# GigaDevice Semiconductor Inc.

# GD32VW553 基本指令用户指南

应用笔记

**AN153**

1.0 版本

（2023 年 10 月）

# 目录

# 图索引

# 表索引

# 1. 用户基本指令

使用 USB 线将测试机与开发板连接，打开 UART 工具，连接到正确的 COM 口。开发板上电并正确启动后，通过 UART 工具下发指令，开发板即可根据指令内容完成相应操作。

本手册中，指令后面< >代表该选项必填，[]代表该选项选填。注意指令严格执行大小写。

## 1.1. help

该指令没有选项。

如*图 1-1. help 指令*所示，help 指令会将开发板支持的所有指令列出。

**注意：**BLE 相关指令需要通过 ble_help 指令查看。

**图 1-1. help 指令**

```
# help
==============================
      ble_help
==============================
      help
      reboot
      tasks
      free
      cpu_stats
      sys_ps
      ping
      join_group
      iperf
      iperf3
      wifi_debug
      wifi_open
      wifi_close
      wifi_mac_addr
      wifi_concurrent
      wifi_auto_conn
      wifi_scan
      wifi_connect
      wifi_connect_bssid
      wifi_disconnect
      wifi_status
      wifi_set_ip
      wifi_ps
      wifi_monitor
      wifi_ap
      wifi_ap_adv
      wifi_stop_ap
#
```

## 1.2. reboot

该指令没有选项。

执行该指令后开发板将重启，串口会打印启动信息。该指令与 reset 按键作用类似。

## 1.3. tasks

该指令没有选项。

执行该指令后将打印 task 相关信息，包括状态，优先级，自任务创建以来该 task stack 剩余的最小空间，task 序号以及 task 所用的 stack 的 base 地址。如*图1-2. tasks 指令*所示。

**图 1-2. tasks 指令**

```
# tasks
TaskName          State Pri   Stack ID    StackBase
-------------------------------------------------
CLI task          X     20    388   1     0x20020580
WiFi core task    R     18    550   7     0x20024c68
IDLE              R     0     172   9     0x20026b10
tcpip_thread      B     19    336   4     0x20022df0
Tmr Svc           B     19    172   10    0x20026f90
wifi_mgmt         B     17    828   8     0x20025a90
BLE APP task      B     17    316   3     0x20021af8
BLE task          S     18    646   2     0x20020e78
RX                B     18    384   5     0x20023b80
TX                B     20    148   6     0x20024788
```

## 1.4. free

该指令没有选项。

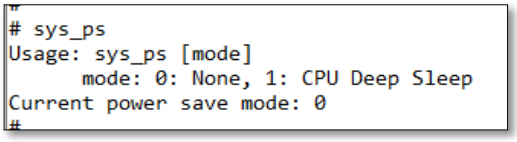
执行该指令后将打印 heap 相关信息，包括剩余 heap，已用 heap，最大使用 heap，最大可用 heap 以及各个可用的 mem block 地址和大小。如*图1-3. free 指令*所示。

**图 1-3. free 指令**

```
#
# free
RTOS HEAP: free=145976 used=36620 max_used=52348/182596
[0]=0x0x20025b68, 56
[1]=0x0x200264e8, 24
[2]=0x0x20027010, 24
[3]=0x0x20027038, 40
[4]=0x0x200272a8, 1480
[5]=0x0x20027bd0, 3768
[6]=0x0x20028ac0, 107824
[7]=0x0x20048000, 32760
[8]=0x0x2004fff8, 0
#
```

## 1.5. sys_ps

**图 1-4. sys_ps 指令**




该指令使用方法如*图 1-4. sys_ps 指令*所示，mode 有 3 种，

不填：不进行任何设置，仅打印当前 CPU power save 模式；

0：禁用 CPU power save。

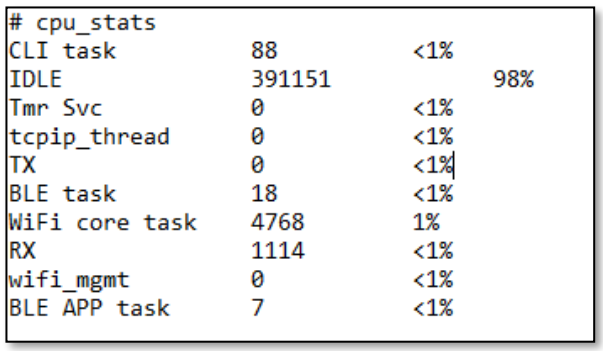
1：启用 CPU power save，模式是 deep sleep。当 CPU 处于空闲状态时，将自动进入 deep sleep，之后可由 wifi/ble 自动唤醒或是通过 uart rx 事件主动唤醒。

## 1.6. cpu_stats

该指令没有选项。

执行该指令后将打印各个 task 的 CPU 使用情况，包括处在 Running 状态的时间和 CPU 占用率。如*图 1-5. cpu_stats 指令*所示。

**图 1-5. cpu_stats 指令**




## 1.7. WIFI

此目录下是 wifi 相关指令的介绍。

### 1.7.1. wifi_open

该指令没有选项。

该指令用于使能 wifi 功能。执行其他 wifi 相关命令时，需要已经使能 wifi。开发板正确启动后，wifi 默认使能，因此不需要执行该指令来重复使能 wifi。该指令通常与 wifi_close 相配合，在

wifi_close 将 wifi 关闭后重新使能 wifi。如果 wifi 已使能，串口会给予提示。

### 1.7.2. wifi_close

该指令没有选项。

wifi_close 可以关闭 wifi，此后一些指令将无法执行，如 wifi_scan、wifi_connect 等。

开发板处于不同情况下，指令执行结果不同，如下：

- 开发板已经与 AP 连接，则会将开发板与 AP 断连，然后关闭 wifi；
- 开发板未与 AP 连接，则直接关闭 wifi；
- 开发板为 softAP 模式，且有 sta 与开发板连接，则会断开该连接，再关闭 wifi；
- 开发板为 softAP 模式，没有 sta 连接，则直接关闭 wifi；
- wifi 已关闭，则串口会提示 wifi 已关闭。

### 1.7.3. wifi_debug

- Usage: wifi_debug <0 or 1>

该指令用于控制 wifi 相关 debug log 信息的打印。0 表示关闭打印；1 表示开启打印。

### 1.7.4. wifi_scan

该指令没有选项，执行该指令时开发板不可处于 softap 模式。

执行该指令后会打印出开发板扫描到的 AP 信息，包括 RSSI，channel，BSSID，SSID 和加密方式。如*图 1-6. wifi_scan 指令*所示。

**图 1-6. wifi_scan 指令**

```
# wifi_scan
# WIFI_SCAN: done
[0] (-34 dBm) CH=  1 BSSID=c4:70:ab:d9:bd:11 SSID=OpenWrt [OPEN]
[1] (-30 dBm) CH=  1 BSSID=1c:5f:2b:fd:be:60 SSID=D-Link_DIR-822  [RSN:WPA-PSK CCMP/CCMP]
[2] (-42 dBm) CH=  1 BSSID=86:e5:81:9b:d4:05 SSID=fly  [RSN:WPA-PSK CCMP/CCMP]
[3] (-47 dBm) CH=  1 BSSID=ba:fa:07:50:63:f6 SSID=Redmi K40  [RSN:WPA-PSK CCMP/CCMP]
[4] (-50 dBm) CH=  1 BSSID=08:3a:38:cc:2f:d0 SSID=GD-internet [OPEN]
[5] (-50 dBm) CH=  1 BSSID=08:3a:38:cc:2f:d1 SSID=GD-guest [OPEN]
[6] (-50 dBm) CH=  1 BSSID=08:3a:38:cc:2f:d2 SSID=GD-lan [OPEN]
[7] (-32 dBm) CH=  6 BSSID=88:c3:97:0d:c3:70 SSID=xiaomi_4a [OPEN]
[8] (-23 dBm) CH=  4 BSSID=68:77:24:bd:86:59 SSID=TP-LINK_8659  [RSN:WPA-PSK,SAE CCMP/CCMP][MFP:AES-128-CMAC]
[9] (-22 dBm) CH=  4 BSSID=72:77:24:bd:86:59 SSID=  [RSN:WPA-PSK CCMP/CCMP]
[10] (-22 dBm) CH=  5 BSSID=a2:aa:95:39:57:72 SSID=HUAWEI_AX3000  [RSN:WPA-PSK,SAE CCMP/CCMP][MFP:AES-128-CMAC]
[11] (-23 dBm) CH=  6 BSSID=60:3a:7c:26:f3:a0 SSID=tplink_8690 [OPEN]
[12] (-48 dBm) CH=  6 BSSID=08:3a:38:cc:2d:f1 SSID=GD-guest [OPEN]
[13] (-48 dBm) CH=  6 BSSID=08:3a:38:cc:2d:f2 SSID=GD-lan [OPEN]
[14] (-47 dBm) CH=  6 BSSID=08:3a:38:cc:2d:f0 SSID=GD-internet [OPEN]
[15] (-49 dBm) CH=  6 BSSID=0e:cc:cb:36:80:24 SSID=WuMingming  [RSN:WPA-PSK CCMP/CCMP]
[16] (-42 dBm) CH=  6 BSSID=ee:cb:9d:ce:33:ad SSID=yzq  [RSN:WPA-PSK,SAE CCMP/CCMP][MFP:AES-128-CMAC]
[17] (-41 dBm) CH=  6 BSSID=00:22:6b:60:0a:98 SSID=cisco  [RSN:WPA-PSK CCMP/CCMP]
[18] (-45 dBm) CH=  6 BSSID=82:8c:b8:9f:24:8b SSID=wlan_test  [RSN:WPA-PSK CCMP/CCMP]
[19] (-72 dBm) CH=  6 BSSID=08:3a:38:cc:0f:12 SSID=GD-lan [OPEN]
[20] (-55 dBm) CH= 11 BSSID=d6:4f:86:cb:c8:d0 SSID=iQOO Neo5  [RSN:WPA-PSK CCMP/CCMP]
[21] (-42 dBm) CH=  9 BSSID=50:eb:f6:06:8a:18 SSID=RT-AX56U [OPEN]
[22] (-69 dBm) CH= 11 BSSID=08:3a:38:cc:27:71 SSID=GD-guest [OPEN]
[23] (-22 dBm) CH= 11 BSSID=8c:53:c3:d8:0d:fd SSID=xiaomi_wifi6  [RSN:WPA-PSK CCMP/CCMP]
[24] (-69 dBm) CH= 11 BSSID=08:3a:38:cc:27:70 SSID=GD-internet [OPEN]
[25] (-69 dBm) CH= 11 BSSID=08:3a:38:cc:27:72 SSID=GD-lan [OPEN]
```

### 1.7.5. wifi_concurrent

- Usage: wifi_concurrent [0 or 1]

该指令用于控制 wifi concurrent 模式的使能。0 表示关闭，1 表示使能，当不设置选项时，仅打印当前使能状态。

使用该指令需要打开宏 CFG_WIFI_CONCURRENT，该宏位于 MSDK\macsw\export\wlan_config.h 文件。

### 1.7.6. wifi_connect

- Usage: wifi_connect <SSID> [PASSWORD]

该指令用于连接 AP，执行该指令时开发板不可处于 softap 模式。

- wifi_connect <SSID>

用于连接没有加密的 AP。

- wifi_connect <SSID> <PASSWORD>

用于连接加密的 AP。

连接过程如*图 1-7. wifi_connect 指令*所示，串口打印出了连接过程信息。如果在已连接 AP 的情况下再执行 wifi_connect 指令，开发板会先与原 AP 断开，再连接新的 AP。

**图 1-7. wifi_connect 指令**

```
# wifi_connect xiaomi_4a
[0] (-34 dBm) CH=  6 BSSID=88:c3:97:0d:c3:70 SSID=xiaomi_4a [OPEN]
MAC: auth req send
MAC: auth rsp received, status = 0
MAC: assoc req send
MAC: assoc rsp received, status = 0
WIFI_MGMT: DHCP got ip 192.168.3.127
#
# wifi_connect TP-LINK_8659 12345678
MAC: deauth send
[0] (-22 dBm) CH=  4 BSSID=68:77:24:bd:86:59 SSID=TP-LINK_8659  [RSN:WPA-PSK,SAE CCMP/CCMP][MFP:AES-128-CMAC]
SAE: commit send
SAE: commit received
SAE: confirm send, status_code = 0
SAE: confirm received, status_code = 0
MAC: assoc req send
MAC: assoc rsp received, status = 0
WPA: 4-1 received
WPA: 4-2 send
WPA: 4-3 received
WPA: 4-4 send
WIFI_MGMT: DHCP got ip 192.168.1.100
#
```

### 1.7.7. wifi_connect_bssid

- Usage: wifi_connect_bssid <BSSID> [PASSWORD]

该指令与 wifi_connect 指令类似，只是选项中的 SSID 变成了 BSSID，使用方法不变。

### 1.7.8. wifi_disconnect

该指令没有选项。

执行该指令后开发板将与 AP 断开。执行成功串口会打印信息：

MAC: deauth send

MGMT: disconnect complete

### 1.7.9. wifi_auto_conn

- Usage: wifi_auto_conn [0 or 1]

该指令用于设置是否开机自动连接 AP。0 表示不自动连接，1 表示自动连接，当不设置选项时，仅打印当前设置。

如果设置了自动连接，再次连接 AP 成功就会将 AP 信息保存到 flash 中，多次连接 AP 只会将最后成功连接的 AP 记为有效 AP，开发板重启后将根据 flash 中的 AP 信息自动连接 AP。如果设置自动连接后没有连接 AP，开发板重启后将不会自动连接 AP。

### 1.7.10. wifi_status

该指令没有选项。

执行该指令后串口将打印当前开发板的 wifi 状态。

wifi 当前有三种模式，分别是 SoftAP，MONITOR 和 STATION。不同模式下指令打印的信息有不同，如*图 1-8. wifi_status 指令*所示。

**图 1-8. wifi_status 指令**

```
# wifi_status
WIFI Status:
==============================
WiFi VIF[0]: 76:ba:ed:71:09:10
SoftAP
       Status: Started
       SSID: ap_test
       Channel: 6
       Security: WPA2
       IP: 192.168.237.1
       Client[0]:   76:ba:ed:ff:ff:02   192.168.237.150

# wifi_status
WIFI Status:
==============================
WiFi VIF[0]: 76:ba:ed:71:09:10
Monitor

# wifi_status
WIFI Status:
==============================
WiFi VIF[0]: 76:ba:ed:71:09:10
STA
       Status: Connected
       SSID: TP-LINK_8659
       BSSID: 68:77:24:bd:86:59
       Channel: 4
       Bandwidth: 0
       Security: WPA3
       RSSI: -22
       IP: 192.168.1.100

# wifi_status
WIFI Status:
==============================
WiFi VIF[0]: 76:ba:ed:71:09:10
STA
       Status: Disconnected
```

第一行是当前 wifi 设备的 mac 地址；第二行当前 wifi 设备的模式，即上述三种模式中的一种。

AP 模式下，会显示状态，SSID，channel，加密方式以及 IP 地址，如果存在连接到此 ap 的设备，还会显示这些设备的信息，包括 mac 地址和 IP 地址，多个设备依次排序。

STATION 模式下，WIFI Status 指示当前 wifi 设备是否已连接到 ap，Connected 表示已连接，Disconnected 表示未连接。已连接情况下会显示该 ap 的 SSID，BSSID，channel 等信息。

### 1.7.11. wifi_monitor

- Usage: wifi_monitor stop | start <channel>

该指令使用方法如*图 1-9. wifi_monitor 指令*所示。指令 wifi_monitor start <channel>用于启动 monitor 模式，需指定监听的 channel；指令 wifi_monitor stop 用于关闭 monitor 模式并切换到 station 模式。

**图 1-9. wifi_monitor 指令**

```
#
# wifi_monitor
Usage: wifi_monitor stop | start <channel>
start: start the monitor mode.
<channel>: 1~14.
stop: stop the monitor mode.
#
```

### 1.7.12. wifi_ps

- Usage: wifi_ps <mode>

**图 1-10. wifi_ps 指令**

```
# wifi_ps
Usage: wifi_ps <mode>
      mode: 0: off, 1: always on, 2: dynamic on
#
```

该指令使用方法如*图 1-10. wifi_ps 指令*所示，mode 有 3 种，

0：禁用 power save；

1：启用 power save，模式是 Normal mode，wifi 模块将一直处于 power save 模式；

2：启用 power save，模式是 Dynamic mode，wifi 模块将根据 wifi TX/RX 的流量决定是否进入或退出 power save 模式；

### 1.7.13. wifi_ap

- Usage: wifi_ap <ssid> <password> <channel> [-a <akm>[,<akm 2>]] [-hide <hide_ap>]

该指令用于开启或关闭 softap 模式，使用方法如*图 1-11. wifi_ap 指令*所示。

图 1-11. wifi_ap 指令

```
#
# wifi_ap
Usage: wifi_ap <ssid> <password> <channel> [-a <akm>[,<akm 2>]] [-hide <hide_ap>]
<ssid>: The length should be between 1 and 32.
<password>: The length should be between 8 and 63, but can be "NULL" indicates open ap.
<channel>: 1~13.
[-a <akm>[,<akm 2>]]: only support following 5 AKM units: open; wpa2; wpa3; wpa2,wpa3 or wpa3,wpa2,
default wpa2.
[-hide <hide_ap>]: 0 means broadcast ssid or 1 means hidden ap, default 0.
for example:
    wifi_ap test_ap NULL 1 -a open -hide 0, means an open ap in channel 1 and can broadcast ssid.
    wifi_ap test_ap 12345678 1, means an WPA2 ap in channel 1.
#
```

其中，ssid 不支持中文字符。password 填为"NULL"时，表明启用一个 open ap，-a 配置将被忽略，此外若开启加密的 AP 且未配置-a 选项指定加密方式，则默认为 wpa2 加密。

### 1.7.14. wifi_ap_adv

该指令与 wifi_ap 指令使用方法相同。

### 1.7.15. wifi_stop_ap

该指令没有选项，执行该指令后 Softap 模式将停止，且转为 station 模式。

### 1.7.16. wifi_set_ip

Usage: wifi_set_ip dhcp |<ip_addr> <gate_way>

该指令用于手动设置静态 IP 或者通过 DHCP 方式自动获取 IP，只在开发板处于 station 模式时起效。使用方法如*图 1-12. wifi_set_ip 指令*所示。

图 1-12. wifi_set_ip 指令

```
# wifi_set_ip
wifi_set_ip: invalid input
Usage: wifi_set_ip dhcp |<ip_addr/mask_bits> <gate_way>
      dhcp: get ip by start dhcp
      ip_addr: ipv4 addr needded to set. eg: 192.168.0.123
      gate_way: eg: 192.168.0.1
Example: wifi_set_ip 192.168.0.123/24 192.168.0.1
#
```

### 1.7.17. wifi_mac_addr

- Usage: wifi_mac_addr [xx:xx:xx:xx:xx:xx]

该指令用于设置 wifi 的临时 mac 地址，reboot 或断电重启后失效。

不设置选项，仅将打印当前 mac 地址。

## 1.8. ping

- Usage: ping <target_ip | stop> [-n count] [-l size] [-i interval] [-t total time]

