InnCable 调试记录文档

文件状态： [√] **草稿** [ ] **正在修改** [ ] **正式发布**	部门	系统软件组
	版本	0.3
	作者	朱坤华
	完成时间	
	审核	
	审核时间	
密级状态：绝密( ) 秘密( ) 内部资料(√) 公开( )		

修改记录:

版本	修订者	时间	说明
0.1	朱坤华	2018 / 3 / 8	InnCable 调试记录文档
0.2	朱坤华	2018 / 4 / 19	InnCable chromecast 和 usb 网卡接入后同时开机问题修改要点
0.3	朱坤华	2018 / 4 / 25	InnCable AM1805 RTC 外部晶振校准方法

一. 背景及问题:

Inncable 项目的调试记录，记录调试过程中遇到的要点，文档是为了方便以后追踪问题。

二. 调试记录:

目录

InnCalbe 简介

InnCable 的定位大概是广告机。比如公共场所的电梯里面的广告机，广告机的工作模式就是不停的自动播放广告，或根据需要在指定的时间段播放广告。

这个项目调试过的驱动：

1. AM1805 RTC 实时时钟支持
2. USB Vbus power 控制提供出接口
3. Uart0 ttlen pin 脚控制提供出接口
4. 客户 chromecast 导致 usb 以太网不能上网问题

S905X_B 项目相关调试资料

\\192.168.1.8\work\home\zkh\Amlogic\项目资料\S905X

版本信息：

\\192.168.1.5\firmware\m9_release\version\s905x_dongle\version.ver

dts：gxl_stvs9_dongle_2g.dts

烧录方法：

全烧：

烧录工具调试资料目录下：USB_Burning_Tool

单烧：

1. 开机串口敲进入 uboot 模式：

烧录版本信息：usb start;fatload usb 0 12000000 version.ver;hw_write 12000000

boot.mg 烧录：usb start;fatload usb 0 $loadaddr boot.img;store write boot $loadaddr 0 2000000

dtb.img：usb start;fatload usb 0 1080000 dtb.img;store dtb write 1080000;

AM1805 调试

AM1805 是集 RTC、Watchdog、Alarm、Countdown 四大功能于一身，功能齐全的实时时钟芯片，包含片上振荡器以提供最低的功耗，完整的 RTC 功能，包括电池备份，可编程计数器和定时器和看门狗功能的报警，以及用于与主机控制器通信的 I2C 或 SPI 串行接口。 集成电源开关和具有计数器，定时器，闹钟和中断功能的复杂系统睡眠管理器使 AM18X5 可用作基于主机微控制器系统的监控组件。

