

1 ACM2108/ACM108 数据采集 DDR3 缓存串口发送实验

该文档主要是以 ACM2108 为例进行说明的，用户如果使用的 ACM108 模块，实验操作步骤同样适用，但是需要注意的是 ACM108 模块只有一个通道 0 可以使用。两个模块对应的工程文件为 “acx720_acm2108_ddr3_uart.rar” 和 “acx720_acm108_ddr3_uart.rar”。

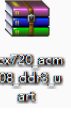
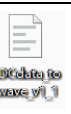
1.1. 文档所涉及文件说明

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

版本记录：

V1.0	首次发布	2022.12.2
.....		

本教程配套的文件压缩包名：acx720_acm2108_108_ddr3_uartV1.0.rar，解压后可以得到配套工程文件、上位机软件和 matlab 绘图代码文件，对于压缩包中的文件说明如下：

	acx720_acm2108_ddr3_uart.rar: 使用 ACM2108 实验使用到的 vivado 工程。其中包括各个模块的源文件代码：串口接收数据模块、串口指令接收和指令解析模块、数据转换模块、ddr3 的含 fifo 的 2 端口封装模块、串口数据发送模块，数据采集模块等。
	acx720_acm108_ddr3_uart.rar: 使用 ACM108 实验使用到的 vivado 工程。其中包括各个模块的源文件代码：串口接收数据模块、串口指令接收和指令解析模块、数据转换模块、ddr3 的含 fifo 的 2 端口封装模块、串口数据发送模块，数据采集模块等。
	data: 测试数据文件夹，用来存放本次实验过程中，生成的数据文件，用户可以使用 Matlab 软件对我们提供的测试数据文件进行数据分析对比。
	ADCdata_to_wave_v1_1.m: Matlab 源文件代码，对于串口传输过来的数据，我们得到的只是一个数据文件，如何判断采集到的数据是否正确，这里采用 Matlab 软件进行数据分析，用户可以通过修改我们提供的 matlab 源文件中的文件路径，进行数据分析。

1.2. 数据采集的意义

在计算机广泛应用的今天，数据采集的在多个领域有着十分重要的应用。

数据采集是计算机与外部物理世界连接的桥梁，通过数据采集工作，自然界的许多模拟量信息能够借助计算机进行保存，分析，还原等操作。技术实践中，我们只需要制订上位机(PC)与移动数据采集器的通信协议，就可以实现两者之间阻塞式通信交互过程。

数据采集系统往往由传感器、模拟多路开关、放大器，采样保持器、AD 转

换器、计算机及外设等组成。

在农业、工业、日常生活和航空航天等领域，尤其是在对信息实时性能要求较高或者恶劣的自然环境中，数据采集有其应用的必要性。比如说:在工业生产和科学技术研究的各行业中，常常有利用 PC 或工控机配合末端传感器对诸如液位、温度、压力、频率等参数进行实时监控和记录的需求，这些环境往往有时候并不适合人类直接作业，或者即使人类进行直接作业，也无法达到和计算机自动采集分析处理某一个任务的实施效果。再比如说：在航天航空领域，卫星数据采集系统利用航天遥测、遥控、遥监等技术，对航天器远地点进行各种监测，并根据需求进行自动采集，经过卫星传输到数据中心处理后，送给用户使用。

谈到数据采集，不得不说说数据转换器。现在常用的数据转换方式是通过数据采集板卡进行，常用的有如 A/D 卡以及 422、485 等总线板卡，此次实验就是利用 ACM2108 数据采集卡实施数据采集。

1.3. ACM2108 高速 ADC+DAC 模块简介

本模块基于国产知名模拟器件设计和制造商杭州瑞盟公司的 8 位 50M 采样速率高速 ADC 芯片 MS9280（完全兼容 AD9280，但是速度比 AD9280 高）和 125Msps 转换速率的高速 DAC 芯片 MS9708（完全兼容 AD9708）。配合前端模拟信号调理电路，实现了 $\pm 5V$ 电压范围内信号的高速采样，以及模拟信号输出。以下图 1.1 为模块实物图。

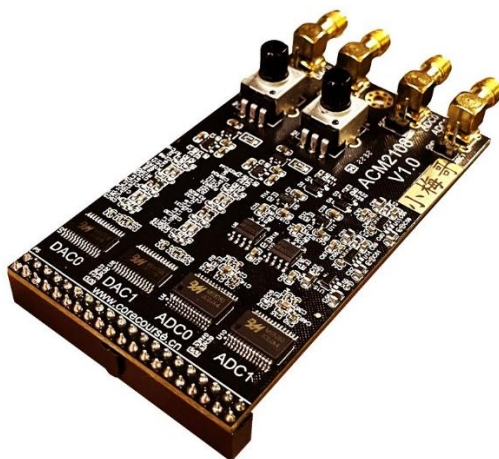


图 1.1 ACM2108 模块实物图

本模块可用于小梅哥全系列 FPGA、SOC、Zynq 开发板，包括国产开发板和各核心板的评估底板。AC620、AC6102、ACX720、ACZ702、AC609、智多晶 FPGA 开发板（AC208-SA5Z）、AC608 评估底板、AC601 评估底板、AC675 评估底板。

1.4. 实验任务

店铺：<https://xiaomeige.taobao.com>
技术博客：<http://www.cnblogs.com/xiaomeige/>

官方网站：www.corecourse.cn
技术群组：

1.4.1. 实验需要实现的功能

本次实验所需要实现的内容如下所示：

1. 实验通过数据采集模块实现模数转换，传递给开发板。在这里，我们采用 ACM2108 双通道数据采集模块作为数据采集卡进行数据采集。
2. 使用相关的串口通信软件，通过串口发码指令，可以设定需要采集的字节数，选择采集通道号，设置采样速率，启动采集。
3. 使用相关的串口通信软件，可以按设定的采集参数，接收采集的数据。数据通过串口发送到串口调试软件（后期可开发对应 PC 上位机）。将读取到的数据我们进行 txt 文件的保存，便于后期分析。
4. 采集到的数据经过 matlab 波形分析，能够得到和输入波形一致的输出波形，无数据丢失，无杂波。后期可结合实际情况，增加对噪声评定的环节。

1.4.2. 实验可行性分析

通过对各部分的参数进行分析，我们进一步论证了数据采集串口接收的方案可行性。

1. ADC 采集数据的最大带宽： $50\text{MSPS} * 8\text{bit} = 400\text{Mbps}$ ，由于 ACM2108 模块需要外部提供工作时钟，所以 FPGA 应向 ACM2108 模块供应 50M 时钟。
2. ACX720 板载 DDR3 带宽： $400 * 16 * 2 = 12800\text{Mbps}$ ，考虑到 DDR3 控制器不能 100%效率，按 80%效率，读写各占一半，也足够大于 400Mbps（ $500 * 8$ ），ADC 采集数据存储到 DDR3 不会存在带宽不够导致数据丢失问题。
3. 先考虑使用串口上传采集的 ADC 采集的数据，常见的串口波特率 9600bps, 115200bps, 这个比 ADC 采集数据的带宽要小的多，想通过串口实时的将采集的数据传出去是不行，会存在数据丢失问题，考虑到这点，将 ADC 采集和串口上传数据分开处理，先 ADC 采样指定个数（或指定时间）的数据，保存在板载 DDR3，然后停止 ADC 采样，之后串口再将 ADC 采样数据依次上传到 PC。
4. DDR3 存储数据量：DDR3 内存 256MByte，ADC 数据位宽为 8bit，按 16bit（2 个 Byte）计算，可存储数据最大个数为 $256 * 1024 * 1024 / 2 = 134,217,728$ （个 16bit 数）。在这里，由于 MS9280 的最高工作频率为 50M，而 DDR3 的默认工作频率为 200M，为了简易的处理硬件的时钟异步问题，同时简化对 DDR3 的控制，可以在程序中分别开拓一个 fifo 写缓冲区和 fifo 读缓冲区，完成 ddr3 的调度控制机制和跨时钟域的处理。

