Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

# AVR Freaks

## AVR Tutorials - [TUT] [C] Bit manipulation

To really understand what's going, it's best to learn C languages bit operators and about truth tables.

| bit OR
& bit AND
~ bit NOT
^ bit EXLUSIVE OR (XOR)
<< bit LEFT SHIFT
>> bit RIGHT SHIFT

These operators work on bits and not logical values. Take two 8 bit bytes, combine with any of these operators, and you will get another 8 bit byte according to the operator's function. These operators work on the individual bits inside the byte.

A truth table helps to explain each operation. In a truth table, a 1 bit stands for true, and a 0 stands for false.

The OR operation truth table:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

The AND operation truth table:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

The XOR operation truth table:

0 XOR 0 = 0
0 XOR 1 = 1

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

1 XOR 0 = 1
1 XOR 1 = 0

The NOT operator inverts the sense of the bit, so a 1 becomes a 0, and a 0 becomes a 1.

So let's say I have a byte foo that is initialized to 0:

**Code:**

```
unsigned char foo = 0;
```

To set bit 0 in foo and then store the result back into foo:

**Code:**

```
foo = foo | 0x01;
```

The OR operation is used between the variable that we want to change and a constant which is called a BIT MASK or simply the MASK. The mask is used to identify the bit that we want changed.

Remember that we write the constants in hexadecimal because it's shorter than writing it in binary. It is assumed that the reader knows how to convert back and forth between

hex and binary.

Usually, though the statement is made shorter in real programming practice to take advantage of C's compound assignment:

**Code:**

```
foo |= 0x01;
```

This is equivalent to the statement above.

To clear bit 0 in foo requires 2 bit operators:

**Code:**

```
foo = foo & ~0x01;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

This uses the AND operator and the NOT operator. Why do we use the NOT operator? Most programmers find it easier to specify a mask wherein the bit that they are interested in changing, is set. However, this kind of mask can only be used in setting a bit (using the OR operator). To clear a bit, the mask must be inverted and then ANDed with the variable in question. It is up to the reader to do the math to show why this works in clearing the desired bit.

Again, the statement is made shorter with a compound assignment:

**Code:**

```
foo &= ~0x01;
```

To see if a bit is set or clear just requires the AND operator, but with no assignment. To see if bit 7 is set in the variable foo:

**Code:**

```
if(foo & 0x80)
{
}
```

The condition will be zero if the bit is clear, and the condition will be non-zero if the bit is set. NOTE! The condition will be NON-ZERO when the bit is set. But the condition will not NECESSARILY BE ONE. It is left to the reader to calculate the value of the condition to understand why this is the case.

There is another useful tool that is not often seen, and that is when you want to flip a bit, but you don't know and you don't care what state the bit is currently in. Then you would use the XOR operator:

**Code:**

```
foo = foo ^ 0x01;
```

Or the shorter statement:

**Code:**

```
foo ^= 0x01;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

A lot of times the bit mask is built up dynamically in other statements and stored in a variable to be used in the assignment statement:

**Code:**

```
foo |= bar;
```

Sometimes, a programmer wants to specify the bit NUMBER that they want to change and not the bit MASK. The bit number always starts at 0 and increases by 1 for each bit. An 8 bit byte has bit numbers 0-7 inclusive. The way to build a bit mask with only a bit number is to LEFT SHIFT a bit by the bit number. To build a bit mask that has bit number 2 set:

**Code:**

```
(0x01 << 2)
```

To build a bit mask that has bit number 7 set:

**Code:**

```
(0x01 << 7)
```

To build a bit mask that has bit number 0 set:

**Code:**

```
(0x01 << 0)
```

Which ends up shifting the constant 0 bytes to the left, leaving it at 0x01.

**MACROS**

Because there are a number of programmers who don't seem to have a familiarity with bit flipping (because they weren't taught it at school, or they don't need to know it because of working on PCs), most programmers usually write macros for all of these operations. Also, it provides a fast way of understanding what is happening when

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

reading the code, or it provides additional functionality.

Below is a set of macros that works with ANSI C to do bit operations:

**Code:**

```
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))
#define LONGBIT(x) ((unsigned long)0x00000001 << (x))
```

To set a bit:

**Code:**

```
bit_set(foo, 0x01);
```

To set bit number 5:

**Code:**

```
bit_set(foo, BIT(5));
```

To clear bit number 6 with a bit mask:

**Code:**

```
bit_clear(foo, 0x40);
```

To flip bit number 0:

**Code:**

```
bit_flip(foo, BIT(0));
```

To check bit number 3:

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
if(bit_get(foo, BIT(3)))
{
}
```

To set or clear a bit based on bit number 4:

**Code:**

```
if(bit_get(foo, BIT(4)))
{
   bit_set(bar, BIT(0));
}
else
{
   bit_clear(bar, BIT(0));
}
```

To do it with a macro:

**Code:**

```
bit_write(bit_get(foo, BIT(4)), bar, BIT(0));
```

If you are using an unsigned long (32 bit) variable foo, and have to change a bit, use the macro LONGBIT which creates un unsigned long mask. Otherwise, using the BIT() macro, the compiler will truncate the value to 16-bits.[/list]

---

**smileymicros** - Apr 22, 2006 - 02:59 PM
**Post subject:** RE: [TUT] [C] Bit manipulation (AKA "Programming 101&am

This was also the basis for excellent an article in Circuit Cellar http://www.dtweed.com/circuitcellar/caj00180.htm#3120
Unfortunately I think you have to pay to get a copy, but it is really worth it.

Smiley

---

**cell** - May 04, 2006 - 10:58 PM
**Post subject:** RE: [TUT] [C] Bit manipulation (AKA "Programming 101&am

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

this is also touched on in "AVR035: Efficient C Coding for AVR" http://www.atmel.com/dyn/resources/prod_documents/doc1497.pdf

I based a few macros off of that app note, and created avr035.h:

**Code:**

```
#ifndef _AVR035_H_
#define _AVR035_H_

// from AVR035: Efficient C Coding for AVR

#define SETBIT(ADDRESS,BIT) (ADDRESS |= (1<<BIT))
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT))
#define FLIPBIT(ADDRESS,BIT) (ADDRESS ^= (1<<BIT))
#define CHECKBIT(ADDRESS,BIT) (ADDRESS & (1<<BIT))

#define SETBITMASK(x,y) (x |= (y))
#define CLEARBITMASK(x,y) (x &= (~y))
#define FLIPBITMASK(x,y) (x ^= (y))
#define CHECKBITMASK(x,y) (x & (y))

#endif
```

---

**abcminiuser** - May 05, 2006 - 06:30 AM
**Post subject:** RE: [TUT] [C] Bit manipulation (AKA "Programming 101&am

A recent thread had a very nice solution which extends on the basic bit-manipulation macros. IIRC it went something along the lines of:

Defines:

**Code:**

```
#ifndef _AVR035_H_
#define _AVR035_H_

// from AVR035: Efficient C Coding for AVR

#define SETBIT(ADDRESS,BIT) (ADDRESS |= (1<<BIT))
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT))
#define FLIPBIT(ADDRESS,BIT) (ADDRESS ^= (1<<BIT))
#define CHECKBIT(ADDRESS,BIT) (ADDRESS & (1<<BIT))

#define SETBITMASK(x,y) (x |= (y))
#define CLEARBITMASK(x,y) (x &= (~y))
#define FLIPBITMASK(x,y) (x ^= (y))
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
#define CHECKBITMASK(x,y) (x & (y))

#define VARFROMCOMB(x, y) x
#define BITFROMCOMB(x, y) y

#define C_SETBIT(comb) SETBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_CLEARBIT(comb) CLEARBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_FLIPBIT(comb) FLIPBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_CHECKBIT(comb) CHECKBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))

#endif
```

Use:

**Code:**

```
#define Status_LED  PORTA, 3

C_SETBIT(Status_LED);
C_CLEARBIT(Status_LED);
```

- Dean

---

**BenG** - Mar 19, 2007 - 03:37 PM
**Post subject:** RE: [TUT] [C] Bit manipulation (AKA "Programming 101&am

As an additional item to check if a bit is clear:

**Code:**

```
if(~(foo) & 0x80)
{
}
```

---

**donblake** - Mar 20, 2007 - 02:49 AM
**Post subject:** Re: RE: [TUT] [C] Bit manipulation (AKA "Programming 10

BenG wrote:

As an additional item to check if a bit is clear:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
if(~(foo) & 0x80)
{
}
```

My 1st choice would be for the following which, IMHO, is easier to "read":

**Code:**

```
if ( ( foo & 0x80 ) == 0 )
{
  ...
}
```

should result in the same compiler generated code.

Don

---

**Bingo600** - Mar 24, 2007 - 01:37 PM
**Post subject:** RE: Re: RE: [TUT] [C] Bit manipulation (AKA "Programmin

With regards to the above i'd use

```
if(!CHECKBITMASK(foo,0x80))
```

or

```
if(!CHECKBIT(foo,7))
```

But i agree Don's code is more readable

/Bingo

---

**danni** - May 11, 2007 - 11:01 AM
**Post subject:** RE: Re: RE: [TUT] [C] Bit manipulation (AKA "Programmin

Another approach:

I like it to access bit variables like any other variables and then I can write:

```
if(i == 1)
if(i == 0)
i = 0;
i = 1;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

which looks easy readable for me.

This can easy be done by casting a portbit as a member of a bit field.

On the attachment there is the definition of the macro SBIT.

Following an example code:

**Code:**

```
#include <io.h>
#include "sbit.h"


#define KEY0          SBIT( PINB, 0 )
#define KEY0_PULLUP     SBIT( PORTB, 0 )

#define LED0          SBIT( PORTB, 1 )
#define LED0_DDR        SBIT( DDRB, 1 )


int main( void )
{
  LED0_DDR = 1;       // output
  KEY0_PULLUP = 1;      // pull up on

  for(;;){

    if( KEY0 == 0 )     // if key pressed (low)
      LED0 = 0;       // LED on (low)
    else
      LED0 = 1;       // LED off (high)
  }
}
```

Naturally this macro can also be used for internal flag variables, not only for IO registers.

Peter

---

**Adam_Y** - Jun 28, 2007 - 02:50 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
TCCR1B |= (1 << CS10);
```

You might want to explain what this means. I didn't realize you could actually left shift a bit to a specific place using this actual name of the bit. It drove me crazy because the timer tutorial was directing me here and your tutorial makes no mention of this.

---

**bloody-orc** - Jun 28, 2007 - 03:18 PM
**Post subject:**

That means, that nr 1 is shifted left as many times as it is needed to reach bit named CS10. Where does the compiler know that CS10 is that? well you give him the AVR's name and it's smart enough to know such things thanks to some smart programmers on

the GCC side

---

**clawson** - Jun 28, 2007 - 03:44 PM
**Post subject:**

I think this tutorial is trying to be as generic as possible. Not all the AVR C compilers have all the bit names defined in the header files for each AVR part so

**Code:**

```
TCCR1B |= (1 << CS10);
```

won't necessarily work on all compilers.

There's no "magic" to it anyway, a typical part definition file that does include the bit positions simply has something like (for mega16):

**Code:**

```
#define CS10    0
#define CS11    1
#define CS12    2
#define WGM12   3
#define WGM13   4
#define ICES1   6
#define ICNC1   7
```

For those C compilers that don't have the bit definitions in their .h collection the AVR

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Studio file \Program Files\Atmel\AVR Tools\AvrStudio4\xmlconvert.exe will probably prove useful to generate .h files from Atmel's own XML part definition files:

**Code:**

```
C:\Program Files\Atmel\AVR Tools\AvrStudio4>xmlconvert
xmlconvert: No source file specified

Usage: xmlconvert [-f output-format] [-o outdir] [-1nbclV] infile ...
      Output formats: a[vrasm] | g[cc] | i[ar] | c[c] (generic c)
Options:
      -1  = Don't generate AVRASM2 #pragma's
      -n  = Don't warn about bad names
      -b  = use DISPLAY_BITS attribute to limit bit definitions
      -c  = Add some definitions for compatibility with old files
      -l  = Produce linker file (IAR only)
      -q  = Allow linked register quadruple (32-bit)
      -V  = print xmlconvert version number
```

Cliff

---

**AlIN** - Jun 29, 2007 - 02:52 AM
**Post subject:**

Thanks all; I'm going to use most of the above for next yr's class. And another lesson you just taught. Show Dons, BenG, and then Bingos to show writing style and how they accomplish the same thing.
I couldn't write (copy) the curriculum without this site.
Thanks for the help and making the world a better place,
John

---

**Adam_Y** - Jun 29, 2007 - 05:32 PM
**Post subject:**

clawson wrote:

I think this tutorial is trying to be as generic as possible. Not all the AVR C compilers have all the bit names defined in the header files for each AVR part so

**Code:**

```
TCCR1B |= (1 << CS10);
```

won't necessarily work on all compilers.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

There's no "magic" to it anyway, a typical part definition file that does include the bit positions simply has something like (for mega16):

**Code:**

```
#define CS10    0
#define CS11    1
#define CS12    2
#define WGM12   3
#define WGM13   4
#define ICES1   6
#define ICNC1   7
```

Where are the definitions in AVR GCC? I am guessing in the IO header file or each header has every single part defined for only the related registers.

---

**clawson** - Jun 29, 2007 - 05:34 PM
**Post subject:**

Yup in GCC all your programs include <avr/io.h> and your Makefile will define a MCU=. As a consequence of these two things it will lead to io.h choosing to pull in one of the io????.h files in \winavr\avr\include\avr\

Cliff

---

**Adam_Y** - Jun 29, 2007 - 05:53 PM
**Post subject:**

clawson wrote:

Yup in GCC all your programs include <avr/io.h> and your Makefile will define a MCU=. As a consequence of these two things it will lead to io.h choosing to pull in one of the io????.h files in \winavr\avr\include\avr\

Cliff

I am guessing Avr Studio is handling the making the Makefile when I compile my program. Oddly enough I don't even think I defined the correct MCU when I first created the project. I defined it as a Mega 168 and only figured out that it was a Mega 48 when my Dragon complained that the parts didn't match up. Is this because the parts are in the same data sheet?

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Jun 29, 2007 - 06:03 PM
**Post subject:**

There's several MAJOR difference between 48 and 168. For example one uses RJMPs for its interrupt vectors and one uses JMPs (because the entire memory is no longer reachable with an RJMP). Far more worryingly the 168 has 1K of SRAM while the 48 has 512 bytes. So the RAMEND used to initialise the stack pointer will be different between the two. I'm therefore astonished that you found that code built for a 168 worked in a 48 !?!

---

**Adam_Y** - Jun 29, 2007 - 06:51 PM
**Post subject:**

clawson wrote:

There's several MAJOR difference between 48 and 168. For example one uses RJMPs for its interrupt vectors and one uses JMPs (because the entire memory is no longer reachable with an RJMP). Far more worryingly the 168 has 1K of SRAM while the 48 has 512 bytes. So the RAMEND used to initialise the stack pointer will be different between the two. I'm therefore astonished that you found that code built for a 168 worked in a 48 !?!

Well the code I wrote was only five lines long and didn't use interrupts. Though I might want to make sure that the I actually did program for the 168. I don't have access to the computer I wrote the code in at the moment.

---

**fleemy** - Jul 03, 2007 - 08:03 PM
**Post subject:**

Another thing I've just learned, you can toggle the PORTxn pins by writhing to the PINxn register.

So instead of toggling the bits with the XOR operator, you can write to the PINxn address.

Do not know if the compiler optimization already does that?

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Jul 03, 2007 - 08:42 PM
**Post subject:**

But that only works on the recent AVRs, not all of them.

---

**robinsm** - Nov 06, 2007 - 08:19 PM
**Post subject:**

**Quote:**

BenG wrote:
As an additional item to check if a bit is clear:

Code:
```
if(~(foo) & 0x80)
{
}
```
My 1st choice would be for the following which, IMHO, is easier to "read":
Code:
```
if ( ( foo & 0x80 ) == 0 )
{
...
}
```
should result in the same compiler generated code.

Don

and another way...

**Code:**

```
if (!( foo & 0x80 ))
{
  ...
}
```

---

**jbrain** - Feb 05, 2008 - 04:17 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

I know this is compiler generic, but I do have a related question:

I've been using (1<<PIN_NAME) for some time in my code, but I am working on a project now where the developer is using _BV(PIN_NAME) all over. Is there a best practice on which to use?

Jim

---

**Taco_Bell** - Apr 13, 2008 - 08:52 PM
**Post subject:**

Will this blink an LED?

**Code:**

```
#inculde <avr/io.h>
#inculde <util/delay.h>

DDRB = 0b11111111;

while(1)
{
PORTB |= (0x01 << 7);
_delay_ms(250);
PORTB &= (0x00 << 7);
_delay_ms(250);
}
```

---

**Koshchi** - Apr 13, 2008 - 10:19 PM
**Post subject:**

**Quote:**

Will this blink an LED?

Well, yes, but it isn't doing what you really want.

