Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Juan Alfredo Cruz Carlón

Asignatura: Fundamentos de Programación

Grupo: 1107

No de Práctica(s): 12

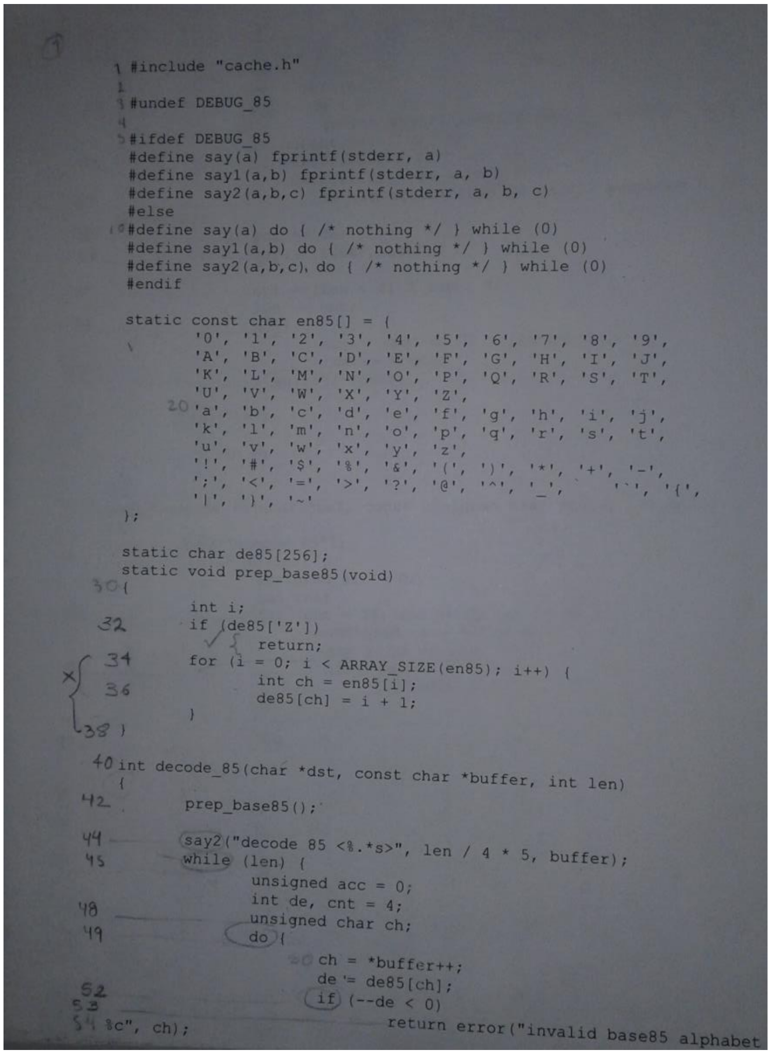
Integrante(s): Avendaño Gavira Paola Guadalupe
Castillo Miranda Camila
Méndez Cambrano Valeria Damari
Rodríguez Morquecho Verónica

Semestre: 2018-1

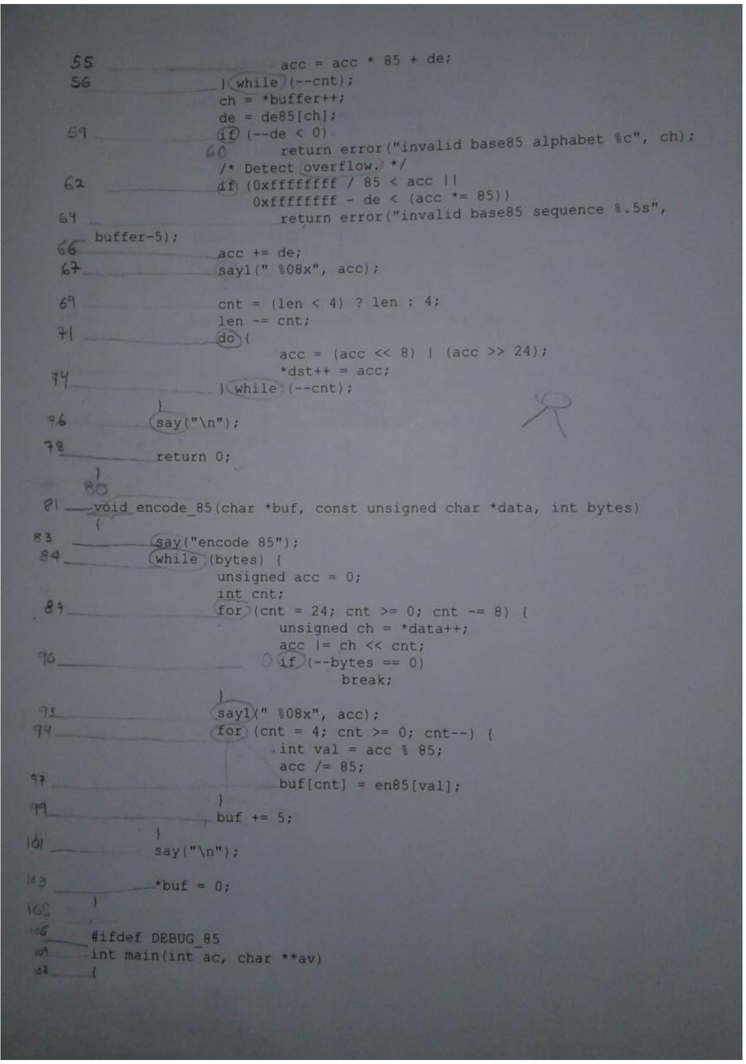
Fecha de entrega: 29/11/17

Observaciones:

