华清远见 FARSIGHT 嵌入式培训专家

嵌入式学院 华清远见旗下品牌

Android 无线应用之Wifi

易松华

2010.10.16

华清远见 FARSIGHT 嵌入式培训专家

主要内容

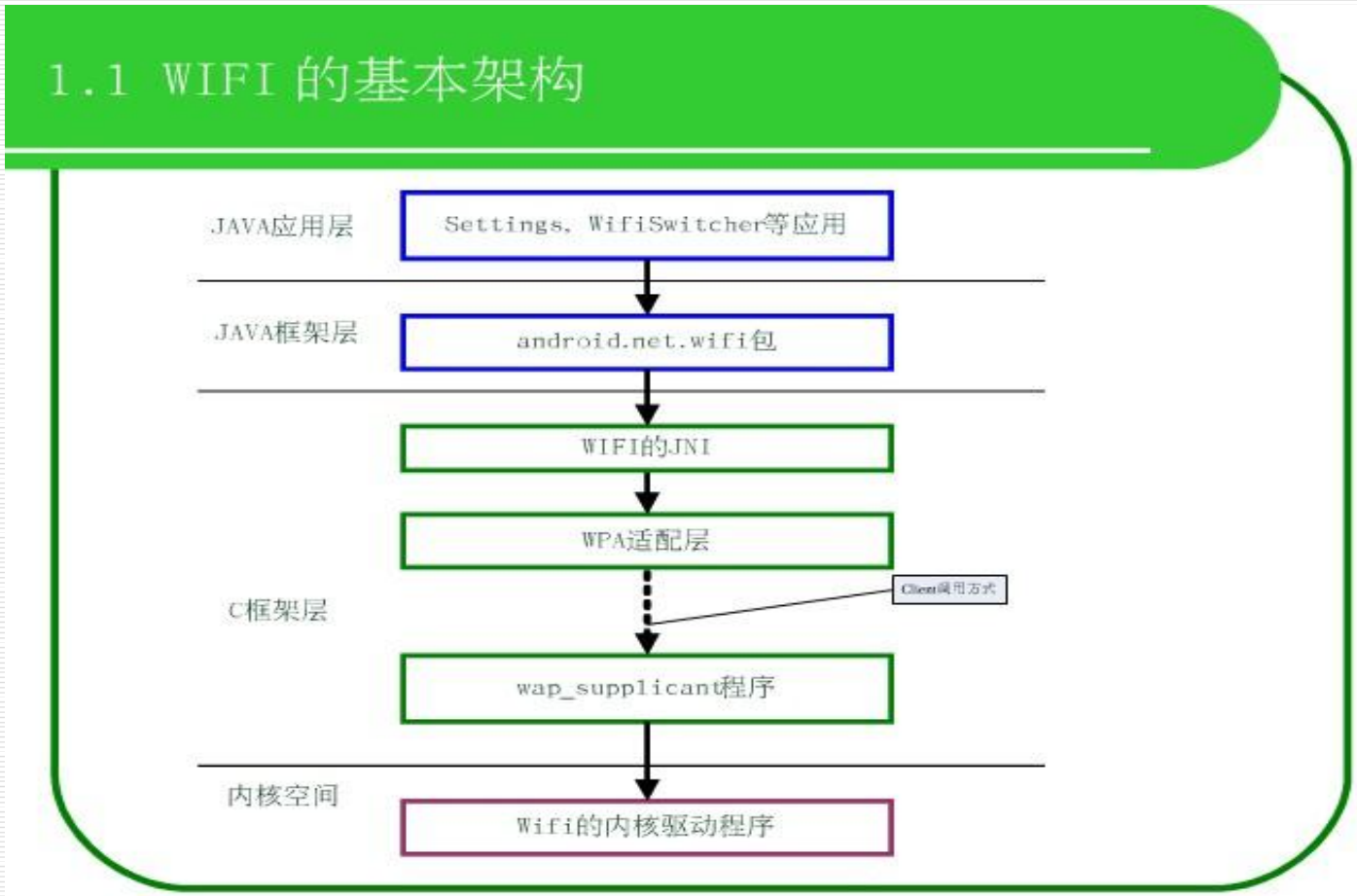
- **WIFI**基本工作原理和**android WIFI**基本架构
- **Marvell8686 SDIO WIFI**基本工作原理
- **SDIO** 内核驱动流程

Wifi运作原理：

- Wi-Fi 的设置至少需要一个Access Point（ap）和一个或一个以上的client（hi）。AP每100ms将SSID（Service Set Identifier）经由beacons（信号台）封包广播一次，beacons封包的传输速率是1 Mbit/s，并且长度相当的短，所以这个广播动作对网络效能的影响不大。因为Wi-Fi规定的最低传输速率是1 Mbit/s，所以确保所有的Wi-Fi client端都能收到这个SSID广播封包，client 可以借此决定是否要和这一个SSID的AP连线。使用者可以设定要连线到哪一个SSID。

华清远见 FARSIGHT 嵌入式培训专家

WIFI的基本框架



硬件特性

- 88w8686特性
- 单芯片集成了 802.11 a/g/b RF、基带、CPU（arm9）、MAC、内存、接口
- IEEE 802.11 数据传输率1和2Mbps
- IEEE 802.11 b 数据传输率5.5和11Mbps
- IEEE 802.11 g 数据传输率6，9，12，18，24，36，48，54Mbpss
- 两个独立的DMA
- 内部工作频率5.5、11、16、20、40、64、80、128MHZ；外部睡眠时钟：100khz

硬件特性

- **Omap3530 sd/mmc/sdio接口特性**
  - **※ 支持SD2.0 、MMC4.2 、SDIO1.1**
  - **※ 1024byte 数据FIFO Tx/Rx**
  - **※ 支持Tx和Rx 2通道 DMA传输模式**
  - **※ 支持SDIO卡中断，挂起，恢复**
  - **※ 支持1bit、4bit sdio传输模式**

华清远见 FARSIGHT 嵌入式培训专家

SDIO接口

- **SDIO Card简介：**
  - 是为高速数据 **I/O**传输，低功耗移动电子设备而设计的。其电压范围为 **2.0~3.6V**。
- **SDIO协议：**
  - **SDIO**协议是由**SD**卡的协议演化升级而来的，很多地方保留了**SD**卡的读写协议，同时**SDIO**协议又在**SD**卡协议之上添加了**CMD52**和**CMD53**命令。
  - 协议规范定义了两种类型的 **SDIO Card**，即高速 **SDIO Card** 和低速 **SDIO Card**。
  - **SDIO**和**SD**卡规范间的一个重要区别是增加了低速标准
  - 这两个命令可以方便的访问某个功能的某个地址寄存器。

SDIO命令

- **SDIO命令：**
  - **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。
  - **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。
  - **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。
  - **CMD52** 命令是 **IO_RW_DIRECT** 命令的简称，由 **HOST** 发往 **DEVICE** 的，它必须有 **DEVICE** 返回来的 **Response** ，不需要占用 **DAT** 线，读写的数据是通过 **CMD52** 或者 **Response** 来传送。每次 只能读或者写一个 **byte**。
  - **CMD53** 是在**CDM52**上对读写进行了扩展，**CMD53** 允许每次读写多个字节或者多个块 **(BLOCK)** 。当读写操作是块操作的时候，块的大小是可以通过设置 **FBR** 中的相关寄存器来设置。

华清远见 FARSIGHT 嵌入式培训专家

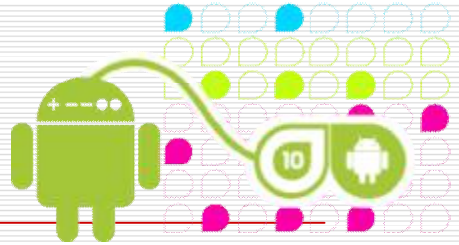
SDIO命令列表

Table 42: SDIO Mode, SDIO Commands

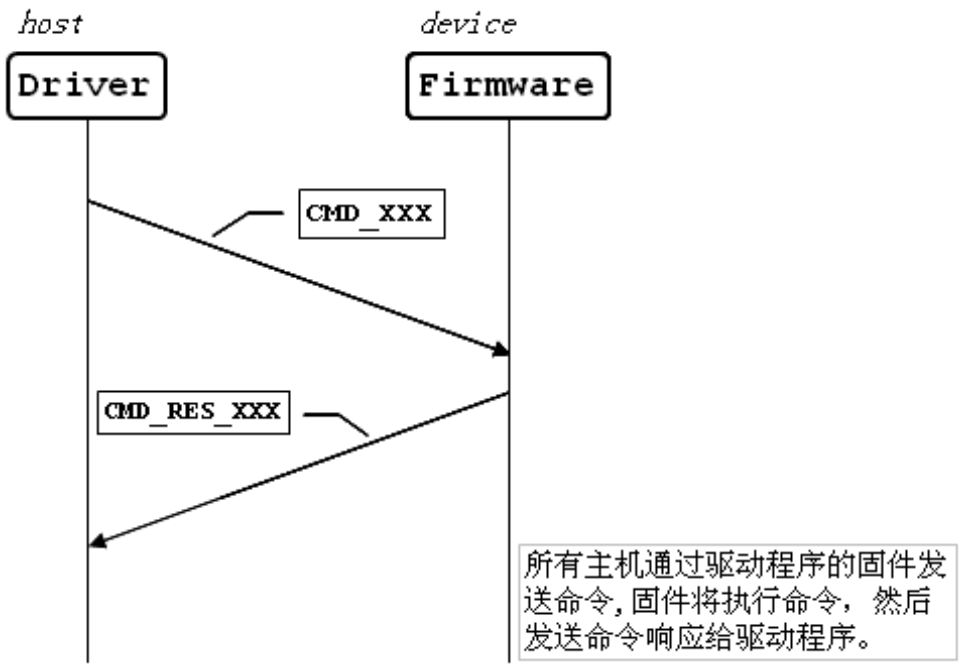
Command	Command Name	Description
CMD0	GO_IDLE_STATE	Used to change from SDIO to SPI mode
CMD3	SEND_RELATIVE_ADDR	SDIO Host asks for RCA
CMD5	IO_SEND_OP_COND	SDIO Host asks for and sets operation voltage
CMD7	SELECT/DESELECT_CARD	Sets SDIO target device to command state or back to standby
CMD15	GO_INACTIVE_STATE	Sets SDIO target device to inactive state
CMD52	IO_RW_DIRECT	Used to read/write host register and CIS table
CMD53	IO_RW_EXTENDED	Used to read/write data from/to SQU memory

