1 USB 应用之传输 ACM2108/ACM108 数据采集

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 “acx720_acm2108_ddr3_usb.rar”和“acx720_acm108_ddr3_usb.rar”。

1.1 文档所涉及文件说明

本文档如有更新，会发布在 www.corecourse.cn 网站。如您需查看并获取更新版本，只需要在该网站以“ACM2108”为关键词搜索，即可查找到相关内容。

版本记录：

V1.0	首次发布	2023.02.06
……	……	……

本教程配套的文件压缩包名：acx720_acm2108_108_ddr3_usbV1.0.rar，解压后可以得到配套工程文件、上位机软件和 matlab 绘图代码文件，对于压缩包中的文件说明如下：

	acx720_acm2108_ddr3_usb.rar：使用 ACM2108 实验使用到的 vivado 工程。其中包括各个模块的源文件代码: USB 接收数据模块、USB 指令接收和指令解析模块、数据转换模块、ddr3 的含 fifo 的 2 端口封装模块、USB 数据发送模块，数据采集模块等。
	acx720_acm108_ddr3_usb.rar：使用 ACM108 实验使用到的 vivado 工程。其中包括各个模块的源文件代码: USB 接收数据模块、USB 指令接收和指令解析模块、数据转换模块、ddr3 的含 fifo 的 2 端口封装模块、USB 数据发送模块，数据采集模块等。
	FX2_USB 数据采集上位机工具文件夹： cypress 上位机，用来烧写 USB 固件程序以及简单的数据发送接收测试。
	data：测试数据文件夹，用来存放本次实验过程中，生成的数据文件，用户可以使用 Matlab 软件对我们提供的测试数据文件进行数据分析对比。
	小梅哥控制台 For ADC 采集：使用 USB 上位机软件采集数据时，无需设置发码指令，只需要在软件界面中填好采集信息即可，相比于通过 cypress 上位机进行数据采集，操作方式更简单直接。
	ADCdata_to_wave_v1_1.m：Matlab 源文件代码，对于 USB 传输过来的数据，我们得到的只是一个数据文件，如何判断采集到的数据是否正确，这里采用 Matlab 软件进行数据数据分析，用户可以通过修改我们提供的 matlab 源文件中的文件路径，进行数据分析。

1.2 系统整体设计

通过电脑上的 FX2_USB 调试工具 CyControl，将命令帧进行发送，然后通过 ACX720 开发板上的 USB 处理器接收数据，随后从 USB 下发的数据中解析出命令，从而实现对 ACM2108 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 模块开始采集数据，并将 ACM2108 模块采集的数据存储进 DDR3 中。最后将 DDR3 中存储的数据通过 USB 传输至电脑，电脑端对采集到的数据使用 MATLAB 进行进一步的分析。系统的整体设计框图如下图 1-1 所示。

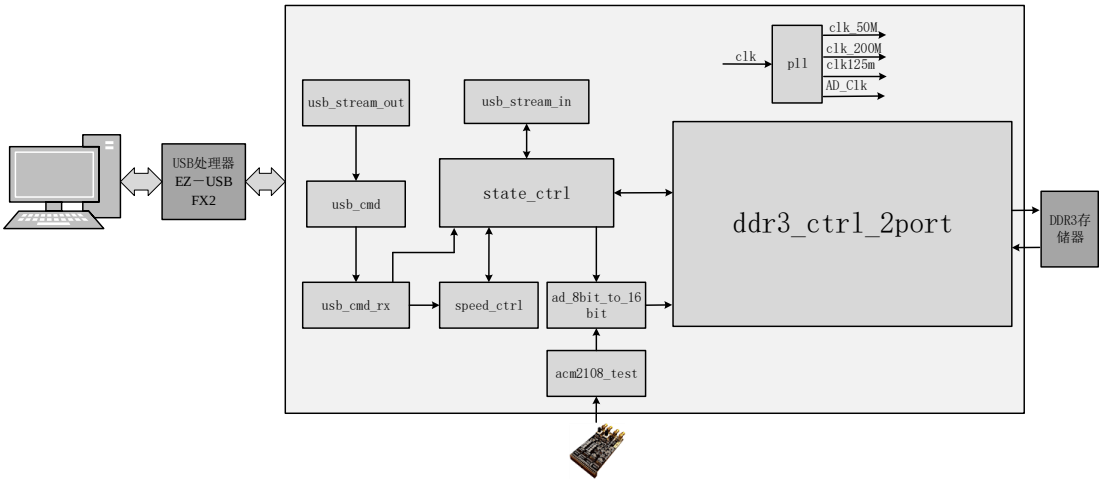


图 1-1 USB 应用之传输 ACM2108 数据采集整体设计框图

对于上图中各个模块的功能介绍如下：

1. PLL 模块：锁相环模块，生成本次实验每个模块所需要的工作时钟，其中输入时钟为 50M 的系统时钟，由开发板上的晶振提供，输出 200M 的时钟给到 DDR3 控制器使用，输出 50M 的时钟给其他模块使用。输出 AD_Clk 和 clk125M 给 ACM2108 测试模块 acm2108_test 使用。
2. usb_stream_out 模块：USB 数据流发送控制模块，不断的将端点 2 中的数据读取出来，数据读取后直接作为端口输出。
3. usb_cmd 模块：接收转命令模块，对 USB 接收到的数据进行分析，提取出每个控制命令帧。
4. usb_cmd_rx 模块：指令转控制模块，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并分别输出到对应的模块。
5. speed_ctrl 模块：采样速率控制模块，控制 ACM2108 的采样速率。
6. ad_8bit_to_16bit 模块：数据转换模块，将 ACM2108 发送过来的 8 位数

据转换成 16 位的数据，方便我们后续进行上位机采集数据。

7. acm2108_test：ACM2108 数据采集测试模块，在该模块中我们也设计了 DDS 模块，方便用户通过该模块自身的 DAC 芯片产生信号源。
8. state_ctrl 模块：ADC 采集数据 DDR3 缓存 USB 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。
9. ddr3_ctrl_2port 模块：ddr3 的含 FIFO 的 2 端口封装模块，主要负责整个数据的存储功能。
10. usb_stream_in 模块：USB 数据流发送模块，将最终采集到的数据通过 USB 发送出去。

本次实验需要设计的模块包括usb_cmd 模块、usb_cmd_rx 模块、speed_ctrl 模块、ad_8bit_to_16bit 模块、acm2108_test 模块和 state_ctrl 模块，关于 USB 数据传输和 DDR 双端口模块的设计和介绍请参看 ACX720 资料中盘 A:\01_教材文档文件夹下的 02 号文件。

1.3 ACM2108 高速 ADC+DAC 模块简介

ACM2108 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 ACM2108 模块实物图。

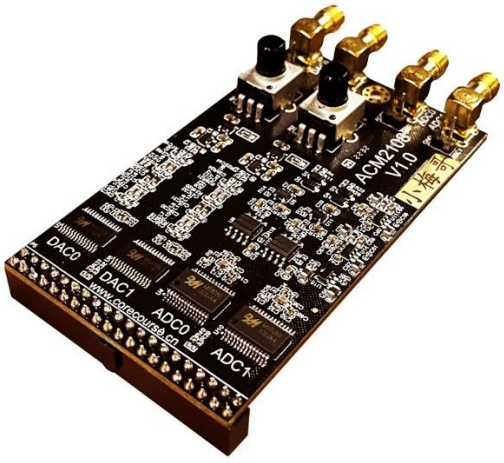


图 1-2 ACM2108 模块实物图

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。

1.4 模块设计

下面给将对本次实验需要设计的模块进行介绍。

1.4.1 接收转命令模块

接收转命令模块 usb_cmd，将 USB 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 1-3 所示。

