UDP 以太网测试

[文档副标题]

摘要

[通过迷人的摘要吸引您的读者。它通常是文件的简短摘要。当您准备好添加内容时，只需单击此处并开始键入。]

Edge Wang

[电子邮件地址]

目录

概述

- **以太网上使用两种标准帧格式。**

第一种是上世纪 80 年代初提出的 DIX v2 格式，即 Ethernet II 帧格式。

Ethernet II 后来被 IEEE 802 标准接纳，并写进了 IEEE 802.3x-1997 的 3.2.6 节。

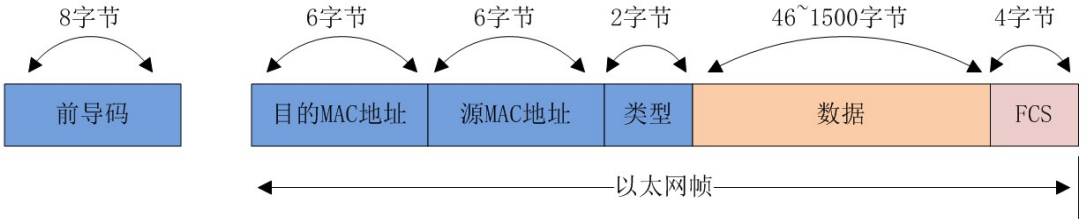
第二种是 1983 年提出的 IEEE 802.3 格式。这两种格式的主要区别在于，Ethernet II 格式中包含一个 **Type** 字段，标识以太帧处理完成之后将被发送到哪个上层协议进行处理。IEEE 802.3 格式中，同样的位置是长度字段。

不同的 Type 字段值可以用来区别这两种帧的类型

当 Type 字段值小于等于 1500 (或者十六进制的 0x05DC) 时，帧使用的是 IEEE 802.3 格式。
当 Type 字段值大于等于 1536 （或者十六进制的 0x0600）时，帧使用的是 Ethernet II 格式。以太网中大多数的数据帧使用的是 Ethernet II 格式。

1、Ethernet_II 帧格式

下图为以太网的帧格式。



前导码（Preamble）：8 字节，连续 7 个 8 ‘h55 加 1 个 8’ hd5，表示一个帧的开始，用于双方设备数据的同步。

目的 MAC 地址：6 字节，存放目的设备的物理地址，即 MAC 地址。

源 MAC 地址：6 字节，存放发送端设备的物理地址

说明：MAC 地址由两部分组成，分别是供应商代码和序列号。其中前 24 位代表该供应商代码，由 IEEE 管理和分配。剩下的 24 位序列号由厂商自己分配。

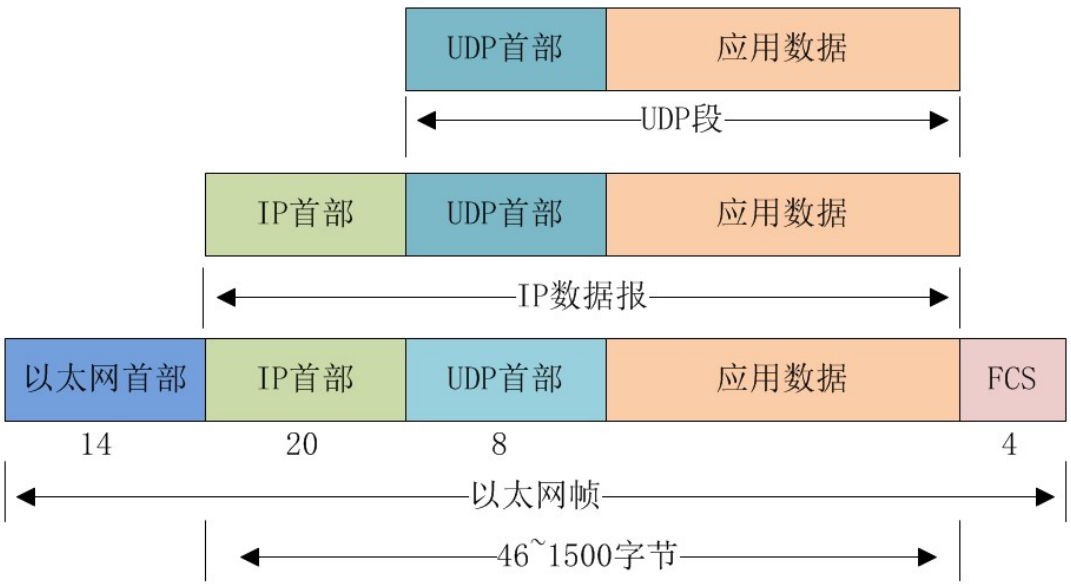
类型：2 字节，用于指定协议类型，常用的有 0800 表示 IP 协议；0806 表示 ARP 协议；8035 表示 RARP 协议 。

数据：46 到 1500 字节，最少 46 字节，不足需要补全 46 字节，例如 IP 协议层就包含在数据部分，包括其 IP 头及数据。

FCS：帧尾，4 字节，称为帧校验序列，采用 32 位 CRC 校验，对目的 MAC 地址字段到数据字段进行校验。

其中前面的目的 MAC 地址、源 MAC 地址和类型共 14 个字节称为**以太网首部**。

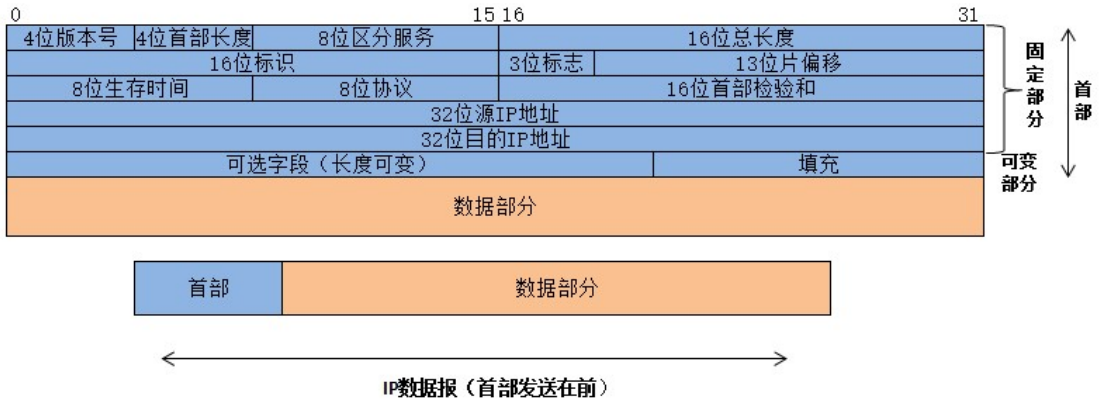
进一步扩展，以 UDP 协议为例，可以看到其结构如下，除了**以太网首部**的 14 字节，数据部分包含 **IP 首部**，**UDP 首部**和**应用数据**共 46~1500 字节。



下面再介绍下 IP 首部

2、IP 首部数据报格式

下图为 IP 分组的报文头格式，报文头的前 20 个字节是固定的，后面的可发



版本:占 4 位,指 IP 协议的版本目前的 IP 协议版本号为 4 (即 IPv4)。

```
localparam  VER = 4'h4;//IPv4
```

首部长度:占 4 位,可表示的最大数值是 15 个单位(一个单位为 4 字节)因此 IP 的首部长度的最大值是 60 字节。

```
localparam  IHL = 4'h5;//Internet Header Length
```

区分服务:占 8 位,用来获得更好的服务,在旧标准中叫做服务类型,但实际上一直未被使用过.1998 年这个字段改名为区分服务.只有在使用区分服务(DiffServ)时,这个字段才起作用.一般的情况下都不使用这个字段。

```
localparam  TOS = 8'h0;//Type Of Service
```

总长度:占 16 位,指首部和数据之和的长度,单位为字节,因此数据报的最大长度为 65535 字节.总长度必须不超过最大传送单元 MTU。MTU 是最大传输单元。

```
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        ip_len <= 16'h0;
    else
        ip_len <= udp_len + 16'd20;
end
```

标识:占 16 位,它是一个计数器,用来产生数据报的标识。

```
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        ip_id <= 16'h0;
    else if((cur_state == PAT_GEN) && (tlast == 1'b1) && (tready == 1'b1))
        ip_id <= ip_id + 1'b1;
end
```

