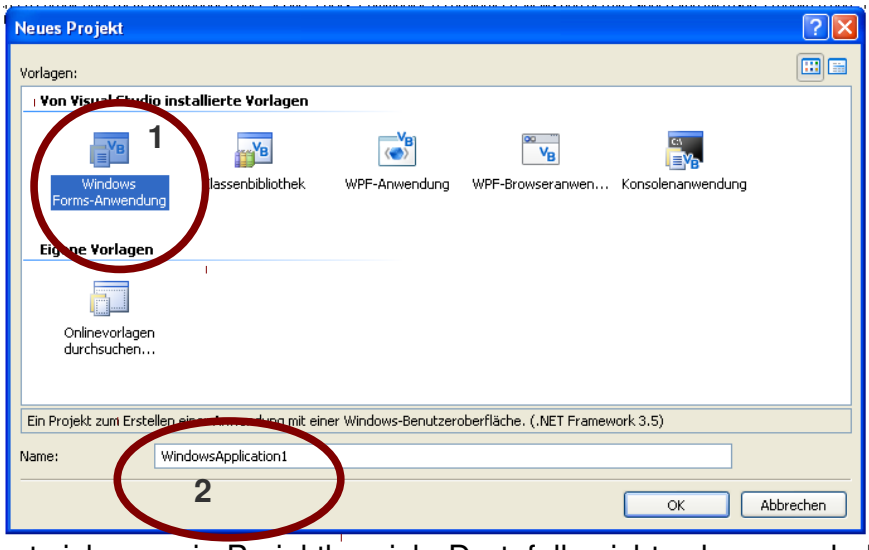
Anleitung zur Ansteuerung des Parallelport mit Visual Basic 2008 Express Edition

1. Einrichten des Systems

1. Visual Basic von http://www.microsoft.com/germany/express/product/visualbasicexpress.aspx herunterladen und installieren.
   Eine (kostenlose) Registrierung ist nach einiger Zeit notwendig!
2. Die Datei inpout32.dll von http://logix4u.net/Legacy_Ports/Parallel_Port/Inpout32.dll_for_Windows_98/2000/NT/XP.html herunterladen. Die Datei befindet sich im Ordner \inpout32_source_and_bins\binaries\Dll\
3. Die Datei inpout32.dll in \windows\system kopieren.
   Achtung: Fremde Dateien können das System beschädigen!