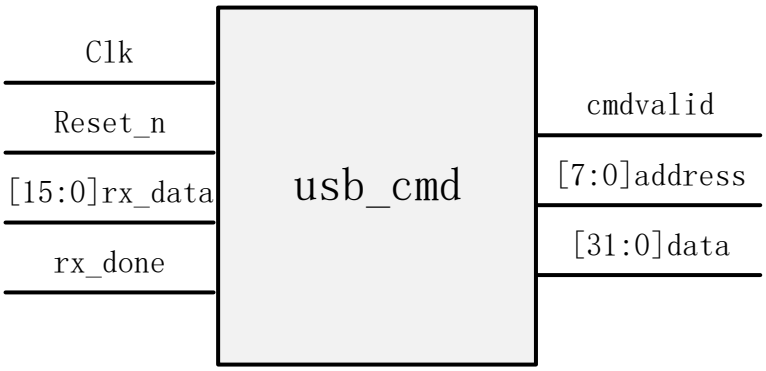


图 1-3 接收转命令模块框图

模块信号说明如下表 1-1 所示。

表 1-1 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，低电平复位
rx_data [15:0]	I	USB 接收数据流模块接收到的 16 位数据
rx_done	I	USB 一次数据接收完成标志信号
cmdvalid	O	输出命令有效标志信号
address[7:0]	O	配置 ACM2108 的寄存器地址信号
data[31:0]	O	写入到寄存器中的数据

USB 一次发送的命令帧内容为 32 个字节，为了实现通过 USB 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 1-2 所示：

表 1-2 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 USB 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 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，当检测到了 rx_done 信号，data_str 连续 4 次存储 USB 接收到的数据，组成 32 字节的命令帧。代码如下所示：

```
always@(posedge Clk)
if(rx_done)begin
    data_str[3] <= #1 rx_data;
    data_str[2] <= #1 data_str[3];
    data_str[1] <= #1 data_str[2];
    data_str[0] <= #1 data_str[1];
end
```

最后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 <= #1 0;
    data <= #1 0;
    cmdvalid <= #1 0;
end else if(r_rx_done)begin
    if((data_str[0][7:0] == 8'h55) && (data_str[0][15:8] == 8'hA5) &&
(data_str[3][15:8] == 8'hF0))begin
        data[7:0] <= #1 data_str[3][7:0];
        data[15:8] <= #1 data_str[2][15:8];
        data[23:16] <= #1 data_str[2][7:0];
        data[31:24] <= #1 data_str[1][15:8];
        address <= #1 data_str[1][7:0];
        cmdvalid <= #1 1;
    end
    else
        cmdvalid <= #1 0; end
else
```

```
    cmdvalid <= #1 0;
```

1.4.2 指令转控制模块

指令转控制模块 usb_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下表 1-3 所示：

表 1-3 寄存器说明表

名称	地址	位宽	功能简介
start_sample	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
adc_ch_sel	1	2	通道设置寄存器，共 2 位。ACM2108 模块提供了 ADC1、ADC2 两个通道进行数据采集，ACM108 只有一个通道可以使用（01：通道 1；10：通道 2）
set_sample_num	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。
set_sample_speed	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下图 1-4 所示。

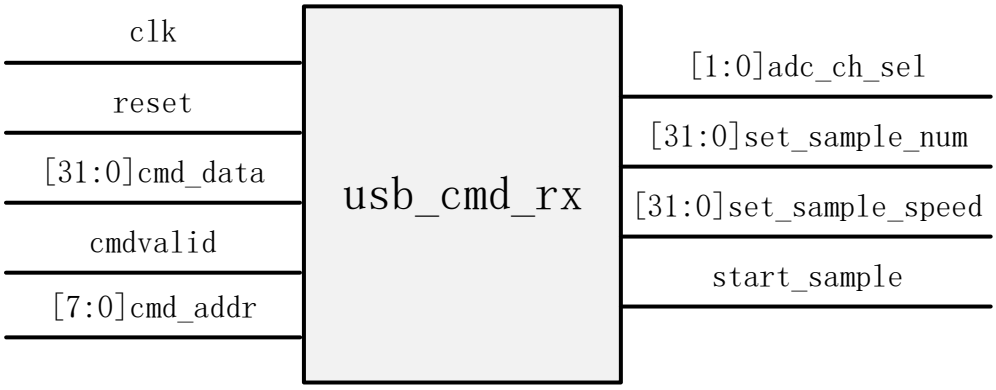


图 1-4 指令转控制模块结构框图

模块信号说明如下表 1-4 所示。

表 1-4 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	I	模块复位信号，高电平有效
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
adc_ch_sel [1:0]	O	通道设置寄存器
set_sample_num [31:0]	O	数据个数寄存器

set_sample_speed [31:0]	O	ADC 采样速率控制寄存器
start_sample	O	重新开始采集请求信号

根据表 1-3 中的内容， 地址 cmd_addr 为 0 时，产生 RestartReq 信号；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 posedge reset)
if(reset)begin
    adc_ch_sel <= 2'b00;
    set_sample_num <= 32'd256;//采样最大数量设定为 4G
    start_sample <= 1'b0;
    set_sample_speed <= 32'd0; //50M 采样率
end
else if(cmdvalid)begin
    case(cmd_addr)
        0: start_sample <= 1'b1;
        1: adc_ch_sel <= cmd_data[1:0];
        2: set_sample_num <= cmd_data[31:0];
        3: set_sample_speed <= cmd_data[31:0];
        4:
        begin
            adc_ch_sel <= cmd_data[1:0];
            set_sample_num <= cmd_data[23:8];
            start_sample <= 1'b1;
        end
    default:;
    endcase
end
else
    start_sample <= 1'b0;
```

1.4.3 采样速率控制模块

采样速率控制（speed_ctrl）模块用来控制 ACM2108 的采样速率，该模块的结构框图如下图 1-5 所示。

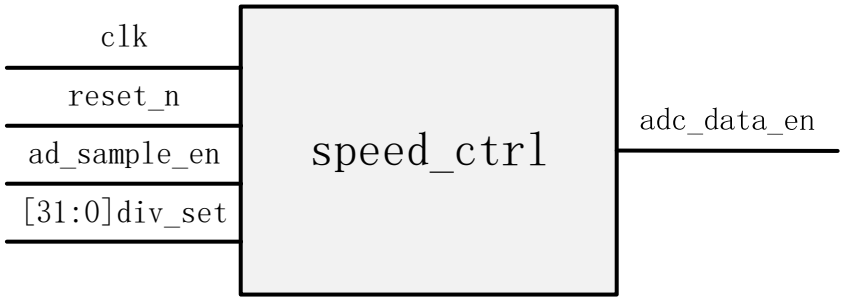


图 1-5 采样速率控制模块

对该模块的信号说明如下表 1-5 所示。

表 1-5 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，低电平复位
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 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;
```

1.4.4 ACM2108 测试模块

ACM2108 数据采集测试模块（acm2108_test），在该模块中我们也设计了

DDS 模块，方便用户通过该模块自身的 DAC 芯片产生信号源，同时也验证了 ACM2108 模块上 DAC 模块的模拟信号输出功能。该模块的基本框图如下图 1-6 所示：

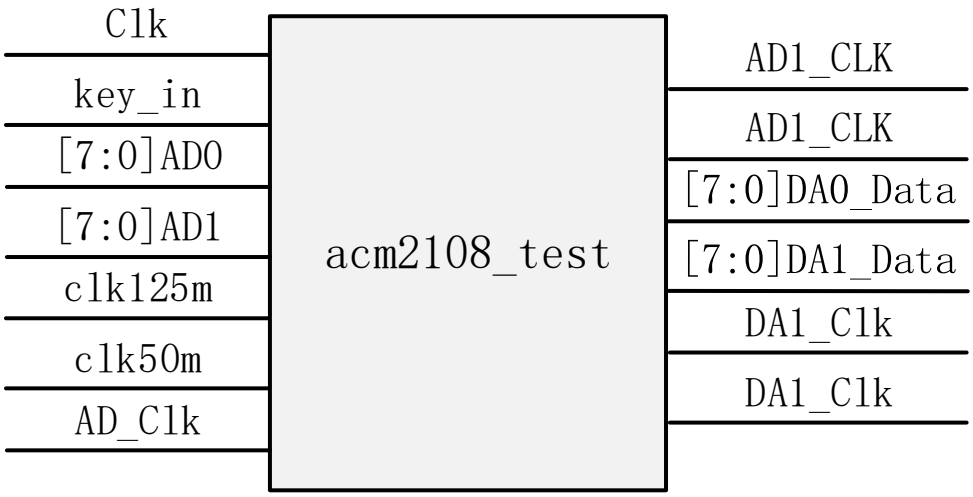


图 1-6 ACM2108 数据采集测试模块基本框图

对该模块的信号说明如下表 1-6 所示。

表 1-6ACM2108 数据采集测试模块信号说明表

端口名	端口类型	描述
clk	input	系统时钟 50MHz
key_in	input	按键切换信号
AD0	input	ACM2108 通道 0 的 8 位数据输入信号
AD1	input	ACM2108 通道 1 的 8 位数据输入信号
clk125m	input	DDS 单元工作时钟 125MHz
clk50m	input	PLL 输出的 50M 的时钟
AD_Clk	input	输入的 ADC 端口工作时钟
AD0_CLK	output	ADC0 端口工作时钟
AD1_CLK	output	ADC1 端口工作时钟
DA0_Data	output	DA0 输出数据信号
DA1_Data	output	DA1 输出数据信号
DA0_Clk	output	DAC0 端口工作时钟
DA1_Clk	output	DAC1 端口工作时钟

下面将介绍模块部分代码的实现，具体完整代码请自行查看源文件。

添加 DDS 模块代码，关与 DDS 的原理及其实现请查看 720 资料盘 C 中的相关视频课程，调用代码如下所示：

