

# **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.

---



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 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

<code>supply</code>	The name of the supply. Initialised by the user before using the bulk regulator APIs.
<code>consumer</code>	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.



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

`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_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

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

<i>rdev</i>	regulator source device
<i>old_selector</i>	selector for starting voltage
<i>new_selector</i>	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_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_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_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_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