RTL8367RB-CG/RTL8363NB-VB-CG SWITCH 调试

一、硬件配置注意事项：

8367RB-CG/8363NB-VB-CG 默认可以通过 3 种方式去配制 switch

- 1: IIC HOST(gpio 模拟 SCK/SDA 读外部 E2PROM)；
- 2: SPI Slave ；
- **3:MIIM 管理接口(MDC/MDIO)．我们目前配合软件调试都采用第 3 种；**

硬件接法：

- **RTL8367RB-CG 硬件连接及 configuration strapping**

8367RB-CG 所支持的总线模式只有 RGMII 是 RK 平台所支持的，MII 平台没有接口支持，硬件接法是主控 MAC 与 switch MAC 通过 RGMII 连接 ，所以硬件上需要交叉接，即 switch 端的 RGMII_TXD[3:0]接到 RK 主控端的 MAC_RXD[3:0]， switch 端的 RGMII_RXD[3:0]接到主控端的 MAC_TXD[3:0].具体的硬件接法见下图。

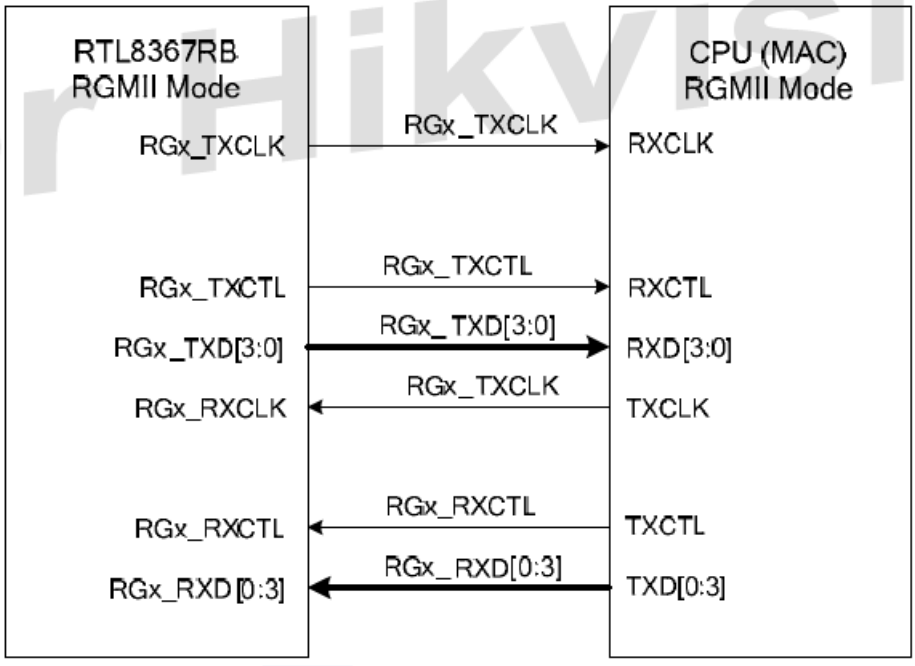


Figure 17. RGMII Mode Interface Signal Diagram

10.4. General Purpose Interface

The RTL8367RB supports two extension interfaces. The interface function mux is summarized in Table 18 and Table 19. The Extension GMAC1 and Extension GMAC2 of the RTL8367RB support RGMII, MII MAC mode, or MII PHY mode via register configuration.

Table 18. RTL8367RB Extension Port 1 Pin Definitions

Pin No.	Extension Interface	Type	RGMII	MII MAC Mode	MII PHY Mode
56	E1_CRS	I	-	M1M_CRS	-
57	E1_DO3	O	RG1_TXD3	M1M_TXD3	M1P_RXD3
58	E1_DO2	O	RG1_TXD2	M1M_TXD2	M1P_RXD2
59	E1_DO1	O	RG1_TXD1	M1M_TXD1	M1P_RXD1
60	E1_DO0	O	RG1_TXD0	M1M_TXD0	M1P_RXD0
61	E1_DOEN	O	RG1_TXCTL	M1M_TXEN	M1P_RXDV
62	E1_GCLK	O	RG1_TXCLK	M1M_TXCLK	M1P_RXCLK
63	E1_DICLK	I	RG1_RXCLK	M1M_RXCLK	M1P_TXCLK
64	E1_DIDV	I	RG1_RXCTL	M1M_RXDV	M1P_TXEN
65	E1_DI0	I	RG1_RXD0	M1M_RXD0	M1P_TXD0
66	E1_DI1	I	RG1_RXD1	M1M_RXD1	M1P_TXD1
67	E1_DI2	I	RG1_RXD2	M1M_RXD2	M1P_TXD2
68	E1_DI3	I	RG1_RXD3	M1M_RXD3	M1P_TXD3

Table 19. RTL8367RB Extension Port 2 Pin Definitions

Pin No.	Extension Interface	Type	RGMII	MII MAC Mode	MII PHY Mode
40	E2_CRS	I	-	M2M_CRS	-
41	E2_DO3	O	RG2_TXD3	M2M_TXD3	M2P_RXD3
42	E2_DO2	O	RG2_TXD2	M2M_TXD2	M2P_RXD2
43	E2_DO1	O	RG2_TXD1	M2M_TXD1	M2P_RXD1
44	E2_DO0	O	RG2_TXD0	M2M_TXD0	M2P_RXD0
45	E2_DOEN	O	RG2_TXCTL	M2M_TXEN	M2P_RXDV
46	E2_GCLK	O	RG2_TXCLK	M2M_TXCLK	M2P_RXCLK
47	E2_DICLK	I	RG2_RXCLK	M2M_RXCLK	M2P_TXCLK
48	E2_DIDV	I	RG2_RXCTL	M2M_RXDV	M2P_TXEN
49	E2_DI0	I	RG2_RXD0	M2M_RXD0	M2P_TXD0
50	E2_DI1	I	RG2_RXD1	M2M_RXD1	M2P_TXD1
51	E2_DI2	I	RG2_RXD2	M2M_RXD2	M2P_TXD2
52	E2_DI3	I	RG2_RXD3	M2M_RXD3	M2P_TXD3

硬件接好后，软件不需要做任何动作，通过网口连接到 pc 端，拔插网线，正常情况下在 pc 端都会显示网络已连接和网络断开的提示。如果没有的话，先检查硬件 configuration strapping pin 脚状态和接线方式是否正确。

见下图截图画红框选中项，未标选的依据 led 实际功能需求选择上下拉。

7.4. Configuration Strapping Pins

Table 8. Configuration Strapping Pins

Pin Name	Pin No.	Type	Description
EEPROM_MOD/ P4LED0	73	I/O_{PU}	EEPROM Mode Selection. Pull Up: EEPROM 24Cxx Size greater than 16Kbits (24C32~24C256) Pull Down: EEPROM 24Cxx Size less than or equal to 16Kbit (24C02~24C16). *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 39 for more details.*
DIS_SPIS/ P4LED2	72	I/O_{PU}	SPI Slave Management Interface Selection. Pull Up: Disable SPI Slave Management Interface Pull Down: Enable SPI Slave Management Interface *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 39 for more details.*
RESERVED/ P3LED2	76	I/O_{PU}	Internal Use/Reserved. *Note: This pin must be kept floating, or pulled high via an external 4.7k ohm resistor upon power on or reset.* *When pulled high, the LED output polarity will be low active. See section 9.19 LED Indicators, page 39 for more details.*
RESERVED/ P3LED0	77	I/O_{PU}	Internal Use/Reserved. *Note: For normal operation, this pin must be pulled low via an external 4.7k ohm resistor upon power on or reset.* *When pulled low, the LED output polarity will be high active. See section 9.19 LED Indicators, page 39 for more details.*