```
DDS_Module DDS_Module(
    .Clk(clk125m),
    .Rst_n(1'd1),
    .EN(1'd1),
```

```
    .MODE(MODE),
    .DA_Clk(DA_Clk),
    .DA_Data(DA_Data)
);
```

添加独立按键消抖模块，关于该模块的设计请查看 720 资料中盘 A\01_教材文档\02 号文档的“独立按键消抖设计与验证”一节的内容，调用代码如下所示：

```
key_filter key_filter0(
    .Clk(Clk),       //50M 时钟输入
    .Rst_n(1'd1),     //模块复位
    .key_in(key_in),    //按键输入
    .key_flag(key_flag), //按键标志信号
    .key_state(key_state) //按键状濁信叮
);
```

按键一共可以切换 23 种波形，每次检测到按键按下，模式加 1，直到加到 23 清 0，代码如下所示：

```
always@(posedge Clk)
if(key_flag & (!key_state))begin
if(MODE >= 23)
    MODE <= 0;
else
    MODE <= MODE + 1'd1;
end
else
    MODE <= MODE;
```

23 种对应波形的说明如下表 1-7 所示。

表 1-7 23 种波形与按键次数说明表

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		

1.4.5 数据位扩展模块

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 1-7 所示。

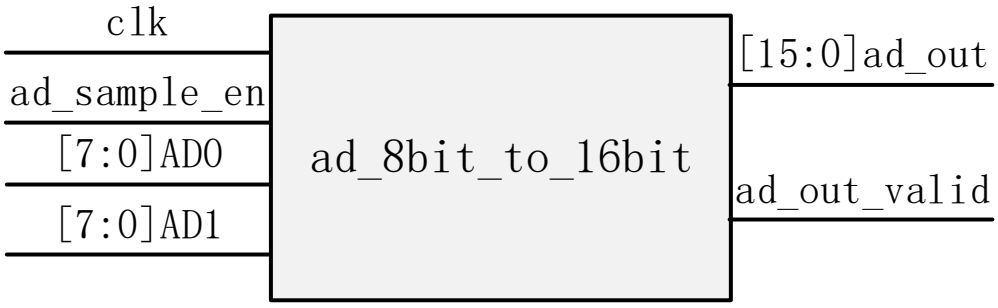


图 1-7 数据位扩展模块结构框图

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

端口名	端口类型	描述
clk	input	系统时钟 50MHz
ad_sample_en	input	ACM2108 模块进行数据采集使能信号
AD0	input [7:0]	ACM2108 通道 0 的 8 位数据输入信号
AD1	input [7:0]	ACM2108 通道 1 的 8 位数据输入信号
ad_out	output[15:0]	16 位数据输出信号
ad_out_valid	output	输出数据有效信号

ACM2108 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，然后将得到的有符号数据通过补 0 的方式实现 16 位的数据输出。模块部分代码如下所示：

```
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;

   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
```

1.4.6 state_ctrl 模块

DDR3 双端口控制器的控制信号如何产生以及 USB 何时开始往外发送数据，这些问题都可以通过一个控制模块 state_ctrl 解决。该模块的结构框图如下图 1-8 所示：

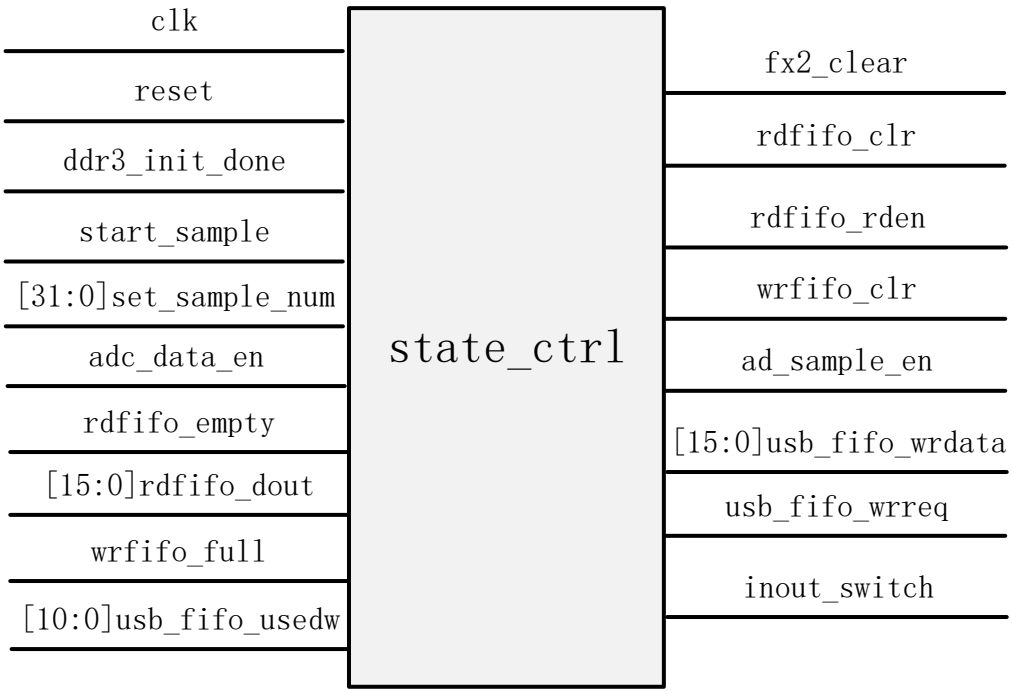


图 1-8 state_ctrl 模块结构框图

对于该模块的信号说明如下表 1-9 所示。

表 1-9state_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
reset	I	模块复位信号，高电平复位
ddr3_init_done	I	DDR3 初始化完成标志信号
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 的读数据输出，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
usb_fifo_usedw[10:0]	I	USB 发送数据流的模块的写 FIFO 计数
fx2_clear	O	USB 清除信号
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
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
ad_sample_en	O	ADC 采样使能标志信号
usb_fifo_wrdata[15:0]	O	USB FIFO 需要发送的 16 位数据
usb_fifo_wrreq	O	USB FIFO 的写请求信号
inout_switch	O	0: read，由 PC 向 FPGA 下发指令　1:write，由 FPGA 向 fx2 芯片继而向 PC 上传数据

DDR3 双端口模块（ddr3_ctrl_2port）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 应该与 ad_10bit_to_16bit 模块的 clk 保持一致为 50M；rdfifo_clk 应该与 usb_stream_in 模块的 usb_fifo_wrclk 保持一致为 50M；wrfifo_wren 信号由 ADC 模块输出数据有效信号 ad_out_valid 和 ADC 采样结果存储使能信号 adc_data_en 相与得到；wrfifo_din 信号为数据位扩展模块输出的 16 位的数据信号 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。

USB 在每次发送数据之前，我们都应该先将其复位，这样做的目的是清除 USB 中遗留的数据，保证 USB 每次发送的都是 ADC 模块当前采集到的数据。USB 在发送数据时都是以 512 字节（USB 数据包大小）的整数倍进行发送的。

根据上述描述，我们可以通过状态机实现该模块的功能。定义状态如下表

1-10 所示。

表 1-10 state_ctrl 状态控制模块状态描述表

状态值	状态名称	状态意义
0	IDLE	空闲状态
1	DDR_WR_FIFO_CLEAR	DDR 写 FIFO 清除状态
2	ADC_SAMPLE	ADC 采样数据状态
3	DDR_RD_FIFO_CLEAR	DDR 读 FIFO 清除状态
4	RESET_USB	复位 USB 状态
5	DATA_SEND_START	数据发送状态
6	DATA_SEND_WORKING	数据发送完成状态

下面编写每个状态对应的代码。

1. IDLE 状态

当处于空闲状态并且 DDR3 初始化完成之后，产生启动传输开始信号 start_sample_rm。代码如下所示：

