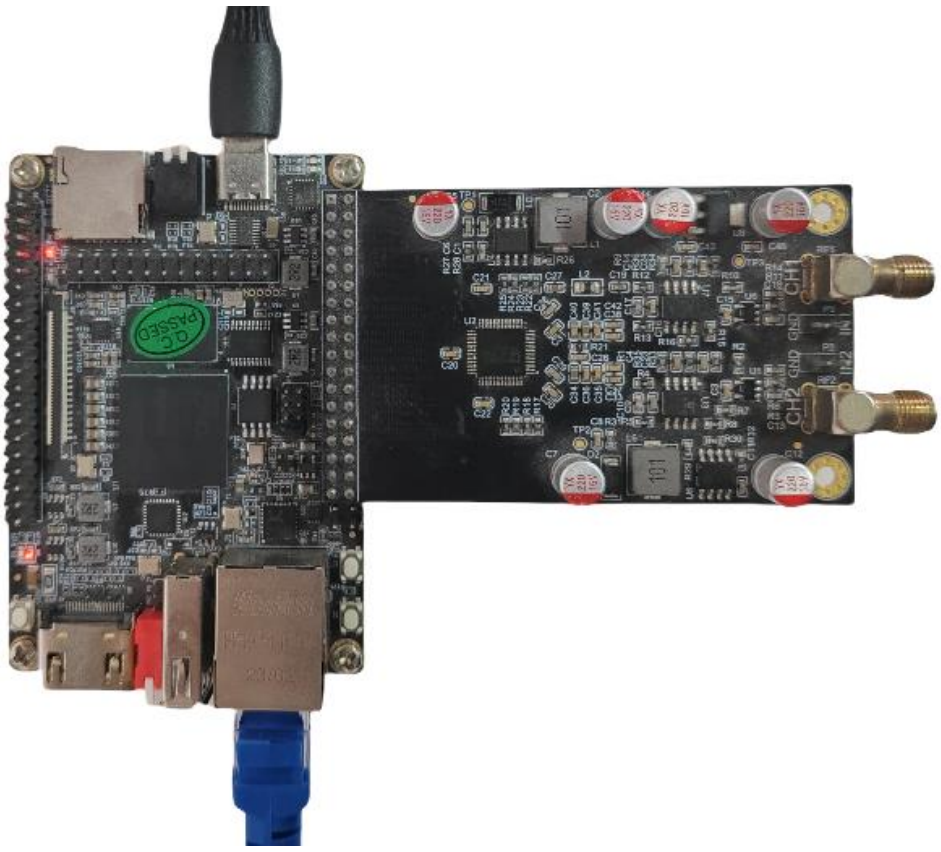

图 1-51 硬件连接图

1.5.3 烧录程序

本次设计需要使用到 PS 端资源，因此我们需要将 bit 文件和硬件规范平台一起导出到 SDK 中，随后运行 SDK。在 SDK 软件中，依次点击 Run->Run Configurations，如下图 1-52 所示。