SDK 中默认没有该 IC 的驱动支持，所以里面的驱动是根据规格书编写的。

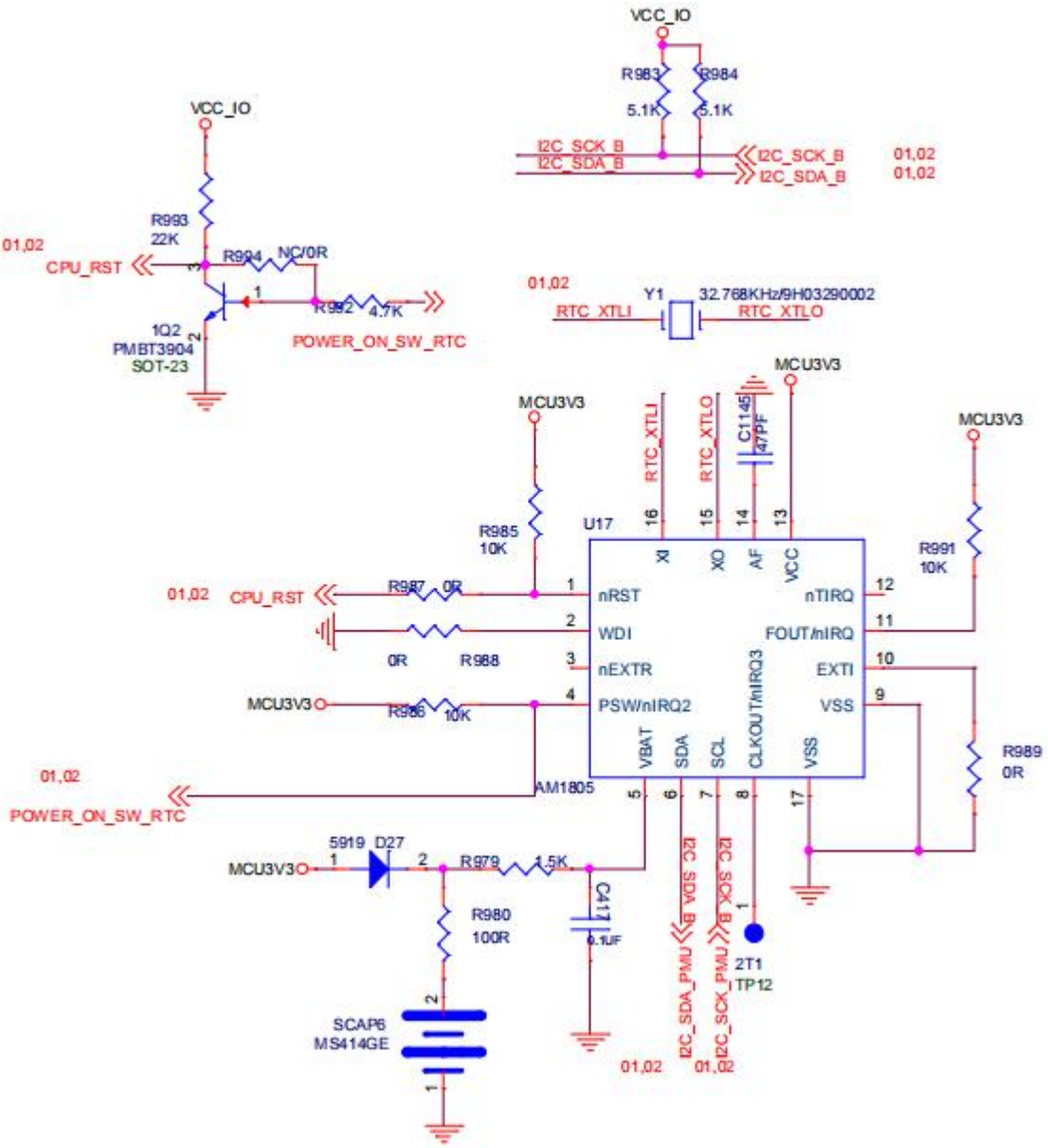
AM1805 相关资料：

\\192.168.1.8\work\home\zkh\Amlogic\项目资料\S905X\AM1805

驱动编写的调试步骤：

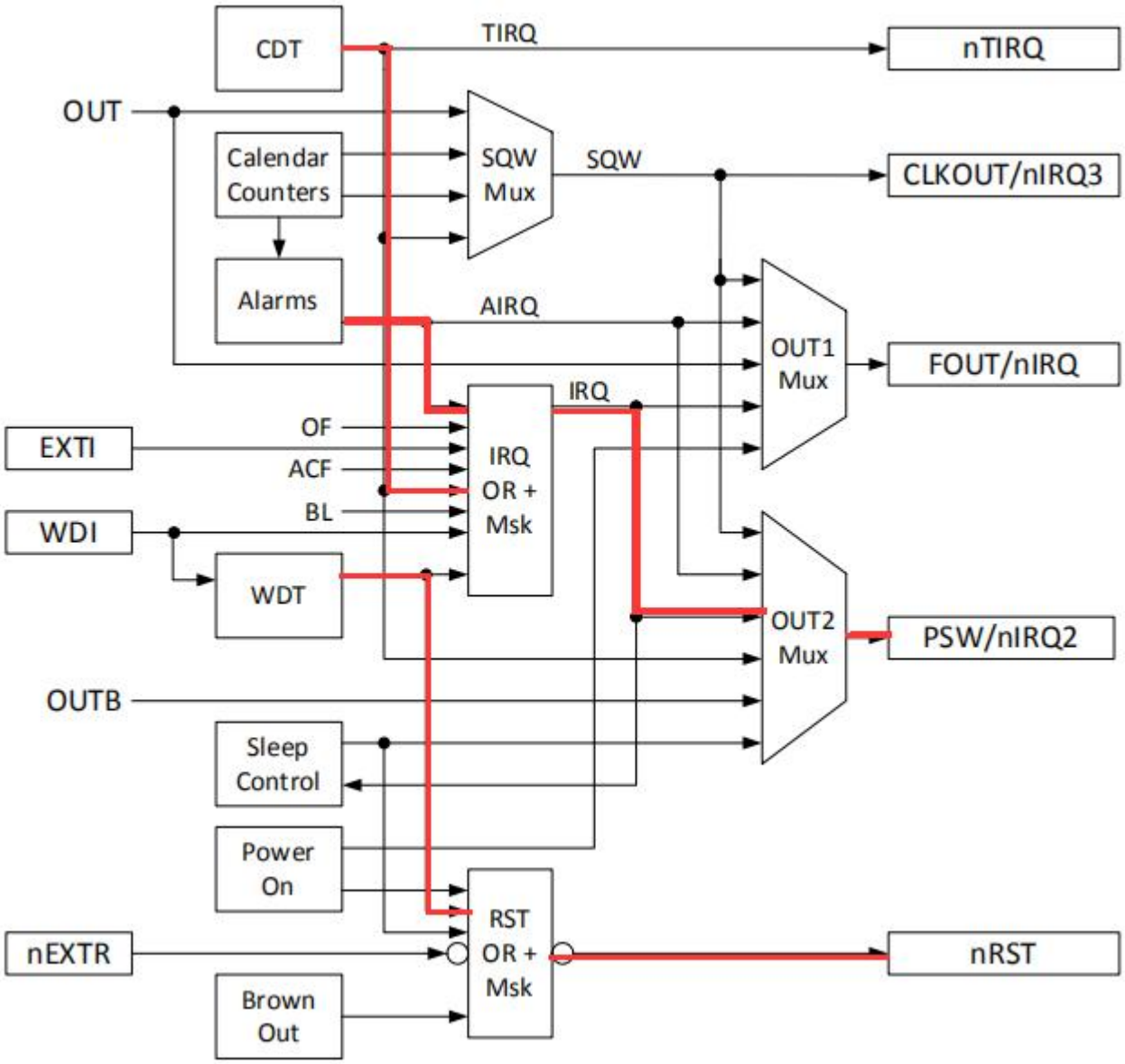
1. 调通 I2C，能通过 I2C 读写寄存器。
2. 做调试工具，驱动里面写一个 sys 控制寄存器，实现读写单个寄存器和打印全部寄存器。然后通过串口命令行用 cat 和 echo 直接去读写寄存器，并在调试 RTC/ALARM/COUNTDOWN/WATCHDOG 之前，先用读写寄存器命令验证功能。
3. 根据规格书提供的寄存器资料，开始通过直接读写寄存器，去先检验 RTC、Alarm、Watchdog、Countdown 功能，确认用寄存器能调试没问题后，在驱动里面把相关代码写好。
4. 调完后，写一个用户空间的程序，通过 IOCTL 来测试驱动。
5. 校正 RTC 外部晶振

硬件原理图接线方式：



硬件上的特性：

硬件原理图上的中断输入 WDI 没有接，中断输出接的有 nRST、PSW/nIRQ2，其他中断引脚 nTIRQ、CLKOUT/nIRQ3、FOUT/nIRQ。目前软件 Watchdog 中断输出走的是 nRST，其他 Alarm、Countdown 走的是 nIRQ2 中断脚。功能总述为下图中红色的走线，CDT 和 Alarm 到或门控制输出到 nIRQ2，WDT 到或门输出到 nRST：



前期调试：

由于硬件上 VCC 直接上电，所以不用去控制 IO 上电。

AM1805 规格书 I2C 地址是 0x8c/0x8d，这个地址包含读写位，bit0 是读写位，所以 bit1~7 是实际的 I2C 地址，0x8c/0x8d>>1 可以算出 I2C 地址是 0x69。I2C 实际挂载为 i2c ao，不是原理图的 i2c b。

在 dts 中加上 RTC 的配置编译烧录了 dtb 后，在驱动的 probe 函数中加读取 chip id 并打印，看到 I2C 通了。

```
+    rtc {
+        compatible = "amlogic, rtc_am1805";
+        dev_name = "rtc_am1805";
+        status = "okay";
+        reg = <0x69>;
+        i2c_bus = "i2c_bus_ao";
+        init_date = "2015/01/01";
+    };
```

