Voltage and current regulator API

Liam Girdwood <lrg@slimlogic.co.uk>
Mark Brown, Wolfson Microelectronics
<broonie@opensource.wolfsonmicro.com>

Voltage and current regulator API

by Liam Girdwood and Mark Brown

Copyright © 2007-2008 Wolfson Microelectronics
Copyright © 2008 Liam Girdwood

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

Table of Contents

1. Introduction 1
    Glossary 1
2. Consumer driver interface 2
    Enabling and disabling 2
    Configuration 2
    Callbacks 2
3. Regulator driver interface 3
4. Machine interface 4
    Supplies 4
    Constraints 4
5. API reference 5
    struct pre_voltage_change_data 6
    struct regulator_bulk_data 7
    struct regulator_state 8
    struct regulation_constraints 9
    struct regulator_consumer_supply 11
    struct regulator_init_data 12
    struct regulator_linear_range 13
    struct regulator_ops 14
    struct regulator_desc 16
    struct regulator_config 19
    regulator_get 20
    regulator_get_exclusive 21
    regulator_get_optional 22
    regulator_put 23
    regulator_register_supply_alias 24
    regulator_unregister_supply_alias 25
    regulator_bulk_register_supply_alias 26
    regulator_bulk_unregister_supply_alias 27
    regulator_enable 28
    regulator_disable 29
    regulator_force_disable 30
    regulator_disable_deferred 31
    regulator_is_enabled 32
    regulator_can_change_voltage 33
    regulator_count_voltages 34
    regulator_list_voltage 35
    regulator_get_hardware_vsel_register 36
    regulator_list_hardware_vsel 37
    regulator_get_linear_step 38
    regulator_is_supported_voltage 39
    regulator_set_voltage 40
    regulator_set_voltage_time 41
    regulator_set_voltage_time_sel 42
    regulator_sync_voltage 43
    regulator_get_voltage 44
    regulator_set_current_limit 45
    regulator_get_current_limit 46
    regulator_set_mode 47
    regulator_get_mode 48
    regulator_set_load 49

regulator_allow_bypass 50
regulator_register_notifier 51
regulator_unregister_notifier 52
regulator_bulk_get 53
regulator_bulk_enable 54
regulator_bulk_disable 55
regulator_bulk_force_disable 56
regulator_bulk_free 57
regulator_notifier_call_chain 58
regulator_mode_to_status 59
regulator_register 60
regulator_unregister 61
regulator_suspend_prepare 62
regulator_suspend_finish 63
regulator_has_full_constraints 64
rdev_get_drvdata 65
regulator_get_drvdata 66
regulator_set_drvdata 67
rdev_get_id 68

Chapter 1. Introduction

This framework is designed to provide a standard kernel interface to control voltage and current regulators.

The intention is to allow systems to dynamically control regulator power output in order to save power and prolong battery life. This applies to both voltage regulators (where voltage output is controllable) and current sinks (where current limit is controllable).

Note that additional (and currently more complete) documentation is available in the Linux kernel source under `Documentation/power/regulator`.

Glossary

The regulator API uses a number of terms which may not be familiar:

Glossary

Regulator	Electronic device that supplies power to other devices. Most regulators can enable and disable their output and some can also control their output voltage or current.
Consumer	Electronic device which consumes power provided by a regulator. These may either be static, requiring only a fixed supply, or dynamic, requiring active management of the regulator at runtime.
Power Domain	The electronic circuit supplied by a given regulator, including the regulator and all consumer devices. The configuration of the regulator is shared between all the components in the circuit.
Power Management Integrated Circuit	An IC which contains numerous regulators and often also other subsystems. In an embedded system the primary PMIC is often equivalent to a combination of the PSU and southbridge in a desktop system.

Chapter 2. Consumer driver interface

This offers a similar API to the kernel clock framework. Consumer drivers use get and put operations to acquire and release regulators. Functions are provided to enable and disable the regulator and to get and set the runtime parameters of the regulator.

When requesting regulators consumers use symbolic names for their supplies, such as "Vcc", which are mapped into actual regulator devices by the machine interface.

A stub version of this API is provided when the regulator framework is not in use in order to minimise the need to use ifdefs.

Enabling and disabling

The regulator API provides reference counted enabling and disabling of regulators. Consumer devices use the `regulator_enable` and `regulator_disable` functions to enable and disable regulators. Calls to the two functions must be balanced.

Note that since multiple consumers may be using a regulator and machine constraints may not allow the regulator to be disabled there is no guarantee that calling `regulator_disable` will actually cause the supply provided by the regulator to be disabled. Consumer drivers should assume that the regulator may be enabled at all times.

Configuration

Some consumer devices may need to be able to dynamically configure their supplies. For example, MMC drivers may need to select the correct operating voltage for their cards. This may be done while the regulator is enabled or disabled.

The `regulator_set_voltage` and `regulator_set_current_limit` functions provide the primary interface for this. Both take ranges of voltages and currents, supporting drivers that do not require a specific value (eg, CPU frequency scaling normally permits the CPU to use a wider range of supply voltages at lower frequencies but does not require that the supply voltage be lowered). Where an exact value is required both minimum and maximum values should be identical.

Callbacks

Callbacks may also be registered for events such as regulation failures.

Chapter 3. Regulator driver interface

Drivers for regulator chips register the regulators with the regulator core, providing operations structures to the core. A notifier interface allows error conditions to be reported to the core.

Registration should be triggered by explicit setup done by the platform, supplying a struct regulator_init_data for the regulator containing constraint and supply information.

Chapter 4. Machine interface

This interface provides a way to define how regulators are connected to consumers on a given system and what the valid operating parameters are for the system.

Supplies

Regulator supplies are specified using struct regulator_consumer_supply. This is done at driver registration time as part of the machine constraints.

Constraints

As well as defining the connections the machine interface also provides constraints defining the operations that clients are allowed to perform and the parameters that may be set. This is required since generally regulator devices will offer more flexibility than it is safe to use on a given system, for example supporting higher supply voltages than the consumers are rated for.

This is done at driver registration time by providing a struct regulation_constraints.

The constraints may also specify an initial configuration for the regulator in the constraints, which is particularly useful for use with static consumers.

Chapter 5. API reference

Due to limitations of the kernel documentation framework and the existing layout of the source code the entire regulator API is documented here.

Name

struct pre_voltage_change_data — Data sent with PRE_VOLTAGE_CHANGE event

Synopsis

```
struct pre_voltage_change_data {
  unsigned long old_uV;
  unsigned long min_uV;
  unsigned long max_uV;
};
```

Members

old_uV Current voltage before change.

min_uV Min voltage we'll change to.

max_uV Max voltage we'll change to.

Name

struct regulator_bulk_data — Data used for bulk regulator operations.

Synopsis

```
struct regulator_bulk_data {
  const char * supply;
  struct regulator * consumer;
};
```

Members

supply The name of the supply. Initialised by the user before using the bulk regulator APIs.

consumer The regulator consumer for the supply. This will be managed by the bulk API.

Description

The regulator APIs provide a series of `regulator_bulk_` API calls as a convenience to consumers which require multiple supplies. This structure is used to manage data for these calls.

Name

struct regulator_state — regulator state during low power system states

Synopsis

```
struct regulator_state {
  int uV;
  unsigned int mode;
  int enabled;
  int disabled;
};
```

Members

uV	Operating voltage during suspend.
mode	Operating mode during suspend.
enabled	Enabled during suspend.
disabled	Disabled during suspend.

Description

This describes a regulators state during a system wide low power state. One of enabled or disabled must be set for the configuration to be applied.

Name

struct regulation_constraints — regulator operating constraints.

Synopsis

```
struct regulation_constraints {
  const char * name;
  int min_uV;
  int max_uV;
  int uV_offset;
  int min_uA;
  int max_uA;
  int ilim_uA;
  int system_load;
  unsigned int valid_modes_mask;
  unsigned int valid_ops_mask;
  int input_uV;
  struct regulator_state state_disk;
  struct regulator_state state_mem;
  struct regulator_state state_standby;
  suspend_state_t initial_state;
  unsigned int initial_mode;
  unsigned int ramp_delay;
  unsigned int enable_time;
  unsigned always_on:1;
  unsigned boot_on:1;
  unsigned apply_uV:1;
  unsigned ramp_disable:1;
  unsigned soft_start:1;
  unsigned pull_down:1;
};
```

Members

name	Descriptive name for the constraints, used for display purposes.
min_uV	Smallest voltage consumers may set.
max_uV	Largest voltage consumers may set.
uV_offset	Offset applied to voltages from consumer to compensate for voltage drops.
min_uA	Smallest current consumers may set.
max_uA	Largest current consumers may set.
ilim_uA	Maximum input current.
system_load	Load that isn't captured by any consumer requests.
valid_modes_mask	Mask of modes which may be configured by consumers.
valid_ops_mask	Operations which may be performed by consumers.

input_uV	Input voltage for regulator when supplied by another regulator.
state_disk	State for regulator when system is suspended in disk mode.
state_mem	State for regulator when system is suspended in mem mode.
state_standby	State for regulator when system is suspended in standby mode.
initial_state	Suspend state to set by default.
initial_mode	Mode to set at startup.
ramp_delay	Time to settle down after voltage change (unit: uV/us)
enable_time	Turn-on time of the rails (unit: microseconds)
always_on	Set if the regulator should never be disabled.
boot_on	Set if the regulator is enabled when the system is initially started. If the regulator is not enabled by the hardware or bootloader then it will be enabled when the constraints are applied.
apply_uV	Apply the voltage constraint when initialising.
ramp_disable	Disable ramp delay when initialising or when setting voltage.
soft_start	Enable soft start so that voltage ramps slowly.
pull_down	Enable pull down when regulator is disabled.

Description

This struct describes regulator and board/machine specific constraints.

Name

struct regulator_consumer_supply — supply -> device mapping

Synopsis

```
struct regulator_consumer_supply {
  const char * dev_name;
  const char * supply;
};
```

Members

dev_name — Result of `dev_name` for the consumer.

supply — Name for the supply.

