1. Ad9238 数据采集 DDR3 缓存串口发送实验

工程源码	针对 ACX720	|----02_设计实例 |--------acx720_ad9238_ddr3_uart.zip
	针对 ACX735	|----02_设计实例 |-------- acx735_ad9238_ddr3_uart.zip
相关视频课程		暂无相关视频课程
说明		无

1.1.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，我们今天要进行的实验，就是利用 ACM9238 数据采集卡实施数据采集。

1.2.工程目标、 任务和难点

本工程介绍如何通过利用 ACX720/ACX735 开发板上的 DDR3 资源，实现 ACM9238 模块的串口数据采集功能。

本工程在实验条件下，期望达到如下功能要求：

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 ACM9238 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，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。

本实验的难点：

1. 对 DDR3 的配置、信号控制和灵活运用；
2. 状态机的跳转设计；
3. 串口调试软件的运用和 MATLAB 的数据分析处理。

1.3.工程和代码讲解

1.3.1. 工程可行性分析

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：65MSPS *12bit = 780Mbps，由于ACM9238模块需要外部提供工作时钟， 所以FPGA应向ACM9238模块供应50M时钟。

（2） ACX720板载DDR3带宽：400*16*2 = 12,800Mbps，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780Mbps，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

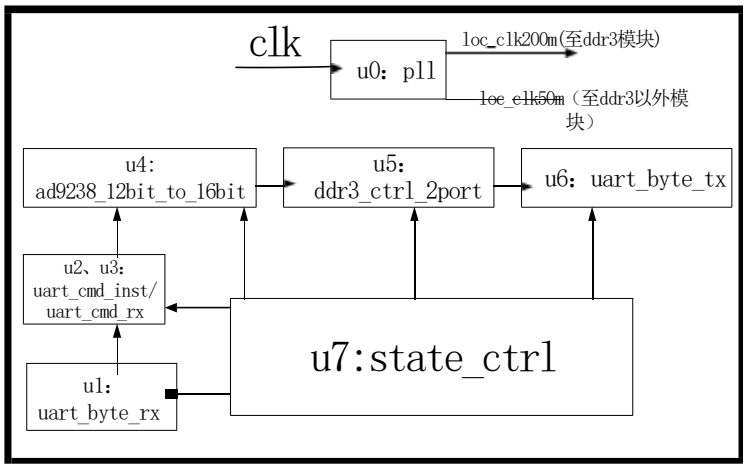
（3） 先考虑使用串口上传采集的ADC采集的数据， 常见的串口波特率9600bps，115200bps，这个比ADC采集数据的带宽要小的多， 想通过串口实时的将采集的数据传出去是不行， 会存在数据丢失问题，考虑到这点， 将ADC采集和串口上传数据分开处理，先ADC采样指定个数（或指定时间）的数据， 保存在板载DDR3，然后停止ADC采样，之后串口再将ADC采样数据依次上传到PC。

（4） DDR3 存储数据量： DDR3 内存 256MByte，ADC 数据位宽为 12bit，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 AD9238 的最高工作频率为 65M，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。

1.3.2. 工程总体框架

整体来说， 程序的硬件架构如下图所示：

其中， ACX720 模块内部的 FPGA 代码，设计了 7 个软件子模块，



内部各子模块功能如下：

u0、全程序 50M 时钟输入，各频率锁相环输出

u1、串口接收模块的指令接收功能。

u2、u3 接收到的指令进行翻译拆解， 指令分类。

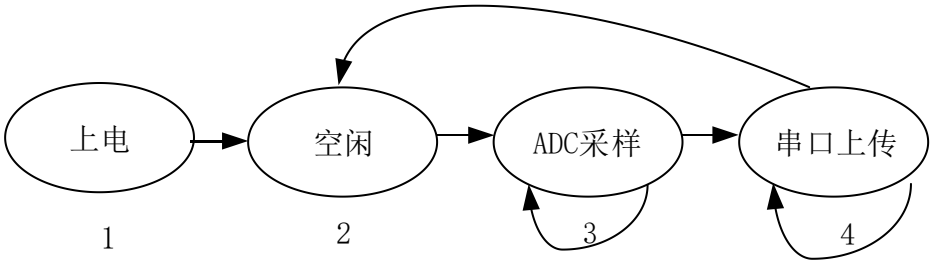
u4、ACM9238 数据输入 12bit 到 16bit 的转换。

u5、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。

u6、串口数据输出模块。

u7、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。

下面给出原理级的简易的状态转移图：

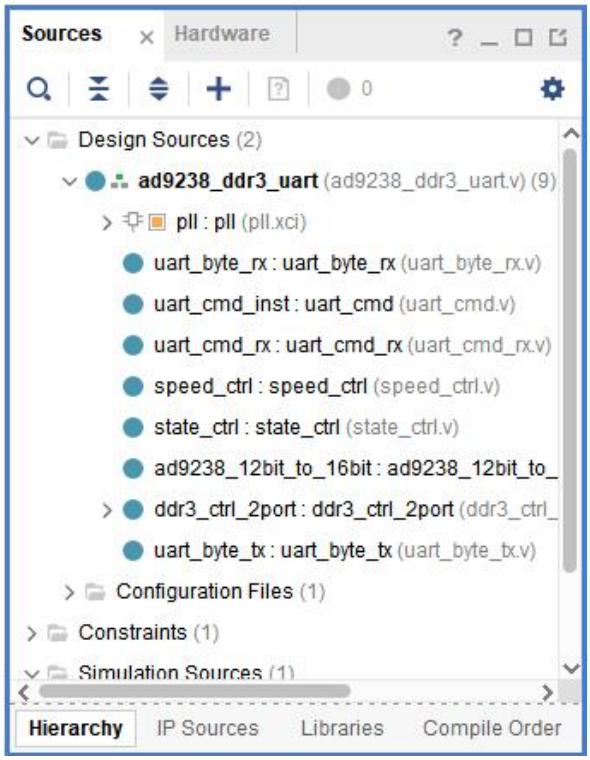


初步分析程序的状态机核心部分，主要分为上电， 空闲， 采样，串口上传四个部分。程序上电后，进入状态 2 空闲状态， 此时可以通过串口指令设置采样个数， 设置采样频率，下发采样开始的指令。

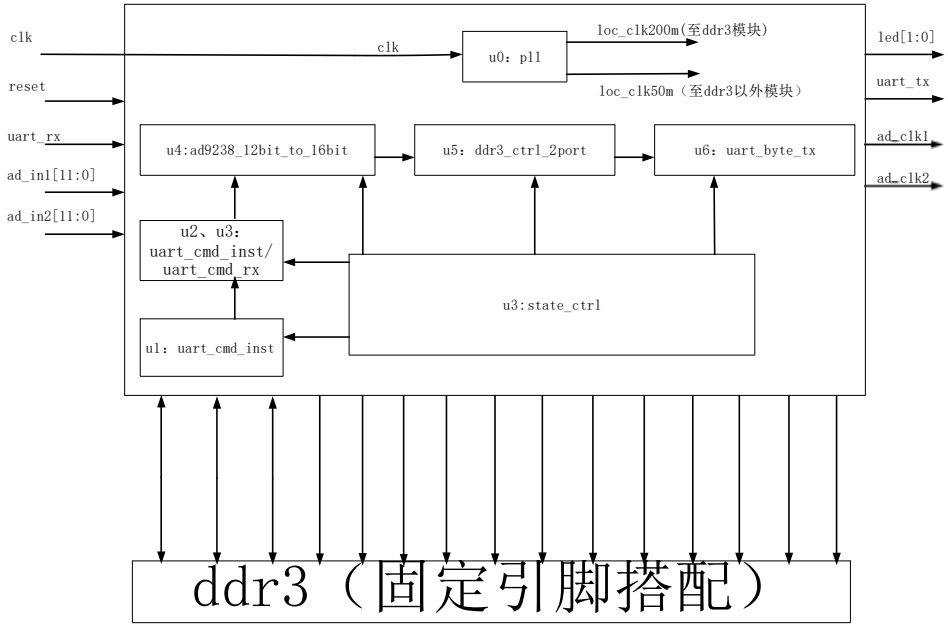
FPGA 收到采样开始指令后，进入状态 3 开始进行 ADC 采样。这时候， FPGA 利用计数器对采样数据个数进行计数，采样的同时数据直接储存在 DDR3 中。如果未达到采样个数， 则进行状态 3 的循环， 如果达到采样个数， 则进入状态 4 串口上传。如果串口上传完成， 则返回空闲状态，如果串口上传未完成， 则继续进行串口上传。

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， 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来保证这些核心状态的正常运行。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，我们在后面程序部分进行详述。

1.3.3. 程序顶层模块及对外接口功能设计



为了保证代码的可读性和风格的美观，我们在顶层模块中，仅进行端口定义，端口信号连线，简单的组合逻辑处理和模块例化工作。



顶层模块声明了输入输出接口以及数据流向，这里，我们对模块的端口列表给出说明，如有需要， 请对照代码进行深入理解：

顶层输入输出端口列表及说明

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
ad_in1	input[11:0]	AD9238 模块的 12 位数据输入引脚第一通道
ad_in2	input[11:0]	AD9238 模块的 12 位数据输入引脚第二通道
ad_clk1	output	AD9238 模块的第一通道时钟
ad_clk2	output	AD9238 模块的第二通道时钟
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

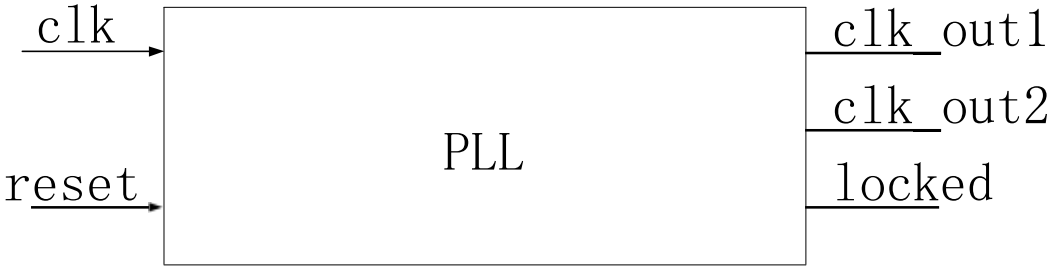
		中，VREFCA 与 VREF 将变得稳定， 并且在后续所有的操作过程中都要保持稳定，包括 SELF REFRESH 过程中。 CKE 必须在读写操作中保持稳定的高电平。在 POWER DOWN 过程中，除 CK_t，CK_c，ODT 以及 CKE 以外的所有输入缓冲都是关闭的。在 SELF REFRESH 过程中， 除 CKE 以外的所有输入缓冲都是关闭的。在正时钟上升边沿采样。
ddr3_cs_n	output[0:0]	片选信号，当 CS_n 锁存为高电平时， 所有的命令都被忽略。在正时钟上升边沿采样。
ddr3_dm	output[0:0]	输入数据掩码，dm 信号是作为写数据的掩码信号，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 信号可使能 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。

u0:pll 实现的功能就是产生各个模块需求的时钟。这里， 由于 xilinx 的内部 buf 寄存器的使用要求，如果使用 pll，则要求锁相环的输入时钟具有独占性而不能和其他模块共用。如果使用 mmcm，则有更优秀的灵活度。

1.3.4. 各子模块设计

1.3.4.1. 锁相环模块

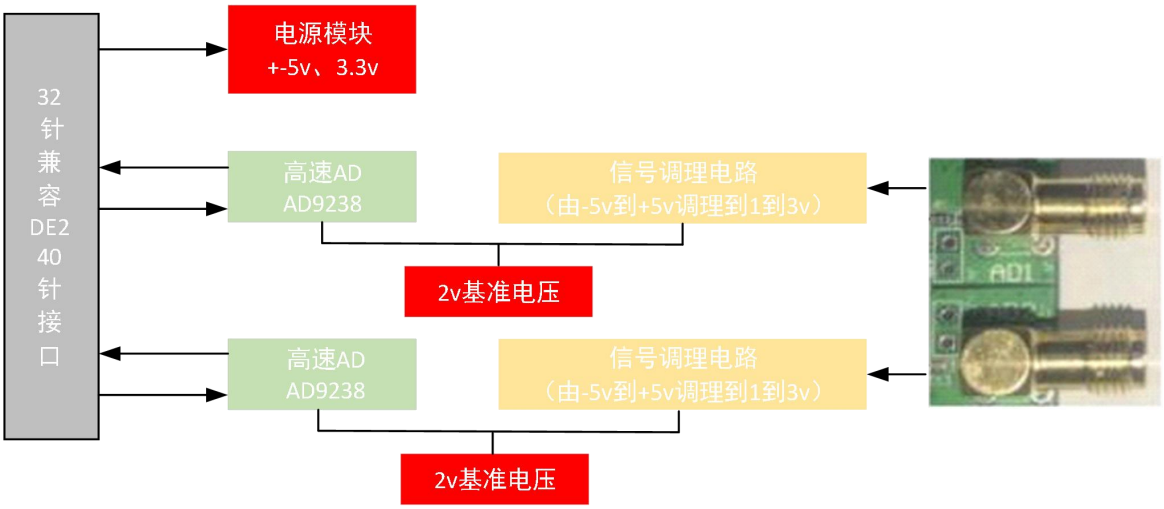
```
pll pll (
  // Clock out ports
  .clk_out1  (loc_clk50m  ), // output clk_out1
  .clk_out2  (loc_clk200m ), // output clk_out2
  .reset     (!reset_n    ), // input reset
  .locked    (pll_locked  ), // output locked
  // Clock in ports
  .clk_in1   (clk         )  // input clk_in1
);
```



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，输出两种时钟频率， 一种 loc_clk50m 作为 DDR3 模块外其他模块的基础的时钟，配合 pll_locked 信号使用，另一种 loc_clk200m 是提供给 DDR3 的工作时钟。

1.3.4.2. 数据采集卡 ACM9238 简介及工程代码 12bit 转 16bit 模块

ACM9238模块使用2片ADI公司的高性能12位6Msps采样率的模数转换芯片AD9238，配合前端信号调理电路,实现了双通道65M采样率的高速采样功能，模块模拟电压输入范围为±5V、采用12位并行数据接口。只需要fpga提供时钟信号,开发板就能在每个时钟周期从模块12位并行数据接口上读取到一个数据。借助其较高的采样速率，配合PC上位机和模拟量幅度变送装置，可以满足绝大多数领域的数据采样需求。下面给出ACM9238模块与FPGA的连接对应关系：



ACM9238模块使用32针排母与开发板母板连接， 支持直接连接的开发板包括但不限于小梅哥出品的AC620教学板，ACX720教学板、 AC6103、Starter入门板， 友晶科技出品的DE2、DE1、DE1_SoC、DE0-Nano-SoC等。

在引脚绑定的时候， 我们要特别注意ACM9238模块的LSB引脚和MSB引脚。与大多数芯片不同， ACM9238模块对应的LSB是数据位最高位，而对应的MSB是数据位的最低位。如果在绑定引脚芯片的时候没有注意，则有可能会得到没有规律的输出数据。

计算机业界，端表示数据在存储器中的存放顺序。大端与小端是两种数据的存储方式！大端方式将高位存放在低地址， 小端方式将高位存放在高地址。（比如将16位数0x1234存放在地址0x00,0x01两个连续地址中时，按照大端方式应该0x00中存放0x12，而0x01中存放0x34，小端方式0x00中存放0x34，而0x01中存放0x12）

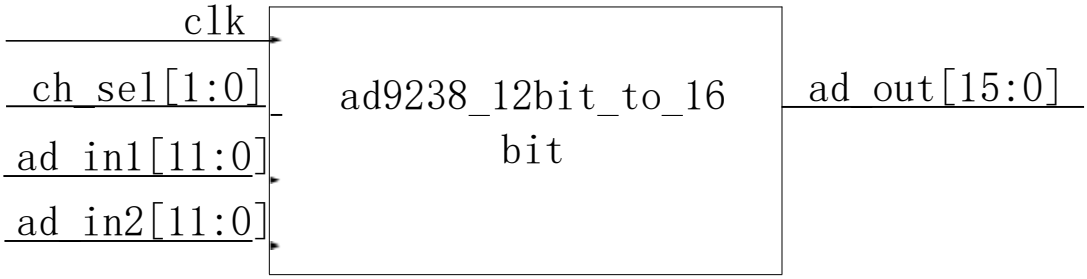
如果我们对引脚采用相同的理论，则写出来的12bit数据和引脚绑定的高位和地位， 将呈现镜像关系。比如，我们引脚输出的12bit数据是12’b1011_0111_1111,则实际反应的模拟采样值是12‘b1111_1110_1101,如果模拟量再提高一个分度值，则得到

12’b1111_1110_1110。如果我们错误的处理了大端和小端，误认为存储值和管脚输出值一一对应，则模拟量再提高一个分度值，得到的12bit数据是12’b1011_1000_0000,这样， 就会丢失数据采样的规律， 得到错误的结果。

数据采集卡采集到的12bit数据不便于计算机存储， 由于计算机对数据进行分析、存储的时候都以8位或16位数据作为统一的存储标准，则我们需要将12bit数据转换成16bit数据进行存储。 这里我们给出代码如下：

```
module ad9238_12bit_to_16bit(
  clk,
  ch_sel,
  ad_in1,
  ad_in2,
  ad_out
);

  input clk;
  input  [1:0]ch_sel;
  input[11:0] ad_in1;
  input[11:0] ad_in2;
  output[15:0] ad_out;
  reg[15:0] ad_out;

  always @(posedge clk)
  if(ch_sel == 2'b01)
    ad_out<={4'd0,ad_in1};
  else if(ch_sel == 2'b10)
    ad_out<={4'd0,ad_in2};
endmodule
```

本模块的功能， 是将输入的 12bit 数据， 转换为 16bit 数据。



关于这里的数据采集和处理，有两种方案， 第一种是在此处认为采集的线信号是无符号数，我们在前面补 4 个 0 位扩宽成为 16bit 数据存储，后面使用 matlab 进一步处理。另一种是把采集到的线信号当作有符号数来处理， 后面使用 matlab 作另一种方案处理。最后的目标是：我们有理有据的得出函数信号发生器发出的波形即可。本工程中，我们采用的第一种方案。

1.3.4.3. 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。

在这里，我们给出指令解析的代码和讲解：

```
always@(posedge loc_clk50m or posedge g_reset)
if(g_reset)begin
  uart_baud_set <= 3'd4; //默认115200bps
  adc_ch_sel <= 2'b00;
  set_sample_num <= 16'd32768;
  start_sample <= 1'b0;
end
else if(cmdvalid)begin
  case(cmd_addr)
      0: start_sample <= 1'b1;
      1: adc_ch_sel <= cmd_data[1:0];
      2: set_sample_num <= cmd_data[15:0];
      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;
```

AD9238 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：

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：

采样通道如果是 9238 的第一通道，则设置为 55 A5 01 00 00 00 01 F0

采样通道如果是 9238 的第二通道，则设置为 55 A5 01 00 00 00 02 F0

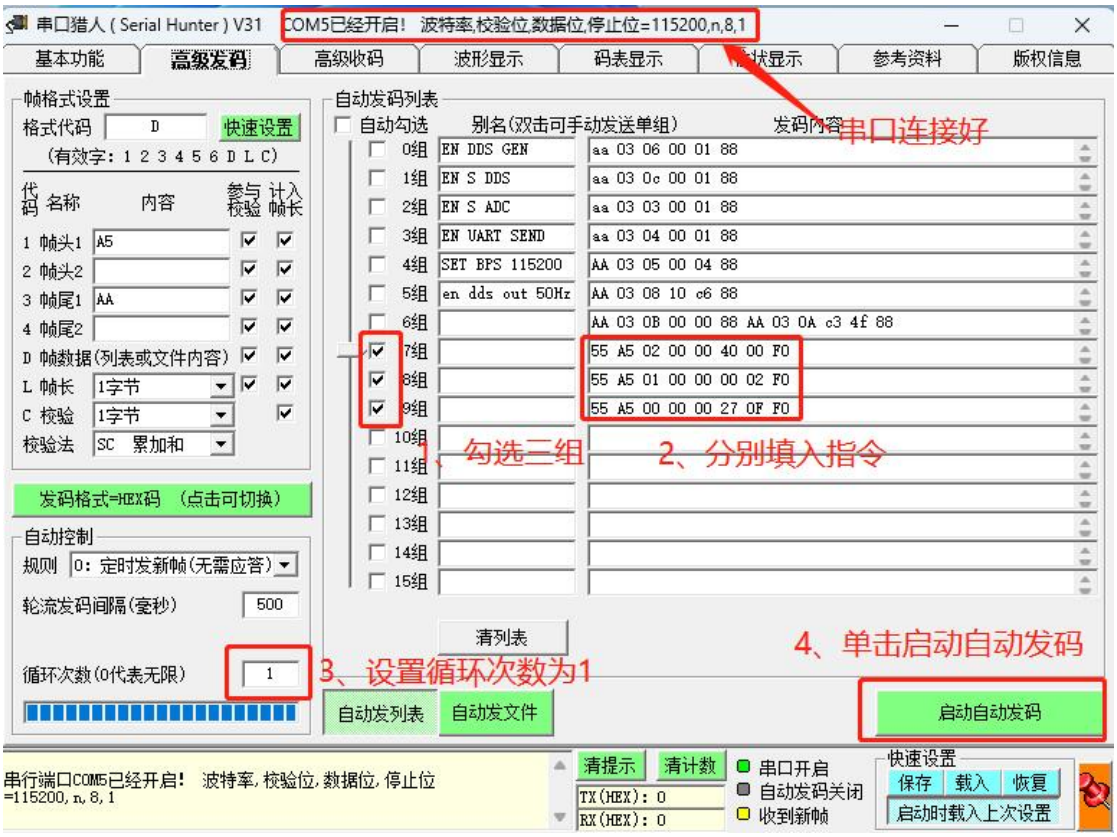
如果采 512 字节的数据， 则设置为： 55 A5 02 00 00 01 00 F0

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

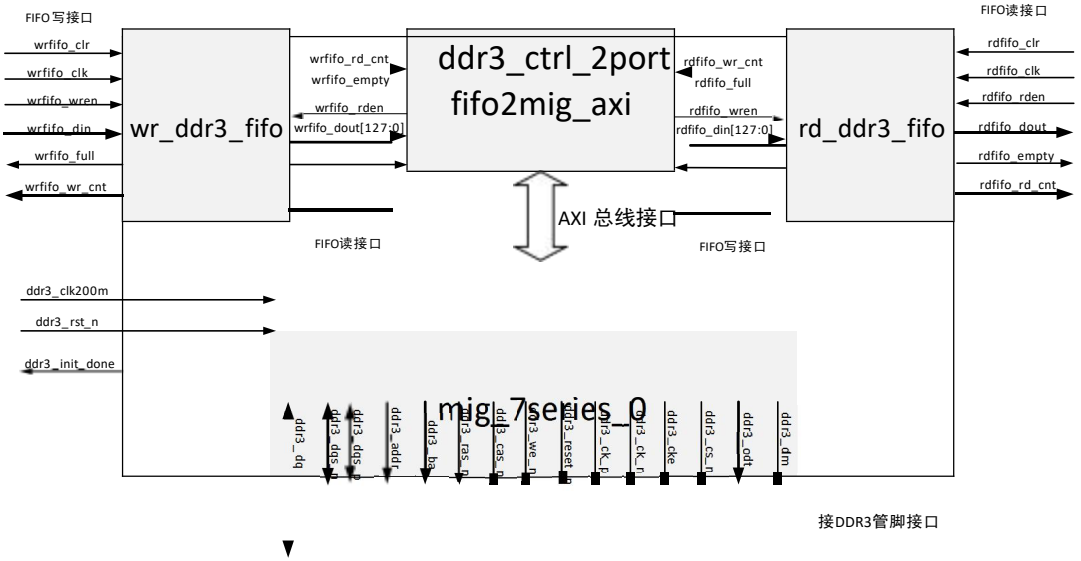
我们在下方给出软件的典型用法



点击启动自动发码后就可以开始采集数据。

1.3.4.4. DDR3 二端口模块简介

1.3.4.4.1. DDR3 二端口模块整体结构框图



1.3.4.4.2. DDR3 二端口模块参数及端口描述

ddr3_ctrl_2port 模块参数说明

参数名	描述
DW	模块写 FIFO 的写数据和读 FIFO 的读数据位宽， 即数据 wrfifo_din 和 rdfifo_dout 的位宽，设置值需被 128 整除
WR_ADDR_BEGIN	写数据存储空间的起始地址，1 个地址对应的数据位宽为（DW） bit
WR_ADDR_END	写数据存储空间的终止地址，1 个地址对应的数据位宽为（DW） bit
RD_ADDR_BEGIN	取数据存储空间的起始地址，1 个地址对应的数据位宽为（DW） bit
RD_ADDR_END	取数据存储空间的终止地址，1 个地址对应的数据位宽为（DW） bit

ddr3_ctrl_2port 模块端口说明

端口名	方向	描述
ddr3_clk200m	I	提供给 DDR3 控制器的基本工作时钟，要求 200MHz
ddr3_rst_n	I	提供给 DDR3 控制器的复位信号，低电平时为复位
ddr3_init_done	O	DDR3 控制器初始化及校准完成标识信号，为高表示初始化及校准成功
写 FIFO 的写接口，用户往 DDR3 写入数据的接口		
wrfifo_clr	I	写 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（wrfifo_clk）周期的高电平
wrfifo_clk	I	写 FIFO 的写操作工作时钟
wrfifo_wren	I	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
wrfifo_din	I	写 FIFO 的写数据信号，数据位宽为 DW
wrfifo_full	O	写 FIFO 的写满标识信号， 用于标识当前 FIFO 是否有被写满
wrfifo_wr_cnt	O	写 FIFO 的写数据计数，表示当前 FIFO 中缓存数据的个数
读 FIFO 的读接口，用户向 DDR3 读取数据的接口		
rdfifo_clr	I	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_clk	I	读 FIFO 的读操作工作时钟
rdfifo_rden	I	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_dout	O	读 FIFO 的读数据输出，数据位宽为 DW，
rdfifo_empty	O	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_rd_cnt	O	读 FIFO 的读数据计数，表示当前 FIFO 中缓存数据的个数
接 DDR3 管脚接口（仿真中与 ddr3_model 对接）		
ddr3_dq[15:0]	IO	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[1:0] ddr3_dqs_p[1:0]	IO	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[13:0]	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[2:0]	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	0	命令输入，这三个信号，连通 cs_n，定义一个命令
ddr3_reset_n	0	复位，低电平复位，复位是异步的
ddr3_ck_p ddr3_ck_n	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	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	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	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	0	On-Die Termination，片上终端电阻；ODT 信号可使能 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。

模块例化模板，参数的设置可查看上述参数描述，端口信号的连接可查看上述端口列表描述。

```
ddr3_ctrl_2port #(
  .DW              ( 32    ),
  .WR_ADDR_BEGIN   ( 0     ),
```

```
  .WR_ADDR_END    ( 1024 ),
  .RD_ADDR_BEGIN  ( 0    ),
  .RD_ADDR_END    ( 1024 )
)ddr3_ctrl_2port_inst
(
  //clock reset
  .ddr3_clk200m    (ddr3_clk200m   ),//input
  .ddr3_rst_n      (ddr3_rst_n     ),//input
  .ddr3_init_done  (ddr3_init_done ),//output
  //wr_fifo Interface
  .wrfifo_clr      (wrfifo_clr     ),//input
  .wrfifo_clk      (wrfifo_clk     ),//input
  .wrfifo_wren     (wrfifo_wren    ),//input
  .wrfifo_din      (wrfifo_din     ),//input  [DW-1:0]
  .wrfifo_full     (wrfifo_full    ),//output
  .wrfifo_wr_cnt   (wrfifo_wr_cnt  ),//output [15:0]
  //rd_fifo Interface
  .rdfifo_clr      (wrfifo_clr     ),//input
  .rdfifo_clk      (wrfifo_clk     ),//input
  .rdfifo_rden     (wrfifo_wren    ),//input
  .rdfifo_dout     (wrfifo_din     ),//output [DW-1:0]
  .rdfifo_empty    (wrfifo_full    ),//output
  .rdfifo_rd_cnt   (wrfifo_wr_cnt  ),//output [15:0]
  //DDR3 Interface
  // Inouts
  .ddr3_dq         (ddr3_dq        ),//inout  [15:0]
  .ddr3_dqs_n      (ddr3_dqs_n     ),//inout  [1:0]
  .ddr3_dqs_p      (ddr3_dqs_p     ),//inout  [1:0]
  // Outputs
  .ddr3_addr       (ddr3_addr      ),//output [13:0]
  .ddr3_ba         (ddr3_ba        ),//output [2:0]
  .ddr3_ras_n      (ddr3_ras_n     ),//output
  .ddr3_cas_n      (ddr3_cas_n     ),//output
  .ddr3_we_n       (ddr3_we_n      ),//output
  .ddr3_reset_n    (ddr3_reset_n   ),//output
  .ddr3_ck_p       (ddr3_ck_p      ),//output [0:0]
  .ddr3_ck_n       (ddr3_ck_n      ),//output [0:0]
  .ddr3_cke        (ddr3_cke       ),//output [0:0]
  .ddr3_cs_n       (ddr3_cs_n      ),//output [0:0]
  .ddr3_dm         (ddr3_dm        ),//output [1:0]
  .ddr3_odt        (ddr3_odt       ) //output [0:0]
);
```

1.3.4.4.3. DDR3 二端口模块使用说明

从模块的整体结构框图可以看出，里面包含 4 个子模块，其中 wr_ddr3_fifo 和 rd_ddr3_fifo 为双时钟 FIFO IP，mig_7series_0 是 Memory Interface Generator (MIG 7 Series) IP，fifo2mig_axi 模块是芯路恒开发的接口转换模块。其中，

（1）mig_7series_0 模块 IP 的详细配置参阅“小梅哥 Xilinx FPGA 自学教程”中“DDR 控制器 Example Design 的使用”章节文档资料。

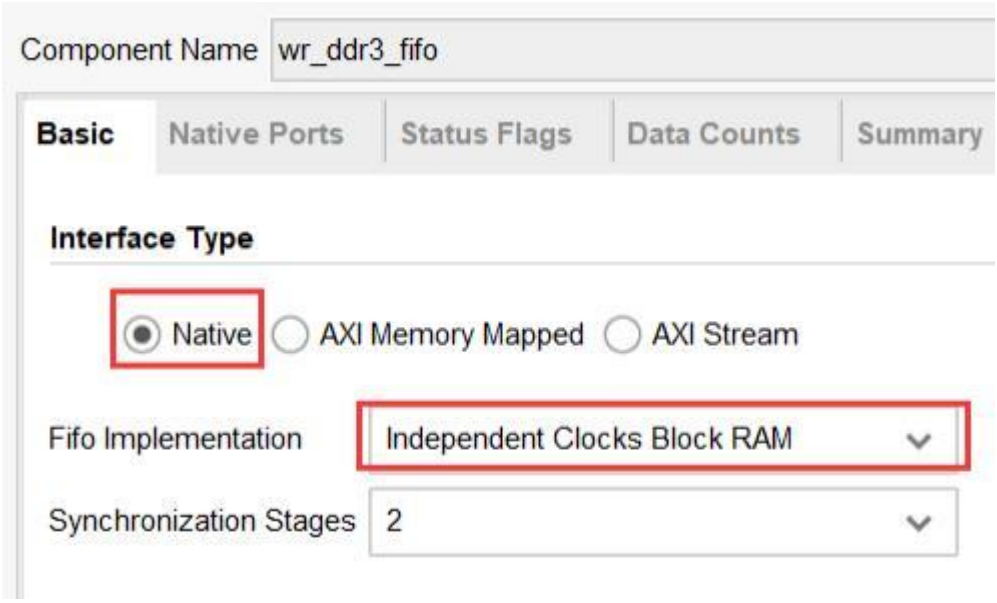
（2）fifo2mig_axi 模块的设计开发参阅“小梅哥 Xilinx FPGA 自学教程”中“基于 DDR3 的串口传图帧缓存系统设计实现”章节中关于该模块的文档资料，模块代码和对应的仿真文件有文件包附带，名称如下：

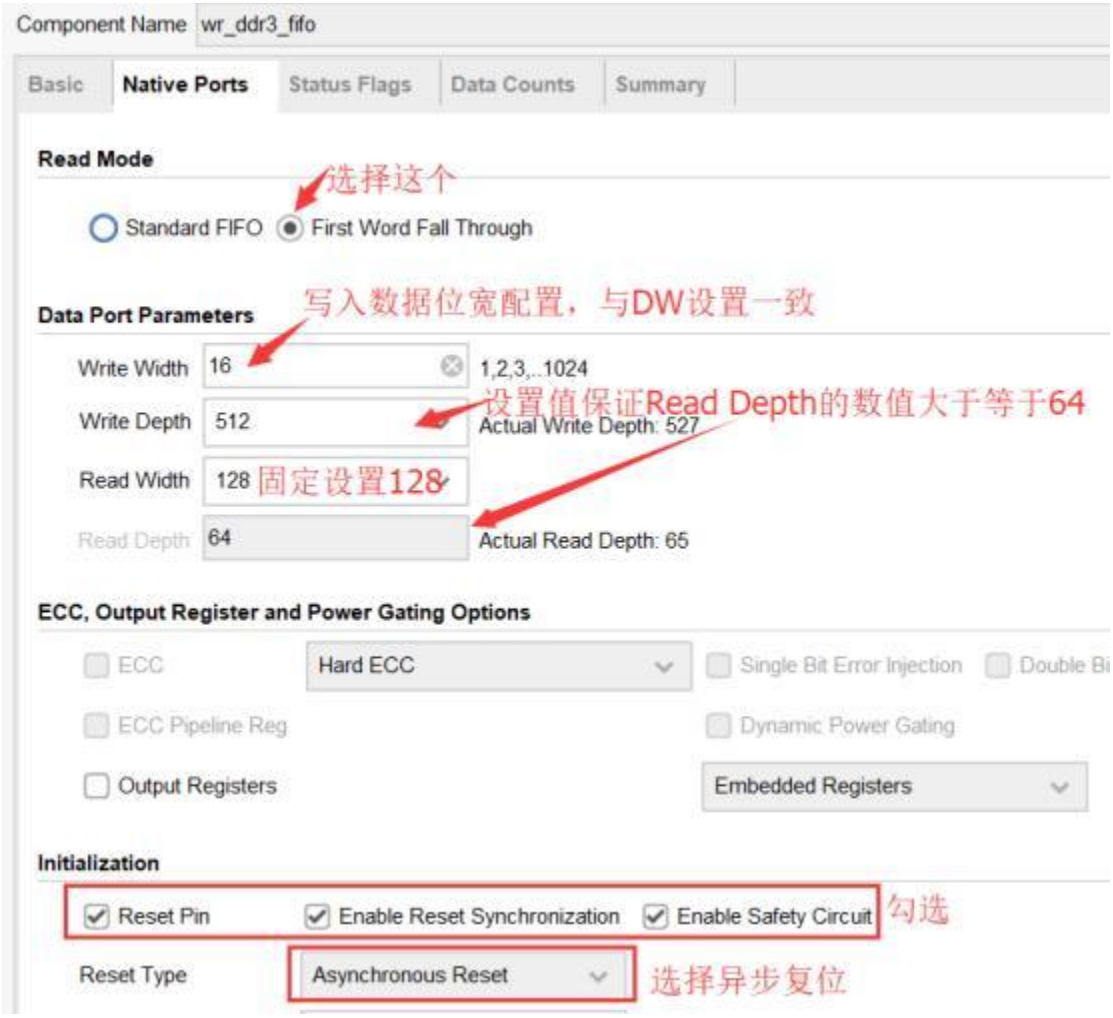
设计文件：fifo2mig_axi.v

仿真文件：fifo2mig_axi_tb.v

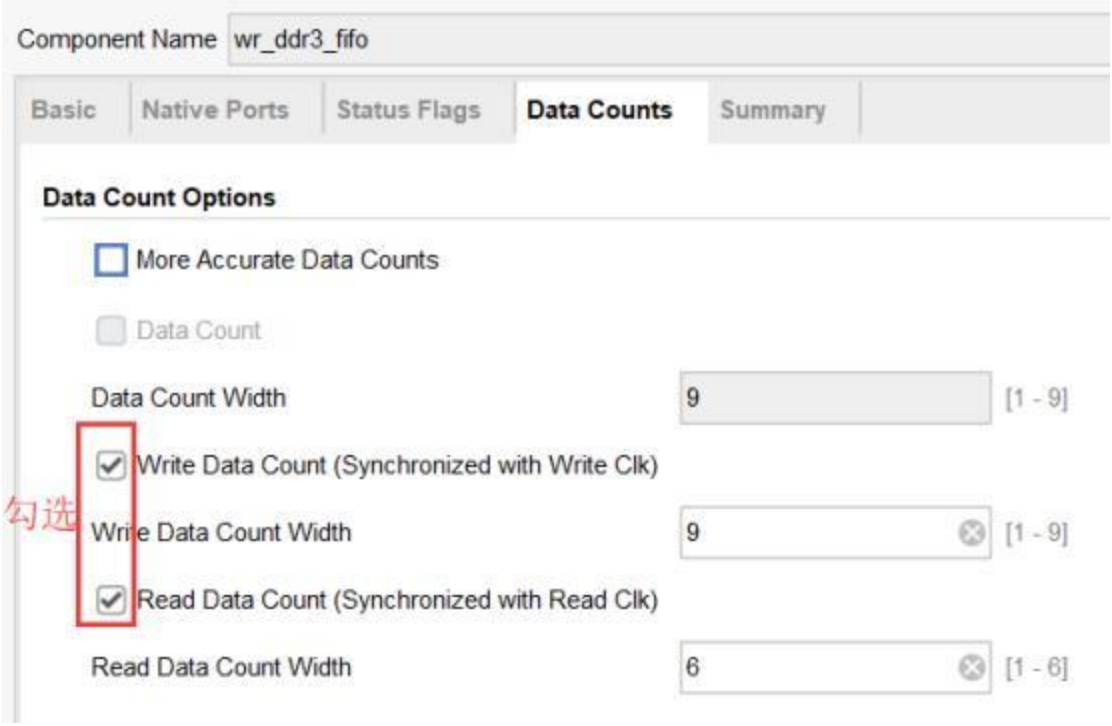
（3）wr_ddr3_fifo 是双时钟 FIFO IP，具体配置如下，

接口选择 Native，FIFO 实现选择 Independent Clocks Block RAM；



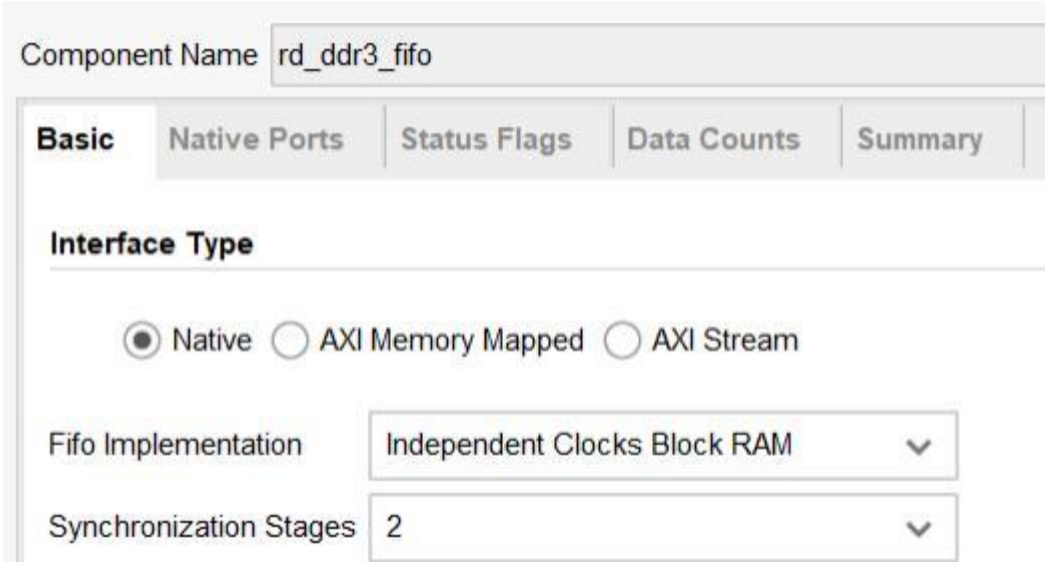

Component Name wr_ddr3_fifo
Basic
Native Ports
Status Flags
Data Counts
Summary
Read Mode
选择这个
Standard FIFO
First Word Fall Through
Data Port Parameters
写入数据位宽配置，与DW设置一致
Write Width
16
1,2,3,..1024
Write Depth
512
设置值保证Read Depth的数值大于等于64
Actual Write Depth: 527
Read Width
128 固定设置128
Read Depth
64
Actual Read Depth: 65
ECC, Output Register and Power Gating Options
ECC
Hard ECC
Single Bit Error Injection
ECC Pipeline Reg
Dynamic Power Gating
Output Registers
Embedded Registers
Initialization
Reset Pin
Enable Reset Synchronization
Enable Safety Circuit
勾选
Reset Type
Asynchronous Reset
选择异步复位

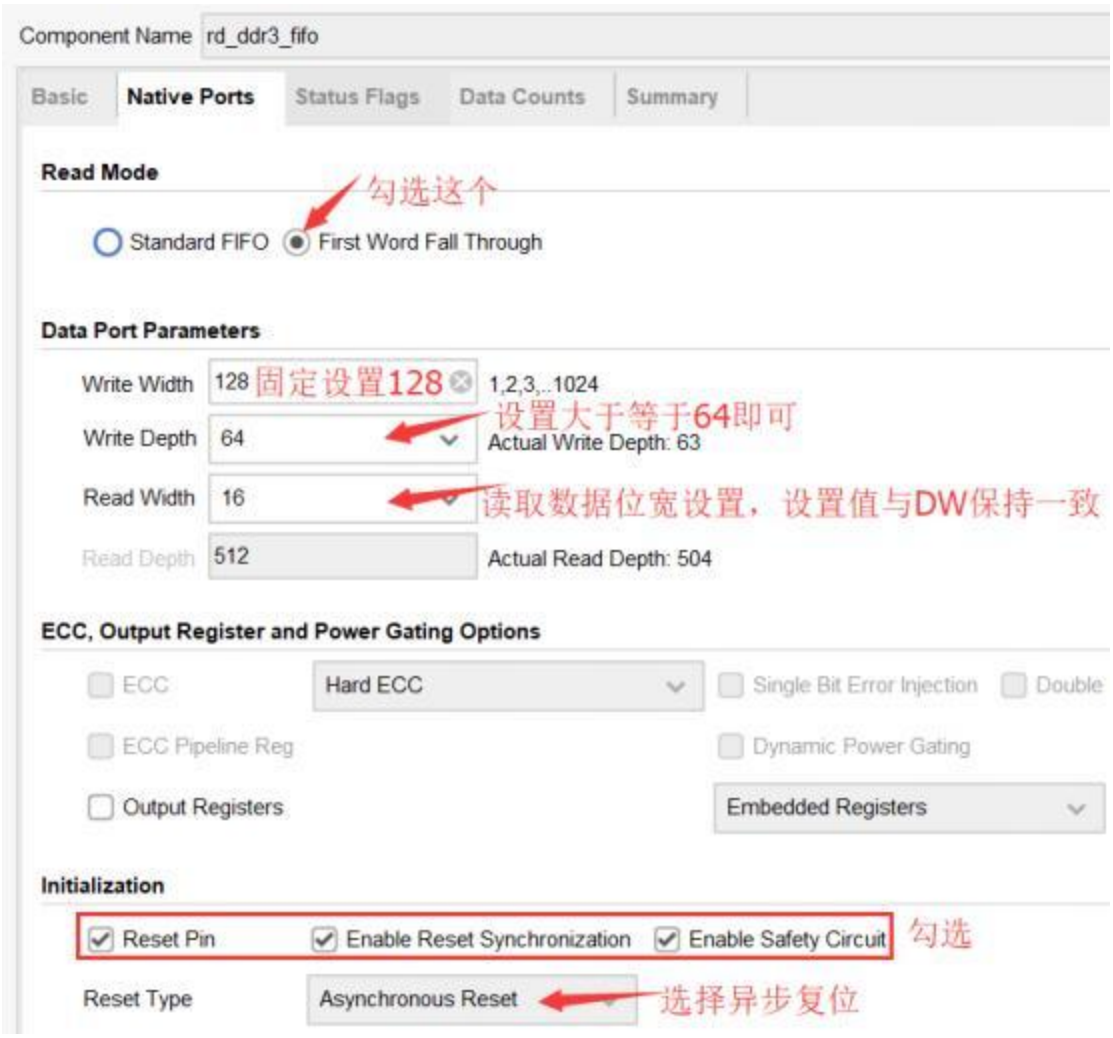

Component Name wr_ddr3_fifo
Basic
Native Ports
Status Flags
Data Counts
Summary
保持默认配置
Optional Flags



