**AR6002 WLAN 모듈 구조 분석.**

작 성 자: 김정호

작 성 일: 2012년 2월 16일

목차

[**1. AR6002 드라이버 Loading 과정.** 4](#_Toc318278095)

[**1.1 하드웨어 연결 구조.** 4](#_Toc318278096)

[**1.2 모듈 Loading 과정.** 4](#_Toc318278097)

[**2. 소프트웨어 Stack** 6](#_Toc318278098)

[**2.1 Host System과의 연동 구조.** 6](#_Toc318278099)

[**2.2 AR6002 디바이스 드라이버 S/W 구조.** 7](#_Toc318278100)

[**3. AR6002 Chipset** 9](#_Toc318278101)

[**3.1 Block Diagram** 9](#_Toc318278102)

[**4. AR6000 모듈과의 데이터 송신 및 수신.** 11](#_Toc318278103)

[**5. AR6000 모듈 Initialize 과정.** 12](#_Toc318278104)

[**5.1 AR6000\_init\_module().** 13](#_Toc318278105)

[**5.2 HIFInit()** 13](#_Toc318278106)

[**5.3 hifDeviceInserted().** 13](#_Toc318278107)

[**5.4 startup\_task** 13](#_Toc318278108)

[**5.5 ar6000\_avail\_ev()** 13](#_Toc318278109)

[**5.6 ar6000\_init()** 13](#_Toc318278110)

[**6. Scan 과 Roaming** 14](#_Toc318278111)

[**6.1 Scan 모드.** 14](#_Toc318278112)

[**6.2 Scan 방법.** 14](#_Toc318278113)

[**6.3 Roaming.** 16](#_Toc318278114)

[**7. AR6002 리눅스에서 Windows CE로 포팅.** 17](#_Toc318278115)

[**7.1 리눅스 소스 폴트** 17](#_Toc318278116)

[**7.2 포팅 순서** 18](#_Toc318278117)

**서문**

WIFI 모듈 디바이스 드라이버 개발을 목표로 프로젝트를 시작 하면서 WLAN 모듈을 알아 보았는데 시중에 많이 나와 있는 제품은 크게 Marvell, Broadcom, Atheros 사의 제품 이었다. 이 세 회사의 제품 중 가장 쉽게 접근할 수 있는 제품이 AR6002 칩 셋 이었다. 인터넷을 통해 리눅스 뿐 아니라 WindowsCE6.0 드라이버 소스도 구할 수 있었다. 그래서 드라이버 개발 모델을 AR6002로 정하고 개발을 조사를 시작 하였다. 우선 이 문서에서는 리눅스의 AR6002 WLAN 디바이스 드라이버의 구조를 분석한다. AR6002 칩 셋의 H/W 연동 방법과 네트워크 드라이버에 등록되는 과정, 그리고 supplicant를 통해 어떻게 제어 되는지를 조사 할 것이다. Supplicant는 WIFI가 장착된 단말기가 무선 AP에 접속할 수 있도록 도와주는 툴 이라고 보는 것이 맞을 것이다. 무선 네트워크 디바이스 드라이버가 하는 가장 중요한 일이 주변의 사용 가능한 AP를 찾고 그 AP중 특정 AP를 선택하여 네트워크에 접속 가능하게 하는 것이다. 드라이버 대부분의 기능은 AP를 찾고 특정 AP와 연결하고 그리고 연결된 AP와의 상태를 주기적으로 확인 하는데 중점을 둔다. 따라서 이 문서도 그 부분에 중점을 둘 것이다.

# **1. AR6002 드라이버 Loading 과정.**

## **1.1 하드웨어 연결 구조.**

Power

MMC

Detect

**PXA320**

**AR6000**

**WLAN 모듈**

Reset

그림 1 H/W 연결 구조

AR6000 모듈은 Host 보드와 SDIO Interface(MMC 라인)로 연결이 되어 있다. 그리고 모듈의 전원 On/Off를 담당하는 Power와 Reset, detect핀이 제공된다. 소프트웨어 적으로는 Host가 사용하는 드라이버와 모듈을 구동하기 위한 Firmware가 제공이 되어 진다.

## **1.2 모듈 Loading 과정.**

여기서는 안드로이드 시스템이 올라가 있는 Host를 대상으로 드라이버 Loading 과정을 설명 한다.

**Insmod ar6000.ko 실행**

**Ar6000\_init\_module()실행**

**HIF\_init() 실행**

**SDIO 드라이버에 ar6000 모듈 디바이스 등록**

**Ar6000 모듈 초기화 실행(async\_task, start\_task 실행)**

**네트워크 드라이버에 ar6000 모듈 디바이스 등록**

**모듈 Firmware 다운로드 실행.**

**WMI\_Init(), HTC\_Start() 실행.**

그림 2 모듈 Loading 과정

그림 2와 같이 안드로이드 시스템에서는 WLAN 모듈을 Loading 하기 위해 드라이버 파일을 insmod 명령을 통해 Loading을 한다. Insmod 명령을 주면 시스템은 ar6000\_init\_module()을 부르게 되며 모듈 Loading이 시작 된다.

