libATA Developer's Guide

Jeff Garzik

libATA Developer's Guide

by Jeff Garzik

Copyright © 2003-2006 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. Introduction **1**

2. libata Driver API **3**

2.1. struct ata_port_operations 3

2.1.1. Disable ATA port 3

2.1.2. Post-IDENTIFY device configuration 3

2.1.3. Set PIO/DMA mode 3

2.1.4. Taskfile read/write 4

2.1.5. PIO data read/write 4

2.1.6. ATA command execute 4

2.1.7. Per-cmd ATAPI DMA capabilities filter 5

2.1.8. Read specific ATA shadow registers 5

2.1.9. Write specific ATA shadow register 5

2.1.10. Select ATA device on bus 5

2.1.11. Private tuning method 6

2.1.12. Control PCI IDE BMDMA engine 6

2.1.13. High-level taskfile hooks 7

2.1.14. Exception and probe handling (EH) 7

2.1.15. Hardware interrupt handling 8

2.1.16. SATA phy read/write 9

2.1.17. Init and shutdown 9

3. Error handling **11**

3.1. Origins of commands 11

3.2. How commands are issued 11

3.3. How commands are processed 12

3.4. How commands are completed 12

3.5. ata_scsi_error() 13

3.6. Problems with the current EH 14

4. libata Library **17**

ata_link_next 17

ata_dev_next 18

atapi_cmd_type 19

ata_tf_to_fis 20

ata_tf_from_fis 21

ata_pack_xfermask 22

ata_unpack_xfermask 23

ata_xfer_mask2mode 24

ata_xfer_mode2mask 25

ata_xfer_mode2shift 26

ata_mode_string 27

ata_dev_classify 28

ata_id_string 29
ata_id_c_string 30
ata_id_xfermask 31
ata_pio_need_iordy 32
ata_do_dev_read_id 33
ata_cable_40wire 34
ata_cable_80wire 35
ata_cable_unknown 36
ata_cable_ignore 37
ata_cable_sata 38
ata_dev_pair 38
sata_set_spd 39
ata_timing_cycle2mode 40
ata_do_set_mode 41
ata_wait_after_reset 42
sata_link_debounce 44
sata_link_resume 45
sata_link_scr_lpm 46
ata_std_prereset 47
sata_link_hardreset 49
sata_std_hardreset 50
ata_std_postreset 51
ata_dev_set_feature 52
ata_std_qc_defer 53
ata_sg_init 55
ata_qc_complete 56
ata_qc_complete_multiple 57
sata_scr_valid 58
sata_scr_read 59
sata_scr_write 60
sata_scr_write_flush 61
ata_link_online 62
ata_link_offline 63
ata_host_suspend 65
ata_host_resume 65
ata_host_alloc 66
ata_host_alloc_pinfo 67
ata_slave_link_init 69
ata_host_start 70
ata_host_init 71
ata_host_register 72
ata_host_activate 73
ata_host_detach 75

ata_pci_remove_one 76
ata_msleep 76
ata_wait_register 77
5. libata Core Internals 81
ata_dev_phys_link 81
ata_force_cbl 82
ata_force_link_limits 83
ata_force_xfermask 84
ata_force_horkage 85
ata_rwcmd_protocol 85
ata_tf_read_block 86
ata_build_rw_tf 87
ata_read_native_max_address 89
ata_set_max_sectors 90
ata_hpa_resize 91
ata_dump_id 92
ata_exec_internal_sg 93
ata_exec_internal 94
ata_do_simple_cmd 96
ata_pio_mask_no_iordy 97
ata_dev_read_id 98
ata_dev_configure 99
ata_bus_probe 100
sata_print_link_status 101
sata_down_spd_limit 102
sata_set_spd_needed 103
ata_down_xfermask_limit 104
ata_wait_ready 105
ata_dev_same_device 107
ata_dev_reread_id 108
ata_dev_revalidate 109
glob_match 110
ata_is_40wire 112
cable_is_40wire 112
ata_dev_xfermask 113
ata_dev_set_xfermode 114
ata_dev_init_params 115
ata_sg_clean 116
atapi_check_dma 117
ata_sg_setup 118
swap_buf_le16 119
ata_qc_new 120
ata_qc_new_init 121

ata_qc_free 122
ata_qc_issue 123
ata_phys_link_online 124
ata_phys_link_offline 125
ata_dev_init 126
ata_link_init 127
sata_link_init_spd 128
ata_port_alloc 129
ata_finalize_port_ops 130
ata_port_detach 132
6. libata SCSI translation/emulation 133
ata_std_bios_param 133
ata_scsi_slave_config 134
ata_scsi_slave_destroy 135
__ata_change_queue_depth 136
ata_scsi_change_queue_depth 137
ata_scsi_queuecmd 138
ata_scsi_simulate 139
ata_sas_port_alloc 140
ata_sas_port_start 141
ata_sas_port_stop 142
ata_sas_async_probe 143
ata_sas_port_init 144
ata_sas_port_destroy 145
ata_sas_slave_configure 146
ata_sas_queuecmd 147
ata_get_identity 147
ata_cmd_ioctl 149
ata_task_ioctl 149
ata_scsi_qc_new 150
ata_dump_status 152
ata_to_sense_error 153
ata_gen_ata_sense 154
atapi_drain_needed 155
ata_scsi_start_stop_xlat 156
ata_scsi_flush_xlat 157
scsi_6_lba_len 158
scsi_10_lba_len 159
scsi_16_lba_len 160
ata_scsi_verify_xlat 161
ata_scsi_rw_xlat 163
ata_scsi_translate 164
ata_scsi_rbuf_get 165

ata_scsi_rbuf_put 166
ata_scsi_rbuf_fill 167
ata_scsiop_inq_std 168
ata_scsiop_inq_00 169
ata_scsiop_inq_80 170
ata_scsiop_inq_83 171
ata_scsiop_inq_89 173
ata_scsiop_noop 174
ata_msense_caching 174
ata_msense_ctl_mode 175
ata_msense_rw_recovery 176
ata_scsiop_mode_sense 177
ata_scsiop_read_cap 178
ata_scsiop_report_luns 179
atapi_xlat 180
ata_scsi_find_dev 181
ata_scsi_pass_thru 182
ata_get_xlat_func 183
ata_scsi_dump_cdb 184
ata_scsi_offline_dev 185
ata_scsi_remove_dev 186
ata_scsi_media_change_notify 187
ata_scsi_hotplug 188
ata_scsi_user_scan 189
ata_scsi_dev_rescan 190
7. ATA errors and exceptions 193
7.1. Exception categories 193
7.1.1. HSM violation 193
7.1.2. ATA/ATAPI device error (non-NCQ / non-CHECK CONDITION) 194
7.1.3. ATAPI device CHECK CONDITION 195
7.1.4. ATA device error (NCQ) 196
7.1.5. ATA bus error 196
7.1.6. PCI bus error 197
7.1.7. Late completion 197
7.1.8. Unknown error (timeout) 197
7.1.9. Hotplug and power management exceptions 197
7.2. EH recovery actions 198
7.2.1. Clearing error condition 198
7.2.2. Reset 198
7.2.3. Reconfigure transport 200

8. ata_piix Internals 201
ich_pata_cable_detect 201
piix_pata_prereset 201
piix_set_piomode 202
do_pata_set_dmamode 203
piix_set_dmamode 204
ich_set_dmamode 205
piix_check_450nx_errata 206
piix_init_one 207
9. sata_sil Internals 209
sil_set_mode 209
sil_dev_config 209
10. Thanks 213

Chapter 1. Introduction

libATA is a library used inside the Linux kernel to support ATA host controllers and devices. libATA provides an ATA driver API, class transports for ATA and ATAPI devices, and SCSI<->ATA translation for ATA devices according to the T10 SAT specification.

This Guide documents the libATA driver API, library functions, library internals, and a couple sample ATA low-level drivers.

Chapter 2. libata Driver API

struct ata_port_operations is defined for every low-level libata hardware driver, and it controls how the low-level driver interfaces with the ATA and SCSI layers.

FIS-based drivers will hook into the system with ->qc_prep() and ->qc_issue() high-level hooks. Hardware which behaves in a manner similar to PCI IDE hardware may utilize several generic helpers, defining at a bare minimum the bus I/O addresses of the ATA shadow register blocks.

2.1. struct ata_port_operations

2.1.1. Disable ATA port

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

Called from ata_bus_probe() error path, as well as when unregistering from the SCSI module (rmmod, hot unplug). This function should do whatever needs to be done to take the port out of use. In most cases, ata_port_disable() can be used as this hook.

Called from ata_bus_probe() on a failed probe. Called from ata_scsi_release().

2.1.2. Post-IDENTIFY device configuration

```
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.

This entry may be specified as NULL in ata_port_operations.

2.1.3. Set PIO/DMA mode

```
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 *);
unsigned int (*mode_filter) (struct ata_port *, struct ata_device *, unsig
```

Hooks called prior to the issue of SET FEATURES - XFER MODE command. The optional ->mode_filter() hook is called when libata has built a mask of the possible modes. This is passed to the ->mode_filter() function which should return a mask of valid modes after filtering those unsuitable due to hardware limits. It is not valid to use this interface to add modes.

dev->pio_mode and dev->dma_mode are guaranteed to be valid when ->set_piomode() and when ->set_dmamode() is called. The timings for any other drive sharing the cable will also be valid at this point. That is the library records the decisions for the modes of each drive on a channel before it attempts to set any of them.

->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.

2.1.4. Taskfile read/write

```
void (*sff_tf_load) (struct ata_port *ap, struct ata_taskfile *tf);
void (*sff_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. Most drivers for taskfile-based hardware (PIO or MMIO) use ata_sff_tf_load() and ata_sff_tf_read() for these hooks.

2.1.5. PIO data read/write

```
void (*sff_data_xfer) (struct ata_device *, unsigned char *, unsigned int,
```

All bmdma-style drivers must implement this hook. This is the low-level operation that actually copies the data bytes during a PIO data transfer. Typically the driver will choose one of ata_sff_data_xfer_noirq(), ata_sff_data_xfer(), or ata_sff_data_xfer32().

2.1.6. ATA command execute

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

causes an ATA command, previously loaded with ->tf_load(), to be initiated in hardware. Most drivers for taskfile-based hardware use ata_sff_exec_command() for this hook.

2.1.7. Per-cmd ATAPI DMA capabilities filter

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

Allow low-level driver to filter ATA PACKET commands, returning a status indicating whether or not it is OK to use DMA for the supplied PACKET command.

This hook may be specified as NULL, in which case libata will assume that atapi dma can be supported.

2.1.8. Read specific ATA shadow registers

```
u8   (*sff_check_status)(struct ata_port *ap);
u8   (*sff_check_altstatus)(struct ata_port *ap);
```

Reads the Status/AltStatus ATA shadow register from hardware. On some hardware, reading the Status register has the side effect of clearing the interrupt condition. Most drivers for taskfile-based hardware use ata_sff_check_status() for this hook.

2.1.9. Write specific ATA shadow register

```
void (*sff_set_devctl)(struct ata_port *ap, u8 ctl);
```

Write the device control ATA shadow register to the hardware. Most drivers don't need to define this.

2.1.10. Select ATA device on bus

```
void (*sff_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. This generally has no meaning on FIS-based devices.

Most drivers for taskfile-based hardware use ata_sff_dev_select() for this hook.

2.1.11. Private tuning method

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

By default libata performs drive and controller tuning in accordance with the ATA timing rules and also applies blacklists and cable limits. Some controllers need special handling and have custom tuning rules, typically raid controllers that use ATA commands but do not actually do drive timing.

Warning

This hook should not be used to replace the standard controller tuning logic when a controller has quirks. Replacing the default tuning logic in that case would bypass handling for drive and bridge quirks that may be important to data reliability. If a controller needs to filter the mode selection it should use the mode_filter hook instead.

2.1.12. Control PCI IDE BMDMA engine

```
void (*bmdma_setup) (struct ata_queued_cmd *qc);
void (*bmdma_start) (struct ata_queued_cmd *qc);
void (*bmdma_stop) (struct ata_port *ap);
u8   (*bmdma_status) (struct ata_port *ap);
```

When setting up an IDE BMDMA transaction, these hooks arm (->bmdma_setup), fire (->bmdma_start), and halt (->bmdma_stop) the hardware's DMA engine. ->bmdma_status is used to read the standard PCI IDE DMA Status register.

These hooks are typically either no-ops, or simply not implemented, in FIS-based drivers.

Most legacy IDE drivers use ata_bmdma_setup() for the bmdma_setup() hook. ata_bmdma_setup() will write the pointer to the PRD table to the IDE PRD Table

Address register, enable DMA in the DMA Command register, and call exec_command() to begin the transfer.

Most legacy IDE drivers use ata_bmdma_start() for the bmdma_start() hook. ata_bmdma_start() will write the ATA_DMA_START flag to the DMA Command register.

Many legacy IDE drivers use ata_bmdma_stop() for the bmdma_stop() hook. ata_bmdma_stop() clears the ATA_DMA_START flag in the DMA command register.

Many legacy IDE drivers use ata_bmdma_status() as the bmdma_status() hook.

2.1.13. High-level taskfile hooks

```
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 implement their own ->qc_issue.

ata_qc_issue_prot() calls ->tf_load(), ->bmdma_setup(), and ->bmdma_start() as necessary to initiate a transfer.

2.1.14. Exception and probe handling (EH)

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

Deprecated. Use ->error_handler() instead.

```
void (*freeze) (struct ata_port *ap);
void (*thaw) (struct ata_port *ap);
```

ata_port_freeze() is called when HSM violations or some other condition disrupts normal operation of the port. A frozen port is not allowed to perform any operation until the port is thawed, which usually follows a successful reset.

The optional ->freeze() callback can be used for freezing the port hardware-wise (e.g. mask interrupt and stop DMA engine). If a port cannot be frozen hardware-wise, the interrupt handler must ack and clear interrupts unconditionally while the port is frozen.

The optional ->thaw() callback is called to perform the opposite of ->freeze(): prepare the port for normal operation once again. Unmask interrupts, start DMA engine, etc.

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

->error_handler() is a driver's hook into probe, hotplug, and recovery and other exceptional conditions. The primary responsibility of an implementation is to call ata_do_eh() or ata_bmdma_drive_eh() with a set of EH hooks as arguments:

'prereset' hook (may be NULL) is called during an EH reset, before any other actions are taken.

'postreset' hook (may be NULL) is called after the EH reset is performed. Based on existing conditions, severity of the problem, and hardware capabilities,

Either 'softreset' (may be NULL) or 'hardreset' (may be NULL) will be called to perform the low-level EH reset.

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

Perform any hardware-specific actions necessary to finish processing after executing a probe-time or EH-time command via ata_exec_internal().

2.1.15. Hardware interrupt handling

```
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.

The second argument, dev_instance, should be cast to a pointer to struct ata_host_set.

Most legacy IDE drivers use ata_sff_interrupt() for the irq_handler hook, which scans all ports in the host_set, determines which queued command was active (if any), and calls ata_sff_host_intr(ap,qc).

Most legacy IDE drivers use ata_sff_irq_clear() for the irq_clear() hook, which simply clears the interrupt and error flags in the DMA status register.

2.1.16. SATA phy read/write

```
int (*scr_read) (struct ata_port *ap, unsigned int sc_reg,
   u32 *val);
int (*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. sc_reg is one of SCR_STATUS, SCR_CONTROL, SCR_ERROR, or SCR_ACTIVE.

2.1.17. Init and shutdown

```
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. Some drivers also use this entry point as a chance to allocate driver-private memory for ap->private_data.

Many drivers use ata_port_start() as this hook or call it from their own port_start() hooks. ata_port_start() allocates space for a legacy IDE PRD table and returns.

->port_stop() is called after ->host_stop(). Its sole function is to release DMA/memory resources, now that they are no longer actively being used. Many drivers also free driver-private data from port at this time.

->host_stop() is called after all ->port_stop() calls have completed. The hook must finalize hardware shutdown, release DMA and other resources, etc. This hook may be specified as NULL, in which case it is not called.

Chapter 3. Error handling

This chapter describes how errors are handled under libata. Readers are advised to read SCSI EH (Documentation/scsi/scsi_eh.txt) and ATA exceptions doc first.

3.1. Origins of commands

In libata, a command is represented with struct ata_queued_cmd or qc. qc's are preallocated during port initialization and repetitively used for command executions. Currently only one qc is allocated per port but yet-to-be-merged NCQ branch allocates one for each tag and maps each qc to NCQ tag 1-to-1.

libata commands can originate from two sources - libata itself and SCSI midlayer. libata internal commands are used for initialization and error handling. All normal blk requests and commands for SCSI emulation are passed as SCSI commands through queuecommand callback of SCSI host template.

3.2. How commands are issued

Internal commands

First, qc is allocated and initialized using ata_qc_new_init(). Although ata_qc_new_init() doesn't implement any wait or retry mechanism when qc is not available, internal commands are currently issued only during initialization and error recovery, so no other command is active and allocation is guaranteed to succeed.

Once allocated qc's taskfile is initialized for the command to be executed. qc currently has two mechanisms to notify completion. One is via qc->complete_fn() callback and the other is completion qc->waiting. qc->complete_fn() callback is the asynchronous path used by normal SCSI translated commands and qc->waiting is the synchronous (issuer sleeps in process context) path used by internal commands.

Once initialization is complete, host_set lock is acquired and the qc is issued.

SCSI commands

All libata drivers use ata_scsi_queuecmd() as hostt->queuecommand callback. scmds can either be simulated or translated. No qc is involved in processing a simulated scmd. The result is computed right away and the scmd is completed.

For a translated scmd, ata_qc_new_init() is invoked to allocate a qc and the scmd is translated into the qc. SCSI midlayer's completion notification function pointer is stored into qc->scsidone.

qc->complete_fn() callback is used for completion notification. ATA commands use ata_scsi_qc_complete() while ATAPI commands use atapi_qc_complete(). Both functions end up calling qc->scsidone to notify upper layer when the qc is finished. After translation is completed, the qc is issued with ata_qc_issue().

Note that SCSI midlayer invokes hostt->queuecommand while holding host_set lock, so all above occur while holding host_set lock.

3.3. How commands are processed

Depending on which protocol and which controller are used, commands are processed differently. For the purpose of discussion, a controller which uses taskfile interface and all standard callbacks is assumed.

Currently 6 ATA command protocols are used. They can be sorted into the following four categories according to how they are processed.

ATA NO DATA or DMA

ATA_PROT_NODATA and ATA_PROT_DMA fall into this category. These types of commands don't require any software intervention once issued. Device will raise interrupt on completion.

ATA PIO

ATA_PROT_PIO is in this category. libata currently implements PIO with polling. ATA_NIEN bit is set to turn off interrupt and pio_task on ata_wq performs polling and IO.

ATAPI NODATA or DMA

ATA_PROT_ATAPI_NODATA and ATA_PROT_ATAPI_DMA are in this category. packet_task is used to poll BSY bit after issuing PACKET command. Once BSY is turned off by the device, packet_task transfers CDB and hands off processing to interrupt handler.

ATAPI PIO

ATA_PROT_ATAPI is in this category. ATA_NIEN bit is set and, as in ATAPI NODATA or DMA, packet_task submits cdb. However, after submitting cdb, further processing (data transfer) is handed off to pio_task.

3.4. How commands are completed

Once issued, all qc's are either completed with ata_qc_complete() or time out. For commands which are handled by interrupts, ata_host_intr() invokes ata_qc_complete(), and, for PIO tasks, pio_task invokes ata_qc_complete(). In error cases, packet_task may also complete commands.

ata_qc_complete() does the following.

1. DMA memory is unmapped.
2. ATA_QCFLAG_ACTIVE is clared from qc->flags.
3. qc->complete_fn() callback is invoked. If the return value of the callback is not zero. Completion is short circuited and ata_qc_complete() returns.
4. __ata_qc_complete() is called, which does
   a. qc->flags is cleared to zero.
   b. ap->active_tag and qc->tag are poisoned.
   c. qc->waiting is claread & completed (in that order).
   d. qc is deallocated by clearing appropriate bit in ap->qactive.

So, it basically notifies upper layer and deallocates qc. One exception is short-circuit path in #3 which is used by atapi_qc_complete().

For all non-ATAPI commands, whether it fails or not, almost the same code path is taken and very little error handling takes place. A qc is completed with success status if it succeeded, with failed status otherwise.

However, failed ATAPI commands require more handling as REQUEST SENSE is needed to acquire sense data. If an ATAPI command fails, ata_qc_complete() is invoked with error status, which in turn invokes atapi_qc_complete() via qc->complete_fn() callback.

This makes atapi_qc_complete() set scmd->result to SAM_STAT_CHECK_CONDITION, complete the scmd and return 1. As the sense data is empty but scmd->result is CHECK CONDITION, SCSI midlayer will invoke EH for the scmd, and returning 1 makes ata_qc_complete() to return without deallocating the qc. This leads us to ata_scsi_error() with partially completed qc.

3.5. ata_scsi_error()

ata_scsi_error() is the current transportt->eh_strategy_handler() for libata. As discussed above, this will be entered in two cases - timeout and ATAPI error

completion. This function calls low level libata driver's eng_timeout() callback, the standard callback for which is ata_eng_timeout(). It checks if a qc is active and calls ata_qc_timeout() on the qc if so. Actual error handling occurs in ata_qc_timeout().

If EH is invoked for timeout, ata_qc_timeout() stops BMDMA and completes the qc. Note that as we're currently in EH, we cannot call scsi_done. As described in SCSI EH doc, a recovered scmd should be either retried with scsi_queue_insert() or finished with scsi_finish_command(). Here, we override qc->scsidone with scsi_finish_command() and calls ata_qc_complete().

If EH is invoked due to a failed ATAPI qc, the qc here is completed but not deallocated. The purpose of this half-completion is to use the qc as place holder to make EH code reach this place. This is a bit hackish, but it works.

Once control reaches here, the qc is deallocated by invoking __ata_qc_complete() explicitly. Then, internal qc for REQUEST SENSE is issued. Once sense data is acquired, scmd is finished by directly invoking scsi_finish_command() on the scmd. Note that as we already have completed and deallocated the qc which was associated with the scmd, we don't need to/cannot call ata_qc_complete() again.

3.6. Problems with the current EH

- Error representation is too crude. Currently any and all error conditions are represented with ATA STATUS and ERROR registers. Errors which aren't ATA device errors are treated as ATA device errors by setting ATA_ERR bit. Better error descriptor which can properly represent ATA and other errors/exceptions is needed.
- When handling timeouts, no action is taken to make device forget about the timed out command and ready for new commands.
- EH handling via ata_scsi_error() is not properly protected from usual command processing. On EH entrance, the device is not in quiescent state. Timed out commands may succeed or fail any time. pio_task and atapi_task may still be running.
- Too weak error recovery. Devices / controllers causing HSM mismatch errors and other errors quite often require reset to return to known state. Also, advanced error handling is necessary to support features like NCQ and hotplug.
- ATA errors are directly handled in the interrupt handler and PIO errors in pio_task. This is problematic for advanced error handling for the following reasons.

  First, advanced error handling often requires context and internal qc execution.

Second, even a simple failure (say, CRC error) needs information gathering and could trigger complex error handling (say, resetting & reconfiguring). Having multiple code paths to gather information, enter EH and trigger actions makes life painful.

Third, scattered EH code makes implementing low level drivers difficult. Low level drivers override libata callbacks. If EH is scattered over several places, each affected callbacks should perform its part of error handling. This can be error prone and painful.

Chapter 4. libata Library

ata_link_next

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_link_next` — link iteration helper

Synopsis

```
struct ata_link * ata_link_next (struct ata_link * link,
struct ata_port * ap, enum ata_link_iter_mode mode);
```

Arguments

link

the previous link, NULL to start

ap

ATA port containing links to iterate

mode

iteration mode, one of ATA_LITER_*

LOCKING

Host lock or EH context.

RETURNS

Pointer to the next link.

ata_dev_next

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_next` — device iteration helper

Synopsis

```
struct ata_device * ata_dev_next (struct ata_device * dev,
struct ata_link * link, enum ata_dev_iter_mode mode);
```

Arguments

dev

the previous device, NULL to start

link

ATA link containing devices to iterate

mode

iteration mode, one of ATA_DITER_*

LOCKING

Host lock or EH context.

RETURNS

Pointer to the next device.

atapi_cmd_type

LINUX

Kernel Hackers ManualOctober 2026

Name

`atapi_cmd_type` — Determine ATAPI command type from SCSI opcode

Synopsis

```
int atapi_cmd_type (u8 opcode);
```

Arguments

opcode

SCSI opcode

Description

Determine ATAPI command type from *opcode*.

LOCKING

None.

RETURNS

ATAPI_{READ|WRITE|READ_CD|PASS_THRU|MISC}

ata_tf_to_fis

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_tf_to_fis` — Convert ATA taskfile to SATA FIS structure

Synopsis

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

Arguments

tf

Taskfile to convert

pmp

Port multiplier port

is_cmd

This FIS is for command

fis

Buffer into which data will output

Description

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

LOCKING

Inherited from caller.

ata_tf_from_fis

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_tf_from_fis` — Convert SATA FIS to ATA taskfile

Synopsis

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

Arguments

fis

Buffer from which data will be input

tf

Taskfile to output

Description

Converts a serial ATA FIS structure to a standard ATA taskfile.

LOCKING

Inherited from caller.

ata_pack_xfermask

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_pack_xfermask` — Pack pio, mwdma and udma masks into xfer_mask

Synopsis

```
unsigned long ata_pack_xfermask (unsigned long pio_mask,
unsigned long mwdma_mask, unsigned long udma_mask);
```

Arguments

`pio_mask`

pio_mask

`mwdma_mask`

mwdma_mask

`udma_mask`

udma_mask

Description

Pack *pio_mask*, *mwdma_mask* and *udma_mask* into a single unsigned int xfer_mask.

LOCKING

None.

RETURNS

Packed xfer_mask.

ata_unpack_xfermask

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_unpack_xfermask` — Unpack xfer_mask into pio, mwdma and udma masks

Synopsis

```
void ata_unpack_xfermask (unsigned long xfer_mask, unsigned long * pio_mask, unsigned long * mwdma_mask, unsigned long * udma_mask);
```

Arguments

xfer_mask

xfer_mask to unpack

pio_mask

resulting pio_mask

mwdma_mask

resulting mwdma_mask

udma_mask

resulting udma_mask

Description

Unpack *xfer_mask* into *pio_mask*, *mwdma_mask* and *udma_mask*. Any NULL distination masks will be ignored.

ata_xfer_mask2mode

LINUX

Kernel Hackers ManualOctober 2026

Name

ata_xfer_mask2mode — Find matching XFER_* for the given xfer_mask

Synopsis

```
u8 ata_xfer_mask2mode (unsigned long xfer_mask);
```

Arguments

xfer_mask

xfer_mask of interest

Description

Return matching XFER_* value for *xfer_mask*. Only the highest bit of *xfer_mask* is considered.

LOCKING

None.

RETURNS

Matching XFER_* value, 0xff if no match found.

ata_xfer_mode2mask

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_xfer_mode2mask` — Find matching xfer_mask for XFER_*

Synopsis

```
unsigned long ata_xfer_mode2mask (u8 xfer_mode);
```

Arguments

xfer_mode

XFER_* of interest

Description

Return matching xfer_mask for *xfer_mode*.

LOCKING

None.

RETURNS

Matching xfer_mask, 0 if no match found.

ata_xfer_mode2shift

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_xfer_mode2shift` — Find matching xfer_shift for XFER_*

Synopsis

```
int ata_xfer_mode2shift (unsigned long xfer_mode);
```

Arguments

xfer_mode

XFER_* of interest

Description

Return matching xfer_shift for *xfer_mode*.

LOCKING

None.

RETURNS

Matching xfer_shift, -1 if no match found.

ata_mode_string

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_mode_string` — convert xfer_mask to string

Synopsis

```
const char * ata_mode_string (unsigned long xfer_mask);
```

Arguments

xfer_mask

mask of bits supported; only highest bit counts.

Description

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

LOCKING

None.

RETURNS

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

ata_dev_classify

LINUX

Kernel Hackers ManualOctober 2026

Name

ata_dev_classify — determine device type based on ATA-spec signature

Synopsis

```
unsigned int ata_dev_classify (const 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`, `ATA_DEV_PMP` or `ATA_DEV_UNKNOWN` the event of failure.

ata_id_string

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_id_string` — Convert IDENTIFY DEVICE page into string

Synopsis

```
void ata_id_string (const 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_id_c_string

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_id_c_string` — Convert IDENTIFY DEVICE page into C string

Synopsis

```
void ata_id_c_string (const 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 odd number.

Description

This function is identical to ata_id_string except that it trims trailing spaces and terminates the resulting string with null. *len* must be actual maximum length (even number) + 1.

LOCKING

caller.

ata_id_xfermask

LINUX

Name

`ata_id_xfermask` — Compute xfermask from the given IDENTIFY data

Synopsis

```
unsigned long ata_id_xfermask (const u16 * id);
```

Arguments

id

IDENTIFY data to compute xfer mask from

Description

Compute the xfermask for this device. This is not as trivial as it seems if we must consider early devices correctly.

FIXME

pre IDE drive timing (do we care ?).

LOCKING

None.

RETURNS

Computed xfermask

ata_pio_need_iordy

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_pio_need_iordy` — check if iordy needed

Synopsis

```
unsigned int ata_pio_need_iordy (const struct ata_device *
adev);
```

Arguments

adev

ATA device

Description

Check if the current speed of the device requires IORDY. Used by various controllers for chip configuration.

ata_do_dev_read_id

LINUX

Name

`ata_do_dev_read_id` — default ID read method

Synopsis

```
unsigned int ata_do_dev_read_id (struct ata_device * dev,
struct ata_taskfile * tf, u16 * id);
```

Arguments

dev

device

tf

proposed taskfile

id

data buffer

Description

Issue the identify taskfile and hand back the buffer containing identify data. For some RAID controllers and for pre ATA devices this function is wrapped or replaced by the driver

ata_cable_40wire

LINUX

Name

`ata_cable_40wire` — return 40 wire cable type

Synopsis

```
int ata_cable_40wire (struct ata_port * ap);
```

Arguments

ap

port

Description

Helper method for drivers which want to hardwire 40 wire cable detection.

ata_cable_80wire

LINUX

Name

`ata_cable_80wire` — return 80 wire cable type

Synopsis

```
int ata_cable_80wire (struct ata_port * ap);
```

Arguments

ap

port

Description

Helper method for drivers which want to hardwire 80 wire cable detection.

ata_cable_unknown

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_cable_unknown` — return unknown PATA cable.

Synopsis

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

Arguments

ap

port

Description

Helper method for drivers which have no PATA cable detection.

ata_cable_ignore

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_cable_ignore` — return ignored PATA cable.

Synopsis

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

Arguments

ap

port

Description

Helper method for drivers which don't use cable type to limit transfer mode.

ata_cable_sata

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_cable_sata` — return SATA cable type

Synopsis

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

Arguments

ap

port

Description

Helper method for drivers which have SATA cables

ata_dev_pair

LINUX

Name

`ata_dev_pair` — return other device on cable

Synopsis

```
struct ata_device * ata_dev_pair (struct ata_device * adev);
```

Arguments

adev

device

Description

Obtain the other device on the same cable, or if none is present NULL is returned

sata_set_spd

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_set_spd` — set SATA spd according to spd limit

Synopsis

```
int sata_set_spd (struct ata_link * link);
```

Arguments

link

Link to set SATA spd for

Description

Set SATA spd of *link* according to sata_spd_limit.

LOCKING

Inherited from caller.

RETURNS

0 if spd doesn't need to be changed, 1 if spd has been changed. Negative errno if SCR registers are inaccessible.

ata_timing_cycle2mode

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_timing_cycle2mode` — find xfer mode for the specified cycle duration

Synopsis

```
u8 ata_timing_cycle2mode (unsigned int xfer_shift, int cycle);
```

Arguments

xfer_shift

ATA_SHIFT_* value for transfer type to examine.

cycle

cycle duration in ns

Description

Return matching xfer mode for *cycle*. The returned mode is of the transfer type specified by *xfer_shift*. If *cycle* is too slow for *xfer_shift*, 0xff is returned. If *cycle* is faster than the fastest known mode, the fasted mode is returned.

LOCKING

None.

RETURNS

Matching xfer_mode, 0xff if no match found.

ata_do_set_mode

LINUX

Name

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

Synopsis

```
int ata_do_set_mode (struct ata_link * link, struct ata_device
** r_failed_dev);
```

Arguments

`link`

link on which timings will be programmed

`r_failed_dev`

out parameter for failed device

Description

Standard implementation of the function used to tune and set ATA device disk transfer mode (PIO3, UDMA6, etc.). If `ata_dev_set_mode` fails, pointer to the failing device is returned in *`r_failed_dev`*.

LOCKING

PCI/etc. bus probe sem.

RETURNS

0 on success, negative errno otherwise

ata_wait_after_reset

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_wait_after_reset` — wait for link to become ready after reset

Synopsis

```
int ata_wait_after_reset (struct ata_link * link, unsigned
long deadline, int (*check_ready) (struct ata_link *link));
```

Arguments

link

link to be waited on

deadline

deadline jiffies for the operation

check_ready

callback to check link readiness

Description

Wait for *link* to become ready after reset.

LOCKING

EH context.

RETURNS

0 if `linke` is ready before `deadline`; otherwise, -errno.

sata_link_debounce

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_link_debounce` — debounce SATA phy status

Synopsis

```
int sata_link_debounce (struct ata_link * link, const unsigned long * params, unsigned long deadline);
```

Arguments

link

ATA link to debounce SATA phy status for

params

timing parameters { interval, duratinon, timeout } in msec

deadline

deadline jiffies for the operation

Description

Make sure SStatus of `link` reaches stable state, determined by holding the same value where DET is not 1 for `duration` polled every `interval`, before `timeout`. Timeout constraints the beginning of the stable state. Because DET gets stuck at 1 on some controllers after hot unplugging, this functions waits until timeout then returns 0 if DET is stable at 1.

`timeout` is further limited by `deadline`. The sooner of the two is used.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno on failure.

sata_link_resume

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_link_resume` — resume SATA link

Synopsis

```
int sata_link_resume (struct ata_link * link, const unsigned
long * params, unsigned long deadline);
```

Arguments

link

ATA link to resume SATA

params

timing parameters { interval, duratinon, timeout } in msec

deadline

deadline jiffies for the operation

Description

Resume SATA phy *link* and debounce it.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno on failure.

sata_link_scr_lpm

LINUX

Kernel Hackers ManualOctober 2026

Name

sata_link_scr_lpm — manipulate SControl IPM and SPM fields

Synopsis

```
int sata_link_scr_lpm (struct ata_link * link, enum
ata_lpm_policy policy, bool spm_wakeup);
```

Arguments

link

ATA link to manipulate SControl for

policy

LPM policy to configure

spm_wakeup

initiate LPM transition to active state

Description

Manipulate the IPM field of the SControl register of *link* according to *policy*. If *policy* is ATA_LPM_MAX_POWER and *spm_wakeup* is `true`, the SPM field is manipulated to wake up the link. This function also clears PHYRDY_CHG before returning.

LOCKING

EH context.

RETURNS

0 on succes, -errno otherwise.

ata_std_prereset

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_std_prereset` — prepare for reset

Synopsis

```
int ata_std_prereset (struct ata_link * link, unsigned long
deadline);
```

Arguments

`link`

ATA link to be reset

`deadline`

deadline jiffies for the operation

Description

`link` is about to be reset. Initialize it. Failure from prereset makes libata abort whole reset sequence and give up that port, so prereset should be best-effort. It does its best to prepare for reset sequence but if things go wrong, it should just whine, not fail.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno otherwise.

sata_link_hardreset

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_link_hardreset` — reset link via SATA phy reset

Synopsis

```
int sata_link_hardreset (struct ata_link * link, const
unsigned long * timing, unsigned long deadline, bool * online,
int (*check_ready) (struct ata_link *));
```

Arguments

link

link to reset

timing

timing parameters { interval, duratinon, timeout } in msec

deadline

deadline jiffies for the operation

online

optional out parameter indicating link onlineness

check_ready

optional callback to check link readiness

Description

SATA phy-reset *link* using DET bits of SControl register. After hardreset, link readiness is waited upon using ata_wait_ready if *check_ready* is specified. LLDs are allowed to not specify *check_ready* and wait itself after this function returns. Device classification is LLD's responsibility.

**online* is set to one iff reset succeeded and *link* is online after reset.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno otherwise.

sata_std_hardreset

LINUX

Kernel Hackers ManualOctober 2026

Name

sata_std_hardreset — COMRESET w/o waiting or classification

Synopsis

```
int sata_std_hardreset (struct ata_link * link, unsigned int *
class, unsigned long deadline);
```

Arguments

link

link to reset

class

resulting class of attached device

deadline

deadline jiffies for the operation

Description

Standard SATA COMRESET w/o waiting or classification.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 if link offline, -EAGAIN if link online, -errno on errors.

ata_std_postreset

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_std_postreset` — standard postreset callback

Synopsis

```
void ata_std_postreset (struct ata_link * link, unsigned int *
classes);
```

Arguments

link

the target ata_link

classes

classes of attached devices

Description

This function is invoked after a successful reset. Note that the device might have been reset more than once using different reset methods before postreset is invoked.

LOCKING

Kernel thread context (may sleep)

ata_dev_set_feature

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_set_feature` — Issue SET FEATURES - SATA FEATURES

Synopsis

```
unsigned int ata_dev_set_feature (struct ata_device * dev, u8
enable, u8 feature);
```

Arguments

dev

Device to which command will be sent

enable

Whether to enable or disable the feature

feature

The sector count represents the feature to set

Description

Issue SET FEATURES - SATA FEATURES command to device *dev* on port *ap* with sector count

LOCKING

PCI/etc. bus probe sem.

RETURNS

0 on success, AC_ERR_* mask otherwise.

ata_std_qc_defer

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_std_qc_defer` — Check whether a qc needs to be deferred

Synopsis

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

Arguments

qc

ATA command in question

Description

Non-NCQ commands cannot run with any other command, NCQ or not. As upper layer only knows the queue depth, we are responsible for maintaining exclusion. This function checks whether a new command *qc* can be issued.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

ATA_DEFER_* if deferring is needed, 0 otherwise.

ata_sg_init

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sg_init` — Associate command with scatter-gather table.

Synopsis

```
void ata_sg_init (struct ata_queued_cmd * qc, struct
scatterlist * sg, unsigned int n_elem);
```

Arguments

qc

Command to be associated

sg

Scatter-gather table.

n_elem

Number of elements in s/g table.

Description

Initialize the data-related elements of queued_cmd *qc* to point to a scatter-gather table *sg*, containing *n_elem* elements.

