

libATA Developer's Guide

Jeff Garzik

libATA Developer's Guide

by Jeff Garzik

Copyright © 2003 Jeff Garzik

The contents of this file are subject to the Open Software License version 1.1 that can be found at <http://www.opensource.org/licenses/osl-1.1.txt> and is included herein by reference.

Alternatively, the contents of this file may be used under the terms of the GNU General Public License version 2 (the "GPL") as distributed in the kernel source COPYING file, in which case the provisions of the GPL are applicable instead of the above. If you wish to allow the use of your version of this file only under the terms of the GPL and not to allow others to use your version of this file under the OSL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the GPL. If you do not delete the provisions above, a recipient may use your version of this file under either the OSL or the GPL.

Table of Contents

1. Thanks.....	1
2. libata Driver API.....	3
2.1. struct ata_port_operations	3
3. libata Library	7
ata_tf_to_fis.....	7
ata_tf_from_fis	8
ata_dev_classify	8
ata_dev_id_string	9
ata_std_dev_select	11
ata_dev_config	11
ata_port_probe	12
__sata_phy_reset.....	13
sata_phy_reset.....	14
ata_port_disable	14
ata_bus_reset.....	15
ata_qc_prep	16
ata_eng_timeout.....	17
ata_qc_complete	18
ata_qc_issue_prot.....	18
ata_host_intr.....	19
ata_interrupt	21
ata_device_add.....	21
ata_scsi_release.....	22
ata_std_ports	23
ata_pci_init_one.....	24
ata_pci_remove_one	25
4. libata Core Internals.....	27
ata_tf_load_pio	27
ata_tf_load_mmio	27
ata_exec_command_pio.....	28
ata_exec_command_mmio.....	29
ata_tf_read_pio	30
ata_tf_read_mmio	31
ata_check_status_pio	32
ata_check_status_mmio	33
ata_prot_to_cmd	34
ata_dev_set_protocol	35
ata_mode_string.....	36
ata_pio_devchk	37
ata_mmio_devchk	38

ata_devchk	39
ata_dev_try_classify	40
ata_dev_select	41
ata_dump_id.....	42
ata_dev_identify.....	43
ata_bus_probe	44
ata_set_mode.....	45
ata_busy_sleep	46
ata_choose_xfer_mode	46
ata_dev_set_xfermode	47
ata_dev_init_params	48
ata_sg_clean.....	49
ata_fill_sg.....	49
ata_check_atapi_dma.....	50
ata_sg_setup_one	51
ata_sg_setup.....	52
ata_pio_poll.....	52
ata_pio_complete	53
ata_pio_block.....	54
ata_qc_timeout.....	54
ata_qc_new	55
ata_qc_new_init	56
ata_qc_free.....	57
ata_qc_issue	57
ata_bmdma_setup_mmio	58
ata_bmdma_start_mmio.....	59
ata_bmdma_setup_pio	60
ata_bmdma_start_pio.....	61
atapi_packet_task.....	61
ata_host_remove	62
ata_host_init.....	63
ata_host_add	64
ata_init.....	65
5. libata SCSI translation/emulation.....	67
ata_std_bios_param.....	67
ata_scsi_slave_config.....	68
ata_scsi_error	69
ata_scsi_queuecmd	70
ata_scsi_simulate	71
ata_cmd_ioctl.....	72
ata_task_ioctl	73
ata_scsi_qc_new	74
ata_to_sense_error	75

ata_scsi_flush_xlat	76
ata_scsi_verify_xlat	77
ata_scsi_rw_xlat.....	78
ata_scsi_translate	79
ata_scsi_rbuf_get	80
ata_scsi_rbuf_put	81
ata_scsi_rbuf_fill.....	82
ata_scsiop_inq_std	83
ata_scsiop_inq_00.....	84
ata_scsiop_inq_80.....	85
ata_scsiop_inq_83.....	86
ata_scsiop_noop.....	87
ata_msense_push.....	88
ata_msense_caching.....	90
ata_msense_ctl_mode	90
ata_msense_rw_recovery	91
ata_scsiop_mode_sense	92
ata_scsiop_read_cap	93
ata_scsiop_report_luns.....	94
ata_scsi_badcmd	95
atapi_xlat.....	97
ata_scsi_find_dev	97
ata_scsi_pass_thru.....	99
ata_get_xlat_func.....	99
ata_scsi_dump_cdb	100
6. ata_piix Internals	103
piix_pata_cbl_detect	103
piix_pata_phy_reset	103
piix_sata_probe.....	104
piix_sata_phy_reset.....	105
piix_set_piomode.....	106
piix_set_dmamode	107
piix_init_one	108
piix_init.....	109
piix_exit	110
7. sata_sil Internals	113
sil_dev_config.....	113

Chapter 1. Thanks

The bulk of the ATA knowledge comes thanks to long conversations with Andre Hedrick (www.linux-ide.org).

Thanks to Alan Cox for pointing out similarities between SATA and SCSI, and in general for motivation to hack on libata.

libata's device detection method, `ata_pio_devchk`, and in general all the early probing was based on extensive study of Hale Landis's probe/reset code in his ATADRVR driver (www.ata-atapi.com).

Chapter 2. libata Driver API

2.1. struct ata_port_operations

```
void (*port_disable) (struct ata_port *);
```

Called from ata_bus_probe() and ata_bus_reset() error paths, as well as when unregistering from the SCSI module (rmmod, hot unplug).

```
void (*dev_config) (struct ata_port *, struct ata_device *);
```

Called after IDENTIFY [PACKET] DEVICE is issued to each device found. Typically used to apply device-specific fixups prior to issue of SET FEATURES - XFER MODE, and prior to operation.

```
void (*set_piomode) (struct ata_port *, struct ata_device *);  
void (*set_dmamode) (struct ata_port *, struct ata_device *);  
void (*post_set_mode) (struct ata_port *ap);
```

Hooks called prior to the issue of SET FEATURES - XFER MODE command.

dev->pio_mode is guaranteed to be valid when ->set_piomode() is called, and

dev->dma_mode is guaranteed to be valid when ->set_dmamode() is called.

->post_set_mode() is called unconditionally, after the SET FEATURES - XFER MODE command completes successfully.

->set_piomode() is always called (if present), but ->set_dma_mode() is only called if DMA is possible.

```
void (*tf_load) (struct ata_port *ap, struct ata_taskfile *tf);  
void (*tf_read) (struct ata_port *ap, struct ata_taskfile *tf);
```

->tf_load() is called to load the given taskfile into hardware registers / DMA

buffers. ->tf_read() is called to read the hardware registers / DMA buffers, to obtain the current set of taskfile register values.

```
void (*exec_command) (struct ata_port *ap, struct ata_taskfile *tf);
```

causes an ATA command, previously loaded with ->tf_load(), to be initiated in hardware.

Chapter 2. libata Driver API

```
u8 (*check_status)(struct ata_port *ap);  
void (*dev_select)(struct ata_port *ap, unsigned int device);
```

Reads the Status ATA shadow register from hardware. On some hardware, this has the side effect of clearing the interrupt condition.

```
void (*dev_select)(struct ata_port *ap, unsigned int device);
```

Issues the low-level hardware command(s) that causes one of N hardware devices to be considered 'selected' (active and available for use) on the ATA bus.

```
void (*phy_reset)(struct ata_port *ap);
```

The very first step in the probe phase. Actions vary depending on the bus type, typically. After waking up the device and probing for device presence (PATA and SATA), typically a soft reset (SRST) will be performed. Drivers typically use the helper functions `ata_bus_reset()` or `sata_phy_reset()` for this hook.

```
void (*bmdma_setup)(struct ata_queued_cmd *qc);  
void (*bmdma_start)(struct ata_queued_cmd *qc);
```

When setting up an IDE BMDMA transaction, these hooks arm (`->bmdma_setup`) and fire (`->bmdma_start`) the hardware's DMA engine.

```
void (*qc_prep)(struct ata_queued_cmd *qc);  
int (*qc_issue)(struct ata_queued_cmd *qc);
```

Higher-level hooks, these two hooks can potentially supercede several of the above taskfile/DMA engine hooks. `->qc_prep` is called after the buffers have been DMA-mapped, and is typically used to populate the hardware's DMA scatter-gather table. Most drivers use the standard `ata_qc_prep()` helper function, but more advanced drivers roll their own.

`->qc_issue` is used to make a command active, once the hardware and S/G tables have been prepared. IDE BMDMA drivers use the helper function `ata_qc_issue_prot()` for taskfile protocol-based dispatch. More advanced drivers roll their own `->qc_issue` implementation, using this as the "issue new ATA command to hardware" hook.

```
void (*eng_timeout)(struct ata_port *ap);
```

This is a high level error handling function, called from the error handling thread, when a command times out.

```
irqreturn_t (*irq_handler)(int, void *, struct pt_regs *);  
void (*irq_clear) (struct ata_port *);
```

->irq_handler is the interrupt handling routine registered with the system, by libata.
->irq_clear is called during probe just before the interrupt handler is registered, to be sure hardware is quiet.

```
u32 (*scr_read) (struct ata_port *ap, unsigned int sc_reg);  
void (*scr_write) (struct ata_port *ap, unsigned int sc_reg,  
                  u32 val);
```

Read and write standard SATA phy registers. Currently only used if ->phy_reset hook called the sata_phy_reset() helper function.

```
int (*port_start) (struct ata_port *ap);  
void (*port_stop) (struct ata_port *ap);  
void (*host_stop) (struct ata_host_set *host_set);
```

->port_start() is called just after the data structures for each port are initialized. Typically this is used to alloc per-port DMA buffers / tables / rings, enable DMA engines, and similar tasks.

->host_stop() is called when the rmmod or hot unplug process begins. The hook must stop all hardware interrupts, DMA engines, etc.

->port_stop() is called after ->host_stop(). It's sole function is to release DMA/memory resources, now that they are no longer actively being used.

Chapter 3. libata Library

ata_tf_to_fis

Name

`ata_tf_to_fis` — Convert ATA taskfile to SATA FIS structure

Synopsis

```
void ata_tf_to_fis (struct ata_taskfile * tf, u8 * fis, u8  
pmp);
```

Arguments

tf

Taskfile to convert

fis

Buffer into which data will output

pmp

Port multiplier port

Description

Converts a standard ATA taskfile to a Serial ATA FIS structure (Register - Host to Device).

LOCKING

Inherited from caller.

ata_tf_from_fis

Name

`ata_tf_from_fis` — Convert SATA FIS to ATA taskfile

Synopsis

```
void ata_tf_from_fis (u8 * fis, struct ata_taskfile * tf);
```

Arguments

fis

Buffer from which data will be input

tf

Taskfile to output

Description

Converts a standard ATA taskfile to a Serial ATA FIS structure (Register - Host to Device).

LOCKING

Inherited from caller.

ata_dev_classify

Name

`ata_dev_classify` — determine device type based on ATA-spec signature

Synopsis

```
unsigned int ata_dev_classify (struct ata_taskfile * tf);
```

Arguments

tf

ATA taskfile register set for device to be identified

Description

Determine from taskfile register contents whether a device is ATA or ATAPI, as per “Signature and persistence” section of ATA/PI spec (volume 1, sect 5.14).

LOCKING

None.

RETURNS

Device type, `ATA_DEV_ATA`, `ATA_DEV_ATAPI`, or `ATA_DEV_UNKNOWN` the event of failure.

ata_dev_id_string

Name

`ata_dev_id_string` — Convert IDENTIFY DEVICE page into string

Synopsis

```
void ata_dev_id_string (u16 * id, unsigned char * s, unsigned  
int ofs, unsigned int len);
```

Arguments

id

IDENTIFY DEVICE results we will examine

s

string into which data is output

ofs

offset into identify device page

len

length of string to return. must be an even number.

Description

The strings in the IDENTIFY DEVICE page are broken up into 16-bit chunks. Run through the string, and output each 8-bit chunk linearly, regardless of platform.

LOCKING

caller.

ata_std_dev_select

Name

`ata_std_dev_select` — Select device 0/1 on ATA bus

Synopsis

```
void ata_std_dev_select (struct ata_port * ap, unsigned int  
device);
```

Arguments

ap

ATA channel to manipulate

device

ATA device (numbered from zero) to select

Description

Use the method defined in the ATA specification to make either device 0, or device 1, active on the ATA channel.

LOCKING

caller.

ata_dev_config

Name

`ata_dev_config` — Run device specific handlers and check for

Synopsis

```
void ata_dev_config (struct ata_port * ap, unsigned int i);
```

Arguments

ap

Bus

i

Device

Description

SATA->PATA bridges

ata_port_probe

Name

`ata_port_probe` —

Synopsis

```
void ata_port_probe (struct ata_port * ap);
```

Arguments

ap

__sata_phy_reset

Name

__sata_phy_reset —

Synopsis

```
void __sata_phy_reset (struct ata_port * ap);
```

Arguments

ap

LOCKING

sata_phy_reset

Name

sata_phy_reset —

Synopsis

```
void sata_phy_reset (struct ata_port * ap);
```

Arguments

ap

LOCKING

ata_port_disable

Name

ata_port_disable —

Synopsis

```
void ata_port_disable (struct ata_port * ap);
```

Arguments

ap

ata_bus_reset

Name

`ata_bus_reset` — reset host port and associated ATA channel

Synopsis

```
void ata_bus_reset (struct ata_port * ap);
```

Arguments

ap

port to reset

Description

This is typically the first time we actually start issuing commands to the ATA channel. We wait for BSY to clear, then issue EXECUTE DEVICE DIAGNOSTIC command, polling for its result. Determine what devices, if any, are on the channel by looking at the device 0/1 error register. Look at the signature stored in each device's taskfile registers, to determine if the device is ATA or ATAPI.

