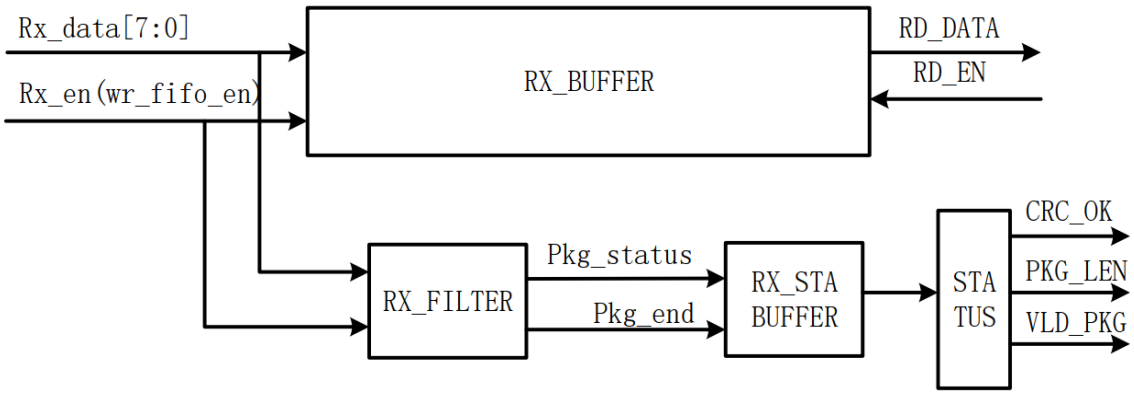
高级练习9 千兆数据接收 fifo， crc 校验和数据包过滤

一、练习内容

使用 PC 机将图像数据信息通过千兆以太网发送给 FPGA， FPGA 接收后需要对 CRC 校验和过滤无效帧传递给下一级模块 。

二、系统框图



三、设计分析

1.各模块功能分析

(1) RX_BUFFER模块

直接将读取的数据写入fifo中即可，rd_en和rd_data分别是fifo的写使能和写入数据。

(2) RX_FILTER模块

这里需要实现三个内容，其一是判定是否为正确发送的数据，其二是判定crc校验是否通过，其三则是判定接收数据的长度，这里都需要提取出来。

- 对于第一部分判定是否是正确发送的数据，主要从3个方面来考虑，即源端口1234，目的端口123和UDP协议字段8'h11;
- 第二部分则是通过crc校验的模块进行判定即可;
- 第三部分通过截取udp首部的字节长度来进行判定，或者更好的方式是通过rd_en下的数据长度的计数，从而准确判定其数据长度。

(3) RX_STA_BUFFER模块

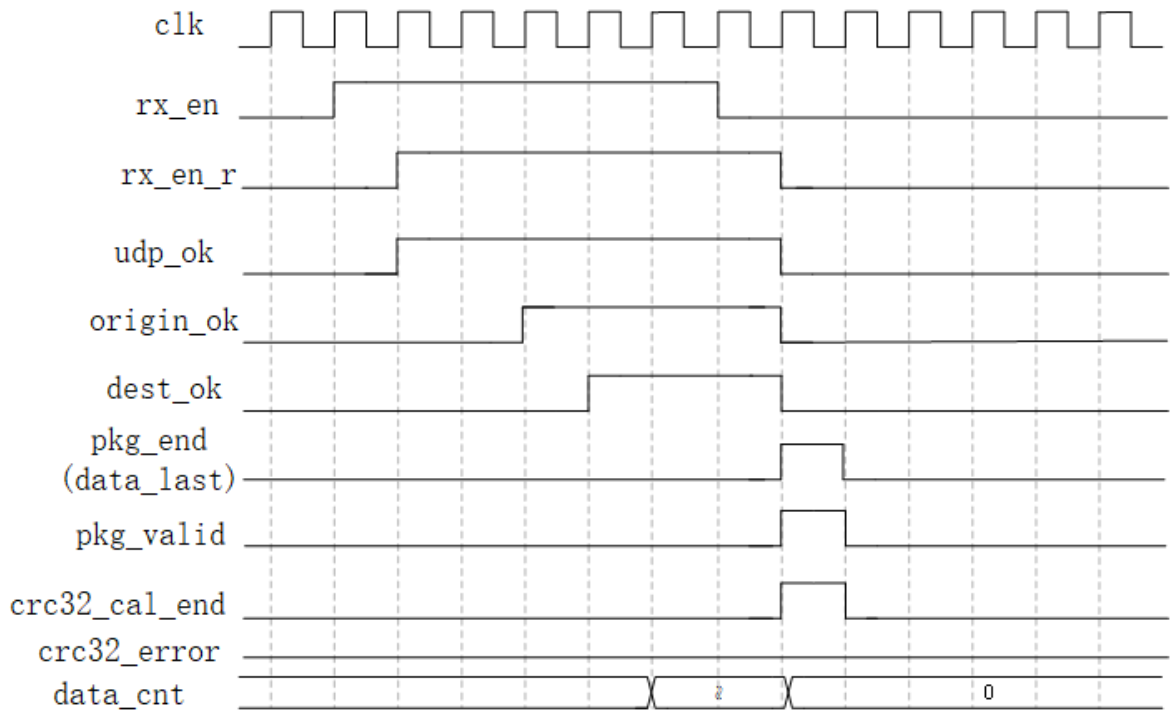
对于最后一个模块，只需要将RX_FILTER截取三个有效数据，在数据读取完毕后，写入fifo中即可。

