

基于 FPGA 以太网 ADC128S102 数据采集系统设计

工程源码	----01_设计实例 ----- ACX720_ADC128S102_UDP_RGMII.zip
相关视频课程	暂无相关视频课程

概述

数据采集和控制系统是对生产过程或科学实验中各种物理量进行实时采集、测试和反馈控制的闭环系统，是测量和计算机控制的重要工具和手段，旨在收集有用的测量数据来进行表征、监测或控制。它广泛应用于电气测量、自动控制 and 大型设备故障诊断等领域。

在进行数据采集系统设计时，如何针对不同的应用领域和功能要求设计数据采集系统是设计中要点。用户的应用各不相同，每一种应用都有着特定的参数需求，而这些参数决定了对数据采集系统分辨率、精度、通道数量和速度的要求。这就对数据采集系统采样率、分辨率、数字信号处理速度、抗干扰能力要求更加严格。不同的应用场合对各项指标的要求也不尽一致，例如、爆炸检测、医疗设备(如 CT、核磁共振)、快速生产过程都需要高速或超高速数据采集系统来完成。而 FPGA 由于其 I/O 口众多，资源众多，可自由编程支配，接口众多方便连接等优点，在进行高速数据通信传输领域有着广泛应用。

1.1 数据采集系统设计概述

随着自动控制和计算机技术的成熟和发展，以及它们和传统工业的结合，加工设备的自动化程度有了大幅度的提高，各种控制设备与控制技术得到了广泛应用，大大提高了加工制造的效率，当然这种高效的生产对检测量的精确性和实时性也提出了更高的要求，而依靠传统的数据采集方法(即由操作工人凭借简易测量工具进行测量)已无法满足这种要求。

例如，在工业生产中，设备长时间运行容易出现故障，为了监控这些设备，通常利用数据采集装置采集他们运行时的数据并送给 PC 机，通过运行在 PC 机上的特定软件对这些数据进行分析，以此判断当前运行设备的状况，进而采取相应措施。

“数据采集”这一术语涉及许多测量应用，无论是何种特定应用，数据采

集系统的作用就是对测量的物理参数（温度、压力、流量等）进行收集，或是根据接收的数据采取特定动作（发出音频警报、开灯等）。被采集数据可以是模拟量，也可以是数字量。采集一般是抽样的采样方式，即隔一定时间（称采样周期）对同一点数据重复采集。采集的数据大多是瞬时值，也可是某段时间内的一个特征值。准确的数据测量是数据采集的基础。

而 FPGA 做为采集控制器广泛应用于数据采集这一领域，对于 FPGA 开发学习者，学会如何进行控制采集，对采集的数据进行实时的远程传输，开发出具有实时数据采集、存储、传输等功能的在线数据采集系统是十分有用的。

1.1.1 系统设计原理

数据采集系统主要内容分为数据采集、数据处理、数据存储和数据传输几个部分。就整个系统而言，为了实现数据的控制采集，系统一般需要包括采集器，A/D 转换模块，数据采集控制器，PC 端，通信线路等一系列硬件支持。

其中 AD 转换模块，对模拟信号进行转换输入，采集控制部分控制采集数据的参数量，PC 端数据分析，对采集的数据进行收集分析。

下图为一般数据采集系统设计系统框图：

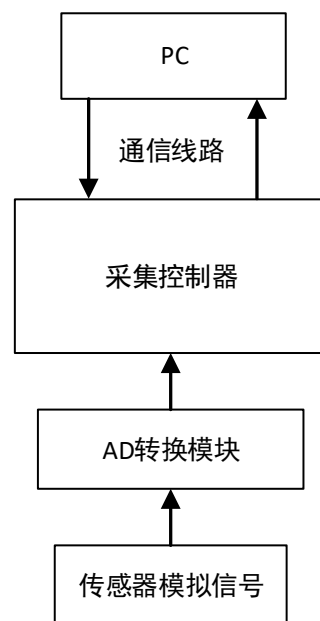


图 1 系统框图

考虑到复杂应用中所遇到的数据采样环节的种种情况，对数据采集系统的设计都会有如下一些基本要求。

- (1) 能够对多路信号进行实时数据采集，并且在一定范围内保证数据的同步采样；
- (2) 对信号实现高速采集，能够达到较高的采样速度；
- (3) 数据采集的精度符合应用要求；
- (4) 具有对采集的数据进行实时处理的能力；
- (5) 包含丰富的通讯接口，能将采集的数据方便、快速、正确的传输给 PC 机或其它嵌入式系统进行通讯；

1.1.2 FPGA 在数据采集领域的应用

自然界中的信号大部分是模拟信号，因此一般的信号处理系统中都要包括数据的采集功能。通常的实现方法是利用 A/D 转换器将模拟信号转换为数字信号后，送给处理器。

对于低速的 A/D 和 D/A 转换器，可以采用标准的 SPI 接口来与 MCU 或者 DSP 通信。但是，高速的 A/D 和 D/A 转换芯片，比如视频 Decoder 或者 Encoder，不能与通用的 MCU 或者 DSP 直接接口。在这种场合下，FPGA 可以完成数据采集的粘合逻辑功能。

在采集到数据后，很多时候需要应用存储器对采集的数据进行存储，例如 SDRAM、SRAM、Flash 等。由于 FPGA 的功能可以完全自己设计，因此可以实现各种存储接口的控制器。

在实际的产品设计中，很多情况下还需要与 PC 机进行数据通信。比如，将采集到的数据送给 PC 机处理，或者将处理后的结果传给 PC 机进行显示等。与 PC 机的数据传输在传统的设计中往往需要专用的接口芯片，比如 PCI 接口芯片。如果需要的接口比较多，就需要较多的外围芯片，体积、功耗都比较大。而采用 FPGA 的方案后，接口逻辑都可以在 FPGA 内部来实现了，大大简化了电路设计。

1.1.3 数据传输分析

在图 1 数据采集系统设计系统框图中我们可以看到，PC 端与采集控制器之间需要通信线路进行通信，这是我们进行数据采集系统设计关注的重点。对于采集器，我们可以将其看做接通了电源的电器，而通信的设计就是线路搭建提供电源，这样搭载的电器（采集器）就可以正常使用。

在数据采集系统的设计中，根据通信方式设计好系统后，根据搭载的数据采集处理器进行系统的调整可以实现相应的功能。采用 FPGA 进行数据采集系统设计，FPGA 和 PC 端通信方式有多种，根据设计需求可以进行合理选择。这里对常用的几种通信方式进行介绍。

1. 标准串口（RS232）数据传输特点

RS-232 串口作为电子设备与计算机以及电子设备之间最早的通信手段现在还广泛的应用于工业控制领域。232 通讯线路简单，只要一根交叉线即可与 PC 主机进行点对点双向通讯。线缆成本低，但传输速度慢、不适于长距离通讯。消费类 PC 机也逐渐取消了该接口，目前多存在于工控机及部分通信设备中。

2. RS-485 总线

而 RS-485 总线作为弥补 RS-232 串口通信技术通信距离短，只能支持点对点数据通信等缺点产生的通信技术 同样广泛的应用于工业控制领域。随着信息社会对于工业自动化的要求不断提升，以及工业自动化程度的不断加深，由于 RS-232 串口支持点对多点通信，通信距离小于 15 米，而 RS-485 总线技术则通信距离为 1200 米，由于采用一主多从的通信模式，从设备接收发送数据都需要等待控制主机的指令，而 RS-485 总线技术用于长距离通信的波特率不能高于 110Kbps,对于大数据量并且 实时性要求高的通信任务则难以满足需求。原有的基于 RS-232/485 串口通信的工业控制系统越来越不能满足工业控制领域现实的需求。

3. 以太网进行数据传输

通过以太网技术作为通信手段相比 RS-232/485 串口通信技术具有以下几个优势：

（1） 以太网设备的配置更加灵活方便。

多个 RS-232 串口设备与计算机通信一般都是在计算机 PCI 插槽上通过多串口卡或者通过 USB 串口连接 USB 转串口集线器来实现 RS-232 串口的扩展，每个计算机的 PCI 插槽或者 USB 接口都是有一定的数量限制，从而使得与计算机通信的 RS-232 串口设备的数量受到相应的限制。而 RS-485 总线虽然布线简单，负载设备多，通信距离可以达到 1200 米，但是其布线必须采用手牵手菊花链拓扑结构，在 RS-485 总线上增加设备需要将线路布设过去或者通过增加 485 中继器或者 485 集线器来解决布线问题，而以太网则不同，只要有网络信息口的

地方，就可以直接将相关的以太网设备连线接入以太网，而且接入设备的数量基本上是没有限制的。

(2) 支持热插拔工作。

能够在系统工作的时候配置相关设备，无需停止系统工作。以太网上增加相应的以太网设备，只需要在附近的信息口上接上网线就可以，再通过计算机上的相关软件进行配置就可以正常工作。

(3) 简单易用，后期的维护方便简捷。

不管是 RS-232 设备还是 RS-485 总线通信一般都是只与单台计算机进行通信，很难形成双服务器冗余热备份系统，而在以太网上可以非常容易的配置双服务器冗余热备份系统。同样的道理，基于 RS-232 串口通信或基于 RS-485 总线通信，不能形成冗余链路，一旦出现问题就可能整个系统崩溃，特别是 RS-485 总线，在 RS-485 总线上出现问题（比如短路），很容易导致整个系统不能使用，而且在 RS-485 总线上查找故障点非常困难，需要一个一个的去排查。而基于工业以太网作为通信手段则可以避免类似问题，采用工业以太网交换机布设环形冗余链路的工业网络，一旦某个链路出现问题，可以在 20ms 之内自愈恢复并及时告警提示维护。

(4) 高扩展性和高扩充性，非常适应弹性布线。

前文所述，以太网设备使用热插拔工作以及配置灵活方便，采用以太网作为通信手段可以没有距离上和数量上的限制，RS-485 总线通信距离为 1200 米，可以通过增加 485 中继器或者通过光纤 modem 转换为光信号通过光纤传输从而达到延长通信距离的作用，但是传输距离总是有一定的限制，而通过以太网可以连接至互联网，通过互联网可以在世界任何一个有网络连接的地方进行数据交换，同样的，RS-485 总线长距离通信的最大速率为 110Kbps，而现在快速以太网（100M）已经基本普及，千兆以太网则正在逐步进入工业控制领域，所以以太网的通信容量以及在以太网上通信的设备数量基本上没有任何限制。

(5) 通过以太网通信实现真正的“管控一体化”。

随着工业控制自动化程度以及办公系统自动化程度的加深，现在提出了工业控制领域“管控一体化”的目标，也就是说工业控制系统与办公自动化系统能够紧密结合，信息互通有无实现无缝对接。由于现在办公系统都是基于以太网进行数据交换，所有的软件都是基于以太网运行，与以太网设备通信无需作

任何修改，可以直接与之通信，能够快速的将工业控制网络中的以太网设备的相关数据整合进办公自动化系统。

这里简单对比了三种通信方式的特点，以太网以其自身优势在数据采集系统设计中多有应用，以太网通信的数据采集系统如何进行设计，我们运用一个案例进行演示，引导读者了解其设计的思路方法。

1.2 ADC128S102 以太网数据采集系统设计

1.2.1 系统设计框图

下面我们就以 ADC128S102 数据采集以太网传输为例进行数据采集系统设计。案例基于 Xilinx 系列 FPGA，结合 ACX720 开发板上板载的 12 位 8 通道串行采样 ADC 芯片，通过开发板上千兆以太网 RGMII 接口，实现对 ADC128S102 型 8 通道 12 位 ADC 的数据转换控制并输出。

相比于 GMII，RGMII 接口可以在保证传输速率不变的情况下减少管脚数。RGMII 接口接收网口下发的数据并转化成控制命令对 ADC128S102 的数据采样个数以及采样通道进行合理配置，采集完成后的数据通过网口传输到电脑。用户可以在电脑上通过网口调试工具进行指令的下发以及对采样到的数据进行查看，可将输出的数据导出，进行进一步的数据处理分析。

数据采集系统设计框图如图所示。

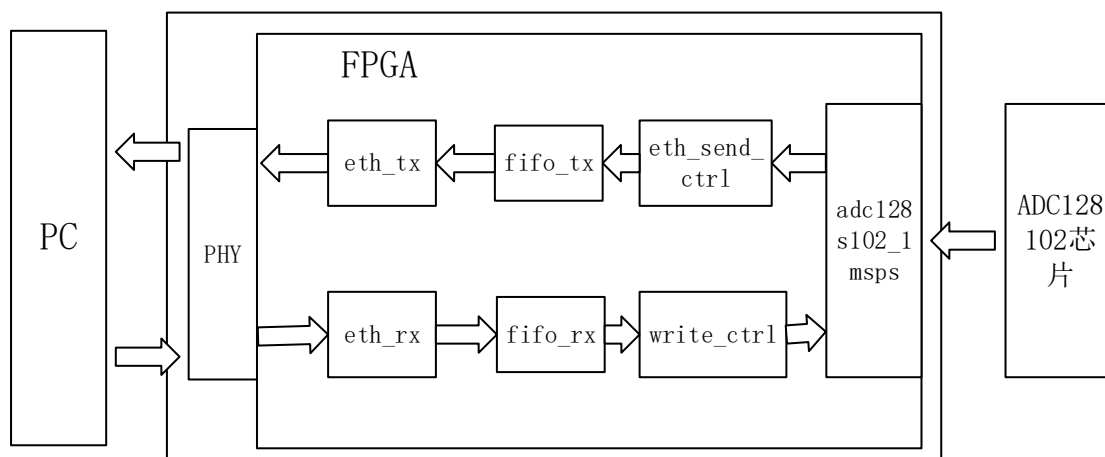


图 2 数据采集系统设计框图

在原理图中清晰的展示了，ADC128S102 数据采集以太网数据传输系统的数据关系。PC 端作为指令的发送端，同时也是数据的接收处理端，发送的指令在通过 PHY 网口芯片后传递到 FPGA，而采集到的数据经由 FPGA 控制处理后，

通过网口传递到 PC 端进行分析。

就本设计而言，其中设计的关注点在于通信方式与采集转换器之间的数据流。从设计框图看到，设计 ADC128S102 以太网数据传输系统需要完成两个部分的内容：一个是指令的下发，要完成指令流的过程；另一个就是数据的上传，完成数据流的控制过程。ADC128S102 作为数据处理转换模块，运用提供的驱动就可以运行，进行系统 FPGA 设计的重点和难点还是数据流和指令流的控制设计，根据不同的传输需求进行合理的设计变换，掌握其设计技巧才能一通百通。

1.2.2 指令传输设计

在对指令的配置了解之后，我们开始进行系统指令下发的设计。指令的传达需要经历下面的过程：

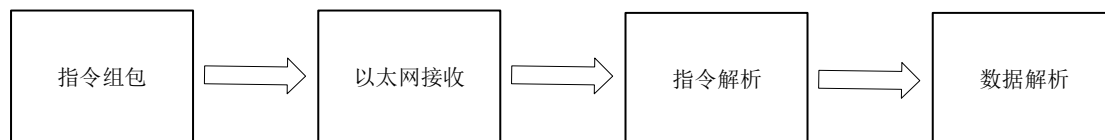


图 3 指令的传达过程

在本次设计中，指令通过以太网进行下发，指令数据需要在 PC 端利用上位机进行组包。组包后的指令数据通过网口下发到 FPGA，FPGA 接收到 PC 端的指令帧数据后将以太网数据在以太网模块进行解析，得到原始的指令数据。在采集控制器部分说到，指令数据本身也是包含帧头和帧尾的数据帧，要实现数据采集的配置需要对寄存器帧数据进行解析，得到配置参数，实现相应的采集控制。

数据采集控制也就是对该模块输入数据的采集，在设计过程中涉及到的数据采集量、采样通道的设置都是围绕该模块进行。在进行设计之前，需要先对配置 ADC128S102 驱动的寄存器进行说明，配置 ADC128S102 的寄存器一共有四个，他们的功能和地址分别如下：

表 5 配置 ADC128S102 的寄存器功能和地址

名称	地址	位宽	功能简介
RestartReq	0	1	重新开始采集请求寄存器，向该寄存器写入任意值即可启动新一轮的采样存储传输。
ChannelSel	1	8	通道设置寄存器，共 8 位，对应了 8 个通道的数据存储开关，如果某通道对应的设置位为 1，则该通道的采样结果就会被存入 FIFO 并通

			过网口发送。
DataNum	2	16	数据个数寄存器，设定总共采集传输多少个数据。
ADC_Speed_Set	3	32	本次实验未使用到，默认使用最高采样速率为 1M（64M/（DIV_PARAM * 2）/16），每个通道也就是 125K（1M/8）的采样速率，因为这个采样速率已经很低了，在程序里面就没有设置对应的命令去控制其采样速率，所以向 03 号寄存器中写入任何值都是没有意义的。

对于设计的控制一个寄存器的数据帧，该帧一帧数据共 8 个字节，包含帧头、帧尾、地址段、数据段。帧格式如下所示：

表 6 命令数据帧格式

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

每帧数据共 8 个字节，分别用 D0~D7 表示，其中，D0 和 D1 两个数据作为帧头，其值分别为固定的 0x55、0xA5，D7 作为帧尾，其值为固定的 0xF0。D2 则为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。

以上面接收转命令模块中介绍到的数据帧格式对 ADC128S102 驱动四个寄存器进行配置。

例如，PC 端要设置采样数据个数(DataNum 寄存器，地址为 2)为 16000（0x3E80）个，则发送的数据帧内容为：0x55 0xA5 0x02 0x00 0x00 0x3E 0x80 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 号寄存器负责启动 ADC，ADC 在未配置完全的情况下开始启动，数据很容易输出错误值。就能实现以 1M 的采样速率，对 1 个通道进行采样，共采集 16384 个数据。四个寄存器对应的代码如下：

