1 USB3.0 传输 ACM1030 双通道数据采集系统

章节导读

本节实验将介绍基于 ACM1030 模块利用 USB3.0 进行数据采集的相关内容。AC6103 开发板上的 USB3.0 接收电脑端发送过来的命令帧，然后 FPGA 从 USB 下发的数据中解析出命令。从而实现对 ACM1030 模块的采样频率、数据采样个数以及采样通道的合理配置。配置完成之后，ACM1030 开始采集数据，并将采集的数据存储至 DDR2 中，再由 USB3.0 将 DDR2 中的数据传输至电脑，用户可以在电脑上通过 USB 调试共苦 CyControl 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，如果进行双通道波形的显示的话，需要使用我们提供的上位机软件，Matlab 文件只针对单个通道进行绘图。

1.1 系统整体设计

通过电脑上的 USB 调试工具 CyControl 或者我们的数据采集上位机，将命令帧进行发送，然后通过 AC6103 开发板上的 CYUSB3014 芯片接收数据，随后从 USB 下发的数据中解析出命令，从而实现对 ACM1030 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，ACM1030 开始采集数据，并将 ACM1030 采集的数据存储进 DDR2 中。最后将 DDR2 中存储的数据通过 USB 传输至电脑，然后通过数据采集上位机显示采集到的波形，或者通过 Matlab 绘制，系统的整体框架如下所示。

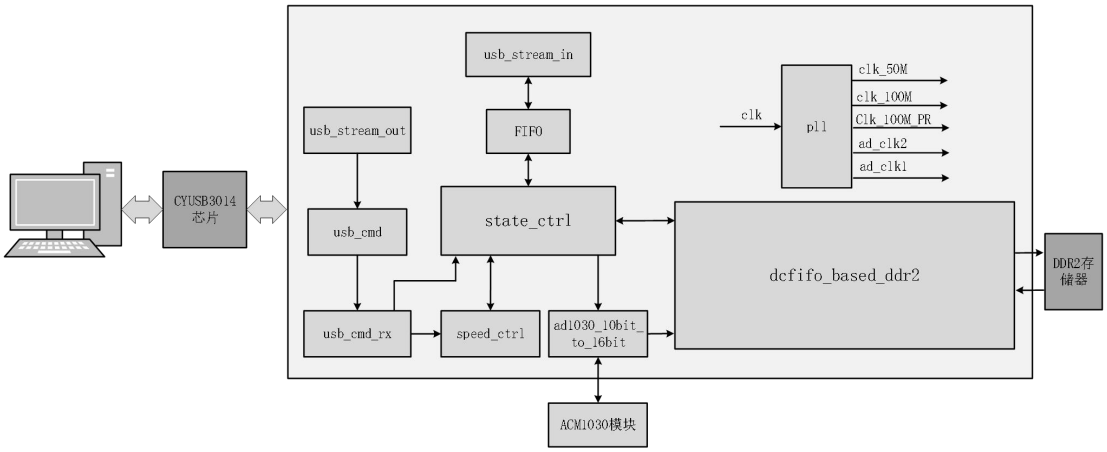

图 1-1USB3.0 传输 ACM1030 数据采集整体设计框图

下面对上述图中的各个模块的功能进行简要说明：

1. pll 模块：锁相环模块，输入时钟 50M，由开发板上的晶振提供；C0 输出 50M 时钟作为 DDR2 控制器的参考时钟，C1 输出 100M 的时钟给到 USB 模块和数据采集控制模块使用，C3 和 C4 输出两个 50M 的时钟给到 ADC 模块使用。
2. usb_stream_out 模块：USB 数据流发送控制模块，不断的将端点 1 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 ACM1030 的采样速率。
6. ad1030_10bit_to_16bit 模块：将 ACM1030 采样到的 10 位数据转换成 16 位的有符号数据。
7. state_ctrl 模块：ADC 采集数据 DDR2 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。
8. dcfifo_based_ddr2 模块：DDR2 双端口模块，用来缓存 ACM1030 采集到的数据。
9. fifo 模块：USB 发送 FIFO，从 DDR2 中读取数据进入该 FIFO 中，然后从该模块中读出交由 usb_stream_in 模块发出，主要是解决位宽和跨时钟域的问题转换的问题。
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

除去使用 IP 和前面章节讲过的内容外，本章主要讲解 usb_cmd 模块、usb_cmd_rx 模块、state_ctrl 模块、ad1030_10bit_to_16bit 模块和 speed_ctrl 模块。

1.2 ACM1030 模块简介

ACM1030 模块是基于国产知名模拟器件设计和制造商思瑞浦（3PEAK）公司的 10 位 50M 采样速率高速 ADC 芯片 3PA1030。ACM1030 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。ACM1030 模块图如下图 1-2 所示。

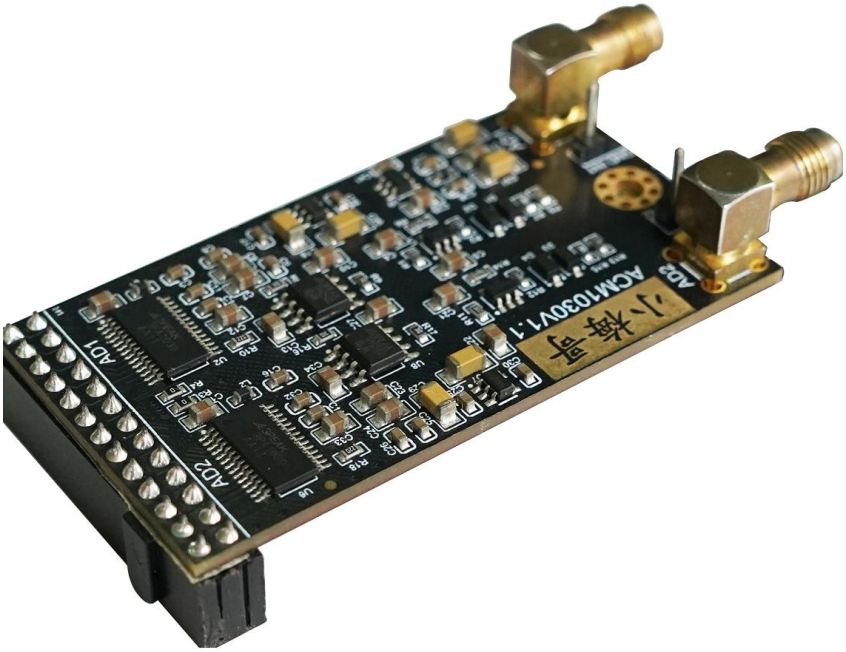

图 1-2 ACM1030 模块图

ACM1030 模块与 FPGA 连接采用并行接口，每路 ADC 包括 10 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_OVR），连接口如下图 1-3 所示。

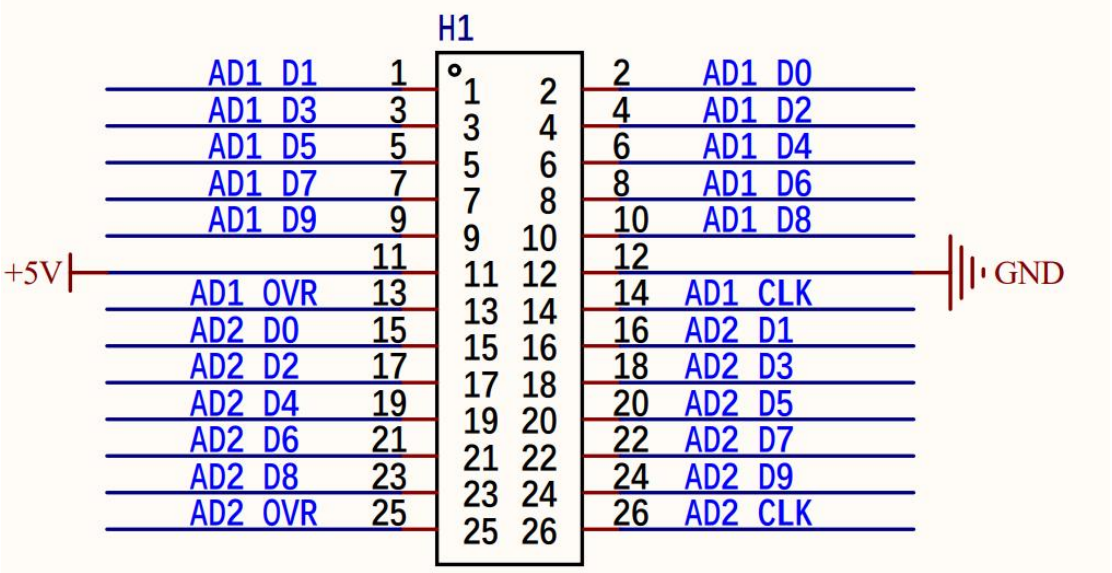

图 1-3 ACM1030 模块接口原理图

本模块在使用时，仅需 FPGA 为每一路 ADC 提供一路时钟信号，ADC 则

会每个时钟周期输出一个 10 位的采样结果，数据结果为无符号格式。

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、AC820、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、ACG525 开发板、AC6103 开发板。

