1 基于 USB3.0 的 AD7606C 数据采集 DDR3 存储系统（串行驱动）

工程源码	|-ACX750_CH569 开发板 |-- ACX750_CH569_100T_USB_7606C_SOFT_8DOUT_SPI
	如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

章节导读

本节实验将介绍基于 USB3.0 的 AD7606C 模块进行数据采集系统设计的相关内容，设计使用 FPGA 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 AD7606C 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，AD7606C 开始采集数据，并将采集的数据存储至 DDR3 中。

再由 USB3.0 将 DDR3 中的数据传输至电脑，用户可以在电脑上通过 bus hound 进行指令的下发，以及通过 bus hound 将数据进行保存，然后使用上位机软件进行绘图，或者直接通过上位机进行指令的下发以及数据的接收，并将采到的数据进行绘图。

1.1 系统整体设计

通过 bus hound 或者数据采集上位机，将命令帧进行发送，然后 CH569 芯片的端点二接收数据，FPGA 将端点二的数据进行读取，并解析出指令，从而实现对 AD7606C 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，AD7606C 开始采集数据，并将 AD7606C 采集的数据存储进 DDR3 中。接着将 DDR3 中存储的数据通过 USB 传输至电脑，最后通过数据采集上位机显示采集到的波形，系统的整体框架如下图 1-1 所示。

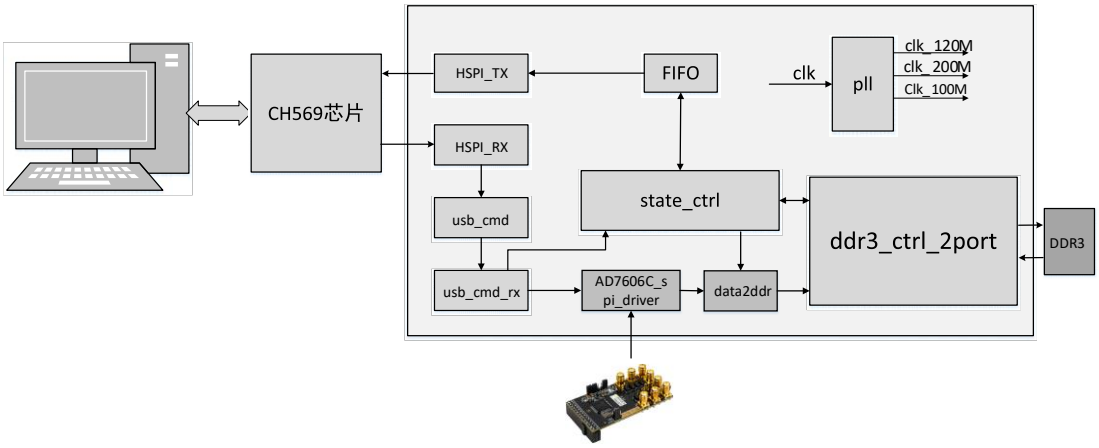


图 1-1 串口传输 AD7606C 数据采集整体设计框图

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，输入时钟 50M，由开发板上的晶振提供，输出 120M 供 USB 模块和数据采集控制模块使用，输出 200M 供 DDR 控制器使用，输出 100M 供 AD7606C 驱动模块使用。

2、HSPI_RX 模块：USB 数据接收模块，接收上位机下发的控制指令。

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：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、ad7606c_spi_driver 模块：AD7606C 控制器串行驱动模块，该控制器实现了对 AD7606C 型 8 通道 16 位 ADC 的数据转换控制并输出。使用该控制器时，用户无需关心 AD7606C 的具体控制时序，一切都在控制器内部完成，用户只需要像使用并行 ADC 一样取用数据即可。

6、data2ddr 模块：ADC 采集控制模块，AD7606C 在配置好采样频率后进行采样，采集的八通道数据由 ad7606c_spi_driver 输入该模块，并通过该模块写进 DDR 中。

7、ddr3_ctrl_2port 模块：DDR3 双端口模块，用来缓存 AD7606C 采集到的数据。

8、state_ctrl 模块：产生 DDR3 双端口模块所需要的控制信号以及产生写进 FIFO 的控制信号。

9、fifo 模块：USB 发送 FIFO，从 DDR3 中读取数据进入该 FIFO 中，然后从该模块中读出交由 HSPI TX 模块发送出去。

10、HSPI TX 模块：数据发送模块，将 ADC 采集到的数据通过 HSPI 接口

发送出去。

1.2 ACM7606C 模块简介

ACM7606C 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 AD7606C，模块图如下图 1-2 所示。

图 1-2 AD7606C 模块图

AD7606C 是一款具有八通道的 16 位同时采样模拟到数字数据采集系统（DAS）。每个通道包含模拟输入箝位保护、可编程增益放大器（PGA）、低通滤波器（LPF）和 16 位逐次逼近寄存器（SAR）模数转换器（ADC）。还包含灵活的数字滤波器、低漂移 2.5 V 精密参考电压源、驱动 ADC 的参考缓冲器以及灵活的并行和串行接口。

同时所有通道均能以高达 1MSPS 的吞吐速率采样。输入箝位保护电路可以耐受最高达±21V 的电压。无论以何种采样频率工作，其模拟输入阻抗均为 1MΩ。这是一个固定的输入阻抗，不随 AD7606C 采样频率而变化。这种高模拟输入阻抗消除了在 AD7606C 前面的驱动器放大器的需要，允许直接连接到源或传感器。因此，双极电源可以从信号链上移除。

芯片对外提供 SPI 和并行的数字接口。当 AD7606C 的 8 个通道全部以 1MSPS 的最高速率进行转换时，数据输出速率达到 128Mbps，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 AD7606C 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 AD7606C 时，可以通过在 FPGA 上设计 AD7606C 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

AD7606C 的功能框图如下图 1-3 所示。

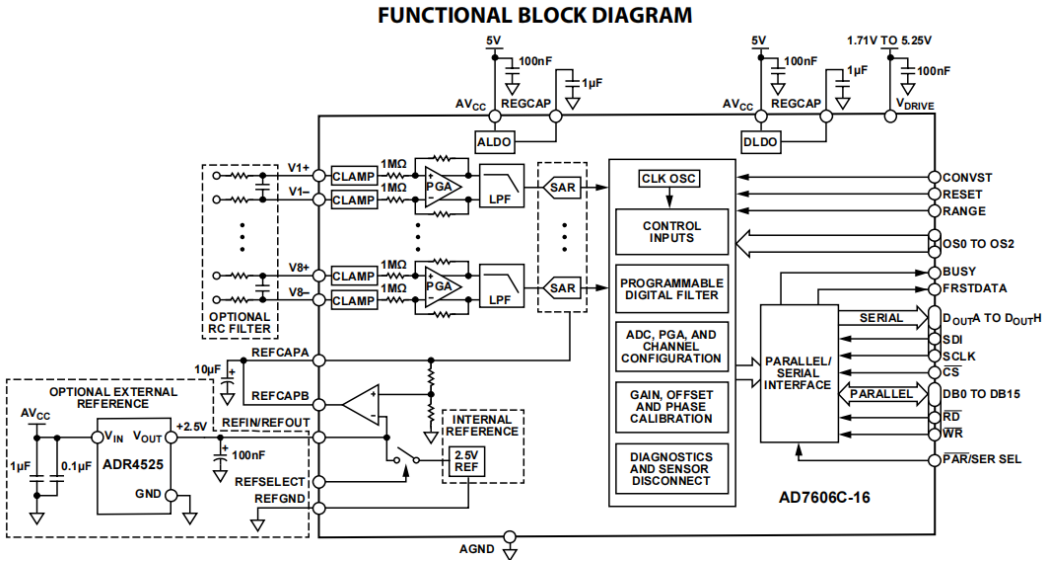


图 1-3 AD7606C 功能框图

如上图所示，当数据以串行模式输出时，数据会从 DoutA~DoutH 中输出，到底使用几根 DOUT 线进行输出，取决于寄存器的配置，数据可以通过 1 根、2 根、4 根、8 根 DOUT 线输出，如下图所示。

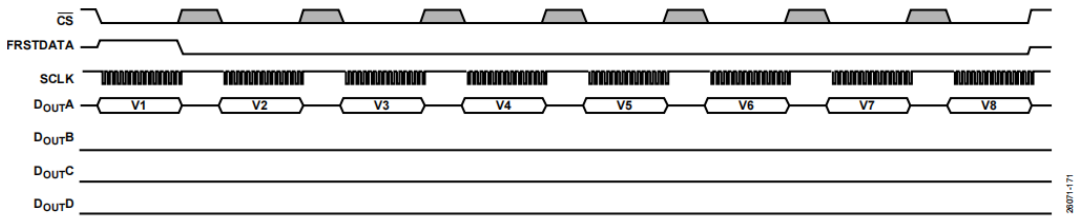


Figure 107. Serial Interface ADC Reading, One DOUTx Line

图 1-4 通过 1 根 DOUT 线进行输出

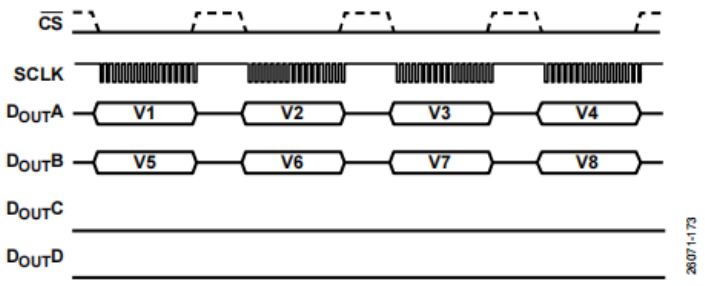


Figure 104. Serial Interface ADC Reading, Two DOUTx Lines

图 1-5 通过 2 根 DOUT 线进行输出

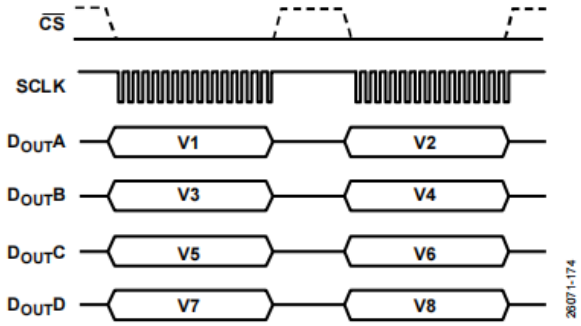


Figure 105. Serial Interface ADC Reading, Four DOUTx Lines

图 1-6 通过 4 根 DOUT 线进行输出

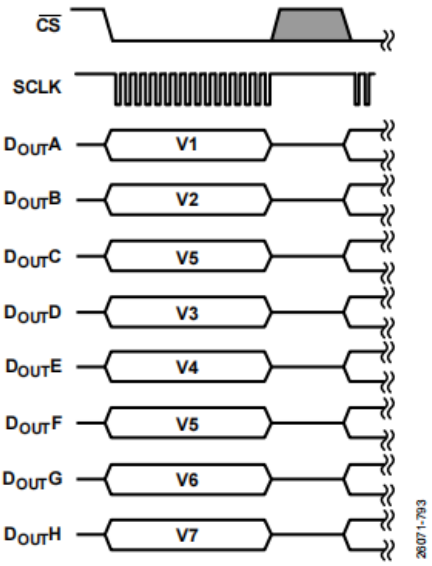


Figure 106. Serial Interface ADC Reading, Eight DOUTx Lines

图 1-7 通过 8 根 DOUT 线进行输出

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

AD7606C 可处理双极单端、单极单端。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。AD7606C 的模拟输入阻抗为 1MΩ。这是固定输入阻抗，不随 AD7606C 采样频率而变化。这种高模拟输入阻抗消除了在 AD7606C 前面的驱动器放大器的需要，允许直接连接到源或传感器。因此，双极电源可以从信号链上移除。AD7606C 的输入结构如下图 1-8 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±21V。

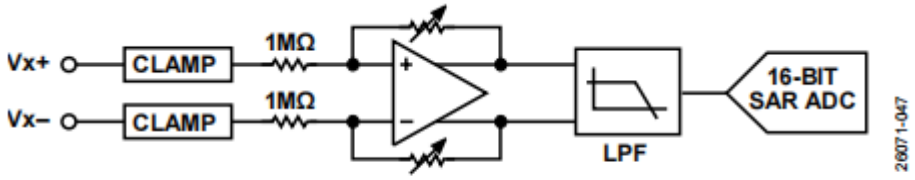


Figure 74. Analog Input Circuitry for Each Channel

图 1-8 AD7606C 模拟输入结构

1.2.3 数字滤波器与过采样

AD7606C 包含一个可选的数字平均滤波器，可以在需要更高信噪比或动态范围的较慢的吞吐量速率应用程序中启用。在硬件模式下，使用过采样引脚 OSx 来控制数字滤波器的过采样比，如下表 1-1 所示。OSx 引脚被锁定在 BUSY 信号的下降边缘或完全返回上。

表 1-1 可选引脚解码

OS2	OS1	OS0	过采样比
0	0	0	无过采样
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	进入软件模式

在软件模式下，如果所有 OSx 引脚都与逻辑高，则通过过采样寄存器（地址 0x08）选择过采样比。在软件模式下，还有两个额外的过采样比（128 过采样和 256 过采样）。

