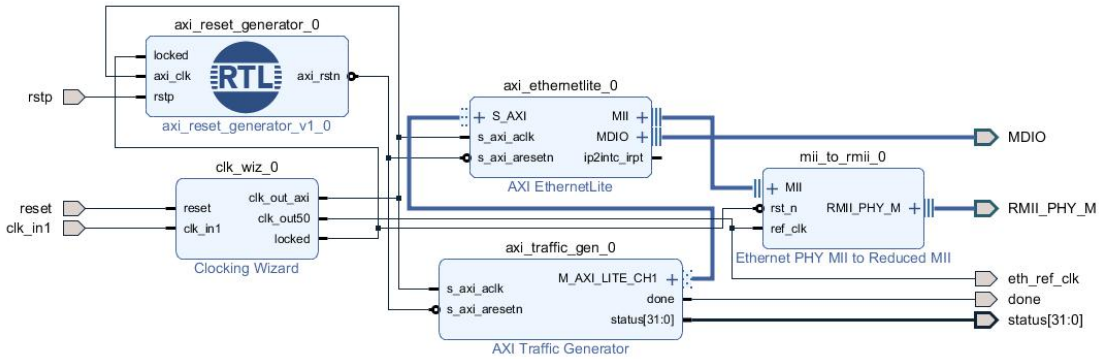
Lab 11 – 利用 FPGA 发送以太网帧

刘紫檀 PB17000232

1. 实现架构综述



a) AXI EthernetLite 是一个可以工作在 AXI4/AXI4-Lite 总线下的 MAC IP 核，该 MAC 实现提供了 PHY 操作的封装（碰撞重发，CRC 校验，接收时目标 MAC 字段判断等），并以 AXI 的 Memory-Mapped IO 形式暴露出来。在实例化本 IP 核时，选择 AXI4-Lite 总线，同时选择 Enable MII Management Interface 选项，来允许 EthernetLite 核处理 PHY 的控制端口 MDIO，具体在 IEEE 802.3ae 中有所说明（然而 Nexys4-DDR 开发板初始化时已经设置了 PHY，所以并没有用到）。

b) MII to RMII 是一个将 MII 接口转换为 RMII 接口的 IP 核。RMII 信号线数目较 MII 少，时钟频率更高，并且是 Nexys4-DDR 板载的 SMSC LAN8720a PHY 芯片使用的接口。

c) AXI Traffic Generator（下简称 ATG）是一个 AXI4/AXI4-Lite 的 Master 设备，可以进行 AXI 总线的测试。在 MII to RMII 的 IP Example Design 中，也采用了 ATG 来生成流量。

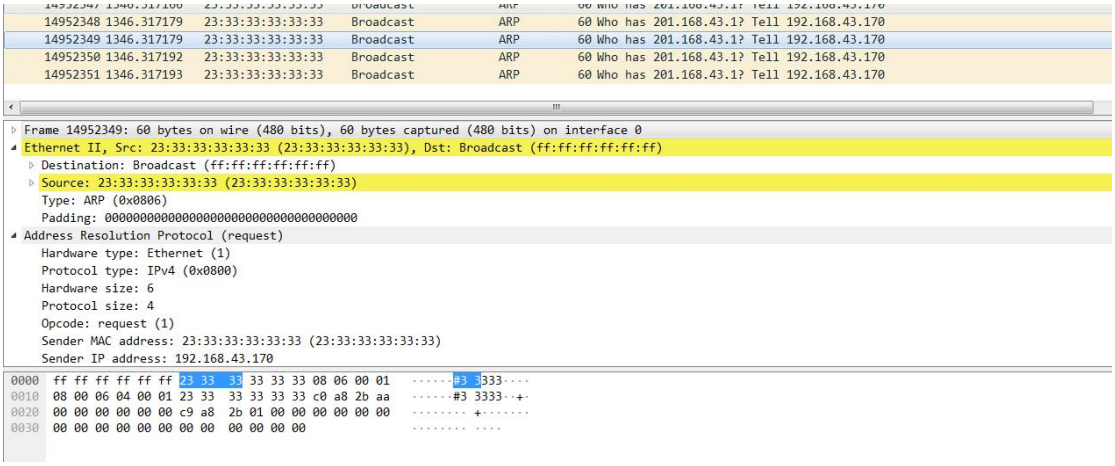
ATG 有多种模式，本实验中采用了其“System Test”模式。该模式下，ATG 模块被综合前导入的四个控制文件“addr.coe”，“ctrl.coe”，“mask.coe”，“data.coe”控制，具体见。ATG 的 status 输出展示了当前的测试状态。

d) Clocking Wizard 输出了 AXI 总线时钟（本例采用 80MHz）和用于 RMII 接口&PHY 参考时钟的 50MHz 输出。

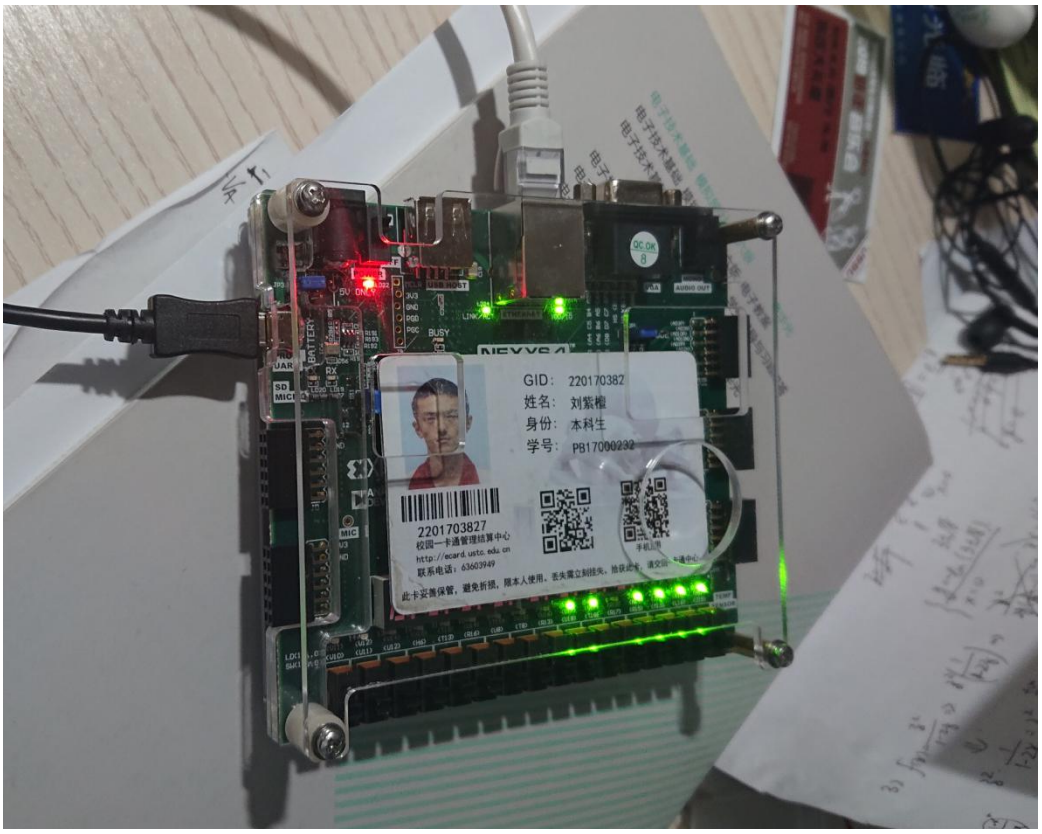
e) AXI Reset Generator 是我自己写的一个 RTL Module，用来保证异步的 rstp 信号（高电平有效）产生的 axi_rstn 信号（低电平有效）和时钟边沿同步置 1，同时保证 Clocking Wizard 输出稳定后复位再停止。

2. 功能演示

将 FPGA 与计算机的网口用双绞线连接（并不清楚是交叉线还是直通线，但是 8720a 支持 Auto MDI/MDIX，所以问题不大），并且给 FPGA 加电烧写程序，在计算机上打开 Wireshark，抓包结果如下：



可以看到有许多源 MAC 为 23:33:33:33:33:33 的 ARP Broadcast 帧。（平均速率≈6MB/s，根据几 w 个包推算的）



3. 实现细节
   a) AXI4 总线

   AXI4 总线在 ARM 的官网上有其规范文件。AXI4 有 5 个 Channel，分别是 Read Address/Control Channel、Write Address/Control Channel、Read Data Channel、Write Data Channel 和 Write Response Channel。每个 Channel 都有 VALID（Channel 的 Master 端提供）、READY（Channel 的 Slave 端提供）信号用于握手。当 VALID 和 READY 同时为高时，标志该 Channel 的一个数据传输完成。

