# 配置线(好像只下发了配置没有修改硬件啊啊啊)

### kochab

int

dl\_wfradio\_set\_chanbw (struct db\_table\_hdr \*tbh, int m,

int chanbw)

{

dl\_iwpriv\_set(tb[m].name, "bhuChanBw", (char \*)&val, 1, sizeof(int))

}

int

dl\_iwpriv\_set(const char \*ifname, const char \*cmdname, const char \*args, int count, int sz)

{

skfd = if\_enter() //创建socket

dl\_iwpirv\_cmd(skfd, ifname, cmdname, &priv, &cmd, &subcmd)//获取正真的ioctl命令

ioctl(skfd, cmd->cmd, &wrq)//下发命令

}

## 获取ioctl命令列表 dl\_iwpirv\_cmd

### kochab

int

dl\_iwpirv\_cmd(int skfd, const char \*ifname, const char \*cmdname,

iwprivargs \*\*privp, iwprivargs \*\*cmd, iwprivargs \*\*subcmd)

{

num = iw\_get\_priv\_info(skfd, ifname, &priv);//获取该接口（ifname）支持的ioctl命令列表。

。。。。

匹配出真正的ioctl命令

}

int

iw\_get\_priv\_info(int skfd,

const char \* ifname,

iwprivargs \*\* ppriv)

{

Do

{

//每次读取16个命令。

iw\_get\_ext(skfd, ifname, SIOCGIWPRIV, &wrq)

}

}

static inline int

iw\_get\_ext(int skfd, /\* Socket to the kernel \*/

const char \* ifname, /\* Device name \*/

int request, /\* WE ID \*/

struct iwreq \* pwrq) /\* Fixed part of the request \*/

{

ioctl(skfd, request, pwrq) //下内核 (SIOCGIWPRIV)

}

### 内核：

static const struct file\_operations socket\_file\_ops = {

.unlocked\_ioctl = sock\_ioctl,

#ifdef CONFIG\_COMPAT

.compat\_ioctl = compat\_sock\_ioctl,

};

sock\_ioctl/ compat\_sock\_ioctl

---🡪 dev\_ioctl(){

if (cmd >= SIOCIWFIRST && cmd <= SIOCIWLAST) // SIOCIWFIRST:0x8B00 SIOCIWLAST:0x8BFF

return wext\_handle\_ioctl(net, &ifr, cmd, arg);

}

---🡪 wext\_handle\_ioctl(){}

---🡪 wext\_ioctl\_dispatch(){}

---🡪 wireless\_process\_ioctl(){

dev = \_\_dev\_get\_by\_name(net, ifr->ifr\_name))//获取接口对应的dev

if (cmd == SIOCGIWSTATS)

return standard(dev, iwr, cmd, info,

&iw\_handler\_get\_iwstats);

if (cmd == SIOCGIWPRIV && dev->wireless\_handlers) //获取ioctl命令列表

return standard(dev, iwr, cmd, info,

&iw\_handler\_get\_private);

/\* New driver API : try to find the handler \*/

handler = get\_handler(dev, cmd); //获取iw ioctl命令的处理函数

if (handler) {

/\* Standard and private are not the same \*/

if (cmd < SIOCIWFIRSTPRIV) // SIOCIWFIRSTPRIV:0x8BE0

return standard(dev, iwr, cmd, info, handler);

else

return private(dev, iwr, cmd, info, handler);

}

/\* Old driver API : call driver ioctl handler \*/

if (dev->netdev\_ops->ndo\_do\_ioctl)

return dev->netdev\_ops->ndo\_do\_ioctl(dev, ifr, cmd);

}

----🡪 iw\_handler\_get\_private(){

memcpy(extra, dev->wireless\_handlers->private\_args,

sizeof(struct iw\_priv\_args) \* wrqu->data.length);

}

### 驱动Ath\_wlan

dev->wireless\_handlers是哪来的呢？

\_\_ath\_attach //

---🡪 ath\_iw\_attach(dev){

dev->wireless\_handlers = &ath\_iw\_handler\_def;

}

#ifdef ATH\_SUPPORT\_HTC

#else

static struct iw\_handler\_def ath\_iw\_handler\_def = {

.standard = (iw\_handler \*) NULL,

.num\_standard = 0,

.private = (iw\_handler \*) ath\_iw\_priv\_handlers,

.num\_private = TABLE\_SIZE(ath\_iw\_priv\_handlers),

.private\_args = (struct iw\_priv\_args \*) ath\_iw\_priv\_args,

.num\_private\_args = TABLE\_SIZE(ath\_iw\_priv\_args),

.get\_wireless\_stats = NULL,

};