2.RX_FILTER模块功能框图

滤除部分	实现部分	输出部分
源端口1234	第42、43字节对应的数据分别是0x04和0xD2	origin_ok
目的端口123	第44、45字节对应的数据分别是0x00和0x7B	dest_ok
UDP协议字段8'h11	第31字节对应的数据分别是0x11	udp_ok

最后的数据输出是根据这三部分输出相与来进行判定的。

3.RX_FILTER模块时序分析



总的来说，代码的编写并不是很难，关键在于时序的对应性要求高，不能有一点偏差，这是很重要的。

四、练习步骤

1.代码编写

按照时序分析编写代码即可，没有什么特别的部分，关键在于每个时序的精确性，这是很重要的，重要的事情说再多遍都无所谓。

2.ila抓取

这里需要编写matlab发送数据，使用ILA抓取，从rx_filter模块开始抓取，再依次往后类推。

matlab代码编写如下。

```
clc;
clear all;
udplink=udp('255.255.255.255','RemotePort',123,'LocalPort',1234);
udplink.OutputBufferSize=8192;%传数据buffer大小
udplink.Timeout=1000;%传输时间限制
fopen(udplink);
data=0:255;
```

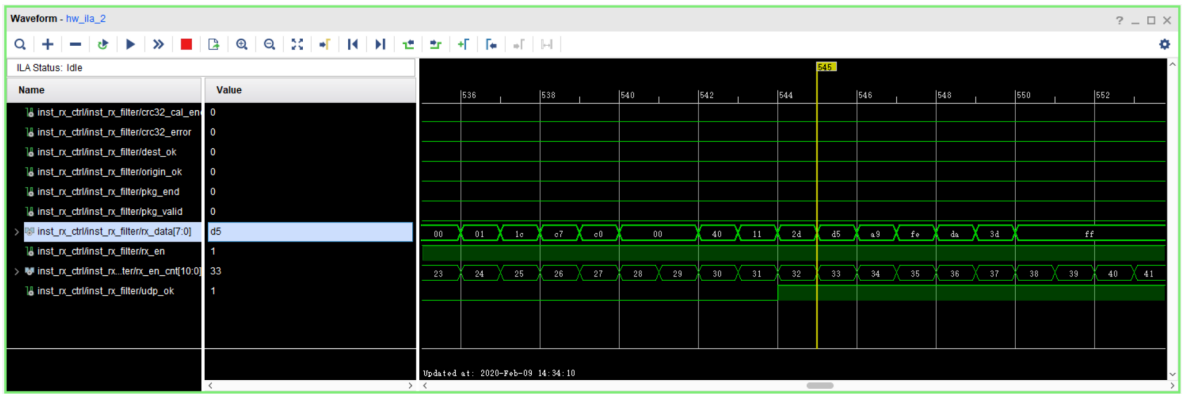
```
%data=zeros(1,256,'uint8');
for i=1:20
    fwrite(udplink,data,'uint8');
    pause(2);
end
fclose(udplink);
delete(udplink);
clear udplink;
```