理。

1.5. 系统整体设计

系统的整体设计框图如下图 1.2 所示：

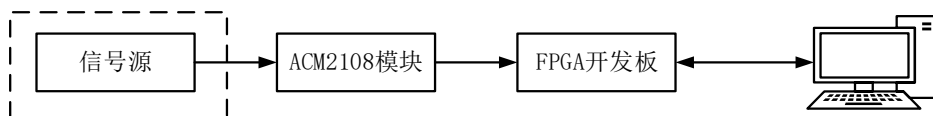


图 1.2 系统的整体设计框图

在整个系统的设计过程中，我们所需要进行的工作就是在我们的 FPGA 开发板上进行各个模块的设计，我们提供的例程在 FPGA 芯片内部进行的模块设计如下图 1.3 所示。上图所示的信号源用户可以通过数据信号发生器产生，也可以通过 ACM2108 模块自身的 DAC 模块产生。

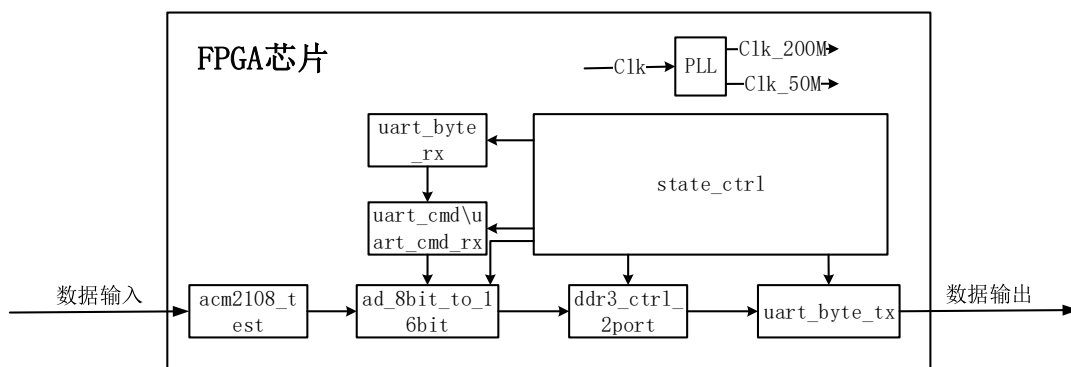


图 1.3 FPGA 芯片内部模块设计图

对于每个模块的功能介绍如下所示：

1. `uart_byte_rx`: 串口接收模块，接收从串口助手发送来的指令。
2. `uart_cmd\uart_cmd_rx`: 串口指令接收和指令解析模块，将收到的指令进行翻译拆解，指令分类。
3. `ad_8bit_to_16bit`: 数据转换模块，将 ACM2108 发送过来的 8 位数据转换成 16 位的数据，方便我们后续进行上位机采集数据。
4. `acm2108_test`: ACM2108 数据采集模块，在该模块中我们也设计了 DDS 模块，方便用户通过该模块自身的 DAC 芯片产生信号源。
5. `ddr3_ctrl_2port`: ddr3 的含 fifo 的 2 端口封装模块，主要负责整个数据的存储功能。
6. `uart_byte_tx`: 串口发送模块，将最终采集到的数据通过串口发送出去。
7. `state_ctrl`: 状态机模块，协调各个模块的信号控制，程序状态的总控制模块。

8. pll: 锁相环模块, 50M 时钟输入, 输出 200M 的时钟给 DDR3, 50M 的时钟给其他模块。由于我们在设计时, 对于时钟频率没有要求的模块, 都可以采用 50MHz 的频率进行设计, 而 DDR3 驱动模块的工作时钟频率最低也得 200MHz 才能支持正常工作, 所以我们需要使用锁相环模块完成整个工程的频率配置, 以实现在工程中引入两种时钟频率的设计需求。

了解了 FPGA 代码的设计架构, 下面需要构思出本次设计的状态转移图。如下图所示。

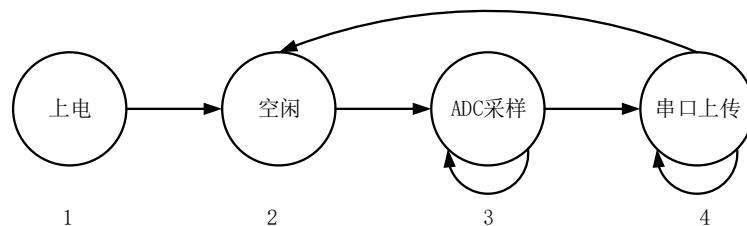


图 1.4 状态转移图

接下来就简单的讲解本工程的状态机设计思想。

1. 程序上电后, 进入空闲状态, 同时允许接收和解析从 PC 上位机发送到 FPGA 的指令信息。
2. 对于本次设计, 此时可以通过串口指令设置采样个数, 设置采样通道, 下发采样开始的指令。
3. 当 FPGA 解析出启动采样的指令, 则立即结束空闲状态, 并根据采样数量和采样通道, 完成采样工作。每一个数据的采样, 都是一个提取、解析、存储的小循环过程。采样完成的数据暂存在 DDR3 中。
4. 当数据采集完成后, 设计的计数器计满, 随后启动数据发送的过程。
5. 每一个数据发送的过程, 又是一个从 DDR3 中提取、串口发送传输的小循环。
6. 由于我们期望采集的数据格式是 16 位位宽, 而串口的标准数据格式不支持 16 位位宽协议。因此我们将需要发送的每一个采集数按照高 8 位和低 8 位拆解后实现发送。
7. 当所有采集的数据都传输完成后, 则系统重新回到空闲状态。
8. 在实际的执行过程中, 状态机还需要考虑: 先有数据信号, 后有控制信号的问题。部分控制信号, 是通过对数据信号的解析来实现状态控制和变更的。所以从细节出发, 在代码实现阶段, 还需要添加一些辅助的状态完成一些诸如延时若干时钟周期等待指令解析完成, 来保证这些核心

状态的正常运行。这就要求在代码设计阶段，通过仿真实观察的实验现象，仔细斟酌状态转移细节，适时插入用于延迟和数据传递的过渡状态。这里，我们给出经过一定分析后的状态转移图，如下图 1.5 所示：

