

## 【实战干货】FPGA实现ARP协议，细节全解析！（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、系统概括

本文主要通过 **FPGA** 实现ARP协议的接收和发送，按键按下后，FPGA会向PC端发送ARP请求指令，PC会对FPGA发送ARP应答。同时当FPGA接收到PC端的ARP请求时，需要把FPGA的IP和MAC地址通过ARP应答发送给PC端。

由于画各个模块的信号流向图比较费时间，所以直接使用vivado的RTL图替代，如下图所示，工程包括5个模块。

key是按键消抖和检测模块， arp\_ctrl是ARP控制模块， 控制ARP模块向PC端发送ARP请求还是ARP应答， ARP模块实现 **ARP协议** 的接收和发送。锁相环模块是将输入100MHz时钟信号转换为200MHz作为IDELAYCTRL的参考时钟信号， rgmii to gmii是RGMII接口信号与GMII接口信号转换模块， 在前文已经做了详细讲解， 本文不再赘述。

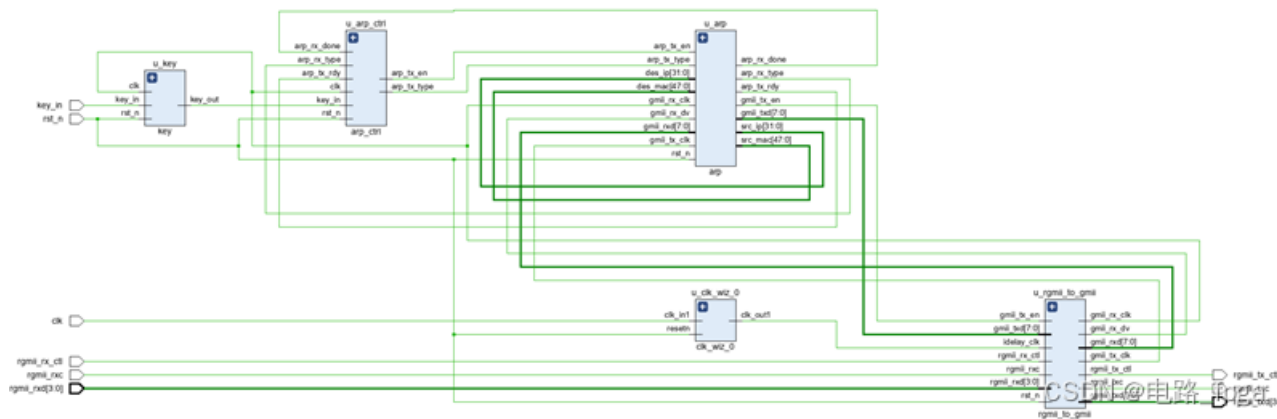


图1 顶层模块信号流向图

每个模块的左侧信号为该模块输入信号，右侧为该模块输出信号。本文主要讲解arp模块和arp\_ctrl模块的设计，按键消抖模块，GMII与RGMII接口信号转换模块在前文均已讲解过，此处直接使用即可。

顶层模块的核心参考代码如下所示:

```
1 assign des_mac = src_mac;
2 assign des_ip  = src_ip;
3
4 //例化锁相环，输出200MHZ时钟，作为IDELAYCTRL的参考时钟。
```

```

5   clk_wiz_0 u_clk_wiz_0 (
6       .clk_out1    ( idelay_clk),//output clk_out1;
7       .resetn      ( rst_n      ),//input resetn;
8       .clk_in1     ( clk         ) //input clk_in1;
9   );
10
11 //例化按键消抖模块。
12 key #(
13     .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。
14     .TIME_CLK     ( TIME_CLK   ) //系统时钟周期，默认8ns。
15 )
16 u_key (
17     .clk          ( gmii_rx_clk ),//系统时钟，125MHz。
18     .rst_n        ( rst_n       ),//系统复位，低电平有效。
19     .key_in       ( key_in      ),//待输入的按键输入信号,默认低电平有效;
20     .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;
21 );
22
23 //例化ARP控制模块;
24 arp_ctrl u_arp_ctrl (
25     .clk          ( gmii_rx_clk ),//输入时钟;
26     .rst_n        ( rst_n       ),//复位信号，低电平有效;
27     .key_in       ( key_out     ),//按键按下，高电平有效;
28     .arp_rx_done  ( arp_rx_done ),//ARP接收完成信号;
29     .arp_rx_type  ( arp_rx_type ),//ARP接收类型 0:请求 1:应答;
30     .arp_tx_rdy   ( arp_tx_rdy  ),//ARP发送模块忙闲指示信号。
31     .arp_tx_en    ( arp_tx_en   ),//ARP发送使能信号;
32     .arp_tx_type  ( arp_tx_type ) //ARP发送类型 0:请求 1:应答;
33 );
34
35 //例化ARP模块;
36 arp #(
37     .BOARD_MAC    ( BOARD_MAC   ),//开发板MAC地址 00-11-22-33-44-55;
38     .BOARD_IP     ( BOARD_IP    ),//开发板IP地址 192.168.1.10;
39     .DES_MAC      ( DES_MAC     ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
40     .DES_IP       ( DES_IP      ) //目的IP地址 192.168.1.102;
41 )
42 u_arp (
43     .rst_n        ( rst_n       ),//复位信号，低电平有效。
44     .gmii_rx_clk  ( gmii_rx_clk ),//GMII接收数据时钟。
45

```

```

46     .gmii_rx_dv      ( gmii_rx_dv      ),//GMII输入数据有效信号。
47     .gmii_rxd       ( gmii_rxd       ),//GMII输入数据。
48     .gmii_tx_clk    ( gmii_tx_clk    ),//GMII发送数据时钟。
49     .arp_tx_en      ( arp_tx_en      ),//ARP发送使能信号。
50     .arp_tx_type    ( arp_tx_type    ),//ARP发送类型 0:请求 1:应答。
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     .gmii_tx_en     ( gmii_tx_en     ),//GMII输出数据有效信号。
54     .gmii_txd       ( gmii_txd       ),//GMII输出数据。
55     .arp_rx_done    ( arp_rx_done    ),//ARP接收完成信号。
56     .arp_rx_type    ( arp_rx_type    ),//ARP接收类型 0:请求 1:应答。
57     .src_mac        ( src_mac        ),//接收到目的MAC地址。
58     .src_ip         ( src_ip         ),//接收到目的IP地址。
59     .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。
60 );
61
62 //例化gmii转RGMII模块。
63 rgmii_to_gmii u_rgmii_to_gmii (
64     .idelay_clk      ( idelay_clk      ),//IDELAY时钟;
65     .rst_n           ( rst_n           ),
66     .gmii_tx_en      ( gmii_tx_en      ),//GMII发送数据使能信号;
67     .gmii_txd        ( gmii_txd        ),//GMII发送数据;
68     .gmii_rx_clk     ( gmii_rx_clk     ),//GMII接收时钟;
69     .gmii_rx_dv      ( gmii_rx_dv      ),//GMII接收数据有效信号;
70     .gmii_rxd        ( gmii_rxd        ),//GMII接收数据;
71     .gmii_tx_clk     ( gmii_tx_clk     ),//GMII发送时钟;
72
73     .rgmii_rxc       ( rgmii_rxc       ),//RGMII接收时钟;
74     .rgmii_rx_ctl    ( rgmii_rx_ctl    ),//RGMII接收数据控制信号;
75     .rgmii_rxd       ( rgmii_rxd       ),//RGMII接收数据;
76     .rgmii_txc       ( rgmii_txc       ),//RGMII发送时钟;
77     .rgmii_tx_ctl    ( rgmii_tx_ctl    ),//RGMII发送数据控制信号;
78     .rgmii_txd       ( rgmii_txd       ) //RGMII发送数据;
79 );

```



## 2、ARP模块设计

下图是ARP模块的内部信号走向，主要包含三个模块，ARP接收模块arp\_rx，ARP发送模块arp\_tx，CRC校验模块。本工程对ARP接收和发送均做了CRC校验，来确保数据的正确性，接收其实可以不做CRC校验的。

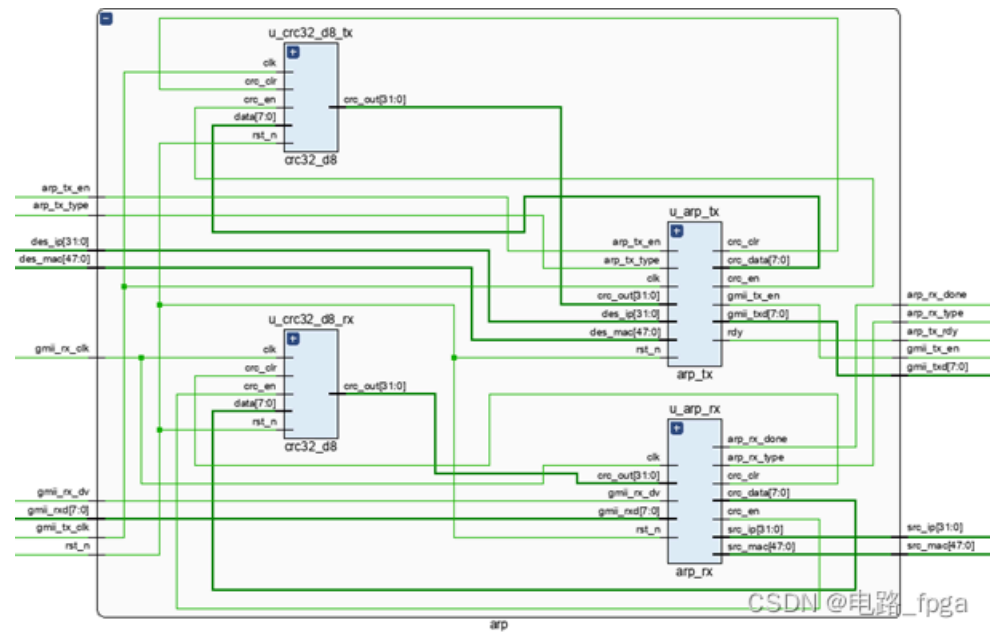


图2 ARP模块内部信号走向

下文分别对这几个模块的设计进行讲解，该模块核心参考代码如下所示：

```
1 //例化arp接收模块：
2 arp_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_arp_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    .arp_rx_done   ( arp_rx_done   ),//ARP接收完成信号；
13    .arp_rx_type   ( arp_rx_type   ),//ARP接收类型 0:请求 1:应答；
14    .src_mac       ( src_mac       ),//接收到的源MAC地址；
15    .src_ip        ( src_ip        ),//接收到的源IP地址；
```

```

16     .crc_data      ( rx_crc_data  ),//需要CRC模块校验的数据;
17     .crc_en       ( rx_crc_en    ),//CRC开始校验使能;
18     .crc_clr      ( rx_crc_clr   ) //CRC数据复位信号;
19 );
20
21 //例化接收数据时需要的CRC校验模块:
22 crc32_d8  u_crc32_d8_rx (
23     .clk          ( gmii_rx_clk   ),//时钟信号;
24     .rst_n        ( rst_n         ),//复位信号, 低电平有效;
25     .data         ( rx_crc_data   ),//需要CRC模块校验的数据;
26     .crc_en       ( rx_crc_en     ),//CRC开始校验使能;
27     .crc_clr      ( rx_crc_clr    ),//CRC数据复位信号;
28     .crc_out      ( rx_crc_out    ) //CRC校验模块输出的数据;
29 );
30
31 //例化CRC发送模块:
32 arp_tx #(
33     .BOARD_MAC     ( BOARD_MAC    ),//开发板MAC地址 00-11-22-33-44-55;
34     .BOARD_IP      ( BOARD_IP     ),//开发板IP地址 192.168.1.10;
35     .DES_MAC       ( DES_MAC      ),//目的MAC地址 ff_ff_ff_ff_ff_ff;
36     .DES_IP        ( DES_IP       ),//目的IP地址 192.168.1.102;
37     .ETH_TYPE      ( ETH_TYPE     ),//以太网帧类型, 16'h0806表示ARP协议, 16'h0800表示IP协议;
38     .HD_TYPE       ( HD_TYPE      ),//硬件类型 以太网;
39     .PROTOCOL_TYPE ( PROTOCOL_TYPE),//上层协议为IP协议;
40     .HD_LEN        ( HD_LEN       ),//硬件地址长度。
41     .PROTOCOL_LEN  ( PROTOCOL_LEN ),//协议地址长度。
42     .OPCODE        ( OPCODE       ) //操作码, 1表示请求, 2表示应答。
43 )
44 u_arp_tx (
45     .clk          ( gmii_tx_clk   ),//时钟信号;
46     .rst_n        ( rst_n         ),//复位信号, 低电平有效;
47     .arp_tx_en    ( arp_tx_en     ),//ARP发送使能信号;
48     .arp_tx_type  ( arp_tx_type   ),//ARP发送类型 0:请求 1:应答;
49     .des_mac      ( des_mac       ),//发送的目标MAC地址;
50     .des_ip       ( des_ip        ),//发送的目标IP地址;
51     .crc_out      ( tx_crc_out    ),//CRC校验数据;
52     .gmii_tx_en   ( gmii_tx_en    ),//GMII输出数据有效信号;
53     .gmii_txd     ( gmii_txd     ),//GMII输出数据;
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

```

```

57     .crc_data      ( tx_crc_data  ),//输出给CRC校验模块进行计算的数据;
58     .rdy          ( arp_tx_rdy   ) //模块忙闲指示信号，高电平表示该模块处于空闲状态;
59 );
60
61 //例化发送数据时需要的CRC校验模块;
62 crc32_d8  u_crc32_d8_tx (
63     .clk          ( gmii_tx_clk   ),//时钟信号;
64     .rst_n        ( rst_n         ),//复位信号，低电平有效;
65     .data         ( tx_crc_data   ),//需要CRC模块校验的数据;
66     .crc_en       ( tx_crc_en     ),//CRC开始校验使能;
67     .crc_clr      ( tx_crc_clr    ),//CRC数据复位信号;
68     .crc_out      ( tx_crc_out    ) //CRC校验模块输出的数据;
69 );

```



arp顶层模块对应的TestBench代码如下所示:

```

1  `timescale 1 ns/1 ns
2  module test();
3      localparam    CYCLE      = 8                      ;//系统时钟周期，单位ns，默认10ns;
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    BOARD_MAC  = 48'h00_11_22_33_44_55   ;
6      localparam    BOARD_IP   = {8'd192,8'd168,8'd1,8'd10} ;
7      localparam    DES_MAC    = 48'h23_45_67_89_0a_bc   ;
8      localparam    DES_IP     = {8'd192,8'd168,8'd1,8'd23} ;
9      localparam    ETH_TPYE   = 16'h0806                ;//以太网帧类型 ARP
10
11     reg            clk        ;//系统时钟，默认100MHz;
12     reg            rst_n      ;//系统复位，默认低电平有效;
13     reg            gmii_rx_dv ;
14     reg            [7 : 0]    gmii_rxd ;
15     reg            arp_tx_en  ;
16     reg            arp_tx_type ;
17     reg            [47 : 0]    des_mac ;
18     reg            [31 : 0]    des_ip  ;
19
20     wire            gmii_tx_en ;

```

```

21 wire          [7 : 0]          gmii_txd          ;
22 wire          arp_rx_done      ;
23 wire          arp_rx_type      ;
24 wire          [47 : 0]         src_mac            ;
25 wire          [31 : 0]         src_ip             ;
26 wire          arp_tx_rdy       ;
27
28 always@(*)begin
29     gmii_rxd = gmii_txd;
30     gmii_rx_dv = gmii_tx_en;
31 end
32
33 //例化ARP模块;
34 arp #(
35     .BOARD_MAC      ( BOARD_MAC      ),
36     .BOARD_IP       ( BOARD_IP       ),
37     .DES_MAC        ( DES_MAC        ),
38     .DES_IP         ( DES_IP         )
39 )
40 u_arp (
41     .rst_n          ( rst_n          ),//系统复位，默认低电平有效;
42     .gmii_rx_clk    ( clk            ),//系统时钟，默认125MHz;
43     .gmii_rx_dv     ( gmii_rx_dv     ),
44     .gmii_rxd       ( gmii_rxd       ),
45     .gmii_tx_clk    ( clk            ),//系统时钟，默认125MHz;
46     .arp_tx_en      ( arp_tx_en      ),
47     .arp_tx_type    ( arp_tx_type    ),
48     .des_mac        ( des_mac        ),
49     .des_ip         ( des_ip         ),
50     .gmii_tx_en     ( gmii_tx_en     ),
51     .gmii_txd       ( gmii_txd       ),
52     .arp_rx_done    ( arp_rx_done    ),
53     .arp_rx_type    ( arp_rx_type    ),
54     .src_mac        ( src_mac        ),
55     .src_ip         ( src_ip         ),
56     .arp_tx_rdy     ( arp_tx_rdy     )
57 );
58
59
60 //生成周期为CYCLE数值的系统时钟;
61

```

```

62     initial begin
63         clk = 0;
64         forever #(CYCLE/2) clk = ~clk;
65     end
66
67     //生成复位信号;
68     initial begin
69         #1;arp_tx_en = 0;arp_tx_type = 0;des_mac = 0; des_ip = 0;
70         rst_n = 1;
71         #2;
72         rst_n = 0;//开始时复位10个时钟;
73         #(RST_TIME*CYCLE);
74         rst_n = 1;
75         #(20*CYCLE);
76         des_mac = BOARD_MAC;
77         des_ip = BOARD_IP;
78         arp_tx_en = 1'b1;
79         arp_tx_type = 1'b0;
80         #(CYCLE);
81         arp_tx_en = 1'b0;
82         @(posedge arp_tx_rdy);
83         #(20*CYCLE);
84         des_mac = src_mac;
85         des_ip = src_ip;
86         arp_tx_en = 1'b1;
87         arp_tx_type = 1'b1;
88         #(CYCLE);
89         arp_tx_en = 1'b0;
90         #(10*CYCLE);
91         @(posedge arp_tx_rdy);
92         #(20*CYCLE);
93         $stop;//停止仿真;
94     end
95
96 endmodule

```





## 2.1、CRC校验模块

CRC校验如果要从原理开始讲解，占用的篇幅会很大，所以本文先不讲解其原理，直接使用工具生成CRC校验的代码，对生成的代码进行修改，得到本文使用的模块即可，具体原理之后用一篇文章对其进行详细讲解。

能够生成CRC校验码的工具很多，我使用的网络链接[Generator for CRC HDL code \(bues.ch\)](http://bues.ch)，设置方式如下图示。

Online generator for CRC HDL code

This code generator creates HDL code (VHDL, Verilog or MyHDL) for any CRC algorithm. The HDL code is synthesizable and combinatorial. That means the calculation runs in one clock cycle on an FPGA.

Please select the CRC parameters and the output language settings below. Then press "generate" to generate the code.

Select CRC algorithm:

Standard algorithm

CRC-32

Use custom CRC parameters:

Bits: 32

Polynomial:  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

☒ Little endian / CRC shift direction to the right

Properties:

Input data word width (bits): 8

Function/module name: crc

Data parameter name: data

CRC input parameter name: crcln

CRC output parameter name: crcOut

Select output language:

Verilog function

Verilog module

VHDL module

MyHDL block

Python (for testing)

C (for testing)

Generate CRC code

The online generator is restricted to a maximum of 128 bit CRC length and a maximum of 512 bit input word length. The downloadable offline version (see below) does not have these restrictions.

图3 生成CRC代码

生成的代码如下所示：

```
1 `ifndef CRC_V_
2 `define CRC_V_
3
4 // CRC polynomial coefficients:  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ 
5 //                                0xEDB88320 (hex)
6 // CRC width:                    32 bits
```

```

7 // CRC shift direction:      right (little endian)
8 // Input word width:        8 bits
9
10 module crc (
11     input [31:0] crcIn,
12     input [7:0] data,
13     output [31:0] crcOut
14 );
15     assign crcOut[0] = crcIn[2] ^ crcIn[8] ^ data[2];
16     assign crcOut[1] = crcIn[0] ^ crcIn[3] ^ crcIn[9] ^ data[0] ^ data[3];
17     assign crcOut[2] = crcIn[0] ^ crcIn[1] ^ crcIn[4] ^ crcIn[10] ^ data[0] ^ data[1] ^ data[4];
18     assign crcOut[3] = crcIn[1] ^ crcIn[2] ^ crcIn[5] ^ crcIn[11] ^ data[1] ^ data[2] ^ data[5];
19     assign crcOut[4] = crcIn[0] ^ crcIn[2] ^ crcIn[3] ^ crcIn[6] ^ crcIn[12] ^ data[0] ^ data[2] ^ data[3] ^ data[6];
20     assign crcOut[5] = crcIn[1] ^ crcIn[3] ^ crcIn[4] ^ crcIn[7] ^ crcIn[13] ^ data[1] ^ data[3] ^ data[4] ^ data[7];
21     assign crcOut[6] = crcIn[4] ^ crcIn[5] ^ crcIn[14] ^ data[4] ^ data[5];
22     assign crcOut[7] = crcIn[0] ^ crcIn[5] ^ crcIn[6] ^ crcIn[15] ^ data[0] ^ data[5] ^ data[6];
23     assign crcOut[8] = crcIn[1] ^ crcIn[6] ^ crcIn[7] ^ crcIn[16] ^ data[1] ^ data[6] ^ data[7];
24     assign crcOut[9] = crcIn[7] ^ crcIn[17] ^ data[7];
25     assign crcOut[10] = crcIn[2] ^ crcIn[18] ^ data[2];
26     assign crcOut[11] = crcIn[3] ^ crcIn[19] ^ data[3];
27     assign crcOut[12] = crcIn[0] ^ crcIn[4] ^ crcIn[20] ^ data[0] ^ data[4];
28     assign crcOut[13] = crcIn[0] ^ crcIn[1] ^ crcIn[5] ^ crcIn[21] ^ data[0] ^ data[1] ^ data[5];
29     assign crcOut[14] = crcIn[1] ^ crcIn[2] ^ crcIn[6] ^ crcIn[22] ^ data[1] ^ data[2] ^ data[6];
30     assign crcOut[15] = crcIn[2] ^ crcIn[3] ^ crcIn[7] ^ crcIn[23] ^ data[2] ^ data[3] ^ data[7];
31     assign crcOut[16] = crcIn[0] ^ crcIn[2] ^ crcIn[3] ^ crcIn[4] ^ crcIn[24] ^ data[0] ^ data[2] ^ data[3] ^ data[4];
32     assign crcOut[17] = crcIn[0] ^ crcIn[1] ^ crcIn[3] ^ crcIn[4] ^ crcIn[5] ^ crcIn[25] ^ data[0] ^ data[1] ^ data[3] ^ data[4] ^ data[5];
33     assign crcOut[18] = crcIn[0] ^ crcIn[1] ^ crcIn[2] ^ crcIn[4] ^ crcIn[5] ^ crcIn[6] ^ crcIn[26] ^ data[0] ^ data[1] ^ data[2] ^ data[4] ^
34     assign crcOut[19] = crcIn[1] ^ crcIn[2] ^ crcIn[3] ^ crcIn[5] ^ crcIn[6] ^ crcIn[7] ^ crcIn[27] ^ data[1] ^ data[2] ^ data[3] ^ data[5] ^
35     assign crcOut[20] = crcIn[3] ^ crcIn[4] ^ crcIn[6] ^ crcIn[7] ^ crcIn[28] ^ data[3] ^ data[4] ^ data[6] ^ data[7];
36     assign crcOut[21] = crcIn[2] ^ crcIn[4] ^ crcIn[5] ^ crcIn[7] ^ crcIn[29] ^ data[2] ^ data[4] ^ data[5] ^ data[7];
37     assign crcOut[22] = crcIn[2] ^ crcIn[3] ^ crcIn[5] ^ crcIn[6] ^ crcIn[30] ^ data[2] ^ data[3] ^ data[5] ^ data[6];
38     assign crcOut[23] = crcIn[3] ^ crcIn[4] ^ crcIn[6] ^ crcIn[7] ^ crcIn[31] ^ data[3] ^ data[4] ^ data[6] ^ data[7];
39     assign crcOut[24] = crcIn[0] ^ crcIn[2] ^ crcIn[4] ^ crcIn[5] ^ crcIn[7] ^ data[0] ^ data[2] ^ data[4] ^ data[5] ^ data[7];
40     assign crcOut[25] = crcIn[0] ^ crcIn[1] ^ crcIn[2] ^ crcIn[3] ^ crcIn[5] ^ crcIn[6] ^ data[0] ^ data[1] ^ data[2] ^ data[3] ^ data[5] ^ data[7];
41     assign crcOut[26] = crcIn[0] ^ crcIn[1] ^ crcIn[2] ^ crcIn[3] ^ crcIn[4] ^ crcIn[6] ^ crcIn[7] ^ data[0] ^ data[1] ^ data[2] ^ data[3] ^ data[5] ^ data[7];
42     assign crcOut[27] = crcIn[1] ^ crcIn[3] ^ crcIn[4] ^ crcIn[5] ^ crcIn[7] ^ data[1] ^ data[3] ^ data[4] ^ data[5] ^ data[7];
43     assign crcOut[28] = crcIn[0] ^ crcIn[4] ^ crcIn[5] ^ crcIn[6] ^ data[0] ^ data[4] ^ data[5] ^ data[6];
44     assign crcOut[29] = crcIn[0] ^ crcIn[1] ^ crcIn[5] ^ crcIn[6] ^ crcIn[7] ^ data[0] ^ data[1] ^ data[5] ^ data[6] ^ data[7];
45     assign crcOut[30] = crcIn[0] ^ crcIn[1] ^ crcIn[6] ^ crcIn[7] ^ data[0] ^ data[1] ^ data[6] ^ data[7];
46     assign crcOut[31] = crcIn[1] ^ crcIn[7] ^ data[1] ^ data[7];
47

```

```

48 | endmodule
49 |
    `endif // CRC_V_

```



上述代码不能直接使用，需要稍微修改，代码中的crcIn[31:0]其实就是crcOut[0:31]打一拍的结果，最终crcOut[31:0]输出也不能直接使用，~crcOut[0:31]才是真正的CRC校验码，为了方便使用，下面是修改后的CRC校验模块，每次输入8个数据，32位CRC校验码在下个时钟周期输出，输出的CRC校验码可以直接使用，不需要做任何处理，那么这个模块的实用性就比较高了。

```

1  module crc32_d8(
2      input                clk          ,//时钟信号
3      input                rst_n        ,//复位信号，低电平有效
4      input [7:0]          data         ,//输入待校验8位数据
5      input                crc_en       ,//crc使能，开始校验标志
6      input                crc_clr      ,//crc数据复位信号
7      output [31:0]        crc_out      //CRC校验数据
8  );
9      reg [31 : 0]          crc_data    ;
10     //CRC32的生成多项式为: G(x)= x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1
11     always@(posedge clk)begin
12         if(!rst_n)
13             crc_data <= 32'hff_ff_ff_ff;
14         else if(crc_clr)//CRC校验值复位
15             crc_data <= 32'hff_ff_ff_ff;
16         else if(crc_en)begin
17             crc_data[0] <= crc_data[24] ^ crc_data[30] ^ data[7] ^ data[1];
18             crc_data[1] <= crc_data[24] ^ crc_data[25] ^ crc_data[30] ^ crc_data[31] ^ data[7] ^ data[6] ^ data[1] ^ data[0];
19             crc_data[2] <= crc_data[24] ^ crc_data[25] ^ crc_data[26] ^ crc_data[30] ^ crc_data[31] ^ data[7] ^ data[6] ^ data[5] ^ data[1] ^
20             crc_data[3] <= crc_data[25] ^ crc_data[26] ^ crc_data[27] ^ crc_data[31] ^ data[6] ^ data[5] ^ data[4] ^ data[0];
21             crc_data[4] <= crc_data[24] ^ crc_data[26] ^ crc_data[27] ^ crc_data[28] ^ crc_data[30] ^ data[7] ^ data[5] ^ data[4] ^ data[3] ^
22             crc_data[5] <= crc_data[24] ^ crc_data[25] ^ crc_data[27] ^ crc_data[28] ^ crc_data[29] ^ crc_data[30] ^ crc_data[31] ^ data[7] ^
23             crc_data[6] <= crc_data[25] ^ crc_data[26] ^ crc_data[28] ^ crc_data[29] ^ crc_data[30] ^ crc_data[31] ^ data[6] ^ data[5] ^ data[4] ^ data[3] ^
24             crc_data[7] <= crc_data[24] ^ crc_data[26] ^ crc_data[27] ^ crc_data[29] ^ crc_data[31] ^ data[7] ^ data[5] ^ data[4] ^ data[2] ^
25             crc_data[8] <= crc_data[0] ^ crc_data[24] ^ crc_data[25] ^ crc_data[27] ^ crc_data[28] ^ data[7] ^ data[6] ^ data[4] ^ data[3];
26             crc_data[9] <= crc_data[1] ^ crc_data[25] ^ crc_data[26] ^ crc_data[28] ^ crc_data[29] ^ data[6] ^ data[5] ^ data[3] ^ data[2];
27             crc_data[10] <= crc_data[2] ^ crc_data[24] ^ crc_data[26] ^ crc_data[27] ^ crc_data[29] ^ data[7] ^ data[5] ^ data[4] ^ data[2];
28

```

```

29     crc_data[11] <= crc_data[3] ^ crc_data[24] ^ crc_data[25] ^ crc_data[27] ^ crc_data[28] ^ data[7] ^ data[6] ^ data[4] ^ data[3];
30     crc_data[12] <= crc_data[4] ^ crc_data[24] ^ crc_data[25] ^ crc_data[26] ^ crc_data[28] ^ crc_data[29] ^ crc_data[30] ^ data[7] ^
31     crc_data[13] <= crc_data[5] ^ crc_data[25] ^ crc_data[26] ^ crc_data[27] ^ crc_data[29] ^ crc_data[30] ^ crc_data[31] ^ data[6] ^
32     crc_data[14] <= crc_data[6] ^ crc_data[26] ^ crc_data[27] ^ crc_data[28] ^ crc_data[30] ^ crc_data[31] ^ data[5] ^ data[3] ^ data[0];
33     crc_data[15] <= crc_data[7] ^ crc_data[27] ^ crc_data[28] ^ crc_data[29] ^ crc_data[31] ^ data[3] ^ data[4] ^ data[2] ^ data[0];
34     crc_data[16] <= crc_data[8] ^ crc_data[24] ^ crc_data[28] ^ crc_data[29] ^ data[7] ^ data[3] ^ data[2];
35     crc_data[17] <= crc_data[9] ^ crc_data[25] ^ crc_data[29] ^ crc_data[30] ^ data[6] ^ data[2] ^ data[1];
36     crc_data[18] <= crc_data[10] ^ crc_data[26] ^ crc_data[30] ^ crc_data[31] ^ data[5] ^ data[1] ^ data[0];
37     crc_data[19] <= crc_data[11] ^ crc_data[27] ^ crc_data[31] ^ data[4] ^ data[0];
38     crc_data[20] <= crc_data[12] ^ crc_data[28] ^ data[3];
39     crc_data[21] <= crc_data[13] ^ crc_data[29] ^ data[2];
40     crc_data[22] <= crc_data[14] ^ crc_data[24] ^ data[7];
41     crc_data[23] <= crc_data[15] ^ crc_data[24] ^ crc_data[25] ^ crc_data[30] ^ data[7] ^ data[6] ^ data[1];
42     crc_data[24] <= crc_data[16] ^ crc_data[25] ^ crc_data[26] ^ crc_data[31] ^ data[6] ^ data[5] ^ data[0];
43     crc_data[25] <= crc_data[17] ^ crc_data[26] ^ crc_data[27] ^ data[5] ^ data[4];
44     crc_data[26] <= crc_data[18] ^ crc_data[24] ^ crc_data[27] ^ crc_data[28] ^ crc_data[30] ^ data[7] ^ data[3] ^ data[4] ^ data[1];
45     crc_data[27] <= crc_data[19] ^ crc_data[25] ^ crc_data[28] ^ crc_data[29] ^ crc_data[31] ^ data[6] ^ data[3] ^ data[2] ^ data[0];
46     crc_data[28] <= crc_data[20] ^ crc_data[26] ^ crc_data[29] ^ crc_data[30] ^ data[5] ^ data[2] ^ data[1];
47     crc_data[29] <= crc_data[21] ^ crc_data[27] ^ crc_data[30] ^ crc_data[31] ^ data[4] ^ data[1] ^ data[0];
48     crc_data[30] <= crc_data[22] ^ crc_data[28] ^ crc_data[31] ^ data[3] ^ data[0];
49     crc_data[31] <= crc_data[23] ^ crc_data[29] ^ data[2];
50     end
51 end
52
53 //将计算的数据各位取反倒序赋值后输出。
54 assign crc_out[31:0] = ~{crc_data[0],crc_data[1],crc_data[2],crc_data[3],crc_data[4],crc_data[5],crc_data[6],crc_data[7],
55     crc_data[8],crc_data[9],crc_data[10],crc_data[11],crc_data[12],crc_data[13],crc_data[14],crc_data[15],
56     crc_data[16],crc_data[17],crc_data[18],crc_data[19],crc_data[20],crc_data[21],crc_data[22],crc_data[23],
57     crc_data[24],crc_data[25],crc_data[26],crc_data[27],crc_data[28],crc_data[29],crc_data[30],crc_data[31]};
endmodule

```



此处不对CRC校验模块进行仿真，在后文ARP接收和发送模块中一起仿真，并且验证该模块功能是否正确。

## 2.2、ARP接收模块

前文对 以太网帧格式 和ARP协议的帧格式做了详细讲解，如图4所示，首先包括7个字节的前导码8'h55，然后帧起始符8'hd5，之后就是14字节的以太网帧头，后跟28字节的ARP数据，18字节的数据0，最后4字节的CRC校验码。

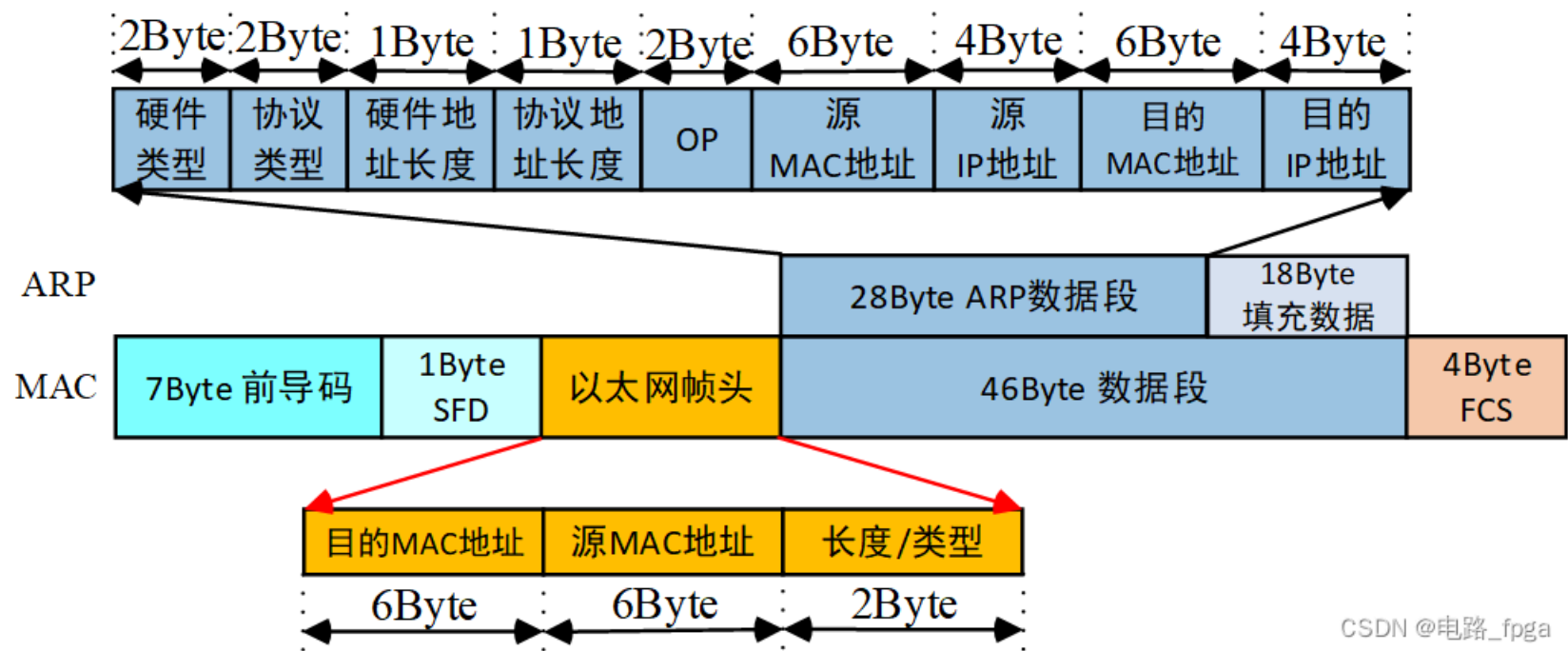
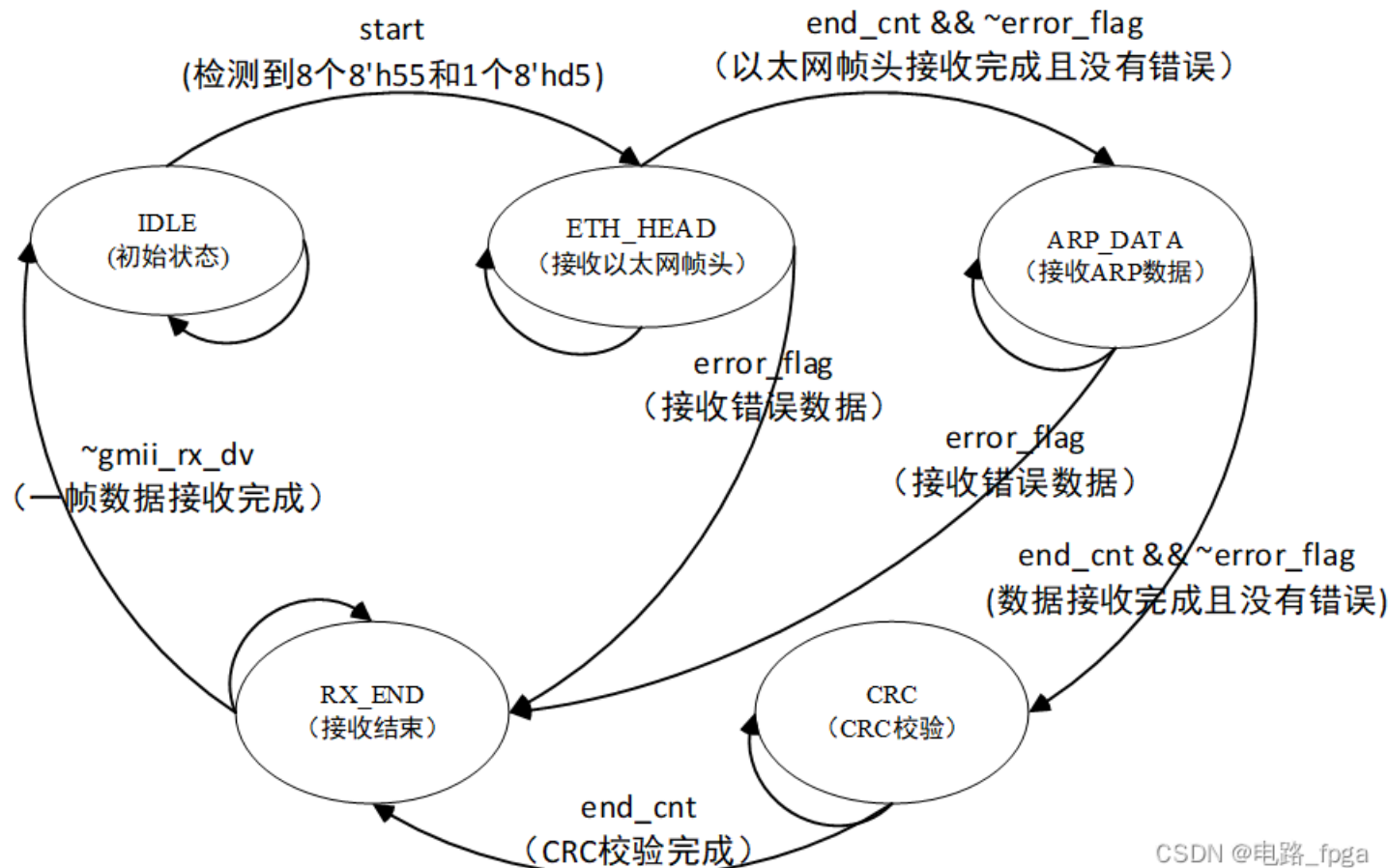


图4 以太网ARP协议数据报

本模块以状态机为主结构，内部通过一个计数器cnt的计数值完成状态之间的跳转，状态转换图如下所示。



CSDN @电路\_fpga

图5 状态转换图

本设计使用了一个7个字节的移位寄存器，把接收的gmii\_rxd信号暂存，后续设计也可以使用移位寄存器中暂存的数据。状态机处于空闲状态时，会去检测移位寄存器中的数据是否全为8'h55且gmii\_rxd为8'hd5，且gmii\_rx\_dv均为高电平，则检测到前导码和帧起始符，那么状态机跳转到接收以太网帧头状态。

在以太网帧头状态，接收目的MAC地址，如果接收到的目的MAC地址不是开发板目的MAC地址或者广播地址，则此数据报不是发给开发板的，状态机直接跳转到RX\_END状态，丢弃该数据报。如果接收到的目的MAC地址满足要求则继续接收，需要检测以太网帧头的类型字节是否为ARP对应的16'h0806，如果是则跳转到ARP\_DATA状态接收ARP数据段，如果不是则跳转到RX\_END，丢弃该数据报。

在接收ARP数据状态时，需要把OP码保存，将源MAC地址和源IP地址保存，如果目的IP地址不是开发板IP地址，则将数据报丢弃，否则继续接收数据，当接收完ARP数据和18字节的填充数据后，跳转到接收CRC校验码状态。

状态机处于接收以太网帧头和接收ARP数据状态时，将CRC模块的使能信号拉高，且把移位寄存器第一个字节输出给CRC校验模块，进行CRC校验。当状态机跳转到接收CRC校验状态时，CRC校验模块的使能信号拉低。当状态机回到空闲状态时，CRC校验模块的清零信号拉高。

接收完PC端发出的CRC校验码之后，与CRC校验模块输出的CRC校验码对比，如果相同，则将接收到的源MAC地址，源IP地址输出，如果OP码等于1，则将ARP请求应答指示信号输出低电平，如果OP码为2，则ARP请求应答指示信号输出高电平。

状态机处于RX\_END状态时，只有检测到gmii\_rx\_dv为低电平时，才会回到空闲状态，继续检测下一帧数据，这是为了防止该帧数据报中可能出现与帧头和前导码一样的数据，从而解析错误。

需要注意计数器cnt在状态机不处于空闲状态和RX\_END状态时就会对gmii\_rxc时钟计数，状态机回到空闲状态时需要清零。计数器在状态机每个状态的最大值不一样，状态机在接收以太网帧头需要接收14字节数据，那么计数器最大值为14-1，在接收ARP数据状态，因为需要接收46字节数据，故计数器最大值为46-1。

状态机与移位寄存器gmii\_rxd\_r[0]的数据对齐，所以后文对数据的截取大多截取的gmii\_rxd\_r[0]信号。

大概含义就这么多，其余细节查看代码即可，参考代码如下所示：

```
1 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register :
2 always@(posedge clk)begin
3     if(!rst_n)begin
4         state_c <= IDLE;
5     end
6     else begin
7         state_c <= state_n;
8     end
9 end
10
11 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
12 always@(*)begin
13     case(state_c)
14         IDLE:begin
15             if(start)begin//检测到前导码和SFD后跳转到接收以太网帧头数据的状态。
16                 state_n = ETH_HEAD;
17             end
18             else begin
19                 state_n = state_c;
20             end
21         end
22         ETH_HEAD:begin
23             if(error_flag)begin//在接收帧头数据时，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ARP协议数据。
27             state_n = ARP_DATA;
28         end
29         else begin
30             state_n = state_c;
31         end
32     end
33     ARP_DATA:begin
34         if(error_flag)begin//在接收ARP协议过程中检测到错误。
35             state_n = RX_END;
36         end
37         else if(end_cnt)begin//接收完ARP协议数据且未检测到数据错误。
38             state_n = CRC;
39         end
40         else begin
41             state_n = state_c;
42         end
43     end
44     CRC:begin
45         if(end_cnt)begin//接收完CRC校验数据。
46             state_n = RX_END;
47         end
48         else begin
49             state_n = state_c;
50         end
51     end
52     RX_END:begin
53         if(~gmii_rx_dv)begin//检测到数据线上数据无效。
54             state_n = IDLE;
55         end
56         else begin
57             state_n = state_c;
58         end
59     end
60     default:begin
61         state_n = IDLE;
62     end
63 endcase
64 end
65