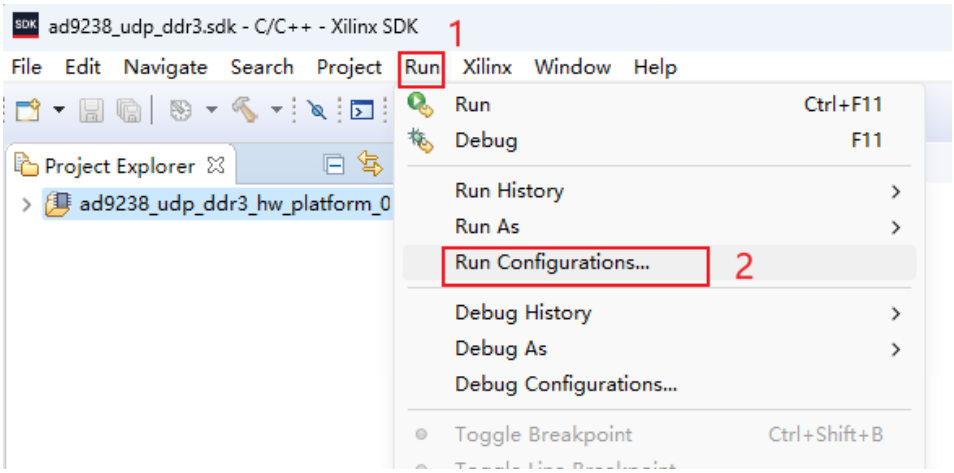

图 1-52 进入下载界面示意图

进入下载界面之后，下载 bit 文件，如下图 1-53 所示。

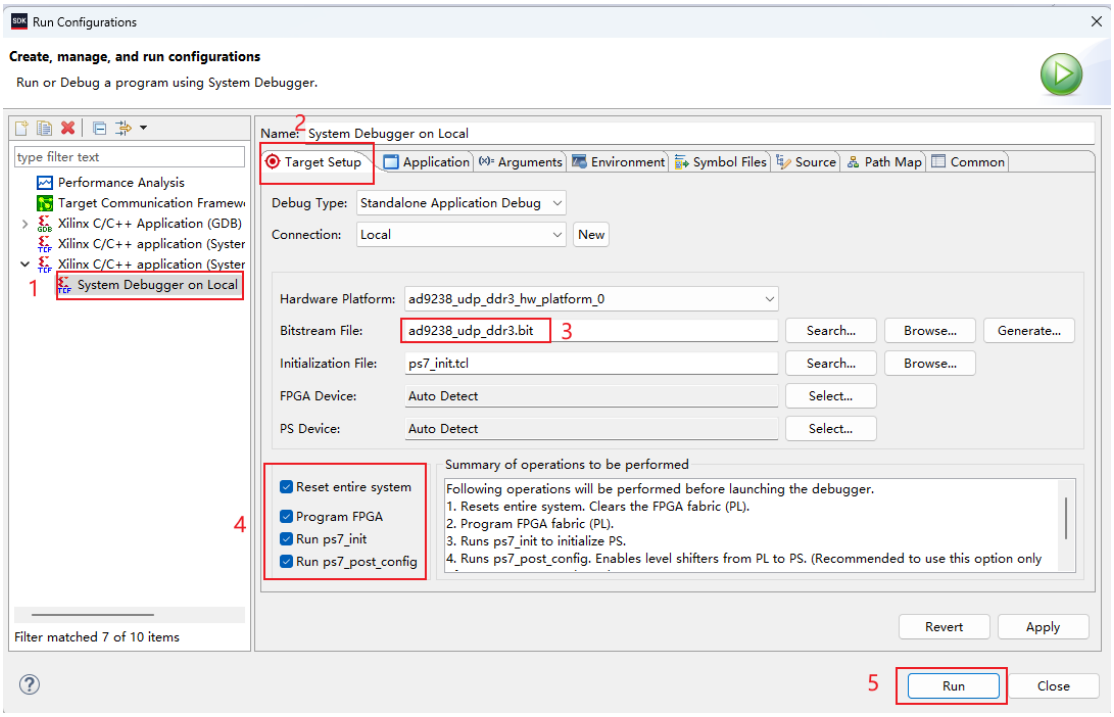

图 1-53 下载 bit 文件

下载成功之后，开发板的 PL 侧的 LED 灯会被点亮，这时说明 PLL 锁相环

工作正常，如下图 1-54 所示。

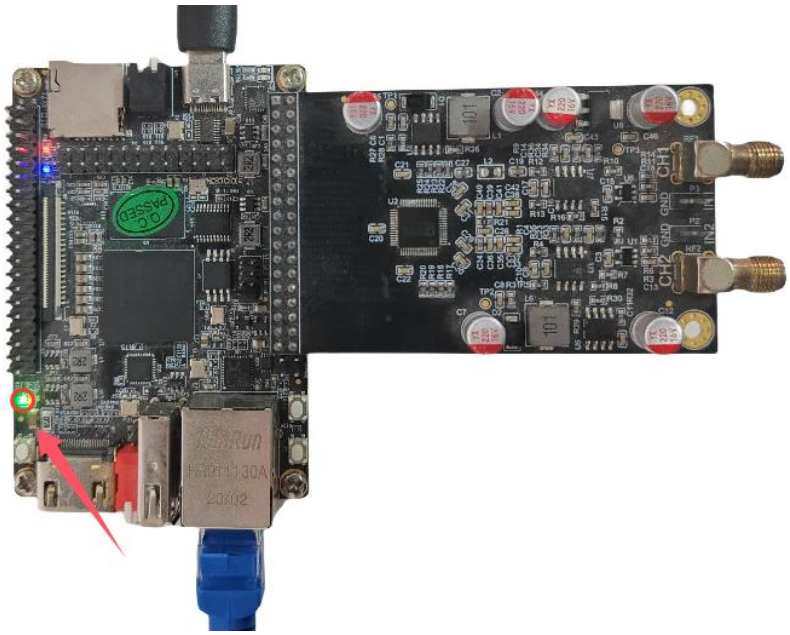

图 1-54 PL 侧 LED 灯被点亮

1.5.4 修改电脑 IP 地址

本次实验设定了目标 IP 地址（PC 端）为 192.168.0.3，用户需要将自己电脑上的以太网 IP 地址修改为该地址，在本地连接状态中，点击属性，并在弹出的属性对话框中双击【Internet 协议版本 4 （TCP/Ipv4）】选项，然后在弹出的属性对话框中设置静态 IP 地址。如下图 1-55 所示。