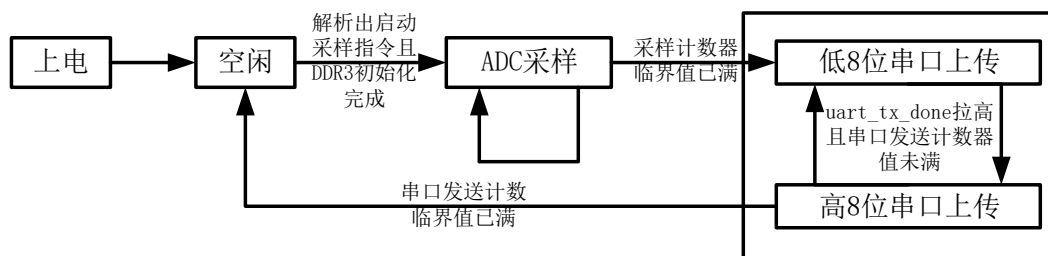


图 1.5 分析后状态转移图

关于状态转换细节处理的相关问题，以及如何选取状态转换的判断条件，我们在后面代码详解部分进行详述。

1.6. 程序模块介绍

1.6.1. 顶层模块对外接口功能介绍

顶层模块整体接口框图如下图 1.6 所示，端口列表如下表 1- 1 所示。

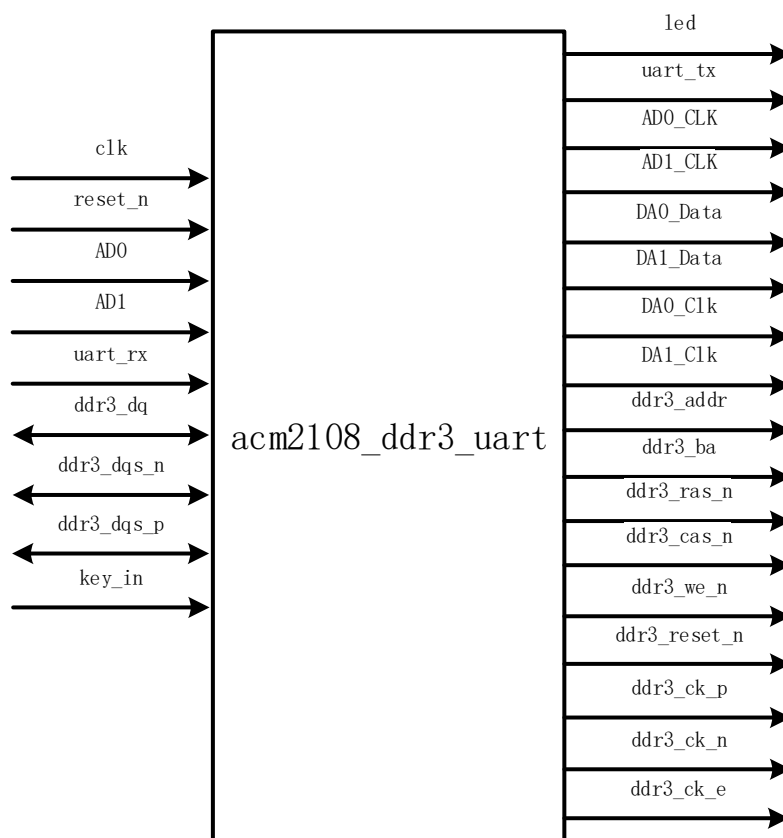


图 1.6 顶层模块整体框图

表 1-1 顶层模块端口列表

端口名	端口类型	描述
clk	input	系统时钟 50MHz
reset_n	input	系统复位，低有效
led	output[1:0]	用于测试的 LED 指示灯，我们定义 led[0]为锁相环初始化完成指示灯，led[1]为 DDR3 初始化完成指示灯。
uart_rx	input	串口接收信号引脚
uart_tx	output	串口发送信号引脚
key_in	input	按键，分配给了开发板上的 S0，控制 DAC 输出信号
AD0	input[7:0]	ACM2108 模块 ADC 的 8 位数据输入引脚第零通道
AD1	input[7:0]	ACM2108 模块 ADC 的 8 位数据输入引脚第一通道
AD0_CLK	output	ACM2108 模块 ADC 的第零通道时钟
AD1_CLK	output	ACM2108 模块 ADC 的第一通道时钟
DA0_Data	output	ACM2108 模块 DAC 的第零通道数据输出
DA1_Data	output	ACM2108 模块 DAC 的第一通道数据输出
DA0_CLK	output	ACM2108 模块 DAC 的第零通道时钟
DA1_CLK	output	ACM2108 模块 DAC 的第一通道时钟
ddr3_dq	inout[15:0]	DDR3 数据输入、输出：双向数据总线。若模式寄存器中使能了 CRC 功能，那么在数据 burst 结束时就会附加一段 CRC 码。
ddr3_dqs_n ddr3_dqs_p	inout[1:0] inout[1:0]	DDR3 差分数据选通信号：差分信号对，作输入时与写数据同时有效，作输出时与读数据同时有效。读数据时与边沿对齐，但是跳变沿位于写数据的中心。DDR3 SDRAM 仅支持选通信号为差分信号，不支持单根信号的数据选通信号。
ddr3_addr	output[13:0]	地址输入，为 ACTIVATE 命令提供行地址和 READ/WRITE 命令的列地址和自动预充电（A10），以便从某个 bank 的内存阵列里选出一个位置
ddr3_ba	output[2:0]	Bank 地址输入，定义 ACTIVATE、READ、WRITE 或 PRECHARGE 命令是对哪一个 bank 操作的
ddr3_ras_n ddr3_cas_n ddr3_we_n	output	命令输入，这三个信号，连通 cs_n，定义一个命令
ddr3_reset_n	output	复位，低电平复位，复位是异步的
ddr3_ck_p ddr3_ck_n	output[0:0] output[0:0]	时钟，差分时钟输入，所有控制和地址输入信号在 ck_p 上升沿和 ck_n 的下降沿交叉处被采样，输出数据选项（dqs_n、dqs_p）参考与 ck_p 和 ck_n 的交叉点。
ddr3_cke	output[0:0]	时钟使能：CKE 为高电平时，启动内部时钟信号、设备输入缓冲以及输出驱动单元。CKE 低电平时则关闭上述单元。当 CKE 为低电平时，可使设备进入 PRECHARGE POWER DOWN、SELF-REFRESH 以及 ACTIVE POWER DOWN 模式。CKE 与 SELF REFRESH 退出命令是同步的。在上电以及初始化序列过程中，VREFCA 与 VREF 将变得稳定，并且在后续所有的操作过程中都要保持稳定，包括 SELF REFRESH 过程中。CKE 必须在读写操作中保持稳定的高电平。在 POWER DOWN 过程中，除 CK_t，CK_c，ODT 以及 CKE 以外的所有输入缓冲都是关闭的。在 SELF

		REFRESH 过程中，除 CKE 以外的所有输入缓冲都是关闭的。在正时钟上升边沿采样。
ddr3_cs_n	output[0:0]	片选信号，当 CS_n 锁存为高电平时，所有的命令都被忽略。在正时钟上升边沿采样。
ddr3_dm	output[0:0]	输入数据掩码，dm 信号是作为写数据的掩码信号，当 dm 信号信号为低电平时，写命令的输入数据对应的位将被丢弃。dm 信号在 DQS 的两个条边沿都采样。
ddr3_odt	output[0:0]	On-Die Termination，片上终端电阻；ODT 信号可使得 DDR SDRAM 内部的 RTT_NOM 终端电阻。该设计通过允许 DRAM 控制器独立地打开/关闭任一或所有 DRAM 设备的终端电阻来改善存储器通道的信号完整性。 DRAM 通过 ODT 控制引脚为每个 DQ，DQS 和 DM 开启/关闭终端电阻。与其他输入命令不同，ODT 引脚直接控制 ODT 动作，不对其进行时钟采样。在自刷新模式下不支持 ODT。可以选择在 CKE 掉电期间通过模式寄存器启用 ODT 操作。 请注意，如果在掉电模式下启用 ODT，则在掉电期间可能无法关闭 VDDQ（I/O 供电），同时 DRAM 也会在读操作期间无法关闭。

1.6.2. 锁相环模块

锁相环模块的整体框图如下图 1.7 所示。

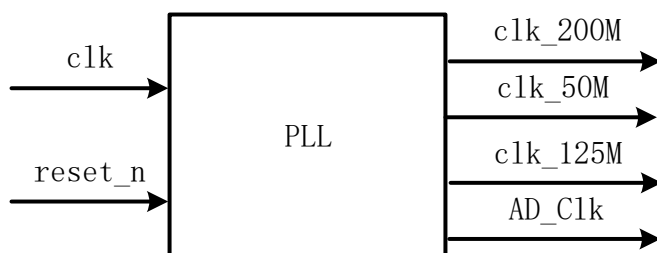


图 1.7 锁相环模块整体框图

我们的锁相环模块直接使用的 vivado 自带的 IP，输入信号为系统时钟 50M，输出四种时钟频率，第一种 clk_50M 作为 DDR3 模块外其他模块的基础的时钟，配合 pll_locked 信号使用，第二种 clk_200M 是提供给 DDR3 的工作时钟。第三种是 clk_125M 给到 DDS 模块使用。第四种是 AD_Clk 作为 ACM2108 的 ADC 时钟信号。模块调用代码如下所示：

```

pll pll
(
  // Clock out ports
  .clk_out1(clk_50M),      // output clk_out1
  .clk_out2(clk_200M),     // output clk_out2
  .clk_out3(clk125m),
  .clk_out4(AD_Clk),
  // Status and control signals
  .resetn(reset_n), // input resetn
  .locked(pll_locked), // output locked

```



```
// Clock in ports
.clk_in1(clk));    // input clk_in1
```

1.6.3. 8bit 转 16bit 模块

为了方便后续进行上位机进行数据采集，将 8bit 数据转换成 16bit 数据进行存储。8bit 转 16bit 模块的基本框图如下图 1.8 所示，端口列表如下表 1-2 所示。

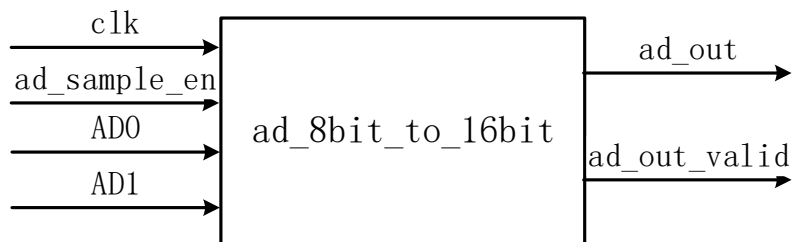


图 1.8 8bit 转 16bit 模块的基本框图

表 1-2 8bit 转 16bit 模块的端口列表

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

这里我们给出代码如下：

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

```
//用于仿真或产生测试数据
reg [7:0]adc_test_data;
//测试数据，当 ad_sample_en 为 1 时，锁相环生成的 50M 时钟每个周期使
adc_test_data 加 1
always@(posedge clk)
    adc_test_data <= ad_sample_en ? (adc_test_data + 1'b1) : 8'd0;

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

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

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

always @(posedge clk)
    ad_out_valid <= ad_sample_en;
endmodule
```

ACM2108 模块采集到的数据是无符号的数据，比如采集的波形为+5V~-5V 的正弦波，ADC 模块最终输出的数据就为 255~0，而我们上位机在分析数据的时候需要数据是有符号的，在这里我们进行的操作就是将 ADC 采集得到的数据加上 128，也就是将最高位取反，最后进行分析时将最高位作为符号位。

1.6.4. 串口指令接收和指令解析模块

这两个模块的作用，是把串口接收到的指令信息进行拆解和识别。从串口接收到指令数据后，uart_byte_rx 模块将串口数据从串行信号转为 8 位的并行数据 uart_rx_data。8 位的并行数据在 uart_cmd 模块中，被解析出地址、数据和使能信号。uart_cmd_rx 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。

这两个模块的信号接口框图如下图 1.9 所示，信号列表如下表 1-3 所示：

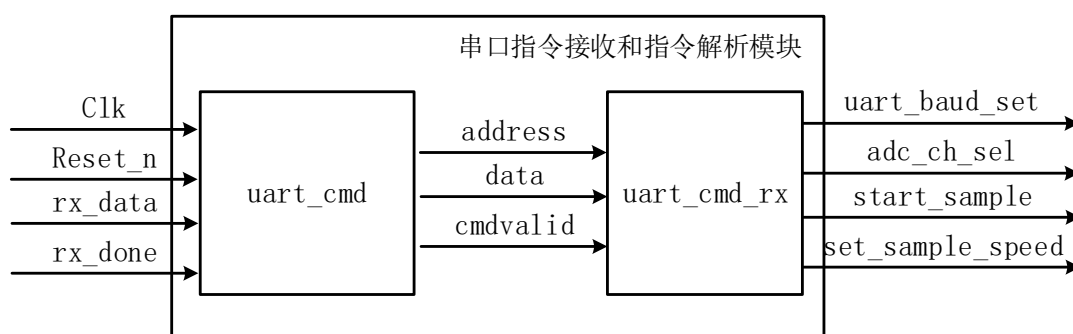


图 1.9 串口接收和指令解析模块

表 1-3 串口接收和指令解析模块信号列表

端口名	端口类型	描述
Clk	input	系统时钟 50MHz
Reset_n	input	模块复位信号
rx_data	input [7:0]	由串口接收模块接收到的 8 位数据信号
rx_done	input	串口接收一个字节结束标志信号
uart_baud_set	output[2:0]	串口波特率设置信号
adc_ch_sel	output [1:0]	ACM2108 模块通道选择信号（01：通道 1；10：通道 2）
set_sample_num	output [15:0]	ACM2108 模块采样深度设置信号
start_sample	output	ACM2108 模块开始采样标志信号
set_sample_speed	output [31:0]	ACM2108 模块采样速率设置信号

串口指令接收模块再在资料的盘 C 的第 13 课有过详细讲解，本次实验就是根据课程源码（14_uart_rx_ctrl_led）修改得到，这里将不做详细介绍。为了保证通信的可靠性和接收数据时的便利性，定义了对应的帧头和帧尾，只有接收到了正确到的帧头帧尾，才将接收到的指令给到指令解析模块进行分析，本次实验中定义帧头为 55 A5，帧尾为 F0。代码如下所示：

```

module uart_cmd(
    Clk,
    Reset_n,
    rx_data,
    rx_done,
    address,
    data,
    cmdvalid
);

input Clk;
input Reset_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] <= #1 rx_data;
    data_str[6] <= #1 data_str[7];
    data_str[5] <= #1 data_str[6];
    data_str[4] <= #1 data_str[5];
    data_str[3] <= #1 data_str[4];
    data_str[2] <= #1 data_str[3];
    data_str[1] <= #1 data_str[2];
    data_str[0] <= #1 data_str[1];
end

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

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] == 8'h55) && (data_str[1] == 8'hA5) &&
(data_str[7] == 8'hF0))begin
        data[7:0] <= #1 data_str[6];
        data[15:8] <= #1 data_str[5];
        data[23:16] <= #1 data_str[4];
        data[31:24] <= #1 data_str[3];
        address <= #1 data_str[2];
        cmdvalid <= #1 1;
    end
end
else
    cmdvalid <= #1 0;

endmodule
```

通过串口指令接收模块之后，得到 address、data 和 cmdvalid 信号输送给指令解析模块，其中 address 代表控制存储地址，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，03 是采样速率，04 是发送启动、采样通道号、采样深度同时设置，05 是串口波特率。data 是一个 32 位的值，其代表的是根据不同的指令，设置的对应值，比如，设置采样深度为 512，data 的值就是 00 00 01 00。cmdvalid 代表接收到的指令是有效的信号。指令解析模块代码如下所示：

```
module uart_cmd_rx(  
    clk,  
    reset,  
    uart_baud_set,  
    adc_ch_sel,  
    set_sample_num,  
    set_sample_speed,  
    start_sample,  
    cmdvalid,  
    cmd_addr,  
    cmd_data  
);  
  
input clk;  
input reset;  
input [31:0]cmd_data;  
input cmdvalid;  
input [7:0]cmd_addr;  
output reg [2:0]uart_baud_set;  
output reg [1:0]adc_ch_sel;  
output reg [15:0]set_sample_num;  
output reg start_sample;  
output reg [31:0]set_sample_speed;  
  
always@(posedge clk or posedge reset)  
if(reset)begin  
    uart_baud_set <= 3'd4; //默认 115200bps  
    adc_ch_sel <= 2'b00;  
    set_sample_num <= 16'd32768; //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];  
    end  
end
```

```

2: set_sample_num <= cmd_data[15:0];
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
5: uart_baud_set <= cmd_data[2:0];
default;;
endcase
end
else
    start_sample <= 1'b0;
endmodule

```

根据上述讲解，可以得知 ACM2108 的控制指令，由 8 个字节的数据组成，前两个字节 D0，D1 帧头用 55 A5，最后一个字节 D7 帧尾用 F0，表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，第四到第七个字节 D3~D6，标明的是设置的对应参数值。

