ATWILC1000/ATWILC3000

ATWILC Devices Linux® Porting Guide

Introduction

This user guide describes how to port the ATWILC1000 and ATWILC3000 Linux drivers to another platform and contains all the required modifications for driver porting.

The relevant source code for ATWILC device drivers and firmware revisions are maintained on GitHub. It is recommended that the user visit the following sites to get the latest code:

- ATWILC1000/3000 Driver: https://github.com/linux4wilc/driver
- ATWILC1000/3000 Firmware: https://github.com/linux4wilc/firmware

Prerequisites

- Hardware prerequisites:
  - ATWILC1000
  - ATWILC3000
- Software prerequisites:
  - ATWILC1000 Linux driver source code
  - ATWILC1000 firmware binary
  - ATWILC3000 Linux driver source code
  - ATWILC3000 firmware binary

Table of Contents

1. Driver Architecture

The driver architecture is illustrated in the following diagrams.

Figure 1-1. ATWILC1000 Linux Driver Architecture

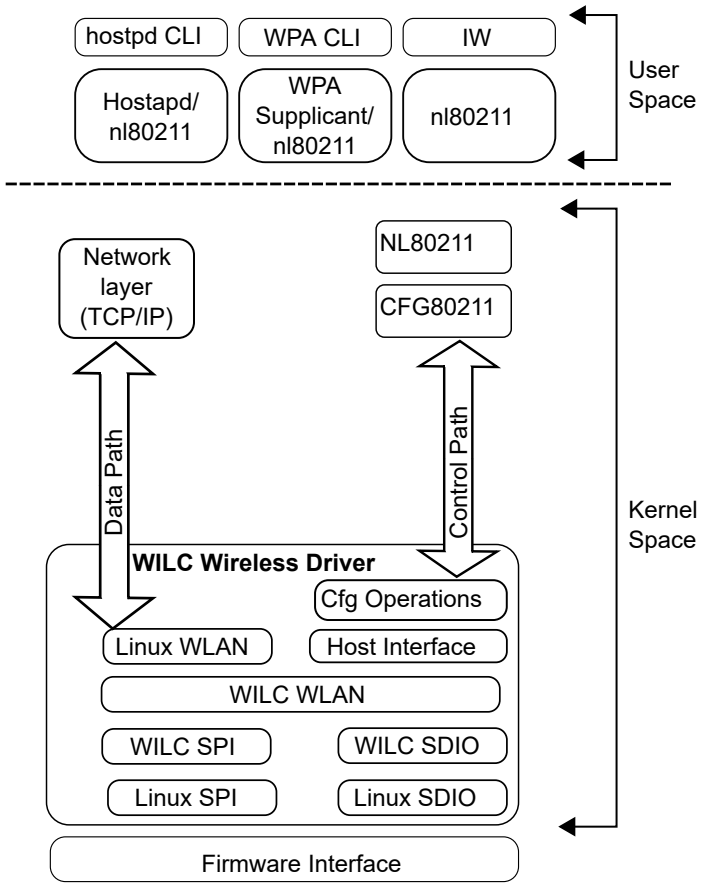
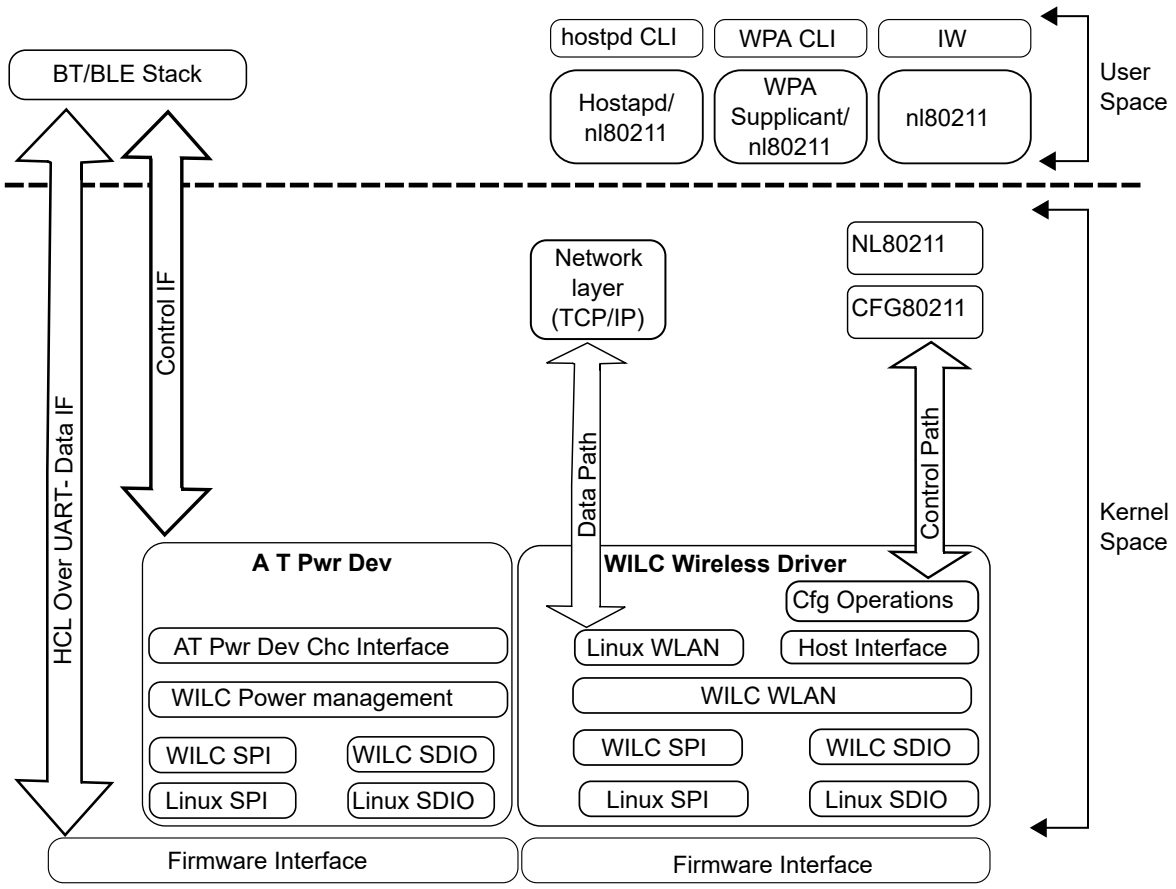


Figure 1-2. ATWILC3000 Linux Driver Architecture



1.1 Driver Modules

The section describes the driver modules.

1.1.1 Linux WLAN

The Linux WLAN implements Linux-specific operations:

- Controls the ATWILC device power
- Initializes/de-initializes the module during insert/remove
- Registers Interrupt Service Register (ISR)
- Turns on and turns off the chip by controlling the `CHIP_EN` and `RESET_N` GPIOs
- Registers the net_device interface
- Downloads and starts the firmware
- Implements the following net_device operations:
  - mac_init_fn
  - mac_open
  - mac_close
  - mac_xmit
  - mac_ioctl
  - mac_stats
  - wilc_set_multicaset_list

- Pushes Tx packets into the Tx queue
- Delivers Rx packets to Linux's stack

1.1.2 ATWILC WLAN

The ATWILC WLAN implements the ATWILC device ASIC-specific operations:

- Initializes the chip
- Downloads and starts the firmware
- Sends configuration packets
- Processes Tx Queue, sending packets to the firmware
- Receives packets from the firmware

1.1.3 Configuration (cfg) Operations

- Implements a wiphy interface
- Registers with `cfg80211`
- Implements `cfg80211_ops` operations
- Receives cfg operation requests from upper layers
- Passes cfg operation requests to the host interface using messages that need to be handled asynchronously
- Passes cfg operation responses/callback to upper layers

1.1.4 Host Interface

- Handles messages that carry cfg requests from cfg operations
- Builds configuration packets
- Sends configuration packets to the ATWILC device, and WLAN to be sent to firmware
- Receives cfg packet responses

1.1.5 ATWILC SDIO/SPI

Implements ATWILC-specific operations related to the communication bus protocol.

1.1.6 Linux SDIO/SPI:

Implements basic Linux-specific operations related to the communication bus.

1.1.7 ATWILC Power Management

- Thin layer driver providing services for Bluetooth® Low Energy (BLE) stack and Wi-Fi® driver
- Implements a character device interface to send commands to the BLE
- Exports APIs for the Wi-Fi driver
- Decides when to turn on and turn off the ATWILC device based on requests from Bluetooth and Wi-Fi
- Downloads BLE firmware to the ATWILC device's Intelligent Random Access Memory (IRAM) using Serial Peripheral Interface (SPI) or Secure Digital Input Output (SDIO)

1.1.8 AT Pwr Dev Character Interface

The Bluetooth/BLE stack uses a character interface that runs in the user space to convey commands to the power management layer.

- Use the `echo BT_POWER_UP > /dev/wilc_bt` command to power-up the chip. This command does not take effect if Wi-Fi is ON
- Use the `echo BT_POWER_DOWN > /dev/wilc_bt` command to power-down the chip. This command does not take effect if Wi-Fi is ON
- Use the `echo BT_DOWNLOAD_FW > /dev/wilc_bt` command to download Bluetooth FW to IRAM using SDIO or SPI
- Use the `echo BT_FW_CHIP_WAKEUP > /dev/wilc_bt` command to wake the chip up, issued before downloading FW for both UART or SDIO/SPI

- Use the `echo BT_FW_CHIP_ALLOW_SLEEP > /dev/wilc_bt` command to allow the chip to enter Sleep mode when appropriate

2. Kernel Modifications

2.1 Driver Source Code Integration

To integrate ATWILC device driver's source code into the kernel's build system, perform the following steps:

1. Add the driver source code including the `Kconfig` file under `linux_root/drivers/net/wireless/mchpw`. Edit `linux_root/drivers/net/wireless/Kconfig` to add the path to the ATWILC's driver Kconfig source `drivers/net/wireless/mchp/Kconfig`.
2. Edit `linux_root/drivers/net/wireless/Makefile` to add the ATWILC's driver source code `obj-$(CONFIG_WLAN_VENDOR_MCHP) += mchp/`.

2.2 Firmware Integration

To build ATWILC's firmware (FW) files as part of the kernel, perform the following steps:

1. Add firmwares to `linux_root/firmware/mchp/`
2. Edit `linux_root/firmware/Makefile` to add the firmware files
   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 Kernel Configuration

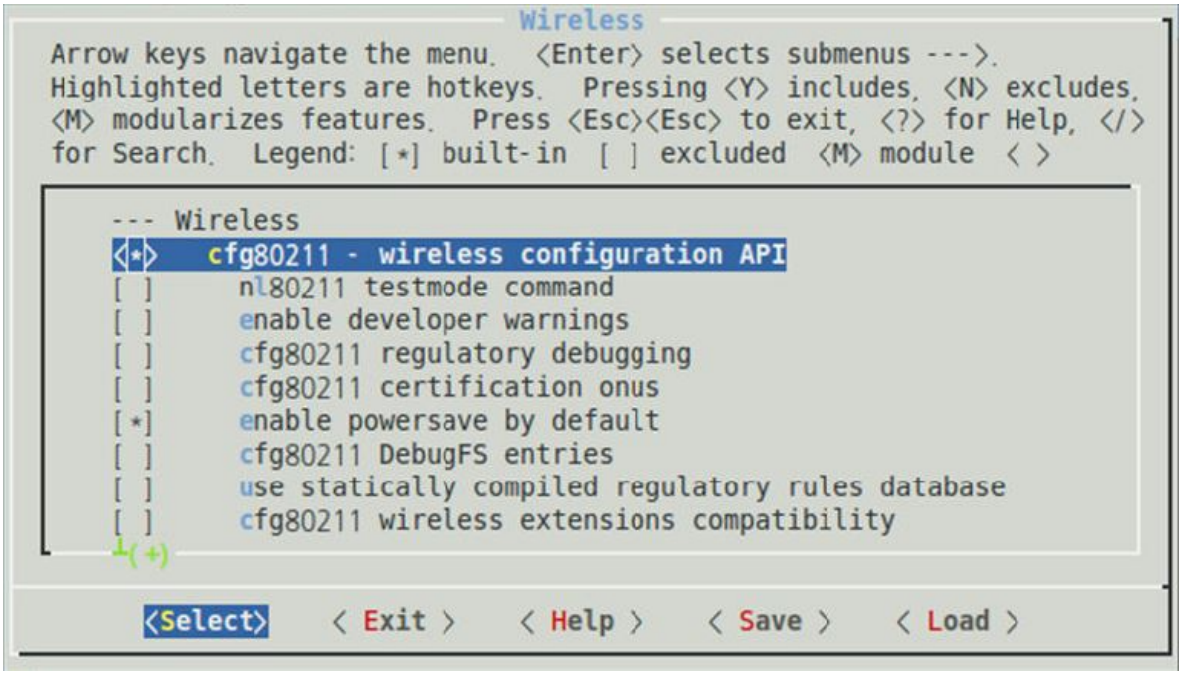
The user must perform the following modifications in the Linux kernel configuration to enable Wi-Fi support. These modifications are performed either by using the `menuconfig`, in case the kernel is being built separately, or by adding the corresponding CONFIG entries in the default `defconfig` file that buildroot is configured to use.

