Gigabit Ethernet Transmission Experiment

Technical Email: alinx@aithtech.com Sales Email: rachel.zhou@alinx.com

1 Experiment Introduction

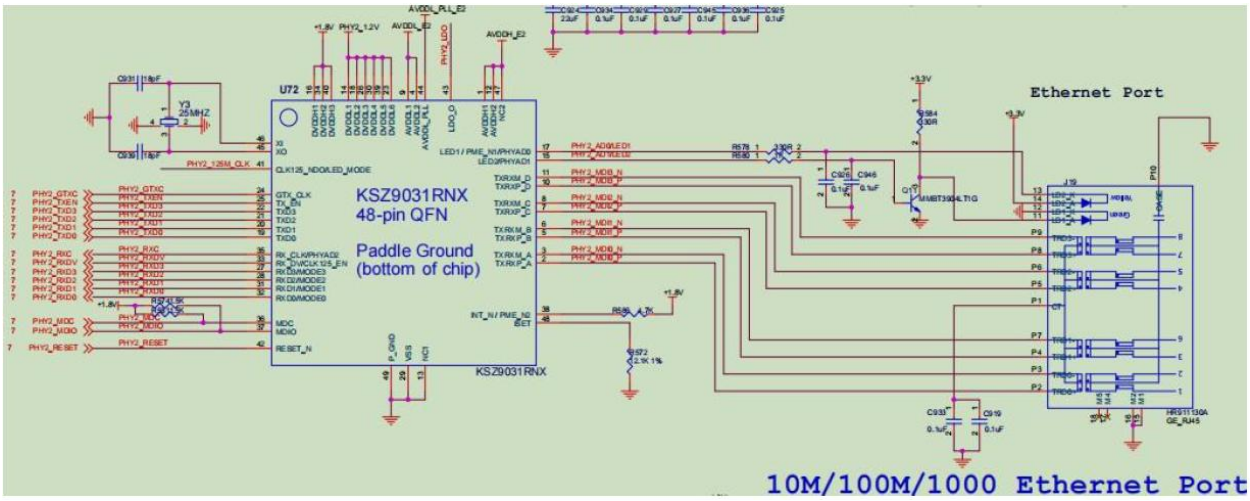
This experiment will realize Gigabit Ethernet data communication between “FPGA” chip and “PC”, and the communication protocol adopts Ethernet UDP communication protocol. The FPGA communicates with the “Gigabit PHY” chip on the development board through the “RGMII” bus. The “Gigabit PHY” chip sends data to the PC through the network cable. The program implements “ARP”, “UDP”, and “PING” functions, and also implements 100/1000M adaptation.

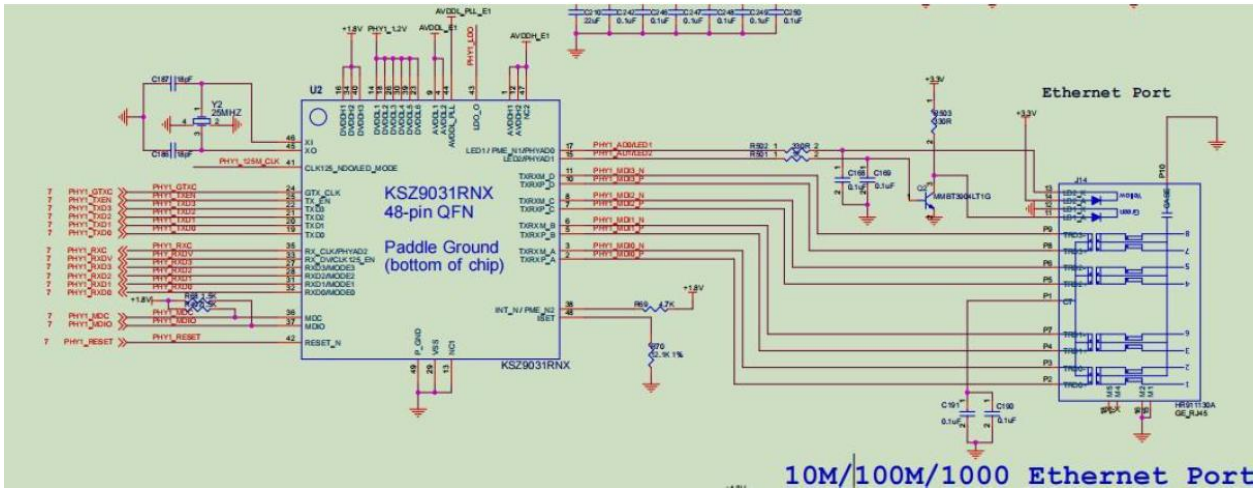
2 Hardware Introduction

The Ethernet chip uses Micrel's KSZ9031RNX Ethernet PHY chip to provide network communication services to users, and connects to FPGA or ZYNQ development board through FMC connector. The KSZ9031RNX chip supports 10/100/1000 Mbps network transmission rate and communicates with the FPGA through the RGMII interface. KSZ9031RNX supports “MDI/MDX” adaptation, various speed adaptation, “Master/Slave” adaptation, and supports “MDIO” bus for PHY register management.

When the network is connected to Gigabit Ethernet, the data transmission of ZYNQ or FPGA and PHY chip KSZ9031RNX is communicated through RGMII bus, the transmission clock is 125Mhz, and the data is sampled on the rising edge and falling samples of the clock.

When the network is connected to 100M Ethernet, the data transmission of ZYNQ or FPGA and PHY chip KSZ9031RNX is communicated through RMII bus, and the transmission clock is 25Mhz. Data is sampled on the rising edge and falling samples of the clock. For detailed pin descriptions and RGMII/MII communication timing, please refer to the KSZ9031RNX chip manual.

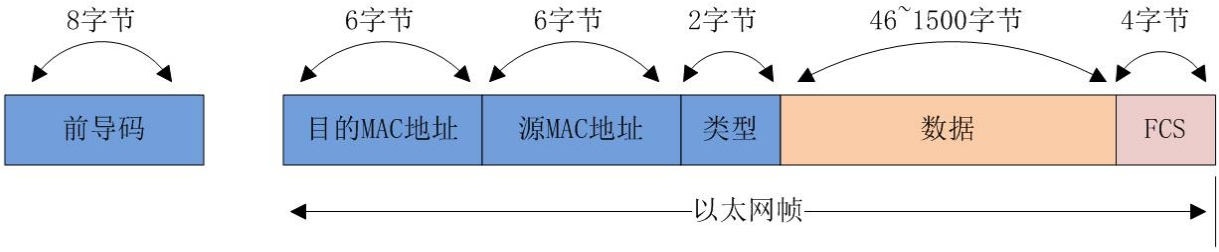




3 Ethernet frame

3.1 Ethernet frame format

The following is the frame format of Ethernet:



Preamble: 8 bytes, 7 consecutive “8'h55” plus “1 8'hd5”, indicating the beginning of a frame, used for synchronization of data of both devices.

Destination MAC address: 6 bytes, which stores the physical address of the destination device, that is, the MAC address.

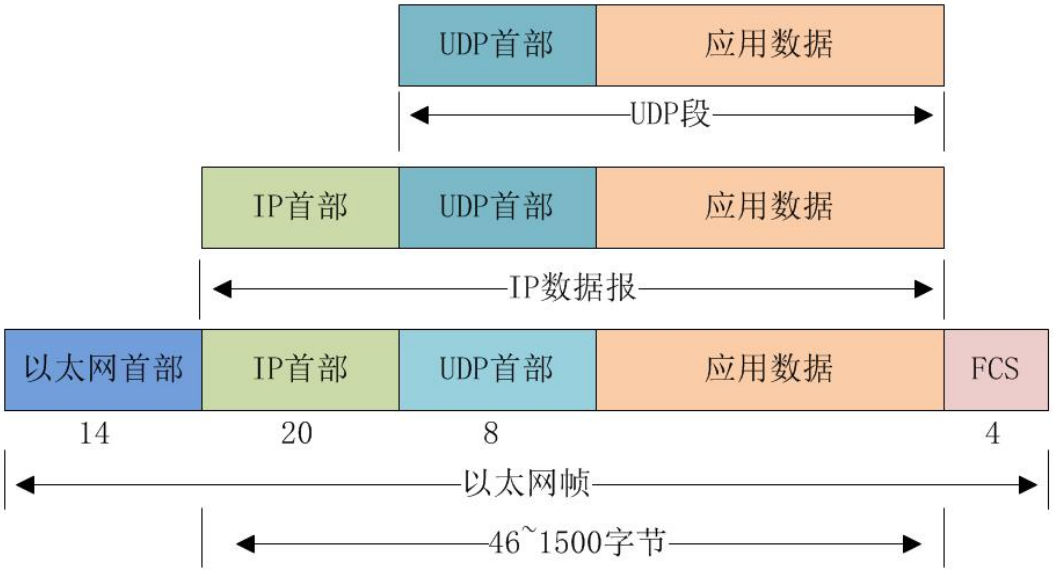
Source MAC address: 6 bytes, that stores the physical address of the sender device.

Type: 2 bytes, used to specify the protocol type. Commonly used are 0800 for IP protocol, 0806 for ARP, and 8035 for RARP.

Data: 46 to 1500 bytes, at least 46 bytes, insufficient to complete 46 bytes, for example, the IP protocol layer is included in the data portion, including its IP header and data.

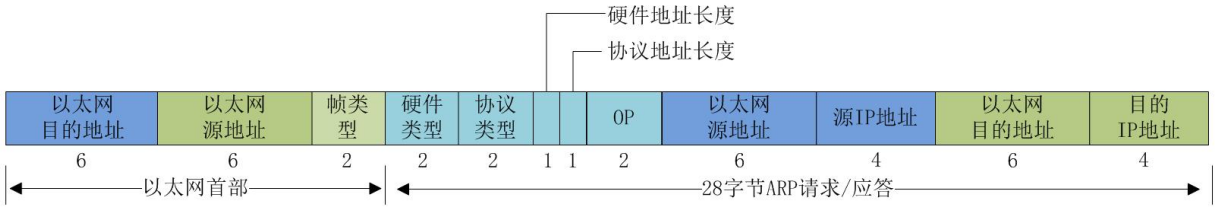
FCS: The end of the frame, 4 bytes, called the frame check sequence. The 32-bit CRC is used to check the destination MAC address field to the data field

Further extension, taking the UDP protocol as an example, you can see that its structure is as follows, except for the 14 bytes of the Ethernet header, the data part contains the IP header, the UDP header, and the application data is 46 to 1500 bytes.



3.2 ARP Datagram format

The ARP address resolution protocol (ARP) is used to obtain a physical address based on an IP address. The host sends an ARP request broadcast (the MAC address is 48'hff_ff_ff_ff_ff_ff) containing the destination IP address to the host on the network, and receives the return message, thereby determining the physical address of the target, and saving the IP address and the physical address after receiving the return message. In the cache, and for a period of time, the ARP cache is directly queried to save resources on the next request. The figure below shows the ARP datagram format.



Frame type: ARP frame type is two bytes 0806

Hardware type: refers to the link layer network type, 1 is Ethernet.

Protocol type: refers to the type of address to be converted, using 0x0800 IP type, and the subsequent hardware address length and protocol address length correspond to 6 and 4 respectively.

In the OP field, 1 indicates an ARP request, and 2 indicate an ARP response.

E.g： |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

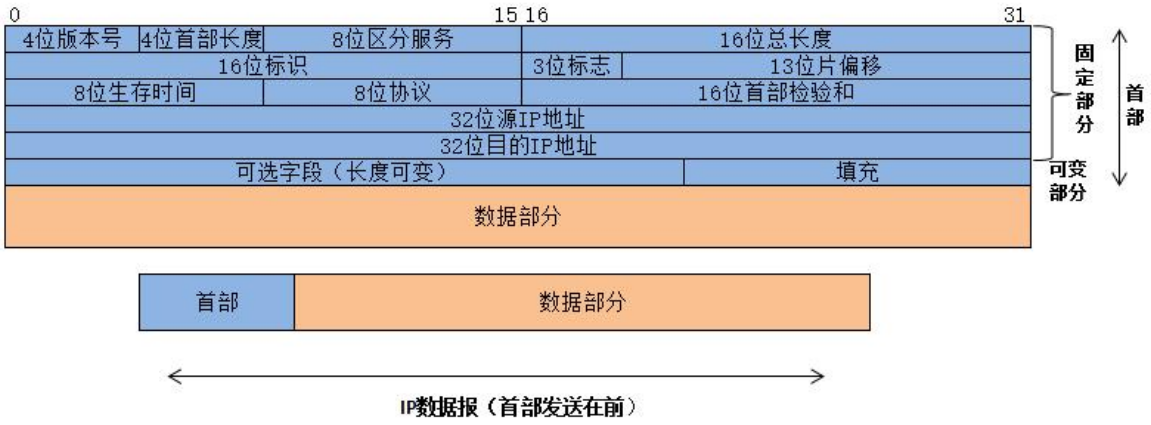
|c0 a8 00 02| ff ff ff ff ff ff|c0 a8 00 03| 1 means sending an ARP request to the 192.168.0.3 address

|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| means sending an ARP reply to the 192.168.0.2 address

3.3 IP Datagram Header

Since the UDP protocol packet is one of the IP packets, let us introduce the data format of the IP packet. The following figure shows the header format of the IP packet. The first 20 bytes of the packet header are fixed and the latter is variable.



Revision:

4 digits, refers to the IP protocol version. The current IP protocol version number is 4 (ie IPv4)

Head length:

4 bits, the maximum value that can be represented is 15 units (one unit is 4 bytes), so the maximum length of the IP header is 60 bytes.

Differentiated services:

8 bits, used to get better service, is called service type in the old standard, but it has never been used. In 1998, this field was renamed to differentiate service. This field is only works when using “DiffServ”. This field is not used under normal circumstances.

Total length:

16 bits, the length sum of the header and the data, in bytes, so the maximum length of the datagram is 65535 bytes. The total length must not exceed the maximum transmission unit MTU

Logo:

16 bits, it is a counter used to generate the identity of the datagram

Flag:

It is 3 digits, and currently only the first two meaningful MFs; the lowest digit of the flag field is MF (More Fragment)

MF=1 means "there are still fragments".

MF=0 means the last slice DF, One bit in the middle of the flag field is DF (Don't Fragment), which is allowed only when DF=0

Slice offset:

12 bits, refer to the relative position of a slice in the original packet after fragmentation. The slice offset is offset by 8 bytes.

Time to live:

8 bits, recorded as TTL (Time To Live). The maximum number of routers the datagram can pass through the network. The TTL field is initially set by the sender with an 8 bit field. The recommended initial value is specified by the assigned number RFC. The current value is 64. The TTL is always set to a maximum of 255 when sending an ICMP echo reply.

Protocol:

8 bits, indicating which protocol is used for the data carried by this datagram, so that the IP layer of the destination host hands over the data part to which process, 1 is ICMP protocol, 2 is IGMP protocol, and 6 is TCP protocol. 17 is expressed as UDP protocol

Header Checksum

16bits, only check the header of the datagram without checking the data portion. Here, the CRC check code is not used and a simple calculation method is used.

Source address and destination address:

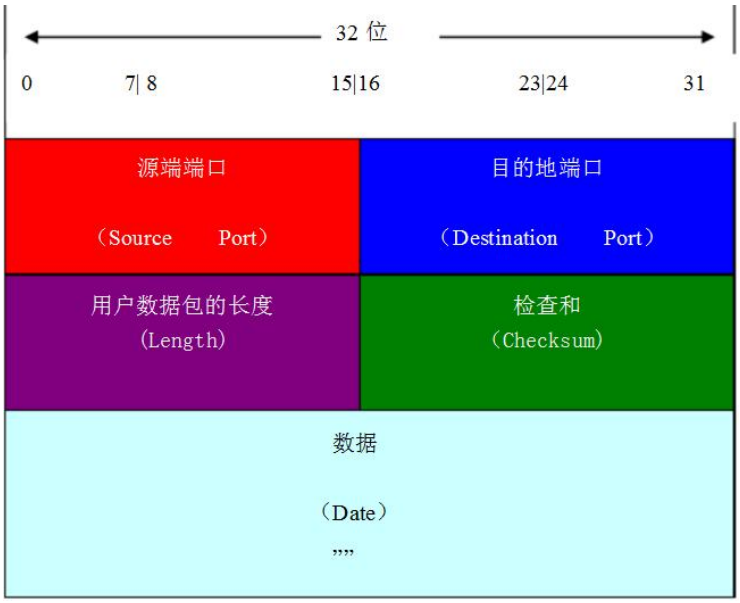
Each takes 4 bytes and records the source and destination addresses respectively.

3.4 UDP Protocol

UDP is the abbreviation of User Datagram Protocol. UDP only provides a basic, low-latency communication called a datagram. The so-called datagram is a kind of self-addressing information, data pack from the sender to the receiver. UDP protocol is often used for image transmission, network monitoring data exchange and other data transmission speed requirements are relatively high

The header format of the UDP protocol:

The UDP header consists of 4 domains, each occupying 2 bytes, details as follows:



1) UPD Source Port

2) Destination Port

3) Length of Diagram

4) Checksum

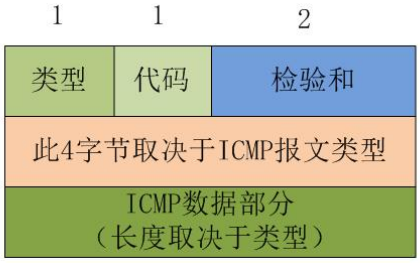
The UDP protocol uses its port number to reserve its own data transmission channel for different applications. The data sender sends the UDP datagram through the source port, and the data receiver receives the data through the destination port.

The length of the datagram refers to the total number of bytes including the header and data parts. Because the length of the header is fixed, this field is primarily used to calculate the variable length portion of the data (also known as the data load). The maximum length of a datagram varies depending on the operating environment. In theory, the maximum length of a datagram containing a header is 65535 bytes. However, some practical applications tend to limit the size of the datagram, sometimes it will be reduced to 8192 bytes.

The UDP protocol uses the checksum in the header to ensure data security. The check value is first calculated by the data sender through a special algorithm, and after being passed to the receiver, it needs to be recalculated. If a datagram is falsified by a third party during transmission or damaged due to line noise, etc., the calculated values of the sender and the receiver will not match, and the UDP protocol can detect an error. Although UDP provides error detection, when an error is detected, the error is corrected, simply throwing away the corrupted message segment or providing a warning message to the application.

3.5 Ping function

"ICMP" is an IP layer sub-protocol of the "TCP/IP" protocol suite. It is included in IP datagram and is used to transfer control messages between IP hosts and routers. The control message refers to whether the network is connected and the host is reachable. The "ping" function uses a loopback request and an answer message. The loopback request packet type is "8'h08", and the response packet type is "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 SMI (MDC/MDIO) Bus Interface

The "Serial Management Interface", also known as the "MII Management Interface", includes two signal lines, "MDC" and "MDIO". "MDIO" is a PHY management interface used to read/write PHY registers to control PHY behavior or to obtain PHY status. MDC provides clocks for MDIO, provided by the MAC side, which is also the FPGA side in this lab. In the RTL8211EG document, you can see that the MDC has a minimum period of 400ns, that is, the maximum clock is 2.5MHz.

Table 61. MDC/MDIO Management Timing Parameters

Symbol	Description	Minimum	Maximum	Unit
t_1	MDC High Pulse Width	160	-	ns
t_2	MDC Low Pulse Width	160	-	ns
t_3	MDC Period	400	-	ns
t_4	MDIO Setup to MDC Rising Edge	10	-	ns
t_5	MDIO Hold Time from MDC Rising Edge	10	-	ns
t_6	MDIO Valid from MDC Rising Edge	0	300	ns

4.1 SMI Frame Format

As shown below, the SMI read and write frame format:

	Management Frame Fields							
	Preamble	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
Read	1…1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
Write	1…1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Name	Description
Preamble	32 consecutive logical "1"s sent by the MAC, synchronized to the MDC signal for synchronization between the MAC and the PHY
ST	Frame start bit, fixed at 01
OP	Opcode, 10 means read, 01 means write
PHYAD	PHY address, 5 bits
REGAD	Register address, 5 bits
TA	Turn Around, MDIO direction conversion, in the write state, no need to change direction, the value is 10, in the read state, the MAC output is high impedance, in the second cycle, the PHY will pull MDIO low
DATA	A total of 16bits of data
IDLE	Idle state, in which MDIO is in a high-impedance state, pulled up by an external pull-up resistor

4.2 Read and Write Timing

4.2.1 Read Timing

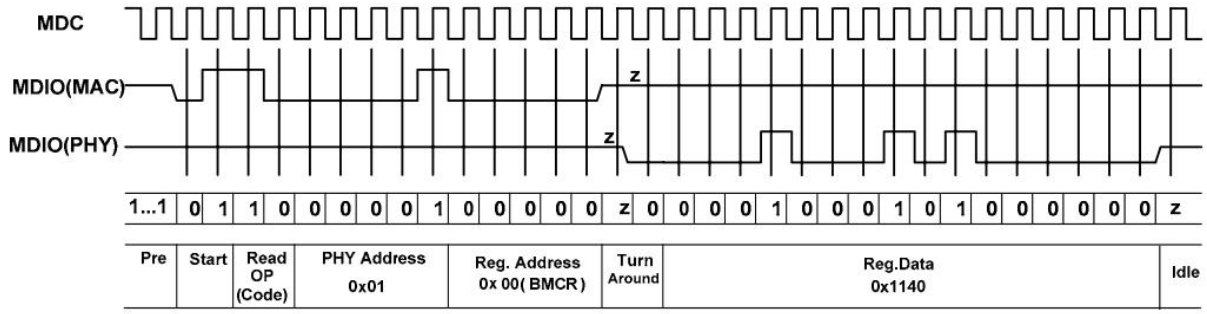


Figure 8. MDC/MDIO Read Timing

It can be seen that in the Turn Around state, MDIO is in a high impedance state in the first cycle, and the second cycle is pulled down by the PHY terminal.

4.2.2 Write timing

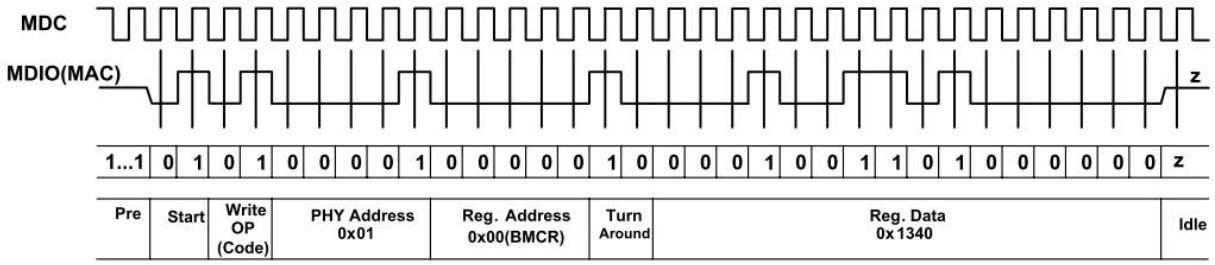
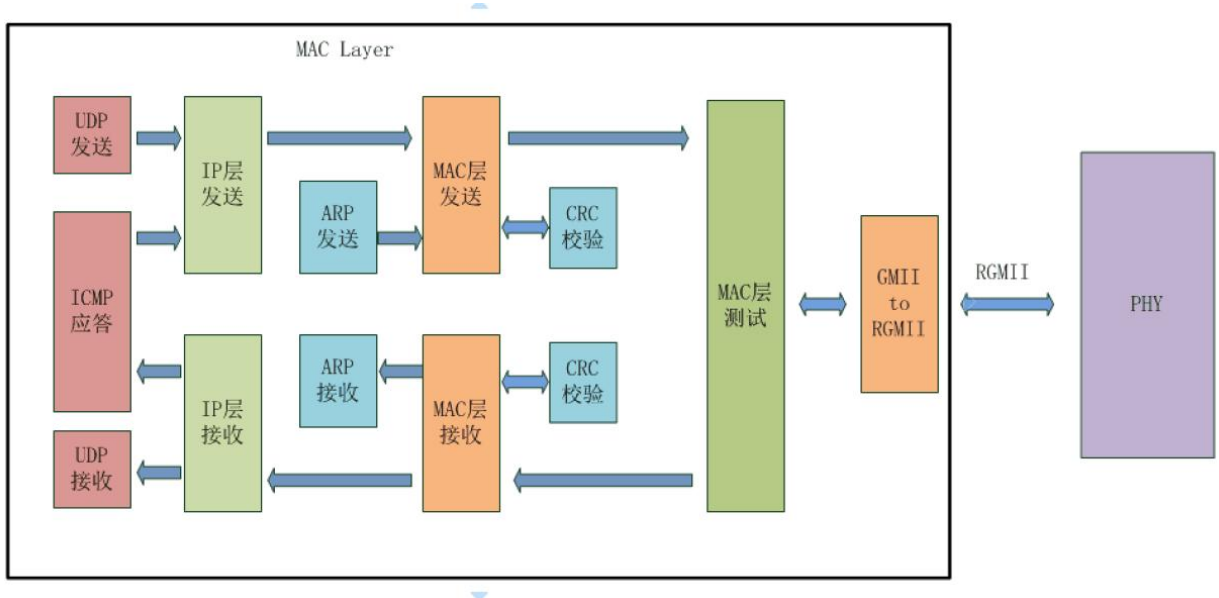


Figure 9. MDC/MDIO Write Timing

In order to ensure that the data can be collected correctly, the data is prepared before the rising edge of the MDC. In this experiment, the data is sent for the falling edge and the rising edge is used to receive the data.

5 Programming

This experiment uses Gigabit Ethernet "RGMII" communication as an example to design the verilog program. It will send the preset "UDP" data to the network first, and send it once every second. If the FPGA detects the "UDP" packet sent by the network port, it will receive it. The data packets are stored in the "RAM" inside the FPGA, and the data packets in the "RAM" are continuously sent back to the ethernet network through the network port. The program is divided into two parts, namely tranismitting and receiving, and implementing "ARP", "UDP", and "Ping" functions. The following is a block diagram of the AX7103 principle implementation:

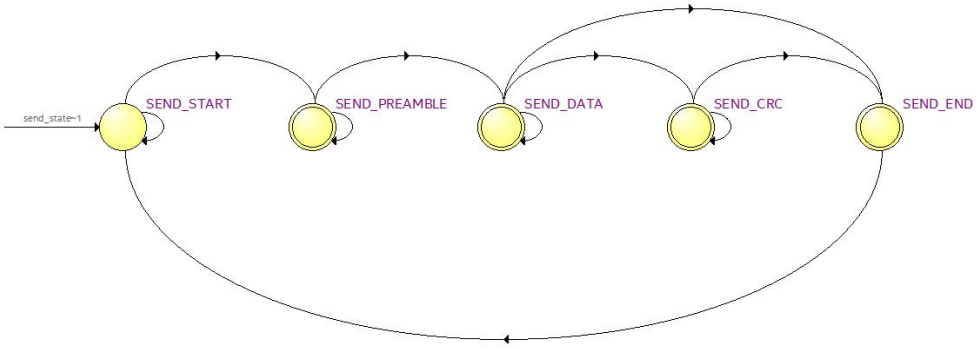


Block diagram

5.1 Transmitting Parts

5.1.1 MAC Layer Transmit

In the transmitting part, “mac_tx.v” is the “MAC” layer transmitting module, first in the “SEND_START” state, waiting for the “mac_tx_ready” signal. If it is valid, it indicates that the “IP” or “ARP” data is ready and can be sent. Then enter the preamble status, transmit “mac_data_req” at the end, request “IP” or “ARP” data, then enter the send data state, and finally enter the send “CRC” state. In the process of transmitting data, it is necessary to perform “CRC” check at the same time.



Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
crc_result	in	32	CRC32 results
crcen	out	1	CRC enable signal
crcre	out	1	CRC reset signal
crc_din	out	8	CRC module input signal
mac_frame_data	in	8	Data from IP or ARP
mac_tx_req	in	1	MAC send request
mac_tx_ready	in	1	IP or ARP data is ready
mac_tx_end	in	1	IP or ARP data has been transferred
mac_tx_data	out	8	Send data to the PHY
mac_send_end	out	1	End of MAC data transmission
mac_data_valid	out	1	MAC data valid signal, ie gmii_tx_en
mac_data_req	out	1	The MAC layer requests data from IP or ARP.

5.1.2 MAC transmission mode

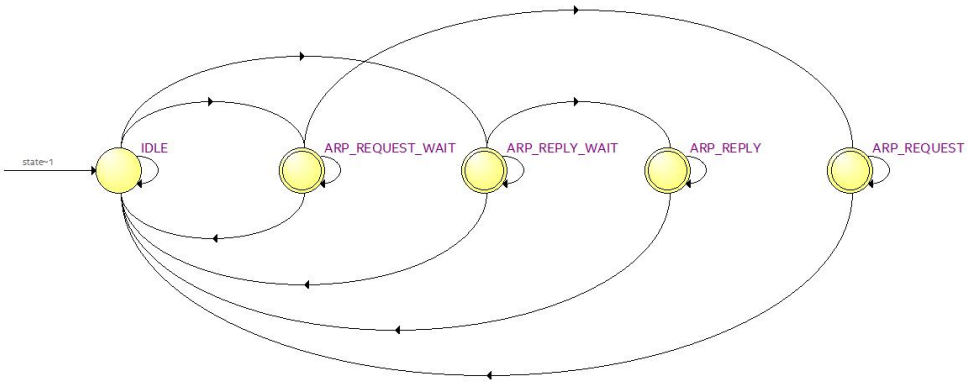
The "mac_tx_mode.v" in the project is the transmission mode selection, and the corresponding signals and data are selected according to the transmission mode of IP or ARP.

Signal Name	Direction	Width (bit)	Description
clk	in	1	System clock

rst_n	in	1	Asynchronous reset, low reset
mac_send_end	in	1	MAC transmission ends
arp_tx_req	in	1	ARP sends a request
arp_tx_ready	in	1	ARP data is ready
arp_tx_data	in	8	ARP data
arp_tx_end	in	1	ARP data is sent to the end of the MAC layer
arp_tx_ack	in	1	ARP sends a response signal
ip_tx_req	in	1	IP send request
ip_tx_ready	in	1	IP data is ready
ip_tx_data	in	8	IP data
ip_tx_end	in	1	IP data is sent to the end of the MAC layer
mac_tx_ready	out	1	MAC data is ready for signal
ip_tx_ack	out	1	IP send response signal
mac_tx_ack	in	1	MAC sends a response signal
mac_tx_req	out	1	MAC send request
mac_tx_data	out	8	MAC send data
mac_tx_end	out	1	End of MAC data transmission

5.1.3 ARP Transmission

In the sending part, “arp_tx.v” is an “ARP” sending module. In the IDLE state, waiting for an “ARP” sending request or an “ARP” reply request signal, then entering a request or response waiting state, and notifying the “MAC” layer that the data is ready, waiting for the “mac_data_req” signal, after Enter the request or reply data transmission status. Since the data is less than 46 bytes, it is necessary to complete the 46-byte transmission.

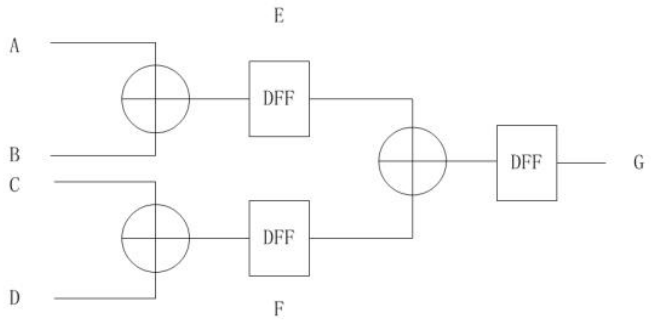


Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
destination_mac_addr	in	48	Destination MAC address sent
source_mac_addr	in	48	Source MAC address sent
source_ip_addr	in	32	Source IP address sent

destination_ip_addr	in	32	Destination IP address sent
mac_data_req	in	1	MAC layer request data signal
arp_request_req	in	1	Request signal for ARP request
arp_reply_ack	out	1	ARP reply response signal
arp_reply_req	in	1	ARP reply request signal
arp_rec_source_ip_addr	in	32	Source IP address received by ARP. When replying, it is placed in the destination IP address.
arp_rec_source_mac_addr	in	48	Source MAC address received by ARP. When replying, it is placed in the destination MAC address.
mac_send_end	in	1	MAC transmission ends
mac_tx_ack	in	1	MAC transmits a response
arp_tx_ready	out	1	ARP data is ready
arp_tx_data	out	8	ARP transmits data
arp_tx_end	out	1	ARP data transmission ends

5.1.4 IP Layer Transmission

In the transmitting part, “ip_tx.v” is the IP layer transmitting module. In the “IDLE” state, if “ip_tx_req” is valid, that is, “UDP” or “ICMP” transmits a request signal, enters the waiting for transmitting data length state, and then enters the checksum state, The checksum is to add all the data of the IP header to 16 bits, and finally add the carry to the lower 16 bits until the entry is 0, then the lower 16 bits are inverted to obtain the checksum result. In this program, the checksum is structured in a tree type and inserted into the pipeline, which can effectively reduce the delay of the addition part, but the disadvantage is that it consumes more logic resources. As shown in the following figure ABCD four inputs, A and B add, the result is E, C and D add, the result is F, then add E and F, the result is G, there is a register after each addition result.



After generating the checksum, it waits for the MAC layer data request, starts transmitting data, and requests UDP or ICMP data immediately after the IP header is sent. After the transmission is completed, enter the IDLE state.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock

rst_n	in	1	Asynchronous reset, low reset
destination_mac_addr	in	48	Destination MAC address Transmit
source_mac_addr	in	48	Source MAC address Transmit
source_ip_addr	in	32	Source IP address Transmit
destination_ip_addr	in	32	Destination IP address Transmit
TTL	in	8	Survival time
ip_send_type	in	8	Upper layer protocol number, such as UDP, ICMP
upper_layer_data	in	8	Data coming from UDP or ICMP
upper_data_req	out	1	Request data from the upper layer
mac_tx_ack	in	1	MAC transmits a response
mac_send_end	in	1	MAC transmission end signal
mac_data_req	in	1	MAC layer request data signal
upper_tx_ready	in	1	Upper “UDP”or “ICMP” data is ready
ip_tx_req	in	1	Transmit a request, come from the top
ip_send_data_length	in	16	Total length of transmit data
ip_tx_ack	out		Generate an IP transmit response
ip_tx_busy	out		P transmits a busy signal
ip_tx_ready	out	1	IP data is ready
ip_tx_data	out	8	IP data
ip_tx_end	out	1	IP data is sent to the end of the MAC layer

5.1.5 IP Transmit Mode

The “ip_tx_mode.v” in the project is the transmission mode selection, and the corresponding signal and data are selected according to the transmission mode of “UDP” or “ICMP”.

Signal Name	Direction	Width (bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
mac_send_end	in	1	End of MAC data transmission
udp_tx_req	in		UDP transmit request
udp_tx_ready	in	1	UDP data is ready
udp_tx_data	in	8	UDP transmit data
udp_send_data_length	in	16	UDP transmit data length
udp_tx_ack	out	1	Output UDP transmit response
icmp_tx_req	in	1	ICMP transmit s a request
icmp_tx_ready	in	1	ICMP data is ready
icmp_tx_data	in	8	ICMP transmit s data
icmp_send_data_length	in	16	ICMP transmit data length
icmp_tx_ack	out	1	ICMP transmit s a response
ip_tx_ack	in	1	IP transmit response
ip_tx_req	in	1	IP transmit request
ip_tx_ready	out	1	IP data is ready
ip_tx_data	out	8	IP data

ip_send_type	out	8	Upper layer protocol number, such as UDP, ICMP
ip_send_data_length	out	16	Total length of transmit data

5.1.6 UDP Transmit

In the transmitting part, “udp_tx.v” is the UDP transmitting module. The first step is to write the data into the UDP transmitting RAM, and at the same time calculate the checksum. The second step is to transmit the data in the RAM. The UDP checksum is consistent with the IP checksum calculation method. The pseudo header is added during the calculation. The pseudo header includes the destination IP address, source IP address, network type, and UDP data length.

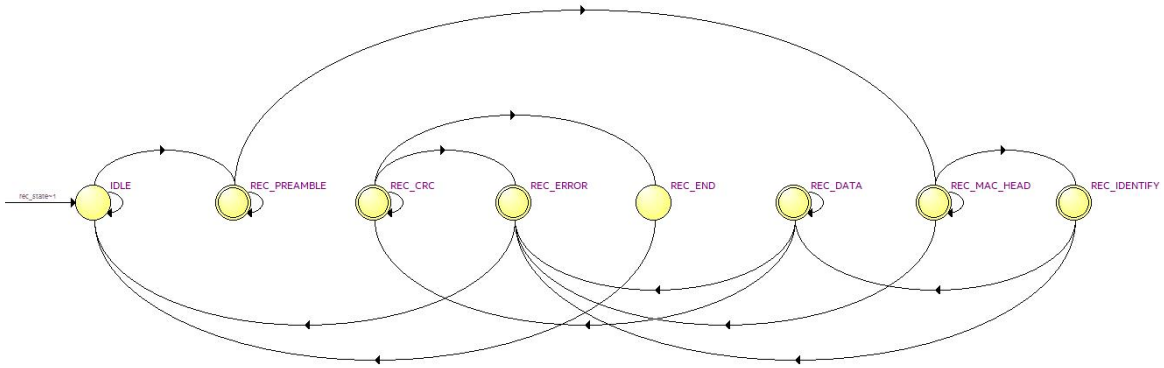
Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
source_ip_addr	in	32	Source IP address transmit
destination_ip_addr	in	32	Destination IP address transmit
udp_send_source_port	in	16	Source port number
udp_send_destination_port	in	16	Destination port number
udp_send_data_length	in	16	UDP transmits the data length, the user needs to give its value
ram_wr_data	in	8	Write UDP RAM data
ram_wr_en	in	1	Write RAM enable
udp_ram_data_req	out	1	Request to write RAM data
mac_send_end	in	1	MAC transmission end signal
udp_tx_req	in	1	UDP transmit request
ip_tx_req	out	1	IP transmit request
ip_tx_ack	in	1	IP response
udp_data_req	in	1	IP layer request data
udp_tx_ready	out	1	UDP data is ready
udp_tx_data	out	8	UDP data
udp_tx_end	out	1	End of UDP transmission (unused)
almost_full	out	1	Fifo close to full sign

5.2 Receiving Parts

5.2.1 MAC Layer Reception

In the receiving part, where “mac_rx.v” is the mac layer receiving file, firstly, in the IDLE state, it is necessary to determine whether the “rx_dv” signal is high, and in the “REC_PREAMBLE” preamble state, the preamble is received. Then enter the receiving MAC header state, that is, the destination MAC address, the source MAC address, the type, cache them, and judge whether the preamble is correct in this state, If the error occurs, the “REC_ERROR” error state is entered, and in the “REC_IDENTIFY” state, the type is judged to be IP (8'h0800) or ARP (8'h0806). Then enter the receive data state, transfer the data to the IP or ARP module, wait for the IP or ARP data to be received, and then receive the CRC data.

And in the process of receiving data, the received data is subjected to CRC processing, and the result is compared with the received CRC data to determine whether the data is received correctly, and if it is correct, the error is entered into the ERROR state.

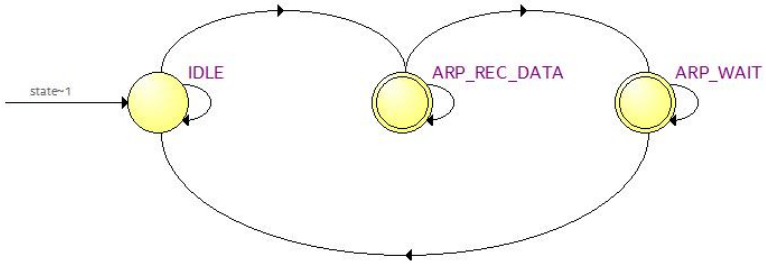


Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
crc_result	in	32	CRC32 results
crcen	out	1	CRC enable signal
crcre	out	1	CRC reset signal
crc_din	out	8	CRC module input signal
rx_dv	in	1	Rx_dv signal coming from the PHY layer
mac_rx_datain	in	8	Data received from the PHY layer
checksum_err	in	1	IP layer checksum error
ip_rx_end	in	1	End of IP layer reception
arp_rx_end	in	1	ARP layer reception ends
ip_rx_req	out	1	Request IP layer to receive
arp_rx_req	out	1	Request ARP reception
mac_rx_dataout	out	8	The MAC layer receives data output to IP or ARP.
mac_rec_error	out	1	MAC layer receiving error
mac_rx_destination_mac_addr	out	48	Destination IP address received by the MAC
mac_rx_source_mac_addr	out	48	Source IP address received by the MAC

5.2.2 ARP Receive

The “arp_rx.v” in the project is an ARP receiving module, which implements ARP data reception. In the IDLE state, it receives the “arp_rx_req” signal sent from the MAC layer and enters the ARP receiving state. In this state, the destination MAC address, the source MAC address, the destination IP address, and the source IP address are extracted, and it is determined whether the operation code OP is a request or a response. If it is a request, it is judged whether the received destination IP address is a local address, and if so, the response request “signal arp_reply_req” is sent, and if not, it is ignored. If the OP is a response, it is determined whether the received destination IP address and the destination MAC

address are consistent with the local device. If yes, the “arp_found signal” is raised to indicate that the address of the other party is received. The other party's MAC address and IP address are stored in the ARP cache.



Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
local_ip_addr	in	32	Local IP address
local_mac_addr	in	48	Local MAC address
arp_rx_data	in	8	ARP receives data
arp_rx_req	in	1	ARP receiving request
arp_rx_end	out	1	ARP reception completed
arp_reply_ack	in	1	ARP reply response
arp_reply_req	out	1	ARP reply request
arp_rec_source_ip_addr	in	32	Source IP address received by ARP
arp_rec_source_mac_addr	in	48	Source MAC address received by ARP
arp_found	out	1	ARP received the request response correctly

5.2.3 IP Layer Receiving Module

In the project, “ip_rx” is the IP layer receiving module, which realizes data reception, information extraction, and checksum checking at the IP layer. First, in the “IDLE” state, it judges the “ip_rx_req” signal sent from the MAC layer, enters the receiving IP header state, first extracts the header length and the total IP length in “REC_HEADER0”, enters the “REC_HEADER1” state, and extracts the destination IP address and source IP address in this state. , protocol type, send “udp_rx_req” or “icmp_rx_req” according to the protocol type. Check the checksum while receiving the header, add all the data received by the header, store it in the 32-bit register, and add the upper 16 bits to the lower 16 bits until the upper 16 bits are 0, then the lower 16 The bit is negated to determine whether it is 0. If it is 0, the test is correct. Otherwise, the error enters the IDLE state, and the frame data is discarded, waiting for the next reception.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System Clock
rst_n	in	1	Asynchronous reset, low reset
local_ip_addr	in	32	Local IP address

local_mac_addr	in	48	Local MAC address
ip_rx_data	in	8	Data received from the MAC layer
ip_rx_req	in	1	IP receiving request signal sent by the MAC layer
mac_rx_destination_mac_addr	in	48	Destination MAC address received by the MAC layer
udp_rx_req	out	1	UDP receive request signal
icmp_rx_req	out	1	ICMP receive request signal
ip_addr_check_error	out	1	Address check error signal
upper_layer_data_length	out	16	Data length of the upper layer protocol
ip_total_data_length	out	16	Total length of data
net_protocol	out	8	Network protocol number
ip_rec_source_addr	out	32	Source IP address received by the IP layer
ip_rec_destination_addr	out	32	Destination IP address received by the IP layer
ip_rx_end	out	1	End of IP layer reception
ip_checksum_error	out	1	IP layer checksum check error signal

5.2.4 UDP Receive

In the project, “udp_rx.v” is the UDP receiving module. In this module, the UDP header is received first, then the data part is received, and the data part is stored in the RAM. The UDP checksum check is performed at the same time as the reception. If the UDP data is an odd number of bytes, when the checksum is calculated, 8'h00 is added after the last byte, and the checksum calculation is performed. The verification method is the same as the IP checksum. If the verification is correct, the “udp_rec_data_valid” signal will be raised to indicate that the received UDP data is valid, otherwise it is invalid and waiting for the next reception.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System Clock
rst_n	in	1	Asynchronous reset, low reset
udp_rx_data	in	8	UDP receive data
udp_rx_req	in	1	UDP receive request
mac_rec_error	in	1	MAC layer receive error
net_protocol	in	8	Network protocol number
ip_rec_source_addr	in	32	Source IP address received by the IP layer
ip_rec_destination_addr	in	32	Destination IP address received by the IP layer
ip_checksum_error	in	1	IP layer checksum check error signal
ip_addr_check_error	in	1	Address check error signal
upper_layer_data_length	in	16	Data length of the upper layer protocol
udp_rec_ram_rdata	out	8	UDP receive RAM read data
udp_rec_ram_read_addr	in	11	UDP receive RAM read address
udp_rec_data_length	out	16	UDP receive data length
udp_rec_data_valid	out	1	UDP receiving data is valid

5.3 SMI Interface

5.3.1 SMI Read and write control module

The “smi_read_write.v” file is used for read/write control of the SMI interface. In the IDLE state, the MDC and MDIO control is implemented according to the SMI timing and the write request or the read request to the corresponding state.

Signal Name	Direction	Width (bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
mdc	out	1	MDC clock
mdio	inout	1	MDIO data
phy_addr	in	5	PHY address
reg_addr	in	5	Register address
write_req	in	1	Write request
write_data	in	16	Write data
read_req	in	1	Read request
read_data	out	16	Reading data
data_valid	out	1	Data valid
done	out	1	End of reading or writing

5.3.2 SMI Configuration module

The “smi_config.v” file reads “bit 10” and “bit[15:14]” of register “0x11” to determine whether Ethernet is connected and the current Ethernet speed. By default, the PHY chip automatically negotiates the speed and matches according to the speed of the other party. If “bit 10” is 1, the link succeeds, otherwise it fails. “Bit[15:14]” is 00, it is 10M, 01 is 100M, 10 or 11 is 1000M, LED lights are lighted according to different speeds, no link no led light, 10M lights one LED, 100M lights two LEDs, 1000M lights three LEDs.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
mdc	out	1	MDC Clock
mdio	inout	1	MDIO Data
speed	out	2	Speed 00：10M; 01：100M; 10：1000M
link	out	1	Register address
led	out	4	Write request

5.4 10/100/1000M Arbitration

Since “10/100M” transmits data at the time of single edge acquisition, it uses the “MII” bus to transmit, so 4bit data is transmitted in one cycle. When 1000M transmits data, “RGMII” collects data for

both edges, and after “RGMII” is converted to “GMII”, it transmits 8 bits of data in one cycle. Therefore, when “10/100M” transmits data, data conversion is required. For example, when transmitting, a period of 8 bits of data is converted into a period of 4 bits of data, so the data buffer is set, and the read data rate is half of the written data.

5.4.1 TX buffer

”gmii_tx_buffer.v” caches the 8bits to be sent by 10/100M into “eth_data_fifo”, and buffers the data length into “len_fifo”. In the state machine, it is judged whether the data in the length FIFO is greater than 0. This indicates that the data has been cached and then enters the read FIFO data status, reading data every two cycles, this function is not easy to understand, it is best to use ”signaltap” to capture signals to help understand.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
eth_10_100m_en	in	1	10/100M enable signal
link	in	1	Link Signal
gmii_tx_en	in	1	Internal gmii send enable
gmii_txd	in	8	Internal gmii sends data
e10_100_tx_en	out	1	10/100M transmit enable
e10_100_txd	out	8	10 / 100M data transmission

5.4.2 RX buffer

”gmii_rx_buffer.v” caches the received 10/100M data into “eth_data_fifo”, the data are written once in 2 clock cycles. At the same time, the data length is buffered into “len_fifo”, and it is judged in the state machine whether the data in the length FIFO is greater than 0. This indicates that the data has been buffered, and then enters the read FIFO data state, and the data is read once per cycle.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
eth_10_100m_en	in	1	100M enable signal
eth_10m_en	in	1	100M enable signal
link	in	1	Link Signal
gmii_rx_dv	in	1	Internal gmii receive enable
gmii_rxd	in	8	Internal gmii receive data
e10_100_rx_dv	out	1	10/100M receive valid signal
e10_100_rxd	out	8	10/100M receiving data

5.4.3 Arbitration

”gmii_arbi.v” is used for arbitration at three speeds, and the interval for sending is set to 1S.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
speed	in	2	Ethernet speed
link	in	1	Link Signal
gmii_rx_dv	in	1	External gmii reception is valid
gmii_rxd	in	8	External gmii receiving data
gmii_tx_en	in	1	Internal gmii transmit enable
gmii_txd	out	8	Internal gmii transmits data
pack_total_len	out	32	Delay value, each speed is set to 1S
e_rst_n	out	1	Reset MAC signal
e_rx_dv	out	1	Receive valid signal after arbitration
e_rxd	out	8	Receive valid data after arbitration
e_tx_en	out	1	Transmit enable signal after arbitration
e_txd	out	8	Transmit data signal after arbitration

5.5 Other Parts

5.5.1 ICMP Answer

In the project, “icmp_reply.v” implements the “ping” function, first receives the “icmp” data sent by other devices, determines whether the type is a loopback request (ECHO REQUEST), and if so, stores the data in RAM, and calculates the checksum, and judges the checksum. If it is correct, it will enter the sending state and send the data out.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System Clock
rst_n	in	1	Asynchronous reset, low reset
mac_send_end	in	1	Mac transmit end signal
ip_tx_ack	in	1	IP transmit response
icmp_rx_data	in	8	ICMP transmits data
icmp_rx_req	in	1	ICMP transmits the request
icmp_rev_error	in	1	Transmits error signal
upper_layer_data_length	in	16	Upper layer protocol length
icmp_data_req	in	1	Transmit request ICMP data
icmp_tx_ready	out	1	ICMP is ready to send
icmp_tx_data	out	8	ICMP transmits data
icmp_tx_end	out	1	End of ICMP transmission
icmp_tx_req	out	1	ICMP transmits a request

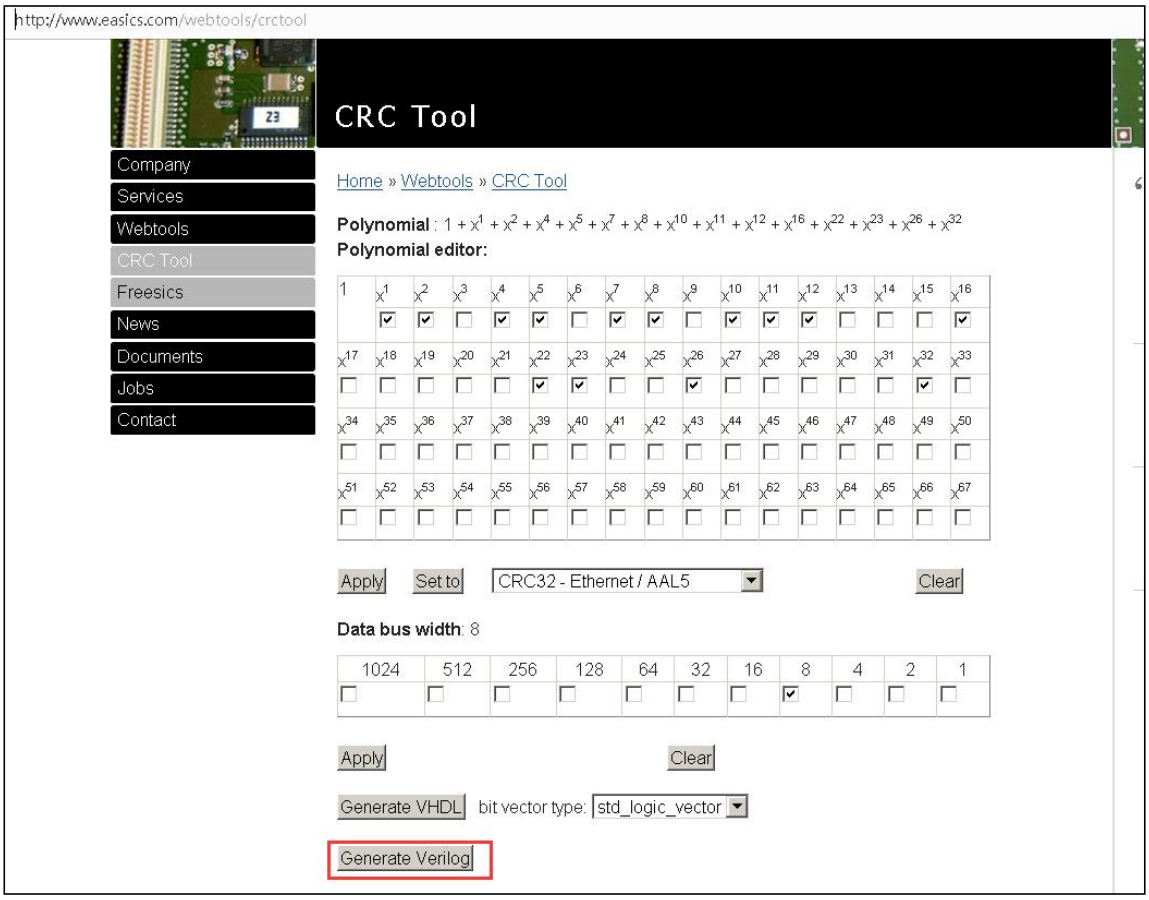
5.5.2 ARP Cache

In the project, “arp_cache.v” is an “arp” cache module that caches the IP addresses and MAC addresses of other devices received. Before sending data, it queries whether the destination address exists. If it does not exist, it sends an ARP request to the destination address, waiting for a response. . In the design file, only one cache space is made, and if necessary, it can be expanded.

Signal Name	Direction	Width(bit)	Description
clk	in	1	System Clock
rst_n	in	1	Asynchronous reset, low reset
arp_found	in	1	ARP received the correct reply
arp_rec_source_ip_addr	in	32	Source IP address received by ARP
arp_rec_source_mac_addr	in	48	Source MAC address received by ARP
destination_ip_addr	in	32	Destination IP address
destination_mac_addr	out	48	Destination MAC address
mac_not_exist	out	1	The MAC address corresponding to the destination address does not exist.

5.5.3 CRC Check Module (crc.v)

The CRC check of an IP packet is calculated at the destination MAC Address and is calculated until the last data of a packet. The verilog algorithm and polynomial of Ethernet's CRC32 can be generated directly at the website below: *http://www.easics.com/webtools/crctool*



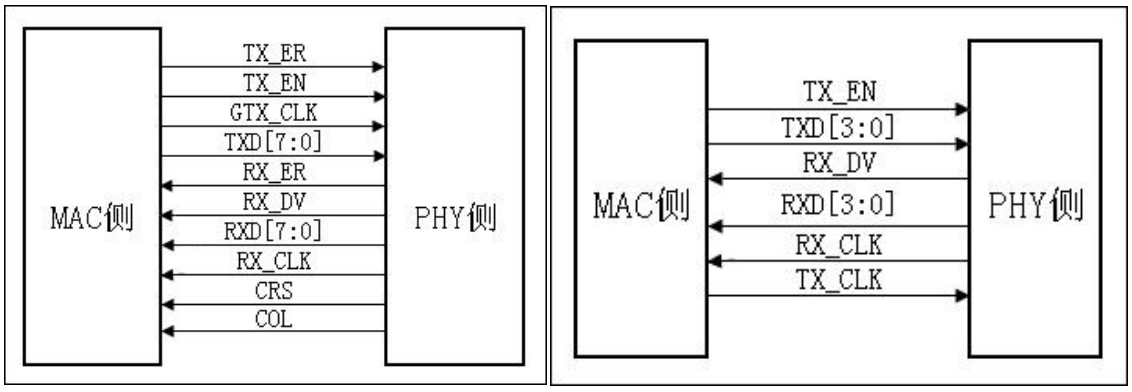
5.5.4 Ethernet test module (mac_test.v)

The test module implements the control of the data stream. First, after the key is pressed, the ARP request signal is sent until the other party responds, and the UDP data state is entered. If the UDP data is found to be valid in the "WAIT" state, the received data is sent out. The loop is sent for 1 second. Before each transmission, it is necessary to check whether the destination IP address can be found in the ARP cache. If not, it sends an ARP request.

Signal Name	Direction	Width(bit)	Description
clk_50m	in	1	System clock
rst_n	in	1	Asynchronous reset, low reset
pack_total_len	in	32	Packet length
push_button	in	1	Key signal
gmii_tx_clk	in	1	GMII Transmit clock
gmii_rx_clk	in	1	GMII receive clock
gmii_rx_dv	in	1	Receive data valid signal
gmii_rxd	in	8	Receive data
gmii_tx_en	out	1	Transmit data valid signal
gmii_txd	out	8	Transmit data

5.5.5 RGMII to GMII module

The “util_gmii_to_rgmii.v” file converts “RGMII” and “GMII” to extract control signals and data signals. The connection to the PHY is the RGMII interface, which is only used in the AX7103 routines.



GMII Interface RGMII Interface

The “RGMII” interface is a simplified version of the “GMII” interface that samples data on both the rising and falling edges of the clock, “TXD[3:0]/RXD[3:0]” are transmitting on the rising edge, and “TXD[7:4]/RXD” are transmitting on the falling edge. 7:4], “TX_EN” transmits both “TX_EN” (rising edge) and “TX_ER” (falling edge). “RX_DV” transmits both “RX_DV” (rising edge) and “RX_ER” (falling edge).

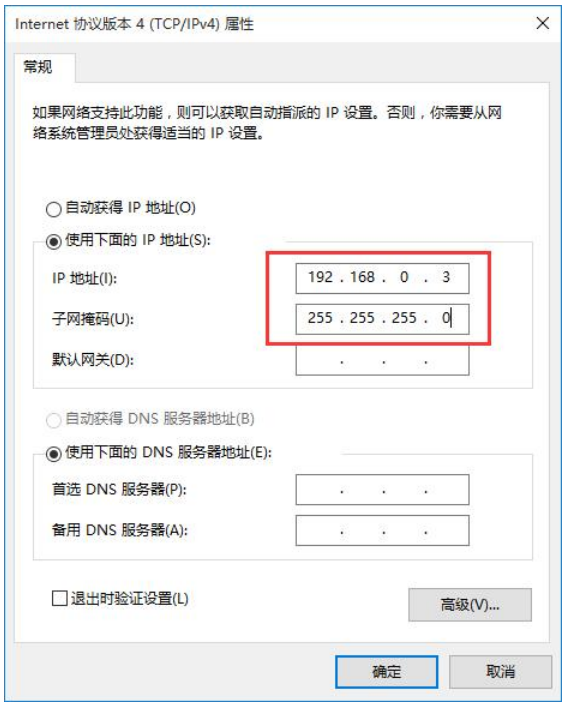
6 Download and Experiment

6.1 Ready to work

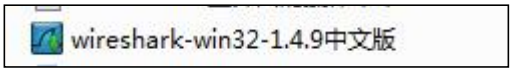
1) First confirm whether your PC's network card is a Gigabit Ethernet card (also supported by the 100M network card). You can click the local connection to view it, and then use the Category 5 or Category 6 network cable to connect the network port of the development board to the network port of the PC.

2) Change the IP address of the PC to “192.168.0.3”. The IP address of the PC needs to be the same as that set in “mac_test.v”, otherwise the network debugging assistant will not receive the UDP packet sent by the development board.

```
.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),
```



3) Wireshark is installed to facilitate the debugging of the user's network communication. Install the network capture tool Wireshark in the directory of "\07_Software Tools and Drivers\Network Debugging Tools\wireshark_cn". We can use this tool to view the PC network port during the experiment. Details of the data sent and the data received.

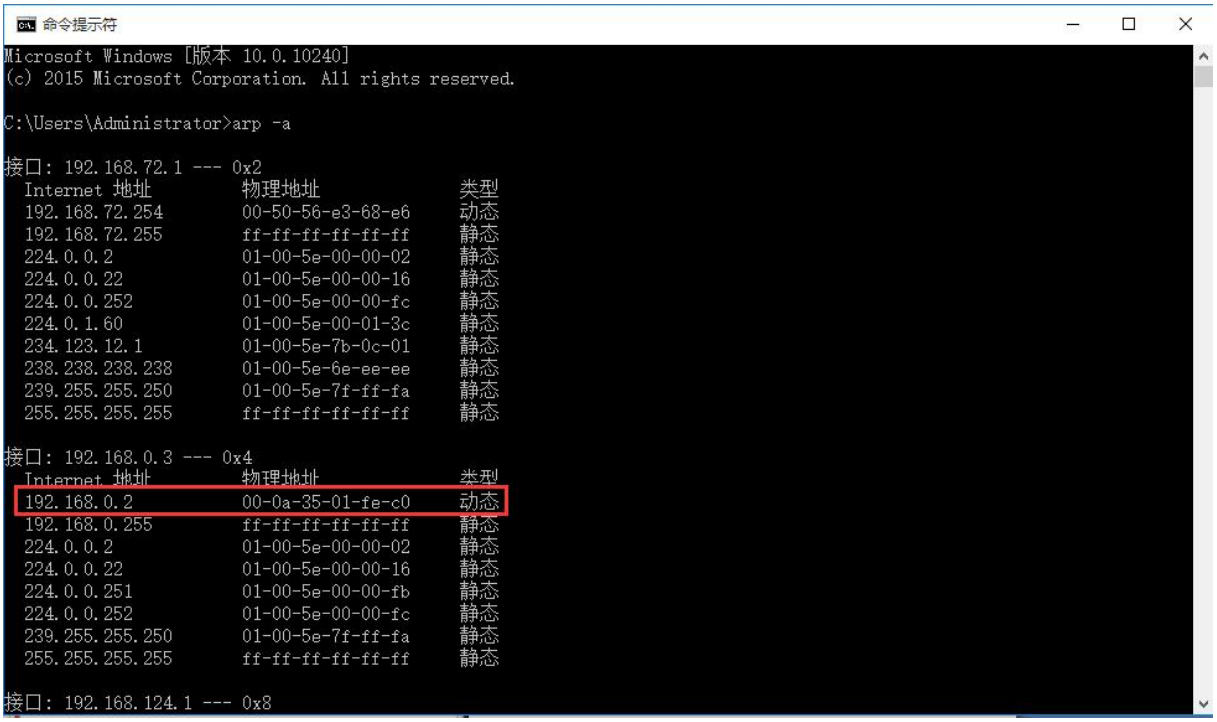


6.2 Ethernet communication test

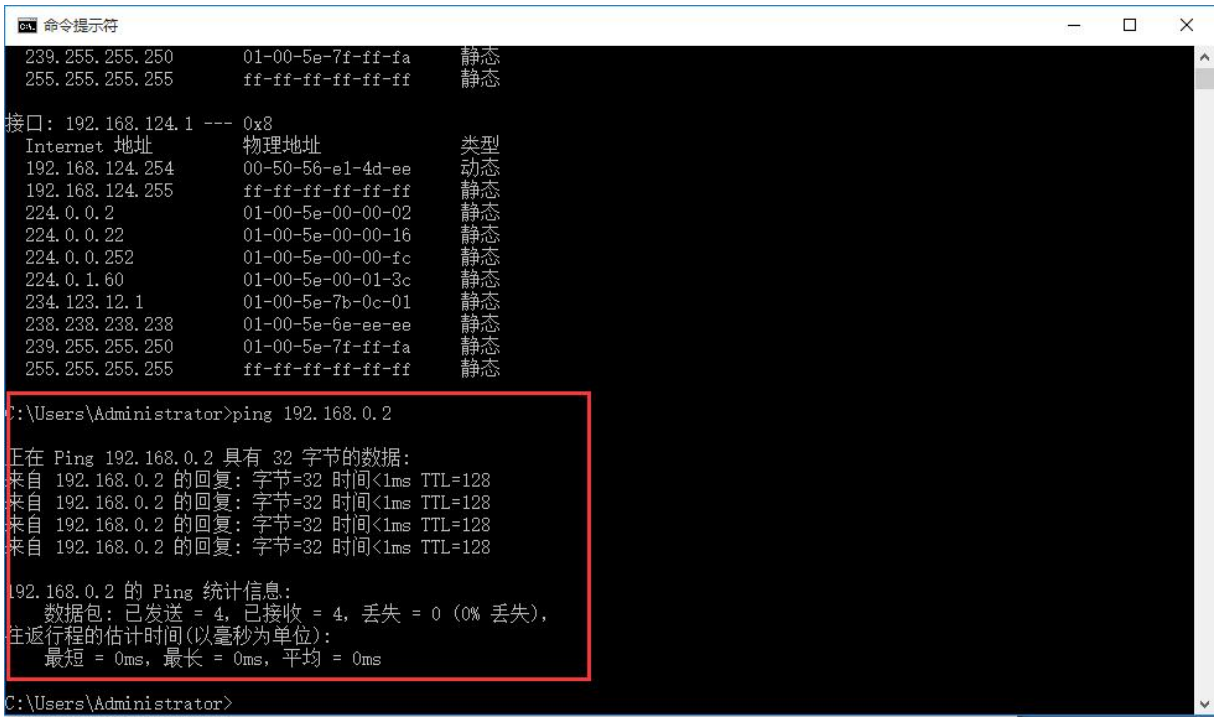
1) Write the “ethernet_test.bit” file to the FPGA, (if you need to solidify to the “ethernet_test.bin” file in FLASH), wait for two or three seconds. On the AX7325 development board three LED lights indicate that it is a Gigabit network card, two lights indicate that it is a 100M network card, if the LED is not lit, need check the network port connection.

2) Open the “CMD” window with administrator privileges, enter “arp –a” to view the ARP binding results, and you can see that the IP address and “MAC”address of the development board have

been cached.

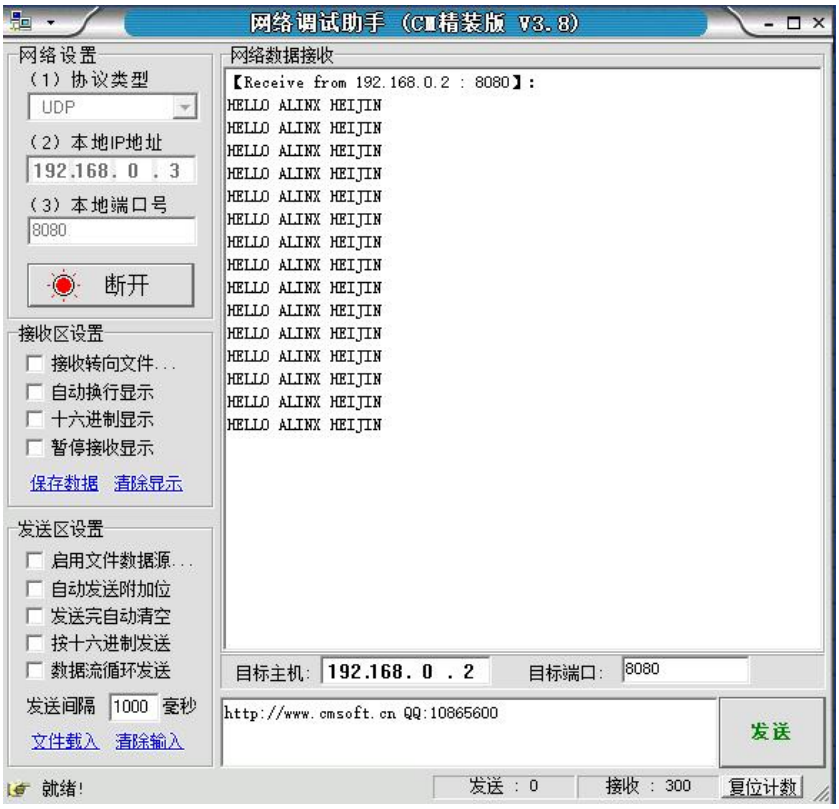


3) In the CMD window, enter “ping 192.168.0.2” to check whether the PC and the development board are “pinged”.



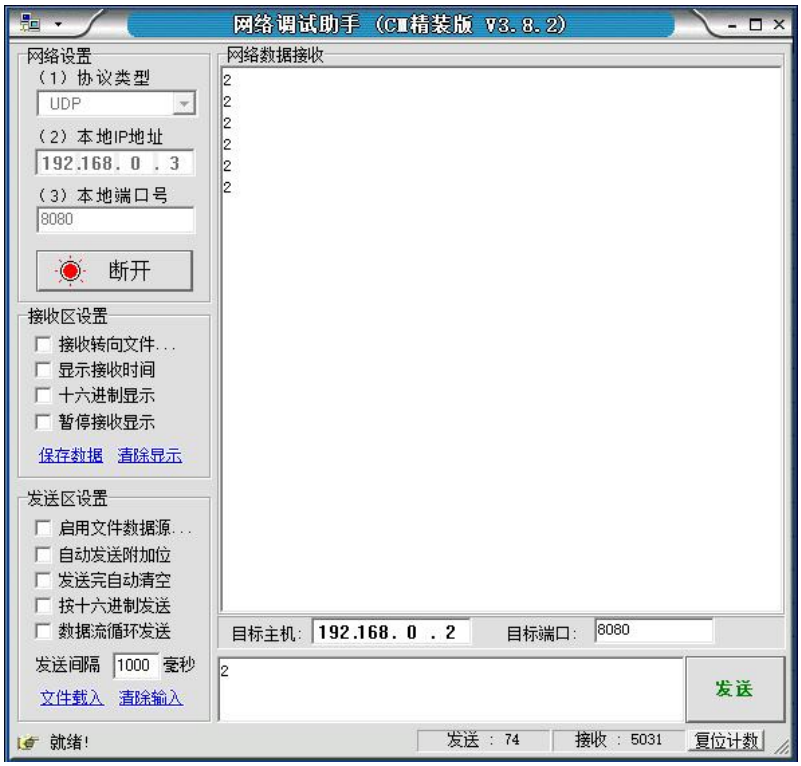
4) Open the Network Debugging Assistant in the “\07_Software Tools and Drivers\Network Debugging Tools\NetAssist” directory and set the parameters as follows, then press the connect button (The local IP address here is the IP address of the PC, and the local port needs to be the same as the one in the FPGA program is 8080).

Click the "Connect" button, then the network data receiving window will display the Ethernet packet "HELLO ALINX HEIJIN" sent by the FPGA to the PC. The IP address of the target host needs to be the same as the IP address in the FPGA program. The target port number also needs to be the same as the FPGA program (8080). The network shown below shows:

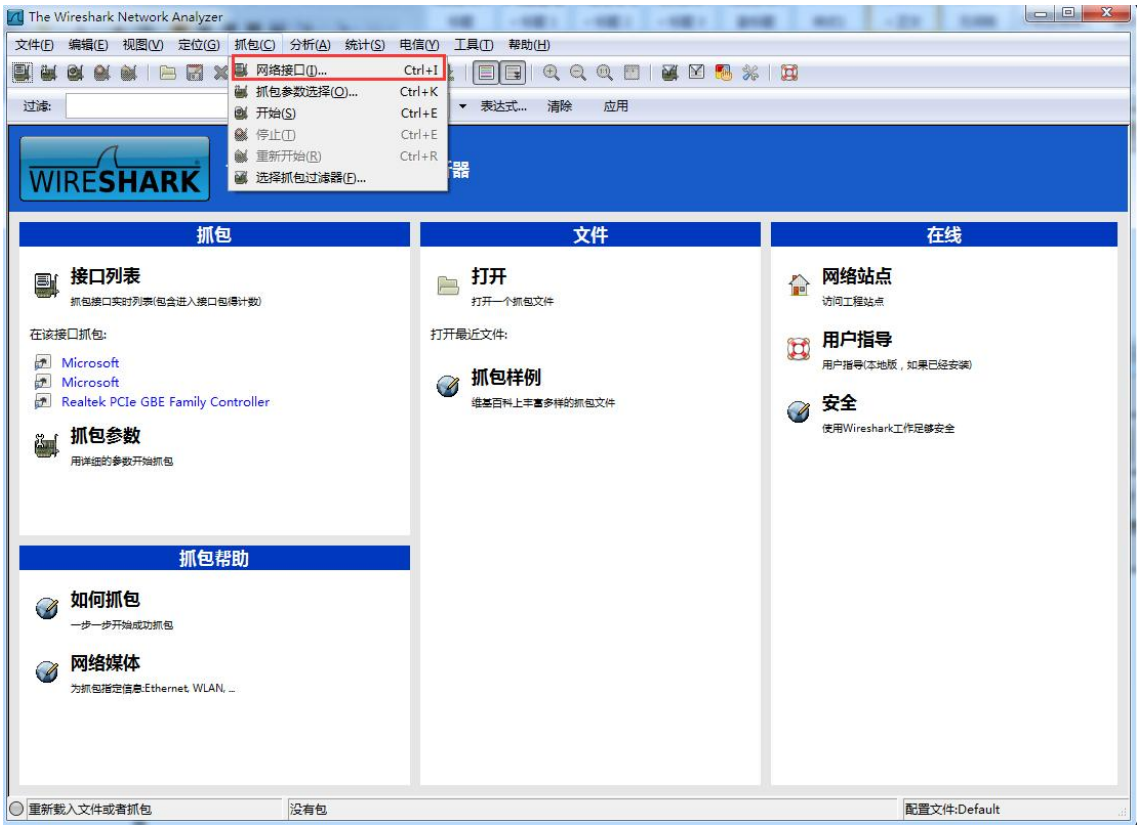


5) Then send a large string of characters in the send window of the network debugging assistant. In the data receiving window of the network, we can see that the data returned from the FPGA also becomes the string just sent.

It can also send fewer characters, less than 46 bytes, the FPGA program will automatically replenish to 46 bytes, as shown below:

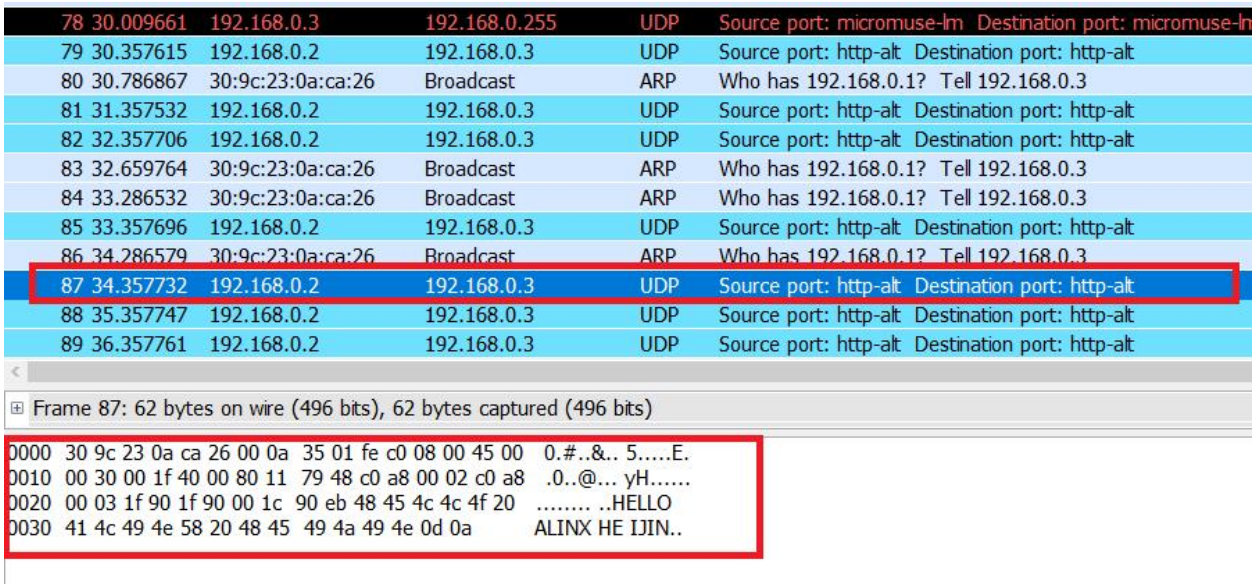


6) This step is optional for the user. If you want to view more information about the packet transmission, you can use the network capture tool “Wireshark” to view the network data received and sent by the PC's network card, and open the installed “ wireshark” packet capture tool. , click on the menu capture package -> network interface.



Select the PC's Gigabit LAN in the Capture Interface window that pops up, and press the Start button to start capturing packets.

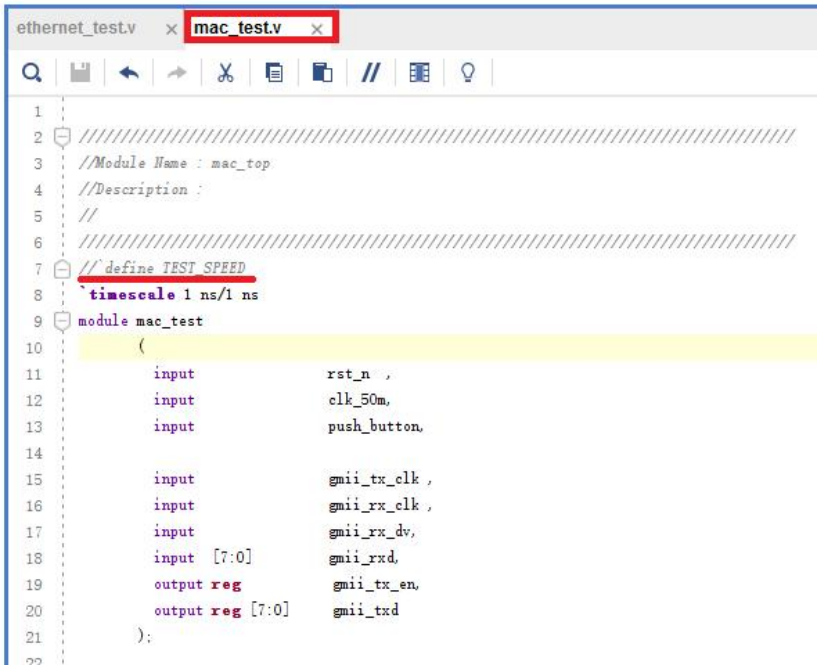
In the “wireshark” capture window, we can see the packet sent by the development board (192.168.0.2) to the PC network port (192.168.0.2). Here, the destination MAC, source MAC, IP header and UDP packet of the packet will be displayed, as shown in the following figure, the development board capture package display window



6.3 Ethernet speed test

We can measure the speed of data transmission in the Ethernet part through the packet capture tool “wireshark”, because the speed of the Gigabit Ethernet ideal mode network can reach 1Gbps, but actually because each packet has a packet header, CRC and other non-data characters, and there is a gap between the each packets. Generally, the data transmission speed of Gigabit Ethernet is up to 950Mbps.Transmission is symmetrical transmission, that is, both uplink and downlink can reach 950Mbps. Here, because the capture “wireshark” can only count the received data packets, but not the transmit data, So here is just testing the speed at which the FPGA transmits data to the computer.

1) modify the program, modify the code in the “mac_test.v” in the project, as shown below

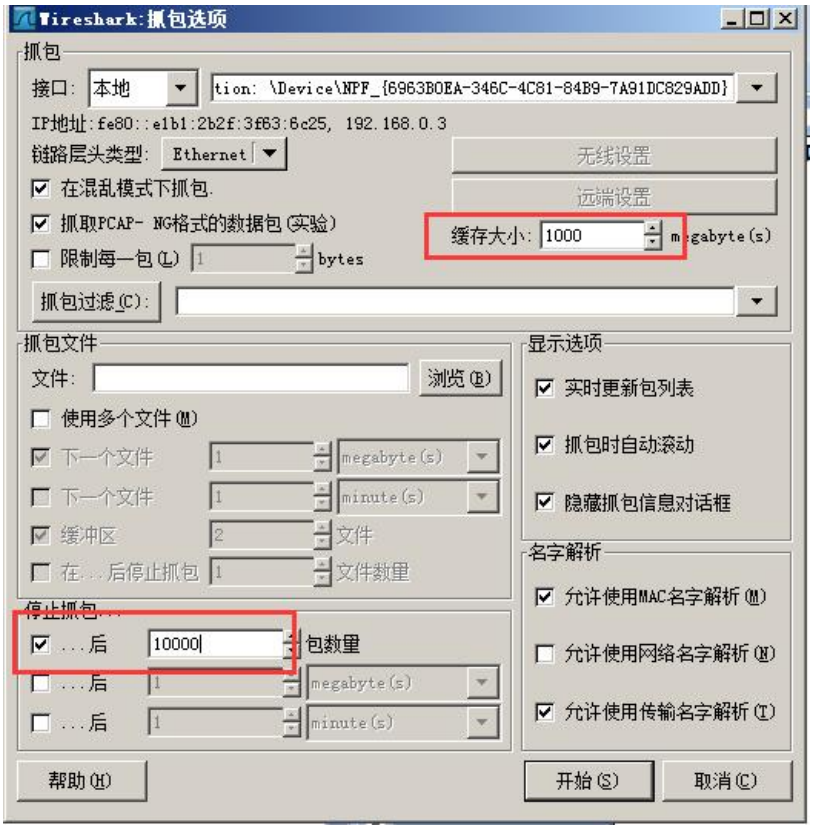


Remove the red mark "//" in the above picture, as follows:

```
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 ;
```

2) After the compilation is completed, the test program is downloaded. After the download is completed, the FPGA network will be dynamically bound at this time. For the IP settings of the PC, please refer to Section 5.2 (if it has been set to ignore).

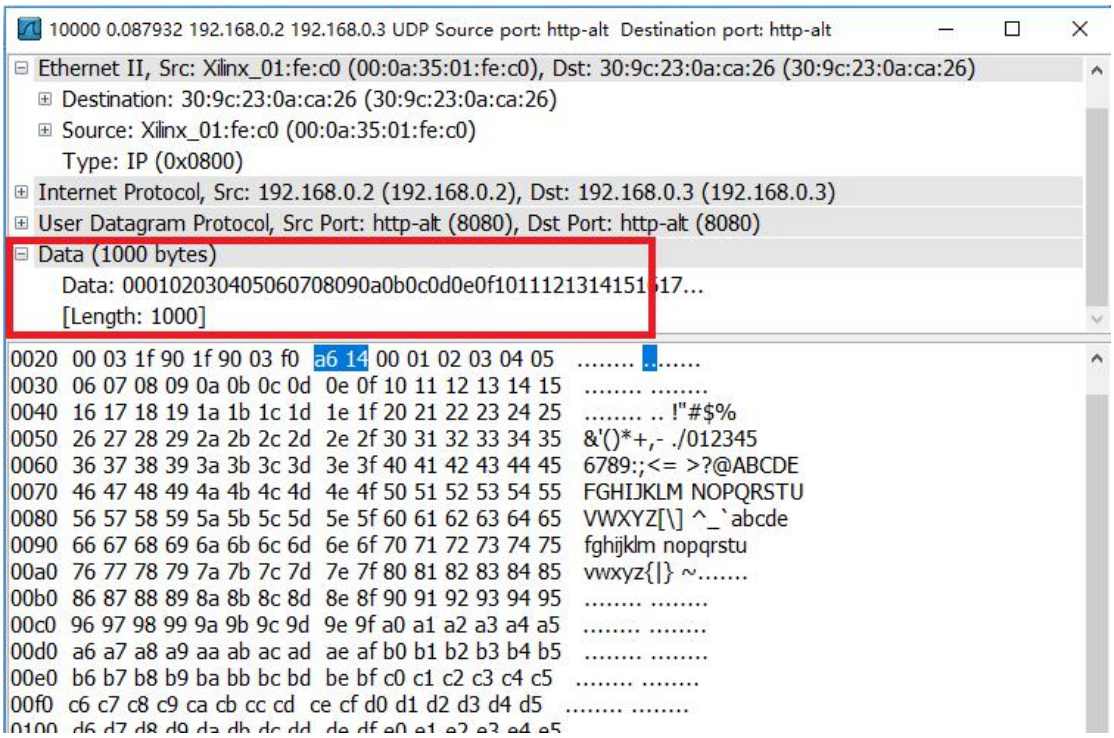
Open “wireshark” and click on the menu "Capture Package" -> "Capture Parameter Selection". Set the buffer size to 1000megabyte in the capture option and stop capturing packets after 10000 packets. (Need to pay attention here: if not set, “wireshark” can not handle the software because it catches too many packages.)



After starting to capture the packet, wait for 10000 packets to be received and stop capturing the packet, which takes “0.087932” seconds.

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

We can see that the content of each packet is 00->FF data, a total of three 00->FF, one 00->e7 data.1000 valid data totally.

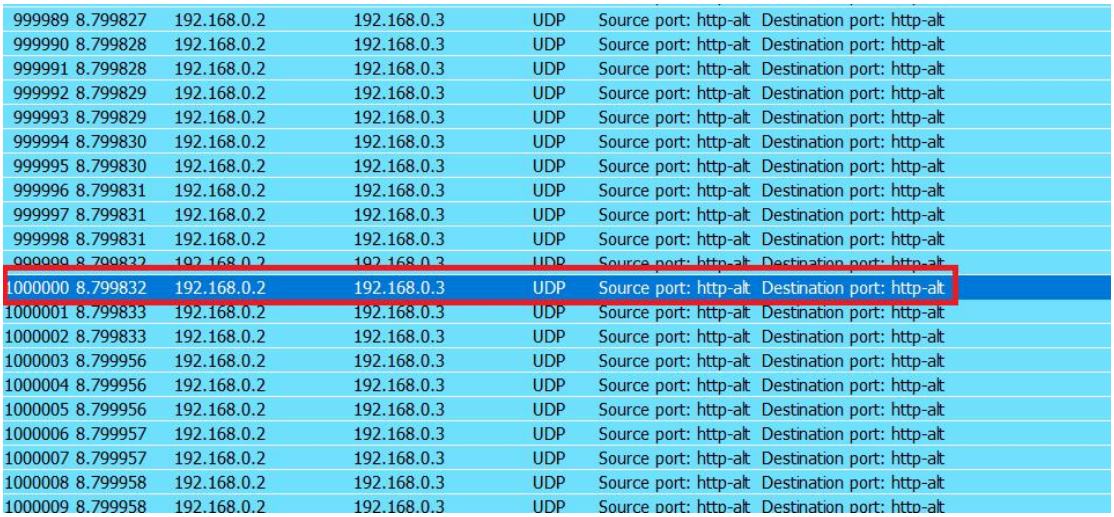


Here we see that the software received 10,000 packets with 0.087932 seconds, and the valid data contained in each packet is 1000 bytes (8 bits). So we can calculate the transmission speed of Ethernet:

Transmission Speed = 1000* 10000 * 8 / 0.087932 = 910Mbps

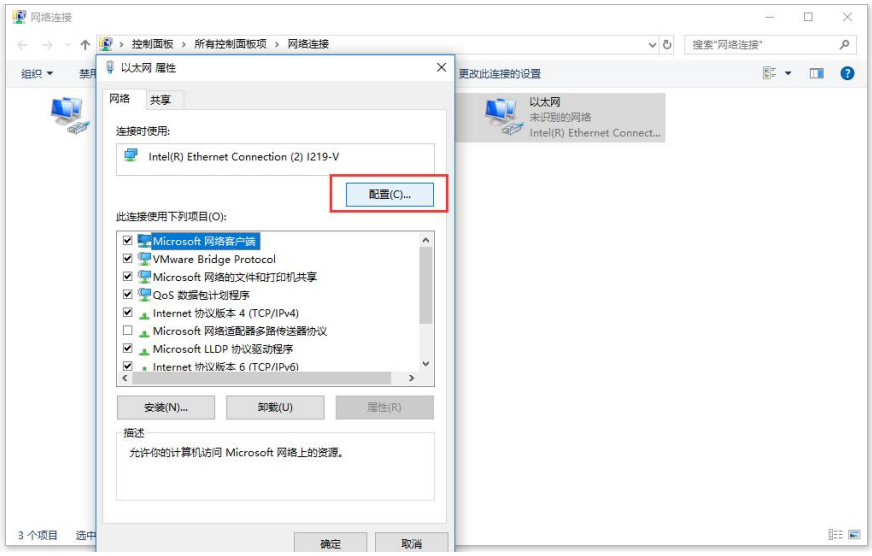
Here, the speed of network transmission is also related to the packet length. The longer the packet length, the higher the efficiency, and the faster the data transmission speed. The shorter the data packet, the lower the efficiency and the slower the data transmission speed. I think everyone should understand this truth!

The following is the time of 1000000 packets is 8.799832 seconds, you can calculate the speed according to the above aspects, the speed is very stable.

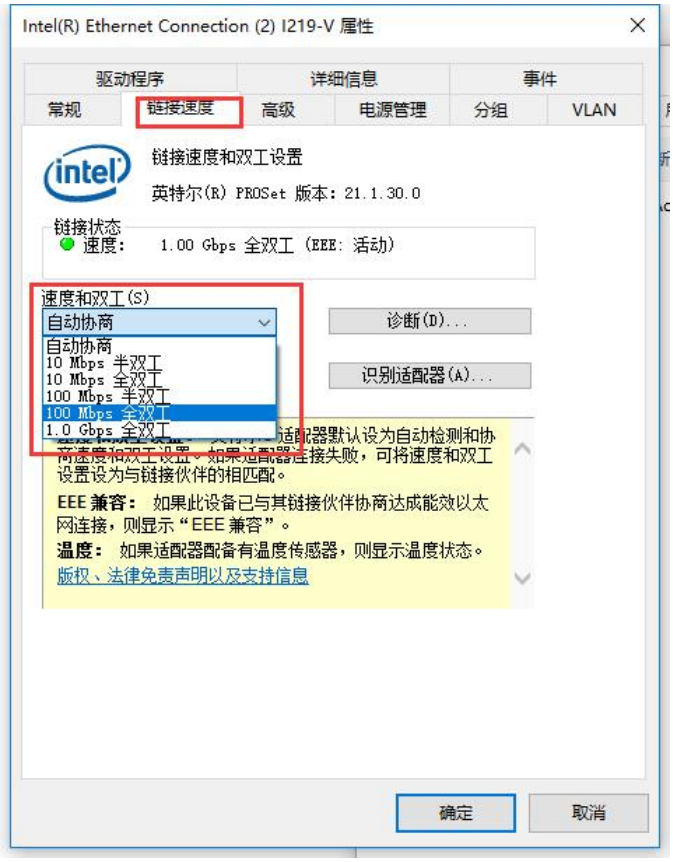


6.4 Ethernet 100M/1000M test

To change the speed of the computer's network card, first open the network connection, find the local network card, right click on the properties to pop up the following window, and then click Configure.



Find the link speed in the pop-up window, select 100Mbps full duplex, click OK, wait a few seconds, you will find that the LED light will change, then press the previous steps to test.



So far, our Ethernet data communication is all over. To realize Ethernet data communication, we need to understand the format and communication protocol of Ethernet data packets. Also need to understand the timing of the GMII, RGMII, MII, and RMII interfaces between the MAC and the PHY. Gigabit Ethernet data transmission speed is very fast, and it is full-duplex transmission, and UDP communication data speed can reach 900Mbps or more, which is very suitable for high-speed data transmission occasions, such as video image transmission, high-speed data acquisition and so on.