密级状态：绝密(　　)　秘密(　　)　内部资料(　　)　公开(√)

Rockchip 平台以太网配置指南

(底层平台中心，第一产品部)

文件状态：	文件标识：	Rockchip 平台以太网配置指南
[　] 草稿	当前版本：	2.5
[√] 正式发布	作　　者：	陈智、胡卫国
[　] 正在修改	完成日期：	2016-04-20
	审　　核：	
	审核日期：	

版 本 历 史

版本号	作者	修改日期	修改说明
v0.1	陈智	2014-03-15	创建文件
v2.0	陈智	2014-09-15	增加对 RK3288/ RK3128/ RK3036 配置的说明
v2.1	陈智	2014-09-25	1. 修改第 2 章关于 PHY 芯片驱动的描述 2. 修改 2.2.3 及 2.2.5 中对 RMII 时钟的配置
V2.2	胡卫国	2015-0309	1. 增加 GMAC 问题排查部分
V2.3	陈智	2016-04-20	1． 增加对 RK3368/RK3366/RK3399 配置的说明(2.2.4/2.2.8/2.2.9) 2． 改名为《Rockchip 平台以太网配置指南》
V2.4	胡卫国	2017-01-22	1. 增加 RK3328/3228H 的配置说明 2. 更新问题排查部分
V2.5	胡卫国	2017-08-08	1. 新增一些排查方法

目 录

1 概述

本文早期基于 Rockchip BOX SDK 进行描述。从 3368 开始 BOX 及 MID 等产品线代码实现统一。目前此文档可适用于所有使用 Rockchip 以太网功能的芯片。 本文将在第 2 章和第 3 章中针对 RMII/RGMII 接口芯片和 USB 以太网两种类型芯片在 SDK 上的配置做详细的描述。第 4 章提供常见问题的排查建议。附录将会列出所支持的芯片列表。

2 以太网 MAC PHY 芯片

由于在 RK 系列的 SoC 中内置了以太网 MAC 控制器，所以只需要搭配一颗以太网 PHY 芯片，即可实现以太网卡功能。按照规范，即使是不同厂家的 PHY，仍然有一部分寄存器的定义是通用的，只要配置了这些通用的寄存器，基本上 PHY 就可以正常工作。因此，在 Linux 驱动中有通用的 PHY 驱动，目前的芯片所配套的 SDK 中使用的都是通用驱动，当然 SoC 中的 MAC 驱动是需要实现的。

10/100M 以太网 PHY 与 MAC 之间的接口主要有 MII 和 RMII。 10/100/1000M 以太网 PHY 与 MAC 之间的接口主要有 RGMII。RK 系列的各个 SoC 支持的 PHY 接口列表如下：

SoC	MII	RMII	RGMII	SoC 内置 PHY	MAC 数目
RK2918	√	√			
RK2908	√	√			
RK3066		√			
RK3188		√			
RK3288		√	√		
RK3368		√	√		
RK3036		√			
RK3128		√	√		
RK322x		√	√	√	
RK3366		√	√		
RK3399		√	√		
RK3328 RK3228H		√	√	√	2

注：

1. 2918/2908 虽然硬件上有 MII 接口，但驱动上并不支持

2. MAC 数目指的是芯片内 MAC 控制器的数目，默认一般都是 1 个，像 RK3328 就有两个。

Kernel 版本信息：

SoC	控制器	Kernel 版本
RK2918	**VMAC**	**2.6.x**
RK2908	**VMAC**	**2.6.x**
RK3066	**VMAC**	**3.0.36**
RK3188	**VMAC**	**3.0.36**
RK3288	**GMAC**	**3.10**
RK3368	**GMAC**	**3.10**
RK3036	**VMAC**	**3.10**
RK3128	**GMAC**	**3.10**
RK322x	**GMAC**	**3.10**
RK3366	**GMAC**	**4.4**
RK3399	**GMAC**	**4.4**
RK3328 RK3228H	**GMAC**	**3.10**

2.1 RK3066

2.1.1 基本配置

以太网在 kernel 中的 menuconfig 配置如下所示，可以根据产品需求开关此配置， SDK 中下述配置默认打开

```
Device Drivers    --->
    Networking support    --->
        [*]    Ethernet (10 or 100Mbit)    --->
                <*> RK29 VMAC ethernet support
```

MAC 驱动代码位于 kernel/drivers/net/rk29_vmac.c

此外，由于各个平台对 PHY 进行上下电操作使用的控制管脚或 PMU 会有所不同，同时各款 RK 芯片的一些和以太网相关的通用寄存器地址也会有所不同，所以和平台相关的操作和驱动代码是分开的，一般存放于 arch/arm/mach-rkxx 文件夹中，因不同芯片而异，如 3066 的上述操作位于文件 arch/arm/mach-rk30/board-rk30-sdk-vmac.c 中。该文件中的接口最终会被驱动代码所调用，需要特别注意的就是对 PHY 的上下电的操作，有可能不同产品所选用的 PMU 与 SDK 不同，或者使用的 PMU 的 LDO 口与 SDK 不同， 所以需要根据实际情况修改这个文件。

2.1.2 时钟配置

根据规范，RMII 接口需要 50M 参考时钟来保证 MAC 和 PHY 之间数据传输的同步。50M 时钟，可以由 MAC 来提供，也可以由 PHY 来提供。50M 时钟必须是精确的，且必须保证尽可能小的 jitter， 太大的 jitter 会导致较大的传输的错包率。SDK 默认使用 MAC，也就是 RK 芯片内部的 MAC 控制器来提供时钟。一般情况下，并不需要使用 PHY 来提供参考时钟，因为这样需要额外增加一颗晶振来实现。除非由于分频的原因，MAC 无法给出符合要求的 50M 参考时钟，RK3188T 上即存在这样的问题，所以需要由 PHY 来提供参考时钟。需要在 kernel 上增加如下配置，并相应修改电路，即可实现(该配置目前只在 3188 相关的 SDK 中有效)。

```
Device Drivers    --->
    Networking support    --->
        [*]    Ethernet (10 or 100Mbit)    --->
            <*> Select the vmac source clock (RK29_VMAC_EXT_CLK )
```

2.1.3 MAC 地址

通常，每个以太网设备只有唯一的 MAC 地址，所以需要有一个地方用来存储这个唯一的地址，同时在打开以太网时读取出这个地址，并写入 PHY 寄存器。SDK 提供了四种获取以太网 MAC 地址的方法：

1) 存储在 NAND 的 IDB 中

首先要保证 kernel 中的配置 CONFIG_ETH_MAC_FROM_IDB 已打开

其次要使用烧写工具将地址写入，烧写工具在 SDK 中有提供。

2) 存储在 EEPROM 中

首先要保证 kernel 中的配置 CONFIG_ETH_MAC_FROM_EEPROM 已打开

其次 EEPROM 的驱动见 drivers/staging/rk29/eeprom，根据不同型号请自行作相应修改

3) 使用 WiFi 的 MAC 地址

该种方法的原理是在系统启动时自动加载一次 Wi-Fi 驱动，同时将 Wi-Fi 的 MAC 地址读出并存储在/data 分区的一个文件中，以太网打开时，读取该文件中的地址。

首先要保证 kernel 中的配置 CONFIG_ETH_MAC_FROM_WIFI_MAC 已打开

其次要保证 Android 上 wlan_mac 程序存在，且已在 init.rc 或 init.rkxx.rc 中已添加如下脚本

```
service wlan_mac /system/bin/wlan_mac
    class main
    oneshot
```

