

①

```
struct hci_rp_read_bd_addr *bda;
struct sk_buff *skb;
```

```
skb = __hci_cmd_sync(hdev, HCI_OP_READ_BD_ADDR, 0, NULL,
                    HCI_INIT_TIMEOUT);
```

```
if (IS_ERR(skb)) {
```

```
int err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Reading
device address failed(%d)", err);
return err;
```

```
if (skb->len != sizeof(*bda)) {
    bt_dev_err(hdev, "BCM: Device
    address length mismatch");
    kfree_skb(skb);
    return -EIO;
```

```
bda = (struct hci_rp_read_bd_addr *)skb->data;
```

```
if (!bacmp(&bda->bdaddr, BDADDR_BCM20702A0) ||
    !bacmp(&bda->bdaddr, BDADDR_BCM4324B3) ||
    !bacmp(&bda->bdaddr, BDADDR_BCM4330B1)) {
```

```
kfree_skb(skb);
return 0;
bt_dev_info(hdev, "BCM: Using
default device address
(%pMR)", &bda->bdaddr);
set_bit(HCI_QUIRK_INVALID_BDADDR,
        &hdev->quirks);
```

②

```
struct sk_buff *skb;
int err;
skb = __hci_cmd_sync(hdev, 0xfc01, 6, bdaddr,
                    HCI_INIT_TIMEOUT);
```

```
if (IS_ERR(skb)) {
```

```
err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Change address
command failed(%d)", err);
return err;
```

```
kfree_skb(skb);
```

```
return 0;
```

③

```
const struct hci_command_hdr *cmd;
const u8 *fw_ptr;
size_t fw_size;
struct sk_buff *skb;
u16 opcode;
int err = 0;
skb = __hci_cmd_sync(hdev, 0xfc2e, 0, NULL, HCI_INIT_TIMEOUT);
```

```
if (IS_ERR(skb)) {
```

```
err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Download
Minidrv command failed
(%d)", err);
goto done;
```

```
kfree_skb(skb);
msleep(50);
fw_ptr = fw->data;
fw_size = fw->size;
```

```
msleep(250);
done:
```

```
return err;
```

```
while (fw_size >= sizeof(*cmd)) {
```

```
const u8 *cmd_param;
```

```
cmd = (struct hci_command_hdr *)fw_ptr;
fw_ptr += sizeof(*cmd);
fw_size -= sizeof(*cmd);
```



```

    if (fw_size < cmd->plen) {
        bt_dev_err(hdev, "BCM: Patch is corrupted");
        err = -EINVAL;
        goto done;
    }
    cmd_param = fw_ptr;
    fw_ptr += cmd->plen;
    fw_size -= cmd->plen;
    opcode = le16_to_cpu(cmd->opcode);
    skb = hci_cmd_sync(hdev, opcode,
        cmd->plen, cmd_param, HCI_INIT_TIMEOUT);

```

Termina

```

    if (IS_ERR(skb)) {
        kfree_skb(skb);
        err = PTR_ERR(skb);
        bt_dev_err(hdev, "BCM: Patch command
            %04x failed (%d)", opcode, err);
        goto done;
    }

```

Termina

```

    struct sk_buff *skb;

    skb = __hci_cmd_sync(hdev, HCI_OP_RESET, 0, NULL,
        HCI_INIT_TIMEOUT);

    if (IS_ERR(skb)) {
        kfree_skb(skb);
        msleep(100);
        return 0;
    }
    int err = PTR_ERR(skb);
    bt_dev_err(hdev, "BCM: Reset failed (%d)", err);
    return err;

```

Termina

```

    struct sk_buff *skb;

    skb = __hci_cmd_sync(hdev, HCI_OP_READ_LOCAL_NAME, 0, NULL,
        HCI_INIT_TIMEOUT);

    if (IS_ERR(skb)) {
        bt_dev_err(hdev, "BCM: Reading local name failed
            (%d)", PTR_ERR(skb));
        return skb;
    }
    if (skb->len != sizeof(struct
        hci_rp_read_local_name)) {
        bt_dev_err(hdev, "BCM:
            Local name length mismatch");
        kfree_skb(skb);
        return ERR_PTR(-EIO);
    }
    return skb;

```

Termina

```

    struct sk_buff *skb;

    skb = __hci_cmd_sync(hdev, HCI_OP_READ_LOCAL_VERSION, 0, NULL,
        HCI_INIT_TIMEOUT);

    if (IS_ERR(skb)) {
        bt_dev_err(hdev, "BCM: Reading local version info
            failed (%d)", PTR_ERR(skb));
        return skb;
    }
    if (skb->len != sizeof(struct
        hci_rp_read_local_version)) {
        bt_dev_err(hdev, "BCM: Local version length mismatch");
        kfree_skb(skb);
        return ERR_PTR(-EIO);
    }
    return skb;