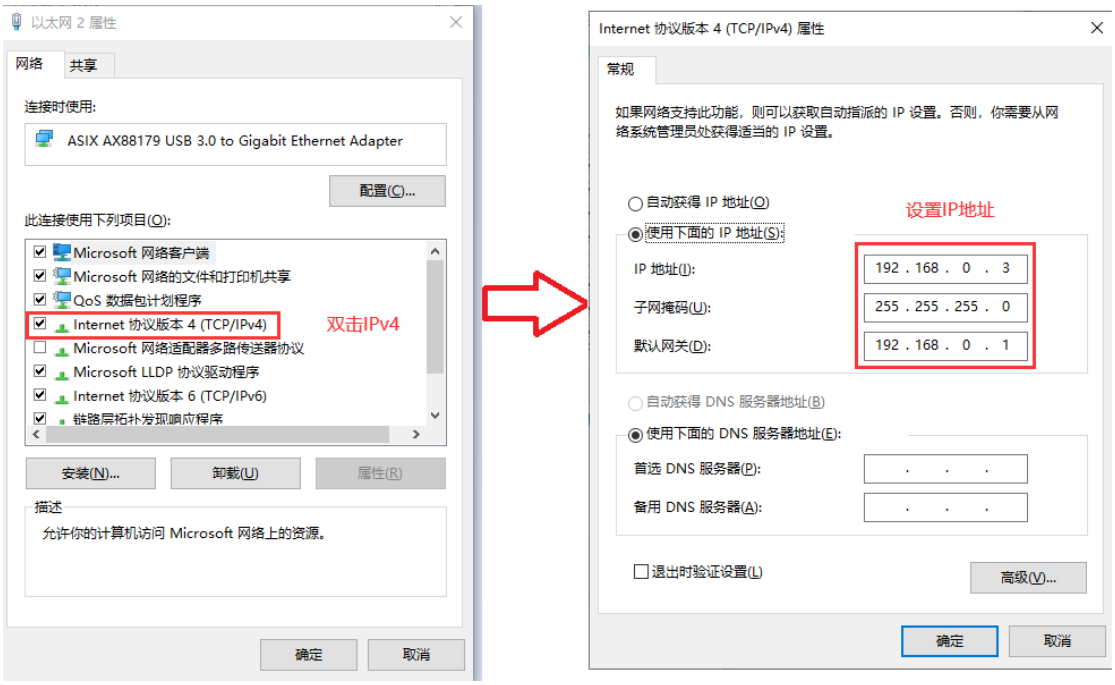

图 1-55 修改电脑 IP 地址

1.5.5 绑定 ARP

本工程不支持 ARP 协议，只能通过静态绑定的方式来强制将开发板的 IP 地址和 MAC 地址关联在一起。这样，当 PC 发送给 192.168.0.2 的数据包的时候，目标 MAC 地址自动为开发板的 MAC 地址。

关于 ARP 的绑定请查看以下帖子内容：

以太网通信静态 ARP 绑定方法与常见问题解决方案

http://www.corecourse.cn/forum.php?mod=viewthread&tid=28645

1.5.6 功能验证

我们首先需要通过信号发生器给 ACM9238 的通道一输入 100Khz，vpp=5V 的正弦波，然后通过网络调试助手或者我们提供的上位机软件“小梅哥控制台 For ADC 采集”采集数据。

1.5.6.1 网络调试助手通信

首先需要打开网络调试助手发送指令去配置，按照如下所述设置各项参数。网络调试助手软件读者可以在论坛中找到。

【小软件合集】FPGA 课程教程中常用软件和小工具合集

https://www.corecourse.cn/forum.php?mod=viewthread&tid=28374

(出处: 芯路恒电子技术论坛)

1. 选择协议类型为 UDP。
2. 设置本地 IP 地址为 192.168.0.3。
3. 设置本地端口号为 6102。
4. 点击【连接】按钮以创建连接，连接上后该按钮为红色“断开”字样。
5. 连接上后，设置目标主机为 192.168.0.2，目标端口为 5000。
6. 点击“接收保存到文件”这几个字，在弹出的界面中设置文件路径、文件名称，如下图 1-56 示。这样在数据接收完成之后会保存一个数据文件。方便后面进行分析。

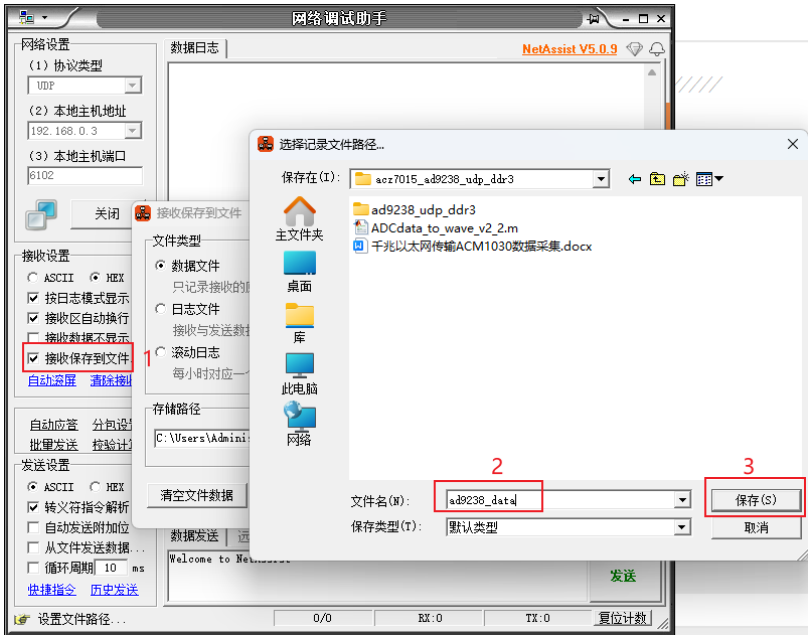

图 1-56 设置保存文件

1.发送设置中数据类型设置为 hex 格式，如下图 1-57 所示。

图 1-57 设置发送格式为 hex 码格式

2.发送命令帧。

在前面接收转命令模块中介绍到数据帧格式对 ACM9238 的四个寄存器的配置。

例如，PC 端要设置采样数据个数(DataNum 寄存器，地址为 2)为 16384(0x4000)个，发送数据帧内容：0x55 0xA5 0x02 0x00 0x00 0x40 0x00 0xF0。PC 端要设置采样速率(ADC_Speed_Set 寄存器，地址为 3)为 50M，则需要发送的数据帧内容为：0x55 0xA5 0x03 0x00 0x00 0x00 0x00 0xF0。

当上述设置都设置完成后，就可以向 0 号寄存器写入任意值，来开始一次采样传输了。数据帧内容可以为 0x55 0xA5 0x00 0x00 0x00 0x00 0x00 0xF0，这里需要注意的是 0 号寄存器必须放在最后，因为 0 号寄存器负责启动 ADC，ADC 在未配置完全的情况下开始启动，数据很容易输出错误值。

开始传输数据帧命令发送完成之后，ACM9238 就能实现以 50M 的采样速率，对 1 个通道进行采样（本次实验以通道 1 为例），共采集 16384 个数据。四个寄存器对应的配置如下表 1-12 所示：

表 1-12 ACM9238 数据帧格式配置表

寄存器名称	数据帧数据
DataNum	55 A5 02 00 00 40 00 F0
ChannelSel	55 A5 01 00 00 00 01 F0
ADC_Speed_Set	55 A5 03 00 00 00 00 F0
RestartReq	55 A5 00 00 00 00 00 F0

配置成网络调试助手发送的数据格式如下：

55A50200004000F055A50100000001F055A50300000000F055A50000000000F0

最终的网络助手配置界面如下图 1-58 所示：

图 1-58 网络调试助手配置界面

点击发送之后可以看到网络调试助手在不断的接收数据，如下图 1-59 所示。从图中可以看出一共接收到 32768 个数据，这是因为设置的 ADC 采样数量为

16384，ADC 采样数据是 16 位的，以太网是以字节（8 位）为单位进行发送的，所以通过以太网接收到字节数应该是 16384*2 个数据，这也就是说明接收到的数据的个数没错。

