regmap

The power of subsystems and abstractions

Overview

- **The quality of the subsystems is key to Linux**
  - Factor common code out of drivers
  - Simplify driver development
  - Encourage best practice
  - Maximise the impact of features
- **regmap provides a good case study**
  - Register I/O for I2C and SPI
  - Originally in ASoC for audio CODECs
  - Traditionally open coded in drivers
  - Now provides benefits for totally different device classes
  - Nothing like it in other operating systems

In the beginning…

- **ASoC CODEC drivers need to provide configuration to users**
- **Saw that there were lots of register bitfields like:**
  - R0 INL_VOL [5:0]: Left input PGA Volume -23.25dB to +24.00dB in 0.75dB steps
  - R1 INR_VOL [5:0]: Right input PGA Volume -23.25dB to +24.00dB in 0.75dB steps
- **Factored this out into standard helpers for drivers:**
  - `SOC_DOUBLE_R_TLV("Capture Volume", WM8962_LEFT_INPUT_VOLUME, WM8962_RIGHT_INPUT_VOLUME, 0, 63, 0, inpga_tlv),`
  - Supported with CODEC callbacks:
    - `int read(struct snd_soc_codec *codec, int reg);`
    - `int write(struct snd_soc_codec *codec, int reg, int value);`

Quick wins

- Save some boilerplate
- Simple factor out of one very common operation
  - `snd_soc_update_bits(struct snd_soc_codec *codec, int reg, int mask, int val);`
- Can suppress no op changes
- Make best practice clear and obvious

Register dumps

- **Using:**
  - Register read and write operations
  - Ideally also the maximum register address
- **The subsystem can provide register dumps as a standard feature:**

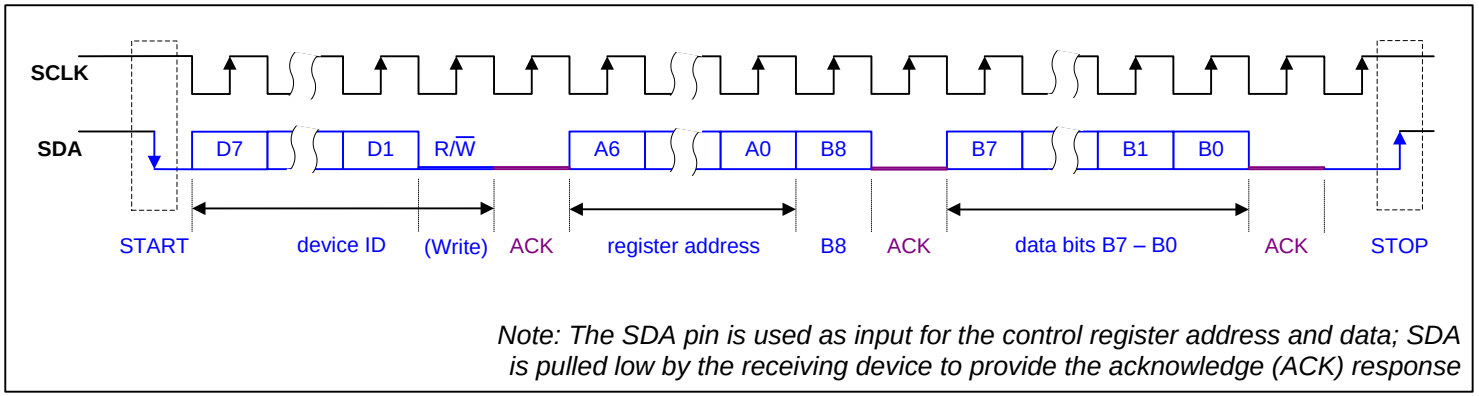
```
0000: abcd
0001: 5e32
```

- **Common output format**
- **Support for reading only specific registers**
- **Write support**
- **Enabled by previous factoring out**

Register caches

- **Had been open coded in drivers**
- **Layered in with a little bit more data**
  - Register default values
  - Volatile registers
- **Really nice feature**
  - Many devices don't support readback
  - Performance improvement
  - Simplifies suspend and resume

- **The hardware interface is very consistent over devices:**



Note: The SDA pin is used as input for the control register address and data; SDA is pulled low by the receiving device to provide the acknowledge (ACK) response

  - Register followed by value, big endian
- **Standard implementation of read and write**
- **Subsystem ensures all drivers get the fiddly bits right**
  - Byte swapping
  - Interoperability with controller features
  - Performance tricks