**Code:**

```
PORTB |= (0x01 << 7);
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

This sets bit 7 (and **only** bit 7) to 1. This is because it translates into:

**Code:**

```
PORTB = PORTB | 0b10000000;
```

So bit 7 is changed, and the rest of the bits remain what they were.

**Code:**

```
PORTB &= (0x00 << 7);
```

This sets **all** bits to 0, **not just bit 7**.
This is because this translates into:

**Code:**

```
PORTB = PORTB & 0b00000000;
```

since 0 shifted by any amount is still going to be 0, and 0 ANDed with anything is 0.
You want:

**Code:**

```
PORTB &= ~(0x01 << 7);
```

---

**Taco_Bell** - Apr 14, 2008 - 07:04 PM
**Post subject:**

Thank You Koshchi,
That is exactly what I wanted. I was going to ask this next:

**Quote:**

This sets all bits to 0, not just bit 7.
This is because this translates into:

**Code:**

```
PORTB = PORTB & 0b00000000;
```

since 0 shifted by any amount is still going to be 0, and 0 ANDed with anything is 0.
You want:
Code:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
PORTB &= ~(0x01 << 7);
```

Thanks again,

---

**mikey_G** - Apr 30, 2008 - 02:42 PM
**Post subject:**

Taco_Bell wrote:

Will this blink an LED?

**Code:**

```
#inculde <avr/io.h>
#inculde <util/delay.h>


DDRB = 0b11111111;

while(1)
{
PORTB |= (0x01 << 7);
_delay_ms(250);
PORTB &= (0x00 << 7);
_delay_ms(250);
}
```

It would blink a led, but it would be a inefficient way to do so.

This is how I would solve it:
**Code:**

```
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
  DDRB = 0xFF;

  while(1) {
    PORTB ^= (1 << 7);
    _delay_ms(250);
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
  }
}
```

Here I use the XOR operator which is ideal for flipping a bit. Using the 1 XOR as opposed to 1 OR and 1 AND also reduced the code size from 443 bytes (your example) to 398 bytes (my example)

The logic behind it is that on the first run the 8th bit of PORTB, which is 0, is XOR'ed with a 1 which results in a 1.
**0 XOR 1 = 1**
On the next loop iteration the 8th bit of PORTB, which is now 1, will be XOR'ed with a 1 which will result in a 0.
**1 XOR 1 = 0**

---

**dandumit** - May 30, 2008 - 08:52 AM
**Post subject:**

Hello ,
Please excuse my dumb question :
how do I do octet/byte rotation in avrgcc ?
I mean how do I wrote in C the equivalent of asm rol cmd ?
Kind regards,
Daniel

---

**clawson** - May 30, 2008 - 11:22 AM
**Post subject:**

I guess the first question is WHY you'd want to. But there's no easy way, you'd need to use something like

**Code:**

```
topbit = (bytevar & 0x80) ? 0 : 1;
bytevar <<= 1;
bytevar |= topbit;
```

---

**dandumit** - May 30, 2008 - 02:50 PM
**Post subject:**

First of all thanks for answer.
I will answer why I need to do this : I writing an dmx receiver and I need to read the

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

address of that receiver from a dipswitch pack. Because I was restricted to single side pcb I have inverted the traces to get the routing done.

I will abuse of your kindness and I would like to ask if it is possible to wrote an inline asm macro like :
http://www.nongnu.org/avr-libc/user-man ... e_asm.html

something like this would work ?
asm volatile("rol %0" : "=r" (value) : "0" (value));

Thanks
Daniel

---

**tigrezno** - Jun 15, 2008 - 09:41 PM
**Post subject:**

those are my definitions, i find them more user-friendly, tell me what you think.

**Code:**

```
#define LOW 0
#define HIGH 1

#define INPUT(port,pin) DDR ## port &= ~(1<<pin)
#define OUTPUT(port,pin) DDR ## port |= (1<<pin)
#define CLEAR(port,pin) PORT ## port &= ~(1<<pin)
#define SET(port,pin) PORT ## port |= (1<<pin)
#define TOGGLE(port,pin) PORT ## port ^= (1<<pin)
#define READ(port,pin) (PIN ## port & (1<<pin))
```

usage:

**Code:**

```
OUTPUT(D, 3); // port D, pin 3 as output
SET(D, 3); // set port D pin 3 to HIGH
CLEAR(D, 3); // set it to LOW

INPUT(B, 5);
if (READ(B, 5) == HIGH)
...
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

feel free to add them to the tutorial
[updated]

---

**MotoDog** - Jun 16, 2008 - 08:59 PM
**Post subject:**

Tigrezno,
I am been C ing with AVR for a few months. I am a novice. I got an A/D working with interrupts and putting the data to RS232 to RealTerm on a PC working!
Big accomplishment for me, lots of hours.

I have tried a few of the examples on this topic posted here and ran into problems. Mostly with the fact that PORTs need indirection methods to do bit fiddling? I think?
I just tried your "toggle" on a mega128 PORTA Bit 1 and it compiled and the hardware worked! Thanks, I wanted PORT bit setting and flipping, not just variables. This seems to be the best method discussed here yet? I will try the rest of thems soon.
Where is the #Define documentation for AVR C, or is it per standard C practice and I find it in a C book?
In the DOC directory in my AVR C installation there are many many HTML files. They seem to just compare features to standard C, with no examples? Not much help for a beginner?
Am I missing a manual for this AVR C somewhere?
I didn't know you could use # for parameter subsitution.
Thanks, I put your defines in my first AVR learning C project!
Mike

---

**clawson** - Jun 16, 2008 - 10:15 PM
**Post subject:**

#define is just one of the C Pre Processor functions. The facilities available are standard to most c compilers so this manual for the GCC variant should be as good as any:

http://gcc.gnu.org/onlinedocs/cpp/

---

**MotoDog** - Jun 16, 2008 - 11:15 PM
**Post subject:**

Thanks!

I am looking at it now. Lots there.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

The use of the # # to pass those port parameters still confuses me? I'll keep thinkin on it!

---

**Koshchi** - Jun 17, 2008 - 12:54 AM
**Post subject:**

## is just the concatenation operator in macros. #define is just a text replacement tool, not a C programming construct. So something like:

**Quote:**

READ(B, 5)

simply gets replaced by:

**Quote:**

PORT**B** & (1<<**5**)

---

**MotoDog** - Jun 17, 2008 - 02:00 AM
**Post subject:**

Got it!

Thanks, this if from the manual Clawson sent the link too! "I wooda never known!"

"This is called token pasting or token concatenation. The `##' preprocessing operator performs token pasting. When a macro is expanded, the two tokens on either side of each `##' operator are combined into a single token, which then replaces the `##' and the two original tokens in the macro expansion."

---

**Lajon** - Jun 17, 2008 - 11:42 AM
**Post subject:**

**Code:**

```
#define READ(port,pin) PORT ## port & (1<<pin)
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

tigrezno, did you actually use this? To read you have to use the PIN register.
Additionally you can not compare the & result with 1 as you do here:

**Code:**

```
if (READ(B, 5) == HIGH)
```

This would work if you need to compare:

**Code:**

```
#define READ(port,pin) ((PIN ## port & (1<<pin)) != 0)
```

or just

**Code:**

```
#define READ(port,pin) ((PIN ## port & (1<<pin))
```

if you are ok with the simpler test condition:

**Code:**

```
if (READ(B,5)) {
```

/Lars

---

**tigrezno** - Jun 17, 2008 - 04:54 PM
**Post subject:**

ouch that was a typo!

You're correct, it should be PIN and not PORT, sorry, i'll change it.

The comparation issue is true too, me bad, very very bad.

I wrote them without using the "input" defines, sorry.
I hope you found the other methods usefull.

---

**dandumit** - Sep 02, 2008 - 11:25 AM
**Post subject:**

I'm a beginner in embedded development and for small applications I really enjoy a simple mode to access a port pin directly ( like PORTA.1 or PORTA_1).
I have found this example a while ago and I would like to hear your comments on it :

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

```
#include <avr/io.h>

// Define the bits in the port
typedef struct
{
unsigned char bit0 : 1,
bit1 : 1,
bit2 : 1,
bit3 : 1,
bit4 : 1,
bit5 : 1,
bit6 : 1,
bit7 : 1;
} bit_field;

// Define macro to get the value of each bit
#define GET_BIT(port) (*(volatile bit_field *) (_SFR_ADDR(port)))

// Define functions for each bit of the I/O ports in the program
#define SIG GET_BIT(PINB).bit0
#define LED GET_BIT(PORTD).bit0

int main (void)
{
for (;;)
{
if (SIG) LED = 1;
else LED = 0;
}
}
```

I would really like to understood how this code is working ... especially part with setting a bit.

---

**clawson** - Sep 02, 2008 - 11:58 AM
**Post subject:**

Well break it down as the pre-processor will be doing for you. Let's say you use:

**Code:**

```
LED = 1;
```

firstly that becomes:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
GET_BIT(PORTD).bit0;
```

which in turn is:

**Code:**

```
(*(volatile bit_field *) (_SFR_ADDR(PORTD))).bit0;
```

and from the GCC header files the _SFR_ADDR() and PORTD macros further break this down to be:

**Code:**

```
 (*(volatile bit_field *) (((uint16_t) &((*(volatile uint8_t *)((0x12) + 0x20))))))).bit0 = 1;
```

The 0x12 in there will vary depending on your AVR - the above is correct for mega16.

So ultimately it's casting a bitfield structure onto memory address 0x32 and then setting 1 into "bit0" of that struct.

When you see:

**Code:**

```
struct {
 :1
 :1
}
```

it's effectively assigning one bit for each element. So this is just allowing individual bit access to the 8 bits being held in memory location 0x32.

Cliff

---

**MIchael_J** - Sep 12, 2008 - 12:30 AM
**Post subject:**

Thank you for the tutorial. I found it very useful coming from beeing used to only sbi(), cbi() macros.

Somewhere else on the net i found these two macros beeing called obsolete... Any idea why is it so? Is it indeed more justified to acces the bits as shown in the tutorial?

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

This article is helping a lot in understanding many other articles around here.
Thanks once more.

Take care.

---

**clawson** - Sep 12, 2008 - 12:10 PM
**Post subject:**

**Quote:**

Somewhere else on the net i found these two macros beeing called obsolete... Any idea why is it so? Is it indeed more justified to acces the bits as shown in the tutorial?

There are three ways to change a single bit using the AVR. If the register in question is in IO space locations 0x00..0x1F then SBI/CBI can be used to set/clear individual bits. Those opcodes occupy 16 bits and execute in 1 cycle. If the register is between 0x20..0x3F then it is no longer reachable using SBI/CBI but it is using IN/OUT but in this case to set a bit it wil require IN/OR/OUT and to clear a bit it will require IN/AND/OUT which is 3 16 bit opcodes and takes 3 cycles. If the register is beyond IO address 0x3F (that is beyond RAM address 0x5F) then the only access is LDS/STS and to set it wil be LDS/OR/STS and to clear it will be LDS/AND/STS. In this case it's five 16 bit opcodes (LDS and STS are 2 each) and takes 5 cycles to execute.

Now when the early versions of the GCC compiler appeared it did not have an optimisation to spot when LDS/OR/STS could be reduced to IN/OR/OUT or even SBI so a macro sbi() was defined to use inline assembler so the programer could force SBI to be used even though the compiler didn't realise it.

In later compiler versions the code generator was improved so that it would always generate the most optimal code (as long as -O0 was not used!) so the need for the programmer to take control and force when SBI should be used was no longer required.

In the latest GCC (and the other AVR C compilers) the totally portable, totally standard PORTB |= (1<<PB5) will always generate just an SBI PORTB,5 as long as PORTB is between 0x00 and 0x1F and the optimiser is enabled.

The joy of using that standard construct is that you don't need the code to rely on some extra .h file that provides sbi()/cbi() so it makes it more portable between copilers and other architectures.

If using GCC then (for the time being) there is still <compat/deprecated.h> and this now includes:

**Code:**

```
#define sbi(port, bit) (port) |= (1 << (bit))
#define cbi(port, bit) (port) &= ~(1 << (bit))
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

so, as you can see, this is just supporting "old" code and turning the macro into the "standard" construct that's been explained above anyway.

As the file is deprecated there's a chance it may be withdrawn from future distributions so it's unwise to write code these days to use it because next time you upgrade compiler you may find that the file has gone.

Cliff

---

**MIchael_J** - Sep 13, 2008 - 02:02 PM
**Post subject:**

Thanks a lot.
Thins makes it clear. You'v definitely convinced me to get used to the current stnadard notation.

---

**khos007** - Sep 15, 2008 - 10:04 AM
**Post subject:**

I just have small query regarding syntax:

Is this syntax valid:
PORTC = (0<<PC0); eg. driving the PC0 pin low.
or are you only able to do bit shift operations with "1"
PORTC = (1<<PC0)

I have tried this out on ATMEGA8 and it seems to work but many ppl have told me that that it is not wise to use the bit shift operation with 0 as it can lead to bugs. Could someone please clarify this for me?

Thanks in Advance.

---

**abcminiuser** - Sep 15, 2008 - 11:05 AM
**Post subject:**

Zero shifted left or right is always zero, so your code there just clears the entire PORTC register by setting it to zero.

You should never need to shift 0 at all, other than to indicate clearly that a bit is not set.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

- Dean

---

**clawson** - Sep 15, 2008 - 11:41 AM
**Post subject:**

**Quote:**

Could someone please clarify this for me?

You did read the whole of this thread didn't you?

Hopefully it should have been obvious to you that to achieve what you think:

**Code:**

```
PORTC = (0<<PC0);
```

might do (assuming the intention is only to set bit PC0 to zero) you would actually use:

**Code:**

```
PORTC &= ~(1<<PC0);
```

Remeber you can only set bits with | and you can only clear them with &. Though you can obviously set/clear all 8 bits in a register in one go with a simple =, but then you cannot affect just single bits.

Cliff

---

**kangtoojee** - Oct 07, 2008 - 05:45 PM
**Post subject:**

Can somebody help me out pls, I just cant access the register definitions because it says undeclared after compiling. Im using atmega324p chip. Thanks.

---

**clawson** - Oct 07, 2008 - 05:54 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

Can somebody help me out pls, I just cant access the register definitions because it says undeclared after compiling. Im using atmega324p chip. Thanks

Any particular language or variant thereof ?

---

**kaneda** - Nov 04, 2008 - 03:26 PM
**Post subject:**

I'm working on a project with an external 16-bits adc using SPI. The first 5 bits are for settling of the adc and can be discarded, I am however receiving them.
I'm reading the following bytes in 3 chars (where x is either 1 or 0):
0-7
msb|0|0|0|0|0|x|x|x|
8-15
|x|x|x|x|x|x|x|x|
& 16-23
|x|x|x|x|x|0|0|0|lsb

I want to put 0-7 in a long and shift it 5+16 to the left, put 8-15 in a long and shift it 5+8 to the left and add it to 0-7, put 16-23 in a long and shift it 5 to the left and also add it. Then convert it to a double.

so:
|0|0|0|0|0|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|0|0|0|
<<5
|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|0|0|0|0|0|0|0|
typecast
|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|
how can I shift all bits to the left in a long?

---

**clawson** - Nov 04, 2008 - 03:41 PM
**Post subject:**

**Code:**

```
unsigned int result = (byte1 << 13) | (byte2 << 5) ( (byte3 >> 3);
```

at a rough guess.

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Koshchi** - Nov 04, 2008 - 11:37 PM
**Post subject:**

I would use a union.

---

**jwcolby** - Nov 10, 2008 - 01:33 PM
**Post subject:**

When I was a boy... I learned to fix "mainframes" for the Navy. Understand that was 35 years ago so my memory of the details is questionable at best but I seem to remember that a shift took X clocks per shift position. For example to shift one position left took one clock, 2 positions took 2 cycles etc.

Is that true in the Atmega processors, or does it shift any number of positions in a single clock?

---

**clawson** - Nov 10, 2008 - 01:55 PM
**Post subject:**

The ROL/ROR/LSL/LSR instructions are all shown as being 1 cycle but to shift 5 places might take 5 of them and cost 5 cycles. Though an intelligent optimiser will spot that a shift of 4 can probably be achieved with a SWAP, an AND mask and one shift for the cost of 3 cycles

Cliff

---

**jwcolby** - Nov 10, 2008 - 03:12 PM
**Post subject:**

I haven't done bit twiddling in a long time but I thought that instead of shifting a one X positions, you just used an XOR to toggle a specific bit.

---

**Koshchi** - Nov 10, 2008 - 03:49 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

That depends on what register and what AVR you are talking about. For the newer AVRs and port registers you simply write a 1 to the proper bit in the PINx register. But I think you are a bit confused. No one uses shifting to toggle a bit. You use the shifting to create the proper bit mask that will be used for toggling (or setting or clearing) a bit. In that case an expression like:

**Code:**

```
1<<PB4
```

takes no clock cycles at all since the shifting is done by the compiler, not the AVR.

---

**jwcolby** - Nov 10, 2008 - 04:30 PM
**Post subject:**

Koshchi wrote:

In that case an expression like:

**Code:**

```
1<<PB4
```

takes no clock cycles at all since the shifting is done by the compiler, not the AVR.