在过采样模式下，ADC 对 CONVST 信号的上升边缘上的每个通道进行第一个采样。转换第一个样本后，由内部生成的采样信号采集后续样本，如图 1-9 所示。

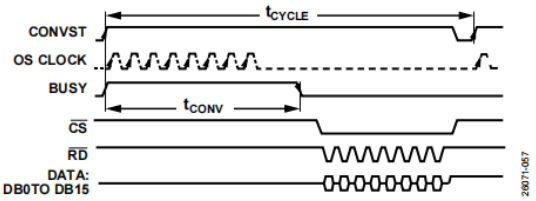


Figure 87. AD7606C-16 Oversampling by 8 Example, Read After Conversion, Parallel Interface, OS_CLOCK Is the Internally Generated Sampling Signal

图 1-9 AD7606C 过采样 8 为例

例如，如果配置了 8 个过采样，则取 8 个样本进行平均，并在输出上提供结果。CONVST 信号上升边缘触发第一个样本，其余 7 个样本用内部生成的采

样信号（OS_CLOCK）采集。因此，开启多个样本的平均功能会以降低最大吞吐量为代价来提高信噪比性能。当过采样函数打开时，BUSY 信号高时间（tCONV）会扩展。

当打开过采样时，转换时间（tCONV）会延长。必须降低吞吐量速率（1/tCYCLE），以适应更长的转换时间，并允许读取操作发生。为了在打开过采样时实现尽可能快的最快吞吐量速率，读取可以在 BUSY 信号高时间内执行读取，如在转换期间的读取部分中所述。

下表 1-2 和表 1-3 提供了用来选择不同的带宽模式以及不同的过采样倍率的过采样位解码。

表 1-2 低带宽不同过采样倍率的过采样位解码

OS [2:0]	过采样倍率	10V 范围 SNR(dB)	10V 范围 3dB 带宽 (kHz)	20V 范围 SNR(dB)	20V 范围 3dB 带宽 (kHz)	0V 到 10V 范围 SNR(dB)	0V 到 10V 范围 3dB 带宽(kHz)	最大吞吐量 CONVST 频率(kSPS)
000	No OS	91.5	25	92	25	89	25	1000
001	2	92.5	24.6	93	24.4	90	24.6	500
010	4	94.5	24	95	23.7	91	24	250
011	8	95	22.3	96	22.2	91.7	22.3	125
100	16	96	17.8	96.5	17.6	92.5	17.8	62.5
101	32	96.5	11.6	97	11.5	93	11.6	31.25
110	64	97	6.5	97.5	6.4	94.5	6.4	15.6

表 1-3 高带宽不同过采样倍率的过采样位解码

OS [2:0]	过采样倍率	10V 范围 SNR(dB)	10V 范围 3dB 带宽 (kHz)	20V 范围 SNR(dB)	20V 范围 3dB 带宽 (kHz)	0V 到 10V 范围 SNR(dB)	0V 到 10V 范围 3dB 带宽(kHz)	最大吞吐量 CONVST 频率(kSPS)
000	No OS	86	220	88	220	81	220	1000
001	2	89	154	91	154	84	155	500
010	4	91	97.5	93	97.5	86.5	97.5	250
011	8	92.5	53	94.5	53	88.5	53.5	125
100	16	95	27.5	96	27.5	90.5	27.5	62.5
101	32	96	13.8	97	13.7	92	13.5	31.25
110	64	97	7	97.5	7	93.5	7	15.6

1.2.4 工作时序

AD7606C 根据采样方式不同具有多种驱动时序，本次实验采用的为串行输出（使用 8 根 DOUT 线同时输出 8 个通道的数据），所以先要进行寄存器的配置。时序图由三部分组成：写寄存器时序、完成 AD 转换和读取 AD 数据。时序图如下所示。

Universal Timing Diagram

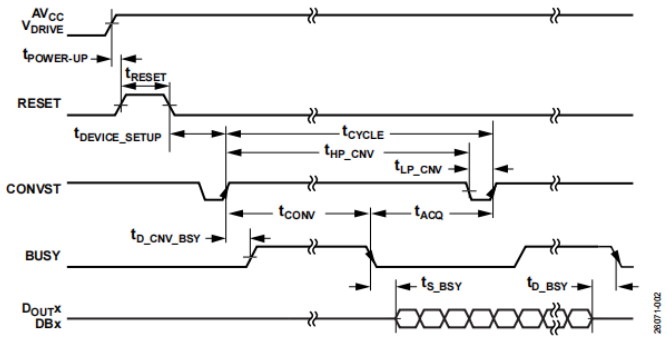


Figure 2. Universal Timing Diagram

图 1-10 通用时序

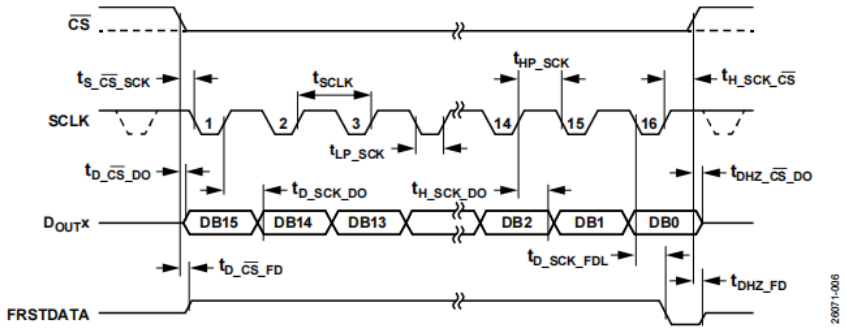


Figure 6. Serial Timing Diagram, ADC Mode (Channel 1)

图 1-11 通道 1 串行时序

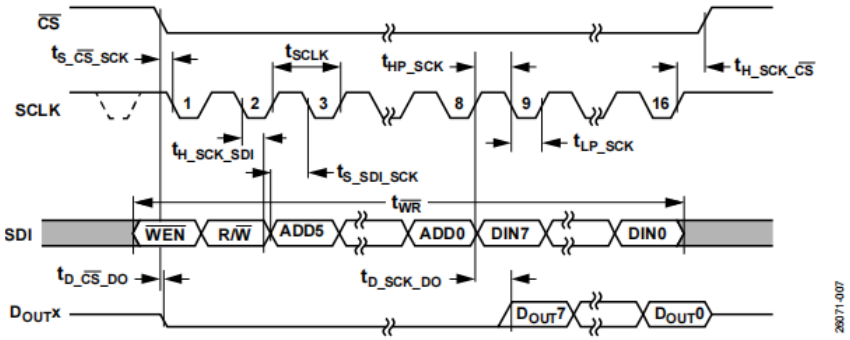


Figure 7. Serial Timing Interface, Register Map Read and Write Operations

图 1-12 寄存器串行模式

当 CONVST 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。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。

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。

1.3.1 HSPI_RX 模块

HSPI 接收模块，FPGA 将 HSPI 接口传输过来的数据进行接收，也就是接收 CH569 端点 2 的数据。系统框图如下所示，模块信号说明如下所示。

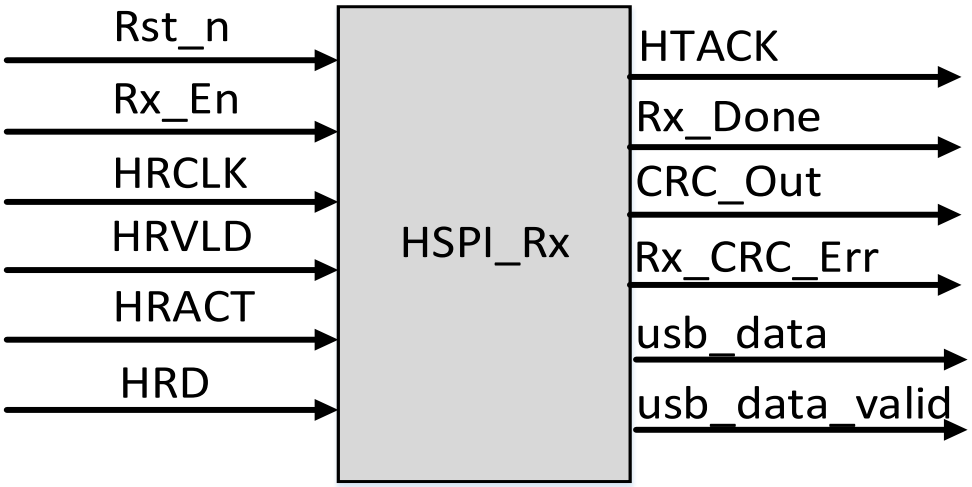


图 1-13 模块接口图

表 1-4 模块信号说明表

信号	I/O	描述
Rst_n	I	系统复位
Rx_En	I	接收使能信号
HRCLK	I	输入采样时钟，作为接收数据的采样时钟基准
HRVLD	I	输入数据接收状态
HRACT	I	输入发送请求信号，有效后可驱动 HTACK 信号
HRD	I	输入的 32 位数据
usb_data	O	输出接收到的数据
usb_data_valid	O	输出数据的有效信号标志
HTACK	O	输出准备接收状态信号，高电平有效
Rx_Done	O	接收完成信号
CRC_Out	O	CRC 校验值
Rx_CRC_Err	O	CRC 校验错误输出标志

1.3.1.1 代码设计

我们使用状态机的方式实现该模块的功能，定义状态如下所示，分别为空闲状态、接收包头状态、接收数据包状态、结束状态。

```
localparam  S_IDLE = 4'b0001;
localparam  S_HEAD = 4'b0010;
localparam  S_DATA = 4'b0100;
localparam  S_TAIL = 4'b1000;
```

整体的状态转移图如下图所示。

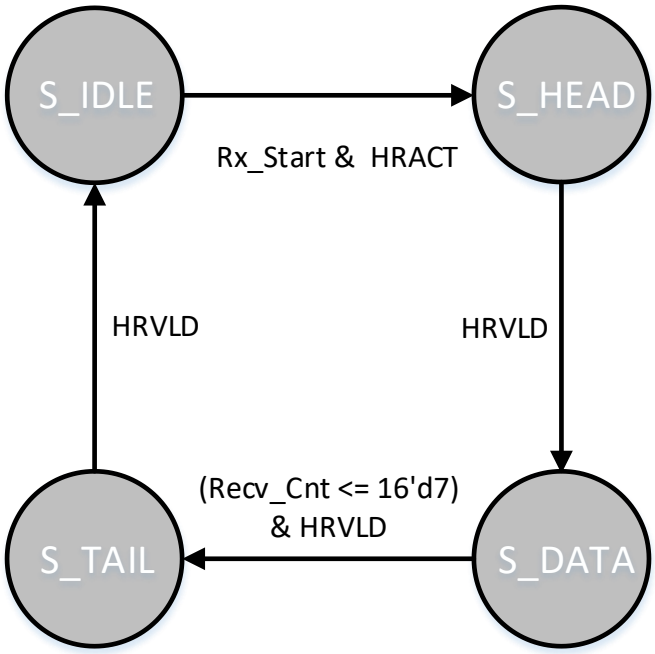


图 1-14 状态转移图

下面将对每个状态的代码设计进行简单的说明。

1、S_IDLE 状态

在 S_IDLE 状态的时候，等待接收开始信号拉高，此时如果检测到 HRACT 信号为高的时候，将 HTACK 信号拉高，并且状态开始跳转，代码如下所示。

```
S_IDLE: begin
    HTACK <= 1'b0;
    Rx_Done <= 1'b0;
    if(Rx_Start) begin
        if(HRACT_R) begin
            HTACK <= 1'b1;
            SM_State <= S_HEAD;
```

```
        end
    end
    else
        SM_State <= S_IDLE;
end
```

接收开始信号什么时候开始拉高呢？这里通过一个使能信号进行控制，只要状态机处于 S_IDLE 状态，并且使能信号拉高的时候就认为这是一次开始传输信号，代码如下所示。

```
always @(posedge HRCLK or negedge Rst_n)
begin
    if(~Rst_n)
        Rx_Start <= 1'b0;
    else if(Rx_En && SM_State == S_IDLE)
        Rx_Start <= 1'b1;
    else if(SM_State == S_HEAD)
        Rx_Start <= 1'b0;
    else
        Rx_Start <= Rx_Start;
end
```

2、S_HEAD 状态

在 S_HEAD 状态的时候，检测到 HRVLD 信号为高的时候，状态机开始跳转，此时接收到的数据就是包头，代码如下所示。

```
S_HEAD: begin
    if(HRVLD_R) begin
        Head_Data <= HRD_R;
        SM_State <= S_DATA;
    end
    else
        SM_State <= S_HEAD;
end
```

3、S_DATA 状态

在 S_DATA 状态的时候，开始接收数据，在接收到 32 个字节的数据后，状态机进行跳转，因为对于本次实验来说，指令长度为 32 个字节，所以只接收 32 字节的数据，代码如下所示。

```
S_DATA: begin
    if((Recv_Cnt <= 16'd7) & HRVLD_R) begin
```

```
        Recv_Cnt <= Recv_Cnt + 1'b1;
        usb_data <= HRD_R;
        usb_data_valid <= 1'b1;
         if(Recv_Cnt == 16'd7)
            SM_State <= S_TAIL;
    end
    else begin
         SM_State <= S_DATA;
         usb_data_valid <= 1'b0;
         Recv_Cnt <= Recv_Cnt;
    end
end
```

