The Regulator Consumer-Supply 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>
```

`struct regulator_consumer_supply`

Describes the power tree:

- Supplies for consumer devices

struct regulator_consumer_supply

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

#define REGULATOR_SUPPLY(_name, _dev_name) \
{ \
  .dev_name = _dev_name, \
  .supply = _name, \
}
```

struct regulator_consumer_supply

```
/* no! */
static struct regulator_consumer_supply vregs[] = {
  REGULATOR_SUPPLY("8901_lvs2", NULL),
};

/* yes! */
static struct regulator_consumer_supply vregs[] = {
  REGULATOR_SUPPLY("IOVDD", "5-0018"),
};
```

`regulator_register()`

Registers a regulator device:

- Used by regulator driver authors
- Binds a regulator to a power node

```
struct regulator_dev *
regulator_register(const struct regulator_desc
                     *regulator_desc,
                   const struct regulator_config
                     *config)
```

`struct regulator_desc`

Describes a regulator device:

- Fixed characteristics of the regulator output
- Voltage range capability, etc.

```
struct regulator_desc;
```

struct regulator_desc

```
struct regulator_desc {
  const char *name;
  const char *supply_name;
  ...
  struct regulator_ops *ops;
  enum regulator_type type;
  ...
  unsigned int min_uV;
  unsigned int uV_step;
  ...
  const unsigned int *volt_table;
  ...
};
```

struct regulator_config

Describes a regulator device installation:

- What devices it supplies

```
struct regulator_config {
  struct device *dev;
  const struct regulator_init_data *init_data;
  void *driver_data;
  ...
};
```

struct regulator_init_data

Describes a regulator device installation:

- Platform initialization data

```
struct regulator_init_data;
```

struct regulator_init_data

```
struct regulator_init_data {
  ...
  int num_consumer_supplies;
  struct regulator_consumer_supply *consumer_supplies;
  ...
};
```

struct regulator_init_data

```
static struct regulator_consumer_supply vregs[] = {
  REGULATOR_SUPPLY("IOVDD", "5-0018"),
  REGULATOR_SUPPLY("IOVDD", "7-003a"),
  ...
};

static struct regulator_init_data vreginit = {
  ...
  .num_consumer_supplies = ARRAY_SIZE(vregs),
  .consumer_supplies = vregs,
  ...
};
```

struct regulator_init_data

```
static struct i2c_board_info foovreg = {
  ...
  .platform_data &vreginit,
  ...
};

i2c_new_device(..., &foovreg);
```

struct regulator_init_data

```
static int foovreg_probe(struct i2c_client *dev, ...)
{
  ...
  regulator_register(...);
  ...
};
```

On Device Names

Where does the device name come from?

- Hard-code after trial-and-error
- Parse after adding the device (but before driver!)

On Device Names

```
struct i2c_client *c = i2c_new_device(...);

static struct regulator_consumer_supply vregs[] = {
  ...
  REGULATOR_SUPPLY("IOVDD", dev_name(&c->dev)),
  ...
};
```

On Device Names

```
/**
 * i2c_new_device - instantiate an i2c device
 *
 * ... This call is not appropriate for use by mainboard
 * initialization logic, which usually runs during an
 * arch_initcall() long before any i2c_adapter could
 * exist...
 *
 */
 struct i2c_client *
 i2c_new_device(struct i2c_adapter *adap, ...
 {
   ...
```

The Regulator Consumer-Supply API

Linux Kernel Voltage Regulator API

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer