

基于FPGA实现ICMP协议（包含源工程文件）

基于FPGA的以太网相关文章导航，[点击查看](#)。

前文对IP协议和ICMP协议格式做了讲解，本文通过FPGA实现ICMP协议，PC端向开发板产生回显请求，FPGA接收到回显请求时，向PC端发出回显应答。为了不去手动绑定开发板的MAC地址和IP地址，还是需要ARP模块。

1、顶层设计

顶层模块直接使用vivado工程截图，如下图所示，顶层包括6个模块，按键消抖模块key、ARP收发模块、RGMII与GMII转换模块rgmii_to_gmii、锁相环模块在前文ARP协议实现时均已详细讲解，故本文不再赘述。

由于arp和icmp的发送模块需要使用同一组数据线，所以需要有一个arp_icmp_ctrl模块去确定输出哪一路信号。

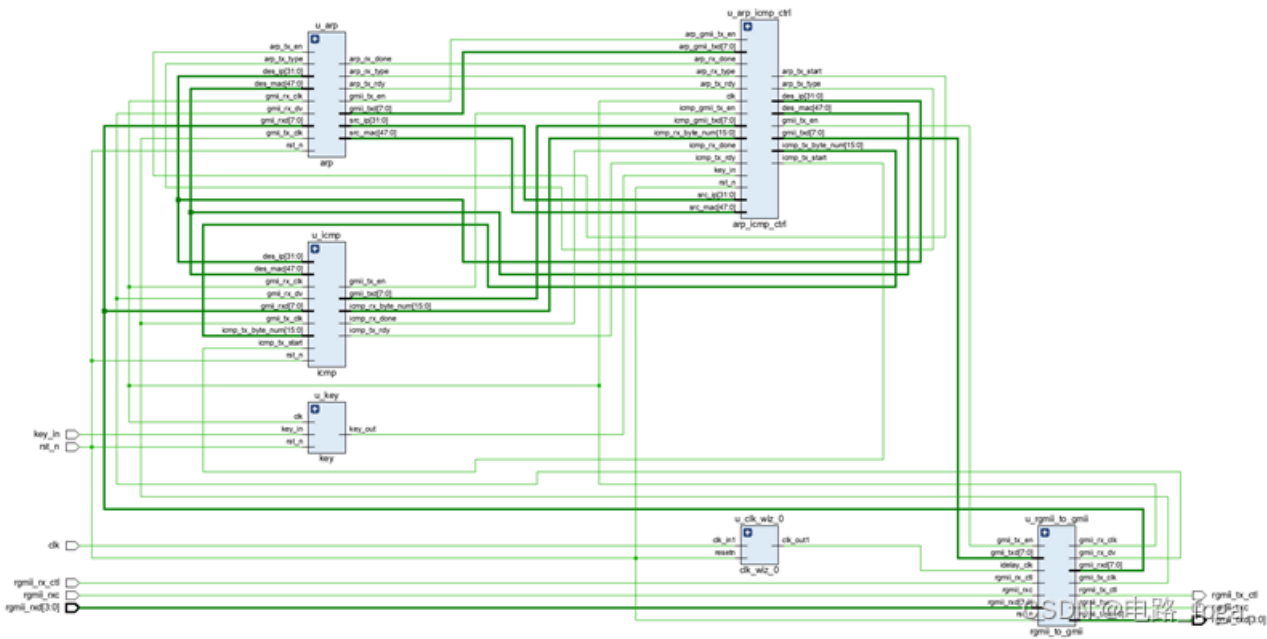


图1 顶层信号流向

顶层参考代码如下所示：

```

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_ctrl u_arp_icmp_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     .arp_rx_done  ( arp_rx_done ),//ARP接收完成信号；
26     .arp_rx_type  ( arp_rx_type ),//ARP接收类型 0:请求 1:应答；
27     .src_mac      ( src_mac    ),//ARP接收到目的MAC地址。
28     .src_ip       ( src_ip     ),//ARP接收到目的IP地址。
29     .arp_tx_rdy   ( arp_tx_rdy ),//ARP发送模块忙闲指示信号。
30     .arp_tx_start ( arp_tx_start ),//ARP发送使能信号；
31     .arp_tx_type  ( arp_tx_type ),//ARP发送类型 0:请求 1:应答；
32     .arp_gmii_tx_en ( arp_gmii_tx_en ),
33     .arp_gmii_txd ( arp_gmii_txd ),
34     .icmp_rx_done ( icmp_rx_done ),//ICMP接收完成信号；
35     .icmp_rx_byte_num ( icmp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
36     .icmp_tx_rdy  ( icmp_tx_rdy ),//ICMP发送模块忙闲指示信号。
37     .icmp_gmii_tx_en ( icmp_gmii_tx_en ),
38     .icmp_gmii_txd ( icmp_gmii_txd ),
39     .des_mac      ( des_mac    ),//发送的目标MAC地址。
40     .des_ip       ( des_ip     ),//发送的目标IP地址。
41

```

```

41     .icmp_tx_start      ( icmp_tx_start      ),//ICMP发送使能信号;
42     .icmp_tx_byte_num   ( icmp_tx_byte_num   ),//以太网发送的有效字节数 单位:byte。
43     .gmii_tx_en         ( gmii_tx_en         ),
44     .gmii_txd           ( gmii_txd           )
45 );
46
47 //例化ARP模块;
48 arp #(
49     .BOARD_MAC          ( BOARD_MAC          ),//开发板MAC地址 00-11-22-33-44-55;
50     .BOARD_IP           ( BOARD_IP           ),//开发板IP地址 192.168.1.10;
51     .DES_MAC            ( DES_MAC            ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
52     .DES_IP             ( DES_IP             ) //目的IP地址 192.168.1.102;
53 )
54 u_arp (
55     .rst_n              ( rst_n              ),//复位信号, 低电平有效。
56     .gmii_rx_clk        ( gmii_rx_clk        ),//GMII接收数据时钟。
57     .gmii_rx_dv         ( gmii_rx_dv         ),//GMII输入数据有效信号。
58     .gmii_rxd           ( gmii_rxd           ),//GMII输入数据。
59     .gmii_tx_clk        ( gmii_tx_clk        ),//GMII发送数据时钟。
60     .arp_tx_en          ( arp_tx_start       ),//ARP发送使能信号。
61     .arp_tx_type        ( arp_tx_type        ),//ARP发送类型 0:请求 1:应答。
62     .des_mac            ( des_mac            ),//发送的目标MAC地址。
63     .des_ip             ( des_ip             ),//发送的目标IP地址。
64     .gmii_tx_en         ( arp_gmii_tx_en     ),//GMII输出数据有效信号。
65     .gmii_txd           ( arp_gmii_txd       ),//GMII输出数据。
66     .arp_rx_done        ( arp_rx_done        ),//ARP接收完成信号。
67     .arp_rx_type        ( arp_rx_type        ),//ARP接收类型 0:请求 1:应答。
68     .src_mac            ( src_mac            ),//接收到目的MAC地址。
69     .src_ip             ( src_ip             ),//接收到目的IP地址。
70     .arp_tx_rdy         ( arp_tx_rdy        ) //ARP发送模块忙闲指示信号, 高电平表示该模块空闲。
71 );
72
73 //例化ICMP模块。
74 icmp #(
75     .BOARD_MAC          ( BOARD_MAC          ),//开发板MAC地址 00-11-22-33-44-55;
76     .BOARD_IP           ( BOARD_IP           ),//开发板IP地址 192.168.1.10;
77     .DES_MAC            ( DES_MAC            ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
78     .DES_IP             ( DES_IP             ),//目的IP地址 192.168.1.102;
79     .ETH_TYPE           ( 16'h0800          ) //以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
80 )
81
82

```

```

82 u_icmp (
83     .rst_n          ( rst_n          ),//复位信号，低电平有效。
84     .gmii_rx_clk    ( gmii_rx_clk    ),//GMII接收数据时钟。
85     .gmii_rx_dv     ( gmii_rx_dv     ),//GMII输入数据有效信号。
86     .gmii_rxd       ( gmii_rxd       ),//GMII输入数据。
87     .gmii_tx_clk    ( gmii_tx_clk    ),//GMII发送数据时钟。
88     .gmii_tx_en     ( icmp_gmii_tx_en ),//GMII输出数据有效信号。
89     .gmii_txd       ( icmp_gmii_txd  ),//GMII输出数据。
90     .icmp_tx_start  ( icmp_tx_start  ),//以太网开始发送信号。
91     .icmp_tx_byte_num ( icmp_tx_byte_num ),//以太网发送的有效字节数 单位:byte。
92     .des_mac        ( des_mac        ),//发送的目标MAC地址。
93     .des_ip         ( des_ip         ),//发送的目标IP地址。
94     .icmp_rx_done   ( icmp_rx_done   ),//ICMP接收完成信号。
95     .icmp_rx_byte_num ( icmp_rx_byte_num ),//以太网接收的有效字节数 单位:byte。
96     .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。
97 );
98
99 //例化gmii转RGMII模块。
100 rgmii_to_gmii u_rgmii_to_gmii (
101     .idelay_clk      ( idelay_clk      ),//IDELAY时钟;
102     .rst_n           ( rst_n           ),
103     .gmii_tx_en      ( gmii_tx_en      ),//GMII发送数据使能信号;
104     .gmii_txd        ( gmii_txd        ),//GMII发送数据;
105     .gmii_rx_clk     ( gmii_rx_clk     ),//GMII接收时钟;
106     .gmii_rx_dv      ( gmii_rx_dv      ),//GMII接收数据有效信号;
107     .gmii_rxd        ( gmii_rxd        ),//GMII接收数据;
108     .gmii_tx_clk     ( gmii_tx_clk     ),//GMII发送时钟;
109
110     .rgmii_rxc       ( rgmii_rxc       ),//RGMII接收时钟;
111     .rgmii_rx_ctl    ( rgmii_rx_ctl    ),//RGMII接收数据控制信号;
112     .rgmii_rxd       ( rgmii_rxd       ),//RGMII接收数据;
113     .rgmii_txc       ( rgmii_txc       ),//RGMII发送时钟;
114     .rgmii_tx_ctl    ( rgmii_tx_ctl    ),//RGMII发送数据控制信号;
115     .rgmii_txd       ( rgmii_txd       ) //RGMII发送数据;
116 );
117
118 /*ila_0 u_ila_0 (
119     .clk              ( gmii_rx_clk      ),//input wire clk
120     .probe0           ( gmii_rx_dv      ),//input wire [0:0] probe0
121     .probe1           ( gmii_rxd        ),//input wire [7:0] probe1
122

```

```

123 .probe2    ( gmii_tx_en                ),//input wire [0:0] probe2
124 .probe3    ( gmii_txd                 ),//input wire [7:0] probe3
125 .probe4    ( u_icmp.u_icmp_rx.state_n ),//input wire [6:0] probe4
126 .probe5    ( u_icmp.u_icmp_rx.state_c ),//input wire [6:0] probe5
127 .probe6    ( icmp_gmii_tx_en         ),//input wire [0:0] probe6
128 .probe7    ( icmp_gmii_txd           ),//input wire [7:0] probe7
129 .probe8    ( icmp_tx_rdy              ),//input wire [0:0] probe8
130 .probe9    ( icmp_rx_done             ),//input wire [0:0] probe9
131 .probe10   ( u_icmp.u_icmp_rx.error_flag ),//input wire [0:0] probe10
132 .probe11   ( u_icmp.u_icmp_rx.fifo_wr ),//input wire [0:0] probe11
133 .probe12   ( u_icmp.u_icmp_rx.cnt     ),//input wire [7:0] probe12
134 .probe13   ( u_icmp.u_icmp_rx.cnt_num ),//input wire [7:0] probe13
135 .probe14   ( u_icmp.u_icmp_rx.gmii_rxd_r[0]),//input wire [7:0] probe14
      .probe15 ( u_icmp.u_icmp_rx.fifo_wdata ) //input wire [7:0] probe15
    );*/

```



icmp模块实现对ICMP协议的接收和发送，下图是该模块的内部模块分布图，包括ICMP接收模块icmp_rx、ICMP发送模块icmp_tx，两个CRC校验模块，分别对接收和发送的数据进行CRC校验。因为回显应答必须把回显请求的数据原封不动的发送出去，因此使用一个FIFO对回显请求的数据进行暂存。

