Arbeiten mit Bits

Einleitung

Bits sind die kleinste Dateneinheit in elektronischen Rechenmaschine, d.h. Computern. Diese binären, logischen Elemente können entweder «true» oder «false» gesetzt sein, wobei «true» meist als «eingeschaltet» oder «1» und «false» als «ausgeschaltet» oder «0» verstanden wird. Wir wissen, dass 8 Bits zu einem Byte zusammengefasst werden können und mehrere Bytes dann die höherwertigen Datentypen oder –strukturen ausmachen.

Vor allem beim Arbeiten mit Mikrokontrollern oder in der Steuerungstechnik kommt es häufig vor, dass einzelne Bits manipuliert werden müssen.

Bitoperatoren

In C sind Operatoren definiert, um Bits zu manipulieren. Es sind dies:

&	bitwise AND		
|	bitwise OR		
^	bitwise XOR		
~	bitwise complement		
<<	shift left	data << n	um «n» nach links
>>	shift right	byte >> n	um «n» nach rechts

Die folgende Wahrheitstabelle gibt Auskunft über die Funktionalität:

a	b	a & b	a | b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Beispiel: Ist das Bit a = 1 und Bit b = 0, dann ist a&b = 0, a|b = 1, a^b = 1, und das complemen vom a, a~ = 0.

Achtung: Bitte diese Operatoren nicht den logischen Operatoren verwechseln:

&	ist nicht gleich	&&	
|	ist nicht gleich	||	und so weiter…

Schiebeoperatoren:

Mit den Schiebeoperatoren << und >> wird die Bitstruktur z.B. eines Bytes nach links oder rechts geschoben. Bits am linken oder rechten Ende «fallen über die Klippe», das heisst, die Information ist verloren und kann durch zurückschieben nicht wiederhergestellt werden.

Beispiel:

12	=	00001100	
		00001100 << 1	
	=	00011000	→ die zwei Bits wurden um eine Position nach links geschoben

Aufgaben

1. Welcher Dezimalzahl entspricht 00011000, das Resultat des obigen Beispiels?
   Mit Rechtsschieben um 1 Position der anfänglichen «12» würden wird 00000110 erhalten.
   Welches Muster stellst Du fest?

2. Schreibe ein kleines Programm, das folgende Funktionen erfüllt:
   a. über die Konsole wird eine (ganzzahlige) Zahl eingelesen, und zwar wird der User nach dem Start dazu aufgefordert. -> Überprüfen, dass es sich um eine Zahl handelt.
   b. Ist eine valide Zahl erfasst, muss der User eingeben, ob er die Zahl nach links oder nach rechts schieben möchte.
   c. Danach muss der User eingeben, wie viele Positionen geschoben werden soll.
   d. Danach wird die Operation durchgeführt und folgende Ausgabe produziert: «'Zahl' bitweise nach 'links' geschoben ergibt 'Ausgabe'».

3. Schreibe ein Modul «bitOps», das die folgenden Spezifikationen erfüllt:

bitOps.c	Implementierungsfile
bitOps.h	Headerfile
main.c	Mainfile zum Testen

   bitOps soll folgende Funktionen enthalten:

   void setBitNInByte(unsigned char* data, unsigned char bitN);
   → setzt das n-te Bit auf "1".

   void clearBitNInByte(unsigned char* data, unsigned char bitN);
   → setzt das n-te Bit auf "0"

   void toggleBitNInByte(unsigned char* data, unsigned char bitN);
   → wechselt das n-te Bit. Falls es «1» ist, dann wird es «0» gesetzt, oder umgekehrt.

   unsigned char BitNIsSet(unsigned char data);
   → falls das n-te Bit gesetzt ist, gibt die Funktion «1» zurück, sonst «0».

   Teste im main.c die folgenden Use Cases:
   Setze in der HEX-Zahl 0xD3 die Bit2, 3, und 5. Welchen Wert erhälst Du?
   Lösche in der HEX-Zahl 0xD3 die Bits4, 6, und 7. Welchen Wert erhälst Du?
   Toggle in der HEX-Zahl 0xD3 das Bit 1, 4, und 6. Welchen Wert erhälst Du?

4. Entwickelt eigene Testfälle für das Modul bitOps. Danach tauscht ihr Euer bitOps Modul mit einer Kollegin/Kollegen aus und versucht die gleichen Testfälle durchzuführen.