Linux Scsi Documentation

Release 6.8.0

The kernel development community

Jan 16, 2026

CONTENTS

1 Introduction 1

2 SCSI driver APIs 3

3 SCSI driver parameters 37

4 SCSI host adapter drivers 41

CHAPTER

ONE

INTRODUCTION

1.1 SCSI subsystem documentation

The Linux Documentation Project (LDP) maintains a document describing the SCSI subsystem in the Linux kernel (lk) 2.4 series. See: https://www.tldp.org/HOWTO/SCSI-2.4-HOWTO . The LDP has single and multiple page HTML renderings as well as postscript and pdf.

1.1.1 Notes on using modules in the SCSI subsystem

The SCSI support in the Linux kernel can be modularized in a number of different ways depending upon the needs of the end user. To understand your options, we should first define a few terms.

The scsi-core (also known as the "mid level") contains the core of SCSI support. Without it you can do nothing with any of the other SCSI drivers. The SCSI core support can be a module (scsi_mod.o), or it can be built into the kernel. If the core is a module, it must be the first SCSI module loaded, and if you unload the modules, it will have to be the last one unloaded. In practice the modprobe and rmmod commands will enforce the correct ordering of loading and unloading modules in the SCSI subsystem.

The individual upper and lower level drivers can be loaded in any order once the SCSI core is present in the kernel (either compiled in or loaded as a module). The disk driver (sd_mod.o), CD-ROM driver (sr_mod.o), tape driver[1] (st.o) and SCSI generics driver (sg.o) represent the upper level drivers to support the various assorted devices which can be controlled. You can for example load the tape driver to use the tape drive, and then unload it once you have no further need for the driver (and release the associated memory).

The lower level drivers are the ones that support the individual cards that are supported for the hardware platform that you are running under. Those individual cards are often called Host Bus Adapters (HBAs). For example the aic7xxx.o driver is used to control all recent SCSI controller cards from Adaptec. Almost all lower level drivers can be built either as modules or built into the kernel.

[1] There is a variant of the st driver for controlling OnStream tape devices. Its module name is osst.o .

CHAPTER

TWO

SCSI DRIVER APIS

2.1 SCSI mid_level - lower_level driver interface

2.1.1 Introduction

This document outlines the interface between the Linux SCSI mid level and SCSI lower level drivers. Lower level drivers (LLDs) are variously called host bus adapter (HBA) drivers and host drivers (HD). A ”host” in this context is a bridge between a computer IO bus (e.g. PCI or ISA) and a single SCSI initiator port on a SCSI transport. An ”initiator” port (SCSI terminology, see SAM-3 at http://www.t10.org) sends SCSI commands to ”target” SCSI ports (e.g. disks). There can be many LLDs in a running system, but only one per hardware type. Most LLDs can control one or more SCSI HBAs. Some HBAs contain multiple hosts.

In some cases the SCSI transport is an external bus that already has its own subsystem in Linux (e.g. USB and ieee1394). In such cases the SCSI subsystem LLD is a software bridge to the other driver subsystem. Examples are the usb-storage driver (found in the drivers/usb/storage directory) and the ieee1394/sbp2 driver (found in the drivers/ieee1394 directory).

For example, the aic7xxx LLD controls Adaptec SCSI parallel interface (SPI) controllers based on that company's 7xxx chip series. The aic7xxx LLD can be built into the kernel or loaded as a module. There can only be one aic7xxx LLD running in a Linux system but it may be controlling many HBAs. These HBAs might be either on PCI daughter-boards or built into the motherboard (or both). Some aic7xxx based HBAs are dual controllers and thus represent two hosts. Like most modern HBAs, each aic7xxx host has its own PCI device address. [The one-to-one correspondence between a SCSI host and a PCI device is common but not required (e.g. with ISA adapters).]

The SCSI mid level isolates an LLD from other layers such as the SCSI upper layer drivers and the block layer.

This version of the document roughly matches linux kernel version 2.6.8 .

2.1.2 Documentation

There is a SCSI documentation directory within the kernel source tree, typically Documentation/scsi . Most documents are in plain (i.e. ASCII) text. This file is named *SCSI mid_level - lower_level driver interface* and can be found in that directory. A more recent copy of this document may be found at http://web.archive.org/web/20070107183357rn_1/sg.torque.net/scsi/. Many LLDs are documented there (e.g. *Adaptec Aic7xxx Fast -> Ultra160 Family Manager Set v7.0*). The SCSI mid-level is briefly described in *SCSI subsystem documentation* which contains a url to a document describing the SCSI subsystem in the lk 2.4 series. Two upper level drivers have documents in that directory: *The SCSI Tape Driver* (SCSI tape driver) and *SCSI Generic (sg) driver* (for the sg driver).

Some documentation (or urls) for LLDs may be found in the C source code or in the same directory as the C source code. For example to find a url about the USB mass storage driver see the /usr/src/linux/drivers/usb/storage directory.

2.1.3 Driver structure

Traditionally an LLD for the SCSI subsystem has been at least two files in the drivers/scsi directory. For example, a driver called ”xyz” has a header file ”xyz.h” and a source file ”xyz.c”. [Actually there is no good reason why this couldn't all be in one file; the header file is superfluous.] Some drivers that have been ported to several operating systems have more than two files. For example the aic7xxx driver has separate files for generic and OS-specific code (e.g. FreeBSD and Linux). Such drivers tend to have their own directory under the drivers/scsi directory.

When a new LLD is being added to Linux, the following files (found in the drivers/scsi directory) will need some attention: Makefile and Kconfig . It is probably best to study how existing LLDs are organized.

As the 2.5 series development kernels evolve into the 2.6 series production series, changes are being introduced into this interface. An example of this is driver initialization code where there are now 2 models available. The older one, similar to what was found in the lk 2.4 series, is based on hosts that are detected at HBA driver load time. This will be referred to the ”passive” initialization model. The newer model allows HBAs to be hot plugged (and unplugged) during the lifetime of the LLD and will be referred to as the ”hotplug” initialization model. The newer model is preferred as it can handle both traditional SCSI equipment that is permanently connected as well as modern ”SCSI” devices (e.g. USB or IEEE 1394 connected digital cameras) that are hotplugged. Both initialization models are discussed in the following sections.

An LLD interfaces to the SCSI subsystem several ways:

a) directly invoking functions supplied by the mid level

b) passing a set of function pointers to a registration function supplied by the mid level. The mid level will then invoke these functions at some point in the future. The LLD will supply implementations of these functions.

c) direct access to instances of well known data structures maintained by the mid level

Those functions in group a) are listed in a section entitled ”Mid level supplied functions” below.

Those functions in group b) are listed in a section entitled ”Interface functions” below. Their function pointers are placed in the members of ”struct scsi_host_template”, an instance of which

is passed to scsi_host_alloc()[1]. Those interface functions that the LLD does not wish to supply should have NULL placed in the corresponding member of struct scsi_host_template. Defining an instance of struct scsi_host_template at file scope will cause NULL to be placed in function pointer members not explicitly initialized.

Those usages in group c) should be handled with care, especially in a ”hotplug” environment. LLDs should be aware of the lifetime of instances that are shared with the mid level and other layers.

All functions defined within an LLD and all data defined at file scope should be static. For example the slave_alloc() function in an LLD called ”xxx” could be defined as `static int xxx_slave_alloc(struct scsi_device * sdev) { /* code */ }`

2.1.4 Hotplug initialization model

In this model an LLD controls when SCSI hosts are introduced and removed from the SCSI subsystem. Hosts can be introduced as early as driver initialization and removed as late as driver shutdown. Typically a driver will respond to a sysfs probe() callback that indicates an HBA has been detected. After confirming that the new device is one that the LLD wants to control, the LLD will initialize the HBA and then register a new host with the SCSI mid level.

During LLD initialization the driver should register itself with the appropriate IO bus on which it expects to find HBA(s) (e.g. the PCI bus). This can probably be done via sysfs. Any driver parameters (especially those that are writable after the driver is loaded) could also be registered with sysfs at this point. The SCSI mid level first becomes aware of an LLD when that LLD registers its first HBA.

At some later time, the LLD becomes aware of an HBA and what follows is a typical sequence of calls between the LLD and the mid level. This example shows the mid level scanning the newly introduced HBA for 3 scsi devices of which only the first 2 respond:

```
    HBA PROBE: assume 2 SCSI devices found in scan
LLD                  mid level                    LLD
===-------------------========---------------------===------
scsi_host_alloc()  -->
scsi_add_host()  ---->
scsi_scan_host()  -------+
                         |
                     slave_alloc()
                     slave_configure() -->  scsi_change_queue_depth()
                         |
                     slave_alloc()
                     slave_configure()
                         |
                     slave_alloc()   ***
                     slave_destroy() ***


*** For scsi devices that the mid level tries to scan but do not
    respond, a slave_alloc(), slave_destroy() pair is called.
```

[1] the scsi_host_alloc() function is a replacement for the rather vaguely named scsi_register() function in most situations.

If the LLD wants to adjust the default queue settings, it can invoke scsi_change_queue_depth() in its slave_configure() routine.

When an HBA is being removed it could be as part of an orderly shutdown associated with the LLD module being unloaded (e.g. with the "rmmod" command) or in response to a "hot unplug" indicated by sysfs()'s remove() callback being invoked. In either case, the sequence is the same:

```
        HBA REMOVE: assume 2 SCSI devices attached
LLD                      mid level                  LLD
===----------------------========-----------------===------
scsi_remove_host() ---------+
                            |
                     slave_destroy()
                     slave_destroy()
scsi_host_put()
```

It may be useful for a LLD to keep track of struct Scsi_Host instances (a pointer is returned by scsi_host_alloc()). Such instances are "owned" by the mid-level. struct Scsi_Host instances are freed from scsi_host_put() when the reference count hits zero.

Hot unplugging an HBA that controls a disk which is processing SCSI commands on a mounted file system is an interesting situation. Reference counting logic is being introduced into the mid level to cope with many of the issues involved. See the section on reference counting below.

The hotplug concept may be extended to SCSI devices. Currently, when an HBA is added, the scsi_scan_host() function causes a scan for SCSI devices attached to the HBA's SCSI transport. On newer SCSI transports the HBA may become aware of a new SCSI device _after_ the scan has completed. An LLD can use this sequence to make the mid level aware of a SCSI device:

```
                SCSI DEVICE hotplug
LLD                   mid level                    LLD
===-------------------========--------------------===------
scsi_add_device()  ------+
                         |
                     slave_alloc()
                     slave_configure()   [--> scsi_change_queue_depth()]
```

In a similar fashion, an LLD may become aware that a SCSI device has been removed (unplugged) or the connection to it has been interrupted. Some existing SCSI transports (e.g. SPI) may not become aware that a SCSI device has been removed until a subsequent SCSI command fails which will probably cause that device to be set offline by the mid level. An LLD that detects the removal of a SCSI device can instigate its removal from upper layers with this sequence:

```
                SCSI DEVICE hot unplug
LLD                      mid level                  LLD
===----------------------========-----------------===------
scsi_remove_device() -------+
                            |
                     slave_destroy()
```

It may be useful for an LLD to keep track of struct scsi_device instances (a pointer is passed as the parameter to slave_alloc() and slave_configure() callbacks). Such instances are "owned" by the mid-level. struct scsi_device instances are freed after slave_destroy().

2.1.5 Reference Counting

The Scsi_Host structure has had reference counting infrastructure added. This effectively spreads the ownership of struct Scsi_Host instances across the various SCSI layers which use them. Previously such instances were exclusively owned by the mid level. LLDs would not usually need to directly manipulate these reference counts but there may be some cases where they do.

There are 3 reference counting functions of interest associated with struct Scsi_Host:

- **scsi_host_alloc():**
  returns a pointer to new instance of struct Scsi_Host which has its reference count ^^ set to 1
- **scsi_host_get():**
  adds 1 to the reference count of the given instance
- **scsi_host_put():**
  decrements 1 from the reference count of the given instance. If the reference count reaches 0 then the given instance is freed

The scsi_device structure has had reference counting infrastructure added. This effectively spreads the ownership of struct scsi_device instances across the various SCSI layers which use them. Previously such instances were exclusively owned by the mid level. See the access functions declared towards the end of include/scsi/scsi_device.h . If an LLD wants to keep a copy of a pointer to a scsi_device instance it should use scsi_device_get() to bump its reference count. When it is finished with the pointer it can use scsi_device_put() to decrement its reference count (and potentially delete it).

Note: struct Scsi_Host actually has 2 reference counts which are manipulated in parallel by these functions.

2.1.6 Conventions

First, Linus Torvalds's thoughts on C coding style can be found in the Documentation/process/coding-style.rst file.

Also, most C99 enhancements are encouraged to the extent they are supported by the relevant gcc compilers. So C99 style structure and array initializers are encouraged where appropriate. Don't go too far, VLAs are not properly supported yet. An exception to this is the use of `//` style comments; `/*...*/` comments are still preferred in Linux.

Well written, tested and documented code, need not be re-formatted to comply with the above conventions. For example, the aic7xxx driver comes to Linux from FreeBSD and Adaptec's own labs. No doubt FreeBSD and Adaptec have their own coding conventions.

2.1.7 Mid level supplied functions

These functions are supplied by the SCSI mid level for use by LLDs. The names (i.e. entry points) of these functions are exported so an LLD that is a module can access them. The kernel will arrange for the SCSI mid level to be loaded and initialized before any LLD is initialized. The functions below are listed alphabetically and their names all start with `scsi_`.

Summary:

- scsi_add_device - creates new scsi device (lu) instance
- scsi_add_host - perform sysfs registration and set up transport class
- scsi_change_queue_depth - change the queue depth on a SCSI device
- scsi_bios_ptable - return copy of block device's partition table
- scsi_block_requests - prevent further commands being queued to given host
- scsi_host_alloc - return a new scsi_host instance whose refcount==1
- scsi_host_get - increments Scsi_Host instance's refcount
- scsi_host_put - decrements Scsi_Host instance's refcount (free if 0)
- scsi_register - create and register a scsi host adapter instance.
- scsi_remove_device - detach and remove a SCSI device
- scsi_remove_host - detach and remove all SCSI devices owned by host
- scsi_report_bus_reset - report scsi _bus_ reset observed
- scsi_scan_host - scan SCSI bus
- scsi_track_queue_full - track successive QUEUE_FULL events
- scsi_unblock_requests - allow further commands to be queued to given host
- scsi_unregister - [calls scsi_host_put()]

Details:

```
/**
* scsi_add_device - creates new scsi device (lu) instance
* @shost:   pointer to scsi host instance
* @channel: channel number (rarely other than 0)
* @id:      target id number
* @lun:     logical unit number
*
*      Returns pointer to new struct scsi_device instance or
*      ERR_PTR(-ENODEV) (or some other bent pointer) if something is
*      wrong (e.g. no lu responds at given address)
*
*      Might block: yes
*
*      Notes: This call is usually performed internally during a scsi
*      bus scan when an HBA is added (i.e. scsi_scan_host()). So it
*      should only be called if the HBA becomes aware of a new scsi
*      device (lu) after scsi_scan_host() has completed. If successful
```

```
*      this call can lead to slave_alloc() and slave_configure() callbacks
*      into the LLD.
*
*      Defined in: drivers/scsi/scsi_scan.c
**/
struct scsi_device * scsi_add_device(struct Scsi_Host *shost,
                                     unsigned int channel,
                                     unsigned int id, unsigned int lun)


/**
* scsi_add_host - perform sysfs registration and set up transport class
* @shost:   pointer to scsi host instance
* @dev:     pointer to struct device of type scsi class
*
*      Returns 0 on success, negative errno of failure (e.g. -ENOMEM)
*
*      Might block: no
*
*      Notes: Only required in "hotplug initialization model" after a
*      successful call to scsi_host_alloc().  This function does not
*   scan the bus; this can be done by calling scsi_scan_host() or
*   in some other transport-specific way.  The LLD must set up
*   the transport template before calling this function and may only
*   access the transport class data after this function has been called.
*
*      Defined in: drivers/scsi/hosts.c
**/
int scsi_add_host(struct Scsi_Host *shost, struct device * dev)


/**
* scsi_change_queue_depth - allow LLD to change queue depth on a SCSI device
* @sdev:       pointer to SCSI device to change queue depth on
* @tags        Number of tags allowed if tagged queuing enabled,
*              or number of commands the LLD can queue up
*              in non-tagged mode (as per cmd_per_lun).
*
*      Returns nothing
*
*      Might block: no
*
*      Notes: Can be invoked any time on a SCSI device controlled by this
*      LLD. [Specifically during and after slave_configure() and prior to
*      slave_destroy().] Can safely be invoked from interrupt code.
*
*      Defined in: drivers/scsi/scsi.c [see source code for more notes]
*
**/
int scsi_change_queue_depth(struct scsi_device *sdev, int tags)
```

```
/**
* scsi_bios_ptable - return copy of block device's partition table
* @dev:        pointer to block device
*
*      Returns pointer to partition table, or NULL for failure
*
*      Might block: yes
*
*      Notes: Caller owns memory returned (free with kfree() )
*
*      Defined in: drivers/scsi/scsicam.c
**/
unsigned char *scsi_bios_ptable(struct block_device *dev)


/**
* scsi_block_requests - prevent further commands being queued to given host
*
* @shost: pointer to host to block commands on
*
*      Returns nothing
*
*      Might block: no
*
*      Notes: There is no timer nor any other means by which the requests
*      get unblocked other than the LLD calling scsi_unblock_requests().
*
*      Defined in: drivers/scsi/scsi_lib.c
**/
void scsi_block_requests(struct Scsi_Host * shost)


/**
* scsi_host_alloc - create a scsi host adapter instance and perform basic
*                   initialization.
* @sht:        pointer to scsi host template
* @privsize:   extra bytes to allocate in hostdata array (which is the
*              last member of the returned Scsi_Host instance)
*
*      Returns pointer to new Scsi_Host instance or NULL on failure
*
*      Might block: yes
*
*      Notes: When this call returns to the LLD, the SCSI bus scan on
*      this host has _not_ yet been done.
*      The hostdata array (by default zero length) is a per host scratch
*      area for the LLD's exclusive use.
*      Both associated refcounting objects have their refcount set to 1.
```

```
*      Full registration (in sysfs) and a bus scan are performed later when
*      scsi_add_host() and scsi_scan_host() are called.
*
*      Defined in: drivers/scsi/hosts.c .
**/
struct Scsi_Host * scsi_host_alloc(const struct scsi_host_template * sht,
                                   int privsize)


/**
* scsi_host_get - increment Scsi_Host instance refcount
* @shost:   pointer to struct Scsi_Host instance
*
*      Returns nothing
*
*      Might block: currently may block but may be changed to not block
*
*      Notes: Actually increments the counts in two sub-objects
*
*      Defined in: drivers/scsi/hosts.c
**/
void scsi_host_get(struct Scsi_Host *shost)


/**
* scsi_host_put - decrement Scsi_Host instance refcount, free if 0
* @shost:   pointer to struct Scsi_Host instance
*
*      Returns nothing
*
*      Might block: currently may block but may be changed to not block
*
*      Notes: Actually decrements the counts in two sub-objects. If the
*      latter refcount reaches 0, the Scsi_Host instance is freed.
*      The LLD need not worry exactly when the Scsi_Host instance is
*      freed, it just shouldn't access the instance after it has balanced
*      out its refcount usage.
*
*      Defined in: drivers/scsi/hosts.c
**/
void scsi_host_put(struct Scsi_Host *shost)


/**
* scsi_register - create and register a scsi host adapter instance.
* @sht:        pointer to scsi host template
* @privsize:   extra bytes to allocate in hostdata array (which is the
*              last member of the returned Scsi_Host instance)
*
*      Returns pointer to new Scsi_Host instance or NULL on failure
```

```
*
*      Might block: yes
*
*      Notes: When this call returns to the LLD, the SCSI bus scan on
*      this host has _not_ yet been done.
*      The hostdata array (by default zero length) is a per host scratch
*      area for the LLD.
*
*      Defined in: drivers/scsi/hosts.c .
**/
struct Scsi_Host * scsi_register(struct scsi_host_template * sht,
                                 int privsize)


/**
* scsi_remove_device - detach and remove a SCSI device
* @sdev:      a pointer to a scsi device instance
*
*      Returns value: 0 on success, -EINVAL if device not attached
*
*      Might block: yes
*
*      Notes: If an LLD becomes aware that a scsi device (lu) has
*      been removed but its host is still present then it can request
*      the removal of that scsi device. If successful this call will
*      lead to the slave_destroy() callback being invoked. sdev is an
*      invalid pointer after this call.
*
*      Defined in: drivers/scsi/scsi_sysfs.c .
**/
int scsi_remove_device(struct scsi_device *sdev)


/**
* scsi_remove_host - detach and remove all SCSI devices owned by host
* @shost:      a pointer to a scsi host instance
*
*      Returns value: 0 on success, 1 on failure (e.g. LLD busy ??)
*
*      Might block: yes
*
*      Notes: Should only be invoked if the "hotplug initialization
*      model" is being used. It should be called _prior_ to
*      scsi_unregister().
*
*      Defined in: drivers/scsi/hosts.c .
**/
int scsi_remove_host(struct Scsi_Host *shost)
```

```
/**
* scsi_report_bus_reset - report scsi _bus_ reset observed
* @shost: a pointer to a scsi host involved
* @channel: channel (within) host on which scsi bus reset occurred
*
*      Returns nothing
*
*      Might block: no
*
*      Notes: This only needs to be called if the reset is one which
*      originates from an unknown location.  Resets originated by the
*      mid level itself don't need to call this, but there should be
*      no harm.  The main purpose of this is to make sure that a
*      CHECK_CONDITION is properly treated.
*
*      Defined in: drivers/scsi/scsi_error.c .
**/
void scsi_report_bus_reset(struct Scsi_Host * shost, int channel)


/**
* scsi_scan_host - scan SCSI bus
* @shost: a pointer to a scsi host instance
*
*   Might block: yes
*
*   Notes: Should be called after scsi_add_host()
*
*   Defined in: drivers/scsi/scsi_scan.c
**/
void scsi_scan_host(struct Scsi_Host *shost)


/**
* scsi_track_queue_full - track successive QUEUE_FULL events on given
*                       device to determine if and when there is a need
*                       to adjust the queue depth on the device.
* @sdev:  pointer to SCSI device instance
* @depth: Current number of outstanding SCSI commands on this device,
*         not counting the one returned as QUEUE_FULL.
*
*      Returns 0  - no change needed
*              >0 - adjust queue depth to this new depth
*              -1 - drop back to untagged operation using host->cmd_per_lun
*                   as the untagged command depth
*
*      Might block: no
*
*      Notes: LLDs may call this at any time and we will do "The Right
*             Thing"; interrupt context safe.
```

```
*
*      Defined in: drivers/scsi/scsi.c .
**/
int scsi_track_queue_full(struct scsi_device *sdev, int depth)


/**
* scsi_unblock_requests - allow further commands to be queued to given host
*
* @shost: pointer to host to unblock commands on
*
*      Returns nothing
*
*      Might block: no
*
*      Defined in: drivers/scsi/scsi_lib.c .
**/
void scsi_unblock_requests(struct Scsi_Host * shost)


/**
* scsi_unregister - unregister and free memory used by host instance
* @shp:        pointer to scsi host instance to unregister.
*
*      Returns nothing
*
*      Might block: no
*
*      Notes: Should not be invoked if the "hotplug initialization
*      model" is being used. Called internally by exit_this_scsi_driver()
*      in the "passive initialization model". Hence a LLD has no need to
*      call this function directly.
*
*      Defined in: drivers/scsi/hosts.c .
**/
void scsi_unregister(struct Scsi_Host * shp)
```

2.1.8 Interface Functions

Interface functions are supplied (defined) by LLDs and their function pointers are placed in an instance of struct scsi_host_template which is passed to scsi_host_alloc() [or scsi_register() / init_this_scsi_driver()]. Some are mandatory. Interface functions should be declared static. The accepted convention is that driver ”xyz” will declare its slave_configure() function as:

```
static int xyz_slave_configure(struct scsi_device * sdev);
```

and so forth for all interface functions listed below.

A pointer to this function should be placed in the 'slave_configure' member of a ”struct scsi_host_template” instance. A pointer to such an instance should be passed to the mid level's

scsi_host_alloc() [or scsi_register() / init_this_scsi_driver()].

The interface functions are also described in the include/scsi/scsi_host.h file immediately above their definition point in "struct scsi_host_template". In some cases more detail is given in scsi_host.h than below.

The interface functions are listed below in alphabetical order.

Summary:

- bios_param - fetch head, sector, cylinder info for a disk
- eh_timed_out - notify the host that a command timer expired
- eh_abort_handler - abort given command
- eh_bus_reset_handler - issue SCSI bus reset
- eh_device_reset_handler - issue SCSI device reset
- eh_host_reset_handler - reset host (host bus adapter)
- info - supply information about given host
- ioctl - driver can respond to ioctls
- proc_info - supports /proc/scsi/{driver_name}/{host_no}
- queuecommand - queue scsi command, invoke 'done' on completion
- slave_alloc - prior to any commands being sent to a new device
- slave_configure - driver fine tuning for given device after attach
- slave_destroy - given device is about to be shut down

Details:

```
/**
*      bios_param - fetch head, sector, cylinder info for a disk
*      @sdev: pointer to scsi device context (defined in
*             include/scsi/scsi_device.h)
*      @bdev: pointer to block device context (defined in fs.h)
*      @capacity:  device size (in 512 byte sectors)
*      @params: three element array to place output:
*               params[0] number of heads (max 255)
*               params[1] number of sectors (max 63)
*               params[2] number of cylinders
*
*      Return value is ignored
*
*      Locks: none
*
*      Calling context: process (sd)
*
*      Notes: an arbitrary geometry (based on READ CAPACITY) is used
*      if this function is not provided. The params array is
*      pre-initialized with made up values just in case this function
*      doesn't output anything.
*
```

```
*      Optionally defined in: LLD
**/
    int bios_param(struct scsi_device * sdev, struct block_device *bdev,
                sector_t capacity, int params[3])


/**
*      eh_timed_out - The timer for the command has just fired
*      @scp: identifies command timing out
*
*      Returns:
*
*      EH_HANDLED:             I fixed the error, please complete the command
*      EH_RESET_TIMER:         I need more time, reset the timer and
*                              begin counting again
*      EH_NOT_HANDLED          Begin normal error recovery
*
*
*      Locks: None held
*
*      Calling context: interrupt
*
*      Notes: This is to give the LLD an opportunity to do local recovery.
*      This recovery is limited to determining if the outstanding command
*      will ever complete.  You may not abort and restart the command from
*      this callback.
*
*      Optionally defined in: LLD
**/
    int eh_timed_out(struct scsi_cmnd * scp)


/**
*      eh_abort_handler - abort command associated with scp
*      @scp: identifies command to be aborted
*
*      Returns SUCCESS if command aborted else FAILED
*
*      Locks: None held
*
*      Calling context: kernel thread
*
*      Notes: If 'no_async_abort' is defined this callback
*   will be invoked from scsi_eh thread. No other commands
*   will then be queued on current host during eh.
*   Otherwise it will be called whenever scsi_timeout()
*      is called due to a command timeout.
*
*      Optionally defined in: LLD
**/
```

```
    int eh_abort_handler(struct scsi_cmnd * scp)


/**
*      eh_bus_reset_handler - issue SCSI bus reset
*      @scp: SCSI bus that contains this device should be reset
*
*      Returns SUCCESS if command aborted else FAILED
*
*      Locks: None held
*
*      Calling context: kernel thread
*
*      Notes: Invoked from scsi_eh thread. No other commands will be
*      queued on current host during eh.
*
*      Optionally defined in: LLD
**/
    int eh_bus_reset_handler(struct scsi_cmnd * scp)


/**
*      eh_device_reset_handler - issue SCSI device reset
*      @scp: identifies SCSI device to be reset
*
*      Returns SUCCESS if command aborted else FAILED
*
*      Locks: None held
*
*      Calling context: kernel thread
*
*      Notes: Invoked from scsi_eh thread. No other commands will be
*      queued on current host during eh.
*
*      Optionally defined in: LLD
**/
    int eh_device_reset_handler(struct scsi_cmnd * scp)


/**
*      eh_host_reset_handler - reset host (host bus adapter)
*      @scp: SCSI host that contains this device should be reset
*
*      Returns SUCCESS if command aborted else FAILED
*
*      Locks: None held
*
*      Calling context: kernel thread
*
*      Notes: Invoked from scsi_eh thread. No other commands will be
```

```
*      queued on current host during eh.
*      With the default eh_strategy in place, if none of the _abort_,
*      _device_reset_, _bus_reset_ or this eh handler function are
*      defined (or they all return FAILED) then the device in question
*      will be set offline whenever eh is invoked.
*
*      Optionally defined in: LLD
**/
    int eh_host_reset_handler(struct scsi_cmnd * scp)


/**
*      info - supply information about given host: driver name plus data
*             to distinguish given host
*      @shp: host to supply information about
*
*      Return ASCII null terminated string. [This driver is assumed to
*      manage the memory pointed to and maintain it, typically for the
*      lifetime of this host.]
*
*      Locks: none
*
*      Calling context: process
*
*      Notes: Often supplies PCI or ISA information such as IO addresses
*      and interrupt numbers. If not supplied struct Scsi_Host::name used
*      instead. It is assumed the returned information fits on one line
*      (i.e. does not included embedded newlines).
*      The SCSI_IOCTL_PROBE_HOST ioctl yields the string returned by this
*      function (or struct Scsi_Host::name if this function is not
*      available).
*      In a similar manner, init_this_scsi_driver() outputs to the console
*      each host's "info" (or name) for the driver it is registering.
*      Also if proc_info() is not supplied, the output of this function
*      is used instead.
*
*      Optionally defined in: LLD
**/
    const char * info(struct Scsi_Host * shp)


/**
*      ioctl - driver can respond to ioctls
*      @sdp: device that ioctl was issued for
*      @cmd: ioctl number
*      @arg: pointer to read or write data from. Since it points to
*            user space, should use appropriate kernel functions
*            (e.g. copy_from_user() ). In the Unix style this argument
*            can also be viewed as an unsigned long.
*
```

```
*      Returns negative "errno" value when there is a problem. 0 or a
*      positive value indicates success and is returned to the user space.
*
*      Locks: none
*
*      Calling context: process
*
*      Notes: The SCSI subsystem uses a "trickle down" ioctl model.
*      The user issues an ioctl() against an upper level driver
*      (e.g. /dev/sdc) and if the upper level driver doesn't recognize
*      the 'cmd' then it is passed to the SCSI mid level. If the SCSI
*      mid level does not recognize it, then the LLD that controls
*      the device receives the ioctl. According to recent Unix standards
*      unsupported ioctl() 'cmd' numbers should return -ENOTTY.
*
*      Optionally defined in: LLD
**/
    int ioctl(struct scsi_device *sdp, int cmd, void *arg)


/**
*      proc_info - supports /proc/scsi/{driver_name}/{host_no}
*      @buffer: anchor point to output to (0==writeto1_read0) or fetch from
*               (1==writeto1_read0).
*      @start: where "interesting" data is written to. Ignored when
*              1==writeto1_read0.
*      @offset: offset within buffer 0==writeto1_read0 is actually
*               interested in. Ignored when 1==writeto1_read0 .
*      @length: maximum (or actual) extent of buffer
*      @host_no: host number of interest (struct Scsi_Host::host_no)
*      @writeto1_read0: 1 -> data coming from user space towards driver
*                           (e.g. "echo some_string > /proc/scsi/xyz/2")
*                       0 -> user what data from this driver
*                           (e.g. "cat /proc/scsi/xyz/2")
*
*      Returns length when 1==writeto1_read0. Otherwise number of chars
*      output to buffer past offset.
*
*      Locks: none held
*
*      Calling context: process
*
*      Notes: Driven from scsi_proc.c which interfaces to proc_fs. proc_fs
*      support can now be configured out of the scsi subsystem.
*
*      Optionally defined in: LLD
**/
    int proc_info(char * buffer, char ** start, off_t offset,
                  int length, int host_no, int writeto1_read0)
```

```
/**
*      queuecommand - queue scsi command, invoke scp->scsi_done on completion
*      @shost: pointer to the scsi host object
*      @scp: pointer to scsi command object
*
*      Returns 0 on success.
*
*      If there's a failure, return either:
*
*      SCSI_MLQUEUE_DEVICE_BUSY if the device queue is full, or
*      SCSI_MLQUEUE_HOST_BUSY if the entire host queue is full
*
*      On both of these returns, the mid-layer will requeue the I/O
*
*      - if the return is SCSI_MLQUEUE_DEVICE_BUSY, only that particular
*      device will be paused, and it will be unpaused when a command to
*      the device returns (or after a brief delay if there are no more
*      outstanding commands to it).  Commands to other devices continue
*      to be processed normally.
*
*      - if the return is SCSI_MLQUEUE_HOST_BUSY, all I/O to the host
*      is paused and will be unpaused when any command returns from
*      the host (or after a brief delay if there are no outstanding
*      commands to the host).
*
*      For compatibility with earlier versions of queuecommand, any
*      other return value is treated the same as
*      SCSI_MLQUEUE_HOST_BUSY.
*
*      Other types of errors that are detected immediately may be
*      flagged by setting scp->result to an appropriate value,
*      invoking the scp->scsi_done callback, and then returning 0
*      from this function. If the command is not performed
*      immediately (and the LLD is starting (or will start) the given
*      command) then this function should place 0 in scp->result and
*      return 0.
*
*      Command ownership.  If the driver returns zero, it owns the
*      command and must take responsibility for ensuring the
*      scp->scsi_done callback is executed.  Note: the driver may
*      call scp->scsi_done before returning zero, but after it has
*      called scp->scsi_done, it may not return any value other than
*      zero.  If the driver makes a non-zero return, it must not
*      execute the command's scsi_done callback at any time.
*
*      Locks: up to and including 2.6.36, struct Scsi_Host::host_lock
*             held on entry (with "irqsave") and is expected to be
*             held on return. From 2.6.37 onwards, queuecommand is
*             called without any locks held.
```

```
*
*      Calling context: in interrupt (soft irq) or process context
*
*      Notes: This function should be relatively fast. Normally it
*      will not wait for IO to complete. Hence the scp->scsi_done
*      callback is invoked (often directly from an interrupt service
*      routine) some time after this function has returned. In some
*      cases (e.g. pseudo adapter drivers that manufacture the
*      response to a SCSI INQUIRY) the scp->scsi_done callback may be
*      invoked before this function returns.  If the scp->scsi_done
*      callback is not invoked within a certain period the SCSI mid
*      level will commence error processing.  If a status of CHECK
*      CONDITION is placed in "result" when the scp->scsi_done
*      callback is invoked, then the LLD driver should perform
*      autosense and fill in the struct scsi_cmnd::sense_buffer
*      array. The scsi_cmnd::sense_buffer array is zeroed prior to
*      the mid level queuing a command to an LLD.
*
*      Defined in: LLD
**/
    int queuecommand(struct Scsi_Host *shost, struct scsi_cmnd * scp)


/**
*      slave_alloc -   prior to any commands being sent to a new device
*                      (i.e. just prior to scan) this call is made
*      @sdp: pointer to new device (about to be scanned)
*
*      Returns 0 if ok. Any other return is assumed to be an error and
*      the device is ignored.
*
*      Locks: none
*
*      Calling context: process
*
*      Notes: Allows the driver to allocate any resources for a device
*      prior to its initial scan. The corresponding scsi device may not
*      exist but the mid level is just about to scan for it (i.e. send
*      and INQUIRY command plus ...). If a device is found then
*      slave_configure() will be called while if a device is not found
*      slave_destroy() is called.
*      For more details see the include/scsi/scsi_host.h file.
*
*      Optionally defined in: LLD
**/
    int slave_alloc(struct scsi_device *sdp)


/**
*      slave_configure - driver fine tuning for given device just after it
```

```
*                    has been first scanned (i.e. it responded to an
*                    INQUIRY)
*      @sdp: device that has just been attached
*
*      Returns 0 if ok. Any other return is assumed to be an error and
*      the device is taken offline. [offline devices will _not_ have
*      slave_destroy() called on them so clean up resources.]
*
*      Locks: none
*
*      Calling context: process
*
*      Notes: Allows the driver to inspect the response to the initial
*      INQUIRY done by the scanning code and take appropriate action.
*      For more details see the include/scsi/scsi_host.h file.
*
*      Optionally defined in: LLD
**/
    int slave_configure(struct scsi_device *sdp)


/**
*      slave_destroy - given device is about to be shut down. All
*                      activity has ceased on this device.
*      @sdp: device that is about to be shut down
*
*      Returns nothing
*
*      Locks: none
*
*      Calling context: process
*
*      Notes: Mid level structures for given device are still in place
*      but are about to be torn down. Any per device resources allocated
*      by this driver for given device should be freed now. No further
*      commands will be sent for this sdp instance. [However the device
*      could be re-attached in the future in which case a new instance
*      of struct scsi_device would be supplied by future slave_alloc()
*      and slave_configure() calls.]
*
*      Optionally defined in: LLD
**/
    void slave_destroy(struct scsi_device *sdp)
```

2.1.9 Data Structures

struct scsi_host_template

There is one "struct scsi_host_template" instance per LLD[2]. It is typically initialized as a file scope static in a driver's header file. That way members that are not explicitly initialized will be set to 0 or NULL. Member of interest:

name

- name of driver (may contain spaces, please limit to less than 80 characters)

proc_name

- name used in "/proc/scsi/<proc_name>/<host_no>" and by sysfs in one of its "drivers" directories. Hence "proc_name" should only contain characters acceptable to a Unix file name.

(*queuecommand)()

- primary callback that the mid level uses to inject SCSI commands into an LLD.

The structure is defined and commented in include/scsi/scsi_host.h

struct Scsi_Host

There is one struct Scsi_Host instance per host (HBA) that an LLD controls. The struct Scsi_Host structure has many members in common with "struct scsi_host_template". When a new struct Scsi_Host instance is created (in scsi_host_alloc() in hosts.c) those common members are initialized from the driver's struct scsi_host_template instance. Members of interest:

host_no

- system wide unique number that is used for identifying this host. Issued in ascending order from 0.

can_queue

- must be greater than 0; do not send more than can_queue commands to the adapter.

this_id

- scsi id of host (scsi initiator) or -1 if not known

sg_tablesize

- maximum scatter gather elements allowed by host. Set this to SG_ALL or less to avoid chained SG lists. Must be at least 1.

max_sectors

- maximum number of sectors (usually 512 bytes) allowed in a single SCSI command. The default value of 0 leads to a setting of SCSI_DEFAULT_MAX_SECTORS (defined in scsi_host.h) which is currently

[2] In extreme situations a single driver may have several instances if it controls several different classes of hardware (e.g. an LLD that handles both ISA and PCI cards and has a separate instance of struct scsi_host_template for each class).

set to 1024. So for a disk the maximum transfer size is 512 KB when max_sectors is not defined. Note that this size may not be sufficient for disk firmware uploads.

cmd_per_lun

- maximum number of commands that can be queued on devices controlled by the host. Overridden by LLD calls to scsi_change_queue_depth().

no_async_abort

- 1=>Asynchronous aborts are not supported
- 0=>Timed-out commands will be aborted asynchronously

hostt

- pointer to driver's struct scsi_host_template from which this struct Scsi_Host instance was spawned

hostt->proc_name

- name of LLD. This is the driver name that sysfs uses

transportt

- pointer to driver's struct scsi_transport_template instance (if any). FC and SPI transports currently supported.

sh_list

- a double linked list of pointers to all struct Scsi_Host instances (currently ordered by ascending host_no)

my_devices

- a double linked list of pointers to struct scsi_device instances that belong to this host.

hostdata[0]

- area reserved for LLD at end of struct Scsi_Host. Size is set by the second argument (named 'xtr_bytes') to scsi_host_alloc() or scsi_register().

vendor_id

- a unique value that identifies the vendor supplying the LLD for the Scsi_Host. Used most often in validating vendor-specific message requests. Value consists of an identifier type and a vendor-specific value. See scsi_netlink.h for a description of valid formats.

The scsi_host structure is defined in include/scsi/scsi_host.h

struct scsi_device

Generally, there is one instance of this structure for each SCSI logical unit on a host. Scsi devices connected to a host are uniquely identified by a channel number, target id and logical unit number (lun). The structure is defined in include/scsi/scsi_device.h

struct scsi_cmnd

Instances of this structure convey SCSI commands to the LLD and responses back to the mid level. The SCSI mid level will ensure that no more SCSI commands become queued against the LLD than are indicated by scsi_change_queue_depth() (or struct Scsi_Host::cmd_per_lun). There will be at least one instance of struct scsi_cmnd available for each SCSI device. Members of interest:

cmnd

- array containing SCSI command

cmnd_len

- length (in bytes) of SCSI command

sc_data_direction

- direction of data transfer in data phase. See ”enum dma_data_direction” in include/linux/dma-mapping.h

request_bufflen

- number of data bytes to transfer (0 if no data phase)

use_sg

- **==0 -> no scatter gather list, hence transfer data**
  to/from request_buffer
- **>0 -> scatter gather list (actually an array) in**
  request_buffer with use_sg elements

request_buffer

- either contains data buffer or scatter gather list depending on the setting of use_sg. Scatter gather elements are defined by 'struct scatterlist' found in include/linux/scatterlist.h .

done

- function pointer that should be invoked by LLD when the SCSI command is completed (successfully or otherwise). Should only be called by an LLD if the LLD has accepted the command (i.e. queuecommand() returned or will return 0). The LLD may invoke 'done' prior to queuecommand() finishing.

result

- should be set by LLD prior to calling 'done'. A value of 0 implies a successfully completed command (and all data (if any) has been transferred to or from the SCSI target device). 'result' is a 32 bit unsigned integer that can be viewed as 2 related bytes. The SCSI status value is in the LSB. See include/scsi/scsi.h status_byte() and host_byte() macros and related constants.

sense_buffer

- an array (maximum size: SCSI_SENSE_BUFFERSIZE bytes) that should be written when the SCSI status (LSB of 'result') is set to CHECK_CONDITION (2). When CHECK_CONDITION is set, if the top nibble of sense_buffer[0] has the value 7 then the mid level will assume the sense_buffer array contains a valid SCSI sense buffer; otherwise the mid level will issue a REQUEST_SENSE SCSI command to retrieve the sense buffer. The latter strategy is error prone in the presence of command queuing so the LLD should always ”auto-sense”.

device

- pointer to scsi_device object that this command is associated with.

resid

- an LLD should set this unsigned integer to the requested transfer length (i.e. 'request_bufflen') less the number of bytes that are actually transferred. 'resid' is preset to 0 so an LLD can ignore it if it cannot detect underruns (overruns should not be reported). An LLD should set 'resid' prior to invoking 'done'. The most interesting case is data transfers from a SCSI target device (e.g. READs) that underrun.

underflow

- LLD should place (DID_ERROR << 16) in 'result' if actual number of bytes transferred is less than this figure. Not many LLDs implement this check and some that do just output an error message to the log rather than report a DID_ERROR. Better for an LLD to implement 'resid'.

It is recommended that a LLD set 'resid' on data transfers from a SCSI target device (e.g. READs). It is especially important that 'resid' is set when such data transfers have sense keys of MEDIUM ERROR and HARDWARE ERROR (and possibly RECOVERED ERROR). In these cases if a LLD is in doubt how much data has been received then the safest approach is to indicate no bytes have been received. For example: to indicate that no valid data has been received a LLD might use these helpers:

```
scsi_set_resid(SCpnt, scsi_bufflen(SCpnt));
```

where 'SCpnt' is a pointer to a scsi_cmnd object. To indicate only three 512 bytes blocks has been received 'resid' could be set like this:

```
scsi_set_resid(SCpnt, scsi_bufflen(SCpnt) - (3 * 512));
```

The scsi_cmnd structure is defined in include/scsi/scsi_cmnd.h

2.1.10 Locks

Each struct Scsi_Host instance has a spin_lock called struct Scsi_Host::default_lock which is initialized in scsi_host_alloc() [found in hosts.c]. Within the same function the struct Scsi_Host::host_lock pointer is initialized to point at default_lock. Thereafter lock and unlock operations performed by the mid level use the struct Scsi_Host::host_lock pointer. Previously drivers could override the host_lock pointer but this is not allowed anymore.

2.1.11 Autosense

Autosense (or auto-sense) is defined in the SAM-2 document as ”the automatic return of sense data to the application client coincident with the completion of a SCSI command” when a status of CHECK CONDITION occurs. LLDs should perform autosense. This should be done when the LLD detects a CHECK CONDITION status by either:

a) instructing the SCSI protocol (e.g. SCSI Parallel Interface (SPI)) to perform an extra data in phase on such responses

b) or, the LLD issuing a REQUEST SENSE command itself

Either way, when a status of CHECK CONDITION is detected, the mid level decides whether the LLD has performed autosense by checking struct scsi_cmnd::sense_buffer[0] . If this byte has an upper nibble of 7 (or 0xf) then autosense is assumed to have taken place. If it has another value (and this byte is initialized to 0 before each command) then the mid level will issue a REQUEST SENSE command.

In the presence of queued commands the ”nexus” that maintains sense buffer data from the command that failed until a following REQUEST SENSE may get out of synchronization. This is why it is best for the LLD to perform autosense.

2.1.12 Changes since lk 2.4 series

io_request_lock has been replaced by several finer grained locks. The lock relevant to LLDs is struct Scsi_Host::host_lock and there is one per SCSI host.

The older error handling mechanism has been removed. This means the LLD interface functions abort() and reset() have been removed. The struct scsi_host_template::use_new_eh_code flag has been removed.

In the 2.4 series the SCSI subsystem configuration descriptions were aggregated with the configuration descriptions from all other Linux subsystems in the Documentation/Configure.help file. In the 2.6 series, the SCSI subsystem now has its own (much smaller) drivers/scsi/Kconfig file that contains both configuration and help information.

struct SHT has been renamed to struct scsi_host_template.

Addition of the ”hotplug initialization model” and many extra functions to support it.

2.1.13 Credits

The following people have contributed to this document:

- Mike Anderson <andmike at us dot ibm dot com>
- James Bottomley <James dot Bottomley at hansenpartnership dot com>
- Patrick Mansfield <patmans at us dot ibm dot com>
- Christoph Hellwig <hch at infradead dot org>
- Doug Ledford <dledford at redhat dot com>
- Andries Brouwer <Andries dot Brouwer at cwi dot nl>
- Randy Dunlap <rdunlap at xenotime dot net>
- Alan Stern <stern at rowland dot harvard dot edu>

Douglas Gilbert dgilbert at interlog dot com

21st September 2004

2.2 SCSI EH

This document describes SCSI midlayer error handling infrastructure. Please refer to *SCSI mid_level - lower_level driver interface* for more information regarding SCSI midlayer.

2.2.1 1. How SCSI commands travel through the midlayer and to EH

1.1 struct scsi_cmnd

Each SCSI command is represented with struct scsi_cmnd (== scmd). A scmd has two list_head's to link itself into lists. The two are scmd->list and scmd->eh_entry. The former is used for free list or per-device allocated scmd list and not of much interest to this EH discussion. The latter is used for completion and EH lists and unless otherwise stated scmds are always linked using scmd->eh_entry in this discussion.

1.2 How do scmd's get completed?

Once LLDD gets hold of a scmd, either the LLDD will complete the command by calling scsi_done callback passed from midlayer when invoking hostt->queuecommand() or the block layer will time it out.

1.2.1 Completing a scmd w/ scsi_done

For all non-EH commands, scsi_done() is the completion callback. It just calls blk_complete_request() to delete the block layer timer and raise SCSI_SOFTIRQ

SCSI_SOFTIRQ handler scsi_softirq calls scsi_decide_disposition() to determine what to do with the command. scsi_decide_disposition() looks at the scmd->result value and sense data to determine what to do with the command.

- SUCCESS

  scsi_finish_command() is invoked for the command. The function does some maintenance chores and then calls scsi_io_completion() to finish the I/O. scsi_io_completion() then notifies the block layer on the completed request by calling blk_end_request and friends or figures out what to do with the remainder of the data in case of an error.

- NEEDS_RETRY
- ADD_TO_MLQUEUE

  scmd is requeued to blk queue.

- otherwise

  scsi_eh_scmd_add(scmd) is invoked for the command. See [1-3] for details of this function.

1.2.2 Completing a scmd w/ timeout

The timeout handler is scsi_timeout(). When a timeout occurs, this function

1. invokes optional hostt->eh_timed_out() callback. Return value can be one of

   - **SCSI_EH_RESET_TIMER**
     This indicates that more time is required to finish the command. Timer is restarted.
   - **SCSI_EH_NOT_HANDLED**
     eh_timed_out() callback did not handle the command. Step #2 is taken.
   - **SCSI_EH_DONE**
     eh_timed_out() completed the command.