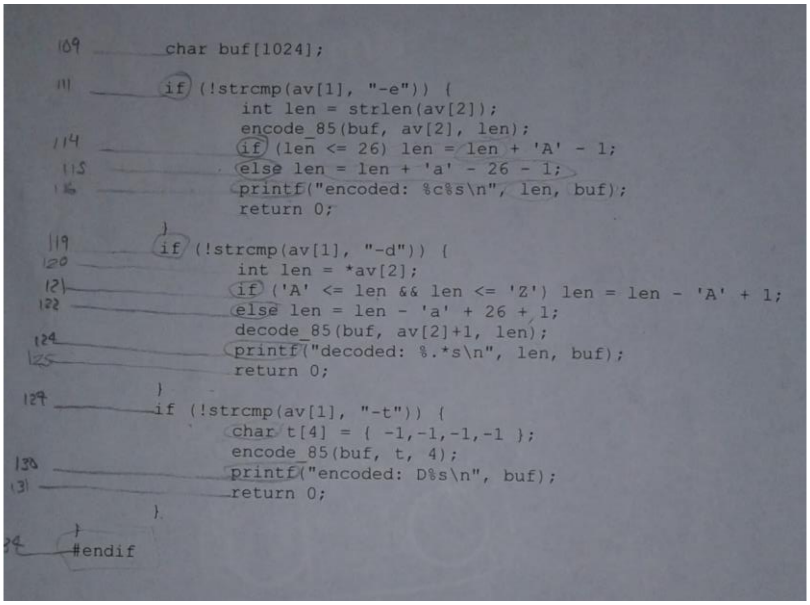
CALIFICACIÓN: ___________



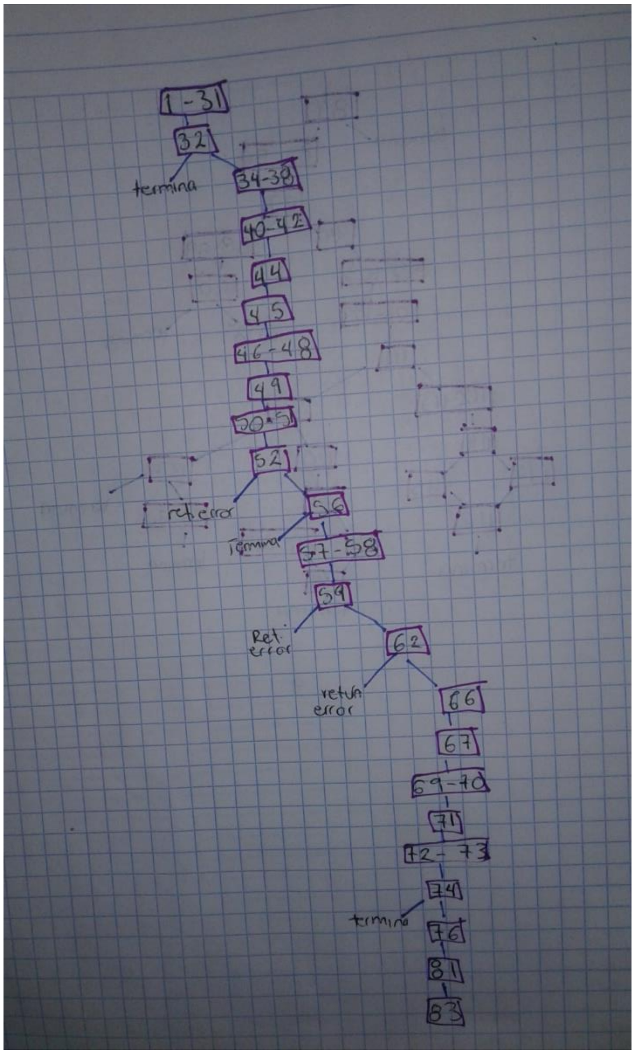
```
#include "cache.h"

#undef DEBUG_85

#ifdef DEBUG_85
#define say(a) fprintf(stderr, a)
#define say1(a,b) fprintf(stderr, a, b)
#define say2(a,b,c) fprintf(stderr, a, b, c)
#else
#define say(a) do { /* nothing */ } while (0)
#define say1(a,b) do { /* nothing */ } while (0)
#define say2(a,b,c) do { /* nothing */ } while (0)
#endif

static const char en85[] = {
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
        'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
        'U', 'V', 'W', 'X', 'Y', 'Z',
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
        'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
        'u', 'v', 'w', 'x', 'y', 'z',
        '!', '#', '$', '%', '&', '(', ')', '*', '+', '-',
        ';', '<', '=', '>', '?', '@', '^', '_',         '`', '{',
        '|', '}', '~'
};

static char de85[256];
static void prep_base85(void)
{
        int i;
        if (de85['Z'])
                return;
        for (i = 0; i < ARRAY_SIZE(en85); i++) {
                int ch = en85[i];
                de85[ch] = i + 1;
        }
}

int decode_85(char *dst, const char *buffer, int len)
{
        prep_base85();

        say2("decode 85 <%.*s>", len / 4 * 5, buffer);
        while (len) {
                unsigned acc = 0;
                int de, cnt = 4;
                unsigned char ch;
                do {
                        ch = *buffer++;
                        de = de85[ch];
                        if (--de < 0)
                                return error("invalid base85 alphabet
%c", ch);
```

