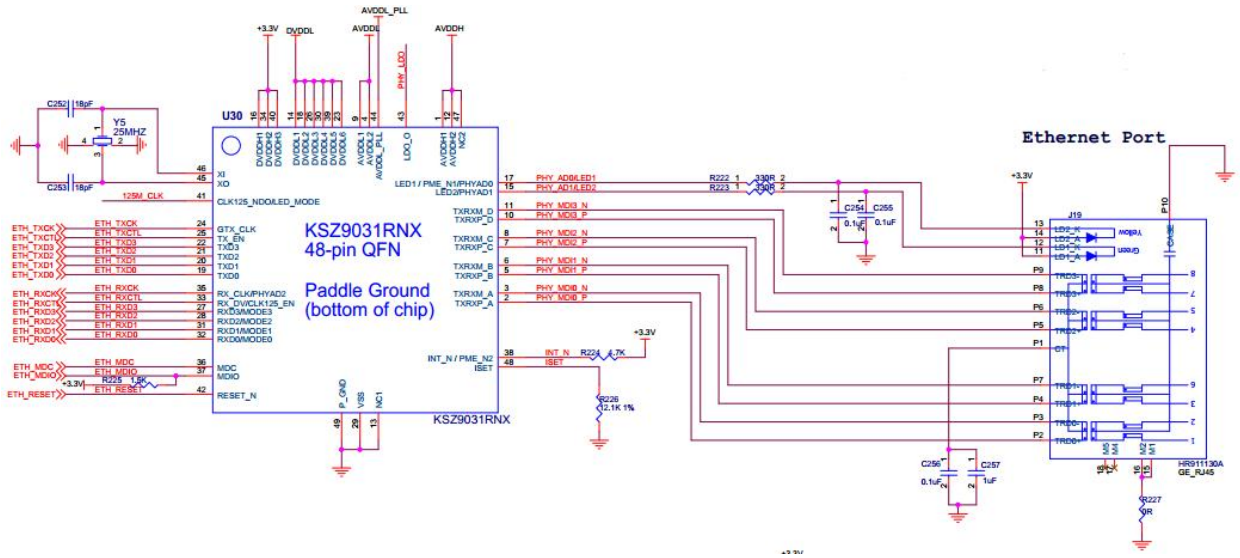
千兆以太网传输实验

1 简介

本实验将实现 FPGA 芯片和 PC 之间进行千兆以太网数据通信, 通信协议采用 Ethernet UDP 通信协议。 FPGA 通过 RGMII 总线和开发板上的 Gigabit PHY 芯片通信, Gigabit PHY 芯片把数据通过网线发给 PC，程序中实现了 ARP，UDP，PING 功能。

2 硬件介绍

在 AXP100 开发板上通过 KSZ9031RNX 以太网 PHY 芯片为用户提供网络通信服务。以太网 PHY 芯片是连接到 Logos2 FPGA 的 IO 接口上。KSZ9031RNX 芯片支持 10/100/1000 Mbps 网络传输速率，通过 RGMII 接口跟 FPGA 进行数据通信。KSZ9031RNX 支持 MDI/MDX 自适应，各种速度自适应，Master/Slave 自适应，支持 MDIO 总线进行 PHY 的寄存器管理。当网口 Link 到千兆以太网时, FPGA 通过 RGMII 总线和 PHY 芯片进行数据通信，当网口 Link 到百兆以太网时, PGA 通过 MII 总线和 PHY 芯片进行数据通信。另外 FPGA 可以通过 MDI/MDIO 管理接口来配置或读取 PHY 芯片内部的寄存器。在千兆的 RGMII 通信模式下, 发送数据时,发送时钟为 125Mhz 的 ETH_TXCK 信号, 数据为 ETH_TXD[3:0], 数据有效信号为 ETH_TXCTL; 接收数据时,接收时钟为 125Mhz 的 ETH_RXCK 信号, 数据为 ETH_RXD[3:0], 数据有效信号为 ETH_RXCTL。关于详细的管脚说明和 RGMII/MII 的通信时序，请大家参考 KSZ9031RNX 的芯片手册。



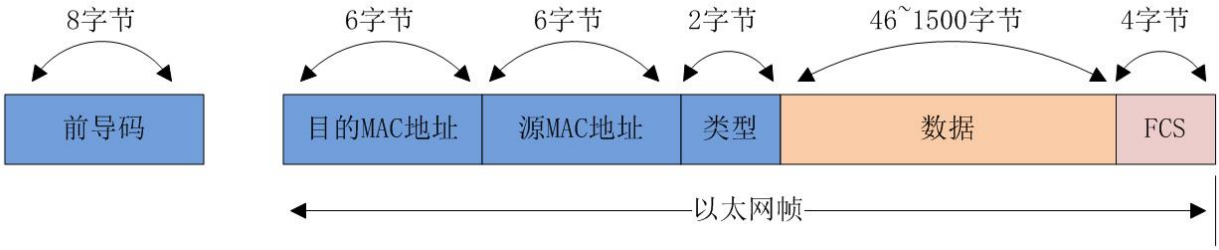
开发板网络部分电路

关于详细的管脚说明和 RGMII 的通信时序，请大家参考相关芯片手册。

3 以太网帧

3.1 以太网帧格式

下图为以太网的帧格式：



前导码（Preamble）：8 字节，连续 7 个 8'h55 加 1 个 8'hd5，表示一个帧的开始，用于双方设备数据的同步。

目的 MAC 地址：6 字节，存放目的设备的物理地址，即 MAC 地址

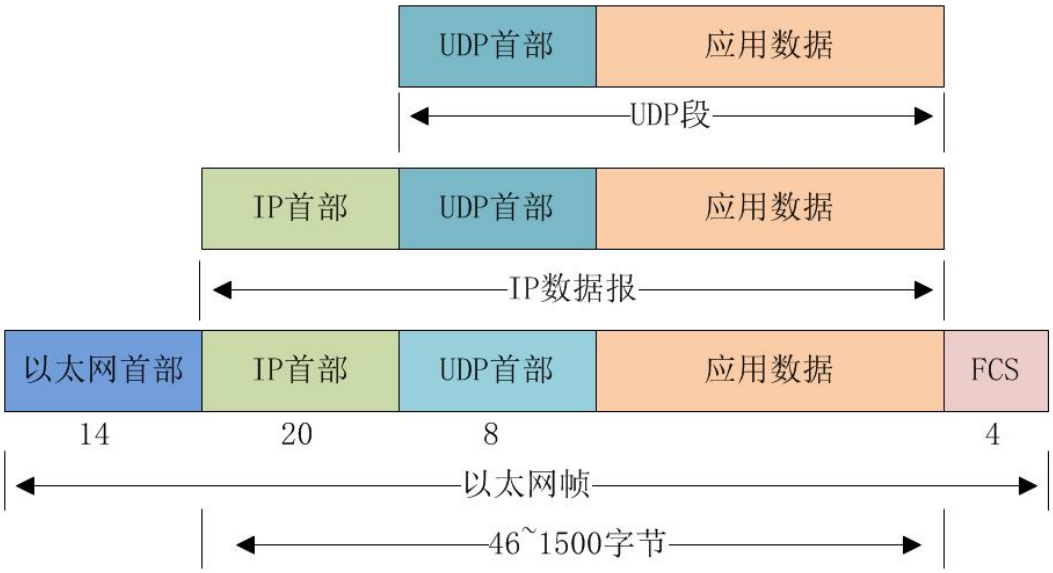
源 MAC 地址：6 字节，存放发送端设备的物理地址

类型：2 字节，用于指定协议类型，常用的有 0800 表示 IP 协议，0806 表示 ARP 协议，8035 表示 RARP 协议

数据：46 到 1500 字节，最少 46 字节，不足需要补全 46 字节，例如 IP 协议层就包含在数据部分，包括其 IP 头及数据。

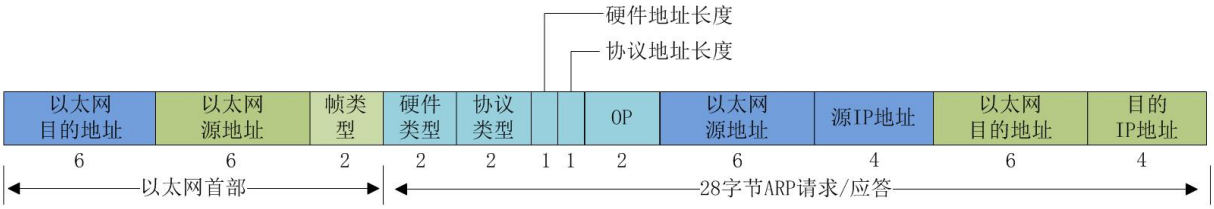
FCS：帧尾，4 字节，称为帧校验序列，采用 32 位 CRC 校验，对目的 MAC 地址字段到数据字段进行校验。

进一步扩展，以 UDP 协议为例，可以看到其结构如下，除了以太网首部的 14 字节，数据部分包含 IP 首部，UDP 首部，应用数据共 46~1500 字节。



3.2 ARP 数据报格式

ARP 地址解析协议，即 ARP（Address Resolution Protocol），根据 IP 地址获取物理地址。主机发送包含目的 IP 地址的 ARP 请求广播（MAC 地址为 48'hff_ff_ff_ff_ff_ff）到网络上的主机，并接收返回消息，以此确定目标的物理地址，收到返回消息后将 IP 地址和物理地址保存到缓存中，并保留一段时间，下次请求时直接查询 ARP 缓存以节约资源。下图为 ARP 数据报格式 。



帧类型：ARP 帧类型为两字节 0806

硬件类型：指链路层网络类型，1 为以太网

协议类型：指要转换的地址类型，采用 0x0800 IP 类型，之后的硬件地址长度和协议地址长度分别对应 6 和 4

OP 字段中 1 表示 ARP 请求，2 表示 ARP 应答

例如：|ff ff ff ff ff ff|00 0a 35 01 fe c0|08 06|00 01|08 00|06|04|00 01|00 0a 35 01 fe c0

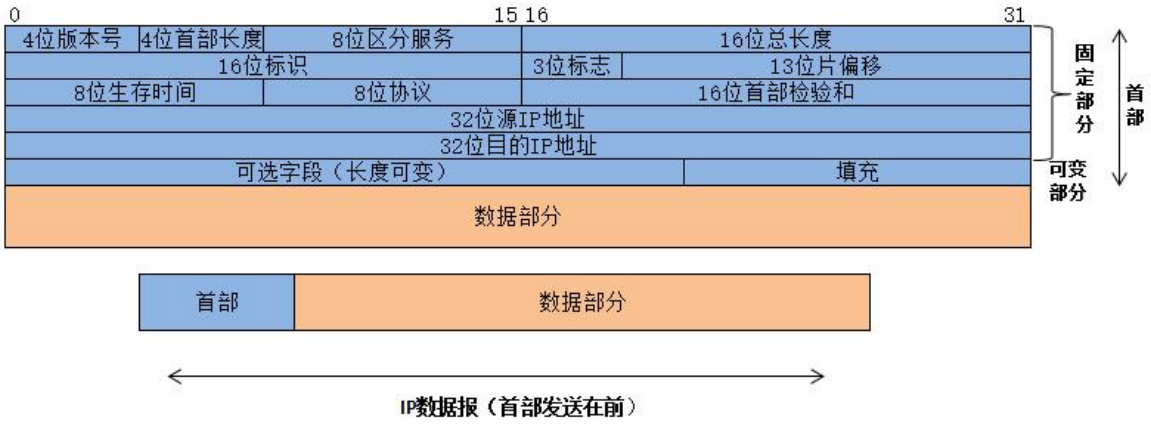
|c0 a8 00 02| ff ff ff ff ff ff|c0 a8 00 03| 表示向 192.168.0.3 地址发送 ARP 请求。

|00 0a 35 01 fe c0 | 60 ab c1 a2 d5 15 |08 06|00 01|08 00|06|04|00 02| 60 ab c1 a2 d5 15

|c0 a8 00 03|00 0a 35 01 fe c0|c0 a8 00 02|表示向 192.168.0.2 地址发送 ARP 应答。

3.3 IP 数据报格式

因为 UDP 协议包只是 IP 包中的一种，所以我们来介绍一下 IP 包的数据格式。下图为 IP 分组的报文头格式，报文头的前 20 个字节是固定的，后面的可变



版本:占 4 位,指 IP 协议的版本目前的 IP 协议版本号为 4 (即 IPv4)

首部长度:占 4 位,可表示的最大数值是 15 个单位(一个单位为 4 字节)因此 IP 的首部长度的最大值是 60 字节

区分服务:占 8 位,用来获得更好的服务,在旧标准中叫做服务类型,但实际上一直未被使用过.1998 年这个字段改名为区分服务.只有在使用区分服务(DiffServ)时,这个字段才起作用.一般的情况下都不使用这个字段

总长度:占 16 位,指首部和数据之和的长度,单位为字节,因此数据报的最大长度为 65535 字节.总长度必须不超过最大传送单元 MTU

标识:占 16 位,它是一个计数器,用来产生数据报的标识

标志(flag):

占 3 位,目前只有前两位有意义

MF

标志字段的最低位是 MF (More Fragment)

MF=1 表示后面“还有分片”。MF=0 表示最后一个分片

DF

标志字段中间的一位是 DF (Don't Fragment)

只有当 DF=0 时才允许分片

片偏移:占 12 位,指较长的分组在分片后某片在原分组中的相对位置.片偏移以 8 个字节为偏移单位

生存时间:占 8 位,记为 TTL (Time To Live) 数据报在网络中可通过的路由器数的最大值,TTL 字段是由发送端初始设置一个 8 bit 字段.推荐的初始值由分配数字 RFC 指定,当前值为 64.发送 ICMP 回显应答时经常把 TTL 设为最大值 255

协议:占 8 位,指出此数据报携带的数据使用何种协议以便目的主机的 IP 层将数据部分上交给哪个处理过程, 1 表示为 ICMP 协议, 2 表示为 IGMP 协议, 6 表示为 TCP 协议, 17 表示为 UDP 协议

首部检验和:占 16 位,只检验数据报的首部不检验数据部分，采用二进制反码求和，即将 16 位数据相加后，再将进位与低 16 位相加，直到进位为 0，最后将 16 位取反。

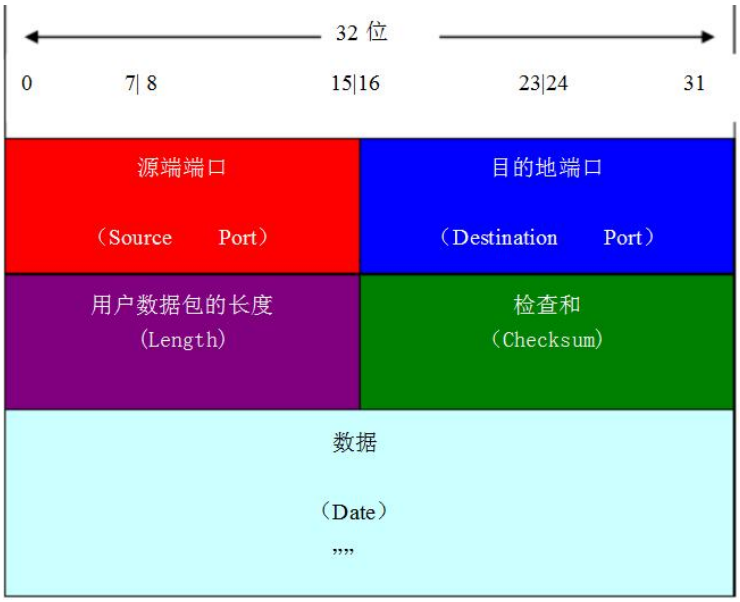
源地址和目的地址:都各占 4 字节,分别记录源地址和目的地址

3.4 UDP 协议

UDP 是 User Datagram Protocol（用户数据报协议）的英文缩写。UDP 只提供一种基本的、低延迟的被称为数据报的通讯。所谓数据报，就是一种自带寻址信息，从发送端走到接收端的数据包。UDP 协议经常用于图像传输、网络监控数据交换等数据传输速度要求比较高的场合。

UDP 协议的报头格式:

UDP 报头由 4 个域组成，其中每个域各占用 2 个字节，具体如下:



① UDP 源端口号

② 目标端口号

③ 数据报长度

④ 校验和

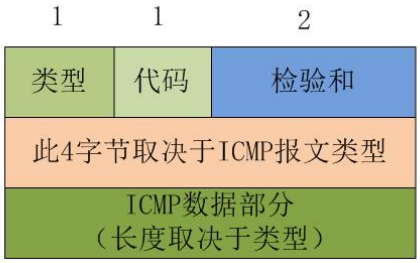
UDP 协议使用端口号为不同的应用保留其各自的数据传输通道。数据发送一方将 UDP 数据报通过源端口发送出去，而数据接收一方则通过目标端口接收数据。

数据报的长度是指包括报头和数据部分在内的总字节数。因为报头的长度是固定的，所以该域主要被用来计算可变长度的数据部分（又称为数据负载）。数据报的最大长度根据操作环境的不同而各异。从理论上说，包含报头在内的数据报的最大长度为 65535 字节。不过，一些实际应用往往会限制数据报的大小，有时会降低到 8192 字节。

UDP 协议使用报头中的校验值来保证数据的安全。校验值首先在数据发送方通过特殊的算法计算得出，在传递到接收方之后，还需要再重新计算。如果某个数据报在传输过程中被第三方篡改或者由于线路噪音等原因受到损坏，发送和接收方的校验计算值将不会相符，由此 UDP 协议可以检测是否出错。虽然 UDP 提供有错误检测，但检测到错误时，错误校正，只是简单地把损坏的消息段扔掉，或者给应用程序提供警告信息。

3.5 Ping 功能

ICMP 是 TCP/IP 协议族的一个 IP 层子协议，包含在 IP 数据报里，用于 IP 主机、路由器之间传递控制消息。控制消息是指网络是否连通，主机是否可达等功能。其中 ping 功能采用回送请求和回答报文，回送请求报文类型为 8’h08，回答报文类型为 8’h00。