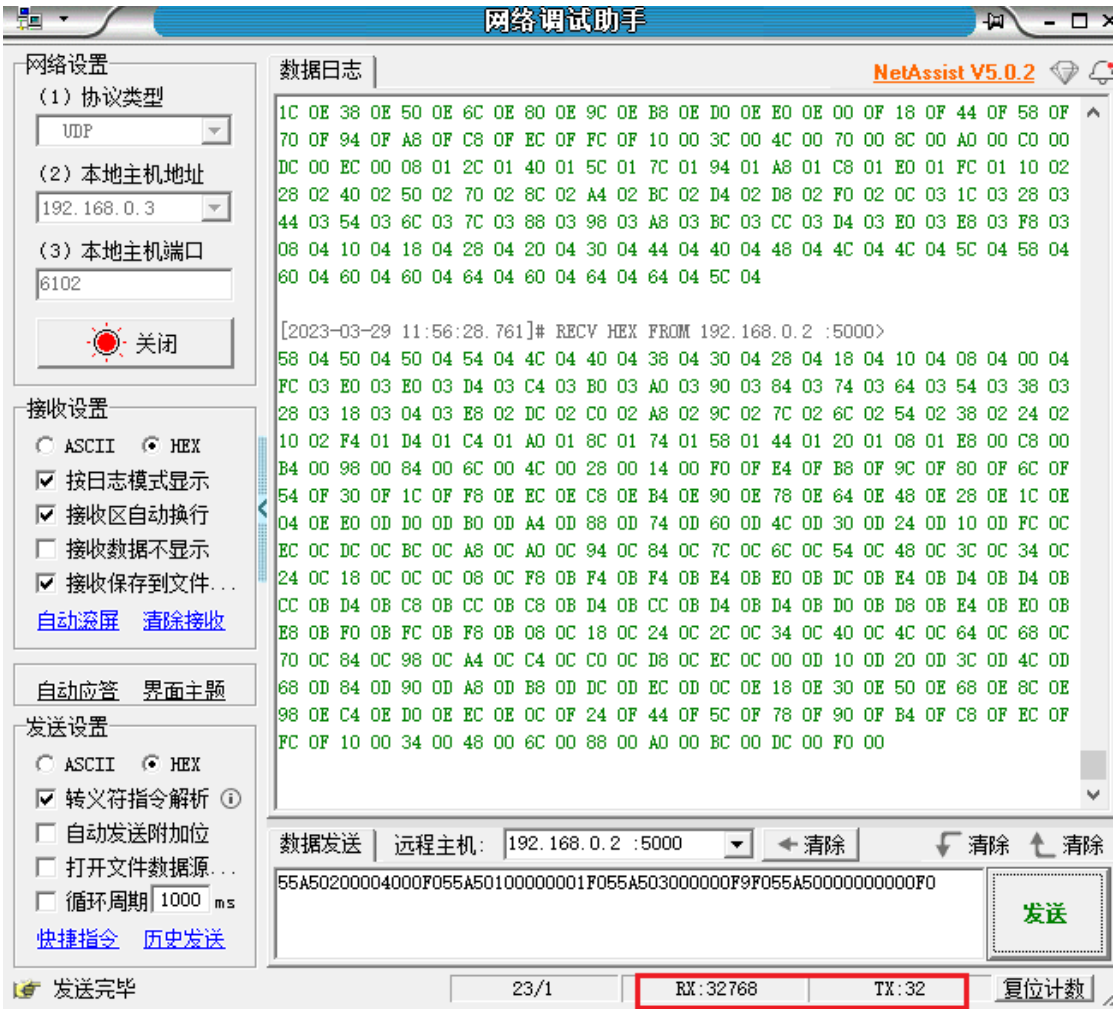

图 1-59 网络调试助手接收数据

1.5.6.2 MATLAB 图像绘制

前面通过网络调试助手得到了 ADC 采集到的数据文件，然后我们就需要对采集到的数据进行分析，本次实验使用 MATLAB 软件进行分析。使用 MATLAB 软件需要读者电脑安装了 MTALAB，如果已经安装好了 MTALAB 软件，则可以双击我们提供的 ADCdata_to_wave_v2_2.m 文件，在打开方式里选择以 MATLAB 打开，该文件位于本次实验的工程压缩包下，将压缩包解压便可以看到该文件。文件打开之后，读者需要将代码中文件路径修改为你保存的数据文件路径，随后点击运行便可以直观的看到数据是否正确，MATLAB 操作如下图 1-60 所示，得到的波形图如下图 1-61 所示。

```
%设置ADC采集数据有效位宽
ADC_DATA_WIDTH = 12;
%ADC采集电压范围±VOL_RANGE (V)
VOL_RANGE = 5;
%采样率(Hz)
SAMPLE_RATE = 50000000;
%ADC采集电压分辨率 LSB
LSB = VOL_RANGE*2/(2^ADC_DATA_WIDTH);
%=========================================================================%
1
%读取文件数据，转换成波形显示
fileID = fopen('C:\Users\Administrator\Desktop\acz7015_ad9238_udp_ddr3\ad9238_data.log');%在这里修改需要分析的数据文件路径
src_data = fread(fileID);
fclose(fileID);
src_data_hex = src_data;

DATA_NUM = length(src_data_hex);
if(DATA_NUM > 1024*64)
  DATA_NUM = 1024*64;
end

voltage code = 1:DATA NUM/2;
```