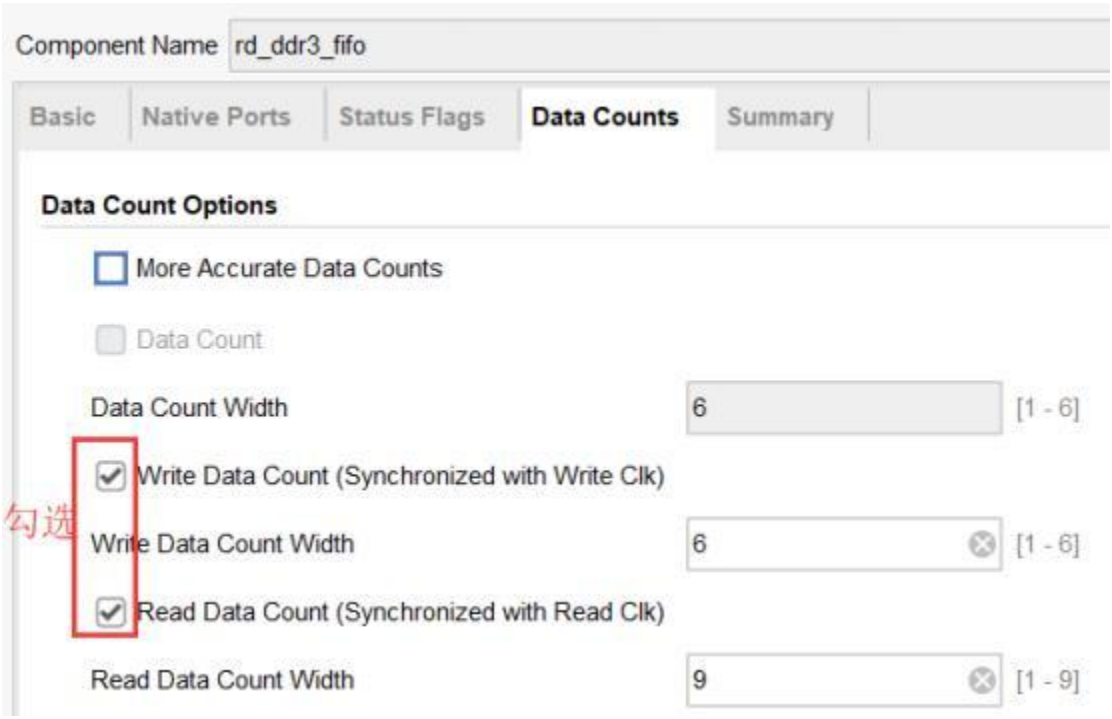
（4）rd_ddr3_fifo 是双时钟 FIFO IP，具体配置如下，

接口选择 Native，FIFO 实现选择 Independent Clocks Block RAM；



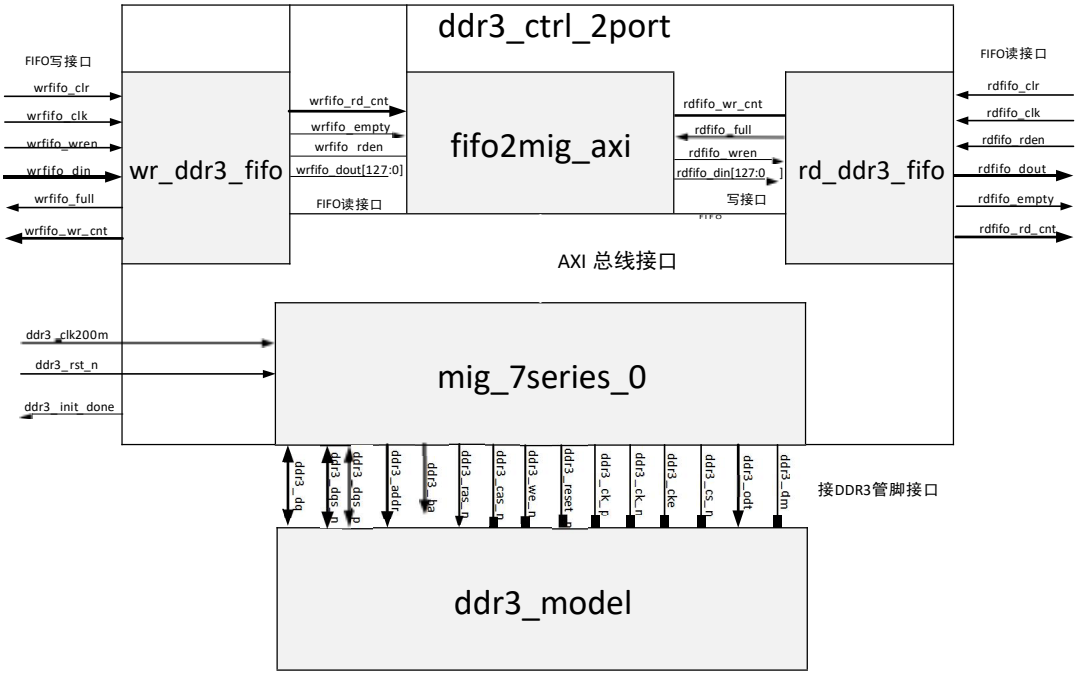

Component Name rd_ddr3_fifo
Basic
Native Ports
Status Flags
Data Counts
Summary
Read Mode
勾选这个
Standard FIFO
First Word Fall Through
Data Port Parameters
Write Width
128 固定设置128
1,2,3,..1024
Write Depth
64
设置大于等于64即可
Actual Write Depth: 63
Read Width
16
读取数据位宽设置，设置值与DW保持一致
Read Depth
512
Actual Read Depth: 504
ECC, Output Register and Power Gating Options
ECC
Hard ECC
Single Bit Error Injection
Double
ECC Pipeline Reg
Dynamic Power Gating
Output Registers
Embedded Registers
Initialization
Reset Pin
Enable Reset Synchronization
Enable Safety Circuit
勾选
Reset Type
Asynchronous Reset
选择异步复位


Component Name rd_ddr3_fifo
Basic
Native Ports
Status Flags
Data Counts
Summary
这里保持默认配置
Optional Flags



1.3.4.4.4.DDR3 二端口模块仿真说明

双端口 FIFO DDR3 控制器模块的仿真需要注意的是， 需要将 DDR3 仿真模型（即 ddr3_model 模块） 例化到 tb 文件中。例化框图可参考如下



只需要将接 ddr3 管脚接口的信号与 ddr3_model 模型端口信号一一连接即可。具体例程可参考“小梅哥 Xilinx FPGA 自学教程”中“基于 DDR3 的串口传图帧缓存系统设计实现”章节中的工程。需要注意的是需要将 ddr3_model.v 文件和 ddr3_model_parameters.vh 文件复制粘贴到需要仿真的工程下， 然后添加仿真文件 ddr3_model.v。相关代码随文件包附

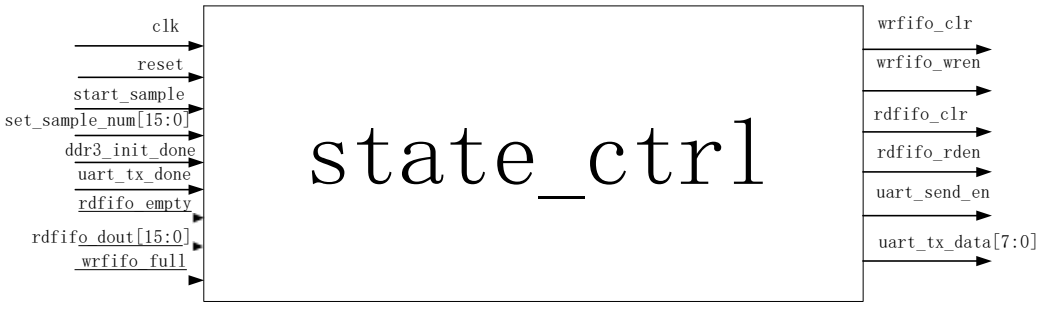
带，文件名如下：
ddr3_model.sv
ddr3_model_parameters.vh

关于ddr3_ctrl_2port模块和模块仿真文件ddr3_ctrl_2port_tb 的代码随文件包附带，文件名如下：
设计文件：ddr3_ctrl_2port.v
仿真文件：ddr3_ctrl_2port_tb.v

下图是对 ddr3_ctrl_2port 模块仿真的波形图，注意波形中 ddr3_init_done 信号大概在 0.1ms 之后一点出现有低电平变高电平，如果超过这个很长时间没有变为高电平，那说明 DDR3 初始化可能存在问题，需要检查 ddr3_model 模块是否例化好， 相关信号的数据位宽是否正确。

1.3.4.5. 状态机模块介绍

我们先给出状态机模块的输入输出框图



1.4.程序仿真

在程序仿真之前，我们先解除工程主文件的代码相关注释行以启用仿真模式。

```
213 `ifdef SIM
214 //SIM 用于仿真或产生测试数据，可在通过添加`define SIM 进行宏定义
215 reg [11:0]adc_test_data;
216    //*//测试数据，当wrfifo_wren为1时，锁相环生成的50M时钟每个周期使adc_test_data加1
217 always@(posedge loc_clk50m)
218  adc_test_data <= wrfifo_wren ? (adc_test_data + 1'b1) : 12'd0;
219    //*//测试数据，ad_out 在adc_test_data变化后无条件发生变化，判断adc_test_data的最高位
220    //*//是否为1，决定是正数还是负数后，进行4位扩充，扩充后进行拼接
221 assign ad_out = {{4{adc_test_data[11]}},adc_test_data}; //有符号数据位宽扩展
222
223 `else
```

在进行仿真的时候， 我们编写的 tb 文件需要加入 ddr3_model 如下：

```
`timescale 1ns/1ns
`define CLK_PERIOD 20

module ad9238_ddr3_uart_tb();
------ ------
ddr3_model ddr3_model(
  .rst_n   (ddr3_reset_n),
  .ck      (ddr3_ck_p   ),
  .ck_n    (ddr3_ck_n   ),
  .cke     (ddr3_cke    ),
  .cs_n    (ddr3_cs_n   ),
  .ras_n   (ddr3_ras_n  ),
  .cas_n   (ddr3_cas_n  ),
  .we_n    (ddr3_we_n   ),
  .dm_tdqs(ddr3_dm     ),
  .ba      (ddr3_ba     ),
  .addr    (ddr3_addr   ),
  .dq      (ddr3_dq     ),
  .dqs     (ddr3_dqs_p  ),
  .dqs_n   (ddr3_dqs_n  ),
  .tdqs_n  (            ),
  .odt     (ddr3_odt    )
);

endmodule
```

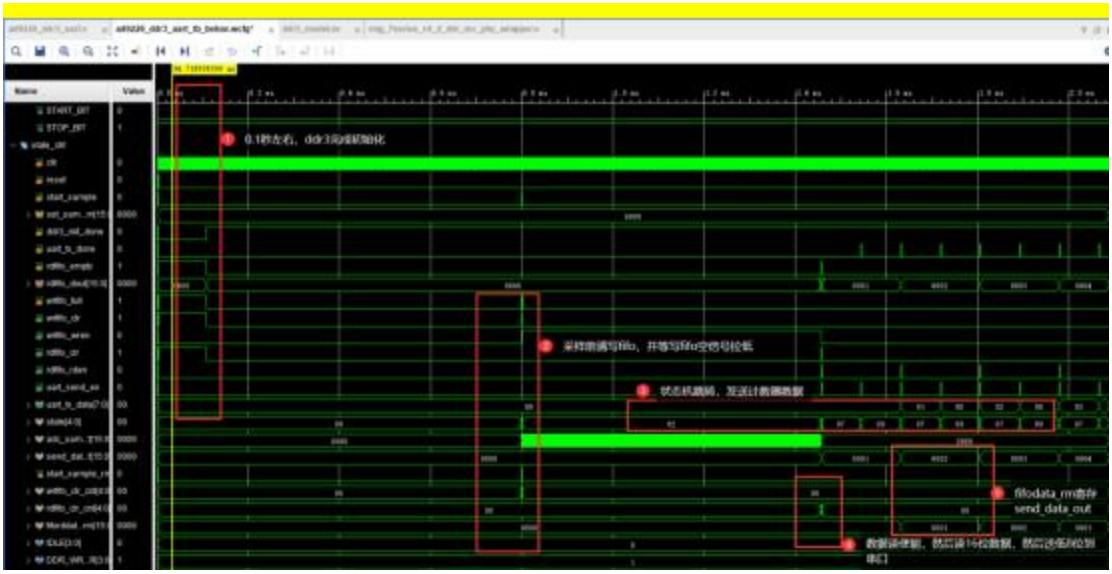
ddr3_model 是用来在仿真中替代真实的硬件 ddr3 工作的一个仿真模型，它可以完美的模拟出硬件 ddr3 初始化过程。

```
initial clk= 1;
always#(`CLK_PERIOD/2) clk = ~clk;

initial begin
  reset_n=1'b0;
  #(`CLK_PERIOD*10);
  reset_n=1'b1;
  uart_send_en = 1'b0;
  uart_tx_data = 8'd0;
  @(posedge ad9238_ddr3_uart.ddr3_init_done);//等待DDR3初始化完成
  #(`CLK_PERIOD*10+1);
  //启动采集,仿真串口发送55 A5 00 00 00 00 00 F0
  uart_tx_data = 8'h55;
  uart_send_en = 1'b1;//-----发送55
  #(`CLK_PERIOD);
  uart_send_en = 1'b0;
  @(posedge uart_tx_done);

  #(`CLK_PERIOD*10+1);

  uart_tx_data = 8'hA5;
  uart_send_en = 1'b1;//-----发送A5
  #(`CLK_PERIOD);
  uart_send_en = 1'b0;
  @(posedge uart_tx_done);

  #(`CLK_PERIOD*10+1);

  uart_tx_data = 8'h00;
  uart_send_en = 1'b1;//-----发送00
  #(`CLK_PERIOD);
  uart_send_en = 1'b0;
  @(posedge uart_tx_done);

  #(`CLK_PERIOD*10+1);

  uart_tx_data = 8'h00;
  uart_send_en = 1'b1;//-----发送00
  #(`CLK_PERIOD);
  uart_send_en = 1'b0;
  @(posedge uart_tx_done);

  #(`CLK_PERIOD*10+1);
```