Oh, shift on then.

---

**damian_** - Dec 22, 2008 - 11:30 AM
**Post subject:**

Hello
I feel stupid, because I read this and some more forums several times about bit set, clear, compare, etc. But, I still don't understand how to do sometihngs.

I'm working with a atmega32 in WinAVR.

For example:

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
PORTD = (PORTD ^ 0b00000001);
```

This code flipp bit0 of PORTD

**Code:**

```
PORTD = (PORTD & 0b11111110);
```

This code switch off PIND.0

**Code:**

```
PORTD = (PORTD | 0b00000001);
```

And this shoudl switch on PIND.0

This is an example to compare.
I use PIND.0 as input, and when PIND.0 is "H" the program show a "0" in the LCD, because I do salida+1, and salida-1, so salida keep "0".
But when I put ground in PIND.0 (externaly), the program shoudl increase de value of salida, but "0" continues in the LCD screen, why?.

**Code:**

```
#include <main.h>//main program


int salida =0;
char y=0;


//MAIN PROGRAM
int main (void)
{

  //Configure Pins / Ports as input- (0) or output (1)
    // Port D (Interupts)
    DDRD  = 0b11111110; //La entrada es INT0 Y DATAOUT
    //activate Pull Ups (1)
    PORTD = 0b11111111;

    // Port B (LED's)
    DDRB  = 0b00001100;
    PORTB = 0b11110011;

    lcd_clrscr();
    lcd_init(LCD_DISP_ON);
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
    lcd_gotoxy(0,0);
    lcd_puts("NUMBER");


  while(1)//bucle principal, pone en pantalla y espera interrupciones
  {


    salida++;
    if(PIND0 & 0b11111111)//I KNOW THIS LINE IS WRONG
    {
    salida--;
    }

    lcd_gotoxy(0,1);
    itoa (salida,&y,10);
    lcd_puts(&y);//I put salida as an string in the LCD
    // PIN3 PORTB set -> LED off
    PORTB |= (1<<PINB2);//to see program its running

  delay(2000);//to decrease the speed

  }
}
```

Well, I want to do:
Compare de value of the input in PIND.0, and make something to know when is 0 or 1.

Thanks so much

---

**clawson** - Dec 22, 2008 - 11:51 AM
**Post subject:**

**Code:**

```
    if(PIND0 & 0b11111111)//I KNOW THIS LINE IS WRONG
```

try:

**Code:**

```
if (PIND & (1<<PD0))
```

Try to get away from 0b???????? - that's the whole point of bit shifts - you don't need to know or care what the other 7 bits are doing when you access just a single bit. So it's

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
TARGET ^= (1<<bit_pos); // flip the bit
TARGET |= (1<<bit_pos); // set the bit
TARGET &= ~(1<<bit_pos); // clear the bit
if (TARGET & (1<<bit_pos)) // test if the bit is set
```

---

**damian_** - Dec 22, 2008 - 01:16 PM
**Post subject:**

Thanks very much, I writte this and works ok:

**Code:**

```
if (PIND & (1<<PD0))
```

My last question about this is:

**Code:**

```
if (PIND & ~(1<<PD0))
```

This code test if bit is not set?

Thanks very much.

---

**JohanEkdahl** - Dec 22, 2008 - 01:54 PM
**Post subject:**

**Quote:**

Code:
```
if (PIND & ~(1<<PD0))
```

This code test if bit is not set?

No, it tests if any other bit is set. Take it step by step:

1:

**Code:**

```
1<<PD0
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

make a bitmask 00000001.

2:

**Code:**

```
~(1<<PD0)
```

negates every bit in that mask so you get 11111110

3:
A bitwise and of that and PIND thus will produce a 1 in any position where the bitmask has a 1 and PIND has a 1. Or in other words, it will produce a non-zero value if any of bits 7 of PIND to 1 is non-zero.

What you want is

**Code:**

```
if ( !(PIND & (1<<PD0)))
```

Work through that, in a similar manner as I did above, step by step, to see why this is diferent and will do what you ask for. (Please note that the logical NOT operator (!) is used.)

An alternative way would be to

**Code:**

```
if (~(PIND) & (1<<PD0))
```

as BenG has pointed out in the first page of this thread.

---

**damian_** - Dec 22, 2008 - 02:52 PM
**Post subject:**

Ok. Now I think I've got it.
Thanks very much

---

**Maximilian** - Jan 08, 2009 - 02:17 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

At this time i'm working on a little project and i just can't get my head around the following

Hardware: hef4052 multiplexer with a0 and a1 tied to PC0 and PC1 of Atmega168

I'm trying to find a nice way to define the four (0/1/2/3) states of those 2 bits and what i can come up with is:

```
# define Sensor0 PORTC &= ~((1<<PORTC0) & (1<<PORTC1))
# define Sensor1 PORTC &= ~((1<<PORTC0)
# define Sensor2 PORTC |= (1<<PORTC0)
# define Sensor3 PORTC |= (1<<PORTC0) | (1<<PORTC1)
```

Is it possible to combine an & and an | operation in a define?, and what is the best way to do it?

---

**clawson** - Jan 08, 2009 - 03:36 PM
**Post subject:**

If you want to both clear some bits and then set some bits you need to do TWO read/modify/write operations. One to clear the bits and then another to set any bits that need setting. Something like

**Code:**

```
# define Sensor0 PORTC &= 0xFC
# define Sensor1 PORTC &= 0xFC; PORTC |= (1<<PORTC0)
# define Sensor2 PORTC &= 0xFC; PORTC |= (1<<PORTC1)
# define Sensor3 PORTC &= 0xFC; PORTC |= ((1<<PORTC0) | (1<<PORTC1))
```

---

**ArnoldB** - Jan 08, 2009 - 04:04 PM
**Post subject:**

**Code:**

```
PORTC &= ~((1<<PORTC0) | (1<<PORTC1));      // 00
PORTC = (PORTC & ~(1<<PORTC1)) | (1<<PORTC0); // 01
PORTC = (PORTC & ~(1<<PORTC0)) | (1<<PORTC1); // 10
PORTC |= (1<<PORTC0) | (1<<PORTC1);         // 11
```

---

**Maximilian** - Jan 08, 2009 - 04:32 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Ok, thats what i thought but didn't know how to do

Will the compiler produce the same code from both your examples?

Thanks!!

---

**clawson** - Jan 08, 2009 - 04:37 PM
**Post subject:**

Unlikely - I think Arnold's may be more efficient though personally I think mine maybe makes it a little more obvious what's going on at the cost of a bit of inefficiency (I'm a great fan of "obvious" code for the person who comes back to read this in 3 months)

---

**ArnoldB** - Jan 08, 2009 - 05:29 PM
**Post subject:**

clawson wrote:

I think mine maybe makes it a little more obvious what's going on

Hey, I even added comments. What more can you ask for?

---

**Maximilian** - Jan 08, 2009 - 06:05 PM
**Post subject:**

ArnoldB wrote:

Hey, I even added comments. What more can you ask for?

No fighting please, LOL

As speed is needed i will go with ArnoldB his version.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Points go to the both of you,

---

**damian_** - Jan 09, 2009 - 07:04 PM
**Post subject:**

Hello again
First of all thanks very much, because of all of you I can continue working in my project.
I think I understand your explanations. But I'm still confuse with one point.
Here it is my code

**Code:**

```
  //PORT INITIALITATION.(1)OUTPUT,(0) INPUT
  DDRB  = 0b00001100;//LEDS AS OUTPUT
  PORTB = 0b11111111;//PULL-UP ACTIVE

  DDRD  = 0b11101011;//DDC112 COMUNICATION
  PORTD = 0b11111111;//PULL-UP ACTIVE

//  PORTD &= ~(1 << 6);//DCLK "LOW"->INITIALITATION
  dclksemiperiod=0;


  PORTB ^= (1 << 3); //Toggle the LED PB3
  PORTD ^= (1 << 6); //Toggle PD6

  if((PIND & (1<<PD6)) && (dclksemiperiod == 0) && (PIND & (1<<PD4)))
```

The problem is that when I uncomment the line

**Code:**

```
PORTD &= ~(1 << 6);
```

The condition in the if its not true, and I don't not why.
I suposse that, at the beggining PD6=1, after

**Code:**

```
PORTD ^= (1 << 6);
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

PD6 shoud be 0, but its not. Why?
(I know the problem is PD6, because I allready checked all the other things and are ok, of course PD4 its allways on by hardware)
THANKS

---

**devilsandy** - Jul 08, 2009 - 04:12 AM
**Post subject:**

tigrezno wrote:

those are my definitions, i find them more user-friendly, tell me what you think.

**Code:**
```
#define LOW 0
#define HIGH 1

#define INPUT(port,pin) DDR ## port &= ~(1<<pin)
#define OUTPUT(port,pin) DDR ## port |= (1<<pin)
#define CLEAR(port,pin) PORT ## port &= ~(1<<pin)
#define SET(port,pin) PORT ## port |= (1<<pin)
#define TOGGLE(port,pin) PORT ## port ^= (1<<pin)
#define READ(port,pin) (PIN ## port & (1<<pin))
```

usage:

**Code:**
```
OUTPUT(D, 3); // port D, pin 3 as output
SET(D, 3); // set port D pin 3 to HIGH
CLEAR(D, 3); // set it to LOW

INPUT(B, 5);
if (READ(B, 5) == HIGH)
...
```

feel free to add them to the tutorial
[updated]

Hello All, I have extended this code one more step. I wanted to keep all the pin definition in one place or one file.

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
#define LED A, 0
#define SWITCH B, 1


SET(LED) // turn on

if(READ(SWITCH)== 0){ // switch active
}
```

The sequence of pre-processor substitution was causing compile problems. I found solution for this as variable arguments
macro. http://en.wikipedia.org/wiki/Variadic_macro
The extended version is as follows

**Code:**

```
// ##############

// GPIO.h file
#define G_INPUT(port,pin) DDR ## port &= ~(1<<pin)
#define G_OUTPUT(port,pin) DDR ## port |= (1<<pin)
#define G_CLEAR(port,pin) PORT ## port &= ~(1<<pin)
#define G_SET(port,pin) PORT ## port |= (1<<pin)
#define G_TOGGLE(port,pin) PORT ## port ^= (1<<pin)
//#define G_READ(port,pin) (PIN ## port & (1<<pin))
#define G_READ(port,pin) ((PIN ## port & (1<<pin)) >> pin)


#define GPIO_INPUT(...)    G_INPUT(__VA_ARGS__)
#define GPIO_OUTPUT(...)   G_OUTPUT(__VA_ARGS__)
#define GPIO_CLEAR(...)    G_CLEAR(__VA_ARGS__)
#define GPIO_SET(...)      G_SET(__VA_ARGS__)
#define GPIO_TOGGLE(...)   G_TOGGLE(__VA_ARGS__)
#define GPIO_READ(...)     G_READ(__VA_ARGS__)
#define GPIO_READ_N(...)   G_READ_N(__VA_ARGS__)

// ########

// project.h file : keep all the definitions here

#define LED A, 0
#define SWITCH B, 1

// ###########

// Project.c file
// This can be used as follows
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
#include "include/GPIO.h"
#include "include/project.h"



GPIO_SET(LED) // Turn on the LED

if ( GPIO_READ(SWITCH)==0){ // Switch on
}
```

Please note my version of READ is different. I prefer return values as either 1 or 0.

Just my 2 cents. Thank you.

---

**milos.soucek** - Oct 09, 2009 - 11:53 PM
**Post subject:** Bit positions vs. bit masks

Hi all,

I have one question about the bit's names definitions of AVR registers, as they are done in avrlib.

In avrlib are these defined like this :

```
#define PA0 0
#define PA1 1
#define PA2 2
```

and so on.

Is there any good reason why to not have them defined directly like this :

```
#define PA0 1
#define PA1 2
#define PA2 4
#define PA3 8
```

So You do not need to shift or use any _BV() macro everytime.

Who and when needs bits defined by their number, and not directly by their mask ?

---

**Koshchi** - Oct 10, 2009 - 12:13 AM
**Post subject:** RE: Bit positions vs. bit masks

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

Is there any good reason why to not have them defined directly like this :

This is the way that Atmel defined them in assembler. Avrlibc (not avrlib, that is a separate product) gets the definitions from Atmel's definitions.

**Quote:**

Who and when needs bits defined by their number, and not directly by their mask ?

Assembler uses them directly in opcodes such a SBI, CBI, SBIC, SBIS.

You can always define your own versions of them that are bit masks instead of bit numbers.

---

**milos.soucek** - Oct 10, 2009 - 04:04 PM
**Post subject:**

Thanks for reply.
Now it is clear.
Of course I was talking about avr-libc, sorry for that.

---

**Baldrian** - Oct 13, 2009 - 08:47 AM
**Post subject:**

Bit Manipulation is clear for me, until it gets to negative values.
I have a function which returns an int16 value with following values:
-1 // OK
-2 // Failure
-3 // something
...
-75 // do something special
-128 // special detail

And of course some combinations with the special detail (-128)
-129 // OK and special detail
-130 // Not OK and special detail

But no combnination with the -75
And I don't have to use the special detail.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

I have a working solution on how to check this, but in my opinion it is not a nice solution.

**Code:**

```
if (ret == -75){  // do something
  SendEvent(DO_SOMETHING);
}
else if (~(ret) & 128)   //special detail
{
  ret +=128;   //we don't need to check the special detail
  SendEvent(ret);
}
else
  Send_Event(ret);
```

So, how can I easily handle negative return values?

---

**clawson** - Oct 13, 2009 - 09:31 AM
**Post subject:**

I don't wholly see the relevance of your question to this tutorial but to try and drag it slowly back on topic how about using negative binary return codes:

**Code:**

```
#define ERROR_CONDITION_1 -1
#define ERROR_CONDITION_2 -2
#define ERROR_CONDITION_3 -4
#define ERROR_CONDITION_4 -8
...
```

In a 16 bit signed int you could hold 15 such conditions with multiple error conditions combined with & and individual conditions tested on the return with bit testing instructions.

---

**Baldrian** - Oct 13, 2009 - 09:50 AM
**Post subject:**

OK, maybe you're right. It's not really bit manipulation what I need.
I cannot change the return value, because the function is in a Lib. I have to live with

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

the strange return values, but I'm searching for a nice solution to handle this life

Edit:
Ok. Found an easy solution:

**Code:**

```
ret *= -1; // and now I have nice positive values
...
else if (ret & 0x80)   // 128 special detail
{
  ret = ret & ~0x80;   //we don't need to check the special detail
  SendEvent(ret);
}
else
  Send_Event(ret);
```

Then I changed the events to positive values. And after that delete the magic numbers by #defines
i.e. #define SPECIAL_DETAIL 0x80

Now I'm happy

---

**LDEVRIES** - Oct 18, 2009 - 04:39 AM
**Post subject:**

I have gone through this tutorial with the view of putting all the "good stuff" in one document. The aim was to pick out the eyes out of it to keep it on track. Ie. Programming 101 - Bit manipulation, Is anyone interested in doing a proof read. Please PM me.
Lee

---

**EW** - Oct 20, 2009 - 02:16 AM
**Post subject:**

This was already put into one document: an article in Circuit Cellar magazine, July 2005 issue:

Bit Flipping Tutorial
An Uncomplicated Guide to Controlling MCU Functional
Eric Weddington

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

http://www.circuitcellar.com/magazine/180toc.htm

Unfortunately it's not a free article. However, the "Programming 101" post at the top of this thread is free. The magazine article is essentially the same information (but with grammar/spelling corrections).

---

**fever2tel** - Oct 25, 2009 - 05:35 PM
**Post subject:**

I am busy learning C, and busy writing my first GCC program, and I read in the GCC FAQ:

**Quote:**

**Why does the compiler compile an 8-bit operation that uses bitwise operators into a 16-bit operation in assembly?**
Bitwise operations in Standard C will automatically promote their operands to an int, which is (by default) 16 bits in avr-gcc.

To work around this use typecasts on the operands, including literals, to declare that the values are to be 8 bit operands.

This may be especially important when clearing a bit:

**Code:**
```
var &= ~mask;  /* wrong way! */
```

The bitwise "not" operator (~) will also promote the value in mask to an int. To keep it an 8-bit value, typecast before the "not" operator:

**Code:**
```
var &= (unsigned char)~mask;
```

This made me cringe!! As all my variables are 1 byte, and I figured my code is probably 4x as long as it needs to be because of this casting to 2 bytes (since AVR is an 8-bit processor). So I added (unsigned char) in front of all my operations. Re-compiled and my code was the exact same size??
.... "Program: 1138 bytes (13.9% Full)"

I see this topic has been posted multiple times, in the forum the general consensus is you just accept it, in the FAQ it says use casting?

---

**Koshchi** - Oct 25, 2009 - 06:57 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

In many cases the explicite cast is unnecessary. Before using casts, I would look at the assembly output to see if it is really needed.

**Quote:**

and I figured my code is probably 4x as long as it needs to be

Why 4 times? Surely the casting only occurs occasionally, not for every single operation.

---

**clawson** - Oct 26, 2009 - 09:02 AM
**Post subject:**

Just to note that a common culprit for "bloated" code is if any of the math library function is used (this can even be */-+) and the code is not set to link against libm.a but links with inferior code in libgcc.a instead. This can "cost" up to about 3K. Another culprit is over-sized versions of printf.a being used.

---

**Internetwarrior** - Dec 07, 2009 - 10:37 AM
**Post subject:** Re: RE: [TUT] [C] Bit manipulation (AKA "Programming 10

abcminiuser wrote:

A recent thread had a very nice solution which extends on the basic bit-manipulation macros. IIRC it went something along the lines of:

Defines:

**Code:**

```
#ifndef _AVR035_H_
#define _AVR035_H_

// from AVR035: Efficient C Coding for AVR

#define SETBIT(ADDRESS,BIT) (ADDRESS |= (1<<BIT))
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT))
#define FLIPBIT(ADDRESS,BIT) (ADDRESS ^= (1<<BIT))
#define CHECKBIT(ADDRESS,BIT) (ADDRESS & (1<<BIT))

#define SETBITMASK(x,y) (x |= (y))
#define CLEARBITMASK(x,y) (x &= (~y))
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
#define FLIPBITMASK(x,y) (x ^= (y))
#define CHECKBITMASK(x,y) (x & (y))

#define VARFROMCOMB(x, y) x
#define BITFROMCOMB(x, y) y

#define C_SETBIT(comb) SETBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_CLEARBIT(comb) CLEARBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_FLIPBIT(comb) FLIPBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
#define C_CHECKBIT(comb) CHECKBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))

#endif
```

Use:

**Code:**

```
#define Status_LED  PORTA, 3

C_SETBIT(Status_LED);
C_CLEARBIT(Status_LED);
```

- Dean

---

**gatoruss** - Jan 01, 2010 - 02:56 PM
**Post subject:** Re: [TUT] [C] Bit manipulation (AKA "Programming 101&qu

First of all - I would like to wish a safe and happy new year and a prosperous 2010 to everyone.

I have a question about clearing bits. I am sorry if this is a stupid question, but I wanted to confirm that I am understanding the following concept regarding "clearing" a bit.

abcminiuser wrote:

To clear bit 0 in foo requires 2 bit operators:

**Code:**

```
foo = foo & ~0x01;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

This uses the AND operator and the NOT operator. Why do we use the NOT operator? Most programmers find it easier to specify a mask wherein the bit that they are interested in changing, is set. However, this kind of mask can only be used in setting a bit (using the OR operator). To clear a bit, the mask must be inverted and then ANDed with the variable in question. It is up to the reader to do the math to show why this works in clearing the desired bit.

In the case of clearing the 0 bit, instead of "ANDing" the 0 bit against the complement of 0x01, couldn't you just "AND" it against 0x00? Further, if you were clearing the 3rd bit, for example, couldn't you "AND" it against 0b11111011?

Is the answer that you can do this, but it just isn't "good form?" And, as clawson says above:

clawson wrote:

Try to get away from 0b???????? - that's the whole point of bit shifts - you don't need to know or care what the other 7 bits are doing when you access just a single bit.

I am just trying to confirm that there isn't some other, more fundamental reason, why you need to "AND" the bit against the complement?

Thanks.

---

**JohanEkdahl** - Jan 01, 2010 - 08:28 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

**Quote:**

couldn't you just "AND" it against 0x00?

No, as that would clear the whole byte.

**Quote:**

if you were clearing the 3rd bit, for example, couldn't you "AND" it against 0b11111011?

Absolutely, but, I would consider that less clear than anding it with ~0b00000100, which in tuen is less clear than ANDing it with ~(1<<2).

I guess it is a matter of getting used to the notation. For a beginner it might be clearer with 0b00000100, but most of us think "I want to manipulate bit 2 and once you get

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

used to 1<<2 for that, where the "two" is spelled out so to say, then this technique is less error prone.

Sooner or later you start to use symbolic bit number for a lot of things, eg the different bits in the registers for eg timer and then you start to write things like

**Code:**

```
((1<<WGM01) | (1<<WGM00))
```

which says a lot more about what you are doing than eg

**Code:**

```
0b00011000
```

Add to this that the "0b" notation is not standard C and the argument in favour of the shift-notation becomes rather strong.

---

**gatoruss** - Jan 01, 2010 - 09:31 PM
**Post subject:** Re: RE: Re: [TUT] [C] Bit manipulation (AKA "Programmin

JohanEkdahl wrote:

**Quote:**

couldn't you just "AND" it against 0x00?

No, as that would clear the whole byte.

> Yes, I guess you would have to "AND" it against 0b11111110 or 0xFE.

Thanks, for your response. Doing it the way I suggested seems like it would be more work, as you would essentially be calculating the complement in your head.

---

**Internetwarrior** - Jan 29, 2010 - 05:26 PM
**Post subject:** Re: [TUT] [C] Bit manipulation (AKA "Programming 101&qu

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Just for new users:
When a bit is checked, the "PINA" must be used.
I.E. #define foo_input PINA,0.

In the same manner "PORTA" are used when bits are set or cleared.
I.E. #define foo_output PORTA,1

Correct me someone if I'm wrong

/Sebastian

---

**Koshchi** - Jan 29, 2010 - 08:33 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

You are correct about the function of the ports is correct. But your defines will work only in assembly, not C which is the focus of this thread.

---

**Internetwarrior** - Jan 31, 2010 - 10:08 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

I'm not sure what you mean by this, please elaborate.
I have been coding C for some time and these kind of definitions work perfectly when
I.E. C_CLEARBIT expects only one argument, which is the whole idea.

I might be wrong as usual, but it has worked for a few years now.

---

**Koshchi** - Jan 31, 2010 - 10:18 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

**Quote:**

I'm not sure what you mean by this, please elaborate.

In what context do you think that PINA,0 will generate legal C syntax? Perhaps it is you who should elaborate as to how C_CLEARBIT is defined.

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Internetwarrior** - Jan 31, 2010 - 10:33 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

These were defined earlier in the thread:

```
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT))
#define VARFROMCOMB(x, y) x
#define BITFROMCOMB(x, y) y
#define C_CLEARBIT(comb) CLEARBIT(VARFROMCOMB(comb), BITFROMCOMB(comb))
```

"PORTA" replaces x which replaces ADDRESS while "1"
replaces y which replaces bit, both in the CLEARBIT definition ->

```
(PORTA &= ~(1<<1))
```

Bad syntax or good syntax, I dunno, it works though...

---

**Koshchi** - Feb 01, 2010 - 01:58 AM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

I guess it is just that I would not use such trash in my programs. I have never seen the point of using 5 separate macros to generate one very simple line of code, particularly when most of those macros are simply designed to get around the limitations of macros. The define of PINA,0 makes sense only in the very specific circumstance of this particular sequence of macros.

---

**Internetwarrior** - Feb 01, 2010 - 07:26 AM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

Sure, I agree, it is not pretty with all these macros.
However, it's nice to only use one simple argument when dealing with almost 100IO's, especially if I have to change pins.
But I don't have to tell you that.

If you have a simpler/prettier way of doing the very same thing please enlighten me.

---

**clawson** - Feb 01, 2010 - 09:32 AM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

I still don't get your argument. How is:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
C_CLEARBIT(PORTA,5);
```

any clearer than:

**Code:**

```
PORTA&=~(1<<5);
```

It's actually MORE typing so it doesn't simplify the text entry for the programmer. Also the second line will be immediately familiar to any C programmer (perhaps once they've

read this thread? ) while the former will require even the most seasoned professional to go digging through .h files to unwind the layers of the onion. This does not make for easily readable/maintainable code (which should be the goal of all programmers - especially those who plan to do it professionally).

---

**JohanEkdahl** - Feb 01, 2010 - 11:33 AM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

It has been argued here from time to time that the abstraction could be placed slightly higher than the "set/clear bit" level. Instead of letting you application code be full of

**Code:**

```
C_CLEARBIT(PORTx, n)
```

you would write a thin hardware abstraction layer, eg

**Code:**

```
LED_ON()
```

Now, the details on where the LED is wired up are in one place, and now it makes even less sense to use the C_CLEARBIT() construct, as you only in one place have (sketchy)

**Code:**

```
inline void LED_ON() { PORTB &= ~(1<<5); }
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

---

**Koshchi** - Feb 01, 2010 - 07:25 PM
**Post subject:** RE: Re: [TUT] [C] Bit manipulation (AKA "Programming 10

Not only is the method that Johan showed more readable, it is far more safe. With the macros you could easily do this:

**Code:**

```
C_CLEARBIT(foo_input);
```

and the preprocessor wouldn't bat an eye, even though the clearing of bit 0 in PINA doesn't make any sense at all.

---

**BadGranola** - Mar 13, 2010 - 10:25 PM
**Post subject:**

what does it mean when the left shift operation is used in statements such as this?

**Code:**

```
REG |= (1<<BIT)
```

or

**Code:**

```
if(REG & (1<<BIT)){}
```

where BIT is a bit belonging to the register REG.

this use seems to differ from your examples of building a bit mask such as:

**Code:**

```
(0x01<<2)
```

which sets bit number 2 of an 8-bit mask.

Thanks

---

**clawson** - Mar 13, 2010 - 10:31 PM
**Post subject:**

**Quote:**

what does it mean when the left shift operation is used in statements such as this?

Have you actually READ this thread? It gives a total and unequivocal explanation of this - that's the WHOLE POINT of the tutorial (to save having to explain it to the 1 millionth newbie)

---

**JohanEkdahl** - Mar 13, 2010 - 10:34 PM
**Post subject:**

**Quote:**

what does it mean when the left shift operation is used in statements such as this?

Code:
REG |= (1<<BIT)

Surprise: The answer to your question is on page 1 of this thread, more specifically in a post by "clawson" made on Jun 28, 2007.

---

**syberraith** - Mar 18, 2010 - 02:49 PM
**Post subject:** Re: RE: Re: RE: [TUT] [C] Bit manipulation (AKA "Progra

danni wrote:

Another approach:

I like it to access bit variables like any other variables and then I can write:

if(i == 1)

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
if(i == 0)
i = 0;
i = 1;
```

which looks easy readable for me.

This can easy be done by casting a portbit as a member of a bit field.

On the attachment there is the definition of the macro SBIT.

Following an example code:

**Code:**

```
#include <io.h>
#include "sbit.h"


#define KEY0          SBIT( PINB, 0 )
#define KEY0_PULLUP     SBIT( PORTB, 0 )

#define LED0          SBIT( PORTB, 1 )
#define LED0_DDR        SBIT( DDRB, 1 )


int main( void )
{
  LED0_DDR = 1;       // output
  KEY0_PULLUP = 1;      // pull up on

  for(;;){

    if( KEY0 == 0 )     // if key pressed (low)
      LED0 = 0;         // LED on (low)
    else
      LED0 = 1;         // LED off (high)
  }
}
```

Naturally this macro can also be used for internal flag variables, not only for IO registers.

Peter

Hi Everybody,

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

I'm just getting started with micro-controllers and have pieced together a little learning project for myself from various tutorials to match what I have available to work with, namely VMLAB and WinAVR. I have found some interesting results using this:

**Code:**

```
R0 VDD RESET 4.7K
R1 N001 PB1 300
R2 N002 PB3 300
D1 VDD N001
D2 VDD N002
K1 VSS PB0
K2 VSS PB2 MONOSTABLE(30u)
```

**Code:**

```
#include <avr\sbit.h>

#define SBIT(port,pin) ((*(volatile struct bits*)&port).b##pin)

#define KEY1         SBIT( PINB, 0 )
#define KEY1_PRESET    SBIT( PORTB, 0 )

#define LED1         SBIT( PORTB, 1 )
#define LED1_DDR      SBIT( DDRB, 1 )

#define KEY2         SBIT( PINB, 2 )
#define KEY2_PRESET    SBIT( PORTB, 2 )

#define LED2         SBIT( PORTB, 3 )
#define LED2_DDR      SBIT( DDRB, 3 )


// ********************************************************
// Main program
//

int main(void) {

  LED1_DDR = 1;        // output
   KEY1_PRESET = 1;      // high

   LED2_DDR = 1;        // output
   KEY2_PRESET = 1;      // high

   for(;;){

    if( KEY1 )        // if ( key == 1 ) (high)
      LED1 = 1;       // LED off (high)
    else
      LED1 = 0;       // LED on (low)
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
      if( KEY2 == 0 )  {
        if ( LED2 )
          LED2 = 0;
        else
          LED2 = 1;
      }

    }

}
```

I dropped sbit.h into the WinAVR header folder. First thing I noticed is that VMLAB complains about accessing reserved memory the first time a write to DDRB is made.

As you can see I extended the example to include a toggled LED. This worked out less well than I thought. I started with a latched key, then found the toggle action was undependable, so I migrated to a monostable key. That helped, although only after I found a pulse width that worked well. I presume reliable toggling take some code for debouncing etc.

Lastly I found the "^= 0x01" technique fails to work for toggling bit field variables, so I had to do it the long way.

Fred

---

**Koshchi** - Mar 18, 2010 - 05:03 PM
**Post subject:** RE: Re: RE: Re: RE: [TUT] [C] Bit manipulation (AKA "Pr

Why are you putting "#define SBIT(port,pin) ((*(volatile struct bits*)&port).b##pin)" in your code? Isn't it already defined in sbit.h? Also I noticed that the download link is SBIT.H, not sbit.h. Did you change the name of that file to lowercase?

---

**syberraith** - Mar 18, 2010 - 06:21 PM
**Post subject:**

I had all the code in the c section first, the extraneous sbit define is remnant. I changed the case of the include statement. Under wine that seems to have worked. I changed it back.

It's been years since I did any coding. It's almost like beginning all over again.

For some reason the stimulation starts up with LED2 on. From the scope it looks like the state test is made before the voltage at PB2 is pulled high, so the toggle is triggered without a key press. After it gets running it works as expected.

I wonder if that is just a simulator thing.

---

**clawson** - Mar 18, 2010 - 06:23 PM
**Post subject:**

**Quote:**

I wonder if that is just a simulator thing.

No sh*t?

(trust simulators about as far as you can comfortably spit them)

---

**BadGranola** - Jul 04, 2010 - 12:40 AM
**Post subject:**

Hi, I'm trying to build a 16-bit word made up from three integer variables; address, control (both in range 0x00-0x03), and data (0x00-0xFF).

The most significant four bits of the word are irrelevant, the next two hold the address, the next two the control information, and the final 8 hold the data.

Will something like this allow me to contruct the word from these variables?

**Code:**

```
word = (address<<10) | (control<<8) | data
```

---

**JohanEkdahl** - Jul 04, 2010 - 01:15 AM
**Post subject:**

Have you tried it out? (For this kind of stuff the simulator in AVR Studio is excellent.)

But yes, that looks reasonable.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Koshchi** - Jul 04, 2010 - 04:57 AM
**Post subject:**

Or you could do this:

**Code:**

```
typefef struct
{
   unsigned int data :8;
   unsigned int control :2;
   unsigned int address :2;
} whateverYouWantToCallIt;
```

or:

**Code:**

```
typefef struct
{
   unsigned int :4; //4 bits of fiiler
   unsigned int control :2;
   unsigned int address :2;
   unsigned int data :8;
 } whateverYouWantToCallIt;
```

depending on endianness.

**Code:**

```
whateverYouWantToCallIt word;
...
word.address = address;
word.control = control;
word.data = data;
```

**qli029** - Jul 21, 2010 - 04:47 AM
**Post subject:**

This one doesnot make sense to me,

To set bit 0 in foo and then store the result back into foo:
Code:

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
foo = foo | 0x01;
```

bit OR"|" with 0x01, we will set bit 1 in foo,,
why is it to set bit 0?
I am a newbie, forgive me if i am wrong. please explain it to me.

---

**JohanEkdahl** - Jul 21, 2010 - 09:16 AM
**Post subject:**

[Removed as I totally mis-understood the post above. Snigelen nailed the correct interpretation and answer below.]

---

**snigelen** - Jul 21, 2010 - 09:55 AM
**Post subject:**

qli029 wrote:

This one doesnot make sense to me,

To set bit 0 in foo and then store the result back into foo:
Code:

**Code:**

```
foo = foo | 0x01;
```

bit OR"|" with 0x01, we will set bit 1 in foo,,
why is it to set bit 0?

This is because bits in a byte are usually numbered from 0 to 7. So the first bit is number 0.