1.3 模块设计

下面将对本次实验需要设计的模块进行介绍

1.3.1 usb_cmd 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所示。

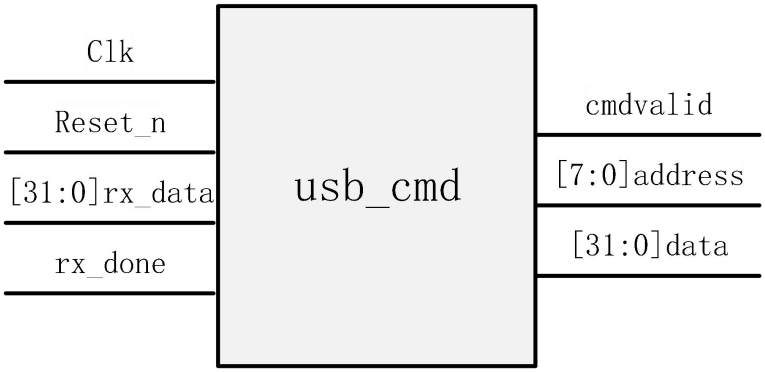

图 1-4 接收转命令模块框图

模块信号说明如下所示。

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 [31:0]	I	USB 接收数据流模块接收到的 32 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 ACM1030 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 ACM1030 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 2 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[1] <= #1 rx_data;
    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[1][31:24] == 8'hF0))begin
        data[7:0] <= #1 data_str[1][23:16];
        data[15:8] <= #1 data_str[1][15:8];
        data[23:16] <= #1 data_str[1][7:0];
        data[31:24] <= #1 data_str[0][31:24];
        address <= #1 data_str[0][23:16];
        cmdvalid <= #1 1;
    end
    else
        cmdvalid <= #1 0;
end
```

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

1.3.2 usb_cmd_rx 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。ACM1030 模块提供了 ADC1、ADC2 两个通道进行数据采集。
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所示。

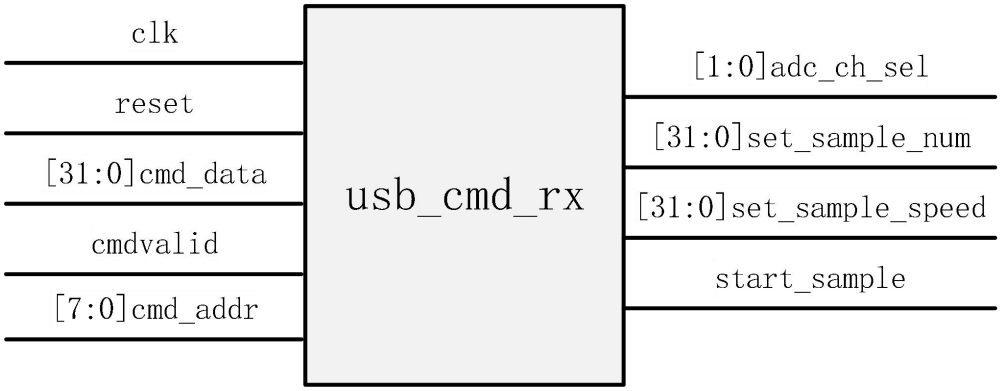

图 1-5 指令转控制模块结构框图

模块信号说明如下所示。

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;
```

1.3.3 speed_ctrl 模块

采样速率控制（speed_ctrl）模块用来控制 ACM1030 的采样速率，该模块的结构框图如下所示。

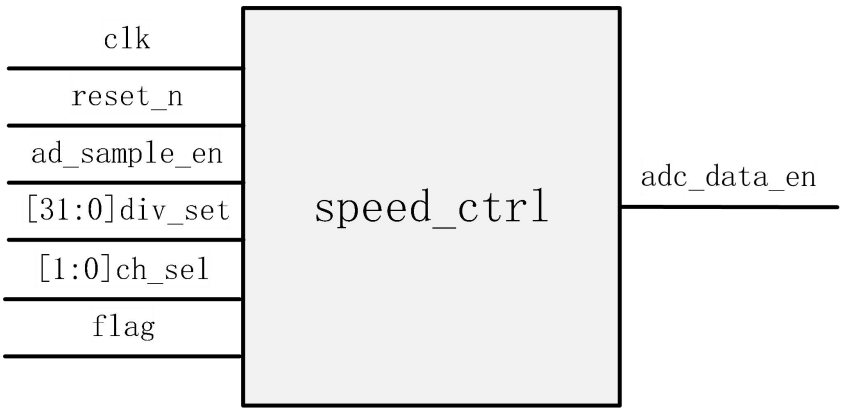

图 1-6 采样速率控制模块

对该模块的信号说明如下所示：

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
flag	I	flag 标志信号，当 flag 等于 0 的时候，采集的是通道 1，当 flag 等于 1 的时候，采集通道 2 的数据，该信号由 ad1030_10bit_to_16bit 模块输出
ch_sel	I	需要采样的通道
adc_data_en	O	ADC 采样结果存储使能信号

本次实验，ACM1030 的时钟给的是 50M 的时钟，其最大采样速率就是 50M，如果使用低于时钟频率的采样速率，可以依旧给 ADC 提供 50M 的时钟信号，但在 FPGA 内部，对 50M 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。但是本模块的时钟信号我们给的是 100M 时钟，我们需要根据 flag 信号进行判断，具体为什么使用 100M 时钟，以及 flag 的信号的产生将在讲解 ad1030_10bit_to_16bit 模块的时候进行说明。

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((ch_sel == 2'b01) || (ch_sel == 2'b11)) begin
        if(flag) begin
```