图 1-60 修改文件路径并运行

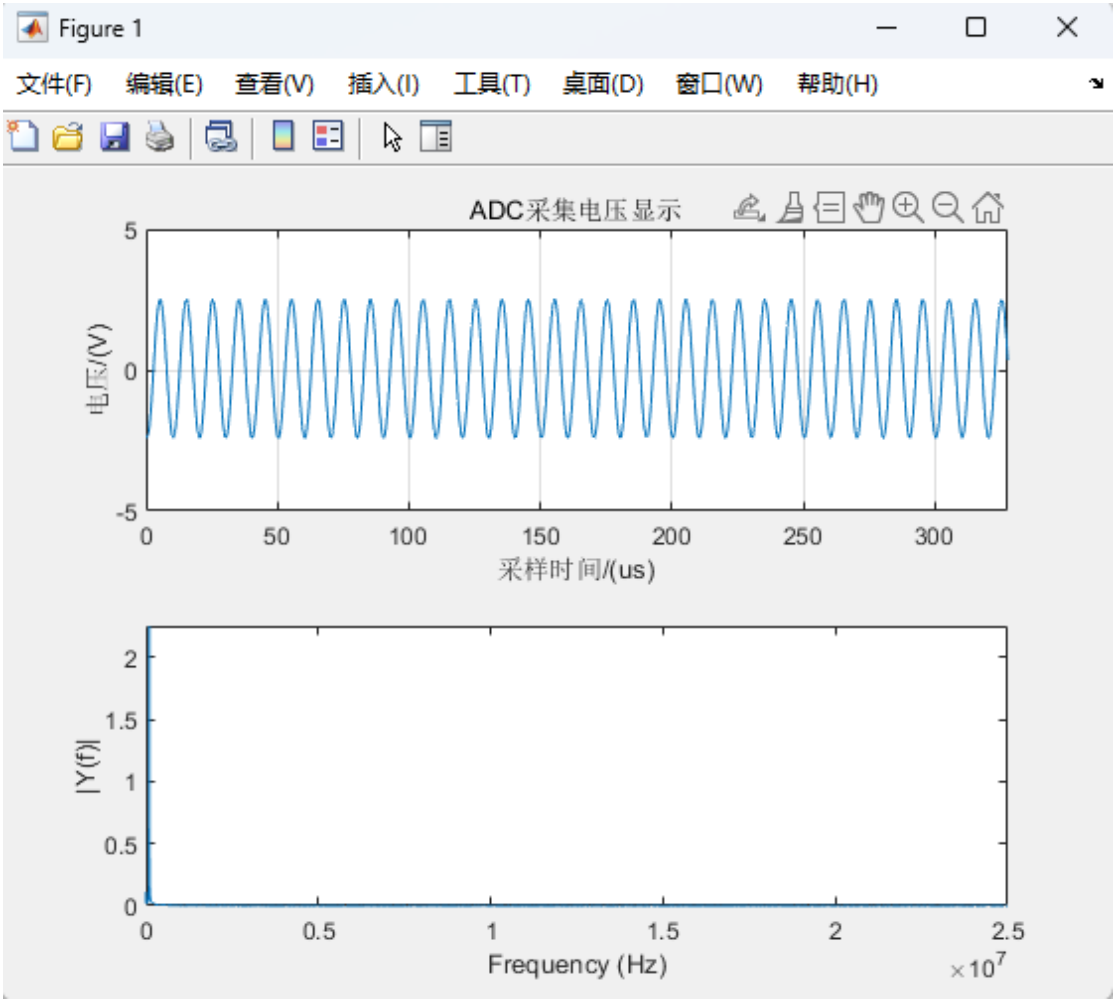

图 1-61 MATLAB 分析波形图

前面我们提到过本次实验提供的信号源为 100Khz，Vpp 为 5V 的正弦波，与 MATLAB 分析出来的波形一致，说明我们本次实验成功。

1.5.6.3 数据采集上位机通信

前面通过网络调试助手采集数据时，每次保存数据都需要重新点击“接收保存文件”一栏，修改寄存器参数的时候，都需要重新计算，然后发送命令，修改之后也不能直接实时观察到数据波形，使用起来不是很方便。基于上述问题，我们设计了上位机软件“小梅哥控制台 For ADC 采集”进行数据采集，上位机内部直接对命令进行了构建，用户只需要在界面上对采样参数进行设置，便可以实时观测到数据变化，该软件的最新下载链接如下所示：

数据采集上位机使用方法说明