---

**clawson** - Jul 21, 2010 - 10:53 AM
**Post subject:**

Just to add a little more to that:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
bit    shift    mask

0      (1<<0) =  0x01
1      (1<<1) =  0x02
2      (1<<2) =  0x04
3      (1<<3) =  0x08
4      (1<<4) =  0x10
5      (1<<5) =  0x20
6      (1<<6) =  0x40
7      (1<<7) =  0x80
```

---

**iA7med** - Jul 24, 2010 - 10:22 AM
**Post subject:**

That was useful , Thanks .

---

**violin_rules** - Jan 09, 2011 - 02:01 AM
**Post subject:**

Thanks!, i was looking for this information.

---

**yaswanth2008** - Mar 02, 2011 - 01:27 PM
**Post subject:**

how should i access the registers R0-R32? using c programming.
Just give me an example..

Thank you

---

**abcminiuser** - Mar 02, 2011 - 01:32 PM
**Post subject:**

**Quote:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

how should i access the registers R0-R32? using c programming.
Just give me an example..

You don't - the point of C is that you don't need to directly access the CPU temporary registers. They are handled by the compiler in order to implement your code.

The I/O and peripheral registers (like the timer control registers) are manipulated by you, but not the CPU scratch registers.

- Dean

---

**clawson** - Mar 02, 2011 - 02:12 PM
**Post subject:**

To illustrate what Dean has said consider this program:

**Code:**

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
  uint8_t mask = 0x55;

  DDRB = 0xFF;
   while(1)
   {
       PORTB ^= mask;
     if (PIND & 1<<PD0) {
        mask ^= 0xFF;
     }
   }
}
```

when built by the compiler it generates:

**Code:**

```
00000080 <main>:

int main(void)
{
  uint8_t mask = 0x55;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
  DDRB = 0xFF;
 80:   8f ef       ldi   r24, 0xFF   ; 255
 82:   84 b9       out   0x04, r24   ; 4
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
  uint8_t mask = 0x55;
 84:   85 e5       ldi   r24, 0x55   ; 85

  DDRB = 0xFF;
   while(1)
   {
      PORTB ^= mask;
 86:   95 b1       in   r25, 0x05   ; 5
 88:   98 27       eor   r25, r24
 8a:   95 b9       out   0x05, r25   ; 5
    if (PIND & 1<<PD0) {
 8c:   48 9b       sbis   0x09, 0   ; 9
 8e:   fb cf       rjmp   .-10      ; 0x86 <main+0x6>
       mask ^= 0xFF;
 90:   80 95       com   r24
 92:   f9 cf       rjmp   .-14      ; 0x86 <main+0x6>
```

In this code the compiler initially uses R24 to output 0xFF to DDRB. It then reuses R24 as 'mask' and initialises it to be 0x55. It reads PORTB into R25 then exclusive-or's this with 'mask' (R24). It then checks bit 0 of PIND and if that is set it complements the contents of 'mask' (so 0x55 switches to 0xAA).

You, the programmer, need never be aware which registers the compiler has chosen to put values or variables into.

---

**ghost800** - Mar 30, 2011 - 06:51 PM
**Post subject:**

What this program does not work help me

**Code:**

```
//************************************************************
// Project:  Analog Comparator example
// Author: winavr.scienceprog.com
// Module description: Analog comparator example with positive comparator
// input connected to Vref 1.23V. When compared voltage exceed 1.23V LED lighst on.
// When voltage drops bellow - LED turns OFF. Comparator inputis connected to ADC3
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
pin.
// ***********************************************************
#include <avr\io.h>
#include <avr\interrupt.h>
#define AINpin PA3
#define LED PD0
void Init(){
  DDRA&=~(1<<AINpin);//as input
  PORTA&=~(1<<AINpin);//no Pull-up
  DDRD|=(1<<LED); //Led pin as output
  PORTD|=(1<<LED);//Initally LED is OFF
  SFIOR|=(1<<ACME);//enable multiplexer
  ADCSRA&=~(1<<ADEN);//make sure ADC is OFF
  ADMUX|=(0<<MUX2)|(1<<MUX1)|(1<<MUX0); //select ADC3 as negative AIN
  ACSR|=
  (0<<ACD)|   //Comparator ON
  (1<<ACBG)|   //Connect 1.23V reference to AIN0
  (1<<ACIE)|   //Comparator Interrupt enable
  (0<<ACIC)|   //input capture disabled
  (0<<ACIS1)| //set interrupt on output toggle
  (0<<ACIS0);
  sei();//enable global interrupts
}
// Interrupt handler for ANA_COMP_vect
//
ISR(ANA_COMP_vect) {
if bit_is_clear(ACSR, ACO)
  PORTD&=~(1<<LED);//LED is ON
  else   PORTD|=(1<<LED);//LED is OFF
}
// ***********************************************************
// Main program
//
int main(void) {
    Init();
   while(1) {    // Infinite loop; interrupts do the rest
  }
}
```

---

**clawson** - Mar 30, 2011 - 07:54 PM
**Post subject:**

ghost800,

How does your post extend the discussion of the article in the first post of this thread?

Moderator.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**JohanEkdahl** - Mar 30, 2011 - 09:10 PM
**Post subject:**

Cliff!

When those post totally unrelated to the thread occur I think we should tossin an explanation of the difference between the "new topic" and the "new reply" buttons. I have a feeling that the distinction is not clear for many newcomers, or thet they even do not see the buttons and think that the only way to post on the forum is the "Quick Reply" area.

ghost800!

If you want to start a discussion on a new topic, use the "new topic" button at the top or bottom thread pages and forum listings here at AVRfreaks. Each discussion should commence in a separate "thread" (topic) or things become very confused and out of order. When creating a new thread please take the time to think out a really smart title (subject) so that it reflects the content of your post (extremely bad subjects are things like "Help!", "Why does my code not work" etc).

Hope this helps!

---

**reaper420** - Apr 01, 2011 - 07:27 AM
**Post subject:**

can the macros be used with ports?
for eg

**Code:**

```
int totalcomplete()
{
int retval;
if ((bit_get(PORTC, BIT(1))) && (bit_get(PORTC, BIT(0)))==0)
{
retval = 1;
}
else retval = 0;

return retval;
}
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Apr 01, 2011 - 10:33 AM
**Post subject:**

I'm curious. What would make you want to type:

**Code:**

```
if ((bit_get(PORTC, BIT(1))) && (bit_get(PORTC, BIT(0)))==0)
{
```

when you could have typed:

**Code:**

```
if ((PORTC & 3) == 0)
{
```

Do the tortuous macros really help?

(BTW do you mean to read PORTC or PINC?)

---

**reaper420** - Apr 01, 2011 - 08:15 PM
**Post subject:**

thanks! ya my mistake it should have been PINC .sorry but am new to programming an ya thanks i guess i could use

**Code:**

```
if ((PORTC & 3) == 0)
{
```

but what i meant was does this macro

**Code:**

```
#define bit_get(p,m) ((p) & (m))
```

work when a pin is specified in place of variable p. Or do we have to use macro such as

**Code:**

```
#define READ(port,pin) ((PIN ## port & (1<<pin))
```

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Apr 01, 2011 - 08:44 PM
**Post subject:**

Yes and my question is why obfuscate code with layers of macros when the plain C to achieve the same would actually be simpler. There seems to be a recent trend to try and wrap code in N levels of macro at the drop of a hat. As someone who has to read/maintain other programmers code I find that it very rarely aids the mainatainability/readability of the code but instead just makes it more difficult to follow. Perhaps you can explain what how your bit_get() macro makes the code easier to follow. ALL programmers know what an AND operation is!

---

**reaper420** - Apr 01, 2011 - 09:02 PM
**Post subject:**

thanks, and well as u said it will be simpler to use plain c but the reason is that, i am working on a project and the panel where i have to present it to has a few members who have well little or no knowledge of programming so i am hoping the macros will help in making them understand more easily.

---

**clawson** - Apr 01, 2011 - 09:55 PM
**Post subject:**

**Quote:**

so i am hoping the macros will help in making them understand more easily.

Eh? You have the option of teaching them your very specific macro or the generic C & operator. With the latter they can write 1000's of programs, with the former they are tied to some very specific code for a single situation. Why not teach them how AND works?

---

**reaper420** - Apr 01, 2011 - 10:57 PM
**Post subject:**

well cos i am a student myself. and i am showing a presentation to justify the funding of my project, and the people on the panel will be staff of various departments, some of who may not know programming .that's why am using the macros to give a generalized

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

presentation. I have even broken down my whole code into functions with general names for hopeful easier understanding.

---

**Koshchi** - Apr 02, 2011 - 01:09 AM
**Post subject:**

**Quote:**

well cos i am a student myself.

Yet you obviously don't know yourself how the macros work. So if one of them happens to ask how the macros work, you'll have a big problem, won't you.

---

**reaper420** - Apr 02, 2011 - 01:39 AM
**Post subject:**

i do have some idea on the working of macros now,thanks to the tutorial. And i do know basic programming so i know what happens when u use a particular macro. I am already done with my coding using c. Just needed the macros to hopefully make it easier to explain.

---

**eigurrola** - Apr 02, 2011 - 08:13 AM
**Post subject:**

Thanks for the great Tutorial, it's awsome!

I just got a really crazy idea and Im not sure if it can actually be impemented.

To create a byte object in which every bit corresponds to a certain Pin on a port?

I'm not sure if Im using the right words here but an example should help:

create some object, maybe a structure (?) that looks like this:
object {Bit0 = Portb bit 3
Bit1 = Portc bit 1
...
Bit7 = Portb bit 0
}

so when you store a value into this object it will assign the respective value to those pins

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

for example,
object = 0xFF
would set Portb bit 3 high, Portc bit 1 high etc.

just wondering, maybe by doing a complicated macro or somethig?

---

**clawson** - Apr 02, 2011 - 01:08 PM
**Post subject:**

**Quote:**

To create a byte object in which every bit corresponds to a certain Pin on a port?

Search in this very forum for "bitfield". The bottom line is that struct members can be defined to be 1 bit long and so you can conglomerate 8 on a single byte address. Anyway read the tutorial.

EDIT: I meant this thread:

http://www.avrfreaks.net/index.php?name ... highlight=

**Quote:**

Just needed the macros to hopefully make it easier to explain.

First rule of programming: simplest is best.

"simplest" rarely involves macros.

---

**bigmessowires** - May 14, 2011 - 06:20 AM
**Post subject:**

I've read this whole thread, and one topic that's not mentioned regarding port bit twiddling (with or without macros) is interrupts.

Is it considered good practice to disable interrupts when modifying a port pin? Especially if you're using some kind of bit_set() or bit_clear() macro?

The reason is that if the bit being set/cleared is not a constant expression at compile time, then the compiler will have to generate several instructions to read the current port value into a register, manipulate the value, and write it back. If an interrupt triggers after the port value is read, and that interrupt modifies another pin on the same port, when the main program resumes it will write back the stale value for that

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

pin. This issue was previously a bug in the Arduino digitalWrite() implementation:http://code.google.com/p/arduino/issues/detail?id=146

I don't see anyone disabling interrupts in their code-- why not? Is there a reason I've overlooked why it's not necessary?

---

**JohanEkdahl** - May 14, 2011 - 09:51 PM
**Post subject:**

**Quote:**

the compiler will have to generate several instructions to read the current port value into a register, manipulate the value, and write it back.

The compiler will not have to generate several instructions, if only one bit is manipulated. AVRs have single machine instructions to manipulate one bit. E.g. activate the optimizer of avr-gcc and see what

**Code:**

```
PORTB |= 1;
```

generates.

If several bits are modified, then a read-modify-write sequence will be generated, and you will need to turn off interrupts if the port that is manipulated is shared with ISR code. The subject has been discussed in length here at 'freaks. The search words you need are e.g. "re modify write", "atomic" etc...

Ther are also tutorials on interrupt handling in general IIRC.

The avr-gcc support library avrlibc has stuff contributed by Dean Camera ('abcminiuser') that makes atomic sequences, saving and restoring state etc easy to code.

---

**abcminiuser** - May 15, 2011 - 05:15 AM
**Post subject:**

**Quote:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

The compiler will not have to generate several instructions, if only one bit is manipulated.

It will generate several instructions for RMW if the extended IO space or SRAM is used, which can indeed lead to atomicity issues. Note that the XMEGA and AVR32s have a larger register address space with dedicated SET and CLEAR registers that allow for single-cycle RMW sequences.

On the older AVRs, you'll need to protect against interrupts if you're writing anything but a single bit to the regular IO space.

- Dean

---

**bigmessowires** - May 17, 2011 - 12:55 AM
**Post subject:**

Thanks, that makes sense, and I'll search for the atomic write threads. And to clarify, the situation I was asking about is when the bit being set/cleared is not a constant expression at compile time. I've looked at the disassembly for contant expressions like PORTB |= 1 and it is a single sbi machine instruction, as JohanEkdahl said.

---

**JohanEkdahl** - May 17, 2011 - 08:27 AM
**Post subject:**

**Quote:**

And to clarify, the situation I was asking about is when the bit being set/cleared is not a constant expression at compile time.

Oh, I missed that vital part of the question. Yes, in that case you are at risk even with single bit operations.

Sorry for any confusion.

---

**danni** - Jun 09, 2011 - 04:09 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

I have added some predefined names to my "sbit.h".

Then you can use "PORT_cn", "DDR_cn", "PIN_cn" direct on the code (c = A..L, n = 0..7).
E.g.:

**Code:**

```
#include "sbit.h"


int main( void )
{
 DDR_B3 = 1;              // output
 DDR_B7 = 1;

 for(;;){
  PORT_B3 = PIN_D0;
  PORT_B7 = !PIN_D0;
 }
}
```

Peter

P.S.:
The original article:
http://www.avrfreaks.net/index.php?name ... 198#318198

---

**jjmcousineau** - Jun 14, 2011 - 03:26 PM
**Post subject:**

Hello, this is a very good tutorial and thank you but I was wondering if some one could explain something.

When I learned this stuff in school a few years back we only ever masked like you did in the 1st half (ie. foo &= 0x01). Why and when is it necessary to use these shift masks? I was taught using motorola micro's and I was wondering if it was some architecture thing? But as far as I can see they both do the same thing.

Thanks in advance

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Jun 14, 2011 - 03:57 PM
**Post subject:**

**Quote:**

Why and when is it necessary to use these shift masks? I was taught using motorola micro's and I was wondering if it was some architecture thing?

Nothing says you HAVE to use bitnumbers with shifts. Some would argue that if you are only setting bit 5 of a DDR register to be an output that:

**Code:**

```
DDRB |= 0x20;
```

is almost as readable as:

**Code:**

```
DDRB |= (1<<5);
```

(thought the former may make the engineer count on his fingers to remember that 0x20 is bit 5! )

However where this bit shifting comes into play is to make SFR bit usage more self-documenting. For example in many AVRs bit 5 of the UCSRB register is the bit to enable the UART data register empty interrupt. Now if you read code that says:

**Code:**

```
UCSRB |= 0x20;
```

or

**Code:**

```
UCSRB |= (1<<UDRIE);
```

many engineers (at least those with a passing knowledge of the AVR) will instantly recognise that the second of those is enabling the UDRE interrupt. However it would require a VERY fastidious reader to know that 0x20 (or even (1<<5)) happens to be the UDRIE bit in that register.

Now one might then ask why do I have to use 1<<, if Atmel had written the XML file that is used for Assembler and C header file generation such that:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
#define UDRIE 0x20
```

rather than

**Code:**

```
#define UDRIE 5
```

then the usage in C would just be:

**Code:**

```
UCSRB |= UDRIE;
```

but the reason they defined the bit names as 0..7 is so that the number can be used with the AVR SBI and CBI opcodes (assuming UCSRB is at the right location). This introduces the minor inconvenience of having to use (1<<UDRIE) to convert 5 into 0x20. In at least one C compiler there is a macro defined as:

**Code:**

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

so that at least:

**Code:**

```
UCSRB |= _BV(UDRIE);
```

can be used which maybe looks a little less "complex" than (1<<UDRIE). However *all* C programmers will recognise the (1<<UDRIE) version but not all are going to know what _BV() achieves without digging into header files - which kind of destroys the readability "gain" in using it.

Remember that at the end of the day all C (and Asm) code should be written with the "next reader" in mind - which could either be the poor sap who has to fix your code 3 years after you left the company or it could simply be you in 9 months when you've forgotten that setting 0x20 in UCSRB is setting the UDRIE bit! (1<<UDRIE) tells you which bit is being set without looking it up (assuming you can at least remember the purpose of "UDRIE")

Cliff

---

**JohanEkdahl** - Jun 14, 2011 - 03:58 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

Why and when is it necessary to use these shift masks? [...] But as far as I can see they both do the same thing.

They do. It is just about clarity of code. And it is opinous.

Which is clearer? This

**Code:**

```
something &= (1<<4) | (1<<3);
```

or this

**Code:**

```
something &= 0x18;
```

?

In the next step, e.g. when dealing with bits in a special purpose register (like a timer control register) it helps a lot to use the symbolic bit numbers rather than some magic numerical constants). Since the bit numbers are, uhmmm..., bit numbers rather than bit masks we need to construct the masks from the numbers. To do this, the shift operator is used.

---

**jjmcousineau** - Jun 14, 2011 - 08:27 PM
**Post subject:**

Cheers, Thanks both of you that makes a lot of sense. Especially since I am that sap

that is fixing the code 3 years later

---

**rickywillson** - Jun 15, 2011 - 06:10 AM
**Post subject:**

Thanks for your information.

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**kumar.rohit0612** - Jun 15, 2011 - 12:59 PM
**Post subject:**

bloody-orc wrote:

That means, that nr 1 is shifted left as many times as it is needed to reach bit named CS10. Where does the compiler know that CS10 is that? well you give him the AVR's name and it's smart enough to know such things thanks to some smart programmers on

the GCC side

What is the difference between
TCCR1B |= (1 << CS10);
and
TCCR1B = (1 << CS10);

Is OR ('|') really needed there or both will work the same.

---

**clawson** - Jun 15, 2011 - 01:02 PM
**Post subject:**

The first only sets one bit and leaves the other 7 bits in the register untouched. The second sets all 8 bits - the other 7 are set to 0.

---

**JohanEkdahl** - Jun 15, 2011 - 01:05 PM
**Post subject:**

**Quote:**

Is OR ('|') really needed there

That depends on the circumstances.

**Quote:**

or both will work the same

Definitively not.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
TCCR1B |= (1 << CS10);
```

is a shorthand for the equivalent

**Code:**

```
TCCR1B = TCCR1B | (1 << CS10);
```

so all bits that was set before this operation will still be set.

OTOH, if you do

**Code:**

```
TCCR1B = (1 << CS10);
```

then only the CS10 bit will be set, and all other will be cleared.

I seriously suspect that this has been covered earlier in this thread, so you might have something to gain from browsing through the complete thread..

---

**nickkennedy** - Jul 18, 2011 - 06:06 PM
**Post subject:**

In the bit manipulation macros of the orignal post:

```
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
```

... etc., should the & in the first one be &= ? Would it make any difference?

---

**clawson** - Jul 18, 2011 - 08:55 PM
**Post subject:**

No,

But the fact that it's not obvious kind of proves how pointless these kind of macros are in aiding readability. Think about:

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
bit_set(foo, 0x20);
bit_clear(foo, 0x08);
if (bit_set(foo,0x80)) {
```

which really says (without all the parenthesis):

**Code:**

```
foo |= 0x20;
foo &= ~0x08;
if (foo & 0x80) {
```

As you can see you don't want an '=' in the last one as it's used in a different context to the other two. The confusion here is that then macro names are so similar you might think they'd all be used in the same context.

Stay clear of macros - you know it makes sense!

---

**Koshchi** - Jul 18, 2011 - 09:07 PM
**Post subject:**

**Code:**

```
if (bit_set(foo,0x80))
```

You mean "bit_get" don't you?

---

**nickkennedy** - Jul 19, 2011 - 12:13 AM
**Post subject:**

OK, I see it now. It was just my poor reading. Or poor reading glasses.

But I do like these macros. I agree I should make sure I understand how they work, but think they can prevent careless errors if I understand their proper use.

Nick

---

**jjmcousineau** - Sep 13, 2011 - 08:47 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

In regard to a problem I just had could some one explain to me the reason this doesn't work properly (it affects all the bits)

**Code:**

```
PORTD &= (0<<D_I);
```

And this one does work properly?

**Code:**

```
PORTD &= ~(1<<D_I);
```

---

**smileymicros** - Sep 13, 2011 - 09:12 PM
**Post subject:**

They would both work just fine as written.

Did you read the tutorial?

What do you mean by properly?

Smiley

---

**Koshchi** - Sep 13, 2011 - 09:25 PM
**Post subject:**

**Quote:**

They would both work just fine as written.

No they wouldn't. The first clears all bits of PORTD.

**Quote:**

I just had could some one explain to me the reason this doesn't work properly

Break it down: 0<<D_I means that you are shifting the number 0 by some number of bits (whatever number D_I is). Shifting 0 by anything will always result in 0. You then AND that with PORTD. ANDing anything with 0 will always result in 0.

**Quote:**

And this one does work properly?

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Again, break it down. 1<<D_I will shift the number 1 by D_I bits. If D_I is 2, that makes the result 4 (so in binary you went from 0b00000001 to 0b00000100). You then invert that (the ~), which in binary gives you 0b11111011. You then AND that with PORTD.

---

**abcminiuser** - Sep 14, 2011 - 02:02 AM
**Post subject:**

**Quote:**

They would both work just fine as written.

ಠ_ಠ

- Dean

---

**smileymicros** - Sep 14, 2011 - 04:47 AM
**Post subject:**

Koshchi wrote:

**Quote:**

They would both work just fine as written.

No they wouldn't. The first clears all bits of PORTD.

Which is a perfectly valid use for that notation so I repeat: 'They would both work just fine as written'. I'd never use the first notation, but who am I to say someone shouldn't use a valid C notation? In fact I've seen (0<<XXX) used in the Atmel Butterfly code, so somebody somewhere must think is okay to do this.

Now if you want to make assumptions about the intent which is unstated, then maybe they don't do what the poster wants them to do, but is that for us to decide or should he tell us what he is trying to do?

Smiley

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**JohanEkdahl** - Sep 14, 2011 - 08:30 AM
**Post subject:**

**Quote:**

In fact I've seen (0<<XXX) used in the Atmel Butterfly code, so somebody somewhere must think is okay to do this.

Perhaps, but most likely not in a statement lacking ~ and using &= . Discussing if (0<<XXX) is meaningful or clarifying must be done in the context of the complete statement.

E.g. you might very well do

**Code:**

```
someRegister = (1<<someBitNumber) | (0<<anotherBitNumber);
```

to make it clear that you relly want anotherBit to be clear. Not that the code actually (0<<anotherBitNumber) has the exact effect to actually clears that bit, but you are doing some documentation of intent in the code.

---

**clawson** - Sep 14, 2011 - 09:13 AM
**Post subject:**

**Quote:**

In fact I've seen (0<<XXX) used in the Atmel Butterfly code, so somebody somewhere must think is okay to do this.

No, no , no. The only use of a 0<< is when you want to document the fact that something is deliberately not being set. maybe something like:

**Code:**

```
TCCRB = (1 << CS02) | (0 << CS01) | (1 << CS00);
```

Too often we see threads here where the user has grasped the fact that to set a bit they use:

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
SFR |= (1 << bit);
```

but then see variants of:

**Code:**

```
SFR |= (0 << bit);
SFR &= (0 << bit);
SFR &= ~(0 << bit);
```

where they believe that using 0 in the mask is the way to switch things off. As we all know, it isn't.

This notion should be stomped on from a great height before it confuses more beginners (especially in this thread we all point them towards to learn this stuff!).

The only use of (0 << foo) is to document that foo is not being used.

---

**JohanEkdahl** - Sep 14, 2011 - 09:37 AM
**Post subject:**

**Quote:**

As we all know

Exchange "some" for "all" and we're closer to the actual situation.

---

**jjmcousineau** - Sep 14, 2011 - 03:22 PM
**Post subject:**

Hey Guys, Thanks for all your input but I believe Koshchi nailed what I was looking for, and smiley was just being ragging on me for not stating my intent explicitly.

Either way Cheers!

---

**smileymicros** - Sep 14, 2011 - 04:34 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

clawson wrote:

**Quote:**

In fact I've seen (0<<XXX) used in the Atmel Butterfly code, so somebody somewhere must think is okay to do this.

No, no , no. The only use of a 0<< is when you want to document the fact that something is deliberately not being set. maybe something like:

**Code:**

```
TCCRB = (1 << CS02) | (0 << CS01) | (1 << CS00);
```

Too often we see threads here where the user has grasped the fact that to set a bit they use:

**Code:**

```
SFR |= (1 << bit);
```

but then see variants of:

**Code:**

```
SFR |= (0 << bit);
SFR &= (0 << bit);
SFR &= ~(0 << bit);
```

where they believe that using 0 in the mask is the way to switch things off. As we all know, it isn't.

This notion should be stomped on from a great height before it confuses more beginners (especially in this thread we all point them towards to learn this stuff!).

The only use of (0 << foo) is to document that foo is not being used.

Cliff - It was late and my thought was 'jeez this guy didn't even glance at the tutorial' since all it would have taken was a glance to answer his question. Usually I just pass on by but for some reason this one exceeded a threshold, thus my response which the poster sees correctly as me ragging on him. And as usual, your response fully explains why his first usage doesn't make sense and the real 'proper' use of (0<<xxx).

Smiley

---

**CyanideChrist** - Oct 14, 2011 - 01:46 PM
**Post subject:** No Matter Gr8 Work Keep it up thnx!!!!

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

robinsm wrote:

**Quote:**

BenG wrote:
As an additional item to check if a bit is clear:

Code:
```
if(~(foo) & 0x80)
{
}
```
My 1st choice would be for the following which, IMHO, is easier to "read":
Code:
```
if ( ( foo & 0x80 ) == 0 )
{
...
}
```
should result in the same compiler generated code.

Don

and another way...

**Code:**

```
if (!( foo & 0x80 ))
{
  ...
}
```

---

**whiteman7777** - Dec 19, 2011 - 06:30 PM
**Post subject:**

now wait a moment if the compiler knows a bit's name as variable then why it hasn ot been writen like this:

```
while (CS10==0);
```

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Dec 19, 2011 - 06:35 PM
**Post subject:**

Do not cross post and do not post to tutorials except to suggest improvements to the original article.

The last 202,030 people who have read this thread clearly understood the reason. If you don't I can only suggest you re-read until you understand. Also see the point I just made regarding bit number vs. bit mask in your cross-post.

---

**ViniciusCarvalho** - Dec 30, 2011 - 09:42 PM
**Post subject:**

how to manipulate each bit of a unsigned char variable doing something like that?

```
#define bit0 variable_unsigned_char
#define bit1 variable_unsigned_char
#define bit2 variable_unsigned_char
#define bit3 variable_unsigned_char
#define bit4 variable_unsigned_char
#define bit5 variable_unsigned_char
#define bit6 variable_unsigned_char
#define bit7 variable_unsigned_char

//in code do something like that
bit0 = 1;
bit3 = 0;

??????????????????
```

---

**clawson** - Dec 30, 2011 - 10:06 PM
**Post subject:**

**Quote:**

how to manipulate each bit of a unsigned char variable doing something like that

You came to the wrong tutorial, try this one:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

http://www.avrfreaks.net/index.php?name ... highlight=

Having said that it's pretty clear you haven't even read the whole of this thread:

http://www.avrfreaks.net/index.php?name ... 728#835728

---

**ViniciusCarvalho** - Dec 31, 2011 - 01:37 AM
**Post subject:**

I'm trying to do something like this.

```
//************************************
ram far unsigned char RAM_used[5];

#define p0 RAM_used[0]
#define p1 RAM_used[1]
#define p2 RAM_used[2]
#define p3 RAM_used[3]
#define p4 RAM_used[4]


//than I'm trying to control every bit of p0,p1,p2,p3,p4


struct _8bits
{
unsigned bit0:1;
unsigned bit1:1;
unsigned bit2:1;
unsigned bit3:1;
unsigned bit4:1;
unsigned bit5:1;
unsigned bit6:1;
unsigned bit7:1;
}

#define p0_b0 p0.bit0
#define p0_b1 p0.bit1
...

matriz
-----> byte
-----------> bit


//for when I do something like this
p0_b0 = 1;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
// bit p0_b0 = 1
// byte p0 = 0x01
// RAM_used[0] = 0x01


//because, sometimes I will modify a single bit, sometimes the entire byte and the
entire RAM too.


//for writing the EEPROM, I used a pointer to RAM_used;
//for some situations I move some value to the byte
//and for rapidly actions I set the bit

//I'm having some troubles... everything that I try generate a error....

HELP!
```

---

**Koshchi** - Dec 31, 2011 - 02:37 AM
**Post subject:**

**Code:**

```
struct _8bits
{
  unsigned bit0:1;
  unsigned bit1:1;
  unsigned bit2:1;
  unsigned bit3:1;
  unsigned bit4:1;
  unsigned bit5:1;
  unsigned bit6:1;
  unsigned bit7:1;
}
```

First, this requires a ";" at the end of it. Second, once you define it, you never use it (just putting .bit0 after some random variable will not work). Third, stop using macros until you actually know how to code. Using macros at this point can only introduce more errors than you already have.

---

**ViniciusCarvalho** - Dec 31, 2011 - 05:15 AM
**Post subject:**

yeahhh! I did it!

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Let's pass to other guys! ;D

**Code:**

```
//*********************************************************************
*****************
/*
=========================================
#define var (*(type*) &address)
=========================================

1.create unsigned char with fix address in ram
#define   var_A (*(unsigned char*) 0x10)

2.create unsigned char into the RAM_test[0]
#define var_B (*(unsigned char*) &RAM_test[0])
2a.It looks like crazy because you already created the matrix, so, do that (SAME
THING)
#define var_B    RAM_test[0] // <----

3.create a struct into the RAM_test[0]

typedef union
{
  unsigned char byte;
    struct
    {
      unsigned bit0:1;
      unsigned bit1:1;
      unsigned bit2:1;
      unsigned bit3:1;
      unsigned bit4:1;
      unsigned bit5:1;
      unsigned bit6:1;
      unsigned bit7:1;
    };
} estrutura;

//************
#define var_C (*(estrutura*) &RAM_test[0])
/************

****** bit:field wide has to be the same wide of RAM_test[0] ******

OBS.: var_C full access variable

>>to access it value
var_C.byte
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
>>to access a single bit
var_C.bit0


//for make it more easily

#define vb0    var_C.bit0
#define vb1    var_C.bit1
#define vb2    var_C.bit2
#define vb3    var_C.bit3
#define vb4    var_C.bit4
#define vb5    var_C.bit5
#define vb6    var_C.bit6
#define vb7    var_C.bit7

//example

vb0 = 1;        //bit

var_C.byte = 0x01; //byte

RAM_test[0] = 0x01; //matrix

*/
```

---

**Koshchi** - Dec 31, 2011 - 06:46 AM
**Post subject:**

**Code:**

1.create unsigned char with fix address in ram

```
#define   var_A (*(unsigned char*) 0x10)
```

And how do you know that this does not stomp on something that is already at that address?

What are var_A and var_B for when you never use them?

Where do you ever define RAM_test?

**Quote:**

//for make it more easily

The only thing that I see that it does is obfuscate the code.

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Dec 31, 2011 - 12:49 PM
**Post subject:**

I don't get it - I gave a link back to danni's excellent sbit.h and you still battle on in this half baked attempt to replicate it - why?

---

**JollyMolly** - Mar 12, 2012 - 05:28 PM
**Post subject:**

#define READ(port,pin) PIN ## port & (1<<pin)

Can anyone explain how this one works

and is it possible to make one that reads when using pullup resistor on pin, Not sure if this one does it.

Is this a valid statement if(Read(port,pin) == 1)

---

**glitch** - Mar 12, 2012 - 05:40 PM
**Post subject:**

the macro uses the past operator to join the text PIN with whatever value is in 'port' thus if you said read(A, 1) the intermediate stage is "PIN ## A" which is converted to "PINA".

There is no difference in reading a PIN register whether the pullups are enabled or not, so the statement works equally as well in both cases.

Unfortunately your if statement does not work, because the result of the read is the setting of the bit in it's natural position, not in the 1's position. [unless reading pin 0].. you can test for 0, but not 1 like that. Consider the result to be boolean, where 0=false, and any non-zero value is true.

if you want 1 or 0, you can sue a little obfuscated c trickery.
if(!!Read(port,pin) == 1)

I'll leave it to you to figure that one out

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**clawson** - Mar 12, 2012 - 05:42 PM
**Post subject:**

Or just drop the ==1, anything non-zero is true. hence:

**Code:**

```
if (Read(port,pin)) {
```

But why not use danni's macros:

http://www.avrfreaks.net/index.php?name ... 728#835728

then you can:

**Code:**

```
#define SENSOR PIN_B3

if (SENSOR) {
```

which seems more readable to me than having to invoke some Read() macro.

---

**JollyMolly** - Mar 12, 2012 - 05:57 PM
**Post subject:**

Would work but I can't use that statement, I just wanted to learn how to use Read, my sensor values will come from the adc so I would just use the standard

**Code:**

```
If(sensor_Value (operator) WHATEVER_VALUE)
{
do this
}
```

thanks anyways, I guess it would work if I was using digital sensors maybe?

One question can you do a double define;
for example

**Code:**

```
#define READ(port,pin)   PIN  ## port &   (1<<pin)
#define LED_CHECK    READ(A,1)
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Does that make sense, so whenever I want to check led status I just

**Code:**

```
if(LED_CHECK){}
```

I guess it's same as danni's macro, just more crap

---

**mahyarelc** - Mar 28, 2012 - 04:40 AM
**Post subject:**

Hi,

I'm new to the assembly
could you explain these instructions (in the datasheet, USART section)

**Code:**

```
/* Enable receiver and transmitter */
UCSRnB = (1<<RXENn)|(1<<TXENn);
/* Set frame format: 8data, 2stop bit */
UCSRnC = (1<<USBSn)|(3<<UCSZn0);
```

I don't know how to use the left shift operator to set these bits

Thanks,
Mahyar

---

**clawson** - Mar 28, 2012 - 09:35 AM
**Post subject:**

**Quote:**

I don't know how to use the left shift operator to set these bits

Suggest you read the thread:

Bit manipulation 101

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

in Tutorial Forum which will explain everything.

