STM32 使用 BSRR 和 BRR 寄存器快速操作 GPIO 端口

STM32 的每个 GPIO 端口都有两个特别的寄存器，GPIOx_BSRR 和 GPIOx_BRR 寄存器，通过这两个寄存器可以直接对对应的 GPIOx 端口置'1'或置'0'。
GPIOx_BSRR 的高 16 位中每一位对应端口 x 的每个位，对高 16 位中的某位置'1'则端口 x 的对应位被清'0'；寄存器中的位置'0'，则对它对应的位不起作用。
GPIOx_BSRR 的低 16 位中每一位也对应端口 x 的每个位，对低 16 位中的某位置'1'则它对应的端口位被置'1'；寄存器中的位置'0'，则对它对应的端口不起作用。

简单地说 GPIOx_BSRR 的高 16 位称作清除寄存器，而 GPIOx_BSRR 的低 16 位称作设置寄存器。另一个寄存器 GPIOx_BRR 只有低 16 位有效，与 GPIOx_BSRR 的高 16 位具有相同功能。

举个例子说明如何使用这两个寄存器和所体现的优势。例如 GPIOE 的 16 个 IO 都被设置成输出，而每次操作仅需要改变低 8 位的数据而保持高 8 位不变，假设新的 8 位数据在变量 Newdata 中，

这个要求可以通过操作这两个寄存器实现，STM32 的固件库中有两个函数 GPIO_SetBits() 和 GPIO_ResetBits()使用了这两个寄存器操作端口。

上述要求可以这样实现：

```
GPIO_SetBits(GPIOE, Newdata & 0xff);
GPIO_ResetBits(GPIOE, (~Newdata & 0xff));
```

也可以直接操作这两个寄存器：

```
GPIOE->BSRR = Newdata & 0xff;
GPIOE->BRR = ~Newdata & 0xff;
```

当然还可以一次完成对 8 位的操作：

```
GPIOE->BSRR = (Newdata & 0xff) | (~Newdata & 0xff)<<16;
```

从最后这个操作可以看出使用 BSRR 寄存器，可以实现 8 个端口位的同时修改操作。

如果不是用 BRR 和 BSRR 寄存器，则上述要求就需要这样实现：

```
GPIOE->ODR = GPIOE->ODR & 0xff00 | Newdata;
```

使用 BRR 和 BSRR 寄存器可以方便地快速地实现对端口某些特定位的操作，而不影响其它位的状态。

比如希望快速地对 GPIOE 的位 7 进行翻转，则可以：

```
GPIOE->BSRR = 0x80; //  置'1'
GPIOE->BRR = 0x80; //  置'0'
```

如果使用常规'读-改-写'的方法：

```
GPIOE->ODR = GPIOE->ODR | 0x80; //  置'1'
GPIOE->ODR = GPIOE->ODR & 0xFF7F; //  置'0'
```

有人问是否 BSRR 的高 16 位是多余的，请看下面这个例子：

假如你想在一个操作中对 GPIOE 的位 7 置'1'，位 6 置'0'，则使用 BSRR 非常方便：

```
GPIOE->BSRR = 0x4080;
```

如果没有 BSRR 的高 16 位，则要分 2 次操作，结果造成位 7 和位 6 的变化不同步！

```
GPIOE->BSRR = 0x80;
GPIOE->BRR = 0x40;
```