#endif

ath\_iw\_priv\_args 即为ioctl参数列表。

## 下发真正的命令

### kochab ioctl(skfd, cmd->cmd, &wrq)

### 内核

同样先到---🡪 **wireless\_process\_ioctl**(){

dev = \_\_dev\_get\_by\_name(net, ifr->ifr\_name))//获取接口对应的dev

if (cmd == SIOCGIWSTATS)

return standard(dev, iwr, cmd, info,

&iw\_handler\_get\_iwstats);

if (cmd == SIOCGIWPRIV && dev->wireless\_handlers) //获取ioctl命令列表

return standard(dev, iwr, cmd, info,

&iw\_handler\_get\_private);

/\* New driver API : try to find the handler \*/

handler = get\_handler(dev, cmd); //获取iw ioctl命令的处理函数

if (handler) {

/\* Standard and private are not the same \*/

if (cmd < SIOCIWFIRSTPRIV) // SIOCIWFIRSTPRIV:0x8BE0

return standard(dev, iwr, cmd, info, handler);

else

return private(dev, iwr, cmd, info, handler);

}

/\* Old driver API : call driver ioctl handler \*/

if (dev->netdev\_ops->ndo\_do\_ioctl)

return dev->netdev\_ops->ndo\_do\_ioctl(dev, ifr, cmd);

}

----🡪 get\_handler (){

/\* Try as a private command \*/

index = cmd - SIOCIWFIRSTPRIV;

if (index < dev->wireless\_handlers->num\_private)

return dev->wireless\_handlers->private[index];

}

----🡪（private）//int ioctl\_private\_call()//见驱动

### 驱动Ath\_wlan

还是这个结构体

#ifdef ATH\_SUPPORT\_HTC

#else

static struct iw\_handler\_def ath\_iw\_handler\_def = {

.standard = (iw\_handler \*) NULL,

.num\_standard = 0,

.private = (iw\_handler \*) **ath\_iw\_priv\_handlers**,

.num\_private = TABLE\_SIZE(ath\_iw\_priv\_handlers),

.private\_args = (struct iw\_priv\_args \*) ath\_iw\_priv\_args,

.num\_private\_args = TABLE\_SIZE(ath\_iw\_priv\_args),

.get\_wireless\_stats = NULL,

};

#endif

static const iw\_handler **ath\_iw\_priv\_handlers**[] = {

(iw\_handler) ath\_iw\_setparam, /\* SIOCWFIRSTPRIV+0 \*/

(iw\_handler) ath\_iw\_getparam, /\* SIOCWFIRSTPRIV+1 \*/

(iw\_handler) ath\_iw\_setcountry, /\* SIOCWFIRSTPRIV+2 \*/

(iw\_handler) ath\_iw\_getcountry, /\* SIOCWFIRSTPRIV+3 \*/

(iw\_handler) ath\_iw\_sethwaddr, /\* SIOCWFIRSTPRIV+4 \*/

(iw\_handler) ath\_iw\_gethwaddr, /\* SIOCWFIRSTPRIV+5 \*/

/\* begin: liurq@BHU, 2013-3-4 \*/

(iw\_handler) ath\_iw\_setparam, /\* SIOCWFIRSTPRIV+6 \*/

(iw\_handler) ath\_iw\_getparam, /\* SIOCWFIRSTPRIV+7 \*/

/\* end: liurq@BHU, 2013-3-4 \*/

};

struct iwreq

{

union

{

char ifrn\_name[IFNAMSIZ]; /\* if name, e.g. "eth0" \*/

} ifr\_ifrn;

/\* Data part (defined just above) \*/

union iwreq\_data u;

};

union iwreq\_data

{

/\* Config - generic \*/

char name[IFNAMSIZ];

/\* Name : used to verify the presence of wireless extensions.