.... Oh, wait a minute ...

OK so which bit of the explanation in this thread did you not understand exactly?

---

**mahyarelc** - Mar 28, 2012 - 05:49 PM
**Post subject:**

I sort of get it,
but to make sure:

**Code:**

```
/* Set frame format: 8data, 2stop bit */
UCSRnC = (1<<USBSn)|(3<<UCSZn0)
```

it means shift 1 to the left for USBSn times (as USBSn is bit no.3 in UCSRnC register, we shift 1, 3 times and get 00001000)
the same pattern for UCSZn0(bit no.1) provides us 00000110

Is it correct?

another question is why we write 1 in UCSZn0 and UCSZn1 for the USART? (the code is from datasheet, page 177, mega1284p)
because datasheet says the initial values for these bits are set to 1 by default. (register in page 191 )

Thanks
Mahyar

---

**clawson** - Mar 28, 2012 - 06:08 PM
**Post subject:**

**Quote:**

Is it correct?

You got it. But I doubt you want to set USBS as almost always you just want 1 stop bit.

**Quote:**

another question is why we write 1 in UCSZn0 and UCSZn1 for the USART? (the code is from datasheet, page 177, mega1284p)
because datasheet says the initial values for these bits are set to 1 by default.
(register in page 191 )

Yup it's quite ridiculous isn't it (even worse on CPUs that use the URSEl bit which makes it more complex). 99% of users will always want the 8N1 default.

---

**ganseree** - Apr 02, 2012 - 04:06 AM
**Post subject:**

really nice instruction for newbies like me

---

**alexotano** - Apr 19, 2012 - 02:01 AM
**Post subject:**

clawson wrote:

I think this tutorial is trying to be as generic as possible. Not all the AVR C compilers have all the bit names defined in the header files for each AVR part so

**Code:**

```
TCCR1B |= (1 << CS10);
```

won't necessarily work on all compilers.

Grat Tutorial Dean!! Im reading all of your tutos from your web!!

About the part of the code wrote here by clawson, if some one is able to answer a silly question...

why do you use |= ?? I download al the example codes and they work the seam without it. I really dont understand why the OR operation bet (1 << CS10) and TCCR1B ... and there's other thing like in the case of:
**Code:**

```
TCCR1B |= (1 << CS10) | (1 << CS12)
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

prescaling to 1024 (ATMEGA2560)

Why all the ORs. Didt understand well that part.

Thanks for the time you spend reading.

Alex

---

**clawson** - Apr 19, 2012 - 11:04 AM
**Post subject:**

**Quote:**

why do you use |= ??

But that is EXACTLY what's explained in this very tutorial? I'm not going to repeat the text of the tutorial here so I suggest you go back to page 1 and re-read it until you understand what it is saying.

Maybe it helps to add that:

1) OR and only switch additional bits from 0 to 1
2) AND can only switch additional bits from 1 to 0
3) << means move the thing on the left the number of places given on the right. If the thing on the left is just 1 then it's saying "form a number with 1 in bit position <whatever is on the right>". This therefore converts a bit number to a bit mask.

---

**capcom** - Jul 24, 2012 - 02:16 PM
**Post subject:**

Hi, thanks for the superb tutorial.

I am having a little bit of trouble wrapping my head around the code:

**Code:**

```
if(foo & 0x80)
{
}
```

So this checks if foo's bit number 7 is a 1 or 0, right?

Supposing foo was, for example, 10100110 and we use the & operator with 10000000 as shown in your if statement above.

Now, this will return 10000000, right? But in order for the if statement to execute, shouldn't the code in the brackets result in a value of 1? I suppose it makes sense if the if statement executes for any value greater than zero.

Sorry if I am not explaining my problem well, I hope you understand.

Thanks again!

---

**Kartman** - Jul 24, 2012 - 02:26 PM
**Post subject:**

True is a non-zero value, false is zero. Therefore, since the result is non-zero, the code in the brackets will execute. You might want to read up on the difference between logical and bitwise operstors whichis covered in any half decent book on C.

---

**capcom** - Jul 24, 2012 - 02:48 PM
**Post subject:**

Got it, thanks. Makes perfect sense now. And I will read up on that to refresh my memory too.

---

**markah** - Aug 09, 2012 - 10:40 AM
**Post subject:** SBIT

Hi, Great tutorial and how I got to get to grips with bit manipulation, still v new to it all though.
Now I am writing a program with several I/Os and I used the SBIT posted and mentioned a few times in this Tutorial.
I used it in places for control of pins eg

**Code:**

```
#define VALVE   SBIT(PORTD,7)
#define ALARM   SBIT (PORTB,2)
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

also I defined ON=1 and OFF=0
so something like

**Code:**

```
ALARM=OFF;
```

seems to work.
Then I needed some flags for telling me various states of things so I added a register

**Code:**

```
volatile unsigned char CONTROL_REG;
```

and thought I could use this eg

**Code:**

```
#define Start_FLAG SBIT(CONTROL_REG,0)
```

so

**Code:**

```
if (Start_FLAG ==ON)
        {   }
```

.
I saw at the bottom of the entry in the TUT that

**Quote:**

Naturally this macro can also be used for internal flag variables, not only for IO registers.

Now when I compile with Studio 6 I get warnings where I refer to the regester eg
**Quote:**

dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

If I compile again they go away.

Please can anyone tell me if I am doing something daft and help me understand what the warning means Thanks
Mark

---

**clawson** - Aug 09, 2012 - 11:36 AM
**Post subject:** RE: SBIT

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

In the miscellaneous section of AVR/GNU C Compiler options you may find it illuminating to add --save-temps. When you build the output directory (probably GccApplicationN\Debug) will then contain a foo.i and foo.s for each foo.c in the build. If I build:

**Code:**

```
#include <avr/io.h>
#include "sbit.h"

#define OFF 0
#define ON 1

volatile unsigned char CONTROL_REG;

#define Start_FLAG SBIT(CONTROL_REG,0)

int main(void) {
  while(1) {
    if (Start_FLAG == ON) {
      PORTB = 0x55;
    }
  }
}
```

and then study the .i file I find that the macros have expanded to be:

**Code:**

```
 if (((*(volatile struct bits*)&CONTROL_REG).b0) == 1) {
```

If you read this interesting page:

http://blog.worldofcoding.com/2010/02/s ... blems.html

You'll understand the issue. As it says there one "fix" is to modify sbit.h to have:

**Code:**

```
struct bits {
  uint8_t b0:1, b1:1, b2:1, b3:1, b4:1, b5:1, b6:1, b7:1;
} __attribute__((__packed__, __may_alias__));
```

but I don't think adding "__may_alias__" in that is really much different from switching off the -Wstrict-alias option.

The bottom line is that it's just a warning of something you may need to be concerned about but as the struct is clearly the same size as the unsigned char I don't really see an issue with it.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**markah** - Aug 09, 2012 - 02:57 PM
**Post subject:** RE: SBIT

Thanks again for your help,

I think I get it, makes for interesting reading, a little bit over the head of me as a novice. They say a little knowledge is dangerous!
I'm still not sure if I should worry about it.
There seems to be conflicting opinnion in the blog.

---

**Magister** - Sep 25, 2012 - 03:57 PM
**Post subject:** RE: SBIT

I am using these defines for bit manipulation (source)

**Code:**

```
// Copyright (c) 2007 Roboterclub Aachen e.V.
#define  PORT(x)       _port2(x)
#define  DDR(x)        _ddr2(x)
#define  PIN(x)        _pin2(x)
#define  REG(x)        _reg(x)
#define  PIN_NUM(x)    _pin_num(x)

// use this block of macros in your code
#define  RESET(x)     RESET2(x)
#define  SET(x)       SET2(x)
#define  TOGGLE(x)    TOGGLE2(x)
#define  SET_OUTPUT(x)  SET_OUTPUT2(x)
#define  SET_INPUT(x)  SET_INPUT2(x)
#define  SET_PULLUP(x)  SET2(x)
#define  IS_SET(x)    IS_SET2(x)
#define  SET_INPUT_WITH_PULLUP(x)  SET_INPUT_WITH_PULLUP2(x)


#define  _port2(x)  PORT ## x
#define  _ddr2(x)  DDR ## x
#define  _pin2(x)  PIN ## x

#define  _reg(x,y)    x
#define  _pin_num(x,y)  y

#define  RESET2(x,y)    PORT(x) &= ~(1<<y)
#define  SET2(x,y)    PORT(x) |= (1<<y)
#define  TOGGLE2(x,y)  PORT(x) ^= (1<<y)
#define  SET_OUTPUT2(x,y)  DDR(x) |= (1<<y)
```

```
#define   SET_INPUT2(x,y)     DDR(x) &= ~(1<<y)
#define   SET_INPUT_WITH_PULLUP2(x,y)   SET_INPUT2(x,y);SET2(x,y)

#define   IS_SET2(x,y)   ((PIN(x) & (1<<y)) != 0)
```

In the code you then use the macro (you do not have to use the one ending with '2').

**Code:**

```
#define EnablePin C,1
#define lbuttonPin B,1
...
SET_OUTPUT(EnablePin);
SET_INPUT_WITH_PULLUP(lbuttonPin);
...
RESET(EnablePin);
...
SET(EnablePin);
...
if (IS_SET(lbuttonPin)) {...}
```

I prefer them to using sbit.h because you do not have to use DDR_ or PORT_ or PIN_, you just use the same name everywhere, wherever you set/reset/test/etc

---

**agelectronic96** - Nov 02, 2012 - 05:22 AM
**Post subject:**

Grateful information , I was following this information for 3 days, finally, I found it.... , thank you moderator

---

**saikat36** - May 03, 2013 - 07:12 AM
**Post subject:**

Thanks to all freaks for this tutorial.This is like a gift for noobs like me.

---

**Arabian** - Jul 12, 2013 - 10:35 AM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

hello
i am new in avr programing.

it is not good that we search 2 or 3 weeks to find out what << or >> do in avr-gcc.

please tell me:
1- what does
<< bit LEFT SHIFT
>> bit RIGHT SHIFT mean?
what is this output?

0b11110000 << 4
output:
0b00001111 or 0b00000000

2- is there any official web site for avr-gcc to help us what is keywords, operators and etc in avr-gcc?

---

**clawson** - Jul 12, 2013 - 11:34 AM
**Post subject:**

**Quote:**

it is not good that we search 2 or 3 weeks to find out what << or >> do in avr-gcc.

Surely if you type "C << operator" into Google you come to:

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

On that page it explains it is a "bitwise left shift" and that is a link to:

http://en.wikipedia.org/wiki/Bitwise_shift

"See Also" on that page leads you to:

http://en.wikipedia.org/wiki/Bitwise_operations_in_C

and apart from anything else how on earth are you learning to program in C if you don't have a manual?

2) none of this is specific to avr-gcc. It is the C Programming Language. It wil be the same in any C compiler that adheres to the C Standard. Most people by a copy of this:

http://en.wikipedia.org/wiki/The_C_Programming_Language

because it explains exactly how C will operate no matter which C compiler you use.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**JohanEkdahl** - Jul 12, 2013 - 12:28 PM
**Post subject:**

**Quote:**

it is not good that we search 2 or 3 weeks to find out what << or >> do in avr-gcc.

1. Welcome to AVRfreaks!

2. Your remark shows that you need to trim your search skills. 2 or 3 weeks to find that tutorial is your problem.

3. You have been searching "for 2 or 3 weeks", but yet you joined AVRfreaks

today..

**Quote:**

is there any official web site for avr-gcc to help us what is keywords, operators and etc in avr-gcc?

The shift operators, as has been remaked by clawson above, are not specific to avr-gcc. They are standard C. Get a good C book. Read. (Or would you expect avr-gcc documentation to hold information about the if- and while-statements, on assignments, on the assignment operator etc too? Of-course bot. The avr-gcc documentation coers what is specific to avr-gcc. For standard C it is any document describing standard C. For GCC specifics it is the GCC documentation. For avr-gcc, most of it is also in the GCC documentation. The GCC documentation is available on The Web. Google it.)

Also, avr-gcc/avrlibc is the work of devoted voluntaries. They do not get paid. You either more or less accept the state of those tools and the documentation that come with them, or you pay money to buy a commercial compiler/IDE and support for it. There is no such thing as a free lunch.

If you not only are new to AVRs and avr-gcc but also to the C programming language then you will have a learning experience ahead of you. There will be no one-stop solution to satisfy all your information needs. You will need at least these:

1. A good C book, or tutorial on the net.

2. The avr-gcc/avrlibc help. If you are using Atmel Studio and the avr-gcc that comes with that then the help is on your hard disk and integrated in Atmel Studio.

3. The data sheet for your AVR model.

4. Tutorials etc on programming AVRs. AVRfreaks have a whole forum devoted to

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

tutorials. Get yourself acquainted with the contents of that forum, and it will serve you well.

5. Help on The Web by unpaid voluntaries. We are here. Your job is to ask your questions in a manner that gets us interested in your problem. Present clear and complete information. Read and give feedback on advice that is given. (A whining post like yours above is not the best way to start out...) By and large the AVRfreaks community will treat you as you treat it.

Once again, welcome to AVRfreaks!

---

**Arabian** - Jul 12, 2013 - 03:33 PM
**Post subject:**

thank you all.
your links are my answer exactly.

finally i write my first project and test it in VMLAB.
so happy.

I was programing Turbo C and Pascal in 1996-2002.
but i studied management in university.

I try to program Atmega16.
where can i find my answers?

my questions are about timer:
how can i turn timer on?
how can i stop timer?
how can i enable timer overflow interrupt?
how should be timer overflow interrupt procedure?

thank you again.

---

**Koshchi** - Jul 12, 2013 - 04:25 PM
**Post subject:**

**Quote:**

where can i find my answers?

my questions are about timer:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Right here in the tutorial section.

---

**clawson** - Jul 12, 2013 - 05:04 PM
**Post subject:**

**Quote:**

Right here in the tutorial section.

Combined with the atmega16 datasheet.

If you have further questions about this can you start a new thread in AVR Forum. The purpose of this thread is simply for corrections and clarifications of issues about Bit Manipulation that are outlined in the first post - not timers.

---

**zu_adha** - Jul 17, 2013 - 11:06 AM
**Post subject:**

noob need help.
can tell me what the different in a simple way

**Code:**

```
PORTB |= (1<<6);
PORTB |=  (1 << PB6);
PORTB |=  (1 << PORTB6);
PORTB |= _BV(6);
PORTB |= _BV(PB6);
```

---

**JohanEkdahl** - Jul 17, 2013 - 12:24 PM
**Post subject:**

**Quote:**

noob need help.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

First thing to do: Shift to the first page and read the tutorial. If someone put time into that then do them the honor of reading it. Then ask specific questions, referring to that text, on what you don't understand in it.

---

**Koshchi** - Jul 17, 2013 - 03:49 PM
**Post subject:**

**Quote:**

can tell me what the different in a simple way

No difference whatsoever. Once the pre-processor is done with parsing the macros and resolving the constants, they all end up as:

**Code:**

```
PORTB |= 0x40;
```

---

**zu_adha** - Jul 18, 2013 - 05:51 AM
**Post subject:**

JohanEkdahl wrote:

**Quote:**

noob need help.

First thing to do: Shift to the first page and read the tutorial. If someone put time into that then do them the honor of reading it. Then ask specific questions, referring to that text, on what you don't understand in it.

i'm sorry, but i have read the 1st page, and i don't have the answer, i mean why do program have many way to write that code (PORTB |= 1<<PB1,etc) if only have one purpose?

---

**JohanEkdahl** - Jul 18, 2013 - 08:23 AM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

i mean why do program have many way to write that code (PORTB |= 1<<PB1,etc) if only have one purpose?

The << operator is a part of the C language itself.

The rest of the variations comes not form the C language as such but from what different people have written in header files that you include.

So, to set bit 6 in PORTB you write

**Code:**

```
PORTB |= (1<<6);
```

E.g. some people think that the use of the << operator is not pretty, or understandable (or whatever) so they hide it behind a macro that they define like so:

**Code:**

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

And can then write

**Code:**

```
PORTB |= _BV(6);
```

Now, the use of the explicit '6' here might not seem so troublesome, but for other registers where bits have special meanings it will make for obscure code if the bit number is given as a number. So there are definitions of bit numbers for all the bits of all special purpose registers (the ones where you control UARTS, timers etc..).

If a Timer/Counter Control Register has a bit at position 4 for Timer Overflow Interrupt Enable then you might write

**Code:**

```
TCCR |= 1<<4;
```

but it would be much clearer to code e.g.