Ping82 是 PHY_ADDR 选择， 上拉 4.7K phy_id 定为 29， 下拉 4.7K phy_id 0；注意软件中的设定要与硬件匹配，否则无法识别，目前提供的代码默认按上拉 4.7K 处理。后面软件部分说明。

MID29/ P1LED0	82	I/O_{PU}	Select MID29. Pull Up: MII Management Interface PHY ID is 29 Pull Down: MII Management Interface PHY ID is 0 *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 43 for more details.*

Pin Name	Pin No.	Type	Description
DIS_8051/ P2LED2	78	I/O_{PU}	Disable Embedded 8051. Pull Up: Disable embedded 8051 Pull Down: Enable embedded 8051 *Note 1: The strapping pin DISAUTOLOAD and DIS_8051 are for power on or reset initial stage configuration. Refer to Table 9 Configuration Strapping Pins (DISAUTOLOAD and DIS_8051), page 22 for details.* *Note 2: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicator, page 39 for more details.*
DISAUTOLOAD/ P2LED0	79	I/O_{PU}	Disable EEPROM Autoload. Pull Up: Disable EEPROM autoload Pull Down: Enable EEPROM autoload *Note 1: The strapping pin DISAUTOLOAD and DIS_8051 are for power on or reset initial stage configuration. Refer to Table 9 Configuration Strapping Pins (DISAUTOLOAD and DIS_8051), page 22 for details.* *Note 2: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 39 for more details.*
RESERVED/ P1LED2	81	I/O_{PU}	Internal Use/Reserved. *Note: This pin must be kept floating, or pulled high via an external 4.7k ohm resistor upon power on or reset.* *When pulled high, the LED output polarity will be low active. See section 9.19 LED Indicators, page 39 for more details.*
RESERVED/ P1LED0	82	I/O_{PU}	Internal Use/Reserved. *Note: For normal operation, this pin must be pulled low via an external 4.7k ohm resistor upon power on or reset.* *When pulled low, the LED output polarity will be high active. See section 9.19 LED Indicators, page 39 for more details.*
EN_PHY/ P0LED2	84	I/O_{PU}	Enable Embedded PHY. Pull Up: Enable embedded PHY Pull Down: Disable embedded PHY *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 39 for more details.*
SMI_SEL/ LED_CK/ P0LED0	86	I/O_{PU}	EEPROM SMI/MII Management Interface Selection. Pull Up: EEPROM SMI interface when DIS_SPIS = 1 Pull Down: MII Management interface when DIS_SPIS = 1 *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.19 LED Indicators, page 39 for more details.*

● RTL8363NB-VB-CG 硬件连接及 configuration strapping

RTL8363NB-VB-CG 所支持的总线模式有 RGMII/RMII 是 RK 平台所支持的， MII 平台没有接口支持。接 RGMII 总线时对应 switch 端的 RGMII_TXD[3:0]接到 RK 主控端的 MAC_RXD[3:0]，switch 端的 RGMII_RXD[3:0]接到主控端的 MAC_TXD[3:0]；

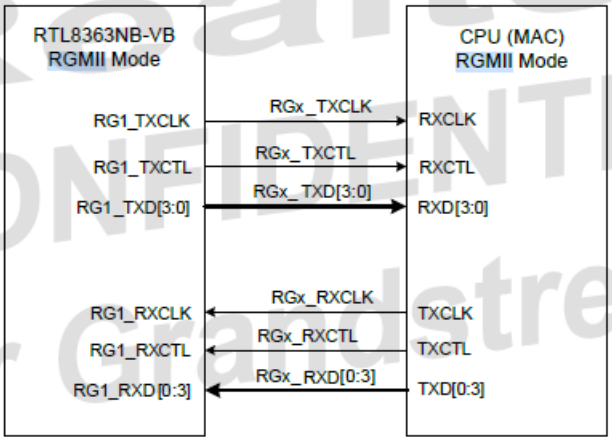


Figure 21. RGMII Mode Interface Signal Diagram

Table 18. RTL8363NB-VB Extension Port 1 Pin Definitions

Pin No.	Type	RGMII	(T)MII MAC Mode	(T)MII PHY Mode	RMII Mode
41	I_{PD}	-	M1M_CRS	-	-
42	O	RG1_TXD3	M1M_TXD3	M1P_RXD3	-
43	O	RG1_TXD2	M1M_TXD2	M1P_RXD2	-
44	O	RG1_TXD1	M1M_TXD1	M1P_RXD1	RM1M_TXD1
45	O	RG1_TXD0	M1M_TXD0	M1P_RXD0	RM1M_TXD0
46	O	RG1_TXCTL	M1M_TXEN	M1P_RXDV	RM1M_TXEN
47	I/O	RG1_TXCLK	M1M_TXCLK	M1P_RXCLK	RM1M_CLK
48	I/O	RG1_RXCLK	M1M_RXCLK	M1P_TXCLK	-
49	I	RG1_RXCTL	M1M_RXDV	M1P_TXEN	RM1M_CRS_DV
50	I	RG1_RXD0	M1M_RXD0	M1P_TXD0	RM1M_RXD0
51	I	RG1_RXD1	M1M_RXD1	M1P_TXD1	RM1M_RXD1
52	I	RG1_RXD2	M1M_RXD2	M1P_TXD2	-
53	I	RG1_RXD3	M1M_RXD3	M1P_TXD3	-

当接 RMII 总线时，跟平台端还是 MAC 与 MAC 的连接方式， switch 端的 RMII_TXD[1:0]接到 RK 主控端的 MAC_RXD[1:0]，switch 端的 RMII_RXD[1:0]接到主控端的 MAC_TXD[1:0]

7.2.3. RMII Pins

The Extension GMAC1 of the RTL8363NB-VB supports RMII interface connection with an external MAC or PHY device when register configuration is set to RMII mode interface. The RMII interface can be configured as RMII Clock Input mode or RMII Clock Output mode via register.

Table 6. Extension GMAC1 RMII Pins (RMII Clock Input Mode or RMII Clock Output Mode)

Pin No.	Type	RMII	Description
44	O	RM1M_TXD1	RMII 1 TXD1
45	O	RM1M_TXD0	RMII 1 TXD0
46	O	RM1M_TXEN	RMII 1 TXEN
47	I/O	RM1M_CLK	RMII 1 Reference 50MHz Clock In RMII Clock Input Mode, this pin is Input. In RMII Clock Output Mode, this pin is Output.
49	I	RM1M_CRS_DV	RMII 1 CRS_DV
50	I	RM1M_RXD0	RMII 1 RXD0
51	I	RM1M_RXD1	RMII 1 RXD1

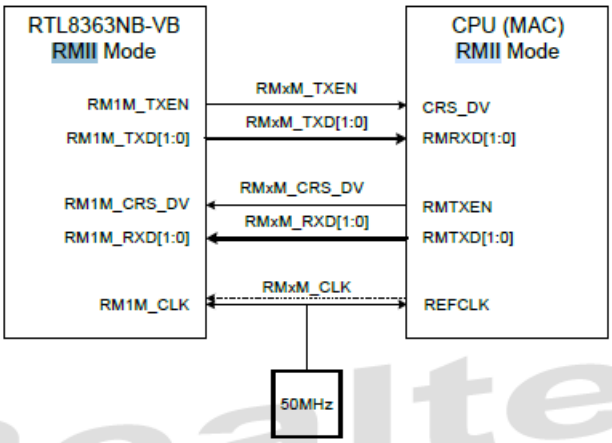


Figure 24. Signal Diagram of RMII Mode (Clock Input Mode)

Note: The termination network on the transmitter (for example, a serial termination resistor) may be used for better signal integrity.

