

AMD 53C974/79C974 SCSI Driver Design Notes

1. General

This driver is primarily intended to be used with the AMD 79C974 PCnet-SCSI combo chip, which combines an Ethernet Controller, a SCSI Host Adapter, and a PCI Bus Interface module on one chip. This driver only controls the SCSI logic on the 79C974; a separate driver is used for the Ethernet logic. The driver also should work with the AMD 53C974, a subset of the 79C974. (This has not been tested as of 25 Jan, 1995.) The 53C974 is, in turn, a superset of the old NCR 53C90A chip, which was the SCSI controller used on all NeXT ("Black") hardware.

This driver implements the following features:

- Disconnect/Reselect
- 10 MB/s "Fast SCSI" Synchronous Transfers
- SCSI-2 Command Queueing
- Auto Request Sense

Synchronous Mode, Command Queueing, and Fast SCSI can be disabled and enabled for the driver as a whole via the driver instance's Instance table. There is a Custom Device Inspector used by the Configure App which allows the user to specify these parameters. Per-target disable flags for Command Queueing and Synchronous Mode are also kept for both of the features, so that if a target rejects an attempt to use the mode (typically via a "Message Reject" message), no further attempts will be made to use the feature with that target. Synchronous Transfer offset and period are also negotiated and kept on a per-target basis, per the SCSI-2 specification.

2. **Architecture**

Unlike most other SCSI Host Adapter drivers for the Intel Platform, this driver is involved in a lot of low-level SCSI events, including dealing with phase changes, setting ATN at appropriate times, doing synchronous transfer negotiation, etc. The driver has to maintain quite a bit of state to keep track of what is occurring on the bus and typically responds to 3 or 4 interrupts for every SCSI command.

The driver is implemented as a subclass of **IOSCSIController** called **AMD_SC SI**. **IOSCSIController** is a subclass of **IODirectDevice**. The **IODirectDevice** categories **IOEISADirectDevice** and **IOPCIDirectDevice** are also used by **AMD_SC SI**.

Control Flow and the I/O thread

The control flow in the driver is the same as for all other NextStep SCSI drivers - all access to the hardware is done by one thread, the I/O thread. This eliminates the need for locks around accesses to the hardware and other critical resources. All communication with the I/O thread by exported methods is performed by passing a command struct, called a **commandBuf**, to the I/O thread via the instance variable **commandQ**. **CommandQ** is protected by **commandLock**. After enqueueing a **commandBuf** on **commandQ**, a message with a **msg_id** of **IO_COMMAND_MSG** is sent to the driver's interrupt port. The I/O thread is the only code which does a **msg_receive()** on the interrupt port. Subsequent to initialization, **commandQ** and **commandLock** are the only data shared by exported methods and the I/O thread.

I/O complete notification is performed via **commandBuf.cmdLock**, which is an **NXConditionLock**. Exported methods wait on this lock after passing a **commandBuf** to the I/O thread.

Subsequent to initialization, the only methods in the driver which do *not* run solely as part of the I/O thread are:

- executeRequest:buffer:client:**
- resetSCSIBus**
- resetStats**
- numQueueSamples**
- sumQueueLengths**
- maxQueueLength**
- executeCmdBuf**

These are all found in **AMD_SCSI.m**. (Note: unless otherwise indicated, all files referred to in this document reside in the AMD53C974SCSIDriver_reloc.tproj directory of the driver project.) The first two methods are exported methods used for normal SCSI I/O and are called by indirect SCSI devices like SCSI disk and SCSTape. The next four are associated with gathering statistics and are called from IOSCSIController. The last one, **-executeCmdBuf**, is the common means by which **-executeRequest:buffer:client:** and **-resetSCSIBus** pass **commandBufs** to the I/O thread and wait for completion.

Processing of commands by the I/O thread

The I/O thread's job is basically to dequeue commands from **commandQ**, start up SCSI transactions on the 79C974, and deal with interrupts. All of the **msg_receive()**s which fetch command requests and interrupt events from the driver's interrupt port are done by **IODirectDevice**'s I/O Thread. If a command message is received, **IODirectDevice** calls **-commandRequestOccurred**. If an interrupt message is received, **-interruptOccurred** is called. Both of these methods are implemented by the **AMD_SCSI** class.

When an exported method passes a **commandBuf** to the I/O thread, it is quite possible that the I/O thread can not immediately process the command due to the fact that there is already a command active on the SCSI bus at that time. If this occurs, the incoming command is enqueued on **pendingQ**. (Note that the decision as to whether a command can be processed, and the act of enqueueing a command on **pendingQ**, are solely the responsibility of the I/O thread. There are no races or deadlock conditions here.) Whenever the I/O thread detects a "Bus Free" condition, it will look at **pendingQ**, and if the queue is non-empty, the first **commandBuf** in the queue is dequeued and used to start up a new SCSI transaction.

