MCA Driver Programming Interface

Alan Cox

<alan@lxorguk.ukuu.org.uk>

David Weinehall

Chris Beauregard

Table of Contents

Chapter 1. Introduction

The MCA bus functions provide a generalised interface to find MCA bus cards, to claim them for a driver, and to read and manipulate POS registers without being aware of the motherboard internals or certain deep magic specific to onboard devices.

The basic interface to the MCA bus devices is the slot. Each slot is numbered and virtual slot numbers are assigned to the internal devices. Using a pci_dev as other busses do does not really make sense in the MCA context as the MCA bus resources require card specific interpretation.

Finally the MCA bus functions provide a parallel set of DMA functions mimicing the ISA bus DMA functions as closely as possible, although also supporting the additional DMA functionality on the MCA bus controllers.

Chapter 2. Known Bugs And Assumptions

None.

Chapter 3. Public Functions Provided

Table of Contents

Name

mca_find_adapter — scan for adapters

Synopsis

```
int mca_find_adapter (id,
                      start);

int  id;
int  start;
```

Arguments

id

MCA identification to search for

start

starting slot

Description

Search the MCA configuration for adapters matching the 16bit ID given. The first time it should be called with start as zero and then further calls made passing the return value of the previous call until MCA_NOTFOUND is returned.

Disabled adapters are not reported.

Name

mca_find_unused_adapter — scan for unused adapters

Synopsis

```
int mca_find_unused_adapter (id,
                             start);

int  id;
int  start;
```

Arguments

id

MCA identification to search for

start

starting slot

Description

Search the MCA configuration for adapters matching the 16bit ID given. The first time it should be called with start as zero and then further calls made passing the return value of the previous call until MCA_NOTFOUND is returned.

Adapters that have been claimed by drivers and those that are disabled are not reported. This function

thus allows a driver to scan for further cards when some may already be driven.

Name

mca_read_stored_pos — read POS register from boot data

Synopsis

```
unsigned char mca_read_stored_pos (slot,
                                   reg);

int slot;
int reg;
```

Arguments

slot

slot number to read from

reg

register to read from

Description

Fetch a POS value that was stored at boot time by the kernel when it scanned the MCA space. The register value is returned. Missing or invalid registers report 0.

Name

mca_read_pos — read POS register from card

Synopsis

```
unsigned char mca_read_pos (slot,
                            reg);

int slot;
int reg;
```

Arguments

slot

slot number to read from

reg

register to read from

Description

Fetch a POS value directly from the hardware to obtain the current value. This is much slower than mca_read_stored_pos and may not be invoked from interrupt context. It handles the deep magic required for onboard devices transparently.

Name

mca_write_pos — read POS register from card

Synopsis

```
void mca_write_pos (slot,
                    reg,
                    byte);

int            slot;
int            reg;
unsigned char  byte;
```

Arguments

slot

slot number to read from

reg

register to read from

byte

byte to write to the POS registers

Description

Store a POS value directly from the hardware. You should not normally need to use this function and should have a very good knowledge of MCA bus before you do so. Doing this wrongly can damage the hardware.

This function may not be used from interrupt context.

Note that this a technically a Bad Thing, as IBM tech stuff says you should only set POS values through their utilities. However, some devices such as the 3c523 recommend that you write back some data to make sure the configuration is consistent. I'd say that IBM is right, but I like my drivers to work.

This function can't do checks to see if multiple devices end up with the same resources, so you might see magic smoke if someone screws up.

Name

mca_set_adapter_name — Set the description of the card

Synopsis

```
void mca_set_adapter_name (slot,
                           name);

int     slot;
char *  name;
```

Arguments

slot

slot to name

name

text string for the namen

Description

This function sets the name reported via /proc for this adapter slot. This is for user information only. Setting a name deletes any previous name.

Name

mca_mark_as_used — claim an MCA device

Synopsis

```
int mca_mark_as_used (slot);

int  slot;
```

Arguments

slot

slot to claim

FIXME

should we make this threadsafe

Claim an MCA slot for a device driver. If the slot is already taken the function returns 1, if it is not taken it is claimed and 0 is returned.

Name

mca_mark_as_unused — release an MCA device

Synopsis

```
void mca_mark_as_unused (slot);

int  slot;
```

Arguments

slot

slot to claim

Description

Release the slot for other drives to use.

Chapter 4. DMA Functions Provided

Table of Contents

Name

mca_enable_dma — channel to enable DMA on

Synopsis

```
void mca_enable_dma (dmanr);

unsigned int  dmanr;
```

Arguments

dmanr

DMA channel

Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.

Name

mca_disable_dma — channel to disable DMA on

Synopsis

```
void mca_disable_dma (dmanr);

unsigned int  dmanr;
```

Arguments

dmanr

DMA channel

Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.

Name

mca_set_dma_addr — load a 24bit DMA address

Synopsis

```
void mca_set_dma_addr (dmanr,
                       a);

unsigned int  dmanr;
unsigned int  a;
```

Arguments

dmanr

DMA channel

a

24bit bus address

Description

Load the address register in the DMA controller. This has a 24bit limitation (16Mb).

Name

mca_get_dma_addr — load a 24bit DMA address

Synopsis

```
unsigned int mca_get_dma_addr (dmanr);

unsigned int  dmanr;
```

Arguments

dmanr

DMA channel

Description

Read the address register in the DMA controller. This has a 24bit limitation (16Mb). The return is a bus address.

Name

mca_set_dma_count — load a 16bit transfer count

Synopsis

```
void mca_set_dma_count (dmanr,
                        count);

unsigned int  dmanr;
unsigned int  count;
```

Arguments

dmanr

DMA channel

count

count

Description

Set the DMA count for this channel. This can be up to 64Kbytes. Setting a count of zero will not do what you expect.

Name

mca_get_dma_residue — get the remaining bytes to transfer

Synopsis

```
unsigned int mca_get_dma_residue (dmanr);

unsigned int  dmanr;
```

Arguments

dmanr

DMA channel

Description

This function returns the number of bytes left to transfer on this DMA channel.

Name

mca_set_dma_io — set the port for an I/O transfer

Synopsis

```
void mca_set_dma_io (dmanr,
                     io_addr);

unsigned int  dmanr;
unsigned int  io_addr;
```

Arguments

dmanr

DMA channel

io_addr

an I/O port number

Description

Unlike the ISA bus DMA controllers the DMA on MCA bus can transfer with an I/O port target.

Name

mca_set_dma_mode — set the DMA mode

Synopsis

```
void mca_set_dma_mode (dmanr,
                       mode);

unsigned int  dmanr;
unsigned int  mode;
```

Arguments

dmanr

DMA channel

mode

mode to set

Description

The DMA controller supports several modes. The mode values you can set are-

`MCA_DMA_MODE_READ` when reading from the DMA device.

`MCA_DMA_MODE_WRITE` to writing to the DMA device.

`MCA_DMA_MODE_IO` to do DMA to or from an I/O port.

`MCA_DMA_MODE_16` to do 16bit transfers.