1 基于 AC6103 的 spi 驱动 AD7606 的数据采集

工程源码	|--AC6103_AD7606_spi_diver_uart.rar
相关视频课程	无

章节导读

本节介绍了基于 FPGA 的 AD7606 串行驱动数据采集系统实验步骤。案例基于 Intel(Altera)CycloneIV 系列 FPGA，结合 ADI 公司的 16 位 8 通道并行采样 ADC 芯片，通过开发板上串口，实现对 AD7606 型 8 通道 16 位 ADC 的数据转换控制并输出。

1.1 系统整体设计

通过电脑上的串口调试助手，将命令帧进行发送，然后通过 AC6103 开发板上的串口接收，随后将接收到的数据转换成命令，从而实现对 ACM7606 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，ACM7606 模块开始采集数据，通过 FIFO 进行缓存，最后数据通过串口传输至电脑，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。

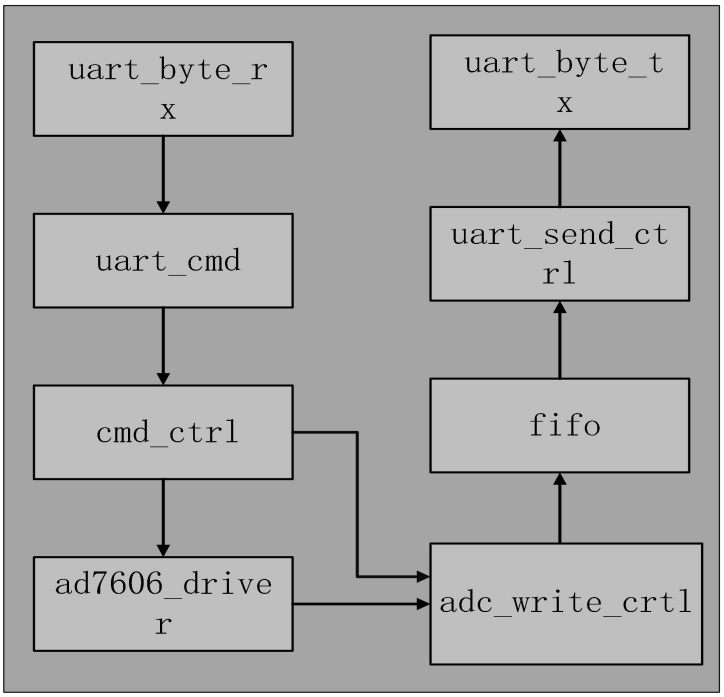


图 1-1 系统框图

对于上图中各个模块的功能介绍如下：

1、uart_byte_rx 模块：串口接收模块。

2、uart_cmd 模块：接收转命令模块，将串口传输过来的指令数据帧进行拆解，提取出每个命令控制帧。

3、cmd_ctrl 模块：指令控制模块，将前面解析出的数据，作为各个类型，分别储存在各个寄存器中，对采样的数据量和采样速率进行设置。

4、ad7606_driver 模块：AD7606 的 SPI 驱动模块。

5、adc_write_ctrl 模块：往 FIFO 写数据控制模块。

6、FIFO 模块：调用 FIFO IP 核。

7、uart_send_ctrl 模块：串口发送控制模块。

8、uart_byte_tx 模块：串口发送模块。

1.2 ACM7606 模块简介

ACM7606 数据采集模块使用的是 ADI 公司的 16 位 8 通道同步采样模数转换器 AD7606，模块图如下图 1-2 所示。

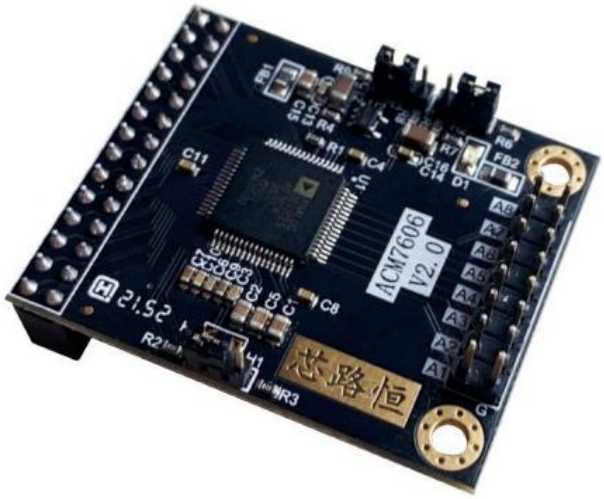


图 1-2 AD7606 模块图

AD7606 是 16 位 8 通道同步采样模数数据采集系统（DAS）。内置模拟输入箝位保护、二阶抗混叠滤波器、跟踪保持放大器、16 位电荷再分配逐次逼近型模数转换器（ADC）、灵活的数字滤波器、2.5V 基准电压源、基准电压缓冲以及高速串行和并行接口。AD7606 采用 5V 单电源供电，可以处理±10V 和±5V 真双极性输入信号。同时所有通道均能以高达 200kSPS 的吞吐速率采样。输入箝位保护电路可以耐

受最高达±16.5V 的电压。无论以何种采样频率工作，其模拟输入阻抗均为 1MΩ。采用单电源工作方式，具有片内滤波和高输入阻抗，因此无需驱动运算放大器和外部双极性电源。AD7606 抗混叠滤波器的 3dB 截止频率为 22kHz；当采样频率为 200Ksps 时，它具有 40dB 的抗混叠抑制特性。

芯片对外提供 SPI 和并行的数字接口。当 AD7606 的 8 个通道全部以 200KPS 的最高速率进行转换时，数据输出速率达到 25.6Mbps，需要使用高性能 MCU 的 SPI 外设才能勉强该速率要求。因此可以使用 16 位并口来进行数据的传输，提高数据传输速率。当 AD7606 应用在 FPGA 系统中的时候，使用 SPI 串行接口和并行接口都能够轻松的满足数据传输的速率需求。当在 FPGA 系统上应用 AD7606 时，可以通过在 FPGA 上设计 AD7606 控制转换逻辑，将转换结果数据直接存储到片上的存储器如 FIFO 或者 RAM 中，也可以存储到 FPGA 片外的存储器如 SRAM 或 SDRAM 中，然后由其他主控芯片如 MCU 或 DSP 读出，或者直接在 FPGA 内部进行数据的运算和处理。当然，由于 FPGA 片上可以设计软核控制器，也可以直接使用软核控制器完成数据的处理和传输工作。

1.2.1 功能框图

AD7606 的功能框图如下图 1-3 所示。

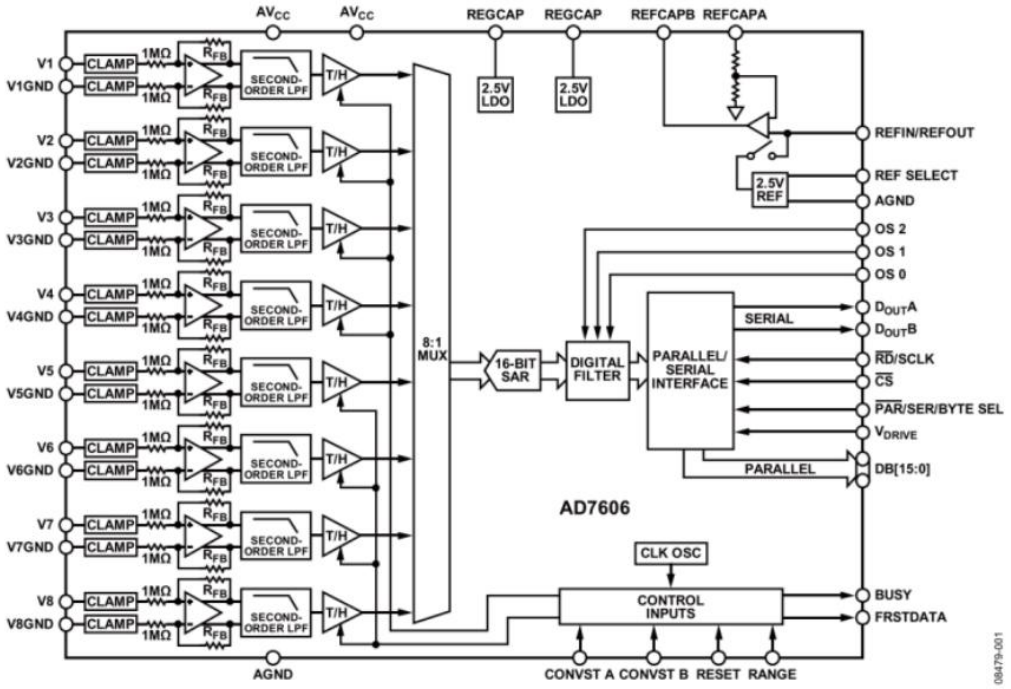


图 1-3 AD7606 功能框图

如上图所示，采集到的数据在经过稳压滤波和采样保持后通过 8 选 1 多路选择器被分别送入到 16 位逐次逼近型 ADC 芯片 AD7606 中进行转换，最后经由数字滤波后输出，当数据以串行模式输出时，数据会从 DoutA、DoutB 中输出，如果数据是以并行方式输出，那么数据将从 DB[15:0]中输出。

1.2.2 模拟输入

AD7606 可处理真双极性、单端输入电压。RANGE 引脚的逻辑电平决定所有模拟输入通道的模拟输入范围。如果此引脚与逻辑高电平相连，则所有通道的模拟输入范围为±10V。如果此引脚与逻辑低电平相连，则所有通道的模拟输入范围为±5V。AD7606 的模拟输入阻抗为 1MΩ。这是固定输入阻抗，不随 AD7606 采样频率而变化。AD7606 的输入结构如下所示，其各路模拟输入均含有箝位保护电路。虽然采用 5V 单电源供电，但此模拟输入箝位保护允许输入过压达到±16.5V。

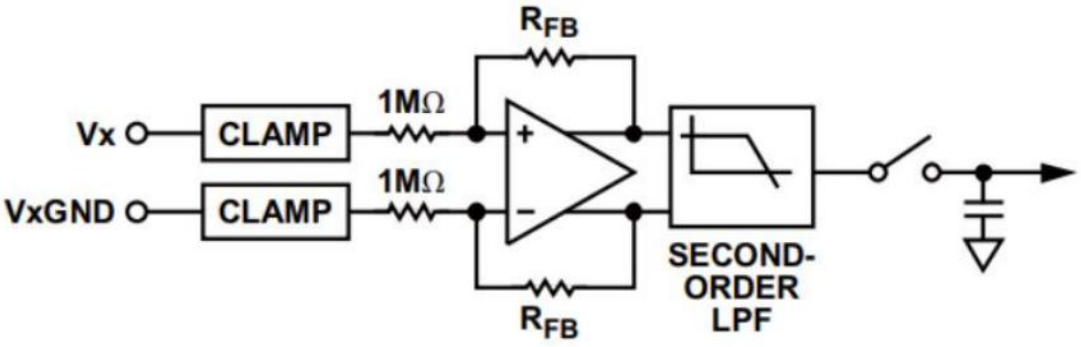


图 1-4 AD7606 模拟输入结构

1.2.3 数字滤波和过采样

AD7606 内置一个可选的数字一阶 sinc 滤波器，在使用较低吞吐率或需要更高信噪比或更宽动态范围的应用中，应使用该滤波器。数字滤波器的过采样引脚 OS[2:0]控制（具体参考 AD7606 数据手册）。OS2 为 MSB 控制位。OS0 为 LSB 控制位，下表 1-1 提供了用来选择不同过采样倍率的过采样位解码。