五、实际波形仿真

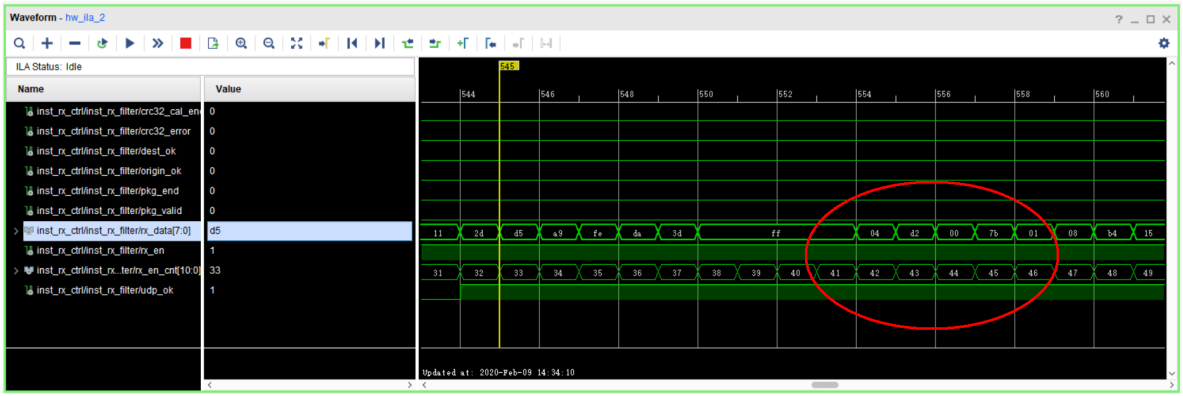
使用ILA抓取信号，可以得到下面的结果。

1.信号时序的一致性不对

- 第一个信号是udp_ok，这个信号是正确的，没有任何问题。刚好在计数到31时，检测到接收数据0x11，获得正确的协议码，故udp_ok拉高；



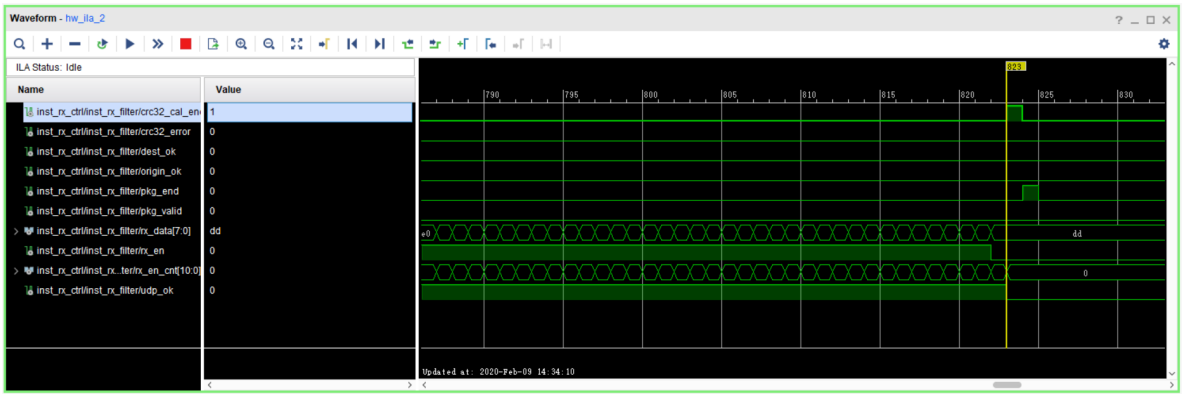
- 第二个信号是origin_ok和dest_ok，这两个信号都是错误的，没有正确的拉高；



我们发现实际上对应的数据都是正确的，当时因为是16位的数据，在原来数据的基础上延时了一个时钟周期，故需要在原来判定个数的基础上增加1，才能得到正确的答案。

```
21  localparam  HEAD_END_CNT    = 7,        //frame end data count
22              ORIGIN_END_CNT  = 43+1,     //origin port
23              DEST_END_CNT    = 45+1,     //destination port
24              UDP_END_CNT     = 31,       //udp protocol
25              DATA_START_CNT  = 49;
```

- 第三个信号则是pkg_end，发现没有与crc32_cal_end对齐，这也是这个时序上的不一致性；

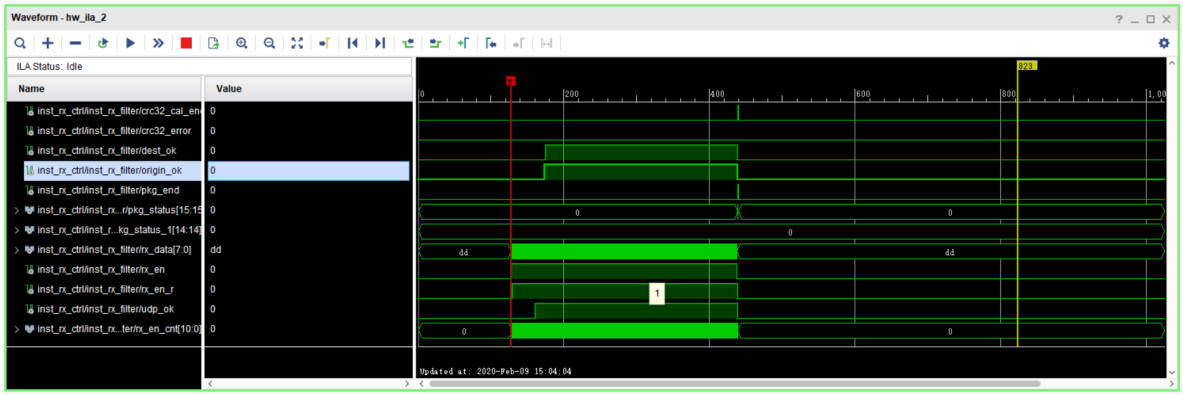


具体可以参考crc32校验的代码，从而发现问题，即pkg_end多拉了一拍。

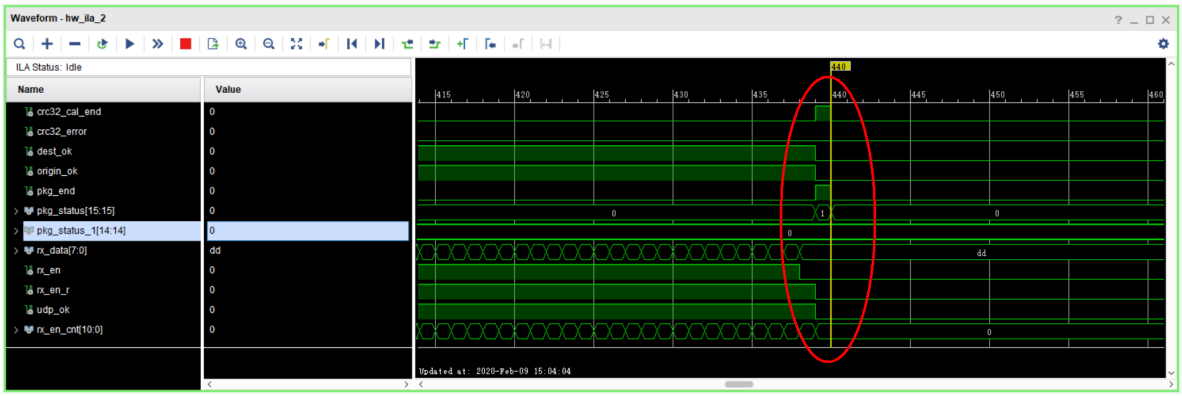
```
180   always @(posedge clk)
181   begin
182       pkg_end_r1 <= pkg_end_r;
183   end
```

重新编译，再次查看结果。

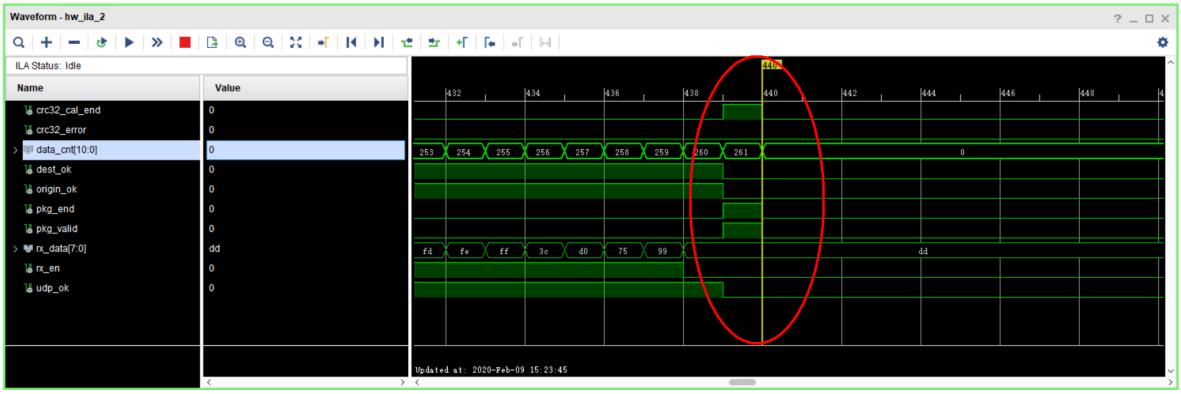
这次查看波形，origin_ok和dest_ok都获得了正确的信号，说明修改上来说是成功的。