Description

This maps a supply name to a device. Use of dev_name allows support for buses which make struct device available late such as I2C.

Name

struct regulator_init_data — regulator platform initialisation data.

Synopsis

```
struct regulator_init_data {
  const char * supply_regulator;
  struct regulation_constraints constraints;
  int num_consumer_supplies;
  struct regulator_consumer_supply * consumer_supplies;
  int (* regulator_init) (void *driver_data);
  void * driver_data;
};
```

Members

supply_regulator	Parent regulator. Specified using the regulator name as it appears in the name field in sysfs, which can be explicitly set using the constraints field 'name'.
constraints	Constraints. These must be specified for the regulator to be usable.
num_consumer_supplies	Number of consumer device supplies.
consumer_supplies	Consumer device supply configuration.
regulator_init	Callback invoked when the regulator has been registered.
driver_data	Data passed to regulator_init.

Description

Initialisation constraints, our supply and consumers supplies.

Name

struct regulator_linear_range — specify linear voltage ranges

Synopsis

```
struct regulator_linear_range {
  unsigned int min_uV;
  unsigned int min_sel;
  unsigned int max_sel;
  unsigned int uV_step;
};
```

Members

min_uV	Lowest voltage in range
min_sel	Lowest selector for range
max_sel	Highest selector for range
uV_step	Step size

Description

Specify a range of voltages for `regulator_map_linar_range` and `regulator_list_linear_range`.

Name

struct regulator_ops — regulator operations.

Synopsis

```
struct regulator_ops {
  int (* list_voltage) (struct regulator_dev *, unsigned selector);
  int (* set_voltage) (struct regulator_dev *, int min_uV, int max_uV,unsigned *se
  int (* map_voltage) (struct regulator_dev *, int min_uV, int max_uV);
  int (* set_voltage_sel) (struct regulator_dev *, unsigned selector);
  int (* get_voltage) (struct regulator_dev *);
  int (* get_voltage_sel) (struct regulator_dev *);
  int (* set_current_limit) (struct regulator_dev *,int min_uA, int max_uA);
  int (* get_current_limit) (struct regulator_dev *);
  int (* set_input_current_limit) (struct regulator_dev *, int lim_uA);
  int (* enable) (struct regulator_dev *);
  int (* disable) (struct regulator_dev *);
  int (* is_enabled) (struct regulator_dev *);
  int (* set_mode) (struct regulator_dev *, unsigned int mode);
  unsigned int (* get_mode) (struct regulator_dev *);
  int (* enable_time) (struct regulator_dev *);
  int (* set_ramp_delay) (struct regulator_dev *, int ramp_delay);
  int (* set_voltage_time_sel) (struct regulator_dev *,unsigned int old_selector,u
  int (* set_soft_start) (struct regulator_dev *);
  int (* get_status) (struct regulator_dev *);
  unsigned int (* get_optimum_mode) (struct regulator_dev *, int input_uV,int outp
  int (* set_load) (struct regulator_dev *, int load_uA);
  int (* set_bypass) (struct regulator_dev *dev, bool enable);
  int (* get_bypass) (struct regulator_dev *dev, bool *enable);
  int (* set_suspend_voltage) (struct regulator_dev *, int uV);
  int (* set_suspend_enable) (struct regulator_dev *);
  int (* set_suspend_disable) (struct regulator_dev *);
  int (* set_suspend_mode) (struct regulator_dev *, unsigned int mode);
  int (* set_pull_down) (struct regulator_dev *);
};
```

Members

list_voltage	Return one of the supported voltages, in microvolts; zero if the selector indicates a voltage that is unusable on this system; or negative errno. Selectors range from zero to one less than regulator_desc.n_voltages. Voltages may be reported in any order.
set_voltage	Set the voltage for the regulator within the range specified. The driver should select the voltage closest to min_uV.
map_voltage	Convert a voltage into a selector
set_voltage_sel	Set the voltage for the regulator using the specified selector.
get_voltage	Return the currently configured voltage for the regulator.
get_voltage_sel	Return the currently configured voltage selector for the regulator.

set_current_limit	Configure a limit for a current-limited regulator. The driver should select the current closest to max_uA.
get_current_limit	Get the configured limit for a current-limited regulator.
set_input_current_limit	Configure an input limit.
enable	Configure the regulator as enabled.
disable	Configure the regulator as disabled.
is_enabled	Return 1 if the regulator is enabled, 0 if not. May also return negative errno.
set_mode	Set the configured operating mode for the regulator.
get_mode	Get the configured operating mode for the regulator.
enable_time	Time taken for the regulator voltage output voltage to stabilise after being enabled, in microseconds.
set_ramp_delay	Set the ramp delay for the regulator. The driver should select ramp delay equal to or less than(closest) ramp_delay.
set_voltage_time_sel	Time taken for the regulator voltage output voltage to stabilise after being set to a new value, in microseconds. The function provides the from and to voltage selector, the function should return the worst case.
set_soft_start	Enable soft start for the regulator.
get_status	Return actual (not as-configured) status of regulator, as a REGULATOR_STATUS value (or negative errno)
get_optimum_mode	Get the most efficient operating mode for the regulator when running with the specified parameters.
set_load	Set the load for the regulator.
set_bypass	Set the regulator in bypass mode.
get_bypass	Get the regulator bypass mode state.
set_suspend_voltage	Set the voltage for the regulator when the system is suspended.
set_suspend_enable	Mark the regulator as enabled when the system is suspended.
set_suspend_disable	Mark the regulator as disabled when the system is suspended.
set_suspend_mode	Set the operating mode for the regulator when the system is suspended.
set_pull_down	Configure the regulator to pull down when the regulator is disabled.

