Edge Wang

[电子邮件地址]

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。]

UDP以太网测试

[文档副标题]

目录

[概述 2](#_Toc146144246)

[1、Ethernet\_II 帧格式 2](#_Toc146144247)

[2、IP 首部数据报格式 4](#_Toc146144248)

[2.2分片机制 示例 : 6](#_Toc146144249)

[3、UDP 协议 7](#_Toc146144250)

[3.1 UDP 协议的报头格式： 8](#_Toc146144251)

[3.2 UDP校验和计算 9](#_Toc146144252)

[3.3 差错检验 10](#_Toc146144253)

[IO配置 10](#_Toc146144254)

[IO管脚分配 10](#_Toc146144255)

[RX IO配置 10](#_Toc146144256)

[TX IO 配置 11](#_Toc146144257)

[Waveform 12](#_Toc146144258)

[5、example 14](#_Toc146144259)

[5.1demo框图 15](#_Toc146144260)

[5.2 仿真 17](#_Toc146144261)

[5.3 Demo测试 17](#_Toc146144262)

[常见问题总结 28](#_Toc146144263)

## 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。

图示

低可信度描述已自动生成

**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。

图表

描述已自动生成

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

表格

描述已自动生成

**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 ;

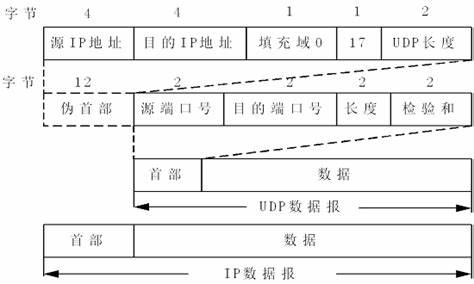
片偏移量是从数据部分开始计数 , 数据部分的开始位置是 0字节 , 其单位是 8字节 , 片偏移量 1代表 8字节 ;

## 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 | ​NO |

### 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。

图示

描述已自动生成

图1

如果接收端RXC与接收的第一个数据是下降沿对齐。

图示

描述已自动生成

图2

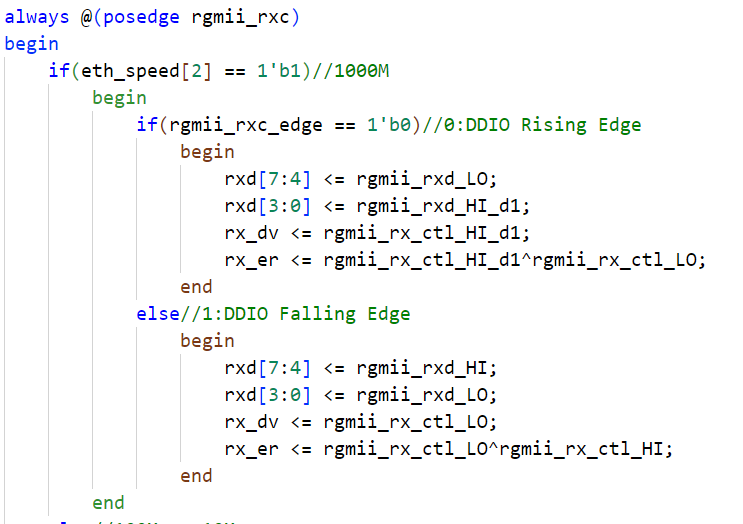
下图是我们抓取的一个数据包头。从前面我们知道前导码是7个0x55和一上0xD5。但是我们发现是rgmii\_rxd\_HI比rgmii\_rxd\_LO提前一拍，并且”D”在rgmii\_rxd\_LO上。它符合上面提到的RXC的上升沿RXD的第一个数据的对应情况。

