

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-04/msg12100.html>

- *From:* ebiederm@xxxxxxxxxxxxx (Eric W. Biederman)
 - *Date:* Mon, 30 Apr 2007 10:32:02 -0600
-

With legacy free systems serial ports have stopped being an option to get early boot traces and other debug information out of a machine.

EHCI USB controllers provide a relatively simple debug interface that can control port 1 of the root hub. This interface is limited to 8 byte packets so it can not be used with most USB devices. But with a USB debug device this is sufficient to talk to another machine.

When the special feature of the EHCI is not enabled the port 1 of the root hub acts just like any other USB port so machines with the necessary support are widely available.

This debug device can be used to replace serial ports for kgdb, kdb, and console support. And gregkh has a simple usb serial driver for it so user space applications that control serial ports should work unmodified.

Currently there only appears to be one manufacturer of debug devices see:

<http://www.plxtech.com/products/NET2000/NET20DC/default.asp>

I think simple RS232 serial ports provide a nicer and simpler interface but the usb debug port looks like a functional alternative when you don't have that.

My code likely doesn't handle all of the corner cases yet. But this is getting it out there so other people can starting using it and help make clean drivers. When writing a polling driver you do have to be careful with your logic, because if you do things like reset a usb device at the wrong time you can completely confuse various EHCI controllers.

My driver should be sufficient to work with any EHCI in a realatively clean state, and needs no special BIOS support just the hardware. This appears to be different than the way the windows drivers are using these debug devices.

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

The big dependency that was holding this code back was the requirement for early fixmap support and that is now present (early in this patchset) so things should work relatively cleanly.

There is currently a conflict between the early debug code and ehci-hcd where if you leave the early debug printk' enabled after the normal console hand off point "earlyprintk=dbgp,keep" the ehci-hcd driver will hang the kernel while initializing, despite detecting that the early debug point is in use.

For users the hard part looks like it will be finding cables and finding which is usb debug port 1 and realizing that there is flow control so the kernel boot will not happen if someone is not reading the serial console data.

Signed-off-by: Eric W. Biederman <ebiederm@xxxxxxxxxxxx>

```
-----
arch/x86_64/kernel/early_printk.c | 571 ++++++
drivers/usb/host/ehci.h | 8 +
include/asm-i386/fixmap.h | 1 +
include/asm-x86_64/fixmap.h | 1 +
4 files changed, 581 insertions(+), 0 deletions(-)
```

```
diff --git a/arch/x86_64/kernel/early_printk.c b/arch/x86_64/kernel/early_printk.c
index 92213d2..dc097aa 100644
```

```
--- a/arch/x86_64/kernel/early_printk.c
+++ b/arch/x86_64/kernel/early_printk.c
```

```
@@ -3,9 +3,19 @@
```

```
#include <linux/init.h>
#include <linux/string.h>
#include <linux/screen_info.h>
+#include <linux/usb/ch9.h>
+#include <linux/pci_regs.h>
+#include <linux/pci_ids.h>
+#include <linux/errno.h>
#include <asm/io.h>
#include <asm/processor.h>
#include <asm/fcntl.h>
+#include <asm/pci-direct.h>
+#include <asm/pgtable.h>
+#include <asm/fixmap.h>
+#define EARLY_PRINTK
+#include "../..../drivers/usb/host/ehci.h"
+
```

```
/* Simple VGA output */
```

```
@@ -155,6 +165,555 @@ static struct console early_serial_console = {
.index = -1,
};
```