4、S_TAIL 状态

在 S_TAIL 状态的时候，如果检测到 HRVLD 信号为高电平，接收计数器清零，状态机开始跳转，拉高接收完成标志，代码如下所示。

```
S_HEAD: begin
    if(HRVLD_R) begin
        Rx_Done <= 1'b1;
        usb_data_valid <= 1'b0;
        usb_data <= 32'h0000_0000;
        Recv_Cnt <= 16'd0;
        Tail_Data <= HRD_R;
        SM_State <= S_IDLE;
    end
end
```

1.3.2 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图所示。

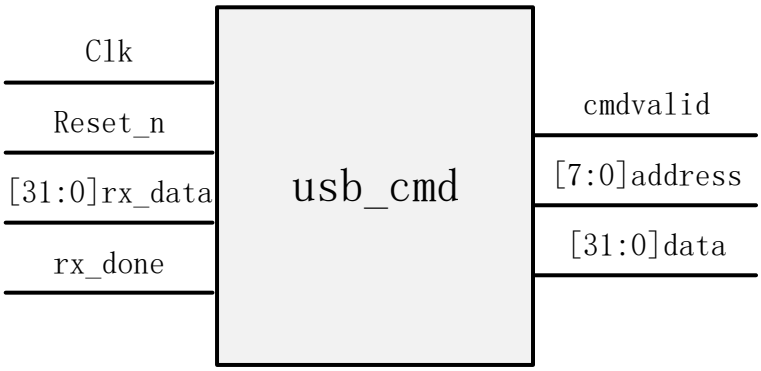


图 1-15 接收转命令模块框图

模块信号说明如下所示

表 1-5 模块信号说明表

信号	I/O	描述
Clk	I	模块工作时钟
Reset_n	I	模块复位信号，低电平复位
rx_data	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
address	O	配置 AD7606C 的寄存器地址信号
data	O	写入到寄存器中的数据
cmdvalid	O	输出命令有效标志信号

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 1-6 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 AD7606C 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] <=  rx_data;
    data_str[0] <=  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 <=  0;
    data <=  0;
    cmdvalid <=  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] <=  data_str[1][23:16];
        data[15:8] <=  data_str[1][15:8];
        data[23:16] <=  data_str[1][7:0];
        data[31:24] <=  data_str[0][31:24];
        address <=  data_str[0][23:16];
        cmdvalid <=  1;
    end
    else
        cmdvalid <=  0;
end
else
       cmdvalid <=  0;
```

1.3.3 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 1-7 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	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为 1，则该通道的采样结果就会被存入 FIFO 并通过串口发送，注意对应的 2 进制的位，不是 10 进制，比如设置通道 3，对应 01 地址需要写入的值为 04(100)，而不是 03(011)
set_sample_num	2	32	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， 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
set_sample_speed	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 1Msps，所以可以通过设置该寄存器的值来让 ADC 的采样速率在 1~1Msps 范围内调整，以适应不同的应用场景。 ADC_Speed_Set=1000000000/10/speed-1，其中 speed 就是实际要设置的采样速率。

指令转控制模块的结构框图如下图所示。

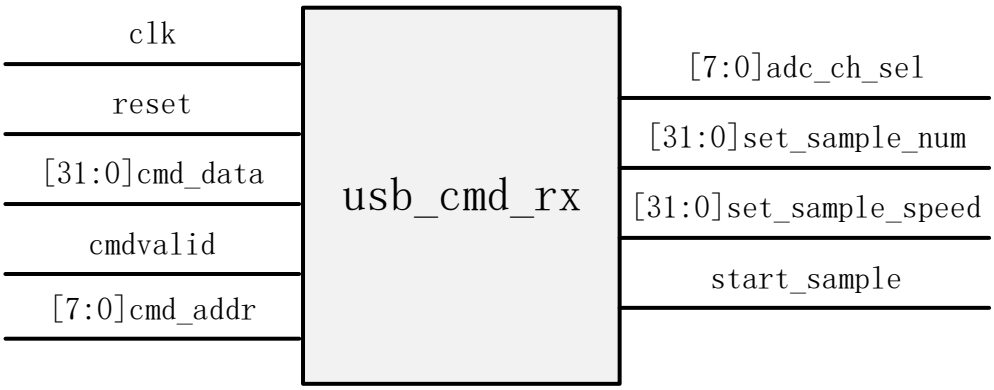


图 1-16 指令转控制模块结构框图

模块信号说明如下所示。

表 1-8 指令转控制模块信号说明表

信号	I/O	描述
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 [7:0]	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 <= 8'b11111111;
    set_sample_num <= 32'd256;
    start_sample <= 1'b0;
    set_sample_speed <= 32'd0;
end
else if(cmdvalid)begin
    case(cmd_addr)
        0: start_sample <= 1'b1;
        1: adc_ch_sel <= cmd_data[7:0];
        2: set_sample_num <= cmd_data[31:0];
        3: set_sample_speed <= cmd_data[31:0];
        4:
        begin
            adc_ch_sel <= cmd_data[7:0];
            set_sample_num <= cmd_data[23:8];
            start_sample <= 1'b1;
        end
    default:;
    endcase
end
else
    start_sample <= 1'b0;
```

1.3.4 AD7606C 控制器驱动模块

AD7606C 控制器驱动模块 ad7606c_spi_driver，该控制器实现了对 AD7606C 型 8 通道 16 位 ADC 的数据转换控制并输出。使用该控制器时，用户无需关心 AD7606C 的具体控制时序，一切都在控制器内部完成，用户只需要像使用并行 ADC 一样取用数据即可。该模块的结构框图如下所示。

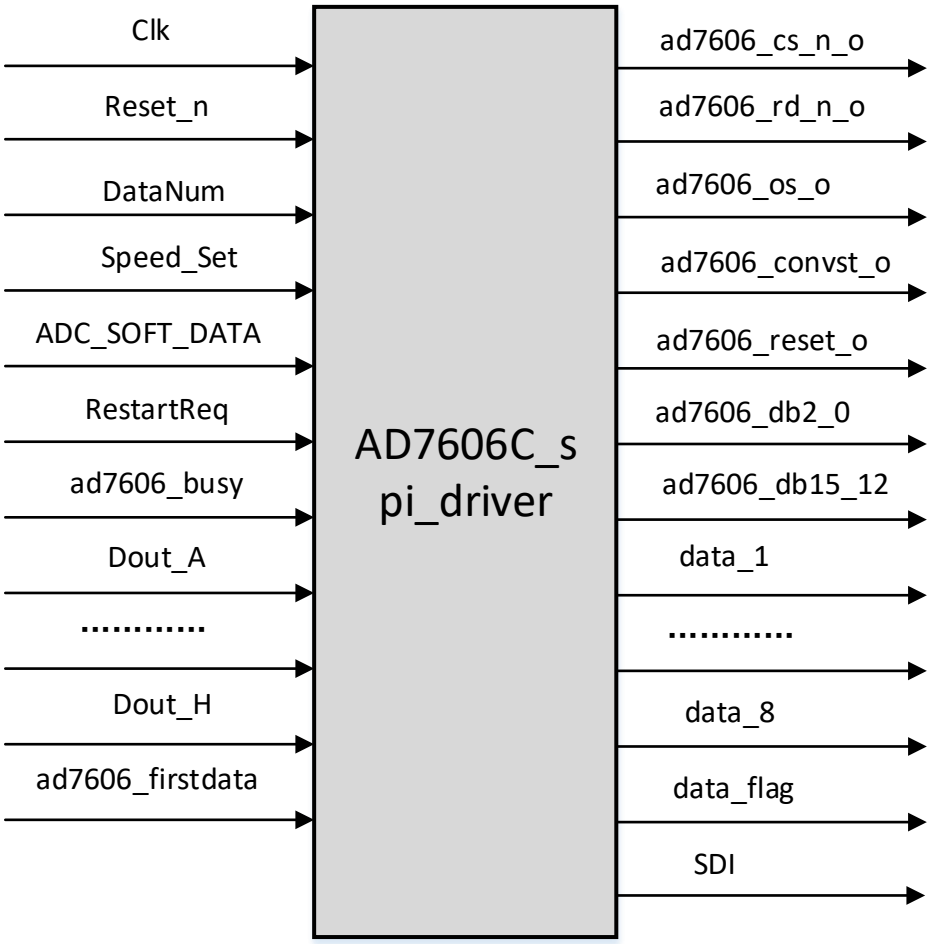


图 1-17 AD7606C 控制器串行驱动模块结构框图

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 AD7606C 控制器与 AD7606C 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。对于该模块的信号说明如下所示。

表 1-9 AD7606C 控制器模块信号说明表

类	信号名称	I/O	信号意义
系统信号	Clk	I	模块系统时钟， 这里使用 100M 时钟
	Reset_n	I	模块复位信号，低电平复位
控制信号	DataNum	I	采样数量控制端口
	Speed_Set	I	采样速率控制端口
	ADC_SOFT_DATA	I	寄存器配置端口
	convst_done	O	一次采样完成标志信号，单时钟周期脉冲信号。每 8 个通道结

			果都输出后，产生一个高脉冲信号
数据结果输出端口和标志信号	data_flag[7:0]	O	转换结果有效标志信号，由于使用 8 根 dout 线同时输出 8 个通道的数据，所以设置 8 个 Flag 信号，每个通道对应 Flag 信号的对应位
	data1[15:0]…data8[15:0]	O	8 个通道的采样结果输出端口，每个端口分别对应一个模拟通道的采样结果，使用时与 data_flag 信号配合，每当 data_flag 中的某一位为 1 时，则对应的通道上的 16 位采样结果已经就绪，可以使用。
ADC 芯片控制信号	ad7606_busy	I	ad7606C 转换状态标志信号，为高电平则表明 ad7606C 当前仍处于转换状态，结果没有更新，如果此时读取，读取的结果就还是前一次的采样转换结果。需要待该信号变为低电平之后，再读取 ad7606C 中的数据
	Dout_A…Dout_H	I	使用串行接口时，DB7/DOUTA 引脚作为 DOUTA 功能，B8/DOUTB 引脚作为 DOUTA 功能，详细的可参考 7606C 手册
	ad7606_cs_n_o	O	ad7606C 芯片选中控制信号，可以从 AD7606C 中读取转换结果时，需要使该信号为低电平
	ad7606_rd_n_o	O	选择串行接口时的串行时钟输入（SCLK）
	ad7606_os_o[2:0]	O	过采样模式引脚。OS0 至 OS2 选择过采样比率或启用软件模式
	ad7606_reset_o	O	ad7606C 的复位信号，复位 ad7606C 内部各个功能单元的工作状态
	ad7606_convst_o	O	ad7606C 转换开始信号，该信号的上升沿启动 ad7606C 内部的采样转换逻辑开始新一轮的采样转换
	ad7606_db2_0	O	在使用串行接口时，DB0~DB2 不使用
	ad7606_db15_12	O	在使用串行接口时，DB15~DB12 不使用
	SDI	O	在软件模式下，通过 SDI 配置 7606C 相关的寄存器

该控制器在工作时会根据主机的指令对采样频率进行修改，当信号转换完成后便对 ADC 写控制器发出 data_flag 信号，控制器将转换完成对应通道的数据写进 FIFO 或者 RAM。

1.3.5 ADC 采集控制模块设计

ADC 采集控制模块（data2ddr）的结构框图如下图所示。将采集到的数据通过 data2ddr 模块写进 DDR3 中。

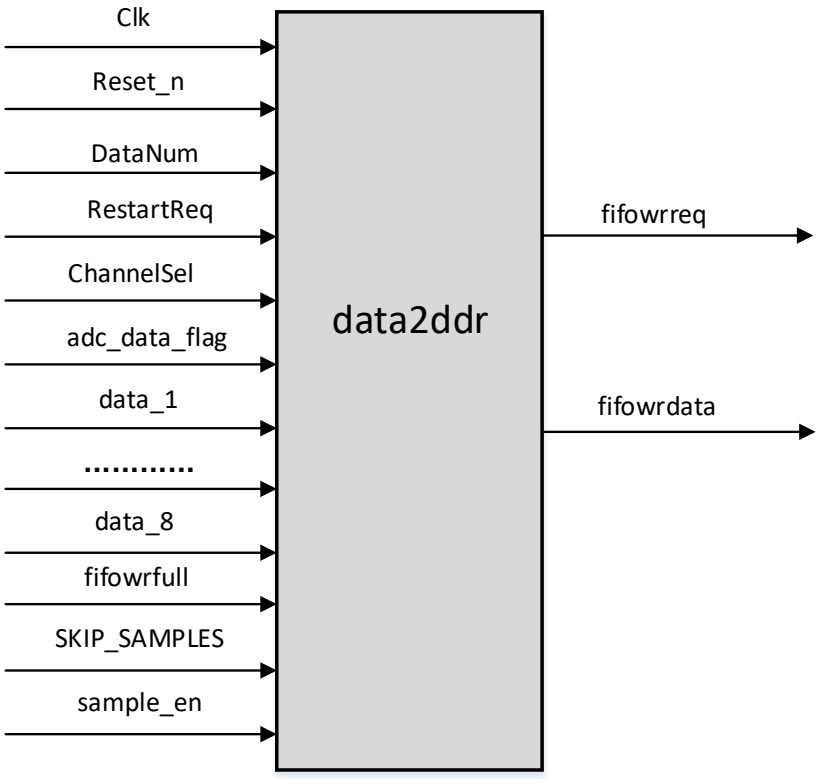


图 1-18 ADC 采集控制模块结构框图

对于该模块的信号说明如下表 1-10 所示。

表 1-10 ADC 采集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
DataNum [31:0]	I	单次采集的数据个数，最大不超过 FIFO 的深度
RestartReq	I	开始采样请求信号
ChannelSel [7:0]	I	需要采样的通道选择 ，每一位对应一个通道的开关
fifowrfull	I	FIFO 写满标志信号
adc_data_flag [7:0]	I	ADC 通道转换有效标志信号
data_1[15:0]…data_8[15:0]	I	数据输入端口
SKIP_SAMPLES	I	舍弃前 SKIP_SAMPLES 个数据
sample_en	I	采样使能信号
fifowrreq	O	采集到的数据写 FIFO 请求信号
fifowrdata[15:0]	O	采集到的写入到 FIFO 中的数据

