1. **U-boot**

AM335x 的启动主要包括ROM，SPL， U-Boot 和kernel四个启动步骤：

1）.ROM为ROM Code 是固化在芯片内部的代码，根据SYSBOOT 引脚的配置，确定启动模式，初始化相应的外设，读取下一阶段启动文件，即SPL 文件。

2）. SPL 由ROM Code 加载到片上SRAM 中，负责完成最小系统，用以硬件初始化：关闭看门狗，关闭中断，设置 CPU 时钟频率、速度等，在DDR 中加载并运行U-Boot 文件。MLO文件应该会被映射到 64 KB的 Internal SRAM 中。**MLO 由ROM Code 读取到片上 RAM中。**

3）. U-Boot 会在SPL 系统配置的基础上，进一步配置包括网口，USB，Nand Flash 等外设或存储设备，并从中读取和加载Kernel，具体在参考文献中有详细描述。

U-boot下载

这个工具的适用场景，不是什么工作也没做，就想直接启动板子进行烧录；而是在你完成了整个系统移植过程后，可以使用该工具来进行USB启动烧录。换言之：若要使用Uniflash，必须要先完成Linux部分的移植。下面会对这个原因进行详细的解释。

## SPL

在uboot-2016中uboot分为uboot-spl和uboot两个组成部分。SPL是Secondary Program Loader的简称，第二阶段程序加载器，这里所谓的第二阶段是相对于SOC中的BROM来说的，SOC启动最先执行的是BROM中的固化程序。

BROM会通过检测启动方式来加载第二阶段bootloader。uboot已经是一个bootloader了，那么为什么还多一个uboot spl呢？

这个主要原因是对于一些SOC来说，它的内部SRAM可能会比较小，小到无法装载下一个完整的uboot镜像，那么就需要spl，它主要负责初始化外部RAM和环境，并加载真正的uboot镜像到外部RAM中来执行。

所以由此来看，SPL应该是一个非常小的loader程序，可以运行于SOC的内部SRAM中，它的主要功能就是加载真正的uboot并运行之。

SPL Makefile依赖文件：

在u-boot-spl.lds中，空间分配如下：

MEMORY { .sram : ORIGIN = CONFIG\_SPL\_TEXT\_BASE,\

LENGTH = CONFIG\_SPL\_MAX\_SIZE }

MEMORY { .sdram : ORIGIN = CONFIG\_SPL\_BSS\_START\_ADDR, \

LENGTH = CONFIG\_SPL\_BSS\_MAX\_SIZE }

其中，CONFIG\_SPL\_TEXT\_BASE、CONFIG\_SPL\_MAX\_SIZE、CONFIG\_SPL\_BSS\_START\_ADDR、CONFIG\_SPL\_BSS\_MAX\_SIZE都是在Am335x\_evm.h定义：

|  |
| --- |
| #define CONFIG\_SPL\_TEXT\_BASE 0x402F0400  #define CONFIG\_SPL\_MAX\_SIZE (101 \* 1024)  #define CONFIG\_SPL\_BSS\_START\_ADDR 0x80000000  #define CONFIG\_SPL\_BSS\_MAX\_SIZE 0x80000 /\* 512 KB \*/ |

SRAM从0x402F0400开始用于存放.TEXT段、.rodata段、.data段内，SDRAM从0x80000000开始用于存放.bss段内容。

|  |
| --- |
| **SPL的功能无非是设置MPU的Clock、PLL，Power，DDR，Uart，Pin Mux**，完成对U-Boot的引导的工作，所以SPL board port主要针对以上几点：  **cpu\_init\_crit**  **board\_init\_f**  **board\_init\_r** |

其中：

|  |
| --- |
| **cpu\_init\_crit**  ----> lowlevel\_init (u-boot-2011.09-psp04.06.00.08\arch\arm\cpu\armv7\omap-common\lowlevel\_init.S)  ----> s\_init (u-boot-2011.09-psp04.06.00.08\board\ti\am335x\Evm.c)  ----> 关看门狗  ----> pll\_init(); //PLL和时钟设置  ----> rtc32k\_enable(); //使能RTC  ----> 串口设置  ----> init\_timer();  ----> preloader\_console\_init();  ----> I2C0初始化，读EEPROM中的配置信息，该部分移植时只需要将代码改为strcpy(header.name,"IRTK2");  ----> DDR设置(DDR2\DDR3) |

## Uboot 配置分析

在Makefile中主要使用配置项是：

**%config: scripts\_basic outputmakefile FORCE**

**$(Q)$(MAKE) $(build)=scripts/kconfig $@**

如：make CROSS\_COMPILE=arm-linux-gnueabihf- O=am335x\_evm am335x\_evm\_defconfig时

展开则为：am335x\_evm\_defconfig: **scripts\_basic outputmakefile FORCE**

$(Q)$(MAKE) $(build)=scripts/kconfig am335x\_evm\_defconfig

其中$(build)在kbuild.include中定义：

build := -f $(srctree)/scripts/Makefile.build obj

2.1 依赖：**scripts\_basic：**

PHONY += scripts\_basic

scripts\_basic:

$(Q)$(MAKE) **$(build)**=scripts/basic$(Q)

$(Q)rm -f .tmp\_quiet\_recordmcount

注：PHONY 目标并非实际的文件名：只是在显式请求时执行命令的名字。已知phony 目标并非是由其它文件生成的实际文件，make 会跳过隐含规则搜索。**.PHONY是一个伪目标，可以防止在Makefile中定义的只执行命令的目标和工作目录下的实际文件出现名字冲突。**

