**Bitwise in C**

**Setting a bit**

Use the bitwise OR operator ( | ) to set a bit.

number |= 1 << x;

That will set bit x.

**Clearing a bit**

Use the bitwise AND operator (&) to clear a bit.

number &= ~(1 << x);

That will clear bit x. You must invert the bit string with the bitwise NOT operator (~), then AND it.

**Toggling a bit**

The XOR operator (^) can be used to toggle a bit.

number ^= 1 << x;

That will toggle bit x.

**Checking a bit**

To check a bit, shift the number x to the right, then bitwise AND it:

bit = (number >> x) & 1;

That will put the value of bit x into the variable bit.

**Changing the nth bit to x**

Setting the nth bit to either 1 or 0 can be achieved with the following:

number ^= (-x ^ number) & (1 << n);

Bit n will be set if x is 1, and cleared if x is 0.

**GPIO Registers**

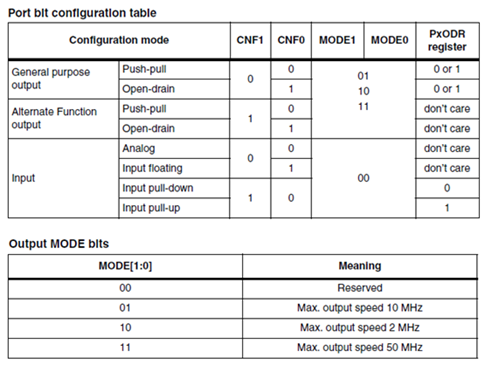
The libmaple libraries, on which STM32duino is based, provides access to registers by the syntax:

GPIOA->regs->REG

where REG can be one of the following:

**CRH and CRL**

**CRH** is used to set type/and or speed of pins 8-15 of the port.  
**CRL** is used to set type/and or speed of pins 0-7 of the port.  
Accessed as a 32 bit word, with 4 bits representing the state of each pin. Out of these 4 bits, the low 2 bits are MODE, and high 2 bits are CNF.

[](https://camo.githubusercontent.com/dae74d1fd228d62653a3235203102a8b53267f66/687474703a2f2f692e696d6775722e636f6d2f3363416e7571302e706e67)

The 4 bits for each pin can be set to:  
0b0011 (binary) or 0x3 (HEX) - Corresponds to setting pin as output, same as pinMode()  
0b1000 or 0x8 - Corresponds to setting pin as input, same as pinMode()

Say I want to set PORTA pins 0, 3 and 4 to OUTPUT and 1, 6, 7 to INPUT, and leave pins 2 and 5 in their original state. The code is:

PORTA->regs->CRL = (PORTA->regs->CRL & 0x00F00F00) | 0x88000080 |0x00033003;

//0x00F00F00 is bitmask to retain value of pins 2 and 5 in original state

//0x88000080 is bitmask to set inputs

//0x00033003 is bitmask to set outputs

**IDR - Input Data Register**

Used to read input of entire 16 pins of port at once. Accessed as a 32 bit word whose lower 16 bits represent each pin. The pins being read must be set to INPUT mode by using CRL/CRH or pinMode() before using this.

Say I want to read pins A2. The code is:

bool result = GPIOA->regs->IDR & 0x0004; //returns true if A2 is HIGH

//0x0004 is 0b0000000000000100

**ODR - Output Data Register**

Used to write output to entire 16 pins of port at once. Accessed and written as a 32 bit word whose lower 16 bits represent each pin. The pins being read must be set to OUTPUT mode by using CRL/CRH or pinMode() before using this.

Say I want to set pins A2, A12 and A13, and **reset (clear)** all other pins in the 16 pin bus. The code is:

GPIOA->regs->ODR = 0b0011000000000100; //note, binary

Now if I want to set and clear A2, A12 and A13 **without** altering other pins, the code is:

//Set A2, A12, A13 (HIGH)

GPIOA->regs->ODR |= 0b0011000000000100;

//Clear A2, A12, A13 (LOW)

GPIOA->regs->ODR &= ~(0b0011000000000100);

but notice how, if we want to touch only some pins, we have to READ, MASK and WRITE. That's why there is BRR and BSRR

**BRR - Bit Reset Register**

32 bit word. Lower 16 bits have 1's where bits are to be set to "LOW". Upper 16 bits have 1's where bits are to be set "HIGH". **0's mean ignore**

Now, to set and clear A2, A12, A13 while preserving the state of all other pins in the port, the code is:

//Set A2, A12, A13 (HIGH)

GPIOA->regs->BRR = 0b0011000000000100 << 16; //move to upper 16 bits

//Clear A2, A12, A13 (LOW)

GPIOA->regs->BRR = 0b0011000000000100;

**BSRR - Bit Set Reset Register**

BSRR is like the complement of BRR. It's also a 32 bit word. Lower 16 bits have 1's where bits are to be set to "HIGH". Upper 16 bits have 1's where bits are to be set "LOW". **0's mean ignore**

In this case, to set and clear A2, A12, A13 while preserving the state of all other pins in the port, the code is:

//Set A2, A12, A13 (HIGH)

GPIOA->regs->BSRR = 0b0011000000000100;

//Clear A2, A12, A13 (LOW)

GPIOA->regs->BSRR = 0b0011000000000100 << 16; //move to upper 16 bits

**Combination of BRR and BSRR**

Since BRR and BSRR are opposite of each other, you can use both if you don't want to do the bit shift left operation .

In this case, to set and clear A2, A12, A13 while preserving the state of all other pins in the port, the code is:

//Set A2, A12, A13 (HIGH)

GPIOA->regs->BSRR = 0b0011000000000100; //lower 16 bits

//Clear A2, A12, A13 (LOW)

GPIOA->regs->BRR = 0b0011000000000100; //lower 16 bits