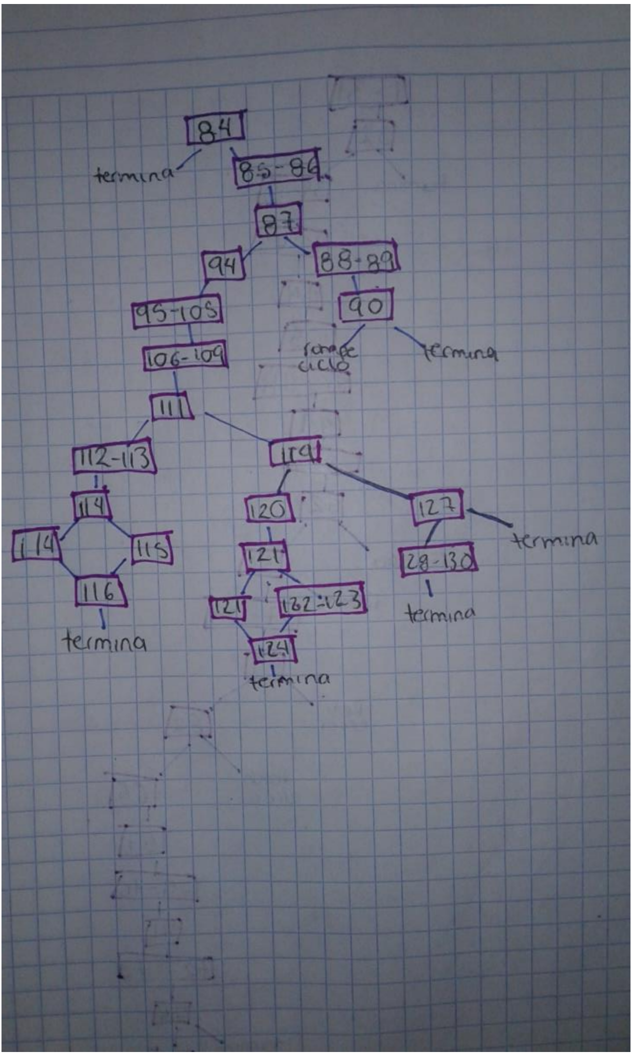


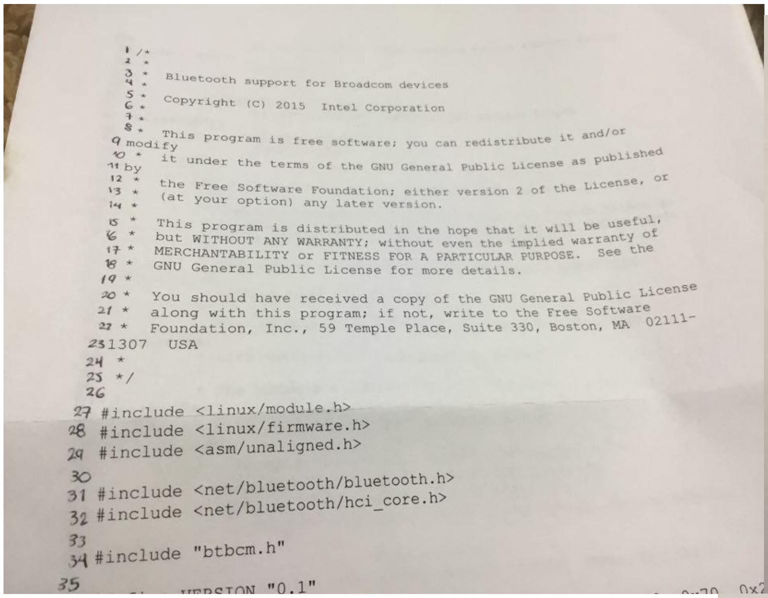
```
                                acc = acc * 85 + de;
                        } while (--cnt);
                        ch = *buffer++;
                        de = de85[ch];
                        if (--de < 0)
                                return error("invalid base85 alphabet %c", ch);
                        /* Detect overflow. */
                        if (0xffffffff / 85 < acc ||
                            0xffffffff - de < (acc *= 85))
                                return error("invalid base85 sequence %.5s",
buffer-5);
                        acc += de;
                        say1(" %08x", acc);

                        cnt = (len < 4) ? len : 4;
                        len -= cnt;
                        do {
                                acc = (acc << 8) | (acc >> 24);
                                *dst++ = acc;
                        } while (--cnt);
                }
                say("\n");

                return 0;
        }

        void encode_85(char *buf, const unsigned char *data, int bytes)
        {
                say("encode 85");
                while (bytes) {
                        unsigned acc = 0;
                        int cnt;
                        for (cnt = 24; cnt >= 0; cnt -= 8) {
                                unsigned ch = *data++;
                                acc |= ch << cnt;
                                if (--bytes == 0)
                                        break;
                        }
                        say1(" %08x", acc);
                        for (cnt = 4; cnt >= 0; cnt--) {
                                int val = acc % 85;
                                acc /= 85;
                                buf[cnt] = en85[val];
                        }
                        buf += 5;
                }
                say("\n");

                *buf = 0;
        }

        #ifdef DEBUG_85
        int main(int ac, char **av)
        {
```



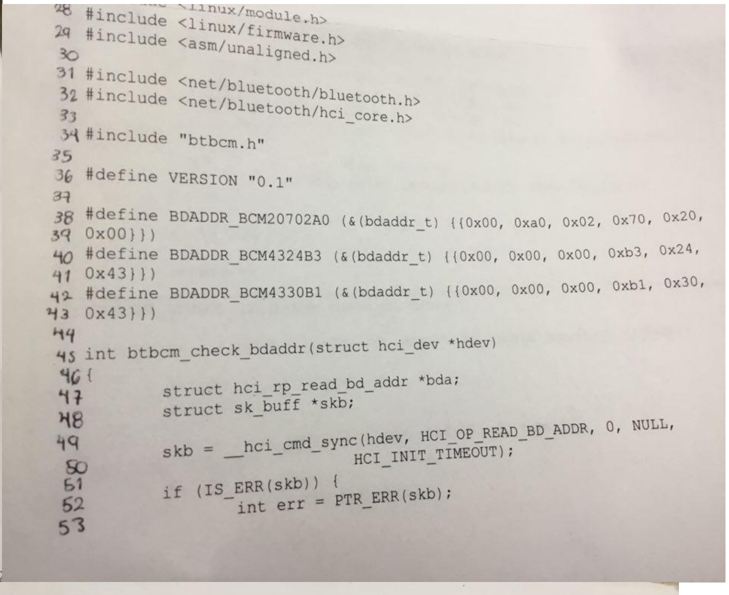
```
                char buf[1024];

                if (!strcmp(av[1], "-e")) {
                        int len = strlen(av[2]);
                        encode_85(buf, av[2], len);
                        if (len <= 26) len = len + 'A' - 1;
                        else len = len + 'a' - 26 - 1;
                        printf("encoded: %c%s\n", len, buf);
                        return 0;
                }
                if (!strcmp(av[1], "-d")) {
                        int len = *av[2];
                        if ('A' <= len && len <= 'Z') len = len - 'A' + 1;
                        else len = len - 'a' + 26 + 1;
                        decode_85(buf, av[2]+1, len);
                        printf("decoded: %.*s\n", len, buf);
                        return 0;
                }
                if (!strcmp(av[1], "-t")) {
                        char t[4] = { -1,-1,-1,-1 };
                        encode_85(buf, t, 4);
                        printf("encoded: D%s\n", buf);
                        return 0;
                }
        }
        #endif
```


1-31
32
termina
34-38
40-42
44
45
46-48
49
50-51
52
ret error
56
termina
57-58
59
Ret error
62
retun error
66
67
69-70
71
72-73
74
termina
76
81
83

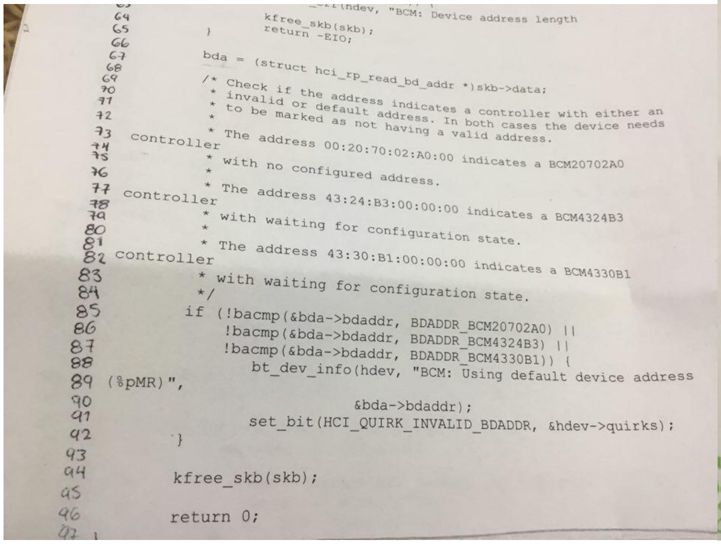

84
termina
85-86
87
94
88-89
90
rompe ciclo
termina
95-105
106-109
111
112-113
119
114
127
termina
114
115
120
121
128-130
116
121
122-123
termina
termina
124
termina