DataNum: 55 A5 02 00 00 40 00 F0

ChannelSel: 55 A5 01 00 00 00 01 F0

ADC_Speed_Set: 55 A5 03 00 00 00 F9 F0

RestartReq: 55 A5 00 00 00 00 00 F0

为了配置四个寄存器，网口发送一次命令数据内容为 32 个字节，指令的解析也是以 32 位指令进行解析，如果只需配置一位寄存器，其它寄存器数据置零即可。根据寄存器配置好指令数据后，在 PC 端利用上位机下发，通过网口传输的这些寄存器的值，FPGA 在接收到指令后需要对发送一次的数据进行拆解才能实现，通过以太网传输指令对 ADC12S102 驱动进行数据控制采集。

1.2.2.1 PLL 的配置设计

对于数据和时钟传输，大多数支持 RGMII 的 PHY 芯片都提供了一个对发送时钟 RX_CLK\TX_CLK 添加延迟的选项。可以根据设计要求启用或禁用此选项。如当启用延迟 PHY 设备内 TX_CLK 的选项时，FPGA 必须生成与数据和波形边缘对齐的时钟，如下图所示。

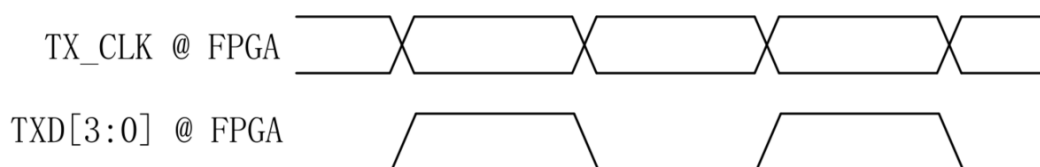


图 4 数据和时钟边缘对齐图

这种情况下，PHY 芯片会移动时钟的相位来捕获数据。如果 PHY 的该功能被关闭，则 FPGA 必须对时钟信号进行一定的相位调整后再作为 TX_CLK 输出，以使 TXD 和 TX_CLK 成中心对齐关系。当然，如果 PCB 上的 TX_CLK 信号线路本身就引入了一定的延迟，则需要将这一部分延迟考虑在内，再计算 FPGA 需要对 TX_CLK 调整的相位值，其传输波形如下图所示。



图 5 数据传输波形

将源时钟与数据对齐的方法有多种。例如可以使用 PLL 产生一路相位相差 90 度的时钟信号，作为 TX_CLK 输出。当然，如果考虑到 PCB 板上的 TX_CLK 相对于 TXD 的延迟值，该相位可以进一步调整以确保 TX_CLK 到达

PHY 时与 TXD 呈中心对齐关系。

由于以上原因，我们在引入 eth_rxc 时同样需要使用 PLL 对其相位进行调整，PLL 设置如所示。

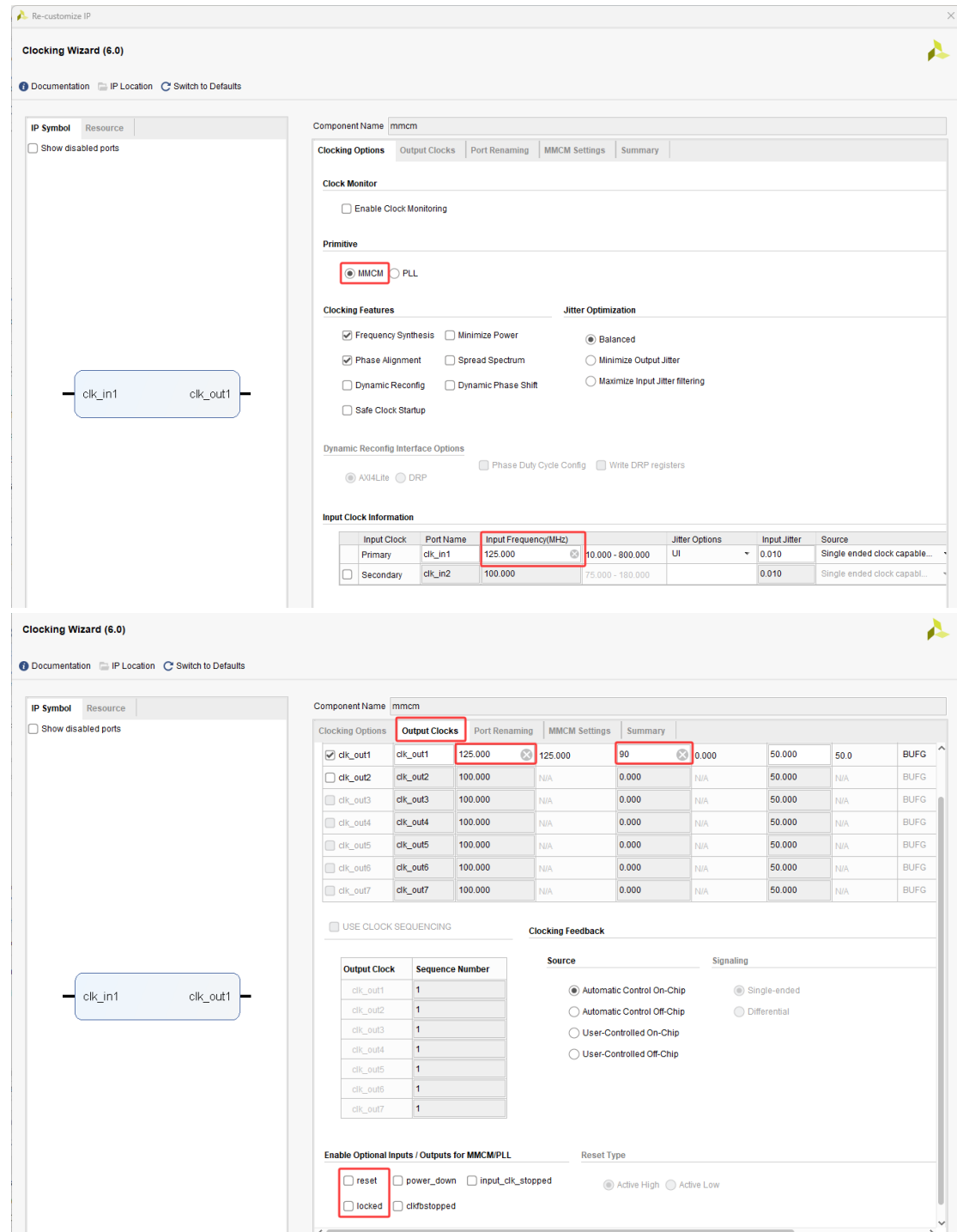


图 6 PLL 配置图

店铺: <https://xiaomeige.taobao.com>

官方网站: www.corecourse.cn

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

技术群组:

由于 PHY 芯片时钟信号高速路径与数据信号普通 IO 走线的差异，在进行相位调整时读者可根据具体情况进行微调，以期达到最好的数据采集效果。下面是 rx_pll 模块图。

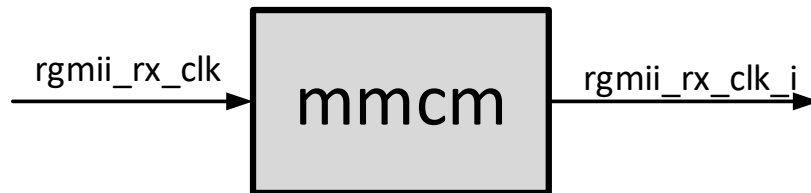


图 7 时钟 IP 核 PLL

表 1 接口信息表

接口名称	I/O	功能描述
rgmii_rx_clk	I	由 PHY 输出工作时钟，频率为 125MHz
rgmii_rx_clk_i	O	相位调整后输出时钟信号，作为以太网相关模块工作时钟

设计中 FPGA 中的 rx_pll 模块对输入的 eth_rxc 时钟信号进行相位调整后，输出调整后的时钟信号提供给 eth_udp_rx_gmii 等模块，可作为 TX_CLK 输出，以使 TXD 和 TX_CLK 成中心对齐关系。在实际调试测试过程中，如果不调整相位，相位偏差会导致在程序的调试过程中指令发送后数据不是每次都可以被接收到。

1.2.2.2 RGMII 接收接口实现

对于 FPGA 来说，实现 RGMII 接口的接收同样是一个非常直接的过程，整个接收逻辑框图如图 8 所示：

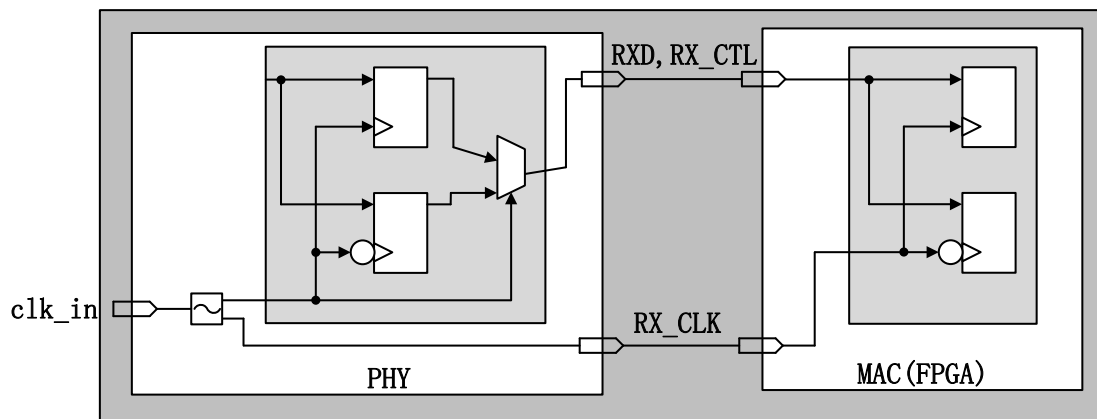


图 8 RGMII 接收设计逻辑框图

同样的，设计实现时，可通过使用 xilinx 的 IDDR 原语，将该接口使用 ILOGIC 块实现。在 ILOGIC 块中，有着专用的寄存器，用于实现输入双倍数据速率（DDR）寄存器，当我们实例化 IDDR 原语时便会自动访问该功能。IDDR 原语的框图如图 9 所示：

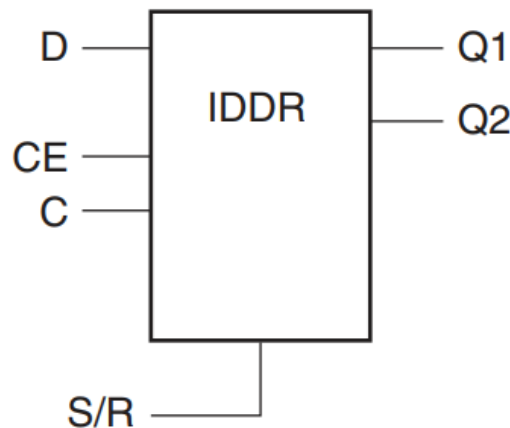


图 9 IDDR 基本框图

其中各个端口的功能及描述如表 2 IDDR 端口信号：

表 2 IDDR 端口信号

Port Name	Function	Description
Q1 和 Q2	数据输出	IDDR 寄存器输出。
C	时钟输入端口	时钟输入引脚
CE	时钟使能端口	时钟使能引脚，高电平时新数据才会被加载到 DDR 触发器中
D	数据输入（DDR）	IOB 的 IDDR 寄存器输入。
S/R	设置/复位	同步/异步的设置/复位引脚。该引脚高电平有效

这里需要注意的是 set 和 reset 同时只能有一个被置高，也因此，描述端口

时，使用的 S/R。除了这些端口外，IDDR 原语还包含一些可用属性，具体如表 3 IDDR 属性所示：

表 3 IDDR 属性

Attribute Name	Description	Possible Values
DDR_CLK_EDGE GE	根据时钟边沿设置 IDDR 操作模式	OPPOSITE_EDGE(默认), SAME_EDGE, SAME_EDGE_PIPELINED
INIT_Q1	设置 Q1 端口的初始值	0 (默认), 1
INIT_Q2	设置 Q2 端口的初始值	0 (默认), 1
SRTYPE	相对于时钟的设置/重置类型 (C)	ASYNCR, SYNC (默认)

这里简单描述下 IDDR 的三种操作模式：

1. OPPOSITE_EDGE

该模式通过 ILOGIC 块中的单个输入实现，Q1 端口在每个时钟周期的上升沿输出数据给 FPGA 逻辑，Q2 端口在每个时钟周期的下降沿输出数据给 FPGA 逻辑。图 10 显示了该模式的输入 DDR 时序：

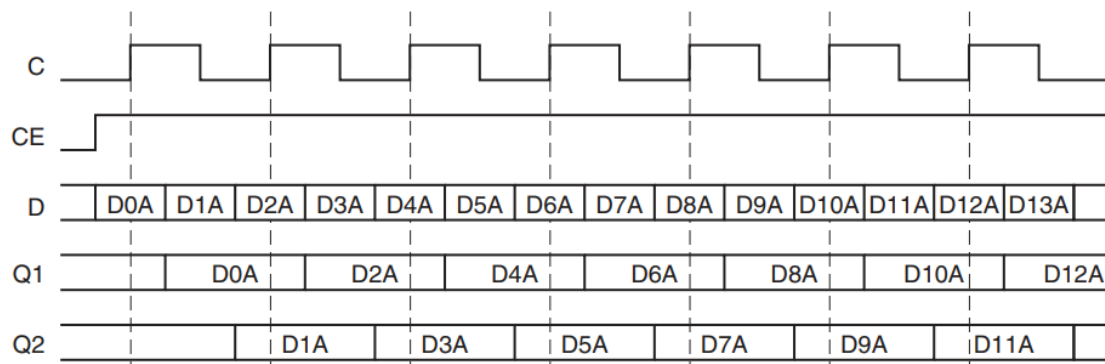


图 10 输入 DDR 时序（OPPOSITE_EDGE 模式）

2. SAME_EDGE Mode

在 SAME_EDGE 模式下，数据在同一时钟边沿被输出给 FPGA 逻辑。图 11 显示了使用 SAME_EDGE 模式的输入 DDR 的时序图。

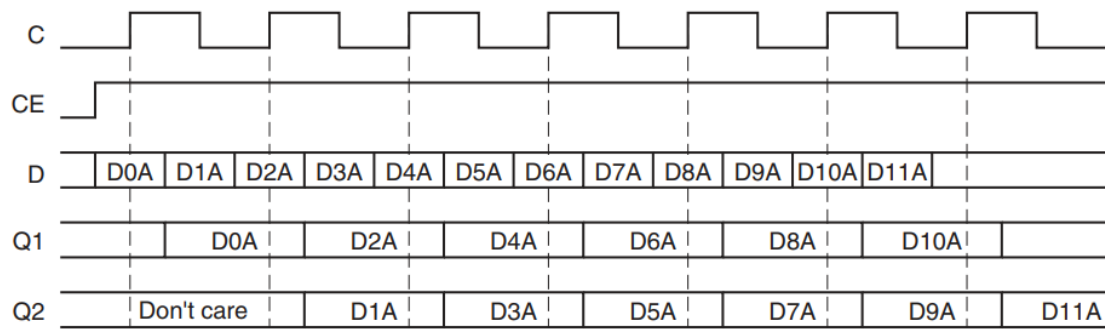


图 11 输入 DDR 时序 (SAME_EDGE 模式)

可以看到，该模式下，输出对 Q1 和 Q2 不再是从 D0A 和 D1A 开始传输。相反，呈现的第一对是对 Q1 (D0A) 和 Q2 (不关心)，然后才是下一个时钟周期的数据对 D1A 和 D2A。

3. SAME_EDGE_PIPELINED

在 SAME_EDGE_PIPELINED 模式下，数据在同一时钟边沿传输给 FPGA 逻辑。与 SAME_EDGE 模式不同，数据对需要额外的时钟延迟来消除 SAME_EDGE 模式的分离效应。该模式的输入 DDR 时序如图 12 所示：

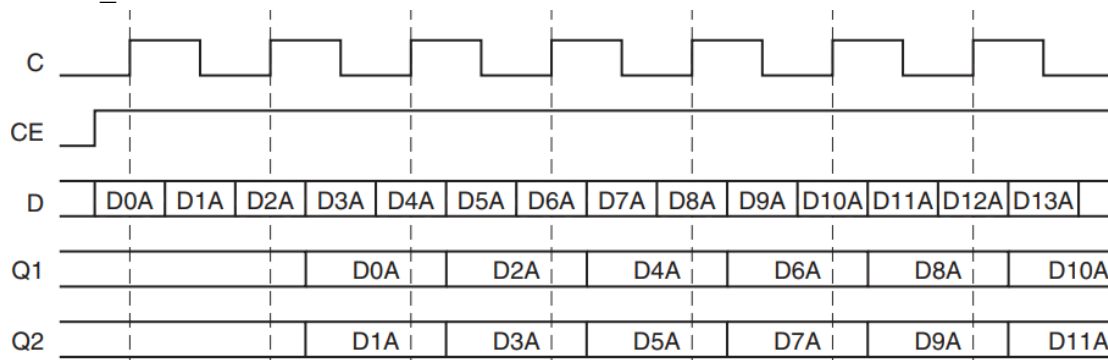


图 12 输入 DDR 时序 (SAME_EDGE_PIPELINED 模式)

这三种操作模式可以在使用 IDDR 原语时通过 DDR_CLK_EDGE 的属性值设置，在 Xilinx 的官方手册 UG768 中，给出了 IDDR 原语的例化示例，如下：

```
// IDDR: Input Double Data Rate Input Register with Set, Reset
// and Clock Enable.
// 7 Series
// Xilinx HDL Libraries Guide, version 14.7
IDDR #(
    .DDR_CLK_EDGE("OPPOSITE_EDGE"), // "OPPOSITE_EDGE", "SAME_EDGE"
    // or "SAME_EDGE_PIPELINED"
    .INIT_Q1(1'b0), // Initial value of Q1: 1'b0 or 1'b1
```

```
.INIT_Q2(1'b0), // Initial value of Q2: 1'b0 or 1'b1
.SRTYPE("SYNC") // Set/Reset type: "SYNC" or "ASYNC"
) IDDR_inst (
.Q1(Q1), // 1-bit output for positive edge of clock
.Q2(Q2), // 1-bit output for negative edge of clock
.C(C), // 1-bit clock input
.CE(CE), // 1-bit clock enable input
.D(D), // 1-bit DDR data input
.R(R), // 1-bit reset
.S(S) // 1-bit set
);
// End of IDDR_inst instantiation
```