The **commandBuf** associated with the currently active SCSI transaction is always kept in **activeCmd**. If **activeCmd** is NULL, the SCSI bus is either free, or there is a reselection in progress for which we do not have

complete target/LUN/queue tag information.

When an active SCSI target disconnects (as opposed to doing a "Command Complete"), the I/O thread places the associated **commandBuf** on **disconnectQ**. Whenever a reselection occurs, **disconnectQ** is scanned for a **commandBuf** which matches the reselecting target, LUN, and (possibly) queue tag. If a match is found, the **commandBuf** is dequeued from **disconnectQ** and becomes **activeCmd**.

The driver contains logic to prevent sending a command to a target/LUN nexus which currently has an active, but disconnected, command - unless command queueing is enabled for the driver, and is not disabled for the given target. This allows higher layers in the system (e.g., SCSI Disk) to enqueue multiple requests for one target and LUN without worrying about the nexus's current state. This logic uses the array **activeArray**[], which is an array of counters which keep track of how many I/Os are active on a per-LUN basis. When command queueing is disabled (either globally, or on a per-target basis), an **activeArray** counter has a maximum value of 1. If command queueing is enabled, an **activeArray** counter has a maximum value of the target's queue length (if known). See the **-cmdBufOk:** method, in AMD_SCSI.m, for the implementation of the decision as to whether a target/lun nexus is ready to accept a new command.

3. Organization

The driver was organized to facilitate porting to other platforms using chips similar to the 79C974, and also to other "dumb" host adapter chips on the x86 platform. With some exceptions, all of the logic which is independent of either the chip or the host bus is contained in **AMD_SCSI.m**. All of the logic which deals directly with the 79C974 is contained in **AMD_Chip.m** and **AMD_ChipPrivate.m**. All of the logic which is specific to the x86 architecture and the PCI bus is contained in **AMD_x86.m**. The API to **AMD_Chip.m** which is public to the other parts of the driver consists of six methods:

- **-probeChip**, performs one-time-only initialization.
- **-hwReset**, reusable chip reset/init.
- **-scsiReset**, performs SCSI reset.
- **-hwStart:**, starts up a new SCSI transaction.
- **-hwInterrupt**, deals with an interrupt event.
- **-logRegs**, a debugging function to dump registers to the console.

When porting this driver to a new chip, these six methods are pretty much all that need to be re-implemented (in

addition to any private, implementation-specific methods needed for the new chip).

The design goal of separating out the chip-specific, architecture-specific, and hardware-independent functionality into separate modules was not religiously adhered to. Sometimes, for example in the routines to handle DMA in **AMD_x86.m**, there is a mixture of chip- and bus-specific functions in one file (or even one method). The tradeoff between design clarity and portability went towards design clarity in these cases.

4. File contents

The following files all exist in the AMD53C974SCSISDriver_reloc.tproj directory. (AMD53C974SCSISDriver_reloc is the driver's loadable binary.)

AMD_SCSI.h

Exported interface to the driver. Typically used by SCSI indirect drivers like SCSIIDisk and SCISITape.

AMD_Private.h

Private hardware-independent methods.

AMD_Types.h

Private structs and #defines used by the entire driver.

AMD_SCSI.m

Implementation of AMD_SCSI.h and AMD_Private.h methods.

AMD_x86.[hm]

Methods specific to Intel platform and PCI bus.

AMD_Chip.[hm]

Chip-specific methods available to the rest of the driver.

AMD_ChipPrivate.m

Chip-specific methods available only to AMD_Chip.m.

AMD_Regs.h

Definitions of AMD 79C974 registers.

bringup.h

Debugging and bringup flags.

AMD_ddm.h

#defines used for DDM calls.

AMD.ddm

Config file used by DDMViewer.

ioPorts.[ch]

A DEBUG version of <driverkit/i386/ioPorts.h>.

pciConf.h

#defines for PCI configuration registers.

configKeys.h

String definition for keys in driver's config table.

Makefile.driver_preamble

Makefile.preamble

Makefile.postamble

Standard Makefile additions, common to all drivers.

Other Files

Here is a brief description of the other files residing in the driver project's root directory.

SCSIInspector.[mh]

Custom Device Inspector for use in the Configure App.

Default.table

Template used by Configure App for creating Instance tables.

English.lproj/AMDInspector.nib

Localizable nib file for Custom Device Inspector.

English.lproj/Localizable.strings

Localizable strings file used by Configure App.

English.lproj/DriverHelp/*

Support for on-line help in the Configure App.

Makefile

PB.project

Created by Project Builder.

SGS_ENV

For NeXT internal use.

Makefile.postamble

Makefile.preamble

Standard Makefile additions, common to all drivers.

changes

Revision history of this project.

README.rtf

This file.