在此基础上，文档解释了基本的读写时序。更详细的可以参考手册。截至目前，我只完成了一个 AXI4-Lite 的读取状态机。

b) IP Block Design

IP Block Design 是 Vivado 提供的一个方便的 Module/IP Core 连线界面。连好在 design_1 右键就可以弄出来 Instantiation Template，非常方便。值得一提的是，在 AXI EthernetLite 中写明的，用 Block Design 实例化 MDIO 接口会自动在实例化的 Instantiation Template 加上三态门。

c) Ethernet II 帧格式 & EthernetLite 的输入

一个 Ethernet II 帧由如下部分组成（来自 EthernetLite 的 IP Guide）：

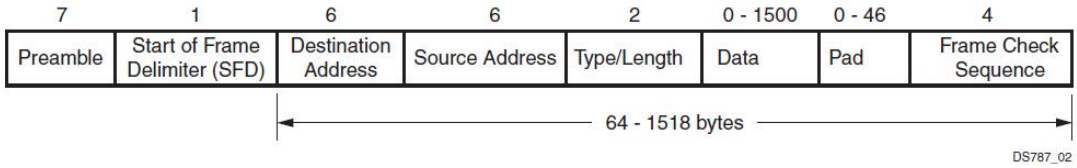


Figure 3-4: **Ethernet Data Frame**

其中，EthernetLite 已经帮我们填好了 Preamble，SFD 和（可能的）Padding，以及 FCS。剩余的部分需要通过 AXI 总线访问，在 EthernetLite 地址空间填入相应字节，最后通过在 Transmit Buffer 的 Control 寄存器(0x07FC)中的最低位写入 1 触发发送：

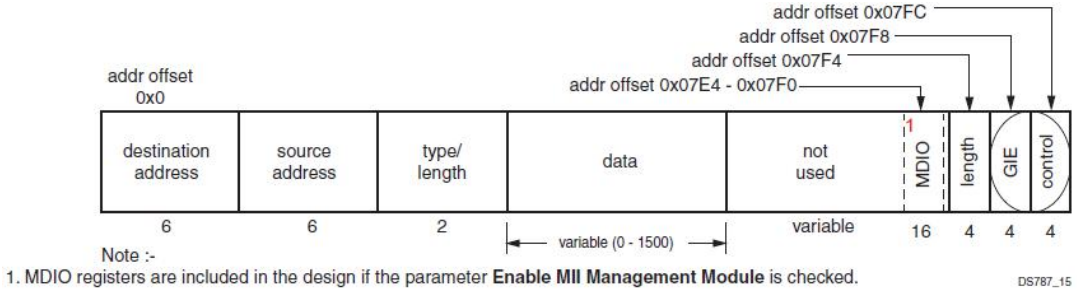


Figure 3-2: **Transmit Dual Port Memory**

接收部分大抵相同，只是地址空间从 0x1000 开始，同时可以打开中断免得轮询 Recv。Buffer 的 control（0x17FC）。接收在本实验中没有涉及。

d) 配置 ATG

在 System Test 模式下，ATG 通过读取 ctrl.coe 中的指令来进行总线读/写操作，同时在成功/失败时分支跳转。相关的手册页（pg125-axi-traffic-gen.pdf，p56）：

AXI4-Lite Protocol

System Init/Test mode allows you to generate the AXI4 transaction on the master interface. Transactions are generated based on the configuration file you provided. When core generates all transactions, it asserts the `done` and `status` signals indicating the status of the generation.

- **Transaction Depth** – Maximum number of address and data entries supported in COE file. Available transaction depth are 16, 32, 64, 128, and 256.
- **Data COE File** – Loads/creates the data COE file. Contains data entries for the corresponding address in the Address COE file.
- **Address COE File** – Loads/creates the address COE file. Contains address entries for the transactions to be generated on the master interface. End of the transaction is defined by NOP (`0xFFFFFFFF`). The core stops generating any further transaction after processing NOP.

  Note: In IP integrator, the addresses in COE files should be updated based on address space allocated by the tool.
- **Control COE File** – Loads/creates the control COE file. Contains the control information of type of transaction to be generated, next COE entry to fetched, and to count if any errors occurred.
- **Mask COE File** – Loads/creates the Mask COE File. Contains the Mask bits to be used during read data comparison.
- **Mode**
  - **System Init** – Generates write transactions.
  - **System Test** – Generates write and read transactions.

