# 1 ACM2108/108 数据采集 DDR3 缓存网口发送实验

该文档主要是以 ACM2108 为例进行说明的，用户如果使用的 ACM108 模块，实验操作步骤同样适用，但是需要注意的是 ACM108 模块只有一个通道 0 可以使用。两个模块对应的工程文件为 “acz702_acm2108_udp_ddr3.rar”和“acz702_acm108_ udp_ddr3.rar”。

## 1.1 系统整体设计

本次实验，我们将通过电脑上的网络调试助手，将命令帧进行发送，然后通过 ACZ702 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 ACM2108/108 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，ACM2108/108 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。

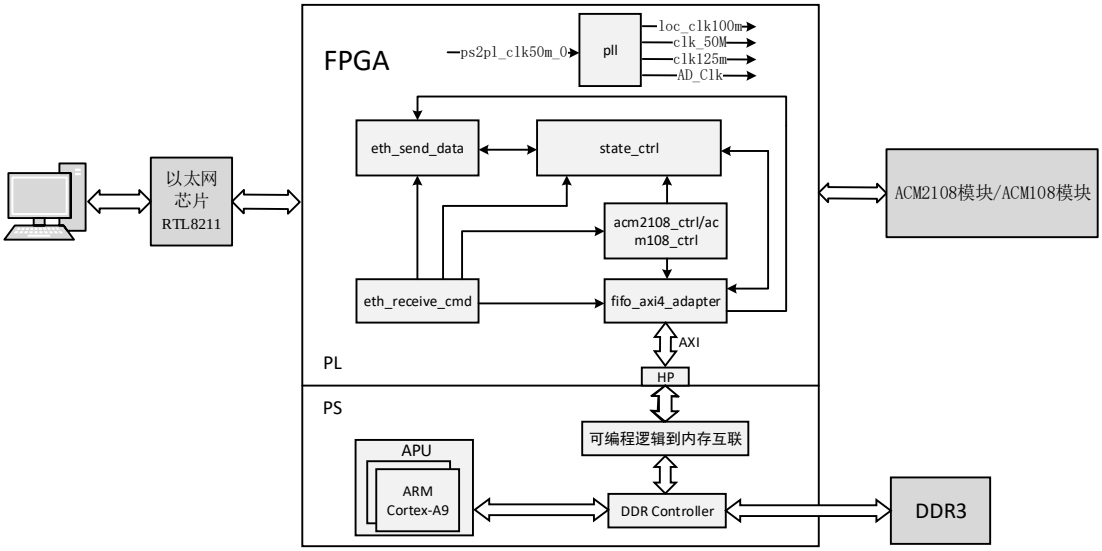


图 1-1 ACM2108/108 数据采集 DDR3 缓存网口发送整体设计框图

本章实验仅针对于上述图中的 PL 部分的代码设计进行讲解，PS 部分的内容请参看帖子：

【ACZ702】ZYNQ PL 读写 PS DDR3 双端口读写控制器设计

http://www.corecourse.cn/forum.php?mod=viewthread&tid=29312

PL 侧模块说明如下：

1. pll 模块：锁相环模块，输入时钟 50M，由 PS 输出给 PL；输出 100M 的

时钟给到 DDR3 控制器使用；输出 50M 的时钟给其它模块使用；输出 125M 的时钟给 DDS 模块使用；输出 50M 时钟，时钟相位偏移负 90 度的 AD_Clk 给 ACM2108/ACM108 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. acm2108_ctrl/ acm108_ctrl 模块：ACM2108/108 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)。

# 1.2ACM2108 高速 ADC+DAC 模块简介

本模块基于国产知名模拟器件设计和制造商杭州瑞盟公司的 8 位 50M 采样速率高速 ADC 芯片 MS9280（完全兼容 AD9280，但是速度比 AD9280 高）和 125Msps 转换速率的高速 DAC 芯片 MS9708（完全兼容 AD9708）。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，以及模拟信号输出。下图 1-2 为模块实物图。

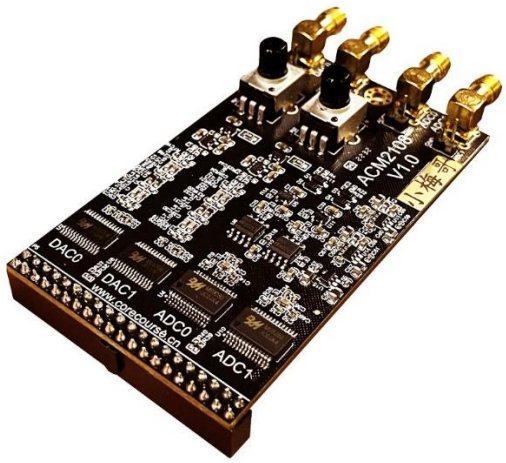


图 1-2 ACM2108 模块实物图

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。AC620、AC6102、ACX720、ACZ702、AC609、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 模块设计

## 1.3.1 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 1-3 所示：

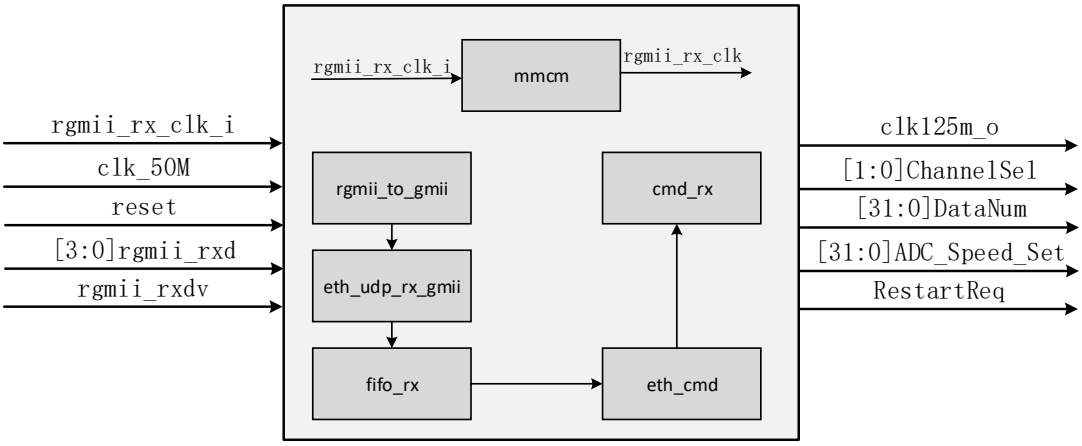


图 1-3 eth_receive_cmd 模块基本结构框图

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 |

以太网相关部分读者自行查看 ACZ702 开发板资料的 02_逻辑文档教程中的相关内容，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。

### 1.3.1.1 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 1-4 所示。

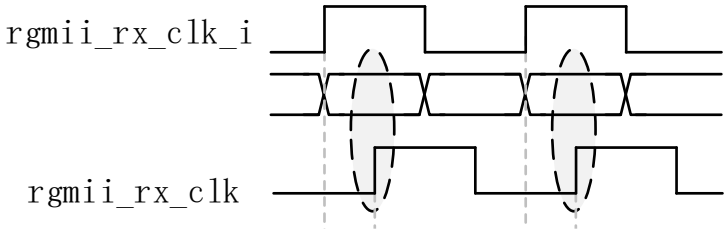


图 1-4 采集数据波形图

锁相环 IP 配置界面如下图 1-5 所示。

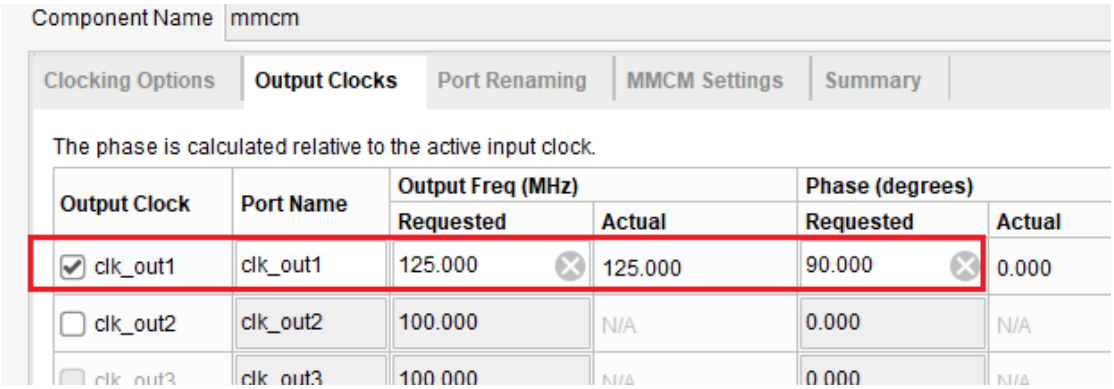


图 1-5 锁相环配置界面

## 1.3.1.2 fifo_rx 模块

FIFO IP 核，以太网接收模块工作时钟 125M，ACM2108/ACM108 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 1-6 所示：