电视游戏的萤幕截图

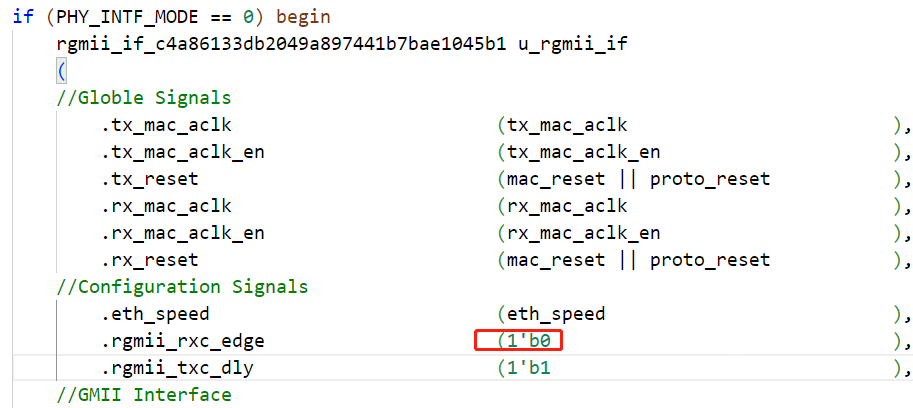
描述已自动生成

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。 |



如下，只需要修改rgmii\_rxc\_edge的参数即可。



## 5、example

### 5.1demo框图

RISC-V

Sapphire soc

UDP Pat

MAC Pat

axi4\_st\_mux

axi4\_st\_mux

1000 Mbps

Triple-Speed ETH MAC Core

rx2tx

APB3 to AXI4-Lite

reg\_apb3

RGMII\_TX

RGMII\_RX

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:

图形用户界面, 文本, 应用程序

描述已自动生成

File🡪 import 🡪 Efinix Projects 🡪 Efinix Makefile Project

图形用户界面, 文本, 应用程序

描述已自动生成

选择bsp位置

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

勾选tesDemo

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

打开工程如下

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在头文件tesDemo.h中修改测试模式，如下图，默认为UDP.

图片包含 图形用户界面

描述已自动生成

Normal Mode Test

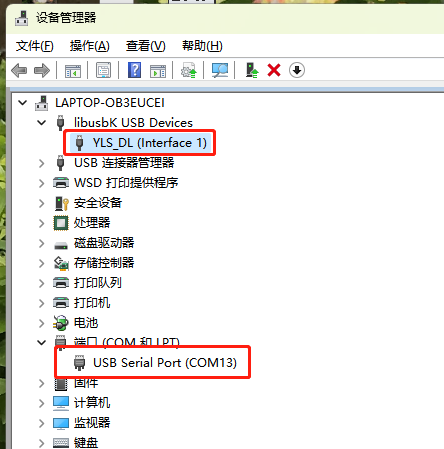
MAC/UDP测试模式，由本地产生数据发送给电脑。

宏参数

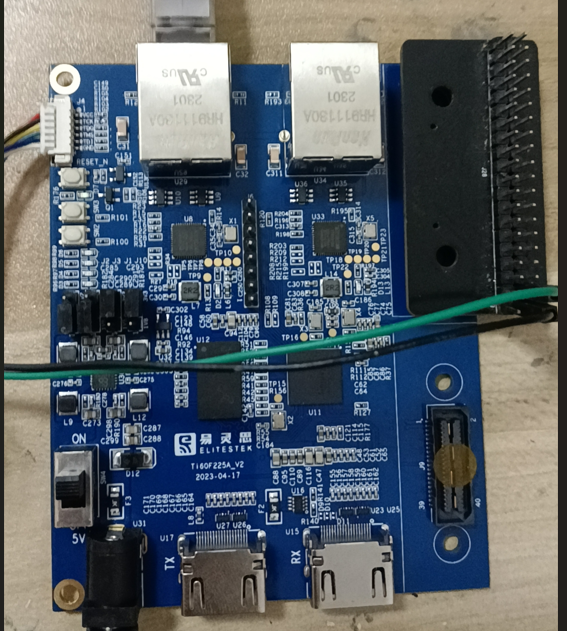
|  |  |
| --- | --- |
| 宏名 | 说明 |
| PAT\_NUM | 发送包数。无限数据包传输用0表示。 |
| PAT\_DLEN | UDP/MAC模式发送报文的长度。 |
| PAT\_IPG | 包间间隙数。一个包间间隔为8ns。 |
| PAT\_TYPE | 发送包模式的类型。  1: MAC模式  0: UDP模式 |

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

图形用户界面, 文本

描述已自动生成

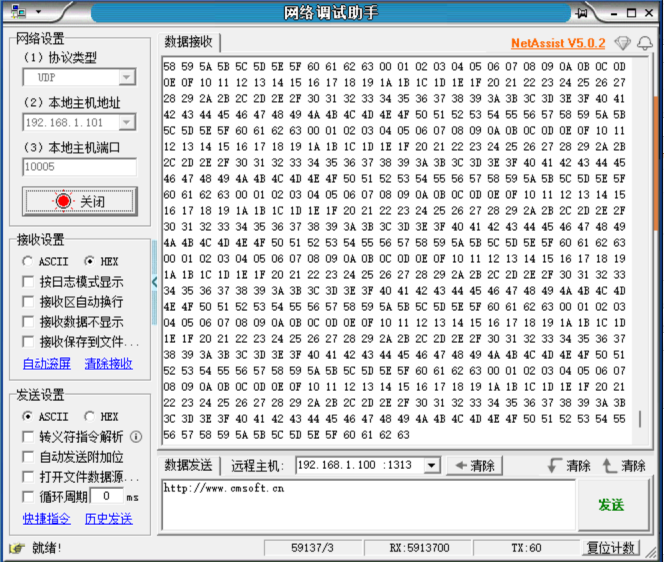
也可以通过查看网品调试助手来看。注意要设置正确的端口号，在tesDemo.h中定义的端口号SRC\_PORT是指FPGA端的端口号。

**#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.

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

点击【抓包】，然后【查看结果】



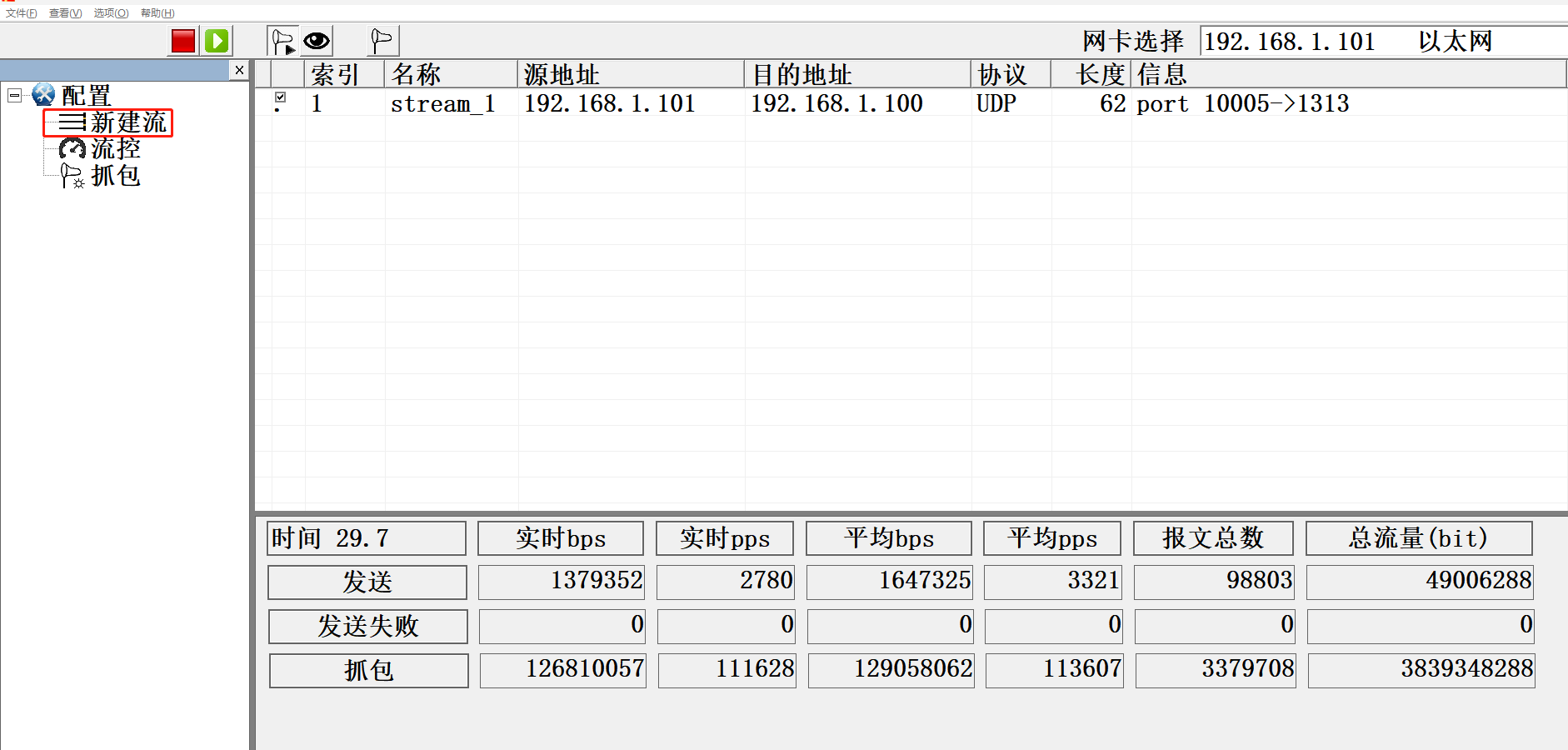
点击查看数据。

表格

描述已自动生成

**发包设置**

新建数据数据流。



选择【常用报文】🡪UDP.