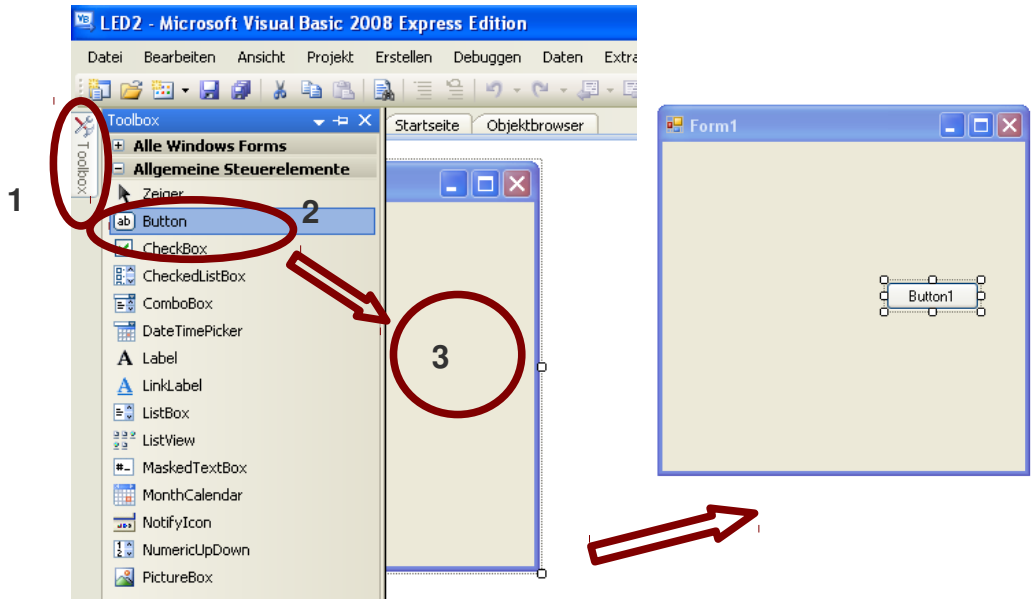
2. Ein erstes Programm in Visual Basic

1. Starten sie Visual Basic
2. Neues Projekt erstellen: Datei → Neues Projekt …
   Dort *Windows Forms-Anwendung* auswählen (1) und entsprechend benennen (2).



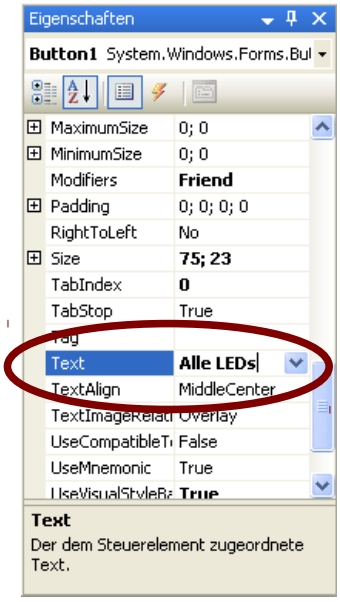
3. Es öffnet sich nun ein Projektbereich. Dort, falls nicht schon geschehen, den Reiter *Form1.vb [Entwurf]* aktivieren. Man sieht nun ein graues Anwendungs-Fenster. Das ist die grafische Oberfläche unseres Programms.

4. Nun fügt man den ersten Button in die Programmoberfläche ein. Dieser Button soll später alle LEDs leuchten lassen. Klicken sie links auf *Toolbar* (1). In dem erscheinenden Fenster Button (2) auswählen und per Drag&Drop auf die Programmoberfläche (3) ziehen.

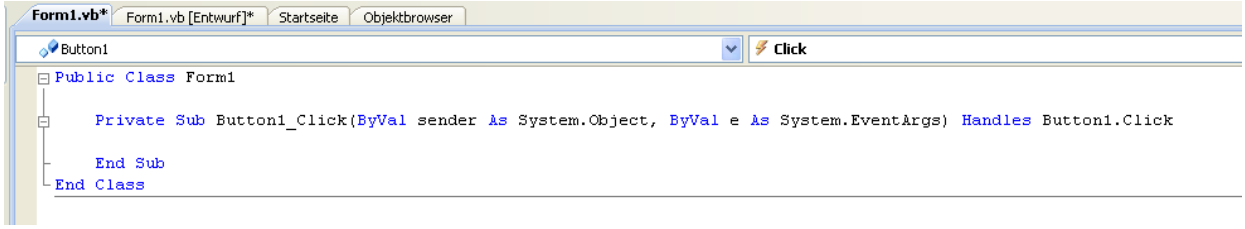


5. Dieser Button kann nun auf die gewünschte Größe zurecht gebracht werden.
Außerdem geben wir ihm im Eigenschaftsfenster rechts unten unter *Text* die Beschriftung *Alle LEDs* (1).
Eventuell muss dazu gescrollt und die Größe des Buttons verändert werden.

Info: Im Eigenschaftsfenster werden Eigenschaften wie zum Beispiel die Beschriftung des Buttons/ Objekts festgelegt.



6. Durch Doppelklick auf den Button gelangt man nun in das Quellcode-Fenster das in seiner Rohform etwa so aussieht:



Durch den Doppelklick wurden die Zeilen Private Sub und End Sub automatisch einfügt. Alles was später dazwischen steht wird ausgeführt, wenn man auf den Button klickt.

7. Spätestens jetzt speichern: Datei → Alle speichern

8. Um die parallele Schnittstelle dem Programm bekannt zu machen, muss man die inpout32.dll im Quellcode einbinden (deklarieren). Deshalb fügt man unter „Public Class Form1“ unten stehende Befehle ein:

```
Public Class Form1
        Public Declare Function Inp Lib "inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer)
        Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal PostAddress As Integer, ByVal Value As Integer)

        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        End Sub
End Class
```

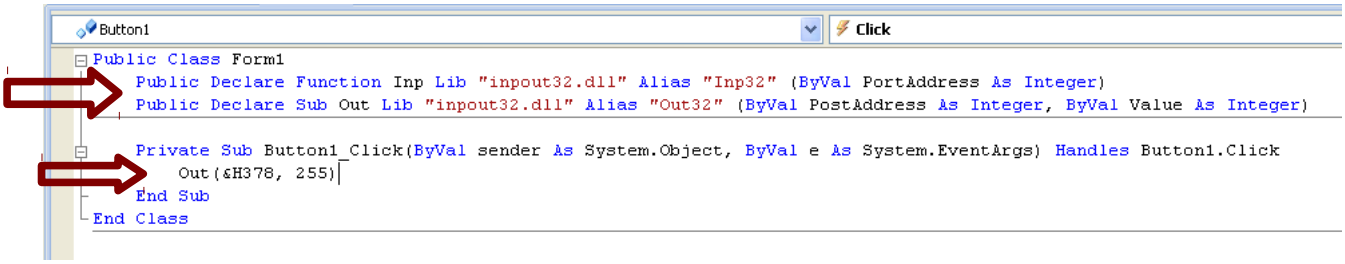
Achtung: Ein kleiner Tippfehler kann dazu führen, dass das Programm nicht funktioniert!

9. Der Befehl *Out (&H378, 255)* veranlasst nun, dass alle LEDs leuchten. Diesen gibt man in Private Sub Button1_Click ein:

```
Public Class Form1
        Public Declare Function Inp Lib "inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer)
        Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal PostAddress As Integer, ByVal Value As Integer)

        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
                Out(&H378, 255)
        End Sub
End Class
```

Info: &H378 ist die Adresse des Parallelport und die Zahl hinter dem Komma bestimmt welche LED(s) angesteuert werden. Sie kann 0-255 betragen.

10. Hat man diese Schritte erledigt, kann man das Programm nun mit diesem Screenshot vergleichen:



11. Das Programm kann mit F5 gestartet werden.

Kleine Erweiterungen

1. Möchte man einen Exit/ Ende-Button, fügt man einfach einen neuen Button ein und gibt diesem den Befehl *End.*
2. Will man später die Sleep Funktion verwenden muss man die Zeile

   Public Declare Sub Sleep Lib “kernel32“ (ByVal dwMilliseconds As Long)
   unter die anderen beiden Deklarationen einfügen. Dann kann man mit dem Befehl *Sleep(t)* arbeiten. Für t die Anzahl der Millisekunden einsetzen. Z.B. *Sleep(1000)*.
3. Jetzt sind der Phantasie fast keine Grenzen mehr gesetzt. Viel Spaß!

3. Ansteuern einzelner LEDs

Man erstellt zunächt ein neues Projekt (Datei → Neues Projekt) und richtet das Projekt wie oben beschrieben ein. Das Ergebnis sollte im Quelltext so aussehen wie in Schritt 10.

```
Public Class Form1
    Public Declare Function Inp Lib "inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer)
    Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal PostAddress As Integer, ByVal Value As Integer)

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Out(&H378, 255)
    End Sub
End Class
```

Um die LEDs einzeln anzusteuern verändert man den letzten Wert im Befehl Out(&H378, 255). Für die 1. LED Out(&H378, 1), für die 2. LED Out(&H378, 2) und für die 3. LED Out(&H378, 4). Diese Steuerungslogik basiert auf dem so genannten Dualzahlensystem. Die folgende Tabelle soll verdeutlichen, welche LEDs mit welchem Wert angesteuert werden können:

LED 8	LED 7	LED 6	LED 5	LED 4	LED 3	LED 2	LED 1
$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Probiere einfach alle Werte nacheinander aus. Das Programm startest du mit F5 und eine Ausgabe erhältst du, wenn du den erstellten Button klickst.

Es lassen sich natürlich auch mehrere LEDs gleichzeitig ansteuern. Sollen die ersten vier LEDs leuchten, gibt man Out(&H378, 15) ein (1+2+4+8).

Alle acht LEDs leuchten bei Out(&H378, 255) (1+2+4+8+16+32+128)

Versuche nun die 3. und die 7. leuchten zu lassen.

4. LEDs nacheinander ansteuern

Aufgabe: „Es sollen die LEDs 1 bis 8 nacheinander für eine Sekunde leuchten und anschließend wieder ausgehen.“ Dazu benötigt man den Befehl *Sleep*. Die Anweisung *Sleep* 1000 lässt den Computer für eine Sekunde (bzw. 1000 Millisekunden) „schlafen“, das heißt die Ausführung des Programms wird hier für eine Sekunde unterbrochen. Die Anweisung Sleep muss man davor allerdings dem Computer erst „beibringen“. Das geht ganz einfach indem man unter *Public Class Form1* eine weitere Zeile einträgt. Das Programm könnte dann so aussehen:

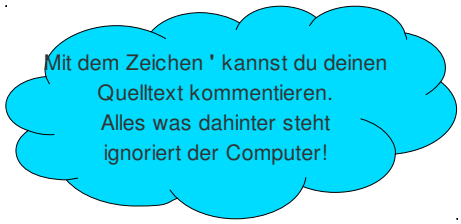
```
Public Class Form1
    Public Declare Function Inp Lib "inpout32.dll" Alias "Inp32" (ByVal PortAddress As Integer)
    Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal PostAddress As Integer, ByVal Value As Integer)
    Public Declare Sub Sleep Lib „kernel32“ (ByVal dwMilliseconds As Long)

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Out(&H378, 1)
        Sleep(1000)
        Out(&H378, 2)
        Sleep(1000)
        Out(&H378, 4)
        Sleep(1000)
        Out(&H378, 8)
        Sleep(1000)
        Out(&H378, 16)
        Sleep(1000)
        Out(&H378, 32)
        Sleep(1000)
        Out(&H378, 64)
        Sleep(1000)
        Out(&H378, 128)
        Sleep(1000)
        Out(&H378, 0)
    End Sub
End Class
```

5. Die For … Next-Schleife

Für dieses Programm musst du zuerst dem Computer die Sleep Funktion „beibringen“ (siehe oben). Außerdem musst ein neues Objekt in dein Programm einfügen, das Text und Zahlen anzeigen kann, ein sogenanntes *Label*. Gehe in den Entwurfsmodus *[Entwurf]* und ziehe aus der Toolbox das Objekt *Label* in dein Programm (siehe Kapitel 2 Schritt 4). Der Name des ersten Labels ist standardmäßig *label1*.

Bevor wir das Programm aus Kapitel 4 mit seinen 16 Anweisungen mit Hilfe der For … Next-Schleife erheblich verkürzen, wollen wir uns die Funktion der For … Next-Schleife an einem einfachen Beispiel etwas genauer anschauen.

```
For zaehler = 1 to 10                'Schleifenkopf
        label1.  Refresh()           'Löscht das Ausgabefeld
        label1.Text = zaehler        'Schreibt den aktuellen Wert
        Sleep(1000)                  'wartet 1 Sekunde
Next                                 'Schleifenende
```



Was passiert? Mit Hilfe der For … Next-Schleife und den dazwischen stehenden Anweisungen kann man die Zahlen 1 bis 10 auf dem Monitor ausgeben. Bei einer For … Next-Schleife (auch Zählschleife genannt) ist von Anfang an klar, wie oft die Anweisungen zwischen dem Schleifenkopf (For) und dem Schleifenende (Next) ausgeführt werden. Im Schleifenkopf *For zaehler = 1 to 10* wird der Variablen zaehler der Startwert 1 zugewiesen. Eine Variable kann man sich wie ein kleines „Körbchen“ vorstellen, in das man etwas hinein legt. Der Endwert der Schleife wird auf 10 gesetzt. Damit gibt man an, wie oft die Schleife durchlaufen werden soll (von 1 bis 10, also 10 mal). Im Schleifenkopf wird jede Runde überprüft, ob die Schleife verlassen werden soll, nämlich dann, wenn der Inhalt der Variablen *zaehler* größer als 10 ist. Da dies im ersten Durchlauf noch nicht der Fall ist, folgen die Anweisungen in der Schleife. Jetzt wird im Ausgabelabel der Inhalt der Variablen *zaehler* ausgegeben. Die Zahl 1. Anschließend folgt die Anweisung *Next*. Hier wird die Variable *zaehler* um 1 erhöht und es folgt ein Sprung zurück zum Schleifenkopf. Dort wird überprüft, ob der Inhalt von *zaehler* schon größer als 10 ist (*zaehler* hat den Inhalt 2). Da dies noch nicht der Fall ist, führen die Anweisungen zu der Ausgabe 2. Bei *Next* wird die Variable *zaehler* wieder um 1 erhöht (ist jetzt also bei 3) und es folgt wieder der Sprung zurück zum Schleifenkopf. Dies wiederholt sich solange bis der Inhalt von *zaehler* größer als 10 ist. Dann wird die Schleife sofort verlassen. Befehle zwischen *For* und *Next* sollen mit der *Tab-Taste* eingerückt werden.

Die Schrittweite, also der Wert um den die Variable zaehler erhöht wird, ist standardmäßig 1. Soll eine andere Schrittweite genommen werden, braucht man den Befehl *Step* im Schleifenkopf.

```
For zaehler = 1 to 10 Step 3         'Schleifenkopf mit Schrittweite 3
        label1.  Refresh()
        label1.Text = zaehler
        Sleep(1000)
Next
```

Eine weitere Möglichkeit einen Zähler zu programmieren ist die Timer-Funktion. Sie zuerst etwas komplizierter liefert dann aber unter Umständen bessere Ergebnisse. Internetrecherche: visual basic 2008 stoppuhr.

6. Mit der For … Next-Schleife LEDs nacheinander ansteuern

Jetzt können wir die For … Next-Schleife nutzen, um die LEDs leuchten zu lassen. Dazu brauchen wir aber nacheinander nicht die Zahlen 1 bis 8 sondern 1, 2, 4, 16 usw.

LED 8	LED 7	LED 6	LED 5	LED 4	LED 3	LED 2	LED 1
$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Wie können diese Zahlen erzeugt werden? Ein Blick auf die Tabelle zeigt die Lösung. Man nimmt als *zaehler* die Hochzahlen von 0 bis 7. Die Schleife könnte dann so aussehen:

```
For zaehler = 0 to 7
        Out(&H378, 2 ^ zaehler)
        Sleep(1000)
Next
```

LEDs	**zaehler**	**2 ^ zaehler**
LED 1	0	$2^0 = 1$
LED 2	1	$2^1 = 2$
LED 3	2	$2^2 = 4$
LED 4	3	$2^3 = 8$
LED 5	4	$2^4 = 16$
LED 6	5	$2^5 = 32$
LED 7	6	$2^6 = 64$
LED 8	7	$2^7 = 128$

Die nachfolgende Tabelle soll die Funktion der Schleife verdeutlichen. Das „Hochzeichen“ befindet sich unterhalb der Esc-Taste. Man drückt erst die „Hochzeichen-Taste“ und dann die Leertaste.

Du siehst Programm 6 und Programm 4 machen genau das gleiche. Programm 6 ist jedoch deutlich kürzer (4 Zeilen anstatt 16)!

7. Die Do … Loop-Schleife

Eine andere Variante ist die Do … Loop-Schleife. Das Prinzip ist sehr ähnlich. Nur, dass die Do … Loop-Schleife das erste mal immer(!) durchläuft. Und wenn keine Abbruchbedingung formuliert wird läuft das Programm endlos weiter. Man nennt sie deshalb auch Endlosschleife.

Das gleiche Programm mit einer Do … Loop-Schleife könnte so aussehen:

```
Dim zaehler As Integer                  'Die Variable zaehler wird dem Computer bekannt gegeben
zaehler = 0                             'Die Variable zaehler bekommt den Wert 0
Out(&H378, 0)                           'Alle LEDs werden gelöscht
Do                                      'Der Schleifenkopf
        Out(&H378, 2 ^ zaehler)         'Die LEDs werden je nach Zählerstand angesteuert
        Sleep(1000)                     'Pause
        zaehler = zaehler + 1           'Die Variable zaehler wird um 1 hochgesetzt
Loop Until zaehler = 8                  'Wenn die Variable zaehler den Wert 8 hat, wird die Schleife beendet
```

8. Zufallszahlen erzeugen mit Rnd()

Das folgende Programm soll eine zufällige Zahl zwischen 1 und 100 erzeugen – wie ein 100-seitiger Würfel.

Ziehe aus der Toolbox ein *Label* in dein Programm (siehe Kapitel 2 Schritt 4). Der Name ist für das erste Label standardmäßig *Label1*. Außerdem brauchst du wie üblich einen Button, nach dessen Klick die Zufallszahl im *Label* erscheint.

Das Programm dazu könnte so aussehen:

```
Randomize()                                  'Der Zufallsgenerator wird aktiviert
Dim zufallszahl As Integer = Int(Rnd() * 100 + 1)   'Die Zufallszahl wird berechnet (siehe unten)
label1.Text = zufallszahl                    'Die Zahl wird im Label angezeigt
```

Was passiert hier?

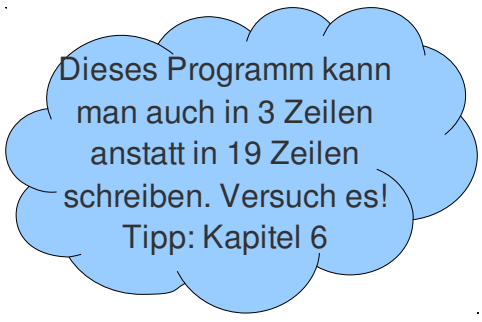
Rnd() erzeugt eine Zahl zwischen 0 und 1 (z.B. 0.001). Mit dem *Int-Befehl* kann man die Kommazahl in eine ganze Zahl umwandeln. Die Zahlen hinter dem Komma werden dabei abgeschnitten (nicht gerundet): Int(0.001) = 0 bzw. Int(0.999) = 0
Deshalb muss man die Zufallszahl mal 100 nehmen. Es ergibt sich dadurch ein Bereich von 0 bis 99. Wenn die Werte aber bei 1 starten sollen und bis 100 gehen sollen muss man zu dem Ergebnis einfach 1 addieren: Int(0.001*100) + 1 = 0 + 1 = 1
bzw. Int(0.999*100) + 1 = 99 + 1 = 100

9. If … Then … ElseIf (Wenn … Dann … Sonst)

Manchmal sollen Programme nicht nur linear ablaufen (Anweisung für Anweisung), sondern verzweigen, wenn Entscheidungen getroffen werden. Dazu kann man die *If … Then … ElseIf-Verzweigung* verwenden.