AR6002 모듈이 SDIO interface로 연결이 되어 있으므로 먼저 SDIO 드라이버에 디바이스 등록을 하게 된다. 그런 다음 모듈의 H/W적인 초기화를 실행한 후 네트워크 드라이버에 다시 디바이스 등록을 하게 된다. 여기서 SDIO의 디바이스 등록과 네트워크 디바이스 등록은 구분 되어 져야 한다. SDIO 디바이스 등록은 Host와의 통신을 위한 H/W연결을 의미 하고 네트워크 디바이스 등록은 안드로이드의 네트워크 시스템과의 연동을 위해 네트워크 드라이버에 디바이스를 등록하는 과정이다. WLAN도 네트워크 디바이스이므로 당연히 네트워크 드라이버에 등록이 되어져야 한다. 네트워크 디바이스 등록이 완료 되면 모듈을 구동시킬 Firmware를 SDIO를 통해 모듈에 전달 한다. 이 Firmware로 모듈이 구동되게 되며 Host에서 전달되는 명령어를 모듈 내부에서 처리하게 된다. Firmware 전달이 완료되면 WMI\_Init()과 HTC\_Start()를 실행하게 된다. WMI는 시스템의 어플리케이션에서 모듈을 제어하기 위해 접근하는 계층으로 socket 통신의 ioctl로 제어가 된다. HTC는 WMI와 H/W를 연동시켜 주는 계층이다.

위의 그림 2에 나와 있는 Loading 과정은 안드로이드 시스템의 경우를 나타낸 것이며 Windows 환경에서는 조금 다를 것이다. 다만 전체적인 Loading 과정은 다르지 않을 것이다. 또한 WMI 및 HTC 는 동일한 것을 사용한다.

# **2. 소프트웨어 Stack**

## **2.1 Host System과의 연동 구조.**

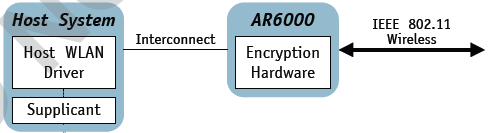
****

그림 3 Host 시스템과의 연동 구조

그림 3에 WLAN S/W 구조가 나타나 있다. Host System은 SDIO Interface를 통해 모듈과 연결이 되어 있고 WLAN 드라이버를 이용하여 모듈을 제어 한다. 모듈은 Host의 제어에 의해 IEEE 802.11 규격의 무선 통신을 하게 된다.

■ Supplicant

WLAN 모듈을 가지고 있는 단말이 무선 AP에 접속하는 것을 도와주는 일종의 Tool 이라고 보면 된다. 사용 가능한 AP를 찾고 특정 AP에 접속하기 위해 인증 번호를 입력하고 하는 등의 일에 이 Tool을 사용한다.

■ Host WLAN Driver

WLAN 모듈을 제어 하는 역할을 하는 드라이버 이다. SDIO interface를 이용하는 모듈의 경우 SDIO 드라이버에 디바이스를 등록하고 또 한 네트워크 시스템과 연동하기 위해 네트워크 드라이버에 디바이스를 등록한다. Supplicant 로부터 받은 명령을 해석하여 AR6000 모듈을 제어 하는 역할도 담당한다.

■ AR6000 Firmware

AR6000 모듈을 구동하는 역할을 하며 supplicant로부터 받은 명령을 처리 한다. Firmware의 경우 칩 제조사에서 제공하여 내부 동작은 알 수 없다.

무선 네트워크 시스템의 기본 동작은 Scan, Probe, Join, Associate 순으로 이루어 진다. 우선 단말이 주변에 사용 가능한 무선 AP(Access Point)가 있는지부터 확인하기 위해 Scan 동작을 하고, 사용 가능한 AP들중 원하는 AP를 선택한다. AP를 선택한다고 해서 네트워크에 연결이 되는 것은 아니다. 선택한 AP가 인증번호가 요구한다면 인증번호를 입력해야 한다. 무선 네트워크에서 가장 중요시 되는 부분이 보안인데, 모든 사용자가 아닌 허용된 사용자만이 AP를 사용할 수 있게 하기 위해서 이다. 또 한 해킹으로부터 네트워크 망을 보호하기 위해서 이기도 하다.

## **2.2 AR6002 디바이스 드라이버 S/W 구조.**

**Wireless Application**

Wireless Device Driver

Wireless Module Interface (WMI)

Host/Target Communication (HTC)

**Wireless Application**

Wireless Module Interface (WMI)

그림 4. AR6000 모듈 Software Stack

■ **Wireless Application.**

윈도우의 NDIS 드라이버 또는 리눅스의 wpa\_supplicant를 지칭한다. 단말기가 AP를 Scan하고 접속하는데 도움을 준다.

