ATWILC1000/ATWILC3000

ATWILC 器件 Linux®移植指南

简介

本用户指南介绍了如何将 ATWILC1000 和 ATWILC3000 Linux 驱动程序移植到另一个平台，以及移植驱动程序需要进行哪些修改。

ATWILC 器件的驱动程序和固件版本的相关源代码在 GitHub 上维护。建议用户访问以下网站获取最新代码：

- ATWILC1000/3000 驱动程序： https://github.com/linux4wilc/driver
- ATWILC1000/3000 固件： https://github.com/linux4wilc/firmware

前提条件

- 硬件前提条件：
  - ATWILC1000
  - ATWILC3000
- 软件前提条件：
  - ATWILC1000 Linux 驱动程序源代码
  - ATWILC1000 固件二进制文件
  - ATWILC3000 Linux 驱动程序源代码
  - ATWILC3000 固件二进制文件

目录

1. 驱动程序架构

驱动程序架构如下列图中所示。

图 1-1. ATWILC1000 Linux 驱动程序架构

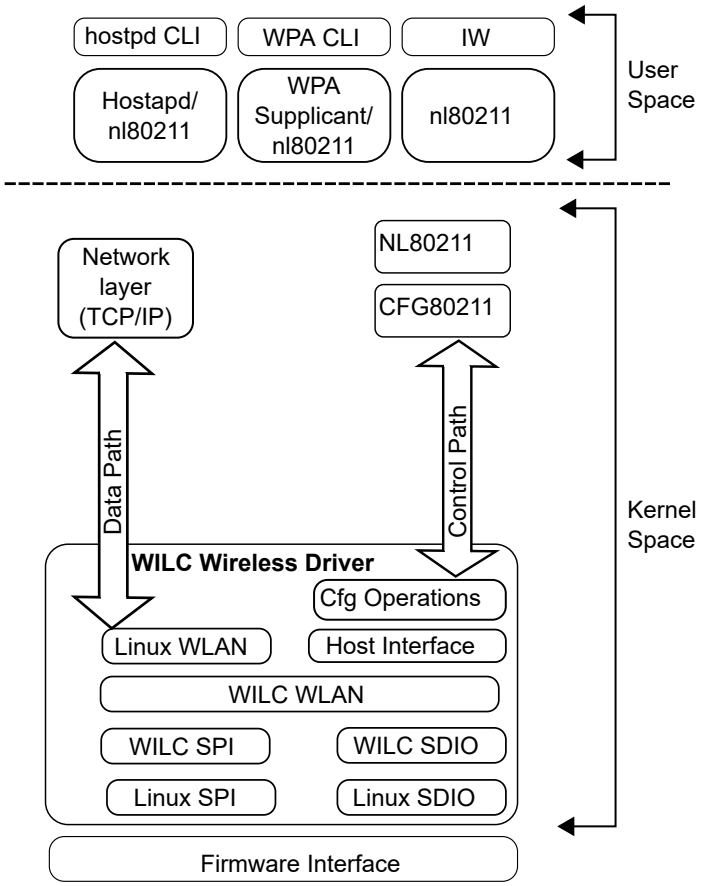
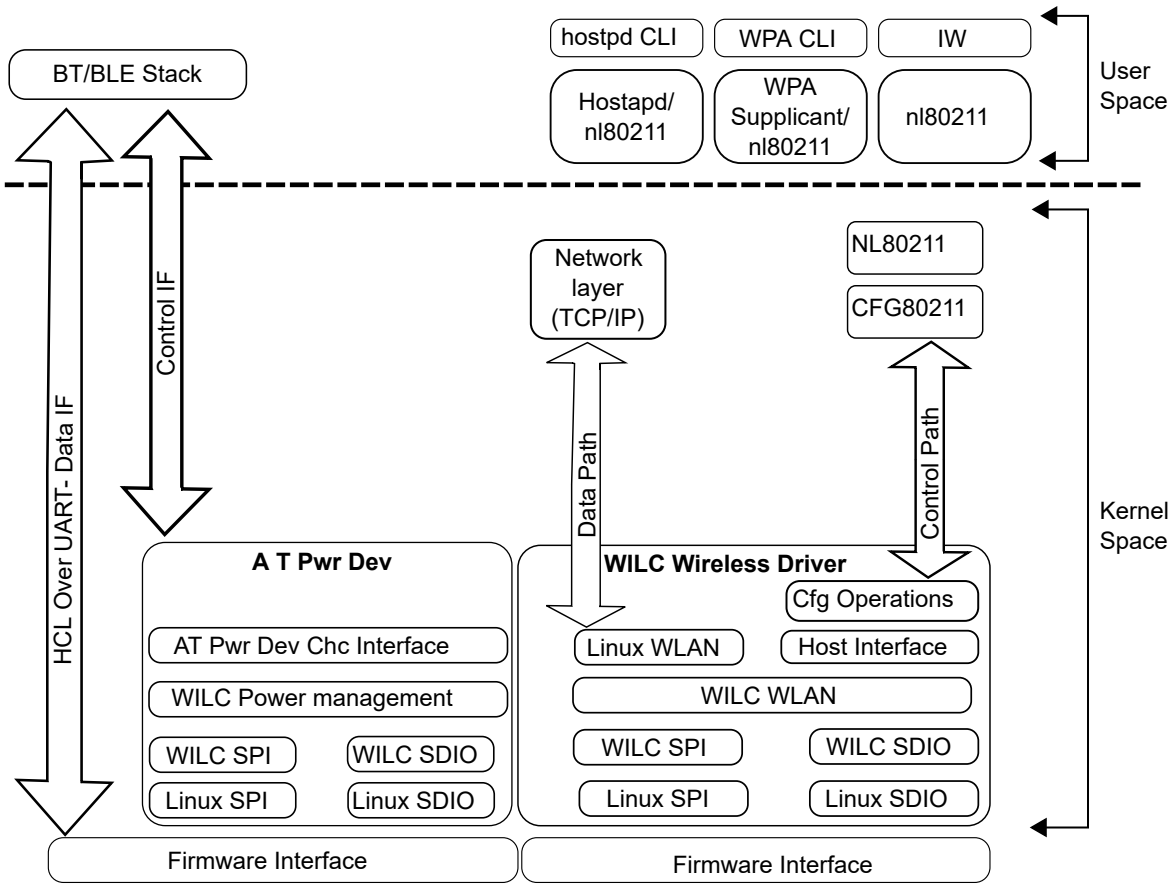


图 1-2. ATWILC3000 Linux 驱动程序架构



1.1 驱动程序模块

本节介绍了驱动程序模块。

1.1.1 Linux WLAN

Linux WLAN 用于实现 Linux 特定操作：

- 控制 ATWILC 器件电源
- 在插入/移除期间初始化/取消初始化模块
- 注册中断服务寄存器（Interrupt Service Register，ISR）
- 通过控制 CHIP_EN 和 RESET_N GPIO 开启和关闭芯片。
- 注册 net_device 接口
- 下载并启动固件
- 实现以下 net_device 操作：
  - mac_init_fn
  - mac_open
  - mac_close
  - mac_xmit
  - mac_ioctl
  - mac_stats

  - wilc_set_multicaset_list
- 将 Tx 包推入 Tx 队列
- 将 Rx 包发送到 Linux 协议栈

1.1.2 ATWILC WLAN

ATWILC WLAN 用于实现 ATWILC 器件的 ASIC 特定操作：

- 初始化芯片
- 下载并启动固件
- 发送配置包
- 处理 Tx 队列，将包发送到固件
- 从固件接收包

1.1.3 配置（cfg）操作

- 实现 wiphy 接口
- 使用 `cfg80211` 注册
- 实现 `cfg80211_ops` 操作
- 从上层接收 cfg 操作请求
- 使用需要异步处理的报文将 cfg 操作请求传递给主机接口
- 将 cfg 操作响应/回调传递给上层

1.1.4 主机接口

- 处理 cfg 操作发出的带有 cfg 请求的报文
- 编译配置包
- 将配置包发送到 ATWILC 器件，并将 WLAN 发送到固件
- 接收 cfg 包响应

1.1.5 ATWILC SDIO/SPI

实现与通信总线协议相关的 ATWILC 特定操作。

1.1.6 Linux SDIO/SPI：

实现与通信总线相关的基本 Linux 特定操作。

1.1.7 ATWILC 功耗管理

- 精简驱动程序，为蓝牙低功耗（Bluetooth® Low Energy，BLE）协议栈和 Wi-Fi®驱动程序提供服务
- 实现字符器件接口，用于将命令发送到 BLE
- 导出 Wi-Fi 驱动程序的 API
- 根据蓝牙和 Wi-Fi 的请求决定何时开启和关闭 ATWILC 器件
- 使用串行外设接口（Serial Peripheral Interface，SPI）或安全数字输入输出（Secure Digital Input Output，SDIO）将 BLE 固件下载到 ATWILC 器件的智能随机存取存储器（Intelligent Random Access Memory，IRAM）

1.1.8 AT Pwr Dev 字符接口

蓝牙/BLE 协议栈使用在用户空间中运行的字符接口将命令传送到电源管理层。

- `echo BT_POWER_UP > /dev/wilc_bt` 命令用于为芯片上电。如果 Wi-Fi 开启，则该命令不会生效
- `echo BT_POWER_DOWN > /dev/wilc_bt` 命令用于使芯片掉电。如果 Wi-Fi 开启，则该命令不会生效
- `echo BT_DOWNLOAD_FW > /dev/wilc_bt` 命令用于通过 SDIO 或 SPI 将蓝牙固件下载到 IRAM
- `echo BT_FW_CHIP_WAKEUP > /dev/wilc_bt` 命令用于唤醒芯片，在通过 UART 或 SDIO/SPI 下载固件前发出该命令
- `echo BT_FW_CHIP_ALLOW_SLEEP > /dev/wilc_bt` 命令用于在适当情况下允许芯片进入休眠模式

2. 内核修改

2.1 驱动程序源代码集成

要将 ATWILC 器件驱动程序的源代码集成到内核的编译系统中，请执行以下步骤：

1. 在 linux_root/drivers/net/wireless/mchp w 下添加包含 Kconfig 文件的驱动程序源代码。编辑 linux_root/drivers/net/wireless/Kconfig 以添加 ATWILC 驱动程序 Kconfig 源的路径 drivers/net/wireless/mchp/Kconfig。
2. 编辑 linux_root/drivers/net/wireless/Makefile 以添加 ATWILC 的驱动程序源代码 obj-$ (CONFIG_WLAN_VENDOR_MCHP) += mchp/。

2.2 固件集成

要将 ATWILC 的固件（FW）文件编译为内核的一部分，请执行以下步骤：

1. 将固件添加到 linux_root/firmware/mchp/
2. 编辑 linux_root/firmware/Makefile 以添加固件文件
   2.1. fw-shipped-$(CONFIG_WILC) += mchp/wilc1000_wifi_firmware.bin
   2.2. fw-shipped-$(CONFIG_WILC) += mchp/wilc3000_wifi_firmware.bin
   2.3. fw-shipped-$(CONFIG_WILC) += mchp/wilc3000_ble_firmware.bin

2.3 内核配置

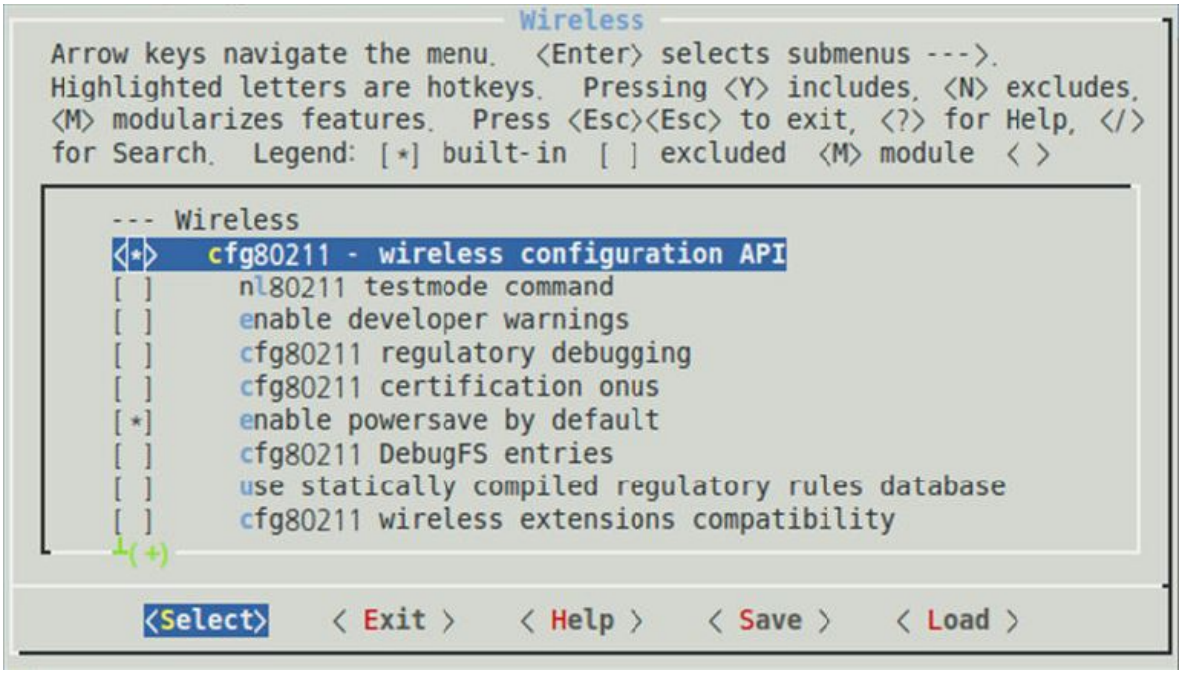
用户必须在 Linux 内核配置中执行以下修改才能启用 Wi-Fi。可使用 menuconfig 进行修改，如果内核是单独编译的，还可通过在 buildroot 默认使用的 defconfig 中文件中添加相应的 CONFIG 条目来进行修改。

1. 转到 Linux 内核的目录并输入以下命令：

```
$ make ARCH=arm menuconfig
```

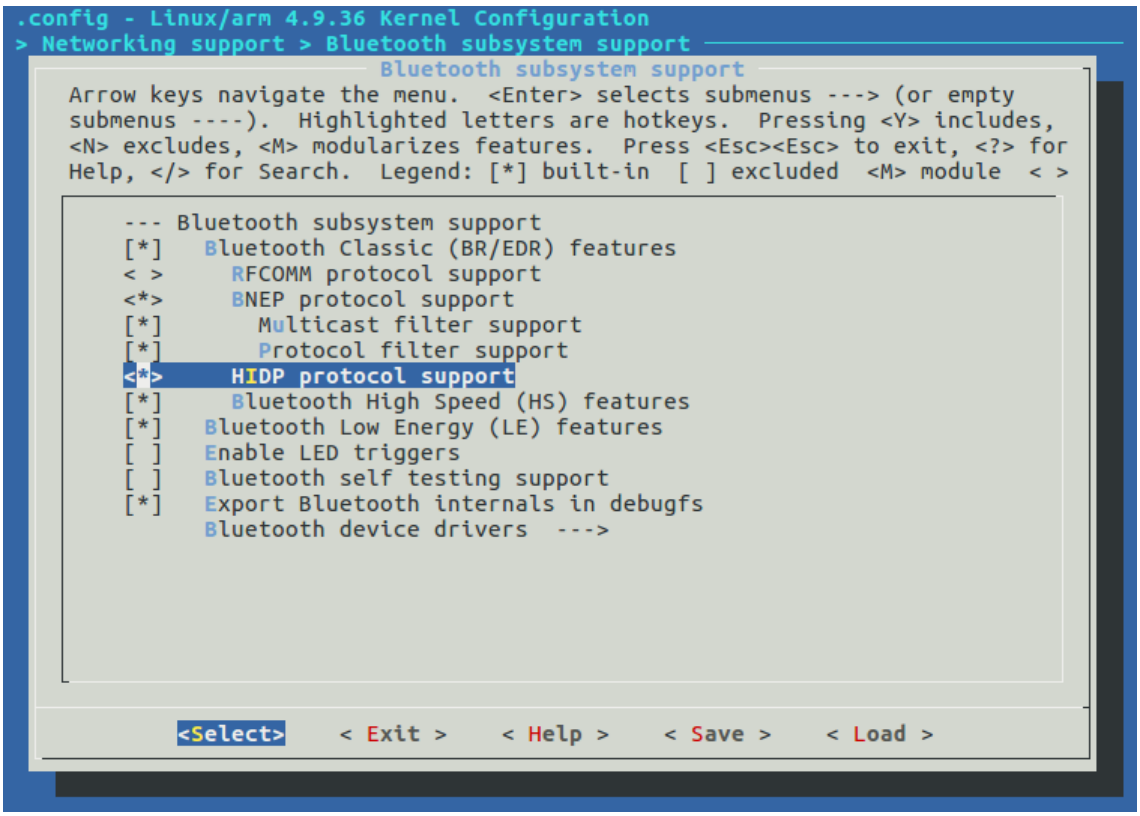
   从蓝牙终端选择要保存和退出的所需配置。
2. 启用 Linux 802.11 配置 API 空间。从 networking support > wireless menu（网络支持 > 无线菜单）中选择 cfg80211 - wireless configuration API（cfg80211 - 无线配置 API）。

图 2-1. 无线终端画面



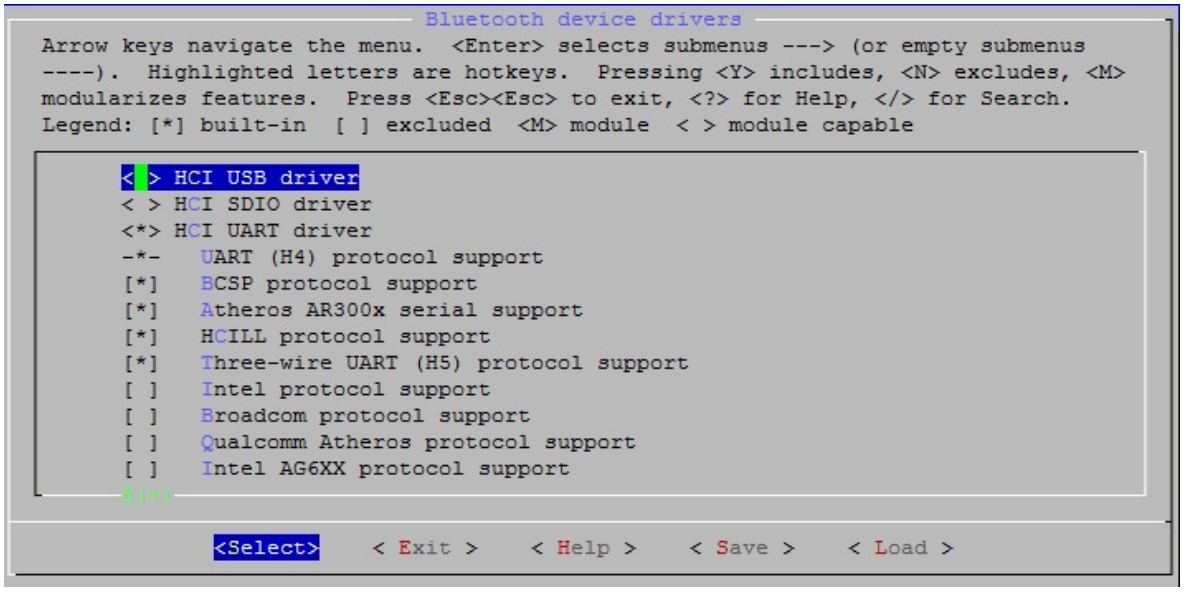
3. 转至 Networking Support > Bluetooth subsystem support（网络支持 > 蓝牙子系统支持），以启用 BLE（见下图）。

图 2-2. 蓝牙子系统支持



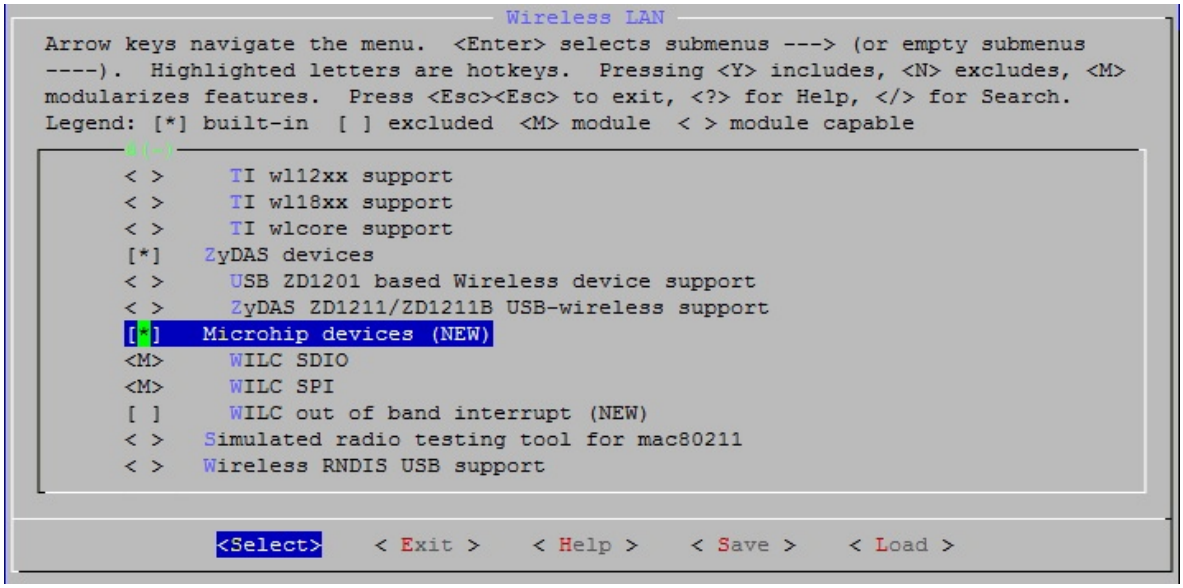
4. 转至蓝牙设备驱动程序以启用 HCI（见下图）。

图 2-3. 蓝牙设备驱动程序



5. 通过 Device Drivers > Network Device Support（设备驱动程序 > 网络设备支持）配置 ATWILC 驱动程序。选择所需配置，如下图所示。
   注： 如果在 linux_root/drivers/staging 下添加了驱动程序代码，则 Device Drivers > Staging drivers（设备驱动程序 > 暂存的驱动程序）下将包含 menuconfig 条目。

图 2-4. 无线 LAN

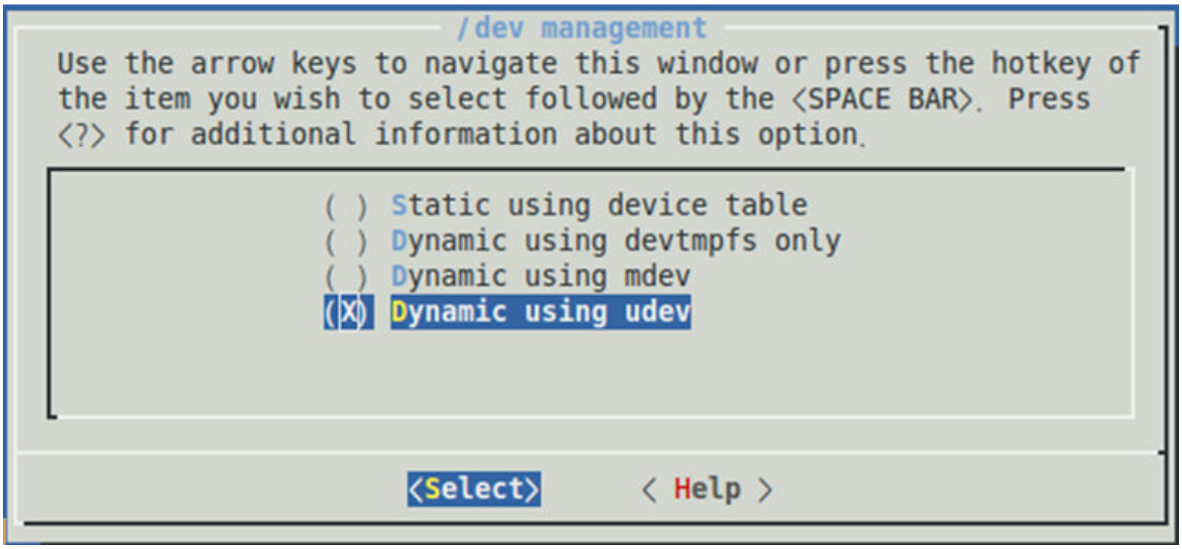


3. Buildroot 修改

用户必须执行以下修改才能启用 buildroot 的包。

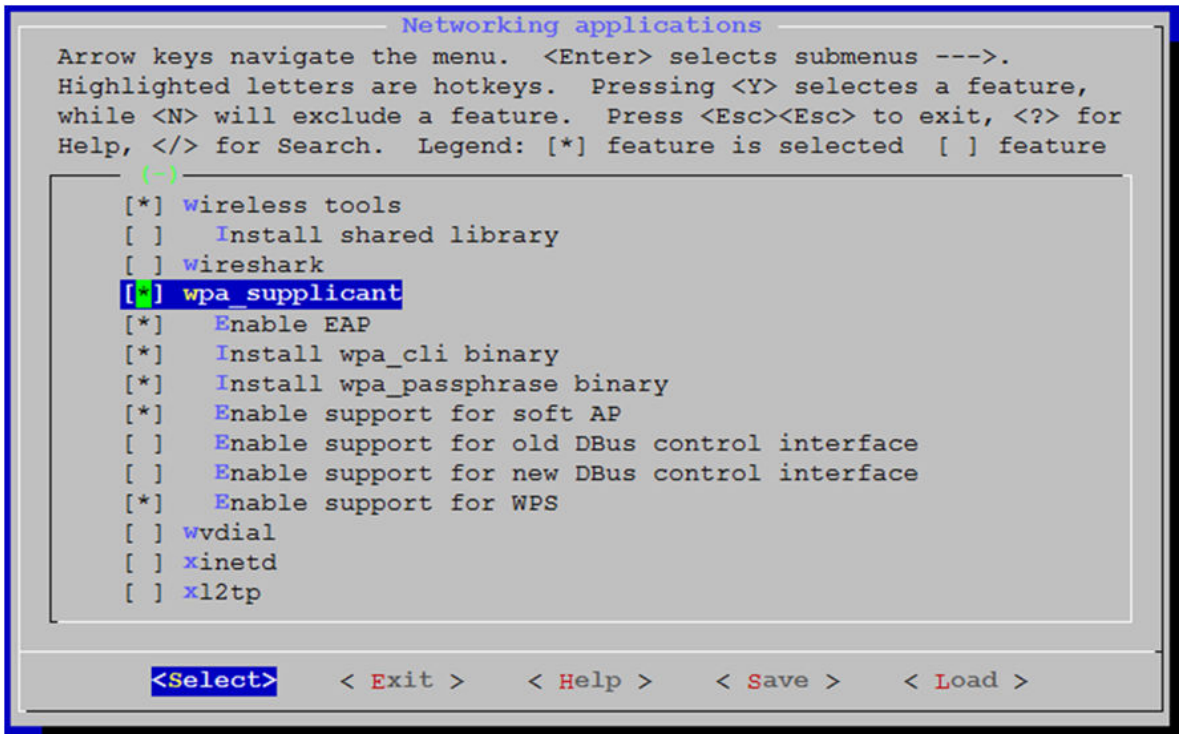
1. 转至 buildroot 的目录，然后输入`$ make menuconfig` 命令。
2. 启用通过 udev 进行动态/dev 管理。转至 `System configuration` > `dev management`（系统配置 > dev 管理），然后选择 `Dynamic using udev`（使用 udev 进行动态管理）。

图 3-1. /dev 管理画面



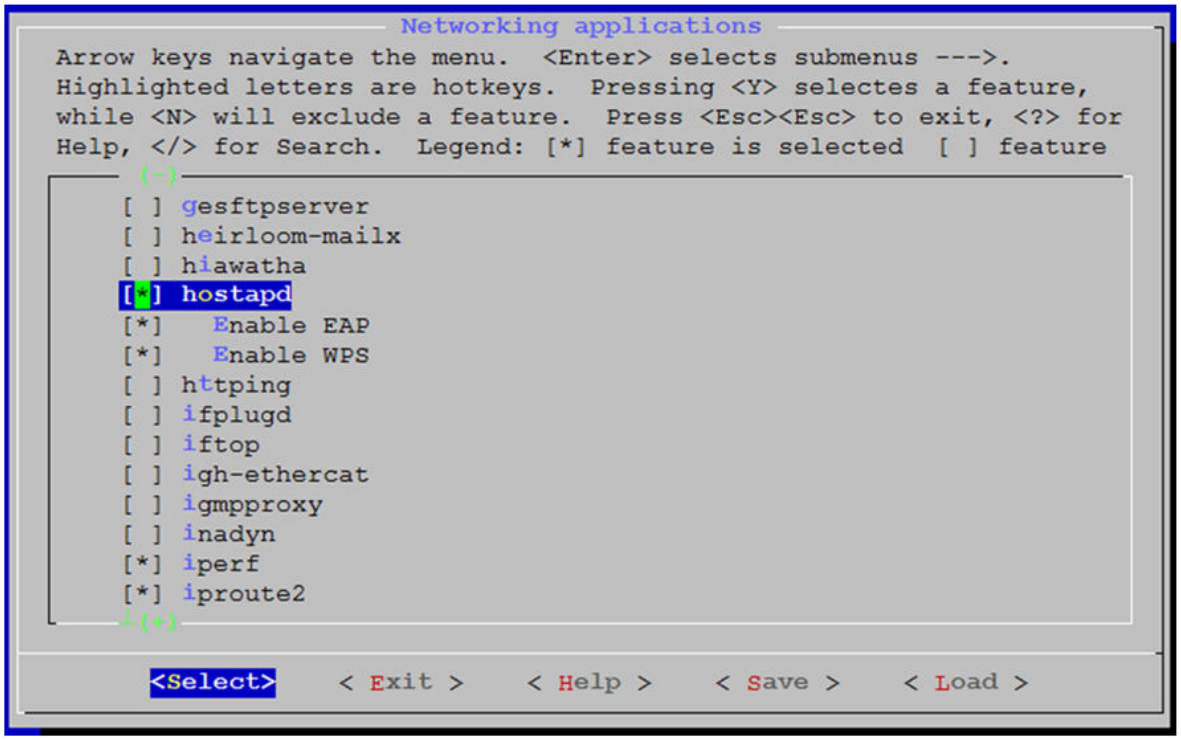
3. 通过 `Target package->Networking applications`（目标包 > 网络应用）启用 wpa_supplicant，然后选择 `wpa_supplicant`。

图 3-2. 网络应用——wpa_supplicant 画面



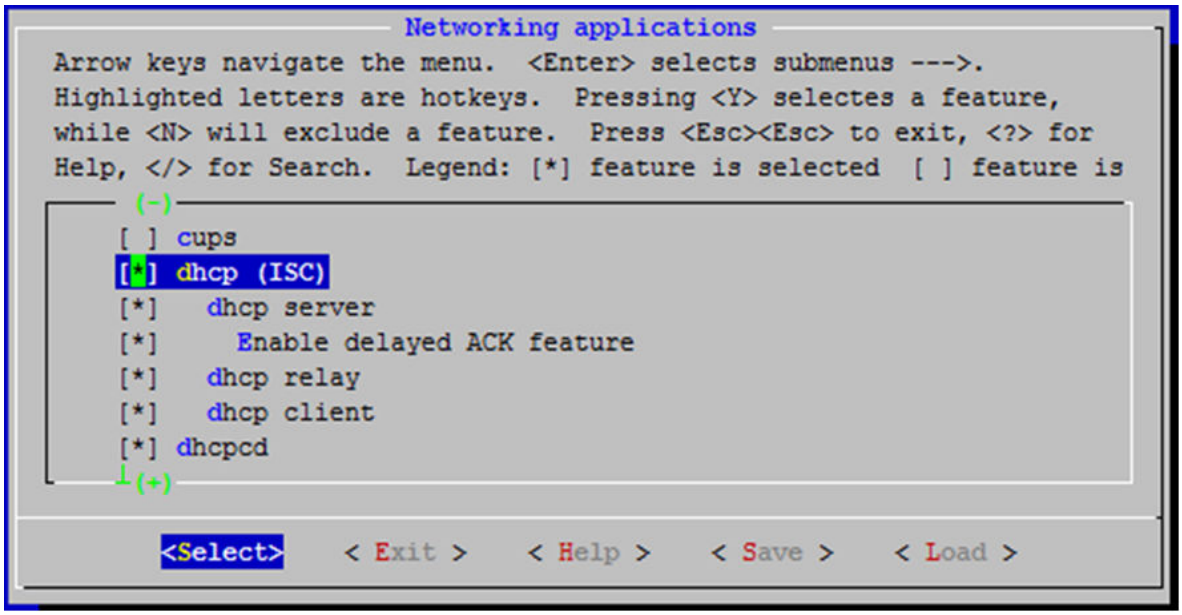
4. 转至 `Target package` > `Networking applications`，然后选择 `hostapd`。

图 3-3. menuconfig：启用 hostapd



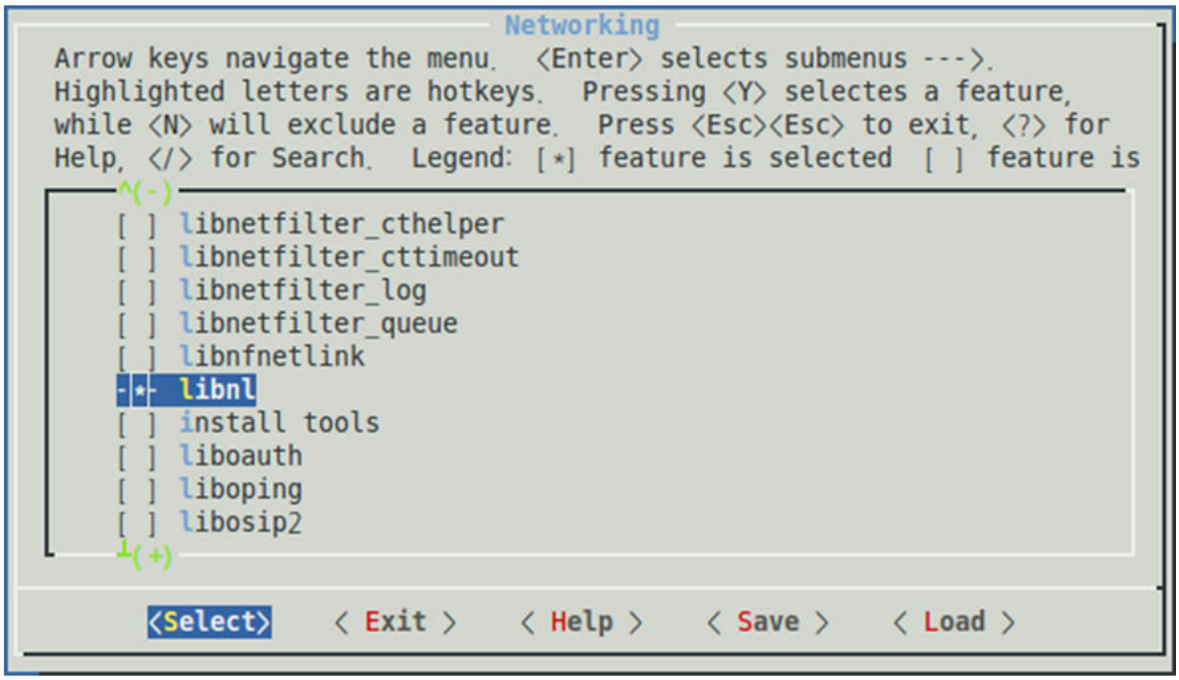
5. 在服务器和客户端之间分配 IP 地址。转至 `Target package` > `Networking applications`，然后选择 `dhcp`。

图 3-4. menuconfig：启用 dhcp



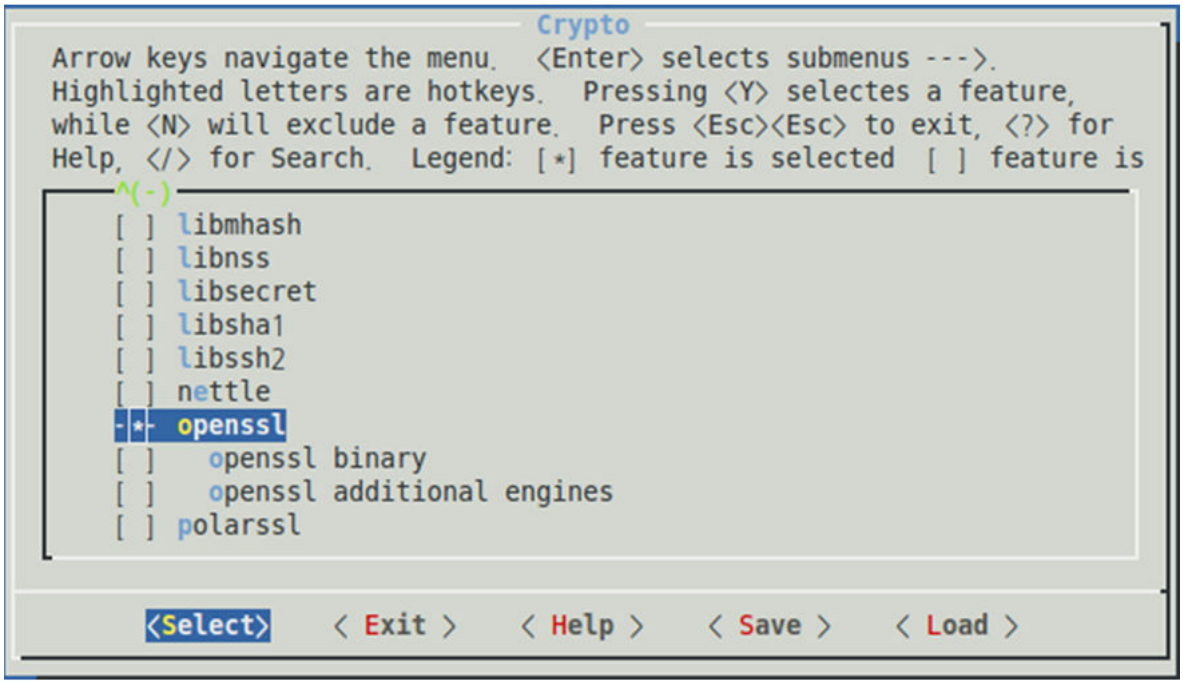
6. 转至 `Target package` > `Libraries` > `Networking`（目标包 > 库 > 网络），然后选择 `libnl`。

图 3-5. menuconfig：启用 libnl



7. 连接安全 AP。转至 `Target packages` > `Libraries`（目标包 > 库），然后选择 `Crypto`（加密）。

图 3-6. menuconfig：启用 openssl



3.1 使用 Build Root（编译根）选项启用 BlueZ 5.x 软件包

执行以下步骤，为 SAMA5D4 和 ATWILC3000 启用与蓝牙相关的软件包和工具。

1. 克隆 Buildroot-at91。有关克隆和编译 buildroot-at9 的更多详细信息，请参见 https://www.at91.com/linux4sam/bin/view/Linux4SAM/BuildRootBuild#How_to_build_Buildroot_for_AT91。

2. 配置 SAMA5D4 Xplained defconfig。
   转到 buildroot 所在的目录并输入以下命令：

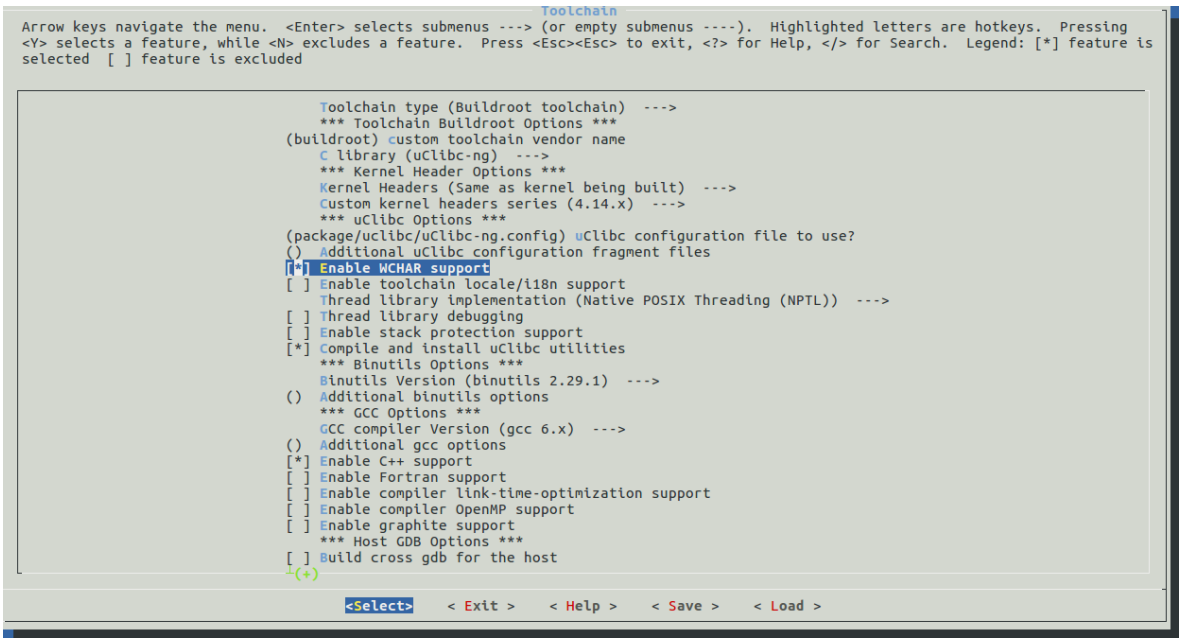
```
$make atmel_sama5d4_xplained_defconfig
```

3. 配置 menuconfig。
   转到 buildroot 所在的目录并输入以下命令：

```
$ make menuconfig
```

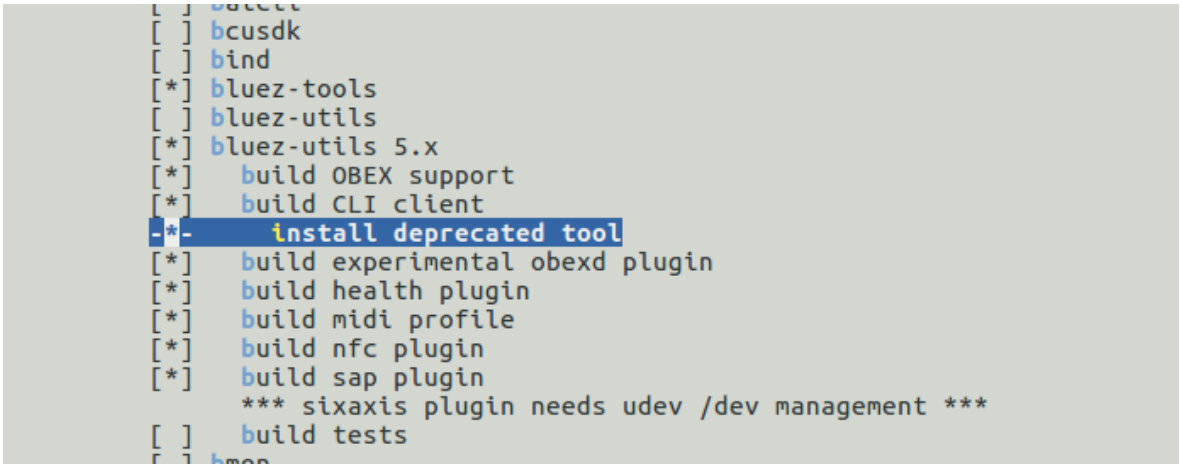
注： 必须启用 WCHAR 支持（**Toolchain > WCHAR**（工具链 > WCHAR））， 因为它依赖于 BlueZ 协议栈。

图 3-7. 启用 WCHAR 支持



4. 要启用 Bluez5 协议栈，导航到 **Target packages > Network Applications**（目标软件包 > 网络应用程序）并选择工具，如下图所示。

图 3-8. 启用 Bluez



注： 确保选中 **install deprecated tool**（安装已弃用工具），其中包括 hciattach 工具。此工具用于启用通过 UART 传输蓝牙。

5. 保存配置文件并运行 `make` 命令。
   成功完成后，将在`/buildroot-at91/output` 文件夹中生成 `rootfs.ubi` 文件。

4. 移植驱动程序

要将 Linux 驱动程序移植到另一个平台，需将依赖于硬件的 API 更改为适合新平台的另一种实现。

4.1 ATWILC 电源控制

主机使用 CHIP_EN 和 RESET_N 这两个引脚控制 ATWILC 器件的电源。如果这些引脚连接到主机的 GPIO 引脚，则应按照 4.5 通用 IO 中所述修改器件树文件。这样驱动程序可检索连接到 ATWILC CHIP_EN 和 RESET_N 的 GPIO 的信息。

其他一些硬件设计使用 RC 网络将 CHIP_EN 和 RESET_N 连接到电源，从而确保 ATWILC 器件通过主机上电。在这种情况下，这些 API 不是必需项，应加以修改以避免改动可能已用于其他功能的 GPIO 配置。

注： 当 ATWILC 的模块装入 Linux 内核后，它将与 ATWILC 的 SDIO/SPI 控制器进行通信。成功后，它将调用相应的探测函数来初始化驱动程序的其余部分。在 ATWILC 的驱动程序初始化之前，Linux 内核必须找到 SDIO/SPI 控制器。因此，在初始化之前，请确保 ATWILC 已上电。