Description

This struct describes regulator operations which can be implemented by regulator chip drivers.

Name

struct regulator_desc — Static regulator descriptor

Synopsis

```
struct regulator_desc {
  const char * name;
  const char * supply_name;
  const char * of_match;
  const char * regulators_node;
  int (* of_parse_cb) (struct device_node *,const struct regulator_desc *,struct r
  int id;
  bool continuous_voltage_range;
  unsigned n_voltages;
  const struct regulator_ops * ops;
  int irq;
  enum regulator_type type;
  struct module * owner;
  unsigned int min_uV;
  unsigned int uV_step;
  unsigned int linear_min_sel;
  int fixed_uV;
  unsigned int ramp_delay;
  int min_dropout_uV;
  const struct regulator_linear_range * linear_ranges;
  int n_linear_ranges;
  const unsigned int * volt_table;
  unsigned int vsel_reg;
  unsigned int vsel_mask;
  unsigned int apply_reg;
  unsigned int apply_bit;
  unsigned int enable_reg;
  unsigned int enable_mask;
  unsigned int enable_val;
  unsigned int disable_val;
  bool enable_is_inverted;
  unsigned int bypass_reg;
  unsigned int bypass_mask;
  unsigned int bypass_val_on;
  unsigned int bypass_val_off;
  unsigned int enable_time;
  unsigned int off_on_delay;
  unsigned int (* of_map_mode) (unsigned int mode);
};
```

Members

name	Identifying name for the regulator.
supply_name	Identifying the regulator supply
of_match	Name used to identify regulator in DT.

regulators_node	Name of node containing regulator definitions in DT.
of_parse_cb	Optional callback called only if of_match is present. Will be called for each regulator parsed from DT, during init_data parsing. The regulator_config passed as argument to the callback will be a copy of config passed to regulator_register, valid only for this particular call. Callback may freely change the config but it cannot store it for later usage. Callback should return 0 on success or negative ERRNO indicating failure.
id	Numerical identifier for the regulator.
continuous_voltage_range	Indicates if the regulator can set any voltage within constrains range.
n_voltages	Number of selectors available for ops.`list_voltage`.
ops	Regulator operations table.
irq	Interrupt number for the regulator.
type	Indicates if the regulator is a voltage or current regulator.
owner	Module providing the regulator, used for refcounting.
min_uV	Voltage given by the lowest selector (if linear mapping)
uV_step	Voltage increase with each selector (if linear mapping)
linear_min_sel	Minimal selector for starting linear mapping
fixed_uV	Fixed voltage of rails.
ramp_delay	Time to settle down after voltage change (unit: uV/us)
min_dropout_uV	The minimum dropout voltage this regulator can handle
linear_ranges	A constant table of possible voltage ranges.
n_linear_ranges	Number of entries in the *`linear_ranges`* table.
volt_table	Voltage mapping table (if table based mapping)
vsel_reg	Register for selector when using regulator_regmap_X_voltage_
vsel_mask	Mask for register bitfield used for selector
apply_reg	Register for initiate voltage change on the output when using regulator_set_voltage_sel_regmap
apply_bit	Register bitfield used for initiate voltage change on the output when using regulator_set_voltage_sel_regmap
enable_reg	Register for control when using regmap enable/disable ops
enable_mask	Mask for control when using regmap enable/disable ops
enable_val	Enabling value for control when using regmap enable/disable ops

disable_val	Disabling value for control when using regmap enable/disable ops
enable_is_inverted	A flag to indicate set enable_mask bits to disable when using regulator_enable_regmap and friends APIs.
bypass_reg	Register for control when using regmap set_bypass
bypass_mask	Mask for control when using regmap set_bypass
bypass_val_on	Enabling value for control when using regmap set_bypass
bypass_val_off	Disabling value for control when using regmap set_bypass
enable_time	Time taken for initial enable of regulator (in uS).
off_on_delay	guard time (in uS), before re-enabling a regulator
of_map_mode	Maps a hardware mode defined in a DeviceTree to a standard mode

Description

Each regulator registered with the core is described with a structure of this type and a struct regulator_config. This structure contains the non-varying parts of the regulator description.