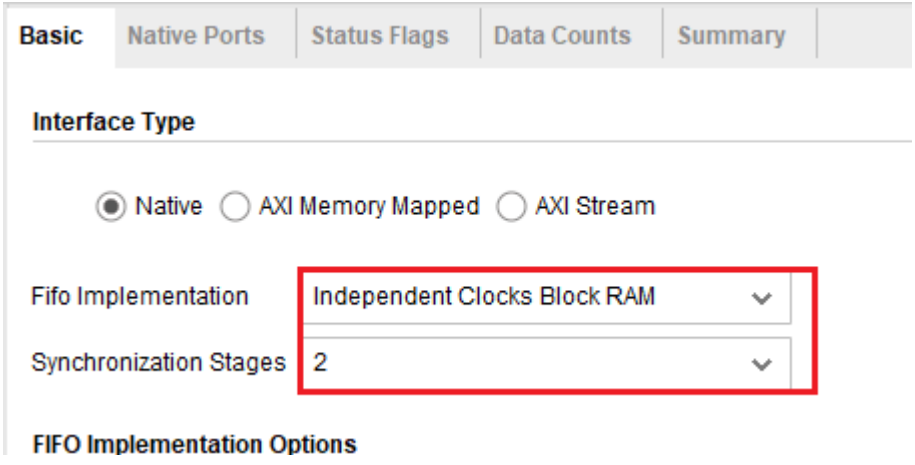


图 1-6 双端口 FIFO

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 1-7 所示：

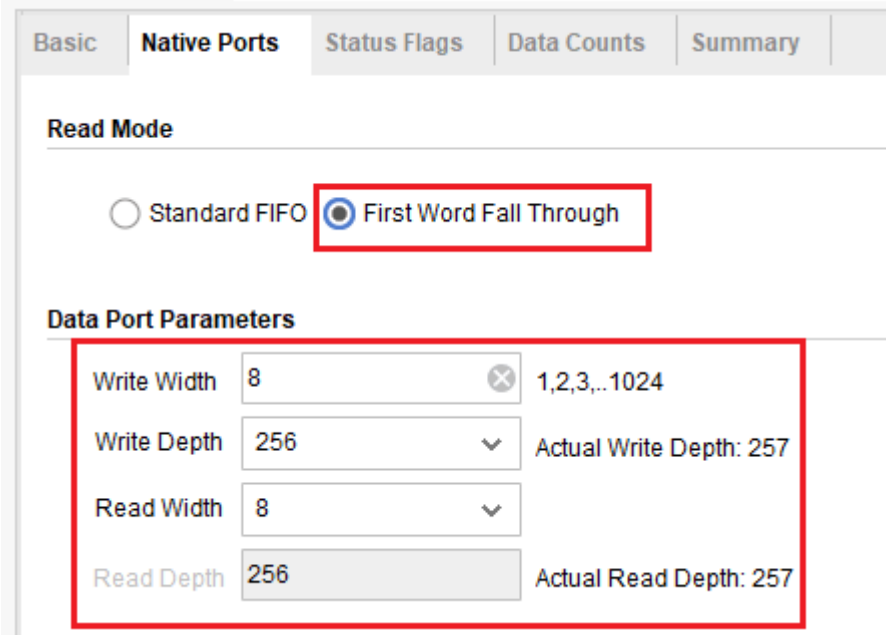


图 1-7 配置 FIFO 的读写位宽及深度

### 1.3.1.3 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 1-8 所示。

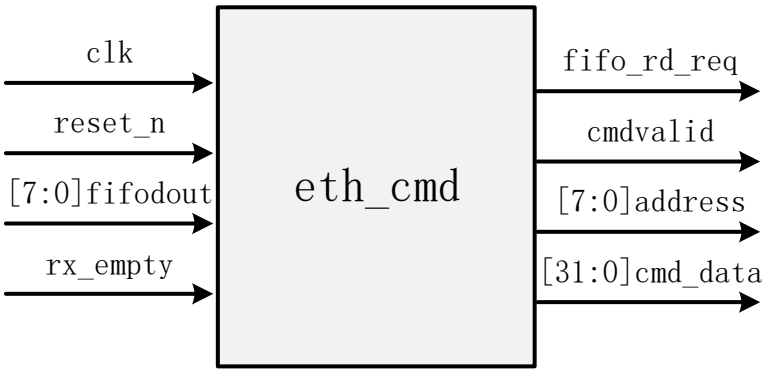


图 1-8 接收转命令模块框图

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 ADC 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 ACM2108/108 模块进行相应的配置。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;
```

### 1.3.1.4 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。ACM2108 模块提供了 ADC0、ADC1 两个通道进行数据采集，ACM108 模块则只有一个 ADC0 通道可以用。 |
| 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 01 |

| | | | |
|---|---|---|---|
| | | | 00。 |
| 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 1-9 所示。

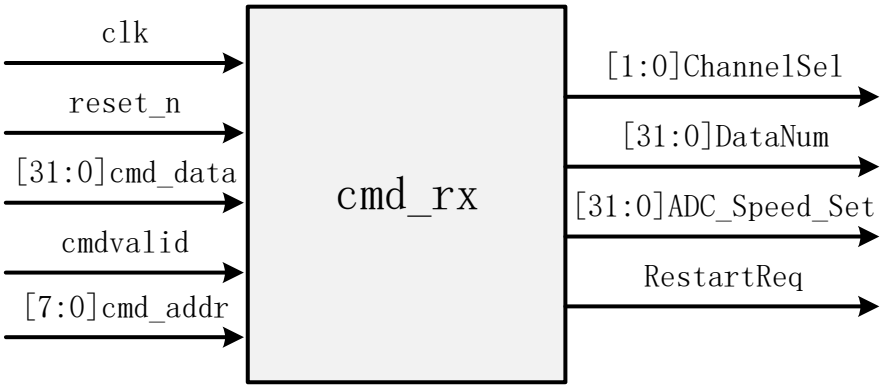


图 1-9 指令转控制模块结构框图

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;
```

## 1.3.2 acm2108_ctrl 模块

ACM2108 控制模块（acm2108_ctrl）的功能是控制 ADC 的采样速率，并且将 ADC 采集到的 8 位数据转换成 16 位的数据，然后传输给电脑进行存储，acm108_ctrl 模块与 acm2108_ctrl 模块相比，就是少了一路 ADC 通道和 DAC 通道，其它部分都是一样的，这里以 acm2108_ctrl 模块为例进行说明，该模块的基本结构框图如下图 1-10 所示：

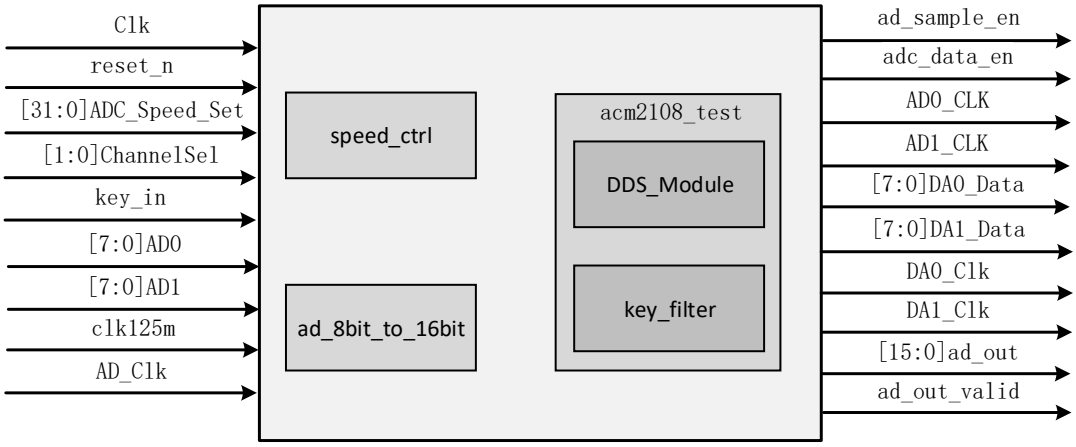


图 1-10 acm2108_ctrl 模块基本结构框图

对上述图中模块的信号说明如下表 1-6 所示：

表 1-6 acm2108_ctrl 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，50M 时钟 |
| reset_n | I | 模块复位信号，低电平有效 |
| ADC_Speed_Set[31:0] | I | 采样速率命令信号 |
| ChannelSel[1:0] | I | 采样通道命令信号 |
| key_in | I | 按键输入信号 |
| AD0[7:0] | I | ADC 模块 ADC0 输入的 8 位数据 |

| | | |
|---|---|---|
| AD1[7:0] | I | ADC 模块 ADC1 输入的 8 位数据 |
| clk125m | I | 输入的 125M 的时钟 |
| AD_Clk | I | 输入的 ADC 时钟 |
| ad_sample_en | O | 输入 ADC 采样使能信号 |
| adc_data_en | O | 输出的 ADC 采样使能标志信号 |
| AD0_CLK | O | ADC 模块的 ADC0 工作时钟信号 |
| AD1_CLK | O | ADC 模块的 ADC1 工作时钟信号 |
| DA0_Data[7:0] | O | ADC 模块的 DAC0 的 8 位数据输出 |
| DA1_Data[7:0] | O | ADC 模块的 DAC1 的 8 位数据输出 |
| DA0_Clk | O | ADC 模块的 DAC0 工作时钟信号 |
| DA1_Clk | O | ADC 模块的 DAC1 工作时钟信号 |
| ad_out[15:0] | O | 对应通道输出的 16 位的数据信号 |
| ad_out_valid | O | 输出数据有效信号 |

acm2108_ctrl 模块内部的 acm2108_test 模块用于 DAC 输出信号波形，并通过按键切换信号类型。

### 1.3.2.1 speed_ctrl 模块

采样速率控制（speed_ctrl）模块用来控制 ACM2108/108 的采样速率，该模块的结构框图如下图 1-11 所示。

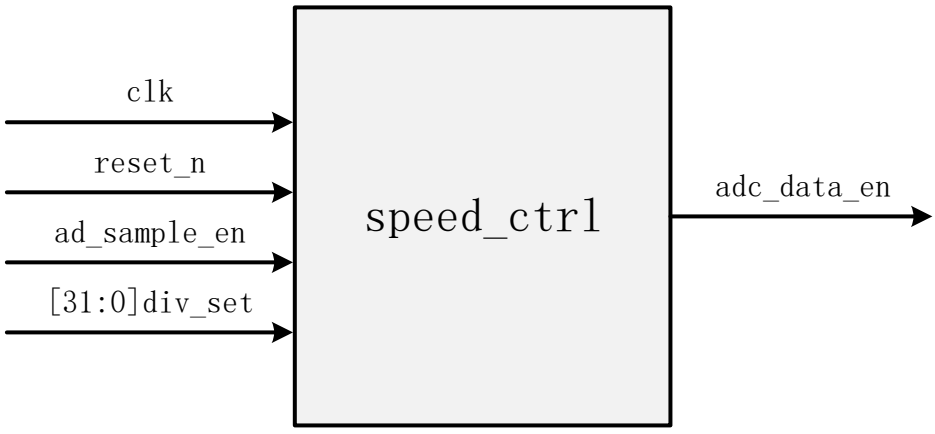


图 1-11 采样速率控制模块

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 |

ACM2108/108 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;
```

### 1.3.2.2 8bit 转 16bit 模块

为了方便后续进行上位机进行数据采集，将 8bit 数据转换成 16bit 数据进行存储。8bit 转 16bit 模块的基本框图如下图 1-12 所示，端口列表如下表 1-8 所示。

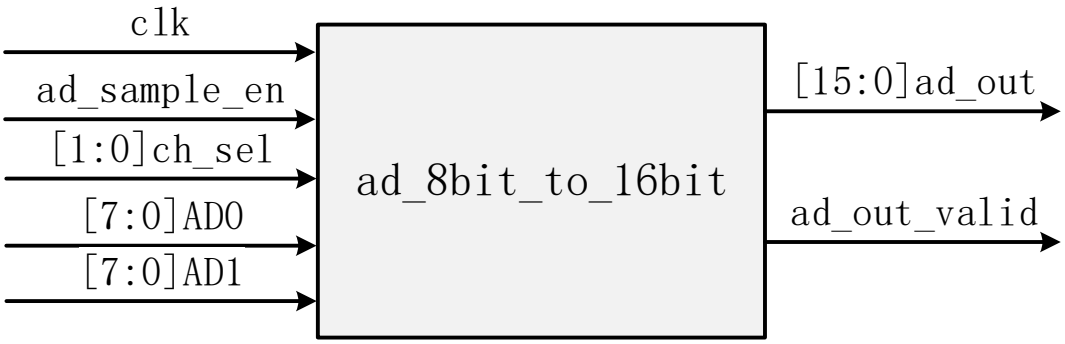


图 1-12 8bit 转 16bit 模块的基本框图

表 1-8 8bit 转 16bit 模块的端口列表

| 信号名称 | I/O | 信号意义 |
|---|---|---|
| clk | I | 模块时钟信号 |
| ad_sample_en | I | 数据采集使能信号 |
| ch_sel[1:0] | I | 采样通道命令信号 |
| AD0[7:0] | I | ADC 通道 0 的 8 位数据输入信号 |
| AD1[7:0] | I | ADC 通道 1 的 8 位数据输入信号 |
| ad_out | O | 16 位数据输出信号 |
| ad_out_valid | O | 输出数据有效信号 |

这里我们给出代码如下：

```
module ad_8bit_to_16bit(
    clk,
    ad_sample_en,
    ch_sel,
    AD0,
    AD1,
    ad_out,
    ad_out_valid
);
  input clk;
  input ad_sample_en;
  input [1:0]ch_sel;
  input [7:0] AD0;
  input [7:0] AD1;
  output[15:0] ad_out;
  output ad_out_valid;
  reg[15:0] ad_out;
  reg ad_out_valid;

    //用于仿真或产生测试数据
  reg [7:0]adc_test_data;
  //测试数据，当 ad_sample_en 为 1 时，锁相环生成的 50M 时钟每个周期使
adc_test_data 加 1
  always@(posedge clk)
```

```
    adc_test_data <= ad_sample_en ? (adc_test_data + 1'b1) : 8'd0;

  wire [7:0]s_ad_in1;
  wire [7:0]s_ad_in2;

  assign s_ad_in1 = AD0 + 8'd128;
  assign s_ad_in2 = AD1 + 8'd128;

  always @(posedge clk)
  if(ad_sample_en && ch_sel == 2'b01)
    ad_out<={4'd0,s_ad_in1,4'd0};//
  else if(ad_sample_en && ch_sel == 2'b10)
    ad_out<={4'd0,s_ad_in2,4'd0};//
  else if(ad_sample_en && ch_sel == 2'b00)
    ad_out<={4'd0,adc_test_data,4'd0};
  else
    ad_out <= 16'd0;

  always @(posedge clk)
    ad_out_valid <= ad_sample_en;

endmodule
```

ACM2108/108 模块采集到的数据是无符号的数据，比如采集的波形为+5V~-5V 的正弦波，ADC 模块最终输出的数据就为 255~0，而我们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 128，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。

## 1.3.3 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 1-13 所示。

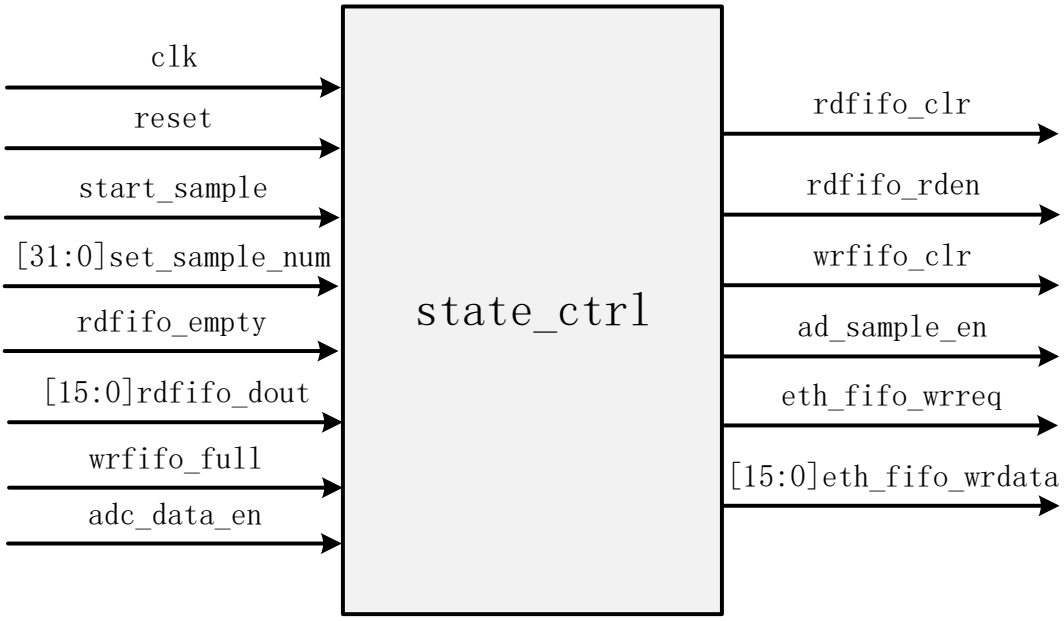


图 1-13 state_ctrl 模块结构框图

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 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，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;
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，当 rdfifo_rden 信号为高电平的时候，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：

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

```
    eth_fifo_wrdata <= 'd0;
end
```

## 1.3.4 fifo_axi4_adapter 模块

fifo 接口到 AXI4 接口的转换模块（fifo_axi4_adapter）的设计，读者自行查看论坛帖子：

【ACZ702】ZYNQ PL 读写 PS DDR3 双端口读写控制器设计

http://www.corecourse.cn/forum.php?mod=viewthread&tid=29312

需要注意的是，这里新增加了一个起始信号，当接收到启动传输之后，启动 fifo2axi4 模块中的状态转移，也也就是启动向 DDR 中写入数据，代码如下所示：

```
S_IDLE:
begin
    if(start)
        next_state = S_ARB;
    else
        next_state = S_IDLE;
end
```

该模块其它部分的设计和前面章节中一致，这里将不再进行说明，只需要将该模块例化进来使用即可，例化代码如下所示：

```
fifo_axi4_adapter #(
    .FIFO_DW                (16                    ),
    .WR_AXI_BYTE_ADDR_BEGIN (DDR_BASE_ADDR + 1'b1 ),
    .WR_AXI_BYTE_ADDR_END   (DDR_BASE_ADDR + 16'd65535),
    .RD_AXI_BYTE_ADDR_BEGIN (DDR_BASE_ADDR + 1'b1 ),
    .RD_AXI_BYTE_ADDR_END   (DDR_BASE_ADDR + 16'd65535),

    .AXI_DATA_WIDTH         (64                    ),
    .AXI_ADDR_WIDTH         (32                    ),
    .AXI_ID                 (4'b0000               ),
    .AXI_BURST_LEN          (8'd15                 )  //axi burst
length = 16
)fifo_axi4_adapter_inst
(
    .start                (RestartReq_ddr1),
    //clock reset
    .clk                  (loc_clk100m         ),
    .reset                (reset               ),
    //wr_fifo Interface
    .wrfifo_clr           (wrfifo_clr          ),
    .wrfifo_clk           (clk_50M             ),
    .wrfifo_wren          (ad_out_valid        ),
```

```
    .wrfifo_din          (ad_out              ),
    .wrfifo_full         (wrfifo_full         ),
    .wrfifo_wr_cnt       (                    ),
    //rd_fifo Interface
    .rdfifo_clr          (rdfifo_clr          ),
    .rdfifo_clk          (clk_50M             ),
    .rdfifo_rden         (rdfifo_rden         ),
    .rdfifo_dout         (rdfifo_dout         ),
    .rdfifo_empty        (rdfifo_empty        ),
    .rdfifo_rd_cnt       (                    ),
    // Master Interface Write Address Ports
    .m_axi_awid          (s_axi_awid          ),
    .m_axi_awaddr        (s_axi_awaddr        ),
    .m_axi_awlen         (s_axi_awlen         ),
    .m_axi_awsize        (s_axi_awsize        ),
    .m_axi_awburst       (s_axi_awburst       ),
    .m_axi_awlock        (s_axi_awlock        ),
    .m_axi_awcache       (s_axi_awcache       ),
    .m_axi_awprot        (s_axi_awprot        ),
    .m_axi_awqos         (s_axi_awqos         ),
    .m_axi_awregion      (s_axi_awregion      ),
    .m_axi_awvalid       (s_axi_awvalid       ),
    .m_axi_awready       (s_axi_awready       ),
    // Master Interface Write Data Ports
    .m_axi_wdata         (s_axi_wdata         ),
    .m_axi_wstrb         (s_axi_wstrb         ),
    .m_axi_wlast         (s_axi_wlast         ),
    .m_axi_wvalid        (s_axi_wvalid        ),
    .m_axi_wready        (s_axi_wready        ),
    // Master Interface Write Response Ports
    .m_axi_bid           (4'b0000             ),
    .m_axi_bresp         (s_axi_bresp         ),
    .m_axi_bvalid        (s_axi_bvalid        ),
    .m_axi_bready        (s_axi_bready        ),
    // Master Interface Read Address Ports
    .m_axi_arid          (s_axi_arid          ),
    .m_axi_araddr        (s_axi_araddr        ),
    .m_axi_arlen         (s_axi_arlen         ),
    .m_axi_arsize        (s_axi_arsize        ),
    .m_axi_arburst       (s_axi_arburst       ),
    .m_axi_arlock        (s_axi_arlock        ),
    .m_axi_arcache       (s_axi_arcache       ),
    .m_axi_arprot        (s_axi_arprot        ),
    .m_axi_arqos         (s_axi_arqos         ),
    .m_axi_arregion      (s_axi_arregion      ),
    .m_axi_arvalid       (s_axi_arvalid       ),
    .m_axi_arready       (s_axi_arready       ),
```