■ **Wireless Module Interface (WMI)**

Wireless Application이 부르는 계층으로 네트워크 컨트롤 메시지를 H/W에 전달 하는 역할을 한다. 이 Stack은 HTC(Host Target Communication)과 연결 되어 있으며 application으로 부터 전달 받은 메시지를 H/W에 전달하는 역할을 한다.

모듈 제어를 위한 일종의 API를 제공하는 부분이라 볼 수 있다. 이 부분은 H/W와는 상관 없는 구조로 되어 있으며 모듈을 제어하기 위한 기능들을 제공한다. Wpa\_supplicant의 경우 socket을 이용하여 이 부분과 접속이 되며 socket의 ioctl을 이용하여 명령어를 주고 받는다. H/W와의 연동을 위해 하위 계층으로 HTC를 가지고 있다.

■ **Host Target Communication (HTC)**

Hard Ware 제어 부분으로 HIF(Hardware Interface) Layer와 연결되어 있으며 네트워크 디바이스 드라이버를 구동하는 부분이다. 모듈에서 보면 데이터를 송수신하기 위한 Hardware 부분 이라고 할 수 있다. AR6002는 MMC 디바이스 드라이버를 사용하므로 이 부분에 MMC 디바이스 제어 부분이 있다고 보면 된다.

WPA\_Supplicant

WMI

HTC

HIF

Linux Application

AR6000 모듈

Wireless network AP

그림 5. 모듈 소프트웨어 구성

\* 그림 2는 안드로이드에서의 모듈 소프트웨어 Stack을 나타낸 것이다. Wpa\_supplicant이라는 모듈제어 lib에서 모듈 제어를 담당하고 여기에 맞게 WMI가 구성이 된다. WMI는 application으로부터 전달 받은 메시지를 HTC에 전달 하는데 WMI는 네트워크 제어 관련 메시지를 알고 있으나 HTC는 메지기 전달 역할만을 하기 때문에 메시지 내용을 알 필요가 없다. 단지 Hardware 컨트롤 부분만 관여 할 뿐이다. 리눅스에서는 이 부분이 네트워크카드 하드웨어 제어 부분이라 할 수 있다. 그리고 실질적인 모듈과 통신하는 부분이 HIF(Hardware Interface)이다. 이 부분은 AR6000 모듈이 MMC 디바이스를 통하여 통신 하므로 MMC 디바이스 제어 부분이 들어 있다.

# **3. AR6002 Chipset**

## **3.1 Block Diagram**

그림 6. AR6002 Block Diagram

■ AR6002 Feature.

- All-CMOS IEEE 802.11a/b/g or 802.11b/g single chip client.

- Integrated PA, LNA and RF switch minimizing external component count.

- Data rates of 1-54Mbps for 802.11g, 6-54Mbps for 802.11a

- Host interface support for SDIO and GSPI.

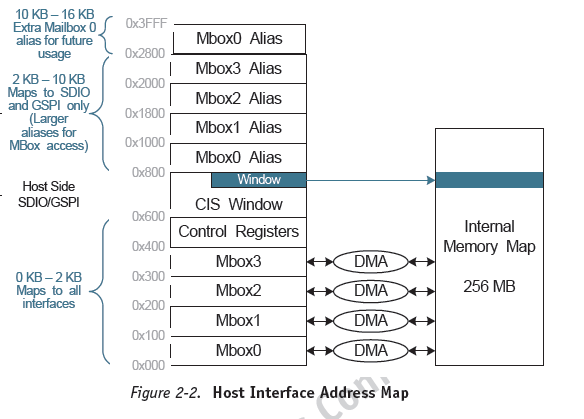
- Security support for WPS, WPA2, WPA, WAPI and protected management frames

- Support for 2.4 and 5 GHz operation in all available band in all regulatory domains.

- Full 802.11e Qos support including WMM and U-APSD.

- SPI or I2C for EEPROM support

그림 6은 AR6002의 블록도 이다. AR6002 모듈의 core이며 주변 에 EEPROM 등의 회로가 추가 된다. 위 그림에서 Host Interface로 SDIO와 GSPI가 있는데 우리는 SDIO를 사용할 것이다. 그리고 Host 와 연결되어 있는 Mailbox가 AR6002의 core인 Xtensa CPU와 Host의 통신을 도와줄 것이다. 칩 셋 회사에서 제공하는 Firmware가 위 시스템을 구동하는 프로그램이다.