Name

struct regulator_config — Dynamic regulator descriptor

Synopsis

```
struct regulator_config {
  struct device * dev;
  const struct regulator_init_data * init_data;
  void * driver_data;
  struct device_node * of_node;
  struct regmap * regmap;
  bool ena_gpio_initialized;
  int ena_gpio;
  unsigned int ena_gpio_invert:1;
  unsigned int ena_gpio_flags;
};
```

Members

dev	struct device for the regulator
init_data	platform provided init data, passed through by driver
driver_data	private regulator data
of_node	OpenFirmware node to parse for device tree bindings (may be NULL).
regmap	regmap to use for core regmap helpers if `dev_get_regmap` is insufficient.
ena_gpio_initialized	GPIO controlling regulator enable was properly initialized, meaning that >= 0 is a valid gpio identifier and < 0 is a non existent gpio.
ena_gpio	GPIO controlling regulator enable.
ena_gpio_invert	Sense for GPIO enable control.
ena_gpio_flags	Flags to use when calling `gpio_request_one`

Description

Each regulator registered with the core is described with a structure of this type and a struct regulator_desc. This structure contains the runtime variable parts of the regulator description.

Name

regulator_get — lookup and obtain a reference to a regulator.

Synopsis

```
struct regulator * regulator_get (struct device * dev, const char * id);
```

Arguments

dev device for regulator “consumer”

id Supply name or regulator ID.

Description

Returns a struct regulator corresponding to the regulator producer, or IS_ERR condition containing errno.

Use of supply names configured via regulator_set_device_supply is strongly encouraged. It is recommended that the supply name used should match the name used for the supply and/or the relevant device pins in the datasheet.

Name

regulator_get_exclusive — obtain exclusive access to a regulator.

Synopsis

```
struct regulator * regulator_get_exclusive (struct device * dev, const
char * id);
```

Arguments

dev device for regulator “consumer”

id Supply name or regulator ID.

Description

Returns a struct regulator corresponding to the regulator producer, or `IS_ERR` condition containing errno. Other consumers will be unable to obtain this regulator while this reference is held and the use count for the regulator will be initialised to reflect the current state of the regulator.

This is intended for use by consumers which cannot tolerate shared use of the regulator such as those which need to force the regulator off for correct operation of the hardware they are controlling.

Use of supply names configured via `regulator_set_device_supply` is strongly encouraged. It is recommended that the supply name used should match the name used for the supply and/or the relevant device pins in the datasheet.

Name

regulator_get_optional — obtain optional access to a regulator.

Synopsis

```
struct regulator * regulator_get_optional (struct device * dev, const
char * id);
```

Arguments

dev device for regulator “consumer”

id Supply name or regulator ID.

Description

Returns a struct regulator corresponding to the regulator producer, or IS_ERR condition containing errno.

This is intended for use by consumers for devices which can have some supplies unconnected in normal use, such as some MMC devices. It can allow the regulator core to provide stub supplies for other supplies requested using normal regulator_get calls without disrupting the operation of drivers that can handle absent supplies.

Use of supply names configured via regulator_set_device_supply is strongly encouraged. It is recommended that the supply name used should match the name used for the supply and/or the relevant device pins in the datasheet.

Name

regulator_put — "free" the regulator source

Synopsis

```
void regulator_put (struct regulator * regulator);
```

Arguments

regulator regulator source

Note

drivers must ensure that all regulator_enable calls made on this regulator source are balanced by regulator_disable calls prior to calling this function.

Name

regulator_register_supply_alias — Provide device alias for supply lookup

Synopsis

```
int regulator_register_supply_alias (struct device * dev, const char *
id, struct device * alias_dev, const char * alias_id);
```

Arguments

dev	device that will be given as the regulator “consumer”
id	Supply name or regulator ID
alias_dev	device that should be used to lookup the supply
alias_id	Supply name or regulator ID that should be used to lookup the supply

Description

All lookups for id on dev will instead be conducted for alias_id on alias_dev.

Name

regulator_unregister_supply_alias — Remove device alias

Synopsis

```
void regulator_unregister_supply_alias (struct device * dev, const char
* id);
```

Arguments

dev device that will be given as the regulator “consumer”

id Supply name or regulator ID

Description

Remove a lookup alias if one exists for id on dev.

Name

regulator_bulk_register_supply_alias — register multiple aliases

Synopsis

```
int regulator_bulk_register_supply_alias (struct device * dev, const
char *const * id, struct device * alias_dev, const char *const * alias_id,
int num_id);
```

Arguments

dev	device that will be given as the regulator “consumer”
id	List of supply names or regulator IDs
alias_dev	device that should be used to lookup the supply
alias_id	List of supply names or regulator IDs that should be used to lookup the supply
num_id	Number of aliases to register

Description

return 0 on success, an errno on failure.

This helper function allows drivers to register several supply aliases in one operation. If any of the aliases cannot be registered any aliases that were registered will be removed before returning to the caller.

Name

regulator_bulk_unregister_supply_alias — unregister multiple aliases

Synopsis

```
void regulator_bulk_unregister_supply_alias (struct device * dev, const
char *const * id, int num_id);
```

Arguments

dev device that will be given as the regulator “consumer”

id List of supply names or regulator IDs

num_id Number of aliases to unregister

Description

This helper function allows drivers to unregister several supply aliases in one operation.