标志(flag): 占 3 位,
最高位：保留位，没有意义
中间位：DF,Don’ t Fragment; DF =1,禁止分片；DF = 0 时，允许分片。
最低位：MF 位，More Freagment; MF = 1 时，后面还有分片；MF = 0 时本分片就是该分组的最后一个分片。
只有 DF = 0 时，MF 才有意义。

```
localparam  FLG = 3'h0;//Flags
```

片偏移:占 13 位,指较长的分组的分片 ，中间的某个分片 ，在原来的 IP 分组中的相对位置 ；单位是 8 字节 ；也就是说除了最后一个分片 ，每个分片的长度是 8 字节的整数倍 ；

```
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        ip_ofs <= 13'h0;
end
```

生存时间:占 8 位,记为 TTL(Time To Live) 数据报在网络中可通过的路由器数的最大值,TTL 字段是由发送端初始设置一个 8 bit 字段.推荐的初始值由分配数字 RFC 指定,当前值为 64.发送 ICMP 回显应答时经常把 TTL 设为最大值 255。

```
localparam  TTL = 8'h40;//Time To Live
```

协议:占 8 位,指出此数据报携带的数据使用何种协议以便目的主机的 IP 层将数据部分上交给哪个处理过程, 1 表示为 ICMP 协议, 2 表示为 IGMP 协议, 6 表示为 TCP 协议, 17 表示为 UDP 协议。

```
localparam  PTC = 8'h11;//UDP Protocol
```

首部检验和:占 16 位,只检验数据报的首部不检验数据部分，采用二进制反码求和:
（1） 将 16 位数据相加得到 32 位的数据，

（2） 如果高 16 位不为 0，再将高 16 位与低 16 位相加，直到进位为 0，

（3） 最后将 16 位取反。

在实际处理过程中，代码如下，程序始终检测第 16 位数据，当检测到第 16 位不为 0，立即执行上面第（2）步。

```
//ip checksum calculate
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        ip_chksum_r <= 17'h0;
    else if(cur_state == IDLE)
        ip_chksum_r <= 17'h0;
    else if(cur_state == IP_CHKSUM) begin
        case(ip_chksum_cnt)
        4'd0  : ip_chksum_r <= ip_chksum_r[15:0] + {VER,IHL,TOS} + ip_chksum_r[16];
        4'd1  : ip_chksum_r <= ip_chksum_r[15:0] + ip_len + ip_chksum_r[16];
        4'd2  : ip_chksum_r <= ip_chksum_r[15:0] + ip_id + ip_chksum_r[16];
        4'd3  : ip_chksum_r <= ip_chksum_r[15:0] + {FLG,ip_ofs} + ip_chksum_r[16];
        4'd4  : ip_chksum_r <= ip_chksum_r[15:0] + {TTL,PTC} + ip_chksum_r[16];
        4'd5  : ip_chksum_r <= ip_chksum_r[15:0] + src_ip_r[31:16] + ip_chksum_r[16];
        4'd6  : ip_chksum_r <= ip_chksum_r[15:0] + src_ip_r[15:0] + ip_chksum_r[16];
        4'd7  : ip_chksum_r <= ip_chksum_r[15:0] + dst_ip_r[31:16] + ip_chksum_r[16];
        4'd8  : ip_chksum_r <= ip_chksum_r[15:0] + dst_ip_r[15:0] + ip_chksum_r[16];
        4'd9  : ip_chksum_r <= ip_chksum_r[15:0] + ip_chksum_r[16];
        default:;
        endcase
    end
    else if(cur_state == PAT_IPG)
        ip_chksum_r <= 17'h0;
end
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        ip_chksum <= 16'h0;
    else
        ip_chksum <= ~ip_chksum_r[15:0];
end
```

源地址和目的地址:都各占 4 字节,分别记录源地址和目的地址

2.2 分片机制 示例 ：

IP 数据报 ： 首部 20 字节 ， 数据部分 3800 字节 ；

将其进行分片处理 ： 每个分片不超过 1420 字节 ；

标识 ：666;

标志 ：DF = 0 ， 表示允许分片 ；MF = 0 ， 表示后续没有分片 ；

片偏移量 ：0

分片后的结果是：分成 三片；
第一片：
分片数据：首部 1（20 字节）+ 1400 字节数据部分；
标识：666, 同一个分组的分片，标识相同；
标志：DF = 0，允许分片；MF = 1，后续还有分片；
片偏移量：片偏移量 是 0，单位是 8 字节，本片偏移量相当于 0 字节；

第二片：
分片数据：首部 2（20 字节）+ 1400 字节数据部分；
标识：666, 同一个分组的分片，标识相同；
标志：DF = 0，允许分片；MF = 1，后续还有分片；
片偏移量：片偏移量 是 175，单位是 8 字节，本片偏移量相当于 1400 字节；

第三片：
分片数据：首部 3（20 字节）+ 1000 字节数据部分；
标识：666, 同一个分组的分片，标识相同；
标志：DF = 0，允许分片；MF = 0，后续没有分片；
片偏移量：片偏移量 是 350，单位是 8 字节，本片偏移量相当于 2800 字节；

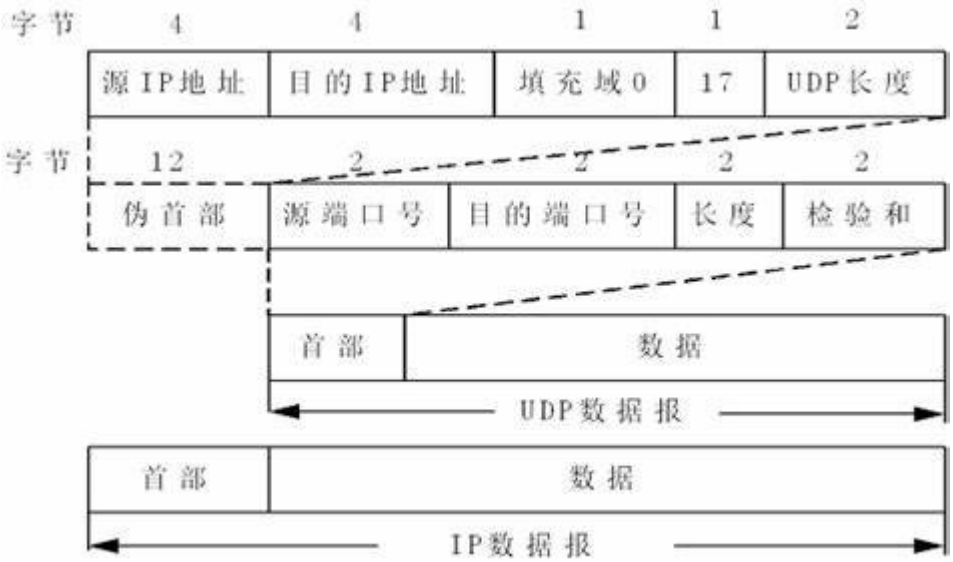
片偏移量是从数据部分开始计数，数据部分的开始位置是 0 字节，其单位是 8 字节，片偏移量 1 代表 8 字节；

3、UDP 协议

UDP 是 User Datagram Protocol（用户数据报协议）的英文缩写。UDP 只提供一种基本的、低延迟的被称为数据报的通讯。所谓数据报，就是一种自带寻址信息，从发送端走到接收端的数据包。UDP 协议经常用于图像传输、网络监控数据交换等数据传输速度要求比较高的场合。

3.1 UDP 协议的报头格式：

UDP 报头由 4 个域组成，其中每个域各占用 2 个字节，具体如下：



源端端口：源端口号，占用两个字节。在需要对方回信时选用。不需要时可用全 0。

目的端端口：目的端口号，占用两个字节。这在终点交付报文时必须要使用到。

用户数据包长度：UDP 用户数据报的长度，占用两个字节，是 UDP 首部与 UDP 数据的长度和，其最小值是 8（仅首部）。

```
udp_dlen_r 是指应用数据的长度，8 是指首部数据长度。
//dlen + source_len(2) +  dest_len(2) + user_len(2) + checksum(2)
//udp_dlen_r 的单位是字节
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        udp_len <= 16'h0;
    else
        udp_len <= udp_dlen_r + 16'd8;
end
```

校验和：两个字节长度。对 UDP 伪首部、UDP 首部和 UDP 数据进行校验，。伪首部长度为 12 个字节，首部为 8 个字节，但是 checksum 数据在首部所以要去掉，所以要加 9。检测 UDP 用户数据报在传输中是否有错。有错就丢弃。

UDP 伪首部指源地址、目的地址、UDP 数据长度、协议类型（0x11），协议类型就一个字节，但需要补一个字节的 0x0，构成 12 个字节。伪首部+UDP 首部+数据一起计算校验和。为何称之为“伪头部”？原因是，UDP 协议只使用它来辅助计算校验和，它并不是发送 IP 数据包时使用的 IP 数据包的头

部。

```
//数据是按 32 位去设计 checksum，所以要 divder2，如果应用数据长度为偶数，除 2 即可，如果是奇数，就要补 1。
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        udp_chksum_num <= 16'h0;
    else if(udp_dlen_r[0] == 1'b1)
        udp_chksum_num <= udp_dlen_r[15:1] + 16'd10;
    else
        udp_chksum_num <= udp_dlen_r[15:1] + 16'd9;
end
```

3.2 UDP 校验和计算

UDP 计算校验和的方法和 IP 数据报首部校验和的方法相似。不同的是：IP 数据报校验和只校验 IP 数据报的首部，但 UDP 的校验和是把 UDP 伪首部、UDP 首部和 UDP 数据部分一起检验。

UDP 检验和的计算方法是：

1. 从“伪头部”开始，按每 16 位当作一个数，逐次求和，最终得出一个 32 位的数；
2. 如果这个 32 位的数的高 16 位不为 0，再将其高 16 位与低 16 位相加，又得到一个 32 位的数；
3. 重复第 2 步直到高 16 位为 0。
4. 最终，将低 16 位取反，得到校验和，填入 checksum 字段中

```
//udp checksum calculate
always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        udp_chksum_r <= 17'h0;
    else if(cur_state == IDLE)
        udp_chksum_r <= 17'h0;
    else if(cur_state == UDP_CHKSUM) begin
        if (udp_chksum_cnt <= 16'd8) begin
            case(udp_chksum_cnt[3:0])
                4'd0 : udp_chksum_r <= udp_chksum_r[15:0] + src_ip_r[31:16] + udp_chksum_r[16];//伪首部
                4'd1 : udp_chksum_r <= udp_chksum_r[15:0] + src_ip_r[15:0] + udp_chksum_r[16];//伪首部
                4'd2 : udp_chksum_r <= udp_chksum_r[15:0] + dst_ip_r[31:16] + udp_chksum_r[16];//伪首部
                4'd3 : udp_chksum_r <= udp_chksum_r[15:0] + dst_ip_r[15:0] + udp_chksum_r[16];//伪首部
                4'd4 : udp_chksum_r <= udp_chksum_r[15:0] + 16'h11 + udp_chksum_r[16];//伪首部
                4'd5 : udp_chksum_r <= udp_chksum_r[15:0] + udp_len + udp_chksum_r[16];//伪首部
                4'd6 : udp_chksum_r <= udp_chksum_r[15:0] + src_port_r + udp_chksum_r[16];
                4'd7 : udp_chksum_r <= udp_chksum_r[15:0] + dst_port_r + udp_chksum_r[16];
                4'd8 : udp_chksum_r <= udp_chksum_r[15:0] + udp_len + udp_chksum_r[16];
                default : udp_chksum_r <= 17'h0;
```

```
            endcase
        end
        else begin
            if(udp_chksum_cnt == udp_chksum_num)
                udp_chksum_r <= udp_chksum_r[15:0] + udp_chksum_r[16];
            else if((udp_chksum_cnt == udp_chksum_num-1) && (udp_dlen_r[0] == 1'b1))
                udp_chksum_r <= udp_chksum_r[15:0] + {udp_data_h,8'h0} + udp_chksum_r[16];
            else
                udp_chksum_r <= udp_chksum_r[15:0] + {udp_data_h,udp_data_l} + udp_chksum_r[16];
        end
    end
end

always @(posedge clk or negedge rstn)
begin
    if(rstn == 1'b0)
        udp_chksum <= 16'h0;
    else
        udp_chksum <= ~udp_chksum_r[15:0];
end
```

3.3 差错检验

当接收到 UDP 报文时，需要如何检验其正确性？方法就是将 UDP 报文中包括校验和在内的，所有的 16 位的数相加，如果低 16 位全为 1，则没有出错。否则表明该分组中出现了错误。需要注意，UDP 对差错具有一定的校验能力，但缺少差错恢复的能力.

IO 配置

IO 管脚分配

管脚分配已经在硬件设计指导中给出。

RX IO 配置

parameter	rgmii_rx_ctl	rgmii_rxc	rgmii_rxd

Instance Name	rgmii_rx_ctl	rgmii_rxc	rgmii_rxd
Mode	input	input	input
I/O standard	1.8V lvcmos	1.8V lvcmos	1.8V lvcmos
pin Name(HI)	rgmii_rx_ctl_HI	rgmi_rxc	rgmii_rxd_HI
Pin Name(LO)	rgmii_rx_ctl_LO		rgmii_rxd_LO
connection Type	normal	gclk	normal
Register Option	register	register	register
Clock Pin Name	rgmii_rxc	-	rgmii_rxc
Double Data I/O Option	resync	-	resync
Enable invert clock	NO	-	NO
Pull Option	none	none	none
static delay	0	15	15

TX IO 配置

parameter	rgmii_tx_ctl	rgmii_txc	rgmii_txd
Instance Name	rgmii_tx_ctl	rgmii_txc	Rgmii_txd
Mode	output	output	output
I/O standard	1.8v lvcmos	1.8v lvcmos	1.8v lvcmos
pin Name(HI)	rgmii_tx_ctl_HI	rgmii_txc_HI	rgmii_txd_HI
Pin Name(LO)	rgmii_tx_ctl_LO	rgmii_txc_LO	rgmii_rxd_LO
Register Option	register	register	register
Double Data I/O Option	resync	resync	resync
Enable Serialization	NO	NO	NO
Drive Strength	4	4	4
Static Delay setting	0	0	0
Output Clock Pin Name	clk_125m	clk_125m_90deg	clk_125m
Enable invert clock	NO	NO	N0

Waveform

数据接收时，RXC 边沿与 RXD 有两种对应情况。一是 RXC 上升沿与 RXD 接收的数据的第一个数据对齐，如图 1；二是 RXC 下降沿与 RXD 接收的数据的第一个数据对齐，如图 2。

如果接收端 RXC 与接收的第一个数据是上升沿对齐。数据就需要重新对齐。

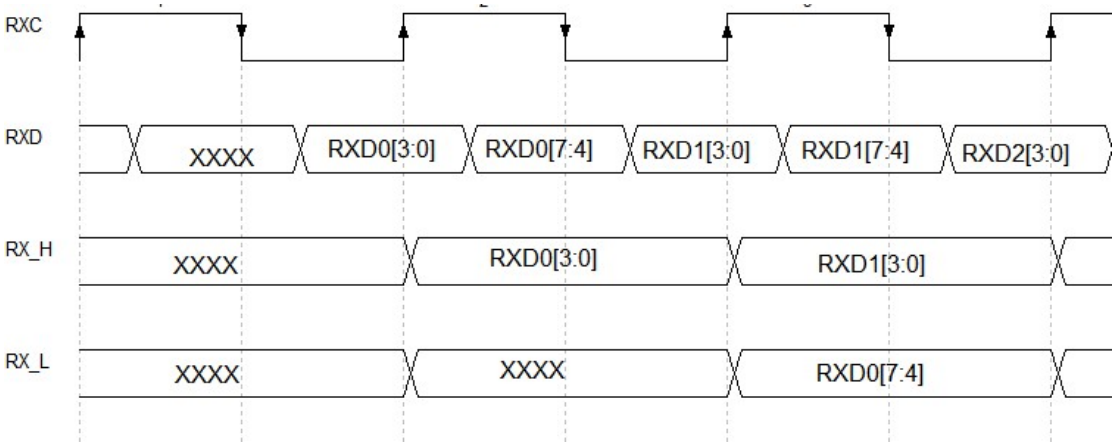


图 1

如果接收端 RXC 与接收的第一个数据是下降沿对齐。

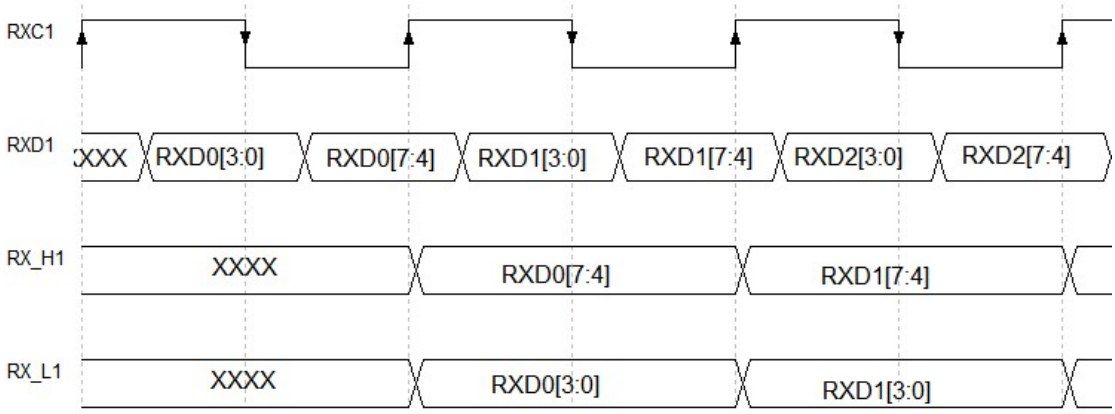
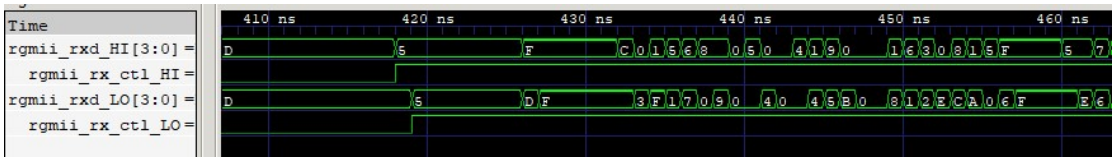


图 2

下图是我们抓取的一个数据包头。从前面我们知道前导码是 7 个 0x55 和一上 0xD5。但是我们发现是 rgmii_rxd_HI 比 rgmii_rxd_LO 提前一拍，并且”D”在 rgmii_rxd_LO 上。它符合上面提到的 RXC 的上升沿 RXD 的第一个数据的对应情况。



那对于这种情况怎么处理呢？其实在 IP 内部已经考虑到这一点。我们只需要打开生成的 tse IP 文件，修改 rgmii_rxc_edge 这个参数即可。

rgmii_rxc_edge	1’b0:rxc 上升沿对应 rxd 第一个有效数据。 1’b1:rxd 下降沿对应 rxd 第一个有效数据。

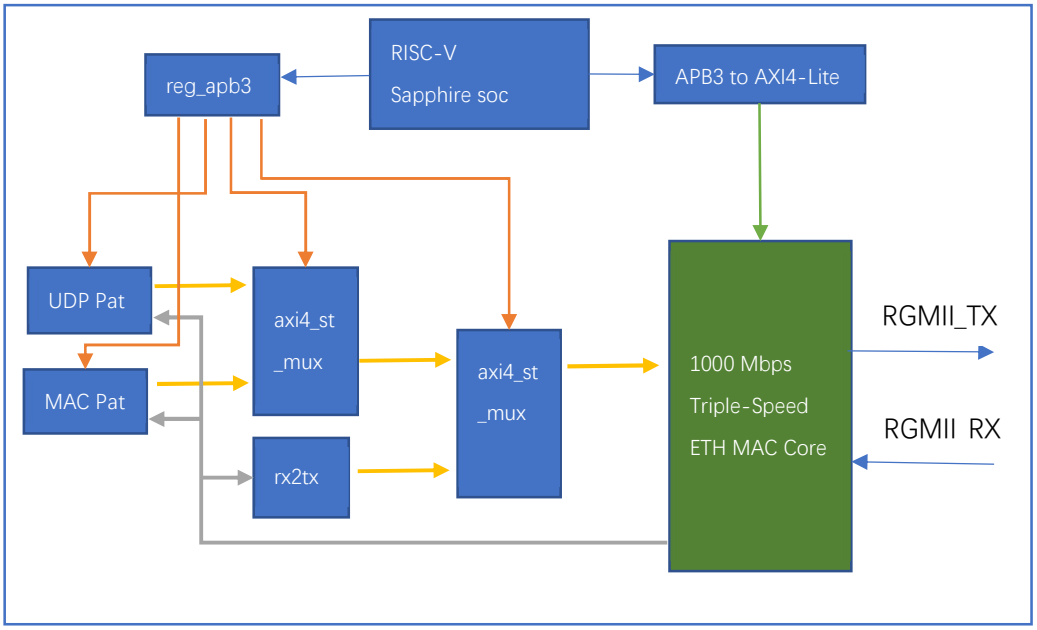
```
always @(posedge rgmii_rxc)
begin
    if(eth_speed[2] == 1'b1)//1000M
        begin
            if(rgmii_rxc_edge == 1'b0)//0:DDIO Rising Edge
                begin
                    rxd[7:4] <= rgmii_rxd_LO;
                    rxd[3:0] <= rgmii_rxd_HI_d1;
                    rx_dv <= rgmii_rx_ctl_HI_d1;
                    rx_er <= rgmii_rx_ctl_HI_d1^rgmii_rx_ctl_LO;
                end
            else//1:DDIO Falling Edge
                begin
                    rxd[7:4] <= rgmii_rxd_HI;
                    rxd[3:0] <= rgmii_rxd_LO;
                    rx_dv <= rgmii_rx_ctl_LO;
                    rx_er <= rgmii_rx_ctl_LO^rgmii_rx_ctl_HI;
                end
        end
```

如下，只需要修改 rgmii_rxc_edge 的参数即可。

```
if (PHY_INTF_MODE == 0) begin
    rgmii_if_c4a86133db2049a897441b7bae1045b1 u_rgmii_if
    (
    //Globle Signals
        .tx_mac_aclk                    (tx_mac_aclk                    ),
        .tx_mac_aclk_en                 (tx_mac_aclk_en                 ),
        .tx_reset                       (mac_reset || proto_reset       ),
        .rx_mac_aclk                    (rx_mac_aclk                    ),
        .rx_mac_aclk_en                 (rx_mac_aclk_en                 ),
        .rx_reset                       (mac_reset || proto_reset       ),
    //Configuration Signals
        .eth_speed                      (eth_speed                      ),
        .rgmii_rxc_edge                 (1'b0                           ),
        .rgmii_txc_dly                  (1'b1                           ),
    //GMII Interface
```

5、example

5.1demo 框图



AXI_TX 数据流

AXI_RX 数据流

TSE IP 控制

REG 控制

（1） 框图有三个测试模式 UDP pattern，MAC pattern 和 rx2tx 环出。其中 UDP 和 MAC 称为 Normal Mode.而 rx2tx 称为 Linked-Partner Test

（2） 在 RISCV 的 tseDemo 工程中，在 tseDemo.h 中，通过修改 TEST_MODE 和 PAT_TYPE 来切换几种测试模式。

	type		
	Udp	Mac	Rx2tx
TEST_MODE	0	0	1
PAT_TYPE	0	1	X

RISCV 通过 APB3 接口来配置相应的参数。该端口对应 reg_apb3 解析模块。

addr	discription
0x200	bits[0]:mac_sw_rst 复位，对除 RISCV 以外的所有模块复位。 bits[31:0]:NC
0x204	bits[0]:axi4_st_mux_select 0: pattern; 1: rx2tx loopbaclk bits[1]:pat_mux_select 0:udp pattern 1:mac pattern
0x208	bits[0]:udp_pat_gen_en 使能 udp_pat_gen 模块 bits[1]:mac_pat_gen_en 使能 mac_pat_gen 模块
0x20c	bits[15:0]:pat_gen_num udp 和 mac 生成模块的数据长度 bits[31:16]:pat_gen_ipg udp 和 mac 模块的包间隔
0x210	bits[31:0]:pat_dst_mac[31:0] udp 和 mac patten 的 dst mac
0x214	bits[15:0]:pat_dst_mac[47:32] udp 和 mac patten 的 dst mac
0x218	bits[31:0]:pat_src_mac[31:0] udp 和 mac pattern 的 src mac
0x21c	bits[15:0]:pat_src_mac[47:32] udp 和 mac pattern 的 src mac
0x220	bits[15:0]:pat_mac_dlen mac pattern 的数据长度
0x224	bits[31:0]:pat_src_ip udp pattern 的源 IP
0x228	bits[31:0]:pat_dst_ip udp pattern 的目地 IP
0x22c	bits[15:0]:pat_src_port UDP pattern 的源端口号 bits[31:16]:pat_dst_port UDP pattern 的目的端口号
0x230	bits[15:0]:pat_udp_dlen UDP pattern 的数据长度