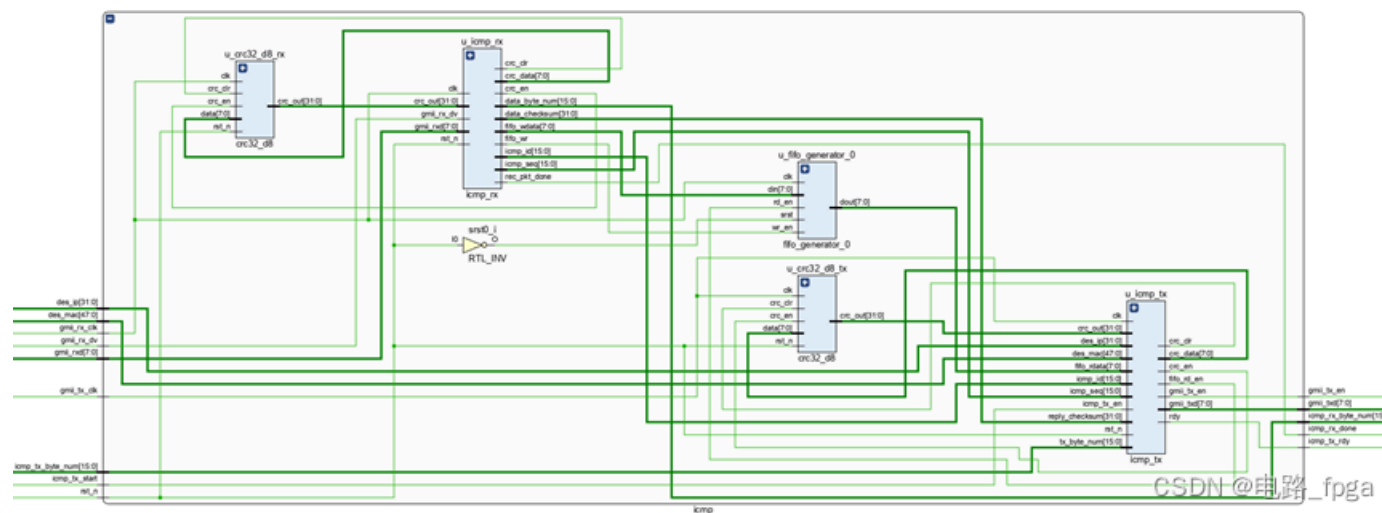


图2 ICMP模块信号流向

ICMP顶层参考代码如下所示：

```
1 //例化ICMP接收模块；
2 icmp_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_icmp_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    .rec_pkt_done      ( icmp_rx_done     ),//ICMP接收完成信号，高电平有效；
13    .fifo_wr           ( fifo_wr_en       ),//fifo写使能。
14    .fifo_wdata        ( fifo_wdata       ),//fifo写数据，将接收到的ICMP数据写入FIFO中。
15    .data_byte_num     ( icmp_rx_byte_num ),//以太网接收的有效数据字节数 单位:byte
16    .icmp_id           ( icmp_id          ),//ICMP标识符；
17    .icmp_seq          ( icmp_seq         ),//ICMP序列号；
18    .data_checksum     ( data_checksum    ),//ICMP数据段的校验和；
19    .crc_data          ( rx_crc_data      ),//需要CRC模块校验的数据；
20    .crc_en            ( rx_crc_en        ),//CRC开始校验使能；
21    .crc_clr           ( rx_crc_clr       ) //CRC数据复位信号；
22 );
23
24 //例化接收数据时需要的CRC校验模块；
25 crc32_d8 u_crc32_d8_rx (
26     .clk              ( gmii_rx_clk      ),//时钟信号；
27     .rst_n            ( rst_n            ),//复位信号，低电平有效；
28     .data             ( rx_crc_data      ),//需要CRC模块校验的数据；
29     .crc_en           ( rx_crc_en        ),//CRC开始校验使能；
30     .crc_clr          ( rx_crc_clr       ),//CRC数据复位信号；
31     .crc_out          ( rx_crc_out       ) //CRC校验模块输出的数据；
32 );
33
34 //例化ICMP发送模块；
35 icmp_tx #(
36     .BOARD_MAC      ( BOARD_MAC      ),//开发板MAC地址 00-11-22-33-44-55;
37     .BOARD_IP        ( BOARD_IP        ),//开发板IP地址 192.168.1.10;
38     .DES_MAC         ( DES_MAC         ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
39 )
```

```

39     .DES_IP          ( DES_IP          ),//目的IP地址 192.168.1.102;
40     .ETH_TYPE        ( ETH_TYPE        ) //以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
41 )
42
43 u_icmp_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     .reply_checksum   ( data_checksum    ),//ICMP数据段的校验和;
47     .icmp_id          ( icmp_id          ),//ICMP标识符;
48     .icmp_seq         ( icmp_seq         ),//ICMP序列号;
49     .icmp_tx_en       ( icmp_tx_start    ),//ICMP发送使能信号;
50     .tx_byte_num      ( icmp_tx_byte_num ),//ICMP数据段需要发送的数据。
51     .des_mac          ( des_mac          ),//发送的目标MAC地址;
52     .des_ip           ( des_ip           ),//发送的目标IP地址;
53     .crc_out          ( tx_crc_out       ),//CRC校验数据;
54     .crc_en           ( tx_crc_en        ),//CRC开始校验使能;
55     .crc_clr          ( tx_crc_clr       ),//CRC数据复位信号;
56     .crc_data         ( tx_crc_data      ),//输出给CRC校验模块进行计算的数据;
57     .fifo_rd_en       ( fifo_rd_en       ),//FIFO读使能信号。
58     .fifo_rdata       ( fifo_rdata      ),//从FIFO读出, 以太网需要发送的数据。
59     .gmii_tx_en       ( gmii_tx_en      ),//GMII输出数据有效信号;
60     .gmii_txd         ( gmii_txd        ),//GMII输出数据;
61     .rdy              ( icmp_tx_rdy     ) //模块忙闲指示信号, 高电平表示该模块处于空闲状态;
62 );
63
64 //例化发送数据时需要的CRC校验模块:
65 crc32_d8 u_crc32_d8_tx (
66     .clk              ( gmii_tx_clk      ),//时钟信号;
67     .rst_n            ( rst_n            ),//复位信号, 低电平有效;
68     .data             ( tx_crc_data      ),//需要CRC模块校验的数据;
69     .crc_en           ( tx_crc_en        ),//CRC开始校验使能;
70     .crc_clr          ( tx_crc_clr       ),//CRC数据复位信号;
71     .crc_out          ( tx_crc_out       ) //CRC校验模块输出的数据;
72 );
73
74 //例化FIFO:
75 fifo_generator_0 u_fifo_generator_0 (
76     .clk              ( gmii_rx_clk      ),//input wire clk
77     .srst             ( ~rst_n           ),//input wire srst
78     .din              ( fifo_wdata       ),//input wire [7 : 0] din
79     .wr_en            ( fifo_wr_en       ),//input wire wr_en
80

```

```

80         .rd_en  ( fifo_rd_en    ),//input wire rd_en
81         .dout   ( fifo_rdata    ),//output wire [7 : 0] dout
82         .full   (                ),//output wire full
83         .empty  (                ) //output wire empty
    );

```



ICMP顶层TestBench参考代码如下所示:

```

1  `timescale 1 ns/1 ns
2  module test();
3      parameter    CYCLE      =    10                ;//系统时钟周期, 单位ns, 默认10ns;
4      parameter    RST_TIME   =    10                ;//系统复位持续时间, 默认10个系统时钟周期;
5      parameter    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      parameter    BOARD_MAC  =  48'h00_11_22_33_44_55 ;
7      parameter    BOARD_IP   = {8'd192,8'd168,8'd1,8'd10} ;
8      localparam   DES_MAC    =  48'h23_45_67_89_0a_bc ;
9      localparam   DES_IP     = {8'd192,8'd168,8'd1,8'd23} ;
10     localparam   ETH_TYPE    =  16'h0800            ;//以太网帧类型 IP
11
12     reg           clk         ;//系统时钟, 默认100MHz;
13     reg           rst_n       ;//系统复位, 默认低电平有效;
14     reg           [7 : 0]     gmii_rxd              ;
15     reg           gmii_rx_dv              ;
16     wire          icmp_tx_start            ;
17     wire          [15 : 0]     icmp_tx_byte_num     ;
18     wire          [47 : 0]     des_mac              ;
19     wire          [31 : 0]     des_ip              ;
20     wire          gmii_tx_en              ;
21     wire          [7 : 0]     gmii_txd            ;
22     wire          icmp_rx_done            ;
23     wire          [15 : 0]     icmp_rx_byte_num     ;
24     wire          icmp_tx_rdy            ;
25
26     reg           [7 : 0]     rx_data [255 : 0]      ;//申请256个数据的存储器
27
28     assign icmp_tx_start = icmp_rx_done;
--

```



```

29 assign icmp_tx_byte_num = icmp_rx_byte_num;
30 assign des_mac = 0;
31 assign des_ip = 0;
32
33 icmp #(
34     .BOARD_MAC   ( BOARD_MAC ),
35     .BOARD_IP    ( BOARD_IP  ),
36     .DES_MAC     ( DES_MAC   ),
37     .DES_IP      ( DES_IP    ),
38     .ETH_TYPE    ( ETH_TYPE  )
39 )
40 u_icmp (
41     .rst_n        ( rst_n      ),
42     .gmii_rx_clk  ( clk        ),
43     .gmii_rx_dv   ( gmii_rx_dv ),
44     .gmii_rxd     ( gmii_rxd   ),
45     .gmii_tx_clk  ( clk        ),
46     .icmp_tx_start ( icmp_tx_start ),
47     .icmp_tx_byte_num ( icmp_tx_byte_num ),
48     .des_mac      ( des_mac    ),
49     .des_ip       ( des_ip     ),
50     .gmii_tx_en   ( gmii_tx_en ),
51     .gmii_txd     ( gmii_txd   ),
52     .icmp_rx_done  ( icmp_rx_done ),
53     .icmp_rx_byte_num ( icmp_rx_byte_num ),
54     .icmp_tx_rdy   ( icmp_tx_rdy )
55 );
56
57 reg          crc_clr          ;
58 reg          gmii_crc_vld     ;
59 reg [7 : 0]  gmii_rxd_r       ;
60 reg          gmii_rx_dv_r     ;
61 reg          crc_data_vld     ;
62 reg [9 : 0]  i                ;
63 reg [15 : 0] num              ;
64 wire [31 : 0] crc_out         ;
65
66 //生成周期为CYCLE数值的系统时钟;
67 initial begin
68     clk = 0;
69

```

```

70     forever #(CYCLE/2) clk = ~clk;
71 end
72
73 //生成复位信号;
74 initial begin
75     #1;gmii_rxd = 0; gmii_rx_dv = 0;gmii_crc_vld = 1'b0;num=0;
76     gmii_rxd_r=0;gmii_rx_dv_r=0;crc_clr=0;
77     for(i = 0 ; i < 256 ; i = i + 1)begin
78         #1;
79         rx_data[i] = {$random} % 256;//初始化存储体;
80     end
81     rst_n = 1;
82     #2;
83     rst_n = 0;//开始时复位10个时钟;
84     #(RST_TIME*CYCLE);
85     rst_n = 1;
86     #(20*CYCLE);
87     repeat(4)begin//发送2帧数据;
88         gmii_tx_test(18);
89         gmii_crc_vld = 1'b1;
90         gmii_rxd_r = crc_out[7 : 0];
91         #(CYCLE);
92         gmii_rxd_r = crc_out[15 : 8];
93         #(CYCLE);
94         gmii_rxd_r = crc_out[23 : 16];
95         #(CYCLE);
96         gmii_rxd_r = crc_out[31 : 24];
97         #(CYCLE);
98         gmii_crc_vld = 1'b0;
99         crc_clr = 1'b1;
100        #(CYCLE);
101        crc_clr = 1'b0;
102        @(posedge icmp_rx_done);
103        #(50*CYCLE);
104    end
105    #(20*CYCLE);
106    $stop;//停止仿真;
107 end
108
109 task gmii_tx_test(
110

```

```

111     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];
112 );
113     reg [31 : 0] ip_check;
114     reg [15 : 0] total_num;
115     reg [31 : 0] icmp_check;
116     begin
117         total_num = data_num + 28;
118         icmp_check = 16'h1 + 16'h8;//ICMP首部相加;
119         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;
120
121         if(~data_num[0])begin//ICMP数据段个数为偶数;
122             for(i=0 ; 2*i < data_num ; i= i+1)begin
123                 #1;//计算ICMP数据段的校验和。
124                 icmp_check = icmp_check + {rx_data[i][7:0],rx_data[i+1][7:0]};
125             end
126         end
127     else begin//ICMP数据段个数为奇数;
128         for(i=0 ; 2*i < data_num+1 ; i= i+1)begin
129             #1;//计算ICMP数据段的校验和。
130             if(2*i + 1 == data_num)
131                 icmp_check = icmp_check + {rx_data[i][7:0]};
132             else
133                 icmp_check = icmp_check + {rx_data[i][7:0],rx_data[i+1][7:0]};
134             end
135         end
136         crc_data_vld = 1'b0;
137         #(CYCLE);
138         repeat(7)begin//发送前导码7个8'H55;
139             gmii_rxd_r = 8'h55;
140             gmii_rx_dv_r = 1'b1;
141             #(CYCLE);
142         end
143         gmii_rxd_r = 8'hd5;//发送SFD，一个字节的8'hd5;
144         #(CYCLE);
145         crc_data_vld = 1'b1;
146         //发送以太网帧头数据;
147         for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的MAC地址;
148             gmii_rxd_r = BOARD_MAC[47-8*i -: 8];
149             #(CYCLE);
150         end
151