还有crc_ok，即pkg_status[15]也获得了正确的信号，但是pkg_valid信号还是存在一定的问题，这里需要进行再次的修改，以查看结果。



还是刚才pkg_valid的延时一拍的问题，未能考虑周全。

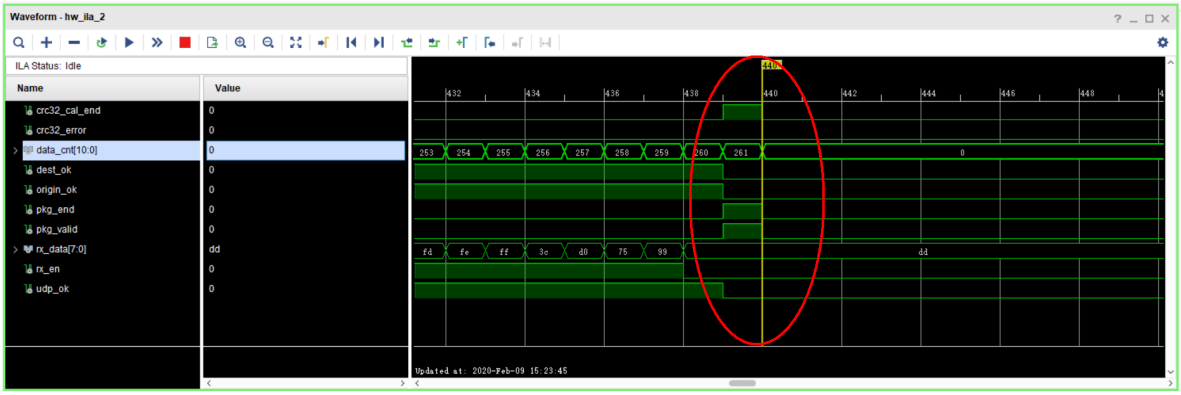


结果正确，信号的不一致性问题解决。

2.查看数据记录问题

这里主要查看的是pkg_status的数据个数的记录，通过data_cnt记录的变化进行查看。

一是内部rx_fliter模块查看。



我们看到data_cnt最后的计数是261，但是我们设置是在rx_en为0时，将数据写入pkg_status，故pkg_status可以获得正确的读数。

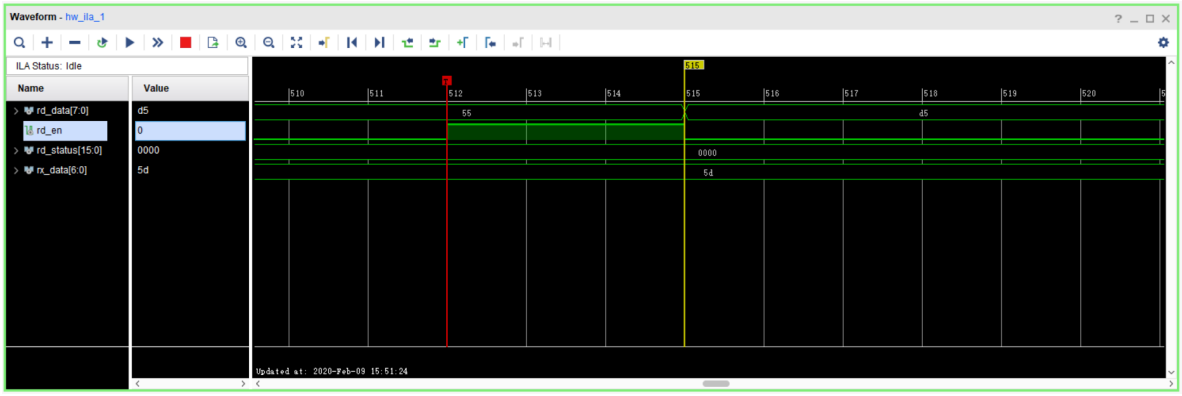
二是外部总体模块查看。

外部顶层文件中调用ILA抓取读取的数据，其中rd_en并没有被拉高，而且rd_data也不对，只有1bit数据。

这里主要是rd_en没有赋值，同时rd_data的wire变量没有定义造成的。对代码做如下修改。

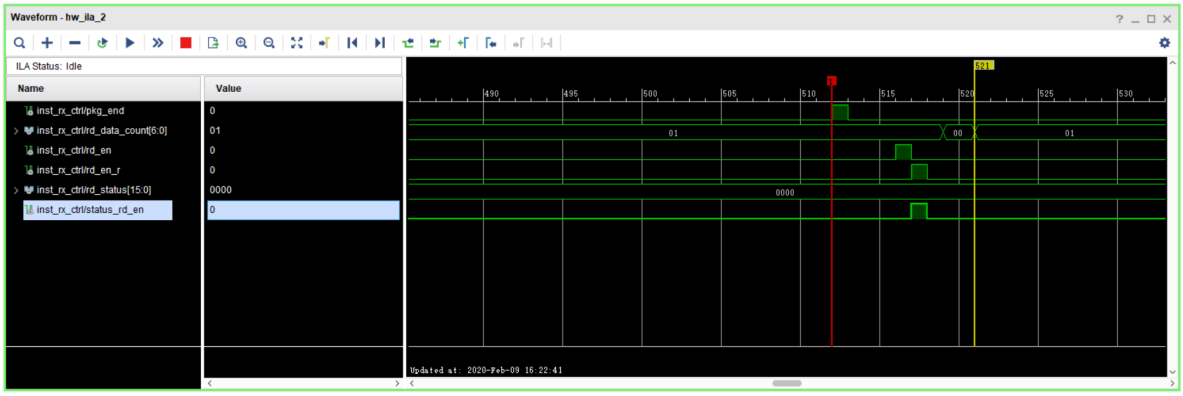
```
30  wire     [7:0]    rd_data;
31