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。帧格式如下表 1-4 所示：

表 1-4 数据帧格式表

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

这里，举例说明对应参数设置指令（ACM108 只能使用通道 0）：

采样通道如果是 2108 的第零通道，则设置为 55 A5 01 00 00 00 01 F0

采样通道如果是 2108 的第一通道，则设置为 55 A5 01 00 00 00 02 F0

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

如果采 65536 字节的数据，则设置为：55 A5 02 00 00 80 00 F0

启动采样命令：整个字节为 55 A5 00 00 00 27 0F F0

实际在启动指令的环节，我们既可以在串口上位机的输入界面按上述格式依次编辑和输入上方提到的指令，也可以先按上述格式要求同时编辑好 3 条指令，然后让上位机进行 1 次发送。

1.6.5. 状态机模块

状态机模块整体框图如下图 1.10 所示，端口列表如下表 1-5 所示。

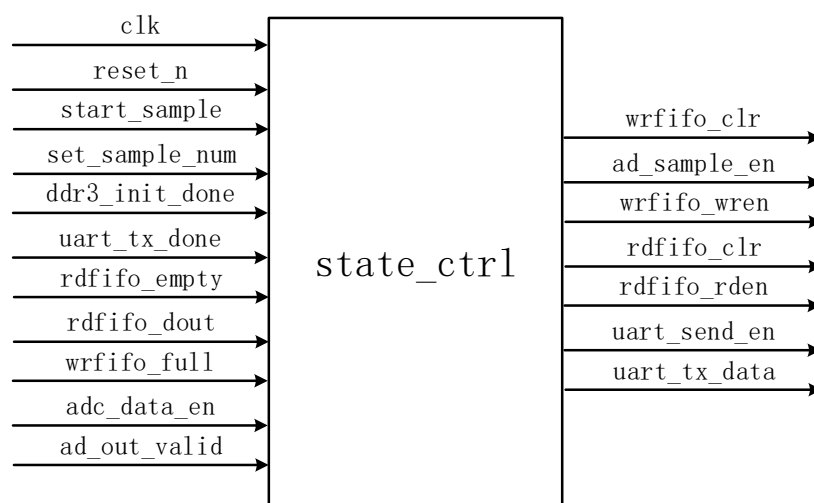


图 1.10 状态机模块的整体框图

表 1-5 状态机模块的端口列表

端口名	端口类型	描述
clk	input	系统时钟 50MHz
reset_n	input	模块复位信号
start_sample	input	ACM2108 模块开始采样标志信号
set_sample_num	input [15:0]	设置的采样深度，16 位计数，最大 65535
ddr3_init_done	input	DDR3 初始化完成标志信号
uart_tx_done	input	串口发送完成标志信号
rdfifo_empty	input	读 FIFO 的读空标识信号，用于标识当前 FIFO 是否为空（即 FIFO 内有无数据）
rdfifo_dout	input [15:0]	读 FIFO 的读数据输出，数据位宽为 16 位
wdfifo_full	input	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	input	ADC 采样结果存储使能信号
ad_out_valid	input	ADC 输出数据有效信号
wrfifo_wren	output	写 FIFO 的写数据使能控制信号，给高电平表示往 FIFO 写入数据，为避免写入数据的丢失，确保在 FIFO 非满（wrfifo_full=0）情况下写入数据
rdfifo_clr	output	读 FIFO 清空控制信号，给高电平表示执行清空，执行清空操作时，需保证给 3 个及以上个时钟（rdfifo_clk）周期的高电平
rdfifo_rden	output	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
uart_send_en	output	串口发送数据使能信号
uart_tx_data	output[7:0]	串口需要发送的 8 位数据

在这个模块中，我们定义了 11 个状态，分别是空闲状态、写 fifo 清零状态、ADC 采样状态、读 fifo 清零状态、延时过渡状态 1、延时过渡状态 2、低 8 位数据开始发送状态、低 8 位数据发送进行状态、高 8 位数据开始发送状态、高 8 位

数据发送进行状态和判断发送是否完成状态。

```
localparam IDLE = 4'd0;
localparam DDR_WR_FIFO_CLEAR = 4'd1;
localparam ADC_SAMPLE = 4'd2;
localparam DDR_RD_FIFO_CLEAR = 4'd3;
localparam DATA_SEND_DELAY1 = 4'd4;
localparam DATA_SEND_DELAY2 = 4'd5;
localparam DATA_SEND_LOW_START = 4'd6;
localparam DATA_SEND_LOW_WORKING = 4'd7;
localparam DATA_SEND_HIGH_START = 4'd8;
localparam DATA_SEND_HIGH_WORKING = 4'd9;
localparam DATA_SWITCH = 4'd10;
```

第一步：程序上电后，状态机进入空闲状态，同时，DDR3 及其 fifo 二端口模块，开启自动初始化模式。通过仿真可以知道，如果按 ACX720 开发板 Xilinx 自学教程相关内容对 fifo 和 u_mig_7series_0 进行设置，官方模型给出的 ddr3 的初始化完成(ddr3_init_done 拉高)时间，大约在上电后 0.11ms。当 ddr3 初始化完成，如果初始化完成并且收到开始采样的指令，则进入状态 1。

第二步：进入状态 1 后开始清除写 fifo 内的原始数据。进入清写 fifo 状态后，FPGA 发送三拍的拉高信号清写 fifo 指令，并且给出 10 拍的基本延时保证，当写 fifo 内的原始数据清除完毕后，fifo 内部会将写端 fifo_full 的信号拉低。如果收到写端 fifo_full 信号拉低,说明 fifo 已经不满，FPGA 可以开始向 fifo 内传递从 ACM2108 采集到的数据。

```
DDR_WR_FIFO_CLEAR: //1
begin
if(!wrfifo_full && (wrfifo_clr_cnt==9))
    state<=ADC_SAMPLE;
else
    state<=DDR_WR_FIFO_CLEAR;
end
```

清除写 fifo 的指令，由三拍延时信号拉高提供，之所以提供的延迟信号时间为 3 拍，是为了给清 FIFO 信号足够的拉高时间，以保证清空指令的可靠。

```
always@(posedge clk or posedge reset)begin
if (reset)
    wrfifo_clr<=0;
else if(dds3_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
```

在程序中 `reg[4:0]wrfifo_clr_cnt`; 信号为写 fifo 清零的状态计数和保持, 当进入写 fifo 清零状态后, 首先开始计数, 先保证计数完成, 再等待 `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。清写端 fifo 的控制信号是由计数器 (在前 3 个计数值将清除控制信号拉高) 产生 3 个时钟周期的高电平脉冲

第三步: 设定采样参数。数据的采样速率, 和 FPGA 的非 DDR3 工作时钟频率保持一致, 为每秒传递 50M 的 16 位数据。采样数据的数量设定, 项目开发时有两种备选方案, 一种是通过 VIO 进行设定, 这种方案较初级, 我们工程中采用的是更高级一点的, 就是前面提到过的, 通过串口指令的方式进行设定。

第四步: 进入数据采样状态即状态 3。可以说: `wrfifo_full` 拉低的状态, 吹响了数据采集的号角。在这个状态, 每发送完一个数, 则计数器加 1, 当计数器计数到和采样需求个数相同时, 跳转进入下一个状态, 开始清除读 fifo。

第五步: 一旦进入读 fifo 清零状态, 我们做和前面写 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。等延时结束并且接收到读 fifo 反馈的 `rdfifo_empty` 信号后, 进入下一个状态。

第六步: 下一个状态是延时过渡状态 1, 这个状态只持续一拍时间, 我们可以开启向读 fifo 发送一个脉冲的数据读使能信号 (`rdfifo_rden==1`)。再进入延时过渡状态 2。

第七步: 进入延时过渡状态 2 后, 关闭读 fifo 的数据读使能信号 (`rdfifo_rden==0`)。进入低 8 位数据开始发送状态, 则通知串口可以向外发送低 8 位数据了。接下来, 就可以按照串口的节奏, 进行数据的发送。

