Basic Bitwise Operators + (Part 1) – Tutorial #8

T.K. HAREENDRAN

AVR tutorial

Share this:

In this part I am going to shed some light on the background of the coding process. The first attempt is to explain the basic bitwise operators available in the C++ language. For better explanation I will express most integer values using binary notation, also known as base two (in the traditional decimal system, the base is ten). In this system, all integer values use only the values 0 and 1 for each digit.Each 0 or 1 digit is called a bit, which is the short form of binary digit.

- In Decimal System (base 10), a number like 453 means $\mathbf{4\times10^2 + 5\times10^1 + 3\times10^0}$.
- In Binary System (base 2), a number like 1101 means $\mathbf{1\times2^3 + 1\times2^2 + 0x2^1 + 1\times2^0}$ (ie 8+4+1 = 13). Remember that any number to the power zero equals one!

The binary scale is a method of writing numbers using only 2 figures, 0 and 1, instead of the decimal scale 0 to 9. Whereas decimal numbers are formed on the base 10 so that each number position has ten times the value of the adjacent position on the right, binary numbers are formed on the base 2 so that each number position has two times the value of the adjacent position to the right. Note that most C++ compilers do not have any means of expressing binary numbers directly in source code. Some allow the prefix 0b followed by a series of 0 and 1 digits.

Bitwise AND

The bitwise AND operator in C++ is a single **&**, used between two other integer expressions. Here, if both input bits are 1, the resulting output is 1, otherwise the output is 0.

- **0 & 0 == 0**
- **0 & 1 == 0**
- **1 & 0 == 0**
- **1 & 1 == 1**

Bitwise OR

The bitwise OR operator in C++ is the vertical bar symbol **|**. The bitwise OR of two bits is 1 if either or both of the input bits is 1, otherwise it is 0.

- **0 | 0 == 0**
- **0 | 1 == 1**
- **1 | 0 == 1**
- **1 | 1 == 1**

Bitwise NOT

The bitwise NOT operator in C++ is the tilde character **~**. Bitwise NOT changes each bit to its opposite: 0 becomes 1, and 1 becomes 0.

- **~0 == 1**
- **~1 == 0**

Bitwise XOR

There is a somewhat unusual operator in C++ called bitwise exclusive OR, also known as bitwise XOR, written using the caret symbol **^**.
This operator is often used to toggle (i.e. change from 0 to 1, or 1 to 0) some of the bits in an integer expression while leaving others alone. This operator is very similar to the bitwise OR operator, only it evaluates to 0 for a given bit position when both of the input bits for that position are 1:

- **0 ^ 0 == 0**
- **0 ^ 1 == 1**
- **1 ^ 0 == 1**
- **1 ^ 1 == 0**

Assignment Operators

Often in programming, you want to operate on the value of a variable x and store the modified value back into x. In most programming languages, for example, you can increase the value of a variable x by 3 using the code:

```
x = x + 3; // increase x by 3
/* or you can write it like this: */
x += 3; // increase x by 3
```

Bit Shift Operators

There are two bit shift operators in C++: the **left shift operator <<** and the **right shift operator >>**. These operators drive the bits in the left operand to be shifted left or right by the number of positions specified by the right operand. In short, an expression like **a<<b** will be evaluated as the binary representation of "a" will be shifted "b" times to the left, and the value is getting larger.

C Code	Decimal Value	Bit Representation
1<<2	4	00000100
1<<3	8	00001000

Then what about **4>>1**?

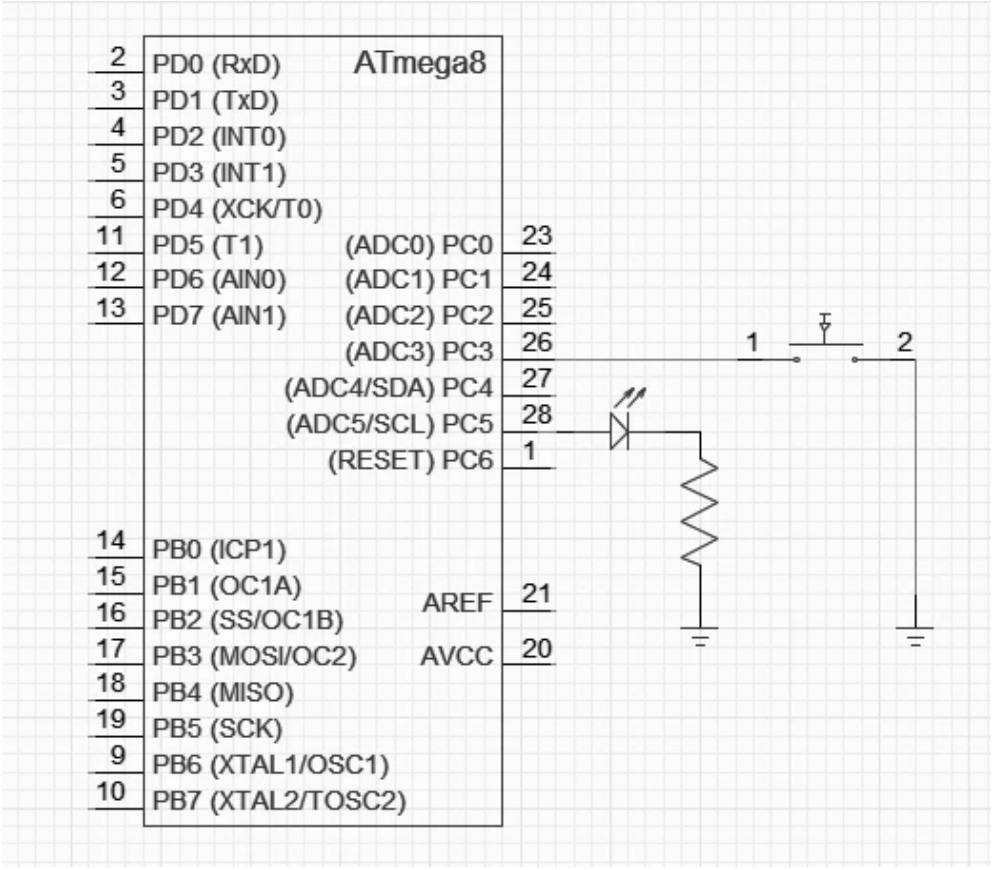
Here the binary representation of 4 will be shifted 1 time to the right, ie

- **4 → 00000100**
- **4>>1 = 00000010** (the decimal value is 2)

Note: Don't think that the bitwise operator in C++ is same as the boolean operator. Remember, the bitwise AND operator (**&**) is not the same as the boolean AND operator (**&&**). I will explain this in the forthcoming part.

	Bitwise	Boolean
AND	&	&&
OR	|	||
XOR	^	!=
NOT	~	!

Following is a simple practical test. I just wired a push button switch on PC3, and one LED on PC5 of ATMega8 as shown in the diagram. I want to turn off the LED when push button is pressed. Try to write the code snippet for this task. Helpful indicators are given in the template (answer will be published on next part)!




```
int main(void) {
   init();
   //Set the Data Direction Register for the LED to output
   ................................;
   //Set the Data Direction Register for the push button switch to input
   ................................;
   //Enable the pullup resistor for the push button switch
   ................................;
   while (1) {
      if ( ….............................. )
         PORTC|=( ….............................. );
      else
         PORTC&=~( ….............................. );
   }
}
```

→ Part 9: **Basic Bitwise Operators + (Part 2)**
← Part 7: **ATmega8 – Basic Input/Output Interfacing**