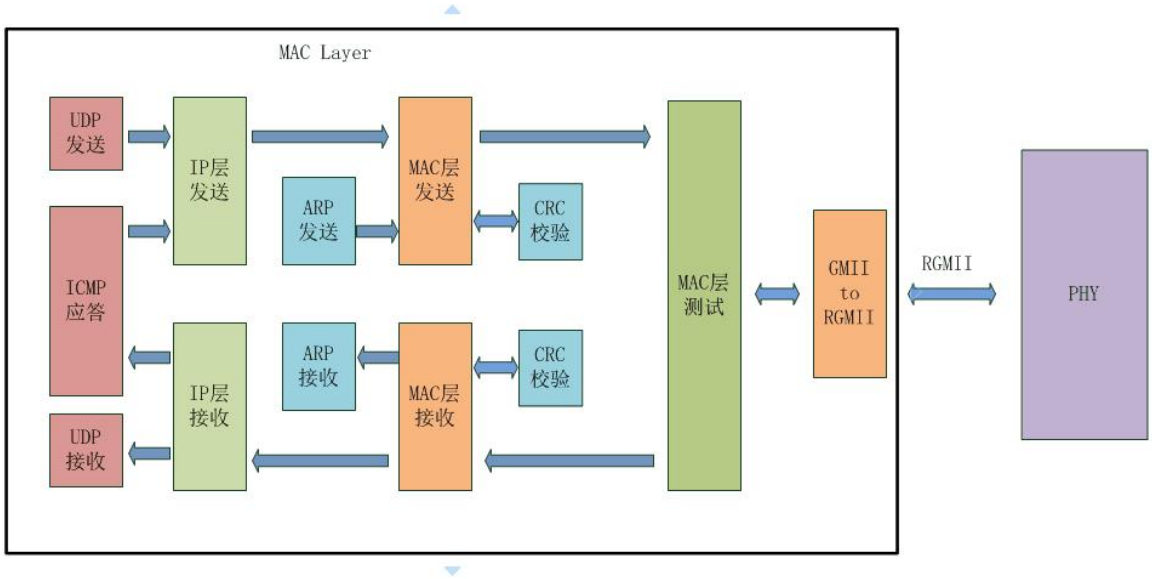


类型值（十进制）	ICMP报文类型
0	回送应答
3	终点不可达
4	源点抑制
5	改变路由
8	回送请求
11	时间超时

4 程序设计

本实验以千兆以太网 RGMII 通信为例来设计 verilog 程序，会先发送预设的 UDP 数据到网络，每秒钟发送一次，如果 FPGA 检测网口发来的 UDP 的数据包，会把接收到的数据包存储在 FPGA

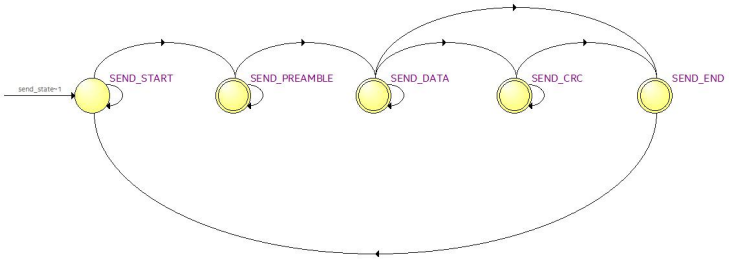
内部的 RAM 中，再不断的把 RAM 中的数据包通过网口发回到 ethernet 网络。程序分为两部分，分别为发送和接收，实现了 ARP，UDP，Ping 功能。以下为原理实现框图：



4.1 发送部分

4.1.1 MAC 层发送

发送部分中， mac_tx.v 为 MAC 层发送模块，首先在 SEND_START 状态，等待 mac_tx_ready 信号，如果有效，表明 IP 或 ARP 的数据已经准备好，可以开始发送。再进入发送前导码状态，结束时发送 mac_data_req，请求 IP 或 ARP 的数据，之后进入发送数据状态，最后进入发送 CRC 状态。在发送数据过程中，需要同时进行 CRC 校验。



信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟

rst_n	in	1	异步复位，低电平复位
crc_result	in	32	CRC32 结果
crcen	out	1	CRC 使能信号
crcre	out	1	CRC 复位信号
crc_din	out	8	CRC 模块输入信号
mac_frame_data	in	8	从 IP 或 ARP 来的数据
mac_tx_req	in	1	MAC 的发送请求
mac_tx_ready	in	1	IP 或 ARP 数据已准备好
mac_tx_end	in	1	IP 或 ARP 数据已经传输完毕
mac_tx_data	out	8	向 PHY 发送数据
mac_send_end	out	1	MAC 数据发送结束
mac_data_valid	out	1	MAC 数据有效信号，即 gmii_tx_en
mac_data_req	out	1	MAC 层向 IP 或 ARP 请求数据

4.1.2 MAC 发送模式

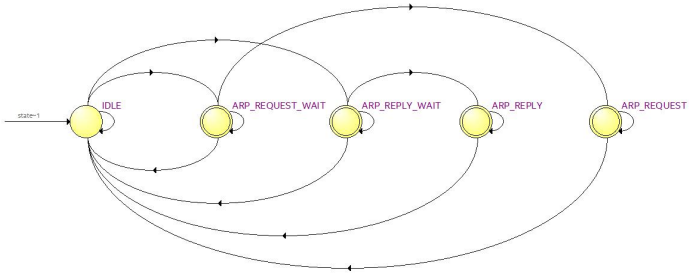
工程中的 mac_tx_mode.v 为发送模式选择，根据发送模式是 IP 或 ARP 选择相应的信号与数据。

信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
mac_send_end	in	1	MAC 发送结束
arp_tx_req	in	1	ARP 发送请求
arp_tx_ready	in	1	ARP 数据已准备好
arp_tx_data	in	8	ARP 数据
arp_tx_end	in	1	ARP 数据发送到 MAC 层结束
arp_tx_ack	in	1	ARP 发送响应信号
ip_tx_req	in	1	IP 发送请求
ip_tx_ready	in	1	IP 数据已准备好
ip_tx_data	in	8	IP 数据

ip_tx_end	in	1	IP 数据发送到 MAC 层结束
mac_tx_ready	out	1	MAC 数据已准备好信号
ip_tx_ack	out	1	IP 发送响应信号
mac_tx_ack	in	1	MAC 发送响应信号
mac_tx_req	out	1	MAC 发送请求
mac_tx_data	out	8	MAC 发送数据
mac_tx_end	out	1	MAC 数据发送结束

4.1.3 ARP 发送

发送部分中，arp_tx.v 为 ARP 发送模块， 在 IDLE 状态下，等待 ARP 发送请求或 ARP 应答请求信号，之后进入请求或应答等待状态，并通知 MAC 层，数据已经准备好，等待 mac_data_req 信号，之后进入请求或应答数据发送状态。由于数据不足 46 字节，需要补全 46 字节发送。

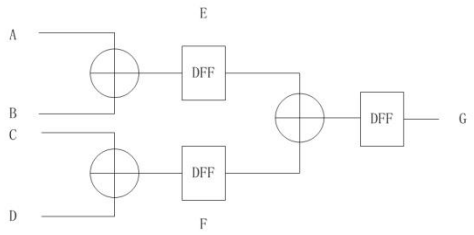


信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
destination_mac_addr	in	48	发送的目的 MAC 地址
source_mac_addr	in	48	发送的源 MAC 地址
source_ip_addr	in	32	发送的源 IP 地址
destination_ip_addr	in	32	发送的目的 IP 地址

mac_data_req	in	1	MAC 层请求数据信号
arp_request_req	in	1	ARP 请求的请求信号
arp_reply_ack	out	1	ARP 回复的应答信号
arp_reply_req	in	1	ARP 回复的请求信号
arp_rec_source_ip_addr	in	32	ARP 接收的源 IP 地址，回复时放到目的 IP 地址
arp_rec_source_mac_addr	in	48	ARP 接收的源 MAC 地址，回复时放到目的 MAC 地址
mac_send_end	in	1	MAC 发送结束
mac_tx_ack	in	1	MAC 发送应答
arp_tx_ready	out	1	ARP 数据准备好
arp_tx_data	out	8	ARP 发送数据
arp_tx_end	out	1	ARP 数据发送结束

4.1.4 IP 层发送

在发送部分，ip_tx.v 为 IP 层发送模块，在 IDLE 状态下，如果 ip_tx_req 有效，也就是 UDP 或 ICMP 发送请求信号，进入等待发送数据长度状态，之后进入产生校验和状态，校验和是将 IP 首部所有数据以 16 位相加，最后将进位再与低 16 位相加，直到进入为 0，再将低 16 位取反，得出校验和结果。此程序中校验和以树型加法结构，并插入流水线，能有效降低加法部分的延迟，但缺点是会消耗较多逻辑资源。如下图 ABCD 四个输入，A 与 B 相加，结果为 E，C 与 D 相加，结果为 F，再将 E 与 F 相加，结果为 G，在每个相加结果之后都有寄存器。



在生成校验和之后，等待 MAC 层数据请求，开始发送数据，并在即将结束发送 IP 首部后请求 UDP 或 ICMP 数据。等发送完，进入 IDLE 状态。

信号名称	方	宽度	说明

	向	(bit)	
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
destination_mac_addr	in	48	发送的目的 MAC 地址
source_mac_addr	in	48	发送的源 MAC 地址
source_ip_addr	in	32	发送的源 IP 地址
destination_ip_addr	in	32	发送的目的 IP 地址
TTL	in	8	生存时间
ip_send_type	in	8	上层协议号，如 UDP,ICMP
upper_layer_data	in	8	从 UDP 或 ICMP 过来的数据
upper_data_req	out	1	向上层请求数据
mac_tx_ack	in	1	MAC 发送应答
mac_send_end	in	1	MAC 发送结束信号
mac_data_req	in	1	MAC 层请求数据信号
upper_tx_ready	in	1	上层 UDP 或 ICMP 数据准备好
ip_tx_req	in	1	发送请求，从上层过来
ip_send_data_length	in	16	发送数据总长度
ip_tx_ack	out		产生 IP 发送应答
ip_tx_busy	out		P 发送忙信号
ip_tx_ready	out	1	IP 数据已准备好
ip_tx_data	out	8	IP 数据
ip_tx_end	out	1	IP 数据发送到 MAC 层结束