```



```

66
67 //将输入数据保存6个时钟周期，用于检测前导码和SFD。
68 //注意后文的state_c与gmii_rxd_r[0]对齐。
69 always@(posedge clk)begin
70     gmii_rxd_r[6] <= gmii_rxd_r[5];
71     gmii_rxd_r[5] <= gmii_rxd_r[4];
72     gmii_rxd_r[4] <= gmii_rxd_r[3];
73     gmii_rxd_r[3] <= gmii_rxd_r[2];
74     gmii_rxd_r[2] <= gmii_rxd_r[1];
75     gmii_rxd_r[1] <= gmii_rxd_r[0];
76     gmii_rxd_r[0] <= gmii_rxd;
77     gmii_rx_dv_r <= {gmii_rx_dv_r[5 : 0],gmii_rx_dv};
78 end
79
80 //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。
81 always@(posedge clk)begin
82     if(rst_n==1'b0)begin//初始值为0;
83         start <= 1'b0;
84     end
85     else if(state_c == IDLE)begin
86         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],
87     end
88 end
89
90 //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。
91 always@(posedge clk)begin
92     if(rst_n==1'b0)begin//
93         cnt <= 0;
94     end
95     else if(add_cnt)begin
96         if(end_cnt)
97             cnt <= 0;
98         else
99             cnt <= cnt + 1;
100     end
101     else begin
102         cnt <= 0;
103     end
104 end
105 //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。
106

```

```

107 assign add_cnt = (state_c != IDLE) && (state_c != RX_END);
108 assign end_cnt = add_cnt && cnt == cnt_num - 1;
109
110 //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。
111 //在接收ARP数据时，需要接收46byte数据，在CRC阶段需要接收4字节CRC校验码。
112 always@(posedge clk)begin
113     if(rst_n==1'b0)begin//初始值为46;
114         cnt_num <= 6'd46;
115     end
116     else begin
117         case(state_c)
118             ETH_HEAD : cnt_num <= 6'd14;
119             CRC       : cnt_num <= 6'd4;
120             default: cnt_num <= 6'd46;
121         endcase
122     end
123 end
124
125 //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。
126 always@(posedge clk)begin
127     if(rst_n==1'b0)begin//初始值为0;
128         des_mac_t <= 48'd0;
129     end
130     else if((state_c == ETH_HEAD) && add_cnt && cnt < 5'd6)begin
131         des_mac_t <= {des_mac_t[39:0],gmii_rxd_r[0]};
132     end
133 end
134
135 //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。
136 always@(posedge clk)begin
137     if(rst_n==1'b0)begin//初始值为0;
138         error_flag <= 1'b0;
139     end
140     else begin
141         case(state_c)
142             ETH_HEAD : begin
143                 if(add_cnt)
144                     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。
145                         error_flag <= ((des_mac_t != BOARD_MAC) && (des_mac_t != 48'HFF_FF_FF_FF_FF));
146                     else if(cnt ==12)//判断接收的数据是不是ARP协议。
147

```

```

148         error_flag <= ({gmii_rxd_r[0],gmii_rxd} != ETH_TPYE);
149     end
150     ARP_DATA : begin
151         if(add_cnt && cnt == 28)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。
152             error_flag <= ((opcode != 16'd1) && (opcode != 16'd2)) || (des_ip_t != BOARD_IP);
153         end
154     end
155     default: error_flag <= 1'b0;
156 endcase
157 end
158 end
159
160 //接收OP操作码，源MAC地址，源IP地址，目的IP地址。
161 always@(posedge clk)begin
162     if(rst_n==1'b0)begin//初始值为0;
163         opcode <= 16'd0;
164         src_mac_t <= 48'd0;
165         src_ip_t <= 32'd0;
166         des_ip_t <= 32'd0;
167     end
168     else if(state_c == ARP_DATA && add_cnt)begin
169         case(cnt)
170             5'd6 : opcode[15:8] <= gmii_rxd_r[0];//操作码;
171             5'd7 : opcode[7:0] <= gmii_rxd_r[0];//操作码;
172             5'd8,5'd9,5'd10,5'd11,5'd12,5'd13 : src_mac_t <= {src_mac_t[39:0],gmii_rxd_r[0]};//源MAC地址;
173             5'd14,5'd15,5'd16,5'd17 : src_ip_t <= {src_ip_t[23:0],gmii_rxd_r[0]};//源IP地址;
174             5'd24,5'd25,5'd26,5'd27 : des_ip_t <= {des_ip_t[23:0],gmii_rxd_r[0]};//目标IP地址;
175             default: ;
176         endcase
177     end
178 end
179
180 //生产CRC校验相关的数据和控制信号。
181 always@(posedge clk)begin
182     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。
183     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。
184     crc_en <= (state_c == ETH_HEAD) || (state_c == ARP_DATA);//CRC校验使能信号。
185 end
186
187 //接收PC端发送来的CRC数据。
188

```

```

189 always@(posedge clk)begin
190     if(rst_n==1'b0)begin//初始值为0;
191         des_crc <= 32'hff_ff_ff_ff;
192     end
193     else if(add_cnt && state_c == CRC)begin//先接收的是低位数据;
194         des_crc <= {gmii_rxd_r[0],des_crc[23:8]};
195     end
196 end
197
198 //生成相应的输出数据。
199 always@(posedge clk)begin
200     if(rst_n==1'b0)begin//初始值为0;
201         arp_rx_type <= 1'b0;
202         arp_rx_done <= 1'b0;
203         src_mac <= 48'd0;
204         src_ip <= 32'd0;
205     end//如果CRC校验成功，把ARP协议接收完成信号拉高，把接收到的源MAC和IP地址输出，并且将ARP协议类型输出。
206     else if(state_c == CRC && end_cnt && ({gmii_rxd_r[0],des_crc[23:0]} == crc_out))begin
207         arp_rx_done <= 1'b1;
208         src_mac <= src_mac_t;
209         src_ip <= src_ip_t;
210         if(opcode == 16'd1)
211             arp_rx_type <= 1'b0;//ARP请求;
212         else
213             arp_rx_type <= 1'b1;//ARP应答;
214     end
215     else begin
216         arp_rx_done <= 1'b0;
217     end
218 end
219 end

```



对应的TestBench如下所示:

```

1 `timescale 1 ns/1 ns
2 module test();
3     parameter    CYCLE            =    8                ;//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 SOURCE_MAC   = 48'h23_45_67_89_0a_bc    ;
9   localparam SOURCE_IP    = {8'd192,8'd168,8'd1,8'd23} ;
10  localparam ETH_TPYPE    = 16'h0806                ;//以太网帧类型  ARP
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
17  wire          crc_clr_r   ;
18  wire          crc_en     ;
19  wire  [7 : 0]  crc_data   ;
20  wire  [31 : 0] crc_out_r   ;
21  wire  [31 : 0] src_ip     ;
22  wire  [47 : 0] src_mac    ;
23  wire          arp_rx_done ;
24  wire          arp_rx_type ;
25
26  arp_rx #(
27      .BOARD_MAC ( BOARD_MAC ),
28      .BOARD_IP  ( BOARD_IP )
29  )
30  u_arp_rx (
31      .clk      ( clk ),
32      .rst_n    ( rst_n ),
33      .gmii_rxd ( gmii_rxd ),
34      .gmii_rx_dv ( gmii_rx_dv ),
35      .crc_out  ( crc_out_r ),
36      .crc_en   ( crc_en ),
37      .crc_clr  ( crc_clr_r ),
38      .crc_data ( crc_data ),
39      .arp_rx_done( arp_rx_done ),
40      .arp_rx_type( arp_rx_type ),
41      .src_mac   ( src_mac ),
42      .src_ip    ( src_ip )
43  );
44

```

```

45 //例化CRC校验模块
46 crc32_d8 u_crc32_d8_2 (
47     .clk      ( clk      ),
48     .rst_n    ( rst_n    ),
49     .data     ( crc_data  ),
50     .crc_en   ( crc_en   ),
51     .crc_clr  ( crc_clr_r ),
52     .crc_out  ( crc_out_r )
53 );
54
55 reg          crc_clr      ;
56 reg          gmii_crc_vld ;
57 reg [7 : 0]   gmii_rxd_r  ;
58 reg          gmii_rx_dv_r ;
59 reg          crc_data_vld ;
60 wire [31 : 0] crc_out     ;
61
62 //生成周期为CYCLE数值的系统时钟;
63 initial begin
64     clk = 0;
65     forever #(CYCLE/2) clk = ~clk;
66 end
67
68 //生成复位信号;
69 initial begin
70     #1;gmii_rxd = 0; gmii_rx_dv = 0;gmii_crc_vld = 1'b0;
71     gmii_rxd_r=0;gmii_rx_dv_r=0;crc_clr=0;
72     rst_n = 1;
73     #2;
74     rst_n = 0;//开始时复位10个时钟;
75     #(RST_TIME*CYCLE);
76     rst_n = 1;
77     #(20*CYCLE);
78     repeat(4)begin//发送4帧ARP指令;
79         gmii_tx_test();
80         gmii_crc_vld = 1'b1;
81         gmii_rxd_r = crc_out[7 : 0];
82         #(CYCLE);
83         gmii_rxd_r = crc_out[15 : 8];
84         #(CYCLE);
85

```

```

86     gmii_rxd_r = crc_out[23 : 16];
87     #(CYCLE);
88     gmii_rxd_r = crc_out[31 : 24];
89     #(CYCLE);
90     gmii_crc_vld = 1'b0;
91     crc_clr = 1'b1;
92     #(CYCLE);
93     crc_clr = 1'b0;
94     #(20*CYCLE);
95 end
96
97 $stop;//停止仿真:
98 end
99
100 reg [5:0] i;
101 task gmii_tx_test;
102     begin
103         crc_data_vld = 1'b0;
104         #(CYCLE);
105         repeat(7)begin//发送前导码7个8'H55;
106             gmii_rxd_r = 8'h55;
107             gmii_rx_dv_r = 1'b1;
108             #(CYCLE);
109         end
110         gmii_rxd_r = 8'hd5;//发送SFD, 一个字节的8'hd5;
111         #(CYCLE);
112         crc_data_vld = 1'b1;
113         for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的MAC地址;
114             gmii_rxd_r = BOARD_MAC[47-8*i -: 8];
115             #(CYCLE);
116         end
117         for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的源MAC地址;
118             gmii_rxd_r = SOURCE_MAC[47-8*i -: 8];
119             #(CYCLE);
120         end
121         for(i=0 ; i<2 ; i=i+1)begin//发送2个字节的以太网类型;
122             gmii_rxd_r = ETH_TPYE[15-8*i -: 8];
123             #(CYCLE);
124         end
125         gmii_rxd_r = 8'd0;//发送2字节的硬件地址类型。
126

```

```

127     #(CYCLE);
128     gmii_rxd_r = 8'd1;
129     #(CYCLE);
130     gmii_rxd_r = 8'h08;//发送2字节的协议类型，0X0800表示上层IP协议。
131     #(CYCLE);
132     gmii_rxd_r = 8'h00;
133     #(CYCLE);
134     gmii_rxd_r = 8'h06;//发送1字节的硬件地址长度。
135     #(CYCLE);
136     gmii_rxd_r = 8'h04;//发送1字节的IP地址长度。
137     #(CYCLE);
138     gmii_rxd_r = 8'h00;//发送2字节的OP编码，1表示ARP请求，2表示ARP应答。
139     #(CYCLE);
140     gmii_rxd_r = 8'h02;
141     #(CYCLE);
142     for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的源MAC地址：
143         gmii_rxd_r = SOURCE_MAC[47-8*i -: 8];
144         #(CYCLE);
145     end
146     for(i=0 ; i<4 ; i=i+1)begin//发送4个字节的源IP地址：
147         gmii_rxd_r = SOURCE_IP[31-8*i -: 8];
148         #(CYCLE);
149     end
150     for(i=0 ; i<6 ; i=i+1)begin//发送6个字节的MAC地址：
151         gmii_rxd_r = BOARD_MAC[47-8*i -: 8];
152         #(CYCLE);
153     end
154     for(i=0 ; i<4 ; i=i+1)begin//发送4个字节的IP地址：
155         gmii_rxd_r = BOARD_IP[31-8*i -: 8];
156         #(CYCLE);
157     end
158     for(i=0 ; i<18 ; i=i+1)begin//补发18个字节的0；
159         gmii_rxd_r = 8'd0;
160         #(CYCLE);
161     end
162     crc_data_vld = 1'b0;
163     gmii_rx_dv_r = 1'b0;
164 end
165 endtask
166
167

```



```

167     crc32_d8  u_crc32_d8_1 (
168         .clk      ( clk      ),
169         .rst_n     ( rst_n    ),
170         .data      ( gmii_rxd_r ),
171         .crc_en     ( crc_data_vld ),
172         .crc_clr    ( crc_clr  ),
173         .crc_out    ( crc_out  )
174     );
175
176     always@(posedge clk)begin
177         if(rst_n==1'b0)begin//初始值为0;
178             gmii_rxd <= 8'd0;
179             gmii_rx_dv <= 1'b0;
180         end
181         else if(gmii_rx_dv_r || gmii_crc_vld)begin
182             gmii_rxd <= gmii_rxd_r;
183             gmii_rx_dv <= 1'b1;
184         end
185         else begin
186             gmii_rx_dv <= 1'b0;
187         end
188     end
189
190 endmodule

```



仿真截图如下所示，检测到前导码和帧起始符，start信号为高电平，状态机从空闲转台跳转到接收以太网帧头状态，且状态机与gmii\_rxd\_r[0]的数据对齐。

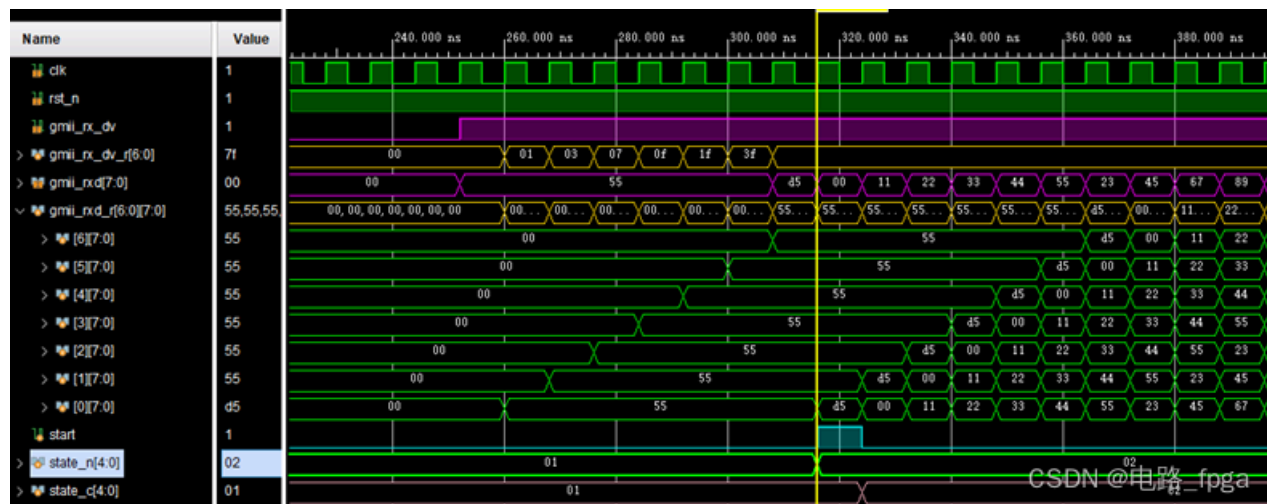


图6 检测到前导码和帧起始符

然后就是接收以太网帧头和ARP数据报了，如下所示，都比较简单，不做赘述，需要细节的可以打开工程自行仿真查看。

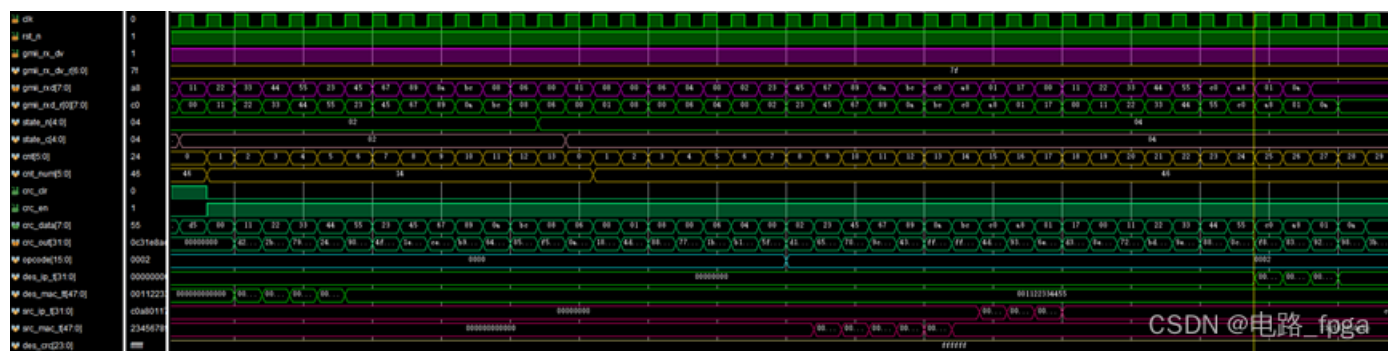


图7 接收以太网帧头和ARP数据

下图为接收CRC校验，天蓝色信号des\_crc是接收到的CRC校验码的低24位，红色信号crc\_out是该模块通过接收数据计算出的CRC校验码，在计数器等于3时，{gmii\_rxd\_r[0],crc\_out}==crc\_out则说明接收的数据正确，将接收完成信号arp\_rx\_done拉高一个时钟周期。

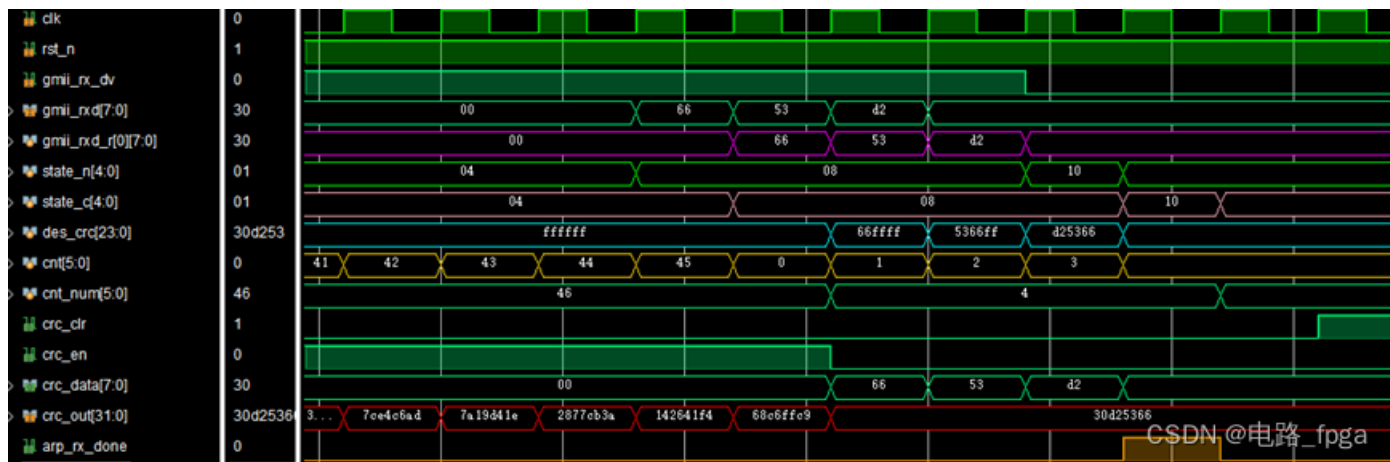


图8 接收CRC校验

注意32位CRC校验码先发低字节数据，后发高字节数据。

如何验证CRC计算正确呢？可以通过CRC计算器，计算出发送这些数据的CRC校验码是多少，与仿真结果对比，进而确定模块输出的CRC校验码是否正常。

如下图所示，将以太网帧头和ARP数据(包括18字节的填充数据)写入1处，选择CRC-32，点击3处进行计算，4就是计算结果，32'h30D25366，与图8中红色信号计算结果一致，证明CRC校验模块设计也没有问题。

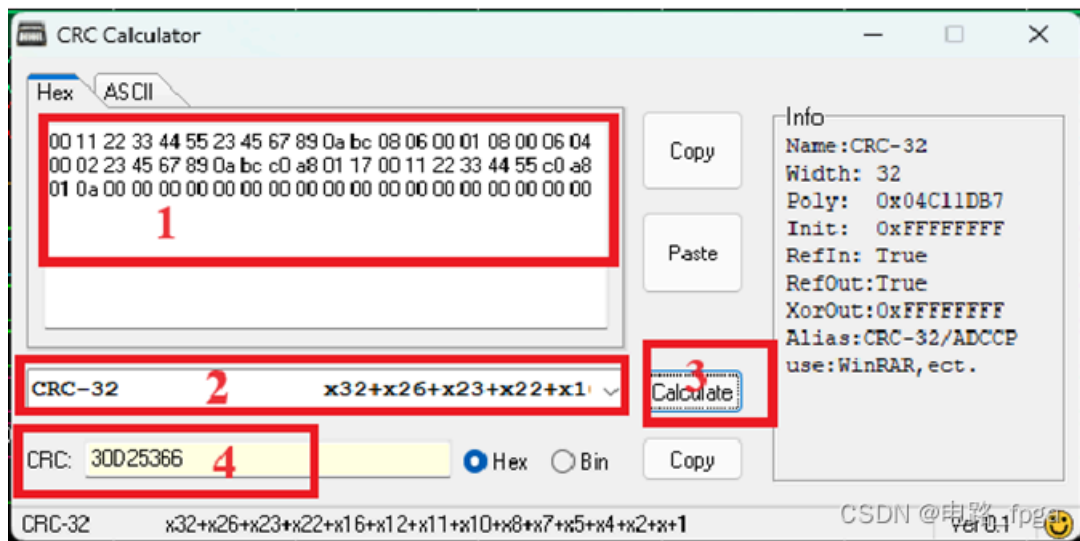


图9 CRC校验计算器

以上就是ARP接收模块的设计，对该模块进行了仿真，仿真也处于正常状态，后续上板可以通过ILA抓包查看上板时序。

## 2.3、ARP发送模块

ARP发送模块相比接收模块会更加简单，最简单的设计方法其实是通过一个计数器记录发送数据个数，然后根据计数器译码输出数据即可，但是这种方式不方便查看。所以本文还是通过一个状态机嵌套计数器的方式实现。

状态机处于空闲状态时，如果gmii\_tx\_start开始信号位高电平时，此时如果检测模块接收的目的MAC和目的IP地址是否为0，不为0就更新目的MAC地址和目的IP地址。开始产生数据，状态机跳转到发送前导码和帧起始符状态。

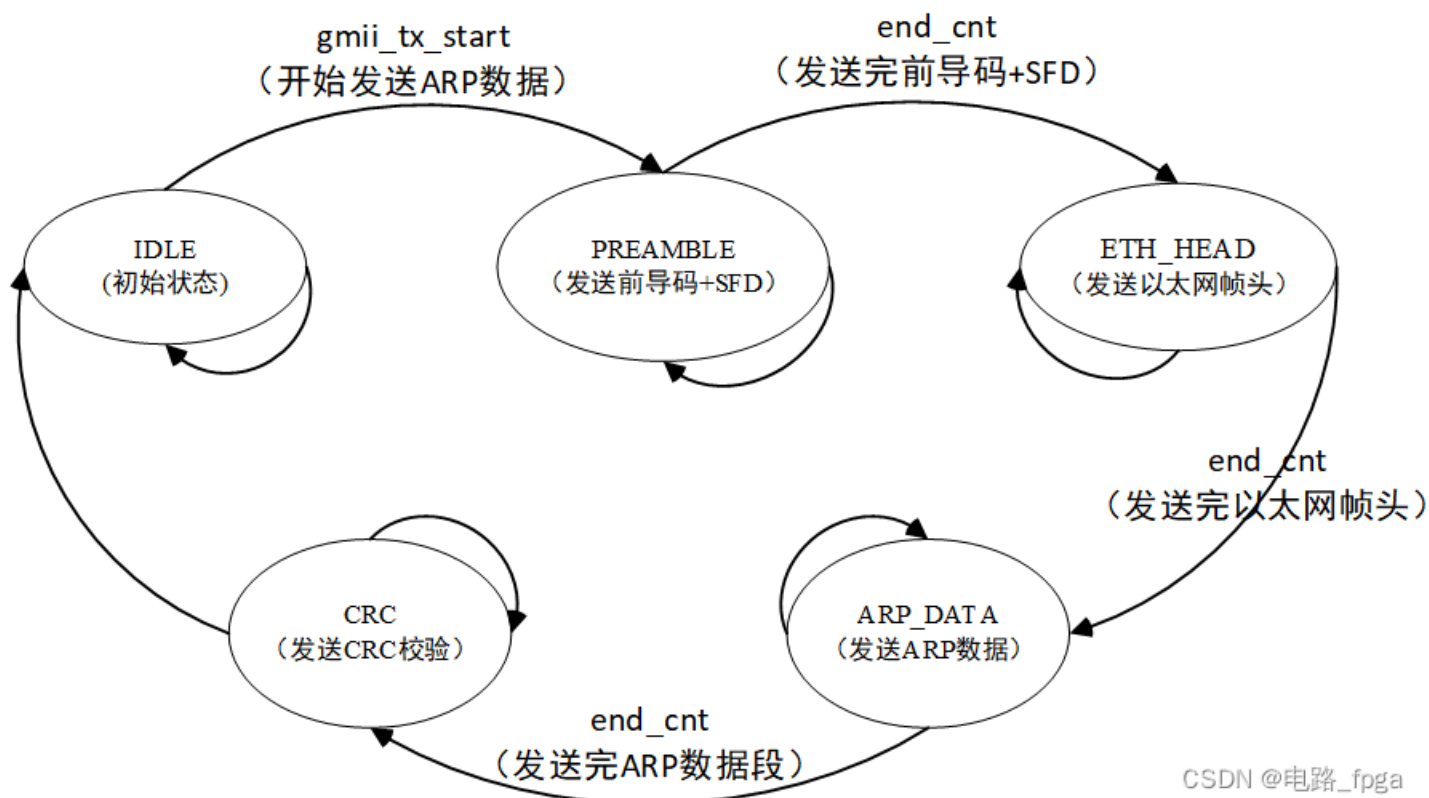


图10 ARP发送模块状态转换图

前导码和帧起始符发送完成后，状态机跳转到发送以太网帧头状态，然后发送ARP数据，最后发送CRC校验码，由于CRC校验码的计算会滞后输入信号一个时钟周期，所以将前面的数据打一拍在输出，就能刚好接上CRC校验码，数据发送完成后状态机回到初始状态。因此该模块其实比较简单。

对应的参考代码如下所示：

```

1  localparam    IDLE          = 5'b0_0001          ;//初始状态，等待开始发送信号；
2  localparam    PREAMBLE      = 5'b0_0010          ;//发送前导码+帧起始界定符；
3  localparam    ETH_HEAD      = 5'b0_0100          ;//发送以太网帧头；
4  localparam    ARP_DATA      = 5'b0_1000          ;//发送ARP协议数据；
5  localparam    CRC           = 5'b1_0000          ;//发送CRC校验值；
6  localparam    MIN_DATA_NUM  = 16'd46             ;//以太网数据最小为46个字节，不足部分填充数据0。
7
8  reg            gmii_tx_en_r          ;//
9  reg            [47 : 0] des_mac_r      ;//
10 reg            [31 : 0] des_ip_r       ;
11 reg            [1  : 0] arp_tx_type_r  ;
12 reg            [4  : 0] state_n        ;
13 reg            [4  : 0] state_c        ;
14 reg            [5  : 0] cnt            ;//
15 reg            [5  : 0] cnt_num        ;//
16
17 wire            add_cnt                ;
18 wire            end_cnt                ;
19
20 //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ARP的操作类型进行暂存。
21 always@(posedge clk)begin
22     if(rst_n==1'b0)begin//初始值为0；
23         des_ip_r <= DES_IP;
24         des_mac_r <= DES_MAC;
25         arp_tx_type_r <= OPCODE;
26     end
27     else if(arp_tx_en && state_c == IDLE)begin
28         arp_tx_type_r <= arp_tx_type ? 2'd2 : 2'd1;
29         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。
30             des_ip_r <= des_ip;
31             des_mac_r <= des_mac;
32         end
33     end
34 end
35
36 //The first section: synchronous timing always module, formatted to describe the transfer of the secondary register to the live register ;
37 always@(posedge clk)begin
38     if(!rst_n)begin
39         state_c <= IDLE;
40     end
41

```

```

41     else begin
42         state_c <= state_n;
43     end
44 end
45
46 //The second paragraph: The combinational logic always module describes the state transition condition judgment.
47 always@(*)begin
48     case(state_c)
49         IDLE:begin
50             if(arp_tx_en)begin//在空闲状态接收到上游发出的使能信号;
51                 state_n = PREAMBLE;
52             end
53             else begin
54                 state_n = state_c;
55             end
56         end
57         PREAMBLE:begin
58             if(end_cnt)begin//发送完前导码和SFD;
59                 state_n = ETH_HEAD;
60             end
61             else begin
62                 state_n = state_c;
63             end
64         end
65         ETH_HEAD:begin
66             if(end_cnt)begin//发送完以太网帧头数据;
67                 state_n = ARP_DATA;
68             end
69             else begin
70                 state_n = state_c;
71             end
72         end
73         ARP_DATA:begin
74             if(end_cnt)begin//发送完ARP协议数据;
75                 state_n = CRC;
76             end
77             else begin
78                 state_n = state_c;
79             end
80         end
81     end
82 end

```

```

82         CRC:begin
83             if(end_cnt)begin//发送完CRC校验码;
84                 state_n = IDLE;
85             end
86             else begin
87                 state_n = state_c;
88             end
89         end
90         default:begin
91             state_n = IDLE;
92         end
93     endcase
94 end
95
96 //计数器cnt，记录状态机每个状态持续的时钟个数。
97 always@(posedge clk)begin
98     if(rst_n==1'b0)begin//
99         cnt <= 0;
100     end
101     else if(add_cnt)begin
102         if(end_cnt)
103             cnt <= 0;
104         else
105             cnt <= cnt + 1;
106     end
107 end
108
109 assign add_cnt = (state_c != IDLE);//状态机不处于空闲状态时对时钟进行计数。
110 assign end_cnt = add_cnt && cnt == cnt_num - 1;
111
112 always@(posedge clk)begin
113     if(rst_n==1'b0)begin//初始值为0;
114         cnt_num <= 6'd46;
115     end
116     else begin
117         case (state_c)
118             PREAMBLE : cnt_num <= 6'd8;
119             ETH_HEAD : cnt_num <= 6'd14;
120             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。
121             default: cnt_num <= 6'd46;
122

```

```

123         endcase
124     end
125 end
126
127 //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。
128 always@(posedge clk)begin
129     if(rst_n==1'b0)begin//初始值为0;
130         crc_data <= 8'd0;
131     end
132     else if(add_cnt)begin
133         case (state_c)
134             PREAMBLE : if(end_cnt)
135                 crc_data <= 8'hd5;//发送1字节SFD编码;
136             else
137                 crc_data <= 8'h55;//发送7字节前导码;
138             ETH_HEAD : if(cnt < 6)
139                 crc_data <= des_mac_r[47 - 8*cnt -: 8];//发送目的MAC地址，先发高字节;
140             else if(cnt < 12)
141                 crc_data <= BOARD_MAC[47 - 8*(cnt-6) -: 8];//发送源MAC地址，先发高字节;
142             else
143                 crc_data <= ETH_TYPE[15 - 8*(cnt-12) -: 8];//发送源以太网协议类型，先发高字节;
144             ARP_DATA : begin
145                 case (cnt)
146                     6'd0 , 6'd1 : crc_data <= HD_TYPE[15 - 8*cnt -: 8];//发送硬件类型，先发高字节;
147                     6'd2 , 6'd3 : crc_data <= PROTOCOL_TYPE[15 - 8*(cnt-2) -: 8];//发送协议类型，先发高字节;
148                     6'd4       : crc_data <= HD_LEN;//发送硬件地址长度;
149                     6'd5       : crc_data <= PROTOCOL_LEN;//发送协议地址长度;
150                     6'd6       : crc_data <= 8'd0;//发送ARP操作类型;
151                     6'd7       : crc_data <= {6'd0,arp_tx_type_r};//发送ARP操作类型;
152                     6'd8 , 6'd9 , 6'd10 , 6'd11 , 6'd12 , 6'd13 : crc_data <= BOARD_MAC[47 - 8*(cnt-8) -: 8];//发送源MAC地址;
153                     6'd14 , 6'd15 , 6'd16 , 6'd17 : crc_data <= BOARD_IP[31 - 8*(cnt-14) -: 8];//发送源IP地址;
154                     6'd18 , 6'd19 , 6'd20 , 6'd21 , 6'd22 , 6'd23 : crc_data <= des_mac_r[47 - 8*(cnt-18) -: 8];//发送目的MAC地址;
155                     6'd24 , 6'd25 , 6'd26 , 6'd27 : crc_data <= des_ip_r[31 - 8*(cnt-24) -: 8];//发送目的IP地址;
156                     default : crc_data <= 8'd0;//其余时间补0;
157                 endcase
158             end
159         endcase
160     end
161 end
162
163

```



```

164 //生成一个crc_data指示信号，用于生成gmii_txd信号。
165 always@(posedge clk)begin
166     if(rst_n==1'b0)begin//初始值为0;
167         gmii_tx_en_r <= 1'b0;
168     end
169     else if(state_c == CRC)begin
170         gmii_tx_en_r <= 1'b0;
171     end
172     else if(state_c == PREAMBLE)begin
173         gmii_tx_en_r <= 1'b1;
174     end
175 end
176
177 //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。
178 always@(posedge clk)begin
179     if(rst_n==1'b0)begin//初始值为0;
180         crc_en <= 1'b0;
181     end
182     else if(state_c == CRC)begin//当ARP和以太网帧头数据全部输出后拉低。
183         crc_en <= 1'b0;
184     end//当开始输出以太网帧头时拉高。
185     else if(state_c == ETH_HEAD && add_cnt)begin
186         crc_en <= 1'b1;
187     end
188 end
189
190 //生产CRC校验模块清零信号，状态机处于空闲时清零。
191 always@(posedge clk)begin
192     crc_clr <= (state_c == IDLE);
193 end
194
195 //生成gmii_txd信号，默认输出0。
196 always@(posedge clk)begin
197     if(rst_n==1'b0)begin//初始值为0;
198         gmii_txd <= 8'd0;
199     end//在输出CRC状态时，输出CRC校验码，先发送低位数据。
200     else if(state_c == CRC && add_cnt && cnt>0)begin
201         gmii_txd <= crc_out[8*cnt-1 -: 8];
202     end//其余时间如果crc_data有效，则输出对应数据。
203     else if(gmii_tx_en_r)begin
204

```

```

205         gmii_txd <= crc_data;
206     end
207 end
208
209 //生成gmii_txd有效指示信号。
210 always@(posedge clk)begin
211     gmii_tx_en <= gmii_tx_en_r || (state_c == CRC);
212 end
213
214 //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。
215 //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。
216 always@(*)begin
217     if(arp_tx_en || state_c != IDLE)
        rdy = 1'b0;
    else
        rdy = 1'b1;
    end
end

```

ARP发送模块仿真如下图所示，当arp\_tx\_en有效时，状态机跳转到发送前导码和帧起始符状态，开始产生数据。

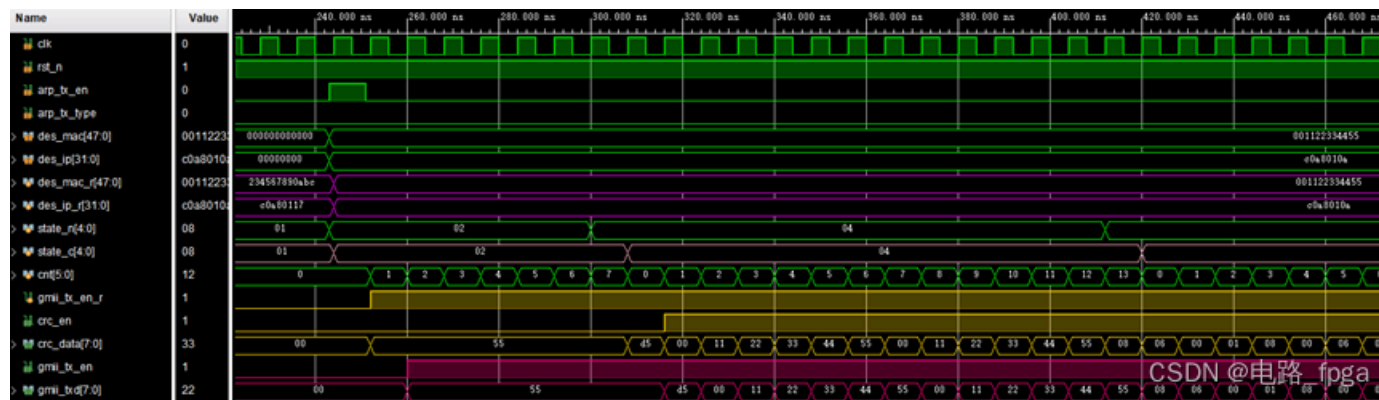


图11 ARP发送模块仿真

下图为发送CRC校验字段的仿真，红色信号是最终输出的数据，而橙色crc\_out是CRC校验模块计算得到的数据。

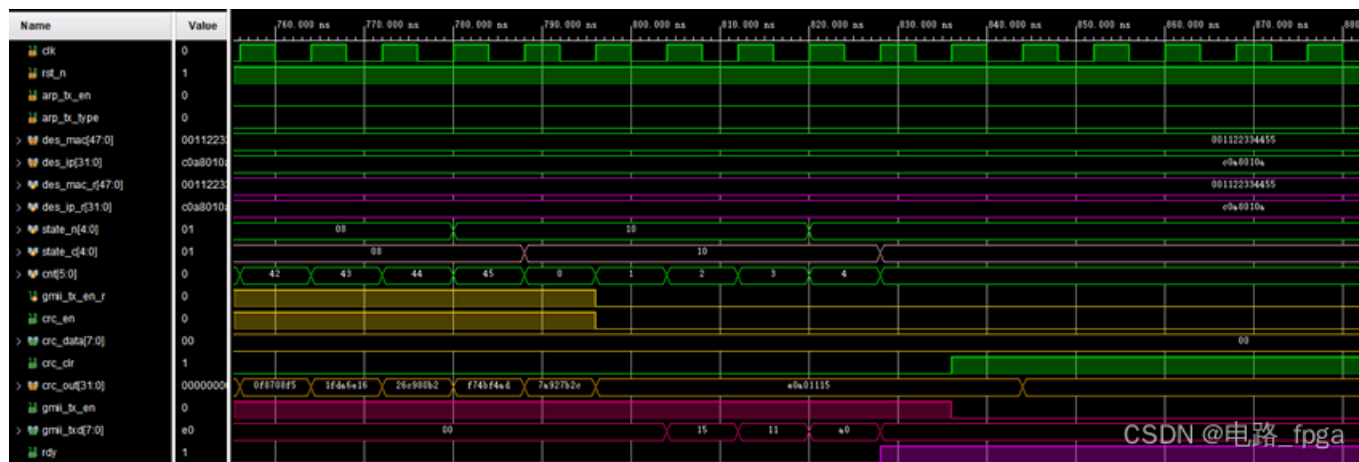


图12 ARP发送模块仿真

ARP发送和接收模块的设计就到此结束了，后续上板时通过ILA抓取，查看代码运行。

### 3、ARP\_CTRL模块设计

如果按键按下，FPGA向PC发送ARP请求，如果FPGA接收到PC发出的ARP请求，则FPGA向PC端发送ARP应答。

ARP控制模块检测到按键按下且ARP发送模块处于空闲时，将ARP发送模块的开始发送信号拉高，向PC端发出ARP请求。如果检测到PC端发给FPGA的ARP请求，则将ARP发送模块的开始信号拉高。

参考代码如下所示：

```

1  always@(posedge clk)begin
2      if(rst_n==1'b0)begin//初始值为0;
3          start <= 1'b0;
4      end
5      else if(key_in || (arp_rx_done && ~arp_rx_type))begin
6          start <= 1'b1;
7      end
8      else if(arp_tx_rdy)begin//当下游模块处于空闲时拉低。
9          start <= 1'b0;
10     end
11 end
12
13 //当需要发送ARP指令且发送模块空闲时有效，其余时间均为低电平。
14

```

```

15     always@(posedge clk)begin
16         arp_tx_en <= start && arp_tx_rdy;
17     end
18
19     always@(posedge clk)begin
20         if(rst_n==1'b0)begin//初始值为0;
21             arp_tx_type <= 1'b0;
22         end
23         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。
24             arp_tx_type <= 1'b1;
25         end
26         else if(key_in || (arp_rx_done && arp_rx_type))begin//其余时间发送请求指令。
27             arp_tx_type <= 1'b0;
28         end
29     end
30 end

```



由于模块过于简单，就不进行仿真了。

## 4、上板测试

将顶层模块综合，加入ILA，然后上板测试，综合工程后下载程序到板子，然后做一些准备，顶层将目的IP地址设置为192.168.1.102，所以我们需要先把电脑的IP地址设置为192.168.1.102。

```

rj > top.srcs > sources_1 > imports > src > top.v
1  module top #(
2      parameter    TIME_20MS    = 20_000_000    ,//按键抖动持续的最长时间，默认最长持续时间为20ms。
3      parameter    TIME_CLK     = 8              ,//系统时钟周期，默认8ns。
4      parameter    BOARD_MAC    = 48'h00_11_22_33_44_55 ,//开发板MAC地址 00-11-22-33-44-55;
5      parameter    BOARD_IP     = {8'd192,8'd168,8'd1,8'd10} ,//开发板IP地址 192.168.1.10;
6      parameter    DES_MAC      = 48'hff_ff_ff_ff_ff_ff ,//目的MAC地址 ff_ff_ff_ff_ff_ff;
7      parameter    DES_IP       = {8'd192,8'd168,8'd1,8'd102} //目的IP地址 192.168.1.102;
8  )