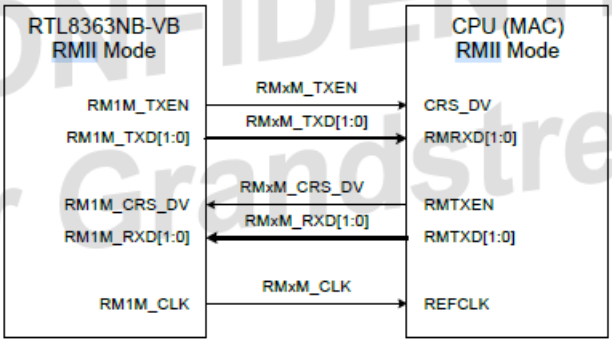


Figure 25. Signal Diagram of RMII Mode (Clock Output Mode)

7.4. Configuration Strapping Pins

Table 8. Configuration Strapping Pins

Pin Name	Pin No.	Type	Description
DISAUTOLOAD / P3LED0	66	I/O_{PU}	Disable EEPROM Auto load. Pull Up: Disable EEPROM auto download upon power on or reset. Pull Down: Enable EEPROM auto download upon power on or reset. *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.17 LED Indicators, page 33 for more details.*
SMI_SEL_1 / P1LED2	68	I/O_{PU}	External CPU Access Interface Selection. SMI_SEL[1:0]: 00: Slave LSB I2C mode 01: Slave MSB I2C mode 10: Slave MII Management MDC/MDIO 11: Realtek Slave I2C-like mode *Note: These pins must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When these pins are pulled high, the LED output polarity will change from high active to low active. See section 9.17 LED Indicators, page 33 for more details.*
SMI_SEL_0	72	I/O_{PU}	
MID29 / P1LED0	69	I/O_{PU}	Select MID29. Pull Up: Slave MII Management Interface PHY ID is 29 Pull Down: Slave MII Management Interface PHY ID is 0 *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.* *When this pin is pulled low, the LED output polarity will be high active. When this pin is pulled high, the LED output polarity will change from high active to low active. See section 9.17 LED Indicators, page 33 for more details.*
EN_PHY	71	I/O_{PU}	Enable Embedded PHY. Pull Up: Enable embedded PHY upon power on or reset. Pull Down: Disable embedded PHY upon power on or reset. *Note: This pin must be kept floating, or pulled high or low via an external 4.7k ohm resistor upon power on or reset.*

二、软件配置

目前已经在 kernel3.10 和 kernel 4.4 的代码上调试了 RTL8367RB 和 RTL8363,其他相关的 switch 可以参考着两份代码做相应的修改，也可以按照以下内容一步步排查。

1、swtich 相关口线配置

1、8367RB 千兆网口配置 switch 不提供 125M mac_clk, 所以需要 RK 主控提供输出 125M tx clock，另外还有两个时钟:RK 主控端的 MAC_TXCLK 和 MAC_RXCLK, 这两个时钟也是 125M, 其中 MAC_TXCLK 是 RK 主控端提供的时钟输出,MAC_RXCLK 是 switch 提供的时钟输出。而对于 8363 百兆配置的话，只需要一个 gmac_clkin ， 需要主控提供 50m 的时钟输出即可。
以 RTL8367RB 为例：

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3399-box.dtsi b/arch/arm64/boot/dts/rockchip/rk3399-box.dtsi
old mode 100644
new mode 100755
index 9c06c46..9034326
--- a/arch/arm64/boot/dts/rockchip/rk3399-box.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3399-box.dtsi
@@ -738,17 +738,19 @@
 &gmac {
        phy-supply = <&vcc_phy>;
        phy-mode = "rgmii";
-       clock_in_out = "input";
-       snps,reset-gpio = <&gpio3 15 GPIO_ACTIVE_LOW>;
+       clock_in_out = "output";
+       snps,reset-gpio = <&gpio3 10 GPIO_ACTIVE_LOW>;
        snps,reset-active-low;
-       snps,reset-delays-us = <0 10000 50000>;
-       assigned-clocks = <&cru SCLK_RMII_SRC>;
-       assigned-clock-parents = <&clkin_gmac>;
+       snps,reset-delays-us = <0 1000000 2000000>;
+       //assigned-clocks = <&cru SCLK_RMII_SRC>;
+       //assigned-clock-parents = <&clkin_gmac>;
+       assigned-clocks = <&cru SCLK_MAC>;
+       assigned-clock-rates = <125000000>;
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&rgmii_pins>;
        pinctrl-1 = <&rgmii_sleep_pins>;
-       tx_delay = <0x28>;
-       rx_delay = <0x11>;
+       tx_delay = <0x1c>;
+       rx_delay = <0>;
        status = "okay";
 };
```

另外需要将<&cru PLL_NPLL> CLK 由 600m 调整到 250M, 保障输出的 NPLL GMII_RXCLK ,TXCLK 能正常的输出 250M 的时钟。

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi b/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi
index f13985c..ee343d5 100644
--- a/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi
@@ -148,7 +148,7 @@
                        <50000000>, <100000000>,
                        <75000000>, <75000000>,
                        <816000000>, <816000000>,
-                       <600000000>, <200000000>,
+                       <250000000>, <200000000>,
                        <800000000>, <150000000>,
                        <75000000>, <37500000>,
                        <300000000>, <100000000>,
```

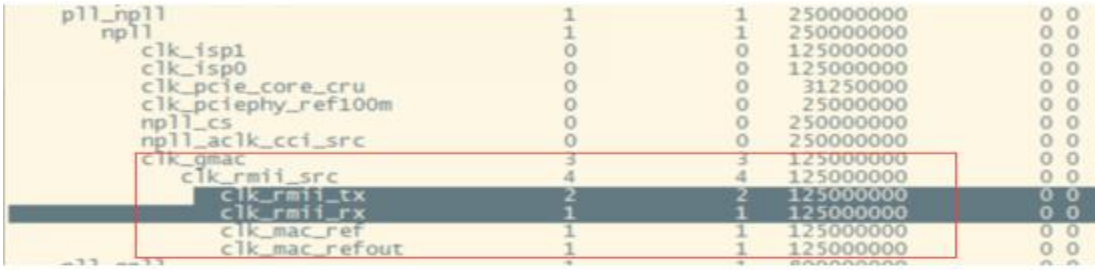
RTL8363 可以参考《针对 RK 平台 KERNEL3.10 RTK8363NB 代码修改》自行查看。

2、需要确定 RTL8367RB 的复位管脚，以及复位延迟。按照 RTL 要求复位管脚低电平有效，然后拉高 延迟 1 到 2s 才会开始通过 MDIO/MDC 去读写寄存器。

3.4. Boot up time

Switch need 1 second to finish booting (after power up or reset). After that, switch will be ready to be initialized and configured. If users try to configure switch before it finishes booting, the configuration may loss.

做完以上 1 和 2 之后，可以通过串口命令 cat /d/clk/clk_summary 查看



其次用示波器量一下主控端的 TXCLK,RXCLK 是否输出 150M 正确的波形出来，确认复位信号正确的复位。

这里需要注意的是：通常情况下是先提供时钟 clk，然后再对 switch 进行复位。在 RK kernel4.4 的代码上默认设置是对的，但是 RK KERNEL3.10 的代码上需要修改下图，先给时钟再复位。

D:\...\针对RK平台KERNEL3.10 RTK8363NB 代码修改\代码比对\kernel_new\drivers\net\ethernet\rockchip\gmac\stmmac_main.c

2019/2/25 17:41:12 93,231 字节 C,C++,C#,ObjC 源代码 ANSI UNIX

```
 *  Description:
 *  This function is the open entry point of the driver.
 *  Return value:
 *  0 on success and an appropriate (-)ve integer as defined in errno.h
 *  file on failure.
 */
static int stmmac_open(struct net_device *dev)
{
    struct stmmac_priv *priv = netdev_priv(dev);
    int ret;

    if ((priv->plat) && (priv->plat->bsp_priv)) {
        struct bsp_priv * bsp_priv = priv->plat->bsp_priv;
        if (bsp_priv) {
            if (bsp_priv->gmac_clk_enable) {
                bsp_priv->gmac_clk_enable(true);
            }
            if (bsp_priv->phy_power_on) {
                bsp_priv->phy_power_on(true);
            }


        }
    }
```

3、需要确认我们主控端网口 pin 脚的复用关系是否配置正确。

这里不做过多描述，具体可以查看《相关资料文档》里面的《如何确认 IO 复用问题.pdf》以及《Rockchip 平台以太网配置指南 v2.6.pdf》里面均有介绍。

2、确认 switch 的读写

1、确认完前面的第 1,2，3 之后，就可以开始调试 rtl8367rb 的 driver 了。由于 RTL switch 厂商提供的 driver 本身都是一些.c 和.h 文件，并没有 makefile 和 kconfig 所以这部分需要自己手动完成。目前我自己提供的针对 RTL8367RB 的 driver 见压缩包 rtl8367c.tgz，将其解压到 drivers/net/ethernet/stmicro/stmmac/目录下。然后需要配置以下截图，使其可以正常编译。

```
diff --git a/arch/arm64/configs/rockchip_defconfig b/arch/arm64/configs/rockchip_defconfig
index 876b26f..880ab5f 100644
--- a/arch/arm64/configs/rockchip_defconfig
+++ b/arch/arm64/configs/rockchip_defconfig
@@ -303,6 +303,7 @@ CONFIG_TUN=y
 # CONFIG_NET_VENDOR_SMSC is not set
 CONFIG_STMMAC_ETH=y
 # CONFIG_DWMAC_GENERIC is not set
+CONFIG_ETHNET_RTL8367C=y
 # CONFIG_NET_VENDOR_SUN is not set
 # CONFIG_NET_VENDOR_SYNOPSYS is not set
 # CONFIG_NET_VENDOR_TEHUTI is not set
```

```
diff --git a/drivers/net/ethernet/stmicro/stmmac/Kconfig b/drivers/net/ethernet/stmicro/stmmac/Kconfig
old mode 100644
new mode 100755
index cec147d..855beef
--- a/drivers/net/ethernet/stmicro/stmmac/Kconfig
+++ b/drivers/net/ethernet/stmicro/stmmac/Kconfig
@@ -128,3 +128,6 @@ config STMMAC_PCI

          If unsure, say N.
 endif
+
+source "drivers/net/ethernet/stmicro/stmmac/rtl8367c/Kconfig"
+
diff --git a/drivers/net/ethernet/stmicro/stmmac/Makefile b/drivers/net/ethernet/stmicro/stmmac/Makefile
old mode 100644
new mode 100755
index b390161..fbaeddc
--- a/drivers/net/ethernet/stmicro/stmmac/Makefile
+++ b/drivers/net/ethernet/stmicro/stmmac/Makefile
@@ -18,3 +18,5 @@ stmmac-platform-objs:= stmmac_platform.o

 obj-$(CONFIG_STMMAC_PCI) += stmmac-pci.o
 stmmac-pci-objs:= stmmac_pci.o
+
+obj-$(CONFIG_ETHNET_RTL8367C) += rtl8367c/
```

2、确认前面谈到的硬件接法 PIN82 是 PHY_ADDR 选择，目前 switch 代码这块位于 drivers/net/ethernet/stmicro/stmmac/rtl8367c/smi.c 中的 MDC_MDIO_PHY_ID,另外 switch 自己封装了一层读写接口，见黄色框，后面使用的是 RK 封装的读写接口，确认可以正常的读写。

```
/*****************************************************************************/
/*  MDC/MDIO porting                                                         */
/*****************************************************************************/
/* define the PHY ID currently used */
#define MDC_MDIO_PHY_ID    29  /* PHY ID 0 or 29 */

extern int stmmac_mdio_read_switch(struct mii_bus *bus, int phyaddr, int phyreg);
extern int stmmac_mdio_write_switch(struct mii_bus *bus, int phyaddr, int phyreg,u16 phydata);

/* MDC/MDIO, redefine/implement the following Macro */
//#define MDC_MDIO_WRITE(preamableLength, phyID, regID, data) rtl8761_mdio_write(phyID,regID, data)
//#define MDC_MDIO_READ(preamableLength, phyID, regID, pData) rtl8761_mdio_read(phyID,regID,pData)

#define MDC_MDIO_WRITE(preamableLength, phyID, regID, data)  stmmac_mdio_write_switch(rtl8761_bus,phyID,regID,data);

#define MDC_MDIO_READ(preamableLength, phyID, regID, pData)   pData=stmmac_mdio_read_switch(rtl8761_bus,phyID,regID);
```

硬件口线决定

3、确认好这些读写接口之后，需要将我们 sdk 中默认的 phy 的 MDC/MDIO 的操作接口关闭，否则会有影响，导致无法正确的拿到数据，这个比较重要，后面也会谈到。确认好这些之后，我们可以尝试一些接口，去读一下 0x1b03 的寄存器看能否拿到正确的数据，default 默认值为 0x0432。

drivers/net/ethernet/stmicro/stmmac/stmmac_main.c 里面 stmmac_init_phy 函数中打开 rtk_switch_reg1b03()这个函数；./drivers/net/ethernet/stmicro/stmmac/rtl8367c/rtk_switch.c 中打开这个函数实体；在./drivers/net/ethernet/stmicro/stmmac/rtl8367c/rtk_switch.h 中打开这个声明。

```
root@lq-ThinkPad-S5:/media/lq/N1_2T/RKSDK_PROJECT/BOX_RK3399_8_1/kernel# grep -i rtk_switch_reg1b03 ./drivers/ -rn
./drivers/net/ethernet/stmicro/stmmac/stmmac_main.c:2974:               //rtk_switch_reg1b03();
./drivers/net/ethernet/stmicro/stmmac/rtl8367c/rtk_switch.c:729://rtk_api_ret_t rtk_switch_reg1b03(void)
./drivers/net/ethernet/stmicro/stmmac/rtl8367c/rtk_switch.h:574://extern rtk_api_ret_t rtk_switch_reg1b03(void);
```

打开这些函数，通过串口打印看能否读取到 0x0432，如果能正确的读取，则 MDC/MDIO 通讯没有问题。

目前 RTL8367RB 和 RTL8363 读取 1b03 的寄存器都是 0x0432，其他 RTL switch 芯片请先和原厂确认，给一个默认寄存器和值，确认软件读写正确。另外，由于 mdc clk 在 phy/switch 这边有特定的时钟要求，一般要求 mdc clk 在 2.5m 以下，所以确认软件有打上《相关补丁资料验证》里面的 fix_mdc_clock_2.5m.txt,不然无法读到正确的值。

3、确认 SWITCH ID.

完成前面 MDC/MDIO 的读写之后，按照 KERNEL 代码本身的框架结构，会去读取 PHY/SWITCH ID ,具体可以参看 driver/net/phy/phy_device.c 里面 get_phy_device 的函数， 以 rtl 8367rb 为例，获取 SWITCH ID 的接口函数

ret = rtl8367c_getAsicPHYReg(addr, MII_PHYSID1, &phy_reg); 获取 switch ID 的高 16 位

ret = rtl8367c_getAsicPHYReg(addr, MII_PHYSID2, &phy_reg); 获取 switch ID 的低 16 位

通过串口可以打印 switch 的值。

pr_err("lq@@@@@@@@@@@@@@@@PHY addr[%d] get phy_id[0x%x]\n", addr, *phy_id);

这里需要和 RTL SWITCH 厂商确认 switch 读取 ID 的接口，确认正确。

正确的情况 RTL8367RB 读到的是 0x001cc942，RTL8363 读到的是 0x001cc943。然后需要将这个值写入到 static struct phy_driver genphy_driver[] 这个结构体里面。

正常情况下，读取 PHY/SWITCH ID 都需要写入到 phy_driver genphy_driver[]这个结构体里面.phy_id和.phy_id_mask，因为代码中拿到 id 之后会去和这个结构体里面进行比对，比对成功就可以拿到 eth0， 可以通过 busybox ifconfig 查看到 eth0 的相关信息。

4、确认 RGMII/RMII 网口配置

1、确认完前面的内容之后，就配置千兆或者百兆网口正常的初始化动作。

```
开启RGMII 的流程如下，其中黄色部分可以根据需求改成enable。
1.  rtk_switch_init ;
2. 调整RGMII DELAY ：rtk_api_ret_t rtk_port_rgmiiDelayExt_set(rtk_port_t port, rtk_data_t txDelay, rtk_data_t rxDelay) ;
3. RGMII 设置 :
rtk_port_mac_ability_t mac_cfg ;
rtk_mode_ext_t mode ;
mode = MODE_EXT_RGMII;
mac_cfg.forcemode = MAC_FORCE;
mac_cfg.speed = PORT_SPEED_1000M;
 mac_cfg.duplex = FULL_DUPLEX;
mac_cfg.link = PORT_LINKUP;
mac_cfg.nway = DISABLED;
 mac_cfg.txpause = DISABLED;
 mac_cfg.rxpause = DISABLED;
rtk_port_macForceLinkExt_set(EXT_PORT0, mode,&mac_cfg);
4. rtk_port_phyEnableAll_set(ENABLE);
关于RGMII delay 要注意，最好的方式是switch 和cpu 都打开tx detay，如果cpu 没打开tx delay ，那么在switch 测要调试RX delay。
调整RGMII delay的方法如下：
CALL 如下的API，（txDelay（范围0～1），rxDelay（范围0～7））
txDelay = 1， 调试。如果固定了，再调RX。
如果EXT port 没有RX到packet， rxDelay这边也就是从0～7一个一个的试。如果能收到packet就OK了。
那么初始化的时候就把这个API call一下。txDelay和rxDelay就是以上调出来的值。
API： rtk_api_ret_t rtk_port_rgmiiDelayExt_set(rtk_port_t port, rtk_data_t txDelay, rtk_data_t rxDelay)
```

这部分代码在 drivers/net/phy/phy_device.c 里面拿到 switch ID 之后，就会去调用 rtl8367rb_init 这个函数，这个函数里面需要怎么写，需要和 SWITCH 原厂确认接口正确。

确认完这些接口之后，可以通过 busybox ifconfig eth0 down 或者 busybox ifconfig eth0 up 查看 link 状态，确认是否有报告出错。Switch 端 也可以通过寄存器值的状态变化，确认前面的初始化函数是否正确（可以咨询 switch 原厂）。以 rtl8367rb 为例，在 link 的时候会打印如下错误信息，或者 ioctrl 0x8914 failed 等错误信息。前者错误信息一般情况 clk tree 以及 switch 初始化相关，检查那几个 clk 以及 switch 初始化函数。后者多半和复位以及 clk 相关，多半是先复位再送时钟导致。

```
17:28:15  [   13.421406] rk_gmac-dwmac fe300000.ethernet: rk_get_eth_addr: mac address: 46:29:f4:fc:57:e1
17:28:15  [   13.421427] eth0: device MAC address 46:29:f4:fc:57:e1
17:28:15  [   13.421437] libphy: lq ~~~~~~~~~~~~~~~~phy_connect_direct begin interface[7]
17:28:15  [   13.501377] liqing debug rtl8367rb_config_init
17:28:15  [   13.501438] libphy: lq phy_init_hw [0]
17:28:15  [   13.501459] libphy: lq ~~~~~~~~~~~~~~~~phy_connect_direct rc[0]
17:28:15  [   13.604366] stmmac_hw_setup: DMA engine initialization failed
17:28:15  [   13.604387] stmmac_open: Hw setup failed
```

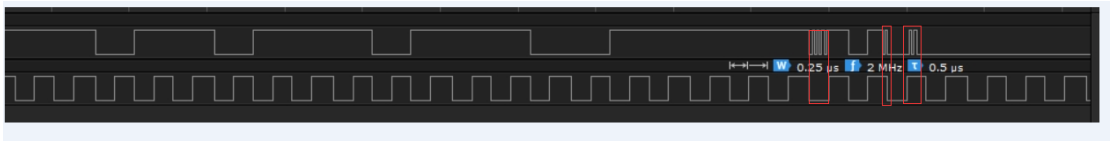
```c
struct phy_device *get_phy_device(struct mii_bus *bus, int addr, bool is_c45)
{
        struct phy_c45_device_ids c45_ids = {0};
        u32 phy_id = 0;

        //added by liqing
        int r;

#if 0
        r = get_phy_id(bus, addr, &phy_id, is_c45, &c45_ids);
        if (r)
                return ERR_PTR(r);

        /* If the phy_id is mostly Fs, there is no device there */
        if ((phy_id & 0x1fffffff) == 0x1fffffff)
                return NULL;
#endif


        rtl8761_bus = bus;

        if(addr != 0)           ///only read 0
                return NULL;

        r = rtk8367rb_getphy_id(bus, addr, &phy_id, is_c45, &c45_ids);

        if (r)
                return ERR_PTR(r);

        /* If the phy_id is mostly Fs, there is no device there */
        if ((phy_id & 0x1fffffff) == 0x1fffffff)
        {
                return NULL;
        }

        if ((phy_id & 0xffff) == 0xffff)
        {
                return NULL;
        }
    if (phy_id == 0x0)
        {
                return NULL;
        }

  rtl8367rb_init();
        return phy_device_create(bus, addr, phy_id, is_c45, &c45_ids);
}
EXPORT_SYMBOL(get_phy_device);
```

```c
int rtl8367rb_init(void)
{
        int ret = -1;
        rtk_port_mac_ability_t abi;
        memset(&abi, 0, sizeof(rtk_port_mac_ability_t));
        abi.forcemode = MAC_FORCE;
        abi.speed =  PORT_SPEED_1000M;                  ///1000M
        abi.duplex = FULL_DUPLEX;
        abi.link = PORT_LINKUP;
        abi.nway = DISABLED;
        abi.txpause = 1;
        abi.rxpause = 1;
        //add by liqing 2019-01-08
        rtk_switch_init();

        ret = rtk_port_macForceLinkExt_set(EXT_PORT0, MODE_EXT_RGMII, &abi);
        if (ret < 0)
        {
                pr_err("lq rtk_port_macForceLinkExt_set failed\n");
                return ret;
        }


        ret=rtk_port_phyEnableAll_set(1);
        if (ret < 0)
        {
                pr_err("lq rtk_port_phyEnableAll_set\n");
                return ret;
        }


        ret = rtk_port_rgmiiDelayExt_set(EXT_PORT0, 1, 1);
        if (ret < 0)
        {
                pr_err("lq rtk_port_rgmiiDelayExt_set failed\n");
                return ret;
        }

        //msleep(2000);
        printk("liqing debug %s\n", __func__);
        //phy_create_sysfs(phydev);
        //ret = genphy_config_init(phydev);
        //if (ret < 0)
        //      return ret;
        return 0;
}
```

三、软件调试问题排查

问题点 1 MDIO/MDC 出现问题

用逻辑分析仪录的 mdc/mdio 数据异常，一个 clock 里有好几笔数据



正常 mdc/mdio 波形应该如下：

RK3399 TRM

0000	60-100 MHz	pclk_gmac/42
0001	100-150 MHz	pclk_gmac/62
0010	20-35 MHz	pclk_gmac/16
0011	35-60 MHz	pclk_gmac/26
0100	150-250 MHz	pclk_gmac/102
0101	250-300 MHz	pclk_gmac/124
0110, 0111	Reserved	

The MDC is the derivative of the application clock pclk_gmac. The management operation is performed through the gmii_mdi_i, gmii_mdo_o and gmii_mdo_o_e signals. A three-state buffer is implemented in the PAD.

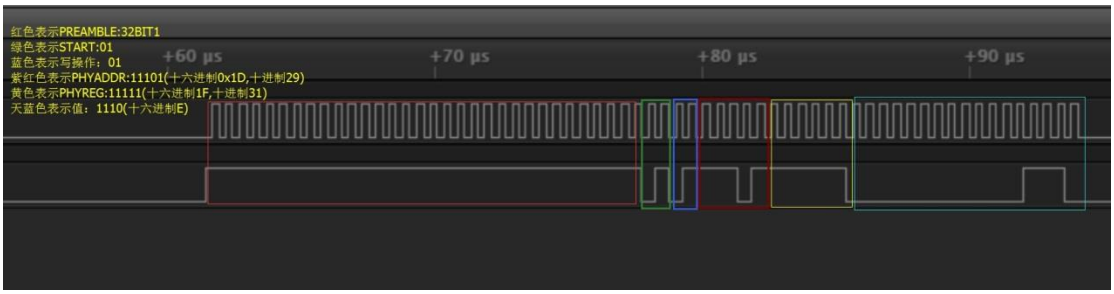
The frame structure on the MDIO line is shown below.

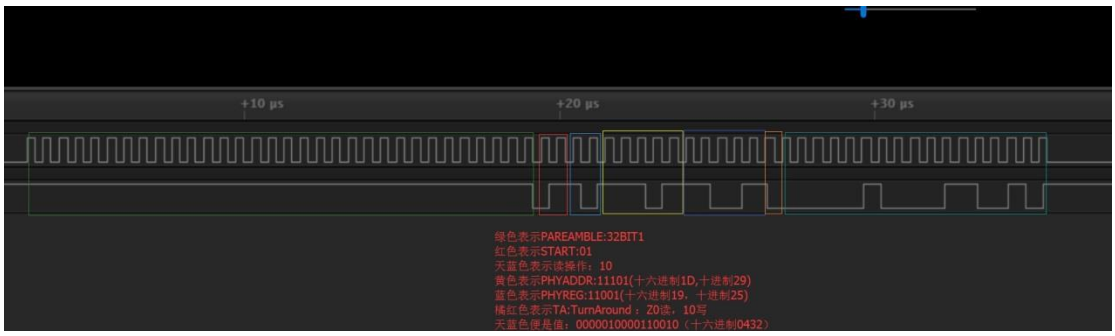
IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE

Fig. 15-9 MDIO frame structure

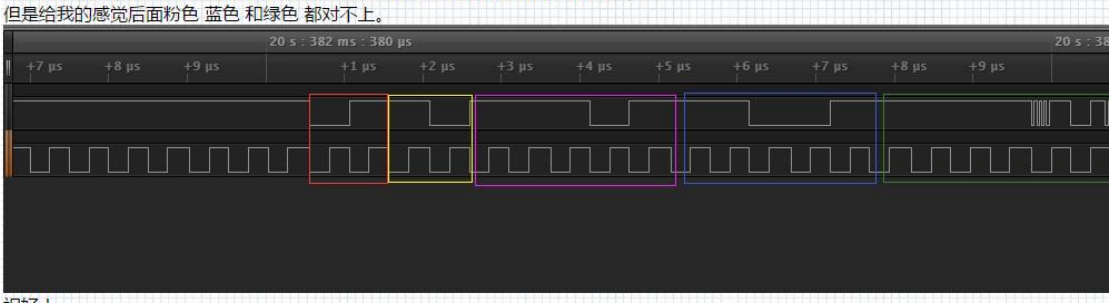
IDLE: The mdio line is three-state; there is no clock on gmii_mdc_o
PREAMBLE: 32 continuous bits of value 1
START: Start-of-frame is 2'b01
OPCODE: 2'b10 for read and 2'b01 for write
PHY ADDR: 5-bit address select for one of 32 PHYs
REG ADDR: Register address in the selected PHY
TA: Turnaround is 2'bZ0 for read and 2'b10 for Write
DATA: Any 16-bit value. In a write operation, the GMAC drives mdio; in a read operation, PHY drives it.

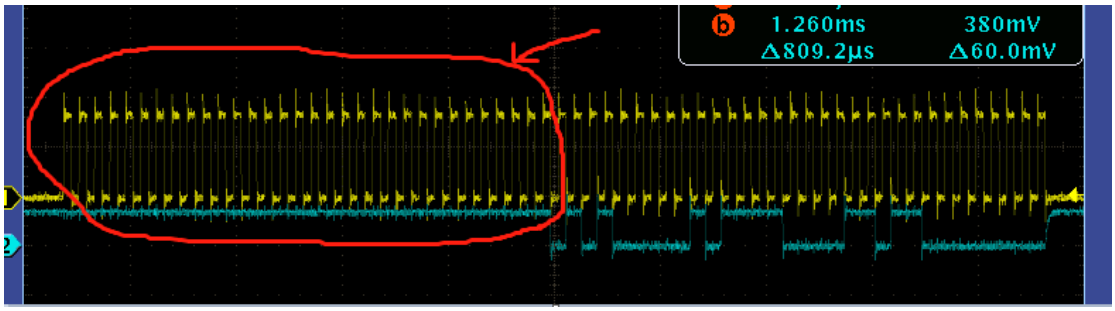
15.3.5 Power Management Block





这个应该就是读的波形了，
红色：01 start
黄色：10 读
粉色： phy地址（十进制29，十六进制1D）
蓝色：phy reg（十六进制0x25)
绿色： 值（怎么感觉一个时钟里面还有几个数据的，不是很明白什么情况？）
但是给我的感觉后面粉色 蓝色 和绿色 都对不上。





后经过查明是因为初始化时有 phy 寄存操作导致，对于 switch 而言需要干净寄存器操作，我们做如下修改，把原本寄存操作直接返回，会对 switch 专门提供寄存器操作函数:

```
int mdiobus_read(struct mii_bus *bus, int addr, u32 regnum)
{
        int retval;

        return 0;

        BUG_ON(in_interrupt());

        mutex_lock(&bus->mdio_lock);
        retval = bus->read(bus, addr, regnum);
        mutex_unlock(&bus->mdio_lock);

        return retval;
}
EXPORT_SYMBOL(mdiobus_read);

/**
/drivers/net/phy/mdio_bus.c
```

```
int mdiobus_write(struct mii_bus *bus, int addr, u32 regnum, u16 val)
{
        int err;

        return 0;

        BUG_ON(in_interrupt());

        mutex_lock(&bus->mdio_lock);
        err = bus->write(bus, addr, regnum, val);
        mutex_unlock(&bus->mdio_lock);

        return err;
}
EXPORT_SYMBOL(mdiobus_write);
```