```
    // Master Interface Read Data Ports
    .m_axi_rid          (4'b0000            ),
    .m_axi_rdata        (s_axi_rdata        ),
    .m_axi_rresp        (s_axi_rresp        ),
    .m_axi_rlast        (s_axi_rlast        ),
    .m_axi_rvalid       (s_axi_rvalid       ),
    .m_axi_rready       (s_axi_rready       )
);
```

## 1.3.5 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 1-14 所示：

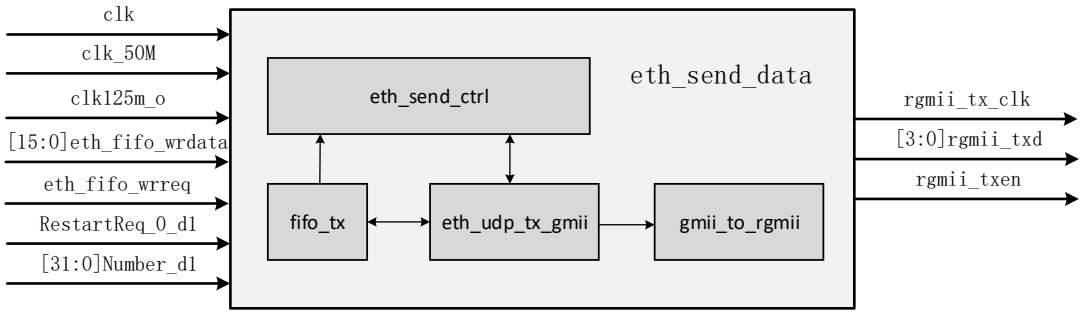


图 1-14 eth_send_data 模块基本结构框图

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 |

以太网相关部分读者自行查看 ACZ702 开发板资料的 02_逻辑文档教程中的相关内容，本章将不再重新进行说明，接下来将介绍 eth_send_ctrl 模块和 fifo_tx 模块。

### 1.3.5.1 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 1-15 所示。

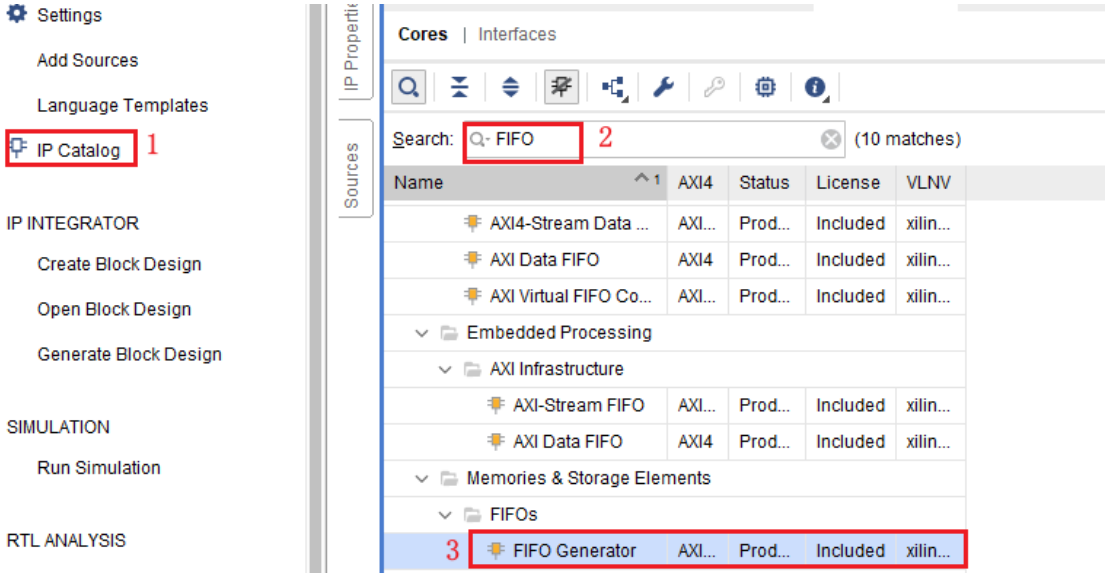


图 1-15 搜索 FIFO IP

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 1-16 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。

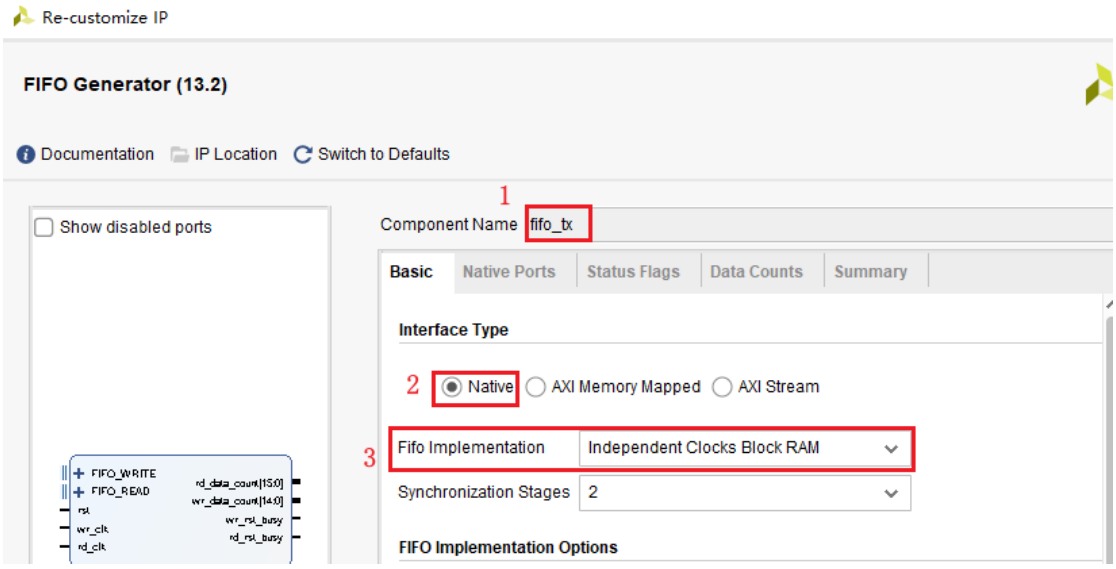


图 1-16 选择独立时钟 FIFO

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 1-17 所示。

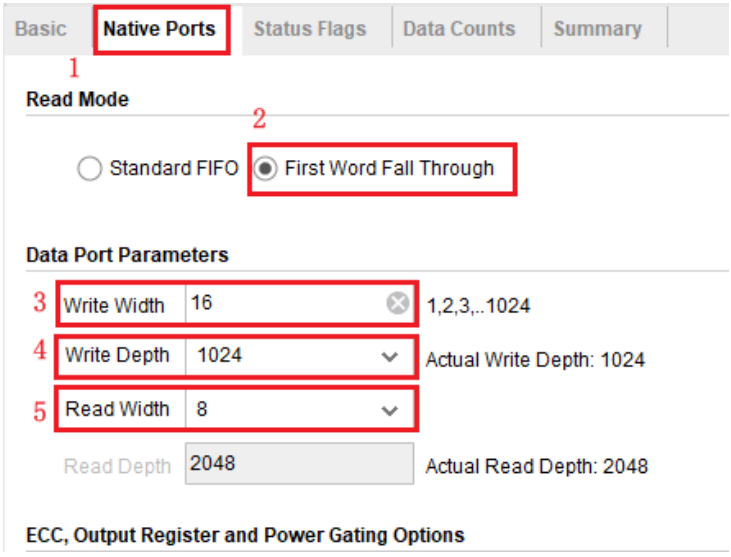


图 1-17 “Native Ports”配置界面

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 1-18 所示。

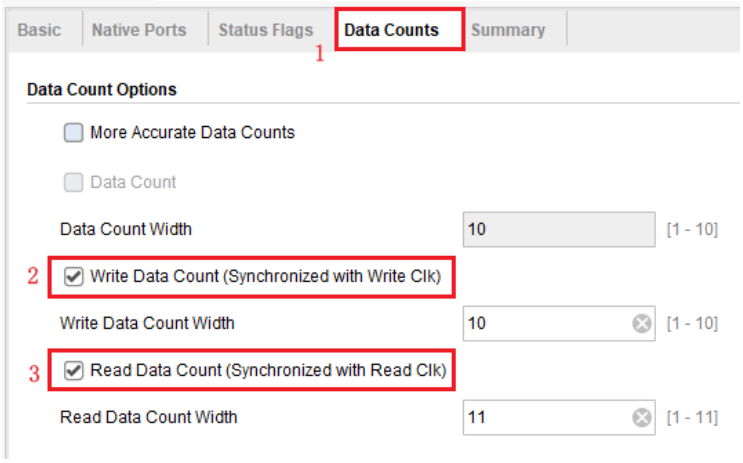


图 1-18 “Data Counts”配置界面

通过上述步骤，完成对 FIFO 的配置。

### 1.3.5.2 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 1-19 所示。

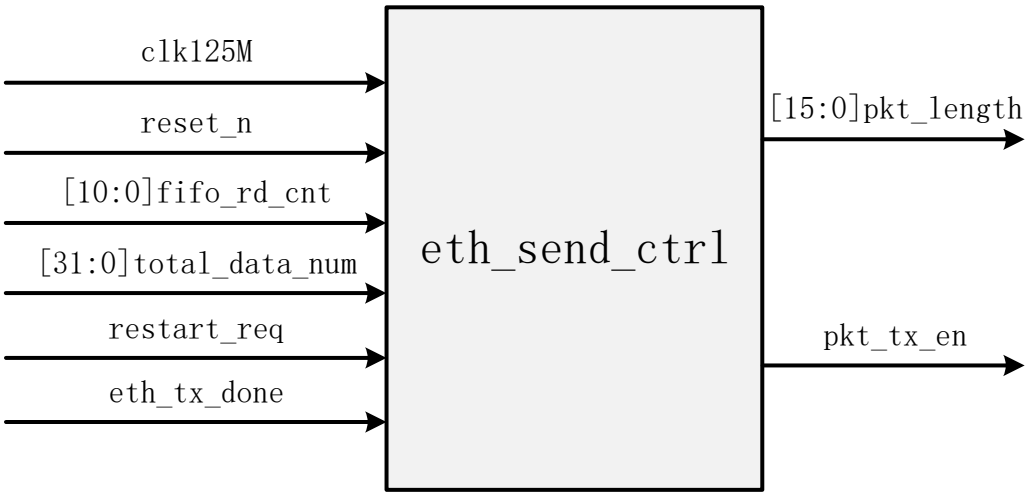


图 1-19 网口发送控制模块

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 ACM2108/108 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 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。

# 1.4 建立 SDK 工程

经过以上工作，代码设计部分的任务已经全部完成，但是用到了 PS 部分的内容，所以我们需要新建一个空的 SDK 工程用于下载程序，步骤如下。

1. 编译工程，然后依次点击 File->Export->Export Hardware，导出 bit 文件，如下图 1-20 所示。

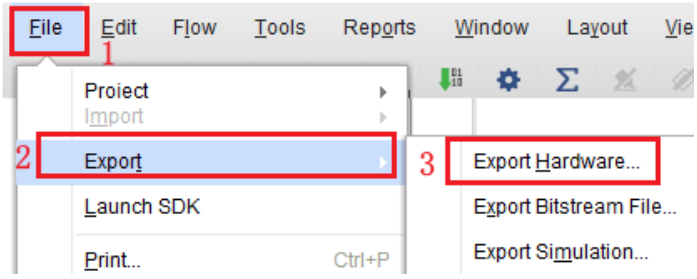


图 1-20 导出 bit 文件

2. 进入 SDK 软件，点击 File->Launch SDK，如下图 1-21 所示。

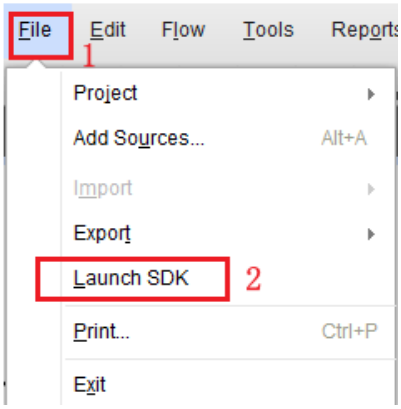


图 1-21 进入 SDK 软件

3. 进入 SDK 软件之后，依次点击 File->New->Application Project，建立一

个 SDK 工程，如下图 1-22 所示，然后给工程命名，比如命名为 ad2108_udp_ddr3，然后点击 Next，选择 Empty Application，就完成了工程的创建。

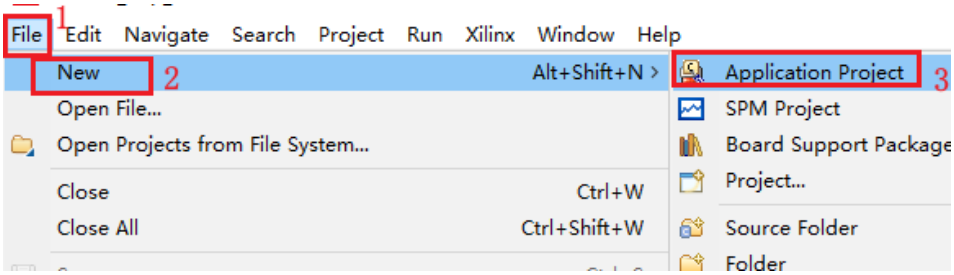


图 1-22 新建 SDK 工程

4. src 文件夹下，新建一个源文件，如下图 1-23 所示，然后在弹出的界面中给文件命名为 main.c。

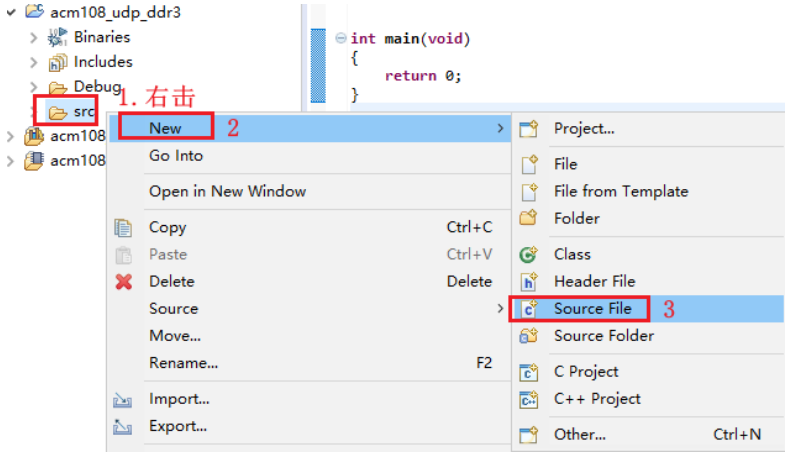


图 1-23 新建源文件

5. 源文件中添加代码，如果不添加会报错，这里添加没有意义的一句代码，如下所示：