32  assign phy_rst_n = phy_rst_cnt[18];
33  assign rst = ~rst_n;
34  assign rd_en = rd_en_r2[3];
```

重新编译后的结果如下，rd_status并没有正确读取出来，这里就需要查看rx_ctrl的时序是否正确。



发现问题，pkg_status还未正确写入，就rd_en就拉高了，自然没有任何数据，需要等待pkg_status正确输出后，再说，其实是时钟用错了。

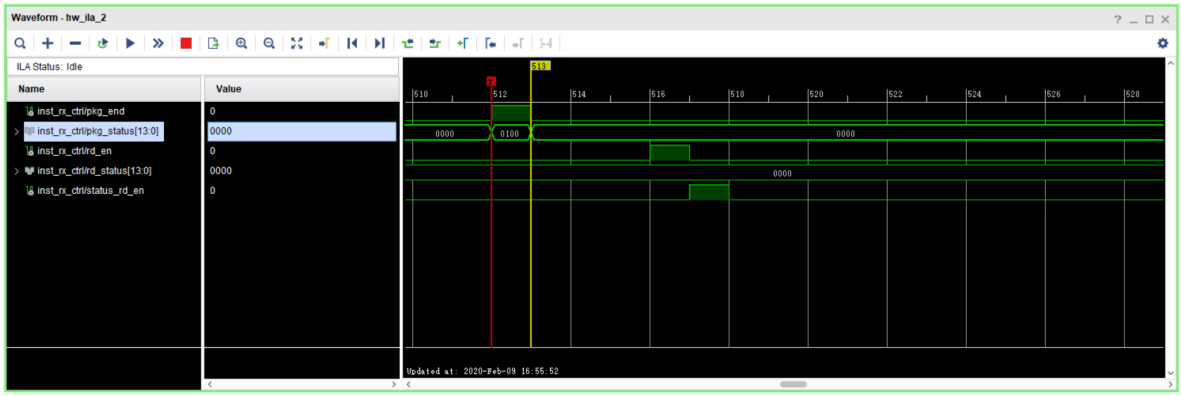
```
always @(posedge clk)
begin
    rx_en_r <= rx_en;
end

always @(posedge clk)
begin
    if (rst == 1'b1)
        rd_en_r1 <= 1'b0;
    else if (rx_en == 1'b0 && rx_en_r == 1'b1)
        rd_en_r1 <= 1'b1;
    else
        rd_en_r1 <= 1'b0;
end
always @(posedge clk)
begin
    rd_en_r2 <= {rd_en_r2[2:0], rd_en_r1};
end
```



发现并没有数据，这就说明pkg_status的数据本来就有问题，没有数据。那就追根溯源，一个个查。

编译过程中发现问题，pkg_status和pkg_end都没有定义变量，导致出错。

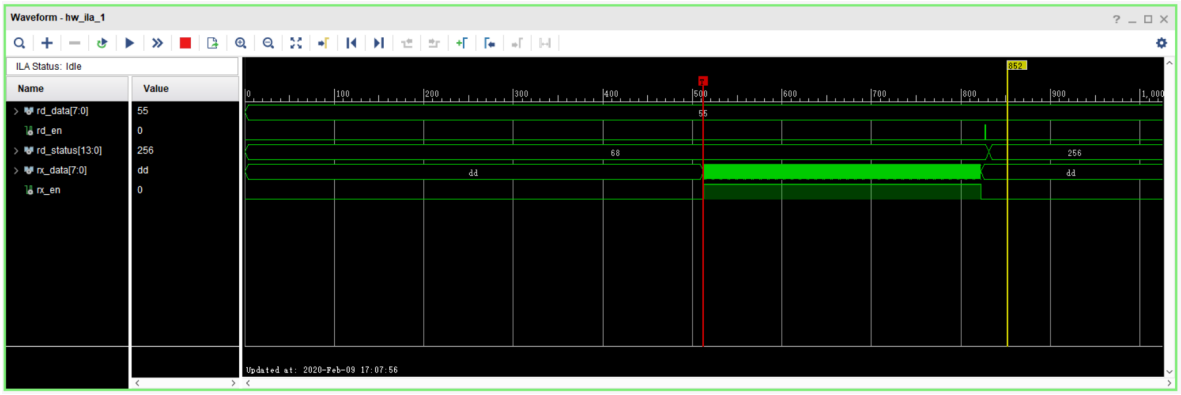
```
wire    [15:0]  pkg_status;
wire            pkg_end;
```



从上图中可以发现，数据正确的写入，但是读取出现了问题。

```
12  module rx_sta_buffer(
13      input   wire            wr_clk,
14      input   wire            rd_clk,
15      input   wire            pkg_end,
16      input   wire    [15:0]  pkg_status,
17      input   wire            status_rd_en,
18      output  wire    [7:0]   rd_status,
19      output  wire    [6:0]   rd_data_count
20      );
```

最终把问题定位在了rd_status的位数上，修改后，重新查看结果。



搞定，获得了正确的数据。

六、总结与讨论

1.关于代码的严谨性

对于下列代码，咋看没有任何问题。

```
always @(posedge clk)
begin
    if (rst == 1'b1)
        status_r <= 'd0;
    else if (rx_en == 1'b0)
        status_r <= data_cnt - 'd4;
    else
        status_r <= 'd0;
end
```

但实际上而言，status_r只需要再最后时刻输出即可，根本不需要一致改变，其他情况默认为0就行啦。这是很重要的，可以节省很多资源，这在写代码时尤其需要注意。

2.写代码过程中的几点注意事项

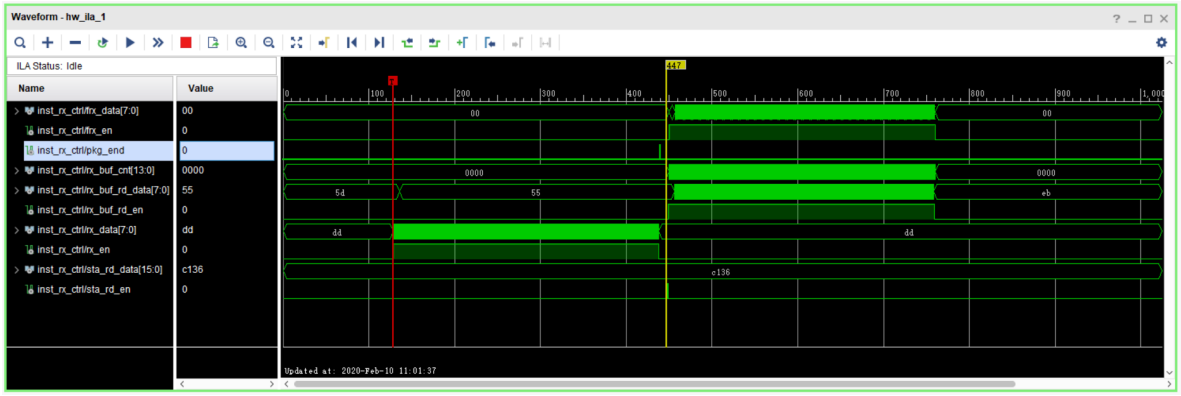
- 先分析清楚，再实现代码
- 每次使用一个变量，都要记得定义，不然，位数会出错的。除非是1bit的变量，但是最好还是不要忘记这一位，很难搞
- 不在于快，而在于正确，写得多了，自然就又快又对，这是我们的终极目标