4.1.5 IP 发送模式

工程中的 ip_tx_mode.v 为发送模式选择，根据发送模式是 UDP 或 ICMP 选择相应的信号与数据。

信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟

rst_n	in	1	异步复位，低电平复位
mac_send_end	in	1	MAC 数据发送结束
udp_tx_req	in		UDP 发送请求
udp_tx_ready	in	1	UDP 数据准备好
udp_tx_data	in	8	UDP 发送数据
udp_send_data_length	in	16	UDP 发送数据长度
udp_tx_ack	out	1	输出 UDP 发送应答
icmp_tx_req	in	1	ICMP 发送请求
icmp_tx_ready	in	1	ICMP 数据准备好
icmp_tx_data	in	8	ICMP 发送数据
icmp_send_data_length	in	16	ICMP 发送数据长度
icmp_tx_ack	out	1	ICMP 发送应答
ip_tx_ack	in	1	IP 发送应答
ip_tx_req	in	1	IP 发送请求
ip_tx_ready	out	1	IP 数据已准备好
ip_tx_data	out	8	IP 数据
ip_send_type	out	8	上层协议号，如 UDP,ICMP
ip_send_data_length	out	16	发送数据总长度

4.1.6 UDP 发送

发送部分中，udp_tx.v 为 UDP 发送模块，第一步将数据写入 UDP 发送 RAM，同时计算校验和，第二步将 RAM 中数据发送出去。UDP 校验和与 IP 校验和计算方法一致。在计算时需要将伪首部加上，伪首部包括目的 IP 地址，源 IP 地址，网络类型，UDP 数据长度。

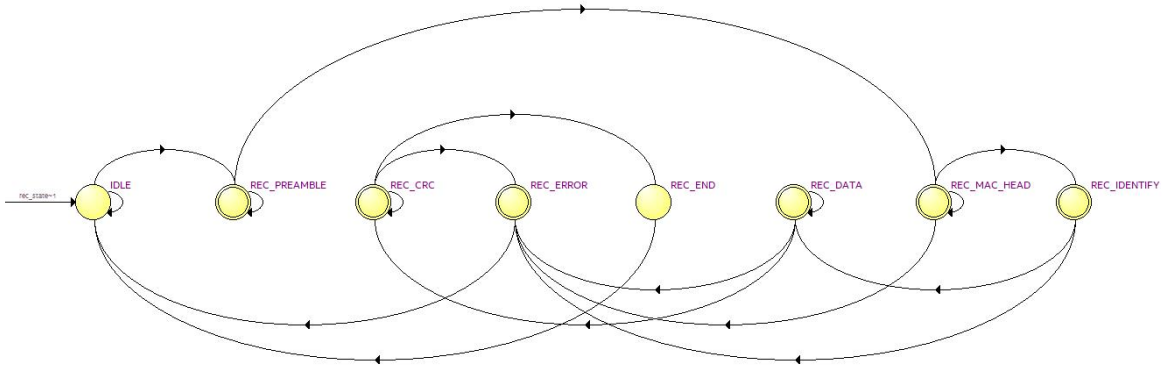
信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
source_ip_addr	in	32	发送的源 IP 地址
destination_ip_addr	in	32	发送的目的 IP 地址
udp_send_source_port	in	16	源端口号

udp_send_destination_port	in	16	目的端口号
udp_send_data_length	in	16	UDP 发送数据长度，用户需给出其值
ram_wr_data	in	8	写 UDP 的 RAM 数据
ram_wr_en	in	1	写 RAM 使能
udp_ram_data_req	out	1	请求写 RAM 数据
mac_send_end	in	1	MAC 发送结束信号
udp_tx_req	in	1	UDP 发送请求
ip_tx_req	out	1	IP 发送请求
ip_tx_ack	in	1	Ip 应答
udp_data_req	in	1	IP 层请求数据
udp_tx_ready	out	1	UDP 数据准备好
udp_tx_data	out	8	UDP 数据
udp_tx_end	out	1	UDP 发送结束（未使用）
almost_full	out	1	Fifo 接近满标志

4.2 接收部分

4.2.1 MAC 层接收

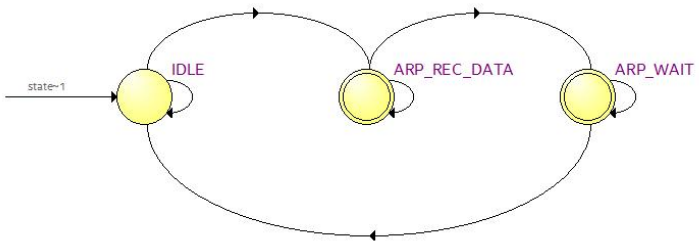
在接收部分，其中 mac_rx.v 为 mac 层接收文件，首先在 IDLE 状态下需要判断 rx_dv 信号是否为高，在 REC_PREAMBLE 前导码状态下，接收前导码。之后进入接收 MAC 头部状态，即目的 MAC 地址，源 MAC 地址，类型，将它们缓存起来，并在此状态判断前导码是否正确，错误则进入 REC_ERROR 错误状态，在 REC_IDENTIFY 状态判断类型是 IP（8'h0800）或 ARP(8'h0806)。然后进入接收数据状态，将数据传送到 IP 或 ARP 模块，等待 IP 或 ARP 数据接收完毕，再接收 CRC 数据。并在接收数据的过程中对接收的数据进行 CRC 处理，将结果与接收到的 CRC 数据进行对比，判断数据是否接收正确，正确则结束，错误则进入 ERROR 状态。



信号名称	方向	宽度(bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
crc_result	in	32	CRC32 结果
crcen	out	1	CRC 使能信号
crcre	out	1	CRC 复位信号
crc_din	out	8	CRC 模块输入信号
rx_dv	in	1	从 PHY 层过来的 rx_dv 信号
mac_rx_datain	in	8	从 PHY 层接收的数据
checksum_err	in	1	IP 层校验和错误
ip_rx_end	in	1	IP 层接收结束
arp_rx_end	in	1	ARP 层接收结束
ip_rx_req	out	1	请求 IP 层接收
arp_rx_req	out	1	请求 ARP 接收
mac_rx_dataout	out	8	MAC 层接收数据输出给 IP 或 ARP
mac_rec_error	out	1	MAC 层接收错误
mac_rx_destination_mac_addr	out	48	MAC 接收的目的 IP 地址
mac_rx_source_mac_addr	out	48	MAC 接收的源 IP 地址

4.2.2 ARP 接收

工程中的 arp_rx.v 为 ARP 接收模块，实现 ARP 数据接收，在 IDLE 状态下，接收到从 MAC 层发来的 arp_rx_req 信号，进入 ARP 接收状态，在此状态下，提取出目的 MAC 地址，源 MAC 地址，目的 IP 地址，源 IP 地址，并判断操作码 OP 是请求还是应答。如果是请求，则判断接收到的目的 IP 地址是否为本机地址，如果是，发送应答请求信号 arp_reply_req，如果不是，则忽略。如果 OP 是应答，则判断接收到的目的 IP 地址及目的 MAC 地址是否与本机一致，如果是，则拉高 arp_found 信号，表明接收到了对方的地址。并将对方的 MAC 地址及 IP 地址存入 ARP 缓存中。



信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
local_ip_addr	in	32	本地 IP 地址
local_mac_addr	in	48	本地 MAC 地址
arp_rx_data	in	8	ARP 接收数据
arp_rx_req	in	1	ARP 接收请求
arp_rx_end	out	1	ARP 接收完成
arp_reply_ack	in	1	ARP 回复应答
arp_reply_req	out	1	ARP 回复请求
arp_rec_source_ip_addr	in	32	ARP 接收的源 IP 地址
arp_rec_source_mac_addr	in	48	ARP 接收的源 MAC 地址
arp_found	out	1	ARP 接收到请求应答正确

4.2.3 IP 层接收模块

在工程中，ip_rx 为 IP 层接收模块，实现 IP 层的数据接收，信息提取，并进行校验和检查。首先在 IDLE 状态下，判断从 MAC 层发过来的 ip_rx_req 信号，进入接收 IP 首部状态，先在 REC_HEADER0 提取出首部长度及 IP 总长度，进入 REC_HEADER1 状态，在此状态提取出目的 IP 地址，源 IP 地址，协议类型，根据协议类型发送 udp_rx_req 或 icmp_rx_req。在接收首部的同时进行校验和的检查，将首部接收的所有数据相加，存入 32 位寄存器，再将高 16 位与低 16 位相加，直到高 16 位为 0 ，再将低 16 位取反，判断其是否为 0，如果是 0，则检验正确，否则错误，进入 IDLE 状态，丢弃此帧数据，等待下次接收。

信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
local_ip_addr	in	32	本地 IP 地址
local_mac_addr	in	48	本地 MAC 地址
ip_rx_data	in	8	从 MAC 层接收的数据
ip_rx_req	in	1	MAC 层发送的 IP 接收请求信号
mac_rx_destination_mac_addr	in	48	MAC 层接收的目的 MAC 地址
udp_rx_req	out	1	UDP 接收请求信号
icmp_rx_req	out	1	ICMP 接收请求信号
ip_addr_check_error	out	1	地址检查错误信号
upper_layer_data_length	out	16	上层协议的数据长度
ip_total_data_length	out	16	数据总长度
net_protocol	out	8	网络协议号
ip_rec_source_addr	out	32	IP 层接收的源 IP 地址
ip_rec_destination_addr	out	32	IP 层接收的目的 IP 地址
ip_rx_end	out	1	IP 层接收结束
ip_checksum_error	out	1	IP 层校验和检查错误信号