由于验证功能需要频繁读写寄存器，所以注册了一个 class 直接操作寄存器。目前调试或者查问题都可以通过命令行直接操作 class 读写寄存器，方便快速定位问题。

驱动里面通过 class_create 来创建一个设备的逻辑类：

```
1108    ret = class_register(&rtc_am1805_class);
1109    if (ret){
1110        pr_info(" class register rtc_class fail!\n");
1111        return -1;
1112    }
```

然后创建三个功能的属性：读单个寄存器，写单个寄存器，读所有寄存器。这三个属性目前基本能满足这个芯片的调试。

```
static struct class_attribute rtc_class_attrs[] = {
    __ATTR(show_all_reg, S_IRUGO | S_IWUSR, show_all_reg, NULL),
    __ATTR(time, S_IRUGO | S_IWUSR, show_time, store_time),
    __ATTR(regwrite, S_IRUGO | S_IWUSR, NULL, store_reg),
    __ATTR(regread, S_IRUGO | S_IWUSR, NULL,show_reg),
    __ATTR(disable_feeddog, S_IRUGO | S_IWUSR, NULL,am1805_disable
    __ATTR(setalarm, S_IRUGO | S_IWUSR, am1805_set_alarm,NULL),
    __ATTR_NULL
};

static struct class rtc_am1805_class = {
    .name = "rtc_am1805",
    .class_attrs = rtc_class_attrs,
};
```

使用示例：

这里举例在命令行直接设置一个闹钟 Alarm

关闭 AIE 使能，设置 IM 为电平模式

echo "0x12 0xe0">/sys/class/rtc_am1805/write_reg

清除 ALM Status

echo "0x0f 0x00">/sys/class/rtc_am1805/write_reg

设置为 IRQ2 中断

echo "0x11 0x0c">/sys/class/rtc_am1805/write_reg

设置时间

echo "0x08 0x01">/sys/class/rtc_am1805/write_reg
echo "0x09 0x01">/sys/class/rtc_am1805/write_reg
echo "0x0a 0x50">/sys/class/rtc_am1805/write_reg
echo "0x0b 0x00">/sys/class/rtc_am1805/write_reg
echo "0x0c 0x02">/sys/class/rtc_am1805/write_reg
echo "0x0e 0x01">/sys/class/rtc_am1805/write_reg

一年一次

echo "0x18 0xc6">/sys/class/rtc_am1805/write_reg

启动

echo "0x12 0xe4">/sys/class/rtc_am1805/write_reg

1.RTC 时间调试

RTC 可给系统提供时间，当系统关机后，RTC 芯片由于有电池供电，时间就不会停止，这样下次设备开机后就知道时间。RTC 时间由于只需要读写时间相关的寄存器，直接在命令行用 echo 设置一下相关寄存器，可直接设置时间，然后在驱动里面实现 RTC 驱动提供的接口。寄存器：

6.1 Register Definitions and Memory Map

Table 18: Register Definitions (0x00 to 0x0F)

Offset	Register	7	6	5	4	3	2	1	0
0x00	Hundredths	Seconds - Tenths				Seconds - Hundredths			
0x01	Seconds	GP0	Seconds - Tens			Seconds - Ones			
0x02	Minutes	GP1	Minutes - Tens			Minutes - Ones			
0x03	Hours (24 hour)	GP3	GP2	Hours - Tens		Hours - Ones			
0x03	Hours (12 hour)	GP3	GP2	AM/PM	Hours - Tens	Hours - Ones			
0x04	Date	GP5	GP4	Date - Tens		Date - Ones			
0x05	Months	GP8	GP7	GP6	Months- Tens	Months - Ones			
0x06	Years	Years - Tens				Years - Ones			
0x07	Weekdays	GP13	GP12	GP11	GP10	GP9	Weekdays		
0x08	Hundredths Alarm	Hundredths Alarm - Tenths				Hundredths Alarm - Hundredths			
0x09	Seconds Alarm	GP14	Seconds Alarm - Tens			Seconds Alarm - Ones			

实现的接口：

```
+
+static const struct rtc_class_ops rtc_am1805_ops = {
+       .open       = rtc_am1805_open,
+       .read_time = rtc_am1805_read_time,
+       .set_time = rtc_am1805_write_time,
+       .read_alarm = rtc_am1805_read_alarm,
+       .set_alarm = rtc_am1805_set_alarm,
+       .set_cdt_time = rtc_am1805_set_timer,
+       .read_cdt_time = rtc_am1805_read_timer,
+       .ioctl      = rtc_am1805_ioctl,
+};
+
```