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
TCCR |= 1<<TOIE;
```

This has then been worked back to the digital I/O ports where the names of those bit numbers simply become e.g. PB0, PB1, PB2...

So now someone can code

**Code:**

```
PORTB |= (1<<PB6);
```

or

**Code:**

```
PORTB |= _BV(PB6);
```

Again, all variation here is due to what is coded in header files. They are not defined in the C language as such.

---

**clawson** - Jul 18, 2013 - 09:52 AM
**Post subject:**

To this specific part of the question:

**Code:**

```
PORTB |= (1<<6);
PORTB |=  (1 << PB6);
PORTB |=  (1 << PORTB6);
```

the choice amongst those is purely down to personal taste. If you take a typical device header file (I used mega16) and look at the definitions of bit 6 relating to PORTB it has all of:

**Code:**

```
#define PINB6   6
#define DDB6    6
#define PB6     6
```

(no definition of PORTB6 in fact - just PB6) so yuou can use anything that will expand

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

out to be "6" in either (1<<n) or _BV(n) or whatever other way you want to expand it out to be a bit mask. So you could use:

**Code:**

```
PORTB |= (1 << DDB6);
PORTB |= _BV(PINB6);
```

Sure those bits are supposedly for the DDR register and the PIN register but they are all 6 so any of them could be used. My own personal preference in fact is always to use the Ppn form so PB6 or whatever and use it for all three when accessing any of PORTB, PINB or DDRB. But each to their own - you may choose something else.

BTW there's nothing magic in any of this a pre-processor macro is just a string substitution so you can use anything that would equate to 6 in order to access bit 6. If you look at the iom16.h file it has:

**Code:**

```
C:\Program Files\Atmel\Atmel Toolchain\AVR8 GCC\Native\3.4.2.1002\avr8-gnu-
toolchain\avr\include\avr>grep #define iom16.
h | grep -w 6
#define TWS6    6
#define TWA5    6
#define ADSC    6
#define REFS0   6
#define ACBG    6
#define TXCIE   6
#define TXC     6
#define SPE     6
#define WCOL    6
#define PIND6   6
#define DDD6    6
#define PD6     6
#define PINC6   6
#define DDC6    6
#define PC6     6
#define PINB6   6
#define DDB6    6
#define PB6     6
#define PINA6   6
#define DDA6    6
#define PA6     6
#define UMSEL   6
#define WGM20   6
#define ICES1   6
#define COM1A0  6
#define ADTS1   6
#define WGM00   6
#define ISC2    6
#define SE      6
#define TWEA    6
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
#define RWWSB   6
#define TOV2    6
#define TOIE2   6
#define INTF0   6
#define INT0    6
#define TIMER1_COMPA_vect_num   6
```

So any of those thing *could* be used. You could write:

**Code:**

```
PORTB |= (1 << TXCIE);
```

and it would still set bit 6. You could even use:

**Code:**

```
PORTB |= (1 << TIMER1_COMPA_vect_num);
```

I'm not suggesting you do this (unless you deliberately wanted to make the code very confusing for the reader!) but simply pointing out that there's nothing special about where the 6 you use comes from.

---

**jgmdesign** - Aug 01, 2013 - 03:01 PM
**Post subject:**

Interesting Tutorial. I am going to have to include a lot of this into what I am doing. Makes C so much easier to work with.

EDIT:

In these defines:

**Quote:**

```
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
```

Is the 'p' and 'm' a standard 'c' tag, or could I use 'x' and 'y' in their place for example?

---

**sternst** - Aug 01, 2013 - 05:20 PM
**Post subject:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

jgmdesign wrote:

In these defines:

**Quote:**

```
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
```

Is the 'p' and 'm' a standard 'c' tag, or could I use 'x' and 'y' in their place for example?

You can use whatever you like, even whole words like:

**Code:**

```
#define bit_set(port,mask) ((port) |= (mask))
```

Does this perhaps give you an idea why 'p' and 'm' were originally chosen?

---

**clawson** - Aug 01, 2013 - 05:32 PM
**Post subject:**

**Quote:**

a standard 'c' tag,

Just to be a pedant but this is not C. This is the pre-processor. A lot of people have a lot of trouble trying to see the distinction between the two. When you type:

**Code:**

```
bit_set(0x1234, 0x40);
```

in your code the C compiler never "see" bit_set() nor p nor m. All it sees is:

**Code:**

```
((0x1234) |= (0x40));
```

The pre-processor has already "eaten" all the bit_set(p,m) stuff before the .i file is even seen by the C compiler.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

It can be enlightening to add --save-temps to your compiler options then for a foo.c file you will find that a foo.i file is generated. If you used:

**Code:**

```
#include <avr/io.h>

#define bit_set(p,m) ((p) |= (m))
#define BIT(x) (0x01 << (x))

int main(void) {
  bit_set(PORTB, BIT(5));
  while(1) {
  }
}
```

the C compiler sees:

**Code:**

```
int main(void) {
 (((*(volatile uint8_t *)((0x05) + 0x20))) |= ((0x01 << (5))));
 while(1) {
 }
}
```

(it's because you don't usually want to type such convoluted nonsense that you use pre-processor macros in the first place - most of the above actually comes from "PORTB" which is also a macro).

---

**jgmdesign** - Aug 02, 2013 - 11:24 AM
**Post subject:**

I 'C'.

Thanks

---

**jgmdesign** - Aug 05, 2013 - 04:27 AM
**Post subject:**

JohanEkdahl wrote:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Quote:**

Code:
if (PIND & ~(1<<PD0))

This code test if bit is not set?

No, it tests if any other bit is set. Take it step by step:

1:

**Code:**

1<<PD0

make a bitmask 00000001.

2:

**Code:**

~(1<<PD0)

negates every bit in that mask so you get 11111110

3:
A bitwise and of that and PIND thus will produce a 1 in any position where the bitmask has a 1 and PIND has a 1. Or in other words, it will produce a non-zero value if any of bits 7 of PIND to 1 is non-zero.

What you want is

**Code:**

if ( !(PIND & (1<<PD0)))

Work through that, in a similar manner as I did above, step by step, to see why this is diferent and will do what you ask for. (Please note that the logical NOT operator (!) is used.)

An alternative way would be to

**Code:**

if (~(PIND) & (1<<PD0))

as BenG has pointed out in the first page of this thread.

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

I read this several times and although I thought it made sense I of course must be missing something.
In the code below I want to upon start up display two lines of text. Which worked. I then want to look at portb.0 and if it is pressed display what is inside the {}

I tried using the example above but it does not appear to get teh unit to change the screen. I know I am doing something wrong because when I hit pause during debugging, the disassembly window pops up. I am using AS6 and an STK600 with a MEGA2560

**Code:**

```
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <stdint.h>
#include <stdio.h>
#include "lcd.h"



void avrinit(void)
{
  DDRA = 0xFF;   //porta all outputs for LCD
  PORTB = 0xFF;
}
int main(void)
{   avrinit();
  lcd_init(LCD_DISP_ON_CURSOR_BLINK);

   /* put string to display (line 1) with linefeed */
   lcd_puts("LCD Test Line 1\n");

   /* cursor is now on second line, write second line */
   lcd_puts("Line 2");

while (~(PIND) & (1<<PD0))
{lcd_clrscr();
  lcd_puts("AVRfreaks");


}
}
```

What I end up with is the screen scrolling through both sets of texts at a very high rate. And if I hit pause, I get the disassembly window letting me know I screwed up. I am gonna work on this for a few more hours before bed.

EDIT:

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**I GOT IT TO WORK!!**

I found my boo boo's with a lot of reading and google but it works now.

**Code:**

```
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <stdint.h>
#include <stdio.h>
#include "lcd.h"



void avrinit(void)
{
  DDRA = 0xFF;   //porta all outputs for LCD
  PORTB = 0xFF;
}
int main(void)
{   avrinit();
  lcd_init(LCD_DISP_ON_CURSOR_BLINK);
while(1)
{


if(bit_is_set(PINB,0))
  { lcd_clrscr();
    /* put string to display (line 1) with linefeed */
   lcd_puts("LCD Test Line 1\n");

   /* cursor is now on second line, write second line */
   lcd_puts("Line 2");
   while(bit_is_set(PINB,0))
   {
   }

  }

if(bit_is_clear(PINB,0))
{lcd_clrscr();
  lcd_puts("AVRfreaks");
  while(bit_is_clear(PINB,0))
  {
  }

}

}
}
```

I know the indentation sucks. I figured that out after I lost track of my {}'s. But I needed to hold after the screen did it's write. In other words, the screen kept refreshing, so I insert the WHILE statements and there it goes.
I have to admit I have some more reading to do as I found a post elsewhere using the Bit_is_set/Bit_is_clear shortcut as opposed to the & != stuff that made it all happen. I will need to look into how those two statements work better.

Ok off to bed. 3:30am is fast approaching

---

**clawson** - Aug 05, 2013 - 09:34 AM
**Post subject:**

**Quote:**

I know the indentation sucks.

Consider using indent.exe to fix it. It is vital for a C program to be indented correctly. It can raise the legibility by 100%.

As to the design of your code, clearly you only want to make output when the state of PINB.0 changes. For that consider how you can use the ^ operator. It is very good for spotting when a bit is "different from last time".

---

**jgmdesign** - Aug 05, 2013 - 12:47 PM
**Post subject:**

Thanks for the tips Cliff I will look into both.

---

**zu_adha** - Aug 22, 2013 - 05:41 PM
**Post subject:**

did #define made different cycle/mileage/time/memory usage or not?

what about void somthin(void); ?

---

**Koshchi** - Aug 22, 2013 - 06:06 PM
**Post subject:**

What #define?

---

**clawson** - Aug 22, 2013 - 06:14 PM
**Post subject:**

A #define has nothing to do with code generation so I don't see how it could influence cycles or memory usage. Of course after something is #define'd then where that macro is then used I guess it might have some influence on those things. But the answer to "did #define made different.." is "no".

---

**zu_adha** - Oct 05, 2013 - 05:40 PM
**Post subject:**

what is the difference

**Code:**

```
#define CLEARBITMASK(x,y) (x &= (~y))
```

with

**Code:**

```
void CLEARBITMASK(x,y) {x &= (~y)}
```

which the best one?

---

**clawson** - Oct 05, 2013 - 07:05 PM
**Post subject:**

As you've written it there the first is best because wherever you use that preprocessor macro it'll just generate a few opcodes to do the AND operation. For the function version it'll (potentially) add a CALL/RET to each invocation. However if you made the function "static inline" there'd be no such overhead and you may benefit from C

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

parameter type checking (talking of which, your function version forgot to specify types).

---

**abcminiuser** - Oct 05, 2013 - 10:29 PM
**Post subject:**

The macro version looks dangerous - move the inversion outside of the inner braces.

- Dean

---

**zu_adha** - Oct 06, 2013 - 08:02 AM
**Post subject:**

clawson wrote:

As you've written it there the first is best because wherever you use that preprocessor macro it'll just generate a few opcodes to do the AND operation. For the function version it'll (potentially) add a CALL/RET to each invocation. However if you made the function "static inline" there'd be no such overhead and you may benefit from C parameter type checking (talking of which, your function version forgot to specify types).

so in another word
#define is like this

**Code:**

```
#define CLEARBITMASK(x,y) (x &= (~y))

while(1)
{
_delay_us(50);
CLEARBITMASK(1,2);
_delay_ms(10);
CLEARBITMASK(1,3);
}
```

will change to this by uC

**Code:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
while(1)
{
_delay_us(50);
(1 &= (~2)
_delay_ms(10);
(1 &= (~3)
}
```

meanwhile void CLEARBITMASK(int x,int y) {x &= (~y)}

**Code:**

```
while(1)
{
mov pB.1,#11101101b;
CLEARBITMASK(1,2);
mov pB.1,#11101101b;
CLEARBITMASK(1,3);
}
```

will change to

**Code:**

```
while(1)
{
mov pB.1,#11101101b;
acall CLEARBITMASK(1,2);
mov pB.1,#11101101b;
acall CLEARBITMASK(1,3);
}

CLEARBITMASK(int x,int y):
x &= (~y);
ret
```

is that like that?

**Quote:**

The macro version looks dangerous - move the inversion outside of the inner braces.

- Dean Twisted Evil

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

why dangerous? because we need to do "function prototype"?

---

**JohanEkdahl** - Oct 06, 2013 - 08:34 AM
**Post subject:**

**Quote:**

why dangerous? because we need to do "function prototype"?

No, because the y parameter can be any, potentially complex thing.

Assume you have

**Code:**

```
#define CLEARBITMASK(x,y) (x &= (~y))
char a, b, c;
```

and you

**Code:**

```
CLEARBITMASK(a, b&c);
```

This results in the following code actually being passed to the compiler:

**Code:**

```
(a &= (~b&c));
```

If the macro was instead defined as

**Code:**

```
#define CLEARBITMASK(x,y) (x &= ~(y))
```

this would be passed to the compiler

**Code:**

```
(a &= ~(b&c));
```

Quite some difference, right?

I suspect that you do not at all understand what macros are.

They are handled by the C preprocessor (CPP), which goes through the source code and does replacements in it before the compiler goes to work. It is nothing at all like writing e.g. a function and calling it.

There will be more on this in your C textbook. (If not you need to buy a better C textbook.)

---

**zu_adha** - Oct 06, 2013 - 10:01 AM
**Post subject:**

ok tq, i know that, but i miss what Dean's means, because of my english, i coppy-paste frome Dean's tutorial code in 1st page.

so what about the diff. between void and define, am i right with my code up there (i scared that i miss again because of my english, so perhaps ex.code will clear it)

one more question
why i have to used

**Code:**

```
foo &= ~0x01;
```

can i used

**Code:**

```
foo &= 0xFE;
```

i understand exactly why i have to used

**Code:**

```
foo &= ~(1<<2);
```

because if we used

**Code:**

```
foo &= ~(0<<2);
```

because it will nothing change has made if we shift 0. but with

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
foo &= 0xFE;
```

its mean 0xnnnnnnnn AND with 0x11111110

but i found this not work, why?

---

**JohanEkdahl** - Oct 06, 2013 - 10:30 AM
**Post subject:**

**Quote:**

so what about the diff. between void and define

They are just totally different things!

The void is specifying a return value from a function.

The #define is a directive to the preprocessor to modify certain text in the C source before the compilation.

Just because they are in the seemingly corresponding places in your two examples does not mean that they are variants of something. Again, they are totally unrelated.

I still suspect that you do not have any (correct) concept of what the preprocessor is and does, and what a macro is. And again, any decent C textbook will give you the details on such subjects.

**Quote:**

why i have to used

**Code:**

```
foo &= ~0x01;
```

can i used

**Code:**

```
foo &= 0xFE;
```

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

Yes. The outcome should be the same.

**Quote:**

but with

**Code:**

```
foo &= 0xFE;
```

its mean 0xnnnnnnnn AND with 0x11111110

but i found this not work, why?

It should work just fine. If you don't say anything more detailed than "not work" then we can not say anything more detailed either.

---

**zu_adha** - Oct 06, 2013 - 11:00 AM
**Post subject:**

**Quote:**

It should work just fine. If you don't say anything more detailed than "not work" then we can not say anything more detailed either.

ops, i found my code seems like something false (typoo) its supposed to 0bNNNNNNNN instead of 0xNNNNNNNN..my bad, it works just fine..

umm one more thing...

this is has same cycle or not?

**Code:**

```
PORTB |= 0x00001111;
```

with

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

**Code:**

```
PORTB |= (1<<1);
PORTB |= (1<<2);
PORTB |= (1<<3);
PORTB |= (1<<4);
```

i guess the 2nd one have more cycle

---

**JohanEkdahl** - Oct 06, 2013 - 11:04 AM
**Post subject:**

**Quote:**

i guess the 2nd one have more cycle

Stop guessing. Look at the generated code. Get the timings for the individual instructions in the "AVR Instruction Set" document available at www.atmel.com .

---

**zu_adha** - Oct 06, 2013 - 11:16 AM
**Post subject:**

JohanEkdahl wrote:

**Quote:**

i guess the 2nd one have more cycle

Stop guessing. Look at the generated code. Get the timings for the individual instructions in the "AVR Instruction Set" document available at www.atmel.com .

ok, tq for the link

i guess because i read in forum that

**Quote:**

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

```
PORTB |= 0b00001111;
```

means PORT B is OR with 0b00001111;
and

**Quote:**

```
PORTB |= (1<<1);
```

is shift 1 to the left once and OR to the PORT B, so thats the reason why i guess it has more cycle because every change the bit need to shift the bit one by one and do OR operation for each bit.

---

**JohanEkdahl** - Oct 06, 2013 - 11:37 AM
**Post subject:**

**Quote:**

i guess it has more cycle

You're still guessing. The facts are available to you - if you study the generated assembly/machine code.

---

**clawson** - Oct 06, 2013 - 12:44 PM
**Post subject:**

Can you take this discussion to a separate thread please? You aren't really giving feedback about the 101 post but asking basic questions about your misunderstanding of the C language.

---

**zu_adha** - Oct 07, 2013 - 08:30 AM
**Post subject:**

no need clawson it is done, thank you for the link btw...

---

Amol Gulhane
Robolab Technologies,
Pune - 411005 (India)
#Tel: +91 879 310 2996
E-mail: amol.robolab@gmail.com

All times are GMT + 1 Hour
Powered by ***PN*phpBB2** © 2003-2006 The PNphpBB Group
Credits