**以太网 调试总结**

|  |  |  |
| --- | --- | --- |
| **文件状态： [√] 草稿 [ ] 正在修改**  **[ ] 正式发布** | **部门** | **系统软件组** |
| **版本** | 0.1 |
| **作者** | **朱坤华** |
| **完成时间** |  |
| **审核** |  |
| **审核时间** |  |
| **密级状态：绝密( ) 秘密( ) 内部资料(√) 公开( )** | | |

**修改记录:**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订者 | 时间 | 说明 |
| 0.1 | 朱坤华 | 2018 / 3 / 22 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

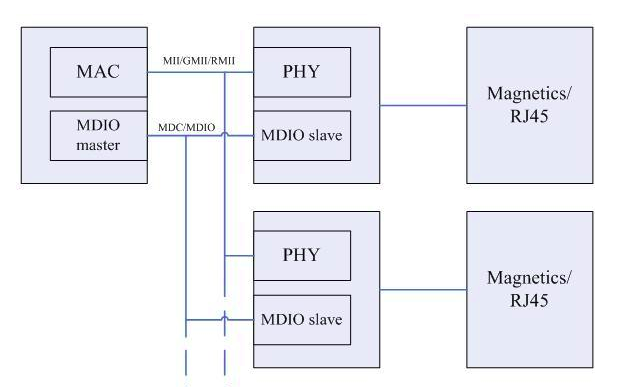
**一.背景及问题:**

调试RTL8201的时候遇到了挺多问题：开始是时钟一直不对，主要是主控要配置成输入，RTL8201F时钟输出；接着是PHY驱动不来RTL8201芯片，原因是PHY驱动不能兼容到这个IC；然后是因为硬件RX\_CLK二极管影响，导致RX一直没有数据，手动设置IP成功，但RX上数据包极少，并且全是异常；链路状态LED灯不亮问题。

该文档是总结调试过程中的一些心得，主要是这类芯片的整体架构，以及框架层中的不同层次间的调试方法。

**二. linux的以太网架构**

IEEE802.3 标准的以太网控制器结构图，下面这图就是典型的符合协议的。

在理解这个图之前，我们先看看几个定义：

**数据链路层** MAC 是 Media Access Control 的缩写，即媒体访问控制子层协议。该协议位于 OSI 七层协议中数据链路层的下半部分，主要负责控制与连接物理层的物理介质。在发送数据的时>候，MAC 协议可以事先判断是否可以发送数据，如果可以发送将给数据加上一些控制信息，最终将数据以及控制信息以规定的格式发送到物理层；在接收数据的时候，MAC 协议首先判断输入的>信息并是否发生传输错误，如果没有错误，则去掉控制信息发送至 LLC 层。以太网 MAC 由 IEEE-802.3 以太网标准定义。

**物理层** PHY 是物理接口收发器，它实现物理层。包括 MII/GMII（介质独立接口）子层、PCS（物理编码子层）、PMA（物理介质附加）子层、 PMD（物理介质相关）子层、MDI 子层。

**MII** 即媒体独立接口 , “媒体独立”表明在不对 MAC 硬件重新设计或替换的情况下，任何类型的 PHY 设备都可以正常工作。包括分别用于发送器和接收器的两条独立信道。每条信道都有自己的

数据、时钟和控制信号。MII 数据接口总共需要 16 个信号，包括 TX\_ER，TXD<3:0>，TX\_EN，TX\_CLK，COL，RXD<3:0>，RX\_EX，RX\_CLK，CRS，RX\_DV 等。

**RMII** (Reduced Media Independant Interface ) 是简化的 MII 接口 ，在数据的收发上它比 MII 接口少了一倍的信号线，所以它一般要求是 50 兆的总线时钟 。RMII 一般用在多端口的交换

机，它不是每个端口安排收、发两个时钟，而是所有的数据端口公用一个时钟用于所有端口的收发 ，这里就节省了不少的端口数目。RMII 的一个端口要求 7 个数据线 ，比 MII 少了一倍，所

