Voltage and current regulator API

Author: Liam Girdwood
Author: Mark Brown

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:

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 (PMIC)

> 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.

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 :c:func:`regulator_enable()` and :c:func:`regulator_disable()` functions to enable and disable regulators. Calls to the two functions must be balanced.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst`, **line 78**); ***backlink***
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst`, **line 78**); ***backlink***
>
> Unknown interpreted text role "c:func".

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 :c:func:`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.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 83); *backlink***
>
> Unknown interpreted text role "c:func".

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 :c:func:`regulator_set_voltage()` and :c:func:`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.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 97); *backlink***
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 97); *backlink***
>
> Unknown interpreted text role "c:func".

Callbacks

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

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.

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 :c:type:`regulator_consumer_supply`. This is done at driver registration time as part of the machine constraints.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 132); *backlink***
>
> Unknown interpreted text role "c:type".

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.

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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 160)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/regulator/consumer.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 163)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/regulator/machine.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 166)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/regulator/driver.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]regulator.rst, line 169)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/regulator/core.c
   :export:
```