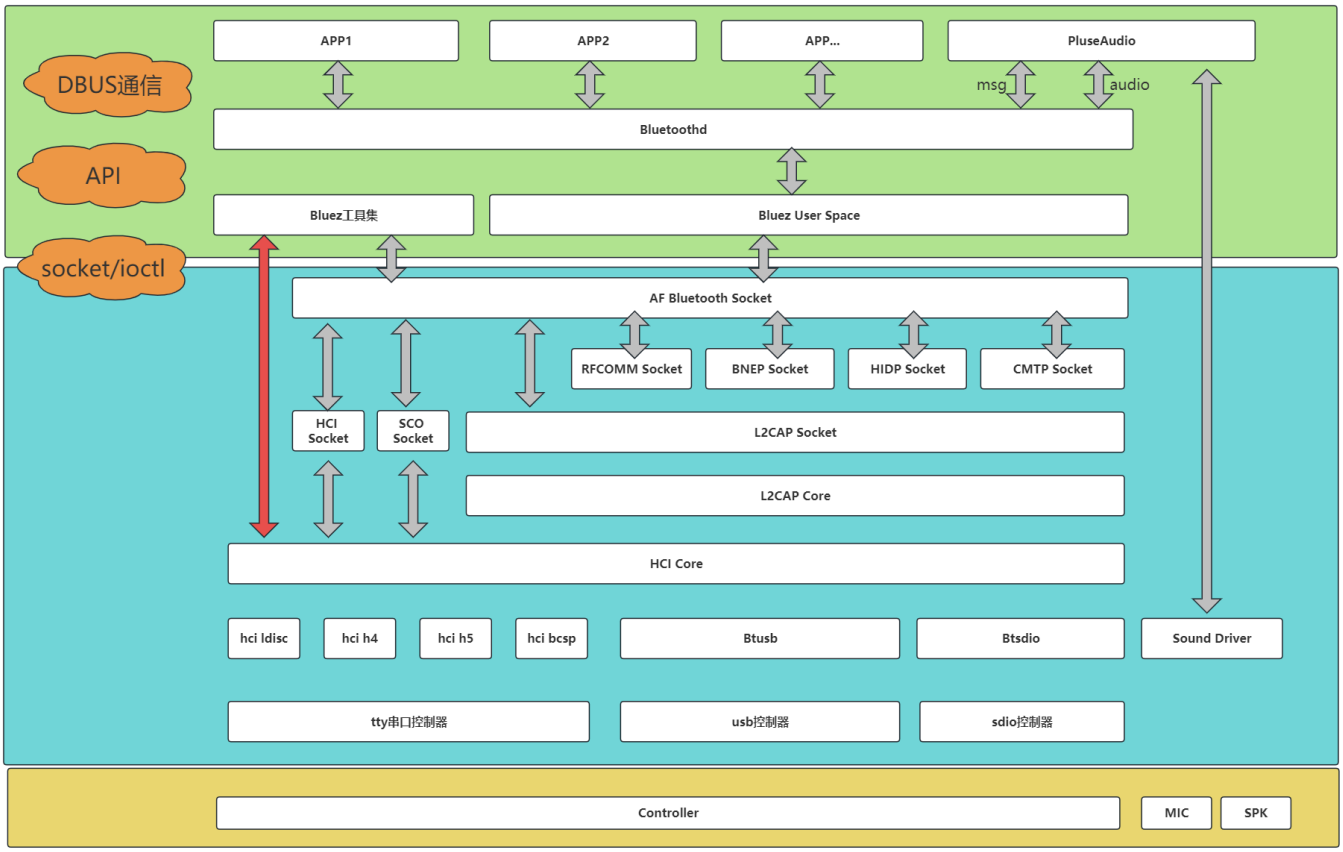
Bluez介绍

版本	日期	作者	变更表述
1.0	2023/08/03	于忠军	文档创建

一. Bluez架构



1.User Space层：

APP：上层应用程序

Pluseaudio/pipewire：A2DP的组件

Bluetoothd: 蓝牙守护进程

Bluez: 包括Bluez tool跟Bluez lib

2.Kernel Space:

内核代码包含以下几部分

driver/bluetooth

net/bluetooth

include/net/bluetooth

二. Bluez Tool

1. hciattach

语法：

> hciattach [OPTIONS] <tty> <type|id> [speed] [flow] [sleep] [bdaddr]
> hciattach [选项] <设备> <类型> [速率] [流控制]

这里是参数的说明：

- 设备：这是串行设备或蓝牙HCI设备的路径。例如，对于基于UART（串行）接口的蓝牙控制器，它可能是/dev/ttyS0，而对于内置的HCI设备，它可能是/dev/hci0。
- 类型：这指定您要附加的蓝牙HCI设备的类型。该值取决于您使用的硬件或通信接口类型。常见类型有uart用于UART（串行）接口，bcm43xx用于Broadcom芯片组的控制器，ath3k用于Atheros AR3011芯片组的控制器等。可用的类型可能因系统配置和驱动程序而异。
- 速率：（可选）这指定HCI设备进行通信的速率（波特率）。如果未提供，它可能默认为特定类型HCI设备通常使用的速率。
- 流控制：（可选）这指定要使用的硬件流控制，可能是none、rtscts或xoob。

可以用命令行来查询下怎么使用

> man hciattach

2. btattach

语法

```
btattach [options] <device> <type> [speed] [flow]
```

这里是参数的说明：

- device：这是串行设备或蓝牙HCI设备的路径。例如，对于基于UART（串行）接口的蓝牙控制器，它可能是/dev/ttyS0，而对于内置的HCI设备，它可能是/dev/hci0。
- type：这指定您要附加的蓝牙HCI设备的类型。该值取决于您使用的硬件或通信接口类型。常见类型有uart用于UART（串行）接口，bcm用于Broadcom芯片组的控制器，ath3k用于Atheros AR3011芯片组的控制器等。可用的类型可能因系统配置和驱动程序而异。
- speed：（可选）这指定HCI设备进行通信的速率（波特率）。如果未提供，它可能默认为特定类型HCI设备通常使用的速率。
- flow：（可选）这指定要使用的硬件流控制，可能是none、rtscts或xoob。

要使用btattach，您需要超级用户（root）权限，因此您可能需要使用sudo来运行它。
例如，以下命令将附加一个基于UART接口的蓝牙控制器：

```
sudo btattach /dev/ttyS0 –P bcm
```

可以用命令行来查询下怎么使用

```
man hciattach
```

3. hciconfig

hciconfig 是一个命令行工具，用于配置和管理蓝牙设备的设置。它通常用于 Linux 系统上，特别是使用 BlueZ 蓝牙协议栈的系统。需要注意的是，截至我的最后更新日期（2021年9月），BlueZ 可能已经更新或改变，因此一些细节可能有所不同。建议查阅您当前系统的文档或运行 man hciconfig 命令来查看最新的用法和选项。
以下是 hciconfig 的一般用法：

```
hciconfig [hciX] [command]
```

- hciX: 这是蓝牙 HCI 设备的标识符，其中 X 是一个数字，例如 hci0、hci1 等。如果不指定 hciX，hciconfig 默认将操作的是第一个蓝牙 HCI 设备，即 hci0。
- command: 这是要执行的命令选项，用于配置或管理蓝牙设备。

一些常见的 hciconfig 命令可以用man hciconfig查看

4. bluetoothctl

bluetoothctl 是 Linux 上用于与蓝牙设备交互的命令行工具，它是 BlueZ 蓝牙协议栈的一部分。通过 bluetoothctl，您可以扫描蓝牙设备、连接到设备、管理配对关系以及配置蓝牙适配器等。下面是 bluetoothctl 的一般用法：

```
Menu main:
Available commands:
-------------------
advertise                                 Advertise Options Submenu
monitor                                   Advertisement Monitor Options Submenu
scan                                      Scan Options Submenu
gatt                                      Generic Attribute Submenu
admin                                     Admin Policy Submenu
player                                    Media Player Submenu
endpoint                                  Media Endpoint Submenu
transport                                 Media Transport Submenu
list                                      List available controllers
show [ctrl]                               Controller information
select <ctrl>                             Select default controller
devices [Paired/Bonded/Trusted/Connected] List available devices, with an optional property as the filter
system-alias <name>                       Set controller alias
reset-alias                               Reset controller alias
power <on/off>                            Set controller power
pairable <on/off>                         Set controller pairable mode
discoverable <on/off>                     Set controller discoverable mode
discoverable-timeout [value]              Set discoverable timeout
agent <on/off/capability>                 Enable/disable agent with given capability
default-agent                             Set agent as the default one
advertise <on/off/type>                   Enable/disable advertising with given type
set-alias <alias>                         Set device alias
scan <on/off/bredr/le>                    Scan for devices
info [dev]                                Device information
pair [dev]                                Pair with device
cancel-pairing [dev]                      Cancel pairing with device
trust [dev]                               Trust device
untrust [dev]                             Untrust device
block [dev]                               Block device
unblock [dev]                             Unblock device
remove <dev>                              Remove device
connect <dev>                             Connect device
disconnect [dev]                          Disconnect device
menu <name>                               Select submenu
version                                   Display version
quit                                      Quit program
exit                                      Quit program
help                                      Display help about this program
export                                    Print environment variables
```

5. btmon

tmon 是一个 Linux 命令行工具，用于监视蓝牙数据流量。它是 BlueZ 蓝牙协议栈的一部分，可用于调试和分析蓝牙设备之间的通信，就是类似于btsnoop工具，抓取host跟controller之前的通信数据。

可以用命令行来查询下怎么使用

> man hciattach

其中一个比较好用的功能是btmon -E ip地址，可以把hci数据injection到ellisys中

6. btmgmt

btmgmt 是 Linux 上的一个命令行工具，用于管理蓝牙控制器和蓝牙适配器的设置。它是 BlueZ 蓝牙协议栈的一部分。btmgmt有以下用法

```
Menu main:
Available commands:
-------------------
select <index>                                Select a different index
version                                       Get the MGMT Version
commands                                      List supported commands
config                                        Show configuration info
info                                          Show controller info
extinfo                                       Show extended controller info
auto-power                                    Power all available features
power <on/off>                                Toggle powered state
discov <yes/no/limited> [timeout]             Toggle discoverable state
connectable <on/off>                          Toggle connectable state
fast-conn <on/off>                            Toggle fast connectable state
bondable <on/off>                             Toggle bondable state
pairable <on/off>                             Toggle bondable state
linksec <on/off>                              Toggle link level security
ssp <on/off>                                  Toggle SSP mode
sc <on/off/only>                              Toogle SC support
hs <on/off>                                   Toggle HS support
le <on/off>                                   Toggle LE support
advertising <on/off>                          Toggle LE advertising
bredr <on/off>                                Toggle BR/EDR support
privacy <on/off>                              Toggle privacy support
class <major> <minor>                         Set device major/minor class
disconnect [-t type] <remote address>         Disconnect device
con                                           List connections
find [-l|-b] [-L]                             Discover nearby devices
find-service [-u UUID] [-r RSSI_Threshold] [-l|-b] Discover nearby service
stop-find [-l|-b]                             Stop discovery
name <name> [shortname]                       Set local name
pair [-c cap] [-t type] <remote address>      Pair with a remote device
cancelpair [-t type] <remote address>         Cancel pairing
unpair [-t type] <remote address>             Unpair device
keys                                          Load Link Keys
ltks                                          Load Long Term Keys
irks [--local <index>] [--file <file path>]   Load Identity Resolving Keys
block [-t type] <remote address>              Block Device
unblock [-t type] <remote address>            Unblock Device
add-uuid <UUID> <service class hint>          Add UUID
rm-uuid <UUID>                                Remove UUID
clr-uuids                                     Clear UUIDs
```

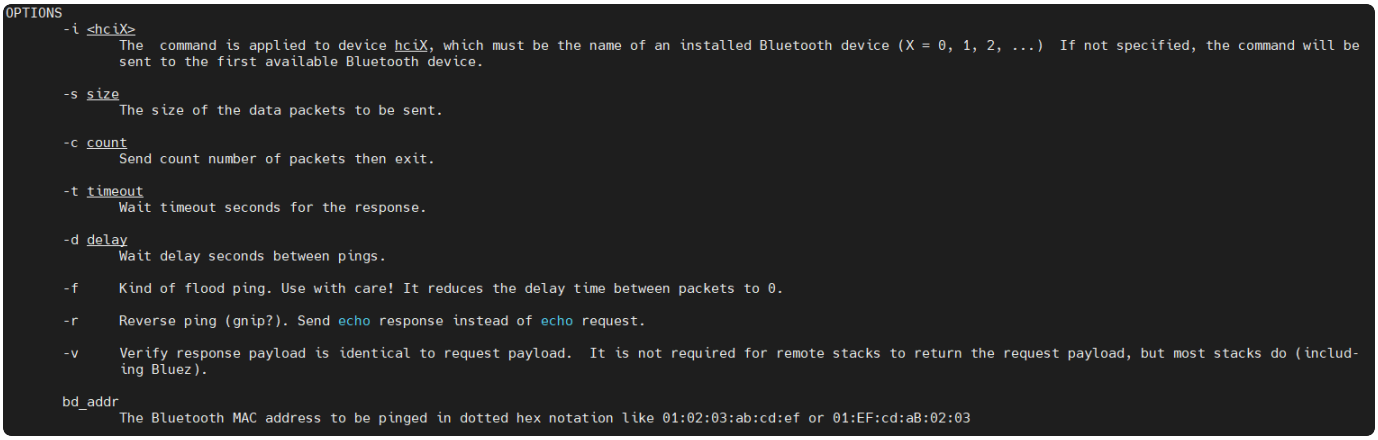
```
local-oob                                     Local OOB data
remote-oob [-t <addr_type>] [-r <rand192>] [-h <hash192>] [-R <rand256>] [-H <hash256>] <addr> Remote OOB data
did <source>:<vendor>:<product>:<version>     Set Device ID
static-addr <address>                         Set static address
public-addr <address>                         Set public address
ext-config <on/off>                           External configuration
debug-keys <on/off>                           Toogle debug keys
conn-info [-t type] <remote address>          Get connection information
io-cap <cap>                                  Set IO Capability
scan-params <interval> <window>               Set Scan Parameters
get-clock [address]                           Get Clock Information
add-device [-a action] [-t type] <address>    Add Device
del-device [-t type] <address>                Remove Device
clr-devices                                   Clear Devices
bredr-oob                                     Local OOB data (BR/EDR)
le-oob                                        Local OOB data (LE)
advinfo                                       Show advertising features
advsize [options] <instance_id>               Show advertising size info
add-adv [options] <instance_id>               Add advertising instance
rm-adv <instance_id>                          Remove advertising instance
clr-adv                                       Clear advertising instances
appearance <appearance>                       Set appearance
phy [LE1MTX] [LE1MRX] [LE2MTX] [LE2MRX] [LECODEDTX] [LECODEDRX] [BR1M1SLOT] [BR1M3SLOT] [BR1M5SLOT][EDR2M1SLOT]
OT] Get/Set PHY Configuration
version                                       Display version
quit                                          Quit program
exit                                          Quit program
help                                          Display help about this program
export                                        Print environment variables
```

7. l2ping

l2ping 是一个 Linux 命令行工具，用于测试蓝牙设备之间的 L2CAP 连接。L2CAP（Logical Link Control and Adaptation Protocol）是蓝牙协议栈中用于提供数据传输的层

语法

> l2ping [OPTIONS] bd_addr



其他工具

bluez还有很多特定的工具，比如gatttool/sdptool/obexctl..

三. Bluez qcom init流程(user space download fw)

1. hciattach /dev/ttyS1 qca –t120 3000000 flow 发生的行为

步骤1：参数解析

在hciattach.c的main函数中

C | 复制代码

```c
for (n = 0; optind < argc; n++, optind++) {
    char *opt;

    opt = argv[optind];

    switch(n) {
            case 0:
                dev[0] = 0;
                if (!strchr(opt, '/'))
                strcpy(dev, "/dev/");

                if (strlen(opt) > PATH_MAX - (strlen(dev) + 1)) {
                fprintf(stderr, "Invalid serial device\n");
                exit(1);
            }

                strcat(dev, opt);
                break;

            case 1:
                if (strchr(argv[optind], ',')) {
                int m_id, p_id;
                sscanf(argv[optind], "%x,%x", &m_id, &p_id);
                u = get_by_id(m_id, p_id);
            } else {
                u = get_by_type(opt); // 通过字符串跟uart数组匹配，里面有qca相关的内容
            }

                if (!u) {
                fprintf(stderr, "Unknown device type or id\n");
                exit(1);
            }

                break;

            case 2:
                u->speed = atoi(argv[optind]);
                break;

            case 3:
                if (!strcmp("flow", argv[optind]))
                u->flags |=  FLOW_CTL;
                else
                u->flags &= ~FLOW_CTL;
```

```
                break;

            case 4:
                if (!strcmp("sleep", argv[optind]))
                u->pm = ENABLE_PM;
                else
                u->pm = DISABLE_PM;
                break;

            case 5:
                u->bdaddr = argv[optind];
                break;
        }
    }
```

步骤2：uart配置

C | 复制代码

```c
struct termios ti;
    int fd, i;
    unsigned long flags = 0;

    if (raw)
        flags |= 1 << HCI_UART_RAW_DEVICE;

    if (u->flags & AMP_DEV)
        flags |= 1 << HCI_UART_CREATE_AMP;

    if (!strncmp(u->type, "qca", 3))
        flags |= 1 << HCI_UART_RESET_ON_INIT;

    fd = open(dev, O_RDWR | O_NOCTTY);
    if (fd < 0) {
        perror("Can't open serial port");
        return -1;
    }

    tcflush(fd, TCIOFLUSH);

    if (tcgetattr(fd, &ti) < 0) {
        perror("Can't get port settings");
        goto fail;
    }

    cfmakeraw(&ti);

    ti.c_cflag |= CLOCAL;
    if (u->flags & FLOW_CTL)
        ti.c_cflag |= CRTSCTS;
    else
        ti.c_cflag &= ~CRTSCTS;

    if (tcsetattr(fd, TCSANOW, &ti) < 0) {
        perror("Can't set port settings");
        goto fail;
    }

    /* Set initial baudrate */
    if (set_speed(fd, &ti, u->init_speed) < 0) {
        perror("Can't set initial baud rate");
        goto fail;
    }

```

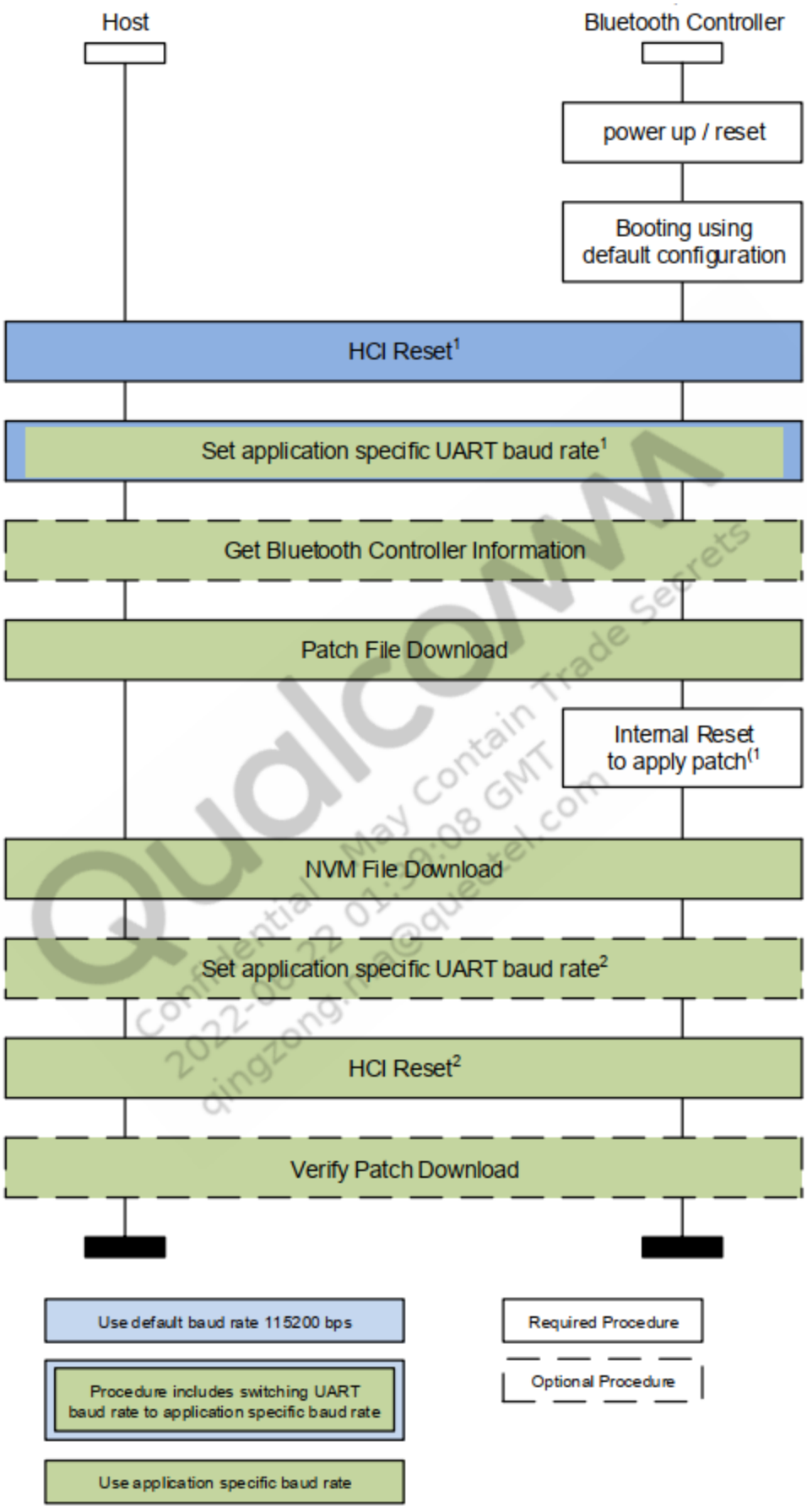
```
    tcflush(fd, TCIOFLUSH);

    if (send_break) {
        tcsendbreak(fd, 0);
        usleep(500000);
    }
```

打开串口，设置流控，设置波特率

步骤3：tlv/nvm下载

init_uart函数中if (u->init && u->init(fd, u, &ti) < 0)触发qca_soc_init 调用

高通的init流程比较简单，流程如下：


Host
Bluetooth Controller
power up / reset
Booting using default configuration
HCI Reset[1]
Set application specific UART baud rate[1]
Get Bluetooth Controller Information
Patch File Download
Internal Reset to apply patch[1]
NVM File Download
Set application specific UART baud rate[2]
HCI Reset[2]
Verify Patch Download
Use default baud rate 115200 bps
Procedure includes switching UART baud rate to application specific baud rate
Use application specific baud rate
Required Procedure
Optional Procedure

代码实现如下：

C | 复制代码

```c
int qca_soc_init(int fd, int speed, char *bdaddr)
{
    int err = -1;
    int local_baud_rate = 0;
    int controller_baud_rate = 0;

    vnd_userial.fd = fd;

#ifdef _PLATFORM_MDM_
    /* Vote for UART CLK prior to FW download */
    err = ioctl(fd, USERIAL_OP_CLK_ON);
    if (err < 0)
    {
        fprintf(stderr, "%s: Failed to vote UART CLK ON\n", __func__);
        return -1;
    }
#endif
    /* Get Rome version information */
    if ((err = rome_patch_ver_req(fd)) < 0)
    {
        fprintf(stderr, "%s: Fail to get Rome Version (0x%x)\n", __FUNCTION__, err);
        goto error;
    }

    fprintf(stderr, "%s: Rome Version (0x%08x) g_soc_id(0x%x)\n", __FUNCTION__, rome_ver, g_soc_id);

    switch (g_soc_id)
    {
    case SOC_VER_QCA6695:
        rampatch_file_path = QCA6595_RAMPATCH_TLV_UART_2_0_PATH;
        nvm_file_path = QCA6595_NVM_TLV_UART_2_0_PATH;
        break;
    case SOC_VER_QCA6696:
        rampatch_file_path = QCA6696_RAMPATCH_TLV_UART_2_0_PATH;
        nvm_file_path = QCA6696_NVM_TLV_UART_2_0_PATH;
        break;
    case SOC_VER_QCA206X:
        rampatch_file_path = QCA206X_RAMPATCH_TLV_UART_2_0_PATH;
        nvm_file_path = QCA206X_NVM_TLV_UART_2_0_PATH;
        break;
    case SOC_VER_QCA206X_G:
        rampatch_file_path = QCA206X_RAMPATCH_TLV_UART_2_1_PATH;
        nvm_file_path = QCA206X_NVM_TLV_UART_2_1_G_PATH;
```

```
44          break;
45      case SCO_VER_QCA6698:
46          rampatch_file_path = QCA6698_RAMPATCH_TLV_UART_2_1_PATH;
47          nvm_file_path = QCA6698_NVM_TLV_UART_2_1_PATH;
48          break;
49      case SOC_VER_QCA9377:
50          rampatch_file_path = QCA9377_RAMPATCH_TLV_1_0_1_PATH;
51          nvm_file_path = QCA9377_NVM_TLV_1_0_1_PATH;
52          break;
53      case SOC_VER_QCC207X:
54          rampatch_file_path = QCC207X_RAMPATCH_TLV_UART_2_0_PATH;
55          nvm_file_path = QCC207X_NVM_TLV_UART_2_0_PATH;
56          break;
57      default:
58          fprintf(stderr, "Detected unknown ROME soc id:0x%x", g_soc_id);
59          goto error;
60      }
61
62      /* Check if user requested for 115200 kbps */
63      if (speed == 115200)
64      {
65          local_baud_rate = USERIAL_BAUD_115200;
66          controller_baud_rate = BAUDRATE_115200;
67      }
68      else
69      {
70          /* Change only if baud rate requested is valid or not */
71          is_speed_valid(speed, &local_baud_rate, &controller_baud_rate);
72          if (local_baud_rate < 0 || controller_baud_rate < 0)
73          {
74              err = -1;
75              goto error;
76          }
77          err = rome_set_baudrate_req(fd, local_baud_rate, controller_baud_
    rate);
78          if (err < 0)
79          {
80              fprintf(stderr, "%s: Baud rate change failed!\n", __FUNCTION_
    _);
81              goto error;
82          }
83      }
84      fprintf(stderr, "%s: Baud rate changed successfully \n", __FUNCTION__
    );
85
86      /* Donwload TLV files (rampatch, NVM) */
87      err = rome_download_tlv_file(fd);
88      if (err < 0)
```

```
    {
        fprintf(stderr, "%s: Download TLV file failed!\n", __FUNCTION__);
        goto error;
    }
    fprintf(stderr, "%s: Download TLV file successfully \n", __FUNCTION__
);

    /*
     * Overriding the baud rate value in NVM file with the user
     * requested baud rate, since default baud rate in NVM file is 3M.
     */
    err = rome_set_baudrate_req(fd, local_baud_rate, controller_baud_rate
);
    if (err < 0)
    {
        fprintf(stderr, "%s: Baud rate change failed!\n", __FUNCTION__);
        goto error;
    }
    /* Perform HCI reset here*/
    err = rome_hci_reset_req(fd, local_baud_rate);
    if (err < 0)
    {
        fprintf(stderr, "HCI Reset Failed !!!\n");
        goto error;
    }
    fprintf(stderr, "HCI Reset is done\n");
error:
#ifdef _PLATFORM_MDM_
    /* Vote UART CLK OFF post to FW download */
    err = ioctl(fd, USERIAL_OP_CLK_OFF);
    if (err < 0)
        fprintf(stderr, "%s: Failed to vote UART CLK OFF!!!\n", __func__
);
#endif

    return err;
}
```

步骤4：Kernel加载前置行为

加载hci_uart.ko

module_init(hci_uart_init);

C | 复制代码

```c
static int __init hci_uart_init(void)
{
    static struct tty_ldisc_ops hci_uart_ldisc;
    int err;

    BT_INFO("HCI UART driver ver %s", VERSION);

    /* Register the tty discipline */

    memset(&hci_uart_ldisc, 0, sizeof(hci_uart_ldisc));
    hci_uart_ldisc.magic        = TTY_LDISC_MAGIC;
    hci_uart_ldisc.name     = "n_hci";
    hci_uart_ldisc.open     = hci_uart_tty_open;
    hci_uart_ldisc.close        = hci_uart_tty_close;
    hci_uart_ldisc.read     = hci_uart_tty_read;
    hci_uart_ldisc.write        = hci_uart_tty_write;
    hci_uart_ldisc.ioctl        = hci_uart_tty_ioctl;
    hci_uart_ldisc.poll     = hci_uart_tty_poll;
    hci_uart_ldisc.receive_buf  = hci_uart_tty_receive;
    hci_uart_ldisc.write_wakeup = hci_uart_tty_wakeup;
    hci_uart_ldisc.owner        = THIS_MODULE;

    err = tty_register_ldisc(N_HCI, &hci_uart_ldisc);
    if (err) {
        BT_ERR("HCI line discipline registration failed. (%d)", err);
        return err;
    }

#ifdef CONFIG_BT_HCIUART_H4
    h4_init();
#endif
#ifdef CONFIG_BT_HCIUART_BCSP
    bcsp_init();
#endif
#ifdef CONFIG_BT_HCIUART_LL
    ll_init();
#endif
#ifdef CONFIG_BT_HCIUART_ATH3K
    ath_init();
#endif
#ifdef CONFIG_BT_HCIUART_3WIRE
    h5_init();
#endif
#ifdef CONFIG_BT_HCIUART_INTEL
    intel_init();
```

```
#endif
#ifdef CONFIG_BT_HCIUART_BCM
    bcm_init();
#endif
#ifdef CONFIG_BT_HCIUART_QCA
    qca_init();
#endif
#ifdef CONFIG_BT_HCIUART_AG6XX
    ag6xx_init();
#endif
#ifdef CONFIG_BT_HCIUART_MRVL
    mrvl_init();
#endif

    return 0;
}
```

这块本身是hci_uart.ko,另外，里面有很多vendor也是一个ko

NOTED：这些也不一定是ko，有可能build in kernel

步骤4：kernel通信 – 1/3

C | 复制代码

```
i = N_HCI;
if (ioctl(fd, TIOCSETD, &i) < 0) {
    perror("Can't set line discipline");
    goto fail;
}
```

步骤5：kernel通信 – 2/3

C | 复制代码

```
if (flags && ioctl(fd, HCIUARTSETFLAGS, flags) < 0) {
    perror("Can't set UART flags");
    goto fail;
}
```

直接调用到kernel的hci_uart_tty_ioctl函数case HCIUARTSETFLAGS

```c
case HCIUARTSETFLAGS:
    if (test_bit(HCI_UART_PROTO_SET, &hu->flags))
        err = -EBUSY;
    else
        err = hci_uart_set_flags(hu, arg);
    break;
```

这个就是bluez tool hciattach下发的flag，qca的有：HCI_UART_RESET_ON_INIT，我们记住hu->hdev_flags是HCI_UART_RESET_ON_INIT,后面可能用到

步骤6：kernel通信 – 3/3

```c
if (ioctl(fd, HCIUARTSETPROTO, u->proto) < 0) {
    perror("Can't set device");
    goto fail;
}
```

可以看到u->proto是H4

```c
{ "qca", 0x0000, 0x0000, HCI_UART_H4, 115200, 3000000,
                FLOW_CTL, DISABLE_PM, NULL, qca, NULL },
```

直接调用到kernel的hci_uart_tty_ioctl函数case HCIUARTSETPROTO

```c
case HCIUARTSETPROTO:
    if (!test_and_set_bit(HCI_UART_PROTO_SET, &hu->flags)) {
        err = hci_uart_set_proto(hu, arg);
        if (err)
            clear_bit(HCI_UART_PROTO_SET, &hu->flags);
    } else
        err = -EBUSY;
    break;
```

```c
static int hci_uart_set_proto(struct hci_uart *hu, int id)
{
    const struct hci_uart_proto *p;
    int err;

    p = hci_uart_get_proto(id);
    if (!p)
        return -EPROTONOSUPPORT;

    hu->proto = p;

    err = hci_uart_register_dev(hu);
    if (err) {
        return err;
    }

    set_bit(HCI_UART_PROTO_READY, &hu->flags);
    return 0;
}
```

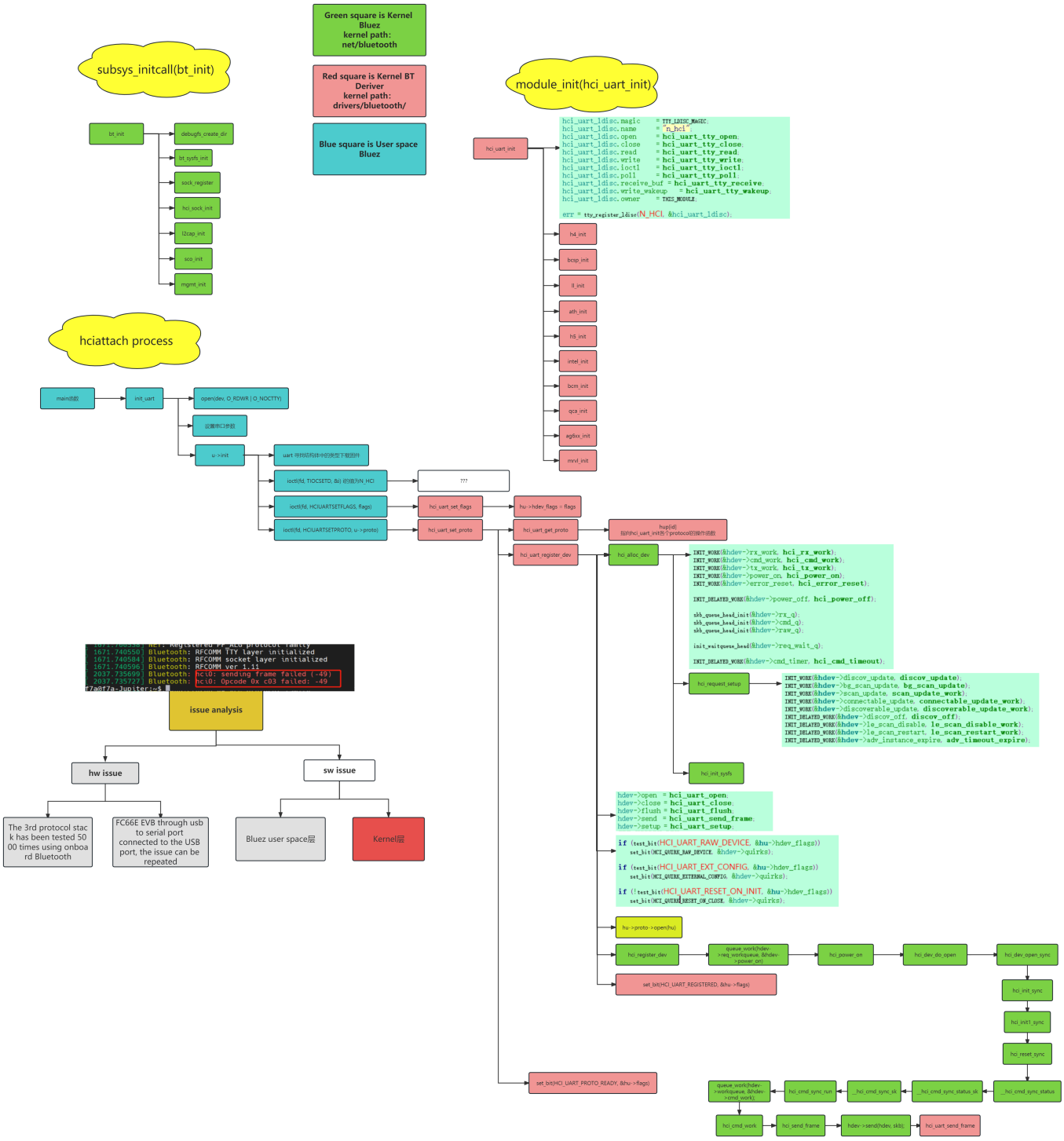
其中hci_uart_get_proto 拿到的struct是

```c
static const struct hci_uart_proto h4p = {
    .id     = HCI_UART_H4,
    .name       = "H4",
    .open       = h4_open,
    .close      = h4_close,
    .recv       = h4_recv,
    .enqueue    = h4_enqueue,
    .dequeue    = h4_dequeue,
    .flush      = h4_flush,
};
```

device的操作

```c
hdev->open  = hci_uart_open;
hdev->close = hci_uart_close;
hdev->flush = hci_uart_flush;
hdev->send  = hci_uart_send_frame;
hdev->setup = hci_uart_setup;
SET_HCIDEV_DEV(hdev, hu->tty->dev);
```

流程太多，就不一一介绍函数了，直接画出流程图



2. hciconfig hci0 up 发生的行为

步骤1：Kernel前置行为

C | 复制代码

```c
static int __init bt_init(void)
{
    int err;

    sock_skb_cb_check_size(sizeof(struct bt_skb_cb));

    BT_INFO("Core ver %s", VERSION);

    err = bt_selftest();
    if (err < 0)
        return err;

    bt_debugfs = debugfs_create_dir("bluetooth", NULL);

    bt_leds_init();

    err = bt_sysfs_init();
    if (err < 0)
        return err;

    err = sock_register(&bt_sock_family_ops);
    if (err)
        goto cleanup_sysfs;

    BT_INFO("HCI device and connection manager initialized");

    err = hci_sock_init();
    if (err)
        goto unregister_socket;

    err = l2cap_init();
    if (err)
        goto cleanup_socket;

    err = sco_init();
    if (err)
        goto cleanup_cap;

    err = mgmt_init();
    if (err)
        goto cleanup_sco;

    return 0;

cleanup_sco:
```

```
    sco_exit();
cleanup_cap:
    l2cap_exit();
cleanup_socket:
    hci_sock_cleanup();
unregister_socket:
    sock_unregister(PF_BLUETOOTH);
cleanup_sysfs:
    bt_sysfs_cleanup();
    return err;
}
```

这个函数是系统子模块加载

```
subsys_initcall(bt_init);
```

里面会创建hci/l2cap/sco的socket

hciconfig –a相关的hci动作操作在

```
static const struct proto_ops hci_sock_ops = {
    .family     = PF_BLUETOOTH,
    .owner      = THIS_MODULE,
    .release    = hci_sock_release,
    .bind       = hci_sock_bind,
    .getname    = hci_sock_getname,
    .sendmsg    = hci_sock_sendmsg,
    .recvmsg    = hci_sock_recvmsg,
    .ioctl      = hci_sock_ioctl,
    .poll       = datagram_poll,
    .listen     = sock_no_listen,
    .shutdown   = sock_no_shutdown,
    .setsockopt = hci_sock_setsockopt,
    .getsockopt = hci_sock_getsockopt,
    .connect    = sock_no_connect,
    .socketpair = sock_no_socketpair,
    .accept     = sock_no_accept,
    .mmap       = sock_no_mmap
};

```

步骤2：User Space行为

直接执行hciconfig.c中的main函数，

首先打开hci socket

```c
socket(AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI)
```

然后up执行cmd_up

```c
static void cmd_up(int ctl, int hdev, char *opt)
{
    /* Start HCI device */
    if (ioctl(ctl, HCIDEVUP, hdev) < 0) {
        if (errno == EALREADY)
            return;
        fprintf(stderr, "Can't init device hci%d: %s (%d)\n",
                        hdev, strerror(errno), errno);
        exit(1);
    }
}
```

步骤3：Kernel Space前置行为

```c
case HCIDEVUP:
    if (!capable(CAP_NET_ADMIN))
        return -EPERM;
    return hci_dev_open(arg);
```

3. hciconfig hci0 down 发生的行为

步骤1：User Space行为

hciconfig hci0 down执行cmd_down

```c
static void cmd_down(int ctl, int hdev, char *opt)
{
    /* Stop HCI device */
    if (ioctl(ctl, HCIDEVDOWN, hdev) < 0) {
        fprintf(stderr, "Can't down device hci%d: %s (%d)\n",
                        hdev, strerror(errno), errno);
        exit(1);
    }
}
```

步骤2：Kernel Space前置行为

```c
case HCIDEVDOWN:
    if (!capable(CAP_NET_ADMIN))
        return -EPERM;
    return hci_dev_close(arg);
```

执行kernel行为

```c
hci_dev_do_close
```

4. hciconfig –a 发生的行为

步骤1. User Space行为

```c
if (ioctl(ctl, HCIGETDEVINFO, (void *) &di)) {
    perror("Can't get device info");
    exit(1);
}
```

步骤2：Kernel Space行为

C | 复制代码

```c
case HCIGETDEVINFO:
    return hci_get_dev_info(argp);
```

C | 复制代码

```c
int hci_get_dev_info(void __user *arg)
{
    struct hci_dev *hdev;
    struct hci_dev_info di;
    unsigned long flags;
    int err = 0;

    if (copy_from_user(&di, arg, sizeof(di)))
        return -EFAULT;

    hdev = hci_dev_get(di.dev_id);
    if (!hdev)
        return -ENODEV;

    /* When the auto-off is configured it means the transport
     * is running, but in that case still indicate that the
     * device is actually down.
     */
    if (hci_dev_test_flag(hdev, HCI_AUTO_OFF))
        flags = hdev->flags & ~BIT(HCI_UP);
    else
        flags = hdev->flags;

    strcpy(di.name, hdev->name);
    di.bdaddr   = hdev->bdaddr;
    di.type     = (hdev->bus & 0x0f) | ((hdev->dev_type & 0x03) << 4);
    di.flags    = flags;
    di.pkt_type = hdev->pkt_type;
    if (lmp_bredr_capable(hdev)) {
        di.acl_mtu  = hdev->acl_mtu;
        di.acl_pkts = hdev->acl_pkts;
        di.sco_mtu  = hdev->sco_mtu;
        di.sco_pkts = hdev->sco_pkts;
    } else {
        di.acl_mtu  = hdev->le_mtu;
        di.acl_pkts = hdev->le_pkts;
        di.sco_mtu  = 0;
        di.sco_pkts = 0;
    }
    di.link_policy = hdev->link_policy;
    di.link_mode   = hdev->link_mode;

    memcpy(&di.stat, &hdev->stat, sizeof(di.stat));
    memcpy(&di.features, &hdev->features, sizeof(di.features));

```

```
46      if (copy_to_user(arg, &di, sizeof(di)))
47          err = -EFAULT;
48
49      hci_dev_put(hdev);
50
51      return err;
52  }
```

直接调用copy_to_user把设备信息返回给user space了

步骤3：打印

```c
static void print_dev_info(int ctl, struct hci_dev_info *di)
{
    struct hci_dev_stats *st = &di->stat;
    char *str;

    print_dev_hdr(di);

    str = hci_dflagstostr(di->flags);
    printf("\t%s\n", str);
    bt_free(str);

    printf("\tRX bytes:%d acl:%d sco:%d events:%d errors:%d\n",
        st->byte_rx, st->acl_rx, st->sco_rx, st->evt_rx, st->err_rx);

    printf("\tTX bytes:%d acl:%d sco:%d commands:%d errors:%d\n",
        st->byte_tx, st->acl_tx, st->sco_tx, st->cmd_tx, st->err_tx);

    if (all && !hci_test_bit(HCI_RAW, &di->flags)) {
        print_dev_features(di, 0);

        if (((di->type & 0x30) >> 4) == HCI_PRIMARY) {
            print_pkt_type(di);
            print_link_policy(di);
            print_link_mode(di);

            if (hci_test_bit(HCI_UP, &di->flags)) {
                cmd_name(ctl, di->dev_id, NULL);
                cmd_class(ctl, di->dev_id, NULL);
            }
        }

        if (hci_test_bit(HCI_UP, &di->flags))
            cmd_version(ctl, di->dev_id, NULL);
    }

    printf("\n");
}
```

四. Bluez qcom init流程(kernel space download fw)

步骤1：Kernel加载前置行为

加载hci_uart.ko

```
module_init(hci_uart_init);
```

C | 复制代码

```c
static int __init hci_uart_init(void)
{
    static struct tty_ldisc_ops hci_uart_ldisc;
    int err;

    BT_INFO("HCI UART driver ver %s", VERSION);

    /* Register the tty discipline */

    memset(&hci_uart_ldisc, 0, sizeof(hci_uart_ldisc));
    hci_uart_ldisc.magic        = TTY_LDISC_MAGIC;
    hci_uart_ldisc.name     = "n_hci";
    hci_uart_ldisc.open     = hci_uart_tty_open;
    hci_uart_ldisc.close        = hci_uart_tty_close;
    hci_uart_ldisc.read     = hci_uart_tty_read;
    hci_uart_ldisc.write        = hci_uart_tty_write;
    hci_uart_ldisc.ioctl        = hci_uart_tty_ioctl;
    hci_uart_ldisc.poll     = hci_uart_tty_poll;
    hci_uart_ldisc.receive_buf  = hci_uart_tty_receive;
    hci_uart_ldisc.write_wakeup = hci_uart_tty_wakeup;
    hci_uart_ldisc.owner        = THIS_MODULE;

    err = tty_register_ldisc(N_HCI, &hci_uart_ldisc);
    if (err) {
        BT_ERR("HCI line discipline registration failed. (%d)", err);
        return err;
    }

#ifdef CONFIG_BT_HCIUART_H4
    h4_init();
#endif
#ifdef CONFIG_BT_HCIUART_BCSP
    bcsp_init();
#endif
#ifdef CONFIG_BT_HCIUART_LL
    ll_init();
#endif
#ifdef CONFIG_BT_HCIUART_ATH3K
    ath_init();
#endif
#ifdef CONFIG_BT_HCIUART_3WIRE
    h5_init();
#endif
#ifdef CONFIG_BT_HCIUART_INTEL
    intel_init();
```

```
#endif
#ifdef CONFIG_BT_HCIUART_BCM
    bcm_init();
#endif
#ifdef CONFIG_BT_HCIUART_QCA
    qca_init();
#endif
#ifdef CONFIG_BT_HCIUART_AG6XX
    ag6xx_init();
#endif
#ifdef CONFIG_BT_HCIUART_MRVL
    mrvl_init();
#endif

    return 0;
}
```

这块本身是hci_uart.ko,另外，里面有很多vendor也是一个ko，高通的是用的btqca.ko

步骤2：btqca.ko的执行入口

C | 复制代码

```c
int __init qca_init(void)
{
    serdev_device_driver_register(&qca_serdev_driver);

    return hci_uart_register_proto(&qca_proto);
}
```

1）其中serdev_device_driver_register(&qca_serdev_driver);是注册serial device driver,这里就牵扯到device platform驱动框架，注册如下：

```c
static struct serdev_device_driver qca_serdev_driver = {
    .probe = qca_serdev_probe,
    .remove = qca_serdev_remove,
    .driver = {
        .name = "hci_uart_qca",
        .of_match_table = of_match_ptr(qca_bluetooth_of_match),
        .acpi_match_table = ACPI_PTR(qca_bluetooth_acpi_match),
        .shutdown = qca_serdev_shutdown,
        .pm = &qca_pm_ops,
    },
};
```

其中of_match_table就是配合dts来使用

```c
.of_match_table = of_match_ptr(qca_bluetooth_of_match),
```

其中acpi_match_table就是配合bios的acpi来使用

```c
.acpi_match_table = ACPI_PTR(qca_bluetooth_acpi_match),
```

匹配后就可以自动运行probe函数

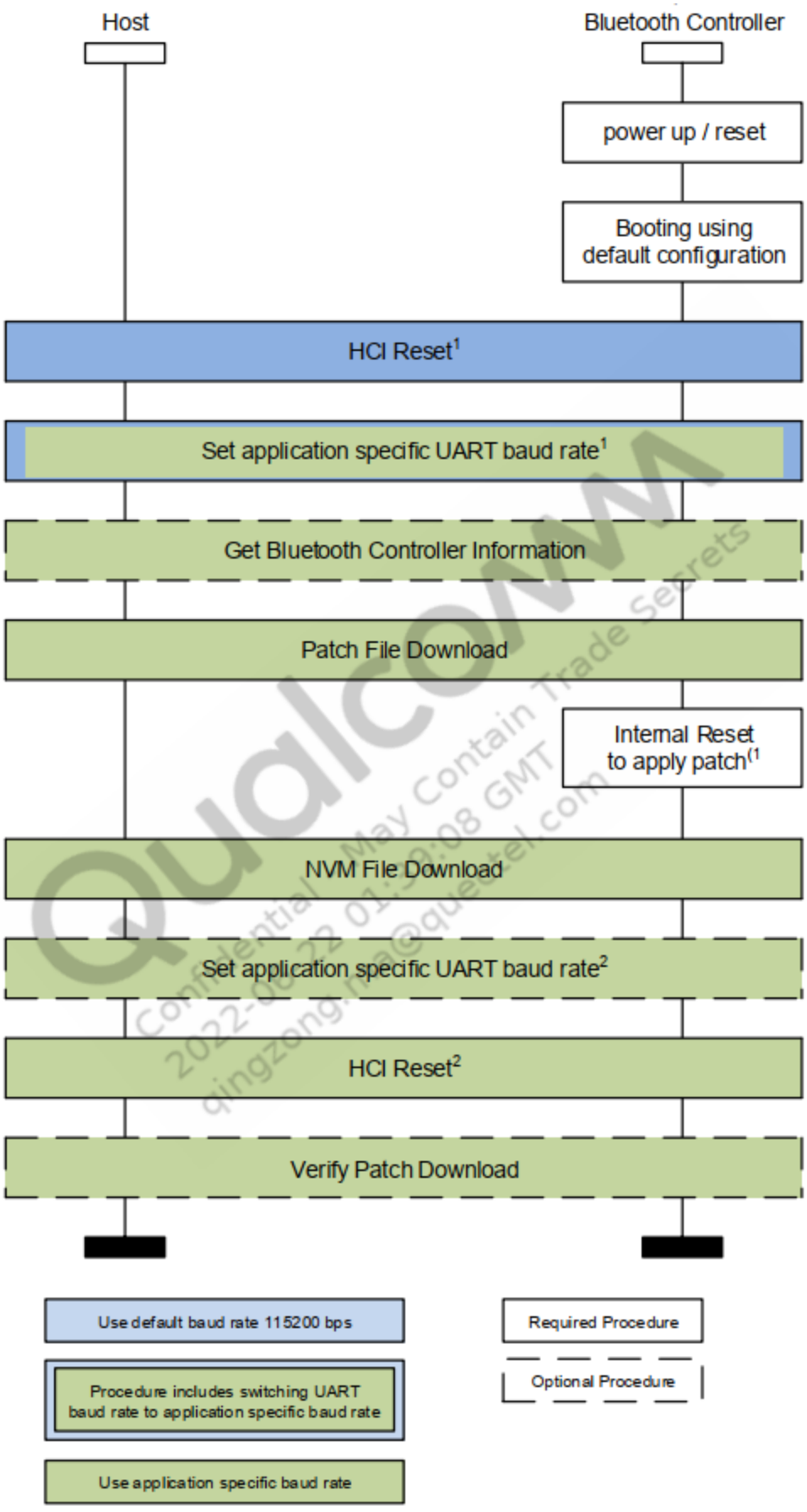
2）return hci_uart_register_proto(&qca_proto);

执行protocol注册一下结构体

```c
static const struct hci_uart_proto qca_proto = {
	.id	= HCI_UART_QCA,
	.name		= "QCA",
	.manufacturer	= 29,
	.init_speed = 115200,
	.oper_speed = 3000000,
	.open		= qca_open,
	.close		= qca_close,
	.flush		= qca_flush,
	.setup		= qca_setup,
	.recv		= qca_recv,
	.enqueue	= qca_enqueue,
	.dequeue	= qca_dequeue,
};
```

会先执行qca_setup函数

里面就执行比较简答了，就是跟user space下载固件基本一样的步骤



在这里不做过多的介绍，看代码即可

五. User space跟Kernel差异

	启动	低功耗	维护
User Space	hciattach	NO	简单，制作patch，基本不存在不同bluez版本patch冲突问题
Kernel Space	自动probe或者btattach	SIBS	patch会有在不同的kernel版本冲突问题，维护需要根据不同的kernel制作不同的patch