5.2 仿真

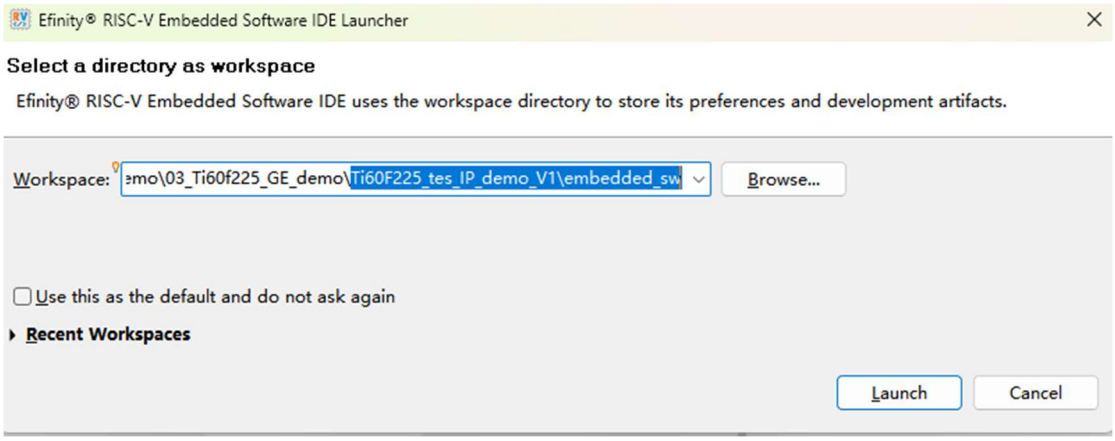
5.3 Demo 测试

（1） 编译程序，并配置 FPGA

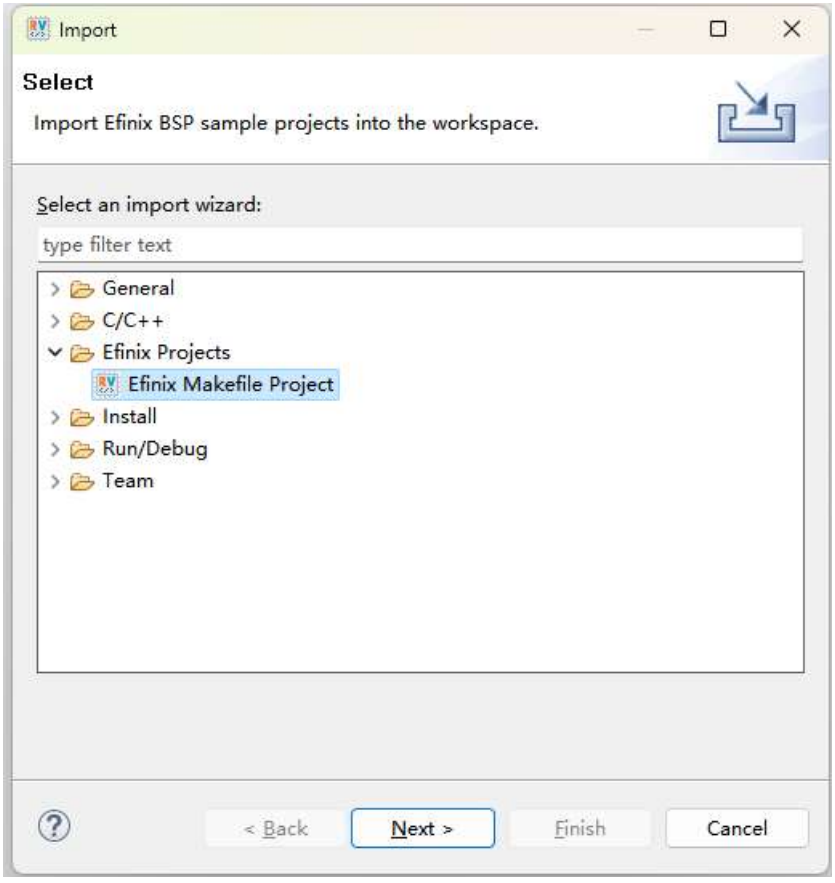
（2） 由于 RISCV 部分只使用了片上 RAM，这会让操作简单很多。把 hex 文件添加到 RISCV 中，这样每次把 hex 或者 bit 文件配置 FPGA 软核都会自动运行起来。

（3） 通过 RISCV IDE 编译 soc 程序，该过程需要配置测试模式

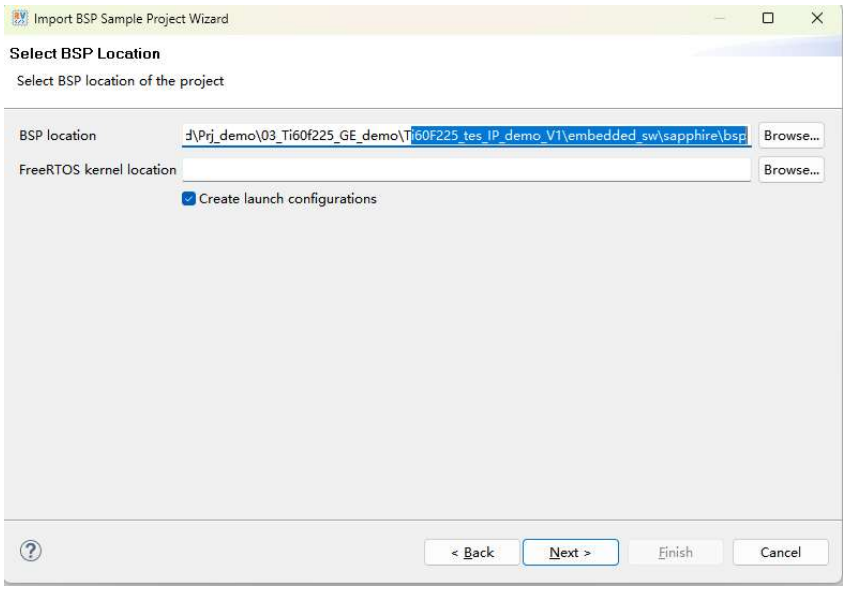
启动 RISCV IDE,路径如下:



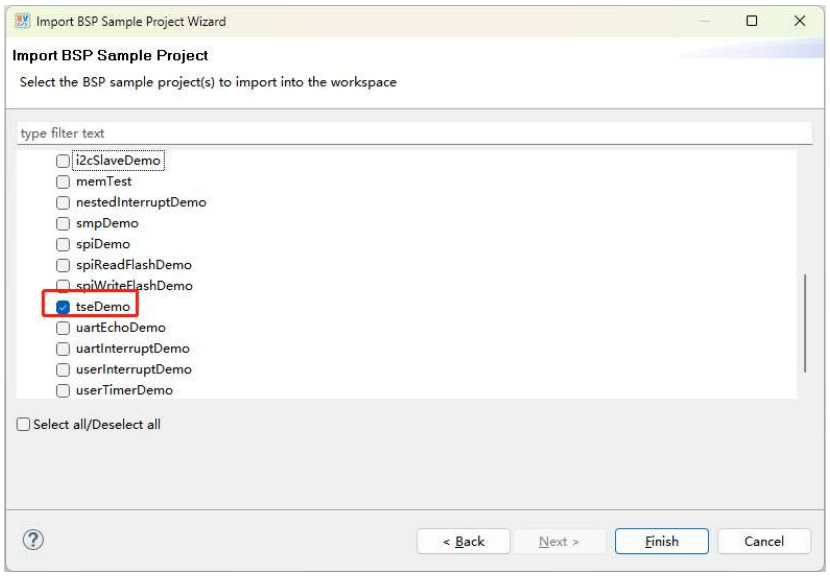
File→ import → Efinix Projects → Efinix Makefile Project



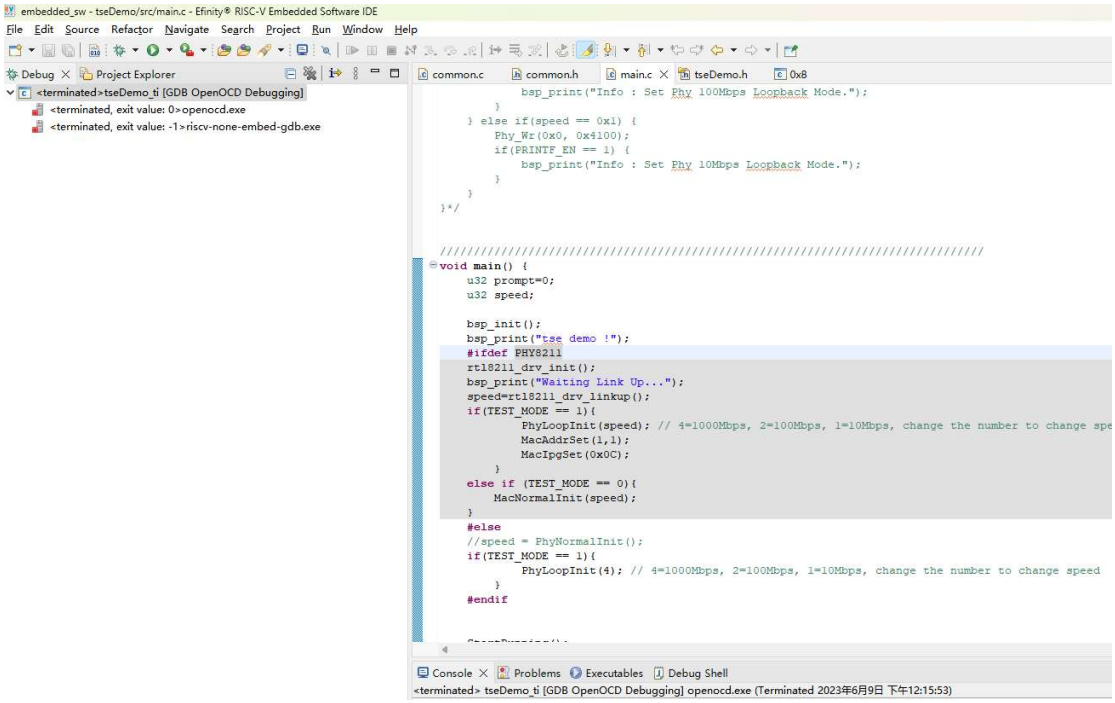
选择 bsp 位置



勾选 tesDemo



打开工程如下



在头文件 tesDemo.h 中修改测试模式，如下图，默认为 UDP.

```
#define PAT_NUM             0
#define PAT_DLEN            100
#define PAT_IPG             100
#define PAT_TYPE            0               //0 : UDP Pattern;
                                             //1 : MAC Pattern;
```

Normal Mode Test

MAC/UDP 测试模式，由本地产生数据发送给电脑。

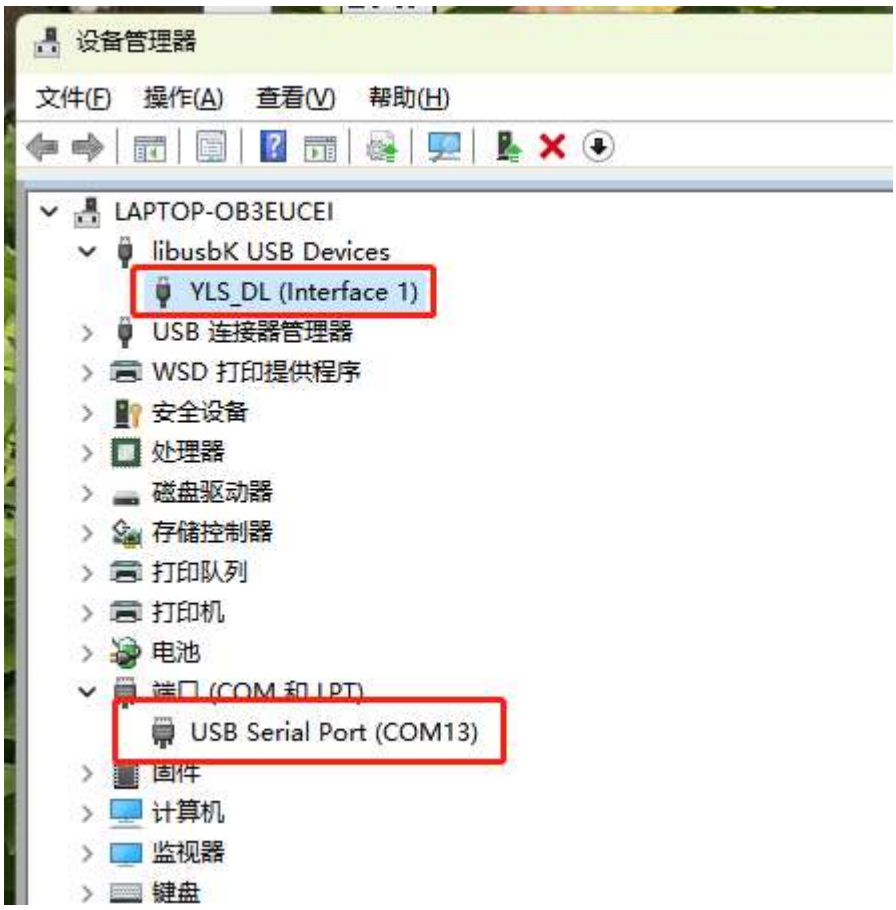
宏参数

宏名	说明
PAT_NUM	发送包数。无限数据包传输用 0 表示。
PAT_DLEN	UDP/MAC 模式发送报文的长度。

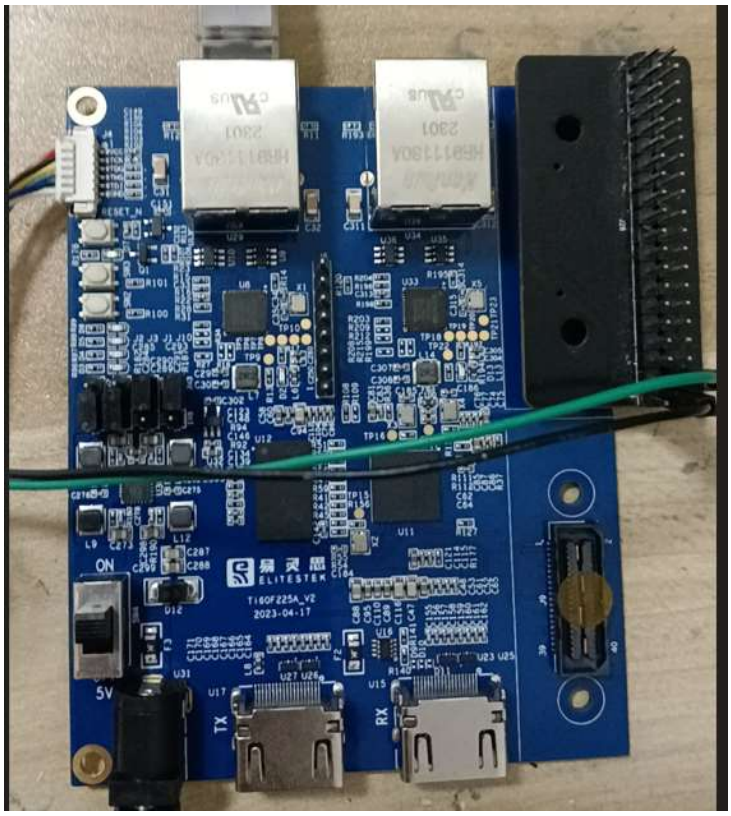
PAT_IPG	包间间隙数。一个包间间隔为 8ns。
PAT_TYPE	发送包模式的类型。 1： MAC 模式 0： UDP 模式