关于串口的数据发送, 在设计之初也有两种预备方案, 一种是通过总节拍数控制串口发送的结束, 另一种, 是利用串口发送完成后给出的 `tx_done` 信号和提取数据的判定条件形成关联, 每当一个串口字节发送完成后, 利用串口给出的 `tx_done` 信号, 切换到下一个字节的发送开始状态。这个 `tx_done` 信号直接控制和协调下一个小循环读 fifo 的 16 位数据提取工作, 提取的数据缓存工作, 缓存的数据拆解字节到串口寄存器的工作。在本工程中, 我们从节约串口时间成本的

店铺: <https://xiaomeige.taobao.com>

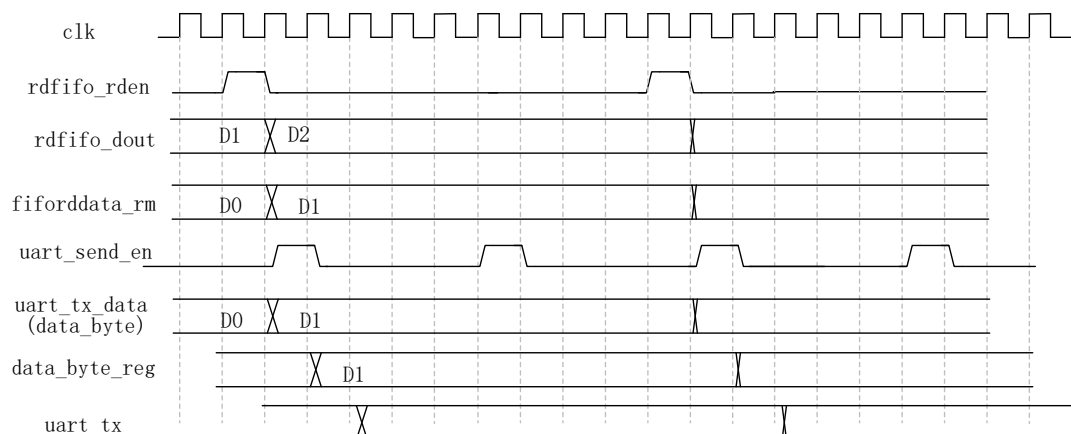
技术博客: <http://www.cnblogs.com/xiaomeige/>

官方网站: www.corecourse.cn

技术群组:

角度考虑，使用了方案 2，而没有纯粹追求更简单的串口控制方法。同时，单个字节发送完成的信号，也可以作为从发送低 8 位到发送高 8 位的判定信号条件。

接下来的几个步骤描述的是串口数据位的发送。由于 `rdfifo_rden` 的脉冲信号拉高后要到下一拍时钟上升沿才能让数据从 `rdfifo` 中读出，而读出的数据发送到串口的 `send_en` 脉冲信号要等数据稳定后下一拍生效才能发出正确的数据，所以，每个 16 位的数据，完成从 `fifo` 中提取到发送到串口，至少要三拍延时。



上图：在状态机使用条件下，从读 `fifo` 中读取数据，数据发送到串口的三拍延时的触发启动流程。

到最后，就是状态 10，状态 10 是个用来判断是否完成数据块发送的状态，如果确实数据块发送完成，则回到 `IDLE` 状态大循环收口关闭；如果没有达到数据总发送次数，则表明没有发送完，回到 `delay1` 状态，发送下一个数据，这也意味着本轮 16 位串口发送小循环收口关闭，进入下一个小循环。

1.7. 板级验证

1.7.1. 硬件连接

第一步：我们连接好 ACM2108 或者 ACM108，示波器，ACX720FPGA 开发板电源、下载器、串口连接线。程序烧写完成后，开发板的 `led0` 和 `led1` 的灯会点亮，此时表明：锁相环工作正常，`ddr3` 初始化正常。