```
int main(void)
{
    return 0;
}
```

最后 crtl+b 编译工程，没有报错，这样本次实验的设计就完成了，接下来便可以进行板级验证了，需要注意的是，如果你修改了 vivado 中的代码，你需要进行上面的步骤 1，导出 bit 文件，然后在 SDK 中重新编译工程，这样才算修改成功了。

# 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 ACZ702 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 ACM2108/108 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，ACM2108/108 模块开始采集数据，将 ACM2108/108 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

1. ACZ702 开发板一块
2. 网线一根
3. ACM2108 模块一个或者 ACM108 模块一个
4. Type-C 下载线一根
5. 电源线一根（可选）
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：

1. 使用 Type-C 线连接开发板调试接口（靠近电源接口）和电脑 USB 口
2. 将开发板电源拨码开关拨到对应侧
3. 将网线连接至 PL 侧的网口上
4. ACM2108 模块/ACM108 模块连接至 40 pin 的排针上，靠右连接，1 脚和 1 脚对应。

ACM2108 模块和 ACM108 模块硬件连接如图 1-24 和图 1-25 所示

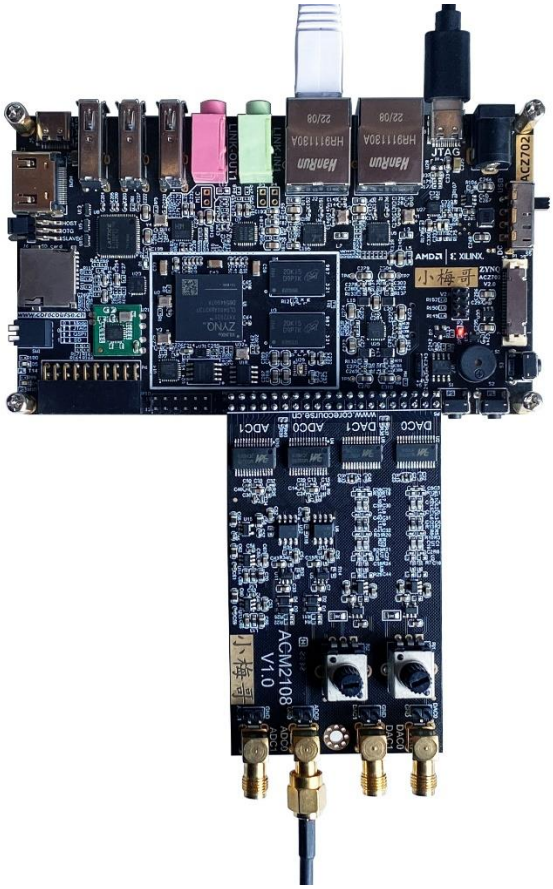


图 1-24 ACM2108 硬件连接图

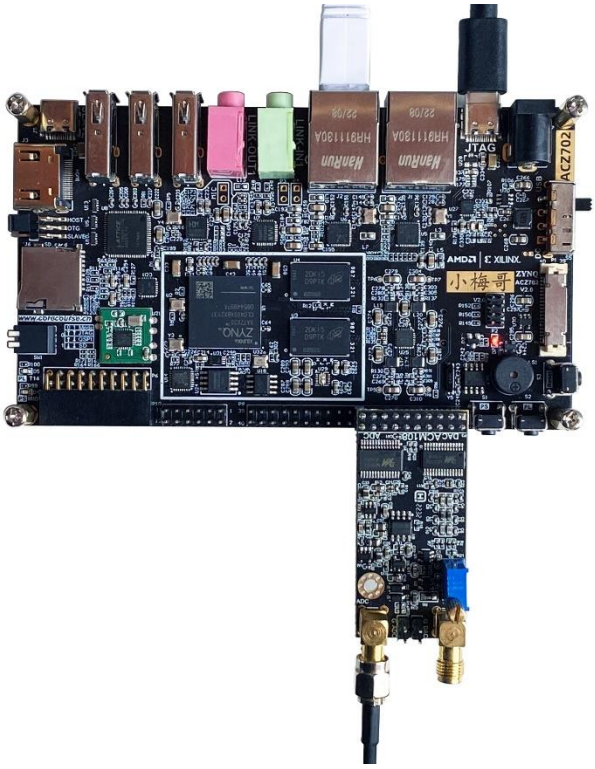


图 1-25 ACM108 硬件连接图

## 1.5.3 烧录程序

这里以烧录 ACM2108 的程序为例，在 SDK 软件中，依次点击 Run->Run Configurations，如下图 1-26 所示。

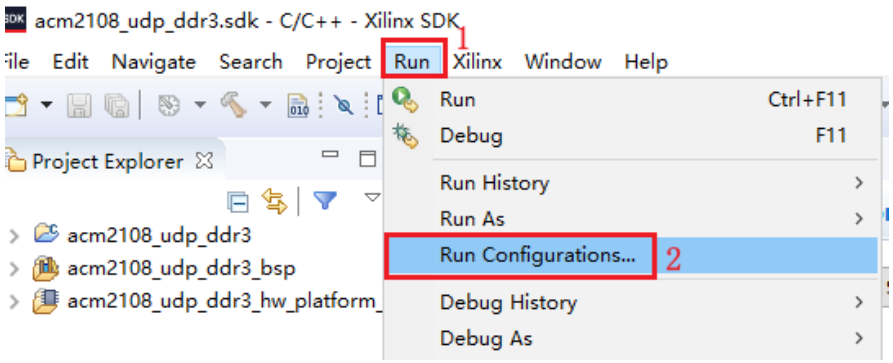


图 1-26 进入下载界面示意图

进入下载界面之后，下载 bit 文件，如下图 1-27 所示。

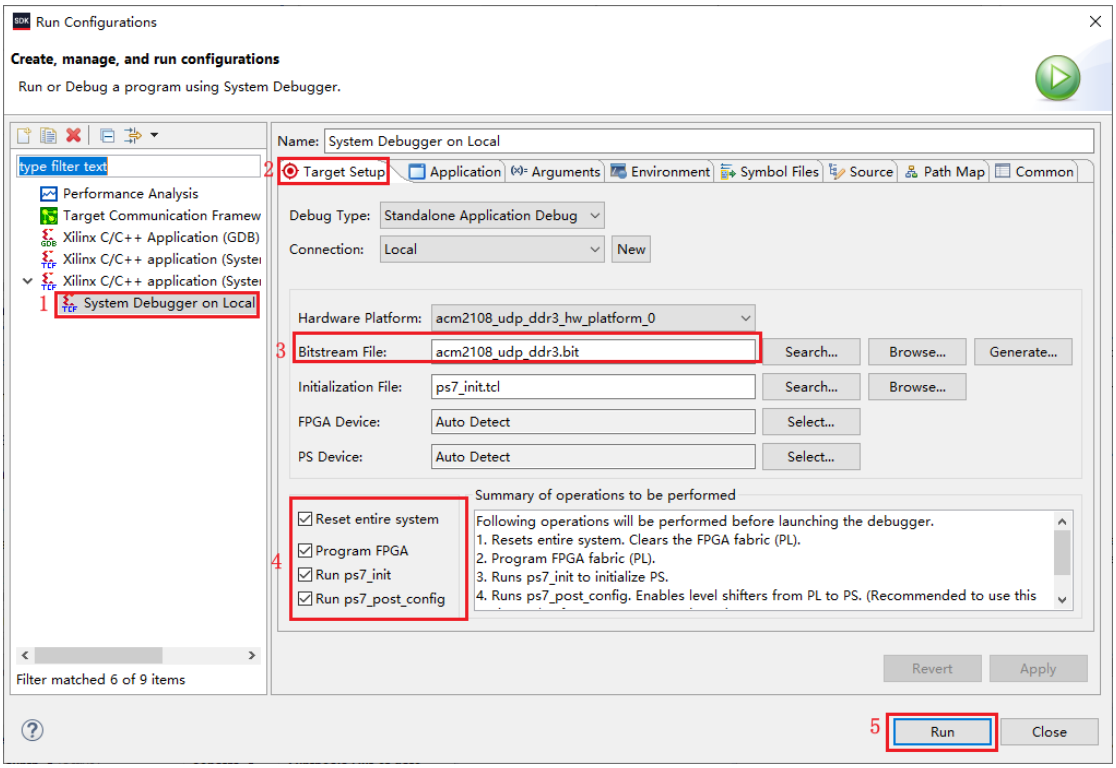


