i386 Micro Channel Architecture Support

MCA support is enabled using the CONFIG_MCA define. A machine with a MCA bus will have the kernel variable MCA_bus set, assuming the BIOS feature bits are set properly (see arch/i386/boot/setup.S for information on how this detection is done).

Adapter Detection

The ideal MCA adapter detection is done through the use of the Programmable Option Select registers. Generic functions for doing this have been added in include/linux/mca.h and arch/i386/kernel/mca.c. Everything needed to detect adapters and read (and write) configuration information is there. A number of MCA-specific drivers already use this. The typical probe code looks like the following:

```
        #include <linux/mca.h>

        unsigned char pos2, pos3, pos4, pos5;
        struct net_device* dev;
        int slot;

        if( MCA_bus ) {
                slot = mca_find_adapter( ADAPTER_ID, 0 );
                if( slot == MCA_NOTFOUND ) {
                        return -ENODEV;
                }
                /* optional - see below */
                mca_set_adapter_name( slot, "adapter name & description" );
                mca_set_adapter_procfn( slot, dev_getinfo, dev );

                /* read the POS registers.  Most devices only use 2 and 3 */
                pos2 = mca_read_stored_pos( slot, 2 );
                pos3 = mca_read_stored_pos( slot, 3 );
                pos4 = mca_read_stored_pos( slot, 4 );
                pos5 = mca_read_stored_pos( slot, 5 );
        } else {
                return -ENODEV;
        }

        /* extract configuration from pos[2345] and set everything up */
```

Loadable modules should modify this to test that the specified IRQ and IO ports (plus whatever other stuff) match. See 3c523.c for example code (actually, smc-mca.c has a slightly more complex example that can handle a list of adapter ids).

Keep in mind that devices should never directly access the POS registers (via inb(), outb(), etc). While it's generally safe, there is a small potential for blowing up hardware when it's done at the wrong time. Furthermore, accessing a POS register disables a device temporarily. This is usually okay during startup, but do _you_ want to rely on it? During initial configuration, mca_init() reads all the POS registers into memory. mca_read_stored_pos() accesses that data. mca_read_pos()

and mca_write_pos() are also available for (safer) direct POS access, but their use is _highly_ discouraged. mca_write_pos() is particularly dangerous, as it is possible for adapters to be put in inconsistent states (i.e. sharing IO address, etc) and may result in crashes, toasted hardware, and blindness.

User level drivers (such as the AGX X server) can use /proc/mca/pos to find adapters (see below).

Some MCA adapters can also be detected via the usual ISA-style device probing (many SCSI adapters, for example). This sort of thing is highly discouraged. Perfectly good information is available telling you what's there, so there's no excuse for messing with random IO ports. However, we MCA people still appreciate any ISA-style driver that will work with our hardware. You take what you can get...

Level-Triggered Interrupts
==========

Because MCA uses level-triggered interrupts, a few problems arise with what might best be described as the ISA mindset and its effects on drivers. These sorts of problems are expected to become less common as more people use shared IRQs on PCI machines.

In general, an interrupt must be acknowledged not only at the ICU (which is done automagically by the kernel), but at the device level. In particular, IRQ 0 must be reset after a timer interrupt (now done in arch/i386/kernel/time.c) or the first timer interrupt hangs the system. There were also problems with the 1.3.x floppy drivers, but that seems to have been fixed.

IRQs are also shareable, and most MCA-specific devices should be coded with shared IRQs in mind.

/proc/mca
=========

/proc/mca is a directory containing various files for adapters and other stuff.

/proc/mca/pos	Straight listing of POS registers
/proc/mca/slot[1-8]	Information on adapter in specific slot
/proc/mca/video	Same for integrated video
/proc/mca/scsi	Same for integrated SCSI
/proc/mca/machine	Machine information

See Appendix A for a sample.