http://www.corecourse.cn/forum.php?mod=viewthread&tid=29224

双击上位机软件，初始界面如下图 1-62 所示。

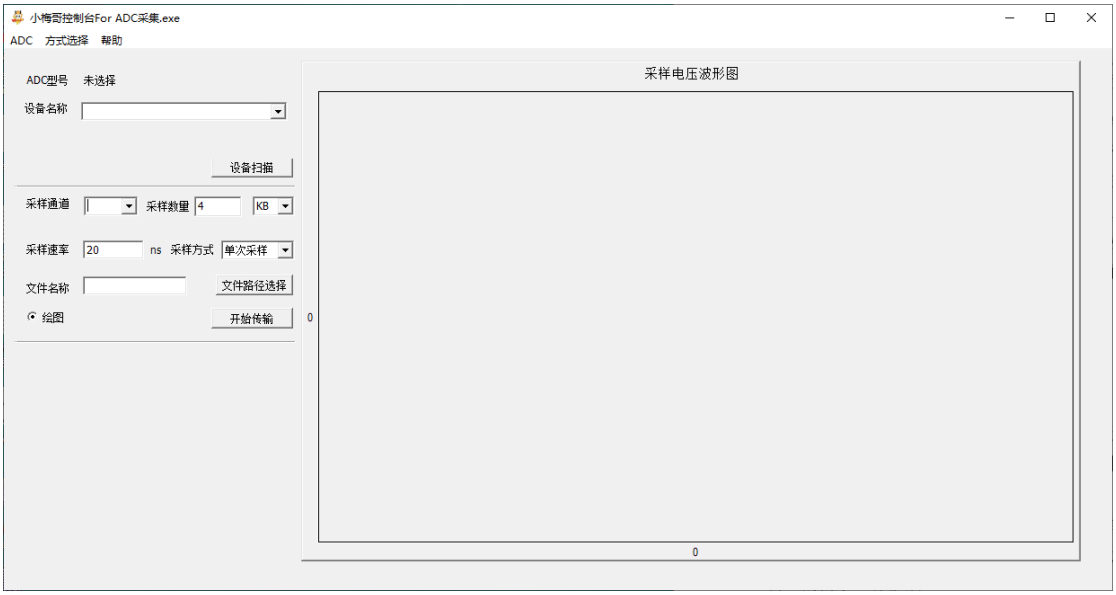

图 1-62 上位机软件初始界面显示

本次实验使用该软件的方式如下所示：

1. 点击 ADC，选择 ACM9226。
2. 点击方式，选择网口，可以看到主机 IP（PC 端）和目的 IP（FPGA）以及对应的端口号。主机 IP：192.168.0.2，主机端口号：6102；目的 IP：192.168.0.3，目的端口号：5000。
3. 选择完成之后，我们可以看到采样通道、采样数量等都已经设置了初始值（默认设置的采样率为 ADC 模块的最大采样率），用户可以根据自己的需求进行修改。
4. 点击网络连接。