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

```
+
+static struct ehci_caps __iomem *ehci_caps;
+static struct ehci_regs __iomem *ehci_regs;
+static struct ehci_dbg_port __iomem *ehci_debug;
+static unsigned dbgp_endpoint_out;
+
+#define USB_DEBUG_DEVNUM 127
+
+#define DBGP_DATA_TOGGLE 0x8800
+#define DBGP_PID_UPDATE(x, tok) \
+ (((x) ^ DBGP_DATA_TOGGLE) & 0xffff00) | ((tok) & 0xff)
+
+#define DBGP_LEN_UPDATE(x, len) (((x) & ~0x0f) | ((len) & 0x0f))
+/*
+ * USB Packet IDs (PIDs)
+ */
+
+/* token */
+#define USB_PID_OUT 0xe1
+#define USB_PID_IN 0x69
+#define USB_PID_SOF 0xa5
+#define USB_PID_SETUP 0x2d
+/* handshake */
+#define USB_PID_ACK 0xd2
+#define USB_PID_NAK 0x5a
+#define USB_PID_STALL 0x1e
+#define USB_PID_NYET 0x96
+/* data */
+#define USB_PID_DATA0 0xc3
+#define USB_PID_DATA1 0x4b
+#define USB_PID_DATA2 0x87
+#define USB_PID_MDATA 0x0f
+/* Special */
+#define USB_PID_PREAMBLE 0x3c
+#define USB_PID_ERR 0x3c
+#define USB_PID_SPLIT 0x78
+#define USB_PID_PING 0xb4
+#define USB_PID_UNDEF_0 0xf0
+
+#define USB_PID_DATA_TOGGLE 0x88
+#define DBGP_CLAIM (DBGP_OWNER | DBGP_ENABLED | DBGP_INUSE)
+
+#define PCI_CAP_ID_EHCI_DEBUG 0xa
+
+#define HUB_ROOT_RESET_TIME 50 /* times are in msec */
+#define HUB_SHORT_RESET_TIME 10
+#define HUB_LONG_RESET_TIME 200
+#define HUB_RESET_TIMEOUT 500
+
+#define DBGP_MAX_PACKET 8
+
```

```
+static int dbgp_wait_until_complete(void)
+{
+ unsigned ctrl;
+ for (;;) {
+ ctrl = readl(&ehci_debug->control);
+ /* Stop when the transaction is finished */
+ if (ctrl & DBGP_DONE)
+ break;
+ }
+ /* Now that we have observed the completed transaction,
+ * clear the done bit.
+ */
+ writel(ctrl | DBGP_DONE, &ehci_debug->control);
+ return (ctrl & DBGP_ERROR) ? -DBGP_ERRCODE(ctrl) : DBGP_LEN(ctrl);
+}
+
+static void dbgp_mdelay(int ms)
+{
+ int i;
+ while (ms-->0) {
+ for (i = 0; i < 1000; i++)
+ outb(0x1, 0x80);
+ }
+}
+
+static void dbgp_breath(void)
+{
+ /* Sleep to give the debug port a chance to breathe */
+}
+
+static int dbgp_wait_until_done(unsigned ctrl)
+{
+ unsigned pids, lpid;
+ int ret;
+
+retry:
+ writel(ctrl | DBGP_GO, &ehci_debug->control);
+ ret = dbgp_wait_until_complete();
+ pids = readl(&ehci_debug->pids);
+ lpid = DBGP_PID_GET(pids);
+
+ if (ret < 0)
+ return ret;
+
+ /* If the port is getting full or it has dropped data
+ * start pacing ourselves, not necessary but it's friendly.
+ */
+ if ((lpid == USB_PID_NAK) || (lpid == USB_PID_NYET))
+ dbgp_breath();
+
+ /* If I get a NACK reissue the transmission */
```

```

+ if (lpid == USB_PID_NAK)
+ goto retry;
+
+ return ret;
+}
+
+static void dbgp_set_data(const void *buf, int size)
+{
+ const unsigned char *bytes = buf;
+ unsigned lo, hi;
+ int i;
+ lo = hi = 0;
+ for (i = 0; i < 4 && i < size; i++)
+ lo |= bytes[i] << (8*i);
+ for (; i < 8 && i < size; i++)
+ hi |= bytes[i] << (8*(i - 4));
+ writel(lo, &ehci_debug->data03);
+ writel(hi, &ehci_debug->data47);
+}
+
+static void dbgp_get_data(void *buf, int size)
+{
+ unsigned char *bytes = buf;
+ unsigned lo, hi;
+ int i;
+ lo = readl(&ehci_debug->data03);
+ hi = readl(&ehci_debug->data47);
+ for (i = 0; i < 4 && i < size; i++)
+ bytes[i] = (lo >> (8*i)) & 0xff;
+ for (; i < 8 && i < size; i++)
+ bytes[i] = (hi >> (8*(i - 4))) & 0xff;
+}
+
+static int dbgp_bulk_write(unsigned devnum, unsigned endpoint, const char *bytes, int size)
+{
+ unsigned pids, addr, ctrl;
+ int ret;
+ if (size > DBGP_MAX_PACKET)
+ return -1;
+
+ addr = DBGP_EPADDR(devnum, endpoint);
+
+ pids = readl(&ehci_debug->pids);
+ pids = DBGP_PID_UPDATE(pids, USB_PID_OUT);
+
+ ctrl = readl(&ehci_debug->control);
+ ctrl = DBGP_LEN_UPDATE(ctrl, size);
+ ctrl |= DBGP_OUT;
+ ctrl |= DBGP_GO;
+
+ dbgp_set_data(bytes, size);