4.2.4 UDP 接收

在工程中，udp_rx.v 为 UDP 接收模块，在此模块首先接收 UDP 首部，再接收数据部分，并将数据部分存入 RAM 中，在接收的同时进行 UDP 校验和检查，如果 UDP 数据是奇数个字节，在计算校验和时，在最后一个字节后加上 8'h00，并进行校验和计算。校验方法与 IP 校验和一样，如果校验正确，将拉高 udp_rec_data_valid 信号，表明接收的 UDP 数据有效，否则无效，等待下次接收。

信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
udp_rx_data	in	8	UDP 接收数据
udp_rx_req	in	1	UDP 接收请求
mac_rec_error	in	1	MAC 层接收错误
net_protocol	in	8	网络协议号
ip_rec_source_addr	in	32	IP 层接收的源 IP 地址
ip_rec_destination_addr	in	32	IP 层接收的目的 IP 地址
ip_checksum_error	in	1	IP 层校验和检查错误信号
ip_addr_check_error	in	1	地址检查错误信号
upper_layer_data_length	in	16	上层协议的数据长度
udp_rec_ram_rdata	out	8	UDP 接收 RAM 读数据
udp_rec_ram_read_addr	in	11	UDP 接收 RAM 读地址
udp_rec_data_length	out	16	UDP 接收数据长度
udp_rec_data_valid	out	1	UDP 接收数据有效

4.3 其他部分

4.3.1 ICMP 应答

在工程中，icmp_reply.v 实现 ping 功能，首先接收其他设备发过来的 icmp 数据，判断类型是否是回送请求（ECHO REQUEST），如果是，将数据存入 RAM，并计算校验和，判断校验和是否正确，如果正确则进入发送状态，将数据发送出去。

信号名称	方向	宽度(bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
mac_send_end	in	1	Mac 发送结束信号
ip_tx_ack	in	1	IP 发送应答
icmp_rx_data	in	8	ICMP 接收数据
icmp_rx_req	in	1	ICMP 接收请求
icmp_rev_error	in	1	接收错误信号
upper_layer_data_length	in	16	上层协议长度
icmp_data_req	in	1	发送请求 ICMP 数据
icmp_tx_ready	out	1	ICMP 发送准备好
icmp_tx_data	out	8	ICMP 发送数据
icmp_tx_end	out	1	ICMP 发送结束
icmp_tx_req	out	1	ICMP 发送请求

4.3.2 ARP 缓存

在工程中，arp_cache.v 为 arp 缓存模块，将接收到的其他设备 IP 地址和 MAC 地址缓存，在发送数据之前，查询目的地址是否存在，如果不存在，则向目的地址发送 ARP 请求，等待应答。在设计文件中，只做了一个缓存空间，如果有需要，可扩展。

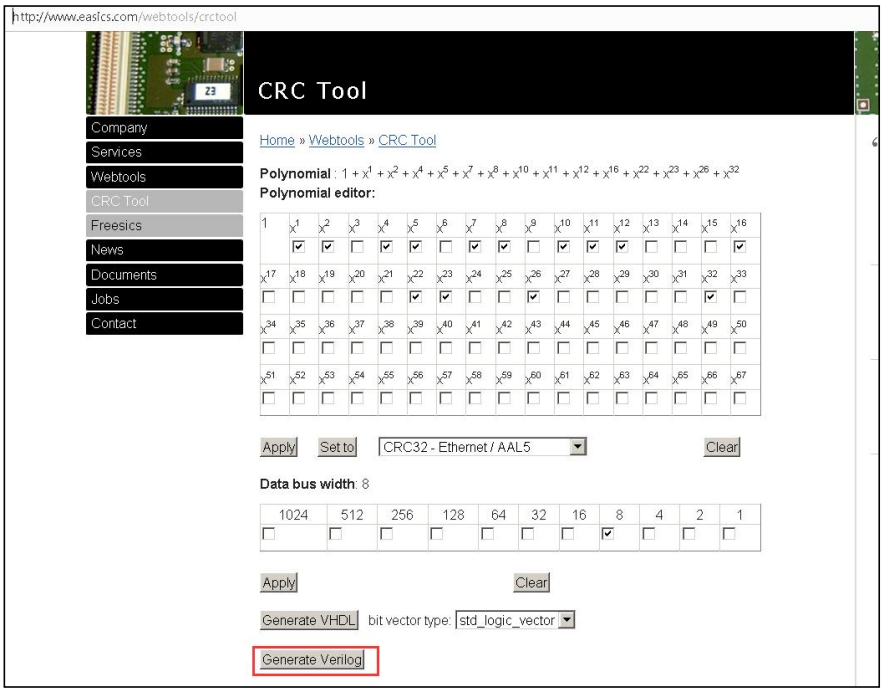
信号名称	方向	宽度(bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
arp_found	in	1	ARP 接收到回复正确
arp_rec_source_ip_addr	in	32	ARP 接收的源 IP 地址
arp_rec_source_mac_addr	in	48	ARP 接收的源 MAC 地址
destination_ip_addr	in	32	目的 IP 地址
destination_mac_addr	out	48	目的 MAC 地址

mac_not_exist	out	1	目的地址对应的 MAC 地址不存在

4.3.3 CRC 校验模块(crc.v)

一个 IP 数据包的 CRC32 校验是在目标 MAC 地址开始计算的，一直计算到一个包的最后一个数据为止。以太网的 CRC32 的 verilog 的算法和多项式可以在以下网站中直接生成：

http://www.easics.com/webtools/crctool



4.3.4 以太网测试模块 （mac_test.v）

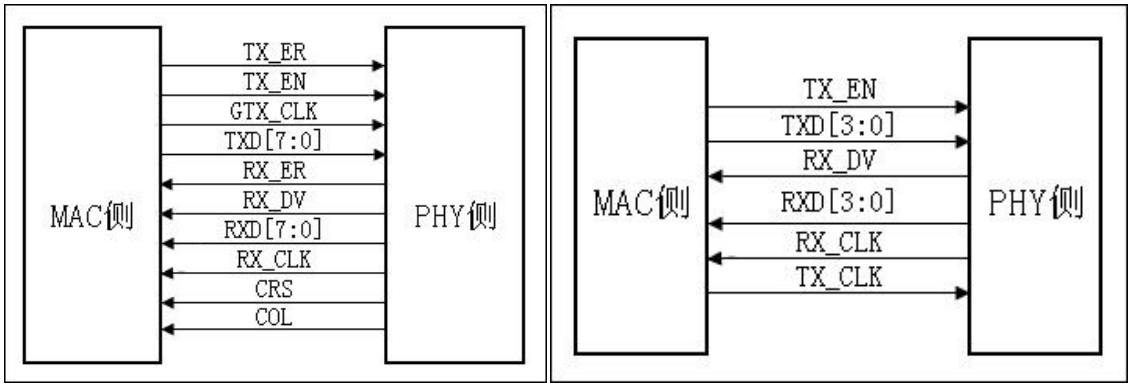
测试模块实现数据流的控制，首先在按下按键后发送 ARP 请求信号，直到对方回应，进入发送 UDP 数据状态，如果在 WAIT 状态时发现 UDP 接收数据有效，则将接收的数据发送出去。循环 1 秒发送，在每次发送前需要检查目的 IP 地址是否能在 ARP 缓存里找到，如果没有，则发送 ARP 请求。

信号名称	方向	宽度 (bit)	说明
clk_50m	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
pack_total_len	in	32	包长度

push_button	in	1	按键信号
gmii_tx_clk	in	1	GMII 发送时钟
gmii_rx_clk	in	1	GMII 接收时钟
gmii_rx_dv	in	1	接收数据有效信号
gmii_rxd	in	8	接收数据
gmii_tx_en	out	1	发送数据有效信号
gmii_txd	out	8	发送数据

4.3.5 RGMII 转 GMII 模块

util_gmii_to_rgmii.v 文件是将 RGMII 与 GMII 转换，提取出控制信号与数据信号。与 PHY 连接是 RGMII 接口。



GMII 接口　　RGMII 接口

RGMII 接口是 GMII 接口的简化版，在时钟的上升沿及下降沿都采样数据，上升沿发送 TXD[3:0]/RXD[3:0]，下降沿发送 TXD[7:4]/RXD[7:4]，TX_EN 传送 TX_EN（上升沿）和 TX_ER（下降沿）两种信息，RX_DV 传送 RX_DV（上升沿）和 RX_ER（下降沿）两种信息。