表 1-1 不同过采样倍率的过采样位解码

OS[2:0]	过采样倍率	5V 范围 SNR(dB)	10V 范围 SNR(dB)	5V 范围 3dB 带宽(kHz)	10V 范围 3dB 带宽(kHz)	最大吞吐量 CONVST 频率 (kHz)

000	No OS	89	90	15	22	200
001	2	91.2	92	15	22	100
010	4	92.6	93.6	13.7	18.5	50
011	8	94.2	95	10.3	11.9	25
100	16	95.5	96	6	6	12.5
101	32	96.4	96.7	3	3	6.25
110	64	96.9	97	1.5	1.5	3.125
111	无效					

OS 引脚在 BUSY 下降沿锁存，从而设置下一个转换的过采样倍率，如下图 1-5 OS 引脚在 BUSY 下降沿锁存时序示意图所示，如果 OS 引脚选择过采样倍率 8，则下一个 CONVST x 上升沿采集各通道的第一个采样点，一个内部产生的采样信号采集所有通道的其余 7 个样点，然后对这些样点求平均值，以改进 SNR 性能。开启过采样时，CONVST A 和 CONVST B 引脚必须连在一起驱动，转换过程中 BUSY 保持高电平的时间会延长。BUSY 保持高电平的实际时间取决于所选的过采样倍率，过采样倍率越高，则 BUSY 保持高电平的时间或总转换时间越长。

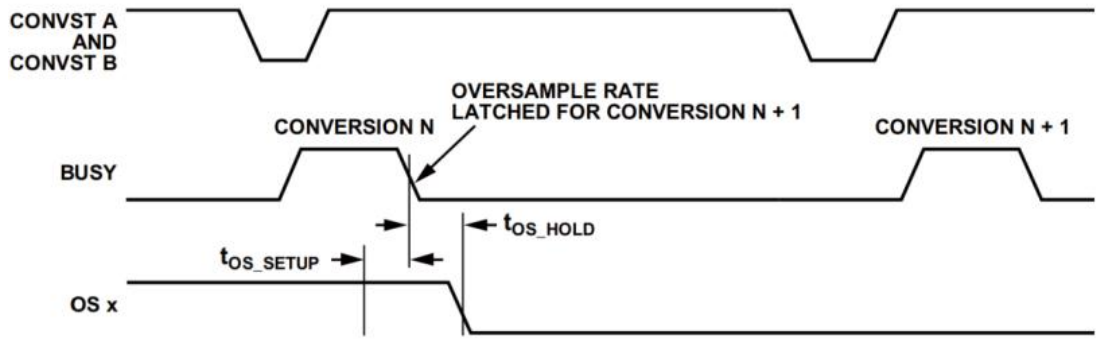


图 1-5 OS 引脚在 BUSY 下降沿锁存时序示意图

1.2.4 工作时序

AD7606 根据采样方式不同具有多种驱动时序，本次实验采用的为并行输出（即 8 个 16 位的数据通过 16 根并行线一个接着一个输出），转换后读取模式。其时序图由两部分组成：完成 AD 转换和读取 AD 数据。其中的时间可以参考 ADI 公司的手册。当 CONVST A 和 CONVST B 通道都变为上升沿时，BUSY 信号转变为高电平，代

表转换开始，知道 BUSY 的下降沿到来，代表数据已经转换完成，正在锁存至输出数据寄存器中，当$\overline{CS}$变为下降沿时，数据将会被输送到总线上。并行工作模式下，当$\overline{CS}$和$\overline{RD}$都为低电平时，会使能总线，将转换结果输出到并行数据总线上，当 V1 转换结果开始输出之后，FRSTDATA 会随后转变为高电平，表示输出数据总线可以提供 V1 的结果。并行模式下，每次数据的输出为 16 位，对应一个通道。

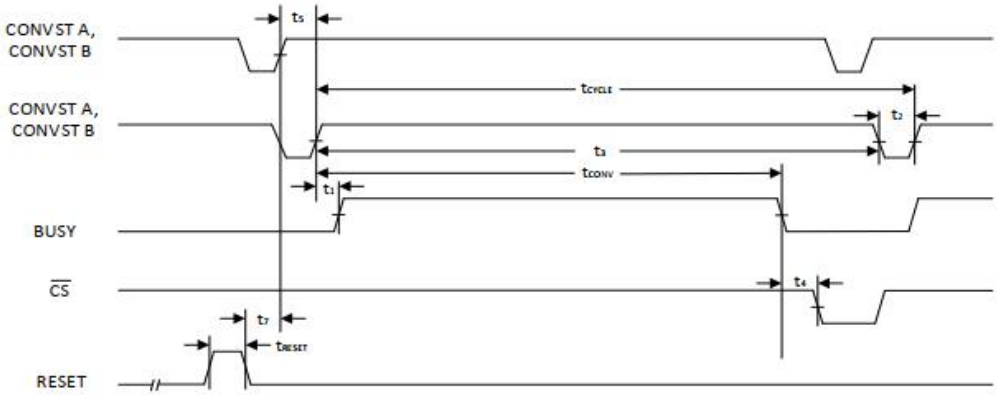


图 1-6 CONVST 时序—转换之后读取

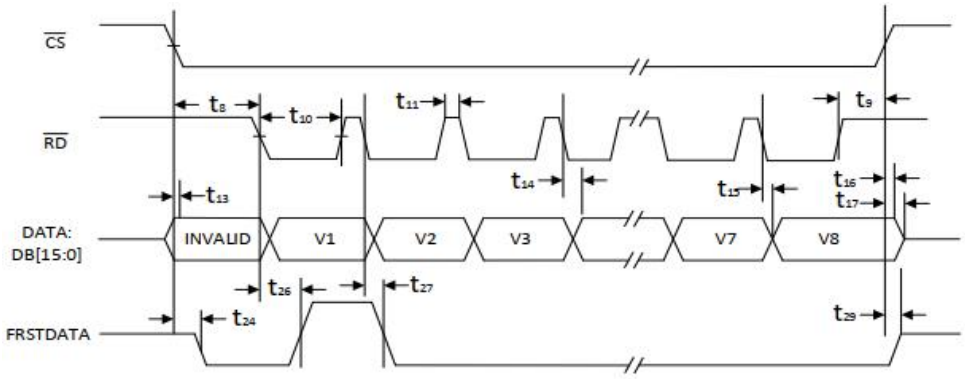


图 1-7 并行模式，独立的$\overline{CS}$和$\overline{RD}$脉冲

1.3 模块设计

下面给将对本次实验需要设计的模块进行介绍。

（注：uart_byte_rx 模块和 uart_byte_tx 模块这里就不进行介绍，详细的请参考教材的内容）

1.3.1 uart_cmd 模块

接收转命令模块 uart_cmd，将串口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。

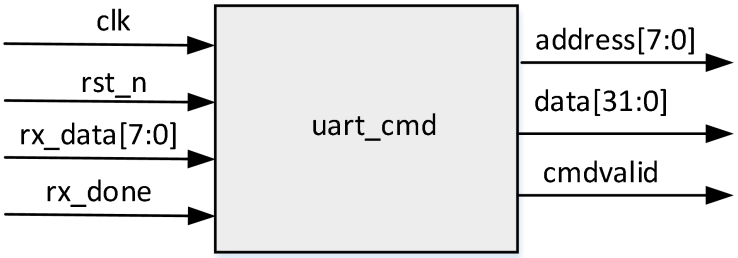


图 1-8 接收转命令模块

表 1-2 串口指令接收模块接口功能描述

接口名称	I/O	功能描述
clk	I	输入工作时钟，频率为50MHz
rst_n	I	模块复位，低电平复位
rx_done	I	1byte 数据接收完成标志信号
rx_data[7:0]	I	指令数据输入端
cmdvalid	O	指令信息解析完成标志信号
address[7:0]	O	寄存器地址信息数据
data[31:0]	O	采样设置配置数据信息

在这里，我们给出接收转命令模块的代码：

```
module uart_cmd(
    clk             ,
    rst_n           ,
    rx_data         ,
    rx_done         ,
    address         ,
    data            ,
    cmdvalid
);

    input clk;
    input rst_n;
    input [7:0] rx_data;
    input rx_done;
    output reg [7:0]address;
    output reg [31:0] data;
```

```
output reg cmdvalid;

reg [7:0] data_str[7:0];

always @ (posedge clk)
if (rx_done)
    begin
        data_str[7] <= rx_data;
        data_str[6] <= data_str[7];
        data_str[5] <= data_str[6];
        data_str[4] <= data_str[5];
        data_str[3] <= data_str[4];
        data_str[2] <= data_str[3];
        data_str[1] <= data_str[2];
        data_str[0] <= data_str[1];
    end

reg r_rx_done;
always @ (posedge clk)
r_rx_done <= rx_done;

always @ (posedge clk or negedge rst_n)
if (!rst_n)
    begin
        address <= 8'd0;
        data <= 32'd0;
        cmdvalid <= 1'd0;
    end
else if (r_rx_done)
begin
if ((data_str[0] == 8'h55) && (data_str[1] == 8'hA5) && (data_str[7] == 8'hF0))
    begin
        data[7:0] <= data_str[6];
```

```
            data[15:8] <= data_str[5];
            data[23:16] <= data_str[4];
            data[31:24] <= data_str[3];
            address <= data_str[2];
            cmdvalid <= 1'b1;
        end
    end
    else
        cmdvalid <= 1'b0;
```

1.3.2 cmd_ctrl 模块

指令控制模块，将前面解析出的数据，作为各个类型，分别储存在各个寄存器中，对采样的数据量和采样速率进行设置。该模块的结构框图如下图 1-9 所示。

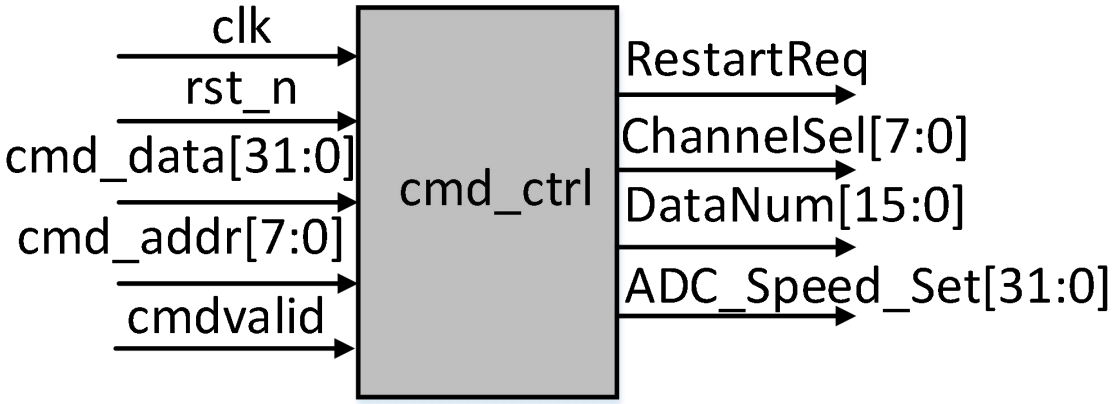


图 1-9 指令解析模块

表 1-3 指令解析模块接口功能描述

接口名称	I/O	功能描述
clk	I	输入工作时钟，频率为50MHz
rst_n	I	模块复位，低电平复位
cmdvalid	I	指令信息解析完成标志信号
cmd_addr[7:0]	I	寄存器地址信息数据
cmd_data[31:0]	I	采样设置配置数据信息
RestartReq	O	采样使能信号
ChannelSel	O	采样通道选择信号

DataNum	O	采样数据量信号
ADC_Speed_Set	O	采样率设置信号

在这里，我们给出指令解析的代码和讲解：

```
module cmd_ctrl(
    clk             ,
    rst_n           ,
    cmd_data        ,
    cmd_addr        ,
    cmdvalid        ,
    RestartReq      ,
    ChannelSel      ,
    DataNum         ,
    ADC_Speed_Set
);

    input clk;
    input rst_n;
    input [31:0]cmd_data;
    input [7:0]cmd_addr;
    input cmdvalid;
    output reg RestartReq;
    output reg [7:0]ChannelSel;
    output reg [15:0] DataNum;
    output reg [31:0] ADC_Speed_Set;

    always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        begin
            RestartReq <= 1'b0;
            ChannelSel <= 8'b1111_1111;
            DataNum <= 16'd32;
            ADC_Speed_Set <= 32'd10;
        end
```

```
    else if (cmdvalid)begin
        case (cmd_addr)
            0   :   begin RestartReq <= 1'b1; end
            1   :   begin ChannelSel <= cmd_data[7:0];end
            2   :   begin DataNum <= cmd_data[15:0];end
            3   :   begin ADC_Speed_Set <= cmd_data;end
            4   :   begin RestartReq  <=  1'b1; ChannelSel<=  cmd_data[7:0];
                    DataNum<=  cmd_data[23:8]; end
            default :;
        endcase
    end
    else
        RestartReq <= 1'b0;

endmodule
```

ACM7606 的控制指令，由 8 个字节的数据组成，前两个字节 D0 ，D1 用 55、A5，最后一个字节 D7 帧尾用 F0，标明这是一个接收的指令，第三个字节 D2 ，标明的是控制存储地址，本工程中，我们定义 00 是发送启动命令，01 是采样通道号，02 是采样深度。

串口一次发送的数据内容为 1 个字节，为了实现通过串口修改这些寄存器的值，需要发送多个字节才能实现，为此，设计了简单的串口数据帧，该帧一帧数据共 8 个字节，包含帧头、帧尾、地址段（决定任务设定目标）、数据段。帧格式如下所示。

表 1-4 指令帧格式

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

下面讲解一下典型的参数设置方法：

如果采 512 字节的数据，则设置为： 55 A5 02 00 00 01 00 F0

如果采 32768 字节的数据，则设置为： 55 A5 02 00 00 40 00 F0

采样速率如果是 50m，整个字节为： 55 A5 00 00 00 00 00 F0

采样速率如果是 5k，整个字节为： 55 A5 00 00 00 27 0F F0

采样通道如果是 7606 的第一通道，则设置为： 55 A5 01 00 00 00 01 F0

采样通道如果是 7606 的第二通道，则设置为： 55 A5 01 00 00 00 02 F0

采样通道如果是 7606 的第三通道，则设置为： 55 A5 01 00 00 00 04 F0

采样通道如果是 7606 的第四通道，则设置为： 55 A5 01 00 00 00 08 F0

采样通道如果是 7606 的第五通道，则设置为： 55 A5 01 00 00 00 10 F0

采样通道如果是 7606 的第六通道，则设置为： 55 A5 01 00 00 00 20 F0

采样通道如果是 7606 的第七通道，则设置为： 55 A5 01 00 00 00 40 F0

采样通道如果是 7606 的第八通道，则设置为： 55 A5 01 00 00 00 80 F0

这里对采样速率的设置做一个说明，这里是设置一个计数的值 27 0F，而如果为 0，采样和时钟保持一致 50M 时钟就是 50M，设置计数值后就可以改变采样频率，设置为 1 就是 25M。27 0F 换算成十进制是 9999，采样速率设置是 5k，

他们的关系如下：

$$设置计数值 = Fclk/Fs - 1$$

Fs 是期望的采样率，Fclk 是系统时钟。

1.3.3 ad7606_driver 模块

AD7606 SPI 驱动模块，该模块的结构框图如下图 1-10 所示。

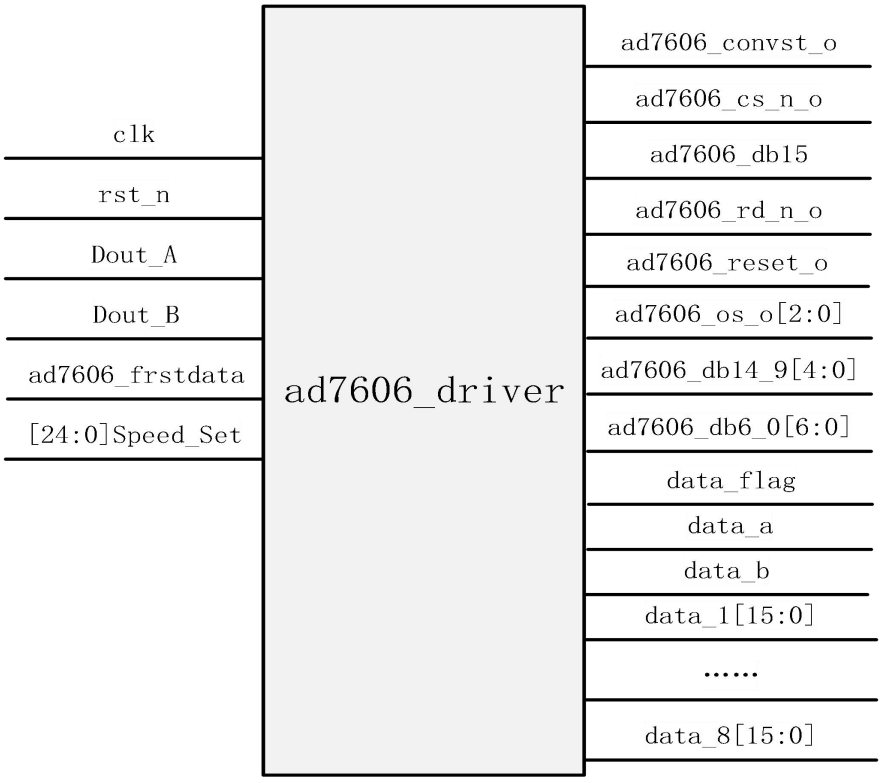


图 1-10 AD7606 控制器驱动模块结构框图

该控制器接口分为四个类，第一类为时序逻辑模工作所必须的时钟和复位信号（Clk、Reset_n），第二类是 AD7606 控制器与 AD7606 芯片管脚相连的各种功能控制和数据信号，第三类是设置控制器工作状态/工作数据的用户控制接口，第四类是控制器的结果输出接口。对于用户来说，只需要关注第三类和第四类接口的使用，即可快速高效的使用该控制器来控制 AD7606 芯片完成数据转换。对于该模块的信号说明如下所示。

表 1-5 AD7606 控制器模块信号说明

类	信号名称	I/O	信号意义
系统信号	Clk	I	模块系统时钟 50M
	rst_n	I	模块复位信号，低电平复位
控制信号	Speed_Set	I	采样速率控制端口，Speed_Set = 1000000000/20/speed - 1
ADC 芯片控制信号	ad7606_cs_n_o	O	ad7606 芯片选中控制信号，可以从 AD7606 中读取转换结果时，需要使该信号为低电平
	ad7606_rd_n_o	O	ad7606 转换结果读取信号，该信号的下降沿，AD7606 将特定通道的采样结果送到 16 位数据线上，供外部读取。外部可以在 rd_n 信号的上升沿读取 16 位数据线上的结果
	ad7606_os_o[2:0]	O	ad7606 过采样控制信号，使用过采样可以进一步提高 ad7606 的采样精度，使用过采样会降低 ad7606 的有效转换速率，关于过采样的功能和使用方法，可以参考 ad7606 的 datasheet，默认为 0，则表示不使用过采样。能够运行在最高的转换速率（200Ksps）
	ad7606_reset_o	O	ad7606 的复位信号，复位 ad7606 内部各个功能单元的工作状态
	ad7606_convst_o	O	ad7606 转换开始信号，该信号的上升沿启动 ad7606 内部的采样转换逻辑开始新一轮的采样转换
	ad7606_db[15:0]	O	ad7606 的 16 位数据线，读取时，输出对应通道的转换结果