Device drivers can easily add their own information function for specific slots (including integrated ones) via the mca_set_adapter_procfn() call. Drivers that support this are ESDI, IBM SCSI, and 3c523. If a device is also a module, make sure that the proc function is removed in the module cleanup. This will require storing the slot information in a private structure somewhere. See the 3c523 driver for details.

Your typical proc function will look something like this:

```
        static int
        dev_getinfo( char* buf, int slot, void* d ) {
                struct net_device* dev = (struct net_device*) d;
                int len = 0;

                len += sprintf( buf+len, "Device: %s\n", dev->name );
                len += sprintf( buf+len, "IRQ: %d\n", dev->irq );
                len += sprintf( buf+len, "IO Port: %#lx-%#lx\n", ... );
                ...

                return len;
        }
```

Some of the standard MCA information will already be printed, so don't bother repeating it. Don't try putting in more than 3K of information.

Enable this function with:

```
        mca_set_adapter_procfn( slot, dev_getinfo, dev );
```

Disable it with:

```
        mca_set_adapter_procfn( slot, NULL, NULL );
```

It is also recommended that, even if you don't write a proc function, to set the name of the adapter (i.e. "PS/2 ESDI Controller") via mca_set_adapter_name(int slot, char* name).

MCA Device Drivers
==================

Currently, there are a number of MCA-specific device drivers.

1) PS/2 SCSI
        drivers/scsi/ibmmca.c
        drivers/scsi/ibmmca.h
   The driver for the IBM SCSI subsystem. Includes both integrated controllers and adapter cards. May require command-line arg "ibmmcascsi=io_port" to force detection of an adapter. If you have a machine with a front-panel display (i.e. model 95), you can use "ibmmcascsi=display" to enable a drive activity indicator.

2) 3c523
        drivers/net/3c523.c
        drivers/net/3c523.h
   3Com 3c523 Etherlink/MC ethernet driver.

3) SMC Ultra/MCA and IBM Adapter/A
        drivers/net/smc-mca.c
        drivers/net/smc-mca.h
        Driver for the MCA version of the SMC Ultra and various other OEM'ed and work-alike cards (Elite, Adapter/A, etc).

4) NE/2
        driver/net/ne2.c
        driver/net/ne2.h

The NE/2 is the MCA version of the NE2000. This may not work with clones that have a different adapter id than the original NE/2.

5) Future Domain MCS-600/700, OEM'd IBM Fast SCSI Adapter/A and Reply Sound Blaster/SCSI (SCSI part)
Better support for these cards than the driver for ISA. Supports multiple cards with IRQ sharing.

Also added boot time option of scsi-probe, which can do reordering of SCSI host adapters. This will direct the kernel on the order which SCSI adapter should be detected. Example:
scsi-probe=ibmmca,fd_mcs,adaptec1542,buslogic

The serial drivers were modified to support the extended IO port range of the typical MCA system (also #ifdef CONFIG_MCA).

The following devices work with existing drivers:
1) Token-ring
2) Future Domain SCSI (MCS-600, MCS-700, not MCS-350, OEM'ed IBM SCSI)
3) Adaptec 1640 SCSI (using the aha1542 driver)
4) Bustek/Buslogic SCSI (various)
5) Probably all Arcnet cards.
6) Some, possibly all, MCA IDE controllers.
7) 3Com 3c529 (MCA version of 3c509) (patched)

8) Intel EtherExpressMC (patched version)
You need to have CONFIG_MCA defined to have EtherExpressMC support.
9) Reply Sound Blaster/SCSI (SB part) (patched version)

Bugs & Other Weirdness
==========

NMIs tend to occur with MCA machines because of various hardware weirdness, bus timeouts, and many other non-critical things. Some basic code to handle them (inspired by the NetBSD MCA code) has been added to detect the guilty device, but it's pretty incomplete. If NMIs are a persistent problem (on some model 70 or 80s, they occur every couple shell commands), the CONFIG_IGNORE_NMI flag will take care of that.

Various Pentium machines have had serious problems with the FPU test in bugs.h. Basically, the machine hangs after the HLT test. This occurs, as far as we know, on the Pentium-equipped 85s, 95s, and some PC Servers. The PCI/MCA PC 750s are fine as far as I can tell. The ``mca-pentium'' boot-prompt flag will disable the FPU bug check if this is a problem with your machine.

The model 80 has a raft of problems that are just too weird and unique to get into here. Some people have no trouble while others have nothing but problems. I'd suspect some problems are related to the age of the average 80 and accompanying hardware deterioration, although others are definitely design problems with the hardware. Among the problems include SCSI controller problems, ESDI controller problems, and serious screw-ups in the floppy controller. Oh, and the parallel port is also pretty flaky. There were about 5 or 6 different model 80 motherboards produced to fix various obscure problems. As far as I know, it's pretty

much impossible to tell which bugs a particular model 80 has (other than triggering them, that is).

Drivers are required for some MCA memory adapters. If you're suddenly short a few megs of RAM, this might be the reason. The (I think) Enhanced Memory Adapter commonly found on the model 70 is one. There's a very alpha driver floating around, but it's pretty ugly (disassembled from the DOS driver, actually). See the MCA Linux web page (URL below) for more current memory info.

The Thinkpad 700 and 720 will work, but various components are either non-functional, flaky, or we don't know anything about them. The graphics controller is supposed to be some WD, but we can't get things working properly. The PCMCIA slots don't seem to work. Ditto for APM. The serial ports work, but detection seems to be flaky.

Credits
=======
A whole pile of people have contributed to the MCA code. I'd include their names here, but I don't have a list handy. Check the MCA Linux home page (URL below) for a perpetually out-of-date list.

==
MCA Linux Home Page: http://www.dgmicro.com/mca/

Christophe Beauregard
chrisb@truespectra.com
cpbeaure@calum.csclub.uwaterloo.ca

==
Appendix A: Sample /proc/mca

This is from my model 8595. Slot 1 contains the standard IBM SCSI adapter, slot 3 is an Adaptec AHA-1640, slot 5 is a XGA-1 video adapter, and slot 7 is the 3c523 Etherlink/MC.

```
/proc/mca/machine:
Model Id: 0xf8
Submodel Id: 0x14
BIOS Revision: 0x5

/proc/mca/pos:
Slot 1: ff 8e f1 fc a0 ff ff ff  IBM SCSI Adapter w/Cache
Slot 2: ff ff ff ff ff ff ff ff
Slot 3: 1f 0f 81 3b bf b6 ff ff
Slot 4: ff ff ff ff ff ff ff ff
Slot 5: db 8f 1d 5e fd c0 00 00
Slot 6: ff ff ff ff ff ff ff ff
Slot 7: 42 60 ff 08 ff ff ff ff  3Com 3c523 Etherlink/MC
Slot 8: ff ff ff ff ff ff ff ff
Video : ff ff ff ff ff ff ff ff
SCSI  : ff ff ff ff ff ff ff ff

/proc/mca/slot1:
Slot: 1
Adapter Name: IBM SCSI Adapter w/Cache
```

```
Id: 8eff
Enabled: Yes
POS: ff 8e f1 fc a0 ff ff ff
Subsystem PUN: 7
Detected at boot: Yes

/proc/mca/slot3:
Slot: 3
Adapter Name: Unknown
Id: 0f1f
Enabled: Yes
POS: 1f 0f 81 3b bf b6 ff ff

/proc/mca/slot5:
Slot: 5
Adapter Name: Unknown
Id: 8fdb
Enabled: Yes
POS: db 8f 1d 5e fd c0 00 00

/proc/mca/slot7:
Slot: 7
Adapter Name: 3Com 3c523 Etherlink/MC
Id: 6042
Enabled: Yes
POS: 42 60 ff 08 ff ff ff ff
Revision: 0xe
IRQ: 9
IO Address: 0x3300-0x3308
Memory: 0xd8000-0xdbfff
Transceiver: External
Device: eth0
Hardware Address: 02 60 8c 45 c4 2a
```