例如：　　.PHONY : clean

　　 clean :

　　 rm \*.o temp

展开后：scripts\_basic:

make**–f /scripts/Makefile.build obj** =scripts/basic

$(Q)rm -f .tmp\_quiet\_recordmcount

2.2依赖：**outputmakefile**

PHONY += **outputmakefile**

# outputmakefile generates a Makefile in the output directory, if using a

# separate output directory. This allows convenient use of make in the

# output directory.

**outputmakefile:**

ifneq ($(KBUILD\_SRC),)

$(Q)ln -fsn $(srctree) source

$(Q)$(CONFIG\_SHELL) $(srctree)/scripts/mkmakefile \

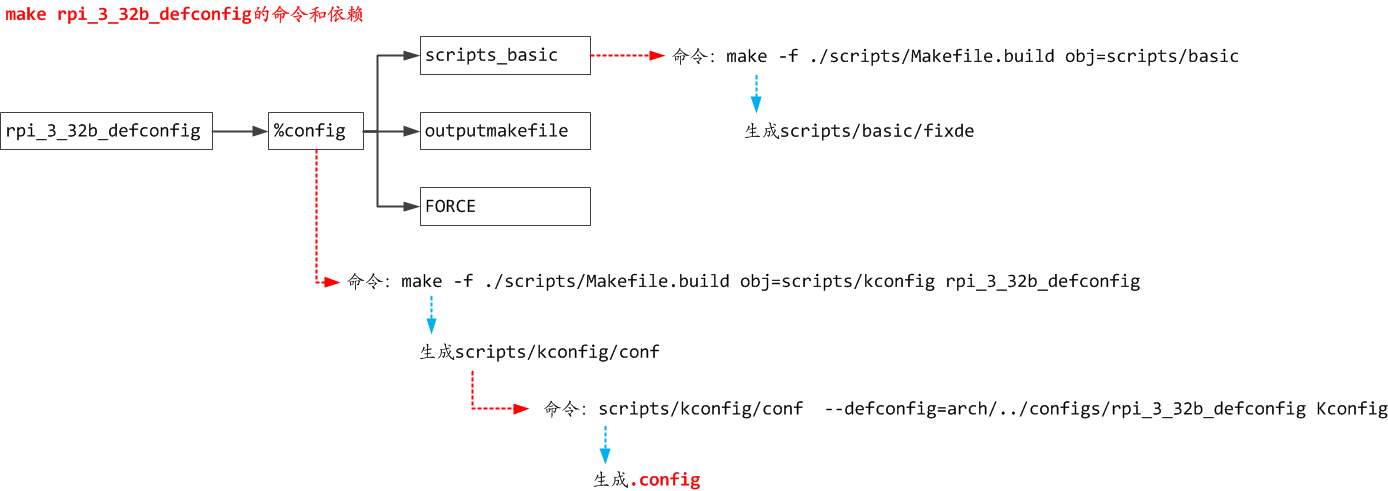
$(srctree) $(objtree) $(VERSION) $(PATCHLEVEL)

2.3依赖：**FORCE**

PHONY += **FORCE**

**FORCE**:

FORCE被定义为一个空目标。如果一个目标添加FORCE依赖，每次编译都会去先去执行FORCE（实际上什么都不做），然后运行命令更新目标，**这样就能确保目标每次都会被更新。**



## Uboot 编译分析

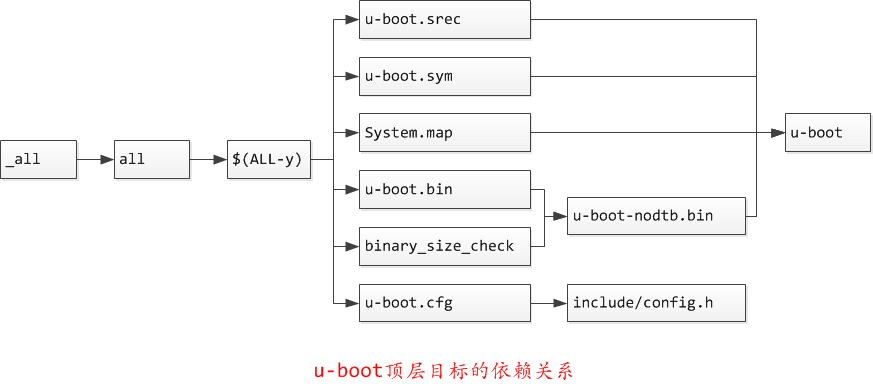
spl的编译是编译uboot的一部分，和uboot.bin走的是两条编译流程，这个要重点注意。 正常来说，会先编译主体uboot，也就是uboot.bin.再编译uboot-spl，也就是uboot-spl.bin,虽然编译命令是一起的，但是编译流程是分开的。

在上一节配置完Uboot之后需要编译uboot，下面开始分析Makefile

3.1顶层目录Makefile:

先寻找all和\_all目标依赖：

|  |
| --- |
| # That's our default target when none is given on the command line  PHONY := \_all  \_all: |