5. 点击开始传输之后，可以看到在右边采样电压波形图界面可以直观看到波形图，如下图 1-63 所示。需要注意的是波形图的横坐标对应的不是频率，而是采样数量。

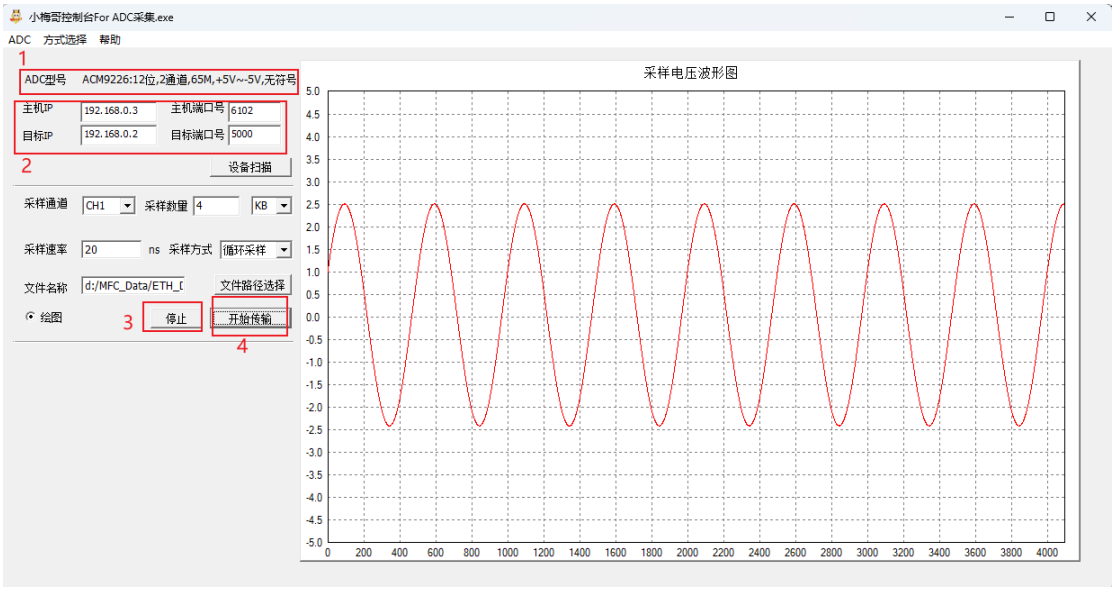

图 1-63 数据采集上位机显示图

通过上位机采集到的数据文件保存在 d:/ MFC_Data 文件夹下，后续可以通过 MATLAB 软件进行进一步的分析，通过 MATLAB 分析的波形图如下图 1-64 所示。从图中可以看出，采集到的数据是频率为 100Khz，电压在正负 2.5V 左右的正弦波，与我们输入的信号一致。

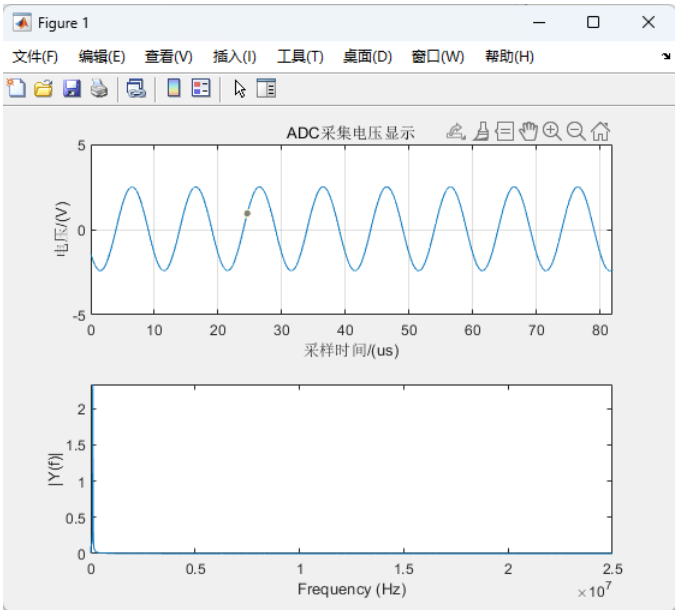

图 1-64 MATLAB 进一步波形分析图

1.6 思考与总结

本次实验介绍了基于 ACM9238 的千兆以太网收发，用户通过网口调试助手以太网帧向开发板发送指令数据配置 ACM9238 的四个寄存器，以此控制 ADC 进行采样，并将数据缓存后再组成以太网帧，经由网口发送至电脑，借由网口调试工具对数据进行查看。如果使用我们提供的上位机软件，则不需要自己设置命令，只需要在界面上修改相关参数，便可以在右边的波形显示界面实时观察到波形变化。

本次实验涉及时钟域的转换，以及采集数据位宽的转换，完成这些转换需要对 FIFO 有一定的了解，想要对 FIFO 进一步了解可以参看逻辑教程中“IP 核使用之 FIFO”一节的内容。本次实验也要求读者对以太网协议有足够的认识，想要进一步理解以太网功能原理可参看逻辑教程中以太网相关章节的内容。