数据结果输出端口和标志信号	data_flag	O	转换结果有效标志信号
	data_a	O	数据输出
	data_b	O	
	data_1 ~data_8	O	

在这里，我们给出驱动的代码：

```
module ad7606_driver(
    Clk                 ,   //驱动模块工作时钟信号 50MHz，四分频可得到模块工作时钟
    Rst_n               ,   //模块复位信号，低电平复位
    Speed_Set           ,   //采样频率控制参数输入
    Dout_A              ,   //AD7606 模块数据采集串行输入 A，输入 1—4 通道数据
    Dout_B              ,   //AD7606 模块数据采集串行输入 B，输入 5—8 通道数据
    ad7606_frstdata     ,   //FRSTDATA 输出信号指示何时在并行、字节或串行接口上回读第一通道 V1。在串行模式下，FRSTDATA 在 CS 下降沿变为高电平，因为此时将在 DoutA 上输出 V1 的 MSB。在 CS 下降沿之后的第 16 个 SCLK 下降沿,它恢复低电平。
    ad7606_convst_o     ,   //转换有效信号，用来启动模拟输人通道转换。本模块默认短接。引脚从低电平变为高电平时，相应模拟输人的前端采样保持电路被设置为保持。
    ad7606_cs_n_o       ,   //在串行模式使能串行数据帧传输，低电平有效，并逐个输出串行输出数据的最高有效位(MSB)。
    ad7606_db15         ,   //BYTE SEL 引脚。当 PAR/SER/BYTESEL = 1 时，BYTE SEL 引脚用来在串行接口模式与并行字节接口模式之间做出选择。串行模式设置为 0。
    ad7606_rd_n_o       ,   //在串行模式下，此引脚用作数据传输的串行时钟输入。
    ad7606_reset_o      ,   //复位输人。当设置为逻辑高电平时，RESET 上升沿复位模块。
    ad7606_os_o         ,   //过采样模式引脚，不使用过采样
    ad7606_db14_9       ,   //在串行模式下这些引脚设置为与 AGND 相连
    ad7606_db6_0        ,   //在串行模式下这些引脚设置为与 AGND 相连
```

```
    data_flag               ,   //数据采集通道有效标志信号
    data_a                  ,   //1—4 通道数据输出
    data_b                  ,   //5—8 通道数据输出
    data_1                  ,   //通道 1 数据单独输出
    data_2                  ,   //通道 2 数据单独输出
    data_3                  ,   //通道 3 数据单独输出
    data_4                  ,   //通道 4 数据单独输出
    data_5                  ,   //通道 5 数据单独输出
    data_6                  ,   //通道 6 数据单独输出
    data_7                  ,   //通道 7 数据单独输出
    data_8                      //通道 8 数据单独输出
);

    input Clk;
    input Rst_n;
    input [24:0]Speed_Set;
    input Dout_A;
    input Dout_B;
    input ad7606_frstdata;
    output reg ad7606_convst_o;
    output reg ad7606_cs_n_o;
    output ad7606_db15;
    output reg ad7606_rd_n_o;
    output reg ad7606_reset_o;
    output [2:0]ad7606_os_o;
    output [4:0]ad7606_db14_9;
    output [6:0]ad7606_db6_0;
    output reg [7:0]data_flag;
    output reg [15:0]data_a;
    output reg [15:0]data_b;
    output [15:0]data_1;
    output [15:0]data_2;
    output [15:0]data_3;
```

```
output [15:0]data_4;
output [15:0]data_5;
output [15:0]data_6;
output [15:0]data_7;
output [15:0]data_8;

assign ad7606_db15   = 0; //模式选择，串行模式为 0
assign ad7606_db14_9 = 0;
assign ad7606_db6_0  = 0;
assign ad7606_os_o   = 0; //不使用过采样

parameter DIV_PARAM = 2;
reg SCLK2X;//2 倍 SCLK 的采样时钟
reg [7:0]DIV_CNT;//分频计数器
reg En;

//生成 2 倍 SCLK 使能时钟
always @(posedge Clk or negedge Rst_n)
if (!Rst_n)
    SCLK2X <= 1'b0;
else if (DIV_CNT == DIV_PARAM - 1'd1)
    SCLK2X <= 1'b1;
else
    SCLK2X <= 1'b0;

//生成 2 倍 SCLK 使能时钟计数器
always @(posedge Clk or negedge Rst_n)
if (!Rst_n)
    DIV_CNT <= 8'd0;
else if (En)
    begin
        if (DIV_CNT == DIV_PARAM - 1'd1)
            DIV_CNT <= 8'd0;
```

```
        else
            DIV_CNT <= DIV_CNT + 1'd1;
    end
else
    DIV_CNT <= 8'd0;

reg [7:0]SCLK_CNT;
reg [1:0]cnt_speed;

always @(posedge Clk or negedge Rst_n)
if (!Rst_n)
    SCLK_CNT <= 8'd0;
else if (SCLK2X)
    begin
        if (SCLK_CNT == 8'd131)
            SCLK_CNT <= 8'd0;
        else if (En)
            SCLK_CNT <= SCLK_CNT + 1'd1;
    end
else
    SCLK_CNT <= SCLK_CNT;

reg [1:0]state;
reg [24:0]cnt;

//采样间隔
always @(posedge Clk or negedge Rst_n)
if (!Rst_n)
    cnt <= 25'd0;
else if (cnt == Speed_Set)
    cnt <= 25'd0;
else if (!En)
    cnt <= cnt + 1'd1;
```

```
wire  trig = cnt == Speed_Set;

//采样率控制
always @ (posedge Clk or negedge Rst_n)
if (!Rst_n)
    begin
        state <= 0;
        En <= 1'b0;
    end
else
    case (state)
        0   :   begin
                    if (trig)
                        begin
                            state <= 1;
                            En <= 1'b1;
                        end
                end

        1   :   begin
                    if (SCLK_CNT == 8'd131)
                        begin
                            state <= 0;
                            En <= 1'b0;
                        end
                end
        default :;
    endcase

always @ (posedge Clk or negedge Rst_n)
if (!Rst_n)
    data_flag <= 8'd0;
```

```
    else
        begin
            data_flag[0] <= SCLK_CNT == 131;
            data_flag[1] <= SCLK_CNT == 35;
            data_flag[2] <= SCLK_CNT == 67;
            data_flag[3] <= SCLK_CNT == 99;
            data_flag[4] <= SCLK_CNT == 131;
            data_flag[5] <= SCLK_CNT == 35;
            data_flag[6] <= SCLK_CNT == 67;
            data_flag[7] <= SCLK_CNT == 99;
        end

    assign  data_1 = (SCLK_CNT == 131)?data_a :   data_1;
    assign  data_2 = (SCLK_CNT == 35) ?data_a :   data_2;
    assign  data_3 = (SCLK_CNT == 67) ?data_a :   data_3;
    assign  data_4 = (SCLK_CNT == 99) ?data_a :   data_4;

    assign  data_5 = (SCLK_CNT == 131)?data_b :   data_5;
    assign  data_6 = (SCLK_CNT == 35) ?data_b :   data_6;
    assign  data_7 = (SCLK_CNT == 67) ?data_b :   data_7;
    assign  data_8 = (SCLK_CNT == 99) ?data_b :   data_8;

    reg  [15:0]data1,data2;

    always  @ (posedge  Clk or  negedge  Rst_n)
    if  (!Rst_n)
        begin
            ad7606_cs_n_o <= 1'b1;
            data_a <= 16'd0;
            data_b <= 16'd0;
            ad7606_convst_o <= 1'b1;
            ad7606_reset_o <= 1'b0;
            data1 <= 16'd0;
```

```
            data2 <= 16'd0;
        end
    else begin
        case (SCLK_CNT)
            0   :   begin
                        ad7606_cs_n_o <= 1'b1;
                        data_a <= data_a;
                        data_b <= data_b;
                        ad7606_convst_o <= 1'b0;
                        ad7606_rd_n_o <= 1'b1;
                    end

            1   :   begin
                        ad7606_convst_o <= 1'b1;
                        ad7606_cs_n_o <= 1'b0;
                        ad7606_rd_n_o <= 1'b1;
                    end

            2   :   begin
                        ad7606_rd_n_o <= 1'b1;
                        ad7606_cs_n_o <= 1'b0;
                    end

            3,
            5, 37,69,101,
            7, 39,71,103,
            9, 41,73,105,
            11,43,75,107,
            13,45,77,109,
            15,47,79,111,
            17,49,81,113,
            19,51,83,115,
            21,53,85,117,
```

```
            23,55,87,119,
            25,57,89,121,
            27,59,91,123,
            29,61,93,125,
            31,63,95,127,
            33,65,97,129:
            begin
                ad7606_rd_n_o <= 1;
            end

            4       ,36,68,100:begin        data1[15]<=         Dout_A;data2[15]<=
Dout_B;ad7606_rd_n_o <= 0;end
            6       ,38,70,102:begin        data1[14]<=         Dout_A;data2[14]<=
Dout_B;ad7606_rd_n_o <= 0;end
            8       ,40,72,104:begin        data1[13]<=         Dout_A;data2[13]<=
Dout_B;ad7606_rd_n_o <= 0;end
            10,42,74,106:begin          data1[12]<=         Dout_A;data2[12]<=
Dout_B;ad7606_rd_n_o <= 0;end
            12,44,76,108:begin          data1[11]<=         Dout_A;data2[11]<=
Dout_B;ad7606_rd_n_o <= 0;end
            14,46,78,110:begin          data1[10]<=         Dout_A;data2[10]<=
Dout_B;ad7606_rd_n_o <= 0;end
            16,48,80,112:begin           data1[9]<=          Dout_A;data2[9]<=
Dout_B;ad7606_rd_n_o <= 0;end
            18,50,82,114:begin           data1[8]<=          Dout_A;data2[8]<=
Dout_B;ad7606_rd_n_o <= 0;end
            20,52,84,116:begin           data1[7]<=          Dout_A;data2[7]<=
Dout_B;ad7606_rd_n_o <= 0;end
            22,54,86,118:begin           data1[6]<=          Dout_A;data2[6]<=
Dout_B;ad7606_rd_n_o <= 0;end
            24,56,88,120:begin           data1[5]<=          Dout_A;data2[5]<=
Dout_B;ad7606_rd_n_o <= 0;end
            26,58,90,122:begin           data1[4]<=          Dout_A;data2[4]<=
```

```
Dout_B;ad7606_rd_n_o <= 0;end
            28,60,92,124:begin              data1[3]<=              Dout_A;data2[3]<=
Dout_B;ad7606_rd_n_o <= 0;end
            30,62,94,126:begin              data1[2]<=              Dout_A;data2[2]<=
Dout_B;ad7606_rd_n_o <= 0;end
            32,64,96,128:begin              data1[1]<=              Dout_A;data2[1]<=
Dout_B;ad7606_rd_n_o <= 0;end
            34,66,98,130:begin              data1[0]<=              Dout_A;data2[0]<=
Dout_B;ad7606_rd_n_o <= 0;end

            35,67,99,131:begin  data_a <= data1;data_b <= data2;ad7606_rd_n_o <=
1;end
            default:
                begin
                    ad7606_cs_n_o <= 1;
                    data_a <= data_a;
                    data_b <= data_b;
                    ad7606_convst_o <= 1;
                    ad7606_reset_o <= 0;
                    data1 <= data1;
                    data2 <= data2;
                end
        endcase
    end
endmodule
```

1.3.4 adc_write_ctrl 模块

写控制模块，该模块的结构框图如下图 1-11 所示。

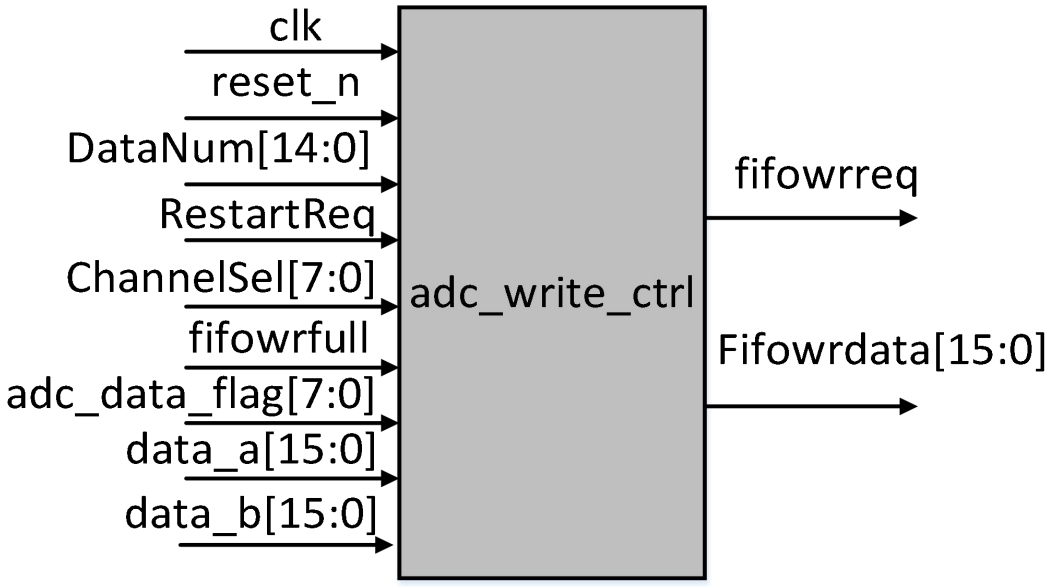


图 1-11　写控制模块框图

表 1-6 写控制模块信号说明

接口名称	I/O	功能描述
Clk	I	输入工作时钟，频率为50MHz
Reset_n	I	模块复位，低电平复位
DataNum	I	单次采集的数据个数
RestartReq	I	开始采样请求信号
ChannelSel	I	需要采样的通道选择
fifowrfull	I	FIFO 写满信号
adc_data_flag	I	ADC 输出的标志
data_a	I	ADC 输出的 1-4 通道数据
data_b	I	ADC 输出的 5-8 通道数据
fifowrreq	O	FIFO 写请求信号
fifowrdata	O	FIFO 写数据

