

[PATCH/RFC 2.6.20-rc4 1/1] fbdev,mm: hecuba/E-Ink fbdev driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-01/msg03123.html>

- *From:* Jaya Kumar <jayakumar.lkml@xxxxxxxx>
 - *Date:* Thu, 11 Jan 2007 15:24:27 +0100
-

This patch adds support for the Hecuba/E-Ink display with deferred IO.
I welcome your feedback and advice.

Signed-off-by: Jaya Kumar <jayakumar.lkml@xxxxxxxx>

drivers/video/Kconfig | 13 +
drivers/video/Makefile | 1
drivers/video/hecubafb.c | 568 +++
mm/filemap.c | 1
mm/rmap.c | 1
5 files changed, 584 insertions(+)

diff --git a/drivers/video/Kconfig b/drivers/video/Kconfig
index 4e83f01..cf2dc50 100644
--- a/drivers/video/Kconfig
+++ b/drivers/video/Kconfig
@@ -568,6 +568,19 @@ config FB_IMAC
help
This is the frame buffer device driver for the Intel-based Macintosh

```
+config FB_HECUBA  
+ tristate "Hecuba board support"  
+ depends on FB && X86 && MMU  
+ select FB_CFB_FILLRECT  
+ select FB_CFB_COPYAREA  
+ select FB_CFB_IMAGEBLIT  
+ help  
+ This enables support for the Hecuba board. This driver was tested  
+ with an E-Ink 800x600 display and x86 SBCs through a 16 bit GPIO  
+ interface (8 bit data, 4 bit control). If you anticipate using  
+ this driver, say Y or M; otherwise say N. You must specify the  
+ GPIO IO address to be used for setting control and data.  
+  
config FB_HGA
```

```
tristate "Hercules mono graphics support"
depends on FB && X86
diff --git a/drivers/video/Makefile b/drivers/video/Makefile
index 309a26d..b4d5655 100644
--- a/drivers/video/Makefile
+++ b/drivers/video/Makefile
@@ -67,6 +67,7 @@ obj-$(CONFIG_FB_SGIVW) += sgivwfb.o
obj-$(CONFIG_FB_ACORN) += acornfb.o
obj-$(CONFIG_FB_ATARI) += atafb.o
obj-$(CONFIG_FB_MAC) += macfb.o
+obj-$(CONFIG_FB_HECUBA) += hecubafb.o
obj-$(CONFIG_FB_HGA) += hgafb.o
obj-$(CONFIG_FB_IGA) += igafb.o
obj-$(CONFIG_FB_APOLLO) += dnfb.o
diff --git a/drivers/video/hecubafb.c b/drivers/video/hecubafb.c
new file mode 100644
index 0000000..4740b92
--- /dev/null
+++ b/drivers/video/hecubafb.c
@@ -0,0 +1,568 @@
+/*
+ * linux/drivers/video/hecubafb.c --- FB driver for Hecuba controller
+ *
+ * Copyright (C) 2006, Jaya Kumar
+ * This work was sponsored by CIS(M) Sdn Bhd
+ *
+ * This file is subject to the terms and conditions of the GNU General Public
+ * License. See the file COPYING in the main directory of this archive for
+ * more details.
+ *
+ * Layout is based on skeletonfb.c by James Simmons and Geert Uytterhoeven.
+ * This work was possible because of apollo display code from E-Ink's website
+ * http://support.eink.com/community
+ * All information used to write this code is from public material made
+ * available by E-Ink on its support site. Some commands such as 0xA4
+ * were found by looping through cmd=0x00 thru 0xFF and supplying random
+ * values. There are other commands that the display is capable of,
+ * beyond the 5 used here but they are more complex.
+ *
+ * This driver is written to be used with the Hecuba display controller
+ * board, and tested with the EInk 800x600 display in 1 bit mode.
+ * The interface between Hecuba and the host is TTL based GPIO. The
+ * GPIO requirements are 8 writable data lines and 6 lines for control.
+ * Only 4 of the controls are actually used here but 6 for future use.
+ * The driver requires the IO addresses for data and control GPIO at
+ * load time. It is also possible to use this display with a standard
+ * PC parallel port.
+ *
+ * General notes:
+ * - User must set hecubafb_enable=1 to enable it
+ * - User must set dio_addr=0xIOADDR cio_addr=0xIOADDR c2io_addr=0xIOADDR
```

```
+ *
+ */
+
+#include <asm/uaccess.h>
+#include <linux/module.h>
+#include <linux/kernel.h>
+#include <linux/errno.h>
+#include <linux/string.h>
+#include <linux/mm.h>
+#include <linux/slab.h>
+#include <linux/vmalloc.h>
+#include <linux/delay.h>
+#include <linux/interrupt.h>
+#include <linux/fb.h>
+#include <linux/init.h>
+#include <linux/platform_device.h>
+#include <linux/list.h>
+
+/* to support deferred IO */
+#include <linux/rmap.h>
+#include <linux/pagemap.h>
+
+/* Apollo controller specific defines */
+#define APOLLO_START_NEW_IMG 0xA0
+#define APOLLO_STOP_IMG_DATA 0xA1
+#define APOLLO_DISPLAY_IMG 0xA2
+#define APOLLO_ERASE_DISPLAY 0xA3
+#define APOLLO_INIT_DISPLAY 0xA4
+
+/* Hecuba interface specific defines */
+/* WUP is inverted, CD is inverted, DS is inverted */
+#define HCB_NWUP_BIT 0x01
+#define HCB_NDS_BIT 0x02
+#define HCB_RW_BIT 0x04
+#define HCB_NCD_BIT 0x08
+#define HCB_ACK_BIT 0x80
+
+/* Display specific information */
+#define DPY_W 600
+#define DPY_H 800
+
+struct hecubafb_par {
+ struct delayed_work deferred_work;
+ unsigned long dio_addr;
+ unsigned long cio_addr;
+ unsigned long c2io_addr;
+ unsigned char ctl;
+ atomic_t ref_count;
+ atomic_t vma_count;
+ struct fb_info *info;
+ unsigned int irq;
+};
```

```

+ spinlock_t lock;
+ struct list_head pagelist;
+};
+
+struct page_list {
+ struct list_head list;
+ struct page *page;
+};
+
+static struct fb_fix_screeninfo hecubafb_fix __initdata = {
+ .id = "hecubafb",
+ .type = FB_TYPE_PACKED_PIXELS,
+ .visual = FB_VISUAL_MONO01,
+ .xpanstep = 0,
+ .ypanstep = 0,
+ .ywrapstep = 0,
+ .accel = FB_ACCEL_NONE,
+};
+
+static struct fb_var_screeninfo hecubafb_var __initdata = {
+ .xres = DPY_W,
+ .yres = DPY_H,
+ .xres_virtual = DPY_W,
+ .yres_virtual = DPY_H,
+ .bits_per_pixel = 1,
+ .nonstd = 1,
+};
+
+static unsigned long dio_addr;
+static unsigned long cio_addr;
+static unsigned long c2io_addr;
+static unsigned long splashval;
+static unsigned int nosplash;
+static unsigned int hecubafb_enable;
+static unsigned int irq;
+
+static DECLARE_WAIT_QUEUE_HEAD(hecubafb_waitq);
+
+static void hcb_set_ctl(struct hecubafb_par *par)
+{
+ outb(par->ctl, par->cio_addr);
+}
+
+static unsigned char hcb_get_ctl(struct hecubafb_par *par)
+{
+ return inb(par->c2io_addr);
+}
+
+static void hcb_set_data(struct hecubafb_par *par, unsigned char value)
+{
+ outb(value, par->dio_addr);