LOCKING

Inherited from caller. Some functions called by this function obtain the host_set lock.

SIDE EFFECTS

Sets ATA_FLAG_PORT_DISABLED if bus reset fails.

ata_qc_prep

Name

`ata_qc_prep` — Prepare taskfile for submission

Synopsis

```
void ata_qc_prep (struct ata_queued_cmd * qc);
```

Arguments

qc

Metadata associated with taskfile to be prepared

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_eng_timeout

Name

`ata_eng_timeout` — Handle timeout of queued command

Synopsis

```
void ata_eng_timeout (struct ata_port * ap);
```

Arguments

ap

Port on which timed-out command is active

Description

Some part of the kernel (currently, only the SCSI layer) has noticed that the active command on port *ap* has not completed after a specified length of time. Handle this condition by disabling DMA (if necessary) and completing transactions, with error if necessary.

This also handles the case of the “lost interrupt”, where for some reason (possibly hardware bug, possibly driver bug) an interrupt was not delivered to the driver, even though the transaction completed successfully.

LOCKING

Inherited from SCSI layer (none, can sleep)

ata_qc_complete

Name

`ata_qc_complete` — Complete an active ATA command

Synopsis

```
void ata_qc_complete (struct ata_queued_cmd * qc, u8  
drv_stat);
```

Arguments

qc

Command to complete

drv_stat

ATA status register contents

LOCKING

ata_qc_issue_prot

Name

`ata_qc_issue_prot` — issue taskfile to device in proto-dependent manner

Synopsis

```
int ata_qc_issue_prot (struct ata_queued_cmd * qc);
```

Arguments

qc

command to issue to device

Description

Using various libata functions and hooks, this function starts an ATA command. ATA commands are grouped into classes called “protocols”, and issuing each type of protocol is slightly different.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Zero on success, negative on error.

ata_host_intr

Name

`ata_host_intr` — Handle host interrupt for given (port, task)

Synopsis

```
unsigned int ata_host_intr (struct ata_port * ap, struct  
ata_queued_cmd * qc);
```

Arguments

ap

Port on which interrupt arrived (possibly...)

qc

Taskfile currently active in engine

Description

Handle host interrupt for given queued command. Currently, only DMA interrupts are handled. All other commands are handled via polling with interrupts disabled (nIEN bit).

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

One if interrupt was handled, zero if not (shared irq).

ata_interrupt

Name

`ata_interrupt` — Default ATA host interrupt handler

Synopsis

```
irqreturn_t ata_interrupt (int irq, void * dev_instance,  
struct pt_regs * regs);
```

Arguments

irq

irq line

dev_instance

pointer to our host information structure

regs

unused

LOCKING

RETURNS

ata_device_add

Name

`ata_device_add` —

Synopsis

```
int ata_device_add (struct ata_probe_ent * ent);
```

Arguments

ent

LOCKING

RETURNS

ata_scsi_release

Name

`ata_scsi_release` — SCSI layer callback hook for host unload

Synopsis

```
int ata_scsi_release (struct Scsi_Host * host);
```

Arguments

host

libata host to be unloaded

Description

Performs all duties necessary to shut down a libata port... Kill port kthread, disable port, and release resources.

LOCKING

Inherited from SCSI layer.

RETURNS

One.

ata_std_ports

Name

`ata_std_ports` — initialize ioaddr with standard port offsets.

Synopsis

```
void ata_std_ports (struct ata_ioports * ioaddr);
```

Arguments

ioaddr

IO address structure to be initialized

ata_pci_init_one

Name

`ata_pci_init_one` — Initialize/register PCI IDE host controller

Synopsis

```
int ata_pci_init_one (struct pci_dev * pdev, struct  
ata_port_info ** port_info, unsigned int n_ports);
```

Arguments

pdev

Controller to be initialized

port_info

Information from low-level host driver

n_ports

Number of ports attached to host controller

LOCKING

Inherited from PCI layer (may sleep).

RETURNS

ata_pci_remove_one

Name

`ata_pci_remove_one` — PCI layer callback for device removal

Synopsis

```
void ata_pci_remove_one (struct pci_dev * pdev);
```

Arguments

pdev

PCI device that was removed

Description

PCI layer indicates to libata via this hook that hot-unplug or module unload event has occurred. Handle this by unregistering all objects associated with this PCI

device. Free those objects. Then finally release PCI resources and disable device.

LOCKING

Inherited from PCI layer (may sleep).

Chapter 4. libata Core Internals

ata_tf_load_pio

Name

`ata_tf_load_pio` — send taskfile registers to host controller

Synopsis

```
void ata_tf_load_pio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

Port to which output is sent

tf

ATA taskfile register set

Description

Outputs ATA taskfile to standard ATA host controller.

LOCKING

Inherited from caller.

ata_tf_load_mmio

Name

`ata_tf_load_mmio` — send taskfile registers to host controller

Synopsis

```
void ata_tf_load_mmio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

Port to which output is sent

tf

ATA taskfile register set

Description

Outputs ATA taskfile to standard ATA host controller using MMIO.

LOCKING

Inherited from caller.

ata_exec_command_pio

Name

`ata_exec_command_pio` — issue ATA command to host controller

Synopsis

```
void ata_exec_command_pio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

port to which command is being issued

tf

ATA taskfile register set

Description

Issues PIO/MMIO write to ATA command register, with proper synchronization with interrupt handler / other threads.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_exec_command_mmio

Name

`ata_exec_command_mmio` — issue ATA command to host controller

Synopsis

```
void ata_exec_command_mmio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

port to which command is being issued

tf

ATA taskfile register set

Description

Issues MMIO write to ATA command register, with proper synchronization with interrupt handler / other threads.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_tf_read_pio

Name

`ata_tf_read_pio` — input device's ATA taskfile shadow registers

Synopsis

```
void ata_tf_read_pio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

Port from which input is read

tf

ATA taskfile register set for storing input

Description

Reads ATA taskfile registers for currently-selected device into *tf*.

LOCKING

Inherited from caller.

ata_tf_read_mmio

Name

`ata_tf_read_mmio` — input device's ATA taskfile shadow registers

Synopsis

```
void ata_tf_read_mmio (struct ata_port * ap, struct  
ata_taskfile * tf);
```

Arguments

ap

Port from which input is read

tf

ATA taskfile register set for storing input

Description

Reads ATA taskfile registers for currently-selected device into *tf* via MMIO.

LOCKING

Inherited from caller.

ata_check_status_pio

Name

`ata_check_status_pio` — Read device status reg & clear interrupt

Synopsis

```
u8 ata_check_status_pio (struct ata_port * ap);
```

Arguments

ap

port where the device is

Description

Reads ATA taskfile status register for currently-selected device and return it's value. This also clears pending interrupts from this device

LOCKING

Inherited from caller.

ata_check_status_mmio

Name

`ata_check_status_mmio` — Read device status reg & clear interrupt

Synopsis