```
            if(div_cnt >= div_set)
                div_cnt <= 0;
            else
                div_cnt <= div_cnt + 1'd1;
        end
        else
            div_cnt <= div_cnt;
    end
    else if(ch_sel == 2'b10) begin
        if(~flag) begin
            if(div_cnt >= div_set)
                div_cnt <= 0;
            else
                div_cnt <= div_cnt + 1'd1;
        end
        else
            div_cnt <= div_cnt;
    end
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;
```

1.3.4 ad1030_10bit_to_16bit 模块

数据采集模块 ACM1030 采集到的 10bit 数据不便于计算机存储，因为计算机对数据进行分析、存储的时候都是以 8 位或者 16 位的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（ad1030_10bit_to_16bit）将 10bit 的数据转换为 16bit 数据进行存储。该模块的结构框图如下所示：

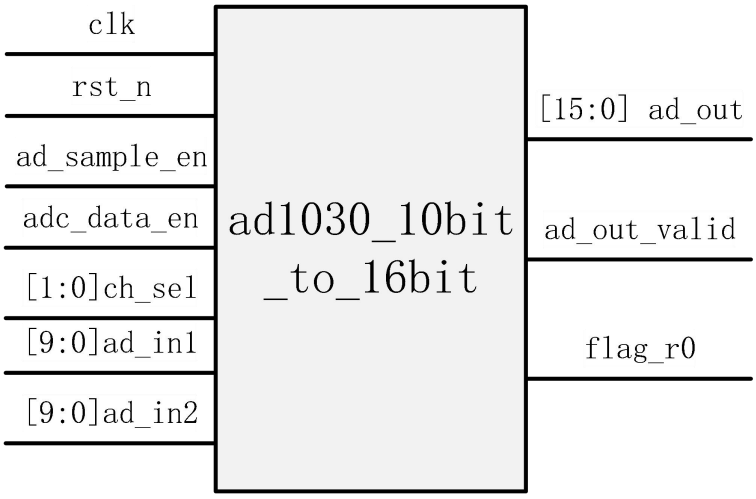

图 1-7 数据位扩展模块

首先，我们需要知道的是，AD1030 的工作时钟是 50M，当进行双通道数据采集的时候，我们还需要判断什么时候取通道 1 或者通道 2 的数据，这里我们考虑将数据位扩展模块的工作时钟提高到 100M，然后根据 100M 的时钟，在其上升沿的时候进行取反，得到 flag 信号，然后根据 flag 信号取数据，下面我们用一个时序图进行说明。

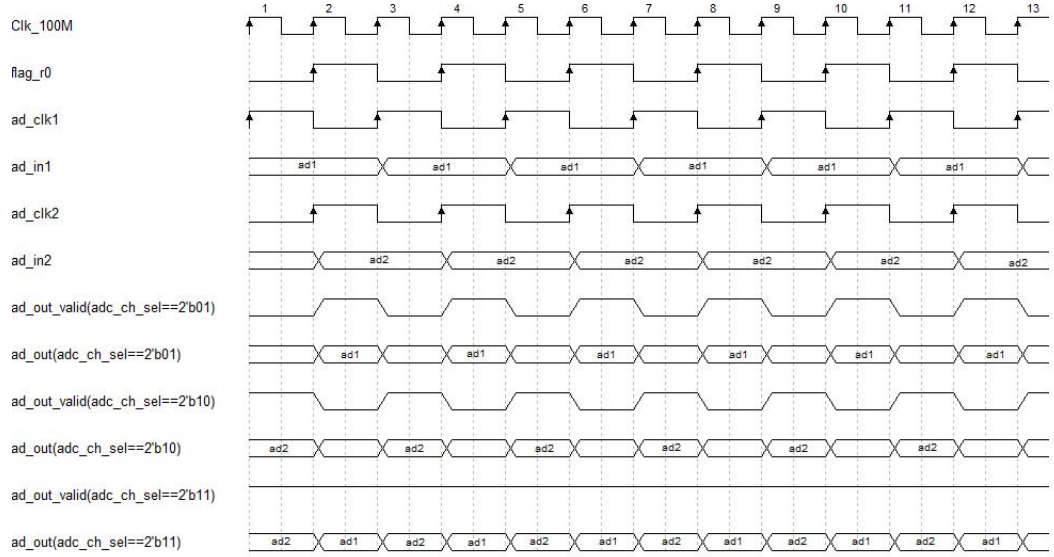

图 1-8 数据位扩展模块输出数据时序说明图（50M 采样率）

从上图可以看出，flag_r0 信号就是对 100M 进行的一个分频，也就是 50M 的时钟，这就和 ADC 的工作时钟一致，当 flag_r0 为高电平的时候，取通道 1 的数据，当 flag_r0 为低电平的时候，取通道 2 的数据，这样就能同时采集两个通道的数据，这里需要注意的是，可以看到 ADC 模块两个通道的工作时钟相差了 180°，这样是为了保证在时钟上升沿取数据的时候，ADC 模块输出的数据

是稳定的，因为如果两个通道的工作时钟完全一样，那么在取通道 2 数据的时候，就刚好会在通道 2 数据变化的时候取，这样采集到的数据是不稳定的，并且 flag_r0 的复位信号需要是 PLL 模块的 locked 信号，这样才能保证，取数据时是时稳定的。

根据上述的时序图，我们便可以根据需要编写对应的代码。

首先是，flag_r0 信号的产生，如下所示，根据每个时钟上升沿进行反向。

```
always@(posedge clk or negedge rst_n)
if(~rst_n)
    flag_r0 <= 0;
else
   flag_r0 <= ~flag_r0;
```

然后就是将 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是 1023~0 的正弦波，但是上位机在分析数据进行绘图的时候需要数据是有符号的，这里我们进行的操作就是将 ADC 采集的数据加上 512，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 512 之后得到的二进制分别为 1000000000（-0）、1111111111（-511）、0111111111（+511），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，并且根据 flag_r0 的信号的不同，输出对应通道的数据，代码如下所示：

```
wire [9:0] sd_ad_in1;
wire [9:0] sd_ad_in2;
wire [9:0] sd_ad_in3;
assign sd_ad_in1 = ad_in1 + 512;
assign sd_ad_in2 = ad_in2 + 512;
assign sd_ad_in3 = flag_r0  ?sd_ad_in1:sd_ad_in2;
```

然后根据选择通道的不同，依次输出对应通道的数据，当 ch_sel == 2'b00 的时候，输出测试通道的数据，也就是我们通过计数器累加的值，当 ch_sel == 2'b01 的时候，输出通道 1 的数据 sd_ad_in1，当 ch_sel == 2'b10 的时候，输出通道 2 的数据 sd_ad_in2，当 ch_sel == 2'b11 的时候，输出通道 1 和 2 的数据 sd_ad_in3，ADC 采集的 10 位的数据，我们通过补 0 的方式，得到最终输出的 16 位的数据，代码如下所示，代码中的 ad_sample_en 是由外部输入的 ADC 采集状态标志信号，当该信号有效的时候，表明我们可以存储 ADC 采集的数据：

```
always @(posedge clk or negedge rst_n)
if(~rst_n)
    ad_out <= 16'd0;
else if(ad_sample_en && ch_sel == 2'b01)
```

```
    ad_out<={4'd0,sd_ad_in1,2’d0};
else if(ad_sample_en && ch_sel == 2'b10)
    ad_out<={4'd0,sd_ad_in2,2’d0};
else if(ad_sample_en && ch_sel == 2'b11)
    ad_out<={4'd0,sd_ad_in3,2’d0};
else if(ad_sample_en && ch_sel == 2'b00)
    ad_out<={4'd0,adc_test_data,2’d0};
else
    ad_out <= 'd0;
```

最后是输出数据有效信号，当选择通道 1 的时候，应该在 flag_r0 为高，ad_sample_en 为高并且由 speed_ctrl 输出的 adc_data_en 有效时输出，选择通道 2，则应该 flag_r0 为低，两个通道则不用考虑 flag_r0 信号，代码如下所示：