接下来将对模块中的部分代码进行介绍。

首先设置延迟计数器，根据延迟计数器控制拉高的 FIFO 的写使能，代码如下所示：

```
    reg [1:0] skip_cnt;
    always @(posedge Clk or negedge Reset_n)
```

```
    if (!Reset_n)
        skip_cnt <= 4'd0;
    else if (RestartReq)
        skip_cnt <= 4'd0;
    else if (sample_en && (adc_data_flag & ChannelSel) &&
skip_cnt < SKIP_SAMPLES)
         skip_cnt <= skip_cnt + 1;
```

ADC 采集控制模块根据 adc_data_flag&ChannelSel 有效、sample_en 有效以及延迟计数器大于设定的值的情况下输出 fifowrreq 信号用于读取所需通道的数据，并将采集到的数据进行输出，通过通道索引值选择写入 FIFO 的数据，代码如下所示：

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

    always @(posedge Clk) begin
        case (channel_index)
           3'd0: fifowrdata <= data_1;
           3'd1: fifowrdata <= data_2;
           3'd2: fifowrdata <= data_3;
           3'd3: fifowrdata <= data_4;
           3'd4: fifowrdata <= data_5;
           3'd5: fifowrdata <= data_6;
           3'd6: fifowrdata <= data_7;
           3'd7: fifowrdata <= data_8;
           default: fifowrdata <= 16'd0;
        endcase
    end
```

1.3.6 state_ctrl 模块

状态控制模块 state_ctrl 的作用就是控制 ADC 的采集，以及向 DDR 读写数据等相关状态的控制，状态控制模块（state_ctrl）的结构框图如下所示。

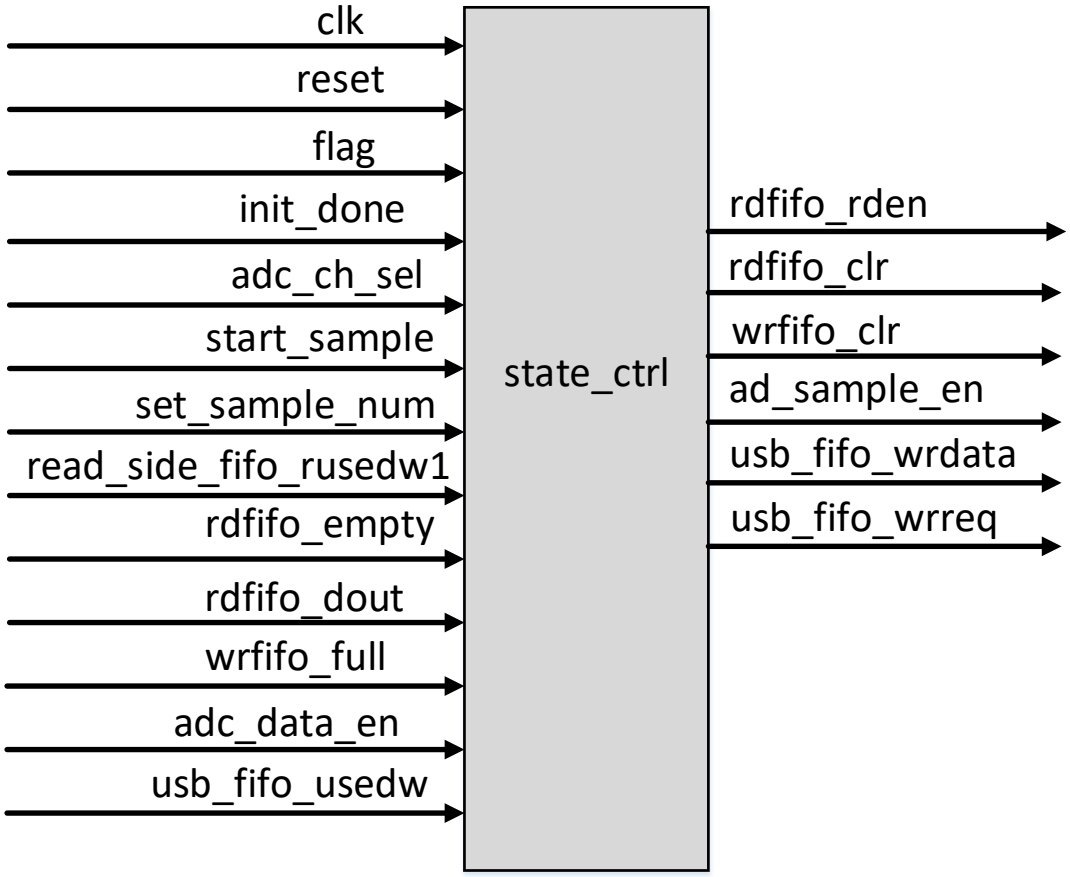


图 1-19 state_ctrl 模块基本结构框图

对该模块的信号说明如下所示。

表 1-11 state_ctrl 模块信号说明表

信号	I/O	描述
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	I	通道命令信号
start_sample	I	ADC 模块开始采样标志信号
set_sample_num	I	采样数量命令信号
read_side_fifo_rusedw1	I	FIFO 中可读的数量
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	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
usb_fifo_usedw	I	写入 FIFO 的数量
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
rdfifo_clr	O	DDR 控制读 FIFO 清除信号
wrfifo_clr	O	DDR 控制写 FIFO 清除信号
usb_fifo_wrdata	O	缓存 FIFO 的写请求信号
usb_fifo_wrreq	O	写入缓存 FIFO 的 16 位数据
ad_sample_en	O	输入的启动采样标志信号

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，跳转到 DDR_WR_FIFO_CLEAR 状态。空闲状态代码代码如下所示：

```
IDLE:
begin
    if(start_sample_rm)begin
        state<=DDR_WR_FIFO_CLEAR;
    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，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_cnt<=0;
else if(state==DDR_WR_FIFO_CLEAR)
begin
    if(wrfifo_clr_cnt == 9)
        wrfifo_clr_cnt<= 9;
    else
        wrfifo_clr_cnt<= wrfifo_clr_cnt + 1'b1;
end
else
    wrfifo_clr_cnt<= 0;
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，代码如下。

```
DDR_WR_FIFO_CLEAR:
begin
    if(!wrfifo_full && (wrfifo_clr_cnt==9))
        state<= ADC_SAMPLE;
    else
        state<=state;
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， ADC 采样个数计数器 adc_sample_cnt 开始计数，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_flag & adc_ch_sel)
           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的时候，ADC 模块数据采集完成，跳转到读 FIFO 清除状态，代码如下所示：

```
ADC_SAMPLE:
begin
if((adc_sample_cnt>=set_sample_num))
    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

当进入读 FIFO 清零状态后，做和前面写 fifo 清零相同的操作，发出三拍的

清零指令，同时保证一个 10 拍的基本延时，代码如下。

```
always@(posedge clk or posedge reset)begin
if (reset)
    rdfifo_clr_cnt<=0;
else if(state==DDR_RD_FIFO_CLEAR)
begin
    if(rdfifo_clr_cnt==9)
        rdfifo_clr_cnt<= 9;
    else
        rdfifo_clr_cnt<= rdfifo_clr_cnt + 1'b1;
end
else
    rdfifo_clr_cnt<= 0;
end
```

然后等待 rdfifo_empty 的信号拉低，拉低后，表示可以从 FIFO 里读出数据，此时进入下一个状态。

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

当处于 DDR_RD_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 rdfifo_clr_cnt 为 0、1 或 2 时，rdfifo_clr 置 1，否则 rdfifo_clr 为 0。代码如下所示：

```
always@(posedge clk or posedge reset)begin
if (reset)
    rdfifo_clr<=0;
else if(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、DATA_SEND_START 状态

进入到 DATA_SEND_START 状态的时候，状态机也进行跳转，代码如下。

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

6、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 4096 的时候，并且读 FIFO 中可读数量大于 16 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，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 < 4096)&& (read_side_fifo_rusedw1 > 16))
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，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
```

1.3.7 HSPI_TX 模块

HSPI 发送模块，通过 HSPI 接口将数据上传至 CH569 的端点 1。系统框图如下图 1-20 所示，模块信号说明如下表 1-12 所示。

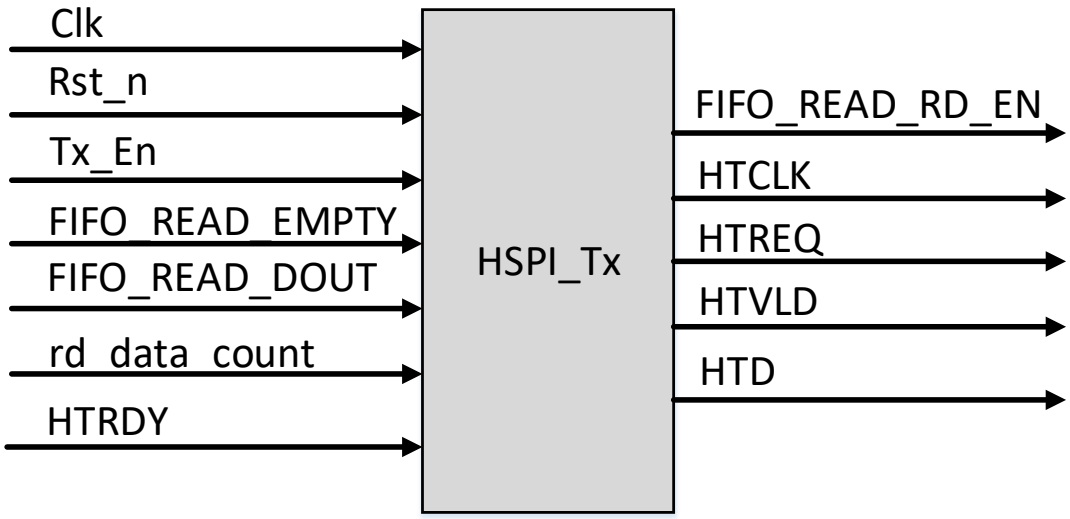


图 1-20 HSPI_TX 模块基本结构框图

对该模块的信号说明如下表所示：

表 1-12 HSPI_TX 模块块信号说明表

信号	I/O	描述
Clk	I	模块时钟信号
Rst_n	I	模块复位信号，低电平复位

Tx_En	I	发送使能信号
rd_data_count	I	FIFO 的数据可读数量
FIFO_READ_EMPTY	I	FIFO 的空信号
FIFO_READ_DOUT	I	FIFO 的数据
HTRDY	I	输入数据接收状态
FIFO_READ_RD_EN	O	FIFO 的读请求信号
HTCLK	O	输出通讯时钟信号
HTREQ	O	输出发送请求信号，高电平有效
HTVLD	O	输出数据发送状态，高电平有效
HTD	O	输出的 32 位数据

1.3.7.1 代码设计

我们使用状态机的方式实现该模块的功能，定义状态如下所示，分别为空闲状态、接收包头状态、接收数据包状态、结束状态、判断 EMPTY 状态、延迟状态。

```
localparam  S_IDLE    = 6'b000001;
localparam  S_HEAD    = 6'b000010;
localparam  S_DATA    = 6'b000100;
localparam  S_TAIL    = 6'b001000;
localparam  S_EMPTY   = 6'b010000;
localparam  S_DELAY   = 6'b100000;
```

状态转移图如下所示。

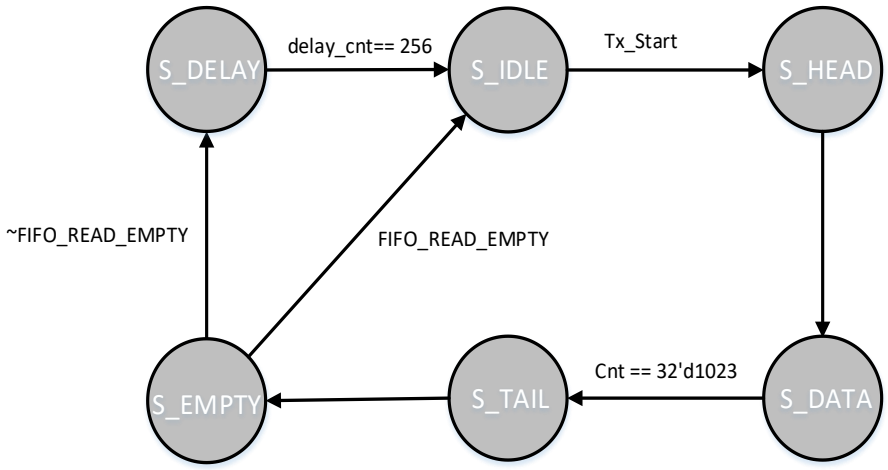


图 1-21 状态转移图

下面将对每个状态的代码设计进行简单的说明。

1、S_IDLE 状态

在 S_IDLE 状态，等待发送开始信号拉高，此时拉高 HTREQ 信号，状态机开始跳转，代码如下所示。

```
S_IDLE: begin
    if(Tx_Start) begin
        HTREQ <= 1'b1;
        SM_State <= S_HEAD;
    end
    else if(~Tx_Done) begin
        SM_State <= S_IDLE;
        HTREQ <= 1'b0;
    end else begin
        SM_State <= S_IDLE;
    end
    Tx_Done <= 1'b0;
    Data_Valid <= 1'b0;
end
```

开始发送信号什么时候开始拉高呢？这里通过一个使能信号进行控制，只要状态机处于 S_IDLE 状态，FIFO 的可读数据量大于或等于 1022，并且使能信号拉高的时候就认为这是一次开始传输信号，代码如下所示。

```
always @(posedge Clk or negedge Rst_n)
begin
    if(~Rst_n)
        Tx_Start <= 1'b0;
    else if(((SM_State == S_IDLE) && (rd_data_count >= 1022)
&& Tx_En))
        Tx_Start <= 1'b1;
    else if(SM_State == S_HEAD)
        Tx_Start <= 1'b0;
    else
        Tx_Start <= Tx_Start;
end
```

2、S_HEAD 状态

在 S_HEAD 状态的时候，开始发送包头（TLL+TSQN+USDF），同时将 FIFO 的读使能提前拉高，以及 Data_Valid 数据有效信号拉高，代码如下所示。

```
Localparam  Tx_Length_Low  = 2'b11;
```

```
Localparam  User_Define_Data=
26'b1010_1010_1010_1010_1010_1010_10;

S_HEAD: begin
    Tx_Data <= {Tx_Length_Low,Tx_Num_Sequence,User_Define_Data};

    if(HTRDY_R2) begin
        Data_Valid <= 1'b1;
        FIFO_READ_RD_EN <= 1'b1;
        if(Tx_Num_Sequence == 4'd15)
            Tx_Num_Sequence <= 4'd0;
        else
            Tx_Num_Sequence <= Tx_Num_Sequence + 1'1b1;
        SM_State <= S_DATA;
    end else
        SM_State <= S_HEAD;
end
```

3、S_DATA 状态

在 S_DATA 状态的时候，开始发送 FIFO 的数据，此时，拉高 Data_Valid 信号，数据发送计数器也开始计数，等到计数器记到 1023 的时候，拉低 FIFO 的读使能信号，状态机开始跳转，代码如下所示。

```
always @(posedge Clk or negedge Rst_n)begin
    if(~Rst_n)
        Cnt <= 32'd0;
    else if(SM_State == S_DATA)
        Cnt <= Cnt + 1'b1;
    else
        Cnt <= 32'd0;
end

S_DATA: begin
    Tx_Data <= FIFO_READ_DOUT;
    Data_Valid <= 1'b1;
    if(Cnt == 32'd1023) begin
        FIFO_READ_RD_EN <= 0;
        SM_State <= S_TAIL;
    end else
        SM_State <= S_DATA;
```

```
end
```

4、S_TAIL 状态

在 S_TAIL 状态，发送 CRC 校验值，并且拉高 Data_Valid 以及 Tx_Done 信号，代码如下所示。

```
S_TAIL: begin
    Tx_Done <= 1'b1;
    Data_Valid <= 1'b1;
    Tx_Data <= CRC_32D;
    SM_State <= S_EMPTY;
end
```

5、S_EMPTY 状态

在 S_EMPTY 状态，将 Tx_Done 信号拉低，让 Tx_Done 只维持一个脉冲。接着判断 FIFO 的空信号，如果 FIFO 里面还有数据，就进入延迟状态，准备开始新一轮的传输，代码如下所示。

```
S_EMPTY: begin
    Tx_Done <= 1'b0;
    if(~FIFO_READ_EMPTY) begin
        SM_State <= S_DELAY;
    end else
        SM_State <= S_IDLE;
end
```

6、S_DELAY 状态

在进入 S_DELAY 状态，延迟计数器开始计数，计数到 256 时，状态机开始跳转，代码如下。

```
always @ (posedge Clk or negedge Rst_n)
if (~Rst_n)
    delay_cnt <= 0;
else if (SM_State == S_DELAY)
    delay_cnt <= delay_cnt + 1;
else
    delay_cnt <= 0;

S_DELAY: begin
    Data_Valid <= 1'b0;
    HTREQ <= 0;
    if(delay_cnt == 256) begin
```

```
        SM_State <= S_IDLE;
    end else
        SM_State <= S_DELAY;
end
```

对于其他信号的赋值代码如下。

```
always@(posedge Clk or negedge Rst_n)
if(~Rst_n)
    HTVLD <= 0;
else
    HTVLD <= Data_Valid;

always@(posedge Clk or negedge Rst_n)
if(~Rst_n)
    HTD <= 0;
else if(Tx_Done)
    HTD <= CRC_32D;
else
    HTD <= Tx_Data;

always@(posedge Clk or negedge Rst_n)
if(~Rst_n) begin
    HTRDY_R1 <= 0;
    HTRDY_R2 <= 0;
end
else begin
    HTRDY_R1 <= HTRDY;
    HTRDY_R2 <= HTRDY_R1;
end
```

1.4 板级验证

1.4.1 系统所需硬件

1、ACX750_CH569 开发板一块

2、12V 电源适配器

3、USB3.0 数据线一根

4、带 USB3.0 接口的电脑一台

5、XILINX 下载器

6、1 根杜邦线

1.4.2 硬件连接图

将 ACM7606C 模块、网线、下载器、电源线依次连接在开发板上，需要注意 ACM7606C 模块连接正确后右边会多出 6 组排针，由于视觉缘故，这里很容易连接错误，并且需要将 ACM7606C 模块上的 H2 接到串行接口上，如下图 1-22 所示。

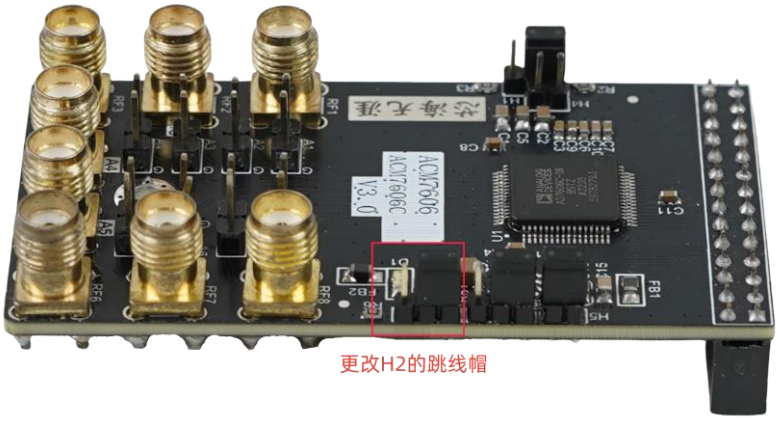


图 1-22 更改跳线帽

还需要给 AD7606C 连接信号源，这里我们连接的是 AD7606C 的通道 1，信号源给的是 1khz，vpp=5V 的正弦波，硬件连接图如下图所示。注：开发板上的 GPIO35 接到开发板 U29 的 8 脚。

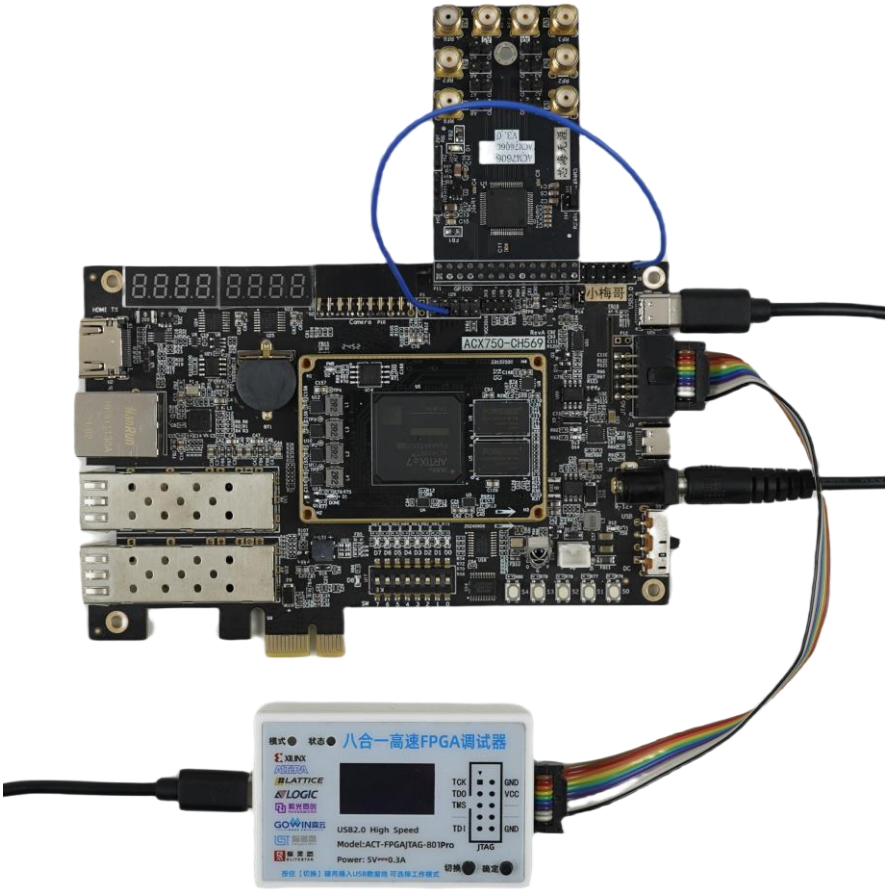


图 1-23 硬件连接图

1.4.3 MCU 固件烧入

双击 WCHISPStudio 图标。

图 1-24 打开 WCHISPStudio

打开 WCHISPStudio 软件下载 MCU 程序，打开界面如下图所示。

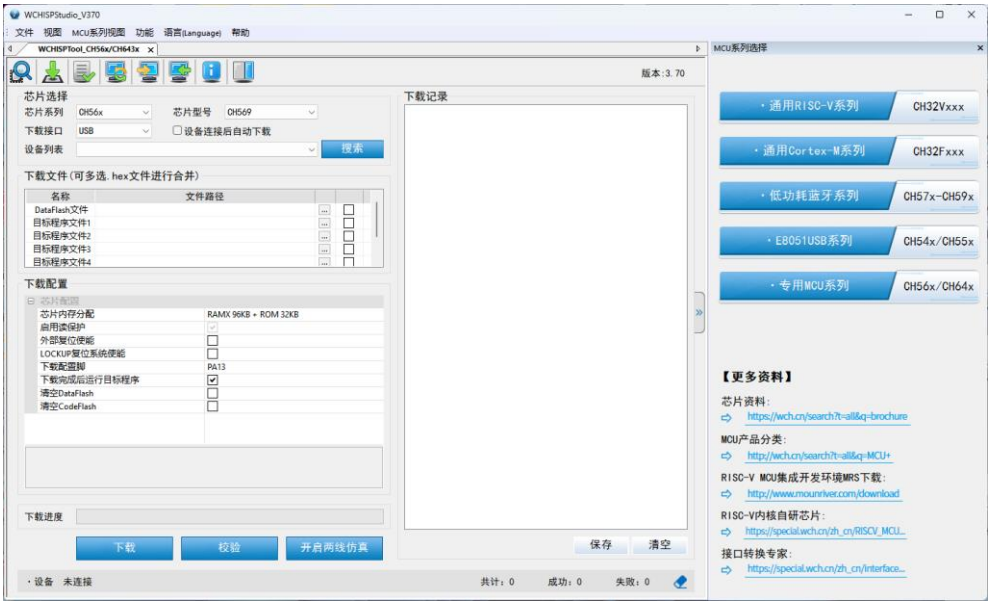


图 1-25 软件打开界面

此时在界面的左下角显示“设备未连接”字样，如下图所示。

图 1-26 设备未连接

接着按住开发板板卡背面的两个按键，一个是 rst 按键，一个是 boot 按键，如下图所示。

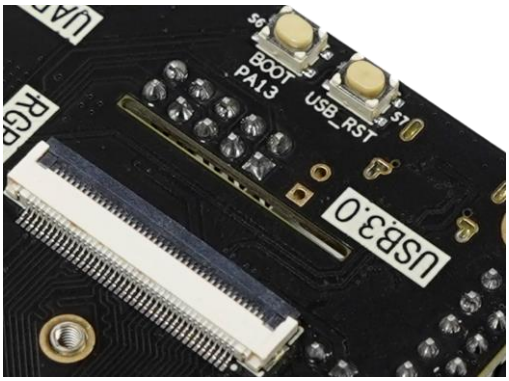


图 1-27 rst 按键与 boot 按键

同时按住两个按键，然后将电源打到 DC 档，接着先松开 rst 按键，再松 boot 按键。此时可以看到设备已经连接上了，如下图所示。

图 1-28 设备已连接

接着就是下载 USB 固件，操作如下所示。

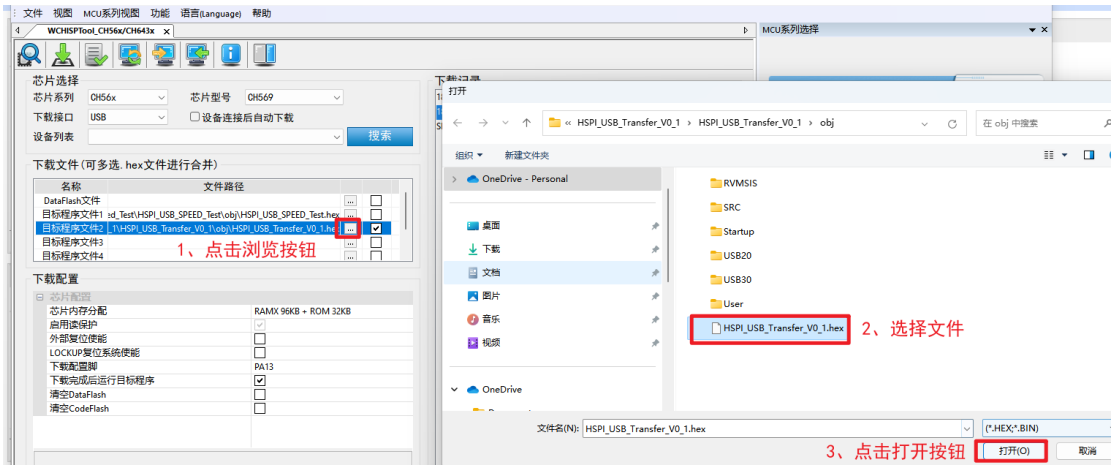


图 1-29 选择 hex 文件

接着点击下载按钮，等待固件下载完成。

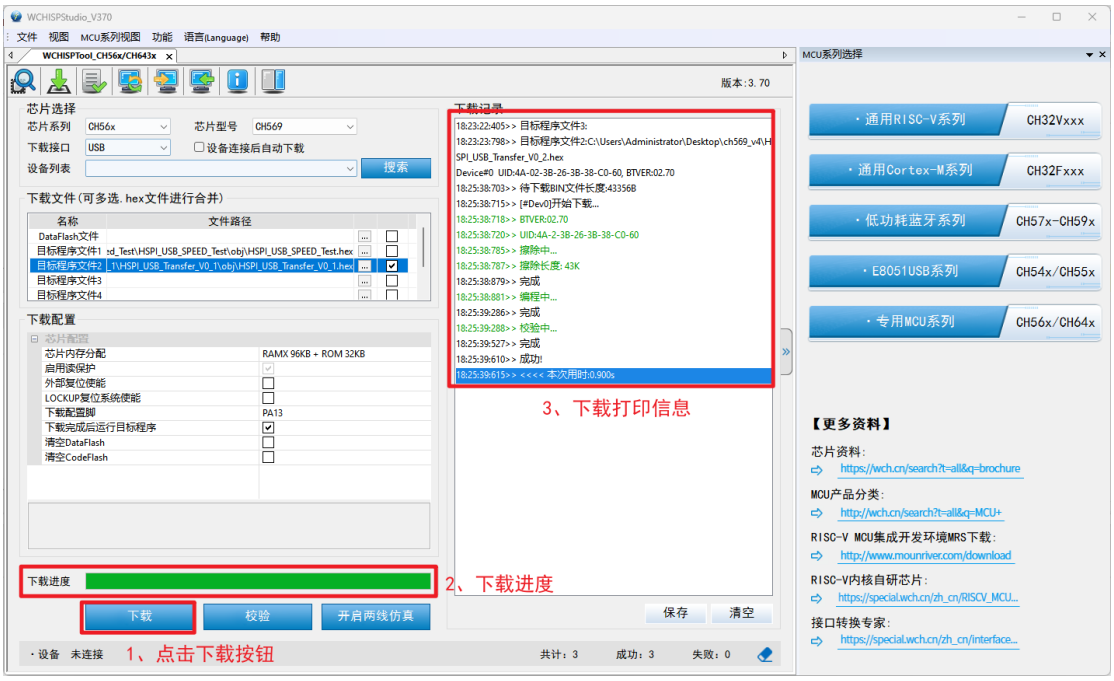


图 1-30 固件下载完成

固件下载完成后，将开发板重新上电。

1.4.4下载 bit 文件

将生成的 bit 文件下载到开发板上，进行验证，下载方式如下所示。

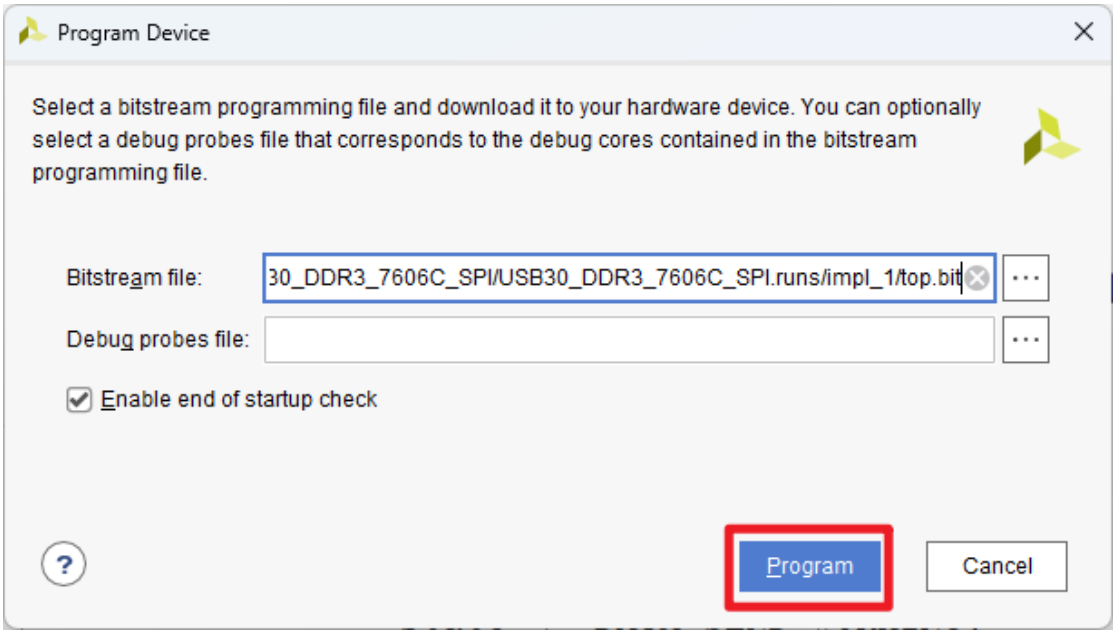


图 1-31 下载 bit

FPGA 程序下载完成后，可以看到设备管理器多了一个外部接口 USB CH372/CH375，如下图所示。如果不下 FPGA 程序，则没有这个设备。

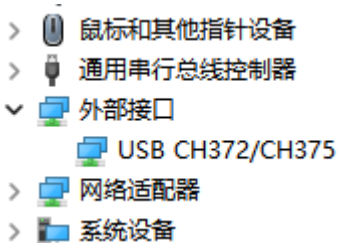


图 1-32 外部接口

1.4.5 使用 BUS hound 下发指令以及收取数据

首先打开 bushound 软件，双击 Devices 下面的 USB CH372/CH375，打开 Bus Commander，如下图所示。

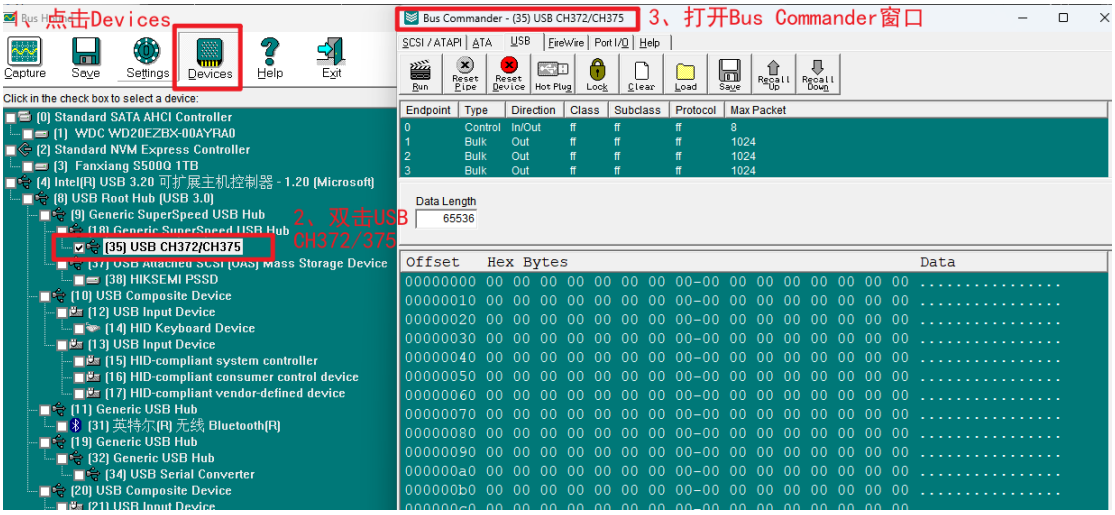


图 1-33 打开 Bus Commander

接着选择端点二，然后在 Data Length 输入要发送指令的长度，本次设计的指令为 32 个字节，输入完成后按键盘上的回车按键即可，如下图所示。

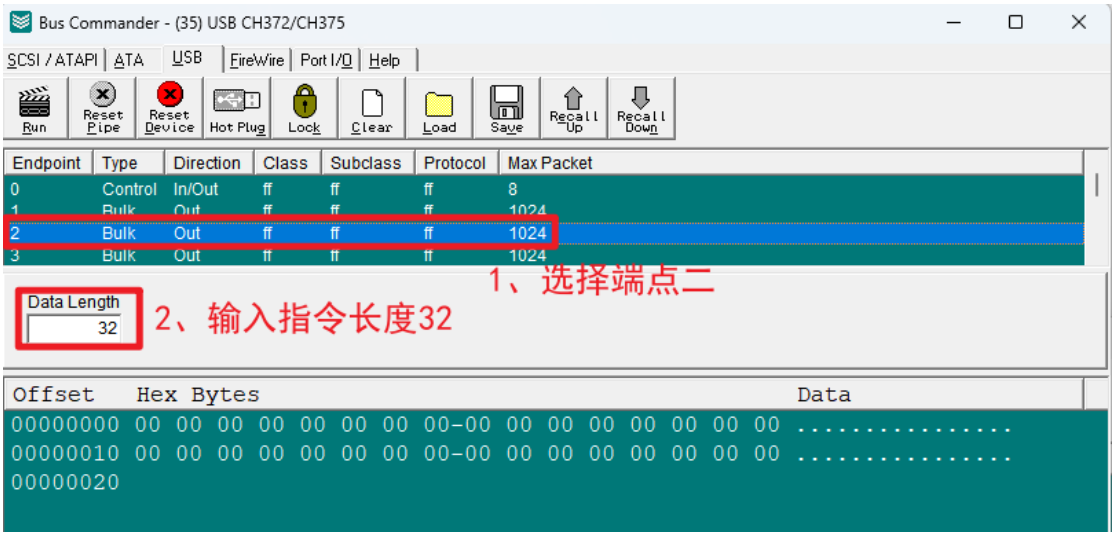
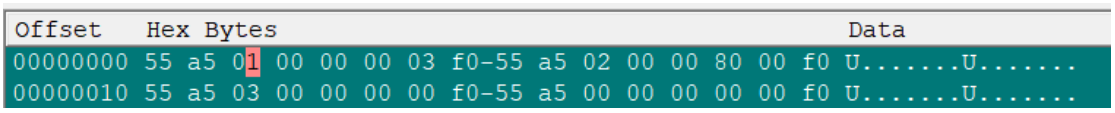


图 1-34 设置待发送的指令长度

接着在数据中填充指令，如下图所示。



在前面接收转命令模块中介绍到数据帧格式对 AD7606C 的四个寄存器进行配置。例如，PC 端要设置采样数据个数（地址为 2，set_sample_num 寄存器）为 0x8000（32768）个，发送数据帧内容为 0x55 0xA5 0x02 0x00 0x00 0x80 0x00 0xF0