```
u8 ata_check_status_mmio (struct ata_port * ap);
```

Arguments

ap

port where the device is

Description

Reads ATA taskfile status register for currently-selected device via MMIO and return it's value. This also clears pending interrupts from this device

LOCKING

Inherited from caller.

ata_prot_to_cmd

Name

`ata_prot_to_cmd` — determine which read/write opcodes to use

Synopsis

```
int ata_prot_to_cmd (int protocol, int lba48);
```


Arguments

protocol

ATA_PROT_xxx taskfile protocol

lba48

true is lba48 is present

Description

Given necessary input, determine which read/write commands to use to transfer data.

LOCKING

None.

ata_dev_set_protocol

Name

`ata_dev_set_protocol` — set taskfile protocol and r/w commands

Synopsis

```
void ata_dev_set_protocol (struct ata_device * dev);
```

Arguments

dev

device to examine and configure

Description

Examine the device configuration, after we have read the identify-device page and configured the data transfer mode. Set internal state related to the ATA taskfile protocol (pio, pio mult, dma, etc.) and calculate the proper read/write commands to use.

LOCKING

caller.

ata_mode_string

Name

`ata_mode_string` — convert UDMA bit offset to string

Synopsis

```
const char * ata_mode_string (unsigned int mask);
```

Arguments

mask

mask of bits supported; only highest bit counts.

Description

Determine string which represents the highest speed (highest bit in *udma_mask*).

LOCKING

None.

RETURNS

Constant C string representing highest speed listed in *udma_mask*, or the constant C string “<n/a>”.

ata_pio_devchk

Name

ata_pio_devchk — PATA device presence detection

Synopsis

```
unsigned int ata_pio_devchk (struct ata_port * ap, unsigned  
int device);
```

Arguments

ap

ATA channel to examine

device

Device to examine (starting at zero)

Description

This technique was originally described in Hale Landis's ATADRVR (www.ata-atapi.com), and later found its way into the ATA/ATAPI spec.

Write a pattern to the ATA shadow registers, and if a device is present, it will respond by correctly storing and echoing back the ATA shadow register contents.

LOCKING

caller.

ata_mmio_devchk

Name

`ata_mmio_devchk` — PATA device presence detection

Synopsis

```
unsigned int ata_mmio_devchk (struct ata_port * ap, unsigned  
int device);
```

Arguments

ap

ATA channel to examine

device

Device to examine (starting at zero)

Description

This technique was originally described in Hale Landis's ATADRVR (www.ata-atapi.com), and later found its way into the ATA/ATAPI spec.

Write a pattern to the ATA shadow registers, and if a device is present, it will respond by correctly storing and echoing back the ATA shadow register contents.

LOCKING

caller.

ata_devchk

Name

ata_devchk — PATA device presence detection

Synopsis

```
unsigned int ata_devchk (struct ata_port * ap, unsigned int  
device);
```

Arguments

ap

ATA channel to examine

device

Device to examine (starting at zero)

Description

Dispatch ATA device presence detection, depending on whether we are using PIO or MMIO to talk to the ATA shadow registers.

LOCKING

caller.

ata_dev_try_classify

Name

`ata_dev_try_classify` — Parse returned ATA device signature

Synopsis

```
u8 ata_dev_try_classify (struct ata_port * ap, unsigned int
device);
```

Arguments

ap

ATA channel to examine

device

Device to examine (starting at zero)

Description

After an event -- SRST, E.D.D., or SATA COMRESET -- occurs, an ATA/ATAPI-defined set of values is placed in the ATA shadow registers, indicating the results of device detection and diagnostics.

Select the ATA device, and read the values from the ATA shadow registers. Then parse according to the Error register value, and the spec-defined values examined by `ata_dev_classify`.

LOCKING

caller.

ata_dev_select

Name

`ata_dev_select` — Select device 0/1 on ATA bus

Synopsis

```
void ata_dev_select (struct ata_port * ap, unsigned int
device, unsigned int wait, unsigned int can_sleep);
```

Arguments

ap

ATA channel to manipulate

device

ATA device (numbered from zero) to select

wait

non-zero to wait for Status register BSY bit to clear

can_sleep

non-zero if context allows sleeping

Description

Use the method defined in the ATA specification to make either device 0, or device 1, active on the ATA channel.

This is a high-level version of `ata_std_dev_select`, which additionally provides the services of inserting the proper pauses and status polling, where needed.

LOCKING

caller.

ata_dump_id

Name

`ata_dump_id` — IDENTIFY DEVICE info debugging output

Synopsis

```
void ata_dump_id (struct ata_device * dev);
```


Arguments

dev

Device whose IDENTIFY DEVICE page we will dump

Description

Dump selected 16-bit words from a detected device's IDENTIFY PAGE page.

LOCKING

caller.

ata_dev_identify

Name

`ata_dev_identify` — obtain IDENTIFY x DEVICE page

Synopsis

```
void ata_dev_identify (struct ata_port * ap, unsigned int  
device);
```

Arguments

ap

port on which device we wish to probe resides

device

device bus address, starting at zero

Description

Following bus reset, we issue the IDENTIFY [PACKET] DEVICE command, and read back the 512-byte device information page. The device information page is fed to us via the standard PIO-IN protocol, but we hand-code it here. (TODO: investigate using standard PIO-IN paths)

After reading the device information page, we use several bits of information from it to initialize data structures that will be used during the lifetime of the `ata_device`. Other data from the info page is used to disqualify certain older ATA devices we do not wish to support.

LOCKING

Inherited from caller. Some functions called by this function obtain the `host_set` lock.

ata_bus_probe

Name

`ata_bus_probe` — Reset and probe ATA bus

Synopsis

```
int ata_bus_probe (struct ata_port * ap);
```

Arguments

ap

Bus to probe

LOCKING

RETURNS

Zero on success, non-zero on error.

ata_set_mode

Name

`ata_set_mode` — Program timings and issue SET FEATURES - XFER

Synopsis

```
void ata_set_mode (struct ata_port * ap);
```

Arguments

ap

port on which timings will be programmed

LOCKING

ata_busy_sleep

Name

`ata_busy_sleep` — sleep until BSY clears, or timeout

Synopsis

```
unsigned int ata_busy_sleep (struct ata_port * ap, unsigned  
long tmout_pat, unsigned long tmout);
```

Arguments

ap

port containing status register to be polled

tmout_pat

impatience timeout

tmout

overall timeout

LOCKING

ata_choose_xfer_mode

Name

`ata_choose_xfer_mode` — attempt to find best transfer mode

Synopsis

```
int ata_choose_xfer_mode (struct ata_port * ap, u8 *  
xfer_mode_out, unsigned int * xfer_shift_out);
```

Arguments

ap

Port for which an xfer mode will be selected

xfer_mode_out

(output) SET FEATURES - XFER MODE code

xfer_shift_out

(output) bit shift that selects this mode

LOCKING

RETURNS

Zero on success, negative on error.

ata_dev_set_xfermode

Name

`ata_dev_set_xfermode` — Issue SET FEATURES - XFER MODE command

Synopsis

```
void ata_dev_set_xfermode (struct ata_port * ap, struct  
ata_device * dev);
```

Arguments

ap

Port associated with device *dev*

dev

Device to which command will be sent

ata_dev_init_params

Name

`ata_dev_init_params` — Issue INIT DEV PARAMS command

Synopsis

```
void ata_dev_init_params (struct ata_port * ap, struct  
ata_device * dev);
```

Arguments

ap

Port associated with device *dev*

dev

Device to which command will be sent

ata_sg_clean

Name

`ata_sg_clean` —

Synopsis

```
void ata_sg_clean (struct ata_queued_cmd * qc);
```

Arguments

qc

ata_fill_sg

Name

`ata_fill_sg` — Fill PCI IDE PRD table

Synopsis

```
void ata_fill_sg (struct ata_queued_cmd * qc);
```

Arguments

qc

Metadata associated with taskfile to be transferred

LOCKING

ata_check_atapi_dma

Name

`ata_check_atapi_dma` — Check whether ATAPI DMA can be supported

Synopsis

```
int ata_check_atapi_dma (struct ata_queued_cmd * qc);
```


Arguments

qc

Metadata associated with taskfile to check

RETURNS

0 when ATAPI DMA can be used nonzero otherwise

ata_sg_setup_one

Name

`ata_sg_setup_one` —

Synopsis

```
int ata_sg_setup_one (struct ata_queued_cmd * qc);
```

Arguments

qc

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

ata_sg_setup

Name

`ata_sg_setup` —

Synopsis

```
int ata_sg_setup (struct ata_queued_cmd * qc);
```

Arguments

qc

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

ata_pio_poll

Name

`ata_pio_poll` —

Synopsis

```
unsigned long ata_pio_poll (struct ata_port * ap);
```

Arguments

ap

LOCKING

RETURNS

ata_pio_complete

Name

`ata_pio_complete` —

Synopsis

```
void ata_pio_complete (struct ata_port * ap);
```

Arguments

ap

ata_pio_block

Name

`ata_pio_block` —

Synopsis

```
void ata_pio_block (struct ata_port * ap);
```

Arguments

ap

ata_qc_timeout

Name

`ata_qc_timeout` — Handle timeout of queued command

Synopsis

```
void ata_qc_timeout (struct ata_queued_cmd * qc);
```

Arguments

qc

Command that timed out

Description

Some part of the kernel (currently, only the SCSI layer) has noticed that the active command on port *ap* has not completed after a specified length of time. Handle this condition by disabling DMA (if necessary) and completing transactions, with error if necessary.

This also handles the case of the “lost interrupt”, where for some reason (possibly hardware bug, possibly driver bug) an interrupt was not delivered to the driver, even though the transaction completed successfully.

ata_qc_new

Name

`ata_qc_new` — Request an available ATA command, for queueing

Synopsis

```
struct ata_queued_cmd * ata_qc_new (struct ata_port * ap);
```

Arguments

ap

Port associated with device *dev*

ata_qc_new_init

Name

`ata_qc_new_init` — Request an available ATA command, and initialize it

Synopsis

```
struct ata_queued_cmd * ata_qc_new_init (struct ata_port * ap,  
struct ata_device * dev);
```

Arguments

ap

Port associated with device *dev*

dev

Device from whom we request an available command structure

ata_qc_free

Name

ata_qc_free — free unused ata_queued_cmd

Synopsis

```
void ata_qc_free (struct ata_queued_cmd * qc);
```

Arguments

qc

Command to complete

Description

Designed to free unused ata_queued_cmd object in case something prevents using it.

LOCKING

ata_qc_issue

Name

`ata_qc_issue` — issue taskfile to device

Synopsis

```
int ata_qc_issue (struct ata_queued_cmd * qc);
```

Arguments

qc

command to issue to device

Description

Prepare an ATA command to submission to device. This includes mapping the data into a DMA-able area, filling in the S/G table, and finally writing the taskfile to hardware, starting the command.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Zero on success, negative on error.

ata_bmdma_setup_mmio

Name

`ata_bmdma_setup_mmio` — Set up PCI IDE BMDMA transaction

Synopsis

```
void ata_bmdma_setup_mmio (struct ata_queued_cmd * qc);
```

Arguments

qc

Info associated with this ATA transaction.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_bmdma_start_mmio

Name

`ata_bmdma_start_mmio` — Start a PCI IDE BMDMA transaction

Synopsis

```
void ata_bmdma_start_mmio (struct ata_queued_cmd * qc);
```

Arguments

qc

Info associated with this ATA transaction.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_bmdma_setup_pio

Name

`ata_bmdma_setup_pio` — Set up PCI IDE BMDMA transaction (PIO)

Synopsis

```
void ata_bmdma_setup_pio (struct ata_queued_cmd * qc);
```

Arguments

qc

Info associated with this ATA transaction.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_bmdma_start_pio

Name

`ata_bmdma_start_pio` — Start a PCI IDE BMDMA transaction (PIO)

Synopsis

```
void ata_bmdma_start_pio (struct ata_queued_cmd * qc);
```

Arguments

qc

Info associated with this ATA transaction.

LOCKING

`spin_lock_irqsave(host_set lock)`

atapi_packet_task

Name

`atapi_packet_task` — Write CDB bytes to hardware

Synopsis

```
void atapi_packet_task (void * _data);
```

Arguments

_data

Port to which ATAPI device is attached.

Description

When device has indicated its readiness to accept a CDB, this function is called. Send the CDB. If DMA is to be performed, exit immediately. Otherwise, we are in polling mode, so poll status under operation succeeds or fails.

LOCKING

Kernel thread context (may sleep)

ata_host_remove

Name

`ata_host_remove` — Unregister SCSI host structure with upper layers

Synopsis

```
void ata_host_remove (struct ata_port * ap, unsigned int  
do_unregister);
```

Arguments

ap

Port to unregister

do_unregister

1 if we fully unregister, 0 to just stop the port

ata_host_init

Name

`ata_host_init` — Initialize an `ata_port` structure

Synopsis

```
void ata_host_init (struct ata_port * ap, struct Scsi_Host *  
host, struct ata_host_set * host_set, struct ata_probe_ent *  
ent, unsigned int port_no);
```

Arguments

ap

Structure to initialize

host

associated SCSI mid-layer structure

host_set

Collection of hosts to which *ap* belongs

ent

Probe information provided by low-level driver

port_no

Port number associated with this *ata_port*

LOCKING

ata_host_add

Name

ata_host_add — Attach low-level ATA driver to system

Synopsis