以太网驱动读取地址的代码存于 drivers/net/eth_mac，请根据实际需求修改此代码。

由于不同的网络设备的 MAC 地址必须是唯一的，所以请考虑使用这种方法的风险性。

4) 使用随机地址

若上述三种方法均未采用，驱动中会在每次打开以太网时随机生成 MAC 地址

由于不同的网络设备的 MAC 地址必须是唯一的，所以请考虑使用这种方法的风险性。

2.2 RK3188

同 RK3066，见 2.1

2.3 RK3288

2.3.1 基本配置

以太网在 kernel 中的 menuconfig 配置如下所示，可以根据产品需求开关此配置，SDK 中下述配置默认打开

```
Device Drivers  --->
    Networking support  --->
        [*]  Ethernet (10 or 100Mbit)  --->
            RockChip devices --->
                RockChip 10/100/1000 Ethernet driver
```

由于 3.10 kernel 的驱动架构与 3.0.36 相比有较大改动，引入了 device tree 的概念，许多配置改为在 dts 文件中进行修改。所以，相应地，对以太网相关的一些配置的修改也需要在 dts 文件中进行。以 3288 box sdk 板为例，相应的 dts 文件为 arch/arm/boot/dts/rk3288-box.dts. GMAC 驱动代码位于 drivers/net/ethernet/rockchip/gmac/

```
&gmac_clkin {
        clock-frequency = <125000000>;          --> PHY 供给 GMAC 的时钟大小
};
&gmac {
//      pmu_regulator = "act_ldo5";                          ---> 给 PHY 供电的 PMU 上的 ldo
//      pmu_enable_level = <1>; //1->HIGH, 0->LOW            ---> ldo 有效电平
//      power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>;   ---> 电源控制 IO 及有效电平
        reset-gpio = <&gpio4 GPIO_B0 GPIO_ACTIVE_LOW>;  ---> 复位 IO 及有效电平
        phy-mode = "rgmii";                              ---> PHY 接口
        clock_in_out = "input";                  --->时钟方向
        tx_delay = <0x30>;                       ---> TX 线上的延时值
        rx_delay = <0x10>;                       ---> RX 线上的延时值
};
```

一般情况下，控制供电方式或者为 PMU，或者为 IO 控制，若使用其中一种，则屏蔽

另外一种。还有一种情况，是对 PHY 长供电，这时候可以考虑把二者都屏蔽。

PHY 接口的配置：根据板上所使用的 PHY 的接口进行配置，一般千兆 PHY 为 RGMII，百兆 PHY 为 RMII。

时钟方向： input 表示由 PHY 提供，output 表示由 GMAC 提供。 关于时钟，请参考《2.3.2 时钟配置》这一节中的说明。

TX/RX 线上的延时值：请参考《2.3.3 时序配置》这一节中的说明

PHY 供给 GMAC 的时钟大小：这个值只有在时钟方向为 input 时才会使用到，RMII 接口时，设为 50000000, RGMII 接口时设为 125000000

100M PHY 配置

```
&gmac_clkin {
        clock-frequency = <50000000>;          --> 修改成 50M
};
&gmac {
//          power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>;
            reset-gpio = <&gpio4 GPIO_B0 GPIO_ACTIVE_LOW>;
            phy-mode = "rmii";                          ---> 修改成 rmii
            clock_in_out = "output";                    ---> 修改成 output，也就是由 RK 主控提供
        tx_delay = <0x30>;
        rx_delay = <0x10>;
};
```

注意：如果是 RK3288 芯片，50M 主时建议由外部 PHY 提供，因为 RK 主控分不出 50M clock 或分出的 clock 可能不精准，会造成 GMAC 丢包或无法工作。

1000M PHY 配置

```
&gmac_clkin {
        clock-frequency = <125000000>;         --> 修改成 125M
};
&gmac {
//          power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>;
            reset-gpio = <&gpio4 GPIO_B0 GPIO_ACTIVE_LOW>;
            phy-mode = "rgmii";                         ---> 修改成 rgmii
            clock_in_out = "input";                     ---> 修改成 input，也就是 125M 由 PHY 提供
            tx_delay = <0x30>;
            rx_delay = <0x10>;
};
```

注意：125M 主时间需要由外部 PHY 提供，这是因为 RK 主控分不出 125M clock 或分出的 clock 可能不精准，会造成 GMAC 丢包或无法工作。

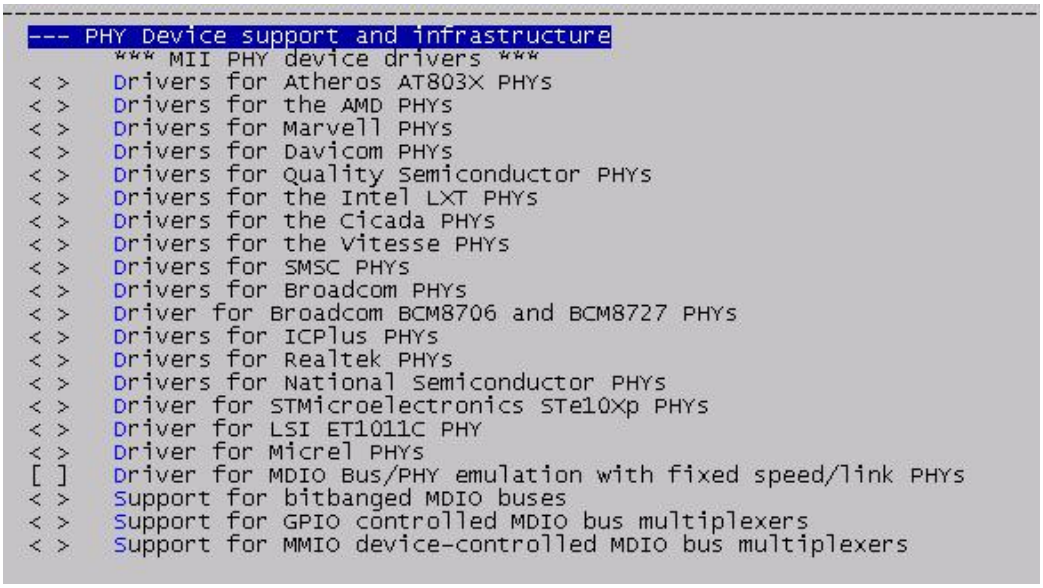
3.10 版本 kernel 的 phy 驱动代码位于 drivers/net/phy/

menuconfig 的配置如下：

Device Drivers ---->

Networking support ---->

PHY Device support and infrastructure



注意：不需要配置任何 PHY 驱动，默认使用 general phy 驱动。

2.3.2 时钟配置

(1) RGMII

RGMII 上分别有 TX_CLK 和 RX_CLK 两个时钟，这两个时钟分别由 MAC 和 PHY 产生，这两个时钟频率的大小和网速的大小相关，千兆网速的时候，时钟频率为 125MHz，百兆为 25MHz, 十兆为 2.5MHz。

TX_CLK 可以由 3288 内部的 PLL 分频产生，也可以由外部的时钟输入经过分频后产生。目前我们使用的是由外部输入的时钟，这样的时钟相对于内部 PLL 产生的时钟更加独立，不受 3288 内部分频策略的影响，因此更加稳定。而对于 PHY 来说，本身就需要一个 25M 的晶振作为时钟源，因此 RX_CLK 正是由这个时钟源倍频或分频得到

的。绝大多数 PHY 还有这样的一个输出管脚，可以输出一个时钟给 MAC，也就是上面描述的相对于 MAC 来说的外部时钟，这个时钟大小为 125MHz，作为 MAC 端 TX_CLK 的时钟源。 2.2.3.1 中描述的时钟方向正是指的是用内部时钟 output 或是外部时钟 input。