1. Go to the directory of the Linux kernel and enter the following command:

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

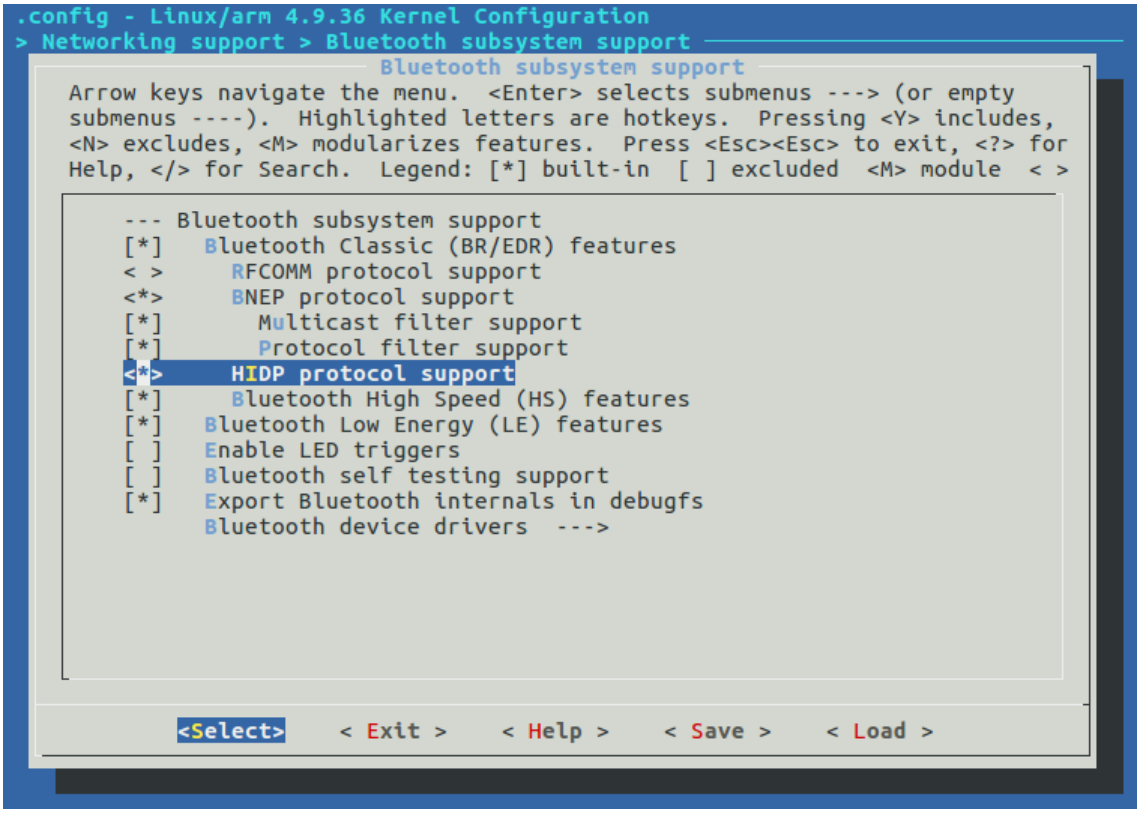
   Select the required configuration from the blue terminal to save and exit.
2. Enable the Linux 802.11 configuration API space. Select cfg80211 - wireless configuration API from `networking support > wireless menu`.

Figure 2-1. Wireless Terminal Screen



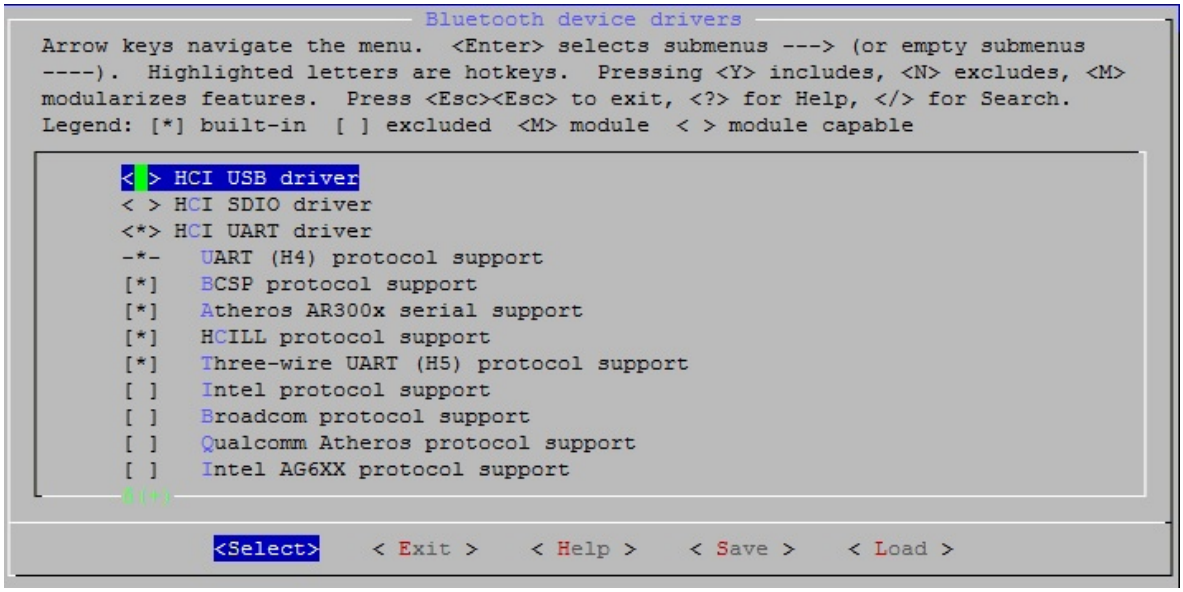
3. Enable BLE by going to `Networking Support > Bluetooth subsystem support` (see the following figure).

Figure 2-2. Bluetooth Subsystem Support



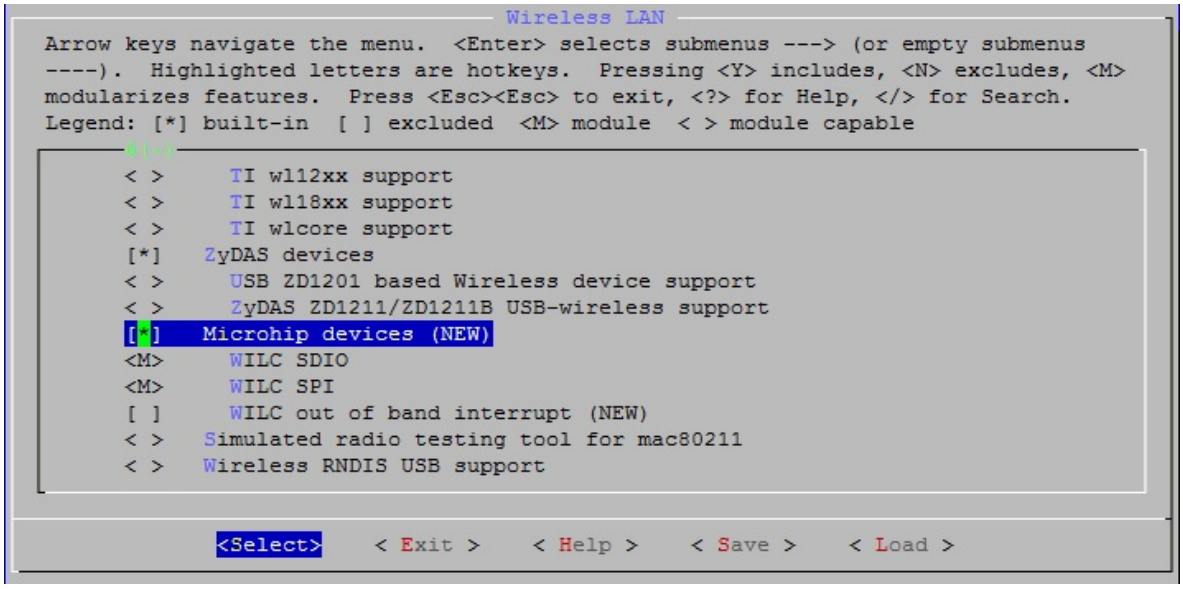
4. Enable HCI by going to Bluetooth device drivers (refer to the following figure).

Figure 2-3. Bluetooth Device Driver



5. Configure ATWILC Driver from `Device Drivers > Network Device Support`. Select the required configuration as mentioned in the following figure.
Note: If the driver code is added under `linux_root/drivers/staging`, the `menuconfig` entries will be available under `Device Drivers > Staging drivers`.

Figure 2-4. Wireless LAN

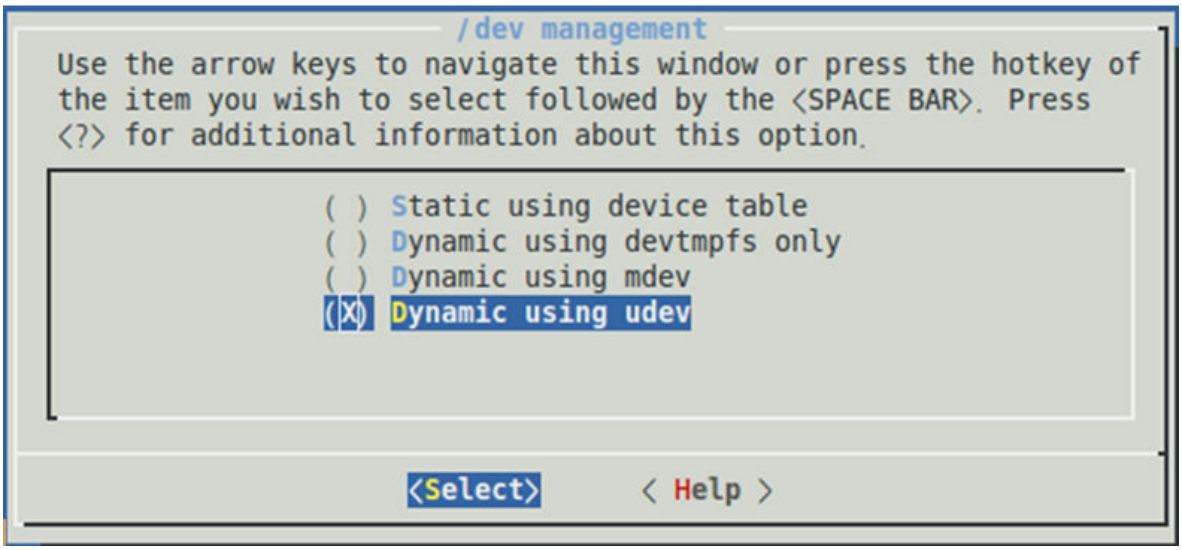


3. Buildroot Modifications

The user must perform the following modifications to enable packages of the buildroot.

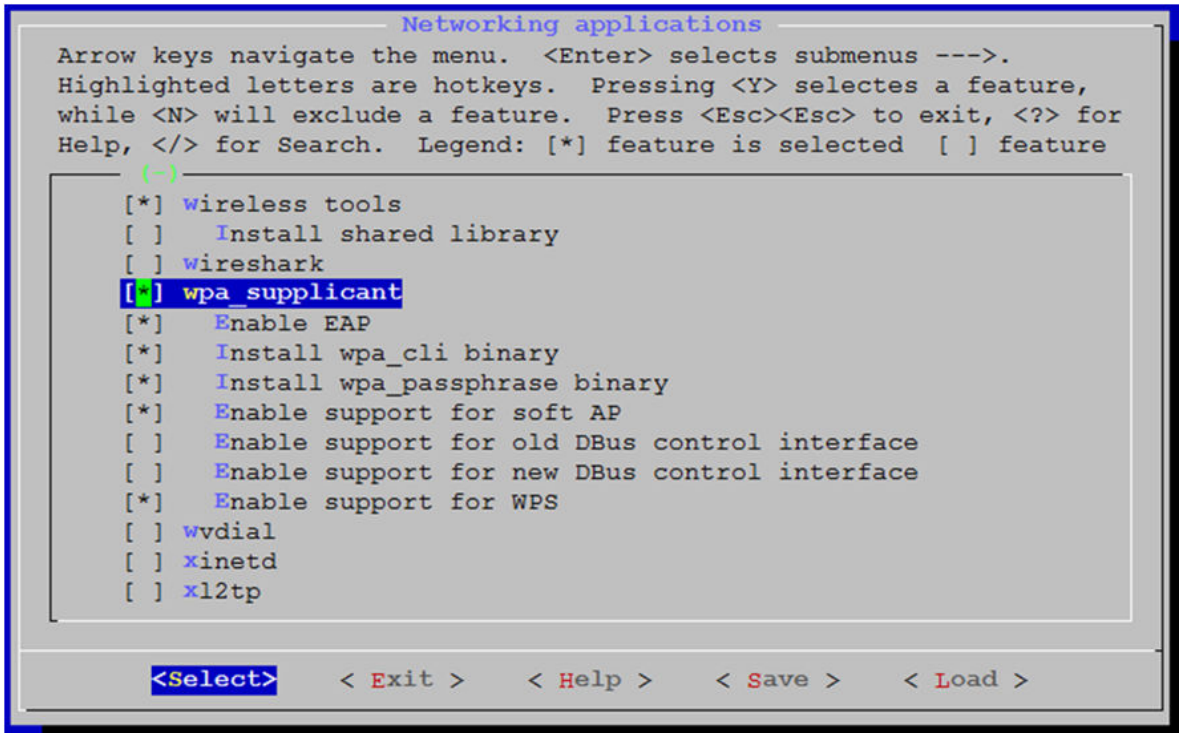
1. Go to the directory of the buildroot and enter the `$ make menuconfig` command.
2. Enable dynamic/dev management using udev. Move to `System configuration` > `dev management` and select `Dynamic using udev`.

Figure 3-1. /dev management Screen



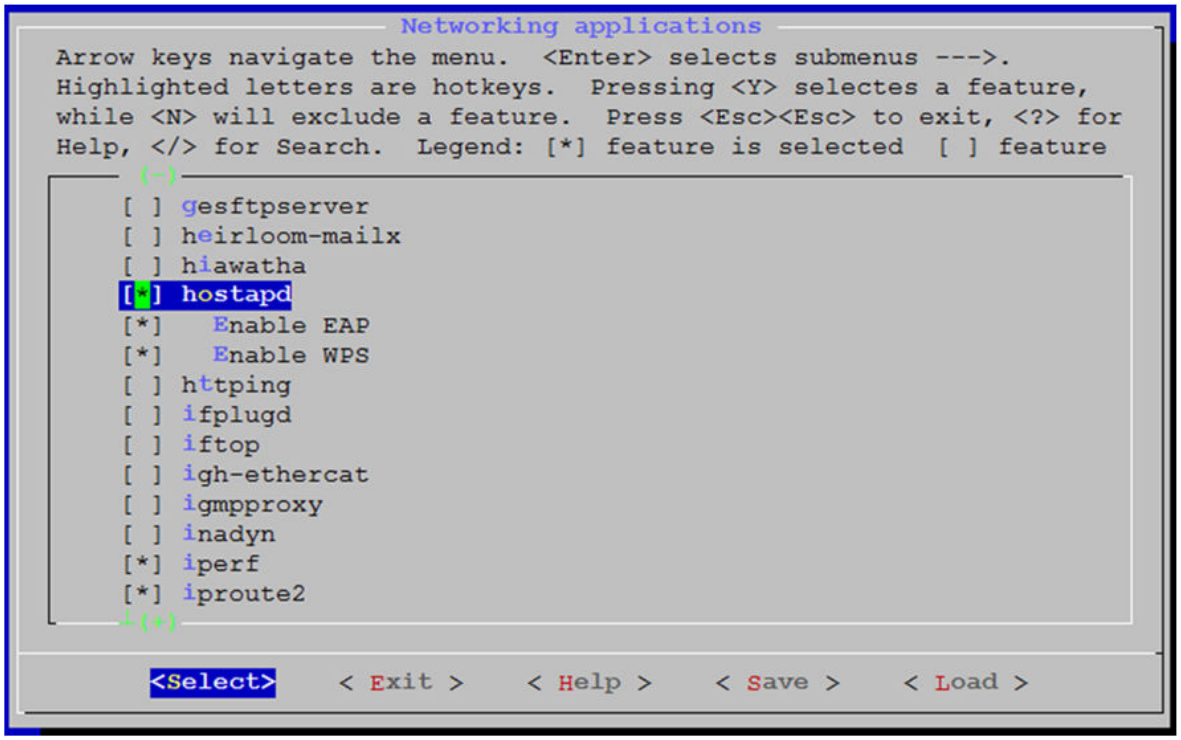
3. Enable wpa_supplicant from `Target package->Networking applications` and select `wpa_supplicant`.

Figure 3-2. Networking applications - wpa_supplicant Screen



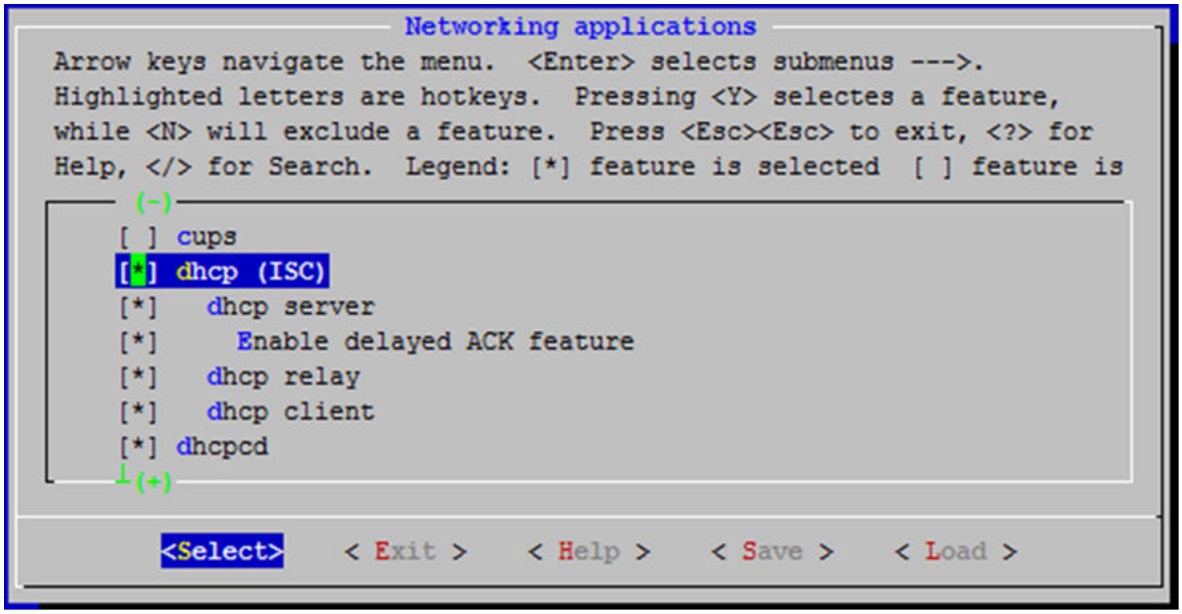
4. Move to `Target package` > `Networking applications` and select `hostapd`.

Figure 3-3. menuconfig: Enable hostapd



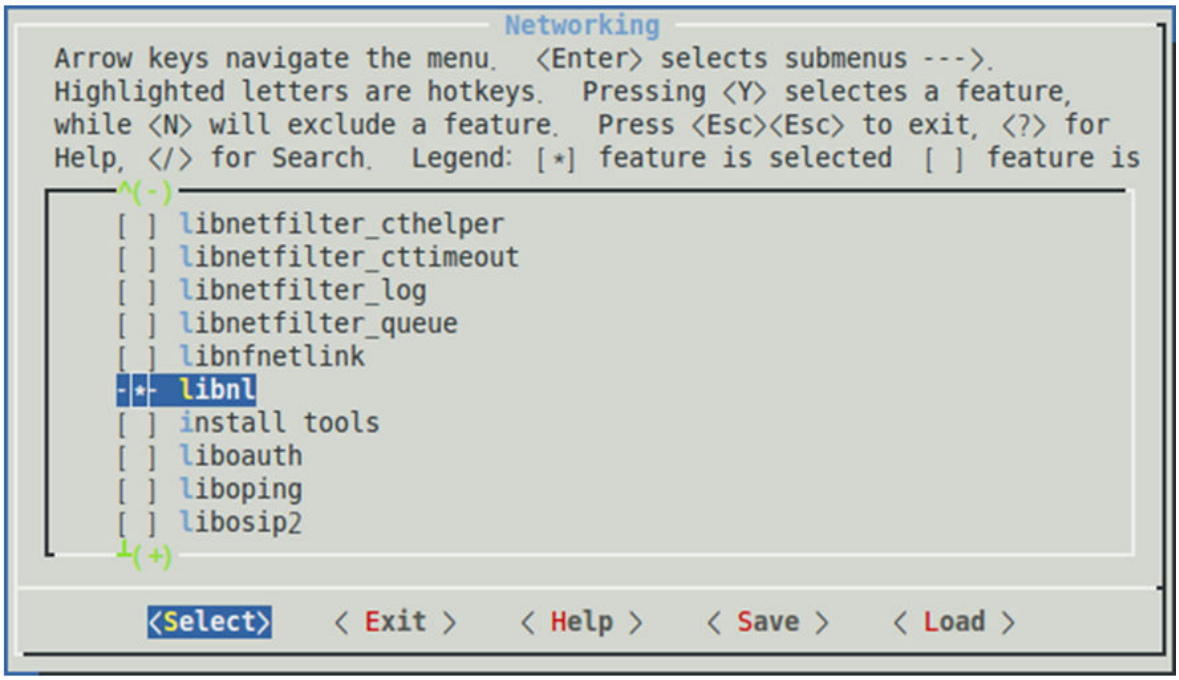
5. Allocate IP addresses between the server and client. Move to `Target package` > `Networking applications` and select `dhcp`.

Figure 3-4. menuconfig: Enable dhcp



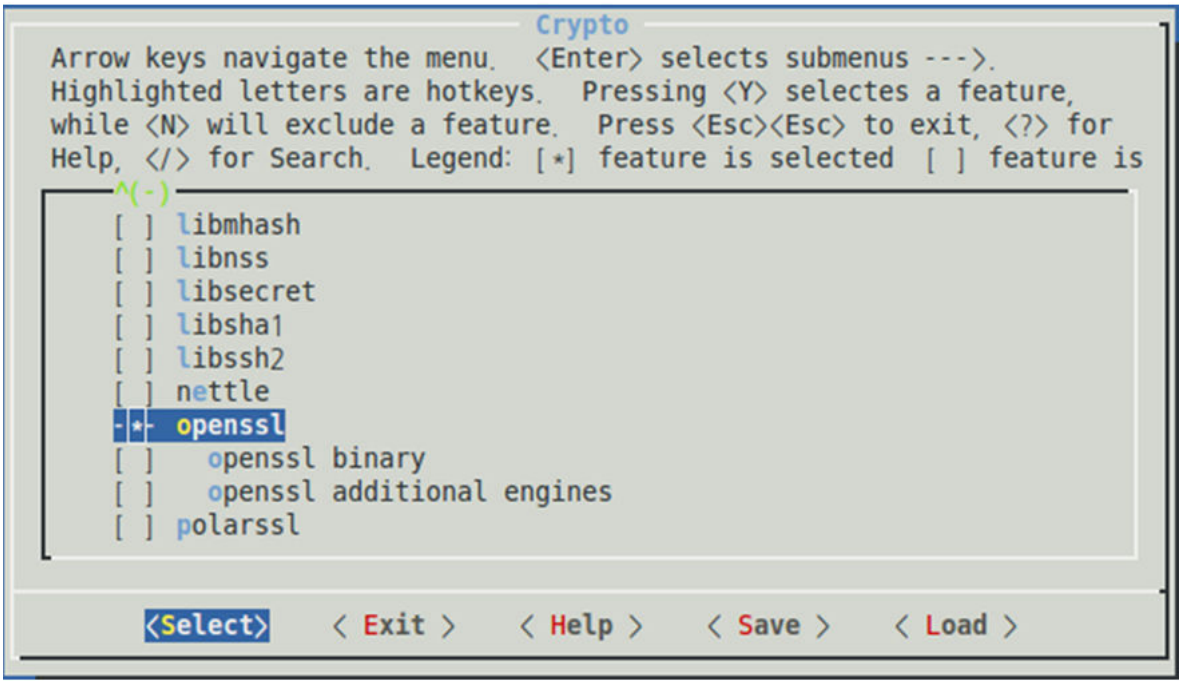
6. Move to `Target package` > `Libraries` > `Networking` and select `libnl`.