```

```

152     for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的源MAC地址;
153         gmii_rxd_r = DES_MAC[47-8*i -: 8];
154         #(CYCLE);
155     end
156     for(i=0 ; i<2 ; i=i+1)begin//发送2个字节的以太网类型;
157         gmii_rxd_r = ETH_TYPE[15-8*i -: 8];
158         #(CYCLE);
159     end
160     //发送IP帧头数据;
161     gmii_rxd_r = 8'H45;
162     #(CYCLE);
163     gmii_rxd_r = 8'd00;
164     ip_check = ip_check[15 : 0] + ip_check[31:16];
165     icmp_check = icmp_check[15 : 0] + icmp_check[31:16];
166     #(CYCLE);
167     gmii_rxd_r = total_num[15:8];
168     ip_check = ip_check[15 : 0] + ip_check[31:16];
169     icmp_check = icmp_check[15 : 0] + icmp_check[31:16];
170     #(CYCLE);
171     gmii_rxd_r = total_num[7:0];
172     ip_check = ~ip_check[15 : 0];
173     icmp_check = ~icmp_check[15 : 0];
174     #(CYCLE);
175     gmii_rxd_r = num[15:8];
176     #(CYCLE);
177     gmii_rxd_r = num[7:0];
178     #(CYCLE);
179     gmii_rxd_r = 8'h40;
180     #(CYCLE);
181     gmii_rxd_r = 8'h00;
182     #(CYCLE);
183     gmii_rxd_r = 8'h80;
184     #(CYCLE);
185     gmii_rxd_r = 8'h01;
186     #(CYCLE);
187     gmii_rxd_r = ip_check[15:8];
188     #(CYCLE);
189     gmii_rxd_r = ip_check[7:0];
190     #(CYCLE);
191     for(i=0 ; i<4 ; i=i+1)begin//发送6个字节的源IP地址;
192

```

```

193         gmii_rxd_r = DES_IP[31-8*i -: 8];
194         #(CYCLE);
195     end
196     for(i=0 ; i<4 ; i=i+1)begin//发送4个字节的IP地址;
197         gmii_rxd_r = BOARD_IP[31-8*i -: 8];
198         #(CYCLE);
199     end
200     //发送ICMP帧头及数据包:
201     gmii_rxd_r = 8'h08;//发送回显请求。
202     #(CYCLE);
203     gmii_rxd_r = 8'h00;
204     #(CYCLE);
205     gmii_rxd_r = icmp_check[31:16];
206     #(CYCLE);
207     gmii_rxd_r = icmp_check[15:0];
208     #(CYCLE);
209     gmii_rxd_r = 8'h00;
210     #(CYCLE);
211     gmii_rxd_r = 8'h01;
212     #(CYCLE);
213     gmii_rxd_r = 8'h00;
214     #(CYCLE);
215     gmii_rxd_r = 8'h08;
216     #(CYCLE);
217     for(i=0 ; i<data_num ; i=i+1)begin
218         gmii_rxd_r = rx_data[i];
219         #(CYCLE);
220     end
221     crc_data_vld = 1'b0;
222     gmii_rx_dv_r = 1'b0;
223     num = num + 1;
224 end
225 endtask
226
227 crc32_d8  u_crc32_d8_1 (
228     .clk      ( clk          ),
229     .rst_n    ( rst_n        ),
230     .data     ( gmii_rxd_r    ),
231     .crc_en   ( crc_data_vld  ),
232     .crc_clr  ( crc_clr       ),
233

```

```

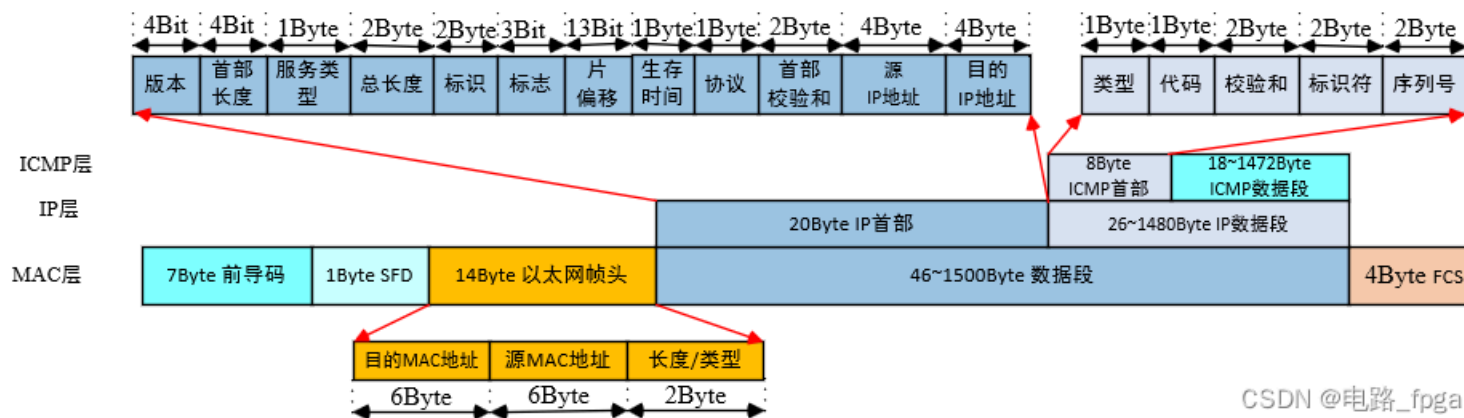
232     .crc_out      ( crc_out      )
233 );
234
235
236 always@(posedge clk)begin
237     if(rst_n==1'b0)begin//初始值为0;
238         gmii_rxd <= 8'd0;
239         gmii_rx_dv <= 1'b0;
240     end
241     else if(gmii_rx_dv_r || gmii_crc_vld)begin
242         gmii_rxd <= gmii_rxd_r;
243         gmii_rx_dv <= 1'b1;
244     end
245     else begin
246         gmii_rx_dv <= 1'b0;
247     end
248 end
249
250 endmodule

```



2、ICMP接收模块

前文对ICMP的数据报做了详细讲解，ICMP数据报文的构成如下所示，包括前导码和帧起始符、以太网帧头、IP首部、ICMP首部、ICMP数据、CRC校验等几个模块。



CSDN @电路_fpga

图3 以太网的ICMP数据报格式

本文检测接收的数据是不是ICMP回显请求，需要将回显请求的标识符、序列号、ICMP数据段保存下来，便于回显应答时使用。同时在接收ICMP数据时，应该把数据端的校验和计算出来，回显应答时就不再需要时间去计算数据段的校验和了。

此模块没有做IP首部校验和以及ICMP校验和，只做了CRC校验和，因为FPGA根据接收到的数据特征，其实能够大致判断是否正确，加上CRC校验无误就基本上不会出现错误了。

本模块以状态机为主体，嵌套一个计数器cnt实现，下图是状态机的状态转换图。

本模块会使用移位寄存器把输入数据gmii_rxd暂存7个时钟周期，便于前导码和帧起始符的检测，该移位寄存器的数据也可以在后续中使用，所以使用移位寄存器会比较方便。

空闲状态(IDLE): 这个状态就一直检测前导码和帧起始符，检测到后把start信号拉高，表示开始接收数据，状态机跳转到接收以太网帧头的状态。状态机的现态与移位寄存器gmii_rxd[0]的数据对齐，后续数据大多都来自该移位寄存器最低位置存储的数据。

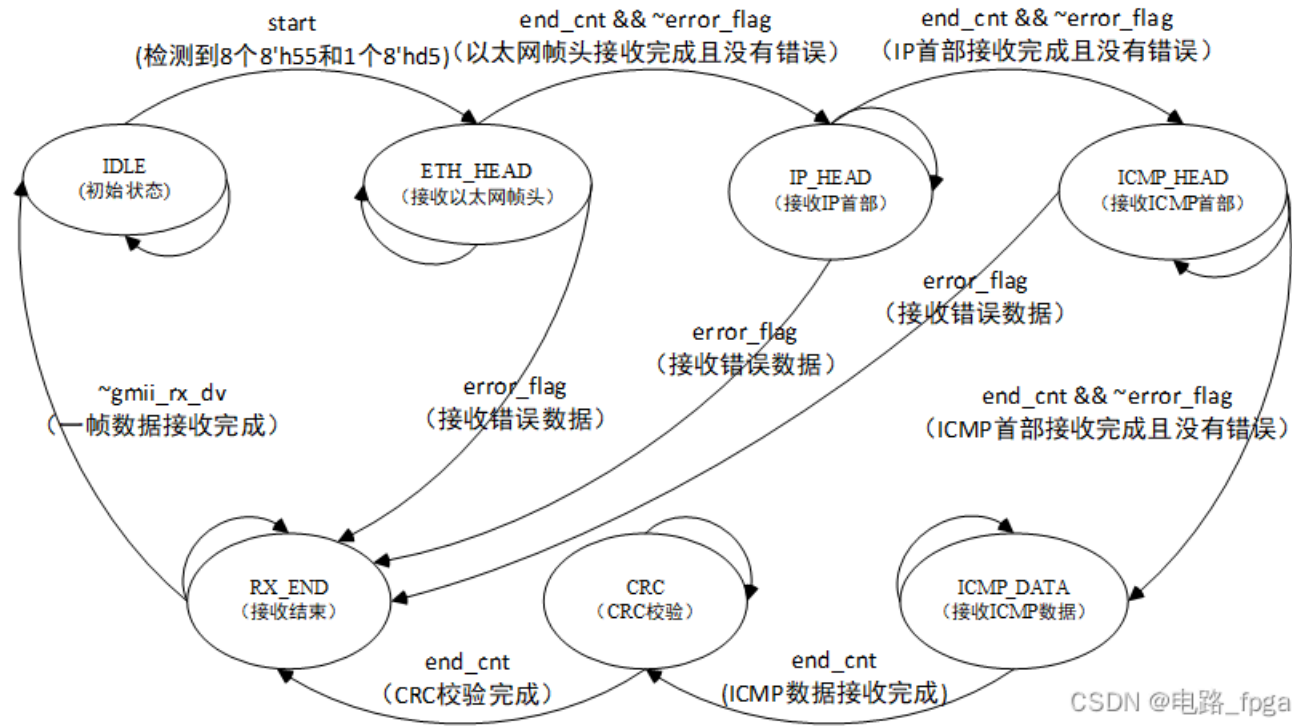


图4 ICMP接收模块状态转换图

后面各个状态分别接收对应数据，计数器cnt用来计数每个状态应该接收的数据个数。其中ICMP数据段的数据个数通过IP首部的总长度减去IP首部长度，在减去ICMP首部长度得到，ICMP数据段的长度还要输出，在后续进行回显应答时，从FIFO中读取对应个数的数据输出。

然后就是错误标志信号error_flag，就是接收的数据不是ICMP或者不是发送给开发板的数据时，就会拉高，此时就会把接收的数据报文丢弃。比如在以太网帧头部分检测到接收的MAC地址不是开发板MAC地址或广播地址，此时error_flag拉高，表示该数据报不是发送给开发板的，直接丢弃，不在继续接收。又比如在接收ICMP首部时，检测到该数据报文不是回显请求，则error_flag拉高，直接丢弃该报文，后续的数据不需要存入FIFO中。

注意在接收ICMP数据时，需要将接收的两字节数据拼接后相加，得到校验和（这是因为回显应答时需要先发送ICMP校验和，后发送ICMP数据，且需要发送的ICMP的数据存在FIFO中，提前取出不方便，所以在接收的时候就把数据段相加，得到数据段的累加和，后续在回显应答时直接使用即可）。也就是前文介绍的IP首部校验和计算方式，但是此处只把接收到的两字节数据相加，因为ICMP的校验和还包括ICMP首部，其余运算在ICMP发送时才能继续。

其余部分都比较简单，可以自行查看工程对应文件，参考代码如下：

```
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//接收完以IP帧头数据，且没有出现错误，则继续接收ICMP协议数据。
38             state_n = ICMP_HEAD;
39         end
40         else begin
41             state_n = state_c;
42         end
43     end
44     ICMP_HEAD:begin
45         if(error_flag)begin//在接收ICMP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ICMP帧头数据，且没有出现错误，则继续接收ICMP数据。
49             state_n = ICMP_DATA;
50         end
51         else begin
52             state_n = state_c;
53         end
54     end
55     ICMP_DATA:begin
56         if(error_flag)begin//在接收ICMP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ICMP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

```

```

72         state_n = state_c;
73     end
74 end
75 RX_END:begin
76     if(~gmii_rx_dv)begin//检测到数据线上数据无效。
77         state_n = IDLE;
78     end
79     else begin
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