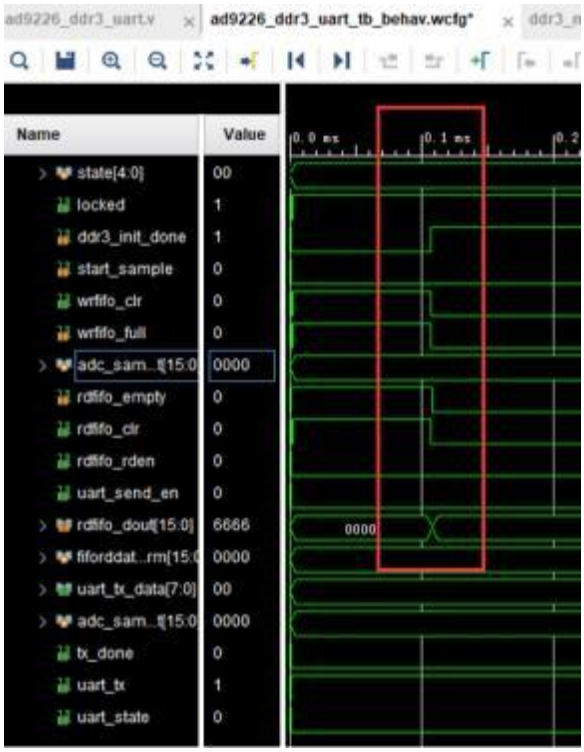
```
    uart_tx_data = 8'h00;
    uart_send_en = 1'b1;//-----发送00
    #(`CLK_PERIOD);
    uart_send_en = 1'b0;
    @(posedge uart_tx_done);

    #(`CLK_PERIOD*10+1);

    uart_tx_data = 8'h00;
    uart_send_en = 1'b1;//-----发送00
    #(`CLK_PERIOD);
    uart_send_en = 1'b0;
    @(posedge uart_tx_done);

    #(`CLK_PERIOD*10+1);

    uart_tx_data = 8'h00;
    uart_send_en = 1'b1;//-----发送00
    #(`CLK_PERIOD);
    uart_send_en = 1'b0;
    @(posedge uart_tx_done);

    #(`CLK_PERIOD*10+1);

    uart_tx_data = 8'hF0;
    uart_send_en = 1'b1;//-----发送F0
    #(`CLK_PERIOD);
    uart_send_en = 1'b0;
    @(posedge uart_tx_done);

    repeat(512)
      @(posedge ad9238_ddr3_uart.uart_tx_done);

    #2000;
    $stop;
  end
```

仿真文件的顶层，除了例化工程文件的顶层和 DDR3 模型以外， 描述了我们的启动指令的生成过程。我们上方代码模拟的正是串口接收口接收 55 A5 00 00 00 00 00 F0 的过程。

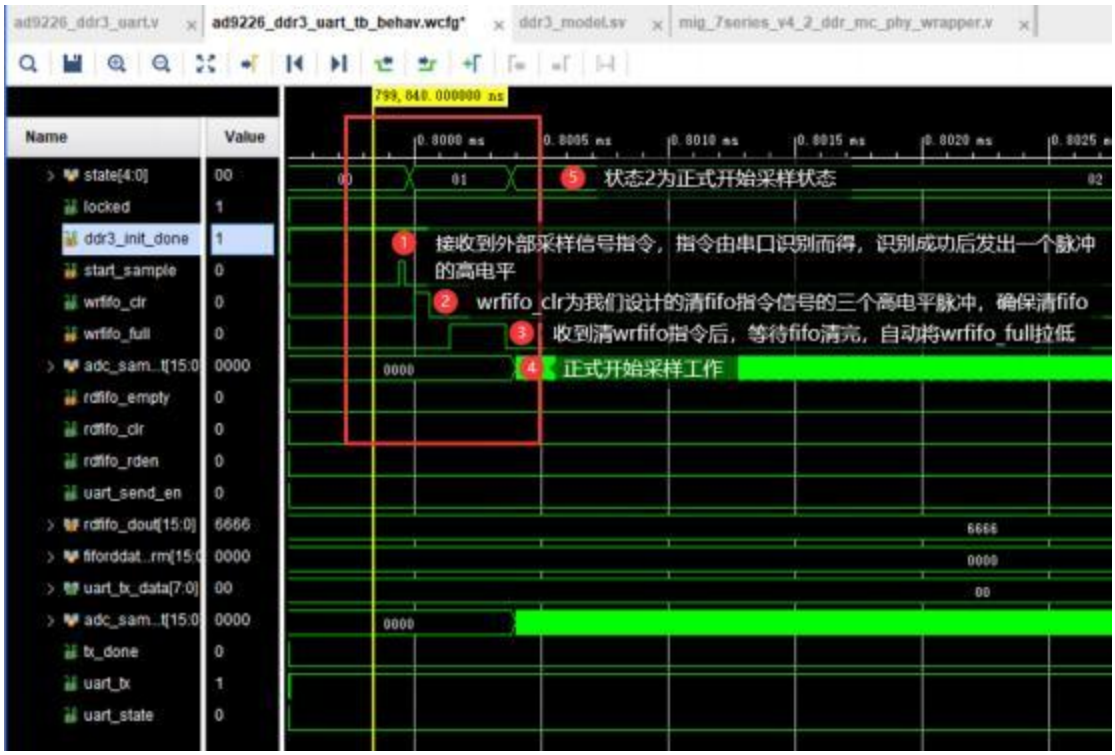
经过仿真，我们能验证整个程序的工作流程符合设计要求。下面，我们给出仿真的整体效果波形图和局部重要信息波形图供各位读者参考。



从整体效果波形图可以依次看出程序的全局信息， 仿真的结果符合我们的设计预期。

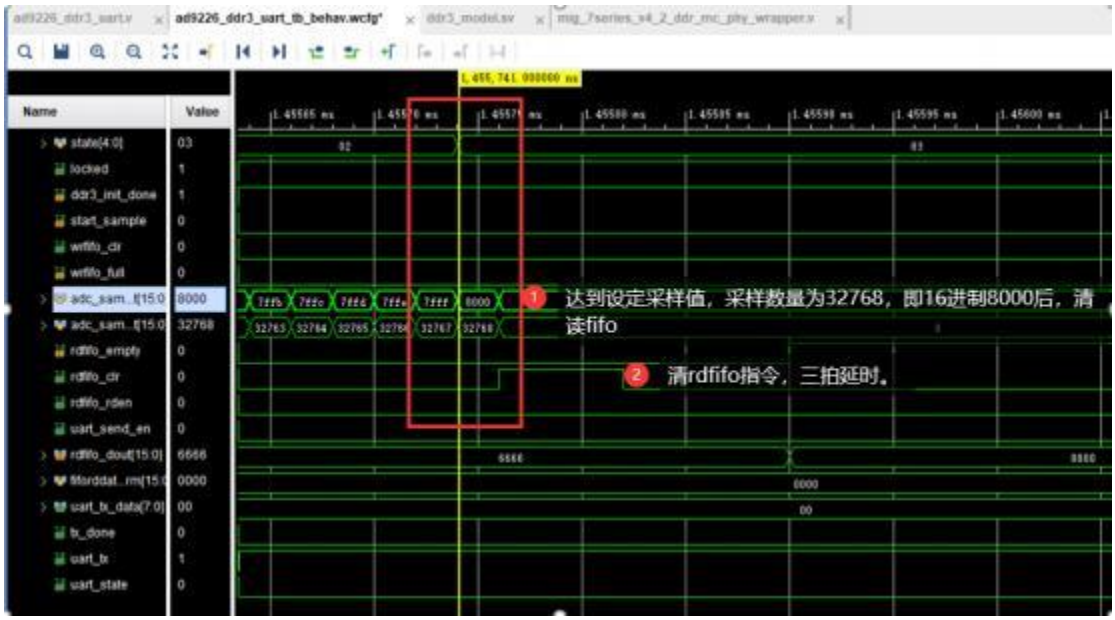


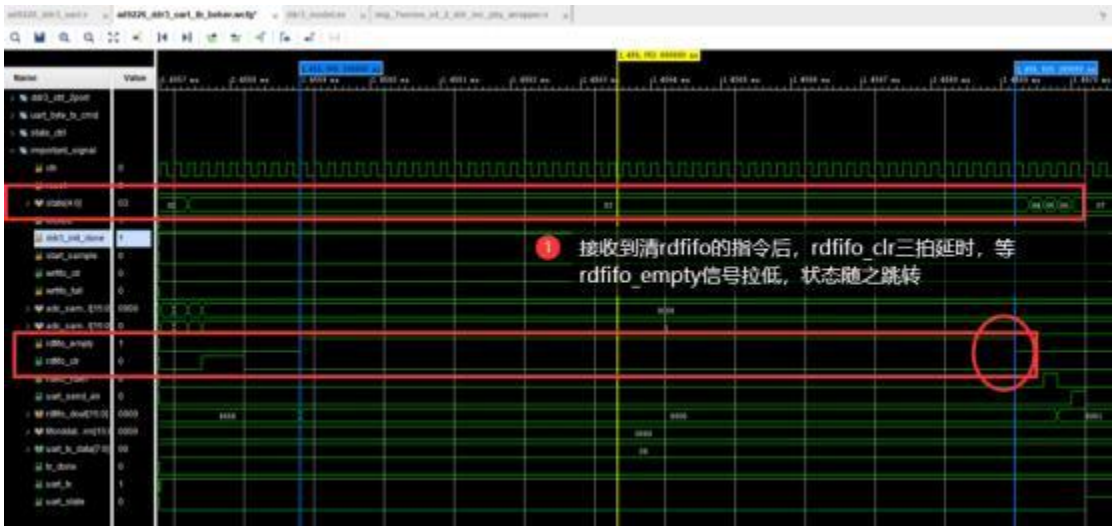
上图可以看出 DDR3 的模型给出 DDR3 上电初始化完成的时间参考值为 0.11ms。

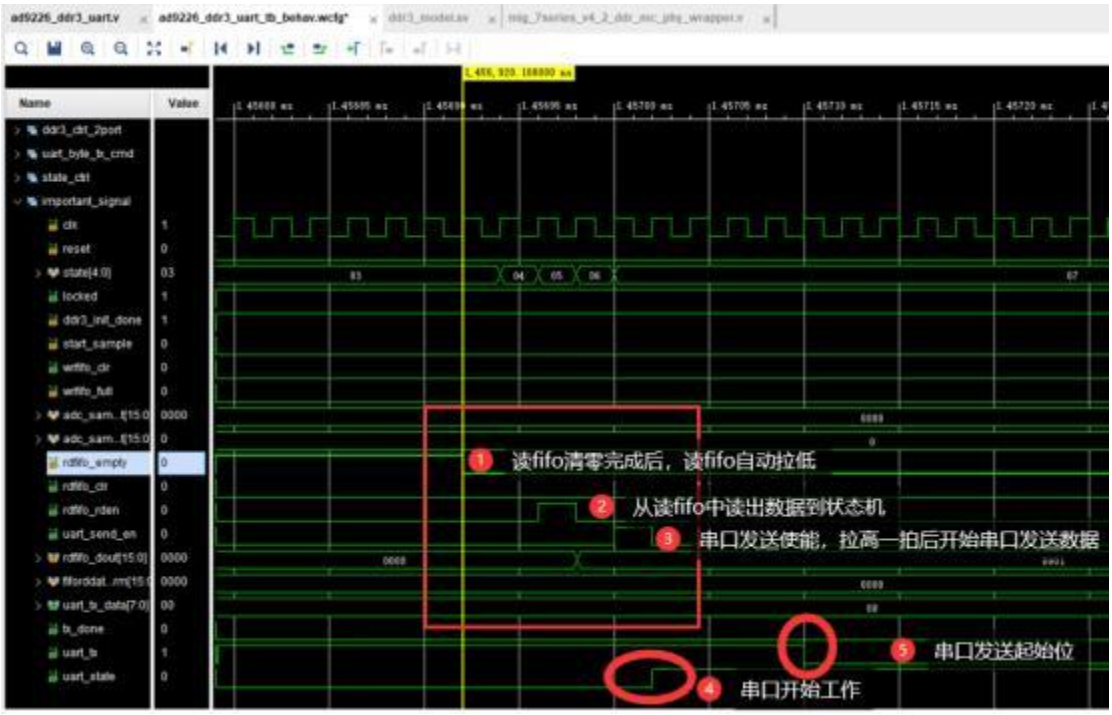


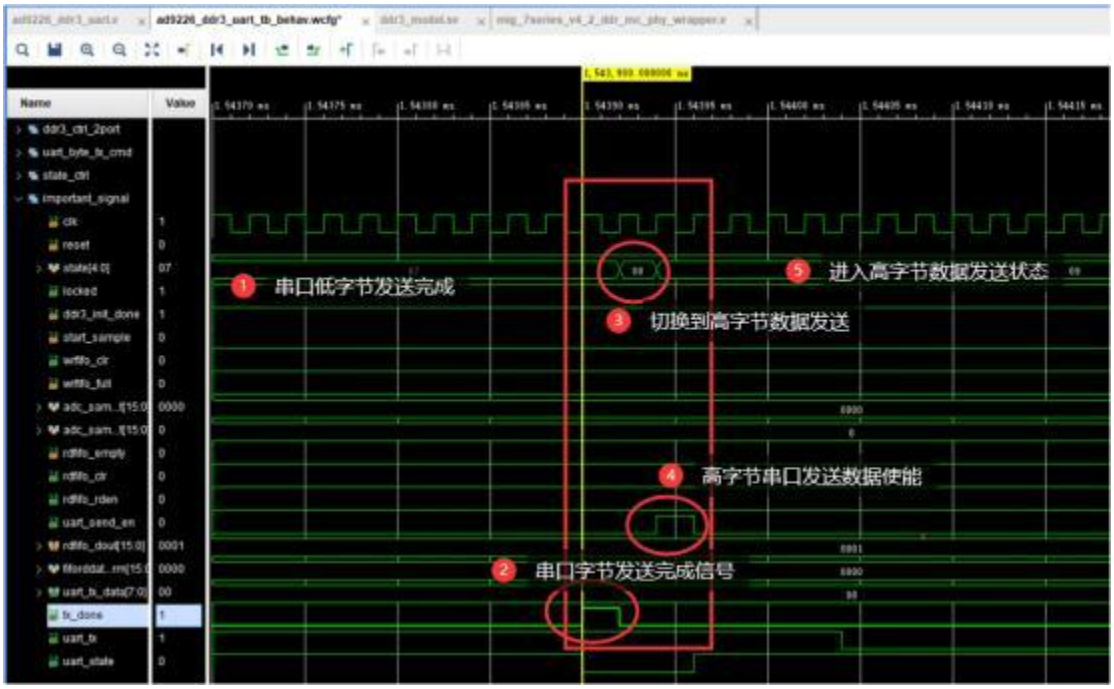
上面的波形展现了当开始采样信号到来时，进入清写 fifo 状态， 并收到写 fifo 拉低反馈， 进入采样状态的过程。