\* Name of the protocol/provider... \*/

struct iw\_point essid; /\* Extended network name \*/

struct iw\_param nwid; /\* network id (or domain - the cell)

struct iw\_freq freq; /\* frequency or channel :

\* 0-1000 = channel

\* > 1000 = frequency in Hz \*/

struct iw\_param sens; /\* signal level threshold \*/

struct iw\_param bitrate; /\* default bit rate \*/

struct iw\_param txpower; /\* default transmit power \*/

struct iw\_param rts; /\* RTS threshold threshold \*/

struct iw\_param frag; /\* Fragmentation threshold \*/

\_\_u32 mode; /\* Operation mode \*/

struct iw\_param retry; /\* Retry limits & lifetime \*/

struct iw\_point encoding; /\* Encoding stuff : tokens \*/

struct iw\_param power; /\* PM duration/timeout \*/

struct iw\_quality qual; /\* Quality part of statistics \*/

struct sockaddr ap\_addr; /\* Access point address \*/

struct sockaddr addr; /\* Destination address (hw/mac) \*/

struct iw\_param param; /\* Other small parameters \*/

struct iw\_point data; /\* Other large parameters \*/

};

()

static int ioctl\_private\_call(struct net\_device \*dev, struct iwreq \*iwr,

unsigned int cmd, struct iw\_request\_info \*info,

iw\_handler handler)

{

int extra\_size = 0, ret = -EINVAL;

const struct iw\_priv\_args \*descr;

extra\_size = get\_priv\_descr\_and\_size(dev, cmd, &descr);

/\* Check if we have a pointer to user space data or not. \*/

if (extra\_size == 0) {

/\* No extra arguments. Trivial to handle \*/

ret = handler(dev, info, &(iwr->u), (char \*) &(iwr->u));

} else {

ret = ioctl\_private\_iw\_point(&iwr->u.data, cmd, descr,

handler, dev, info, extra\_size);

}

/\* Call commit handler if needed and defined \*/

if (ret == -EIWCOMMIT)

ret = call\_commit\_handler(dev);

return ret;

}

int

dl\_iwpriv\_set(const char \*ifname, const char \*cmdname, const char \*args, int count, int sz)

{

…

u8 buff[1024];

if(offset) {

((u32 \*) buff)[0] = subcmd->cmd;

}

memcpy(buff+offset, args, sz\*count<1024 ? sz\*count : 1024);

wrq.u.data.pointer = (caddr\_t) buff;

wrq.u.data.flags = cmd->cmd;

…

}

(handler(dev, info, &(iwr->u), (char \*) &(iwr->u));)

static int **ath\_iw\_setparam**(struct net\_device \*dev,

struct iw\_request\_info \*info,

void \*w,

char \*extra)

{

int \*i = (int \*) extra;

if (param & ATH\_PARAM\_SHIFT){

param -= ATH\_PARAM\_SHIFT;

retval = scn->sc\_ops->ath\_set\_config\_param(scn->sc\_dev,

(ath\_param\_ID\_t)param,

&(i[1]));//&value); //liurq@BHU, 2013-3-4

}

}

{ **ATH\_PARAM\_TXPOWER\_LIMIT2G** | ATH\_PARAM\_SHIFT,

IW\_PRIV\_TYPE\_INT | IW\_PRIV\_SIZE\_FIXED | 1, 0, "TXPowLim2G" },

static const struct ath\_ops ath\_ar\_ops = {

…

ath\_set\_config // sc\_ops->ath\_set\_config\_param

…

}

int

ath\_set\_config(ath\_dev\_t dev, ath\_param\_ID\_t ID, void \*buff)

{

case **ATH\_PARAM\_TXPOWER\_LIMIT2G**:

if (\*(int \*)buff > ATH\_TXPOWER\_MAX\_2G) {

retval = -1;

} else {

**sc->sc\_config.txpowlimit2G = \*(int \*)buff;**

}

}

struct ifreq \*ifr, 🡸🡺 struct iwreq \*iwr

struct iw\_point

{

**void \_\_user \*pointer; /\* Pointer to the data (in user space) \*/**

\_\_u16 length; /\* number of fields or size in bytes \*/

\_\_u16 flags; /\* Optional params \*/

};

# 创建VAP

## Kochab

char \*

dl\_wfvap\_create\_vap (char \*radio, const char \*name, char mode)

{

struct ieee80211\_clone\_params cp;

struct ifreq ifr;

…

switch(mode) {

case 1:/\* **AP** \*/

cp.icp\_opmode = IEEE80211\_M\_HOSTAP;

cp.icp\_flags = IEEE80211\_CLONE\_BSSID;

break;

case 2:/\* **STA** \*/

cp.icp\_opmode = IEEE80211\_M\_STA;

cp.icp\_flags = IEEE80211\_CLONE\_BSSID;

break;

…

strncpy(ifr.ifr\_name, radio, IFNAMSIZ);

strncpy(cp.icp\_name, name, IFNAMSIZ);

ifr.**ifr\_data** = (void \*) &cp;

ioctl(skfd, SIOC80211IFCREATE, &ifr)

}

## 内核

先到---🡪 **wireless\_process\_ioctl**()

{

struct net\_device \*dev;

dev = \_\_dev\_get\_by\_name(net, ifr->**ifr\_name**)) /\* namePCI设备名： wifi0 \*/

if (dev->netdev\_ops->ndo\_do\_ioctl)

return dev->netdev\_ops->ndo\_do\_ioctl(dev, ifr, cmd);

}

static const struct net\_device\_ops athdev\_net\_ops = {

.ndo\_open = ath\_netdev\_open,

.ndo\_stop = ath\_netdev\_stop,

.ndo\_start\_xmit = ath\_netdev\_hardstart,

.ndo\_set\_mac\_address = ath\_netdev\_set\_macaddr,

.ndo\_tx\_timeout = ath\_netdev\_tx\_timeout,

.ndo\_get\_stats = ath\_getstats,

.ndo\_change\_mtu = ath\_change\_mtu,

.ndo\_set\_multicast\_list = ath\_netdev\_set\_mcast\_list,

**.ndo\_do\_ioctl = ath\_ioctl,**

};

## 驱动

static int

ath\_ioctl(struct net\_device \*dev, struct ifreq \*ifr, int cmd)

{ switch (cmd){

。。。

case SIOC80211IFCREATE:

error = osif\_ioctl\_create\_vap(dev, ifr, scn->sc\_osdev);

。。。

}

}

---🡪

int

osif\_ioctl\_create\_vap(struct net\_device \*comdev, struct ifreq \*ifr, osdev\_t os\_handle)

{

struct ieee80211\_clone\_params cp;

wlan\_dev\_t devhandle = ath\_netdev\_priv(comdev);

ifc\_name2unit(cp.icp\_name, &unit); /\* wlan1 🡺 1 \*/

**\_copy\_from\_user(&cp, ifr->ifr\_data, sizeof(cp))**

strncpy(name, cp.icp\_name, sizeof(name));

// Allocate net device for this network interface （for vap like : **wlan0**）

dev = alloc\_netdev(sizeof(osif\_dev), name, ether\_setup);

if (cp.icp\_opmode == IEEE80211\_M\_HOSTAP) {

scan\_priority\_mapping\_base = DEF\_VAP\_SCAN\_PRI\_MAP\_OPMODE\_AP\_BASE;

}

vap = **wlan\_vap\_create**(**devhandle**, cp.icp\_opmode, scan\_priority\_mapping\_base, cp.icp\_flags, cp.icp\_bssid);

**osifp->****os\_if = vap;**

osifp->os\_handle = os\_handle;

osifp->os\_devhandle = devhandle;

osifp->os\_comdev = comdev;

osifp->os\_opmode = cp.icp\_opmode;

osifp->os\_unit = unit;

osif\_vap\_setup(vap, dev, cp.icp\_opmode); /\* ? 各种event注册，状态机初始化 ? \*/

#if LINUX\_VERSION\_CODE > KERNEL\_VERSION(2,6,30)

dev->netdev\_ops = &osif\_dev\_ops;

register\_netdevice(dev);

}

------🡪

wlan\_if\_t

**wlan\_vap\_create**(wlan\_dev\_t **devhandle**,

enum ieee80211\_opmode opmode,

int scan\_priority\_base,

int flags,

u\_int8\_t \*bssid)

{

**struct ieee80211vap \*vap;**

struct ieee80211com \*ic = devhandle;

vap = ic->ic\_vap\_create(ic, opmode, scan\_priority\_base, flags, bssid);

//ath\_vap\_create

TAILQ\_INSERT\_TAIL(&ic->ic\_vaps, vap, iv\_next);

}

---🡪

static struct ieee80211vap \*

ath\_vap\_create(struct ieee80211com \*ic,

int opmode,

int scan\_priority\_base,

int flags,

const u\_int8\_t bssid[IEEE80211\_ADDR\_LEN])

{

struct ath\_vap\_net80211 \*avn;

ic\_opmode = IEEE80211\_M\_HOSTAP;

avn = (struct ath\_vap\_net80211 \*)OS\_ALLOC\_VAP(scn->sc\_osdev,

sizeof(struct ath\_vap\_net80211));

avn->av\_sc = scn;

vap = &avn->av\_vap;

scn->**sc\_ops**->add\_interface(scn->sc\_dev, id, vap, ic\_opmode, ath\_opmode, nostabeacons) // ath\_vap\_attach

ieee80211\_vap\_setup(); //配置VAP，

}

static int

ath\_vap\_attach(ath\_dev\_t dev, int if\_id, ieee80211\_if\_t if\_data, HAL\_OPMODE opmode, HAL\_OPMODE iv\_opmode, int nostabeacons)