```

```
+ writel(addr, &ehci_debug->address);
+ writel(pids, &ehci_debug->pids);
+
+ ret = dbgp_wait_until_done(ctrl);
+ if (ret < 0) {
+ return ret;
+ }
+ return ret;
+}
+
+static int dbgp_bulk_read(unsigned devnum, unsigned endpoint, void *data, int size)
+{
+ unsigned pids, addr, ctrl;
+ int ret;
+
+ if (size > DBGP_MAX_PACKET)
+ return -1;
+
+ addr = DBGP_EPADDR(devnum, endpoint);
+
+ pids = readl(&ehci_debug->pids);
+ pids = DBGP_PID_UPDATE(pids, USB_PID_IN);
+
+ ctrl = readl(&ehci_debug->control);
+ ctrl = DBGP_LEN_UPDATE(ctrl, size);
+ ctrl &= ~DBGP_OUT;
+ ctrl |= DBGP_GO;
+
+ writel(addr, &ehci_debug->address);
+ writel(pids, &ehci_debug->pids);
+ ret = dbgp_wait_until_done(ctrl);
+ if (ret < 0)
+ return ret;
+ if (size > ret)
+ size = ret;
+ dbgp_get_data(data, size);
+ return ret;
+}
+
+static int dbgp_control_msg(unsigned devnum, int requesttype, int request,
+ int value, int index, void *data, int size)
+{
+ unsigned pids, addr, ctrl;
+ struct usb_ctrlrequest req;
+ int read;
+ int ret;
+
+ read = (requesttype & USB_DIR_IN) != 0;
+ if (size > (read?DBGP_MAX_PACKET:0))
+ return -1;
+
+ }
```

```

+ /* Compute the control message */
+ req.bRequestType = requesttype;
+ req.bRequest = request;
+ req.wValue = value;
+ req.wIndex = index;
+ req.wLength = size;
+
+ pids = DBGP_PID_SET(USB_PID_DATA0, USB_PID_SETUP);
+ addr = DBGP_EPADDR(devnum, 0);
+
+ ctrl = readl(&ehci_debug->control);
+ ctrl = DBGP_LEN_UPDATE(ctrl, sizeof(req));
+ ctrl |= DBGP_OUT;
+ ctrl |= DBGP_GO;
+
+ /* Send the setup message */
+ dbgp_set_data(&req, sizeof(req));
+ writel(addr, &ehci_debug->address);
+ writel(pids, &ehci_debug->pids);
+ ret = dbgp_wait_until_done(ctrl);
+ if (ret < 0)
+ return ret;
+
+
+ /* Read the result */
+ ret = dbgp_bulk_read(devnum, 0, data, size);
+ return ret;
+}
+
+
+/* Find a PCI capability */
+static __u32 __init find_cap(int num, int slot, int func, int cap)
+{
+ u8 pos;
+ int bytes;
+ if (!(read_pci_config_16(num,slot,func,PCI_STATUS) & PCI_STATUS_CAP_LIST))
+ return 0;
+ pos = read_pci_config_byte(num,slot,func,PCI_CAPABILITY_LIST);
+ for (bytes = 0; bytes < 48 && pos >= 0x40; bytes++) {
+ u8 id;
+ pos &= ~3;
+ id = read_pci_config_byte(num,slot,func,pos+PCI_CAP_LIST_ID);
+ if (id == 0xff)
+ break;
+ if (id == cap)
+ return pos;
+ pos = read_pci_config_byte(num,slot,func,pos+PCI_CAP_LIST_NEXT);
+ }
+ return 0;
+}
+
+