增加单独接口，其实这个呼叫以前接口就可以了，只是调试时候改成这样了。

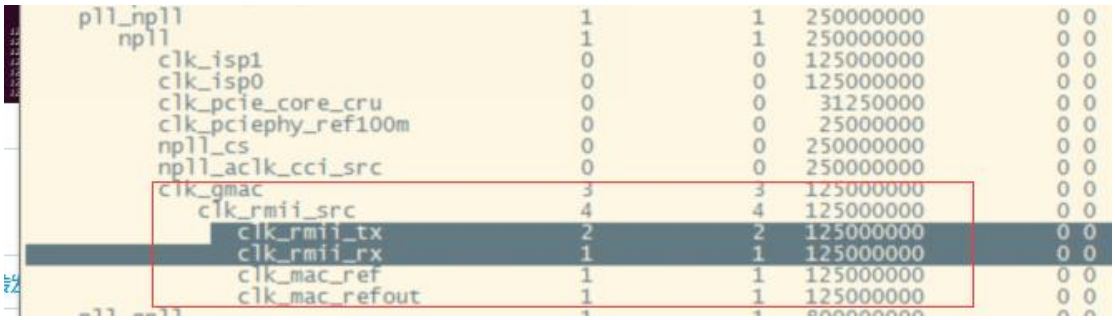
```
int stmmac_mdio_read_switch(struct mii_bus *bus, int phyaddr, int phyreg)
{
        struct net_device *ndev = rtl8761_bus->priv;
        struct stmmac_priv *priv = netdev_priv(ndev);
        unsigned int mii_address = priv->hw->mii.addr;
        unsigned int mii_data = priv->hw->mii.data;

        int data;
        u16 regValue = (((phyaddr << 11) & (0x0000F800)) |
                        ((phyreg << 6) & (0x000007C0)));
        //printk("-------------mdio_read : phyaddr=%d,phyreg=%d\n",phyaddr,phyreg);
        regValue |= MII_BUSY | ((priv->clk_csr & 0xF) << 2);

        if (stmmac_mdio_busy_wait(priv->ioaddr, mii_address))
                return -EBUSY;
        //dump_stack();
        writel(regValue, priv->ioaddr + mii_address);

        if (stmmac_mdio_busy_wait(priv->ioaddr, mii_address))
                return -EBUSY;

        /* Read the data from the MII data register */
        data = (int)readl(priv->ioaddr + mii_data);
        //printk("=============mdio_read: data=%d\n",data);
        return data;
}
```

```
int stmmac_mdio_write_switch(struct mii_bus *bus, int phyaddr, int phyreg,
                             u16 phydata)
{
        struct net_device *ndev = bus->priv;
        struct stmmac_priv *priv = netdev_priv(ndev);
        unsigned int mii_address = priv->hw->mii.addr;
        unsigned int mii_data = priv->hw->mii.data;
       u16 value =
            (((phyaddr << 11) & (0x0000F800)) | ((phyreg << 6) & (0x000007C0)))
            | MII_WRITE;
        //printk("++++++++++++++++++++++phyaddr=%d,phy_reg=%d,phydata=0x%x\n",phyaddr,phyreg,phydata);
        value |= MII_BUSY | ((priv->clk_csr & 0xF) << 2);

        /* Wait until any existing MII operation is complete */
        if (stmmac_mdio_busy_wait(priv->ioaddr, mii_address))
                return -EBUSY;
        //dump_stack();
        /* Set the MII address register to write */
        writel(phydata, priv->ioaddr + mii_data);
        writel(value, priv->ioaddr + mii_address);

        /* Wait until any existing MII operation is complete */
        return stmmac_mdio_busy_wait(priv->ioaddr, mii_address);
}
```

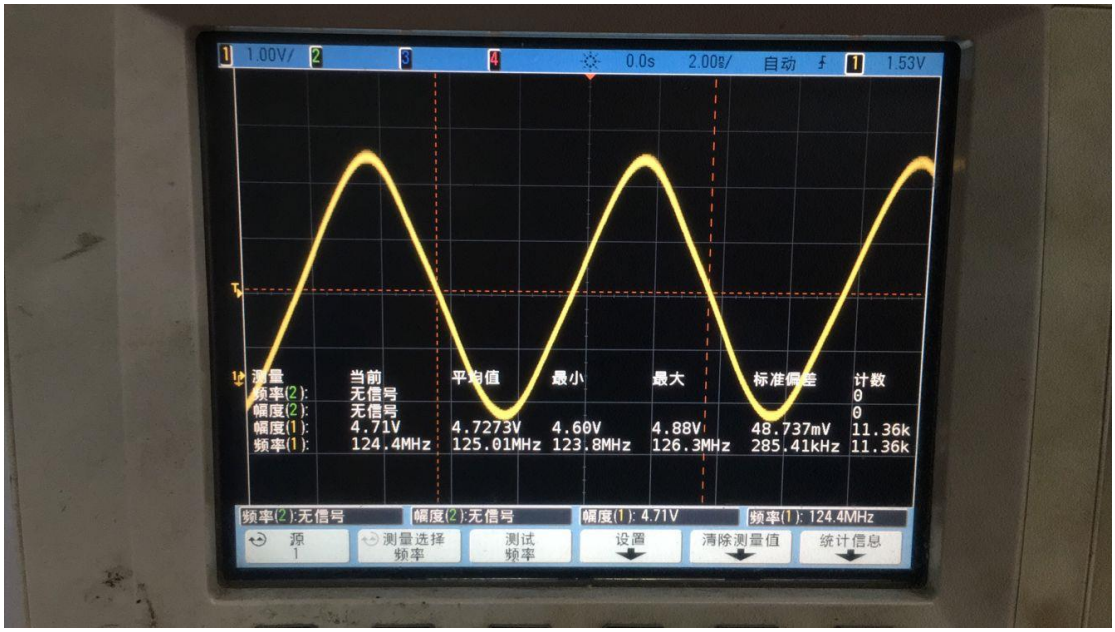
问题点 2

量不到 125M clock, 请确认主控 clock tree 有设置起来,正常如下

Cat /d/clk/clk_summary



正常的话会出来这样的波形：



问题点 3

千兆或者百兆 switch 经常会出现 busybox ifconfig eth0 RX 无包，TX 有数据包的情况。通常的排除方法有以下几种：

1、 针对千兆网口的排查：

A. 由于千兆网口对 RX delay 和 TX delay 有特定的要求，我们 RK 也会去设置 RX delay 和 TX delay，通常情况下，现将我们 RX delay 和 TX delay 都设置成 0，由 switch 端的 RX/TX delay 去控制，switch 一般时序在 2ns 以内可正常锁存数据。调整 RGMII delay 的方法如下：

rtk_api_ret_t rtk_port_rgmiiDelayExt_set(rtk_port_t port, rtk_data_t txDelay, rtk_data_t rxDelay) （txDelay（范围 0～1）， rxDelay（范围 0～7））

txDelay = 1， EXT port 没有 RX 到 packet， rxDelay 这边也就是从 0～7 一个一个的试。如果能收到 packet 就 OK 了。

B.如果调整 switch 的 RX delay 和 Tx delay 仍然没有效果，检查一下 RK 端的配置，通常千兆网口配置，RK 和 switch 都要设置固定成千兆;如果是百兆网口配置，RK 和 swich 都要设置固定成百兆，不可协商，将 RK 端这边后面读取 phy 的寄存器拿取状态都强行固定。例如：

```
phydev->duplex = DUPLEX_FULL;
phydev->speed = SPEED_1000;
```