2.watchdog 调试

看门狗是用来定期监控系统的运行情况，一旦系统死机，看门狗就发出重启电路的信号。用途上比如广告机，我们需要它一直播放广告，如果突然死机了怎么办，所以我们就通过看门狗来解决，死机后看门狗自动重启设备，然后继续播放广告。

由于原理图上没有 WDI 中断输入，所以需要驱动模拟实现。模拟喂狗驱动上用一个计时器和一个工作队列来实现。计时器每隔一定时间发出一个需要喂狗的工作，工作队列对看门狗倒计时的寄存器重新填值。

目前我在驱动上设计喂狗的时间为 2 秒，超时的时间 4 秒。也就是 4 秒内没有喂狗，看门狗就产生一个中断到 nRST 引脚复位设备。

目前驱动里面计时器所做的内容就是添加一个工作到队列里：

```
//Timer for feed dog
setup_timer(&rtc_info->timer, am1805_watchdog_timer, 0);
rtc_info->timer.data = (unsigned long)rtc_info;
add_timer(&rtc_info->timer);
```

```
static void am1805_watchdog_timer(unsigned long data)
{
    struct rtc_am1805_priv *plat = (struct rtc_am1805_priv *)data;
    unsigned long flags = 0;

    //RTC_DBG(RTC_DBG_VAL, "%s ,run \n",__func__);
    spin_lock_irqsave(&plat->watchdog_lock, flags);
    queue_work(plat->watchdog_workqueue, &plat->work);
    spin_unlock_irqrestore(&plat->watchdog_lock, flags);
}
```

工作的内容就是喂狗：

```
//Workqueue for set watchdog timer
INIT_WORK(&rtc_info->work, am1805_watchdog_work);
rtc_info->watchdog_workqueue = create_singlethread_workqueue("am1805wd");
if (rtc_info->watchdog_workqueue == NULL) {
    pr_err("%s :Create workqueue failed. \n",__func__);
    goto out;
}
```

```
static void am1805_watchdog_work(struct work_struct *work)
{
    struct rtc_am1805_priv *plat = (struct rtc_am1805_priv *)container_of(work,struct rtc_am1805_priv , work);

    //RTC_DBG(RTC_DBG_VAL, "%s ,run \n",__func__);
    if(plat->watchdog_enable == TYPE_ENABLE) {
        am1805_watchdog_feeddog(WATCHDOG_TIMER_TIMER,WATCHDOG_USR_PIN);
        mod_timer(&rtc_info->timer,  jiffies +  msecs_to_jiffies(WATCHDOG_TIMER_FOR_FEED_DOG));
    }
}
```

工作流程：第一次 add_timer 的时候会启动一次计时器，计时器会添加一个工作到工作队列，工作队列填完值后，mod_timer 会改动计时器的启动时间，也就是 2 秒，2 秒后会再次进入计时器函数，然后计时器又添加一个工作到工作队列，如此循环。也就是不停的喂狗。

假如 kernel 跑飞了，这时驱动不会去喂狗了，这时看门狗的计时器会自动减到 0，进而出发电平中断到 nRST 引脚，设备就会复位。

寄存器：

6.7.4 0x1B - Watchdog Timer

This register controls the Watchdog Timer function.

Table 84: Watchdog Timer Register

Bit	7	6	5	4	3	2	1	0
Name	WDS	BMB					WRB	
Reset	0	0	0	0	0	0	0	0

Table 85: Watchdog Timer Register Bits

Bit	Name	Function
7	WDS	Watchdog Steering. When 0, the Watchdog Timer will generate WIRQ when it times out. When 1, the Watchdog Timer will generate a reset when it times out.
6:2	BMB	The number of clock cycles which must occur before the Watchdog Timer times out. A value of 00000 disables the Watchdog Timer function.
1:0	WRB	The clock frequency of the Watchdog Timer, as shown in Table 86.

