The Regulator-Consumer API

Linux Kernel Voltage Regulator API

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

Terminology

Regulator use cases:

- Consumer
- Driver
- Machine

Regulator “consumer”:

- Something powered by a voltage regulator
- Constrained by datasheet, board, and regulator limits

Terminology

Regulator “driver”:

- Code to control a power supply
- Code to help supplies pick their ideal modes

Regulator “machine”:

- Tells Linux what regulators are tied to what consumers
- Expresses limits imposed by the platform

Terminology

“Regulator”:

- An abstraction, really
- Represents a power node in the circuit
- Might be one-to-one with a regulator device
- (... or not, we really don’t care)

```
#include <linux/regulator/consumer.h>
#include <linux/regulator/machine.h>
```

regulator_get()

Returns a handle to a regulator device:

- Needed for all other regulator operations
- Prevents regulator driver from unbinding
- (Increments regulator device reference count)

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

regulator_get()

```
int foo_probe(...)
{
  ...
  /* turn on LVSW14 */
  struct regulator *lvsw14
      = regulator_get(NULL, "LVSW14");   /* no! */
  regulator_enable(lvsw14);              /* no! */
  ...
}
```

regulator_get()

```
int foo_probe(...)
{
  ...
  /* turn on our VDD pin */
  struct regulator *vdd
      = regulator_get(dev, "VDD");      /* yes! */
  regulator_enable(vdd);                /* yes! */
  ...
}
```

regulator_get()

```
int foo_probe(...)
{
  ...
  /* TODO: no! */
  r = regulator_get(NULL, foo->board_data->vdd_name);
  ...
  regulator_enable(r);
  ...
}
```

`regulator_get()`

Why the device reference?

- Tracks device constraint requests
- Establishes (temporary) parentage
- Helps regulator framework clean up properly later

regulator_put()

Releases a handle to a regulator device:

- Decrements regulator device reference count
- Doesn't disturb regulator state

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

regulator_put()

```
int foo_remove(...)
{
  ...
  regulator_put(r);
  ...
}
```

`regulator_enable()`

"Turns on" a regulator output:

- ... if it isn't on already, that is
- Requests to enable and disable must be balanced
- Returned value indicates error if nonzero

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

regulator_disable()

“Turns off” a regulator output:

- ... if the regulator isn’t enabled elsewhere
- Turns off upstream regulators if possible
- Returns nonzero on error (not status of regulator)

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

regulator_set_voltage()

Specifies a suitable voltage range:

- Does not turn on regulator
- Adjustment is immediate if regulator is turned on
- Range is checked against other consumers
- Use the range from your device datasheet

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

regulator_is_enabled()

Returns nonzero if regulator is active:

- Watch out for test-then-act!

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

regulator_is_enabled()

```
int regulator_is_enabled(...)
{
  ...
  mutex_lock(&regulator->rdev->mutex);
  ret = _regulator_is_enabled(regulator->rdev);
  mutex_unlock(&regulator->rdev->mutex);
  ...
  return ret;
}
```

Regulator Notifiers

Notification of regulator state changes:

- Changes in voltage
- Error conditions

Beware: test-then-act!

Regulator Notifiers

```
REGULATOR_EVENT_UNDER_VOLTAGE
REGULATOR_EVENT_OVER_CURRENT
REGULATOR_EVENT_REGULATION_OUT
REGULATOR_EVENT_FAIL
REGULATOR_EVENT_OVER_TEMP
REGULATOR_EVENT_FORCE_DISABLE
REGULATOR_EVENT_VOLTAGE_CHANGE
REGULATOR_EVENT_DISABLE
```

regulator_register_notifier()

Request a callback on notifier events:

- Watch out for test-then-act!

```
int regulator_request_notifier(struct regulator *r,
                               struct notifier_block *nb);
```

regulator_register_notifier()

```
static struct notifier_block my_nb;

static int my_callback(struct notifier_block *nb,
                       unsigned long event,
                       void *ignored)
{
  ...
}

...
nb.notifier_call = my_callback;
regulator_register_notifier(r, &nb);
...
```

regulator_register_notifier()

```
static struct notifier_block my_nb;

static int my_callback(struct notifier_block *nb,
                       unsigned long event,
                       void *ignored)
{
  struct my *my = container_of(nb, struct my, nb);
  int vdd;

  /* no! (mutex) */
  vdd = regulator_get_voltage(my->r);
}
```

regulator_register_notifier()

```
diff --git a/drivers/regulator/core.c b/drivers/regulator/core.c
index 5b2328d..9a57ee6 100644
--- a/drivers/regulator/core.c
+++ b/drivers/regulator/core.c
@@ -643,7 +643,8 @@ static int update_voltage(struct regulator *regulator ...
    if (!ret)
            goto out;

-       _notifier_call_chain(rdev, REGULATOR_EVENT_VOLTAGE_CHANGE, NULL);
+       _notifier_call_chain(rdev, REGULATOR_EVENT_VOLTAGE_CHANGE,
+                            (void*)_regulator_get_voltage(rdev));
```

regulator_register_notifier()

```
static struct notifier_block my_nb;

static int my_callback(struct notifier_block *nb,
                       unsigned long event,
                       void *v)
{
  struct my *my = container_of(nb, struct my, nb);
  int vdd;

  if (event == REGULATOR_EVENT_VOLTAGE_CHANGE)
    vdd = v;
  ...
}
```

The Regulator-Consumer API

Linux Kernel Voltage Regulator API

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer