9.9 Funktionen zur Bit-Manipulation

Um zum Beispiel die Register der seriellen Schnittstelle RS232 (V24) mit korrekten Werten zu initialisieren oder den Status der Schnittstelle abzufragen sowie ein Byte zu senden oder zu empfangen ist es notwendig die Funktionen zur Bit-Manipulation zu verstehen und sicher anwenden zu können.

Gambas stellt Ihnen Funktionen zur Bit-Manipulation zur Verfügung:

- Ein bestimmtes Bit in einer Bit-Folge zu setzen, zu löschen oder zu invertieren
- Testfunktion um festzustellen, welchen Wert ein bestimmtes Bit in einer Bit-Folge besitzt
- Funktionen zur Verschiebung jedes Bits (links/rechts) in einer Bit-Folge
- Funktionen zur zyklischen Verschiebung oder Rotation jedes Bits (links/rechts) in einer Bit-Folge

9.9.1 Übersicht zu den Funktionen zur Bit-Manipulation

Für alle Funktionen in der folgenden Tabelle gelten diese Hinweise für die Argumente *Number* und *Bit*:

- Der Daten-Typ von *Number* kann Byte, Short, Integer oder Long sein.
- Wenn der Daten-Typ für das Argument *Number* in den Funktionen zur Bit-Manipulation nicht explizit angegeben wird, so nimmt der Compiler für *Number* als (Standard-)Daten-Typ 'Integer' an oder 'Long', wenn die Zahl zu groß ist oder 'Float', wenn das Argument *Number* nicht vom Daten-Typ 'Long' sein kann.
- Der Daten-Typ des Funktionswertes entspricht dem Daten-Typ des Arguments *Number*.
- Ein Bit gilt als gesetzt, wenn es den Wert 1 hat.
- Der gültige Bereich der Bits B_k hängt vom Daten-Typ des Arguments *Number* ab. Dabei gilt:

```
Daten-Typ     Bit-Bereich
-------------------------
Byte          B0...B7
Short         B0...B15
Integer       B0...B31
Long          B0...B63
```

Funktion	Beschreibung
BClr (Number , Bit)	Gibt Number mit gelöschtem Bit 'Bit' zurück.
BSet (Number , Bit)	Gibt Number mit gesetztem Bit 'Bit' zurück.
BTst (Number , Bit)	Gibt *True* zurück, wenn das Bit 'Bit' gesetzt ist, sonst False.
BChg (Number , Bit)	Gibt Number zurück, dessen Bit 'Bit' invertiert wurde.
Lsl (Number , Bit)	Jedes Bit in der Bitfolge von Number wird um 'Bit' Bits nach links geschoben. Für die links wegfallenden Bits werden rechts Null-Bits an gehangen. Das Vorzeichen wird ignoriert. (Lsl = logical shift left)
Lsr (Number , Bit)	Jedes Bit in der Bitfolge von Number wird um 'Bit' Bits nach rechts geschoben. Für die rechts wegfallenden Bits werden links Null-Bits eingefügt. (Lsr = logical shift right)
Shl (Number , Bit)	Jedes Bit in der Bitfolge von Number wird um 'Bit' Bits nach links geschoben. Für die links wegfallenden Bits werden rechts Null-Bits an gehangen.
Asl (Number , Bit)	Synonym für Shl (Number , Bit). Das Vorzeichen wird nicht ignoriert.
Shr (Number , Bit)	Jedes Bit in der Bitfolge von Number wird um 'Bit' Bits nach rechts geschoben. Für die rechts wegfallenden Bits wird links das Vorzeichen-Bit eingefügt.
Asr (Number , Bit)	Synonym für Shr (Number , Bit)
Rol (Number , Bit)	Bei der Operation Rol *rotieren* die Bits um die Anzahl 'Bit' so, als ob MSB (most significant bit - höchstwertige Bitposition) und LSB (least significant bit - niedrigstwertige Bitposition) miteinander verbunden wären. Das Bit, das aus der Bit-Folge nach links hinausgeschoben wird hat den selben Wert wie das Bit, das von rechts hinein geschoben wird. (Rol = Rotate left)
Ror (Number , Bit)	Bei der Operation Ror *rotieren* die Bits um die Anzahl 'Bit' so, als ob MSB und LSB miteinander verbunden wären. Das Bit, das aus der Bit-Folge nach rechts hinausgeschoben wird hat den selben Wert wie das Bit, das von links hinein geschoben wird.

