AN14213

Wi-Fi Firmware Automatic Recovery

Rev. 3.0 — 12 November 2025 Application note

Document information

Information	Content
Keywords	AN14213, Wi-Fi, firmware, automatic recovery, driver, independent reset, firmware command timeout, firmware wake-up timeout, TX watchdog timeout, firmware trigger dump, station (STA), mobile access point (Mobile AP), Linux OS
Abstract	Explains the Wi-Fi Firmware Automatic Recovery feature for Linux OS and provides examples for STA and Mobile AP modes.

1 About this document

This document describes the Wi-Fi Firmware Automatic Recovery feature. The feature is triggered when the Wi-Fi driver detects firmware exceptions or firmware timeouts.

1.1 Prerequisites

The following is required for Wi-Fi Firmware Automatic Recovery:

- Device operating in STA or mobile AP mode
- PCIe or SDIO Wi-Fi host interface
- Wi-Fi Automatic Recovery feature included in the software release
  Note: *Check the release notes.*

1.2 Supported devices

The following devices support Wi-Fi Firmware Automatic Recovery:

- 88W8987 ref.[1]
- 88W8997 ref.[2]
- 88Q9098 ref.[4]
- 88W9098 ref.[3]
- AW611 ref.[5]
- AW690 ref.[6]
- AW692 ref.[7]
- AW693 ref.[8]
- IW416 ref.[9]
- IW610 ref.[10]
- IW611 ref.[11]
- IW612 ref.[12]

Refer to the section *Feature list* in the release notes of the device software release package to check if Wi-Fi Firmware Automatic Recovery is available.

2 What is Wi-Fi Firmware Automatic Recovery?

Wi-Fi Firmware Automatic Recovery is an NXP proprietary feature that uses Independent Reset (IR) for automatic Wi-Fi firmware re-download and for the re-initialization of the Wi-Fi interfaces. The following events trigger the feature:

- Firmware command timeout
- Firmware wake-up timeout
- TX watchdog timeout
- Firmware dump triggered

Wi-Fi Firmware Automatic Recovery triggers an Independent Reset (IR) via SDIO or PCIe host interface. The firmware is downloaded again without affecting the Bluetooth/802.15.4 activity. The Wi-Fi driver sends a recovery event to WLAN Middleware (MW) that controls wpa_supplicant and/or hostapd for the reinitialization of the Wi-Fi interfaces (mlan0, uap0, p2p0). If wpa_supplicant and/or hostapd are not required, the Wi-Fi driver skips this step.

Figure 1 shows the software architecture.

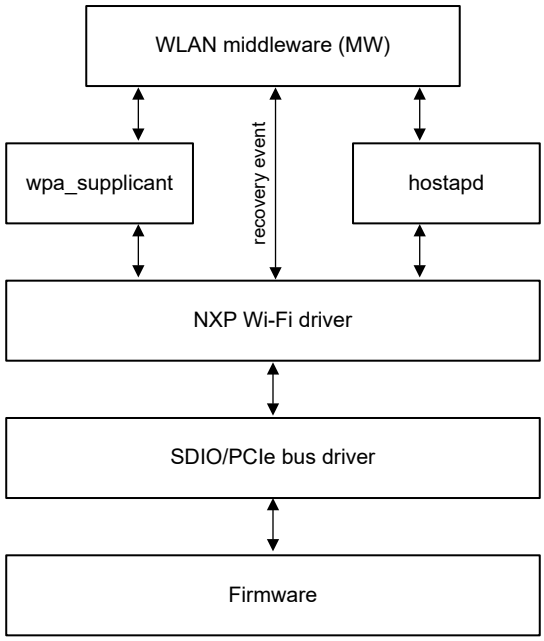

Figure 1. NXP Wi-Fi Automatic Recovery software architecture

2.1 Flow

Figure 2 shows the firmware automatic recovery flow for Wi-Fi.

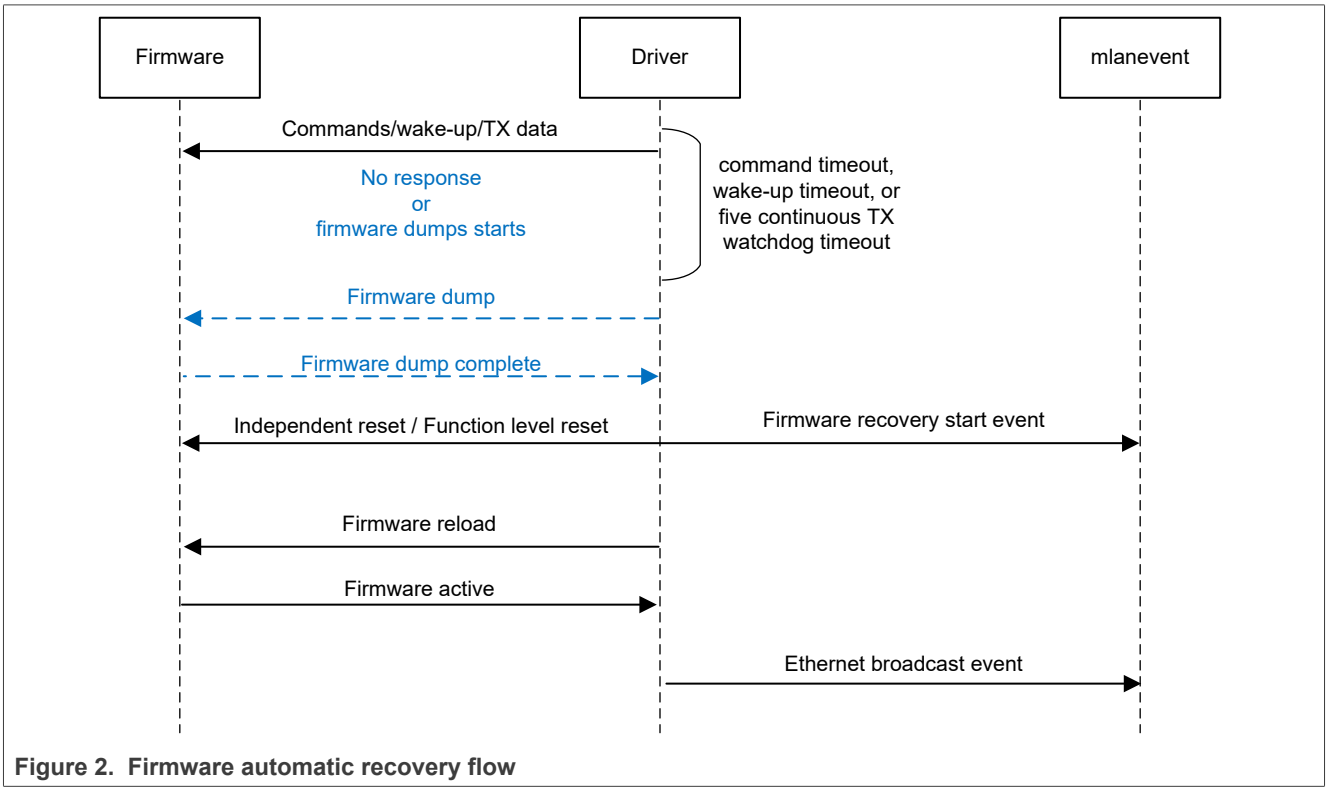

Figure 2. Firmware automatic recovery flow

- The Wi-Fi driver detects one of the trigger events:
  - Firmware command timeout: after a command timeout, the driver does not get a command response from the firmware.
  - Firmware wake-up timeout: after a wake-up timeout, the driver does not get a wake-up response from the firmware.
  - TX watchdog timeout: the Wi-Fi driver receives five continuous TX watchdog timeout from the kernel.
  - Firmware trigger dump: the firmware detects an exception or crash that causes the firmware dump.
- The driver triggers:
  - In-band reset on devices with SDIO host interface for Wi-Fi.
  - Function level reset (FLR) on PCIe devices with PCIe host interface for Wi-Fi.
- After a reset, the driver reloads the Wi-Fi-only firmware binary and waits for the firmware activation. Bluetooth/802.15.4 remain unchanged.
- The driver sends an event to notify mlanevent application that the firmware recovery is complete.

3 Configuration

By default, the Wi-Fi Automatic Recovery feature is enabled. This feature can also be enabled by the driver load parameter (`auto_fw_reload=3` for PCIe, `auto_fw_reload=1` for SDIO).

When Wi-Fi automatic recovery is triggered, the driver automatically reloads the default standalone Wi-Fi firmware in the firmware directory of the host system */lib/firmware/nxp*. The firmware binary (`fw_name = nxp/<fw_name>`) does not need to be specified.

Ensure that both combo and standalone firmware files with the default name are placed under */lib/firmware/nxp*. For instance, on IW612, the default combo FW name is *sduart_nw61x_v1.bin.se* and the default standalone Wi-Fi FW name is *sd_w61x_v1.bin.se*. Refer to the software release package for the default firmware names.

Note: *Refer to the README in the software release package for more information.*

3.1 Linux OS

For Linux OS, some changes are required in the hostapd source for Wi-Fi auto recovery to work. Modify the following hostapd source files and recompile the hostapd image:

- *hostapd/config_file.c*
- *hostapd/hostapd.conf*
- *src/ap/ap_config.c*
- *src/ap/ap_config.h*
- *src/ap/drv_callbacks.c*

Note: *The below changes are for hostapd 2.11. However, the same changes can be mapped to other Android versions. These changes must be applied for Wi-Fi Firmware Auto recovery to work on Android OS.*

Below are the changes to hostapd source:

```
diff -Naur hostapd-2.11/hostapd/config_file.c with-patch/hostapd-2.11/hostapd/
config_file.c
--- hostapd-2.11/hostapd/config_file.c 2024-07-20 23:34:37.000000000 +0530
+++ with-patch/hostapd-2.11/hostapd/config_file.c 2025-08-14 11:43:07.297326292 +0530
@@ -5099,6 +5099,10 @@
   bss->mld_indicate_disabled = atoi(pos);
#endif /* CONFIG_TESTING_OPTIONS */
 #endif /* CONFIG_IEEE80211BE */
+ } else if (os_strcmp(buf, "post_recovery_script") == 0) {
+               bss->post_recovery_script = os_strdup(pos);
+               wpa_printf(MSG_ERROR, "Hostapd: post recovery script: %s",
+                                  bss->post_recovery_script);
  } else {
   wpa_printf(MSG_ERROR,
       "Line %d: unknown configuration item '%s'",
diff -Naur hostapd-2.11/hostapd/hostapd.conf with-patch/hostapd-2.11/hostapd/hostapd.conf
--- hostapd-2.11/hostapd/hostapd.conf 2024-07-20 23:34:37.000000000 +0530
+++ with-patch/hostapd-2.11/hostapd/hostapd.conf 2025-08-14 11:43:54.162390059 +0530
@@ -3322,6 +3322,10 @@
 #bssid=00:13:10:95:fe:0b
 # ...
 #
+# This parameter is used to receive a user supplied script
+# to be executed after receiving the FW_RECOVER_SUCCESS event
+#post_recovery_script=user_script.sh
+#
 # Multiple BSSID Advertisement in IEEE 802.11ax
 # IEEE Std 802.11ax-2021 added a feature where instead of multiple interfaces
 # on a common radio transmitting individual Beacon frames, those interfaces can
diff -Naur hostapd-2.11/src/ap/ap_config.c with-patch/hostapd-2.11/src/ap/ap_config.c
--- hostapd-2.11/src/ap/ap_config.c 2024-07-20 23:34:37.000000000 +0530
```

```
+++ with-patch/hostapd-2.11/src/ap/ap_config.c 2025-08-14 11:44:34.135243201 +0530
@@ -177,6 +177,7 @@
  bss->pasn_comeback_after = 10;
  bss->pasn_noauth = 1;
 #endif /* CONFIG_PASN */
+ bss->post_recovery_script = NULL;
 }

diff -Naur hostapd-2.11/src/ap/ap_config.h with-patch/hostapd-2.11/src/ap/ap_config.h
--- hostapd-2.11/src/ap/ap_config.h 2024-07-20 23:34:37.000000000 +0530
+++ with-patch/hostapd-2.11/src/ap/ap_config.h 2025-08-14 11:45:12.772027587 +0530
@@ -981,6 +981,11 @@
  bool mld_indicate_disabled;
 #endif /* CONFIG_TESTING_OPTIONS */
 #endif /* CONFIG_IEEE80211BE */
+ /**
+        * post_recovery_script - User provided script to be run post
+        * receiving the FW_RECOVER_SUCCESS event
+        */
+       char *post_recovery_script;
 };

 /**
diff -Naur hostapd-2.11/src/ap/drv_callbacks.c with-patch/hostapd-2.11/src/ap/
drv_callbacks.c
--- hostapd-2.11/src/ap/drv_callbacks.c 2024-07-20 23:34:37.000000000 +0530
+++ with-patch/hostapd-2.11/src/ap/drv_callbacks.c 2025-08-14 11:46:42.641728843 +0530
@@ -43,7 +43,7 @@
 #include "fils_hlp.h"
 #include "neighbor_db.h"
 #include "nan_usd_ap.h"
-
+#include <sys/wait.h>

 #ifdef CONFIG_FILS
 void hostapd_notify_assoc_fils_finish(struct hostapd_data *hapd,
@@ -2416,6 +2416,8 @@
  struct sta_info *sta;
 #ifndef CONFIG_NO_STDOUT_DEBUG
  int level = MSG_DEBUG;
+ pid_t pid;
+       int pid_status;

  if (event == EVENT_RX_MGMT && data->rx_mgmt.frame &&
      data->rx_mgmt.frame_len >= 24) {
@@ -2762,6 +2764,36 @@
   hostapd_event_color_change(hapd, true);
   break;
 #endif /* CONFIG_IEEE80211AX */
+ case EVENT_FW_RECOVER_SUCCESS:
+               wpa_printf(MSG_DEBUG, "FW Recover Success event received: %d",
+                                  data->fw_recover_success_data.status);
+
+               if (!hapd->conf->post_recovery_script) {
+                       wpa_printf(MSG_ERROR, "Error: No user provided script!!");
+                       break;
+               }
+
+               /* Fork & exec a child process to run the user supplied
+                * script post FW recovery success event */
+               pid = fork();
+               if (pid < 0) {
+                       wpa_printf(MSG_ERROR, "fork: %s", strerror(errno));
+                       break;
+               }
+               if (pid == 0) {
+                       char *argv[3];
```

```
+
+                        wpa_printf(MSG_DEBUG, "Post Recovery Script: %s",
+                                   hapd->conf->post_recovery_script);
+                        argv[0] = "bash";
+                        argv[1] = os_strdup(hapd->conf->post_recovery_script);
+                       argv[2] = NULL;
+                        execv("/bin/bash", argv);
+                }
+                wpa_printf(MSG_DEBUG, "execv: %s", strerror(errno));
+
+               /* wait for the child process to complete in the parent */
+                waitpid(pid, &pid_status, 0);
  default:
   wpa_printf(MSG_DEBUG, "Unknown event %d", event);
   break;
```

3.2 Android OS

For Android OS, some changes are required in the hostapd source for Wi-Fi auto recovery to work. Modify the following hostapd source files and recompile the Android image:

- */hostapd/aidl/hostapd.cpp*
- */hostapd/ctrl_iface.c*
- */src/ap/drv_callbacks.c*
- */src/ap/hostapd.c*

Apart from modifying the hostapd source code, the Android framework must also be altered. These changes are host-platform specific and relate to the support of auto enablement of the uAP mode after in-band reset and autoconnect with the external STA.

Note: *The below changes are for Android 14. However, the same changes can be mapped to other Android versions. These changes must be applied for Wi-Fi Firmware Auto recovery to work on Android OS.*

Below are the changes to the hostapd source:

```
diff --git a/hostapd/aidl/hostapd.cpp b/hostapd/aidl/hostapd.cpp
index 9eb08762..01de5b28 100644
--- a/hostapd/aidl/hostapd.cpp
+++ b/hostapd/aidl/hostapd.cpp
@@ -1070,12 +1070,13 @@ std::vector<uint8_t>  generateRandomOweSsid()
   } else if (os_strncmp(txt, AP_EVENT_DISABLED, strlen(AP_EVENT_DISABLED)) == 0
                            || os_strncmp(txt, INTERFACE_DISABLED,
 strlen(INTERFACE_DISABLED)) == 0)
   {
+   wpa_printf(MSG_ERROR, "__debug__: Avoid failure callback invocation: %s", txt);
    // Invoke the failure callback on all registered clients.
-   for (const auto& callback : callbacks_) {
+   /*for (const auto& callback : callbacks_) {
     callback->onFailure(strlen(iface_hapd->conf->bridge) > 0 ?
      iface_hapd->conf->bridge : iface_hapd->conf->iface,
          iface_hapd->conf->iface);
-   }
+   }*/
   }
  };

diff --git a/hostapd/ctrl_iface.c b/hostapd/ctrl_iface.c
index b46d9210..9820e2e9 100644
--- a/hostapd/ctrl_iface.c
+++ b/hostapd/ctrl_iface.c
@@ -1375,6 +1375,7 @@ static int hostapd_ctrl_iface_enable(struct hostapd_iface *iface)
   wpa_printf(MSG_ERROR, "Enabling of interface failed");
```

```
    return -1;
   }
+ iface->user_disabled_iface = false;
   return 0;
 }

@@ -1405,6 +1406,7 @@ static int hostapd_ctrl_iface_disable(struct hostapd_iface *iface)
    wpa_printf(MSG_ERROR, "Disabling of interface failed");
    return -1;
   }
+ iface->user_disabled_iface = true;
   return 0;
 }

diff --git a/src/ap/drv_callbacks.c b/src/ap/drv_callbacks.c
index 510a06c6..2f4e5655 100644
--- a/src/ap/drv_callbacks.c
+++ b/src/ap/drv_callbacks.c
@@ -1736,7 +1736,7 @@ static void hostapd_event_iface_unavailable(struct hostapd_data
 *hapd)
   }

   // inform framework that interface is unavailable
- hostapd_disable_iface(hapd->iface);
+ //hostapd_disable_iface(hapd->iface);
 }


@@ -2110,9 +2110,13 @@ void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
    }
    break;
   case EVENT_INTERFACE_DISABLED:
-   hostapd_free_stas(hapd);
    wpa_msg(hapd->msg_ctx, MSG_INFO, INTERFACE_DISABLED);
    hapd->disabled = 1;
+   if (hostapd_disable_iface(hapd->iface) < 0 ) {
+    wpa_printf(MSG_ERROR,
+        "Failed to disable iface on event:%d\n",
+        event);
+   }
    break;
 #ifdef CONFIG_ACS
   case EVENT_ACS_CHANNEL_SELECTED:
@@ -2158,6 +2162,21 @@ void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
    hostapd_cleanup_cca_params(hapd);
    break;
 #endif /* CONFIG_IEEE80211AX */
+ case EVENT_INTERFACE_STATUS:
+  /* Event is handled only in autorecovery instance */
index 510a06c6..2f4e5655 100644
--- a/src/ap/drv_callbacks.c
+++ b/src/ap/drv_callbacks.c
@@ -1736,7 +1736,7 @@ static void hostapd_event_iface_unavailable(struct hostapd_data
 *hapd)
   }

   // inform framework that interface is unavailable
- hostapd_disable_iface(hapd->iface);
+ //hostapd_disable_iface(hapd->iface);
 }


@@ -2110,9 +2110,13 @@ void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
    }
    break;
   case EVENT_INTERFACE_DISABLED:
-   hostapd_free_stas(hapd);
```

```
    wpa_msg(hapd->msg_ctx, MSG_INFO, INTERFACE_DISABLED);
    hapd->disabled = 1;
+   if (hostapd_disable_iface(hapd->iface) < 0 ) {
+    wpa_printf(MSG_ERROR,
+        "Failed to disable iface on event:%d\n",
+        event);
+   }
    break;
  #ifdef CONFIG_ACS
   case EVENT_ACS_CHANNEL_SELECTED:
@@ -2158,6 +2162,21 @@ void wpa_supplicant_event(void *ctx, enum wpa_event_type event,
    hostapd_cleanup_cca_params(hapd);
    break;
  #endif /* CONFIG_IEEE80211AX */
+ case EVENT_INTERFACE_STATUS:
+  /* Event is handled only in autorecovery instance */

   int (*enable_iface_cb)(struct hostapd_iface *iface);
   int (*disable_iface_cb)(struct hostapd_iface *iface);
+
+ /* Set if user disabled interface */
+ bool user_disabled_iface;
  };

  /* hostapd.c */
--
2.34.1
```

4 Examples

This section provides examples for Wi-Fi Firmware Automatic Recovery in STA and Mobile AP modes using the 88W9098 PCIe interface. Any device with a host interface – SDIO or PCIe – mentioned in Section 1.2 "Supported devices" can follow the examples.