```

```

+static __u32 __init find_dbgp(int ehci_num, unsigned *rbus, unsigned *rslot, unsigned *rfunc)
+{
+ unsigned bus, slot, func;
+
+ for (bus = 0; bus < 256; bus++) {
+ for (slot = 0; slot < 32; slot++) {
+ for (func = 0; func < 8; func++) {
+ u32 class;
+ unsigned cap;
+ class = read_pci_config(bus, slot, func, PCI_CLASS_REVISION);
+ if ((class >> 8) != PCI_CLASS_SERIAL_USB_EHCI)
+ continue;
+ cap = find_cap(bus, slot, func, PCI_CAP_ID_EHCI_DEBUG);
+ if (!cap)
+ continue;
+ if (ehci_num-- != 0)
+ continue;
+ *rbus = bus;
+ *rslot = slot;
+ *rfunc = func;
+ return cap;
+ }
+ }
+ }
+ return 0;
+}
+
+static int ehci_reset_port(int port)
+{
+ unsigned portsc;
+ unsigned delay_time, delay;
+
+ /* Reset the usb debug port */
+ portsc = readl(&ehci_regs->port_status[port - 1]);
+ portsc &= ~PORT_PE;
+ portsc |= PORT_RESET;
+ writel(portsc, &ehci_regs->port_status[port - 1]);
+
+ delay = HUB_ROOT_RESET_TIME;
+ for (delay_time = 0; delay_time < HUB_RESET_TIMEOUT;
+ delay_time += delay) {
+ dbgp_mdelay(delay);
+
+ portsc = readl(&ehci_regs->port_status[port - 1]);
+ if (portsc & PORT_RESET) {
+ /* force reset to complete */
+ writel(portsc & ~(PORT_RWC_BITS | PORT_RESET),
+ &ehci_regs->port_status[port - 1]);
+ while (portsc & PORT_RESET)
+ portsc = readl(&ehci_regs->port_status[port - 1]);
+ }
+ }

```

```

+
+ /* Device went away? */
+ if (!(portsc & PORT_CONNECT))
+ return -ENOTCONN;
+
+ /* bomb out completely if something weird happend */
+ if ((portsc & PORT_CSC))
+ return -EINVAL;
+
+ /* If we've finished resetting, then break out of the loop */
+ if (!(portsc & PORT_RESET) && (portsc & PORT_PE))
+ return 0;
+ }
+ return -EBUSY;
+}
+
+static int ehci_wait_for_port(int port)
+{
+ unsigned status;
+ int ret, reps;
+ for (reps = 0; reps >= 0; reps++) {
+ status = readl(&ehci_regs->status);
+ if (status & STS_PCD) {
+ ret = ehci_reset_port(port);
+ if (ret == 0)
+ return 0;
+ }
+ }
+ return -ENOTCONN;
+}
+
+
+
+#define DBGP_DEBUG 0
+#if DBGP_DEBUG
+void early_printk(const char *fmt, ...);
+# define dbgp_printk early_printk
+#else
+static inline void dbgp_printk(const char *fmt, ...) { }
+#endif
+
+static int ehci_setup(void)
+{
+ unsigned cmd, ctrl, status, portsc, hcs_params, debug_port, n_ports;
+ int ret;
+
+ hcs_params = readl(&ehci_caps->hcs_params);
+ debug_port = HCS_DEBUG_PORT(hcs_params);
+ n_ports = HCS_N_PORTS(hcs_params);
+
+ dbgp_printk("debug_port: %d\n", debug_port);
+ dbgp_printk("n_ports: %d\n", n_ports);

```

```

+
+ /* Reset the EHCI controller */
+ cmd = readl(&ehci_regs->command);
+ cmd |= CMD_RESET;
+ writel(cmd, &ehci_regs->command);
+ while (cmd & CMD_RESET)
+ cmd = readl(&ehci_regs->command);
+
+ /* Claim ownership, but do not enable yet */
+ ctrl = readl(&ehci_debug->control);
+ ctrl |= DBGP_OWNER;
+ ctrl &= ~(DBGP_ENABLED | DBGP_INUSE);
+ writel(ctrl, &ehci_debug->control);
+
+ /* Start the ehci running */
+ cmd = readl(&ehci_regs->command);
+ cmd &= ~(CMD_LRESET | CMD_IAAD | CMD_PSE | CMD_ASE | CMD_RESET);
+ cmd |= CMD_RUN;
+ writel(cmd, &ehci_regs->command);
+
+ /* Ensure everything is routed to the EHCI */
+ writel(FLAG_CF, &ehci_regs->configured_flag);
+
+ /* Wait until the controller is no longer halted */
+ do {
+ status = readl(&ehci_regs->status);
+ } while (status & STS_HALT);
+
+ /* Wait for a device to show up in the debug port */
+ ret = ehci_wait_for_port(debug_port);
+ if (ret < 0) {
+ dbgp_printk("No device found in debug port\n");
+ return -1;
+ }
+
+ /* Enable the debug port */
+ ctrl = readl(&ehci_debug->control);
+ ctrl |= DBGP_CLAIM;
+ writel(ctrl, &ehci_debug->control);
+ ctrl = readl(&ehci_debug->control);
+ if ((ctrl & DBGP_CLAIM) != DBGP_CLAIM) {
+ dbgp_printk("No device in debug port\n");
+ writel(ctrl & ~DBGP_CLAIM, &ehci_debug->control);
+ return -1;
+ }
+
+ /* Completely transfer the debug device to the debug controller */
+ portsc = readl(&ehci_regs->port_status[debug_port - 1]);
+ portsc &= ~PORT_PE;
+ writel(portsc, &ehci_regs->port_status[debug_port - 1]);

```

```

+
+ return 0;
+}
+
+static __init void early_dbgp_init(char *s)
+{
+ struct usb_debug_descriptor dbgp_desc;
+ void __iomem *ehci_bar;
+ unsigned ctrl, devnum;
+ unsigned bus, slot, func, cap;
+ unsigned debug_port, bar, offset;
+ unsigned bar_val;
+ unsigned dbgp_num;
+ char *e;
+ int ret;
+
+ if (!early_pci_allowed())
+ return;
+
+ dbgp_num = 0;
+ if (*s) {
+ dbgp_num = simple_strtoul(s, &e, 10);
+ }
+ dbgp_printk("dbgp_num: %d\n", dbgp_num);
+ cap = find_dbgp(dbgp_num, &bus, &slot, &func);
+ if (!cap)
+ return;
+
+ dbgp_printk("Found EHCI debug port\n");
+
+ debug_port = read_pci_config(bus, slot, func, cap);
+ bar = (debug_port >> 29) & 0x7;
+ bar = (bar * 4) + 0xc;
+ offset = (debug_port >> 16) & 0xfff;
+ if (bar != PCI_BASE_ADDRESS_0) {
+ dbgp_printk("only debug ports on bar 1 handled.\n");
+ return;
+ }
+
+ /* FIXME this assumes the bar is a 32bit mmio bar */
+ bar_val = read_pci_config(bus, slot, func, PCI_BASE_ADDRESS_0);
+
+ /* FIXME I don't have the bar size so just guess PAGE_SIZE is more
+ * than enough. 1K is the biggest I have seen.
+ */
+ set_fixmap_nocache(FIX_DBGP_BASE, bar_val & PAGE_MASK);
+ ehci_bar = (void __iomem *)fix_to_virt(FIX_DBGP_BASE);
+ ehci_bar += bar_val & ~PAGE_MASK;
+
+ ehci_caps = ehci_bar;
+ ehci_regs = ehci_bar + HC_LENGTH(readl(&ehci_caps->hc_capbase));

```

```

+ ehci_debug = ehci_bar + offset;
+
+ ret = ehci_setup();
+ if (ret < 0) {
+ dbgp_printk("ehci_setup failed\n");
+ return;
+ }
+
+ /* Find the debug device and make it device number 127 */
+ for (devnum = 0; devnum <= 127; devnum++) {
+ ret = dbgp_control_msg(devnum,
+ USB_DIR_IN | USB_TYPE_STANDARD | USB_RECIP_DEVICE,
+ USB_REQ_GET_DESCRIPTOR, (USB_DT_DEBUG << 8), 0,
+ &dbgp_desc, sizeof(dbgp_desc));
+ if (ret > 0)
+ break;
+ }
+ if (devnum > 127) {
+ dbgp_printk("Could not find attached debug device\n");
+ goto err;
+ }
+ if (ret < 0) {
+ dbgp_printk("Attached device is not a debug device\n");
+ goto err;
+ }
+ dbgp_endpoint_out = dbgp_desc.bDebugOutEndpoint;
+
+ /* Move the device to 127 if it isn't already there */
+ if (devnum != USB_DEBUG_DEVNUM) {
+ ret = dbgp_control_msg(devnum,
+ USB_DIR_OUT | USB_TYPE_STANDARD | USB_RECIP_DEVICE,
+ USB_REQ_SET_ADDRESS, USB_DEBUG_DEVNUM, 0, NULL, 0);
+ if (ret < 0) {
+ dbgp_printk("Could not move attached device to %d\n",
+ USB_DEBUG_DEVNUM);
+ goto err;
+ }
+ devnum = USB_DEBUG_DEVNUM;
+ }
+
+ /* Enable the debug interface */
+ ret = dbgp_control_msg(USB_DEBUG_DEVNUM,
+ USB_DIR_OUT | USB_TYPE_STANDARD | USB_RECIP_DEVICE,
+ USB_REQ_SET_FEATURE, USB_DEVICE_DEBUG_MODE, 0, NULL, 0);
+ if (ret < 0) {
+ dbgp_printk(" Could not enable the debug device\n");
+ goto err;
+ }
+
+ /* Perform a small write to get the even/odd data state in sync
+ */

```

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

```
+ ret = dbgp_bulk_write(USB_DEBUG_DEVNUM, dbgp_endpoint_out, " ",1);
+ if (ret < 0) {
+ dbgp_printk("dbgp_bulk_write failed: %d\n", ret);
+ goto err;
+ }
+
+
+ return;
+err:
+ /* Things didn't work so remove my claim */
+ ctrl = readl(&ehci_debug->control);
+ ctrl &= ~(DBGP_CLAIM | DBGP_OUT);
+ writel(ctrl, &ehci_debug->control);
+ return;
+}
+
+static void early_dbgp_write(struct console *con, const char *str, unsigned n)
+{
+ int chunk, ret;
+ if (!ehci_debug)
+ return;
+ while (n > 0) {
+ chunk = n;
+ if (chunk > DBGP_MAX_PACKET)
+ chunk = DBGP_MAX_PACKET;
+ ret = dbgp_bulk_write(USB_DEBUG_DEVNUM,
+ dbgp_endpoint_out, str, chunk);
+ str += chunk;
+ n -= chunk;
+ }
+}
+
+static struct console early_dbgp_console = {
+ .name = "earlydbg",
+ .write = early_dbgp_write,
+ .flags = CON_PRINTBUFFER,
+ .index = -1,
+};
+
+/* Console interface to a host file on AMD's SimNow! */
+
+static int simnow_fd;
+@@ -242,8 +801,20 @@ static int __init setup_early_printk(char *buf)
+ simnow_init(buf + 6);
+ early_console = &simnow_console;
+ keep_early = 1;
+ } else if (!strcmp(buf, "dbgp", 4)) {
+ early_dbgp_init(buf + 4);
+ early_console = &early_dbgp_console;
+ }
+ register_console(early_console);
```

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

```
+ #if DBGP_DEBUG
+ {
+ static const char dbgp_test_str[] =
+ "The quick brown fox jumped over the lazy dog!\n";
+ early_dbgp_init("");
+ early_dbgp_write(&early_dbgp_console,
+ dbgp_test_str, sizeof(dbgp_test_str) - 1);
+ }
+ #endif
return 0;
}
```

diff --git a/drivers/usb/host/ehci.h b/drivers/usb/host/ehci.h

index 46fa57a..8455f13 100644

--- a/drivers/usb/host/ehci.h

+++ b/drivers/usb/host/ehci.h

@@ -46,6 +46,7 @@ struct ehci_stats {

#define EHCI_MAX_ROOT_PORTS 15 /* see HCS_N_PORTS */

#ifndef EARLY_PRINTK

struct ehci_hcd { /* one per controller */

/* glue to PCI and HCD framework */

struct ehci_caps __iomem *caps;

@@ -166,6 +167,7 @@ timer_action (struct ehci_hcd *ehci, enum ehci_timer_action action)

mod_timer (&ehci->watchdog, t);

}

}

#endif /* EARLY_PRINTK */

/*-----*/

@@ -390,6 +392,7 @@ union ehci_shadow {

* These appear in both the async and (for interrupt) periodic schedules.

*/

#ifndef EARLY_PRINTK

struct ehci_qh {

/* first part defined by EHCI spec */

__le32 hw_next; /* see EHCI 3.6.1 */

@@ -438,6 +441,7 @@ struct ehci_qh {

#define NO_FRAME ((unsigned short)~0) /* pick new start */

struct usb_device *dev; /* access to TT */

} __attribute__((aligned (32)));

#endif /* EARLY_PRITNK */

/*-----*/

@@ -607,6 +611,8 @@ struct ehci_fstn {

union ehci_shadow fstn_next; /* ptr to periodic q entry */

} __attribute__((aligned (32)));

[PATCH 09/12] i386/x86_64: EHCI usb debug port early printk support.

```
+#ifndef EARLY_PRINTK
+
/*-----*/

#ifdef CONFIG_USB_EHCI_ROOT_HUB_TT
@@ -704,4 +710,6 @@ static inline void ehci_writel (const struct ehci_hcd *ehci,

/*-----*/

+#endif /* EARLY_PRINTK */
+
#endif /* __LINUX_EHCI_HCD_H */
diff --git a/include/asm-i386/fixmap.h b/include/asm-i386/fixmap.h
index 80ea052..c644922 100644
--- a/include/asm-i386/fixmap.h
+++ b/include/asm-i386/fixmap.h
@@ -54,6 +54,7 @@ extern unsigned long __FIXADDR_TOP;
enum fixed_addresses {
FIX_HOLE,
FIX_VDSO,
+ FIXDBGP_BASE,
#ifdef CONFIG_X86_LOCAL_APIC
FIX_APIC_BASE, /* local (CPU) APIC) --- required for SMP or not */
#endif
diff --git a/include/asm-x86_64/fixmap.h b/include/asm-x86_64/fixmap.h
index e90e167..3e716d9 100644
--- a/include/asm-x86_64/fixmap.h
+++ b/include/asm-x86_64/fixmap.h
@@ -35,6 +35,7 @@ enum fixed_addresses {
VSYSCALL_LAST_PAGE,
VSYSCALL_FIRST_PAGE = VSYSCALL_LAST_PAGE + ((VSYSCALL_END-VSYSCALL_START)
>> PAGE_SHIFT) - 1,
VSYSCALL_HPETS,
+ FIXDBGP_BASE,
FIX_HPETS_BASE,
FIX_APIC_BASE, /* local (CPU) APIC) --- required for SMP or not */
FIX_IO_APIC_BASE_0,
--
```

1.5.1.1.181.g2de0

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>