Tabelle 9.9.1.1: Übersicht zu den Funktionen zur Bit-Manipulation

9.9.2 Beispiele Bit-Manipulation

Für alle o.a. Funktionen zur Bit-Manipulation wird je ein Beispiel angegeben:

```
Print 23; " -> "; Bin(CByte(23), 8); " BCLR(CByte(23),2) "; Bin(BClr(CByte(23), 2), 8); " ->> "; BClr(CByte(23), 2)
Print 23; " -> "; Bin(CByte(23), 8); " BSET(CByte(23),3) "; Bin(BSet(CByte(23), 3), 8); " ->> "; BSet(CByte(23), 3)
Print 23; " -> "; Bin(CByte(23), 8); " BCHG(CByte(23),1) "; Bin(BChg(CByte(23), 1), 8); " ->> "; BChg(CByte(23), 1)
Print 23; " -> "; Bin(CByte(23), 8); " ->> "; IIf(BTst(CByte(23), 2), "Das Bit B2 ist gesetzt!", "Das Bit B2 ist 0.")
Print
Print 23; " -> "; Bin(CByte(23), 8); " LSL(CByte(23),3) "; Bin(Lsl(CByte(23), 3), 8); " ->> "; Lsl(CByte(23), 3)
Print 23; " -> "; Bin(CByte(23), 8); " LSR(CByte(23),3) "; Bin(Lsr(CByte(23), 3), 8); " ->> "; Lsr(CByte(23), 3)
Print
Print +23; "  -> "; Bin(CShort(23), 16); " SHL(CShort(23),3) "; Bin(Shl(CShort(23), 3), 16); " ->> "; Shl(CShort(23), 3)
Print -23; " -> "; Bin(CShort(-23), 16); " SHR(CShort(-23),3) "; Bin(Shr(CShort(-23), 3), 16); " ->> "; Shr(CShort(-23), 3)
Print
Print 23; " -> "; Bin(CShort(23), 16); " ROL(CShort(23),3) "; Bin(Rol(CShort(23), 3), 16); " ->> "; Rol(CShort(23), 3)
Print 23; " -> "; Bin(CShort(23), 16); " ROR(CShort(23),3) "; Bin(Ror(CShort(23), 3), 16); " ->> "; Ror(CShort(23), 3)
Print
Print 23; " -> "; Bin(CByte(23), 8); " NOT 23 "; Bin(Not CByte(22), 8); " ->> "; Not 23
```

Ausgabe in der Konsole der IDE:

```
23 -> 00010111 BCLR(CByte(23),2) 00010011 ->> 19
23 -> 00010111 BSET(CByte(23),3) 00011111 ->> 31
23 -> 00010111 BCHG(CByte(23),1) 00010101 ->> 21
23 -> 00010111 ->> Das Bit B2 ist gesetzt!

23 -> 00010111 LSL(CByte(23),3) 10111000 ->> 184
23 -> 00010111 LSR(CByte(23),3) 00000010 ->> 2

23  -> 0000000000010111 SHL(CShort(23),3) 0000000010111000 ->> 184
-23 -> 1111111111101001 SHR(CShort(-23),3) 1111111111111101 ->> -3

23 -> 0000000000010111 ROL(CShort(23),3) 0000000010111000 ->> 184
23 -> 0000000000010111 ROR(CShort(23),3) 1110000000000010 ->> -8190

23 -> 00010111 NOT 23 11101001 ->> -24
```

9.9.3 Simultane Bit-Manipulationen

Für die Funktionen BClr(*Number, Bit*), BSet(*Number, Bit*), BChg(*Number, Bit*) und BTst(*Number, Bit*) in der oben aufgeführten Tabelle 9.9.1.1 gilt:

```
Es wird genau ein Bit geändert oder dessen (Bit-)Wert abgefragt.
```

In einigen Programmiersprachen existieren neben den logischen Operatoren wie AND oder OR oder NOT auch Bit-Operatoren. Da Gambas keine speziellen (logischen) Bit-Operatoren kennt, können Sie die logischen Operatoren auf Operanden anwenden, die als Zahlen vom Daten-Typ Byte, Short, Integer oder Long vorliegen. Damit ist es Ihnen möglich, mehrere Bits in einer Bit-Folge gleichzeitig (simultan) zu setzen oder zu löschen oder zu invertieren.

9.9.3.1 Aufgabe 1 – Simultanes Setzen von ausgewählten Bits in einer Bit-Folge

Das Bit *B2* sowie das Bit *B5* im Operanden $(10011011)_{bin} = 155_{dez}$ sollen (simultan) gesetzt werden. Alle anderen Bits dagegen behalten ihren Wert! Zuerst muss die Bit-Maske – vom gleichen Daten-Typ wie der Operand – ermittelt werden, die an den zu *setzenden* Bit-Positionen den Wert 1 hat und sonst 0. Dann ist durch die Anwendung des ODER-Operators mit *operand OR mask* gesichert, das nur die beiden Bits B2 und B5 gesetzt werden:

```
operand :    (10011011)bin  = (155)dez
mask    :    (00100100)bin  = (36)dez
result  :    (10111111)bin  = (191)dez
```

Um die erforderlichen Bit-Masken zu erzeugen ist es von Vorteil, wenn die Operatoren als binäre Zahlen vorliegen. Da jedes Bit des Operanden mit jedem Bit der Bit-Maske mit den logischen Operatoren verknüpft wird, ist die Kenntnis der folgenden Tabellen hilfreich:

```
A      B      A AND B
----------------------
0      0        0
0      1        0
1      0        0
1      1        1
```

```
A      B      A OR B
--------------------
0      0        0
0      1        1
1      0        1
1      1        1
```

```
A      B      A XOR B
----------------------
0      0        0
0      1        1
1      0        1
1      1        0
```

```
A      NOT A
-------------
0        1
1        0
```

9.9.3.2 Aufgabe 2 – Simultanes Löschen von ausgewählten Bits in einer Bit-Folge

Das Bit *B3* und Bit *B4* im Operanden $(10011011)_{bin} = 155_{dez}$ sollen gelöscht werden, während alle anderen Bits ihren Wert nicht verändern.

Durch den Einsatz des AND-Operators mit *operand AND mask* werden nur die beiden Bits B3 und B4 gelöscht, wenn Sie eine Bit-Maske *mask benutzen*, die an den zu *löschenden* Bit-Positionen den Wert 0 hat und sonst 1.

```
operand :    (10011011)bin  = (155)dez
mask    :    (11100111)bin  = (231)dez
result  :    (10000011)bin  = (131)dez
```

9.9.3.3 Aufgabe 3 – Simultanes Invertieren von ausgewählten Bits in einer Bit-Folge

Die vier Bits *B0 bis B3* im Operanden $(10011011)_{bin} = 155_{dez}$ sollen invertiert werden und alle anderen Bits behalten ihren Wert. Die passende Bit-Maske hat an den zu *invertierenden* Bit-Positionen den Wert 1 hat und sonst 0. Damit ist durch *operand XOR mask* gesichert, das nur die ersten 4 niederwertigen Bits invertiert werden:

```
operand :    (10011011)bin  = (155)dez
mask    :    (00001111)bin  = (15)dez
result  :    (10010100)bin  = (148)dez
```

9.9.3.4 Aufgabe 4 – Simultanes Invertieren aller Bits in einer Bit-Folge

Alle Bits im Operanden $(10011011)_{bin} = 155_{dez}$ sollen invertiert werden. Mit *operand XOR mask* gesichert, das alle Bits invertiert werden, wenn die Bit-Maske an *allen* Bit-Positionen den Wert 1hat:

```
operand :    (10011011)bin  = (155)dez
mask    :    (11111111)bin  = (255)dez
result  :    (01100100)bin  = (100)dez
```

Eine schnelle *Alternative* bietet sich mit dem NOT-Operator an, da beim bitweisen NOT eine logische Negation jedes einzelnen Bits durchgeführt wird. Jede 1 wird durch 0 und jede 0 durch eine 1 ersetzt:

```
NOT (155) = 100
NOT (10011011)  =  01100100
```

Zusammenfassung:

- operand OR mask: Die Bits, die in *mask* 1 sind, werden in *operand* gesetzt (auf 1 gesetzt)
- operand AND mask: Die Bits, die in *mask* 0 sind, werden in *operand* gelöscht (auf 0 gesetzt)
- operand XOR mask: Die Bits, die in *mask* 1 sind, werden in *operand* invertiert.
- NOT operand: Alle Bits in *operand* werden invertiert.

Für die Erprobung der Aufgaben 1 bis 4 wurden die Funktion *BitManipulation(bOperand As Byte, sMode As String, Optional sMask As Byte[])* eingesetzt, in der Sie die Operatoren AND, OR, XOR und NOT verwenden können und eine Beispiel-Prozedur:

```
Public Function BitManipulation(bOperand As Byte, sOperation As String, Optional sMask As Byte[]) As Byte
  Dim bBitMask, bElement As Byte
  Dim k As Integer

  If sMask Then
     sMask.Reverse()
     For k = 0 To 7
       bBitMask = bBitMask + sMask[k] * 2 ^ k
     Next
  Endif

  Select Case Upper(sOperation)
    Case "NOT"
      Return Not bOperand
    Case "AND"
      Return bOperand And bBitMask
    Case "OR"
      Return bOperand Or bBitMask
    Case "XOR"
      Return bOperand Xor bBitMask
  End Select

End ' BitManipulation(..)

Public Sub btnClearBits_Click()
  Print ZahlToDezimal("10011011", 2) ' Zur Kontrolle: Zahl, Basis
  Print ZahlToDezimal("11100111", 2)
  Print Bin(BitManipulation(155, "AND", [1, 1, 1, 0, 0, 1, 1, 1]), 8)
  Print ZahlToDezimal("10000011", 2)
End ' btnClearBits_Click()
```