ACM2108 模块完成并下载程序后开发板如下图 1.11 所示，ACM108 模块程序下载之后开发板如下图 1.12 所示：



图 1.11 bit 文件下载完成后开发板效果图（ACM2108 模块）

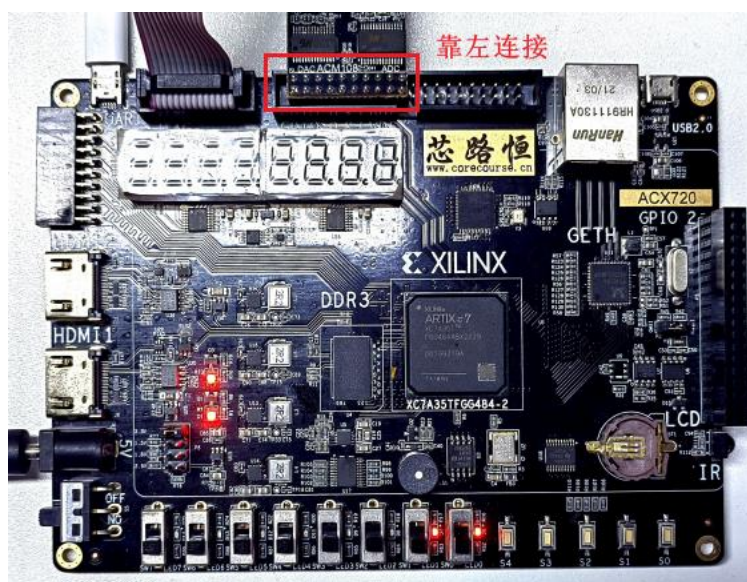


图 1.12 bit 文件下载完成后开发板效果图（ACM108 模块）

第二步：通过信号发生器发送 100Khz， V_{pp} 为 5V 的正弦波给到 ACM2108 模块。

第三步：启动串口上位机并按前面讲解的内容依次输入控制指令，串口设置如下图 1.13 所示，串口波特率为 115200。（ACM108 模块只能使用通道 0）



图 1.13 串口设置示意图

点击启动发送之后，串口会接受到数据，如下图 1.14 所示，将接收到的数据进行保存。

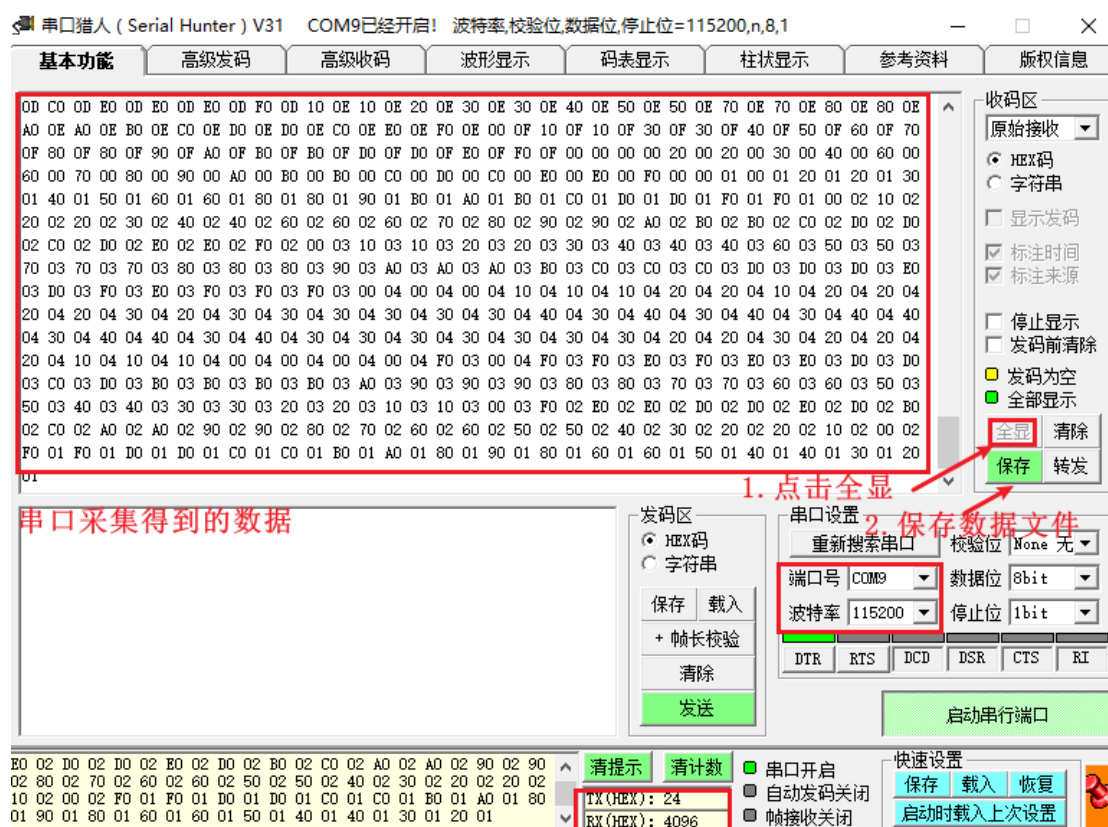


图 1.14 串口接收到的数据

可以看到，采集到的数据和下发的指令是匹配的。同时，各位读者也需明白，由于采样起始指令由上位机向 FPGA 发出，而发出指令的同时，ACM2108 模块提供的被采样的正弦波模拟量信号的相位是随机的，所以每完成一次数据采集后，得到的采集结果不同，也是完全正常而合理的，但最终得到的采集结果如果以绘图的方式复现，则应当是周期和幅值相同，而相位允许不同的正弦波。

1.7.2. 数据的简易分析与评定

前面的实验能够进行，能够得出结论，但是采样得到的数据是否有个别数据在某些特殊情况下发生数据丢失呢？仅凭人工，无法完成这个分析工作。要保证本系统在各个环节的稳定性，做到数据在采样全过程无一丢失，我们可以借助 matlab 的打印函数图形的功能完成对前面生成数据的分析和评定。一旦有任何的数据丢失，Matlab 绘制的波形将会直观而突兀的给出带有毛刺的采样数据绘制图形

下面给出 matlab 数据分析代码函数部分的简单介绍：

MATLAB 代码：
<pre>src_data=textread('C:\Users\Administrator\Desktop\003.txt', '%s'); src_data_hex=hex2dec(src_data) DATA_NUM = length(src_data_hex);</pre>


```
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>2048  
        wave_data(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256-4096;  
    else  
        wave_data(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256;  
    end  
end  
end
```

简单介绍一下 MATLAB 代码的含义。代码的第一行，指定的是数据文件的路径及格式，使用时需在执行 matlab 程序前替换第一个单引号内容，该部分内容描述了需要打印的数据文件位置、文件名称和格式，代码的第二行，是将 16 进制转为 10 进制的函数运算，代码的第三行，是计算转换后的数据的总长度，代码的第四行，是对采样得到的数据进行减半，以还原从 8bit 到 16bit 的数据原貌。下方的循环语句，用来判断纵坐标的极值是否超过 2048，如果超过 2048，则整个图形向下平移 4096 个单位，如果极值没有超出 2048，则保持不变。最后一句是执行 matlab 的打印指令。

为了保证我们的波形能够准确体现实验结果并且显示直观，我们采用 200k 频率下只进行 2048 个点的采样输出策略，只要保证这 2048 个点能够完全覆盖一个周期，就可以证明我们的设计任务是能够得到满足的。

