密级状态：绝密(　　)　　秘密(　　)　　内部资料(　　)　　公开(√)

Rockchip BOX 以太网开发指南

(技术研发部，电视事业部)

文件状态： ［　］草稿 ［√］正式发布 ［　］正在修改	文件标识：	Rockchip BOX 以太网开发指南
	当前版本：	2.2
	作　　者：	陈智
	完成日期：	2014-09-25
	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

目 录

1 概述

本文基于 Rockchip BOX SDK 进行描述。 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 章介绍以太网在 Android 上的相关调用接口。附录将会列出所支持的芯片列表。

2 以太网 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，3288 之前的芯片所配套的 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。所以理论上，如果不需要使用 PHY 厂家提供的自定义的寄存器配置实现一些个性化的功能，那么 PHY 的驱动就不需要修改。从 3288 之后的 SDK 开始，打开了各个 PHY 厂家在 Linux 上提供驱动的配置，但是基本上，各家的驱动差异很小，基本上也是调用通用驱动的接口。

2.1 接口

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
RK2918	√	√	
RK2908	√	√	
RK3066		√	
RK3188		√	
RK3288		√	√
RK3036		√	
RK3128		√	√

（注： 2918/2908 虽然硬件上有 MII 接口，但驱动上并不支持）

2.2 Kernel 配置

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
RK3036	VMAC	3.10
RK3128	GMAC	3.10

2.2.1 RK3066

2.2.1.1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。

2.2.1.2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 )
```

2.2.1.3MAC 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。

2.2.2 RK3188

同 RK3066，见 2.2.1

2.2.3 RK3288

2.2.3.1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 2.2.3.2 的说明。

TX/RX 线上的延时值：请参考 2.2.3.3

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 主控分出的 50M clock 可能不精准，会造成 GMAC 丢包或无法工作。

1000M PHY 配置

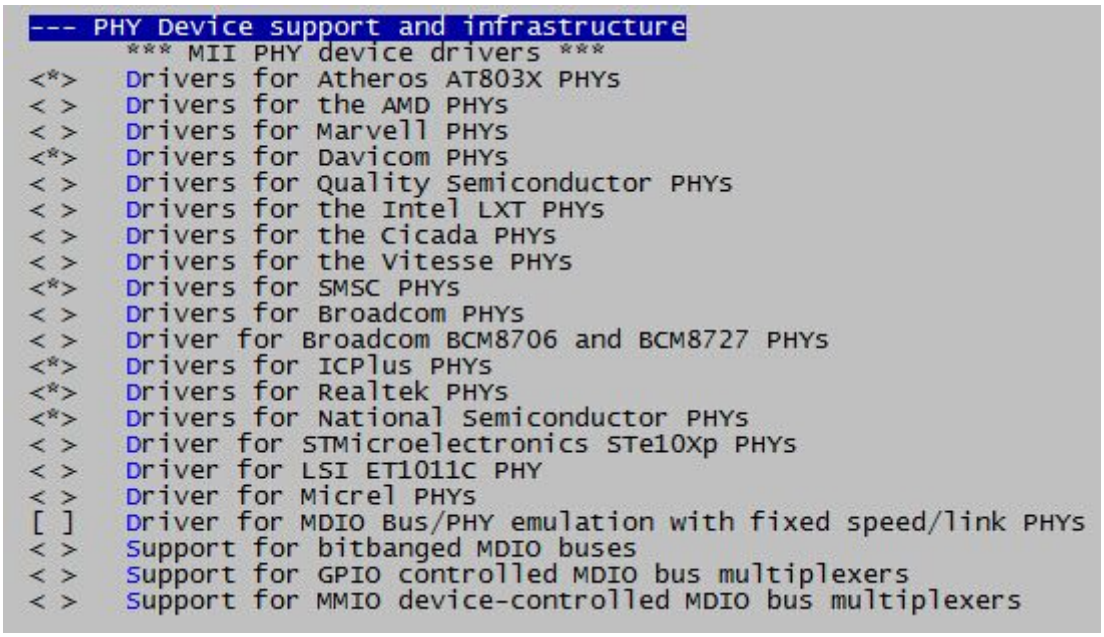
```
&gmac_clkin {
    clock-frequency = <125000000>;        --> 修改成 125M
};
&gmac {
//          power-gpio = <&gpio0 GPIO_A6 GPIO_ACTIVE_HIGH>;
            reset-gpio = <&gpio4 GPIO_B0 GPIO_ACTIVE_LOW>;
            phy-mode = "rgmii";                     ---> 修改成 rgmii
            clock_in_out = "input";                 ---> 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 主控分出的 125M clock 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
```



2.2.3.2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，这个概念可以参考 2.2.1.2。

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>,
```

2.2.3.3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的情况，可以适当调整这两个值，范围是 0x0—0x7F. 在实际的调整过程中，可以在上述这两个值的基础上适当加减，正常不应该会有太大的偏差。

2.2.3.4MAC 地址

目前 32 上对 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，则使用随机生成的地址。烧写工具在 SDK 中会附带。

2.2.4 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 2.2.3.1
MAC 驱动代码位于 drivers/net/ethernet/rockchip/vmac/rk29_vmac.c

2.2.5 RK3128

同 RK3288, 见 2.2.3

注：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 上不需要进行

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

和 PHY 情况类似，Linux kernel 中有 USB 以太网卡的通用驱动，只需要打开如下配置即可

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

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

此处 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，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！

3.3 字节对齐问题

注意：此部分只针对 kernel 3.0 的平台

由于 RK 芯片的 USB 驱动内部没有处理字节对齐问题，所以所有调用 USB 传输接口的驱动必须自行处理好字节对齐。若发现 kernel 的 log 中打出 USB 相关报错，请确认如下补丁是否已经打上。SDK 默认已经打上下述补丁。此外，若使用供应商提供的驱动代码，也需要参照下述补丁在驱动中调用 USB 传输接口的地方做相应处理。

```
diff --git a/kernel/drivers/net/usb/usbnet.c b/kernel/drivers/net/usb/usbnet.c
old mode 100644
new mode 100755
index d1ab169..809f416
--- a/kernel/drivers/net/usb/usbnet.c
+++ b/kernel/drivers/net/usb/usbnet.c
@@ -1103,6 +1103,28 @@ netdev_tx_t usbnet_start_xmit (struct sk_buff *skb,
                                }
                        }
                }
+
+
+               length = ((unsigned long)skb->data) & 0x3;
+               if (length) {
+                       if (skb_cloned(skb) ||
+                           ((skb_headroom(skb) < length) &&
+                            (skb_tailroom(skb) < (4-length)))) {
+                               struct sk_buff *skb2;
+                               /* copy skb with proper alignment */
+                               skb2 = skb_copy_expand(skb, 0, 4, GFP_ATOMIC);
+                               dev_kfree_skb_any(skb);
+                               skb = skb2;
+                               if (!skb)
+                                       goto drop;
+                       } else {
+                               /* move data inside buffer */
+                               length = ((skb_headroom(skb) >= length) ? 0 : 4)-length;
+                               memmove(skb->data+length, skb->data, skb->len);
+                               skb_reserve(skb, length);
+                       }
+               }
+
                length = skb->len;
```

```
        if (!(urb = usb_alloc_urb (0, GFP_ATOMIC))) {
```

4 Android 接口

4.1 Android 4.2 4.4(Android 5.0 之前版本）

下述接口代码存放于 frameworks/base/ethernet

public int getEthernetConnectState()

获取连接状态

0:：未连接；1：连接中；2：已连接

public int getEthernetIfaceState()

获取以太网接口状态（是否使能以太网）

0：未使能；1：已使能

public int getEthernetCarrierState()

获取是否有载波信号（可用来判断网线是否已插入）

0：无载波信号；1：有载波信号

public boolean setEthernetEnabled(boolean enabled)

打开/关闭以太网

public String getEthernetIfaceName()

获取以太网接口名， 一般为“eth0”

public String getEthernetHwaddr(String iface)

获取 MAC 地址

此外还可以在应用程序中监听如下 Intent 消息以获取接口和连接状态

```
public static final String ETHERNET_STATE_CHANGED_ACTION =
"android.net.ethernet.ETHERNET_STATE_CHANGED";
public static final String EXTRA_ETHERNET_STATE = "ethernet_state";
public static final String ETHERNET_IFACE_STATE_CHANGED_ACTION =
"android.net.ethernet.ETHERNET_IFACE_STATE_CHANGED";
public static final String EXTRA_ETHERNET_IFACE_STATE = "ethernet_iface_state";
public static final int ETHER_STATE_DISCONNECTED=0;
public static final int ETHER_STATE_CONNECTING=1;
public static final int ETHER_STATE_CONNECTED=2;
public static final int ETHER_IFACE_STATE_DOWN = 0;
public static final int ETHER_IFACE_STATE_UP = 1;
```

此外，以太网相关接口的具体调用，包括静态 IP 的设置方法等，可参考如下目录中的代码

packages/apps/Settings/src/com/android/settings/ethernet

4.2 Android 5.0

Android 5.0 中 google 集成了以太网框架代码，具体在：

frameworks/opt/net/ethernet/

客户如果想自己开发以太网设置部分代码，可参考：

src/com/android/settings/EthernetSettings.java

src/com/android/settings/SettingsPreferenceFragment.java

src/com/android/settings/etherent_static_ip_dialog.java

src/com/android/settings/getStaticIpInfo.java

5 以太网常见问题排查

5.1 3.10 版本 Kernel 部分

5.1.1 phy 寄存器读写调试

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
```

5.1.2 以太网无法正常工作

① 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 MAC1_CLK 是否有从 PHY 供给主控：

使用 100M PHY 时，其频率是 50M

使用 100M PHY 时，其频率是 125M

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。

② 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
```

PHY 正常识别会有如下打印：

```
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》

B) 需要确认 PHY 的供电是否正常，复位信号是否正常

C) 还可以尝试增加以下 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
```

③ **IOMUX 确认**

注意：GMAC RGMII 接口与其它功能脚利用，需要确认 IOMUX 状态是否正确，可通过 io 命令直接查看 GRF 寄存器确认：

RK3128：与 lcdc0 复用

```
io -4 0x200080cc
    200080cc: 0000ffff    // 低 16bit
io -4 0x200080d0
    200080d0: 000000ff    // 低 8bit
io -4 0x200080d4
    200080d4: 0000000c    // 低 4bit
```

RK3288：与 flash1 及 sdio1 复用

```
io -4 -l 20 0xFF77002C
```

```
ff77002c:  00003333 00003333 00003333 00003333  // 低 16bit
ff77003c:  00000033  // 低 8bit
```

④ **确认 GMAC 工作主时钟是否正常**

确认时钟是否是有正常产生 ，时钟具体频率及来源如下表：

	PHY	时钟来源	时间频率
RK3288	1000M	PHY	125M
RK3288	100M	RK3288 或 PHY (建议使用 PHY 提供)	50M
RK3128	1000M	PHY	125M
RK3128	100M	RK3128	50M

clock 的幅度是否达标，一般需要 3.0V 以上

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
```

注意：如果出现 2 小节中提到的“**PHY 初始化异常**”，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

```
+++ 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) {
```

⑤ **如果出现无法接收 RX 数据**

```
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
```

那么可能是 clock 精度存在问题，、晶体的频偏要求在 10ppm 以内。

⑥ **如果出现 RX 数据正常，但是 TX 一直发送不出去**

或者无法获取 IP 地址

```
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:22 errors:0 dropped:0 overruns:0 carrier:0
```

首先确认 tx 硬件通路是否正常？其次尝试调整 dts 中的 tx_delay = <0x30>;

---> TX 线上的延时值，这个值的具体含义见《2.2.3.3 时序配置》。

5.1.3 USB 以太网异常排查

5.1.3.1以太网无法使用问题排查

第一步：

USB 以太网接到通过 USB HOST 口连接到主控，先确认是否有以下 USB 设备枚举到的打印：

```
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)
```

如果没有正确获取到 IP 地址，那么可能是所在局域网有问题，可以在设置中设置成静态 IP 地址试试。

5.1.3.2USB 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
```

这个可能是 USB HOST 口供电不足造成的，需要硬件上修改。

5.1.3.3MAC 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。

```
diff --git a/drivers/net/usb/usbnet.c b/drivers/net/usb/usbnet.c
old mode 100644
new mode 100755
index 3d50e7d..1af08ed
--- a/drivers/net/usb/usbnet.c
+++ b/drivers/net/usb/usbnet.c
@@ -1595,6 +1595,26 @@ usbnet_probe (struct usb_interface *udev, const
struct usb_device_id *prod)
        if ((dev->driver_info->flags & FLAG_WWAN) != 0)
                SET_NETDEV_DEVTYPE(net, &wwan_type);
```

```
+#if 1
+    if ((net->dev_addr[0] == 0x00) &&
+                (net->dev_addr[1] == 0x00) &&
+                (net->dev_addr[2] == 0x00) &&
+                (net->dev_addr[3] == 0x00) &&
+                (net->dev_addr[4] == 0x00) &&
+                (net->dev_addr[5] == 0x00) )
+    {
+                extern int eth_mac_idb(u8 *eth_mac);
+
+                printk("[MAC] try mac addr from idb first.\n");
+                eth_mac_idb(net->dev_addr);
+                if (!is_valid_ether_addr(net->dev_addr))
+                {
+                        printk("[MAC] no mac addr in idb, use random mac
addr.\n");
+                        random_ether_addr(net->dev_addr);
+                }
+     }
+#endif
+
        status = register_netdev (net);
```

5.1.3.4USB 以太网注册成了 eth1 导致无法使用

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";
};
```

6 附录

6.1 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）
	RTL8211F	RGMII	
ATHEROS	AR8032	RMII	（RK2918/RK2908 不支持）
IC PLUS	IP101	RMII	（RK2918/RK2908 不支持）
TI	DP83848C	RMII	（RK3288 不支持）

6.2 已验证 USB 以太网卡芯片列表

供应商	型号	备注
DAVICOM	DM9620	
SMSC	LAN9500	
REALTEK	RTL8152B	