

Compiling a 2.4 driver under 2.6

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-09/9177.html>

From: Brian McGrew (*Brian_at_doubledimension.com*)

Date: 09/30/04

Date: Thu, 30 Sep 2004 13:31:50 -0700
To: <linux-kernel@vger.kernel.org>

Hello all:

We've got a device driver that we currently use under the 2.4.20 kernel and it works. I'm trying to get the same driver compiled up under the 2.6.8 (Fedora Core 2) kernel and getting a lot of errors. I'm no expert software engineer or kernel guy, so any help would be great. I've included the error output as well as the drive source.

Thanks,

-brian

====[Begin Error Output]====

```
119_ mk
/usr/bin/gcc -D__SMP__ -DLINUX -DUNIXHOST -DMODULE -DMODVERSIONS
-DEXPORT_SYMTAB -g -O1 -I../..../include/localinc -I../..../include/libinc -I..
-DCONFIG_MODVERSIONS -E -D__GENKSYMS__ ibb.c | \
  /sbin/genksyms -k 2.6.8-1.521smp > ../ibb.ver
unrecognised kernel version : 2.6.8-1.521smp
In file included from ibb.c:16:
/usr/include/linux/config.h:5:2: #error Incorrectly using glibc headers for a kernel module
ibb.c:19:2: #error "This driver needs PCI support"
ibb.c:46:42: linux/slab.h: No such file or directory
ibb.c:48:53: linux/mm.h: No such file or directory
make: *** [ibb.ver] Error 255
120_
```

====[End Error Output]====

====[Begin ibb.c code]====

```
static const char *ibb_c = "$Id: ibb.c,v 1.25 2003/11/04 19:09:12 celeste Exp $(c) MVP ";
```

```
#ifndef __KERNEL__
#define __KERNEL__
#endif
```

```
#ifndef MODULE
#define MODULE
```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
#endif

/*
 * Force CONFIG_PCI to be defined in the kernel. If you
 * hit an error here, you may need to play with /usr/include/linux
 */

#include <linux/config.h> /* access CONFIG_PCI macro */

#ifndef CONFIG_PCI
#error "This driver needs PCI support"
#endif

#if defined(CONFIG_MODVERSIONS) && !defined(MODVERSIONS)
# define MODVERSIONS /* force it on */
#endif

#if defined(MODVERSIONS) && !defined(__GENKSYMS__)
#include <linux/modversions.h>
#include "ibb.ver"
#endif

#ifndef EXPORT_SYMTAB
# define EXPORT_SYMTAB /* need this one cause we export ibb_rtc_wakeup */
#endif

#include <linux/module.h>

void ibb_rtc_wakeup(void);

EXPORT_SYMBOL(ibb_rtc_wakeup);

/* license info */
MODULE_LICENSE("Proprietary");

#include <linux/module.h>
#include <linux/slab.h> /* kcalloc() */
#include <linux/pci.h>
#include <linux/mm.h> /* memory mapping */
#include <linux/types.h>
#ifdef CONFIG_DEVFS_FS
#include <linux/devfs_fs_kernel.h>
#endif

#include <linux/fs.h>
#include <asm/io.h> /* ioremap */
#include <linux/wrapper.h> /* mem_map_reserve */

#include "sysdep.h"
```

```

#include "dev/ibb.h"
#include "ibbconfig.h"
#include "ibbsoft.h"

/*
 * Split minors in two parts
 */
#define TYPE(dev) (MINOR(dev) >> 4) /* high nibble */
#define NUM(dev) (MINOR(dev) & 0xf) /* low nibble */

static void ibb_intr(int irq, void *dev_id, struct pt_regs *regs);

/*
 * array of devices
 */
static IbbSoftDev *IbbSoft[kMaxIbbs];

static const uint32_t kIoWaiting = 0x04; /* ioctl waiting for intr */
static const uint32_t MAG_FB_NUMBER = 0xCE1E57E;
static const uint16_t kIbbWakeup = 0x0fff;

/*
 * local static variable
 */

static u_int ibb_debug = 2;
#define dcmn_err(lvl, msg) { if ((lvl) < ibb_debug) printk msg; }

/*
 * clamp the max frame buffer size to the lowest available memory size
 */
typedef struct ibb_ishared_attr {
    u_int32_t frame_buffer_size;
    int32_t dev_instance[kMaxIbbs];
} ibb_ishared_attr;

static ibb_ishared_attr ishared;

static void
lock_ibb(IbbSoftDev *ibb_sp, u_long *flags, const char * where)
{
    dcmn_err(70,("<1>" "lock_ibb(%d): %s:\n",ibb_sp->dev_num,where) );
    spin_lock_irqsave(&ibb_sp->mutex,*flags);
    dcmn_err(70,("<1>" "lock_ibb(%d): Done %s:\n",ibb_sp->dev_num,where) );
}

static void
unlock_ibb(IbbSoftDev *ibb_sp, u_long *flags, const char * where)
{
    dcmn_err(70,("<1>" "unlock_ibb(%d): %s:\n",ibb_sp->dev_num,where) );
    spin_unlock_irqrestore(&ibb_sp->mutex,*flags);
}

```