PC 端要设置采样通道（地址为 1，adc_ch_sel 寄存器)为双通道，发送的数

据帧内容为 0x55 0xA5 0x01 0x00 0x00 0x00 0x01 0xF0。

PC 端设置采样速率（地址为 3，set_sample_speed 寄存器），发送的数据帧内容为 0x55 0xA5 0x03 0x00 0x00 0x00 0x63 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 号寄存器是启动指令，ADC 在为配置完全的情况下启动，数据很容易出现错误值。

配置成数据格式如下

55A50100000001F055A50200008000F055A5030000063F055A50000000000F0

这里为了方便后续使用，将这条指令进行保存，点击 Bus Commander 窗口中的保存按钮，在弹出的对话框中设置保存的位置以及文件名，最后点击保存按钮即可，操作如下图所示。

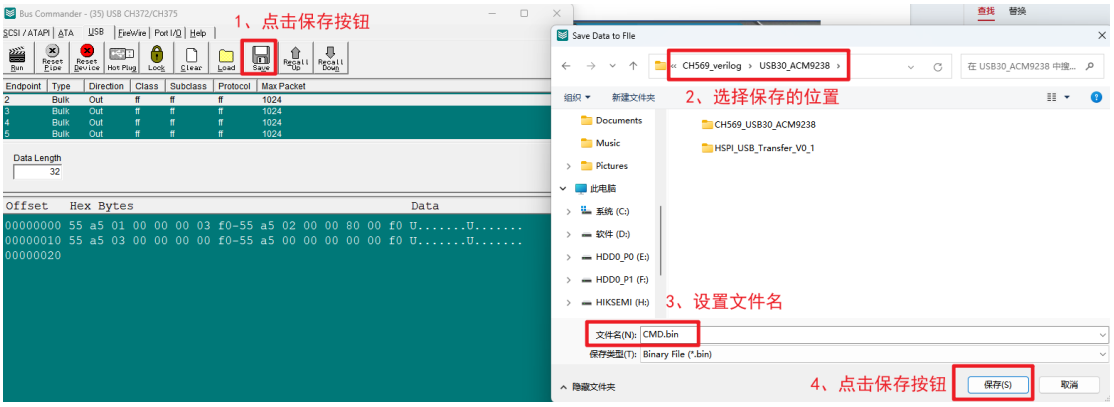


图 1-35 保存指令操作步骤

这样，下次使用的使用指令的时候只用点击 Bus Commander 窗口中的 load，然后选择保存的指令文件，最后点击打开按钮即可加载指令，操作如下所示。

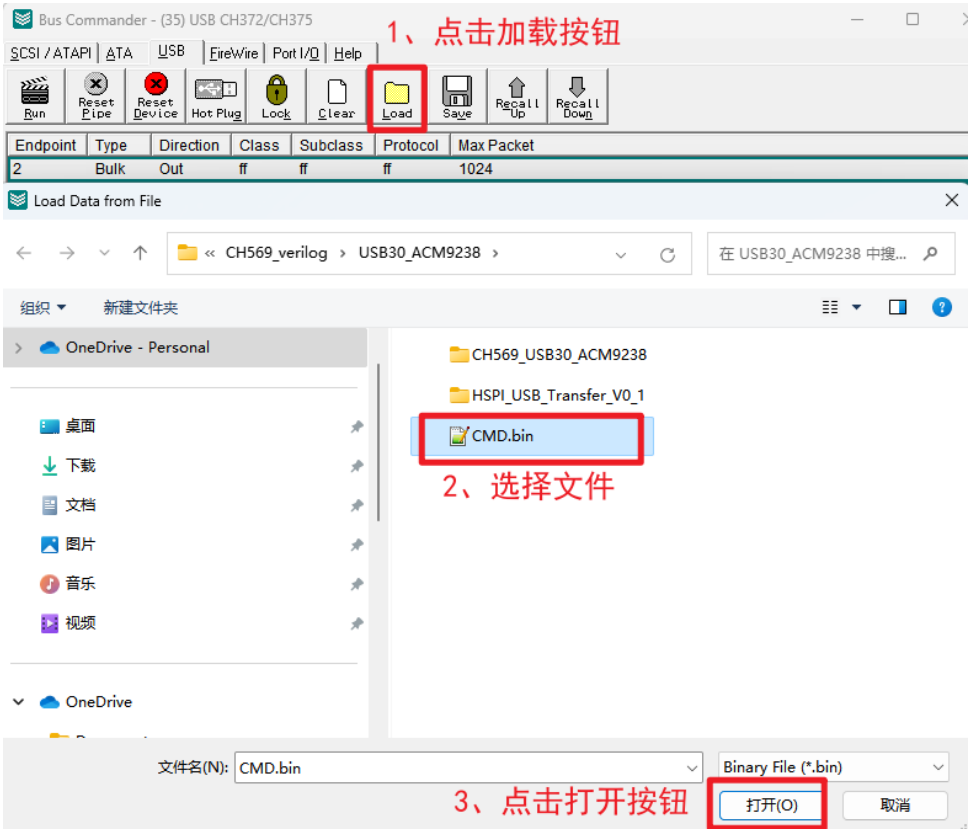


图 1-36 加载已保存指令

指令加载完成后，点击 RUN 按钮即可下发指令。

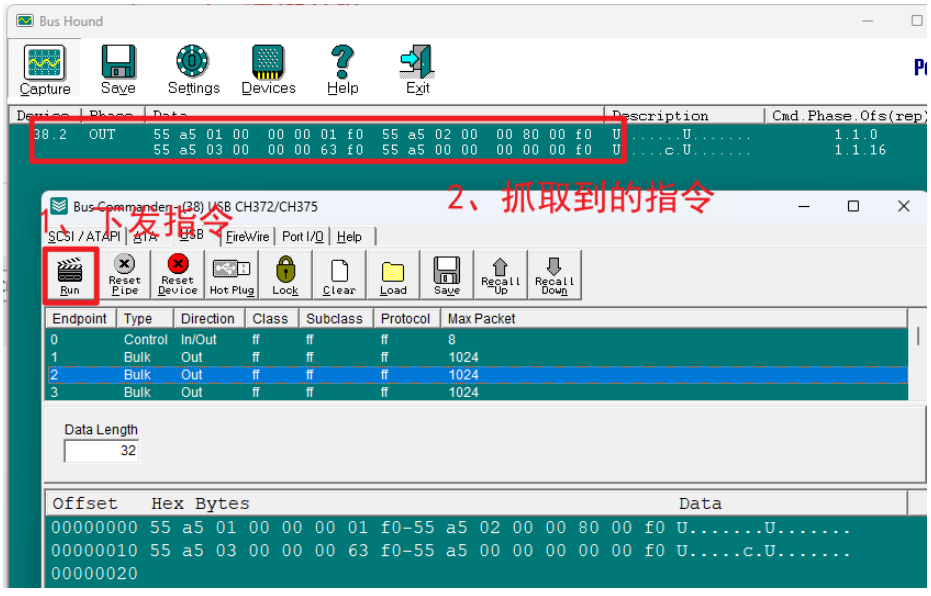


图 1-37 下发指令

接着通过 BUS hound 将写进端点一的数据进行读取，选择端点一，然后在 Data Length 输入 65536，因为下发的指令为采集 0X8000 个数据，也就是采集 32768 个 16 位的数据，也就是采集 65536 个字节的数据，最后点击 run 按钮读取

数据，如下图所示。

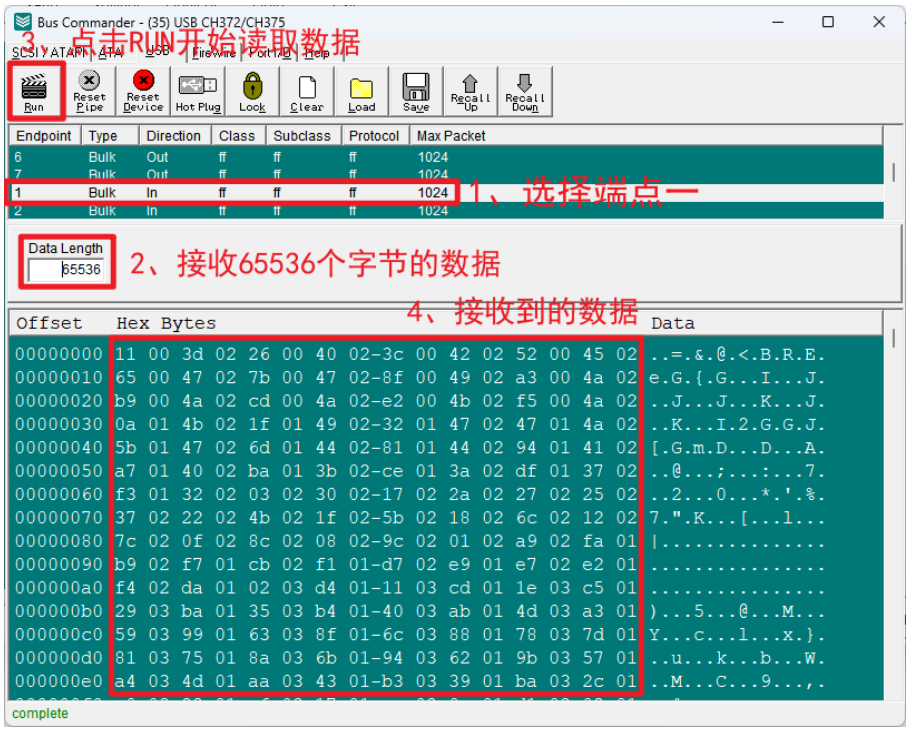


图 1-38 读取数据

接着点击保存按钮，将读取到的数据进行保存，然后通过上位机进行绘图。

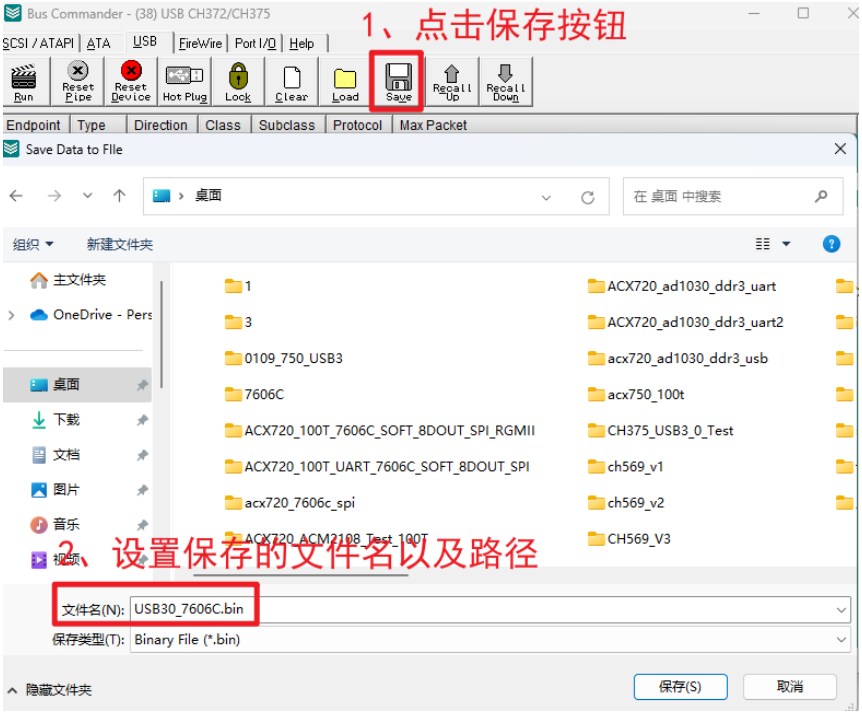


图 1-39 保存数据

接着打开小梅哥数据采集仪软件，该软件可在论坛进行获取，地址如下：

https://www.corecourse.cn/forum.php?mod=viewthread&tid=29728&highlight=

QT

打开界面如下所示。

图 1-40 上位机软件

接着设置一下参数，操作如下所示。

图 1-41 选择 ADC

接着点击“文件”下的“导入数据文件”按钮，选择前面保存的文件，如下图所示。

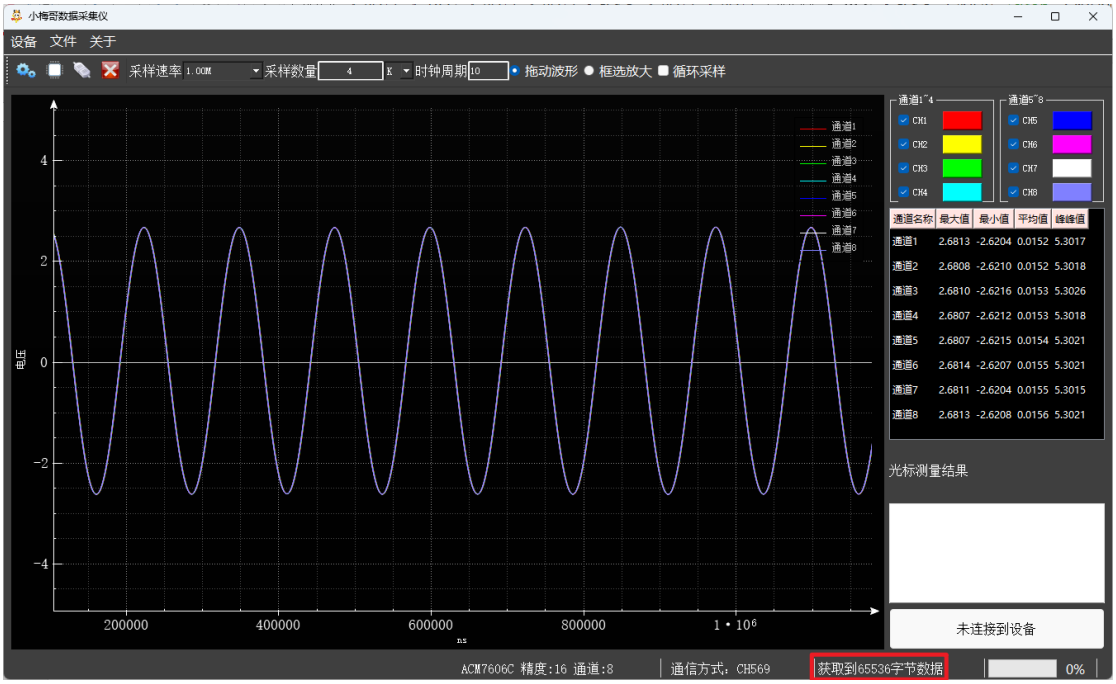


图 1-42 绘图

由上图可以看到，获取到的字节数为 65536 个字节，数据量是没有问题的，波形也是正确的。接着测量一下频率，这里在测量前，先将其他通道进行取消勾选，只勾选通道 1，如下图所示。