(2) RMII

对于 RMII 接口来说，需要的只是一个 50MHz 的参考时钟。针对 RK3288 芯片，需要注意的是，若时钟方向设为 output，也就是从 3288 的 GMAC 向 PHY 提供参考时钟，就需要额外修改 arch/arm/boot/dts/rk3288.dtsi，把 npll 初始值设为 1200000000， 也就是 1.2G， 如下：

```
diff --git a/arch/arm/boot/dts/rk3288.dtsi b/arch/arm/boot/dts/rk3288.dtsi
index 75392fd..503f6b6 100755
--- a/arch/arm/boot/dts/rk3288.dtsi
+++ b/arch/arm/boot/dts/rk3288.dtsi
@@ -496,7 +496,7 @@
<&usbphy_480m &otgphy2_480m>;
rockchip,clocks-init-rate =
<&clk_core 792000000>,   <&clk_gpll 600000000>,//box changed
-          <&clk_cpll 29700000>,     <&clk_npll 1250000000>,
+          <&clk_cpll 29700000>,     <&clk_npll 1200000000>,
           <&aclk_bus_src 300000000>,          <&aclk_bus 300000000>,
           <&hclk_bus 150000000>,   <&pclk_bus 75000000>,
           <&clk_crypto 150000000>,             <&aclk_peri 300000000>,
```

2.3.3 时序配置

时序配置是千兆网卡上才开始有的概念，具体到 3288 上，也就是 RGMII 接口才需要去特别配置时序。这是因为时钟频率更高了以后更容易受到 PCB layout 走线，电磁干扰等因素的影响，因此需要更加精确的调整。比如 TX 或 RX 出现走线太长的情况，就有可能需要配置不同的延时值来调整时序。2.2.3.1 中列出的两个值是 SDK 板上用的，目前在多数客户板上也是使用这两个值，如果有出现测试吞吐率不足、吞吐率不稳定或者 TX、RX 不通的情况，可以适当调整这两个值(dts 中的 tx_delay, rx_delay 参

数)，范围是 0x0—0x7F. 可使用 ftp 上的补丁《GMAC tx rx delay 动态调整补丁》来获取合适的取值。

2.3.4 IO 驱动强度配置

如果发现 gmac 信号质量不佳，幅度偏大或偏小，可通过调整相应的 IO 驱动强度来改善。驱动强度有 4 档，分别是 2mA, 4mA, 8mA 和 12mA，以 RK3288 为例，将 txpins 的驱动强度调整为 4mA，调整方法如下：

```
--- a/arch/arm/boot/dts/rk3288-pinctrl.dtsi
+++ b/arch/arm/boot/dts/rk3288-pinctrl.dtsi
@@ -908,35 +908,35 @@
                                mac_clk: mac-clk {
                                        rockchip,pins = <MAC_CLK>;
                                        rockchip,pull = <VALUE_PULL_DISABLE>;
-                                       rockchip,drive = <VALUE_DRV_DEFAULT>;
                                        // 注意：VALUE_DRV_4MA 这个宏名字在各个芯片平台可能有差异
+                                       rockchip,drive = <VALUE_DRV_4MA>;
                                        //rockchip,tristate = <VALUE_TRI_DEFAULT>;
                                };

                                mac_txpins: mac-txpins {
                                        rockchip,pins = <MAC_TXD0>, <MAC_TXD1>, <MAC_TXD2>, <MAC_TXD3>,
<MAC_TXEN>, <MAC_TXCLK>;
                                        rockchip,pull = <VALUE_PULL_DISABLE>;
-                                       rockchip,drive = <VALUE_DRV_DEFAULT>;
+                                       rockchip,drive = <VALUE_DRV_4MA>;
                                        //rockchip,tristate = <VALUE_TRI_DEFAULT>;
                                };

                                mac_rxpins: mac-rxpins {
                                        rockchip,pins = <MAC_RXD0>, <MAC_RXD1>, <MAC_RXD2>, <MAC_RXD3>,
<MAC_RXDV>, <MAC_RXER>, <MAC_RXCLK>,
                                        rockchip,pull = <VALUE_PULL_DISABLE>;
-                                       rockchip,drive = <VALUE_DRV_DEFAULT>;
+                                       rockchip,drive = <VALUE_DRV_4MA>;
                                        //rockchip,tristate = <VALUE_TRI_DEFAULT>;
```

```
                        };

                        mac_crs: mac-crs {
                                rockchip,pins = <MAC_CRS>;
                                rockchip,pull = <VALUE_PULL_DISABLE>;
-                               rockchip,drive = <VALUE_DRV_DEFAULT>;
+                               rockchip,drive = <VALUE_DRV_4MA>;
                                //rockchip,tristate = <VALUE_TRI_DEFAULT>;
                        };

                        mac_mdpins: mac-mdpins {
                                rockchip,pins = <MAC_MDIO>, <MAC_MDC>;
                                rockchip,pull = <VALUE_PULL_DISABLE>;
-                               rockchip,drive = <VALUE_DRV_DEFAULT>;
+                               rockchip,drive = <VALUE_DRV_4MA>;
                                //rockchip,tristate = <VALUE_TRI_DEFAULT>;
                        };
                };
```

2.3.5 MAC 地址

目前对 MAC 地址的读取策略是，优先使用烧写在 IDB 中的 MAC 地址，若该地址符合规范，则使用，若不符合或没有烧写，则使用随机生成的地址（重启开机 MAC 地址会变化）。

在 RK3399、RK3328/RK3228H 及以后的版本中，对策略进行了完善：优先使用烧写在 IDB 或 vendor Storage 中的 MAC 地址，若该地址符合规范，则使用，若不符合或没有烧写，则随机生成 MAC 地址保存到 Vendor 分区中并使用，重启或恢复出厂设置不会丢失。

烧写工具在 SDK 中会附带。

2.4 RK3368

同 RK3288, 见 2.3

2.5 RK3036

RK3036 使用 3.10 kernel，配置如下

```
Device Drivers    --->
    Networking support    --->
        [*]    Ethernet (10 or 100Mbit)    --->
            RockChip devices --->
                RockChip 10/100 Ethernet driver
```

由于 3.10 kernel 的驱动架构与 3.0.36 相比有较大改动，引入了 device tree 的概念，许多配置改为在 dts 文件中进行修改。所以，相应地，对以太网相关的一些配置的修改也需要在 dts 文件中进行。SDK 上的相应 3036 的 dts 文件为 arch/arm/boot/dts/rk3036-sdk.dts

```
&vmac {
//          pmu_regulator = "act_ldo5";
//          pmu_enable_level = <1>; //1->HIGH, 0->LOW
//          power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>;
            reset-gpio = <&gpio2 GPIO_C6 GPIO_ACTIVE_LOW>;
};
```

上述 dts 文件中各个字段的含义可参考《2.3.1 基本配置》中的配置
MAC 驱动代码位于 drivers/net/ethernet/rockchip/vmac/rk29_vmac.c

2.6 RK3128

同 RK3288, 见 2.3

注：3128 SDK 的时钟方案，可使内部 PLL 分出 50M 时钟供给 GMAC，因此 2.2.3.2 时钟配置(2)RMII 这个章节中描述的对 npll 的配置在 3128 上并不需要.