下方三幅图，展现的是达到设定采样数量(16’h8000 即十进制 32768)后，清读 fifo 的过程。发送 rdfifo_clr 后，收到 rdfifo_empty 拉高的反馈， 这时候， 等待 rdfifo_empty 拉低进入下个状态。

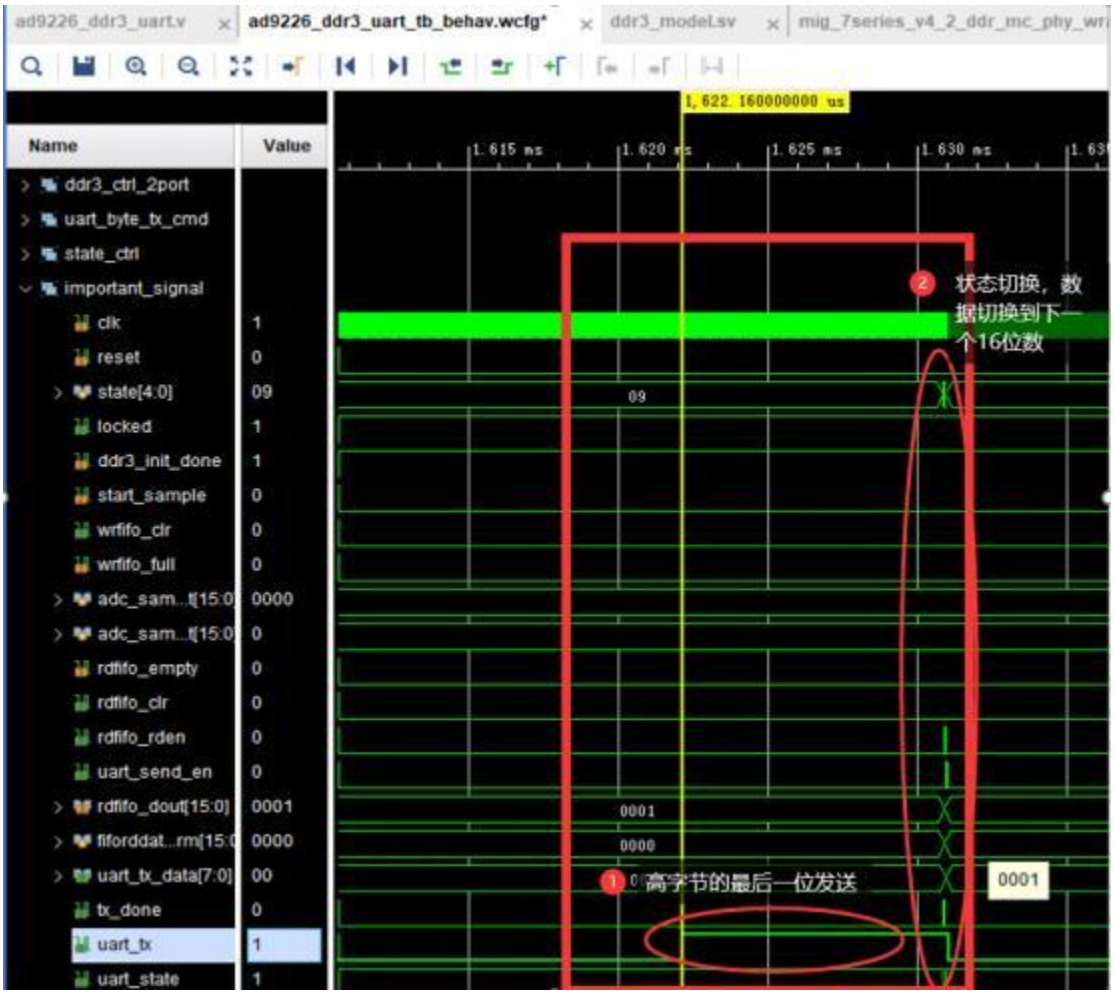


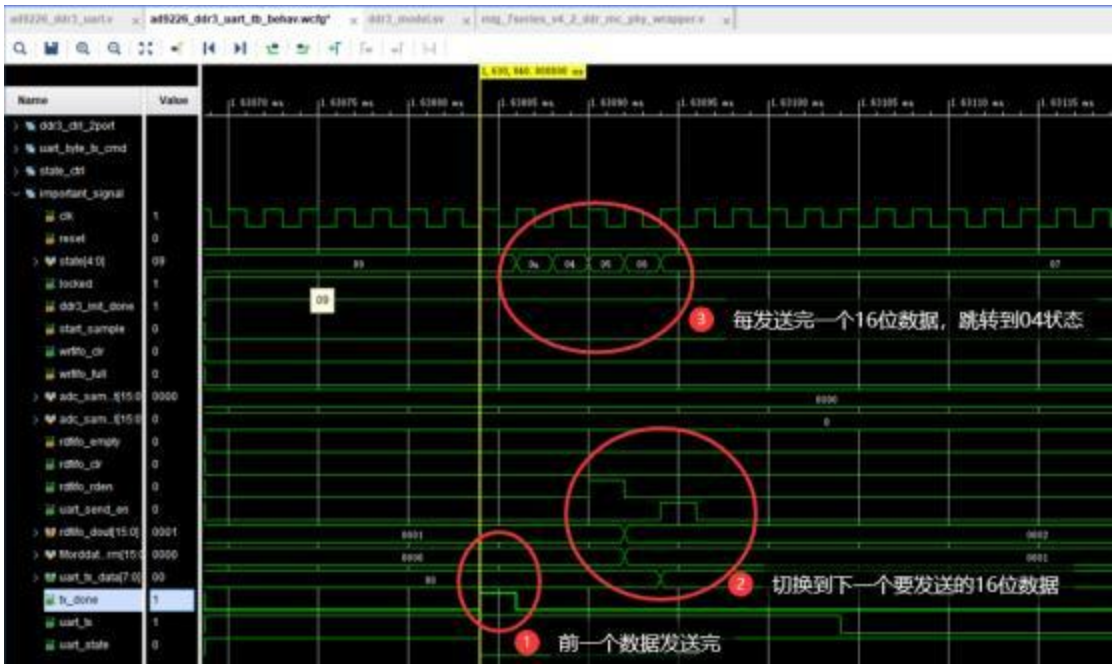

1 接收到清rdfifo的指令后，rdfifo_clr三拍延时，等rdfifo_empty信号拉低，状态随之跳转


ad9226_ddr3_uart.v
ad9226_ddr3_uart_tb_behav.wcfg*
Name
Value
ddr3_ctrl_2port
uart_byte_tx_cmd
state_ctrl
important_signal
clk
reset
state[4:0]
locked
ddr3_init_done
start_sample
wrfifo_clr
wrfifo_full
adc_sam...[15:0]
adc_sam...[15:0]
rdfifo_empty
rdfifo_clr
rdfifo_rden
uart_send_en
rdfifo_dout[15:0]
fiforoddat...rm[15:0]
uart_tx_data[7:0]
tx_done
uart_tx
uart_state
1 读fifo清零完成后，读fifo自动拉低
2 从读fifo中读出数据到状态机
3 串口发送使能，拉高一拍后开始串口发送数据
4 串口开始工作
5 串口发送起始位

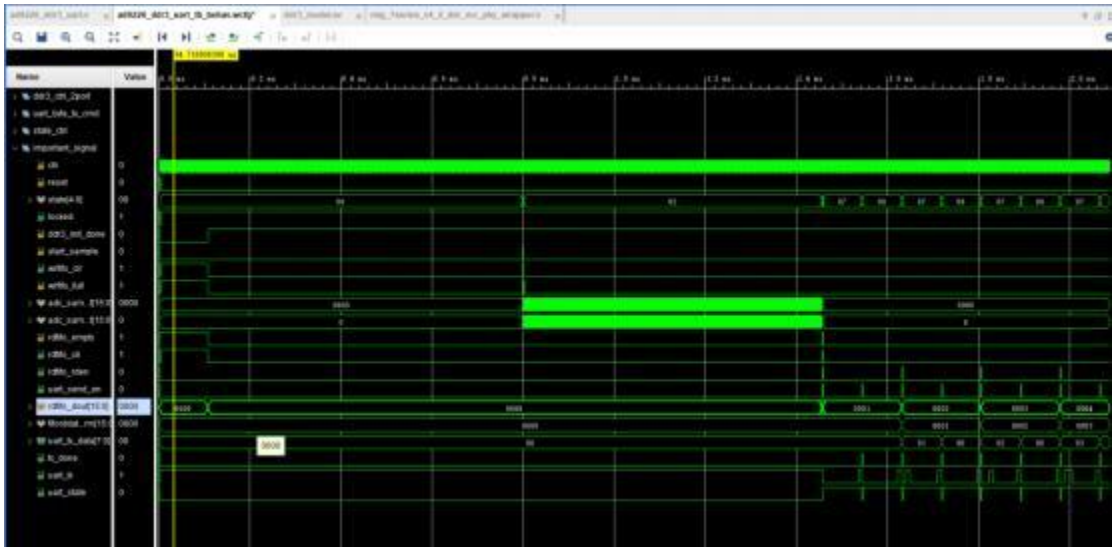


上图展现的是低 8 位数据发送完成后，在状态 8 读取高 8 位，并向串口发送 send_en 信号的细节过程。

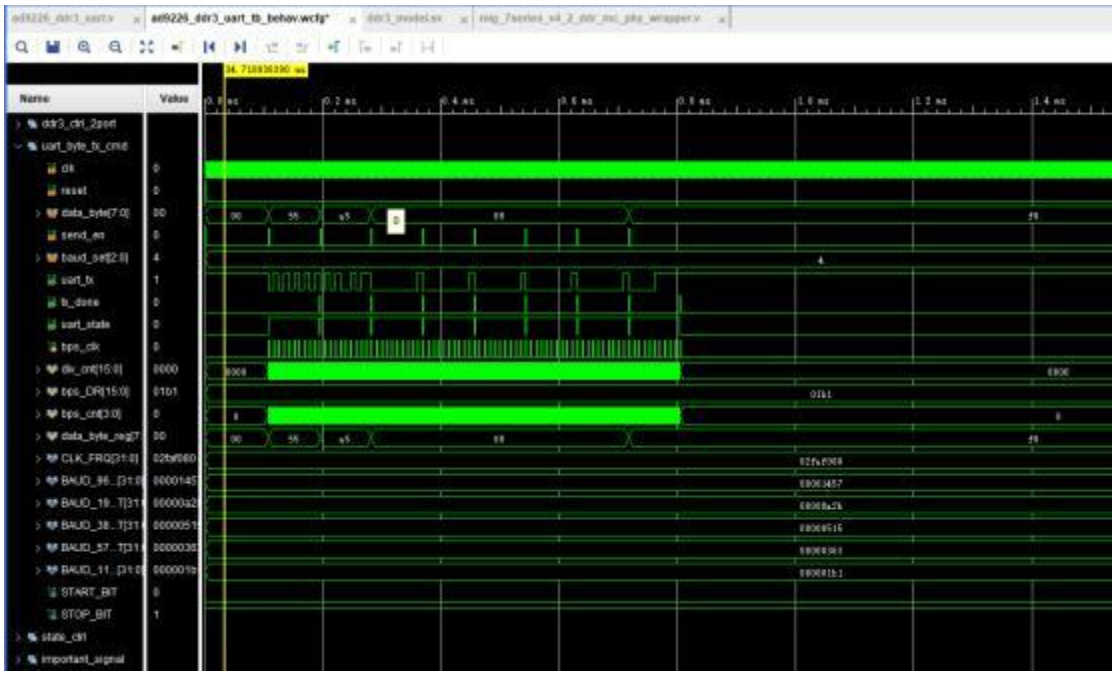




上面两幅图展现的是数据高 8 位发送完成后， 进入下一个小循环， 从读 fifo 中提取新一轮 16bit 数据，并提取新的低 8 位的过程。 第二幅图， 为第一幅图的状态切换细节展示。