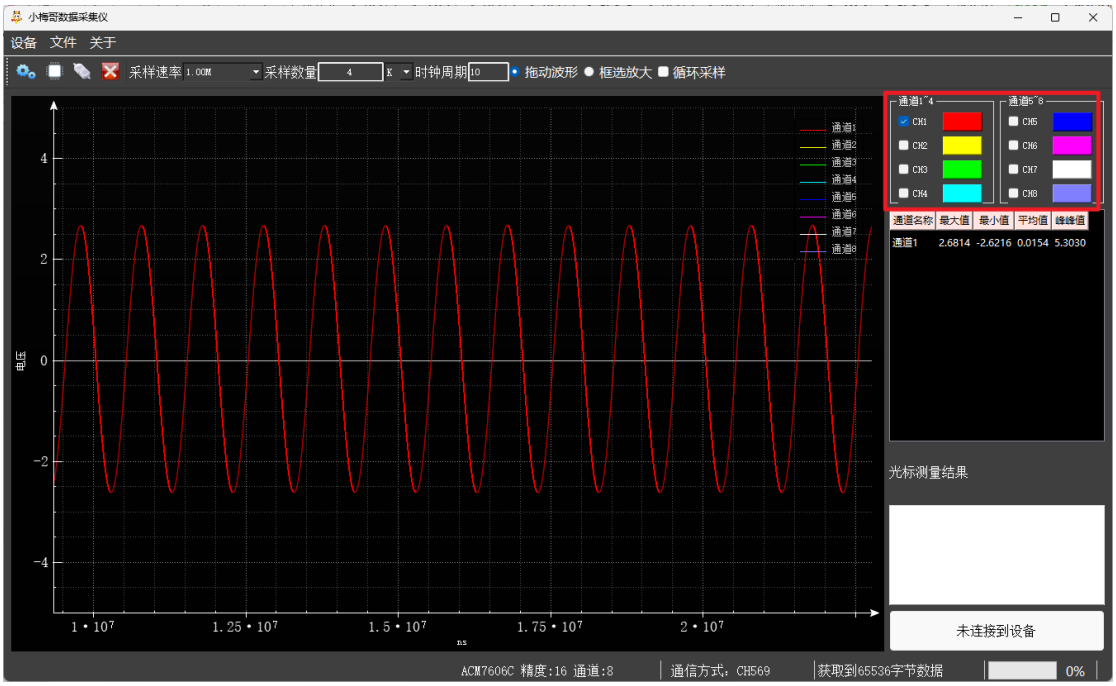


图 1-43 取消勾选其他通道

取消勾选完成后，在波形显示区域右击，然后点击“设置光标”按钮，弹

出添加光标对话框，如下图所示。

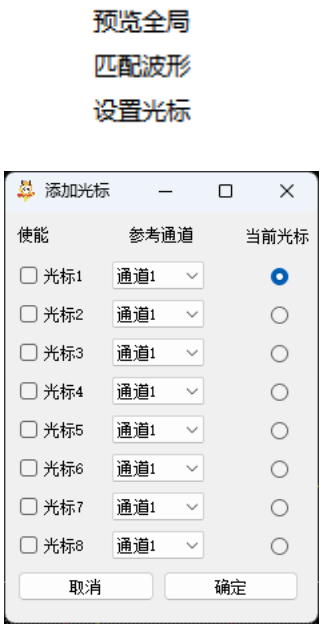


图 1-44 弹出添加光标按钮

将光标一前面的复选框进行勾选，参考通道选择“通道 1”，最后点击确定按钮，如下图所示。

图 1-45 设置光标 1

接着在波形显示界面会出现一根跟随鼠标移动的白线，此时在合适位置单击鼠标左键，白线就会固定住，如下图所示。

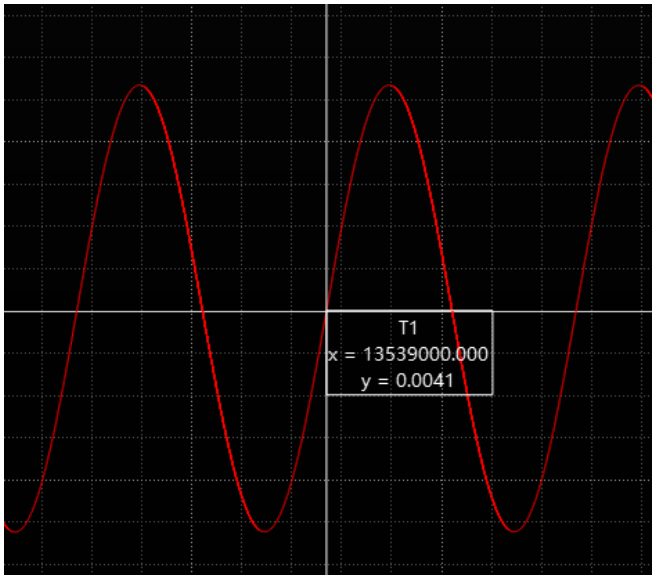


图 1-46 固定光标一

接着在添加一根光标，将光标二前面的复选框进行勾选，参考通道选择“通道 1”，最后点击确定按钮，如下图所示。

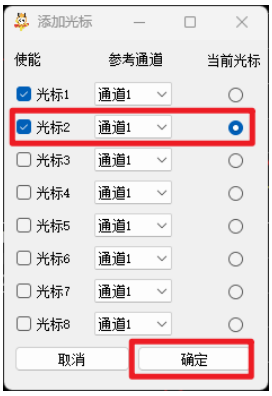


图 1-47 设置光标 2

接着在波形显示界面会出现另一根跟随鼠标移动的白线，此时在合适位置单击鼠标左键，白线就会固定住，如下图所示。

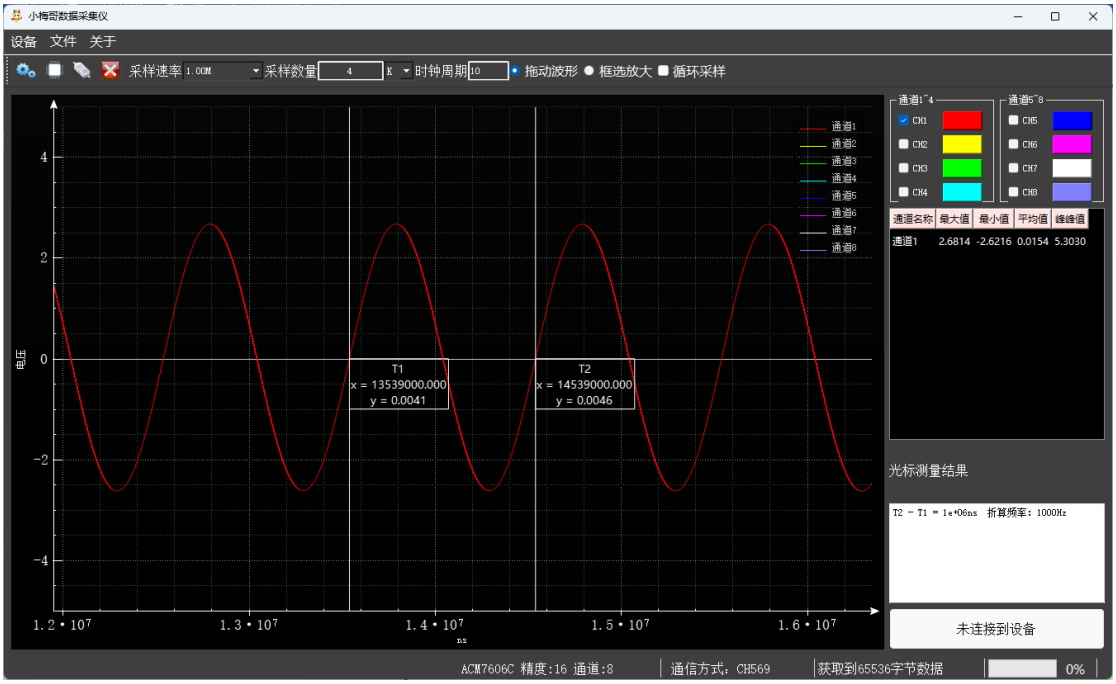


图 1-48 固定光标二

此时，在光标测量结果窗口中会显示出波形的频率，如下图所示。

图 1-49 测量频率

由上图可知，计算出来的频率为 1KHz，和信号发生器输出的频率一致。

1.4.6 使用小梅哥数据采集仪上位机采集数据

首先修改通信方式，将通信方式修改成 CH569，如下图所示。

图 1-50 修改通信方式

由于前面选择的ADC，这里就不再进行修改，接着修改采样数量，例如8K，设置完成后击连接按钮，如下图所示。

图 1-51 连接设备

接着点击“点击开始采样”按钮，如下所示。

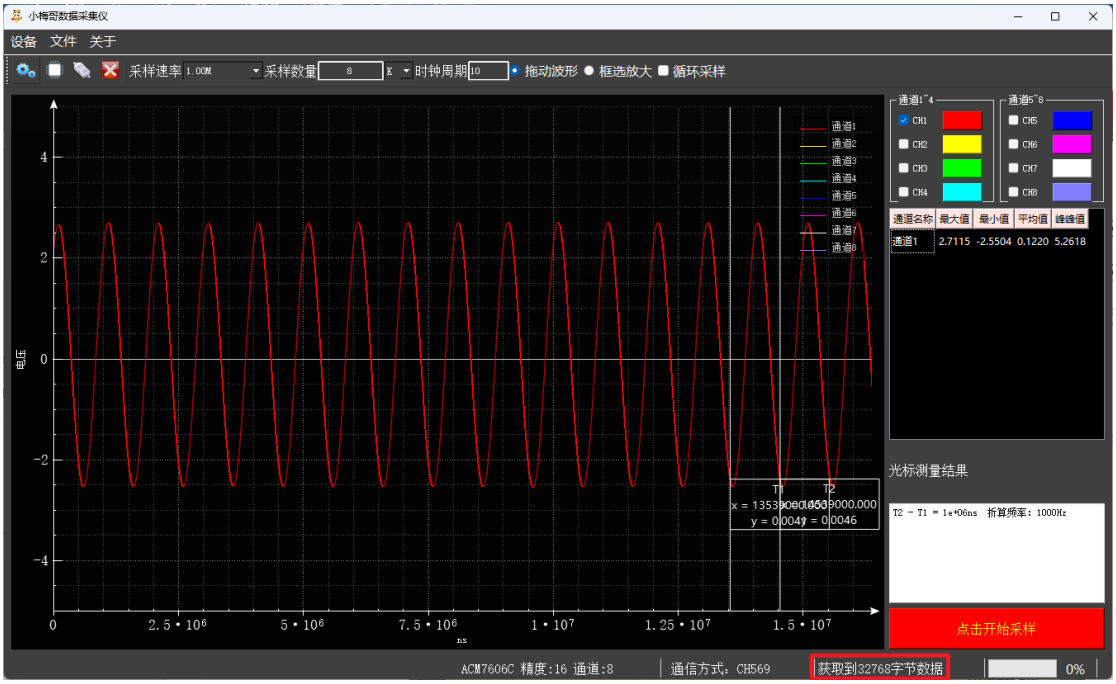


图 1-52 获取到的数据

由上图可知，波形没有毛刺，采集到的数据量也是没有问题的，至此基于 USB3.0 的 AD7606C 数据采集 DDR3 存储系统就已经验证完成。

1.5 小结

本实验介绍了基于 USB3.0 的 AD7606C 的数据采集实验，用户通过 bus hound 向开发板发送指令数据配置 AD7606C 的四个寄存器，以此控制 ADC 进行采样并将数据进行放进 DDR 中缓存，随后再通过 USB3.0 将数据上传至 PC，最后再由 bus hound 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。