Atheros SDK 驱动原理详解

路涛

修改记录

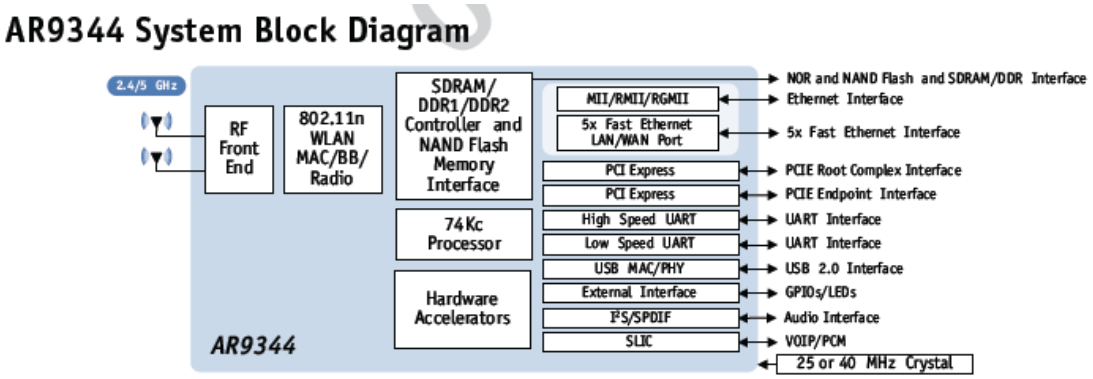
版本号	拟制人/修改人	拟制/修改日期	更改理由	主要更改内容（写要点即可）
1.0	路涛	2012/7/30	创建	

目　　录

1 Atheros 芯片概要介绍

1.1 序言

Atheros 802.11 系列芯片采用 SOC 方式设计，即在单一芯片内集成包括 CPU,802.11MAC,PHY,I/O 接口等多个功能。典型如下图 AR9344 的框图所示：



1.2 WLAN MAC 典型示例介绍：AR9344

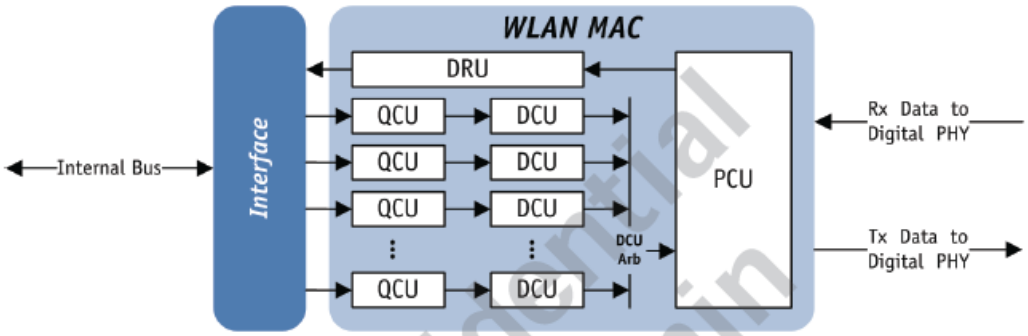


Figure 6-1. **WLAN MAC Block Diagram**

1.2.1 QCU（Queue Control Unit）

QCU 负责发送描述符的管理，配合 DCU 把数据帧从无线空口发送出去。在 9344 上的 QCU 每个队列的深度是“8”。也就是最大可以缓存 8 个待发送的无线帧。QCU 一共有 10 个硬件队列，每个队列在从无线发送出去的调度是严格优先级区分。

1.2.2 DCU (DCF Control Unit)

Distributed coordination function（DCF） control unit，DCU 主要负责前面所讲到的优先级调度仲裁，该对应 QCU 待发送的报文是否允许占用无线空口资源。DCU 还负责无线竞争中可以通过硬件实现的相关处理，如 SIFS 的设置，就是通过配置 DCU。