2. scsi_abort_command() is invoked to schedule an asynchronous abort which may issue a retry scmd->allowed + 1 times. Asynchronous aborts are not invoked for commands for which the SCSI_EH_ABORT_SCHEDULED flag is set (this indicates that the command already had been aborted once, and this is a retry which failed), when retries are exceeded, or when the EH deadline is expired. In these cases Step #3 is taken.
3. scsi_eh_scmd_add(scmd, SCSI_EH_CANCEL_CMD) is invoked for the command. See [1-4] for more information.

1.3 Asynchronous command aborts

After a timeout occurs a command abort is scheduled from scsi_abort_command(). If the abort is successful the command will either be retried (if the number of retries is not exhausted) or terminated with DID_TIME_OUT.

Otherwise scsi_eh_scmd_add() is invoked for the command. See [1-4] for more information.

1.4 How EH takes over

scmds enter EH via scsi_eh_scmd_add(), which does the following.

1. Links scmd->eh_entry to shost->eh_cmd_q
2. Sets SHOST_RECOVERY bit in shost->shost_state
3. Increments shost->host_failed
4. Wakes up SCSI EH thread if shost->host_busy == shost->host_failed

As can be seen above, once any scmd is added to shost->eh_cmd_q, SHOST_RECOVERY shost_state bit is turned on. This prevents any new scmd to be issued from blk queue to the host; eventually, all scmds on the host either complete normally, fail and get added to eh_cmd_q, or time out and get added to shost->eh_cmd_q.

If all scmds either complete or fail, the number of in-flight scmds becomes equal to the number of failed scmds - i.e. shost->host_busy == shost->host_failed. This wakes up SCSI EH thread. So, once woken up, SCSI EH thread can expect that all in-flight commands have failed and are linked on shost->eh_cmd_q.

Note that this does not mean lower layers are quiescent. If a LLDD completed a scmd with error status, the LLDD and lower layers are assumed to forget about the scmd at that point. However, if a scmd has timed out, unless hostt->eh_timed_out() made lower layers forget about the scmd, which currently no LLDD does, the command is still active as long as lower layers are concerned and completion could occur at any time. Of course, all such completions are ignored as the timer has already expired.

We'll talk about how SCSI EH takes actions to abort - make LLDD forget about - timed out scmds later.

2.2.2 2. How SCSI EH works

LLDD's can implement SCSI EH actions in one of the following two ways.

- **Fine-grained EH callbacks**
  LLDD can implement fine-grained EH callbacks and let SCSI midlayer drive error handling and call appropriate callbacks. This will be discussed further in [2-1].
- **eh_strategy_handler() callback**
  This is one big callback which should perform whole error handling. As such, it should do all chores the SCSI midlayer performs during recovery. This will be discussed in [2-2].

Once recovery is complete, SCSI EH resumes normal operation by calling scsi_restart_operations(), which

1. Checks if door locking is needed and locks door.
2. Clears SHOST_RECOVERY shost_state bit
3. Wakes up waiters on shost->host_wait. This occurs if someone calls scsi_block_when_processing_errors() on the host. (*QUESTION* why is it needed? All operations will be blocked anyway after it reaches blk queue.)
4. Kicks queues in all devices on the host in the asses

2.1 EH through fine-grained callbacks

2.1.1 Overview

If eh_strategy_handler() is not present, SCSI midlayer takes charge of driving error handling. EH's goals are two - make LLDD, host and device forget about timed out scmds and make them ready for new commands. A scmd is said to be recovered if the scmd is forgotten by lower layers and lower layers are ready to process or fail the scmd again.

To achieve these goals, EH performs recovery actions with increasing severity. Some actions are performed by issuing SCSI commands and others are performed by invoking one of the following fine-grained hostt EH callbacks. Callbacks may be omitted and omitted ones are considered to fail always.

```
int (* eh_abort_handler)(struct scsi_cmnd *);
int (* eh_device_reset_handler)(struct scsi_cmnd *);
int (* eh_bus_reset_handler)(struct scsi_cmnd *);
int (* eh_host_reset_handler)(struct scsi_cmnd *);
```

Higher-severity actions are taken only when lower-severity actions cannot recover some of failed scmds. Also, note that failure of the highest-severity action means EH failure and results in offlining of all unrecovered devices.

During recovery, the following rules are followed

- Recovery actions are performed on failed scmds on the to do list, eh_work_q. If a recovery action succeeds for a scmd, recovered scmds are removed from eh_work_q.

  Note that single recovery action on a scmd can recover multiple scmds. e.g. resetting a device recovers all failed scmds on the device.

- Higher severity actions are taken iff eh_work_q is not empty after lower severity actions are complete.
- EH reuses failed scmds to issue commands for recovery. For timed-out scmds, SCSI EH ensures that LLDD forgets about a scmd before reusing it for EH commands.

When a scmd is recovered, the scmd is moved from eh_work_q to EH local eh_done_q using scsi_eh_finish_cmd(). After all scmds are recovered (eh_work_q is empty), scsi_eh_flush_done_q() is invoked to either retry or error-finish (notify upper layer of failure) recovered scmds.

scmds are retried iff its sdev is still online (not offlined during EH), REQ_FAILFAST is not set and ++scmd->retries is less than scmd->allowed.

2.1.2 Flow of scmds through EH

1. Error completion / time out

   ACTION
   scsi_eh_scmd_add() is invoked for scmd

   - add scmd to shost->eh_cmd_q
   - set SHOST_RECOVERY
   - shost->host_failed++

   LOCKING
   shost->host_lock

2. EH starts

   ACTION
   move all scmds to EH's local eh_work_q. shost->eh_cmd_q is cleared.

   LOCKING
   shost->host_lock (not strictly necessary, just for consistency)

3. scmd recovered

   ACTION
   scsi_eh_finish_cmd() is invoked to EH-finish scmd

   - scsi_setup_cmd_retry()
   - move from local eh_work_q to local eh_done_q

   LOCKING
   none

   CONCURRENCY
   at most one thread per separate eh_work_q to keep queue manipulation lockless

4. EH completes

   ACTION
   scsi_eh_flush_done_q() retries scmds or notifies upper layer of failure. May be called concurrently but must have a no more than one thread per separate eh_work_q to manipulate the queue locklessly

   - scmd is removed from eh_done_q and scmd->eh_entry is cleared
   - if retry is necessary, scmd is requeued using scsi_queue_insert()
   - otherwise, scsi_finish_command() is invoked for scmd
   - zero shost->host_failed

   LOCKING
   queue or finish function performs appropriate locking

2.1.3 Flow of control

EH through fine-grained callbacks start from scsi_unjam_host().

`scsi_unjam_host`

1. Lock shost->host_lock, splice_init shost->eh_cmd_q into local eh_work_q and unlock host_lock. Note that shost->eh_cmd_q is cleared by this action.
2. Invoke scsi_eh_get_sense.

`scsi_eh_get_sense`

This action is taken for each error-completed (!SCSI_EH_CANCEL_CMD) commands without valid sense data. Most SCSI transports/LLDDs automatically acquire sense data on command failures (autosense). Autosense is recommended for performance reasons and as sense information could get out of sync between occurrence of CHECK CONDITION and this action.

Note that if autosense is not supported, scmd->sense_buffer contains invalid sense data when error-completing the scmd with scsi_done(). scsi_decide_disposition() always returns FAILED in such cases thus invoking SCSI EH. When the scmd reaches here, sense data is acquired and scsi_decide_disposition() is called again.

1. Invoke scsi_request_sense() which issues REQUEST_SENSE command. If fails, no action. Note that taking no action causes higher-severity recovery to be taken for the scmd.
2. Invoke scsi_decide_disposition() on the scmd
   - **SUCCESS**
     scmd->retries is set to scmd->allowed preventing scsi_eh_flush_done_q() from retrying the scmd and scsi_eh_finish_cmd() is invoked.
   - **NEEDS_RETRY**
     scsi_eh_finish_cmd() invoked
   - **otherwise**
     No action.
3. If !list_empty(&eh_work_q), invoke scsi_eh_abort_cmds().

`scsi_eh_abort_cmds`

This action is taken for each timed out command when no_async_abort is enabled in the host template. hostt->eh_abort_handler() is invoked for each scmd. The handler returns SUCCESS if it has succeeded to make LLDD and all related hardware forget about the scmd.

If a timedout scmd is successfully aborted and the sdev is either offline or ready, scsi_eh_finish_cmd() is invoked for the scmd. Otherwise, the scmd is left in eh_work_q for higher-severity actions.

Note that both offline and ready status mean that the sdev is ready to process new scmds, where processing also implies immediate failing; thus, if a sdev is in one of the two states, no further recovery action is needed.

Device readiness is tested using scsi_eh_tur() which issues TEST_UNIT_READY command. Note that the scmd must have been aborted successfully before reusing it for TEST_UNIT_READY.

4. If !list_empty(&eh_work_q), invoke scsi_eh_ready_devs()

`scsi_eh_ready_devs`

This function takes four increasingly more severe measures to make failed sdevs ready for new commands.

1. Invoke scsi_eh_stu()

`scsi_eh_stu`

For each sdev which has failed scmds with valid sense data of which scsi_check_sense()'s verdict is FAILED, START_STOP_UNIT command is issued w/ start=1. Note that as we explicitly choose error-completed scmds, it is known that lower layers have forgotten about the scmd and we can reuse it for STU.

If STU succeeds and the sdev is either offline or ready, all failed scmds on the sdev are EH-finished with scsi_eh_finish_cmd().

NOTE If hostt->eh_abort_handler() isn't implemented or failed, we may still have timed out scmds at this point and STU doesn't make lower layers forget about those scmds. Yet, this function EH-finish all scmds on the sdev if STU succeeds leaving lower layers in an inconsistent state. It seems that STU action should be taken only when a sdev has no timed out scmd.

2. If !list_empty(&eh_work_q), invoke scsi_eh_bus_device_reset().

`scsi_eh_bus_device_reset`

This action is very similar to scsi_eh_stu() except that, instead of issuing STU, hostt->eh_device_reset_handler() is used. Also, as we're not issuing SCSI commands and resetting clears all scmds on the sdev, there is no need to choose error-completed scmds.

3. If !list_empty(&eh_work_q), invoke scsi_eh_bus_reset()

`scsi_eh_bus_reset`

hostt->eh_bus_reset_handler() is invoked for each channel with failed scmds. If bus reset succeeds, all failed scmds on all ready or offline sdevs on the channel are EH-finished.

4. If !list_empty(&eh_work_q), invoke scsi_eh_host_reset()

`scsi_eh_host_reset`

This is the last resort. hostt->eh_host_reset_handler() is invoked. If host reset succeeds, all failed scmds on all ready or offline sdevs on the host are EH-finished.

5. If !list_empty(&eh_work_q), invoke scsi_eh_offline_sdevs()

`scsi_eh_offline_sdevs`

Take all sdevs which still have unrecovered scmds offline and EH-finish the scmds.

5. Invoke scsi_eh_flush_done_q().

   `scsi_eh_flush_done_q`

   At this point all scmds are recovered (or given up) and put on eh_done_q by scsi_eh_finish_cmd(). This function flushes eh_done_q by either retrying or notifying upper layer of failure of the scmds.

2.2 EH through transportt->eh_strategy_handler()

transportt->eh_strategy_handler() is invoked in the place of scsi_unjam_host() and it is responsible for whole recovery process. On completion, the handler should have made lower layers forget about all failed scmds and either ready for new commands or offline. Also, it should perform SCSI EH maintenance chores to maintain integrity of SCSI midlayer. IOW, of the steps described in [2-1-2], all steps except for #1 must be implemented by eh_strategy_handler().

2.2.1 Pre transportt->eh_strategy_handler() SCSI midlayer conditions

The following conditions are true on entry to the handler.

- Each failed scmd's eh_flags field is set appropriately.
- Each failed scmd is linked on scmd->eh_cmd_q by scmd->eh_entry.
- SHOST_RECOVERY is set.
- shost->host_failed == shost->host_busy

2.2.2 Post transportt->eh_strategy_handler() SCSI midlayer conditions

The following conditions must be true on exit from the handler.

- shost->host_failed is zero.
- Each scmd is in such a state that scsi_setup_cmd_retry() on the scmd doesn't make any difference.
- shost->eh_cmd_q is cleared.
- Each scmd->eh_entry is cleared.
- Either scsi_queue_insert() or scsi_finish_command() is called on each scmd. Note that the handler is free to use scmd->retries and ->allowed to limit the number of retries.

2.2.3 Things to consider

- Know that timed out scmds are still active on lower layers. Make lower layers forget about them before doing anything else with those scmds.
- For consistency, when accessing/modifying shost data structure, grab shost->host_lock.
- On completion, each failed sdev must have forgotten about all active scmds.
- On completion, each failed sdev must be ready for new commands or offline.

Tejun Heo htejun@gmail.com

11th September 2005

CHAPTER

THREE

SCSI DRIVER PARAMETERS

3.1 SCSI Kernel Parameters

See Documentation/admin-guide/kernel-parameters.rst for general information on specifying module parameters.

This document may not be entirely up to date and comprehensive. The command `modinfo -p ${modulename}` shows a current list of all parameters of a loadable module. Loadable modules, after being loaded into the running kernel, also reveal their parameters in /sys/module/${modulename}/parameters/. Some of these parameters may be changed at runtime by the command `echo -n ${value} > /sys/module/${modulename}/parameters/${parm}`.

```
advansys=       [HW,SCSI]
                See header of drivers/scsi/advansys.c.

aha152x=        [HW,SCSI]
                See Documentation/scsi/aha152x.rst.

aha1542=        [HW,SCSI]
                Format: <portbase>[,<buson>,<busoff>[,<dmaspeed>]]

aic7xxx=        [HW,SCSI]
                See Documentation/scsi/aic7xxx.rst.

aic79xx=        [HW,SCSI]
                See Documentation/scsi/aic79xx.rst.

atascsi=        [HW,SCSI]
                See drivers/scsi/atari_scsi.c.

BusLogic=       [HW,SCSI]
                See drivers/scsi/BusLogic.c, comment before function
                BusLogic_ParseDriverOptions().

gvp11=          [HW,SCSI]

ips=            [HW,SCSI] Adaptec / IBM ServeRAID controller
                See header of drivers/scsi/ips.c.
```

```
mac5380=        [HW,SCSI]
                See drivers/scsi/mac_scsi.c.

scsi_mod.max_luns=
                [SCSI] Maximum number of LUNs to probe.
                Should be between 1 and 2^32-1.

scsi_mod.max_report_luns=
                [SCSI] Maximum number of LUNs received.
                Should be between 1 and 16384.

NCR_D700=       [HW,SCSI]
                See header of drivers/scsi/NCR_D700.c.

ncr5380=        [HW,SCSI]
                See Documentation/scsi/g_NCR5380.rst.

ncr53c400=      [HW,SCSI]
                See Documentation/scsi/g_NCR5380.rst.

ncr53c400a=     [HW,SCSI]
                See Documentation/scsi/g_NCR5380.rst.

ncr53c8xx=      [HW,SCSI]

osst=           [HW,SCSI] SCSI Tape Driver
                Format: <buffer_size>,<write_threshold>
                See also Documentation/scsi/st.rst.

scsi_debug_*=   [SCSI]
                See drivers/scsi/scsi_debug.c.

scsi_mod.default_dev_flags=
                [SCSI] SCSI default device flags
                Format: <integer>

scsi_mod.dev_flags=
                [SCSI] Black/white list entry for vendor and model
                Format: <vendor>:<model>:<flags>
                (flags are integer value)

scsi_mod.scsi_logging_level=
                [SCSI] a bit mask of logging levels
                See drivers/scsi/scsi_logging.h for bits.  Also
                settable via sysctl at dev.scsi.logging_level
                (/proc/sys/dev/scsi/logging_level).
                There is also a nice 'scsi_logging_level' script in the
                S390-tools package, available for download at
                https://github.com/ibm-s390-linux/s390-tools/blob/master/
↪scripts/scsi_logging_level
```

```
scsi_mod.scan=  [SCSI] sync (default) scans SCSI busses as they are
                discovered.  async scans them in kernel threads,
                allowing boot to proceed.  none ignores them, expecting
                user space to do the scan.

sim710=         [SCSI,HW]
                See header of drivers/scsi/sim710.c.

st=             [HW,SCSI] SCSI tape parameters (buffers, etc.)
                See Documentation/scsi/st.rst.

wd33c93=        [HW,SCSI]
                See header of drivers/scsi/wd33c93.c.
```

3.2 Link Power Managent Policy

This parameter allows the user to set the link (interface) power management. There are 3 possible options:

Value	Effect
min_power	Tell the controller to try to make the link use the least possible power when possible. This may sacrifice some performance due to increased latency when coming out of lower power states.
max_performance	Generally, this means no power management. Tell the controller to have performance be a priority over power management.
medium_power	Tell the controller to enter a lower power state when possible, but do not enter the lowest power state, thus improving latency over min_power setting.

CHAPTER

FOUR

SCSI HOST ADAPTER DRIVERS

4.1 The 53c700 Driver Notes

4.1.1 General Description

This driver supports the 53c700 and 53c700-66 chips. It also supports the 53c710 but only in 53c700 emulation mode. It is full featured and does sync (-66 and 710 only), disconnects and tag command queueing.

Since the 53c700 must be interfaced to a bus, you need to wrapper the card detector around this driver. For an example, see the NCR_D700.[ch] or lasi700.[ch] files.

The comments in the 53c700.[ch] files tell you which parts you need to fill in to get the driver working.

4.1.2 Compile Time Flags

A compile time flag is:

```
CONFIG_53C700_LE_ON_BE
```

define if the chipset must be supported in little endian mode on a big endian architecture (used for the 700 on parisc).

4.1.3 Using the Chip Core Driver

In order to plumb the 53c700 chip core driver into a working SCSI driver, you need to know three things about the way the chip is wired into your system (or expansion card).

1. The clock speed of the SCSI core
2. The interrupt line used
3. The memory (or io space) location of the 53c700 registers.

Optionally, you may also need to know other things, like how to read the SCSI Id from the card bios or whether the chip is wired for differential operation.

Usually you can find items 2. and 3. from general spec. documents or even by examining the configuration of a working driver under another operating system.

The clock speed is usually buried deep in the technical literature. It is required because it is used to set up both the synchronous and asynchronous dividers for the chip. As a general rule of thumb, manufacturers set the clock speed at the lowest possible setting consistent with the best operation of the chip (although some choose to drive it off the CPU or bus clock rather than going to the expense of an extra clock chip). The best operation clock speeds are:

53c700	25MHz
53c700-66	50MHz
53c710	40Mhz

4.1.4 Writing Your Glue Driver

This will be a standard SCSI driver (I don't know of a good document describing this, just copy from some other driver) with at least a detect and release entry.

In the detect routine, you need to allocate a struct NCR_700_Host_Parameters sized memory area and clear it (so that the default values for everything are 0). Then you must fill in the parameters that matter to you (see below), plumb the NCR_700_intr routine into the interrupt line and call NCR_700_detect with the host template and the new parameters as arguments. You should also call the relevant request_*_region function and place the register base address into the 'base' pointer of the host parameters.

In the release routine, you must free the NCR_700_Host_Parameters that you allocated, call the corresponding release_*_region and free the interrupt.

Handling Interrupts

In general, you should just plumb the card's interrupt line in with

request_irq(irq, NCR_700_intr, <irq flags>, <driver name>, host);

where host is the return from the relevant NCR_700_detect() routine.

You may also write your own interrupt handling routine which calls NCR_700_intr() directly. However, you should only really do this if you have a card with more than one chip on it and you can read a register to tell which set of chips wants the interrupt.

Settable NCR_700_Host_Parameters

The following are a list of the user settable parameters:

clock: (MANDATORY)
Set to the clock speed of the chip in MHz.

base: (MANDATORY)
Set to the base of the io or mem region for the register set. On 64 bit architectures this is only 32 bits wide, so the registers must be mapped into the low 32 bits of memory.

pci_dev: (OPTIONAL)
Set to the PCI board device. Leave NULL for a non-pci board. This is used for the pci_alloc_consistent() and pci_map_*() functions.

dmode_extra: (OPTIONAL, 53c710 only)
Extra flags for the DMODE register. These are used to control bus output pins on the 710. The settings should be a combination of DMODE_FC1 and DMODE_FC2. What these pins actually do is entirely up to the board designer. Usually it is safe to ignore this setting.

differential: (OPTIONAL)
Set to 1 if the chip drives a differential bus.

force_le_on_be: (OPTIONAL, only if CONFIG_53C700_LE_ON_BE is set)
Set to 1 if the chip is operating in little endian mode on a big endian architecture.

chip710: (OPTIONAL)
Set to 1 if the chip is a 53c710.

burst_disable: (OPTIONAL, 53c710 only)
Disable 8 byte bursting for DMA transfers.

4.2 AACRAID Driver for Linux (take two)

4.2.1 Introduction

The aacraid driver adds support for Adaptec (http://www.adaptec.com) RAID controllers. This is a major rewrite from the original Adaptec supplied driver. It has significantly cleaned up both the code and the running binary size (the module is less than half the size of the original).

4.2.2 Supported Cards/Chipsets

PCI ID (pci.ids)	OEM	Product
9005:0285:9005:0285	Adaptec	2200S (Vulcan)
9005:0285:9005:0286	Adaptec	2120S (Crusader)
9005:0285:9005:0287	Adaptec	2200S (Vulcan-2m)
9005:0285:9005:0288	Adaptec	3230S (Harrier)
9005:0285:9005:0289	Adaptec	3240S (Tornado)
9005:0285:9005:028a	Adaptec	2020ZCR (Skyhawk)
9005:0285:9005:028b	Adaptec	2025ZCR (Terminator)
9005:0286:9005:028c	Adaptec	2230S (Lancer)
9005:0286:9005:028c	Adaptec	2230SLP (Lancer)
9005:0286:9005:028d	Adaptec	2130S (Lancer)
9005:0285:9005:028e	Adaptec	2020SA (Skyhawk)
9005:0285:9005:028f	Adaptec	2025SA (Terminator)
9005:0285:9005:0290	Adaptec	2410SA (Jaguar)
9005:0285:103c:3227	Adaptec	2610SA (Bearcat HP release)
9005:0285:9005:0293	Adaptec	21610SA (Corsair-16)
9005:0285:9005:0296	Adaptec	2240S (SabreExpress)
9005:0285:9005:0292	Adaptec	2810SA (Corsair-8)
9005:0285:9005:0297	Adaptec	4005 (AvonPark)
9005:0285:9005:0298	Adaptec	4000 (BlackBird)
9005:0285:9005:0299	Adaptec	4800SAS (Marauder-X)
9005:0285:9005:029a	Adaptec	4805SAS (Marauder-E)

continues on next page

Table 1 – continued from previous page

PCI ID (pci.ids)	OEM	Product
9005:0286:9005:029b	Adaptec	2820SA (Intruder)
9005:0286:9005:029c	Adaptec	2620SA (Intruder)
9005:0286:9005:029d	Adaptec	2420SA (Intruder HP release)
9005:0286:9005:02ac	Adaptec	1800 (Typhoon44)
9005:0285:9005:02b5	Adaptec	5445 (Voodoo44)
9005:0285:15d9:02b5	SMC	AOC-USAS-S4i
9005:0285:9005:02b6	Adaptec	5805 (Voodoo80)
9005:0285:15d9:02b6	SMC	AOC-USAS-S8i
9005:0285:9005:02b7	Adaptec	5085 (Voodoo08)
9005:0285:9005:02bb	Adaptec	3405 (Marauder40LP)
9005:0285:9005:02bc	Adaptec	3805 (Marauder80LP)
9005:0285:9005:02c7	Adaptec	3085 (Marauder08ELP)
9005:0285:9005:02bd	Adaptec	31205 (Marauder120)
9005:0285:9005:02be	Adaptec	31605 (Marauder160)
9005:0285:9005:02c3	Adaptec	51205 (Voodoo120)
9005:0285:9005:02c4	Adaptec	51605 (Voodoo160)
9005:0285:15d9:02c9	SMC	AOC-USAS-S4iR
9005:0285:15d9:02ca	SMC	AOC-USAS-S8iR
9005:0285:9005:02ce	Adaptec	51245 (Voodoo124)
9005:0285:9005:02cf	Adaptec	51645 (Voodoo164)
9005:0285:9005:02d0	Adaptec	52445 (Voodoo244)
9005:0285:9005:02d1	Adaptec	5405 (Voodoo40)
9005:0285:15d9:02d2	SMC	AOC-USAS-S8i-LP
9005:0285:15d9:02d3	SMC	AOC-USAS-S8iR-LP
9005:0285:9005:02d4	Adaptec	ASR-2045 (Voodoo04 Lite)
9005:0285:9005:02d5	Adaptec	ASR-2405 (Voodoo40 Lite)
9005:0285:9005:02d6	Adaptec	ASR-2445 (Voodoo44 Lite)
9005:0285:9005:02d7	Adaptec	ASR-2805 (Voodoo80 Lite)
9005:0285:9005:02d8	Adaptec	5405Z (Voodoo40 BLBU)
9005:0285:9005:02d9	Adaptec	5445Z (Voodoo44 BLBU)
9005:0285:9005:02da	Adaptec	5805Z (Voodoo80 BLBU)
1011:0046:9005:0364	Adaptec	5400S (Mustang)
1011:0046:9005:0365	Adaptec	5400S (Mustang)
9005:0287:9005:0800	Adaptec	Themisto (Jupiter)
9005:0200:9005:0200	Adaptec	Themisto (Jupiter)
9005:0286:9005:0800	Adaptec	Callisto (Jupiter)
1011:0046:9005:1364	Dell	PERC 2/QC (Quad Channel, Mustang)
1011:0046:9005:1365	Dell	PERC 2/QC (Quad Channel, Mustang)
1028:0001:1028:0001	Dell	PERC 2/Si (Iguana)
1028:0003:1028:0003	Dell	PERC 3/Si (SlimFast)
1028:0002:1028:0002	Dell	PERC 3/Di (Opal)
1028:0004:1028:0004	Dell	PERC 3/SiF (Iguana)
1028:0004:1028:00d0	Dell	PERC 3/DiF (Iguana)
1028:0002:1028:00d1	Dell	PERC 3/DiV (Viper)
1028:0002:1028:00d9	Dell	PERC 3/DiL (Lexus)
1028:000a:1028:0106	Dell	PERC 3/DiJ (Jaguar)
1028:000a:1028:011b	Dell	PERC 3/DiD (Dagger)

continues on next page

Table 1 – continued from previous page

PCI ID (pci.ids)	OEM	Product
1028:000a:1028:0121	Dell	PERC 3/DiB (Boxster)
9005:0285:1028:0287	Dell	PERC 320/DC (Vulcan)
9005:0285:1028:0291	Dell	CERC 2 (DellCorsair)
1011:0046:103c:10c2	HP	NetRAID-4M (Mustang)
9005:0285:17aa:0286	Legend	S220 (Crusader)
9005:0285:17aa:0287	Legend	S230 (Vulcan)
9005:0285:9005:0290	IBM	ServeRAID 7t (Jaguar)
9005:0285:1014:02F2	IBM	ServeRAID 8i (AvonPark)
9005:0286:1014:9540	IBM	ServeRAID 8k/8k-l4 (AuroraLite)
9005:0286:1014:9580	IBM	ServeRAID 8k/8k-l8 (Aurora)
9005:0285:1014:034d	IBM	ServeRAID 8s (Marauder-E)
9005:0286:9005:029e	ICP	ICP9024RO (Lancer)
9005:0286:9005:029f	ICP	ICP9014RO (Lancer)
9005:0286:9005:02a0	ICP	ICP9047MA (Lancer)
9005:0286:9005:02a1	ICP	ICP9087MA (Lancer)
9005:0285:9005:02a4	ICP	ICP9085LI (Marauder-X)
9005:0285:9005:02a5	ICP	ICP5085BR (Marauder-E)
9005:0286:9005:02a6	ICP	ICP9067MA (Intruder-6)
9005:0285:9005:02b2	ICP	(Voodoo 8 internal 8 external)
9005:0285:9005:02b8	ICP	ICP5445SL (Voodoo44)
9005:0285:9005:02b9	ICP	ICP5085SL (Voodoo80)
9005:0285:9005:02ba	ICP	ICP5805SL (Voodoo08)
9005:0285:9005:02bf	ICP	ICP5045BL (Marauder40LP)
9005:0285:9005:02c0	ICP	ICP5085BL (Marauder80LP)
9005:0285:9005:02c8	ICP	ICP5805BL (Marauder08ELP)
9005:0285:9005:02c1	ICP	ICP5125BR (Marauder120)
9005:0285:9005:02c2	ICP	ICP5165BR (Marauder160)
9005:0285:9005:02c5	ICP	ICP5125SL (Voodoo120)
9005:0285:9005:02c6	ICP	ICP5165SL (Voodoo160)
9005:0286:9005:02ab		(Typhoon40)
9005:0286:9005:02ad		(Aurora ARK)
9005:0286:9005:02ae		(Aurora Lite ARK)
9005:0285:9005:02b0		(Sunrise Lake ARK)
9005:0285:9005:02b1	Adaptec	(Voodoo 8 internal 8 external)
9005:0285:108e:7aac	SUN	STK RAID REM (Voodoo44 Coyote)
9005:0285:108e:0286	SUN	STK RAID INT (Cougar)
9005:0285:108e:0287	SUN	STK RAID EXT (Prometheus)
9005:0285:108e:7aae	SUN	STK RAID EM (Narvi)

4.2.3 People

Alan Cox <alan@lxorguk.ukuu.org.uk>

Christoph Hellwig <hch@infradead.org>

- updates for new-style PCI probing and SCSI host registration, small cleanups/fixes

Matt Domsch <matt_domsch@dell.com>

- revision ioctl, adapter messages

Deanna Bonds

- non-DASD support, PAE fibs and 64 bit, added new adaptec controllers added new ioctls, changed scsi interface to use new error handler, increased the number of fibs and outstanding commands to a container
- fixed 64bit and 64G memory model, changed confusing naming convention where fibs that go to the hardware are consistently called hw_fibs and not just fibs like the name of the driver tracking structure

Mark Salyzyn <Mark_Salyzyn@adaptec.com>

- Fixed panic issues and added some new product ids for upcoming hbas.
- Performance tuning, card failover and bug mitigations.

Achim Leubner <Achim_Leubner@adaptec.com>

- Original Driver

Adaptec Unix OEM Product Group

4.2.4 Mailing List

linux-scsi@vger.kernel.org (Interested parties troll here) Also note this is very different to Brian's original driver so don't expect him to support it.

Adaptec does support this driver. Contact Adaptec tech support or aacraid@adaptec.com

Original by Brian Boerner February 2001

Rewritten by Alan Cox, November 2001

4.3 AdvanSys Driver Notes

AdvanSys (Advanced System Products, Inc.) manufactures the following RISC-based, Bus-Mastering, Fast (10 Mhz) and Ultra (20 Mhz) Narrow (8-bit transfer) SCSI Host Adapters for the ISA, EISA, VL, and PCI buses and RISC-based, Bus-Mastering, Ultra (20 Mhz) Wide (16-bit transfer) SCSI Host Adapters for the PCI bus.

The CDB counts below indicate the number of SCSI CDB (Command Descriptor Block) requests that can be stored in the RISC chip cache and board LRAM. A CDB is a single SCSI command. The driver detect routine will display the number of CDBs available for each adapter detected. The number of CDBs used by the driver can be lowered in the BIOS by changing the 'Host Queue Size' adapter setting.

Laptop Products:

- ABP-480 - Bus-Master CardBus (16 CDB)

Connectivity Products:

- ABP510/5150 - Bus-Master ISA (240 CDB)
- ABP5140 - Bus-Master ISA PnP (16 CDB)
- ABP5142 - Bus-Master ISA PnP with floppy (16 CDB)
- ABP902/3902 - Bus-Master PCI (16 CDB)
- ABP3905 - Bus-Master PCI (16 CDB)
- ABP915 - Bus-Master PCI (16 CDB)
- ABP920 - Bus-Master PCI (16 CDB)
- ABP3922 - Bus-Master PCI (16 CDB)
- ABP3925 - Bus-Master PCI (16 CDB)
- ABP930 - Bus-Master PCI (16 CDB)
- ABP930U - Bus-Master PCI Ultra (16 CDB)
- ABP930UA - Bus-Master PCI Ultra (16 CDB)
- ABP960 - Bus-Master PCI MAC/PC (16 CDB)
- ABP960U - Bus-Master PCI MAC/PC Ultra (16 CDB)

Single Channel Products:

- ABP542 - Bus-Master ISA with floppy (240 CDB)
- ABP742 - Bus-Master EISA (240 CDB)
- ABP842 - Bus-Master VL (240 CDB)
- ABP940 - Bus-Master PCI (240 CDB)
- ABP940U - Bus-Master PCI Ultra (240 CDB)
- ABP940UA/3940UA - Bus-Master PCI Ultra (240 CDB)
- ABP970 - Bus-Master PCI MAC/PC (240 CDB)
- ABP970U - Bus-Master PCI MAC/PC Ultra (240 CDB)
- ABP3960UA - Bus-Master PCI MAC/PC Ultra (240 CDB)
- ABP940UW/3940UW - Bus-Master PCI Ultra-Wide (253 CDB)
- ABP970UW - Bus-Master PCI MAC/PC Ultra-Wide (253 CDB)
- ABP3940U2W - Bus-Master PCI LVD/Ultra2-Wide (253 CDB)

Multi-Channel Products:

- ABP752 - Dual Channel Bus-Master EISA (240 CDB Per Channel)
- ABP852 - Dual Channel Bus-Master VL (240 CDB Per Channel)
- ABP950 - Dual Channel Bus-Master PCI (240 CDB Per Channel)

- ABP950UW - Dual Channel Bus-Master PCI Ultra-Wide (253 CDB Per Channel)
- ABP980 - Four Channel Bus-Master PCI (240 CDB Per Channel)
- ABP980U - Four Channel Bus-Master PCI Ultra (240 CDB Per Channel)
- ABP980UA/3980UA - Four Channel Bus-Master PCI Ultra (16 CDB Per Chan.)
- ABP3950U2W - Bus-Master PCI LVD/Ultra2-Wide and Ultra-Wide (253 CDB)
- ABP3950U3W - Bus-Master PCI Dual LVD2/Ultra3-Wide (253 CDB)

4.3.1 Driver Compile Time Options and Debugging

The following constants can be defined in the source file.

1. ADVANSYS_ASSERT - Enable driver assertions (Def: Enabled)

   Enabling this option adds assertion logic statements to the driver. If an assertion fails a message will be displayed to the console, but the system will continue to operate. Any assertions encountered should be reported to the person responsible for the driver. Assertion statements may proactively detect problems with the driver and facilitate fixing these problems. Enabling assertions will add a small overhead to the execution of the driver.

2. ADVANSYS_DEBUG - Enable driver debugging (Def: Disabled)

   Enabling this option adds tracing functions to the driver and the ability to set a driver tracing level at boot time. This option is very useful for debugging the driver, but it will add to the size of the driver execution image and add overhead to the execution of the driver.

   The amount of debugging output can be controlled with the global variable 'asc_dbglvl'. The higher the number the more output. By default the debug level is 0.

   If the driver is loaded at boot time and the LILO Driver Option is included in the system, the debug level can be changed by specifying a 5th (ASC_NUM_IOPORT_PROBE + 1) I/O Port. The first three hex digits of the pseudo I/O Port must be set to 'deb' and the fourth hex digit specifies the debug level: 0 - F. The following command line will look for an adapter at 0x330 and set the debug level to 2:

   ```
   linux advansys=0x330,0,0,0,0xdeb2
   ```

   If the driver is built as a loadable module this variable can be defined when the driver is loaded. The following insmod command will set the debug level to one:

   ```
   insmod advansys.o asc_dbglvl=1
   ```

   Debugging Message Levels:

0	Errors Only
1	High-Level Tracing
2-N	Verbose Tracing

   To enable debug output to console, please make sure that:

   a. System and kernel logging is enabled (syslogd, klogd running).

b. Kernel messages are routed to console output. Check /etc/syslog.conf for an entry similar to this:

```
kern.*                  /dev/console
```

c. klogd is started with the appropriate -c parameter (e.g. klogd -c 8)

This will cause printk() messages to be displayed on the current console. Refer to the klogd(8) and syslogd(8) man pages for details.

Alternatively you can enable printk() to console with this program. However, this is not the 'official' way to do this.

Debug output is logged in /var/log/messages.

```
main()
{
        syscall(103, 7, 0, 0);
}
```

Increasing LOG_BUF_LEN in kernel/printk.c to something like 40960 allows more debug messages to be buffered in the kernel and written to the console or log file.

3. ADVANSYS_STATS - Enable statistics (Def: Enabled)

Enabling this option adds statistics collection and display through /proc to the driver. The information is useful for monitoring driver and device performance. It will add to the size of the driver execution image and add minor overhead to the execution of the driver.

Statistics are maintained on a per adapter basis. Driver entry point call counts and transfer size counts are maintained. Statistics are only available for kernels greater than or equal to v1.3.0 with the CONFIG_PROC_FS (/proc) file system configured.

AdvanSys SCSI adapter files have the following path name format:

```
/proc/scsi/advansys/{0,1,2,3,...}
```

This information can be displayed with cat. For example:

```
cat /proc/scsi/advansys/0
```

When ADVANSYS_STATS is not defined the AdvanSys /proc files only contain adapter and device configuration information.

4.3.2 Driver LILO Option

If init/main.c is modified as described in the 'Directions for Adding the AdvanSys Driver to Linux' section (B.4.) above, the driver will recognize the 'advansys' LILO command line and /etc/lilo.conf option. This option can be used to either disable I/O port scanning or to limit scanning to 1 - 4 I/O ports. Regardless of the option setting EISA and PCI boards will still be searched for and detected. This option only affects searching for ISA and VL boards.

Examples:

1. Eliminate I/O port scanning:

   boot:

```
linux advansys=
```

or:

```
boot: linux advansys=0x0
```

2. Limit I/O port scanning to one I/O port:

   boot:

   ```
   linux advansys=0x110
   ```

3. Limit I/O port scanning to four I/O ports:

   boot:

   ```
   linux advansys=0x110,0x210,0x230,0x330
   ```

For a loadable module the same effect can be achieved by setting the 'asc_iopflag' variable and 'asc_ioport' array when loading the driver, e.g.:

```
insmod advansys.o asc_iopflag=1 asc_ioport=0x110,0x330
```

If ADVANSYS_DEBUG is defined a 5th (ASC_NUM_IOPORT_PROBE + 1) I/O Port may be added to specify the driver debug level. Refer to the 'Driver Compile Time Options and Debugging' section above for more information.

4.3.3 Credits (Chronological Order)

Bob Frey <bfrey@turbolinux.com.cn> wrote the AdvanSys SCSI driver and maintained it up to 3.3F. He continues to answer questions and help maintain the driver.

Nathan Hartwell <mage@cdc3.cdc.net> provided the directions and basis for the Linux v1.3.X changes which were included in the 1.2 release.

Thomas E Zerucha <zerucha@shell.portal.com> pointed out a bug in advansys_biosparam() which was fixed in the 1.3 release.

Erik Ratcliffe <erik@caldera.com> has done testing of the AdvanSys driver in the Caldera releases.

Rik van Riel <H.H.vanRiel@fys.ruu.nl> provided a patch to AscWaitTixISRDone() which he found necessary to make the driver work with a SCSI-1 disk.

Mark Moran <mmoran@mmoran.com> has helped test Ultra-Wide support in the 3.1A driver.

Doug Gilbert <dgilbert@interlog.com> has made changes and suggestions to improve the driver and done a lot of testing.

Ken Mort <ken@mort.net> reported a DEBUG compile bug fixed in 3.2K.

Tom Rini <trini@kernel.crashing.org> provided the CONFIG_ISA patch and helped with PowerPC wide and narrow board support.

Philip Blundell <philb@gnu.org> provided an advansys_interrupts_enabled patch.

Dave Jones <dave@denial.force9.co.uk> reported the compiler warnings generated when CONFIG_PROC_FS was not defined in the 3.2M driver.

Jerry Quinn <jlquinn@us.ibm.com> fixed PowerPC support (endian problems) for wide cards.

Bryan Henderson <bryanh@giraffe-data.com> helped debug narrow card error handling.

Manuel Veloso <veloso@pobox.com> worked hard on PowerPC narrow board support and fixed a bug in AscGetEEPConfig().

Arnaldo Carvalho de Melo <acme@conectiva.com.br> made save_flags/restore_flags changes.

Andy Kellner <AKellner@connectcom.net> continued the Advansys SCSI driver development for ConnectCom (Version > 3.3F).

Ken Witherow for extensive testing during the development of version 3.4.

4.4 Adaptec AHA-1520/1522 SCSI driver for Linux (aha152x)

Copyright © 1993-1999 Jürgen Fischer <fischer@norbit.de>

TC1550 patches by Luuk van Dijk (ldz@xs4all.nl)

In Revision 2 the driver was modified a lot (especially the bottom-half handler complete()).

The driver is much cleaner now, has support for the new error handling code in 2.3, produced less cpu load (much less polling loops), has slightly higher throughput (at least on my ancient test box; a i486/33Mhz/20MB).

4.4.1 Configuration Arguments

IOPORT	base io address	(0x340/0x140)
IRQ	interrupt level	(9-12; default 11)
SCSI_ID	scsi id of controller	(0-7; default 7)
RECONNECT	allow targets to disconnect from the bus	(0/1; default 1 [on])
PARITY	enable parity checking	(0/1; default 1 [on])
SYNCHRONOUS	enable synchronous transfers	(0/1; default 1 [on])
DELAY:	bus reset delay	(default 100)
EXT_TRANS:	enable extended translation (see NOTES)	(0/1: default 0 [off])

4.4.2 Compile Time Configuration

(go into AHA152X in drivers/scsi/Makefile):

- **DAUTOCONF**
  use configuration the controller reports (AHA-152x only)
- **DSKIP_BIOSTEST**
  Don't test for BIOS signature (AHA-1510 or disabled BIOS)
- **DSETUP0="{ IOPORT, IRQ, SCSI_ID, RECONNECT, PARITY, SYNCHRONOUS, DELAY, EXT_TRANS }"**
  override for the first controller

- **DSETUP1="{ IOPORT, IRQ, SCSI_ID, RECONNECT, PARITY, SYNCHRONOUS, DELAY, EXT_TRANS }"**
  override for the second controller
- **DAHA152X_DEBUG**
  enable debugging output
- **DAHA152X_STAT**
  enable some statistics

4.4.3 LILO Command Line Options

```
aha152x=<IOPORT>[,<IRQ>[,<SCSI-ID>[,<RECONNECT>[,<PARITY>[,
↪<SYNCHRONOUS>[,<DELAY> [,<EXT_TRANS]]]]]]
```

The normal configuration can be overridden by specifying a command line. When you do this, the BIOS test is skipped. Entered values have to be valid (known). Don't use values that aren't supported under normal operation. If you think that you need other values: contact me. For two controllers use the aha152x statement twice.

4.4.4 Symbols for Module Configuration

Choose from 2 alternatives:

1. specify everything (old):

```
aha152x=IOPORT,IRQ,SCSI_ID,RECONNECT,PARITY,SYNCHRONOUS,DELAY,EXT_TRANS
```

configuration override for first controller

```
aha152x1=IOPORT,IRQ,SCSI_ID,RECONNECT,PARITY,SYNCHRONOUS,DELAY,EXT_
↪TRANS
```

configuration override for second controller

2. specify only what you need to (irq or io is required; new)

io=IOPORT0[,IOPORT1]
IOPORT for first and second controller

irq=IRQ0[,IRQ1]
IRQ for first and second controller

scsiid=SCSIID0[,SCSIID1]
SCSIID for first and second controller

reconnect=RECONNECT0[,RECONNECT1]
allow targets to disconnect for first and second controller

parity=PAR0[PAR1]
use parity for first and second controller

sync=SYNCHRONOUS0[,SYNCHRONOUS1]
enable synchronous transfers for first and second controller

delay=DELAY0[,DELAY1]
reset DELAY for first and second controller

exttrans=EXTTRANS0[,EXTTRANS1]
enable extended translation for first and second controller

If you use both alternatives the first will be taken.

4.4.5 Notes on EXT_TRANS

SCSI uses block numbers to address blocks/sectors on a device. The BIOS uses a cylinder/head/sector addressing scheme (C/H/S) scheme instead. DOS expects a BIOS or driver that understands this C/H/S addressing.

The number of cylinders/heads/sectors is called geometry and is required as base for requests in C/H/S addressing. SCSI only knows about the total capacity of disks in blocks (sectors).

Therefore the SCSI BIOS/DOS driver has to calculate a logical/virtual geometry just to be able to support that addressing scheme. The geometry returned by the SCSI BIOS is a pure calculation and has nothing to do with the real/physical geometry of the disk (which is usually irrelevant anyway).

Basically this has no impact at all on Linux, because it also uses block instead of C/H/S addressing. Unfortunately C/H/S addressing is also used in the partition table and therefore every operating system has to know the right geometry to be able to interpret it.

Moreover there are certain limitations to the C/H/S addressing scheme, namely the address space is limited to up to 255 heads, up to 63 sectors and a maximum of 1023 cylinders.

The AHA-1522 BIOS calculates the geometry by fixing the number of heads to 64, the number of sectors to 32 and by calculating the number of cylinders by dividing the capacity reported by the disk by 64*32 (1 MB). This is considered to be the default translation.

With respect to the limit of 1023 cylinders using C/H/S you can only address the first GB of your disk in the partition table. Therefore BIOSes of some newer controllers based on the AIC-6260/6360 support extended translation. This means that the BIOS uses 255 for heads, 63 for sectors and then divides the capacity of the disk by 255*63 (about 8 MB), as soon it sees a disk greater than 1 GB. That results in a maximum of about 8 GB addressable diskspace in the partition table (but there are already bigger disks out there today).

To make it even more complicated the translation mode might/might not be configurable in certain BIOS setups.

This driver does some more or less failsafe guessing to get the geometry right in most cases:

- for disks<1GB: use default translation (C/32/64)
- for disks>1GB:
  - take current geometry from the partition table (using scsicam_bios_param and accept only 'valid' geometries, ie. either (C/32/64) or (C/63/255)). This can be extended translation even if it's not enabled in the driver.
  - if that fails, take extended translation if enabled by override, kernel or module parameter, otherwise take default translation and ask the user for verification. This might on not yet partitioned disks.

4.4.6 References Used

”AIC-6260 SCSI Chip Specification”, Adaptec Corporation.

”SCSI COMPUTER SYSTEM INTERFACE - 2 (SCSI-2)”, X3T9.2/86-109 rev. 10h

”Writing a SCSI device driver for Linux”, Rik Faith (faith@cs.unc.edu)

”Kernel Hacker's Guide”, Michael K. Johnson (johnsonm@sunsite.unc.edu)

”Adaptec 1520/1522 User's Guide”, Adaptec Corporation.

Michael K. Johnson (johnsonm@sunsite.unc.edu)

Drew Eckhardt (drew@cs.colorado.edu)

Eric Youngdale (eric@andante.org)

special thanks to Eric Youngdale for the free(!) supplying the documentation on the chip.

4.5 Adaptec Ultra320 Family Manager Set

README for The Linux Operating System

4.5.1 1. Supported Hardware

The following Adaptec SCSI Host Adapters are supported by this driver set.

Ultra320 ASIC	Description
AIC-7901A	Single Channel 64-bit PCI-X 133MHz to Ultra320 SCSI ASIC
AIC-7901B	Single Channel 64-bit PCI-X 133MHz to Ultra320 SCSI ASIC with Retained Training
AIC-7902A4	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI ASIC
AIC-7902B	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI ASIC with Retained Training

Ultra320 Adapters	Description	ASIC
Adaptec SCSI Card 39320	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (one external 68-pin, two internal 68-pin)	7902A4/7902B
Adaptec SCSI Card 39320A	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (one external 68-pin, two internal 68-pin)	7902B
Adaptec SCSI Card 39320D	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (two external VHDC and one internal 68-pin)	7902A4
Adaptec SCSI Card 39320D	Dual Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (two external VHDC and one internal 68-pin) based on the AIC-7902B ASIC	7902A4
Adaptec SCSI Card 29320	Single Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (one external 68-pin, two internal 68-pin, one internal 50-pin)	7901A
Adaptec SCSI Card 29320A	Single Channel 64-bit PCI-X 133MHz to Ultra320 SCSI Card (one external 68-pin, two internal 68-pin, one internal 50-pin)	7901B
Adaptec SCSI Card 29320LP	Single Channel 64-bit Low Profile PCI-X 133MHz to Ultra320 SCSI Card (One external VHDC, one internal 68-pin)	7901A
Adaptec SCSI Card 29320ALP	Single Channel 64-bit Low Profile PCI-X 133MHz to Ultra320 SCSI Card (One external VHDC, one internal 68-pin)	7901B

4.5.2 2. Version History

- **3.0 (December 1st, 2005)**
  - Updated driver to use SCSI transport class infrastructure
  - Upported sequencer and core fixes from adaptec released version 2.0.15 of the driver.
- **1.3.11 (July 11, 2003)**
  - Fix several deadlock issues.
  - Add 29320ALP and 39320B Id's.
- **1.3.10 (June 3rd, 2003)**
  - Align the SCB_TAG field on a 16byte boundary. This avoids SCB corruption on some PCI-33 busses.
  - Correct non-zero luns on Rev B. hardware.
  - Update for change in 2.5.X SCSI proc FS interface.