通过串口打印信息

YLS_DL 下载器采用的是 FT2232 方案，其中 interface0 是 SPI，只要不安装相应的驱动，就可以用作 UART，如下图，只安装了 interface1 驱动用于 JTAG。串口的 COM13。



程序里面 UART 的 IO 已经设置好，连接方式如下，J7 座子的 38，40pin。

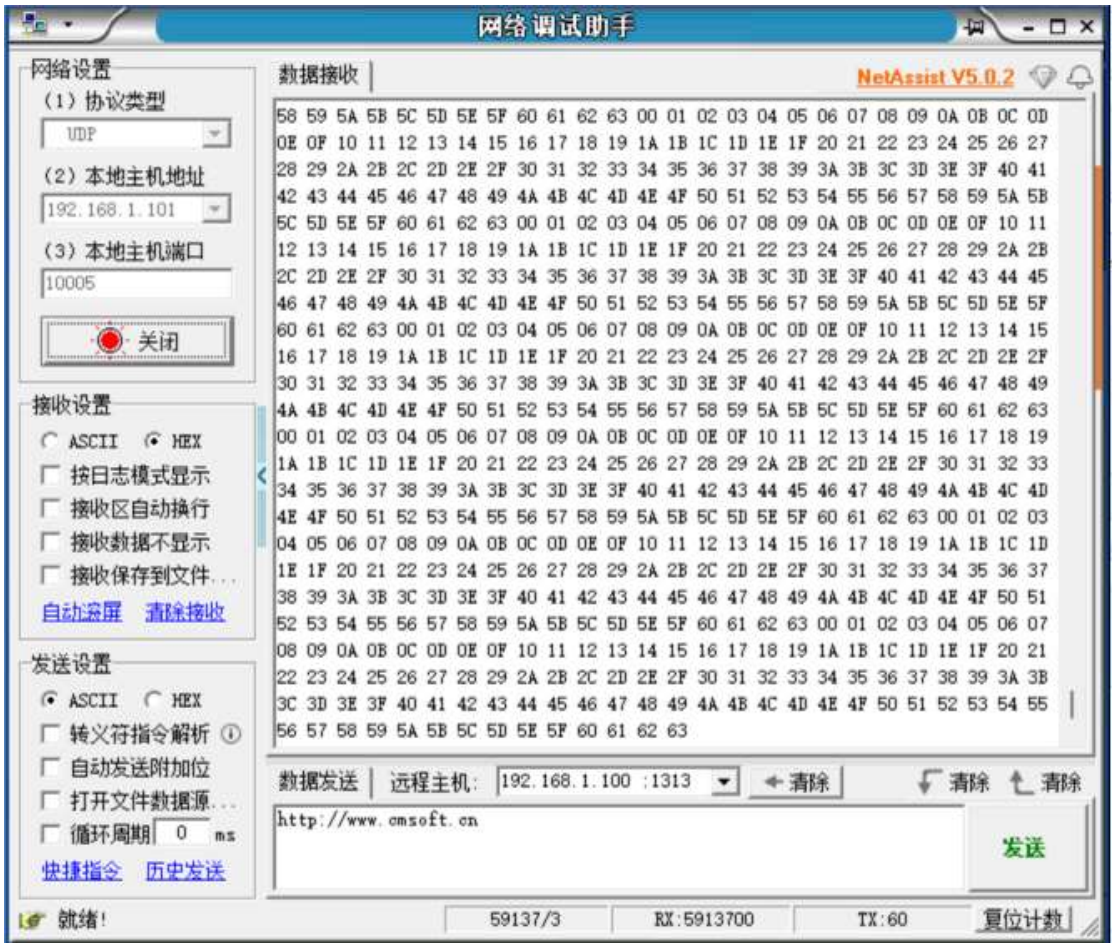


串口打印信息如下

也可以通过查看网品调试助手来看。注意要设置正确的端口号，在 tesDemo.h 中定义的端口号 SRC_PORT 是指 FPGA 端的端口号。

```
#define SRC_PORT          0x0521// (1313)
#define DST_PORT          0x2715// (10005)

#define SRC_IP                0xc0a80164(192.168.1.100)
#define DST_IP                0xc0a80165(192.168.1.101)
```



速率

左侧绿灯闪是 1000Mbps,右侧黄灯闪认为是 100Mbps。

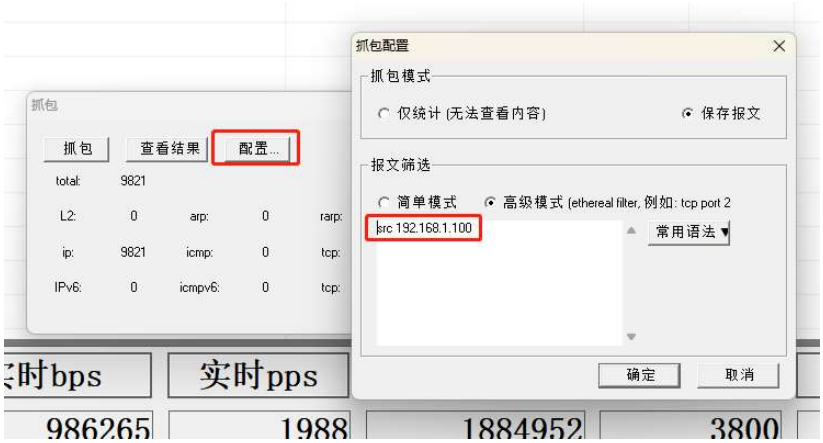
Linked-Partner Test

发送端由电脑发送。AXI4-ST 环回。MAC/UDP 模型关闭。

抓包设置

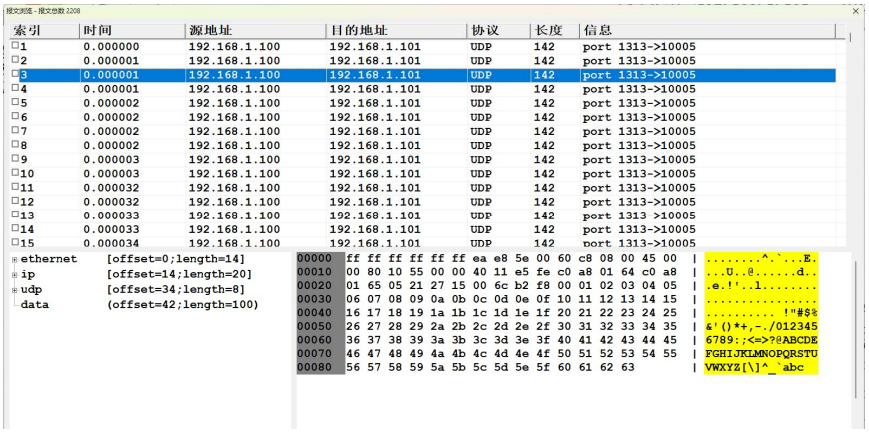
这里通过【小兵以太网测试仪】来查看。

点击【独立抓包】→【配置】，设置源 IP，目前 FPGA 配置的是 192.168.1.100.



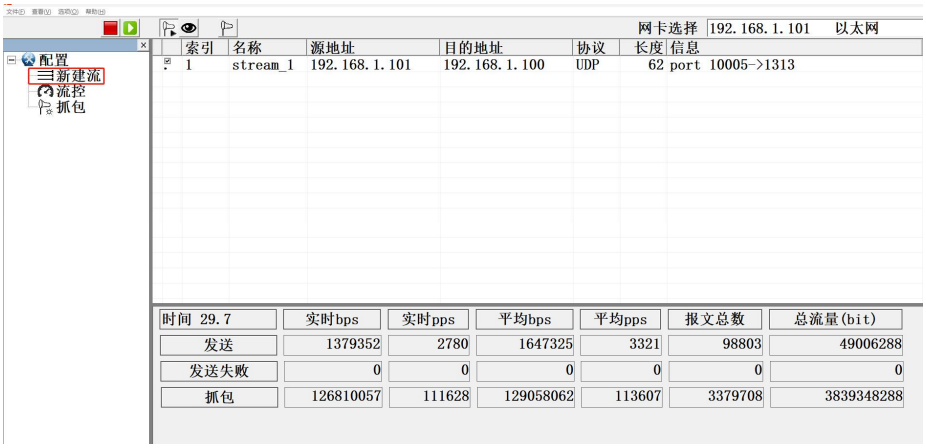
点击【抓包】，然后【查看结果】

点击查看数据。

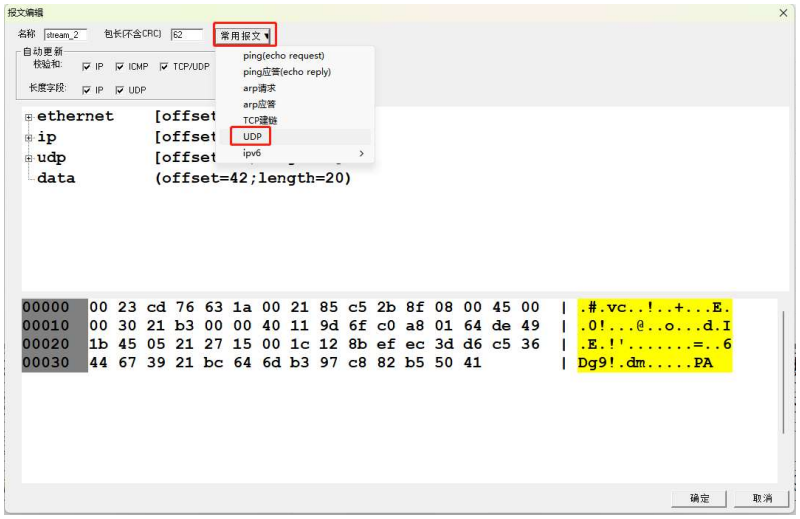


发包设置

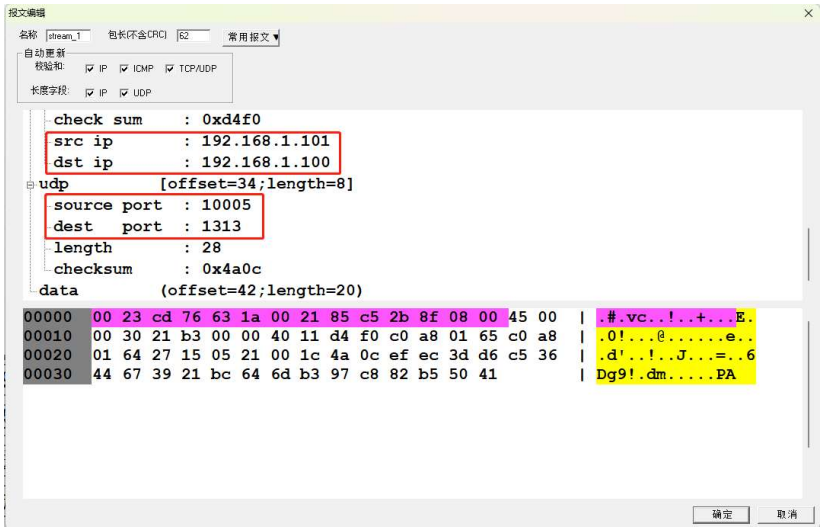
新建数据数据流。



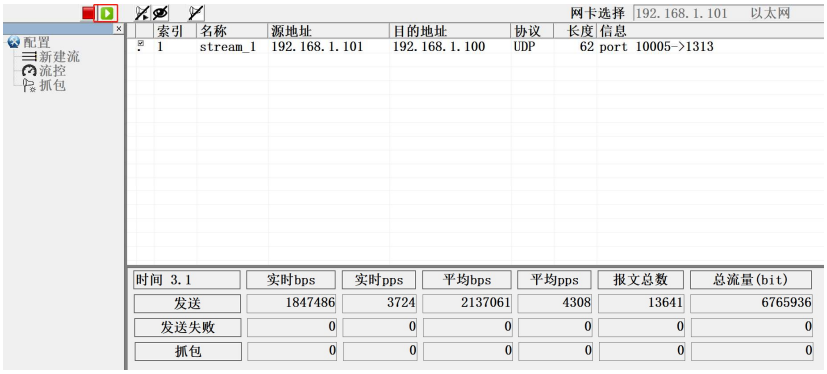
选择【常用报文】→UDP.



然后设置好 IP 和端口。

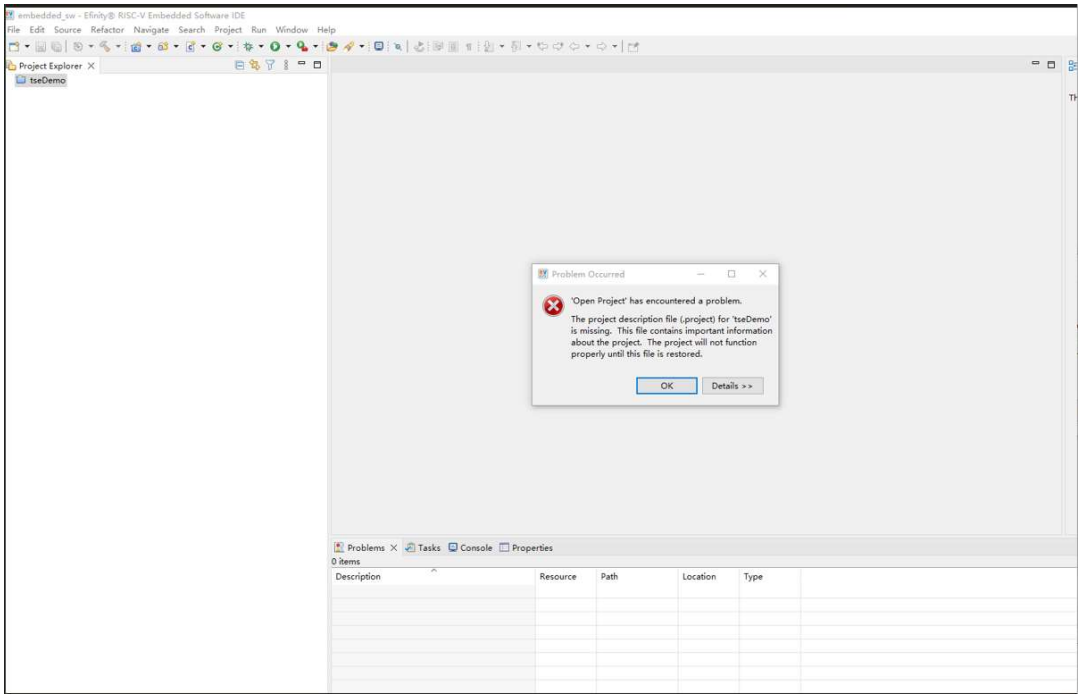

报文编辑
名称 stream_1
包长(不含CRC) 62
常用报文
自动更新
校验和: IP ICMP TCP/UDP
长度字段: IP UDP
check sum : 0xd4f0
src ip : 192.168.1.101
dst ip : 192.168.1.100
udp [offset=34;length=8]
source port : 10005
dest port : 1313
length : 28
checksum : 0x4a0c
data (offset=42;length=20)
00000 00 23 cd 76 63 1a 00 21 85 c5 2b 8f 08 00 45 00 | .#.vc..!..+...E.
00010 00 30 21 b3 00 00 40 11 d4 f0 c0 a8 01 65 c0 a8 | .0!...@.....e..
00020 01 64 27 15 05 21 00 1c 4a 0c ef ec 3d d6 c5 36 | .d'..!..J...=..6
00030 44 67 39 21 bc 64 6d b3 97 c8 82 b5 50 41 | Dg9!.dm.....PA
确定
取消

点击开始发包。


网卡选择 192.168.1.101 以太网
配置
新建流
流控
抓包
索引 名称 源地址 目的地址 协议 长度 信息
1 stream_1 192.168.1.101 192.168.1.100 UDP 62 port 10005->1313
时间 3.1 实时bps 实时pps 平均bps 平均pps 报文总数 总流量(bit)
发送 1847486 3724 2137061 4308 13641 6765936
发送失败 0 0 0 0 0 0
抓包 0 0 0 0 0 0

常见问题总结

（1）按 Demo 说明打开 ide 以后报这个错是什么情况呢（全局搜索可以在 workspace 路径里搜到.project）



这是因为原来的路路径不一样了，要右键删除下面这个文件夹，然后再重新 import。

(2)