[regmap使用介绍](http://blog.csdn.net/hellowxwworld/article/details/10737569)

内核3.1引入一套新的API regmap，目的是提取出关于I2C SPI irq等相关注册、使能以及读写的公共部分，以提高代码的可重用性，并且使得在使用如上内核基础组件时变得更为简单易用。

**0 基础结构**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hellowxwworld/article/details/10737569)

1. **struct** regmap\_config {
2. **int** reg\_bits; // 寄存器地址的位数，必须配置，例如I2C寄存器地址位数为 8
3. **int** pad\_bits; // 寄存器值的位数，必须配置
4. **int** val\_bits;
5. **bool** (\*writeable\_reg)(**struct** device \*dev, unsigned **int** reg); // 可写寄存器回调，maintain一个可写寄存器表
6. **bool** (\*readable\_reg)(**struct** device \*dev, unsigned **int** reg); // 可读寄存器回调, maintain一个可读寄存器表
7. **bool** (\*volatile\_reg)(**struct** device \*dev, unsigned **int** reg); // 可要求读写立即生效的寄存器回调，不可以被cache,maintain一个可立即生效寄存器表
8. **bool** (\*precious\_reg)(**struct** device \*dev, unsigned **int** reg); // 要求寄存器数值维持在一个数值范围才正确，maintain一个数值准确表
9. unsigned **int** max\_register; // max\_register: 最大寄存器地址
10. **const** **struct** reg\_default \*reg\_defaults;
11. unsigned **int** num\_reg\_defaults;
12. **enum** regcache\_type cache\_type;
13. **const** **void** \*reg\_defaults\_raw;
14. unsigned **int** num\_reg\_defaults\_raw;
15. u8 read\_flag\_mask;
16. u8 write\_flag\_mask;
17. };

**1 初始化regmap**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hellowxwworld/article/details/10737569)

1. regmap\_init\_i2c(**struct** i2c\_client \*i2c, **struct** regmap\_config \*config);
2. regmap\_init\_spi(**struct** spi\_device \*spi, strcut regmap\_config \*config);
3. regmap\_add\_irq\_chip(**struct** regmap \*map, **int** irq, **int** irq\_flags, **int** irq\_base, **struct** regmap\_irq\_chip \*chip, **struct** regmap\_irq\_chip\_data \*\*data);

前两个是用来关联i2c或者spi设备和regmap\_config的，第三个用来在关联后的regmap上注册 irq

**2 使用regmap**

在初始化好regmap之后，就可以调用regmap提供的read/write/update等操作了。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hellowxwworld/article/details/10737569)

1. regmap\_write(**struct** regmap \*map, **int** reg, **int** val); // 向reg写入val
2. regmap\_raw\_write(**struct** regmap \*map, **int** reg, **void** \*val, **size\_t** val\_len);  // 向reg写入指定长度的数据，数据存放在val中
3. regmap\_read(**struct** regmap \*map, **int** reg, **int** \*val); // 读取reg的数据到val中
4. regmap\_raw\_read(**struct** regmap \*map, **int** reg, **void** \*val, **size\_t** val\_len); // 读取reg中指定长度的数据
5. regmap\_bulk\_read(**struct** regmap \*map, **int** reg, **void** \*val, **size\_t** val\_count); // 读取从reg开始之后val\_count个寄存器的数据到val中
6. regmap\_update\_bits(**struct** regmap \*map, **int** reg, **int** mask, **int** val); // 更新reg寄存器中mask指定的位
7. regcache\_cache\_bypass(arizona->regmap, **true**); // 设置读写寄存器不通过cache模式而是bypass模式，读写立即生效，一般在audio等确保时序性驱动中用到

**3 释放regmap**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hellowxwworld/article/details/10737569)

1. regmap\_exit(**struct** regmap \*map);