  - When negotiation async via an 8bit WDTR message, send an SDTR with an offset of 0 to be sure the target knows we are async. This works around a firmware defect in the Quantum Atlas 10K.
  - Implement controller suspend and resume.
  - Clear PCI error state during driver attach so that we don't disable memory mapped I/O due to a stray write by some other driver probe that occurred before we claimed the controller.

- **1.3.9 (May 22nd, 2003)**
  - Fix compiler errors.
  - Remove S/G splitting for segments that cross a 4GB boundary. This is guaranteed not to happen in Linux.
  - Add support for scsi_report_device_reset() found in 2.5.X kernels.
  - Add 7901B support.
  - Simplify handling of the packetized lun Rev A workaround.
  - Correct and simplify handling of the ignore wide residue message. The previous code would fail to report a residual if the transaction data length was even and we received an IWR message.
- **1.3.8 (April 29th, 2003)**
  - Fix types accessed via the command line interface code.
  - Perform a few firmware optimizations.
  - Fix ”Unexpected PKT busfree” errors.
  - Use a sequencer interrupt to notify the host of commands with bad status. We defer the notification until there are no outstanding selections to ensure that the host is interrupted for as short a time as possible.
  - Remove pre-2.2.X support.
  - Add support for new 2.5.X interrupt API.
  - Correct big-endian architecture support.
- **1.3.7 (April 16th, 2003)**
  - Use del_timer_sync() to ensure that no timeouts are pending during controller shutdown.
  - For pre-2.5.X kernels, carefully adjust our segment list size to avoid SCSI malloc pool fragmentation.
  - Cleanup channel display in our /proc output.
  - Workaround duplicate device entries in the mid-layer device list during add-single-device.
- **1.3.6 (March 28th, 2003)**
  - Correct a double free in the Domain Validation code.
  - Correct a reference to free'ed memory during controller shutdown.

  - Reset the bus on an SE->LVD change. This is required to reset our transceivers.

- **1.3.5 (March 24th, 2003)**
  - Fix a few register window mode bugs.
  - Include read streaming in the PPR flags we display in diagnostics as well as /proc.
  - Add PCI hot plug support for 2.5.X kernels.
  - Correct default precompensation value for RevA hardware.
  - Fix Domain Validation thread shutdown.
  - Add a firmware workaround to make the LED blink brighter during packetized operations on the H2A4.
  - Correct /proc display of user read streaming settings.
  - Simplify driver locking by releasing the io_request_lock upon driver entry from the mid-layer.
  - Cleanup command line parsing and move much of this code to aiclib.
- **1.3.4 (February 28th, 2003)**
  - Correct a race condition in our error recovery handler.
  - Allow Test Unit Ready commands to take a full 5 seconds during Domain Validation.
- **1.3.2 (February 19th, 2003)**
  - Correct a Rev B. regression due to the GEM318 compatibility fix included in 1.3.1.
- **1.3.1 (February 11th, 2003)**
  - Add support for the 39320A.
  - Improve recovery for certain PCI-X errors.
  - Fix handling of LQ/DATA/LQ/DATA for the same write transaction that can occur without interveining training.
  - Correct compatibility issues with the GEM318 enclosure services device.
  - Correct data corruption issue that occurred under high tag depth write loads.
  - Adapt to a change in the 2.5.X daemonize() API.
  - Correct a ”Missing case in ahd_handle_scsiint” panic.
- **1.3.0 (January 21st, 2003)**
  - Full regression testing for all U320 products completed.
  - Added abort and target/lun reset error recovery handler and interrupt coalescing.
- **1.2.0 (November 14th, 2002)**
  - Added support for Domain Validation
  - Add support for the Hewlett-Packard version of the 39320D and AIC-7902 adapters.

  Support for previous adapters has not been fully tested and should only be used at the customer's own risk.

- **1.1.1 (September 24th, 2002)**
  - Added support for the Linux 2.5.X kernel series
- **1.1.0 (September 17th, 2002)**
  - Added support for four additional SCSI products: ASC-39320, ASC-29320, ASC-29320LP, AIC-7901.
- **1.0.0 (May 30th, 2002)**
  - Initial driver release.
- **2.1. Software/Hardware Features**
  - Support for the SPI-4 ”Ultra320” standard: - 320MB/s transfer rates - Packetized SCSI Protocol at 160MB/s and 320MB/s - Quick Arbitration Selection (QAS) - Retained Training Information (Rev B. ASIC only)
  - Interrupt Coalescing
  - Initiator Mode (target mode not currently supported)
  - Support for the PCI-X standard up to 133MHz
  - Support for the PCI v2.2 standard
  - Domain Validation
- **2.2. Operating System Support:**
  - Redhat Linux 7.2, 7.3, 8.0, Advanced Server 2.1
  - SuSE Linux 7.3, 8.0, 8.1, Enterprise Server 7
  - only Intel and AMD x86 supported at this time
  - >4GB memory configurations supported.

  Refer to the User's Guide for more details on this.

4.5.3 3. Command Line Options

> **Warning:** ALTERING OR ADDING THESE DRIVER PARAMETERS INCORRECTLY CAN RENDER YOUR SYSTEM INOPERABLE. USE THEM WITH CAUTION.

Put a .conf file in the /etc/modprobe.d/ directory and add/edit a line containing `options aic79xx aic79xx=[command[,command...]]` where `command` is one or more of the following:

verbose

Definition
enable additional informative messages during driver operation.

Possible Values
This option is a flag

Default Value
disabled

debug:[value]

Definition
Enables various levels of debugging information The bit definitions for the debugging mask can be found in drivers/scsi/aic7xxx/aic79xx.h under the "Debug" heading.

Possible Values
0x0000 = no debugging, 0xffff = full debugging

Default Value
0x0000

no_reset

Definition
Do not reset the bus during the initial probe phase

Possible Values
This option is a flag

Default Value
disabled

extended

Definition
Force extended translation on the controller

Possible Values
This option is a flag

Default Value
disabled

periodic_otag

Definition
Send an ordered tag periodically to prevent tag starvation. Needed for some older devices

Possible Values
This option is a flag

Default Value
disabled

reverse_scan

Definition
Probe the scsi bus in reverse order, starting with target 15

Possible Values
This option is a flag

Default Value
disabled

global_tag_depth

Definition
Global tag depth for all targets on all busses. This option sets the default tag depth which may be selectively overridden vi the tag_info option.

Possible Values
1 - 253

Default Value
32

tag_info:{{value[,value...]}[,{value[,value...]}...]}

Definition
Set the per-target tagged queue depth on a per controller basis. Both controllers and targets may be omitted indicating that they should retain the default tag depth.

Possible Values
1 - 253

Default Value
32

Examples:

```
tag_info:{{16,32,32,64,8,8,,32,32,32,32,32,32,32,32,32}
```

On Controller 0

- specifies a tag depth of 16 for target 0
- specifies a tag depth of 64 for target 3
- specifies a tag depth of 8 for targets 4 and 5
- leaves target 6 at the default
- specifies a tag depth of 32 for targets 1,2,7-15

All other targets retain the default depth.

```
tag_info:{{},{32,,32}}
```

On Controller 1

- specifies a tag depth of 32 for targets 0 and 2

All other targets retain the default depth.

rd_strm: {rd_strm_bitmask[,rd_strm_bitmask...]}

Definition
Enable read streaming on a per target basis. The rd_strm_bitmask is a 16 bit hex value in which each bit represents a target. Setting the target's bit to '1' enables read streaming for that target. Controllers may be omitted indicating that they should retain the default read streaming setting.

Examples:

```
rd_strm:{0x0041}
```

On Controller 0

- enables read streaming for targets 0 and 6.
- disables read streaming for targets 1-5,7-15.

All other targets retain the default read streaming setting.

```
rd_strm:{0x0023,,0xFFFF}
```

On Controller 0

- enables read streaming for targets 1,2, and 5.
- disables read streaming for targets 3,4,6-15.

On Controller 2

- enables read streaming for all targets.

All other targets retain the default read streaming setting.

Possible Values
0x0000 - 0xffff

Default Value
0x0000

dv: {value[,value...]}

Definition

Set Domain Validation Policy on a per-controller basis.
Controllers may be omitted indicating that they should retain the default read streaming setting.

Possible Values

< 0	Use setting from serial EEPROM.
0	Disable DV
> 0	Enable DV

Default Value
DV Serial EEPROM configuration setting.

Example:

```
dv:{-1,0,,1,1,0}
```

- On Controller 0 leave DV at its default setting.
- On Controller 1 disable DV.
- Skip configuration on Controller 2.
- On Controllers 3 and 4 enable DV.
- On Controller 5 disable DV.

seltime:[value]

Definition
Specifies the selection timeout value

Possible Values
0 = 256ms, 1 = 128ms, 2 = 64ms, 3 = 32ms

Default Value
0

precomp: {value[,value...]}

Definition
Set IO Cell precompensation value on a per-controller basis. Controllers may be omitted indicating that they should retain the default precompensation setting.

Possible Values
0 - 7

Default Value
Varies based on chip revision

Examples:

```
precomp:{0x1}
```

On Controller 0 set precompensation to 1.

```
precomp:{1,,7}
```

- On Controller 0 set precompensation to 1.
- On Controller 2 set precompensation to 8.

slewrate: {value[,value...]}

Definition
Set IO Cell slew rate on a per-controller basis. Controllers may be omitted indicating that they should retain the default slew rate setting.

Possible Values
0 - 15

Default Value
Varies based on chip revision

Examples:

```
slewrate:{0x1}
```

- On Controller 0 set slew rate to 1.

```
slewrate :{1,,8}
```

- On Controller 0 set slew rate to 1.
- On Controller 2 set slew rate to 8.

amplitude: {value[,value...]}

Definition
Set IO Cell signal amplitude on a per-controller basis. Controllers may be omitted indicating that they should retain the default read streaming setting.

Possible Values
1 - 7

Default Value
Varies based on chip revision

Examples:

```
amplitude:{0x1}
```

On Controller 0 set amplitude to 1.

```
amplitude :{1,,7}
```

- On Controller 0 set amplitude to 1.
- On Controller 2 set amplitude to 7.

Example:

```
options aic79xx aic79xx=verbose,rd_strm:{{0x0041}}
```

enables verbose output in the driver and turns read streaming on for targets 0 and 6 of Controller 0.

4.5.4 4. Additional Notes

4.1. Known/Unresolved or FYI Issues

- Under SuSE Linux Enterprise 7, the driver may fail to operate correctly due to a problem with PCI interrupt routing in the Linux kernel. Please contact SuSE for an updated Linux kernel.

4.2. Third-Party Compatibility Issues

- Adaptec only supports Ultra320 hard drives running the latest firmware available. Please check with your hard drive manufacturer to ensure you have the latest version.

4.3. Operating System or Technology Limitations

- PCI Hot Plug is untested and may cause the operating system to stop responding.
- Luns that are not numbered contiguously starting with 0 might not be automatically probed during system startup. This is a limitation of the OS. Please contact your Linux vendor for instructions on manually probing non-contiguous luns.
- Using the Driver Update Disk version of this package during OS installation under RedHat might result in two versions of this driver being installed into the system module directory. This might cause problems with the /sbin/mkinitrd program and/or other RPM packages

that try to install system modules. The best way to correct this once the system is running is to install the latest RPM package version of this driver, available from http://www.adaptec.com.

4.5.5 5. Adaptec Customer Support

A Technical Support Identification (TSID) Number is required for Adaptec technical support.

- The 12-digit TSID can be found on the white barcode-type label included inside the box with your product. The TSID helps us provide more efficient service by accurately identifying your product and support status.

Support Options

- Search the Adaptec Support Knowledgebase (ASK) at http://ask.adaptec.com for articles, troubleshooting tips, and frequently asked questions about your product.
- For support via Email, submit your question to Adaptec's Technical Support Specialists at http://ask.adaptec.com/.

North America

- Visit our Web site at http://www.adaptec.com/.
- For information about Adaptec's support options, call 408-957-2550, 24 hours a day, 7 days a week.
- To speak with a Technical Support Specialist,
  - For hardware products, call 408-934-7274, Monday to Friday, 3:00 am to 5:00 pm, PDT.
  - For RAID and Fibre Channel products, call 321-207-2000, Monday to Friday, 3:00 am to 5:00 pm, PDT.

  To expedite your service, have your computer with you.
- To order Adaptec products, including accessories and cables, call 408-957-7274. To order cables online go to http://www.adaptec.com/buy-cables/.

Europe

- Visit our Web site at http://www.adaptec.com/en-US/_common/world_index.
- To speak with a Technical Support Specialist, call, or email,
  - German: +49 89 4366 5522, Monday-Friday, 9:00-17:00 CET, http://ask-de.adaptec.com/.
  - French: +49 89 4366 5533, Monday-Friday, 9:00-17:00 CET, http://ask-fr.adaptec.com/.
  - English: +49 89 4366 5544, Monday-Friday, 9:00-17:00 GMT, http://ask.adaptec.com/.
- You can order Adaptec cables online at http://www.adaptec.com/buy-cables/.

Japan

- Visit our web site at http://www.adaptec.co.jp/.
- To speak with a Technical Support Specialist, call +81 3 5308 6120, Monday-Friday, 9:00 a.m. to 12:00 p.m., 1:00 p.m. to 6:00 p.m.

Copyright © 2003 Adaptec Inc. 691 S. Milpitas Blvd., Milpitas CA 95035 USA. All rights reserved.

You are permitted to redistribute, use and modify this README file in whole or in part in conjunction with redistribution of software governed by the General Public License, provided that the following conditions are met:

1. Redistributions of README file must retain the above copyright notice, this list of conditions, and the following disclaimer, without modification.
2. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.
3. Modifications or new contributions must be attributed in a copyright notice identifying the author (”Contributor”) and added below the original copyright notice. The copyright notice is for purposes of identifying contributors and should not be deemed as permission to alter the permissions given by Adaptec.

THIS README FILE IS PROVIDED BY ADAPTEC AND CONTRIBUTORS `AS IS` AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTIES OF NON-INFRINGEMENT OR THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ADAPTEC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS README FILE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.6 Adaptec Aic7xxx Fast -> Ultra160 Family Manager Set v7.0

README for The Linux Operating System

The following information is available in this file:

1. Supported Hardware
2. Version History
3. Command Line Options
4. Contacting Adaptec

4.6.1 1. Supported Hardware

The following Adaptec SCSI Chips and Host Adapters are supported by the aic7xxx driver.

Chip	MIPS	Host Bus	MaxSync	MaxWidth	SCBs	Notes
aic7770	10	EISA/VL	10MHz	16Bit	4	1
aic7850	10	PCI/32	10MHz	8Bit	3	
aic7855	10	PCI/32	10MHz	8Bit	3	
aic7856	10	PCI/32	10MHz	8Bit	3	
aic7859	10	PCI/32	20MHz	8Bit	3	
aic7860	10	PCI/32	20MHz	8Bit	3	
aic7870	10	PCI/32	10MHz	16Bit	16	
aic7880	10	PCI/32	20MHz	16Bit	16	
aic7890	20	PCI/32	40MHz	16Bit	16	3 4 5 6 7 8
aic7891	20	PCI/64	40MHz	16Bit	16	3 4 5 6 7 8
aic7892	20	PCI/64-66	80MHz	16Bit	16	3 4 5 6 7 8
aic7895	15	PCI/32	20MHz	16Bit	16	2 3 4 5
aic7895C	15	PCI/32	20MHz	16Bit	16	2 3 4 5 8
aic7896	20	PCI/32	40MHz	16Bit	16	2 3 4 5 6 7 8
aic7897	20	PCI/64	40MHz	16Bit	16	2 3 4 5 6 7 8
aic7899	20	PCI/64-66	80MHz	16Bit	16	2 3 4 5 6 7 8

1. Multiplexed Twin Channel Device - One controller servicing two busses.
2. Multi-function Twin Channel Device - Two controllers on one chip.
3. Command Channel Secondary DMA Engine - Allows scatter gather list and SCB prefetch.
4. 64 Byte SCB Support - Allows disconnected, untagged request table for all possible target/lun combinations.
5. Block Move Instruction Support - Doubles the speed of certain sequencer operations.
6. 'Bayonet' style Scatter Gather Engine - Improves S/G prefetch performance.
7. Queuing Registers - Allows queuing of new transactions without pausing the sequencer.
8. Multiple Target IDs - Allows the controller to respond to selection as a target on multiple SCSI IDs.

Controller	Chip	Host-Bus	Int-Connectors	Ext-Connectors	Nc
AHA-274X[A]	aic7770	EISA	SE-50M	SE-HD50F	
AHA-274X[A]W	aic7770	EISA	SE-HD68F SE-50M	SE-HD68F	
AHA-274X[A]T	aic7770	EISA	2 X SE-50M	SE-HD50F	
AHA-2842	aic7770	VL	SE-50M	SE-HD50F	
AHA-2940AU	aic7860	PCI/32	SE-50M	SE-HD50F	
AVA-2902I	aic7860	PCI/32	SE-50M		
AVA-2902E	aic7860	PCI/32	SE-50M		
AVA-2906	aic7856	PCI/32	SE-50M	SE-DB25F	

continues on next p

Table 2 – continued from previous page

Controller	Chip	Host-Bus	Int-Connectors	Ext-Connectors	No
APC-7850	aic7850	PCI/32	SE-50M		1
AVA-2940	aic7860	PCI/32	SE-50M		
AHA-2920B	aic7860	PCI/32	SE-50M		
AHA-2930B	aic7860	PCI/32	SE-50M		
AHA-2920C	aic7856	PCI/32	SE-50M	SE-HD50F	
AHA-2930C	aic7860	PCI/32	SE-50M		
AHA-2930C	aic7860	PCI/32	SE-50M		
AHA-2910C	aic7860	PCI/32	SE-50M		
AHA-2915C	aic7860	PCI/32	SE-50M		
AHA-2940AU/CN	aic7860	PCI/32	SE-50M	SE-HD50F	
AHA-2944W	aic7870	PCI/32	HVD-HD68F HVD-50M	HVD-HD68F	
AHA-3940W	aic7870	PCI/32	2 X SE-HD68F	SE-HD68F	2
AHA-2940UW	aic7880	PCI/32	SE-HD68F SE-50M	SE-HD68F	
AHA-2940U	aic7880	PCI/32	SE-50M	SE-HD50F	
AHA-2940D	aic7880	PCI/32			
aHA-2940 A/T	aic7880	PCI/32			
AHA-2940D A/T	aic7880	PCI/32			
AHA-3940UW	aic7880	PCI/32	2 X SE-HD68F	SE-HD68F	3
AHA-3940UWD	aic7880	PCI/32	2 X SE-HD68F	2 X SE-VHD68F	3
AHA-3940U	aic7880	PCI/32	2 X SE-50M	SE-HD50F	3
AHA-2944UW	aic7880	PCI/32	HVD-HD68F HVD-50M	HVD-HD68F	
AHA-3944UWD	aic7880	PCI/32	2 X HVD-HD68F	2 X HVD-VHD68F	3
AHA-4944UW	aic7880	PCI/32			
AHA-2930UW	aic7880	PCI/32			
AHA-2940UW Pro	aic7880	PCI/32	SE-HD68F SE-50M	SE-HD68F	4
AHA-2940UW/CN	aic7880	PCI/32			
AHA-2940UDual	aic7895	PCI/32			
AHA-2940UWDual	aic7895	PCI/32			
AHA-3940UWD	aic7895	PCI/32			
AHA-3940AUW	aic7895	PCI/32			
AHA-3940AUWD	aic7895	PCI/32			
AHA-3940AU	aic7895	PCI/32			
AHA-3944AUWD	aic7895	PCI/32	2 X HVD-HD68F	2 X HVD-VHD68F	
AHA-2940U2B	aic7890	PCI/32	LVD-HD68F	LVD-HD68F	
AHA-2940U2 OEM	aic7891	PCI/64			
AHA-2940U2W	aic7890	PCI/32	LVD-HD68F SE-HD68F SE-50M	LVD-HD68F	
AHA-2950U2B	aic7891	PCI/64	LVD-HD68F	LVD-HD68F	
AHA-2930U2	aic7890	PCI/32	LVD-HD68F SE-50M	SE-HD50F	
AHA-3950U2B	aic7897	PCI/64			
AHA-3950U2D	aic7897	PCI/64			
AHA-29160	aic7892	PCI/64-66			
AHA-29160 CPQ	aic7892	PCI/64-66			
AHA-29160N	aic7892	PCI/32	LVD-HD68F SE-50M	SE-HD50F	
AHA-29160LP	aic7892	PCI/64-66			
AHA-19160	aic7892	PCI/64-66			
AHA-29150LP	aic7892	PCI/64-66			
AHA-29130LP	aic7892	PCI/64-66			

continues on next p

Table 2 – continued from previous page

Controller	Chip	Host-Bus	Int-Connectors	Ext-Connectors	No
AHA-3960D	aic7899	PCI/64-66	2 X LVD-HD68F LVD-50M	2 X LVD-VHD68F	
AHA-3960D CPQ	aic7899	PCI/64-66	2 X LVD-HD68F LVD-50M	2 X LVD-VHD68F	
AHA-39160	aic7899	PCI/64-66	2 X LVD-HD68F LVD-50M	2 X LVD-VHD68F	

1. No BIOS support
2. DEC21050 PCI-PCI bridge with multiple controller chips on secondary bus
3. DEC2115X PCI-PCI bridge with multiple controller chips on secondary bus
4. All three SCSI connectors may be used simultaneously without SCSI ”stub” effects.

4.6.2 2. Version History

- **7.0 (4th August, 2005)**
  - Updated driver to use SCSI transport class infrastructure
  - Upported sequencer and core fixes from last adaptec released version of the driver.
- **6.2.36 (June 3rd, 2003)**
  - Correct code that disables PCI parity error checking.
  - Correct and simplify handling of the ignore wide residue message. The previous code would fail to report a residual if the transaction data length was even and we received an IWR message.
  - Add support for the 2.5.X EISA framework.
  - Update for change in 2.5.X SCSI proc FS interface.
  - Correct Domain Validation command-line option parsing.
  - When negotiation async via an 8bit WDTR message, send an SDTR with an offset of 0 to be sure the target knows we are async. This works around a firmware defect in the Quantum Atlas 10K.
  - Clear PCI error state during driver attach so that we don't disable memory mapped I/O due to a stray write by some other driver probe that occurred before we claimed the controller.
- **6.2.35 (May 14th, 2003)**
  - Fix a few GCC 3.3 compiler warnings.
  - Correct operation on EISA Twin Channel controller.
  - Add support for 2.5.X's scsi_report_device_reset().
- **6.2.34 (May 5th, 2003)**
  - Fix locking regression introduced in 6.2.29 that could cause a lock order reversal between the io_request_lock and our per-softc lock. This was only possible on RH9, SuSE, and kernel.org 2.4.X kernels.
- **6.2.33 (April 30th, 2003)**

  - Dynamically disable PCI parity error reporting after 10 errors are reported to the user. These errors are the result of some other device issuing PCI transactions with bad parity. Once the user has been informed of the problem, continuing to report the errors just degrades our performance.

- **6.2.32 (March 28th, 2003)**
  - Dynamically sized S/G lists to avoid SCSI malloc pool fragmentation and SCSI mid-layer deadlock.
- **6.2.28 (January 20th, 2003)**
  - Domain Validation Fixes
  - Add ability to disable PCI parity error checking.
  - Enhanced Memory Mapped I/O probe
- **6.2.20 (November 7th, 2002)**
  - Added Domain Validation.

4.6.3 3. Command Line Options

> **Warning:** ALTERING OR ADDING THESE DRIVER PARAMETERS INCORRECTLY CAN RENDER YOUR SYSTEM INOPERABLE. USE THEM WITH CAUTION.

Put a .conf file in the /etc/modprobe.d directory and add/edit a line containing `options aic7xxx aic7xxx=[command[,command...]]` where `command` is one or more of the following:

verbose

Definition
enable additional informative messages during driver operation.

Possible Values
This option is a flag

Default Value
disabled

debug:[value]

Definition
Enables various levels of debugging information

Possible Values
0x0000 = no debugging, 0xffff = full debugging

Default Value
0x0000

no_probe

probe_eisa_vl

Definition
Do not probe for EISA/VLB controllers. This is a toggle. If the driver is compiled to not probe EISA/VLB controllers by default, specifying ”no_probe” will enable this probing. If the driver is compiled to probe EISA/VLB controllers by default, specifying ”no_probe” will disable this probing.

Possible Values
This option is a toggle

Default Value
EISA/VLB probing is disabled by default.

pci_parity

Definition
Toggles the detection of PCI parity errors. On many motherboards with VIA chipsets, PCI parity is not generated correctly on the PCI bus. It is impossible for the hardware to differentiate between these ”spurious” parity errors and real parity errors. The symptom of this problem is a stream of the message:

```
"scsi0:     Data Parity Error Detected during address or␣
↪write data phase"
```

output by the driver.

Possible Values
This option is a toggle

Default Value
PCI Parity Error reporting is disabled

no_reset

Definition
Do not reset the bus during the initial probe phase

Possible Values
This option is a flag

Default Value
disabled

extended

Definition
Force extended translation on the controller

Possible Values
This option is a flag

Default Value
disabled

periodic_otag

Definition
Send an ordered tag periodically to prevent tag starvation. Needed for

some older devices

Possible Values
This option is a flag

Default Value
disabled

reverse_scan

Definition
Probe the scsi bus in reverse order, starting with target 15

Possible Values
This option is a flag

Default Value
disabled

global_tag_depth:[value]

Definition
Global tag depth for all targets on all busses. This option sets the default tag depth which may be selectively overridden vi the tag_info option.

Possible Values
1 - 253

Default Value
32

tag_info:{{value[,value...]}[,{value[,value...]}...]}

Definition
Set the per-target tagged queue depth on a per controller basis. Both controllers and targets may be omitted indicating that they should retain the default tag depth.

Possible Values
1 - 253

Default Value
32

Examples:

```
tag_info:{{16,32,32,64,8,8,,32,32,32,32,32,32,32,32,32}
```

On Controller 0:

- specifies a tag depth of 16 for target 0
- specifies a tag depth of 64 for target 3
- specifies a tag depth of 8 for targets 4 and 5
- leaves target 6 at the default
- specifies a tag depth of 32 for targets 1,2,7-15
- All other targets retain the default depth.

```
tag_info:{{},{32,,32}}
```

On Controller 1:

- specifies a tag depth of 32 for targets 0 and 2
- All other targets retain the default depth.

seltime:[value]

Definition
Specifies the selection timeout value

Possible Values
0 = 256ms, 1 = 128ms, 2 = 64ms, 3 = 32ms

Default Value
0

dv: {value[,value...]}

Definition
Set Domain Validation Policy on a per-controller basis. Controllers may be omitted indicating that they should retain the default read streaming setting.

Possible Values

< 0	Use setting from serial EEPROM.
0	Disable DV
> 0	Enable DV

Default Value
SCSI-Select setting on controllers with a SCSI Select option for DV. Otherwise, on for controllers supporting U160 speeds and off for all other controller types.

Example:

```
dv:{-1,0,,1,1,0}
```

- On Controller 0 leave DV at its default setting.
- On Controller 1 disable DV.
- Skip configuration on Controller 2.
- On Controllers 3 and 4 enable DV.
- On Controller 5 disable DV.

Example:

```
options aic7xxx aic7xxx=verbose,no_probe,tag_info:{{},{,,10}},seltime:1
```

enables verbose logging, Disable EISA/VLB probing, and set tag depth on Controller 1/Target 2 to 10 tags.

4.6.4 4. Adaptec Customer Support

A Technical Support Identification (TSID) Number is required for Adaptec technical support.

- The 12-digit TSID can be found on the white barcode-type label included inside the box with your product. The TSID helps us provide more efficient service by accurately identifying your product and support status.

Support Options

- Search the Adaptec Support Knowledgebase (ASK) at http://ask.adaptec.com for articles, troubleshooting tips, and frequently asked questions about your product.
- For support via Email, submit your question to Adaptec's Technical Support Specialists at http://ask.adaptec.com/.

North America

- Visit our Web site at http://www.adaptec.com/.
- For information about Adaptec's support options, call 408-957-2550, 24 hours a day, 7 days a week.
- To speak with a Technical Support Specialist,
  - For hardware products, call 408-934-7274, Monday to Friday, 3:00 am to 5:00 pm, PDT.
  - For RAID and Fibre Channel products, call 321-207-2000, Monday to Friday, 3:00 am to 5:00 pm, PDT.

  To expedite your service, have your computer with you.
- To order Adaptec products, including accessories and cables, call 408-957-7274. To order cables online go to http://www.adaptec.com/buy-cables/.

Europe

- Visit our Web site at http://www.adaptec.com/en-US/_common/world_index.
- To speak with a Technical Support Specialist, call, or email,
  - German: +49 89 4366 5522, Monday-Friday, 9:00-17:00 CET, http://ask-de.adaptec.com/.
  - French: +49 89 4366 5533, Monday-Friday, 9:00-17:00 CET, http://ask-fr.adaptec.com/.
  - English: +49 89 4366 5544, Monday-Friday, 9:00-17:00 GMT, http://ask.adaptec.com/.
- You can order Adaptec cables online at http://www.adaptec.com/buy-cables/.

Japan

- Visit our web site at http://www.adaptec.co.jp/.
- To speak with a Technical Support Specialist, call +81 3 5308 6120, Monday-Friday, 9:00 a.m. to 12:00 p.m., 1:00 p.m. to 6:00 p.m.

Copyright © 2003 Adaptec Inc. 691 S. Milpitas Blvd., Milpitas CA 95035 USA.

All rights reserved.

You are permitted to redistribute, use and modify this README file in whole or in part in conjunction with redistribution of software governed by the General Public License, provided that the following conditions are met:

1. Redistributions of README file must retain the above copyright notice, this list of conditions, and the following disclaimer, without modification.
2. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.
3. Modifications or new contributions must be attributed in a copyright notice identifying the author ("Contributor") and added below the original copyright notice. The copyright notice is for purposes of identifying contributors and should not be deemed as permission to alter the permissions given by Adaptec.

THIS README FILE IS PROVIDED BY ADAPTEC AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTIES OF NON-INFRINGEMENT OR THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ADAPTEC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS README FILE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.7 ARECA FIRMWARE SPEC

4.7.1 Usage of IOP331 adapter

(All In/Out is in IOP331's view)

1. Message 0

- InitThread message and return code

2. Doorbell is used for RS-232 emulation

inDoorBell

bit0
data in ready zDRIVER DATA WRITE OK)

bit1
data out has been read (DRIVER DATA READ OK)

outDooeBell:

bit0
data out ready (IOP331 DATA WRITE OK)

bit1
data in has been read (IOP331 DATA READ OK)

3. Index Memory Usage

offset 0xf00	for RS232 out (request buffer)
offset 0xe00	for RS232 in (scratch buffer)
offset 0xa00	for inbound message code message_rwbuffer (driver send to IOP331)
offset 0xa00	for outbound message code message_rwbuffer (IOP331 send to driver)

4. RS-232 emulation

Currently 128 byte buffer is used:

1st uint32_t	Data length (1--124)
Byte 4--127	Max 124 bytes of data

5. PostQ

All SCSI Command must be sent through postQ:

(inbound queue port)
Request frame must be 32 bytes aligned:

#bit27--bit31
flag for post ccb

#bit0--bit26
real address (bit27--bit31) of post arcmsr_cdb

bit31	0 256 bytes frame 1 512 bytes frame
bit30	0 normal request 1 BIOS request
bit29	reserved
bit28	reserved
bit27	reserved

(outbount queue port)
Request reply:

#bit27--bit31
flag for reply

#bit0--bit26
real address (bit27--bit31) of reply arcmsr_cdb

bit31	must be 0 (for this type of reply)
bit30	reserved for BIOS handshake
bit29	reserved
bit28	0: no error, ignore AdapStatus/DevStatus/SenseData 1: Error, error code in AdapStatus/DevStatus/SenseData
bit27	reserved

6. BIOS request

All BIOS request is the same with request from PostQ

Except:

Request frame is sent from configuration space:

offset: 0x78	Request Frame (bit30 == 1)
offset: 0x18	writeonly to generate IRQ to IOP331

Completion of request:

```
(bit30 == 0, bit28==err flag)
```

7. Definition of SGL entry (structure)

8. Message1 Out - Diag Status Code (????)

9. Message0 message code

<table>
<tr><td>0x00</td><td>NOP</td></tr>
<tr><td>0x01</td><td>Get Config ->offset 0xa00 :for outbound message code message_rwbuffer (IOP331 send to driver)

<table><tr><td>Signature</td><td>0x87974060(4)</td></tr><tr><td>Request len</td><td>0x00000200(4)</td></tr><tr><td>numbers of queue</td><td>0x00000100(4)</td></tr><tr><td>SDRAM Size</td><td>0x00000100(4)-->256 MB</td></tr><tr><td>IDE Channels</td><td>0x00000008(4)</td></tr><tr><td>vendor</td><td>40 bytes char</td></tr><tr><td>model</td><td>8 bytes char</td></tr><tr><td>FirmVer</td><td>16 bytes char</td></tr><tr><td>Device Map</td><td>16 bytes char</td></tr><tr><td>FirmwareVersion</td><td>DWORD
Added for checking of new firmware capability</td></tr></table></td></tr>
<tr><td>0x02</td><td>Set Config ->offset 0xa00 :for inbound message code message_rwbuffer (driver send to IOP331)

<table><tr><td>Signature</td><td>0x87974063(4)</td></tr><tr><td>UPPER32 of Request Frame</td><td>(4)-->Driver Only</td></tr></table></td></tr>
<tr><td>0x03</td><td>Reset (Abort all queued Command)</td></tr>
<tr><td>0x04</td><td>Stop Background Activity</td></tr>
<tr><td>0x05</td><td>Flush Cache</td></tr>
<tr><td>0x06</td><td>Start Background Activity (re-start if background is halted)</td></tr>
<tr><td>0x07</td><td>Check If Host Command Pending (Novell May Need This Function)</td></tr>
<tr><td>0x08</td><td>Set controller time ->offset 0xa00 for inbound message code message_rwbuffer (driver to IOP331)

<table><tr><td>byte 0</td><td>0xaa <-- signature</td></tr><tr><td>byte 1</td><td>0x55 <-- signature</td></tr><tr><td>byte 2</td><td>year (04)</td></tr><tr><td>byte 3</td><td>month (1..12)</td></tr><tr><td>byte 4</td><td>date (1..31)</td></tr><tr><td>byte 5</td><td>hour (0..23)</td></tr><tr><td>byte 6</td><td>minute (0..59)</td></tr><tr><td>byte 7</td><td>second (0..59)</td></tr></table></td></tr>
</table>

4.7.2 RS-232 Interface for Areca Raid Controller

The low level command interface is exclusive with VT100 terminal

1. Sequence of command execution

(A) **Header**
3 bytes sequence (0x5E, 0x01, 0x61)

(B) **Command block**
variable length of data including length, command code, data and checksum byte

(C) **Return data**
variable length of data

2. Command block

(A) **1st byte**
command block length (low byte)

(B) **2nd byte**
command block length (high byte)

Note: command block length shouldn't > 2040 bytes, length excludes these two bytes

---

(C) **3rd byte**
command code

(D) **4th and following bytes**
variable length data bytes

depends on command code

(E) last byte checksum byte (sum of 1st byte until last data byte)

3. Command code and associated data

The following are command code defined in raid controller Command code 0x10--0x1? are used for system level management, no password checking is needed and should be implemented in separate well controlled utility and not for end user access. Command code 0x20--0x?? always check the password, password must be entered to enable these command:

```
enum
{
        GUI_SET_SERIAL=0x10,
        GUI_SET_VENDOR,
        GUI_SET_MODEL,
        GUI_IDENTIFY,
        GUI_CHECK_PASSWORD,
        GUI_LOGOUT,
```

```
        GUI_HTTP,
        GUI_SET_ETHERNET_ADDR,
        GUI_SET_LOGO,
        GUI_POLL_EVENT,
        GUI_GET_EVENT,
        GUI_GET_HW_MONITOR,
        //    GUI_QUICK_CREATE=0x20, (function removed)
        GUI_GET_INFO_R=0x20,
        GUI_GET_INFO_V,
        GUI_GET_INFO_P,
        GUI_GET_INFO_S,
        GUI_CLEAR_EVENT,
        GUI_MUTE_BEEPER=0x30,
        GUI_BEEPER_SETTING,
        GUI_SET_PASSWORD,
        GUI_HOST_INTERFACE_MODE,
        GUI_REBUILD_PRIORITY,
        GUI_MAX_ATA_MODE,
        GUI_RESET_CONTROLLER,
        GUI_COM_PORT_SETTING,
        GUI_NO_OPERATION,
        GUI_DHCP_IP,
        GUI_CREATE_PASS_THROUGH=0x40,
        GUI_MODIFY_PASS_THROUGH,
        GUI_DELETE_PASS_THROUGH,
        GUI_IDENTIFY_DEVICE,
        GUI_CREATE_RAIDSET=0x50,
        GUI_DELETE_RAIDSET,
        GUI_EXPAND_RAIDSET,
        GUI_ACTIVATE_RAIDSET,
        GUI_CREATE_HOT_SPARE,
        GUI_DELETE_HOT_SPARE,
        GUI_CREATE_VOLUME=0x60,
        GUI_MODIFY_VOLUME,
        GUI_DELETE_VOLUME,
        GUI_START_CHECK_VOLUME,
        GUI_STOP_CHECK_VOLUME
};
```

Command description

GUI_SET_SERIAL

Set the controller serial#

byte 0,1	length
byte 2	command code 0x10
byte 3	password length (should be 0x0f)
byte 4-0x13	should be ”ArEcATecHnoLogY”
byte 0x14--0x23	Serial number string (must be 16 bytes)

GUI_SET_VENDOR
Set vendor string for the controller

byte 0,1	length
byte 2	command code 0x11
byte 3	password length (should be 0x08)
byte 4-0x13	should be ”ArEcAvAr”
byte 0x14--0x3B	vendor string (must be 40 bytes)

GUI_SET_MODEL
Set the model name of the controller

byte 0,1	length
byte 2	command code 0x12
byte 3	password length (should be 0x08)
byte 4-0x13	should be ”ArEcAvAr”
byte 0x14--0x1B	model string (must be 8 bytes)

GUI_IDENTIFY
Identify device

byte 0,1	length
byte 2	**command code 0x13** return ”Areca RAID Subsystem ”

GUI_CHECK_PASSWORD
Verify password

byte 0,1	length
byte 2	command code 0x14
byte 3	password length
byte 4-0x??	user password to be checked

GUI_LOGOUT
Logout GUI (force password checking on next command)

byte 0,1	length
byte 2	command code 0x15

GUI_HTTP
HTTP interface (reserved for Http proxy service)(0x16)

GUI_SET_ETHERNET_ADDR
Set the ethernet MAC address

byte 0,1	length
byte 2	command code 0x17
byte 3	password length (should be 0x08)
byte 4-0x13	should be ”ArEcAvAr”
byte 0x14--0x19	Ethernet MAC address (must be 6 bytes)

GUI_SET_LOGO
Set logo in HTTP

byte 0,1	length
byte 2	command code 0x18
byte 3	Page# (0/1/2/3) (0xff --> clear OEM logo)
byte 4/5/6/7	0x55/0xaa/0xa5/0x5a
byte 8	TITLE.JPG data (each page must be 2000 bytes) **Note:** page0 1st 2 byte must be actual length of the JPG file

GUI_POLL_EVENT
Poll If Event Log Changed

byte 0,1	length
byte 2	command code 0x19

GUI_GET_EVENT
Read Event

byte 0,1	length
byte 2	command code 0x1a
byte 3	Event Page (0:1st page/1/2/3:last page)

GUI_GET_HW_MONITOR
Get HW monitor data

byte 0,1	length
byte 2	command code 0x1b
byte 3	# of FANs(example 2)
byte 4	# of Voltage sensor(example 3)
byte 5	# of temperature sensor(example 2)
byte 6	# of power
byte 7/8	Fan#0 (RPM)
byte 9/10	Fan#1
byte 11/12	Voltage#0 original value in *1000
byte 13/14	Voltage#0 value
byte 15/16	Voltage#1 org
byte 17/18	Voltage#1
byte 19/20	Voltage#2 org
byte 21/22	Voltage#2
byte 23	Temp#0
byte 24	Temp#1
byte 25	Power indicator (bit0 power#0, bit1 power#1)
byte 26	UPS indicator

GUI_QUICK_CREATE

Quick create raid/volume set

byte 0,1	length
byte 2	command code 0x20
byte 3/4/5/6	raw capacity
byte 7	raid level
byte 8	stripe size
byte 9	spare
byte 10/11/12/13	device mask (the devices to create raid/volume)

This function is removed, application like to implement quick create function need to use GUI_CREATE_RAIDSET and GUI_CREATE_VOLUMESET function.

GUI_GET_INFO_R

Get Raid Set Information

byte 0,1	length
byte 2	command code 0x20
byte 3	raidset#

```
typedef struct sGUI_RAIDSET
{
        BYTE grsRaidSetName[16];
        DWORD grsCapacity;
        DWORD grsCapacityX;
        DWORD grsFailMask;
        BYTE grsDevArray[32];
        BYTE grsMemberDevices;
```

```
        BYTE grsNewMemberDevices;
        BYTE grsRaidState;
        BYTE grsVolumes;
        BYTE grsVolumeList[16];
        BYTE grsRes1;
        BYTE grsRes2;
        BYTE grsRes3;
        BYTE grsFreeSegments;
        DWORD grsRawStripes[8];
        DWORD grsRes4;
        DWORD grsRes5; //     Total to 128 bytes
        DWORD grsRes6; //     Total to 128 bytes
} sGUI_RAIDSET, *pGUI_RAIDSET;
```

GUI_GET_INFO_V
Get Volume Set Information

byte 0,1	length
byte 2	command code 0x21
byte 3	volumeset#

```
typedef struct sGUI_VOLUMESET
{
        BYTE gvsVolumeName[16]; //     16
        DWORD gvsCapacity;
        DWORD gvsCapacityX;
        DWORD gvsFailMask;
        DWORD gvsStripeSize;
        DWORD gvsNewFailMask;
        DWORD gvsNewStripeSize;
        DWORD gvsVolumeStatus;
        DWORD gvsProgress; //     32
        sSCSI_ATTR gvsScsi;
        BYTE gvsMemberDisks;
        BYTE gvsRaidLevel; //     8
        BYTE gvsNewMemberDisks;
        BYTE gvsNewRaidLevel;
        BYTE gvsRaidSetNumber;
        BYTE gvsRes0; //     4
        BYTE gvsRes1[4]; //     64 bytes
} sGUI_VOLUMESET, *pGUI_VOLUMESET;
```

GUI_GET_INFO_P
Get Physical Drive Information

byte 0,1	length
byte 2	command code 0x22
byte 3	drive # (from 0 to max-channels - 1)

```
typedef struct sGUI_PHY_DRV
{
        BYTE gpdModelName[40];
        BYTE gpdSerialNumber[20];
        BYTE gpdFirmRev[8];
        DWORD gpdCapacity;
        DWORD gpdCapacityX; //     Reserved for expansion
        BYTE gpdDeviceState;
        BYTE gpdPioMode;
        BYTE gpdCurrentUdmaMode;
        BYTE gpdUdmaMode;
        BYTE gpdDriveSelect;
        BYTE gpdRaidNumber; //     0xff if not belongs to a raid set
        sSCSI_ATTR gpdScsi;
        BYTE gpdReserved[40]; //     Total to 128 bytes
} sGUI_PHY_DRV, *pGUI_PHY_DRV;
```

GUI_GET_INFO_S

Get System Information

byte 0,1	length
byte 2	command code 0x23

```
typedef struct sCOM_ATTR
{
        BYTE comBaudRate;
        BYTE comDataBits;
        BYTE comStopBits;
        BYTE comParity;
        BYTE comFlowControl;
} sCOM_ATTR, *pCOM_ATTR;
typedef struct sSYSTEM_INFO
{
        BYTE gsiVendorName[40];
        BYTE gsiSerialNumber[16];
        BYTE gsiFirmVersion[16];
        BYTE gsiBootVersion[16];
        BYTE gsiMbVersion[16];
        BYTE gsiModelName[8];
        BYTE gsiLocalIp[4];
        BYTE gsiCurrentIp[4];
        DWORD gsiTimeTick;
        DWORD gsiCpuSpeed;
        DWORD gsiICache;
        DWORD gsiDCache;
        DWORD gsiScache;
        DWORD gsiMemorySize;
        DWORD gsiMemorySpeed;
        DWORD gsiEvents;
        BYTE gsiMacAddress[6];
```

```
        BYTE gsiDhcp;
        BYTE gsiBeeper;
        BYTE gsiChannelUsage;
        BYTE gsiMaxAtaMode;
        BYTE gsiSdramEcc; //     1:if ECC enabled
        BYTE gsiRebuildPriority;
        sCOM_ATTR gsiComA; //     5 bytes
        sCOM_ATTR gsiComB; //     5 bytes
        BYTE gsiIdeChannels;
        BYTE gsiScsiHostChannels;
        BYTE gsiIdeHostChannels;
        BYTE gsiMaxVolumeSet;
        BYTE gsiMaxRaidSet;
        BYTE gsiEtherPort; //     1:if ether net port supported
        BYTE gsiRaid6Engine; //     1:Raid6 engine supported
        BYTE gsiRes[75];
} sSYSTEM_INFO, *pSYSTEM_INFO;
```

GUI_CLEAR_EVENT
Clear System Event

byte 0,1	length
byte 2	command code 0x24

GUI_MUTE_BEEPER
Mute current beeper

byte 0,1	length
byte 2	command code 0x30

GUI_BEEPER_SETTING
Disable beeper

byte 0,1	length
byte 2	command code 0x31
byte 3	0->disable, 1->enable

GUI_SET_PASSWORD
Change password

byte 0,1	length
byte 2	command code 0x32
byte 3	pass word length (must <= 15)
byte 4	password (must be alpha-numerical)

GUI_HOST_INTERFACE_MODE
Set host interface mode

byte 0,1	length
byte 2	command code 0x33
byte 3	0->Independent, 1->cluster

GUI_REBUILD_PRIORITY
Set rebuild priority

byte 0,1	length
byte 2	command code 0x34
byte 3	0/1/2/3 (low->high)

GUI_MAX_ATA_MODE
Set maximum ATA mode to be used

byte 0,1	length
byte 2	command code 0x35
byte 3	0/1/2/3 (133/100/66/33)

GUI_RESET_CONTROLLER
Reset Controller

byte 0,1	length
byte 2	command code 0x36 * Response with VT100 screen (discard it)

GUI_COM_PORT_SETTING
COM port setting

byte 0,1	length
byte 2	command code 0x37
byte 3	0->COMA (term port), 1->COMB (debug port)
byte 4	0/1/2/3/4/5/6/7 (1200/2400/4800/9600/19200/38400/57600/115200)
byte 5	data bit (0:7 bit, 1:8 bit must be 8 bit)
byte 6	stop bit (0:1, 1:2 stop bits)
byte 7	parity (0:none, 1:off, 2:even)
byte 8	flow control (0:none, 1:xon/xoff, 2:hardware => must use none)

GUI_NO_OPERATION
No operation

byte 0,1	length
byte 2	command code 0x38

GUI_DHCP_IP
Set DHCP option and local IP address

byte 0,1	length
byte 2	command code 0x39
byte 3	0:dhcp disabled, 1:dhcp enabled
byte 4/5/6/7	IP address

GUI_CREATE_PASS_THROUGH
Create pass through disk

byte 0,1	length
byte 2	command code 0x40
byte 3	device #
byte 4	scsi channel (0/1)
byte 5	scsi id (0-->15)
byte 6	scsi lun (0-->7)
byte 7	tagged queue (1 enabled)
byte 8	cache mode (1 enabled)
byte 9	max speed (0/1/2/3/4, async/20/40/80/160 for scsi) (0/1/2/3/4, 33/66/100/133/150 for ide)

GUI_MODIFY_PASS_THROUGH
Modify pass through disk

byte 0,1	length
byte 2	command code 0x41
byte 3	device #
byte 4	scsi channel (0/1)
byte 5	scsi id (0-->15)
byte 6	scsi lun (0-->7)
byte 7	tagged queue (1 enabled)
byte 8	cache mode (1 enabled)
byte 9	max speed (0/1/2/3/4, async/20/40/80/160 for scsi) (0/1/2/3/4, 33/66/100/133/150 for ide)

GUI_DELETE_PASS_THROUGH
Delete pass through disk

byte 0,1	length
byte 2	command code 0x42
byte 3	device# to be deleted

GUI_IDENTIFY_DEVICE
Identify Device

byte 0,1	length
byte 2	command code 0x43
byte 3	Flash Method (0:flash selected, 1:flash not selected)
byte 4/5/6/7	IDE device mask to be flashed .. Note:: no response data available

GUI_CREATE_RAIDSET
Create Raid Set

byte 0,1	length
byte 2	command code 0x50
byte 3/4/5/6	device mask
byte 7-22	raidset name (if byte 7 == 0:use default)

GUI_DELETE_RAIDSET
Delete Raid Set

byte 0,1	length
byte 2	command code 0x51
byte 3	raidset#

GUI_EXPAND_RAIDSET
Expand Raid Set

byte 0,1	length
byte 2	command code 0x52
byte 3	raidset#
byte 4/5/6/7	device mask for expansion
byte 8/9/10	(8:0 no change, 1 change, 0xff:terminate, 9:new raid level, 10:new stripe size 0/1/2/3/4/5->4/8/16/32/64/128K)
byte 11/12/13	repeat for each volume in the raidset

GUI_ACTIVATE_RAIDSET
Activate incomplete raid set

byte 0,1	length
byte 2	command code 0x53
byte 3	raidset#

GUI_CREATE_HOT_SPARE
Create hot spare disk

byte 0,1	length
byte 2	command code 0x54
byte 3/4/5/6	device mask for hot spare creation

GUI_DELETE_HOT_SPARE
Delete hot spare disk

byte 0,1	length
byte 2	command code 0x55
byte 3/4/5/6	device mask for hot spare deletion

GUI_CREATE_VOLUME
Create volume set

byte 0,1	length
byte 2	command code 0x60
byte 3	raidset#
byte 4-19	volume set name (if byte4 == 0, use default)
byte 20-27	volume capacity (blocks)
byte 28	raid level
byte 29	stripe size (0/1/2/3/4/5->4/8/16/32/64/128K)
byte 30	channel
byte 31	ID
byte 32	LUN
byte 33	1 enable tag
byte 34	1 enable cache
byte 35	speed (0/1/2/3/4->async/20/40/80/160 for scsi) (0/1/2/3/4->33/66/100/133/150 for IDE)
byte 36	1 to select quick init

GUI_MODIFY_VOLUME
Modify volume Set

byte 0,1	length
byte 2	command code 0x61
byte 3	volumeset#
byte 4-19	new volume set name (if byte4 == 0, not change)
byte 20-27	new volume capacity (reserved)
byte 28	new raid level
byte 29	new stripe size (0/1/2/3/4/5->4/8/16/32/64/128K)
byte 30	new channel
byte 31	new ID
byte 32	new LUN
byte 33	1 enable tag
byte 34	1 enable cache
byte 35	speed (0/1/2/3/4->async/20/40/80/160 for scsi) (0/1/2/3/4->33/66/100/133/150 for IDE)

GUI_DELETE_VOLUME
Delete volume set

byte 0,1	length
byte 2	command code 0x62
byte 3	volumeset#

GUI_START_CHECK_VOLUME
Start volume consistency check

byte 0,1	length
byte 2	command code 0x63
byte 3	volumeset#

GUI_STOP_CHECK_VOLUME
Stop volume consistency check

byte 0,1	length
byte 2	command code 0x64

4. Returned data

(A) Header 3 bytes sequence (0x5E, 0x01, 0x61)

(B) Length 2 bytes (low byte 1st, excludes length and checksum byte)

(C) status or data:

1) If length == 1 ==> 1 byte status code:

```
#define GUI_OK                    0x41
#define GUI_RAIDSET_NOT_NORMAL    0x42
#define GUI_VOLUMESET_NOT_NORMAL  0x43
#define GUI_NO_RAIDSET            0x44
#define GUI_NO_VOLUMESET          0x45
#define GUI_NO_PHYSICAL_DRIVE     0x46
#define GUI_PARAMETER_ERROR       0x47
#define GUI_UNSUPPORTED_COMMAND   0x48
#define GUI_DISK_CONFIG_CHANGED   0x49
#define GUI_INVALID_PASSWORD      0x4a
#define GUI_NO_DISK_SPACE         0x4b
#define GUI_CHECKSUM_ERROR        0x4c
#define GUI_PASSWORD_REQUIRED     0x4d
```

2) If length > 1:

data block returned from controller and the contents depends on the command code

(E) Checksum checksum of length and status or data byte

4.8 Linux driver for Brocade FC/FCOE adapters

4.8.1 Supported Hardware

bfa 3.0.2.2 driver supports all Brocade FC/FCOE adapters. Below is a list of adapter models with corresponding PCIIDs.

PCIID	Model
1657:0013:1657:0014	425 4Gbps dual port FC HBA
1657:0013:1657:0014	825 8Gbps PCIe dual port FC HBA
1657:0013:103c:1742	HP 82B 8Gbps PCIedual port FC HBA
1657:0013:103c:1744	HP 42B 4Gbps dual port FC HBA
1657:0017:1657:0014	415 4Gbps single port FC HBA
1657:0017:1657:0014	815 8Gbps single port FC HBA
1657:0017:103c:1741	HP 41B 4Gbps single port FC HBA
1657:0017:103c 1743	HP 81B 8Gbps single port FC HBA
1657:0021:103c:1779	804 8Gbps FC HBA for HP Bladesystem c-class
1657:0014:1657:0014	1010 10Gbps single port CNA - FCOE
1657:0014:1657:0014	1020 10Gbps dual port CNA - FCOE
1657:0014:1657:0014	1007 10Gbps dual port CNA - FCOE
1657:0014:1657:0014	1741 10Gbps dual port CNA - FCOE
1657:0022:1657:0024	1860 16Gbps FC HBA
1657:0022:1657:0022	1860 10Gbps CNA - FCOE

4.8.2 Firmware download

The latest Firmware package for 3.0.2.2 bfa driver can be found at:

http://www.brocade.com/services-support/drivers-downloads/adapters/Linux.page

and then click following respective util package link:

Version	Link
v3.0.0.0	Linux Adapter Firmware package for RHEL 6.2, SLES 11SP2

4.8.3 Configuration & Management utility download

The latest driver configuration & management utility for 3.0.2.2 bfa driver can be found at:

http://www.brocade.com/services-support/drivers-downloads/adapters/Linux.page

and then click following respective util package link

Version	Link
v3.0.2.0	Linux Adapter Firmware package for RHEL 6.2, SLES 11SP2

4.8.4 Documentation

The latest Administration's Guide, Installation and Reference Manual, Troubleshooting Guide, and Release Notes for the corresponding out-of-box driver can be found at:

http://www.brocade.com/services-support/drivers-downloads/adapters/Linux.page

and use the following inbox and out-of-box driver version mapping to find the corresponding documentation:

Inbox Version	Out-of-box Version
v3.0.2.2	v3.0.0.0

4.8.5 Support

For general product and support info, go to the Brocade website at:

http://www.brocade.com/services-support/index.page

4.9 Operating FCoE using bnx2fc

Broadcom FCoE offload through bnx2fc is full stateful hardware offload that cooperates with all interfaces provided by the Linux ecosystem for FC/FCoE and SCSI controllers. As such, FCoE functionality, once enabled is largely transparent. Devices discovered on the SAN will be registered and unregistered automatically with the upper storage layers.

Despite the fact that the Broadcom's FCoE offload is fully offloaded, it does depend on the state of the network interfaces to operate. As such, the network interface (e.g. eth0) associated with the FCoE offload initiator must be 'up'. It is recommended that the network interfaces be configured to be brought up automatically at boot time.

Furthermore, the Broadcom FCoE offload solution creates VLAN interfaces to support the VLANs that have been discovered for FCoE operation (e.g. eth0.1001-fcoe). Do not delete or disable these interfaces or FCoE operation will be disrupted.

4.9.1 Driver Usage Model:

1. Ensure that fcoe-utils package is installed.

2. Configure the interfaces on which bnx2fc driver has to operate on. Here are the steps to configure:

   a. cd /etc/fcoe

   b. copy cfg-ethx to cfg-eth5 if FCoE has to be enabled on eth5.

   c. Repeat this for all the interfaces where FCoE has to be enabled.

   d. Edit all the cfg-eth files to set ”no” for DCB_REQUIRED** field, and ”yes” for AUTO_VLAN.

   e. Other configuration parameters should be left as default

3. Ensure that ”bnx2fc” is in SUPPORTED_DRIVERS list in /etc/fcoe/config.

4. Start fcoe service. (service fcoe start). If Broadcom devices are present in the system, bnx2fc driver would automatically claim the interfaces, starts vlan discovery and log into the targets.

5. ”Symbolic Name” in 'fcoeadm -i' output would display if bnx2fc has claimed the interface.

Eg:

```
[root@bh2 ~]# fcoeadm -i
   Description:      NetXtreme II BCM57712 10 Gigabit Ethernet
   Revision:         01
```

```
  Manufacturer:     Broadcom Corporation
  Serial Number:    0010186FD558
  Driver:           bnx2x 1.70.00-0
  Number of Ports:  2

      Symbolic Name:     bnx2fc v1.0.5 over eth5.4
      OS Device Name:    host11
      Node Name:         0x10000010186FD559
      Port Name:         0x20000010186FD559
      FabricName:        0x2001000DECB3B681
      Speed:             10 Gbit
      Supported Speed:   10 Gbit
      MaxFrameSize:      2048
      FC-ID (Port ID):   0x0F0377
      State:             Online
```

6. Verify the vlan discovery is performed by running ifconfig and notice <INTERFACE>.<VLAN>-fcoe interfaces are automatically created.

Refer to fcoeadm manpage for more information on fcoeadm operations to create/destroy interfaces or to display lun/target information.

4.9.2 NOTE

** Broadcom FCoE capable devices implement a DCBX/LLDP client on-chip. Only one LLDP client is allowed per interface. For proper operation all host software based DCBX/LLDP clients (e.g. lldpad) must be disabled. To disable lldpad on a given interface, run the following command:

```
lldptool set-lldp -i <interface_name> adminStatus=disabled
```

4.10 BusLogic MultiMaster and FlashPoint SCSI Driver for Linux

Version 2.0.15 for Linux 2.0

Version 2.1.15 for Linux 2.1

PRODUCTION RELEASE

17 August 1998

Leonard N. Zubkoff

Dandelion Digital

lnz@dandelion.com

Copyright 1995-1998 by Leonard N. Zubkoff <lnz@dandelion.com>

4.10.1 Introduction

BusLogic, Inc. designed and manufactured a variety of high performance SCSI host adapters which share a common programming interface across a diverse collection of bus architectures by virtue of their MultiMaster ASIC technology. BusLogic was acquired by Mylex Corporation in February 1996, but the products supported by this driver originated under the BusLogic name and so that name is retained in the source code and documentation.

This driver supports all present BusLogic MultiMaster Host Adapters, and should support any future MultiMaster designs with little or no modification. More recently, BusLogic introduced the FlashPoint Host Adapters, which are less costly and rely on the host CPU, rather than including an onboard processor. Despite not having an onboard CPU, the FlashPoint Host Adapters perform very well and have very low command latency. BusLogic has recently provided me with the FlashPoint Driver Developer's Kit, which comprises documentation and freely redistributable source code for the FlashPoint SCCB Manager. The SCCB Manager is the library of code that runs on the host CPU and performs functions analogous to the firmware on the MultiMaster Host Adapters. Thanks to their having provided the SCCB Manager, this driver now supports the FlashPoint Host Adapters as well.

My primary goals in writing this completely new BusLogic driver for Linux are to achieve the full performance that BusLogic SCSI Host Adapters and modern SCSI peripherals are capable of, and to provide a highly robust driver that can be depended upon for high performance mission critical applications. All of the major performance features can be configured from the Linux kernel command line or at module initialization time, allowing individual installations to tune driver performance and error recovery to their particular needs.

The latest information on Linux support for BusLogic SCSI Host Adapters, as well as the most recent release of this driver and the latest firmware for the BT-948/958/958D, will always be available from my Linux Home Page at URL "http://sourceforge.net/projects/dandelion/".

Bug reports should be sent via electronic mail to "lnz@dandelion.com". Please include with the bug report the complete configuration messages reported by the driver and SCSI subsystem at startup, along with any subsequent system messages relevant to SCSI operations, and a detailed description of your system's hardware configuration.

Mylex has been an excellent company to work with and I highly recommend their products to the Linux community. In November 1995, I was offered the opportunity to become a beta test site for their latest MultiMaster product, the BT-948 PCI Ultra SCSI Host Adapter, and then again for the BT-958 PCI Wide Ultra SCSI Host Adapter in January 1996. This was mutually beneficial since Mylex received a degree and kind of testing that their own testing group cannot readily achieve, and the Linux community has available high performance host adapters that have been well tested with Linux even before being brought to market. This relationship has also given me the opportunity to interact directly with their technical staff, to understand more about the internal workings of their products, and in turn to educate them about the needs and potential of the Linux community.

More recently, Mylex has reaffirmed the company's interest in supporting the Linux community, and I am now working on a Linux driver for the DAC960 PCI RAID Controllers. Mylex's interest and support is greatly appreciated.

Unlike some other vendors, if you contact Mylex Technical Support with a problem and are running Linux, they will not tell you that your use of their products is unsupported. Their latest product marketing literature even states "Mylex SCSI host adapters are compatible with all major operating systems including: ... Linux ...".

Mylex Corporation is located at 34551 Ardenwood Blvd., Fremont, California 94555, USA and can be reached at 510/796-6100 or on the World Wide Web at http://www.mylex.com. Mylex HBA Technical Support can be reached by electronic mail at techsup@mylex.com, by Voice at 510/608-2400, or by FAX at 510/745-7715. Contact information for offices in Europe and Japan is available on the Web site.

4.10.2 Driver Features

Configuration Reporting and Testing

During system initialization, the driver reports extensively on the host adapter hardware configuration, including the synchronous transfer parameters requested and negotiated with each target device. AutoSCSI settings for Synchronous Negotiation, Wide Negotiation, and Disconnect/Reconnect are reported for each target device, as well as the status of Tagged Queuing. If the same setting is in effect for all target devices, then a single word or phrase is used; otherwise, a letter is provided for each target device to indicate the individual status. The following examples should clarify this reporting format:

Synchronous Negotiation: Ultra

Synchronous negotiation is enabled for all target devices and the host adapter will attempt to negotiate for 20.0 mega-transfers/second.

Synchronous Negotiation: Fast

Synchronous negotiation is enabled for all target devices and the host adapter will attempt to negotiate for 10.0 mega-transfers/second.

Synchronous Negotiation: Slow

Synchronous negotiation is enabled for all target devices and the host adapter will attempt to negotiate for 5.0 mega-transfers/second.

Synchronous Negotiation: Disabled

Synchronous negotiation is disabled and all target devices are limited to asynchronous operation.

Synchronous Negotiation: UFSNUUU#UUUUUUUU

Synchronous negotiation to Ultra speed is enabled for target devices 0 and 4 through 15, to Fast speed for target device 1, to Slow speed for target device 2, and is not permitted to target device 3. The host adapter's SCSI ID is represented by the ”#”.

The status of Wide Negotiation, Disconnect/Reconnect, and Tagged Queuing are reported as ”Enabled”, Disabled”, or a sequence of ”Y” and ”N” letters.

Performance Features

BusLogic SCSI Host Adapters directly implement SCSI-2 Tagged Queuing, and so support has been included in the driver to utilize tagged queuing with any target devices that report having the tagged queuing capability. Tagged queuing allows for multiple outstanding commands to be issued to each target device or logical unit, and can improve I/O performance substantially. In addition, BusLogic's Strict Round Robin Mode is used to optimize host adapter performance, and scatter/gather I/O can support as many segments as can be effectively utilized by the Linux I/O subsystem. Control over the use of tagged queuing for each target device as well as individual selection of the tagged queue depth is available through driver options provided on the kernel command line or at module initialization time. By default, the queue depth is determined automatically based on the host adapter's total queue depth and the number, type, speed, and capabilities of the target devices found. In addition, tagged queuing is automatically disabled whenever the host adapter firmware version is known not to implement it correctly, or whenever a tagged queue depth of 1 is selected. Tagged queuing is also disabled for individual target devices if disconnect/reconnect is disabled for that device.

Robustness Features

The driver implements extensive error recovery procedures. When the higher level parts of the SCSI subsystem request that a timed out command be reset, a selection is made between a full host adapter hard reset and SCSI bus reset versus sending a bus device reset message to the individual target device based on the recommendation of the SCSI subsystem. Error recovery strategies are selectable through driver options individually for each target device, and also include sending a bus device reset to the specific target device associated with the command being reset, as well as suppressing error recovery entirely to avoid perturbing an improperly functioning device. If the bus device reset error recovery strategy is selected and sending a bus device reset does not restore correct operation, the next command that is reset will force a full host adapter hard reset and SCSI bus reset. SCSI bus resets caused by other devices and detected by the host adapter are also handled by issuing a soft reset to the host adapter and re-initialization. Finally, if tagged queuing is active and more than one command reset occurs in a 10 minute interval, or if a command reset occurs within the first 10 minutes of operation, then tagged queuing will be disabled for that target device. These error recovery options improve overall system robustness by preventing individual errant devices from causing the system as a whole to lock up or crash, and thereby allowing a clean shutdown and restart after the offending component is removed.

PCI Configuration Support

On PCI systems running kernels compiled with PCI BIOS support enabled, this driver will interrogate the PCI configuration space and use the I/O port addresses assigned by the system BIOS, rather than the ISA compatible I/O port addresses. The ISA compatible I/O port address is then disabled by the driver. On PCI systems it is also recommended that the AutoSCSI utility be used to disable the ISA compatible I/O port entirely as it is not necessary. The ISA compatible I/O port is disabled by default on the BT-948/958/958D.

/proc File System Support

Copies of the host adapter configuration information together with updated data transfer and error recovery statistics are available through the /proc/scsi/BusLogic/<N> interface.

Shared Interrupts Support

On systems that support shared interrupts, any number of BusLogic Host Adapters may share the same interrupt request channel.

4.10.3 Supported Host Adapters

The following list comprises the supported BusLogic SCSI Host Adapters as of the date of this document. It is recommended that anyone purchasing a BusLogic Host Adapter not in the following table contact the author beforehand to verify that it is or will be supported.

FlashPoint Series PCI Host Adapters:

FlashPoint LT (BT-930)	Ultra SCSI-3
FlashPoint LT (BT-930R)	Ultra SCSI-3 with RAIDPlus
FlashPoint LT (BT-920)	Ultra SCSI-3 (BT-930 without BIOS)
FlashPoint DL (BT-932)	Dual Channel Ultra SCSI-3
FlashPoint DL (BT-932R)	Dual Channel Ultra SCSI-3 with RAIDPlus
FlashPoint LW (BT-950)	Wide Ultra SCSI-3
FlashPoint LW (BT-950R)	Wide Ultra SCSI-3 with RAIDPlus
FlashPoint DW (BT-952)	Dual Channel Wide Ultra SCSI-3
FlashPoint DW (BT-952R)	Dual Channel Wide Ultra SCSI-3 with RAIDPlus

MultiMaster ”W” Series Host Adapters:

BT-948	PCI	Ultra SCSI-3
BT-958	PCI	Wide Ultra SCSI-3
BT-958D	PCI	Wide Differential Ultra SCSI-3

MultiMaster ”C” Series Host Adapters:

BT-946C	PCI	Fast SCSI-2
BT-956C	PCI	Wide Fast SCSI-2
BT-956CD	PCI	Wide Differential Fast SCSI-2
BT-445C	VLB	Fast SCSI-2
BT-747C	EISA	Fast SCSI-2
BT-757C	EISA	Wide Fast SCSI-2
BT-757CD	EISA	Wide Differential Fast SCSI-2

MultiMaster ”S” Series Host Adapters:

BT-445S	VLB	Fast SCSI-2
BT-747S	EISA	Fast SCSI-2
BT-747D	EISA	Differential Fast SCSI-2
BT-757S	EISA	Wide Fast SCSI-2
BT-757D	EISA	Wide Differential Fast SCSI-2
BT-742A	EISA	SCSI-2 (742A revision H)

MultiMaster ”A” Series Host Adapters:

BT-742A	EISA	SCSI-2 (742A revisions A - G)

AMI FastDisk Host Adapters that are true BusLogic MultiMaster clones are also supported by this driver.

BusLogic SCSI Host Adapters are available packaged both as bare boards and as retail kits. The BT- model numbers above refer to the bare board packaging. The retail kit model numbers are found by replacing BT- with KT- in the above list. The retail kit includes the bare board and manual as well as cabling and driver media and documentation that are not provided with bare boards.

4.10.4 FlashPoint Installation Notes

RAIDPlus Support

FlashPoint Host Adapters now include RAIDPlus, Mylex's bootable software RAID. RAIDPlus is not supported on Linux, and there are no plans to support it. The MD driver in Linux 2.0 provides for concatenation (LINEAR) and striping (RAID-0), and support for mirroring (RAID-1), fixed parity (RAID-4), and distributed parity (RAID-5) is available separately. The built-in Linux RAID support is generally more flexible and is expected to perform better than RAIDPlus, so there is little impetus to include RAIDPlus support in the BusLogic driver.

Enabling UltraSCSI Transfers

FlashPoint Host Adapters ship with their configuration set to ”Factory Default” settings that are conservative and do not allow for UltraSCSI speed to be negotiated. This results in fewer problems when these host adapters are installed in systems with cabling or termination that is not sufficient for UltraSCSI operation, or where existing SCSI devices do not properly respond to synchronous transfer negotiation for UltraSCSI speed. AutoSCSI may be used to load ”Optimum Performance” settings which allow UltraSCSI speed to be negotiated with all devices, or UltraSCSI speed can be enabled on an individual basis. It is recommended that SCAM be manually disabled after the ”Optimum Performance” settings are loaded.

4.10.5 BT-948/958/958D Installation Notes

The BT-948/958/958D PCI Ultra SCSI Host Adapters have some features which may require attention in some circumstances when installing Linux.

PCI I/O Port Assignments

When configured to factory default settings, the BT-948/958/958D will only recognize the PCI I/O port assignments made by the motherboard's PCI BIOS. The BT-948/958/958D will not respond to any of the ISA compatible I/O ports that previous BusLogic SCSI Host Adapters respond to. This driver supports the PCI I/O port assignments, so this is the preferred configuration. However, if the obsolete BusLogic driver must be used for any reason, such as a Linux distribution that does not yet use this driver in its boot kernel, BusLogic has provided an AutoSCSI configuration option to enable a legacy ISA compatible I/O port.

To enable this backward compatibility option, invoke the AutoSCSI utility via Ctrl-B at system startup and select ”Adapter Configuration”, ”View/Modify Configuration”, and then change the ”ISA Compatible Port” setting from ”Disable” to ”Primary” or ”Alternate”. Once this driver has been installed, the ”ISA Compatible Port” option should be set back to ”Disable” to avoid possible future I/O port conflicts. The older BT-946C/956C/956CD also have this configuration option, but the factory default setting is ”Primary”.

PCI Slot Scanning Order

In systems with multiple BusLogic PCI Host Adapters, the order in which the PCI slots are scanned may appear reversed with the BT-948/958/958D as compared to the BT-946C/956C/956CD. For booting from a SCSI disk to work correctly, it is necessary that the host adapter's BIOS and the kernel agree on which disk is the boot device, which requires that they recognize the PCI host adapters in the same order. The motherboard's PCI BIOS provides a standard way of enumerating the PCI host adapters, which is used by the Linux kernel. Some PCI BIOS implementations enumerate the PCI slots in order of increasing bus number and device number, while others do so in the opposite direction.

Unfortunately, Microsoft decided that Windows 95 would always enumerate the PCI slots in order of increasing bus number and device number regardless of the PCI

BIOS enumeration, and requires that their scheme be supported by the host adapter's BIOS to receive Windows 95 certification. Therefore, the factory default settings of the BT-948/958/958D enumerate the host adapters by increasing bus number and device number. To disable this feature, invoke the AutoSCSI utility via Ctrl-B at system startup and select ”Adapter Configuration”, ”View/Modify Configuration”, press Ctrl-F10, and then change the ”Use Bus And Device # For PCI Scanning Seq.” option to OFF.

This driver will interrogate the setting of the PCI Scanning Sequence option so as to recognize the host adapters in the same order as they are enumerated by the host adapter's BIOS.

Enabling UltraSCSI Transfers

The BT-948/958/958D ship with their configuration set to ”Factory Default” settings that are conservative and do not allow for UltraSCSI speed to be negotiated. This results in fewer problems when these host adapters are installed in systems with cabling or termination that is not sufficient for UltraSCSI operation, or where existing SCSI devices do not properly respond to synchronous transfer negotiation for UltraSCSI speed. AutoSCSI may be used to load ”Optimum Performance” settings which allow UltraSCSI speed to be negotiated with all devices, or UltraSCSI speed can be enabled on an individual basis. It is recommended that SCAM be manually disabled after the ”Optimum Performance” settings are loaded.

4.10.6 Driver Options

BusLogic Driver Options may be specified either via the Linux Kernel Command Line or via the Loadable Kernel Module Installation Facility. Driver Options for multiple host adapters may be specified either by separating the option strings by a semicolon, or by specifying multiple ”BusLogic=” strings on the command line. Individual option specifications for a single host adapter are separated by commas. The Probing and Debugging Options apply to all host adapters whereas the remaining options apply individually only to the selected host adapter.

The BusLogic Driver Probing Options comprise the following:

NoProbe

The ”NoProbe” option disables all probing and therefore no BusLogic Host Adapters will be detected.

NoProbePCI

The ”NoProbePCI” options disables the interrogation of PCI Configuration Space and therefore only ISA Multimaster Host Adapters will be detected, as well as PCI Multimaster Host Adapters that have their ISA Compatible I/O Port set to ”Primary” or ”Alternate”.

NoSortPCI

The ”NoSortPCI” option forces PCI MultiMaster Host Adapters to be enumerated in the order provided by the PCI BIOS, ignoring any setting of the AutoSCSI ”Use Bus And Device # For PCI Scanning Seq.” option.

MultiMasterFirst

The ”MultiMasterFirst” option forces MultiMaster Host Adapters to be probed before FlashPoint Host Adapters. By default, if both FlashPoint and PCI MultiMaster Host Adapters are present, this driver will probe for FlashPoint Host Adapters first unless the BIOS primary disk is controlled by the first PCI MultiMaster Host Adapter, in which case MultiMaster Host Adapters will be probed first.

FlashPointFirst

The ”FlashPointFirst” option forces FlashPoint Host Adapters to be probed before MultiMaster Host Adapters.

The BusLogic Driver Tagged Queuing Options allow for explicitly specifying the Queue Depth and whether Tagged Queuing is permitted for each Target Device (assuming that the Target Device supports Tagged Queuing). The Queue Depth is the number of SCSI Commands that are allowed to be concurrently presented for execution (either to the Host Adapter or Target Device). Note that explicitly enabling Tagged Queuing may lead to problems; the option to enable or disable Tagged Queuing is provided primarily to allow disabling Tagged Queuing on Target Devices that do not implement it correctly. The following options are available:

QueueDepth:<integer>

The ”QueueDepth:” or QD:” option specifies the Queue Depth to use for all Target Devices that support Tagged Queuing, as well as the maximum Queue Depth for devices that do not support Tagged Queuing. If no Queue Depth option is provided, the Queue Depth will be determined automatically based on the Host Adapter's Total Queue Depth and the number, type, speed, and capabilities of the detected Target Devices. Target Devices that do not support Tagged Queuing always have their Queue Depth set to BusLogic_UntaggedQueueDepth or BusLogic_UntaggedQueueDepthBB, unless a lower Queue Depth option is provided. A Queue Depth of 1 automatically disables Tagged Queuing.

QueueDepth:[<integer>,<integer>...]

The ”QueueDepth:[...]” or ”QD:[...]” option specifies the Queue Depth individually for each Target Device. If an <integer> is omitted, the associated Target Device will have its Queue Depth selected automatically.

TaggedQueuing:Default

The ”TaggedQueuing:Default” or ”TQ:Default” option permits Tagged Queuing based on the firmware version of the BusLogic Host Adapter and based on whether the Queue Depth allows queuing multiple commands.

TaggedQueuing:Enable

The ”TaggedQueuing:Enable” or ”TQ:Enable” option enables Tagged Queuing for all Target Devices on this Host Adapter, overriding any limitation that would otherwise be imposed based on the Host Adapter firmware version.

TaggedQueuing:Disable

The ”TaggedQueuing:Disable” or ”TQ:Disable” option disables Tagged Queuing for all Target Devices on this Host Adapter.

TaggedQueuing:<Target-Spec>

The ”TaggedQueuing:<Target-Spec>” or ”TQ:<Target-Spec>” option controls Tagged Queuing individually for each Target Device. <Target-Spec> is a sequence of ”Y”, ”N”, and ”X” characters. ”Y” enables Tagged Queuing, ”N” disables Tagged

Queuing, and ”X” accepts the default based on the firmware version. The first character refers to Target Device 0, the second to Target Device 1, and so on; if the sequence of ”Y”, ”N”, and ”X” characters does not cover all the Target Devices, unspecified characters are assumed to be ”X”.

The BusLogic Driver Miscellaneous Options comprise the following:

BusSettleTime:<seconds>

The ”BusSettleTime:” or ”BST:” option specifies the Bus Settle Time in seconds. The Bus Settle Time is the amount of time to wait between a Host Adapter Hard Reset which initiates a SCSI Bus Reset and issuing any SCSI Commands. If unspecified, it defaults to BusLogic_DefaultBusSettleTime.

InhibitTargetInquiry

The ”InhibitTargetInquiry” option inhibits the execution of an Inquire Target Devices or Inquire Installed Devices command on MultiMaster Host Adapters. This may be necessary with some older Target Devices that do not respond correctly when Logical Units above 0 are addressed.

The BusLogic Driver Debugging Options comprise the following:

TraceProbe

The ”TraceProbe” option enables tracing of Host Adapter Probing.

TraceHardwareReset

The ”TraceHardwareReset” option enables tracing of Host Adapter Hardware Reset.

TraceConfiguration

The ”TraceConfiguration” option enables tracing of Host Adapter Configuration.

TraceErrors

The ”TraceErrors” option enables tracing of SCSI Commands that return an error from the Target Device. The CDB and Sense Data will be printed for each SCSI Command that fails.

Debug

The ”Debug” option enables all debugging options.

The following examples demonstrate setting the Queue Depth for Target Devices 1 and 2 on the first host adapter to 7 and 15, the Queue Depth for all Target Devices on the second host adapter to 31, and the Bus Settle Time on the second host adapter to 30 seconds.

Linux Kernel Command Line:

```
linux BusLogic=QueueDepth:[,7,15];QueueDepth:31,BusSettleTime:30
```

LILO Linux Boot Loader (in /etc/lilo.conf):

```
append = "BusLogic=QueueDepth:[,7,15];QueueDepth:31,BusSettleTime:30"
```

INSMOD Loadable Kernel Module Installation Facility:

```
insmod BusLogic.o \
    'BusLogic="QueueDepth:[,7,15];QueueDepth:31,BusSettleTime:30"'
```

Note: Module Utilities 2.1.71 or later is required for correct parsing of driver options containing commas.

4.10.7 Driver Installation

This distribution was prepared for Linux kernel version 2.0.35, but should be compatible with 2.0.4 or any later 2.0 series kernel.

To install the new BusLogic SCSI driver, you may use the following commands, replacing "/usr/src" with wherever you keep your Linux kernel source tree:

```
cd /usr/src
tar -xvzf BusLogic-2.0.15.tar.gz
mv README.* LICENSE.* BusLogic.[ch] FlashPoint.c linux/drivers/scsi
patch -p0 < BusLogic.patch (only for 2.0.33 and below)
cd linux
make config
make zImage
```

Then install "arch/x86/boot/zImage" as your standard kernel, run lilo if appropriate, and reboot.

4.10.8 BusLogic Announcements Mailing List

The BusLogic Announcements Mailing List provides a forum for informing Linux users of new driver releases and other announcements regarding Linux support for BusLogic SCSI Host Adapters. To join the mailing list, send a message to "buslogic-announce-request@dandelion.com" with the line "subscribe" in the message body.

4.11 Chelsio S3 iSCSI Driver for Linux

4.11.1 Introduction

The Chelsio T3 ASIC based Adapters (S310, S320, S302, S304, Mezz cards, etc. series of products) support iSCSI acceleration and iSCSI Direct Data Placement (DDP) where the hardware handles the expensive byte touching operations, such as CRC computation and verification, and direct DMA to the final host memory destination:

- iSCSI PDU digest generation and verification

  On transmitting, Chelsio S3 h/w computes and inserts the Header and Data digest into the PDUs. On receiving, Chelsio S3 h/w computes and verifies the Header and Data digest of the PDUs.

- Direct Data Placement (DDP)

  S3 h/w can directly place the iSCSI Data-In or Data-Out PDU's payload into pre-posted final destination host-memory buffers based on the Initiator Task Tag (ITT) in Data-In or Target Task Tag (TTT) in Data-Out PDUs.

- PDU Transmit and Recovery

  On transmitting, S3 h/w accepts the complete PDU (header + data) from the host driver, computes and inserts the digests, decomposes the PDU into multiple TCP segments if necessary, and transmit all the TCP segments onto the wire. It handles TCP retransmission if needed.

  On receiving, S3 h/w recovers the iSCSI PDU by reassembling TCP segments, separating the header and data, calculating and verifying the digests, then forwarding the header to the host. The payload data, if possible, will be directly placed into the pre-posted host DDP buffer. Otherwise, the payload data will be sent to the host too.

The cxgb3i driver interfaces with open-iscsi initiator and provides the iSCSI acceleration through Chelsio hardware wherever applicable.

4.11.2 Using the cxgb3i Driver

The following steps need to be taken to accelerates the open-iscsi initiator:

1. Load the cxgb3i driver: ”modprobe cxgb3i”

   The cxgb3i module registers a new transport class ”cxgb3i” with open-iscsi.

   - in the case of recompiling the kernel, the cxgb3i selection is located at:

   ```
   Device Drivers
           SCSI device support --->
                   [*] SCSI low-level drivers  --->
                           <M>   Chelsio S3xx iSCSI support
   ```

2. Create an interface file located under /etc/iscsi/ifaces/ for the new transport class ”cxgb3i”.

   The content of the file should be in the following format:

   ```
   iface.transport_name = cxgb3i
   iface.net_ifacename = <ethX>
   iface.ipaddress = <iscsi ip address>
   ```

   - if iface.ipaddress is specified, <iscsi ip address> needs to be either the same as the ethX's ip address or an address on the same subnet. Make sure the ip address is unique in the network.

3. edit /etc/iscsi/iscsid.conf The default setting for MaxRecvDataSegmentLength (131072) is too big; replace with a value no bigger than 15360 (for example 8192):

   ```
   node.conn[0].iscsi.MaxRecvDataSegmentLength = 8192
   ```

   - The login would fail for a normal session if MaxRecvDataSegmentLength is too big. A error message in the format of ”cxgb3i: ERR! MaxRecvSegmentLength <X> too big. Need to be <= <Y>.” would be logged to dmesg.

4. To direct open-iscsi traffic to go through cxgb3i's accelerated path, ”-I <iface file name>” option needs to be specified with most of the iscsiadm command. <iface file name> is the transport interface file created in step 2.

4.12 dc395x SCSI driver

4.12.1 Status

The driver has been tested with CD-R and CD-R/W drives. These should be safe to use. Testing with hard disks has not been done to any great degree and caution should be exercised if you want to attempt to use this driver with hard disks.

This driver is evolved from the original 2.4 driver. Problems, questions and patches should be submitted to the Linux SCSI mailing list.

4.12.2 Parameters

The driver uses the settings from the EEPROM set in the SCSI BIOS setup. If there is no EEPROM, the driver uses default values. Both can be overridden by command line parameters (module or kernel parameters).

The following parameters are available:

safe
Default: 0, Acceptable values: 0 or 1

If safe is set to 1 then the adapter will use conservative (”safe”) default settings. This sets:

shortcut for dc395x=7,4,9,15,2,10

adapter_id
Default: 7, Acceptable values: 0 to 15

Sets the host adapter SCSI ID.

max_speed
Default: 1, Acceptable value: 0 to 7

0	20 Mhz
1	12.2 Mhz
2	10 Mhz
3	8 Mhz
4	6.7 Mhz
5	5.8 Hhz
6	5 Mhz
7	4 Mhz

dev_mode
Bitmap for device configuration

DevMode bit definition:

Bit	Val(hex)	Val(dec)	Meaning
0	0x01	1	Parity check
1	0x02	2	Synchronous Negotiation
2	0x04	4	Disconnection
3	0x08	8	Send Start command on startup. (Not used)
4	0x10	16	Tagged Command Queueing
5	0x20	32	Wide Negotiation

adapter_mode
Bitmap for adapter configuration

AdaptMode bit definition

Bit	Val(hex)	Val(dec)	Meaning
0	0x01	1	Support more than two drives. (Not used)
1	0x02	2	Use DOS compatible mapping for HDs greater than 1GB.
2	0x04	4	Reset SCSI Bus on startup.
3	0x08	8	Active Negation: Improves SCSI Bus noise immunity.
4	0x10	16	Immediate return on BIOS seek command. (Not used)
(*)5	0x20	32	Check for LUNs >= 1.

tags
Default: 3, Acceptable values: 0-5

The number of tags is 1<<x, if x has been specified

reset_delay
Default: 1, Acceptable values: 0-180

The seconds to not accept commands after a SCSI Reset

For the built in driver the parameters should be prefixed with dc395x. (eg ”dc395x.safe=1”)

4.12.3 Copyright

The driver is free software. It is protected by the GNU General Public License (GPL). Please read it, before using this driver. It should be included in your kernel sources and with your distribution. It carries the filename COPYING. If you don't have it, please ask me to send you one by email.

Note: The GNU GPL says also something about warranty and liability. Please be aware the following: While we do my best to provide a working and reliable driver, there is a chance, that it will kill your valuable data. We refuse to take any responsibility for that. The driver is provided as-is and YOU USE IT AT YOUR OWN RESPONSIBILITY.

4.13 Adaptec dpti driver

Redistribution and use in source form, with or without modification, are permitted provided that redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

This software is provided `as is` by Adaptec and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are disclaimed. In no event shall Adaptec be liable for any direct, indirect, incidental, special, exemplary or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruptions) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this driver software, even if advised of the possibility of such damage.

This driver supports the Adaptec I2O RAID and DPT SmartRAID V I2O boards.

4.13.1 Credits

The original linux driver was ported to Linux by Karen White while at Dell Computer. It was ported from Bob Pasteur's (of DPT) original non-Linux driver. Mark Salyzyn and Bob Pasteur consulted on the original driver.

2.0 version of the driver by Deanna Bonds and Mark Salyzyn.

4.13.2 History

The driver was originally ported to linux version 2.0.34

V2.0	Rewrite of driver. Re-architectured based on i2o subsystem. This was the first full GPL version since the last version used i2osig headers which were not GPL. Developer Testing version.
V2.1	Internal testing
V2.2	First released version
V2.3	Changes: • Added Raptor Support • Fixed bug causing system to hang under extreme load with • management utilities running (removed GFP_DMA from kmalloc flags)
V2.4	First version ready to be submitted to be embedded in the kernel Changes: • Implemented suggestions from Alan Cox • Added calculation of resid for sg layer • Better error handling • Added checking underflow conditions • Added DATAPROTECT checking • Changed error return codes • Fixed pointer bug in bus reset routine • Enabled hba reset from ioctls (allows a FW flash to reboot and use the new FW without having to reboot) • Changed proc output

4.13.3 TODO

- Add 64 bit Scatter Gather when compiled on 64 bit architectures
- Add sparse lun scanning
- Add code that checks if a device that had been taken offline is now online (at the FW level) when test unit ready or inquiry command from scsi-core
- Add proc read interface
- busrescan command
- rescan command
- Add code to rescan routine that notifies scsi-core about new devices
- Add support for C-PCI (hotplug stuff)
- Add ioctl passthru error recovery

4.13.4 Notes

The DPT card optimizes the order of processing commands. Consequently, a command may take up to 6 minutes to complete after it has been sent to the board.

The files dpti_ioctl.h dptsig.h osd_defs.h osd_util.h sys_info.h are part of the interface files for Adaptec's management routines. These define the structures used in the ioctls. They are written to be portable. They are hard to read, but I need to use them 'as is' or I can miss changes in the interface.

4.14 The BusLogic FlashPoint SCSI Driver

The BusLogic FlashPoint SCSI Host Adapters are now fully supported on Linux. The upgrade program described below has been officially terminated effective 31 March 1997 since it is no longer needed.

```
        MYLEX INTRODUCES LINUX OPERATING SYSTEM SUPPORT FOR ITS
            BUSLOGIC FLASHPOINT LINE OF SCSI HOST ADAPTERS


FREMONT, CA, -- October 8, 1996 -- Mylex Corporation has expanded Linux
operating system support to its BusLogic brand of FlashPoint Ultra SCSI
host adapters.  All of BusLogic's other SCSI host adapters, including the
MultiMaster line, currently support the Linux operating system.  Linux
drivers and information will be available on October 15th at
http://sourceforge.net/projects/dandelion/.

"Mylex is committed to supporting the Linux community," says Peter Shambora,
vice president of marketing for Mylex.  "We have supported Linux driver
development and provided technical support for our host adapters for several
years, and are pleased to now make our FlashPoint products available to this
user base."
```

4.14.1 The Linux Operating System

Linux is a freely-distributed implementation of UNIX for Intel x86, Sun SPARC, SGI MIPS, Motorola 68k, Digital Alpha AXP and Motorola PowerPC machines. It supports a wide range of software, including the X Window System, Emacs, and TCP/IP networking. Further information is available at http://www.linux.org and http://www.ssc.com/.

4.14.2 FlashPoint Host Adapters

The FlashPoint family of Ultra SCSI host adapters, designed for workstation and file server environments, are available in narrow, wide, dual channel, and dual channel wide versions. These adapters feature SeqEngine automation technology, which minimizes SCSI command overhead and reduces the number of interrupts generated to the CPU.

4.14.3 About Mylex

Mylex Corporation (NASDAQ/NM SYMBOL: MYLX), founded in 1983, is a leading producer of RAID technology and network management products. The company produces high performance disk array (RAID) controllers, and complementary computer products for network servers, mass storage systems, workstations and system boards. Through its wide range of RAID controllers and its BusLogic line of Ultra SCSI host adapter products, Mylex provides enabling intelligent I/O technologies that increase network management control, enhance CPU utilization, optimize I/O performance, and ensure data security and availability. Products are sold globally through a network of OEMs, major distributors, VARs, and system integrators. Mylex Corporation is headquartered at 34551 Ardenwood Blvd., Fremont, CA.

4.14.4 Contact:

```
Peter Shambora
Vice President of Marketing
Mylex Corp.
510/796-6100
peters@mylex.com
```

```
                            ANNOUNCEMENT
              BusLogic FlashPoint LT/BT-948 Upgrade Program
                           1 February 1996

                       ADDITIONAL ANNOUNCEMENT
              BusLogic FlashPoint LW/BT-958 Upgrade Program
                           14 June 1996

Ever since its introduction last October, the BusLogic FlashPoint LT has
been problematic for members of the Linux community, in that no Linux
drivers have been available for this new Ultra SCSI product.  Despite its
officially being positioned as a desktop workstation product, and not being
particularly well suited for a high performance multitasking operating
system like Linux, the FlashPoint LT has been touted by computer system
vendors as the latest thing, and has been sold even on many of their high
end systems, to the exclusion of the older MultiMaster products.  This has
caused grief for many people who inadvertently purchased a system expecting
that all BusLogic SCSI Host Adapters were supported by Linux, only to
discover that the FlashPoint was not supported and would not be for quite
some time, if ever.

After this problem was identified, BusLogic contacted its major OEM
```

customers to make sure the BT-946C/956C MultiMaster cards would still be made available, and that Linux users who mistakenly ordered systems with the FlashPoint would be able to upgrade to the BT-946C. While this helped many purchasers of new systems, it was only a partial solution to the overall problem of FlashPoint support for Linux users. It did nothing to assist the people who initially purchased a FlashPoint for a supported operating system and then later decided to run Linux, or those who had ended up with a FlashPoint LT, believing it was supported, and were unable to return it.

In the middle of December, I asked to meet with BusLogic's senior management to discuss the issues related to Linux and free software support for the FlashPoint. Rumors of varying accuracy had been circulating publicly about BusLogic's attitude toward the Linux community, and I felt it was best that these issues be addressed directly. I sent an email message after 11pm one evening, and the meeting took place the next afternoon. Unfortunately, corporate wheels sometimes grind slowly, especially when a company is being acquired, and so it's taken until now before the details were completely determined and a public statement could be made.

BusLogic is not prepared at this time to release the information necessary for third parties to write drivers for the FlashPoint. The only existing FlashPoint drivers have been written directly by BusLogic Engineering, and there is no FlashPoint documentation sufficiently detailed to allow outside developers to write a driver without substantial assistance. While there are people at BusLogic who would rather not release the details of the FlashPoint architecture at all, that debate has not yet been settled either way. In any event, even if documentation were available today it would take quite a while for a usable driver to be written, especially since I'm not convinced that the effort required would be worthwhile.

However, BusLogic does remain committed to providing a high performance SCSI solution for the Linux community, and does not want to see anyone left unable to run Linux because they have a Flashpoint LT. Therefore, BusLogic has put in place a direct upgrade program to allow any Linux user worldwide to trade in their FlashPoint LT for the new BT-948 MultiMaster PCI Ultra SCSI Host Adapter. The BT-948 is the Ultra SCSI successor to the BT-946C and has all the best features of both the BT-946C and FlashPoint LT, including smart termination and a flash PROM for easy firmware updates, and is of course compatible with the present Linux driver. The price for this upgrade has been set at US $45 plus shipping and handling, and the upgrade program will be administered through BusLogic Technical Support, which can be reached by electronic mail at techsup@buslogic.com, by Voice at +1 408 654-0760, or by FAX at +1 408 492-1542.

As of 14 June 1996, the original BusLogic FlashPoint LT to BT-948 upgrade program has now been extended to encompass the FlashPoint LW Wide Ultra SCSI Host Adapter. Any Linux user worldwide may trade in their FlashPoint LW (BT-950) for a BT-958 MultiMaster PCI Ultra SCSI Host Adapter. The

```
price for this upgrade has been set at US $65 plus shipping and handling.

I was a beta test site for the BT-948/958, and versions 1.2.1 and 1.3.1 of
my BusLogic driver already included latent support for the BT-948/958.
Additional cosmetic support for the Ultra SCSI MultiMaster cards was added
subsequent releases.  As a result of this cooperative testing process,
several firmware bugs were found and corrected.  My heavily loaded Linux
test system provided an ideal environment for testing error recovery
processes that are much more rarely exercised in production systems, but
are crucial to overall system stability.  It was especially convenient
being able to work directly with their firmware engineer in demonstrating
the problems under control of the firmware debugging environment; things
sure have come a long way since the last time I worked on firmware for an
embedded system.  I am presently working on some performance testing and
expect to have some data to report in the not too distant future.

BusLogic asked me to send this announcement since a large percentage of the
questions regarding support for the FlashPoint have either been sent to me
directly via email, or have appeared in the Linux newsgroups in which I
participate.  To summarize, BusLogic is offering Linux users an upgrade
from the unsupported FlashPoint LT (BT-930) to the supported BT-948 for US
$45 plus shipping and handling, or from the unsupported FlashPoint LW
(BT-950) to the supported BT-958 for $65 plus shipping and handling.
Contact BusLogic Technical Support at techsup@buslogic.com or +1 408
654-0760 to take advantage of their offer.

             Leonard N. Zubkoff
             lnz@dandelion.com
```

4.15 g_NCR5380 driver

Copyright © 1993 Drew Eckhard

NCR53c400 extensions Copyright © 1994,1995,1996 Kevin Lentin

This file documents the NCR53c400 extensions by Kevin Lentin and some enhancements to the NCR5380 core.

