目录

[uboot移植 2](#_Toc36710892)

[Uboot启动流程 3](#_Toc36710893)

[reset //(arch/arm/cpu/armv7/start.S) 3](#_Toc36710894)

[SPL阶段： 3](#_Toc36710895)

[Uboot阶段： 4](#_Toc36710896)

[DDR配置涉及代码 14](#_Toc36710897)

[Mpu、ddr、dsp频率配置方法 14](#_Toc36710898)

[ddr频率设置代码 14](#_Toc36710899)

[EMIF工具输出I / O控制寄存器结构 15](#_Toc36710900)

[EMIF工具输出一系列寄存器，以方便动态内存管理器的映射 15](#_Toc36710901)

[EMIF工具输出包含EMIF配置的结构（以下示例）： 16](#_Toc36710902)

[EMIF工具输出用于配置外部PHY控制寄存器（以下示例）的结构 16](#_Toc36710903)

[与板子选择相关的代码 17](#_Toc36710904)

[驱动移植 17](#_Toc36710905)

[1、usb 17](#_Toc36710906)

[spi 18](#_Toc36710907)

[can 18](#_Toc36710908)

[4、以太网 19](#_Toc36710909)

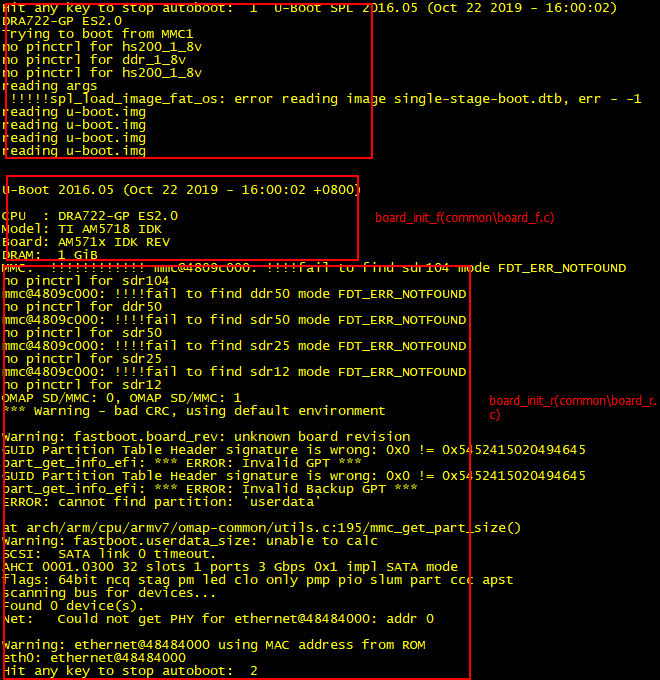
[设备树 19](#_Toc36710910)

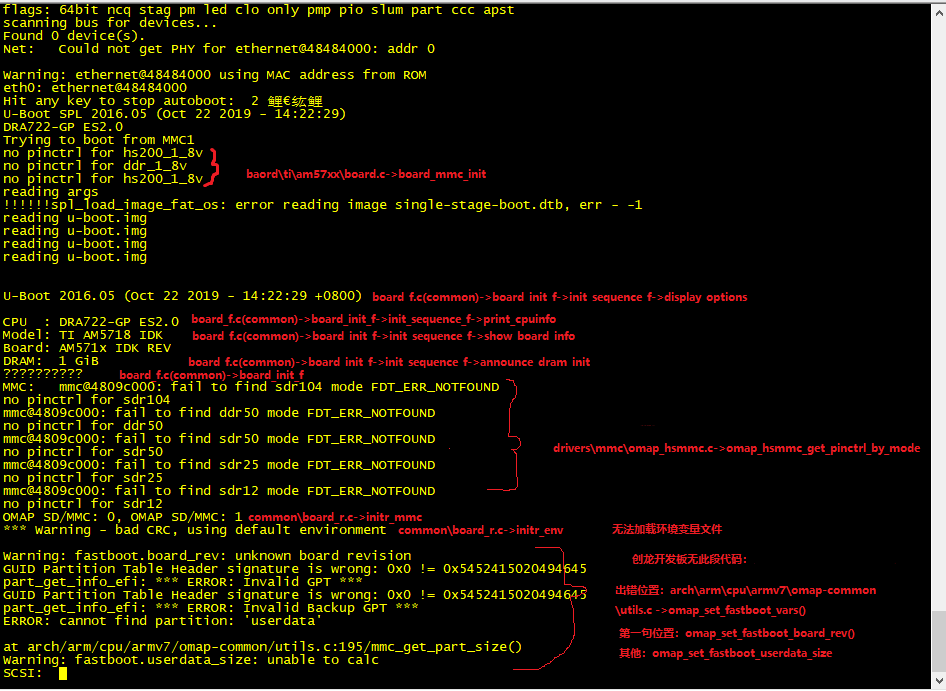
[一、DTS编写语法 20](#_Toc36710911)

[二、常用函数 20](#_Toc36710912)

[4.日常积累 25](#_Toc36710913)

# uboot移植

****

****

## Uboot启动流程

### reset //(arch/arm/cpu/armv7/start.S)

save\_boot\_params\_ret //(arch/arm/cpu/armv7/start.S)

disable interrupts

cpu\_init\_cp15 //(arch/arm/cpu/armv7/start.S)

Invalidate L1 I/D

disable MMU stuff and caches

cpu\_init\_crit //(arch/arm/cpu/armv7/start.S)

lowlevel\_init //(arch/arm/cpu/armv7/lowlevel\_init.S)

Setup a temporary stack

Set up global data

s\_init //( arch\arm\mach-omp2\hwinit-common.c)

**\_Main(arch\arm\lib\crt0.S)**

### SPL阶段：

**Board\_init\_f(arch\arm\mach-omp2\hwinit-common.c)**

Early\_system\_init(arch\arm\mach-omp2\hwinit-common.c)

Init\_omap\_revision(arch\arm\mach-omp2\omap5\hwinit.c)根据芯片版本进行相应设置

Hw\_data\_init(board/ti/am57xx/board.c) 根据芯片版本进行相应设置

Do\_io\_settings()//通过emif tool 配置

Do\_board\_detcect(board/ti/am57xx/board.c)开发板型号检测

Ti\_i2c\_eeprom\_am\_get(board/ti/common/board\_detect.c) 将从eeprom读取板子信息改为自定义

Vocres\_init(board/ti/am57xx/board.c) core、mpu、eve、gpu电压初始化，需更改PMIC

Prcm\_init(arch/arm/match-omap2/clocks-common.c)

scale\_vcores(\*omap\_vcores);(输入参数在Vocres\_init中有修改)

setup\_dplls

sdram\_init(arch/arm/mach-omap2/emif-common.c)初始化ddr3，修改频率，emif\_configuration

dmm\_init();通过emif tool配置

do\_ sdram\_init(arch/arm/mach-omap2/emif-common.c)通过emif tool配置

ddr3\_init();(arch/arm/mach-omap2/emif-common.c)

dra7\_ddr3\_init();

dra\_ddr3\_leveling(base,regs)重要，需根据EMIF\_RegisterConfiguration.excel文件配置regs参数

**board\_init\_r(common/spl/spl.c)**

spl\_board\_init(arch/arm/mach-omap2/boot-common.c)

save\_omap\_boot\_params(arch/arm/mach-omap2/boot-common.c)保存ROM中的启动参数

preloder\_console\_init(初始化串口，打印uboot版本信息和cpu信息)

Spl\_load\_image(common/spl/spl.c)

Spl\_mmc\_load\_image(common/spl/spl\_mmc.c)

Spl\_mmc\_init(common/spl/spl\_mm1c.c)mmc初始化，此处报错

Spl\_mmc\_do\_fs\_boot(common/spl/spl\_mmc.c)加载完成uboot.img,进入uboot阶段

### Uboot阶段：

**Board\_init\_f（common\board\_f.c）与spl阶段board\_init\_f不同**

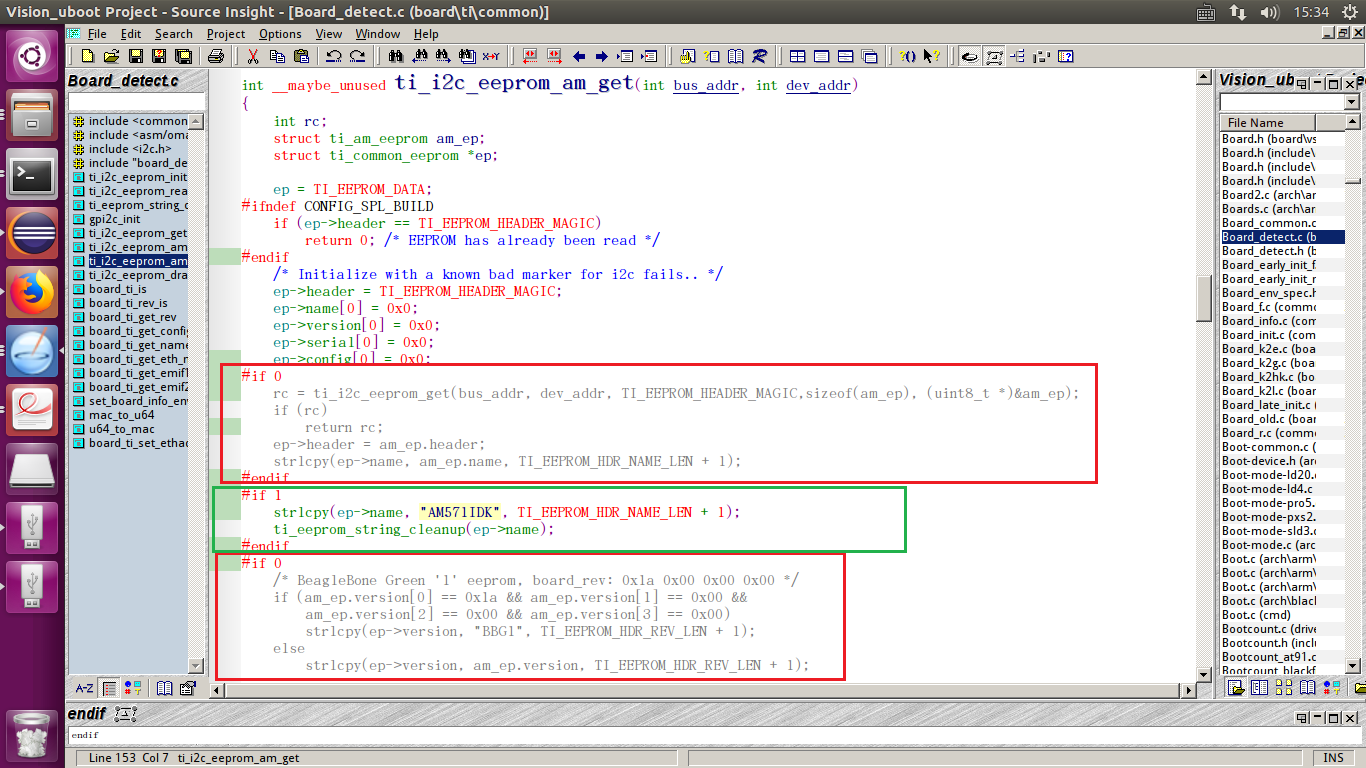
Initcall\_run\_list(init\_sequence\_f)结构体init\_sequence\_f中包含此阶段中所有的函数表

**Board\_init\_r（common\board\_r.c）**

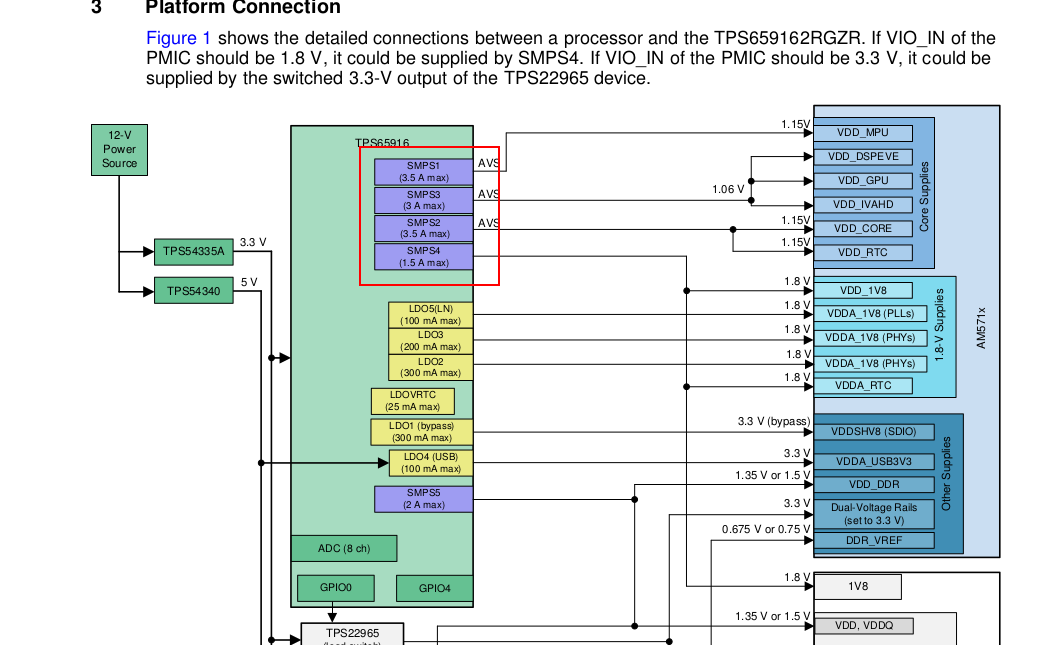
Initcall\_run\_list(init\_sequence\_r) 结构体init\_sequence\_r中包含此阶段中所有的函数表

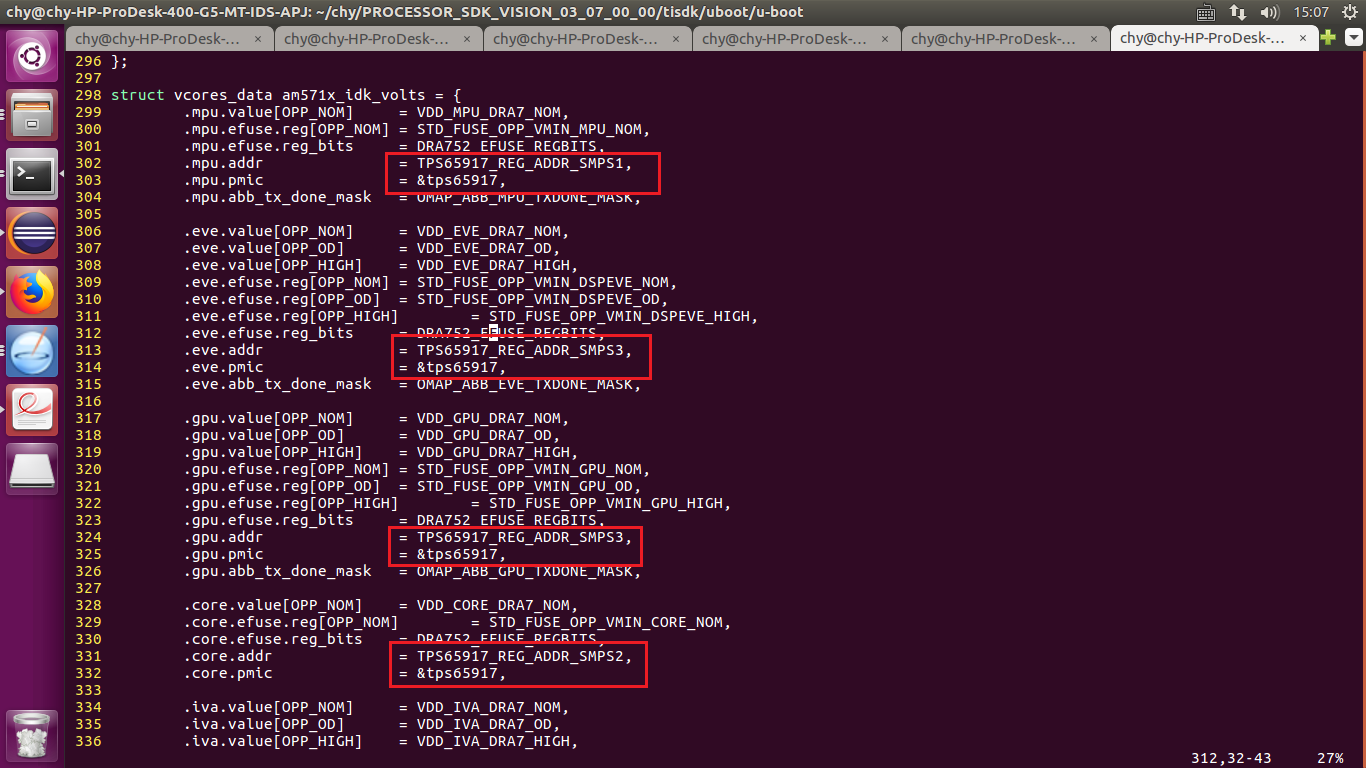
移植修改记录

1. board\ti\common\board\_detect.c :ti\_i2c\_eeprom\_am\_get()将从eeprom读取板子信息改为自定义

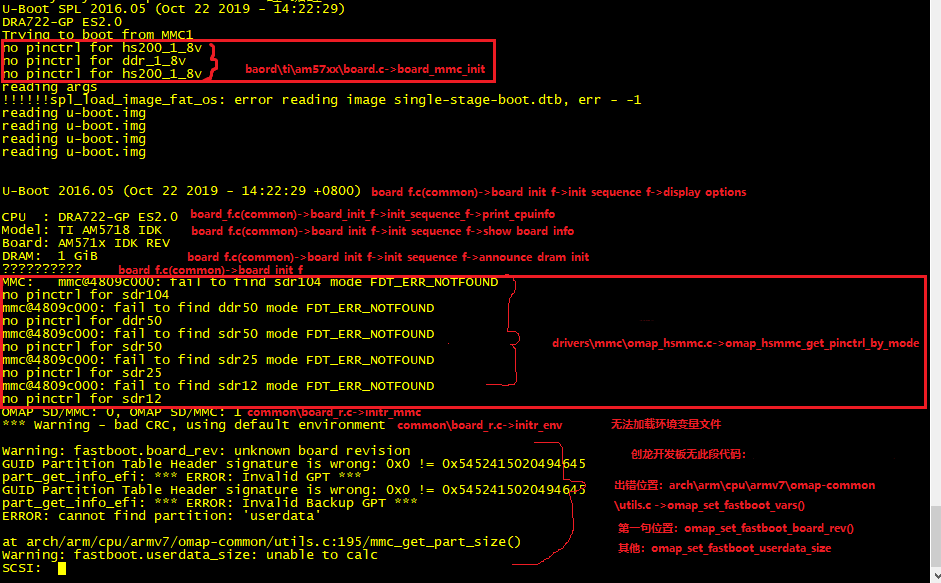


1. 修改board\ti\am57xx\board.c:vcores\_init()后串口有打印信息，PMIC默认TPS659038修改为TPS65916后串口有打印信息。



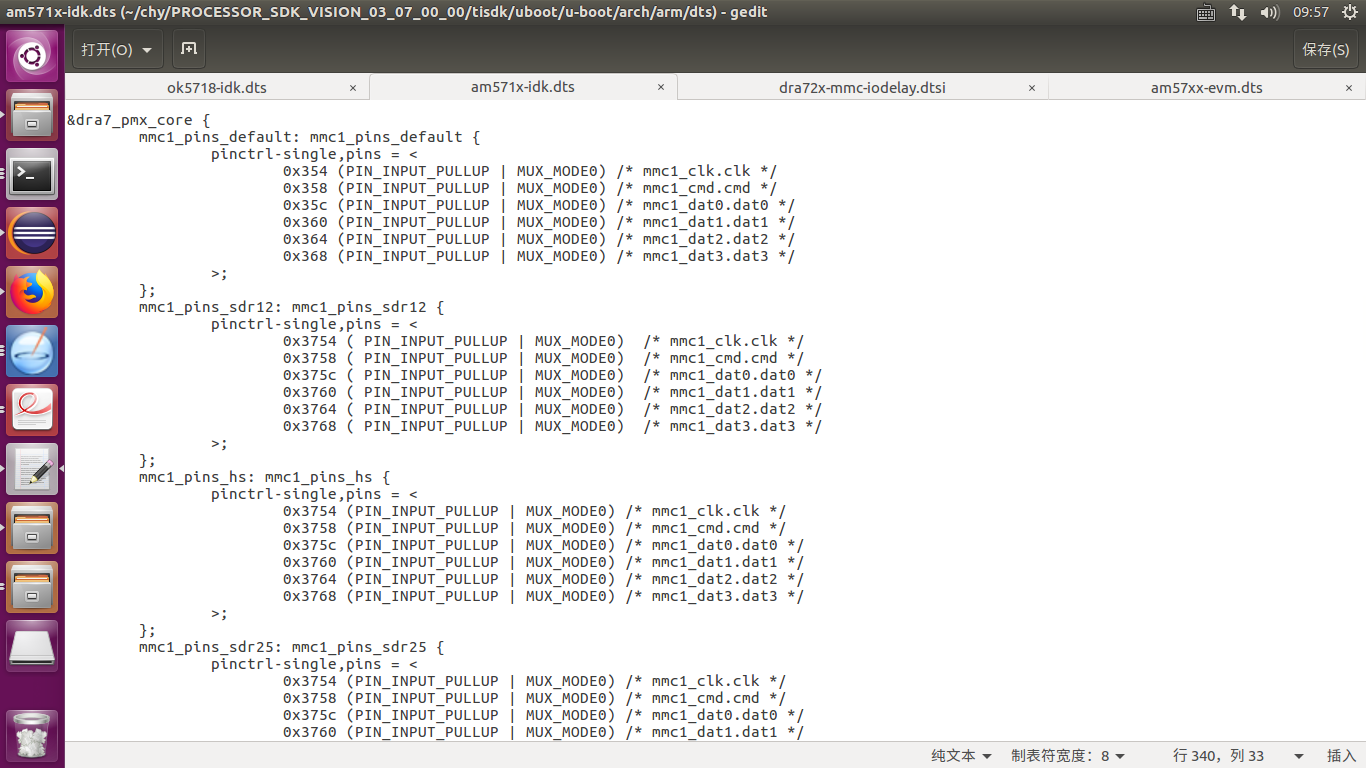
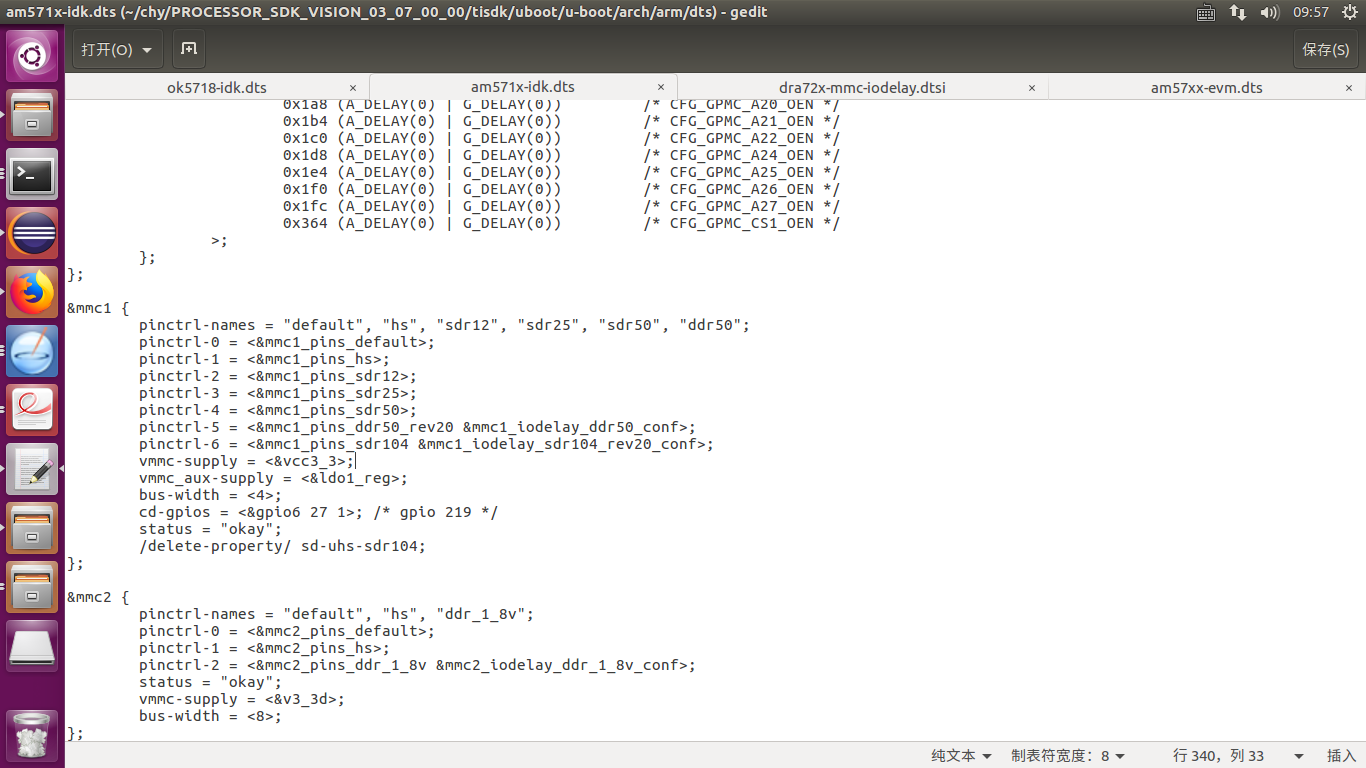