{

struct ath\_softc \*sc;

avp = (struct ath\_vap \*)OS\_MALLOC(sc->sc\_osdev, sizeof(struct ath\_vap), GFP\_KERNEL);

avp->av\_if\_data = if\_data;

sc->sc\_vaps[if\_id] = avp;

}

# 创建PCI设备

static int \_\_init init\_ath\_pci(void){

pci\_register\_driver(&ath\_pci\_drv\_id)

}

struct pci\_driver ath\_pci\_drv\_id = {

.name = "ath\_pci",

.id\_table = ath\_pci\_id\_table,

.probe = ath\_pci\_probe,

.remove = ath\_pci\_remove,

#ifdef ATH\_BUS\_PM

.suspend = ath\_pci\_suspend,

.resume = ath\_pci\_resume,

#endif /\* ATH\_BUS\_PM \*/

/\* Linux 2.4.6 has save\_state and enable\_wake that are not used here \*/

};

static int

ath\_pci\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*id)

{

struct **net\_device** \*dev;

struct ath\_pci\_softc \*sc；

/\* 一些PCI设备的的注册 \*/

。。。。。。

dev = alloc\_netdev(sizeof(struct ath\_pci\_softc), "**wifi%d**", ether\_setup);

sc = ath\_netdev\_priv(dev);

sc->aps\_osdev.netdev = dev;

sc->aps\_osdev.bdev = (void \*) pdev;

\_ath\_attach(id->device, dev, &bus\_context, &sc->**aps\_osdev**)

}

\_\_ath\_attach(u\_int16\_t devid, struct net\_device \*dev, HAL\_BUS\_CONTEXT \*bus\_context, **osdev\_t osdev**)

{

ath\_attach(devid, bus\_context, scn, **osdev**, &ath\_params, &hal\_conf\_parm, &wlan\_reg\_params);

dev->netdev\_ops = &**athdev\_net\_ops**; /\* 在kernel里面用到了呀 \*/

}

static const struct net\_device\_ops athdev\_net\_ops = {

.ndo\_open = ath\_netdev\_open,

.ndo\_stop = ath\_netdev\_stop,

.ndo\_start\_xmit = ath\_netdev\_hardstart,

.ndo\_set\_mac\_address = ath\_netdev\_set\_macaddr,

.ndo\_tx\_timeout = ath\_netdev\_tx\_timeout,

.ndo\_get\_stats = ath\_getstats,

.ndo\_change\_mtu = ath\_change\_mtu,

.ndo\_set\_multicast\_list = ath\_netdev\_set\_mcast\_list,

**.ndo\_do\_ioctl = ath\_ioctl,**

};

int

ath\_attach(u\_int16\_t devid, void \*base\_addr,

**struct ath\_softc\_net80211 \*scn**,

**osdev\_t osdev**, struct ath\_reg\_parm \*ath\_conf\_parm,

struct hal\_reg\_parm \*hal\_conf\_parm, IEEE80211\_REG\_PARAMETERS \*ieee80211\_conf\_parm)

{

**struct ieee80211com \*ic;**

scn->sc\_osdev = osdev;

ic = &scn->sc\_ic;

ic->ic\_osdev = osdev;

**ic->ic\_vap\_create = ath\_vap\_create;**

ic->ic\_vap\_delete = ath\_vap\_delete;

/\*

\* Create an Atheros Device object

\*/

ath\_dev\_attach(devid, base\_addr,

ic, &**net80211\_ops**, **osdev**,

&**scn->sc\_dev**, **&scn->sc\_ops**,

scn->amem.handle,

ath\_conf\_parm, hal\_conf\_parm);

/\* attach channel width management \*/

error = ath\_cwm\_attach(scn, ath\_conf\_parm);

/\*

\* Set the Atheros Advanced Capabilities from station config before

\* starting 802.11 state machine.

\* 这个东西好像很厉害的样子

\*/

ieee80211com\_set\_athcap(ic, (ops->have\_capability(dev, ATH\_CAP\_BURST) ? IEEE80211\_ATHC\_BURST : 0));

}

int ath\_dev\_attach(u\_int16\_t devid,

void \*bus\_context,

ieee80211\_handle\_t ieee,

struct ieee80211\_ops \*ieee\_ops,

osdev\_t **osdev**,

**ath\_dev\_t \*dev**,

**struct ath\_ops \*\*ops,**

asf\_amem\_instance\_handle amem\_handle,

struct ath\_reg\_parm \*ath\_conf\_parm,

struct hal\_reg\_parm \*hal\_conf\_parm)

{

struct ath\_softc \*sc = NULL;

sc = (struct ath\_softc \*)OS\_MALLOC(osdev, sizeof(struct ath\_softc), GFP\_KERNEL);

**\*dev = (ath\_dev\_t)sc;**

sc->sc\_ath\_ops = ath\_ar\_ops;

**\*ops = &sc->sc\_ath\_ops;**

**sc->sc\_osdev = osdev;**

sc->sc\_ieee = ieee;

sc->sc\_ieee\_ops = ieee\_ops;

ah = \_ath\_hal\_attach(devid, osdev, sc, bus\_context, hal\_conf\_parm,

amem\_handle, &halCallbackTable, &status);

sc->sc\_ah = ah;

}

\_ath\_hal\_attach(u\_int16\_t devid,

HAL\_ADAPTER\_HANDLE osdev,

HAL\_SOFTC sc,

HAL\_BUS\_CONTEXT \*bus\_context,

struct hal\_reg\_parm \*hal\_conf\_parm,

asf\_amem\_instance\_handle amem\_handle,

const struct ath\_hal\_callback \*callbackTable,

void \*s)

{

ah = ath\_hal\_attach(

devid, osdev, sc,

bus\_context->bc\_info.bc\_tag, bus\_context->bc\_handle, bus\_context->bc\_bustype,

amem\_handle, hal\_conf\_parm, &status);

}

struct ath\_hal\* \_\_ahdecl

ath\_hal\_attach(u\_int16\_t devid, HAL\_ADAPTER\_HANDLE osdev, HAL\_SOFTC sc,

HAL\_BUS\_TAG st, HAL\_BUS\_HANDLE sh, HAL\_BUS\_TYPE bustype,

asf\_amem\_instance\_handle amem\_handle,

struct hal\_reg\_parm \*hal\_conf\_parm, HAL\_STATUS \*error)

{

case AR9300\_DEVID\_AR9340:

ah = ar9300Attach(devid, osdev, sc, st, sh, bustype, amem\_handle,

hal\_conf\_parm, error);

}

ar9300Attach(u\_int16\_t devid, HAL\_ADAPTER\_HANDLE osdev, HAL\_SOFTC sc,

HAL\_BUS\_TAG st, HAL\_BUS\_HANDLE sh, HAL\_BUS\_TYPE bustype,

asf\_amem\_instance\_handle amem\_handle,

struct hal\_reg\_parm \*hal\_conf\_parm, HAL\_STATUS \*status)

{

struct ath\_hal\_9300 \*ahp;

struct ath\_hal \*ah;

struct ath\_hal\_private \*ahpriv;

ahp = ar9300NewState(

devid, osdev, sc, st, sh, bustype, amem\_handle, hal\_conf\_parm, status);

ah = &ahp->ah\_priv.priv.h;

/\* 下面大概是在初始化wifi芯片 和各种结构体 \*/

if (!ar9300SetResetReg(ah, **HAL\_RESET\_POWER\_ON**)) { /\* reset chip \*/

if (!ar9300SetPowerMode(ah, HAL\_PM\_AWAKE, AH\_TRUE)) {

}

struct ath\_hal\_9300 \*

ar9300NewState(u\_int16\_t devid, HAL\_ADAPTER\_HANDLE osdev, HAL\_SOFTC sc,

HAL\_BUS\_TAG st, HAL\_BUS\_HANDLE sh, HAL\_BUS\_TYPE bustype,

asf\_amem\_instance\_handle amem\_handle,

struct hal\_reg\_parm \*hal\_conf\_parm, HAL\_STATUS \*status)

{

struct ath\_hal\_9300 \*ahp;

struct ath\_hal \*ah;

ahp = amalloc\_adv(

amem\_handle, sizeof(struct ath\_hal\_9300), adf\_os\_mem\_zero\_outline);

**ah = &ahp->****ah\_priv.priv.h;**

OS\_MEMCPY(&ahp->ah\_priv.priv, &ar9300hal, sizeof(ahp->ah\_priv.priv));

ah->ah\_osdev = osdev;

ah->ah\_sc = sc;

return ahp;

}