```
always@(posedge clk or posedge reset)begin
if(reset)
    start_sample_rm <= 1'b0;
else if(state==IDLE && ddr3_init_done==1'b1)
    start_sample_rm <= start_sample;
else
    start_sample_rm <= 1'b0;
  end
```

当产生 start_sample_rm 信号之后将 inout_switch 至 1，代表由 FPGA 向 fx2 芯片继而向 PC 上传数据，跳转到 DDR_WR_FIFO_CLEAR 状态。空闲状态代码代码如下所示：

```
begin
    if(start_sample_rm)begin
        state<=DDR_WR_FIFO_CLEAR;
        inout_switch<=1'b1;
    end
    else begin
        state<=state;
        inout_switch<=1'b0;
    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'b0;
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。DDR 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(ddr3_init_done==1'b0)
    wrfifo_clr<=1'b1;
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，当产生 adc_data_en 信号之后，我们就将 adc_sample_cnt 计数值加 1，对 ADC 采集的数据进行计数。代码如下所示：

```
always@(posedge clk or posedge reset)
if(reset)
    adc_sample_cnt<=32'd0;
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;
```

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'b0;
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 RESET_USB 状态，fx2_clear 信号拉高，开始清除 USB，读 FIFO 清除状态代码如下所示：

```
begin
    if(!rdfifo_empty && (rdfifo_clr_cnt==9))begin
        state<=RESET_USB;
        fx2_clear <= 1'b1;
    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(ddr3_init_done==1'b0)
    rdfifo_clr<=1'b1;
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. RESET_USB 状态

复位 USB 状态是为了清除 USB 中遗留的数据，当进入该状态之后，USB 复位计数器 rst_usb_cnt 开始计数，计数到一定值之后，拉低 fx2_clear，使 USB 内遗留数据能够充分清除。代码如下所示：

```
begin
    if(rst_usb_cnt >= 20'hffff0)begin
        rst_usb_cnt <= 0;
        state<=DATA_SEND_START;
    end
    else if(rst_usb_cnt >= 20'h7fff0)begin
        rst_usb_cnt <= rst_usb_cnt + 1'd1;
        fx2_clear <= 1'b0;
    end
    else
        rst_usb_cnt <= rst_usb_cnt + 1'd1;
end
```

6. DATA_SEND_START 状态

进入DATA_SEND_START 状态之后，state 直接跳转到数据发送状态，USB 启动传输，代码如下所示：

```
begin
    state <= DATA_SEND_WORKING;
end
```

7. DATA_SEND_WORKING 状态

进入数据发送状态之后，当发送数据计数器 send_data_cnt 计数到需要采集的数据个数 set_sample_num 时，跳转到 IDLE 状态，完成一次数据采集发送；当 USB 发送 FIFO 计数小于 512 时，使 DDR3 双端口控制器的读 FIFO 使能信号 rdfifo_rden 为高电平，开始从 DDR3 中读数据；每发送一个 16bit 数据，如果不满足前两个条件，则重新回到本状态。代码如下所示：

```
if(send_data_cnt>=set_sample_num-1'b1)begin
    state <= IDLE;
    rdfifo_rden <= 1'b0;
end
else if(usb_fifo_usedw < 512) begin
    rdfifo_rden <= 1'b1;
    state <= DATA_SEND_WORKING;
end
else begin
//*//每发送一个 16bit 数据，如果不满足 if 条件，则重新回到本状态
    rdfifo_rden <= 1'b0;
    state <= DATA_SEND_WORKING;
end
```

```
end
```

当 rdfifo_rden 为 1 的时候，每个时钟上升沿到来之后，send_data_cnt 计数值加 1，实现对 USB 发送的数据进行计数，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
```

当 rdfifo_rden 信号到来之后，我们需要产生 USB 写 FIFO 请求信号并且需要将从 DDR 读出的数据提取出来，最终交由 USB 数据流发送控制模块进行处理，代码如下所示：

```
always@(posedge clk or posedge reset)
if(reset) begin
    usb_fifo_wrreq <= 1'b0;
    usb_fifo_wrdata <= 16'd0;
end
else if(rdfifo_rden) begin
    usb_fifo_wrreq <= 1'b1;
    usb_fifo_wrdata <= rdfifo_dout;
end
else begin
    usb_fifo_wrreq <= 1'b0;
    usb_fifo_wrdata <=16'd0;
end
```

DDR 控制器的使用请查看 720 资料中 02 号文档的“DDR 控制器 Example Design 的使用”一章的内容。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

经过以上工作，代码设计部分的任务已经全部完成，接下来就可以进行板级验证了。本次实验的板级验证环节，主要验证：通过电脑上 FX2_USB 调试工具 CyControl，将命令帧进行发送， 然后通过 ACX720 开发板上的 USB 处理器接收，随后从 USB 下发的数据中解析出命令，最终实现对 ACM2108 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 开始采集数据，

将 ACM2108 采集的数据通过 USB 传输到电脑。电脑端将接收到的数据以文件的形式进行保存，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，便可以实时观察到数据波形变化，使用起来非常方便。

1.5.1 硬件连接

我们依次将 ACM2108 模块、USB 线、下载器、电源线依次连接在开发板上，并且还需要给 ACM2108 连接信号源，这里我们连接的是 ACM2108 的通道 0（ADC0），信号源给的是 100Khz，vpp=5V 的正弦波，整体的硬件连接图如下图 1-9 所示。

