Bitwise Operators in C- Embedded Systems Context

A. Mujeeb · Follow

7 min read · Sep 21, 2023

Embedded Programming Series

Prerequisite:

In order to understand this article, you should have a good working knowledge of C language, and familiarity with the binary number system, including the hexadecimal format.

Table of Contents

1. Bitwise Operators in C

Bitwise operators are extensively used in embedded systems programming. Before we understand the bitwise operations in the context of embedded systems, lets familiarize with some fundamentals bitwise operators in C language.

There are four fundamental bitwise operators in C language as shown below. AND, OR, and XOR are binary operators, which require two bits. The NOT is a unary operators, i.e. it operates on

one bit only.

Operator	Meaning	Example
AND (&)	Bitwise AND	1111 & 1010 = 1011
OR (|)	Bitwise OR	1011 | 1011 = 1111
Not (~)	Complement	~1011 = 0100
XOR (^)	Exclusive OR	1011 ^ 1100 = 0111

2. READING an Input Bit

The bitwise operators are applied to **integer** or **character** type data elements in C. A character size is 8-bits and the integer size is typically 32-bits (or 64-bits).

Example

Consider an embedded system, with three switches. A C-language function reads the status of the three switches, and returns them as the three LSBs of character type variable.

Step 1:

First step is to create a “mask” for each bit position. A mask is a sequence of bits with 1 in the desired bit position, and 0s in all other position.

For example:

Mask for bit 0 is “0000–0001”
Mask for bit 1 is “0000–0010” and so on.

Step 2:

Apply AND operation between the input data sequence and the mask. If the resulting vlaue is non-zero, then the particular bit is ON, otherwise it must be OFF.

Following code demonstrates by implementing above steps how to find if a particular switch is ON or OFF?

```
#include <stdio.h>

char readSwitches(void)
{
    //Change the vlaues to play with different options
    /* switch 0 = ON    (0th bit)
       switch 1 = OFF   (1st bit)
       switch 2 = ON    (2nd bit)
    */
```

```
    char switchValues = 0x05;   //1111-0101
    return switchValues;
}


void main(void)
{
    char mask0 = 0x01;  /* mask for bit 0: 0000-0001 */
    char mask1 = 0x02;  /* mask for bit 2: 0000-0010 */
    char mask2 = 0x04;  /* mask for bit 3: 0000-0100 */

    char inputBits = readSwitches();

    /* check for bit 0 (swtich 0) */
    if ((mask0 & inputBits) == 0)
    {
      printf("Switch 0 is OFF\n");
    }else
    {
      printf("Switch 0 is ON\n");
    }

     /* check for bit 1 (swtich 1) */
    if ((mask1 & inputBits) == 0)
    {
      printf("Switch 1 is OFF\n");
    }else
    {
      printf("Switch 1 is ON\n");
    }

     /* check for bit 2 (swtich 2) */
    if ((mask2 & inputBits) == 0)
    {
      printf("Switch 2 is OFF\n");
    }else
    {
      printf("Switch 2 is ON\n");
    }


    printf("\nProgram ends.");
}
```

3. Setting and Resetting Output Bits

Writing either 0 or 1 to a particular bit is similar to reading a bit value, in that a mask is created in both cases. Following are the main steps to write either 1 or 0 to a particular bit value in a char or integer data type.

Example

Consider an embedded system, with three switches. A C-language function writes the value to particular bit position(s) for three switches represented by three LSBs of character variable.

Step 1: Create masks for each bit position

switch 0: Set Mask =(0000–0001), Reset Mask = (1111–1110)
switch 1: Set Mask =(0000–0010), Reset Mask = (1111–1101)
switch 2: Set Mask =(0000–0100), Reset Mask = (1111–1011)

switch 1 and 2: Set Mask = (0000–0011), Reset Mask = (1111–1100)

Step 2: Apply bitwise operation on output data using the following masks

Bitwise operation for SET is = OR
Bitwise Operation for RESET = AND

When a mask is applied to an output byte, only the bits specified in the mask are updated, and the other bits remain UNCHANGED.