上图为完整的从 ddr 初始化到前三个 16 位数据发送的过程关键信号全貌图。



上图反映的是接收到启动指令后，仿真的波形效果。

1.5. 板级调试

1.5.1 管脚绑定

按下表给出管脚对应关系后，编写好 xdc 文件。

ACM9238 在 ACX720 开发板输入输出端口列表及说明

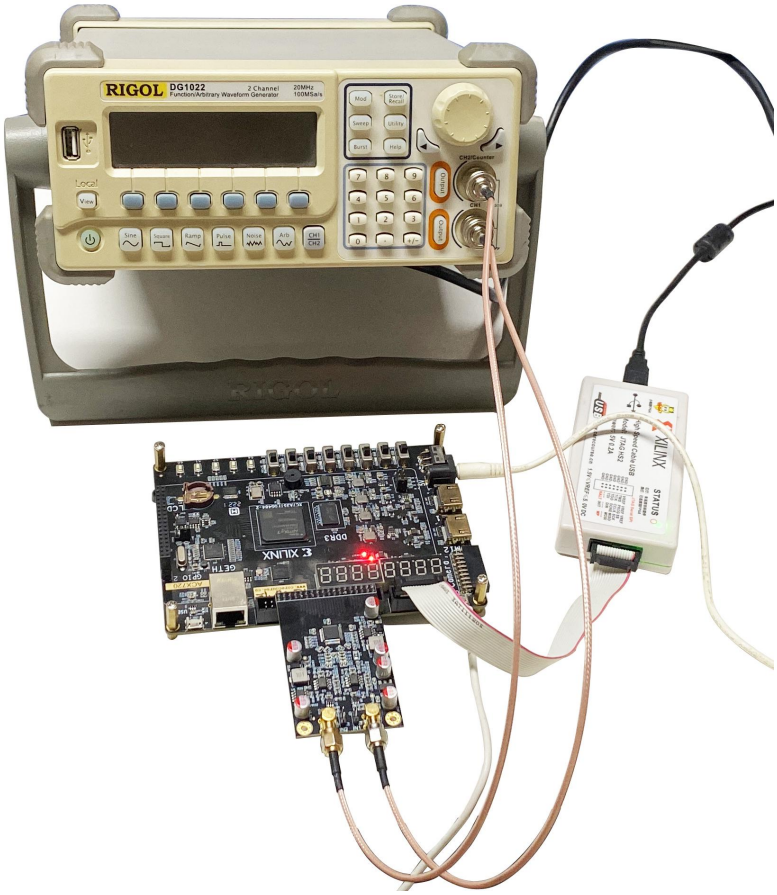
端口名	信号名	720 板管脚位置	方向	描述
ADCA_D0	GPIO0-0	PIN_A13	**input**	A 通道输出 D0 端， MSB
ADCA_D1	GPIO0-3	PIN_A16	**input**	A 通道输出 D1 端
ADCA_D2	GPIO0-2	PIN_A15	**input**	A 通道输出 D2 端
ADCA_D3	GPIO0-5	PIN_A19	**input**	A 通道输出 D3 端
ADCA_D4	GPIO0-4	PIN_A18	**input**	A 通道输出 D4 端
ADCA_D5	GPIO0-7	PIN_F14	**input**	A 通道输出 D5 端
ADCA_D6	GPIO0-6	PIN_F13	**input**	A 通道输出 D6 端
ADCA_D7	GPIO0-9	PIN_E14	**input**	A 通道输出 D7 端
ADCA_D8	GPIO0-8	PIN_E13	**input**	A 通道输出 D8 端
ADCA_D9	GPIO0-11	PIN_B13	**input**	A 通道输出 D9 端
ADCA_D10	GPIO0-10	PIN_C13	**input**	A 通道输出 D10 端
ADCA_D11	GPIO0-13	PIN_D15	**input**	A 通道输出 D11 端， LSB
CLKA	GPIO0-1	PIN_A14	**output**	ADCA 时钟，最高支持 65MHz
ADCB_D0	GPIO0-15	PIN_C15	**input**	B 通道输出 D0 端， MSB
ADCB_D1	GPIO0-14	PIN_C14	**input**	B 通道输出 D1 端
ADCB_D2	GPIO0-17	PIN_B16	**input**	B 通道输出 D2 端
ADCB_D3	GPIO0-16	PIN_B15	**input**	B 通道输出 D3 端
ADCB_D4	GPIO0-19	PIN_C17	**input**	B 通道输出 D4 端

ADCB_D5	GPIO0-18	PIN_D17	**input**	B 通道输出 D5 端
ADCB_D6	GPIO0-21	PIN_D16	**input**	B 通道输出 D6 端
ADCB_D7	GPIO0-20	PIN_E16	**input**	B 通道输出 D7 端
ADCB_D8	GPIO0-23	PIN_B18	**input**	B 通道输出 D8 端
ADCB_D9	GPIO0-22	PIN_B17	**input**	B 通道输出 D9 端
ADCB_D10	GPIO0-25	PIN_C19	**input**	B 通道输出 D10 端
ADCB_D11	GPIO0-24	PIN_C18	**input**	B 通道输出 D11 端， LSB
CLKB	GPIO0-27	PIN_E17	**output**	ADCB 时钟，最高支持 65MHz

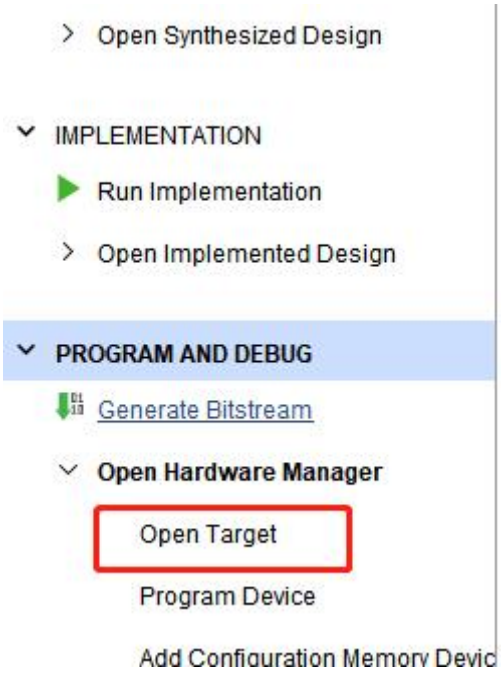
在此再次特别提醒：AD9238 的 12 位输出数据接口中，bit11 为 LSB、bit0 为 MSB，这与我们 FPGA 中对于多位宽信号的高低位的处理恰好相反， 所以使用时， 要么在分配引脚时直接调换过来，要么在程序中对所有位的高低位置进行调换。不然采集到的数据将看不到规律。