1.2.3 PCU（Protocol Control Unit）

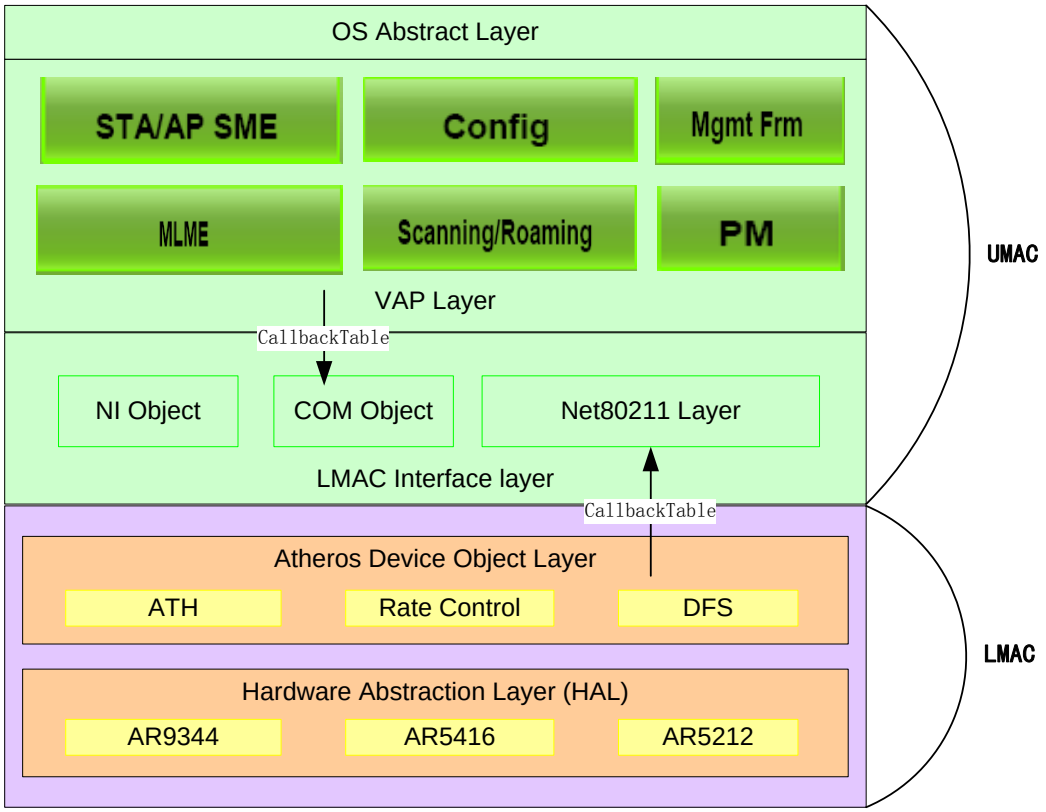
PCU，顾名思义，主要负责MAC协议中定义的诸多子功能的实现：

- 加密/解密；
- 缓存收发的报文；
- 自动产生 ACK,RTS,CTS 等无线控制帧；
- 插入 FCS 或者校验 FCS；
- Etc；

2 Atheros 驱动原理详解

2.1 Atheros 驱动架构说明

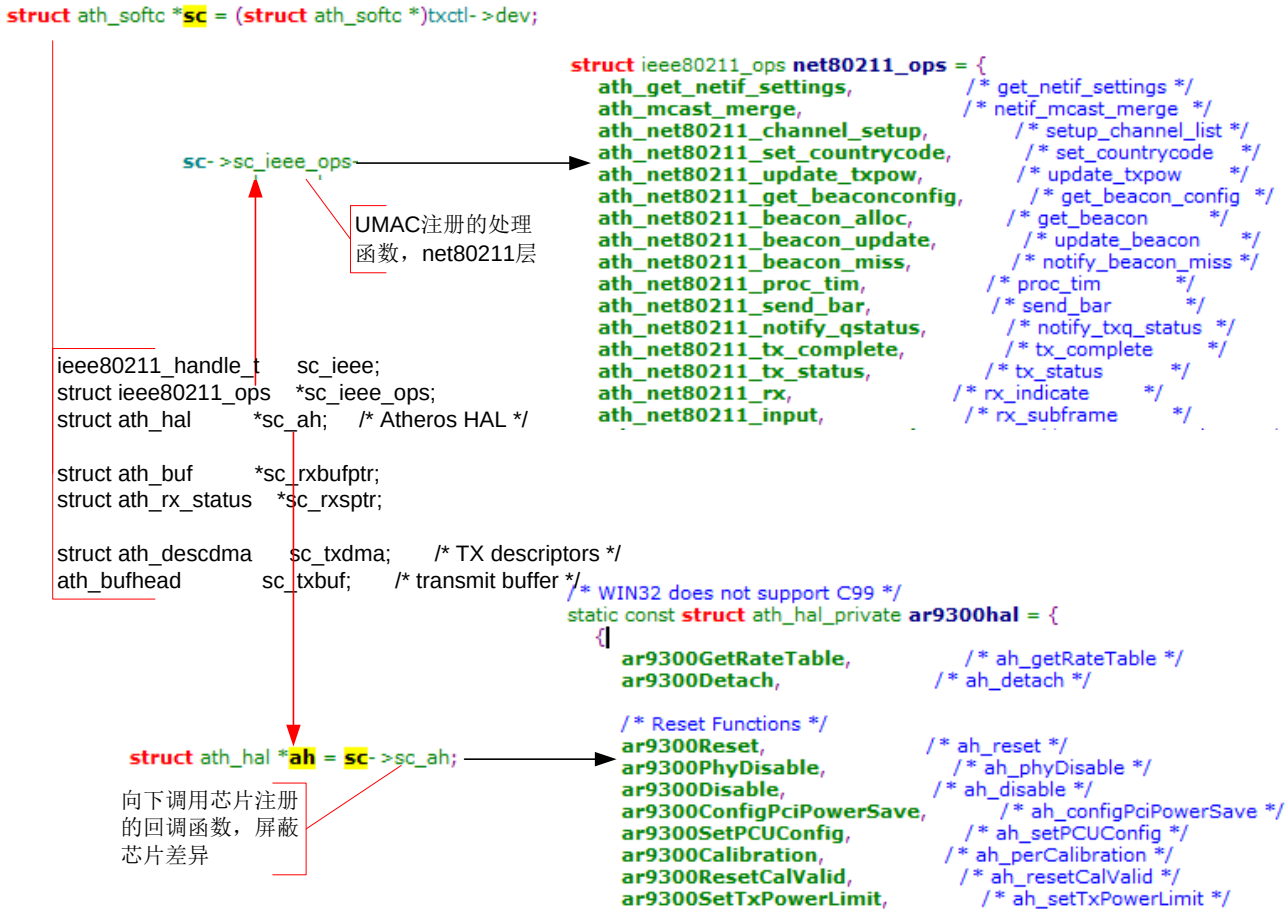
Atheros 驱动模块设计相对复杂，由于历史的原因，驱动原始的架构在后续演进过程中，出现了交叉调用，导致层次关系变得复杂，不够清晰。



- LMAC：主要负责和芯片相关的底层操作，以及对应的底层特性实现；
- UMAC：主要负责 IEEE80211 相关协议的实现，和对操作系统驱动架构的适配；

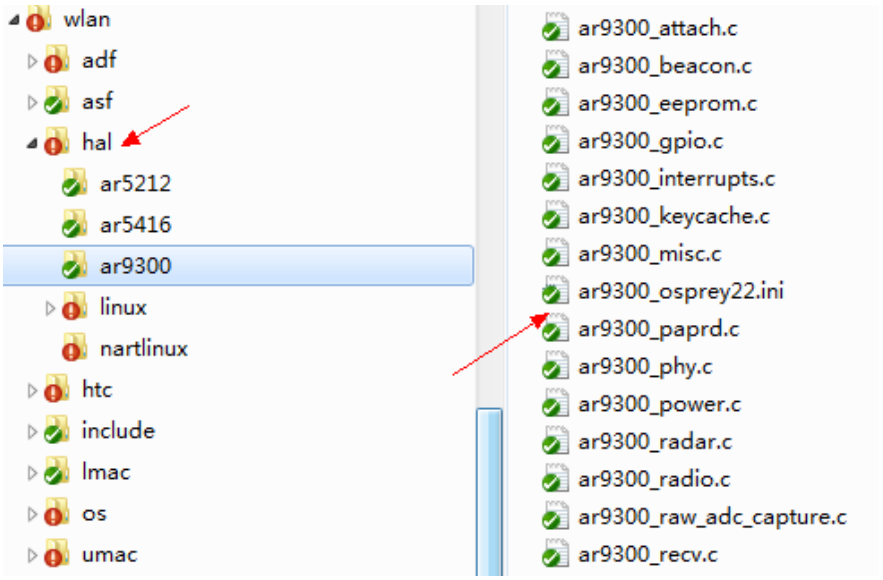
2.1.1 LMAC

LMAC 主要包括两部分功能：硬件抽象层 HAL，和 MAC 下半部相关的协议处理，典型如 11n 下的聚合处理。LMAC 的核心数据结构如下图所示：



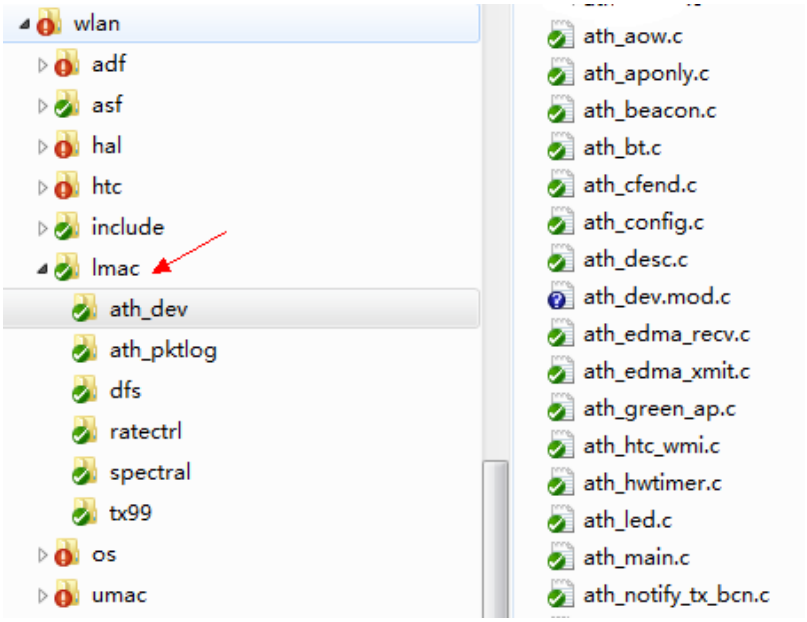
1. HAL 层

   硬件抽象层主要是屏蔽不同芯片的差异，像上层调用注册统一的回调函数，完成芯片的各项子功能。因此特定芯片的访问，寄存器读写，初始化等等，都是通过对应芯片具体的驱动代码实现。



2. ATH 设备层（Atheros Device Object）

   ATH 设备层主要负责 MAC 下半部的特性实现，包括电源管理，聚合，Becaon 帧等。



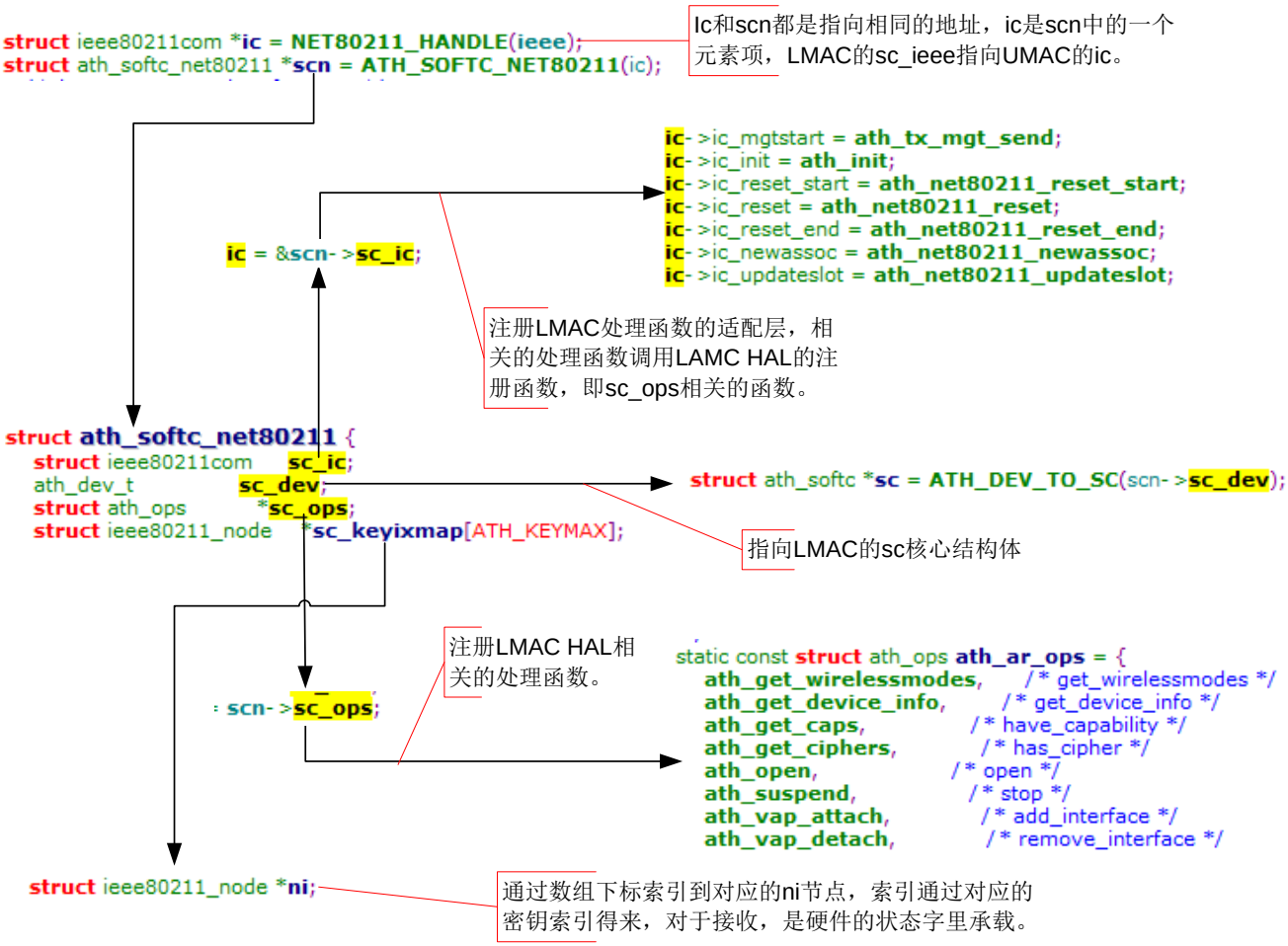
ATH 层面向 HAL 层进行抽象，通过回调调用不同的芯片的底层驱动：

- ath_hal：Hardware Access Layer，定义，抽象，芯片相关的统一调用接口，芯片驱动初始化时注册具体的回调函数，如下图所示，9300 注册的私有接口。
- ath_softc：ATH 层的内部结构体，主要承载该注册的接口相关属性，待发送报文的缓存队列，另外重要的是可以索引到对应的 HAL 结构体。

3. LMAC 其它比较重要的功能模块
   - Rate Control：根据目前空口无线传输质量，比如丢包率等等，计算当下报文的发送速率；
   - DFS：Dynamic Frequency Selection algorithm，在 5G 频段，能够检测到雷达等干扰源，并自动切换频率；
   - Spectral：频谱扫描相关功能的实现；

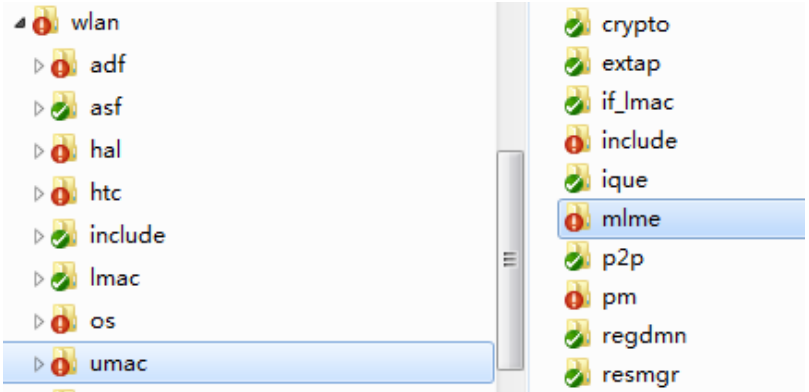
2.1.2 UMAC

UMAC 主要负责 802.11 协议相关的处理。已经诸多增强特性的具体实现。核心数据结构如下图所示：



- Net80211 层：LMAC 的适配层，注册到 LMAC 相关的 IEEE 相关的操作。避免底层 LMAC 直接调用 UMAC 的函数；
- VAP Layer：IEEE80211 实现层，以及和操作系统相关的接口层，VAP 层尽可能和具体的硬件无关，因此对于 VAP 层，向下的部分操作通过调用 COM 组件的接口函数实现。
- Ni 节点：每个具体的 Station 都对应一个 ni 节点，用来承载该 station 具体的报文收发缓存，无线属性，状态等；
- Com 组件：接口组件，向上提供给 80211 层回调的函数，以实现具体的 80211 功能，向下和具体的硬件结合，因此它是硬件的适配层

2.1.3 UMAC 特性说明



UMAC 目录下，包含了诸多 MAC 特性的具体实现，本节以代码宏作为索引进行特性说明：

特性	功能	产品支持
UMAC_SUPPORT_ACL	MAC 地址过滤，实现原理是通过 HASH 表，在关联过程中直接过滤	打开
UMAC_SUPPORT_ACS	信道自动选择	打开
ATH_SUPPORT_AOW	Audio over WIFI	无用
ATH_EXT_AP	Extensible Access Point (ExtAP) operation mode, the 802.11 station operates as a wireless LAN (WLAN) access point	无用
MLME	关联，Probe，Becaon 等相关的实现	重要
UMAC_SUPPORT_AP_POWERSAVE	节电模式	打开
UMAC_SUPPORT_P2P	A peer-to-peer (P2P) network allows wireless devices to directly communicate with each other. Wireless devices within range of each other can discover and communicate directly without involving central access points. This method is typically used by two computers so that they can connect to each other to form a network.	无用
UMAC_SUPPORT_REGDMN	The Beacon frame contains information on the country code，允许修改 country code	打开
UMAC_SUPPORT_RESMGR	resource manager	
UMAC_SUPPORT_RPTPLACEMENT	Repeater placement assist feature，直放站，基于 Atheros 私有消息传递，实现功能。	关闭
UMAC_SUPPORT_SCAN	基础组件，信道的扫描，是 ACS 等功能的基础包	打开
UMAC_SUPPORT_SMARTANTENNA	智能天线，依赖 atheros 私有消息实现功能	关闭
UMAC_SUPPORT_TDLS	Tunneled Direct Link Setup (TLDS) enables devices to intelligently negotiate between themselves and determine methods that will avoid or reduce network congestion. IEEE 802.11z. This 802.11z amendment defines mechanisms that allow IEEE 802.11™ to set up a direct link between client devices while also remaining associated with the (AP). These mechanisms are referred to as Tunneled Direct Link Setup (TDLS). A TDLS direct link is set up automatically, without need for user intervention, while the connection with the AP is maintained. TDLS is a client-only feature	无用
ATH_SUPPORT_TxBF	Transmit Beamforming	关闭
UMAC_SUPPORT_WDS	Wireless Distribution System，即无线分布式系统。以往在无线应用领域中他都是帮助无线基站与无线基站之	

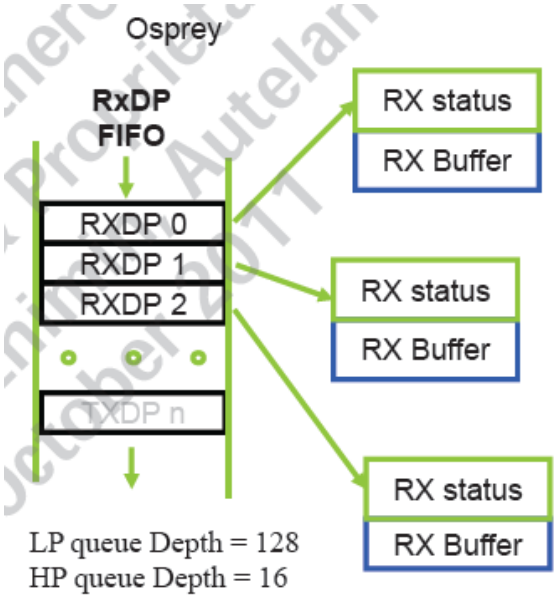
	间进行联系通讯的系统。在家庭应用方面则略有不同，WDS 的功能是充当无线网络的中继器，通过在无线路由器上开启 WDS 功能，让其可以延伸扩展无线信号，从而覆盖更广更大的范围。说白了 WDS 就是可以让无线 AP 或者无线路由器之间通过无线进行桥接（中继），而在中继的过程中并不影响其无线设备覆盖效果的功能。	
UMAC_SUPPORT_NAWDS	Non-Associated WDS - NAWDS	

2.2 Atheros SDK 收发包详解

Atheros SDK 收发包包括驱动中断部分的处理，和对应 Tasklet 的后续处理。本节主要对驱动的工作原理，数据结构，调用流程进行详细的介绍。

注：以下的调用流程里的函数命名，以“APONLY”的流程作为示例，相关函数去掉“APONLY”既是标准流程。

2.2.1 驱动收报原理



Atheros 新的芯片系列支持两个收报队列，采用不同的优先级，以保证高优先级队列能够优先处理。

初始化时，驱动软件把 RxDP 写入芯片寄存器，每个 RxDP 包括两个部分，RxStatus 和 RxBuffer。

- RxStatus 字段用来存放接收报文对应的状态信息；
- RxBuffer 用来存放报文；

硬件首先把报文 DMA 到 RxBuffer 指向的地址，然后把对应的状态信息填写到 RxStatus 字段中。这里 RxBuffer 和 Status 是一段连续的空间，Status 在起始地址，RxBuffer 跳过 RxStatus 字段后就是。

接收流程中比较重要的函数是“**ath_rx_intr**”，这个函数把硬件映射的缓存，搬运到延后处理要处理的队列中，保证中断处理可以快速的结束退出。

这里的核心数据结构如下图：

```
/* Receive FIFO management */
struct ath_rx_edma {
    wbuf_t              *rxfifo;
    u_int8_t            rxfifoheadindex;
    u_int8_t            rxfifotailindex;
    u_int8_t            rxfifodepth;       /
    u_int32_t           rxfifohwsize;
    ath_bufhead           rxqueue;
    spinlock_t          rxqlock;
};
```

Rxfifodepth：硬件缓冲区缓存可用的数量；

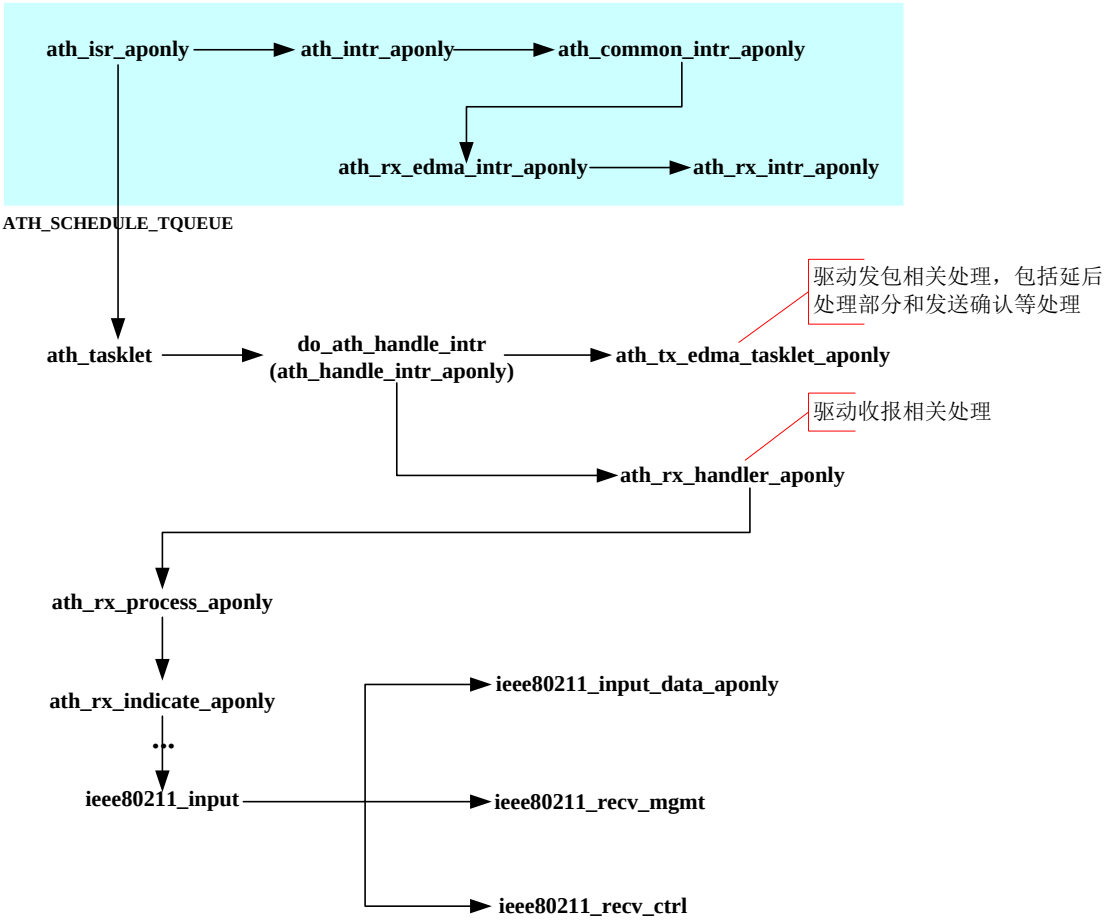
Rxfifo：硬件缓冲区对应的缓存；

Rxqueue：延后队列待软件处理的报文队列；在中断处理中，会把硬件接收完成的报文，从 rxfifodepth 中拷贝到 rxqueue 中，同时会从 sc->sc_rxedma 取得新的 buffer 挂接到硬件缓冲区 rxfifo 里。