打开 matlab 代码的工程文件，修改好文件路径，进行保存后点击运行，如下图 1.15 所示。

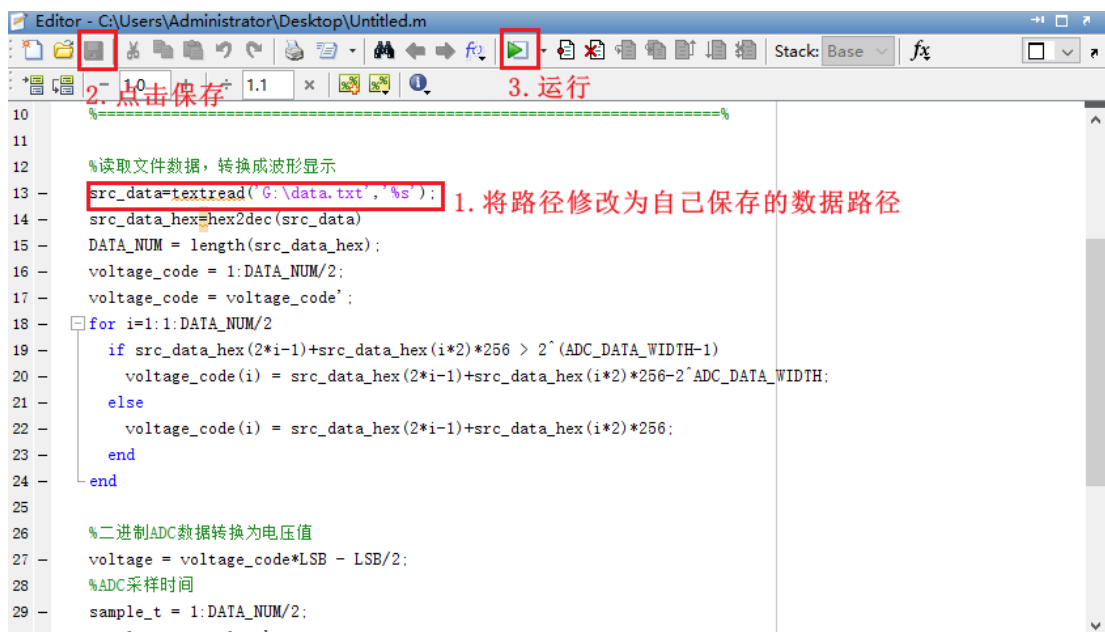


图 1.15 matlab 操作界面示意图

输出波形如下图 1.16 所示：

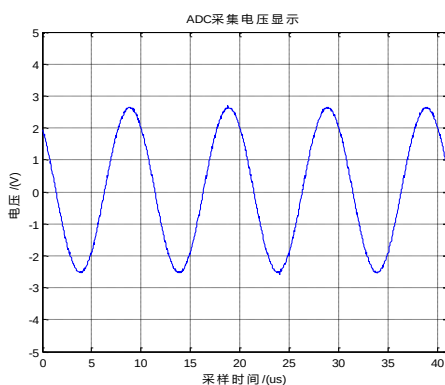


图 1.16 利用串口采集到的数据完成的 MATLAB 波形绘制

从上图中可以看出其电压范围在正负 2.5V 左右，频率为 100Khz（10us），和我们输出的信号是一致的，这样，工程的上板实验数据，就得到了初步验证。

我们给出 4096 点采样下，1Mhz 的发生频率，500khz 的发生频率，200khz 的发生频率和 50khz 的发生频率图，如下图 1.17 所示。

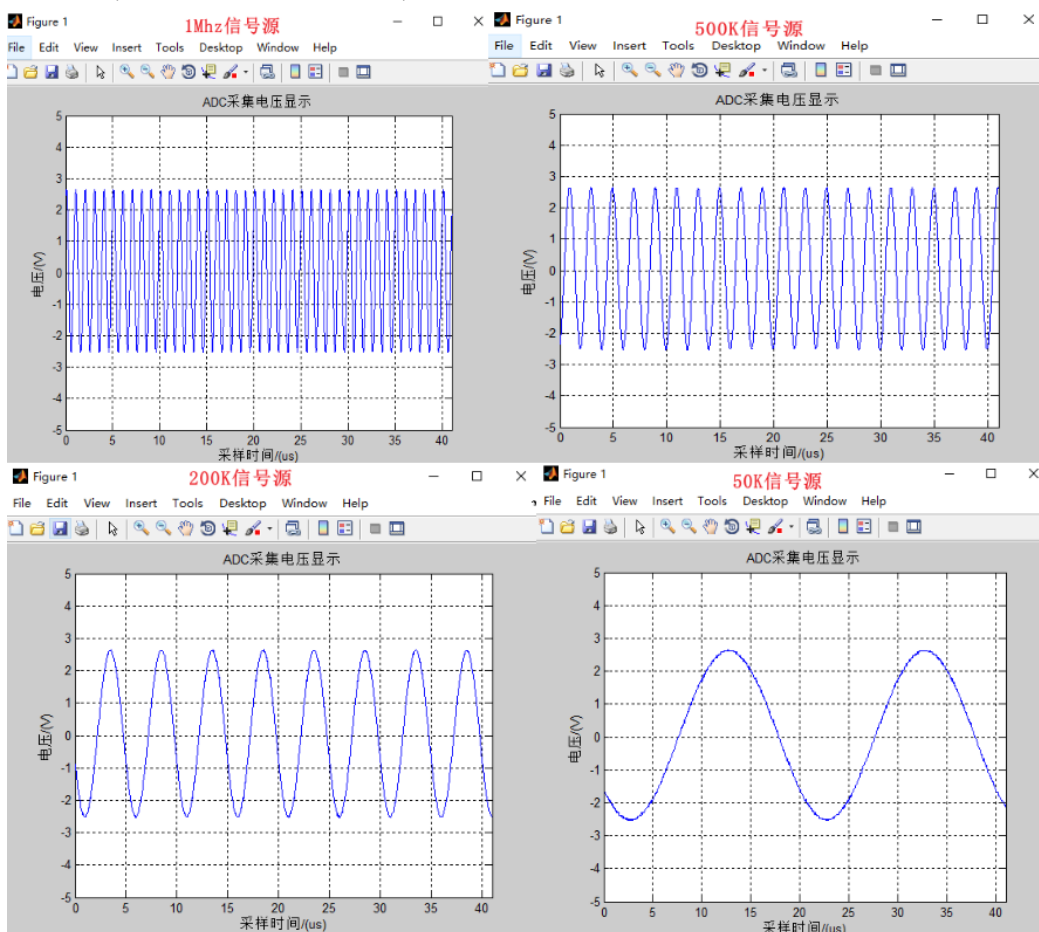


图 1.17 不同信号发生频率对比图

1.7.3. ACM2108 模块\ACM108 模块产生信号源

在我们提供的例程中，包含 DDS 模块，用户可以用杜邦线直接将 ACM2108 模块的 ADC 和 DAC 相连，进行数据采集，其中按键 S0 可以修改 DAC 的信号输出。连接方式如下图 1.18 所示，ACM108 模块的连接方式如下图 1.19 所示，将 ADC0 与 DAC0 相连，这样我们就可以通过 DAC0 输出信号给 ADC0 采集，注意不要连接到 GND。

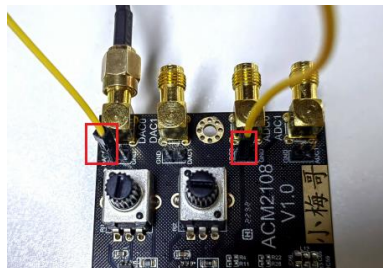


图 1.18 ACM2108 模块 ADC 和 DAC 连接示意图

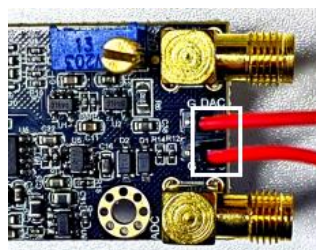


图 1.19 ACM108 模块 ADC 和 DAC 连接示意图

通过按下 S0 按键切换信号源，通过示波器观测采集到的信号源，如下图 1.20 所示为 DAC 产生的 125Khz 的正弦波信号。然后通过串口助手采集数据，MATLAB 分析数据得到波形图如下图 1.21 所示，操作步骤参考 1.7.1 和 1.7.2 节的内容。

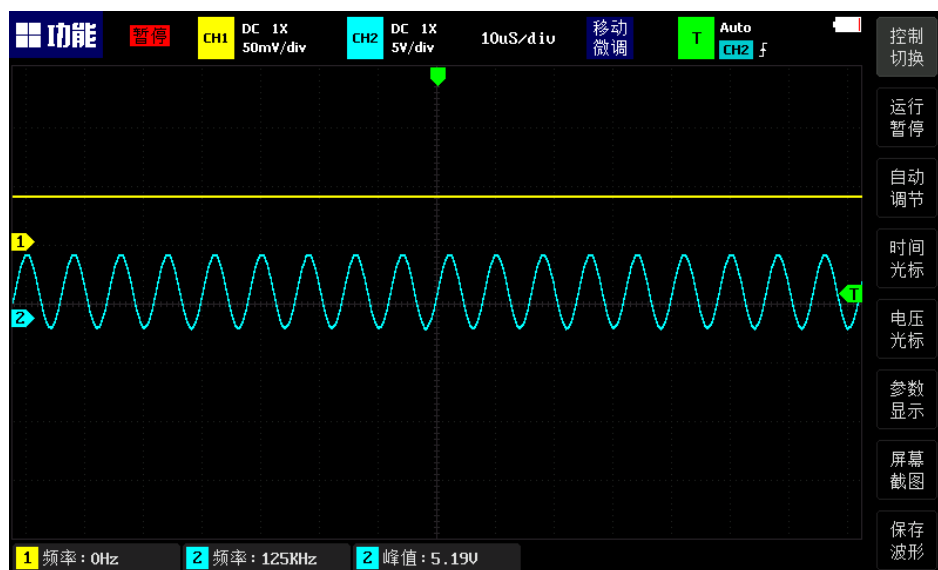


图 1.20 DAC 输出的信号

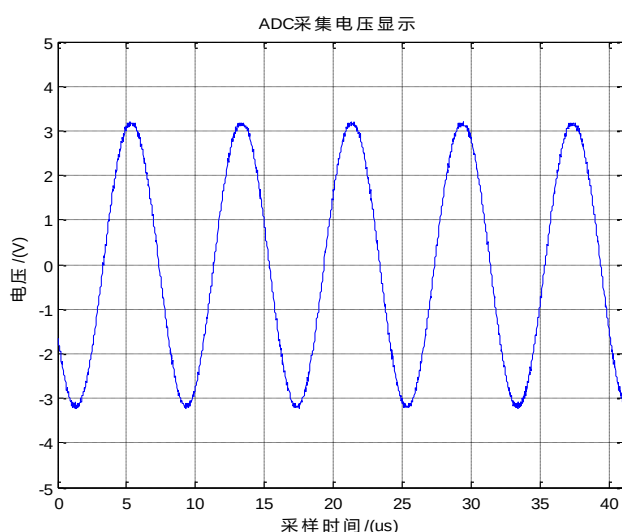


图 1.21 ADC 采集到的信号

当通过 MATLAB 观测到的信号不明显时，可以通过按键 S0 切换到更高频率的波形进行采集。按键切换次数与对应的波形的说明如下表 1-6 所示。

表 1-6 按键切换对应波形表

按键次数	对应波形	按键次数	对应波形	按键次数	对应波形
0	1.25K 正弦波	9	125K 正弦波	18	6.25M 正弦波
1	1.25K 方波	10	125K 方波	19	6.25M 方波
2	1.25K 三角波	11	125K 三角波	20	6.25M 三角波
3	12.5K 正弦波	12	625K 正弦波	21	12.5M 正弦波
4	12.5K 方波	13	625K 方波	22	12.5M 方波
5	12.5K 三角波	14	625K 三角波	23	12.5M 三角波
6	62.5K 正弦波	15	1.25M 正弦波		
7	62.5K 方波	16	1.25M 方波		
8	62.5K 三角波	17	1.25M 三角波		

1.8. 总结

通过本实验，我们介绍了如下知识点：

1. 数据采集的意义以及 ACM2108 模块的使用方式和硬件连接方法
2. 设计一种串口发码程序向 FPGA 发布工作模式配置指令的基本思路
3. 本工程系统框架、数据运行流程和状态机的状态转换策略
4. 使用 matlab 数据绘图分析方法的原因和一种配合本工程数据分析的程序代码