2.7 RK322x

由于 RK322x 系列芯片，内部已经集成了一个 10/100M 的 PHY，所以 RK322x 系列的 dts 配置有了少许变动。对于使用内部 PHY 的 dts 配置如下：

```
&gmac {
        /* pmu_regulator = "act_ldo5"; */
        /* power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>; */
        /* reset-gpio = <&gpio2 GPIO_D0 GPIO_ACTIVE_LOW>; */
        /* phyirq-gpio = <&gpio0 GPIO_B1 GPIO_ACTIVE_LOW>; */
        /* control link LED */
        link-gpio = <&gpio2 GPIO_B6 GPIO_ACTIVE_HIGH>;    // link 灯的 GPIO 配置
        led-gpio = <&gpio2 GPIO_B0 GPIO_ACTIVE_HIGH>;     //TX/RX 灯的 GPIO 配置
        phy-mode = "rmii";
        pinctrl-names = "default";
        /* rmii_pins, rgmii_pins, phy_pins */
        pinctrl-0 = <&phy_pins>;
        clock_in_out = "output";
        tx_delay = <0x30>;
        rx_delay = <0x10>;
        /* internal or external */
        phy-type = "internal";
};
```

对于使用外部 PHY 的配置主要注意以下几处标红的地方，至于使用的 100M 还是 1000M 的 PHY，配置方法跟之前（312x，3288）类似：

```
&gmac {
        /* pmu_regulator = "act_ldo5"; */
        /* power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>; */
        reset-gpio = <&gpio2 GPIO_D0 GPIO_ACTIVE_LOW>;
        /* phyirq-gpio = <&gpio0 GPIO_B1 GPIO_ACTIVE_LOW>; */
        phy-mode = "rgmii";                    // 使用 1000M
        pinctrl-names = "default";
        pinctrl-0 = <&rgmii_pins>;
        clock_in_out = "input";                // 由 PHY 提供 125M clock
        tx_delay = <0x30>;
        rx_delay = <0x10>;
        /* internal or external */
        phy-type = "external";
};
```

2.8 RK3366

从 3366 开始，kernel 开始使用 4.4 版本，各驱动尽可能使用官方 Linux kernel 的代码。此前，很多 RK 的驱动已经 upstream 到官方 Linux kernel，也包括 GMAC。因此，在

官方 4.4 kernel 中已包含了支持 RK 所使用的 GMAC 的代码框架。在此框架上，已添加了对 3288 及 3368 的支持。所以往后芯片的 GMAC 驱动会套用此框架，必要时也会进行 upstream。代码位置在 drivers/net/ethernet/stmicro/stmmac/. RK 平台相关代码主要位于 dwmac-rk.c 中.

DTS 配置有一些不同，以 arch/arm64/boot/dts/rockchip/rk3366-tb.dts 为例，说明如下：

(也可以参考 Documentation/devicetree/bindings/net/rockchip-dwmac.txt)

```
&gmac {
        phy-supply = <&vcc_phy>;   ---PHY 供电由&vcc_phy 提供，见后续说明
        phy-mode = "rgmii";        --- rgmii 或 rmii， pinctrl-0 字段必须与此字段相匹配
        clock_in_out = "input";    --- input: 时钟由 PHY 输入给 MAC， output: 与 input 相反
        snps,reset-gpio = <&gpio2 15 GPIO_ACTIVE_LOW>; ---用于复位 PHY 的 GPIO
        snps,reset-active-low;                          ---复位 PHY 的 GPIO 低有效
        snps,reset-delays-us = <0 10000 50000>;         ---表示复位 PHY 前的延时为 0ms，拉低维持的时间为 10ms, 拉高后延时 50ms
        assigned-clocks = <&cru SCLK_MAC>;      ---MAC 的时钟源
        assigned-clock-parents = <&ext_gmac>;   ---MAC 父时钟由&ext_gmac 提供，见后续说明
        pinctrl-names = "default";
        pinctrl-0 = <&rgmii_pins>;              --- 设为&rgmii_pins 或&rmii_pins， 必须和 phy-mode 字段匹配
        tx_delay = <0x30>;
        rx_delay = <0x10>;
        status = "okay";
};

        vcc_phy: vcc-phy-regulator {
                compatible = "regulator-fixed";
                enable-active-high;
                gpio = <&gpio0 25 GPIO_ACTIVE_HIGH>;    ---控制 LDO 开关的 GPIO
                pinctrl-names = "default";
                pinctrl-0 = <&eth_phy_pwr>;             ---设置 GPIO 的 IOMUX
                regulator-name = "vcc_phy";
                regulator-always-on;
                regulator-boot-on;
        };

        ext_gmac: external-gmac-clock {     // 名字不一定是 ext_gmac，可能是 clkin_gmac
                compatible = "fixed-clock";     ---表示这是一个固定的时钟（由外部 PHY 输入）
                clock-frequency = <125000000>;  ---默认为 125M
                clock-output-names = "ext_gmac";
                #clock-cells = <0>;
        };
```

2.9 RK3399

同 RK3366, 参见 2.8

DTS 请参考 arch/arm64/boot/dts/rockchip/rk3399-tb.dtsi

2.10 RK3328/3228H

RK3328/3228H 带有两个 GMAC 控制器，并且内置一个 100M PHY，可作双网卡：

一是 GMAC+内部 100M PHY

二是 GMAC+外部 1000M PHY

如果使用的是 Kernel 4.4 版本，参见 2.8 节中的配置

如果使用的是 Kernel 3.10 版本，配置文件在 arch/arm64/boot/dts/rk322xh-evb.dtsi

```
&gmac2io {                              网卡 2---需要外接 PHY 使用，配置信息参考 2.3 节
        /* pmu_regulator = "act_ldo5"; */
        /* power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>; */
        reset-gpio = <&gpio3 GPIO_B0 GPIO_ACTIVE_LOW>;
        /* phyirq-gpio = <&gpio0 GPIO_B1 GPIO_ACTIVE_LOW>; */
        phy-mode = "rmii";
        pinctrl-names = "default";
        pinctrl-0 = <&rmiim1_pins>;
        clock_in_out = "output";
        /* delay line for rgmii + 1000M mode */
        tx_delay = <0x25>;
        rx_delay = <0x11>;
};

&gmac2phy {                             网卡 1----使用内部 100M PHY，不需要修改
        clock_in_out = "output";
        status = "okay";
};
```

3 USB 以太网卡芯片

USB 以太网芯片内部集成了以太网 MAC 和 PHY，因此并不需要 RK 芯片内部的 MAC，驱动和第 2 章所述的也完全不同。所以如果产品中使用的是内置的 USB 以太网芯片，可以考虑把 2.2 中提到的配置关闭。

3.1 Kernel 配置

Linux kernel 中有 USB 以太网卡的驱动，只需要打开如下配置即可

Device Drivers ---->
Networking support ---->
USB Network Adapters ---->
Multi-purpose USB Networking Framework

```
<*> USB CATC NetMate-based Ethernet device support
<*> USB KLSI KL5USB101-based ethernet device support
<*> USB Pegasus/Pegasus-II based ethernet device support
<*> USB RTL8150 based ethernet device support
<*> Realtek RTL8152 Based USB 2.0 Ethernet Adapters
<*> Multi-purpose USB Networking Framework
<*>   ASIX AX88xxx Based USB 2.0 Ethernet Adapters
<*>   ASIX AX88179/178A USB 3.0/2.0 to Gigabit Ethernet
-*-   CDC Ethernet support (smart devices such as cable modems)
<*>   CDC EEM support
-*-   CDC NCM support
<*>   CDC MBIM support
<*>   Davicom DM9601 based USB 1.1 10/100 ethernet devices
<*>   Davicom DM9620 based USB 2.0 10/100 ethernet devices
<*>   SMSC LAN75XX based USB 2.0 gigabit ethernet devices
<*>   SMSC LAN95XX based USB 2.0 10/100 ethernet devices
<*>   GeneSys GL620USB-A based cables
<*>   NetChip 1080 based cables (Laplink, ...)
<*>   Prolific PL-2301/2302/25A1 based cables
<*>   Moschip MCS7830 based Ethernet adapters
<*>   Host for RNDIS and ActiveSync devices
<*>   Simple USB Network Links (CDC Ethernet subset)
[*]     ALi M5632 based 'USB 2.0 Data Link' cables
[*]     AnchorChips 2720 based cables (Xircom PGUNET, ...)
[*]     eTEK based host-to-host cables (Advance, Belkin, ...)
[*]     Embedded ARM Linux links (iPaq, ...)
[*]     Epson 2888 based firmware (DEVELOPMENT)
[*]     KT Technology KC2190 based cables (InstaNet)
<*>   Sharp Zaurus (stock ROMs) and compatible
<*>   Conexant CX82310 USB ethernet port
<*>   Samsung Kalmia based LTE USB modem
<*>   QMI WWAN driver for Qualcomm MSM based 3G and LTE modems
<*> Option USB High Speed Mobile Devices
<*> Intellon PLC based usb adapter
<*> Apple iPhone USB Ethernet driver
<*> USB-to-WWAN Driver for Sierra Wireless modems
< > LG VL600 modem dongle
```

通常各款 USB 以太网卡还会有自己的驱动，可以根据实际情况打开上述配置下一级相应的配置即可。另外有些厂家还会额外提供驱动的更新，按厂家指导进行即可。

3.2 MAC 地址烧写

USB 以太网卡 MAC 地址烧写的概念和 2.4 所述不同，由于 USB 以太网芯片是一个独立的网卡，所以必须事先固化 MAC 地址到芯片的 OTP 中。不同厂家会有不同的实现方式，有些是事先内置好的（比如 USB 以太网 Dongle），有些则需要使用相应的工具进行烧写，可以联系供应商提供相应支持。

注意：没有烧写 MAC 地址的 USB 以太网芯片无法正常工作！

4 以太网常见问题排查

注意：以下涉及的讨论都是针对 Kernel 3.10 版本。如果是 Kernel 3.0 版本平台，请见文档《RK_Android 平台以太网调试说明 V1.2.pdf》

4.1 以太网补丁 ftp 地址

本文中所描述的补丁可从以下 ftp 下载：

ftp://www.rockchip.com.cn

用户名：rkwifi

密码 ：Cng9280H8t

补丁目录：27-以太网相关补丁

4.2 phy 寄存器读写调试

Kernel 3.10 版本：

系统中提供以下节点（以 rk3288 为例，其它芯片可通过查找关键字“phy_reg”找到具体路径）供 phy 寄存器读写：

```
/sys/devices/ff290000.eth/stmmac-0:01/phy_reg
/sys/devices/ff290000.eth/stmmac-0:01/phy_regValue
```

例如，如果要读写 0x01 寄存器，操作如下：

```
echo 1 > /sys/devices/ff290000.eth/stmmac-0:01/phy_reg
# 读寄存器 0x01
cat /sys/devices/ff290000.eth/stmmac-0:01/phy_regValue
#  写 0x10 到寄存器 0x01
echo 0x10 > /sys/devices/ff290000.eth/stmmac-0:01/phy_regValue
```

Kernel 4.4 版本：

如果要读写 0x01 寄存器，操作如下：

```
# 读寄存器
cat /sys/bus/mdio_bus/devices/stmmac-0\:01/phy_registers
#  写 0x10 到寄存器 0x01
echo 16 1 > /sys/bus/mdio_bus/devices/stmmac-0\:01/phy_registers
```

4.3 拔掉网线后 statusbar 仍然显示以太网图标

Kernel menuconfig 中不要选择任何 PHYDevice Driver，设置成如下配置：

```
--- PHY Device support and infrastructure
        *** MII PHY device drivers ***
< >   Drivers for Atheros AT803X PHYs
< >   Drivers for the AMD PHYs
< >   Drivers for Marvell PHYs
< >   Drivers for Davicom PHYs
< >   Drivers for Quality Semiconductor PHYs
< >   Drivers for the Intel LXT PHYs
< >   Drivers for the Cicada PHYs
< >   Drivers for the Vitesse PHYs
< >   Drivers for SMSC PHYs
< >   Drivers for Broadcom PHYs
< >   Driver for Broadcom BCM8706 and BCM8727 PHYs
< >   Drivers for ICPlus PHYs
< >   Drivers for Realtek PHYs
< >   Drivers for National Semiconductor PHYs
< >   Driver for STMicroelectronics STe10Xp PHYs
< >   Driver for LSI ET1011C PHY
< >   Driver for Micrel PHYs
[ ]   Driver for MDIO Bus/PHY emulation with fixed speed/link PHYs
< >   Support for bitbanged MDIO buses
< >   Support for GPIO controlled MDIO bus multiplexers
< >   Support for MMIO device-controlled MDIO bus multiplexers
```

如果选择了某个特定的 PHY Device Driver，反而会造成异常。（例如断开 RJ45 网线后，上层仍然显示以太网连接。即可能出现 kernel 不能识别插拔网线的动作，正常情况，插拔网线 kernel 的 log 中会有 Link is down /Link is up 类似的 log。）

4.4 MAC 以太网无法正常工作

以太网无法正常工作主要指以太网不通。例如接上网线后无法获取到 IP 地址或设置静态 IP 地址后无法上网。原因有多种，可按以下小节顺序排查。

4.4.1 DMA initialization failed

如果 kernel 打印以下异常 log：

```
stmmac_open: DMA initialization failed
```

这种情况只有在“clock_in_out = "input"情况下才出现。

A) **需要确认 GMAC 工作主时钟 MAC_CLK 是否有从 PHY 供给主控：**

使用 100M PHY 时，其频率是 50M

使用 1000M PHY 时，其频率是 125M

参考《工作时钟确认》小节排查

B) 如果有 clock，需要确认 clock 的幅度是否达标，一般需要 3.0V 以上

C) **需要确认 iomux 是否正确，见下面《引脚复用 IOMUX 确认》小节**

D) 如果是 3368 kernel 4.4 版本，需要修改

```
arch/arm64/boot/dts/rockchip/rk3368-r88.dts
clkin_gmac: external-gmac-clock {
compatible = "fixed-clock";
clock-frequency = <125000000>;
- clock-output-names = "clkin_gmac";
+ clock-output-names = "ext_gmac";
#clock-cells = <0>;
};
```

4.4.2 PHY 初始化失败

如果出现 PHY 初始化异常

类似如下异常打印：

```
stmmac_open: Cannot attach to PHY
```

或

```
eth0: No PHY found
```

或

```
stmmac_open: Cannot attach to PHY (error: -19)
```

PHY 正常识别会有类似如下打印：

```
eth0: PHY ID 20005c90 at 1 IRQ 0 (stmmac-0:01) active
```

A) **需要先确认硬件是否有异常**，对比 RK 发布的《以太网 PHY 参考电路 V1.0_20150129》，或者找 RK 硬件同事 check 下原理图

B) 参考<引脚复用 IOMUX 确认>**确认下 IOMUX 是否正常**

C) **需要确认 PHY 的供电是否正常，如 VCC_LAN 等电源脚**

D) **如果 PHY 有 reset 脚控制，确认是否正常控制到**

E) **量下 MAC 信号脚的电平，需要工作在 3.3V**

E) **还可以尝试在控制 PHY reset 脚时增加以下 delay 时间试试**

```
+++ b/drivers/net/ethernet/rockchip/gmac/stmmac_platform.c
@@ -358,10 +358,10 @@ static int phy_power_on(bool enable)
                //reset
                if (gpio_is_valid(bsp_priv->reset_io)) {
                        gpio_direction_output(bsp_priv->reset_io,
bsp_priv->reset_io_level);
-                       mdelay(5);
+                       mdelay(100);

gpio_direction_output(bsp_priv->reset_io, !bsp_priv->reset_io_level);
                }
-               mdelay(30);
+               mdelay(100);

        } else {
                //pull down reset
```