****

编译出的**SPL**生成可执行文件如下：

a).**u-boot-spl**：作为带有调试信息的image，由于大约有500K bytes，所以不能直接下载到片上SRAM（109K bytes），所以只能用于导入symbol。

b). **u-boot-spl.bin**：包含有调试信息但没有头信息，调试时可将其通过JTAG 下载到片上SRAM上。

c). **u-boot-spl.map**：是SPL 的内存映射信息，可以找到各函数的入口地址。

d). **MLO**：是SD卡中所存储的，用于启动的image。

编译出的**uboot**生成可执行文件如下：

a). **u-boot.img** 为第二阶段启动所使用的image。

b). **u-boot** 包含有调试信息，属于ELF 格式，是调试时需要的image。

c). **u-boot.map** 这个文件里面存储了U-Boot 的内存映射信息，可以找到各函数的入口地址。

## Uboot环境变量

4.1常用环境变量描述

|  |  |
| --- | --- |
| 环境变量 | 描述 |
| bootdelay | 执行自动启动的等候秒数 |
| baudrate | 串口控制台的波特率 |
| netmask | 以太网接口的掩码 |
| ethaddr | 以太网卡的网卡物理地址 |
| bootfile | 缺省的下载文件 |
| bootargs | 传递给内核的启动参数 |
| bootcmd | 自动启动时执行的命令 |
| serverip | 服务器端的ip地址 |
| ipaddr | 本地ip 地址 |
| stdin | 标准输入设备 |
| stdout | 标准输出设备 |
| stderr | 标准出错设备 |

其中：

**Bootcmd:**

bootcmd是自动启动时默认执行的一些命令，因此你可以在当前环境中定义各种不同配置，不同环境的参数设置，然后设置bootcmd为你经常使用的那种参数。

**Bootargs:**

bootargs是环境变量中的重中之重，甚至可以说整个环境变量都是围绕着bootargs来设置的。bootargs的种类非常的多，也非常的灵活，内核和文件系统的不同搭配就会有不同的设置方法；甚至你也可以不设置bootargs，而直接将其写到内核中去（在配置内核的选项中可以进行这样的设置），正是这些原因导致了bootargs使用上的困难。

* 1. bootargs 常用参数

1). root： 用来指定rootfs的位置

Ram 根文件系统：

Root=/dev/ram rw 或者 root=/dev/ram0 rw

2). rootfstype：指明文件系统的类型

这个选项需要跟root一起配合使用，一般如果根文件系统是ext2的话，有没有这个选项是无所谓的，但是如果是jffs2,squashfs等文件系统的话，就需要rootfstype指明文件系统的类型，不然会无法挂载根分区.

3). console：串口终端

4). mem ：内存大小

5). ramdisk\_size ：ramdisk尺寸

6). initrd, noinitrd：是否是ramdisk启动

7). init：启动后运行的第一个脚本

8). mtdparts：分区结构

9). ip：nfs使用的时候必须有

常见组合：

1). 假设文件系统是ramdisk，且直接就在内存中，bootargs的设置应该如下：

setenv bootargs 'initrd=0x32000000,0xa00000 root=/dev/ram0 console=ttySAC0 mem=64M init=/linuxrc'

2). 假设文件系统是ramdisk，且在flash中，bootargs的设置应该如下：

setenv bootargs 'mem=32M console=ttyS0,115200 root=/dev/ram rw init=/linuxrc'

注意这种情况下你应该要在bootm命令中指定ramdisk在flash中的地址，如bootm kernel\_addr ramdisk\_addr (fdt\_addr)

1. **git 基础学习**

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理起来速度飞快。

1. **工作流程**

首先需要说明的是，git的版本库是以分支的形式进行管理的。当我们初始化一个新的版本库时，会自动创建版本库的第一个分支master，我们对本地仓库的修改都是提交到这个master分支的。

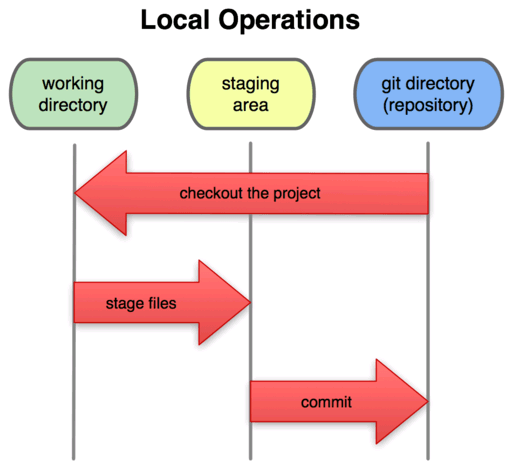
a).在工作目录中修改某些文件。

b).对修改后的文件进行快照，然后保存到暂存区域。

c).提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。

1. **文件的三种状态**

在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。



1. **Git 使用**

3.1 配置环境，全局变量

初次运行git需要配置一些基本信息，使用 git config 来配置或读取工作环境边变量。

$ git config --global user.name "Lu Yingjian"

$ git config --global user.email Luyj@example.com

如果用了 --global 选项，以后所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者邮箱，只要去掉 --global 选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

* 1. git命令

a).git init //初始化新仓库

b).git add \*.c(增加到仓库的文件) //增加文件到仓库

c).git commit //

d).git status //检查文件状态

e). git clone //从现有仓库进行克隆

f). git diff //对比文件

g).git rm (file) //移除文件（从暂存区移除）

h).git mv file\_form file\_to //移动文件

一般使用git只需要

* git add .
* git commit -m "提交注释"
* git push origin  分支名称
* git reset --hard HEAD^   //版本回退
* git reset --hard [版本号] //返回固定版本号
* git commit ammend //用于修改已经提交的注释
  1. 忽略某些文件

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。

文件 .gitignore 的格式规范如下：

a).所有空行或者以注释符号 ＃ 开头的行都会被 Git 忽略。

b).可以使用标准的 glob 模式匹配。

c).匹配模式最后跟反斜杠（/）说明要忽略的是目录。

d).要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

# 此为注释 – 将被 Git 忽略

# 忽略所有 .o或.a 结尾的文件

\*.[oa]

# 但 lib.a 除外

!lib.a

# 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

/TODO

# 忽略 build/ 目录下的所有文件

build/

# 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

doc/\*.txt

# 忽略 doc/ 目录下所有扩展名为 txt 的文件

doc/\*\*/\*.txt

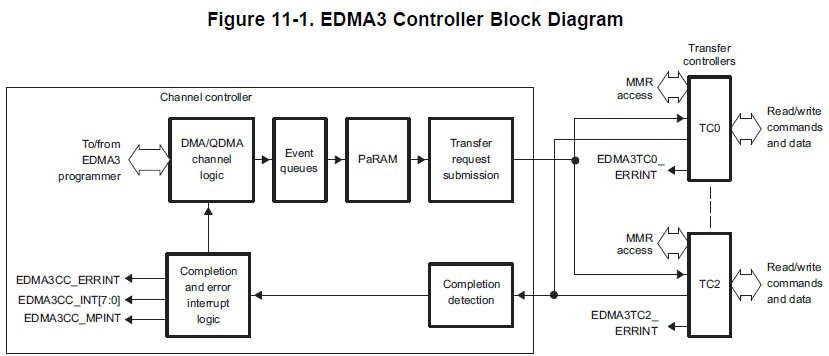
1. **EDMA学习**
2. **EDMA简介**

EDMA（Enhanced Direct Memory Access）是增强型的DMA控制器，用于控制存储器与存储器、存储器与外设、外设与外设之间的数据传输。其结构主要包含TPCC（Third-Party Channel Controller）与TPTC（Third-Party Transfer Controller）两大部分，即通道控制器与传输控制器，原文如下：

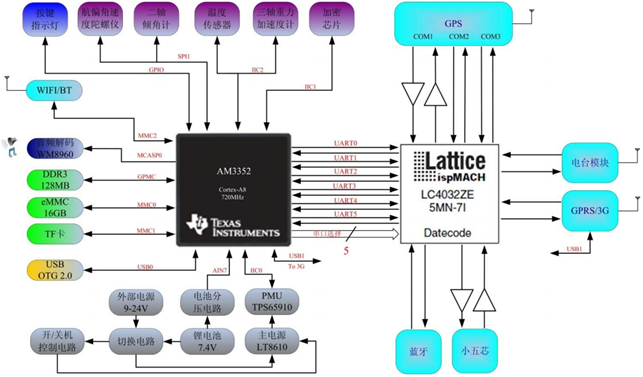
The EDMA3 controller consists of two principal blocks:

• EDMA3 channel controller (EDMA3CC).

• EDMA3 transfer controller(s) (EDMA3TC).



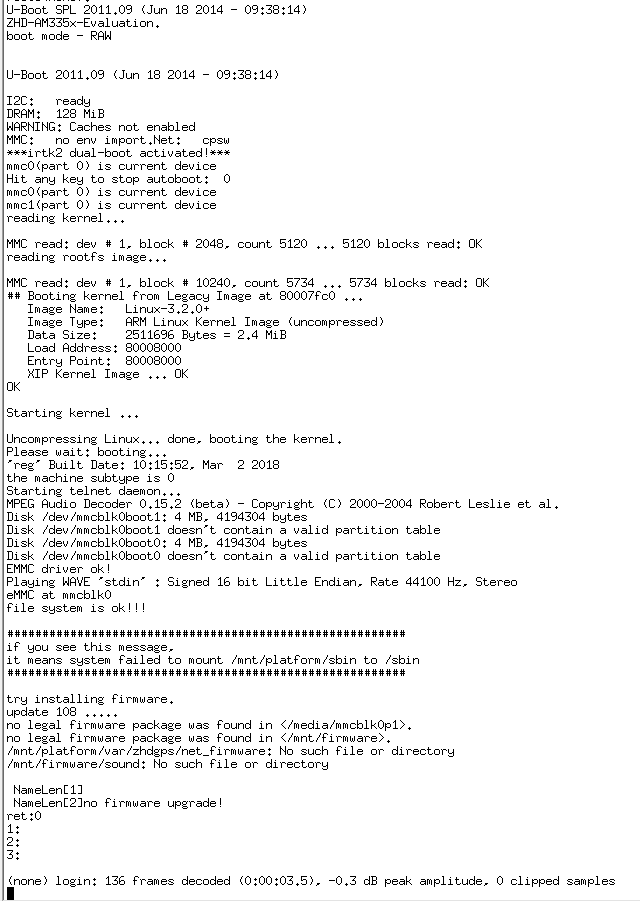
1. **IRTK2**
2. **IRTK2原始信息记录**
   1. 硬件相关
      1. emmc型号为THGBM5G6A2JBAIR
      2. 语音芯片为 WM8960 GEFL
      3. GPS板卡为BD970
      4. WIFI模块为 WG7311-EA



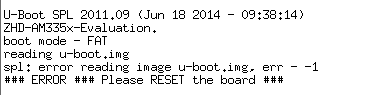
* 1. 软件相关
     1. uboot 为 2011.09



1.2.2 linux内核为 linux-3.2.0+



MLO单独启动：



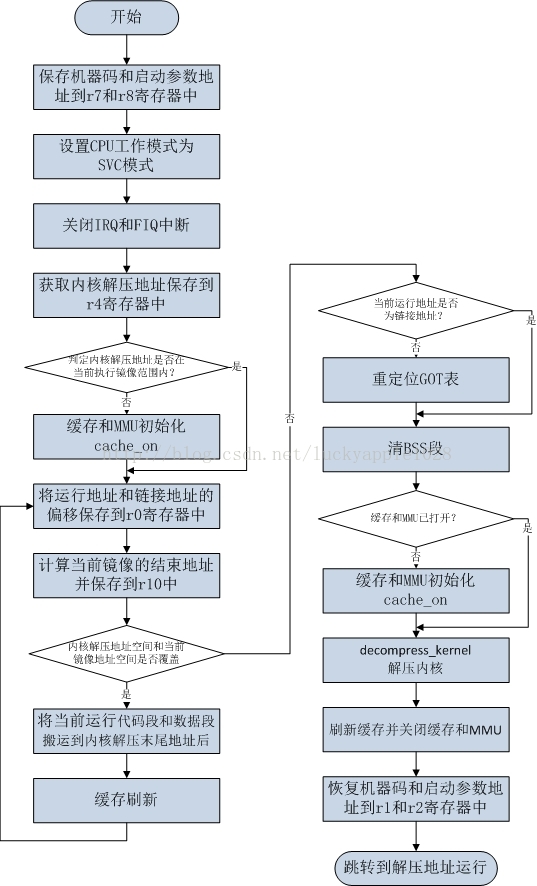
1. **登录模式**

分为3个登陆状态：uboot模式，安全模式，用户模式。

Uboot模式只有在插入带系统文件的SD卡才有这一项。

1. **发**
2. **内核流程分析**
3. 内核自解压

内核自解压程序的入口：参见arch/arm/boot/compressed/vmlinux.lds



1. **Wifi/bluetooth**
2. **硬件基础**

iRTK2:Wifi/Bluetooth 型号为WG7311-EA（WL1271L）

Specificaiton:

---WG7311, **a WiFi , Bluetooth and FM** SiP (system in package) module

---WLAN, Bluetooth and FM radio on a SiP module

---LGA96 pin package

---Dimension 10mm(L) x 10mm(W) x 1.4mm(H)

---**Based on TI WL1271L** 65-nm CMOS technology packaged in WSP for module

---Seamless integration with TI OMAP application processor and GSM-GPRS-UMTS chipset

---Internal support for WLAN and Bluetooth Co-existence (bandwidth sharing, antenna sharing)

---Direct connection to battery using external switching mode power supply supporting 5.5V to 2.3V operation

---VIO in the 1.8V domain

硬件连接：

WIFI: SDIO ----> MMC2

MMC2\_CMD ----> GPMC\_CSn3[MODE3]

MMC2\_CLK ----> GPMC\_CLK[MODE3]

WL\_EN ----> GPMC\_AD9

WL\_IRQ ----> GPMC\_AD11

SLOWCLK ----> XDMA\_EVENT\_INTR1

BLUETOOTH: BT\_EN ----> GPMC\_AD10

BT\_TXD ----> 连接到LC4032ZE-7TN48I

BT\_RXD ----> 连接到LC4032ZE-7TN48I

ispMACH LC4032ZE-7TN48I（？？？）

CPLD？？？

TI官方:Wifi/Bluetooth 型号为WL1271

WIFI: SLEEP\_CLK ----> XDMA\_EVENT\_INTR1 （同SLOWCLK）

WL\_EN ----> GPMC\_CSn0.gpio1\_29

WLAN\_IRQ ----> GPMC\_WPn.gpio0\_31 (设备树中配置0\_31中断)

SDIO ----> MMC1 (修改设备树为MMC2)

需要配置电压引脚

SN74AVC4T245配置电压

Am335x调节电压引脚配置：

设备树中

clkout2\_pin: pinmux\_clkout2\_pin { //配置XDMA\_EVENT\_INTR1,配置中断

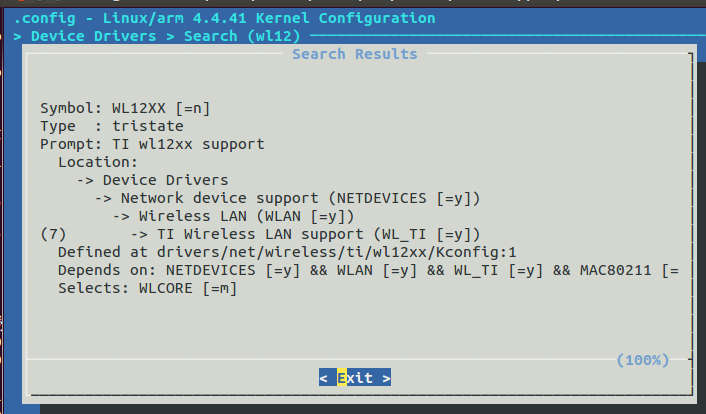
wl12xx\_gpio: pinmux\_wl12xx\_gpio //wlan\_en引脚

wl12xx\_vmmc: fixedregulator@2 {

**compatible = "regulator-fixed" //**wlan\_en**调节引脚电压**

**}**

wifi模块正常工作WLAN ENABLE GPIO必须高， WLAN IRQ GPIO 应为低。



1. **SDIO接口wifi驱动**

SDIO-Wifi模块是基于SDIO接口的符合wifi无线网络标准的嵌入式模块，内置无线网络协议**IEEE802.11协议栈**以及**TCP/IP协议栈**，能够实现用户主平台数据通过SDIO口到无线网络之间的转换。SDIO具有传输数据快，兼容SD、MMC接口等特点。

对于**SDIO接口的wifi**，首先，**它是一个sdio的卡的设备，然后具备了wifi的功能**，所以，注册的时候还是**先以sdio的卡的设备进行注册**。然后检测到卡之后就要**驱动他的wifi功能了**，显然，他是**用sdio的协议**，**通过发命令和数据来控制的**。

**SDIO总线上都是HOST端发起请求，然后DEVICE端回应请求。**sdio命令**由6个字节**组成。

(1).**Command:**用于开始传输的命令，是由HOST端发往DEVICE端的。其中命令是通过CMD信号线传送的。

(2).**Response:**回应是DEVICE返回的HOST的命令，作为Command的回应。也是通过CMD线传送的。

(3).**Data:**数据是双向的传送的。可以设置为1线模式，也可以设置为4线模式。数据是通过DAT0-DAT3信号线传输的。

**SDIO读写命令流程：**

(1).**读命令**，首先HOST会向DEVICE发送命令，紧接着DEVICE会返回一个握手信号，此时，当HOST收到回应的握手信号后，会将数据放在4位的数据线上，在传送数据的同时会跟随着CRC校验码。当整个读传送完毕后，HOST会再次发送一个命令，通知DEVICE操作完毕，DEVICE同时会返回一个响应。

(2).**写命令**，首先HOST会向DEVICE发送命令，紧接着DEVICE会返回一个握手信号，此时，当HOST收到回应的握手信号后，会将数据放在4位的数据线上，在传送数据的同时会跟随着CRC校验码。当整个写传送完毕后，HOST会再次发送一个命令，通知DEVICE操作完毕，DEVICE同时会返回一个响应。

1. **Hostapd**

hostapd 是一个用户态用于AP和认证服务器的守护进程。

Hostapd的功能就是作为AP的认证服务器，负责控制管理stations(通常可以认为带无线网卡的PC)的接入和认证。**通过Hostapd可以将无线网卡切换为AP/Master模式，通过修改配置文件，**可以建立一个开放式的(不加密)的，WEP，WPA或WPA2的无线网络。并且通过修改配置文件可以设置无线网卡的各种参数，包括频率，信号，beacon包时间间隔，是否发送beacon包，如果响应探针请求等等。还可以设置mac地址过滤条件等。

它**实现了IEEE 802.11相关的接入管理**，IEEE 802.1X/WPA/WPA2/EAP 认证, RADIUS客户端，EAP服务器和RADIUS 认证服务器。Linux下支持的驱动有：Host AP，madwifi，**基于mac80211的驱动。**

* 1. Hostapd常用术语

(1).MLME：

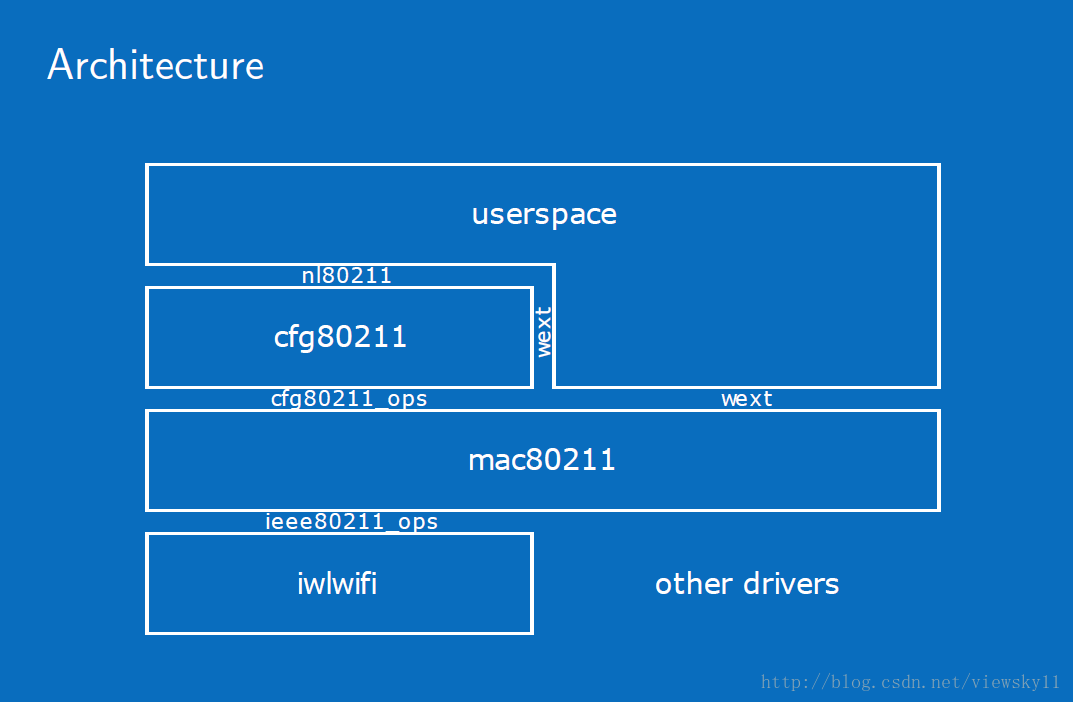
即**MAC(Media Access Control ) Layer Management Entity**，它管理物理层MAC状态机。

(2).mac8021：

mac80211是**linux kernel中的一个子系统**，它为无线设备soft-MAC/half-MAC提供了分享实施方案，包含MLME和另外一些代码。

它是一个driver开发者**可用于为SoftMAC无线设备写驱动的框架**，mac80211**为SoftMAC**设备**实现了cfg80211回调函数**，且mac80211通过cfg80211实现了向网络子系统注册和配置。配置由cfg80211通过nl80211和wext实现。

mac80211的架构如下图所示：



(3).cfg80211

用于对无线设备进行配置管理。与FullMAC, mac80211和nl80211一起工作。(Kernel态)

**cfg80211是Linux 802.11配置API。**cfg80211用于代码wext(Wireless-Extensions)，**nl80211用于配置一个cfg80211设备**，且用于kernel与userspace间的通信。wext现处理维护状态，没有新的功能被增加，只是修改bug。如果需要通过wext操作，则需要定义CONFIG\_CFG80211\_WEXT。

wext: 基于ioctl机制

(4).nl80211

**用于对无线设备进行配置管理**，它是一个基本Netlink的用户态协议(User态)

(5).SoftMAC

其MLME由软件实现，**mac80211为SoftMAC实现提供了一个driver API**。 即：SoftMAC设备允许对硬件执行更好地控制，允许用软件实现对802.11的帧管理，包括解析和产生802.11无线帧。目前大多数802.11设备为SoftMAC,而FullMAC设备较少。

(6).FullMAC

其MLME由硬件管理，当写FullMAC无线驱动时，不需要使用mac80211。

(7).wpa\_supplicant

是用户空间一个应用程序，主要发起MLME命令，然后处理相关结果。

* 1. hostapd配置

(1).基本配置例如：

|  |
| --- |
| **ssid**=mytest  **hw\_mod**e=g  channel=10  **interface**=wlan0  bridge=br0  **driver**=nl80211  ignore\_broadcast\_ssid=0  **macaddr\_acl**=0  accept\_mac\_file=/etc/hostapd.accept  deny\_mac\_file=/etc/hostapd.deny |

* **ssid**：别人所看到的我们这个无线接入点的名称；
* **hw\_mode**：指定802.11协议，包括 a = IEEE 802.11a, b = IEEE 802.11b, g = IEEE 802.11g；
* channel：设定无线频道；
* **interface**：接入点设备名称，注意不要包含ap后缀，即如果该设备称为wlan0ap，填写wlan0即可；
* bridge：指定所处网桥，对于一个同时接入公网、提供内部网和无线接入的路由器来说，设定网桥很有必要；
* **driver**：设定无线驱动，我这里是nl80211；
* **macaddr\_acl**：可选，指定MAC地址过滤规则，0表示除非在禁止列表否则允许，1表示除非在允许列表否则禁止，2表示使用外部RADIUS服务器；
* accept\_mac\_file：指定允许MAC列表文件所在；

deny\_mac\_file：指定禁止MAC列表文件所在；

(2.)认证方式的配置：

**auth\_algs指定采用哪种认证算法**，采用**位域**（bit fields）方式来制定。

其中bit 0表示开放系统认证（Open System Authentication, OSA），bit 1 表示共享密钥认证（Shared Key Authentication, SKA）。例如设置alth\_algs=1，表示只采用OSA；如果设置为3则两种认证方式都支持。

加密有wpa、wep、wap/wpa2等加密方式

下面为wpa加密（wpa已被破解基本不使用，了解就好）

|  |
| --- |
| #wep\_default\_key=0  #wep\_key0=1234567890  #wep\_key1="vwxyz"  #wep\_key2=0102030405060708090a0b0c0d  #wep\_key3=".2.4.6.8.0.23"  #wep\_key\_len\_broadcast=13  #wep\_key\_len\_unicast=13  #wep\_rekey\_period=300 |

如果要**启用WEP加密**，只需**配置wep\_default\_key和其中一个wep\_keyx**，其中x的值只能在0~3之间，**wep\_default\_key的值必须启用**。注意**wep\_keyx的值不是任意的，只能是5、13或16个字符（用双引号括住）**，或者是10、26或32个16进制数字。由于WEP加密算法已经被破解了，所以通常不启用它，全部注释掉。

使用wap/wpa2加密方式（推荐使用）：

|  |
| --- |
| wpa=2  wpa\_passphrase=12345678  wpa\_key\_mgmt=WPA-PSK  #wpa\_pairwise=TKIP CCMP  rsn\_pairwise=CCMP |

* **wpa**：指定WPA类型，这是一个位域值（bit fields），bit 0表示启用WPA，bit 1表示启用WPA2。在我的配置中，置成2 可以正常连接；
* **wpa\_passphrase**：WPA/WPA2加密需要指定密钥，这个选项就是配置WPA/WPA2的密钥。**注意wpa\_passphrase要求8~63个字符**。另外还可以通过配置**wpa\_psk**来制定密钥，不过要设置**一个256位的16进制密钥**，不适合我们的需求；
* wpa\_pairwise/**rsn\_pairwise**：如果启用了WPA，需要指定wpa\_pairwise；如果启用了WPA2，需要指定rsn\_pairwise，或者采用wpa\_pairwise的设定。
  1. DHCP

DHCP（Dynamic Host Configuration Protocol，**动态主机配置协议**）是一个**局域网的网络协议**，**使用UDP协议工作**。

**主要作用是集中的管理、分配IP地址**，使网络环境中的主机动态的获得IP地址、Gateway地址、DNS服务器地址等信息，并能够提升地址的使用率。

DHCP协议采用客户端/服务器模型，**主机地址的动态分配任务由网络主机驱动**。当DHCP服务器接收到来自网络主机申请地址的信息时，才会向网络主机发送相关的地址配置等信息，以实现网络主机地址信息的动态配置。DHCP具有以下功能：

1. 保证任何IP地址在同一时刻**只能由一台DHCP客户机**所使用。

2. DHCP应当可以给用户**分配永久固定的IP地址**。

3. DHCP应当可以同用其他方法获得IP地址的主机共存（如手工配置IP地址的主机）。

4. DHCP服务器应当向现有的BOOTP客户端提供服务。

DHCP有**三种机制分配IP**地址：

1. 自动分配方式（Automatic Allocation）
2. 动态分配方式（Dynamic Allocation）
3. 手工分配方式（Manual Allocation）

三种地址分配方式中，**只有动态分配可以重复使用客户端不再需要的地址**。

DHCP消息的格式是基于**BOOTP（Bootstrap Protocol）消息格式**的，这就要求设备具有BOOTP中继代理的功能，并能够与BOOTP客户端和DHCP服务器实现交互。

DHCP客户端:

在支持DHCP功能的网络设备上将指定的端口作为**DHCP Client**，通过**DHCP协议从DHCP Server动态获取IP地址等信息**，来实现设备的集中管理。一般应用于网络设备的网络管理接口上。

DHCP服务器:

DHCP服务器指的是由服务器控制一段IP地址范围，客户端登录服务器时就可以自动获得服务器分配的IP地址和子网掩码。

* 1. udhcpc

直接使用udhcpc只能分配ip地址，但是没有写入到设备中，即这个ip地址并没有生效。因为udhcpc**需要一个默认的配置文件default.script**，查看这个配置文件，它实际的作用就是**将分配到的ip地址**通过ifconfig命令**写入设备中**。

* 1. DBUS数据总线

**DBUS，数据总线，是一个低延迟，低开销，高可用性的ipc机制。**

**D-Bus是一个消息总线系统**，其功能已涵盖进程间通信的所有需求，并具备一些特殊的用途。D-Bus是三层架构的进程间通信系统，其中包括：

**1).接口层**：接口层由函数库libdbus提供，进程可通过该库使用D-Bus的能力。

**2).总线层**：总线层实际上是由D-Bus总线守护进程提供的。它在Linux系统启动时运行，负责进程间的消息路由和传递，其中包括Linux内核和Linux桌面环境的消息传递。

**3).包装层**：包装层一系列基于特定应用程序框架的Wrapper库。

1. **公司wifi信息**

init.d/enwlan:3:insmod /lib/modules/3.18.0-linux4sam\_5.0-alpha7+/kernel/pck\_wifi.ko

rcS.d/S99rc.local:42:/etc/wifi start &

udev/scripts/network.sh:9:# We get two "add" events for hostap cards due to wifi0

udev/scripts/network.sh:10:echo "$INTERFACE" | grep -q wifi && exit 0

wifi:8: echo 1 > /sys/devices/platform/leds-gpio/leds/wifi\_pwr/brightness

wifi:22: echo 0 > /sys/devices/platform/leds-gpio/leds/wifi\_pwr/brightness

xinetd.d/wifi:8: echo 1 > /sys/devices/platform/leds-gpio/leds/wifi\_pwr/brightness

xinetd.d/wifi:22: echo 0 > /sys/devices/platform/leds-gpio/leds/wifi\_pwr/brightness

xinetd.d/wpa\_supplicant.conf:13: ssid="ZHD-WIFI-B"

1. **LINUX命令及工具**
2. modprobe
3. lrz工具
4. stty

stty -F /dev/ttyS0 –a //用于查看串口所有参数

//-F, --file=设备 打开并使用指定设备代替标准输入

stty -F /dev/ttyS0 ispeed 115200 ospeed 115200 cs8 //设置波特率和数据位

// ispeed N 设置输入速度为N 波特

// ospeed N 设置输出速度为N 波特

// csN 设置字符大小为N 位，N 的范围为5 到8

//将串口1（/dev/ttyS0）设置成115200波特率，8位数据模式。

1. exec，source，fork

在运行shell脚本时候，有三种方式来调用外部的脚本，exec（exec script.sh）、source（source script.sh）、fork（./script.sh）

* 1. exec

使用exec来调用脚本，**被执行的脚本会继承当前shell的环境变量**。但事实上**exec产生了新的进程**，他会**把主shell的进程资源占用并替换脚本内容，继****承了原主shell的PID号，即原主shell剩下的内容不会执行**。

* 1. source

使用source或者“.”来调用外部脚本，**不会产生新的进程**，**继承当前shell环境变量**，而且**被调用的脚本运行结束后，它拥有的环境变量和声明变量会被当前shell保留**，类似将调用脚本的内容复制过来直接执行。**执行完毕后原主shell继续运行**。

* 1. fork

直接运行脚本，**会以当前shell为父进程，产生新的进程**，并且**继承主脚本的环境变量和声明变量**。执行完毕后，**主脚本不会保留其环境变量和声明变量**。

1. diff