图 1-27 下载 bit 文件

下载成功之后，开发板右侧的 PL 的 LED 灯会被点亮，这时说明 PLL 锁相环工作正常，如下图 1-28 所示。

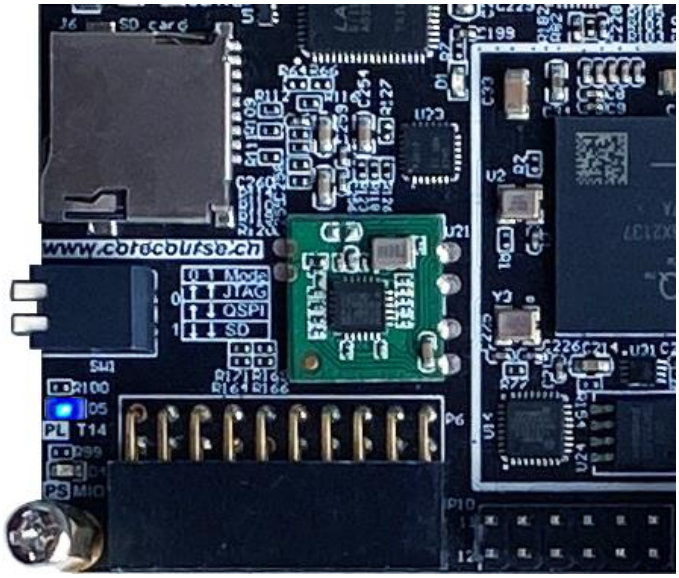


图 1-28 PL 侧 LED 灯被点亮

## 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 1-29 所示。

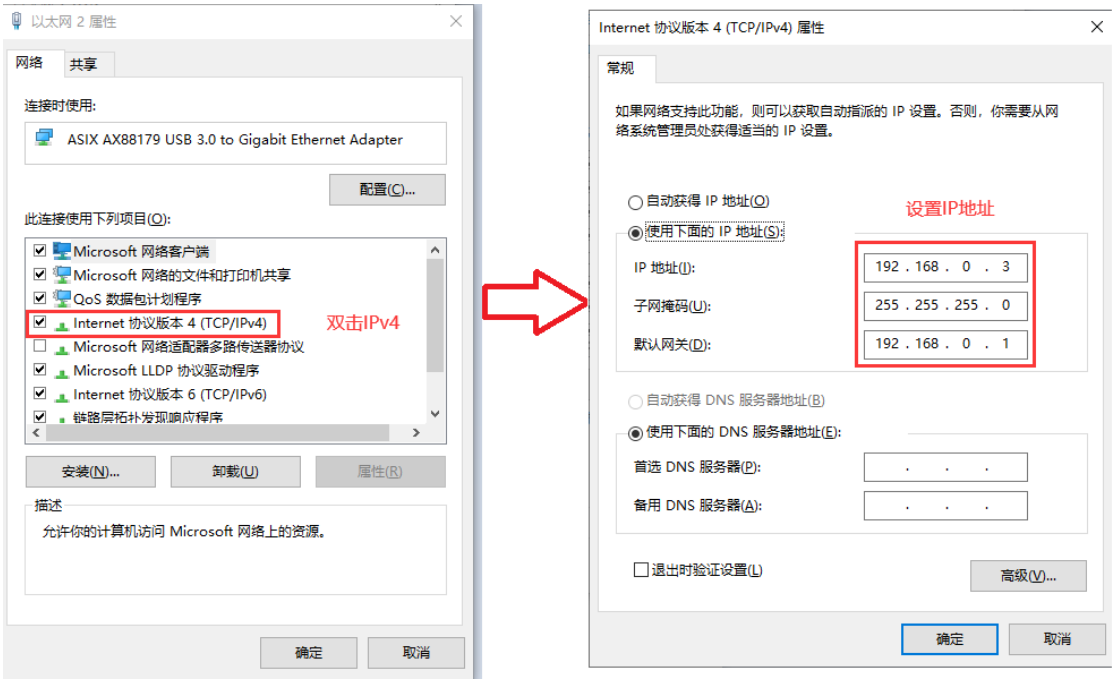


图 1-29 修改电脑 IP 地址

### 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 ACM2108/108 模块的 DAC 通道 0 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 ACZ702 开发板资料包的常用软件文件夹中找到。

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 1-30 所示。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。

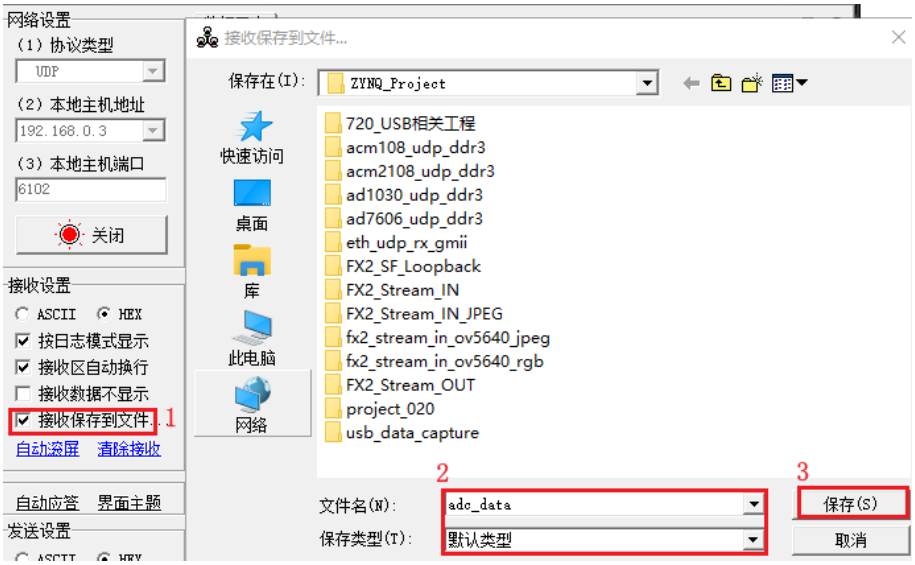


图 1-30 设置保存文件

7. 发送设置中数据类型设置为 hex 格式，如下图 1-31 所示。




图 1-31 设置发送格式为 hex 码格式

8. 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 ACM2108/108 模块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，ACM2108/108 模块就能实现以 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 ACM2108/108 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 1-32 所示：




图 1-32 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 1-33 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。

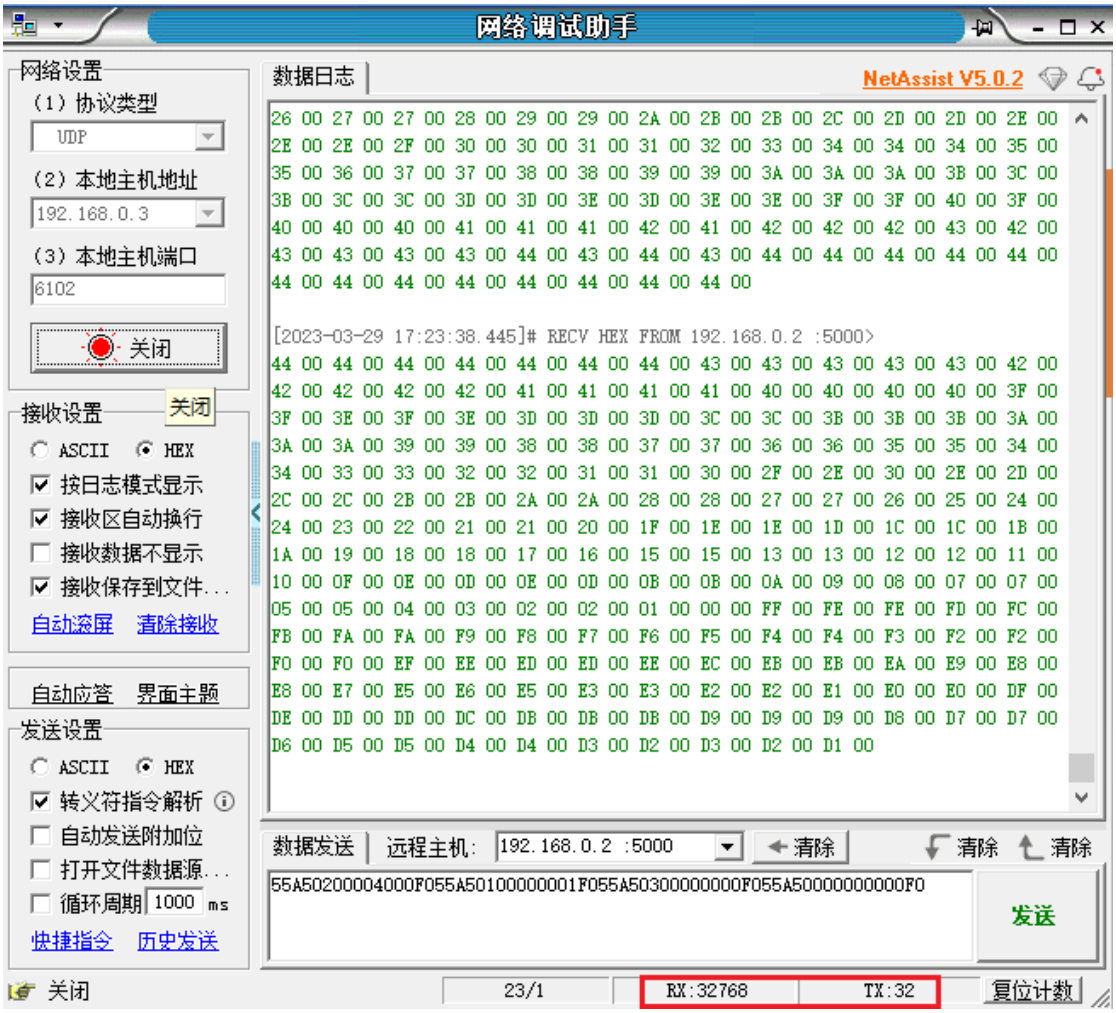


图 1-33 网络调试助手接收数据

## 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 1-34 所示，得到的波形图如下图 1-35 所示。

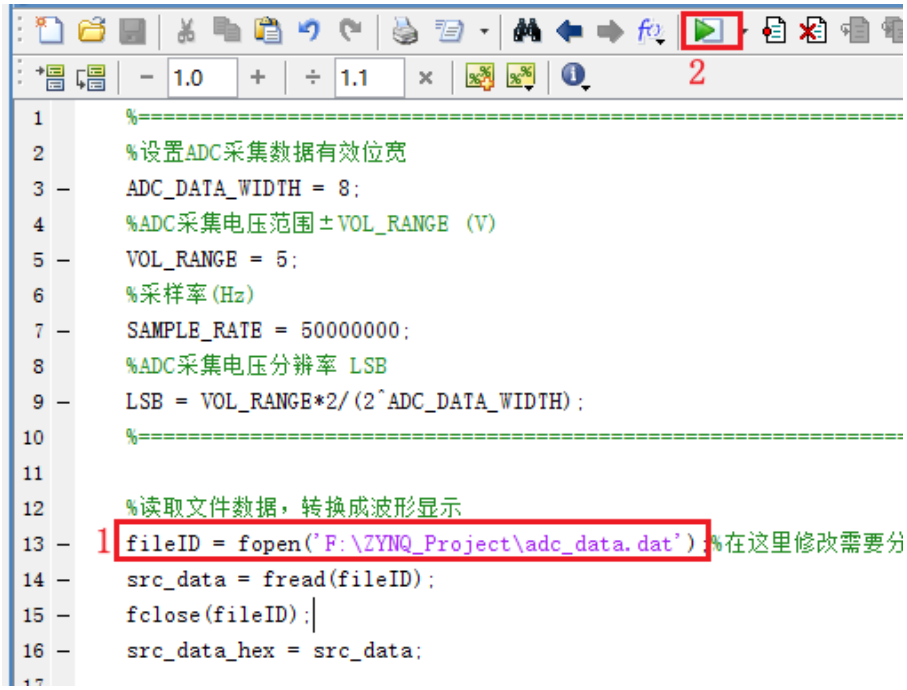


图 1-34 修改文件路径并运行

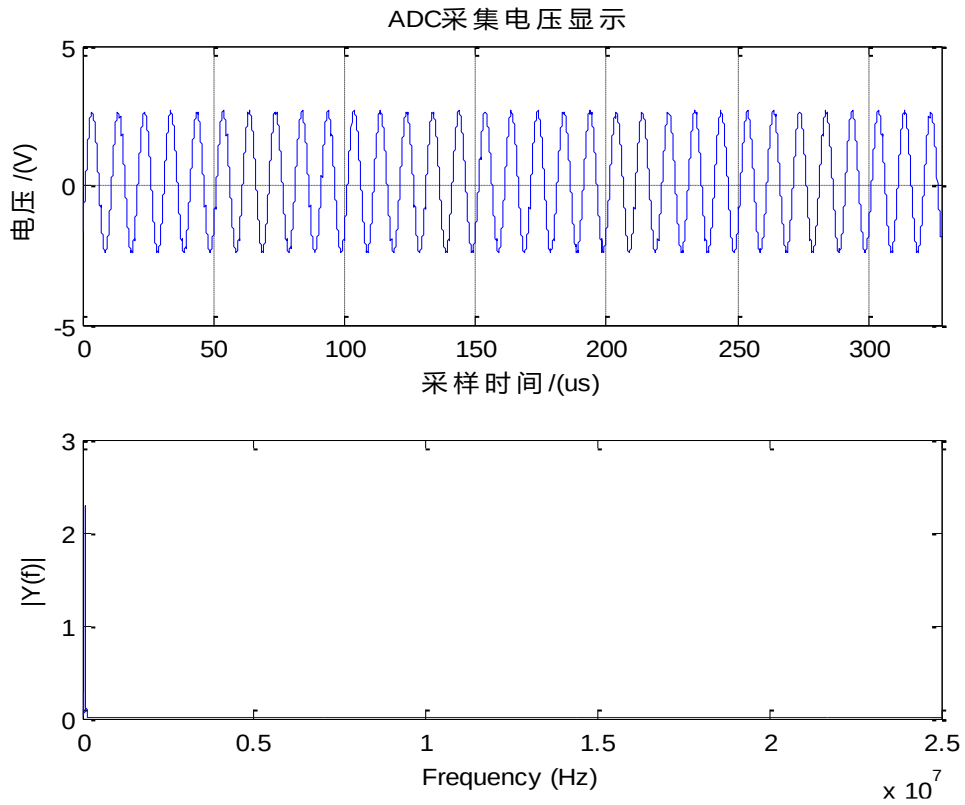


图 1-35 MATLAB 分析波形图

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 1-36 所示。

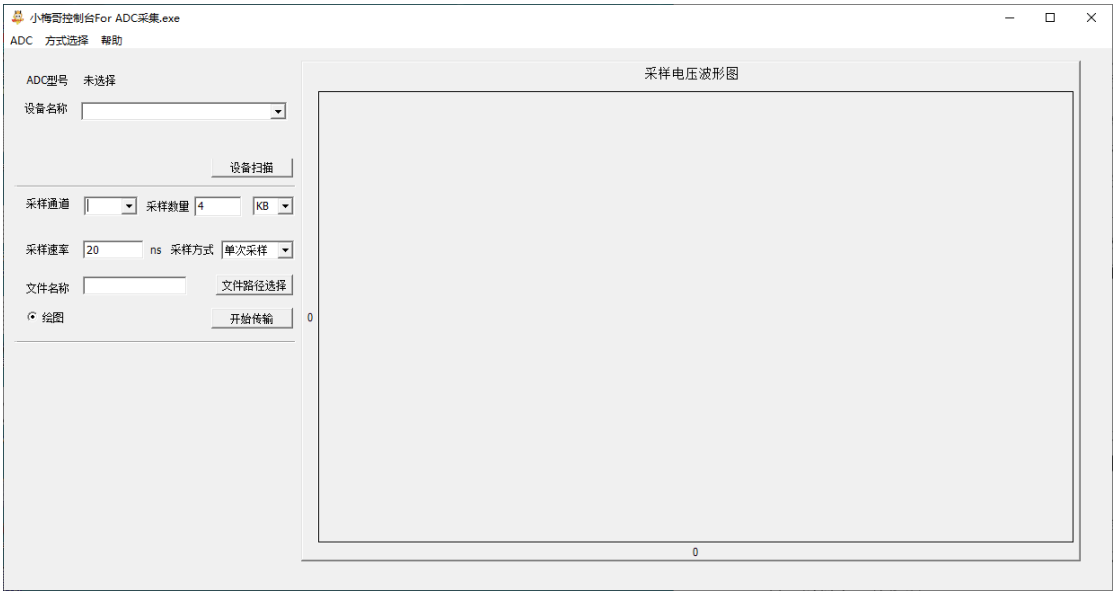


图 1-36 上位机软件初始界面显示

本次实验使用该软件的方式如下所示：

1. 点击 ADC，选择 ACM2108/ACM108。
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 1-37 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。

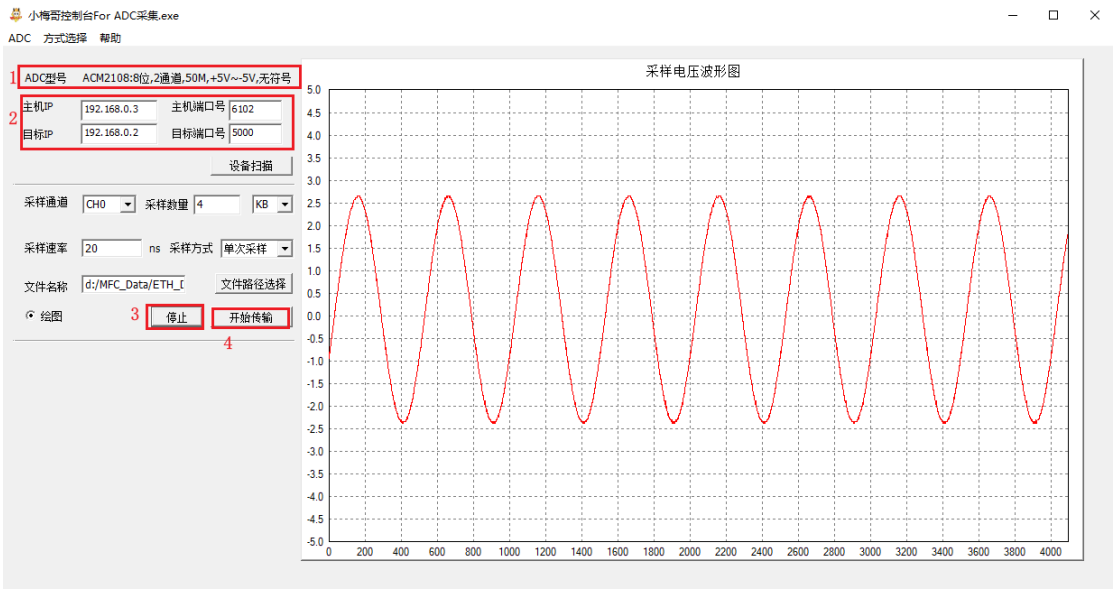


图 1-37 数据采集上位机显示图

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 1-38 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。

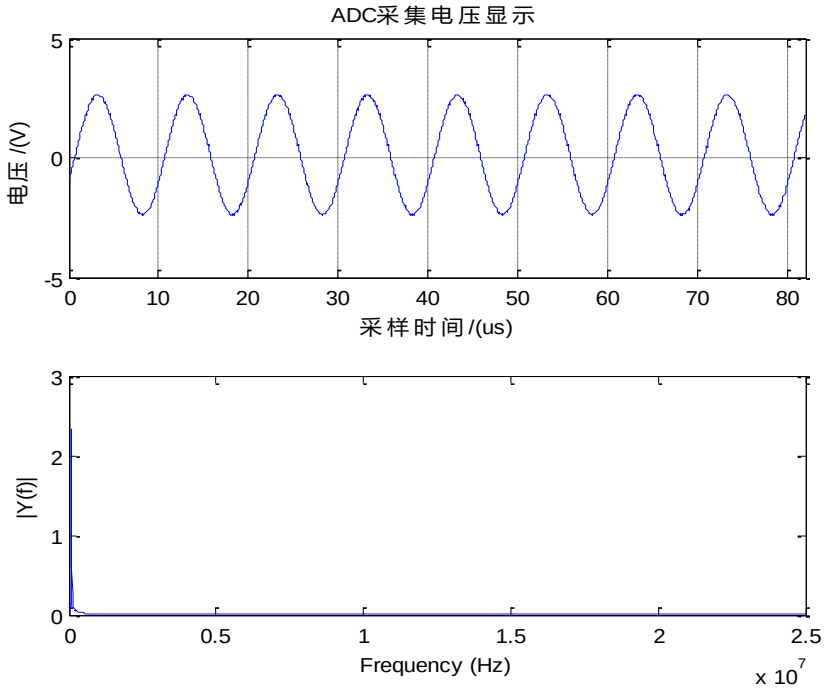

图 1-38 MATLAB 进一步波形分析图

## 1.5.7ACM2108 模块\ACM108 模块产生信号源

在我们提供的例程中，包含 DDS 模块，用户可以用杜邦线直接将 ACM2108 模块的 ADC 和 DAC 相连，进行数据采集，其中按键 S2 可以修改 DAC 的信号输出。连接方式如下图 1-39 所示，ACM108 模块的连接方式如下图 1-40 所示，将 ADC0 与 DAC0 相连，这样我们就可以通过 DAC0 输出信号给 ADC0 采集，注意不要连接到 GND。

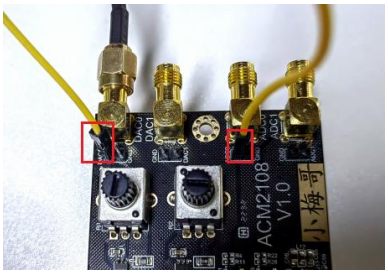


图 1-39 ACM2108 模块 ADC 和 DAC 连接示意图

