

# 基于FPGA的UDP实现（包含源工程文件）

基于FPGA的以太网相关文章导航，[点击查看](#)。

## 1、概括

前文通过 **FPGA** 实现了**ARP**和**ICMP**协议，ARP协议一般用来获取目的IP地址主机的MAC地址，ICMP通过回显请求和回显应答来判断 **以太网** 链路是否通畅，这两个协议都不是用来传输用户数据的。如果用户需要向PC端传输大量数据，那么就必须使用TCP或者UDP协议了。

网上关于UDP和TCP的优缺点对比其实很多，可以自行搜索，本文简要概括一下优缺点。

TCP优点是稳定，接收端接收到TCP数据报文后会回复发送端，如果接收的报文有误，发送端会把错误的报文重新发送一遍。而且TCP本来就有握手机制，所以数据的传输会更可靠。正是由于握手机制，导致实现的TCP协议的逻辑比较复杂，传输速度也不会很高，还需要更多存储资源取存储已经发送的数据，直到收到该数据传输无误后才能丢弃。因此FPGA一般不会采用该协议进行大量数据的传输（当然如果通过Verilog HDL实现可靠的TCP协议，那还是很有用的，毕竟这的代码很贵）。

UDP优点是协议简单，没有握手机制，传输数据的速度就很快，这对于FPGA传输图像数据之类的设计比较实用。由于UDP没有握手机制，可靠性相比TCP就会低很多，有得必有失嘛。

因此，FPGA一般通过UDP协议向PC端发送大量数据，所以本文通过FPGA实现UDP协议。

## 2、UDP协议讲解

UDP协议的框图如下所示，与前文的 **ICMP**协议 构成类似，UDP协议数据报文位于IP的数据段，IP首部只有协议类型与ICMP协议类型参数不一致，ICMP的IP协议类型编号为1，UDP的IP协议类型编号为17。

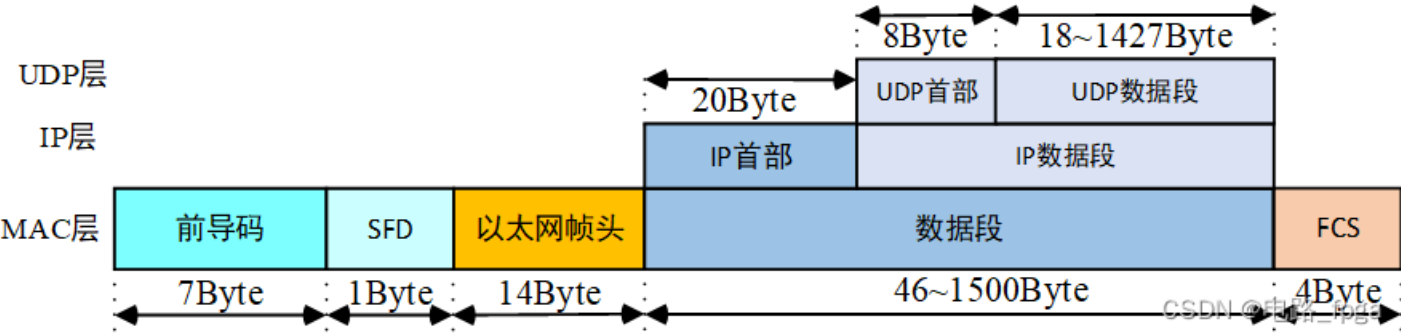


图1 UDP协议框图

前导码、帧起始符、以太网帧头、IP首部、FCS校验在前文讲解ARP协议和ICMP协议的时候都详细讲解过，所以本文就不再赘述了。

UDP的首部组成如下所示，包括源UDP源端口地址、UDP目的端口地址、UDP长度、UDP校验码。

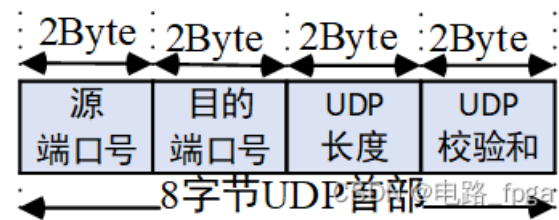


图2 UDP首部组成

源端口号：2个字节的发送端端口号，用于区分不同的发送端口。

目的端口号：2个字节的接收端端口号。

UDP长度：UDP首部和数据段的长度，单位字节，对于接收方来说该长度其实作用不大，因为UDP数据段的长度可以通过IP首部的总长度和IP首部长度计算出来。

UDP校验和：计算方式与IP首部校验和一致，需要对UDP伪首部、UDP首部、UDP数据进行校验。伪首部包括源IP地址、目的IP地址、协议类型、UDP长度。

这种校验方式其实对于FPGA来说很麻烦，因为校验码需要在数据之前发送，而计算校验码有需要得到数据，就意味着如果想要计算校验码，就必须使用存储资源把待发送的数据存起来，计算出校验码以后，才开始传输数据。比较友好的是该校验码可以直接置零处理（如果不做校验，该值必须为0，否则校验失败的数据报文会被直接丢弃）。不校验数据的UDP协议变得特别简单。

UDP协议的数据组成如下所示：

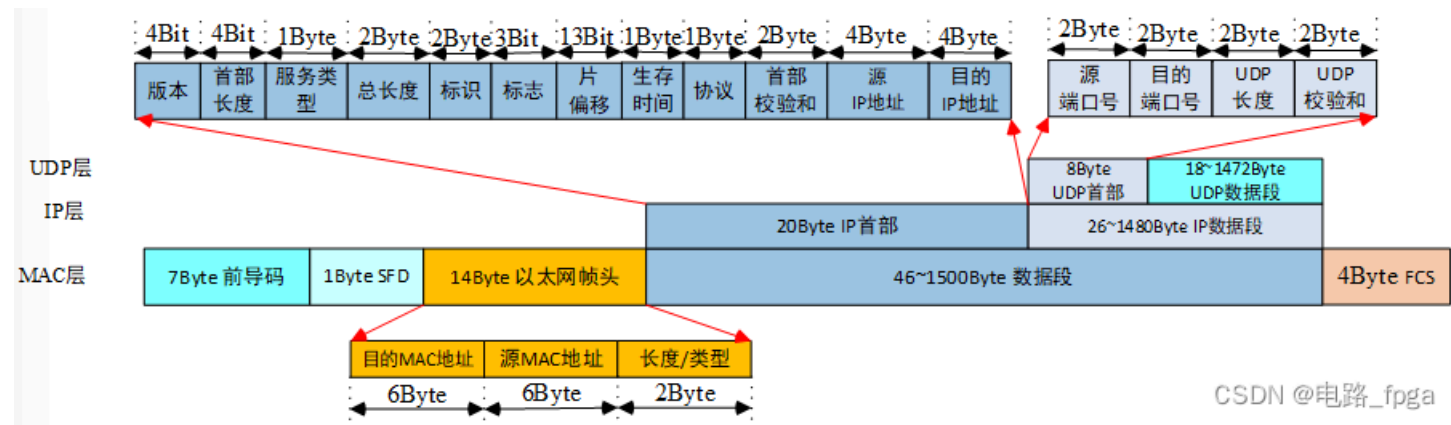


图3 以太网UDP协议帧组成

以太网的协议组成就介绍这么多了，最后注意在发送数据时，数据段必须大于等于18字节，少于18字节数据时，应补零凑齐18字节数据发送。

### 3、UDP顶层模块

UDP的设计与前文的ARP、ICMP模块设计差不多，UDP顶层模块如下图所示，包括UDP接收模块udp\_rx、UDP接收的CRC校验模块、UDP的发送模块udp\_tx、UDP发送的CRC校验模块。

UDP接收模块内部没有做IP首部校验，只做了CRC校验模块，在加上FPGA逻辑判断，基本上都能判断对错了，最后把接收的数据和数据个数输出。发送模块检测到开始发送信号后，开始发送信号，当发送到数据段之后，把数据请求信号拉高，从外部输入需要发送的数据流。

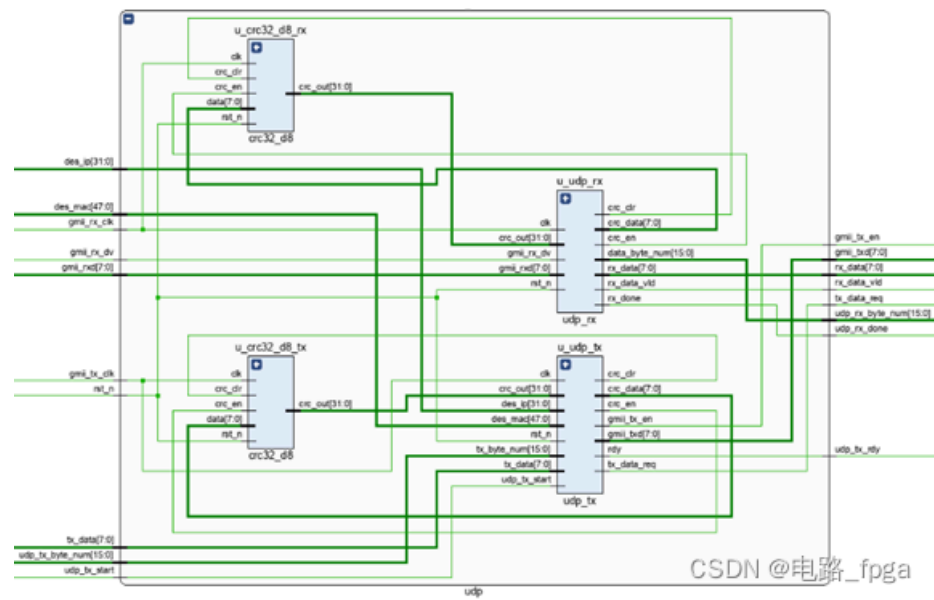


图4 UDP顶层模块

UDP顶层模块的参考代码如下所示：

```
1 //例化udP接收模块；
2 udp_rx #(
3     .BOARD_MAC      ( BOARD_MAC      ),//开发板MAC地址 00-11-22-33-44-55;
4     .BOARD_IP        ( BOARD_IP        ) //开发板IP地址 192.168.1.10;
5 )
6 u_udp_rx (
7     .clk              ( gmii_rx_clk      ),//时钟信号；
8     .rst_n            ( rst_n            ),//复位信号，低电平有效；
9     .gmii_rx_dv       ( gmii_rx_dv      ),//GMII输入数据有效信号；
```

```

10     .gmii_rxd      ( gmii_rxd      ),//GMII输入数据;
11     .crc_out       ( rx_crc_out     ),//CRC校验模块输出的数据;
12     .rx_done       ( udp_rx_done    ),//UDP接收完成信号, 高电平有效;
13     .rx_data_vld   ( rx_data_vld    ),//以太网接收到有效数据指示信号;
14     .rx_data       ( rx_data        ),//以太网接收数据。
15     .data_byte_num ( udp_rx_byte_num ),//以太网接收的有效数据字节数 单位:byte
16     .des_port      (                 ),//UDP接收的目的端口号;
17     .source_port   (                 ),//UDP接收到的源端口号;
18     .crc_data      ( rx_crc_data     ),//需要CRC模块校验的数据;
19     .crc_en        ( rx_crc_en       ),//CRC开始校验使能;
20     .crc_clr       ( rx_crc_clr      ) //CRC数据复位信号;
21 );
22
23 //例化接收数据时需要的CRC校验模块;
24 crc32_d8 u_crc32_d8_rx (
25     .clk      ( gmii_rx_clk ) ,//时钟信号;
26     .rst_n    ( rst_n       ) ,//复位信号, 低电平有效;
27     .data     ( rx_crc_data ) ,//需要CRC模块校验的数据;
28     .crc_en   ( rx_crc_en   ) ,//CRC开始校验使能;
29     .crc_clr  ( rx_crc_clr  ) ,//CRC数据复位信号;
30     .crc_out  ( rx_crc_out  ) //CRC校验模块输出的数据;
31 );
32
33 //例化UDP发送模块;
34 udp_tx #(
35     .BOARD_MAC      ( BOARD_MAC      ),//开发板MAC地址 00-11-22-33-44-55;
36     .BOARD_IP       ( BOARD_IP       ),//开发板IP地址 192.168.1.10;
37     .DES_MAC        ( DES_MAC        ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
38     .DES_IP         ( DES_IP         ),//目的IP地址 192.168.1.102;
39     .BOARD_PORT     ( BOARD_PORT     ),//板子的UDP端口号;
40     .DES_PORT       ( DES_PORT       ),//源端口号;
41     .ETH_TYPE       ( ETH_TYPE       ) //以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
42 )
43 u_udp_tx (
44     .clk      ( gmii_tx_clk      ),//时钟信号;
45     .rst_n    ( rst_n            ),//复位信号, 低电平有效;
46     .udp_tx_start ( udp_tx_start  ),//UDP发送使能信号;
47     .tx_byte_num ( udp_tx_byte_num ),//UDP数据段需要发送的数据。
48     .des_mac    ( des_mac        ),//发送的目标MAC地址;
49     .des_ip     ( des_ip         ),//发送的目标IP地址;
50

```

```

51     .crc_out      ( tx_crc_out      ),//CRC校验数据;
52     .crc_en       ( tx_crc_en       ),//CRC开始校验使能;
53     .crc_clr      ( tx_crc_clr      ),//CRC数据复位信号;
54     .crc_data     ( tx_crc_data     ),//输出给CRC校验模块进行计算的数据;
55     .tx_data_req   ( tx_data_req     ),//需要发送数据请求信号;
56     .tx_data      ( tx_data         ),//需要发送的数据;
57     .gmii_tx_en    ( gmii_tx_en     ),//GMII输出数据有效信号;
58     .gmii_txd     ( gmii_txd        ),//GMII输出数据;
59     .rdy          ( udp_tx_rdy      ) //模块忙闲指示信号, 高电平表示该模块处于空闲状态;
60 );
61
62 //例化发送数据时需要的CRC校验模块:
63 crc32_d8 u_crc32_d8_tx (
64     .clk          ( gmii_tx_clk     ),//时钟信号;
65     .rst_n        ( rst_n           ),//复位信号, 低电平有效;
66     .data         ( tx_crc_data     ),//需要CRC模块校验的数据;
67     .crc_en       ( tx_crc_en       ),//CRC开始校验使能;
68     .crc_clr      ( tx_crc_clr      ),//CRC数据复位信号;
69     .crc_out      ( tx_crc_out      ) //CRC校验模块输出的数据;
70 );

```



对应的TestBench文件如下所示:

```

1  `timescale 1 ns/1 ns
2  module test();
3      localparam CYCLE      = 8          ;//系统时钟周期, 单位ns, 默认8ns;
4      localparam RST_TIME   = 10         ;//系统复位持续时间, 默认10个系统时钟周期;
5      localparam STOP_TIME  = 1000       ;//仿真运行时间, 复位完成后运行1000个系统时钟后停止;
6      localparam BOARD_MAC  = 48'h00_11_22_33_44_55 ;
7      localparam BOARD_IP   = {8'd192,8'd168,8'd1,8'd10} ;
8      localparam BOARD_PORT = 16'd1234   ;//开发板的UDP端口号;
9      localparam DES_PORT   = 16'd5678   ;//UDP目的端口号;
10     localparam DES_MAC    = 48'h23_45_67_89_0a_bc ;
11     localparam DES_IP     = {8'd192,8'd168,8'd1,8'd23} ;
12     localparam ETH_TYPE    = 16'h0800   ;//以太网帧类型 IP
--

```

```

13
14     reg                clk                ;//系统时钟，默认100MHz;
15     reg                rst_n              ;//系统复位，默认低电平有效;
16     reg                [7 : 0]           tx_data                ;
17     reg                udp_tx_start       ;
18     reg                [15 : 0]          udp_tx_byte_num        ;
19
20     wire                [7 : 0]           gmii_rxd              ;
21     wire                gmii_rx_dv        ;
22     wire                gmii_tx_en        ;
23     wire                [7 : 0]           gmii_txd              ;
24     wire                udp_rx_done        ;
25     wire                [15 : 0]          udp_rx_byte_num        ;
26     wire                udp_tx_rdy        ;
27     wire                tx_data_req       ;
28     wire                rx_data_vld       ;
29     wire                [7 : 0]           rx_data                ;
30
31     assign gmii_rx_dv = gmii_tx_en;
32     assign gmii_rxd = gmii_txd;
33
34     udp #(
35         .BOARD_MAC    ( BOARD_MAC ),
36         .BOARD_IP      ( BOARD_IP ),
37         .DES_MAC       ( DES_MAC ),
38         .DES_IP        ( DES_IP ),
39         .BOARD_PORT    ( BOARD_PORT ),//板子的UDP端口号;
40         .DES_PORT      ( DES_PORT ),//源端口号;
41         .ETH_TYPE      ( ETH_TYPE )
42     )
43     u_udp (
44         .rst_n          ( rst_n          ),
45         .gmii_rx_clk    ( clk            ),
46         .gmii_rx_dv     ( gmii_rx_dv     ),
47         .gmii_rxd       ( gmii_rxd       ),
48         .gmii_tx_clk    ( clk            ),
49         .udp_tx_start   ( udp_tx_start   ),
50         .udp_tx_byte_num ( udp_tx_byte_num ),
51         .des_mac        ( BOARD_MAC      ),
52         .des_ip         ( BOARD_IP       ),
53

```

```

54     .gmii_tx_en      ( gmii_tx_en      ),
55     .gmii_txd       ( gmii_txd       ),
56     .udp_rx_done    ( udp_rx_done     ),
57     .udp_rx_byte_num ( udp_rx_byte_num ),
58     .udp_tx_rdy     ( udp_tx_rdy     ),
59     .rx_data        ( rx_data         ),
60     .rx_data_vld    ( rx_data_vld     ),
61     .tx_data_req    ( tx_data_req     ),
62     .tx_data        ( tx_data         )
63 );
64
65 //生成周期为CYCLE数值的系统时钟;
66 initial begin
67     clk = 0;
68     forever #(CYCLE/2) clk = ~clk;
69 end
70
71 //生成复位信号;
72 initial begin
73     #1;udp_tx_start = 0; udp_tx_byte_num = 19;tx_data = 0;
74     rst_n = 1;
75     #2;
76     rst_n = 0;//开始时复位10个时钟;
77     #(RST_TIME*CYCLE);
78     rst_n = 1;
79     #(20*CYCLE);
80     repeat(3)begin
81         udp_tx_start = 1'b1;
82         udp_tx_byte_num = {$random} % 64;//只产生64以内随机数，便于测试，不把数据报发的太长了;
83         #(CYCLE);
84         udp_tx_start = 1'b0;
85         #(CYCLE);
86         @(posedge udp_tx_rdy);
87         #(100*CYCLE);
88     end
89     #(20*CYCLE);
90     $stop;//停止仿真;
91 end
92
93 always@(posedge clk)begin
94

```

```

95     if(tx_data_req)begin//产生0~255随机数作为测试;
96         tx_data <= {$random} % 256;
97     end
98 end

endmodule

```



## 4、UDP接收模块

UDP接收模块与前文的ICMP接收模块的设计类似，都可以采用状态机嵌套一个计数器进行实现，状态机对应的状态转换图如下所示。

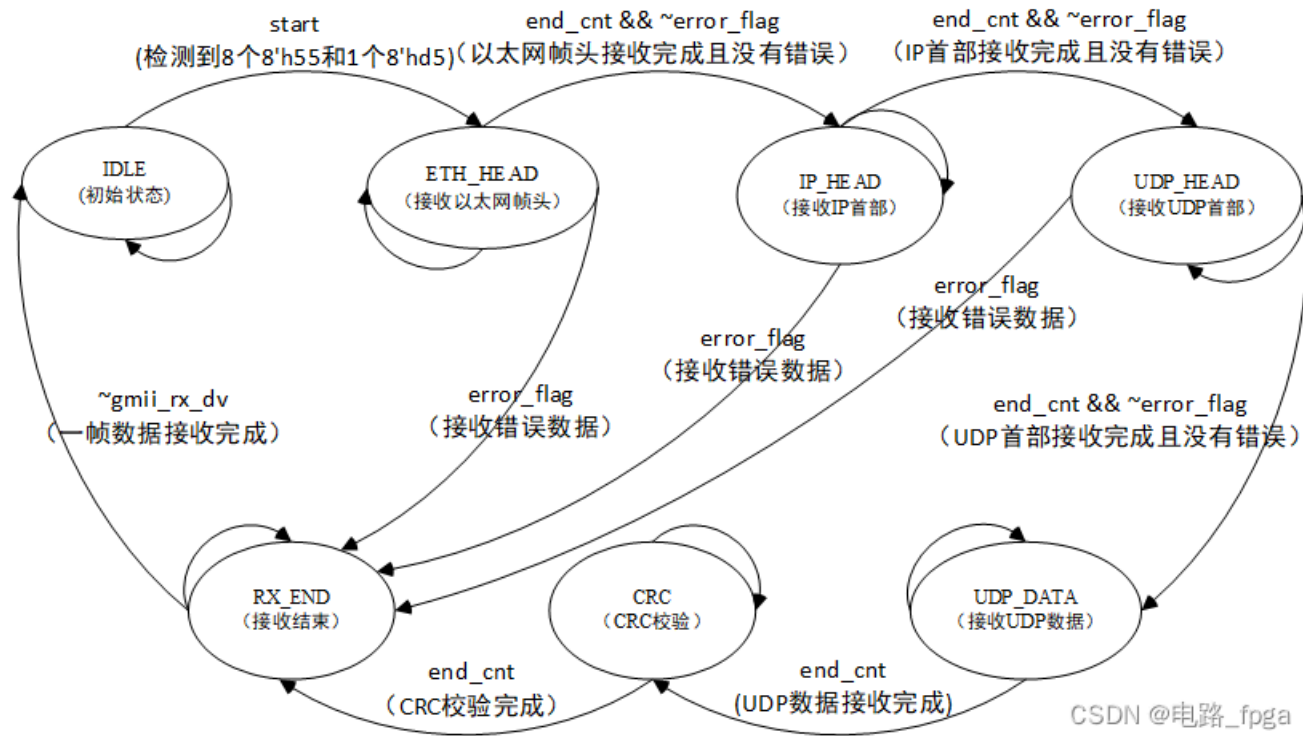


图5 状态转换图

需要注意判断UDP首部的目的端口地址是不是开发板的端口地址，其余部分与ICMP的接收模块差不多，不在赘述了。



该模块对应的代码如下所示：

```
1 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
2 always@(posedge clk)begin
3     if(!rst_n)begin
4         state_c <= IDLE;
5     end
6     else begin
7         state_c <= state_n;
8     end
9 end
10
11 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
12 always@(*)begin
13     case(state_c)
14         IDLE:begin
15             if(start)begin//检测到前导码和SFD后跳转到接收以太网帧头数据的状态。
16                 state_n = ETH_HEAD;
17             end
18             else begin
19                 state_n = state_c;
20             end
21         end
22         ETH_HEAD:begin
23             if(error_flag)begin//在接收以太网帧头过程中检测到错误。
24                 state_n = RX_END;
25             end
26             else if(end_cnt)begin//接收完以太网帧头数据，且没有出现错误，则继续接收IP协议数据。
27                 state_n = IP_HEAD;
28             end
29             else begin
30                 state_n = state_c;
31             end
32         end
33         IP_HEAD:begin
34             if(error_flag)begin//在接收IP帧头过程中检测到错误。
35                 state_n = RX_END;
36             end
37             else if(end_cnt)begin//接收完IP帧头数据，且没有出现错误，则继续接收UDP协议数据。
38                 state_n = UDP_HEAD;
39             end
40         end
41     endcase
42 end
```

```

39         end
40     else begin
41         state_n = state_c;
42     end
43 end
44 UDP_HEAD:begin
45     if(error_flag)begin//在接收UDP协议帧头过程中检测到错误。
46         state_n = RX_END;
47     end
48     else if(end_cnt)begin//接收完以UDP帧头数据，且没有出现错误，则继续接收UDP数据。
49         state_n = UDP_DATA;
50     end
51     else begin
52         state_n = state_c;
53     end
54 end
55 UDP_DATA:begin
56     if(error_flag)begin//在接收UDP协议数据过程中检测到错误。
57         state_n = RX_END;
58     end
59     else if(end_cnt)begin//接收完UDP协议数据且未检测到数据错误。
60         state_n = CRC;
61     end
62     else begin
63         state_n = state_c;
64     end
65 end
66 CRC:begin
67     if(end_cnt)begin//接收完CRC校验数据。
68         state_n = RX_END;
69     end
70     else begin
71         state_n = state_c;
72     end
73 end
74 RX_END:begin
75     if(~gmii_rx_dv)begin//检测到数据线上数据无效。
76         state_n = IDLE;
77     end
78     else begin
79

```

```

80             state_n = state_c;
81         end
82     end
83     default:begin
84         state_n = IDLE;
85     end
86 endcase
87 end
88
89 //将输入数据保存6个时钟周期，用于检测前导码和SFD。
90 //注意后文的state_c与gmii_rxd_r[0]对齐。
91 always@(posedge clk)begin
92     gmii_rxd_r[6] <= gmii_rxd_r[5];
93     gmii_rxd_r[5] <= gmii_rxd_r[4];
94     gmii_rxd_r[4] <= gmii_rxd_r[3];
95     gmii_rxd_r[3] <= gmii_rxd_r[2];
96     gmii_rxd_r[2] <= gmii_rxd_r[1];
97     gmii_rxd_r[1] <= gmii_rxd_r[0];
98     gmii_rxd_r[0] <= gmii_rxd;
99     gmii_rx_dv_r <= {gmii_rx_dv_r[5 : 0],gmii_rx_dv};
100 end
101
102 //在状态机处于空闲状态下，检测到连续7个8'h55后又检测到一个8'hd5后表示检测到帧头，此时将介绍数据的开始信号拉高，其余时间保持为低电平。
103 always@(posedge clk)begin
104     if(rst_n==1'b0)begin//初始值为0;
105         start <= 1'b0;
106     end
107     else if(state_c == IDLE)begin
108         start <= ({gmii_rx_dv_r,gmii_rx_dv} == 8'hFF) && ({gmii_rxd,gmii_rxd_r[0],gmii_rxd_r[1],gmii_rxd_r[2],gmii_rxd_r[3],gmii_rxd_r[4],
109     end
110 end
111
112 //计数器，状态机在不同状态需要接收的数据个数不一样，使用一个可变进制的计数器。
113 always@(posedge clk)begin
114     if(rst_n==1'b0)begin//
115         cnt <= 0;
116     end
117     else if(add_cnt)begin
118         if(end_cnt)
119             cnt <= 0;
120

```

```

121         else
122             cnt <= cnt + 1;
123         end
124     else begin
125         cnt <= 0;
126     end
127 end
128 //当状态机不在空闲状态或接收数据结束阶段时计数，计数到该状态需要接收数据个数时清零。
129 assign add_cnt = (state_c != IDLE) && (state_c != RX_END) && gmii_rx_dv_r[0];
130 assign end_cnt = add_cnt && cnt == cnt_num - 1;
131
132 //状态机在不同状态，需要接收不同的数据个数，在接收以太网帧头时，需要接收14byte数据。
133 always@(posedge clk)begin
134     if(rst_n==1'b0)begin//初始值为20;
135         cnt_num <= 16'd20;
136     end
137     else begin
138         case(state_c)
139             ETH_HEAD : cnt_num <= 16'd14;//以太网帧头长度位14字节。
140             IP_HEAD  : cnt_num <= ip_head_byte_num;//IP帧头为20字节数据。
141             UDP_HEAD : cnt_num <= 16'd8;//UDP帧头为8字节数据。
142             UDP_DATA : cnt_num <= udp_data_length;//UDP数据段需要根据数据长度进行变化。
143             CRC      : cnt_num <= 16'd4;//CRC校验为4字节数据。
144             default: cnt_num <= 16'd20;
145         endcase
146     end
147 end
148
149 //接收目的MAC地址，需要判断这个包是不是发给开发板的，目的MAC地址是不是开发板的MAC地址或广播地址。
150 always@(posedge clk)begin
151     if(rst_n==1'b0)begin//初始值为0;
152         des_mac_t <= 48'd0;
153     end
154     else if((state_c == ETH_HEAD) && add_cnt && cnt < 5'd6)begin
155         des_mac_t <= {des_mac_t[39:0],gmii_rxd_r[0]};
156     end
157 end
158
159 //判断接收的数据是否正确，以此来生成错误指示信号，判断状态机跳转。
160 always@(posedge clk)begin
161

```

```

162     if(rst_n==1'b0)begin//初始值为0;
163         error_flag <= 1'b0;
164     end
165     else begin
166         case(state_c)
167             ETH_HEAD : begin
168                 if(add_cnt)
169                     if(cnt == 6)//判断接收的数据是不是发送给开发板或者广播数据。
170                         error_flag <= ((des_mac_t != BOARD_MAC) && (des_mac_t != 48'HFF_FF_FF_FF_FF));
171                     else if(cnt ==12)//判断接收的数据是不是IP协议。
172                         error_flag <= ({gmii_rxd_r[0],gmii_rxd} != ETH_TPYE);
173                 end
174                 IP_HEAD : begin
175                     if(add_cnt)begin
176                         if(cnt == 9)//如果当前接收的数据不是UDP协议，停止解析数据。
177                             error_flag <= (gmii_rxd_r[0] != UDP_TYPE);
178                         else if(cnt == 16'd18)//判断目的IP地址是否为开发板的IP地址。
179                             error_flag <= ({des_ip,gmii_rxd_r[0],gmii_rxd} != BOARD_IP);
180                     end
181                 end
182                 default: error_flag <= 1'b0;
183             endcase
184         end
185     end
186
187 //接收IP首部相关数据：
188 always@(posedge clk)begin
189     if(rst_n==1'b0)begin//初始值为0;
190         ip_head_byte_num <= 6'd20;
191         ip_total_length <= 16'd28;
192         des_ip <= 16'd0;
193         udp_data_length <= 16'd0;
194     end
195     else if(state_c == IP_HEAD && add_cnt)begin
196         case(cnt)
197             16'd0 : ip_head_byte_num <= {gmii_rxd_r[0][3:0],2'd0};//接收IP首部的字节个数。
198             16'd2 : ip_total_length[15:8] <= gmii_rxd_r[0];//接收IP报文总长度的高八位数据。
199             16'd3 : ip_total_length[7:0] <= gmii_rxd_r[0];//接收IP报文总长度的低八位数据。
200             16'd4 : udp_data_length <= ip_total_length - ip_head_byte_num - 8;//计算UDP报文数据段的长度，UDP帧头为8字节数据。
201             16'd16,16'd17: des_ip <= {des_ip[7:0],gmii_rxd_r[0]};//接收目的IP地址。
202

```

```

203         default: ;
204     endcase
205 end
206 end
207
208 //接收UDP首部相关数据:
209 always@(posedge clk)begin
210     if(rst_n==1'b0)begin//初始值为0;
211         des_port <= 16'd0;//目的端口号;
212         source_port <= 16'd0;//源端口号;
213     end
214     else if(state_c == UDP_HEAD && add_cnt)begin
215         case(cnt)
216             16'd0,16'd1 : source_port <= {source_port[7:0],gmii_rxd_r[0]};//接收源端口号。
217             16'd2,16'd3 : des_port <= {des_port[7:0],gmii_rxd_r[0]};//接收目的端口号。
218             default: ;
219         endcase
220     end
221 end
222
223 //接收UDP的数据段，并输出使能信号。
224 always@(posedge clk)begin
225     rx_data <= (state_c == UDP_DATA) ? gmii_rxd_r[0] : rx_data;//在接收UDP数据阶段时，接收数据。
226     rx_data_vld <= (state_c == UDP_DATA);//在接收数据阶段时，将FIFO写使能信号拉高，其余时间均拉低。
227 end
228
229 //生产CRC校验相关的数据和控制信号。
230 always@(posedge clk)begin
231     crc_data <= gmii_rxd_r[0];//将移位寄存器最低位存储的数据作为CRC输入模块的数据。
232     crc_clr <= (state_c == IDLE);//当状态机处于空闲状态时，清除CRC校验模块计算。
233     crc_en <= (state_c != IDLE) && (state_c != RX_END) && (state_c != CRC);//CRC校验使能信号。
234 end
235
236 //接收PC端发送来的CRC数据。
237 always@(posedge clk)begin
238     if(rst_n==1'b0)begin//初始值为0;
239         des_crc <= 24'hff_ff_ff;
240     end
241     else if(add_cnt && state_c == CRC)begin//先接收的是低位数据;
242         des_crc <= {gmii_rxd_r[0],des_crc[23:8]};
243

```

```

244     end
245 end
246
247 //生成相应的输出数据。
248 always@(posedge clk)begin
249     if(rst_n==1'b0)begin//初始值为0;
250         rx_done <= 1'b0;
251         data_byte_num <= 16'd0;
252     end//如果CRC校验成功，把UDP协议接收完成信号拉高，把接收到UDP数据个数和数据段的校验和输出。
253     else if(state_c == CRC && end_cnt && ({gmii_rxd_r[0],des_crc[23:0]} == crc_out))begin
254         rx_done <= 1'b1;
255         data_byte_num <= udp_data_length;
256     end
257     else begin
258         rx_done <= 1'b0;
259     end
260 end
end

```



该模块的仿真结果如下图所示，仿真表示该模块接收到三帧UDP数据，UDP数据段长度分别为36字节、19字节、54字节。橙色信号是gmii\_rxd\_r[0]，紫红色信号是状态机现态，粉色信号是计数器和计数器的最大值，黄色信号是CRC校验模块的清零、使能、输入信号、计算结果。天蓝色信号是接收到的UDP数据段信号。

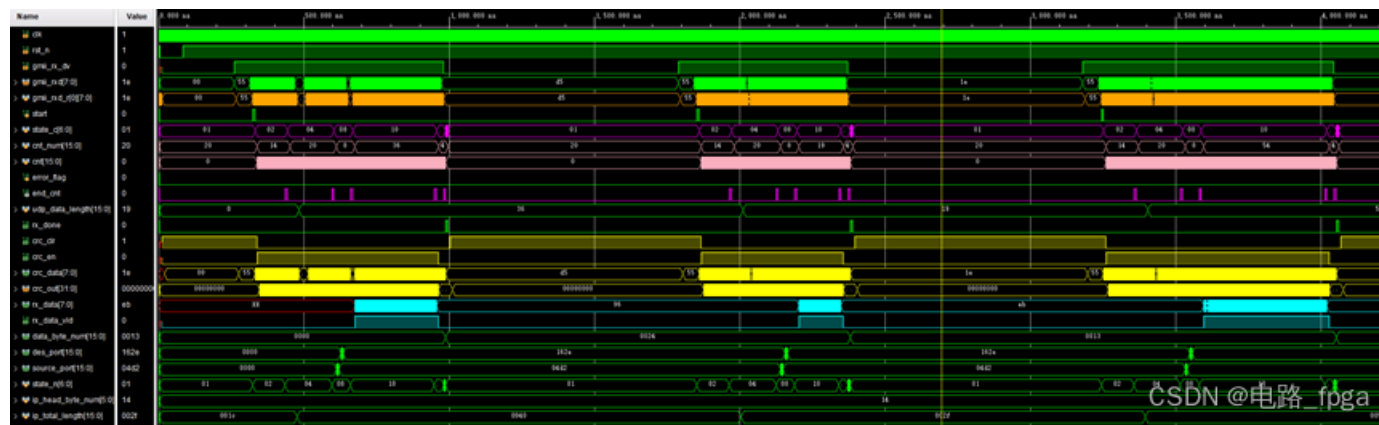


图6 UDP接收模块仿真

将UDP接收模块接收的第二帧数据放大，如下图所示，天蓝色信号将UDP数据段内容稳定输出。当CRC校验无误后，将rx\_done信号拉高，表示接收完一帧UDP数据报文。

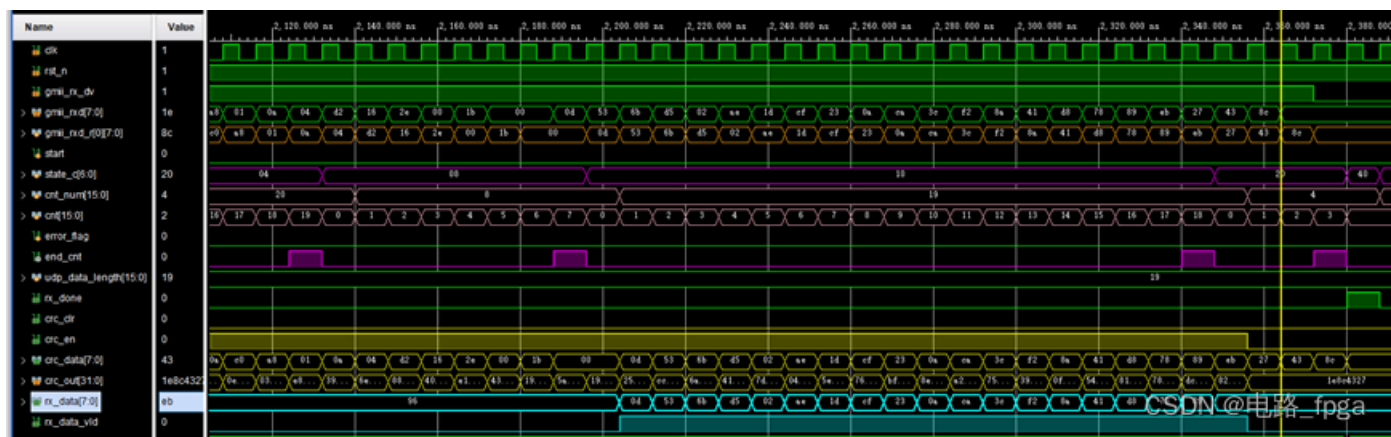


图7 UDP第二帧数据段放大

UDP接收模块的仿真就这么多了，需要详细了解的可以打开工程进行查看，工程中有对应的TestBench文件。

5、UDP发送模块

UDP发送模块同样可以采用状态机和计数器作为主体架构实现，状态机对应的状态转换图如下所示。该模块的实现相对于ICMP发送模块会简单一点，不需要计算UDP校验码，只需要计算IP首部校验码即可。最后需要注意如果UDP数据段不足18个字节数据，需要补零填充到18字节数据。



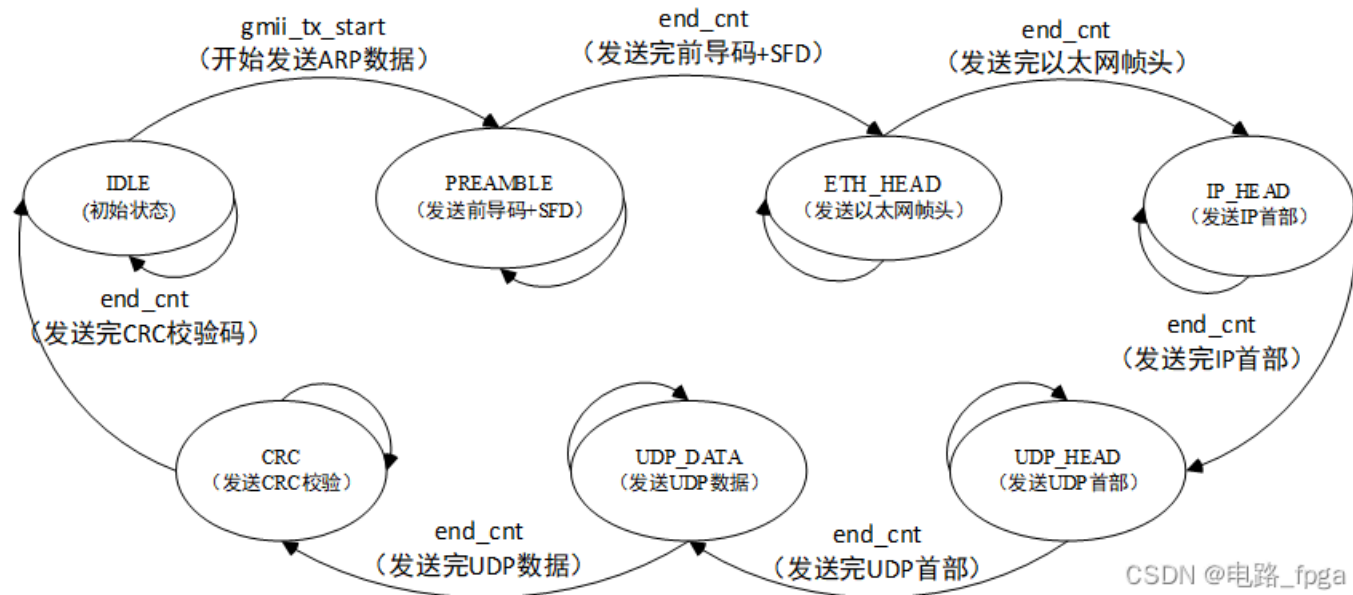


图8 状态转换图

该模块的核心代码如下所示，完整代码在工程中查看。

```

1  always@(posedge clk)begin
2      if(rst_n==1'b0)begin//初始值为0;
3          ip_head[0] <= 32'd0;
4          ip_head[1] <= 32'd0;
5          ip_head[2] <= 32'd0;
6          ip_head[3] <= 32'd0;
7          ip_head[4] <= 32'd0;
8          udp_head[0] <= {BOARD_PORT,DES_PORT};
9          udp_head[1] <= 32'd0;
10         ip_head_check <= 32'd0;
11         des_ip_r <= DES_IP;
12         des_mac_r <= DES_MAC;
13         tx_byte_num_r <= MIN_DATA_NUM;
14         ip_total_num <= MIN_DATA_NUM + 28;
15     end
16     //在状态机空闲状态下，上游发送使能信号时，将目的MAC地址和目的IP以及UDP需要发送的数据个数进行暂存。
17     else if(state_c == IDLE && udp_tx_start)begin
18         udp_head[0] <= {BOARD_PORT,DES_PORT}; //16位源端口和目的端口地址。
19

```

```

20 udp_head[1][31:16] <= (((tx_byte_num >= MIN_DATA_NUM) ? tx_byte_num : MIN_DATA_NUM) + 8); //计算UDP需要发送报文的长度。
21 tx_byte_num_r <= tx_byte_num;
22 //如果需要发送的数据多余最小长度要求，则发送的总数居等于需要发送的数据加上UDP和IP帧头数据。
23 ip_total_num <= (((tx_byte_num >= MIN_DATA_NUM) ? tx_byte_num : MIN_DATA_NUM) + 28);
24 if((des_mac != 48'd0) && (des_ip != 48'd0))begin//当接收到目的MAC地址和目的IP地址时更新。
25     des_ip_r <= des_ip;
26     des_mac_r <= des_mac;
27 end
28 end
29 //在发送以太网帧头时，就开始计算IP帧头和UDP的校验码，并将计算结果存储，便于后续直接发送。
30 else if(state_c == ETH_HEAD && add_cnt)begin
31     case (cnt)
32         16'd0 : begin//初始化需要发送的IP头部数据。
33             ip_head[0] <= {IP_VERSION,IP_HEAD_LEN,8'h00,ip_total_num[15:0]}; //依次表示IP版本号，IP头部长度，IP服务类型，IP包的总长度。
34             ip_head[2] <= {8'h80,8'd17,16'd0}; //分别表示生存时间，协议类型，1表示UDP，2表示IGMP，6表示TCP，17表示UDP协议，低16位校验和先默认为0；
35             ip_head[3] <= BOARD_IP; //源IP地址。
36             ip_head[4] <= des_ip_r; //目的IP地址。
37         end
38         16'd1 : begin//开始计算IP头部校验和数据，并且将计算结果存储到对应位置。
39             ip_head_check <= ip_head[0][31 : 16] + ip_head[0][15 : 0];
40         end
41         16'd2 : begin
42             ip_head_check <= ip_head_check + ip_head[1][31 : 16];
43         end
44         16'd3 : begin
45             ip_head_check <= ip_head_check + ip_head[1][15 : 0];
46         end
47         16'd4 : begin
48             ip_head_check <= ip_head_check + ip_head[2][31 : 16];
49         end
50         16'd5 : begin
51             ip_head_check <= ip_head_check + ip_head[3][31 : 16];
52         end
53         16'd6 : begin
54             ip_head_check <= ip_head_check + ip_head[3][15 : 0];
55         end
56         16'd7 : begin
57             ip_head_check <= ip_head_check + ip_head[4][31 : 16];
58         end
59         16'd8 : begin
60

```

```

50         ip_head_check <= ip_head_check + ip_head[4][15 : 0];
61     end
62     16'd9,16'd10 : begin
63         ip_head_check <= ip_head_check[31 : 16] + ip_head_check[15 : 0];
64     end
65     16'd11 : begin
66         ip_head[2][15:0] <= ~ip_head_check[15 : 0];
67         ip_head_check <= 32'd0;//校验和清零，用于下次计算。
68     end
69     default: begin
70         ip_head_check <= 32'd0;//校验和清零，用于下次计算。
71     end
72 endcase
73 end
74 else if(state_c == IP_HEAD && end_cnt)
75     ip_head[1] <= {ip_head[1][31:16]+1,16'h4000};//高16位表示标识，每次发送数据后会加1，低16位表示不分片。
76 end
77
78 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
79 always@(posedge clk)begin
80     if(!rst_n)begin
81         state_c <= IDLE;
82     end
83     else begin
84         state_c <= state_n;
85     end
86 end
87
88 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
89 always@(*)begin
90     case(state_c)
91         IDLE:begin
92             if(udp_tx_start)begin//在空闲状态接收到上游发出的使能信号;
93                 state_n = PREAMBLE;
94             end
95             else begin
96                 state_n = state_c;
97             end
98         end
99     end
100     PREAMBLE:begin
101

```

```

101         if(end_cnt)begin//发送完前导码和SFD;
102             state_n = ETH_HEAD;
103         end
104         else begin
105             state_n = state_c;
106         end
107     end
108     ETH_HEAD:begin
109         if(end_cnt)begin//发送完以太网帧头数据;
110             state_n = IP_HEAD;
111         end
112         else begin
113             state_n = state_c;
114         end
115     end
116     IP_HEAD:begin
117         if(end_cnt)begin//发送完IP帧头数据;
118             state_n = UDP_HEAD;
119         end
120         else begin
121             state_n = state_c;
122         end
123     end
124     UDP_HEAD:begin
125         if(end_cnt)begin//发送完UDP帧头数据;
126             state_n = UDP_DATA;
127         end
128         else begin
129             state_n = state_c;
130         end
131     end
132     UDP_DATA:begin
133         if(end_cnt)begin//发送完udp协议数据;
134             state_n = CRC;
135         end
136         else begin
137             state_n = state_c;
138         end
139     end
140     CRC:begin
141

```

```

142         if(end_cnt)begin//发送完CRC校验码;
143             state_n = IDLE;
144         end
145     else begin
146         state_n = state_c;
147     end
148 end
149 default:begin
150     state_n = IDLE;
151 end
152 endcase
153 end
154
155 //计数器，用于记录每个状态机每个状态需要发送的数据个数，每个时钟周期发送1byte数据。
156 always@(posedge clk)begin
157     if(rst_n==1'b0)begin//
158         cnt <= 0;
159     end
160     else if(add_cnt)begin
161         if(end_cnt)
162             cnt <= 0;
163         else
164             cnt <= cnt + 1;
165     end
166 end
167
168 assign add_cnt = (state_c != IDLE); //状态机不在空闲状态时计数。
169 assign end_cnt = add_cnt && cnt == cnt_num - 1; //状态机对应状态发送完对应个数的数据。
170
171 //状态机在每个状态需要发送的数据个数。
172 always@(posedge clk)begin
173     if(rst_n==1'b0)begin//初始值为20;
174         cnt_num <= 16'd20;
175     end
176     else begin
177         case (state_c)
178             PREAMBLE : cnt_num <= 16'd8; //发送7个前导码和1个8'h5。
179             ETH_HEAD : cnt_num <= 16'd14; //发送14字节的以太网帧头数据。
180             IP_HEAD : cnt_num <= 16'd20; //发送20个字节是IP帧头数据。
181             UDP_HEAD : cnt_num <= 16'd8; //发送8字节的UDP帧头数据。
182

```

```

183         UDP_DATA : if(tx_byte_num_r >= MIN_DATA_NUM)//如果需要发送的数据多余以太网最短数据要求，则发送指定个数数据。
184             cnt_num <= tx_byte_num_r;
185         else//否则需要将指定个数数据发送完成，不足长度补零，达到最短的以太网帧要求。
186             cnt_num <= MIN_DATA_NUM;
187         CRC : cnt_num <= 6'd5;//CRC在时钟1时才开始发送数据，这是因为CRC计算模块输出的数据会延后一个时钟周期。
188         default: cnt_num <= 6'd20;
189     endcase
190 end
191 end
192
193 //根据状态机和计数器的值产生输出数据，只不过这不是真正的输出，还需要延迟一个时钟周期。
194 always@(posedge clk)begin
195     if(rst_n==1'b0)begin//初始值为0;
196         crc_data <= 8'd0;
197     end
198     else if(add_cnt)begin
199         case (state_c)
200             PREAMBLE : if(end_cnt)
201                 crc_data <= 8'hd5;//发送1字节SFD编码;
202             else
203                 crc_data <= 8'h55;//发送7字节前导码;
204             ETH_HEAD : if(cnt < 6)
205                 crc_data <= des_mac_r[47 - 8*cnt -: 8];//发送目的MAC地址，先发高字节;
206             else if(cnt < 12)
207                 crc_data <= BOARD_MAC[47 - 8*(cnt-6) -: 8];//发送源MAC地址，先发高字节;
208             else
209                 crc_data <= ETH_TYPE[15 - 8*(cnt-12) -: 8];//发送源以太网协议类型，先发高字节;
210             IP_HEAD : if(cnt < 4)//发送IP帧头。
211                 crc_data <= ip_head[0][31 - 8*cnt -: 8];
212             else if(cnt < 8)
213                 crc_data <= ip_head[1][31 - 8*(cnt-4) -: 8];
214             else if(cnt < 12)
215                 crc_data <= ip_head[2][31 - 8*(cnt-8) -: 8];
216             else if(cnt < 16)
217                 crc_data <= ip_head[3][31 - 8*(cnt-12) -: 8];
218             else
219                 crc_data <= ip_head[4][31 - 8*(cnt-16) -: 8];
220             UDP_HEAD : if(cnt < 4)//发送UDP帧头数据。
221                 crc_data <= udp_head[0][31 - 8*cnt -: 8];
222             else
223

```

```

224         crc_data <= udp_head[1][31 - 8*(cnt-4) -: 8];
225     UDP_DATA : if(tx_byte_num_r >= MIN_DATA_NUM)//需要判断发送的数据是否满足以太网最小数据要求。
226         crc_data <= tx_data;//如果满足最小要求，将需要配发送的数据输出。
227     else if(cnt < tx_byte_num_r)//不满足最小要求时，先将需要发送的数据发送完。
228         crc_data <= tx_data;//将需要发送的数据输出即可。
229     else//剩余数据补充0。
230         crc_data <= 8'd0;
231     default : ;
232 endcase
233 end
234 end
235
236 //生成数据请求输入信号，外部输入数据延后该信号一个时钟周期，所以需要提前产生一个时钟周期产生请求信号；
237 always@(posedge clk)begin
238     if(rst_n==1'b0)begin//初始值为0；
239         tx_data_req <= 1'b0;
240     end
241     //在数据段的前三个时钟周期拉高；
242     else if(state_c == UDP_HEAD && add_cnt && (cnt == cnt_num - 2))begin
243         tx_data_req <= 1'b1;
244     end//在ICMP或者UDP数据段时，当发送完数据的前三个时钟拉低；
245     else if(state_c == UDP_DATA && add_cnt && (cnt == cnt_num - 2))begin
246         tx_data_req <= 1'b0;
247     end
248 end
249
250 //生成一个crc_data指示信号，用于生成gmii_txd信号。
251 always@(posedge clk)begin
252     if(rst_n==1'b0)begin//初始值为0；
253         gmii_tx_en_r <= 1'b0;
254     end
255     else if(state_c == CRC)begin
256         gmii_tx_en_r <= 1'b0;
257     end
258     else if(state_c == PREAMBLE)begin
259         gmii_tx_en_r <= 1'b1;
260     end
261 end
262
263 //生产CRC校验模块使能信号，初始值为0，当开始输出以太网帧头时拉高，当ARP和以太网帧头数据全部输出后拉低。
264

```

```

265 always@(posedge clk)begin
266     if(rst_n==1'b0)begin//初始值为0;
267         crc_en <= 1'b0;
268     end
269     else if(state_c == CRC)begin//当ARP和以太网帧头数据全部输出后拉低.
270         crc_en <= 1'b0;
271     end//当开始输出以太网帧头时拉高。
272     else if(state_c == ETH_HEAD && add_cnt)begin
273         crc_en <= 1'b1;
274     end
275 end
276
277 //生产CRC校验模块清零信号，状态机处于空闲时清零。
278 always@(posedge clk)begin
279     crc_clr <= (state_c == IDLE);
280 end
281
282 //生成gmii_txd信号，默认输出0。
283 always@(posedge clk)begin
284     if(rst_n==1'b0)begin//初始值为0;
285         gmii_txd <= 8'd0;
286     end//在输出CRC状态时，输出CRC校验码，先发送低位数据。
287     else if(state_c == CRC && add_cnt && cnt>0)begin
288         gmii_txd <= crc_out[8*cnt-1 -: 8];
289     end//其余时间如果crc_data有效，则输出对应数据。
290     else if(gmii_tx_en_r)begin
291         gmii_txd <= crc_data;
292     end
293 end
294
295 //生成gmii_txd有效指示信号。
296 always@(posedge clk)begin
297     gmii_tx_en <= gmii_tx_en_r || (state_c == CRC);
298 end
299
300 //模块忙闲指示信号，当接收到上游模块的使能信号或者状态机不处于空闲状态时拉低，其余时间拉高。
301 //该信号必须使用组合逻辑产生，上游模块必须使用时序逻辑检测该信号。
always@(*)begin
    if(udp_tx_start || state_c != IDLE)
        rdy = 1'b0;

```



```

else
    rdy = 1'b1;
end

```



以太网发送数据模块仿真结果如下所示，发送了三帧数据。



图9 发送数据模块仿真

该模块需要注意什么？其实需要考虑的是CRC校验模块输出的数据会滞后输入一个时钟周期，为了实现数据对齐，需要把crc\_data延迟一个时钟周期得到gmii\_txd，仿真结果如下所示。

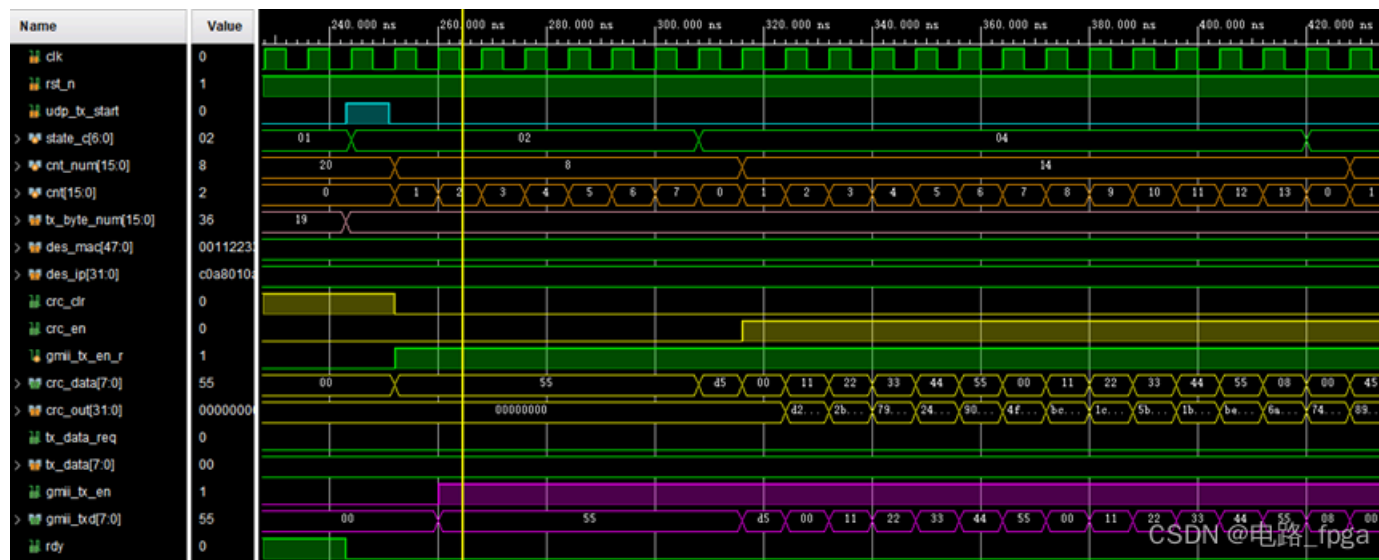


图10 开始发送数据

前文在实现ICMP发送模块的时候，FIFO使用了超前模式，读使能与读数据对齐，但是UDP发送的数据未必来自FIFO，更多情况可能是数据会滞后请求信号一个时钟。所以本文把FIFO换成常规模式，输出的数据会滞后读使能一个时钟周期。

那么就需要提前一个时钟周期产生请求信号，对应的仿真结果如下所示，在状态机发送UDP首部最后一个字节数据时，将请求信号req拉高。

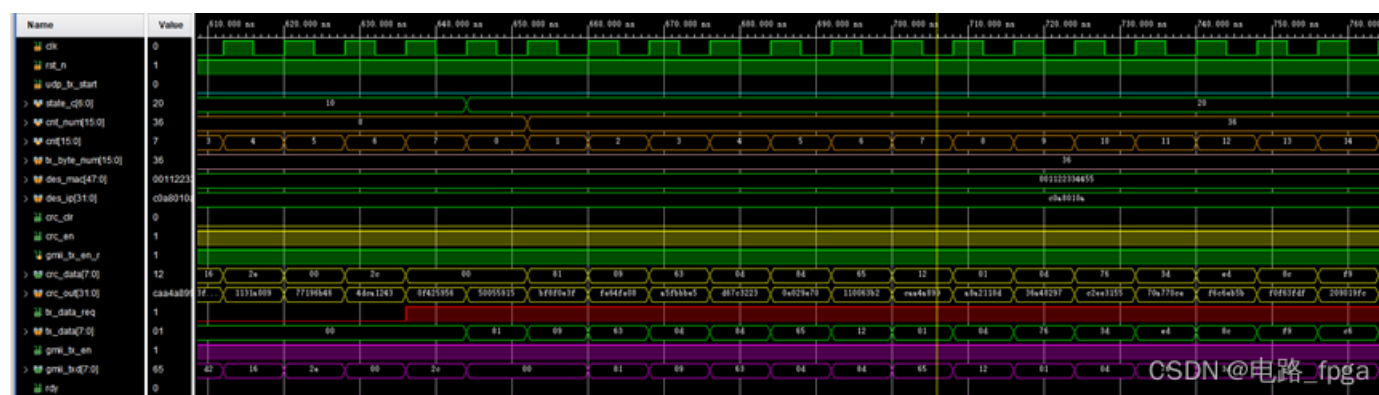


图11 数据请求仿真

该模块的仿真到此结束，具体的CRC仿真还有IP首部、UDP首部这些细节就不再赘述了，与前文的ARP和ICMP道里差不多，需要详细了解的可以在公众号获取工程文件自行查看。

## 6、ARP、ICMP、UDP控制模块

本文实现UDP的回环，为了不去手动绑定开发板的IP地址和MAC地址，所以需要ARP模块，还要能够判断以太网链路是否畅通，就需要使用ICMP协议。开发板只使用一个网口，但是ARP、ICMP、UDP均会输出gmii\_txd信号，所以就需要一个控制模块对三个模块的输出进行仲裁。

该模块接收到ARP请求时，就会使能ARP发送模块，向PC端发出ARP应答指令。当开发板上某个按键被按下后，也会向PC端发出ARP请求指令。当接收到PC端发出的回显请求指令时，该模块使能ICMP发送模块向PC端发送回显应答指令。最后当接收到UDP数据报文后，将接收的数据通过UDP发送模块传送给PC，实现数据回环。

该模块的核心代码如下所示，由于篇幅原因，需要代码可以从公众号的工程获取。

```
1 //ARP发送数据报的类型。
2 always@(posedge clk)begin
3     if(rst_n==1'b0)begin//初始值为0;
4         arp_tx_type <= 1'b0;
5     end
6     else if(arp_rx_done && ~arp_rx_type)begin//接收到PC的ARP请求时，应该回发应答信号。
7         arp_tx_type <= 1'b1;
8     end
9     else if(key_in || (arp_rx_done && arp_rx_type))begin//其余时间发送请求指令。
10         arp_tx_type <= 1'b0;
11     end
12 end
13
14 //接收到ARP请求数据报文时，将接收到的目的MAC和IP地址输出。
15 always@(posedge clk)begin
16     if(rst_n==1'b0)begin//初始值为0;
17         arp_tx_start <= 1'b0;
18         des_mac <= 48'd0;
19         des_ip <= 32'd0;
20     end
21     else if(arp_rx_done && ~arp_rx_type)begin
22         arp_tx_start <= 1'b1;
23         des_mac <= src_mac;
24         des_ip <= src_ip;
25     end
26     else if(key_in)begin
27         arp_tx_start <= 1'b1;
28     end
29     else begin
30         arp_tx_start <= 1'b0;
31     end
32 end
33
```

```

33
34 //接收到ICMP请求数据报文时，发送应答数据报。
35 always@(posedge clk)begin
36     if(rst_n==1'b0)begin//初始值为0;
37         icmp_tx_start <= 1'b0;
38         icmp_tx_byte_num <= 16'd0;
39     end
40     else if(icmp_rx_done)begin
41         icmp_tx_start <= 1'b1;
42         icmp_tx_byte_num <= icmp_rx_byte_num;
43     end
44     else begin
45         icmp_tx_start <= 1'b0;
46     end
47 end
48
49 //接收到UDP数据报文后，将数据发送回源端。
50 always@(posedge clk)begin
51     if(rst_n==1'b0)begin//初始值为0;
52         udp_tx_start <= 1'b0;
53         udp_tx_byte_num <= 16'd0;
54     end
55     else if(udp_rx_done)begin
56         udp_tx_start <= 1'b1;
57         udp_tx_byte_num <= udp_rx_byte_num;
58     end
59     else begin
60         udp_tx_start <= 1'b0;
61     end
62 end
63
64 //对三个模块需要发送的数据进行整合。
65 always@(posedge clk)begin
66     if(rst_n==1'b0)begin//初始值为0;
67         gmii_tx_en <= 1'b0;
68         gmii_txd <= 8'd0;
69     end//如果ARP发送模块输出有效数据，且ICMP发送模块和UDP发送模块都处于空闲状态，则将ARP相关数据输出。
70     else if(arp_gmii_tx_en && icmp_tx_rdy && udp_tx_rdy)begin
71         gmii_tx_en <= arp_gmii_tx_en;
72         gmii_txd <= arp_gmii_txd;
73

```

```

74      end//如果ICMP发送模块输出有效数据且ARP发送模块和UDP发送模块均处于空闲，则将ICMP相关数据输出。
75      else if(icmp_gmii_tx_en && arp_tx_rdy && udp_tx_rdy)begin
76          gmii_tx_en <= icmp_gmii_tx_en;
77          gmii_txd <= icmp_gmii_txd;
78      end//如果udp发送模块输出有效数据且ARP发送模块和icmp发送模块均处于空闲，则将ICMP相关数据输出。
79      else if(udp_gmii_tx_en && arp_tx_rdy && icmp_tx_rdy)begin
80          gmii_tx_en <= udp_gmii_tx_en;
81          gmii_txd <= udp_gmii_txd;
82      end
83      else begin
84          gmii_tx_en <= 1'b0;
85      end
end
end

```



该模块的思路比较简单，就不再仿真，后续直接上板即可。

## 7、顶层模块

顶层模块主要将ARP、ICMP、UDP、RGMII与GMI转换模块、按键消抖模块、暂存UDP数据的FIFO模块、锁相环模块的输入输出端口进行连线。

顶层对应的框图如下所示，由于比较复杂，直接采用vivado的RTL视图。



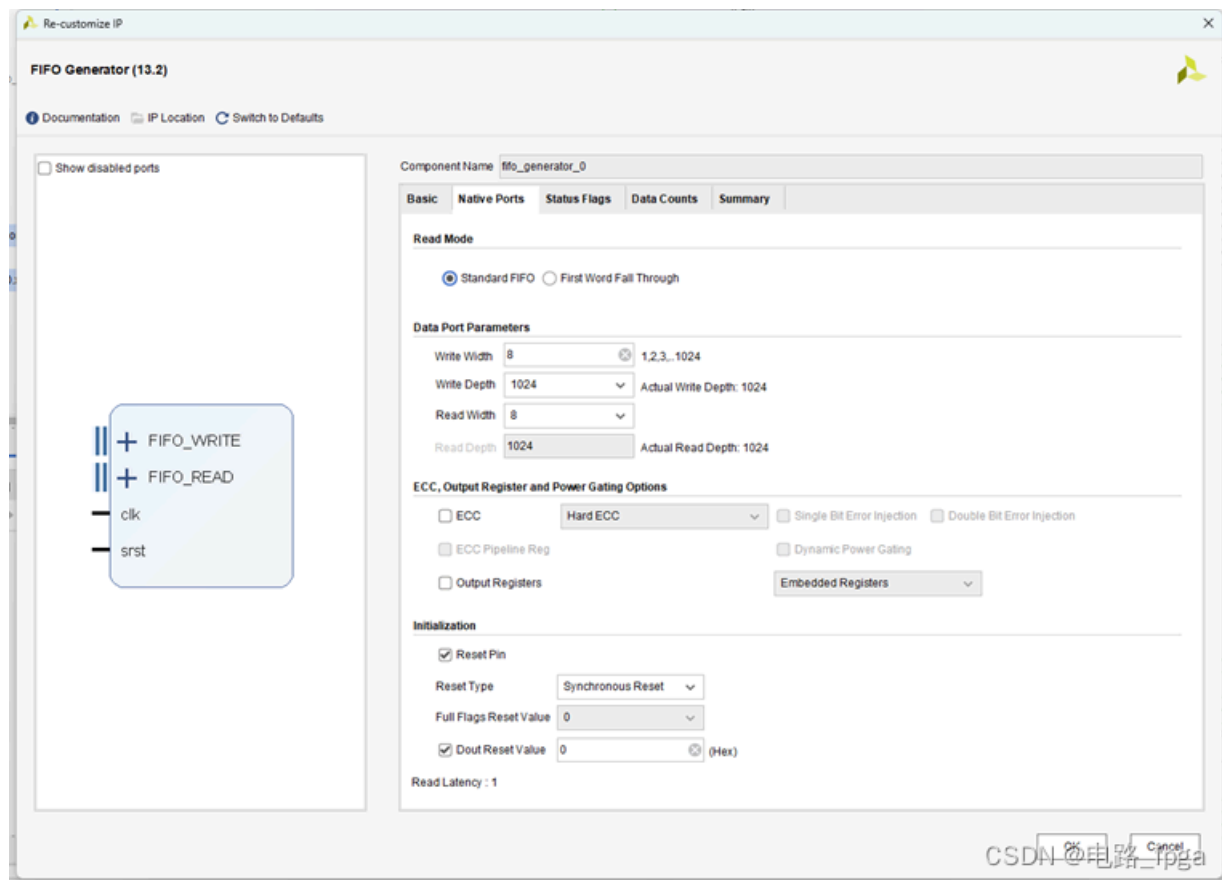


图13 FIFO配置

该模块对应核心代码如下所示：

```

1 //例化锁相环，输出200MHZ时钟，作为IDELAYCTRL的参考时钟。
2 clk_wiz_0 u_clk_wiz_0 (
3     .clk_out1    ( idelay_clk), //output clk_out1;
4     .resetn      ( rst_n      ), //input resetn;
5     .clk_in1     ( clk        ) //input clk_in1;
6 );
7
8 //例化按键消抖模块。
9 key #(
10     .TIME_20MS   ( TIME_20MS ), //按键抖动持续的最长时间，默认最长持续时间为20ms。
11     .TIME_CLK    ( TIME_CLK   ) //系统时钟周期，默认8ns。

```

```

12 )
13 u_key (
14     .clk          ( gmii_rx_clk    ),//系统时钟，125MHz。
15     .rst_n        ( rst_n          ),//系统复位，低电平有效。
16     .key_in       ( key_in         ),//待输入的按键输入信号，默认低电平有效；
17     .key_out      ( key_out        ) //按键消抖后输出信号，当按键按下一次时，输出一个时钟宽度的高电平；
18 );
19
20 //例化ARP和ICMP的控制模块
21 arp_icmp_udp_ctrl u_arp_icmp_udp_ctrl (
22     .clk          ( gmii_rx_clk    ),//输入时钟；
23     .rst_n        ( rst_n          ),//复位信号，低电平有效；
24     .key_in       ( key_out        ),//按键按下，高电平有效；
25     .des_mac      ( des_mac        ),//发送的目标MAC地址。
26     .des_ip       ( des_ip         ),//发送的目标IP地址。
27     //ARP
28     .arp_rx_done  ( arp_rx_done    ),//ARP接收完成信号；
29     .arp_rx_type  ( arp_rx_type    ),//ARP接收类型 0:请求 1:应答；
30     .src_mac      ( src_mac        ),//ARP接收到目的MAC地址。
31     .src_ip       ( src_ip         ),//ARP接收到目的IP地址。
32     .arp_tx_rdy   ( arp_tx_rdy     ),//ARP发送模块忙闲指示信号。
33     .arp_tx_start ( arp_tx_start    ),//ARP发送使能信号；
34     .arp_tx_type  ( arp_tx_type    ),//ARP发送类型 0:请求 1:应答；
35     .arp_gmii_tx_en ( arp_gmii_tx_en ),
36     .arp_gmii_txd ( arp_gmii_txd   ),
37     //ICMP
38     .icmp_rx_done ( icmp_rx_done    ),//ICMP接收完成信号；
39     .icmp_rx_byte_num ( icmp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
40     .icmp_tx_rdy  ( icmp_tx_rdy     ),//ICMP发送模块忙闲指示信号。
41     .icmp_gmii_tx_en ( icmp_gmii_tx_en ),
42     .icmp_gmii_txd ( icmp_gmii_txd   ),
43     .icmp_tx_start ( icmp_tx_start   ),//ICMP发送使能信号；
44     .icmp_tx_byte_num ( icmp_tx_byte_num ),//以太网发送的有效字节数 单位:byte。
45     //udp
46     .udp_rx_done  ( udp_rx_done     ),//UDP接收完成信号；
47     .udp_rx_byte_num ( udp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
48     .udp_tx_rdy   ( udp_tx_rdy     ),//UDP发送模块忙闲指示信号。
49     .udp_gmii_tx_en ( udp_gmii_tx_en ),
50     .udp_gmii_txd ( udp_gmii_txd   ),
51     .udp_tx_start ( udp_tx_start    ),//UDP发送使能信号；
52

```



```

53     .udp_tx_byte_num    ( udp_tx_byte_num    ),//以太网发送的有效字节数 单位:byte。
54
55     .gmii_tx_en         ( gmii_tx_en         ),
56     .gmii_txd           ( gmii_txd           )
57 );
58
59 //例化ARP模块:
60 arp #(
61     .BOARD_MAC         ( BOARD_MAC         ),//开发板MAC地址 00-11-22-33-44-55;
62     .BOARD_IP          ( BOARD_IP          ),//开发板IP地址 192.168.1.10;
63     .DES_MAC           ( DES_MAC           ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
64     .DES_IP            ( DES_IP            ),//目的IP地址 192.168.1.102;
65     .ETH_TYPE          ( 16'h0806         ) //以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
66 )
67 u_arp (
68     .rst_n              ( rst_n              ),//复位信号, 低电平有效。
69     .gmii_rx_clk        ( gmii_rx_clk        ),//GMII接收数据时钟。
70     .gmii_rx_dv         ( gmii_rx_dv         ),//GMII输入数据有效信号。
71     .gmii_rxd           ( gmii_rxd           ),//GMII输入数据。
72     .gmii_tx_clk        ( gmii_tx_clk        ),//GMII发送数据时钟。
73     .arp_tx_en          ( arp_tx_start       ),//ARP发送使能信号。
74     .arp_tx_type        ( arp_tx_type        ),//ARP发送类型 0:请求 1:应答。
75     .des_mac            ( des_mac            ),//发送的目标MAC地址。
76     .des_ip             ( des_ip             ),//发送的目标IP地址。
77     .gmii_tx_en         ( arp_gmii_tx_en     ),//GMII输出数据有效信号。
78     .gmii_txd           ( arp_gmii_txd       ),//GMII输出数据。
79     .arp_rx_done        ( arp_rx_done        ),//ARP接收完成信号。
80     .arp_rx_type        ( arp_rx_type        ),//ARP接收类型 0:请求 1:应答。
81     .src_mac            ( src_mac            ),//接收到目的MAC地址。
82     .src_ip             ( src_ip             ),//接收到目的IP地址。
83     .arp_tx_rdy         ( arp_tx_rdy        ) //ARP发送模块忙闲指示指示信号, 高电平表示该模块空闲。
84 );
85
86 //例化ICMP模块。
87 icmp #(
88     .BOARD_MAC         ( BOARD_MAC         ),//开发板MAC地址 00-11-22-33-44-55;
89     .BOARD_IP          ( BOARD_IP          ),//开发板IP地址 192.168.1.10;
90     .DES_MAC           ( DES_MAC           ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
91     .DES_IP            ( DES_IP            ),//目的IP地址 192.168.1.102;
92     .ETH_TYPE          ( 16'h0800         ) //以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
93

```

```

94 )
95 u_icmp (
96     .rst_n          ( rst_n          ),//复位信号，低电平有效。
97     .gmii_rx_clk    ( gmii_rx_clk    ),//GMII接收数据时钟。
98     .gmii_rx_dv     ( gmii_rx_dv     ),//GMII输入数据有效信号。
99     .gmii_rxd       ( gmii_rxd       ),//GMII输入数据。
100    .gmii_tx_clk     ( gmii_tx_clk     ),//GMII发送数据时钟。
101    .gmii_tx_en      ( icmp_gmii_tx_en ),//GMII输出数据有效信号。
102    .gmii_txd        ( icmp_gmii_txd   ),//GMII输出数据。
103    .icmp_tx_start   ( icmp_tx_start   ),//以太网开始发送信号。
104    .icmp_tx_byte_num ( icmp_tx_byte_num ),//以太网发送的有效字节数 单位:byte。
105    .des_mac         ( des_mac         ),//发送的目标MAC地址。
106    .des_ip          ( des_ip          ),//发送的目标IP地址。
107    .icmp_rx_done    ( icmp_rx_done    ),//ICMP接收完成信号。
108    .icmp_rx_byte_num ( icmp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
109    .icmp_tx_rdy     ( icmp_tx_rdy     ) //ICMP发送模块忙闲指示指示信号，高电平表示该模块空闲。
110 );
111
112 //例化UDP模块。
113 udp #(
114     .BOARD_MAC ( BOARD_MAC ),//开发板MAC地址 00-11-22-33-44-55;
115     .BOARD_IP  ( BOARD_IP  ),//开发板IP地址 192.168.1.10;
116     .DES_MAC   ( DES_MAC   ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
117     .DES_IP    ( DES_IP    ),//目的IP地址 192.168.1.102;
118     .BOARD_PORT ( BOARD_PORT ),//板子的UDP端口号;
119     .DES_PORT   ( DES_PORT   ),//源端口号;
120     .ETH_TYPE   ( 16'h0800 ) //以太网帧类型，16'h0806表示ARP协议，16'h0800表示IP协议;
121 )
122 u_udp (
123     .rst_n          ( rst_n          ),//复位信号，低电平有效。
124     .gmii_rx_clk    ( gmii_rx_clk    ),//GMII接收数据时钟。
125     .gmii_rx_dv     ( gmii_rx_dv     ),//GMII输入数据有效信号。
126     .gmii_rxd       ( gmii_rxd       ),//GMII输入数据。
127     .gmii_tx_clk     ( gmii_tx_clk     ),//GMII发送数据时钟。
128     .gmii_tx_en      ( udp_gmii_tx_en ),//GMII输出数据有效信号。
129     .gmii_txd        ( udp_gmii_txd   ),//GMII输出数据。
130
131     .udp_tx_start   ( udp_tx_start   ),//以太网开始发送信号。
132     .udp_tx_byte_num ( udp_tx_byte_num ),//以太网发送的有效字节数 单位:byte。
133     .des_mac        ( des_mac        ),//发送的目标MAC地址。
134

```

```

135     .des_ip          ( des_ip          ),//发送的目标IP地址。
136     .udp_rx_done     ( udp_rx_done     ),//UDP接收完成信号。
137     .udp_rx_byte_num ( udp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
138     .udp_tx_rdy      ( udp_tx_rdy      ),//UDP发送模块忙闲指示指示信号，高电平表示该模块空闲。
139     .rx_data         ( udp_rx_data     ),
140     .rx_data_vld     ( udp_rx_data_vld ),
141     .tx_data         ( udp_tx_data     ),
142     .tx_data_req     ( udp_tx_data_req )
143 );
144
145 //例化FIFO;
146 fifo_generator_0 u_fifo_generator_0 (
147     .clk      ( gmii_rx_clk      ),//input wire clk
148     .srst     ( ~rst_n           ),//input wire srst
149     .din      ( udp_rx_data      ),//input wire [7 : 0] din
150     .wr_en    ( udp_rx_data_vld  ),//input wire wr_en
151     .rd_en    ( udp_tx_data_req  ),//input wire rd_en
152     .dout     ( udp_tx_data      ),//output wire [7 : 0] dout
153     .full     (                  ),//output wire full
154     .empty    (                  ) //output wire empty
155 );
156
157 //例化gmii转RGMII模块。
158 rgmii_to_gmii u_rgmii_to_gmii (
159     .idelay_clk      ( idelay_clk      ),//IDELAY时钟;
160     .rst_n           ( rst_n           ),
161     .gmii_tx_en      ( gmii_tx_en      ),//GMII发送数据使能信号;
162     .gmii_txd        ( gmii_txd        ),//GMII发送数据;
163     .gmii_rx_clk     ( gmii_rx_clk     ),//GMII接收时钟;
164     .gmii_rx_dv      ( gmii_rx_dv      ),//GMII接收数据有效信号;
165     .gmii_rxd        ( gmii_rxd        ),//GMII接收数据;
166     .gmii_tx_clk     ( gmii_tx_clk     ),//GMII发送时钟;
167
168     .rgmii_rxc       ( rgmii_rxc       ),//RGMII接收时钟;
169     .rgmii_rx_ctl    ( rgmii_rx_ctl    ),//RGMII接收数据控制信号;
170     .rgmii_rxd       ( rgmii_rxd       ),//RGMII接收数据;
171     .rgmii_txc       ( rgmii_txc       ),//RGMII发送时钟;
172     .rgmii_tx_ctl    ( rgmii_tx_ctl    ),//RGMII发送数据控制信号;
173     .rgmii_txd       ( rgmii_txd       ) //RGMII发送数据;
174 );

```



## 8、上板测试

将顶层模块中的ILA注释取消，然后将程序综合、实现，最后下载到开发板中进行测试。打开电脑的控制面板->网络和Internet->网络连接，鼠标右击以太网，双击Internet协议版本4，进行如下设置，与代码顶层模块设置的目的IP一致，[具体步骤可以查看前文](#)。



图14 电脑IP设置

然后把wirrshark和网络调试助手打开，如下所示：

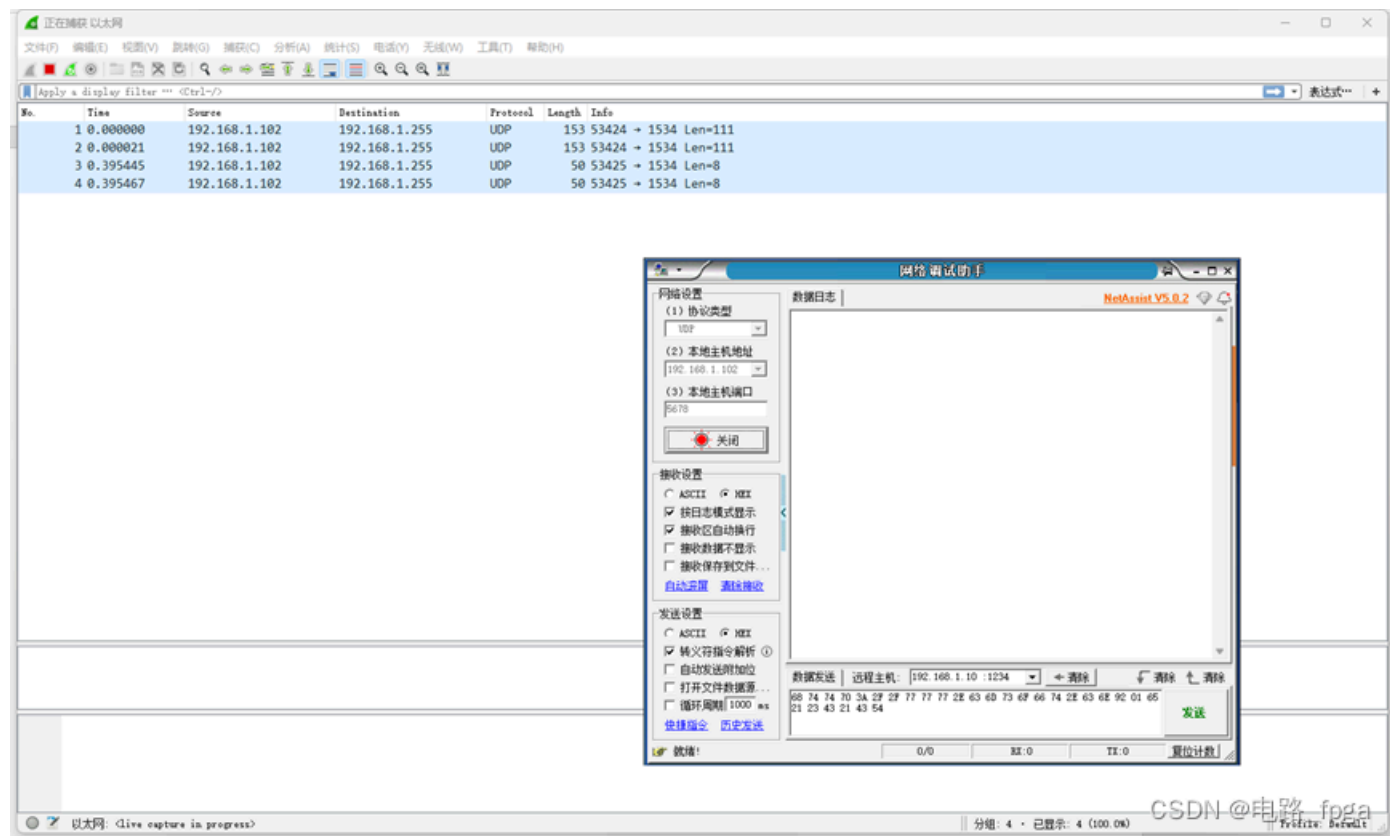


图15 wireshark与网络调试助手

网络调试助手需要设置协议类型为UDP，PC端的IP地址和UDP地址，需要与顶层文件的数值保持一致。然后打开连接，就会显示出FPGA的IP地址和UDP端口地址，如果该地址与开发板的地址不一样，可以手动进行修改。

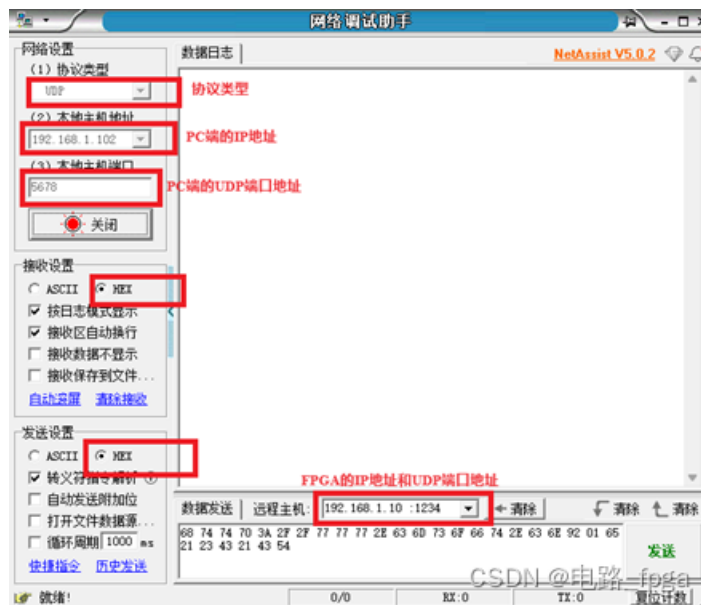


图16 网络调试助手的设置

之后将ILA设置为gmii\_rx\_dv的上升沿触发，连续抓取32个数据报文，然后wireshark也运行，最后点击网络调试助手的发送指令，即可抓取相关数据。网络调试助手发送三帧数据，如下图所示，FPGA向PC端返回接收到的三帧数据（蓝色数据是PC端通过UDP向FPGA发送的数据，绿色数据是FPGA通过UDP向PC端发送的数据）。

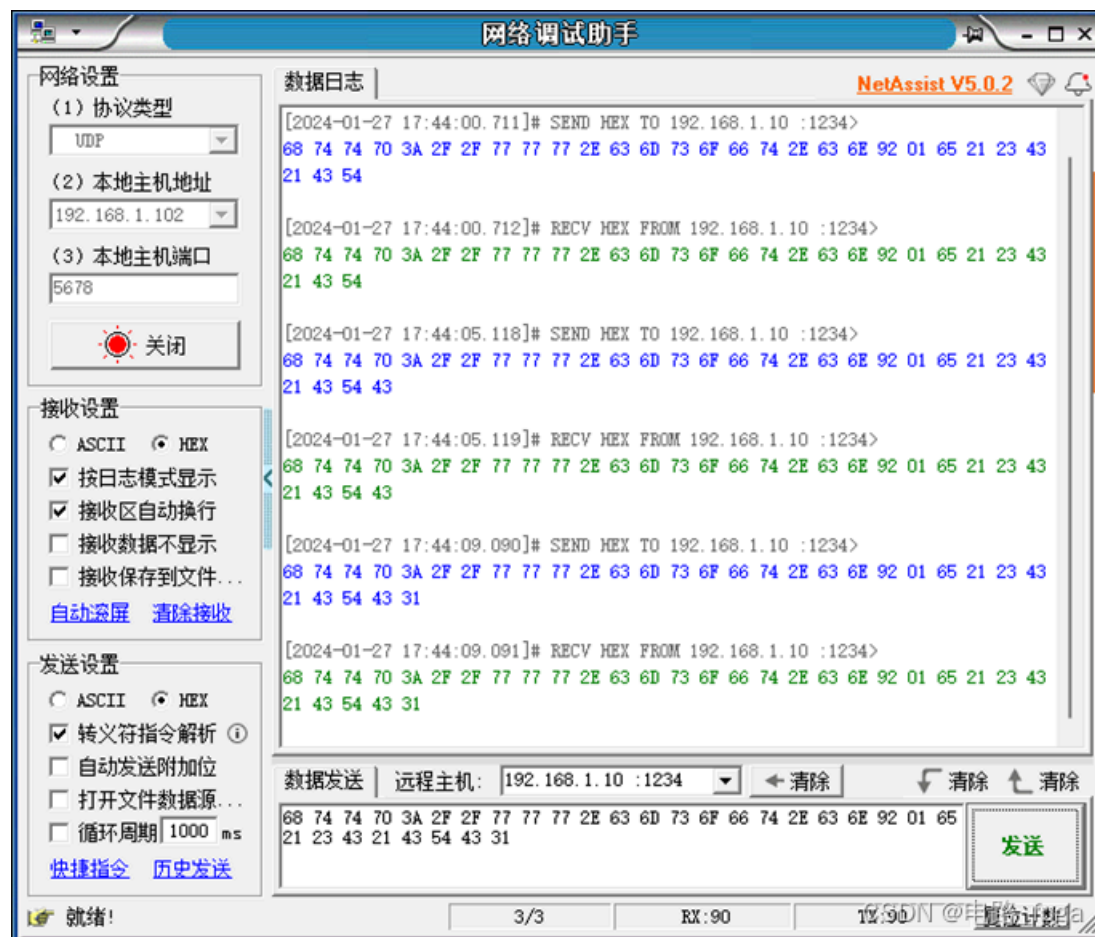


图17 网络调试助手收发数据

对比网络调试助手收发数据一致，由此证明FPGA接收和发送数据无误。

然后查看wireshark在这段时间抓取的数据报文，如下图所示。

PC端在通过UDP向FPGA发送数据报文之前，先通过广播的形式发送了一个ARP请求指令，去获取开发板的MAC地址，FPGA接收到ARP请求后，也是向PC端返回了ARP应答数据报文。

然后PC端通过UDP向FPGA发送三个数据报文，如下图所示，FPGA也对该报文进行了应答。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	192.168.1.255	UDP	50	53425 → 1534 Len=8
2	0.000019	192.168.1.102	192.168.1.255	UDP	50	53425 → 1534 Len=8
3	1.796547	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.102
4	1.796557	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.102
5	1.796709	Cimsys_33:44:55	d8:43:ae:16:19:d8	ARP	60	192.168.1.10 is at 00:11:22:33:44:55
6	1.796722	192.168.1.102	192.168.1.10	UDP	71	5678 → 1234 Len=29
7	1.796728	192.168.1.102	192.168.1.10	UDP	71	5678 → 1234 Len=29
8	1.796886	192.168.1.10	192.168.1.102	UDP	71	1234 → 5678 Len=29
9	6.204331	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
10	6.204345	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
11	6.204399	192.168.1.10	192.168.1.102	UDP	72	1234 → 5678 Len=30
12	8.460841	192.168.1.102	192.168.1.255	UDP	50	1534 → 1534 Len=8
13	8.460859	192.168.1.102	192.168.1.255	UDP	50	1534 → 1534 Len=8
14	10.175616	192.168.1.102	192.168.1.10	UDP	73	5678 → 1234 Len=31
15	10.175627	192.168.1.102	192.168.1.10	UDP	73	5678 → 1234 Len=31
16	10.175683	192.168.1.10	192.168.1.102	UDP	73	1234 → 5678 Len=31
17	10.972853	192.168.1.102	192.168.1.255	UDP	50	56496 → 1534 Len=8
18	10.972871	192.168.1.102	192.168.1.255	UDP	50	56496 → 1534 Len=8
19	14.483253	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.102
20	14.483274	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.102

图18 wireshark抓取数据报文

前文对ARP的报文已经做了详细讲解，所以此处不对其报文进行分析了，我们双击UDP报文，查看其发送的数据段，如下图蓝色背景文字部分，与图17中第一帧数据保持一致。



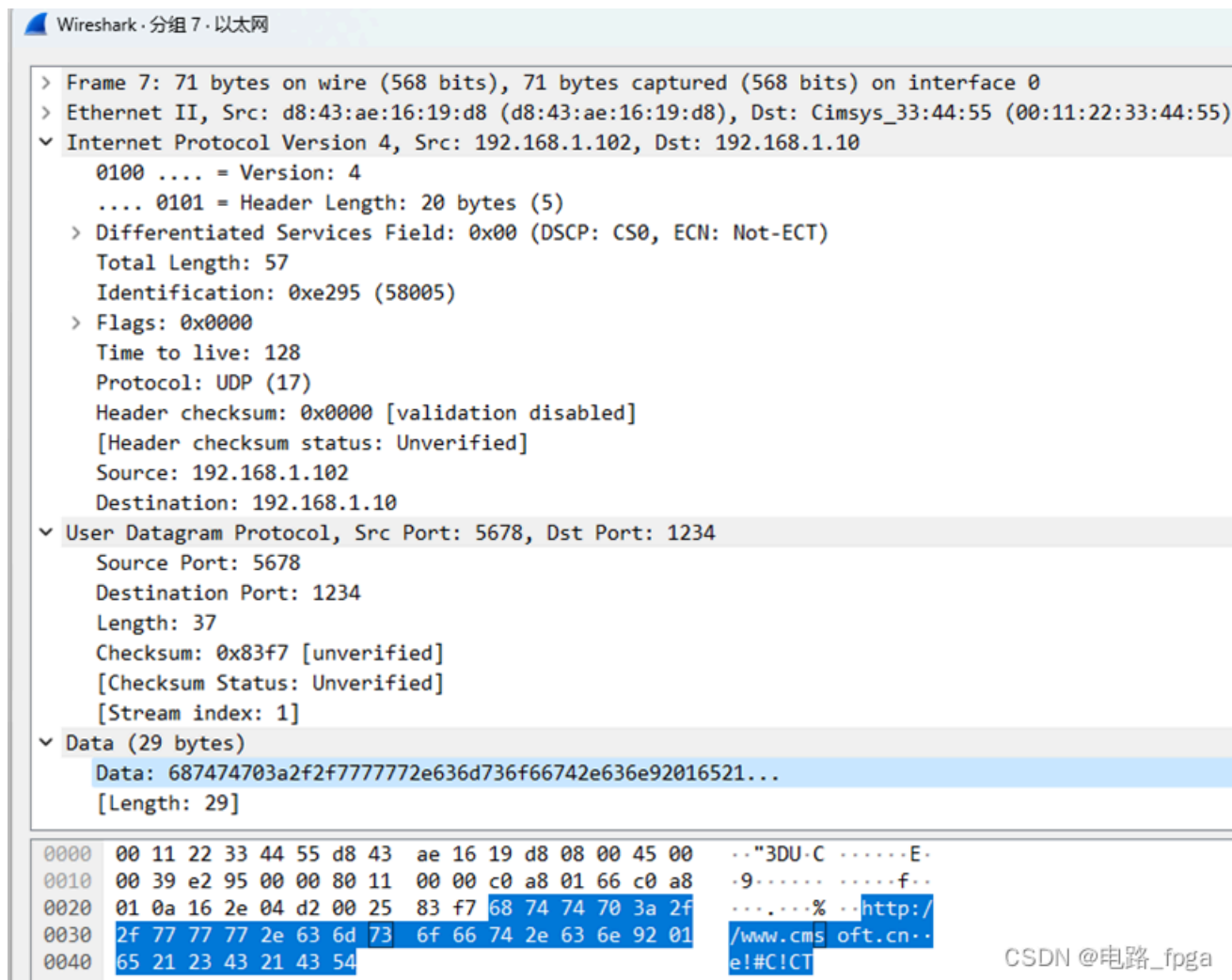


图19 wireshark抓取发送的第一帧数据报文

如下图所示是wireshark抓取的FPGA通过UDP给PC端发送的第一帧报文，可以从红框处得知源MAC和源IP地址为开发板，目的MAC和目的IP都是PC端的地址。接收的UDP数据就是蓝色文字，与图19PC端发送的数据保持一致。



图21 ILA抓取接收的第一帧数据报文

将接收的UDP数据段放大后如下图所示，与图19和图17PC端发送的第一帧数据保持一致，因此FPGA这边接收数据没有问题。

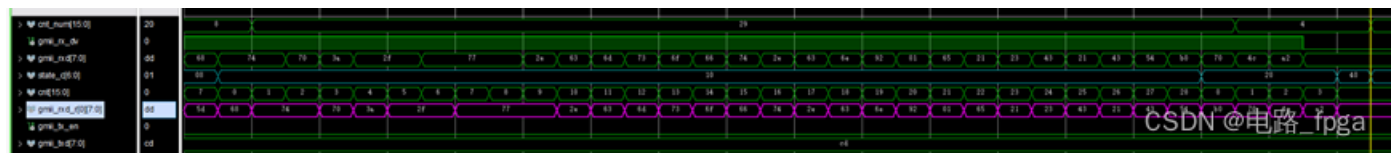


图22 UDP数据段放大

当FPGA接收到UDP数据包后，立马回复一帧UDP数据，如下所示，紫红色信号是接收的数据报文，橙色信号是发送的数据报文。

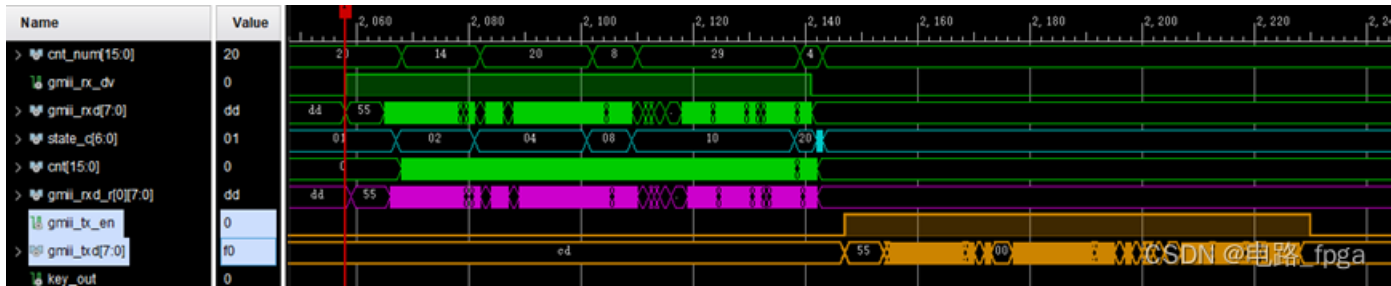


图23 UDP接收和发送报文

将发送报文的数据段放大，结果如下所示，与图17和图20wireshark抓取的数据一致，由此证明该设计接收和发送数据均没有问题。

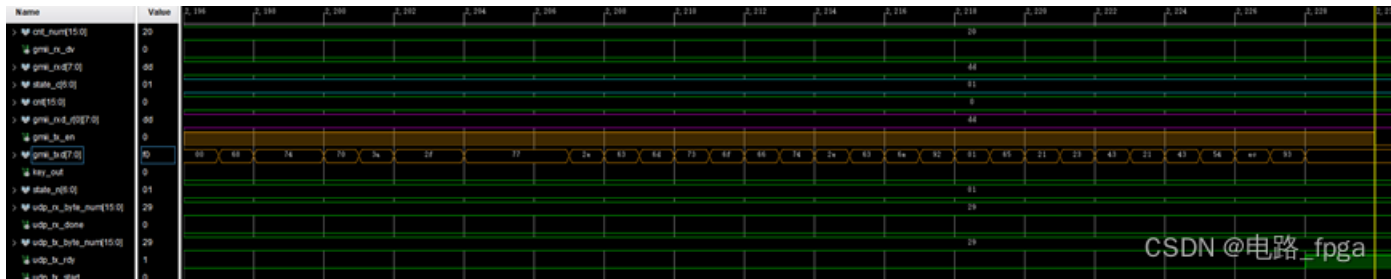


图24 发送报文数据段

最后就是验证ICMP的问题了，直接打开命令提示符，然后输入ping 192.168.1.10指令，运行结果如下所示：

```
管理员: 命令提示符
Microsoft Windows [版本 10.0.22631.3007]
(c) Microsoft Corporation。保留所有权利。

C:\Windows\System32>ping 192.168.1.10

正在 Ping 192.168.1.10 具有 32 字节的数据:
来自 192.168.1.10 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.1.10 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.1.10 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.1.10 的回复: 字节=32 时间<1ms TTL=128

192.168.1.10 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Windows\System32>
```

CSDN @电路\_fpga

图25 ping指令验证

上图表示FPGA接收到PC端的回显请求时，能够向PC端发送回显应答数据报文，以此验证以太网链路是否通畅。

关于UDP的发送和接收本文就做这么多讲解，当然这并不是我们最终想要使用的模块，因为ARP、UDP、ICMP这三个模块其实很多地方都是类似的，使用三个独立的模块完全没有必要，会额外消耗很多资源。

后文会把这三个模块进行整合设计，将模块合成一个eth模块，该模块可以实现对ARP、ICMP、UDP报文的接收，并根据需要发送相应报文。

获取本文工程的方式是在公众号后台回复“**基于FPGA的UDP回环设计**”（不包括引号）。

**您的支持是我更新的最大动力！将持续更新工程，如果本文对您有帮助，还请多多点赞👍、评论💬和收藏★！**