```
always @(posedge clk or negedge rst_n)
if(~rst_n)
    ad_out_valid <= 1'd0;
else if(ch_sel == 2'b01) begin
    ad_out_valid <= ad_sample_en & flag_r0 & adc_data_en;
end
else if(ch_sel == 2'b10) begin
    ad_out_valid <= ad_sample_en & (~flag_r0) & adc_data_en;
end
else
    ad_out_valid <= ad_sample_en & adc_data_en;
```

1.3.5 state_ctrl 模块

状态控制模块 state_ctrl 的作用就是控制 ADC 开始采集、USB 端点 1 的方向以及向 DDR 读写数据等相关状态的控制，该模块的基本结构框图如下所示。

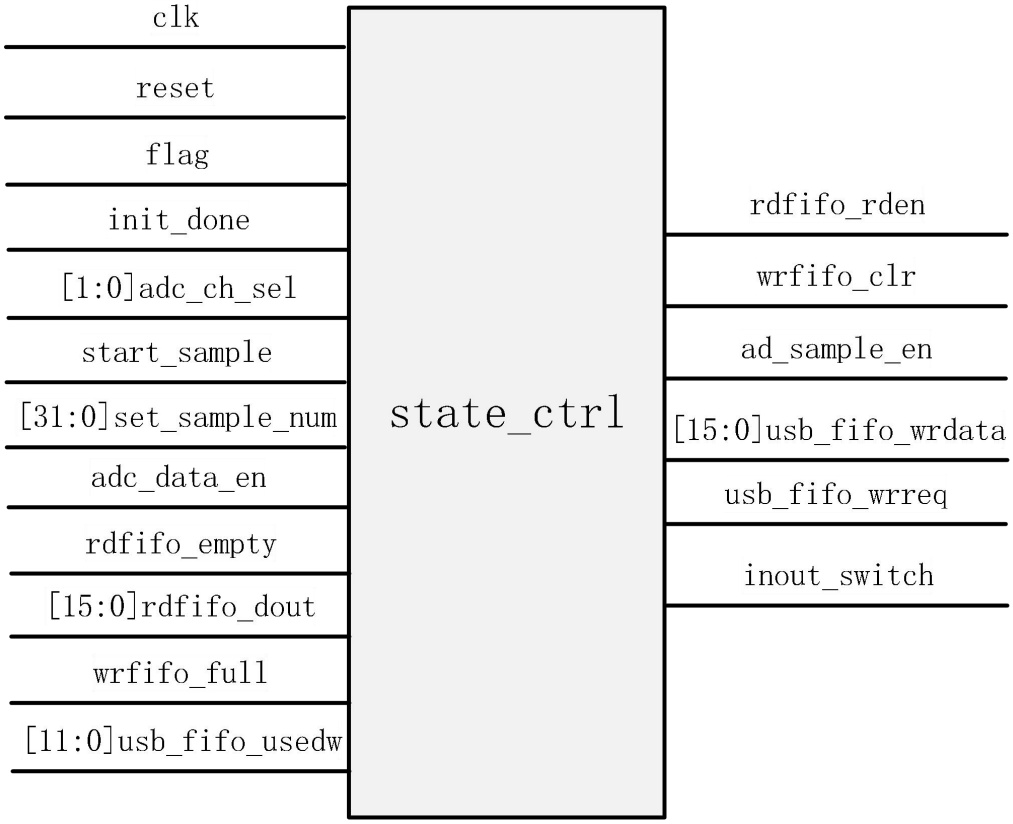

图 1-9 state_ctrl 模块基本结构框图

对该模块的信号说明如下所示。

表 1-6 state_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
flag	I	通道数据输出标志信号，为 1 表示写入通道 1 的数据，为 0 表示写入通道 2 的数据
init_done	I	DDR 初始化完成标志信号
adc_ch_sel[1:0]	I	通道命令信号
start_sample	I	ADC 模块开始采样标志信号
set_sample_num [31:0]	I	采样数量命令信号
rdfifo_empty	I	读 FIFO 为空标识信号，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 输出数据使能信号
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	DDR 控制写 FIFO 清除信号
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

该模块我们通过状态机的方式去实现，定义如下状态：

```
localparam IDLE                  = 4'd0;   //空闲状态
localparam DDR_WR_FIFO_CLEAR     = 4'd1;   //写 FIFO 清除状态
localparam ADC_SAMPLE            = 4'd3;   //ADC 采样数据状态
localparam DATA_SEND_WORKING     = 4'd4;   //数据发送状态
```

下面我们将对每个状态的作用和具体的实现代码进行说明。

1. IDLE 状态

当处于空闲状态并且 DDR 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 && 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 USB 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：

```
IDLE: //0
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，确保 FIFO 中的数据清除完毕，代码如下所示：

```
always@(posedge clk or posedge reset)begin
if (reset)
    wrfifo_clr<=0;
else if(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
```

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。还需要根据选择通道的不同，确保写入 DDR 中的第一个数据是通道 1 的，这样确保读出的时候，第一个数据是通道 1 的，确保上位机绘制的两个通道的数据对应的通道不出错，不然我们在绘制图像的时候，并不能判断哪个数据是通道 1 的，哪个数据是通道 2 的，所以当选择的通道 2 时， flag 为 0 的时候开始写入数据，进入 ADC 采集数据状态，当选择通道 1 或者两个通道选择时， flag 为 1 的时候，开始向 DDR 中写入数据，并且进入 ADC 采集数据状态，DDR 的写 FIFO 清除状态代码如下所示：

```
DDR_WR_FIFO_CLEAR: //1
begin
if(!wrfifo_full && (wrfifo_clr_cnt==9))
begin
    if(adc_ch_sel == 2'b10) begin
        if(~flag)
            state<=ADC_SAMPLE;
        else
            state<=state;
    end
    else begin
        if(flag)
            state<=ADC_SAMPLE;
        else
            state<=state;
    end
end
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，之所以提供的延迟的信号时间为 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。代码如下所示：

```
always@(posedge clk or posedge reset)begin
if (reset)
    wrfifo_clr<=0;
else if(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，根据选择通道以及采样率的不同进行计数，当选择通道 1 的时候，adc_data_en 信号为高并且 flag 信号为高的时候进行计数，当选择通道 2 的时候，adc_data_en 信号为高并且 flag 信号为低的时候进行计数，当选择双通道的时候，只需要 adc_data_en 信号为高时，便可以进行计数，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_ch_sel == 2'b01) // 通道 1
    begin
        if(adc_data_en & flag)
            adc_sample_cnt<=adc_sample_cnt+1'b1;
        else
            adc_sample_cnt<=adc_sample_cnt;
        end
    else if(adc_ch_sel == 2'b10)    //通道 2
    begin
        if(adc_data_en & ~flag)
            adc_sample_cnt<=adc_sample_cnt+1'b1;
```

```
        else
            adc_sample_cnt<=adc_sample_cnt;
    end
    else
    begin
        if(adc_data_en)
            adc_sample_cnt<=adc_sample_cnt+1'b1;
        else
            adc_sample_cnt<=adc_sample_cnt;
    end
end
else
    adc_sample_cnt<=1'b0;
end
```

当 adc_sample_cnt 达到设定的采样数据个数的时候，ADC 模块数据采集完成，跳转到数据发送状态，代码如下所示：

```
ADC_SAMPLE: //2
begin
if((adc_sample_cnt>=set_sample_num))
    state<= DATA_SEND_WORKING;
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. 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 发送中写入的数据小于 1020 的时候，开始从 DDR 中读取出数据写入到 FIFO 中，当然这个数据并不是固定的，可以修改，我们只是为了防止 FIFO 中被写满，代码如下所示：

```
DATA_SEND_WORKING: //6
begin
```

```
 //*//如果 send_data_cnt（USB 发送计数）等于给定的值,则跳转进入 IDLE 状态
    //如果整个数据块发送完成，则大循环收口
    if(send_data_cnt>=set_sample_num-1)begin
        state <= IDLE;
        rdfifo_rden <= 1'b0;
    end
    else if((usb_fifo_usedw < 1020)) 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
```