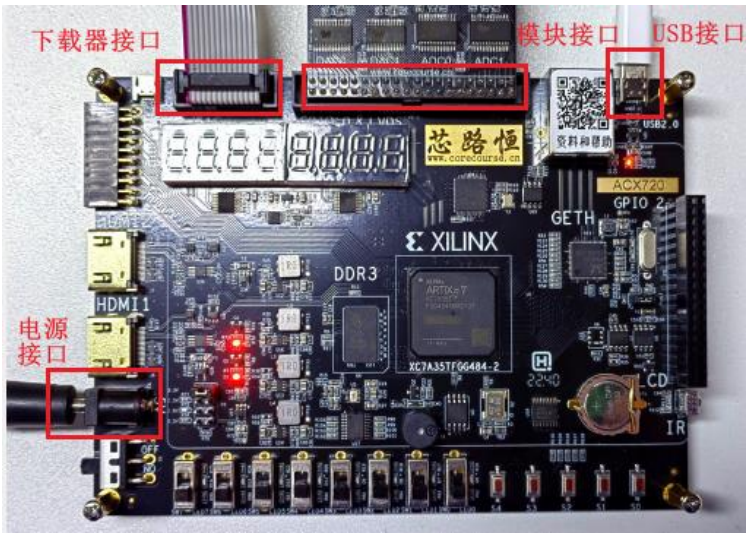


图 1-9 ACM2108 模块硬件连接图

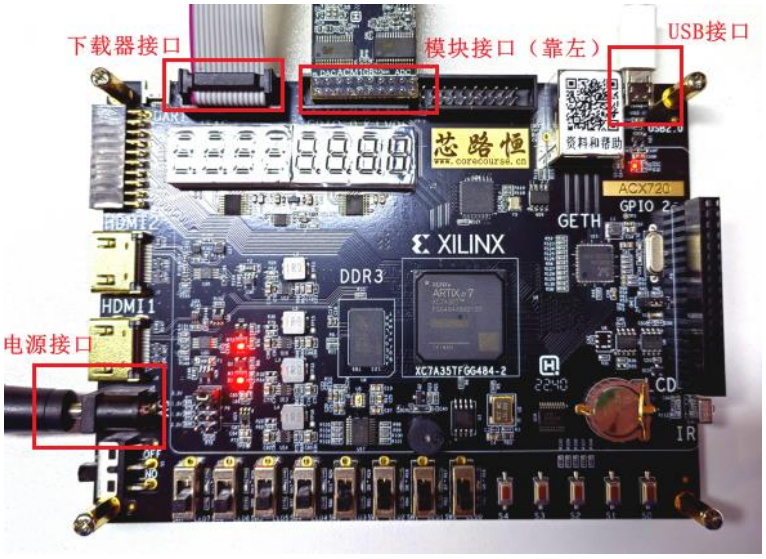


图 1-10 ACM108 模块硬件连接图

1.5.2 下载 bit 文件

将生成的 bit 文件下载至开发板上，进行验证，下载方式如下图 1-11 所示：

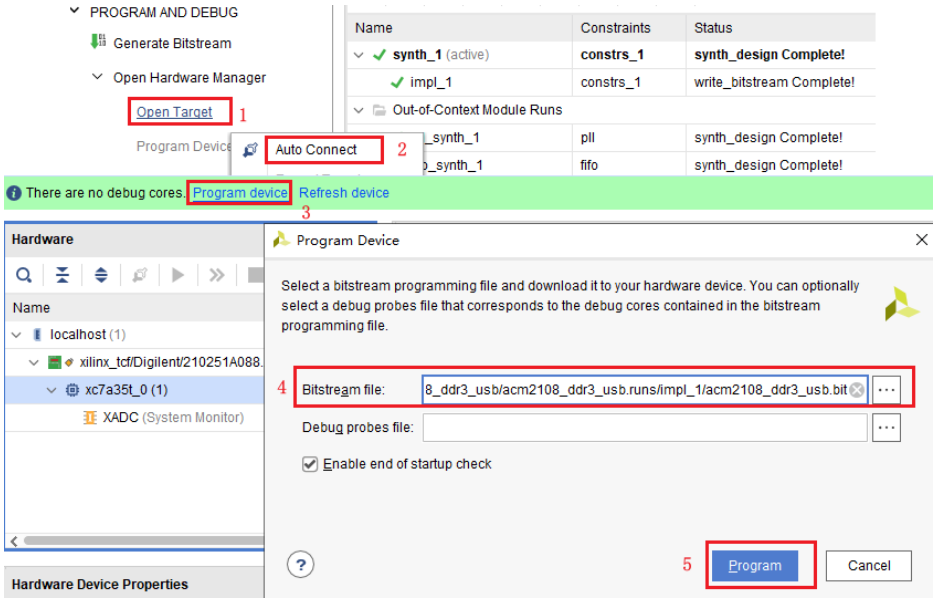


图 1-11 下载 bit 文件

bit 文件下载成功之后，可以看到开发板上的 LED0 和 LED1 会被点亮，LED0 灯亮锁相环工作正常，LED1 灯亮 ddr3 初始化正常，如下图 1-12 所示。

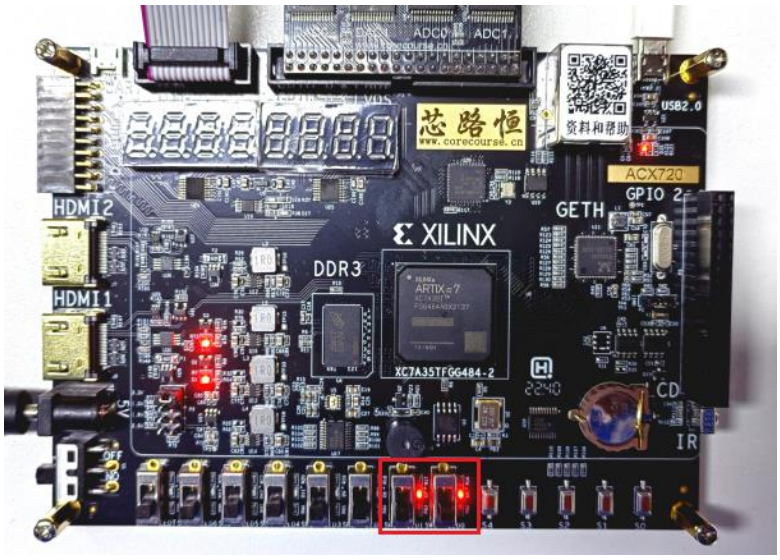


图 1-12 bit 文件下载成功示意图

1.5.3 安装 USB 驱动

安装 USB 驱动方式请参看 720 资料：盘 A\01_教材文档\02 文件中的 USB 章节的内容，驱动安装成功之后，用户打开设备管理，可以在“通用串行总线控

制器”中看到“Cypress FX3 USB StreamerExample Device”，之所以识别为 FX3，是因为 FX2 和 FX3 的这个例子使用了同一个 PID 和 VID，所以被识别为了 FX3，但是这并不影响我们的正常使用。如下图 1-13 所示：



图 1-13 USB 驱动安装成功示意图

1.5.4 cypress 上位机数据通信

本节使用 cypress 上位机发送命令帧，并将接收的数据进行存储。读者首先在我们提供的资料包中找到 cypress 软件（FX2_USB 数据采集上位机工具\CyControl 文件夹），然后双击打开软件。

打开软件之后，如果 USB 连接正常并且驱动安装成功，我们可以看到列表框中有“Cypress FX3 USB StreamerExample Device”，如下图 1-14 所示。

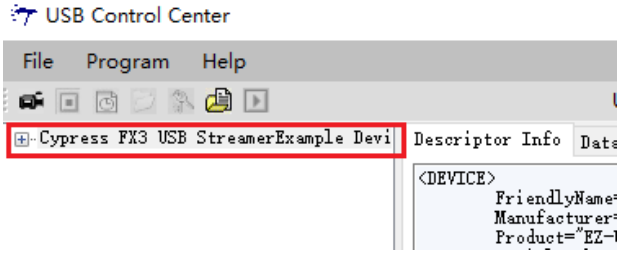


图 1-14 cypress 软件识别到 FX3

识别成功之后，我们使用该软件进行数据通信，软件的使用方式如下所示。

1. 点击“Cypress FX3 USB StreamerExample Device”前面的“+”找到 bulk out endpoint（0x02）。
2. 点击“Data Transfers”。
3. 在 Data to Send 中输入指令串。在前面接收转命令模块中介绍到数据帧

格式对 ACM2108 的四个寄存器进行配置。例如 ACM2108 以 50M 的采样速率，对 1 个通道进行采样（本次实验以通道 0 为例），共采集 4K 个数据，此时 Cypress 软件需要发送的指令串如下：

```
55A50200001000F055A50100000001F055A50300000000F055A50000000000F0
```

4. 点击“Transfer Data-OUT”进行命令传输，传输完成之后如下图 1-15 所示。

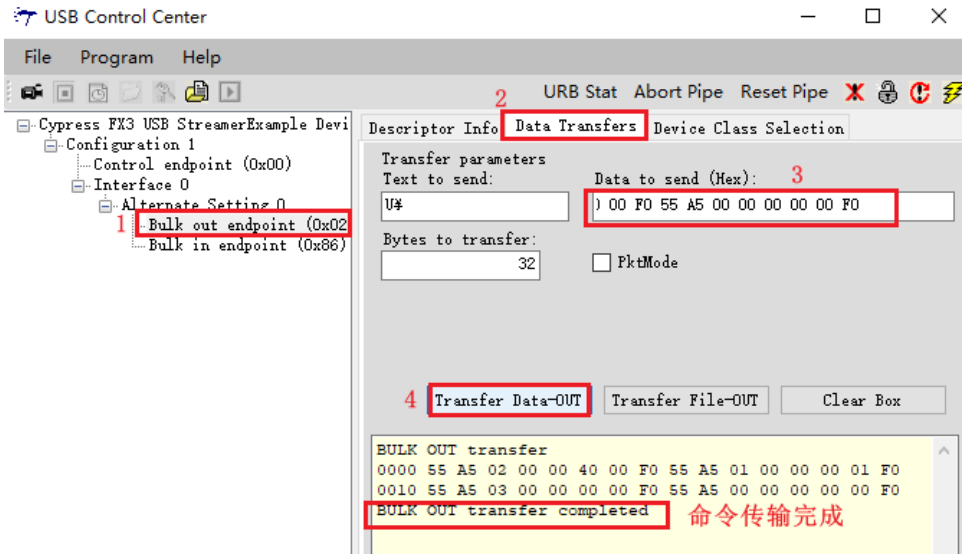


图 1-15 命令传输完成标志

5. 点击“bulk in endpoint”。

6. 设置采样数量值，在“Bytes to transfer”一栏中设置需要的数据个数，ACM2108 采集的数据是 16 位的，而 cypress 软件是以字节为单位传输的，那么这里设置的数值应该是采样数量*2（4096*2=8192）。