为了让接收的数据能够更加容易的被 MAC 捕获，大多数支持 RGMII 的 PHY 芯片都提供了一个对接收时钟 RX_CLK 添加延迟的选项。可以根据设计要求启用或禁用此选项。当启用延迟 PHY 设备内 RX_CLK 的选项时，PHY 输出的 RX_CLK 与 RXD 会呈中心对齐的关系，如图 13 所示。在这种情况下，FPGA 就可以直接使用 RX_CLK 来捕获输入的数据，而不需要再对 RX_CLK 添加 PCB 板级延迟，或者是在 FPGA 内部对 RX_CLK 添加延迟。

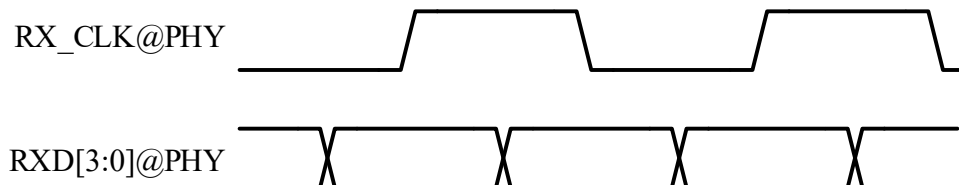


图 13 时钟与数据对齐关系（PHY，启用延迟）

如果 PHY 对 RX_CLK 添加延迟的功能被关闭（波形图如图 14 所示），则 FPGA 必须对 RX_CLK 时钟信号进行一定的相位调整后再用来捕获数据。这种情况，可以使用 FPGA 内的 PLL 将该信号进行一定的相位移动后再用来捕获数据。当然，让 RX_CLK 能够进入 PLL，必须要求该信号是从 FPGA 器件的专用时钟输入管脚或 DQS 脚输入。

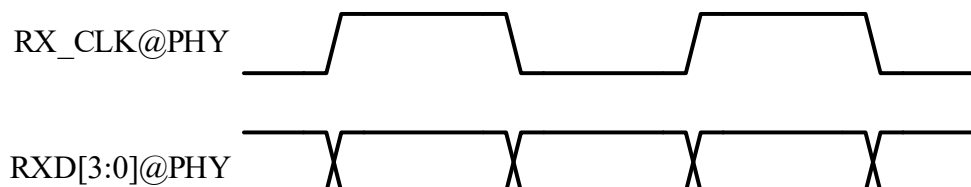


图 14 时钟与数据对齐关系（PHY，禁用延迟）

可以看到，通过 FPGA 侧实现 RX_CLK 和 RXD 的中心对齐需要使用到 PLL 资源，因此，使用 PHY 内部的 RX_CLK 延迟功能将降低对 FPGA 的资源

占用。推荐有条件的情况下尽量选择 PHY 内部增加延迟的方案。

当然，对于本次设计而言，我们可以直接使用 IDDR 原语实现。

1.2.2.3 GMII 以太网接收模块的应用

GMII 以太网相关的模块读者可以使用我们已经验证好的驱动模块。以太网接收模块接收到 RGMII 转 GMII 模块的数据信息，并对接收的数据帧进行判断解析，验证以太网帧数据的准确性，对准确的以太网数据帧进行解析后，得到我们需要的指令数据。

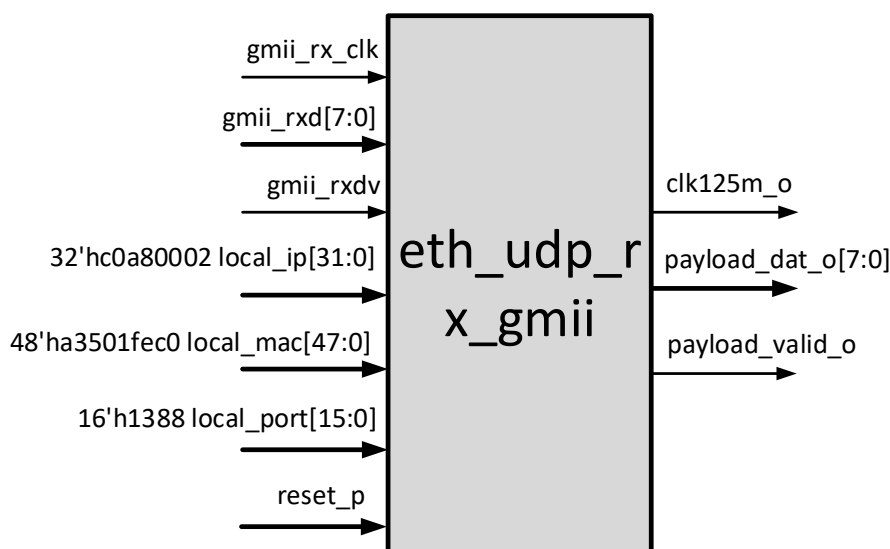


图 15 gmii 以太网接收模块

接口名称	I/O	功能描述
gmii_rx_clk	I	gmii 工作时钟，125MHz
gmii_rxd[7:0]	I	八位数据输入
gmii_rxdv	I	数据输入控制信号
local_ip	I	本地 IP 地址
local_mac	I	本地 mac 地址
local_port	I	本地端口号
clk125m_o	O	输出 125M 时钟
payload_dat_o[7:0]	O	以太网解析后八位数据输出
payload_valid_o	O	输出控制使能信号，有效时数据输出

GMII 以太网接收模块的端口中一部分为以太网设置固定参数端口，读者可根据实际情况进行合理配置。本地主机（PC）端为目的端，与之连接的 FPGA 端为源端，其中 local_mac、local_ip、local_port 分别为源 MAC 地址、源 IP 地

址与源设备端口号，把设置的 MAC 地址转换为十进制表示为：00_10_53_01_254_192，源 IP 地址十进制为：192_168_00_02，源设备端口号十进制表示为：5000。

GMII 以太网接收模块的工作时钟 gmii_rx_clk 为 125MHz，数据接收 gmii_rxd 信号为 8 位，每次接收一个字节。gmii_rxdv 为接收数据有效信号，gmii_rxdv 有效 FPGA 接收数据。

以太网接收模块对数据进行分析处理后可输出我们所需要的信号，根据本次实验需要，这里我们输出三个信号。Clk125m_0 作为时钟信号输出，payload_dat_o 则是从 PC 端接收的数据信息，后续需要对 payload_dat_o 输出的数据做进一步的处理得到所需要的控制指令数据。payload_valid_o 为 payload_dat_o 输出标志信号。

对 GMII 以太网接收模块进行仿真验证观察，模块仿真数据如所示。

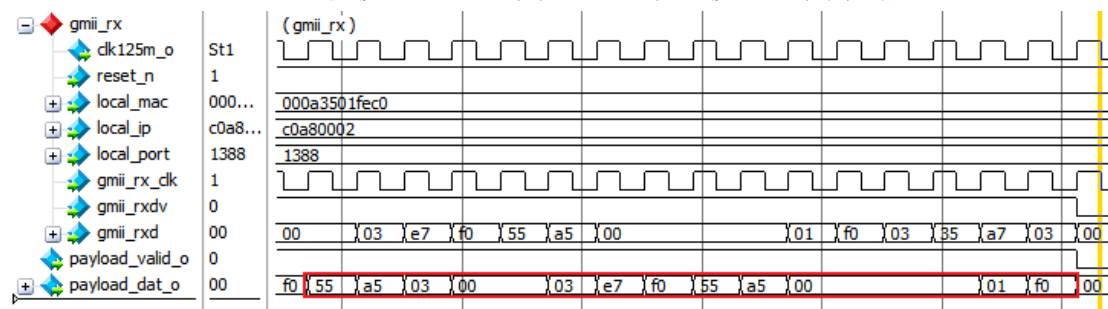


图 16 GMII 以太网接收模块仿真

通过观察 payload_dat_o 数据与下发指令对比可以判断指令是否被 FPGA 正确接收。

1.2.2.4 FIFO IP 核设计

FIFO (First In First Out)，即先进先出缓存器。在指令下达过程中，千兆以太网接收模块工作时钟为 125MHz，而 ADC128S102 控制器驱动模块的工作时钟为 64MHz，在对 ADC128S102 控制器采集设置时，由于两者数据速率不匹配，不能使用 125MHz 速率的数据来直接配置 64MHz 速率的驱动进行数据采集，否则在指令传达过程中会出现包括亚稳态问题在内的一系列问题，导致指令传达失败，所以这里使用一个具备双时钟结构的 FIFO 来进行指令数据的收发，实现在 64M 时钟下的指令传输控制。

FIFO 的配置信息如下，选择双时钟 FIFO 模式。数据的输入输出都设置为

八位，数据深度为 1024。

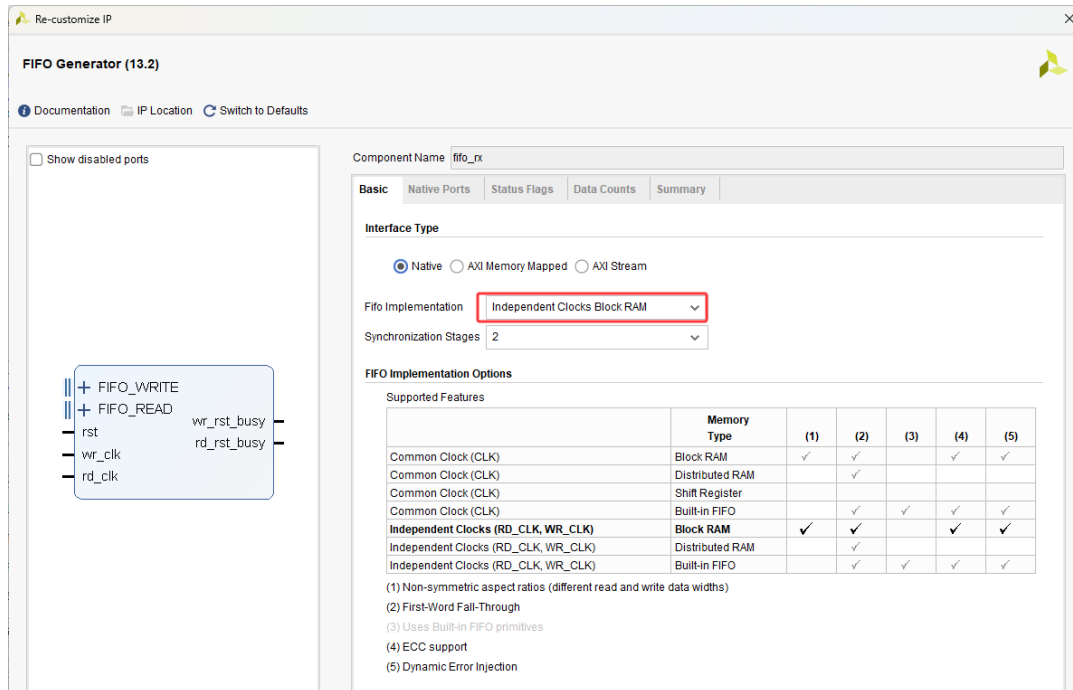


图 17 FIFO 配置信息图

选择 First Word Fall Through 模式，当前数据提前已经到读数据线上，在读使能到来后，下一个数据会到数据线上。

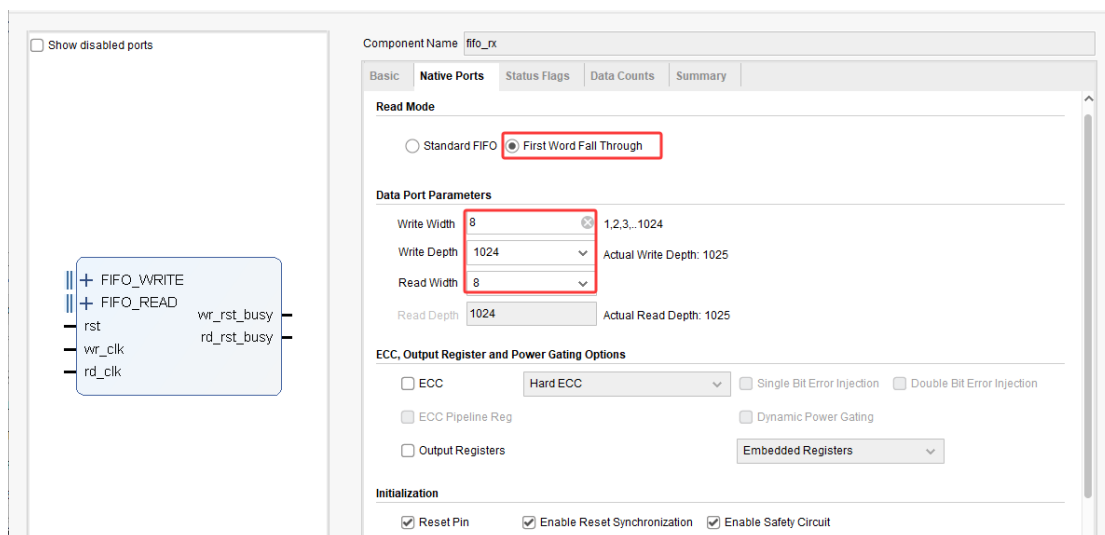


图 18 FIFO 配置信息图

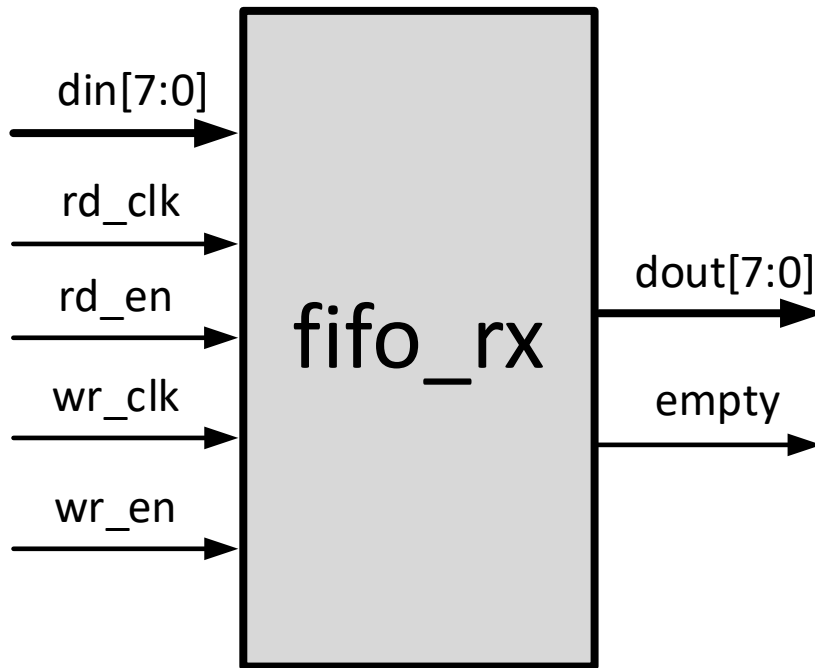


图 19 FIFO IP 核

接口名称	I/O	功能描述
din[7:0]	I	以太网数据位八位输入数据
rd_clk	I	双时钟 fifo 读时钟信号
rd_en	I	读使能信号
wr_clk	I	双时钟 fifo 写时钟信号
wr_en	I	写控制使能信号
dout[7:0]	O	fifo 缓存后八位数据输出
empty	O	fifo 读空信号

wr_clk 输入 125MHz 时钟信号，wr_en、din[7:0]信号同步于 wr_clk，wr_en 有效时使 FIFO 通过 din[7:0]开始往里面写数据。rd_clk 输入 64MHz 时钟信号，同样的，empty、dout[7:0]、rd_en 同步于 rd_clk，当 rd_en 有效时外部从 FIFO 读数据。

指令端 fifo 的写控制由 payload_valid_o 控制，payload_valid_o 有效时向 fifo 中写入 GMII 以太网 payload_dat_o 数据。读控制由接收转命令模块 fifo_red_req 信号进行控制。

1.2.2.5 命令解析模块设计

接收转命令该模块的作用就是将网口接收到的数据拆解成寄存器的数据帧，同时对寄存器的数据帧进行参数提取，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 ADC128S102 进行相对应的配置。

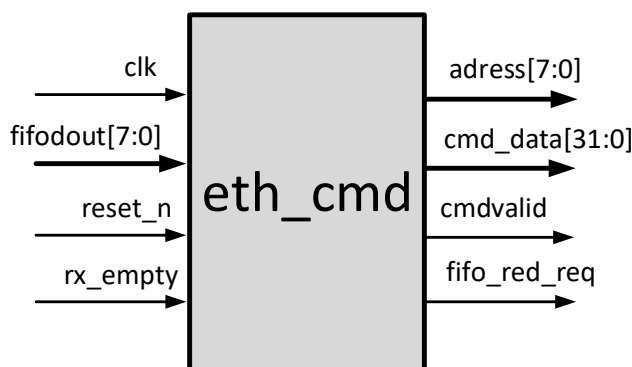


图 20 接收转命令模块

接口名称	I/O	功能描述
clk	I	模块工作时钟 50MHz
reset_n	I	模块复位信号
fifodout[7:0]	I	读的 fifo 的八位数据，数据输入后进行指令数据的解析
rx_empty	I	fifo 空标志信号
adress	O	八位地址位输出
cmdvalid	O	指令数据解析完成标志信号，在一次指令判断完成后输出
fifo_red_req	O	fifo 读使能信号，控制对 fifo 进行读数据
cmd_data[31:0]	O	寄存器参数数据

