

# **MCA Driver Programming Interface**

**Alan Cox**

**`alan@redhat.com`**

**David Weinehall**

**Chris Beauregard**

# **MCA Driver Programming Interface**

by Alan Cox, David Weinehall, and Chris Beauregard

Copyright © 2000 Alan CoxDavid WeinehallChris Beauregard

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Known Bugs And Assumptions .....</b>	<b>3</b>
<b>3. Public Functions Provided .....</b>	<b>5</b>
/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/mca.c .....	5
<b>4. DMA Functions Provided.....</b>	<b>7</b>
mca_enable_dma.....	7
mca_disable_dma.....	7
mca_set_dma_addr .....	8
mca_get_dma_addr .....	9
mca_set_dma_count.....	9
mca_get_dma_residue.....	10
mca_set_dma_io .....	11
mca_set_dma_mode.....	12



# 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

**/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/mca.c**

## Name

`/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/mca.c`  
— Document generation inconsistency

## Oops

### Warning

The template for this document tried to insert the structured comment from the file

`/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/mca.c`  
at this point, but none was found. This dummy section is inserted to allow generation to continue.



# Chapter 4. DMA Functions Provided

## **mca\_enable\_dma**

### **Name**

`mca_enable_dma` — channel to enable DMA on

### **Synopsis**

```
void mca_enable_dma (unsigned int dmanr);
```

### **Arguments**

*dmanr*

DMA channel

### **Description**

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

## **mca\_disable\_dma**

### **Name**

`mca_disable_dma` — channel to disable DMA on

## Synopsis

```
void mca_disable_dma (unsigned int dmanr);
```

## Arguments

*dmanr*

DMA channel

## Description

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

# mca\_set\_dma\_addr

## Name

`mca_set_dma_addr` — load a 24bit DMA address

## Synopsis

```
void mca_set_dma_addr (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).

# mca\_get\_dma\_addr

## Name

`mca_get_dma_addr` — load a 24bit DMA address

## Synopsis

```
unsigned int mca_get_dma_addr (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.

## **mca\_set\_dma\_count**

### **Name**

`mca_set_dma_count` — load a 16bit transfer count

### **Synopsis**

```
void mca_set_dma_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.

## **mca\_get\_dma\_residue**

### **Name**

`mca_get_dma_residue` — get the remaining bytes to transfer

## Synopsis

```
unsigned int mca_get_dma_residue (unsigned int dmanr);
```

## Arguments

*dmanr*

DMA channel

## Description

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

# mca\_set\_dma\_io

## Name

`mca_set_dma_io` — set the port for an I/O transfer

## Synopsis

```
void mca_set_dma_io (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.

# mca\_set\_dma\_mode

## Name

`mca_set_dma_mode` — set the DMA mode

## Synopsis

```
void mca_set_dma_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.