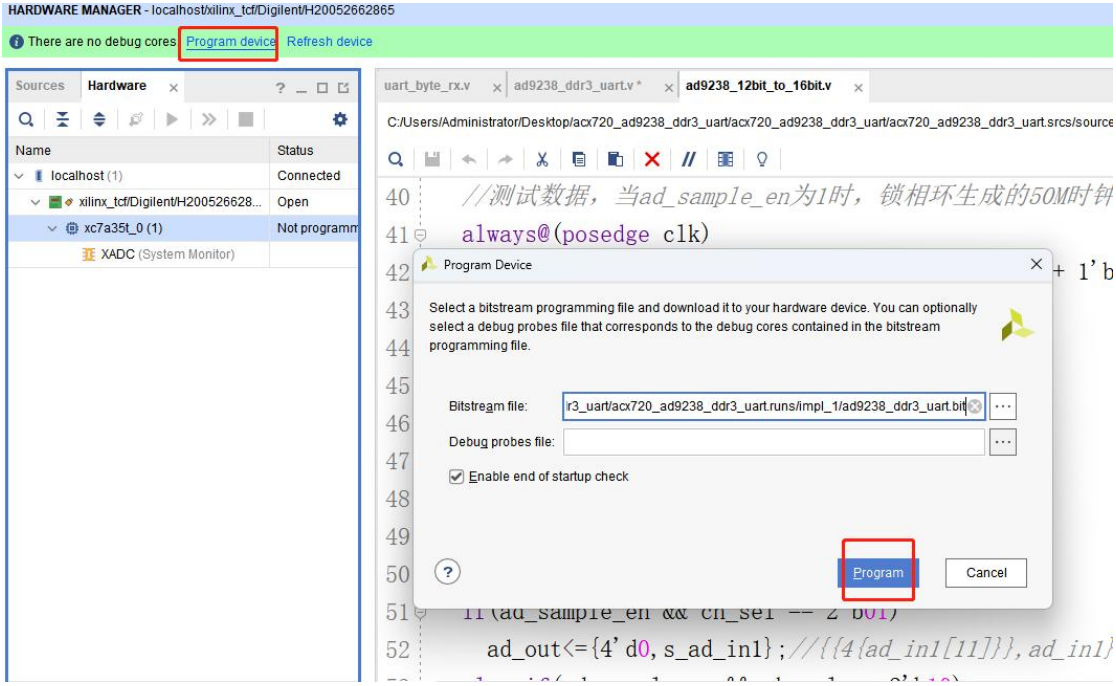
1.5.2 硬件连接

和前面的描述一致， 我们连接好信号发生器， ACM9238，ACX720FPGA 开发板电源、下载器、串口连接线。完成后如下图：

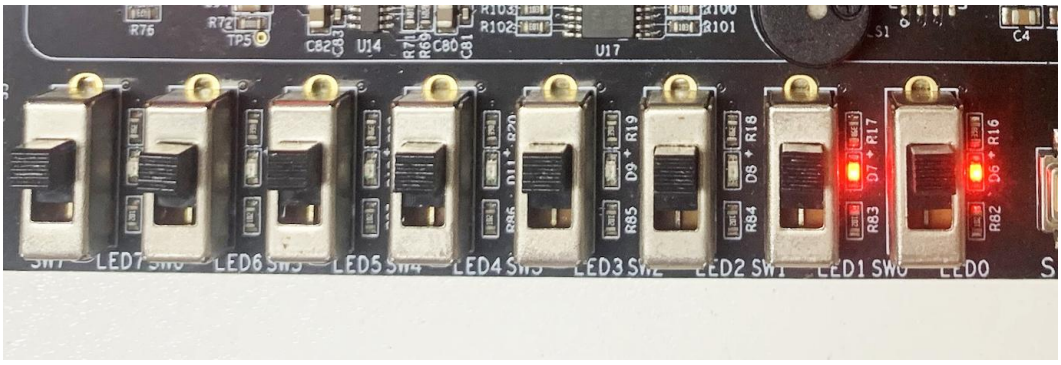


给 FPGA 开发板上电，然后确认文件路径正确后，烧写我们编写好的程序 bit 文件，

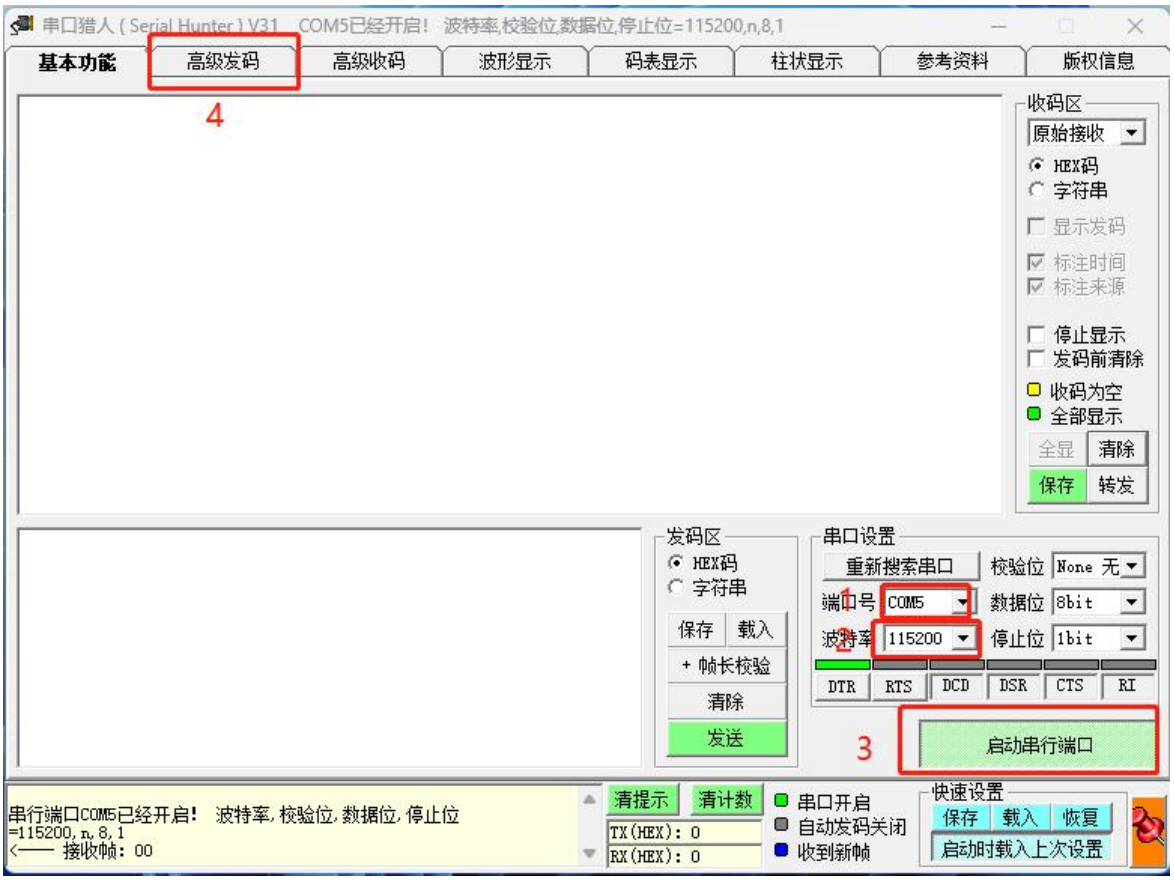




程序烧写完成后，开发板的 led0 和 led1 的灯会点亮。

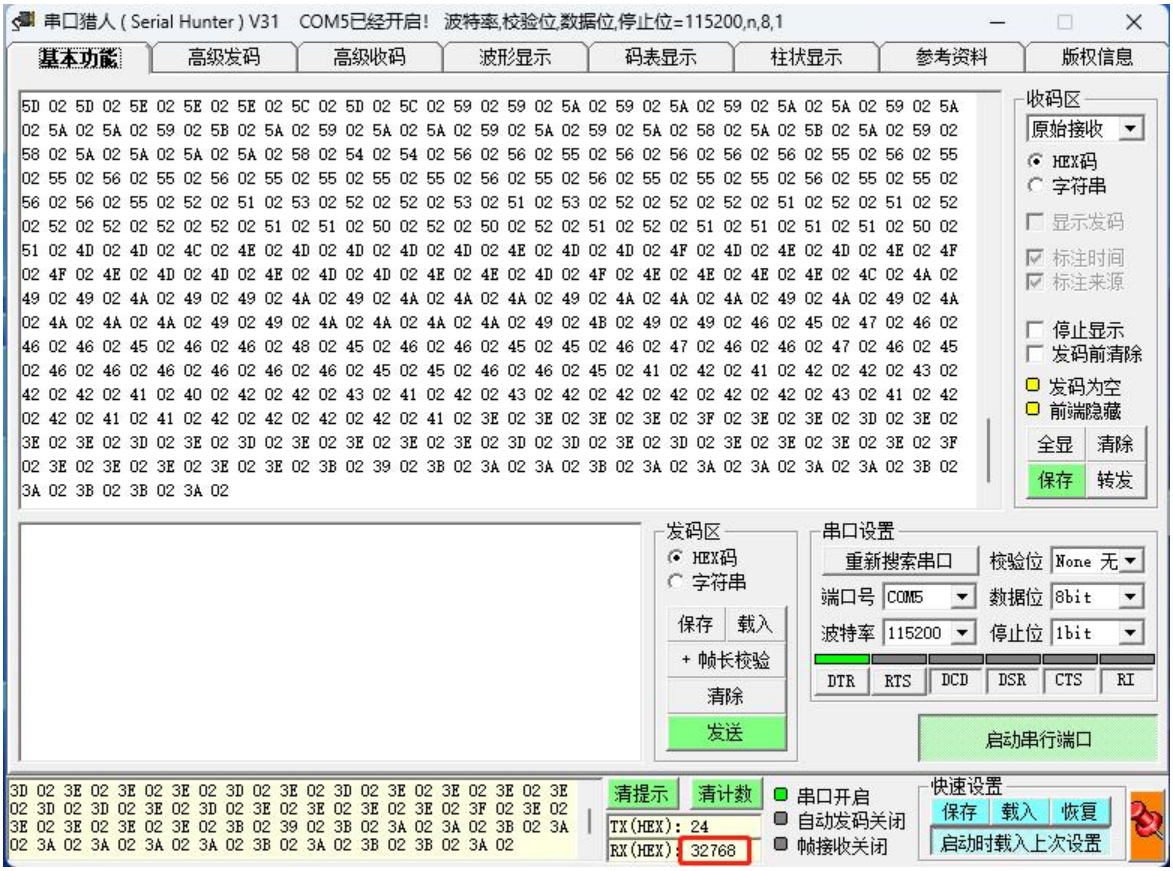


此时表明： 锁相环工作正常， ddr3 初始化正常。接下来打开串口猎人，界面如下图。



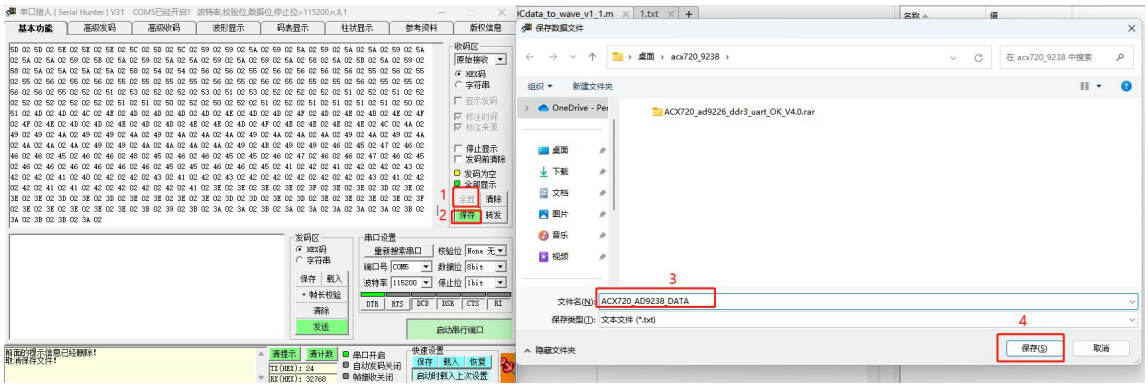
依次设定好 com 端口，波特率， 然后打开串口，然后选择高级发码选项卡清。

设置好参数，点击启动自动发码，开始采集数据。

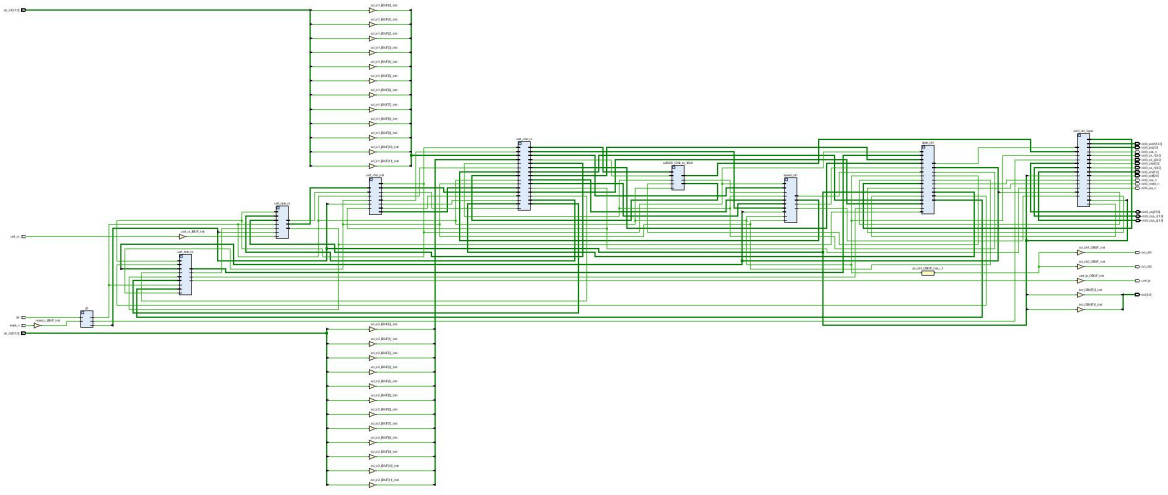


可以看到， 采集的数据和下发的指令是匹配的。

后面我们点击全显，接着点击保存按钮在弹出的对话框中选择保存的路径以及文件名，最后单击保存按钮。



再看我们工程生成的 RTL 级 Schematic 图，和我们的工程介绍，也是吻合的。



1.6. 数据的简易分析与评定

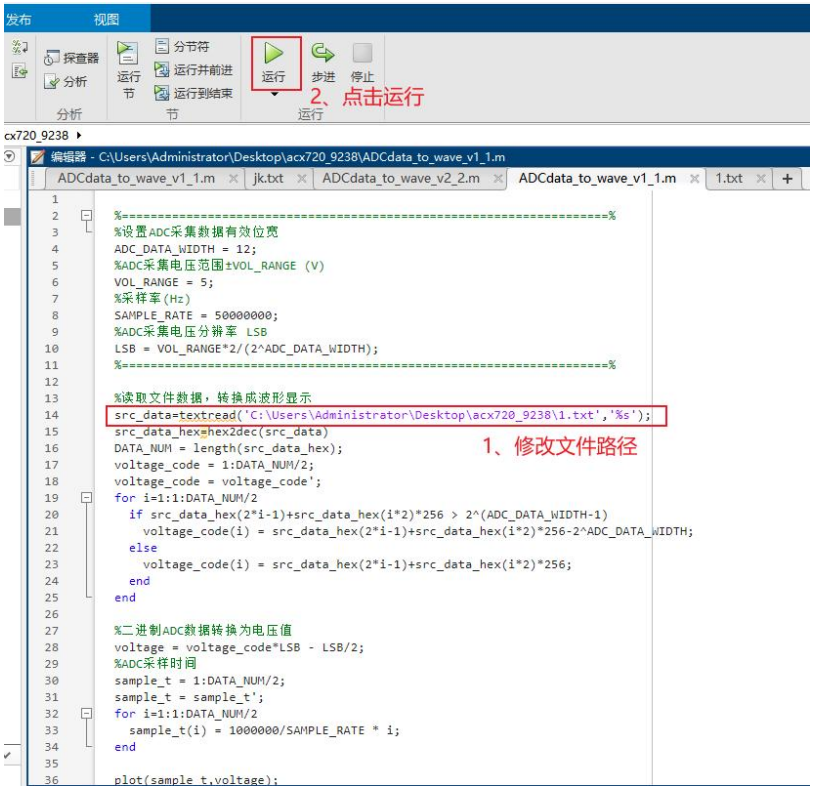
前面的实验能够进行，能够得出结论，但是采样得到的数据是否有丢失的情况发生，仅凭人工，无法完成这个分析工作。因此我们需要借助 matlab 的打印函数图形的功能。

下面给出 matlab 数据分析代码函数部分的简单介绍：

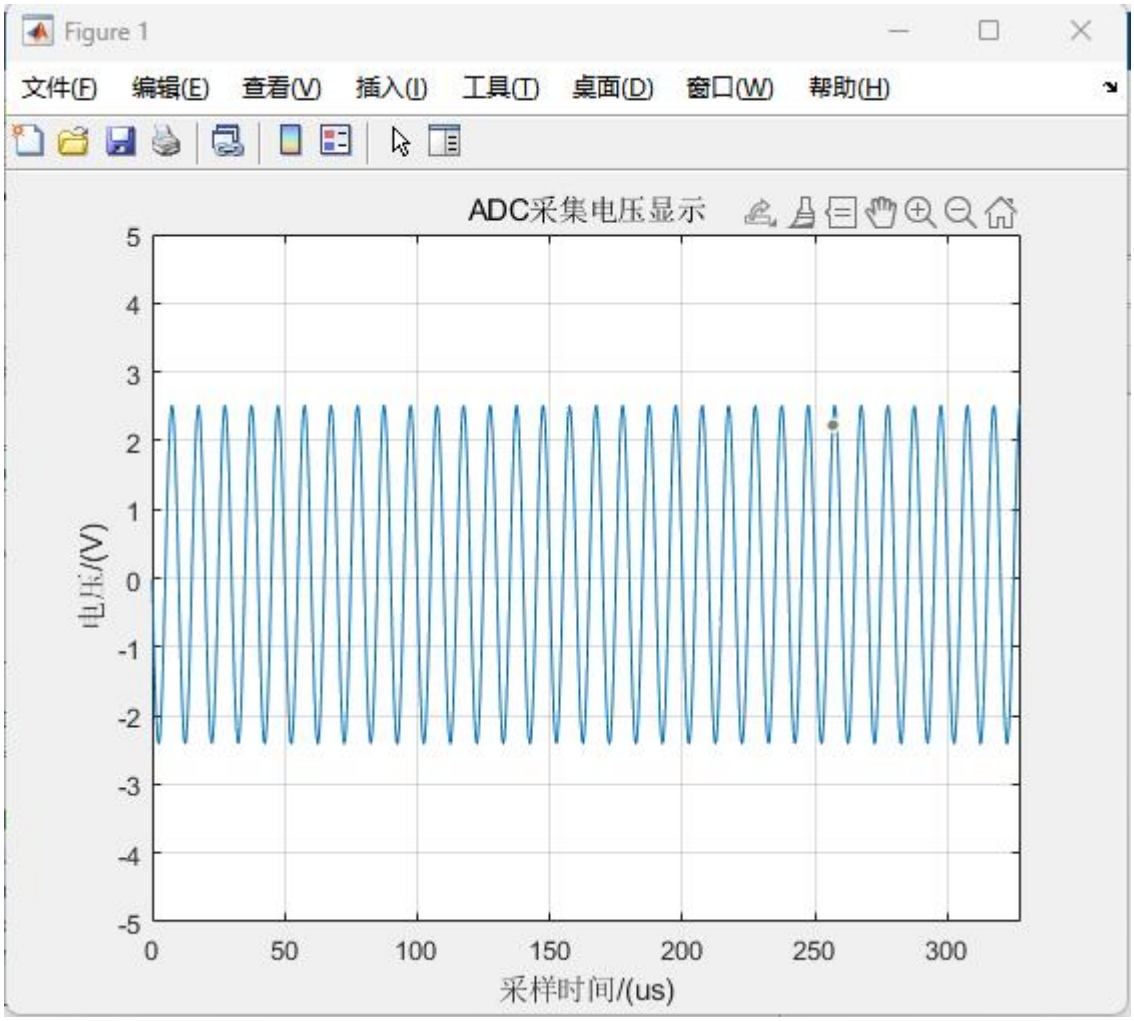
```
src_data=textread('C:\Users\Administrator\Desktop\003.txt
','%s');
src_data_hex=hex2dec(src_data)
DATA_NUM = length(src_data_hex);
voltage_code = 1:DATA_NUM/2;
voltage_code = voltage_code';
for i=1:1:DATA_NUM/2
   if src_data_hex(2*i-1)+src_data_hex(i*2)*256>2048
    wave_data(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256-4096;
  else
    wave_data(i) = src_data_hex(2*i-1)+src_data_hex(i*2)*256;
  end
end
```

代码的第一行， 指定的是数据文件的路径及格式， 使用时执行 matlab 程序前替换第一个单引号内为自己需要打印的数据的文件位置、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。

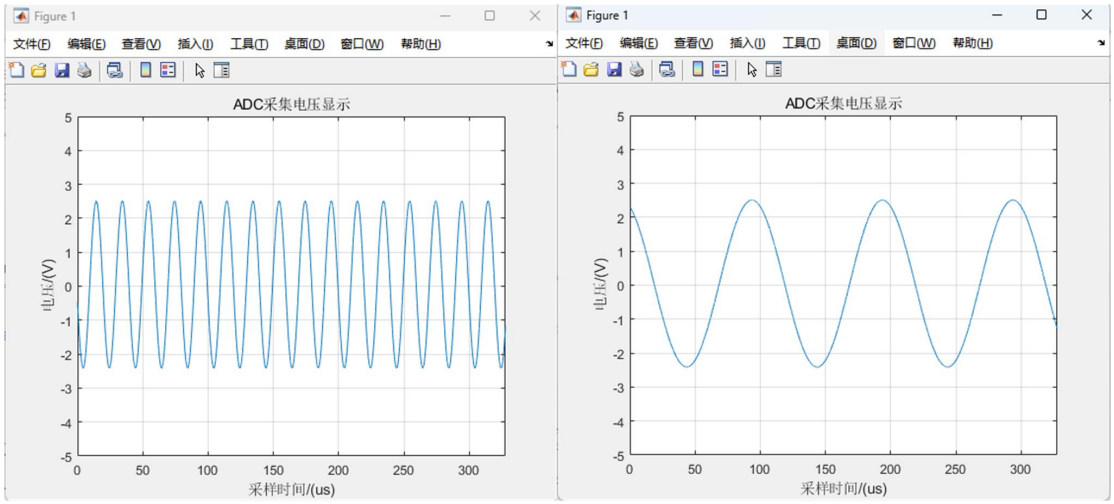
打开 matlab 代码的工程文件，修改好文件路径，也就是前面所保存的文件，进行保存后点击运行：



输出得到的波形如图：



这样工程的上板实验数据，就得到了验证。另外我们在修改频率为50kHZ和10kHZ，看看结果分别如何。



50KHZ　　10KHZ

1.7.总结

通过本实验，我们介绍了如下知识点：

1、ACM9238 模块的使用方式和硬件连接方法

2、串口发码程序向 FPGA 发布工作控制指令的设计和应用方法

3、二端口带双 fifo 的 ddr3 模块使用方法

4、含二端口带双 fifo 的 ddr3 仿真模块使用注意事项

5、本工程的数据运行流程和状态机的状态转换策略

6、matlab 数据绘图分析方法和程序代码

这些知识点，你都收获了吗？