This driver supports NCR5380 and NCR53c400 and compatible cards in port or memory mapped modes.

Use of an interrupt is recommended, if supported by the board, as this will allow targets to disconnect and thereby improve SCSI bus utilization.

If the irq parameter is 254 or is omitted entirely, the driver will probe for the correct IRQ line automatically. If the irq parameter is 0 or 255 then no IRQ will be used.

The NCR53c400 does not support DMA but it does have Pseudo-DMA which is supported by the driver.

This driver provides some information on what it has detected in /proc/scsi/g_NCR5380/x where x is the scsi card number as detected at boot time. More info to come in the future.

This driver works as a module. When included as a module, parameters can be passed on the insmod/modprobe command line:

irq=xx[,...]	the interrupt(s)
base=xx[,...]	the port or base address(es) (for port or memory mapped, resp.)
card=xx[,...]	card type(s): 0 NCR5380, 1 NCR53C400, 2 NCR53C400A, 3 Domex Technology Corp 3181E (DTC3181E) 4 Hewlett Packard C2502

These old-style parameters can support only one card:

ncr_irq=xx	the interrupt
ncr_addr=xx	the port or base address (for port or memory mapped, resp.)
ncr_5380=1	to set up for a NCR5380 board
ncr_53c400=1	to set up for a NCR53C400 board
ncr_53c400a=1	to set up for a NCR53C400A board
dtc_3181e=1	to set up for a Domex Technology Corp 3181E board
hp_c2502=1	to set up for a Hewlett Packard C2502 board

E.g. Trantor T130B in its default configuration:

```
modprobe g_NCR5380 irq=5 base=0x350 card=1
```

or alternatively, using the old syntax:

```
modprobe g_NCR5380 ncr_irq=5 ncr_addr=0x350 ncr_53c400=1
```

E.g. a port mapped NCR5380 board, driver to probe for IRQ:

```
modprobe g_NCR5380 base=0x350 card=0
```

or alternatively:

```
modprobe g_NCR5380 ncr_addr=0x350 ncr_5380=1
```

E.g. a memory mapped NCR53C400 board with no IRQ:

```
modprobe g_NCR5380 irq=255 base=0xc8000 card=1
```

or alternatively:

```
modprobe g_NCR5380 ncr_irq=255 ncr_addr=0xc8000 ncr_53c400=1
```

E.g. two cards, DTC3181 (in non-PnP mode) at 0x240 with no IRQ and HP C2502 at 0x300 with IRQ 7:

```
modprobe g_NCR5380 irq=0,7 base=0x240,0x300 card=3,4
```

Kevin Lentin K.Lentin@cs.monash.edu.au

4.16 HPSA - Hewlett Packard Smart Array driver

This file describes the hpsa SCSI driver for HP Smart Array controllers. The hpsa driver is intended to supplant the cciss driver for newer Smart Array controllers. The hpsa driver is a SCSI driver, while the cciss driver is a "block" driver. Actually cciss is both a block driver (for logical drives) AND a SCSI driver (for tape drives). This "split-brained" design of the cciss driver is a source of excess complexity and eliminating that complexity is one of the reasons for hpsa to exist.

4.16.1 Supported devices

- Smart Array P212
- Smart Array P410
- Smart Array P410i
- Smart Array P411
- Smart Array P812
- Smart Array P712m
- Smart Array P711m
- StorageWorks P1210m

Additionally, older Smart Arrays may work with the hpsa driver if the kernel boot parameter "hpsa_allow_any=1" is specified, however these are not tested nor supported by HP with this driver. For older Smart Arrays, the cciss driver should still be used.

The "hpsa_simple_mode=1" boot parameter may be used to prevent the driver from putting the controller into "performant" mode. The difference is that with simple mode, each command completion requires an interrupt, while with "performant mode" (the default, and ordinarily better performing) it is possible to have multiple command completions indicated by a single interrupt.

4.16.2 HPSA specific entries in /sys

In addition to the generic SCSI attributes available in /sys, hpsa supports the following attributes:

4.16.3 HPSA specific host attributes

```
/sys/class/scsi_host/host*/rescan
/sys/class/scsi_host/host*/firmware_revision
/sys/class/scsi_host/host*/resettable
/sys/class/scsi_host/host*/transport_mode
```

the host ”rescan” attribute is a write only attribute. Writing to this attribute will cause the driver to scan for new, changed, or removed devices (e.g. hot-plugged tape drives, or newly configured or deleted logical drives, etc.) and notify the SCSI midlayer of any changes detected. Normally this is triggered automatically by HP's Array Configuration Utility (either the GUI or command line variety) so for logical drive changes, the user should not normally have to use this. It may be useful when hot plugging devices like tape drives, or entire storage boxes containing pre-configured logical drives.

The ”firmware_revision” attribute contains the firmware version of the Smart Array. For example:

```
root@host:/sys/class/scsi_host/host4# cat firmware_revision
7.14
```

The transport_mode indicates whether the controller is in ”performant” or ”simple” mode. This is controlled by the ”hpsa_simple_mode” module parameter.

The ”resettable” read-only attribute indicates whether a particular controller is able to honor the ”reset_devices” kernel parameter. If the device is resettable, this file will contain a ”1”, otherwise, a ”0”. This parameter is used by kdump, for example, to reset the controller at driver load time to eliminate any outstanding commands on the controller and get the controller into a known state so that the kdump initiated i/o will work right and not be disrupted in any way by stale commands or other stale state remaining on the controller from the previous kernel. This attribute enables kexec tools to warn the user if they attempt to designate a device which is unable to honor the reset_devices kernel parameter as a dump device.

HPSA specific disk attributes

```
/sys/class/scsi_disk/c:b:t:l/device/unique_id
/sys/class/scsi_disk/c:b:t:l/device/raid_level
/sys/class/scsi_disk/c:b:t:l/device/lunid
```

(where c:b:t:l are the controller, bus, target and lun of the device)

For example:

```
root@host:/sys/class/scsi_disk/4:0:0:0/device# cat unique_id
600508B1001044395355323037570F77
root@host:/sys/class/scsi_disk/4:0:0:0/device# cat lunid
0x0000004000000000
root@host:/sys/class/scsi_disk/4:0:0:0/device# cat raid_level
RAID 0
```

4.16.4 HPSA specific ioctls

For compatibility with applications written for the cciss driver, many, but not all of the ioctls supported by the cciss driver are also supported by the hpsa driver. The data structures used by these are described in include/linux/cciss_ioctl.h

CCISS_DEREGDISK, CCISS_REGNEWDISK, CCISS_REGNEWD
The above three ioctls all do exactly the same thing, which is to cause the driver to rescan for new devices. This does exactly the same thing as writing to the hpsa specific host "rescan" attribute.

CCISS_GETPCIINFO
Returns PCI domain, bus, device and function and "board ID" (PCI subsystem ID).

CCISS_GETDRIVVER
Returns driver version in three bytes encoded as:

```
(major_version << 16) | (minor_version << 8) | (subminor_version)
```

CCISS_PASSTHRU, CCISS_BIG_PASSTHRU
Allows "BMIC" and "CISS" commands to be passed through to the Smart Array. These are used extensively by the HP Array Configuration Utility, SNMP storage agents, etc. See cciss_vol_status at http://cciss.sf.net for some examples.

4.17 Highpoint RocketRAID 3xxx/4xxx Adapter Driver (hptiop)

4.17.1 Controller Register Map

For RR44xx Intel IOP based adapters, the controller IOP is accessed via PCI BAR0 and BAR2

BAR0 offset	Register
0x11C5C	Link Interface IRQ Set
0x11C60	Link Interface IRQ Clear

BAR2 offset	Register
0x10	Inbound Message Register 0
0x14	Inbound Message Register 1
0x18	Outbound Message Register 0
0x1C	Outbound Message Register 1
0x20	Inbound Doorbell Register
0x24	Inbound Interrupt Status Register
0x28	Inbound Interrupt Mask Register
0x30	Outbound Interrupt Status Register
0x34	Outbound Interrupt Mask Register
0x40	Inbound Queue Port
0x44	Outbound Queue Port

For Intel IOP based adapters, the controller IOP is accessed via PCI BAR0:

BAR0 offset	Register
0x10	Inbound Message Register 0
0x14	Inbound Message Register 1
0x18	Outbound Message Register 0
0x1C	Outbound Message Register 1
0x20	Inbound Doorbell Register
0x24	Inbound Interrupt Status Register
0x28	Inbound Interrupt Mask Register
0x30	Outbound Interrupt Status Register
0x34	Outbound Interrupt Mask Register
0x40	Inbound Queue Port
0x44	Outbound Queue Port

For Marvell not Frey IOP based adapters, the IOP is accessed via PCI BAR0 and BAR1:

BAR0 offset	Register
0x20400	Inbound Doorbell Register
0x20404	Inbound Interrupt Mask Register
0x20408	Outbound Doorbell Register
0x2040C	Outbound Interrupt Mask Register

BAR1 offset	Register
0x0	Inbound Queue Head Pointer
0x4	Inbound Queue Tail Pointer
0x8	Outbound Queue Head Pointer
0xC	Outbound Queue Tail Pointer
0x10	Inbound Message Register
0x14	Outbound Message Register
0x40-0x1040	Inbound Queue
0x1040-0x2040	Outbound Queue

For Marvell Frey IOP based adapters, the IOP is accessed via PCI BAR0 and BAR1:

BAR0 offset	Register
0x0	IOP configuration information.

BAR1 offset	Register
0x4000	Inbound List Base Address Low
0x4004	Inbound List Base Address High
0x4018	Inbound List Write Pointer
0x402C	Inbound List Configuration and Control
0x4050	Outbound List Base Address Low
0x4054	Outbound List Base Address High
0x4058	Outbound List Copy Pointer Shadow Base Address Low
0x405C	Outbound List Copy Pointer Shadow Base Address High
0x4088	Outbound List Interrupt Cause
0x408C	Outbound List Interrupt Enable
0x1020C	PCIe Function 0 Interrupt Enable
0x10400	PCIe Function 0 to CPU Message A
0x10420	CPU to PCIe Function 0 Message A
0x10480	CPU to PCIe Function 0 Doorbell
0x10484	CPU to PCIe Function 0 Doorbell Enable

4.17.2 I/O Request Workflow of Not Marvell Frey

All queued requests are handled via inbound/outbound queue port. A request packet can be allocated in either IOP or host memory.

To send a request to the controller:

- Get a free request packet by reading the inbound queue port or allocate a free request in host DMA coherent memory.

  The value returned from the inbound queue port is an offset relative to the IOP BAR0.

  Requests allocated in host memory must be aligned on 32-bytes boundary.

- Fill the packet.

- Post the packet to IOP by writing it to inbound queue. For requests allocated in IOP memory, write the offset to inbound queue port. For requests allocated in host memory, write (0x80000000|(bus_addr>>5)) to the inbound queue port.

- The IOP process the request. When the request is completed, it will be put into outbound queue. An outbound interrupt will be generated.

  For requests allocated in IOP memory, the request offset is posted to outbound queue.

  For requests allocated in host memory, (0x80000000|(bus_addr>>5)) is posted to the outbound queue. If IOP_REQUEST_FLAG_OUTPUT_CONTEXT flag is set in the request, the low 32-bit context value will be posted instead.

- The host read the outbound queue and complete the request.

  For requests allocated in IOP memory, the host driver free the request by writing it to the outbound queue.

Non-queued requests (reset/flush etc) can be sent via inbound message register 0. An outbound message with the same value indicates the completion of an inbound message.

4.17.3 I/O Request Workflow of Marvell Frey

All queued requests are handled via inbound/outbound list.

To send a request to the controller:

- Allocate a free request in host DMA coherent memory.

  Requests allocated in host memory must be aligned on 32-bytes boundary.

- Fill the request with index of the request in the flag.

  Fill a free inbound list unit with the physical address and the size of the request.

  Set up the inbound list write pointer with the index of previous unit, round to 0 if the index reaches the supported count of requests.

- Post the inbound list writer pointer to IOP.

- The IOP process the request. When the request is completed, the flag of the request with or-ed IOPMU_QUEUE_MASK_HOST_BITS will be put into a free outbound list unit and the index of the outbound list unit will be put into the copy pointer shadow register. An outbound interrupt will be generated.

- The host read the outbound list copy pointer shadow register and compare with previous saved read pointer N. If they are different, the host will read the (N+1)th outbound list unit.

  The host get the index of the request from the (N+1)th outbound list unit and complete the request.

Non-queued requests (reset communication/reset/flush etc) can be sent via PCIe Function 0 to CPU Message A register. The CPU to PCIe Function 0 Message register with the same value indicates the completion of message.

4.17.4 User-level Interface

The driver exposes following sysfs attributes:

NAME	R/W	Description
driver-version	R	driver version string
firmware-version	R	firmware version string

Copyright © 2006-2012 HighPoint Technologies, Inc. All Rights Reserved.

This file 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.

linux@highpoint-tech.com

http://www.highpoint-tech.com

4.18 SAS Layer

The SAS Layer is a management infrastructure which manages SAS LLDDs. It sits between SCSI Core and SAS LLDDs. The layout is as follows: while SCSI Core is concerned with SAM/SPC issues, and a SAS LLDD+sequencer is concerned with phy/OOB/link management, the SAS layer is concerned with:

- SAS Phy/Port/HA event management (LLDD generates, SAS Layer processes),
- SAS Port management (creation/destruction),
- SAS Domain discovery and revalidation,
- SAS Domain device management,
- SCSI Host registration/unregistration,
- Device registration with SCSI Core (SAS) or libata (SATA), and
- Expander management and exporting expander control to user space.

A SAS LLDD is a PCI device driver. It is concerned with phy/OOB management, and vendor specific tasks and generates events to the SAS layer.

The SAS Layer does most SAS tasks as outlined in the SAS 1.1 spec.

The sas_ha_struct describes the SAS LLDD to the SAS layer. Most of it is used by the SAS Layer but a few fields need to be initialized by the LLDDs.

After initializing your hardware, from the probe() function you call sas_register_ha(). It will register your LLDD with the SCSI subsystem, creating a SCSI host and it will register your SAS driver with the sysfs SAS tree it creates. It will then return. Then you enable your phys to actually start OOB (at which point your driver will start calling the notify_* event callbacks).

4.18.1 Structure descriptions

struct sas_phy

Normally this is statically embedded to your driver's phy structure:

```
struct my_phy {
        blah;
        struct sas_phy sas_phy;
        bleh;
};
```

And then all the phys are an array of my_phy in your HA struct (shown below).

Then as you go along and initialize your phys you also initialize the sas_phy struct, along with your own phy structure.

In general, the phys are managed by the LLDD and the ports are managed by the SAS layer. So the phys are initialized and updated by the LLDD and the ports are initialized and updated by the SAS layer.

There is a scheme where the LLDD can RW certain fields, and the SAS layer can only read such ones, and vice versa. The idea is to avoid unnecessary locking.

enabled

- must be set (0/1)

id

- must be set [0,MAX_PHYS)]

class, proto, type, role, oob_mode, linkrate

- must be set

oob_mode

- you set this when OOB has finished and then notify the SAS Layer.

sas_addr

- this normally points to an array holding the sas address of the phy, possibly somewhere in your my_phy struct.

attached_sas_addr

- set this when you (LLDD) receive an IDENTIFY frame or a FIS frame, _before_ notifying the SAS layer. The idea is that sometimes the LLDD may want to fake or provide a different SAS address on that phy/port and this allows it to do this. At best you should copy the sas address from the IDENTIFY frame or maybe generate a SAS address for SATA directly attached devices. The Discover process may later change this.

frame_rcvd

- this is where you copy the IDENTIFY/FIS frame when you get it; you lock, copy, set frame_rcvd_size and unlock the lock, and then call the event. It is a pointer since there's no way to know your hw frame size _exactly_, so you define the actual array in your phy struct and let this pointer point to it. You copy the frame from your DMAable memory to that area holding the lock.

sas_prim

- this is where primitives go when they're received. See sas.h. Grab the lock, set the primitive, release the lock, notify.

port

- this points to the sas_port if the phy belongs to a port -- the LLDD only reads this. It points to the sas_port this phy is part of. Set by the SAS Layer.

ha

- may be set; the SAS layer sets it anyway.

lldd_phy

- you should set this to point to your phy so you can find your way around faster when the SAS layer calls one of your callbacks and passes you a phy. If the sas_phy is embedded you can also use container_of -- whatever you prefer.

struct sas_port

The LLDD doesn't set any fields of this struct -- it only reads them. They should be self explanatory.

phy_mask is 32 bit, this should be enough for now, as I haven't heard of a HA having more than 8 phys.

lldd_port

- I haven't found use for that -- maybe other LLDD who wish to have internal port representation can make use of this.

struct sas_ha_struct

It normally is statically declared in your own LLDD structure describing your adapter:

```
struct my_sas_ha {
    blah;
    struct sas_ha_struct sas_ha;
    struct my_phy phys[MAX_PHYS];
    struct sas_port sas_ports[MAX_PHYS]; /* (1) */
    bleh;
};

(1) If your LLDD doesn't have its own port representation.
```

What needs to be initialized (sample function given below).

pcidev

sas_addr

- since the SAS layer doesn't want to mess with memory allocation, etc, this points to statically allocated array somewhere (say in your host adapter structure) and holds the SAS address of the host adapter as given by you or the manufacturer, etc.

sas_port

sas_phy

- an array of pointers to structures. (see note above on sas_addr). These must be set. See more notes below.

num_phys

- the number of phys present in the sas_phy array, and the number of ports present in the sas_port array. There can be a maximum num_phys ports (one per port) so we drop the num_ports, and only use num_phys.

The event interface:

```
/* LLDD calls these to notify the class of an event. */
void sas_notify_port_event(struct sas_phy *, enum port_event, gfp_t);
void sas_notify_phy_event(struct sas_phy *, enum phy_event, gfp_t);
```

The port notification:

```
/* The class calls these to notify the LLDD of an event. */
void (*lldd_port_formed)(struct sas_phy *);
void (*lldd_port_deformed)(struct sas_phy *);
```

If the LLDD wants notification when a port has been formed or deformed it sets those to a function satisfying the type.

A SAS LLDD should also implement at least one of the Task Management Functions (TMFs) described in SAM:

```
/* Task Management Functions. Must be called from process context. */
int (*lldd_abort_task)(struct sas_task *);
int (*lldd_abort_task_set)(struct domain_device *, u8 *lun);
int (*lldd_clear_task_set)(struct domain_device *, u8 *lun);
int (*lldd_I_T_nexus_reset)(struct domain_device *);
int (*lldd_lu_reset)(struct domain_device *, u8 *lun);
int (*lldd_query_task)(struct sas_task *);
```

For more information please read SAM from T10.org.

Port and Adapter management:

```
/* Port and Adapter management */
int (*lldd_clear_nexus_port)(struct sas_port *);
int (*lldd_clear_nexus_ha)(struct sas_ha_struct *);
```

A SAS LLDD should implement at least one of those.

Phy management:

```
/* Phy management */
int (*lldd_control_phy)(struct sas_phy *, enum phy_func);
```

lldd_ha

- set this to point to your HA struct. You can also use container_of if you embedded it as shown above.

A sample initialization and registration function can look like this (called last thing from probe()) *but* before you enable the phys to do OOB:

```
static int register_sas_ha(struct my_sas_ha *my_ha)
{
        int i;
        static struct sas_phy   *sas_phys[MAX_PHYS];
        static struct sas_port  *sas_ports[MAX_PHYS];

        my_ha->sas_ha.sas_addr = &my_ha->sas_addr[0];
```

```
        for (i = 0; i < MAX_PHYS; i++) {
                sas_phys[i] = &my_ha->phys[i].sas_phy;
                sas_ports[i] = &my_ha->sas_ports[i];
        }

        my_ha->sas_ha.sas_phy  = sas_phys;
        my_ha->sas_ha.sas_port = sas_ports;
        my_ha->sas_ha.num_phys = MAX_PHYS;

        my_ha->sas_ha.lldd_port_formed = my_port_formed;

        my_ha->sas_ha.lldd_dev_found = my_dev_found;
        my_ha->sas_ha.lldd_dev_gone = my_dev_gone;

        my_ha->sas_ha.lldd_execute_task = my_execute_task;

        my_ha->sas_ha.lldd_abort_task     = my_abort_task;
        my_ha->sas_ha.lldd_abort_task_set = my_abort_task_set;
        my_ha->sas_ha.lldd_clear_task_set = my_clear_task_set;
        my_ha->sas_ha.lldd_I_T_nexus_reset= NULL; (2)
        my_ha->sas_ha.lldd_lu_reset       = my_lu_reset;
        my_ha->sas_ha.lldd_query_task     = my_query_task;

        my_ha->sas_ha.lldd_clear_nexus_port = my_clear_nexus_port;
        my_ha->sas_ha.lldd_clear_nexus_ha = my_clear_nexus_ha;

        my_ha->sas_ha.lldd_control_phy = my_control_phy;

        return sas_register_ha(&my_ha->sas_ha);
}
```

(2) SAS 1.1 does not define I_T Nexus Reset TMF.

4.18.2 Events

Events are **the only way** a SAS LLDD notifies the SAS layer of anything. There is no other method or way a LLDD to tell the SAS layer of anything happening internally or in the SAS domain.

Phy events:

```
PHYE_LOSS_OF_SIGNAL, (C)
PHYE_OOB_DONE,
PHYE_OOB_ERROR,      (C)
PHYE_SPINUP_HOLD.
```

Port events, passed on a _phy_:

```
PORTE_BYTES_DMAED,      (M)
PORTE_BROADCAST_RCVD,   (E)
```

```
PORTE_LINK_RESET_ERR,   (C)
PORTE_TIMER_EVENT,      (C)
PORTE_HARD_RESET.
```

Host Adapter event:
HAE_RESET

A SAS LLDD should be able to generate

- at least one event from group C (choice),
- events marked M (mandatory) are mandatory (only one),
- events marked E (expander) if it wants the SAS layer to handle domain revalidation (only one such).
- Unmarked events are optional.

Meaning:

HAE_RESET

- when your HA got internal error and was reset.

PORTE_BYTES_DMAED

- on receiving an IDENTIFY/FIS frame

PORTE_BROADCAST_RCVD

- on receiving a primitive

PORTE_LINK_RESET_ERR

- timer expired, loss of signal, loss of DWS, etc.[1]

PORTE_TIMER_EVENT

- DWS reset timeout timer expired[1]

PORTE_HARD_RESET

- Hard Reset primitive received.

PHYE_LOSS_OF_SIGNAL

- the device is gone[1]

PHYE_OOB_DONE

- OOB went fine and oob_mode is valid

PHYE_OOB_ERROR

- Error while doing OOB, the device probably got disconnected.[1]

PHYE_SPINUP_HOLD

- SATA is present, COMWAKE not sent.

The Execute Command SCSI RPC:

[1] should set/clear the appropriate fields in the phy, or alternatively call the inlined sas_phy_disconnected() which is just a helper, from their tasklet.

```
int (*lldd_execute_task)(struct sas_task *, gfp_t gfp_flags);
```

Used to queue a task to the SAS LLDD. @task is the task to be executed. @gfp_mask is the gfp_mask defining the context of the caller.

This function should implement the Execute Command SCSI RPC,

That is, when lldd_execute_task() is called, the command go out on the transport *immediately*. There is *no* queuing of any sort and at any level in a SAS LLDD.

Returns:

- -SAS_QUEUE_FULL, -ENOMEM, nothing was queued;
- 0, the task(s) were queued.

```
struct sas_task {
        dev -- the device this task is destined to
        task_proto -- _one_ of enum sas_proto
        scatter -- pointer to scatter gather list array
        num_scatter -- number of elements in scatter
        total_xfer_len -- total number of bytes expected to be transferred
        data_dir -- PCI_DMA_...
        task_done -- callback when the task has finished execution
};
```

4.18.3 Discovery

The sysfs tree has the following purposes:

a) It shows you the physical layout of the SAS domain at the current time, i.e. how the domain looks in the physical world right now.

b) Shows some device parameters _at_discovery_time_.

This is a link to the tree(1) program, very useful in viewing the SAS domain: ftp://mama.indstate.edu/linux/tree/

I expect user space applications to actually create a graphical interface of this.

That is, the sysfs domain tree doesn't show or keep state if you e.g., change the meaning of the READY LED MEANING setting, but it does show you the current connection status of the domain device.

Keeping internal device state changes is responsibility of upper layers (Command set drivers) and user space.

When a device or devices are unplugged from the domain, this is reflected in the sysfs tree immediately, and the device(s) removed from the system.

The structure domain_device describes any device in the SAS domain. It is completely managed by the SAS layer. A task points to a domain device, this is how the SAS LLDD knows where to send the task(s) to. A SAS LLDD only reads the contents of the domain_device structure, but it never creates or destroys one.

4.18.4 Expander management from User Space

In each expander directory in sysfs, there is a file called ”smp_portal”. It is a binary sysfs attribute file, which implements an SMP portal (Note: this is *NOT* an SMP port), to which user space applications can send SMP requests and receive SMP responses.

Functionality is deceptively simple:

1. Build the SMP frame you want to send. The format and layout is described in the SAS spec. Leave the CRC field equal 0.

open(2)

2. Open the expander's SMP portal sysfs file in RW mode.

write(2)

3. Write the frame you built in 1.

read(2)

4. Read the amount of data you expect to receive for the frame you built. If you receive different amount of data you expected to receive, then there was some kind of error.

close(2)

All this process is shown in detail in the function do_smp_func() and its callers, in the file ”expander_conf.c”.

The kernel functionality is implemented in the file ”sas_expander.c”.

The program ”expander_conf.c” implements this. It takes one argument, the sysfs file name of the SMP portal to the expander, and gives expander information, including routing tables.

The SMP portal gives you complete control of the expander, so please be careful.

4.19 LPFC Driver Release Notes

Important: Starting in the 8.0.17 release, the driver began to be targeted strictly toward the upstream kernel. As such, we removed #ifdefs for older kernels (pre 2.6.10). The 8.0.16 release should be used if the driver is to be run on one of the older kernels.

The proposed modifications to the transport layer for FC remote ports and extended attribute support is now part of the upstream kernel as of 2.6.12. We no longer need to provide patches for this support, nor a *full* version which has old an new kernel support.

The driver now requires a 2.6.12 (if pre-release, 2.6.12-rc1) or later kernel.

Please heed these dependencies....

The following information is provided for additional background on the history of the driver as we push for upstream acceptance.

Cable pull and temporary device Loss:

> In older revisions of the lpfc driver, the driver internally queued i/o received from the midlayer. In the cases where a cable was pulled, link jitter, or a device temporarily

loses connectivity (due to its cable being removed, a switch rebooting, or a device reboot), the driver could hide the disappearance of the device from the midlayer. I/O's issued to the LLDD would simply be queued for a short duration, allowing the device to reappear or link come back alive, with no inadvertent side effects to the system. If the driver did not hide these conditions, i/o would be errored by the driver, the mid-layer would exhaust its retries, and the device would be taken offline. Manual intervention would be required to re-enable the device.

The community supporting kernel.org has driven an effort to remove internal queuing from all LLDDs. The philosophy is that internal queuing is unnecessary as the block layer already performs the queuing. Removing the queues from the LLDD makes a more predictable and more simple LLDD.

As a potential new addition to kernel.org, the 8.x driver was asked to have all internal queuing removed. Emulex complied with this request. In explaining the impacts of this change, Emulex has worked with the community in modifying the behavior of the SCSI midlayer so that SCSI devices can be temporarily suspended while transport events (such as those described) can occur.

The proposed patch was posted to the linux-scsi mailing list. The patch is contained in the 2.6.10-rc2 (and later) patch kits. As such, this patch is part of the standard 2.6.10 kernel.

By default, the driver expects the patches for block/unblock interfaces to be present in the kernel. No #define needs to be set to enable support.

4.19.1 Kernel Support

This source package is targeted for the upstream kernel only. (See notes at the top of this file). It relies on interfaces that are slowing migrating into the kernel.org kernel.

At this time, the driver requires the 2.6.12 (if pre-release, 2.6.12-rc1) kernel.

If a driver is needed for older kernels please utilize the 8.0.16 driver sources.

4.19.2 Patches

Thankfully, at this time, patches are not needed.

4.20 Megaraid Common Management Module

4.20.1 Overview

Different classes of controllers from LSI Logic accept and respond to the user applications in a similar way. They understand the same firmware control commands. Furthermore, the applications also can treat different classes of the controllers uniformly. Hence it is logical to have a single module that interfaces with the applications on one side and all the low level drivers on the other.

The advantages, though obvious, are listed for completeness:

i. Avoid duplicate code from the low level drivers.

ii. Unburden the low level drivers from having to export the character node device and related handling.

iii. Implement any policy mechanisms in one place.

iv. Applications have to interface with only module instead of multiple low level drivers.

Currently this module (called Common Management Module) is used only to issue ioctl commands. But this module is envisioned to handle all user space level interactions. So any 'proc', 'sysfs' implementations will be localized in this common module.

4.20.2 Credits

```
"Shared code in a third module, a "library module", is an acceptable
solution. modprobe automatically loads dependent modules, so users
running "modprobe driver1" or "modprobe driver2" would automatically
load the shared library module."
```

- Jeff Garzik (jgarzik@pobox.com), 02.25.2004 LKML

```
"As Jeff hinted, if your userspace<->driver API is consistent between
your new MPT-based RAID controllers and your existing megaraid driver,
then perhaps you need a single small helper module (lsiioctl or some
better name), loaded by both mptraid and megaraid automatically, which
handles registering the /dev/megaraid node dynamically. In this case,
both mptraid and megaraid would register with lsiioctl for each
adapter discovered, and lsiioctl would essentially be a switch,
redirecting userspace tool ioctls to the appropriate driver."
```

- Matt Domsch, (Matt_Domsch@dell.com), 02.25.2004 LKML

4.20.3 Design

The Common Management Module is implemented in megaraid_mm.[ch] files. This module acts as a registry for low level hba drivers. The low level drivers (currently only megaraid) register each controller with the common module.

The applications interface with the common module via the character device node exported by the module.

The lower level drivers now understand only a new improved ioctl packet called uioc_t. The management module converts the older ioctl packets from the older applications into uioc_t. After driver handles the uioc_t, the common module will convert that back into the old format before returning to applications.

As new applications evolve and replace the old ones, the old packet format will be retired.

Common module dedicates one uioc_t packet to each controller registered. This can easily be more than one. But since megaraid is the only low level driver today, and it can handle only one ioctl, there is no reason to have more. But as new controller classes get added, this will be tuned appropriately.

4.21 NCR53C8XX/SYM53C8XX drivers

Written by Gerard Roudier <groudier@free.fr>

21 Rue Carnot

95170 DEUIL LA BARRE - FRANCE

29 May 1999

4.21.1 1. Introduction

The initial Linux ncr53c8xx driver has been a port of the ncr driver from FreeBSD that has been achieved in November 1995 by:

- Gerard Roudier <groudier@free.fr>

The original driver has been written for 386bsd and FreeBSD by:

- Wolfgang Stanglmeier <wolf@cologne.de>
- Stefan Esser <se@mi.Uni-Koeln.de>

It is now available as a bundle of 2 drivers:

- ncr53c8xx generic driver that supports all the SYM53C8XX family including the earliest 810 rev. 1, the latest 896 (2 channel LVD SCSI controller) and the new 895A (1 channel LVD SCSI controller).
- sym53c8xx enhanced driver (a.k.a. 896 drivers) that drops support of oldest chips in order to gain advantage of new features, as LOAD/STORE instructions available since the 810A and hardware phase mismatch available with the 896 and the 895A.

You can find technical information about the NCR 8xx family in the PCI-HOWTO written by Michael Will and in the SCSI-HOWTO written by Drew Eckhardt.

Information about new chips is available at LSILOGIC web server:

- http://www.lsilogic.com/

SCSI standard documentations are available at SYMBIOS ftp server:

- ftp://ftp.symbios.com/

Useful SCSI tools written by Eric Youngdale are available at tsx-11:

- ftp://tsx-11.mit.edu/pub/linux/ALPHA/scsi/scsiinfo-X.Y.tar.gz
- ftp://tsx-11.mit.edu/pub/linux/ALPHA/scsi/scsidev-X.Y.tar.gz

These tools are not ALPHA but quite clean and work quite well. It is essential you have the 'scsiinfo' package.

This short documentation describes the features of the generic and enhanced drivers, configuration parameters and control commands available through the proc SCSI file system read / write operations.

This driver has been tested OK with linux/i386, Linux/Alpha and Linux/PPC.

Latest driver version and patches are available at:

- ftp://ftp.tux.org/pub/people/gerard-roudier

or

- ftp://ftp.symbios.com/mirror/ftp.tux.org/pub/tux/roudier/drivers

I am not a native speaker of English and there are probably lots of mistakes in this README file. Any help will be welcome.

4.21.2 2. Supported chips and SCSI features

The following features are supported for all chips:

- Synchronous negotiation
- Disconnection
- Tagged command queuing
- SCSI parity checking
- Master parity checking

”Wide negotiation” is supported for chips that allow it. The following table shows some characteristics of NCR 8xx family chips and what drivers support them.

Chip	On board SDMS BIOS	Wide	SCSI std.	Max. sync	Supported by the generic driver	Supported by the enhanced driver
810	N	N	FAST10	10 MB/s	Y	N
810A	N	N	FAST10	10 MB/s	Y	Y
815	Y	N	FAST10	10 MB/s	Y	N
825	Y	Y	FAST10	20 MB/s	Y	N
825A	Y	Y	FAST10	20 MB/s	Y	Y
860	N	N	FAST20	20 MB/s	Y	Y
875	Y	Y	FAST20	40 MB/s	Y	Y
876	Y	Y	FAST20	40 MB/s	Y	Y
895	Y	Y	FAST40	80 MB/s	Y	Y
895A	Y	Y	FAST40	80 MB/s	Y	Y
896	Y	Y	FAST40	80 MB/s	Y	Y
897	Y	Y	FAST40	80 MB/s	Y	Y
1510D	Y	Y	FAST40	80 MB/s	Y	Y
1010	Y	Y	FAST80	160 MB/s	N	Y
1010_66[1]	Y	Y	FAST80	160 MB/s	N	Y

Summary of other supported features:

Module
: allow to load the driver

Memory mapped I/O
: increases performance

Profiling information
: read operations from the proc SCSI file system

[1] Chip supports 33MHz and 66MHz PCI buses.

Control commands
write operations to the proc SCSI file system

Debugging information
written to syslog (expert only)

Serial NVRAM
Symbios and Tekram formats

- Scatter / gather
- Shared interrupt
- Boot setup commands

4.21.3 3. Advantages of the enhanced 896 driver

3.1 Optimized SCSI SCRIPTS

The 810A, 825A, 875, 895, 896 and 895A support new SCSI SCRIPTS instructions named LOAD and STORE that allow to move up to 1 DWORD from/to an IO register to/from memory much faster that the MOVE MEMORY instruction that is supported by the 53c7xx and 53c8xx family. The LOAD/STORE instructions support absolute and DSA relative addressing modes. The SCSI SCRIPTS had been entirely rewritten using LOAD/STORE instead of MOVE MEMORY instructions.

3.2 New features of the SYM53C896 (64 bit PCI dual LVD SCSI controller)

The 896 and the 895A allows handling of the phase mismatch context from SCRIPTS (avoids the phase mismatch interrupt that stops the SCSI processor until the C code has saved the context of the transfer). Implementing this without using LOAD/STORE instructions would be painful and I didn't even want to try it.

The 896 chip supports 64 bit PCI transactions and addressing, while the 895A supports 32 bit PCI transactions and 64 bit addressing. The SCRIPTS processor of these chips is not true 64 bit, but uses segment registers for bit 32-63. Another interesting feature is that LOAD/STORE instructions that address the on-chip RAM (8k) remain internal to the chip.

Due to the use of LOAD/STORE SCRIPTS instructions, this driver does not support the following chips:

- SYM53C810 revision < 0x10 (16)
- SYM53C815 all revisions
- SYM53C825 revision < 0x10 (16)

4.21.4 4. Memory mapped I/O versus normal I/O

Memory mapped I/O has less latency than normal I/O. Since linux-1.3.x, memory mapped I/O is used rather than normal I/O. Memory mapped I/O seems to work fine on most hardware configurations, but some poorly designed motherboards may break this feature.

The configuration option CONFIG_SCSI_NCR53C8XX_IOMAPPED forces the driver to use normal I/O in all cases.

4.21.5 5. Tagged command queueing

Queuing more than 1 command at a time to a device allows it to perform optimizations based on actual head positions and its mechanical characteristics. This feature may also reduce average command latency. In order to really gain advantage of this feature, devices must have a reasonable cache size (No miracle is to be expected for a low-end hard disk with 128 KB or less). Some known SCSI devices do not properly support tagged command queuing. Generally, firmware revisions that fix this kind of problems are available at respective vendor web/ftp sites. All I can say is that the hard disks I use on my machines behave well with this driver with tagged command queuing enabled:

- IBM S12 0662
- Conner 1080S
- Quantum Atlas I
- Quantum Atlas II

If your controller has NVRAM, you can configure this feature per target from the user setup tool. The Tekram Setup program allows to tune the maximum number of queued commands up to 32. The Symbios Setup only allows to enable or disable this feature.

The maximum number of simultaneous tagged commands queued to a device is currently set to 8 by default. This value is suitable for most SCSI disks. With large SCSI disks (>= 2GB, cache >= 512KB, average seek time <= 10 ms), using a larger value may give better performances.

The sym53c8xx driver supports up to 255 commands per device, and the generic ncr53c8xx driver supports up to 64, but using more than 32 is generally not worth-while, unless you are using a very large disk or disk array. It is noticeable that most of recent hard disks seem not to accept more than 64 simultaneous commands. So, using more than 64 queued commands is probably just resource wasting.

If your controller does not have NVRAM or if it is managed by the SDMS BIOS/SETUP, you can configure tagged queueing feature and device queue depths from the boot command-line. For example:

```
ncr53c8xx=tags:4/t2t3q15-t4q7/t1u0q32
```

will set tagged commands queue depths as follow:

- target 2 all luns on controller 0 --> 15
- target 3 all luns on controller 0 --> 15
- target 4 all luns on controller 0 --> 7
- target 1 lun 0 on controller 1 --> 32

- all other target/lun --> 4

In some special conditions, some SCSI disk firmwares may return a QUEUE FULL status for a SCSI command. This behaviour is managed by the driver using the following heuristic:

- Each time a QUEUE FULL status is returned, tagged queue depth is reduced to the actual number of disconnected commands.
- Every 1000 successfully completed SCSI commands, if allowed by the current limit, the maximum number of queueable commands is incremented.

Since QUEUE FULL status reception and handling is resource wasting, the driver notifies by default this problem to user by indicating the actual number of commands used and their status, as well as its decision on the device queue depth change. The heuristic used by the driver in handling QUEUE FULL ensures that the impact on performances is not too bad. You can get rid of the messages by setting verbose level to zero, as follow:

1st method:
boot your system using 'ncr53c8xx=verb:0' option.

2nd method:
apply ”setverbose 0” control command to the proc fs entry corresponding to your controller after boot-up.

4.21.6 6. Parity checking

The driver supports SCSI parity checking and PCI bus master parity checking. These features must be enabled in order to ensure safe data transfers. However, some flawed devices or mother boards will have problems with parity. You can disable either PCI parity or SCSI parity checking by entering appropriate options from the boot command line. (See 10: Boot setup commands).

4.21.7 7. Profiling information

Profiling information is available through the proc SCSI file system. Since gathering profiling information may impact performances, this feature is disabled by default and requires a compilation configuration option to be set to Y.

The device associated with a host has the following pathname:

```
/proc/scsi/ncr53c8xx/N     (N=0,1,2 ....)
```

Generally, only 1 board is used on hardware configuration, and that device is:

```
/proc/scsi/ncr53c8xx/0
```

However, if the driver has been made as module, the number of the hosts is incremented each time the driver is loaded.

In order to display profiling information, just enter:

```
cat /proc/scsi/ncr53c8xx/0
```

and you will get something like the following text:

```
General information:
Chip NCR53C810, device id 0x1, revision id 0x2
IO port address 0x6000, IRQ number 10
Using memory mapped IO at virtual address 0x282c000
Synchronous transfer period 25, max commands per lun 4
Profiling information:
num_trans    = 18014
num_kbytes   = 671314
num_disc     = 25763
num_break    = 1673
num_int      = 1685
num_fly      = 18038
ms_setup     = 4940
ms_data      = 369940
ms_disc      = 183090
ms_post      = 1320
```

General information is easy to understand. The device ID and the revision ID identify the SCSI chip as follows:

Chip	Device id	Revision Id
810	0x1	< 0x10
810A	0x1	>= 0x10
815	0x4	
825	0x3	< 0x10
860	0x6	
825A	0x3	>= 0x10
875	0xf	
895	0xc	

The profiling information is updated upon completion of SCSI commands. A data structure is allocated and zeroed when the host adapter is attached. So, if the driver is a module, the profile counters are cleared each time the driver is loaded. The ”clearprof” command allows you to clear these counters at any time.

The following counters are available:

(”num” prefix means ”number of”, ”ms” means milli-seconds)

num_trans
Number of completed commands Example above: 18014 completed commands

num_kbytes
Number of kbytes transferred Example above: 671 MB transferred

num_disc
Number of SCSI disconnections Example above: 25763 SCSI disconnections

num_break
number of script interruptions (phase mismatch) Example above: 1673 script interruptions

num_int
Number of interrupts other than ”on the fly” Example above: 1685 interruptions not ”on the fly”

num_fly
Number of interrupts ”on the fly” Example above: 18038 interruptions ”on the fly”

ms_setup
Elapsed time for SCSI commands setups Example above: 4.94 seconds

ms_data
Elapsed time for data transfers Example above: 369.94 seconds spent for data transfer

ms_disc
Elapsed time for SCSI disconnections Example above: 183.09 seconds spent disconnected

ms_post
Elapsed time for command post processing (time from SCSI status get to command completion call) Example above: 1.32 seconds spent for post processing

Due to the 1/100 second tick of the system clock, ”ms_post” time may be wrong.

In the example above, we got 18038 interrupts ”on the fly” and only 1673 script breaks generally due to disconnections inside a segment of the scatter list.

4.21.8 8. Control commands

Control commands can be sent to the driver with write operations to the proc SCSI file system. The generic command syntax is the following:

```
echo "<verb> <parameters>" >/proc/scsi/ncr53c8xx/0
(assumes controller number is 0)
```

Using ”all” for ”<target>” parameter with the commands below will apply to all targets of the SCSI chain (except the controller).

Available commands:

8.1 Set minimum synchronous period factor

setsync <target> <period factor>

target
target number

period
minimum synchronous period. Maximum speed = 1000/(4*period factor) except for special cases below.

Specify a period of 255, to force asynchronous transfer mode.

- 10 means 25 nano-seconds synchronous period
- 11 means 30 nano-seconds synchronous period
- 12 means 50 nano-seconds synchronous period

8.2 Set wide size

setwide <target> <size>

target
target number

size
0=8 bits, 1=16bits

8.3 Set maximum number of concurrent tagged commands

settags <target> <tags>

target
target number

tags
number of concurrent tagged commands must not be greater than SCSI_NCR_MAX_TAGS (default: 8)

8.4 Set order type for tagged command

setorder <order>

order
3 possible values:

simple:
use SIMPLE TAG for all operations (read and write)

ordered:
use ORDERED TAG for all operations

default:
use default tag type, SIMPLE TAG for read operations ORDERED TAG for write operations

8.5 Set debug mode

setdebug <list of debug flags>

Available debug flags:

alloc	print info about memory allocations (ccb, lcb)
queue	print info about insertions into the command start queue
result	print sense data on CHECK CONDITION status
scatter	print info about the scatter process
scripts	print info about the script binding process
tiny	print minimal debugging information
timing	print timing information of the NCR chip
nego	print information about SCSI negotiations
phase	print information on script interruptions

Use ”setdebug” with no argument to reset debug flags.

8.6 Clear profile counters

clearprof

The profile counters are automatically cleared when the amount of data transferred reaches 1000 GB in order to avoid overflow. The ”clearprof” command allows you to clear these counters at any time.

8.7 Set flag (no_disc)

setflag <target> <flag>

target: target number

For the moment, only one flag is available:

no_disc: not allow target to disconnect.

Do not specify any flag in order to reset the flag. For example:

setflag 4
will reset no_disc flag for target 4, so will allow it disconnections.

setflag all
will allow disconnection for all devices on the SCSI bus.

8.8 Set verbose level

setverbose #level

The driver default verbose level is 1. This command allows to change th driver verbose level after boot-up.

8.9 Reset all logical units of a target

resetdev <target>

target
target number

The driver will try to send a BUS DEVICE RESET message to the target. (Only supported by the SYM53C8XX driver and provided for test purpose)

8.10 Abort all tasks of all logical units of a target

cleardev <target>

target
target number

The driver will try to send a ABORT message to all the logical units of the target.

(Only supported by the SYM53C8XX driver and provided for test purpose)

4.21.9 9. Configuration parameters

If the firmware of all your devices is perfect enough, all the features supported by the driver can be enabled at start-up. However, if only one has a flaw for some SCSI feature, you can disable the support by the driver of this feature at linux start-up and enable this feature after boot-up only for devices that support it safely.

CONFIG_SCSI_NCR53C8XX_IOMAPPED (default answer: n)
Answer "y" if you suspect your mother board to not allow memory mapped I/O.

May slow down performance a little. This option is required by Linux/PPC and is used no matter what you select here. Linux/PPC suffers no performance loss with this option since all IO is memory mapped anyway.

CONFIG_SCSI_NCR53C8XX_DEFAULT_TAGS (default answer: 8)
Default tagged command queue depth.

CONFIG_SCSI_NCR53C8XX_MAX_TAGS (default answer: 8)
This option allows you to specify the maximum number of tagged commands that can be queued to a device. The maximum supported value is 32.

CONFIG_SCSI_NCR53C8XX_SYNC (default answer: 5)
This option allows you to specify the frequency in MHz the driver will use at boot time for synchronous data transfer negotiations. This frequency can be changed later with the "setsync" control command. 0 means "asynchronous data transfers".

CONFIG_SCSI_NCR53C8XX_FORCE_SYNC_NEGO (default answer: n)
Force synchronous negotiation for all SCSI-2 devices.

Some SCSI-2 devices do not report this feature in byte 7 of inquiry response but do support it properly (TAMARACK scanners for example).

CONFIG_SCSI_NCR53C8XX_NO_DISCONNECT (default and only reasonable answer: n)
If you suspect a device of yours does not properly support disconnections, you can answer "y". Then, all SCSI devices will never disconnect the bus even while performing long SCSI operations.

CONFIG_SCSI_NCR53C8XX_SYMBIOS_COMPAT
Genuine SYMBIOS boards use GPIO0 in output for controller LED and GPIO3 bit as a flag indicating singled-ended/differential interface. If all the boards of your system are genuine SYMBIOS boards or use BIOS and drivers from SYMBIOS, you would want to enable this option.

This option must NOT be enabled if your system has at least one 53C8XX based scsi board with a vendor-specific BIOS. For example, Tekram DC-390/U, DC-390/W and DC-390/F

scsi controllers use a vendor-specific BIOS and are known to not use SYMBIOS compatible GPIO wiring. So, this option must not be enabled if your system has such a board installed.

CONFIG_SCSI_NCR53C8XX_NVRAM_DETECT
Enable support for reading the serial NVRAM data on Symbios and some Symbios compatible cards, and Tekram DC390W/U/F cards. Useful for systems with more than one Symbios compatible controller where at least one has a serial NVRAM, or for a system with a mixture of Symbios and Tekram cards. Enables setting the boot order of host adaptors to something other than the default order or ”reverse probe” order. Also enables Symbios and Tekram cards to be distinguished so CONFIG_SCSI_NCR53C8XX_SYMBIOS_COMPAT may be set in a system with a mixture of Symbios and Tekram cards so the Symbios cards can make use of the full range of Symbios features, differential, led pin, without causing problems for the Tekram card(s).

4.21.10 10. Boot setup commands

10.1 Syntax

Setup commands can be passed to the driver either at boot time or as a string variable using 'insmod'.