需要注意的是，每一个 COE 文件都是 32bit-32bit 互相对应的。ATG 把每一个 32bits 视为一个 Entry，如果 Control COE 写了要 Read，ATG 就会到对应的 Address COE 的对应 Entry 找到地址发起一个 Read，同时把读到的数据和对应的 Mask COE Entry 做与运算之后和 Data COE 比较，如果成功/失败就跳转（写在了 Control COE 里面）。Control COE 格式见下：

Table 1-14: **Control COE File – 32-bit Control information**

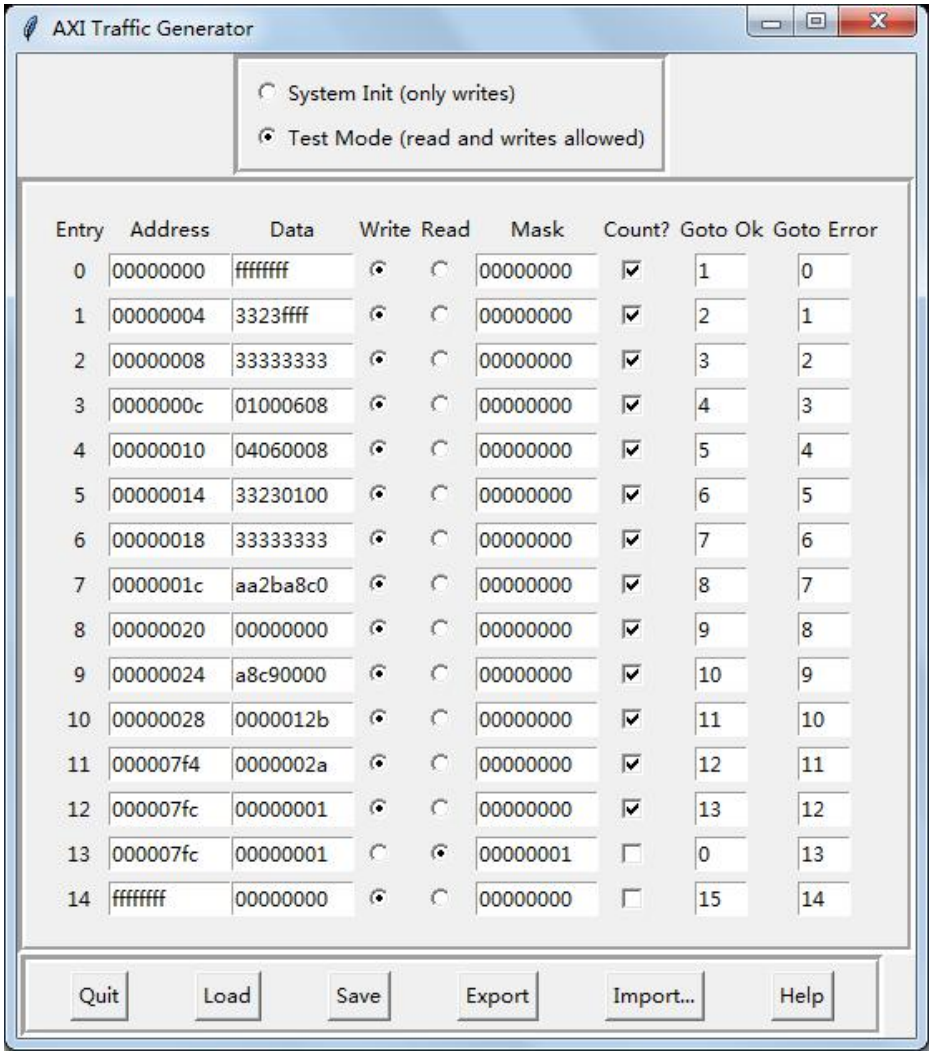
Bits	Description
31:18	Reserved. Must be filled to zeros.
17	Count as Error Checks the status of the transaction. For Write: BRESP is monitored to be OKAY. For Read: RDATA compared against the entry in Data COE File. 0 = check the BRESP/RDATA and do not increment error counter 1 = check the BRESP/RDATA and increment error counter
16	0 = read transaction 1 = write transaction
15:8	Next COE entry to be fetched upon successful completion of the current transaction.
7:0	Next COE entry to be fetched if the current transaction failed.