Figure 3-5. menuconfig: Enable libnl



7. Connect to a secured AP. Move to `Target packages` > `Libraries` and select `Crypto`.

Figure 3-6. menuconfig: Enable openssl



3.1 Enabling BlueZ 5.x Package using Build Root Options

Perform the following steps to enable Bluetooth related packages and tools for SAMA5D4 and ATWILC3000.

1. Clone the Buildroot-at91. For more details on cloning and building the buildroot-at9, refer https://www.at91.com/linux4sam/bin/view/Linux4SAM/BuildRootBuild#How_to_build_Buildroot_for_AT91.

2. Configure the SAMA5D4 Xplained defconfig.
   Go to the directory of the buildroot and enter the following command:

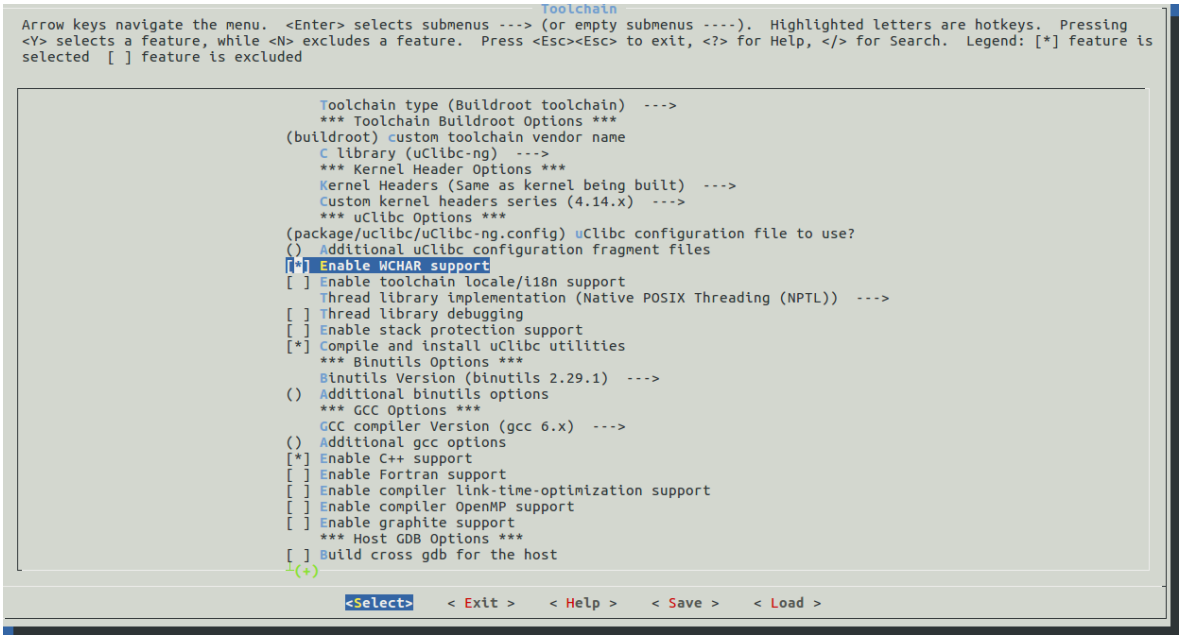
```
$make atmel_sama5d4_xplained_defconfig
```

3. Configure the menuconfig.
   Go to the directory of the buildroot and enter the following command:

```
$ make menuconfig
```

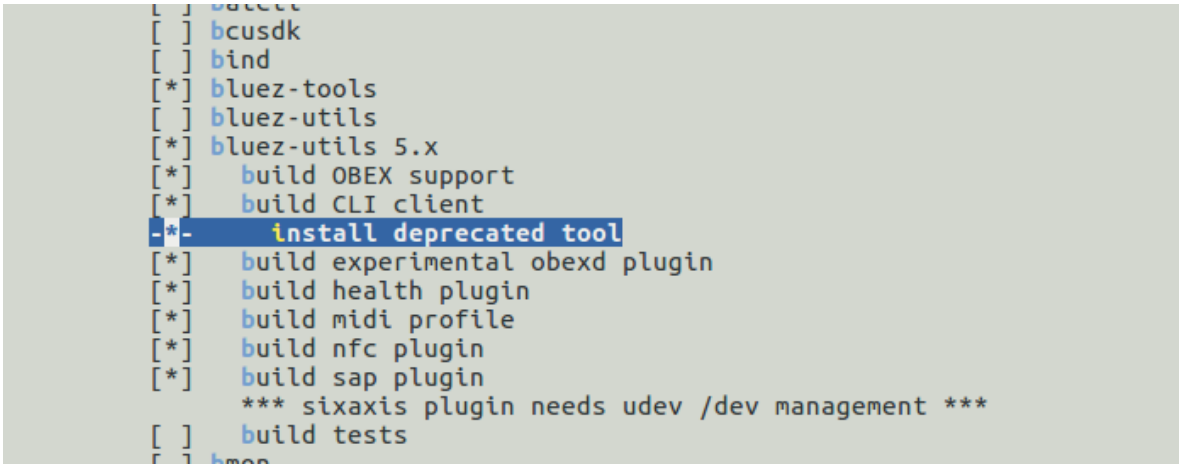
Note: The WCHAR support (**Toolchain** > **WCHAR**) must be enabled, as it has dependency on BlueZ stack.

Figure 3-7. Enable WCHAR Support



4. To enable Bluez5 stack, navigate to **Target packages** > **Network Applications** and select the tools as shown in the following figure.

Figure 3-8. Enable Bluez



Note: Make sure that **install deprecated tool** is selected; the tool includes the hciattach tool. This is used to enable Bluetooth over UART.

5. Save the config file and run `make` command.
   After successful completion, the `rootfs.ubi` file is generated in the `/buildroot-at91/output` folder.

4. Porting Driver

To port the Linux driver to another platform, the APIs that are dependent on the hardware are changed to another implementation that suits the new platform.

4.1 ATWILC Power Control

The host controls the ATWILC device's power using two pins, CHIP_EN and RESET_N. If these pins are connected to the host's GPIO pins, the device tree file should be modified as explained in 4.5 General Purpose IOs. Therefore, the driver can retrieve information of the GPIOs connected to ATWILC's CHIP_EN and RESET_N.

Other hardware designs connect the CHIP_EN and RESET_N to the power line using an RC network, which ensures that the ATWILC device is powered on with the host. In this case, these APIs are not required and should be modified to avoid changing the GPIO configurations that might be already used for another function.

Note: When ATWILC's module is loaded to Linux kernel, it communicates with the ATWILC's SDIO/SPI controller. Upon success, it calls the corresponding probing function, which initializes the rest of the driver. The Linux kernel must find the SDIO/SPI controller prior to ATWILC's driver initialization. Therefore, make sure that ATWILC is powered-up prior to initialization.

4.1.1 ATWILC Power Off

To turn off the chipset, set CHIP_EN and RESET_N low.

4.1.2 ATWILC Power On

To turn on the chip, set the CHIP_EN line high, and then set the RESET_N high with a 5 ms delay.

A reference power control API implementation is provided below.

```
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);
}
```

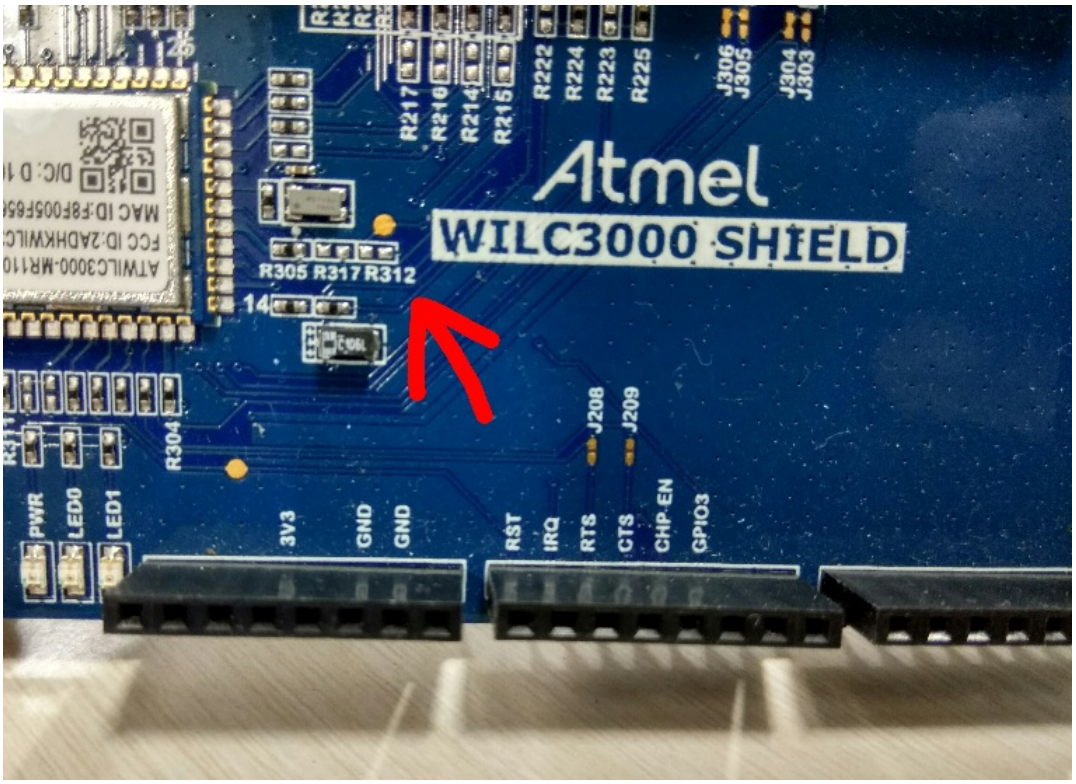
4.2 SDIO

4.2.1 Rescan SDIO Card

When the ATWILC device is connected to the host using the SDIO bus, the host's SDIO controller initializes and then interrogates the ATWILC device's SDIO controller. The interrogation operation happens when the host is booting. If any SDIO device is powered on after the host is booted, a rescan operation must be triggered to detect it.

The ATWILC device should be powered on by default at startup. For ATWILC3000 Shield Board, this can be done by mounting resistor R312 with an approximate value of 120k in the location shown below on the ATWILC3000 Shield Board.

Figure 4-1. SDIO Resistor on Shield Board



Tip: If the ATWILC module powers up automatically during system boot, the rescan SDIO is not required since the ATWILC SDIO initialization and interrogation is performed during system boot.

4.2.2 SDIO Card Detect

MMC slots have a card detect line that is used by Linux to indicate that an SDIO card was inserted, which can be used if ATWILC is connected through the MMC slot such as when ATWILC1000 SD board is used. However, not all setups would have this line connected. For example, ATWILC3000 Shield Board + SAMA5D4 Xplained setup uses the Arduino header for connection, which doesn't expose the card detect GPIO (PE3). In this case, it's essential to remove the dependency on the card detect line for the platform to detect ATWILC's SDIO controller.

Under the corresponding MMC entry in the dts file, remove the `cd-gpios` entry, and add a `non-removable` property. An example of such change for `at91-sama5d4_xplained.dts` is shown below:

```
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

The ATWILC device driver identifies the SPI port that is to be used for communication by passing the Linux SPI registration API, `spi_register_driver() a .of_match_table` that specifies an appropriate `.compatible` parameter.

The ATWILC device's spi_driver is defined below in the `wilc_spi.c` file.

```
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,
};
```

The user ensures that the `.compatible` parameter defined in `of_device_id wilc_of_match[]` is defined in the board's device tree file that describes the board.

For example, the SAMA5D4's device tree `at91-sama5d4_xplained.dts` file in the `kernel_tree/arch/arm/boot/dts` path is modified to match the driver, as shown below. In addition, the SPI clock is changed to improve communication with the host processor. Therefore, the spi-max-frequency property should be changed appropriately under the corresponding SPI device node.

```
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

For the ATWILC3000 only, the user must ensure that the Direct Memory Access (DMA) is enabled in the Universal Synchronous/Asynchronous Receiver/Transmitter (USART).

For example, the SAMA5D4's `sama5d4.dtsi` file in the `kernel_tree/arch/arm/boot/dts` path is modified as shown below.

```
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";
};
```

Note: SDIO/SPI DMA may be enabled for Wi-Fi depending on the application requirements.

4.5 General Purpose IOs

The ATWILC device driver uses 3 GPIOs connected to CHIP_EN, RESET_N, and IRQn. To port the driver to a new host, the corresponding GPIO numbers should be updated.

The driver reads the corresponding GPIO numbers from the device tree file. In case the GPIOs weren't found in the device tree file, the driver would use a static definition of the GPIO numbers defined in `wilc_wlan.h`

```
    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");
    }
```

The corresponding change in the device tree file should be done by adding the GPIOs' information either in the SPI node, or the SDIO node. The example below shows this change for at91-sama5d4_xplained.dts.

```
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. Vendor Specific HCI Commands

Some non-standard Host Controller Interface (HCI) commands are used by the ATWILC3000 Bluetooth core to provide extended services and options for the host.

5.1 Updating UART Parameters Command

When the host needs to change the UART settings of the Bluetooth controller, the command mentioned in the table below is used to change the baud rate and flow control options.

Table 5-1. UART Parameters Command Structure

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload	
1	OxFC53	5	Baud rate (4bytes)	Flow control (1byte)

5.2 Changing BD Address

The host uses the command mentioned in the following table to change the BD address. After performing this command, a standard Reset command (Op Code 0x0c03) is performed to update the new address.

Table 5-2. Command Structure to Change BD Address

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload
1	0xFC54	6	BD Address (6 bytes)

5.3 Write Memory

The following table describes the write memory command structure. This command is used when the Boot ROM is running on the host controller only to write a block of memory to the Bluetooth controller. Its main function is to download the firmware to the controller.

Table 5-3. Write Memory Command Structure

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload		Data Block
1	0xFC52	8	Address (4 bytes)	Size (4 bytes)	Data block to be written to the memory

5.4 Vendor-Specific Reset

The following table describes the vendor-specific Reset. This command is used when the Boot ROM is running on the host controller only. It is issued after new firmware is downloaded to the Bluetooth controller's memory to start the command.

Table 5-4. Change BD Address Command Structure

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload
1	0xFC55	0	N/A

5.5 Read Register

The following table describes the Read registers. This command is used to read registers from the Bluetooth controller using UART.

Table 5-5. Read Register Command Structure

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload		
1	0xFC01	6	Register Address (4 bytes)	0x20	0x01

5.6 Set BT TX Power

The following table describes how to set the BT Tx power command structure. This command is used to set the transmit power to a specific level.

Table 5-6. Set BT Tx Power Command Structure

HCI Packet Indicator	Op Code	Parameters Length	Parameters Payload	
1	0xFC3B	3	Reserved (2 bytes)	Tx Level (1 Byte)

The TX levels can be set to one of the following values, 0, 3, 6, 9, 12, 15, 18. Values other than those recommended will be floored to the nearest supported level. A maximum level might be enforced by the firmware to comply to RF regulations.

6. Bluetooth Firmware Download

This chapter describes the different ways to download Bluetooth firmware for the ATWILC3000 only.

6.1 Using SDIO Or SPI

As described in the 1.1.8 AT Pwr Dev Character Interface section, the power device module is used to download the Bluetooth firmware using the below command.

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

6.2 Using UART

The Bluetooth controller has a Boot ROM that starts execution as soon as the device is powered up. It is used to receive the firmware, download it to the instruction memory, and trigger the controller to start executing it. The Boot ROM runs initially at 115200 bps baud rate, and flow control is disabled. If the Bluetooth firmware is already running on the CPU, do not download firmware.

6.2.1 Firmware Download Detection

The host determines if the ATWILC Microprocessor Control Unit (MCU) is running firmware that is downloaded or running from the Boot ROM. The host determines if a new version of firmware is to be used. The detection is finished by reading the local version through the standard Read Local Version HCI command `Op Code 0x1001` and checking for byte number `'7'` in the HCI event packet sent back from the Bluetooth controller as follows:

- If byte 7 is 255, then the Boot ROM code is running on the CPU
- If byte 7 is 6, then the firmware code is running on the CPU

6.2.2 Firmware is Already Downloaded

If the firmware is already downloaded, the host can perform the following:

- Change the UART parameters like the baud rate and flow control options
- Change the BD address of the Bluetooth controller if the BD address is not stored on the controller
- Send a Reset command. This is mandatory if the BD address is changed

6.2.3 Start Firmware Download

The download process goes through the following steps:

1. Send standard Reset command `Op Code 0x0c03` to the controller.
2. Raise the controller baud rate to a higher baud rate to speed up the download process; this step is done by issuing the vendor-specific UART Parameters Command to the controller with the required baud rate. Disable the flow control at this point.
3. Raise the host baud rate to a matching baud rate.
4. Start downloading the firmware image to the controller IRAM starting at address `0x80000000` through the Write Memory vendor-specific HCI commands.
5. Send the vendor-specific reset HCI command to transfer the execution from the Boot ROM code to the downloaded firmware code.
6. Update the host UART’s baud rate and flow control settings to the initial settings that the firmware operates and communicates with the host UART's settings.
7. Raise the controller baud rate and flow control settings to the required run time operational settings.
8. Raising the host UART settings to match the controller settings.
9. Change the BD Address of the controller if the BD Address is not stored on the Bluetooth controller.
10. Send the standard HCI Reset command to the controller to operate with the new BD Address.

7. Suspend/Resume

This chapter describes the required changes for the suspend and resume mechanism.

7.1 Host Wake-up

When the host is in Suspend mode, the controller wakes up the host on certain events using the wake-up GPIO available on the host board. The wake-up GPIO can be identified using the following:

- If the host is using SDIO bus with in-band interrupt, WILC will use a separate interrupt line through the IRQ to wake-up the host.
- If the host is using SDIO bus with out of band interrupt, or SPI bus, IRQ line will be used by both the buses to indicate new event to the host, and to wake up the host if it's in Suspend mode. For more information on how to configure the host's GPIO number connected to ATWILC's IRQ pin, refer 4.5 General Purpose IOs.

The example below shows how to configure the GPIO on PC27 as the wake-up GPIO for the SAMA5D4 by modifying the `at91-sama5d4_xplained.dts` file (dts file) as follows:

Modify the `pinctrl@fc06a000` block, the underlined text, to define PC27 as a GPIO:

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

Where, `pinctrl@fc06a000` is a part of “apb” block which itself is a part of “ahb” block.

Then, modify the `gpio_keys` block, underlined text to define PC27 as an active-high wake-up 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. Appendix A - Loading ATWILC Module

After the kernel is booted, call the `modprobe/insmod` command for loading the wilc-spi or wilc-sdio module, if the ATWILC module is not built along with kernel. The sequence followed by both the SDIO and SPI interfaces are almost same and are explained as follows.

`modprobe/insmod` command add or remove the modules from the Linux kernel. `modprobe` command checks in the module directory `/lib/modules/` for all the modules, while the `insmod` command checks the base directory of the driver that is mentioned as parameter.

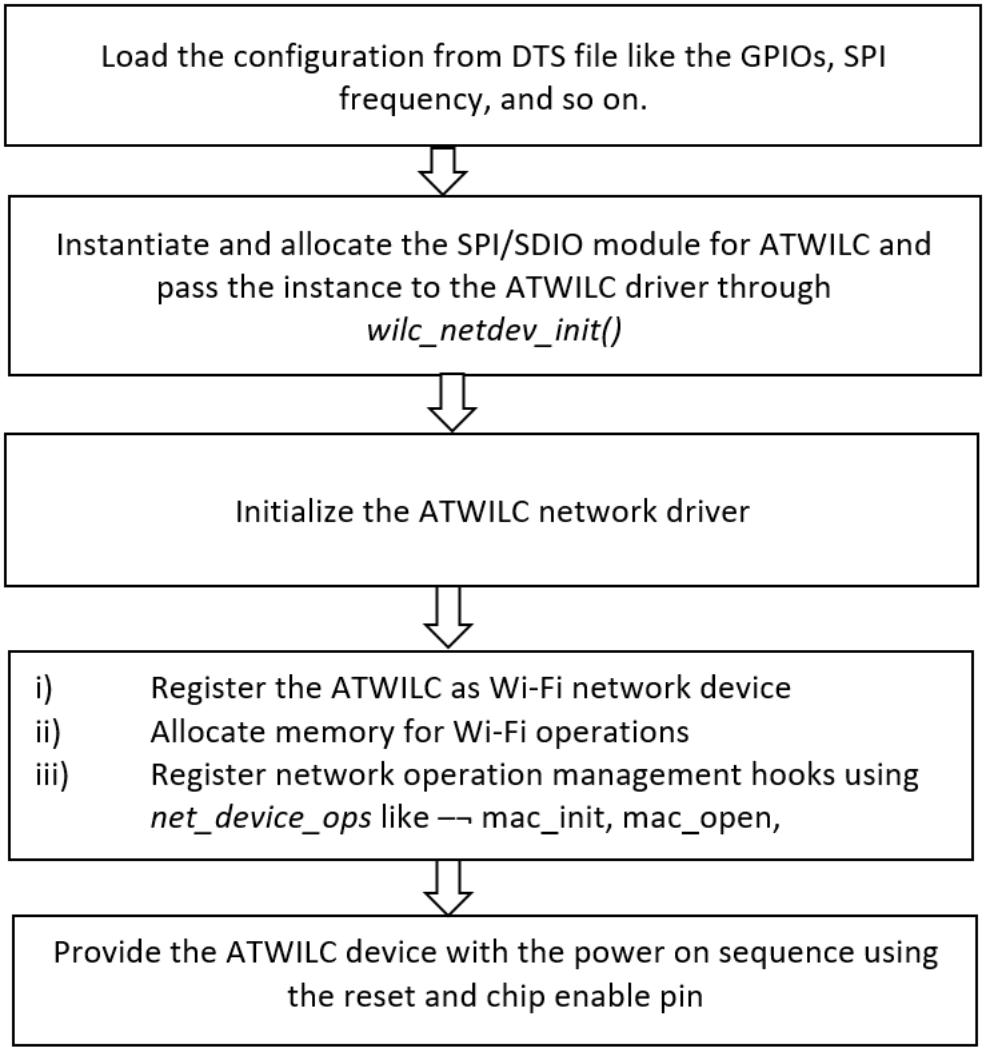
During compilation, the build process extracts this information from all the drivers. Upon loading the module, the device in the driver is compared with the devices declared in the device tree file.

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

On receiving the `modprobe` command, the driver is loaded and the execution starts from the `module_spi_driver()*` function for SPI or `module_driver()` function for SDIO. This function passes to the kernel, the structure containing the device_id, probe and remove function, and so on. Once a device ID match is found in the DTS file, the probe function is called. Either the `wilc_bus_probe()` or `linux_sdio_probe()` function is called respectively for SPI /SDIO interface. The probe function performs early initialization for modules and registers the device ATWILC with the kernel.

The sequence of actions performed by the probe function is shown in the following figure:

Figure 8-1. Sequence of Probe Function



This concludes the probe operation and the success log is shown in the following figure.

Figure 8-2. `modprobe/insmod` Log

```
# 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. Appendix B - ATWILC SDIO Communication

SDIO protocol provides communication over the SD bus and is based on command and data bit streams that are initiated by a Start bit and terminated by a Stop bit.

Command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.

Response is a token that is sent from an addressed card, or (synchronously) from all connected cards, to the hosts as an answer to a previously received command. A response is transferred serially on the CMD line.

Data can be transferred from the card to the host or vice versa. Data is transferred via the data lines.

Data transfer to/from the SD Card are done in blocks. Data blocks are always succeeded by CRC bits. Single and multiple block operations are defined.

10. Appendix C - ATWILC SDIO Protocol Example

After successfully loading the ATWILC module, enter `if config wlan0 up` command to bring up the Wi-Fi interface. It initializes ATWILC SDIO, reads the Chip ID, downloads Wi-Fi firmware and brings up the WLAN interface.

Upon calling `if config wlan0 up` command, `wilc_mac_open()` function is invoked first, in which `wilc_wlan_initialize()` is called to initialize WLAN interface `wilc_wlan_init` calls the corresponding `hif_init()` to bring up HIF layer. If SDIO interface is used `sdio_init()` is called and if SPI interface is used `wilc_spi_init()` is called. The sequence of SDIO packets while initializing WLAN interface in the ATWILC3000 is described in this section.

As per ATWILC's driver implementation, the first command to send in `sdio_init()` function is as follows.

Figure 10-1. Enable 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;
}
```

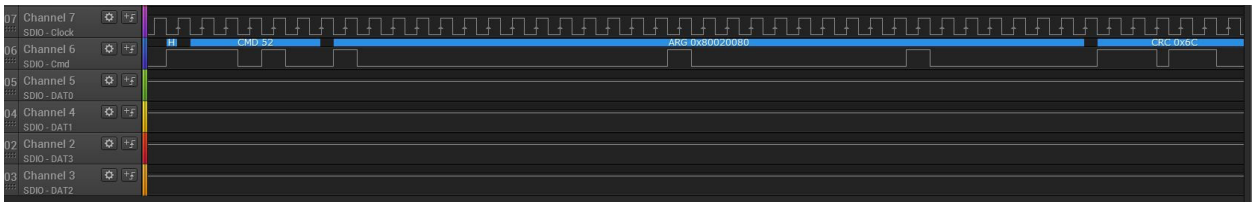
CSA Enable bit controls access to the Code Storage Area for this function. If this bit is cleared to 0, then any read or write access to the CSA shall be blocked. If this bit is set to 1, then access to the CSA is allowed. This bit is cleared to 0 upon reset. The code will set 7th bit of Function Basic Registers (FBR) of function 0 with address 0x100 to enable CSA.

Figure 10-2. Function Basic Register with Address 0x100

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			

The corresponding SDIO logs are as follows.

Figure 10-3. Enable CSA Command from Host



DIR: from Host, CMD:0x34, ARG:0x80020080 and CRC:0x6C.

As per CMD52 format,

- Command starts with Start bit as '0'
- Direction as '1' that is from host
- CMD52 identifier as 110100b

- Argument value of 0x80020080 which has R/W flag as 1
- Function number as 0
- RAW flag a ‘0’
- Register address as 0x10
- Write data as 0x80 (which sets CSA Enable bit of above said register)
- CRC as ‘0x6C’
- End bit as ‘1’.

ATWILC’s response is as follows.

Figure 10-4. Enable CSA Response from Slave

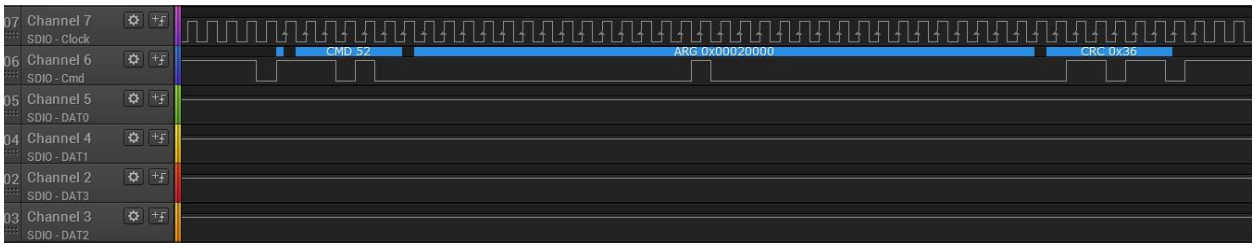
DIR: from Slave, CMD:0x34, ARG:0x1080 and CRC:0x5A

As per CMD52 response R5,

- Start bit as ‘0’
- Dir ‘0’ which is from card to host
- CMD52 identifier as 110100b
- Argument value of 0x00001080 which has response flag as 0x10
- Function number as 0
- RAW flag a ‘0’
- Register address as 0x10
- Write data as 0x80 (same data which is sent in previous packet)
- CRC as ‘0x5A’
- End bit as ‘1’

Since RAW flag is set, the same register is read immediately to make sure if the bit is set. Read command from the host is as follows.

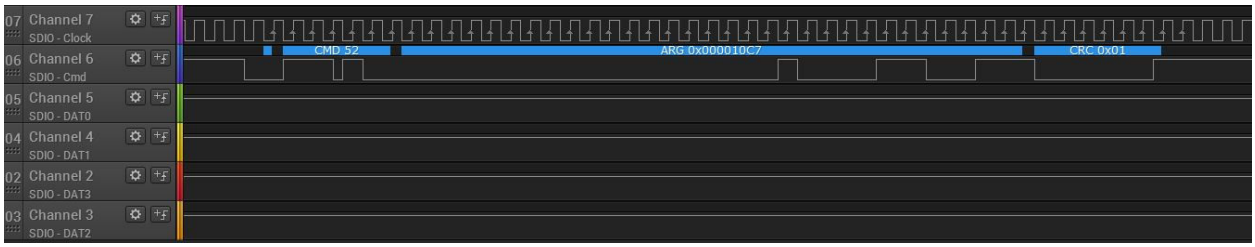
Figure 10-5. Read CSA Command from Host



DIR: from Host, CMD:0x34, ARG:0x20000 and CRC:0x36

ATWILC’s response is as follows.

Figure 10-6. Read CSA Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x10C7 and CRC:0x01. From the above log, C7 is returned as the data which has CSA Enable bit as set.

Set function 0 block size as 512 in address 0x10-0x11 which is in Card Common Control Registers (CCCR) of Common I/O Area (CIA).

Figure 10-7. CCCR Register with Address 0x10-0x11

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

Figure 10-8. Set Function 0 Block Size Command from Host

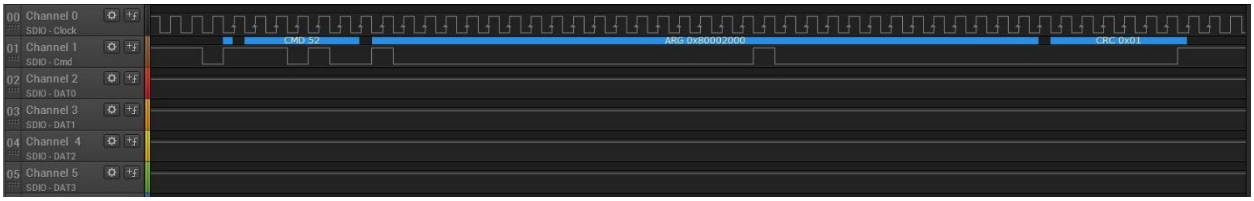
```
/**
 *      function 0 block size
 **/
if (!sdio_set_func0_block_size(wilc, WILC_SDIO_BLOCK_SIZE)) {
    dev_err(&func->dev, "Fail cmd 52, set func 0 block size...\n");
    goto fail;
}
g_sdio.block_size = WILC_SDIO_BLOCK_SIZE;
```

The above function is sent as two commands from the host.

```
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;
}
```

The SDIO logs for above commands are as follows.

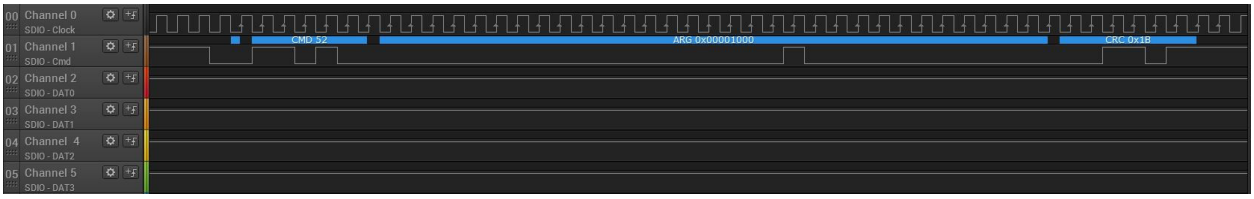
Figure 10-9. Set Function 0 Block Size Command from Host



DIR: from Host, CMD:0x34, ARG:0x80002000 and CRC:0x01.

The response for the above command from ATWILC is as follows. It is sent for address 0x10, with data as 0x00 (lower byte of 512, 0x0200).

Figure 10-10. Set Function 0 Block Size Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1000 and CRC:0x1B.

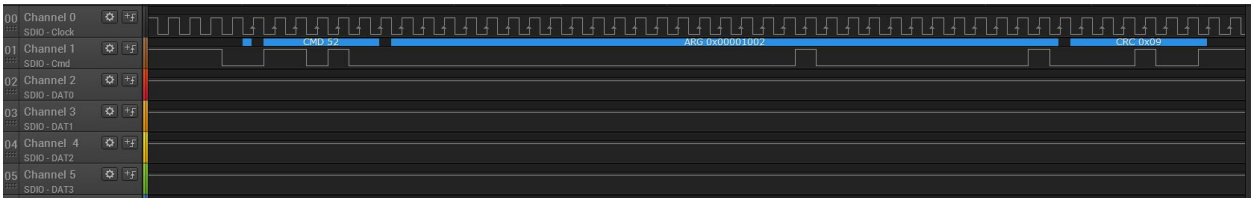
The next CMD52 is sent for address 0x11, with data as 0x02 (higher byte of 512, 0x0200).

Figure 10-11. Set Function 0 Block Size Command from Host

It has DIR from Host, CMD:0x34, ARG:0x80002202 and CRC:0x05 .

The response for the above command from ATWILC is as follows.

Figure 10-12. Set Function 0 Block Size Response from Slave



It has DIR: from Slave, CMD:0x34, ARG:0x1002 and CRC:0x09

After setting function 0 block size successfully, enable function 1 by setting IOE1 bit (I/O enable) in Card Common Control Registers, with address as 0x02 and data as 0x02.

Figure 10-13. CCCR Register with Address 0x02 and 0x03

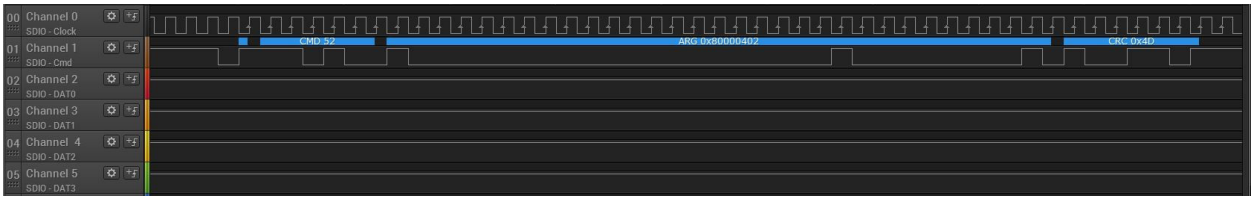
0x02	I/O Enable	IOE7	IOE6	IOE5	IOE4	IOE3	IOE2	IOE1	RFU
0x03	I/O Ready	IOR7	IOR6	IOR5	IOR4	IOR3	IOR2	IOR1	RFU

Figure 10-14. Enable Function1 I/O Register

```
/**
 *      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;
}
```

The SDIO logs are as follows.

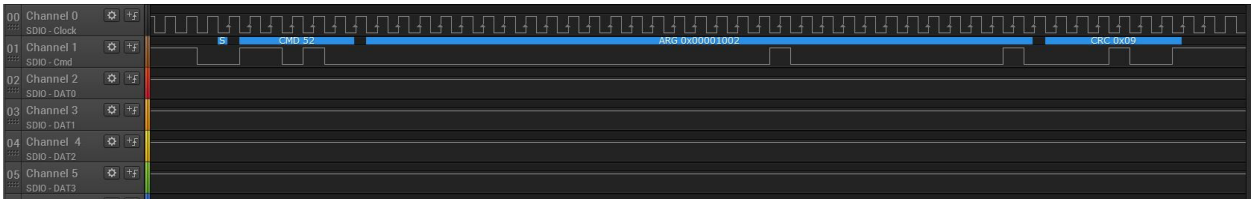
Figure 10-15. Enable Function1 I/O Register Command from Host



DIR: from Host, CMD:0x34, ARG:0x80000402 and CRC:0x4D

The response from ATWILC for above command is as follows.

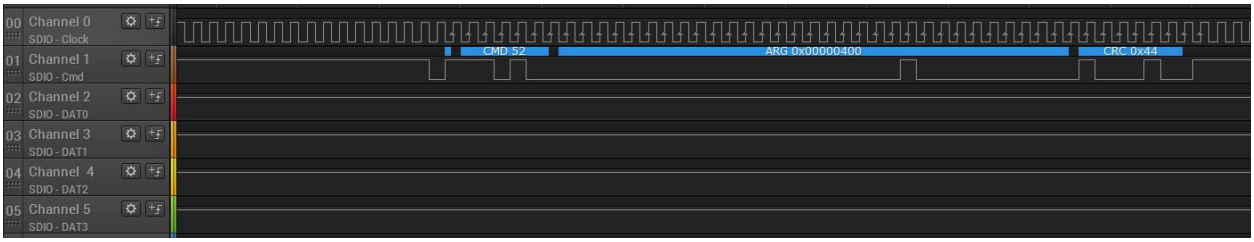
Figure 10-16. Enable Function1 I/O Register Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1002 and CRC:0x09

If RAW flag is set as '1', register with address '0x02' is read using `sdio_readb()` function immediately after writing using `sdio_writeb()`.

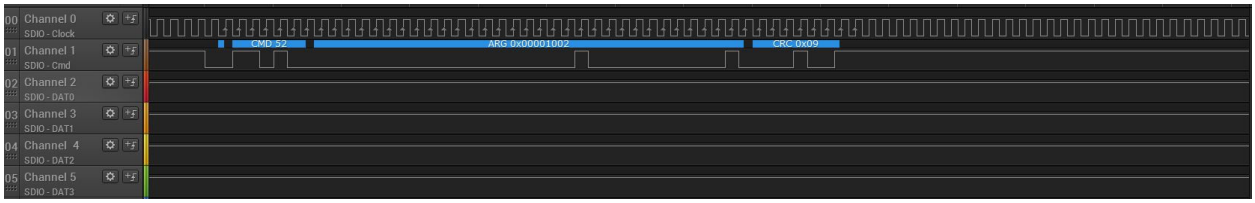
Figure 10-17. Read Function1 I/O Register Command from Host



It has DIR: from Host, CMD:0x34, ARG:0x400 which has R/W flag as 0, function number as 0, RAW flag a '0', register address as 0x02, read data as 0x00 and CRC:0x44

Response from ATWILC is as follows.

Figure 10-18. Read Function1 I/O Register Response from Slave



As per CMD52 response R5, argument value of 0x00001002 which has response flag as 0x10, read data as 0x02. The read data is as same as the one written previously. This means function 1 is enabled (IE1) in CCCR.

After enabling function 1 as above, make sure function 1 is ready by reading IOR1 bit (I/O Ready) in CCCR with address 0x03. The register must return value of 0x02 to set the IOR1 bit.

Figure 10-19. Read Function1 IOR Register

```
/**
 *      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;
}
```

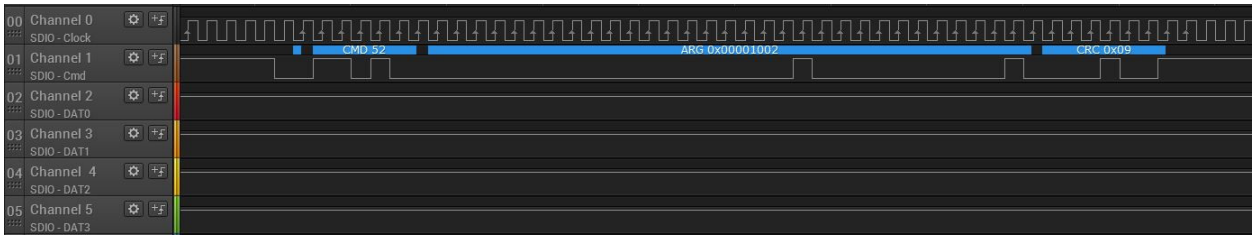
SDIO logs are as follows.

Figure 10-20. Read Function1 IOR Register Command from Host

DIR: from Host, CMD:0x34, ARG:0x600 and CRC:0x52.

ATWILC's response are as follows.

Figure 10-21. Read Function1 IOR Register Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1002 and CRC:0x09.

From above argument, read data is returned as '0x02' which means IOR1 is set and function 1 is up.

Set function 1 block size as `WILC_SDIO_BLOCK_SIZE` which is 512. Address 0x110-0x111 holds I/O block size for Function 1. This is sent as two CMD52 commands.

Figure 10-22. Set Function1 Block Size

```
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 logs are as follows.

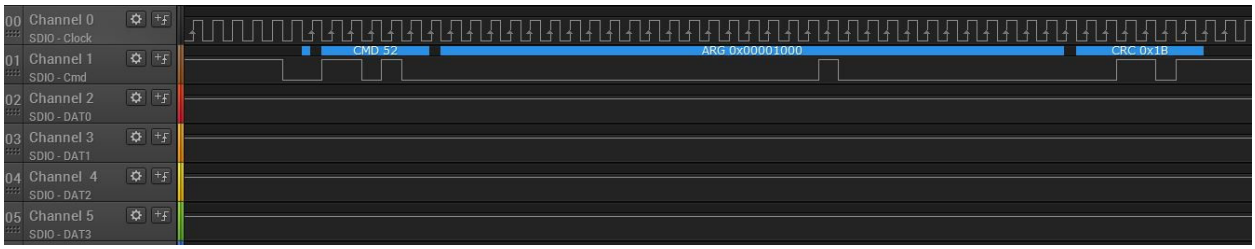
Figure 10-23. Set Function1 Block Size Command from Host

DIR: from Host, CMD:0x34, ARG:0x80022000 and CRC:0x5F

The above command set data is 0x00 (last byte of 512, 0x0200) to address 0x110.

ATWILC's response is as follows.

Figure 10-24. Set Function1 Block Size Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1000 and CRC:0x1B

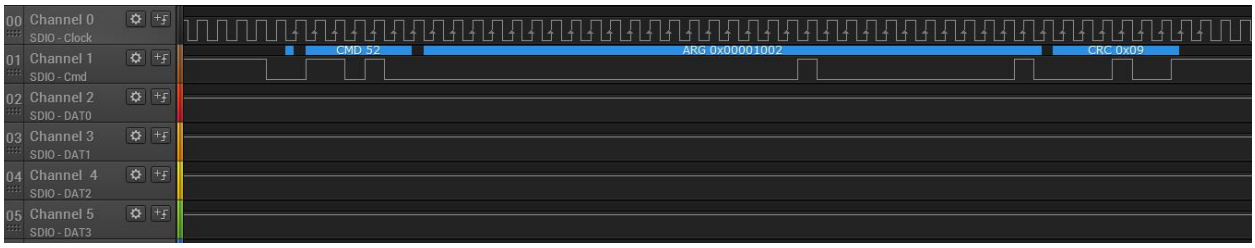
The next command set data is 0x02 (first byte of 512, 0x0200) to address 0x111.

Figure 10-25. Set Function1 Block Size Command from Host

DIR: from Host, CMD:0x34, ARG:0x80022202 and CRC:0x5B

ATWILC's response is as follows.

Figure 10-26. Set Function1 Block Size Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1002 and CRC:0x09

Set Interrupt enable for function 1. To set this IEN1 and IENM bit in CCCR register in address 0x04 needs to be set.

Figure 10-27. Card Common Control Registers (CCCR) with Address 0x04

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

IENx: Interrupt Enable for function x. If this bit is cleared to 0, any interrupt from this function shall not be sent to the host. If this bit is set to 1, then this function's interrupt shall be sent to the host if the master Interrupt Enable (bit 0) is also set to 1.

IENM: R/W Interrupt Enable Master. If this bit is cleared to 0, no interrupts from this card shall be sent to the host. If this bit is set to 1, then any function's interrupt shall be sent to the host.

Figure 10-28. Enable Function0 Interrupt

```
/**
 *      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;
}
```

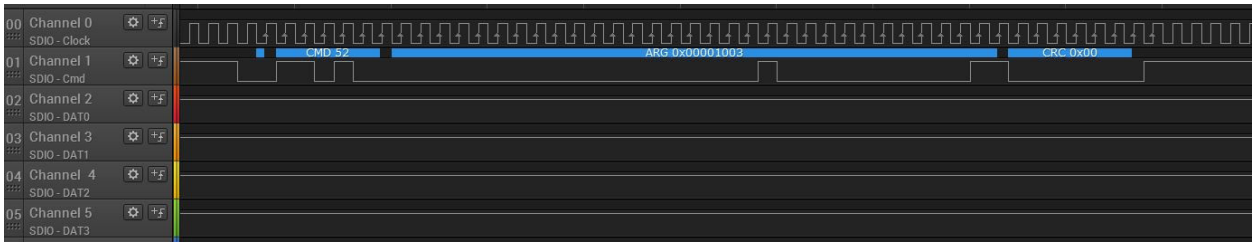
Commands sent from host to ATWILC are as follows.

Figure 10-29. Enable Function0 Interrupt Command from Host

DIR: from Host, CMD:0x34, ARG:0x80000803 and CRC:0x30

ATWILC's response is as follows.

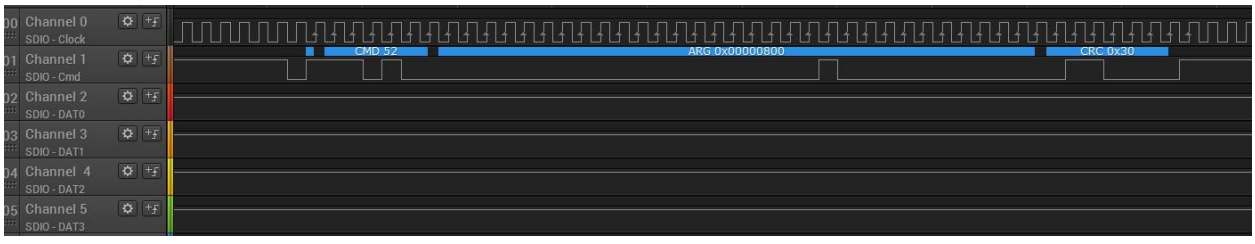
Figure 10-30. Enable Function0 Interrupt Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1003 and CRC:0x00

Since RAW flag is set as 1, the same register is read again to make sure IE1 and IENM bits are set. Commands sent from host to ATWILC are as follows.

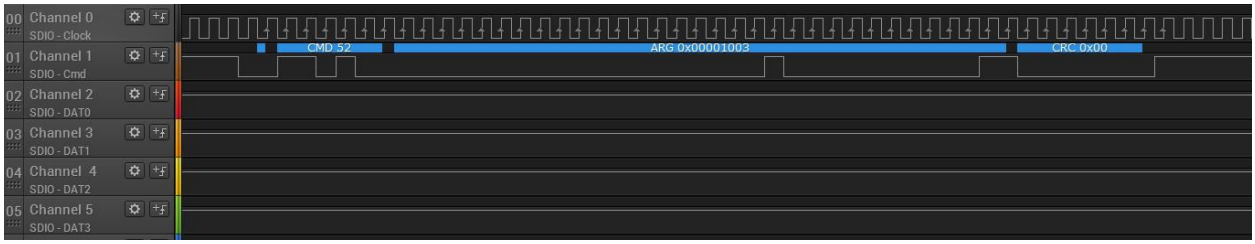
Figure 10-31. Read Function0 Interrupt Enable Command from Host



DIR: from Host, CMD:0x34, ARG:0x800 and CRC:0x30

ATWILC's response is as follows.

Figure 10-32. Read Function0 Interrupt Enable Response from Slave



DIR: from Slave, CMD:0x34, ARG:0x1003 and CRC:0x00. From above log, it is returning the data as 0x03 which indicates IE1 and IENM bits are set.

Read chip ID from the device to know if the device is ATWILC1000 or ATWILC3000.

Figure 10-33. Read Chip ID from the Device

```
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;
}
```

Read register with address 0x3b0000 to see if the Chip ID is ATWILC3000 and if it is not, then read register with address 0x1000 to get ATWILC1000 Chip ID. `wilc->hif_func->hif_read_reg()` will call either `sdio_read_reg()` or `wilc_spi_read_reg()` based on the host interface used.

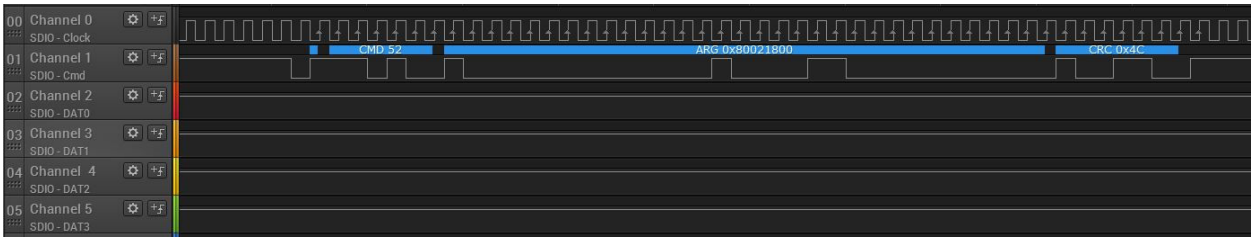
Since SDIO interface is used, `sdio_read_reg()` will be called. Address 0xF0-0xFF is reserved for Vendor Unique registers and CMD52 is used to access these registers and for others CMD53 is used.

Before read/write operation, address pointer to Function CSA needs to be set first. These three bytes make up a 24-bit pointer to the desired byte in the CSA to read or write. CSA is already enabled in previous commands. Address 0x10C-0x10E holds pointer to Function 1 Code Storage Area (CSA).

```
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);
```

The following commands set address 0x3b000000 to address 0x10C-0x10E.

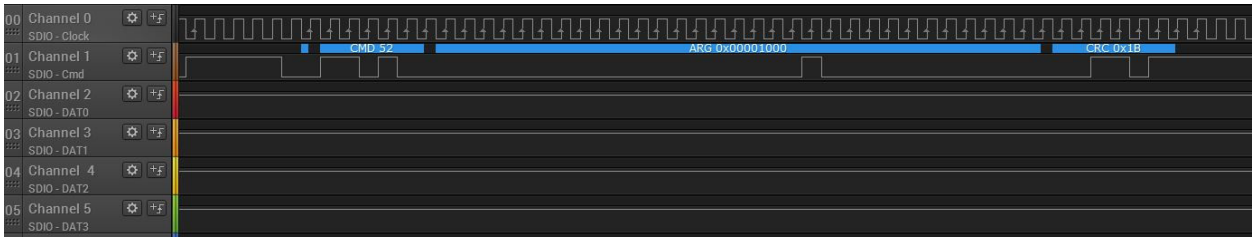
Figure 10-34. Set Pointer to Function 1 Code Storage Area Command from the Host



DIR: from Host, CMD:0x34, ARG:0x80021800 and CRC:0x4C. It sets data 0x00 (last byte of 0x3b00000) to address 0x10C0.

ATWILC's response is as follows.

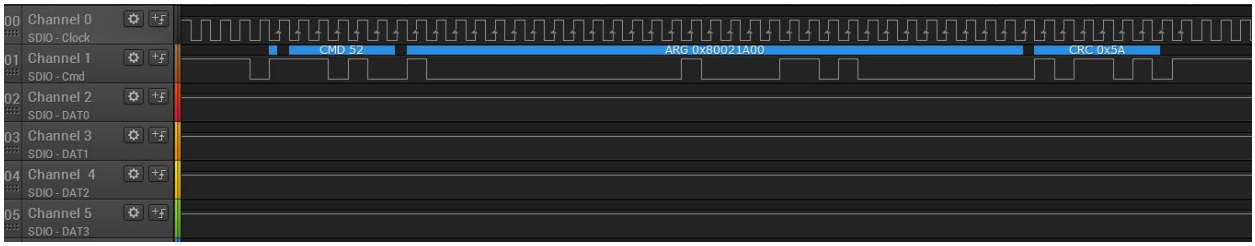
Figure 10-35. Set Pointer to Function 1 Code Storage Area Response from the Slave



DIR: from Slave, CMD:0x34, ARG:0x1000 and CRC:0x1B

Next command sets data 0x00 (second byte of 0x3b00000) to address 0x10D0.

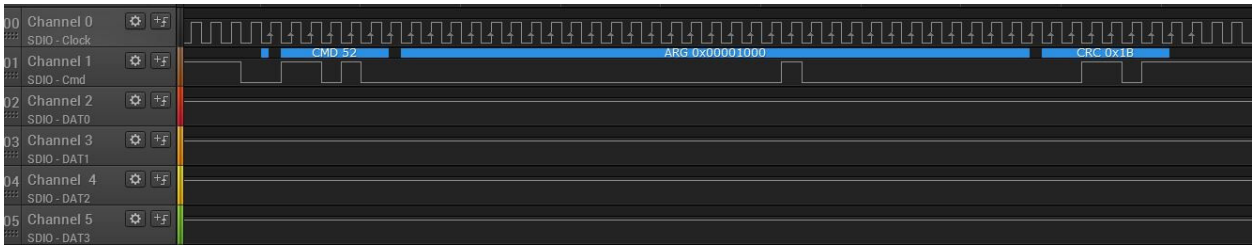
Figure 10-36. Set Pointer to Function 1 Code Storage Area Command from the Host



DIR: from Host, CMD:0x34, ARG:0x80021A00 and CRC:0x5A

ATWILCC response is as follows.

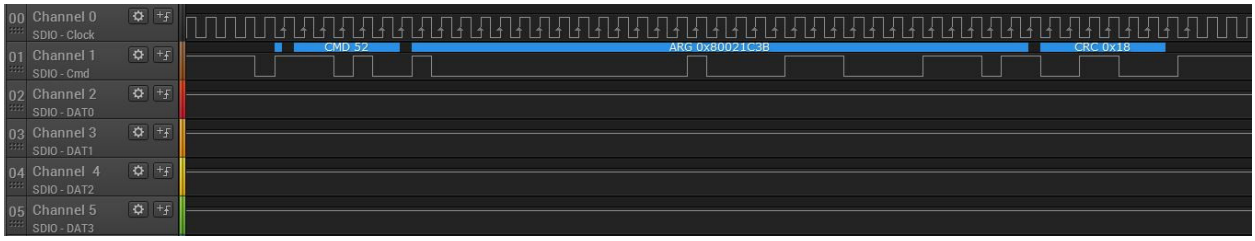
Figure 10-37. Set Pointer to Function 1 Code Storage Area Response from the Slave



DIR: from Slave, CMD:0x34, ARG:0x1000 and CRC:0x1B

The next command sets data 0x3B (first byte of 0x3b00000) to address 0x10E0.

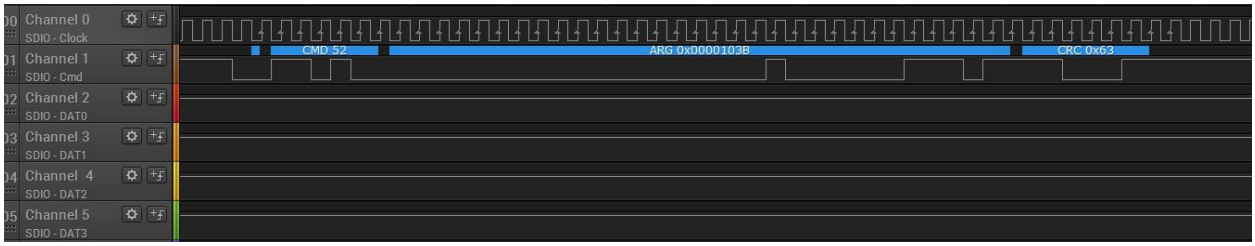
Figure 10-38. Set Pointer to Function 1 Code Storage Area Command from the Host



DIR: from Host, CMD:0x34, ARG:0x80021C3B and CRC:0x18

ATWILC's response is as follows.

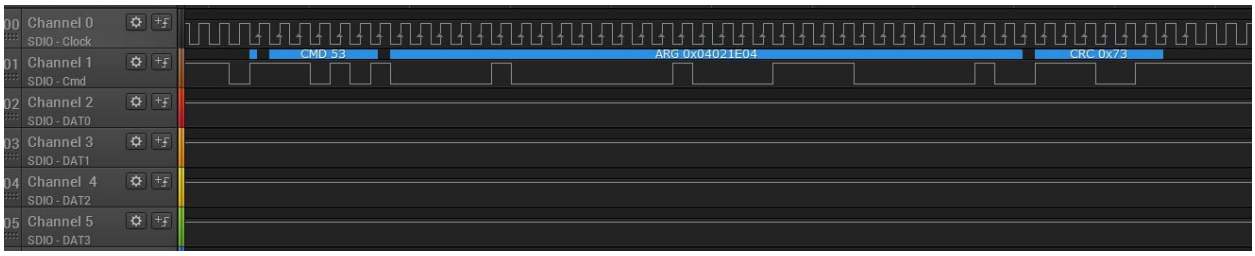
Figure 10-39. Set Pointer to Function 1 Code Storage Area Response from the Slave



DIR: from Slave, CMD:0x34, ARG:0x103B and CRC:0x63

Send CMD53 to read data from above set address by using byte transfer (Block mode=0). The address 0x10F is a data access window to Function 1 Code Storage Area (CSA). Byte count for this transfer is set as 4.

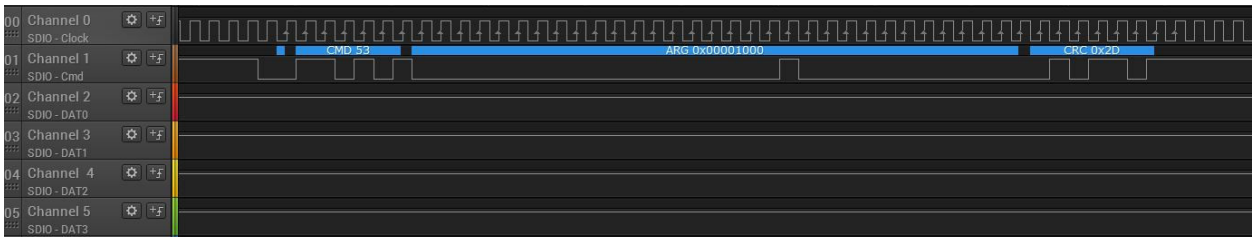
Figure 10-40. CMD53 to read data from Host



DIR: from Host, CMD:0x35, ARG:0x4021E04 and CRC:0x73

ATWILC's response is as follows.

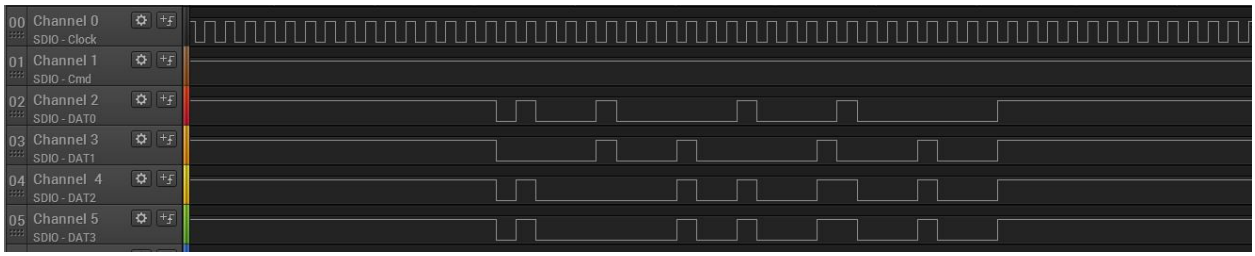
Figure 10-41. CMD53 Read Response from Slave



DIR: from Slave, CMD:0x35, ARG:0x1000 and CRC:0x2D

CHIP ID is returned from ATWILC using DAT[0]:DAT[3] lines. For example, ATWILC3000 Chip ID is 003000D0 and is returned as follows. Start and end bits, as well as the CRC bits, are transmitted for every one of the DAT lines. CRC bits are calculated and checked for every DAT line individually.

Figure 10-42. CHIPID Response from Slave



`sdio_init()` function is executed successfully and the response from ATWILC is successfully sent to the host. The ATWILC Initialization is done successfully. This can be seen in kernel logs as follows.

Figure 10-43. ifconfig wlan0 up Command Logs

```
# 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
```

IRQ GPIO is requested and initialized in the function `init_irq()`. After this, kernel threads are initialized for transmission and debugging using the `wlan_initialize_threads()` function. Its logs are as follows.

Figure 10-44. ifconfig wlan0 up Command Logs

```
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
```

Download the Wi-Fi firmware based on device AWILC1000 or ATWILC3000. It is performed using two functions `wilc_wlan_get_firmware()` and `wilc_firmware_download()`. Once the firmware is downloaded successfully, start the firmware using `linux_wlan_start_firmware()` function. If there is an error in starting the firmware, this function will return a negative value.

Once firmware is started successfully, the firmware version can be seen from kernel log. Configure firmware parameters such as operation mode, BSS type, power management and so on., in `linux_wlan_init_test_config()`.

Figure 10-45. ifconfig wlan0 up Command Logs

```
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. Appendix D - Troubleshooting Kernel Bootup Issue

If the .dtb and the zImage size is greater than the default size, the kernel does not boot up. Example kernel log is as follows.

```
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 ...
```

The procedure to edit the environment commands to fix above error is as follows.

1. Press any key when the `Hit any key to stop autoboot:` command is shown to stop autoboot. This "=>" symbol appears if successful.
2. Enter `pri` to print environment variables.

   ```
   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. If the dtb and zimage is bigger than the default size, use the following command to adjust the size.

   ```
   editenv bootcmd
   ```

   - Replace `0x0000817C` to `0x0000a17C` for `bootcmd` variable (dtb file size).
   - Replace `0x003DF7C8` to `0x004DF7C8` (zImage file size).
   - Enter `saveenv` command to save the command.
   - Enter `pri` to print the saved environment variables.
4. Type `boot` to restart the booting process with the saved values.

12. Document Revision History

Revision	Date	Section	Description
70005329D	08/2019	• 3.1 Enabling BlueZ 5.x Package using Build Root Options • 8. Appendix A - Loading ATWILC Module • 9. Appendix B - ATWILC SDIO Communication • 10. Appendix C - ATWILC SDIO Protocol Example • 11. Appendix D - Troubleshooting Kernel Bootup Issue	Added new sections
		• 4.1 ATWILC Power Control • 4.4 UART DMA	Updated the sections
70005329C	12/2018	• Kernel Modifications • Kernel Configuration • Buildroot Modifications • ATWILC Power Control • ATWILC Power On • Rescan SDIO Card • SPI • General Purpose IOs	Updated the sections

.....continued

Revision	Date	Section	Description
70005329B	07/2018	• Chapter 2. Kernel Modifications. • Section 4.2 SDIO. • Subsection 4.1.2 ATWILC Power On, 4.2.1 Rescan SDIO Card, 4.2.2 SDIO Card Detect, 4.3 SPI, 4.4 UART DMA, and 5.6 Set BT TX Power.	Updated the sections
		Section 2.1 Driver Source Code Integration, 2.2 Firmware Integration, and 4.5 General Purpose IOs.	Added new sections
		• Chapter 3. Makefile Porting and Chapter 4. Antenna Switching • Section 5.5 Interrupt Request (IRQ).	Removed these sections
70005329A	08/2017	Document	Initial release

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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 logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

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 logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, 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, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-4938-6

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820