至此，本次实验需要设计的模块我们就讲解完成了，关于 DDR 控制模块以及 USB 的相关内容我们这里将不再进行说明了，需要用户自己去了解相关部分的内容，关于 DDR 控制模块我们在本次实验中就把当一个存储容量大的 FIFO 来使用，我们只需要通过其 FIFO 的接口，向里面和读取数据，该模块实现起来比较复杂，这里不进行说明，用户就算对其模块的实现不了解，也很容易使用，比如本次实验，我们需要使用的就是 wrfifo 和 rdfifo 的几个信号，wrfifo_clk 和 rdfifo_clk 的工作时钟都是 100M，wrfifo_wr 就是 ad_out_valid 信号，需要写入的数据就是 ad_out，关于读都是由 state_ctrl 控制的，dcfifo_based_ddr2 模块在顶层的调用如下所示：

```
dcfifo_based_ddr2 #(
    .WRFIFO_DW   ( 16    ),
    .RDFIFO_DW   ( 16 ),
    .DDR_DW      ( 16*2   ),
    .DDR_AW      ( 25             ),
    .DDR_SW      ( 7              ),
    .SHOWAHEAD_EN(1'b1            )
  )dcfifo_based_ddr2_inst
  (
    .rst_n          (rst_n & sys_rst_n   ),
    .ddr_ref_clk    (Clk_50M           ),
    .ddr_init_done  (ddr_init_done    ),
    //fifo wr
    .wrfifo_clk     (Clk_100M   ),
    .wrfifo_wrfull  (wrfifo_full    ),
    .wrfifo_wr      (ad_out_valid    ),
    .wrfifo_din     (ad_out   ),
    .wrfifo_clr      (wrfifo_clr),
    .wrusedw        ( ),
    //fifo rd
    .rdfifo_clk     (Clk_100M          ),
    .rdfifo_rdempty (rdfifo_empty ),
    .rdfifo_rd      (rdfifo_rden      ),
    .rdfifo_dout    (rdfifo_dout    ),
    .rdfifo_clr      (rdfifo_clr),
    .rdusedw        (read_side_fifo_rusedw1 ),
    //ddr interface
    .mem_odt        (ddr2_mem_odt          ),
    .mem_cs_n       (ddr2_mem_cs_n         ),
    .mem_cke        (ddr2_mem_cke          ),
    .mem_addr       (ddr2_mem_addr         ),
    .mem_ba         (ddr2_mem_ba           ),
    .mem_ras_n      (ddr2_mem_ras_n        ),
    .mem_cas_n      (ddr2_mem_cas_n        ),
```

```
    .mem_we_n       (ddr2_mem_we_n          ),
    .mem_dm         (ddr2_mem_dm            ),
    .mem_clk        (ddr2_mem_clk           ),
    .mem_clk_n      (ddr2_mem_clk_n         ),
    .mem_dq         (ddr2_mem_dq            ),
    .mem_dqs        (ddr2_mem_dqs           )
);
```

1.4 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 USB 调试工具 CyControl，将命令帧进行发送， 然后通过 AC6103 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 ACM1030 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，ACM1030 开始采集数据，将 ACM1030 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。

1.4.1 硬件连接

将 ACM1030 模块、USB3.0 连接线、下载器依次连接在开发板上，连接 ACM1030 模块的时候将模块的 1 脚与 GPIO1 的 1 脚对应，还需要给 ACM1030 的通道给信号源，ACM1030 模块通道 1 信号源给的是 200khz，vpp=5V 的正弦波，整体的硬件连接图如下图 1-13 所示。

图 1-10 硬件连接图

连接完成之后，下载生成的 sof 文件。

1.4.2 下载 USB 固件

打开我们提供的 CyControl 软件，识别到如下所示的硬件：

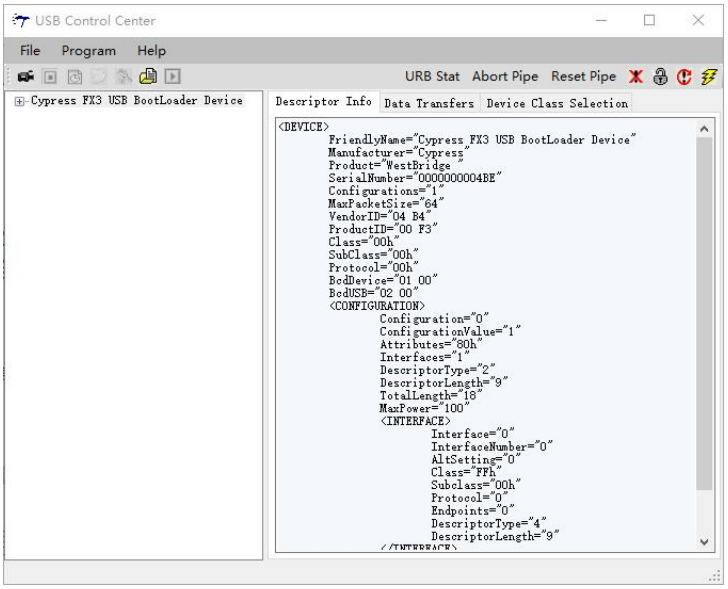

图 1-11 CyControl 识别到硬件

然后选中该硬件，依次点击 Program->FX3->RAM，如下所示。

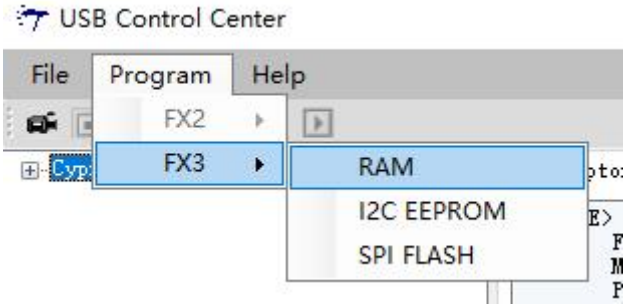

图 1-12 下载固件操作

点击之后，找到我们提供的固件 SlaveFifoSync.img，点击打开，即可下载，如下所示。

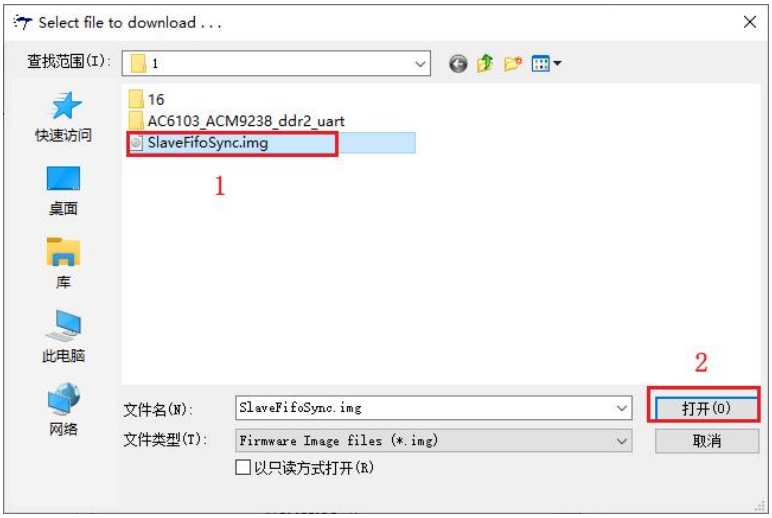

图 1-13 选择需要下载的固件

下载完成之后，软件会提示下载成功，并且设备名称，会改变，如下所示。

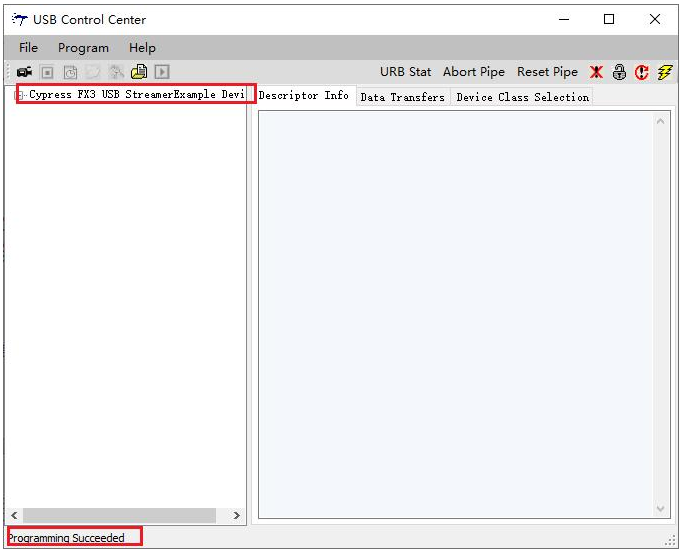

图 1-14 固件下载成功提示

注意，这种方式下载固件，每次断电之后，都需要重新下载，固化的方式请自行查看 AC6103 中 USB 固化部分的内容，我们这里就不再进行说明了。

1.4.3 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。打开软件之后，使用该软件进行数据传输的方式如下所示：

1. 点击“Cypress FX3 USB StreamerExample Device”前面的“+”找到 bulk out endpoint（0x01）。
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 ACM1030 的四个寄存器进行配置，例如 ACM1030 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，这里需要注意的是，使用 USB3.0 进行传输的时候，我们每次传输的数据必须大于 16384 字节，也就是 8192 个数据，不然会回读不了数据，此时 Cypress 软件需要发送的指令串如下：

```
55A50200004000F055A50100000001F055A50300000000F055A50000000000F0
```

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

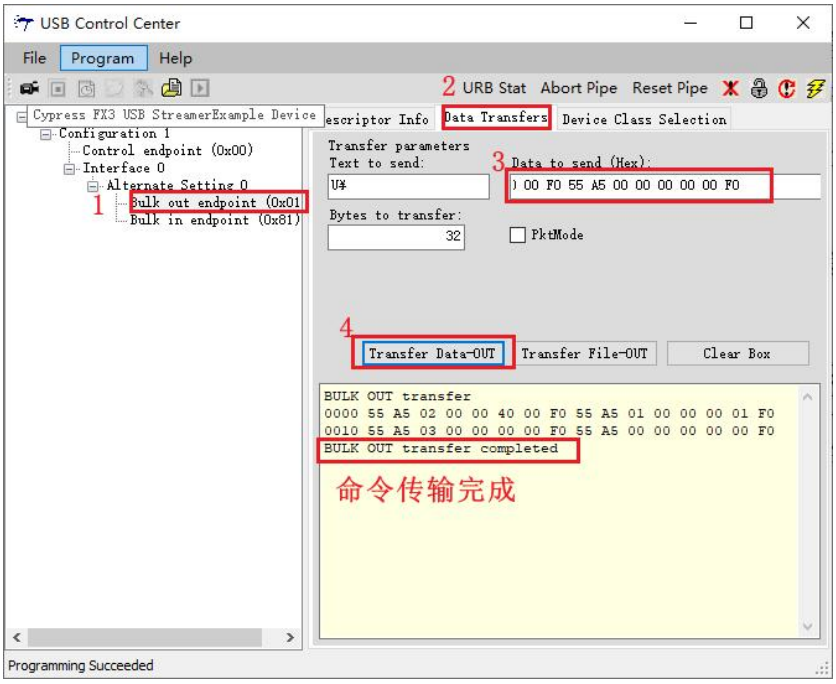

图 1-15 传输命令操作

5. 点击“bulk in endpoint（0x81）”。

6. 设置采样数量值，在“Bytes to transfer”一栏中设置需要的数据个数，ACM1030 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（16384*2=32768）。

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。

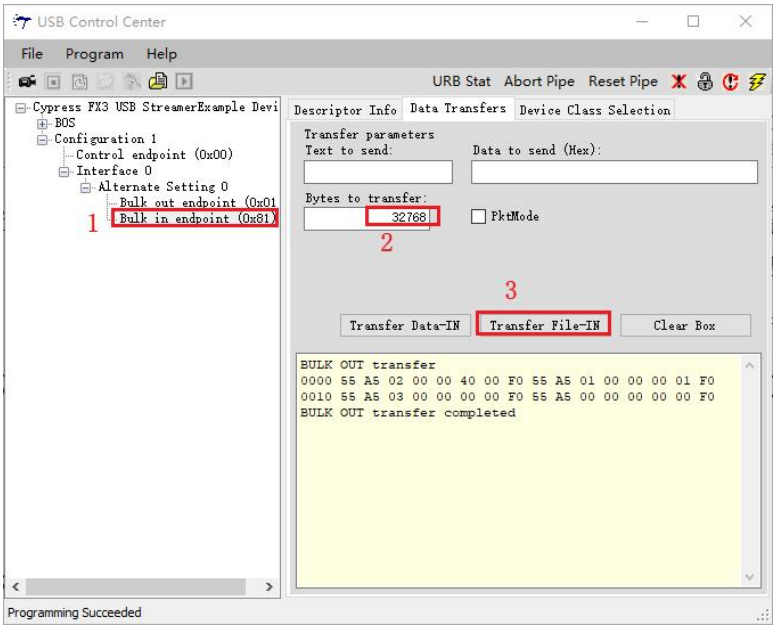

图 1-16 数据接收界面设置

8. 在弹出的文件保存界面中，我们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“ad1030_16384_usb”，然后点击保存。如下所示。

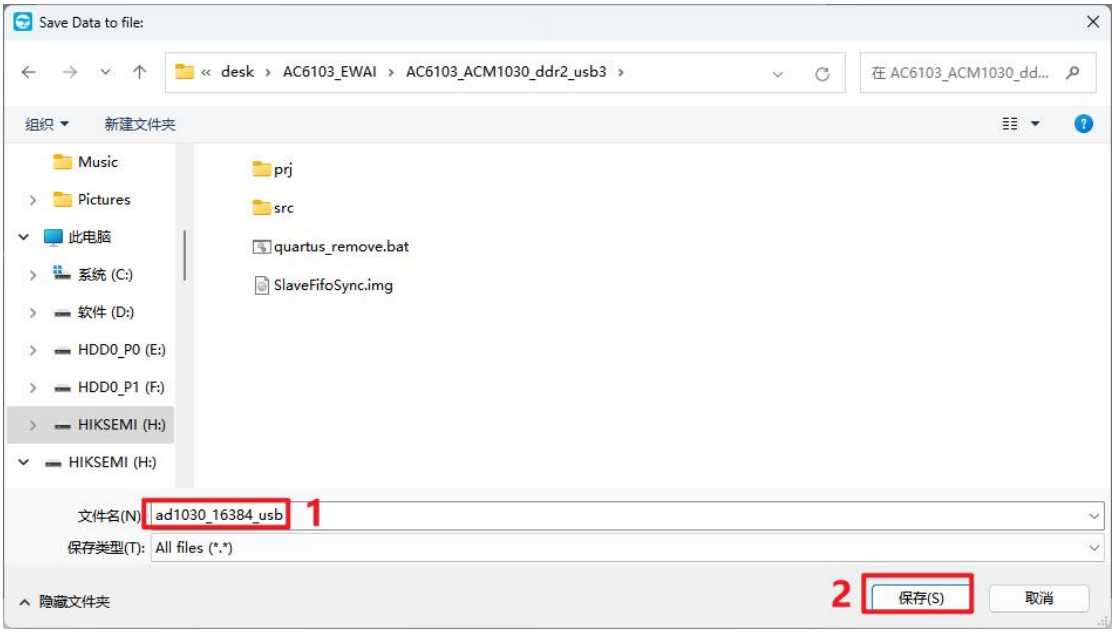

图 1-17 设置文件名称

9. Cypress 软件接收数据，并提示传输成功，如下所示。

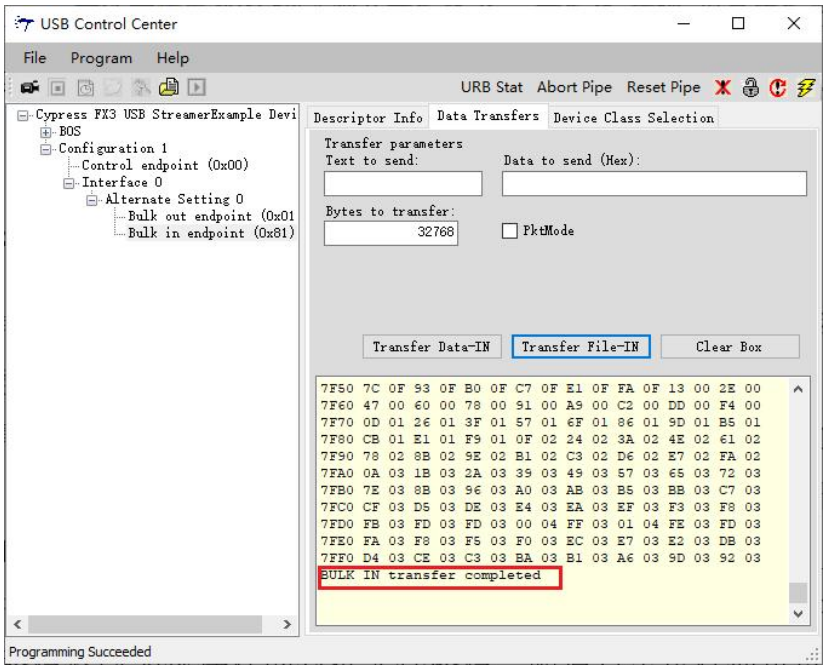

图 1-18 数据传输完成界面显示

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。

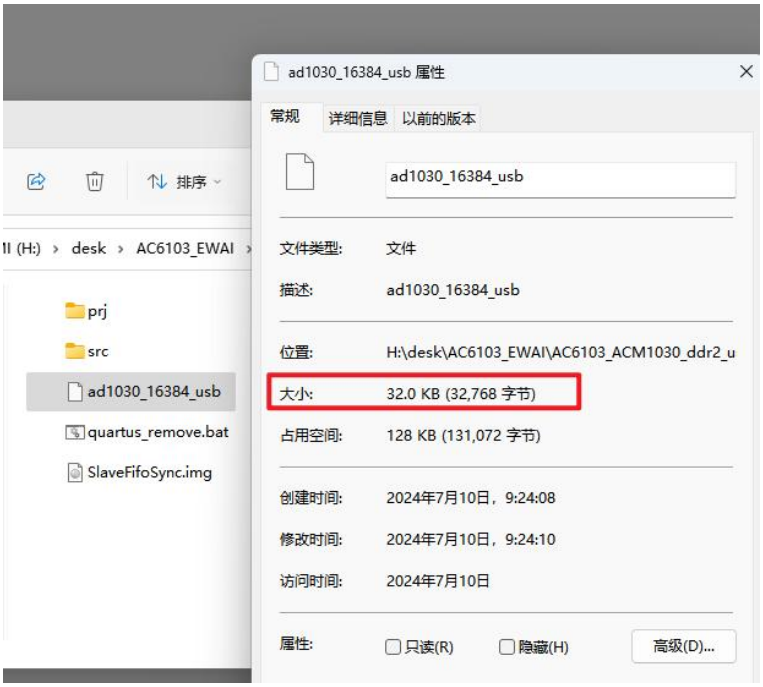

图 1-19 查看接收文件的大小

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. 采样数量必须大于 16384 字节，并且要是 16384 的整数倍，否则会出现输出框为 997 的故障码，出现之后，建议重新给开发板上电，重新下载程序和固件。

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. 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）。

1.4.4 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，。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。

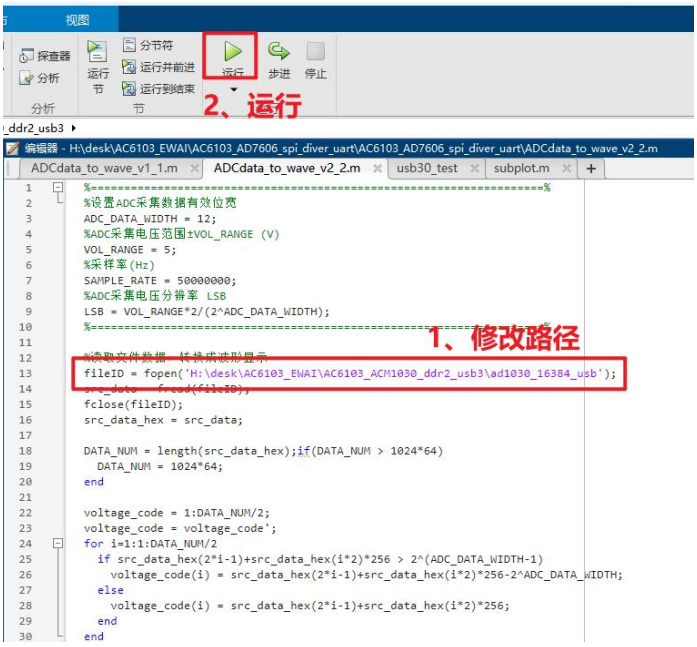

图 1-20 修改文件路径并运行

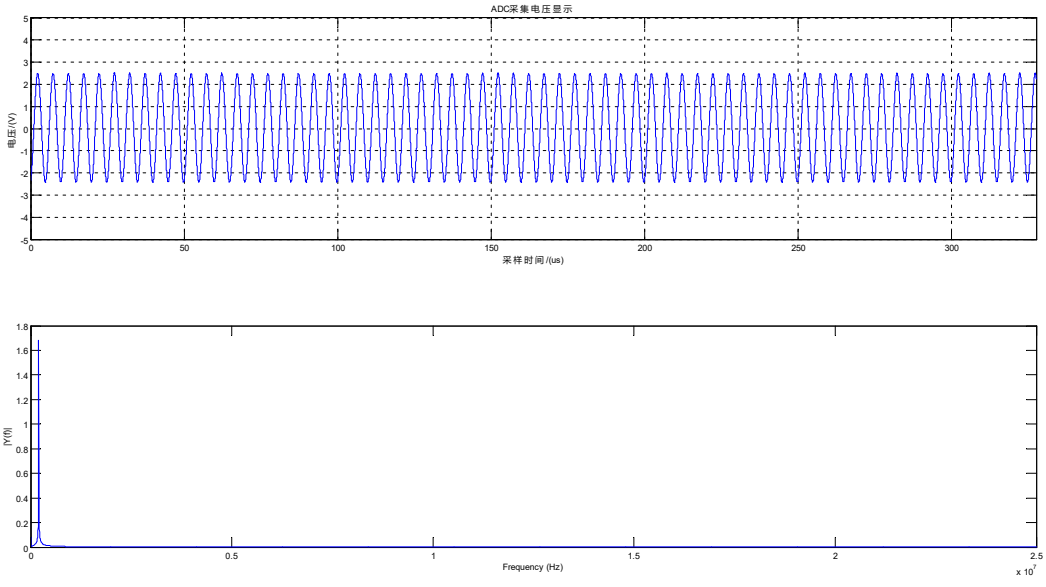

图 1-21 MATLAB 分析波形图

前面我们提到过本次实验提供的信号源为 200k，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。

1.4.5 数据采集上位机通信

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，读者可以在论坛上下载最新版本的软件，链接如下：

http:/www.corecourse.cn/forum.php?mod=viewthread&tid=29224(出处:%20 芯路恒电子技术论坛)

双击上位机软件，初始界面如下图 1-22 所示。

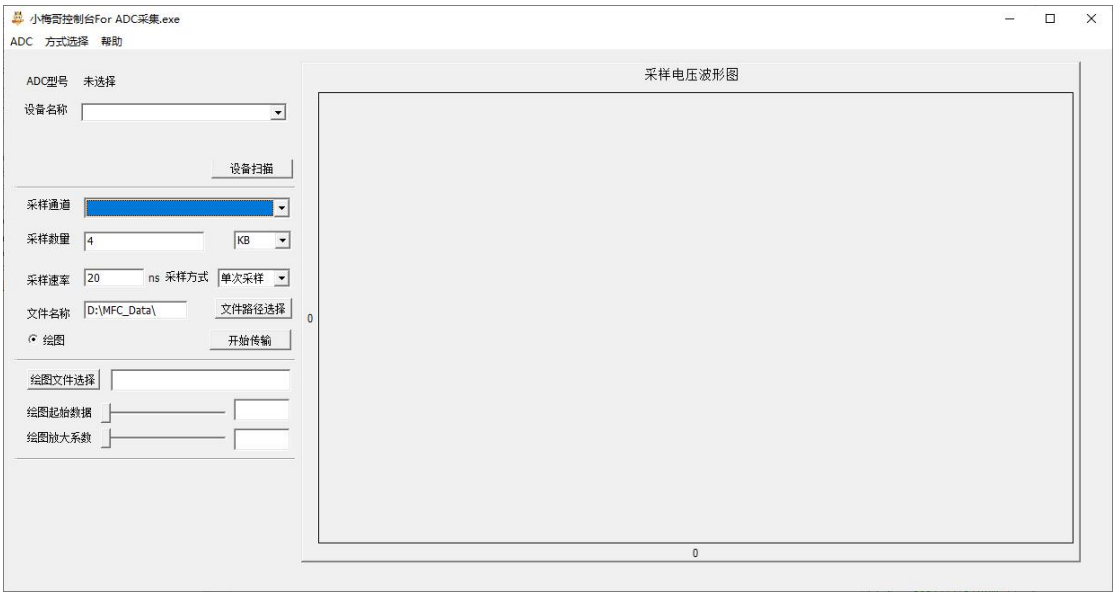

图 1-22 上位机软件初始界面显示

本次实验使用该软件的方式如下所示：

1. 点击 ADC，选择 ACM1030。
2. 点击方式，选择 USB2.0\USB3.0，这里可以看到识别到设备“(SuperSpeed)Cypress FX3 USB StreamerExample Device”，这里可以看到前缀有“SuperSpeed”，这个才是正确识别到了 USB3.0 设备，如果是“HighSpeed”则说明识别成了 USB2.0 的设备，那就请检查一下硬件连接以及是否下载了固件
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），采样通道我们这里设置为 CH1。
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，这里采样数量显示 4K，实际我们上位机 USB 实际传输了 8K 的数据，只保存了 4K 的数据，这是因为 USB3.0 传输小于 8K 的数据时，会报错，所以我们进行了该处理。

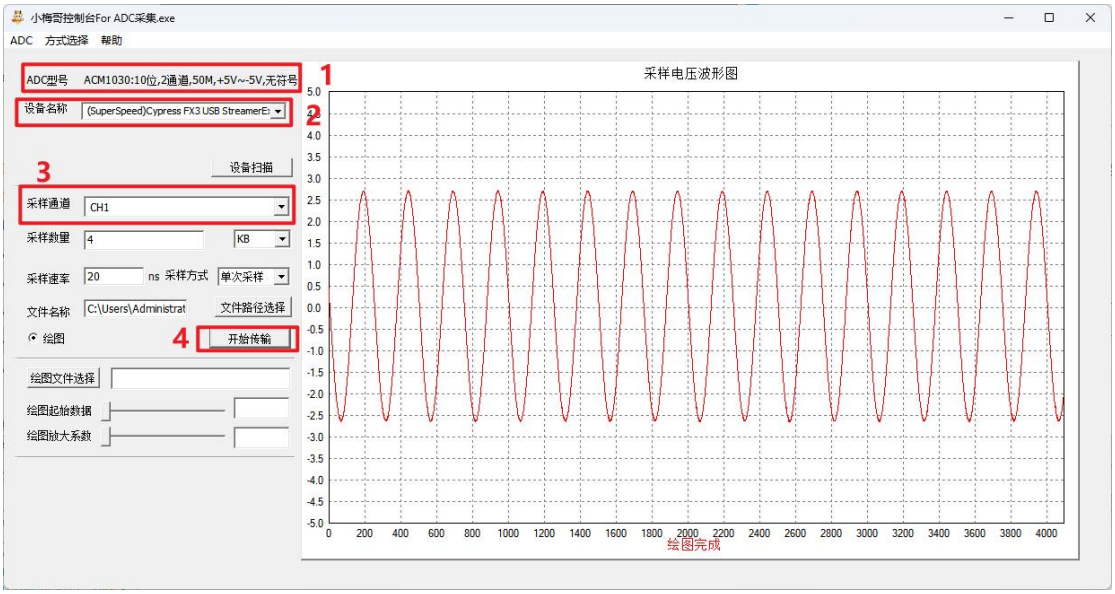

图 1-23 数据采集上位机显示图

通过上位机采集到的数据文件，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 200khz，电压在正负 2.5V 左右，与我们输入的信号一致。

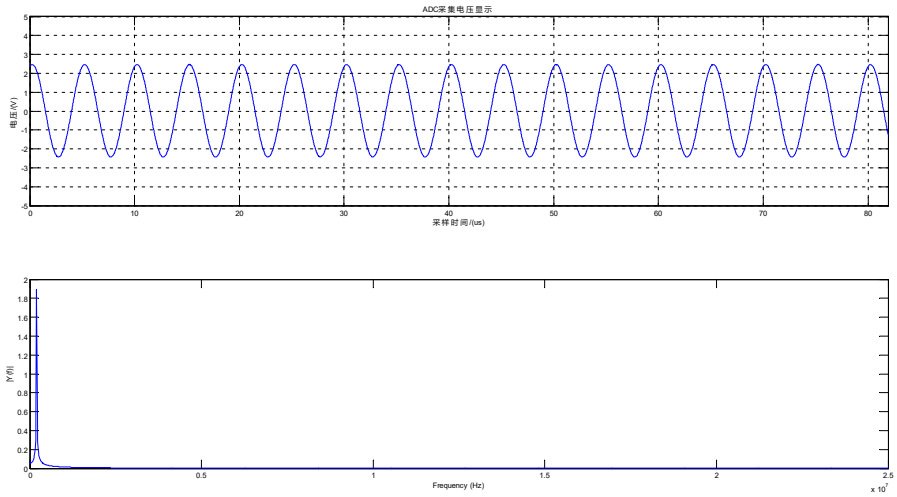

图 1-24 MATLAB 进一步波形分析图

本次实验我们还支持双通道采集，我们这里再给 ADC 模块的通道 2 给一个 100K，VPP 为 5V 的正弦波，然后上位机上勾选 CH1、CH2，这时我们便可以

看到采集的两个通道的数据，波形图右边会显示对应通道的波形颜色，如下所示。这里的采样数量指的是两个通道加起来的数量，比如采样数据设置为 4K，则两个通道每个通道采集的数量为 2K。

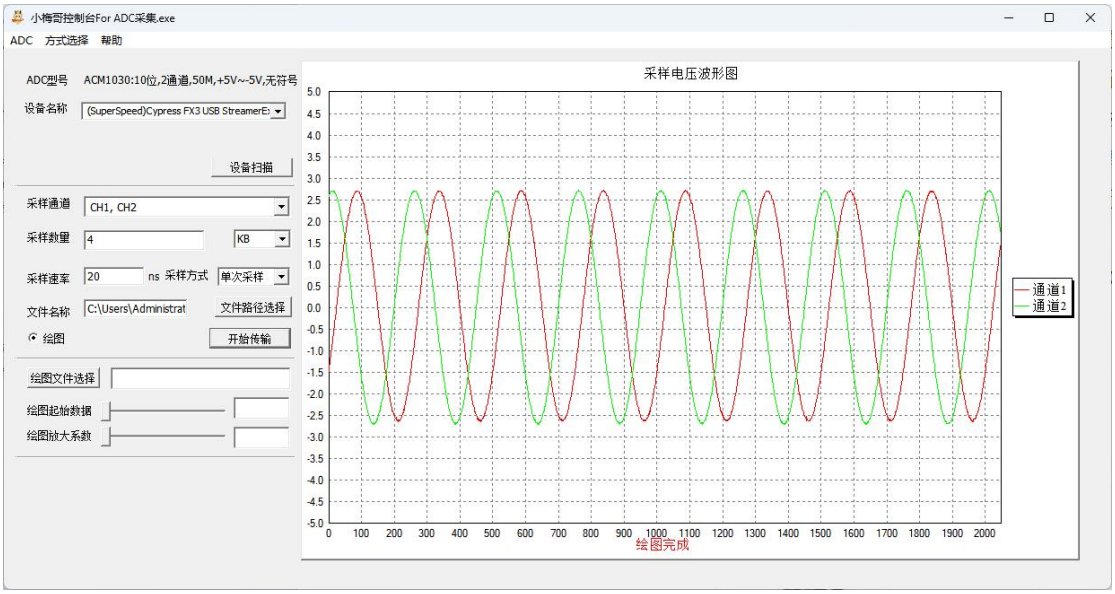

图 1-25 双通道波形显示

1.5 思考与总结

本次实验介绍了基于 ACM1030 的 USB 收发，用户通过 USB 调试工具 CyControl 向开发板发送指令数据配置 ACM1030 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 DDR2 中，再由 USB 将 DDR2 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。