- **CMD0**：**SDIO**模式和**SPI**模式的转换
- **CMD3**：读取**SDIO**的**RCA(**相关卡地址**)**
- **CMD5**：询问和设置卡的电压范围
- **CMD7**：使卡进入命令状态或者是准备好状态
- **CMD15**：设置卡为不激活状态
- **CMD52**：读写寄存器和**CIS(**卡信息结构**)**表
- **CMD53**：对多字节数据进行读写

SDIO读写操作



- **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。
- **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。



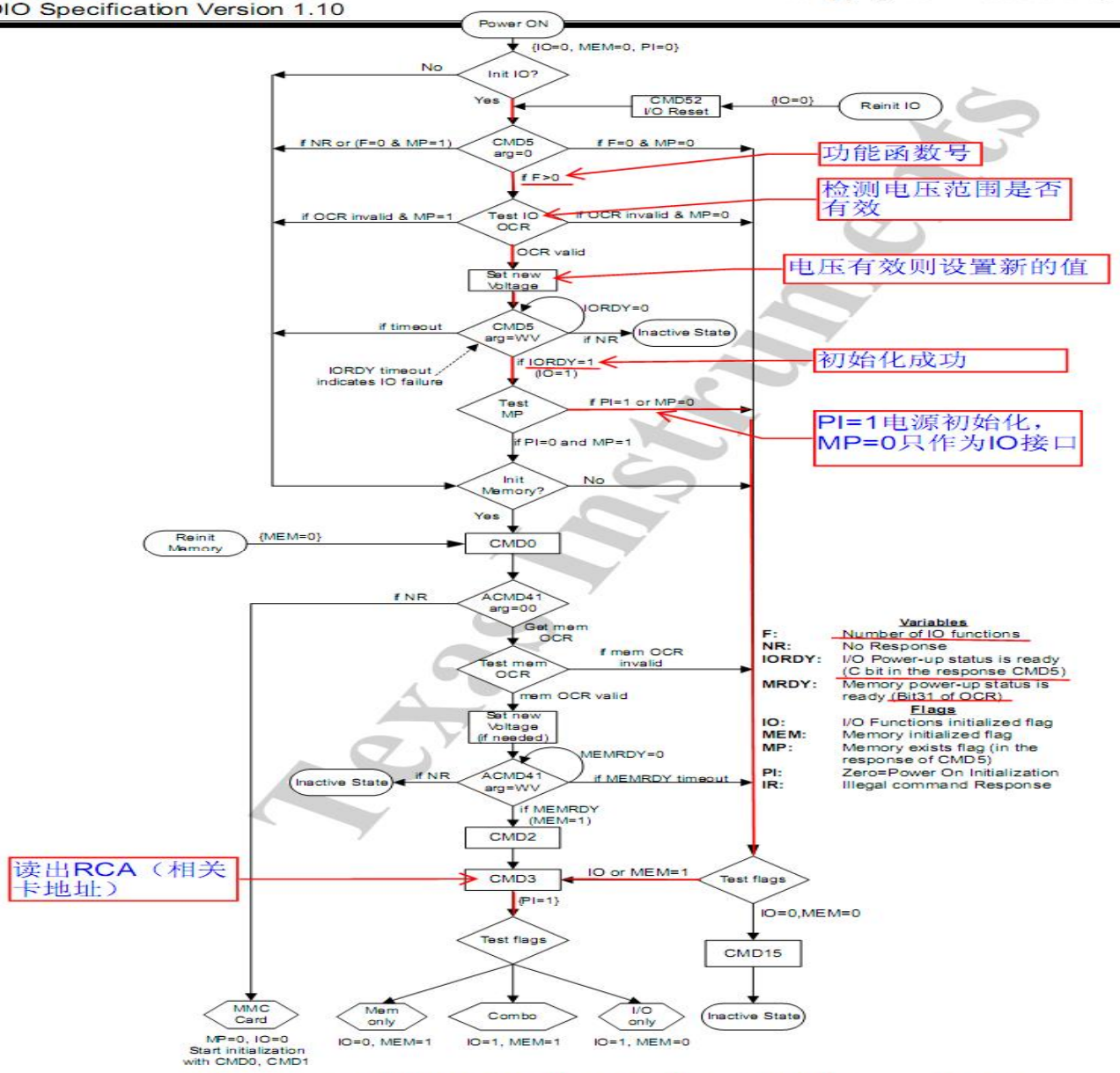


Figure 3 Card initialization flow in SD mode (SDIO aware host)

Android WIFI工作流程

- 进入设置菜单打开**WIFI**
- **wifi**模块上电，复位
- **SDIO rescan** 识别出**SDIO**卡
- **insmod** 驱动模块
- 驱动注册，调用**wlan_probe**
- **download firmware**到模块，注册网卡，初始化数据结构和回调函数
- 设置网卡参数
- 扫描网络
- 连接到**ap**
- 获取**dhcp**

Wifi 内核驱动支持

- 内核编译：

  定制内核模块：

  1.Networking support à Wireless à <*>Common routines for IEEE802.11 drivers

  2.Device Drivers à Network device support à Wireless LAN à <M> Marvell 8xxx Libertas WLAN driver support with thin firmware

  <M>Marvell 8xxx Libertas WLAN driver support

  <M>Marvell Libertas SDIO 802.11b/g cards

  制作内核映象文件：

  make uImage

  make modules

SDIO内核驱动注册流程(1)

- **platform_driver_register(&s3cmci_driver)**
  - 注册一个平台设备
- **s3cmci_probe**函数
  - 首先分配一个**mmc_host**结构体，**mmc_alloc_host(sizeof(struct s3cmci_host), &pdev ->dev)**，这样就能在**mmc_host**中找到了**s3cmci_host**，对**mmc_host**结构体的部分成员进行了初始化，并且初始化了工作队列，对应的操作函数为**mmc_rescan**，此函数对插入的卡进行扫描区分卡的类型并且对其初始化。
  - 接下来向系统请求**GPIO**口（**GPE[5-10]**），并注册一个**tasklet**，回调函数是**pio_tasklet**（使用**PIO**模式进行读写数据，在中断处理函数**s3cmci_irq**会回调该函数）。
  - 向设备平台资源获取寄存器的物理地址（开始地址（**0x5A000000**）和结束地址**(0x5A0FFFFF)**），并向内存申请大小一样的区域块，最后对物理地址动态映射成虚拟地址。
  - 向设备平台资源获取中断号**(37)**，并进行中断注册，中断处理函数是**s3cmci_irq**。

SDIO内核驱动注册流程（2）

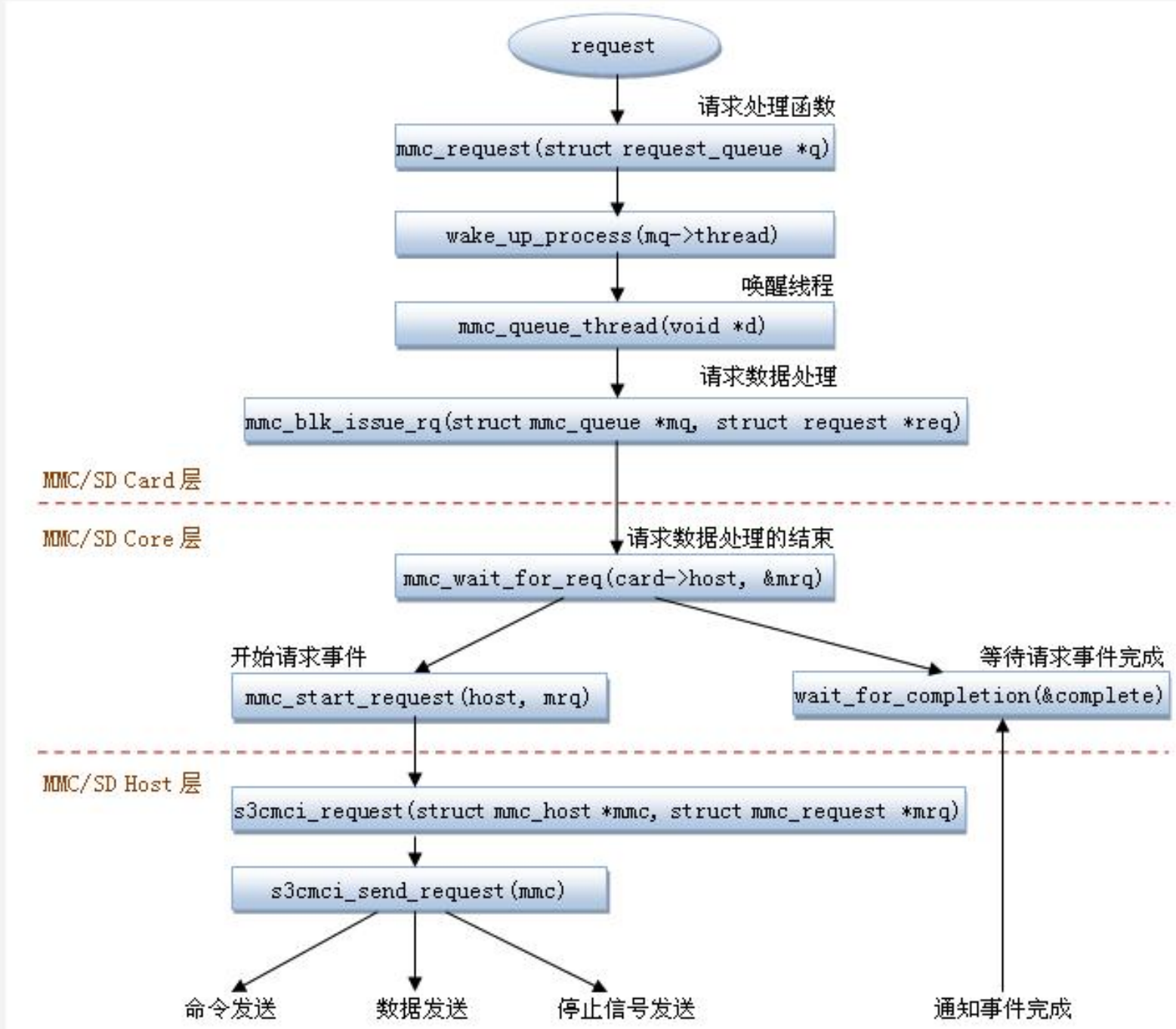
- 为mmc->pos赋值，s3cmci_ops结构体

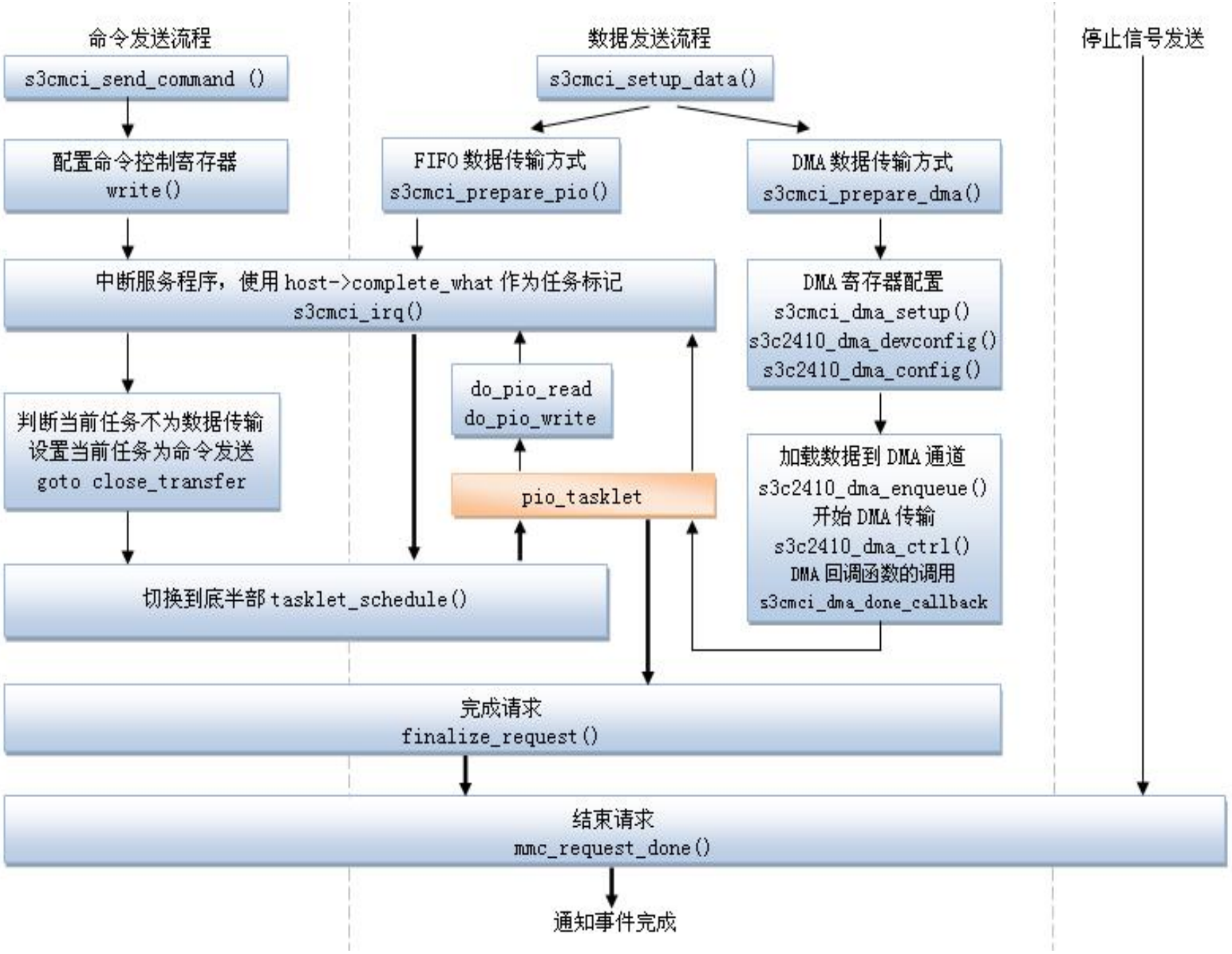
```
static struct mmc_host_ops  s3cmci_ops = {
.request        = s3cmci_request, //实现了命令和数据的发送和接收
.set_ios= s3cmci_set_ios, //来设置硬件IO,包括引脚配置,使能时钟,和配置总线带宽.
.get_ro= s3cmci_get_ro,// 通过从GPIO读取，来判断我们的卡是否是写保护的
.get_cd= s3cmci_card_present, //通过从GPIO读取来判断卡是否存在
.enable_sdio_irq = s3cmci_enable_sdio_irq,//使能SDIO的IRQ
};
```

- 命令和数据的发送和接收

s3cmci_request

- 命令和数据的发送和接收 **(.request= s3cmci_request)**
  - 先确定目前是否有卡，如果检测没有，则调用**mmc**请求去完成进程。如果卡存在则调用**s3cmci_sned_request**来发送请求
  - 先对**command,cmd,fifo**寄存器清**0**，如果确定在**cmd**中有数据，将调用**s3cmci_setup_data**（）数据请求处理设置，然后对这个函数做个检测，是否更新成功，如果不成功，再次调用**mmc_request_done**（）请求。
  - 判断主机是采用**dma**还是使用**pio**通道，这里默认为**PIO**通道，再检测它是否准备成功
  - **S3cmci_send_command(host,cmd);**做第二次发送命令。
  - 最后使能中断**s3cmci_enable_irq()** 。


request
请求处理函数
mmc_request(struct request_queue *q)
wake_up_process(mq->thread)
唤醒线程
mmc_queue_thread(void *d)
请求数据处理
mmc_blk_issue_rq(struct mmc_queue *mq, struct request *req)
MMC/SD Card 层
MMC/SD Core 层
请求数据处理的结束
mmc_wait_for_req(card->host, &mrq)
开始请求事件
mmc_start_request(host, mrq)
等待请求事件完成
wait_for_completion(&complete)
MMC/SD Host 层
s3cmci_request(struct mmc_host *mmc, struct mmc_request *mrq)
s3cmci_send_request(mmc)
命令发送
数据发送
停止信号发送
通知事件完成


命令发送流程
s3cmci_send_command ()
配置命令控制寄存器
write()
数据发送流程
s3cmci_setup_data()
FIFO 数据传输方式
s3cmci_prepare_pio()
DMA 数据传输方式
s3cmci_prepare_dma()
停止信号发送
中断服务程序，使用 host->complete_what 作为任务标记
s3cmci_irq()
DMA 寄存器配置
s3cmci_dma_setup()
s3c2410_dma_devconfig()
s3c2410_dma_config()
判断当前任务不为数据传输
设置当前任务为命令发送
goto close_transfer
do_pio_read
do_pio_write
pio_tasklet
加载数据到 DMA 通道
s3c2410_dma_enqueue()
开始 DMA 传输
s3c2410_dma_ctrl()
DMA 回调函数的调用
s3cmci_dma_done_callback
切换到底半部 tasklet_schedule()
完成请求
finalize_request()
结束请求
mmc_request_done()
通知事件完成

mmc_rescan分析

- 当检测初始化好了， 发送**cmd52**复位，（读写寄存器和**CIS(**卡信息结构**)**表）

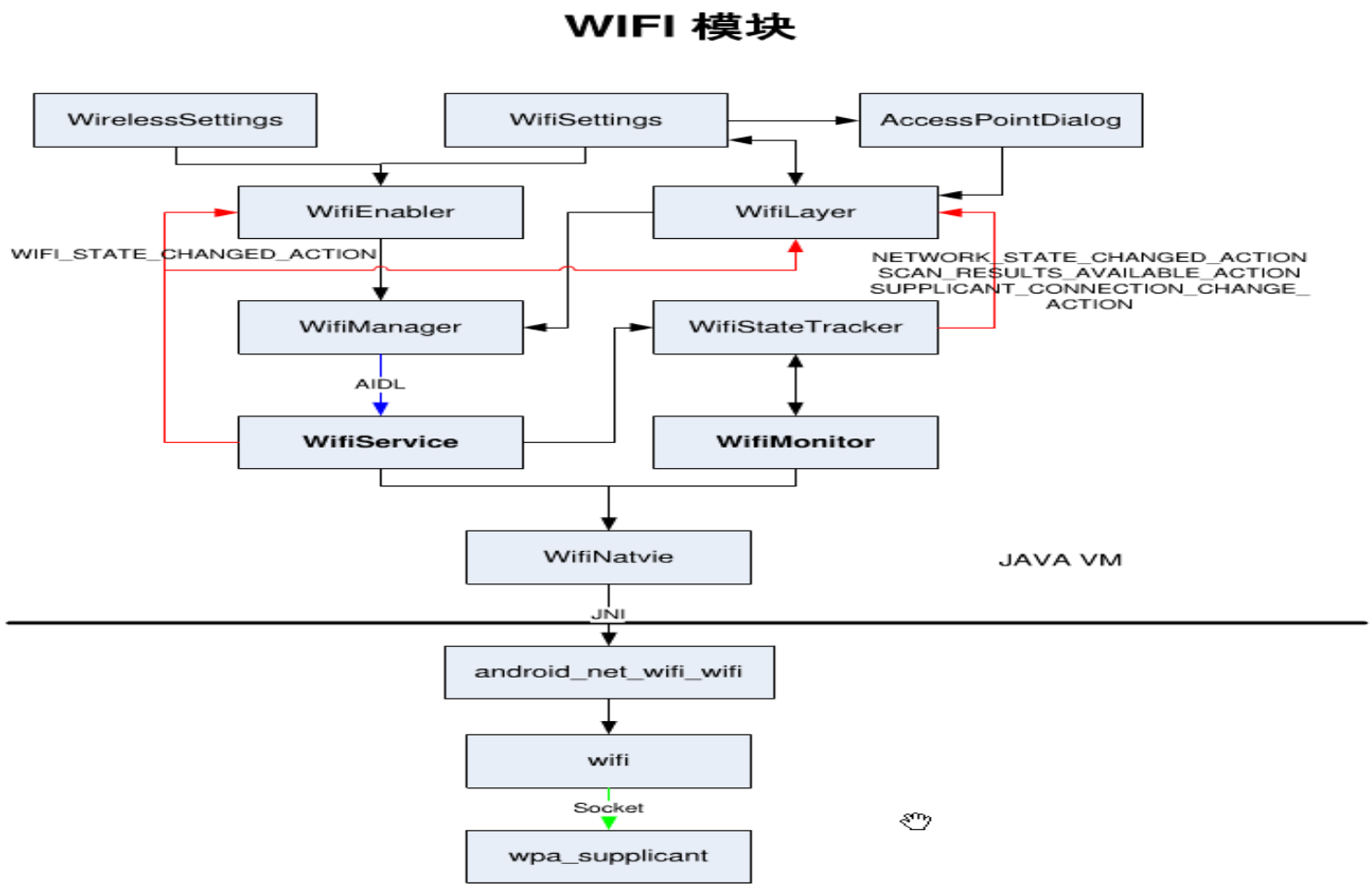
- **mmc_send_io_op_cond(host, 0, &ocr),CMD5**发送，只有当**SDIO**卡收到**cmd5**之后，才会返回**R4**寄存器（询问和设置卡的电压范围），当没接受到**cmd5**命令时，不会对任何命令作出反应。

- 接着进入**sdio**卡初始化**mmc_sdio_init_card()**

- **mmc_alloc_card()**分配卡结构体，然后发送**CDM3,CMD7**命令，相应取得卡相关地址，然后以**CMD3**返回来的地址**RCA**作为**CMD7**的参数，使卡进入命令状态或是准备状态，如果卡设定已分配到的**RCA**地址后，将从**stand-by**状态转化到**command**状态，并将回应以后的读取指令及其他指令。如果给卡设定**RCA**以处的地址和**RCA=000h**时，卡将转换到**stand-by**状态。然后读命令状态寄存器**CCCR**与**CIS**，设置卡速度。

华清远见 FARSIGHT 嵌入式培训专家

Android应用和中间层



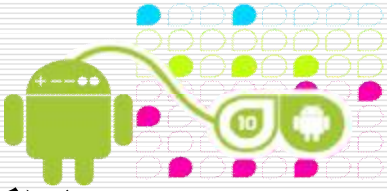
华清远见 FARSIGHT 嵌入式培训专家

Wifi模块的初始化:

- 在 **SystemServer** 启动的时候，会生成一个 **ConnectivityService** 的实例，
-

```
try {
Log.i(TAG, "Starting Connectivity Service.");
ServiceManager.addService(Context.CONNECTIVITY_SERVICE, new
ConnectivityService(context));
} catch (Throwable e) {
Log.e(TAG, "Failure starting Connectivity Service", e);
}
```

华清远见 FARSIGHT 嵌入式培训专家



Android应用和中间层工作流程

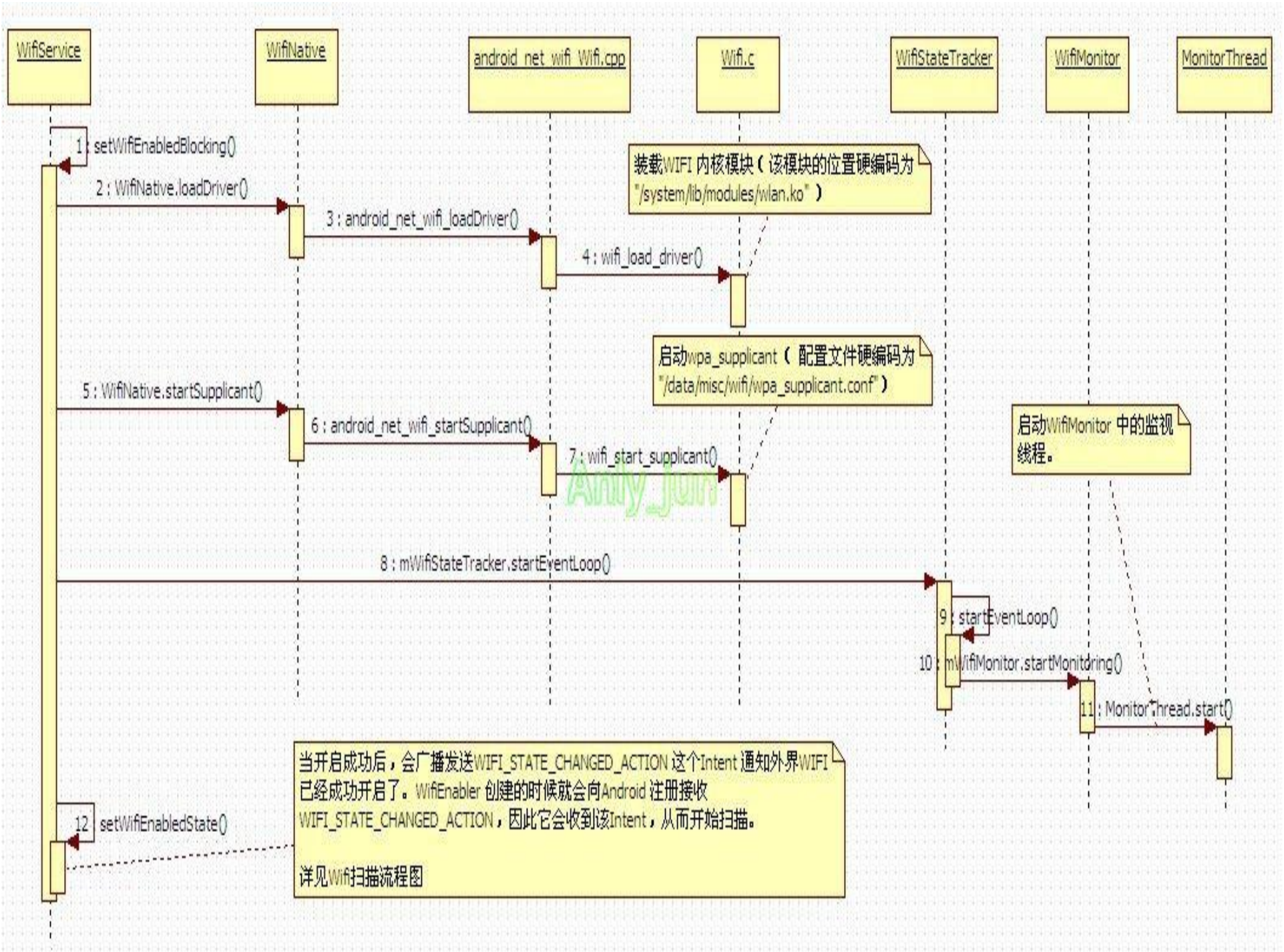
- 初始化
- 使能**WIFI**
- 查找**AP**
- 配置**AP**参数
- 连接
- 配置**IP**地址

Wifi模块的初始化:

- 在 **SystemServer** 启动的时候，会生成一个 **ConnectivityService** 的实例，

- **ConnectivityService** 的构造函数会创建**WifiService**，

- **WifiStateTracker** 会创建**WifiMonitor** 接收来自底层的事件，**wifimonitor**把它转化成**wifistateTracker**能识别的消息。**WifiService** 和**WifiMonitor** 是整个模块的核心。**WifiService** 负责启动关闭**wpa_supplicant**、启动关闭**WifiMonitor** 监视线程和把命令下发给**wpa_supplicant**，而**WifiMonitor** 则负责从**wpa_supplicant** 接收事件通知。


WifiService
WifiNative
android_net_wifi_Wifi.cpp
Wifi.c
WifiStateTracker
WifiMonitor
MonitorThread
1 : setWifiEnabledBlocking()
2 : WifiNative.loadDriver()
3 : android_net_wifi_loadDriver()
4 : wifi_load_driver()
装载WIFI 内核模块（该模块的位置硬编码为"/system/lib/modules/wlan.ko"）
启动wpa_supplicant（配置文件硬编码为"/data/misc/wifi/wpa_supplicant.conf"）
5 : WifiNative.startSupplicant()
6 : android_net_wifi_startSupplicant()
7 : wifi_start_supplicant()
启动WifiMonitor 中的监视线程。
8 : mWifiStateTracker.startEventLoop()
9 : startEventLoop()
10 : mWifiMonitor.startMonitoring()
11 : MonitorThread.start()
12 setWifiEnabledState()
当开启成功后，会广播发送WIFI_STATE_CHANGED_ACTION 这个Intent 通知外界WIFI已经成功开启了。WifiEnabler 创建的时候就会向Android 注册接收WIFI_STATE_CHANGED_ACTION，因此它会收到该Intent，从而开始扫描。
详见Wifi扫描流程图

华清远见
FARSIGHT
嵌入式培训专家

Q&A

THANKS!