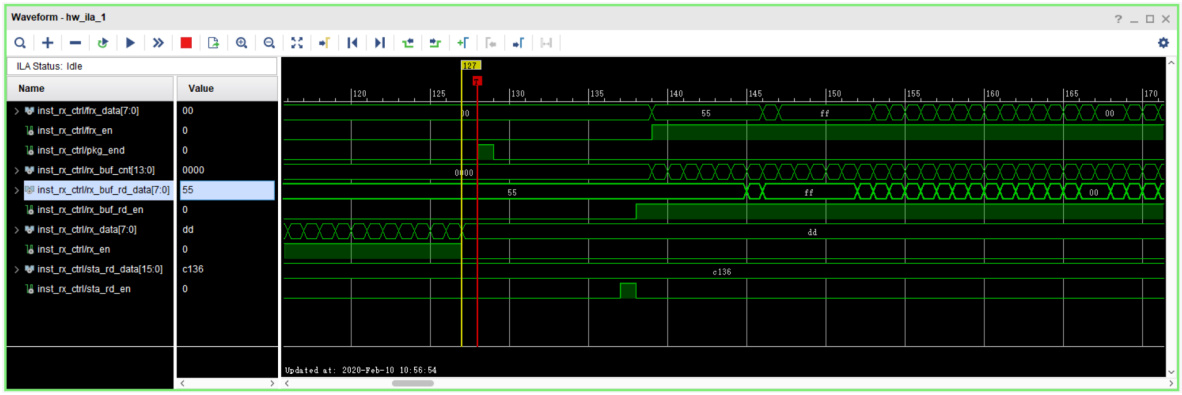
3.功能补足

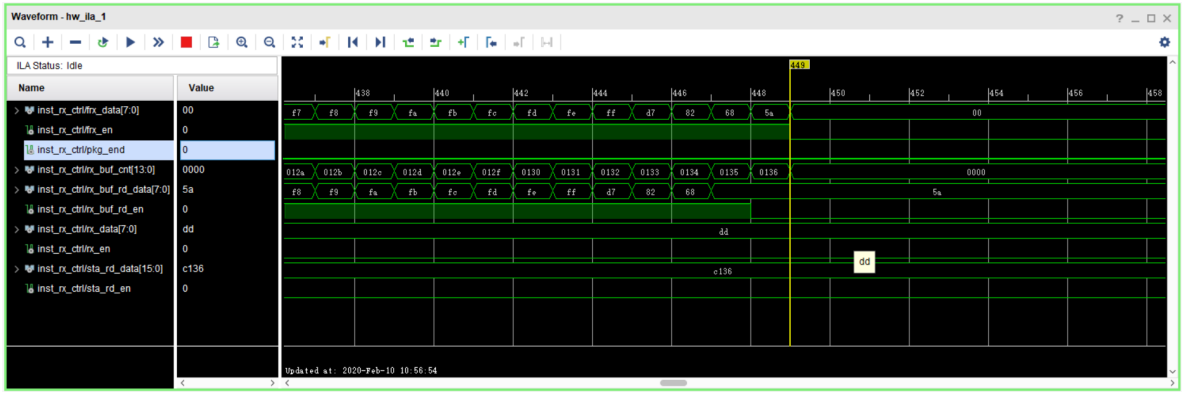
最终的数据是需要输出的，这里需要添加些东西，实际上的设计存在一些问题。

- 先将sta_buf的status读取出去
- 然后判定status中的数据，看是否接收到了有效的数据
- 最后将有效的数据输出，同时输出相应的使能位

最终得到的ILA抓取结果如下。







总的来说，从数据输入到数据输出的过程都是正确的。