```

}

static void
init_ibb_soft_state(void)
{
    int i;
    dcmn_err(70,("<1>" "init_ibb_soft_state:\n" ));
    for( i = 0; i < kMaxIbbs; i ++)
        IbbSoft[i] = NULL;
}

static int
alloc_ibb_soft_state(void)
{
    int i;
    dcmn_err(70,("<1>" "alloc_ibb_soft_state:\n" ));
    for( i = 0; i < kMaxIbbs; i ++)
    {
        if(IbbSoft[i] == NULL)
        {
            IbbSoft[i] = kmalloc(sizeof(IbbSoftDev), GFP_KERNEL);
            if(IbbSoft[i] == NULL)
            {
                dcmn_err(1,("<1>" "alloc_ibb_soft_state: out of memory.\n" ));
                return -1;
            }
            dcmn_err(70,("<1>" "alloc_ibb_soft_state: return %d\n",i) );
            return (i);
        }
    }
    return -1;
}

/*
 * There may be a better way to do this!
 *
 * I need to make sure the data I initialized in ibb_probe()
 * for device "X" is the same data I use in ibb_open().
 *
 * I could use the minor numbers, but I have to trust that
 * the 'load' script for the ibb device is written correctly,
 * since that's where minor numbers are assigned.
 *
 * So I'm using the major numbers. It's a little slower, since
 * major numbers tend to be >200 I'm not just going to use
 * them as indexes into my array. Instead I'm going though
 * all the devices looking for the matching major.
 *
 * My concern, frankly, is that no other drivers seem to be
 * using this method. But it seems the only foolproof way
 * to keep the data with the device

```

```

*/
static int
find_ibb_soft_state(int ibb_major)
{
    int i;
    dcmn_err(70,("<1>" "find_ibb_soft_state: major %d\n",ibb_major) );
    for( i = 0; i < kMaxIbbs; i ++)
    {
        if(IbbSoft[i] != NULL)
        {
            if( ibb_major == IbbSoft[i]->ibb_major )
            {
                dcmn_err(70,("<1>" "find_ibb_soft_state: return %d\n",i) );
                return (i);
            }
        }
    }
    dcmn_err(1,("<1>"
        "find_ibb_soft_state: Can't find matching soft_state %d!\n",ibb_major));
    return -1;
}

static void
free_ibb_soft_state(int i)
{
    if(IbbSoft[i] == NULL)
        return;
    dcmn_err(70,("<1>" "free_ibb_soft_state(%d):\n",IbbSoft[i]->dev_num) );
    kfree(IbbSoft[i]);
    IbbSoft[i] = NULL;
}

ssize_t
ibb_read(struct file *filp, char * buf, size_t count, loff_t *f_pos)
{
    dcmn_err(1,("<1>" "ibb_read:\n") );
    return -EINVAL;
}

ssize_t
ibb_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
    dcmn_err(1,("<1>" "ibb_write:\n") );
    return -EINVAL;
}

/*
 * IBBWAIT ioctl routine
 * wait for ushared->image_table_index to *exceed* value
 * passed in - (IBBWAIT,0) waits for snap 0 to be processed and

```

```

* index to advance to 1
*/
static inline int
ibb_wait(IbbSoftDev *ibb_sp,u_long arg)
{
    uint32_t table_index = (uint32_t)arg;
    u_long flags;
    int dev_num = ibb_sp->dev_num;
    int wq_ret = 0;

    if(ibb_sp->ushared == NULL)
        return EAGAIN;

    if(ibb_sp->ushared->image_table_index > table_index ) {
        dcmn_err(27, ("<1>" "ibb_ioctl(%d): IBBWAIT already got index %d\n",
            dev_num, ibb_sp->ushared->image_table_index) );
        return 0;
    }

    lock_ibb(ibb_sp,&flags,"IBBWAIT 0");
    ibb_sp->flags |= kIoWaiting;
    ibb_sp->wait_for = table_index;
    unlock_ibb(ibb_sp,&flags,"IBBWAIT 0");

    dcmn_err(27, ("<1>" "ibb_ioctl(%d): IBBWAIT wait for %u, %u\n",
        dev_num, ibb_sp->wait_for, ibb_sp->ushared->image_table_index) );

    wq_ret = wait_event_interruptible(ibb_sp->wq,
        (ibb_sp->ushared->image_table_index > table_index) );

    dcmn_err(27, ("<1>" "ibb_ioctl(%d): IBBWAIT got index %d\n",
        dev_num,ibb_sp->wait_for) );