当 `fifo_rd_req` 有效时开始赋值，8 位 `fifodout` 的值开始赋给 `[7:0]data_0[7:0]` 数组高八位，`data_0` 代表寄存器的数据帧，连续对 `data_0` 进行赋值。

`fifo_rx_done` 同步于 `fifo_rd_req`，有效时该模块会对数组 `data_0` 数据进行判定。当数据符合 D0 为 8' d55，D1 为 8' dA5，D7 为 8' dF0，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。同时数组 `data_0[3:6]` 赋值给 `cmd_data[0:31]` 指令参数，`data_0[2]` 赋值给 `address` 地址参数，完成寄存器的数据帧的解析。代码设计如下。

```
always@(posedge clk or negedge reset_n)
```

```
if(!reset_n)
    fifo_rd_req <= 1'b0;
else if(!rx_empty)
    fifo_rd_req <= 1'b1;
else
    fifo_rd_req <= 1'b0;

always@(posedge clk)
if(fifo_rd_req)begin
    data_0[7] <= #1 fifodout;
    data_0[6] <= #1 data_0[7];
    data_0[5] <= #1 data_0[6];
    data_0[4] <= #1 data_0[5];
    data_0[3] <= #1 data_0[4];
    data_0[2] <= #1 data_0[3];
    data_0[1] <= #1 data_0[2];
    data_0[0] <= #1 data_0[1];
end

reg fifo_rx_done;
always@(posedge clk)
    fifo_rx_done <= fifo_rd_req;
always@(posedge clk or negedge reset_n)
if(!reset_n)begin
    address <= 0;
    cmd_data <= 32'd0;
    cmdvalid <= 1'b0;
end
else if(fifo_rx_done)begin
    if((data_0[0]==8'h55)&& (data_0[1] == 8'hA5) && (data_0[7] == 8'hF0))begin
        cmd_data[7:0] <= #1 data_0[6];
        cmd_data[15:8] <= #1 data_0[5];
        cmd_data[23:16] <= #1 data_0[4];
        cmd_data[31:24] <= #1 data_0[3];
        address <= #1 data_0[2];
        cmdvalid <= #1 1;
    end
end
else
    cmdvalid <= #1 0;
```

end

仿真中可以观察到，fifo 输入的寄存器数据帧在经过解析后得到寄存器地址值 address，参数值 cmd_data 以及解析完成标志信号 cmdvalid，实现了数据帧到配置数据的转换，如所示。

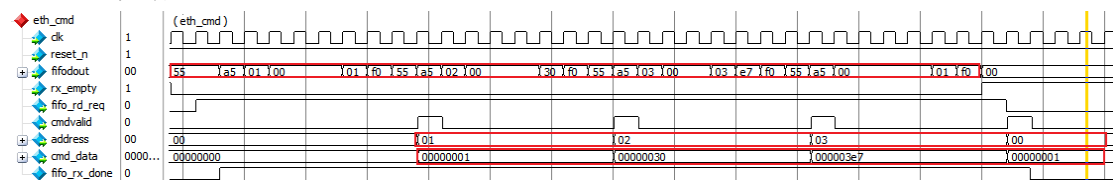


图 21 指令帧解析模块仿真

1.2.2.6 指令转控制模块设计

该模块的作用旨在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，每当接收到 cmdvalid 信号时，该模块便通过地址信号 address 判断对那个寄存器执行操作，并从数据中提取对应的数据将其输出到相应的寄存器。

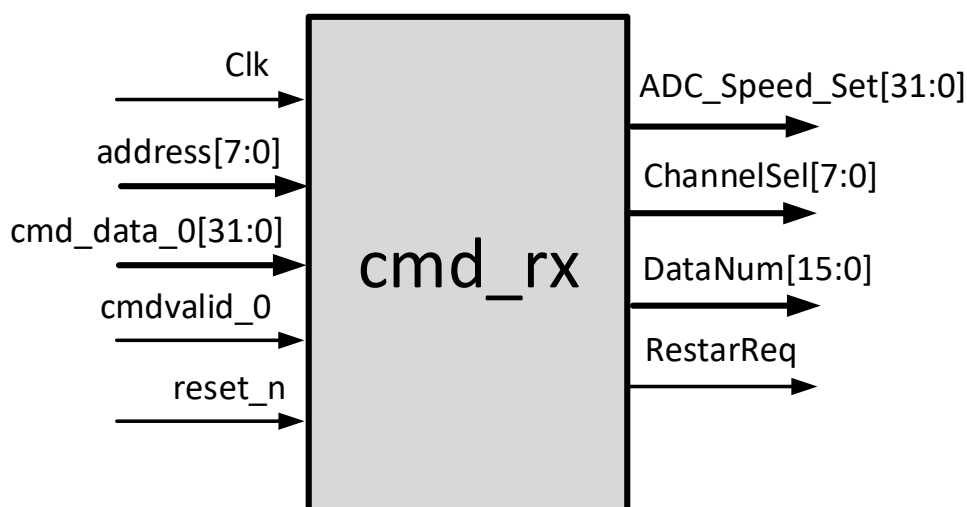


图 22 指令转控制模块

接口名称	I/O	功能描述
Clk	I	模块工作时钟 50MHz
adrsss[7:0]	I	地址数据
cmd_data_0[31:0]	I	参数数据
cmdvalid_0	I	转换有效标志信号
reset_n	I	模块复位信号

ADC_Speed_Set[31:0]	O	采样频率设置数据，代码中默认最高采样速率 1M，每个通道也就是 125K 的采样速率
ChannelSel[7:0]	O	采样通道设置数据
DataNum[15:0]	O	采样量设置数据
RestarReq	O	采样请求信号

当 cmdvalid 有效时，根据 address_0 的不同对应的分别给 RestartReq、ChannelSel、DataNum、ADC_Speed_Set 赋值。代码设计如下。

```
always@(posedge Clk or negedge Reset_n)

    if(!Reset_n)begin

        ChannelSel <= 8'b1111_1111;

        DataNum <= 16'd32;

        ADC_Speed_Set <= 32'd9999;

        RestartReq <= 1'b0;

    end

    else if(cmdvalid)begin

        case(cmd_addr)

            0: RestartReq <= 1'b1;

            1: ChannelSel <= cmd_data[7:0];

            2: DataNum <= cmd_data[15:0];

            3: ADC_Speed_Set <= cmd_data;

        end

        default::;

    endcase

end

else

    RestartReq <= 1'b0;
```

对 cmd_rx 模块进行仿真可以观察到，根据寄存器地址分别对采样通道 ChannelSel、采样数据量 DataNum、采样频率 ChannelSel、采样使能 RestarReq 进行参数设置，如图 23 所示。

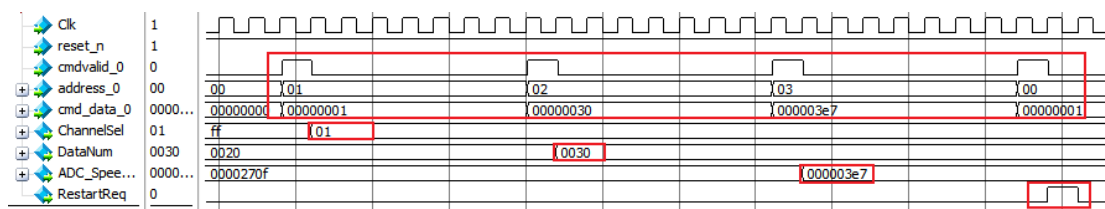


图 23 指令转控制模块仿真

到这里指令流的设计就全部完成了，通过对调用和设计的模块的验证仿真可以对整个指令流设计进行验证，确保 ADC128S102 控制模块在接收到指令数据后能够按照配置要求进行数据的采集。

1.2.3 数据传输设计

在正确配置指令后系统可以按照需求进行数据的采集，下面对采集的数据进行传输控制发送到 PC 端。

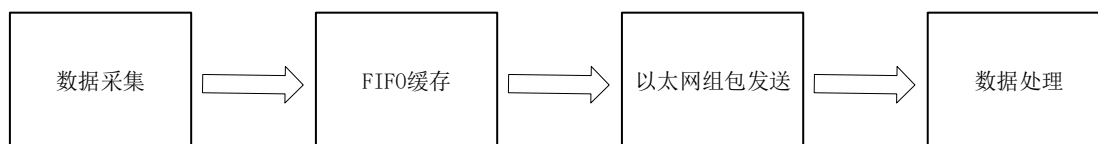


图 24 数据流设计流程

数据在本次设计中要经过上面的流程。首先 ADC128S102 采集数据后，ADC128S102 控制器驱动模块根据指令对寄存器配置信息采集数据。数据采集后首先要考虑的就是数据的缓存问题，以太网的高速通信能力可以保证数据的实时传输，但以太网和 ADC128S102 控制器驱动模块的工作时钟不一致以及考虑到数据的位宽转换，需要设计 FIFO 对数据作缓存处理。经过 FIFO 缓存后的数据就可以利用以太网发送模块进行组包发送到 PC 端上位机软件，PC 端对收集的数据可以进行分析验证。到此整个的系统功能才算完全实现，下面对数据传输各个功能模块进行设计实现。

1.2.3.1 ADC128S102 控制器驱动模块运用

该控制器实现了对 ADC128S102 型 8 通道 12 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 ADC128S102 的具体控制时序，一切都在控制器内部完成，用户只需要像使用串行 ADC 一样取用数据即可。

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、Rst_n），第二类是 ADC128S102 控制器与 ADC128S102 芯片管脚相

连的各种功能控制和数据信号，第三类是设置控制器工作状态/工作数据的用户控制接口，第四类是控制器的结果输出接口。对于用户来说，只需要关注第三类和第四类接口的使用，即可快速高效的使用该控制器来控制 ADC128S102 芯片完成数据转换。

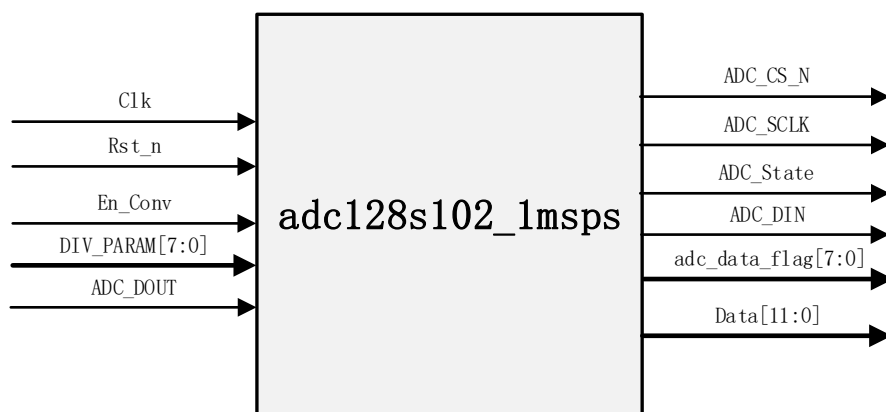


图 25 ADC128S102 控制器接口说明

下表为端口信号功能介绍表，对 ADC128S102 控制器四类接口进行详细的说明。

类	信号名	位	功能简介
	Clk	1	系统时钟，64MHz
	Rst_n	1	系统复位，低电平复位
控制信号	En_Conv	1	使能单次转换，该信号为单周期有效，高脉冲使能一次转换
	DIV_PARAM	8	时钟分频设置，实际 SCLK 时钟 频率 = fclk / (DIV_PARAM * 2)
数据结果 输出端口 和标志信号	data_flag	8	转换结果有效标志信号，因为 ADC128S102 有 8 个通道，转换结果是依次输出，并非同时的，所以设置 8 个 Flag 信号，每个通道的结果就绪之后，就产生一个 Flag 信号，通知外部可以取用。另外，如果只关心其中的部分通道，则只需要关心 data_flag 中对应的位即可。
	Data	12	多通道数据输出端口，该通道 12 位，在不同的时刻，输出不同通道的转换结果，使用时，与 data_flag 信号配合，data_flag 的哪一位出现高脉冲，则代表当前 Data 的值为该通道的转换结果。该端口设计的目的用于往 FIFO、RAM 等存储器中存储结果时使用。
ADC 芯片 控制信号	ADC_SCLK	1	ADC 串行数据接口时钟信号
	ADC_CS_N	1	ADC 串行数据接口使能信号
	ADC_DOUT	1	ADC 转换结果，由 ADC 输给 FPGA
	ADC_DIN	1	ADC 控制信号输出，由 FPGA 发送通道控制字给 ADC

该控制器根据 DIV_PARAM 的值计算得出其采样频率，比如本次实验的系统时钟为 64M，DIV_PARAM 的值为 2，那么实际的 SCLK 时钟为 12.5M，得到的采样速率大约为 1M（16M/16），对应到每个通道的采样速率为 125K（1M/8）。最终在存储数据的时候可以根据 data_flag 的值存储到对应通道的数

据 Data。

1.2.3.2 ADC 采集控制模块设计

ADC128S102 采集的数据由 `adc_data_mult_ch` 输入 ADC 采集控制模块，需要注意的是 ADC128S102 的采样位宽为 12 位的，但是最终交由 FIFO 存储的位宽为 16 位的，所以我们需要将 ADC128S102 采样的数据高位补 4 个 0 之后再交由 `adc_data_mult_ch` 输入 ADC 采集控制模块。

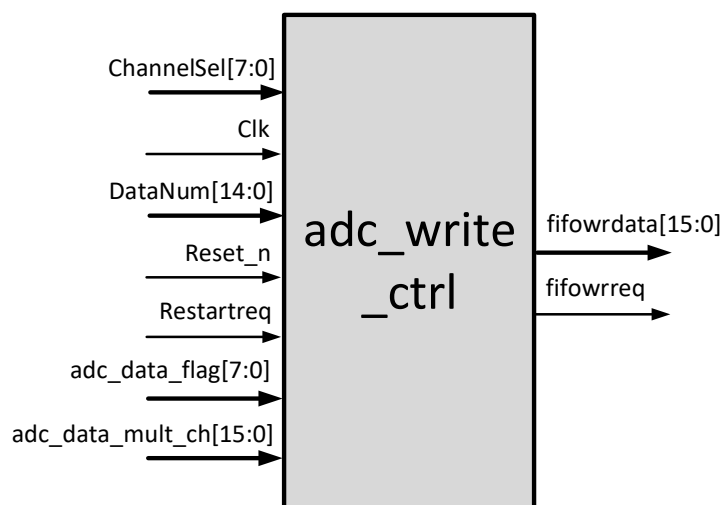


图 26 ADC 采集控制模块

接口名称	I/O	功能描述
Clk	I	模块工作时钟，64MHz
Reset_n	I	模块复位信号
ChannelSel[7:0]	I	数据采集通道设置信号
DataNum[15:0]	I	采集数据量设置信号
Restartreq	I	采样有效请求信号
adc_data_flag[7:0]	I	数据通道有效标志
adc_data_mult_ch[15:0]	I	通道数据值
fifowrdata[15:0]	O	采集数据
fifowrrreq	O	采集到的数据写 FIFO 请求信号

同时，输入 `adc_data_flag` 采样标志信号，使不同通道采集数据有效时产生对应位的有效信号，ADC 采集控制模块根据 `adc_data_flag & ChannelSel` 有效输出 `fifowrrreq` 信号用于读取所需通道的数据。采集的数据存入 FIFO，经由 FIFO 后通过以太网发送模块发送到 PC 端。采集的数据 `adc_data_mult_ch` 是 16 位的，两个字节。以太网发送模块数据接收信号共 8 根信号线，每次可从 FIFO 读取

一字节数据，读数据先读低八位。

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

always@(posedge Clk)
    fifowrreq <= #1 (adc_data_flag & ChannelSel) && sample_en;
//fifo 先读低字节
always@(posedge Clk)
begin
    fifowrdata[7:0] <= #1 adc_data_mult_ch[7:0];
    fifowrdata[15:8] <= #1 adc_data_mult_ch[15:8];
end
```

通过仿真可以观察到，当设置采集数据量为 16'h30 时，data_mult_ch 最大为 16'h2f，与写入 fifo_tx 中的值保持一致，数据量与设置的 16'h30 保持一致。

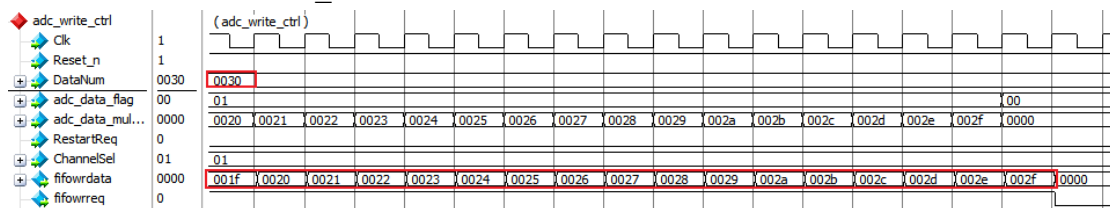


图 27 ADC 采集控制模块仿真验证

1.2.3.3 fifo_tx IP 核配置

fifo_tx 模块需要接收从 ADC 采集控制模块输出的 16 位数据，数据经缓存后由 GMII 以太网发送模块读取。ADC 采集控制模块的工作时钟为 64MHz，千兆以太网接收模块工作时钟为 125MHz，模块 fifo_tx 仍需配置为一个具备双时钟结构的 FIFO 进行异步数据的收发。

店铺: <https://xiaomeige.taobao.com>

官方网站: www.corecourse.cn

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

技术群组:

选择双时钟 FIFO 模式。FIFO 读位宽设置的为 16 位，写位宽设置的为 8 位，数据深度设置为 16384。

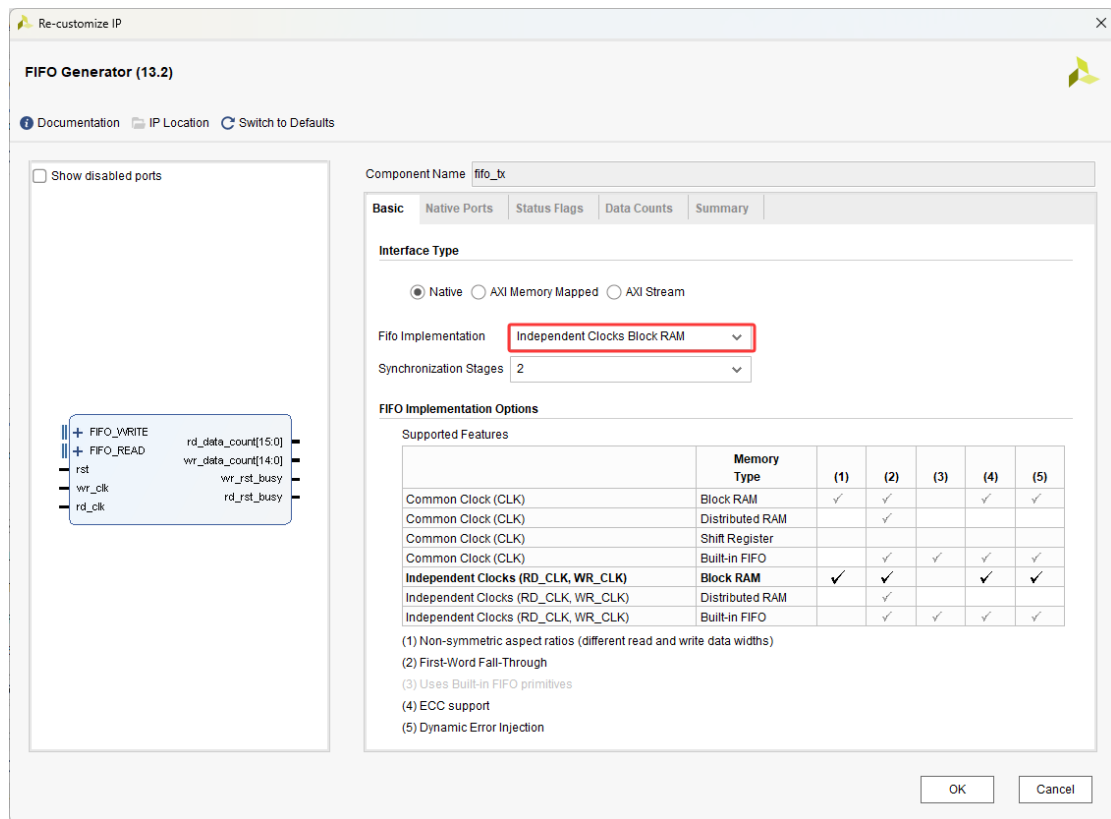


图 28 fifo_tx 配置图

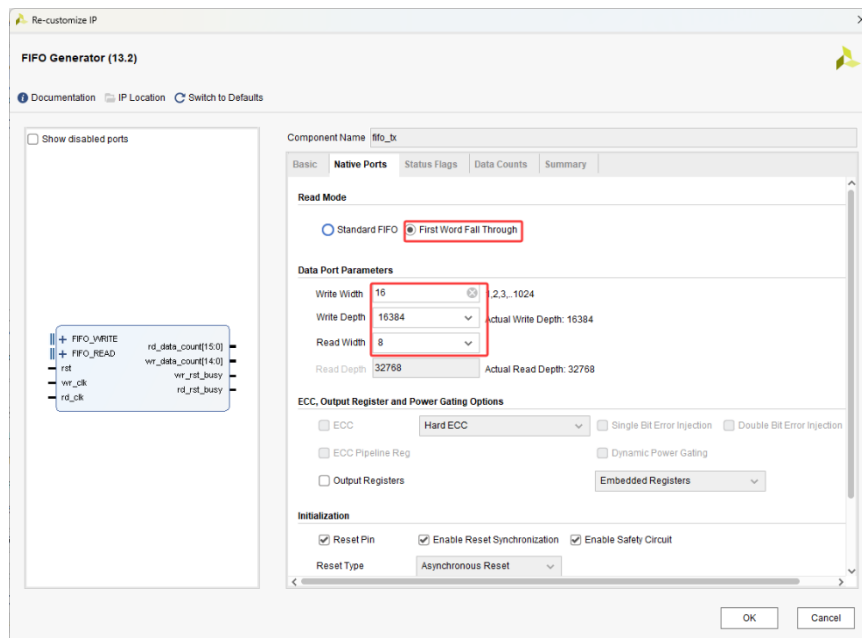


图 29 fifo_tx 配置图

店铺: <https://xiaomeige.taobao.com>官方网站: www.corecourse.cn技术博客: <http://www.cnblogs.com/xiaomeige/>

技术群组:

这里需引出计数接口，输出 FIFO 中可读取数据的数据量，用于 FIFO 读控制的参考信号。配置信息如下图所示。

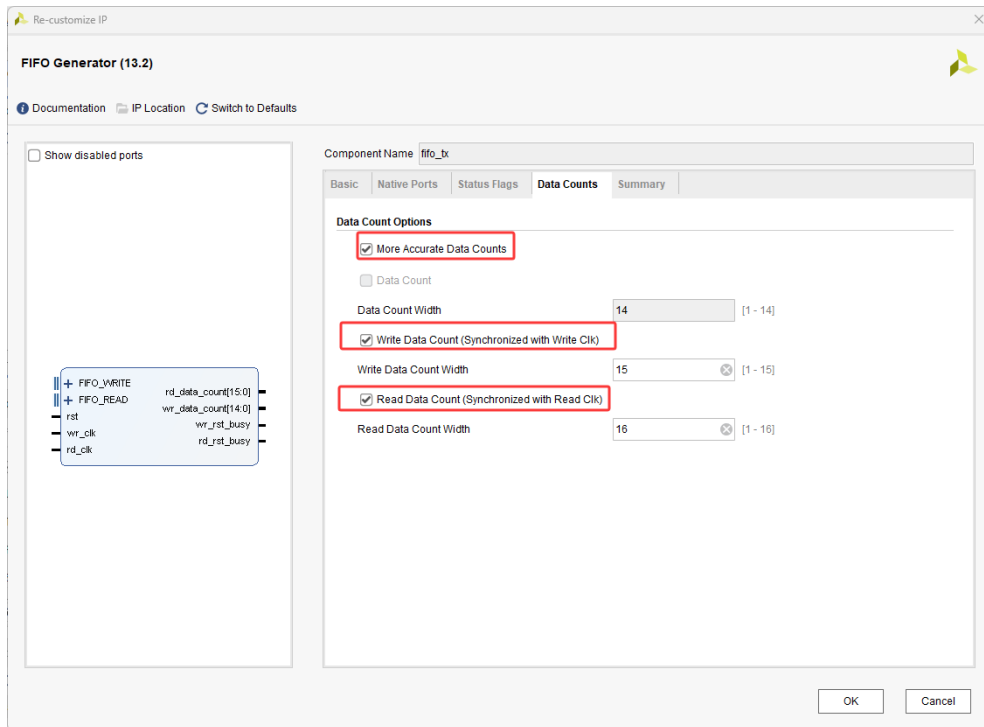


图 30 fifo_tx 配置图

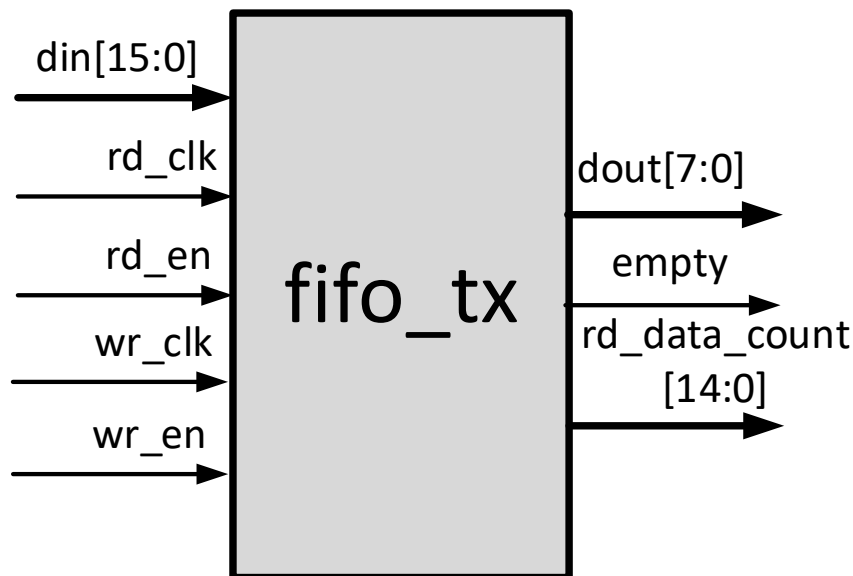


图 31 fifo_tx IP 核

接口名称	I/O	功能描述
rd_clk	I	tx_fifo 读时钟信号, 125MHz
wr_clk	I	tx_fifo 写时钟信号, 64MHz
din[15:0]	I	fifo 写数据
rd_en	I	读使能信号, 有效时读数据
wr_en	I	写使能信号, 有效时写数据
dout[7:0]	O	fifo 读数据
empty	O	fifo 空标志信号
rd_data_count[14:0]	O	fifo 读计数值

wr_clk 连接 Clk 输入 64MHz 时钟信号, wr_en、din[15:0]信号同步于 wr_clk, FIFO 写数据由 ADC 采集控制模块 fifowrreq 控制, 命令设置通道数据有效时使通过 din[15:0]往 FIFO 写数据。rd_clk 输入 125MHz 时钟信号, empty、dout[7: 0]、rdreq、usedw 同步于 rd_clk, 当 rdreq 有效时外部从 FIFO 读数据, rdusedw 信号输出到网口发送控制模块 eth_send_ctrl。

对 fifo_tx 仿真验证, 数据输出与预期一致。

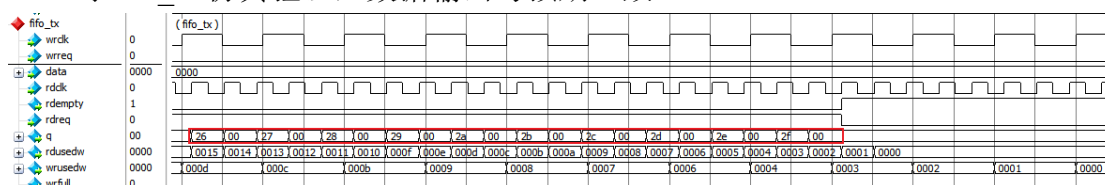


图 32 fifo_tx 仿真验证

1.2.3.4 数据发送控制模块设计

该模块主要负责配置控制网口发送模块的使能控制信号, 通过 pkt_length 信号对以太网数据帧数据长度进行控制。

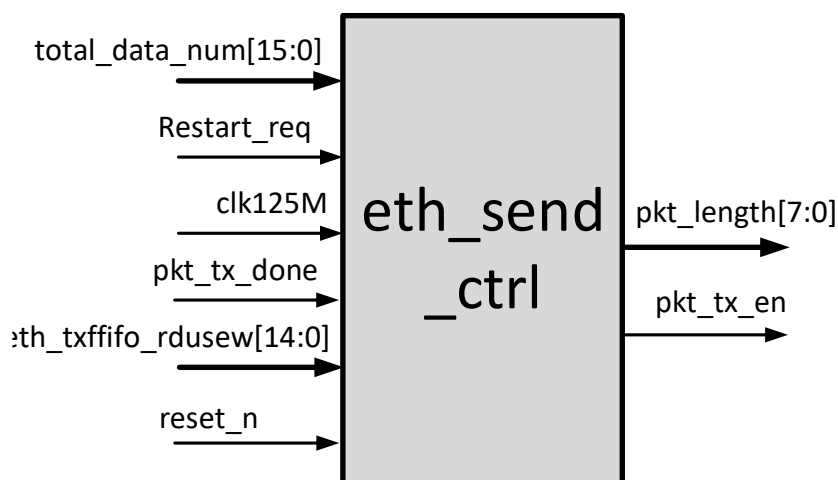


图 33 网口发送控制模块

接口名称	I/O	功能描述
total_data_num [15:0]	I	采集数据量
restart_req	I	采样有效请求信号
clk125M	I	模块工作时钟 125MHz
pkt_tx_done	I	gmii 发送完成信号
eth_txfifo_rdusew [14:0]	I	从 fifo 读数据计数
reset_n	I	模块复位信号
pkt_length [7:0]	O	一帧以太网数据长度设置量
pkt_tx_en	O	以太网发送使能脉冲信号

从 cmd_rx 模块获取采样请求信号 restart_req、采集的数据总量 total_data_num。

状态零：

restart_req 信号有效，系统开始采样，同时，根据 total_data_num 信号的值对数据帧发送的 ADC128S102 采集数据的最大长度进行设置。total_data_num 决定了采样的数据量，可根据 total_data_num 的值设置 pkt_length 的大小。

由于 ADC128S102 采样数据为 16 位，以太网每次接收数据为 8 位，为保证以太网有效传输效率，以太网帧最大长度 1518 字节（数据段 1500 字节），其中数据段 1500 字节还包括 20 字节 IP 报文头部和 8 字节 UDP 报文头部，所以数据帧发送的 ADC128S102 采集的数据最大长度为 1472 字节。当 total_data_num > 16'd736 时，pkt_length 的值为 1472，其余时候 lenh_val <=

total_data_num + total_data_num。

状态一：

pkt_tx_en 以太网发送使能脉冲信号，数据帧发送的 ADC128S102 采集数据的最大长度作为判断条件，FIFO 计数信号 eth_txfifo_rdusew 的数值满足一帧数据帧发送采集数据长度。满足则产生 pkt_tx_en 信号，开始读取 fifo 中的数据进行组包发送。

状态二：

拉低 pkt_tx_en，等待以太网帧发送完成信号，data_num - pkt_length/2 得到剩余待发送数据量。

状态三：

cnt_dly_min 计数延时。这里 cnt_dly_min 发送速率延时控制逻辑的目的在于保证以太网发送模块数据传输和处理的时间，降低 FPGA 每秒发包的次数，避免发包太快给 PC 网卡带来负担。

状态四：

判断余下的数据是否全部发送完成，依次判断是进入状态一继续进行发送还是进入状态零等待下一次发送指令。

代码设计如下。

```
parameter cnt_dly_min = 16'd256;

//ADC数据为16位，每个数据占2个字节，所以发送N个采样数据。则需要发送2*N个字节

//采集数据最大字节数：1500-IP报文头部（20字节）-UDP报文头部（8字节）=
1472字节
always@(posedge clk125M or negedge reset_n)
if(!reset_n)begin
    pkt_length <= 16'd0;
    data_num <= 32'd0;
    state <= 0;
    pkt_tx_en <= 1'b0;
    cnt_dly_time <= 28'd0;
end

else begin
    case(state)
        0:
            if(restart_req)begin
                data_num <= total_data_num;
                if((total_data_num << 1) >= 16'd1472)begin
                    pkt_length <= 16'd1472; //一个数据2个字节
```

```
        state <= 1;
    end
    else if((total_data_num << 1) > 0)begin
        pkt_length <= total_data_num << 1;
        state <= 1;
    end
    else begin
        state <= 0;
    end
end

1:
    if(eth_txfifo_rdusew >= pkt_length)begin
        pkt_tx_en <= 1'b1;
        state <= 2;
    end
    else begin
        state <= 1;
        pkt_tx_en <= 1'b0;
    end

2:
    begin
        pkt_tx_en <= 1'b0;
        if(pkt_tx_done)begin
            data_num <= data_num - pkt_length/2;
            state <= 3;
        end
    end

3:
    if(cnt_dly_time >= cnt_dly_min)begin
        state <= 4;
        cnt_dly_time <= 28'd0;
    end
    else begin
        cnt_dly_time <= cnt_dly_time + 1'b1;
        state <= 3;
    end