该指令用于进行 ping test。

target_ip 是对端地址。IPv4 格式是<ipv4_addr>，IPv6 是<-6 ipv6_addr>（如果使能了 IPv6）。

其中，count 是 ping 包的数量；size 是包长度，单位是 byte；interval 是发包间隔，单位是 ms；total time 是总运行时间，单位是 s。默认情况下 count 为 5，size 为 120，interval 为 10，total time 不使用；如果使用 total time 选项，count 与 interval 选项将不起作用，interval 默认为 1000ms，count 将等于 total time 值。

ping 指令的使用方法如*图 1-13. ping 指令*所示，

**图 1-13. ping 指令**

```
16:04:22.596  # ping 192.168.1.1
16:04:22.599  # [ping_test] PING 192.168.1.1 120 bytes of data
16:04:22.647  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=1 time=19 ms
16:04:22.648  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=2 time=1 ms
16:04:22.649  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=3 time=2 ms
16:04:22.698  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=4 time=4 ms
16:04:22.700  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=5 time=1 ms
16:04:22.702  [ping_test] 5 packets transmitted, 5 received, 0% packet loss
16:04:22.703  [ping_test] delay: min 1 ms, max 19 ms, avg 5 ms
16:04:23.769
16:04:31.693  # ping 192.168.1.1 -n 3
16:04:31.694  # [ping_test] PING 192.168.1.1 120 bytes of data
16:04:31.697  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=1 time=1 ms
16:04:31.698  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=2 time=1 ms
16:04:31.702  [ping_test] 120 bytes from 192.168.1.1: icmp_seq=3 time=1 ms
16:04:31.742  [ping_test] 3 packets transmitted, 3 received, 0% packet loss
16:04:31.743  [ping_test] delay: min 1 ms, max 1 ms, avg 1 ms
16:04:32.457
16:04:39.214  # ping 192.168.1.1 -n 3 -l 1000
16:04:39.217  # [ping_test] PING 192.168.1.1 1000 bytes of data
16:04:39.218  [ping_test] 1000 bytes from 192.168.1.1: icmp_seq=1 time=1 ms
16:04:39.265  [ping_test] 1000 bytes from 192.168.1.1: icmp_seq=2 time=1 ms
16:04:39.266  [ping_test] 1000 bytes from 192.168.1.1: icmp_seq=3 time=1 ms
16:04:39.270  [ping_test] 3 packets transmitted, 3 received, 0% packet loss
16:04:39.272  [ping_test] delay: min 1 ms, max 1 ms, avg 1 ms
16:04:39.826
16:05:02.193  # ping 192.168.1.1 -n 3 -l 500 -i 5000
16:05:02.194  # [ping_test] PING 192.168.1.1 500 bytes of data
16:05:02.196  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=1 time=1 ms
16:05:07.231  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=2 time=6 ms
16:05:12.209  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=3 time=3 ms
16:05:12.211  [ping_test] 3 packets transmitted, 3 received, 0% packet loss
16:05:12.215  [ping_test] delay: min 1 ms, max 6 ms, avg 3 ms
16:05:15.208
16:11:03.842  # ping 192.168.1.1 -n 3 -l 500 -i 5000 -t 5
16:11:03.844  # [ping_test] PING 192.168.1.1 500 bytes of data
16:11:03.845  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=1 time=8 ms
16:11:04.859  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=2 time=3 ms
16:11:05.876  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=3 time=1 ms
16:11:06.843  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=4 time=1 ms
16:11:07.860  [ping_test] 500 bytes from 192.168.1.1: icmp_seq=5 time=1 ms
16:11:07.861  [ping_test] 5 packets transmitted, 5 received, 0% packet loss
16:11:07.867  [ping_test] delay: min 1 ms, max 8 ms, avg 2 ms
```

- ping stop

ping stop 用于终止 ping test，如*图 1-14. ping stop 指令*所示，

**图 1-14. ping stop 指令**

```
# ping 192.168.1.1 -n 3 -l 500 -i 5000 -t 50
# [ping_test] PING 192.168.1.1 500 bytes of data
[ping_test] 500 bytes from 192.168.1.1: icmp_seq=1 time=1 ms
[ping_test] 500 bytes from 192.168.1.1: icmp_seq=2 time=1 ms
[ping_test] 500 bytes from 192.168.1.1: icmp_seq=3 time=1 ms
[ping_test] 500 bytes from 192.168.1.1: icmp_seq=4 time=1 ms
ping stop
# [ping_test] 4 packets transmitted, 4 received, 0% packet loss
[ping_test] delay: min 1 ms, max 1 ms, avg 1 ms
```

## 1.9. join_group

- Usage: join_group <group ip eg:224.0.0.5>

执行该指令前开发板必须已连接到 AP。执行该指令后开发板将加入一个多播组，例如：

- join_group 224.0.0.5

期间使用 sniffer 可以在指令执行后抓到开发板发出的 IGMP 协议包。

## 1.10. iperf3

iperf3 指令使用 iperf3 进行网络速度测试。

### 1.10.1. iperf3 -h

如*图 1-15. iperf3 -h 指令*所示，串口将打印出 iperf3 指令相关选项。

**图 1-15. iperf3 -h 指令**

```
#
# iperf3 -h
Usage:
    iperf3 <-s|-c hostip|stop|-h> [options]
Server or Client:
    -i #         seconds between periodic bandwidth reports
    -p #         server port to listen on/connect to
Server specific:
    -s           run in server mode
Client specific:
    -c <host>    run in client mode, connecting to <host>
    -u           use UDP rather than TCP
    -b #[KMG][/#] target bandwidth in bits/sec (0 for unlimited)
                 (default 1 Mbit/sec for UDP, unlimited for TCP)
                 (optional slash and packet count for burst mode)
    -t #         time in seconds to transmit for (default 10 secs)
    -l #[KMG]    length of buffer to read or write
    -S #         set the IP 'type of service'
#
```

### 1.10.2. iperf3 -s [options]

- iperf3 -s

开启一个 iperf3 server，默认监听端口 5201 上 TCP/UDP 数据。其他选项为默认值。

- -p <port>

设置服务端监听的端口，port 范围 0-65535，默认 5201。

举例：iperf3 -s -p 5003

服务端在 5003 端口监听。

- -i <interval>

设置串口打印的测试结果的周期（Interval 这一列），单位为 second（秒），范围是 0.1-60 以及 0。当设置为 0 时代表不打印周期性报告，只输出最终的测试结果。默认是 4。

举例：iperf3 -s -i 0.5，

串口打印测试结果的周期为 0.5s。

### 1.10.3. iperf3 -c <host> [options]

- iperf3 -c <host>

开启一个 iperf3 的 client 端，并与 IP 为<host>的 server 在默认端口 5201 进行 TCP 连接，其他选项均为默认值。

- -u

开启一个 iperf 的 client 端，并与 ip 为<host>的 server 在默认端口 5201 进行 UDP 连接。-u 选项通常与-b 选项联合使用，指定发送的数据带宽。

- -p <port>

设置客户端连接的端口，需与服务端监听的端口相同。

- -i <interval>

-i 选项设置与服务端相同。

- -b <bandwidth/number>

bandwidth 单位为 bits/sec，格式为：data[KMG]。如 50K、50k 或 50000，表示带宽设置为 50Kbits/sec；当 bandwidth 为 0 时，表示没有限制。udp 默认 1 Mbit/sec，tcp 连接下无限制。

bandwidth 后面不加“/number”时，iperf3 会根据每个数据包的长度，算出达到指定带宽每秒需要发送的数据包数量，然后每个数据包以平均时间间隔发送。

举例：iperf3 -c 192.168.3.132 -u -b 200k

bandwidth 后面加“/number”时，进入 burst mode，iperf3 会一次性连续发送指定数量（number）的数据包，中间没有间隔，但每一批次之间有间隔，且间隔均匀。

举例：iperf3 -c 192.168.3.132 -u -b 200k/60

- -t <time>

设置数据传输的时间，以秒为单位，默认值为 10。

- -l <length>

设置读写 buffer 的长度，单位为 byte，格式为：data[KMG]，与-n 选项相同。udp 模式下该值建议设置为 1472，tcp 模式下设置为 1460。

- -S <QOS value>

设置出栈数据包的 QOS 服务类型。Number 范围为 0-255，可以使用 16 进制（0x 前置符）、8 进制（0 前置符）和 10 进制，如 0x16 == 026 == 22。

### 1.10.4. iperf3 stop

该指令用于终止 iperf3 测试。

### 1.10.5. iperf3 test example

- 开发板与测试机连接同一个 AP，然后查看自身 IP。
  - 开发板使用 wifi_connect 指令连接 AP，wifi_status 指令查看 IP。
- 测试机打开 iperf3 指令窗口，开始测试。
  - server 端先执行指令：iperf3 -s -p <port> -i <interval>
  - client 端随即执行指令：iperf3 -c <host> -l <length> -p <port> -i <interval> -u -b <bandwidth/number> -t <time>
  - 其中，-l、-p、-i、-u、-b、-t 选项可选。-p 选项必须 server 与 client 同时使用且值相同；-i 选项两端可不同时使用且值可不同；
  - 例如：
  - iperf3 -s -p 5004 -i 1
  - iperf3 -c 192.168.1.104 -l 1460 -p 5004 -i 2 -t 20 //TCP
  - iperf3 -c 192.168.1.104 -l 1472 -p 5004 -i 4 -t 30 -u -b 50M //UDP
- server 端执行指令后会在窗口看到打印信息，告诉我们 server 已打开且在对应 port 监听，client 端执行指令后测试机与开发板会同时打印测试信息。

## 1.11. iperf

iperf 指令调用 iperf2 进行网络速度测试。iperf 默认运行在 tcp 模式，udp 模式必须使用-u 选项指定。下面是指令的相关选项（注意大小写）。

### 1.11.1. iperf -h

如*图 1-16. iperf -h 指令*所示，串口将打印出 iperf 指令相关选项。

图 1-16. iperf -h 指令

```
# iperf -h
Usage:
    iperf <-s|-c hostip|exit|-h> [options]
Client/Server:
    -u #      use UDP rather than TCP
    -i #      seconds between periodic bandwidth reports
    -l #      length of buffer to read or write (default 1460 Bytes)
    -p #      server port to listen on/connect to (default 5001)
Server specific:
    -s        run in server mode
Client specific:
    -b #      bandwidth to send at in bits/sec (default 1 Mbit/sec, implies -u)
    -S #      set the IP 'type of service'
    -c <host> run in client mode, connecting to <host>
    -t #      time in seconds to transmit for (default 10 secs)
#
```

### 1.11.2. iperf -s [options]

- iperf -s

开启一个 iperf2 的 TCP 模式的 server，默认在 5001 端口监听，其他选项为默认值。

- iperf -s -u

开启一个 iperf2 的 UDP 模式的 server，默认在 5001 端口监听，其他选项为默认值。

- -i <interval>

设置串口打印的测试结果的周期（Interval 这一列），单位为 second（秒），范围是 1-3600 之间的整数(非整数向下取整)。默认是 1。

- -l <length>

设置读写缓冲区的长度，单位是 byte，默认是 1460bytes，udp 最大值为 2380，tcp 最大值为 4380。udp 建议值为 1472，tcp 为 1460。

- -p <port>

设置服务端监听的端口。port 范围 0-65535，默认 5001。

### 1.11.3. iperf -c <host> [options]

- iperf -c <host>

开启一个 iperf2 的 client 端，并与 ip 为<host>的 server 在默认端口 5001 进行 TCP 连接，其他选项为默认值。

- iperf -c <host> -u

开启一个 iperf3 的 client 端，并与 ip 为<host>的 server 在默认端口 5001 进行 UDP 连接，其他选项为默认值。

- -i <interval>
- -l <length>

-l、-i 选项设置与服务端相同。

- -p <port>

设置客户端去连接的端口，与服务端监听的端口相同。

- -b <bandwidth>

bandwidth 单位为 bits/sec，格式为：data[KMG]。如 50K、50k 或 50000，表示带宽为 50Kbits/sec；当 bandwidth 为 0 时，表示没有限制。默认为 1 Mbit/sec。只在 UDP 模式使用。

- -t <time>

设置传输的总时间。默认是 10 秒。

- -S <QOS value>

设置 IP 数据包的 QOS 服务类型。number 范围为 0-255，可以使用 16 进制（0x 前置符）或 10 进制，如 0x16 = 22。

### 1.11.4. iperf exit

该指令用于终止 iperf2 测试。

### 1.11.5. iperf2 test example

- 开发板与测试机连接同一个 AP，然后查看自身 IP。
  - 开发板使用 wifi_connect 指令连接 AP，wifi_status 指令查看 IP。
  - 测试机打开 iperf2 指令窗口，开始测试。
  - server 端先执行指令：
  - iperf -s -p <port> -i <interval> -l <length> //TCP
  - iperf -s -p <port> -i <interval> -l <length> -u //UDP
  - client 端随即执行指令：
  - iperf -c <host> -l <length> -p <port> -i <interval> -b <bandwidth/number> -t <time> -S <number>//TCP
  - iperf -c <host> -l <length> -p <port> -i <interval> -u -b <bandwidth/number> -t <time> -S <number>//UDP
  - 其中，-l、-p、-i、-u、-b、-t、-S 选项可选。
  - !! 注意：-p 选项必须 server 与 client 同时使用且值相同；-i 选项两端可不同时使用且值可不同；-u 选项必须 server 与 client 同时使用。
  - 例如：
  - iperf -s -p 5004 -i 1 //TCP
  - iperf -s -p 5004 -i 1 –u //UDP
  - iperf -c 192.168.1.104 -l 1460 -p 5004 -i 2 -t 20 -S 0xe0 //TCP
  - iperf -c 192.168.1.104 -l 1472 -p 5004 -i 4 -t 30 -S 0xe0 -u -b 50M //UDP
- server 端执行指令后会在窗口看到打印信息，告诉我们 server 已打开且在对应 port 监听，client 端执行指令后测试机与开发板会同时打印测试信息。

## 1.12. BLE

此目录下是 ble 相关指令的介绍。

### 1.12.1. ble_help

该指令没有选项。

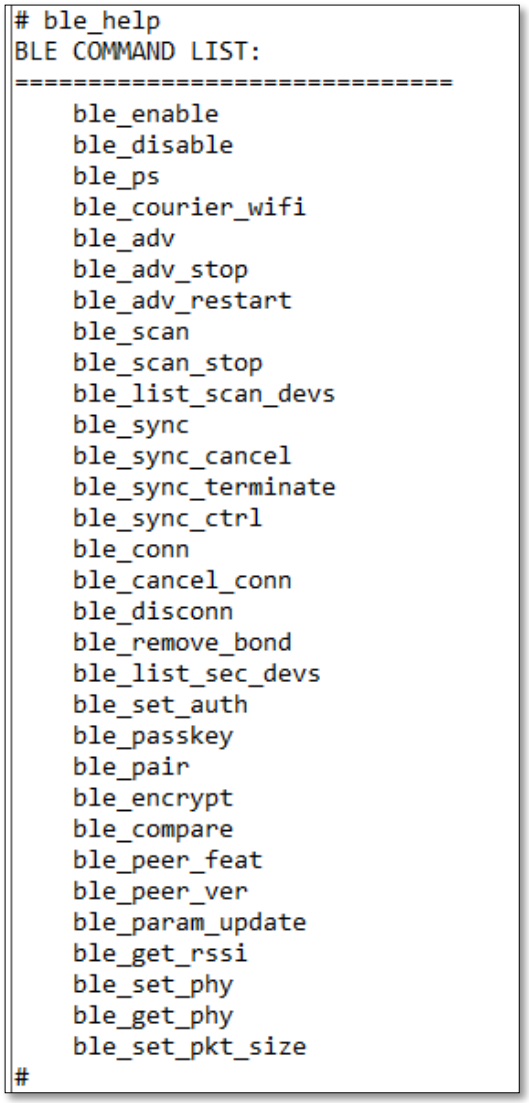
如*图 1-17. ble_help 指令（msdk configuration）*及

*图 1-18. ble_help 指令（msdk_ffd* **configuration**）所示，ble_help 指令会将 ble 所有指令列出。根据 configuration 的不同，可以使用的 ble 指令也会有所区别，所以 ble_help 指令列出来的内容也会不一样。

**图 1-17. ble_help 指令（msdk configuration）**

```
ble_help
BLE COMMAND LIST:
==============================
    ble_enable
    ble_disable
    ble_ps
    ble_courier_wifi
    ble_adv
    ble_adv_stop
    ble_adv_restart
    ble_disconn
    ble_remove_bond
    ble_list_sec_devs
    ble_set_auth
    ble_pair
    ble_encrypt
    ble_passkey
    ble_compare
    ble_peer_feat
    ble_peer_ver
    ble_param_update
    ble_get_rssi
#
```

图 1-18. ble_help 指令（msdk_ffd configuration）



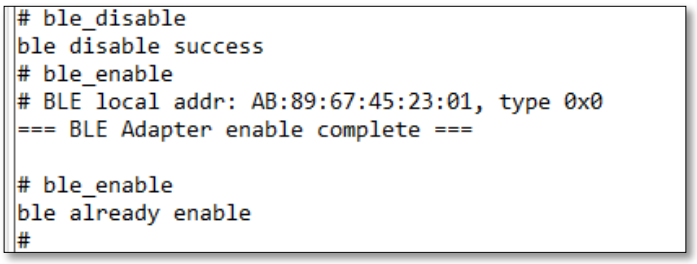

### 1.12.2. ble_enable

该指令没有选项。

ble_enable 用于打开 ble，执行其他 ble 相关命令时，需要在 ble 打开的情况下才有效。开发板正确启动后，ble 默认打开，因此不需要执行该指令来重复打开 ble。该指令通常与 ble_disable 相配合，在 ble 关闭后使用指令 ble_enable，ble 会进入初始状态，并不会恢复成 ble_disable 前的状态。

如*图 1-19. ble_enable 指令*所示，ble 关闭后执行 ble_enable，ble 将打开，串口显示 reset 的日志；若 ble 已打开，串口会提示 ble 已打开。

图 1-19. ble_enable 指令




### 1.12.3. ble_disable

该指令没有选项。

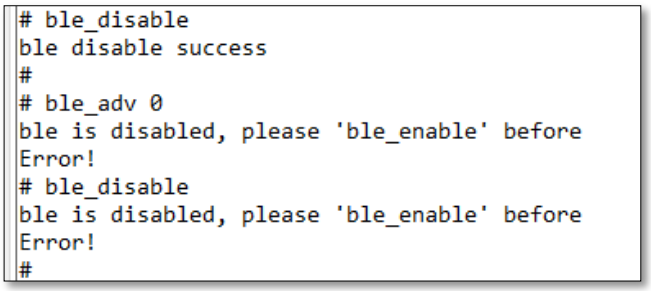
ble_disable 可以关闭 ble，此后一些指令将无法执行，如 ble_adv, ble_scan, ble_conn 等。

该指令执行后会对 ble 软硬件执行 reset 动作，然后关闭 ble，因此开发板处于不同场景下的执行结果会略有差异，例如：

- 开发板未打开 ble 任何功能，则直接关闭 ble；
- 开发板已经建立了 connection，则会将开发板与 peer 断线，然后关闭 ble；
- 开发板打开了 advertising，则会将开发板 stop advertising，然后关闭 ble；
- 开发板打开了 scanning，则会将开发板 stop scanning，然后关闭 ble；
- ble 已关闭，则串口会提示 ble 已关闭。

如*图 1-20. ble_disable 指令*所示，ble_disable 执行后会打印提示。

图 1-20. ble_disable 指令




### 1.12.4. ble_ps

- Usage: ble_ps <0 or 1>

该指令用来配置 ble 的 power save 功能，默认是启用状态。当 ps mode 为 1 时，启用 power save 模式，在没有任务处理或者 adv/scan interval 间隔时间大于 5ms 时，软件会让 ble core 进入 sleep，来节省功耗。当 ps mode 为 0 时，禁用 power save 模式，ble core 不会进入 sleep 状态。

如*图 1-21. ble_ps 指令*所示，ble_ps 执行后会打印提示。

图 1-21. ble_ps 指令

```
# ble_ps
Current ps mode: 1
Usage: ble_ps <0 or 1>
    0: ble not deep sleep
    1: ble deep sleep
# ble_ps 0
ble_ps config complete. ps mode: 0
# ble_ps 1
ble_ps config complete. ps mode: 1
#
```

### 1.12.5. ble_courier_wifi

- Usage: ble_courier_wifi <0:disable or 1:enable>

该指令用来打开或关闭蓝牙配网(配置 wifi 网络)功能，默认该功能是关闭的。打开该功能后，设备会发送 advertising 报文供手机端发现，可以使用微信小程序“GD 蓝牙配网”进行操作。关闭该功能后，advertising 会被关闭。

如*图 1-22. ble_courier_wifi 指令*所示，ble_courier_wifi 执行后会打印提示。

图 1-22. ble_courier_wifi 指令

```
# ble_courier_wifi
Usage: ble_courier_wifi <0:disable; 1:enable>
#
# ble_courier_wifi 1
bcwl_adv_mgr_evt_hdlr adv[255] state change 0x0 ==> 0x1, reason 0x0
ble_courier_wifi ret:0
# bcwl_adv_mgr_evt_hdlr adv[1] state change 0x1 ==> 0x2, reason 0x0
bcwl_adv_mgr_evt_hdlr adv[1] state change 0x2 ==> 0x3, reason 0x0
bcwl_adv_mgr_evt_hdlr adv[1] state change 0x3 ==> 0x4, reason 0x0
bcwl_adv_mgr_evt_hdlr adv[1] state change 0x4 ==> 0x6, reason 0x0

# ble_courier_wifi 0
ble_courier_wifi ret:0
# bcwl_adv_mgr_evt_hdlr adv[1] state change 0x6 ==> 0x2, reason 0x0
bcwl_adv_mgr_evt_hdlr adv[255] state change 0x2 ==> 0x0, reason 0x0
```

### 1.12.6. ble_adv

- Usage: ble_adv <adv type>

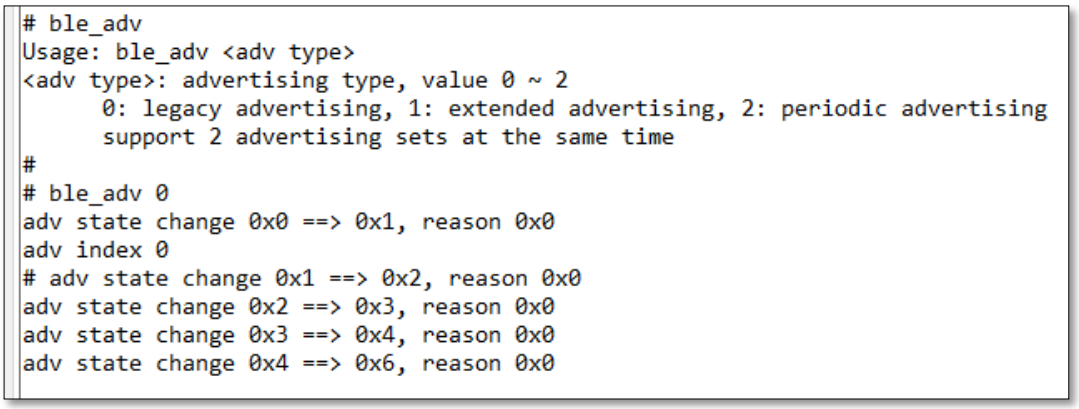
该指令用于打开 advertising，使本地设备可以被其它 BLE 设备发现并连接，通过 adv type 可以设置广播类型为 legacy advertising(scannable connectable undirected),extended advertising(connectable undirected), periodic advertising(undirected periodic)。同时刻最大可支持 2 组 advertising。

在被其它设备成功连接后对应的 advertising 会被停止，但不会被删除。

如*图 1-23. ble_adv 指令*所示，ble_adv 执行后会打印提示，当 adv state 为 0x6 的时候，表示成功，否则表示执行失败。adv index 也会提示出来，可用于 ble_adv_stop 或 ble_adv_restart

指令，例如下图的 adv idx 为 0。

**图 1-23. ble_adv 指令**



```
# ble_adv
Usage: ble_adv <adv type>
<adv type>: advertising type, value 0 ~ 2
      0: legacy advertising, 1: extended advertising, 2: periodic advertising
      support 2 advertising sets at the same time
#
# ble_adv 0
adv state change 0x0 ==> 0x1, reason 0x0
adv index 0
# adv state change 0x1 ==> 0x2, reason 0x0
adv state change 0x2 ==> 0x3, reason 0x0
adv state change 0x3 ==> 0x4, reason 0x0
adv state change 0x4 ==> 0x6, reason 0x0
```

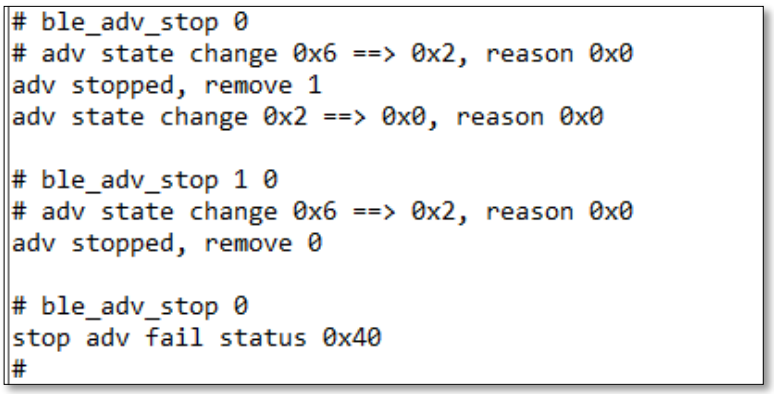
### 1.12.7. ble_adv_stop

- Usage: ble_adv_stop <adv idx> [remove]
- adv idx：advertising index，执行 ble_adv 命令的 log 中可以获取
- remove：表示 stop advertising 后是否需要 remove 操作，默认值为 1，advertising stop 后会被 remove；若配置值为 0，将不会 remove advertising，可以通过 ble_adv_restart 再次开启 advertising，该操作会比 ble_adv 开启 advertising 少一个创建的过程。

该指令用于关闭 advertising。

如*图 1-24. ble_adv_stop 指令*所示，ble_adv_stop 执行后会打印提示。当 stop 一个非法的 adv idx 时，会提示 fail 并给出非 0 的 status。

**图 1-24. ble_adv_stop 指令**



```
# ble_adv_stop 0
# adv state change 0x6 ==> 0x2, reason 0x0
adv stopped, remove 1
adv state change 0x2 ==> 0x0, reason 0x0

# ble_adv_stop 1 0
# adv state change 0x6 ==> 0x2, reason 0x0
adv stopped, remove 0

# ble_adv_stop 0
stop adv fail status 0x40
#
```

### 1.12.8. ble_adv_restart

- Usage: ble_adv_restart <adv idx>
- adv idx：advertising index，执行 ble_adv 命令的 log 中可以获取

该指令用于重新开启 advertising。以下两个场景的 advertising 可以通过 ble_adv_restart 重新

start：一是在 ble_adv 打开 advertising 后作为 slave 建立连线，对应的 advertising 被 stop；二是执行“ble_adv_stop <idx> 0”后，对应的 advertising 处于 stop 状态没有被 remove。

如*图 1-25. ble_adv_restart 指令*所示，ble_adv_restart 执行后会打印提示，当 adv state 为 0x6 时，表示 restart success，否则为失败；若 adv idx 为非法的 index，将会打印失败日志。

**图 1-25. ble_adv_restart 指令**

```
# ble_adv 0
adv state change 0x0 ==> 0x1, reason 0x0
adv index 0
# adv state change 0x1 ==> 0x2, reason 0x0
adv state change 0x2 ==> 0x3, reason 0x0
adv state change 0x3 ==> 0x4, reason 0x0
adv state change 0x4 ==> 0x6, reason 0x0
ble_adv_stop 0 0
# adv state change 0x6 ==> 0x2, reason 0x0
adv stopped, remove 0

# ble_adv_restart 0
# adv state change 0x2 ==> 0x6, reason 0x0

# ble_adv_restart 1
restart adv fail 0x40
#
```

### 1.12.9. ble_scan

该指令没有选项。

该指令仅在 msdk_ffd configuration 下可以使用。

用于打开 scan 功能，扫描到的设备信息会被打印出来，包括设备地址、设备地址类型、rssi、name 和 dev idx 等，其中 dev idx 可用来 connect 或 sync。扫描到的设备信息会被一直记录直至开始新一次的 scan 或者执行 ble_disable。可以使用 ble_scan_stop 停止 scan 功能。

如*图 1-26. ble_scan 指令*所示，ble_scan 执行后会打印提示。

**图 1-26. ble_scan 指令**

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr A0:0B:16:90:45:D4, addr type 0x0, rssi -93, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr B8:7C:6F:A9:80:91, addr type 0x0, rssi -76, sid 0xff, dev idx 1, peri_adv_int 0, name
new device addr 61:A2:D2:6C:AB:32, addr type 0x1, rssi -91, sid 0xff, dev idx 2, peri_adv_int 0, name
new device addr 7D:F5:F7:70:77:8C, addr type 0x1, rssi -65, sid 0xff, dev idx 3, peri_adv_int 0, name
new device addr 79:C8:B9:04:03:AA, addr type 0x1, rssi -62, sid 0xff, dev idx 4, peri_adv_int 0, name
new device addr 05:55:95:51:C4:D7, addr type 0x1, rssi -81, sid 0xff, dev idx 5, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0
```

### 1.12.10. ble_scan_stop

该指令没有选项。

该指令仅在 msdk_ffd configuration 下可以使用。

用于关闭 scan 功能。success 后 status 为 0，否则 fail。

如*图 1-27. ble_scan_stop 指令*所示，ble_scan_stop 执行后会打印提示。

**图 1-27. ble_scan_stop 指令**

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr A0:0B:16:90:45:D4, addr type 0x0, rssi -93, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr B8:7C:6F:A9:80:91, addr type 0x0, rssi -76, sid 0xff, dev idx 1, peri_adv_int 0, name
new device addr 61:A2:D2:6C:AB:32, addr type 0x1, rssi -91, sid 0xff, dev idx 2, peri_adv_int 0, name
new device addr 7D:F5:F7:70:77:8C, addr type 0x1, rssi -65, sid 0xff, dev idx 3, peri_adv_int 0, name
new device addr 79:C8:B9:04:03:AA, addr type 0x1, rssi -62, sid 0xff, dev idx 4, peri_adv_int 0, name
new device addr 05:55:95:51:C4:D7, addr type 0x1, rssi -81, sid 0xff, dev idx 5, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0
```

### 1.12.11. ble_list_scan_devs

该指令没有选项。

该指令仅在 msdk_ffd configuration 下可以使用。

用于查询最近一次 scan 到的设备，会显示 dev idx 和 device addr。

如*图 1-28. ble_list_scan_devs 指令*所示，ble_list_scan_devs 执行后会打印提示。

**图 1-28. ble_list_scan_devs 指令**

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr A0:0B:16:90:45:D4, addr type 0x0, rssi -93, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr B8:7C:6F:A9:80:91, addr type 0x0, rssi -76, sid 0xff, dev idx 1, peri_adv_int 0, name
new device addr 61:A2:D2:6C:AB:32, addr type 0x1, rssi -91, sid 0xff, dev idx 2, peri_adv_int 0, name
new device addr 7D:F5:F7:70:77:8C, addr type 0x1, rssi -65, sid 0xff, dev idx 3, peri_adv_int 0, name
new device addr 79:C8:B9:04:03:AA, addr type 0x1, rssi -62, sid 0xff, dev idx 4, peri_adv_int 0, name
new device addr 05:55:95:51:C4:D7, addr type 0x1, rssi -81, sid 0xff, dev idx 5, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0

# ble_list_scan_devs
dev idx: 0, device addr: A0:0B:16:90:45:D4
dev idx: 1, device addr: B8:7C:6F:A9:80:91
dev idx: 2, device addr: 61:A2:D2:6C:AB:32
dev idx: 3, device addr: 7D:F5:F7:70:77:8C
dev idx: 4, device addr: 79:C8:B9:04:03:AA
dev idx: 5, device addr: 05:55:95:51:C4:D7
```

### 1.12.12. ble_sync

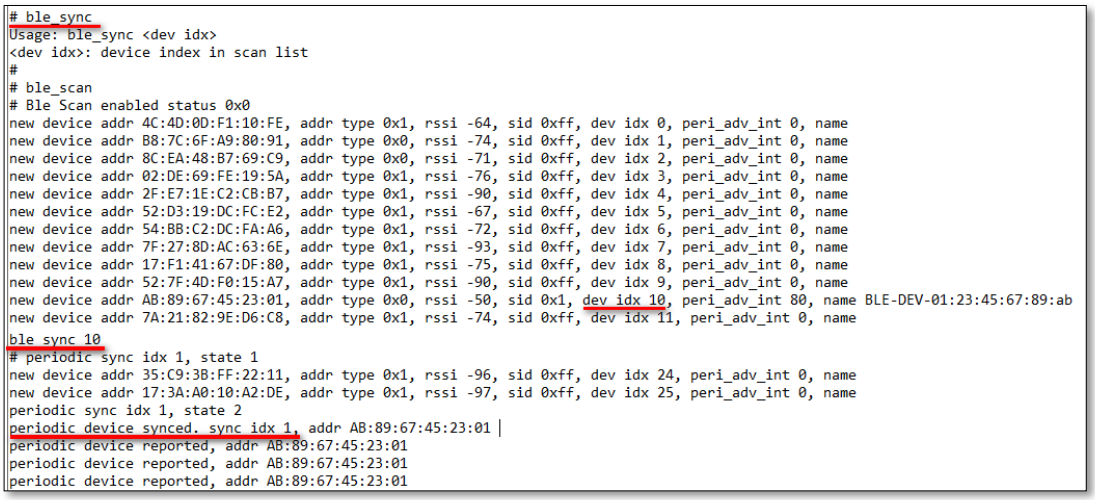
- Usage: ble_sync <dev idx>
- dev idx 需从 scan list 中获取。

该指令仅在 msdk_ffd configuration 下可以使用。

该指令用于 sync periodic advertising，建立 sync 的过程中需要保持 scan 功能打开，建立成功后才可以将 scan 功能关闭。sync 成功会打印 sync idx 日志，用于 ble_sync_terminate 或 ble_syc_ctrl 指令。该指令会默认打开 periodic advertising report 功能，因此在收到 periodic advertising 报文后 app 会打印相关日志，若需要关闭 report 功能，可使用 ble_sync_ctrl 指令。

如*图 1-29. ble_sync 指令*所示，ble_sync 执行后会打印提示。

**图 1-29. ble_sync 指令**




## 1.12.13. ble_sync_cancel

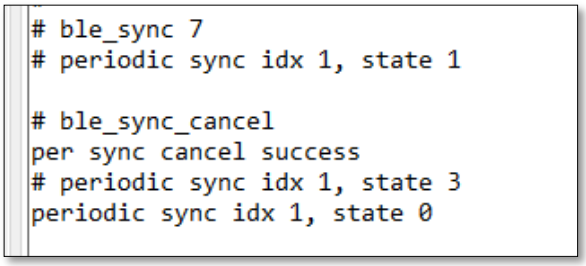
该指令没有选项。

该指令仅在 msdk_ffd configuration 下可以使用。

在使用 ble_sync 指令开始同步 periodic advertising 但没有成功同步上时，可以使用该命令来取消同步操作。

如*图 1-30. ble_sync_cancel 指令*所示，ble_sync_cancel 执行后会打印提示。

**图 1-30. ble_sync_cancel 指令**




## 1.12.14. ble_sync_terminate

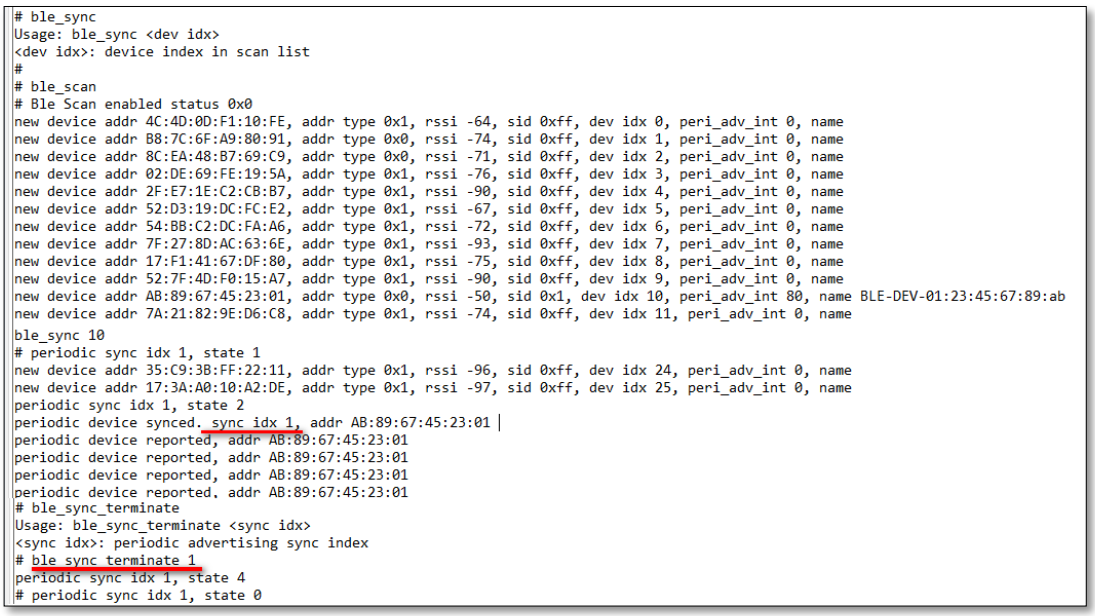
- Usage: ble_sync_terminate <sync idx>
- sync idx：需要从 ble_sync 指令创建 sync 成功的日志中获取。

该指令用于 terminate 指定的 sync 链路。

该指令仅在 msdk_ffd configuration 下可以使用。

如*图 1-31. ble_sync_terminate 指令*所示，ble_sync_terminate 执行后会打印提示。

图 1-31. ble_sync_terminate 指令




## 1.12.15. ble_sync_ctrl

- Usage: ble_sync_ctrl <sync idx> <report>
- sync idx：需要从 ble_sync 指令创建 sync 成功的日志中获取。

该指令仅在 msdk_ffd configuration 下可以使用。

该指令用于打开或关闭 periodic advertising report 功能，默认 report 功能是打开的，每次收到 sync 到的报文，均会上报至 app。

如*图 1-32. ble_sync_ctrl 指令*所示，ble_sync_ctrl 执行后会打印提示。

图 1-32. ble_sync_ctrl 指令

```
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
ble_sync_ctrl
Usage: ble_sync_ctrl <sync idx> <report>
<sync idx>: periodic advertising sync index
<report>: control bitfield for periodic advertising report
      bit 0: report periodic advertising event
# periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
periodic device reported, addr AB:89:67:45:23:01
ble_sync_ctrl 1 0
# periodic device report ctrl status 0x0
```

### 1.12.16. ble_conn

- Usage: ble_conn <dev idx>
- dev idx 需从 scan list 中获取。

该指令仅在 msdk_ffd configuration 下可以使用。

该指令用于主动发起连接，执行该命令前需要执行 ble_scan 获取扫描信息中的 dev idx，若没有扫描到对端设备，将无法建立连接。

如*图 1-33. ble_conn 指令*所示，ble_conn 执行后会打印提示。如果连接成功会打印下图红线 log，其中 conn idx 需要在 ble_disconn, ble_pair, ble_encrypt 等命令中用到。

图 1-33. ble_conn 指令

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr 36:35:B7:B1:CA:7D, addr type 0x1, rssi -75, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr 4B:73:32:D6:24:65, addr type 0x1, rssi -94, sid 0xff, dev idx 1, peri_adv_int 0, name
new device addr CC:89:67:45:23:01, addr type 0x0, rssi -41, sid 0xff, dev idx 2, peri_adv_int 0, name GD-BLE - 01:23:45:67:89:cc
new device addr B8:7C:6F:A9:80:91, addr type 0x0, rssi -74, sid 0xff, dev idx 3, peri_adv_int 0, name
new device addr 77:B1:A9:CC:E0:8B, addr type 0x1, rssi -94, sid 0xff, dev idx 4, peri_adv_int 0, name
new device addr 57:CB:E6:E5:05:93, addr type 0x1, rssi -91, sid 0xff, dev idx 5, peri_adv_int 0, name
new device addr 5E:02:4A:6A:18:68, addr type 0x1, rssi -63, sid 0xff, dev idx 6, peri_adv_int 0, name
new device addr 70:3F:81:48:EC:47, addr type 0x1, rssi -92, sid 0xff, dev idx 7, peri_adv_int 0, name
new device addr 49:55:1F:60:FA:7D, addr type 0x1, rssi -94, sid 0xff, dev idx 8, peri_adv_int 0, name
new device addr 45:A2:52:2B:DE:67, addr type 0x1, rssi -90, sid 0xff, dev idx 9, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0

# ble_conn
Usage: ble_conn <dev idx>
<dev idx>: dev index in scan list
#
# ble_conn 2
# ===> init conn starting idx 1, wl_used 0
===> init conn started idx 1, wl_used 0
connect success, conn_idx:0, conn_hdl:0x1
===> init conn idle idx 1, wl_used 0 reason 0x0
le pkt size ind: conn idx 0, tx oct 251, tx time 2120, rx oct 251, rx time 2120
conn_idx 0 encrypted, pairing_lvl 0x0 status 0x25
conn idx: 0, peer version: 0xb, subversion: 0xc, comp id 0xc2b
conn idx: 0, peer feature: 0x0000000ff70179ff
```

## 1.12.17. ble_cancel_conn

该指令没有选项。

该指令仅在 msdk_ffd configuration 下可以使用。

该指令用于取消未建立成功的连接。在执行 ble_conn 指令后并未成功连接时，可通过 ble_cancel_conn 来取消连接操作。若成功建立了连接，需要断开，可执行 ble_disconn 指令。

如*图 1-34. ble_cancel_conn 指令*所示，ble_cancel_conn 执行后会打印提示，当 init conn 进入 idle 状态下表示执行成功。

图 1-34. ble_cancel_conn 指令

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr 0A:E2:AC:E6:73:A0, addr type 0x1, rssi -97, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr CC:89:67:45:23:01, addr type 0x0, rssi -38, sid 0xff, dev idx 1, peri_adv_int 0, name GD-BLE - 01:23:45:67:89:cc
new device addr 4C:AD:03:32:B8:FF, addr type 0x1, rssi -72, sid 0xff, dev idx 2, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0

# ble_conn 1
# ===> init conn starting idx 1, wl_used 0
===> init conn started idx 1, wl_used 0

# ble_cancel_conn
# ===> init conn disabling idx 1, wl_used 0 reason 0x0
===> init conn idle idx 1, wl_used 0 reason 0x0

# ble_cancel_conn
cancel connect fail status 0x43
#
```

## 1.12.18. ble_disconn

- Usage: ble_disconn <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于断开已经建立的 connection。

如*图 1-35. ble_disconn 指令*所示，ble_disconn 执行后会打印提示。

图 1-35. ble_disconn 指令

```
# ble_scan
# Ble Scan enabled status 0x0
new device addr B8:7C:6F:A9:80:91, addr type 0x0, rssi -87, sid 0xff, dev idx 0, peri_adv_int 0, name
new device addr 6B:50:35:8E:6D:A4, addr type 0x1, rssi -96, sid 0xff, dev idx 1, peri_adv_int 0, name
new device addr CC:89:67:45:23:01, addr type 0x0, rssi -38, sid 0xff, dev idx 2, peri_adv_int 0, name GD-BLE - 01:23:45:67:89:cc
new device addr 5B:6E:DC:46:92:36, addr type 0x1, rssi -63, sid 0xff, dev idx 3, peri_adv_int 0, name
new device addr A0:0B:16:90:45:D4, addr type 0x0, rssi -96, sid 0xff, dev idx 4, peri_adv_int 0, name
new device addr 55:16:5F:A2:D9:55, addr type 0x1, rssi -72, sid 0xff, dev idx 5, peri_adv_int 0, name
new device addr 57:39:4F:F4:83:50, addr type 0x1, rssi -60, sid 0xff, dev idx 6, peri_adv_int 0, name
ble_scan_stop
# Ble Scan disabled  status 0x0
ble_conn 2
# ===> init conn starting idx 1, wl_used 0
===> init conn started idx 1, wl_used 0
connect success. conn idx:0, conn_hdl:0x1
===> init conn idle idx 1, wl_used 0 reason 0x0
le pkt size ind: conn idx 0, tx oct 251, tx time 2120, rx oct 251, rx time 2120
conn_idx 0 encrypted, pairing_lvl 0x0 status 0x25
conn idx: 0, peer version: 0xb, subversion: 0xc, comp id 0xc2b
conn idx: 0, peer feature: 0x0000000ff70179ff

# ble_disconn
Usage: ble_disconn <conn idx>
<conn idx>: index of connection to disconnect
#
# ble_disconn 0
# disconnected. conn idx: 0, conn_hdl: 0x1 reason 0x16
```

## 1.12.19. ble_list_sec_devs

该指令没有选项。

用于查询 flash 中存储的 bonded device 信息和当前正在 connect 的 device 信息。其中包括 dev idx、 id_addr、LTK 和 IRK 等信息。

如*图 1-36. ble_list_sec_devs 指令*所示，ble_list_sec_devs 执行后会打印提示。

图 1-36. ble_list_sec_devs 指令

```
# ble_list_sec_devs
======= dev idx 0 =========
-->   sec device cur_addr 80:0C:67:21:EF:9F
-->   sec device id_addr 80:0C:67:21:EF:9F
local key size 16, ltk(hex): 12d0157d8147eb7853f4212aadb37cca
peer key size 16, ltk(hex): 68a78d360c5208bcf1f46e4c4fe2c110
peer irk(hex): 4cc1178f4c11d8b79def464e279b1c66
======= dev idx 1 =========
-->   sec device cur_addr CC:89:67:45:23:01
-->   sec device id_addr CC:89:67:45:23:01
local key size 16, ltk(hex): 7ee66fd8e2eb316bee12ad376a0d5e96
peer key size 16, ltk(hex): d098c8f4d864b604f65757d7f864f5c6
peer irk(hex): a421c66a2af80b16e354bc8056f9fdd7
local csrk(hex): 192a8799f937f9db48e30ab20f324f93
peer csrk(hex): e1aa971a9fa7fdc099e6aabbf920222f
#
```

## 1.12.20. ble_remove_bond

- Usage: ble_remove_bond <dev idx>
- dev idx 需要从 ble_list_sec_devs 指令中获取。

该指令用于删除设备的 bond 信息，若该设备正处于连接状态，会先断开连线再删除 bond 信息，flash 中对应的内容也将删除。

如*图 1-37. ble_remove_bond 指令*所示，ble_remove_bond 执行后会打印提示。

图 1-37. ble_remove_bond 指令

```
# ble_list_sec_devs
======= dev idx 0 =========
-->   sec device cur_addr 80:0C:67:21:EF:9F
-->   sec device id_addr 80:0C:67:21:EF:9F
local key size 16, ltk(hex): 12d0157d8147eb7853f4212aadb37cca
peer key size 16, ltk(hex): 68a78d360c5208bcf1f46e4c4fe2c110
peer irk(hex): 4cc1178f4c11d8b79def464e279b1c66
#
# ble_remove_bond
Usage: ble_remove_bond <dev idx>
<dev idx>: device index in bond list
#
# ble_remove_bond 0
remove bond success
#
# ble_list_sec_devs
======= list empty =========
#
```

### 1.12.21. ble_set_auth

- Usage: ble_set_auth <bond> <mitm> <sc> <iocap>

该指令用于配置设备安全策略：配对完成后是否保存配对信息，是否支持中间人攻击保护，是否支持安全连接和 IO 能力等。

如果配置了 bond flag，设备配对成功后会保存 peer 的 LTK、IRK 和 CSRK 等信息至 flash；配置 mitm flag 表示支持中间人攻击保护，若对端也支持，可根据 IO 能力来选择不同的配对方式；配置 sc flag 表示设备支持安全连接，若对端也支持，可通过 ECDH 密钥交换算法来生成长期密钥；配置 iocap 可以选择在配对过程中使用的 IO 的能力，支持 display only, display yes no, keyboard only, no input no output, keyboard display 等方式。

如*图 1-38. ble_set_auth 指令*所示，ble_set_auth 执行后会打印提示。

图 1-38. ble_set_auth 指令

```
# ble_set_auth
Usage: ble_set_auth <bond> <mitm> <sc> <iocap>
<bond>: bonding flag for authentication
      0x00: no bonding
      0x01: bonding
<mitm>: mitm flag for authentication
      0x00: mitm protection not required
      0x01: mitm protection required
<sc>: secure connections flag for authention
      0x00: secure connections pairing is not supported
      0x00: secure connections pairing is supported
<iocap>: io capability to set
      0x00: display only
      0x01: display yes no
      0x02: keyboard only
      0x03: no input no output
      0x04: keyboard display
#
# ble_set_auth 1 0 0 2
ble set auth success.
```

### 1.12.22. ble_pair

- Usage: ble_pair <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于主动与指定连线的设备进行配对，配对操作用以生成可用于加密链接的密钥。

如*图 1-39. ble_pair 指令*所示，ble_pair 执行后会打印提示。

图 1-39. ble_pair 指令

```
# ble_pair
Usage: ble_pair <conn idx>
<conn idx>: index of the connection to pair
#
# ble_pair 0
# bond ind, key size 16, ltk: 0xbf528921c3f9e555e3b71972b0951ca7
rcv remote irk: 0x4cc1178f4c11d8b79def464e279b1c66
rcv remote identity addr: 0x80:0xc:0x67:0x21:0xef:0x9f, type 0
conn_idx 0 pairing success, level 0x1 ltk_present 1 sc 0
local key size 16, ltk(hex): 6d99cb37930a4a239034ac67dc32a7f9
peer key size 16, ltk(hex): bf528921c3f9e555e3b71972b0951ca7
peer irk(hex): 4cc1178f4c11d8b79def464e279b1c66
bond data ind: gatt_start_hdl 0, gatt_end_hdl 0, svc_chg_hdl 0, cli_info 1, cli_feat 0, srv_feat 0
```

### 1.12.23. ble_passkey

- Usage: ble_passkey <conn idx> <passkey>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于与指定连线的设备进行配对的过程中输入 passkey(6 位的数字)，需要与对端一致才

能配对成功。

如*图 1-40. ble_passkey 指令*所示，ble_passkey 执行后会打印提示。

**图 1-40. ble_passkey 指令**

```
# ble_set_auth 1 1 0 2
ble set auth success.
# ble_pair 0
# conn_idx 0 waiting for user to input key ......
ble_passkey
Usage: ble_passkey <conn idx> <passkey>
<conn idx>: index of connection to input passkey
<passkey>: passkey value to input, should be 6-digit value between 000000 and 999999
#
# ble_passkey 0 366279
input passkey0: 366279 passkey1: 0
# bond ind, key size 16, ltk: 0xe7b672e24a20a327567cc89d208c2f04
rcv remote irk: 0x4cc1178f4c11d8b79def464e279b1c66
rcv remote identity addr: 0x80:0xc:0x67:0x21:0xef:0x9f, type 0
conn_idx 0 pairing success, level 0x5 ltk_present 1 sc 0
local key size 16, ltk(hex): 9957c1d5710148fdf36cdbc7eb4cf8f3
peer key size 16, ltk(hex): e7b672e24a20a327567cc89d208c2f04
peer irk(hex): 4cc1178f4c11d8b79def464e279b1c66
bond data ind: gatt_start_hdl 0, gatt_end_hdl 0, svc_chg_hdl 0, cli_info 1, cli_feat 0, srv_feat 0
```

### 1.12.24. ble_encrypt

- Usage: ble_encrypt <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于对指定连线进行加密，如果链路已处于加密状态，会重新生成 encryption key。

如*图 1-41. ble_encrypt 指令*所示，ble_encrypt 执行后会打印提示。

**图 1-41. ble_encrypt 指令**

```
# ble_encrypt
Usage: ble_encrypt <conn idx>
<conn idx>: index of the connection to start encryption
#
# ble_encrypt 0
# conn_idx 0 encrypted, pairing_lvl 0x5 status 0x0
conn idx 0 ping timeout set status 0x0
```

### 1.12.25. ble_compare

- Usage: ble_compare <conn idx> <result>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于与指定连线的设备进行配对的过程中，判断两端生成的临时 key 是否相同。

如*图 1-42. ble_compare 指令*所示，ble_compare 执行后会打印提示。

图 1-42. ble_compare 指令

```
ble_conn 13
# ===> init conn starting idx 1, wl_used 0
===> init conn started idx 1, wl_used 0
connect success. conn idx:0, conn_hdl:0x1
===> init conn idle idx 1, wl_used 0 reason 0x0
le pkt size ind: conn idx 0, tx oct 251, tx time 2120, rx oct 251, rx time 2120
conn idx: 0, peer version: 0xb, subversion: 0xc, comp id 0xc2b
conn idx: 0, peer feature: 0x0000000ff70179ff
conn_idx 0 num val: 365294
waiting for user to compare......

# ble_compare
Usage: ble_compare <conn idx> <result>
<conn idx> index of connection
<result>: numeric comparison result, 0 for fail and 1 for success
#
# ble_compare 0 1
compare result: 1
# bond ind, key size 16, ltk: 0x1316d3d3bdb200f9bb006e9c9a663480
rcv remote irk: 0x9db73b59862a11c553732ca71f6e894
rcv remote identity addr: 0xab:0x89:0x67:0x45:0x23:0x1, type 0
bond ind csrk: e4 63 4c 41 7c 0d 04 57 fa c1 3e ca 38 8f 13 27
conn_idx 0 pairing success, level 0xd ltk_present 1 sc 1
local key size 16, ltk(hex): 1316d3d3bdb200f9bb006e9c9a663480
peer key size 16, ltk(hex): 1316d3d3bdb200f9bb006e9c9a663480
peer irk(hex): 9db73b59862a11c5530732ca71f6e894
local csrk(hex): 2e43fe4c2eda3d9ce2d5eedd8995d0dc
peer csrk(hex): e4634c417c0d0457fac13eca388f1327
```

### 1.12.26. ble_peer_feat

- Usage: ble_peer_feat <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于获取指定连线设备支持的 feature，每个 bit 对应的含义可参考 BLE Core Spec 的 FEATURE SUPPORT。

如*图 1-43. ble_peer_feat 指令*所示，ble_peer_feat 执行后会打印提示。

图 1-43. ble_peer_feat 指令

```
# ble_peer_feat
Usage: ble_peer_feat <conn idx>
<conn idx>: index of connection
#
# ble_peer_feat 0
# conn idx: 0, peer feature: 0x0000000ff70179ff
```

### 1.12.27. ble_peer_ver

- Usage: ble_peer_ver <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于获取指定连线设备的版本信息，包括蓝牙版本信息(0xb:BT5.2)，子版本信息，

company identifier(GigaDevice：0x0C2B) 。

如*图 1-44. ble_peer_ver 指令*所示，ble_peer_ver 执行后会打印提示。

**图 1-44. ble_peer_ver 指令**

```
# ble_peer_ver
Usage: ble_peer_ver <conn idx>
<conn idx>: index of connection
#
# ble_peer_ver 0
# conn idx: 0, peer version: 0xb, subversion: 0xc, comp id 0xc2b
```

### 1.12.28. ble_get_rssi

- Usage: ble_get_rssi <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用来获取指定连线上收到的对端设备发送的最新报文的 rssi。

如*图 1-45. ble_get_rssi 指令*所示，ble_get_rssi 执行后会打印提示。

**图 1-45. ble_get_rssi 指令**

```
# ble_get_rssi
Usage: ble_get_rssi <conn idx>
<conn idx>: index of connection
#
# ble_get_rssi 0
# conn idx 0 rssi: -42
ble_get_rssi 0
# conn idx 0 rssi: -55
```

### 1.12.29. ble_param_update

- Usage: ble_param_update <conn idx> <interval> <latency> <supv tout> <ce len>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于更新指定连线的 connection interval, latency, supervision timeout 等参数。

如*图 1-46. ble_param_update 指令*所示，ble_param_update 执行后会打印提示。

**图 1-46. ble_param_update 指令**

```
# ble_param_update
Usage: ble_param_update <conn idx> <interval> <latency> <supv tout> <ce len>
<conn idx>: index of connection
<interval>: connection interval in unit of 1.25ms, range from 0x0006 to 0x0C80 in hex value
<latency>: connection latency to update in hex value
<supv tout>: supervision timeout in unit of 10ms, range from 0x000A to 0x0C80 in hex value
<ce len>: connection event length in unit of 0.625 ms in hex value
#
# ble_param_update 0 6 0 a 0
# conn idx 0, param update ind: interval 6, latency 0, sup to 10
conn idx 0, param update result status: 0x0
```

### 1.12.30. ble_set_phy

- Usage: ble_set_phy <conn idx> <tx phy> <rx phy> <phy opt>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令仅在 msdk_ffd configuration 下可以使用。

该指令用于设置在指定连线上使用的 tx/rx phy，其中设置的 tx/rx phy 参数为 0，表示所有都支持，否则如*图 1-47. ble_set_phy 指令*各个 bit 指示。

如*图 1-47. ble_set_phy 指令*所示，ble_set_phy 执行后会打印提示。

**图 1-47. ble_set_phy 指令**

```
# ble_set_phy
Usage: ble_set_phy <conn idx> <tx phy> <rx phy> <phy opt>
<conn idx>: index of connection
<tx phy>: transmit phy to set
      bit 0: 1M phy, bit 1: 2M phy, bit 2: coded phy
<rx phy>: receive phy to set
      bit 0: 1M phy, bit 1: 2M phy, bit 2: coded phy
<phy opt>: phy options for coded phy
      0x00: no prefer coding
      0x01: prefer S=2 coding be used
      0x02: prefer S=8 coding be used
# ble_set_phy 0 2 2 0
# le phy ind conn idx 0: tx phy 0x2, rx phy 0x2
conn idx 0 le phy set status 0x0
```

### 1.12.31. ble_get_phy

- Usage: ble_get_phy <conn idx>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于获取指定连线当前使用的 tx/rx phy。

该指令仅在 msdk_ffd configuration 下可以使用。

如*图 1-48. ble_get_phy 指令*所示，ble_get_phy 执行后会打印提示，其中 0x1：1M；0x2：2M；0x3：coded。

**图 1-48. ble_get_phy 指令**

```
# ble_get_phy
Usage: ble_get_phy <conn idx>
<conn idx>: index of connection
#
# ble_get_phy 0
# le phy ind conn idx 0: tx phy 0x1, rx phy 0x1
conn idx 0 le phy get status 0x0
```

### 1.12.32. ble_set_pkt_size

- Usage: ble_set_pkt_size <conn idx> <tx oct> <tx time>
- conn idx 在设备建立 connection success 时会打印，可从 ble_conn 日志中获取。

该指令用于设置指定连线上发送 PDU 时可使用的最大字节数及时间。

该指令仅在 msdk_ffd configuration 下可以使用。

如*图 1-49. ble_set_pkt_size 指令*所示，ble_set_pkt_size 执行后会打印提示。

**图 1-49. ble_set_pkt_size 指令**

```
# ble_set_pkt_size
Usage: ble_set_pkt_size <conn idx> <tx oct> <tx time>
<conn idx>: index of connection
<tx oct>: preferred maximum number of payload octets in a single data PDU, Range 27 to 251
<tx time>: preferred maximum number of microseconds used to transmit a single data PDU, Range 328 to 17040
#
# ble_set_pkt_size 0 27 328
# conn idx 0, packet size set status 0x0
le pkt size ind: conn idx 0, tx oct 27, tx time 328, rx oct 251, rx time 17040
```

# 2. 版本历史

表 2-1. 版本历史

| 版本号. | 说明 | 日期 |
| --- | --- | --- |
| 1.0 | 首次发布 | 2023 年 10 月 17 日 |

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.