```

```

113 //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。
114 always@(posedge clk)begin
115     if(rst_n==1'b0)begin//
116         cnt <= 0;
117     end
118     else if(add_cnt)begin
119         if(end_cnt)
120             cnt <= 0;
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             ICMP_HEAD: cnt_num <= 16'd8;//ICMP帧头为8字节数据。
142             ICMP_DATA: cnt_num <= icmp_data_length;//ICMP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     end
155     else if((state_c == ETH_HEAD) && add_cnt && cnt < 5'd6)begin
156         des_mac_t <= {des_mac_t[39:0],gmii_rxd_r[0]};
157     end
158 end
159
160 //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。
161 always@(posedge clk)begin
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ICMP协议，停止解析数据。
177                         error_flag <= (gmii_rxd_r[0] != ICMP_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             ICMP_HEAD : begin
183                 if(add_cnt && cnt == 1)begin//ICMP报文类型不是回显请求。
184                     error_flag <= (icmp_type != ECHO_REQUEST);
185                 end
186             end
187             default: error_flag <= 1'b0;
188         endcase
189     end
190 end
191
192 //接收IP首部相关数据;
193 always@(posedge clk)begin
194

```

```

194     if(rst_n==1'b0)begin//初始值为0;
195         ip_head_byte_num <= 6'd20;
196         ip_total_length <= 16'd28;
197         des_ip <= 16'd0;
198         icmp_data_length <= 16'd0;
199     end
200     else if(state_c == IP_HEAD && add_cnt)begin
201         case(cnt)
202             16'd0 : ip_head_byte_num <= {gmii_rxd_r[0][3:0],2'd0};//接收IP首部的字节个数。
203             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。
204             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。
205             16'd4 : icmp_data_length <= ip_total_length - ip_head_byte_num - 8;//计算ICMP报文数据段的长度，ICMP帧头为8字节数据。
206             16'd16,16'd17: des_ip <= {des_ip[7:0],gmii_rxd_r[0]};//接收目的IP地址。
207             default: ;
208         endcase
209     end
210 end
211
212 //接收ICMP首部相关数据;
213 always@(posedge clk)begin
214     if(rst_n==1'b0)begin//初始值为0;
215         icmp_type <= 8'd0;
216         icmp_code <= 8'd0;
217         icmp_checksum <= 16'd0;
218         icmp_id <= 16'd0;
219         icmp_seq <= 16'd0;
220     end
221     else if(state_c == ICMP_HEAD && add_cnt)begin
222         case(cnt)
223             16'd0 : icmp_type <= gmii_rxd_r[0];//接收ICMP报文类型。
224             16'd1 : icmp_code <= gmii_rxd_r[0];//接收ICMP报文代码。
225             16'd2,16'd3 : icmp_checksum <= {icmp_checksum[7:0],gmii_rxd_r[0]};//接收ICMP报文帧头和数据的校验和。
226             16'd4,16'd5 : icmp_id <= {icmp_id[7:0],gmii_rxd_r[0]};//接收ICMP的ID。
227             16'd6,16'd7 : icmp_seq <= {icmp_seq[7:0],gmii_rxd_r[0]};//接收ICMP报文的序列号。
228             default: ;
229         endcase
230     end
231 end
232
233 //计算接收到的数据的校验和。
234
235

```

```

235 always@(posedge clk)begin
236     if(rst_n==1'b0)begin//初始值为0;
237         reply_checksum_add <= 16'd0;
238     end
239     else if(state_c == RX_END)begin//累加器清零。
240         reply_checksum_add <= 16'd0;
241     end
242     else if(state_c == ICMP_DATA && add_cnt)begin
243         if(end_cnt && icmp_data_length[0])begin//如果计数器计数结束且数据个数为奇数个，那么直接将当前数据与累加器相加。
244             reply_checksum_add <= reply_checksum_add + {8'd0,gmii_rxd_r[0]};
245         end
246         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。
247             reply_checksum_add <= reply_checksum_add + {gmii_rxd_r[1],gmii_rxd_r[0]};
248     end
249 end
250
251 //控制FIFO使能信号，以及数据信号。
252 always@(posedge clk)begin
253     fifo_wdata <= (state_c == ICMP_DATA) ? gmii_rxd_r[0] : fifo_wdata;//在接收ICMP数据阶段时，接收数据。
254     fifo_wr <= (state_c == ICMP_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。
255 end
256
257 //生产CRC校验相关的数据和控制信号。
258 always@(posedge clk)begin
259     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。
260     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。
261     crc_en <= (state_c != IDLE) && (state_c != RX_END) && (state_c != CRC);//CRC校验使能信号。
262 end
263
264 //接收PC端发送来的CRC数据。
265 always@(posedge clk)begin
266     if(rst_n==1'b0)begin//初始值为0;
267         des_crc <= 24'hff_ff_ff;
268     end
269     else if(add_cnt && state_c == CRC)begin//先接收的是低位数据;
270         des_crc <= {gmii_rxd_r[0],des_crc[23:8]};
271     end
272 end
273
274 //生成相应的输出数据。
275

```

```

276     always@(posedge clk)begin
277         if(rst_n==1'b0)begin//初始值为0;
278             rec_pkt_done <= 1'b0;
279             data_byte_num <= 16'd0;
280             data_checksum <= 16'd0;
281             end//如果CRC校验成功，把ICMP协议接收完成信号拉高，把接收到ICMP数据个数和数据段的校验和输出。
282             else if(state_c == CRC && end_cnt && ({gmii_rxd_r[0],des_crc[23:0]} == crc_out))begin
283                 rec_pkt_done <= 1'b1;
284                 data_byte_num <= icmp_data_length;
285                 data_checksum <= reply_checksum_add;
286             end
287             else begin
288                 rec_pkt_done <= 1'b0;
289             end
290         end
291     end

```



仿真结果如下图所示，TestBench与ICMP发送模块共用，在下文ICMP顶层模块处提供。

如下图所示，当移位寄存器和输入数据gmii_rdv检测到前导码和帧起始符后，start信号拉高(天蓝色信号)，然后状态机（紫红色信号分别是状态机的次态跟现态）跳转到接收以太网帧头的状态，并且可以看到移位寄存器的最低数据gmii_rxd_r[0]数据与状态机的现态state_c是对齐的。

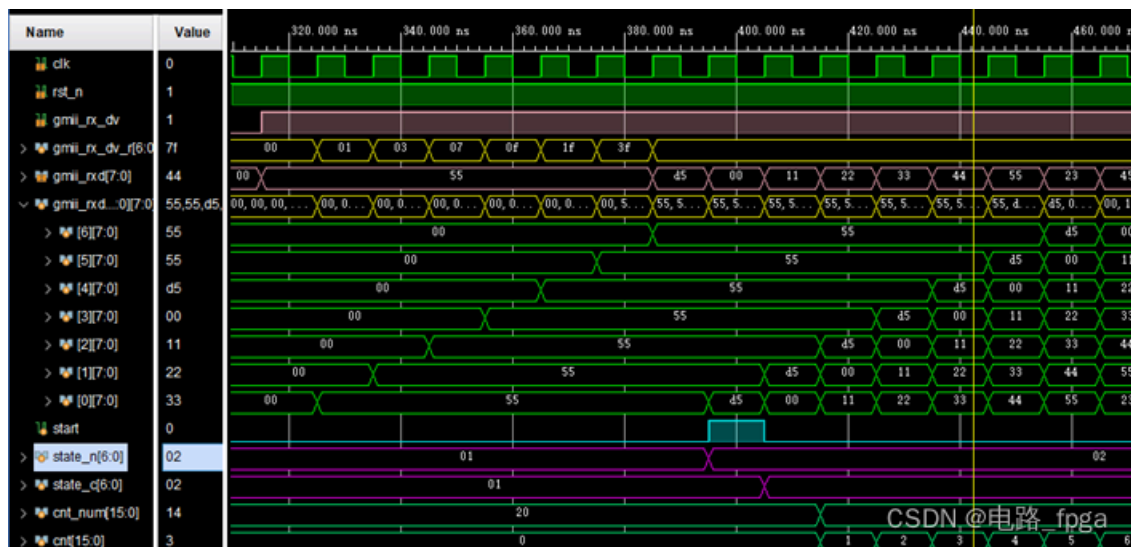


图5 ICMP接收模块仿真

然后接收以太网帧头，接收到目的MAC地址为48'h001122334455，与开发板的目的MAC一致，则继续接收数据。并且后续协议类型为16'h0800，是IP协议，则状态跳转到接收IP头部数据。

把crc校验模块的使能信号拉高，且把接收到的数据gmii_rxd_r[0]的数据输出给crc校验模块进行计算，如下图四个紫红色与crc有关的信号就是crc校验模块相关的信号。

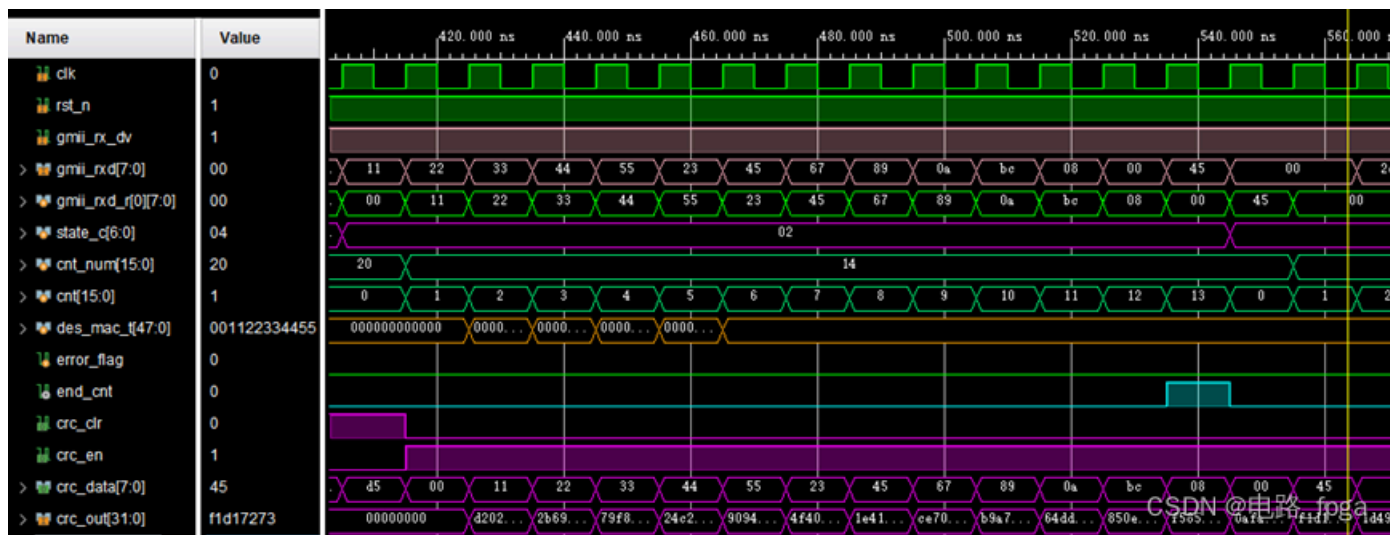


图6 接收以太网帧头数据

下图是接收IP头部数据，首先接收IP头部长度为20字节，然后接收IP报文总长度为46字节，计算出ICMP数据段为18字节的长度。在计数器cnt为9时，gmii_rxd_r[0]为1，表示后面是ICMP协议，计数器cnt等于18时，{des_ip, gmii_rxd_r[0], gmii_rxd}=32'hc0a8010a，与开发板的目的IP地址一致，则接收的数据报是发送给开发板的ICMP数据报。

计数器计数到最大值后，状态机跳转到接收ICMP首部数据状态。整个过程中CRC校验模块一直在对接收的数据进行校验计算。

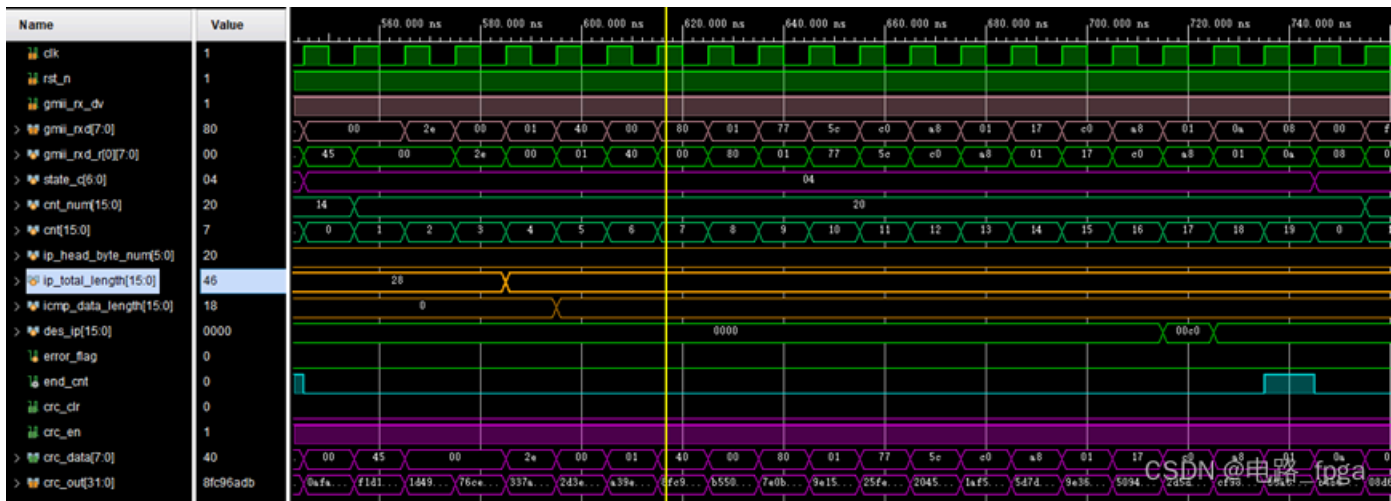


图7 接收IP头部数据

状态机在接收ICMP首部数据的仿真如下图所示，接收到的类型为8，代码为0，则表示该ICMP数据报是ICMP回显请求。继续接收ICMP的标识符为1，序列号为8，将序列号和标识符输出。注意crc校验模块依旧在对接收的数据进行计算。ICMP首部接收完成后状态机跳转到接收ICMP数据状态。