4.1.1 ATWILC 掉电

要关闭芯片组，将 CHIP_EN 和 RESET_N 设为低电平。

4.1.2 ATWILC 上电

要启动芯片，请将 CHIP_EN 线设为高电平，然后延时 5 ms 再将 RESET_N 设为高电平。

下面提供了电源控制API的参考实现。

```
static void wilc_wlan_power(struct wilc *wilc, int power)
{
    struct gpio_desc *gpio_reset;
    struct gpio_desc *gpio_chip_en;

    pr_info("wifi_pm : %d\n", power);

    gpio_reset = gpiod_get(wilc->dt_dev, "reset", GPIOD_ASIS);
    if (IS_ERR(gpio_reset)) {
        dev_warn(wilc->dev, "failed to get Reset GPIO, try default\r\n");
        gpio_reset = gpio_to_desc(GPIO_NUM_RESET);
        if (!gpio_reset) {
            dev_warn(wilc->dev,
                 "failed to get default Reset GPIO\r\n");
            return;
        }
    } else {
        dev_info(wilc->dev, "succesfully got gpio_reset\r\n");
    }

    gpio_chip_en = gpiod_get(wilc->dt_dev, "chip_en", GPIOD_ASIS);
    if (IS_ERR(gpio_chip_en)) {
        gpio_chip_en = gpio_to_desc(GPIO_NUM_CHIP_EN);
        if (!gpio_chip_en) {
            dev_warn(wilc->dev,
                 "failed to get default chip_en GPIO\r\n");
            gpiod_put(gpio_reset);
            return;
        }
    } else {
        dev_info(wilc->dev, "succesfully got gpio_chip_en\r\n");
    }

    if (power) {
        gpiod_direction_output(gpio_chip_en, 1);
        mdelay(5);
        gpiod_direction_output(gpio_reset, 1);
    } else {
        gpiod_direction_output(gpio_reset, 0);
        gpiod_direction_output(gpio_chip_en, 0);
    }
```

```
    gpiod_put(gpio_chip_en);
    gpiod_put(gpio_reset);
}
```

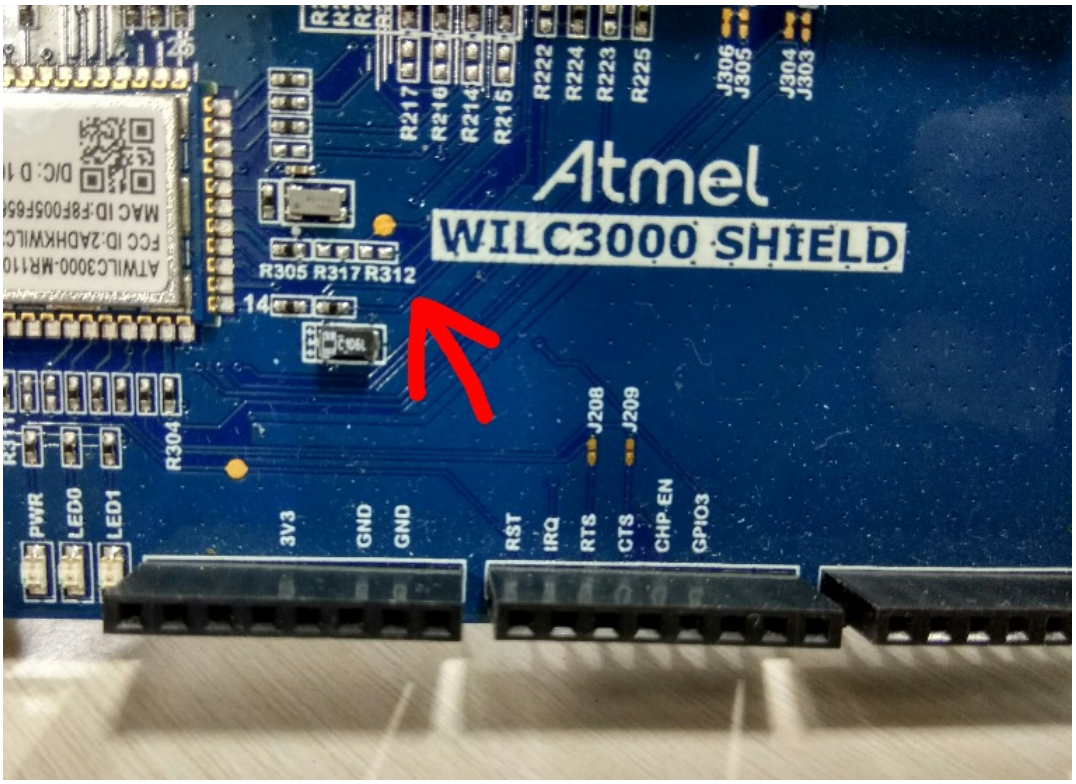
4.2 SDIO

4.2.1 重新扫描 SDIO 卡

ATWILC 器件使用 SDIO 总线连接到主机时，主机的 SDIO 控制器将初始化，然后询问 ATWILC 器件的 SDIO 控制器。在主机引导时会出现询问操作。如果在主机引导后有任何 SDIO 器件已上电，则必须触发重新扫描操作才能检测到该器件。

ATWILC 器件应在启动时默认上电。对于 ATWILC3000 Shield 评估板，可以通过在下图 ATWILC3000 Shield 评估板上所示的位置安装大小约为 120 kΩ 的电阻 R312 来实现上述过程。

图 4-1. Shield 评估板上的 SDIO 电阻



提示： 如果 ATWILC 模块在系统引导期间自动上电，则不需要重新扫描 SDIO，因为会在系统引导期间执行 ATWILC SDIO 初始化和询问。

4.2.2 SDIO 卡检测

MMC 插槽有一根卡检测线，由 Linux 用来指示已插入 SDIO 卡，如果 ATWILC 通过 MMC 插槽连接（与使用 ATWILC1000 SD 评估板时类似），可使用这条线。但是，并非所有设置都会连接这条线。例如，ATWILC3000 Shield 评估板 + SAMA5D4 Xplained 设置使用 Arduino 调试头进行连接，不会用到卡检测 GPIO（PE3）。在这种情况下，必须解除该平台与卡检测线的依赖关系，这样才能检测 ATWILC 的 SDIO 控制器。

在 dts 文件的相应 MMC 条目下，移除 cd-gpios 条目，然后添加 non-removable 属性。以下是有关 at91-sama5d4_xplained.dts 的更改示例：

```
mmc1: mmc@fc000000 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_mmc1_clk_cmd_dat0 &pinctrl_mmc1_dat1_3 &pinctrl_mmc1_cd>;
        non-removable;
        vmmc-supply = <&vcc_mmc1_reg>;
        vqmmc-supply = <&vcc_3v3_reg>;
        status = "okay";
        slot@0 {
            reg = <0>;
            bus-width = <4>;
            cd-gpios = <&pioE 3 0>;
        };
    };
```

4.3 SPI

ATWILC 器件驱动程序通过传递已指定相应.compatible 参数的 Linux SPI 注册 API spi_register_driver() a .of_match_table 来识别用于通信的 SPI 端口。

ATWILC 器件的 spi_driver 在 wilc_spi.c 文件中的定义如下。

```
static const struct of_device_id wilc_of_match[] = {
    { .compatible = "microchip,wilc1000", },
    { .compatible = "microchip,wilc3000", },
    { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, wilc_of_match);
static const struct dev_pm_ops wilc_spi_pm_ops = {
     .suspend = wilc_spi_suspend,
     .resume    = wilc_spi_resume,
        };

static struct spi_driver wilc_spi_driver = {
    .driver = {
        .name = MODALIAS,
        .of_match_table = wilc_of_match,
        .pm = &wilc_spi_pm_ops,
    },
    .probe =  wilc_bus_probe,
    .remove = wilc_bus_remove,
};
```

用户需确保已在对电路板进行描述的器件树文件中定义 of_device_id wilc_of_match[]中指定的.compatible 参数。

在以下示例中，我们对 kernel_tree/arch/arm/boot/dts 器件树 at91-sama5d4_xplained.dts 文件进行修改以匹配驱动程序。此外，还更改了 SPI 时钟以改善与主机处理器的通信。因此，应在相应的 SPI 器件节点下适当更改 spi-max-frequency 属性。

```
spi1: spi@fc018000 {
    cs-gpios = <&pioB 21 0>;
    status = "okay";
    wilc_spi@0 {
        compatible = "microchip,wilc1000", "microchip,wilc3000";
        reg = <0>;
        status = "okay";
    };
};
```

4.4 UART DMA

对于 ATWILC3000，用户必须确保在通用同步/异步收发器（Universal Synchronous/Asynchronous Receiver/Transmitter，USART）中使能直接存储器访问（Direct Memory Access，DMA）。

例如，对 kernel_tree/arch/arm/boot/dts 路径中的 SAMA5D4 sama5d4.dtsi 文件进行如下修改。

```
usart4: serial@fc010000 {
    compatible = "atmel,at91sam9260-usart";
    reg = <0xfc010000 0x100>;
    interrupts = <31 IRQ_TYPE_LEVEL_HIGH 5>;
    atmel,use-dma-rx;
    atmel,use-dma-tx;
    dmas = <&dma1
        (AT91_XDMAC_DT_MEM_IF(0) | AT91_XDMAC_DT_PER_IF(1)
        | AT91_XDMAC_DT_PERID(20))>,
           <&dma1
        (AT91_XDMAC_DT_MEM_IF(0) | AT91_XDMAC_DT_PER_IF(1)
        | AT91_XDMAC_DT_PERID(21))>;
    dma-names = "tx", "rx";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_usart4 &pinctrl_usart4_rts &pinctrl_usart4_cts>;
    clocks = <&usart4_clk>;
    clock-names = "usart";
    status = "disabled";
};
```

注： 可以根据应用要求为 Wi-Fi 使能 SDIO/SPI DMA。

4.5 通用 IO

ATWILC 器件的驱动程序使用 3 个连接到 CHIP_EN、RESET_N 和 IRQn 的 GPIO。要将驱动程序移植到新主机，应更新相应的 GPIO 编号。

驱动程序从器件树文件中读取相应的 GPIO 编号。如果在器件树文件中未找到 GPIO，驱动程序将使用 wilc_wlan.h 中定义的 GPIO 编号的静态定义。

```
    gpio_reset = gpiod_get(wilc->dt_dev, "reset", GPIOD_ASIS);
    if (IS_ERR(gpio_reset)) {
        dev_warn(wilc->dev,"failed to get Reset GPIO, try default\r\n");
        gpio_reset = gpio_to_desc(GPIO_NUM_RESET);
        if (!gpio_reset) {
            dev_warn(wilc->dev,
                 "failed to get default Reset GPIO\r\n");
            return;
        }
    } else {
        dev_info(wilc->dev, "succesfully got gpio_reset\r\n");
    }

    gpio_chip_en = gpiod_get(wilc->dt_dev, "chip_en", GPIOD_ASIS);
    if (IS_ERR(gpio_chip_en)) {
        gpio_chip_en = gpio_to_desc(GPIO_NUM_CHIP_EN);
        if (!gpio_chip_en) {
            dev_warn(wilc->dev,
                 "failed to get default chip_en GPIO\r\n");
            gpiod_put(gpio_reset);
            return;
        }
    } else {
        dev_info(wilc->dev, "succesfully got gpio_chip_en\r\n");
    }
```

应通过在 SPI 节点或 SDIO 节点中添加 GPIO 信息来完成对器件树文件的相应更改。下面的示例显示了针对 at91-sama5d4_xplained.dts 的此类更改。

```
mmc1: mmc@fc000000 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_mmc1_clk_cmd_dat0 &pinctrl_mmc1_dat1_3>;
        non-removable;
        vmmc-supply = <&vcc_mmc1_reg>;
        vqmmc-supply = <&vcc_3v3_reg>;
        status = "okay";
        wilc_sdio@0{
            compatible = "microchip,wilc1000", "microchip,wilc3000", "atmel,wilc_sdio";
```

```
            reset-gpios = <&pioB 28 0>;
            chip_en-gpios = <&pioC 30 0>;
            irq-gpios = <&pioC 27 0>;
            status = "okay";
            reg = <0>;
            bus-width = <4>;
        };
    };

    spi1: spi@fc018000 {
        cs-gpios = <&pioB 21 0>;
        status = "okay";
        wilc_spi@0 {
            compatible = "microchip,wilc1000", "microchip,wilc3000";
            reg = <0>;
            reset-gpios = <&pioB 28 0>;
            chip_en-gpios = <&pioC 30 0>;
            irq-gpios = <&pioC 27 0>;
            status = "okay";
        };
    };
```

5. 供应商特定的 HCI 命令

ATWILC3000 蓝牙核心使用一些非标准主机控制器接口（Host Controller Interface，HCI）命令为主机提供扩展服务和选项。

5.1 更新 UART 参数命令

主机需要更改蓝牙控制器的 UART 设置时，可使用下表中提及的命令更改波特率和流控制选项。

表 5-1. UART 参数命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷	
1	OxFC53	5	波特率（4 字节）	流控制（1 字节）

5.2 更改 BD 地址

主机使用下表提及的命令更改 BD 地址。执行此命令后，执行标准复位命令（操作码 0x0c03）更新新地址。

表 5-2. 更改 BD 地址的命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷
1	0xFC54	6	BD 地址（6 字节）

5.3 写存储器

下表介绍了写存储器的命令结构。仅当引导 ROM 在主机控制器上运行时，才使用该命令将存储器块写入蓝牙控制器。其主要功能是将固件下载到控制器。

表 5-3. 写存储器命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷		数据块
1	0xFC52	8	地址（4 字节）	大小（4 字节）	要写入到存储器的数据块

5.4 供应商特定的复位

下表介绍了供应商特定的复位命令。仅在引导 ROM 在主机控制器上运行时，才使用该命令。此命令在新固件下载到蓝牙控制器的存储器后发出。

表 5-4. 供应商特定的复位命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷
1	0xFC55	0	N/A

5.5 读寄存器

下表介绍了读寄存器命令。可使用该命令通过 UART 从蓝牙控制器读取寄存器。

表 5-5. 读寄存器命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷		
1	0xFC01	6	寄存器地址（4 字节）	0x20	0x01

5.6 设置 BT TX 功率

下表介绍了设置 BT Tx 功率的命令结构。此命令用于将发送功率设置为特定级别。

表 5-6. 设置 BT Tx 功率命令结构

HCI 包指示符	操作码	参数长度	参数有效载荷	
1	0xFC3B	3	保留（2 字节）	Tx 级别（1 字节）

TX 级别可设为以下值之一：0、3、6、9、12、15 和 18。对于除这些推荐值之外的其他值，将向下取整为最接近的支持级别。固件可强制执行最高级别以符合 RF 规定。

6. 蓝牙固件下载

本章介绍的不同蓝牙固件下载方法仅适用于 ATWILC3000。

6.1 使用 SDIO 或 SPI

如 1.1.8 AT Pwr Dev 字符接口部分所述，可使用电源设备模块通过以下命令下载蓝牙固件。

```
echo BT_POWER_UP    > /dev/wilc_bt
echo BT_DOWNLOAD_FW > /dev/wilc_bt
```

6.2 使用 UART

蓝牙控制器有一个引导 ROM，其在器件上电后会立即开始执行。它用于接收固件，将其下载到指令存储器，然后触发控制器以开始执行该固件。引导 ROM 最初以 115200 bps 波特率运行，并禁止流控制。如果蓝牙固件已在 CPU 上运行，请勿下载固件。

6.2.1 固件下载检测

主机确定 ATWILC 单片机（Microprocessor Control Unit，MCU）是否正在运行从引导 ROM 下载或运行的固件，并确定是否要使用新版本的固件。主机通过标准的“读本地版本”HCI 命令 `Op Code 0x1001` 读取本地版本，并检查从蓝牙控制器回送的 HCI 事件包中的字节编号 7 来完成检测，具体如下：

- 如果字节 7 为 255，则引导 ROM 代码正在 CPU 上运行
- 如果字节 7 为 6，则固件代码正在 CPU 上运行

6.2.2 固件已下载

如果固件已下载，则主机可执行以下操作：

- 更改 UART 参数，如波特率和流控制选项
- 如果 BD 地址未存储在控制器上，则更改蓝牙控制器的 BD 地址
- 发送复位命令。如果 BD 地址已更改，则必须执行此操作

6.2.3 开始固件下载

下载过程分为以下几步：

1. 将标准复位命令 `Op Code 0x0c03` 发送到控制器。
2. 将控制器波特率提高到更高的波特率水平，以加快下载过程；此步骤通过向具有所需波特率的控制器发出供应商特定的 UART 参数命令来完成。此时禁止流控制。
3. 提高控制器的波特率。
4. 通过供应商特定的“写存储器”HCI 命令开始将固件映像下载到控制器 IRAM（起始地址为 `0x80000000`）。
5. 发送供应商特定的复位 HCI 命令，从执行引导 ROM 代码转向执行下载的固件代码。
6. 将主机 UART 的波特率和流控制设置更新为初始设置，固件将使用此设置运行并与主机 UART 的设置进行通信。
7. 提高控制器的波特率和流控制水平，以满足运行时工作环境的要求。
8. 提高主机 UART 设置以与控制器设置相匹配。
9. 如果 BD 地址未存储在蓝牙控制器上，则更改控制器的 BD 地址。
10. 将标准 HCI 复位命令发送到控制器以使用新的 BD 地址。

7. 暂停/恢复

本章介绍了暂停和恢复机制所需的更改。

7.1 主机唤醒

主机处于暂停模式时，控制器使用主板上提供的唤醒 GPIO 在发生特定事件时唤醒主机。可使用以下方法识别唤醒 GPIO：

- 如果主机使用包含带内中断的 SDIO 总线，WILC 将通过 IRQ 使用单独的中断线来唤醒主机。
- 如果主机使用包含带外中断的 SDIO 总线或 SPI 总线，则两条总线均会使用 IRQ 线向主机指示新事件，并在主机处于暂停模式时唤醒主机。有关如何配置连接到 ATWILC IRQ 引脚的主机 GPIO 编号的更多信息，请参见 4.5 通用 IO。

以下示例显示如何通过修改 `at91-sama5d4_xplained.dts` 文件（dts 文件）将 PC27 的 GPIO 配置为 SAMA5D4 的唤醒 GPIO，具体如下：

修改 `pinctrl@fc06a000` 块中的带下划线文本，以将 PC27 定义为 GPIO：

```
pinctrl@fc06a000 {
board {
pinctrl_key_gpio1: key_gpio_1 {
        atmel,pins =
            <AT91_PIOC 27 AT91_PERIPH_GPIO AT91_PINCTRL_PULL_UP_DEGLITCH>;
        };    };    };
```

其中，`pinctrl@fc06a000` 是本身属于“ahb”块的“apb”块的一部分。

然后，修改 `gpio_keys` 块中的带下划线文本，以将 PC27 定义为高电平有效的唤醒 GPIO：

```
gpio_keys {    compatible = "gpio-keys";
        #address-cells = <1>;
        #size-cells = <0>;
pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_key_gpio>;
        pinctrl-1 = <&pinctrl_key_gpio1>;
pb_user1 {
            label = "pb_user1";
            gpios = <&pioE 8 GPIO_ACTIVE_HIGH>;
            linux,code = <0x100>;
            gpio-key,wakeup;    };
pb_suspend_resume_wakeup {
            label = "pb_suspend_resume_wakeup";
            gpios = <&pioC 27 GPIO_ACTIVE_HIGH>;
            linux,code = <0x100>;
            gpio-key,wakeup;};      };
```

8. 附录 A——装入 ATWILC 模块

引导内核后，如果 ATWILC 模块未与内核一起编译，则调用 modprobe/insmod 命令以装入 wilc-spi 或 wilc-sdio 模块。SDIO 和 SPI 接口几乎遵循相同的顺序，具体说明如下。

Modprobe/insmod 命令用于从 Linux 内核添加或删除模块。modprobe 命令用于检查模块目录/lib/modules/中的所有模块，而 insmod 命令用于检查作为参数提到的驱动程序的根目录。

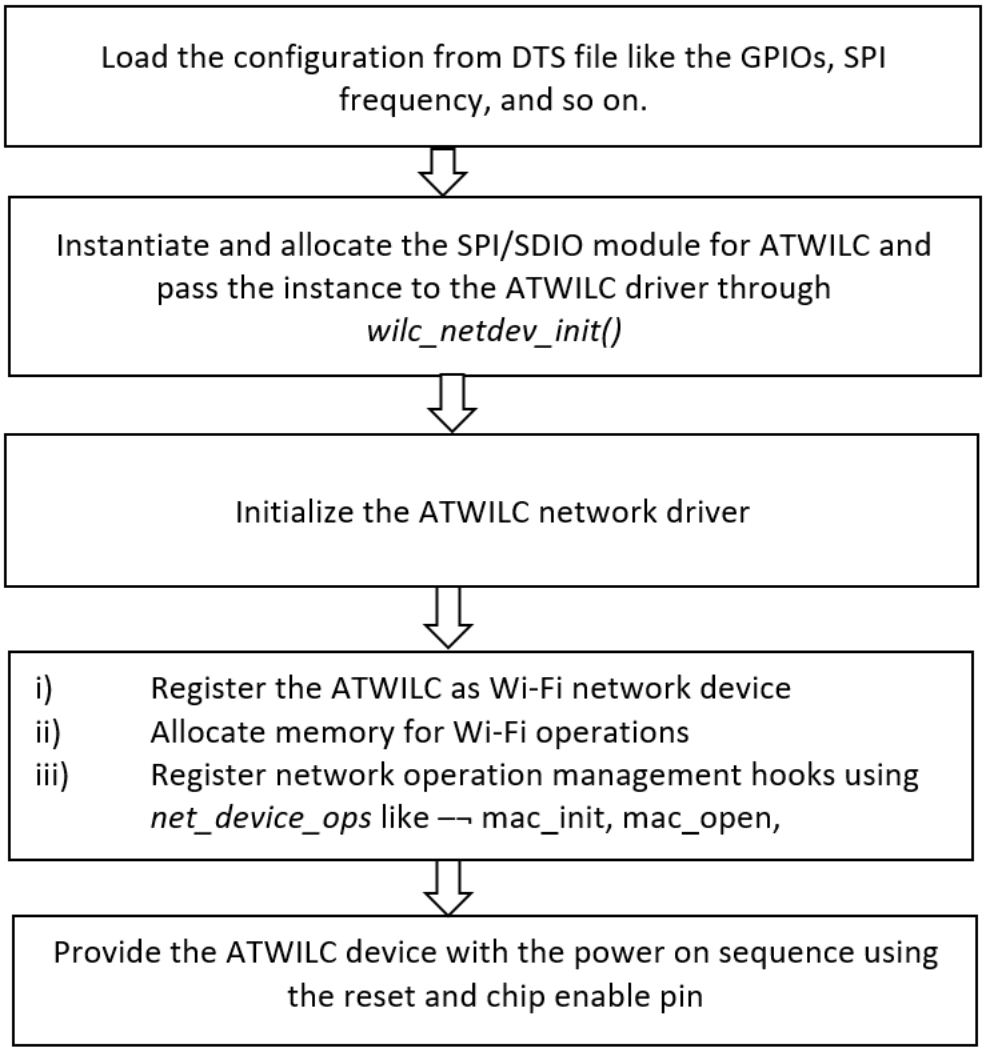
在编译期间，编译过程将从所有驱动程序中提取此信息。装入模块后，会将驱动程序中的设备与设备树文件中声明的设备进行比较。

```
$ modprobe wilc-spi/ modprobe wilc-sdio
```

接收到 modprobe 命令后，将装入驱动程序并将从 SPI 的 module_spi_driver()*函数或 SDIO 的 module_driver()函数开始执行。该函数将传递到内核，即包含 device_id、探测函数和删除函数等信息的结构。在 DTS 文件中发现器件 ID 匹配后，将调用探测函数。将分别为 SPI/SDIO 接口调用 wilc_bus_probe()或 linux_sdio_probe()函数。探测函数对模块执行提前初始化，并向内核注册器件 ATWILC。

下图给出了探测函数执行的操作顺序。

图 8-1. 探测函数的顺序



上图总结了探测操作，下图给出了成功日志。

图 8-2. modprobe/insmod 日志

```
# insmod wilc-sdio.ko
wilc_sdio: module is from the staging directory, the quality is unknown, you have been warned.
(unnamed net_device) (uninitialized): INFO [wilc_create_wiphy]Registering wifi device
(unnamed net_device) (uninitialized): INFO [wilc_wfi_cfg_alloc]Allocating wireless device
(unnamed net_device) (uninitialized): INFO [wilc_create_wiphy]Successful Registering
(unnamed net_device) (uninitialized): INFO [wilc_create_wiphy]Registering wifi device
(unnamed net_device) (uninitialized): INFO [wilc_wfi_cfg_alloc]Allocating wireless device
(unnamed net_device) (uninitialized): INFO [wilc_create_wiphy]Successful Registering
wilc_sdio mmc0:0001:1: WILC got 60 for gpio_reset
wilc_sdio mmc0:0001:1: WILC got 94 for gpio_chip_en
wilc_sdio mmc0:0001:1: WILC got 91 for gpio_irq
wifi_pm : 0
wifi_pm : 1
wilc_sdio mmc0:0001:1: Driver Initializing success
```

9. 附录 B——ATWILC SDIO 通信

SDIO 协议通过 SD 总线提供通信，并以命令和数据比特流为基础，这些比特流由启动位启动，由停止位终止。

命令是用于启动操作的令牌。命令从主机发送到单个卡（寻址命令）或所有已连接的卡（广播命令）。命令在 CMD 线上以串行方式传输。

响应是从被寻址的卡或所有已连接的卡（同步）发送到主机的令牌，作为对先前收到的命令的应答。响应在 CMD 线上以串行方式传输。

数据可以在卡和主机之间来回传输。数据通过数据线传输。

与 SD 卡之间的数据传输以块的形式进行。数据块始终后跟 CRC 位。系统定义了单个和多个块操作。

10. 附录 C——ATWILC SDIO 协议示例

成功装入 ATWILC 模块后，输入 `if config wlan0 up` 命令以启动 Wi-Fi 接口。该操作将初始化 ATWILC SDIO，读取芯片 ID，下载 Wi-Fi 固件并启动 WLAN 接口。

调用 `if config wlan0 up` 命令后，将先调用 `wilc_mac_open()` 函数，此函数中会调用 `wilc_wlan_initialize()` 来初始化 WLAN 接口，而 `wilc_wlan_init` 将调用相应的 `hif_init()` 来启动 HIF 层。如果使用 SDIO 接口，则调用 `sdio_init()`；如果使用 SPI 接口，则调用 `wilc_spi_init()`。本章介绍初始化 ATWILC3000 中的 WLAN 接口时 SDIO 数据包的序列。

根据 ATWILC 的驱动程序实现，`sdio_init()` 函数中发送的第一条命令如下。

图 10-1. 使能代码存储区（Code Storage Area，CSA）

```
/**
 *      function 0 csa enable
 **/
cmd.read_write = 1;
cmd.function = 0;
cmd.raw = 1;
cmd.address = 0x100;
cmd.data = 0x80;
ret = wilc_sdio_cmd52(wilc, &cmd);
if (ret) {
    dev_err(&func->dev, "Fail cmd 52, enable csa...\n");
    goto fail;
}
```

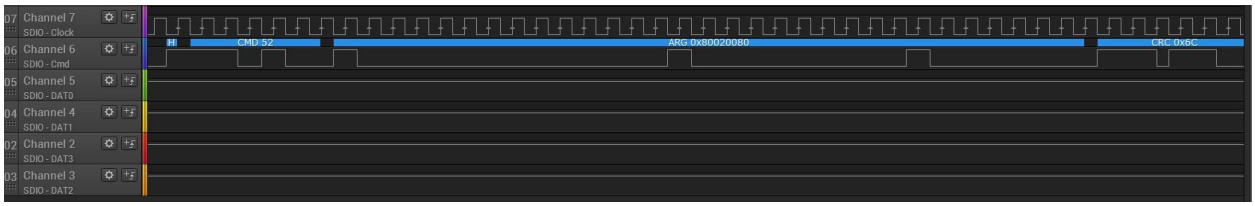
CSA 使能位控制此函数对代码存储区的访问。如果该位清零，则将阻止对 CSA 的任何读/写访问。如果该位置 1，则会允许对 CSA 的访问。复位后该位清零。代码会将函数 0 的函数基本寄存器（Function Basic Register，FBR）的第 7 位置 1 以使能 CSA，寄存器地址为 0x100。

图 10-2. 地址为 0x100 的函数基本寄存器

Address	7	6	5	4	3	2	1	0
0x100	Function 1 CSA enable	Function 1 supports CSA	RFU	RFU	Function 1 Standard SDIO Function interface code			

相应的 SDIO 日志如下。

图 10-3. 来自主机的使能 CSA 命令



DIR：来自主机，CMD：0x34，ARG：0x80020080，CRC：0x6C。

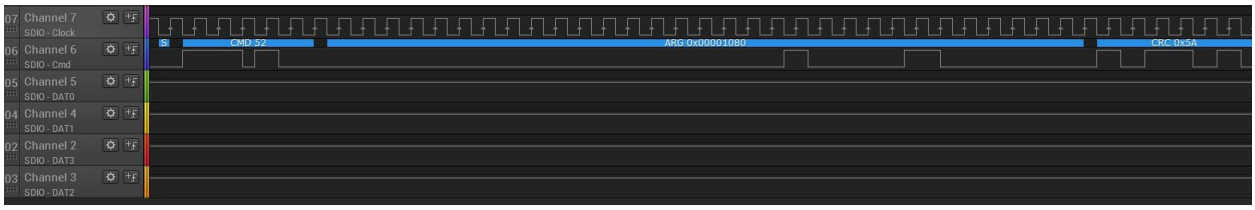
根据 CMD52 格式：

- 命令以启动位 0 开始
- 方向为 1，即来自主机
- CMD52 标识符为 110100b
- 参数值为 0x80020080，其 R/W 标志为 1

- 函数编号为 0
- RAW 标志为 0
- 寄存器地址为 0x10
- 写入数据 0x80（该数据会将上述寄存器的 CSA 使能位置 1）
- CRC 为 0x6C
- 结束位为 1。

ATWILC 的响应如下。

图 10-4. 来自从机的使能 CSA 响应



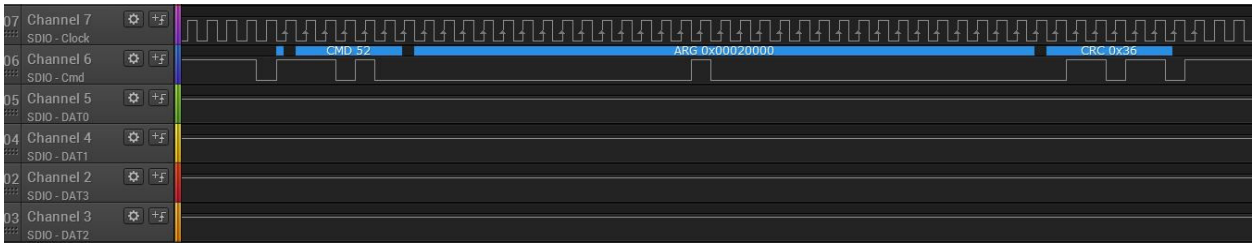
DIR：来自从机，CMD：0x34，ARG：0x1080，CRC：0x5A

根据 CMD52 响应 R5：

- 启动位为 0
- 方向为 0，即从卡到主机
- CMD52 标识符为 110100b
- 参数值为 0x00001080，其响应标志为 0x10
- 函数编号为 0
- RAW 标志为 0
- 寄存器地址为 0x10
- 写入数据 0x80（与前一个数据包中发送的数据相同）
- CRC 为 0x5A
- 结束位为 1

RAW 标志置 1 后，将立即读取同一寄存器以确任该位是否置 1。来自主机的读取命令如下。

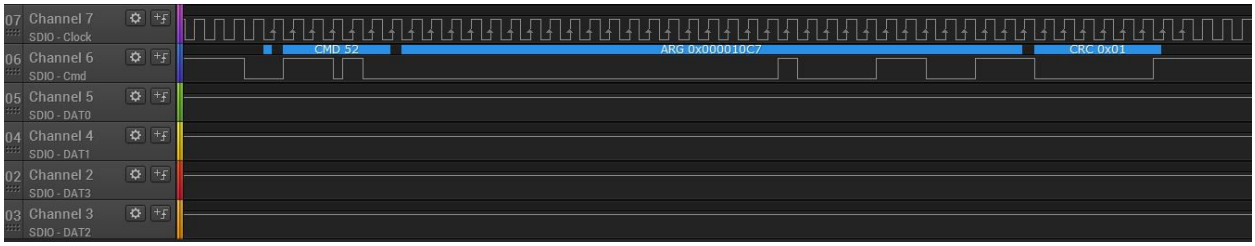
图 10-5. 来自主机的读取 CSA 命令



DIR：来自主机，CMD：0x34，ARG：0x20000，CRC：0x36

ATWILC 的响应如下。

图 10-6. 来自从机的读取 CSA 响应



DIR：来自从机，CMD：0x34，ARG：0x10C7，CRC：0x01。从上述日志可知，C7 将以数据形式返回，其中 CSA 使能位置 1。

在公共 I/O 区域（Common I/O Area，CIA）的卡通用控制寄存器（Card Common Control Register，CCCR）中将函数 0 的块大小设置为 512，寄存器地址为 0x10-0x11。

图 10-7. 地址为 0x10-0x11 的 CCCR 寄存器

0x10-0x11	FN0 Block Size	I/O block size for Function 0

图 10-8. 来自主机的设置函数 0 块大小命令

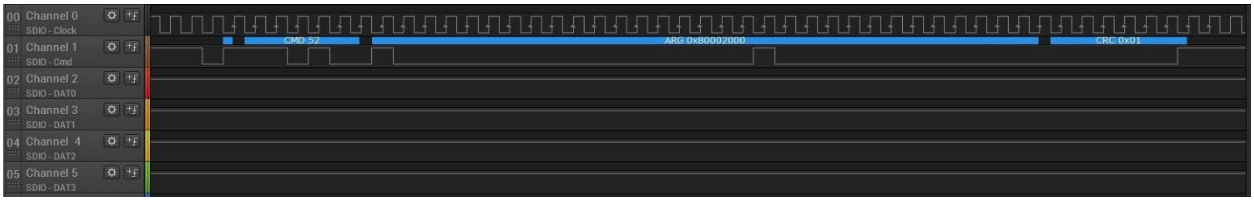
```
/**
 *      function 0 block size
 **/
if (!sdio_set_func0_block_size(wilc, WILC_SDIO_BLOCK_SIZE)) {
    dev_err(&func->dev, "Fail cmd 52, set func 0 block size...\n");
    goto fail;
}
g_sdio.block_size = WILC_SDIO_BLOCK_SIZE;
```

上述函数以两条命令的形式从主机发送。

```
static int sdio_set_func0_block_size(struct wilc *wilc, u32 block_size)
{
    struct sdio_func *func = dev_to_sdio_func(wilc->dev);
    struct sdio_cmd52 cmd;
    int ret;

    cmd.read_write = 1;
    cmd.function = 0;
    cmd.raw = 0;
    cmd.address = 0x10;
    cmd.data = (u8)block_size;
    ret = wilc_sdio_cmd52(wilc, &cmd);
    if (ret) {
        dev_err(&func->dev, "Failed cmd52, set 0x10 data...\n");
        goto fail;
    }

    cmd.address = 0x11;
    cmd.data = (u8)(block_size >> 8);
    ret = wilc_sdio_cmd52(wilc, &cmd);
    if (ret) {
        dev_err(&func->dev, "Failed cmd52, set 0x11 data...\n");
        goto fail;
    }

    return 1;
fail:
    return 0;
}
```

上述命令的 SDIO 日志如下。

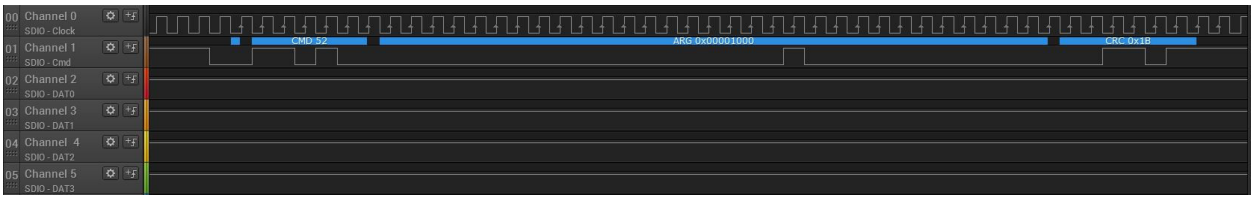
图 10-9. 来自主机的设置函数 0 块大小命令



DIR：来自主机，CMD：0x34，ARG：0x80002000，CRC：0x01。

ATWILC 对上述命令的响应如下。该命令将以数据 0x00（512 的低字节，0x0200）的形式发送到地址 0x10。

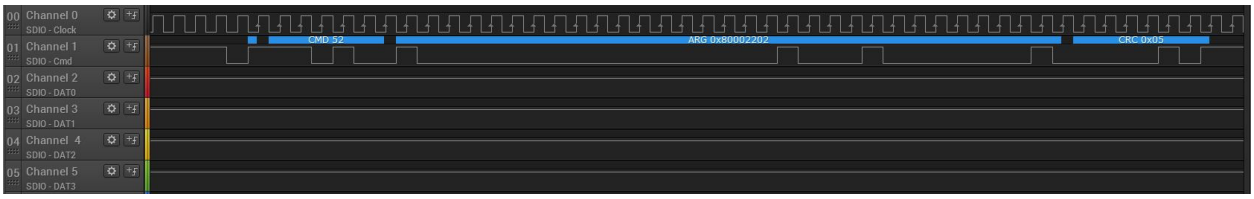
图 10-10. 来自从机的设置函数 0 块大小响应



DIR：来自从机，CMD：0x34，ARG：0x1000，CRC：0x1B。

下一个 CMD52 将以数据 0x02（512 的高字节，0x0200）的形式发送到地址 0x11。

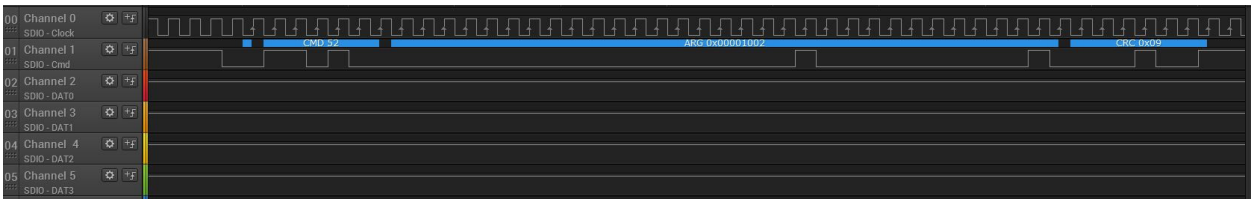
图 10-11. 来自主机的设置函数 0 块大小命令



DIR：来自主机，CMD：0x34，ARG：0x80002202，CRC：0x05。

ATWILC 对上述命令的响应如下。

图 10-12. 来自从机的设置函数 0 块大小响应



DIR：来自从机，CMD：0x34，ARG：0x1002，CRC：0x09

成功设置函数 0 块大小后，通过将卡通用控制寄存器（地址寄存器为 0x02，数据寄存器为 0x02）中的 IOE1 位（I/O 使能）置 1 来使能函数 1。

图 10-13. 地址为 0x02 和 0x03 的 CCCR 寄存器

0x02	I/O Enable	IOE7	IOE6	IOE5	IOE4	IOE3	IOE2	IOE1	RFU
0x03	I/O Ready	IOR7	IOR6	IOR5	IOR4	IOR3	IOR2	IOR1	RFU

图 10-14. 使能函数 1 I/O 寄存器

```
/**
 *      enable func1 IO
 **/
cmd.read_write = 1;
cmd.function = 0;
cmd.raw = 1;
cmd.address = 0x2;
cmd.data = 0x2;
ret = wilc_sdio_cmd52(wilc, &cmd);
if (ret) {
    dev_err(&func->dev,
        "Fail cmd 52, set IOE register...\n");
    goto fail;
}
```

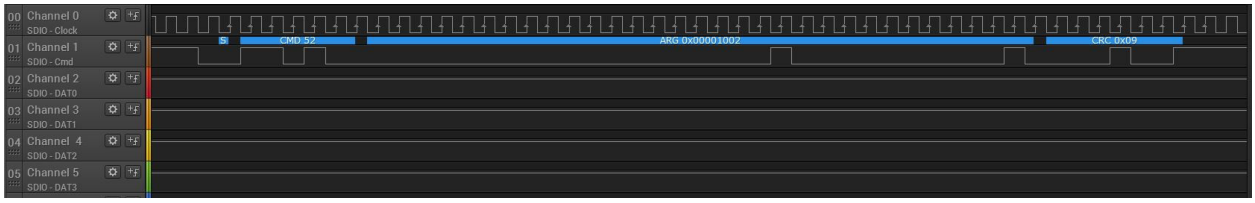
SDIO 日志如下。

图 10-15. 来自主机的使能函数 1 I/O 寄存器命令

DIR：来自主机，CMD：0x34，ARG：0x80000402，CRC：0x4D

ATWILC 的对上述命令的响应如下。

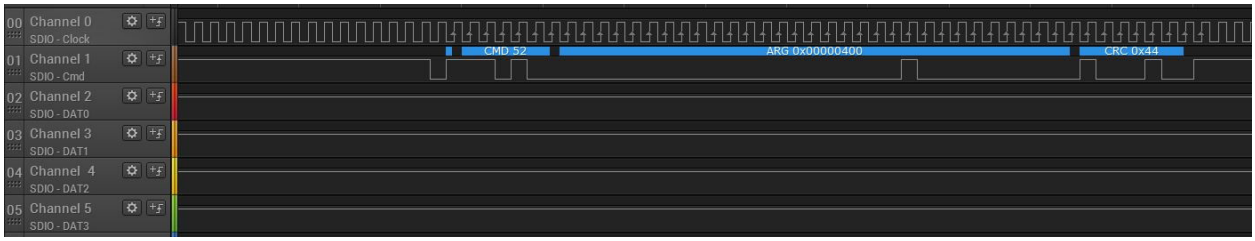
图 10-16. 来自从机的使能函数 1 I/O 寄存器响应



DIR：来自从机，CMD：0x34，ARG：0x1002，CRC：0x09

如果 RAW 标志置 1，则在使用 `sdio_writeb()` 写入后，将立即使用 `sdio_readb()` 函数读取地址为 0x02 的寄存器。

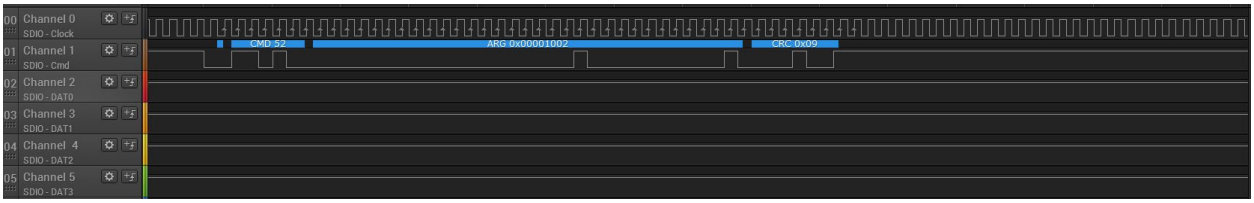
图 10-17. 来自主机的读取函数 1 I/O 寄存器命令



DIR：来自主机，CMD：0x34，ARG：0x400，其 R/W 标志为 0，函数编号为 0，RAW 标志为 0，寄存器地址为 0x02，读取数据为 0x00，CRC：0x44

ATWILC 的响应如下。

图 10-18. 来自从机的读取函数 1 I/O 寄存器响应



根据 CMD52 响应 R5，参数值为 0x00001002，其响应标志为 0x10，读取数据为 0x02。读取的数据与先前写入的数据相同。这意味着在 CCCR 中使能了函数 1（IE1）。

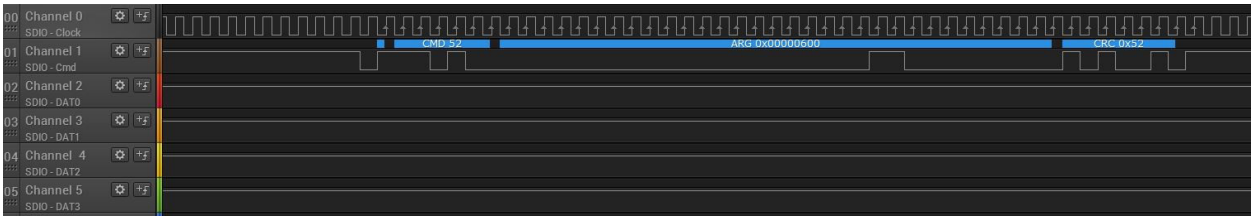
按上文所述使能函数 1 之后，通过读取地址为 0x03 的 CCCR 中的 IOR1 位（I/O 就绪）来确保函数 1 就绪。寄存器必须返回值 0x02 以将 IOR1 位置 1。

图 10-19. 读取函数 1 IOR 寄存器

```
/**
 *      make sure func 1 is up
 **/
cmd.read_write = 0;
cmd.function = 0;
cmd.raw = 0;
cmd.address = 0x3;
loop = 3;
do {
    cmd.data = 0;
    ret = wilc_sdio_cmd52(wilc, &cmd);
    if (ret) {
        dev_err(&func->dev,
            "Fail cmd 52, get IOR register...\n");
        goto fail;
    }
    if (cmd.data == 0x2)
        break;
} while (loop--);

if (loop <= 0) {
    dev_err(&func->dev, "Fail func 1 is not ready...\n");
    goto fail;
}
```

SDIO 日志如下。

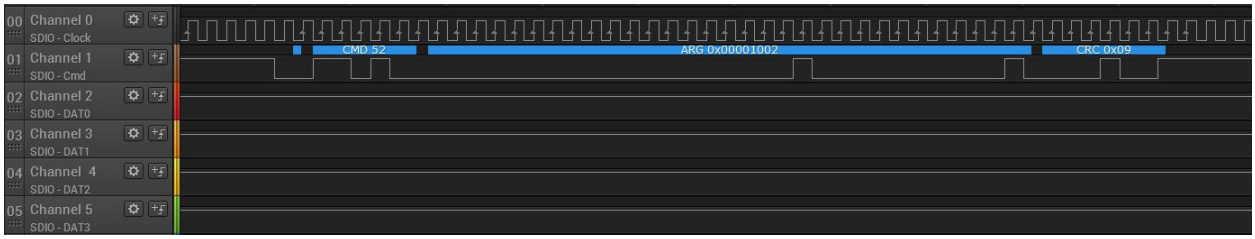
图 10-20. 来自主机的读取函数 1 IOR 寄存器命令



DIR：来自主机，CMD：0x34，ARG：0x600，CRC：0x52。

ATWILC 的响应如下。

图 10-21. 来自从机的读取函数 1 IOR 寄存器响应



DIR：来自从机，CMD：0x34，ARG：0x1002，CRC：0x09。

从上面的参数可知，读取数据返回 0x02，这意味着 IOR1 已置 1 且函数 1 已运行。

将函数 1 的块大小 `WILC_SDIO_BLOCK_SIZE` 设置为 512。地址 0x110-0x111 保存函数 1 的 I/O 块大小。该函数以两条 CMD52 命令的形式发送。

图 10-22. 设置函数 1 块大小

```
static int sdio_set_func1_block_size(struct wilc *wilc, u32 block_size)
{
    struct sdio_func *func = dev_to_sdio_func(wilc->dev);
    struct sdio_cmd52 cmd;
    int ret;

    cmd.read_write = 1;
    cmd.function = 0;
    cmd.raw = 0;
    cmd.address = 0x110;
    cmd.data = (u8)block_size;
    ret = wilc_sdio_cmd52(wilc, &cmd);
    if (ret) {
        dev_err(&func->dev, "Failed cmd52, set 0x110 data...\n");
        goto fail;
    }
    cmd.address = 0x111;
    cmd.data = (u8)(block_size >> 8);
    ret = wilc_sdio_cmd52(wilc, &cmd);
    if (ret) {
        dev_err(&func->dev, "Failed cmd52, set 0x111 data...\n");
        goto fail;
    }

    return 1;
fail:
    return 0;
}
```

SDIO 日志如下。

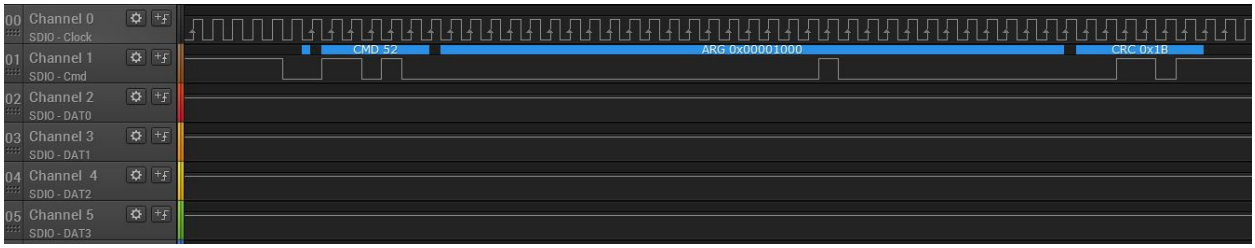
图 10-23. 来自主机的设置函数 1 块大小命令

DIR：来自主机，CMD：0x34，ARG：0x80022000，CRC：0x5F

上述命令将数据 0x00（512 的最后一个字节，0x0200）设置为地址 0x110。

ATWILC 的响应如下。

图 10-24. 来自从机的设置函数 1 块大小响应



DIR：来自从机，CMD：0x34，ARG：0x1000，CRC：0x1B

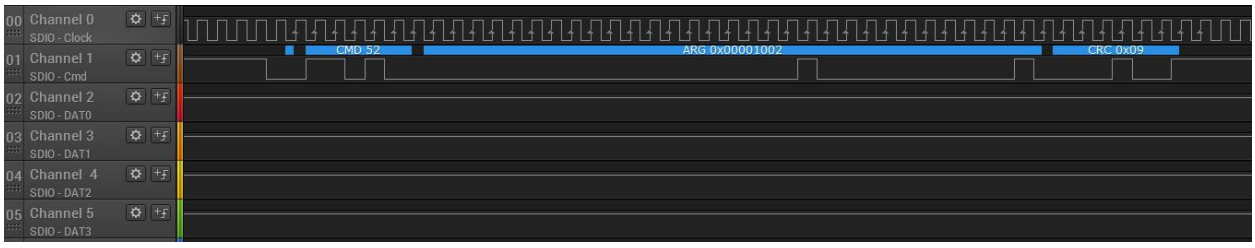
下一条命令将数据 0x02（512 的第一个字节，0x0200）设置为地址 0x111。

图 10-25. 来自主机的设置函数 1 块大小命令

DIR：来自主机，CMD：0x34，ARG：0x80022202，CRC：0x5B

ATWILC 的响应如下。

图 10-26. 来自从机的设置函数 1 块大小响应



DIR：来自从机，CMD：0x34，ARG：0x1002，CRC：0x09

设置函数 1 的中断允许。要将该 IEN1 置 1，需要将地址为 0x04 的 CCCR 寄存器中的 IENM 位置 1。

图 10-27. 地址为 0x04 的 CCCR 寄存器

0x04	Int Enable	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IENM

IENx：函数 x 的中断允许。如果该位清零，则该函数的任何中断都不会发送到主机。如果该位置 1，则当主中断允许位（bit 0）也置 1 时，该函数的中断才会发送到主机。

IENM：R/W 主中断允许位。如果该位清零，则该卡的任何中断都不会发送到主机。如果该位置 1，则任何函数的中断都会发送到主机。

图 10-28. 允许函数 0 中断

```
/**
 *      func 0 interrupt enable
 **/
cmd.read_write = 1;
cmd.function = 0;
cmd.raw = 1;
cmd.address = 0x4;
cmd.data = 0x3;
ret = wilc_sdio_cmd52(wilc, &cmd);
if (ret) {
    dev_err(&func->dev, "Fail cmd 52, set IEN register...\n");
    goto fail;
}
```

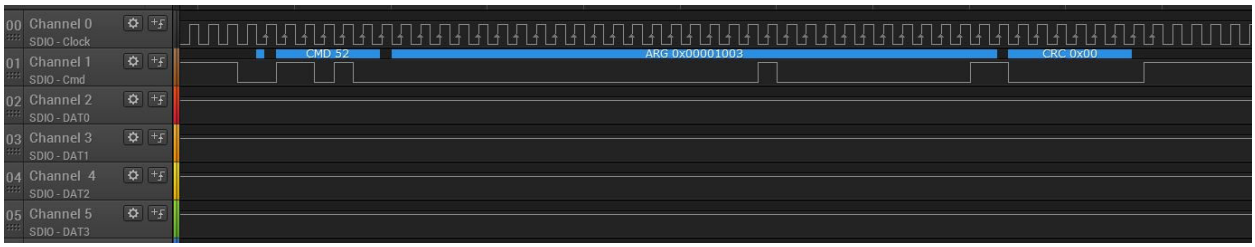
从主机发送到 ATWILC 的命令如下。

图 10-29. 来自主机的允许函数 0 中断命令

DIR：来自主机，CMD：0x34，ARG：0x80000803，CRC：0x30

ATWILC 的响应如下。

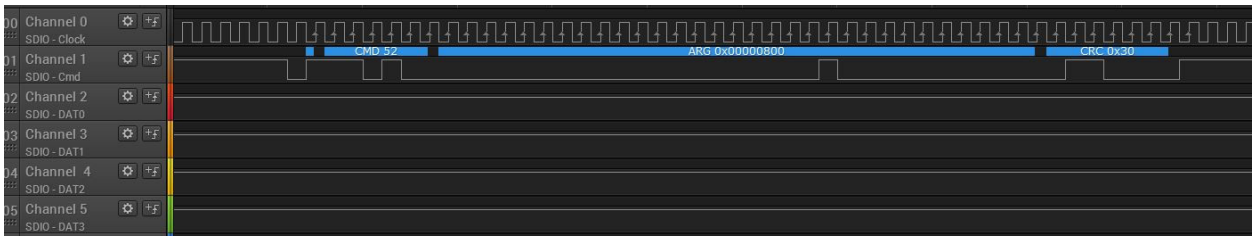
图 10-30. 来自从机的允许函数 0 中断响应



DIR：来自从机，CMD：0x34，ARG：0x1003，CRC：0x00

RAW 标志置 1 后，将再次读取同一寄存器以确保 IE1 和 IENM 位置 1。从主机发送到 ATWILC 的命令如下。

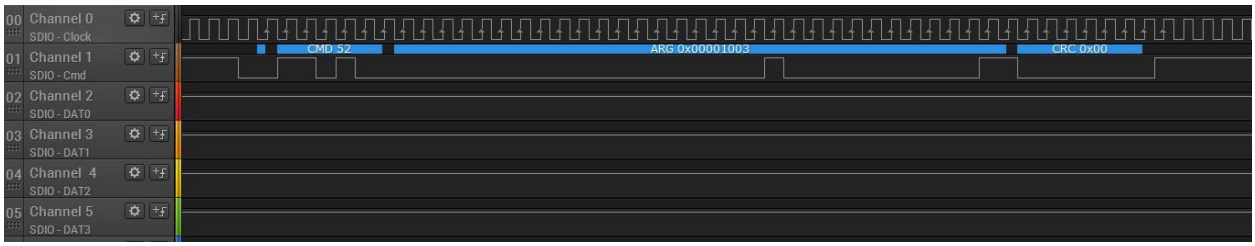
图 10-31. 来自主机的读取函数 0 中断允许命令



DIR：来自主机，CMD：0x34，ARG：0x800，CRC：0x30

ATWILC 的响应如下。

图 10-32. 来自从机的读取函数 0 中断允许响应



DIR：来自从机，CMD：0x34，ARG：0x1003，CRC：0x00。从上述日志可知，返回的数据为 0x03，表示 IE1 和 IENM 位已置 1。

从器件读取芯片 ID，以确认器件是 ATWILC1000 还是 ATWILC3000。

图 10-33. 从器件读取芯片 ID

```
u32 wilc_get_chipid(struct wilc *wilc, bool update)
{
    static u32 chipid;
    int ret;
    u32 tempchipid = 0;

    if (chipid == 0 || update) {
        ret = wilc->hif_func->hif_read_reg(wilc, 0x3b0000,
                                           &tempchipid);
        if (!ret)
            pr_err("[wilc start]: fail read reg 0x3b0000\n");
        if (!ISWILC3000(tempchipid)) {
            wilc->hif_func->hif_read_reg(wilc, 0x1000,
                                         &tempchipid);
            if (!ISWILC1000(tempchipid)) {
                chipid = 0;
                return chipid;
            }
            if(tempchipid < 0x1003a0){
                pr_err("WILC1002 isn't suported %x\n", chipid);
                chipid = 0;
                return chipid;
            }
        }
        chipid = tempchipid;
    }

    return chipid;
}
```

读取地址为 0x3b0000 的寄存器以查看芯片 ID 是否为 ATWILC3000，如果不是，则读取地址为 0x1000 的寄存器以获取 ATWILC1000 芯片 ID。`wilc->hif_func->hif_read_reg()`将基于使用的主机接口调用 `sdio_read_reg()`或 `wilc_spi_read_reg()`。

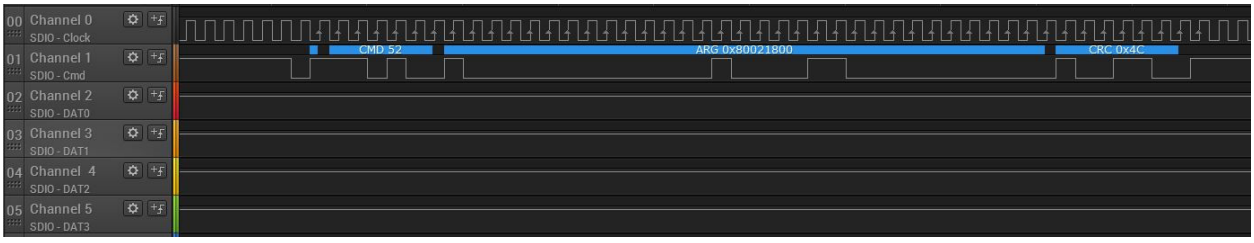
由于使用了 SDIO 接口，因此将调用 `sdio_read_reg()`。地址 0xF0-0xFF 保留给供应商惟一寄存器，CMD52 用于访问这些寄存器，而 CMD53 用于访问其他寄存器。

在进行读/写操作之前，需要先设置指向函数 CSA 的地址指针。这三个字节组成一个 24 位指针，指向 CSA 中需要读/写的字节。CSA 已通过之前的命令使能。地址 0x10C-0x10E 保存指向函数 1 代码存储区（CSA）的指针。

```
static int sdio_read_reg(struct wilc *wilc, u32 addr, u32 *data)
{
    struct sdio_func *func = dev_to_sdio_func(wilc->dev);
    int ret;

    if (addr >= 0xf0 && addr <= 0xff) {
        struct sdio_cmd52 cmd;

        cmd.read_write = 0;
        cmd.function = 0;
        cmd.raw = 0;
        cmd.address = addr;
        ret = wilc_sdio_cmd52(wilc, &cmd);
        if (ret) {
            dev_err(&func->dev,
                "Failed cmd 52, read reg (%08x) ...\n", addr);
            goto fail;
        }
        *data = cmd.data;
    } else {
        struct sdio_cmd53 cmd;

        if (!sdio_set_func0_csa_address(wilc, addr))
            goto fail;

        cmd.read_write = 0;
        cmd.function = 0;
        cmd.address = 0x10f;
        cmd.block_mode = 0;
        cmd.increment = 1;
        cmd.count = 4;
        cmd.buffer = (u8 *)data;

        cmd.block_size = g_sdio.block_size;
        ret = wilc_sdio_cmd53(wilc, &cmd);
        if (ret) {
            dev_err(&func->dev,
                "Failed cmd53, read reg (%08x)...\n", addr);
            goto fail;
        }
    }

    *data = cpu_to_le32(*data);
```

以下命令将地址 0x3b000000 设置为地址 0x10C-0x10E。

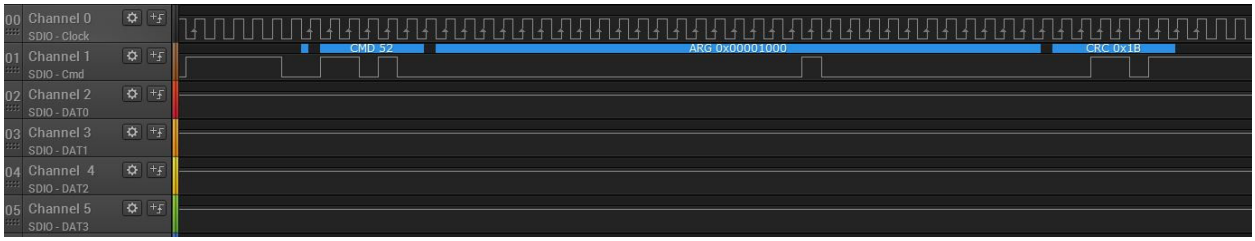
图 10-34. 来自主机的设置指向函数 1 代码存储区的指针命令



DIR：来自主机，CMD：0x34，ARG：0x80021800，CRC：0x4C。它将数据 0x00（0x3b00000 的最后一个字节）设置为地址 0x10C0。

ATWILC 的响应如下。

图 10-35. 来自从机的设置指向函数 1 代码存储区的指针响应



DIR：来自从机，CMD：0x34，ARG：0x1000，CRC：0x1B

下一条命令将数据 0x00（0x3b00000 的第二个字节）设置为地址 0x10D0。

图 10-36. 来自主机的设置指向函数 1 代码存储区的指针命令

DIR：来自主机，CMD：0x34，ARG：0x80021A00，CRC：0x5A

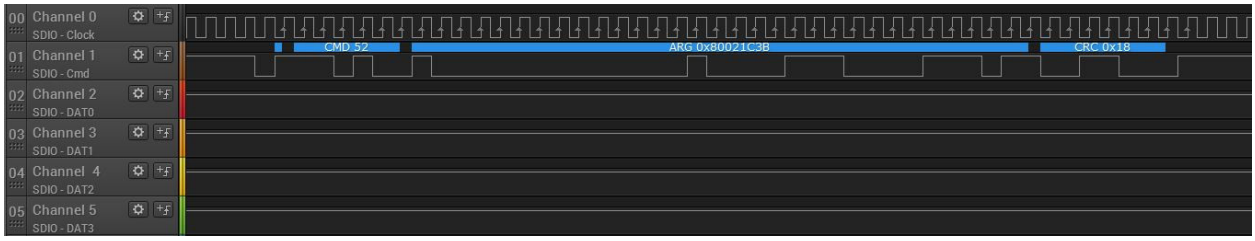
ATWILCC 的响应如下。

图 10-37. 来自从机的设置指向函数 1 代码存储区的指针响应

DIR：来自从机，CMD：0x34，ARG：0x1000，CRC：0x1B

下一条命令将数据 0x3B（0x3b00000 的第一个字节）设置为地址 0x10E0。

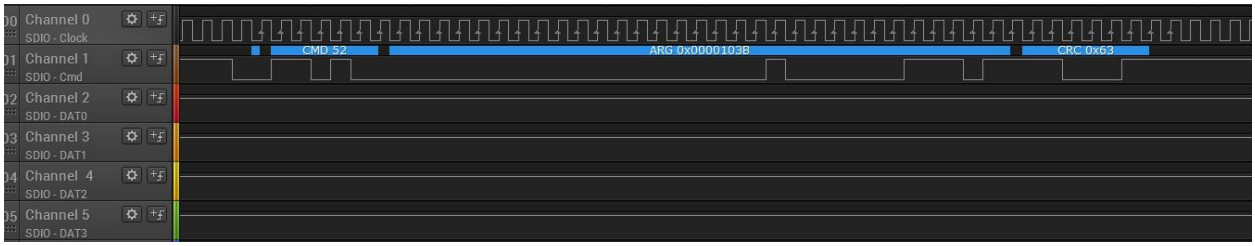
图 10-38. 来自主机的设置指向函数 1 代码存储区的指针命令



DIR：来自主机，CMD：0x34，ARG：0x80021C3B，CRC：0x18

ATWILC 的响应如下。

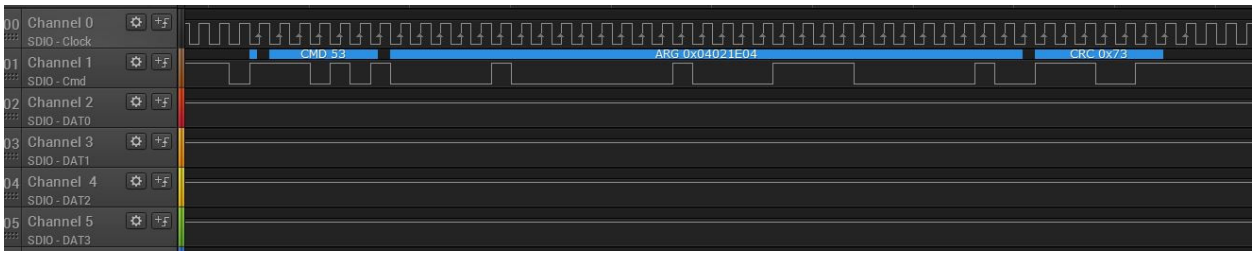
图 10-39. 来自从机的设置指向函数 1 代码存储区的指针响应



DIR：来自从机，CMD：0x34，ARG：0x103B，CRC：0x63

通过字节传输（块模式 = 0）发送 CMD53 以从上面设置的地址读取数据。地址 0x10F 是函数 1 代码存储区（CSA）的数据访问窗口。此传输的字节数设置为 4。

图 10-40. 来自主机的 CMD53 读取命令



DIR：来自主机，CMD：0x35，ARG：0x4021E04，CRC：0x73

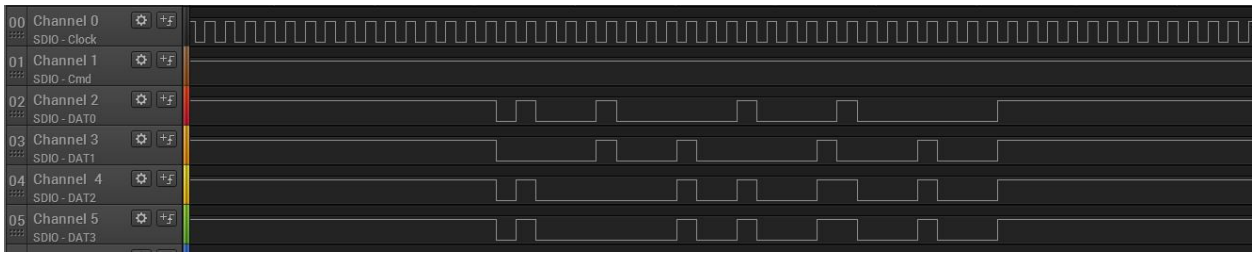
ATWILC 的响应如下。

图 10-41. 来自从机的 CMD53 读取响应

DIR：来自从机，CMD：0x35，ARG：0x1000，CRC：0x2D

将使用 DAT[0]:DAT[3]线从 ATWILC 返回芯片 ID。例如，ATWILC3000 芯片 ID 为 003000D0，返回方式如下所示。每条 DAT 线都会发送启动位、结束位和 CRC 位，然后分别计算和校验 CRC 位。

图 10-42. 来自从机的 CHIPID 响应



`sdio_init()`函数成功执行，来自 ATWILC 的响应成功发送到主机。ATWILC 初始化成功完成。此过程将在内核日志中显示为如下内容。

图 10-43. ifconfig wlan0 up 命令日志

```
# ifconfig wlan0 up
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_mac_open]MAC OPEN[dedb4000] wlan0
WILC POWER UP
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_init_host_int]Host[dedb4000][df55c340]
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_mac_open]*** re-init ***
wilc_sdio mmc0:0001:1 wlan0: INFO [wlan_init_locks]Initializing Locks ...
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_init]Initializing WILC_Wlan
wilc_sdio mmc0:0001:1: SDIO speed: 50000000
wilc_sdio mmc0:0001:1: chipid 001003a0
```

在函数 `init_irq()`中请求和初始化 IRQ GPIO。此后，使用 `wlan_initialize_threads()`函数初始化内核线程以进行传输和调试。其日志如下。

图 10-44. ifconfig wlan0 up 命令日志

```
wilc_sdio mmc0:0001:1 wlan0: INFO [init_irq]IRQ request succeeded IRQ-NUM= 134 on GPIO: 91
wilc_sdio mmc0:0001:1 wlan0: INFO [wlan_initialize_threads]Initializing Threads ...
wilc_sdio mmc0:0001:1 wlan0: INFO [wlan_initialize_threads]Creating kthread for transmission
wilc_sdio mmc0:0001:1 wlan0: INFO [wlan_initialize_threads]Creating kthread for Debugging
```

下载 AWILC1000 或 ATWILC3000 器件的 Wi-Fi 固件。此过程使用 `wilc_wlan_get_firmware()`和 `wilc_firmware_download()`这两个函数执行。成功下载固件后，使用 `linux_wlan_start_firmware()`函数启动固件。如果启动固件时出错，此函数将返回负值。

成功启动固件后，可以从内核日志中查看固件版本。在 `linux_wlan_init_test_config()`中配置工作模式、BSS 类型和电源管理等固件参数。

图 10-45. ifconfig wlan0 up 命令日志

```
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_get_firmware]Detect chip WILC1000
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_get_firmware]loading firmware mchp/wilc1000_wifi_firmware.bin
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_get_firmware]WLAN firmware: mchp/wilc1000_wifi_firmware.bin
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_firmware_download]Downloading Firmware ...
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_firmware_download]Downloading firmware size = 134964
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_firmware_download]Offset = 119660
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_firmware_download]Offset = 134964
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_firmware_download]Download Succeeded
wilc_sdio mmc0:0001:1 wlan0: INFO [linux_wlan_start_firmware]Starting Firmware ...
wilc_sdio mmc0:0001:1 wlan0: INFO [linux_wlan_start_firmware]Waiting for FW to get ready ...
wilc_sdio mmc0:0001:1 wlan0: INFO [linux_wlan_start_firmware]Firmware successfully started
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_wlan_initialize]WILC Firmware Ver = WILC_WIFI_FW_REL_15_01_RC3 Build: 9792
wilc_sdio mmc0:0001:1 wlan0: INFO [linux_wlan_init_test_config]Start configuring Firmware
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_mac_open]Mac address: fa:f0:05:f1:48:63
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Setting mcast List with count = 2.
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[0]: 33:33:0:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[1]: 33:33:0:0:0:2
wilc_sdio mmc0:0001:1 wlan0: INFO [set_power_mgmt] Power save Enabled= 1 , TimeOut = -1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Setting mcast List with count = 3.
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[0]: 33:33:0:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[1]: 33:33:0:0:0:2
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[2]: 1:0:5e:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Setting mcast List with count = 4.
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[0]: 33:33:0:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[1]: 33:33:0:0:0:2
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[2]: 1:0:5e:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[3]: 33:33:ff:f1:48:63
# wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Setting mcast List with count = 5.
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[0]: 33:33:0:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[1]: 33:33:0:0:0:2
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[2]: 1:0:5e:0:0:1
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[3]: 33:33:ff:f1:48:63
wilc_sdio mmc0:0001:1 wlan0: INFO [wilc_set_multicast_list]Entry[4]: 33:33:ff:0:0:0
wilc_sdio mmc0:0001:1 wlan0: INFO [debug_thread]*** Debug Thread Running ***
```

11. 附录 D——内核引导问题故障排除

如果.dtb 和 zImage 的文件大小大于默认值，则内核不会引导。内核日志如下所示。

```
U-Boot 2017.03-linux4sam_5.6 (Jan 14 2019 - 15:10:15 -0700)

CPU: SAMA5D44
Crystal frequency: 12 MHz
CPU clock : 600 MHz
Master clock : 200 MHz
DRAM: 512 MiB
NAND: 512 MiB
MMC: Atmel mci: 0
In: serial
Out: serial
Err: serial
Net: eth0: ethernet@f8020000
Hit any key to stop autoboot: 0

NAND read: device 0 offset 0x180000, size 0x817c
33148 bytes read: OK

NAND read: device 0 offset 0x200000, size 0x3df7c8
4061128 bytes read: OK
## Flattened Device Tree blob at 21000000
Booting using the fdt blob at 0x21000000
Loading Device Tree to 3f951000, end 3f95c77d ...OK

Starting kernel ...
```

可通过编辑环境命令修复上述错误，过程如下：

1. 当显示 `Hit any key to stop autoboot:`命令时，按任意键停止自动引导。
   如果成功，将出现“=>”符号。
2. 输入 `pri` 以打印环境变量。

   ```
   baudrate=115200
   bootargs=console=ttyS0,115200 mtdparts=atmel_nand:256k(bootstrap)ro,512k(uboot)ro,
   256k(env),256k(env_redundant),256k(spare),512k(dtb),6M(kernew
   bootcmd=nand read 0x21000000 0x00180000 0x0000817C; nand read 0x22000000 0x00200000
   0x003DF7C8; bootz 0x22000000 - 0x21000000
   bootdelay=1
   ethaddr=fc:c2:3d:0c:8e:e1
   fdtcontroladdr=3f95dc50
   stderr=serial
   stdin=serial
   stdout=serial
   Environment size: 468/131067 bytes
   ```

3. 如果 dtb 和 zimage 的文件大于默认值，请使用以下命令调整。

   ```
   editenv bootcmd
   ```

   - 将 `bootcmd` 变量（dtb 文件大小）从 `0x0000817C` 改为 `0x0000a17C`。
   - 将 `0x003DF7C8` 改为 `0x004DF7C8`（zImage 文件大小）。
   - 输入 `saveenv` 命令以保存命令。
   - 输入 `pri` 以打印保存的环境变量。
4. 输入 `boot` 以通过保存的值重启引导过程。

12. 文档版本历史

版本	日期	章节	说明
70005329D	2019 年 8 月	• 3.1 使用 Build Root（编译根）选项启用 BlueZ 5.x 软件包 • 8. 附录 A——装入 ATWILC 模块 • 9. 附录 B——ATWILC SDIO 通信 • 10. 附录 C——ATWILC SDIO 协议示例 • 11. 附录 D——内核引导问题故障排除	增加了这些章节
		• 4.1 ATWILC 电源控制 • 4.4 UART DMA	更新了这些章节
70005329C	12/2018	• 内核修改 • 内核配置 • Buildroot 修改 • ATWILC 电源控制 • ATWILC 上电 • 重新扫描 SDIO 卡 • SPI • 通用 IO	更新了这些章节
70005329B	02/2018	• 2. 内核修改一章。 • 4.2 SDIO 一节。 • 4.1.2 ATWILC 上电、4.2.1 重新扫描 SDIO 卡、4.2.2 SDIO 卡检测、4.3 SPI、4.4 UART DMA 和 5.6 设置 BT TX 功率小节。	更新了这些章节
		2.1 驱动程序源代码集成、2.2 固件集成和 4.5 通用 IO 章节。	增加了这些章节
		• 第 3 章“Makefile 移植”和第 4 章天线切换 • 第 5.5 节“中断请求（IRQ）”。	删除了这些章节
70005329A	2017 年 8 月	文档	初始版本

Microchip 网站

Microchip 网站（http://www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 http://www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 http://www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担

保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。除非另外声明，否则在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

商标

Microchip 的名称和徽标组合、Microchip 徽标、Adaptec、AnyRate、AVR、AVR 徽标、AVR Freaks、BesTime、BitCloud、chipKIT、chipKIT 徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi 徽标、MOST、MOST 徽标、MPLAB、OptoLyzer、PackeTime、PIC、picoPower、PICSTART、PIC32 徽标、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST 徽标、SuperFlash、Symmetricom、SyncServer、Tachyon、TempTrackr、TimeSource、tinyAVR、UNI/O、Vectron 及 XMEGA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

APT、ClockWorks、The Embedded Control Solutions Company、EtherSynch、FlashTec、Hyper Speed Control、HyperLight Load、IntelliMOS、Libero、motorBench、mTouch、Powermite 3、Precision Edge、ProASIC、ProASIC Plus、ProASIC Plus 徽标、Quiet-Wire、SmartFusion、SyncWorld、Temux、TimeCesium、TimeHub、TimePictra、TimeProvider、Vite、WinPath 和 ZL 均为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BlueSky、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet 徽标、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA 和 ZENA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Incorporated 在美国的服务标记。

Adaptec 徽标、Frequency on Demand、Silicon Storage Technology 和 Symmcom 均为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 为 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN：978-1-5224-5798-5

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 http://www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： http://www.microchip.com/support 网址： http://www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820