在这里，我们给出代码：

```
`timescale 1ns/1ps
module adc_write_ctrl(
    Clk,
    Reset_n,
    DataNum,
    RestartReq,
```

```
    ChannelSel,
    fifowrreq,
    fifowrdata,
    fifowrfull,

    adc_data_flag,
     data_a,
     data_b
);

    input  Clk;
    input  Reset_n;
    input  [14:0]DataNum;      //单次采集的数据个数，最大不超过 FIFO 的深度
    input  RestartReq;         //开始采样请求信号
    input  [7:0]ChannelSel;   //需要采样的通道选择  ，每一位对应一个通道的开关
    output reg  fifowrreq;          //采集到的数据写 FIFO 请求信号
    output reg[15:0]fifowrdata; //采集到的数据
    input  fifowrfull;

    input  [7:0]adc_data_flag;
     input  [15:0]data_a;
     input  [15:0]data_b;

    reg  sample_en;   //采样
    reg  [14:0]data_cnt;

    //采样控制逻辑，每次采样请求信号到来开始采样，采样个数满了停止采样。
    always@(posedge  Clk or  negedge  Reset_n)
    if(!Reset_n)
        sample_en <= #1   1'b0;
    else  if(RestartReq)
        sample_en <= #1   1'b1;
```

```
    else if(data_cnt >= DataNum)
        sample_en <= #1  1'b0;

    //采样个数计数器，在采样使能阶段，每个 flag 到来的时候计数器自加 1
    always@(posedge Clk or negedge Reset_n)
    if(!Reset_n)
        data_cnt <= #1  15'd0;
    else if(sample_en)begin
        if(adc_data_flag & ChannelSel)
            data_cnt <= #1  data_cnt + 1'd1;
        else
            data_cnt <= #1  data_cnt;
    end
    else
        data_cnt <= #1  15'd0;

    always@(posedge Clk)
        fifowrreq <= #1  (adc_data_flag & ChannelSel)  && sample_en;

    //数据输出选择
    always@(posedge Clk)
       if(ChannelSel >= 16)
           fifowrdata <= #1  data_b;
       else
           fifowrdata <= #1 data_a;

endmodule
```