Table 86: Watchdog Timer Frequency Select

WRB Value	Watchdog Timer Frequency
00	16 Hz
01	4 Hz
10	1 Hz
11	1/4 Hz

第七个 bit 是使能 watchdog 的，第 6 到 2bit 是设置计时器超时时间，第 0 到 1bit 是时钟频率选择的。

Watchdog 测试

串口命令行中输入命令 su;然后输入 echo 1>/sys/class/rtc_am1805/disable_feeddog,设备能够自动 reset 重启

3.Alarm/cdt 调试

cdt/alarm 都是为了在 nIRQ2 上输出想要的电平或脉冲中断。现在调试的 Alarm 功能是为了实现定时开机的功能。当然，这个功能也可以用 CDT（计时器来做），但 ALARM 和 CDT 不同的是设置时间的方式，Alarm 可直接设置为我们熟悉的时分秒的时间格式，但 CDT 只能设置多久后触发中断，不能直接指定时间。

硬件上我们需要从 nIRQ2 引脚输出 alarm 的中断，中断的类型可以分为电平和脉冲。

alarm 时间的寄存器，当设置对应的时间并启动 alarm 功能后，IC 会自动对比 RTC 时间和 alarm 时间，如果对应的寄存器值都匹配，就会发出中断。

					Tens				
0x04	Date	GP5	GP4	Date - Tens		Date - Ones			
0x05	Months	GP8	GP7	GP6	Months - Tens	Months - Ones			
0x06	Years	Years - Tens				Years - Ones			
0x07	Weekdays	GP13	GP12	GP11	GP10	GP9	Weekdays		
0x08	Hundredths Alarm	Hundredths Alarm - Tenths				Hundredths Alarm - Hundredths			
0x09	Seconds Alarm	GP14	Seconds Alarm - Tens			Seconds Alarm - Ones			
0x0A	Minutes Alarm	GP15	Minutes Alarm - Tens			Minutes Alarm - Ones			
0x0B	Hours Alarm (24 hour)	GP17	GP16	Hours Alarm - Tens		Hours Alarm - Ones			
0x0B	Hours Alarm (12 hour)	GP17	GP16	AM/PM	Hours Alarm - Tens	Hours Alarm - Ones			
0x0C	Date Alarm	GP19	GP18	Date Alarm - Tens		Date Alarm - Ones			
0x0D	Months Alarm	GP22	GP21	GP20	Months Alarm - Tens	Months Alarm - Ones			
0x0E	Weekdays Alarm	GP27	GP26	GP25	GP24	GP23	Weekdays Alarm		
0x0F	Status	CB	BAT	WDT	BL	TIM	ALM	EX2	EX1

设置的相关的寄存器，然后用示波器量中断脚，看看时间到了后是否有中断输出。

```
关闭Alarm Interrupt repeat
echo "0x18 0x00">/sys/class/rtc_am1805/write_reg
1.Alarm
关闭AIE使能，设置IM为电平模式
echo "0x12 0xe0">/sys/class/rtc_am1805/write_reg

清除ALM Status
echo "0x0f 0x00">/sys/class/rtc_am1805/write_reg

设置为IRQ2中断
echo "0x11 0x0c">/sys/class/rtc_am1805/write_reg

设置时间
echo "0x08 0x01">/sys/class/rtc_am1805/write_reg
echo "0x09 0x01">/sys/class/rtc_am1805/write_reg
echo "0x0a 0x50">/sys/class/rtc_am1805/write_reg
echo "0x0b 0x00">/sys/class/rtc_am1805/write_reg
echo "0x0c 0x02">/sys/class/rtc_am1805/write_reg
echo "0x0e 0x01">/sys/class/rtc_am1805/write_reg
一年一次
echo "0x18 0xc6">/sys/class/rtc_am1805/write_reg

启动
echo "0x12 0xe4">/sys/class/rtc_am1805/write_reg
```

4.AM1805 外部晶振校正方法

由于实测 RTC 时间 1 天会慢 30 秒左右，所以要做外部晶振校准，把晶振频率调到 16.3840KHz。

校准参考资料目录的《T0001_RTC 外部晶振校准.pdf》文档。计算公式的参考值是 16.384KHz。做校准先根据例程 am_config_sqw 输出方波，方波输出到 CLKOUT 或者 FOUT 引脚上，板子上现在有一个测试点，是连接到 CLKOUT 上，也可以输出到这个引脚。下面是我根据 am_config_sqw 计算，得到需要设置的寄存器的值，直接在命令行输入后就可以输出方波：

```
echo "0x13 0xb6">/sys/class/rtc_am1805/regwrite
echo "0x11 0x11">/sys/class/rtc_am1805/regwrite
```

方波输出后，然后用频率计测量频率，注意频率用示波器测不准，因为示波器的精度只能到小数点后两位，所以要用专门的频率计，频率计需要向供应商 FAE 借。

校准值计算方法：

假如用频率计量到的频率为 16.3880Khz，则可用下面计算公式计算出值来，基准值是 16.3840

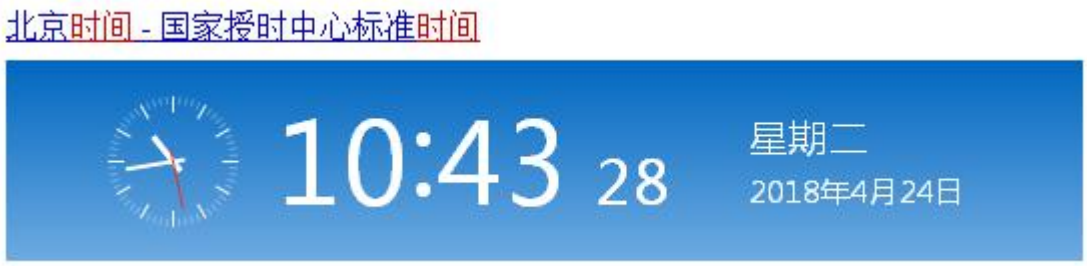
(（16.384-16.388）/16.384)*1000000=-244.14

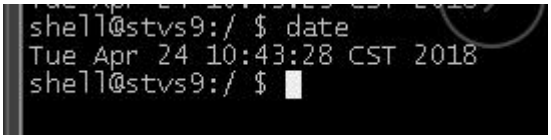
计算出的校准值小数点可以不记。将-244 填入到驱动 H 文件的宏 CALIBRATION_VALUE，编译烧录到板子后再量频率，正常的话应该是 16.384KHz。

H 头文件里面校准控制还有另外一个宏 CALIBRATION_POSITIVELY_BIASED_VALUE，这个宏是微调值。添加

这个微调值是因为当初用-244 的时候，一直调不到 16.384KHz，查代码的二进制计算过程，发现-244 在计算校准值时，中间判断会通过 switch case，>-129 会进入一个 case，否则进入另外一个 case，而我们计算的值刚好是 128.4 左右，这时代码是进入>-129 部分，但调出来的频率，没有达到效果，所以根据实际情况，加了一个这个宏，将这个宏微调加 3 到 5 左右试试，这样 switch 就会进入另一个 case。目前这个值设置的是 5，测试频率看到的是 16.384。

另外，矫正后时间是否精准，需要把板子上的时间先网络同步，时区设置里面调一下+8 时区，时间同步完后要把网络关掉，不然下次开机后会自动同步，这样可能会导致看不到是否校准了。参考时间尽量用网络时间，尽量不要用电脑上的时间，因为电脑上的时间可能也是不精准的，但网络时间一般是原子钟的，时间比较精确，看板子上 RTC 跑的时间和网络时间，秒数对得上就可以了。





由于晶振用料的好与不好的差异会直接导致时间偏差，所以不同项目的板子都要做个时间测试，就是放个几天然后看时间偏差，如果有偏差，需要到供应商 FAE 借频率计调一下.

5.AM1805 已知问题：

1.安卓时钟应用设置闹钟，时间到了后 APK 会提示，但用示波器量不到中断出来。

这个问题是，驱动已经实现了 alarm 的 syspend 和 resume 函数，但系统没 suspend 的时候，设置闹钟并不会往 rtc 芯片的寄存器上写数据，因为不需要唤醒系统。上层直接把闹钟设置写到/dev/alarm，AlarmThread 会不停的去轮寻下一个时间有没有闹钟。系统要是进入 susupend 的话，alarm 的 alarm_suspend 就把设置的闹钟写到驱动 rtc 芯片的寄存器上去，然后即使系统 suspend 之后，闹钟通过 rtc 也能唤醒系统。