    lock_ibb(ibb_sp,&flags,"IBBWAIT 1");
    ibb_sp->flags &= ~kIoWaiting;
    unlock_ibb(ibb_sp,&flags,"IBBWAIT 1");

    if (ibb_sp->wait_for == kIbbWakeup ) {
        dcmn_err(27, ("<1>" "ibb_ioctl(%d): IBBWAIT got Wakeup\n", dev_num) );
    }
    else if(ibb_sp->ushared->image_table_index > ibb_sp->wait_for) {
        dcmn_err(27, ("<1>" "ibb_ioctl(%d): IBBWAIT got index %d\n"
            ,dev_num,ibb_sp->wait_for) );
    }
    else {
        if(wq_ret != 0) {
            /* wait_event_interruptible() returned due to
            * a system kill()
            * this is a legitimate condition – return an
            * error code
            */

```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
    dcmn_err(0, ("<1>" "ibb_ioctl(%d): IBBWAIT signal intr\n",dev_num));
}
else {
    /* we really shouldn't get here */
    dcmn_err(0, ("<1>" "ibb_ioctl(%d): IBBWAIT Weird cond.\n",dev_num));
}
return -1; /* !! there's been a problem */
}
return 0;
}

int
ibb_ioctl(struct inode *inode, struct file *filp, u_int cmd, u_long arg)
{
    int retval = 0;
    IbbSoftDev *ibb_sp = filp->private_data; /* retrieve our info */
    u_long flags;

    if(ibb_sp == NULL)
    {
        dcmn_err(1,("<1>" "ibb_ioctl(%d): Soft Info Lost\n", ibb_sp->dev_num) );
        return -EFAULT;
    }
    switch (cmd)
    {
    case IBBWAIT:
        dcmn_err(27, ("<1>" "ibb_ioctl(%d): cmd IBBWAIT arg %lu\n",
                    ibb_sp->dev_num,arg) );
        retval = ibb_wait(ibb_sp, arg);
        break;
    case IBBTST:
        if(ibb_sp->image_table == NULL)
            return 0;
        dcmn_err(1, ("<1>" "ibb_ioctl(%d): table 0 %x \n",
                    ibb_sp->dev_num,ibb_sp->image_table[0] ) );
        dcmn_err(1, ("<1>" "ibb_ioctl(%d): table 1 %x \n",
                    ibb_sp->dev_num,ibb_sp->image_table[1] ) );
        break;
    case IBBWAKEUP:
        dcmn_err(30, ("<1>" "ibb_ioctl(%d): cmd IBBWAKEUP\n",ibb_sp->dev_num));
        retval = 0;
        if( ibb_sp->flags & kIoWaiting)
        {
            /* tell IBBWAIT that we got wakeup */
            lock_ibb(ibb_sp,&flags,"IBBWAKEUP");
            ibb_sp->wait_for = kIbbWakeup;
            unlock_ibb(ibb_sp,&flags,"IBBWAKEUP");
            wake_up_interruptible(&ibb_sp->wq);
        }
        break;
    case IBBFBACOUNT:
```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
dcmn_err(30, ("<1>" "ibb_ioctl(%d): cmd IBBFBACOUNT arg %lu\n",
            ibb_sp->dev_num, arg) );
lock_ibb(ibb_sp,&flags,"IBBFBACOUNT");
*(ibb_sp->fbac_virt) = (uint32_t)arg;
unlock_ibb(ibb_sp,&flags,"IBBFBACOUNT");
dcmn_err(37, ("<1>" "ibb_ioctl(%d): set fb count to %u\n",
            ibb_sp->dev_num,*(ibb_sp->fbac_virt) ) );
retval = 0;
break;
case IBBFSIZE:
dcmn_err(30, ("<1>" "ibb_ioctl(%d): cmd IBBFSIZE arg %lu\n",
            ibb_sp->dev_num,arg) );
lock_ibb(ibb_sp,&flags,"IBBFSIZE");
*(ibb_sp->fsize_virt) = (uint32_t)arg;
unlock_ibb(ibb_sp,&flags,"IBBFSIZE");
dcmn_err(37, ("<1>" "ibb_ioctl(%d): set frame size to %u\n",
            ibb_sp->dev_num,*(ibb_sp->fsize_virt) ) );
break;
case IBBSIZE:
dcmn_err(30,("<1>" "ibb_ioctl(%d): cmd is IBBSIZE: %d\n",
            ibb_sp->dev_num, ishared.frame_buffer_size) );
lock_ibb(ibb_sp,&flags,"IBBSIZE");
retval = put_user(ishared.frame_buffer_size, (u_long *)arg);
unlock_ibb(ibb_sp,&flags,"IBBSIZE");
break;
case IBBVERSION:
if (put_user(kIbbVersion, (u_long *)arg))
{
dcmn_err(1, ("<1>" "ibb_ioctl(%d): can't copy Version info %d\n",
            ibb_sp->dev_num,kIbbVersion) );
retval = -EFAULT;
}
dcmn_err(1, ("<1>" "ibb_ioctl(%d): Copied version info %d\n",
            ibb_sp->dev_num,kIbbVersion));

break;
case IBBCLIBB_ID:
if (put_user(ibb_sp->clibb_id, (short *)arg))
{
dcmn_err(1, ("<1>" "ibb_ioctl(%d): can't copy CLIBB ID %d\n",
            ibb_sp->dev_num,ibb_sp->clibb_id) );
retval = -EFAULT;
}
dcmn_err(1, ("<1>" "ibb_ioctl(%d): Copied CLIBB IDD %d\n",
            ibb_sp->dev_num,ibb_sp->clibb_id));

break;
case IBBHEIGHTGO:
if(ibb_sp->clibb_id != kClibb_HSC)
{
/* Our IBB isn't enabled to do height inspections */
```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
    dcmn_err(1, (<1>" "ibb_ioctl(%d): ERROR: No Height Sensor\n",
                ibb_sp->dev_num) );
    return -EFAULT;
}
ibb_sp->height_inspection = 1;
dcmn_err(1, (<1>" "ibb_ioctl(%d): Start Height Inspection %d\n",
            ibb_sp->dev_num,ibb_sp->height_inspection));

    break;
case IBBHEIGHTSTOP:
    ibb_sp->height_inspection = 0;
    dcmn_err(1, (<1>" "ibb_ioctl(%d): Stop Height Inspection %d\n",
                ibb_sp->dev_num,ibb_sp->height_inspection));

    break;
default:
    dcmn_err(1,(<1>" "ibb_ioctl(%d): unknown case: %d\n",
                ibb_sp->dev_num, cmd) );
    return -ENOTTY;
}

return retval;
}

int
ibb_open( struct inode *inode, struct file *filep )
{
    IbbSoftDev *ibb_sp; /* device information */
    int major = MAJOR(inode->i_rdev);
    int minor = MINOR(inode->i_rdev);
    int dev_num;

    dcmn_err(40, (<1>" "ibb_open: major %d minor %d\n",major, minor) );

    dev_num = find_ibb_soft_state(major);
    if(dev_num < 0)
    {
        dcmn_err(0, (<1>" "ibb_open: dev_num %d failed\n",dev_num) );
        return -1;
    }
    ibb_sp = IbbSoft[dev_num];
    /*
     * assign the private data so we can access it later
     * ( in read, write, ioctl, etc );
     */
    filep->private_data = ibb_sp;

    dcmn_err(40, (<1>" "ibb_open(%d): opened by %s pid %d PAGE_SIZE %#010lx\n",
                dev_num,current->comm, current->pid, PAGE_SIZE) );

    MOD_INC_USE_COUNT;
```

```

return 0; /* success */
}

int
ibb_close( struct inode *inode, struct file *file )
{
    IbbSoftDev *ibb_sp = file->private_data;

    MOD_DEC_USE_COUNT;
    dcmn_err(40, ("<1>" "ibb_close(%d): called\n",ibb_sp->dev_num) );
    return 0;
}

static void
free_ibb_usr_shared(IbbSoftDev *ibb_sp)
{
    u_long virt_addr,ushared_addr = (u_long)ibb_sp->ushared;

    /* unreserve all pages */
    for(virt_addr = ushared_addr;
        virt_addr < ushared_addr + PAGE_ALIGN(IBB_SHARED_SIZE);
        virt_addr += PAGE_SIZE)
    {
        mem_map_unreserve(virt_to_page(virt_addr));
    }
    if (ibb_sp->ushared)
        kfree(ibb_sp->ushared);
    ibb_sp->ushared = NULL;
}

static int
alloc_ibb_usr_shared(IbbSoftDev *ibb_sp)
{
    u_long virt_addr,ushared_addr;
    /*
     * From Linux Device Drivers – memory that is going to
     * be mmaped must be PAGE_SIZE grained. Since we do mmap
     * ushared to user space, we need to allocated it in
     * PAGE_SIZE chunks.
     */
    int size = PAGE_ALIGN(IBB_SHARED_SIZE);
    ibb_sp->ushared = (IbbUserShared *)kmalloc(size, GFP_KERNEL);
    if(ibb_sp->ushared == NULL)
    {
        dcmn_err(1,("<1>" "alloc_ibb_usr_shared(%d): out of memory.\n",
                    ibb_sp->dev_num) );

        return -1;
    }
    ushared_addr = (u_long) ibb_sp->ushared;
    for (virt_addr = ushared_addr; virt_addr < ushared_addr + size;
        virt_addr += PAGE_SIZE)