1.3.5 FIFO 模块

FIFO 配置如下图所示。

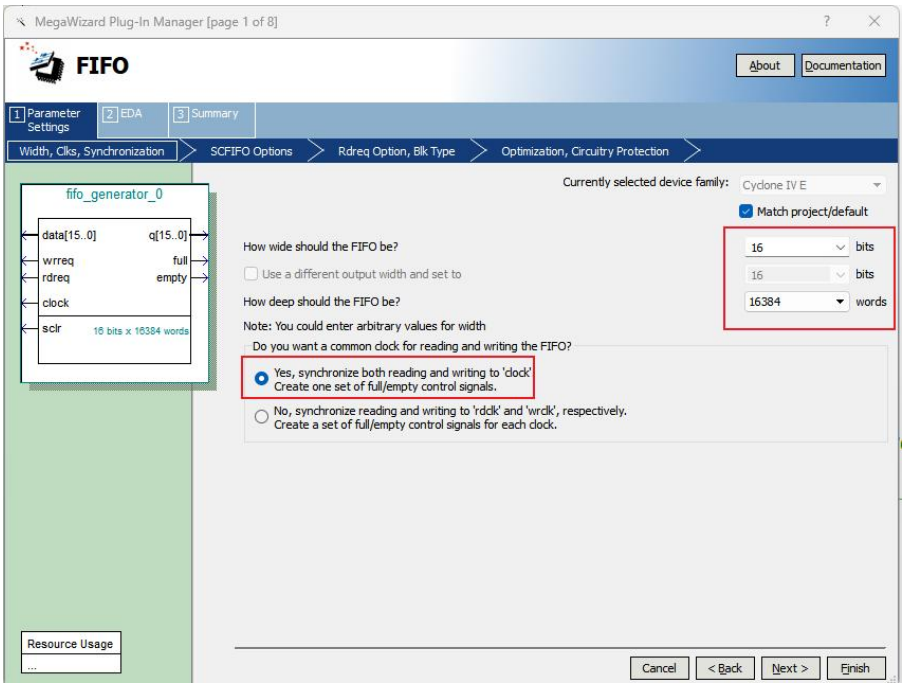


图 1-12 FIFO 配置界面 1

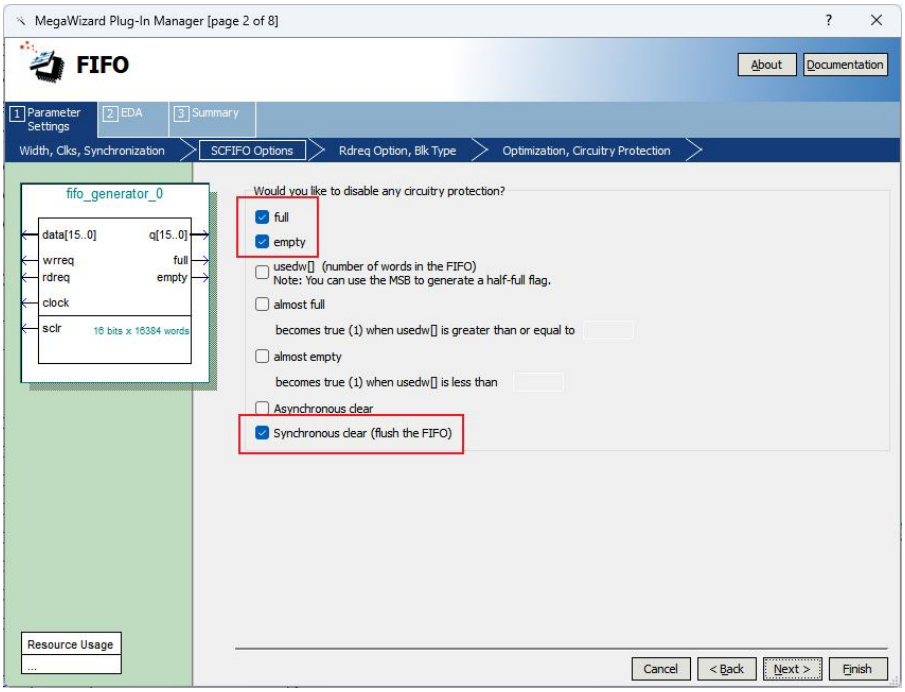


图 1-13 FIFO 配置界面 2

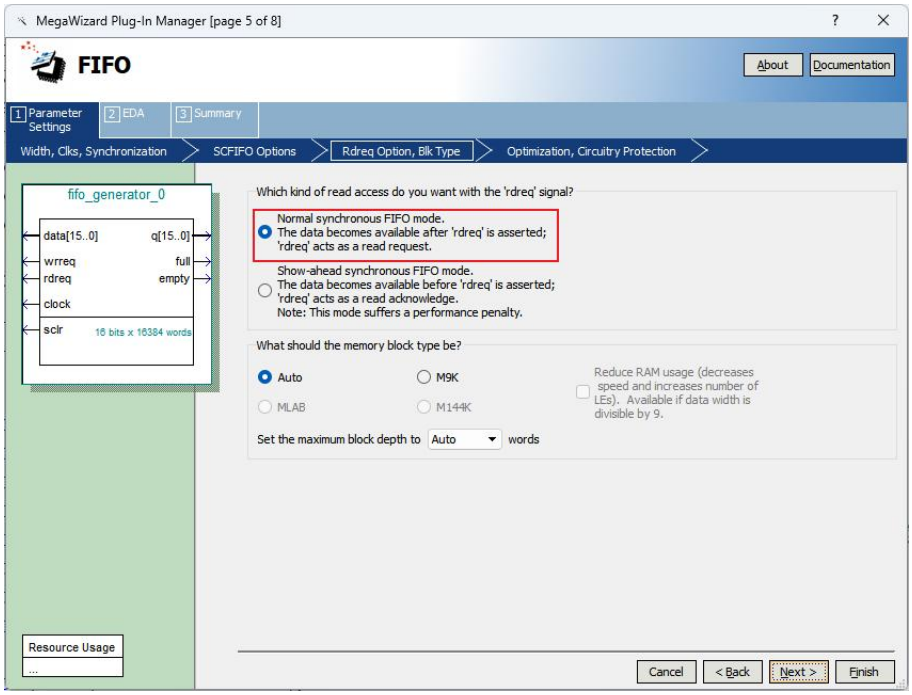


图 1-14 FIFO 配置界面 3

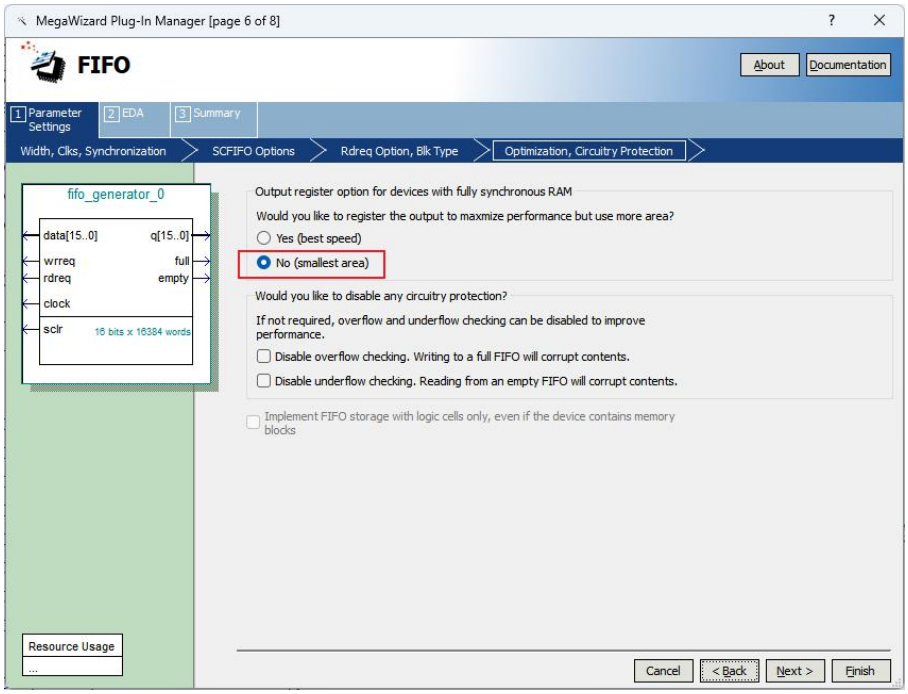


图 1-15 FIFO 配置界面 4

1.3.6 uart_send_ctrl 模块

串口发送控制模块，因为串口一次只能发一个字节的数据，所以我们要对 FIFO 出来的数据进行控制。该模块的结构框图如下所示。

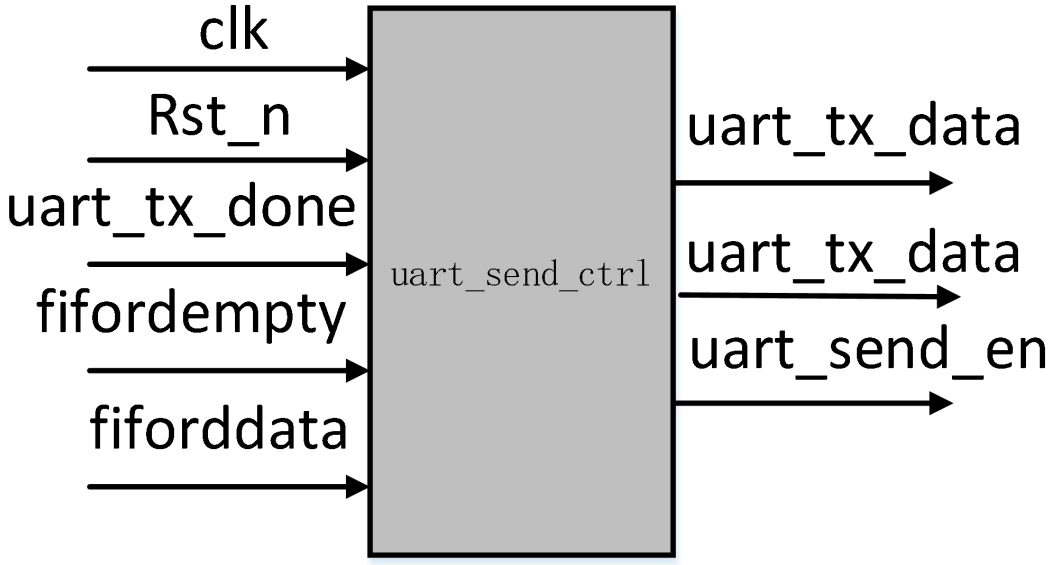


图 1-16 串口发送控制模块

表 1-7 串口发送控制模块信号说明

接口名称	I/O	功能描述
clk	I	输入工作时钟，频率为 50MHz
rst_n	I	模块复位，低电平复位
uart_tx_done	I	一个字节发送完成信号
fifordempty	I	FIFO 读空信号
fiforddata	I	FIFO 输出数据
uart_tx_data	O	串口输出数据
uart_send_en	O	串口发送使能
fifordreq	O	FIFO 读请求信号

在这里，我们给出代码：

```
module uart_send_ctrl(
    clk              ,
    rst_n            ,
    uart_tx_data,
    uart_send_en     ,
```

```
    uart_tx_done    ,
    fifordempty     ,
    fifordreq       ,
    fiforddata
);

    input clk;
    input rst_n;
    output reg [7:0] uart_tx_data;
    output reg uart_send_en;
    input uart_tx_done;
    input fifordempty;
    output reg fifordreq;
    input [15:0]fiforddata;

    reg [2:0] state;

    always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        begin
            state <= 0;
            fifordreq <= 1'b0;
        end
    else
        case (state)
            0   :   begin
                        if (!fifordempty)
                            begin
                                fifordreq <= 1'b1;
                                state <= 1;
                            end
                        else
                            begin
```

```
                    fifordreq <= 1'b0;
                    state <= 0;
                end
            end

        1   :   begin
                    state <=  2;
                    fifordreq <= 1'b0;
                end

        2   :   begin
                    state <=  3;
                end

        3   :   begin
                    state <= 4;
                    uart_send_en <= 1'b1;
                    uart_tx_data <= fiforddata[7:0];
                end

        4   :   begin
                    uart_send_en <= 1'b0;
                    if  (uart_tx_done)
                        state <= 5;
                    else
                        state <= 4;
                end

        5   :   begin
                    state <= 6;
                    uart_send_en <= 1'b1;
                    uart_tx_data <= fiforddata[15:8];
                end
```

```
            6   :   begin
                        uart_send_en <= 1'b0;
                        if (uart_tx_done)
                            state <= 0;
                        else
                            state <= 6;
                    end
            default :begin
                            state <= 0;
                            fifordreq <= 1'b0;
                            uart_send_en <= 1'b0;
                        end
        endcase

endmodule
```

1.4 板级验证

本次实验的板级验证环节，主要验证：通过电脑上的串口调试助手，将命令帧进行发送，然后通过 AC6103 开发板上的串口进行接收，随后将接收到的数据转换命令，最终实现对 AD7606 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，AD7606 模块开始采集数据，将 AD7606 模块采集的数据通过串口传输到电脑。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。

1.4.1 系统所需硬件

1、AC6103 开发板一块
2、AD7606 模块一块
3、串口线一跟
4、电源线

5、下载器

6、信号发生器

1.4.2 硬件连接

根据前面的描述准备好硬件，我们可以进行连接，连接图如下图 1-17 所示。

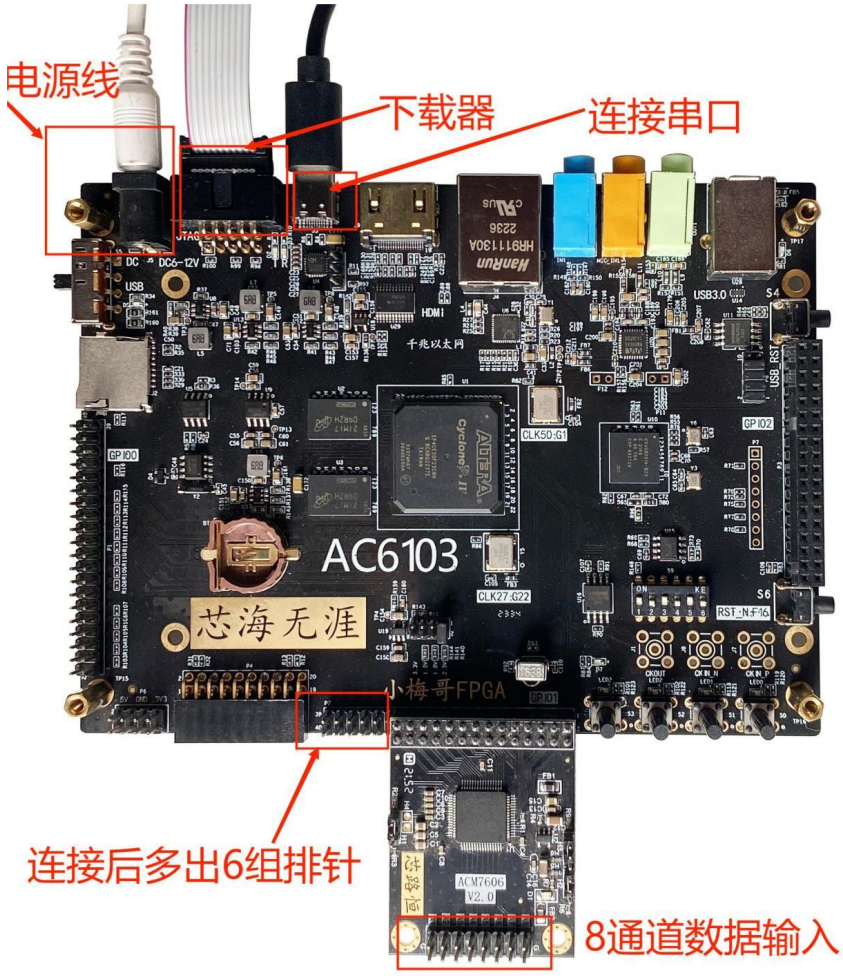


图 1-17 硬件连接图

当然，我们的 AD7606 也要设置成 SPI 模式。下图为设置 AD7606 SPI 模式的跳线帽的接法，也就是将 H2 的跳线帽接 1。

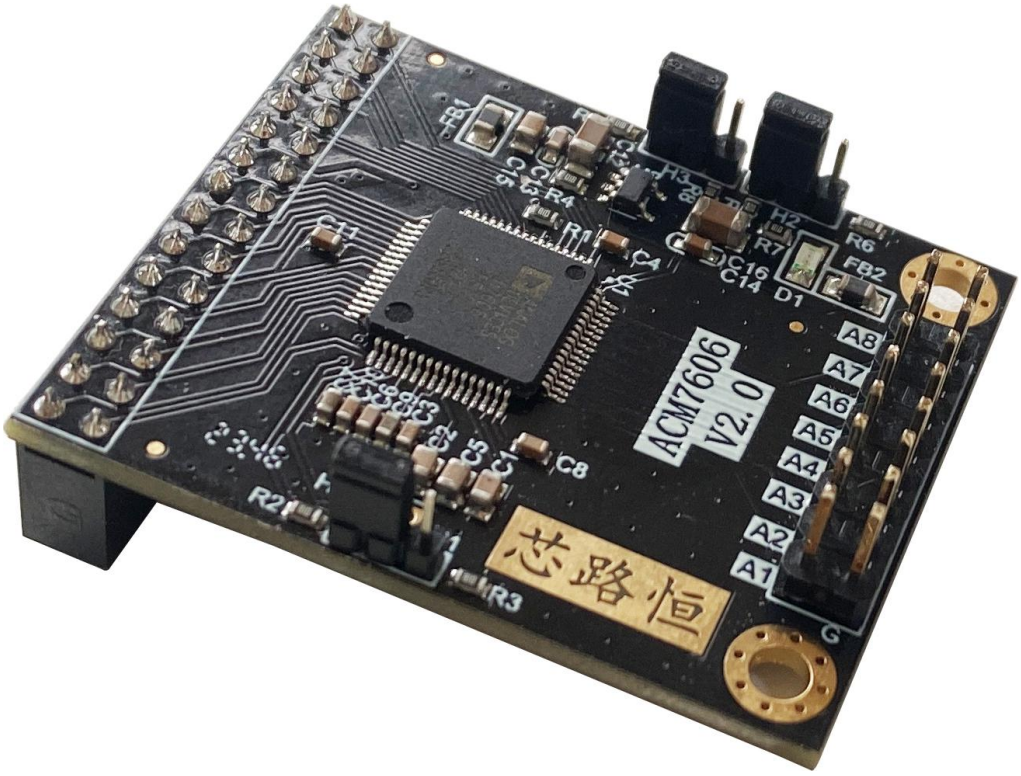


图 1-18 AD7606 SPI 模式跳线帽接法

（注：跳线 H3 用来设置 AD7606 的第 8 号管脚 Range 的电平，用来调整 AD7606 的模拟输入电压范围：H3 接 0：±5V 输入范围（模块出厂默认方式）；H3 接 1：±10V 输入范围。