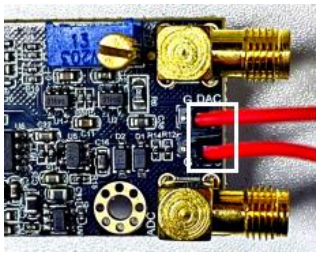

图 1-40 ACM108 模块 ADC 和 DAC 连接示意图

通过按下 S0 按键切换信号源，通过示波器观测采集到的信号源，如下图 1-41 所示为 DAC 产生的 125Khz 的正弦波信号。然后通过网口助手采集数据，MATLAB 分析数据得到波形图如下图 1-42 所示。

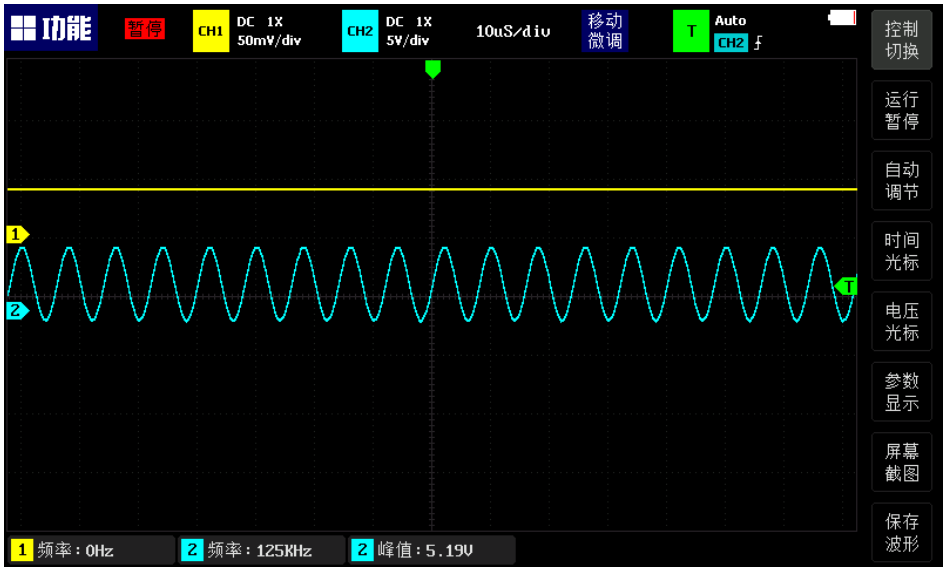


图 1-41 DAC 输出的信号

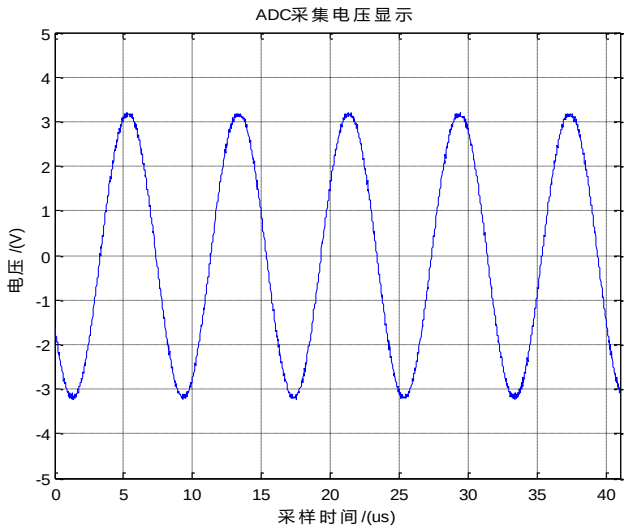


图 1-42 ADC 采集到的信号

当通过 MATLAB 观测到的信号不明显时，可以通过按键 S0 切换到更高频率的波形进行采集。按键切换次数与对应的波形的说明如下表 1-13 所示。

表 1-13 按键切换对应波形表

| 按键次数 | 对应波形 | 按键次数 | 对应波形 | 按键次数 | 对应波形 |
|---|---|---|---|---|---|
| 0 | 1.25K 正弦波 | 9 | 125K 正弦波 | 18 | 6.25M 正弦波 |
| 1 | 1.25K 方波 | 10 | 125K 方波 | 19 | 6.25M 方波 |
| 2 | 1.25K 三角波 | 11 | 125K 三角波 | 20 | 6.25M 三角波 |
| 3 | 12.5K 正弦波 | 12 | 625K 正弦波 | 21 | 12.5M 正弦波 |
| 4 | 12.5K 方波 | 13 | 625K 方波 | 22 | 12.5M 方波 |
| 5 | 12.5K 三角波 | 14 | 625K 三角波 | 23 | 12.5M 三角波 |
| 6 | 62.5K 正弦波 | 15 | 1.25M 正弦波 | | |

| | | | | | |
|---|---|---|---|---|---|
| 7 | 62.5K 方波 | 16 | 1.25M 方波 | | |
| 8 | 62.5K 三角波 | 17 | 1.25M 三角波 | | |

## 1.6 思考与总结

本次实验介绍了基于 ACM2108/108 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 ACM2108/108 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。