4.1 Linux OS

This section provides an example for Wi-Fi Firmware Automatic Recovery in STA and Mobile AP modes for Linux OS.

4.1.1 STA mode

The following steps demonstrate Wi-Fi Firmware Automatic Recovery in STA mode:

Step 1 – Copy both combo firmware and standalone Wi-Fi firmware to the firmware (*/lib/firmware/nxp/*) directory of the host system.

Step 2 – Load the DUT with drivers and firmware.

```
insmod mlan.ko
insmod moal.ko drvdbg=0x20037
```

Note: *`drvdbg=0x20037` is optional and is used to print debug logs.*

Command output example:

```
wlan: Loading MWLAN driver
wlan: Register to Bus Driver...
wlan_pcie 0000:04:00.0: enabling device (0000 -> 0002)
Attach moal handle ops, card interface type: 0x206
combo fw:nxp/pcieuart9098_combo_v1.bin wlan fw:nxp/pcie9098_wlan_v1.bin
Request firmware: nxp/pcieuart9098_combo_v1.bin
FW download over, size 745944 bytes
WLAN FW is active
on_time is 844041139284779
fw hotfix ver=75
fw ver=15.1
uap fw ver=2.0
max_p2p_conn = 8, max_sta_conn = 48
fw_cap_info=0xc8fcefa3 fw_cap_ext=0x310280
wlan: version = PCIE9098--17.68.1.p149.75-MXM5X17405.p56-GPL-(FP68)
```

In the example above, the firmware was loaded successfully and Wi-Fi Firmware Automatic Recovery is enabled (by default).

Step 3 – Bring up the DUT in STA mode using wpa_supplicant.

Example of wpa_supplicant content:

```
network={
 SSID="SSID"
 psk="Password"
 key_mgmt=WPA-PSK
}
```

Step 4 – Run wpa_supplicant.

```
wpa_supplicant -i mlan0 -D nl80211 -c <path_to_file>/wpa_supplicant.conf -B
```

Command output example:

```
mlan0: SME: Trying to authenticate with 7c:10:c9:02:da:48 (SSID='ASUS_2G' freq=2427 MHz)
mlan0: Trying to associate with 7c:10:c9:02:da:48 (SSID='ASUS_2G' freq=2427 MHz)
mlan0: Associated with 7c:10:c9:02:da:48
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
mlan0: CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=US
mlan0: WPA: Key negotiation completed with 7c:10:c9:02:da:48 [PTK=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to 7c:10:c9:02:da:48 completed [id=0 id_str=]
```

Step 5 – The driver detects any firmware condition including command timeout, wake-up timeout, TX watchdog timeout, and firmware trigger dump. Look for the following output on the `dmesg`:

Note: *`Bad address` in the output shows that the device entered into a bad state.*

```
==== Start Receive FW dump event  ====
Create directory /data/dump_768779 error, try create dir in /var
Create directory /var/dump_768779 successfully
Firmware Dump directory name is /var/dump_768779
=== START DRIVER INFO DUMP===
DRV dump data in /var/dump_768779/file_drv_info_2
func1: Wakeup device...
Drv info total bytes = 302619 (0x49e1b)
[768823.612682] === DRIVER INFO DUMP END===
=== START DRIVER INFO DUMP===
DRV dump data in /var/dump_768779/file_drv_info
func0: Wakeup device...
Drv info total bytes = 302620 (0x49e1c)
=== DRIVER INFO DUMP END===
==== FW DUMP END: 2476592 bytes ====
wlan: Notify FW dump complete event
```

Step 6 – Verify that the DUT firmware automatically recovers, restarts in STA mode, and reconnects to the external AP.

Note: *The ongoing Wi-Fi data transmission is impacted and can be lost.*

Example of the expected result:

```
WIFI auto_fw_reload: fw_reload=6
========START IN-BAND RESET===========
Request firmware: nxp/pcie9098_wlan_v1.bin
FW download over, size 508016 bytes
WLAN FW is active
on_time is 844227431289970
wlan: version = PCIE9098--17.68.1.p149.75-MXM5X17405.p56-GPL-(FP68)
========END IN-BAND RESET===========
mlan0: CTRL-EVENT-SSID-TEMP-DISABLED id=0 ssid="ASUS_2G" auth_failures=1 duration=10
 reason=CONN_FAILED
mlan0: CTRL-EVENT-SSID-REENABLED id=0 ssid="ASUS_2G"
mlan0: Trying to associate with bc:a5:11:a2:b3:4d (SSID='ASUS_2G' freq=2427 MHz)
mlan0: Associated with bc:a5:11:a2:b3:4d
mlan0: CTRL-EVENT-SUBNET-STATUS-UPDATE status=0
mlan0: CTRL-EVENT-REGDOM-CHANGE init=COUNTRY_IE type=COUNTRY alpha2=DE
mlan0: WPA: Key negotiation completed with bc:a5:11:a2:b3:4d [PTK=CCMP GTK=CCMP]
mlan0: CTRL-EVENT-CONNECTED - Connection to bc:a5:11:a2:b3:4d completed [id=0 id_str=]
```

4.1.2 Mobile AP mode

The following steps are used for Wi-Fi Firmware Automatic Recovery in Mobile AP mode:

Step 1 – Copy both combo firmware and standalone Wi-Fi firmware to the firmware directory of the system (*/lib/firmware/nxp/*).

Step 2 – Load the DUT with drivers and firmware.

```
wlan: Loading MWLAN driver
wlan: Register to Bus Driver...
wlan_pcie 0000:04:00.0: enabling device (0000 -> 0002)
Attach moal handle ops, card interface type: 0x206
combo fw:nxp/pcieuart9098_combo_v1.bin wlan fw:nxp/pcie9098_wlan_v1.bin
Request firmware: nxp/pcieuart9098_combo_v1.bin
FW download over, size 745944 bytes
WLAN FW is active
on_time is 844041139284779
fw hotfix ver=75
fw ver=15.1
uap fw ver=2.0
max_p2p_conn = 8, max_sta_conn = 48
fw_cap_info=0xc8fcefa3 fw_cap_ext=0x310280
wlan: version = PCIE9098--17.68.1.p149.75-MXM5X17405.p56-GPL-(FP68)
```

Note: *`drvdbg=0x20037` is optional and is used to print debug logs.*

Command output example:

```
wlan: Loading MWLAN driver
wlan: Register to Bus Driver...
wlan_pcie 0000:04:00.0: enabling device (0000 -> 0002)
Attach moal handle ops, card interface type: 0x206
combo fw:nxp/pcieuart9098_combo_v1.bin wlan fw:nxp/pcie9098_wlan_v1.bin
Request firmware: nxp/pcieuart9098_combo_v1.bin
FW download over, size 745944 bytes
WLAN FW is active
on_time is 844041139284779
fw hotfix ver=75
fw ver=15.1
uap fw ver=2.0
max_p2p_conn = 8, max_sta_conn = 48
fw_cap_info=0xc8fcefa3 fw_cap_ext=0x310280
wlan: version = PCIE9098--17.68.1.p149.75-MXM5X17405.p56-GPL-(FP68)
```

In the example above, the firmware was loaded successfully and Wi-Fi Firmware Automatic Recovery is enabled (by default).

Step 3 – Bring up the DUT in Mobile AP mode using hostapd.

Example of hostapd content:

```
interface=uap0
hw_mode=g
channel=6
country_code=US
ssid=NXP_Demo
ieee80211n=1
```

Step 4 – Start hostapd.

```
./hostapd hostapd.conf &
```

Command output example:

```
uap0: interface state UNINITIALIZED->COUNTRY_UPDATE
wlan: Starting AP
wlan: AP started
wlan: HostMlme uap0 send deauth/disassoc
Set AC=3, txop=47 cwmin=3, cwmax=7 aifs=1
Set AC=2, txop=94 cwmin=7, cwmax=15 aifs=1
Set AC=0, txop=0 cwmin=15, cwmax=63 aifs=3
Set AC=1, txop=0 cwmin=15, cwmax=1023 aifs=7
uap0: interface state COUNTRY_UPDATE->ENABLED
uap0: AP-ENABLED
```

Step 5 – Connect to a peer STA device.

Command output example:

```
uap0: STA 6c:c7:ec:90:5f:e5 IEEE 802.11: authenticated
uap0: STA 6c:c7:ec:90:5f:e5 IEEE 802.11: associated (aid 1)
uap0: AP-STA-CONNECTED 6c:c7:ec:90:5f:e5
uap0: STA 6c:c7:ec:90:5f:e5 RADIUS: starting accounting session CF66B83E17126696
uap0: STA 6c:c7:ec:90:5f:e5 WPA: pairwise key handshake completed (WPA)
uap0: EAPOL-4WAY-HS-COMPLETED 6c:c7:ec:90:5f:e5
uap0: STA 6c:c7:ec:90:5f:e5 WPA: group key handshake completed (WPA)
```

Step 6 – The driver detects any firmware condition including command timeout, wake-up timeout, TX watchdog timeout, and firmware trigger dump. Look for the following output on the `dmesg`:

Note: *`Bad address` in the output shows that the device entered into a bad state.*

```
==== Start Receive FW dump event  ====
Create directory /data/dump_768779 error, try create dir in /var
Create directory /var/dump_768779 successfully
Firmware Dump directory name is /var/dump_768779
=== START DRIVER INFO DUMP===
DRV dump data in /var/dump_768779/file_drv_info_2
func1: Wakeup device...
Drv info total bytes = 302619 (0x49e1b)
=== DRIVER INFO DUMP END===
=== START DRIVER INFO DUMP===
DRV dump data in /var/dump_768779/file_drv_info
func0: Wakeup device...
Drv info total bytes = 302620 (0x49e1c)
=== DRIVER INFO DUMP END===
==== FW DUMP END: 2476592 bytes ====
wlan: Notify FW dump complete event
WIFI auto_fw_reload: fw_reload=6
========START IN-BAND RESET===========
```

Step 7 – Verify that the DUT firmware automatically recovers, restarts Mobile AP mode, and reconnects to the external STA.

Note: *The ongoing Wi-Fi data transmission is impacted and can be lost.*

Example of the expected result:

```
WIFI auto_fw_reload: fw_reload=6
========START IN-BAND RESET===========
Request firmware: nxp/pcie9098_wlan_v1.bin
FW download over, size 508016 bytes
WLAN FW is active
on_time is 844227431289970
wlan: version = PCIE9098--17.68.1.p149.75-MXM5X17405.p56-GPL-(FP68)
========END IN-BAND RESET===========
```

```
uap0: AP-DISABLED
uap0: AP-STA-DISCONNECTED 6c:c7:ec:90:5f:e5
Failed to set beacon parameters
nl80211: deinit ifname=uap0 disabled_11b_rates=0
uap0: interface state ENABLED->DISABLED
uap0: interface state DISABLED->ENABLED
uap0: AP-ENABLED
uap0: STA 6c:c7:ec:90:5f:e5 IEEE 802.11: authenticated
uap0: STA 6c:c7:ec:90:5f:e5 IEEE 802.11: associated (aid 1)
uap0: AP-STA-CONNECTED 6c:c7:ec:90:5f:e5
uap0: STA 6c:c7:ec:90:5f:e5 RADIUS: starting accounting session 78CA40AD2A06EF45
```

4.2 Android OS

This section provides an example for Wi-Fi Firmware Automatic Recovery in STA and Mobile AP modes for Android OS.

4.2.1 STA mode

The following steps demonstrate Wi-Fi Firmware Automatic Recovery in STA mode:

Step 1 – Copy both combo firmware and standalone Wi-Fi firmware to the firmware (*/lib/firmware/nxp/*) directory of the host system.

Step 2 – Boot up the wireless SoC and host platform. The drivers and firmware must be loaded automatically.

Example of logcat log:

```
mlan: loading out-of-tree module taints kernel.
wlan: Loading MWLAN driver
wlan: Register to Bus Driver…
wlan_pcie 0001:01:00.0: enabling device (0000 -> 0002)
Attach moal handle ops, card interface type: 0x204
PCIE8997: init module param from usr cfg
card_type: PCIE8997, config block: 0
cfg80211_wext=0xf
sta_name=wlan
uap_name=wlan
wfd_name=p2p
max_vir_bss=1
cal_data_cfg=none
ps_mode = 2
auto_ds = 2
auto_fw_reload 3
Attach mlan adapter operations.card_type is 0x204.
Request firmware: nxp/pcieuart8997_combo_v4.bin
FW download over, size 625312 bytes
WLAN FW is active
wlan: version = PCIE8997--16.92.21.p151.4-MM6X16537.p9-(FP92)
wlan: Register to Bus Driver Done
wlan: Driver loaded successfully
```

Step 3 – Using the command-line interface or GUI, enable STA mode and connect to an external AP.

Step 4 – The driver detects any firmware condition including command timeout, wake-up timeout, TX watchdog timeout, and firmware trigger dump.

Step 5 – Verify that the DUT firmware automatically recovers, restarts in STA mode, and reconnects to the external AP.

Note: *The ongoing Wi-Fi data transmission is impacted and can be lost.*

Example of logcat log:

```
WIFI auto_fw_reload: fw_reload=6
========START IN-BAND RESET===========
PCIE Trigger FW In-band Reset success.
PCIE8997: init module param from usr cfg
card_type: PCIE8997, config block: 0
cfg80211_wext=0xf
sta_name=wlan
uap_name=wlan
wfd_name=p2p
max_vir_bss=1
cal_data_cfg=none
ps_mode = 2
auto_ds = 2
auto_fw_reload 3
rx_work=1 cpu_num=4
Enable moal_recv_amsdu_packet
Attach mlan adapter operations.card_type is 0x204.
Request firmware: nxp/pcie8997_wlan_v4.bin
FW download over, size 440432 bytes
WLAN FW is active
wlan: version = PCIE8997--16.92.21.p151.4-MM6X16537.p9-(FP92)
========END IN-BAND RESET===========
init: starting service 'wificond'…
init: starting service 'wpa_supplicant'…
wlan: HostMlme wlan0 send auth to bssid 6e:XX:XX:XX:5f:e5
wlan0:
wlan: HostMlme Auth received from 6e:XX:XX:XX:5f:e5
wlan: HostMlme wlan0 Connected to bssid 6e:XX:XX:XX:5f:e5 successfully
```

4.2.2 Mobile AP mode

The following steps demonstrate Wi-Fi Firmware automatic recovery in mobile AP mode for Android OS:

Step 1 – Modify the hostapd source and recompile the Android image. Refer to Section 3.2 "Android OS" for more information.

Step 2 – Copy both combo firmware and standalone Wi-Fi firmware to the firmware (*/lib/firmware/nxp/*) directory of the host system.

Step 3 – Boot up the wireless SoC and host platform. The drivers and firmware must be loaded automatically.

Example of logcat log:

```
mlan: loading out-of-tree module taints kernel.
wlan: Loading MWLAN driver
wlan: Register to Bus Driver…
wlan_pcie 0001:01:00.0: enabling device (0000 -> 0002)
Attach moal handle ops, card interface type: 0x204
PCIE8997: init module param from usr cfg
card_type: PCIE8997, config block: 0
cfg80211_wext=0xf

sta_name=wlan

uap_name=wlan

wfd_name=p2p

max_vir_bss=1

cal_data_cfg=none
```

```
ps_mode = 2

auto_ds = 2

auto_fw_reload 3
Attach mlan adapter operations.card_type is 0x204.

Request firmware: nxp/pcieuart8997_combo_v4.bin
FW download over, size 625312 bytes

WLAN FW is active
wlan: version = PCIE8997--16.92.21.p151.4-MM6X16537.p9-(FP92)

wlan: Register to Bus Driver Done

wlan: Driver loaded successfully
```

Step 4 – Using the command-line interface or GUI, start uAP mode and connect to an external STA.

Step 5 – The driver detects any firmware condition including command timeout, wake-up timeout, TX watchdog timeout, and firmware trigger dump.

Step 6 – Verify that the DUT firmware automatically recovers, restarts in uAP mode, and reconnects to the external STA.

Note: *The ongoing Wi-Fi data transmission is impacted and can be lost.*

Example of logcat log:

```
WIFI auto_fw_reload: fw_reload=6
========START IN-BAND RESET===========
PCIE Trigger FW In-band Reset success.
PCIE8997: init module param from usr cfg

card_type: PCIE8997, config block: 0

cfg80211_wext=0xf

sta_name=wlan

uap_name=wlan

wfd_name=p2p

max_vir_bss=1

cal_data_cfg=none

ps_mode = 2

auto_ds = 2

auto_fw_reload 3

rx_work=1 cpu_num=4

Enable moal_recv_amsdu_packet

Attach mlan adapter operations.card_type is 0x204.

Request firmware: nxp/pcie8997_wlan_v4.bin

FW download over, size 440432 bytes

WLAN FW is active
wlan: version = PCIE8997--16.92.21.p151.4-MM6X16537.p9-(FP92
```

```
========END IN-BAND RESET===========
init: starting service 'wificond'…
init: starting service 'wpa_supplicant'…
wlan: HostMlme wlan0 send auth to bssid 6e:XX:XX:XX:5f:e5

wlan0:

wlan: HostMlme Auth received from 6e:XX:XX:XX:5f:e5

wlan: HostMlme wlan0 Connected to bssid 6e:XX:XX:XX:5f:e5 successfully
```

5 Abbreviations

Table 1. Abbreviations

Abbreviation	Definition
AP	Access point
DUT	Device under test
FLR	Function level reset
STA	Station

6 References

[1] Webpage – 88W8987: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 5 (802.11ac) + Bluetooth® Solution (link)
[2] Webpage – 88W8997: 2.4/5 GHz Dual-band 2x2 Wi-Fi® 5 (802.11ac) + Bluetooth® Solution (link)
[3] Webpage – 88Q9098/88Q9098S: 2.4/5 GHz Dual-band 2x2 Wi-Fi® 6 (802.11ax) + Bluetooth® Automotive Solution (link)
[4] Webpage – 88W9098: 2.4/5 GHz Dual-band 2x2 Wi-Fi® 6 (802.11ax) + Bluetooth® (link)
[5] Webpage – AW611: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 (802.11ax) + Bluetooth® Automotive Solution (link)
[6] Webpage – AW690: Wi-Fi® 6 1x1 Concurrent Dual Wi-Fi (CDW) and Bluetooth® Combo SoC (link)
[7] Webpage – AW692: 2x2 Single-band (5 GHz) Concurrent Dual Wi-Fi® 6, 1x1 (2.4 GHz) Wi-Fi 6, and Bluetooth® Combo Solution (link)
[8] Webpage – AW693: 2x2 Dual-band (5-7 GHz), 1x1 (2.4 GHz) Concurrent Dual Wi-Fi 6/6E and Bluetooth Combo Solution (link)
[9] Webpage – IW416: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 4 (802.11n) + Bluetooth® Solution (link)
[10] Webpage – IW610: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 + Bluetooth Low Energy + 802.15.4 Tri-Radio Solution (link)
[11] Webpage – IW611: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 (802.11ax) + Bluetooth® Solution (link)
[12] Webpage – IW612: 2.4/5 GHz Dual-band 1x1 Wi-Fi® 6 (802.11ax) + Bluetooth® + 802.15.4 Tri-radio Solution (link)

7 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024-2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 2. Revision history

Document ID	Release date	Description
AN14213 v.3.0	12 November 2025	• Section 3 "Configuration": Updated for Section 3.1 "Linux OS" and Section 3.2 "Android OS". • Section 4 "Examples": Updated for Section 4.1 "Linux OS" and Section 4.2 "Android OS".
AN14213 v.2.0	16 May 2025	• Section 1.2 "Supported devices": added IW610, AW692, and AW693. • Section 2 "What is Wi-Fi Firmware Automatic Recovery?": updated the description and figure. • Section 2.1 "Flow": updated the description and figure. • Section 3 "Configuration": removed the note about Wi-Fi-only firmware. • Section 4.2.1 "STA mode": reworded **Step 6**. • Section 4.1.2 "Mobile AP mode": reworded **Step 7**. • Section 5 "Abbreviations": added FLR. • Section 6 "References": added IW610, AW692, and AW693 webpages.
AN14213 v.1.0	12 September 2024	• Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Tables

Figures

Contents