A boot setup command for the ncr53c8xx (sym53c8xx) driver begins with the driver name ”ncr53c8xx=”(sym53c8xx). The kernel syntax parser then expects an optional list of integers separated with comma followed by an optional list of comma-separated strings. Example of boot setup command under lilo prompt:

```
lilo: linux root=/dev/hda2 ncr53c8xx=tags:4,sync:10,debug:0x200
```

- enable tagged commands, up to 4 tagged commands queued.
- set synchronous negotiation speed to 10 Mega-transfers / second.
- set DEBUG_NEGO flag.

Since comma seems not to be allowed when defining a string variable using 'insmod', the driver also accepts <space> as option separator. The following command will install driver module with the same options as above:

```
insmod ncr53c8xx.o ncr53c8xx="tags:4 sync:10 debug:0x200"
```

For the moment, the integer list of arguments is discarded by the driver. It will be used in the future in order to allow a per controller setup.

Each string argument must be specified as ”keyword:value”. Only lower-case characters and digits are allowed.

In a system that contains multiple 53C8xx adapters insmod will install the specified driver on each adapter. To exclude a chip use the 'excl' keyword.

The sequence of commands:

```
insmod sym53c8xx sym53c8xx=excl:0x1400
insmod ncr53c8xx
```

installs the sym53c8xx driver on all adapters except the one at IO port address 0x1400 and then installs the ncr53c8xx driver to the adapter at IO port address 0x1400.

10.2 Available arguments

10.2.1 Master parity checking

mpar:y	enabled
mpar:n	disabled

10.2.2 Scsi parity checking

spar:y	enabled
spar:n	disabled

10.2.3 Scsi disconnections

disc:y	enabled
disc:n	disabled

10.2.4 Special features

Only apply to 810A, 825A, 860, 875 and 895 controllers. Have no effect with other ones.

specf:y	(or 1) enabled
specf:n	(or 0) disabled
specf:3	enabled except Memory Write And Invalidate

The default driver setup is 'specf:3'. As a consequence, option 'specf:y' must be specified in the boot setup command to enable Memory Write And Invalidate.

10.2.5 Ultra SCSI support

Only apply to 860, 875, 895, 895a, 896, 1010 and 1010_66 controllers. Have no effect with other ones.

ultra:n	All ultra speeds enabled
ultra:2	Ultra2 enabled
ultra:1	Ultra enabled
ultra:0	Ultra speeds disabled

10.2.6 Default number of tagged commands

tags:0 (or tags:1)	tagged command queuing disabled
tags:#tags (#tags > 1)	tagged command queuing enabled

#tags will be truncated to the max queued commands configuration parameter. This option also allows to specify a command queue depth for each device that support tagged command queueing.

Example:

```
ncr53c8xx=tags:10/t2t3q16-t5q24/t1u2q32
```

will set devices queue depth as follow:

- controller #0 target #2 and target #3 -> 16 commands,
- controller #0 target #5 -> 24 commands,
- controller #1 target #1 logical unit #2 -> 32 commands,
- all other logical units (all targets, all controllers) -> 10 commands.

10.2.7 Default synchronous period factor

<table>
<tr><td>sync:255</td><td>disabled (asynchronous transfer mode)</td></tr>
<tr><td>sync:#factor</td><td>
<table>
<tr><td>#factor = 10</td><td>Ultra-2 SCSI 40 Mega-transfers / second</td></tr>
<tr><td>#factor = 11</td><td>Ultra-2 SCSI 33 Mega-transfers / second</td></tr>
<tr><td>#factor < 25</td><td>Ultra SCSI 20 Mega-transfers / second</td></tr>
<tr><td>#factor < 50</td><td>Fast SCSI-2</td></tr>
</table>
</td></tr>
</table>

In all cases, the driver will use the minimum transfer period supported by controllers according to NCR53C8XX chip type.

10.2.8 Negotiate synchronous with all devices

(force sync nego)

fsn:y	enabled
fsn:n	disabled

10.2.9 Verbosity level

verb:0	minimal
verb:1	normal
verb:2	too much

10.2.10 Debug mode

debug:0	clear debug flags
debug:#x	set debug flags #x is an integer value combining the following power-of-2 values:

DEBUG_ALLOC	0x1
DEBUG_PHASE	0x2
DEBUG_POLL	0x4
DEBUG_QUEUE	0x8
DEBUG_RESULT	0x10
DEBUG_SCATTER	0x20
DEBUG_SCRIPT	0x40
DEBUG_TINY	0x80
DEBUG_TIMING	0x100
DEBUG_NEGO	0x200
DEBUG_TAGS	0x400
DEBUG_FREEZE	0x800
DEBUG_RESTART	0x1000

You can play safely with DEBUG_NEGO. However, some of these flags may generate bunches of syslog messages.

10.2.11 Burst max

burst:0	burst disabled
burst:255	get burst length from initial IO register settings.
burst:#x	burst enabled (1<<#x burst transfers max) #x is an integer value which is log base 2 of the burst transfers max. The NCR53C875 and NCR53C825A support up to 128 burst transfers (#x = 7). Other chips only support up to 16 (#x = 4). This is a maximum value. The driver set the burst length according to chip and revision ids. By default the driver uses the maximum value supported by the chip.

10.2.12 LED support

led:1	enable LED support
led:0	disable LED support

Do not enable LED support if your scsi board does not use SDMS BIOS. (See 'Configuration parameters')

10.2.13 Max wide

wide:1	wide scsi enabled
wide:0	wide scsi disabled

Some scsi boards use a 875 (ultra wide) and only supply narrow connectors. If you have connected a wide device with a 50 pins to 68 pins cable converter, any accepted wide negotiation will break further data transfers. In such a case, using ”wide:0” in the bootup command will be helpful.

10.2.14 Differential mode

diff:0	never set up diff mode
diff:1	set up diff mode if BIOS set it
diff:2	always set up diff mode
diff:3	set diff mode if GPIO3 is not set

10.2.15 IRQ mode

irqm:0	always open drain
irqm:1	same as initial settings (assumed BIOS settings)
irqm:2	always totem pole
irqm:0x10	driver will not use IRQF_SHARED flag when requesting irq

(Bits 0x10 and 0x20 can be combined with hardware irq mode option)

10.2.16 Reverse probe

revprob:n	probe chip ids from the PCI configuration in this order: 810, 815, 820, 860, 875, 885, 895, 896
revprob:y	probe chip ids in the reverse order.

10.2.17 Fix up PCI configuration space

pcifix:<option bits>

Available option bits:

0x0	No attempt to fix PCI configuration space registers values.
0x1	Set PCI cache-line size register if not set.
0x2	Set write and invalidate bit in PCI command register.
0x4	Increase if necessary PCI latency timer according to burst max.

Use 'pcifix:7' in order to allow the driver to fix up all PCI features.

10.2.18 Serial NVRAM

nvram:n	do not look for serial NVRAM
nvram:y	test controllers for onboard serial NVRAM

(alternate binary form) mvram=<bits options>

0x01	look for NVRAM (equivalent to nvram=y)
0x02	ignore NVRAM ”Synchronous negotiation” parameters for all devices
0x04	ignore NVRAM ”Wide negotiation” parameter for all devices
0x08	ignore NVRAM ”Scan at boot time” parameter for all devices
0x80	also attach controllers set to OFF in the NVRAM (sym53c8xx only)

10.2.19 Check SCSI BUS

buschk:<option bits>

Available option bits:

0x0:	No check.
0x1:	Check and do not attach the controller on error.
0x2:	Check and just warn on error.
0x4:	Disable SCSI bus integrity checking.

10.2.20 Exclude a host from being attached

excl=<io_address>

Prevent host at a given io address from being attached. For example 'ncr53c8xx=excl:0xb400,excl:0xc000' indicate to the ncr53c8xx driver not to attach hosts at address 0xb400 and 0xc000.

10.2.21 Suggest a default SCSI id for hosts

hostid:255	no id suggested.
hostid:#x	(0 < x < 7) x suggested for hosts SCSI id.

If a host SCSI id is available from the NVRAM, the driver will ignore any value suggested as boot option. Otherwise, if a suggested value different from 255 has been supplied, it will use it. Otherwise, it will try to deduce the value previously set in the hardware and use value 7 if the hardware value is zero.

10.2.22 Enable use of IMMEDIATE ARBITRATION

(only supported by the sym53c8xx driver. See 10.7 for more details)

<table>
<tr><td>iarb:0</td><td>do not use this feature.</td></tr>
<tr><td>iarb:#x</td><td>use this feature according to bit fields as follow:

<table><tr><td>bit 0 (1)</td><td>enable IARB each time the initiator has been reselected when it arbitrated for the SCSI BUS.</td></tr><tr><td>(#x >> 4)</td><td>maximum number of successive settings of IARB if the initiator win arbitration and it has other commands to send to a device.</td></tr></table></td></tr>
</table>

Boot fail safe

safe:y load the following assumed fail safe initial setup

master parity	disabled	mpar:n
scsi parity	enabled	spar:y
disconnections	not allowed	disc:n
special features	disabled	specf:n
ultra scsi	disabled	ultra:n
force sync negotiation	disabled	fsn:n
reverse probe	disabled	revprob:n
PCI fix up	disabled	pcifix:0
serial NVRAM	enabled	nvram:y
verbosity level	2	verb:2
tagged command queuing	disabled	tags:0
synchronous negotiation	disabled	sync:255
debug flags	none	debug:0
burst length	from BIOS settings	burst:255
LED support	disabled	led:0
wide support	disabled	wide:0
settle time	10 seconds	settle:10
differential support	from BIOS settings	diff:1
irq mode	from BIOS settings	irqm:1
SCSI BUS check	do not attach on error	buschk:1
immediate arbitration	disabled	iarb:0

10.3 Advised boot setup commands

If the driver has been configured with default options, the equivalent boot setup is:

```
ncr53c8xx=mpar:y,spar:y,disc:y,specf:3,fsn:n,ultra:2,fsn:n,revprob:n,verb:1\
          tags:0,sync:50,debug:0,burst:7,led:0,wide:1,settle:2,diff:0,irqm:0
```

For an installation diskette or a safe but not fast system, boot setup can be:

```
ncr53c8xx=safe:y,mpar:y,disc:y
ncr53c8xx=safe:y,disc:y
ncr53c8xx=safe:y,mpar:y
ncr53c8xx=safe:y
```

My personal system works flawlessly with the following equivalent setup:

```
ncr53c8xx=mpar:y,spar:y,disc:y,specf:1,fsn:n,ultra:2,fsn:n,revprob:n,verb:1\
          tags:32,sync:12,debug:0,burst:7,led:1,wide:1,settle:2,diff:0,irqm:0
```

The driver prints its actual setup when verbosity level is 2. You can try ”ncr53c8xx=verb:2” to get the ”static” setup of the driver, or add ”verb:2” to your boot setup command in order to check the actual setup the driver is using.

10.4 PCI configuration fix-up boot option

pcifix:<option bits>

Available option bits:

0x1	Set PCI cache-line size register if not set.
0x2	Set write and invalidate bit in PCI command register.

Use 'pcifix:3' in order to allow the driver to fix both PCI features.

These options only apply to new SYMBIOS chips 810A, 825A, 860, 875 and 895 and are only supported for Pentium and 486 class processors. Recent SYMBIOS 53C8XX scsi processors are able to use PCI read multiple and PCI write and invalidate commands. These features require the cache line size register to be properly set in the PCI configuration space of the chips. On the other hand, chips will use PCI write and invalidate commands only if the corresponding bit is set to 1 in the PCI command register.

Not all PCI bioses set the PCI cache line register and the PCI write and invalidate bit in the PCI configuration space of 53C8XX chips. Optimized PCI accesses may be broken for some PCI/memory controllers or make problems with some PCI boards.

This fix-up worked flawlessly on my previous system. (MB Triton HX / 53C875 / 53C810A) I use these options at my own risks as you will do if you decide to use them too.

10.5 Serial NVRAM support boot option

nvram:n	do not look for serial NVRAM
nvram:y	test controllers for onboard serial NVRAM

This option can also been entered as an hexadecimal value that allows to control what information the driver will get from the NVRAM and what information it will ignore. For details see '17. Serial NVRAM support'.

When this option is enabled, the driver tries to detect all boards using a Serial NVRAM. This memory is used to hold user set up parameters.

The parameters the driver is able to get from the NVRAM depend on the data format used, as follow:

	Tekram format	Symbios format
General and host parameters		
• Boot order	N	Y
• Host SCSI ID	Y	Y
• SCSI parity checking	Y	Y
• Verbose boot messages	N	Y
SCSI devices parameters		
• Synchronous transfer speed	Y	Y
• Wide 16 / Narrow	Y	Y
• Tagged Command Queuing enabled	Y	Y
• Disconnections enabled	Y	Y
• Scan at boot time	N	Y

In order to speed up the system boot, for each device configured without the ”scan at boot time” option, the driver forces an error on the first TEST UNIT READY command received for this device.

Some SDMS BIOS revisions seem to be unable to boot cleanly with very fast hard disks. In such a situation you cannot configure the NVRAM with optimized parameters value.

The 'nvram' boot option can be entered in hexadecimal form in order to ignore some options configured in the NVRAM, as follow:

mvram=<bits options>

0x01	look for NVRAM (equivalent to nvram=y)
0x02	ignore NVRAM ”Synchronous negotiation” parameters for all devices
0x04	ignore NVRAM ”Wide negotiation” parameter for all devices
0x08	ignore NVRAM ”Scan at boot time” parameter for all devices
0x80	also attach controllers set to OFF in the NVRAM (sym53c8xx only)

Option 0x80 is only supported by the sym53c8xx driver and is disabled by default. Result is that, by default (option not set), the sym53c8xx driver will not attach controllers set to OFF in the NVRAM.

The ncr53c8xx always tries to attach all the controllers. Option 0x80 has not been added to the ncr53c8xx driver, since it has been reported to confuse users who use this driver since a long time. If you desire a controller not to be attached by the ncr53c8xx driver at Linux boot, you must use the 'excl' driver boot option.

10.6 SCSI BUS checking boot option.

When this option is set to a non-zero value, the driver checks SCSI lines logic state, 100 micro-seconds after having asserted the SCSI RESET line. The driver just reads SCSI lines and checks all lines read FALSE except RESET. Since SCSI devices shall release the BUS at most 800 nano-seconds after SCSI RESET has been asserted, any signal to TRUE may indicate a SCSI BUS problem. Unfortunately, the following common SCSI BUS problems are not detected:

- Only 1 terminator installed.
- Misplaced terminators.
- Bad quality terminators.

On the other hand, either bad cabling, broken devices, not conformant devices, ... may cause a SCSI signal to be wrong when the driver reads it.

10.7 IMMEDIATE ARBITRATION boot option

This option is only supported by the SYM53C8XX driver (not by the NCR53C8XX).

SYMBIOS 53C8XX chips are able to arbitrate for the SCSI BUS as soon as they have detected an expected disconnection (BUS FREE PHASE). For this process to be started, bit 1 of SCNTL1 IO register must be set when the chip is connected to the SCSI BUS.

When this feature has been enabled for the current connection, the chip has every chance to win arbitration if only devices with lower priority are competing for the SCSI BUS. By the way, when the chip is using SCSI id 7, then it will for sure win the next SCSI BUS arbitration.

Since, there is no way to know what devices are trying to arbitrate for the BUS, using this feature can be extremely unfair. So, you are not advised to enable it, or at most enable this feature for the case the chip lost the previous arbitration (boot option 'iarb:1').

This feature has the following advantages:

a) Allow the initiator with ID 7 to win arbitration when it wants so.

b) Overlap at least 4 micro-seconds of arbitration time with the execution of SCRIPTS that deal with the end of the current connection and that starts the next job.

Hmmm... But (a) may just prevent other devices from reselecting the initiator, and delay data transfers or status/completions, and (b) may just waste SCSI BUS bandwidth if the SCRIPTS execution lasts more than 4 micro-seconds.

The use of IARB needs the SCSI_NCR_IARB_SUPPORT option to have been defined at compile time and the 'iarb' boot option to have been set to a non zero value at boot time. It is not that useful for real work, but can be used to stress SCSI devices or for some applications that can gain advantage of it. By the way, if you experience badnesses like 'unexpected disconnections',

'bad reselections', etc... when using IARB on heavy IO load, you should not be surprised, because force-feeding anything and blocking its arse at the same time cannot work for a long time. :-))

4.21.11 11. Some constants and flags of the ncr53c8xx.h header file

Some of these are defined from the configuration parameters. To change other ”defines”, you must edit the header file. Do that only if you know what you are doing.

SCSI_NCR_SETUP_SPECIAL_FEATURES (default: defined)
If defined, the driver will enable some special features according to chip and revision id.

For 810A, 860, 825A, 875 and 895 scsi chips, this option enables support of features that reduce load of PCI bus and memory accesses during scsi transfer processing: burst opcode fetch, read multiple, read line, prefetch, cache line, write and invalidate, burst 128 (875 only), large dma fifo (875 only), offset 16 (875 only). Can be changed by the following boot setup command:

```
ncr53c8xx=specf:n
```

SCSI_NCR_IOMAPPED (default: not defined)
If defined, normal I/O is forced.

SCSI_NCR_SHARE_IRQ (default: defined)
If defined, request shared IRQ.

SCSI_NCR_MAX_TAGS (default: 8)
Maximum number of simultaneous tagged commands to a device.

Can be changed by ”settags <target> <maxtags>”

SCSI_NCR_SETUP_DEFAULT_SYNC (default: 50)
Transfer period factor the driver will use at boot time for synchronous negotiation. 0 means asynchronous.

Can be changed by ”setsync <target> <period factor>”

SCSI_NCR_SETUP_DEFAULT_TAGS (default: 8)
Default number of simultaneous tagged commands to a device.

< 1 means tagged command queuing disabled at start-up.

SCSI_NCR_ALWAYS_SIMPLE_TAG (default: defined)
Use SIMPLE TAG for read and write commands.

Can be changed by ”setorder <ordered|simple|default>”

SCSI_NCR_SETUP_DISCONNECTION (default: defined)
If defined, targets are allowed to disconnect.

SCSI_NCR_SETUP_FORCE_SYNC_NEGO (default: not defined)
If defined, synchronous negotiation is tried for all SCSI-2 devices.

Can be changed by ”setsync <target> <period>”

SCSI_NCR_SETUP_MASTER_PARITY (default: defined)
If defined, master parity checking is enabled.

SCSI_NCR_SETUP_SCSI_PARITY (default: defined)
If defined, SCSI parity checking is enabled.

SCSI_NCR_PROFILE_SUPPORT (default: not defined)
If defined, profiling information is gathered.

SCSI_NCR_MAX_SCATTER (default: 128)
Scatter list size of the driver ccb.

SCSI_NCR_MAX_TARGET (default: 16)
Max number of targets per host.

SCSI_NCR_MAX_HOST (default: 2)
Max number of host controllers.

SCSI_NCR_SETTLE_TIME (default: 2)
Number of seconds the driver will wait after reset.

SCSI_NCR_TIMEOUT_ALERT (default: 3)
If a pending command will time out after this amount of seconds, an ordered tag is used for the next command.

Avoids timeouts for unordered tagged commands.

SCSI_NCR_CAN_QUEUE (default: 7*SCSI_NCR_MAX_TAGS)
Max number of commands that can be queued to a host.

SCSI_NCR_CMD_PER_LUN (default: SCSI_NCR_MAX_TAGS)
Max number of commands queued to a host for a device.

SCSI_NCR_SG_TABLESIZE (default: SCSI_NCR_MAX_SCATTER-1)
Max size of the Linux scatter/gather list.

SCSI_NCR_MAX_LUN (default: 8)
Max number of LUNs per target.

4.21.12 12. Installation

This driver is part of the linux kernel distribution. Driver files are located in the sub-directory "drivers/scsi" of the kernel source tree.

Driver files:

```
README.ncr53c8xx        : this file
ChangeLog.ncr53c8xx     : change log
ncr53c8xx.h             : definitions
ncr53c8xx.c             : the driver code
```

New driver versions are made available separately in order to allow testing changes and new features prior to including them into the linux kernel distribution. The following URL provides information on latest available patches:

ftp://ftp.tux.org/pub/people/gerard-roudier/README

4.21.13 13. Architecture dependent features

<Not yet written>

4.21.14 14. Known problems

14.1 Tagged commands with Iomega Jaz device

I have not tried this device, however it has been reported to me the following: This device is capable of Tagged command queuing. However while spinning up, it rejects Tagged commands. This behaviour is conforms to 6.8.2 of SCSI-2 specifications. The current behaviour of the driver in that situation is not satisfying. So do not enable Tagged command queuing for devices that are able to spin down. The other problem that may appear is timeouts. The only way to avoid timeouts seems to edit linux/drivers/scsi/sd.c and to increase the current timeout values.

14.2 Device names change when another controller is added

When you add a new NCR53C8XX chip based controller to a system that already has one or more controllers of this family, it may happen that the order the driver registers them to the kernel causes problems due to device name changes. When at least one controller uses NvRAM, SDMS BIOS version 4 allows you to define the order the BIOS will scan the scsi boards. The driver attaches controllers according to BIOS information if NvRAM detect option is set.

If your controllers do not have NvRAM, you can:

- Ask the driver to probe chip ids in reverse order from the boot command line: ncr53c8xx=revprob:y
- Make appropriate changes in the fstab.
- Use the 'scsidev' tool from Eric Youngdale.

14.3 Using only 8 bit devices with a WIDE SCSI controller

When only 8 bit NARROW devices are connected to a 16 bit WIDE SCSI controller, you must ensure that lines of the wide part of the SCSI BUS are pulled-up. This can be achieved by ENABLING the WIDE TERMINATOR portion of the SCSI controller card.

The TYAN 1365 documentation revision 1.2 is not correct about such settings. (page 10, figure 3.3).

14.4 Possible data corruption during a Memory Write and Invalidate

This problem is described in SYMBIOS DEL 397, Part Number 69-039241, ITEM 4.

In some complex situations, 53C875 chips revision <= 3 may start a PCI Write and Invalidate Command at a not cache-line-aligned 4 DWORDS boundary. This is only possible when Cache Line Size is 8 DWORDS or greater. Pentium systems use a 8 DWORDS cache line size and so are concerned by this chip bug, unlike i486 systems that use a 4 DWORDS cache line size.

When this situation occurs, the chip may complete the Write and Invalidate command after having only filled part of the last cache line involved in the transfer, leaving to data corruption the remainder of this cache line.

Not using Write And Invalidate obviously gets rid of this chip bug, and so it is now the default setting of the driver. However, for people like me who want to enable this feature, I have added part of a work-around suggested by SYMBIOS. This work-around resets the addressing logic when the DATA IN phase is entered and so prevents the bug from being triggered for the first SCSI MOVE of the phase. This work-around should be enough according to the following:

The only driver internal data structure that is greater than 8 DWORDS and that is moved by the SCRIPTS processor is the 'CCB header' that contains the context of the SCSI transfer. This data structure is aligned on 8 DWORDS boundary (Pentium Cache Line Size), and so is immune to this chip bug, at least on Pentium systems.

But the conditions of this bug can be met when a SCSI read command is performed using a buffer that is 4 DWORDS but not cache-line aligned. This cannot happen under Linux when scatter/gather lists are used since they only refer to system buffers that are well aligned. So, a work around may only be needed under Linux when a scatter/gather list is not used and when the SCSI DATA IN phase is reentered after a phase mismatch.

4.21.15 15. SCSI problem troubleshooting

15.1 Problem tracking

Most SCSI problems are due to a non conformant SCSI bus or to buggy devices. If unfortunately you have SCSI problems, you can check the following things:

- SCSI bus cables
- terminations at both end of the SCSI chain
- linux syslog messages (some of them may help you)

If you do not find the source of problems, you can configure the driver with no features enabled.

- only asynchronous data transfers
- tagged commands disabled
- disconnections not allowed

Now, if your SCSI bus is ok, your system have every chance to work with this safe configuration but performances will not be optimal.

If it still fails, then you can send your problem description to appropriate mailing lists or newsgroups. Send me a copy in order to be sure I will receive it. Obviously, a bug in the driver code is possible.

My email address: Gerard Roudier <groudier@free.fr>

Allowing disconnections is important if you use several devices on your SCSI bus but often causes problems with buggy devices. Synchronous data transfers increases throughput of fast devices like hard disks. Good SCSI hard disks with a large cache gain advantage of tagged commands queuing.

Try to enable one feature at a time with control commands. For example:

```
echo "setsync all 25" >/proc/scsi/ncr53c8xx/0
```

Will enable fast synchronous data transfer negotiation for all targets.

```
echo "setflag 3" >/proc/scsi/ncr53c8xx/0
```

Will reset flags (no_disc) for target 3, and so will allow it to disconnect the SCSI Bus.

```
echo "settags 3 8" >/proc/scsi/ncr53c8xx/0
```

Will enable tagged command queuing for target 3 if that device supports it.

Once you have found the device and the feature that cause problems, just disable that feature for that device.

15.2 Understanding hardware error reports

When the driver detects an unexpected error condition, it may display a message of the following pattern:

```
sym53c876-0:1: ERROR (0:48) (1-21-65) (f/95) @ (script 7c0:19000000).
sym53c876-0: script cmd = 19000000
sym53c876-0: regdump: da 10 80 95 47 0f 01 07 75 01 81 21 80 01 09 00.
```

Some fields in such a message may help you understand the cause of the problem, as follows:

```
sym53c876-0:1: ERROR (0:48) (1-21-65) (f/95) @ (script 7c0:19000000).
............A.........B.C....D.E..F....G.H.......I.....J...K.......
```

Field A
: [target number.] SCSI ID of the device the controller was talking with at the moment the error occurs.

Field B
: [DSTAT io register (DMA STATUS)]

Bit 0x40	MDPE Master Data Parity Error Data parity error detected on the PCI BUS.
Bit 0x20	BF Bus Fault PCI bus fault condition detected
Bit 0x01	IID Illegal Instruction Detected Set by the chip when it detects an Illegal Instruction format on some condition that makes an instruction illegal.
Bit 0x80	DFE Dma Fifo Empty Pure status bit that does not indicate an error.

If the reported DSTAT value contains a combination of MDPE (0x40), BF (0x20), then the cause may be likely due to a PCI BUS problem.

Field C
: [SIST io register (SCSI Interrupt Status)]

Bit 0x08	SGE SCSI GROSS ERROR Indicates that the chip detected a severe error condition on the SCSI BUS that prevents the SCSI protocol from functioning properly.
Bit 0x04	UDC Unexpected Disconnection Indicates that the device released the SCSI BUS when the chip was not expecting this to happen. A device may behave so to indicate the SCSI initiator that an error condition not reportable using the SCSI protocol has occurred.
Bit 0x02	RST SCSI BUS Reset Generally SCSI targets do not reset the SCSI BUS, although any device on the BUS can reset it at any time.
Bit 0x01	PAR Parity SCSI parity error detected.

On a faulty SCSI BUS, any error condition among SGE (0x08), UDC (0x04) and PAR (0x01) may be detected by the chip. If your SCSI system sometimes encounters such error conditions, especially SCSI GROSS ERROR, then a SCSI BUS problem is likely the cause of these errors.

For fields D,E,F,G and H, you may look into the sym53c8xx_defs.h file that contains some minimal comments on IO register bits.

Field D
[SOCL Scsi Output Control Latch] This register reflects the state of the SCSI control lines the chip want to drive or compare against.

Field E
[SBCL Scsi Bus Control Lines] Actual value of control lines on the SCSI BUS.

Field F
[SBDL Scsi Bus Data Lines] Actual value of data lines on the SCSI BUS.

Field G
[SXFER SCSI Transfer] Contains the setting of the Synchronous Period for output and the current Synchronous offset (offset 0 means asynchronous).

Field H
[SCNTL3 Scsi Control Register 3] Contains the setting of timing values for both asynchronous and synchronous data transfers.

Understanding Fields I, J, K and dumps requires to have good knowledge of SCSI standards, chip cores functionnals and internal driver data structures. You are not required to decode and understand them, unless you want to help maintain the driver code.

4.21.16 16. Synchronous transfer negotiation tables

Tables below have been created by calling the routine the driver uses for synchronisation negotiation timing calculation and chip setting. The first table corresponds to Ultra chips 53875 and 53C860 with 80 MHz clock and 5 clock divisors. The second one has been calculated by setting the scsi clock to 40 Mhz and using 4 clock divisors and so applies to all NCR53C8XX chips in fast SCSI-2 mode.

Periods are in nano-seconds and speeds are in Mega-transfers per second. 1 Mega-transfers/second means 1 MB/s with 8 bits SCSI and 2 MB/s with Wide16 SCSI.

16.1 Synchronous timings for 53C895, 53C875 and 53C860 SCSI controllers

Negotiated			NCR settings
Factor	Period	Speed	Period
10	25	40.000	25
11	30.2	33.112	31.25
12	50	20.000	50
13	52	19.230	62
14	56	17.857	62
15	60	16.666	62
16	64	15.625	75
17	68	14.705	75
18	72	13.888	75
19	76	13.157	87
20	80	12.500	87
21	84	11.904	87
22	88	11.363	93
23	92	10.869	93
24	96	10.416	100
25	100	10.000	100
26	104	9.615	112
27	108	9.259	112
28	112	8.928	112
29	116	8.620	125
30	120	8.333	125
31	124	8.064	125
32	128	7.812	131
33	132	7.575	150
34	136	7.352	150
35	140	7.142	150
36	144	6.944	150
37	148	6.756	150
38	152	6.578	175
39	156	6.410	175
40	160	6.250	175
41	164	6.097	175
42	168	5.952	175
43	172	5.813	175
44	176	5.681	187
45	180	5.555	187
46	184	5.434	187
47	188	5.319	200
48	192	5.208	200
49	196	5.102	200

16.2 Synchronous timings for fast SCSI-2 53C8XX controllers

Negotiated			NCR settings	
Factor	Period	Speed	Period	Speed
25	100	10.000	100	10.000
26	104	9.615	125	8.000
27	108	9.259	125	8.000
28	112	8.928	125	8.000
29	116	8.620	125	8.000
30	120	8.333	125	8.000
31	124	8.064	125	8.000
32	128	7.812	131	7.619
33	132	7.575	150	6.666
34	136	7.352	150	6.666
35	140	7.142	150	6.666
36	144	6.944	150	6.666
37	148	6.756	150	6.666
38	152	6.578	175	5.714
39	156	6.410	175	5.714
40	160	6.250	175	5.714
41	164	6.097	175	5.714
42	168	5.952	175	5.714
43	172	5.813	175	5.714
44	176	5.681	187	5.333
45	180	5.555	187	5.333
46	184	5.434	187	5.333
47	188	5.319	200	5.000
48	192	5.208	200	5.000
49	196	5.102	200	5.000

4.21.17 17. Serial NVRAM

(added by Richard Waltham: dormouse@farsrobt.demon.co.uk)

17.1 Features

Enabling serial NVRAM support enables detection of the serial NVRAM included on Symbios and some Symbios compatible host adaptors, and Tekram boards. The serial NVRAM is used by Symbios and Tekram to hold set up parameters for the host adaptor and its attached drives.

The Symbios NVRAM also holds data on the boot order of host adaptors in a system with more than one host adaptor. This enables the order of scanning the cards for drives to be changed from the default used during host adaptor detection.

This can be done to a limited extent at the moment using ”reverse probe” but this only changes the order of detection of different types of cards. The NVRAM boot order settings can do this as well as change the order the same types of cards are scanned in, something ”reverse probe” cannot do.

Tekram boards using Symbios chips, DC390W/F/U, which have NVRAM are detected and this is used to distinguish between Symbios compatible and Tekram host adaptors. This is used

to disable the Symbios compatible ”diff” setting incorrectly set on Tekram boards if the CONFIG_SCSI_53C8XX_SYMBIOS_COMPAT configuration parameter is set enabling both Symbios and Tekram boards to be used together with the Symbios cards using all their features, including ”diff” support. (”led pin” support for Symbios compatible cards can remain enabled when using Tekram cards. It does nothing useful for Tekram host adaptors but does not cause problems either.)

17.2 Symbios NVRAM layout

typical data at NVRAM address 0x100 (53c810a NVRAM):

```
00 00
64 01
8e 0b

00 30 00 00 00 00 07 00 00 00 00 00 00 00 07 04 10 04 00 00

04 00 0f 00 00 10 00 50 00 00 01 00 00 62
04 00 03 00 00 10 00 58 00 00 01 00 00 63
04 00 01 00 00 10 00 48 00 00 01 00 00 61
00 00 00 00 00 00 00 00 00 00 00 00 00 00

0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

fe fe
00 00
00 00
```

NVRAM layout details

NVRAM Address	
0x000-0x0ff	not used
0x100-0x26f	initialised data
0x270-0x7ff	not used

general layout:

```
header  -   6 bytes,
data    - 356 bytes (checksum is byte sum of this data)
trailer -   6 bytes
          ---
total     368 bytes
```

data area layout:

```
controller set up  -  20 bytes
boot configuration -  56 bytes (4x14 bytes)
device set up      - 128 bytes (16x8 bytes)
unused (spare?)    - 152 bytes (19x8 bytes)
                     ---
total                356 bytes
```

header:

```
00 00   - ?? start marker
64 01   - byte count (lsb/msb excludes header/trailer)
8e 0b   - checksum (lsb/msb excludes header/trailer)
```

controller set up:

```
00 30 00 00 00 00 07 00 00 00 00 00 00 00 07 04 10 04 00 00
                |     |           |     |
                |     |           |      -- host ID
                |     |           |
```

```
                |     |            --Removable Media Support
                |     |               0x00 = none
                |     |               0x01 = Bootable Device
                |     |               0x02 = All with Media
                |     |
                |      --flag bits 2
                |        0x00000001= scan order hi->low
                |            (default 0x00 - scan low->hi)
                    --flag bits 1
                    0x00000001 scam enable
                    0x00000010 parity enable
                    0x00000100 verbose boot msgs
```

remaining bytes unknown - they do not appear to change in my current set up for any of the controllers.

default set up is identical for 53c810a and 53c875 NVRAM (Removable Media added Symbios BIOS version 4.09)

boot configuration

boot order set by order of the devices in this table:

```
04 00 0f 00 00 10 00 50 00 00 01 00 00 62 -- 1st controller
04 00 03 00 00 10 00 58 00 00 01 00 00 63    2nd controller
04 00 01 00 00 10 00 48 00 00 01 00 00 61    3rd controller
00 00 00 00 00 00 00 00 00 00 00 00 00 00    4th controller
    |  |  |  |     |         |     |  |
    |  |  |  |     |         |      ---- PCI io port adr
    |  |  |  |     |          --0x01 init/scan at boot time
    |  |  |  |      --PCI device/function number (0xdddddfff)
    |  |   ----- ?? PCI vendor ID (lsb/msb)
         ----PCI device ID (lsb/msb)

?? use of this data is a guess but seems reasonable
```

remaining bytes unknown - they do not appear to change in my current set up

default set up is identical for 53c810a and 53c875 NVRAM

device set up (up to 16 devices - includes controller):

```
0f 00 08 08 64 00 0a 00 - id 0
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

0f 00 08 08 64 00 0a 00
```

```
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00 - id 15
|     |  |  |     |  |
|     |  |  |      ----timeout (lsb/msb)
|     |  |   --synch period (0x?? 40 Mtrans/sec- fast 40) (probably 0x28)
|     |  |                  (0x30 20 Mtrans/sec- fast 20)
|     |  |                  (0x64 10 Mtrans/sec- fast )
|     |  |                  (0xc8  5 Mtrans/sec)
|     |  |                  (0x00  asynchronous)
|     |   -- ?? max sync offset (0x08 in NVRAM on 53c810a)
|     |                         (0x10 in NVRAM on 53c875)
|      --device bus width (0x08 narrow)
|                        (0x10 16 bit wide)
--flag bits
    0x00000001 - disconnect enabled
    0x00000010 - scan at boot time
    0x00000100 - scan luns
    0x00001000 - queue tags enabled
```

remaining bytes unknown - they do not appear to change in my current set up

?? use of this data is a guess but seems reasonable (but it could be max bus width)

default set up for 53c810a NVRAM default set up for 53c875 NVRAM

- bus width - 0x10
- sync offset ? - 0x10
- sync period - 0x30

?? spare device space (32 bit bus ??)

```
00 00 00 00 00 00 00 00  (19x8bytes)
.
.
00 00 00 00 00 00 00 00
```

default set up is identical for 53c810a and 53c875 NVRAM

trailer:

```
fe fe   - ? end marker ?
00 00
00 00
```

default set up is identical for 53c810a and 53c875 NVRAM

17.3 Tekram NVRAM layout

nvram 64x16 (1024 bit)

Drive settings:

```
Drive ID 0-15 (addr 0x0yyyy0 = device setup, yyyy = ID)
            (addr 0x0yyyy1 = 0x0000)

    x x x x  x x x x  x x x x  x x x x
             | | |       | |  | | | |
             | | |       | |  | | |  ----- parity check   0 - off
             | | |       | |  | | |                       1 - on
             | | |       | |  | | |
             | | |       | |  | |  ------- sync neg       0 - off
             | | |       | |  | |                         1 - on
             | | |       | |  | |
             | | |       | |  |  --------- disconnect     0 - off
             | | |       | |  |                           1 - on
             | | |       | |  |
             | | |       | |   ----------- start cmd      0 - off
             | | |       | |                             1 - on
             | | |       | |
             | | |       |  -------------- tagged cmds    0 - off
             | | |       |                               1 - on
             | | |       |
             | | |        ---------------- wide neg       0 - off
             | | |                                       1 - on
             | | |
                 --------------------------- sync rate      0 - 10.0 Mtrans/sec
                                                           1 -  8.0
                                                           2 -  6.6
                                                           3 -  5.7
                                                           4 -  5.0
                                                           5 -  4.0
                                                           6 -  3.0
                                                           7 -  2.0
                                                           7 -  2.0
                                                           8 - 20.0
                                                           9 - 16.7
                                                           a - 13.9
                                                           b - 11.9
```

Global settings

Host flags 0 (addr 0x100000, 32):

```
x x x x  x x x x  x x x x  x x x x
| | | |  | | | |           | | | |
| | | |  | | | |            ----------- host ID    0x00 - 0x0f
| | | |  | | | |
```

```
| | | |  | | |  ----------------------- support for    0 - off
| | | |  | | |                         > 2 drives     1 - on
| | | |  | | |
| | | |  | |  ------------------------- support drives 0 - off
| | | |  | |                           > 1Gbytes      1 - on
| | | |  | |
| | | |  |  --------------------------- bus reset on   0 - off
| | | |  |                               power on     1 - on
| | | |  |
| | | |   ----------------------------- active neg     0 - off
| | | |                                                1 - on
| | | |
| | |  ------------------------------- imm seek       0 - off
| | |                                                  1 - on
| | |
| |  --------------------------------- scan luns      0 - off
| |                                                    1 - on
| |
 ------------------------------------- removable      0 - disable
                                       as BIOS dev    1 - boot device
                                                      2 - all
```

Host flags 1 (addr 0x100001, 33):

```
x x x x  x x x x  x x x x  x x x x
           | | |             | | |
           | | |              --------- boot delay     0 -   3 sec
           | | |                                      1 -   5
           | | |                                      2 -  10
           | | |                                      3 -  20
           | | |                                      4 -  30
           | | |                                      5 -  60
           | | |                                      6 - 120
           | | |
            --------------------------- max tag cmds   0 -  2
                                                       1 -  4
                                                       2 -  8
                                                       3 - 16
                                                       4 - 32
```

Host flags 2 (addr 0x100010, 34):

```
x x x x  x x x x  x x x x  x x x x
                               |
                                ----- F2/F6 enable   0 - off ???
                                                     1 - on  ???
```

checksum (addr 0x111111)

checksum = 0x1234 - (sum addr 0-63)

default nvram data:

```
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000

0x0f07 0x0400 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0xfbbc
```

4.21.18 18. Support for Big Endian

The PCI local bus has been primarily designed for x86 architecture. As a consequence, PCI devices generally expect DWORDS using little endian byte ordering.

18.1 Big Endian CPU

In order to support NCR chips on a Big Endian architecture the driver has to perform byte reordering each time it is needed. This feature has been added to the driver by Cort <cort@cs.nmt.edu> and is available in driver version 2.5 and later ones. For the moment Big Endian support has only been tested on Linux/PPC (PowerPC).

18.2 NCR chip in Big Endian mode of operations

It can be read in SYMBIOS documentation that some chips support a special Big Endian mode, on paper: 53C815, 53C825A, 53C875, 53C875N, 53C895. This mode of operations is not software-selectable, but needs pin named BigLit to be pulled-up. Using this mode, most of byte reorderings should be avoided when the driver is running on a Big Endian CPU. Driver version 2.5 is also, in theory, ready for this feature.

4.22 WorkBiT NinjaSCSI-3/32Bi driver for Linux

4.22.1 1. Comment

This is Workbit corp.'s(http://www.workbit.co.jp/) NinjaSCSI-3 for Linux.

4.22.2 2. My Linux environment

Linux kernel
2.4.7 / 2.2.19

pcmcia-cs
3.1.27

gcc
gcc-2.95.4

PC card
I-O data PCSC-F (NinjaSCSI-3), I-O data CBSC-II in 16 bit mode (NinjaSCSI-32Bi)

SCSI device
I-O data CDPS-PX24 (CD-ROM drive), Media Intelligent MMO-640GT (Optical disk drive)

4.22.3 3. Install

(a) Check your PC card is true ”NinjaSCSI-3” card.

If you installed pcmcia-cs already, pcmcia reports your card as UNKNOWN card, and write [”WBT”, ”NinjaSCSI-3”, ”R1.0”] or some other string to your console or log file.

You can also use ”cardctl” program (this program is in pcmcia-cs source code) to get more info.

```
# cat /var/log/messages
...
Jan  2 03:45:06 lindberg cardmgr[78]: unsupported card in socket 1
Jan  2 03:45:06 lindberg cardmgr[78]:   product info: "WBT", "NinjaSCSI-3",
 ↪ "R1.0"
...
# cardctl ident
Socket 0:
  no product info available
Socket 1:
  product info: "IO DATA", "CBSC16       ", "1"
```

(b) Get the Linux kernel source, and extract it to /usr/src. Because the NinjaSCSI driver requires some SCSI header files in Linux kernel source, I recommend rebuilding your kernel; this eliminates some versioning problems.

```
$ cd /usr/src
$ tar -zxvf linux-x.x.x.tar.gz
$ cd linux
$ make config
...
```

(c) If you use this driver with Kernel 2.2, unpack pcmcia-cs in some directory and make & install. This driver requires the pcmcia-cs header file.

```
$ cd /usr/src
$ tar zxvf cs-pcmcia-cs-3.x.x.tar.gz
...
```

(d) Extract this driver's archive somewhere, and edit Makefile, then do make:

```
$ tar -zxvf nsp_cs-x.x.tar.gz
$ cd nsp_cs-x.x
$ emacs Makefile
...
$ make
```

(e) Copy nsp_cs.ko to suitable place, like /lib/modules/<Kernel version>/pcmcia/ .

(f) Add these lines to /etc/pcmcia/config .

If you use pcmcia-cs-3.1.8 or later, we can use ”nsp_cs.conf” file. So, you don't need to edit file. Just copy to /etc/pcmcia/ .

```
device "nsp_cs"
  class "scsi" module "nsp_cs"

card "WorkBit NinjaSCSI-3"
  version "WBT", "NinjaSCSI-3", "R1.0"
  bind "nsp_cs"

card "WorkBit NinjaSCSI-32Bi (16bit)"
  version "WORKBIT", "UltraNinja-16", "1"
  bind "nsp_cs"

# OEM
card "WorkBit NinjaSCSI-32Bi (16bit) / IO-DATA"
  version "IO DATA", "CBSC16       ", "1"
  bind "nsp_cs"

# OEM
card "WorkBit NinjaSCSI-32Bi (16bit) / KME-1"
  version "KME    ", "SCSI-CARD-001", "1"
  bind "nsp_cs"
card "WorkBit NinjaSCSI-32Bi (16bit) / KME-2"
  version "KME    ", "SCSI-CARD-002", "1"
  bind "nsp_cs"
card "WorkBit NinjaSCSI-32Bi (16bit) / KME-3"
  version "KME    ", "SCSI-CARD-003", "1"
  bind "nsp_cs"
card "WorkBit NinjaSCSI-32Bi (16bit) / KME-4"
  version "KME    ", "SCSI-CARD-004", "1"
  bind "nsp_cs"
```

(f) Start (or restart) pcmcia-cs:

```
# /etc/rc.d/rc.pcmcia start        (BSD style)
```

or:

```
# /etc/init.d/pcmcia start         (SYSV style)
```

4.22.4 4. History

See README.nin_cs .

4.22.5 5. Caution

If you eject card when doing some operation for your SCSI device or suspend your computer, you encount some *BAD* error like disk crash.

It works good when I using this driver right way. But I'm not guarantee your data. Please backup your data when you use this driver.

4.22.6 6. Known Bugs

In 2.4 kernel, you can't use 640MB Optical disk. This error comes from high level SCSI driver.

4.22.7 7. Testing

Please send me some reports(bug reports etc..) of this software. When you send report, please tell me these or more.

- card name
- kernel version
- your SCSI device name(hard drive, CD-ROM, etc...)

4.22.8 8. Copyright

See GPL.

2001/08/08 yokota@netlab.is.tsukuba.ac.jp <YOKOTA Hiroshi>

4.23 Terse where to get ZIP Drive help info

General Iomega ZIP drive page for Linux: http://web.archive.org/web/%2E/http://www.torque.net/~campbell/

Driver archive for old drivers: http://web.archive.org/web/%2E/http://www.torque.net/~campbell/ppa

Linux Parport page (parallel port) http://web.archive.org/web/%2E/http://www.torque.net/parport/

Email list for Linux Parport linux-parport@torque.net

4.24 Qlogic FASXXX Family Driver Notes

This driver supports the Qlogic FASXXX family of chips. This driver only works with the ISA, VLB, and PCMCIA versions of the Qlogic FastSCSI! cards as well as any other card based on the FASXX chip (including the Control Concepts SCSI/IDE/SIO/PIO/FDC cards).

This driver does NOT support the PCI version. Support for these PCI Qlogic boards:

- IQ-PCI
- IQ-PCI-10
- IQ-PCI-D

is provided by the qla1280 driver.

Nor does it support the PCI-Basic, which is supported by the 'am53c974' driver.

4.24.1 PCMCIA Support

This currently only works if the card is enabled first from DOS. This means you will have to load your socket and card services, and QL41DOS.SYS and QL40ENBL.SYS. These are a minimum, but loading the rest of the modules won't interfere with the operation. The next thing to do is load the kernel without resetting the hardware, which can be a simple ctrl-alt-delete with a boot floppy, or by using loadlin with the kernel image accessible from DOS. If you are using the Linux PCMCIA driver, you will have to adjust it or otherwise stop it from configuring the card.

I am working with the PCMCIA group to make it more flexible, but that may take a while.

4.24.2 All Cards

The top of the qlogic.c file has a number of defines that controls configuration. As shipped, it provides a balance between speed and function. If there are any problems, try setting SLOW_CABLE to 1, and then try changing USE_IRQ and TURBO_PDMA to zero. If you are familiar with SCSI, there are other settings which can tune the bus.

It may be a good idea to enable RESET_AT_START, especially if the devices may not have been just powered up, or if you are restarting after a crash, since they may be busy trying to complete the last command or something. It comes up faster if this is set to zero, and if you have reliable hardware and connections it may be more useful to not reset things.

4.24.3 Some Troubleshooting Tips

Make sure it works properly under DOS. You should also do an initial FDISK on a new drive if you want partitions.

Don't enable all the speedups first. If anything is wrong, they will make any problem worse.

4.24.4 Important

The best way to test if your cables, termination, etc. are good is to copy a very big file (e.g. a doublespace container file, or a very large executable or archive). It should be at least 5 megabytes, but you can do multiple tests on smaller files. Then do a COMP to verify that the file copied properly. (Turn off all caching when doing these tests, otherwise you will test your RAM and not the files). Then do 10 COMPs, comparing the same file on the SCSI hard drive, i.e. ”COMP realbig.doc realbig.doc”. Then do it after the computer gets warm.

I noticed my system which seems to work 100% would fail this test if the computer was left on for a few hours. It was worse with longer cables, and more devices on the SCSI bus. What seems to happen is that it gets a false ACK causing an extra byte to be inserted into the stream (and this is not detected). This can be caused by bad termination (the ACK can be reflected), or by noise when the chips work less well because of the heat, or when cables get too long for the speed.

Remember, if it doesn't work under DOS, it probably won't work under Linux.

4.25 SCSI media changer driver

This is a driver for SCSI Medium Changer devices, which are listed with ”Type: Medium Changer” in /proc/scsi/scsi.