```

图13 顶层模块

如下图所示，在电脑的搜索框中输入网络连接，然后点击查看网络连接，开发板网口与电脑通过网线连接后，图中3处就会出现以太网。

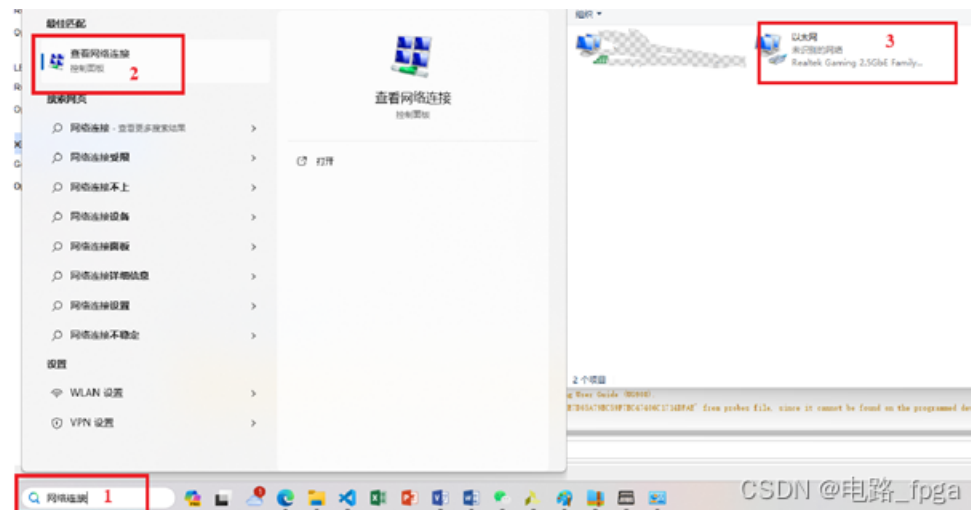


图14 查看网络连接

然后选中以太网鼠标右键，点击属性。



图15 查看属性

然后进行下图所示操作，首先双击协议版本，然后手动设置IP，将IP地址设置成图13中目的IP地址一致，其余设置与图16保持一致，之后点击确定。

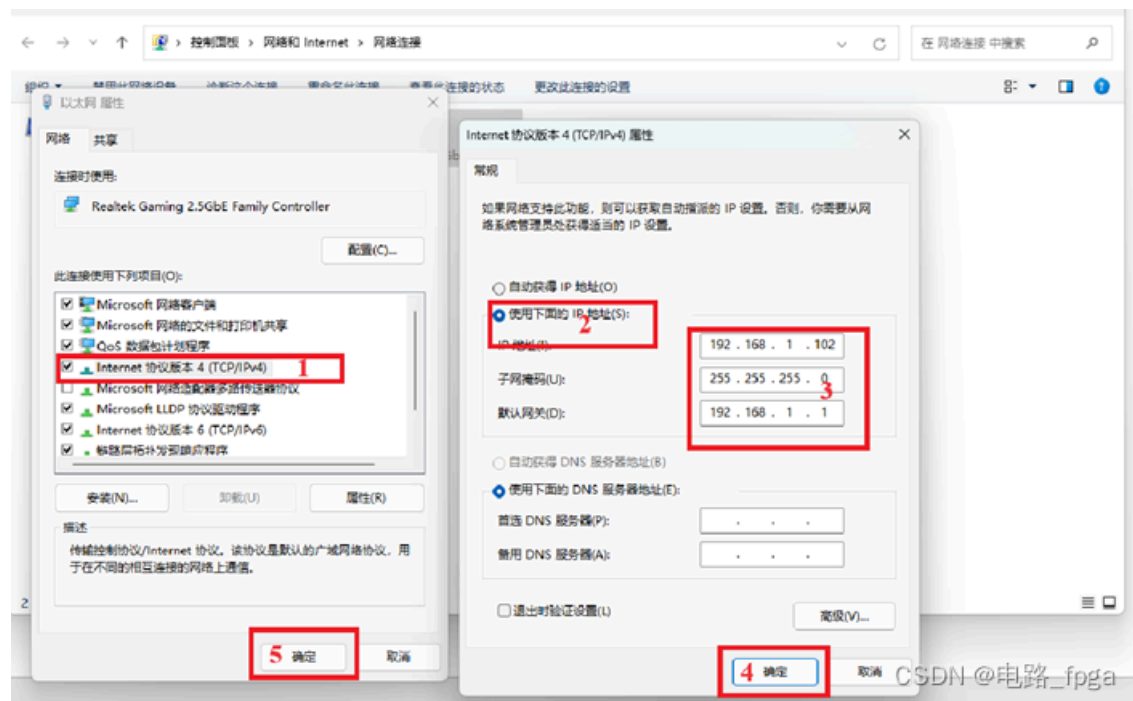


图16 PC的IP设置

然后再电脑的搜索栏中输入dos，鼠标右键命令提示符，然后以管理员身份运，如下图所示。

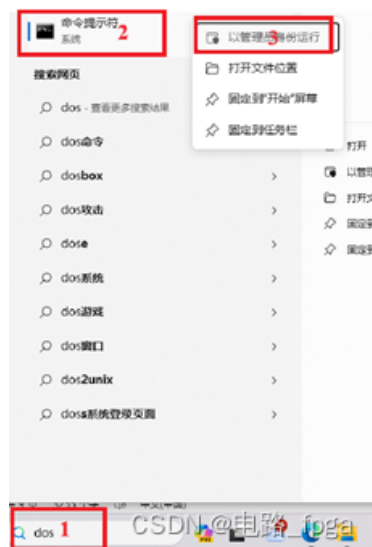






图20 PC端发出ARP应答数据报

将应答数据报放大后，如下图所示，此时目的MAC地址就是开发板的目的MAC地址，因为PC已经知道FPGA的MAC地址了。arp\_rx\_done信号拉高表示该数据报接收完成且CRC校验通过。

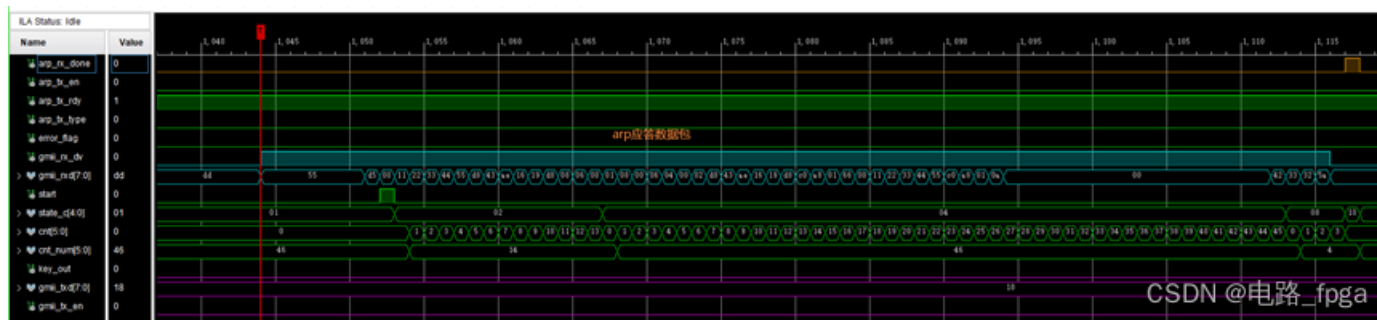


图21 PC端发出的ARP应答数据报

此时在命令提示符中输入arp -a，就可以查到开发板的MAC地址和IP地址了，如下图所示。



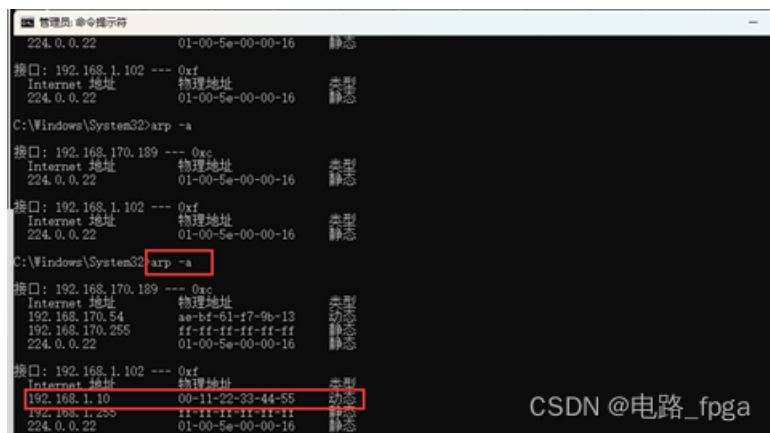


图22 命令提示符查询arp连接

此时我们还可以打开wireshark软件抓取以太网数据报，打开wireshark软件，如下图所示，点击以太网。

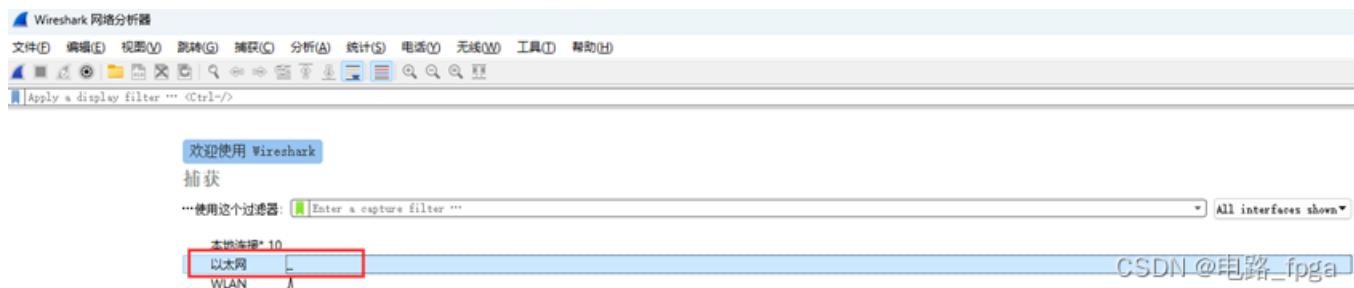


图23 打开wireshark软件

然后按下开发板的按键，wireshark软件就能够抓到FPGA发送给PC的ARP请求数据报和PC回复FPGA的ARP应答包，如下图所示。

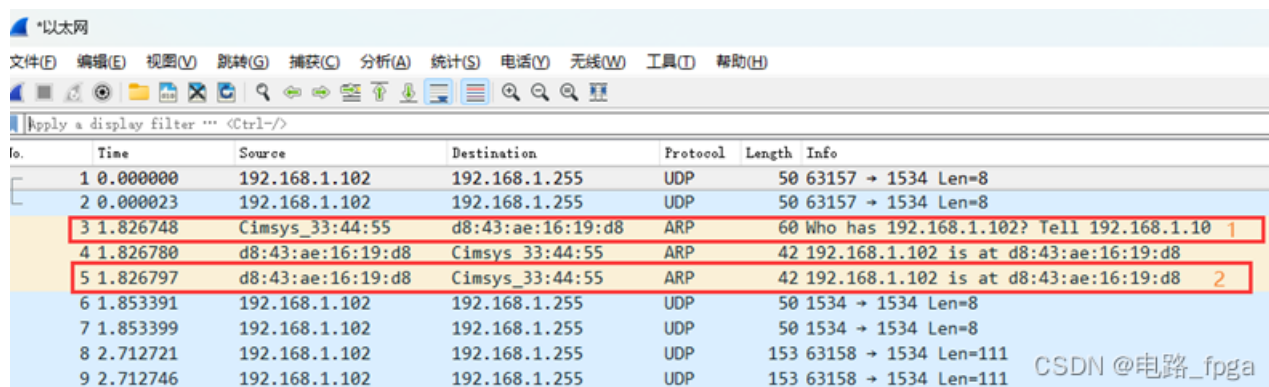


图24 wireshark软件抓取ARP数据报

1是FPGA发送的ARP请求数据报，双击该数据报，得到如下图所示，可以看到下图wireshark抓到的数据与图20中ILA抓取的数据是一致的。

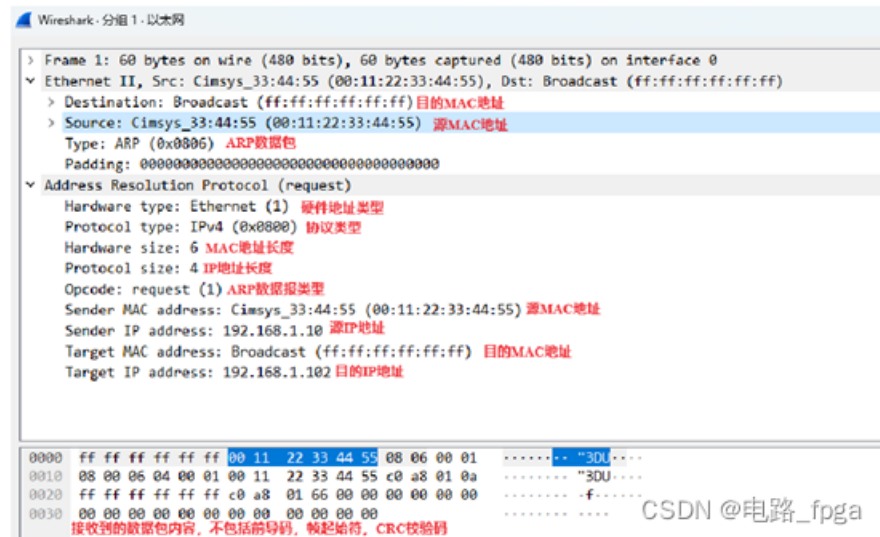


图25 wireshark抓取的ARP请求包

2是PC回复FPGA的ARP应答数据报，双击打开该数据报，如下图所示，报文类型为2，证明是ARP应答包，该数据报发送的数据与图21中ILA抓取的数据一致。

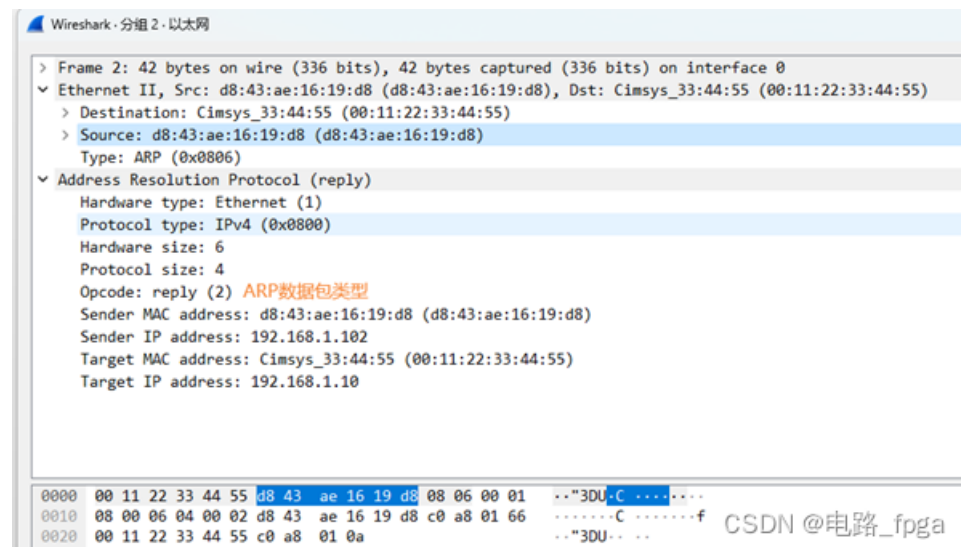
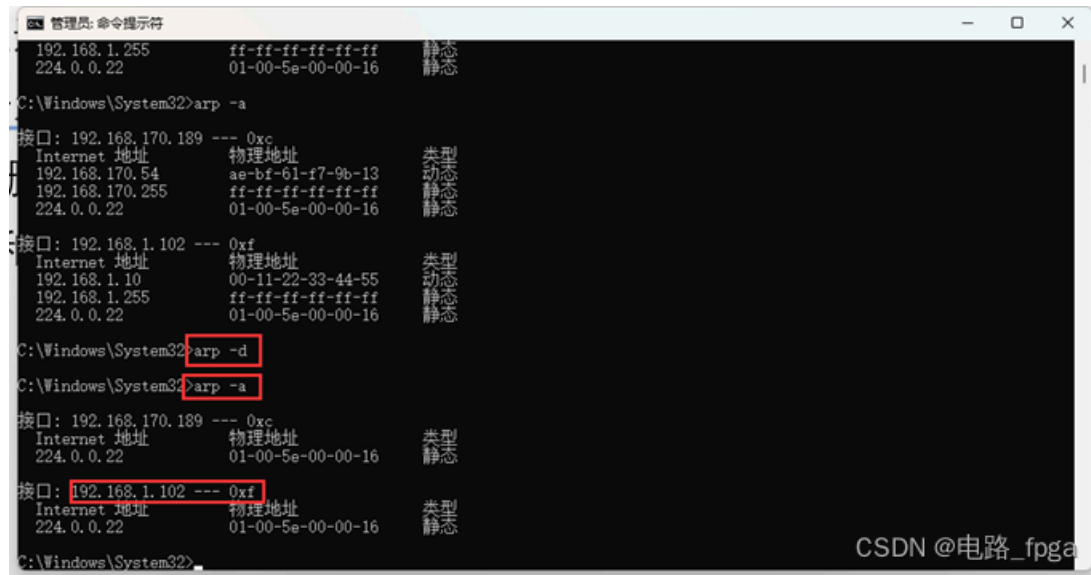