Name

regulator_enable — enable regulator output

Synopsis

```
int regulator_enable (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Request that the regulator be enabled with the regulator output at the predefined voltage or current value. Calls to `regulator_enable` must be balanced with calls to `regulator_disable`.

NOTE

the output value can be set by other drivers, boot loader or may be hardwired in the regulator.

Name

regulator_disable — disable regulator output

Synopsis

```
int regulator_disable (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Disable the regulator output voltage or current. Calls to `regulator_enable` must be balanced with calls to `regulator_disable`.

NOTE

this will only disable the regulator output if no other consumer devices have it enabled, the regulator device supports disabling and machine constraints permit this operation.

Name

regulator_force_disable — force disable regulator output

Synopsis

```
int regulator_force_disable (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Forcibly disable the regulator output voltage or current.

NOTE

this *will* disable the regulator output even if other consumer devices have it enabled. This should be used for situations when device damage will likely occur if the regulator is not disabled (e.g. over temp).

Name

regulator_disable_deferred — disable regulator output with delay

Synopsis

```
int regulator_disable_deferred (struct regulator * regulator, int ms);
```

Arguments

regulator	regulator source
ms	miliseconds until the regulator is disabled

Description

Execute `regulator_disable` on the regulator after a delay. This is intended for use with devices that require some time to quiesce.

NOTE

this will only disable the regulator output if no other consumer devices have it enabled, the regulator device supports disabling and machine constraints permit this operation.

Name

regulator_is_enabled — is the regulator output enabled

Synopsis

```
int regulator_is_enabled (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Returns positive if the regulator driver backing the source/client has requested that the device be enabled, zero if it hasn't, else a negative errno code.

Note that the device backing this regulator handle can have multiple users, so it might be enabled even if `regulator_enable` was never called for this particular source.

Name

regulator_can_change_voltage — check if regulator can change voltage

Synopsis

```
int regulator_can_change_voltage (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Returns positive if the regulator driver backing the source/client can change its voltage, false otherwise. Useful for detecting fixed or dummy regulators and disabling voltage change logic in the client driver.

Name

regulator_count_voltages — count `regulator_list_voltage` selectors

Synopsis

```
int regulator_count_voltages (struct regulator * regulator);
```

Arguments

`regulator` regulator source

Description

Returns number of selectors, or negative errno. Selectors are numbered starting at zero, and typically correspond to bitfields in hardware registers.

Name

regulator_list_voltage — enumerate supported voltages

Synopsis

```
int regulator_list_voltage (struct regulator * regulator, unsigned selector);
```

Arguments

regulator regulator source

selector identify voltage to list

Context

can sleep

Description

Returns a voltage that can be passed to *regulator_set_voltage*(), zero if this selector code can't be used on this system, or a negative errno.

Name

regulator_get_hardware_vsel_register — get the HW voltage selector register

Synopsis

```
int regulator_get_hardware_vsel_register (struct regulator * regulator,
unsigned * vsel_reg, unsigned * vsel_mask);
```

Arguments

regulator regulator source

vsel_reg voltage selector register, output parameter

vsel_mask mask for voltage selector bitfield, output parameter

Description

Returns the hardware register offset and bitmask used for setting the regulator voltage. This might be useful when configuring voltage-scaling hardware or firmware that can make I2C requests behind the kernel's back, for example.

On success, the output parameters *vsel_reg* and *vsel_mask* are filled in and 0 is returned, otherwise a negative errno is returned.

Name

regulator_list_hardware_vsel — get the HW-specific register value for a selector

Synopsis

```
int regulator_list_hardware_vsel (struct regulator * regulator, unsigned selector);
```

Arguments

regulator regulator source

selector identify voltage to list

Description

Converts the selector to a hardware-specific voltage selector that can be directly written to the regulator registers. The address of the voltage register can be determined by calling *regulator_get_hardware_vsel_register*.

On error a negative errno is returned.

Name

regulator_get_linear_step — return the voltage step size between VSEL values

Synopsis

```
unsigned int regulator_get_linear_step (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Returns the voltage step size between VSEL values for linear regulators, or return 0 if the regulator isn't a linear regulator.

Name

regulator_is_supported_voltage — check if a voltage range can be supported

Synopsis

```
int regulator_is_supported_voltage (struct regulator * regulator, int
min_uV, int max_uV);
```

Arguments

regulator	Regulator to check.
min_uV	Minimum required voltage in uV.
max_uV	Maximum required voltage in uV.

Description

Returns a boolean or a negative error code.

Name

regulator_set_voltage — set regulator output voltage

Synopsis

```
int regulator_set_voltage (struct regulator * regulator, int min_uV,
int max_uV);
```

Arguments

regulator	regulator source
min_uV	Minimum required voltage in uV
max_uV	Maximum acceptable voltage in uV

Description

Sets a voltage regulator to the desired output voltage. This can be set during any regulator state. IOW, regulator can be disabled or enabled.

If the regulator is enabled then the voltage will change to the new value immediately otherwise if the regulator is disabled the regulator will output at the new voltage when enabled.

NOTE

If the regulator is shared between several devices then the lowest request voltage that meets the system constraints will be used. Regulator system constraints must be set for this regulator before calling this function otherwise this call will fail.