This is for *real* Jukeboxes. It is *not* supported to work with common small CD-ROM changers, neither one-lun-per-slot SCSI changers nor IDE drives.

Userland tools available from here:
http://linux.bytesex.org/misc/changer.html

4.25.1 General Information

First some words about how changers work: A changer has 2 (possibly more) SCSI ID's. One for the changer device which controls the robot, and one for the device which actually reads and writes the data. The later may be anything, a MOD, a CD-ROM, a tape or whatever. For the changer device this is a ”don't care”, he *only* shuffles around the media, nothing else.

The SCSI changer model is complex, compared to - for example - IDE-CD changers. But it allows to handle nearly all possible cases. It knows 4 different types of changer elements:

media transport	this one shuffles around the media, i.e. the transport arm. Also known as ”picker”.
storage	a slot which can hold a media.
import/export	the same as above, but is accessible from outside, i.e. there the operator (you !) can use this to fill in and remove media from the changer. Sometimes named ”mailslot”.
data transfer	this is the device which reads/writes, i.e. the CD-ROM / Tape / whatever drive.

None of these is limited to one: A huge Jukebox could have slots for 123 CD-ROM's, 5 CD-ROM readers (and therefore 6 SCSI ID's: the changer and each CD-ROM) and 2 transport arms. No problem to handle.

4.25.2 How it is implemented

I implemented the driver as character device driver with a NetBSD-like ioctl interface. Just grabbed NetBSD's header file and one of the other linux SCSI device drivers as starting point. The interface should be source code compatible with NetBSD. So if there is any software (anybody knows ???) which supports a BSDish changer driver, it should work with this driver too.

Over time a few more ioctls where added, volume tag support for example wasn't covered by the NetBSD ioctl API.

4.25.3 Current State

Support for more than one transport arm is not implemented yet (and nobody asked for it so far...).

I test and use the driver myself with a 35 slot cdrom jukebox from Grundig. I got some reports telling it works ok with tape autoloaders (Exabyte, HP and DEC). Some People use this driver with amanda. It works fine with small (11 slots) and a huge (4 MOs, 88 slots) magneto-optical Jukebox. Probably with lots of other changers too, most (but not all :-) people mail me only if it does *not* work...

I don't have any device lists, neither black-list nor white-list. Thus it is quite useless to ask me whenever a specific device is supported or not. In theory every changer device which supports the SCSI-2 media changer command set should work out-of-the-box with this driver. If it doesn't, it is a bug. Either within the driver or within the firmware of the changer device.

4.25.4 Using it

This is a character device with major number is 86, so use ”mknod /dev/sch0 c 86 0” to create the special file for the driver.

If the module finds the changer, it prints some messages about the device [try ”dmesg” if you don't see anything] and should show up in /proc/devices. If not.... some changers use ID ? / LUN 0 for the device and ID ? / LUN 1 for the robot mechanism. But Linux does *not* look for LUNs other than 0 as default, because there are too many broken devices. So you can try:

1) echo ”scsi add-single-device 0 0 ID 1” > /proc/scsi/scsi (replace ID with the SCSI-ID of the device)

2) boot the kernel with ”max_scsi_luns=1” on the command line (append=”max_scsi_luns=1” in lilo.conf should do the trick)

4.25.5 Trouble?

If you insmod the driver with ”insmod debug=1”, it will be verbose and prints a lot of stuff to the syslog. Compiling the kernel with CONFIG_SCSI_CONSTANTS=y improves the quality of the error messages a lot because the kernel will translate the error codes into human-readable strings then.

You can display these messages with the dmesg command (or check the logfiles). If you email me some question because of a problem with the driver, please include these messages.

4.25.6 Insmod options

debug=0/1
Enable debug messages (see above, default: 0).

verbose=0/1
Be verbose (default: 1).

init=0/1
Send INITIALIZE ELEMENT STATUS command to the changer at insmod time (default: 1).

timeout_init=<seconds>
timeout for the INITIALIZE ELEMENT STATUS command (default: 3600).

timeout_move=<seconds>
timeout for all other commands (default: 120).

dt_id=<id1>,<id2>,... / dt_lun=<lun1>,<lun2>,...
These two allow to specify the SCSI ID and LUN for the data transfer elements. You likely don't need this as the jukebox should provide this information. But some devices don't ...

vendor_firsts=, vendor_counts=, vendor_labels=
These insmod options can be used to tell the driver that there are some vendor-specific element types. Grundig for example does this. Some jukeboxes have a printer to label fresh burned CDs, which is addressed as element 0xc000 (type 5). To tell the driver about this vendor-specific element, use this:

```
$ insmod ch                     \
        vendor_firsts=0xc000    \
        vendor_counts=1         \
        vendor_labels=printer
```

All three insmod options accept up to four comma-separated values, this way you can configure the element types 5-8. You likely need the SCSI specs for the device in question to find the correct values as they are not covered by the SCSI-2 standard.

4.25.7 Credits

I wrote this driver using the famous mailing-patches-around-the-world method. With (more or less) help from:

- Daniel Moehwald <moehwald@hdg.de>
- Dane Jasper <dane@sonic.net>
- R. Scott Bailey <sbailey@dsddi.eds.com>
- Jonathan Corbet <corbet@lwn.net>

Special thanks go to

- Martin Kuehne <martin.kuehne@bnbt.de>

for a old, second-hand (but full functional) cdrom jukebox which I use to develop/test driver and tools now.

Have fun,

Gerd

Gerd Knorr <kraxel@bytesex.org>

4.26 SCSI FC Transport

Date: 11/18/2008

Kernel Revisions for features:

```
rports : <<TBS>>
vports : 2.6.22
bsg support : 2.6.30 (?TBD?)
```

4.26.1 Introduction

This file documents the features and components of the SCSI FC Transport. It also provides documents the API between the transport and FC LLDDs.

The FC transport can be found at:

```
drivers/scsi/scsi_transport_fc.c
include/scsi/scsi_transport_fc.h
include/scsi/scsi_netlink_fc.h
include/scsi/scsi_bsg_fc.h
```

This file is found at *SCSI FC Transport*

4.26.2 FC Remote Ports (rports)

<< To Be Supplied >>

4.26.3 FC Virtual Ports (vports)

Overview

New FC standards have defined mechanisms which allows for a single physical port to appear on as multiple communication ports. Using the N_Port Id Virtualization (NPIV) mechanism, a point-to-point connection to a Fabric can be assigned more than 1 N_Port_ID. Each N_Port_ID appears as a separate port to other endpoints on the fabric, even though it shares one physical link to the switch for communication. Each N_Port_ID can have a unique view of the fabric based on fabric zoning and array lun-masking (just like a normal non-NPIV adapter). Using the Virtual Fabric (VF) mechanism, adding a fabric header to each frame allows the port to interact with the Fabric Port to join multiple fabrics. The port will obtain an N_Port_ID on each fabric it joins. Each fabric will have its own unique view of endpoints and configuration parameters. NPIV may be used together with VF so that the port can obtain multiple N_Port_IDs on each virtual fabric.

The FC transport is now recognizing a new object - a vport. A vport is an entity that has a world-wide unique World Wide Port Name (wwpn) and World Wide Node Name (wwnn). The transport also allows for the FC4's to be specified for the vport, with FCP_Initiator being the primary role expected. Once instantiated by one of the above methods, it will have a distinct N_Port_ID and view of fabric endpoints and storage entities. The fc_host associated with the physical adapter will export the ability to create vports. The transport will create the vport object within the Linux device tree, and instruct the fc_host's driver to instantiate the virtual port. Typically, the driver will create a new scsi_host instance on the vport, resulting in a unique <H,C,T,L> namespace for the vport. Thus, whether a FC port is based on a physical port or on a virtual port, each will appear as a unique scsi_host with its own target and lun space.

Note: At this time, the transport is written to create only NPIV-based vports. However, consideration was given to VF-based vports and it should be a minor change to add support if needed. The remaining discussion will concentrate on NPIV.

Note: World Wide Name assignment (and uniqueness guarantees) are left up to an administrative entity controlling the vport. For example, if vports are to be associated with virtual machines, a XEN mgmt utility would be responsible for creating wwpn/wwnn's for the vport, using its own naming authority and OUI. (Note: it already does this for virtual MAC addresses).

Device Trees and Vport Objects:

Today, the device tree typically contains the scsi_host object, with rports and scsi target objects underneath it. Currently the FC transport creates the vport object and places it under the scsi_host object corresponding to the physical adapter. The LLDD will allocate a new scsi_host for the vport and link its object under the vport. The remainder of the tree under the vports scsi_host is the same as the non-NPIV case. The transport is written currently to easily allow the parent of the vport to be something other than the scsi_host. This could be used in the future to link the object onto a vm-specific device tree. If the vport's parent is not the physical port's scsi_host, a symbolic link to the vport object will be placed in the physical port's scsi_host.

Here's what to expect in the device tree :

The typical Physical Port's Scsi_Host:

```
/sys/devices/.../host17/
```

and it has the typical descendant tree:

```
/sys/devices/.../host17/rport-17:0-0/target17:0:0/17:0:0:0:
```

and then the vport is created on the Physical Port:

```
/sys/devices/.../host17/vport-17:0-0
```

and the vport's Scsi_Host is then created:

```
/sys/devices/.../host17/vport-17:0-0/host18
```

and then the rest of the tree progresses, such as:

```
/sys/devices/.../host17/vport-17:0-0/host18/rport-18:0-0/
↪target18:0:0/18:0:0:0:
```

Here's what to expect in the sysfs tree:

```
scsi_hosts:
  /sys/class/scsi_host/host17                physical port's scsi_host
  /sys/class/scsi_host/host18                vport's scsi_host
fc_hosts:
  /sys/class/fc_host/host17                  physical port's fc_host
  /sys/class/fc_host/host18                  vport's fc_host
fc_vports:
  /sys/class/fc_vports/vport-17:0-0          the vport's fc_vport
fc_rports:
  /sys/class/fc_remote_ports/rport-17:0-0    rport on the physical port
  /sys/class/fc_remote_ports/rport-18:0-0    rport on the vport
```

Vport Attributes

The new fc_vport class object has the following attributes

node_name: Read_Only
The WWNN of the vport

port_name: Read_Only
The WWPN of the vport

roles: Read_Only
Indicates the FC4 roles enabled on the vport.

symbolic_name: Read_Write
A string, appended to the driver's symbolic port name string, which is registered with the switch to identify the vport. For example, a hypervisor could set this string to "Xen Domain 2 VM 5 Vport 2", and this set of identifiers can be seen on switch management screens to identify the port.

vport_delete: Write_Only
When written with a "1", will tear down the vport.

vport_disable: Write_Only
When written with a "1", will transition the vport to a disabled. state. The vport will still be instantiated with the Linux kernel, but it will not be active on the FC link. When written with a "0", will enable the vport.

vport_last_state: Read_Only
Indicates the previous state of the vport. See the section below on "Vport States".

vport_state: Read_Only
Indicates the state of the vport. See the section below on "Vport States".

vport_type: Read_Only
Reflects the FC mechanism used to create the virtual port. Only NPIV is supported currently.

For the fc_host class object, the following attributes are added for vports:

max_npiv_vports: Read_Only
Indicates the maximum number of NPIV-based vports that the driver/adapter can support on the fc_host.

npiv_vports_inuse: Read_Only
Indicates how many NPIV-based vports have been instantiated on the fc_host.

vport_create: Write_Only
A "simple" create interface to instantiate a vport on an fc_host. A "<WWPN>:<WWNN>" string is written to the attribute. The transport then instantiates the vport object and calls the LLDD to create the vport with the role of FCP_Initiator. Each WWN is specified as 16 hex characters and may *not* contain any prefixes (e.g. 0x, x, etc).

vport_delete: Write_Only
A "simple" delete interface to teardown a vport. A

”<WWPN>:<WWNN>” string is written to the attribute. The transport will locate the vport on the fc_host with the same WWNs and tear it down. Each WWN is specified as 16 hex characters and may *not* contain any prefixes (e.g. 0x, x, etc).

Vport States

Vport instantiation consists of two parts:

- Creation with the kernel and LLDD. This means all transport and driver data structures are built up, and device objects created. This is equivalent to a driver ”attach” on an adapter, which is independent of the adapter's link state.
- Instantiation of the vport on the FC link via ELS traffic, etc. This is equivalent to a ”link up” and successful link initialization.

Further information can be found in the interfaces section below for Vport Creation.

Once a vport has been instantiated with the kernel/LLDD, a vport state can be reported via the sysfs attribute. The following states exist:

FC_VPORT_UNKNOWN - Unknown
An temporary state, typically set only while the vport is being instantiated with the kernel and LLDD.

FC_VPORT_ACTIVE - Active
The vport has been successfully been created on the FC link. It is fully functional.

FC_VPORT_DISABLED - Disabled
The vport instantiated, but ”disabled”. The vport is not instantiated on the FC link. This is equivalent to a physical port with the link ”down”.

FC_VPORT_LINKDOWN - Linkdown
The vport is not operational as the physical link is not operational.

FC_VPORT_INITIALIZING - Initializing
The vport is in the process of instantiating on the FC link. The LLDD will set this state just prior to starting the ELS traffic to create the vport. This state will persist until the vport is successfully created (state becomes FC_VPORT_ACTIVE) or it fails (state is one of the values below). As this state is transitory, it will not be preserved in the ”vport_last_state”.

FC_VPORT_NO_FABRIC_SUPP - No Fabric Support
The vport is not operational. One of the following conditions were encountered:

- The FC topology is not Point-to-Point
- The FC port is not connected to an F_Port
- The F_Port has indicated that NPIV is not supported.

FC_VPORT_NO_FABRIC_RSCS - No Fabric Resources
The vport is not operational. The Fabric failed FDISC with a status indicating that it does not have sufficient resources to complete the operation.

FC_VPORT_FABRIC_LOGOUT - Fabric Logout
The vport is not operational. The Fabric has LOGO'd the N_Port_ID associated with the vport.

FC_VPORT_FABRIC_REJ_WWN - Fabric Rejected WWN
The vport is not operational. The Fabric failed FDISC with a status indicating that the WWN's are not valid.

FC_VPORT_FAILED - VPort Failed
The vport is not operational. This is a catchall for all other error conditions.

The following state table indicates the different state transitions:

State	Event	New State
n/a	Initialization	Unknown
Unknown:	Link Down	Linkdown
	Link Up & Loop	No Fabric Support
	Link Up & no Fabric	No Fabric Support
	Link Up & FLOGI response indicates no NPIV support	No Fabric Support
	Link Up & FDISC being sent	Initializing
	Disable request	Disable
Linkdown:	Link Up	Unknown
Initializing:	FDISC ACC	Active
	FDISC LS_RJT w/ no resources	No Fabric Resources
	FDISC LS_RJT w/ invalid pname or invalid nport_id	Fabric Rejected WWN
	FDISC LS_RJT failed for other reasons	Vport Failed
	Link Down	Linkdown
	Disable request	Disable
Disable:	Enable request	Unknown
Active:	LOGO received from fabric	Fabric Logout
	Link Down	Linkdown
	Disable request	Disable
Fabric Logout:	Link still up	Unknown

The following 4 error states all have the same transitions:

```
No Fabric Support:
No Fabric Resources:
Fabric Rejected WWN:
Vport Failed:
                    Disable request                 Disable
                    Link goes down                  Linkdown
```

Transport <-> LLDD Interfaces

Vport support by LLDD:

The LLDD indicates support for vports by supplying a vport_create() function in the transport template. The presence of this function will cause the creation of the new attributes on the fc_host. As part of the physical port completing its initialization relative to the transport, it should set the max_npiv_vports attribute to indicate the maximum number of vports the driver and/or adapter supports.

Vport Creation:

The LLDD vport_create() syntax is:

```
int vport_create(struct fc_vport *vport, bool disable)
```

where:

vport	Is the newly allocated vport object
disable	If ”true”, the vport is to be created in a disabled stated. If ”false”, the vport is to be enabled upon creation.

When a request is made to create a new vport (via sgio/netlink, or the vport_create fc_host attribute), the transport will validate that the LLDD can support another vport (e.g. max_npiv_vports > npiv_vports_inuse). If not, the create request will be failed. If space remains, the transport will increment the vport count, create the vport object, and then call the LLDD's vport_create() function with the newly allocated vport object.

As mentioned above, vport creation is divided into two parts:

- Creation with the kernel and LLDD. This means all transport and driver data structures are built up, and device objects created. This is equivalent to a driver ”attach” on an adapter, which is independent of the adapter's link state.
- Instantiation of the vport on the FC link via ELS traffic, etc. This is equivalent to a ”link up” and successful link initialization.

The LLDD's vport_create() function will not synchronously wait for both parts to be fully completed before returning. It must validate that the infrastructure exists to support NPIV, and complete the first part of vport creation (data structure build up) before returning. We do not hinge vport_create() on the link-side operation mainly because:

- The link may be down. It is not a failure if it is. It simply means the vport is in an inoperable state until the link comes up. This is consistent with the link bouncing post vport creation.
- The vport may be created in a disabled state.
- This is consistent with a model where: the vport equates to a FC adapter. The vport_create is synonymous with driver attachment to the adapter, which is independent of link state.

Note: special error codes have been defined to delineate infrastructure failure cases for quicker resolution.

The expected behavior for the LLDD's vport_create() function is:

- Validate Infrastructure:
  - **If the driver or adapter cannot support another vport, whether** due to improper firmware, (a lie about) max_npiv, or a lack of some other resource - return VPCERR_UNSUPPORTED.
  - **If the driver validates the WWN's against those already active on** the adapter and detects an overlap - return VPCERR_BAD_WWN.
  - **If the driver detects the topology is loop, non-fabric, or the** FLOGI did not support NPIV - return VPCERR_NO_FABRIC_SUPP.
- **Allocate data structures. If errors are encountered, such as out** of memory conditions, return the respective negative Exxx error code.
- If the role is FCP Initiator, the LLDD is to :
  - Call scsi_host_alloc() to allocate a scsi_host for the vport.
  - Call scsi_add_host(new_shost, &vport->dev) to start the scsi_host and bind it as a child of the vport device.
  - Initializes the fc_host attribute values.
- **Kick of further vport state transitions based on the disable flag and** link state - and return success (zero).

LLDD Implementers Notes:

- It is suggested that there be a different fc_function_templates for the physical port and the virtual port. The physical port's template would have the vport_create, vport_delete, and vport_disable functions, while the vports would not.
- It is suggested that there be different scsi_host_templates for the physical port and virtual port. Likely, there are driver attributes, embedded into the scsi_host_template, that are applicable for the physical port only (link speed, topology setting, etc). This ensures that the attributes are applicable to the respective scsi_host.

Vport Disable/Enable:

The LLDD vport_disable() syntax is:

```
int vport_disable(struct fc_vport *vport, bool disable)
```

where:

vport	Is vport to be enabled or disabled
disable	If ”true”, the vport is to be disabled. If ”false”, the vport is to be enabled.

When a request is made to change the disabled state on a vport, the transport will validate the request against the existing vport state. If the request is to disable and the vport is already disabled, the request will fail. Similarly, if the request is to enable, and the vport is not in a disabled state, the request will fail. If the request is valid for the vport state, the transport will call the LLDD to change the vport's state.

Within the LLDD, if a vport is disabled, it remains instantiated with the kernel and LLDD, but it is not active or visible on the FC link in any way. (see Vport Creation and the 2 part instantiation discussion). The vport will remain in this state until it is deleted or re-enabled. When enabling a vport, the LLDD reinstantiates the vport on the FC link - essentially restarting the LLDD statemachine (see Vport States above).

Vport Deletion:

The LLDD vport_delete() syntax is:

```
int vport_delete(struct fc_vport *vport)
```

where:

vport: Is vport to delete

When a request is made to delete a vport (via sgio/netlink, or via the fc_host or fc_vport vport_delete attributes), the transport will call the LLDD to terminate the vport on the FC link, and teardown all other datastructures and references. If the LLDD completes successfully, the transport will teardown the vport objects and complete the vport removal. If the LLDD delete request fails, the vport object will remain, but will be in an indeterminate state.

Within the LLDD, the normal code paths for a scsi_host teardown should be followed. E.g. If the vport has a FCP Initiator role, the LLDD will call fc_remove_host() for the vports scsi_host, followed by scsi_remove_host() and scsi_host_put() for the vports scsi_host.

Other:

fc_host port_type attribute:
There is a new fc_host port_type value - FC_PORTTYPE_NPIV. This value must be set on all vport-based fc_hosts. Normally, on a physical port, the port_type attribute would be set to NPORT, NLPORT, etc based on the topology type and existence of the fabric. As this is not applicable to a vport, it makes more sense to report the FC mechanism used to create the vport.

Driver unload:
FC drivers are required to call fc_remove_host() prior to calling scsi_remove_host(). This allows the fc_host to tear down all remote ports prior the scsi_host being torn down. The fc_remove_host() call was updated to remove all vports for the fc_host as well.

Transport supplied functions

The following functions are supplied by the FC-transport for use by LLDs.

fc_vport_create	create a vport
fc_vport_terminate	detach and remove a vport

Details:

```
/**
* fc_vport_create - Admin App or LLDD requests creation of a vport
* @shost:     scsi host the virtual port is connected to.
* @ids:       The world wide names, FC4 port roles, etc for
*              the virtual port.
*
* Notes:
*     This routine assumes no locks are held on entry.
*/
struct fc_vport *
fc_vport_create(struct Scsi_Host *shost, struct fc_vport_identifiers *ids)

/**
* fc_vport_terminate - Admin App or LLDD requests termination of a vport
* @vport:      fc_vport to be terminated
*
* Calls the LLDD vport_delete() function, then deallocates and removes
* the vport from the shost and object tree.
*
* Notes:
*      This routine assumes no locks are held on entry.
*/
int
fc_vport_terminate(struct fc_vport *vport)
```

4.26.4 FC BSG support (CT & ELS passthru, and more)

<< To Be Supplied >>

4.26.5 Credits

The following people have contributed to this document:

James Smart james.smart@broadcom.com

4.27 SCSI Generic (sg) driver

20020126

4.27.1 Introduction

The SCSI Generic driver (sg) is one of the four ”high level” SCSI device drivers along with sd, st and sr (disk, tape and CD-ROM respectively). Sg is more generalized (but lower level) than its siblings and tends to be used on SCSI devices that don't fit into the already serviced categories. Thus sg is used for scanners, CD writers and reading audio CDs digitally amongst other things.

Rather than document the driver's interface here, version information is provided plus pointers (i.e. URLs) where to find documentation and examples.

4.27.2 Major versions of the sg driver

There are three major versions of sg found in the Linux kernel (lk):

- sg version 1 (original) from 1992 to early 1999 (lk 2.2.5) . It is based in the sg_header interface structure.
- sg version 2 from lk 2.2.6 in the 2.2 series. It is based on an extended version of the sg_header interface structure.
- sg version 3 found in the lk 2.4 series (and the lk 2.5 series). It adds the sg_io_hdr interface structure.

4.27.3 Sg driver documentation

The most recent documentation of the sg driver is kept at

- https://sg.danny.cz/sg/

This describes the sg version 3 driver found in the lk 2.4 series.

Documentation (large version) for the version 2 sg driver found in the lk 2.2 series can be found at

- https://sg.danny.cz/sg/p/scsi-generic_long.txt.

The original documentation for the sg driver (prior to lk 2.2.6) can be found in the LDP archives at

- https://tldp.org/HOWTO/archived/SCSI-Programming-HOWTO/index.html

A more general description of the Linux SCSI subsystem of which sg is a part can be found at https://www.tldp.org/HOWTO/SCSI-2.4-HOWTO .

4.27.4 Example code and utilities

There are two packages of sg utilities:

sg3_utils	for the sg version 3 driver found in lk 2.4
sg_utils	for the sg version 2 (and original) driver found in lk 2.2 and earlier

Both packages will work in the lk 2.4 series. However, sg3_utils offers more capabilities. They can be found at: https://sg.danny.cz/sg/sg3_utils.html and freecode.com

Another approach is to look at the applications that use the sg driver. These include cdrecord, cdparanoia, SANE and cdrdao.

4.27.5 Mapping of Linux kernel versions to sg driver versions

Here is a list of Linux kernels in the 2.4 series that had the new version of the sg driver:

- lk 2.4.0 : sg version 3.1.17
- lk 2.4.7 : sg version 3.1.19
- lk 2.4.10 : sg version 3.1.20[1]
- lk 2.4.17 : sg version 3.1.22

For reference here is a list of Linux kernels in the 2.2 series that had the new version of the sg driver:

- lk 2.2.0 : original sg version [with no version number]
- lk 2.2.6 : sg version 2.1.31
- lk 2.2.8 : sg version 2.1.32
- lk 2.2.10 : sg version 2.1.34 [SG_GET_VERSION_NUM ioctl first appeared]
- lk 2.2.14 : sg version 2.1.36
- lk 2.2.16 : sg version 2.1.38
- lk 2.2.17 : sg version 2.1.39
- lk 2.2.20 : sg version 2.1.40

The lk 2.5 development series currently contains sg version 3.5.23 which is functionally equivalent to sg version 3.1.22 found in lk 2.4.17.

Douglas Gilbert

26th January 2002

dgilbert@interlog.com

[1] There were 3 changes to sg version 3.1.20 by third parties in the next six Linux kernel versions.

4.28 Linux SCSI Disk Driver (sd) Parameters

4.28.1 cache_type (RW)

Enable/disable drive write & read cache.

cache_type string	WCE	RCD	Write cache	Read cache
write through	0	0	off	on
none	0	1	off	off
write back	1	0	on	on
write back, no read (daft)	1	1	on	off

To set cache type to ”write back” and save this setting to the drive:

```
# echo "write back" > cache_type
```

To modify the caching mode without making the change persistent, prepend ”temporary ” to the cache type string. E.g.:

```
# echo "temporary write back" > cache_type
```

4.29 SMARTPQI - Microchip Smart Storage SCSI driver

This file describes the smartpqi SCSI driver for Microchip (http://www.microchip.com) PQI controllers. The smartpqi driver is the next generation SCSI driver for Microchip Corp. The smartpqi driver is the first SCSI driver to implement the PQI queuing model.

The smartpqi driver will replace the aacraid driver for Adaptec Series 9 controllers. Customers running an older kernel (Pre-4.9) using an Adaptec Series 9 controller will have to configure the smartpqi driver or their volumes will not be added to the OS.

For Microchip smartpqi controller support, enable the smartpqi driver when configuring the kernel.

For more information on the PQI Queuing Interface, please see:

- http://www.t10.org/drafts.htm
- http://www.t10.org/members/w_pqi2.htm

4.29.1 Supported devices

<Controller names to be added as they become publicly available.>

4.29.2 smartpqi specific entries in /sys

smartpqi host attributes

- /sys/class/scsi_host/host*/rescan
- /sys/class/scsi_host/host*/driver_version

The host rescan attribute is a write only attribute. Writing to this attribute will trigger the driver to scan for new, changed, or removed devices and notify the SCSI mid-layer of any changes detected.

The version attribute is read-only and will return the driver version and the controller firmware version. For example:

```
driver: 0.9.13-370
firmware: 0.01-522
```

smartpqi sas device attributes

HBA devices are added to the SAS transport layer. These attributes are automatically added by the SAS transport layer.

/sys/class/sas_device/end_device-X:X/sas_address /sys/class/sas_device/end_device-X:X/enclosure_identifier /sys/class/sas_device/end_device-X:X/scsi_target_id

4.29.3 smartpqi specific ioctls

For compatibility with applications written for the cciss protocol.

CCISS_DEREGDISK, CCISS_REGNEWDISK, CCISS_REGNEWD
The above three ioctls all do exactly the same thing, which is to cause the driver to rescan for new devices. This does exactly the same thing as writing to the smartpqi specific host "rescan" attribute.

CCISS_GETPCIINFO
Returns PCI domain, bus, device and function and "board ID" (PCI subsystem ID).

CCISS_GETDRIVVER
Returns driver version in three bytes encoded as:

```
(DRIVER_MAJOR << 28) | (DRIVER_MINOR << 24) | (DRIVER_RELEASE <<␣
↪16) | DRIVER_REVISION;
```

CCISS_PASSTHRU
Allows "BMIC" and "CISS" commands to be passed through to the Smart Storage Array. These are used extensively by the SSA Array Configuration Utility, SNMP storage agents, etc.

4.30 The SCSI Tape Driver

This file contains brief information about the SCSI tape driver. The driver is currently maintained by Kai Mäkisara (email Kai.Makisara@kolumbus.fi)

Last modified: Tue Feb 9 21:54:16 2016 by kai.makisara

4.30.1 Basics

The driver is generic, i.e., it does not contain any code tailored to any specific tape drive. The tape parameters can be specified with one of the following three methods:

1. Each user can specify the tape parameters he/she wants to use directly with ioctls. This is administratively a very simple and flexible method and applicable to single-user workstations. However, in a multiuser environment the next user finds the tape parameters in state the previous user left them.

2. The system manager (root) can define default values for some tape parameters, like block size and density using the MTSETDRVBUFFER ioctl. These parameters can be programmed to come into effect either when a new tape is loaded into the drive or if writing begins at the beginning of the tape. The second method is applicable if the tape drive performs auto-detection of the tape format well (like some QIC-drives). The result is that any tape can be read, writing can be continued using existing format, and the default format is used if the tape is rewritten from the beginning (or a new tape is written for the first time). The first method is applicable if the drive does not perform auto-detection well enough and there is a single ”sensible” mode for the device. An example is a DAT drive that is used only in variable block mode (I don't know if this is sensible or not :-).

The user can override the parameters defined by the system manager. The changes persist until the defaults again come into effect.

3. By default, up to four modes can be defined and selected using the minor number (bits 5 and 6). The number of modes can be changed by changing ST_NBR_MODE_BITS in st.h. Mode 0 corresponds to the defaults discussed above. Additional modes are dormant until they are defined by the system manager (root). When specification of a new mode is started, the configuration of mode 0 is used to provide a starting point for definition of the new mode.

Using the modes allows the system manager to give the users choices over some of the buffering parameters not directly accessible to the users (buffered and asynchronous writes). The modes also allow choices between formats in multi-tape operations (the explicitly overridden parameters are reset when a new tape is loaded).

If more than one mode is used, all modes should contain definitions for the same set of parameters.

Many Unices contain internal tables that associate different modes to supported devices. The Linux SCSI tape driver does not contain such tables (and will not do that in future). Instead of that, a utility program can be made that fetches the inquiry data sent by the device, scans its database, and sets up the modes using the ioctls. Another alternative is to make a small script that uses mt to set the defaults tailored to the system.

The driver supports fixed and variable block size (within buffer limits). Both the auto-rewind (minor equals device number) and non-rewind devices (minor is 128 + device number) are implemented.

In variable block mode, the byte count in write() determines the size of the physical block on tape. When reading, the drive reads the next tape block and returns to the user the data if the read() byte count is at least the block size. Otherwise, error ENOMEM is returned.

In fixed block mode, the data transfer between the drive and the driver is in multiples of the block size. The write() byte count must be a multiple of the block size. This is not required when reading but may be advisable for portability.

Support is provided for changing the tape partition and partitioning of the tape with one or two partitions. By default support for partitioned tape is disabled for each driver and it can be enabled with the ioctl MTSETDRVBUFFER.

By default the driver writes one filemark when the device is closed after writing and the last operation has been a write. Two filemarks can be optionally written. In both cases end of data is signified by returning zero bytes for two consecutive reads.

Writing filemarks without the immediate bit set in the SCSI command block acts as a synchronization point, i.e., all remaining data form the drive buffers is written to tape before the command returns. This makes sure that write errors are caught at that point, but this takes time. In some applications, several consecutive files must be written fast. The MTWEOFI operation can be used to write the filemarks without flushing the drive buffer. Writing filemark at close() is always flushing the drive buffers. However, if the previous operation is MTWEOFI, close() does not write a filemark. This can be used if the program wants to close/open the tape device between files and wants to skip waiting.

If rewind, offline, bsf, or seek is done and previous tape operation was write, a filemark is written before moving tape.

The compile options are defined in the file linux/drivers/scsi/st_options.h.

4. If the open option O_NONBLOCK is used, open succeeds even if the drive is not ready. If O_NONBLOCK is not used, the driver waits for the drive to become ready. If this does not happen in ST_BLOCK_SECONDS seconds, open fails with the errno value EIO. With O_NONBLOCK the device can be opened for writing even if there is a write protected tape in the drive (commands trying to write something return error if attempted).

4.30.2 Minor Numbers

The tape driver currently supports up to 2^17 drives if 4 modes for each drive are used.

The minor numbers consist of the following bit fields:

```
dev_upper non-rew mode dev-lower
20 -  8     7    6 5  4      0
```

The non-rewind bit is always bit 7 (the uppermost bit in the lowermost byte). The bits defining the mode are below the non-rewind bit. The remaining bits define the tape device number. This numbering is backward compatible with the numbering used when the minor number was only 8 bits wide.

4.30.3 Sysfs Support

The driver creates the directory /sys/class/scsi_tape and populates it with directories corresponding to the existing tape devices. There are autorewind and non-rewind entries for each mode. The names are stxy and nstxy, where x is the tape number and y a character corresponding to the mode (none, l, m, a). For example, the directories for the first tape device are (assuming four modes): st0 nst0 st0l nst0l st0m nst0m st0a nst0a.

Each directory contains the entries: default_blksize default_compression default_density defined dev device driver. The file 'defined' contains 1 if the mode is defined and zero if not defined. The files 'default_*' contain the defaults set by the user. The value -1 means the default is not set. The file 'dev' contains the device numbers corresponding to this device. The links 'device' and 'driver' point to the SCSI device and driver entries.

Each directory also contains the entry 'options' which shows the currently enabled driver and mode options. The value in the file is a bit mask where the bit definitions are the same as those used with MTSETDRVBUFFER in setting the options.

A link named 'tape' is made from the SCSI device directory to the class directory corresponding to the mode 0 auto-rewind device (e.g., st0).

4.30.4 Sysfs and Statistics for Tape Devices

The st driver maintains statistics for tape drives inside the sysfs filesystem. The following method can be used to locate the statistics that are available (assuming that sysfs is mounted at /sys):

1. Use opendir(3) on the directory /sys/class/scsi_tape
2. Use readdir(3) to read the directory contents
3. Use regcomp(3)/regexec(3) to match directory entries to the extended regular expression "^st[0-9]+$"
4. Access the statistics from the /sys/class/scsi_tape/<match>/stats directory (where <match> is a directory entry from /sys/class/scsi_tape that matched the extended regular expression)

The reason for using this approach is that all the character devices pointing to the same tape drive use the same statistics. That means that st0 would have the same statistics as nst0.

The directory contains the following statistics files:

1. **in_flight**
   - The number of I/Os currently outstanding to this device.
2. **io_ns**
   - The amount of time spent waiting (in nanoseconds) for all I/O to complete (including read and write). This includes tape movement commands such as seeking between file or set marks and implicit tape movement such as when rewind on close tape devices are used.
3. **other_cnt**

- The number of I/Os issued to the tape drive other than read or write commands. The time taken to complete these commands uses the following calculation io_ms-read_ms-write_ms.

4. **read_byte_cnt**
   - The number of bytes read from the tape drive.
5. **read_cnt**
   - The number of read requests issued to the tape drive.
6. **read_ns**
   - The amount of time (in nanoseconds) spent waiting for read requests to complete.
7. **write_byte_cnt**
   - The number of bytes written to the tape drive.
8. **write_cnt**
   - The number of write requests issued to the tape drive.
9. **write_ns**
   - The amount of time (in nanoseconds) spent waiting for write requests to complete.
10. **resid_cnt**
    - The number of times during a read or write we found the residual amount to be non-zero. This should mean that a program is issuing a read larger thean the block size on tape. For write not all data made it to tape.

Note: The in_flight value is incremented when an I/O starts the I/O itself is not added to the statistics until it completes.

The total of read_cnt, write_cnt, and other_cnt may not total to the same value as iodone_cnt at the device level. The tape statistics only count I/O issued via the st module.

When read the statistics may not be temporally consistent while I/O is in progress. The individual values are read and written to atomically however when reading them back via sysfs they may be in the process of being updated when starting an I/O or when it is completed.

The value shown in in_flight is incremented before any statstics are updated and decremented when an I/O completes after updating statistics. The value of in_flight is 0 when there are no I/Os outstanding that are issued by the st driver. Tape statistics do not take into account any I/O performed via the sg device.

4.30.5 BSD and Sys V Semantics

The user can choose between these two behaviours of the tape driver by defining the value of the symbol ST_SYSV. The semantics differ when a file being read is closed. The BSD semantics leaves the tape where it currently is whereas the SYS V semantics moves the tape past the next filemark unless the filemark has just been crossed.

The default is BSD semantics.

4.30.6 Buffering

The driver tries to do transfers directly to/from user space. If this is not possible, a driver buffer allocated at run-time is used. If direct i/o is not possible for the whole transfer, the driver buffer is used (i.e., bounce buffers for individual pages are not used). Direct i/o can be impossible because of several reasons, e.g.:

- one or more pages are at addresses not reachable by the HBA
- the number of pages in the transfer exceeds the number of scatter/gather segments permitted by the HBA
- one or more pages can't be locked into memory (should not happen in any reasonable situation)

The size of the driver buffers is always at least one tape block. In fixed block mode, the minimum buffer size is defined (in 1024 byte units) by ST_FIXED_BUFFER_BLOCKS. With small block size this allows buffering of several blocks and using one SCSI read or write to transfer all of the blocks. Buffering of data across write calls in fixed block mode is allowed if ST_BUFFER_WRITES is non-zero and direct i/o is not used. Buffer allocation uses chunks of memory having sizes 2^n * (page size). Because of this the actual buffer size may be larger than the minimum allowable buffer size.

NOTE that if direct i/o is used, the small writes are not buffered. This may cause a surprise when moving from 2.4. There small writes (e.g., tar without -b option) may have had good throughput but this is not true any more with 2.6. Direct i/o can be turned off to solve this problem but a better solution is to use bigger write() byte counts (e.g., tar -b 64).

Asynchronous writing. Writing the buffer contents to the tape is started and the write call returns immediately. The status is checked at the next tape operation. Asynchronous writes are not done with direct i/o and not in fixed block mode.

Buffered writes and asynchronous writes may in some rare cases cause problems in multivolume operations if there is not enough space on the tape after the early-warning mark to flush the driver buffer.

Read ahead for fixed block mode (ST_READ_AHEAD). Filling the buffer is attempted even if the user does not want to get all of the data at this read command. Should be disabled for those drives that don't like a filemark to truncate a read request or that don't like backspacing.

Scatter/gather buffers (buffers that consist of chunks non-contiguous in the physical memory) are used if contiguous buffers can't be allocated. To support all SCSI adapters (including those not supporting scatter/gather), buffer allocation is using the following three kinds of chunks:

1. The initial segment that is used for all SCSI adapters including those not supporting scatter/gather. The size of this buffer will be (PAGE_SIZE << ST_FIRST_ORDER) bytes if the system can give a chunk of this size (and it is not larger than the buffer size specified

by ST_BUFFER_BLOCKS). If this size is not available, the driver halves the size and tries again until the size of one page. The default settings in st_options.h make the driver to try to allocate all of the buffer as one chunk.

2. The scatter/gather segments to fill the specified buffer size are allocated so that as many segments as possible are used but the number of segments does not exceed ST_FIRST_SG.

3. The remaining segments between ST_MAX_SG (or the module parameter max_sg_segs) and the number of segments used in phases 1 and 2 are used to extend the buffer at run-time if this is necessary. The number of scatter/gather segments allowed for the SCSI adapter is not exceeded if it is smaller than the maximum number of scatter/gather segments specified. If the maximum number allowed for the SCSI adapter is smaller than the number of segments used in phases 1 and 2, extending the buffer will always fail.

4.30.7 EOM Behaviour When Writing

When the end of medium early warning is encountered, the current write is finished and the number of bytes is returned. The next write returns -1 and errno is set to ENOSPC. To enable writing a trailer, the next write is allowed to proceed and, if successful, the number of bytes is returned. After this, -1 and the number of bytes are alternately returned until the physical end of medium (or some other error) is encountered.

4.30.8 Module Parameters

The buffer size, write threshold, and the maximum number of allocated buffers are configurable when the driver is loaded as a module. The keywords are:

buffer_kbs=xxx	the buffer size for fixed block mode is set to xxx kilobytes
write_threshold_kbs=xxx	the write threshold in kilobytes set to xxx
max_sg_segs=xxx	the maximum number of scatter/gather segments
try_direct_io=x	try direct transfer between user buffer and tape drive if this is non-zero

Note that if the buffer size is changed but the write threshold is not set, the write threshold is set to the new buffer size - 2 kB.

4.30.9 Boot Time Configuration

If the driver is compiled into the kernel, the same parameters can be also set using, e.g., the LILO command line. The preferred syntax is to use the same keyword used when loading as module but prepended with 'st.'. For instance, to set the maximum number of scatter/gather segments, the parameter 'st.max_sg_segs=xx' should be used (xx is the number of scatter/gather segments).

For compatibility, the old syntax from early 2.5 and 2.4 kernel versions is supported. The same keywords can be used as when loading the driver as module. If several parameters are set, the keyword-value pairs are separated with a comma (no spaces allowed). A colon can be used instead of the equal mark. The definition is prepended by the string st=. Here is an example:

```
st=buffer_kbs:64,write_threshold_kbs:60
```

The following syntax used by the old kernel versions is also supported:

```
st=aa[,bb[,dd]]
```

where:

- aa is the buffer size for fixed block mode in 1024 byte units
- bb is the write threshold in 1024 byte units
- dd is the maximum number of scatter/gather segments

4.30.10 IOCTLs

The tape is positioned and the drive parameters are set with ioctls defined in mtio.h The tape control program 'mt' uses these ioctls. Try to find an mt that supports all of the Linux SCSI tape ioctls and opens the device for writing if the tape contents will be modified (look for a package mt-st* from the Linux ftp sites; the GNU mt does not open for writing for, e.g., erase).

The supported ioctls are:

The following use the structure mtop:

MTFSF
Space forward over count filemarks. Tape positioned after filemark.

MTFSFM
As above but tape positioned before filemark.

MTBSF
Space backward over count filemarks. Tape positioned before filemark.

MTBSFM
As above but ape positioned after filemark.

MTFSR
Space forward over count records.

MTBSR
Space backward over count records.

MTFSS
Space forward over count setmarks.

MTBSS
Space backward over count setmarks.

MTWEOF
Write count filemarks.

MTWEOFI
Write count filemarks with immediate bit set (i.e., does not wait until data is on tape)

MTWSM
Write count setmarks.

MTREW
Rewind tape.

MTOFFL
Set device off line (often rewind plus eject).

MTNOP
Do nothing except flush the buffers.

MTRETEN
Re-tension tape.

MTEOM
Space to end of recorded data.

MTERASE
Erase tape. If the argument is zero, the short erase command is used. The long erase command is used with all other values of the argument.

MTSEEK
Seek to tape block count. Uses Tandberg-compatible seek (QFA) for SCSI-1 drives and SCSI-2 seek for SCSI-2 drives. The file and block numbers in the status are not valid after a seek.

MTSETBLK
Set the drive block size. Setting to zero sets the drive into variable block mode (if applicable).

MTSETDENSITY
Sets the drive density code to arg. See drive documentation for available codes.

MTLOCK and MTUNLOCK
Explicitly lock/unlock the tape drive door.

MTLOAD and MTUNLOAD
Explicitly load and unload the tape. If the command argument x is between MT_ST_HPLOADER_OFFSET + 1 and MT_ST_HPLOADER_OFFSET + 6, the number x is used sent to the drive with the command and it selects the tape slot to use of HP C1553A changer.

MTCOMPRESSION
Sets compressing or uncompressing drive mode using the SCSI mode page 15. Note that some drives other methods for control of compression. Some drives (like the Exabytes) use density codes for compression control. Some drives use another mode page but this page has not been implemented in the driver. Some drives without compression capability will accept any compression mode without error.

MTSETPART
Moves the tape to the partition given by the argument at the next tape operation. The block at which the tape is positioned is the block where the tape was previously positioned in the new active partition unless the next tape operation is MTSEEK. In this case the tape is moved directly to the block specified by MTSEEK. MTSETPART is inactive unless MT_ST_CAN_PARTITIONS set.

MTMKPART
Formats the tape with one partition (argument zero) or two partitions (argument non-zero). If the argument is positive, it specifies the size of partition 1 in megabytes. For DDS drives and several early drives this is the physically first partition of the tape. If the argument

is negative, its absolute value specifies the size of partition 0 in megabytes. This is the physically first partition of many later drives, like the LTO drives from LTO-5 upwards. The drive has to support partitions with size specified by the initiator. Inactive unless MT_ST_CAN_PARTITIONS set.

MTSETDRVBUFFER

Is used for several purposes. The command is obtained from count with mask MT_SET_OPTIONS, the low order bits are used as argument. This command is only allowed for the superuser (root). The subcommands are:

- **0**

  The drive buffer option is set to the argument. Zero means no buffering.

- **MT_ST_BOOLEANS**

  Sets the buffering options. The bits are the new states (enabled/disabled) the following options (in the parenthesis is specified whether the option is global or can be specified differently for each mode):

  MT_ST_BUFFER_WRITES
  write buffering (mode)

  MT_ST_ASYNC_WRITES
  asynchronous writes (mode)

  MT_ST_READ_AHEAD
  read ahead (mode)

  MT_ST_TWO_FM
  writing of two filemarks (global)

  MT_ST_FAST_EOM
  using the SCSI spacing to EOD (global)

  MT_ST_AUTO_LOCK
  automatic locking of the drive door (global)

  MT_ST_DEF_WRITES
  the defaults are meant only for writes (mode)

  MT_ST_CAN_BSR
  backspacing over more than one records can be used for repositioning the tape (global)

  MT_ST_NO_BLKLIMS
  the driver does not ask the block limits from the drive (block size can be changed only to variable) (global)

  MT_ST_CAN_PARTITIONS
  enables support for partitioned tapes (global)

  MT_ST_SCSI2LOGICAL
  the logical block number is used in the MTSEEK and MTIOCPOS for SCSI-2 drives instead of the device dependent address. It is recommended to set this flag unless there are tapes using the device dependent (from the old times) (global)

  MT_ST_SYSV
  sets the SYSV semantics (mode)

MT_ST_NOWAIT
enables immediate mode (i.e., don't wait for the command to finish) for some commands (e.g., rewind)

MT_ST_NOWAIT_EOF
enables immediate filemark mode (i.e. when writing a filemark, don't wait for it to complete). Please see the BASICS note about MTWEOFI with respect to the possible dangers of writing immediate filemarks.

MT_ST_SILI
enables setting the SILI bit in SCSI commands when reading in variable block mode to enhance performance when reading blocks shorter than the byte count; set this only if you are sure that the drive supports SILI and the HBA correctly returns transfer residuals

MT_ST_DEBUGGING
debugging (global; debugging must be compiled into the driver)

- **MT_ST_SETBOOLEANS, MT_ST_CLEARBOOLEANS**
  Sets or clears the option bits.
- **MT_ST_WRITE_THRESHOLD**
  Sets the write threshold for this device to kilobytes specified by the lowest bits.
- **MT_ST_DEF_BLKSIZE**
  Defines the default block size set automatically. Value 0xffffff means that the default is not used any more.
- **MT_ST_DEF_DENSITY, MT_ST_DEF_DRVBUFFER**
  Used to set or clear the density (8 bits), and drive buffer state (3 bits). If the value is MT_ST_CLEAR_DEFAULT (0xfffff) the default will not be used any more. Otherwise the lowermost bits of the value contain the new value of the parameter.
- **MT_ST_DEF_COMPRESSION**
  The compression default will not be used if the value of the lowermost byte is 0xff. Otherwise the lowermost bit contains the new default. If the bits 8-15 are set to a non-zero number, and this number is not 0xff, the number is used as the compression algorithm. The value MT_ST_CLEAR_DEFAULT can be used to clear the compression default.
- **MT_ST_SET_TIMEOUT**
  Set the normal timeout in seconds for this device. The default is 900 seconds (15 minutes). The timeout should be long enough for the retries done by the device while reading/writing.
- **MT_ST_SET_LONG_TIMEOUT**
  Set the long timeout that is used for operations that are known to take a long time. The default is 14000 seconds (3.9 hours). For erase this value is further multiplied by eight.
- **MT_ST_SET_CLN**
  Set the cleaning request interpretation parameters using the lowest 24 bits of the argument. The driver can set the generic status bit GMT_CLN if a cleaning request bit pattern is found from the extended sense data. Many drives set one or more bits in the extended sense data when the drive needs cleaning. The bits are device-dependent. The driver is given the number of the sense data byte (the lowest eight bits of the argument; must be >= 18 (values 1 - 17 reserved) and <= the maximum

requested sense data sixe), a mask to select the relevant bits (the bits 9-16), and the bit pattern (bits 17-23). If the bit pattern is zero, one or more bits under the mask indicate cleaning request. If the pattern is non-zero, the pattern must match the masked sense data byte.