4.4.3 引脚复用 IOMUX 确认

GMAC 使用的引脚如下：

MAC_CLK	MAC_TXCLK	MAC_TXEN	MAC_TXD3	MAC_TXD2	MAC_TXD1
MAC_TXD0	MAC_RXCLK	MAC_RXDV	MAC_RXD3	MAC_RXD2	MAC_RXD1
MAC_RXD0			MAC_MDIO	MAC_MDC	MAC_RXER

黄色为 100M(RMII)单独使用的引脚，蓝色为 1000M(RGMII)单独使用的引脚，绿色为两者共同使用的引脚。

GMAC RGMII 接口与其它功能脚复用（无法同时使用），需要确认 IOMUX 状态是否正确。

例如 RK3368 平台出现如下打印（可在 kernel log 中搜索 pinctrl）：

```
rk3368-pinctrl pinctrl.20: pin gpio3-8 already requested by ff680000.pwm; cannot claim for ff290000.eth
```

通过上面 log 看到：**发现 gpio3b0 先被 pwm 申请了，导致无法再被 gmac 申请。**

			2'b11: reserved
1:0	RW	0x0	gpio3b0_sel GPIO3B[0] iomux select 2'b00: gpio 2'b01: mac_txd0 2'b10: pwm_0 2'b11: vop_pwm

需要在 dts 中去查看 pwm 是那个模块在用，可以临时去掉再来调试以太网。

可通过 io 命令直接查看 GRF 寄存器确认：

- RK3126/3128

  RK3128：与 lcdc0 复用，工作在 GMAC 时寄存器值应该如下：

  ```
  io -4 0x200080cc
      200080cc: 0000ffff    // 第 0-3, 6-15bit
  io -4 0x200080d0
      200080d0: 000000ff    // 第 0-7bit
  io -4 0x200080d4
      200080d4: 0000000c    // 第 2-3bit
  ```

  如果 iomux 状态不对，需要按以下排查

  A) gmac io 脚与 lcdc0 复用，需要将将 dts 中 lcd 相关项修改为(**删除红**

色部分)：

```
&lcdc {
status = "okay";
rockchip,fb-win-map = <FB0_WIN0_FB1_WIN1_FB2_WIN2>;
 - pinctrl-0 = <&lcdc0_lcdc>;
 - pinctrl-1 = <&lcdc0_gpio>;
};
```

B) gmac rxd2, rxd3 脚还与 i2c2 口复用，如果 i2c2 口有使用，那么可能出现以下 iomux 申请时错误，导致 gmac iomux 申请失败

rockchip-pinctrl 20008000.pinctrl: pin gpio2-20 already requested by 2005a000.i2c; cannot claim for 2008c000.eth

如果使用 100M PHY，可将 arch/arm/boot/dts/rk312x.dtsi 中的 gmac 中定义的 pinctrl-0 中 iomux 定义从原来的 1000M 修改成 100M

pinctrl-0 = <&gmac_rxdv &gmac_mdio &gmac_txen &gmac_clk &gmac_rxer &gmac_rxd1 &gmac_rxd0 &gmac_txd1 &gmac_txd0 &gmac_mdc>; /* 100M */

pinctrl-0 = <&gmac_rxdv &gmac_txclk &gmac_crs &gmac_rxclk &gmac_mdio &gmac_txen &gmac_clk &gmac_rxer &gmac_rxd1 &gmac_rxd0 &gmac_txd1 &gmac_txd0 &gmac_rxd3 &gmac_rxd2 &gmac_col_gpio &gmac_mdc>; /* 1000M */

- RK3288

RK3288: 与 flash1 及 sdio1 复用，工作在 GMAC 时寄存器值应该如下：

```
io -4 -l 12 0xFF77002C
ff77002c:  00003333 00003333 00003333   // 低 16bit
ff770038:  00003333 // 第 0-11bit
ff77003c:  00000033  // 第 4-7bit
```

- RK3368

RK3368: 与 ISP、I2C4、UART3 复用 ，工作在 GMAC 时寄存器值应该如下：

```
io -4 -l 12 0xff770024
ff770024:  00005415 // 第 0-5bit, 10-15bit
ff770028:  00005155 // 第 0-9bit, 12-13bit
ff77002c:  00003155 // 第 0-3bit, 8-9bit
```

- RK3399

工作在 GMAC 时寄存器值应该如下：

```
io -4 -l 12 0xff77e010

ff77e010:  00005555   // 第 0-15bit

ff77e014:  00001545   // 第 0-13bit

ff77e018:  00000007   // 第 2-3bit
```

4.4.4 工作时钟确认

确认 GMAC 工作主时钟 MAC_CLK 是否正常.

如果用的是 100M PHY，MAC_CLK 的频率是 50M；如果用的是 1000M PHY，MAC_CLK 的频率是 125M.125M clock 必须由 PHY 提供，100M clock 可由 RK 主控提供。

例如 RTL8211E PHY 规格书中关于 125M clock 描述如下：

46	1	CLK125	O/PD	125MHz Reference Clock Generated from Internal PLL. This pin should be kept floating if the 125MHz clock is not be used for MAC.

A) 测量 MAC_CLK 引脚是否有 clock，频率是否正常

1000M 时，如果 PHY 输出的 clock 不是 125M，需要检查 PHY 外围电路是否正常

确认《引脚复用 IOMUX 确认》这一节中 iomux 是否正常

B) clock 的幅度是否达标，一般需要 3.0V 以上

C) 通过 clk_summary 确认

主控端可通过以下 clock tree 信息：cat d/clk/clk_summary 来确认 clock 是否设置正确（注意对应的 enable_cnt 需要为 1，这才表示这个 clock 已经使能）

例如**由 PHY 提供 125M clock**:

```
clock                    enable_cnt   prepare_cnt          rate

gmac_clkin                    1            1            125000000
```

例如**由主控提供 50M clock**:

```
 clock                   enable_cnt   prepare_cnt          rate
```

```
clk_mac_pll                1             1          50000000
```

如果是 3128 芯片，如果这里的 clk_mac_pll 不是 50M，参考上面《引脚复用 IOMUX 确认》这一节修改

注意：如果出现《PHY 初始化失败》这一节中的情况，那么 clock 会被 disable 掉，所以看到的 clock 会是以下情况(对应的 enable_cnt 为 0)：

```
  clock                  enable_cnt  prepare_cnt          rate

clk_mac_pll                0           0          50000000
```

可加以下调试代码，异常时不去关 clock 来调试

```
+++ b/drivers/net/ethernet/rockchip/gmac/stmmac_platform.c
@@ -219,6 +219,7 @@ static int gmac_clk_enable(bool enable) {
        struct bsp_priv * bsp_priv = &g_bsp_priv;
        phy_iface = bsp_priv->phy_iface;

+        enable = 1;
        if (enable) {
                if (!bsp_priv->clk_enable) {
```