# **4. AR6000 모듈과의 데이터 송신 및 수신.**

**AR6002 모듈**

**WMI Layer**

**HTC Layer**

**HIF Layer (Send Packet using the SDIO Interface)**

**Mail Box로 데이터 송 수신**

**데이터 처리**

IOCTL을 이용하여 명령어 전달

H/W와의 연동을 위해 HTC 포맷으로 변경

HTC 패킷으로 변경 후 데이터 전송 준비

SDIO Interface를 이용하여 데이터 통신

그림 8. AR6002 모듈 데이터 통신 흐름도

그림 8은 AR6002 모듈의 리눅스 드라이버와 모듈간의 통신 구조를 나타낸 것이다. 위의 그림은 안드로이드 시스템상의 wpa\_supplicant를 이용하여 모듈간 통신이 어떻게 이루어 지는지를 나타내었다.

■ 데이터 송/수신.

우선 wpa\_supplicant에서 ioctl을 이용하여 명령어를 보내면 드라이버의 WMI Layer에서 이 명령을 받는다. WMI Layer는 H/W와 상관 없이 구현이 되어 있다. 이는 다양한 OS 침 개발 환경을 지원하기 위해서이다. 이 WMI Layer와 H/W의 연결을 도와 주기 위해 HTC Layer가 존재 한다. WMI Layer는 받은 메시지를 H/W에 전달하기 위해 HTC Layer로 보낸다. HTC Layer에서는 모듈로 데이터를 전송하기 위해 데이터에 Header를 붙이고 HTC 패킷으로 가공한다. 이렇게 패킷 형태로 가공된 데이터는 HIF Layer에 의해 SDIO를 통해 모듈로 전송이 된다. 모듈은 H/W Mailbox를 통해 데이터를 전달 받는다. 모듈 내부에서의 데이터 처리는 칩 제조회사가 제공해준 Firmware를 통해 이루어지므로 자세한 동작 방법은 알 수가 없다.

# **5. AR6000 모듈 Initialize 과정.**

**Ar6000\_init\_module()**

- ar6000\_gpio\_init()

- HIFInit()

**HIFInit()**

- sdio\_register\_driver()

이 함수가 불려지면 struct sdio\_driver의 .probe에 등록된 함수가 불려지게 된다.

\*모듈은 SDIO Interface를 이용하여 통신한다.

**hifDeviceInserted()**

- Init SDIO 드라이버.

- async\_Task, Startup\_Task 생성 및 실행.

**Startup\_Task**

- deviceInsertedHandler() 🡪 callback 함수로 ar6000\_avail\_ev() 함수가 등록되어 있다.

**ar6000\_avail\_ev()**

- ether\_setup().

- HTCCreate().

- register\_etherdrv().

- ar6000\_download\_image()

\* 네트웍 드라이버가 등록되고 모듈 firmware가 다운로드 된다.

**AR6000\_Init**

- wmi\_init().

- HTCStart().

그림 6. AR6000 모듈 Initialize 과정.

## **5.1 AR6000\_init\_module().**

- 시스템에 리눅스 커널이 올라가 있는 상태에서 insmod를 통해 모듈을 로드하면 먼저 ar6000\_init\_module()이 불려 지게 된다. 여기서는 모듈의 GPIO를 설정하고 HIFInit()함수를 부른다.

## **5.2 HIFInit()**

- AR6000 모듈의 경우 SDIO Interface를 통해 Host와 통신을 하게 된다. 그래서 이 함수에서 SDIO 드라이버를 사용할 수 있게 디바이스 초기화 및 디바이스를 SDIO 드라이버에 등록을 한다.

## **5.3 hifDeviceInserted().**

- HIFInit()에서 sdio\_register\_driver()를 부를 때 넘겨주는 인자에 .probe = hifDeviceInserted 라고 되어 있다. 그래서 sdio 디바이스가 등록되면서 hifDeviceInserted() 함수가 불려 지게 된다. 이 함수에서는 SDIO 드라이버 초기화 및 async\_task와 start\_task를 생성한다. Async\_task는 모듈의 Read & Write 의 sync를 담당하고 start\_task는 전반적인 시스템 초기화를 담당 한다.

## **5.4 startup\_task**

- deviceInsertedHandler()함수를 실행하고 네트워크 디바이스 드라이버 초기화를 시작한다. deviceInsertedHandler()는 함수 포인터로 ar6000\_avail\_ev()와 연동이 된다.

## **5.5 ar6000\_avail\_ev()**

- 이 함수에서 네트워크 드라이버를 초기화 하고 등록한다. 네트워크 드라이버를 등록함으로써 네트워크 드라이버를 사용 가능하게 하고 모듈에 Firmware를 다운로드하여 모듈을 실행시킨다. 네트워크 디바이스 등록시에 넘겨지는 struct net\_device\_ops 에 필요 실행 함수들이 정의되어 있으며 이 실행 함수들이 실질적으로 네트워크 디바이스 제어에서 불려지는 함수들이다.

## **5.6 ar6000\_init()**

- 네트워크 디바이스 구조체의 init 함수이며 이 함수에서 WMI\_Init()이 실행되며 HTCStart()가 불려 진다. 이 함수가 실행됨으로써 무선 네트워크 디바이스가 실행 준비가 완료 되며 네트워크 어플리케이션으로부터의 명령 수행이 가능해 진다.

BMI\_Done()

ar6000\_download\_image()

wmi\_init()

ar6000\_init()

ar6000\_connect\_service()

Start Scan()

Ar6000\_rx\_refill

ar6000\_setup\_credit\_dist()

ar6000\_cookie\_init()

HTCStart()

그림 7. R6000\_init() 함수 호출 과정

# **6. Scan 과 Roaming**

## **6.1 Scan 모드.**

AR6002는 연결 상태에 따라 두 가지의 Scan 모드가 있다. 하나는 Foreground Scan 이고 다른 하나는 Background Scan 이다. Foreground Scan 모드는 현재 AR6002의 상태가 특정 AP와 연결이 되지 않은 상태일 때 수행되는 Scan이고 Background Scan은 현재 AR6002의 상태가 특정 AP와 연결이 되어 있는 상태이고 BSS의 리스트를 업데이트 하기 위해 행해진다. 즉 모듈의 상태가 AP와 연결이 되어 있지 않으면 Foreground SCAN 모드로 진입하여 사용 가능한 AP를 SCAN하고, 모듈의 상태가 AP와 연결이 되어 있다면 사용 가능한 주변의 BSS의 리스트를 업데이트 하기 위해 Background Scan을 실시한다. Wireless Network 상태에서는 주변의 AP정보가 수시로 바뀔 수 있기 때문에 단말이 AP와 연결이 되어 있다 하더라도 주기적으로 AP를 Scan하여 사용 가능한 AP의 리스트를 업데이트 하는 것이 중요 하다.

■ Foreground Scan

- 단말이 AP와 연결이 되어 있지 않는 상태에서의 Scan.

- 채널별 Scan을 하여 이용 가능한 AP를 찾는다.

■ Background Scan

- 단말이 AP와 연결되어 데이터를 주고 받는 상태에서의 Scan.

- AP 리스트를 업데이트 하는 역할을 한다.

## **6.2 Scan 방법.**

Scan 방법에는 두 가지가 있다. 하나는 Active Scan(능동 Scan) 이고, 다른 하나는 Passive Scan(수동 Scan)이다.

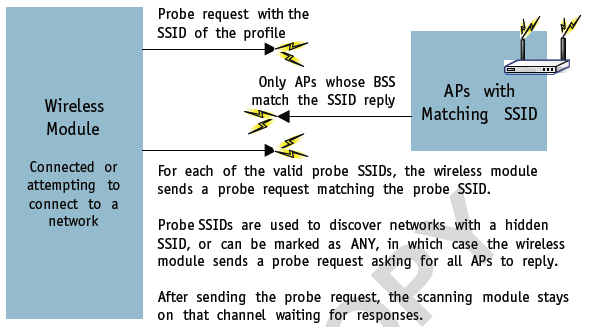
****

그림 8. Scanning

Active Scan 방법은 우선 채널 별로 Probe Request 메시지를 전송한다. 만약 해당 채널에 이용 가능한 AP가 있다면 AP는 Probe Response 메시지를 전송하여 이용 가능한 AP가 있다는 것을 알린다. 이런 방식으로 모든 채널에 대해 수행하면서 AP 리스트를 구한다.

Passive Scan 방식은 각 AP의 Beacon 메시지를 받아 AP의 리스트를 구하는 방법이다. Beacon 메시지는 AP가 약 100msec의 주기로 발송하는 메시지로 SSID 및 각종 정보들이 들어 있다. 단 말은 이 Beacon 메시지를 수신하여 AP의 리스트를 구한다.

SET\_SCAN\_PARAMS

START\_SCAN

**Linux Driver**

**AR6K Module**

**(firmware)**

BSSINFO

Scan complete event

Get Scan Result

그림 9. Scan 데이터 흐름도

■ **AR6002 Scan 순서**

- 위 그림은 AR6002 모듈의 Scan 동작 순서를 나타낸 것이다. Foreground Scan을 예로 들었다. 우선 Scan 하기 전에 Scan Parameter를 설정 한다. Scan에 필요한 시간 또는 채널 정보를 설정 한다. 그 다음 Start Scan명령을 보내 Scan을 시작 한다. Scan이 시작되면 모듈은 Scan을 시작하며, Scan이 완료 되면 Scan complete 이벤트를 보낸다. BSSINFO는 Scan을 시작한 후 모듈이 BSS를 찾으면 그 정보를 BSSINFO 이벤트를 통해 드라이버로 보내 주는 것이다. 모듈로부터 scan complete 이벤트를 받으면 Get Scan Result를 이용하여 BSS 리스트(AP 리스트)를 가져 온다.

- iwlist 툴을 이용하여 scan 명령을 내렸을 때 드라이버에서 불려지는 함수 순서.

**ar6000\_ioctl\_giwrange()**

**ar6000\_ioctl\_siwscan()**

**ar6000\_ioctl\_giwscan()**

그림 10. Scan 순서

위 그림10 에서 ar6000\_ioctl\_siwscan()함수가 최종족으로 start\_scan()을 부른다. 그리고 ar6000\_ioctl\_giwscan()함수가 scan result 값을 가져 가는 값이다.

## **6.3 Roaming.**

# **7. AR6002 리눅스에서 Windows CE로 포팅.**

## **7.1 리눅스 소스 폴트**

**- BMI**

: bmi.c

**- BTFILTER**

: btfilter\_action.c

: btfilter\_core.c

**- HIF**

: hif.c

**- HTC2**

: htc.c

: htc\_recv.c

: htc\_send.c

: htc\_service.c

**- MISCDRV**

: common\_drv.c

: credit\_dist.c

**- OS**

: ar6000.mod.c

: ar6000\_drv.c

: ar6000\_raw\_if.c

: eeprom.c

: engine.c

: ioctl.c

: netbuf.c

: wireless\_ext.c

**- WLAN**

: wlan\_node.c

: wlan\_recv\_beacon.c

: wlan\_utils.c

**- WMI**

: wmi.c

■ **BMI**

Bootloader Message Interface로 모듈 Firmware 다운로드를 담당한다.

■ **BTFILTER**

블루투스 관련 내용으로 WLAN에서는 사용하지 않는다..

■ **HIF**

Host Interface 로 SDIO를 통한 데이터 전송 Layer를 담당한다. H/W에 가장 가까운 위치에 있다. HTC로부터 받은 Packet을 SDIO Interface를 통해 모듈로 전달 한다.

■ **HTC2**

Host Target Communication으로 WMI Layer와 HIF Layer 의 중간에 위치하고 있으며 S/W stack과 H/W 간의 Interface를 담당한다. 즉 WMI Layer에서 받은 명령을 HIF로 전달 하는 역할을 하며, HIF로 전달하기 전에 모듈과 통신할 수 있게 데이터를 Packet형태로 변환한다.

■ **MISCDRV**

모듈 통신 관련 툴 함수들을 가지고 있다.

■ **OS**

리눅스 SDIO 드라이버 초기화 및 Firmware download 코드 들이 들어 있다. 그리고 HTC, WMI 초기화 호출 함수를 호출 한다. 또 Power Management 관련 코드들이 들어 있다.

■ **WLAN**

IEEE 802.11 node 제어 함수 들이 들어 있다.

■ **WMI**

Wireless Module Interface로 리눅스의 네트워크 Layer에서 호출하는 IOCTL을 제공한다. H/W와 독립적으로 구현이 되어 있으며 OS의 네트워크 계층 아래 부분에 위치한다.

## **7.2 포팅 순서**

**SDIO 디바이스 등록**

**HIF Layer 초기화 및 포팅**

**HTC Layer 초기화 및 포팅**

**모듈 Firmware 다운로드**

**네트워크 드라이버에**

**디바이스 등록**

**WMI Layer 초기화 및 포팅**

**WMI Layer IOCTL 함수 등록.**

그림 11. Windows 포팅 순서

Windows CE로의 포팅의 위의 순서대로 할 예정이다. 우선 SDIO 디바이스 등록을 통해 SDIO 드라이버를 사용 가능하게 하고 그 다음 HIF Layer를 초기화 하여 모듈과의 통신을 가능하게 한다. 다음으로 모듈에 Firmware를 다운로드하고 모듈을 구동 시킨다. 그리고 이 디바이스를 네트워크 드라이버에 등록하여 네트워크 드라이버와의 연동을 준비한다. 그 다음은 WMI Layer를 초기화 하고 HTC Layer를 구동 한다. 이러한 순서로 포팅을 진행할 것이고 포팅 과정에 있는 리눅스 API들은 Windows CE API로 교체할 것이다.

■ **SDIO 디바이스 등록**

리눅스에서는 HIFInit()함수에서 sdio\_register\_driver()함수를 호출하여 SDIO 디바이스를 등록하지만 WM(이 후로 Windows CE를 WM으로 함)에서는 Platform.reg 파일에

**"Dll"="ar6k\_ndis\_sdio.dll"**

**"Prefix"="DRG"**

을 등록하여 DRG\_Init(), DRG\_Open(), DRG\_Close()함수를 호출 하는 것으로 SDIO 디바이스를 등록한다. 이 함수들은 AR6001의 Windows CE 소스에 있으므로 이 소스들을 참조하여 포팅 작업을 진행한다.

■ **HIF Layer 초기화**.

모듈의 Read & Write 함수를 만들고 동기화를 위해 async\_task()생성하고, Start\_task()를 생성한다. Hif.c 파일을 포팅하는 작업이다. 초기화시 불려지는 함수는 hifDeviceInserted() 함수로 리눅스에서는 sdio 디바이스 등록시 불려 지는 함수이다. Windows에서는 NDIS 드라이버 초기화 과정에서 불려 진다.

**hifDeviceInserted() 호출**

**SDIO 디바이스 Configure**

**Async\_task() 생성.**

**Start\_task() 생성.**

그림 12. Windows 포팅 순서

위 그림은 SDIO 드라이버가 구동되면서 Probe함수를 통해 hifDeviceInserted()함수가 호출되어 실행되는 과정을 나타내었다. hifDeviceInserted()함수가 호출 되면 이 함수에서는 SDIO 디바이스 설정을 하고, Async\_task()와 Start\_Task()를 생성하여 모듈과 통신을 할 수 있게 준비한다. 여기서 Async\_Task()의 역할을 \_\_HIFReadWrite()함수를 호출하는 것인데, SDIO Read & Write의 동기화를 위해 사용된다.

HIF에서 HTC에 제공되는 **HIFReadWrite()**함수는 직접 SDIO를 제어 하지 않는다. 이 함수는 SDIO에 쓰여질 데이터를 버퍼에 쌓아 놓는다. 그러면 Async\_Task()에서 Buffer를 확인하여 SDIO를 통해 데이터를 내보낸다.

**\* HIF Layer에서의 주요 포팅 함수들**.

- hifDeviceInserted()

- async\_task().

- startup\_task().

- HIFReadWrite().

- \_\_HIFReadWrite();

- hifIRQHandler()

■ **HTC Layer 초기화 및 포팅**

**HTCCreate() 호출**

**Target 생성**

- AR6002 모듈 제어에 관련된 변수들을 가지고 있다.

- Mutex, Buffer, Device Info 등의 변수들을 멤버로 가지고 있다.

**Mutex 및 Packet Queue 생성**

- RX, TX Mutex 생성.

- RX, TX Buffer Queue 생성.

**DevSetup() 호출**

- callback 함수 등록

- Q-Buffer 초기화

- Get Mailbox Address.

- Configure HIF Device

그림 13. HTCCreate() 함수 내용

HTC Layer 초기화 에서는 Target을 생성하고 모듈과 통신을 하기 위한 전반적인 준비를 한다. Mutex를 생성하고, 사용할 Buffer를 생성하고, 인터럽트를 활성화 한다. HIF가 단순히 SDIO를 통해 데이터를 내보내고 받는 역할을 하는 곳이라면, 여기는 AR6002 모듈의 레지스터, 인터럽트, Mailbox등을 제어하는 역할을 하는 곳이라 보면 된다. 실질적인 모듈 제어부분이 이 곳이라 할 수 있다.

hifDeviceInserted()함수의 호출로 Start\_task()가 생성되면, Start\_task()에서는 HTCCreate() 함수를 호출 한다. HTCCreta() 함수에서는 Target 생성, Mutex 및 Buffer 생성, DevSetup() 함수를 호출 한다. 여기서 Target은 모듈 관리에 필요한 멤버 변수들을 가지고 있는데 아래와 같은 구조로 되어 있다.

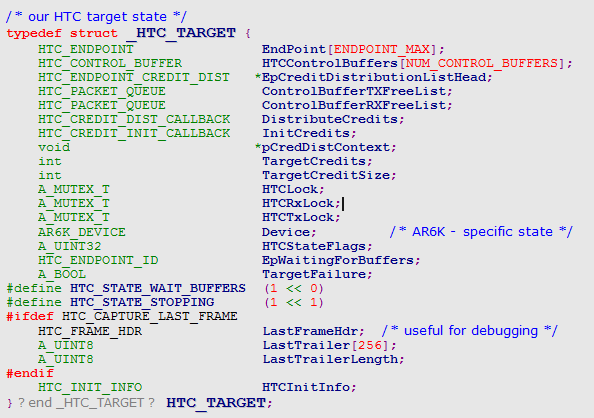


그림 14. HTC\_TARGET 구조체

위의 그림에서 볼 수 있듯이 Target 구조체는 Mutex, Buffer, Device 등의 모듈 제어 변수들을 포함하고 있다. 이 구조체에 모듈 제어에 관한 대부분의 내용이 들어 있고 이 소스에서 가장 중요한 변수 중에 하나 이다. Target 구조체를 생성하고 이 구조체 멤버에 필요한 정보를 넣는 것이 HTCCreate() 함수의 역할 이다.

DevSetup() 함수에서는 DevRWCompletionHandler()함수와, DevDsrHandler()함수를 등록한다. 그리고 Mailbox와 관련된 H/W 설정을 하고, Buffer를 초기화 한다. 이 드라이버에서 Buffer는 Linked List를 사용하여 구현 되어 있다.

**\* HTC Layer 주요 포팅 함수.**

- HTCIssueSend()

- HTCTrySend()

- HTCSendPkt()

■ **펌웨어 다운로드**.

AR6002의 경우 칩 제조사에서 AR6002를 구동하는데 필요한 Firmware 파일과 Patch 파일을 제공한다. 따라서 네트워크 드라이버를 구동하기 전에 AR6002에 Firmware와 Patch 파일을 다운로드 해야 한다. 펌웨어를 다운로드 하기 위해서는 우선 제공된 펌웨어 파일을 읽어 와야 한다. 그 다음 BMI(Bootloader Module Interface)에 제공해주는 함수들을 이용하여 펌웨어를 모듈로 전송한다.

이 부분은 모듈화가 잘 이루어져 있어 WM으로 포팅하는데 큰 어려움이 없다. 리눅스 시스템의 의존부분이 크게 없으며 HIF Layer만 포팅이 되어 있으면 어려움이 없이 진행 할 수 있다. 그리고 모듈 시작시 한번만 실행하면 된다.

■ **네트워크 드라이버에 디바이스 등록.**

네트워크 디바이스 등록은 리눅스와 WM이 다르게 구현이 되어있다. 리눅스에서는 ether\_setup() 함수와 register\_ethdrv()함수의 호출에 의해 네트워크에 디바이스를 등록하지만 WM의 경우는 Miniport 드라이버를 통해 네트워크 디바이스가 등록이 된다.

WM에서는 SD Detect를 통해 AR6002 모듈이 확이 되면, SDIO에 연결되어 있는 장치가 어떤 종류의 장치인지 검색하게 된다. AR6002 모듈은 장치 정보를 레지스터리에 입력함으로사 자신이 어떤 장치인가를 WM에 알려 줘야 한다. 이 작업은 Platform을 빌드하기 전에 Platform.reg 파일에 레지스트리 정보를 입력하면 된다. 이렇게 해서 WM은 SDIO를 통해 어떤 장치가 연결이 되었는지를 알게 되고 관련 DLL파일을 로드 한다. 좀더 자세한 내용을 원하면 **“Windows CE 실전가이드”**책의 네트워크 디바이스 드라이버 부분을 참고 하면 된다.

**SDIO Detect**

**AR6K\_NDIS\_SDIO.DLL 로드**

**DriverEntry() 실행**

**NDIS Miniport 등록**

**AR6K 디바이스 초기화**

그림 15. WM 네트워크 드라이버 등록 절차

위 그림 15의 순서로 WM에서는 네트워크 Miniport가 등록된다. 이 것으로 등록이 끝나는 것이 아니라 Miniport가 호출될 때 불려져야 할 Callback 함수를 작성해야 한다. Callback 함수는 WM이 사용하는 NDIS의 버전에 따라 다르므로 버전을 확인하고 작업을 해야 한다.

**\* WM 네트워크 디바이스 등록시 필요한 중요 Callback 함수.**

- W\_HALT\_HANDLER HaltHandler;

- W\_INITIALIZE\_HANDLER InitializeHandler;

- W\_ISR\_HANDLER ISRHandler;

- W\_HANDLE\_INTERRUPT\_HANDLER HandleInterruptHandler;

- W\_QUERY\_INFORMATION\_HANDLER QueryInformationHandler;

- W\_SET\_INFORMATION\_HANDLER SetInformationHandler;

- W\_SEND\_HANDLER SendHandler;

- W\_TRANSFER\_DATA\_HANDLER TransferDataHandler;

■ **WMI Layer 초기화 및 포팅.**

WMI Layer는 네트워크 계층에서 접속하는 Layer로 H/W와는 무관하게 작성이 되어 있다. 또한 리눅스와 WM의 네트워크 계층에 물려 있는 곳이어서 서로 다른 점이 많은 곳이다.

초기화 과정은 wlan\_node\_table\_init() 함수와, wmi\_qos\_state\_init() 함수를 호출 하는 것으로 끝이 난다. 이 Layer의 포팅 작업은 함수 자체를 가져오면 안되고, WM에서 지원하는 IOCTL을 확인하여 필요한 부분만 가져오는 식으로 작업을 해야 한다. 이 부분을 포팅하는 것은 문제가 되지 않으나 차 후 모듈 제어 관련 이슈가 발생하면 제일 먼저 확인해야 하는 부분이다.

■ **WMI Layer IOCTL 함수 포팅.**

AR6001의 WM(windows mobile) IOCTL 지원 함수를 확인하여 필요한 IOCTL을 AR6002로 부터 가져오는 작업을 진행 한다. 유니퀘스트로부터 받은 “AR6000 Family Programmer’s Guide.pdf”를 참고 하여 진행한다.