(The cleaning bit is set if the additional sense code and qualifier 00h 17h are seen regardless of the setting of MT_ST_SET_CLN.)

The following ioctl uses the structure mtpos:

MTIOCPOS
Reads the current position from the drive. Uses Tandberg-compatible QFA for SCSI-1 drives and the SCSI-2 command for the SCSI-2 drives.

The following ioctl uses the structure mtget to return the status:

MTIOCGET
Returns some status information. The file number and block number within file are returned. The block is -1 when it can't be determined (e.g., after MTBSF). The drive type is either MTISSCSI1 or MTISSCSI2. The number of recovered errors since the previous status call is stored in the lower word of the field mt_erreg. The current block size and the density code are stored in the field mt_dsreg (shifts for the subfields are MT_ST_BLKSIZE_SHIFT and MT_ST_DENSITY_SHIFT). The GMT_xxx status bits reflect the drive status. GMT_DR_OPEN is set if there is no tape in the drive. GMT_EOD means either end of recorded data or end of tape. GMT_EOT means end of tape.

4.30.11 Miscellaneous Compile Options

The recovered write errors are considered fatal if ST_RECOVERED_WRITE_FATAL is defined.

The maximum number of tape devices is determined by the define ST_MAX_TAPES. If more tapes are detected at driver initialization, the maximum is adjusted accordingly.

Immediate return from tape positioning SCSI commands can be enabled by defining ST_NOWAIT. If this is defined, the user should take care that the next tape operation is not started before the previous one has finished. The drives and SCSI adapters should handle this condition gracefully, but some drive/adapter combinations are known to hang the SCSI bus in this case.

The MTEOM command is by default implemented as spacing over 32767 filemarks. With this method the file number in the status is correct. The user can request using direct spacing to EOD by setting ST_FAST_EOM 1 (or using the MT_ST_OPTIONS ioctl). In this case the file number will be invalid.

When using read ahead or buffered writes the position within the file may not be correct after the file is closed (correct position may require backspacing over more than one record). The correct position within file can be obtained if ST_IN_FILE_POS is defined at compile time or the MT_ST_CAN_BSR bit is set for the drive with an ioctl. (The driver always backs over a filemark crossed by read ahead if the user does not request data that far.)

4.30.12 Debugging Hints

Debugging code is now compiled in by default but debugging is turned off with the kernel module parameter debug_flag defaulting to 0. Debugging can still be switched on and off with an ioctl. To enable debug at module load time add debug_flag=1 to the module load options, the debugging output is not voluminous. Debugging can also be enabled and disabled by writing a '0' (disable) or '1' (enable) to the sysfs file /sys/bus/scsi/drivers/st/debug_flag.

If the tape seems to hang, I would be very interested to hear where the driver is waiting. With the command 'ps -l' you can see the state of the process using the tape. If the state is D, the process is waiting for something. The field WCHAN tells where the driver is waiting. If you have the current System.map in the correct place (in /boot for the procps I use) or have updated /etc/psdatabase (for kmem ps), ps writes the function name in the WCHAN field. If not, you have to look up the function from System.map.

Note also that the timeouts are very long compared to most other drivers. This means that the Linux driver may appear hung although the real reason is that the tape firmware has got confused.

4.31 The sym53c500_cs Driver

The sym53c500_cs driver originated as an add-on to David Hinds' pcmcia-cs package, and was written by Tom Corner (tcorner@via.at). A rewrite was long overdue, and the current version addresses the following concerns:

(1) extensive kernel changes between 2.4 and 2.6.

(2) deprecated PCMCIA support outside the kernel.

All the USE_BIOS code has been ripped out. It was never used, and could not have worked anyway. The USE_DMA code is likewise gone. Many thanks to YOKOTA Hiroshi (nsp_cs driver) and David Hinds (qlogic_cs driver) for the code fragments I shamelessly adapted for this work. Thanks also to Christoph Hellwig for his patient tutelage while I stumbled about.

The Symbios Logic 53c500 chip was used in the ”newer” (circa 1997) version of the New Media Bus Toaster PCMCIA SCSI controller. Presumably there are other products using this chip, but I've never laid eyes (much less hands) on one.

Through the years, there have been a number of downloads of the pcmcia-cs version of this driver, and I guess it worked for those users. It worked for Tom Corner, and it works for me. Your mileage will probably vary.

Bob Tracy (rct@frus.com)

4.32 SYM-2 driver

Written by Gerard Roudier <groudier@free.fr>

21 Rue Carnot

95170 DEUIL LA BARRE - FRANCE

Updated by Matthew Wilcox <matthew@wil.cx>

2004-10-09

4.32.1 1. Introduction

This driver supports the whole SYM53C8XX family of PCI-SCSI controllers. It also support the subset of LSI53C10XX PCI-SCSI controllers that are based on the SYM53C8XX SCRIPTS language.

It replaces the sym53c8xx+ncr53c8xx driver bundle and shares its core code with the FreeBSD SYM-2 driver. The 'glue' that allows this driver to work under Linux is contained in 2 files named sym_glue.h and sym_glue.c. Other drivers files are intended not to depend on the Operating System on which the driver is used.

The history of this driver can be summarized as follows:

1993: ncr driver written for 386bsd and FreeBSD by:

- Wolfgang Stanglmeier <wolf@cologne.de>
- Stefan Esser <se@mi.Uni-Koeln.de>

1996: port of the ncr driver to Linux-1.2.13 and rename it ncr53c8xx.

- Gerard Roudier

1998: new sym53c8xx driver for Linux based on LOAD/STORE instruction and that adds full support for the 896 but drops support for early NCR devices.

- Gerard Roudier

1999: port of the sym53c8xx driver to FreeBSD and support for the LSI53C1010 33 MHz and 66MHz Ultra-3 controllers. The new driver is named 'sym'.

- Gerard Roudier

2000: Add support for early NCR devices to FreeBSD 'sym' driver. Break the driver into several sources and separate the OS glue code from the core code that can be shared among different O/Ses. Write a glue code for Linux.

- Gerard Roudier

2004: Remove FreeBSD compatibility code. Remove support for versions of Linux before 2.6. Start using Linux facilities.

This README file addresses the Linux version of the driver. Under FreeBSD, the driver documentation is the sym.8 man page.

Information about new chips is available at LSILOGIC web server:

http://www.lsilogic.com/

SCSI standard documentations are available at T10 site:

http://www.t10.org/

Useful SCSI tools written by Eric Youngdale are part of most Linux distributions:

scsiinfo	command line tool
scsi-config	TCL/Tk tool using scsiinfo

4.32.2 2. Supported chips and SCSI features

The following features are supported for all chips:

- Synchronous negotiation
- Disconnection
- Tagged command queuing
- SCSI parity checking
- PCI Master parity checking

Other features depends on chip capabilities.

The driver notably uses optimized SCRIPTS for devices that support LOAD/STORE and handles PHASE MISMATCH from SCRIPTS for devices that support the corresponding feature.

The following table shows some characteristics of the chip family.

Chip	On board SDMS BIOS	Wide	SCSI std.	Max. sync	Load/store scripts	Hardware phase mismatch
810	N	N	FAST10	10 MB/s	N	N
810A	N	N	FAST10	10 MB/s	Y	N
815	Y	N	FAST10	10 MB/s	N	N
825	Y	Y	FAST10	20 MB/s	N	N
825A	Y	Y	FAST10	20 MB/s	Y	N
860	N	N	FAST20	20 MB/s	Y	N
875	Y	Y	FAST20	40 MB/s	Y	N
875A	Y	Y	FAST20	40 MB/s	Y	Y
876	Y	Y	FAST20	40 MB/s	Y	N
895	Y	Y	FAST40	80 MB/s	Y	N
895A	Y	Y	FAST40	80 MB/s	Y	Y
896	Y	Y	FAST40	80 MB/s	Y	Y
897	Y	Y	FAST40	80 MB/s	Y	Y
1510D	Y	Y	FAST40	80 MB/s	Y	Y
1010	Y	Y	FAST80	160 MB/s	Y	Y
1010_66[1]	Y	Y	FAST80	160 MB/s	Y	Y

Summary of other supported features:

Module
: allow to load the driver

[1] Chip supports 33MHz and 66MHz PCI bus clock.

Memory mapped I/O
increases performance

Control commands
write operations to the proc SCSI file system

Debugging information
written to syslog (expert only)

Serial NVRAM
Symbios and Tekram formats

- Scatter / gather
- Shared interrupt
- Boot setup commands

4.32.3 3. Advantages of this driver for newer chips.

3.1 Optimized SCSI SCRIPTS

All chips except the 810, 815 and 825, support new SCSI SCRIPTS instructions named LOAD and STORE that allow to move up to 1 DWORD from/to an IO register to/from memory much faster that the MOVE MEMORY instruction that is supported by the 53c7xx and 53c8xx family.

The LOAD/STORE instructions support absolute and DSA relative addressing modes. The SCSI SCRIPTS had been entirely rewritten using LOAD/STORE instead of MOVE MEMORY instructions.

Due to the lack of LOAD/STORE SCRIPTS instructions by earlier chips, this driver also incorporates a different SCRIPTS set based on MEMORY MOVE, in order to provide support for the entire SYM53C8XX chips family.

3.2 New features appeared with the SYM53C896

Newer chips (see above) allows handling of the phase mismatch context from SCRIPTS (avoids the phase mismatch interrupt that stops the SCSI processor until the C code has saved the context of the transfer).

The 896 and 1010 chips support 64 bit PCI transactions and addressing, while the 895A supports 32 bit PCI transactions and 64 bit addressing. The SCRIPTS processor of these chips is not true 64 bit, but uses segment registers for bit 32-63. Another interesting feature is that LOAD/STORE instructions that address the on-chip RAM (8k) remain internal to the chip.

4.32.4 4. Memory mapped I/O versus normal I/O

Memory mapped I/O has less latency than normal I/O and is the recommended way for doing IO with PCI devices. Memory mapped I/O seems to work fine on most hardware configurations, but some poorly designed chipsets may break this feature. A configuration option is provided for normal I/O to be used but the driver defaults to MMIO.

4.32.5 5. Tagged command queueing

Queuing more than 1 command at a time to a device allows it to perform optimizations based on actual head positions and its mechanical characteristics. This feature may also reduce average command latency. In order to really gain advantage of this feature, devices must have a reasonable cache size (No miracle is to be expected for a low-end hard disk with 128 KB or less).

Some known old SCSI devices do not properly support tagged command queuing. Generally, firmware revisions that fix this kind of problems are available at respective vendor web/ftp sites.

All I can say is that I never have had problem with tagged queuing using this driver and its predecessors. Hard disks that behaved correctly for me using tagged commands are the following:

- IBM S12 0662
- Conner 1080S
- Quantum Atlas I
- Quantum Atlas II
- Seagate Cheetah I
- Quantum Viking II
- IBM DRVS
- Quantum Atlas IV
- Seagate Cheetah II

If your controller has NVRAM, you can configure this feature per target from the user setup tool. The Tekram Setup program allows to tune the maximum number of queued commands up to 32. The Symbios Setup only allows to enable or disable this feature.

The maximum number of simultaneous tagged commands queued to a device is currently set to 16 by default. This value is suitable for most SCSI disks. With large SCSI disks (>= 2GB, cache >= 512KB, average seek time <= 10 ms), using a larger value may give better performances.

This driver supports up to 255 commands per device, and but using more than 64 is generally not worth-while, unless you are using a very large disk or disk arrays. It is noticeable that most of recent hard disks seem not to accept more than 64 simultaneous commands. So, using more than 64 queued commands is probably just resource wasting.

If your controller does not have NVRAM or if it is managed by the SDMS BIOS/SETUP, you can configure tagged queueing feature and device queue depths from the boot command-line. For example:

```
sym53c8xx=tags:4/t2t3q15-t4q7/t1u0q32
```

will set tagged commands queue depths as follow:

- target 2 all luns on controller 0 --> 15
- target 3 all luns on controller 0 --> 15
- target 4 all luns on controller 0 --> 7
- target 1 lun 0 on controller 1 --> 32
- all other target/lun --> 4

In some special conditions, some SCSI disk firmwares may return a QUEUE FULL status for a SCSI command. This behaviour is managed by the driver using the following heuristic:

- Each time a QUEUE FULL status is returned, tagged queue depth is reduced to the actual number of disconnected commands.
- Every 200 successfully completed SCSI commands, if allowed by the current limit, the maximum number of queueable commands is incremented.

Since QUEUE FULL status reception and handling is resource wasting, the driver notifies by default this problem to user by indicating the actual number of commands used and their status, as well as its decision on the device queue depth change. The heuristic used by the driver in handling QUEUE FULL ensures that the impact on performances is not too bad. You can get rid of the messages by setting verbose level to zero, as follow:

1st method:
boot your system using 'sym53c8xx=verb:0' option.

2nd method:
apply ”setverbose 0” control command to the proc fs entry corresponding to your controller after boot-up.

4.32.6 6. Parity checking

The driver supports SCSI parity checking and PCI bus master parity checking. These features must be enabled in order to ensure safe data transfers. Some flawed devices or mother boards may have problems with parity. The options to defeat parity checking have been removed from the driver.

4.32.7 7. Profiling information

This driver does not provide profiling information as did its predecessors. This feature was not this useful and added complexity to the code. As the driver code got more complex, I have decided to remove everything that didn't seem actually useful.

4.32.8 8. Control commands

Control commands can be sent to the driver with write operations to the proc SCSI file system. The generic command syntax is the following:

```
echo "<verb> <parameters>" >/proc/scsi/sym53c8xx/0
(assumes controller number is 0)
```

Using ”all” for ”<target>” parameter with the commands below will apply to all targets of the SCSI chain (except the controller).

Available commands:

8.1 Set minimum synchronous period factor

setsync <target> <period factor>

target
target number

period
minimum synchronous period. Maximum speed = 1000/(4*period factor) except for special cases below.

Specify a period of 0, to force asynchronous transfer mode.

- 9 means 12.5 nano-seconds synchronous period
- 10 means 25 nano-seconds synchronous period
- 11 means 30 nano-seconds synchronous period
- 12 means 50 nano-seconds synchronous period

8.2 Set wide size

setwide <target> <size>

target
target number

size
0=8 bits, 1=16bits

8.3 Set maximum number of concurrent tagged commands

settags <target> <tags>

target
target number

tags
number of concurrent tagged commands must not be greater than configured (default: 16)

8.4 Set debug mode

setdebug <list of debug flags>

Available debug flags:

alloc	print info about memory allocations (ccb, lcb)
queue	print info about insertions into the command start queue
result	print sense data on CHECK CONDITION status
scatter	print info about the scatter process
scripts	print info about the script binding process
tiny	print minimal debugging information
timing	print timing information of the NCR chip
nego	print information about SCSI negotiations
phase	print information on script interruptions

Use ”setdebug” with no argument to reset debug flags.

8.5 Set flag (no_disc)

setflag <target> <flag>

target
target number

For the moment, only one flag is available:

no_disc: not allow target to disconnect.

Do not specify any flag in order to reset the flag. For example:

setflag 4
will reset no_disc flag for target 4, so will allow it disconnections.

setflag all
will allow disconnection for all devices on the SCSI bus.

8.6 Set verbose level

setverbose #level

The driver default verbose level is 1. This command allows to change th driver verbose level after boot-up.

8.7 Reset all logical units of a target

resetdev <target>

target
target number

The driver will try to send a BUS DEVICE RESET message to the target.

8.8 Abort all tasks of all logical units of a target

cleardev <target>

target
target number

The driver will try to send a ABORT message to all the logical units of the target.

4.32.9 9. Configuration parameters

Under kernel configuration tools (make menuconfig, for example), it is possible to change some default driver configuration parameters. If the firmware of all your devices is perfect enough, all the features supported by the driver can be enabled at start-up. However, if only one has a flaw for some SCSI feature, you can disable the support by the driver of this feature at linux start-up and enable this feature after boot-up only for devices that support it safely.

Configuration parameters:

Use normal IO (default answer: n)
Answer ”y” if you suspect your mother board to not allow memory mapped I/O. May slow down performance a little.

Default tagged command queue depth (default answer: 16)
Entering 0 defaults to tagged commands not being used. This parameter can be specified from the boot command line.

Maximum number of queued commands (default answer: 32)
This option allows you to specify the maximum number of tagged commands that can be queued to a device. The maximum supported value is 255.

Synchronous transfers frequency (default answer: 80)
This option allows you to specify the frequency in MHz the driver will use at boot time for synchronous data transfer negotiations. 0 means ”asynchronous data transfers”.

4.32.10 10. Boot setup commands

10.1 Syntax

Setup commands can be passed to the driver either at boot time or as parameters to modprobe, as described in Documentation/admin-guide/kernel-parameters.rst

Example of boot setup command under lilo prompt:

```
lilo: linux root=/dev/sda2 sym53c8xx.cmd_per_lun=4 sym53c8xx.sync=10 sym53c8xx.
↪debug=0x200
```

- enable tagged commands, up to 4 tagged commands queued.
- set synchronous negotiation speed to 10 Mega-transfers / second.
- set DEBUG_NEGO flag.

The following command will install the driver module with the same options as above:

```
modprobe sym53c8xx cmd_per_lun=4 sync=10 debug=0x200
```

10.2 Available arguments

10.2.1 Default number of tagged commands

- cmd_per_lun=0 (or cmd_per_lun=1) tagged command queuing disabled
- cmd_per_lun=#tags (#tags > 1) tagged command queuing enabled

#tags will be truncated to the max queued commands configuration parameter.

10.2.2 Burst max

burst=0	burst disabled
burst=255	get burst length from initial IO register settings.
burst=#x	burst enabled (1<<#x burst transfers max) #x is an integer value which is log base 2 of the burst transfers max.

By default the driver uses the maximum value supported by the chip.

10.2.3 LED support

led=1	enable LED support
led=0	disable LED support

Do not enable LED support if your scsi board does not use SDMS BIOS. (See 'Configuration parameters')

10.2.4 Differential mode

diff=0	never set up diff mode
diff=1	set up diff mode if BIOS set it
diff=2	always set up diff mode
diff=3	set diff mode if GPIO3 is not set

10.2.5 IRQ mode

irqm=0	always open drain
irqm=1	same as initial settings (assumed BIOS settings)
irqm=2	always totem pole

10.2.6 Check SCSI BUS

buschk=<option bits>

Available option bits:

0x0	No check.
0x1	Check and do not attach the controller on error.
0x2	Check and just warn on error.

10.2.7 Suggest a default SCSI id for hosts

hostid=255	no id suggested.
hostid=#x	(0 < x < 7) x suggested for hosts SCSI id.

If a host SCSI id is available from the NVRAM, the driver will ignore any value suggested as boot option. Otherwise, if a suggested value different from 255 has been supplied, it will use it. Otherwise, it will try to deduce the value previously set in the hardware and use value 7 if the hardware value is zero.

10.2.8 Verbosity level

verb=0	minimal
verb=1	normal
verb=2	too much

10.2.9 Debug mode

<table>
<tr><td>debug=0</td><td>clear debug flags</td></tr>
<tr><td>debug=#x</td><td>set debug flags
#x is an integer value combining the following power-of-2 values:
<table><tr><td>DEBUG_ALLOC</td><td>0x1</td></tr><tr><td>DEBUG_PHASE</td><td>0x2</td></tr><tr><td>DEBUG_POLL</td><td>0x4</td></tr><tr><td>DEBUG_QUEUE</td><td>0x8</td></tr><tr><td>DEBUG_RESULT</td><td>0x10</td></tr><tr><td>DEBUG_SCATTER</td><td>0x20</td></tr><tr><td>DEBUG_SCRIPT</td><td>0x40</td></tr><tr><td>DEBUG_TINY</td><td>0x80</td></tr><tr><td>DEBUG_TIMING</td><td>0x100</td></tr><tr><td>DEBUG_NEGO</td><td>0x200</td></tr><tr><td>DEBUG_TAGS</td><td>0x400</td></tr><tr><td>DEBUG_FREEZE</td><td>0x800</td></tr><tr><td>DEBUG_RESTART</td><td>0x1000</td></tr></table></td></tr>
</table>

You can play safely with DEBUG_NEGO. However, some of these flags may generate bunches of syslog messages.

10.2.10 Settle delay

settle=n	delay for n seconds

After a bus reset, the driver will delay for n seconds before talking to any device on the bus. The default is 3 seconds and safe mode will default it to 10.

10.2.11 Serial NVRAM

Note: option not currently implemented.

---

nvram=n	do not look for serial NVRAM
nvram=y	test controllers for onboard serial NVRAM

(alternate binary form)

nvram=<bits options>

0x01	look for NVRAM (equivalent to nvram=y)
0x02	ignore NVRAM ”Synchronous negotiation” parameters for all devices
0x04	ignore NVRAM ”Wide negotiation” parameter for all devices
0x08	ignore NVRAM ”Scan at boot time” parameter for all devices
0x80	also attach controllers set to OFF in the NVRAM (sym53c8xx only)

10.2.12 Exclude a host from being attached

excl=<io_address>,...

Prevent host at a given io address from being attached. For example 'excl=0xb400,0xc000' indicate to the driver not to attach hosts at address 0xb400 and 0xc000.

10.3 Converting from old style options

Previously, the sym2 driver accepted arguments of the form:

```
sym53c8xx=tags:4,sync:10,debug:0x200
```

As a result of the new module parameters, this is no longer available. Most of the options have remained the same, but tags has become cmd_per_lun to reflect its different purposes. The sample above would be specified as:

```
modprobe sym53c8xx cmd_per_lun=4 sync=10 debug=0x200
```

or on the kernel boot line as:

```
sym53c8xx.cmd_per_lun=4 sym53c8xx.sync=10 sym53c8xx.debug=0x200
```

10.4 SCSI BUS checking boot option

When this option is set to a non-zero value, the driver checks SCSI lines logic state, 100 micro-seconds after having asserted the SCSI RESET line. The driver just reads SCSI lines and checks all lines read FALSE except RESET. Since SCSI devices shall release the BUS at most 800 nano-seconds after SCSI RESET has been asserted, any signal to TRUE may indicate a SCSI BUS problem. Unfortunately, the following common SCSI BUS problems are not detected:

- Only 1 terminator installed.
- Misplaced terminators.
- Bad quality terminators.

On the other hand, either bad cabling, broken devices, not conformant devices, ... may cause a SCSI signal to be wrong when the driver reads it.

4.32.11 15. SCSI problem troubleshooting

15.1 Problem tracking

Most SCSI problems are due to a non conformant SCSI bus or too buggy devices. If unfortunately you have SCSI problems, you can check the following things:

- SCSI bus cables
- terminations at both end of the SCSI chain
- linux syslog messages (some of them may help you)

If you do not find the source of problems, you can configure the driver or devices in the NVRAM with minimal features.

- only asynchronous data transfers
- tagged commands disabled
- disconnections not allowed

Now, if your SCSI bus is ok, your system has every chance to work with this safe configuration but performances will not be optimal.

If it still fails, then you can send your problem description to appropriate mailing lists or newsgroups. Send me a copy in order to be sure I will receive it. Obviously, a bug in the driver code is possible.

My current email address: Gerard Roudier <groudier@free.fr>

Allowing disconnections is important if you use several devices on your SCSI bus but often causes problems with buggy devices. Synchronous data transfers increases throughput of fast devices like hard disks. Good SCSI hard disks with a large cache gain advantage of tagged commands queuing.

15.2 Understanding hardware error reports

When the driver detects an unexpected error condition, it may display a message of the following pattern:

```
sym0:1: ERROR (0:48) (1-21-65) (f/95/0) @ (script 7c0:19000000).
sym0: script cmd = 19000000
sym0: regdump: da 10 80 95 47 0f 01 07 75 01 81 21 80 01 09 00.
```

Some fields in such a message may help you understand the cause of the problem, as follows:

```
sym0:1: ERROR (0:48) (1-21-65) (f/95/0) @ (script 7c0:19000000).
.....A.........B.C....D.E..F....G.H..I.......J.....K...L.......
```

Field A
[target number.] SCSI ID of the device the controller was talking with at the moment the error occurs.

Field B
[DSTAT io register (DMA STATUS)]

Bit 0x40	MDPE Master Data Parity Error Data parity error detected on the PCI BUS.
Bit 0x20	BF Bus Fault PCI bus fault condition detected
Bit 0x01	IID Illegal Instruction Detected Set by the chip when it detects an Illegal Instruction format on some condition that makes an instruction illegal.
Bit 0x80	DFE Dma Fifo Empty Pure status bit that does not indicate an error.

If the reported DSTAT value contains a combination of MDPE (0x40), BF (0x20), then the cause may be likely due to a PCI BUS problem.

Field C
[SIST io register (SCSI Interrupt Status)]

Bit 0x08	SGE SCSI GROSS ERROR Indicates that the chip detected a severe error condition on the SCSI BUS that prevents the SCSI protocol from functioning properly.
Bit 0x04	UDC Unexpected Disconnection Indicates that the device released the SCSI BUS when the chip was not expecting this to happen. A device may behave so to indicate the SCSI initiator that an error condition not reportable using the SCSI protocol has occurred.
Bit 0x02	RST SCSI BUS Reset Generally SCSI targets do not reset the SCSI BUS, although any device on the BUS can reset it at any time.
Bit 0x01	PAR Parity SCSI parity error detected.

On a faulty SCSI BUS, any error condition among SGE (0x08), UDC (0x04) and PAR (0x01) may be detected by the chip. If your SCSI system sometimes encounters such error conditions, especially SCSI GROSS ERROR, then a SCSI BUS problem is likely the cause of these errors.

For fields D,E,F,G and H, you may look into the sym53c8xx_defs.h file that contains some minimal comments on IO register bits.

Field D
[SOCL Scsi Output Control Latch] This register reflects the state of the SCSI control lines the chip want to drive or compare against.

Field E
[SBCL Scsi Bus Control Lines] Actual value of control lines on the SCSI BUS.

Field F
[SBDL Scsi Bus Data Lines] Actual value of data lines on the SCSI BUS.

Field G
[SXFER SCSI Transfer] Contains the setting of the Synchronous Period for output and the current Synchronous offset (offset 0 means asynchronous).

Field H
[SCNTL3 Scsi Control Register 3] Contains the setting of timing values for both asynchronous and synchronous data transfers.

Field I
[SCNTL4 Scsi Control Register 4] Only meaningful for 53C1010 Ultra3 controllers.

Understanding Fields J, K, L and dumps requires to have good knowledge of SCSI standards, chip cores functionnals and internal driver data structures. You are not required to decode and

understand them, unless you want to help maintain the driver code.

4.32.12 17. Serial NVRAM (added by Richard Waltham: dormouse@farsrobt.demon.co.uk)

17.1 Features

Enabling serial NVRAM support enables detection of the serial NVRAM included on Symbios and some Symbios compatible host adaptors, and Tekram boards. The serial NVRAM is used by Symbios and Tekram to hold set up parameters for the host adaptor and its attached drives.

The Symbios NVRAM also holds data on the boot order of host adaptors in a system with more than one host adaptor. This information is no longer used as it's fundamentally incompatible with the hotplug PCI model.

Tekram boards using Symbios chips, DC390W/F/U, which have NVRAM are detected and this is used to distinguish between Symbios compatible and Tekram host adaptors. This is used to disable the Symbios compatible ”diff” setting incorrectly set on Tekram boards if the CONFIG_SCSI_53C8XX_SYMBIOS_COMPAT configuration parameter is set enabling both Symbios and Tekram boards to be used together with the Symbios cards using all their features, including ”diff” support. (”led pin” support for Symbios compatible cards can remain enabled when using Tekram cards. It does nothing useful for Tekram host adaptors but does not cause problems either.)

The parameters the driver is able to get from the NVRAM depend on the data format used, as follow:

	Tekram format	Symbios format
General and host parameters		
• Boot order	N	Y
• Host SCSI ID	Y	Y
• SCSI parity checking	Y	Y
• Verbose boot messages	N	Y
SCSI devices parameters		
• Synchronous transfer speed	Y	Y
• Wide 16 / Narrow	Y	Y
• Tagged Command Queuing enabled	Y	Y
• Disconnections enabled	Y	Y
• Scan at boot time	N	Y

In order to speed up the system boot, for each device configured without the ”scan at boot time” option, the driver forces an error on the first TEST UNIT READY command received for this device.

17.2 Symbios NVRAM layout

typical data at NVRAM address 0x100 (53c810a NVRAM):

```
00 00
64 01
8e 0b

00 30 00 00 00 00 07 00 00 00 00 00 00 00 07 04 10 04 00 00

04 00 0f 00 00 10 00 50 00 00 01 00 00 62
04 00 03 00 00 10 00 58 00 00 01 00 00 63
04 00 01 00 00 10 00 48 00 00 01 00 00 61
00 00 00 00 00 00 00 00 00 00 00 00 00 00

0f 00 08 08 64 00 0a 00
```

```
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

fe fe
00 00
00 00
```

NVRAM layout details

NVRAM Address	
0x000-0x0ff	not used
0x100-0x26f	initialised data
0x270-0x7ff	not used

general layout:

```
header  -   6 bytes,
data    - 356 bytes (checksum is byte sum of this data)
trailer -   6 bytes
          ---
total     368 bytes
```

data area layout:

```
controller set up  -  20 bytes
boot configuration -  56 bytes (4x14 bytes)
device set up      - 128 bytes (16x8 bytes)
unused (spare?)    - 152 bytes (19x8 bytes)
                     ---
total                356 bytes
```

header:

```
00 00   - ?? start marker
64 01   - byte count (lsb/msb excludes header/trailer)
8e 0b   - checksum (lsb/msb excludes header/trailer)
```

controller set up:

```
00 30 00 00 00 00 07 00 00 00 00 00 00 00 07 04 10 04 00 00
                 |     |           |     |
                 |     |           |      -- host ID
                 |     |           |
                 |     |            --Removable Media Support
                 |     |               0x00 = none
                 |     |               0x01 = Bootable Device
                 |     |               0x02 = All with Media
                 |     |
                 |      --flag bits 2
                 |        0x00000001= scan order hi->low
                 |            (default 0x00 - scan low->hi)
                     --flag bits 1
                     0x00000001 scam enable
                     0x00000010 parity enable
                     0x00000100 verbose boot msgs
```

remaining bytes unknown - they do not appear to change in my current set up for any of the controllers.

default set up is identical for 53c810a and 53c875 NVRAM (Removable Media added Symbios BIOS version 4.09)

boot configuration

boot order set by order of the devices in this table:

```
04 00 0f 00 00 10 00 50 00 00 01 00 00 62 -- 1st controller
04 00 03 00 00 10 00 58 00 00 01 00 00 63    2nd controller
```

```
04 00 01 00 00 10 00 48 00 00 01 00 00 61    3rd controller
00 00 00 00 00 00 00 00 00 00 00 00 00 00    4th controller
    |  |  |  |     |        |     |  |
    |  |  |  |     |        |      ---- PCI io port adr
    |  |  |  |     |         --0x01 init/scan at boot time
    |  |  |  |      --PCI device/function number (0xdddddfff)
    |  |   ----- ?? PCI vendor ID (lsb/msb)
        ----PCI device ID (lsb/msb)
```

?? use of this data is a guess but seems reasonable

remaining bytes unknown - they do not appear to change in my current set up

default set up is identical for 53c810a and 53c875 NVRAM

device set up (up to 16 devices - includes controller):

```
0f 00 08 08 64 00 0a 00 - id 0
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00

0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00
0f 00 08 08 64 00 0a 00 - id 15
|     |  |  |     |  |
|     |  |  |      ----timeout (lsb/msb)
|     |  |   --synch period (0x?? 40 Mtrans/sec- fast 40) (probably 0x28)
|     |  |                  (0x30 20 Mtrans/sec- fast 20)
|     |  |                  (0x64 10 Mtrans/sec- fast )
|     |  |                  (0xc8  5 Mtrans/sec)
|     |  |                  (0x00  asynchronous)
|     |   -- ?? max sync offset (0x08 in NVRAM on 53c810a)
|     |                         (0x10 in NVRAM on 53c875)
|      --device bus width (0x08 narrow)
|                         (0x10 16 bit wide)
--flag bits
    0x00000001 - disconnect enabled
    0x00000010 - scan at boot time
    0x00000100 - scan luns
    0x00001000 - queue tags enabled
```

remaining bytes unknown - they do not appear to change in my current set up

?? use of this data is a guess but seems reasonable (but it could be max bus width)

default set up for 53c810a NVRAM default set up for 53c875 NVRAM

- bus width - 0x10
- sync offset ? - 0x10
- sync period - 0x30

?? spare device space (32 bit bus ??):

```
00 00 00 00 00 00 00 00  (19x8bytes)
.
.
00 00 00 00 00 00 00 00
```

default set up is identical for 53c810a and 53c875 NVRAM

trailer:

```
fe fe   - ? end marker ?
00 00
00 00
```

default set up is identical for 53c810a and 53c875 NVRAM

17.3 Tekram NVRAM layout

nvram 64x16 (1024 bit)

Drive settings:

```
Drive ID 0-15 (addr 0x0yyyy0 = device setup, yyyy = ID)
            (addr 0x0yyyy1 = 0x0000)

    x x x x  x x x x  x x x x  x x x x
             | | |       | |   | | | |
             | | |       | |   | | |  ----- parity check   0 - off
             | | |       | |   | | |                       1 - on
             | | |       | |   | | |
             | | |       | |   | |  ------- sync neg       0 - off
             | | |       | |   | |                         1 - on
             | | |       | |   | |
             | | |       | |   |  --------- disconnect     0 - off
             | | |       | |   |                           1 - on
             | | |       | |   |
             | | |       | |    ----------- start cmd      0 - off
             | | |       | |                               1 - on
             | | |       | |
             | | |       |  -------------- tagged cmds    0 - off
             | | |       |                               1 - on
             | | |       |
             | | |        ---------------- wide neg       0 - off
             | | |                                       1 - on
```

```
            | | |
                --------------------------- sync rate      0 - 10.0 Mtrans/sec
                                                           1 -  8.0
                                                           2 -  6.6
                                                           3 -  5.7
                                                           4 -  5.0
                                                           5 -  4.0
                                                           6 -  3.0
                                                           7 -  2.0
                                                           7 -  2.0
                                                           8 - 20.0
                                                           9 - 16.7
                                                           a - 13.9
                                                           b - 11.9
```

Global settings

Host flags 0 (addr 0x100000, 32):

```
x x x x  x x x x  x x x x  x x x x
| | | |  | | | |           | | | |
| | | |  | | | |            ----------- host ID    0x00 - 0x0f
| | | |  | | | |
| | | |  | | |  ----------------------- support for    0 - off
| | | |  | | |                          > 2 drives     1 - on
| | | |  | | |
| | | |  | |  ------------------------- support drives 0 - off
| | | |  | |                          > 1Gbytes      1 - on
| | | |  | |
| | | |  |  --------------------------- bus reset on   0 - off
| | | |  |                               power on     1 - on
| | | |  |
| | | |   ---------------------------- active neg     0 - off
| | | |                                               1 - on
| | | |
| | |  ------------------------------- imm seek       0 - off
| | |                                               1 - on
| | |
| |  --------------------------------- scan luns      0 - off
| |                                               1 - on
| |
 ------------------------------------- removable      0 - disable
                                       as BIOS dev    1 - boot device
                                                      2 - all
```

Host flags 1 (addr 0x100001, 33):

```
x x x x  x x x x  x x x x  x x x x
           | | |             | | |
           | | |              --------- boot delay     0 -   3 sec
           | | |                                     1 -   5
```

```
            | | |                                          2 -  10
            | | |                                          3 -  20
            | | |                                          4 -  30
            | | |                                          5 -  60
            | | |                                          6 - 120
            | | |
             --------------------------- max tag cmds   0 -  2
                                                          1 -  4
                                                          2 -  8
                                                          3 - 16
                                                          4 - 32
```

Host flags 2 (addr 0x100010, 34):

```
x x x x  x x x x  x x x x  x x x x
                                |
                                 ----- F2/F6 enable   0 - off ???
                                                      1 - on  ???
```

checksum (addr 0x111111)

checksum = 0x1234 - (sum addr 0-63)

---

default nvram data:

```
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000
0x0037 0x0000 0x0037 0x0000 0x0037 0x0000 0x0037 0x0000

0x0f07 0x0400 0x0001 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0xfbbc
```

4.33 tcm_qla2xxx Driver Notes

4.33.1 tcm_qla2xxx jam_host attribute

There is now a new module endpoint attribute called jam_host attribute:

```
jam_host: boolean=0/1
```

This attribute and accompanying code is only included if the Kconfig parameter TCM_QLA2XXX_DEBUG is set to Y

By default this jammer code and functionality is disabled

Use this attribute to control the discarding of SCSI commands to a selected host.

This may be useful for testing error handling and simulating slow drain and other fabric issues.

Setting a boolean of 1 for the jam_host attribute for a particular host will discard the commands for that host.

Reset back to 0 to stop the jamming.

Enable host 4 to be jammed:

```
echo 1 > /sys/kernel/config/target/qla2xxx/21:00:00:24:ff:27:8f:ae/tpgt_1/
↪attrib/jam_host
```

Disable jamming on host 4:

```
echo 0 > /sys/kernel/config/target/qla2xxx/21:00:00:24:ff:27:8f:ae/tpgt_1/
↪attrib/jam_host
```

4.34 Universal Flash Storage

4.34.1 1. Overview

Universal Flash Storage (UFS) is a storage specification for flash devices. It aims to provide a universal storage interface for both embedded and removable flash memory-based storage in mobile devices such as smart phones and tablet computers. The specification is defined by JEDEC Solid State Technology Association. UFS is based on the MIPI M-PHY physical layer standard. UFS uses MIPI M-PHY as the physical layer and MIPI Unipro as the link layer.

The main goals of UFS are to provide:

- Optimized performance:

  For UFS version 1.0 and 1.1 the target performance is as follows:

  - Support for Gear1 is mandatory (rate A: 1248Mbps, rate B: 1457.6Mbps)
  - Support for Gear2 is optional (rate A: 2496Mbps, rate B: 2915.2Mbps)

  Future version of the standard,

  - Gear3 (rate A: 4992Mbps, rate B: 5830.4Mbps)
- Low power consumption
- High random IOPs and low latency

4.34.2 2. UFS Architecture Overview

UFS has a layered communication architecture which is based on SCSI SAM-5 architectural model.

UFS communication architecture consists of the following layers.

2.1 Application Layer

The Application layer is composed of the UFS command set layer (UCS), Task Manager and Device manager. The UFS interface is designed to be protocol agnostic, however SCSI has been selected as a baseline protocol for versions 1.0 and 1.1 of the UFS protocol layer.

UFS supports a subset of SCSI commands defined by SPC-4 and SBC-3.

- **UCS:**
  It handles SCSI commands supported by UFS specification.
- **Task manager:**
  It handles task management functions defined by the UFS which are meant for command queue control.
- **Device manager:**
  It handles device level operations and device configuration operations. Device level operations mainly involve device power management operations and commands to Interconnect layers. Device level configurations involve handling of query requests which are used to modify and retrieve configuration information of the device.

2.2 UFS Transport Protocol (UTP) layer

The UTP layer provides services for the higher layers through Service Access Points. UTP defines 3 service access points for higher layers.

- UDM_SAP: Device manager service access point is exposed to device manager for device level operations. These device level operations are done through query requests.
- UTP_CMD_SAP: Command service access point is exposed to UFS command set layer (UCS) to transport commands.
- UTP_TM_SAP: Task management service access point is exposed to task manager to transport task management functions.

UTP transports messages through UFS protocol information unit (UPIU).

2.3 UFS Interconnect (UIC) Layer

UIC is the lowest layer of the UFS layered architecture. It handles the connection between UFS host and UFS device. UIC consists of MIPI UniPro and MIPI M-PHY. UIC provides 2 service access points to upper layer:

- UIC_SAP: To transport UPIU between UFS host and UFS device.
- UIO_SAP: To issue commands to Unipro layers.

4.34.3 3. UFSHCD Overview

The UFS host controller driver is based on the Linux SCSI Framework. UFSHCD is a low-level device driver which acts as an interface between the SCSI Midlayer and PCIe-based UFS host controllers.

The current UFSHCD implementation supports the following functionality:

3.1 UFS controller initialization

The initialization module brings the UFS host controller to active state and prepares the controller to transfer commands/responses between UFSHCD and UFS device.

3.2 UTP Transfer requests

Transfer request handling module of UFSHCD receives SCSI commands from the SCSI Midlayer, forms UPIUs and issues the UPIUs to the UFS Host controller. Also, the module decodes responses received from the UFS host controller in the form of UPIUs and intimates the SCSI Midlayer of the status of the command.

3.3 UFS error handling

Error handling module handles Host controller fatal errors, Device fatal errors and UIC interconnect layer-related errors.

3.4 SCSI Error handling

This is done through UFSHCD SCSI error handling routines registered with the SCSI Midlayer. Examples of some of the error handling commands issues by the SCSI Midlayer are Abort task, LUN reset and host reset. UFSHCD Routines to perform these tasks are registered with SCSI Midlayer through .eh_abort_handler, .eh_device_reset_handler and .eh_host_reset_handler.

In this version of UFSHCD, Query requests and power management functionality are not implemented.

4.34.4 4. BSG Support

This transport driver supports exchanging UFS protocol information units (UPIUs) with a UFS device. Typically, user space will allocate struct ufs_bsg_request and struct ufs_bsg_reply (see ufs_bsg.h) as request_upiu and reply_upiu respectively. Filling those UPIUs should be done in accordance with JEDEC spec UFS2.1 paragraph 10.7. *Caveat emptor*: The driver makes no further input validations and sends the UPIU to the device as it is. Open the bsg device in /dev/ufs-bsg and send SG_IO with the applicable sg_io_v4:

```
io_hdr_v4.guard = 'Q';
io_hdr_v4.protocol = BSG_PROTOCOL_SCSI;
io_hdr_v4.subprotocol = BSG_SUB_PROTOCOL_SCSI_TRANSPORT;
io_hdr_v4.response = (__u64)reply_upiu;
```

```
io_hdr_v4.max_response_len = reply_len;
io_hdr_v4.request_len = request_len;
io_hdr_v4.request = (__u64)request_upiu;
if (dir == SG_DXFER_TO_DEV) {
        io_hdr_v4.dout_xfer_len = (uint32_t)byte_cnt;
        io_hdr_v4.dout_xferp = (uintptr_t)(__u64)buff;
} else {
        io_hdr_v4.din_xfer_len = (uint32_t)byte_cnt;
        io_hdr_v4.din_xferp = (uintptr_t)(__u64)buff;
}
```

If you wish to read or write a descriptor, use the appropriate xferp of sg_io_v4.

The userspace tool that interacts with the ufs-bsg endpoint and uses its UPIU-based protocol is available at:

https://github.com/westerndigitalcorporation/ufs-tool

For more detailed information about the tool and its supported features, please see the tool's README.

UFS specifications can be found at:

- UFS - http://www.jedec.org/sites/default/files/docs/JESD220.pdf
- UFSHCI - http://www.jedec.org/sites/default/files/docs/JESD223.pdf

4.34.5 5. UFS Reference Clock Frequency configuration

Devicetree can define a clock named ”ref_clk” under the UFS controller node to specify the intended reference clock frequency for the UFS storage parts. ACPI-based system can specify the frequency using ACPI Device-Specific Data property named ”ref-clk-freq”. In both ways the value is interpreted as frequency in Hz and must match one of the values given in the UFS specification. UFS subsystem will attempt to read the value when executing common controller initialization. If the value is available, UFS subsystem will ensure the bRefClkFreq attribute of the UFS storage device is set accordingly and will modify it if there is a mismatch.

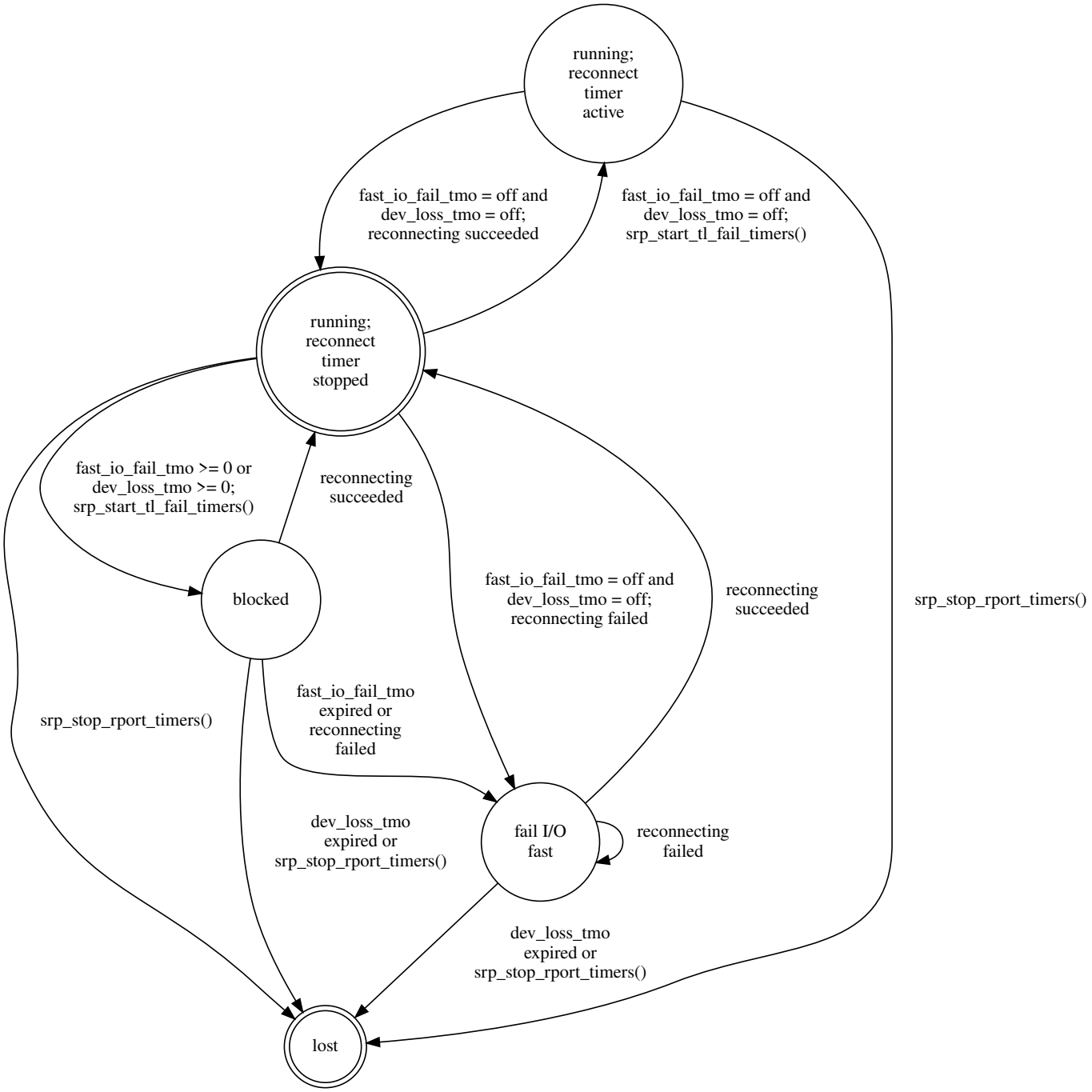
4.35 Driver for Western Digital WD7193, WD7197 and WD7296 SCSI cards

The card requires firmware that can be cut out of the Windows NT driver that can be downloaded from WD at: http://support.wdc.com/product/download.asp?groupid=801&sid=27&lang=en

There is no license anywhere in the file or on the page - so the firmware probably cannot be added to linux-firmware.

This script downloads and extracts the firmware, creating wd719x-risc.bin and d719x-wcs.bin files. Put them in /lib/firmware/:

```
#!/bin/sh
wget http://support.wdc.com/download/archive/pciscsi.exe
```

```
lha xi pciscsi.exe pci-scsi.exe
lha xi pci-scsi.exe nt/wd7296a.sys
rm pci-scsi.exe
dd if=wd7296a.sys of=wd719x-risc.bin bs=1 skip=5760 count=14336
dd if=wd7296a.sys of=wd719x-wcs.bin bs=1 skip=20096 count=514
rm wd7296a.sys
```

4.36 SCSI RDMA (SRP) transport class diagram