In diesem Programm soll auf Knopfdruck zufällig eine LED leuchten.
Das Programm könnte so aussehen:

```
Randomize()                                  'Der Zufallsgenerator wird gestartet
Dim wert As Integer = Int(8 * Rnd())         'Zufallszahl zwischen 0 und 7 erzeugen und in der Variablen ...
                                             ...wert gespeichert
If wert = 0 Then                             'Wenn wert gleich 0 ...
        Out(&H378, 2 ^ 0)                    '...dann leuchtet die 1. LED
ElseIf wert = 1 Then                         'wenn aber wert gleich 1
        Out(&H378, 2 ^ 1)                    '… dann leuchtet die 2. LED
ElseIf wert = 2 Then                         'usw.
        Out(&H378, 2 ^ 2)
ElseIf wert = 3 Then
        Out(&H378, 2 ^ 3)
ElseIf wert = 4 Then
        Out(&H378, 2 ^ 4)
ElseIf wert = 5 Then
        Out(&H378, 2 ^ 5)
ElseIf wert = 6 Then
        Out(&H378, 2 ^ 6)
ElseIf wert = 7 Then
        Out(&H378, 2 ^ 7)
End If                                       'Ende der Verzweigung
```



Die If … Then-Verzweigung funktioniert alleine, kann aber um die Anweisung ElseIf erweiter werden. Benötigt eine einzelne If … Then … ElseIf-Verzweigung mehrere Zeilen, dann muss sie mit End If abgeschlossen werden.

10. Mit der Tastatur steuern

Achtung: Dieses Kapitel ist experimentell und kein gutes Design für eine Steuerung, da die gedrückten Tasten als Text in einer Box erscheinen.

Ziehe aus der Toolbox eine *TextBox* in dein Programm (siehe Kapitel 2 Schritt 4). Der Name ist standardmäßig *TextBox1*. Durch Eingabe von Buchstaben in diese Textbox sollen LEDs gezielt angesteuert werden.

Ein Programm dafür könnte so aussehen:

```
Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown
        If e.KeyCode = Keys.W Then              'Wenn W gedrückt wird ...
                Out(&H378, 1)                   '… leuchtet die 1. LED
        ElseIf e.KeyCode = Keys.A Then          'Wenn aber A  gedrückt wird ...
                Out(&H378, 2)                   '… leuchtet die 2. LED
        ElseIf e.KeyCode = Keys.S Then          'Wenn aber S gedrückt wird ...
                Out(&H378, 4)                   '...leuchtet die 3. LED
        ElseIf e.KeyCode = Keys.D Then          'Wenn aber D gedrückt wird ...
                Out(&H378, 8)                   '… leuchtet die 4. LED
        End If
End Sub
```

Im Programm klickt man erst auf die *TextBox* und kann dann mit den Tasten steuern.

Man könnte auf diese Weise eine Fahrzeugsteuerung realisieren.

11. Aufgaben für die Programmierung des GSR-Interface

a) Die LEDs 1 bis 8 sollen nacheinander 1 Sekunde leuchten.

b) Der Benutzer soll die Leuchtdauer der LEDs in Aufgabe a) über den Bildschirm selbst bestimmen können.

c) Es sollen nacheinander die LEDs 1 bis 8 leuchten.

d) Die LEDs 1 bis 8 sollen nacheinander für 1 Sekunde leuchten. Nachdem LED 8 geleuchtet hat, sollen wieder die LEDs 7 bis 1 für eine Sekunde leuchten.

e) Die LEDs in Aufgabe d) sollen solange leuchten bis die Taste E gedrückt wird.

f) Es sollen immer zwei LEDs gleichzeitig leuchten. LED 1+8, LED 7+2, LED 6+3 usw...

g) Das Programm aus Aufgabe f) soll jederzeit mit der Taste E unterbrochen werden können.

h) Über den Bildschirm soll der Benutzer eine Zahl von 1 bis 6 raten, die mit einem zufälligen Würfelergebnis verglichen wird.

i) Nach Eingabe/Auswahl einer Note als Zahl (z.B. 3) soll die Verbalnote (hier: befriedigend) erscheinen.