The _BV Macro

In the C language one assigns and tests bits using bit operators, the assign operator, and the concept of bit masks:

```
PORTC |= 0x01;  // Set bit 0 only.
PORTC &= ~0x01; // Clear bit 0 only.
PORTC ^= 0x01;  // Toggle bit 0 only.
PORTC & 0x01;  // Test bit 0 only.
PORTC |= 0x80; // Set bit 7 only.
```

Using macros make this easier to read. The `_BV()` macro in avr-libc takes a number as the argument and converts it to the appropriate bit mask. (The BV stands for Bit Value). The _BV() macro is defined as:

```
#define _BV(x)   (1 << x)
```

`this allows:`

```
PORTC |= _BV(0);  // Set bit 0 only.
PORTC &= ~(_BV(1));  // Clear bit 1 only.
PORTC ^= _BV(7);  // Toggle bit 7 only.
```

This can be further enhanced with the defines found in the processor header files:

```
// For atmega128
       #include <avr/io.h>
       UCSR0B |= _BV(TXEN0);  // Set bit 3 in UCSR0B only.
```

Using bit operators, one can do multiple, non-contiguous bits at a time:

```
PORTC |= (_BV(0) | _BV(2) | _BV(7));  // Set bits 0,2,7
PORTC &= ~(_BV(1) | _BV(2) | _BV(6));  // Clear bits 1,2,6
PORTC ^= (_BV(5) | _BV(3));   // Toggle bits 3,5
```

The **|** symbol between each _BV macro statement means logically OR.

(_BV(0) | _BV(2) | _BV(7)); logically OR's the bits together
e.g

Name		**bit7**	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit**	**bit0**
`_BV(0)`	=	0	0	0	0	0	0	0	1
`_BV(2)`	=	0	0	0	0	0	1	0	0
`_BV(7)`	=	1	0	0	0	0	0	0	0
or'ed	**=**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**

A further example is

```
UCSRB = _BV(TXEN)|_BV(RXEN)|_BV(RXCIE); /* tx/rx enable, rx complete*/
```