

Adaptec 154x SCSI Host Adaptor Driver

This directory contains the all of the source files needed to build a driver for the Adaptec 1542B/C series of SCSI Host Adapters.

Files

The following files comprise this driver.

PB.project	Used by Project Builder.
Makefile	Created by Project Builder; used by 'make'.
Makefile.preamble	Standard boilerplate driverkit Makefile extension.
Makefile.postamble	Standard boilerplate driverkit Makefile extension.
Default.table	Default table file, used by Configure app.
English.lproj/Localizable.strings	Used by Configure app.
README.rtf	This file.

These files compile to build the actual relocatable driver binary. They reside in the Adaptec1542B_reloc.tproj subdirectory.

AHAController.h	Public class declaration.
AHAController.m	Implementation of public and common methods.
AHAThread.h	I/O thread definitions.
AHAThread.m	I/O Thread implementation.
AHARoutines.m	Common low-level I/O routines.
AHATypes.h	H/W defintions for Adaptec 1542B Host Adaptor.
AHAControllerPrivate.h	Driver-private typedefs.
AHAInline.h	Static inlines and macros.
scsivar.h	Device-independent SCSI definitions.
AHA.ddm	DDM defintions (for debugging only).
Default.table	Default config table.
Load_Commands.sect	Standard boilerplate driverkit tool.
Makefile	Builds the driver.

Architecture

This driver uses one I/O thread to perform all hardware operations which occur subsequent to +probe time. This eliminates the need for using locking protocols to gain access to the hardware or related resources. The I/O thread is actually implemented in IODirectDevice; it is started up by our superclass, IO SCSIController, in the -initFromDeviceDescription method.

Requests are passed from public methods (-executeRequest:buffer:client and -resetSCSIBus) to the I/O thread

via the struct `AHACmdBuf`, defined in `AHAControllerPrivate.h`. These `AHACmdBuf`'s are enqueued on the instance variable `commandQ` in the common method `-executeCmdBuf`; after enqueueing, a message with `msg_id` of `IO_COMMAND_MSG` is sent to the I/O thread (via the `interruptPort`). This wakes up the I/O thread, resulting in a call to our `-commandRequestOccurred` method. Command requests are then dispatched from `-commandRequestOccurred` to the appropriate command-specific code in `AHAThread.m`. During the execution of a command, `-executeCmdBuf` is waiting on the `cmdLock` field (an `NXConditionLock`) in `AHACmdBuf`. When the I/O thread completes a command, it unlocks the associated `cmdLock` with `condition = CMD_COMPLETE`; this wakes up `-executeCmdBuf`, which returns to the originating caller.

This implementation, using `AHACmdBuf`'s, is a textbook case of the mechanism described in the Driverkit Technical Documentation, Chapter 2 ("Designing a Driver"), in the section entitled "Communicating using Condition Locks".

The methods in this class can be grouped into the following 4 categories:

Initialization

The `+probe` and `-initWithDeviceDescription` methods each run once per hardware instance. They run in the I/O task context, but not in our I/O thread context.

Public Exported Methods

These consist of `-executeRequest:buffer:client` and `-resetSCSIBus`. Each of these merely create an `AHACmdBuf` and call `-executeCmdBuf`, which passes the `AHACmdBuf` over to the I/O thread for processing.

Methods Called by IODevice's I/O Thread

These methods are implemented in `AHAController.m` for a consistent API (since they are called from outside of the class), but they run in the I/O thread context. The three methods of interest are:

`-interruptOccurred`: a hardware interrupt has been detected. This generally means an I/O is complete.

`-timeoutOccurred`: A pending I/O has timed out. This is detected by a callout to the function `ahaTimeout()`, in `AHAThread.m`. A callout is scheduled to this function when a command is sent to the host adaptor in the form of a CCB. If the timeout specified in `IO SCSIRequest.timeoutLength` elapses without an I/O complete for the associated CCB, `ahaTimeout()` is called. `AhaTimeout()` then sends a message with `msg_id = IO_TIMEOUT_MSG` to the I/O thread via the `interruptPort`. The I/O thread then calls `-timeoutOccurred`. We abort the offending command with `driverStatus = SR_IOST_IOTO`, reset the SCSI bus, and abort all other pending commands with `driverStatus = SR_IOST_RESET`.

`-commandRequestOccurred`: This is called by the I/O thread when it receives a message on `interruptPort` with `msg_id = IO_COMMAND_MSG`. Our response is to dequeue all of the commands we find on `commandQ` and

send them to the host adaptor. One possible problem: the Host Adaptor's input queue (i.e., its array of mailboxes) might be full. In that case, `commandRequestOccurred` leaves the new I/O request(s) on `commandQ`. Later on, when we detect an I/O complete - thus freeing up one of the Host Adaptor's mailboxes, we call `-commandRequestOccurred` on ourself to process any possible pending commands in `commandQ`.

Private I/O thread methods

These methods, implemented in `AHAThread.m`, perform the bulk of the hardware manipulation. All of these methods run solely in the context of the I/O thread.

Debugging

This driver has been extensively instrumented with DDM calls. The file `AHA.ddm` can be loaded into `DDMViewer` (currently in `/NextDeveloper/Demos`) to allow selective access to various events in the driver. See the `Driverkit` Technical Documentation for more information on `DDMViewer` and the DDM module in general. Note that the DDM calls are only enabled in the `DEBUG` build configuration.