4:
    begin
        if(data_num * 2 >= 16'd1472)begin
            pkt_length <= 16'd1472;
            state <= 1;
        end
        else if(data_num * 2 > 0)begin
            pkt_length <= data_num * 2;
            state <= 1;
        end
        else begin
            state <= 0;
        end
    end

end
```



```
        default: state <= 0;  
    endcase  
end  
  
endmodule
```

1.2.3.5 网口发送模块运用

网口发送模块在获取到采集数据并对数据进行打包，然后以以太网帧的形式发送到 PC 端。为配置以太网数据帧，网口发送模块需要获取目的 MAC 地址（dst_mac）、目的 IP 地址(dst_ip)、目的端口号(dst_port)、源 MAC 地址(src_mac)、源 IP 地址(src_ip)、源端口号(src_port)。上板验证中目的 MAC 地址为电脑的实际 MAC 地址，如果不知道自己电脑网卡的 MAC 地址，就在 DOS 命令窗口，用 ipconfig - all 命令查看。

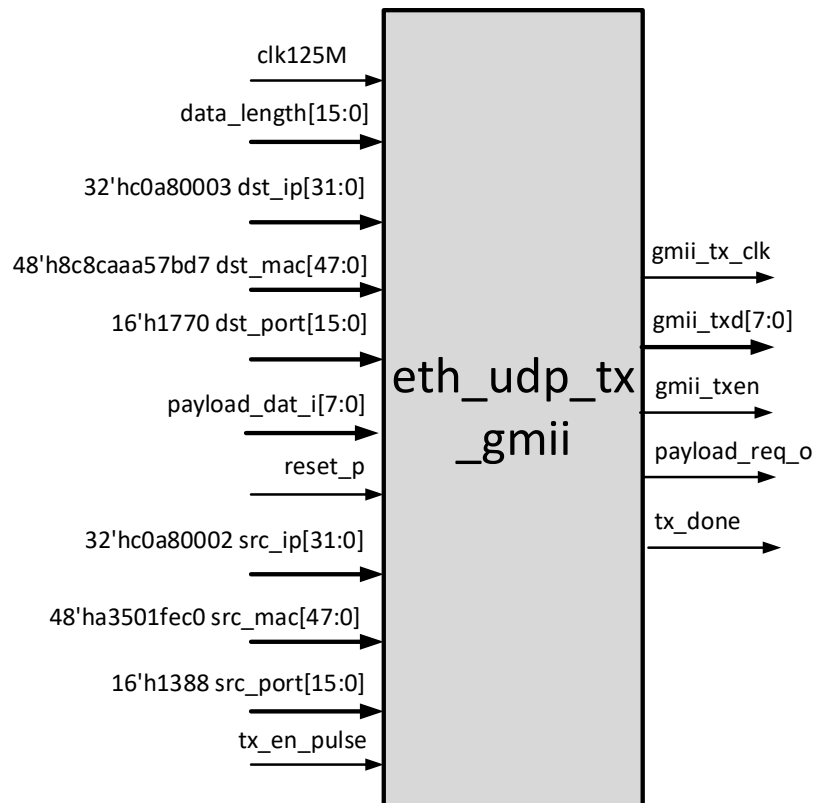


图 34 网口发送模块

接口名称	I/O	功能描述
CLK125M	I	gmii 工作时钟信号
data_length[15:0]	I	采集数据发送长度

dst_ip[31:0]	I	目的 IP 地址
dst_mac[47:0]	I	目的 mac 地址
dst_port[15:0]	I	目的地址端口号
payload_dat_i[7:0]	I	八位数据信号
reset	I	模块复位信号
src_ip[31:0]	I	源 IP 地址
src_mac[47:0]	I	源 MAC 地址
src_port[15:0]	I	源端口号
tx_en_pulse	I	发送脉冲标志信号
gmii_tx_clk	O	输出时钟信号, 125MHz
gmii_txd[7:0]	O	gmii 八位数据发送信号
gmii_txen	O	发送有效信号
payload_req_o	O	数据请求信号, 有效时接收数据
tx_done	O	发送完成标志信号

网口发送模块接收到 tx_en_pulse 信号后, 输出 payload_req_o 信号开始读取 fifo_tx 中的采集数据, 每次采集一字节数据。GMII 接口中 Transmit Data 数据发送信号, 共 8 根信号线, 当 gmii_txen 有效, gmii_txd 开始向 PC 端发送数据, 每次可发送一字节数据。从 FIFO 读取采集数据的数据量由 eth_send_ctrl 模块的 lenth_val 输出值决定, 一次数据帧发送完成网口发送模块产生 tx_done 信号, 等待下一次的发送。

1.2.3.6 RGMII 发送接口实现

对于 FPGA 来说, 实现 RGMII 接口的发送是一个非常直接的过程, 整个发送逻辑框图如图 35 所示:

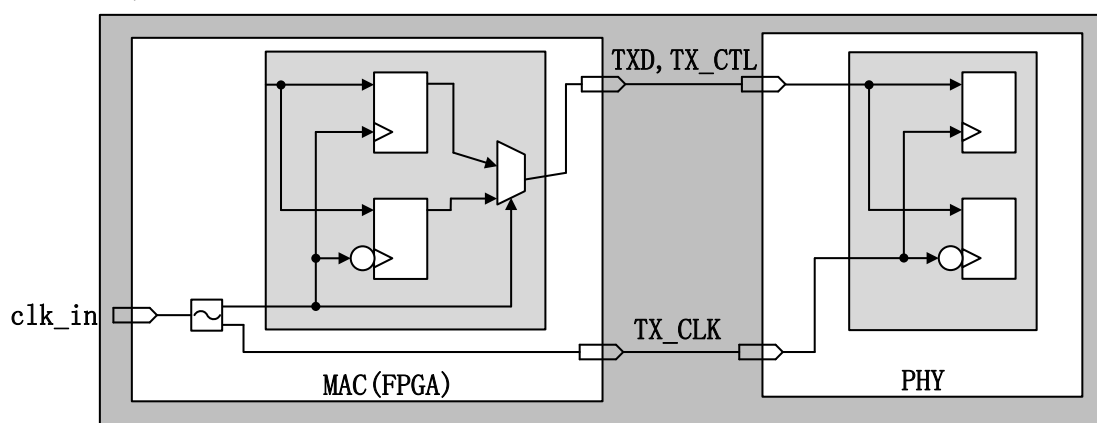


图 35 RGMII 发送设计逻辑框图

设计实现时, 我们需要使用 xilinx 的 ODDR (Output Double Data Rate, 输

出双倍数据速率)原语,将该接口使用 OLOGIC 块实现。OLOGIC 块在 7 系列 FPGA 内的位置紧挨着 IOB,其作用是 FPGA 通过 IOB 发送数据到器件外部的专用同步块。在 OLOGIC 块中,有着专用的寄存器,用于实现输出 DDR 寄存器,当我们实例化 ODDR 原语时便会自动访问该功能。

ODDR 原语只有一个时钟输入,下降沿数据由输入时钟的本地反转来计时,反馈到 I/O 块的所有的时钟被完全复用,ODDR 原语的框图如图 36 所示:

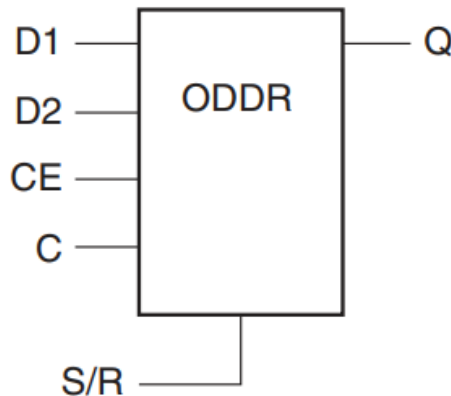


图 36 ODDR 原语框图

其中各个端口的功能及描述如表 4:

表 4 ODDR 端口信号

Port Name	Function	Description
Q	数据输出 (DDR)	ODDR 寄存器输出。
C	时钟输入端口	时钟输入引脚
CE	时钟使能端口	CE 表示时钟使能引脚。当断言为低时,此端口禁用端口 Q 上的输出时钟。
D1 和 D2	数据输入	ODDR 寄存器输入
S/R	设置/复位	同步/异步的设置/复位引脚。该引脚高电平有效

这里的 D1 和 D2 是数据的两个输入端口,输出端口会在上升沿这里需要注意的是 set 和 reset 同时只能有一个被置高,也因此,描述端口时,使用的 S/R。除了这些端口外,ODDR 原语还包含一些可用属性,具体如表 5 所示:

表 5 ODDR 属性

Attribute Name	Description	Possible Values
DDR_CLK_ED GE	根据时钟边沿设置 ODDR 操作模式	OPPOSITE_EDGE(默认), SAME_EDGE
INIT	设置 Q 端口的初始值	0 (默认), 1
SRTYPE	相对于时钟的设置/重置类型 (C)	ASync, SYNC (默认)

这里简单描述下 ODDR 的两种操作模式:

店铺: <https://xiaomeige.taobao.com>

官方网站: www.corecourse.cn

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

技术群组:

1. OPPOSITE_EDGE 模式

在该模式中，时钟的两个边沿被用来以两倍的吞吐量从 FPGA 逻辑中捕获数据。这种结构与 virtex-6 的实现比较相似。两个输出都提供给 IOB 的数据输入或者三态控制输入。使用 OPPOSITE_EDGE 模式的输出 DDR 时序图如图 37 所示：

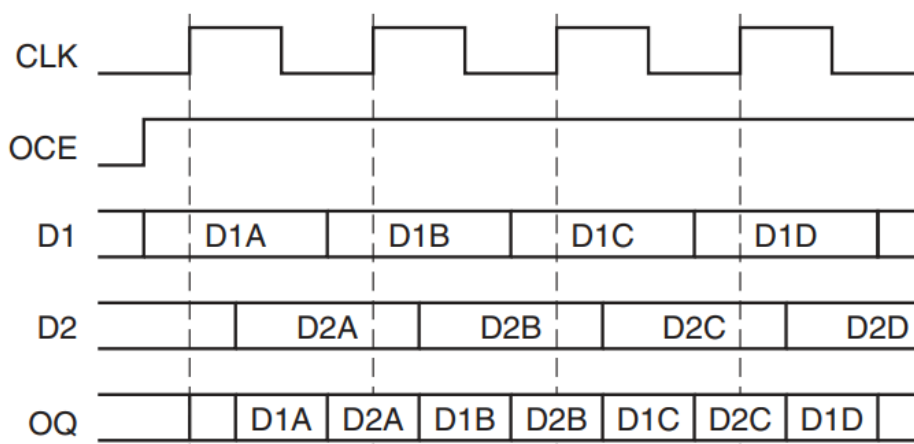


图 37 OPPOSITE_EDGE 模式下 ODDR 时序

在该模式下，输入数据在两个边沿被采样，可以看到输出端 OQ 上首先输出的是 D1A，随后再输出 D2A。

2. SAME_EDGE 模式

在该模式下，数据可以在相同的时钟边沿送给 IOB。相同的时钟沿将数据送给 IOB 可以避免建立时间违规，并允许用户使用最小的寄存器来执行更高的 DDR 频率来进行寄存器的延迟，而不是使用 CLB 寄存器。图 38 显示了使用 SAME_EDGE 模式的输出 DDR 的时序图：

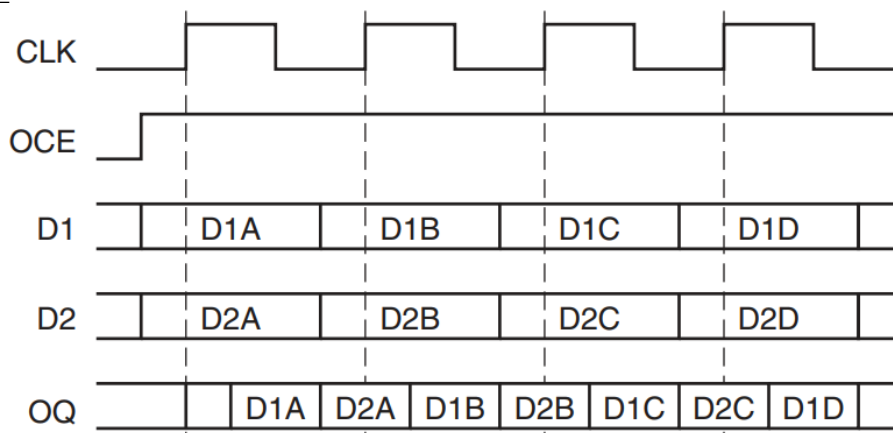


图 38 SAME_EDGE 模式下 ODDR 时序

可以看到，在该模式下，输出端 OQ 同样也是先输出 D1 端采集到的值，再输出 D2 端采集到的值。

这两种操作模式可以在使用 ODDR 原语时使用 DDR_CLK_EDGE 的属性值设置，在 Xilinx 的官方手册 UG768 中，给出了 ODDR 原语的例化示例，如下：

```
// ODDR: Output Double Data Rate Output Register with Set, Reset
// and Clock Enable.
// 7 Series
// Xilinx HDL Libraries Guide, version 14.7
ODDR #(
    .DDR_CLK_EDGE("OPPOSITE_EDGE"), // "OPPOSITE_EDGE" or "SAME_EDGE"
    .INIT(1'b0), // Initial value of Q: 1'b0 or 1'b1
    .SRTYPE("SYNC") // Set/Reset type: "SYNC" or "ASYNC"
) ODDR_inst (
    .Q(Q), // 1-bit DDR output
    .C(C), // 1-bit clock input
    .CE(CE), // 1-bit clock enable input
    .D1(D1), // 1-bit data input (positive edge)
    .D2(D2), // 1-bit data input (negative edge)
    .R(R), // 1-bit reset
    .S(S) // 1-bit set
);
// End of ODDR_inst instantiation
```

在例化 ODDR 原语时，值得注意的一点是，一个 8 位的 GMII 数据在转换为 RGMII 后，在支持 RGMII 的 PHY 芯片会在时钟上升沿发送数据的低 4 位，在时钟的下降沿发送数据的高 4 位。因此，在例化 ODDR 原语时，我们需要将数据的高 4 位数据连接到 D2 端口，将低 4 位连接到 D1 端口，以满足 PHY 对数据的格式需求。

对于数据和时钟传输，大多数支持 RGMII 的 PHY 芯片都提供了一个对发送时钟 TX_CLK 添加延迟的选项。可以根据设计要求启用或禁用此选项。当启用延迟 PHY 设备内 TX_CLK 的选项时，FPGA 必须生成与数据和波形边缘对齐的时钟，如图 39 所示：

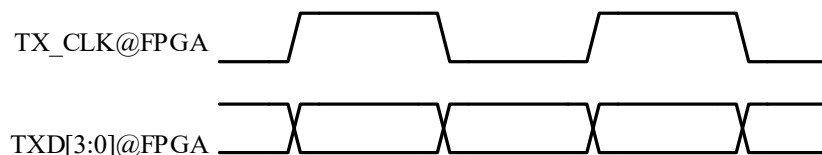


图 39 时钟与数据对齐关系（FPGA，启用延迟）

因此，在这种情况下，使用 ODDR 原语时，必须将 DDR_CLK_EDGE 的属性

性值设置为 SAME_EDGE。这样，PHY 芯片在对 TX_CLK 时钟进行相移后才能与数据中心对齐。

如果 PHY 的该功能被关闭，则 FPGA 必须对时钟信号进行一定的相位调整后再作为 TX_CLK 输出，以使 TXD 和 TX_CLK 成中心对齐关系。当然，如果 PCB 上的 TX_CLK 信号线路本身就引入了一定的延迟，则需要将这一部分延迟考虑在内，再计算 FPGA 需要对 TX_CLK 调整的相位值，其传输波形如图 40 所示：

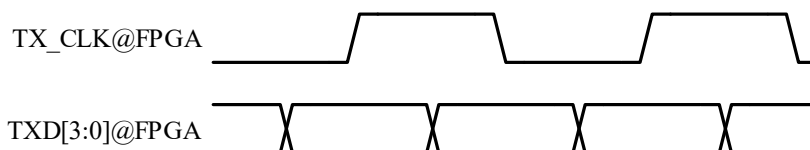


图 40 时钟与数据对齐关系（FPGA，禁用延迟）

将源时钟与数据对齐的方法有多种，例如，我们可以使用 PLL 产生 2 路相位相差 90 度的时钟信号，一路作为 ODDR 寄存器的工作时钟，一路作为 TX_CLK 输出。当然，如果考虑到 PCB 板上的 TX_CLK 相对于 TXD 的延迟值，该相位可以进一步调整以确保 TX_CLK 到达 PHY 时与 TXD 呈中心对齐关系。如图 41 所示：

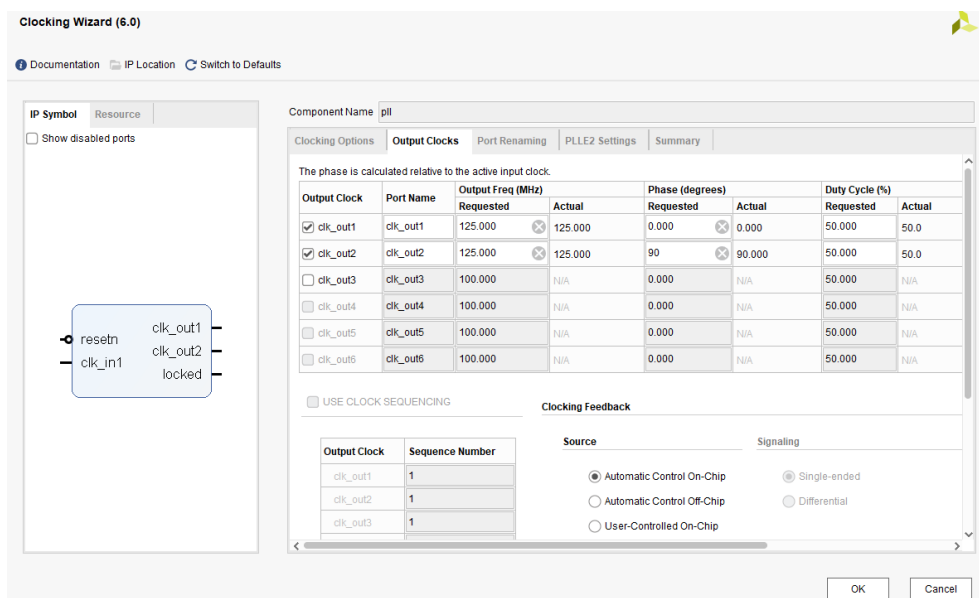


图 41 使用 PLL 产生同源相移时钟

使用 PLL 能够实现任意相位差的两路时钟信号，可以随意调整时钟和数据传输的对齐关系，但是这种方式会占用更多 FPGA 资源。因此，在使用时，用户可以根据实际情况选择不同的方案。

1.2.4 系统的功能仿真测试

经过上述工作，我们已经完成了基于以太网的数据采集系统各个功能单元的设计和验证。事实上，对这些功能单元的验证，我们并没有针对每个单元分别设计仿真测试激励（Testbench）文件，而是直接对整个系统顶层设计了一个模拟以太网指令发送的测试激励，通过在仿真测试激励中模拟产生以太网指令并传递给数据采集系统，以测试数据采集系统对指令的接收、解析以及执行的过程。

在 Testbench 中产生以太网测试数据包的方法有很多，例如预先在数组中存储好要发送的指令数据帧，然后发送时依次读取数组中的内容并发出。或者编写一个以太网帧发送任务，对需要发送的数据进行组包并发送。本设计采用了最简单的方法，也就是直接使用设计中用到的以太网发送逻辑，例化在 Testbench 中，将其当做一个以太网帧生成器。需要发送的数据送给该模块后，模块就能自动组建出正确的 UDP 数据包并发送，如图所示。由于该以太网发送逻辑是已经经过了反复验证的，所以用来产生测试以太网帧也是非常可靠的。

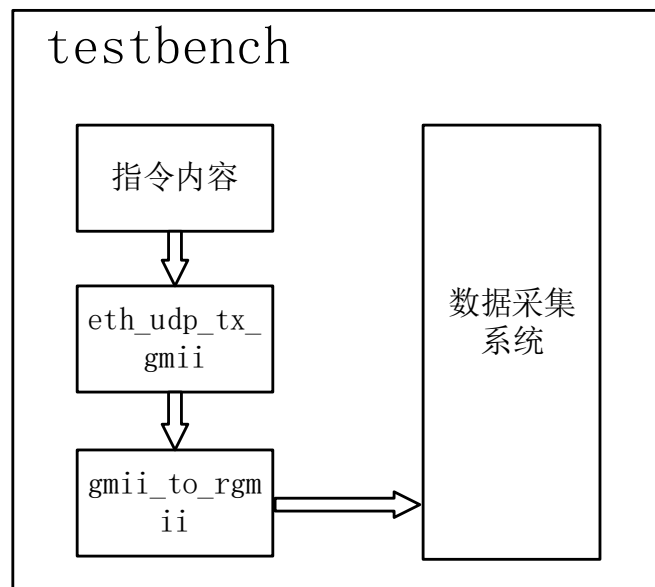


图 42 以太网测试指令组包流程

以下为测试文件的完整内容。