C.如果前面两者都做了仍然那不到 RX 的数据的话，只能检查一下 PC 和 盒子两边双方向 ping 包，用示波器去检查一下我们主控端的(RX/TX)发和收，以及 switch 端的(RX/TX)发和收.以及对应的 RXCLK 和 TXCLK 等。我们 RK 这边通常可以通过 busybox ifconfig 里面 eth0 里面的 RX package 就是我们拿到的包，TX package 就是我们发出的包， 同样 switch 也有一

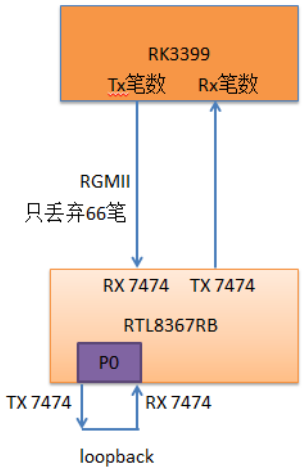
些调试方法可以确认他们拿到的包和发出的包，后面会有介绍。

D. 以上方式如果仍然不行的话，需要打上 GMAC tx rx delay 动态调整补丁 V2.0 回环测试补丁 ，设置 switch 为回环模式(以 rtl8367rb 为例需要设置回环的寄存器：register0 bit[14]置 1，其他的需要咨询 rtl 原厂获得)

此补丁只做测试验证主控和 switch 之间的通讯状态，以及 switch 经过变压器到网口整个链路状态，此补丁只做测试验证问题，并不代表一定要打上做功能。补丁包里面也有说明。

回环测试是 RGMII，原理如下：

从mib来看，switch内部并没有丢包。
以第2次dump的信息来看：
67RB RGMII口收到7474笔封包，后转到UTP port0，通过loopback回到67RB RGMII TX还是7474笔封包。
需要确认一下RK端TX多少笔封包，且最终RX到多少笔封包。若RK端也收到7474笔封包的话，证明67RB→RK3399 RGMII方向是ok的。
（理论上，若loopback设置RK端能收到包的话，ping包同样也能抓到包）
从mib信息来看RK RGMII TX方向有少量丢包（66笔），这部分需从delay设置以及RGMII Timing量测看是否符合67RB RX的要求



建议优先调RK3399和67RB的TX Delay（67RB TX delay :0x1307 设置0x0008）。

2、 百兆网口的排查：

A 同样需要确认 RK 和 switch 都要设置固定成百兆不可协商，将 RK 端这边后面读取 phy 的寄存器拿取状态都强行固定。例如：

```
phydev->duplex = DUPLEX_FULL;
phydev->speed = SPEED_1000;
```

B 同样需要确认前面 千兆网口的排查第 C 步和第 D 步。

问题点 4

Switch 调试过程的一些命令方法：

1、 通过 mdc/mdio 读写 switch 相关寄存器方法

比方查看寄存器 0x1305 寄存器的值

echo 0x1305 > ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_reg

cat ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_regValue

会打印：cat phy reg 0x1305 = 0xc010

如果想设置寄存器 0x1305 寄存器的值为 0x0010

echo 0x1305 > ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_reg

echo 0x0010 > ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_regValue

2、 SDK 默认调试 eth0 命令的方法

设置盒子端 IP

busybox ifconfig eth0 192.168.1.3

eth 口抓包命令

tcpdump -i eth0 –s 0 –w /data/snf.pcap 抓取网络包，然后通过 wireshark 去打开查看

3、 Switch 通过抓取 MIB 信息的一些方法

清除 MIB 统计：

echo 1 > ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_Rst

查看 MIB 信息：

一般分为两块：一个是查看主控和 switch 端(RGMII)，另外一个就是 switch 和网口端(VLAN),从 switch 的规格书中可以确认，自己的硬件是接的那个 portX(1,2)，比如以 8367rb 为例：

The RTL8367RB supports two extension interfaces. The interface function mux is summarized in Table 18 and Table 19. The Extension GMAC1 and Extension GMAC2 of the RTL8367RB support RGMII, MII MAC mode, or MII PHY mode via register configuration.

Table 18. RTL8367RB Extension Port 1 Pin Definitions

Pin No.	Extension Interface	Type	RGMII	MII MAC Mode	MII PHY Mode
56	E1_CRS	I	-	M1M_CRS	-
57	E1_DO3	O	RG1_TXD3	M1M_TXD3	M1P_RXD3
58	E1_DO2	O	RG1_TXD2	M1M_TXD2	M1P_RXD2
59	E1_DO1	O	RG1_TXD1	M1M_TXD1	M1P_RXD1
60	E1_DO0	O	RG1_TXD0	M1M_TXD0	M1P_RXD0
61	E1_DOEN	O	RG1_TXCTL	M1M_TXEN	M1P_RXDV
62	E1_GCLK	O	RG1_TXCLK	M1M_TXCLK	M1P_RXCLK
63	E1_DICLK	I	RG1_RXCLK	M1M_RXCLK	M1P_TXCLK
64	E1_DIDV	I	RG1_RXCTL	M1M_RXDV	M1P_TXEN
65	E1_DI0	I	RG1_RXD0	M1M_RXD0	M1P_TXD0
66	E1_DI1	I	RG1_RXD1	M1M_RXD1	M1P_TXD1
67	E1_DI2	I	RG1_RXD2	M1M_RXD2	M1P_TXD2
68	E1_DI3	I	RG1_RXD3	M1M_RXD3	M1P_TXD3

有两个口分别对应的 EXT_PORT 和 UTP_PORT,分别打印出 mib 信息。代码在 phy_device.c 里面可以自行查看。对应的命令：

echo 1 > ./sys/devices/platform/fe300000.ethernet/mdio_bus/stmmac-0/stmmac-0:00/rtl_phy_Mib

realtek mib 这边主要看下面统计，如果看不懂，可以发给 rtl 原厂确认

EXT_PORT0 是 switch MAC 端统计， UTP_PORT0 代表 phy 物理接口统计：

[5976.088116] port 0: ifInBroadcastPkts [60]: 7906358

[5976.150769] port 16: ifOutBroadcastPkts [32]: 7906358

ifInBroadcastPkts 代表输入广播包多少笔，ifOutBroadcastPkts 代表输出广播包多少笔

另外 rtl 还有很多相关接口，具体使用方法还是需要咨询 rtl switch 厂商获得更多相关信息，RK 这边只负责配合跑通芯片，SWITCH API 由 rtl 提供技术支持。

附件中：

针对 RK 平台 KERNEL4.4 RTK8367RB 代码修改 ---针对 RK KERNEL4.4 的代码修改开发

针对 RK 平台 KERNEL3.10 RTK8363NB 代码修改---针对 RK KERNEL3.10 的代码修改开发

相关资料文档：

Realtek_Unmanaged_Switch_API_Document.pdf --- rtl8367rb switch 厂商提供的相关文档

Realtek_Unmanaged_Switch_ProgrammingGuide.pdf--- rtl8367rb switch 厂商提供的相关文档

Realtek_Unmanaged_Switch_ReleaseNote.pdf --- rtl8367rb switch 厂商提供的相关文档

RTL8367RB-CG_DataSheet_1.2.pdf --- rtl8367rb switch 厂商提供的 datasheet

Rockchip 平台以太网配置指南 v2.6.pdf --- RK 平台以太网开发文档

如何确认 IO 复用问题.pdf ---RK 平台确认 io 复用关系相关文档。

相关补丁资料验证：

fix_mdc_clock_2.5M.txt ---确认 PHY/SWITCH MDC CLK 输出 2.5M clk。（补丁包里面已经打上，其他 SWITCH 请自行确认是否有打）

GMAC tx rx delay 动态调整补丁 V2.0 ---验证 RK 回环测试补丁包（仅供验证问题排查需要，正式补丁不需要）