```
struct ata_port * ata_host_add (struct ata_probe_ent * ent,  
struct ata_host_set * host_set, unsigned int port_no);
```

Arguments

ent

Information provided by low-level driver

host_set

Collections of ports to which we add

port_no

Port number associated with this host

LOCKING

RETURNS

ata_init

Name

`ata_init` —

Synopsis

```
int __init ata_init ( void );
```

Arguments

void

no arguments

Description

LOCKING

RETURNS

Chapter 5. libata SCSI translation/emulation

ata_std_bios_param

Name

`ata_std_bios_param` — generic bios head/sector/cylinder calculator used by `sd`.

Synopsis

```
int ata_std_bios_param (struct scsi_device * sdev, struct  
block_device * bdev, sector_t capacity, int geom[]);
```

Arguments

sdev

SCSI device for which BIOS geometry is to be determined

bdev

block device associated with *sdev*

capacity

capacity of SCSI device

geom[]

location to which geometry will be output

Description

Generic bios head/sector/cylinder calculator used by sd. Most BIOSes nowadays expect a XXX/255/16 (CHS) mapping. Some situations may arise where the disk is not bootable if this is not used.

LOCKING

Defined by the SCSI layer. We don't really care.

RETURNS

Zero.

ata_scsi_slave_config

Name

`ata_scsi_slave_config` — Set SCSI device attributes

Synopsis

```
int ata_scsi_slave_config (struct scsi_device * sdev);
```

Arguments

sdev

SCSI device to examine

Description

This is called before we actually start reading and writing to the device, to configure certain SCSI mid-layer behaviors.

LOCKING

Defined by SCSI layer. We don't really care.

ata_scsi_error

Name

`ata_scsi_error` — SCSI layer error handler callback

Synopsis

```
int ata_scsi_error (struct Scsi_Host * host);
```

Arguments

host

SCSI host on which error occurred

Description

Handles SCSI-layer-thrown error events.

LOCKING

Inherited from SCSI layer (none, can sleep)

RETURNS

Zero.

ata_scsi_queuecmd

Name

`ata_scsi_queuecmd` — Issue SCSI cdb to libata-managed device

Synopsis

```
int ata_scsi_queuecmd (struct scsi_cmnd * cmd, void (*done)
(struct scsi_cmnd *));
```

Arguments

cmd

SCSI command to be sent

done

Completion function, called when command is complete

Description

In some cases, this function translates SCSI commands into ATA taskfiles, and queues the taskfiles to be sent to hardware. In other cases, this function simulates a

SCSI device by evaluating and responding to certain SCSI commands. This creates the overall effect of ATA and ATAPI devices appearing as SCSI devices.

LOCKING

Releases scsi-layer-held lock, and obtains host_set lock.

RETURNS

Zero.

ata_scsi_simulate

Name

`ata_scsi_simulate` — simulate SCSI command on ATA device

Synopsis

```
void ata_scsi_simulate (u16 * id, struct scsi_cmnd * cmd, void  
(*done) (struct scsi_cmnd *));
```

Arguments

id

current IDENTIFY data for target device.

cmd

SCSI command being sent to device.

done

SCSI command completion function.

Description

Interprets and directly executes a select list of SCSI commands that can be handled internally.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_cmd_ioctl

Name

`ata_cmd_ioctl` — Handler for HDIO_DRIVE_CMD ioctl

Synopsis

```
int ata_cmd_ioctl (struct scsi_device * scsidev, void __user *  
arg);
```

Arguments

scsidev

-- undescribed --

arg

User provided data for issuing command

LOCKING

Defined by the SCSI layer. We don't really care.

RETURNS

Zero on success, negative errno on error.

ata_task_ioctl

Name

`ata_task_ioctl` — Handler for HDIO_DRIVE_TASK ioctl

Synopsis

```
int ata_task_ioctl (struct scsi_device * scsidev, void __user  
* arg);
```

Arguments

scsidev

-- undscribed --

arg

User provided data for issuing command

LOCKING

Defined by the SCSI layer. We don't really care.

RETURNS

Zero on success, negative errno on error.

ata_scsi_qc_new

Name

`ata_scsi_qc_new` — acquire new `ata_queued_cmd` reference

Synopsis

```
struct ata_queued_cmd * ata_scsi_qc_new (struct ata_port * ap,  
struct ata_device * dev, struct scsi_cmnd * cmd, void (*done)  
(struct scsi_cmnd *));
```

Arguments

ap

ATA port to which the new command is attached

dev

ATA device to which the new command is attached

cmd

SCSI command that originated this ATA command

done

SCSI command completion function

Description

Obtain a reference to an unused `ata_queued_cmd` structure, which is the basic libata structure representing a single ATA command sent to the hardware.

If a command was available, fill in the SCSI-specific portions of the structure with information on the current command.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Command allocated, or `NULL` if none available.

ata_to_sense_error

Name

`ata_to_sense_error` — convert ATA error to SCSI error

Synopsis

```
void ata_to_sense_error (struct ata_queued_cmd * qc, u8  
drv_stat);
```

Arguments

qc

Command that we are erroring out

drv_stat

value contained in ATA status register

Description

Converts an ATA error into a SCSI error. While we are at it we decode and dump the ATA error for the user so that they have some idea what really happened at the non make-believe layer.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsi_flush_xlat

Name

`ata_scsi_flush_xlat` — Translate SCSI SYNCHRONIZE CACHE command

Synopsis

```
unsigned int ata_scsi_flush_xlat (struct ata_queued_cmd * qc,  
u8 * scsicmd);
```

Arguments

qc

Storage for translated ATA taskfile

scsicmd

SCSI command to translate (ignored)

Description

Sets up an ATA taskfile to issue FLUSH CACHE or FLUSH CACHE EXT.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Zero on success, non-zero on error.

ata_scsi_verify_xlat

Name

`ata_scsi_verify_xlat` — Translate SCSI VERIFY command into an ATA one

Synopsis

```
unsigned int ata_scsi_verify_xlat (struct ata_queued_cmd * qc,  
u8 * scsicmd);
```

Arguments

qc

Storage for translated ATA taskfile

scsicmd

SCSI command to translate

Description

Converts SCSI VERIFY command to an ATA READ VERIFY command.

LOCKING

spin_lock_irqsave(host_set lock)

RETURNS

Zero on success, non-zero on error.

ata_scsi_rw_xlat

Name

`ata_scsi_rw_xlat` — Translate SCSI r/w command into an ATA one

Synopsis

```
unsigned int ata_scsi_rw_xlat (struct ata_queued_cmd * qc, u8  
* scsicmd);
```

Arguments

qc

Storage for translated ATA taskfile

scsi cmd

SCSI command to translate

Description

Converts any of six SCSI read/write commands into the ATA counterpart, including starting sector (LBA), sector count, and taking into account the device's LBA48 support.

Commands READ_6, READ_10, READ_16, WRITE_6, WRITE_10, and WRITE_16 are currently supported.

LOCKING

spin_lock_irqsave(host_set lock)

RETURNS

Zero on success, non-zero on error.

ata_scsi_translate

Name

`ata_scsi_translate` — Translate then issue SCSI command to ATA device

Synopsis

```
void ata_scsi_translate (struct ata_port * ap, struct  
ata_device * dev, struct scsi_cmnd * cmd, void (*done) (struct  
scsi_cmnd *), ata_xlat_func_t xlat_func);
```

Arguments

ap

ATA port to which the command is addressed

dev

ATA device to which the command is addressed

cmd

SCSI command to execute

done

SCSI command completion function

xlat_func

Actor which translates *cmd* to an ATA taskfile

Description

Our `->queuecommand` function has decided that the SCSI command issued can be directly translated into an ATA command, rather than handled internally.

This function sets up an `ata_queued_cmd` structure for the SCSI command, and sends that `ata_queued_cmd` to the hardware.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsi_rbuf_get

Name

`ata_scsi_rbuf_get` — Map response buffer.

Synopsis

```
unsigned int ata_scsi_rbuf_get (struct scsi_cmnd * cmd, u8 **  
buf_out);
```

Arguments

cmd

SCSI command containing buffer to be mapped.

buf_out

Pointer to mapped area.

Description

Maps buffer contained within SCSI command *cmd*.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Length of response buffer.

ata_scsi_rbuf_put

Name

`ata_scsi_rbuf_put` — Unmap response buffer.

Synopsis

```
void ata_scsi_rbuf_put (struct scsi_cmnd * cmd, u8 * buf);
```

Arguments

cmd

SCSI command containing buffer to be unmapped.

buf

buffer to unmap

Description

Unmaps response buffer contained within *cmd*.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsi_rbuf_fill

Name

`ata_scsi_rbuf_fill` — wrapper for SCSI command simulators

Synopsis

```
void ata_scsi_rbuf_fill (struct ata_scsi_args * args, unsigned  
int (*actor) (struct ata_scsi_args *args, u8 *rbuf, unsigned  
int buflen));
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

actor

Callback hook for desired SCSI command simulator

Description

Takes care of the hard work of simulating a SCSI command... Mapping the response buffer, calling the command's handler, and handling the handler's return value. This return value indicates whether the handler wishes the SCSI command to be completed successfully, or not.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_inq_std

Name

`ata_scsiop_inq_std` — Simulate INQUIRY command

Synopsis

```
unsigned int ata_scsiop_inq_std (struct ata_scsi_args * args,  
u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Returns standard device identification data associated with non-EVPD INQUIRY command output.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_inq_00

Name

`ata_scsiop_inq_00` — Simulate INQUIRY EVPD page 0, list of pages

Synopsis

```
unsigned int ata_scsiop_inq_00 (struct ata_scsi_args * args,  
u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Returns list of inquiry EVPD pages available.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_inq_80

Name

`ata_scsiop_inq_80` — Simulate INQUIRY EVPD page 80, device serial number

Synopsis

```
unsigned int ata_scsiop_inq_80 (struct ata_scsi_args * args,  
u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Returns ATA device serial number.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_inq_83

Name

`ata_scsiop_inq_83` — Simulate INQUIRY EVPD page 83, device identity

Synopsis

```
unsigned int ata_scsiop_inq_83 (struct ata_scsi_args * args,  
u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Returns device identification. Currently hardcoded to return “Linux ATA-SCSI simulator”.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_noop

Name

`ata_scsiop_noop` —

Synopsis

```
unsigned int ata_scsiop_noop (struct ata_scsi_args * args, u8  
* rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

No operation. Simply returns success to caller, to indicate that the caller should successfully complete this SCSI command.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_msense_push

Name

`ata_msense_push` — Push data onto MODE SENSE data output buffer

Synopsis

```
void ata_msense_push (u8 ** ptr_io, const u8 * last, const u8  
* buf, unsigned int buflen);
```

Arguments

ptr_io

(input/output) Location to store more output data

last

End of output data buffer

buf

Pointer to BLOB being added to output buffer

buflen

Length of BLOB

Description

Store MODE SENSE data on an output buffer.

LOCKING

None.

ata_msense_caching

Name

`ata_msense_caching` — Simulate MODE SENSE caching info page

Synopsis

```
unsigned int ata_msense_caching (u16 * id, u8 ** ptr_io, const  
u8 * last);
```

Arguments

id

device IDENTIFY data

ptr_io

(input/output) Location to store more output data

last

End of output data buffer

Description

Generate a caching info page, which conditionally indicates write caching to the SCSI layer, depending on device capabilities.

LOCKING

None.

ata_msense_ctl_mode

Name

`ata_msense_ctl_mode` — Simulate MODE SENSE control mode page

Synopsis

```
unsigned int ata_msense_ctl_mode (u8 ** ptr_io, const u8 *  
last);
```

Arguments

ptr_io

(input/output) Location to store more output data

last

End of output data buffer

Description

Generate a generic MODE SENSE control mode page.

LOCKING

None.

ata_msense_rw_recovery

Name

`ata_msense_rw_recovery` — Simulate MODE SENSE r/w error recovery page

Synopsis

```
unsigned int ata_msense_rw_recovery (u8 ** ptr_io, const u8 *  
last);
```

Arguments

ptr_io

(input/output) Location to store more output data

last

End of output data buffer

Description

Generate a generic MODE SENSE r/w error recovery page.

LOCKING

None.

ata_scsiop_mode_sense

Name

`ata_scsiop_mode_sense` — Simulate MODE SENSE 6, 10 commands

Synopsis

```
unsigned int ata_scsiop_mode_sense (struct ata_scsi_args *  
args, u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Simulate MODE SENSE commands.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_read_cap

Name

`ata_scsiop_read_cap` — Simulate READ CAPACITY[16] commands

Synopsis

```
unsigned int ata_scsiop_read_cap (struct ata_scsi_args * args,  
u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Simulate READ CAPACITY commands.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsiop_report_luns

Name

`ata_scsiop_report_luns` — Simulate REPORT LUNS command

Synopsis

```
unsigned int ata_scsiop_report_luns (struct ata_scsi_args *  
args, u8 * rbuf, unsigned int buflen);
```

Arguments

args

device IDENTIFY data / SCSI command of interest.

rbuf

Response buffer, to which simulated SCSI cmd output is sent.

buflen

Response buffer length.

Description

Simulate REPORT LUNS command.

LOCKING

`spin_lock_irqsave(host_set lock)`

ata_scsi_badcmd

Name

`ata_scsi_badcmd` — End a SCSI request with an error

Synopsis

```
void ata_scsi_badcmd (struct scsi_cmnd * cmd, void (*done)
(struct scsi_cmnd *), u8 asc, u8 ascq);
```

Arguments

cmd

SCSI request to be handled

done

SCSI command completion function

asc

SCSI-defined additional sense code

ascq

SCSI-defined additional sense code qualifier

Description

Helper function that completes a SCSI command with `SAM_STAT_CHECK_CONDITION`, with a sense key `ILLEGAL_REQUEST` and the specified additional sense codes.

LOCKING

`spin_lock_irqsave(host_set lock)`

atapi_xlat

Name

`atapi_xlat` — Initialize PACKET taskfile

Synopsis

```
unsigned int atapi_xlat (struct ata_queued_cmd * qc, u8 *  
scsi cmd);
```

Arguments

qc

command structure to be initialized

scsi cmd

SCSI CDB associated with this PACKET command

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Zero on success, non-zero on failure.

ata_scsi_find_dev

Name

`ata_scsi_find_dev` — lookup ata_device from scsi_cmnd

Synopsis

```
struct ata_device * ata_scsi_find_dev (struct ata_port * ap,  
struct scsi_device * scsidev);
```

Arguments

ap

ATA port to which the device is attached

scsidev

SCSI device from which we derive the ATA device

Description

Given various information provided in struct `scsi_cmnd`, map that onto an ATA bus, and using that mapping determine which `ata_device` is associated with the SCSI command to be sent.

LOCKING

`spin_lock_irqsave(host_set lock)`

RETURNS

Associated ATA device, or `NULL` if not found.

ata_scsi_pass_thru

Name

`ata_scsi_pass_thru` — convert ATA pass-thru CDB to taskfile

Synopsis

```
unsigned int ata_scsi_pass_thru (struct ata_queued_cmd * qc,  
u8 * scsicmd);
```

Arguments

qc

command structure to be initialized

scsicmd

-- undescribed --

Description

Handles either 12 or 16-byte versions of the CDB.

RETURNS

Zero on success, non-zero on failure.

ata_get_xlat_func

Name

`ata_get_xlat_func` — check if SCSI to ATA translation is possible

Synopsis

```
ata_xlat_func_t ata_get_xlat_func (struct ata_device * dev, u8  
cmd);
```

Arguments

dev

ATA device

cmd

SCSI command opcode to consider

Description

Look up the SCSI command given, and determine whether the SCSI command is to be translated or simulated.

RETURNS

Pointer to translation function if possible, `NULL` if not.

ata_scsi_dump_cdb

Name

`ata_scsi_dump_cdb` — dump SCSI command contents to dmesg

Synopsis

```
void ata_scsi_dump_cdb (struct ata_port * ap, struct scsi_cmnd  
* cmd);
```

Arguments

ap

ATA port to which the command was being sent

cmd

SCSI command to dump

Description

Prints the contents of a SCSI command via `printk`.

Chapter 6. ata_piix Internals

piix_pata_cbl_detect

Name

`piix_pata_cbl_detect` — Probe host controller cable detect info

Synopsis

```
void piix_pata_cbl_detect (struct ata_port * ap);
```

Arguments

ap

Port for which cable detect info is desired

Description

Read 80c cable indicator from ATA PCI device's PCI config register. This register is normally set by firmware (BIOS).

LOCKING

None (inherited from caller).

piix_pata_phy_reset

Name

`piix_pata_phy_reset` — Probe specified port on PATA host controller

Synopsis

```
void piix_pata_phy_reset (struct ata_port * ap);
```

Arguments

ap

Port to probe

Description

Probe PATA phy.

LOCKING

None (inherited from caller).

piix_sata_probe

Name

`piix_sata_probe` — Probe PCI device for present SATA devices

Synopsis

```
int piix_sata_probe (struct ata_port * ap);
```

Arguments

ap

Port associated with the PCI device we wish to probe

Description

Reads SATA PCI device's PCI config register Port Configuration and Status (PCS) to determine port and device availability.

LOCKING

None (inherited from caller).

RETURNS

Non-zero if device detected, zero otherwise.

piix_sata_phy_reset

Name

`piix_sata_phy_reset` — Probe specified port on SATA host controller

Synopsis

```
void piix_sata_phy_reset (struct ata_port * ap);
```

Arguments

ap

Port to probe

Description

Probe SATA phy.

LOCKING

None (inherited from caller).

piix_set_piomode

Name

`piix_set_piomode` — Initialize host controller PATA PIO timings

Synopsis

```
void piix_set_piomode (struct ata_port * ap, struct ata_device  
* adev);
```


Arguments

ap

Port whose timings we are configuring

adev

um

Description

Set PIO mode for device, in host controller PCI config space.

LOCKING

None (inherited from caller).

piix_set_dmamode

Name

`piix_set_dmamode` — Initialize host controller PATA PIO timings

Synopsis

```
void piix_set_dmamode (struct ata_port * ap, struct ata_device  
* adev);
```

Arguments

ap

Port whose timings we are configuring

adev

um

Description

Set UDMA mode for device, in host controller PCI config space.

LOCKING

None (inherited from caller).

piix_init_one

Name

`piix_init_one` — Register PIIX ATA PCI device with kernel services

Synopsis

```
int piix_init_one (struct pci_dev * pdev, const struct  
pci_device_id * ent);
```

Arguments

pdev

PCI device to register

ent

Entry in `piix_pci_tbl` matching with *pdev*

Description

Called from kernel PCI layer. We probe for combined mode (sigh), and then hand over control to libata, for it to do the rest.

LOCKING

Inherited from PCI layer (may sleep).

RETURNS

Zero on success, or -ERRNO value.

piix_init

Name

`piix_init` —

Synopsis

```
int __init piix_init ( void );
```

Arguments

void

no arguments

Description

LOCKING

RETURNS

piix_exit

Name

`piix_exit` —

Synopsis

```
void __exit piix_exit ( void );
```

Arguments

void

no arguments

Description

LOCKING

Chapter 7. sata_sil Internals

sil_dev_config

Name

`sil_dev_config` — Apply device/host-specific errata fixups

Synopsis

```
void sil_dev_config (struct ata_port * ap, struct ata_device *  
dev);
```

Arguments

ap

Port containing device to be examined

dev

Device to be examined

Description

After the IDENTIFY [PACKET] DEVICE step is complete, and a device is known to be present, this function is called. We apply two errata fixups which are specific to Silicon Image, a Seagate and a Maxtor fixup.

For certain Seagate devices, we must limit the maximum sectors to under 8K.

For certain Maxtor devices, we must not program the drive beyond udma5.

Both fixups are unfairly pessimistic. As soon as I get more information on these errata, I will create a more exhaustive list, and apply the fixups to only the specific devices/hosts/firmwares that need it.

20040111 - Seagate drives affected by the Mod15Write bug are blacklisted The Maxtor quirk is in the blacklist, but I'm keeping the original pessimistic fix for the following reasons... - There seems to be less info on it, only one device gleaned off the Windows driver, maybe only one is affected. More info would be greatly appreciated. - But then again UDMA5 is hardly anything to complain about