Factoring out regmap

- **These patterns are present in many other devices**
  - PMICs
  - Input controllers
  - GPIO expanders
- **Move the code out of ASoC**
  - drivers/base/regmap
- **Gradual merge**
  - v3.1: simple register I/O functionality for I2C and SPI
  - v3.2: caches, tracepoints and debugfs

Device description

```
struct regmap_config {
    int reg_bits;
    int pad_bits;
    int val_bits;

    bool (*writeable_reg)(struct device *dev, unsigned int reg);
    bool (*readable_reg)(struct device *dev, unsigned int reg);
    bool (*volatile_reg)(struct device *dev, unsigned int reg);
    bool (*precious_reg)(struct device *dev, unsigned int reg);

    unsigned int max_register;
    const struct reg_default *reg_defaults;
    unsigned int num_reg_defaults;
};
```

Core API

```
struct regmap *devm_regmap_init_i2c(struct i2c_client *i2c,
                                    const struct regmap_config *config);

int regmap_read(struct regmap *map, unsigned int reg,
                unsigned int *val);
int regmap_write(struct regmap *map, unsigned int reg,
                 unsigned int val);
int regmap_update_bits(struct regmap *map, unsigned int reg,
                       unsigned int mask, unsigned int val);

int regcache_sync(struct regmap *map);
void regcache_cache_only(struct regmap *map, bool enable);
void regcache_mark_dirty(struct regmap *map);
```

Complex register caches

- **Initially caches just used a flat array**
- **Not so good when caching devices with 32 bit addresses**
- **Solved with better cache types**
  - rbtree stores blocks of contiguous registers in a red/black tree (436 lines)
  - Compressed stores blocks of compressed data (380 lines)
- **Both rely on existing kernel libraries**

```
enum regcache_type cache_type;
```

Tracepoints/Logging

- Simple, low overhead logging subsystem
- Can be built in all the time and running all the time
- Standard format allows reusable tooling in userspace
- Key tracepoints for regmap:
  - regmap_reg_write 0-001a reg=1a val=3c
  - regmap_reg_read 0-001a reg=1 val=3c
- See more in debugfs/trace/events/regmap/
- Also a simple define LOG_DEVICE for early init logging

Register patches

- **Magic register writes done at device startup**
  - Performance tuning
  - Workarounds
- **Integrated into cache sync**

```
int regmap_register_patch(struct regmap *map,
      const struct reg_default *regs,
      int num_regs);
```


Paging support

- Common hardware pattern, adds another level of addressing
- Supported in regmap by creating virtual registers
- Standard interface allows upper level code to ignore paging

```
struct regmap_range_cfg {
        const char *name;
        unsigned int range_min; unsigned int range_max;
        unsigned int selector_reg; unsigned int selector_mask;
        int selector_shift;
        unsigned int window_start; unsigned int window_len;
};
```


MMIO support

- Cache and diagnostic infrastructure isn't just useful to I2C and SPI
- Allows really simple integration with runtime PM

Interrupt controller

- **Common patterns in interrupt controllers**
  - Status register
  - Mask register
- **Lots of fiddly stuff with interrupt core due to blocking in "interrupt" context**
- **Frequently cut'n'pasted**
  - Including the comments!

Interrupt controller

```
struct regmap_irq {
        unsigned int reg_offset; unsigned int mask;
};

struct regmap_irq_chip {
        const char *name;
        unsigned int status_base;
        unsigned int mask_base;
        unsigned int ack_base;
        unsigned int wake_base;
        unsigned int irq_reg_stride;
        unsigned int mask_invert;
        bool runtime_pm;

        const struct regmap_irq *irqs;
        int num_irqs;
};
```

Feature summary

- v3.1: simple register I/O functionality for I2C and SPI
- v3.2: caches, tracepoints and debugfs
- v3.3: interrupt controller
- v3.4: patches
- v3.5: MMIO bus
- v3.6: paging support

- regmap based helpers for ASoC, regulator and IRQ

Future work

- Support for devices providing their own set and get register operations without formatting (eg, for USB)
- Performance improvements in cache sync
- Combine rbtree and compressed into a single cache type
- Common helpers for register access patterns
- Simplify chips with dense interrupt controller bitfields
- More helpers for subsystems

Thanks

- Liam Girdwood, ASoC comaintainer and original author
- Dimitris Papastamos, contributed advanced caches
- Lars-Peter Clausen, early adopter & bug fixer
- Stephen Warren, contributed regmap-mmio
- Krystian Garbaciak, contributed paging support
- Laxman Dewangan, contributed a bunch of improvements

Summary

- Small abstractions pave the way for bigger ones
- Solving things at the right level saves time and effort
- Register I/O is very simple on Linux