```

```

+}
+
+static int __devinit apollo_init_control(struct hecubafb_par *par)
+{
+ unsigned char ctl;
+ /* for init, we want the following setup to be set:
+ WUP = lo
+ ACK = hi
+ DS = hi
+ RW = hi
+ CD = lo
+ */
+
+ /* write WUP to lo, DS to hi, RW to hi, CD to lo */
+ par->ctl = HCB_NWUP_BIT | HCB_RW_BIT | HCB_NCD_BIT ;
+ par->ctl &= ~HCB_NDS_BIT;
+ hcb_set_ctl(par);
+
+ /* check ACK is not lo */
+ ctl = hcb_get_ctl(par);
+ if ((ctl & HCB_ACK_BIT)) {
+ printk(KERN_ERR "Fail because ACK is already low\n");
+ return -ENXIO;
+ }
+
+ return 0;
+}
+
+void hcb_wait_for_ack(struct hecubafb_par *par)
+{
+
+ int timeout;
+ unsigned char ctl;
+
+ timeout=500;
+ do {
+ ctl = hcb_get_ctl(par);
+ if ((ctl & HCB_ACK_BIT))
+ return;
+ udelay(1);
+ } while (timeout--);
+ printk(KERN_ERR "timed out waiting for ack\n");
+}
+
+void hcb_wait_for_ack_clear(struct hecubafb_par *par)
+{
+
+ int timeout;
+ unsigned char ctl;
+
+ timeout=500;

```

```

+ do {
+ ctl = hcb_get_ctl(par);
+ if (!(ctl & HCB_ACK_BIT))
+ return;
+ udelay(1);
+ } while (timeout--);
+ printk(KERN_ERR "timed out waiting for clear\n");
+}
+
+void apollo_send_data(struct hecubafb_par *par, unsigned char data)
+{
+ /* set data */
+ hcb_set_data(par, data);
+
+ /* set DS low */
+ par->ctl |= HCB_NDS_BIT;
+ hcb_set_ctl(par);
+
+ hcb_wait_for_ack(par);
+
+ /* set DS hi */
+ par->ctl &= ~(HCB_NDS_BIT);
+ hcb_set_ctl(par);
+
+ hcb_wait_for_ack_clear(par);
+}
+
+void apollo_send_command(struct hecubafb_par *par, unsigned char data)
+{
+ /* command so set CD to high */
+ par->ctl &= ~(HCB_NCD_BIT);
+ hcb_set_ctl(par);
+
+ /* actually strobe with command */
+ apollo_send_data(par, data);
+
+ /* clear CD back to low */
+ par->ctl |= (HCB_NCD_BIT);
+ hcb_set_ctl(par);
+}
+
+/* main hecubafb functions */
+
+static void hecubafb_dpy_update(struct hecubafb_par *par)
+{
+ int i;
+ unsigned char *buf = par->info->screen_base;
+
+ apollo_send_command(par, 0xA0);
+
+ for (i=0; i < (DPY_W*DPY_H/8); i++) {

```

```

+ apollo_send_data(par, *(buf++));
+ }
+
+ apollo_send_command(par, 0xA1);
+ apollo_send_command(par, 0xA2);
+}
+
+static void hecubafb_fillrect(struct fb_info *info,
+ const struct fb_fillrect *rect)
+{
+ struct hecubafb_par *par = info->par;
+
+ cfb_fillrect(info, rect);
+
+ hecubafb_dpy_update(par);
+}
+
+static void hecubafb_copyarea(struct fb_info *info,
+ const struct fb_copyarea *area)
+{
+ struct hecubafb_par *par = info->par;
+
+ cfb_copyarea(info, area);
+
+ hecubafb_dpy_update(par);
+}
+
+static void hecubafb_imageblit(struct fb_info *info,
+ const struct fb_image *image)
+{
+ struct hecubafb_par *par = info->par;
+
+ cfb_imageblit(info, image);
+
+ hecubafb_dpy_update(par);
+}
+
+/*
+ * this is the slow path from userspace. they can seek and write to
+ * the fb. it's inefficient to do anything less than a full screen draw
+ */
+static ssize_t hecubafb_write(struct file *file, const char __user *buf,
+ size_t count, loff_t *ppos)
+{
+ struct inode *inode;
+ int fbidx;
+ struct fb_info *info;
+ unsigned long p;
+ int err=-EINVAL;
+ struct hecubafb_par *par;
+ unsigned int xres;

```

```

+ unsigned int fbmemlength;
+
+ p = *ppos;
+ inode = file->f_dentry->d_inode;
+ fbidx = iminor(inode);
+ info = registered_fb[fbidx];
+
+ if (!info || !info->screen_base)
+ return -ENODEV;
+
+ par = info->par;
+ xres = info->var.xres;
+ fbmemlength = (xres * info->var.yres)/8;
+
+ if (p > fbmemlength)
+ return -ENOSPC;
+
+ err = 0;
+ if ((count + p) > fbmemlength) {
+ count = fbmemlength - p;
+ err = -ENOSPC;
+ }
+
+ if (count) {
+ char *base_addr;
+
+ base_addr = info->screen_base;
+ count -= copy_from_user(base_addr + p, buf, count);
+ *ppos += count;
+ err = -EFAULT;
+ }
+
+ hecubafb_dpy_update(par);
+
+ if (count)
+ return count;
+
+ return err;
+}
+
+/* this is to find and return the vmalloc-ed fb pages */
+static struct page* hecubafb_vm_nopage(struct vm_area_struct *vma,
+ unsigned long vaddr, int *type)
+{
+ unsigned long offset;
+ struct page *page;
+ struct fb_info *info = vma->vm_private_data;
+
+ offset = (vaddr - vma->vm_start) + (vma->vm_pgoff << PAGE_SHIFT);
+ if (offset >= (DPY_W*DPY_H)/8)
+ return NOPAGE_SIGBUS;

```

```

+
+ page = vmalloc_to_page(info->screen_base + offset);
+ if (!page)
+ return NOPAGE_OOM;
+
+ get_page(page);
+ if (type)
+ *type = VM_FAULT_MINOR;
+ return page;
+}
+
+static void hecubafb_work(struct work_struct *work)
+{
+ struct hecubafb_par *par = container_of(work, struct hecubafb_par,
+ deferred_work.work);
+ struct list_head *node, *next;
+ struct page_list *cur;
+
+ /* here we unmap the pages, then do all deferred IO */
+ spin_lock(&par->lock);
+ list_for_each_safe(node, next, &par->pagelist) {
+ cur = list_entry(node, struct page_list, list);
+ list_del(node);
+ lock_page_nosync(cur->page);
+ page_mkclean(cur->page);
+ unlock_page(cur->page);
+ kfree(cur);
+ }
+ spin_unlock(&par->lock);
+ hecubafb_dpy_update(par);
+}
+
+static int hecubafb_page_mkwrite(struct vm_area_struct *vma,
+ struct page *page)
+{
+ struct fb_info *info = vma->vm_private_data;
+ struct hecubafb_par *par = info->par;
+ struct page_list *new;
+
+ /* this is a callback we get when userspace first tries to
+ write to the page. we schedule a workqueue. that workqueue
+ will eventually unmap the touched pages and execute the
+ deferred framebuffer IO. then if userspace touches a page
+ again, we repeat the same scheme */
+
+ new = kzalloc(sizeof(struct page_list), GFP_KERNEL);
+ if (!new)
+ return -ENOMEM;
+ new->page = page;
+
+ /* protect against the workqueue changing the page list */

```

```

+ spin_lock(&par->lock);
+ list_add(&new->list, &par->pagelist);
+ spin_unlock(&par->lock);
+
+ /* come back in 1s to process the deferred IO */
+ schedule_delayed_work(&par->deferred_work, HZ);
+ return 0;
+}
+
+static struct vm_operations_struct hecubafb_vm_ops = {
+ .nopage = hecubafb_vm_nopage,
+ .page_mkwrite = hecubafb_page_mkwrite,
+};
+
+static int hecubafb_mmap(struct fb_info *info, struct vm_area_struct *vma)
+{
+ vma->vm_ops = &hecubafb_vm_ops;
+ vma->vm_flags |= ( VM_IO | VM_RESERVED | VM_DONTEXPAND );
+ vma->vm_private_data = info;
+ return 0;
+}
+
+static struct fb_ops hecubafb_ops = {
+ .owner = THIS_MODULE,
+ .fb_write = hecubafb_write,
+ .fb_fillrect = hecubafb_fillrect,
+ .fb_copyarea = hecubafb_copyarea,
+ .fb_imageblit = hecubafb_imageblit,
+ .fb_mmap = hecubafb_mmap,
+};
+
+static int __devinit hecubafb_probe(struct platform_device *dev)
+{
+ struct fb_info *info;
+ int retval = -ENOMEM;
+ int videomemorysize;
+ unsigned char *videomemory;
+ struct hecubafb_par *par;
+
+ videomemorysize = (DPY_W*DPY_H)/8;
+
+ if (!(videomemory = vmalloc(videomemorysize)))
+ return retval;
+
+ memset(videomemory, 0, videomemorysize);
+
+ info = framebuffer_alloc(sizeof(struct hecubafb_par), &dev->dev);
+ if (!info)
+ goto err;
+
+ info->screen_base = (char __iomem *) videomemory;

```

```

+ info->fbops = &hecubafb_ops;
+
+ info->var = hecubafb_var;
+ info->fix = hecubafb_fix;
+ par = info->par;
+ par->info = info;
+
+ if (!dio_addr || !cio_addr || !c2io_addr) {
+ printk(KERN_WARNING "no IO addresses supplied\n");
+ goto err1;
+ }
+ par->dio_addr = dio_addr;
+ par->cio_addr = cio_addr;
+ par->c2io_addr = c2io_addr;
+ info->flags = FBINFO_FLAG_DEFAULT;
+ spin_lock_init(&par->lock);
+ INIT_DELAYED_WORK(&par->deferred_work, hecubafb_work);
+ INIT_LIST_HEAD(&par->pagelist);
+ retval = register_framebuffer(info);
+ if (retval < 0)
+ goto err1;
+ platform_set_drvdata(dev, info);
+
+ printk(KERN_INFO
+ "fb%d: Hecuba frame buffer device, using %dK of video memory\n",
+ info->node, videomemorysize >> 10);
+
+ /* this inits the dpy */
+ apollo_init_control(par);
+
+ apollo_send_command(par, APOLLO_INIT_DISPLAY);
+ apollo_send_data(par, 0x81);
+
+ /* have to wait while display resets */
+ udelay(1000);
+
+ /* if we were told to splash the screen, we just clear it */
+ if (!nosplash) {
+ apollo_send_command(par, APOLLO_ERASE_DISPLAY);
+ apollo_send_data(par, splashval);
+ }
+
+ return 0;
+err1:
+ framebuffer_release(info);
+err:
+ vfree(videomemory);
+ return retval;
+}
+
+static int __devexit hecubafb_remove(struct platform_device *dev)

```

```

+{
+ struct fb_info *info = platform_get_drvdata(dev);
+ struct hecubafb_par *par;
+
+ if (info) {
+ par = info->par;
+ cancel_delayed_work(&par->deferred_work);
+ flush_scheduled_work();
+ unregister_framebuffer(info);
+ vfree(info->screen_base);
+ framebuffer_release(info);
+ }
+ return 0;
+}
+
+static struct platform_driver hecubafb_driver = {
+ .probe = hecubafb_probe,
+ .remove = hecubafb_remove,
+ .driver = {
+ .name = "hecubafb",
+ },
+};
+
+static struct platform_device *hecubafb_device;
+
+static int __init hecubafb_init(void)
+{
+ int ret;
+
+ if (!hecubafb_enable) {
+ printk(KERN_ERR "Use hecubafb_enable to enable the device\n");
+ return -ENXIO;
+ }
+
+ ret = platform_driver_register(&hecubafb_driver);
+ if (!ret) {
+ hecubafb_device = platform_device_alloc("hecubafb", 0);
+ if (hecubafb_device)
+ ret = platform_device_add(hecubafb_device);
+ else
+ ret = -ENOMEM;
+
+ if (ret) {
+ platform_device_put(hecubafb_device);
+ platform_driver_unregister(&hecubafb_driver);
+ }
+ }
+ return ret;
+}
+
+

```

```

+static void __exit hecubafb_exit(void)
+{
+ platform_device_unregister(hecubafb_device);
+ platform_driver_unregister(&hecubafb_driver);
+}
+
+module_param(nosplash, uint, 0);
+MODULE_PARM_DESC(nosplash, "Disable doing the splash screen");
+module_param(hecubafb_enable, uint, 0);
+MODULE_PARM_DESC(hecubafb_enable, "Enable communication with Hecuba board");
+module_param(dio_addr, ulong, 0);
+MODULE_PARM_DESC(dio_addr, "IO address for data, eg: 0x480");
+module_param(cio_addr, ulong, 0);
+MODULE_PARM_DESC(cio_addr, "IO address for control, eg: 0x400");
+module_param(c2io_addr, ulong, 0);
+MODULE_PARM_DESC(c2io_addr, "IO address for secondary control, eg: 0x408");
+module_param(splashval, ulong, 0);
+MODULE_PARM_DESC(splashval, "Splash pattern: 0x00 is black, 0x01 is white");
+module_param(irq, uint, 0);
+MODULE_PARM_DESC(irq, "IRQ for the Hecuba board");
+
+module_init(hecubafb_init);
+module_exit(hecubafb_exit);
+
+MODULE_DESCRIPTION("fbdev driver for Hecuba board");
+MODULE_AUTHOR("Jaya Kumar");
+MODULE_LICENSE("GPL");
diff --git a/mm/filemap.c b/mm/filemap.c
index 8332c77..f835f68 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -583,6 +583,7 @@ void fastcall __lock_page_nosync(struct page *page)
__wait_on_bit_lock(page_waitqueue(page), &wait, __sleep_on_page_lock,
TASK_UNINTERRUPTIBLE);
}
+EXPORT_SYMBOL_GPL(__lock_page_nosync);

/**
 * find_get_page - find and get a page reference
diff --git a/mm/rmap.c b/mm/rmap.c
index 669acb2..0fa0521 100644
--- a/mm/rmap.c
+++ b/mm/rmap.c
@@ -496,6 +496,7 @@ int page_mkclean(struct page *page)

return ret;
}
+EXPORT_SYMBOL_GPL(page_mkclean);

/**
 * page_set_anon_rmap - setup new anonymous rmap

```

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>