Name

regulator_set_voltage_time — get raise/fall time

Synopsis

```
int regulator_set_voltage_time (struct regulator * regulator, int old_uV, int new_uV);
```

Arguments

regulator	regulator source
old_uV	starting voltage in microvolts
new_uV	target voltage in microvolts

Description

Provided with the starting and ending voltage, this function attempts to calculate the time in microseconds required to rise or fall to this new voltage.

Name

regulator_set_voltage_time_sel — get raise/fall time

Synopsis

```
int regulator_set_voltage_time_sel (struct regulator_dev * rdev,
unsigned int old_selector, unsigned int new_selector);
```

Arguments

rdev	regulator source device
old_selector	selector for starting voltage
new_selector	selector for target voltage

Description

Provided with the starting and target voltage selectors, this function returns time in microseconds required to rise or fall to this new voltage

Drivers providing ramp_delay in regulation_constraints can use this as their `set_voltage_time_sel` operation.

Name

regulator_sync_voltage — re-apply last regulator output voltage

Synopsis

```
int regulator_sync_voltage (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Re-apply the last configured voltage. This is intended to be used where some external control source the consumer is cooperating with has caused the configured voltage to change.

Name

regulator_get_voltage — get regulator output voltage

Synopsis

```
int regulator_get_voltage (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

This returns the current regulator voltage in uV.

NOTE

If the regulator is disabled it will return the voltage value. This function should not be used to determine regulator state.

Name

regulator_set_current_limit — set regulator output current limit

Synopsis

```
int regulator_set_current_limit (struct regulator * regulator, int min_uA, int max_uA);
```

Arguments

regulator	regulator source
min_uA	Minimum supported current in uA
max_uA	Maximum supported current in uA

Description

Sets current sink to the desired output current. This can be set during any regulator state. IOW, regulator can be disabled or enabled.

If the regulator is enabled then the current will change to the new value immediately otherwise if the regulator is disabled the regulator will output at the new current when enabled.

NOTE

Regulator system constraints must be set for this regulator before calling this function otherwise this call will fail.

Name

regulator_get_current_limit — get regulator output current

Synopsis

```
int regulator_get_current_limit (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

This returns the current supplied by the specified current sink in uA.

NOTE

If the regulator is disabled it will return the current value. This function should not be used to determine regulator state.

Name

regulator_set_mode — set regulator operating mode

Synopsis

```
int regulator_set_mode (struct regulator * regulator, unsigned int mode);
```

Arguments

regulator regulator source

mode operating mode - one of the REGULATOR_MODE constants

Description

Set regulator operating mode to increase regulator efficiency or improve regulation performance.

NOTE

Regulator system constraints must be set for this regulator before calling this function otherwise this call will fail.

Name

regulator_get_mode — get regulator operating mode

Synopsis

```
unsigned int regulator_get_mode (struct regulator * regulator);
```

Arguments

regulator regulator source

Description

Get the current regulator operating mode.

Name

regulator_set_load — set regulator load

Synopsis

```
int regulator_set_load (struct regulator * regulator, int uA_load);
```

Arguments

regulator regulator source

uA_load load current

Description

Notifies the regulator core of a new device load. This is then used by DRMS (if enabled by constraints) to set the most efficient regulator operating mode for the new regulator loading.

Consumer devices notify their supply regulator of the maximum power they will require (can be taken from device datasheet in the power consumption tables) when they change operational status and hence power state. Examples of operational state changes that can affect power

consumption are

-

o Device is opened / closed. o Device I/O is about to begin or has just finished. o Device is idling in between work.

This information is also exported via sysfs to userspace.

DRMS will sum the total requested load on the regulator and change to the most efficient operating mode if platform constraints allow.

On error a negative errno is returned.

Name

regulator_allow_bypass — allow the regulator to go into bypass mode

Synopsis

```
int regulator_allow_bypass (struct regulator * regulator, bool enable);
```

Arguments

regulator Regulator to configure

enable enable or disable bypass mode

Description

Allow the regulator to go into bypass mode if all other consumers for the regulator also enable bypass mode and the machine constraints allow this. Bypass mode means that the regulator is simply passing the input directly to the output with no regulation.

Name

regulator_register_notifier — register regulator event notifier

Synopsis

```
int regulator_register_notifier (struct regulator * regulator, struct
notifier_block * nb);
```

Arguments

regulator regulator source

nb notifier block

Description

Register notifier block to receive regulator events.

Name

regulator_unregister_notifier — unregister regulator event notifier

Synopsis

```
int regulator_unregister_notifier (struct regulator * regulator, struct
notifier_block * nb);
```

Arguments

regulator regulator source

nb notifier block

Description

Unregister regulator event notifier block.

Name

regulator_bulk_get — get multiple regulator consumers

Synopsis

```
int regulator_bulk_get (struct device * dev, int num_consumers, struct
regulator_bulk_data * consumers);
```

Arguments

dev	Device to supply
num_consumers	Number of consumers to register
consumers	Configuration of consumers; clients are stored here.

Description

return 0 on success, an errno on failure.

This helper function allows drivers to get several regulator consumers in one operation. If any of the regulators cannot be acquired then any regulators that were allocated will be freed before returning to the caller.

Name

regulator_bulk_enable — enable multiple regulator consumers

Synopsis