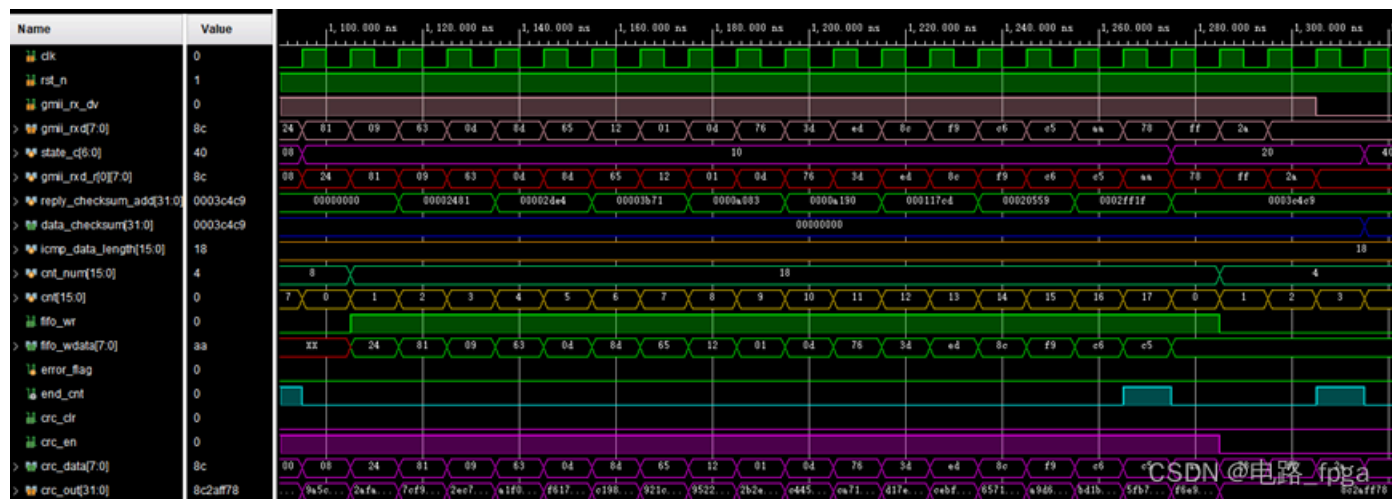


图9 接收ICMP数据

下图是接收CRC校验阶段，如图所示，计数器为3时，{gmii_rxd_r[0],des_crc} = 32'h8c2aff78，与crc校验模块计算的结果一致，则表示接收的数据正确，把rec_pkt_done信号拉高一个时钟周期，表示接收数据完成，把ICMP数据段的长度和数据校验和输出，便于后面ICMP回显应答使用。

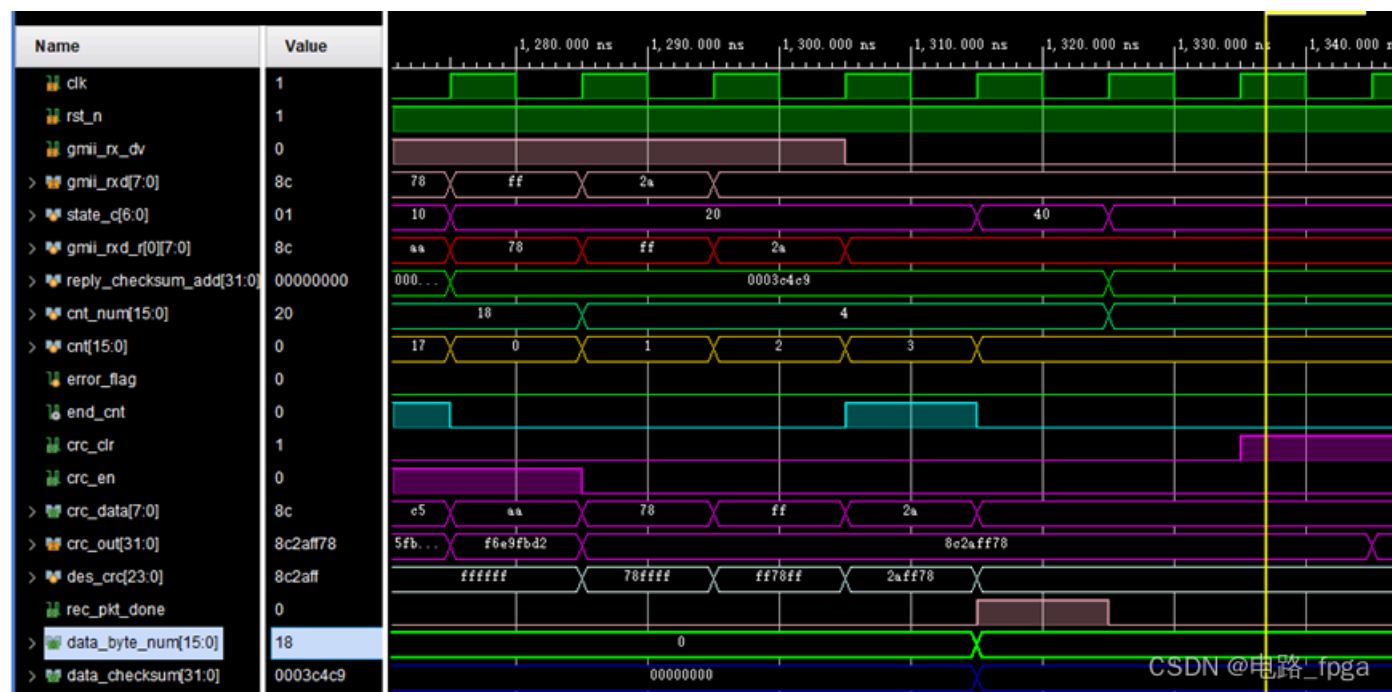


图10 接收CRC校验数据

3、ICMP发送模块

该模块设计比较简单，通过一个状态机，嵌套计数器就可以完成。状态转换图如下所示。

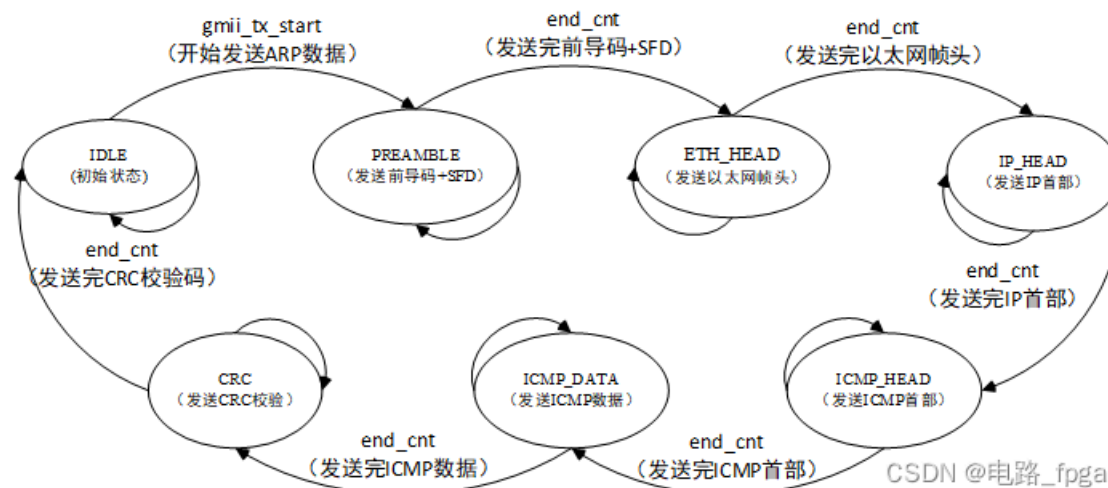


图11 ICMP发送模块状态转换图

设计思路与ARP发送模块没有太大区别，相比ARP发送模块，会稍微复杂一点，需要注意两点：

1. ICMP发送模块需要在发送IP首部和ICMP首部之前计算校验码，本设计是在发送以太网帧头的时候，同步计算出IP首部校验和、ICMP校验和，然后发送IP首部和ICMP首部时直接使用即可，也不会占用额外的时钟周期。
2. ICMP的数据段需要从外部FIFO（FIFO的配置在后文出现）中读取数据，本文使用的FIFO工作在超前模式，也就是读使能有效的时候，读数据就是有效的，不需要提前产生读使能。特别注意FIFO输出数据与数据流的对接问题。

计数器cnt的位宽扩展到16位，因为ICMP数据段可能会很长，所有计数器就与IP首部的总长度位宽保持一致。

其余设计与ARP发送模块基本一致，本文不再赘述，参考代码如下：

```

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      icmp_head[0] <= 32'd0;
9      icmp_head[1] <= 32'd0;
10     ip_head_check <= 32'd0;
11     icmp_check <= 32'd0;
12     des_ip_r <= DES_IP;
13     des_mac_r <= DES_MAC;
14     tx_byte_num_r <= MIN_DATA_NUM;
15     ip_total_num <= MIN_DATA_NUM + 28;
16 end
17 //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ICMP需要发送的数据个数进行暂存。
18 else if(state_c == IDLE && icmp_tx_en)begin
19     icmp_head[1] <= {icmp_id,icmp_seq};//16位ICMP标识符和16位序列号。
20     icmp_check <= reply_checksum;//将数据段的校验和暂存。
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ICMP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ICMP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头部长度的，IP服务类型，IP包的总长度。
34             ip_head[2] <= {8'h80,8'd01,16'd0};//分别表示生存时间，协议类型，1表示ICMP，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             icmp_head[0] <= {ECHO_REPLY,24'd0};//8位类型与8位代码，16位的校验码。
38         end
39         16'd1 : begin//开始计算IP头部和ICMP的校验和数据，并且将计算结果存储到对应位置。
40             ip_head_check <= ip_head[0][31 : 16] + ip_head[0][15 : 0];
41             icmp_check <= icmp_check + icmp_head[0][31 : 16];
42         end
43         16'd2 : begin
44             ip_head_check <= ip_head_check + ip_head[1][31 : 16];
45             icmp_check <= icmp_check + icmp_head[1][31 : 16];
46         end
47     end

```

```

48     16'd3 : begin
49         ip_head_check <= ip_head_check + ip_head[1][15 : 0];
50         icmp_check <= icmp_check + icmp_head[1][15 : 0];
51     end
52     16'd4 : begin
53         ip_head_check <= ip_head_check + ip_head[2][31 : 16];
54         icmp_check <= icmp_check[31 : 16] + icmp_check[15 : 0]; //可能出现进位,累加一次。
55     end
56     16'd5 : begin
57         ip_head_check <= ip_head_check + ip_head[3][31 : 16];
58         icmp_check <= icmp_check[31 : 16] + icmp_check[15 : 0]; //可能出现进位,再累加一次。
59     end
60     16'd6 : begin
61         ip_head_check <= ip_head_check + ip_head[3][15 : 0];
62         icmp_head[0][15:0] <= ~icmp_check[15 : 0]; //按位取反得到校验和。
63         icmp_check <= 32'd0; //将校验和清零,便于下次使用。
64     end
65     16'd7 : begin
66         ip_head_check <= ip_head_check + ip_head[4][31 : 16];
67     end
68     16'd8 : begin
69         ip_head_check <= ip_head_check + ip_head[4][15 : 0];
70     end
71     16'd9,16'd10 : begin
72         ip_head_check <= ip_head_check[31 : 16] + ip_head_check[15 : 0];
73     end
74     16'd11 : begin
75         ip_head[2][15:0] <= ~ip_head_check[15 : 0];
76         ip_head_check <= 32'd0; //校验和清零,用于下次计算。
77     end
78     default: begin
79         icmp_check <= 32'd0; //将校验和清零,便于下次使用。
80         ip_head_check <= 32'd0; //校验和清零,用于下次计算。
81     end
82 endcase
83 end
84 else if(state_c == IP_HEAD && end_cnt)
85     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。
86 end
87
88