LOCKING

spin_lock_irqsave(host lock)

ata_qc_complete

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_qc_complete` — Complete an active ATA command

Synopsis

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

Arguments

qc

Command to complete

Description

Indicate to the mid and upper layers that an ATA command has completed, with either an ok or not-ok status.

Refrain from calling this function multiple times when successfully completing multiple NCQ commands. `ata_qc_complete_multiple` should be used instead, which will properly update IRQ expect state.

LOCKING

spin_lock_irqsave(host lock)

ata_qc_complete_multiple

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_qc_complete_multiple` — Complete multiple qcs successfully

Synopsis

```
int ata_qc_complete_multiple (struct ata_port * ap, u32
qc_active);
```

Arguments

ap

port in question

qc_active

new qc_active mask

Description

Complete in-flight commands. This functions is meant to be called from low-level driver's interrupt routine to complete requests normally. ap->qc_active and *qc_active* is compared and commands are completed accordingly.

Always use this function when completing multiple NCQ commands from IRQ handlers instead of calling `ata_qc_complete` multiple times to keep IRQ expect status properly in sync.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Number of completed commands on success, -errno otherwise.

sata_scr_valid

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_scr_valid` — test whether SCRs are accessible

Synopsis

```
int sata_scr_valid (struct ata_link * link);
```

Arguments

`link`

ATA link to test SCR accessibility for

Description

Test whether SCRs are accessible for *`link`*.

LOCKING

None.

RETURNS

1 if SCRs are accessible, 0 otherwise.

sata_scr_read

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_scr_read` — read SCR register of the specified port

Synopsis

```
int sata_scr_read (struct ata_link * link, int reg, u32 *
val);
```

Arguments

link

ATA link to read SCR for

reg

SCR to read

val

Place to store read value

Description

Read SCR register *reg* of *link* into **val*. This function is guaranteed to succeed if *link* is ap->link, the cable type of the port is SATA and the port implements ->scr_read.

LOCKING

None if *link* is ap->link. Kernel thread context otherwise.

RETURNS

0 on success, negative errno on failure.

sata_scr_write

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_scr_write` — write SCR register of the specified port

Synopsis

```
int sata_scr_write (struct ata_link * link, int reg, u32 val);
```

Arguments

link

ATA link to write SCR for

reg

SCR to write

val

value to write

Description

Write *val* to SCR register *reg* of *link*. This function is guaranteed to succeed if *link* is ap->link, the cable type of the port is SATA and the port implements ->scr_read.

LOCKING

None if *link* is ap->link. Kernel thread context otherwise.

RETURNS

0 on success, negative errno on failure.

sata_scr_write_flush

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_scr_write_flush` — write SCR register of the specified port and flush

Synopsis

```
int sata_scr_write_flush (struct ata_link * link, int reg, u32 val);
```

Arguments

link

ATA link to write SCR for

reg

SCR to write

val

value to write

Description

This function is identical to `sata_scr_write` except that this function performs flush after writing to the register.

LOCKING

None if *link* is ap->link. Kernel thread context otherwise.

RETURNS

0 on success, negative errno on failure.

ata_link_online

LINUX

Name

`ata_link_online` — test whether the given link is online

Synopsis

```
bool ata_link_online (struct ata_link * link);
```

Arguments

`link`

ATA link to test

Description

Test whether *`link`* is online. This is identical to `ata_phys_link_online` when there's no slave link. When there's a slave link, this function should only be called on the master link and will return true if any of M/S links is online.

LOCKING

None.

RETURNS

True if the port online status is available and online.

ata_link_offline

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_link_offline` — test whether the given link is offline

Synopsis

```
bool ata_link_offline (struct ata_link * link);
```

Arguments

link

ATA link to test

Description

Test whether *link* is offline. This is identical to `ata_phys_link_offline` when there's no slave link. When there's a slave link, this function should only be called on the master link and will return true if both M/S links are offline.

LOCKING

None.

RETURNS

True if the port offline status is available and offline.

ata_host_suspend

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_suspend` — suspend host

Synopsis

```
int ata_host_suspend (struct ata_host * host, pm_message_t
mesg);
```

Arguments

host

host to suspend

mesg

PM message

Description

Suspend *host*. Actual operation is performed by port suspend.

ata_host_resume

LINUX

Name

`ata_host_resume` — resume host

Synopsis

```
void ata_host_resume (struct ata_host * host);
```

Arguments

host

host to resume

Description

Resume *host*. Actual operation is performed by port resume.

ata_host_alloc

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_alloc` — allocate and init basic ATA host resources

Synopsis

```
struct ata_host * ata_host_alloc (struct device * dev, int
max_ports);
```

Arguments

dev

generic device this host is associated with

max_ports

maximum number of ATA ports associated with this host

Description

Allocate and initialize basic ATA host resources. LLD calls this function to allocate a host, initializes it fully and attaches it using `ata_host_register`.

max_ports ports are allocated and host->n_ports is initialized to *max_ports*. The caller is allowed to decrease host->n_ports before calling `ata_host_register`. The unused ports will be automatically freed on registration.

RETURNS

Allocate ATA host on success, NULL on failure.

LOCKING

Inherited from calling layer (may sleep).

ata_host_alloc_pinfo

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_alloc_pinfo` — alloc host and init with port_info array

Synopsis

```
struct ata_host * ata_host_alloc_pinfo (struct device * dev,
const struct ata_port_info *const * ppi, int n_ports);
```

Arguments

dev

generic device this host is associated with

ppi

array of ATA port_info to initialize host with

n_ports

number of ATA ports attached to this host

Description

Allocate ATA host and initialize with info from *ppi*. If NULL terminated, *ppi* may contain fewer entries than *n_ports*. The last entry will be used for the remaining ports.

RETURNS

Allocate ATA host on success, NULL on failure.

LOCKING

Inherited from calling layer (may sleep).

ata_slave_link_init

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_slave_link_init` — initialize slave link

Synopsis

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

Arguments

ap

port to initialize slave link for

Description

Create and initialize slave link for *ap*. This enables slave link handling on the port.

In libata, a port contains links and a link contains devices. There is single host link but if a PMP is attached to it, there can be multiple fan-out links. On SATA, there's usually a single device connected to a link but PATA and SATA controllers emulating TF based interface can have two - master and slave.

However, there are a few controllers which don't fit into this abstraction too well - SATA controllers which emulate TF interface with both master and slave devices

but also have separate SCR register sets for each device. These controllers need separate links for physical link handling (e.g. onlineness, link speed) but should be treated like a traditional M/S controller for everything else (e.g. command issue, softreset).

slave_link is libata's way of handling this class of controllers without impacting core layer too much. For anything other than physical link handling, the default host link is used for both master and slave. For physical link handling, separate `ap`->slave_link is used. All dirty details are implemented inside libata core layer. From LLD's POV, the only difference is that prereset, hardreset and postreset are called once more for the slave link, so the reset sequence looks like the following.

prereset(M) -> prereset(S) -> hardreset(M) -> hardreset(S) -> softreset(M) -> postreset(M) -> postreset(S)

Note that softreset is called only for the master. Softreset resets both M/S by definition, so SRST on master should handle both (the standard method will work just fine).

LOCKING

Should be called before host is registered.

RETURNS

0 on success, -errno on failure.

ata_host_start

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_start` — start and freeze ports of an ATA host

Synopsis

```
int ata_host_start (struct ata_host * host);
```

Arguments

host

ATA host to start ports for

Description

Start and then freeze ports of *host*. Started status is recorded in host->flags, so this function can be called multiple times. Ports are guaranteed to get started only once. If host->ops isn't initialized yet, its set to the first non-dummy port ops.

LOCKING

Inherited from calling layer (may sleep).

RETURNS

0 if all ports are started successfully, -errno otherwise.

ata_host_init

LINUX

Kernel Hackers ManualOctober 2026

Name

ata_host_init — Initialize a host struct for sas (ipr, libsas)

Synopsis

```
void ata_host_init (struct ata_host * host, struct device *
dev, struct ata_port_operations * ops);
```

Arguments

host

host to initialize

dev

device host is attached to

ops

port_ops

ata_host_register

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_register` — register initialized ATA host

Synopsis

```
int ata_host_register (struct ata_host * host, struct
scsi_host_template * sht);
```

Arguments

host

ATA host to register

sht

template for SCSI host

Description

Register initialized ATA host. *host* is allocated using `ata_host_alloc` and fully initialized by LLD. This function starts ports, registers *host* with ATA and SCSI layers and probe registered devices.

LOCKING

Inherited from calling layer (may sleep).

RETURNS

0 on success, -errno otherwise.

ata_host_activate

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_activate` — start host, request IRQ and register it

Synopsis

```
int ata_host_activate (struct ata_host * host, int irq,
irq_handler_t irq_handler, unsigned long irq_flags, struct
scsi_host_template * sht);
```

Arguments

host

target ATA host

irq

IRQ to request

irq_handler

irq_handler used when requesting IRQ

irq_flags

irq_flags used when requesting IRQ

sht

scsi_host_template to use when registering the host

Description

After allocating an ATA host and initializing it, most libata LLDs perform three steps to activate the host - start host, request IRQ and register it. This helper takes necessasry arguments and performs the three steps in one go.

An invalid IRQ skips the IRQ registration and expects the host to have set polling mode on the port. In this case, *irq_handler* should be NULL.

LOCKING

Inherited from calling layer (may sleep).

RETURNS

0 on success, -errno otherwise.

ata_host_detach

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_host_detach` — Detach all ports of an ATA host

Synopsis

```
void ata_host_detach (struct ata_host * host);
```

Arguments

host

Host to detach

Description

Detach all ports of *host*.

LOCKING

Kernel thread context (may sleep).

ata_pci_remove_one

LINUX

Kernel Hackers ManualOctober 2026

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. Detach all ports. Resource release is handled via devres.

LOCKING

Inherited from PCI layer (may sleep).

ata_msleep

LINUX

Name

`ata_msleep` — ATA EH owner aware msleep

Synopsis

```
void ata_msleep (struct ata_port * ap, unsigned int msecs);
```

Arguments

ap

ATA port to attribute the sleep to

msecs

duration to sleep in milliseconds

Description

Sleeps *msecs*. If the current task is owner of *ap*'s EH, the ownership is released before going to sleep and reacquired after the sleep is complete. IOW, other ports sharing the *ap*->host will be allowed to own the EH while this task is sleeping.

LOCKING

Might sleep.

ata_wait_register

LINUX

Name

`ata_wait_register` — wait until register value changes

Synopsis

```
u32 ata_wait_register (struct ata_port * ap, void __iomem *
reg, u32 mask, u32 val, unsigned long interval, unsigned long
timeout);
```

Arguments

ap

ATA port to wait register for, can be NULL

reg

IO-mapped register

mask

Mask to apply to read register value

val

Wait condition

interval

polling interval in milliseconds

timeout

timeout in milliseconds

Description

Waiting for some bits of register to change is a common operation for ATA controllers. This function reads 32bit LE IO-mapped register *reg* and tests for the

following condition.

(*`reg` & mask) != val

If the condition is met, it returns; otherwise, the process is repeated after `interval_msec` until timeout.

LOCKING

Kernel thread context (may sleep)

RETURNS

The final register value.

Chapter 5. libata Core Internals

ata_dev_phys_link

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_phys_link` — find physical link for a device

Synopsis

```
struct ata_link * ata_dev_phys_link (struct ata_device * dev);
```

Arguments

dev

ATA device to look up physical link for

Description

Look up physical link which *dev* is attached to. Note that this is different from *dev*->link only when *dev* is on slave link. For all other cases, it's the same as *dev*->link.

LOCKING

Don't care.

RETURNS

Pointer to the found physical link.

ata_force_cbl

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_force_cbl` — force cable type according to libata.force

Synopsis

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

Arguments

ap

ATA port of interest

Description

Force cable type according to libata.force and whine about it. The last entry which has matching port number is used, so it can be specified as part of device force parameters. For example, both “a:40c,1.00:udma4” and “1.00:40c,udma4” have the same effect.

LOCKING

EH context.

ata_force_link_limits

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_force_link_limits` — force link limits according to libata.force

Synopsis

```
void ata_force_link_limits (struct ata_link * link);
```

Arguments

link

ATA link of interest

Description

Force link flags and SATA spd limit according to libata.force and whine about it. When only the port part is specified (e.g. 1:), the limit applies to all links connected to both the host link and all fan-out ports connected via PMP. If the device part is specified as 0 (e.g. 1.00:), it specifies the first fan-out link not the host link. Device number 15 always points to the host link whether PMP is attached or not. If the controller has slave link, device number 16 points to it.

LOCKING

EH context.

ata_force_xfermask

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_force_xfermask` — force xfermask according to libata.force

Synopsis

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

Arguments

dev

ATA device of interest

Description

Force xfer_mask according to libata.force and whine about it. For consistency with link selection, device number 15 selects the first device connected to the host link.

LOCKING

EH context.

ata_force_horkage

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_force_horkage` — force horkage according to libata.force

Synopsis

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

Arguments

dev

ATA device of interest

Description

Force horkage according to libata.force and whine about it. For consistency with link selection, device number 15 selects the first device connected to the host link.