```

Termina

7 struct sk_buff *skb;

skb = __hci_cmd_sync(hdev, 0xfc79, 0, NULL, HCI_INIT_TIMEOUT);

if (IS_ERR(skb)) {

if (skb->len != 7) {

bt_dev_err(hdev, "BCM: Verbose
config length mismatch");

kfree_skb(skb);

return ERR_PTR(-EIO);

return skb;

bt_dev_err(hdev, "BCM: Read verbose config info failed
(%ld)",

PTR_ERR(skb));

return skb;

8. struct sk_buff *skb;

skb = __hci_cmd_sync(hdev, 0xfc6e, 0, NULL, HCI_INIT_TIMEOUT);

if (IS_ERR(skb)) {

bt_dev_err(hdev, "BCM: Read
controller features failed

(%ld)", PTR_ERR(skb));

return skb;

if (skb->len != 9) {

bt_dev_err(hdev, "BCM: Controller
features length mismatch");

kfree_skb(skb);

return ERR_PTR(-EIO);

return skb;

9. struct sk_buff *skb;

skb = __hci_cmd_sync(hdev, 0xfc5a, 0, NULL, HCI_INIT_TIMEOUT);

if (IS_ERR(skb)) {

bt_dev_err(hdev, "BCM: Read USB
product info failed

(%ld)", PTR_ERR(skb));

return skb;

if (skb->len != 5) {

bt_dev_err(hdev, "BCM:
USB product length mismatch");

kfree_skb(skb);

return ERR_PTR(-EIO);

return skb;

10. struct sk_buff *skb;

skb = btbcm_read_verbose_config(hdev);

if (IS_ERR(skb))

return PTR_ERR(skb);

bt_dev_info(hdev, "BCM: chip
id %u", skb->data[1]);

kfree_skb(skb);

if (IS_ERR(skb))

return PTR_ERR(skb);

bt_dev_info(hdev, "BCM: features 0x%2.2x", skb->data[1]);

kfree_skb(skb);

skb = btbcm_read_local_name(hdev);

if (IS_ERR(skb))

return PTR_ERR(skb);

bt_dev_info(hdev, "%s", (char *) (skb->data + 1));

kfree_skb(skb);

return 0;


```

11.          ul6 subver;
              const char *name;

              { 0x4103, "BCM4330B1" }, /* 002.001.003 */
              { 0x410e, "BCM43341B0" }, /* 002.001.014 */
              { 0x4406, "BCM4324B3" }, /* 002.004.006 */
              { 0x610c, "BCM4354" }, /* 003.001.012 */
              { 0x2209, "BCM43430A1" }, /* 001.002.009 */
              { 0x6119, "BCM4345C0" }, /* 003.001.025 */
              { 0x230f, "BCM4356A2" }, /* 001.003.015 */
              { }

12. ul6 subver, rev;
    const char *hw_name = NULL;
    struct sk_buff *skb;
    struct hci_rp_read_local_version *ver;
    int i, err;
    err = btbcm_reset(hdev);

    if (err)
        return err;
    skb = btbcm_read_local_version(hdev);
    if (IS_ERR(skb))
        return PTR_ERR(skb);
    ver = (struct hci_rp_read_local_version *)skb->data;
    rev = le16_to_cpu(ver->hci_rev);
    subver = le16_to_cpu(ver->lmp_subver);
    kfree_skb(skb);

    switch ((rev & 0xf000) >> 12) {
    case 0:
    case 1:
    case 2:
    case 3:
        for (i = 0; bcm_uart_subver_table[i].name; i++) {
            if (subver == bcm_uart_subver_table[i].subver) {
                hw_name = bcm_uart_subver_table[i].name;
                break;
            }
        }
        if (err)
            return err;
        snprintf(fw_name, len, "brcm/%s.hcd",
                 hw_name ? : "BCM", (subver & 0xe000) >> 13,
                 (subver & 0x1f00) >> 8, (subver & 0x00ff), rev &
                 0x0fff);
        return 0;
    default:
        return 0;
    }

```



```

    struct sk_buff *skb;
    struct hci_rp_read_local_version *ver;
    u16 subver, rev;
    int err;

    if (err)
        return err;
    skb = btbcm_read_local_version(hdev);
    if (IS_ERR(skb))
        return PTR_ERR(skb);

    ver = (struct hci_rp_read_local_version *)skb->data;
    rev = le16_to_cpu(ver->hci_rev);
    subver = le16_to_cpu(ver->lmp_subver);
    kfree_skb(skb);

    bt_dev_info(hdev, "BCM (%3.3u.%3.3u.%3.3u) build %4.4u",
        (subver & 0xe000) >> 13, (subver & 0x1f00) >> 8,
        (subver & 0x00ff), rev & 0x0fff);

    btbcm_check_bdaddr(hdev);

    set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER, &hdev->quirks);

    return 0;

```

14

```

    u16 subver;
    const char *name;

    { 0x210b, "BCM43142A0" }, /* 001.001.011 */
    { 0x2112, "BCM4314A0" }, /* 001.001.018 */
    { 0x2118, "BCM20702A0" }, /* 001.001.024 */
    { 0x2126, "BCM4335A0" }, /* 001.001.038 */
    { 0x220e, "BCM20702A1" }, /* 001.002.014 */
    { 0x230f, "BCM4354A2" }, /* 001.003.015 */
    { 0x4106, "BCM4335B0" }, /* 002.001.006 */
    { 0x410e, "BCM20702B0" }, /* 002.001.014 */
    { 0x6109, "BCM4335C0" }, /* 003.001.009 */
    { 0x610c, "BCM4354" }, /* 003.001.012 */
    { }

```

15

```

    char fw_name[64];
    const struct firmware *fw;
    u16 subver, rev, pid, vid;
    const char *hw_name = NULL;
    struct sk_buff *skb;
    struct hci_rp_read_local_version *ver;
    int i, err;

```

```

    if (err)
        return err;

    if (IS_ERR(skb))
        return PTR_ERR(skb);

    ver = (struct hci_rp_read_local_version *)skb->data;
    rev = le16_to_cpu(ver->hci_rev);
    subver = le16_to_cpu(ver->lmp_subver);
    kfree_skb(skb);

```



```

        if (err)
            return err;

switch ((rev & 0xf000) >> 12) {
case 0:
case 3:
for (i = 0; bcm_uart_subver_table[i].name; i++) {
    if (subver == bcm_uart_subver_table[i].subver) {
        hw_name = bcm_uart_subver_table[i].name;
        snprintf(hw_name, sizeof(hw_name), "bcm/%s.hcd",
            hw_name ? : "BCM");
        break;
        case 1:
        case 2:
            if (IS_ERR(skb))
                return PTR_ERR(skb);
            vid = get_unaligned_le16(skb->data + 1);
            pid = get_unaligned_le16(skb->data + 3);
            kfree_skb(skb);
            for (i = 0; bcm_usb_subver_table[i].name; i++) {
                if (subver == bcm_usb_subver_table[i].subver) {
                    hw_name = bcm_usb_subver_table[i].name;
                    snprintf(hw_name, sizeof(hw_name), "bcm/%s-%4.4x",
                        hw_name ? : "BCM", vid, pid);
                    break;
                    default:
                        return 0;
                }
            }
            bt_dev_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",
                hw_name ? : "BCM", (subver & 0xe000) >> 13,
                (subver & 0x1f00) >> 8, (subver & 0x00ff), rev &
                err = request_firmware(&fw, fw_name, &hdev->dev);
            if (err < 0) {
                btbcm_patchram(hdev, fw);
                bt_dev_info(hdev, "BCM: Patch %s not found", fw_name);
                goto done;
            }
            release_firmware(fw);
            if (err)
                return err;

skb = btbcm_read_local_version(hdev);
if (IS_ERR(skb))
    return PTR_ERR(skb);

ver = (struct hci_rp_read_local_version *)skb->data;
rev = le16_to_cpu(ver->hci_rev);
subver = le16_to_cpu(ver->lmp_subver);
kfree_skb(skb);

bt_dev_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",
    hw_name ? : "BCM", (subver & 0xe000) >> 13,
    (subver & 0x1f00) >> 8, (subver & 0x00ff), rev);
skb = btbcm_read_local_name(hdev);
if (IS_ERR(skb))
    return PTR_ERR(skb);

```


x/

```
bt_dev_info(hdev, "%s", (char *) (skb->data + 1));
kfree_skb(skb);
```

```
btbcm_check_bdaddr(hdev);
```

```
set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER, &hdev->quirks);
```

```
return 0;
```

Termina

```
15 struct sk_buff *skb;
    int err;
```

```
/* Reset */
```

```
err = btbcm_reset(hdev);
```

x if (err)

return err;

```
skb = btbcm_read_verbose_config(hdev);
```

x if (!IS_ERR(skb)) {

```
skb = btbcm_read_usb_product(hdev); bt_dev_info(hdev, "BCM: chip id %u build %4.4u",
                                             skb->data[1], get_unaligned_le16(skb->data +
```

5));

kfree_skb(skb);

x if (!IS_ERR(skb)) {

```
bt_dev_info(hdev, "BCM: product %4.4x:%4.4x",
             get_unaligned_le16(skb->data + 1),
             get_unaligned_le16(skb->data + 3));
```

```
kfree_skb(skb);
```

```
skb = btbcm_read_controller_features(hdev);
```

x if (!IS_ERR(skb)) {

```
bt_dev_info(hdev, "BCM: features 0x%2.2x", skb->data[0],
             get_unaligned_le16(skb->data + 1));
kfree_skb(skb);
```

Termina

```
skb = btbcm_read_local_name(hdev);
```

x if (!IS_ERR(skb)) {

```
bt_dev_info(hdev, "%s", (char *) (skb->data + 1));
```

```
kfree_skb(skb);
```

Termina

```
set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER, &hdev->quirks);
```

```
return 0;
```

Termina