```

```

89 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
90 always@(posedge clk)begin
91     if(!rst_n)begin
92         state_c <= IDLE;
93     end
94     else begin
95         state_c <= state_n;
96     end
97 end
98
99 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
100 always@(*)begin
101     case(state_c)
102         IDLE:begin
103             if(icmp_tx_en)begin//在空闲状态接收到上游发出的使能信号;
104                 state_n = PREAMBLE;
105             end
106             else begin
107                 state_n = state_c;
108             end
109         end
110         PREAMBLE:begin
111             if(end_cnt)begin//发送完前导码和SFD;
112                 state_n = ETH_HEAD;
113             end
114             else begin
115                 state_n = state_c;
116             end
117         end
118         ETH_HEAD:begin
119             if(end_cnt)begin//发送完以太网帧头数据;
120                 state_n = IP_HEAD;
121             end
122             else begin
123                 state_n = state_c;
124             end
125         end
126         IP_HEAD:begin
127             if(end_cnt)begin//发送完IP帧头数据;
128                 state_n = ICMP_HEAD;
129

```

```

129         end
130     else begin
131         state_n = state_c;
132     end
133 end
134 ICMP_HEAD:begin
135     if(end_cnt)begin//发送完ICMP帧头数据:
136         state_n = ICMP_DATA;
137     end
138     else begin
139         state_n = state_c;
140     end
141 end
142 ICMP_DATA:begin
143     if(end_cnt)begin//发送完icmp协议数据:
144         state_n = CRC;
145     end
146     else begin
147         state_n = state_c;
148     end
149 end
150 CRC:begin
151     if(end_cnt)begin//发送完CRC校验码;
152         state_n = IDLE;
153     end
154     else begin
155         state_n = state_c;
156     end
157 end
158 default:begin
159     state_n = IDLE;
160 end
161 endcase
162 end
163
164 //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。
165 always@(posedge clk)begin
166     if(rst_n==1'b0)begin//
167         cnt <= 0;
168     end
169 end
170

```



```

170         else if(add_cnt)begin
171             if(end_cnt)
172                 cnt <= 0;
173             else
174                 cnt <= cnt + 1;
175         end
176     end
177 end
178
179 assign add_cnt = (state_c != IDLE); //状态机不在空闲状态时计数。
180 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。
181
182 //状态机在每个状态需要发送的数据个数。
183 always@(posedge clk)begin
184     if(rst_n==1'b0)begin //初始值为20;
185         cnt_num <= 16'd20;
186     end
187     else begin
188         case (state_c)
189             PREAMBLE : cnt_num <= 16'd8; //发送7个前导码和1个8'hd5。
190             ETH_HEAD : cnt_num <= 16'd14; //发送14字节的以太网帧头数据。
191             IP_HEAD : cnt_num <= 16'd20; //发送20个字节是IP帧头数据。
192             ICMP_HEAD : cnt_num <= 16'd8; //发送8字节的ICMP帧头数据。
193             ICMP_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。
194                 cnt_num <= tx_byte_num_r;
195                 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。
196                     cnt_num <= MIN_DATA_NUM;
197             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。
198             default: cnt_num <= 6'd20;
199         endcase
200     end
201 end
202
203 //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。
204 always@(posedge clk)begin
205     if(rst_n==1'b0)begin //初始值为0;
206         crc_data <= 8'd0;
207     end
208     else if(add_cnt)begin
209         case (state_c)
210             PREAMBLE : if(end_cnt)

```

```

211         crc_data <= 8'h5; //发送1字节SFD编码;
212     else
213         crc_data <= 8'h55; //发送7字节前导码;
214 ETH_HEAD : if(cnt < 6)
215         crc_data <= des_mac_r[47 - 8*cnt -: 8]; //发送目的MAC地址, 先发高字节;
216     else if(cnt < 12)
217         crc_data <= BOARD_MAC[47 - 8*(cnt-6) -: 8]; //发送源MAC地址, 先发高字节;
218     else
219         crc_data <= ETH_TYPE[15 - 8*(cnt-12) -: 8]; //发送源以太网协议类型, 先发高字节;
220 IP_HEAD : if(cnt < 4) //发送IP帧头。
221         crc_data <= ip_head[0][31 - 8*cnt -: 8];
222     else if(cnt < 8)
223         crc_data <= ip_head[1][31 - 8*(cnt-4) -: 8];
224     else if(cnt < 12)
225         crc_data <= ip_head[2][31 - 8*(cnt-8) -: 8];
226     else if(cnt < 16)
227         crc_data <= ip_head[3][31 - 8*(cnt-12) -: 8];
228     else
229         crc_data <= ip_head[4][31 - 8*(cnt-16) -: 8];
230 ICMP_HEAD : if(cnt < 4) //发送ICMP帧头数据。
231         crc_data <= icmp_head[0][31 - 8*cnt -: 8];
232     else
233         crc_data <= icmp_head[1][31 - 8*(cnt-4) -: 8];
234 ICMP_DATA : if(cnt_num >= 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。
235         crc_data <= fifo_rdata; //如果满足最小要求, 将从FIFO读出的数据输出即可。
236     else if(cnt < cnt_num) //不满足最小要求时, 先将需要发送的数据发送完。
237         crc_data <= fifo_rdata; //将从FIFO读出的数据输出即可。
238     else //剩余数据补充0。
239         crc_data <= 8'd0;
240     default : ;
241 endcase
242 end
243 end
244
245 //fifo读使能信号, 初始值为0, 当发送完ICMP帧头时拉高, 当发送完ICMP数据时拉低。
246 always@(posedge clk)begin
247     if(rst_n==1'b0)begin //初始值为0;
248         fifo_rd_en <= 1'b0;
249     end
250     else if(state_c == ICMP_HEAD && end_cnt)begin
251

```

```

252         fifo_rd_en <= 1'b1;
253     end
254     else if(state_c == ICMP_DATA && end_cnt)begin
255         fifo_rd_en <= 1'b0;
256     end
257 end
258
259 //生成一个crc_data指示信号，用于生成gmii_txd信号。
260 always@(posedge clk)begin
261     if(rst_n==1'b0)begin//初始值为0;
262         gmii_tx_en_r <= 1'b0;
263     end
264     else if(state_c == CRC)begin
265         gmii_tx_en_r <= 1'b0;
266     end
267     else if(state_c == PREAMBLE)begin
268         gmii_tx_en_r <= 1'b1;
269     end
270 end
271
272 //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。
273 always@(posedge clk)begin
274     if(rst_n==1'b0)begin//初始值为0;
275         crc_en <= 1'b0;
276     end
277     else if(state_c == CRC)begin//当ARP和以太网帧头数据全部输出后拉低。
278         crc_en <= 1'b0;
279     end//当开始输出以太网帧头时拉高。
280     else if(state_c == ETH_HEAD && add_cnt)begin
281         crc_en <= 1'b1;
282     end
283 end
284
285 //生产CRC校验模块清零信号，状态机处于空闲时清零。
286 always@(posedge clk)begin
287     crc_clr <= (state_c == IDLE);
288 end
289
290 //生成gmii_txd信号，默认输出0。
291 always@(posedge clk)begin
292

```

```

293     if(rst_n==1'b0)begin//初始值为0;
294         gmii_txd <= 8'd0;
295     end//在输出CRC状态时，输出CRC校验码，先发送低位数据。
296     else if(state_c == CRC && add_cnt && cnt>0)begin
297         gmii_txd <= crc_out[8*cnt-1 -: 8];
298     end//其余时间如果crc_data有效，则输出对应数据。
299     else if(gmii_tx_en_r)begin
300         gmii_txd <= crc_data;
301     end
302 end
303
304 //生成gmii_txd有效指示信号。
305 always@(posedge clk)begin
306     gmii_tx_en <= gmii_tx_en_r || (state_c == CRC);
307 end
308
309 //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。
310 //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(icmp_tx_en || state_c != IDLE)
        rdy = 1'b0;
    else
        rdy = 1'b1;
end

```



TestBench与ICMP接收模块共用，在后文出现，仿真如下所示，当检测到开始发送数据信号有效时，将ICMP数据长度、数据段的校验和reply_checksum、目的MAC地址、目的IP地址保存，计算出IP的报文总长度。

状态机跳转到发送前导码和帧起始符状态，crc_data这个数据延时一拍就会作为输出数据gmii_txd。

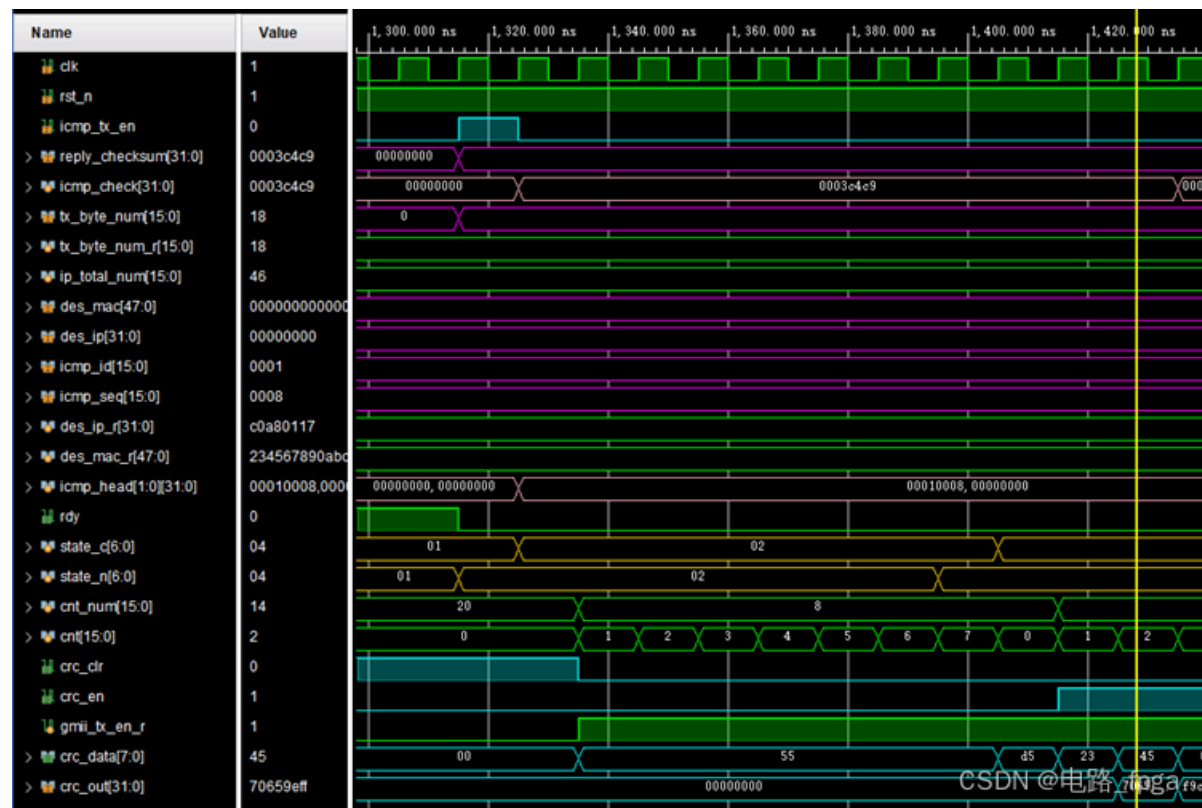


图12 开始发送数据

状态机处于发送以太网帧头状态时，还在计算IP首部和ICMP的校验和，并且将计算结果存储到IP首部和ICMP首部存储体的对应位置，仿真如下图所示。



图13 产生以太网帧头且计算校验和

下图时状态机处于发送IP首部状态，将IP首部存储体中的数据依次输出，蓝色信号为IP首部存储体数据，crc_data是输出给crc校验模块计算的数据，该信号延迟一个时钟周期后得到gmii_txd输出信号。