```
`timescale 1ns / 1ps
`define CLK_PERIOD 8
//建立仿真时请替换顶层中ADC128S102_driver模块为ADC128S102_datacnt
module ADC128S102_eth_tb();
    reg Clk;
    reg reset_n;
```

```
reg eth_rxc;

wire led;
wire gmii_tx_clk;

reg [7:0]dout;
wire payload_req;
reg tx_en_pulse;
wire tx_done;

wire [3:0]eth_rxd;
wire eth_rxdv;
wire eth_gtxc;
wire [3:0]eth_txd;
wire eth_txen;
wire eth_rst_n;
wire eth_mdc;
wire eth_mdio;

wire gmii_gtxc;
wire [7:0] gmii_txd;
wire gmii_txen;
wire rgmii_gtxc;
wire [3:0] rgmii_txd;

assign eth_rxd = rgmii_txd;

ADC128S102_UDP_RGMII ADC128S102_UDP_RGMII_tb(
    .Clk(Clk),
    .reset_n(reset_n),

    .eth_rxc(eth_rxc),
    .eth_rxd(eth_rxd),
    .eth_rxdv(eth_rxdv),

    .led(led),

    .eth_gtxc(eth_gtxc),
    .eth_txd(eth_txd),
    .eth_txen(eth_txen),
    .eth_rst_n(eth_rst_n),
    .eth_mdc(eth_mdc),
    .eth_mdio(eth_mdio)
);

eth_udp_tx_gmii eth_udp_tx_gmii
(
    .clk125M(eth_rxc),
    .reset_n(reset_n),

    .tx_en_pulse(tx_en_pulse),
    .tx_done(tx_done),

    .dst_mac(48'h00_0a_35_01_fe_c0),
    .src_mac(48'h8C_8C_AA_A5_7B_D7),
    .dst_ip(32'hc0_a8_00_02),
```



```
.src_ip(32'hc0_a8_00_03),
.dst_port(16'd5000),
.src_port(16'd6000),

.data_length(16'd32),

.payload_req_o (payload_req),
.payload_dat_i (dout),

.gmii_tx_clk(gmii_tx_clk),
.gmii_txen(gmii_txen),
.gmii_txd(gmii_txd)
);

assign gmii_gtxc = eth_rxc;

gmii_to_rgmii gmii_to_rgmii_0(
    .gmii_gtxc(gmii_gtxc),
    .gmii_txd(gmii_txd),
    .gmii_txen(gmii_txen),
    .rgmii_gtxc(rgmii_gtxc),
    .rgmii_txd(rgmii_txd),
    .rgmii_txen(eth_rxdv)
);

initial Clk = 1;
always #10 Clk = ~Clk;
initial eth_rxc = 1;
always #4 eth_rxc = ~eth_rxc;

initial begin
    reset_n = 0;
    tx_en_pulse = 0;
    #201;
    reset_n = 1;
    #2000;
    tx_en_pulse = 1;
    #8;
    tx_en_pulse = 0;
    #40;
    @(posedge tx_done);
    #80000;
    tx_en_pulse = 1;
    #8;
    tx_en_pulse = 0;
    #40;
    tx_en_pulse = 1;
    #8;
    tx_en_pulse = 0;
    #40;

    @(posedge tx_done);
    #80000;
    tx_en_pulse = 1;
    #8;
    tx_en_pulse = 0;
```

```
#40;
$stop;

end

reg [5:0]data_cnt;

reg [39:0]CMD0,CMD1,CMD2,CMD3;
// reg [39:0]CMD0,CMD1,CMD2;
// reg [39:0]CMD0,CMD1;
// reg [39:0]CMD0;

initial begin
    CMD0 = 40'h01_00_00_00_01; //设置通道
    CMD1 = 40'h02_00_00_00_30; //设置采集数据个数 ( )
    CMD2 = 40'h03_00_00_03_E7; //设置采集塑料为50KHz (999)
    CMD3 = 40'h00_00_00_00_01; //启动发送
end

always@(posedge eth_rxc or negedge reset_n)
if(!reset_n)
    data_cnt <= 0;
else begin
    if(payload_req)
        data_cnt <= data_cnt + 1'd1;
    else
        data_cnt <= 0;
end

always@(*)
    case(data_cnt)
        0,8,16,24:dout = 8'h55;
        1,9,17,25:dout = 8'hA5;
        7,15,23,31:dout = 8'hF0;

        2:dout = CMD0[39:32]; //地址
        3:dout = CMD0[31:24];
        4:dout = CMD0[23:16];
        5:dout = CMD0[15:8];
        6:dout = CMD0[7:0];

        2+8:dout = CMD1[39:32]; //地址
        3+8:dout = CMD1[31:24];
        4+8:dout = CMD1[23:16];
        5+8:dout = CMD1[15:8];
        6+8:dout = CMD1[7:0];

        2+16:dout = CMD2[39:32]; //地址
        3+16:dout = CMD2[31:24];
        4+16:dout = CMD2[23:16];
        5+16:dout = CMD2[15:8];
        6+16:dout = CMD2[7:0];

        2+24:dout = CMD3[39:32];
```

```
3+24:dout = CMD3[31:24];  
4+24:dout = CMD3[23:16];  
5+24:dout = CMD3[15:8];  
6+24:dout = CMD3[7:0];  
default:dout = 0;  
endcase  
  