```

```

{
    /* reserve all pages to make them remappable */
    mem_map_reserve(virt_to_page(virt_addr));
}
ibb_sp->ushared->badsnap = 0;
ibb_sp->ushared->image_table_index = 0;
return 0;
}

static void
free_ibb_image_table_mem(IbbSoftDev *ibb_sp)
{
    uint32_t *virt_addr;
    dcmn_err(1,("<1>" "free_ibb_image_table_mem(%d): Free size %d.\n",
                ibb_sp->dev_num,ibb_sp->image_table_size) );

    /* unreserve all pages */
    for( virt_addr = ibb_sp->image_table;
        virt_addr < ibb_sp->image_table + ibb_sp->image_table_size;
        virt_addr += PAGE_SIZE)
    {
        mem_map_unreserve(virt_to_page(virt_addr));
    }
    if (ibb_sp->image_table)
    {
        kfree(ibb_sp->image_table);
    }
    ibb_sp->image_table = NULL;
    ibb_sp->image_table_size = 0;
}

static int
alloc_ibb_image_table_mem(IbbSoftDev *ibb_sp, int size)
{
    uint32_t *virt_addr;
    u_int table_size = PAGE_ALIGN(size);
    dcmn_err(1,("<1>" "alloc_ibb_image_table_mem(%d): Alloc size %d.\n",
                ibb_sp->dev_num,table_size) );

    /*
     * get a memory area with kmalloc and aligned it to a page. This area
     * will be physically contiguous
     */
    ibb_sp->image_table = (uint32_t *)kmalloc(table_size, GFP_KERNEL);
    if(ibb_sp->image_table == NULL)
    {
        dcmn_err(1,("<1>" "alloc_ibb_image_table_mem(%d): out of memory.\n",
                    ibb_sp->dev_num) );
        ibb_sp->image_table_size = 0;
        return -1;
    }
    for (virt_addr = ibb_sp->image_table;

```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
    virt_addr < ibb_sp->image_table + table_size;
    virt_addr += PAGE_SIZE)
{
    /* reserve all pages to make them remappable */
    mem_map_reserve(virt_to_page(virt_addr));
}
ibb_sp->image_table_size = table_size;
return 0;
}

static int
ibb_map_one(IbbSoftDev *ibb_sp, u_long phys, u_long size, uint32_t **virt, const char *what)
{
    struct resource *res;

    if((res = request_mem_region(phys,size,ibb_sp->devname)) == NULL)
    {
        dcmn_err(0, (<1>"
            "ibb_map_one(%d): can't allocate %s at %010lx %#010lx %s\n",
                ibb_sp->dev_num,what,phys,size,ibb_sp->devname) );
        return -1;
    }
    else
    {
        *virt = ioremap_nocache(phys,size);
        if (!virt)
        {
            dcmn_err(0, (<1>" "ibb_map_one(%d): Error in ioremap\n",
                ibb_sp->dev_num));
            release_mem_region(phys,size);
            return -1;
        }
    }
    dcmn_err(40, (<1>" "ibb_map_one(%d): Successful map %s\n",
        ibb_sp->dev_num,what));
    return 0;
}

static void
free_all_mappings(IbbSoftDev *ibb_sp)
{
    if(ibb_sp->creg_virt != NULL)
    {
        iounmap(ibb_sp->creg_virt);
        ibb_sp->creg_virt = NULL;
    }
    /* we can call release_mem_region() without checking anything -
    * if the region hasn't been requested, the function will
    * just exit with an error message.
    * ( unlike iounmap() )
    */
}
```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
release_mem_region(ibt_sp->creg_phys,ibt_sp->creg_size);

if(ibt_sp->height_virt != NULL)
{
    iounmap(ibt_sp->height_virt);
    ibt_sp->height_virt = NULL;
}
release_mem_region(ibt_sp->height_phys,ibt_sp->height_size);

if(ibt_sp->csram_virt != NULL)
{
    iounmap(ibt_sp->csram_virt);
    ibt_sp->csram_virt = NULL;
}
release_mem_region(ibt_sp->csram_phys,ibt_sp->csram_size);

if(ibt_sp->rsram_virt != NULL)
{
    iounmap(ibt_sp->rsram_virt);
    ibt_sp->rsram_virt = NULL;
}
release_mem_region(ibt_sp->rsram_phys,ibt_sp->rsram_size);

if(ibt_sp->fbac_virt != NULL)
{
    iounmap(ibt_sp->fbac_virt);
    ibt_sp->fbac_virt = NULL;
}
release_mem_region(ibt_sp->fbac_phys,ibt_sp->fbac_size);

if(ibt_sp->fsize_virt != NULL)
{
    iounmap(ibt_sp->fsize_virt);
    ibt_sp->fsize_virt = NULL;
}
release_mem_region(ibt_sp->fsize_phys,ibt_sp->fsize_size);

if(ibt_sp->fb0_virt != NULL)
{
    iounmap(ibt_sp->fb0_virt);
    ibt_sp->fb0_virt = NULL;
}
release_mem_region(ibt_sp->fb0_phys,ibt_sp->fb0_size);

if(ibt_sp->fb1_virt != NULL)
{
    iounmap(ibt_sp->fb1_virt);
    ibt_sp->fb1_virt = NULL;
}
release_mem_region(ibt_sp->fb1_phys,ibt_sp->fb1_size);
```

Linux–Kernel: Compiling a 2.4 driver under 2.6

```
if(ibt_sp->image_table != NULL)
    free_ibb_image_table_mem(ibt_sp);

if(ibt_sp->ushared != NULL)
    free_ibb_usr_shared(ibt_sp);
}

static int
ibt_map_frame_buffer(IbbSoftDev *ibt_sp, u_long phys, ulong *size, uint32_t **virt, const char *what)
{
    char whatsize[100];
    long check;
    uint32_t *check_addr;

    sprintf(whatsize, "%s 64", what);

    *size = IBB_FB_64_MG;

    if(ibt_map_one(ibt_sp, phys, *size, virt, whatsize) < 0)
    {
        return -1;
    }

    check = MAG_FB_NUMBER;
    check_addr = *virt;
    writel(check, check_addr);

    dcmn_err(45, (<1> "ibt_map_frame_buffer(%d): %s put %lX at addr %p\n",
                ibt_sp->dev_num, whatsize, check, check_addr));
    check = readl(check_addr);
    dcmn_err(45, (<1> "ibt_map_frame_buffer(%d): %s read %lX at addr %p\n",
                ibt_sp->dev_num, whatsize, check, check_addr));

    check_addr = (*virt) + (IBB_FB_32_MG / 4);
    check = readl(check_addr);
    dcmn_err(45, (<1> "ibt_map_frame_buffer(%d): %s read %lX at addr %p\n",
                ibt_sp->dev_num, whatsize, check, check_addr));
    if(check == MAG_FB_NUMBER)
    {
        /* we only have 32 MB, but we've mapped 64 – try again */
        iounmap(*virt);
        (*virt) = NULL;
        release_mem_region(phys, *size);

        sprintf(whatsize, "%s 32", what);
        *size = IBB_FB_32_MG;
        if(ibt_map_one(ibt_sp, phys, *size, virt, whatsize) < 0)
        {
            return -1;
        }
        ishared.frame_buffer_size = IBB_FB_32_MG;
    }
}
```

```

    }
    return 0;
}

static int
ibb_do_all_mappings(IbbSoftDev *ibb_sp)
{
    /*
     * Initialize here so we can call free_all_mappings without
     * freeing unallocated mem
     */
    ibb_sp->creg_virt = NULL;
    ibb_sp->height_virt = NULL;
    ibb_sp->csram_virt = NULL;
    ibb_sp->rsram_virt = NULL;
    ibb_sp->fbac_virt = NULL;
    ibb_sp->fsize_virt = NULL;
    ibb_sp->fb0_virt = NULL;
    ibb_sp->fb1_virt = NULL;
    ibb_sp->image_table = NULL;
    ibb_sp->image_table_size = 0;

    /* Map Control Register */
    ibb_sp->creg_phys = ibb_sp->iobase + IBB_CONTROL_OFF;
    ibb_sp->creg_size = IBB_CONTROL_SIZE;
    if(ibb_map_one(ibb_sp,ibb_sp->creg_phys,ibb_sp->creg_size,
                  &ibb_sp->creg_virt,"Control Register") < 0)
        return -1;

    /* Map Height Sensor Register */
    ibb_sp->height_phys = ibb_sp->iobase + IBB_HEIGHT_OFF;
    ibb_sp->height_size = IBB_HEIGHT_SIZE;
    if(ibb_map_one(ibb_sp,ibb_sp->height_phys,ibb_sp->height_size,
                  &ibb_sp->height_virt,"Height Sensor") < 0)
        return -1;

    /* Map Col Sram */
    ibb_sp->csram_phys = ibb_sp->iobase + IBB_CS RAM_OFF;
    ibb_sp->csram_size = IBB_CS RAM_SIZE;
    if(ibb_map_one(ibb_sp,ibb_sp->csram_phys,ibb_sp->csram_size,
                  &ibb_sp->csram_virt,"Col Sram") < 0)
    {
        free_all_mappings(ibb_sp);
        return -1;
    }

    /* Map Row Sram */
    ibb_sp->rsram_phys = ibb_sp->iobase + IBB_RS RAM_OFF;
    ibb_sp->rsram_size = IBB_RS RAM_SIZE;

```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
if(ibt_map_one(ibt_sp, ibt_sp->rsram_phys, ibt_sp->rsram_size,
               &ibt_sp->rsram_virt, "Row Sram") < 0)
{
    free_all_mappings(ibt_sp);
    return -1;
}

/* map FB Address Counter */
ibt_sp->fbac_phys = ibt_sp->iobase + IBB_FBACOUNT_OFF;
ibt_sp->fbac_size = IBB_FBACOUNT_SIZE;
if(ibt_map_one(ibt_sp,ibt_sp->fbac_phys,ibt_sp->fbac_size,
               &ibt_sp->fbac_virt,"FB Addr Count") < 0)
{
    free_all_mappings(ibt_sp);
    return -1;
}

/* map Frame Size Register */
ibt_sp->fsize_phys = ibt_sp->iobase + IBB_FSIZE_OFF;
ibt_sp->fsize_size = IBB_FSIZE_SIZE;
if(ibt_map_one(ibt_sp,ibt_sp->fsize_phys,ibt_sp->fsize_size,
               &ibt_sp->fsize_virt,"Frame Size Reg") < 0)
{
    free_all_mappings(ibt_sp);
    return -1;
}

/* map Frame Buffers - now it gets a little trickier */
ibt_sp->fb0_phys = ibt_sp->iobase + IBB_FB0_OFF;
ibt_sp->fb0_size = 0;
if(ibt_map_frame_buffer(ibt_sp,ibt_sp->fb0_phys,
                        &ibt_sp->fb0_size,&ibt_sp->fb0_virt,"Frame Buffer 0") < 0)
{
    free_all_mappings(ibt_sp);
    return -1;
}

dcmn_err(40,("<1>" "ibt_do_all_mappings(%d): fb0 Mapped Size %lx\n",
            ibt_sp->dev_num,ibt_sp->fb0_size) );

ibt_sp->fb1_phys = ibt_sp->iobase + IBB_FB1_OFF;
ibt_sp->fb1_size = 0;
if(ibt_map_frame_buffer(ibt_sp,ibt_sp->fb1_phys,
                        &ibt_sp->fb1_size,&ibt_sp->fb1_virt,"Frame Buffer 1") < 0)
{
    free_all_mappings(ibt_sp);
    return -1;
}

dcmn_err(40,("<1>" "ibt_do_all_mappings(%d): fb1 Mapped Size %lx\n",
            ibt_sp->dev_num,ibt_sp->fb1_size) );
```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```

#ifdef IMAGETABLEALLOC
    /* two pages of memory should always be enough ? */
    ibb_sp->image_table_mem = NULL;
    if(alloc_ibb_image_table_mem(ibt_sp, PAGE_SIZE * 2) < 0)
    {
        free_all_mappings(ibt_sp);
        return -1;
    }
    ibb_sp->image_table_size = PAGE_SIZE * 2;

    dcmn_err(40,("<1>" "ibt_do_all_mappings(%d): Image Table Mapped Size %lx\n",
                ibb_sp->dev_num,PAGE_SIZE * 2) );
#endif

    if(alloc_ibb_usr_shared(ibt_sp) < 0)
    {
        free_all_mappings(ibt_sp);
        return -1;
    }

    dcmn_err(40,("<1>" "ibt_do_all_mappings(%d): IbbUserShared %lx\n",
                ibb_sp->dev_num,IBB_SHARED_SIZE) );
    return 0;
}

int
ibt_mmap( struct file *filep, struct vm_area_struct *vma )
{
    IbbSoftDev *ibt_sp = filep->private_data;

    u_long offset = vma->vm_pgoff << PAGE_SHIFT;
    u_long vsize = vma->vm_end - vma->vm_start;

    if( offset > __pa(high_memory) || (filep->f_flags & O_SYNC) )
        vma->vm_flags |= VM_IO;
    vma->vm_flags |= VM_RESERVED;

    dcmn_err(90, ("<1>"
"ibt_mmap(%d): st %#010lx offset %#010lx(%#010lx) size %#010lx (end %#010lx)\n",
                ibb_sp->dev_num,vma->vm_start,ibt_sp->iobase + offset,
                offset,vsize,vma->vm_end) );
    dcmn_err(30, ("<1>" "ibt_mmap(%d): opened by %s pid %d\n",
                ibb_sp->dev_num,current->comm, current->pid) );

    if(offset == IBB_IMAGE_ADDR)
    {
        if(ibt_sp->image_table != NULL && vsize > ibb_sp->image_table_size) {
            /* release the current memory and allocate new memory
             * below */
            dcmn_err(1, ("<1>" "ibt_mmap(%d): Free Image Table\n",
                        ibb_sp->dev_num) );
        }
    }
}

```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```

    free_ibt_image_table_mem(ibt_sp);
}
if(ibt_sp->image_table == NULL) {
    dcmn_err(1, (<1>" "ibt_mmap(%d): Alloc New Image Table\n",
                ibt_sp->dev_num) );
    alloc_ibt_image_table_mem(ibt_sp,vsize);
}
if(ibt_sp->image_table != NULL && vsize <= ibt_sp->image_table_size)
{
    if (remap_page_range(vma->vm_start,
        virt_to_phys((void*)((u_long)ibt_sp->image_table)),
        vsize, PAGE_SHARED))
    {
        dcmn_err(1, (<1>" "ibt_mmap(%d): image table remap failed\n",
                    ibt_sp->dev_num) );
        return -EAGAIN;
    }
}
else
{
    dcmn_err(1, (<1>"
        "ibt_mmap(%d): image table params failed vs %d is %d\n",
                ibt_sp->dev_num,vsize,ibt_sp->image_table_size) );
    return -EAGAIN;
}
}
else if(offset == IBB_SHARED_ADDR)
{
    if(ibt_sp->ushared != NULL && vsize <= PAGE_ALIGN(IBB_SHARED_SIZE))
    {
        if (remap_page_range(vma->vm_start,
            virt_to_phys((void*)((u_long)ibt_sp->ushared)),
            vsize, PAGE_SHARED))
        {
            dcmn_err(1, (<1>" "ibt_mmap(%d): ushared remap failed\n",
                        ibt_sp->dev_num) );
            return -EAGAIN;
        }
    }
}
else
{
    dcmn_err(1, (<1>" "ibt_mmap(%d): ushared mmap failed\n",
                ibt_sp->dev_num));
    return -EAGAIN;
}
}
else if(offset == IBB_CONTROL_OFF && vsize == PAGE_SIZE)
{
    if(remap_page_range(vma->vm_start, ibt_sp->creg_phys,
        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}

```

```

}
else if(offset == IBB_CS RAM_OFF && vsize == IBB_CS RAM_SIZE)
{
    if(remap_page_range(vma->vm_start, ibb_sp->csram_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else if(offset == IBB_RS RAM_OFF && vsize == IBB_RS RAM_SIZE)
{
    if(remap_page_range(vma->vm_start, ibb_sp->rsram_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else if(offset == IBB_FBACOUNT_OFF && vsize == PAGE_SIZE)
{
    if(remap_page_range(vma->vm_start, ibb_sp->fbac_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else if(offset == IBB_FSIZE_OFF && vsize == PAGE_SIZE)
{
    if(remap_page_range(vma->vm_start, ibb_sp->fsize_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else if(offset == IBB_FB0_OFF &&
        (vsize == IBB_FB_32_MG || vsize == IBB_FB_64_MG))
{
    if(remap_page_range(vma->vm_start, ibb_sp->fb0_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else if(offset == IBB_FB1_OFF &&
        (vsize == IBB_FB_32_MG || vsize == IBB_FB_64_MG))
{
    if(remap_page_range(vma->vm_start, ibb_sp->fb1_phys,
                        vsize, vma->vm_page_prot) )
        return -EAGAIN;
}
else
    return -EAGAIN;
return 0;
}

struct file_operations ibb_fops = {
    read: ibb_read,
    write: ibb_write,
    ioctl: ibb_ioctl,
    mmap: ibb_mmap,
    open: ibb_open,
    release: ibb_close,
}

```

```

};

#ifdef MISCLOAD
struct miscdevice ibb_misc_device = {
    -1, "ibb", &ibb_fops
};
#endif

static int __initdata
ibb_probe(struct pci_dev *pdev, const struct pci_device_id *pci_id)
{
    IbbSoftDev *ibb_sp;
    int result;
    int dev_num;
    int ibb_major;
    u16 device_id;
    u8 rev_id = 0;
    uint16_t subsystem_id = 0;
    char revid = ' ';
    int i = 0;
    int res;

    dcmn_err(40,("<1>" "ibb_probe: enter\n"))

    if (pci_enable_device(pdev))
        return -ENODEV;

    result = 0;

    /* stealing code from cciss.c – thanks compaq! */
    dcmn_err(0, ("<1>"
        "ibb_probe: IBB Device 0x%08x has been found @bus %d dev %d func %d\n",
        pdev->device, pdev->bus->number, PCI_SLOT(pdev->devfn),
        PCI_FUNC(pdev->devfn) ));

    /* alloc our per-device data */
    dev_num = alloc_ibb_soft_state();
    if(dev_num < 0)
    {
        /* I've already printk'd error message */
        return -ENOMEM;
    }

    /* initialize data */
    memset(IbbSoft[dev_num],0,sizeof(IbbSoft[dev_num]));
    ibb_sp = IbbSoft[dev_num];
    //sprintf(ibb_sp->devname, "ibb");
    sprintf(ibb_sp->devname, "ibb%d", dev_num);
    spin_lock_init(&ibb_sp->mutex);
    ibb_sp->dev_num = dev_num;
    ibb_sp->pdev = pdev;

```

Linux-Kernel: Compiling a 2.4 driver under 2.6

```
ibb_sp->height_inspection = 0;
sema_init(&ibb_sp->sem, 1);
pci_set_drvdata(pdev,ibb_sp); /* we'll need this in remove: */

dcmn_err(40, ( "<1>" "ibb_probe: device %d\n",dev_num) );

#define REQUES
```