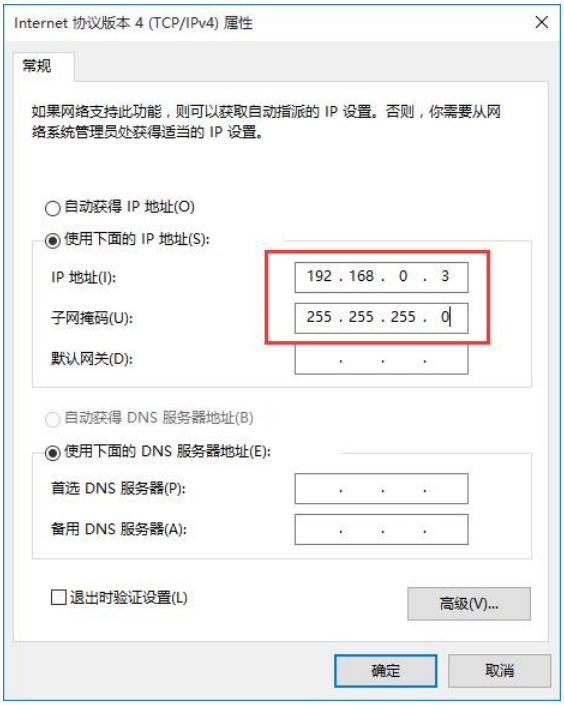
5 下载和试验

5.1 准备工作

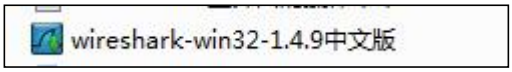
第一步：首先确认一下自己PC的网卡是否是千兆网卡，用户可以点击本地连接查看，再用五类+或者六类网线连接开发板的网口和PC的网口。

第二步：修改PC的IP地址为192.168.0.3。PC的IP Address需要和mac_test.v中设置一致，不然网络调试助手会接收不到开发板发送的UDP数据包。

```
.source_mac_addr              (48'h00_0a_35_01_fe_c0)
.TTL                          (8'h80),
.source_ip_addr               (32'hc0a80002),
.destination_ip_addr          (32'hc0a80003),
.udp_send_source_port         (16'h1f90),
.udp_send_destination_port    (16'h1f90),
```



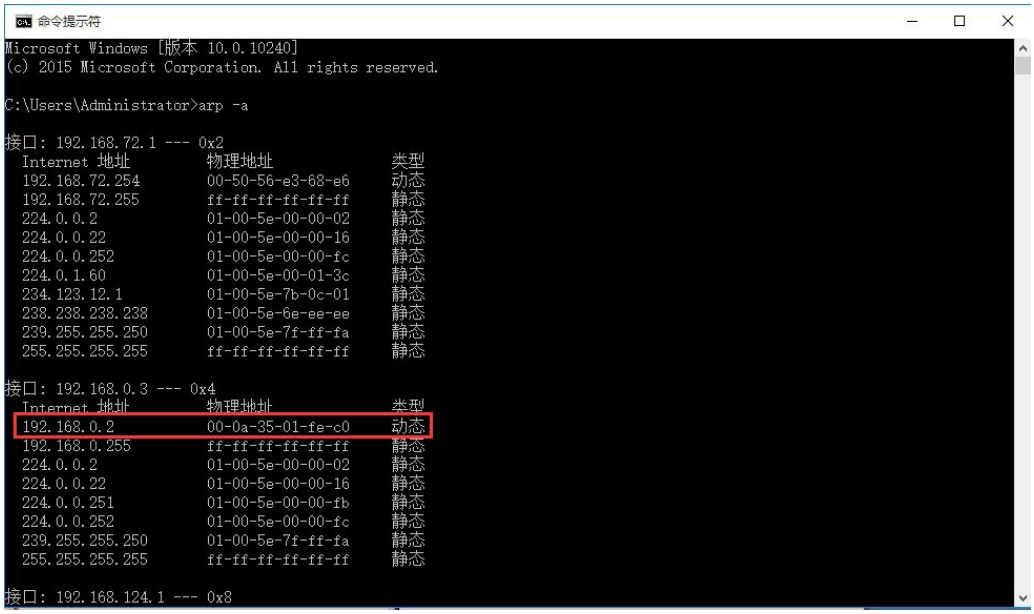
第三步：安装 Wireshark 是为了方便用户网络通信的调试，安装”\07_软件工具及驱动\网络调试工具\wireshark_cn”目录下的网络抓包工具 Wireshark, 我们在实验的时候可以用这工具来查看 PC 网口发送的数据和接收到的数据的详细信息。



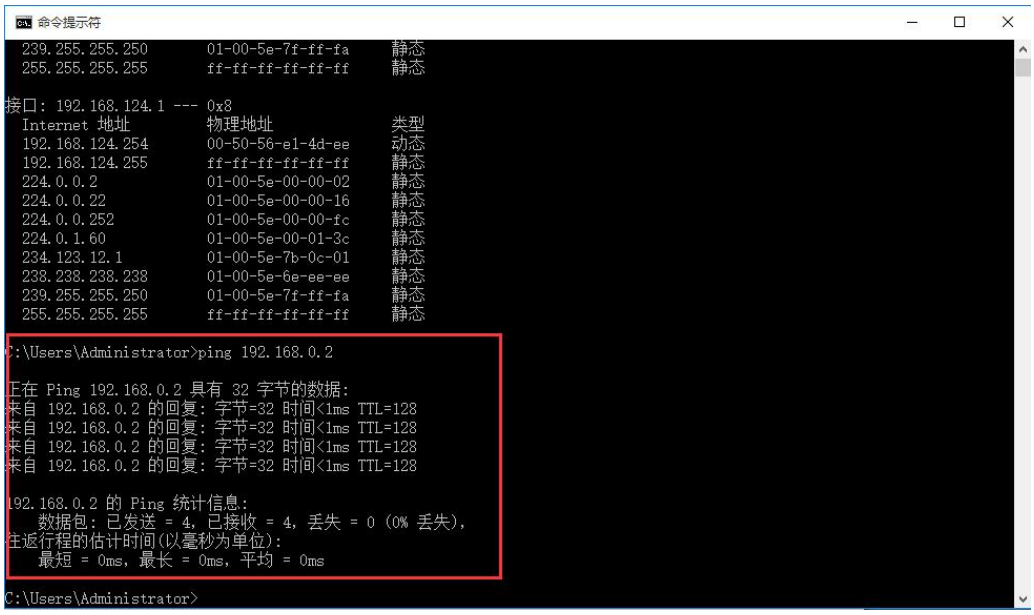
5.2 以太网通信测试

第一步：烧写ethernet_test.bit文件到FPGA，（如果需固化到FLASH中烧写ethernet_test.bin文件），等待两三秒，看到三个LED灯亮，为千兆网卡，两个灯亮为百兆网卡，如果都没亮，需要检查网口连接情况。

第二步：以管理员权限打开CMD窗口，输入arp –a查看ARP绑定结果，可以看到开发板的IP地址和MAC地址已经缓存。

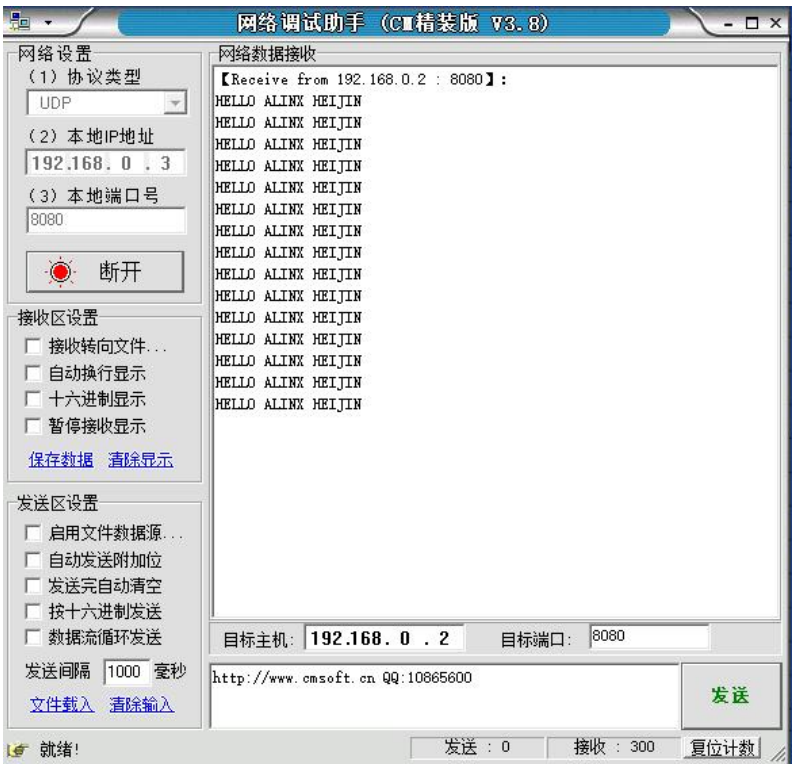


第三步：在CMD窗口中，输入ping 192.168.0.2查看PC与开发板是否ping通。



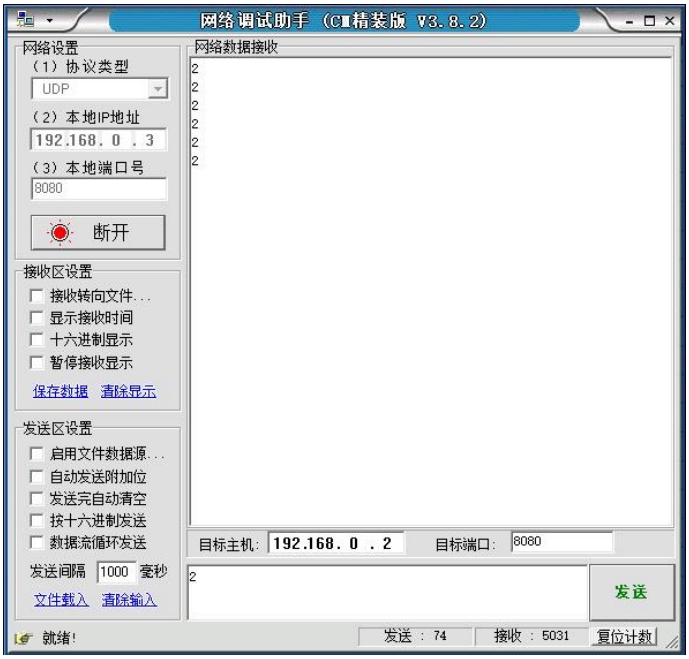
第四步：打开“\07_软件工具及驱动\网络调试工具\NetAssist “目录下的网络调试助手并设置参数如下，再按连接按钮(这里的本地的IP地址为PC的IP Address, 本地端口需要跟FPGA程序中的一致，为8080)。