4.4.5 无法接收 RX 数据

（表现出的现象是：无法获取到 IP 地址，或者配置静态 IP 地址后无法上网）

正常情况下通过网线连接上路由器（不管有没有拿到 IP 地址），本机会收到很多局域网内的广播包。通过 busybox ifconfig eth0 查看 RX packets 应该不为 0。如果为 0 表示 RX 不通，RX 通路有异常，需要排查。

```
busybox ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 7E:13:69:61:1C:32
          inet6 addr: fe80::7c13:69ff:fe61:1c32/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:83 dropped:0 overruns:0 frame:83
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```
          RX bytes:0 (0.0 B)  TX bytes:2188 (2.1 KiB)
          Interrupt:51
```

A) 参考《时序配置》小节调整 rx_delay，看是否有改善

B) 硬件排查原理图是否正确，RX 通路上的 PHY，变压器，RJ45 座子是否正常

C) clock 精度存在问题，PHY 所用的 25M 晶体的频偏要求在 20ppm 以内

D) 确认下连接的 100M 还是 1000M？（在插入网线时 kernel 会打印相应速率信息），如果是 1000M，尝试连接 100M 看看是否网络会通（如果找不到 100M 的环境，可以将机器直连电脑<电脑上将网卡强制成 100M>，两端都设置静态 IP 地址，看能否 ping 通<注意需要关闭电脑的防火墙，不然可能 ping 不通电脑>）。

4.4.6 TX 发送异常

如果出现 RX 数据正常，但是 TX 一直发送不出去

（表现出的现象是：无法获取到 IP 地址，或者配置静态 IP 地址后无法上网）

```
buysbox ifconfig eth0
// RX 接收数据正常
RX packets:341 errors:0 dropped:0 overruns:0 frame:0
// TX 一直发送不去（TX packets 数目很少）
TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
```

（只要驱动发送成功，TX packets 就会统计，所以 TX packets 看到的是非 0，但是有可能 GMAC 没有成功发送到 PHY，或 PHY 没有成功发送出去）

A) 参考《时序配置》小节调整 tx_delay，看是否有改善

B) 参考《驱动强度配置》小节调整驱动强度

C) 硬件排查原理图是否正确，TX 通路上的 PHY，变压器，RJ45 座子是否正常

D) clock 精度存在问题，PHY 所用的 25M 晶体的频偏要求在 20ppm 以内

4.5 以太网吞吐率异常

例如 TX 或 RX 吞吐率较低，或不稳定。

A) 参考《时序配置》小节调整 tx_delay，看是否有改善

B) 参考《驱动强度配置》小节调整驱动强度

C) 测试下 GMAC 使用的晶体频率是否达标

如果是由 PHY 产生 mac clk(PHY 上使用晶体），需要测试 PHY 上晶体的频偏；

如果是由主控产生 mac clk，则需要测试上主控 24M 晶体的频偏；

4.6 路由器兼容问题

如果连接某些路由器 kernel kmsg 都没有 link up 消息，或者只有偶才能出来；连接另外的路由器可能又正常；

1). 这可能跟 GMAC 使用的 clock 偏频过大有关系。如果使用的是外部 PHY 提供的主时钟，需要测试下 PHY 晶体的偏频，如果使用的是主控提供的主时钟，需要测试下主控 24M 晶体的偏频。

2). 以下 Kernel make menuconfig 中不要选择仍然 PHY 驱动，使用默认的 genenal phy 驱动就可：

| Prompt: PHY Device support and infrastructure
| Defined at drivers/net/phy/Kconfig:5
| Depends on: !S390 && NETDEVICES [=y]
| Location:
| -> Device Drivers
| -> Network device support (NETDEVICES [=y])

3). 跟 PHY 的 EEE 功能有关系，可尝试关掉 EEE（不同的 PHY 需要操作不同的寄存器）

打上 ftp 上的补丁 disable_eee.diff

4.7 以太网断线问题

例如接着网线的情况下随机出现 link down(kmsg 中打印)，或者接上网线时随机出现无 link up，可参考上一节<路由器兼容问题>排查。

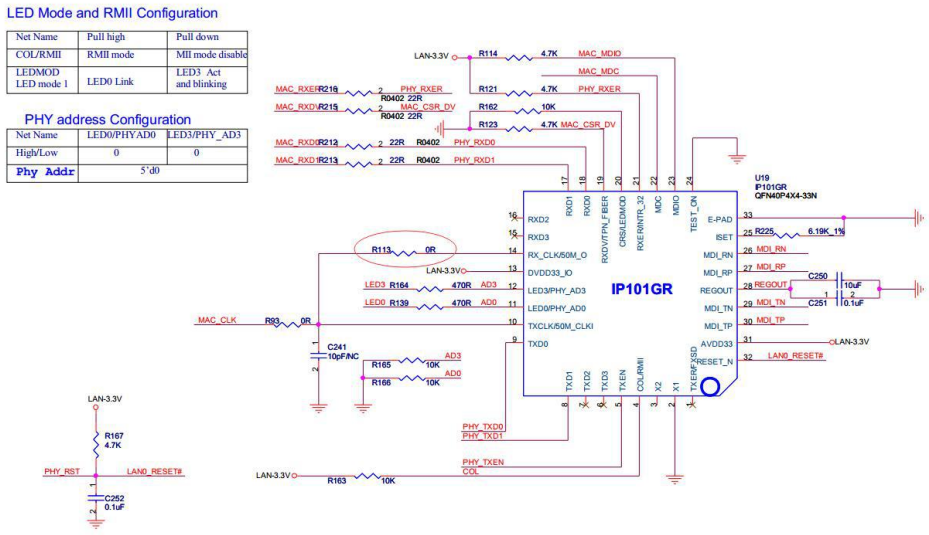
4.8 100M（百兆）正常、1000M(千兆)上不了网（获取不到 IP 地址）

A) 参考《时序配置》小节调整 rx_delay, tx_delay，看是否有改善

B) 参考《驱动强度配置》小节调整驱动强度

4.9 使用 IP101GR PHY 注意

当时钟由 RK 主控提供时，下图中 R113 这个电阻要去掉，不然就 PHY 会有时钟出来，与主控分出来的时钟冲突。可能会出现概率性开机后 PHY 无法工作，无法获取到 IP 地址。



4.10 RK3399 以太网 MAC 地址变化问题

打上 ftp 上的补丁《RK3399(Kernel 4.4) WiFi BT 以太网 MAC 地址自定义补丁.rar》

4.11 PHY Link/Active LED 灯异常问题

打上 ftp 上的补丁《PHY LinkActive LED 灯异常补丁》

4.12 开机概率性识别不到网线

也就是插入网线时 kmsg 无"Link is up"打印，重启机器可能就恢复正常了。

A) 请参考 4.13 节《4.13 写 PHY 寄存器异常》。

B) 确认下 MAC_MDC/MAC_MDIO 引脚的 iomux 状态，看下是否被切换成了 GPIO。

可通过系统类似节点查看 iomux 状态：

```
cat ./sys/kernel/debug/pinctrl/20008000.pinctrl/*
```

4.13 写 PHY 寄存器异常

有遇到过 PHY 寄存器 16 写不进去，比如写入 0xAA，但实际读出来不是 0xAA，可以尝试以下《解决 MDC clock 不准的问题》补丁：

kernel 3.10:

```
--- a/kernel/drivers/net/ethernet/rockchip/gmac/stmmac_main.c
+++ b/kernel/drivers/net/ethernet/rockchip/gmac/stmmac_main.c
@@ -2836,7 +2857,7 @@ struct stmmac_priv *stmmac_dvr_probe(struct device *device,

                priv->mdio_registered = false;

-               priv->stmmac_clk = ((struct bsp_priv *)(priv->plat->bsp_priv))->clk_mac;
+               priv->stmmac_clk = ((struct bsp_priv *)(priv->plat->bsp_priv))->pclk_mac;
                if (IS_ERR(priv->stmmac_clk)) {
                        pr_warn("%s: warning: cannot get CSR clock\n", __func__)
;
                        goto error_clk_get;
```