Step 3: Merge masks or reset masks if more than one bit is to be set or reset

If we need to SET two bits simultaneously (e.g.bit-0 and bit-2), then merge both masks by ***using OR operation*** on masks. (see the example code below)
If we need to RESET two bits simultaneously (e.g. bit-0 and bit-2), then merge RESET masks for both using AND operation.

```
#include <stdio.h>

/* Function to SET particular bits in data */
char setSwitches(char originalVals, char mask)
{
    char updatedValue = originalVals | mask;
    return updatedValue;
}

/* Function to RESET particular bits in data */
char resetSwitches(char originalVals, char mask)
{
    char updatedValue = originalVals & mask;
    return updatedValue;
}


/* Test Program */
void main(void)
{
    char switchVals;
    char updatedSwitchVals;
```

```
    char setMask0 = 0x01;  /* mask for bit 0: 0000-0001 */
    char setMask1 = 0x02;  /* mask for bit 2: 0000-0010 */
    char setMask2 = 0x04;  /* mask for bit 3: 0000-0100 */

    /* We do not need to create RESET masks, because they can be created by complement oper


    //Original values are following...change to play with different options
    /* switch 0 = OFF    (0th bit)
       switch 1 = ON   (1st bit)
       switch 2 = OFF    (2nd bit)
    */
    switchVals = 0x02;

    /* SET bit-0 using setMask0*/
    updatedSwitchVals = setSwitches(switchVals, setMask0);
    printf("\nSET Bit0: Original vals = 0x%0x, and updated values = 0x%0x", switchVals, upd

    /* SET bit-1 using setMask1*/
    updatedSwitchVals = setSwitches(switchVals, setMask1);
    printf("\nSET Bit1: Original vals = 0x%0x, and updated values = 0x%0x", switchVals, upd

    /* SET bit-2 using setMask2*/
    updatedSwitchVals = setSwitches(switchVals, setMask2);
    printf("\nSET Bit2: Original vals = 0x%0x, and updated values = 0x%0x", switchVals, upd

    /* Setting bit-0 and bit-2 in one operation: Combbine both masks using OR operation*/
    updatedSwitchVals = setSwitches(switchVals, setMask0 | setMask2);
    printf("\nSET Bit 0 and 2: Original vals = 0x%0x, and updated values = 0x%0x", switchVa


    /* RESET bit-1 using RESET Mask1
        RESET mask is obtained by inverting the SET mask
    */
    switchVals = 0x03;
    updatedSwitchVals = resetSwitches(switchVals, ~setMask0);
    printf("\nRESET Bit0: Original vals = 0x%0x, and updated values = 0x%0x", switchVals, u

    /* multiple bit are reset. combine SET masks and invert themto get RESET mask
       Then apply the mask using OR operation as explained in the theory above
    */
    switchVals = 0x07;
    updatedSwitchVals = resetSwitches(switchVals, ~(setMask0 | setMask2));
    printf("\nRESET Bit0 and bit3: Original vals = 0x%0x, and updated values = 0x%0x", swit

    printf("\nProgram ends.");

}
```

Following is the output of the above program.

SET Bit0: Original vals = 0x2, and updated values = 0x3

SET Bit1: Original vals = 0x2, and updated values = 0x2

SET Bit2: Original vals = 0x2, and updated values = 0x6
SET Bit 0 and 2: Original vals = 0x2, and updated values = 0x7
RESET Bit0: Original vals = 0x3, and updated values = 0x2
RESET Bit0 and bit3: Original vals = 0x7, and updated values = 0x2

4. Toggling Output Bits

Besides SET and RESET, another common operation on bits is TOGGLE operation, which means to change the state of a bit. If the value of a bit is 0, it is changed to 1. If the value of bit is 1, it is changed to 0.

XOR operation is used to toggle a particular bit in a sequence, just the same way we used AND and OR operations for setting and resetting bits.

Remember that if both bits are same, the value is unchanged in an XOR operation. But if the values are different, the values is set to 1.

Hence the mask at the desired position is set to 1. If the data value at that position is 0, it will be changed to 0, and if data value is 1 at the mask position, it will be changed to 1.

If Data value = 0, Mask value is 1: Result is 1 (i.e. Toggling of data value)
If Data value = 1, Mask value is 1: Result is 0 (i.e. Toggling of data valiue)

For Toggling:

Mask: 1, For example, 0000–0001 (to toggle LSB bit, i.e. bit-0).
Bitwise Operator = XOR

```
#include <stdio.h>

/* Function to TOGGLE  particular bits in data */
char toggleSwitch(char originalVals, char mask)
{
    char updatedValue = originalVals ^ mask;  /* XOR operator */
    return updatedValue;
}


/* Test Program */
void main(void)
{
    char switchVals;
    char updatedSwitchVals;

    /* Toggle operation mask...same as SET operation mask */
    char setMask0 = 0x01;  /* mask for bit 0: 0000-0001 */
    char setMask1 = 0x02;  /* mask for bit 2: 0000-0010 */
```

```
    char setMask2 = 0x04;  /* mask for bit 3: 0000-0100 */


    /* Test data: All three switches are ON initially... 0000-0111 */
    switchVals = 0x07;

    /* Toggle bit-0 using setMask0... SET and TOGGLE use same masks, but different bitwise
    updatedSwitchVals = toggleSwitch(switchVals, setMask0);
    printf("\nTOGGLE Bit-0: Original vals = 0x%0x, and updated values = 0x%0x", switchVals,

    /* Toggle bit-1 using setMask1... SET and TOGGLE use same masks, but different bitwise
    switchVals = 0x07;
    updatedSwitchVals = toggleSwitch(switchVals, setMask1);
    printf("\nTOGGLE Bit-1: Original vals = 0x%0x, and updated values = 0x%0x", switchVals,

    /* Toggle bit-1 and bit-2-- Merge both masks by using AND operation (just like set oper
    switchVals = 0x07;
    updatedSwitchVals = toggleSwitch(switchVals, (setMask1 | setMask2) );
    printf("\nTOGGLE Bit-1 and 2: Original vals = 0x%0x, and updated values = 0x%0x", switc

    printf("\nProgram ends");
```

Following is the output of the above program.

TOGGLE Bit-0: Original vals = 0x7, and updated values = 0x6

TOGGLE Bit-1: Original vals = 0x7, and updated values = 0x5

TOGGLE Bit-1 and 2: Original vals = 0x7, and updated values = 0x1

5. Summary

In this article, bitwise operators in C language are covered in the context of embedded systems, where bits represents input and output devices.

Common bitwise operations on I/O devices include:

READ, SET, RESET, TOGGLE.

Each Operation has a certain MASK and Bitwise operaton

i) READ: Mask = 1 (at desire positions), and Operation = AND

ii) SET: Mask = 1, and Operation = OR

iii) RESET: Mask= 0, and Operation = AND

iv) TOGGLE Mask = 1, and Operation = XOR

Multiple bits can be updated or read by combining the masks.