这时网络数据接收窗口会显示FPGA发给PC的以太网数据包"HELLO ALINX HEIJIN"目标主机的IP地址需要和FPGA程序中的IP地址一致，目标端口号也需要和FPGA程序的一致(8080)。如下图网络显示:

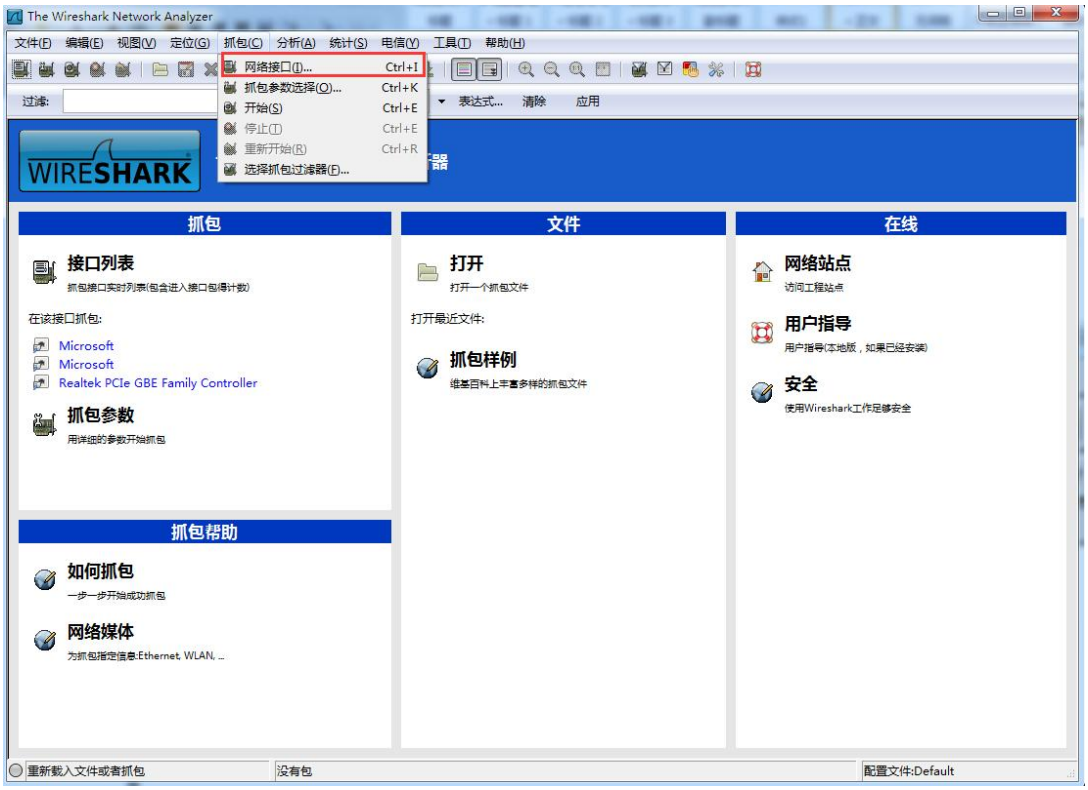


第五步：再在网络调试助手的发送窗口发送一大串字符,在网络的数据接收窗口我们可以看到从FPGA返回的数据也变成刚发送的字符串。

也可以发送较少字符，低于 46 字节，FPGA 程序会自动补充至 46 字节，如下图：

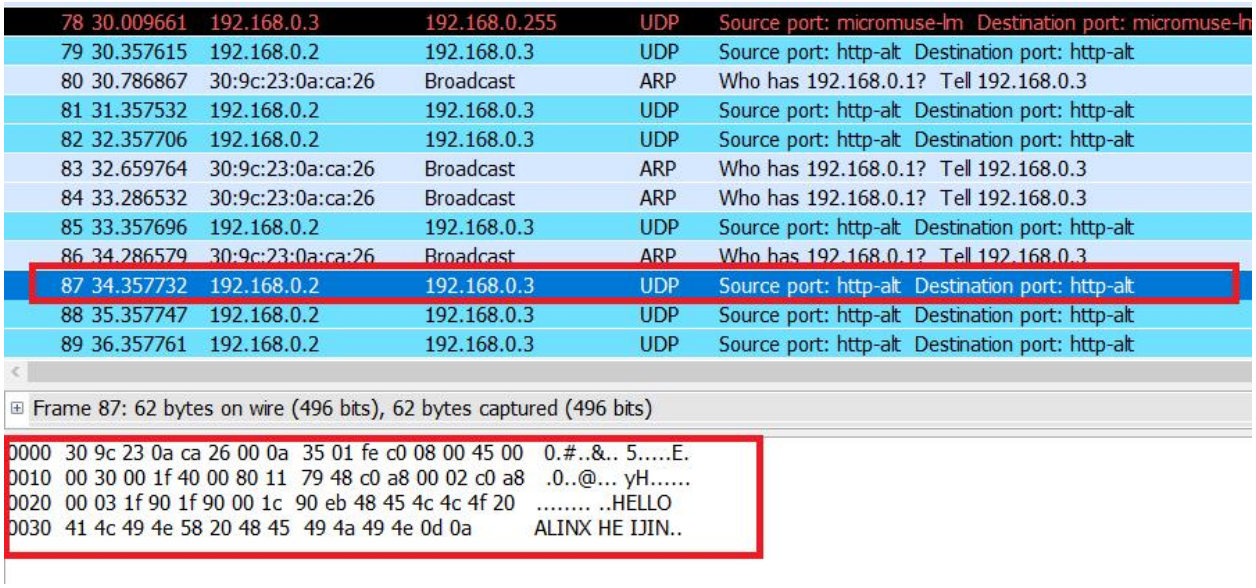


第六步：这一步对用户来讲是可选的，用户如果想查看更多数据包传输的信息，可以使用网络抓包工具Wireshark来查看PC的网卡接收和发送的网络数据，打开安装好的wireshark抓包工具，点击菜单抓包->网络接口。



在弹出的抓包接口窗口选择PC的千兆网卡，按开始按钮开始抓包。

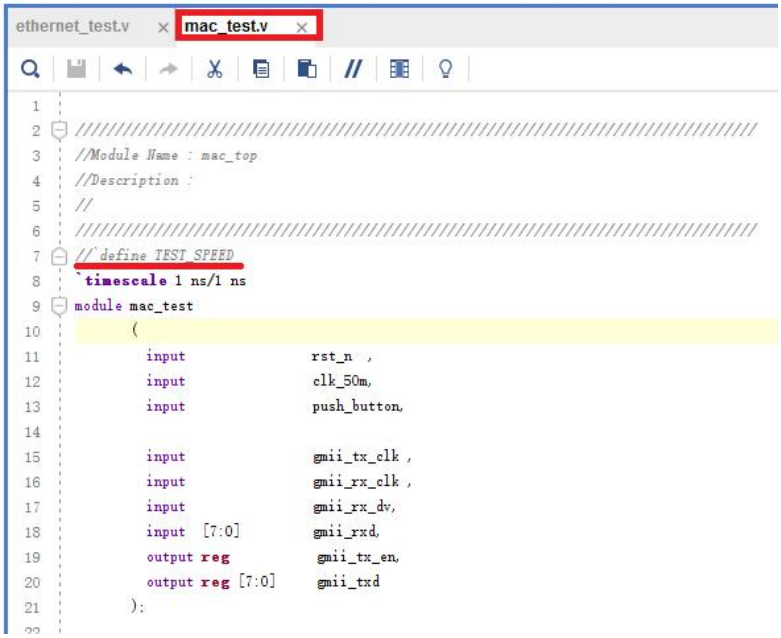
在wireshark抓包窗口我们可以看到开发板(192.168.0.2)向PC网口(192.168.0.2)发来的数据包，这里会显示数据包的目标MAC, 源MAC，IP包头和UDP包等信息, 如下图开发板抓包窗口显示：



5.3 以太网速度测试

我们可以通过抓包工具 wireshark 测量以太网部分的数据发送的速度，因为千兆以太网理想模式网络的速度可以达到 1Gbps,但实际因为每个数据包中有包头，CRC 等非数据字符，而且每包之间有空隙，一般千兆以太网的数据传输速度最高在 950Mbps 左右。传输是上下对称传输，就是说上行和下行都能达到 950Mbps 左右。在这里，因为抓包 wireshark 只能统计接收的数据包，而不会发送数据，所以这里只是测试 FPGA 发给电脑的输送数据的速度。

第一步：修改程序，修改工程中 mac_test.v 中的代码，如下图所示

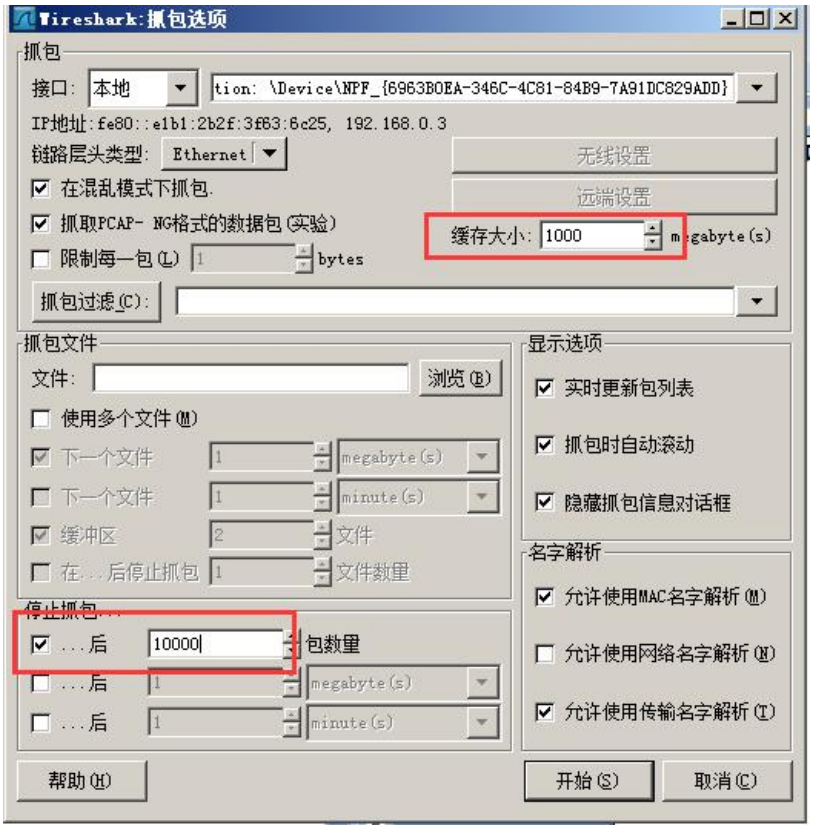


把上图中的红色标的”//”去掉，如下：

```
2  //////////////////////////////////////////////////////////////////////////////////
3  //Module Name : mac_top
4  //Description :
5  //
6  //////////////////////////////////////////////////////////////////////////////////
7  `define TEST_SPEED
8  `timescale 1 ns/1 ns
9  module mac_test
10       (
11         input              rst_n  ,
12         input              clk_50m,
13         input              push_button,
14
15         input              gmii_tx_clk ,
16         input              gmii_rx_clk ,
17         input              gmii_rx_dv,
18         input  [7:0]       gmii_rxd,
19         output reg         gmii_tx_en,
20         output reg [7:0]   gmii_txd
21       );
22
23 localparam UDP_WIDTH = 32 ;
```

第二步：编译综合完成后下载测试程序，下载完成后这时FPGA网络会进行动态绑定，PC机的IP设置请参照5.2章节（如已设置忽略）。

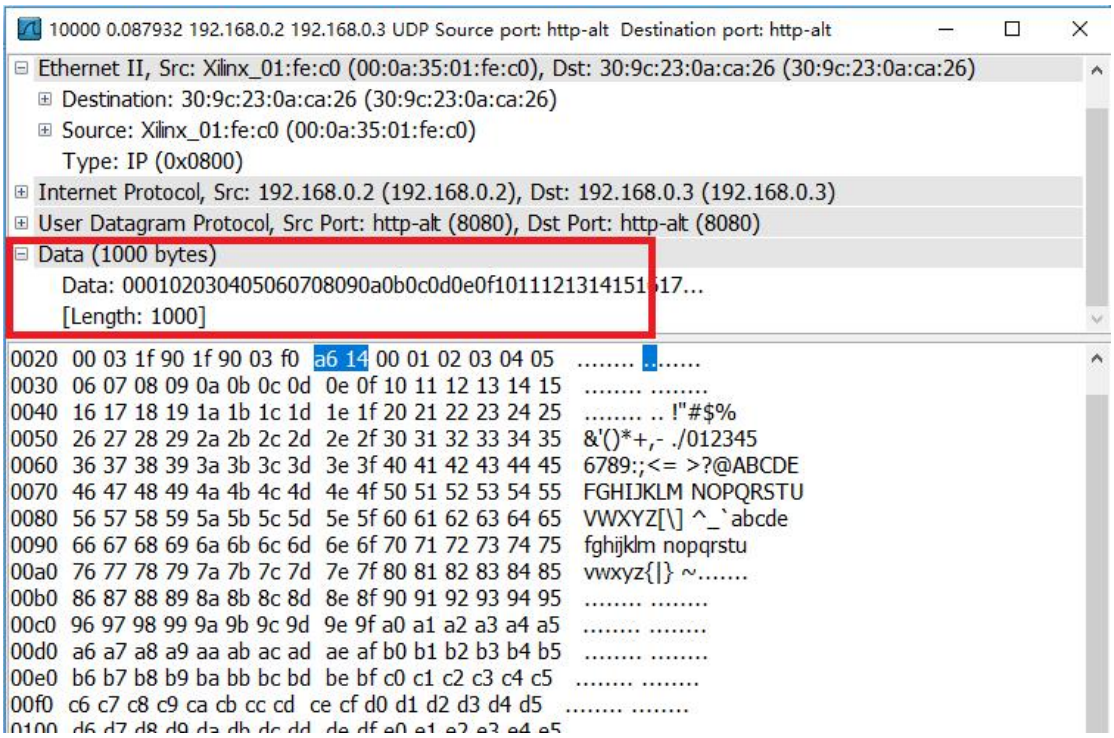
打开 wireshark，点击菜单“抓包”->”抓包参数选择”。在抓包选项中设置缓冲大小为 1000megabyte，在 10000 个数据包之后停止抓包。（这里需要注意：如果不设置的话 wireshark 因为抓的包太多，处理不了导致软件崩溃）



按开始后抓包，等待接收到 10000 个包后停止抓包，用时 0.087932 秒。

9994	0.087930	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
9995	0.087930	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
9996	0.087930	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
9997	0.087931	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
9998	0.087932	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
9999	0.087932	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10000	0.087932	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10001	0.087933	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10002	0.087933	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10003	0.087934	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10004	0.087934	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10005	0.087934	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10006	0.087935	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10007	0.087935	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10008	0.088059	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10009	0.088060	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10010	0.088060	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10011	0.088060	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10012	0.088061	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt
10013	0.088062	192.168.0.2	192.168.0.3	UDP	Source port: http-alt Destination port: http-alt

我们可以看到每个数据包的内容都是 00->FF 的数据，总共是 3 个 00->FF，1 个 00->e7 的数据。总共是 1000 个有效数据。

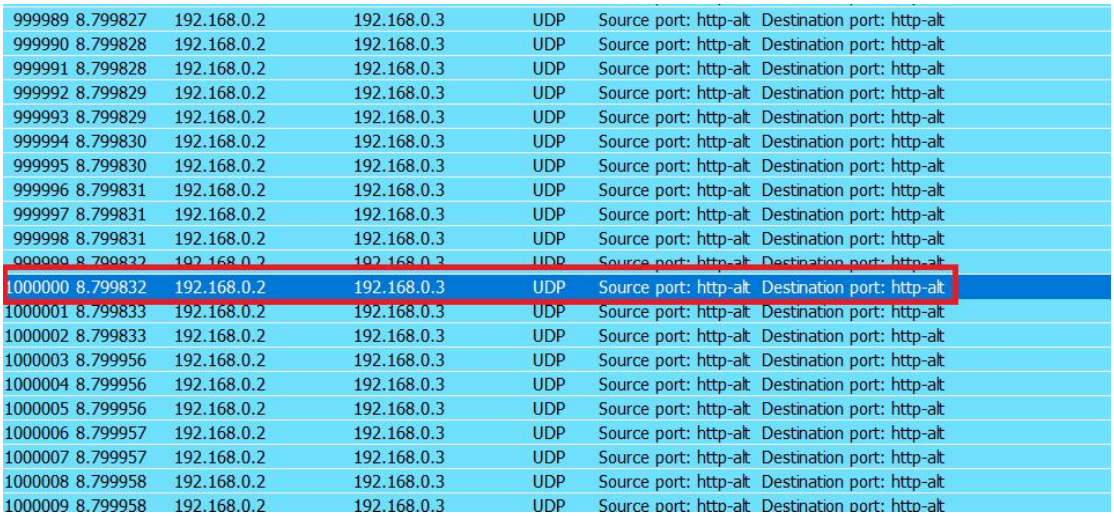


这里我们看到软件用了 0.087932 秒收到了 10000 个数据包，每个数据包里含有的有效数据是 1000 个字节(8bit)。所以我们可以算出以太网的传输速度为：

传输速度 = 1000* 10000 * 8 / 0.087932 = 910Mbps。

这里提示一下，网络传输的速度跟包长也有关系，数据包长越长，效率就越高，传输数据的速度会变高；数据包越短，效率就越低，传输数据的速度就慢。这个道理我想大家应该明白的吧！

下面是 1000000 包的时间是 8.799832 秒，自己可以按照上面的方面算下速度，速度是非常稳定的。



到这里为止，我们的以太网数据通信就全部结束了，要实现以太网的数据通信首先需要理解以太网数据包的格式和通信协议，还需要了解 MAC 和 PHY 之间的 GMII,RGMII,MII,RMII 接口的时序。千兆以太网的数据传输速度非常快，而且是全双工传输，而且 UDP 通信的数据速度可以达到 900Mbps 以上，非常适合高速数据传输的场合，比如视频图像传输，高速数据采集等等。