跳线 H2 用来设置 AD7606 的第 6 号管脚 PSB-SEL 的电平，用来调整 AD7606 的输出为并行或串行接口：H2 接 0：并口（模块出厂默认方式）；H2 接 1：SPI 串口。

跳线 H1 用来设置 AD7606 的第 34 号管脚 REF-SEL 的电平，用来调整 AD7606 的参考电压为内部或外部源：H1 接 0：外部基准源；H1 接 1：内部基准源（模块出厂默认方式）

在 AD7606 模块背面有标注 0 和 1）

1.4.3 引脚分配

引脚分配如下所示。

表 1-8 引脚分配表

信号名	管脚
Clk	PIN_G1
Dout_A	PIN_D20

Dout_B	PIN_F20
Reset_n	PIN_F16
Rs232_Rx	PIN_W1
Rs232_Tx	PIN_Y1
ad7606_convst_o	PIN_B15
ad7606_cs_n_o	PIN_A18
ad7606_db6_0[6]	PIN_E21
ad7606_db6_0[5]	PIN_C21
ad7606_db6_0[4]	PIN_B22
ad7606_db6_0[3]	PIN_C19
ad7606_db6_0[2]	PIN_G17
ad7606_db6_0[1]	PIN_B19
ad7606_db6_0[0]	PIN_A20
ad7606_db14_9[14]	PIN_J22
ad7606_db14_9[13]	PIN_H21
ad7606_db14_9[12]	PIN_G13
ad7606_db14_9[11]	PIN_H16
ad7606_db14_9[10]	PIN_F22
ad7606_db14_9[9]	PIN_D22
ad7606_db15	PIN_H20
ad7606_frstdata	PIN_A19
ad7606_os_o[2]	PIN_A16
ad7606_os_o[1]	PIN_B14
ad7606_os_o[0]	PIN_A15
ad7606_rd_n_o	PIN_C17
ad7606_reset_o	PIN_A17

1.4.4 数据采集与分析

用户可以在电脑上发送特定格式的串口数据帧来设置 FPGA 中与控制功能相关的寄存器。FPGA 中共设计了 4 个寄存器。以下为寄存器介绍。

表 1-9 寄存器介绍

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	8	通道设置寄存器，共 8 位，对应了 8 个通道的数据存储开关，如果某通道对应的设置位为 1，则该通道的采样结果就会被存入 FIFO 并通过串口发送
DataNum	2	16	数据个数寄存器，设定总共采集传输多少个数据。注意，该寄存器设置的是总共采集的数据个数，假设设置采集 100 个数据，而且设置了 ChannelSel 为 0000_0011b，则实际每个通道采样的次数就是 50，2 个通道的数据加起来是 100 个。假设设置采集 100 个数据，而且设置了 ChannelSel 为 0011_0011b，则实际每个通道采样的次数就是 25。4 个通道加起来采集 100 个数据
ADC_Speed_Set	3	32	ADC 采样速率设置寄存器。该寄存器用来设置 ADC 每多久执行一次转换。由于 ADC 的最大采样速率为 200Ksps，所以可以通过设置该寄存器的值来让 ADC 的采样速率在 1~200Ksps 范围内调整，以适应不同的应用场景。该寄存器的值与采样速率关系为：ADC_Speed_Set = 1000000000/20/speed － 130，其中 speed 就是实际要设置的采样速率。

我们打开串口调试助手（串口猎人）。设置如下图 1-19 所示

1、选择端口号（用户可到设备管理器查看）

2、设置波特率 115200

3、启动串行端口

4、点击高级发码

图 1-19 串口猎人设置图 1

高级发码设置如下图 1-20 所示。

1、设置采样信息

（1）55 A5 02 00 00 40 00 F0

（2）55 A5 01 00 00 00 01 F0

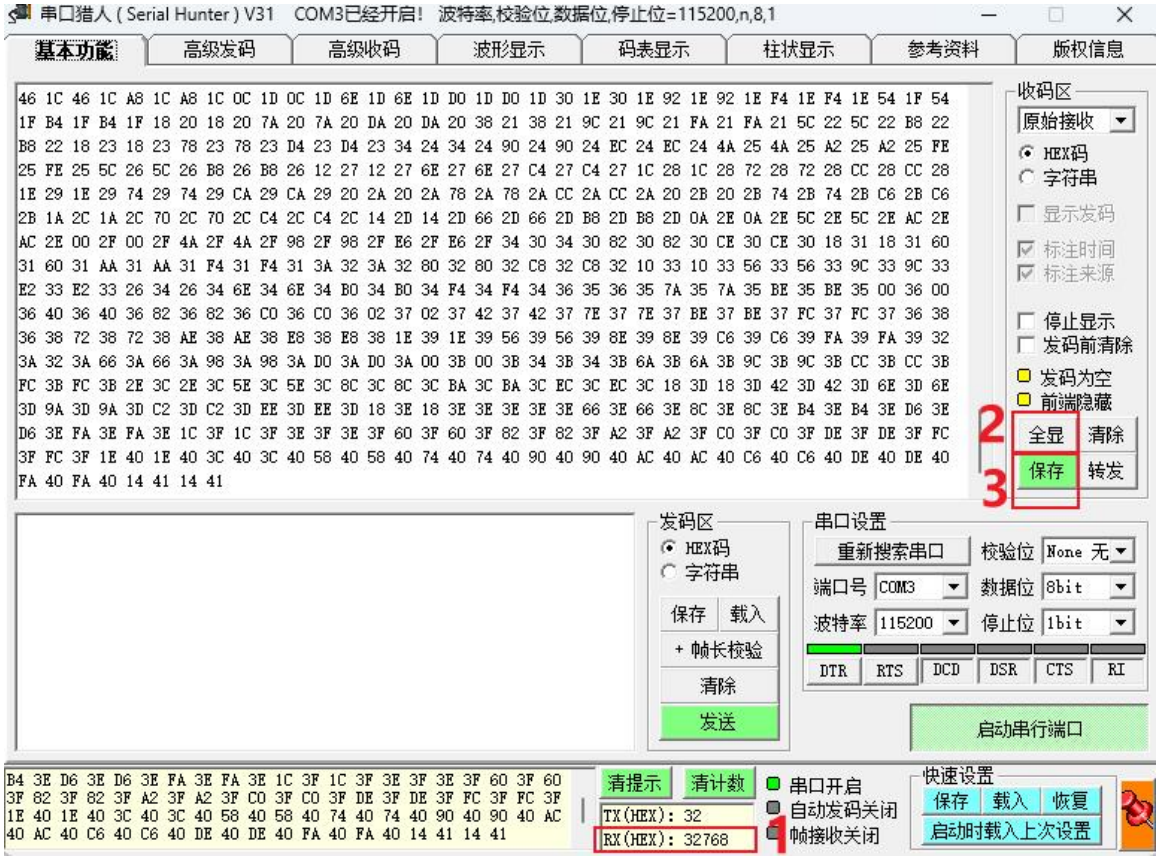
（3）55 A5 03 00 00 00 00 F0

（4）55 A5 00 00 00 00 00 F0

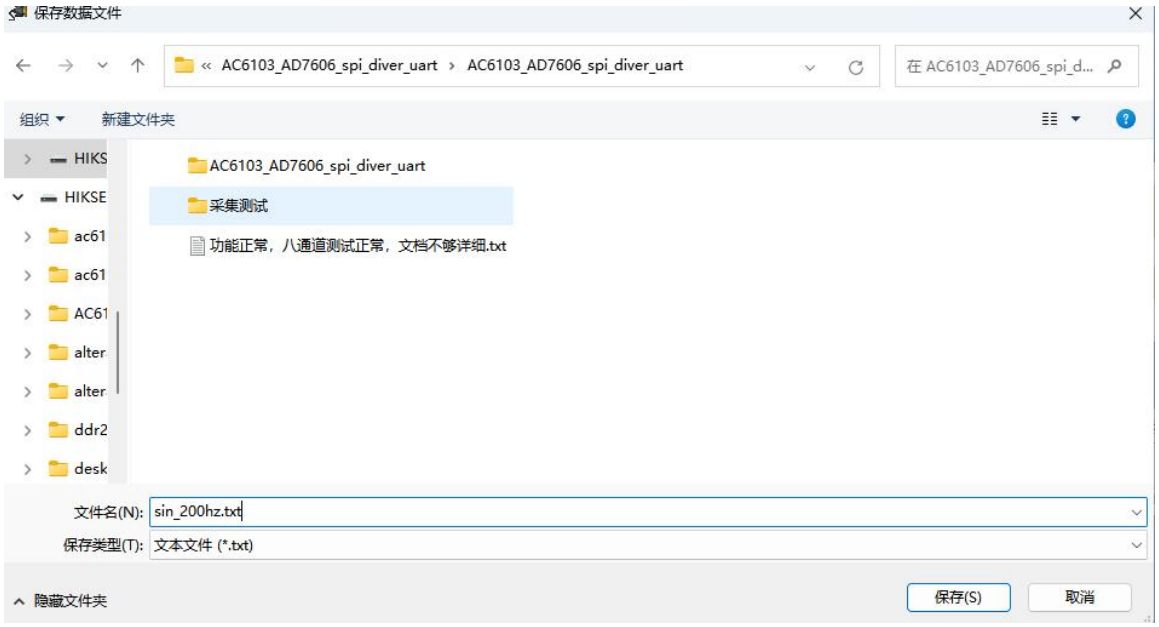
2、设置循环次数为 1

3、启动自动发码

图 1-20 串口猎人设置图 2



可以看到上图接收到的数据是 32768。然后我们点击全显按钮，接着点击保存按钮。然后设置保存的路径以及文件名。



接着我们打开 ADCdata_to_wave_v1_1.m 程序，接着修改文件的路径。

```
%==========================================================================%
%设置ADC采集数据有效位宽
ADC_DATA_WIDTH = 16;
%ADC采集电压范围±VOL_RANGE (V)
VOL_RANGE = 5;
%采样率(Hz)
SAMPLE_RATE = 5000000;
%ADC采集电压分辨率 LSB
LSB = VOL_RANGE*2/(2^ADC_DATA_WIDTH);
%==========================================================================%

%读取文件数据，转换成波形显示
src_data=textread('H:\desk\AC6103_AD7606_spi_diver_uart\AC6103_AD7606_spi_diver_uart\sin_200hz.txt','%s');
src_data_hex=hex2dec(src_data)
DATA_NUM = length(src_data_hex);
voltage_code = 1:DATA_NUM/2;
voltage_code = voltage_code';
for i=1:1:DATA_NUM/2
  if src_data_hex(2*i-1)+src_data_hex(i*2)*256 > 2^(ADC_DATA_WIDTH-1)
    voltage_code(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256-2^ADC_DATA_WIDTH;
  else
    voltage_code(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256;
  end
end

%二进制ADC数据转换为电压值
voltage = voltage_code*LSB - LSB/2;
%ADC采样时间
sample_t = 1:DATA_NUM/2;
sample_t = sample_t';
```