COE 文件由 memory_initialization_radix 和 memory_initialization_vector 两个字段组成。

PS. 然而徒手编辑还是太费劲…好在在 GitHub 上面找到了一个 ATG 的 COE 生成器，

不过还有个小 bug，顺手提了个 Issue…

下面是我的 COE’s 的界面：



AXI4-Lite 总线宽度是 32 位，所以我们需要一次送入 4 字节数据。这里面有个有趣的问题：高字节应该存高位还是低位呢？
根据 https://stackoverflow.com/questions/21449/types-of-endianness/61730#61730 的回答，Data 的高位和低位的相对关系和 endianness 无关，所以 Data COE 的高位就是总线传输数据的高位，如果 ATG 没有什么鬼畜逻辑。

> If a little endian processor writes the value 0xaabbccdd uncached to address 0 via a 32-bit wide AXI4 bus, are bits 31-24 of WDATA 0xaa or 0xdd?
>
> 31-24 bits of WDATA are 0xAA independent of processor endianness. But the order of bytes in memory is determined by the endianness. See this answer for explanation of byte-invariant endianness.

假设 EthernetLite 内存空间输入的时候是 Little 的，即 0x3323ffff 在 WDATA 上最后会变成 0x33 @ 0x0000，0x23@0x0001，0xff@0x0002，0xff@0x0003，那么在写 data 字段的时候就要注意每个 32bit 里面要颠倒着写（相对于一个字节一个字节的 dump

来说）。事实证明这个猜测没什么问题（毕竟 Microblaze 也是 Little Endian 的，要不然这 IP 核不是故意找麻烦吗..）

本例采用的帧配置如下：

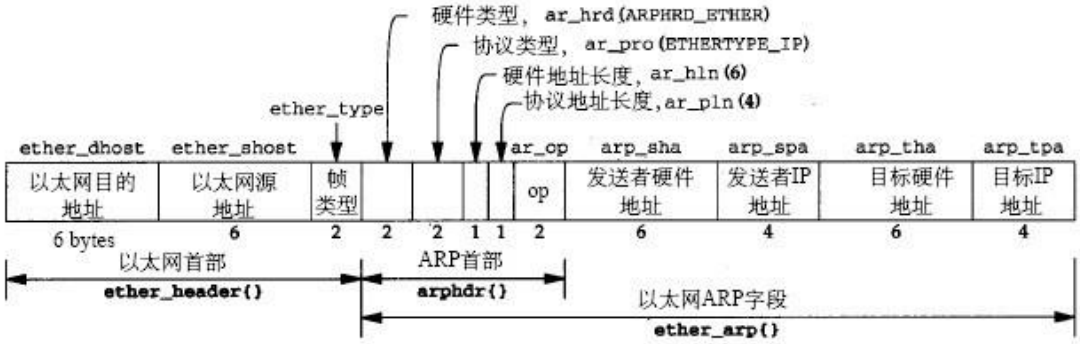


图21-7 在以太网上使用时ARP请求或回答的格式

ff ff ff ff ff ff	目标地址：Broadcast Addr.
23 33 33 33 33 33	源地址：23:33:33:33:33:33
08 06	帧类型：ARP
00 01	硬件类型：Ethernet
08 00	协议类型：IPv4
06	硬件地址长度（MAC Address 长）
04	协议地址长度（IP Address 长）
00 01	操作类型：Request（01）
23 33 33 33 33 33	发送者硬件地址：23:33:33:33:33:33
c0 a8 2b aa	发送者 IP 地址：192.168.43.170
00 00 00 00 00 00	目标硬件地址：00:00:00:00:00:00 （因为正常情况下肯定不知道对面 MAC 才要发 ARP Request）
c9 a8 2b 01	目标 IP 地址：201.168.43.1

按着这个填成前面的 COE’s，并且在最后给 0x000007f4 放一个帧长度（d'42=0x0000002A），再在 0x000007FC 放一个 0x00000001 就可以触发发送了。在成功时，通过配置返回 Entry 0 继续执行，可以重复发包，直到达到 Max Clocks To Run 或 Max Retry Count。

4. 还没有实现，但是下一步要做的事情
   a) 实现 AXI Lite Reader 和 AXI Lite Writer
   b) 实现 Ethernet FSM，可以基于 Layer−2 和 FPGA 发送命令，交给 FSM 处理。