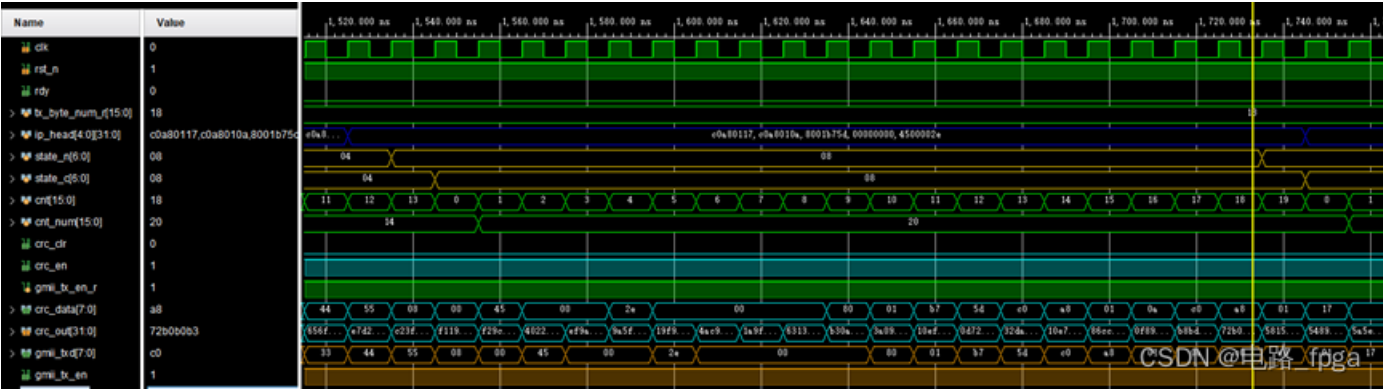


图14 发送IP首部数据

下图是发送ICMP首部存储体中的数据，与上图类似。

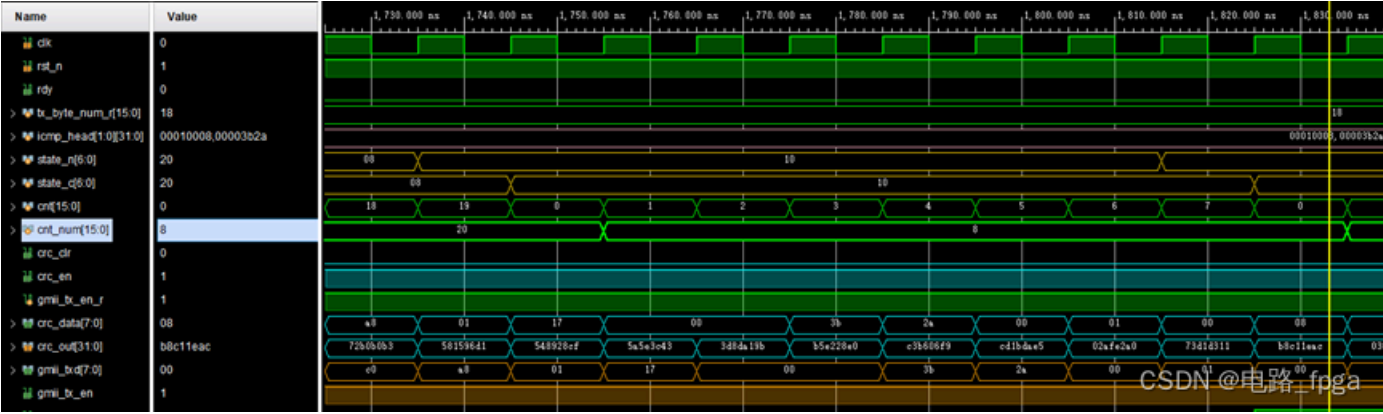


图15 发送ICMP首部数据

发送完ICMP首部数据后，从fifo中读取tx_byte_num_r个数据输出，如下图所示。FIFO读使能与读数据对齐，所以直接使用即可。

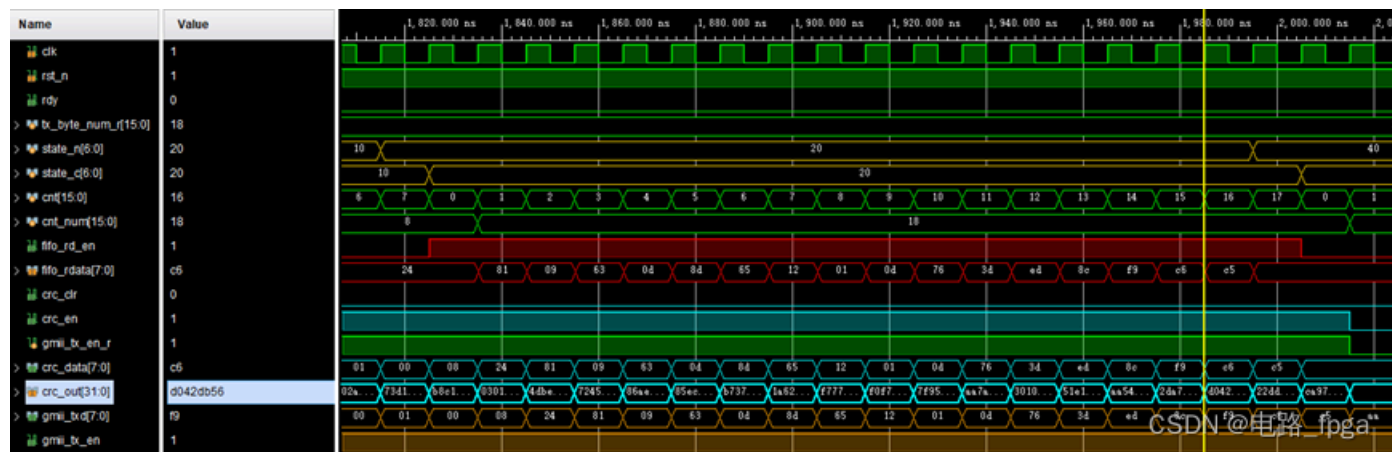


图16 发送ICMP数据

最后就是CRC校验，由于CRC校验模块输出数据会滞后输入数据一个时钟周期，导致需要把crc_data延迟一个时钟周期后在接上CRC校验模块输出的数据，才算正确，这也是为什么需要把crc_data延时一个时钟得到gmii_txd的原因。

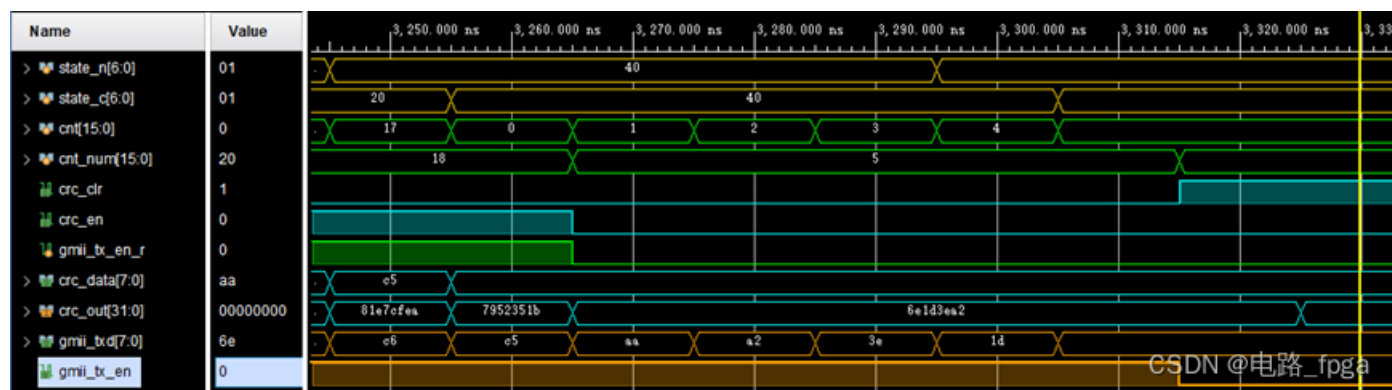


图17 发送CRC数据

ICMP发送模块的设计和仿真到此结束了。

4、FIFO IP设置

FIFO IP设置为超前模式，这样读数据时，读使能和读数据就能直接对齐了，读数据不会滞后读使能，这样用起来更方便。

位宽设置为8位，数据深度设置为1024字节，设置为2048更好。

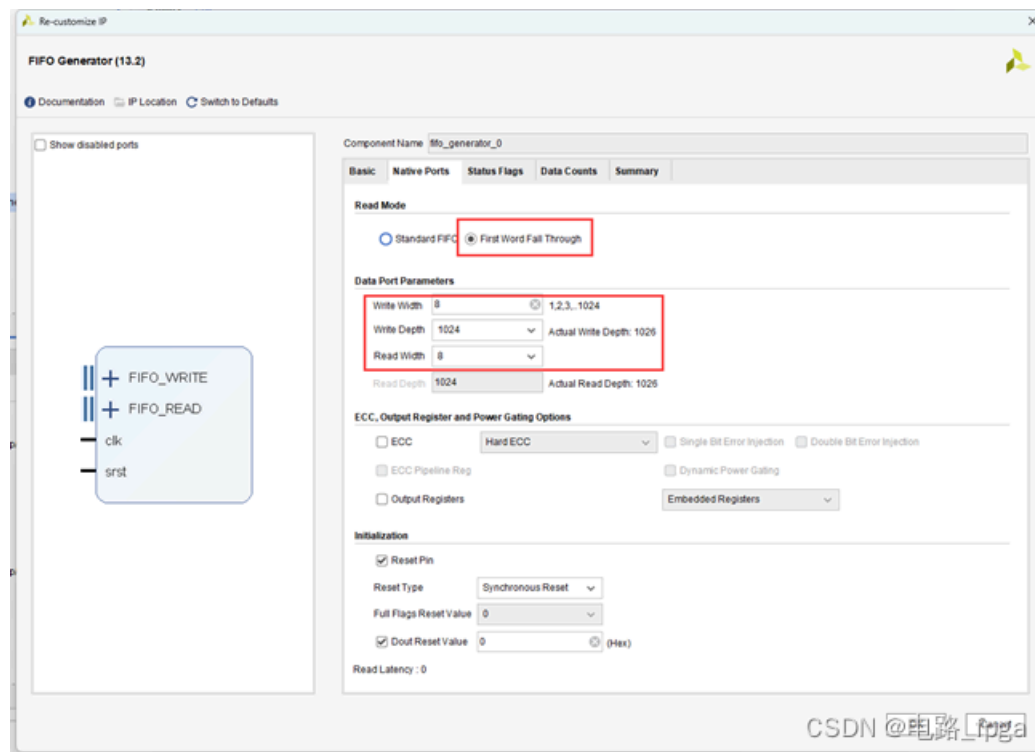


图18 FIFO IP设置

其余设置默认即可，复位采用低电平有效。

5、ARP和ICMP控制模块

arp和icmp的控制模块如下所示，当arp发送模块输出数据且icmp发送模块空闲时，将arp发送模块的输出作为gmii_txd的数据。如果icmp发送模块输出有效数据且arp发送模块空闲时，将icmp发送模块的输出作为gmii_txd的数据。

当arp接收模块接收到数据后，将arp发送模块使能信号拉高，当icmp接收模块到回显请求时，把icmp发送模块的使能信号拉高，实现回显应答。

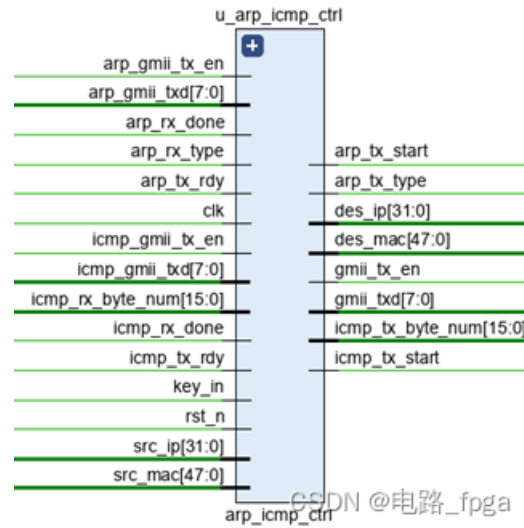


图19 控制模块

该模块的参考代码如下所示：

```

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 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
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 //对两个模块需要发送的数据进行整合。
50 always@(posedge clk)begin
51     if(rst_n==1'b0)begin//初始值为0;
52         gmii_tx_en <= 1'b0;
53         gmii_txd <= 8'd0;
54     end//如果ARP发送模块输出有效数据，且ICMP发送模块处于空闲状态，则将ARP相关数据输出。
55     else if(arp_gmii_tx_en && icmp_tx_rdy)begin
56         gmii_tx_en <= arp_gmii_tx_en;
57         gmii_txd <= arp_gmii_txd;
58     end//如果ICMP发送模块输出有效数据且ARP发送模块处于空闲，则将ICMP相关数据输出。
59     else if(icmp_gmii_tx_en && arp_tx_rdy)begin
60         gmii_tx_en <= icmp_gmii_tx_en;
61
62

```

```

62         gmii_txd <= icmp_gmii_txd;
63     end
64     else begin
65         gmii_tx_en <= 1'b0;
66     end
end

```



由于模块比较简单，所以不再单独仿真，后文直接上板测试即可。

6、上板测试

在工程中加入ILA，综合工程，然后下载到开发板，最后打开wireshark软件，该软件在ARP实现文中已经使用过，不再赘述。将电脑的IP设置为顶层文件的目的IP地址，设置方式在ARP文中也做了详细介绍，不知道怎么设置的可以去看看。

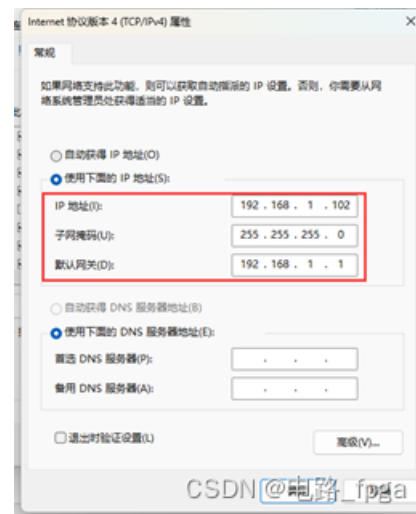


图20 设置电脑的IP

然后以管理员身份选打开命令提示符，然后运行wireshark，把gmii_rx_dv上升沿作为ILA的触发条件，连续抓取32帧数据。在命令提示符中发送ping 192.168.1.10，如下图所示。

```
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

CSDN @电路_fpga
```

图图21 ping指令

Ping指令运行结果如下所示，一般PC会发送4次回显请求，如果四次回显请求都被应答，则认为ping通了，丢失为0。

```
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22 ping结果

Wireshark抓取的数据报如下所示，粉色信号就是ICMP的回显请求和回显应答数据报。比如第9和10数据报，分别是PC发给FPGA的回显请求和FPGA发送给PC端的回显应答。注意两个报文的标识符和序列号是一致的，这也是分别应答和请求数据报的对应关系。

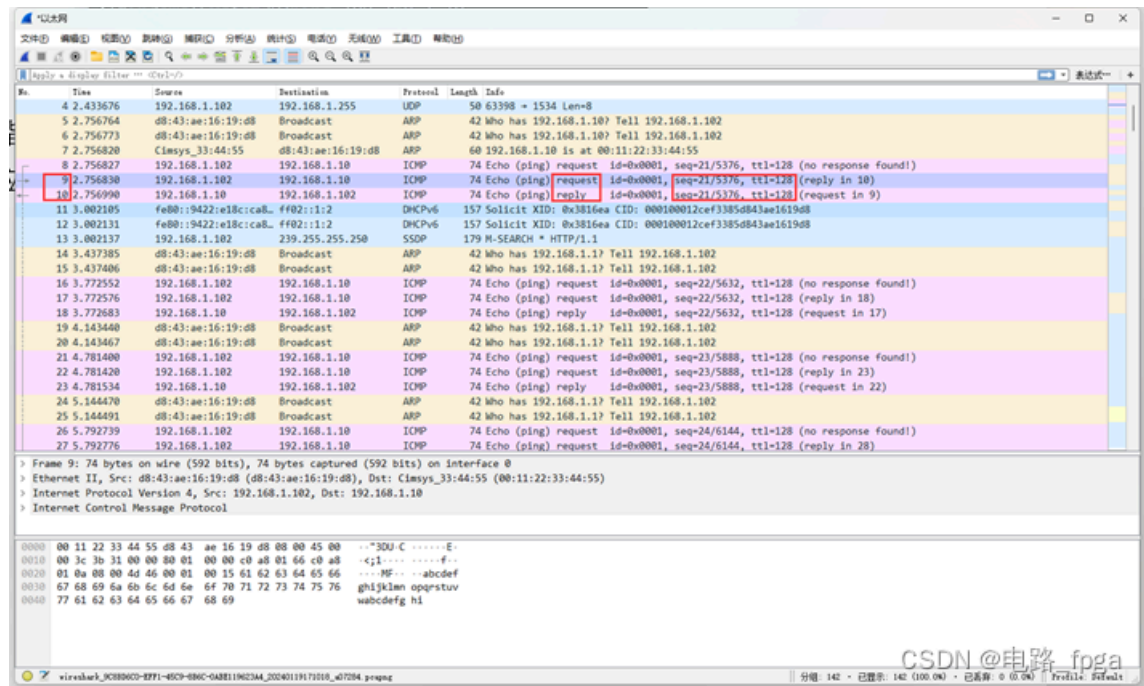


图23 wireshark抓取的回显请求数据报

ILA抓取的PC端发送的回显请求数据报如下所示。

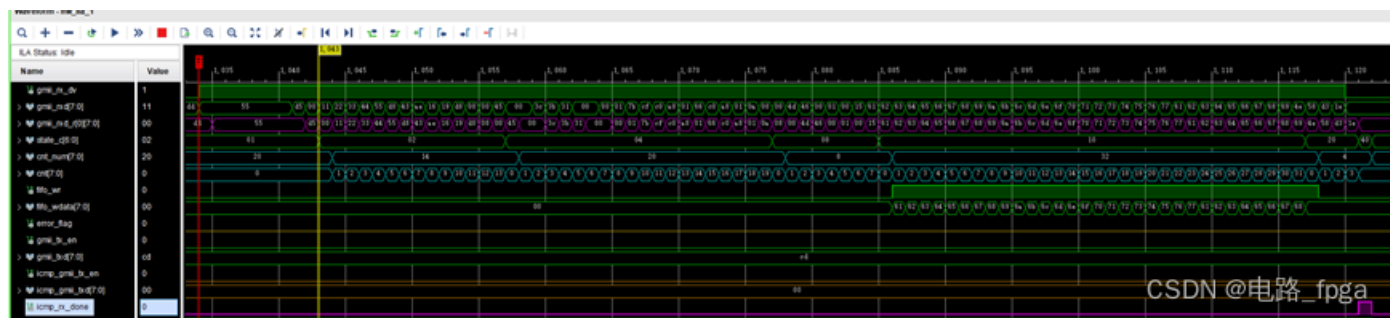


图24 ILA抓取的回显请求数据报

将回显请求数据报的数据段放大，如下图所示，并且与Wireshark的9号数据报的数据段进行对比，可知FPGA接收的数据正确。

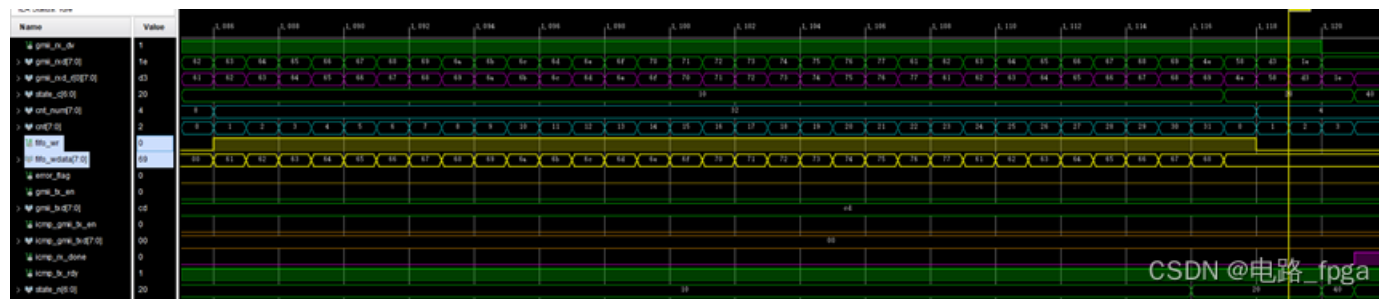


图25 回显请求数据段

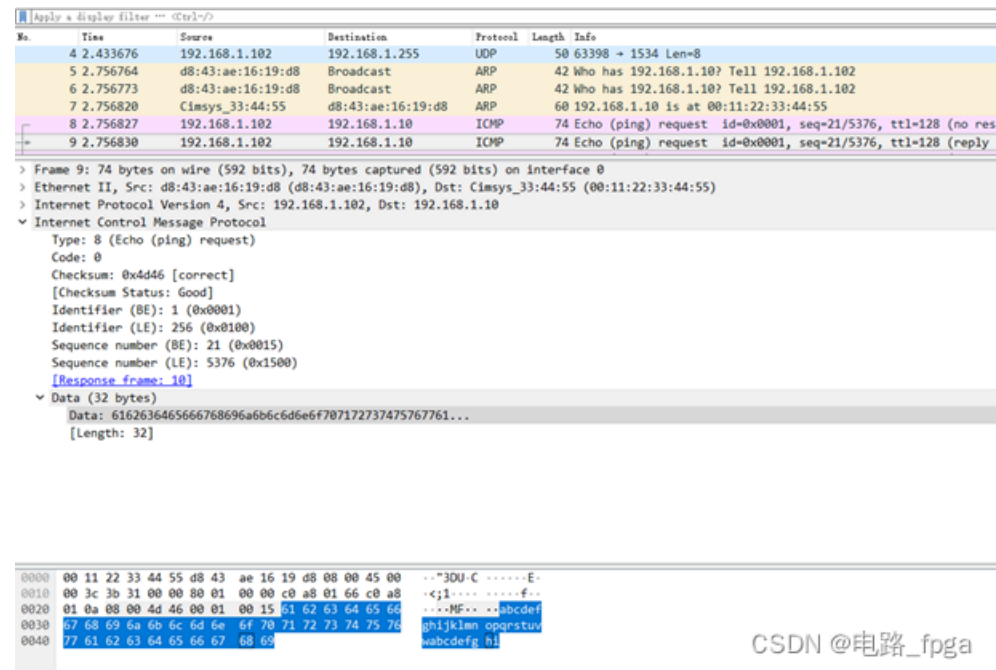


图26 Wireshark回显请求数据报

FPGA在接收到回显请求时，给PC端发出回显应答数据报，ILA抓取该数据报如下图所示。

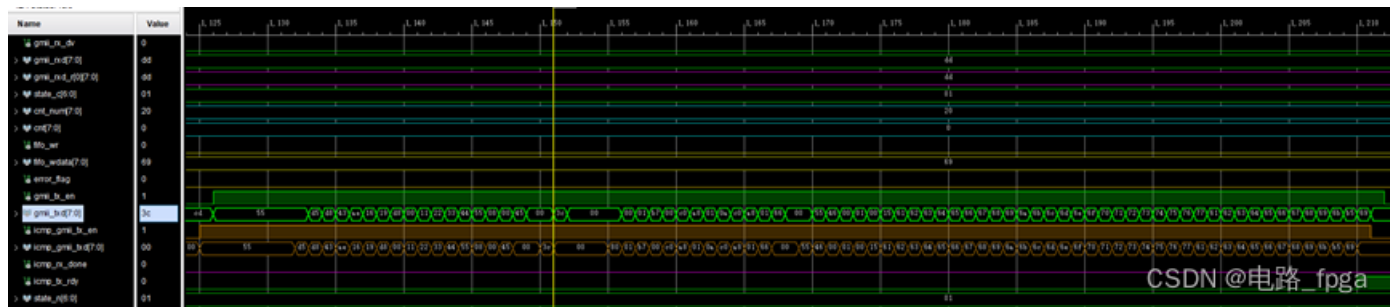


图27 ILA抓取的回显应答数据报

Wireshark抓取的回显应答数据报如下所示，感兴趣的可以使用工程查看。

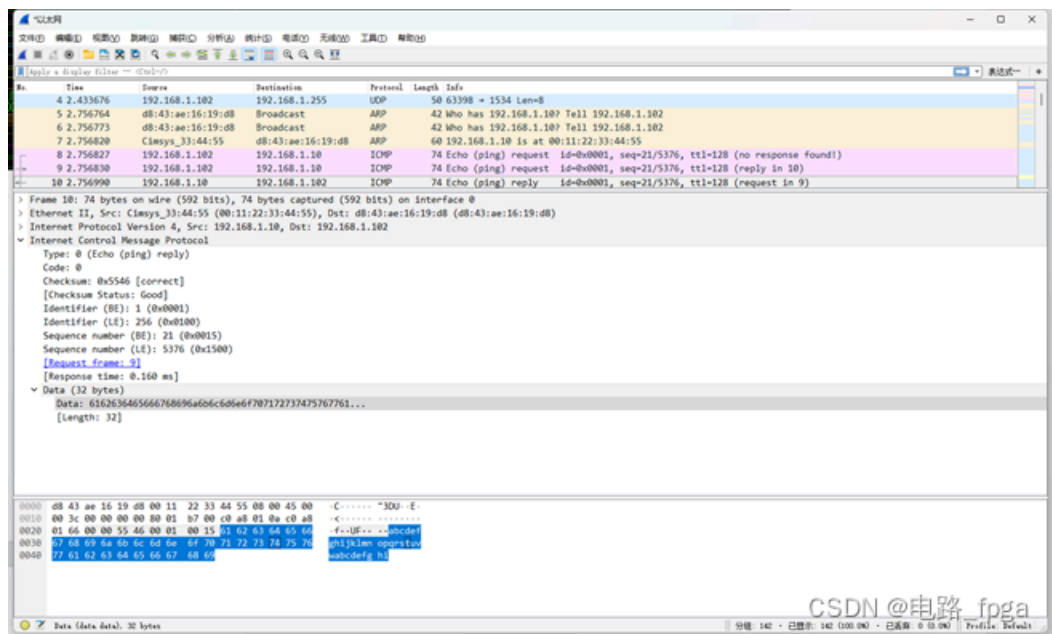


图28 Wireshark回显应答数据报

至于CRC校验这些，与ARP实现的文中是一致的，本文不再赘述，需要了解的可以查看前文。

本工程可以在公众号后台回复“基于FPGA的ICMP实现”（不包含引号）获取。

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⭐！