7. 点击“Transfer File-IN”将采集到的数据以文件的形式保存，操作如下图 1-16 所示。

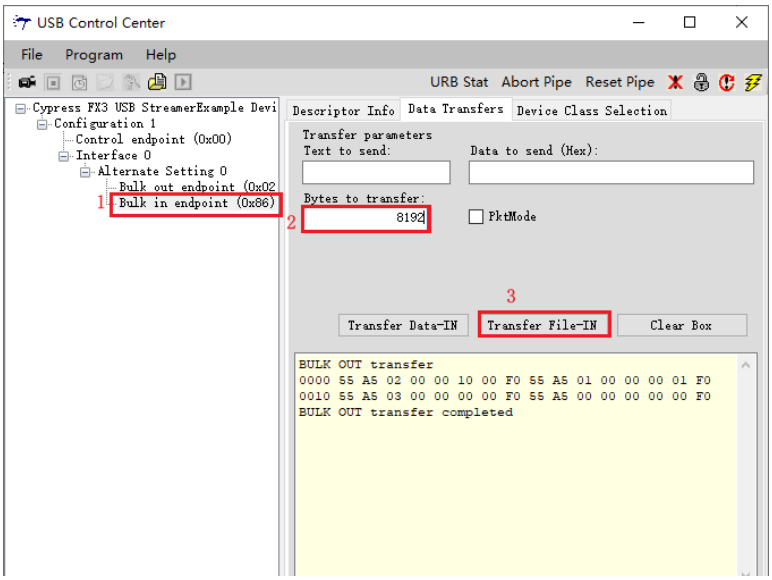


图 1-16 数据接收界面设置

8. 在弹出的文件保存界面中，读者需要设置文件保存的路径并给文件命名，比如我们这里给文件命名为“acm2108_16384_usb”，然后点击保存。如下图 1-17 所示。

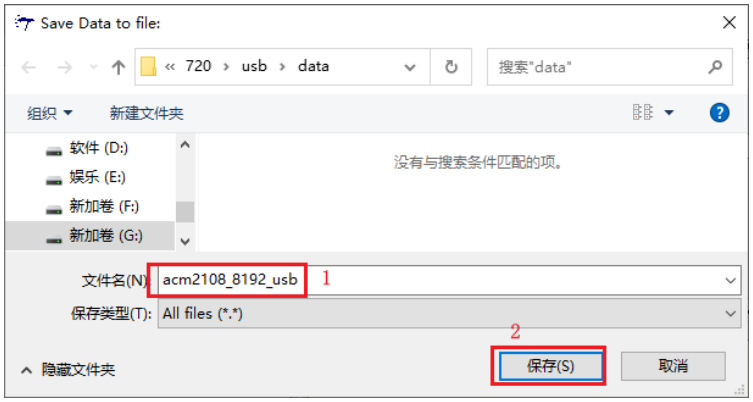


图 1-17 设置文件名称

9. Cypress 软件接收数据，并提示传输成功。如下图 1-18 所示。

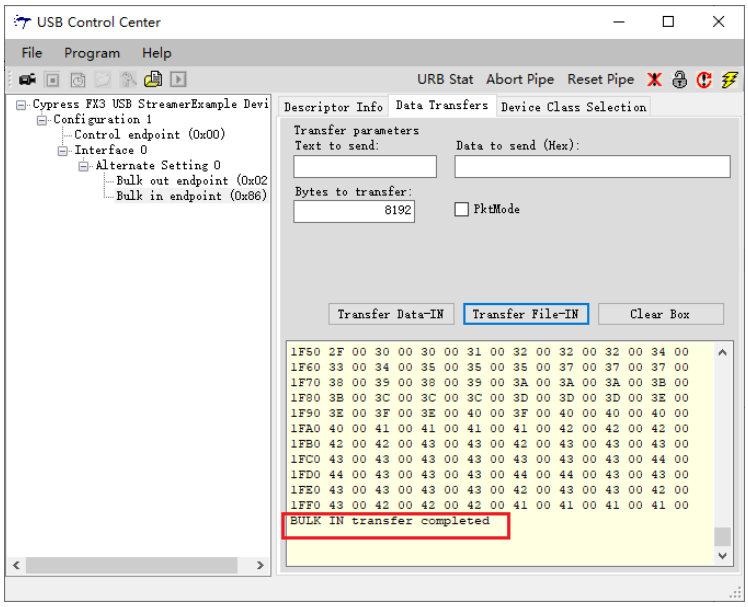


图 1-18 数据传输完成界面显示

10. 我们根据前面第 8 步设置的文件路径，找到数据文件，然后右击选中属性，查看文件大小，如下图 1-19 所示。从图中可以看出，文件大小为 32768 字节，符合我们之前设置的采样数量的大小。



图 1-19 查看接收文件大小

通过上述步骤，我们就完成了一次数据采集，并将采集到的数据进行了保存，方便后续进行数据分析。在操作的过程中还需要注意以下几点：

1. 采样数量必须为 256 个 16bit 数的整数倍，即接收的 bytes to transfer，一

定是 512 个 8bit 数据的整数倍。如果操作失误要求读数据的数量大于指令码设置的写入数据故障，会出现输出框为 997 的故障码。这时候建议将开发板复位，清除上次不正常读写的数据。开发板复位，按下 S4 按钮即可，fx2 芯片复位，在背面按下 USB-RST 按钮。

2. 如果是逐条发送指令，则必须确保启动指令在设置通道和设置采样数量指令之后发送，否则一旦发送采样启动指令，通道和采样数量设置将不会生效。
3. ACX720 开发板上的 DDR3 的存储容量为 128M，所以设置的最大采样数量不能超过 128M，即参数设置为 55 A5 02 08 00 00 00 F0。此时 cypress 软件最大读出量 stream in 为 268435456 的 8bit 数。（最大读出量 stream in 读出时，cypress 软件必须勾选 PktMode，且读出时间较长，不稳定。建议分批次读出，每次 stream in 读出小于等于 16777216 个数）。

1.5.5 MATLAB 图像绘制

前面通过 Cypress 软件得到了 ADC 采集到的数据文件，我们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，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 1-20 所示，得到的波形图如下图 1-21 所示。