以交换机能够接入多一倍数据的端口。和 MII 一样，RMII 支持 10 兆和 100 兆的总线接口速度 。

**GMII**(Gigabit MII) 是千兆网的 MII 接口，这个也有相应的 RGMII 接口，表示简化了的 GMII 接口。GMII 采用 8 位接口数据，工作时钟 125MHz，因此传输速率可达 1000Mbps 。同时兼容 MII 所规定的 10/100 Mbps 工作方式。

**MII** 管理接口是个双信号接口，一个是时钟信号 MDC，另一个是数据信号 MDIO。通过管理接口，上层能监视和控制 PHY 的寄存器。PHY 里面的部分寄存器是 IEEE 定义的，这样 PHY 把自己的

目前的状态反映到寄存器里面，MAC 通过管理接口不断的读取 PHY 的状态寄存器以得知目前 PHY 的状态，例如连接速度，双工的能力等。当然也可以通过管理接口设置 PHY 的寄存器达到控制

的目的，例如流控的打开关闭，自协商模式还是强制模式等，这也是 ethtool 的工作原理。

**MDIO/MDC**，即 PHY 管理接口串行通信总线，该总线由 IEEE 通过以太网标准 IEEE 802.3 的若干条款加以定义。MDIO 是一种简单的双线串行接口，将管理器件 ( 如 MAC 控制器、微处理器 ) 与具备管理功能的收发器 ( 如多端口吉比特以太网收发器或 10GbE XAUI 收发器 ) 相连接，从而控制收发器并从收发器收集状态信息。可收集的信息包括链接状态、传输速度与选择、断电、>低功率休眠状态、TX/RX 模式选择、自动协商控制、环回模式控制等。除了拥有 IEEE 要求的功能之外，收发器厂商还可添加更多的信息收集功能。

MDC 则是管理数据的时钟输入，最高速率可达 8.3MHz。MDIO 是管理数据的输入输出双向接口，数据是与 MDC 时钟同步的。MDIO 的工作流程为：

MDIO 接口在没有传输数据的空闲状态（IDLE）数据线 MDIO 处于高阻态。

MDIO 出现一个 2bit 的开始标识码 (01) 一个读 / 写操作开始。

MDIO 出现一个 2bit 数据来标识是读操作 (10) 还是写操作 (01)。

MDIO 出现一个 5bit 数据标识 PHY 的地址。

MDIO 出现一个 5bitPHY 寄存器地址。

MDIO 需要 2 个时钟的访问时间。

MDIO 串行读出 / 写入 16bit 的寄存器数据。

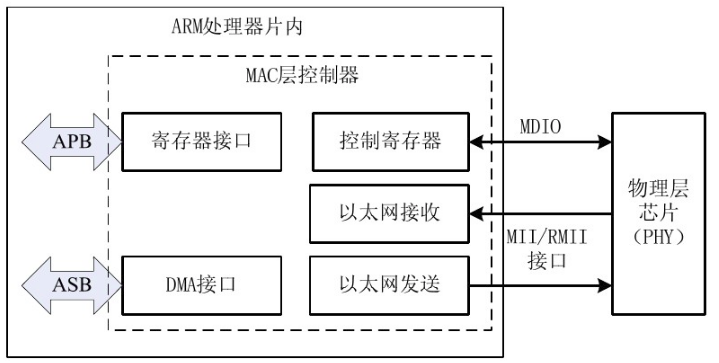
MDIO 恢复成 IDLE 状态，同时 MDIO 进入高阻状态。

看完这几个定义，我们再回来看图，图中我们看到，左边的是MAC控制器，并有MDIO 总线作为主设备，而中间是PHY芯片，以及作为从设备的MDIO总线，并且一个主MDIO总线下可以挂载多个PHY芯片，也就是可以挂很多网口。

这份文档主要想描述PHY芯片的驱动，若想了解MAC控制器，可先了解以下IEEE 802.3协议，这个协议定义了包括MII、RMII、GMII的相关寄存器和操作规范，也可以到代码中直接查看代码的驱动实现。