图片包含 表格

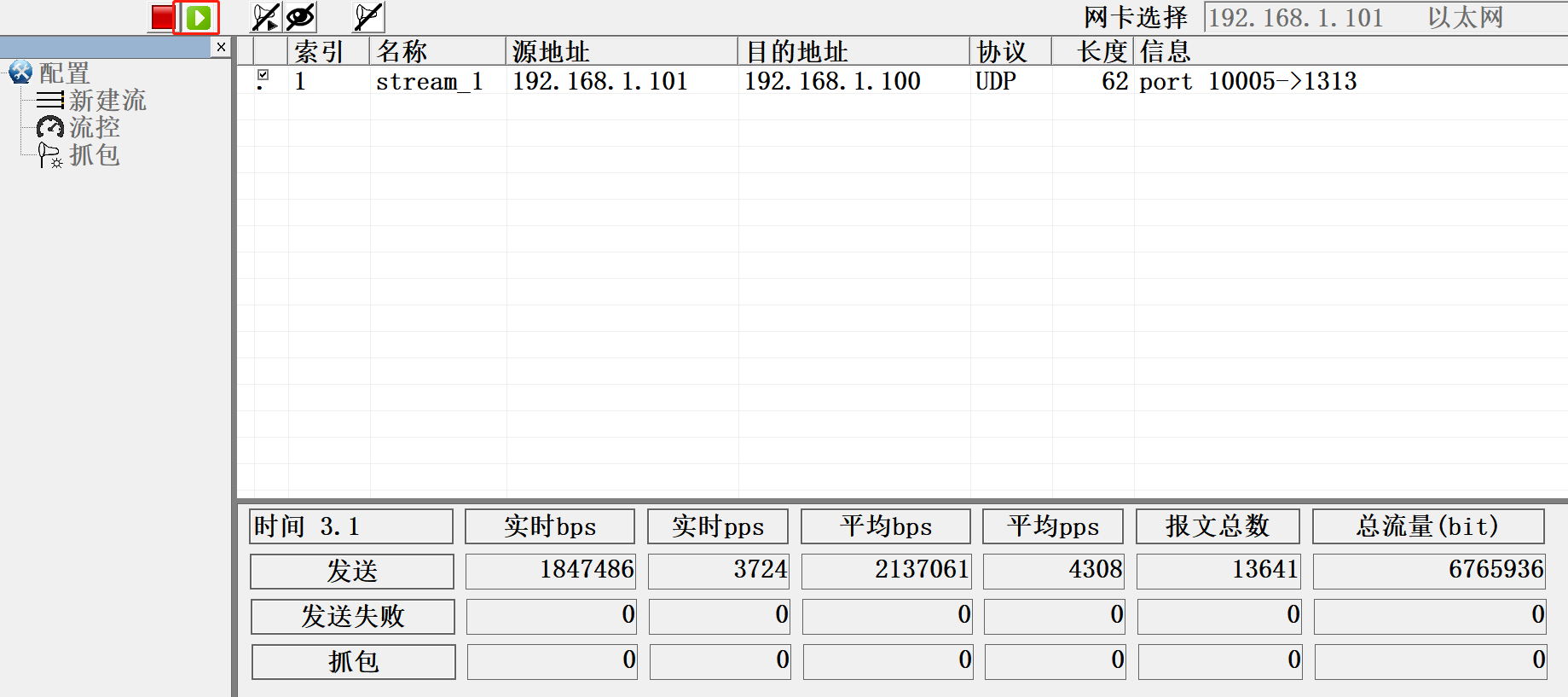
描述已自动生成

然后设置好IP和端口。

图片包含 文本

描述已自动生成

点击开始发包。



## 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）

图形用户界面, 应用程序, Word

描述已自动生成

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。

图形用户界面, 文本, 应用程序

描述已自动生成

（2）