2.2.2 驱动收报调用流程

下面首先对主要函数进行概要说明，调用关系如下图所示：

- ath_rx_handler_aponly： 中断处理调用函数，从 SC->EDMA 中获取报文和对应的描述符，并进一步处理。和 Linux 系统相关又与具体的硬件相关。
- ath_rx_process_aponly：更新 rx_status。
- ath_rx_indicate_aponly：调用 ath_net80211_rx_aponly，并向硬件描述符挂接新的 buffer。
- ath_net80211_rx_aponly：1. 获取上层使用的结构的指针，主要是 NI 的获取；2.可能调用 ieee 层的函数，接收处理报文，聚合流程特殊，直接在 OS 层里的函数进行处理。
- ieee80211_input_aponly：根据帧的类型，调用 IEEE 层的函数处理，如果是数据帧，则继续调用本层的聚合处理函数处理；

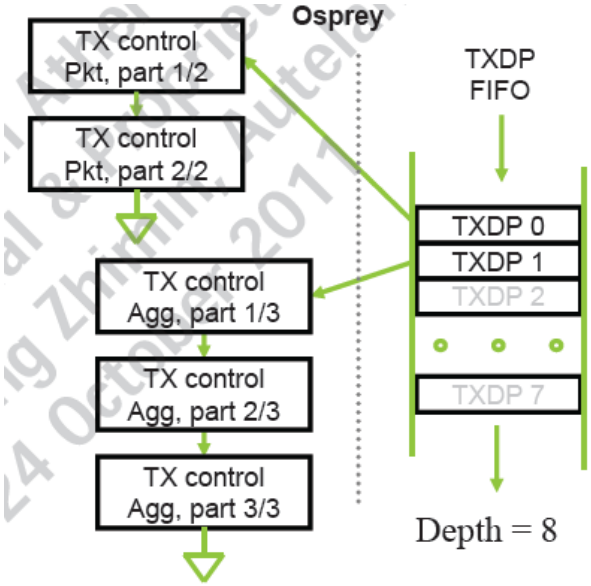


较为详细的中断侧处理：

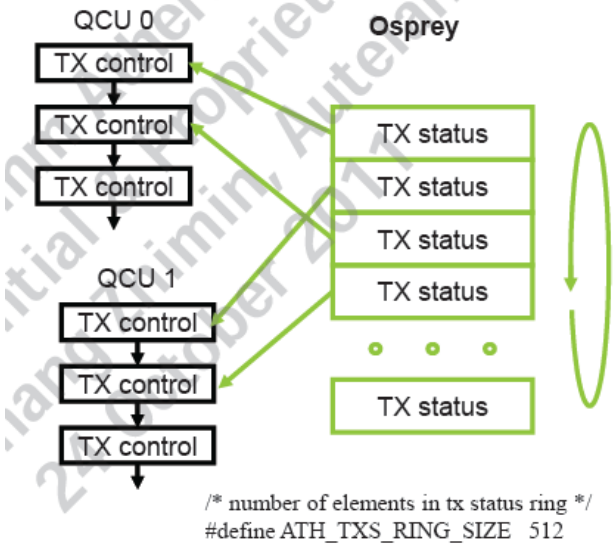
Rx ISR Function Flows

ath_rx_edma_intr(), ath_dev\ath_edma_recv.c
- if (status & (HAL_INT_RXHP | HAL_INT_RXEOL | HAL_INT_RXORN))
  - ath_rx_intr(dev, HAL_RX_QUEUE_HP);
- if (status & (HAL_INT_RXLP | HAL_INT_RXEOL | HAL_INT_RXORN))
  - ath_rx_intr(dev, HAL_RX_QUEUE_LP);
    - wbuf = rxedma->rxfifo[rxedma->rxfifoheadindex]; Get a wbuf from Rx FIFO
    - bf = ATH_GET_RX_CONTEXT_BUF(wbuf); Get ath_buf from the wbuf.
    - ath_hal_rxprocdescfast()=ar9300ProcRxDescFast(), ath_dev\ath_recv.c Process an RX descriptor and populate ath_rx_status *rxs
    - TAILQ_INSERT_TAIL(&rxedma->rxqueue, bf, bf_list); add ath_buf for DPC
      rxedma->rxfifo[rxedma->rxfifoheadindex] = NULL; INCR(rxedma->rxfifoheadindex, rxedma->rxfifohwsize); rxedma->rxfifodepth--; advance the head pointer
    - /* remove ath_bufs from free list and add it to fifo */
      frames = rxedma->rxfifohwsize - rxedma->rxfifodepth;
      if (frames > 0) ath_rx_addbuffer(sc, qtype, frames);
      - ath_rx_buf_link(), ath_edma_recv.c →ath_hal_putrxbuf()→ar9300SetRxDP()
        →OS_REG_WRITE(ah, AR_HP_RXDP/ AR_LP_RXDP, rxdp);

2.2.3 驱动发包原理



硬件支持 10 个发送队列，每个硬件发送队列最大支持 8 个缓存包；队列的调度是基于优先级的调度；这里所有的发送队列共用一个发送状态描述符队列，软件跟踪队列的尾部，获取发送报文的状态，状态队列是一个“环”，如下图所示：

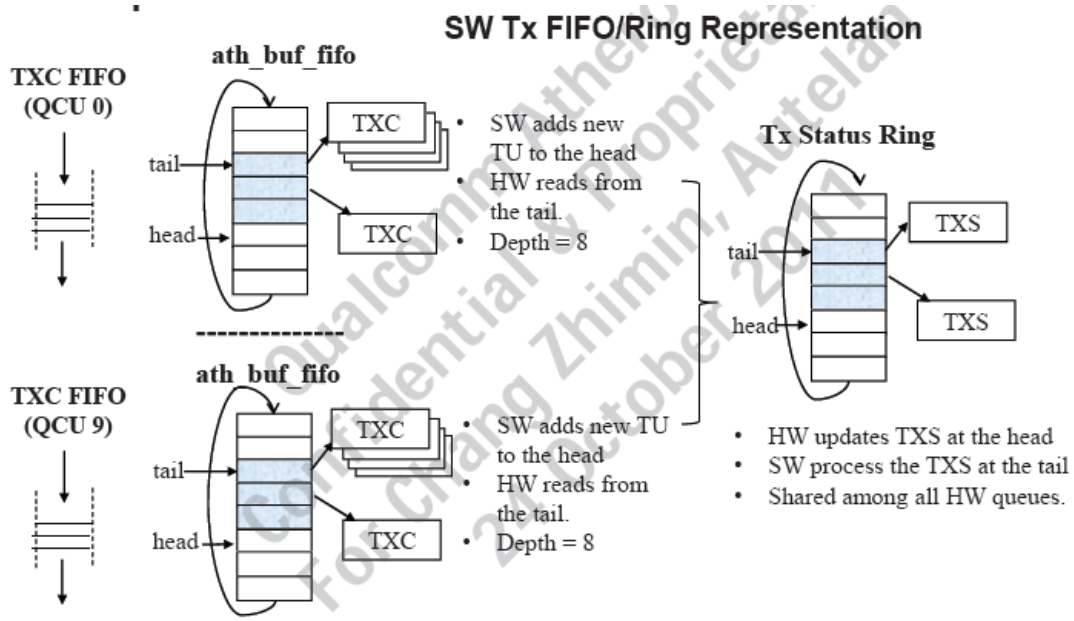


工作时，驱动把准备发送的帧写入硬件队列的寄存器，硬件 QCU 则读出，并最终通过 PHY 发送出去，触发后续的中断，更新发送的状态。

操作系统的协议栈调用注册的驱动发包函数发送报文下，驱动先根据硬件队列状态判断该帧是直接发送还是需要放入延后处理的软件缓存队列。驱动通过软件的缓存，可以把协议栈发送的报文缓存后进行聚合处理，以提高无线的发送效率。这里 Atheros 驱动的延后处理是通过 TxOK 中断触发的 Tasklet 进行处理。

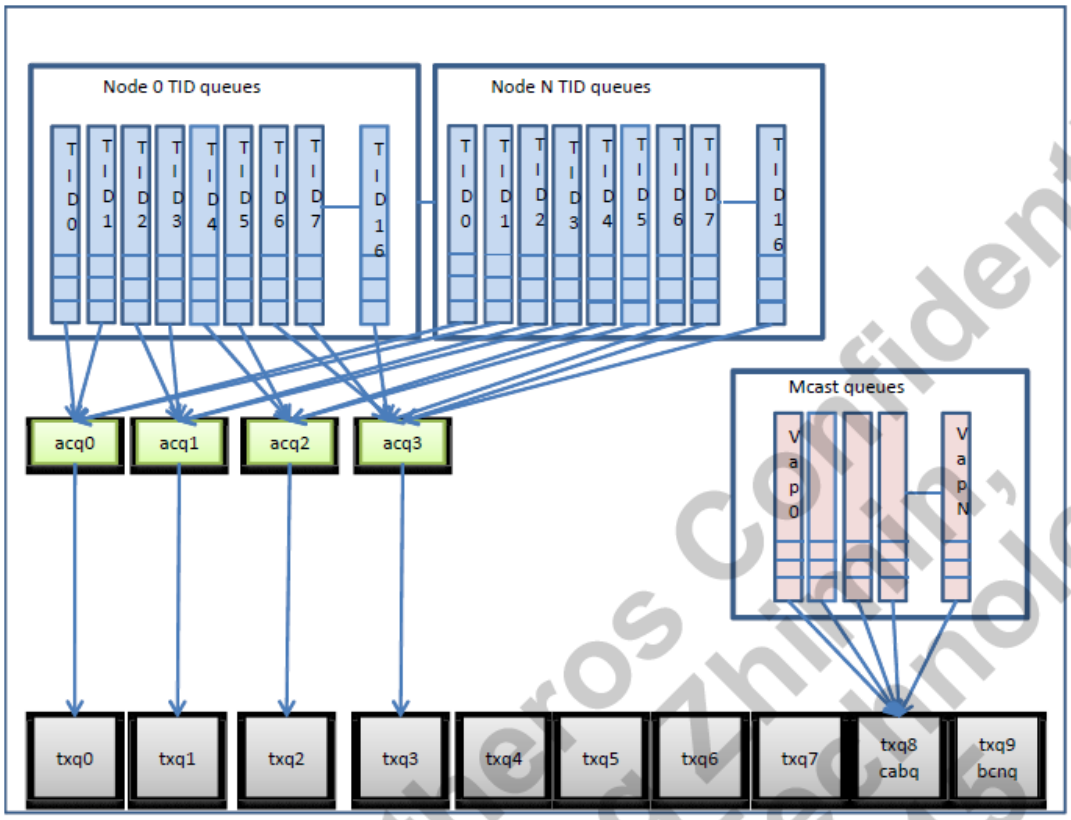
因此如果当前的硬件队列为空，就需要直接进行发送。要不硬件队列没有待发送的报文，就无法触发中断，这时该报文如果放入软件队列，就会形成“死循环”，不会有后续的被触发的 tasklet 把报文搬运给硬件。

中断触发的 tasklet 里会进行当前是否可以进行聚合处理还是单帧发送的判断和对应的处理。



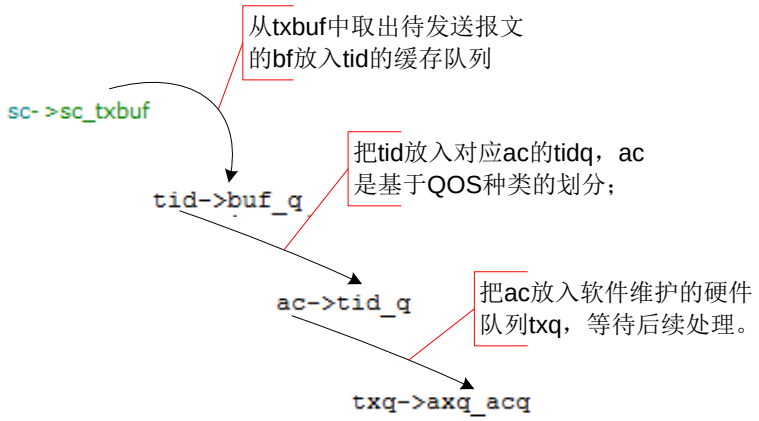
下面针对典型的延后处理流程描述对应的缓存操作：

- txq(ath_txq)： 是软件发送相关的队列和对应的状态信息，包含了延后缓存的链表组织，也包含了软件跟踪发送状态的链表；
  - axq_fifo(axq_q 不支持 EDMA 的芯片)，对于延后处理，当该报文放入到硬件队列后，其对应的描述符就从 tid 队列中提取出来，挂接到这里，在 txok 触发的延后处理，会读取本队列里的状态描述符，来判断硬件是否处理完成；该队列是个数组，一共八个，软件对于新增报文插入到 head，然后读取 tail，轮训发送结果，对于每一个数组元素，又是一个链表，对于单播帧，就是一个 bf，对于聚合帧，其实挂接了该聚合帧的所有子帧。
- txctl(ieee80211_tx_control_t)：每个帧对应的控制块；
- tid(ath_atx_tid)：每个 station 对应的结构体，根据控制块 txctl 的 an 字段获取，tid 中的 buf_q 字段存放这个目的 station 的缓存待发送的报文，具体的发送根据条件判断，会进行延后处理，延后处理从这里取帧在放入硬件队列。



sc->sc_txbuf 用来缓存初始化时候申请的所有描述符，在协议栈准备发包的时候，上半部处理要先在 sc->sc_txbuf 中申请 buf（ath_tx_get_buf），如果申请不到，就会报 nobuf 错误后丢包。申请到的报文会缓存到 tid 的队列中，供后续处理。

对于一个报文，根据其对应的 QOS，也就是 txctl 描述符，获取到对应的 txq 硬件队列（txq = &sc->sc_txq[txctl->qnum]），和对应的 tid（tid = ATH_AN_2_TID(an, txctl->tidno)，“an->an_aggr.tx.tid[(tidno)]”），然后通过 ath_tx_queue_tid 进行 txq，tid，ac 之间的指针互指，关联到一起。其实 tid，ac，txq 都是数组形式，初始化好，只是每个报文通过 QOS 映射后，进行另一层面的数据结构的关联，比如通过 txq 索引，就变成了 ac 的双向链表，在 tid 的双向链表，一级一级的又一次指向。



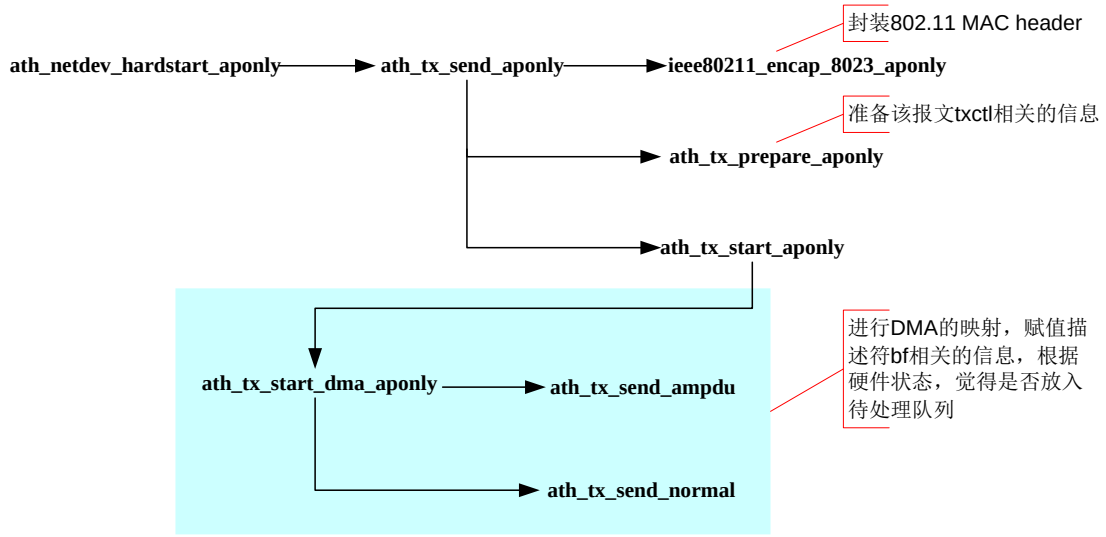
驱动的调度发包原理：

每次调度触发轮训，round-robin 机制，首先取得对头的 ac，然后在 ac 上在找到对应的

tid，并取包发送，调度后的 tid 如果还有待处理的报文，则挂回 ac 链表的尾部。对应的 ac 如果还有 tid，则进一步挂回 axq_acq 的队列尾部。

2.2.4 驱动发包调用流程

下图主要描述了从协议栈下发的报文处理流程：对报文进行 80211 的封装，准备发送描述符，针对硬件队列的状态进行判断是否放入软件的缓存队列还是直接进行发送处理。



下图主要描述了中断触发的 Tasklet 的延后处理的调用流程，包括如何从软件队列把报文聚合发送，如何处理已经发送的报文，和重传报文。