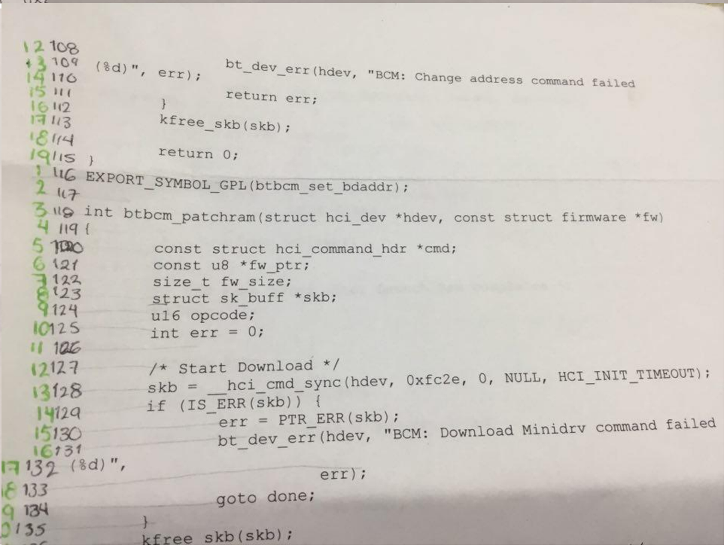
```
/*
 *
 *  Bluetooth support for Broadcom devices
 *
 *  Copyright (C) 2015  Intel Corporation
 *
 *
 *  This program is free software; you can redistribute it and/or
modify
 *  it under the terms of the GNU General Public License as published
by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA
 *
 */

#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"

        VERSION "0.1"
```



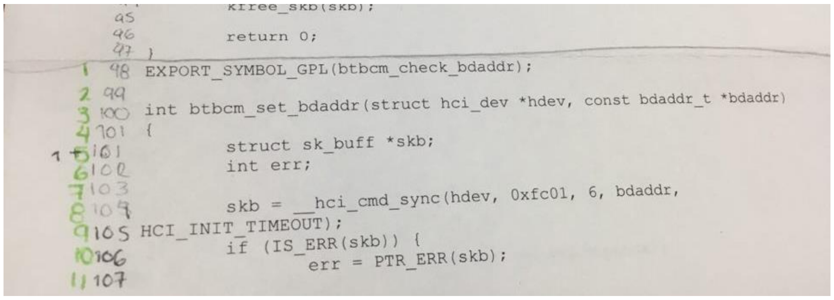
```
#include <linux/module.h>
#include <linux/firmware.h>
#include <asm/unaligned.h>

#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);
```



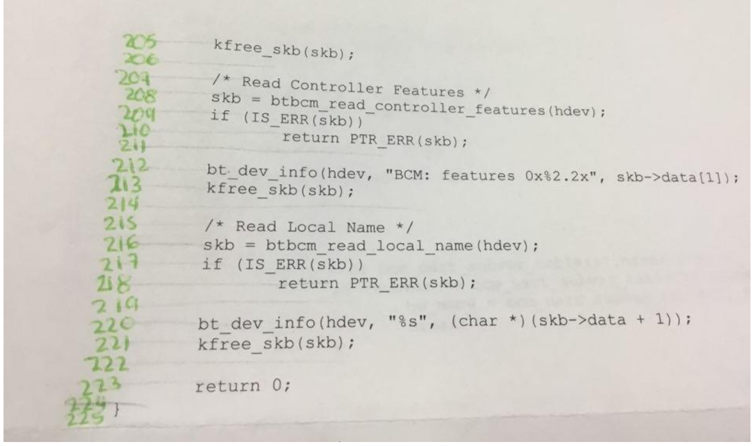
```
                        (hdev, "BCM: Device address length
                kfree_skb(skb);
        }       return -EIO;

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

        /* Check if the address indicates a controller with either an
         * invalid or default address. In both cases the device needs
         * to be marked as not having a valid address.
         *
         * The address 00:20:70:02:A0:00 indicates a BCM20702A0
controller
         * with no configured address.
         *
         * The address 43:24:B3:00:00:00 indicates a BCM4324B3
controller
         * with waiting for configuration state.
         *
         * The address 43:30:B1:00:00:00 indicates a BCM4330B1
controller
         * with waiting for configuration state.
         */
        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;
}
```



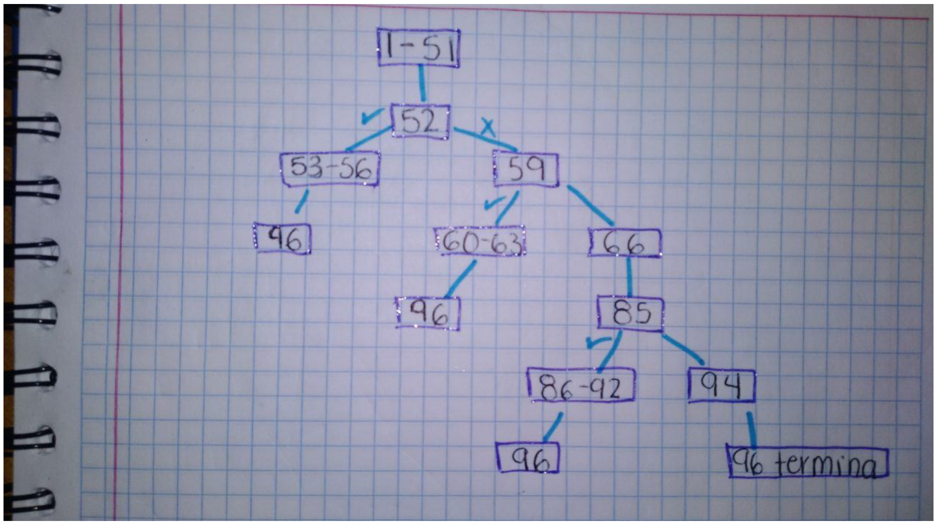
```
                bt_dev_err(hdev, "BCM: Change address command failed
(%d)", err);
                return err;
        }
        kfree_skb(skb);

        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);
```

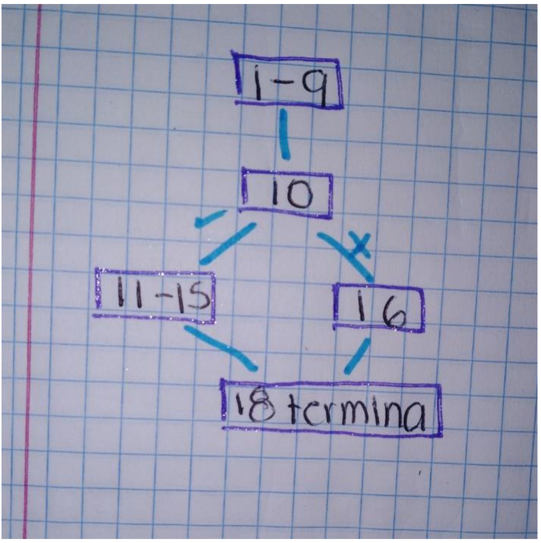


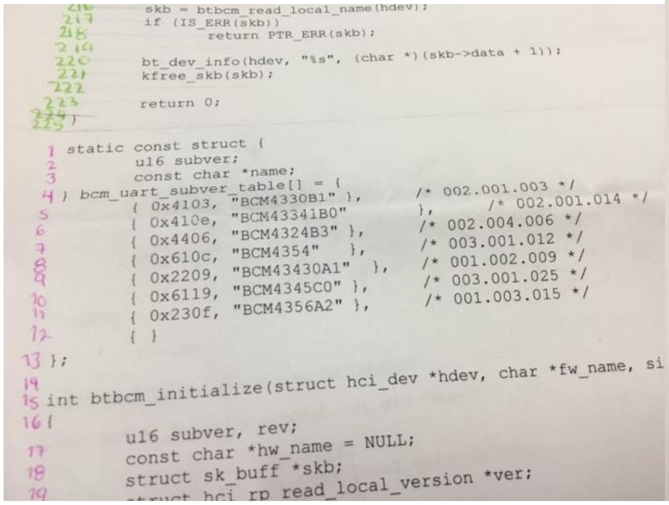
```
        kfree_skb(skb);

        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);
```



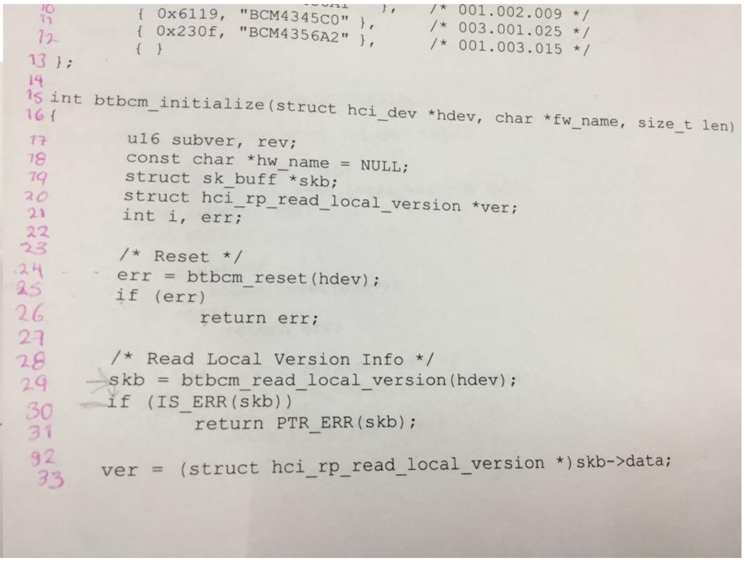
```
        kfree_skb(skb);

        /* Read Controller Features */
        skb = btbcm_read_controller_features(hdev);
        if (IS_ERR(skb))
                return PTR_ERR(skb);

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

        /* Read Local Name */
        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;
}
```



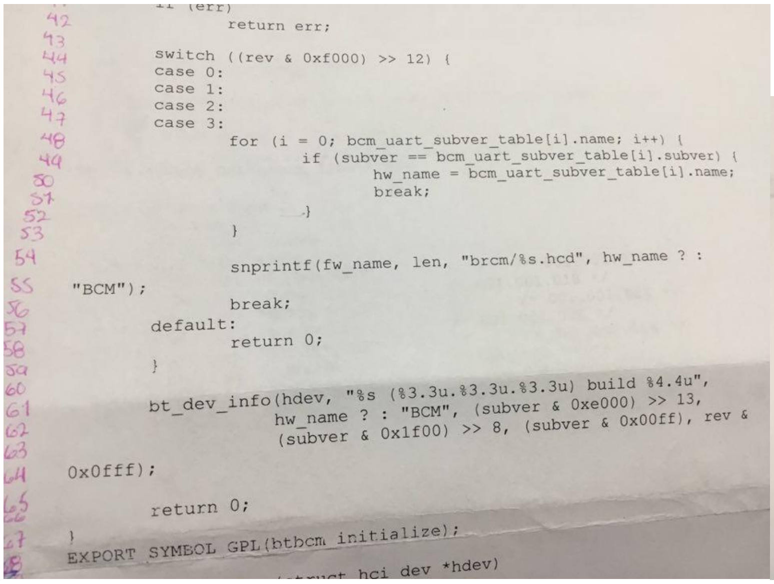




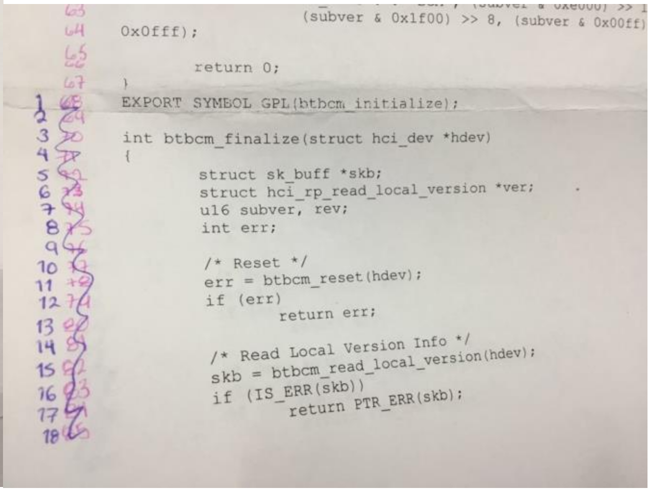
```
216          skb = btbcm_read_local_name(hdev);
217          if (IS_ERR(skb))
218                  return PTR_ERR(skb);
219
220          bt_dev_info(hdev, "%s", (char *)(skb->data + 1));
221          kfree_skb(skb);
222
223          return 0;
~~224~~ 225 }

1 static const struct {
2          u16 subver;
3          const char *name;
4 } bcm_uart_subver_table[] = {          /* 002.001.003 */
           { 0x4103, "BCM4330B1" },             /* 002.001.014 */
5          { 0x410e, "BCM43341B0"        },
6          { 0x4406, "BCM4324B3" },         /* 002.004.006 */
7          { 0x610c, "BCM4354"    },         /* 003.001.012 */
8 9        { 0x2209, "BCM43430A1"   },       /* 001.002.009 */
10         { 0x6119, "BCM4345C0" },         /* 003.001.025 */
11         { 0x230f, "BCM4356A2" },         /* 001.003.015 */
12         { }
13 };
14
15 int btbcm_initialize(struct hci_dev *hdev, char *fw_name, si
16 {
           u16 subver, rev;
17         const char *hw_name = NULL;
18         struct sk_buff *skb;
19         struct hci_rp read_local_version *ver;
```



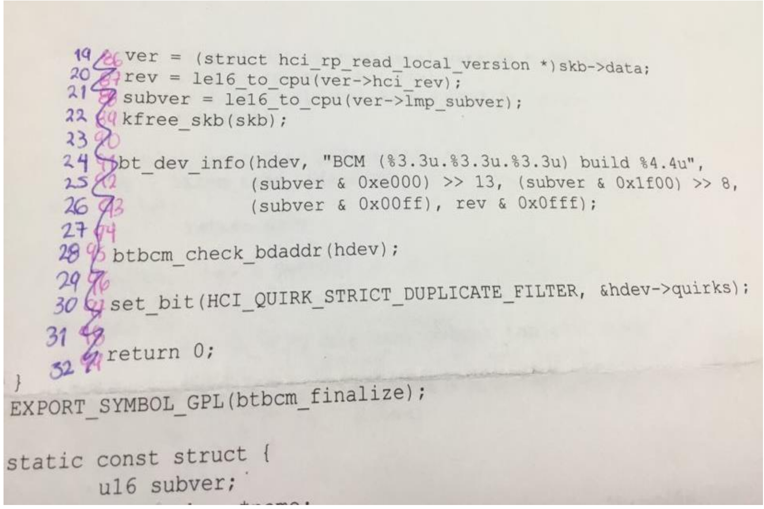
```
10
11          { 0x6119, "BCM4345C0" },          /* 001.002.009 */
12          { 0x230f, "BCM4356A2" },          /* 003.001.025 */
            { }                               /* 001.003.015 */
13 };
14
15 int btbcm_initialize(struct hci_dev *hdev, char *fw_name, size_t len)
16 {
17          u16 subver, rev;
18          const char *hw_name = NULL;
19          struct sk_buff *skb;
20          struct hci_rp_read_local_version *ver;
21          int i, err;
22
23          /* Reset */
24          err = btbcm_reset(hdev);
25          if (err)
26                  return err;
27
28          /* Read Local Version Info */
29          skb = btbcm_read_local_version(hdev);
30          if (IS_ERR(skb))
31                  return PTR_ERR(skb);
92
33          ver = (struct hci_rp_read_local_version *)skb->data;
```



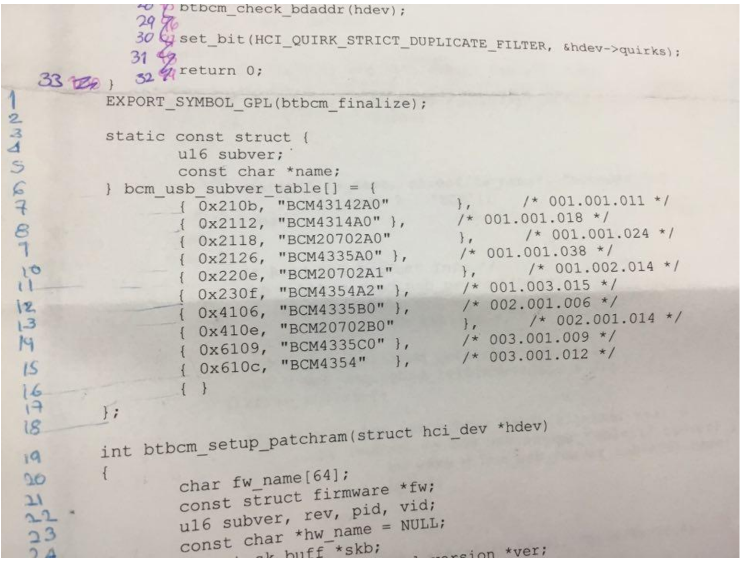
```
            if (err)
42                  return err;
43
44          switch ((rev & 0xf000) >> 12) {
45          case 0:
46          case 1:
47          case 2:
48          case 3:
                    for (i = 0; bcm_uart_subver_table[i].name; i++) {
49                          if (subver == bcm_uart_subver_table[i].subver) {
50                                  hw_name = bcm_uart_subver_table[i].name;
51                                  break;
52                          }
53                  }
54
                    snprintf(fw_name, len, "brcm/%s.hcd", hw_name ? :
55 "BCM");
56                  break;
57          default:
58                  return 0;
59          }
60
61          bt_dev_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",
                        hw_name ? : "BCM", (subver & 0xe000) >> 13,
62                      (subver & 0x1f00) >> 8, (subver & 0x00ff), rev &
63
64 0x0fff);
65
66          return 0;
67 }
68 EXPORT_SYMBOL_GPL(btbcm_initialize);
```



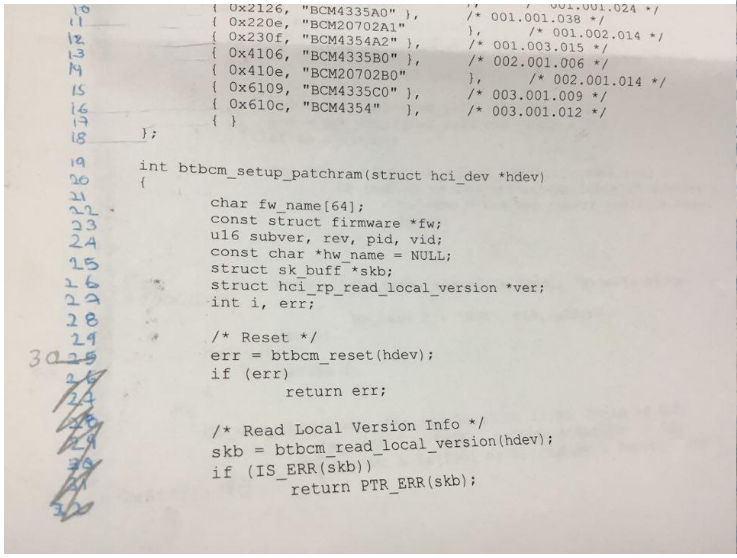
```
63                                  (subver & 0x1f00) >> 8, (subver & 0x00ff)
64   0x0fff);
65
66           return 0;
67   }
1 ~~68~~   EXPORT_SYMBOL_GPL(btbcm_initialize);
2 ~~69~~
3 ~~70~~   int btbcm_finalize(struct hci_dev *hdev)
4 ~~71~~   {
5 ~~72~~           struct sk_buff *skb;
6 ~~73~~           struct hci_rp_read_local_version *ver;
7 ~~74~~           u16 subver, rev;
8 ~~75~~           int err;
9 ~~76~~
10 ~~77~~          /* Reset */
11 ~~78~~          err = btbcm_reset(hdev);
12 ~~79~~          if (err)
13 ~~80~~                  return err;
14 ~~81~~
15 ~~82~~          /* Read Local Version Info */
16 ~~83~~          skb = btbcm_read_local_version(hdev);
17 ~~84~~          if (IS_ERR(skb))
18 ~~85~~                  return PTR_ERR(skb);
```



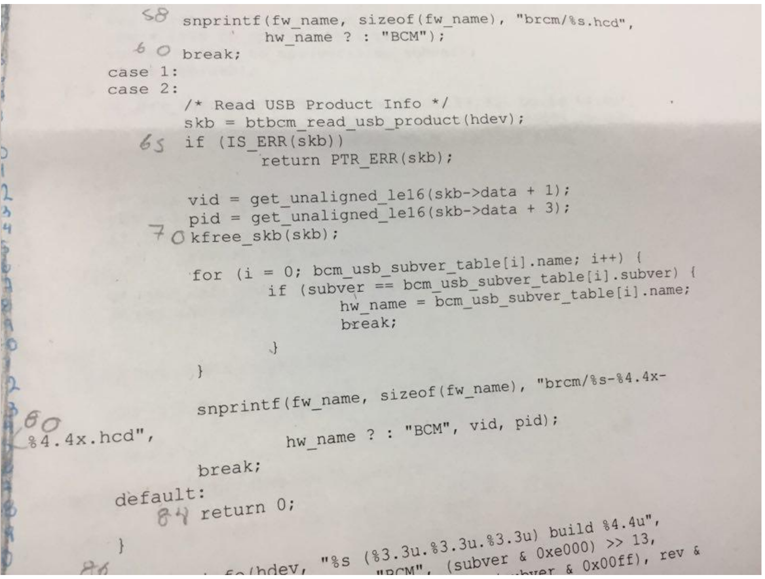
```
        ver = (struct hci_rp_read_local_version *)skb->data;
        rev = le16_to_cpu(ver->hci_rev);
        subver = le16_to_cpu(ver->lmp_subver);
        kfree_skb(skb);

        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;
}
EXPORT_SYMBOL_GPL(btbcm_finalize);

static const struct {
        u16 subver;
```



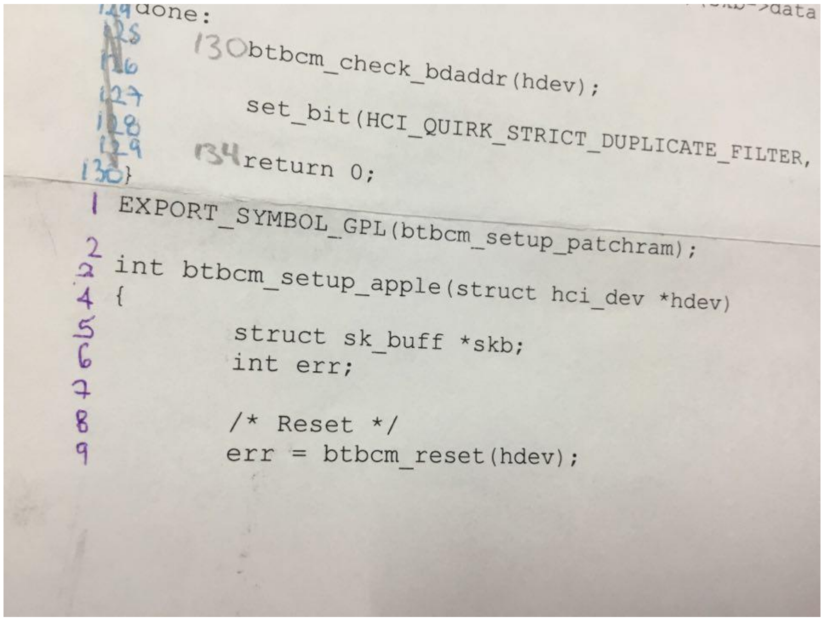
```
        btbcm_check_bdaddr(hdev);

        set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER, &hdev->quirks);

        return 0;
}
EXPORT_SYMBOL_GPL(btbcm_finalize);

static const struct {
        u16 subver;
        const char *name;
} bcm_usb_subver_table[] = {
        { 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 */
        { }
};

int btbcm_setup_patchram(struct hci_dev *hdev)
{
        char fw_name[64];
        const struct firmware *fw;
        u16 subver, rev, pid, vid;
        const char *hw_name = NULL;
```



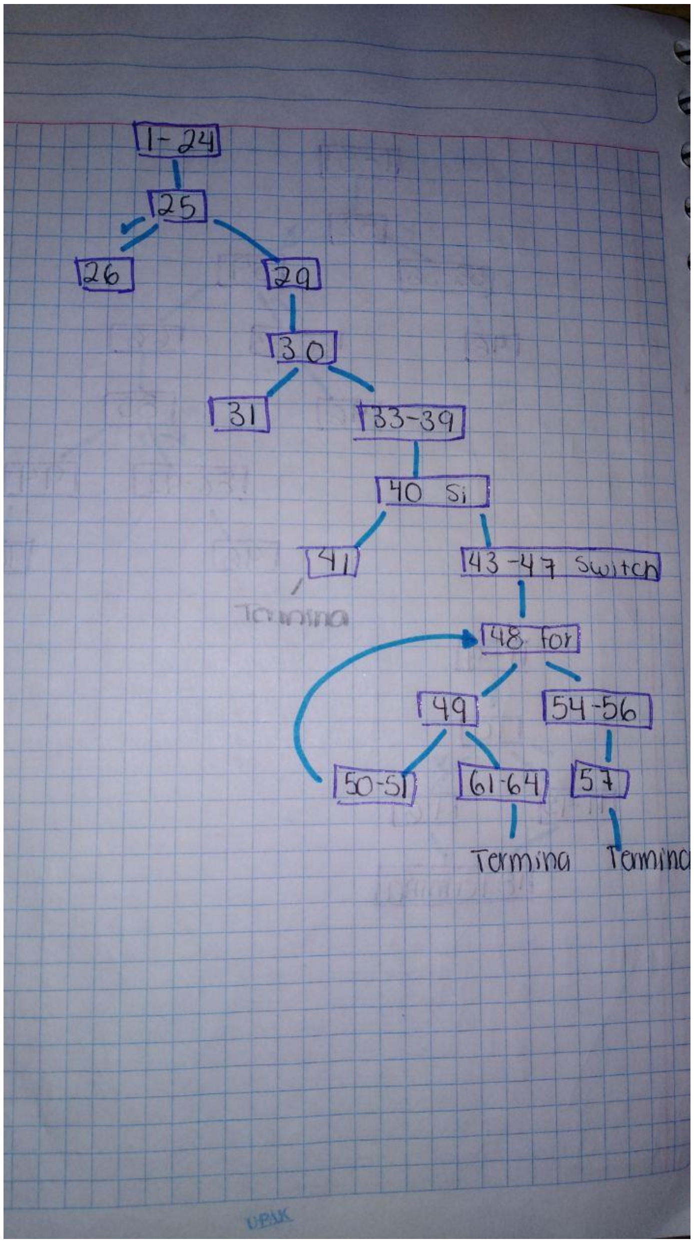
```
        { 0x2126, "BCM4335A0" },        /* 001.001.038 */
        { 0x220e, "BCM20702A1"          },      /* 001.002.014 */
        { 0x230f, "BCM4354A2" },        /* 001.003.015 */
        { 0x4106, "BCM4335B0" },        /* 002.001.006 */
        { 0x410e, "BCM20702B0"          },      /* 002.001.014 */
        { 0x6109, "BCM4335C0" },        /* 003.001.009 */
        { 0x610c, "BCM4354"   },        /* 003.001.012 */
        { }
};

int btbcm_setup_patchram(struct hci_dev *hdev)
{
        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;

        /* Reset */
        err = btbcm_reset(hdev);
        if (err)
                return err;

        /* Read Local Version Info */
        skb = btbcm_read_local_version(hdev);
        if (IS_ERR(skb))
                return PTR_ERR(skb);
```

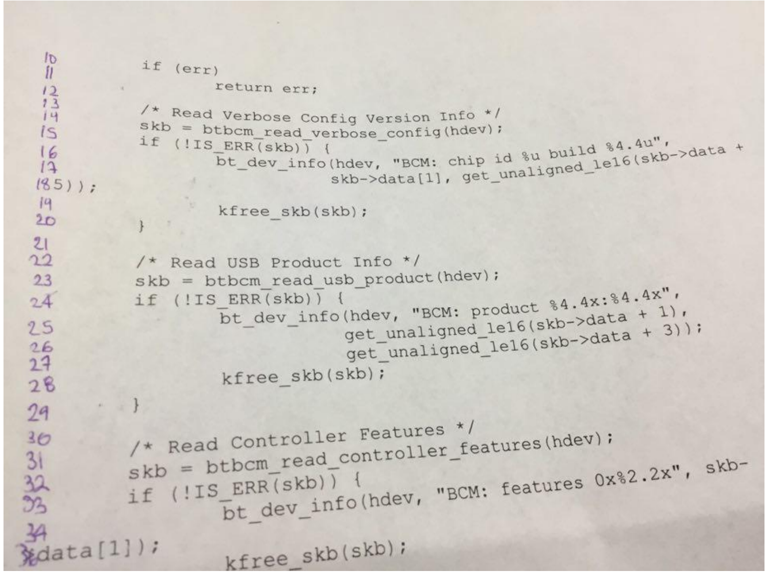


```
                snprintf(fw_name, sizeof(fw_name), "brcm/%s.hcd",
                         hw_name ? : "BCM");
                break;
        case 1:
        case 2:
                /* Read USB Product Info */
                skb = btbcm_read_usb_product(hdev);
                if (IS_ERR(skb))
                        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;
                                break;
                        }
                }

                snprintf(fw_name, sizeof(fw_name), "brcm/%s-%4.4x-
%4.4x.hcd",
                         hw_name ? : "BCM", vid, pid);
                break;
        default:
                return 0;
        }

                fo(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",
                         "BCM", (subver & 0xe000) >> 13,
                         bver & 0x00ff), rev &
```

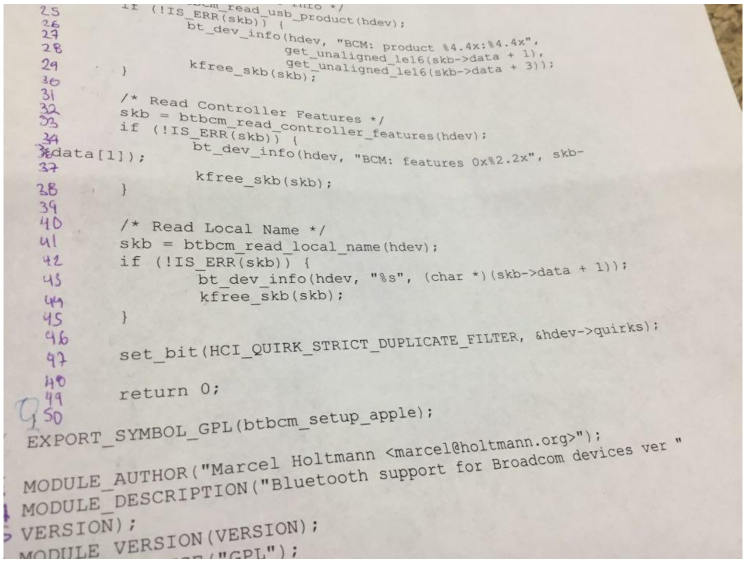


```
done:
        btbcm_check_bdaddr(hdev);

        set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER,

        return 0;
}
EXPORT_SYMBOL_GPL(btbcm_setup_patchram);

int btbcm_setup_apple(struct hci_dev *hdev)
{
        struct sk_buff *skb;
        int err;

        /* Reset */
        err = btbcm_reset(hdev);
```





```
10          if (err)
11                  return err;
12
13          /* Read Verbose Config Version Info */
14          skb = btbcm_read_verbose_config(hdev);
15          if (!IS_ERR(skb)) {
16                  bt_dev_info(hdev, "BCM: chip id %u build %4.4u",
17                                  skb->data[1], get_unaligned_le16(skb->data +
185));
19                  kfree_skb(skb);
20          }
21
22          /* Read USB Product Info */
23          skb = btbcm_read_usb_product(hdev);
24          if (!IS_ERR(skb)) {
25                  bt_dev_info(hdev, "BCM: product %4.4x:%4.4x",
26                                  get_unaligned_le16(skb->data + 1),
27                                  get_unaligned_le16(skb->data + 3));
28                  kfree_skb(skb);
29          }
30
31          /* Read Controller Features */
32          skb = btbcm_read_controller_features(hdev);
33          if (!IS_ERR(skb)) {
                    bt_dev_info(hdev, "BCM: features 0x%2.2x", skb-
34
35data[1]);
                    kfree_skb(skb);
```



```
25          if (!IS_ERR(skb)) {   ...cm_read_usb_product(hdev);
26                  bt_dev_info(hdev, "BCM: product %4.4x:%4.4x",
27                                  get_unaligned_le16(skb->data + 1),
28                                  get_unaligned_le16(skb->data + 3));
29          }       kfree_skb(skb);
30
31          /* Read Controller Features */
32          skb = btbcm_read_controller_features(hdev);
33          if (!IS_ERR(skb)) {
34                  bt_dev_info(hdev, "BCM: features 0x%2.2x", skb-
35data[1]);
37                  kfree_skb(skb);
38          }
39
40          /* Read Local Name */
41          skb = btbcm_read_local_name(hdev);
42          if (!IS_ERR(skb)) {
43                  bt_dev_info(hdev, "%s", (char *)(skb->data + 1));
44                  kfree_skb(skb);
45          }
46
47          set_bit(HCI_QUIRK_STRICT_DUPLICATE_FILTER, &hdev->quirks);
48
49          return 0;
50  }
EXPORT_SYMBOL_GPL(btbcm_setup_apple);

MODULE_AUTHOR("Marcel Holtmann <marcel@holtmann.org>");
MODULE_DESCRIPTION("Bluetooth support for Broadcom devices ver "
VERSION);
MODULE_VERSION(VERSION);
MODULE_LICENSE("GPL");
```