USB VBUS Power/UARTA0 TTL_EN 控制

这个功能的控制主要是 pin 脚提供接口出来，现在是做一个 pin 脚的 class 控制，通过 echo 可验证功能。

Usb vbus 控制方法：

串口进入命令行下，输入下面命令控制

1. 拉低：su;echo 0x00 > /sys/class/usb_vbus_pin/vbuspin
2. 拉高：su;echo 0x01 > /sys/class/usb_vbus_pin/vbuspin

Uart0 ttlen 控制方法：

串口进入命令行下，输入下面命令控制

1. 拉低：su;echo 0x00 > /sys/class/uart_ttl_pin/ttlenpin 进入 RS232 模式
2. 拉高：su;echo 0x01 > /sys/class/uart_ttl_pin/ttlenpin 进入 TTL 模式

发送数据 echo “字符串” > /dev/ttyS2

客户 chromecast 和 usb 以太网同时接入后开机以太网连接不上问题修改要点

同时接入后，开机后，busybox ifconfig -a 可以看到 usb 以太网的网络节点是 usbnet0，chromecast 的网络节点是 usbnet1。Usbnet0 的 ip 地址可以获取到，但是不能上网，设置里面以太网选项，点击查看 ip 是没有获取到的。Logcat 看到是 usbnet0 先加载，usbnet1 后加载，usbnet1 的相关配置会冲掉 usbnet0 的配置，导致以太网不能上网。

解决方法是底层将 usb 网卡口默认为 usbnet0，另外两个 usb 口接的 chromecast 固定为 usbnet1 和 usbnet2。Framework 层修改代码保证 usbnet0 的相关配置不会被 usbnet1 或 usbnet2 的值覆盖掉。单独接 chromecast 的 时候，也会将将网络节点固定为 usbnet1 或者 usbnet2，例如：

```
root@stvs9:/ # busybox ifconfig -a
ip6tnl0   Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:1452  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:4096  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

tunl0     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usbnet2   Link encap:Ethernet  HWaddr 00:E0:4C:36:00:08
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 10:D0:7A:84:14:6D
          inet addr:192.168.123.198  Bcast:192.168.123.255  Mask:255.255.255.0
          inet6 addr: fe80::12d0:7aff:fe84:146d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:175 errors:0 dropped:0 overruns:0 frame:0
          TX packets:293 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:27846 (27.1 KiB)  TX bytes:28137 (27.4 KiB)
```

修改方法是通过识别 usb 的 id 节点，并更改名称固定 usbnet1 和 usbnet2：

```
jsh@ubt124:/data/jsh/tmp/s905x_b/common$ show 769d8e3e02215f95246ac6552633973d548b4d2b
commit 769d8e3e02215f95246ac6552633973d548b4d2b
Author: jsh <jiang.shaohui@geniatech.com>
Date:   Wed Apr 18 16:59:47 2018 +0800

    MOD: 更改usb接入chromecast节点名称

diff --git a/drivers/net/usb/usbnet.c b/drivers/net/usb/usbnet.c
index eb103bc..62d35d2 100644
--- a/drivers/net/usb/usbnet.c
+++ b/drivers/net/usb/usbnet.c
@@ -1630,8 +1630,16 @@ usbnet_probe (struct usb_interface *udev, const struct usb_device_id *prod)
                // can rename the link if it knows better.
                if ((dev->driver_info->flags & FLAG_ETHER) != 0 &&
                    ((dev->driver_info->flags & FLAG_POINTTOPOINT) == 0 ||
-                    (net->dev_addr [0] & 0x02) == 0))
-                       strcpy(net->name, "usbnet%d");
+                    (net->dev_addr [0] & 0x02) == 0)){
+                       if (!strcmp(xdev->devpath, "1.1")){
+                               strcpy(net->name, "usbnet2");
+                       }else if (!strcmp(xdev->devpath, "2")){
+                               strcpy(net->name, "usbnet1");
+                       }else{
+                               strcpy(net->name, "usbnet%d");
+                       }
+               }
+
                /* WLAN devices should always be named "wlan%d" */
                if ((dev->driver_info->flags & FLAG_WLAN) != 0)
                        strcpy(net->name, "wlan%d");
@@ -1698,7 +1706,6 @@ usbnet_probe (struct usb_interface *udev, const struct usb_device_id *prod)
                   xdev->bus->bus_name, xdev->devpath,
                   dev->driver_info->description,
                   net->dev_addr);
-
        // ok, it's ready to go.
        usb_set_intfdata (udev, dev);
```