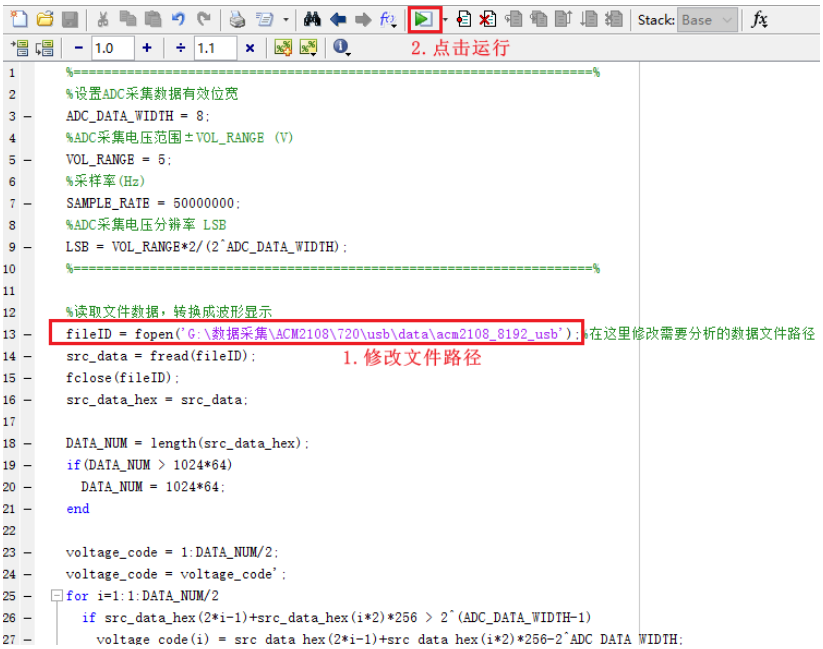


图 1-20 修改文件路径并运行

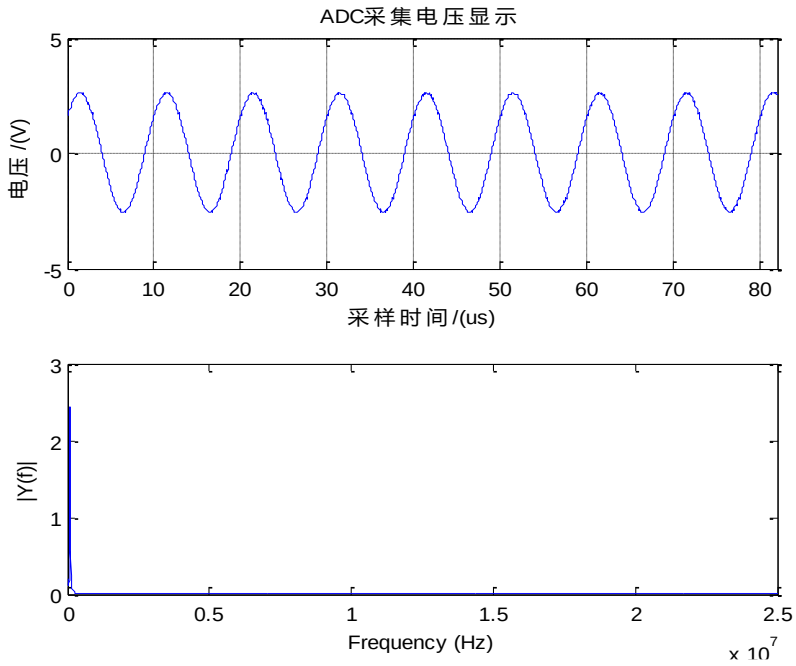


图 1-21 MATLAB 分析波形图

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（正负 2.5V），与 MATLAB 分析出来的波形一致，说明我们本次实验成功。

1.5.6 数据采集上位机通信

前面通过 Cypress 软件采集数据时，每次保存数据都需要重新点击“Transfer File-IN”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，读者在盘 A ACX720 开发板标准配套资料\05_常用软件中可以找到该软件。双击上位机软件，初始界面如下图 1-22 所示。

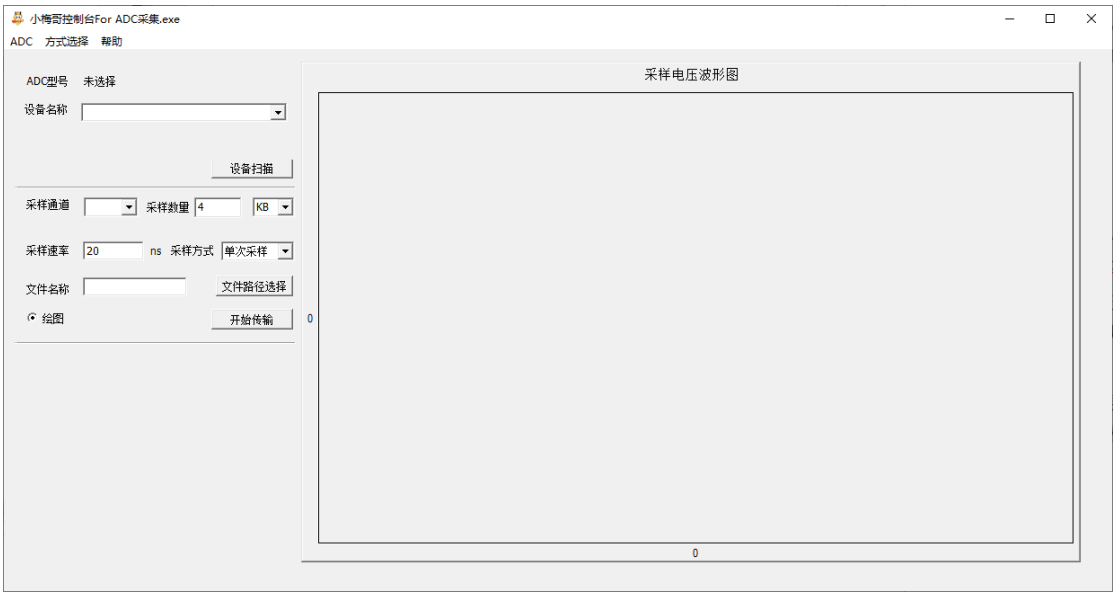


图 1-22 上位机软件初始界面显示

本次实验使用该软件的方式如下所示：

1. 点击 ADC，选择 ACM2108。
2. 点击方式，选择 USB，将会检测 USB 设备，设备检测成功之后，设备名称将会显示“(0x04B4 - 0x00F1) Cypress FX3 USB StreamerExample”。
3. 选择完成之后，可以看到对采样通道、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 1-23 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。

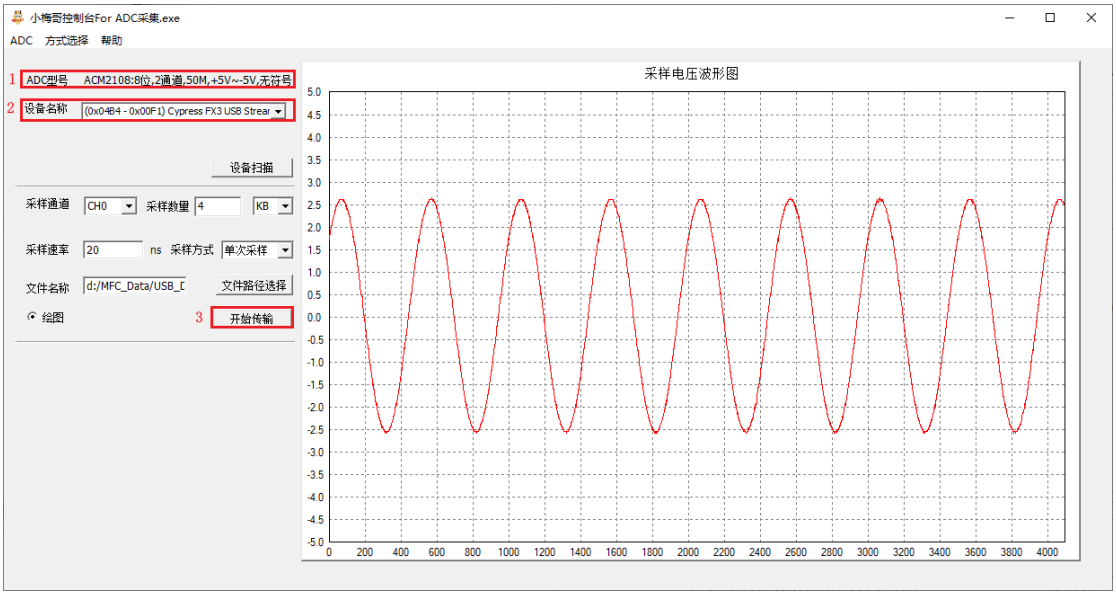


图 1-23 数据采集上位机显示图

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 1-24 所示。从图中可以看出，采集到的数据的频率为 100Khz，电压在正负 2.5V 左右，与我们输入的信号一致。

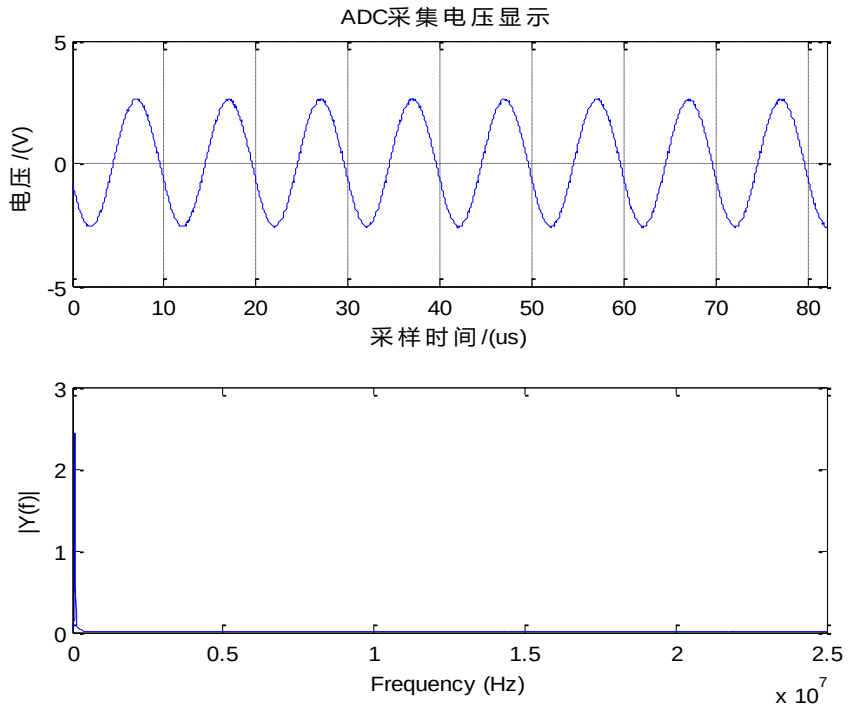


图 1-24 MATLAB 进一步波形分析图

1.5.7 ACM2108 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 S0 可以修改 DAC 的信号输出，ACM2108 模块连接方式如下图 1-25 所示，ACM108 模块的连接方式如下图 1-26 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。

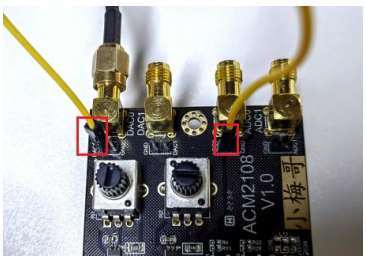


图 1-25 ACM2108 模块 ADC 和 DAC 连接示意图

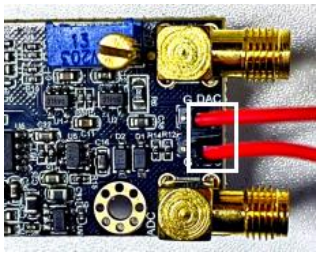


图 1-26ACM108 模块 ADC 和 DAC 连接示意图

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 1-27 所示为 DAC 产生的 125Khz 的正弦波信号。

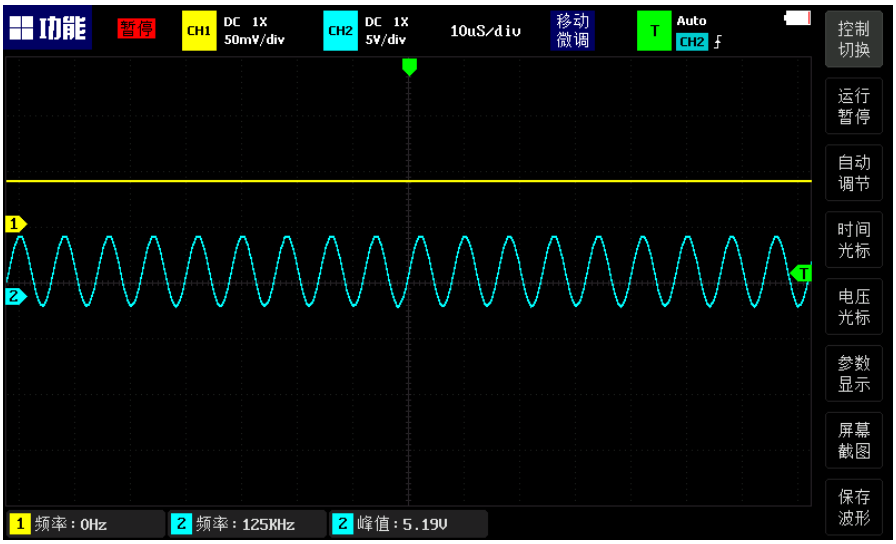


图 1-27 DAC 输出的信号

然后通过上位机采集数据，如下图 1-28 所示。

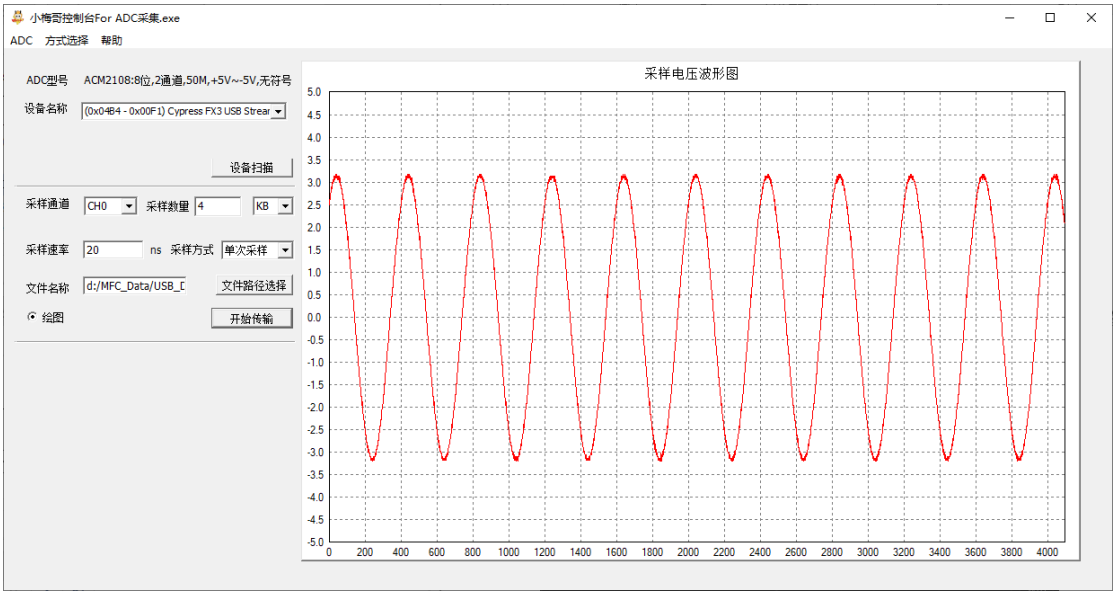


图 1-28 上位机采集到的波形图像数据

MATLAB 分析数据得到波形图如下图 1-29 **错误!未找到引用源。**所示。

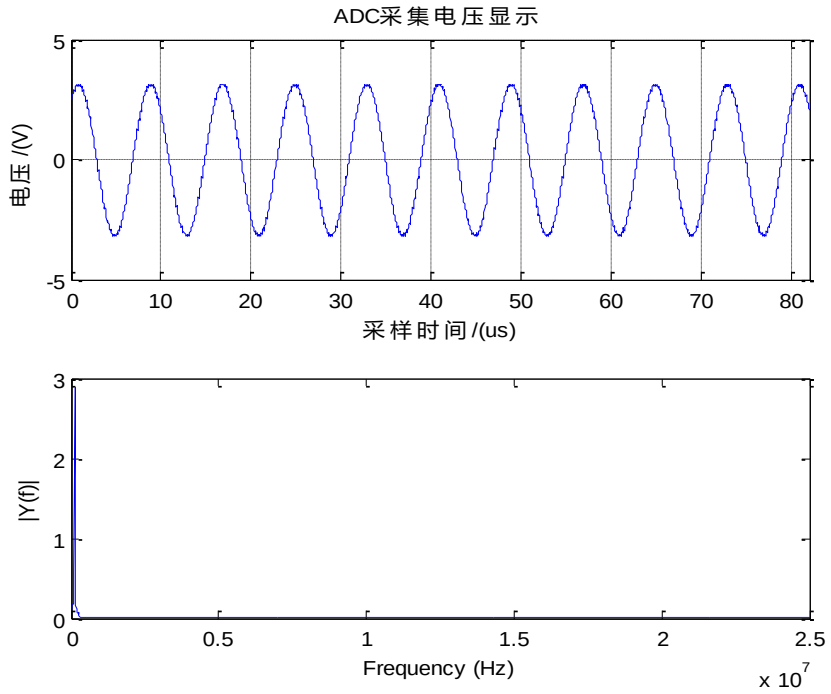


图 1-29 MATLAB 波形分析图

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参看 1.4.4 一节中的表 1-7 中的内容。

1.6 思考与总结

本次实验介绍了基于 ACM2108 的 USB 收发，用户通过 FX2_USB 调试工具 CyControl 向开发板发送指令数据配置 ACM2108 的四个寄存器，以此控制 ADC 进行采样。ADC 采样完成之后，将采集到的数据存储进 DDR3 中，再由 USB 将 DDR3 中的数据传输至电脑，电脑端通过 CyControl 软件将 USB 传输过来的数据以文件的形式保存，最终通过 MATLAB 对数据进行进一步的分析。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。