endmodule
```

在对 ADC128S102 控制器驱动模块进行介绍时我们说到可以利用计数器的方式代替 ADC128S102 进行仿真验证，并设计了计数模块，读者在我们提供的工程顶层中用计数模块代替 ADC128S102 模块进行例化就可以进行整个系统的仿真验证。在仿真中可以观察到，配置的指令数据被以太网模块接收，解析后得到采集配置参数数据，根据指令参数数据得到模拟采样的计数器数据。

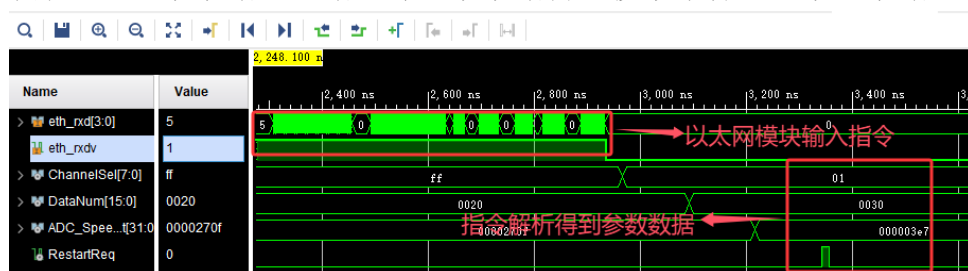


图 43 系统仿真

1.3 基于 FPGA 开发板的系统测试

在上一节，我们通过仿真的方式，模拟产生以太网命令帧和 ADC 采样数据，验证了整个系统的功能。经过验证，证明了系统能够接收并正确的解析指令，并按照指令要求采集和发送相应的数据。下面我们对设计进行基于开发板的系统测试，以验证设计在硬件环境下的功能。

1.3.1 系统所需硬件

- 1、实验开发板。
- 2、下载器。
- 3、电源适配器。
- 4、网线。
- 5、信号发生器（可接开发板上 3.3V 电源对信号进行观察验证）。

1.3.2 硬件连接

要验证本次实验，首先第一步就是对本次实验的整个系统硬件进行连接，这里我们用信号发生器，分别输出 200Hz 的正弦波、方波、三角波，幅值为：高电平：3.3V，低电平：0V，用户可以直接使用信号发生器作为信号源。网线连接好开发板以太网接口和电脑的以太网接口，信号发生器输出连接 ADC128S102 模块 A0 端口，需要注意的是开发板上的 ADC128S102 的采样电压范围为 0~3.3V，所以信号发生器输出的信号电压不要超过这个范围。

(1) ADC128S102 模块通道连接口。



图 44 ACX720 上 ADC128S102 通道对应图

整体的硬件连接如图中所示，在硬件连接好后打开开关就可以下载程序。



图 45 ACX720 硬件连接图

1.3.3 网络连接设置

硬件连接好之后我们就可以开始对以太网的连接设置。网络连接设置请参考芯路恒官方论坛以太网网络连接设置方法，链接如下<http://www.corecourse.cn/forum.php?mod=viewthread&tid=28645>。更多问题可上www.corecourse.cn 搜索“以太网实验常见问题”关键词搜索相关帖子。

1.3.4 指令的下发设置与数据接收

程序下载完成后开发板上会有两个红灯亮起，接着便可以对 ADC128S102 进行配置了，首先打开网络调试助手（NetAssist.exe）并按照如下所述设置各项参数。

- (1) 选择协议类型为 UDP。
- (2) 设置本地 IP 地址为 192.168.0.3。

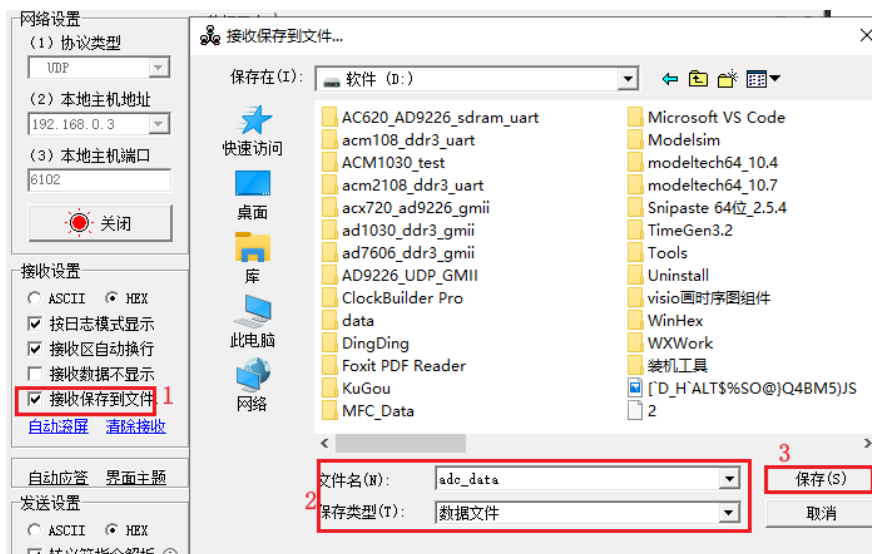
店铺: <https://xiaomeige.taobao.com>

官方网站: www.corecourse.cn

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

技术群组:

- (3) 设置本地端口号为 6102。
- (4) 点击【连接】按钮以创建连接，连接上后该按钮为红色“关闭”字样。
- (5) 连接上后，设置目标主机为 192.168.0.2，目标端口为 5000。
- (6) 点击“接收保存到文件”这几个字，在弹出的界面中设置文件路径、文件名称，如下图所示。这样在数据接收完成之后会保存一个数据文件。方便后面进行分析。



- (7) 发送设置中设置为 HEX 格式

- (8) 在文本框中输入希望发送的文本内容，网络调试助手最终配置界面如下图所示。



图 46 网口调试窗口

以上面接收转命令模块中介绍到的数据帧格式对 ADC128S102 的四个寄存器进行配置。四个寄存器对应的指令帧代码如下：

DataNum: 55 A5 02 00 00 40 00 F0

ChannelSel: 55 A5 01 00 00 00 01 F0

ADC_Speed_Set: 55 A5 03 00 00 00 F9 F0（这条命令可删除，无实际意义，程序中默认设置每个通道最高采样速率 125K）

RestartReq: 55 A5 00 00 00 00 00 F0

配置成网络调试助手发送的数据格式为：

55A50200004000F055A50100000001F055A503000000F9F055A50000000000F0

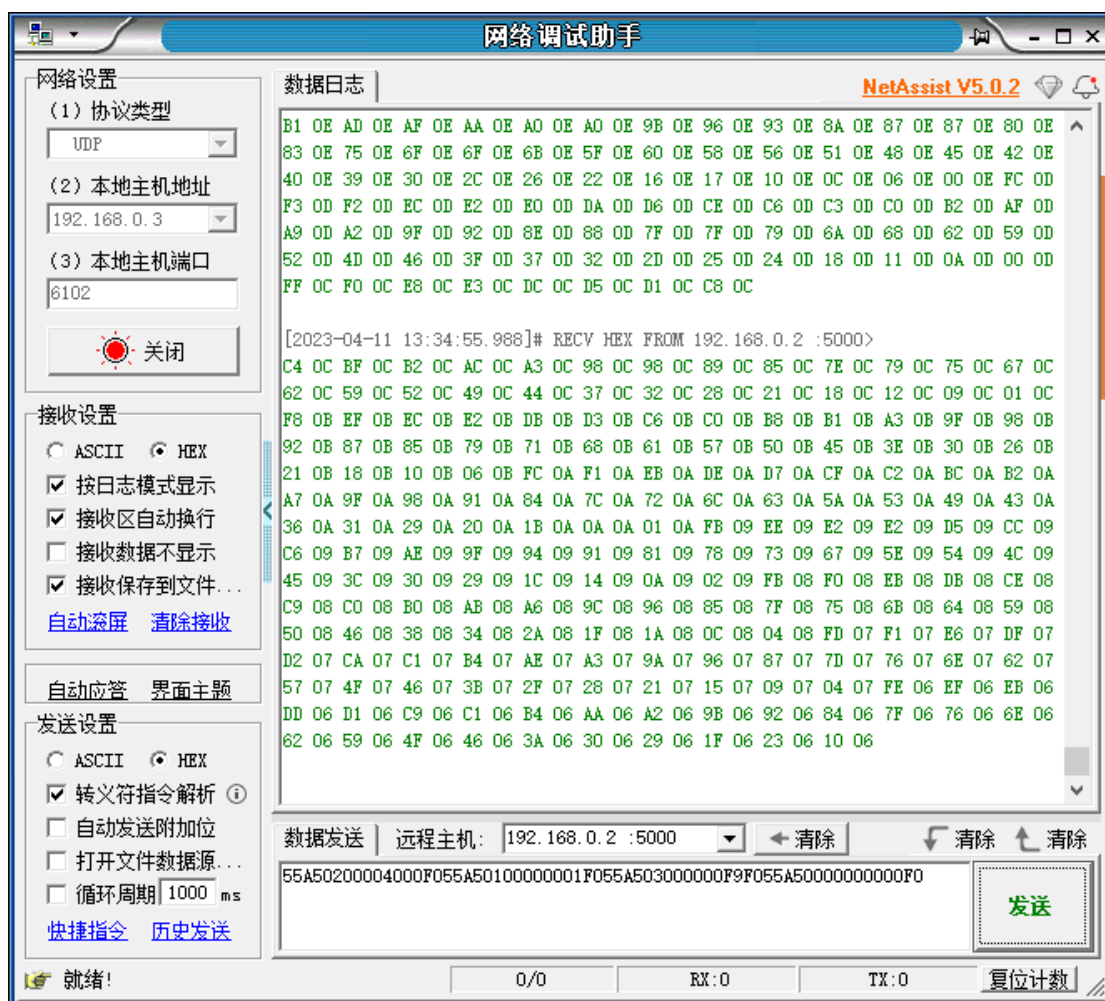


图 47 网口调试窗口接收数据

共采集 16384 个数据（16 位），通过观察接收数据量为 32768（8 位），与预期一致，也可以设置不同的采样数据量来简单验证设计是否达到预期效果。比如采集一个数据观察网络调试助手收到的数据量是否正确。

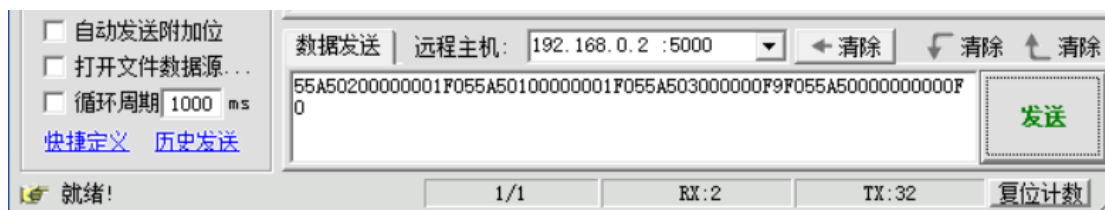


图 48 指令设置

1.3.5 运用 Wireshark 进行数据抓包

同时也可以安装网络抓包工具 Wireshark，我们在实验的时候可以用这工具

店铺: <https://xiaomeige.taobao.com>

官方网站: www.corecourse.cn

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

技术群组:

来查看 PC 网口发送的数据和接收到的数据。打开安装好的 wireshark 抓包工具。在软件界面选择您 PC 的有线网卡，按开始按钮开始抓包。（不同的版本界面有所差异，我们提供的是 Wireshark_win64_V1.12.4_setup.1427187922.exe）。

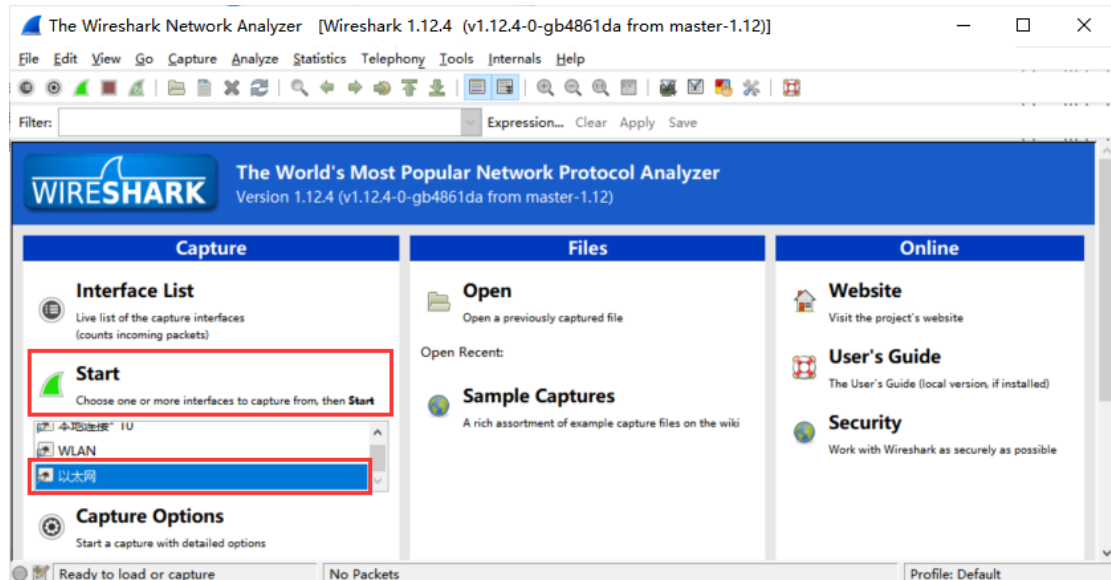
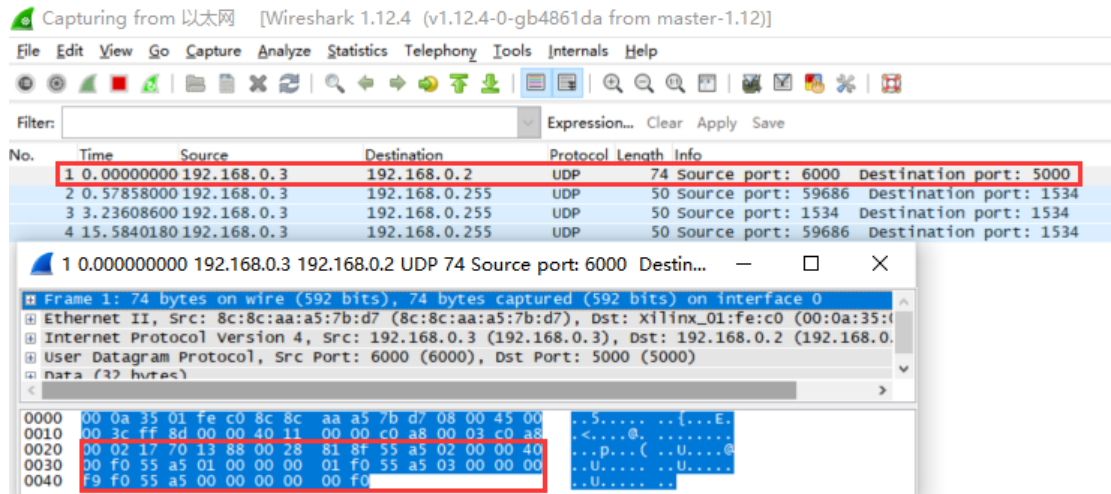


图 49 抓包工具 Wireshark

从 wireshark 抓到的数据包可以到与网络调试助手发送的数据一致。



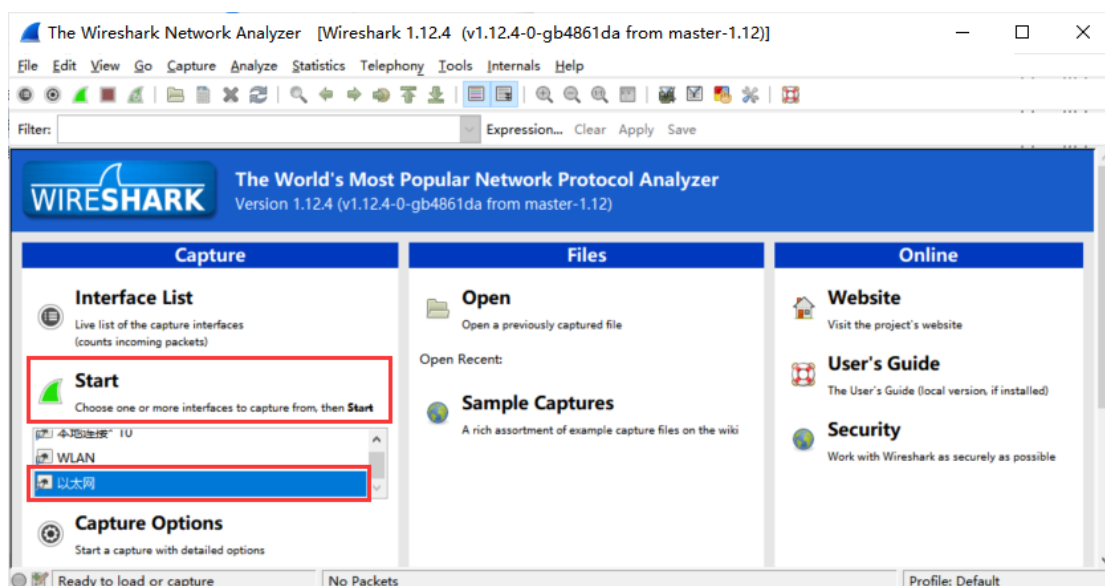


图 50 wireshark 抓到的数据包

同时也可以对采集到的数据进行查看

Capturing from 以太网 [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)]

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.0.3	192.168.0.255	UDP	50	Source port: 1534 Destination port: 1534
2	3.69413800	192.168.0.3	192.168.0.2	UDP	74	Source port: 6000 Destination port: 5000
3	3.69870800	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
4	3.70240000	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
5	3.70606800	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
6	3.70975100	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
7	3.71359600	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
8	3.71710100	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
9	3.72078100	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
10	3.72446400	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
11	3.72814600	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
12	3.73182100	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
13	3.73554100	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
14	3.73919300	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
15	3.74286500	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
16	3.74654200	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
17	3.75022400	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
18	3.75390200	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
19	3.75758200	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
20	3.76191000	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
21	3.76545800	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
22	3.76862400	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
23	3.77230000	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
24	3.77598600	192.168.0.2	192.168.0.3	UDP	1514	Source port: 5000 Destination port: 6000
25	3.77692900	192.168.0.2	192.168.0.3	UDP	426	Source port: 5000 Destination port: 6000

图 51 数据进行查看 1

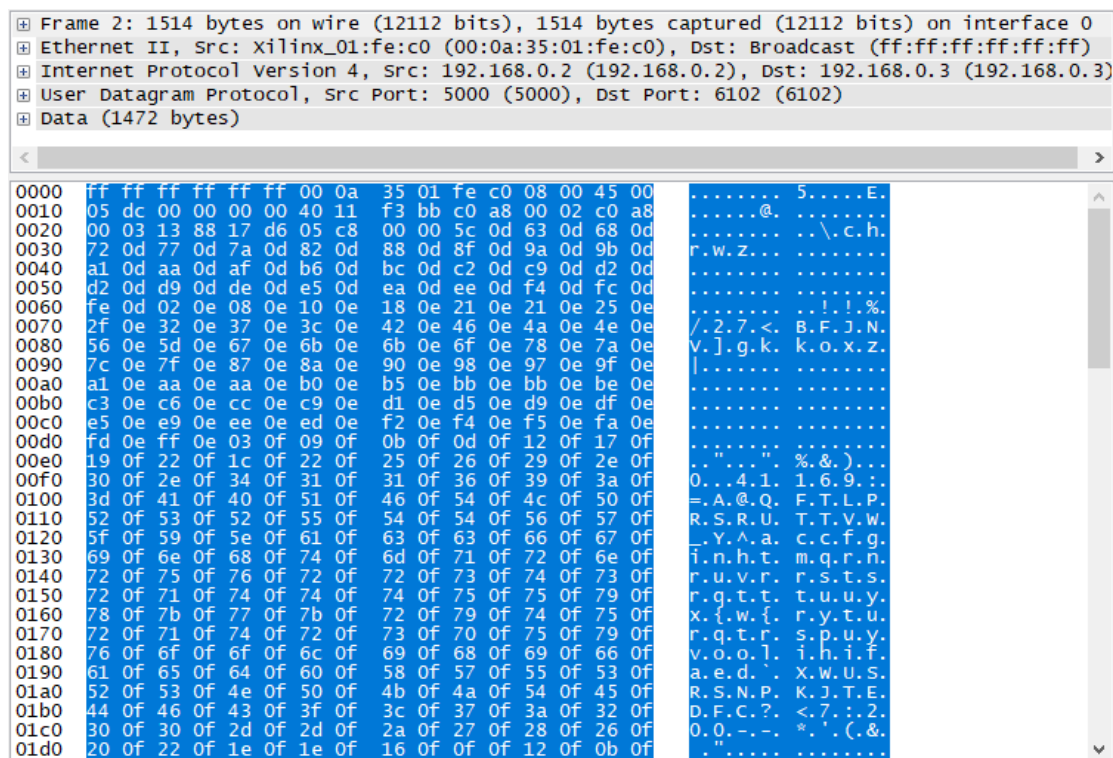


图 52 数据进行查看 2

1.3.6 数据采集上位机通信

前面通过网络调试助手采集数据时，每次保存数据都需要重新点击“接收保存文件”一栏，修改寄存器参数的时候，都需要重新计算，然后发送命令，修改之后也不能直接实时观察到数据波形，使用起来不是很方便。基于上述问题，我们设计了上位机软件“小梅哥控制台 For ADC 采集”进行数据采集，上位机内部直接对命令进行了构建，用户只需要在界面上对采样参数进行设置，便可以实时观测到数据变化，该软件读者可以在 ACX720 开发板资料包的盘 A/05_驱动和软件/小软件 中找到。双击上位机软件，初始界面如下所示。

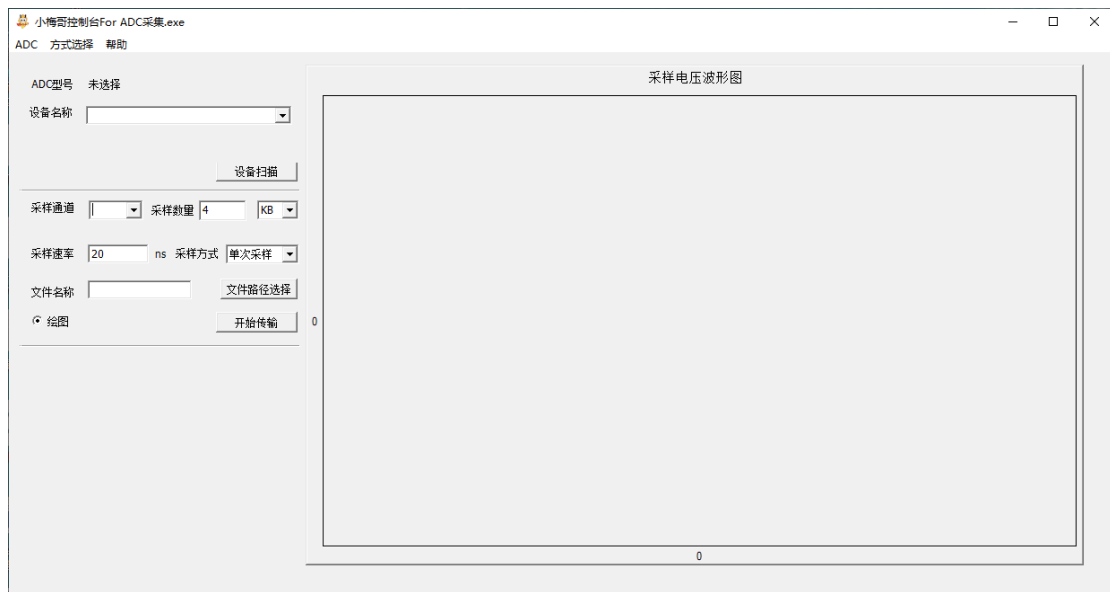
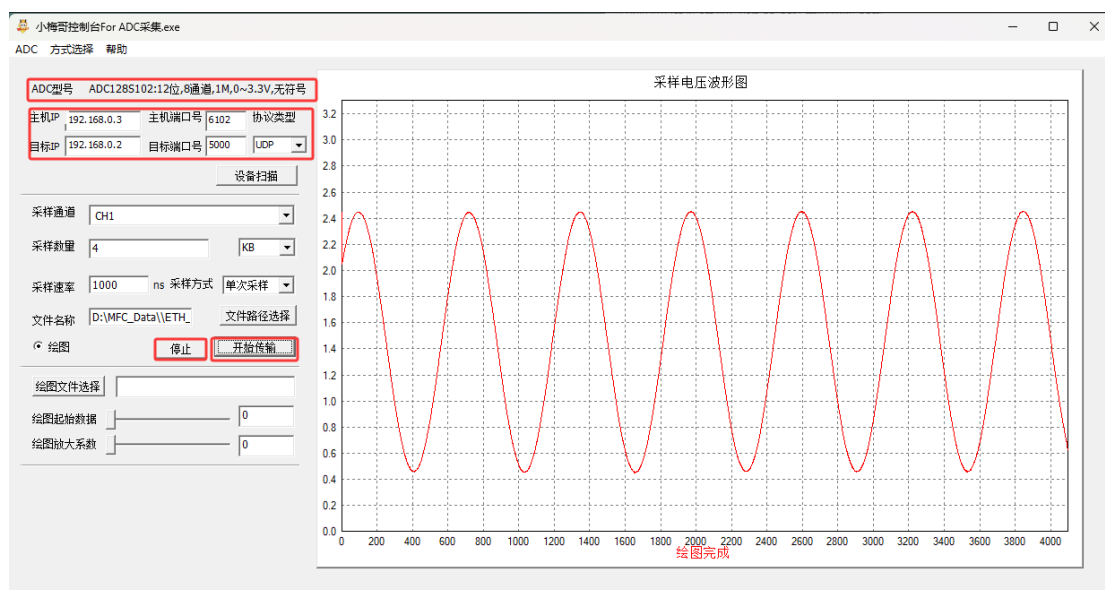


图 53 上位机软件初始界面显示

本次实验使用该软件的方式如下所示：

1. 点击 ADC，选择 ADC128S102。
2. 点击方式，选择网口，可以看到主机 IP（PC 端）和目的 IP（FPGA）以及对应的端口号。主机 IP：192.168.0.2，主机端口号：6102；目的 IP：192.168.0.3，目的端口号：5000。
3. 选择完成之后，我们可以看到采样通道、采样数量等都已经设置了初始值（默认设置的采样率为 ADC 模块的最大采样率，本次实验采样速率不可修改，所以该选项无意义），用户可以根据自己的需求进行修改。
4. 点击网络连接。
5. 点击开始传输之后，可以看到在右边采样电压波形图界面可以直观看得到波形图，如下所示。需要注意的是波形图的横坐标对应的不是频率，而是采样数量。



通过上位机采集到的数据文件保存在 D:/ MFC_Data 文件夹下，后续可以通过 MATLAB 软件进行进一步的分析，这里就不再进行说明，请读者参看前一节的内容进行分析。

1.4 总结

本次实验介绍了基于 FPGA 数据采集系统基本设计思路，并进行了基于 ADC128S102 的千兆以太网收发系统设计。用户通过网口调试助手向开发板发送指令数据配置 ADC128S102 驱动四个寄存器获取 ADC 采样数据，并将采集的数据组成以太网帧，经由 RGMII 网口发送至电脑，借由网口调试工具对数据进行查看。

本次实验利用以太网 RGMII 进行数据通信，对 GMII 和 RGMII 进行了转换，我们需要使用 xilinx 的 ODDR (Output Double Data Rate, 输出双倍数据速率) 原语，将该接口使用 OLOGIC 块实现。同时负责启动 ADC 的 0 号寄存器应该放在代码指令的最后进行配置。本次实验涉及时钟域的转换，以及采集数据位宽的转换，完成这些转换需要对 FIFO 进行合理的配置。

在设计的过程中建立起仿真便于我们在设计中进行纠错和验证，读者也可在本次系统设计进行验证的过程中通过仿真去观察系统数据的变换过程，更直观的了解系统。读者在进行系统设计时也可按照本次设计的思路建立起合适的仿真，协助系统设计。

小梅哥 FPGA 团队

武汉芯路恒科技

专注于培养您的 FPGA 独立开发能力 开发板 培训 项目研发三位一体

本次实验也要求读者对以太网 UDP 协议有足够的认识，可以先学习小梅哥以太网件设计和开发相关教程，获取资料搜索芯路恒官方论坛 www.corecourse.cn。

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:

小梅哥 FPGA 团队

武汉芯路恒科技

专注于培养您的 FPGA 独立开发能力

开发板 培训 项目研发三位一体

修订记录

V1.0	2021/011/25	首次发布

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: