

基于FPGA的实用UDP设计（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、概述

前文对ARP协议、ICMP协议、UDP协议分别做了讲解，并且通过FPGA 实现了三种协议，最终实现的UDP协议工程中也包含了ARP和ICMP协议，对应的总体框架如图所示。

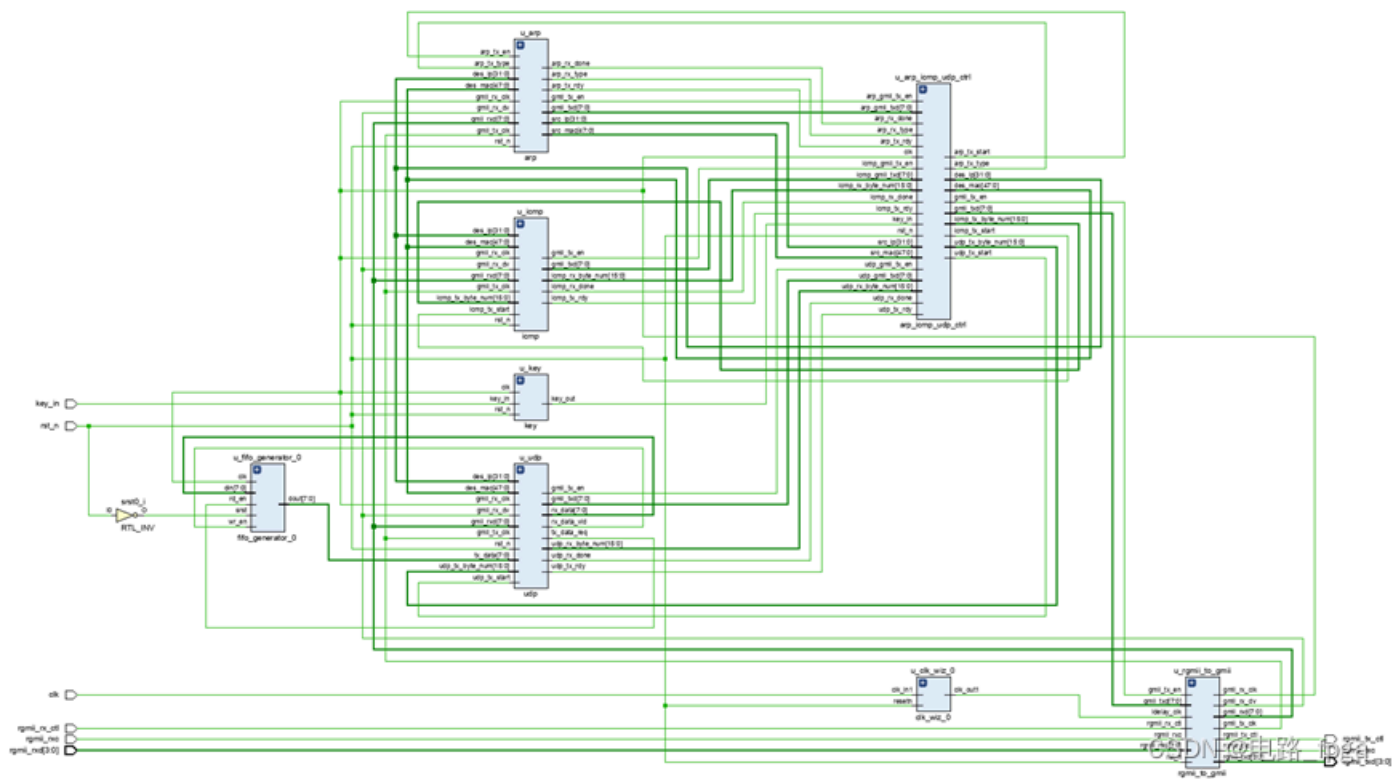


图1 基于FPGA的UDP协议实现

尽管上述模块包含3种协议的接收和发送，但实际上都是通过一个网口收发数据，所以三部分的接收模块和发送模块均只有一个在工作，其余模块均处于空闲状态，造成资源浪费。

所以本文将对这部分内容进行重新设计，最终只会有一个接收数据的模块，能够识别协议类型，进行对应协议的数据解析。也只会存在一个发送模块，通过协议类型指示信号确定具体发送哪种协议。当接收到PC的ARP请求时，依旧会向PC端回复ARP应答指令，不需要用户接口进行干预。FPGA接收到回显请求时，也会自动向PC端发出回显应答指令。当接收到PC端的UDP数据报文时，会将拆包后的数据段输出到用户接口，并且将数据的长度一起输出。当用户需要通过UDP发送数据到PC端时，只需要在发送模块处于空闲时，将

UDP_tx_en拉高，并且把需要发送数据的字节数传输给以太网模块即可，当UDP_tx_req请求数据信号为高电平时，用户在下个时钟周期将需要发送的数据输入以太网模块即可。可以通过拉高用户接口的ARP_req信号，向目的IP地址发出ARP请求。注意该模块的ARP应答和回显应答是不需要外部信号干预的，在模块内部自动完成。

这种设计方式会节省4个CRC校验模块，以及很多计数器和移位寄存器资源，但是控制会稍微复杂一点，主要是涉及的信号比较多，但是最后也是实现了，能够达到上述要求，后续使用比较方便。

当UDP_rx_data_vld有效时，表示接收到UDP数据，并且此时可以根据数据长度信号得知这帧数据的长度。需要发送数据时，也只需要把数据个数输入，将发送使能拉高一个时钟周期，然后等待数据请求信号拉高，之后输入需要发送的数据即可。ARP应答和回显请求用户都不需要关心，所以比较方便。

2、工程设计

本文依旧使用UDP回环进行测试，实现功能与前文一致，主要是对以太网接收和发送模块进行修改，顶层模块连线图如下所示。

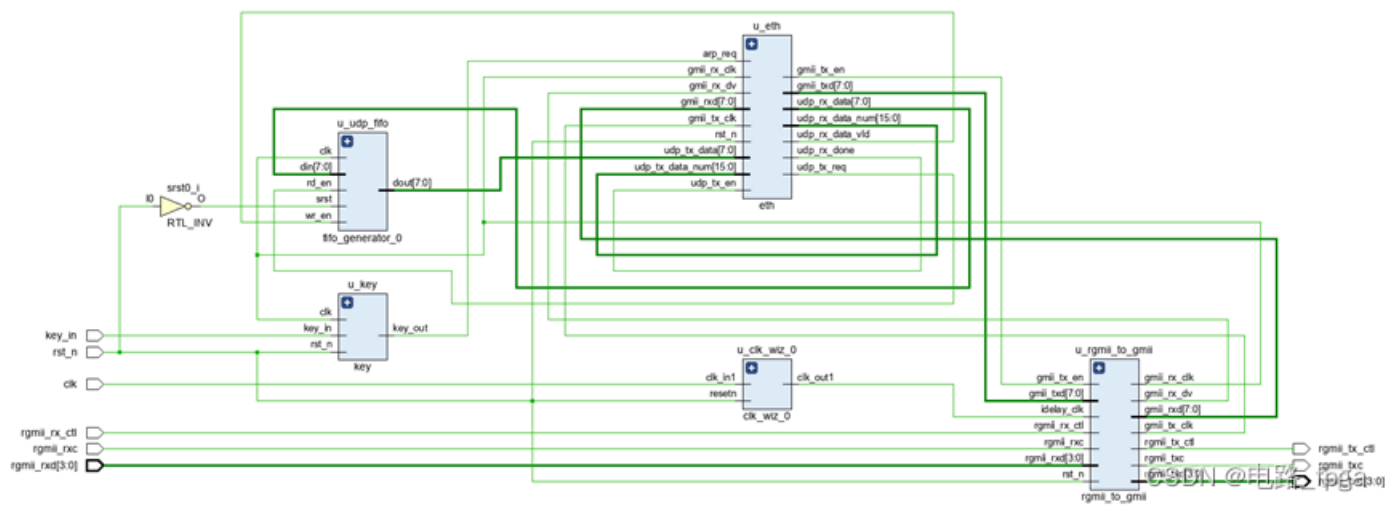


图2 工程顶层模块

工程的顶层模块连接相比图1的工程会简单很多，eth模块可以实现ARP、ICMP、UDP协议的接收和发送，比图1中ARP、ICMP、UDP三个模块实现的功能更复杂，但是开放给用户的接口更加简单。

顶层模块就是将按键消抖模块key、锁相环模块、rgmii转gmii模块、UDP数据暂存FIFO模块、以太网接收发送模块eth的端口进行连线，所以此处就不把其代码贴出来了，需要了解的可以打开工程自行查看。

注意按键模块的输出直接接在ARP_req模块上，按下该按键后，FPGA向目的IP地址发送ARP请求数据报文，获取目的IP地址对应的目的MAC地址，然后作为以太网发送模块的目的MAC地址和目的IP地址。

这里的FIFO用来暂存UDP接收的数据，作为UDP发送模块的数据来源，从而实现UDP回环。

3、以太网模块eth

该模块的设计稍显复杂，对应的框图如下所示，包含以太网接收模块eth_rx、以太网发送模块eth_tx、以太网控制模块eth_ctrl、ICMP回显数据暂存FIFO、两个CRC校验模块。

其中以太网接收模块eth_rx，能够接收ARP、ICMP、UDP的数据报文，将接收到的报文类型输出，如果接收的报文是ARP报文，需要将源MAC地址、源IP地址输出。如果接收的报文是ICMP报文，需要把报文的类型、代码、标识符、序列号以及数据段输出。如果接收的报文是UDP报文，则需要把接收的数据输出到控制模块。由于此处ICMP和UDP都有数据段，但是同一时间又只有会存在一种报文，所以共用同一个数据信号iUDP_rx_data，该数据的具体含义根据此时接收报文的类型eth_rx_type的值确定。

如果接收的报文是ICMP回显请求报文，则将接收的数据存入ICMP FIFO中，便于发送回显应答报文时取用。

以太网发送模块eth_tx，当接收到发送数据使能信号时，根据发送协议类型开始产生对应数据报文。ICMP和UDP均需要从外部取数据，同一时刻只可能发送一种报文，所以也可以共用同一个请求数据输入信号和数据输入信号。这里还需要考虑一个问题，在前文讲解 **ARP协议** 时，讲到过帧间隙，也就是两帧数据之间的最小间隔，发送96位（12字节）数据的时间，为了便于用户使用，所以设计时应该考虑帧间隙问题。

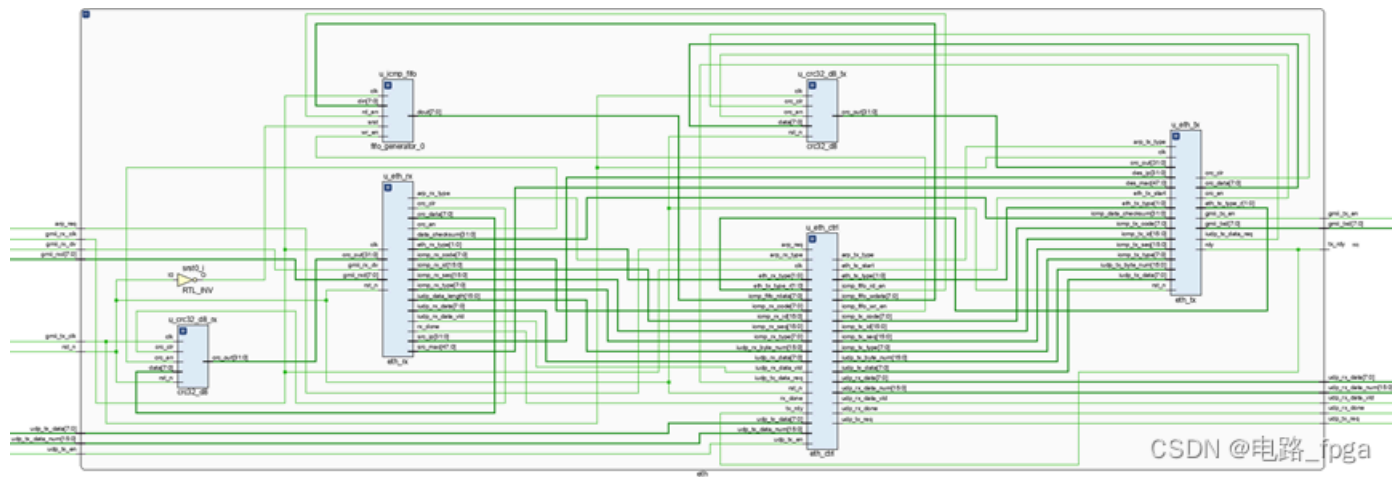


图3 以太网顶层模块

以太网控制模块eth_ctrl：当接收到ARP请求数据报文时，向PC端回复ARP应答数据报文。当用户端口的ARP请求(arq_req)信号拉高时，向PC端发出ARP请求数据报文。接收到ICMP的回显请求数据报文时，将数据段存入ICMP FIFO中，然后向PC端发送回显应答数据报文。如果接收的ICMP指令不是回显请求，目前不做处理，但是会接收该数据报文，不会把数据段存入FIFO中，后续如果需要处理其他ICMP协议，可以添加对应代码即可。当接收到UDP报文时，将数据段输出到用户接口，当用户需要发送UDP报文时，向以太网发送模块产生发送使能信号。该模块还具有仲裁功能，当同时需要发送ARP、ICMP、UDP报文时，依次发送，因为ARP和ICMP报文一般比较短，所以先发送。

顶层模块仅对6个子模块端口连线，所以代码此处就不给出，此处给出顶层模块的两个TestBench文件，一个用于ARP和UDP的仿真，另一个用于ICMP协议的仿真，因为ICMP只对回显请求进行应答，用户接口并没有引出ICMP协议，所以无法开发板无法主动向PC端发出回显请求指令，这也是这个设计的缺陷吧。但是开发板一般不需要发出回显请求，所以对使用不会有影响。

用于对ARP和UDP进行仿真的TestBench文件:

```
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 IP_TYPE     = 16'h0800        ;//16'h0800表示IP协议;
13    localparam ARP_TYPE    = 16'h0806        ;//16'h0806表示ARP协议;
14
15    reg                clk                ;//系统时钟, 默认100MHz;
16    reg                rst_n              ;//系统复位, 默认低电平有效;
17
18    wire                [7 : 0]           gmii_rxd                ;
19    wire                gmii_rx_dv                ;
20    wire                gmii_tx_en                ;
21    wire                [7 : 0]           gmii_txd                ;
22    wire                udp_rx_done                ;
23    wire                [15 : 0]          udp_rx_byte_num         ;
24    wire                [7 : 0]           udp_rx_data            ;
25    wire                [15 : 0]          udp_rx_data_num        ;
26    wire                udp_rx_data_vld            ;
27    wire                tx_rdy                ;
28
29    wire                udp_tx_req                ;
30    reg                [7 : 0]           udp_tx_data            ;
31    reg                [15 : 0]          udp_tx_data_num        ;
32    reg                arp_req                ;
33    reg                udp_tx_en                ;
34
35    assign gmii_rx_dv = gmii_tx_en;
36    assign gmii_rxd = gmii_txd;
37
38    eth #(
39
```

```

39     .BOARD_MAC      ( BOARD_MAC      ),
40     .BOARD_IP       ( BOARD_IP       ),
41     .DES_MAC        ( BOARD_MAC      ),//仿真的时候目的地址也使用开发板地址，不然接收模块不会接收数据；
42     .DES_IP         ( BOARD_IP       ),//仿真的时候目的地址也使用开发板地址，不然接收模块不会接收数据；
43     .BOARD_PORT     ( BOARD_PORT     ),
44     .DES_PORT       ( DES_PORT       ),
45     .IP_TYPE        ( IP_TYPE        ),
46     .ARP_TYPE       ( ARP_TYPE       )
47 )
48
49 u_eth (
50     .rst_n          ( rst_n          ),
51     .gmii_rx_clk    ( clk            ),
52     .gmii_rx_dv     ( gmii_rx_dv     ),
53     .gmii_rxd       ( gmii_rxd       ),
54     .gmii_tx_clk    ( clk            ),
55     .arp_req        ( arp_req        ),
56     .udp_tx_en      ( udp_tx_en      ),
57     .udp_tx_data    ( udp_tx_data    ),
58     .udp_tx_data_num ( udp_tx_data_num ),
59     .gmii_tx_en     ( gmii_tx_en     ),
60     .gmii_txd       ( gmii_txd       ),
61     .tx_rdy         ( tx_rdy         ),
62     .udp_tx_req     ( udp_tx_req     ),
63     .udp_rx_done    ( udp_rx_done    ),
64     .udp_rx_data    ( udp_rx_data    ),
65     .udp_rx_data_num ( udp_rx_data_num ),
66     .udp_rx_data_vld ( udp_rx_data_vld )
67 );
68
69 //生成周期为CYCLE数值的系统时钟；
70 initial begin
71     clk = 0;
72     forever #(CYCLE/2) clk = ~clk;
73 end
74
75 //生成复位信号；
76 initial begin
77     udp_tx_en <= 0; udp_tx_data_num <= 19;udp_tx_data <= 0;arp_req <= 0;
78     rst_n <= 1;
79     #2;
80

```

```

80      rst_n <= 0;//开始时复位10个时钟;
81      #(RST_TIME*CYCLE);
82      rst_n <= 1;
83      #(20*CYCLE);
84      repeat(1)begin
85          @(posedge clk);
86          arp_req <= 1'b1;
87          @(posedge clk);
88          arp_req <= 1'b0;
89          repeat(10)@(posedge clk);
90          arp_req <= 1'b1;
91          @(posedge clk);
92          arp_req <= 1'b0;
93      end
94      @(posedge tx_rdy);
95      repeat(10)@(posedge clk);
96      repeat(7)begin
97          udp_tx_en <= 1'b1;
98          udp_tx_data_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;
99          @(posedge clk);
100          udp_tx_en <= 1'b0;
101          @(posedge tx_rdy);
102          repeat(30)@(posedge clk);
103      end
104      #(20*CYCLE);
105      $stop;//停止仿真;
106  end
107
108  always@(posedge clk)begin
109      if(udp_tx_req)begin//产生0~255随机数作为测试;
110          udp_tx_data <= {$random} % 256;
111      end
112  end
113
endmodule

```



用于对ICMP仿真的TestBench文件:

```
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 IP_TYPE     = 16'h0800        ;//16'h0800表示IP协议;
13    localparam ARP_TYPE    = 16'h0806        ;//16'h0806表示ARP协议;
14
15    reg                clk                ;//系统时钟, 默认100MHz;
16    reg                rst_n              ;//系统复位, 默认低电平有效;
17
18    reg                [7 : 0]            gmii_rxd ;
19    reg                gmii_rx_dv         ;
20    wire               gmii_tx_en         ;
21    wire               [7 : 0]            gmii_txd ;
22    wire               udp_rx_done        ;
23    wire               [15 : 0]           udp_rx_byte_num ;
24    wire               [7 : 0]            udp_rx_data ;
25    wire               [15 : 0]           udp_rx_data_num ;
26    wire               udp_rx_data_vld    ;
27    wire               tx_rdy             ;
28
29    wire               udp_tx_req          ;
30    reg                [7 : 0]            udp_tx_data ;
31    wire               [15 : 0]           udp_tx_data_num ;
32    wire               udp_tx_en          ;
33    reg                [7 : 0]            rx_data [255 : 0] ;//申请256个数据的存储器
34
35    assign udp_tx_data_num = udp_rx_data_num;
36    assign udp_tx_en = udp_rx_done;
37
38    eth #(
39
```

```

39     .BOARD_MAC      ( BOARD_MAC      ),
40     .BOARD_IP       ( BOARD_IP       ),
41     .DES_MAC        ( BOARD_MAC      ),//仿真的时候目的地址也使用开发板地址，不然接收模块不会接收数据；
42     .DES_IP         ( BOARD_IP       ),//仿真的时候目的地址也使用开发板地址，不然接收模块不会接收数据；
43     .BOARD_PORT     ( BOARD_PORT     ),
44     .DES_PORT       ( DES_PORT       ),
45     .IP_TYPE        ( IP_TYPE        ),
46     .ARP_TYPE       ( ARP_TYPE       )
47 )
48 u_eth (
49     .rst_n           ( rst_n           ),
50     .gmii_rx_clk     ( clk             ),
51     .gmii_rx_dv      ( gmii_rx_dv     ),
52     .gmii_rxd        ( gmii_rxd       ),
53     .gmii_tx_clk     ( clk             ),
54     .arp_req         ( 1'b0           ),
55     .udp_tx_en       ( udp_tx_en       ),
56     .udp_tx_data     ( udp_tx_data     ),
57     .udp_tx_data_num ( udp_tx_data_num ),
58     .gmii_tx_en      ( gmii_tx_en     ),
59     .gmii_txd        ( gmii_txd       ),
60     .tx_rdy          ( tx_rdy          ),
61     .udp_tx_req      ( udp_tx_req      ),
62     .udp_rx_done     ( udp_rx_done     ),
63     .udp_rx_data     ( udp_rx_data     ),
64     .udp_rx_data_num ( udp_rx_data_num ),
65     .udp_rx_data_vld ( udp_rx_data_vld )
66 );
67
68 reg          crc_clr          ;
69 reg          gmii_crc_vld     ;
70 reg          gmii_rxd_r       [7 : 0] ;
71 reg          gmii_rx_dv_r     ;
72 reg          crc_data_vld     ;
73 reg          i                [9 : 0] ;
74 reg          num              [15 : 0] ;
75 wire        [31 : 0]         crc_out ;
76
77 //生成周期为CYCLE数值的系统时钟；
78 initial begin
79
80

```



```

80         clk = 0;
81         forever #(CYCLE/2) clk = ~clk;
82     end
83
84     //生成复位信号;
85     initial begin
86         num <= 0;
87         crc_clr <= 0;
88         gmii_rxd <= 0;
89         gmii_rx_dv <= 0;
90         gmii_rxd_r <= 0;
91         gmii_rx_dv_r <= 0;
92         gmii_crc_vld <= 1'b0;
93         for(i = 0 ; i < 256 ; i = i + 1)begin
94             rx_data[i] <= {$random} % 256;//初始化存储体;
95             #1;
96         end
97         rst_n <= 1;
98         #2;
99         rst_n <= 0;//开始时复位10个时钟;
100        repeat(RST_TIME) @(posedge clk);
101        rst_n <= 1;
102        repeat(20) @(posedge clk);
103        repeat(4)begin//发送2帧数据;
104            gmii_tx_test({$random} % 64 + 18);
105            #1;
106            gmii_crc_vld <= 1'b1;
107            gmii_rxd_r <= crc_out[7 : 0];
108            @(posedge clk);
109            gmii_rxd_r <= crc_out[15 : 8];
110            @(posedge clk);
111            gmii_rxd_r <= crc_out[23 : 16];
112            @(posedge clk);
113            gmii_rxd_r <= crc_out[31 : 24];
114            @(posedge clk);
115            gmii_crc_vld <= 1'b0;
116            crc_clr <= 1'b1;
117            @(posedge clk);
118            crc_clr <= 1'b0;
119            repeat(50) @(posedge clk);
120

```

```

121     end
122     repeat(20) @(posedge clk);
123     $stop;//停止仿真:
124 end
125
126 task gmii_tx_test(
127     input [15 : 0]  data_num    //需要把多少个存储体中的数据进行发送，取值范围[18,255];
128 );
129     reg [31 : 0] ip_check;
130     reg [15 : 0] total_num;
131     reg [31 : 0] icmp_check;
132     begin
133         total_num <= data_num + 28;
134         #1;
135         icmp_check <= 16'h1 + 16'h8;//ICMP首部相加;
136         ip_check <= DES_IP[15:0] + BOARD_IP[15:0] + DES_IP[31:16] + BOARD_IP[31:16] + 16'h4500 + total_num + 16'h4000 + num + 16'h8001;
137         if(~data_num[0])begin//ICMP数据段个数为偶数:
138             for(i=0 ; 2*i < data_num ; i= i+1)begin
139                 #1;//计算ICMP数据段的校验和。
140                 icmp_check <= icmp_check + {rx_data[i][7:0],rx_data[i+1][7:0]};
141             end
142         end
143         else begin//ICMP数据段个数为奇数:
144             for(i=0 ; 2*i < data_num+1 ; i = i+1)begin
145                 //计算ICMP数据段的校验和。
146                 if(2*i + 1 == data_num)
147                     icmp_check <= icmp_check + {rx_data[i][7:0]};
148                 else
149                     icmp_check <= icmp_check + {rx_data[i][7:0],rx_data[i+1][7:0]};
150             end
151         end
152         crc_data_vld <= 1'b0;
153         @(posedge clk);
154         repeat(7)begin//发送前导码7个8'H55;
155             gmii_rxd_r <= 8'h55;
156             gmii_rx_dv_r <= 1'b1;
157             @(posedge clk);
158         end
159         gmii_rxd_r <= 8'hd5;//发送SFD，一个字节的8'hd5;
160         @(posedge clk);
161

```

```

162     crc_data_vld <= 1'b1;
163     //发送以太网帧头数据;
164     for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的MAC地址;
165         gmii_rxd_r <= BOARD_MAC[47-8*i -: 8];
166         @(posedge clk);
167     end
168     for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的源MAC地址;
169         gmii_rxd_r <= DES_MAC[47-8*i -: 8];
170         @(posedge clk);
171     end
172     for(i=0 ; i<2 ; i=i+1)begin//发送2个字节的以太网类型;
173         gmii_rxd_r <= IP_TYPE[15-8*i -: 8];
174         @(posedge clk);
175     end
176     //发送IP帧头数据;
177     gmii_rxd_r <= 8'H45;
178     @(posedge clk);
179     gmii_rxd_r <= 8'd00;
180     ip_check <= ip_check[15 : 0] + ip_check[31:16];
181     icmp_check <= icmp_check[15 : 0] + icmp_check[31:16];
182     @(posedge clk);
183     gmii_rxd_r <= total_num[15:8];
184     ip_check <= ip_check[15 : 0] + ip_check[31:16];
185     icmp_check <= icmp_check[15 : 0] + icmp_check[31:16];
186     @(posedge clk);
187     gmii_rxd_r <= total_num[7:0];
188     ip_check[15 : 0] <= ~ip_check[15 : 0];
189     icmp_check <= ~icmp_check[15 : 0];
190     @(posedge clk);
191     gmii_rxd_r <= num[15:8];
192     @(posedge clk);
193     gmii_rxd_r <= num[7:0];
194     @(posedge clk);
195     gmii_rxd_r <= 8'h40;
196     @(posedge clk);
197     gmii_rxd_r <= 8'h00;
198     @(posedge clk);
199     gmii_rxd_r <= 8'h80;
200     @(posedge clk);
201     gmii_rxd_r <= 8'h01;
202

```

```

203         @(posedge clk);
204         gmii_rxd_r <= ip_check[15:8];
205         @(posedge clk);
206         gmii_rxd_r <= ip_check[7:0];
207         @(posedge clk);
208         for(i=0 ; i<4 ; i=i+1)begin//发送6个字节的源IP地址;
209             gmii_rxd_r <= DES_IP[31-8*i -: 8];
210             @(posedge clk);
211         end
212         for(i=0 ; i<4 ; i=i+1)begin//发送4个字节的IP地址;
213             gmii_rxd_r <= BOARD_IP[31-8*i -: 8];
214             @(posedge clk);
215         end
216         //发送ICMP帧头及数据包;
217         gmii_rxd_r <= 8'h08;//发送回显请求。
218         @(posedge clk);
219         gmii_rxd_r <= 8'h00;
220         @(posedge clk);
221         gmii_rxd_r <= icmp_check[31:16];
222         @(posedge clk);
223         gmii_rxd_r <= icmp_check[15:0];
224         @(posedge clk);
225         gmii_rxd_r <= 8'h00;
226         @(posedge clk);
227         gmii_rxd_r <= 8'h01;
228         @(posedge clk);
229         gmii_rxd_r <= 8'h00;
230         @(posedge clk);
231         gmii_rxd_r <= 8'h08;
232         @(posedge clk);
233         for(i=0 ; i<data_num ; i=i+1)begin
234             gmii_rxd_r <= rx_data[i];
235             @(posedge clk);
236         end
237         crc_data_vld <= 1'b0;
238         gmii_rx_dv_r <= 1'b0;
239         num = num + 1;
240     end
241 endtask
242
243

```

```

244     crc32_d8  u_crc32_d8_1 (
245         .clk      ( clk      ),
246         .rst_n     ( rst_n     ),
247         .data      ( gmii_rxd_r ),
248         .crc_en     ( crc_data_vld ),
249         .crc_clr    ( crc_clr    ),
250         .crc_out    ( crc_out    )
251     );
252
253     always@(posedge clk)begin
254         if(rst_n==1'b0)begin//初始值为0;
255             gmii_rxd <= 8'd0;
256             gmii_rx_dv <= 1'b0;
257         end
258         else if(gmii_rx_dv_r || gmii_crc_vld)begin
259             gmii_rxd <= gmii_rxd_r;
260             gmii_rx_dv <= 1'b1;
261         end
262         else begin
263             gmii_rx_dv <= 1'b0;
264         end
265     end
266
267 endmodule

```



4、以太网接收模块

首先查看ARP、ICMP、UDP协议的数据帧格式，如下图所示：

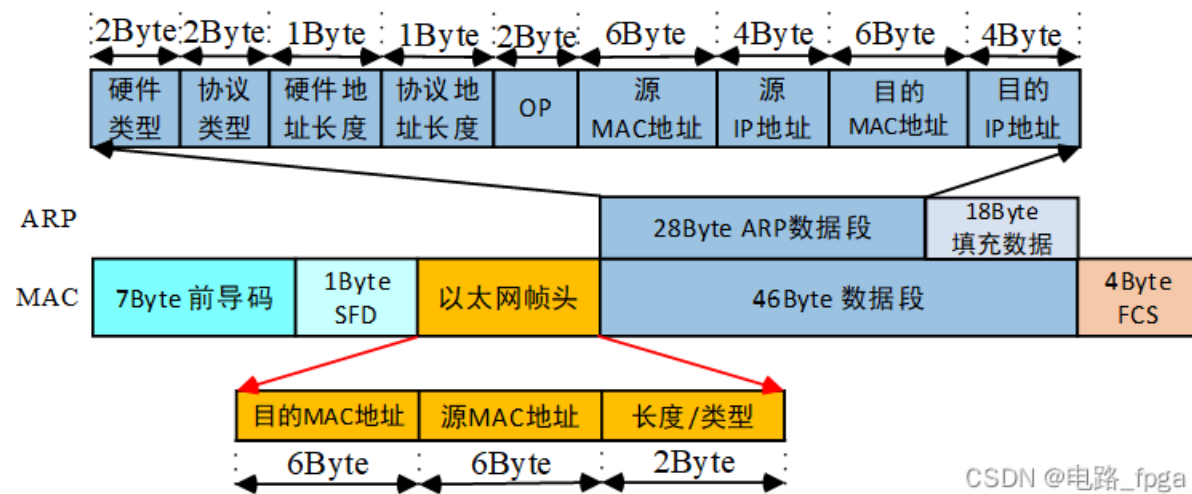


图4 ARP数据帧格式

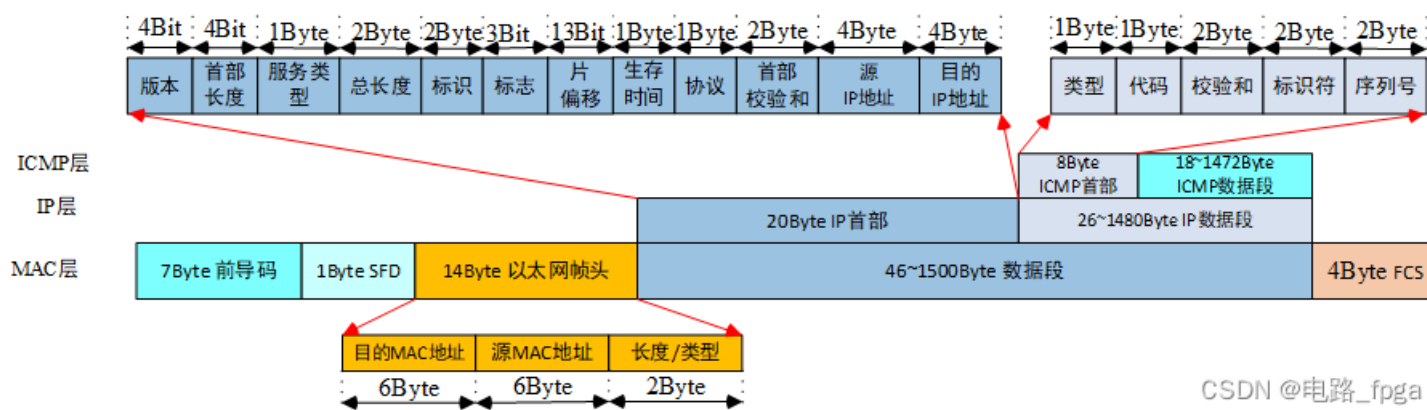


图5 ICMP数据帧格式

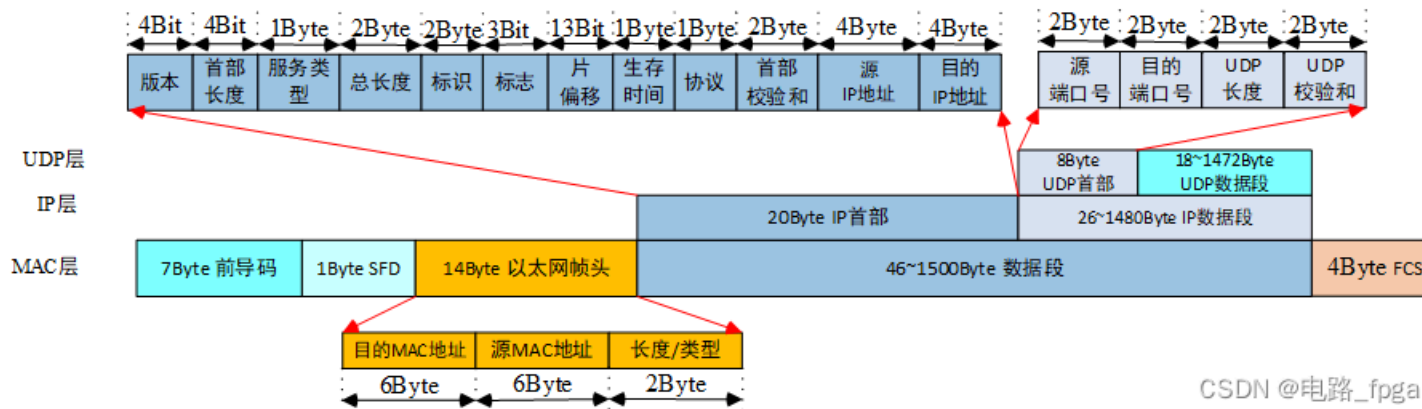


图6 UDP数据帧格式

通过对比上面三图可知，前导码、帧起始符、以太网帧头、CRC校验码均一致，所以这几部分还是可以根据前文一样设计，不做改变。当以太网帧头的类型为16'h0806时表示ARP，等于16'h0800时表示报文是IP协议。此时还是可以通过状态机进行实现，对应的状态转换图如下所示：

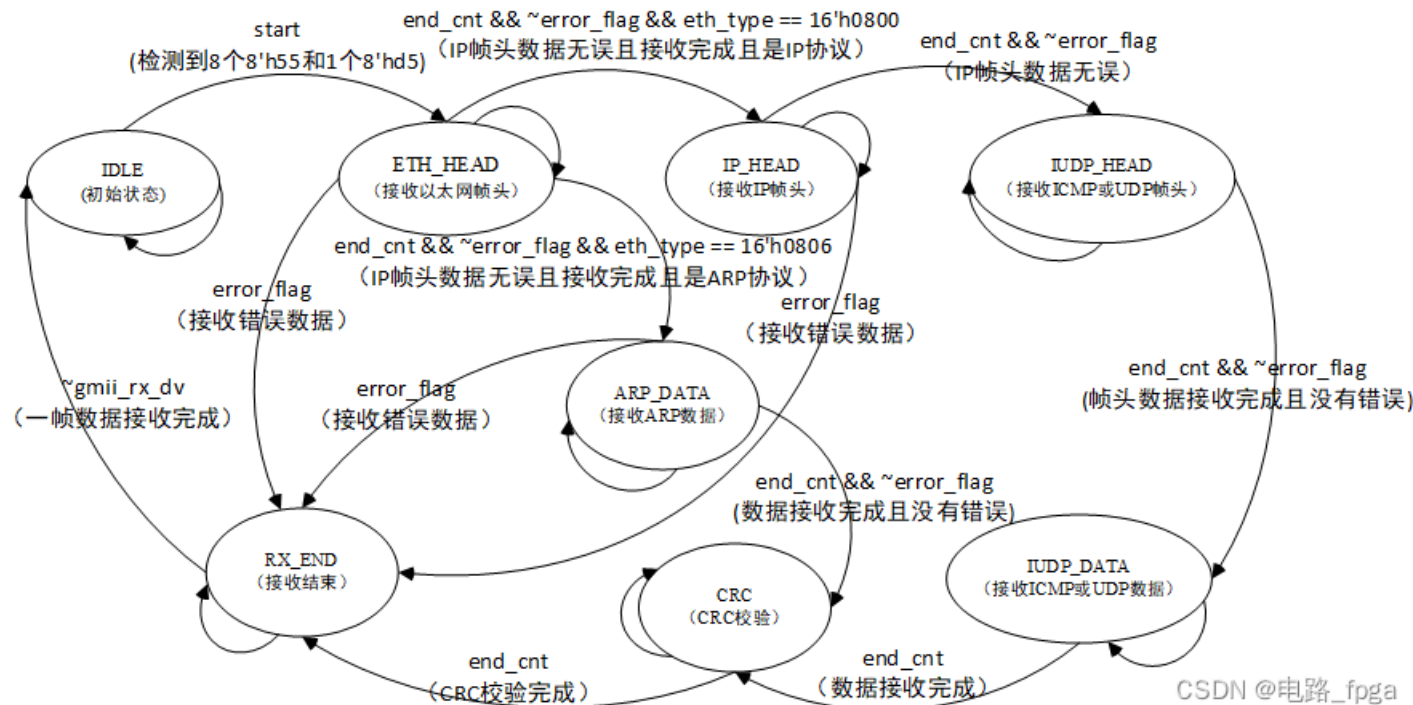


图7 以太网接收模块状态转换图


```

--
19 reg          [7 : 0]      state_c          ;//状态机现态。
20 reg          [15 : 0]     cnt              ;//计数器，辅助状态机的跳转。
21 reg          [15 : 0]     cnt_num          ;//计数器的状态机每个状态下接收数据的个数。
22 reg          [5 : 0]      ip_head_byte_num ;//IP首部数据的字节数。
23 reg          [15 : 0]     ip_total_length  ;//IP报文总长度。
24 reg          [15 : 0]     des_ip           ;//目的IP地址。
25 reg          [7 : 0]      gmii_rxd_r      [6 : 0] ;//接收信号的移位寄存器；
26 reg          [6 : 0]      gmii_rx_dv_r    ;
27 reg          [23 : 0]     des_crc          ;//接收的CRC校验数值；
28 reg          [47 : 0]     des_mac         ;
29 reg          [15 : 0]     opcode          ;
30 reg          [47 : 0]     src_mac_t       ;
31 reg          [31 : 0]     src_ip_t        ;
32 reg          [31 : 0]     reply_checksum_add ;
33
34 wire          add_cnt      ;
35 wire          end_cnt     ;
36
37 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
38 always@(posedge clk)begin
39     if(!rst_n)begin
40         state_c <= IDLE;
41     end
42     else begin
43         state_c <= state_n;
44     end
45 end
46
47 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
48 always@(*)begin
49     case(state_c)
50         IDLE:begin
51             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。
52                 state_n = ETH_HEAD;
53             end
54             else begin
55                 state_n = state_c;
56             end
57         end
58         ETH_HEAD:begin
59

```

```

55         if(error_flag)begin//在接收以太网帧头过程中检测到错误。
60             state_n = RX_END;
61         end
62     else if(end_cnt)begin//接收完以太网帧头数据，且没有出现错误。
63         if(eth_rx_type == 2'd1)//如果该数据报是ARP类型，则跳转到ARP接收数据状态；
64             state_n = ARP_DATA;
65         else//否则跳转到接收IP报头的状态；
66             state_n = IP_HEAD;
67         end
68     else begin
69         state_n = state_c;
70     end
71 end
72 IP_HEAD:begin
73     if(error_flag)begin//在接收IP帧头过程中检测到错误。
74         state_n = RX_END;
75     end
76     else if(end_cnt)begin//接收完以IP帧头数据，且没有出现错误。
77         state_n = IUDP_HEAD;//跳转到接收ICMP或UDP报头状态；
78     end
79     else begin
80         state_n = state_c;
81     end
82 end
83 IUDP_HEAD:begin
84     if(end_cnt)begin//接收完以ICMP帧头或UDP帧头数据，则继续接收ICMP数据或UDP数据。
85         state_n = IUDP_DATA;
86     end
87     else begin
88         state_n = state_c;
89     end
90 end
91 IUDP_DATA:begin
92     if(end_cnt)begin//接收完ICMP数据或UDP数据，跳转到CRC校验状态。
93         state_n = CRC;
94     end
95     else begin
96         state_n = state_c;
97     end
98 end
99 end

```

```

100     ARP_DATA:begin
101         if(error_flag)begin//接收数据出现错误。
102             state_n = RX_END;
103         end
104         else if(end_cnt)begin//接收完所有数据。
105             state_n = CRC;
106         end
107         else begin
108             state_n = state_c;
109         end
110     end
111     CRC:begin
112         if(end_cnt)begin//接收完CRC校验数据。
113             state_n = RX_END;
114         end
115         else begin
116             state_n = state_c;
117         end
118     end
119     RX_END:begin
120         if(~gmii_rx_dv)begin//检测到数据线上数据无效。
121             state_n = IDLE;
122         end
123         else begin
124             state_n = state_c;
125         end
126     end
127     default:begin
128         state_n = IDLE;
129     end
130 endcase
131 end
132
133 //将输入数据保存6个时钟周期，用于检测前导码和SFD。
134 //注意后文的state_c与gmii_rxd_r[0]对齐。
135 always@(posedge clk)begin
136     gmii_rxd_r[6] <= gmii_rxd_r[5];
137     gmii_rxd_r[5] <= gmii_rxd_r[4];
138     gmii_rxd_r[4] <= gmii_rxd_r[3];
139     gmii_rxd_r[3] <= gmii_rxd_r[2];
140

```

```

141     gmii_rxd_r[2] <= gmii_rxd_r[1];
142     gmii_rxd_r[1] <= gmii_rxd_r[0];
143     gmii_rxd_r[0] <= gmii_rxd;
144     gmii_rx_dv_r <= {gmii_rx_dv_r[5 : 0],gmii_rx_dv};
145 end
146
147 //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。
148 always@(posedge clk)begin
149     if(rst_n==1'b0)begin//初始值为0;
150         start <= 1'b0;
151     end
152     else if(state_c == IDLE)begin
153         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],
154     end
155 end
156
157 //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。
158 always@(posedge clk)begin
159     if(rst_n==1'b0)begin//
160         cnt <= 0;
161     end
162     else if(add_cnt)begin
163         if(end_cnt)
164             cnt <= 0;
165         else
166             cnt <= cnt + 1;
167     end
168     else begin//如果加一条件无效，计数器必须清零。
169         cnt <= 0;
170     end
171 end
172 //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。
173 assign add_cnt = (state_c != IDLE) && (state_c != RX_END) && gmii_rx_dv_r[0];
174 assign end_cnt = add_cnt && cnt == cnt_num - 1;
175
176 //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。
177 always@(posedge clk)begin
178     if(rst_n==1'b0)begin//初始值为20;
179         cnt_num <= 16'd20;
180     end
181

```

```

182     else begin
183         case(state_c)
184             ETH_HEAD : cnt_num <= 16'd14; //以太网帧头长度位14字节。
185             IP_HEAD  : cnt_num <= ip_head_byte_num; //IP帧头为20字节数据。
186             IUDP_HEAD : cnt_num <= 16'd8; //UDP和ICMP帧头为8字节数据。
187             IUDP_DATA : cnt_num <= iudp_data_length; //UDP数据段需要根据数据长度进行变化。
188             ARP_DATA  : cnt_num <= 16'd46; //ARP数据段46字节。
189             CRC        : cnt_num <= 16'd4; //CRC校验为4字节数据。
190             default: cnt_num <= 16'd20;
191         endcase
192     end
193 end
194
195 //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。
196 always@(posedge clk)begin
197     if(rst_n==1'b0)begin//初始值为0;
198         des_mac <= 48'd0;
199     end
200     else if((state_c == ETH_HEAD) && add_cnt && cnt < 16'd6)begin
201         des_mac <= {des_mac[39:0],gmii_rxd_r[0]};
202     end
203 end
204
205 //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。
206 always@(posedge clk)begin
207     if(rst_n==1'b0)begin//初始值为0;
208         error_flag <= 1'b0;
209     end
210     else if(add_cnt)begin
211         case(state_c)
212             ETH_HEAD : begin
213                 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。
214                     error_flag <= ((des_mac != BOARD_MAC) && (des_mac != 48'HFF_FF_FF_FF_FF));
215                 else if(cnt == 12)//接收的数据报不是IP协议且不是ARP协议。
216                     error_flag <= ({gmii_rxd_r[0],gmii_rxd} != IP_TPYE) && ({gmii_rxd_r[0],gmii_rxd} != ARP_TPYE);
217             end
218             IP_HEAD : begin
219                 if(cnt == 9)//如果当前接收的数据不是UDP协议，且不是ICMP协议;
220                     error_flag <= (gmii_rxd_r[0] != UDP_TYPE) && (gmii_rxd_r[0] != ICMP_TYPE);
221                 else if(cnt == 16'd18)//判断目的IP地址是否为开发板的IP地址。
222

```

```

223         error_flag <= ({des_ip,gmii_rxd_r[0],gmii_rxd} != BOARD_IP);
224     end
225     ARP_DATA : begin
226         if(cnt == 27)begin//判断接收的目的IP地址是否正确，操作码是否为ARP的请求或应答指令。
227             error_flag <= ((opcode != 16'd1) && (opcode != 16'd2)) || ({des_ip,gmii_rxd_r[1],gmii_rxd_r[0]} != BOARD_IP);
228         end
229     end
230     IUDP_DATA : begin
231         if((cnt == 3) && (eth_rx_type == 2'd3))begin//UDP的目的端口地址不等于开发板的目的端口地址。
232             error_flag <= ({gmii_rxd_r[1],gmii_rxd_r[0]} != BOARD_PORT);
233         end
234     end
235     default: error_flag <= 1'b0;
236 endcase
237 end
238 else begin
239     error_flag <= 1'b0;
240 end
241 end
242
243 //根据接收的数据判断该数据报的类型。
244 always@(posedge clk)begin
245     if(rst_n==1'b0)begin//初始值为0;
246         eth_rx_type <= 2'd0;
247     end//接收的协议是ARP协议;
248     else if(state_c == ETH_HEAD && add_cnt && cnt == 12)begin
249         if({gmii_rxd_r[0],gmii_rxd} == ARP_TPYE)begin
250             eth_rx_type <= 1;
251         end
252         else begin
253             eth_rx_type <= 0;
254         end
255     end
256     else if(state_c == IP_HEAD && add_cnt && cnt == 9)begin
257         if(gmii_rxd_r[0] == UDP_TYPE)//接收的数据包是UDP协议;
258             eth_rx_type <= 3;
259         else if(gmii_rxd_r[0] == ICMP_TYPE)//接收的协议是ICMP协议;
260             eth_rx_type <= 2;
261     end
262 end
263

```

```

264
265 //接收IP首部和ARP数据段的数据。
266 always@(posedge clk)begin
267     if(rst_n==1'b0)begin//初始值为0;
268         ip_head_byte_num <= 6'd20;
269         ip_total_length <= 16'd28;
270         des_ip <= 16'd0;
271         iudp_data_length <= 16'd0;
272         opcode <= 16'd0;//ARP的OP编码。
273         src_mac_t <= 48'd0;//ARP传输的源MAC地址;
274         src_ip_t <= 32'd0;//ARP传输的源IP地址;
275     end
276     else if(state_c == IP_HEAD && add_cnt)begin
277         case(cnt)
278             16'd0 : ip_head_byte_num <= {gmii_rxd_r[0][3:0],2'd0};//接收IP首部的字节个数。
279             16'd3 : ip_total_length <= {gmii_rxd_r[1],gmii_rxd_r[0]};//接收IP报文总长度的低八位数据。
280             16'd4 : iudp_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。
281             16'd17: des_ip <= {gmii_rxd_r[1],gmii_rxd_r[0]};//接收目的IP地址。
282             default: ;
283         endcase
284     end
285     else if(state_c == ARP_DATA && add_cnt)begin
286         case(cnt)
287             16'd7 : opcode <= {gmii_rxd_r[1],gmii_rxd_r[0]};//操作码;
288             16'd13 : src_mac_t <= {gmii_rxd_r[5],gmii_rxd_r[4],gmii_rxd_r[3],gmii_rxd_r[2],gmii_rxd_r[1],gmii_rxd_r[0]};//源MAC地址;
289             16'd17 : src_ip_t <= {gmii_rxd_r[3],gmii_rxd_r[2],gmii_rxd_r[1],gmii_rxd_r[0]};//源IP地址;
290             16'd25 : des_ip <= {gmii_rxd_r[1],gmii_rxd_r[0]};//接收目的IP地址高16位。
291             default: ;
292         endcase
293     end
294 end
295
296 //接收ICMP首部相关数据，UDP首部数据不需要保存。
297 always@(posedge clk)begin
298     if(rst_n==1'b0)begin//初始值为0;
299         icmp_rx_type <= 8'd0;//ICMP类型;
300         icmp_rx_code <= 8'd0;//ICMP代码;
301         icmp_rx_id <= 16'd0;//ICMP标识符
302         icmp_rx_seq <= 16'd0;//ICMP请求;
303     end
304

```

```

305     else if(state_c == IUDP_HEAD && add_cnt)begin
306         if(eth_rx_type == 2'd2)//如果是ICMP协议。
307             case(cnt)
308                 16'd0 : icmp_rx_type <= gmii_rxd_r[0];//接收ICMP报文类型。
309                 16'd1 : icmp_rx_code <= gmii_rxd_r[0];//接收ICMP报文代码。
310                 16'd5 : icmp_rx_id <= {gmii_rxd_r[1],gmii_rxd_r[0]};//接收ICMP的ID。
311                 16'd7 : icmp_rx_seq <= {gmii_rxd_r[1],gmii_rxd_r[0]};//接收ICMP报文的序列号。
312                 default: ;
313             endcase
314         end
315     end
316
317     //接收ICMP或者UDP的数据段，并输出使能信号。
318     always@(posedge clk)begin
319         iudp_rx_data <= (state_c == IUDP_DATA) ? gmii_rxd_r[0] : iudp_rx_data;//在接收UDP数据阶段时，接收数据。
320         iudp_rx_data_vld <= (state_c == IUDP_DATA);//在接收数据阶段时，将数据输出。
321     end
322
323     //生产CRC校验相关的数据和控制信号。
324     always@(posedge clk)begin
325         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。
326         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。
327         crc_en <= (state_c != IDLE) && (state_c != RX_END) && (state_c != CRC);//CRC校验使能信号。
328     end
329
330     //接收PC端发送来的CRC数据。
331     always@(posedge clk)begin
332         if(rst_n==1'b0)begin//初始值为0;
333             des_crc <= 24'hff_ff_ff;
334         end
335         else if(add_cnt && state_c == CRC)begin//先接收的是低位数据;
336             des_crc <= {gmii_rxd_r[0],des_crc[23:8]};
337         end
338     end
339
340     //计算接收到的ICMP数据段校验和。
341     always@(posedge clk)begin
342         if(rst_n==1'b0)begin//初始值为0;
343             reply_checksum_add <= 32'd0;
344         end
345

```



```

346     else if(state_c == RX_END)begin//累加器清零。
347         reply_checksum_add <= 32'd0;
348     end
349     else if(state_c == IUDP_DATA && add_cnt && eth_rx_type == 2'd2)begin
350         if(end_cnt && iudp_data_length[0])begin//如果计数器计数结束且数据个数为奇数个(最低位为1)，那么直接将当前数据与累加器相加。
351             reply_checksum_add <= reply_checksum_add + {8'd0,gmii_rxd_r[0]};
352         end
353         else if(cnt[0])//计数器计数到奇数时，将前后两字节数据拼接相加。
354             reply_checksum_add <= reply_checksum_add + {gmii_rxd_r[1],gmii_rxd_r[0]};
355     end
356 end
357
358 //生成相应的输出数据。
359 always@(posedge clk)begin
360     if(rst_n==1'b0)begin//初始值为0;
361         rx_done <= 1'b0;//接收一帧数据完成信号，高电平有效;
362         src_mac <= 48'd0;//ARP接收的源MAC地址;
363         src_ip <= 32'd0;//ARP接收的源IP地址;
364         arp_rx_type <= 1'b0;
365         data_checksum <= 32'd0;//ICMP数据段校验和;
366     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。
367     else if(state_c == CRC && end_cnt && ({gmii_rxd_r[0],des_crc[23:0]} == crc_out))begin//CRC校验无误。
368         if(eth_rx_type == 2'd1)begin//如果接收的是ARP协议;
369             src_mac <= src_mac_t;//将接收的源MAC地址输出;
370             src_ip <= src_ip_t;//将接收的源IP地址输出;
371             arp_rx_type <= (opcode == 16'd1) ? 1'b0 : 1'b1;//接收ARP数据报的类型;
372         end
373         else begin//如果接收的协议是IP协议;
374             data_checksum <= (eth_rx_type == 2'd2) ? reply_checksum_add : data_checksum;//如果是ICMP，需要计算数据段的校验和。
375         end
376         rx_done <= 1'b1;//将接收一帧数据完成信号拉高一个时钟周期;
377     end
378     else begin
379         rx_done <= 1'b0;
380     end
381 end

```



该模块仿真结果如下所示，接收ARP协议：

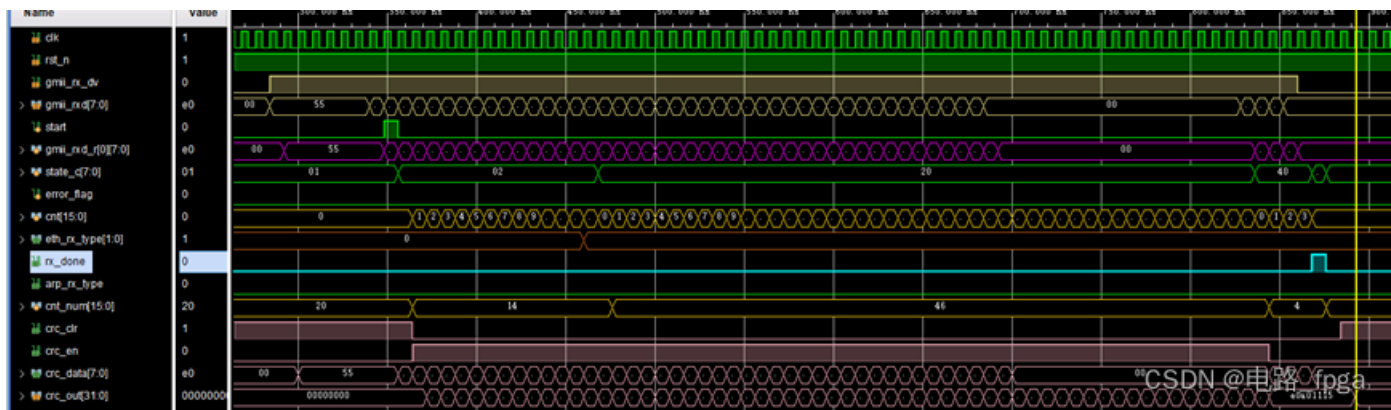


图8 接收ARP报文

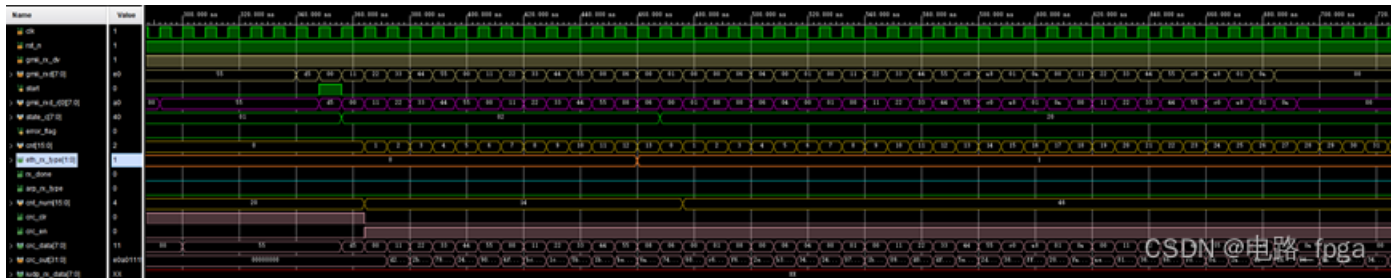


图9 ARP报文放大

接收ICMP协议：

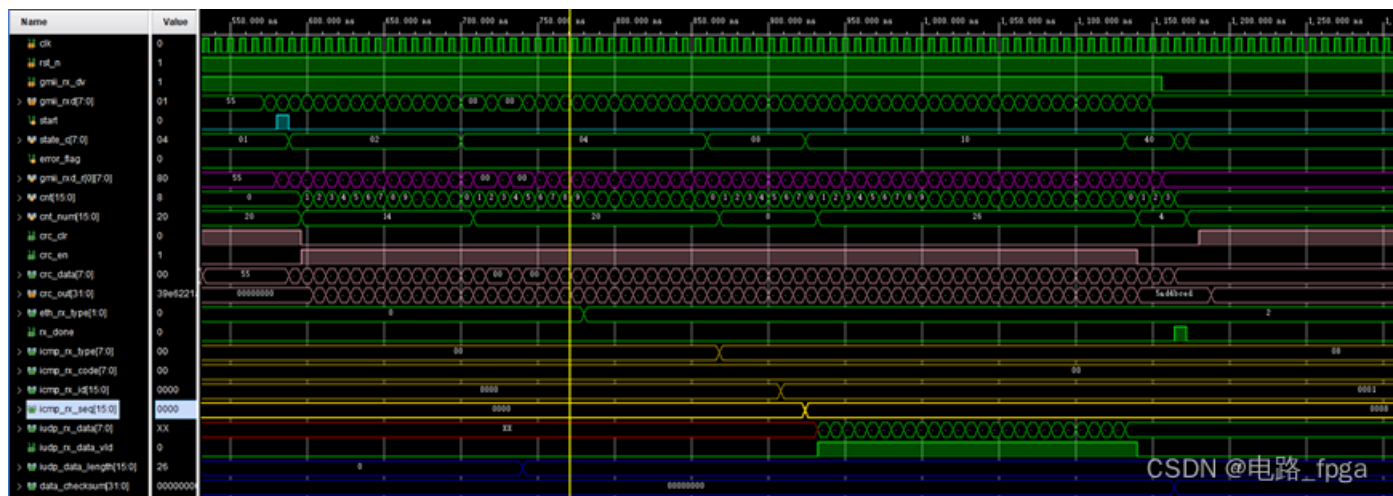


图10 接收ICMP报文

把ICMP的数据段放大，如下图所示：

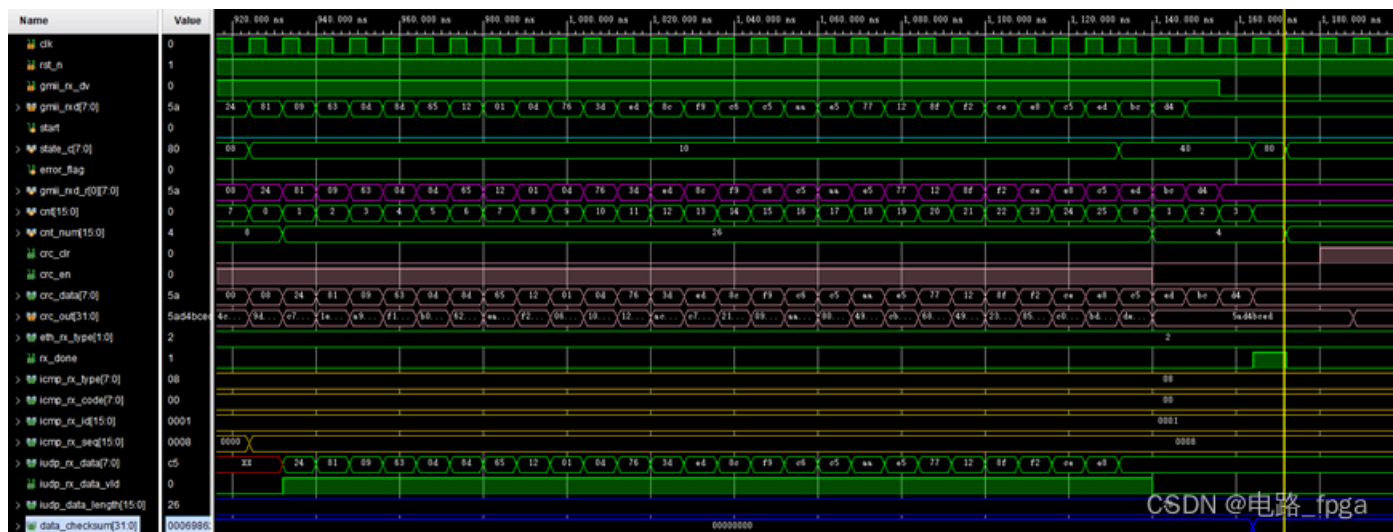


图11 ICMP报文数据段放大

接收UDP协议的仿真如下图所示，将接收的数据段输出到，蓝色信号就是输出的数据信号。

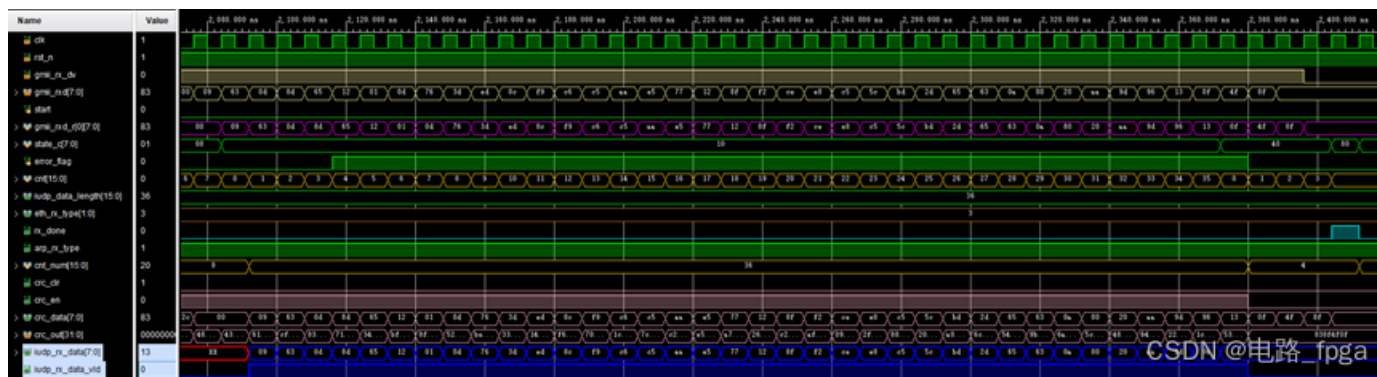


图12 ARP报文放大

5、以太网发送模块

发送模块依旧可以使用状态机嵌套计数器的形式实现，状态机对应的状态转换图如下所示。

状态机包括初始状态、发送前导码帧起始符状态、发送以太网帧头状态、发送IP帧头状态、发送ICMP或UDP帧头状态、发送ICMP数据或UDP数据状态、发送ARP数据状态、发送CRC校验状态、帧间隙等待状态。总共9个状态，由于ICMP帧头和UDP帧头长度基本一样，且数据段都需要从外部输入数据，所以ICMP和UDP的帧头、数据段共用一个状态。

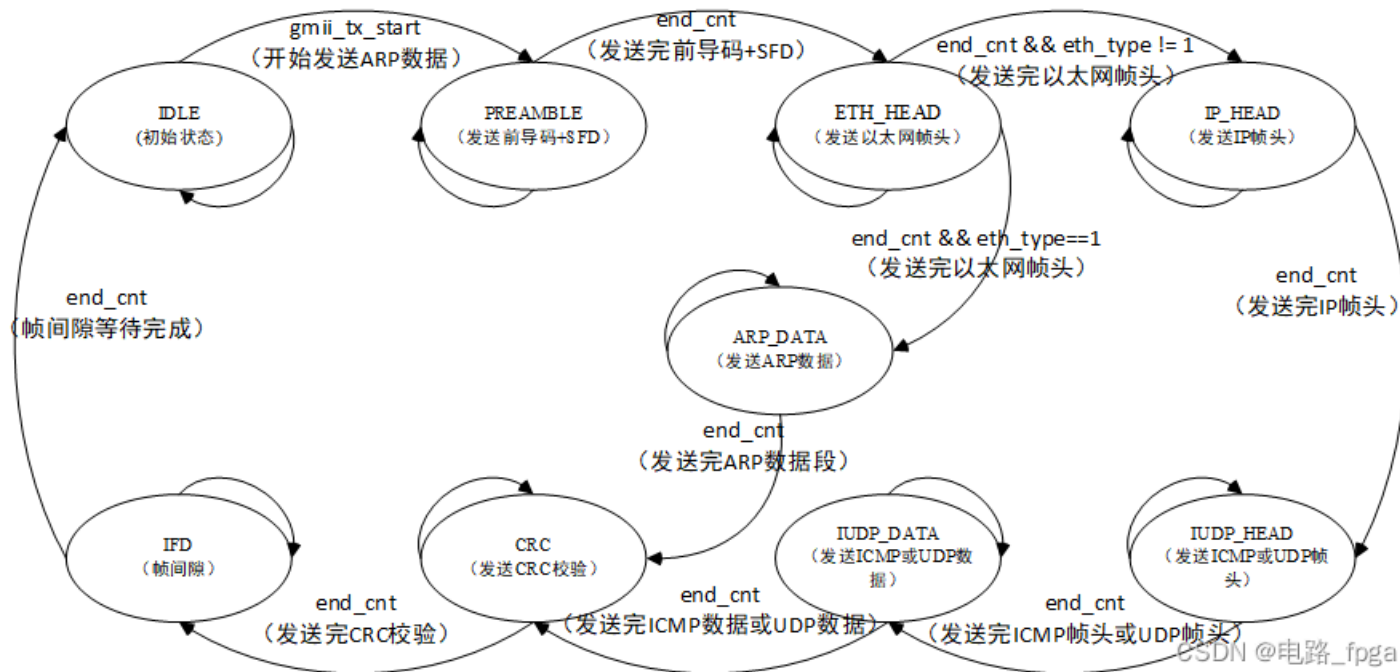


图13 以太网发送模块状态转换图

该模块设计还是比较简单的，在合并ARP发送模块、ICMP发送模块、UDP发送模块的基础上，增加了一个帧间隙的状态，一帧数据发完后，等待12字节的时间后回到空闲状态，那么上游模块可以马上调用该模块发送下一帧数据，上游模块不必做什么延时，方便使用。

注意该模块的数据请求信号需要提前数据输入信号三个时钟周期产生，这是因为请求信号和数据输入信号都是ICMP和UDP复用的，以太网控制模块需要根据发送协议类型，生成对应的请求信号，到ICMP的FIFO或者UDP的用户端口去请求数据输入，输入的数据还要整合成输入数据，所以需要消耗三个时钟周期。

参考代码的主要部分如下所示：

```

1      localparam  IDLE      = 9'b00000_0001      ;//初始状态，等待开始发送信号；
2      localparam  PREAMBLE  = 9'b00000_0010      ;//发送前导码+帧起始界定符；
3      localparam  ETH_HEAD  = 9'b00000_0100      ;//发送以太网帧头；
4      localparam  IP_HEAD   = 9'b00000_1000      ;//发送IP帧头；
5      localparam  IUDP_HEAD = 9'b00001_0000      ;//发送ICMP或UDP帧头；
6      localparam  IUDP_DATA = 9'b00010_0000      ;//发送ICMP或UDP协议数据；
7      localparam  ARP_DATA  = 9'b00100_0000      ;//发送ARP数据段；
8      localparam  CRC       = 9'b01000_0000      ;//发送CRC校验值；
9      localparam  IFG       = 9'b10000_0000      ;//帧间隙，也就是传输96bit的时间，对应12Byte数据。
10
11     localparam  MIN_DATA_NUM = 16'd18          ;//以太网数据最小46个字节，IP首部20个字节+UDP首部8个字节，所以数据至少46-20-8=18个
12
13     reg          gmii_tx_en_r          ;//
14     reg          [47 : 0] des_mac_r      ;//
15     reg          [31 : 0] des_ip_r       ;
16     reg          [8 : 0] state_n         ;
17     reg          [8 : 0] state_c         ;
18     reg          [15 : 0] cnt            ;//
19     reg          [15 : 0] cnt_num        ;//
20     reg          [15 : 0] iudp_tx_byte_num_r ;
21     reg          [31 : 0] ip_head        [4 : 0] ;
22     reg          [31 : 0] iudp_head      [1 : 0] ;//
23     reg          [7 : 0] arp_data        [17 : 0] ;
24     reg          [15 : 0] ip_total_num    ;
25     reg          [31 : 0] ip_head_check  ;//IP头部校验码；
26     reg          [31 : 0] icmp_check     ;//ICMP校验；
27
28     wire          add_cnt                ;
29     wire          end_cnt                ;
30
31

```

```

31 always@(posedge clk)begin
32     if(rst_n==1'b0)begin//初始值为0;
33         ip_head[0] <= 32'd0;
34         ip_head[1] <= {16'd0,16'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。
35         ip_head[2] <= 32'd0;
36         ip_head[3] <= 32'd0;
37         ip_head[4] <= 32'd0;
38         iudp_head[0] <= 32'd0;
39         iudp_head[1] <= 32'd0;
40         arp_data[0] <= 8'd0;
41         arp_data[1] <= 8'd0;
42         arp_data[2] <= 8'd0;
43         arp_data[3] <= 8'd0;
44         arp_data[4] <= 8'd0;
45         arp_data[5] <= 8'd0;
46         arp_data[6] <= 8'd0;
47         arp_data[7] <= 8'd0;
48         arp_data[8] <= 8'd0;
49         arp_data[9] <= 8'd0;
50         arp_data[10] <= 8'd0;
51         arp_data[11] <= 8'd0;
52         arp_data[12] <= 8'd0;
53         arp_data[13] <= 8'd0;
54         arp_data[14] <= 8'd0;
55         arp_data[15] <= 8'd0;
56         arp_data[16] <= 8'd0;
57         arp_data[17] <= 8'd0;
58         icmp_check <= 32'd0;
59         ip_head_check <= 32'd0;//IP头部校验和;
60         des_mac_r <= DES_MAC;
61         des_ip_r <= DES_IP;
62         iudp_tx_byte_num_r <= MIN_DATA_NUM;
63         ip_total_num <= MIN_DATA_NUM + 28;
64         eth_tx_type_r <= 0;
65     end
66     //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的数据进行暂存。
67 else if(state_c == IDLE && eth_tx_start)begin
68     if(eth_tx_type == 2'd1)begin//如果需要发送ARP报文;
69         arp_data[0] <= 8'h00;//ARP硬件类型;
70         arp_data[1] <= 8'h01;
71     end

```

```

72         arp_data[2] <= 8'h08;//发送协议类型;
73         arp_data[3] <= 8'h00;
74         arp_data[4] <= 8'h06;//硬件地址长度;
75         arp_data[5] <= 8'h04;//协议地址长度;
76         arp_data[6] <= 8'h00;//发送ARP操作类型;
77         arp_data[7] <= arp_tx_type ? 8'h02 : 8'h01;
78         arp_data[8] <= BOARD_MAC[47 : 40];//源MAC地址;
79         arp_data[9] <= BOARD_MAC[39 : 32];
80         arp_data[10] <= BOARD_MAC[31 : 24];
81         arp_data[11] <= BOARD_MAC[23 : 16];
82         arp_data[12] <= BOARD_MAC[15 : 8];
83         arp_data[13] <= BOARD_MAC[7 : 0];
84         arp_data[14] <= BOARD_IP[31 : 24];//源IP地址;
85         arp_data[15] <= BOARD_IP[23 : 16];
86         arp_data[16] <= BOARD_IP[15 : 8];
87         arp_data[17] <= BOARD_IP[7 : 0];
88     end
89     else if(eth_tx_type == 2'd2)begin//发送ICMP协议数据报;
90         iudp_head[0][31 : 16] <= {icmp_tx_type,icmp_tx_code};//存储ICMP的类型和代码。
91         iudp_head[1] <= {icmp_tx_id,icmp_tx_seq};//存储ICMP的标识符和ID;
92         ip_head[2] <= {8'h80,8'd1,16'd0};//分别表示生存时间,协议类型,1表示ICMP,6表示TCP,17表示UDP协议,低16位校验和先默认为0;
93         iudp_tx_byte_num_r <= iudp_tx_byte_num;//把数据段的长度暂存;
94         icmp_check <= icmp_data_checksum;//ICMP的校验和初始值为数据端的校验和。
95     end
96     else if(eth_tx_type == 2'd3)begin//发送UDP协议数据报;
97         iudp_head[0] <= {BOARD_PORT,DES_PORT};//16位源端口和目的端口地址。
98         iudp_head[1][31 : 16] <= (((iudp_tx_byte_num >= MIN_DATA_NUM) ? iudp_tx_byte_num : MIN_DATA_NUM) + 8);//计算UDP需要发送报文的长度
99         iudp_head[1][15 : 0] <= 16'd0;//UDP的校验和设置为0。
100        ip_head[2] <= {8'h80,8'd17,16'd0};//分别表示生存时间,协议类型,1表示ICMP,6表示TCP,17表示UDP协议,低16位校验和先默认为0;
101        iudp_tx_byte_num_r <= iudp_tx_byte_num;//把数据段的长度暂存;
102    end
103    eth_tx_type_r <= eth_tx_type;//把以太网数据报的类型暂存;
104    //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。
105    ip_total_num <= (((iudp_tx_byte_num >= MIN_DATA_NUM) ? iudp_tx_byte_num : MIN_DATA_NUM) + 28);
106    if((des_mac != 48'd0) && (des_ip != 32'd0))begin//当接收到目的MAC地址和目的IP地址时更新。
107        des_ip_r <= des_ip;
108        des_mac_r <= des_mac;
109    end
110    else begin
111        des_ip_r <= DES_IP;
112

```

```

113         des_mac_r <= DES_MAC;
114     end
115 end
116 //在发送以太网帧头时, 就开始计算IP帧头和ICMP的校验码, 并将计算结果存储, 便于后续直接发送。
117 else if(state_c == ETH_HEAD && add_cnt)begin
118     case (cnt)
119         16'd0 : begin//初始化需要发送的IP头部数据。
120             ip_head[0] <= {8'h45,8'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。
121             ip_head[3] <= BOARD_IP;//源IP地址。
122             ip_head[4] <= des_ip_r;//目的IP地址。
123         end
124         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。
125             ip_head_check <= ip_head[0][31 : 16] + ip_head[0][15 : 0];
126             if(eth_tx_type == 2'd2)
127                 icmp_check <= icmp_check + iudp_head[0][31 : 16];
128             end
129         16'd2 : begin
130             ip_head_check <= ip_head_check + ip_head[1][31 : 16];
131             if(eth_tx_type == 2'd2)
132                 icmp_check <= icmp_check + iudp_head[1][31 : 16];
133             end
134         16'd3 : begin
135             ip_head_check <= ip_head_check + ip_head[1][15 : 0];
136             if(eth_tx_type == 2'd2)
137                 icmp_check <= icmp_check + iudp_head[1][15 : 0];
138             end
139         16'd4 : begin
140             ip_head_check <= ip_head_check + ip_head[2][31 : 16];
141             if(eth_tx_type == 2'd2)
142                 icmp_check <= icmp_check[31 : 16] + icmp_check[15 : 0];//可能出现进位,累加一次。
143             end
144         16'd5 : begin
145             ip_head_check <= ip_head_check + ip_head[3][31 : 16];
146             if(eth_tx_type == 2'd2)
147                 icmp_check <= icmp_check[31 : 16] + icmp_check[15 : 0];//可能出现进位,累加一次。
148             end
149         16'd6 : begin
150             ip_head_check <= ip_head_check + ip_head[3][15 : 0];
151             if(eth_tx_type == 2'd2)
152                 iudp_head[0][15 : 0] <= ~icmp_check[15 : 0];//按位取反得到校验和。
153

```



```

154         end
155         16'd7 : begin
156             ip_head_check <= ip_head_check + ip_head[4][31 : 16];
157         end
158         16'd8 : begin
159             ip_head_check <= ip_head_check + ip_head[4][15 : 0];
160         end
161         16'd9,16'd10 : begin
162             ip_head_check <= ip_head_check[31 : 16] + ip_head_check[15 : 0];
163         end
164         16'd11 : begin
165             ip_head[2][15:0] <= ~ip_head_check[15 : 0];
166         end
167         default: begin
168             ip_head_check <= 32'd0;//校验和清零，用于下次计算。
169         end
170     endcase
171 end
172 else if(state_c == IP_HEAD && end_cnt)
173     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。
174 end
175
176 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
177 always@(posedge clk)begin
178     if(!rst_n)begin
179         state_c <= IDLE;
180     end
181     else begin
182         state_c <= state_n;
183     end
184 end
185
186 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
187 always@(*)begin
188     case(state_c)
189         IDLE:begin
190             if(eth_tx_start && (eth_tx_type != 2'd0))begin//在空闲状态接收到上游发出的使能信号;
191                 state_n = PREAMBLE;
192             end
193         else begin
194

```

```
195         state_n = state_c;
196     end
197 end
198 PREAMBLE:begin
199     if(end_cnt)begin//发送完前导码和SFD;
200         state_n = ETH_HEAD;
201     end
202     else begin
203         state_n = state_c;
204     end
205 end
206 ETH_HEAD:begin
207     if(end_cnt)begin//发送完以太网帧头数据;
208         if(~eth_tx_type_r[1])//如果发送ARP数据，则跳转到发送ARP数据状态;
209             state_n = ARP_DATA;
210         else//否则跳转到发送IP首部状态;
211             state_n = IP_HEAD;
212         end
213     else begin
214         state_n = state_c;
215     end
216 end
217 IP_HEAD:begin
218     if(end_cnt)begin//发送完IP帧头数据;
219         state_n = IUDP_HEAD;
220     end
221     else begin
222         state_n = state_c;
223     end
224 end
225 IUDP_HEAD:begin
226     if(end_cnt)begin//发送完UDP帧头数据;
227         state_n = IUDP_DATA;
228     end
229     else begin
230         state_n = state_c;
231     end
232 end
233 IUDP_DATA:begin
234     if(end_cnt)begin//发送完udp协议数据;
235
```

```

236         state_n = CRC;
237     end
238     else begin
239         state_n = state_c;
240     end
241 end
242 ARP_DATA:begin
243     if(end_cnt)begin//发送完ARP数据;
244         state_n = CRC;
245     end
246     else begin
247         state_n = state_c;
248     end
249 end
250 CRC:begin
251     if(end_cnt)begin//发送完CRC校验码;
252         state_n = IFG;
253     end
254     else begin
255         state_n = state_c;
256     end
257 end
258 IFG:begin
259     if(end_cnt)begin//延时帧间隙对应时间。
260         state_n = IDLE;
261     end
262     else begin
263         state_n = state_c;
264     end
265 end
266 default:begin
267     state_n = IDLE;
268 end
269 endcase
270 end
271
272 //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。
273 always@(posedge clk)begin
274     if(rst_n==1'b0)begin//
275         cnt <= 0;
276

```

```

277     end
278     else if(add_cnt)begin
279         if(end_cnt)
280             cnt <= 0;
281         else
282             cnt <= cnt + 1;
283     end
284 end
285
286 assign add_cnt = (state_c != IDLE); //状态机不在空闲状态时计数。
287 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。
288
289 //状态机在每个状态需要发送的数据个数。
290 always@(posedge clk)begin
291     if(rst_n==1'b0)begin //初始值为20;
292         cnt_num <= 16'd20;
293     end
294     else begin
295         case (state_c)
296             PREAMBLE : cnt_num <= 16'd8; //发送7个前导码和1个8'hd5。
297             ETH_HEAD : cnt_num <= 16'd14; //发送14字节的以太网帧头数据。
298             IP_HEAD : cnt_num <= 16'd20; //发送20个字节是IP帧头数据。
299             IUDP_HEAD : cnt_num <= 16'd8; //发送8字节的UDP帧头数据。
300             IUDP_DATA : if(iudp_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。
301                 cnt_num <= iudp_tx_byte_num_r;
302                 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。
303                     cnt_num <= MIN_DATA_NUM;
304             ARP_DATA : cnt_num <= 16'd46; //ARP数据阶段，发送46字节数据;
305             CRC : cnt_num <= 16'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。
306             IFG : cnt_num <= 16'd12; //帧间隙对应时间为12Byte数据传输时间。
307             default: cnt_num <= 16'd20;
308         endcase
309     end
310 end
311
312 //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。
313 always@(posedge clk)begin
314     if(rst_n==1'b0)begin //初始值为0;
315         crc_data <= 8'd0;
316     end
317

```

```

317 else if(add_cnt)begin
318     case (state_c)
319     PREAMBLE : if(end_cnt)
320         crc_data <= 8'h05;//发送1字节SFD编码;
321     else
322         crc_data <= 8'h55;//发送7字节前导码;
323     ETH_HEAD : if(cnt < 6)
324         crc_data <= des_mac_r[47 - 8*cnt -: 8];//发送目的MAC地址, 先发高字节;
325     else if(cnt < 12)
326         crc_data <= BOARD_MAC[47 - 8*(cnt-6) -: 8];//发送源MAC地址, 先发高字节;
327     else if(cnt == 12)
328         crc_data <= 8'h08;//发送源以太网协议类型, 先发高字节;
329     else
330         crc_data <= eth_tx_type_r[1] ? 8'h00 : 8'h06;//如果高位有效, 表示发送IP协议, 否则ARP协议。
331     ARP_DATA : if(cnt < 18)
332         crc_data <= arp_data[cnt];
333     else if(cnt < 24)
334         crc_data <= des_mac_r[47 - 8*(cnt - 18) -: 8];//发送目的MAC地址, 先发高字节;
335     else if(cnt < 28)
336         crc_data <= des_ip_r[31 - 8*(cnt - 24) -: 8];//发送目的IP地址, 先发高字节;
337     else//后面18位数据补0;
338         crc_data <= 8'd0;
339     IP_HEAD : if(cnt < 4)//发送IP帧头。
340         crc_data <= ip_head[0][31 - 8*cnt -: 8];
341     else if(cnt < 8)
342         crc_data <= ip_head[1][31 - 8*(cnt-4) -: 8];
343     else if(cnt < 12)
344         crc_data <= ip_head[2][31 - 8*(cnt-8) -: 8];
345     else if(cnt < 16)
346         crc_data <= ip_head[3][31 - 8*(cnt-12) -: 8];
347     else
348         crc_data <= ip_head[4][31 - 8*(cnt-16) -: 8];
349     IUDP_HEAD : if(cnt < 4)//发送UDP帧头数据。
350         crc_data <= iudp_head[0][31 - 8*cnt -: 8];
351     else
352         crc_data <= iudp_head[1][31 - 8*(cnt-4) -: 8];
353     IUDP_DATA : if(iudp_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。
354         crc_data <= iudp_tx_data;//如果满足最小要求, 将需要配发送的数据输出。
355     else if(cnt < iudp_tx_byte_num_r)//不满足最小要求时, 先将需要发送的数据发送完。
356         crc_data <= iudp_tx_data;//将需要发送的数据输出即可。
357
358

```

```

358         else//剩余数据补充0。
359             crc_data <= 8'd0;
360         default : ;
361     endcase
362 end
363 end
364
365 //生成数据请求输入信号，外部输入数据延后该信号三个时钟周期，所以需要提前产生三个时钟周期产生请求信号；
366 always@(posedge clk)begin
367     if(rst_n==1'b0)begin//初始值为0；
368         iudp_tx_data_req <= 1'b0;
369     end
370     //在数据段的前三个时钟周期拉高；
371     else if(state_c == IUDP_HEAD && add_cnt && (cnt == cnt_num - 4))begin
372         iudp_tx_data_req <= 1'b1;
373     end//在ICMP或者UDP数据段时，当发送完数据的前三个时钟拉低；
374     else if(iudp_tx_byte_num_r >= MIN_DATA_NUM)begin//发送的数据段长度大于等于18.
375         if(state_c == IUDP_DATA && add_cnt && (cnt == cnt_num - 4))begin
376             iudp_tx_data_req <= 1'b0;
377         end
378     end
379     else begin//发送的数据段长度小于4；
380         if(state_c == IUDP_HEAD && (iudp_tx_byte_num_r <= 3) && add_cnt && (cnt == cnt_num + iudp_tx_byte_num_r - 4))begin
381             iudp_tx_data_req <= 1'b0;
382         end//发送的数据段有效长度大于等于4，小于18时；
383         else if(state_c == IUDP_DATA && (iudp_tx_byte_num_r > 3) && add_cnt && (cnt == iudp_tx_byte_num_r - 4))begin
384             iudp_tx_data_req <= 1'b0;
385         end
386     end
387 end
388
389 //生成一个crc_data指示信号，用于生成gmii_txd信号。
390 always@(posedge clk)begin
391     if(rst_n==1'b0)begin//初始值为0；
392         gmii_tx_en_r <= 1'b0;
393     end
394     else if(state_c == CRC)begin
395         gmii_tx_en_r <= 1'b0;
396     end
397     else if(state_c == PREAMBLE)begin
398

```

```

399         gmii_tx_en_r <= 1'b1;
400     end
401 end
402
403 //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。
404 always@(posedge clk)begin
405     if(rst_n==1'b0)begin//初始值为0;
406         crc_en <= 1'b0;
407     end
408     else if(state_c == CRC)begin//当ARP和以太网帧头数据全部输出后拉低。
409         crc_en <= 1'b0;
410     end//当开始输出以太网帧头时拉高。
411     else if(state_c == ETH_HEAD && add_cnt)begin
412         crc_en <= 1'b1;
413     end
414 end
415
416 //生产CRC校验模块清零信号，状态机处于空闲时清零。
417 always@(posedge clk)begin
418     crc_clr <= (state_c == IDLE);
419 end
420
421 //生成gmii_txd信号，默认输出0。
422 always@(posedge clk)begin
423     if(rst_n==1'b0)begin//初始值为0;
424         gmii_txd <= 8'd0;
425     end//在输出CRC状态时，输出CRC校验码，先发送低位数据。
426     else if(state_c == CRC && add_cnt && cnt > 0)begin
427         gmii_txd <= crc_out[(8*cnt - 1) -: 8];
428     end//其余时间如果crc_data有效，则输出对应数据。
429     else if(gmii_tx_en_r)begin
430         gmii_txd <= crc_data;
431     end
432 end
433
434 //生成gmii_txd有效指示信号。
435 always@(posedge clk)begin
436     gmii_tx_en <= gmii_tx_en_r || (state_c == CRC);
437 end
438

```

//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。

//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(eth_tx_start || state_c != IDLE)
        rdy = 1'b0;
    else
        rdy = 1'b1;
end
```



该模块发送ARP报文仿真如下所示：

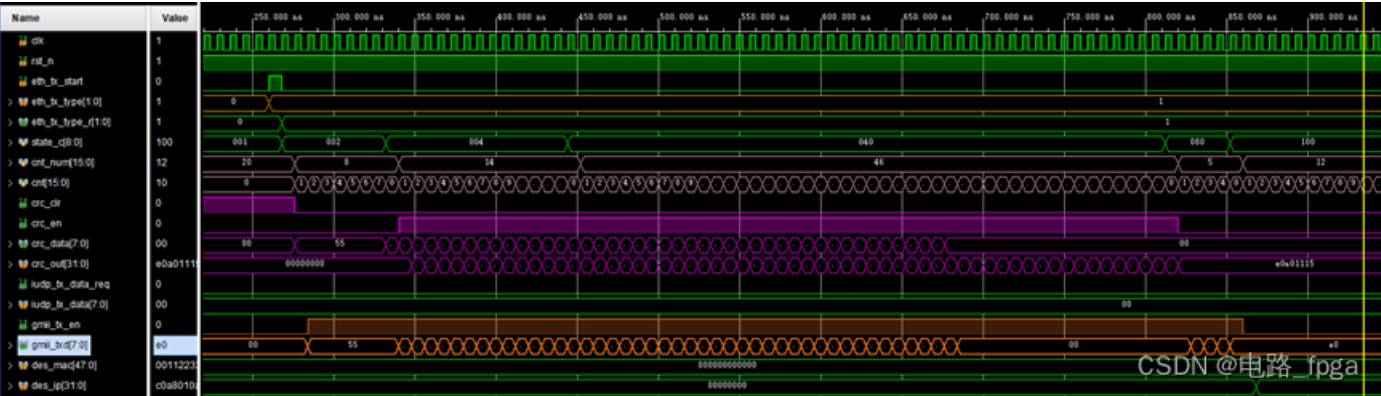


图14 ARP发送报文仿真

发送ICMP报文仿真如下所示：

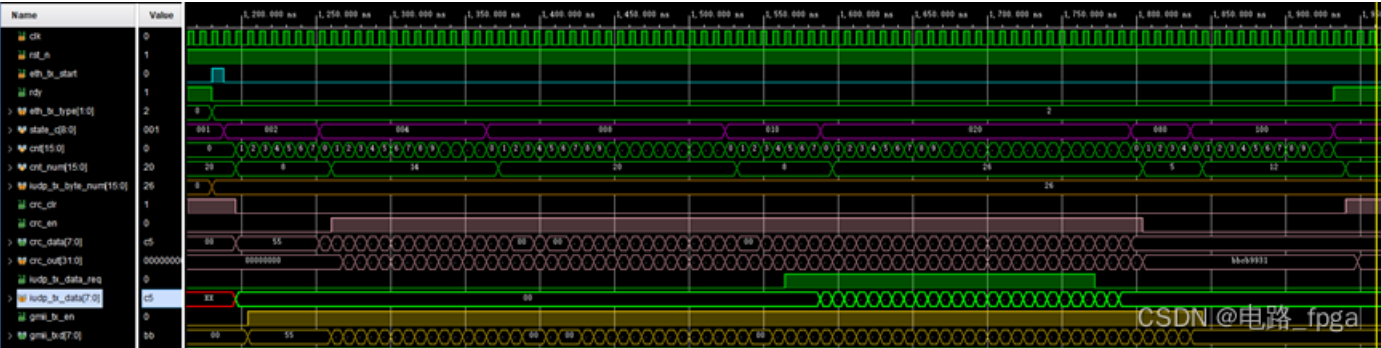


图15 ICMP发送报文仿真

发送UDP报文仿真如下所示:

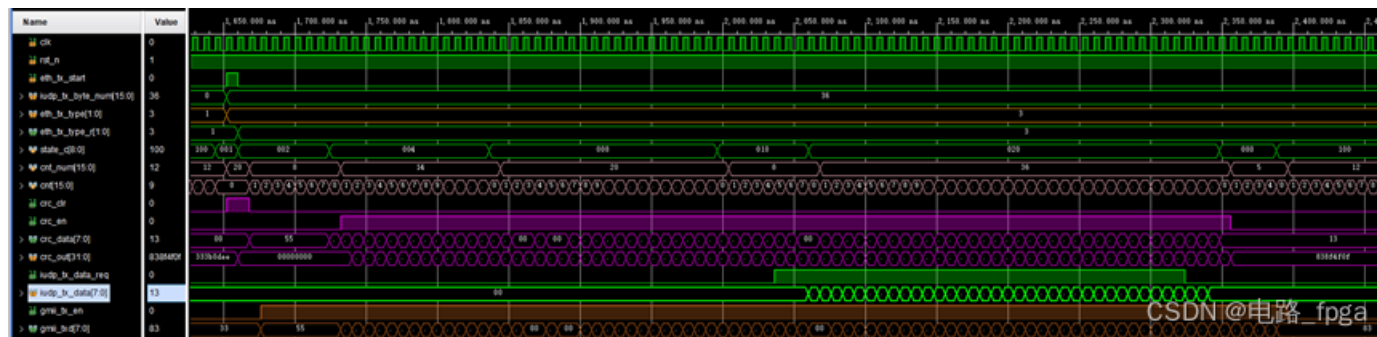


图16 UDP发送报文仿真

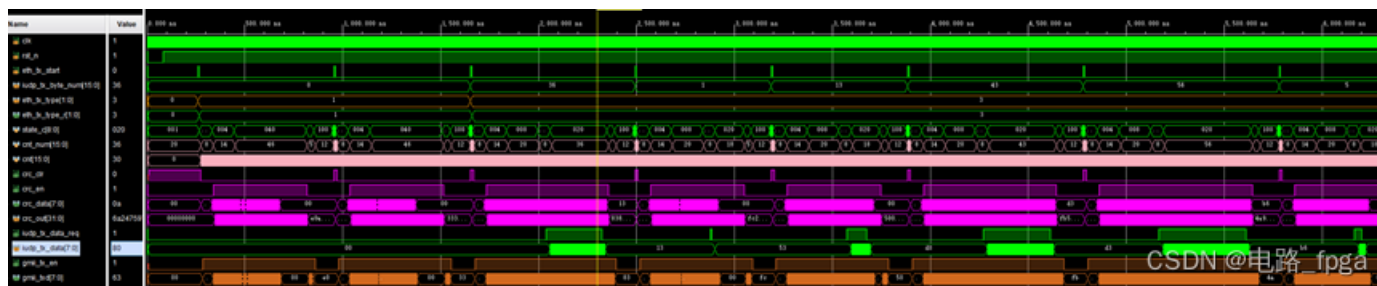


图17 整体仿真

6、以太网控制模块

该模块的难度在于相关信号比较多，会涉及以太网发送模块信号、以太网接模块信号、UDP用户接口信号、ICMP的FIFO控制信号。

当接收到ARP请求报文后，需要将ARP发送报文使能拉高，等待以太网发送模块空闲时，开始发送ARP应答报文。当接收到用户端口的ARP请求时，驱动以太网发送模块向目的IP地址发送ARP请求报文。

当接收到ICMP回显请求报文后，需要把ICMP数据段内容存入ICMP FIFO中，并且把ICMP发送报文使能信号拉高，等待以太网发送模块空闲时，开始发送ICMP回显应答报文。当发送模块的数据请求信号为高电平时，如果发送的报文是ICMP数据报文，则从ICMP FIFO中读取数据输入以太网发送模块。

当以太网接收模块接收到UDP报文后，把接收的UDP数据段输出到UDP用户端口。当用户端口的UDP开始发送信号有效时，把UDP发送使能信号拉高，等待以太网发送模块空闲时，驱动以太网发送模块发送UDP数据报文。当以太网发送模块发送UDP报文请求数据输入时，向用户端口产生数据输入使能，并且把UDP用户端口输入的数据输出到以太网发送模块作为UDP数据段的数据。

该模块的核心参考代码如下所示：

```
1 //高电平表示接收的数据报文是ICMP回显请求；
2 assign icmp_echo_request = (eth_rx_type == 2'd2) && (icmp_rx_type == 8) && (icmp_rx_code == 0);
3
4 //把UDP发送使能信号暂存，可能当前发送模块处于工作状态；
5 always@(posedge clk)begin
6     if(rst_n==1'b0)begin//初始值为0；
7         udp_tx_flag <= 1'b0;
8     end
9     else if(udp_tx_en)begin
10         udp_tx_flag <= 1'b1;
11     end
12     else if(eth_tx_start && (&eth_tx_type))begin
13         udp_tx_flag <= 1'b0;
14     end
15 end
16
17 //把arp发送使能信号暂存，可能当前发送模块处于工作状态；
18 always@(posedge clk)begin
19     if(rst_n==1'b0)begin//初始值为0；
20         arp_tx_flag <= 1'b0;
21         arp_req_r <= 1'b0;
22     end
23     //当接受到ARP请求数据包，或者需要发出ARP请求时拉高；
24     else if((rx_done && (eth_rx_type == 2'd1) && ~arp_rx_type) || arp_req)begin
25         arp_tx_flag <= 1'b1;
26         arp_req_r <= arp_req;
27     end//当ARP指令发送出去后拉低。
28     else if(eth_tx_start && (eth_tx_type == 2'd1))begin
29         arp_tx_flag <= 1'b0;
30         arp_req_r <= 1'b0;
31     end
32 end
33
34 //把icmp发送使能信号暂存，可能当前发送模块处于工作状态；
35 always@(posedge clk)begin
36     if(rst_n==1'b0)begin//初始值为0；
37         icmp_tx_flag <= 1'b0;
38     end
39 end
```

```

39 //当接受到ICMP回显请求时拉高;
40 else if(rx_done && icmp_echo_request)begin
41     icmp_tx_flag <= 1'b1;
42 end//当ICMP指令发送出去后拉低。
43 else if(eth_tx_start && (eth_tx_type == 2'd2))begin
44     icmp_tx_flag <= 1'b0;
45 end
46 end
47
48 //开始发送以太网帧;
49 always@(posedge clk)begin
50     if(rst_n==1'b0)begin//初始值为0;
51         eth_tx_start <= 1'b0;
52         eth_tx_type <= 2'd0;
53         arp_tx_type <= 1'b0;
54         icmp_tx_type <= 8'd0;
55         icmp_tx_code <= 8'd0;
56         icmp_tx_id <= 16'd0;
57         icmp_tx_seq <= 16'd0;
58         iudp_tx_byte_num <= 16'd0;
59     end
60     //接收到ARP的请求数据报时，把开始发送信号拉高;
61     else if(arp_tx_flag && tx_rdy)begin
62         eth_tx_start <= 1'b1;
63         eth_tx_type <= 2'd1;
64         arp_tx_type <= arp_req_r ? 1'b0 : 1'b1;//发送ARP应答报文;
65     end//当接收到ICMP回显请求时，把开始发送信号拉高;
66     else if(icmp_tx_flag && tx_rdy)begin
67         eth_tx_start <= 1'b1;
68         eth_tx_type <= 2'd2;
69         icmp_tx_type <= 8'd0;//发送ICMP回显应答数据报文。
70         icmp_tx_code <= 8'd0;
71         icmp_tx_id <= icmp_rx_id;//将回显请求的ID传回去。
72         icmp_tx_seq <= icmp_rx_seq;
73         iudp_tx_byte_num <= iudp_rx_byte_num;
74     end//当需要发送udp数据时，把开始发送信号拉高;
75     else if(udp_tx_flag && tx_rdy)begin
76         eth_tx_start <= 1'b1;
77         eth_tx_type <= 2'd3;
78         iudp_tx_byte_num <= udp_tx_data_num;
79
80

```

```

80         end//如果检测到模块处于空闲状态，则将开始信号拉低。
81     else begin
82         eth_tx_start <= 1'b0;
83     end
84 end
85
86 //将接收的ICMP数据存入FIFO中。
87 always@(posedge clk)begin
88     if(rst_n==1'b0)begin//初始值为0;
89         icmp_fifo_wr_en <= 1'b0;
90         icmp_fifo_wdata <= 8'd0;
91     end//如果接收的数据是ICMP数据段的数据，把ICMP的数据存储到FIFO中。
92     else if(iudp_rx_data_vld && icmp_echo_request)begin
93         icmp_fifo_wr_en <= 1'b1;
94         icmp_fifo_wdata <= iudp_rx_data;
95     end
96     else begin
97         icmp_fifo_wr_en <= 1'b0;
98     end
99 end
100
101 //通过数据请求信号产生从ICMP的FIFO中读取数据或者向用户接口发送UDP数据请求信号；
102 always@(posedge clk)begin
103     if(rst_n==1'b0)begin//初始值为0;
104         udp_tx_req <= 1'b0;
105         icmp_fifo_rd_en <= 1'b0;
106     end
107     else if(iudp_tx_data_req)begin
108         if(eth_tx_type_r == 2'd2)begin//如果发送的是ICMP数据报，则从FIFO中读取数据；
109             udp_tx_req <= 1'b0;
110             icmp_fifo_rd_en <= 1'b1;
111         end
112         else begin//则表示发送的UDP数据报，则从外部获取UDP数据。
113             udp_tx_req <= 1'b1;
114             icmp_fifo_rd_en <= 1'b0;
115         end
116     end
117     else begin
118         udp_tx_req <= 1'b0;
119         icmp_fifo_rd_en <= 1'b0;
120
121

```

```

121     end
122 end
123
124 //将ICMP FIFO或者外部UDP获取的数据发送给以太网发送模块;
125 always@(posedge clk)begin
126     if(rst_n==1'b0)begin//初始值为0;
127         iudp_tx_data <= 8'd0;
128     end
129     else if(eth_tx_type_r == 2'd2)begin
130         iudp_tx_data <= icmp_fifo_rdata;
131     end
132     else begin
133         iudp_tx_data <= udp_tx_data;
134     end
135 end
136
137 //将接收的UDP数据输出。
138 always@(posedge clk)begin
139     if(rst_n==1'b0)begin//初始值为0;
140         udp_rx_data_vld <= 1'b0;
141         udp_rx_data <= 8'd0;
142     end//如果接收到UDP数据段信号，将UDP的数据输出。
143     else if(iudp_rx_data_vld && eth_rx_type == 2'd3)begin
144         udp_rx_data_vld <= 1'b1;
145         udp_rx_data <= iudp_rx_data;
146     end
147     else begin
148         udp_rx_data_vld <= 1'b0;
149     end
150 end
151
152 //一帧UDP数据接收完成。
153 always@(posedge clk)begin
154     if(rst_n==1'b0)begin//初始值为0;
155         udp_rx_done <= 1'b0;
156         udp_rx_data_num <= 16'd0;
157     end
158     else if(&eth_rx_type)begin//如果接收的是UDP数据报;
159         udp_rx_done <= rx_done;//将输出完成信号输出;
160         udp_rx_data_num <= iudp_rx_byte_num;//把UDP数据长度输出;

```

end
end



该模块不贴仿真结果了，需要的打开工程自行查看。

7、上板测试

CRC校验模块、FIFO的设置在前文都已经详细讲解了，所以本文不在赘述。

最后把顶层模块的ILA注释取消，综合工程，查看工程的使用量，如下所示，使用了1195个查找表，而以太网模块使用了1141个查找表，1131个触发器资源。

Name	Slice LUTs (78600)	Slice Registers (157200)	F7 Muxes (39300)	Slice (19650)	LUT as Logic (78600)	Block RAM Tile (265)	Bonded IOB (250)	IDELAYCTRL (5)	IDELAYE2IDELAYE2_FINEDELAY (250)	ILOGIC (250)	OLOGIC (250)	BUFGCTRL (32)	BUFGIO (20)	MMCME2_ADV (5)
top	1195	1202	7	495	1195	1	15	1		5	5	5	3	1
u_eth_wiz_0 (eth_wiz_0)	0	0	0	0	0	0	0	0	0	0	0	2	0	1
u_eth (eth)	1141	1131	7	474	1141	0.5	0	0	0	0	0	0	0	0
u_key (key)	9	27	0	10	9	0	0	0	0	0	0	0	0	0
u_rmii_to_rmii (rmii_to_rmii)	7	0	0	7	7	0	0	1		5	5	0	0	0
u_udp_fifo (fifo_generator_0_HD2)	39	44	0	15	39	0.5	0	0	0	0	0	0	0	0

图18 本工程消耗资源占比

前文实现udp回环，ARP应，ICMP应答的工程资源消耗如下图所示，工程消耗1979个查找表，2073个触发器。

Name	Slice LUTs (78600)	Slice Registers (157200)	F7 Muxes (39300)	Slice (19650)	LUT as Logic (78600)	Block RAM Tile (265)	Bonded IOB (250)	IDELAYCTRL (5)	IDELAYE2IDELAYE2_FINEDELAY (250)	ILOGIC (250)	OLOGIC (250)	BUFGCTRL (32)	BUFGIO (20)	MMCME2_ADV (5)
top	1979	2073	12	833	1979	1	15	1		5	5	5	3	1
u_udp (udp)	369	527	6	200	369	0	0	0	0	0	0	0	0	0
u_udp_icmp_udp_driver	119	125	0	75	119	0	0	0	0	0	0	0	0	0
u_eth_wiz_0 (eth_wiz_0)	0	0	0	0	0	0	0	0	0	0	0	2	0	1
u_fifo_generator_0 (fifo)	39	44	0	15	39	0.5	0	0	0	0	0	0	0	0
u_icmp (icmp)	815	778	5	372	815	0.5	0	0	0	0	0	0	0	0
u_key (key)	10	27	0	13	10	0	0	0	0	0	0	0	0	0
u_rmii_to_rmii (rmii)	15	0	0	14	15	0	0	1		5	5	0	0	0
u_udp (udp)	618	572	1	285	618	0	0	0	0	0	0	0	0	0

图19 前一工程消耗资源占比

对比图17、18可知，本文实现相同功能后，本文工程能够节约785查找表，节约大概百分之四十的查找表资源。节约了871触发器资源，大概节约原工程的42%触发器资源。

将程序下载到开发板中，然后打开网络调试助手，wireshark软件，发送UDP数据，网络调试助手抓取结果如下所示。

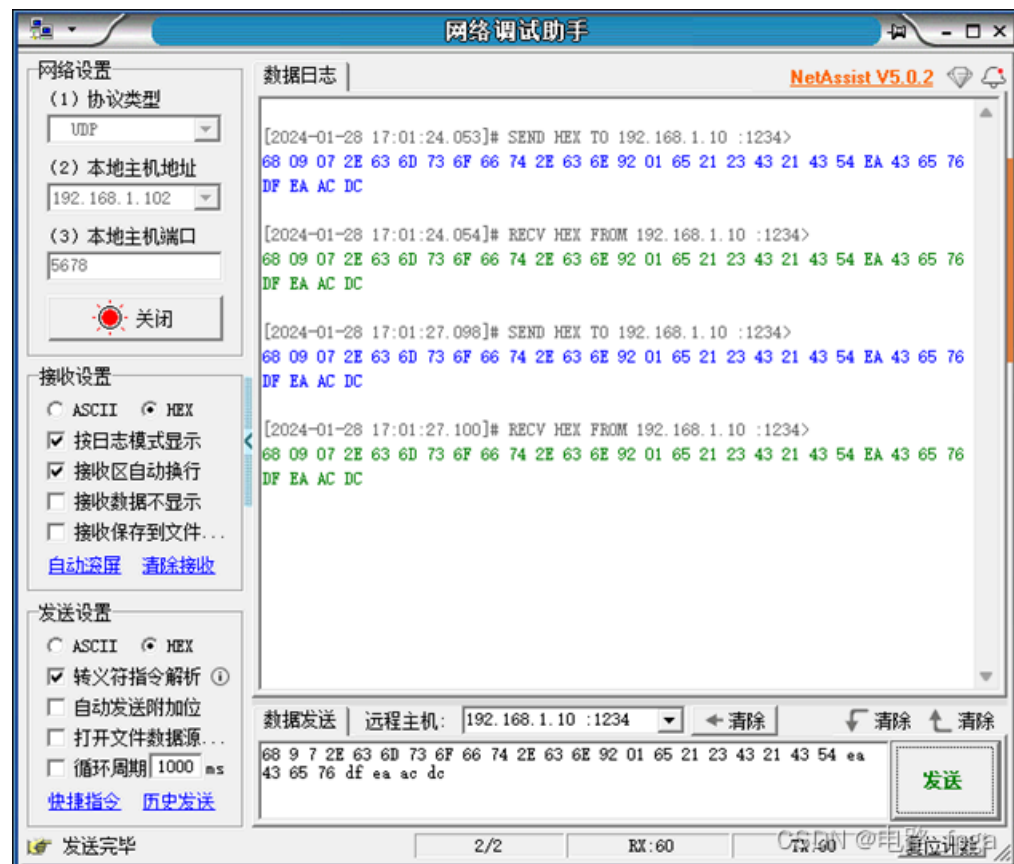


图20 网络调试助手抓取数据

在命令提示符中使用ping指令，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

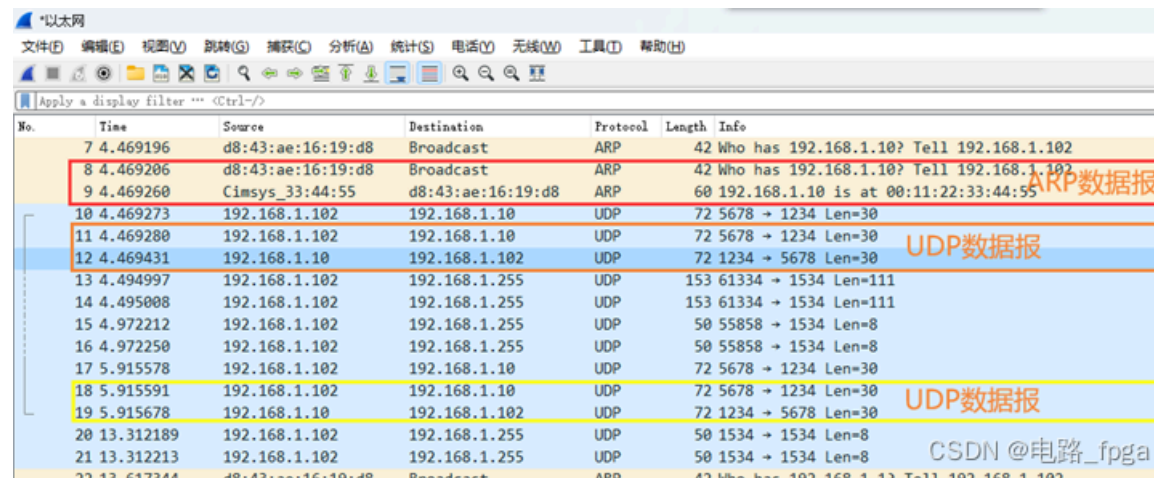
192.168.1.10 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Windows\System32>
```

CSDN @电路_fpga

图21 ping验证

通过使用wireshark抓取UDP数据报文，如下图所示，PC先向FPGA发出ARP请求报文获取FPGA的MAC地址，然后再发送UDP报文，FPGA接收到UDP报文后，将数据传回PC端。



The image shows a Wireshark network packet capture window. The packet list on the left shows several packets. Packets 8, 9, and 10 are highlighted in pink, indicating ICMP (ping) traffic. Packets 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, and 22 are highlighted in blue, indicating UDP traffic. The packet details pane on the right shows the structure of the selected packets. The packet bytes pane at the bottom shows the raw data of the selected packets. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
7	4.469196	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.102
8	4.469206	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.102
9	4.469260	Cimsys_33:44:55	d8:43:ae:16:19:d8	ARP	60	192.168.1.10 is at 00:11:22:33:44:55
10	4.469273	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
11	4.469280	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
12	4.469431	192.168.1.10	192.168.1.102	UDP	72	1234 → 5678 Len=30
13	4.494997	192.168.1.102	192.168.1.255	UDP	153	61334 → 1534 Len=111
14	4.495008	192.168.1.102	192.168.1.255	UDP	153	61334 → 1534 Len=111
15	4.972212	192.168.1.102	192.168.1.255	UDP	50	55858 → 1534 Len=8
16	4.972250	192.168.1.102	192.168.1.255	UDP	50	55858 → 1534 Len=8
17	5.915578	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
18	5.915591	192.168.1.102	192.168.1.10	UDP	72	5678 → 1234 Len=30
19	5.915678	192.168.1.10	192.168.1.102	UDP	72	1234 → 5678 Len=30
20	13.312189	192.168.1.102	192.168.1.255	UDP	50	1534 → 1534 Len=8
21	13.312213	192.168.1.102	192.168.1.255	UDP	50	1534 → 1534 Len=8
22	13.617344	d8:43:ae:16:19:d8	Broadcast	ARP	42	Who has 192.168.1.10? Tell 192.168.1.102

图22 wireshark抓取UDP数据报

下图粉红色报文是wireshark抓取的ICMP报文，FPGA接收PC端发出的回显请求报文后，向PC端发出回显应答报文。

20	13.312189	192.168.1.102	192.168.1.255	UDP	50 1534 → 1534 Len=8
21	13.312213	192.168.1.102	192.168.1.255	UDP	50 1534 → 1534 Len=8
22	13.617344	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
23	13.617355	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
24	14.258073	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=122/31232, ttl=128 (no response found!)
25	14.258082	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=122/31232, ttl=128 (reply in 26)
26	14.258129	192.168.1.10	192.168.1.102	ICMP	74 Echo (ping) reply id=0x0001, seq=122/31232, ttl=128 (request in 25)
27	14.403519	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
28	14.403542	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
29	15.267118	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=123/31488, ttl=128 (no response found!)
30	15.267142	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=123/31488, ttl=128 (reply in 31)
31	15.267295	192.168.1.10	192.168.1.102	ICMP	74 Echo (ping) reply id=0x0001, seq=123/31488, ttl=128 (request in 30)
32	15.405995	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
33	15.406024	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
34	16.019781	192.168.1.102	192.168.1.255	UDP	50 62076 → 1534 Len=8
35	16.019793	192.168.1.102	192.168.1.255	UDP	50 62076 → 1534 Len=8
36	16.272493	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=124/31744, ttl=128 (no response found!)
37	16.272501	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=124/31744, ttl=128 (reply in 38)
38	16.272550	192.168.1.10	192.168.1.102	ICMP	74 Echo (ping) reply id=0x0001, seq=124/31744, ttl=128 (request in 37)
39	16.411410	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
40	16.411428	d8:43:ae:16:19:d8	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
41	17.279479	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=125/32000, ttl=128 (no response found!)
42	17.279490	192.168.1.102	192.168.1.10	ICMP	74 Echo (ping) request id=0x0001, seq=125/32000, ttl=128 (reply in 43)
43	17.279549	192.168.1.10	192.168.1.102	ICMP	74 Echo (ping) reply id=0x0001, seq=125/32000, ttl=128 (request in 42)

图23 wireshark抓取ICMP数据报

关于报文详细内容，本文就不再赘述了，前文讲解ARP、ICMP、UDP时已经经过详细分析，本文分析原理一致，不再对比ILA抓取数据和wireshark工具抓取的数据了。

本文对前文学到的几种协议进行了总结、简化设计，使用一个模块发送和接收三种协议数据，这三种协议往往一起使用，后续可以直接使用该模块。

可以在公众号后台回复“**基于FPGA实用UDP设计**”（不包括引号）获取本文工程。

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★！