1. uboot输出错误1：

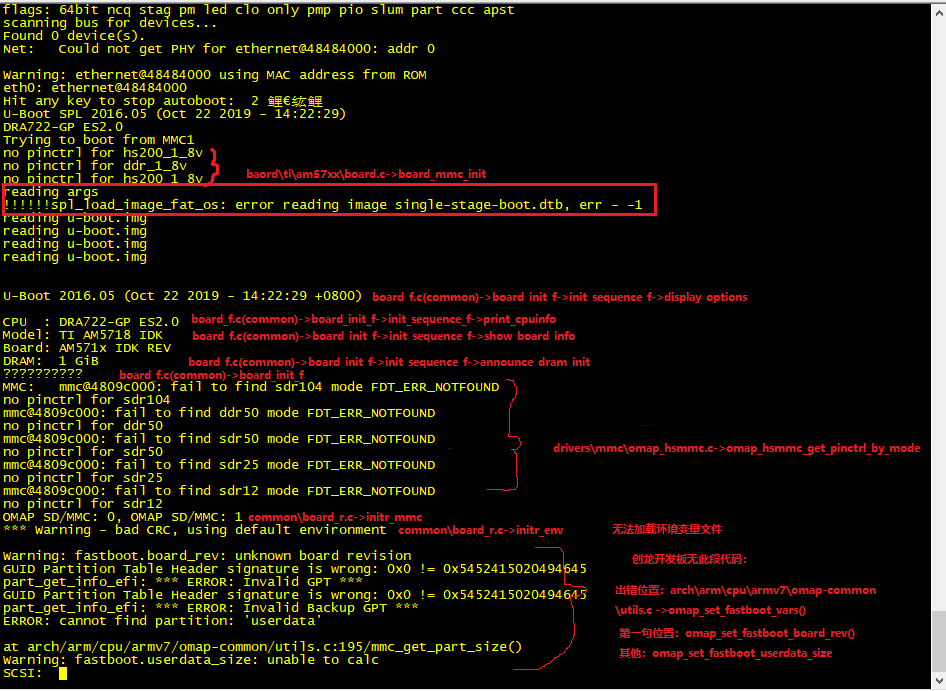


错误原因：pinctrl作用为设置mmc读写速度模式，未设置这部分模式导致错误。

解决办法：在uboot设备树am571x-idk.dts中添加对应模式：添加代码如下图

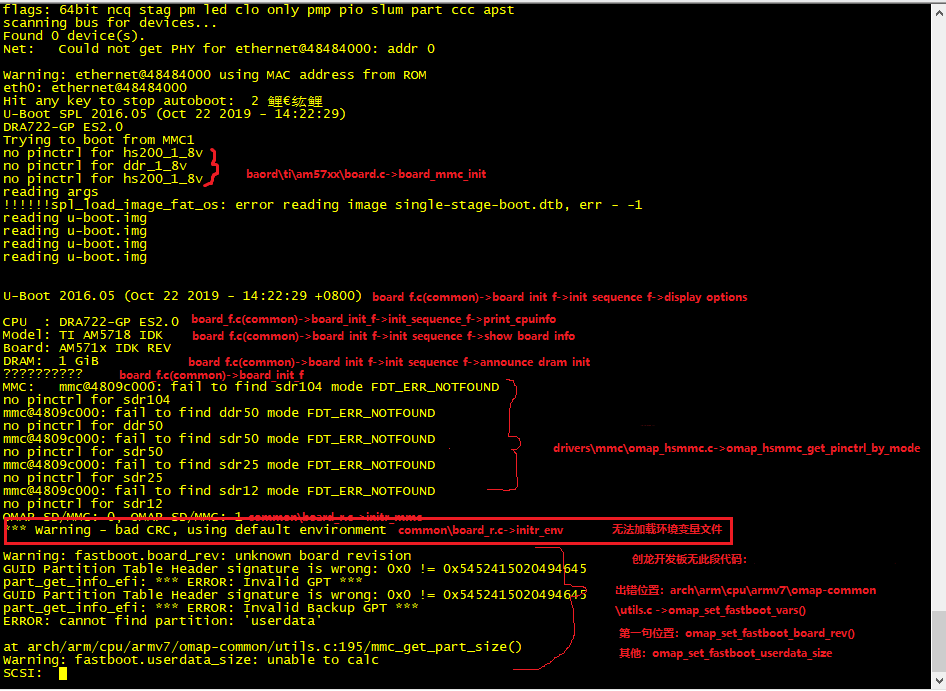


1. uboot输出错误2：



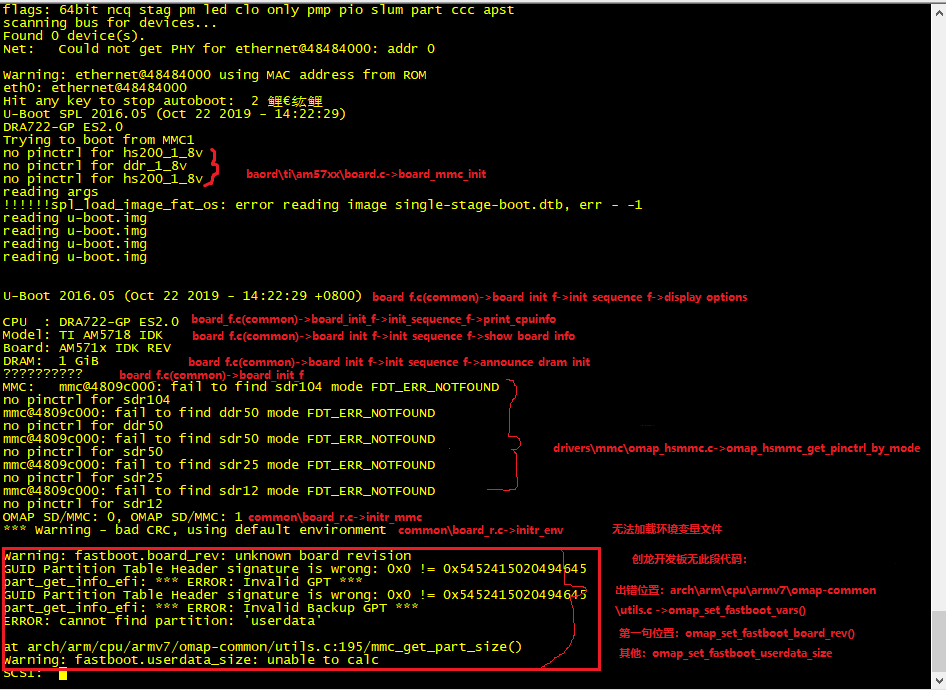
错误原因：此部分为通过MLO直接引导内核启动的程序，目前主流为MLO->uboot->kernel启动方式，所以不需此部分代码，将其注释。

1. uboot输出错误3：



警告原因，uboot启动会自动读取mmc中是否存在自定义保存的环境变量，若有则会优先读取emmc中的环境变量，没有则输出此警告

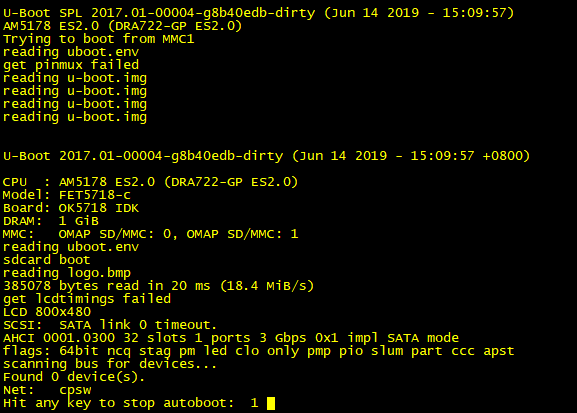
1. uboot输出错误4：

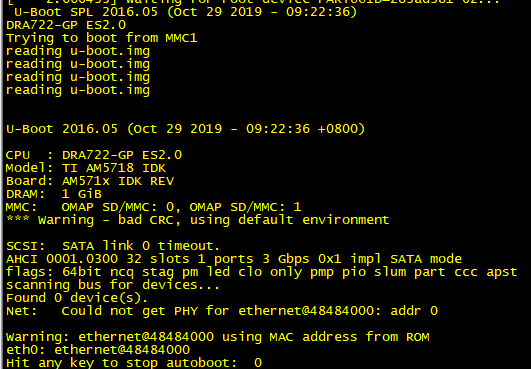


在部分uboot源码中不存在此段代码，暂不理解此段代码作用，不影响引导内核启动。

6、移植成功后可加载内核启动



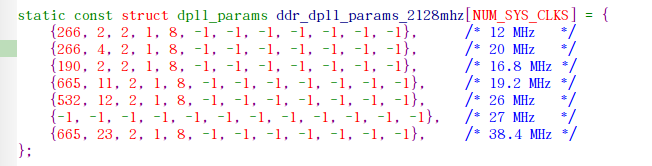
OK5718原始打印信息移植后打印信息



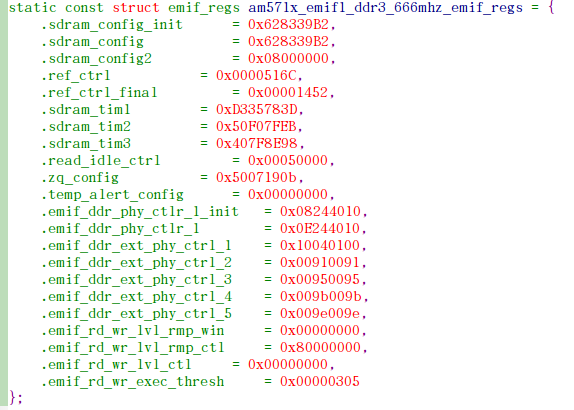
7、ddr使用532MHz频率系统运行正常，修改为666MHz后系统有概率启动失败。测试使用飞凌原始MLO文件＋自己移植uboot文件启动没有问题，使用自己移植MLO文件＋飞凌原始uboot文件启动会出现无法启动现象。最终确定为MLO阶段问题所导致。后经过详细检查此部分代码发现漏设置一处所导致。修改后进行多次启动测试无异常。

Fdpll = Fref x 2 x M / (N+1) = 20x2 x 266 / (4 + 1) = 2128

CLKOUT\_M2 = Fdpll / (2 x M2) = 2128 / (2 x 2) = 532MHz

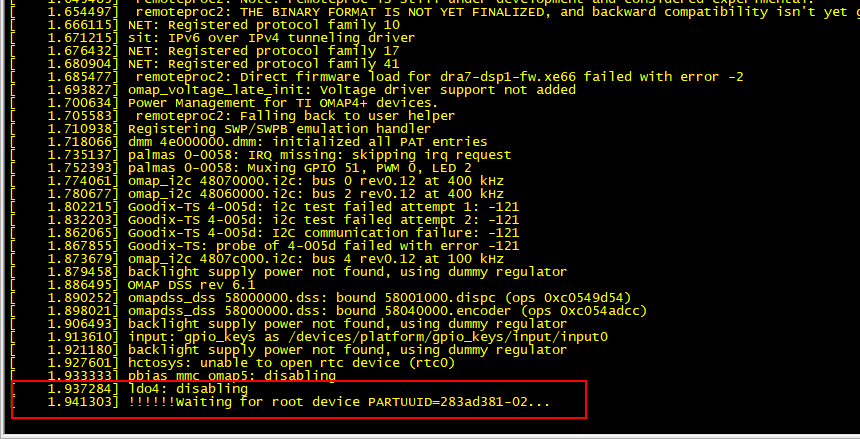


修改下图部分后解决问题



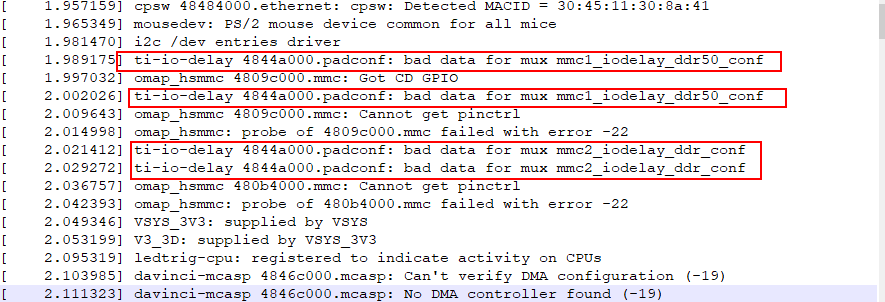
引导VinsonSDK kernel启动截图

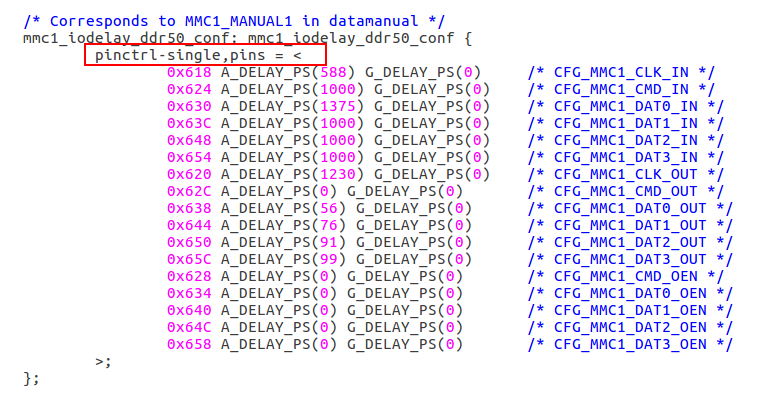
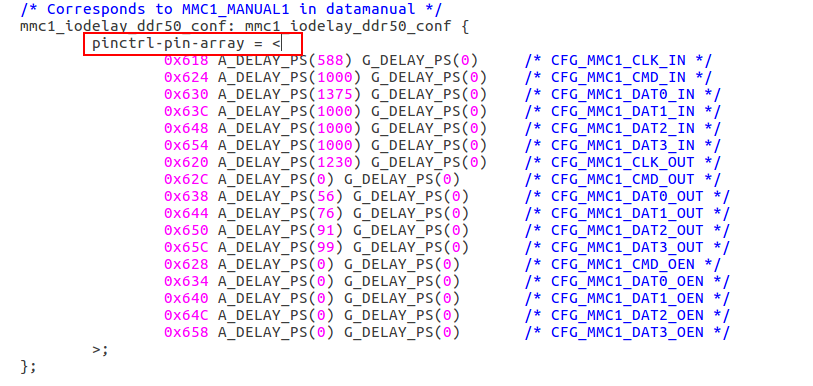
错误位置:init\do\_mounts.c->prepare\_namespace()



无法挂载文件系统，将文件系统拷贝至ram中可以启动，在/dev/下没有sd卡和emmc设备。

1. 怀疑linux的mmc驱动原因导致，将飞凌mmc驱动移植至visionSDK还是同样的问题。
2. 怀疑pmic原因导致，使用万用表测试开发板电源芯片的tp点，电压均正常。





添加环境变量支持uboot从SD卡读取镜像文件烧写进emmc

emmc分区

setenv xld\_start\_blk 0x100

setenv xld\_size\_blks 0x200

setenv ub\_start\_blk 0x300

setenv ub\_size\_blks 0x800

setenv ub1\_start\_blk 0xb00

setenv ub1\_size\_blks 0x800

setenv fdt\_start\_blk 0x1300

setenv fdt\_size\_blks 0x100

setenv k\_start\_blk 0x1400

setenv k\_size\_blks 0x2800

setenv fs\_start\_blk 0x3c00

setenv fs\_size\_blks 0x2800

saveenv

内存加载地址

setenv spl\_buff 0x81000000

setenv ub\_buff 0x81060000

setenv knl\_buff 0x81100000

setenv fs\_buff 0x81500000

setenv fdt\_buff 0x81030000

saveenv

添加环境变量

/\*擦除emmc全部内容\*/

erase\_mmc\_all=mmc dev ${mmc\_regular}; echo \*\*\* saving partiton000 table \*\*\*;mmc read ${ptable\_buff} 0 1;echo erasing MLO...; mmc erase ${xld\_start\_blk} ${xld\_size\_blks}; echo erasing U-BOOT 1...; mmc erase ${ub\_start\_blk} ${ub\_size\_blks}; echo erasing U-BOOT 2...; mmc erase ${ub1\_start\_blk} ${ub1\_size\_blks}; echo erasing FDT...; mmc erase ${fdt\_start\_blk} ${fdt\_size\_blks};echo erasing kernel...; mmc erase ${k\_start\_blk} ${k\_size\_blks}; echo erasing rootfs...; mmc erase ${fs\_start\_blk} ${fs\_size\_blks}

/\*读取全部内容到内存中\*/

mmc\_fat\_read\_all=mmc dev ${mmc\_update};fatload mmc ${mmc\_update} ${spl\_buff} ${mlo};fatload mmc ${mmc\_update} ${ub\_buff} ${u-boot}; fatload mmc ${mmc\_update} ${knl\_buff} ${bootfile}; fatload mmc ${mmc\_update} ${fdt\_buff} ${fdtfile}; fatload mmc ${mmc\_update} ${fs\_buff} ${ramdisk}

/\*写入内容到emmc中\*/

update\_all=mmc dev ${mmc\_regular}; echo flashing bootloader...; mmc write ${spl\_buff} ${xld\_start\_blk} ${xld\_size\_blks}; mmc write ${ub\_buff} ${ub\_start\_blk} ${ub\_size\_blks}; mmc write ${ub\_buff} ${ub1\_start\_blk} ${ub1\_size\_blks}; echo flashing kernel...; mmc write ${knl\_buff} ${k\_start\_blk} ${k\_size\_blks}; echo flashing FDT...; mmc write ${fdt\_buff} ${fdt\_start\_blk} ${fdt\_size\_blks}; echo flashing rootfs...; mmc write ${fs\_buff} ${fs\_start\_blk} ${fs\_size\_blks}; echo \*\*\* recovering partition table \*\*\*; mmc write ${ptable\_buff} 0 1

/\*更新系统，先擦除全部，在读取到内存中，最后写入\*/

update\_sys=echo entering update mode...;mmc dev ${mmc\_update}; if mmc rescan; then echo found SD/MMC in slot(1), enter upgrade mode.;if run erase\_mmc\_all; then run mmc\_fat\_read\_all;run update\_all;echo update system sucessfully!;echo Please reboot and unplug the card...;run halt;fi;fi

设备树：

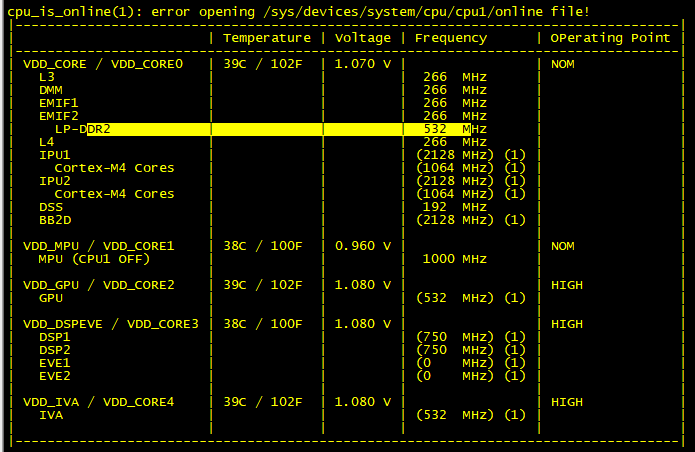
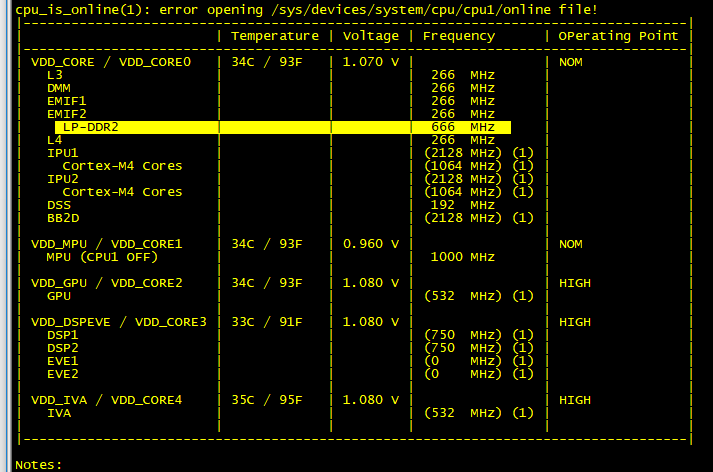
ok5718-idk.dts

dra72x.dtsi

dra70.dtsi

dra72xx-mmc-iodelay.dtsi

am57xx-industrial-grade.dtsi

Ddr频率设置方法：<https://e2e.ti.com/support/processors/f/791/p/756609/2796675?tisearch=e2e-sitesearch&keymatch=am57xx%2525252520Modify%2525252520DDR%2525252520frequency#2796675>

modprobe g\_multi file=/dev/mmcblk1p1,/mnt/p1/static.img,/dev/mmcblk0p1 luns=3 ro=0,1,1 stall=0,0,0 removable=1,1,1 iSerialNumber=3000111 iProduct=zhdgnss iManufactuy=zhd\_survey



Linux提供了五种内核类型，对于内核类型的选择，可以参考如下的解释：  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
If you do not require low latency for your system then please use the -generic kernel.  
If you need a low latency system (e.g. for recording audio) then please use the -preempt kernel as a fist choice. This reduces latency but doesn't sacrifice power saving features. It is available only for 64 bit systems (also called amd64).  
If the -preempt kernel does not provide enough low latency for your needs (or you have an 32 bit system) then you should try the -lowlatency kernel.  
If the -lowlatency kernel isn't enough then you should try the -rt kernel  
If the -rt kernel isn't enough stable for you then you should try the -realtime kernel  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## DDR配置涉及代码

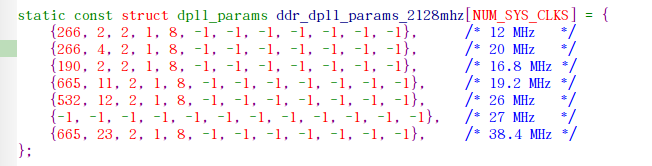
### Mpu、ddr、dsp频率配置方法

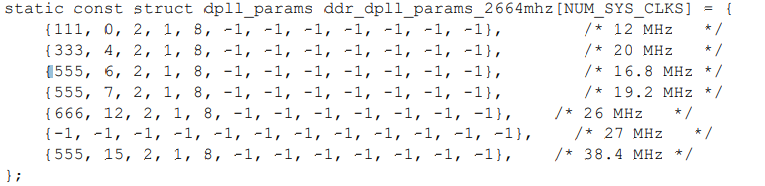
<https://e2e.ti.com/support/processors/f/791/t/822591?tisearch=e2e-sitesearch&keymatch=mpu>

### ddr频率设置代码

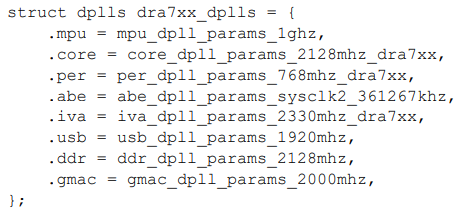
Fdpll = Fref x 2 x M / (N+1) = 20x2 x 266 / (4 + 1) = 2128

CLKOUT\_M2 = Fdpll / (2 x M2) = 2128 / (2 x 2) = 532MHz

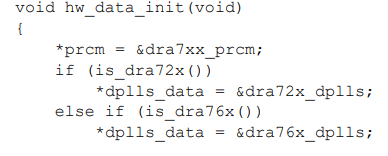




频率设置结构体在如下代码中使用

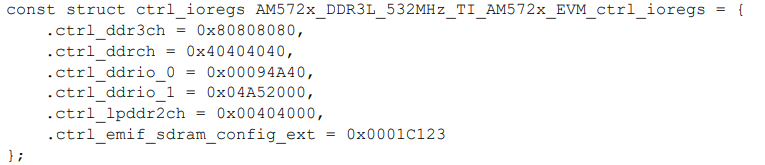


dra7xx\_dplls结构体在如下函数进行使用



### EMIF工具输出I / O控制寄存器结构

Early\_system\_init->do\_io\_settings()->io\_setting\_ddr3()

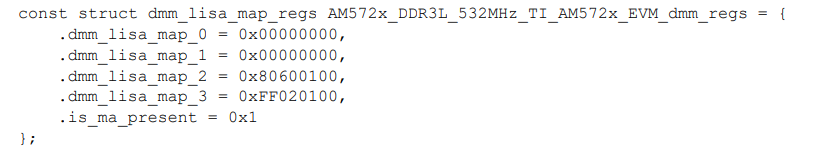


Ioregs结构体在get\_ioregs中被调用。

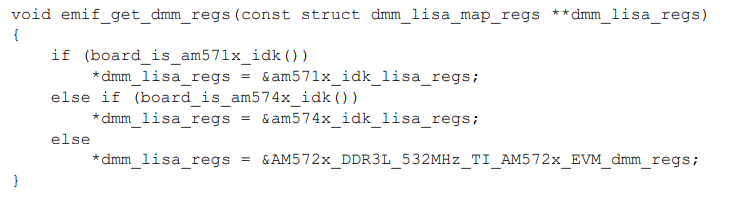
### EMIF工具输出一系列寄存器，以方便动态内存管理器的映射

Sdram\_init()->dmm\_\_init()

（下面的示例）：

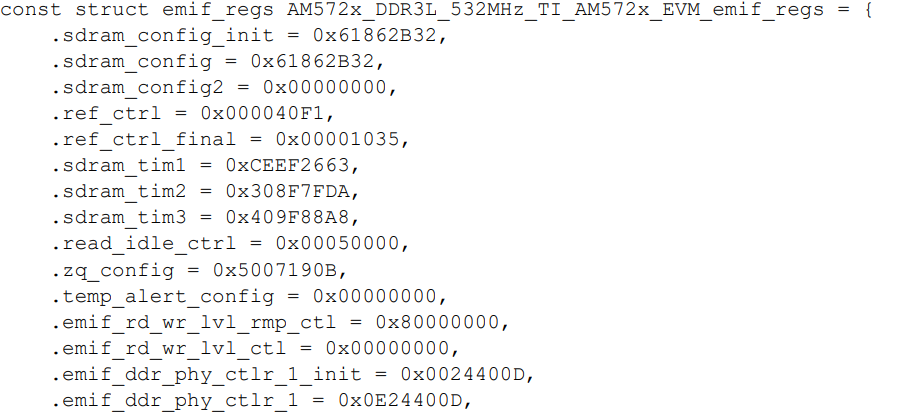


此结构在emif\_get\_dmm\_regs（）中使用。

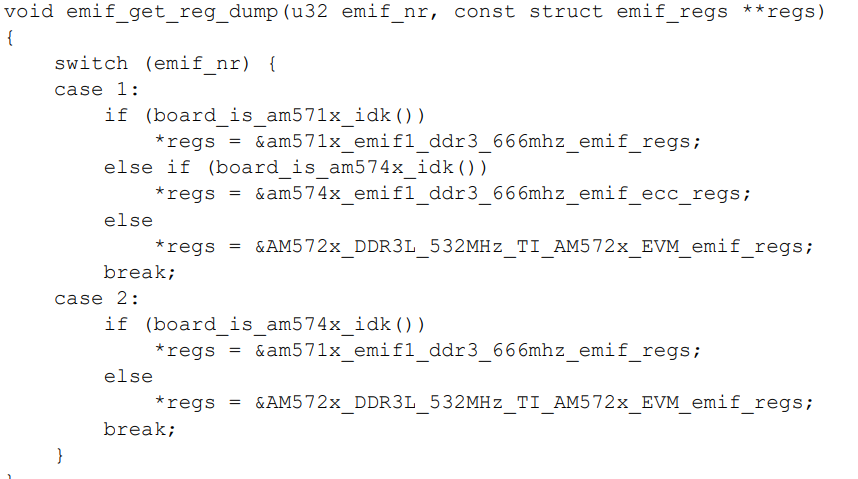


### EMIF工具输出包含EMIF配置的结构（以下示例）：

Sdram\_init()->do\_sdram\_init()

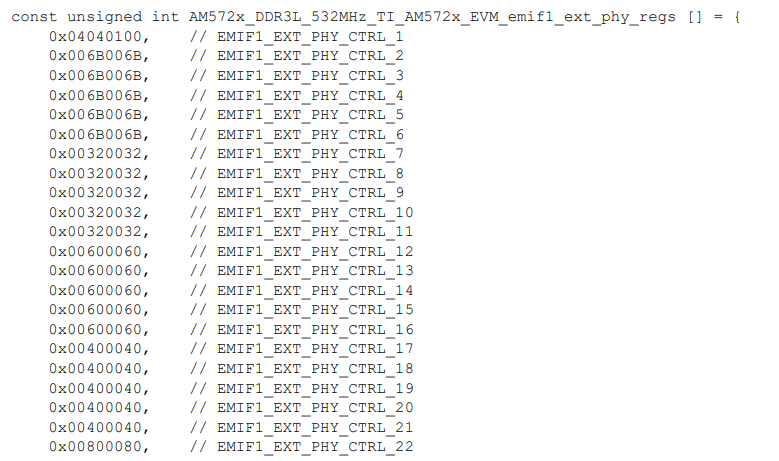


在emif\_get\_reg\_dump（）中使用此结构

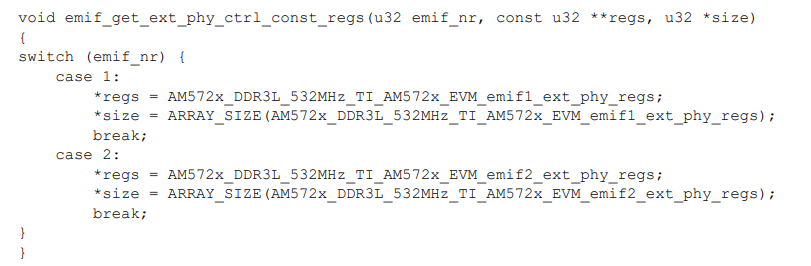


### EMIF工具输出用于配置外部PHY控制寄存器（以下示例）的结构

Sdram\_init()->do\_sdram\_init()



并用于emif\_get\_ext\_phy\_ctrl\_const\_regs（）中



## 与板子选择相关的代码

# 驱动移植

## 1、usb

存在问题：驱动可正常加载，插入usb无反应。

可能原因： ①内核版本不同，驱动略有差异。

②内核配置问题：未配置相应驱动

③设备树配置错误：不同内核版本设备树配置略有差异可能引起。

④硬件连接不同：飞凌开发板usb硬件连接方式不同可能引起问题（ID引脚悬空，跳线帽选择主从）。

分析：①对比两个版本内核usb相关驱动源码，虽然存在部分差异，驱动都为内核自带源码，并非跨版本移植代码，所以驱动源码存在问题的可能性不大。

②内核配置问题，根据飞凌开发板配置表配置内核后还是存在问题。

③硬件连接不同：ID引脚悬空，跳线帽选择主从，所以可能是此问题导致，所以设备树配置存在不同。由于visionsdk芯片型号不同，内核版本差异较大，创龙开发板内核在飞凌开发板启动也存在此问题，而创龙开发板内核版本与飞凌开发板内核同为linux4.9，芯片型号相同，所以选择先解决同芯片同内核版本下的问题，再解决visionSDK问题。

对比两个内核包含USB配置的dts文件,绿色为涉及到usb配置的内容相同的dts文件，红色为涉及到usb配置的内容不同的 dts文件

创龙开发板：

tl570x-evm.dts

am57xx-evm-common.dtsi

am57xx-beagle-x15-revb1.dts

am57xx-beagle-x15-common.dtsi

dra74x.dts

dra7.dtsi

dra7xx-clocks.dtsi

am57xx-commercial-grade.dtsi

dra74x-mmc-iodelay.dts

am57xx-evm-cmem.dtsi

am57xx-industrial-grade.dtsi

飞凌开发板

ok5718-idk.dts

dra72x.dtsi

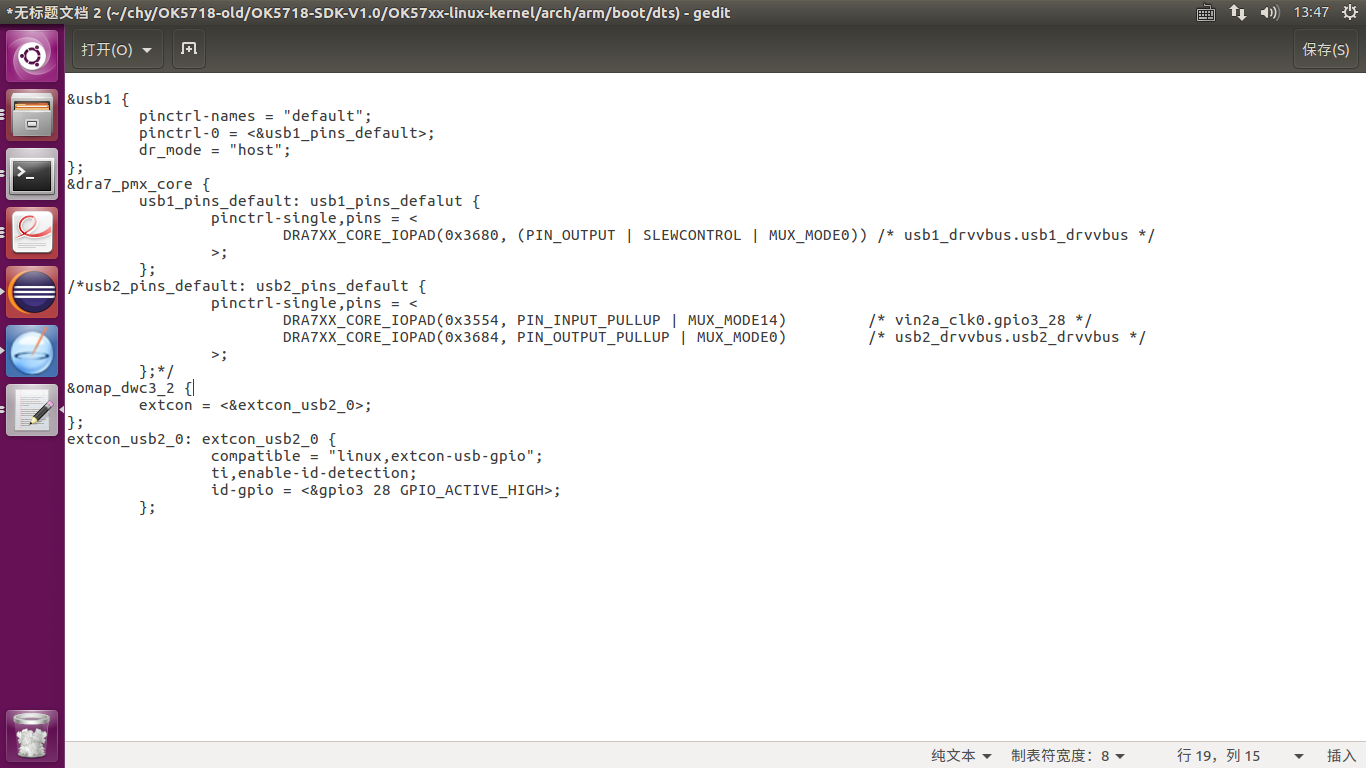
dra7.dtsi

dra7xx-clocks.dtsi

dra72x-mmc-iodelay.dtsi

am57xx-industrial-grade.dtsi

整理出如下差别，飞凌开发板缺少一下设置，无法检测到usb插入原因为usb2 ID引脚未配置，添加后此部分代码后功能正常。



## spi

未生成设备节点

原因：内核未配置，选上device drivers->spi support->user mode spi device driver support

## can

无法设置波特率，无法启动can，无canddump、cansend工具，设置波特率报错如下：

ip: either "dev" is duplicate, or "type" is garbage

原因：文件系统ip指令不支持

解决：移植iproute2的ip命令， iproute2源码下载地址：<https://mirrors.edge.kernel.org/pub/linux/utils/net/iproute2/>

解压并进入源码目录：

$ tar -xf iproute2-3.10.0.tar.gz

$ cd iproute2-3.10.0/

修改Makefile

修改CC为交叉编译的gcc

#CC = gcc

CC = arm-none-linux-gnueabi-gcc

只需要ip命令，所以简化编译内容:

#SUBDIRS=lib ip tc bridge misc netem genl man

SUBDIRS=lib ip

执行make

make编译成功后，将iproute2-3.10.0/ip目录下生成可执行的ip命令拷贝至文件系统/sbin替换原ip命令。

## 4、以太网

以太网复位引脚一直处于复位状态

# 设备树

 \*note:<name>[@<unit-address>]是节点的格式，其中unit-address是单位偏移地址，本人验证去掉[]内的内容依然可以运行，也没在source code找到必须存在的理由。

“/"代表根节点；

“model”是板的ID；

"compatible"是平台兼容，一般格式是"manufacturer,model"。内核或者uboot依靠这个属性找到相对应driver，若"compatible"出现多个属性，按序匹配driver；

“#address-cells”是address的单位(32bit)；

“#size-cells”是length的单位(32bit)；

"reg"是寄存器，格式是"<address,length>"，作为平台内存资源；

"aliase" 是别名，必须节点全称，可以通过地址引用获取；

”chosen“是板级启动参数；

"cpus"是SOC的CPU信息，可以改变运行频率或者开关CPU；

"memory"是板级内存的信息。

"interrupts"是中断控制器，根据SOC自定义格式，这里是<输入类型 中断号 触发方式>，作为平台中断资源；

“interrupt-controller”指示这个节点是中断控制节点；

"[label:]"如gic: interrupt-controller@1c81000，这个标签可以作为地址赋值到其他节点的属性；

“device\_type":设备类型，寻找节点可以依据这个属性；

"status"是开关节点设备的状态，取值"okay"或者"ok"表示使能，"disabled"表示失能。

## 一、DTS编写语法

## 二、常用函数

设备树函数思路是：

uboot启动时将设备树地址传给内核，内核解析设备树，并创建设备，初始化相关属性，驱动中通过of\_get\_XXX函数去获取设备树加载时创建的设备。想要知道of函数做了什么，就去追踪这个函数最后调用了什么，同时也就知道了内核解析设备树的时候为我们创建了什么。

（1）of\_get\_named\_gpio

/\*\*

 \* include/of\_gpio.h

 \* of\_get\_named\_gpio - 从设备树中提取gpio口

 \* @np - 设备节点指针

 \* @propname - 属性名

 \* @index - gpio口引脚标号

 \* 成功：得到GPIO口编号；失败：负数，绝对值是错误码

 \*/

int of\_get\_named\_gpio(struct device\_node \*np, const char \*propname, int index);

（2）gpio\_to\_irq

/\*\*

 \* include/gpio.h

 \* PIN值转换为相应的IRQ值,中断编号可以传给request\_irq()和free\_irq()

 \* @gpio - gpio口引脚标号

 \* 成功：得到GPIO口编号

 \*/

static inline int gpio\_to\_irq(unsigned gpio)

（3）devm\_request\_any\_context\_irq

/\*\*

 \* 注册中断

 \*/

devm\_request\_any\_context\_irq

（4）of\_match\_ptr

/\*\*

 \* 匹配设备树上的参数，将设备int\_demo\_dt\_ids与驱动int\_demo\_driver联系起来

 \* 系统会根据设备树种定义的compatible参数比较驱动中的int\_demo\_dt\_ids中定义的 .compatible 参数

 \*/

of\_match\_ptr(int\_demo\_dt\_ids)

例子：

static const struct of\_device\_id int\_demo\_dt\_ids[] = {

    { .compatible = "tiny4412,interrupt\_demo", },

    {},

};

MODULE\_DEVICE\_TABLE(of, int\_demo\_dt\_ids);

static struct platform\_driver int\_demo\_driver = {

    .driver        = {

        .name      = "interrupt\_demo",

        .of\_match\_table    = of\_match\_ptr(int\_demo\_dt\_ids),

    },

    .probe         = int\_demo\_probe,

    .remove        = int\_demo\_remove,

};

（5）of\_get\_property

/\*

 \*/drivers/of/base.c

 \* Find a property with a given name for a given node

 \* and return the value.

 \* 通过给定的设备节点和属性名字得到value。

 \*/

const void \*of\_get\_property(const struct device\_node \*np, const char \*name,

int \*lenp)

{

struct property \*pp = of\_find\_property(np, name, lenp);

return pp ? pp->value : NULL;

}

（6）devm\_pinctrl\_get

获取一个pinctrl句柄，参数是dev是包含这个pin的device结构体即xxx这个设备的device

获取设备操作句柄（设备模型中的struct device）的pin control state holder（struct pinctrl）

/\*\*

 \* struct devm\_pinctrl\_get() - Resource managed pinctrl\_get()

 \* @dev: the device to obtain the handle for

 \*

 \* If there is a need to explicitly destroy the returned struct pinctrl,

 \* devm\_pinctrl\_put() should be used, rather than plain pinctrl\_put().

 \*/

struct pinctrl \*devm\_pinctrl\_get(struct device \*dev)

（7）pinctrl\_lookup\_state

获取这个pin对应pin\_state（引脚状态-turnon\_tes/turnoff\_tes）

/\*\*

 \* pinctrl\_lookup\_state() - retrieves a state handle from a pinctrl handle

 \* @p: the pinctrl handle to retrieve the state from

 \* @name: the state name to retrieve

 \*/

struct pinctrl\_state \*pinctrl\_lookup\_state(struct pinctrl \*p, const char \*name)

（8）pinctrl\_select\_state

设置引脚为为某个stata -- turnon\_tes/turnoff\_tes

/\*\*

 \* pinctrl\_select\_state() - select/activate/program a pinctrl state to HW

 \* @p: the pinctrl handle for the device that requests configuration

 \* @state: the state handle to select/activate/program

 \*/

int pinctrl\_select\_state(struct pinctrl \*p, struct pinctrl\_state \*state)

（9）of\_get\_named\_gpio

得到GPIO的编号

./\*\*

 \* include/of\_gpio.h

 \* of\_get\_named\_gpio - 从设备树中提取gpio口

 \* @np - 设备节点指针

 \* @propname - 属性名

 \* @index - gpio口引脚标号

 \* 成功：得到GPIO口编号int型；失败：负数，绝对值是错误码

 \*/

int of\_get\_named\_gpio(struct device\_node \*np, const char \*propname, int index);

of\_get\_named\_gpio：此函数是解析设备树的函数，我们通过这个函数去解析设备树，tiny4412,int\_gpio1 = <&gpx3 2 GPIO\_ACTIVE\_HIGH>;

跟踪下去会发现这个函数掉用了list = of\_get\_property(np, "tiny4412,int\_gpio2", &size);设备树解析是创界了设备节点，现在通过这个函数去获取属性。

（10）devm\_gpio\_request\_one

获取一个GPIO并初始化属性

/\*\*

 \* devm\_gpio\_request\_one - request a single GPIO with initial setup

 \* @dev:   device to request for

 \* @gpio: the GPIO number

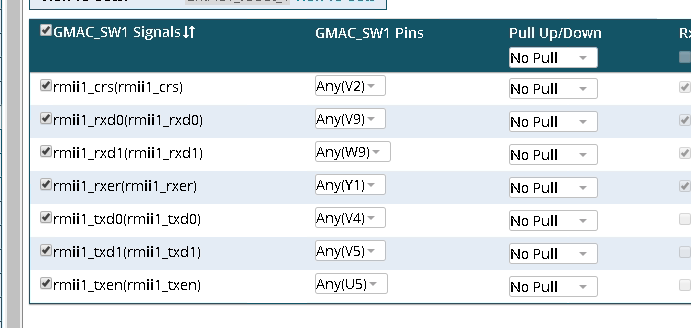
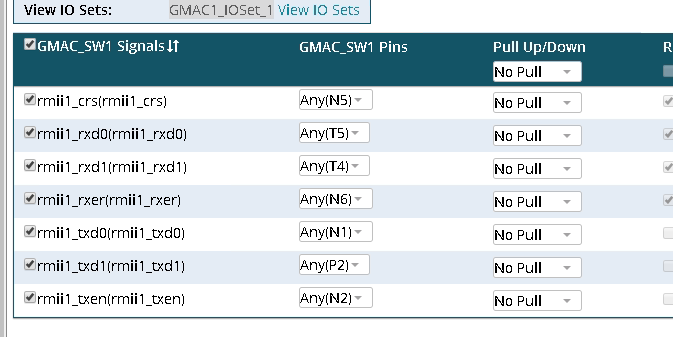
 \* @flags: GPIO configuration as specified by GPIOF\_\*

 \* @label: a literal description string of this GPIO

 \*/

int devm\_gpio\_request\_one(struct device \*dev, unsigned gpio,

 unsigned long flags, const char \*label)



修改uboot debug口：修改uboot设备树文件中的chosen节点

设备树memory设置

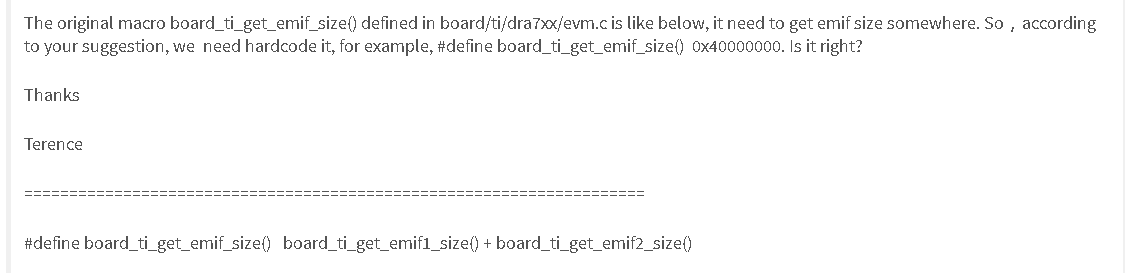
## 4.日常积累

函数名称前面加上\_\_weak 修饰符，我们一般称这个函数为“弱函数”。

加上了\_\_weak 修饰符的函数，用户可以在用户文件中重新定义一个同名函数，最终编译器编译的时候，会选择用户定义的函数，如果用户没有重新定义这个函数，那么编译器就会执行\_\_weak 声明的函数，并且编译器不会报错。所以我们可以在别的地方定义一个相同名字的函数，而不必也尽量不要修改之前的函数

usb虚拟串口，用的是USB2.0的话，理论上480 Mbps是USB2.0总线速度的上限.考虑通信协议的开销后，实际数据的传输速度存理论上最高也只有53 MB/s(426Mbps).实际综合条件下15 MB/s至25 MB/s都可以作为合理的高速目标。

uboot修改ram大小



Debugfs挂载

mount -t debugfs none /sys/kernel/debug/

消息队列：<https://www.jianshu.com/p/36a7775b04ec>

Linux Kernel 下的Mailbox驱动框架

（1）mailbox是一种框架，通过消息队列和中断驱动信号处理多处理器间的通讯；

（2）mailbox的实现分为contoller和client。简单的说就是client 可以通过controller提供的channel发送信息给controller；

（3）在drivers/mailbox下实现的都是controller的源码；

例: interrupts = <GIC\_SPI 110 IRQ\_TYPE\_LEVEL\_HIGH>;

1. GIC type // include\dt-bindings\Interrupt-controller\arm-gic.h

#define GIC\_SPI 0 // 共享中断

#define GIC\_PPI 1 // 每个处理器拥有独立中断

2. IRQ type // include\dt-bindings\interrupt-controller\irq.h

#define IRQ\_TYPE\_NONE  0内核不改变它，开机或uboot设置它是什么样就什么样。

#define IRQ\_TYPE\_EDGE\_RISING 1            上升沿触发

#define IRQ\_TYPE\_EDGE\_FALLING 2            下降沿

#define IRQ\_TYPE\_EDGE\_BOTH (IRQ\_TYPE\_EDGE\_FALLING | IRQ\_TYPE\_EDGE\_RISING)            双边沿

#define IRQ\_TYPE\_LEVEL\_HIGH 4             电平触发-高电平

#define IRQ\_TYPE\_LEVEL\_LOW 8              电平触发-低电平

IPI：inter-processer interrupt   中断号0～15

PPI：per processor interrupts    中断号16～31

SPI：shared processor interrupts  中断号 32 ～32+224

SGI：software generated interrupts (SGI).

I2c\_tools: <https://blog.csdn.net/mantis_1984/article/details/18254767/>