```
int regulator_bulk_enable (int num_consumers, struct
regulator_bulk_data * consumers);
```

Arguments

num_consumers Number of consumers

consumers Consumer data; clients are stored here. *return* 0 on success, an errno on failure

Description

This convenience API allows consumers to enable multiple regulator clients in a single API call. If any consumers cannot be enabled then any others that were enabled will be disabled again prior to return.

Name

regulator_bulk_disable — disable multiple regulator consumers

Synopsis

```
int regulator_bulk_disable (int num_consumers, struct
regulator_bulk_data * consumers);
```

Arguments

num_consumers Number of consumers

consumers Consumer data; clients are stored here. *return* 0 on success, an errno on failure

Description

This convenience API allows consumers to disable multiple regulator clients in a single API call. If any consumers cannot be disabled then any others that were disabled will be enabled again prior to return.

Name

regulator_bulk_force_disable — force disable multiple regulator consumers

Synopsis

```
int regulator_bulk_force_disable (int num_consumers, struct
regulator_bulk_data * consumers);
```

Arguments

num_consumers Number of consumers

consumers Consumer data; clients are stored here. *return* 0 on success, an errno on failure

Description

This convenience API allows consumers to forcibly disable multiple regulator clients in a single API call.

NOTE

This should be used for situations when device damage will likely occur if the regulators are not disabled (e.g. over temp). Although regulator_force_disable function call for some consumers can return error numbers, the function is called for all consumers.

Name

regulator_bulk_free — free multiple regulator consumers

Synopsis

```
void regulator_bulk_free (int num_consumers, struct regulator_bulk_data
* consumers);
```

Arguments

num_consumers	Number of consumers
consumers	Consumer data; clients are stored here.

Description

This convenience API allows consumers to free multiple regulator clients in a single API call.

Name

regulator_notifier_call_chain — call regulator event notifier

Synopsis

```
int regulator_notifier_call_chain (struct regulator_dev * rdev, unsigned long event, void * data);
```

Arguments

rdev regulator source

event notifier block

data callback-specific data.

Description

Called by regulator drivers to notify clients a regulator event has occurred. We also notify regulator clients downstream. Note lock must be held by caller.

Name

regulator_mode_to_status — convert a regulator mode into a status

Synopsis

```
int regulator_mode_to_status (unsigned int mode);
```

Arguments

mode Mode to convert

Description

Convert a regulator mode into a status.

Name

regulator_register — register regulator

Synopsis

```
struct regulator_dev * regulator_register (const struct regulator_desc
* regulator_desc, const struct regulator_config * cfg);
```

Arguments

regulator_desc	regulator to register
cfg	runtime configuration for regulator

Description

Called by regulator drivers to register a regulator. Returns a valid pointer to struct regulator_dev on success or an ERR_PTR on error.

Name

regulator_unregister — unregister regulator

Synopsis

```
void regulator_unregister (struct regulator_dev * rdev);
```

Arguments

rdev regulator to unregister

Description

Called by regulator drivers to unregister a regulator.

Name

regulator_suspend_prepare — prepare regulators for system wide suspend

Synopsis

```
int regulator_suspend_prepare (suspend_state_t state);
```

Arguments

state system suspend state

Description

Configure each regulator with it's suspend operating parameters for state. This will usually be called by machine suspend code prior to supending.

Name

regulator_suspend_finish — resume regulators from system wide suspend

Synopsis

```
int regulator_suspend_finish ( void);
```

Arguments

void no arguments

Description

Turn on regulators that might be turned off by regulator_suspend_prepare and that should be turned on according to the regulators properties.

Name

regulator_has_full_constraints — the system has fully specified constraints

Synopsis

```
void regulator_has_full_constraints ( void);
```

Arguments

void no arguments

Description

Calling this function will cause the regulator API to disable all regulators which have a zero use count and don't have an always_on constraint in a late_initcall.

The intention is that this will become the default behaviour in a future kernel release so users are encouraged to use this facility now.

Name

rdev_get_drvdata — get rdev regulator driver data

Synopsis

```
void * rdev_get_drvdata (struct regulator_dev * rdev);
```

Arguments

rdev regulator

Description

Get rdev regulator driver private data. This call can be used in the regulator driver context.

Name

regulator_get_drvdata — get regulator driver data

Synopsis

```
void * regulator_get_drvdata (struct regulator * regulator);
```

Arguments

regulator regulator

Description

Get regulator driver private data. This call can be used in the consumer driver context when non API regulator specific functions need to be called.

Name

regulator_set_drvdata — set regulator driver data

Synopsis

```
void regulator_set_drvdata (struct regulator * regulator, void * data);
```

Arguments

regulator regulator

data data

Name

rdev_get_id — get regulator ID

Synopsis

```
int rdev_get_id (struct regulator_dev * rdev);
```

Arguments

rdev regulator