图26 wireshark抓取的ARP应答包

上述对FPGA发送ARP请求，PC回复ARP应答进行了验证，后面需要对PC端发出ARP请求，FPGA是否能够回复PC端进行验证。首先通过arp -d指令，删除PC端arp连接，然后再使用arp -a查询arp列表，得知开发板的MAC地址和IP地址已经被PC清除。



```
管理员: 命令提示符
192.168.1.255      ff-ff-ff-ff-ff-ff 静态
224.0.0.22        01-00-5e-00-00-16 静态

C:\Windows\System32>arp -a

接口: 192.168.170.189 --- 0xc
Internet 地址      物理地址          类型
192.168.170.54      ae-bf-61-f7-9b-13 动态
192.168.170.255     ff-ff-ff-ff-ff-ff 静态
224.0.0.22          01-00-5e-00-00-16 静态

接口: 192.168.1.102 --- 0xf
Internet 地址      物理地址          类型
192.168.1.10       00-11-22-33-44-55 动态
192.168.1.255      ff-ff-ff-ff-ff-ff 静态
224.0.0.22          01-00-5e-00-00-16 静态

C:\Windows\System32>arp -d

C:\Windows\System32>arp -a

接口: 192.168.170.189 --- 0xc
Internet 地址      物理地址          类型
224.0.0.22          01-00-5e-00-00-16 静态

接口: 192.168.1.102 --- 0xf
Internet 地址      物理地址          类型
224.0.0.22          01-00-5e-00-00-16 静态

C:\Windows\System32>
```

图27 清除PC端arp列表

然后开发板重新下载程序，ILA准备抓取数据，wireshark清除抓取的数据后重新抓取数据，然后再命令提示符窗口输入ping 192.168.1.10，因为开发板没有实现ICMP协议，所以并不能响应，但是该指令PC端也会向开发板发送ARP请求指令，所以本文可以用该指令进行测试。

当该指令运行结束后，输入arp -a指令，就可以查看到FPGA的MAC地址和IP地址已经存在PC端的ARP列表里了，验证成功。

```
管理工具: 命令提示符
接口: 192.168.1.102 --- 0xf
Internet 地址      物理地址      类型
224.0.0.22         01-00-5e-00-00-16 静态

C:\Windows\System32>ping 192.168.1.10

正在 Ping 192.168.1.10 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

192.168.1.10 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 0, 丢失 = 4 (100% 丢失),

C:\Windows\System32>arp -a

接口: 192.168.170.189 --- 0xc
Internet 地址      物理地址      类型
192.168.170.54     ae-bf-61-f7-9b-13 动态
192.168.170.255     ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态

接口: 192.168.1.102 --- 0xf
Internet 地址      物理地址      类型
192.168.1.10       00-11-22-33-44-55 动态
192.168.1.255      ff-ff-ff-ff-ff-ff 静态
224.0.0.22         01-00-5e-00-00-16 静态
224.0.0.251        01-00-5e-00-00-fb 静态
239.255.255.250    01-00-5e-7f-ff-fa 静态

C:\Windows\System32>
```

图28 PC端发起ARP请求

接下来查看ILA抓取的数据报吧，ILA深度设置的比较大，并且可以触发32次，所以基本上不会漏掉数据报。

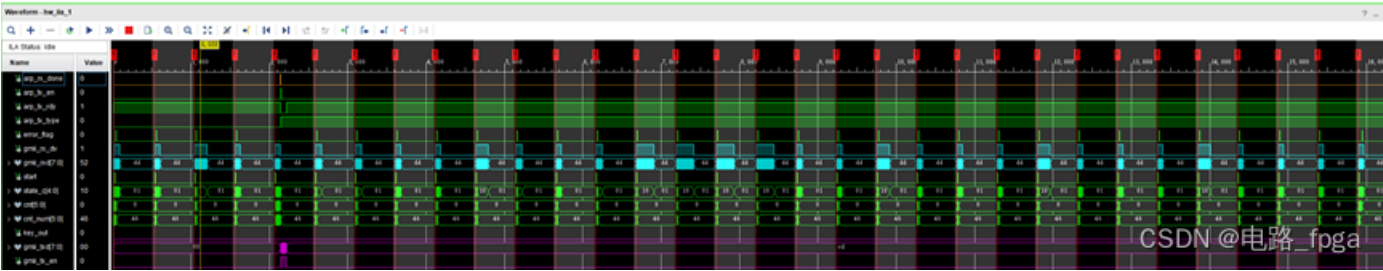


图29 ILA抓取的数据报

如下图所示，ILA抓取PC端发送给FPGA的ARP请求数据报后，FPGA马上向PC端回复了ARP应答数据报。

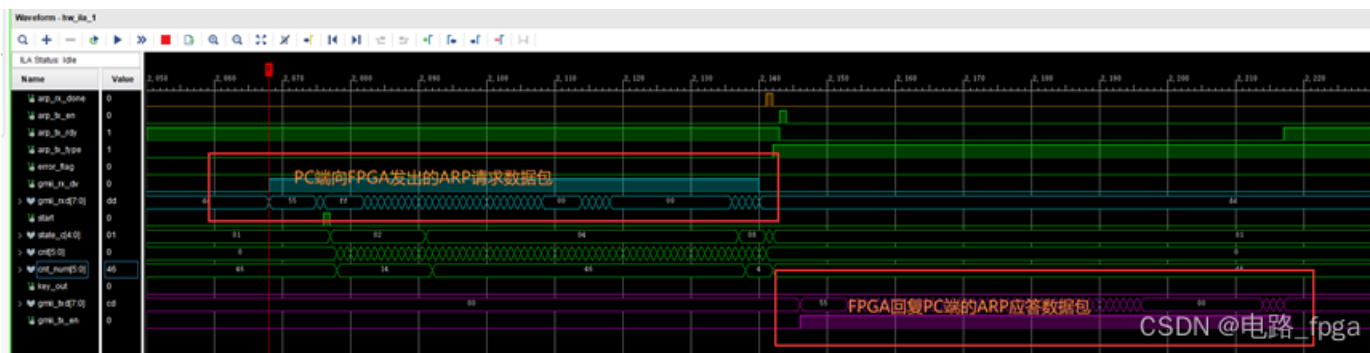


图30 ILA抓取的ARP数据报

下图为ILA抓取的ARP请求数据报，PC端发出的ARP请求数据报的以太网帧头的目的MAC地址为广播地址，但是在ARP数据段中，发送的目的MAC地址全为0，并不是广播地址，这也许是因为接收端不会解析这部分数据，所以就没有做处理吧。

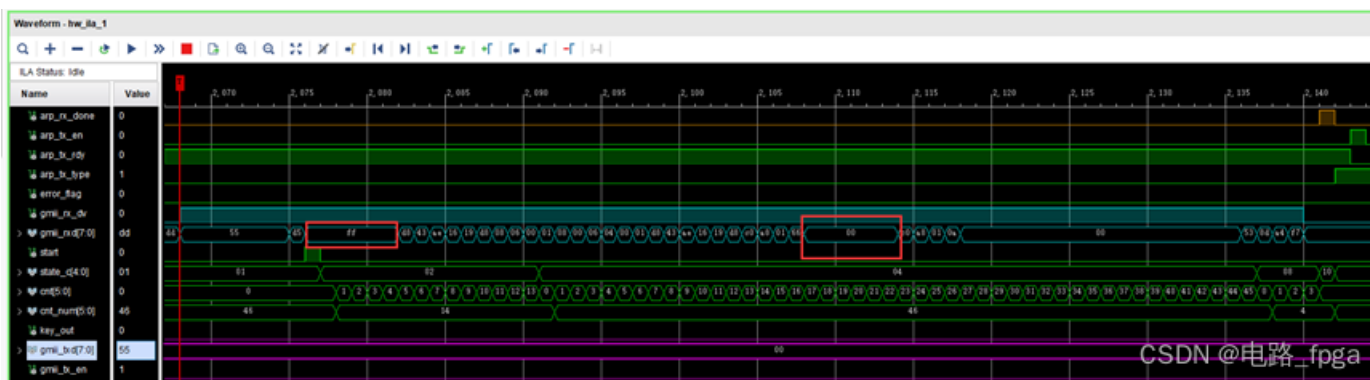


图31 ILA抓取PC端发出的ARP请求包

同时wireshark抓取的数据报问如下所示，不知道为什么，我的wireshark每次都会显示发送了2次的ARP请求，但是ILA和FPGA都只能收到一个数据报。

| No. | Time      | Source            | Destination       | Protocol | Length | Info   |
|-----|-----------|-------------------|-------------------|----------|--------|--|
| 1   | 0.000000  | 192.168.1.102     | 192.168.1.255     | UDP      | 50     | 1534 → 1534 Len=8  |
| 2   | 0.000009  | 192.168.1.102     | 192.168.1.255     | UDP      | 50     | 1534 → 1534 Len=8  |
| 3   | 0.680695  | 192.168.1.102     | 192.168.1.255     | UDP      | 153    | 63158 → 1534 Len=111   |
| 4   | 0.680713  | 192.168.1.102     | 192.168.1.255     | UDP      | 153    | 63158 → 1534 Len=111   |
| 5   | 6.904943  | 192.168.1.102     | 192.168.1.255     | UDP      | 50     | 61014 → 1534 Len=8   |
| 6   | 6.904969  | 192.168.1.102     | 192.168.1.255     | UDP      | 50     | 61014 → 1534 Len=8   |
| 7   | 8.100038  | d8:43:ae:16:19:d8 | Broadcast         | ARP      | 42     | Who has 192.168.1.10? Tell 192.168.1.102                                 |
| 8   | 8.100051  | d8:43:ae:16:19:d8 | Broadcast         | ARP      | 42     | Who has 192.168.1.10? Tell 192.168.1.102                                 |
| 9   | 8.100100  | 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  | 8.100111  | 192.168.1.102     | 192.168.1.10      | ICMP     | 74     | Echo (ping) request id=0x0001, seq=13/3328, ttl=128 (no response found!) |
| 11  | 8.100117  | 192.168.1.102     | 192.168.1.10      | ICMP     | 74     | Echo (ping) request id=0x0001, seq=13/3328, ttl=128 (no response found!) |
| 12  | 12.997107 | 192.168.1.102     | 192.168.1.10      | ICMP     | 74     | Echo (ping) request id=0x0001, seq=14/3584, ttl=128 (no response found!) |
| 13  | 12.997129 | 192.168.1.102     | 192.168.1.10      | ICMP     | 74     | Echo (ping) request id=0x0001, seq=14/3584, ttl=128 (no response found!) |

图32 wireshark抓取数据报

打开ARP请求数据报，如下图所示，两个目的MAC地址的数值也不一致，应该跟猜想一样吧。

```

Wireshark - 分组 7 - 以太网
> Frame 7: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
  > Ethernet II, Src: d8:43:ae:16:19:d8 (d8:43:ae:16:19:d8), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    > Source: d8:43:ae:16:19:d8 (d8:43:ae:16:19:d8)
      Type: ARP (0x0806)
    > Address Resolution Protocol (request)
      Hardware type: Ethernet (1)
      Protocol type: IPv4 (0x0800)
      Hardware size: 6
      Protocol size: 4
      Opcode: request (1)
      Sender MAC address: d8:43:ae:16:19:d8 (d8:43:ae:16:19:d8)
      Sender IP address: 192.168.1.102
      Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
      Target IP address: 192.168.1.10
  
```

0000 ff ff ff ff ff d8 43 ae 16 19 d8 08 06 00 01 .....C .....  
 0010 08 00 06 04 00 01 d8 43 ae 16 19 d8 c0 a8 01 66 .....C .....f  
 0020 00 00 00 00 00 00 c0 a8 01 0a ..... ..

CSDN @电路\_fpga

图33 wireshark抓取ARP请求数据报

然后查看FPGA发送的ARP应答数据报，ILA抓取的结果如下图所示，紫红色信号就是以太网发送信号。目的MAC地址就是图33中PC端的MAC地址，目的IP也是电脑的IP地址。





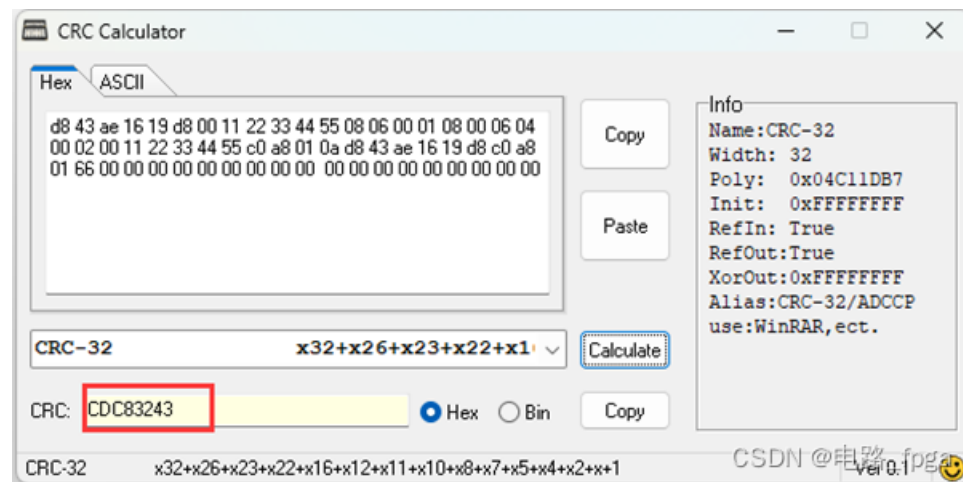


图36 验证CRC计算

至此，本文的验证就结束了，主要通过FPGA解析ARP数据报，然后向PC端发送ARP数据报，也简单介绍了wireshark软件的使用，利用该软件，结合ILA再调试时可以事半功倍。后续的ICMP和UDP其实都离不开ARP协议，学会设计ARP协议后，ICMP和UDP的实现也就比较简单了。

在接收数据时充分使用移位寄存器还可以将代码简化，有些地方接收数据根本不需要进行移位操作，在计数器某个状态时，直接对移位寄存器中某段数据保存即可，也可以简化判断电路。

本工程可以在公众号后台回复“**基于FPGA的ARP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⭐！**