LOCKING

EH context.

ata_rwcmd_protocol

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
int ata_rwcmd_protocol (struct ata_taskfile * tf, struct
ata_device * dev);
```

Arguments

tf

command to examine and configure

dev

device tf belongs to

Description

Examine the device configuration and tf->flags to calculate the proper read/write commands and protocol to use.

LOCKING

caller.

ata_tf_read_block

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_tf_read_block` — Read block address from ATA taskfile

Synopsis

```
u64 ata_tf_read_block (struct ata_taskfile * tf, struct
ata_device * dev);
```

Arguments

tf

ATA taskfile of interest

dev

ATA device *tf* belongs to

LOCKING

None.

Read block address from *tf*. This function can handle all three address formats - LBA, LBA48 and CHS. tf->protocol and flags select the address format to use.

RETURNS

Block address read from *tf*.

ata_build_rw_tf

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_build_rw_tf` — Build ATA taskfile for given read/write request

Synopsis

```
int ata_build_rw_tf (struct ata_taskfile * tf, struct
ata_device * dev, u64 block, u32 n_block, unsigned int
tf_flags, unsigned int tag);
```

Arguments

tf

Target ATA taskfile

dev

ATA device *tf* belongs to

block

Block address

n_block

Number of blocks

tf_flags

RW/FUA etc...

tag

tag

LOCKING

None.

Build ATA taskfile *tf* for read/write request described by *block*, *n_block*, *tf_flags* and *tag* on *dev*.

RETURNS

0 on success, -ERANGE if the request is too large for *dev*, -EINVAL if the request is invalid.

ata_read_native_max_address

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_read_native_max_address` — Read native max address

Synopsis

```
int ata_read_native_max_address (struct ata_device * dev, u64
* max_sectors);
```

Arguments

dev

target device

max_sectors

out parameter for the result native max address

Description

Perform an LBA48 or LBA28 native size query upon the device in question.

RETURNS

0 on success, -EACCES if command is aborted by the drive. -EIO on other errors.

ata_set_max_sectors

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_set_max_sectors` — Set max sectors

Synopsis

```
int ata_set_max_sectors (struct ata_device * dev, u64
new_sectors);
```

Arguments

dev

target device

new_sectors

new max sectors value to set for the device

Description

Set max sectors of *dev* to *new_sectors*.

RETURNS

0 on success, -EACCES if command is aborted or denied (due to previous non-volatile SET_MAX) by the drive. -EIO on other errors.

ata_hpa_resize

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_hpa_resize` — Resize a device with an HPA set

Synopsis

```
int ata_hpa_resize (struct ata_device * dev);
```

Arguments

dev

Device to resize

Description

Read the size of an LBA28 or LBA48 disk with HPA features and resize it if required to the full size of the media. The caller must check the drive has the HPA feature set enabled.

RETURNS

0 on success, -errno on failure.

ata_dump_id

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dump_id` — IDENTIFY DEVICE info debugging output

Synopsis

```
void ata_dump_id (const u16 * id);
```

Arguments

id

IDENTIFY DEVICE page to dump

Description

Dump selected 16-bit words from the given IDENTIFY DEVICE page.

LOCKING

caller.

ata_exec_internal_sg

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_exec_internal_sg` — execute libata internal command

Synopsis

```
unsigned ata_exec_internal_sg (struct ata_device * dev, struct
ata_taskfile * tf, const u8 * cdb, int dma_dir, struct
scatterlist * sgl, unsigned int n_elem, unsigned long
timeout);
```

Arguments

dev

Device to which the command is sent

tf

Taskfile registers for the command and the result

cdb

CDB for packet command

dma_dir

Data tranfer direction of the command

`sgl`

sg list for the data buffer of the command

`n_elem`

Number of sg entries

`timeout`

Timeout in msecs (0 for default)

Description

Executes libata internal command with timeout. `tf` contains command on entry and result on return. Timeout and error conditions are reported via return value. No recovery action is taken after a command times out. It's caller's duty to clean up after timeout.

LOCKING

None. Should be called with kernel context, might sleep.

RETURNS

Zero on success, AC_ERR_* mask on failure

ata_exec_internal

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_exec_internal` — execute libata internal command

Synopsis

```
unsigned ata_exec_internal (struct ata_device * dev, struct
ata_taskfile * tf, const u8 * cdb, int dma_dir, void * buf,
unsigned int buflen, unsigned long timeout);
```

Arguments

dev

Device to which the command is sent

tf

Taskfile registers for the command and the result

cdb

CDB for packet command

dma_dir

Data tranfer direction of the command

buf

Data buffer of the command

buflen

Length of data buffer

timeout

Timeout in msecs (0 for default)

Description

Wrapper around `ata_exec_internal_sg` which takes simple buffer instead of sg list.

LOCKING

None. Should be called with kernel context, might sleep.

RETURNS

Zero on success, AC_ERR_* mask on failure

ata_do_simple_cmd

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_do_simple_cmd` — execute simple internal command

Synopsis

```
unsigned int ata_do_simple_cmd (struct ata_device * dev, u8
cmd);
```

Arguments

dev

Device to which the command is sent

cmd

Opcode to execute

Description

Execute a 'simple' command, that only consists of the opcode 'cmd' itself, without filling any other registers

LOCKING

Kernel thread context (may sleep).

RETURNS

Zero on success, AC_ERR_* mask on failure

ata_pio_mask_no_iordy

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_pio_mask_no_iordy` — Return the non IORDY mask

Synopsis

```
u32 ata_pio_mask_no_iordy (const struct ata_device * adev);
```

Arguments

adev

ATA device

Description

Compute the highest mode possible if we are not using iordy. Return -1 if no iordy mode is available.

ata_dev_read_id

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_read_id` — Read ID data from the specified device

Synopsis

```
int ata_dev_read_id (struct ata_device * dev, unsigned int *
p_class, unsigned int flags, u16 * id);
```

Arguments

dev

target device

p_class

pointer to class of the target device (may be changed)

flags

ATA_READID_* flags

id

buffer to read IDENTIFY data into

Description

Read ID data from the specified device. ATA_CMD_ID_ATA is performed on ATA devices and ATA_CMD_ID_ATAPI on ATAPI devices. This function also issues ATA_CMD_INIT_DEV_PARAMS for pre-ATA4 drives.

FIXME

ATA_CMD_ID_ATA is optional for early drives and right now we abort if we hit that case.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno otherwise.

ata_dev_configure

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_configure` — Configure the specified ATA/ATAPI device

Synopsis

```
int ata_dev_configure (struct ata_device * dev);
```

Arguments

`dev`

Target device to configure

Description

Configure `dev` according to `dev`->id. Generic and low-level driver specific fixups are also applied.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, -errno otherwise

ata_bus_probe

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_bus_probe` — Reset and probe ATA bus

Synopsis

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

Arguments

ap

Bus to probe

Description

Master ATA bus probing function. Initiates a hardware-dependent bus reset, then attempts to identify any devices found on the bus.

LOCKING

PCI/etc. bus probe sem.

RETURNS

Zero on success, negative errno otherwise.

sata_print_link_status

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_print_link_status` — Print SATA link status

Synopsis

```
void sata_print_link_status (struct ata_link * link);
```

Arguments

link

SATA link to printk link status about

Description

This function prints link speed and status of a SATA link.

LOCKING

None.

sata_down_spd_limit

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_down_spd_limit` — adjust SATA spd limit downward

Synopsis

```
int sata_down_spd_limit (struct ata_link * link, u32
spd_limit);
```

Arguments

link

Link to adjust SATA spd limit for

spd_limit

Additional limit

Description

Adjust SATA spd limit of *link* downward. Note that this function only adjusts the limit. The change must be applied using `sata_set_spd`.

If *spd_limit* is non-zero, the speed is limited to equal to or lower than *spd_limit* if such speed is supported. If *spd_limit* is slower than any supported speed, only the lowest supported speed is allowed.

LOCKING

Inherited from caller.

RETURNS

0 on success, negative errno on failure

sata_set_spd_needed

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_set_spd_needed` — is SATA spd configuration needed

Synopsis

```
int sata_set_spd_needed (struct ata_link * link);
```

Arguments

link

Link in question

Description

Test whether the spd limit in SControl matches *link*->sata_spd_limit. This function is used to determine whether hardreset is necessary to apply SATA spd configuration.

LOCKING

Inherited from caller.

RETURNS

1 if SATA spd configuration is needed, 0 otherwise.

ata_down_xfermask_limit

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_down_xfermask_limit` — adjust dev xfer masks downward

Synopsis

```
int ata_down_xfermask_limit (struct ata_device * dev, unsigned int sel);
```

Arguments

dev

Device to adjust xfer masks

sel

ATA_DNXFER_* selector

Description

Adjust xfer masks of `dev` downward. Note that this function does not apply the change. Invoking `ata_set_mode` afterwards will apply the limit.

LOCKING

Inherited from caller.

RETURNS

0 on success, negative errno on failure

ata_wait_ready

LINUX

Name

`ata_wait_ready` — wait for link to become ready

Synopsis

```
int ata_wait_ready (struct ata_link * link, unsigned long
deadline, int (*check_ready) (struct ata_link *link));
```

Arguments

`link`

link to be waited on

`deadline`

deadline jiffies for the operation

`check_ready`

callback to check link readiness

Description

Wait for *`link`* to become ready. *`check_ready`* should return positive number if *`link`* is ready, 0 if it isn't, -ENODEV if link doesn't seem to be occupied, other errno for other error conditions.

Transient -ENODEV conditions are allowed for ATA_TMOUT_FF_WAIT.

LOCKING

EH context.

RETURNS

0 if `linke` is ready before `deadline`; otherwise, -errno.

ata_dev_same_device

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_same_device` — Determine whether new ID matches configured device

Synopsis

```
int ata_dev_same_device (struct ata_device * dev, unsigned int
new_class, const u16 * new_id);
```

Arguments

dev

device to compare against

new_class

class of the new device

new_id

IDENTIFY page of the new device

Description

Compare `new_class` and `new_id` against `dev` and determine whether `dev` is the device indicated by `new_class` and `new_id`.

LOCKING

None.

RETURNS

1 if `dev` matches `new_class` and `new_id`, 0 otherwise.

ata_dev_reread_id

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_reread_id` — Re-read IDENTIFY data

Synopsis

```
int ata_dev_reread_id (struct ata_device * dev, unsigned int
readid_flags);
```

Arguments

dev

target ATA device

readid_flags

read ID flags

Description

Re-read IDENTIFY page and make sure *dev* is still attached to the port.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, negative errno otherwise

ata_dev_revalidate

LINUX

Kernel Hackers ManualOctober 2026

Name

ata_dev_revalidate — Revalidate ATA device

Synopsis

```
int ata_dev_revalidate (struct ata_device * dev, unsigned int
new_class, unsigned int readid_flags);
```

Arguments

dev

device to revalidate

new_class

new class code

readid_flags

read ID flags

Description

Re-read IDENTIFY page, make sure *dev* is still attached to the port and reconfigure it according to the new IDENTIFY page.

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, negative errno otherwise

glob_match

LINUX

Kernel Hackers ManualOctober 2026

Name

`glob_match` — match a text string against a glob-style pattern

Synopsis

```
int glob_match (const char * text, const char * pattern);
```

Arguments

text

the string to be examined

pattern

the glob-style pattern to be matched against

Description

Either/both of text and pattern can be empty strings.

Match text against a glob-style pattern, with wildcards and simple sets:

? matches any single character. * matches any run of characters. [xyz] matches a single character from the set: x, y, or z. [a-d] matches a single character from the range: a, b, c, or d. [a-d0-9] matches a single character from either range.

The special characters ?, [, -, or *, can be matched using a set, eg. [*] Behaviour with malformed patterns is undefined, though generally reasonable.

Sample patterns

"SD1?“, ”SD1[0-5]“, ”*R0“, ”SD*1?[012]*xx"

This function uses one level of recursion per ’*’ in pattern. Since it calls _nothing_ else, and has _no_ explicit local variables, this will not cause stack problems for any reasonable use here.

RETURNS

0 on match, 1 otherwise.

ata_is_40wire

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_is_40wire` — check drive side detection

Synopsis

```
int ata_is_40wire (struct ata_device * dev);
```

Arguments

dev

device

Description

Perform drive side detection decoding, allowing for device vendors who can't follow the documentation.

cable_is_40wire

LINUX

Name

`cable_is_40wire` — 40/80/SATA decider

Synopsis

```
int cable_is_40wire (struct ata_port * ap);
```

Arguments

ap

port to consider

Description

This function encapsulates the policy for speed management in one place. At the moment we don't cache the result but there is a good case for setting ap->cbl to the result when we are called with unknown cables (and figuring out if it impacts hotplug at all).

Return 1 if the cable appears to be 40 wire.

ata_dev_xfermask

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_xfermask` — Compute supported xfermask of the given device

Synopsis

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

Arguments

dev

Device to compute xfermask for

Description

Compute supported xfermask of *dev* and store it in dev->*_mask. This function is responsible for applying all known limits including host controller limits, device blacklist, etc...

LOCKING

None.

ata_dev_set_xfermode

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_set_xfermode` — Issue SET FEATURES - XFER MODE command

Synopsis