然后我们点击运行节（运行）按钮。

下图 1-21 为 MATLAB 实验结果图。

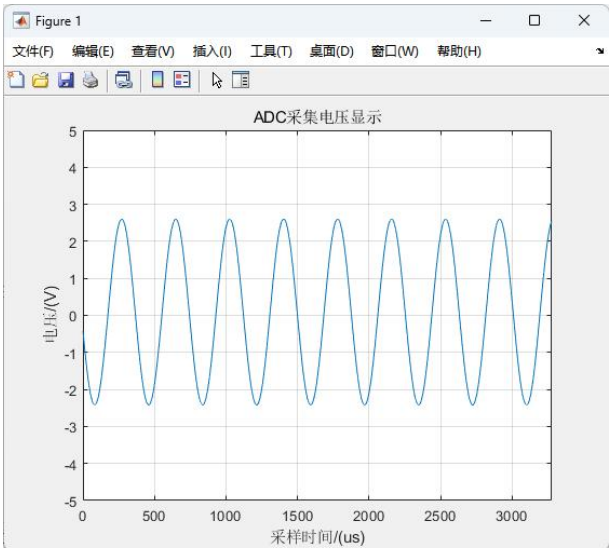
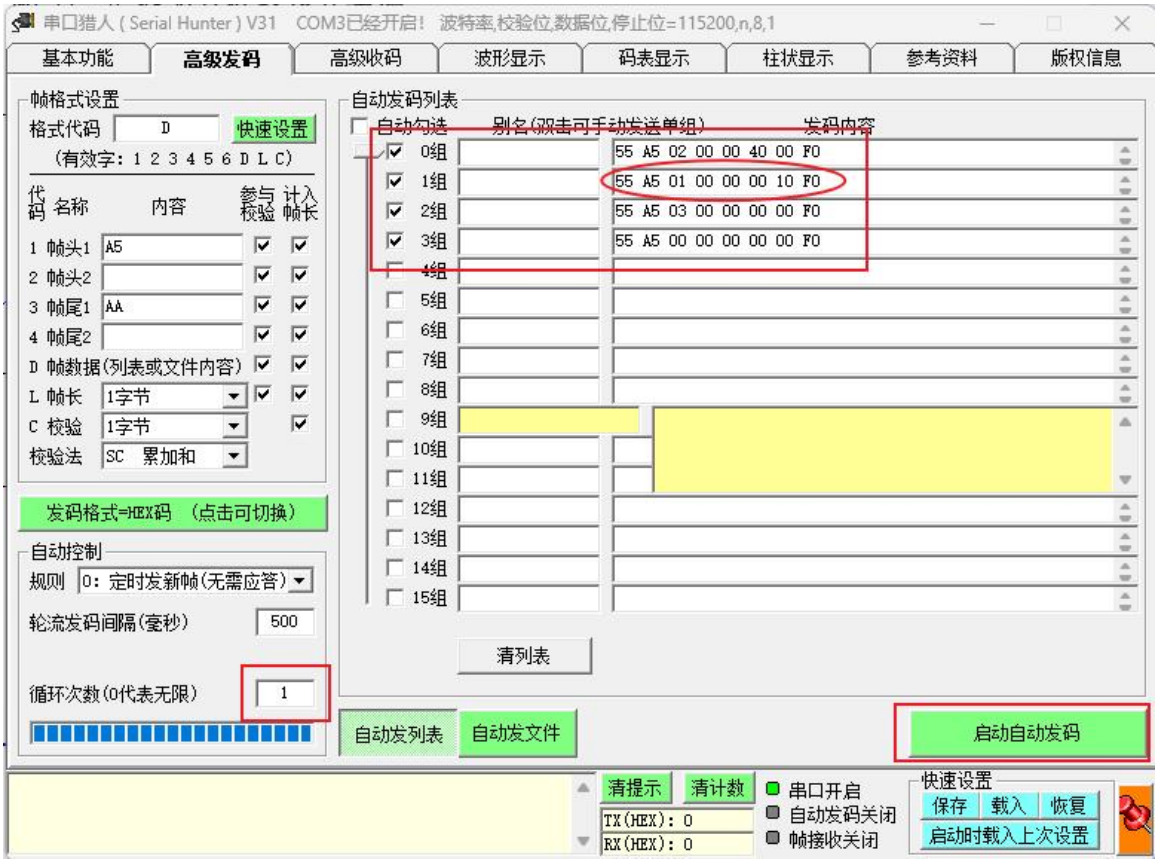


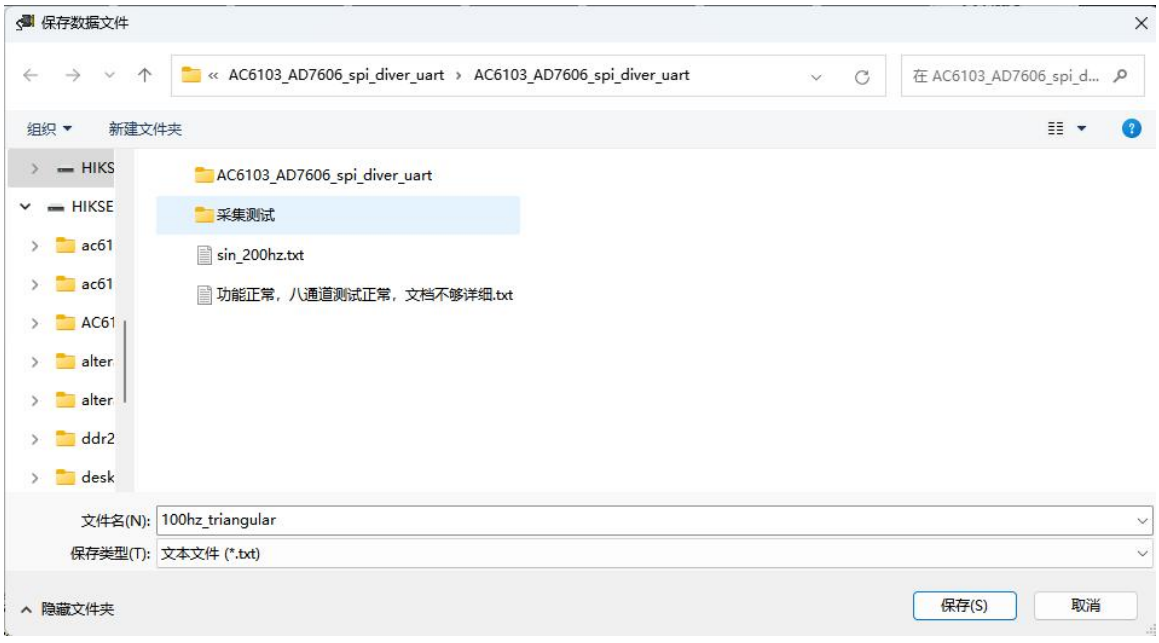
图 1-21 MATLAB 实验结果

接着我们将波形改为三角波，通道改成 5，频率改成 100hz，再来看看实验效果。在发送指令前请将串口调试助手前一次收到的数据清空，防止 MATLAB 绘图的时候把前一次的数据也绘制出来。这里点击清除按钮即可。

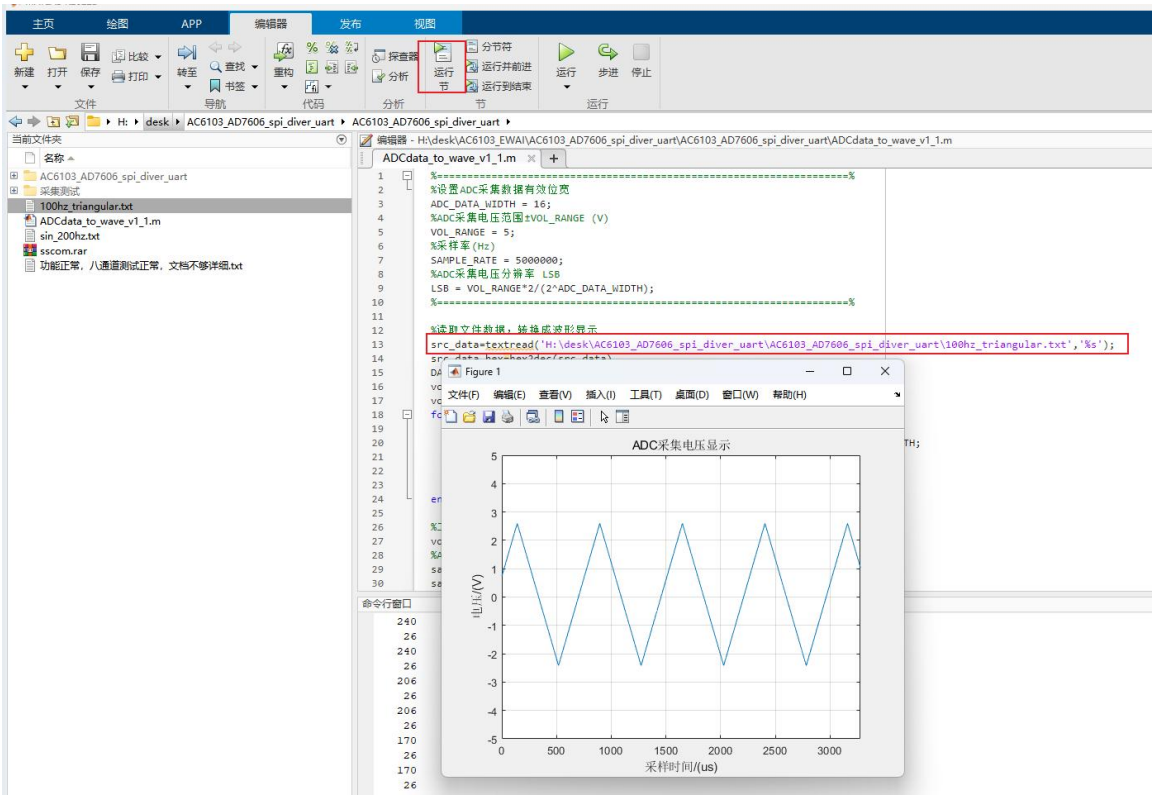
接着我们继续发指令，如下图所示。



接着我们将接收到的数据进行保存，然后用 MATLAB 进行绘制。



然后修改文件的路径，接着点击运行节按钮，效果如下图所示。



1.5 思考与总结

本次实验介绍了基于 AC6103 的 SPI 驱动 AD7606 的数据采集，用户通过串口调试助手发送指令来设置 AD7606 的寄存器，以此来控制 ADC 进行采样，并将数据缓存后再通过串口发送至电脑，然后通过 MATLAB 来验证。