kernel 4.4:

```
diff --git a/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
b/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
index e230519..7b7f696 100644
--- a/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
+++ b/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
@@ -165,7 +165,7 @@
{
u32 clk_rate;
- clk_rate = clk_get_rate(priv->stmmac_clk);
+ clk_rate = clk_get_rate(priv->pclk);
/* Platform provided default clk_csr would be assumed valid
* for all other cases except for the below mentioned ones.
@@ -2886,7 +2886,7 @@ int stmmac_dvr_probe(struct device *device,
}
clk_prepare_enable(priv->stmmac_clk);
- priv->pclk = devm_clk_get(priv->device, "pclk");
+ priv->pclk = devm_clk_get(priv->device, "pclk_mac");
if (IS_ERR(priv->pclk)) {
if (PTR_ERR(priv->pclk) == -EPROBE_DEFER) {
ret = -EPROBE_DEFER;
```

4.14 WiFi 以太网共存问题

参考 ftp 上的补丁《WiFi 以太网络共存补丁》

4.15 USB 以太网异常排查

4.15.1以太网无法使用问题排查

第一步：

USB 以太网通过 USB HOST 口连接到主控，先确认是否有以下 USB 以太网设备枚举到的打印（以 RTL8152 为例）：

```
usb 2-1: new high speed USB device number 2 using usb20_host
usb 2-1: New USB device found, idVendor=0bda, idProduct=8152 //这里打印出枚举到的设备信息
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-1: Product: USB 10/100 LAN
usb 2-1: Manufacturer: Realtek
usb 2-1: SerialNumber: 00E04C360001
Ethernet Device, 00:e0:4c:36:00:01
```

如果没有打印以上信息，证明 USB 枚举失败，需要排查：

1) USB HOST 口工作是否正常，可以接鼠标等设备测试；

2) USB HOST 口供电是否足够；

3) 如果是接在 HUB，确认 HUB 工作是否正常；

第二步：

如果 USB 设备被正确枚举到了，那么可通过以下以下命令查看 eth0 接口状态：

```
busybox ifconfig eth0
        eth0      Link encap:Ethernet  HWaddr 00:E0:4C:36:00:01
        // 确认是否有获取到以下 IP 地址
                  inet      addr:192.168.0.120              Bcast:192.168.0.255
Mask:255.255.255.0
                  inet6 addr: fe80::2e0:4cff:fe36:1/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:577 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:465 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:751796 (734.1 KiB)  TX bytes:39502 (38.5 KiB)
```

如果无法拿到 IP 地址，可能是没有 MAC 地址的问题，可使用 ftp 上的补丁《rtl8152

解决 MAC 地址为空补丁》

4.15.2USB 以太网不稳定

例如在播放网络视频时，容易断线，从 log 来看到有以下 USB 异常信息：

```
usb 2-1: USB disconnect, device number 2
```

A) 可能是 USB HOST 口供电不足造成的，需要硬件上修改

B) 也可能是 USB HOST 不稳定造成的，需要 USB 部分硬件软件排查下

4.15.3MAC 地址为 0 导致异常进不了 Android 系统

如果使用 USB 以太网芯片（非 Dongle)可能没有 MAC 地址

可加入如下补丁，先从 flash 保留区读取 MAC 地址，如果读不到，会随机产生 MAC 地址。

可使用 ftp 上的补丁《rtl8152 解决 MAC 地址为空补丁》固定 MAC 地址。

4.15.4USB 以太网注册成了 eth1 导致无法使用

通过 busybox ifconfig 查看出现 eth0 与 eth1 那个以太网接口。

这是因为系统中存在了 GMAC，它会注册成 eth0，导致在插入 usb 以太网时，注册成了 eth1。但是上层只会监听 eth0，所以导致 usb 以太网无法使用。

需要去掉 GMAC 的支持，才能使用 usb 以太网，具体修改如下。

在板级 dts 中 disable 掉 gmac：

```
&gmac {
    status = "disabled";
};
```

5 附录

5.1 已验证以太网 PHY 芯片列表

供应商	型号	接口	备注
DAVICOM	DM9161	RMII	
SMSC	LAN8720	RMII	
REALTEK	RTL8201F	RMII	（RK2918/RK2908 不支持）
	RTL8211E/F	RGMII	
ATHEROS	AR8032	RMII	（RK2918/RK2908 不支持）
IC PLUS	IP101	RMII	（RK2918/RK2908 不支持）
TI	DP83848C	RMII	

5.2 已验证 USB 以太网卡芯片列表

如果使用的是 Kernel 3.0 平台，已验证过以下网卡

供应商	型号	备注
DAVICOM	DM9601/DM9620	
	SR9700	
REALTEK	RTL8152B	

如果使用的是 Kernel 3.10 平台，默认 kernel 已经支持很多 USB 以太网卡，具体可见以下配置：

```
---------------------------------------------------------------------------
  <*> USB CATC NetMate-based Ethernet device support
  <*> USB KLSI KL5USB101-based ethernet device support
  <*> USB Pegasus/Pegasus-II based ethernet device support
  <*> USB RTL8150 based ethernet device support
  <*> Realtek RTL8152 Based USB 2.0 Ethernet Adapters
  <*> Multi-purpose USB Networking Framework
  <*>   ASIX AX88xxx Based USB 2.0 Ethernet Adapters
  <*>   ASIX AX88179/178A USB 3.0/2.0 to Gigabit Ethernet
  -*-   CDC Ethernet support (smart devices such as cable modems)
  <*>   CDC EEM support
  -*-   CDC NCM support
  <*>   CDC MBIM support
  <*>   Davicom DM9601 based USB 1.1 10/100 ethernet devices
  <*>   Davicom DM9620 based USB 2.0 10/100 ethernet devices
  <*>   SMSC LAN75XX based USB 2.0 gigabit ethernet devices
  <*>   SMSC LAN95XX based USB 2.0 10/100 ethernet devices
  <*>   GeneSys GL620USB-A based cables
  <*>   NetChip 1080 based cables (Laplink, ...)
  <*>   Prolific PL-2301/2302/25A1 based cables
  <*>   MosChip MCS7830 based Ethernet adapters
  <*>   Host for RNDIS and ActiveSync devices
  <*>   Simple USB Network Links (CDC Ethernet subset)
  [*]     ALi M5632 based 'USB 2.0 Data Link' cables
  [*]     AnchorChips 2720 based cables (Xircom PGUNET, ...)
  [*]     eTEK based host-to-host cables (Advance, Belkin, ...)
  [*]     Embedded ARM Linux links (iPaq, ...)
  [*]     Epson 2888 based firmware (DEVELOPMENT)
  [*]     KT Technology KC2190 based cables (InstaNet)
  <*>   Sharp Zaurus (stock ROMs) and compatible
  <*>   Conexant CX82310 USB ethernet port
  <*>   Samsung Kalmia based LTE USB modem
  <*>   QMI WWAN driver for Qualcomm MSM based 3G and LTE modems
  <*> Option USB High Speed Mobile Devices
  <*> Intellon PLC based usb adapter
  <*> Apple iPhone USB Ethernet driver
  <*> USB-to-WWAN Driver for Sierra Wireless modems
  < > LG VL600 modem dongle
```