```
unsigned int ata_dev_set_xfermode (struct ata_device * dev);
```

Arguments

dev

Device to which command will be sent

Description

Issue SET FEATURES - XFER MODE command to device *dev* on port *ap*.

LOCKING

PCI/etc. bus probe sem.

RETURNS

0 on success, AC_ERR_* mask otherwise.

ata_dev_init_params

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_init_params` — Issue INIT DEV PARAMS command

Synopsis

```
unsigned int ata_dev_init_params (struct ata_device * dev, u16
heads, u16 sectors);
```

Arguments

dev

Device to which command will be sent

heads

Number of heads (taskfile parameter)

sectors

Number of sectors (taskfile parameter)

LOCKING

Kernel thread context (may sleep)

RETURNS

0 on success, AC_ERR_* mask otherwise.

ata_sg_clean

LINUX

Name

`ata_sg_clean` — Unmap DMA memory associated with command

Synopsis

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

Arguments

qc

Command containing DMA memory to be released

Description

Unmap all mapped DMA memory associated with this command.

LOCKING

spin_lock_irqsave(host lock)

atapi_check_dma

LINUX

Name

`atapi_check_dma` — Check whether ATAPI DMA can be supported

Synopsis

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

Arguments

qc

Metadata associated with taskfile to check

Description

Allow low-level driver to filter ATA PACKET commands, returning a status indicating whether or not it is OK to use DMA for the supplied PACKET command.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

0 when ATAPI DMA can be used nonzero otherwise

ata_sg_setup

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sg_setup` — DMA-map the scatter-gather table associated with a command.

Synopsis

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

Arguments

qc

Command with scatter-gather table to be mapped.

Description

DMA-map the scatter-gather table associated with queued_cmd *qc*.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Zero on success, negative on error.

swap_buf_le16

LINUX

Kernel Hackers ManualOctober 2026

Name

`swap_buf_le16` — swap halves of 16-bit words in place

Synopsis

```
void swap_buf_le16 (u16 * buf, unsigned int buf_words);
```

Arguments

buf

Buffer to swap

buf_words

Number of 16-bit words in buffer.

Description

Swap halves of 16-bit words if needed to convert from little-endian byte order to native cpu byte order, or vice-versa.

LOCKING

Inherited from caller.

ata_qc_new

LINUX

Kernel Hackers ManualOctober 2026

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

target port

Description

Some ATA host controllers may implement a queue depth which is less than ATA_MAX_QUEUE. So we shouldn't allocate a tag which is beyond the hardware limitation.

LOCKING

None.

ata_qc_new_init

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
struct ata_queued_cmd * ata_qc_new_init (struct ata_device *
dev);
```

Arguments

dev

Device from whom we request an available command structure

LOCKING

None.

ata_qc_free

LINUX

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

spin_lock_irqsave(host lock)

ata_qc_issue

LINUX

Name

`ata_qc_issue` — issue taskfile to device

Synopsis

```
void 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 lock)

ata_phys_link_online

LINUX

Name

`ata_phys_link_online` — test whether the given link is online

Synopsis

```
bool ata_phys_link_online (struct ata_link * link);
```

Arguments

`link`

ATA link to test

Description

Test whether *`link`* is online. Note that this function returns 0 if online status of *`link`* cannot be obtained, so ata_link_online(link) != !ata_link_offline(link).

LOCKING

None.

RETURNS

True if the port online status is available and online.

ata_phys_link_offline

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_phys_link_offline` — test whether the given link is offline

Synopsis

```
bool ata_phys_link_offline (struct ata_link * link);
```

Arguments

link

ATA link to test

Description

Test whether *link* is offline. Note that this function returns 0 if offline status of *link* cannot be obtained, so ata_link_online(link) != !ata_link_offline(link).

LOCKING

None.

RETURNS

True if the port offline status is available and offline.

ata_dev_init

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dev_init` — Initialize an ata_device structure

Synopsis

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

Arguments

dev

Device structure to initialize

Description

Initialize *dev* in preparation for probing.

LOCKING

Inherited from caller.

ata_link_init

LINUX

Name

`ata_link_init` — Initialize an ata_link structure

Synopsis

```
void ata_link_init (struct ata_port * ap, struct ata_link *
link, int pmp);
```

Arguments

ap

ATA port link is attached to

link

Link structure to initialize

pmp

Port multiplier port number

Description

Initialize *link*.

LOCKING

Kernel thread context (may sleep)

sata_link_init_spd

LINUX

Kernel Hackers ManualOctober 2026

Name

`sata_link_init_spd` — Initialize link->sata_spd_limit

Synopsis

```
int sata_link_init_spd (struct ata_link * link);
```

Arguments

`link`

Link to configure sata_spd_limit for

Description

Initialize *`link`*->[hw_]sata_spd_limit to the currently configured value.

LOCKING

Kernel thread context (may sleep).

RETURNS

0 on success, -errno on failure.

ata_port_alloc

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_port_alloc` — allocate and initialize basic ATA port resources

Synopsis

```
struct ata_port * ata_port_alloc (struct ata_host * host);
```

Arguments

host

ATA host this allocated port belongs to

Description

Allocate and initialize basic ATA port resources.

RETURNS

Allocate ATA port on success, NULL on failure.

LOCKING

Inherited from calling layer (may sleep).

ata_finalize_port_ops

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_finalize_port_ops` — finalize ata_port_operations

Synopsis

```
void ata_finalize_port_ops (struct ata_port_operations * ops);
```

Arguments

ops

ata_port_operations to finalize

Description

An ata_port_operations can inherit from another ops and that ops can again inherit from another. This can go on as many times as necessary as long as there is no loop in the inheritance chain.

Ops tables are finalized when the host is started. NULL or unspecified entries are inherited from the closet ancestor which has the method and the entry is populated with it. After finalization, the ops table directly points to all the methods and ->inherits is no longer necessary and cleared.

Using ATA_OP_NULL, inheriting ops can force a method to NULL.

LOCKING

None.

ata_port_detach

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_port_detach` — Detach ATA port in prepration of device removal

Synopsis

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

Arguments

ap

ATA port to be detached

Description

Detach all ATA devices and the associated SCSI devices of *ap*; then, remove the associated SCSI host. *ap* is guaranteed to be quiescent on return from this function.

LOCKING

Kernel thread context (may sleep).

Chapter 6. libata SCSI translation/emulation

ata_std_bios_param

LINUX

Kernel Hackers ManualOctober 2026

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

LINUX

Kernel Hackers ManualOctober 2026

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_slave_destroy

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_slave_destroy` — SCSI device is about to be destroyed

Synopsis

```
void ata_scsi_slave_destroy (struct scsi_device * sdev);
```

Arguments

sdev

SCSI device to be destroyed

Description

sdev is about to be destroyed for hot/warm unplugging. If this unplugging was initiated by libata as indicated by NULL dev->sdev, this function doesn't have to do

anything. Otherwise, SCSI layer initiated warm-unplug is in progress. Clear dev->sdev, schedule the device for ATA detach and invoke EH.

LOCKING

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

__ata_change_queue_depth

LINUX

Kernel Hackers ManualOctober 2026

Name

`__ata_change_queue_depth` — helper for ata_scsi_change_queue_depth

Synopsis

```
int __ata_change_queue_depth (struct ata_port * ap, struct
scsi_device * sdev, int queue_depth, int reason);
```

Arguments

ap

-- undescribed --

sdev

-- undescribed --

queue_depth

-- undescribed --

reason

-- undescribed --

Description

libsas and libata have different approaches for associating a sdev to its ata_port.

ata_scsi_change_queue_depth

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_change_queue_depth` — SCSI callback for queue depth config

Synopsis

```
int ata_scsi_change_queue_depth (struct scsi_device * sdev,
int queue_depth, int reason);
```

Arguments

sdev

SCSI device to configure queue depth for

queue_depth

new queue depth

`reason`

calling context

Description

This is libata standard hostt->change_queue_depth callback. SCSI will call into this callback when user tries to set queue depth via sysfs.

LOCKING

SCSI layer (we don't care)

RETURNS

Newly configured queue depth.

ata_scsi_queuecmd

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
int ata_scsi_queuecmd (struct Scsi_Host * shost, struct
scsi_cmnd * cmd);
```

Arguments

shost

SCSI host of command to be sent

cmd

SCSI command to be sent

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

ATA host lock

RETURNS

Return value from `__ata_scsi_queuecmd` if *cmd* can be queued, 0 otherwise.

ata_scsi_simulate

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_simulate` — simulate SCSI command on ATA device

Synopsis

```
void ata_scsi_simulate (struct ata_device * dev, struct
scsi_cmnd * cmd);
```

Arguments

dev

the target device

cmd

SCSI command being sent to device.

Description

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

LOCKING

spin_lock_irqsave(host lock)

ata_sas_port_alloc

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sas_port_alloc` — Allocate port for a SAS attached SATA device

Synopsis

```
struct ata_port * ata_sas_port_alloc (struct ata_host * host,
struct ata_port_info * port_info, struct Scsi_Host * shost);
```

Arguments

host

ATA host container for all SAS ports

port_info

Information from low-level host driver

shost

SCSI host that the scsi device is attached to

LOCKING

PCI/etc. bus probe sem.

RETURNS

ata_port pointer on success / NULL on failure.

ata_sas_port_start

LINUX

Name

`ata_sas_port_start` — Set port up for dma.

Synopsis

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

Arguments

ap
Port to initialize

Description

Called just after data structures for each port are initialized.

May be used as the `port_start` entry in ata_port_operations.

LOCKING

Inherited from caller.

ata_sas_port_stop

LINUX

Name

`ata_sas_port_stop` — Undo `ata_sas_port_start`

Synopsis

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

Arguments

ap

Port to shut down

Description

May be used as the `port_stop` entry in ata_port_operations.

LOCKING

Inherited from caller.

ata_sas_async_probe

LINUX

Name

`ata_sas_async_probe` — simply schedule probing and return

Synopsis

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

Arguments

ap

Port to probe

Description

For batch scheduling of probe for sas attached ata devices, assumes the port has already been through `ata_sas_port_init`

ata_sas_port_init

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sas_port_init` — Initialize a SATA device

Synopsis

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

Arguments

ap

SATA port to initialize

LOCKING

PCI/etc. bus probe sem.

RETURNS

Zero on success, non-zero on error.

ata_sas_port_destroy

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sas_port_destroy` — Destroy a SATA port allocated by ata_sas_port_alloc

Synopsis

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

Arguments

ap

SATA port to destroy

ata_sas_slave_configure

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_sas_slave_configure` — Default slave_config routine for libata devices

Synopsis

```
int ata_sas_slave_configure (struct scsi_device * sdev, struct
ata_port * ap);
```

Arguments

sdev

SCSI device to configure

ap

ATA port to which SCSI device is attached

RETURNS

Zero.

ata_sas_queuecmd

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
int ata_sas_queuecmd (struct scsi_cmnd * cmd, struct ata_port
* ap);
```

Arguments

cmd

SCSI command to be sent

ap

ATA port to which the command is being sent

RETURNS

Return value from `__ata_scsi_queuecmd` if *cmd* can be queued, 0 otherwise.

ata_get_identity

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_get_identity` — Handler for HDIO_GET_IDENTITY ioctl

Synopsis

```
int ata_get_identity (struct ata_port * ap, struct scsi_device
* sdev, void __user * arg);
```

Arguments

ap

target port

sdev

SCSI device to get identify data for

arg

User buffer area for identify data

LOCKING

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

RETURNS

Zero on success, negative errno on error.

ata_cmd_ioctl

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_cmd_ioctl` — Handler for HDIO_DRIVE_CMD ioctl

Synopsis

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

Arguments

scsidev

Device to which we are issuing command

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

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_task_ioctl` — Handler for HDIO_DRIVE_TASK ioctl

Synopsis

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

Arguments

scsidev

Device to which we are issuing command

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

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_qc_new` — acquire new ata_queued_cmd reference

Synopsis

```
struct ata_queued_cmd * ata_scsi_qc_new (struct ata_device *
dev, struct scsi_cmnd * cmd);
```

Arguments

dev

ATA device to which the new command is attached

cmd

SCSI command that originated this ATA command

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 lock)

RETURNS

Command allocated, or `NULL` if none available.

ata_dump_status

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_dump_status` — user friendly display of error info

Synopsis

```
void ata_dump_status (unsigned id, struct ata_taskfile * tf);
```

Arguments

id

id of the port in question

tf

ptr to filled out taskfile

Description

Decode and dump the ATA error/status registers for the user so that they have some idea what really happened at the non make-believe layer.

LOCKING

inherited from caller

ata_to_sense_error

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_to_sense_error` — convert ATA error to SCSI error

Synopsis

