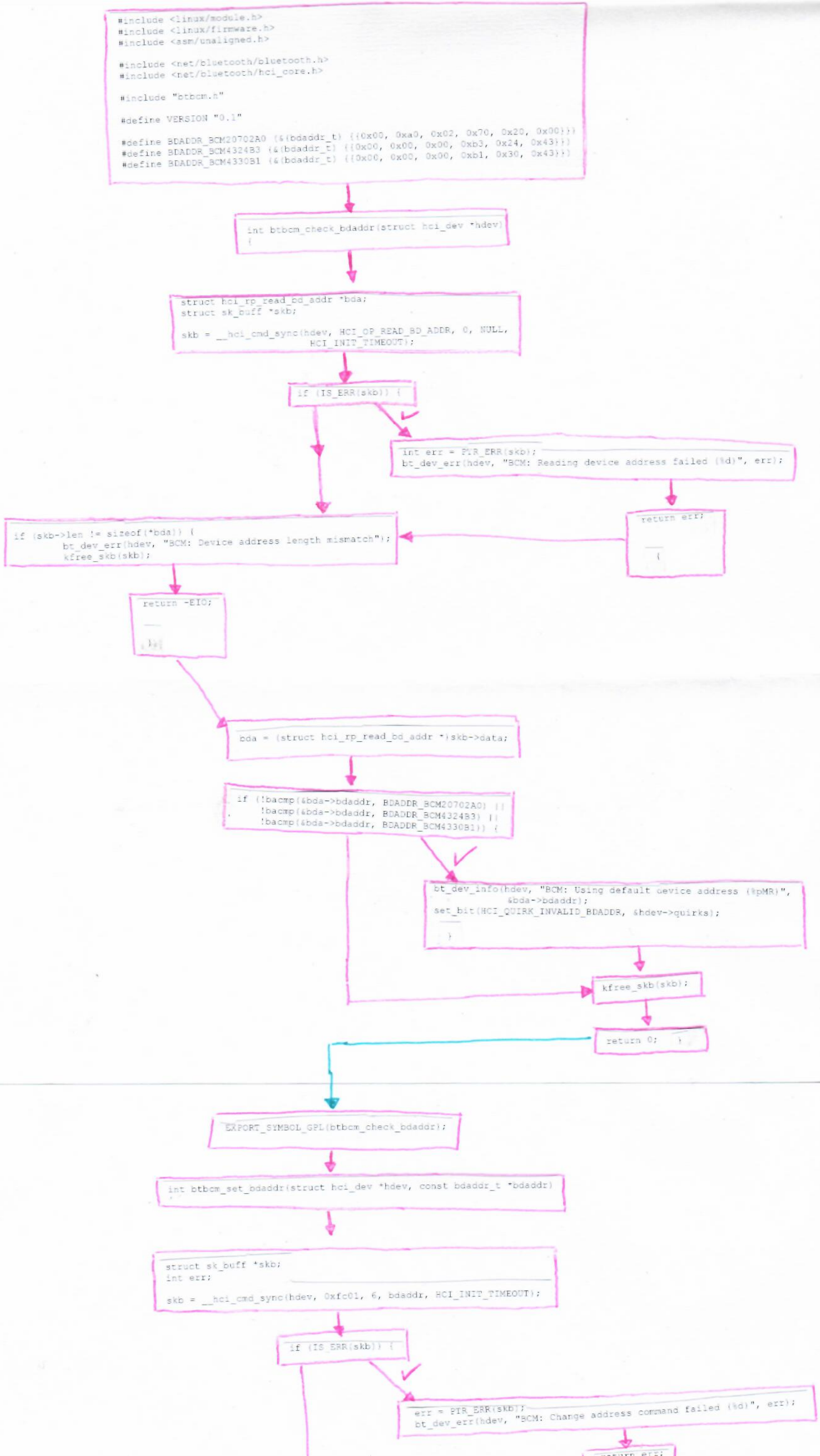


```
#include <linux/module.h>
#include <linux/firmware.h>
#include <asm/unaligned.h>

#include <net/bluetooth/bluetooth.h>
#include <net/bluetooth/hci_core.h>

#include "btbcm.h"

#define VERSION "0.1"

#define BDADDR_BCM20702A0 (&(bdaddr_t) {{0x00, 0xa0, 0x02, 0x70, 0x20, 0x00}})
#define BDADDR_BCM4324B3 (&(bdaddr_t) {{0x00, 0x00, 0x00, 0xb3, 0x24, 0x43}})
#define BDADDR_BCM4330B1 (&(bdaddr_t) {{0x00, 0x00, 0x00, 0xb1, 0x30, 0x43}})
```

```
int btbcm_check_bdaddr(struct hci_dev *hdev)
{
```

```
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)) {
```

✓

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

}
```

```
kfree_skb(skb);
```

```
return 0;  }
```

```
EXPORT_SYMBOL_GPL(btbcm_check_bdaddr);
```

```
int btbcm_set_bdaddr(struct hci_dev *hdev, const bdaddr_t *bdaddr)
```

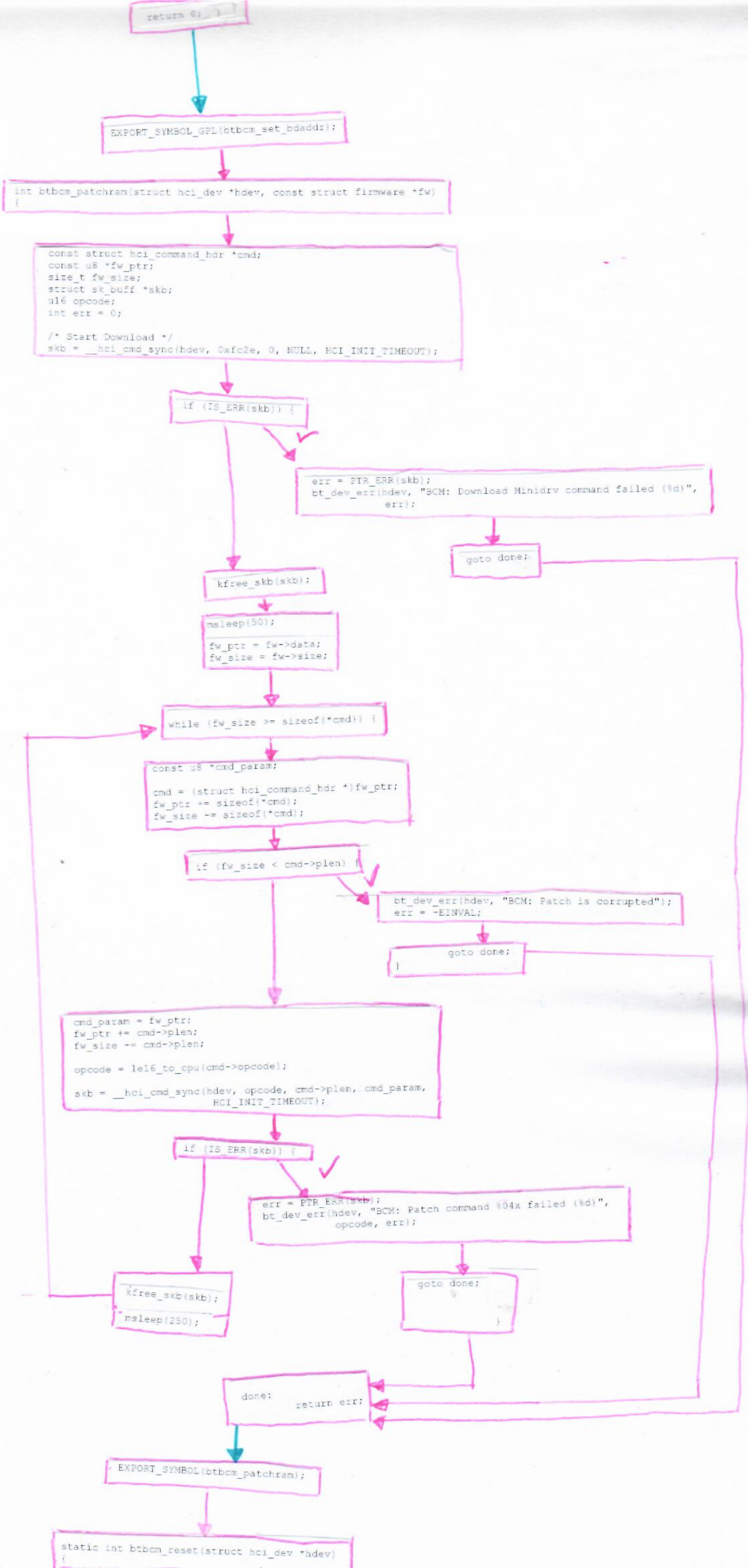
```
struct sk_buff *skb;
int err;

skb = __hci_cmd_sync(hdev, 0xfc01, 6, bdaddr, HCI_INIT_TIMEOUT);
```

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

✓

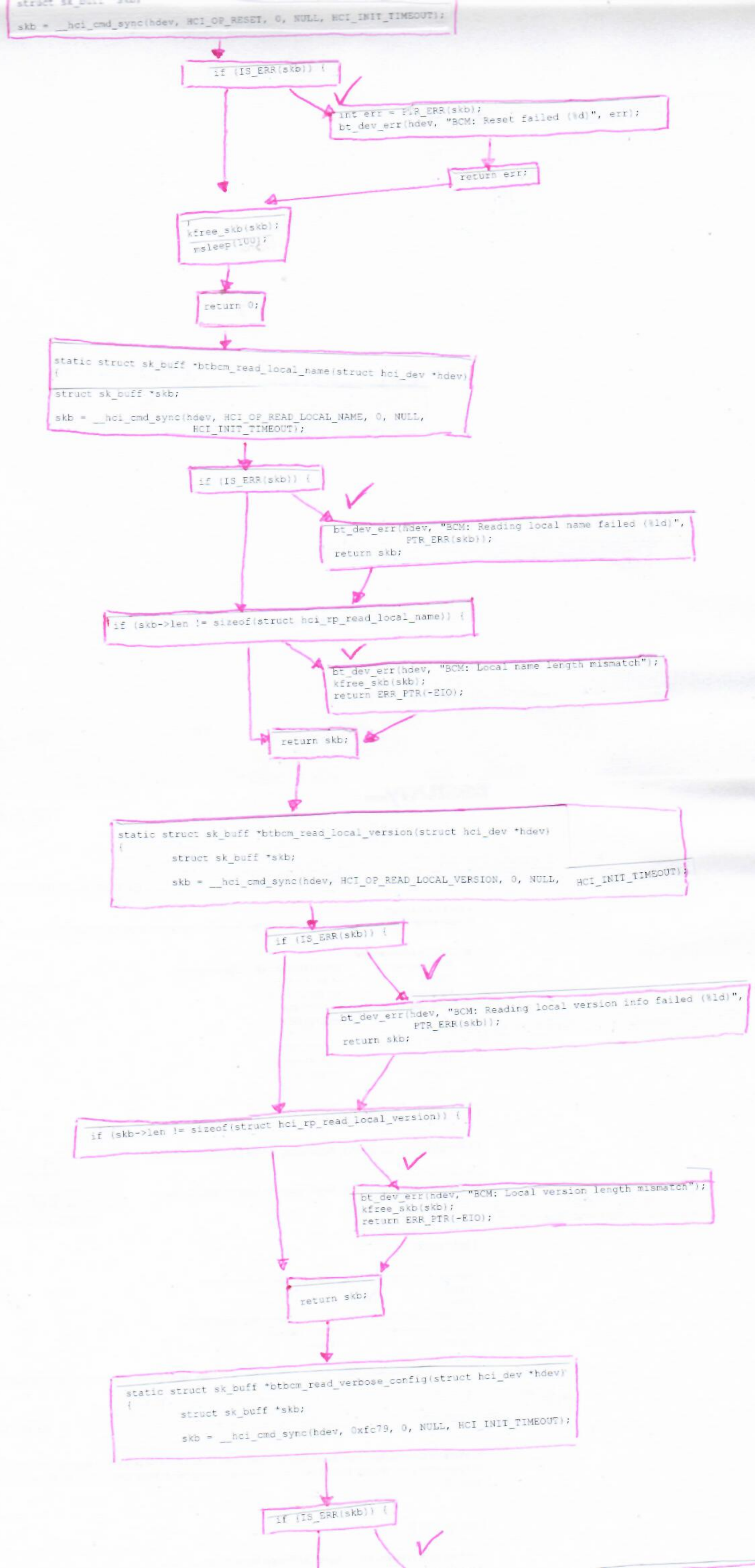
```
err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Change address command failed (%d)", err);
```

```
return err;
```

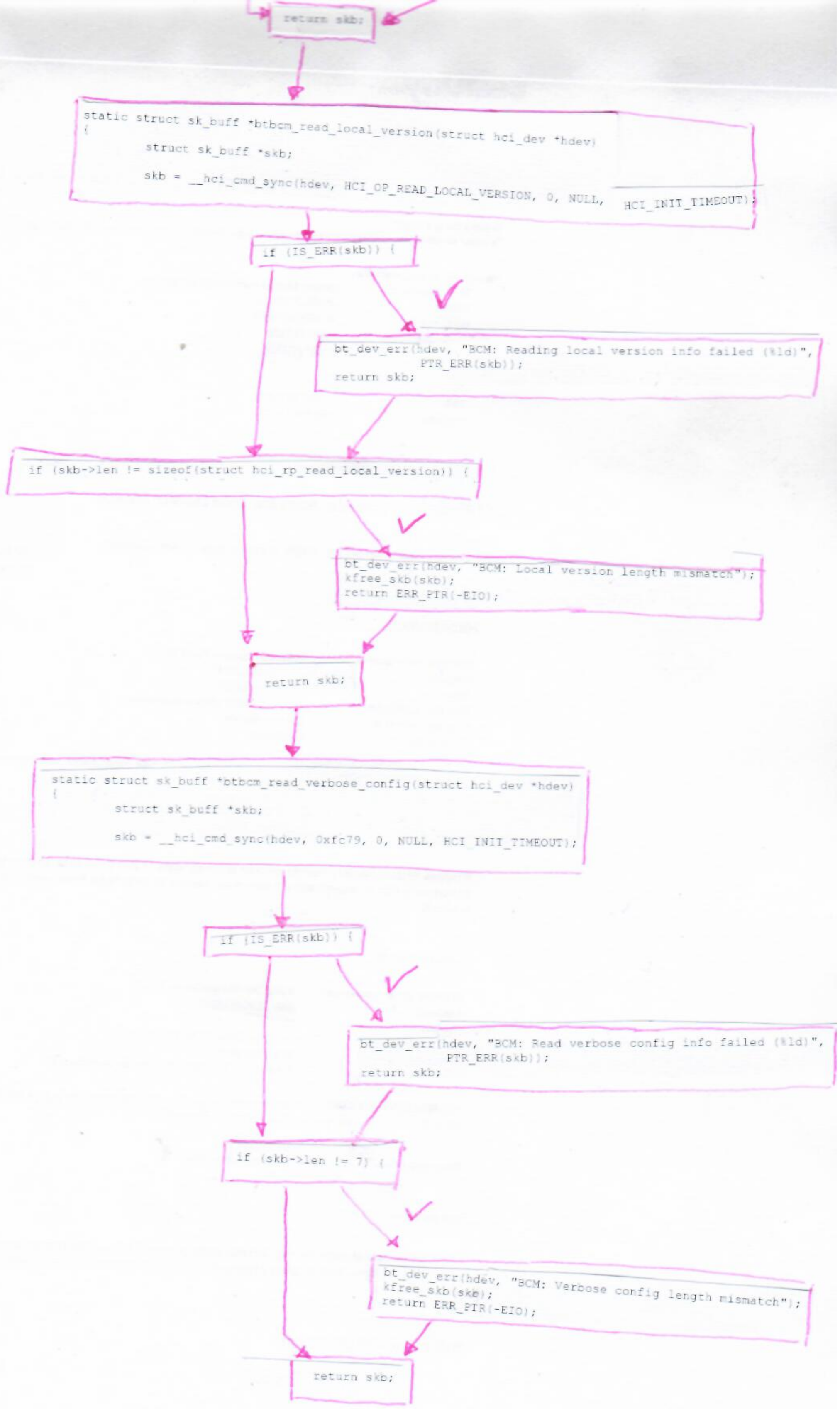


```
return 0; }

EXPORT_SYMBOL_GPL(btbcm_set_bdaddr);

int btbcm_patchram(struct hci_dev *hdev, const struct firmware *fw)
{

const struct hci_command_hdr *cmd;
const u8 *fw_ptr;
size_t fw_size;
struct sk_buff *skb;
u16 opcode;
int err = 0;

/* Start Download */
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;

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);

if (IS_ERR(skb)) {

err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Patch command %04x failed (%d)",
           opcode, err);

goto done;
}

kfree_skb(skb);

msleep(250);

done:    return err;

EXPORT_SYMBOL(btbcm_patchram);

static int btbcm_reset(struct hci_dev *hdev)
{
```

2



```
struct sk_buff *skb;

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

if (IS_ERR(skb)) {

int err = PTR_ERR(skb);
bt_dev_err(hdev, "BCM: Reset failed (%d)", err);

return err;

}
kfree_skb(skb);
msleep(100);

return 0;

static struct sk_buff *btbcm_read_local_name(struct hci_dev *hdev)
{
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 (%ld)",
           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;

static struct sk_buff *btbcm_read_local_version(struct hci_dev *hdev)
{
        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 (%ld)",
           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;

static struct sk_buff *btbcm_read_verbose_config(struct hci_dev *hdev)
{
        struct sk_buff *skb;

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

if (IS_ERR(skb)) {
```



```
return skb;
```

```
static struct sk_buff *btbcm_read_local_version(struct hci_dev *hdev)
{
        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 (%ld)",
           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;
```

```
static struct sk_buff *btbcm_read_verbose_config(struct hci_dev *hdev)
{
        struct sk_buff *skb;

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

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

```
bt_dev_err(hdev, "BCM: Read verbose config info failed (%ld)",
           PTR_ERR(skb));
return skb;
```

```
if (skb->len != 7) {
```

```
bt_dev_err(hdev, "BCM: Verbose config length mismatch");
kfree_skb(skb);
return ERR_PTR(-EIO);
```

```
return skb;
```