目前RK已经将在Android体系中加入了以太网的控制，在设置的app里面，我们可以看到以太网的开关。

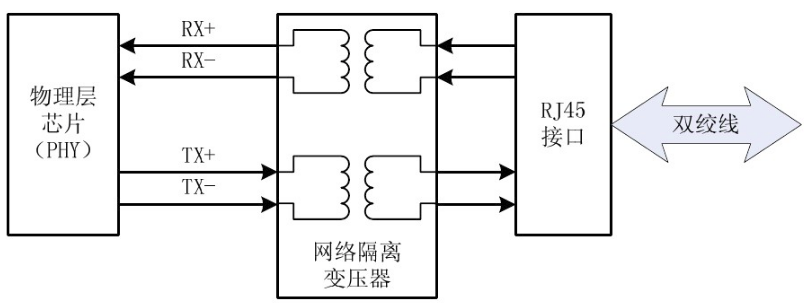
我们知道，以太网接口分为协议层和物理层。PX3这款主控的协议层是由MAC控制器控制，物理层由PHY芯片控制。MAC控制器已经集成在PX3内部了，所以我们要通过以太网上网的话，还需要一个PHY芯片。按功能来分的话，MAC控制器实现的是协议层的功能，PHY芯片实现的是物理层的功能。



如上图，MAC控制器的功能类似于一个Controller。发送数据的时候，以太网协议层将数据传送给MAC，由MAC通过DMA发送到外部接口，外部接口连接着PHY。或者接收从PHY传过来的信号，DMA搬运到内存中存储。

PX3 用的RTL8201F芯片作为PHY。一般来说这种MAC-PHY分开的，PHY都是一个独立芯片，有数字和模拟两部分，也可以集成在ARM芯片内部。负责把从MAC传送过来的数据转换成可以在网线上传输的信号>，或者接收网线上传输过来的信号，转换成数字信号回传给MAC。分为百兆PHY和千兆PHY。

PHY一般和具体的MAC控制驱动联系一起，这里以PX3的MAC驱动为例，MAC的控制驱动是rk29\_vmac.c,由它切入到PHY驱动，通过mdio总线访问、控制PHY，mdio源码实现在driver/net/phy/mdio\_bus.c中。8201F的PHY配置是自协商关闭状态，强制full duplex， 100Mbps。

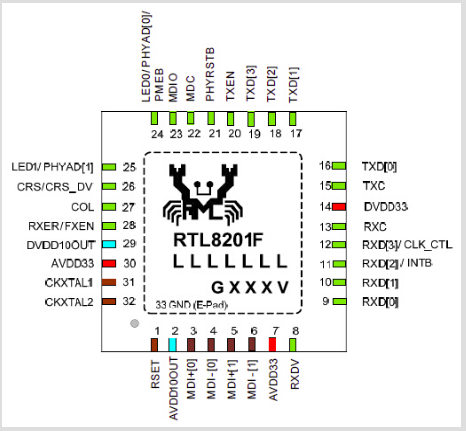


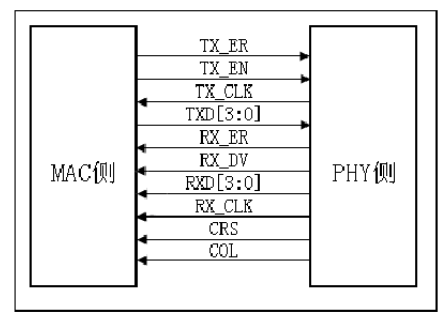
先看一下各层的代码框架：

|  |  |
| --- | --- |
| 层次 | 代码 |
| 板配、架构控制(上电、复位、时钟) | arch/arm/mach-rkpx3/board-rkpx3-sdk.c 定义了pwr/rst引脚  arch/arm/mach-rk30/board-rk31-sdk-vmac.c 包括了reset、clk的控制  arch/arm/mach-rkpx3/clock\_data.c 时钟配置  arch/arm/plat-rk/iomux.c |
| RK29 MAC控制器驱动 | drivers/net/rk29\_vmac.c 平台驱动 |
| PHY驱动 | drivers/net/phy/realtek.c 在此添加支持8201f的驱动  driver/net/phy/mdio\_bus.c  drivers/net/phy/phy\_device.c |
| framework层 | frameworks/opt/net/ethernet/ 定义了接口和服务  frameworks/base/core/java/android/net/EthernetManager.java 管理器和监听器，会通过aidl进程间通信和hal通讯  frameworks/base/core/java/android/net/IEthernetManager.aidl  frameworks/base/core/java/android/net/IEthernetServiceListener.aidl |
| APP层(设置里面的以太网选项) | packages/apps/Settings/src/com/android/settings/EthernetSettings.java |

2. RTL8201F 支持

首先，我们先看看8201F的硬件，先了解这个IC的特性。

 我们将这个图和我们的PX3的主控衔接上，PHY侧代表8201F IC。

MAC侧就是PX3主控这边，我们看看所有pin脚的功能，这图比较丑，但有助于我们理解。

TXD[1:0]：数据发送信号线，数据位宽为2，是MII接口的一半；

RXD[1:0]：数据接收信号线，数据位宽为2，是MII接口的一半；

TX\_EN(TransmitEnable)：数据发送使能信号，与MII接口中的该信号线功能一样；

RX\_ER(ReceiveError)：数据接收错误提示信号，与MII接口中的该信号线功能一样；

CLK\_REF：是由外部时钟源提供的50MHz参考时钟，与MII接口不同，MII接口中的接收时钟和发送时钟是分开的，而且都是由PHY芯片提供给MAC芯片的。这里需要注意的是，由于数据接收时钟是由外部晶振提供而不是由载波信号提取，所以在PHY层芯片内的数据接收部分需要设计一个FIFO，用来协调两个不同的时钟。

CRS\_DV：此信号是由MII接口中的RX\_DV和CRS两个信号合并而成。当介质不空闲时，CRS\_DV和RE\_CLK相异步的方式给出。当CRS比RX\_DV早结束时(即载波消失而队列中还有数据要传输时)，就会出现CRS\_DV在半位元组的边界以25MHz/2.5MHz的频率在0、1之间的来回切换。因此，MAC能够从    CRS\_DV中精确的恢复出 RX\_DV和CRS。

在100Mbps速率时，TX/RX每个时钟周期采样一个数据；在10Mbps速率时，TX/RX每隔10个周期采样一个数据，因而TX/RX数据需要在数据线上保留10个周期，相当于一个数据发送10次。

当PHY层芯片收到有效的载波信号后，CRS\_DV信号变为有效，此时如果FIFO中还没有数据，则它会发送出全0的数据给MAC，然后当FIFO中填入有效的数据帧，数据帧的开头是“101010---”交叉的前导码，当数据中出现“01”的比特时，代表正式数据传输开始，MAC芯片检测到这一变化，从而开始接收数据。

当外部载波信号消失后，CRS\_DV会变为无效，但如果FIFO中还有数据要发送时，CRS\_DV在下一周期又会变为有效，然后再无效再有效，知道FIFO中数据发送完为止。

我们要让主控的MAC模块工作起来，也就是MAC Controller，工作起来了后，我们再去调 RTL8201F。PX3的mac驱动使用的是RK29的vmac驱动，这驱动是兼容PX3的，我们直接make menuconfig使能起来。

makemenuconfig中需要打开

“RK29VMACethernetsupport”|Location:

|->DeviceDrivers

|->Networkdevicesupport|

->Ethernet(10or100Mbit)

**三. 技术总结**

主要描述此问题相关技术总结, 可以有哪些扩展的应用, 可以用来解决哪些其它的问题.

文档整体描述简洁明了, 可以插入代码及图片以助理解.