```
void ata_to_sense_error (unsigned id, u8 drv_stat, u8 drv_err,
u8 * sk, u8 * asc, u8 * ascq, int verbose);
```

Arguments

id

ATA device number

drv_stat

value contained in ATA status register

drv_err

value contained in ATA error register

sk

the sense key we'll fill out

asc

the additional sense code we'll fill out

ascq

the additional sense code qualifier we'll fill out

verbose

be verbose

Description

Converts an ATA error into a SCSI error. Fill out pointers to SK, ASC, and ASCQ bytes for later use in fixed or descriptor format sense blocks.

LOCKING

spin_lock_irqsave(host lock)

ata_gen_ata_sense

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_gen_ata_sense` — generate a SCSI fixed sense block

Synopsis

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

Arguments

qc

Command that we are erroring out

Description

Generate sense block for a failed ATA command *qc*. Descriptor format is used to accommodate LBA48 block address.

LOCKING

None.

atapi_drain_needed

LINUX

Kernel Hackers ManualOctober 2026

Name

`atapi_drain_needed` — Check whether data transfer may overflow

Synopsis

```
int atapi_drain_needed (struct request * rq);
```

Arguments

rq

request to be checked

Description

ATAPI commands which transfer variable length data to host might overflow due to application error or hardare bug. This function checks whether overflow should be drained and ignored for *request*.

LOCKING

None.

RETURNS

1 if ; otherwise, 0.

ata_scsi_start_stop_xlat

LINUX

Kernel Hackers ManualOctober 2026

Name

ata_scsi_start_stop_xlat — Translate SCSI START STOP UNIT command

Synopsis

```
unsigned int ata_scsi_start_stop_xlat (struct ata_queued_cmd *
qc);
```

Arguments

qc

Storage for translated ATA taskfile

Description

Sets up an ATA taskfile to issue STANDBY (to stop) or READ VERIFY (to start). Perhaps these commands should be preceded by CHECK POWER MODE to see what power mode the device is already in. [See SAT revision 5 at www.t10.org]

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Zero on success, non-zero on error.

ata_scsi_flush_xlat

LINUX

Name

`ata_scsi_flush_xlat` — Translate SCSI SYNCHRONIZE CACHE command

Synopsis

```
unsigned int ata_scsi_flush_xlat (struct ata_queued_cmd * qc);
```

Arguments

qc

Storage for translated ATA taskfile

Description

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

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Zero on success, non-zero on error.

scsi_6_lba_len

LINUX

Kernel Hackers ManualOctober 2026

Name

`scsi_6_lba_len` — Get LBA and transfer length

Synopsis

```
void scsi_6_lba_len (const u8 * cdb, u64 * plba, u32 * plen);
```

Arguments

cdb

SCSI command to translate

plba

the LBA

plen

the transfer length

Description

Calculate LBA and transfer length for 6-byte commands.

scsi_10_lba_len

LINUX

Kernel Hackers ManualOctober 2026

Name

`scsi_10_lba_len` — Get LBA and transfer length

Synopsis

```
void scsi_10_lba_len (const u8 * cdb, u64 * plba, u32 * plen);
```

Arguments

cdb

SCSI command to translate

plba

the LBA

plen

the transfer length

Description

Calculate LBA and transfer length for 10-byte commands.

scsi_16_lba_len

LINUX

Kernel Hackers ManualOctober 2026

Name

`scsi_16_lba_len` — Get LBA and transfer length

Synopsis

```
void scsi_16_lba_len (const u8 * cdb, u64 * plba, u32 * plen);
```

Arguments

cdb

SCSI command to translate

plba

the LBA

plen

the transfer length

Description

Calculate LBA and transfer length for 16-byte commands.

ata_scsi_verify_xlat

LINUX

Kernel Hackers ManualOctober 2026

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);
```

Arguments

qc

Storage for translated ATA taskfile

Description

Converts SCSI VERIFY command to an ATA READ VERIFY command.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Zero on success, non-zero on error.

ata_scsi_rw_xlat

LINUX

Kernel Hackers ManualOctober 2026

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);
```

Arguments

qc

Storage for translated ATA taskfile

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 lock)

RETURNS

Zero on success, non-zero on error.

ata_scsi_translate

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
int ata_scsi_translate (struct ata_device * dev, struct
scsi_cmnd * cmd, ata_xlat_func_t xlat_func);
```

Arguments

dev

ATA device to which the command is addressed

cmd

SCSI command to execute

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.

The xlat_func argument (actor) returns 0 if ready to execute ATA command, else 1 to finish translation. If 1 is returned then cmd->result (and possibly cmd->sense_buffer) are assumed to be set reflecting an error condition or clean (early) termination.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

0 on success, SCSI_ML_QUEUE_DEVICE_BUSY if the command needs to be deferred.

ata_scsi_rbuf_get

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_rbuf_get` — Map response buffer.

Synopsis

```
void * ata_scsi_rbuf_get (struct scsi_cmnd * cmd, bool
copy_in, unsigned long * flags);
```

Arguments

cmd

SCSI command containing buffer to be mapped.

copy_in

copy in from user buffer

flags

unsigned long variable to store irq enable status

Description

Prepare buffer for simulated SCSI commands.

LOCKING

spin_lock_irqsave(ata_scsi_rbuf_lock) on success

RETURNS

Pointer to response buffer.

ata_scsi_rbuf_put

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_rbuf_put` — Unmap response buffer.

Synopsis

```
void ata_scsi_rbuf_put (struct scsi_cmnd * cmd, bool copy_out,
unsigned long * flags);
```

Arguments

cmd

SCSI command containing buffer to be unmapped.

copy_out

copy out result

flags

flags passed to `ata_scsi_rbuf_get`

Description

Returns rbuf buffer. The result is copied to *cmd*'s buffer if *copy_back* is true.

LOCKING

Unlocks ata_scsi_rbuf_lock.

ata_scsi_rbuf_fill

LINUX

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));
```

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 (0), or not (in which case cmd->result and sense buffer are assumed to be set).

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_inq_std

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsiop_inq_std` — Simulate INQUIRY command

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

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

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_inq_00

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Returns list of inquiry VPD pages available.

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_inq_80

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Returns ATA device serial number.

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_inq_83

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Yields two logical unit device identification designators

- vendor specific ASCII containing the ATA serial number - SAT defined “t10 vendor id based” containing ASCII vendor name (“ATA ”), model and serial numbers.

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_inq_89

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsiop_inq_89` — Simulate INQUIRY VPD page 89, ATA info

Synopsis

```
unsigned int ata_scsiop_inq_89 (struct ata_scsi_args * args,
u8 * rbuf);
```

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Yields SAT-specified ATA VPD page.

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_noop

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsiop_noop` — Command handler that simply returns success.

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

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

LOCKING

spin_lock_irqsave(host lock)

ata_msense_caching

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_msense_caching` — Simulate MODE SENSE caching info page

Synopsis

```
unsigned int ata_msense_caching (u16 * id, u8 * buf);
```

Arguments

id

device IDENTIFY data

buf

output 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

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_msense_ctl_mode` — Simulate MODE SENSE control mode page

Synopsis

```
unsigned int ata_msense_ctl_mode (u8 * buf);
```

Arguments

buf

output buffer

Description

Generate a generic MODE SENSE control mode page.

LOCKING

None.

ata_msense_rw_recovery

LINUX

Name

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

Synopsis

```
unsigned int ata_msense_rw_recovery (u8 * buf);
```

Arguments

buf

output buffer

Description

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

LOCKING

None.

ata_scsiop_mode_sense

LINUX

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);
```

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Simulate MODE SENSE commands. Assume this is invoked for direct access devices (e.g. disks) only. There should be no block descriptor for other device types.

LOCKING

spin_lock_irqsave(host lock)

ata_scsiop_read_cap

LINUX

Name

`ata_scsiop_read_cap` — Simulate READ CAPACITY[16] commands

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Simulate READ CAPACITY commands.

LOCKING

None.

ata_scsiop_report_luns

LINUX

Name

`ata_scsiop_report_luns` — Simulate REPORT LUNS command

Synopsis

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

Arguments

`args`

device IDENTIFY data / SCSI command of interest.

`rbuf`

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

Description

Simulate REPORT LUNS command.

LOCKING

spin_lock_irqsave(host lock)

atapi_xlat

LINUX

Name

`atapi_xlat` — Initialize PACKET taskfile

Synopsis

```
unsigned int atapi_xlat (struct ata_queued_cmd * qc);
```

Arguments

qc

command structure to be initialized

LOCKING

spin_lock_irqsave(host lock)

RETURNS

Zero on success, non-zero on failure.

ata_scsi_find_dev

LINUX

Name

`ata_scsi_find_dev` — lookup ata_device from scsi_cmnd

Synopsis

```
struct ata_device * ata_scsi_find_dev (struct ata_port * ap,
const 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 lock)

RETURNS

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

ata_scsi_pass_thru

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

```
unsigned int ata_scsi_pass_thru (struct ata_queued_cmd * qc);
```

Arguments

qc

command structure to be initialized

Description

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

RETURNS

Zero on success, non-zero on failure.

ata_get_xlat_func

LINUX

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

LINUX

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`.

ata_scsi_offline_dev

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_offline_dev` — offline attached SCSI device

Synopsis

```
int ata_scsi_offline_dev (struct ata_device * dev);
```

Arguments

dev

ATA device to offline attached SCSI device for

Description

This function is called from `ata_eh_hotplug` and responsible for taking the SCSI device attached to *dev* offline. This function is called with host lock which protects dev->sdev against clearing.

LOCKING

spin_lock_irqsave(host lock)

RETURNS

1 if attached SCSI device exists, 0 otherwise.

ata_scsi_remove_dev

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_remove_dev` — remove attached SCSI device

Synopsis

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

Arguments

dev

ATA device to remove attached SCSI device for

Description

This function is called from `ata_eh_scsi_hotplug` and responsible for removing the SCSI device attached to *dev*.

LOCKING

Kernel thread context (may sleep).

ata_scsi_media_change_notify

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_media_change_notify` — send media change event

Synopsis

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

Arguments

dev

Pointer to the disk device with media change event

Description

Tell the block layer to send a media change notification event.

LOCKING

spin_lock_irqsave(host lock)

ata_scsi_hotplug

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_hotplug` — SCSI part of hotplug

Synopsis

```
void ata_scsi_hotplug (struct work_struct * work);
```

Arguments

work

Pointer to ATA port to perform SCSI hotplug on

Description

Perform SCSI part of hotplug. It's executed from a separate workqueue after EH completes. This is necessary because SCSI hot plugging requires working EH and hot unplugging is synchronized with hot plugging with a mutex.

LOCKING

Kernel thread context (may sleep).

ata_scsi_user_scan

LINUX

Kernel Hackers ManualOctober 2026

Name

`ata_scsi_user_scan` — indication for user-initiated bus scan

Synopsis

```
int ata_scsi_user_scan (struct Scsi_Host * shost, unsigned int
channel, unsigned int id, unsigned int lun);
```

Arguments

`shost`

SCSI host to scan

`channel`

Channel to scan

`id`

ID to scan

`lun`

LUN to scan

Description

This function is called when user explicitly requests bus scan. Set probe pending flag and invoke EH.

LOCKING

SCSI layer (we don't care)

RETURNS

Zero.

ata_scsi_dev_rescan

LINUX

Name

`ata_scsi_dev_rescan` — initiate `scsi_rescan_device`

Synopsis

```
void ata_scsi_dev_rescan (struct work_struct * work);
```

Arguments

work

Pointer to ATA port to perform `scsi_rescan_device`

Description

After ATA pass thru (SAT) commands are executed successfully, libata need to propagate the changes to SCSI layer.

LOCKING

Kernel thread context (may sleep).

Chapter 7. ATA errors and exceptions

This chapter tries to identify what error/exception conditions exist for ATA/ATAPI devices and describe how they should be handled in implementation-neutral way.

The term 'error' is used to describe conditions where either an explicit error condition is reported from device or a command has timed out.

The term 'exception' is either used to describe exceptional conditions which are not errors (say, power or hotplug events), or to describe both errors and non-error exceptional conditions. Where explicit distinction between error and exception is necessary, the term 'non-error exception' is used.

7.1. Exception categories

Exceptions are described primarily with respect to legacy taskfile + bus master IDE interface. If a controller provides other better mechanism for error reporting, mapping those into categories described below shouldn't be difficult.

In the following sections, two recovery actions - reset and reconfiguring transport - are mentioned. These are described further in Section 7.2.

7.1.1. HSM violation

This error is indicated when STATUS value doesn't match HSM requirement during issuing or excution any ATA/ATAPI command.

Examples

- ATA_STATUS doesn't contain !BSY && DRDY && !DRQ while trying to issue a command.
- !BSY && !DRQ during PIO data transfer.
- DRQ on command completion.
- !BSY && ERR after CDB tranfer starts but before the last byte of CDB is transferred. ATA/ATAPI standard states that "The device shall not terminate the PACKET command with an error before the last byte of the command packet has been written" in the error outputs description of PACKET command and the state diagram doesn't include such transitions.

In these cases, HSM is violated and not much information regarding the error can be acquired from STATUS or ERROR register. IOW, this error can be anything - driver bug, faulty device, controller and/or cable.

As HSM is violated, reset is necessary to restore known state. Reconfiguring transport for lower speed might be helpful too as transmission errors sometimes cause this kind of errors.

7.1.2. ATA/ATAPI device error (non-NCQ / non-CHECK CONDITION)

These are errors detected and reported by ATA/ATAPI devices indicating device problems. For this type of errors, STATUS and ERROR register values are valid and describe error condition. Note that some of ATA bus errors are detected by ATA/ATAPI devices and reported using the same mechanism as device errors. Those cases are described later in this section.

For ATA commands, this type of errors are indicated by !BSY && ERR during command execution and on completion.

For ATAPI commands,

- !BSY && ERR && ABRT right after issuing PACKET indicates that PACKET command is not supported and falls in this category.
- !BSY && ERR(==CHK) && !ABRT after the last byte of CDB is transferred indicates CHECK CONDITION and doesn't fall in this category.
- !BSY && ERR(==CHK) && ABRT after the last byte of CDB is transferred *probably* indicates CHECK CONDITION and doesn't fall in this category.

Of errors detected as above, the followings are not ATA/ATAPI device errors but ATA bus errors and should be handled according to Section 7.1.5.

CRC error during data transfer

This is indicated by ICRC bit in the ERROR register and means that corruption occurred during data transfer. Up to ATA/ATAPI-7, the standard specifies that this bit is only applicable to UDMA transfers but ATA/ATAPI-8 draft revision 1f says that the bit may be applicable to multiword DMA and PIO.

ABRT error during data transfer or on completion

Up to ATA/ATAPI-7, the standard specifies that ABRT could be set on ICRC errors and on cases where a device is not able to complete a command. Combined with the fact that MWDMA and PIO transfer errors aren't allowed to use ICRC bit up to ATA/ATAPI-7, it seems to imply that ABRT bit alone could indicate tranfer errors.

However, ATA/ATAPI-8 draft revision 1f removes the part that ICRC errors can turn on ABRT. So, this is kind of gray area. Some heuristics are needed here.

ATA/ATAPI device errors can be further categorized as follows.

Media errors

This is indicated by UNC bit in the ERROR register. ATA devices reports UNC error only after certain number of retries cannot recover the data, so there's nothing much else to do other than notifying upper layer.

READ and WRITE commands report CHS or LBA of the first failed sector but ATA/ATAPI standard specifies that the amount of transferred data on error completion is indeterminate, so we cannot assume that sectors preceding the failed sector have been transferred and thus cannot complete those sectors successfully as SCSI does.

Media changed / media change requested error

<<TODO: fill here>>

Address error

This is indicated by IDNF bit in the ERROR register. Report to upper layer.

Other errors

This can be invalid command or parameter indicated by ABRT ERROR bit or some other error condition. Note that ABRT bit can indicate a lot of things including ICRC and Address errors. Heuristics needed.

Depending on commands, not all STATUS/ERROR bits are applicable. These non-applicable bits are marked with "na" in the output descriptions but up to ATA/ATAPI-7 no definition of "na" can be found. However, ATA/ATAPI-8 draft revision 1f describes "N/A" as follows.

3.2.3.3a N/A

A keyword the indicates a field has no defined value in this standard and should not be checked by the host or device. N/A fields should be cleared to zero.

So, it seems reasonable to assume that "na" bits are cleared to zero by devices and thus need no explicit masking.

7.1.3. ATAPI device CHECK CONDITION

ATAPI device CHECK CONDITION error is indicated by set CHK bit (ERR bit) in the STATUS register after the last byte of CDB is transferred for a PACKET command. For this kind of errors, sense data should be acquired to gather information regarding the errors. REQUEST SENSE packet command should be used to acquire sense data.

Once sense data is acquired, this type of errors can be handled similary to other SCSI errors. Note that sense data may indicate ATA bus error (e.g. Sense Key 04h HARDWARE ERROR && ASC/ASCQ 47h/00h SCSI PARITY ERROR). In such cases, the error should be considered as an ATA bus error and handled according to Section 7.1.5.

7.1.4. ATA device error (NCQ)

NCQ command error is indicated by cleared BSY and set ERR bit during NCQ command phase (one or more NCQ commands outstanding). Although STATUS and ERROR registers will contain valid values describing the error, READ LOG EXT is required to clear the error condition, determine which command has failed and acquire more information.

READ LOG EXT Log Page 10h reports which tag has failed and taskfile register values describing the error. With this information the failed command can be handled as a normal ATA command error as in Section 7.1.2 and all other in-flight commands must be retried. Note that this retry should not be counted - it's likely that commands retried this way would have completed normally if it were not for the failed command.

Note that ATA bus errors can be reported as ATA device NCQ errors. This should be handled as described in Section 7.1.5.

If READ LOG EXT Log Page 10h fails or reports NQ, we're thoroughly screwed. This condition should be treated according to Section 7.1.1.

7.1.5. ATA bus error

ATA bus error means that data corruption occurred during transmission over ATA bus (SATA or PATA). This type of errors can be indicated by

- ICRC or ABRT error as described in Section 7.1.2.
- Controller-specific error completion with error information indicating transmission error.

- On some controllers, command timeout. In this case, there may be a mechanism to determine that the timeout is due to transmission error.
- Unknown/random errors, timeouts and all sorts of weirdities.

As described above, transmission errors can cause wide variety of symptoms ranging from device ICRC error to random device lockup, and, for many cases, there is no way to tell if an error condition is due to transmission error or not; therefore, it's necessary to employ some kind of heuristic when dealing with errors and timeouts. For example, encountering repetitive ABRT errors for known supported command is likely to indicate ATA bus error.

Once it's determined that ATA bus errors have possibly occurred, lowering ATA bus transmission speed is one of actions which may alleviate the problem. See Section 7.2.3 for more information.

7.1.6. PCI bus error

Data corruption or other failures during transmission over PCI (or other system bus). For standard BMDMA, this is indicated by Error bit in the BMDMA Status register. This type of errors must be logged as it indicates something is very wrong with the system. Resetting host controller is recommended.

7.1.7. Late completion

This occurs when timeout occurs and the timeout handler finds out that the timed out command has completed successfully or with error. This is usually caused by lost interrupts. This type of errors must be logged. Resetting host controller is recommended.

7.1.8. Unknown error (timeout)

This is when timeout occurs and the command is still processing or the host and device are in unknown state. When this occurs, HSM could be in any valid or invalid state. To bring the device to known state and make it forget about the timed out command, resetting is necessary. The timed out command may be retried.

Timeouts can also be caused by transmission errors. Refer to Section 7.1.5 for more details.

7.1.9. Hotplug and power management exceptions

<<TODO: fill here>>

7.2. EH recovery actions

This section discusses several important recovery actions.

7.2.1. Clearing error condition

Many controllers require its error registers to be cleared by error handler. Different controllers may have different requirements.

For SATA, it's strongly recommended to clear at least SError register during error handling.

7.2.2. Reset

During EH, resetting is necessary in the following cases.

- HSM is in unknown or invalid state
- HBA is in unknown or invalid state
- EH needs to make HBA/device forget about in-flight commands
- HBA/device behaves weirdly

Resetting during EH might be a good idea regardless of error condition to improve EH robustness. Whether to reset both or either one of HBA and device depends on situation but the following scheme is recommended.

- When it's known that HBA is in ready state but ATA/ATAPI device is in unknown state, reset only device.
- If HBA is in unknown state, reset both HBA and device.

HBA resetting is implementation specific. For a controller complying to taskfile/BMDMA PCI IDE, stopping active DMA transaction may be sufficient iff BMDMA state is the only HBA context. But even mostly taskfile/BMDMA PCI IDE complying controllers may have implementation specific requirements and mechanism to reset themselves. This must be addressed by specific drivers.

OTOH, ATA/ATAPI standard describes in detail ways to reset ATA/ATAPI devices.

PATA hardware reset

This is hardware initiated device reset signalled with asserted PATA RESET- signal. There is no standard way to initiate hardware reset from software although some hardware provides registers that allow driver to directly tweak the RESET- signal.

Software reset

This is achieved by turning CONTROL SRST bit on for at least 5us. Both PATA and SATA support it but, in case of SATA, this may require controller-specific support as the second Register FIS to clear SRST should be transmitted while BSY bit is still set. Note that on PATA, this resets both master and slave devices on a channel.

EXECUTE DEVICE DIAGNOSTIC command

Although ATA/ATAPI standard doesn't describe exactly, EDD implies some level of resetting, possibly similar level with software reset. Host-side EDD protocol can be handled with normal command processing and most SATA controllers should be able to handle EDD's just like other commands. As in software reset, EDD affects both devices on a PATA bus.

Although EDD does reset devices, this doesn't suit error handling as EDD cannot be issued while BSY is set and it's unclear how it will act when device is in unknown/weird state.

ATAPI DEVICE RESET command

This is very similar to software reset except that reset can be restricted to the selected device without affecting the other device sharing the cable.

SATA phy reset

This is the preferred way of resetting a SATA device. In effect, it's identical to PATA hardware reset. Note that this can be done with the standard SCR Control register. As such, it's usually easier to implement than software reset.

One more thing to consider when resetting devices is that resetting clears certain configuration parameters and they need to be set to their previous or newly adjusted values after reset.

Parameters affected are.

- CHS set up with INITIALIZE DEVICE PARAMETERS (seldom used)
- Parameters set with SET FEATURES including transfer mode setting

- Block count set with SET MULTIPLE MODE
- Other parameters (SET MAX, MEDIA LOCK...)

ATA/ATAPI standard specifies that some parameters must be maintained across hardware or software reset, but doesn't strictly specify all of them. Always reconfiguring needed parameters after reset is required for robustness. Note that this also applies when resuming from deep sleep (power-off).

Also, ATA/ATAPI standard requires that IDENTIFY DEVICE / IDENTIFY PACKET DEVICE is issued after any configuration parameter is updated or a hardware reset and the result used for further operation. OS driver is required to implement revalidation mechanism to support this.

7.2.3. Reconfigure transport

For both PATA and SATA, a lot of corners are cut for cheap connectors, cables or controllers and it's quite common to see high transmission error rate. This can be mitigated by lowering transmission speed.

The following is a possible scheme Jeff Garzik suggested.

> If more than $N (3?) transmission errors happen in 15 minutes,
>
> - if SATA, decrease SATA PHY speed. if speed cannot be decreased,
> - decrease UDMA xfer speed. if at UDMA0, switch to PIO4,
> - decrease PIO xfer speed. if at PIO3, complain, but continue

Chapter 8. ata_piix Internals

ich_pata_cable_detect

LINUX

Kernel Hackers ManualOctober 2026

Name

`ich_pata_cable_detect` — Probe host controller cable detect info

Synopsis

```
int ich_pata_cable_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_prereset

LINUX

Kernel Hackers ManualOctober 2026

Name

`piix_pata_prereset` — prereset for PATA host controller

Synopsis

```
int piix_pata_prereset (struct ata_link * link, unsigned long
deadline);
```

Arguments

link

Target link

deadline

deadline jiffies for the operation

LOCKING

None (inherited from caller).

piix_set_piomode

LINUX

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).

do_pata_set_dmamode

LINUX

Name

`do_pata_set_dmamode` — Initialize host controller PATA PIO timings

Synopsis

```
void do_pata_set_dmamode (struct ata_port * ap, struct
ata_device * adev, int isich);
```

Arguments

ap

Port whose timings we are configuring

adev

Drive in question

isich

set if the chip is an ICH device

Description

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

LOCKING

None (inherited from caller).

piix_set_dmamode

LINUX

Kernel Hackers ManualOctober 2026

Name

`piix_set_dmamode` — Initialize host controller PATA DMA 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 MW/UDMA mode for device, in host controller PCI config space.

LOCKING

None (inherited from caller).

ich_set_dmamode

LINUX

Kernel Hackers ManualOctober 2026

Name

`ich_set_dmamode` — Initialize host controller PATA DMA timings

Synopsis

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

Arguments

ap

Port whose timings we are configuring

adev

um

Description

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

LOCKING

None (inherited from caller).

piix_check_450nx_errata

LINUX

Kernel Hackers ManualOctober 2026

Name

`piix_check_450nx_errata` — Check for problem 450NX setup

Synopsis

```
int piix_check_450nx_errata (struct pci_dev * ata_dev);
```

Arguments

ata_dev

the PCI device to check

Description

Check for the present of 450NX errata #19 and errata #25. If they are found return an error code so we can turn off DMA

piix_init_one

LINUX

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.

Chapter 9. sata_sil Internals

sil_set_mode

LINUX

Kernel Hackers ManualOctober 2026

Name

`sil_set_mode` — wrap set_mode functions

Synopsis

```
int sil_set_mode (struct ata_link * link, struct ata_device **
r_failed);
```

Arguments

link

link to set up

r_failed

returned device when we fail

Description

Wrap the libata method for device setup as after the setup we need to inspect the results and do some configuration work

sil_dev_config

LINUX

Kernel Hackers ManualOctober 2026

Name

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

Synopsis

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

Arguments

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

Chapter 10. Thanks

The bulk of the ATA knowledge comes thanks to long conversations with Andre Hedrick (www.linux-ide.org), and long hours pondering the ATA and SCSI specifications.

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).